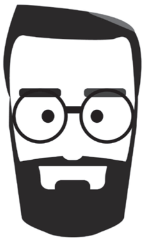
ניתוח אלגוריתמים וסיבוכיות

מסוכם משיעורי מרן הרב
פרופ' יעקב הכהן-קרנר שליט"א
ובתוספות תרגולי הרב הגאון
אבירם זילברמן שליט"א

עם פירוש
"רי"ח טוב"
מאיתי הצב"י יוחנן חאיק

שנת 01011001111111



"כשם שאי אפשר לבר בלא תבן, כך אי אפשר לתוכנית בלי שגיאות של הבודק האוטומטי"

להערות, הארות ותיקונים:
yohananha@gmail.com
yohanan@ - בטלגרם

ניתן להשתמש בסיכום באופן חופשי לכולם!!

תוכן עניינים

תוכן עניינים **1**

ניתוח אלגוריתמים וסיבוכיות **5**

אלגוריתמים חמדניים Greedy Algorithm **5**

בעיית תרמיל הגב בשברים 6

בעיית בחירת הפעילויות 7

בעיית תזמון התדלוק 10

תכנון דינאמי **12**

כפל סדרת מטריצות 13

אפיון המבנה של הפתרון האופטימלי 14

הפתרון הרקורסיבי 15

חישוב הערך האופטימלי "מלמטה למעלה" 15

בניית פיתרון אופטימלי 18

בעיית אופטימיזציית מצולעים – טריאנגולציה 19

הקבלה להצבת סוגריים מלאה – אפיון המבנה 20

פתרון רקורסיבי 23

חישוב הערך האופטימלי מלמטה למעלה 24

בעיית תרמיל הגב בשלמים – הגרסה הדינאמית 27

תת סדרה משותפת ארובה ביותר Longest Common Subsequence 30

הפתרון הנאיבי 31

הפתרון הרקורסיבי 32

חישוב ערכו של פתרון אופטימלי "מלמטה למעלה" 33

בנית פתרון אופטימלי מתוך המידע שהושג 34

פסאודו אלגוריתם 35

שיטת התזכור (הפתקאות) Memoization 36

גרפים **38**

עץ פורש מינימלי בגרף **38**

האלגוריתם הכללי למציאת עפ"מ 39

האלגוריתם של קרוסקל (Kruskal) 41

האלגוריתם של פרים 45

שאלות הרחבה 49

מסלולים קצרים ביותר **53**

גרסאות שונות עבור הבעיה 53

הגדרות בלליות עבור הפרק 53

מבנה הפתרונות המוצעים 54
למות להוכחה -חלק א' 55
טכניקת ההקלה Relaxation 56
למות להוכחה – חלק ב' 57
עצי מסלולים קצרים ביותר 58
האלגוריתם של דייקסטרה 59
הסבר האלגוריתם 59
הוכחת נכונות האלגוריתם 63
ניתוח זמן ריצה 64
בעיית המשקלים השליליים 64
הבדלים בין דייקסטרה לפרים 65
האלגוריתם של בלמן-פורד 66
הסבר האלגוריתם 66
ניתוח זמן ריצה 68
הוכחת נכונות האלגוריתם 69
שאלות הרחבה 69
אלגוריתם פלויד וורשאל 73
הסבר האלגוריתם 73
ניתוח זמן ריצה 76
תפוסת מקום בזיכרון 76
רשתות זרימה 77
מושגים כלליים לרשתות זרימה 77
מקורות ובורות מרובים 79
עבודה עם זרימות 80
שיטת פורד פולקרסון 81
זרימה מקסימלית חתך מינימלי 83
אלגוריתם פורד פולקרסון הבסיסי 84
אלגוריתם אדמונדס-קארפ 85
ניתוח זמן ריצה 87
בעיית הזיווג המקסימלי בגרף דו צדדי 87
שאלות הרחבה 89
מחלקות סיבוכיות 92
קידוד 93
מסגרת של שפות פורמליות 94

אימות פולינומיאלי.... 95
מעגלים המילטוניים.... 96
מחלקת הסיבוכיות NP 97
שאלות הרחבה 99
שלמות ב-NP ורדוקציות.... 102
רדוקציות 102
שלמות ב-NP.... 104
ספיקות מעגלים.... 105
הוכחות שלמות ב-NP.... 107
ספיקות נוסחאות 108
ספיקת נוסחאות 3-CNF 110
בעיית הקליקה Clique.... 114
בעיית ביסוי הקדקודים Vertex-Cover 117
בעיית סכום התת-קבוצה Subset-Sum.... 118
בעיית המעגל ההמילטוני Hamilton's Cycle.... 119
בעיית הסוכן הנוסע The Traveling Salesman.... 119
מחלקת coNPC – המשלימה ל-NPC.... 120
בעיית הטאוטולגיה TAUT.... 120
שאלות הרחבה 121
בעיית החלוקה (Partition).... 121
בעיית תרמיל הגב בשלמים 0-1 (Knapsack).... 123
בעיית אריזות המיכלים (Bin Packing) 126
אלגוריתמי קירוב.... 128
חסמים על ביצועי אלגוריתמי קירוב 128
אלגוריתם קירוב Vertex-Cover 129
בעיית ביסוי הקבוצה 131
בעיית הסוכן הנוסע 135
אי שיוויון המשולש 135
בעיית הסוכן הנוסע הכללית 138
בעיית אריזות המיכלים (Bin Packing) 139
נספח א' סיכום אלגוריתמים על גרפים 142
אלגוריתמים כלליים.... 142

אלגוריתמים לעצים פורשים מינימליים 143
אלגוריתמים למציאת מסלולים קצרים ביותר 143
אלגוריתמים לרשת זרימה 144
נספח ב' 145
סדר הוכחה לשייכות לNPC 145
הוכחת שייכות ל-NP 145
הוכחת שייכות ל-NPH 145

ניתוח אלגוריתמים וסיבוכיות

נושא הקורס הוא ניתוח אלגוריתמים. אנו נראה במהלך הקורס קבוצות שונות של אלגוריתמים מסוגים שונים, כאשר מה שיעניין אותנו יהיו האלגוריתמים עצמם ולא מבני הנתונים בהם אנחנו משתמשים. הרעיון של שימוש באלגוריתמים הוא למצוא פיתרון אופטימאלי לבעיה נתונה בעזרת רצף פעולות מסוים. הבעיות שיוצגו לנו הם בעיקר מהתחום הבדיד, להם אנחנו נצטרך למצוא יעד – מקסימום או מינימום. ניתקל גם בהמשך הקורס בקבוצות שונות של אלגוריתמים, כאשר כל סוג שונה של קבוצה בא לפתור טווח מסוים של בעיות בדרך אחרת.

אלגוריתמים חמדניים Greedy Algorithm

הגדרה: "אלגוריתם שמקבל החלטות לפי נתונים קיימים, ללא תכנון קדימה".

אלגוריתם זה לא בוחן את ההשפעות על "מה יקרה אם", אלא מוצא איזה דרך שנראית מתאימה ומובילה לפיתרון ועובד עליה עד הסוף. במקרים המתאימים, אכן ניתן להוכיח שמדובר על אלגוריתם שהוא אופטימלי, כלומר הוא מביא את התוצאה הטובה ביותר. כמובן, שאלגוריתם זה לא פותר **כל** בעיה – אם נרצה לטפס לראש ההר ונראה דרך שהיא ישרה, אולי זה יהיה יותר נוח, אבל כנראה שלא נגיע כך לפסגה.

נראה מספר דוגמאות שמקיימות את האלגוריתם הזה -

עלינו לסדר תיק. לתיק יש תבולה של משקל מסוים W אותו אנחנו נדרשים למלא, ועלינו להחליט מבין כל החפצים המונחים לפנינו, אילו חפצים ייכנסו, על מנת לקחת איתנו את המקסימום האפשרי. הצעה פשוטה לפתרון הבעיה היא, להתחיל בלהכניס את החפצים הקטנים יותר לתוך התיק, וכך גם אם לא נמלא את התיק עד סופו, נדע לפחות שהכנסנו כמה שיותר פריטים.

הפתרון הזה נראה יחסית פשוט, אבל יש לנו מספר שאלות עליו:

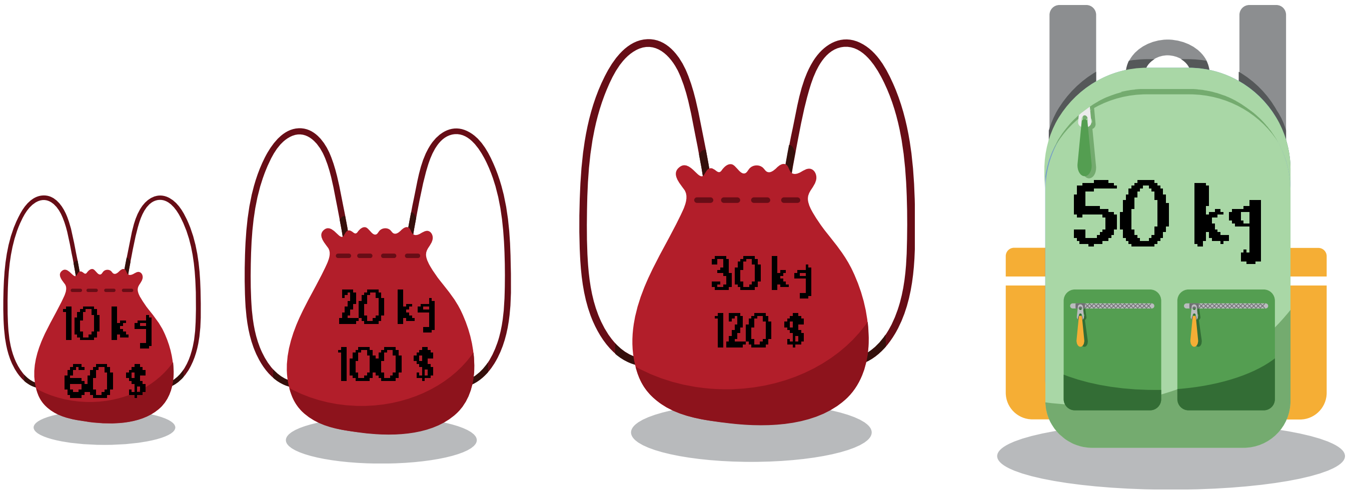
- מי אמר שהפתרון הזה הוא באמת הטוב ביותר?
- מי יכול להבטיח לנו שהפתרון הזה יהיה יעיל גם בפעם הבאה שננסה אותו?

ההצעה שאנחנו הבאנו (הכנסה של החפץ הקל יותר ראשון) שייכת לסוג פתרונות שנקרא "פתרון חמדני"- פתרנו את הבעיה באופן המהיר ביותר, אך ללא שום התחשבות באפשרות שיש פתרון אחר יעיל יותר. בהמשך נראה שעבור לא מעט בעיות אותן ניתן לפתור באופן חמדני שכזה, אם נשנה את התנאים ולו במעט, לא נוכל למצוא את הפתרון הזה והוא לא יעבוד. באופן כללי ניתן לומר שהתהליך אותו אנחנו עוברים מורכב משני שלבים: 1. ביצוע הפתרון החמדני. 2. קבלה של תת-מבנה אופטימלי – באופן שדומה קצת לאינדוקציה, רק הפוך. אנחנו ביצענו את הפיתרון החמדני, וקיבלנו עוד חלק קטן יותר שגם עליו אנחנו יכולים לבצע פתרון חמדני דומה.

עכשיו ניקח את הבעיה שהצגנו קודם, שלב אחד קדימה. בעת במקום שיהיה לנו רק שדה של משקל עבור כל פריט, יהיה לנו גם חשיבות. האם יש לנו דרך למצוא איזה פתרון אופטימלי על פי משקל / חשיבות / שילוב של שניהם?

אם נרצה למשל, לקחת את סכום החשיבויות הגבוה ביותר, יכול להיות שאם נלך בשיטה שרצינו לפי המשקל, הפתרון לא יקיים בכלל את הדרוש, מאחר והוא לא יעמוד בתנאי הכללי, כי אין לנו הבטחה לאיזה יחס משקל-חשיבות.

ניתן להשתמש עבור זה גם בפתרון נאיבי – חישוב של "חשיבות סגולית" של כל פריט בשקלול של משקל ומספר חשיבות, והציון שייצא לנו יוכל בעת להוות איזה מדד להכנסה נכונה. בשביל להבין כיצד זה יעבוד בצורה נכונה, נציג את הבעיה בצורתה המוכרת והרשמית יותר –



בעיית תרמיל הגב בשברים

גנב נכנס לחנות עם שק שמוגבל במשקל מסוים. לפניו בחנות מונחים פריטים אותם הוא מעוניין לגנוב. עבור כל פריט יש משקל ומחיר, ומתוך כל פריט ניתן לקחת גם חלקים ממנו ולא חייבים את כולו בשלמותו.

<u>המטרה:</u> למלא את השק כך שערך הפריטים שהגנב לוקח יהיה מקסימלי[1].

קל לראות, שניתן לחשב באופן די פשוט, ולהגיע למסקנה שנכניס קודם את השק של 10 ק"ג ולאחר מכן את ה20, ולבסוף עוד 20 ק"ג מהשק הנותר. הפתרון הזה נעשה בצורה חמדנית על פי התנאים שהגדרנו לעיל – ראשית, הבאנו פתרון חמדני – הכנסנו את מה ששווה הכי הרבה, ואחר כך נותרנו עם תת-מבנה אופטימלי דומה – צריך להכניס 40 ק"ג לתוך התיק.

ושוב נשאלת השאלה – כיצד ניתן להוכיח שזה באמת פיתרון אופטימלי, ואין שום פיתרון שיהיה יותר טוב? (בהנחה שיכול להיות שנמצא פתרונות שהם שווים באיכותם, אך לא כאלה טובים יותר)

דבר נוסף שכדאי לשים לב – בשונה מבעיות אחרות שנתקלנו בהם, גם אם היינו רוצים אין לנו אפשרות להציג את כל הקומבינציות האפשריות, וזאת מאחר שאנחנו תמידי יכולים להתקזז על המשקל מכל שקית עד שזה יביא לנו מרחב פתרונות אינסופי (לקחת 0.1, 0.01, 0.001, 0.0001 וכן על זה הדרך).

נוכיח את נכונות האלגוריתם בצורה פורמלית:

נתונים n חומרים כך שלחומר ה-i, P_i יש משקל W_i ומחיר V_i ולכן מחיר סגולי C_i (ממויינים לפי המחיר הסגולי, כלומר החומר שעבור i=1 הוא בעל המחיר הסגולי הגבוה ביותר).
הפתרון החמדני נותן לנו רווח כולל V.

1. <u>תכונת הבחירה החמדנית</u>

---

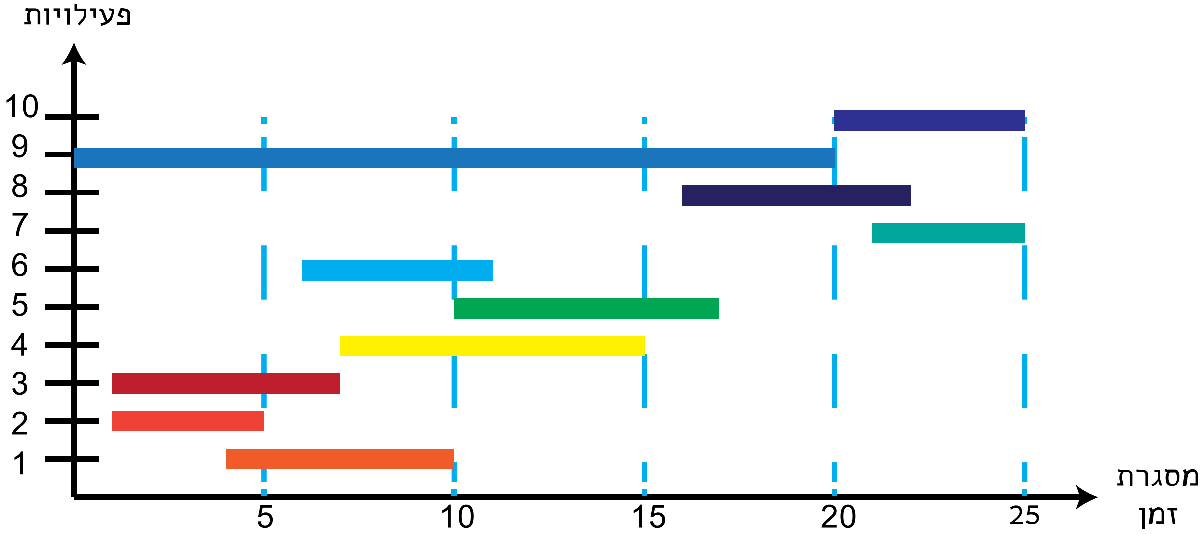
[1] במובן שמאחר וניתן לקחת גם חלקים מהסחורה, השק בסופו של דבר צריך להיות מלא.

נניח (בשלילה) שקיים חומר ששימוש דווקא בו יביא לנו פיתרון אופטימלי V_1-
- אם החומר P_1 קיים גם בפיתרון הזה, אז $V_1 = V$ וזו סתירה להנחת השלילה.
- אחרת: ניקח מהחומר P_1 כמות כלשהי שאינה גדולה מ- W_1 ונחליף אותה עם כמות חומר זהה מהחומר P_x. בהכרח קיבלנו פתרון עם רווח גדול יותר – סתירה. (במילים פשוטות יותר: הוצאת החומר בעל הערך הגבוה ביותר בהכרח מביא לנו אופציה שהיא פחות טובה – בחירה בחומר שהמחיר הסגולי שלו נמוך יותר)

2. <u>תת המבנה האופטימלי:</u>

לאחר שכבר בחרנו את החומר P_1, נשארנו עם תת-בעיה, עלינו להוכיח שגם אותה נוכל לפתור באותו אופן ולקבל פיתרון אופטימאלי.
נניח שהפתרון לתת הבעיה ייתן רווח V
נניח (בשלילה) שקיים פיתרון אחר לתת הבעיה, שייתן רווח V'>V
לכן נצרף לפתרון הזה את P_1 ונקבל סה"כ פתרון טוב יותר עבור הבעיה המקורית.
(ובעברית: אם נמצא פתרון לתת הבעיה שהוא אופטימלי ושונה ממה שכבר הצענו, אז מעצם ההגדרה נוכל להחיל את הפתרון גם על הבעיה בכללות, והפתרון שהצענו הוא לא אופטימלי, אבל כבר הוכחנו שהוא כן אופטימלי ולכן יש פה סתירה).

בעיית בחירת הפעילויות



בעיה מוכרת וידועה – צריך לשבץ קורסים בתוך כיתה. עבור כל קורס יש את טווח הזמנים בו הוא צריך להיות פעיל. אנחנו רוצים למצוא את הדרך האופטימלית לשיבוץ הקורסים בכיתות. מן הסתם, אסור שתהיינה התנגשות בין שני שיעורים, ואנחנו מחפשים את הפתרון שייתן את התוצאה הטובה ביותר.

קודם כל, עלינו לוודא מה דרוש מאתנו על מנת שייחשב סידור אופטימלי - האם אנחנו מחפשים ניצולת מקסימלית של הכיתה, חוסר בטלה מינימלי (דבר שפה פחות משמעותי אך בתחומים אחרים משפיע יותר), או כל הגדרה אחרת שתבוא בשאלה. במקרה זה אנחנו מדברים על מספר פעילויות מקסימלי, כאשר אין הבדל מבחינת חשיבות בין שיעור של יחידת זמן אחת לבין 10 יחידות זמן.

הגדרה רשמית של השאלה תיראה באופן הבא:

> נתונה קבוצה של n פעילויות המבקשות להשתמש באותו משאב אשר יכול לשרת רק פעילות אחת בו-זמנית. לכל פעילות יש זמן התחלה (s_i) וזמן סיום (f_i) כאשר מתקיים $f_i>s_i$. כאשר $0 \le s_i \le f_i \le$ (T) ונתון זמן מקסימלי לביצוע כל הפעולות (T), וכל הפעילויות מתבצעות בזמן החצי-פתוח [0,t).

בפשטות – כל עוד שתי פעילויות לא מתנגשות אחת עם השנייה, הכל בסדר. כמובן שפעילות יכולה להסתיים באותה שעה בדיוק בה מתחילה פעילות חדשה.

בנוסף נגדיר את S = {1,2….,n} בקבוצת הפעילויות הקיימת, וכל תת קבוצה אפשרית תסומן באות גדולה מתחילת הABC.

השאיפה שלנו היא למצוא אלגוריתם / דרך פעולה שיביא לנו את מקסימום הפעילויות תחת תת-הזמן האפשרי. נבחן מספר אפשרויות –

<u>משך שיעור מינימלי</u> – A = {7,6}

בדאי לשים לב, פה קודם כל הגדרנו את 7 שהוא הכי קצר, ורק לאחר מכן את 6 שהוא הקצר ביותר מבין אלו שנשארו לא בחפיפה.

<u>משך שיעור מקסימלי</u> – A = {9,10}

מתחילים עם שיעור באורך 20, ואז כבר לא ממש נשאר אופציות.

<u>זמן תחילת שיעור</u> – A = {9,10}

השיעור הראשון הוא 0-20 מה שתופס לנו במעט את כל מרווח הזמן.

<u>זמן סיום השיעור</u> - A = {2,6,8}, B = {3, 4, 8} C = {1, 5, 7}

ברור לנו שזהו אלגוריתם הרבה יותר אופטימלי מכל מה שהצענו עד עכשיו. בדרך כלל אנחנו עלולים לחשוב שהפתרון החמדני הוא דווקא האינטואיטיבי, אך כאן זה בהחלט לא אינטואיטיבי.

אנחנו יכולים גם להוכיח שהאלגוריתם הזה עובד – איך ניתן להוכיח זאת? ניתן לראות ששום פיתרון אחר מבין כל האלגוריתמים שהצענו לא הביא לתוצאה מוצלחת יותר, אך כמובן ש"לא ראיתי אינה ראיה", ואנחנו מחפשים דרך פורמלית יותר.

דרך הפתרון המלא לתכנון חמדני כולל שלושה שלבים:

1. מגדירים אלגוריתם שיקול חמדני ובודקים שהוא מוביל לפתרון אופטימלי עבור מספר דוגמאות שונות. אין צורך לבדוק את כל הקומבינציות האפשריות, אלא רק לראות שהרעיון נשמר.
2. (בסיס האינדוקציה החמדנית) מוכיחים שקיים פתרון אופטימלי (לפחות אחד) שמקיים את הפתרון החמדני (לפחות בפעם הראשונה).
3. (שלב מעבר האינדוקציה) מוודאים שלבעיה יש תת מבנה אופטימלי – מהפתרון שמקיים את (2) נגזר פתרון לתת הבעיה – שהוא אופטימלי לאותה תת-בעיה. אנחנו לוקחים את הקבוצה שהצענו כפיתרון אופטימלי ומורידים את האיבר הראשון מתוכו (כולל כל הפעילויות המתנגשות איתו) ובודקים כעת עבור קבוצת הפעילויות הנותרת את הפתרון האופטימלי מתוך שארית הקבוצה A.

הוכחה:

ללא הגבלת הכלליות נניח שהפעילויות ממוינות לפי זמני סיום מוקדמים תחילה.

1. <u>טענה</u>: קיים פיתרון $A \subseteq S$ אופטימלי המקיים את השיקול החמדני. כלומר $1 \in A$
<u>הוכחה</u>: S סופית. לכן מספר הקבוצות $A \subseteq S$ סופי ולכן לבעיה קיים פתרון אופטימלי.
תהי A פתרון אופטימלי .
אם $1 \in A$ אז סיימנו.
אחרת, קיימת פעילות $i \neq 1$ כך ש $i \in A$ ובעלת זמן סיום הכי מוקדם שם.
אם $s_i \geq f_1$ אז פעילות 1 לא מתנגשת עם i ולא עם שאר הפעילויות של A. לכן הקבוצה A כולל {1} מכילה פעילויות המתיישבות זו עם זו, ו- $|A \cup \{1\}| > |A|$ בסתירה לאופטימליות של A.
לכן בהכרח הם משולבים, ו-A אופטימלית.
2. <u>טענה</u>: נגדיר s', המכיל i שאינו 1 ומתיישב איתו (לא מתנגשים) – הגדרת תת בעיה מאותה סוג
תהי $A' = A-1$ כאשר A היא מהטענה הנ"ל. אז A' אוטימאלית עבור S'.
<u>הוכחה</u> :(בשלילה) אם A' איננה אופטימלית עבור S' אז קיימת $B' \subseteq S'$ כך ש $|B'| > |A'|$. נגדיר $B = B' \cup \{1\}$. אזי B מכילה פעילויות המתיישבות זו עם זו שכן $B' \subseteq S'$ ו- S' מכילה רק פעילויות המתיישבות עם 1.
בעת $B \subseteq S$ ו- $|b|>|a|$ - סתירה.

נסתכל בעת על הקוד. נתון לנו מערך D באורך n. המכיל 3 שדות
Sj = D.1[j], Fj = D.2 [j], j = D.3[j], b[j] = 1

```
Greedy-Selector(D,n)
  Sort D with respect to D.2
  with non-increasing order
  Initialize array B of zero bits with length n
  B[1]<-1
  Last<-1
  For j=2 to n do
    if D.1[j]>=D.2[last] then
       b[j]<-1
       last<-j
  Initialize empty set A
  for j =1 to n do
       if b[j] ==1 then
       A <- Au{D.3[j]}
  return A
```

קבוצת שורות	זמן ריצה	הערות
1	Nlogn	מיון מיזוג או ערימה
2	N	
3	1	
4	1	
5	N	

6	1	
7	N	
8	1	
סה"כ	$N\log_2 n$	

בעיית תזמון התדלוק

מספר רכבים עומדים בתחנת דלק, כאשר ניתן להכניס בכל פעם רק רכב אחד לתדלוק. כמובן שלכל רכב יש זמן תדלוק אחר.

<u>הבעיה:</u> כיצד ניתן למצוא את זמן התדלוק האופטימלי, כלומר לשחרר את כל הרכבים תחת הזמן הקצר ביותר?

אנחנו שואפים שזמן ההמתנה הכולל יהיה מינימלי, ולכן גם פה אנחנו נלך על הSJF (Shortest Job First) שאנחנו מכירים וניתן לרכב שמתדלק הכי מהר להיות ראשון. האם זה באמת אופטימלי? כן. גם במקרה זה הפיתרון האינטואיטיבי הביא לו את התוצאה הטובה יותר, וגם נוכיח זאת פורמלית במבנה שלמדנו:

1. <u>תכונת הבחירה החמדנית:</u>
   יש להוכיח שאמנם קיים פתרון אופטימלי שבו הבחירה הראשונה היא החמדנית.
   נניח בשלילה שקיים תזמון אופטימלי של זמני התדלוק, שבו המכונית הראשונה אינה זו עם זמן התדלוק המינימלי.
   אם נחליף את המכונית הראשונה עם המכונית שיש לה זמן תדלוק מינימלי, נקבל סך הכל זמן המתנה שאינו ארוך יותר.
   סתירה!
2. <u>תכונת תת המבנה האופטימלי:</u>
   יש להוכיח שבהינתן פתרון אופטימלי לבעיה, אם נוותר על המכונית הראשונה (ונשאל את אותה בעיה בדיוק באופן חדש) נקבל פתרון אופטימלי לבעיה בלי המכונית הראשונה.
   - נניח בשלילה שבהינתן תזמון אופטימלי למכוניות 1...n בפתרון של תזמון המכוניות 2...n אינו אופטימלי.
   - אזי קיים פיתרון אחר שמשבץ את המכוניות 2...n באופן טוב יותר.

- נבחר את התזמון הבא:
  - נבחר את התזמון של מכונית 1, ואחר כך את התזמון של שאר המכונית על פי האלגוריתם החדש (הטוב יותר, לכאורה) נקבל תזמון טוב יותר לקבוצת המכוניות 1...n
  - סתירה!

לו היה בנמצא אופציה בלשהי לסדר את כל המכוניות מלבד הראשונה בצורה אופטימלית, אזי ההוספה של אותו רכב (כמובן בהקשר של ההוספה שלו לכלל המערך של המכוניות האמורות לתדלק) אמור להיות עדיין אופטימלי, אבל יש פה סתירה.

תכנון דינאמי

התכנון הדינאמי הוא הנדבך הנוסף שאנחנו בונים על גבי התכנון החמדני. הבעיות שנתקלנו בהן עד עכשיו, דרשו מאיתנו למצוא איזה אלגוריתם שייתן לנו פתרון אופטימלי, ויעבוד בהמשך לאורך כל הדרך (תזכורת: היו לנו שתי דרישות: 1. הוכחת התכנון החמדני 2. הוכחת תת הבעיה האופטימלית). הבעיות שניתקל בהם בעת, הינם מסוג שתכנון שכזה לא יספק אותנו מאחר ולא יעמוד בשתי הדרישות הנ"ל.

הפתרון שנציע כאן, כשמו, הוא לפתור את הבעיות בצורה דינאמית – הווה אומר – לפתור את הבעיה הנוכחית בצורה אופטימלית, ובצעד הבא לחשב מחדש מה באמת יהיה הדרך הנכונה ביותר ברגע זה.

על מנת להדגיש – בתכנון החמדני, אנחנו מוצאים פתרון ורצים **רק** עליו עד לסוף הבעיה. בתכנון הדינאמי, אנחנו עוצרים **בכל** צעד ובודקים מה תהיה הפניה הנכונה להמשך.

בעיות שנדרשות לפיתרון מצורה דינאמית, הן למעשה בעיות רקורסיביות (גם החמדניות הן כאלה, אך יש ביניהם הבדלים). כל פעם אנחנו מתמודדים עם ברירה בין אפשרות אחת (או יותר) שתקיים לנו את הדרישות. הבעיה בלגשת לפיתרון בצורה רקורסיבית, היא זמן החישוב והמשאבים שזה ידרוש מאיתנו.

ניקח דוגמא פשוטה להבנת הבעיה – סדרת פיבונצ'י. אמרנו רקורסיה, אמרנו פיבונצ'י.

למקרה שמישהו הגיע לשלב הזה ועדיין לא יודע מה זו סדרת פיבונצ'י – רצף מספרים המתחילים מ-0 וכל מספר מורכב מחיבור שני האיברים שלפניו.

עלינו למצוא את המספר השמיני בסדרה. כיצד נגיע אליו? נחבר את המספרים שש ושבע. איך נגיע אליהם? וכו' וכו'. מה שזה ייצור לנו זה שעבור הגעה לכל רכיב של מספר נרד עד לשורש הכי קטן ונרכיב את כל העץ. למעשה, נעשה את כל החישובים מספר פעמים - כל מספר שנרצה לקבל, נרכיב את מרכיביו בלי קשר לזה שכבר עשינו את זה בעבר.

לעומת זאת, אם ניקח את מגדלי האנוי[2], אין לנו פתרונות חופפים – פתרון הבעיה מורכב מעשיית החלק הקטן יותר של הבעיה בצורה אחת ויחידה.

מסקנה: לא כל דבר רקורסיבי ייפתר על ידי פיתרון דינאמי.

יש לשים לב – דבר שמייחד לנו את הפיתרון של התכנון הדינאמי מהחמדני, זה הדרישה לתת-מבנה אופטימלי שלא מתיישבת בתכנון הדינאמי, ולכן עלינו לבדוק מחדש עבור תת-המבנה בכל פעם מחדש את הפתרון המתאים.

ארבעת השלבים לפתרון דינאמי הם:

1. <u>אפיון המבנה של פתרון אופטימלי</u> – שלב שהוא די דומה לפתרון החמדני.
2. <u>הגדרה רקורסיבית של ערכו של פתרון אופטימלי</u> – נוסחת נסיגה – הפתרון בדרך כלל יוצג לנו באלגוריתם שמורכב מכל האופציות השונות אותם אנחנו יכולים לממש. הדרך הנוחה ביותר להראות את האופציות השונות תהיה נוסחת נסיגה.

---

[2] https://he.wikipedia.org/wiki/מגדלי_האנוי (לא שבאמת צריך)

3. <u>חישוב ערכו של פתרון אופטימלי "מלמטה למעלה"</u> – לצאת מסדר מעריכי (בדיקה של כל האופציות) לפולינומי. הבעיה העיקרית של נוסחת הנסיגה הרקורסיבית, היא מה שציינו כבר קודם – אנחנו צריכים לעשות המו חישובים שמתבססים אחד על השני. השלב הזה נותן לנו לעשות כל חישוב רק פעם אחת, לרשום אותו בצד, ועל ידי זה לחשב את הצעדים הבאים על ידי גישה לתוצאה ((O(1 ולא חישוב מחדש שבוודאי ייקח יותר.
4. <u>בנית פתרון אופטימלי מתוך המידע שהושג</u> – לאחר שנקבל את התוצאה שתביא לנו את הערך האופטימלי אליו אנחנו שואפים, נחשב את הדרך שתביא לנו את הפתרון (הווקטור שמרכיב את התוצאה האופטימלית)

הערה: ישנם מקרים, בהם ניתן לעבור לצורה פולינומיאלית (שלב 3) גם בלי להפוך את הבעיה מלמטה למעלה.

כפל סדרת מטריצות

כפל סדרת מטריצות, הינה הבעיה הראשונה איתה אנחנו מתמודדים ב"תכנות דינאמי". ניתן תזכורת קטנה ביצד הרעיון עובד – אנחנו רוצים להכפיל שתי מטריצות, נגדיר אותן כ–A ו–B. כעת, על מנת שבכלל נוכל להתחיל את הכפל, אנחנו צריכים לוודא את התנאי המקדים – מספר השורות ב–A שווה למספר העמודות ב–B (או להיפך). אם תנאי זה לא יתקיים, אין לנו בכלל מה להתחיל.

אם אתם זוכרים ליניארית א', ברכותי! אתם יכולים לדלג להמשך תיאור הבעיה, כל השאר מוזמנים להתרענן.

איך מבצעים הכפלת מטריצות? מכפילים שורה בעמודה, שורה בעמודה (עם מבטא של רייסין זה הולך יותר טוב). מכפילים כל איבר מהשורה עם המקביל לו בעמודה השניה ומחברים את התוצאה.

דוגמא פשוטה:

$$\begin{pmatrix}1 & 5\\2 & -4\end{pmatrix} * \begin{pmatrix}3\\-1\end{pmatrix} = \begin{pmatrix}\{1*3\}+\{5*(-1)\}\\\{2*3\}+\{(-4)*(-1)\}\end{pmatrix} = \begin{pmatrix}-2\\10\end{pmatrix}$$

בצורה יותר כללית ניתן לומר, כי אם יש לנו שתי מטריצות שמוגדרות במימדים שונים A(p,q), B(q,r), אז נקבל מטריצה C(p,r).

עד כאן להכפלת מטריצה אחת, ועכשיו לבעיה האמתית שלשמה התכנסנו.

מה קורה אם יש שרשרת של מטריצות? קיבלנו רשימה של מטריצות שמשורשרות אחת אחרי השניה, ואנחנו רוצים להכפיל את כל הרשימה לכדי מטריצה אחת בלבד. כמובן נקודת המוצא היא, שהרשימות מסודרות באופן כזה, שבאמת ניתן לשרשר ולהכפיל אחת את השניה. ברור שניתן פשוט לעבור ולהתחיל להכפיל כל מטריצה עם הסמוכה לה, והכל יהיה בסדר.

אבל זה לא כזה פשוט כמו שזה נדמה. עבור כל הכפלת מטריצות יש לנו מספר מסוים של פעולות שצריך לבצע, אם ניקח את המטריצות שראינו קודם, A ו–B, מספר ההכפלות יהיה p*q*r. בדוגמה שהבאנו למעלה, מדובר על 2*2*1=4, שזה קליל, אבל אם אנחנו מדברים על מספרים גבוהים יותר זה מתחיל להיות בעיה. ולא רק המספרים גדלים, אלא גם הפער בין צורות שונות של הכפלה יהיה משמעותי. נגדיר למשל את המטריצות הידועות, כמטריצות מנוגדות אחת לשניה בצורה (משוחלפות): A(50,100) B(100,50). עכשיו יש לנו 2 אופציות להכפיל את המטריצות, ובכל אחת מהן לעשות את השורה המשותפת שונה. אם ניקח למשל, את 50 להיות המספר המשותף, נקבל 100*100*50=500,000 פעולות הכפלה שנצטרך לבצע,

אך אם ניקח את ה100 להיות השורה המשותפת, נקבל 250,000 = 100*50*50. לא כיף אבל קטן בחצי (!) מכמות ההכפלות הקודמת ובוודאי שזה מאוד משמעותי.

נעלה רמה – רשימה של שלוש מטריצות <A1 A2,A3> כאשר $A_1(10*100)\ A_2(100*5)\ A_3(5*50)$.

בשביל לחשב את הפתרון הזה, נצטרך לעשות שתי הכפלות שונות, קודם בין שתי מטריצות, ולאחר מכן בין התוצאה של הקודם למטריצה הנותרת (חשוב לציין, שהמטריצה תיווצר לנו אחרי ההכפלה, תהיה מתאימה למטריצה הראשונית הצמודה לה), מה שנותן לנו בעצם שתי אופציות להכפלות שאת כמות ההכפלות הסופית נחשב עכשיו:

$((A_1,A_2)A_3) = ((A_1A_2)+(A_{12}A_3)) = (10*100*5)+(10*5*50) = 5{,}000+2{,}500 = 7{,}500$

$(A_1(A_2A_3)) = ((A_1A_{23})+(A_2A_3)) = (10*100*50)+(100*5*50) = 50{,}000+25{,}000 = 75{,}000$

השיטה הראשונה חסכה לי פי 10 הכפלות!

בוודאי שכל רשימה עם מספר שונה תיתן לנו אפשרויות שונות ומשונות שהפער ביניהן הולך וגדל, ואנחנו צריכים למצוא את הדרך האופטימלית להזין כמה שפחות הכפלות במהלך העבודה.

רק בשביל להבין את רמת הסיבוכיות של בדיקה מלאה של כל האופציות העומדות בפנינו, נעשה את החישוב הבא – נניח שיש לנו מערך של n מטריצות. נבחר מקום אקראי שנקרא לו k בתוך המטריצה, ונחלק את המערך לשני חלקים שיסומנו k, n-k. עבור כל הצבות הסוגריים האפשריות לבחירה מבין המערך הקיים נגדיר את P(n) להיות המספר של ההצבות האפשרי. כמובן שהסיפור הזה רקורסיבי, ולכל חלק שנבחר יהיה עוד תת חלק וכו'. ועכשיו, נזיז את הk למיקום אחר. שוב, נוצרו המון אופציות ותתי פרמוטציות חדשות. על מנת לסכם את כל האופציות, מקבלים את נוסחת הנסיגה הבאה:

$$P(n) = \begin{cases} 1 & \text{אם } n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{אם } n \geq 2 \end{cases}$$

בסופו של דבר, הפירוק של הנוסחה הזאת מגיע לרמה שנקראת **מספרי קאטאלאן**, שזה בעברית יפה "מלנתלפים" - $\Omega = 4^n/n^{3/2}$

הנוסחה הנ"ל גדלה בצורה אקספוננציאלית ולא נוחה בכלל לשימוש. אם ניקח למשל 10 בתור גודל המערך, יש לנו בערך 33,159 אפשרויות לבדיקה, ובהצלחה לנו עם זה, בואו נחפש דרך נוחה יותר.

אפיון המבנה של הפתרון האופטימלי

דיברנו על כך שהשלב הראשון בתכנות הדינאמי, הוא עצם ההגדרה של מה ייחשב לנו פתרון אופטימלי. אם כן, לפי ההגדרה שתיארנו קודם על הצבת ה-k בתוך המערך של המטריצות באופן שייתן לנו את כמות ההכפלות המינימלית, וכן גם בתת הבעיה – המערך החדש מ1 ועד k אנחנו צריכים להציב בו נקודה מסוימת שתחלק את ההכפלות שלו בצורה אופטימלית, עד שנגיע בסוף להכפלה של שתי מטריצות שתהיה נבונה וקלילה ומשם נחזור לעשות את ההכפלות עצמן.

הפתרון הרקורסיבי

נגדיר כעת את הבעיה באופן הבא – יש לנו את המערך A המכיל n מטריצות שיסומנו כל אחת $<A_1,A_2...A_n>$, נגדיר את התחומים הנדרשים לבדיקה של תתי הבעיות כ- i, j כאשר $1 \le i \le j \le n$. ונגדיר בהתאמה את m[i,j] להיות הדרך הזולה ביותר לבצע את ההכפלות הנ"ל.

כמובן שלפי ההגדרה, יכול להיות שנגיע למצב בו i=j, ובמקרה זה אין לנו הכפלות ופשוט מחזירים m [i, j] = 0. אך מה קורה כאשר $i < j$? אז אנחנו משמשים בהצבה חדשה של k שתהיה בטווח $i \le k \le j$ ונמצא עבורה מה יהיה הערך המינימלי. איך נעשה את זה? ניכנס לתוך כל חלק ונבדוק שם וחוזר חלילה.

בסופו של דבר כאשר נגיע למכלה הסקלרית בעצמה, העלות שלה תהיה פשוט העלות של הכפלת השורות בעמודות שמוגדרת ב $P_{i-1}P_kP_j$. עכשיו על מנת לחבר את כל התוצאה להחזרת תשובה אמיתית, הנוסחה תהיה בצורה הבאה:

m [i, j] =	Min{	m [i, k]	+ m [k+1, j]	$+ P_{i-1}P_kP_j\}$
	התוצאה המינימלית המבוקשת	התוצאה האופטימלית עד לנקודת החלוקה	התוצאה האופטימלית לחלק השני	הוספת ההכפלה של שתי המטריצות המתקבלות

ואחרי שהבנו את שני חלקי הנוסחה הרקורסיבית, נתאר אותה באופן שלם:

$$m[i,j] = \begin{cases} 0 & \text{אם } i = j \\ \text{Min}\{ m[i,k] + m[k+1,j] + P_{i-1}P_kP_j \} & \text{אם } i < j \end{cases}$$

נתונה שרשרת של מטריצות שעלינו להכפיל אותם בזוגות. אנחנו צריכים למצוא את הדרך היעילה ביותר להכפיל אותם.

חישוב הערך האופטימלי "מלמטה למעלה"

ראינו כבר קודם שחישוב כל העלויות הוא יקר מאוד וברמה אקספוננציאלית, למרות שבסופו של דבר, מספר תתי הבעיות לא באמת עולה על n^2 . אמנם גם n^2 לא להיט, אבל באופן יחסי הוא הרבה יותר מוצלח מהנוסחה שהבאנו. אנחנו שואפים להתקרב לדבר הזה כמה שניתן. לצורך זה אנחנו בונים טבלה חדשה בגודל מערך המטריצות, על מנת לשמור בה את עלויות החישובים השונים. טבלה זאת תקרא m[i, j] וכל קריאה אליה תחזיר לנו את התוצאה האופטימלית עבור ההכפלה בטווח שבין i ל-j. בנוסף, נבנה טבלה דומה שתיקרא s, ובה נשמור את הנקודות k עבורם מצאנו חלוקה שהיא אופטימלית לאותו טווח. כאשר אם נראה ש s [1,6] = 3, המשמעות היא שהחלוקה תהיה [1,3][4,6].

נראה את האלגוריתם ונבין איך הוא עובד:

```
Matrix-Chain-Order(p)
  n = length[p]-1
  for i = 1 to n                    //  איפוס הטבלה כאשר מדובר על מפגשים של
        m [i, i] = 0                //  אותה מטריצה
  for l = 2 to n
        for i = 1 to n – l +1
```

```
j = i + l -1                          // הגדרת שאר הטבלה בצורה משולשת
m [i , j] = ∞                         // הכנסת ערכי מקסימום לכל התאים
for k = i to j – 1
      q = m [i, k] + m [k + 1, j] + Pi-1PkPj   // בדיקת כמות ההכפלות עבור שני החלקים הנתונים
      if q < m [i, j]                  // בדיקה האם קיבלנו תוצאה שהיא אופטימלית
         m[i, j] = q                   // הכנסה לטבלה של הערך המינימלי
         s [i, j] = k                  // הכנסה לטבלה של מיקום החלוקה
return m and s                        // החזרת שתי הטבלאות
```

על מנת להבין איך פועל האלגוריתם, נשתמש בדוגמה שמופיעה בספר של קורמן, אך ננסה לבנות אותה לאט יותר.

קיבלנו 6 מטריצות בגדלים הבאים, כאשר נחלק כל מימד של מטריצה לעמודת הכפלות P_i:

מטריצה	ממדים
A1	30*35
A2	35*15
A3	15*5
A4	5*10
A5	10*20
A6	20*25

עמודה	מס' הכפלות
P_0	30
P_1	35
P_2	15
P_3	5
P_4	10
P_5	20
P_6	25

שלב ראשון

יצירת המטריצה m ואיפוס השורות הדומות:

j\i	1	2	3	4	5	6
6						0
5					0	
4				0		
3			0			
2		0				
1	0					

שלב שני

הכנסת ערכי מקסימום לטבלה:

j\i	1	2	3	4	5	6
6	∞	∞	∞	∞	∞	0
5	∞	∞	∞	∞	0	
4	∞	∞	∞	0		
3	∞	∞	0			
2	∞	0				
1	0					

שלב שלישי

נתחיל מהבסיס – $i = 1, l = 2, j = 1+2-1 = 2$

חישוב ה-q יהיה $q = m[1,1] + m[2,2] + P_0P_1P_2$

$q = 0 + 0 + 30*35*15$

$q = 15,750$

עכשיו, 15,750 הוא גבוה אבל הוא לא אינסוף, ולכן $m\ [1, 2] = 15,750$

למעשה כל האלכסון הראשון של כל מטריצה עם הצמודה לה, יחסית פשוט לחישוב:

j\i	1	2	3	4	5	6
6	∞	∞	∞	∞	5,000	0
5	∞	∞	∞	1,000	0	
4	∞	∞	750	0		
3	∞	2,625	0			
2	15,750	0				
1	0					

עכשיו נבדוק את האלכסון השני, שמוצא לנו את הפתרון האופטימלי עבור טווח של 3 מטריצות. כמובן שהאופציות הן רק שניים – או צמד המטריצות הראשון ואז להכפיל בנותר, או להיפך. נבדוק את האפשרויות עבור $m\ [1, 3]$:

$i = 1,\ j = 3,\ k = 1 \rightarrow m\ [1, 1] + m\ [2,3] + P_0P_1P_3\ = 0 + 2,625 + 30*35*5 = 7875$

$i = 1,\ j = 3,\ k = 2 \rightarrow m\ [1, 2] + m\ [3, 3] + P_0P_2P_3 = 15,750 + 0 + 30*15*5 = 18,000$

ברור לנו שעכשיו נרשום כי $m\ [1, 3] = 7,875$.

נמשיך הלאה ונמלא את האלכסון הזה:

j\i	1	2	3	4	5	6
6	∞	∞	∞	3,500	5,000	0
5	∞	∞	2,500	1,000	0	
4	∞	4,375	750	0		
3	7,875	2,625	0			
2	15,750	0				
1	0					

שלב רביעי

מכאן זה כבר מתחיל להסתבך אבל זה רק עניין טכני. פשוט לוקחים את הגבול התחתון (i), ומתחילים ממנו ועד j-1 ובבל פעם מחשבים בהתאם. כמובן שבגלל שאנחנו עובדים מלמטה למעלה, עשינו לעצמנו עבודה הרבה יותר קלה.

<u>טיפ קטן לעבודה על הכפלת מטריצות</u> – בשלב הזה, שיש לנו כבר יותר משתי הכפלות, אנחנו נוטים להסתבך עם ההזזה של המספרים, אבל אין צורך להילחץ. אם רוצים לעשות את כל זה בצורה פשוטה, צריך רק להתייחס לטווח [i,j], ולעבוד איתו. איך עושים את זה? כל טווח הכפלות הוא n-1 בייחס לטווח [i,j]. ההכפלה היחידה שלא נעשה היא ישר בין i ל-j, ולכן אם פה יש לנו 4-1, אנחנו צריכים לעשות 3 בדיקות סך הכל. עכשיו, על מנת לוודא שעושים הכל בסדר נכון – נכתוב ישר את שלושת השורות, כאשר ה- m הראשון תמיד יהיה צד ה-i, ואנחנו יכולים ישר לשים אותו משמאל. באופן דומה, הגבול תמיד יהיה ה-j והוא יהיה הערך הימני ביותר. ואז, כל מה שייותר לנו לעשות הוא למלא את הכפלות הביניים – בהכפלות השמאליות יהיה החל מ-i בסדר עולה עד j-1, ומצד ימין החל מi+1 ועד ל-j. באותו אופן, הכפלות ה-P יהיו אותו דבר, בקצוות זה תמיד אותו דבר, והמשתנה הוא רק ה-P האמצעי שהוא בסדר עולה מ-i.

נבדוק את הלולאה שרצה על m [1, 4]:

$m[1, 1] + m[2,4] + P_0P_1P_4 = 0 + 4,375 + 30*35*10 = 14,875$
$m[1, 2] + m[3, 4] + P_0P_2P_4 = 15,750 + 750 + 30*15*10 = 21,000$
$m[1,3] + m[4,4] + P_0P_3P_4 = 7,875 + 0 + 30*5*10 = 9,375$

קל לראות, כי m [1, 4] = 9,375. באותו אופן נמשיך הלאה למלא את הטבלה עד הסוף:

j\i	1	2	3	4	5	6
6	15,125	10,500	5,375	3,500	5,000	0
5	11,875	7,125	2,500	1,000	0	
4	9,375	4,375	750	0		
3	7,875	2,625	0			
2	15,750	0				
1	0					

מבחינת זמן הריצה, אנחנו עושים את הבדיקה בגודל n^2 על מערך בגודל n, מה שהופך את הסיבוכיות של כל הסיפור הזה ל $\Omega(n^3)$, וכמובן שאנחנו דורשים טבלה שתופסת מקום בגודל n^2 (גם אם אנחנו לא משתמשים בכולה, אנחנו מגדירים את זה כתטא).

למעשה, מה שלא ציירתי פה זה את הטבלה s. האלגוריתם עצמו דואג בכל פעם לעדכן את הטבלה בחלוקה המתאימה בה השתמשנו. כך שאם למשל עבור הדוגמה האחרונה בחרנו ב k=3 בתורה הפתרון האופטימלי עבור m[1,4], אז בטבלה s נכניס לשם את הערך 3.

בניית פיתרון אופטימלי

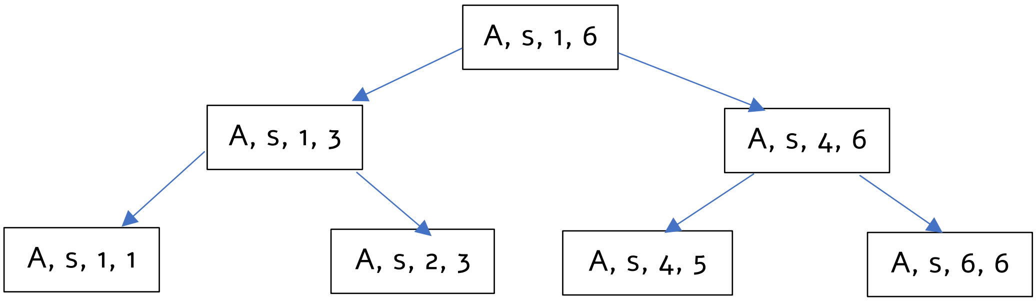
כל האלגוריתם שעבדנו עליו עד עכשיו, מתייחס בעיקר למציאת המינימום הכפלות שאנחנו צריכים לבצע, אבל עדיין לא אמרנו מה החלוקה הבי נכונה. בשביל זה אנחנו בודקים את הטבלה s.

j\i	1	2	3	4	5
6	3	3	3	5	5
5	3	3	3	4	
4	3	3	3		
3	1	2			
2	1				

כמו שאמרנו קודם, כל מספר מבטא פה את ה-k עבורו החלוקה לאותו טווח היא אופטימלית עבור [i,k][k+1,j]. מה שזה נותן לנו, הוא את האפשרות לחשב באופן רקורסיבי את מערך המטריצות ולקבוע איפה לשים כל סוגריים. האלגוריתם של שרשור ההכפלות יקבל את מערך המטריצות A, הטבלה s, ואת הטווח שבודקים [i, j], ויוצג באופן הבא:

```
Matrix-Chain-Multiply(A, s, i, j)
  if j > i                                  // בדיקה שלא סיימנו לעבור על תתי הבעיה
        X = MCM(A, s, i, s[i,j])            // הכנסה רקורסיבית של שני החלקים לתוך מטריצה חדשה
        Y = MCM(A, s, s[i,j]+1, j)
        return Matrix-Multiply(X,Y)         // הכפלת המטריצות שהתקבלו
 else return Ai                             //  אם התנאי לא מתקיים, אז הגענו למטריצה בודדת
```

נריץ עכשיו את האלגוריתם על המערך שקיבלנו, ונעקוב אחרי עץ הקריאות:



כאשר אנחנו מגיעים לרמה הזאת, אנחנו יכולים לחלק את הזוגות על פי הרמה שהגענו אליה:

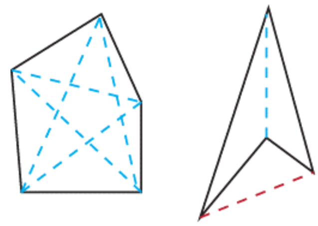
$$(((A_1(A_2A_3))((A_4A_5)A_6)$$

אז קיבלנו את התוצאה האופטימלית של ההכפלות, ואת הסדר הנכון של הכפלת המטריצות, והכל בא על מקומו בשלום.

בעיית אופטימיזציית מצולעים – טריאנגולציה

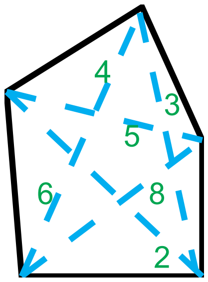
עכשיו אחרי שהבנו עד הסוף את כפל המטריצות, נוכל לעבור לבעיית הטריאנגולציה. זה לא נראה קשור, אבל זה יתחבר בהמשך.

נתחיל עם מספר הגדרות חשובות, ואז נגדיר את הבעיה עצמה – **מצולע** – אני מאוד מקווה שלא באמת צריך להסביר את זה, הוא עקום (כלומר לא קו) סגור (כלומר יש בו שטח) במישור, המורכב מקטעים ישרים. **צלעות** – אלו המקטעים השונים המרכיבים את המצולע, ומחוברים ביניהם ב**קדקודים**. כל השטח שתחום על ידי המצולע מוגדר כ**פנים המצולע**, כאשר **שפת המצולע** – הנקודות שעל הצלעות, מפרידות את השטח לביוון **חוץ המצולע**. אנחנו מדברים בבעיה זה על **מצולע קעור** – שעבור כל שני קדקודים במצולע, אם נמתח קשת, או **מיתר**, נהיה או בפנים המצולע או על השפה.

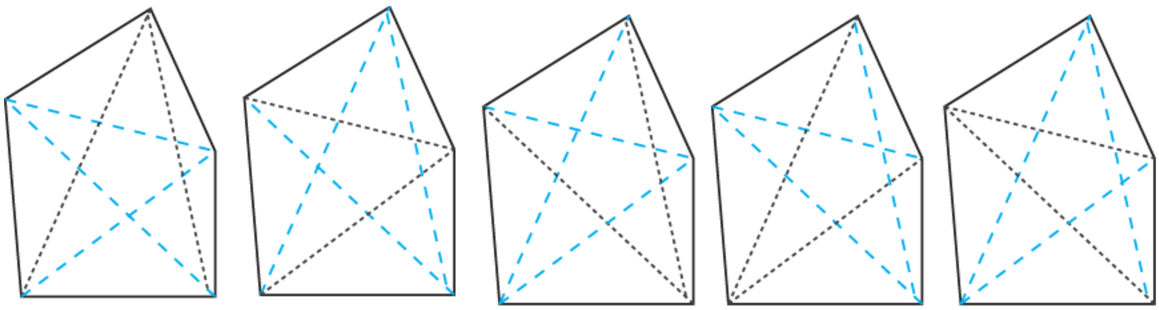


המצולע משמאל הוא קעור, ואילו סמל הפדרציה הבין-כוכבית הוא לא קעור, בגלל הקשת החיצונית שם.

אנחנו מעוניינים לקחת כל מצולע שנקבל, ולחלק אותו למספר משולשים מקסימלי, ולעשות את זה בדרך הטובה ביותר, כלומר למתוך כמה שפחות קווים ולעשות אותם קצרים בכל האפשר. מי שעשה את מבוא להנדסת תכנה, זוכר בוודאי שדיברנו על כך שכל צורה תלת מימדית מעובדת למעשה ממשולשים קטנטנים. אנחנו מעוניינים לממש את הדבר הזה בצורה כמה שיותר אופטימלית, כלומר, אנחנו לא רוצים לצייר סתם משולשים לא יעילים שיתקעו לנו את כל הרינדור של התלת מימד, אלא לעשות הכל כמה שיותר יעיל. נדגים את הכוונה שלנו. נניח ואנחנו נקבל מצולע מסוים, שעבור כל המיתרים שלו יש פונקציית משקל מתאימה –



נתעלם כרגע מהמשקל הנתון על השפה, מאחר ואנחנו בכל אופן נשתמש בו, אבל כבר המצולע הזה שיש בו "רק" חמישה מיתרים יש לנו לא מעט אפשרויות – כמובן שאנחנו לא יכולים לחתוך מיתרים, ולכן מרחב האפשרויות יהיה כבה –



כמובן, שבכבל שהסיבוך של המצולע יעלה, כמות האפשרויות תעלה, ובהתאם המשקל של המשולש ישתנה. על מנת לחשב את המשקל הנכון עבור כל משולש נתון, נשתמש בנוסחה הדי-פשוטה הבאה:

$$w(\Delta v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_k v_i|$$

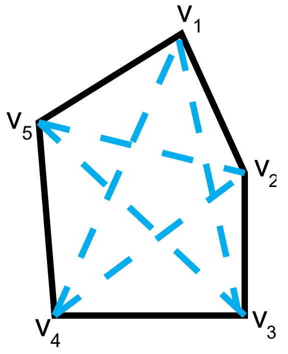
כאשר כל סימון v מבטא לנו קדקוד מסוים במשולש (Vertex).

הקבלה להצבת סוגריים מלאה – אפיון המבנה

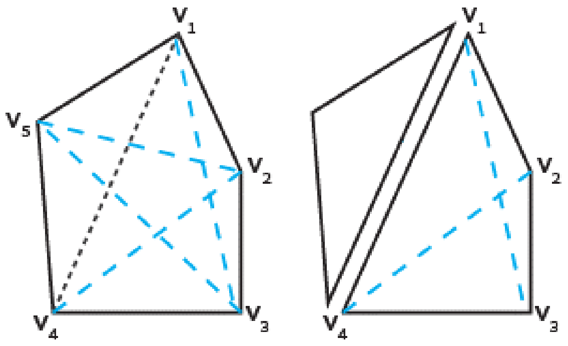
כאן אנחנו מתחילים להכנס לתחום הכפלת המטריצות. איך זה קשור בכלל? למעשה, אם נזכר ברעיון של כפל המטריצות, אנחנו חיפשנו איזה דרך, לסדר את ההכפלה בצורה מתאימה, כאשר כל k שהגדרנו, יצר

לנו אוסף בעיות חדשות שפשוט נשארו אותו דבר אבל רק בהגבלה של k. באופן דומה, אם ניקח כל מצולע ונבחר מיתר שאנחנו עובדים עליו, אז הגדרנו איזה משולש ראשוני, ויש לנו עכשיו בעצם סוג של תת בעיה עם המצולע החדש. נתאר את זה בצורה ויזואלית, עם החיתוך השמאלי ביותר מבין מה שהבאנו למעלה –

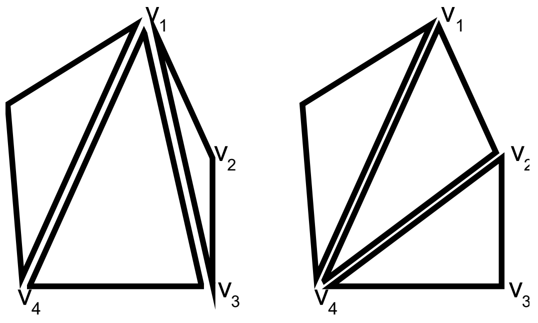
בהתחלה נתון לנו המצולע הבא –



נניח שהחלטנו שהמיתר הראשון שנבחר יהיה v_1v_4. נסמן אותו, ונפרוס אותו החוצה מהמצולע –



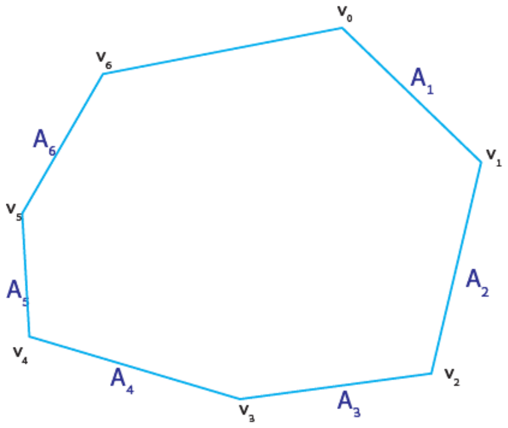
עכשיו יש לנו מצולע חדש שיש בו שני מיתרים שאני יכול לבחור מהם, וכל אחד יביא לי תוצאה שונה –



אם נחזור לבעיית כפל המטריצות, נניח שיש לנו את רצף המטריצות $A_1A_2A_3A_4A_5$, והחלטנו להכפיל קודם את A_4A_5, מה שיישאר לנו הוא לבחור עכשיו בין $(A_1A_2)A_3$ לבין $A_1(A_2A_3)$, וכבר ראינו קודם, עד כמה המשמעות גדולה בין שתי האפשרויות. עכשיו רואים את המשותף כבר בין שתי הבעיות, ורק נשאר להסביר איך אנחנו משתמשים בזה לטובתנו.

את סדר הכפלת המטריצות, אנחנו הצלחנו לייצג בעזרת עץ, שכל צומת פנימית הוגדרה כתיחום של סוגריים, וכל עלה היה אחת המטריצות. אנחנו נשתמש בעץ ההכפלות באופן דומה מאוד.

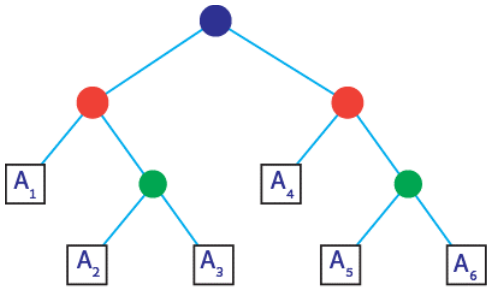
נשתמש בדוגמה של הכפלת המטריצות שקיבלנו, ונראה איך אנחנו יכולים להשתמש בזה בשביל הטריאנגולציה. עבור זה ניקח את המשובע (מצולע של שבע צלעות) הבא, וניתן לכל צלע שם כמו אחת המטריצות, ואת הקדקודים נמספר כמו העמודות:



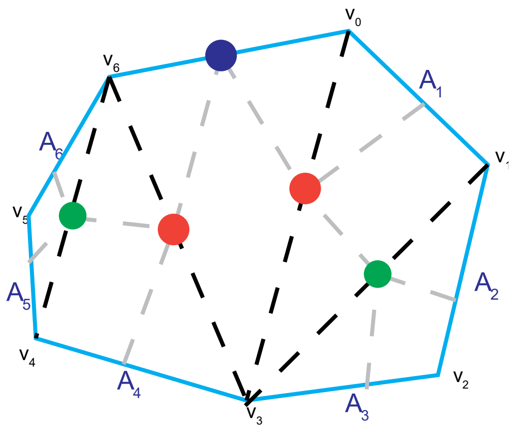
נניח שכל המשקלים מסודרים באופן דומה לגודל המטריצות, כי אנחנו עדיין לא מנסים להוכיח נכונות אלא רק להראות את בניית הדרך. את סדר הכפלת המטריצות אנחנו חילקנו באופן הבא –

$$(((A_1(A_2A_3))((A_4A_5)A_6)$$

וכמו כן, בנינו את עץ ההכפלה המתאים לו –



אנחנו נשתמש באותו העץ, כאשר העלים יהיו הקשתות שמספרנו, והצמתים יהיו מפגש של קשתות בתוך מיתרים. קצת מסורבל, אבל בדוגמא יפה עם צבעים הכל יהיה (קצת) יותר מובן, ונשתמש במשובע שהבאנו קודם –



כל צומת פנימית, מבטאת בעצם מיתר שאנחנו משתמשים בו. כאשר העלים מגדירים את איזור החסימה. למשל – A_2A_3 חסומים באופן ישיר על ידי המיתר v_1v_3, ולכן סימנתי את החיבור ביניהם בקו בהיר יותר. הם מצידם, למעשה נחתכו כבר מהמצולע, ולכן החיבור ממשיך בצורה דומה עם המקטע של A_1, והם מתחברים ביחד למשולש תחת המיתר v_0v_3. אותו דבר קורה גם בצד השני, ובסוף הכל מתאחד לשורש – הקשת שאותה לא מספרנו.

כלומר, אם נמצא איזה שיטה שדומה לכפל המטריצות ברמת האופטימיזציה שהיא מעניקה, נוכל לבנות לנו עץ מתאים בתוך המצולע שיבנה לנו טריאנגולציה אופטימלית. עכשיו נשאר רק למצוא את ההמרה הזאת.

מאחר אנחנו יכולים להראות שבל מופע של כפל מטריצות אנחנו יכולים להעביר הלאה לבעיה דומה בטריאנגולציה (אבל לא להיפך!), נוכל להגדיר את ההתאמה כך שבמקום רשימה של מטריצות $A_1..A_n$, או ליתר דיוק רשימה של ממדי מטריצות, אנחנו נקבל רשימה של קדקודים $<v_1..v_n>$, כך ש- $w(\Delta v_iv_jv_k) = P_iP_kP_j$. כלומר אנחנו מגדירים את המשקל של הטריאנגולציה במקביל לכפל המטריצות באותם ערכים. ואת שורת ההשמה באלגוריתם Matrix-Chain-Order, אנחנו משנים להיות: $q \leftarrow m[i, k] + m[k + 1] + w(\Delta v_iv_jv_k)$, כך שלמעשה, אנחנו נבדוק עבור כל מיתר את האופטימום של תתי הסידורים האפשריים שלו, ולבסוף נחזיר למעלה את הסידור האופטימלי.

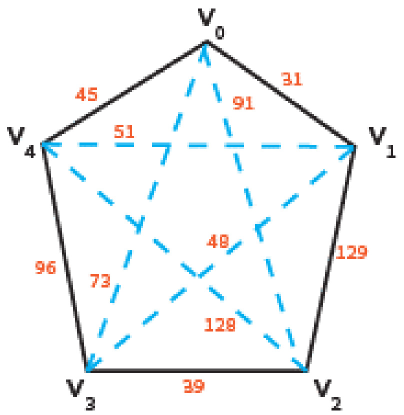
פתרון רקורסיבי

נגדיר את אותם הגדרות בדיוק שעשינו לכפל המטריצות עם השינויים המתאימים – $t[i, j]$, כאשר $1 \le i \le j \le n$ ייתן לנו בסופו של דבר את המשקל של הטריאנגולציה המינימלית של המצולע שקדקודיו הם $<v_{i-1}...v_j>$. כאשר את המצולע שהוא בעצם קו $<v_{i-1},v_i>$ נגדיר שהמשקל שלו יהיה שווה 0 (מה שיהיה גם תנאי העצירה). לאור הגדרות אלו, נוכל לומר ש-$t[1, n]$ ייתן לנו את משקל הטריאנגולציה המינימלית.

עכשיו נגדיר את $t[i, j]$ באופן רקורסיבי, כאשר עבור כל $j-i \ge 1$ יוצר לנו מצולע בעל שלושה קדקודים לפחות. כך שעבור כל חלוקה פנימית של מצולעים בין הקדקודים $[i, j]$ בעזרת k, נצטרך לחשב את המינימום האפשרי של המצולעים $<v_{i-1}..v_k>$ ו-$<v_{k+1}..v_j>$. אם נאחד את הכל לכדי נוסחת נסיגה, יתקבל לנו הדבר הבא:

$$t[i, j] = \begin{cases} 0 & \text{אם } i = j \\ \min\{ t[i, k] + t[k+1, j] + w(\Delta v_{i-1}v_jv_k)\} & \text{אם } i < j \end{cases}$$

שזה כמובן, בדיוק אותה נוסחה רקורסיבית שהשתמשנו בה להכפלת המטריצות. עכשיו נעשה הרצה קלה של האלגוריתם על מצולע, ונראה מה התוצאה.



ניקח את המחומש הבא, ונמיר אותו לטבלה המכילה את כל מרחקים בין הקדקודים – כמובן, שנשתמש פה מטבלה משולשית, בדומה לטבלה של כפל המטריצות (כי אין לנו סיבה לעשות שני תאים (A,B) ו-(B,A)) שיכילו את אותו ערך. הטבלה תיראה כך:

j\i	V_0	V_1	V_2	V_3
V_4	45	51	128	96
V_3	73	48	39	
V_2	91	129		
V_1	31			

את הטבלה הזאת אנחנו נשאיר לרפרנס בשביל שנוכל לחשב להמשך את אורך המיתרים. ונבנה טבלה מקבילה עבור החישוב המצטבר (אני עושה את כל זה בתמצות, מאחר ומדובר באותו תהליך של כפל מטריצות), כאשר החלק החשוב באמת הוא הטבלאות שנמלא למטה שיהוו את המסלול ואת החלוקות השונות של הטריאנגולציות האופטימליות.

חישוב הערך האופטימלי מלמטה למעלה

שלב ראשון

ניקח את הטבלה ונתחיל לחשב את הערכים הבסיסים ביותר של האלכסון התחתון ביותר:

j\i	V_1	V_2	V_3	V_4
V_4				3
V_3			2	
V_2		1		
V_1	0			

j\i	V_1	V_2	V_3	V_4
V_4				0
V_3			0	
V_2		0		
V_1	0			

אנחנו נבצע פה את ההכפלות של המטריצות, כאשר את כל הסכימה והתוצאה הסופית אנחנו נשרשר למעלה לעבר (V_0V_4), בצורה של העץ שראינו מקודם. כל שלב בהכפלה יהיה בעצם הגדלה של המצולעים כאשר נתחיל מבנייה של משולשים, ולאט לאט נראה את כל האפשרויות העומדות בפנינו.

בטבלה הימנית אנחנו מכניסים את הערך של ה-k שהגדרנו שיהיה החלוקה האופטימלית לאותו חישוב. כמובן שבין שני קדקודים סמוכים, פשוט נגדיר את הנמוך מבין שניהם.

שלב שני – מצולעים בעלי 3 צלעות

האלכסון השני יטפל לנו בשלושה משולשים, שיתנו לנו את האפשרות לעשות חישובים עתידיים, בזכור "מלמטה מעלה". אמנם יש לנו יותר משלושה משולשים, אך נכון לרגע זה, אנחנו לא משתמשים במשולשים שמקושרים לצלע V_4V_0.

$t[1, 2] \rightarrow \Delta v_0v_1v_2 = 31+91+129 = 368$

$t[2, 3] \rightarrow \Delta v_1v_2v_3 = 129+39+48 = 216$

$t[3, 4] \rightarrow \Delta v_2v_3v_4 = 39+96+128 = 263$

את השורה הזאת נוכל להכניס כמו שהיא. למעשה דבר שלא הכנסתי פה, הוא החישוב שהיה ברקע שבדק גם את האפסים מצדדי המשולשים, אבל כמובן שזה מיותר.

נעדכן את הטבלאות בהתאם-

j\i	V_1	V_2	V_3	V_4
V_4			3	3
V_3		2	2	
V_2	1	1		
V_1	0			

j\i	V_1	V_2	V_3	V_4
V_4			263	0
V_3		216	0	
V_2	326	0		
V_1	0			

שלב שלישי – מצולעים בעלי 4 צלעות

בשב זה אנחנו נבדוק שניים משני הטרפזים שמתחבאים לנו – $t[1,3]$ שלמעשה מבטא לנו את קבוצת הקדקודים $<v_0, v_1, v_2, v_3>$, ואת $t[2, 4]$ שמכיל את $<v_1, v_2, v_3, v_4>$. שימו לב – אנחנו מכסים פה את כל המחומש מלבד הצלע שממנו אנחנו אמורים לסיים את כל התהליך. כל k שנבחר ייתן לנו אפשרות לחלק את הטרפז לשני משולשים שונים, כאשר אחד מהם אנחנו כבר מכירים מהאיטרציה הקודמת –

$t[1,3]$

$k = 1 \rightarrow \Delta v_0v_1v_3 + t[2, 3] = 152 + 216 = 368$

$k =2 \rightarrow \Delta v_0v_2v_3 + t[1, 2] = 203 + 326 = 529$

$t[2,4]$

$k = 2 \rightarrow \Delta v_1v_2v_4 + t[3, 4] = 308 + 263 = 571$

$k = 3 \rightarrow \Delta v_1v_3v_4 + t[2, 3] = 195 + 216 = 411$

ועכשיו נוכל למלא את המצולעים שהגדרנו תחת הטבלאות המתאימות –

j\i	V_1	V_2	V_3	V_4
V_4		3	3	3
V_3	1	2	2	
V_2	1	1		
V_1	0			

j\i	V_1	V_2	V_3	V_4
V_4		411	263	0
V_3	368	216	0	
V_2	326	0		
V_1	0			

שלב רביעי – חישוב המחומש

בשלב האחרון, אנחנו לוקחים את הצלע איתה לא התעסקנו בכלל (v_4v_0) ומתחילים לבדוק את האפשרויות השונות לגביה. תכלס, אנחנו לוקחים את הצלע המתאימה בסיס המשולש, ובכל פעם בודקים האם הקדקוד

יכול להיות באחד מהקדקודים הפנויים, ואם כן מה ההשלכה של זה על הטריאנגולציה הסופית. כמובן שחלק גדול מזה כבר מחושב לנו, ואנחנו נדרשים רק לקחת את הפרטים, אבל נראה בהמשך איפה זה משתנה –

$t[1,4]$

$k = 1 \rightarrow \Delta v_0v_1v_4 + t[2,4] = 127 + 411 = 538$

$k = 2 \rightarrow \Delta v_0v_2v_4 + t[1, 2] + t[3, 4] = 264 + 326 + 263 = 853$

$k = 3 \rightarrow \Delta v_0v_3v_4 + t[1, 3] = 214 + 368 = 582$

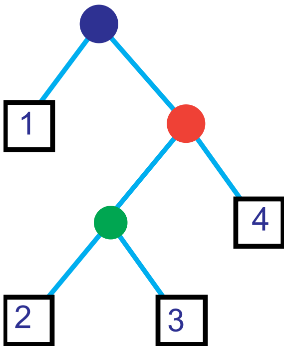
כמובן שנבחר את k = 1, ונמלא את הטבלה בהתאם –

j\i	V_1	V_2	V_3	V_4
V_4	538	411	263	0
V_3	368	216	0	
V_2	326	0		
V_1	0			

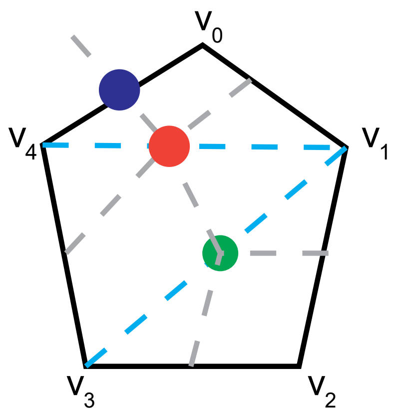
j\i	V_1	V_2	V_3	V_4
V_4	1	3	3	3
V_3	1	2	2	
V_2	1	1		
V_1	0			

ועכשיו כל מה שנותר לנו הוא לבנות את עץ ההכפלות, ולהכניס אותו למצולע המקורי. קלי קלות.

בשביל זה נעבור כמובן מלמעלה, ובכל חלוקה תגדיר לנו איזור חדש לסוגריים, עד שנקבל את מחרוזת ההכפלות הבאה – ((2,3)4)1), שאותה נמיר לעץ הבא:



ואת העץ הזה נוכל לשתול בתוך המחומש שלנו –



כך שהמיתרים הפנימיים הצבועים בתכלת, הם אלו שיגדירו לנו את הטריאנגולציה האופטימלית. מי שעדיין מסתפק בעניין, אפשר לראות שבכל אופן שלא נבחר, הצלעות של השפה תמיד יהיו בכל פורמט של

טריאנגולציה, כך מה שיש לנו לבחור זה את החיבור הפנימי הנמוך ביותר, ושתי הצלעות האלו, מחזיקות את המשקלים הנמוכים ביותר. מש"ל.

בעיית תרמיל הגב בשלמים – הגרסה הדינאמית

בעיה זו הינה וריאציה של אותו תרגיל שראינו כבר קודם על בעיית תיק הגב[3]. ההבדל בדרישות כאן, שיש לנו התחשבות שהיא לא חד מימדית ולכן קשה יותר להגיע לפתרון שהוא נכון.

נתונים לנו n פריטים בצורה מסודרת בווקטור $[i_1...i_n]$. עבור על פריט i בווקטור יש משקל מסוים w_i וערך בספי v_i. הגנב הידוע מגיע עם תיק גב שיכול להכיל עד משקל W ורוצה לצאת עם ערך V מקסימלי.
ההגבלה: כל פריט יכול להילקח אך ורק בשלמותו.

קל לראות, שאם נתחשב רק באחת מהדרישות של משקל / ערך כאינדיקציה עיקרית לא נוכל להתקדם בצורה וודאית על מנת למלא את התיק בצורה נכונה.

הפתרון S יוצג לנו בצורה של וקטור בינארי, כאשר בכל אינדקס i יסומן אותו פריט האם נלקח או לא על ידי 1/0 (1 – נלקח, 0 – נשאר). באופן פורמלי, ניתן לומר שאנחנו רוצים שעבור הווקטור S יתקיים לנו $\sum_{i=1}^{n} v_i\, x_i$ שיהיה מקסימלי[4] ובמקביל יקיים גם $\sum_{i=1}^{n} w_i\, x_i \le W$ (משקל הפריטים בתיק לא יחרוג מקיבולת התיק)

כמובן שאם S הוא הווקטור של כל הפריטים שכבר נלקחו, אזי $S' = S - \{i\}$, שעבורו תת הבעיה תהיה עבור $[i_1...i_{n-1}]$ ומשקל התיק יהיה $W-w_i$, והפתרון הכללי יהיה $V_i + V(S')$.

הסבר העיקרון של הפתרון שנביא: ננסה לעבור פריט פריט, ונראה על כל פריט, האם ישתלם לנו לקחת אותו או לא. כיצד נדע האם ישתלם לנו? על פי חישוב של כל האפשרויות הנגררות מכל אחת מהאופציות שעלולות להיבחר. באופן שיטתי נעבור על כל פריט ובסופו של דבר נוכל לראות את השלב הבא. עד כאן אפיינו את המבנה (שלב 1). רק בשביל לפרט בצורה שהיא יותר מוחלטת נגדיר כך:

ניקח את הפריט ה-n.
אם אנחנו מחליטים שאנחנו מכניסים אותו לתוך השק – תת הבעיה שתיווצר לנו תשתנה לסכימה הבאה: $\sum_{i=1}^{n-1} v_i\, x_i$ וכמובן שגם המגבלה תשתנה בהתאם ל $\sum_{i=1}^{n-1} w_i\, x_i \le W - w_n$.
אם נחליט שלא לקחת את האיבר ה-n אזי תת הבעיה תישאר לאותו הטווח, אך תת המבנה של המשקל לא יוגבל כמו הקודם, ויהיה $\sum_{i=1}^{n-1} w_i\, x_i \le W$.

כמובן, כמו שכבר אמרנו קודם, יש לנו פה עץ ריקורסיה ענק בו נעשה את אותם חישובים מספר פעמים – אם נקח *i=3* ונרצה לבדוק אותו, נעשה את החישוב האם כדאי לקחת אותו ביחס לכל איבר אחר, וכן הלאה עד שנגיע לתוצאה הראויה. אם נעשה את כל הקומבינציות האלה, נגיע לסדר גודל של 2^n שהוא כמובן סדר גודל שאנחנו לא מסוגלים לחשב.

בעת ננסה לבנות לנו נוסחת נסיגה שנוכל לעבוד איתה:

---

[3] בפרק הקודם עסקנו בבעיית תיק הגב בחלקים, ובשיעורי הבית מופיעה וריאציה על תיק גב בשלמים, אך עם דרישות מצומצמות יותר.

[4] כאשר יהיה בווקטור 1 זה יביא לנו את ערך הפריט ו-0 בוודאי שלא יחזיר כלום.

עבור כל פריט נחשב את המיקום שלו בווקטור ואת המשקל שלו, בצורה הבאה –

$$C[i,w]=\begin{cases} 0 & \text{If } i = 0 \text{ or } w = 0 \\ C[i-1,w] & \text{If } w_i > W \\ \text{Max}\{V_i + C[i-1, W-w_i], C[i-1,w]\} & \text{If } i > 0 \text{ and } W \geq w_i \end{cases}$$

// אם אין מה להכניס, או שאין מקום בתיק
// אם חורגים מהמשקל, אז לוקחים את
// הפתרון הקודם
// כל מקרה אחר – בודקים מה יניב ערך גבוה
// יותר – הוספה של הערך הנוכחי למה
// שהצטבר, או להמשיך הלאה בלי לקחת את
// הפריט

עד כאן מימשנו את שלב נוסחת הנסיגה (שלב 2), ועכשיו עלינו לחשב את כל האופציות השונות. אבל בשביל שלא נעבור על כל דבר עשרות אלפי פעמים, אנחנו פשוט בונים מטריצה בה נשמור את כל הערכים אותם אנחנו מקבלים. נתחיל לבנות אותה מהחלק הפשוט יותר, ונעלה למעלה לחלקים היותר גדולים שירכיבו לנו בסוף את הפיתרון האופטימלי. יש פה הרבה עבודה שחוזרת ולא כיפית (בגדול) אבל זה דרך בטוחה ופשוטה יחסית לבצע את זה.

את המטריצה נבנה בצורה הבאה – מספר <u>השורות</u> יהיה n+1 שורה אחת עבור כל פריט שייכנס, ובן שורה אחת עבור 0 בו אף פריט לא נכנס (השורה הראשונה בנוסחת הנסיגה). מספר <u>העמודות</u> יהיה בצורה בדידה כל האופציות בו תיק הגב יתמלא, וגם תיק ריק – 0.

ניקח דוגמא מוחשית, ונראה איך מחשבים. נעשה פה כמה דילוגים, אבל הרעיון הכללי יובן על ידי זה.

עבור הדוגמה נגדיר n=5 W=10. w_i=[2,2,6,5,4] (וקטור המשקל לפי i) v_i = [2,3,5,4,6] (וקטור הערכים של i).

בעת בניית הטבלה נכניס באופן אוטומטי אפסים בשורה ובעמודה האפסיים:

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0										
2	0										
3	0										
4	0										
5	0										

בעת, ננסה לחשב את C[1,5]. משמעות החישוב הוא – אם יש לנו רק 5 ק"ג פנויים בשק, האם שווה לנו לקחת את הפריט 1 או לא?

על פי נוסחת הנסיגה נחשב – C[1,5] = max (2 + C[0,3], C[0,5]).

כמובן שבשלב זה, נוכל באופן אוטומטי, לראות שברגע שיש 0 באחד החלקים של הנוסחה התוצאה תהיה גם היא 0, ולכן פיתוח הנוסחה ייראה כך: C[1,5] = max(2,0).

ולכן C[1,5] = 2. נמלא זאת בטבלה.

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0					2					
2	0										
3	0										
4	0										

5	0										

תבל'ס, את הפריט הראשון אנחנו ישר מכניסים ברגע שאנחנו מגיעים למשקל המינימלי בו הוא יכול להיכנס לתוך השק (במקרה שלנו החל מ-2). נמשיך ונחשב (לכאורה) את כל השורה ונקבל את התוצאה הבאה:

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2
2	0										
3	0										
4	0										
5	0										

עכשיו נוכל לחשב את השורה הבאה – כל חישוב מתייחס ל-i-1 ולכן ברגע שנמלא שורה אחת נוכל לעבור לשורה הבאה ללא שום חשש.

נחשב עכשיו את $C[2,7]$.

$C[2,7] = max(3+C[1,5], C[1,7])$.

את שני החלקים של נוסחת הנסיגה כבר יש לנו, ואין צורך לחשב. כיף! ולכן נמשיך ונפתח -

$C[2,7] = max(3+2,2)$.

$C[2,7] = max(5,2)$.

$C[2,7] = 5$.

שוב – ננסה להבין את זה באופן אינטואיטיבי – אנחנו מדברים ברגע על להכניס רק שני פריטים, במגבלות השונות של משקל התיק. כלומר, אם אחד הפריטים בעל משקל נמוך יותר, ברגע שאפשר נכניס אותו ונכתוב את הערך שלו בטבלה. ברגע שאנחנו מגיעים למשקל שרק אחד מהם יכול להכנס ויש לנו אפשרות בחירה, או ששניהם במשקל שווה, מכניסים את הפריט בעל הערך הגבוה יותר. בשלב השלישי, אם יש לנו מצב להכניס את שני הפריטים, בוודאי שנכניס את שניהם, ונמשיך את השורה הזאת עד הסוף באותו ערך.

ונמלא גם את שאר השורות עד לשורה האחרונה:

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2
2	0	0	3	3	5	5	5	5	5	5	5
3	0	0	3	3	5	5	5	5	8	8	10
4	0	0	3	3	5	5	5	7	8	9	10
5	0										

כמובן, שניתן לראות שכל חישוב בעצם מביא לנו את האופציה הטובה יותר מבין כל האפשרויות המוצעות לנו. על מנת להביא את התוצאה הטובה ביותר נבדוק עכשיו את $C[5,10]$, שהוא הפינה הימנית התחתונה שסוגרת לנו את הטבלה

$C[5,10] = max(6+C[4,6], C[4,10])$

C[5,10] = max(6+5,10)

C[5,10] = max(11,10)

.C[5,10] = 11

נמלא את שאר הטבלה:

i\w	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	2	2	2	2	2	2	2	2
2	0	0	3	3	5	5	5	5	5	5	5
3	0	0	3	3	5	5	5	5	8	8	10
4	0	0	3	3	5	5	5	7	8	9	10
5	0	0	3	3	6	6	9	9	11	11	11

עכשיו – התוצאה אליה הגענו, היא למעשה שהרווח האופטימלי שאנחנו מסוגלים להגיע אליו הוא 11.

הדבר היחיד שנותר לנו לבדוק (שלב 4), מה הסידור שיעניק לנו את הפתרון הנ"ל. בשביל זה יש שני דברים שבדאי לציין:

דבר ראשון, מה שכבר הזכרנו בעבר – יכול להיות שיהיו לנו שני פתרונות שיביא לנו את התוצאה המתאימה. השילוב הוא אפשרי ואם זה עומד בדרישות, הכל בסדר.

דבר שני, אנחנו צריכים לדעת לנתח את התוצאות שמופיעות בטבלה.

אם נסתכל על התוצאות, נבים שהדרך היחידה להשיג 11 הוא לקחת את הפריט החמישי. כל קומבינציה שלא תכלול את הפריט הזה, תגיע מקסימום ל10. באופן דומה אנחנו נחפש רק את הפריטים שאותם אנחנו חייבים לקחת בשביל להגיע לפתרון האופטימלי – ברגע שניקח את הפריט החמישי, שהמשקל שלו הוא 4, אנחנו קובצים ישר לעמודה 6, ומסתכלים על הערכים הגבוהים שלה. עכשיו, בשורה של 5, יש לנו ערך 9, אבל אנחנו לא מסתכלים עליו כי לקחנו אותו כבר, ובעצם כאילו צמצמנו את הטבלה בשורה. אנחנו רואים שהמקסימום של כולם הוא 5. איך נוצר לנו הערך הזה? אנחנו פשוט עולים עד הפעם הראשונה בה הוא הופיע – שורה 2 – כלומר לקחת את שני הפריטים הראשונים. ברגע שניקח אותם נוכל לקבל את וקטור התוצאה S= [1,1,0,0,1][5].

תת סדרה משותפת ארוכה ביותר[6] Longest Common Subsequence

בכלליות, בעיית תת הסדרה המוצגת לפנינו, היא משהו שנתקלנו בדברים דומים בעבר. נתונות לנו שתי מחרוזות עם רצפים סדורים תחת אותה קבוצת א"ב, ועלינו למצוא תת סדרה מקסימלית בין שתי הסדרות.

---

[5] מי שיסתכל, יוכל לראות שזה גם המשקל הנמוך ביותר מבין כל התוצאות האופטימליות שמביאות לנו את הערך 11, אבל מאחר שזה לא פקטור בבחירה שלנו, סתם נציין את זה ונמשיך הלאה.

[6] בהמשך נשתמש בראשי התיבות תמ"א. זה יותר קצר.

ההגבלה היחידה היא שמירה על הסדר הקיים, כאשר אנחנו יכולים להוריד כמה איברים שנרצה על מנת להגיע לתוצאה רצויה.

המוטיבציה לבעייה הזאת באה לידי ביטוי בחקר הדנ"א. סליל הדנ"א מורכב מארבעה סוגים של חלבונים ברצף מסויים. מדעני ביו-אינפורמטיקה עורכים השוואות רבות בין שני רצפים, כאשר הרצפים הם יחסית גדולים ומרחב האפשרויות לתתי סדרות אפשריות בין שתי סדרות הוא מרחב עצום. על ידי התכנון הדינאמי ניתן לפתור לפחות חלק מהבעיה בצורה שנראה בהמשך.

נגדיר כעת את מושגי היסוד לבעיה:
נתונה לנו שפה $S=\{s_1,s_2...s_n\}$ המכילה קבוצת אותיות ונתונות לנו שתי סדרות מתוך השפה $X=<x_1...x_n>$, וכן $Y=<y_1...y_n>$ באורכים כלשהם (לאו דווקא שווים/שונים). כאשר $X,Y \subseteq S$. תת סדרה תוגדר להיות בתור רצף $Z= <z_1..z_k>$ המוכל ב-S, ובנוסף, הינו תת-סדרה משותפת לX ולY כאחד. תת הסדרה חייבת לקיים את הסדר הנמצא בכל אחת מהסדרות, כאשר אין מגבלה לעקיבה בין האיברים – הווה אומר – יכול להיות שמשתמשים באיברים עם דילוג ביניהם. לדוג' – $Z = <2,4,6,7>$ הוא תת סדרה של $X =$ $<1,2,3,4,5,6,7,8,9>$, בניקוי האיברים הצבועים באדום.
מעצם הגדרה זו, נגדיר גם כי בעבור כל קבוצה של אינדקסים (איברים בסדרה מסוימת) הממויינים בסדר עולה ממש החסום בין $i_1<i_2<...<i_k$ מתקיים כי $z_i = x_{ia}$ לכל האיברים שבטווח הסדרה Z. וכמו כן – מספר האפשרויות הקיים לתתי הסדרות עומד על 2^n. לא שצריך להסביר בשלב זה של התואר, אבל עבור כל איבר יש לנו אפשרות בינארית אם יהיה או לא יהיה בתת הסדרה, מה שיוצר לנו את הקומבינציה הזאת.
עוד מספר הגדרות רלוונטיות (ודי אינטואיטיביות):

- יהיו X,Y סדרות המוכלות בסדרה S, נאמר כי Z היא תת קבוצה משותפת של X ושל Y אמ"ם
  - Z היא תת סדרה של X
  - Z היא תת סדרה של Y

מתוך ההגדרות קל לראות כי יכול להיות מצב בו $X=Y$, שיתקיים רק בתנאי ש $Z=X$ וגם $Z=Y$, כלומר ברגע שתת הסדרה Z שווה בפני עצמה לשתי תתי הסדרות, אז בוודאי ששתיהן שוות אחת לשניה. כמו כן, יכול להיות שZ תהיה סדרה ריקה – במידה ואין שום אות משותפת בין שתי הסדרות.

עכשיו נגדיר את הבעיה אותה אנחנו רוצים לפתור:

בהינתן 2 סדרות X,Y המוכלות ב-S בעיית **תת הסדרה המשותפת (תמ"א)** היא למצוא את תת הסדרה המשותפת Z בעלת האורך המקסימלי (מספר איברים מקסימלי).

נשתמש באותה שיטה שהגדרנו למציאת פיתרון דינאמי. ראשית נאפיין את מבנה הפיתרון הדינאמי, המכונה גם "הפתרון הנאיבי". (יש לציין – כל הדרך הזאת אינה "סיעור מוחות" שאנחנו מתחילים לחשוב על רעיונות, ואז מתחילים לצמצם ולזקק אותם עד לרעיון הסופי, אלא שיטה ברורה ומסודרת להגיע אל הדרך הנכונה.)

הפתרון הנאיבי

לא סתם החלק הזה נקרא פתרון נאיבי. אנחנו חושבים על הדרך הפשוטה ביותר (לכאורה) למצוא תתי רצפים קיימים – להשוות אחד אחד. ננסה את כל הקומבינציות האפשריות מבין אחת מתתי הסדרות, ונבדוק בסדרה השניה, האם תת הרצף קיים. במידה והוא קיים – נזכור את תת הסדרה ואת האורך שלה, עד שנגיע לאפשרות המקסימלית.

כמה זמן ייקח לנו לחשב את כל האפשרויות האלה? בקטנה! $O(m2^n)$ – לוקחים את אחת מהסדרות, ועבור כל אות בודקים אם קיים לנו רצף איתה במחרוזת השנייה, זה פיתרון רע. נניח שיש לנו שתי סדרות באורך 3, נצטרך לעבור 24 איטרציות. עבור 10 – 10,240. ואלו עוד מספרים קטנים. רצף דנ"א מכיל מכיל סדרות הרבה יותר ארוכות. רק ננסה לחשוב על סדרות באורך 100. 2^{100} זה מספר שיגרום לנו להתחרט שנולדנו.

אז מה עושים עכשיו? פתרון דינאמי רקורסיבי.

הפתרון הרקורסיבי

ראשית, נגדיר את ה**רישא** עבור כל סדרה $X = \langle x_1..x_n \rangle$ להיות תת הסדרה $X_i = \langle x_1...x_i \rangle$, כאשר $0 \leq i \leq n$.
זאת אומרת – הרישא יכולה להיות כל הסדרה או כלום, וכל מה שבאמצע.

בעת נגדיר את הפונקציה LCS[7] (X, Y) המקבלת שתי סדרות ומחזירה לנו את Z שהיא תת הסדרה המקסימלית. כיצד הפונקציה תעבוד? בחלוקה של הסדרה לרישא ולזנב. לצורך ההדגמה, נגיד שירד נביא מהשמיים, ובאותות ובמופתים הפך מטה לנחש, נכנס לכיתה ואמר: "כה אמר ה', תת הסדרה $Z = \langle z_1...z_k \rangle$ היא תת הסדרה המקסימלית!" ונעלם בעננת עשן. עכשיו, אנחנו רוצים לדעת האם הוא נביא אמת, או שעלינו לסקול אותו פעם הבאה שהוא מופיע.

ראשית נבדוק את הזנב של שתי הסדרות – X_n, Y_k. עבור צמד האיברים האחרון בכל סדרה, יש לנו בדיוק שלוש אפשרויות:

1. $x_n = y_m$ – אם שני האיברים האחרונים בסדרות שווים, אזי מן הסתם $x_n = y_m = z_k$ כי הרי אם הם שווים אז הם שייכים לאותה תת סדרה. ומאחר שהם שייכים לתמ"א, בוודאי שהאיבר הזה יהיה גם האיבר האחרון בסדרה Z. כמו כן, אם נמחק עכשיו מהסדרות את האיבר האחרון, אז גם מהסדרה של התמ"א יימחק האיבר האחרון, ויש לנו פה תת מבנה שנשאר באותה צורה.
במקרה בזה המשך האלגוריתם יהיה – $LCS\ (x_{n-1},y_{m-1})$
2. $x_n \neq y_m$ **וגם** ($x_n = z_k$ **או** $y_m = z_k$, אך לא שניהם) – למעשה יש פה אפשרות כפולה. האיברים האחרונים לא שווים זה לזה, אבל אחד מהם נמצא בתת הסדרה המשותפת. במקרה כזה אנחנו מבינים שיכול להיות שיהיו הסתעפויות. אמנם במבט ראשון אנחנו נוטים ללכת על האיבר שאנחנו רואים שקיים בתת הסדרה שקיבלנו, אבל מאחר ויכול להיות תתי סדרות נוספות שיהיו באותו אופטימום, או אפילו יותר טובות, אנחנו בודקים את שני הכיוונים.
במקרה בזה, נבדוק מה נותן לנו ערך מקסימלי בין שני הפיצולים של חיתוך הסיפא של כל אחד מהם.
3. $x_n \neq y_m$ **וגם** $x_n \neq z_k$ **וגם** $y_m \neq z_k$ - אפשרות זאת בעצם אומרת ששום דבר לא מתאים.
במקרה בזה המשך האלגוריתם יהיה – $LCS\ (x_{n-1},y_{m-1})$. אך בניגוד למקרה הראשון ה"מצביע" בסדרה Z יישאר באינדקס z_k.

בעת אחרי שקבענו את סדר הפעולות אותו נבצע, אנחנו יכולים לבנות את נוסחת הנסיגה הרקורסיבית. עבור זה נגדיר את התצורה הבאה – יהיו X,Y סדרות באמור. נסמן ב C[i,j] את אורך הסדרות (מספר האיברים) של $LCS\ (x_i,y_j)$,

0   If i = 0 or j = 0   *// תנאי העצירה – אחת הסדרות התרוקנה*

---

[7] Longest Common Subsequence

$$C[i,j]=\begin{cases} C[i-1,j-1]+1 & \text{If } i,j>0 \text{ and } x_i=y_j \\ Max\{C[i,j-1],\ C[i-1,j]\} & \text{If } i,j>0 \text{ and } x_i\neq y_j \end{cases}$$

// המקרה הראשון – סופרים 1 עבור התמ"א
// וממשיכים לבדוק הלאה
// המקרה השני – בודקים את שני הקצוות
// האפשריים עבור המקסימום

חישוב ערכו של פתרון אופטימלי "מלמטה למעלה"

חשוב להבין, התכנון הדינאמי הזה מממש בפועל את הנוסחה הרקורסיבית על ידי שימוש במטריצת זיכרון. באופן זה, אנחנו חוסכים את כל הכניסה עבור כל הסתעפות של כל החישובים פעם אחר פעם. מסיבה זאת, אנחנו מתחילים את החישוב מלמטה למעלה, שייתן לנו חיסכון בדרך הארוכה יותר.

על מנת לפתור בצורה אופטימלית, נשתמש במטריצה דומה מאוד לזו שעשינו לבעיית השק. נגדיר את המטריצה לגודל (n+1, m+1) ביחס לגודל הסדרות, כך שייתן לנו גם שורת אפסים עבור כל תת סדרה, ונתחיל למלא את השורות.

לצורך הדוגמא נגדיר את שתי הסדרות הבאות: Y = <B,D,C,A,B,A> X = <A,B,C,B,D,A,B,>, שיובילו אותנו ליצירת הטבלה הבאה:

I \ J		0	1	2	3	4	5	6
		Y_j	B	D	C	A	B	A
0	X_i	0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

עכשיו, נמשיך לבנות את i=1. האיברים הראשונים בY אינם שווים לx_i, ולכן נמשיך עד שנגיע ל C [1, 4]. את האיברים שאין בהם שיוויון בכלל נמלא בתור 0, וכשנגיע ל(1,4), ונראה שהוא שווה נזכור 1, ונבדוק עבור C[i-1,j-1], הווה אומר – C[0,3]. הפלא ופלא, יש לנו כבר את תוצאת ה0 שלו (תנאי העצירה שקבענו), נוסיף אותו לו שזכרנו, ונמשיך הלאה ל C[1,5]. אמנם כבר מצאנו שיש לנו אופציה מתאימה ב(1,4), אך מאחר שיבולים להיות מספר רצפים, שיקיימו את האופטימום, אנחנו ממשיכים לבדוק.

C[1,5] לבשעצמו, אינו שווה, אך בבדיקה של שתי הזנבות (שלמעשה הם בדיקה של האיבר השמאלי, והאיבר העליון בטבלה) נוכל למצוא את (1,4) שערכו 1, שהוא בוודאי גדול יותר מ C[0,5] = 0. ולכן נסמן גם את C[1,5] = 1. עכשיו, על מנת לעשות לנו חיים טיפה יותר קלים, אנחנו מסמנים חץ קטן, שיציין לנו מי האיבר אליו אנחנו מתייחסים – אם מדובר באיבר שבעצמו נותן תוצאה, כלומר, יש לנו התאמה, אז החץ יהיה באלכסון, בשביל לקיים את (i-1, j-1). אך אם אנחנו מתייחסים לאחד מהקצוות השניים, החץ יהיה למעלה או ימינה, בהתאמה לערך המקסימלי.

בעת, הטבלה שלנו תיראה באופן הבא:

	J	0	1	2	3	4	5	6
I		Y_j	B	D	C	A	B	A
0	X_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

נמשיך ונמלא את הטבלה. אין צורך שנעבור פה צעד צעד, ובסוף נקבל את טבלת הפלא הבאה:

	J	0	1	2	3	4	5	6
I		Y_j	B	D	C	A	B	A
0	X_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

בנית פתרון אופטימלי מתוך המידע שהושג

עכשיו, אחרי שהטבלה כבר מלאה, וכל החיצים מיושרים למקום הנכון, אנחנו יכולים פשוט לעבור מנקודה (i+1, j+1) שהיא הפינה המקסימלית, ולעבור עם החיצים עד שנקבל את התמ"א המבוקש. כמובן, שעל פי כל מה שאמרנו מקודם, אנחנו צריכים להתייחס בסופו של דבר רק לאיברים עם חץ אלכסוני, שהם האיברים השווים בין שתי הסדרות –

	J	0	1	2	3	4	5	6
I		Y_j	B	D	C	A	B	A
0	X_i	0	0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2

4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

על פי כל האמור, תת הסדרה Z = <B,C,B,A> היא תת הסדרה המשותפת המקסימלית.

פסאודו אלגוריתם

לעיתים נידרש לכתוב פסאודו-קוד שיממש את כל מה שתיארנו. הקוד לפונקציה שתיארנו עכשיו נראה כך:

```
LCS (X,Y)
  m ← length[X]        // איפוס התחלתי למטריצה. קביעת ערכי הרוחב והגובה על פי אורך הסדרות
  n ← length[Y]
  for i ← 1 to m do    // הכנסת ערך 0 בשורה ובעמודה הראשוניים
        c[i,0] ← 0
  for j ← 1 to n do
        c[0,j] ← 0
```

החלק הראשון של האלגוריתם, התעסק באתחול של הטבלה, ועכשיו מתחילים את הבדיקה עצמה (אמנם הפרדתי פה בטקסט, אבל זה רק מתודי, הכל פונקציה אחת!). הבדיקה עכשיו עוברת בין שתי אותיות, אחת מכל תת מחרוזת ובודקת האם הם שווים. אם כן, מעדכנים שמצאנו התאמה, מסמנים "חץ" אלכסוני (האות D) וממשיכים לחפש נוספים. אם לא, בודקים איזה חיתוך של תת מחרוזת יוציא לנו תוצאה טוב יותר ומעדכנים את החיצים למעלה (U) או שמאלה (L).

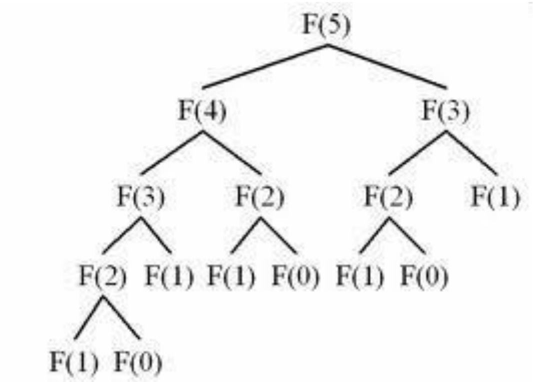
```
  for i ← 1 to m do
        for j ← 1 to n do
           if xi = yj                    // המקרה  הראשון – שני האיברים האחרונים זהים
                 c[i,j] ← c[i-1, j-1] +1  // כניסה מחודשת עם הרישא, וזכירה של ערך 1
                 b[i,j] ← "D"             //  סימון חץ אלכסוני
           else                           // המקרה בו האיברים האחרונים אינם שווים
                 if c[i-1, j] ≥ c[i, j-1] // הכנסת שני קצוות שונים לרקורסיה ובדיקת מקסימום
                   c[i,j] ← c[i-1, j]
                   b[i,j] ← "U"
                 else
                   c[i,j] ← c[I, j-1]
                   b[i,j] ← "L"

  Return c and b                         // החזרת תתי הסדרות לצורך בדיקת הערכים שלהם
```

שיטת התזכור (הפתקאות) Memoization

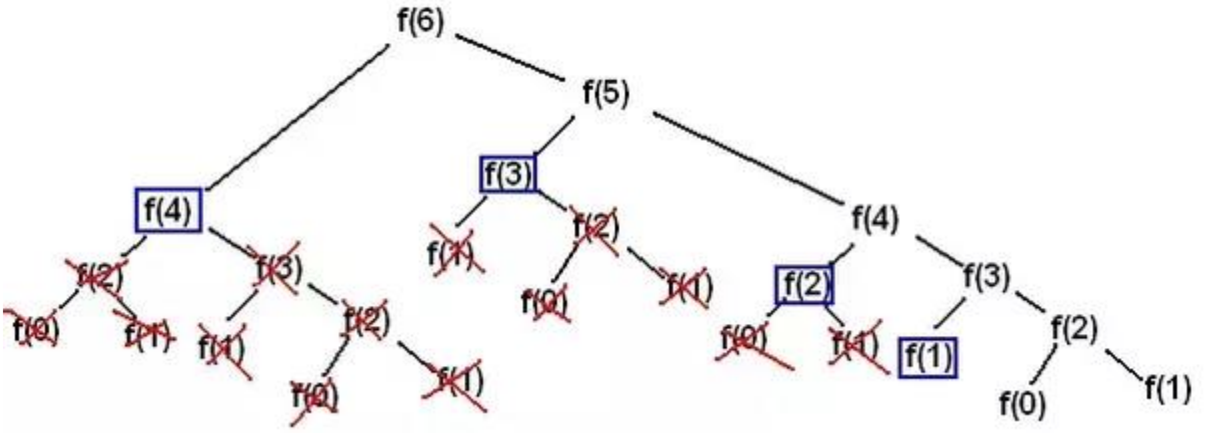
שיטת ה"תזכור" או ה"פתקאות", היא דרך שונה לפתור בעיות בתכנון דינאמי. אם עד כמה עבדנו בצורה טבלאית מלמטה למעלה – מהערכים הנמוכים יותר של הרקורסיות, כלפי הערכים הגבוהים, בשיטת התזכור אנחנו עובדים דווקא הפוך – מלמעלה למטה.

תהליך העבודה של האלגוריתם הזה עובד בצורה הבאה – ניקח את הטבלה שלנו, ונכניס בכל התאים את הערך ∞ (אינסוף). עכשיו נעבור על כל הרקורסיה באופן הרגיל מהחלקים הגדולים יותר כלפי מטה, וכאשר נגיע לאיזור בו יש לנו פיצול או כניסה עמוקה יותר, נבדוק את הטבלה. אם עדיין מופיע לנו ∞, אזי כנראה שאין לנו חישוב לאותו ערך, ונרשה לעצמנו להכנס אליו. עכשיו יכול להיות שבכמה רמות שונות של הרקורסיה נכנס לאותו חלק, אבל פעם אחת מהקריאה השניה ופעם אחת מהקריאה השלישית, מצב זה בסופו של דבר יגיע לבדי התוצאה הדרושה רק פעם אחת. כי ברגע שהראשון יגיע לתחתית אותה אנחנו מחשבים, הענף השני של עץ הרקורסיה יגיע קצת אחריו, יראה שיש לנו ערך קיים שהוא מוחלט ואינו אינסוף, ויתחיל לשרשר כלפי מעלה את התוצאות הרצויות.

על מנת להבי ויזואלית את הערך של שיטת התזכור, ניתן לראות את עץ הרקורסיה הבא:



הקריאה מתחילה מ5 ומתפצלת ל-4 ול-3, וממשיך הלאה מטה-מטה. סך הכל יש לנו 15 קריאות לפונקציה. אבל, אם נוריד את כל הקריאות הכפולות (בהנחה שאחרי פעם אחת שהפונקציה נקראת יש לנו את התוצאה שלה) נוכל לקבל את העץ הבא:



אם ניקח כל קריאה לפונקציה, יש לנו בעצם רק שני חישובים שאנחנו צריכים לעשות – ימינה ושמאלה. אם לכל קריאה בזאת נקבל רק את הערכים הסופיים, מספר הקריאות שנותר לנו כרגע הוא רק 9. כמובן

שביחס לעץ המקור שהביל 15 קריאות זה לא נראה משמעותי, אבל יש לזכור שהעץ הזה הוא קטן יחסית, רק קריאות מס-5. ביחסים גדולים יותר של עצי ריקורסיה נגיע לחיסכון משמעותי בחישוב.

נסתכל על האלגוריתם של השיטה, המחולק לשניים – אלגוריתם מילוי הערכים, ובדיקת הרקורסיה:

```
Memoized-Matrix-Chain(n)
  for i ←1 to n do
        for j i ← to n do
          m [i,j]← ∞                    // אתחול המטריצה
  return Lookup-Chain(1,n)


Lookup-Chain(i,j)
  if m[i,j] < ∞ then
        return m[i,j]                   // תנאי עצירה
  else
        if i=j then
          m[i,j] <- 0
        else
          for k <-1 to j-1 do                   // רקורסיה + זיכרון
                q <- Lookup-Chain (i,k)+LookupChain(k+1,j) + p_{i-1}p_k p_j
          if q < m[i,j] then
                m[i,j] <- q
        return m[i,j]
```

זמן הריצה של הפונקציה הוא $O(n^3)$. זמן זה הוא פולינומיאלי, ועד עכשיו זה לא היה להיט. אבל ביחס לפונקציה מעריבית (כמו למשל ה- $O(m2^n)$ שחישבנו קודם) אין ספק שזה שיפור משמעותי.

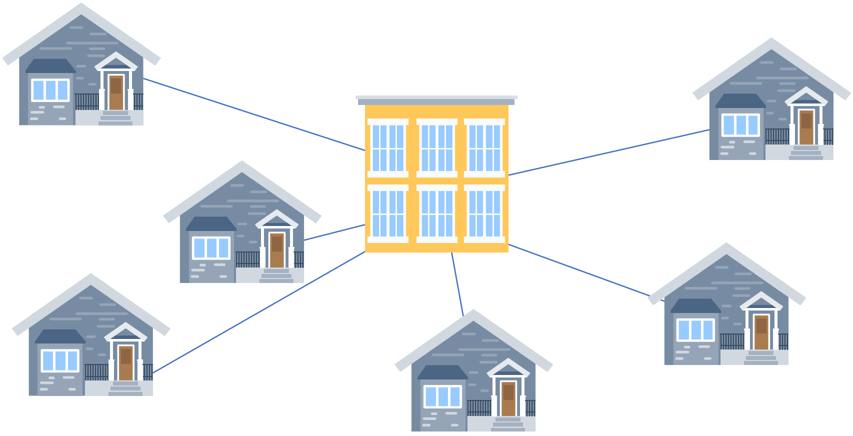
גרפים

עץ פורש מינימלי בגרף

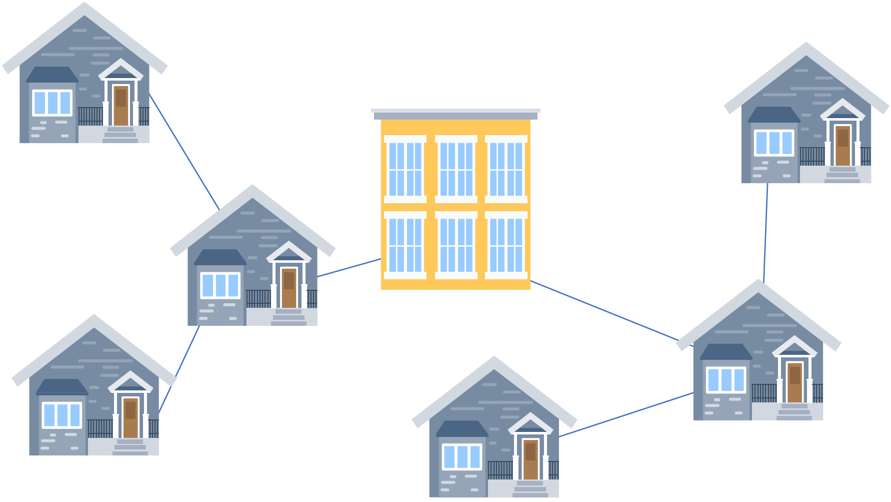
החלקים הקודמים של התכנון החמדני והדינאמי, הם הבסיס לניתוח האלגוריתמים. כעת אנחנו הולכים להתמקד בחלק השני של הניתוח, שהוא ניתוחים על עצים וגרפים. נתחיל עם עץ פורש מינימלי, ולאחר מכן נראה מסלולים קצרים, רשתות זרימה ועוד.

בבעיית העץ הפורש, ובחלק מהפתרונות האפשריים כבר נתקלנו בקורס מבנה נתונים ב', ולכן חלק זה יהיה יחסית מובן (למי שזוכר את הקורס לפחות), אך כעת נבין את הדברים יותר לעומק.

המוטיבציה לבעיית העפ"מ מתוארת בדרך כלל בתור שכונה שרוצים לפרוש בה קווי טלפון. אנחנו מחפשים דרך לעשות זאת בצורה יעילה ככל האפשר. הדרך ה"נאיבית" תהיה פשוט לשים את המרכזיה באמצע השכונה ולמשוך אליה את החוטים מכל הבתים מסביב. כמובן שזה יהיה הכי נח ומהיר, ואפילו יהיה הכי יציב – אם חוט אחד נקרע, הבעיה היא מקומית לאותו הבית ואין שום בית אחר שיושפע מזה, אך מצד שני, כמות החוטים שנצטרך לפרוש היא יחסית גדולה וזה לא ממש יעיל.



הפתרון השני, שיגיע אחרי מחשבה קצת יותר רצינית, היא לנצל את הקרבה של הבתים ולהעביר כבלים בין הבתים, כך שכל בית גם יעזור לשכן שלו להעביר את המידע הרצוי למרכזיה. כמובן, שהפתרון הזה עלול



לגרום לנו בעיה של קריסה גדולה בבעיה בכל בית, שתגרור את כל אלו התלויים בו. וכמו כן, ברמת האבטחה לא בטוח שכולם יאהבו לדעת שהמידע שלהם עובר דרך השכן, אבל אנחנו מתעסקים פה ביעילות, ולכן האבטחה שלהם לא מעניינת אותנו בכלל.

עכשיו הבנו את הרעיון של החלוקה המינימלית הזאת, אבל איך נגיע לפרישה הטובה ביותר?

בשביל זה נגדיר את בעיית העץ הפורש המינימלי.

הקלט לטובת הבעיה: **גרף G=(V,E)** [גרף (G) המורכב מקדקודים (Vertices) המקושר בקשתות (Edges)] **לא מכוון**[אין משמעות או הפרש משמעותי בין שני הקדקודים המקושרים ביניהם] , **קשיר** [כל הקדקודים מחוברים ביניהם ואין אף אחד בודד] **ופונקציית משקל w: E->R** [יש ערך מוגדר לכל צלע].

הפלט הדרוש: תת גרף T = (V,E) קשיר ללא מעגלים (עץ) עבורו הערך $w(T) = \sum_{(u,v)\in E'} w(u,v)$ מינימלי מבין כל תתי הגרפים הקשירים ללא מעגלים אפשריים.

אנחנו מחפשים להוציא מהגרף הנתון בסופו של דבר, עץ קשיר ללא מעגלים שמשקל הצלעות בו יהיה מינימלי. כמובן, שמספר הצלעות בעץ יהיה n-1. זה מהסיבה הפשוטה, שאם יהיה לנו n קדקודים בעץ, הדרך היחידה לקשר בין כולם ללא מעגל הוא צלע אחת פחות מאשר מספר הקדקודים (ניתן להוכיח זאת באינדוקציה שמתחילה משני קדקודים, אותם ניתן לחבר רק באמצעות צלע אחת. וכן על זה הדרך).

ראשית, נעבור על האלגוריתם הכללי, שרק מסביר את הרעיון של "איך אנחנו מגיעים לפתרון", ולאחריו נציג שני אלגוריתמים יותר פרטניים – קרוסקל ופרים, שמתמודדים עם הפתרון בצורה יותר קונקרטית. שני האלגוריתמים מתבססים על רעיונות שכבר למדנו במבנה נתונים, ולכן יהיו לנו מוכרים ופשוטים יחסית.

האלגוריתם הכללי למציאת עפ"מ

על מנת להבין את האלגוריתם, יש לעבור תחילה על מספר מושגים:

<u>חתך –</u> קו שעובר בתוך הגרף, ומחלק את הגרף לשני חלקים – S, V-S . החתך לא חייב להיות ישר, והוא יכול פשוט לעבור ו"לשלוף" קדקודים להיות שייכים לחלק מסוים. כמובן שה"חתך" הוא רק מושג רעיוני – אפשר להגדיר סתם רשימת קדקודים שתהיה תחת חלק מסוים בחתך גם אם זה לא נראה כאפשרי. הדבר החשוב לזכור – כל הקדקודים בגרף שייכים לאחד משני החלקים – או S, או V-S.

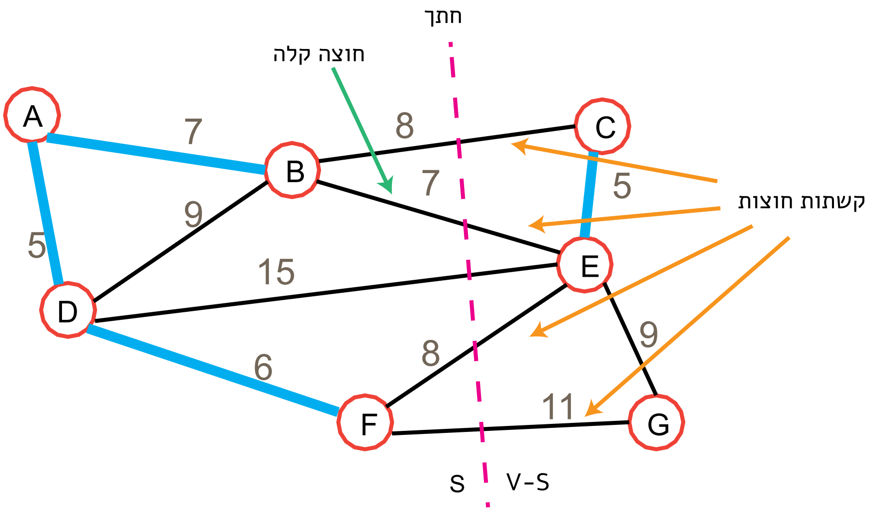
<u>קשת חוצה –</u> לאחר שחילקנו את הגרף לשני חלקים, באופן טבעי יהיו לנו קשתות (לפחות אחת) שיעברו על החתך לעבר הקדקודים בצד השני. הקשתות האלו נקראות "חוצות". כמובן שחייבות להיות קשתות באלה, מהסיבה הפשוטה שאם אין אף קשת שעוברת על החתך, אז החתך לא חותך שום דבר.

<u>חוצה קלה–</u> הקשת החוצה בעל המשקל הנמוך ביותר.

<u>חתך מכבד –</u> אנחנו מתעסקים בתהליך למציאת עפ"מ בתת גרף שאנחנו מכנים אותו A. אותו A יהיה אחר כך עץ, אבל הוא יבונה כך גם בכל שלב בתהליך. למעשה, בחירת צלע מינימלית תתואר כך:
A = A U |(u,v)|. לעניינו – חתך ייקרא "מכבד" אם לאחר החיתוך אף אחת מהצלעות החוצות לא שייכת לA.

<u>קשת בטוחה –</u> הקשת החוצה הקלה (ע"ע) בחתך מכבד (ע"ע).

בציור הבא, נוכל לראות גרף עם חיתוך. סימנתי את כל המושגים שראינו עכשיו על הגרף. הצלעות בצבע תכלת, הן הצלעות המשוייכות כבר ל-A, מה שכמובן הופך את החתך הקיים פה לחתך מכבד, ואת החוצה הקלה לקשת בטוחה -



עכשיו, לאחר שהבנו את כל המושגים, נוכל להסתכל על האלגוריתם עצמו:

```
GeneralMST(G,w)
A ←ϕ
While A does not form a spanning tree
  do find an edge (u,v) that safe for A
        A← A U |(u,v)|
return A
```

האלגוריתם הכללי עובד בשיטה שכבר ראינו לא מעט בקורס הזה, שניתן להגדיר תחת קבוצות הפתרונות "טא-דא!". או במילים אחרות – אין פה שום פיתרון אמיתי. אנחנו מתחילים מ-A כאשר הוא ריק, והלולאה מוגדרת ב"כל עוד זה לא מה שרצינו, תמשיך לעשות מה שאנחנו רוצים". בכוונה אני ככה מקצין את זה, כי ההגדרה הכתובה באלגוריתם "find an edge (u,v) that safe for A", היא בדיוק השאלה – מה הצלע הבאה שניקח לA? התשובה לשאלה הזאת, מוגדרת בשני האלגוריתמים הבאים שיפרטו איך למצוא צלע "בטוחה".

אבל, אם ניקח את ארבעת המושגים שהסברנו עליהם מקודם, נוכל לומר באופן כללי, שהתהליך הוא כזה:

1. חותכים את הגרף (באופן מכבד, כמובן).
2. בוחנים את הקשתות החוצות.
3. לוקחים את החוצה הקלה.
4. מבצעים שוב, עד שיש לנו את העץ.

אין צורך אמיתי להוכיח את הדבר הזה, כי מדובר במשהו די אינטואיטיבי – אם ניקח תמיד את הצלעות הקטנות, ברור שנגיע לתוצאה טובה (היתה הוכחה לזה בכיתה שנמצאת גם בצילומי האוניברסיטה הפתוחה בעמ' 110, אם יזדמן לי אשלים את זה).

בגדול, האלגוריתמים הבאים, כמו שנאמר כבר, מפרטים קצת יותר, והופכים את ה"טא-דא!" מקסם למדע.

האלגוריתם של קרוסקל (Kruskal)

אלגוריתם זה פורסם לראשונה בשנת 1956 על ידי ג'וזף קרוסקל, ומסתמך על מבנה נתונים לקבוצות זרות.

כזכור – כאשר אנחנו מדברים על קבוצות זרות יש לנו שלוש פעולות בסיסיות:

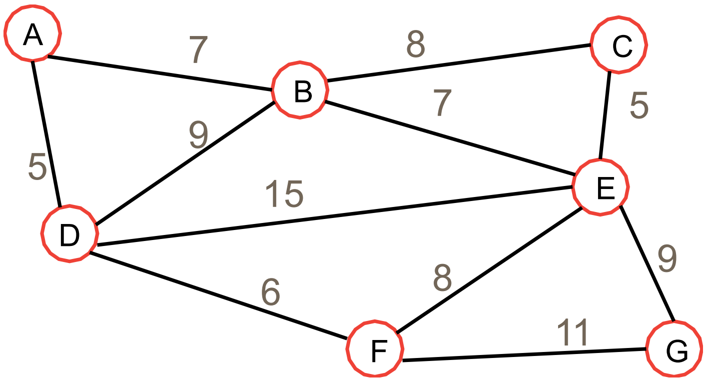
1. **Make-Set(x)** – יצירת קבוצה.
פעולה זו מתבצעת רק פעם אחת בכל רצף האלגוריתם, ומהווה בעצם בנאי של הקבוצות. הפעולה מקבלת ארגומנט יחיד שיתקיים כאיבר היחיד ברגע בקבוצה, איבר זה גם יישאר הנציג של הקבוצה להמשך, כל עוד לא יוכרח אחרת. יש לציין, אין חשיבות לאיבר הראשון, והוא לא אמור לקיים איזה חוקיות מסוימת, אלא פשוט האיבר הראשון הופך בצורה אוטומטית לנציג.
2. **Find-Set(x)** – מחזירה מצביע ל**נציג** של הקבוצה המכילה את האיבר x. המצביע לא מוביל לאיבר עצמו שהוכנס לפונקציה, אלא לנציג בלבד. (אם בשני זימונים שונים על שני איברים שונים הערך המוחזר מהפונקציה יהיה זהה, אנו מבינים שמדובר על כך שהאיברים נמצאים באותה קבוצה, וכן להיפך – שתי קבוצות שונות יחזירו שני נציגים שונים)
3. **Union(x,y)** - מקבלת שני פרמטרים (**לאו דווקא נציגים**) ומאחדת בין שתי הקבוצות השונות של האיברים. האיחוד בין שתי הקבוצות מתבצע על ידי קריאה כפולה לFind-set בעזרת שני הערכים המוכנסים לפונקציית האיחוד.

הרעיון בבסיס האלגוריתם – ראשית, נמיין את כל המשקלים הנתונים לנו מהצלעות. לאחר מכן, נפעיל את האלגוריתם לקבוצות זרות, על פי סדר הצלעות מהקטן לגדול.
הייחוד של הפעולות הללו, הוא שאם נקבל שני איברים השייכים כבר לאותה קבוצה, אין לנו צורך לחבר ביניהם, וכך נשמור על הדרישה לגרף ללא מעגלים.

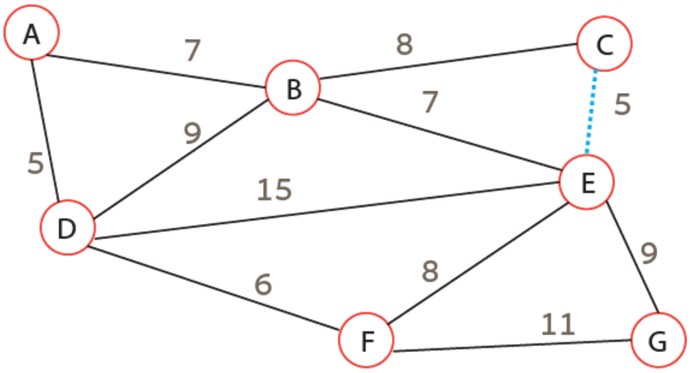
פסאודו קוד של האלגוריתם ממומש באופן הבא:

```
KRUSKAL(G):
  A = Ø                                          // סימון העץ המוחזר בתור עץ ריק
  foreach v ∈ G.V:
    MAKE-SET(v)                                  // יצירת סטים בודדים מכל האיברים
  foreach (u, v) ordered by weight(u, v), increasing:
     if FIND-SET(u) ≠ FIND-SET(v):               // בדיקה האם הקדקודים שייכים כבר
        A = A ∪ {(u, v)}                         // לאותו עץ. במידה ולא, מאחדים אותו
        UNION(u, v)                              // עם העץ המוחזר
return A
```

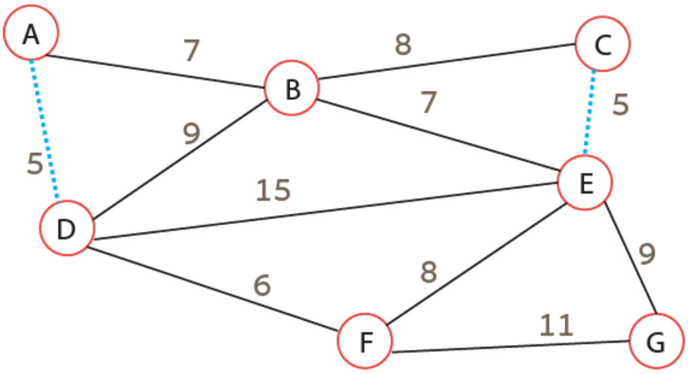
נעבור בעת על דוגמת מימוש על מנת לראות את כל השלבים באופן ברור:



זה הגרף הראשוני הנתון לנו. 7 קדקודים עם לא מעט צלעות. נתחיל בשתי הצלעות הקטנות ביותר:

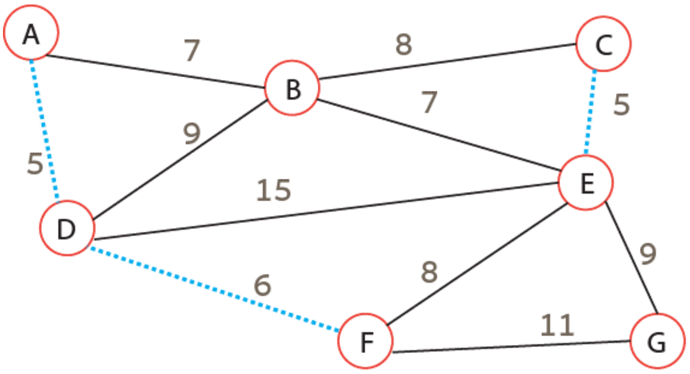


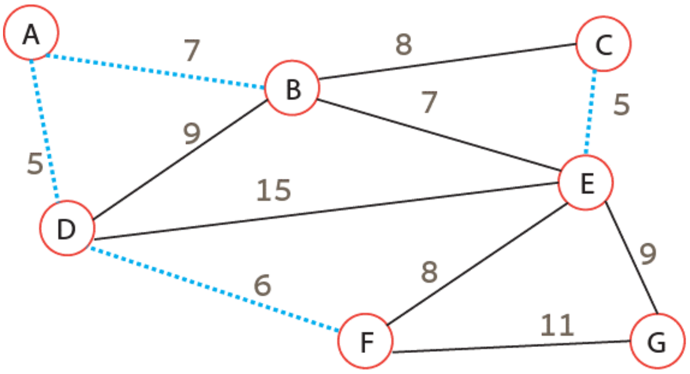
כמובן, שאין הבדל מהותי איזה צלע ראשונה ניקח מבין השניים כאשר הם בעלי אותו משקל. התחלתי בצלע (c,e). הצלע הבא בעלת אותו משקל היא (a,d) ונסמן גם אותה כשייכת לעץ הפורש המינימלי:



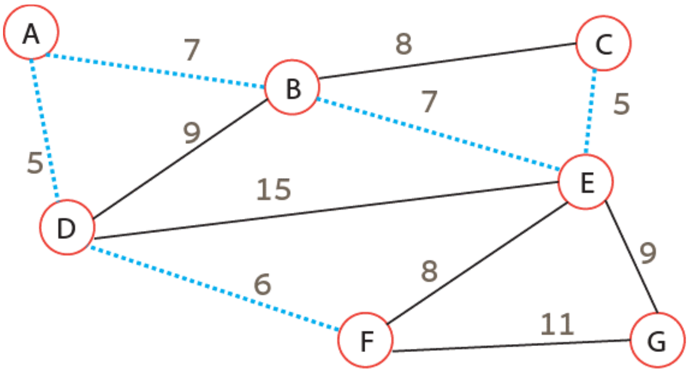
יש לזכור, כאשר אנחנו "מסמנים" צלע, אנחנו קודם כל מריצים Find-Set על כל אחד מהקדקודים, על מנת לוודא שהם כבר לא מחוברים ביניהם במקום אחר, אם המצב אכן כך, וקיבלנו נציג שונה עבור כל קדקוד – ניתן לחבר אותם.

נעבור הלאה:

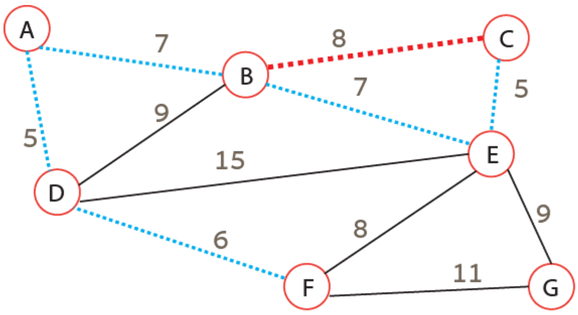


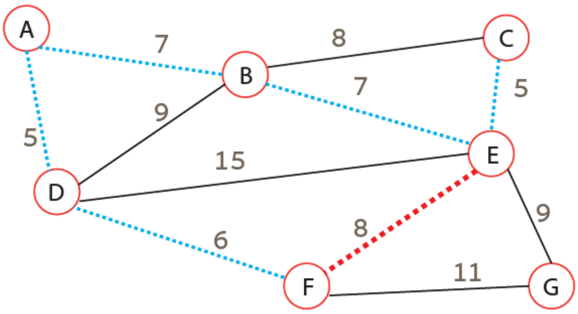


החיבור הבא ייתן לנו את הצלע השניה במשקל 7, שכבר תחבר לנו את החלקים השונים לעץ אחד. עדיין לא הסתיימה העבודה:

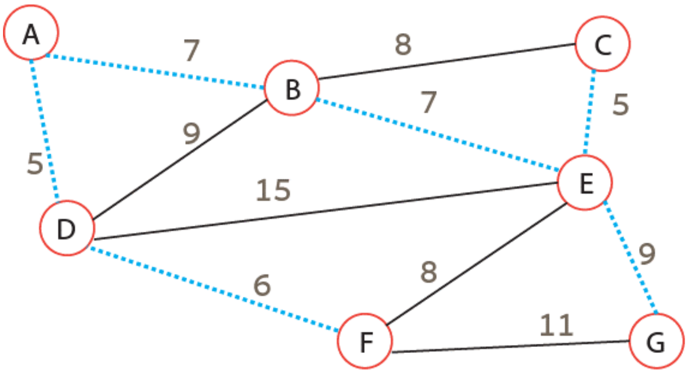


עכשיו הגענו לחלק החשוב – מכניסים את שני השמיניות ובודקים אותם. אבל ניתן לראות, שאם נצרף הצלעות במשקל שמונה, נסגור פה מעגל, או אם נסתכל על זה באופן פורמלי יותר, FindSet(B)=FindSet(C), ובכן FindSet(E)=FindSet(F):

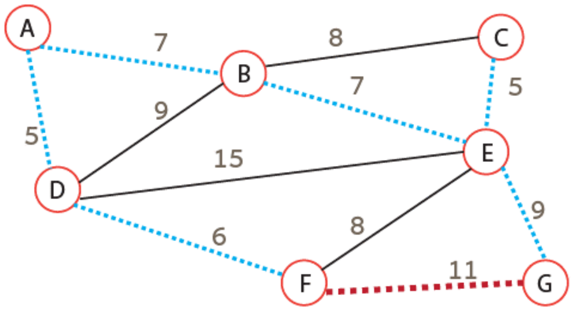


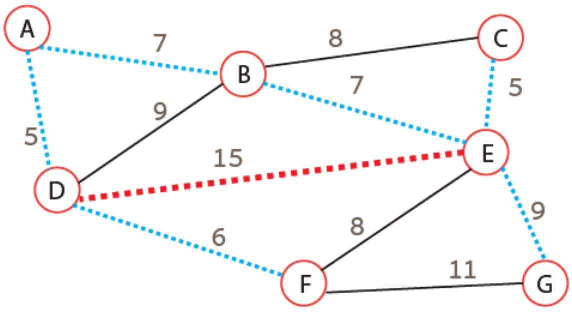


לכן, כמובן, לא נשתמש בצלעות האלה ונמשיך הלאה:

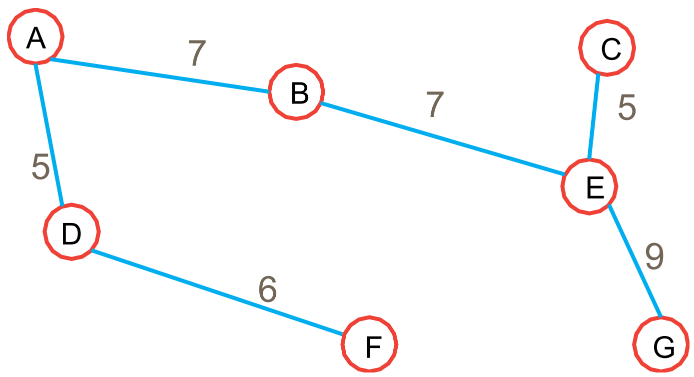


עכשיו אנחנו יכולים לעצור. מדוע? ציינו כבר קודם שיש לנו 7 קדקודים. עד כה חיברנו 6 צלעות לגרף הפורש. ובידוע – על מנת שעץ יהיה קשיר וללא מעגלים, עליו להיות עם צלע אחת פחות מכמות הקדקודים. ניתן גם לראות בגרף הזה, שכל ניסיון להוסיף צלע ייצור לנו מעגל שהוא פסול:





ועכשיו, אחרי שסיימנו להריץ את האלגוריתם, ניתן להחזיר את העץ החדש, וסיימנו את העבודה:



העץ הזה הוא העץ המינימלי ביותר שניתן להוציא תחת הגרף המקורי. האלגוריתם עבד בצורה חמדנית, על ידי לקיחת הצלע בעלת המשקל הקטן ביותר האפשרי בכל פעם. וניתן להוכיח בקלות שכל הוספה של צלע אחרת לעץ לא תיתן את האופטימליות הקיימת בו עכשיו.

מבחינת זמני ריצה:

1. מעבר ב Make-Set על כל הקדקודים O(|v|)
2. מיון הצלעות לפי המשקל – O(ElogE)
3. מעבר על הצלעות לבניית העץ O(E) (יכול להיות שקיבלנו כבר עץ מינימלי)

סה"כ O(V+ElogE + E) שזה שווה בסדר גודל לO(ElogE)

עד כאן האלגוריתם של קרוסקל.

האלגוריתם של פרים

האלגוריתם של פרים פותח לראשונה בשנת 1930 על ידי וויטיך ירניק, וקיבל את מעמדו בעקבות העבודה של רוברט פרים בשנת 1957. אלגוריתם זה אינו חמדני באופן מובהק וגם אינו אינטואיטיבי, אך יש בו יעילות. ובשונה מקרוסקל, הוא גם אינו מתמקד בצלעות אלא דווקא בקדקודים.

האלגוריתם הזה גם הוא דומה לאחד שראינו בעבר, אך יש בו הבדלים דקים.

רעיון המימוש: עבור כל קדקוד, נוסיף שני שדות: key - שיציין את הערך המינימלי המוביל אליו באותו רגע. π – שישמור מצביע לאותו הקדקוד שהkey מתייחס אליו.

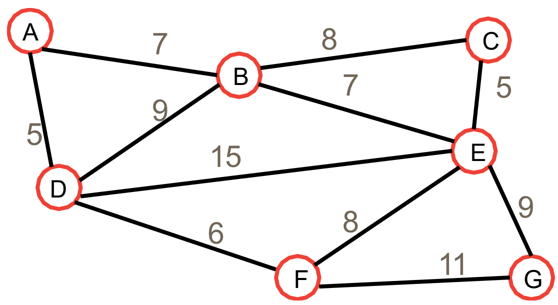
בתחילת הריצה, נאפס את המפתחות של כל הקדקודים ל ∞, על מנת שבטוח נכניס לשם ערכים מינימליים בהמשך, וניצור תור-קדימויות-מינימלי של הקודקודים ע"פ הערך Key שלהם.
נבחר קדקוד (r) באופן שרירותי שממנו אנו מתחילים לבנות את העץ, ועבור r נסמן את ה-Key להיות שווה ל-0 (הגיוני), ואת ה-π להיות שווה ל-NIL (או NULL).
מובן ש-r יהיה הקדקוד המינימלי שה"תור-קדימויות-מינימלי" שבנינו יוציא לי. כי ה-Key שלו שווה ל-0.
בעת נתחיל להוציא איבר אחר איבר מתור-הקדימויות, כאשר כל פעם ייצא לי הקדקוד u שמשקל הצלע שמחברת אותו לאחד מהקדקודים שכבר נמצאים על העץ שאני בונה, הוא מינימלי ביחס למשקל שאר כל הצלעות היוצאות מהקדקודים שכבר מחוברים לעץ (כלומר ערך ה-Key שלו הוא מינימלי מבין ערכי ה-Key של כל הקדקודים שנמצאים עדיין בתור).
בכל פעם שאני מוציא איבר מהתור קדימויות הוא כמובן נמחק ממנו, ואוטומטית מתחבר לעץ. כך שהתור

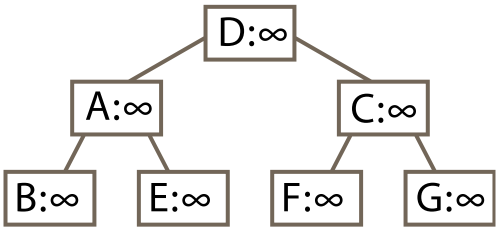
יכיל רק את הקדקודים שעדיין לא נמצאים על העץ.
עבור אותו קדקוד u שהוצאתי, אעבור על כל הקדקודים שעדיין נמצאים בתור-קדימויות ושמתחברים אליו ע"י צלע מסויימת. עבור כל קדקוד כזה v, במידה ומשקל הצלע (u,v) שמחבר אותו ל-u, קטן יותר מה-Key של v, נעדכן את ה-Key להיות אותו משקל, ואת ה-π להיות הקדקוד u שזה עתה הוצאתי מהתור (וחיברתי אותו לעץ).
נשים לב שהערך Key של קדקוד כלשהו v שנמצא בתור-קדימויות, יכול להתעדכן כמה וכמה פעמים.
כאשר בסוף כל שלב (שהתחיל בהוצאת הקדקוד u מהתור-קדימויות) הוא יציין את המשקל המינימלי מבין הצלעות שמחברות אותו (את הקדקוד v) לעץ החלקי שכבר נבנה, ושזה עתה השתנה כאשר חיברנו אליו את הקדקוד u.
קדקודים מסויימים, ערך ה-Key שלהם יכול להיות שווה ל-∞. וזה בא לציין שאין צלע שמחבר את הקודודים האלו לעץ החלקי שכבר נבנה.

פסאודו קוד עבור האלגוריתם ייראה כך:

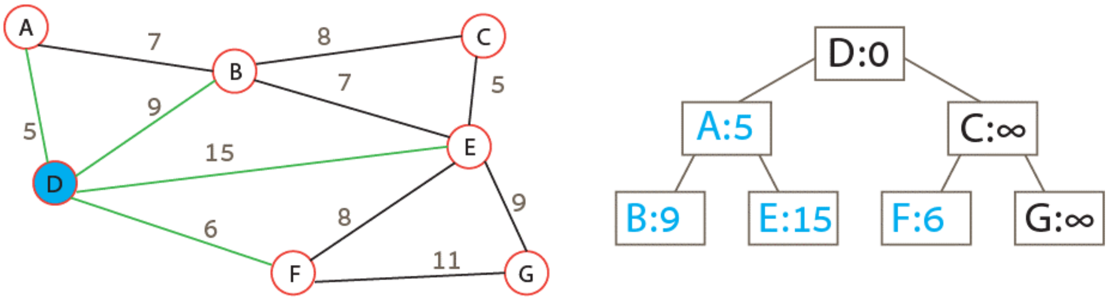
```
MST-PRIM(G,w,r)
  Q ← V
  for each u∈Q
        do key[u]=∞                    // הכנסת ערך מקסימלי לקדקודים
  key[r]←0                             // בחירת הקדקוד הראשון בערך 0
  Π[r]=NIL                             // קביעת המקור של בריך
  while Q≠∅
        u←EXTRACT-MIN(Q)               // הוצאת המינימום מהתור
        for each v∈Adj[u]              // בדיקת השכנים הנוכחיים
           if v∈Q and w(u,v)<key[v]    // הכנסת הערך המנומך יותר לשכן
              then Π[v]←u              // קביעת המקור לקדקוד בעל הערך ביניים הנמוך
                 key[v]←w(u,v)         // קביעת המשקל עבור הצלע המתאימה
```

נבדוק את האלגוריתם על אותו גרף שראינו קודם:

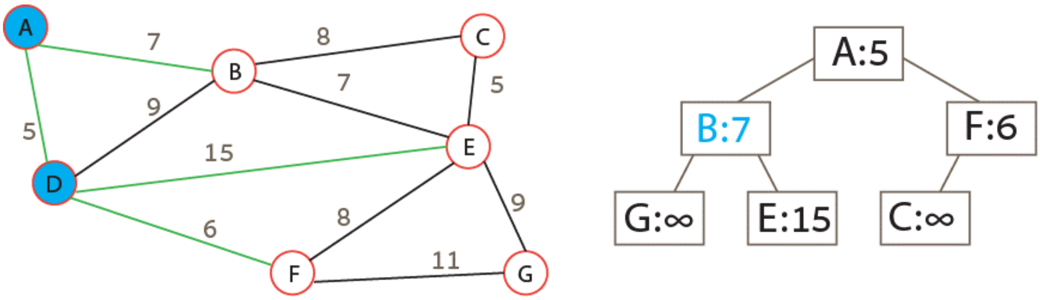




מתחילים באותו גרף, כאשר בצד אנחנו מכינים את התור קדימויות. כאן מימשנו אותו באמצעות ערימה. מכניסים בכל הערכים את הערך אינסוף, ובוחרים קדקוד אקראי. התחלתי כאן מD, רק בשביל שנוכל לראות שנקודת ההתחלה לא משנה כלום.

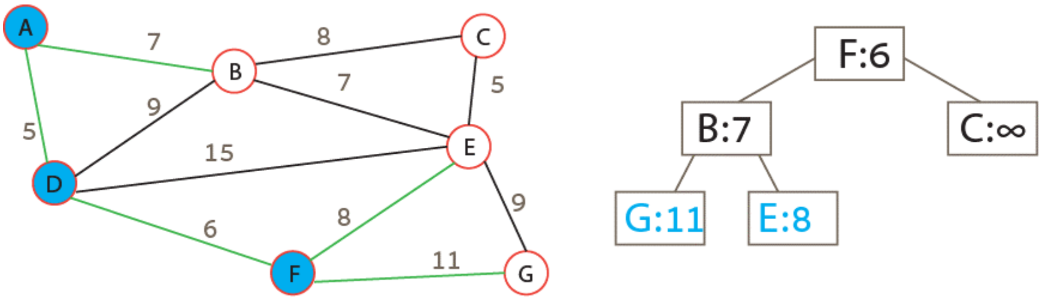


את הערך D סימנו ב-0, מאחר והוא נקודת המוצא. את כל הצלעות היוצאות ממנו, סימנו עם הכיוון אליו אנחנו הולכים, ועדכנו את אותם ערכים בקדקודים בהם פגענו. בעת אנחנו מוציאים את D מהתור, עושים heapify (וידוי קטן: אני חלוד קצת בערימות, יש מצב שזה לא מושלם. אבל הרעיון יובן בכל אופן), בודקים מה נמצא בראש הערימה וממשיכים הלאה:

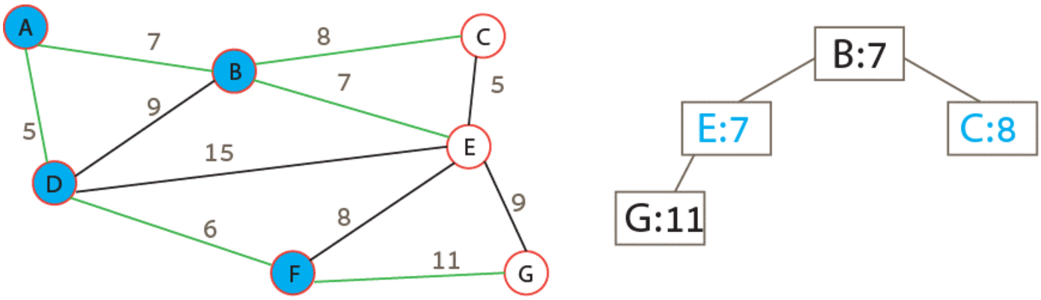


A עלה לראש הערימה, ולכן אנחנו בודקים אותו. לפתע אנחנו מגלים (אמוג'י נדהם) שבקדקוד B אנחנו יכולים לעדכן ערך נמוך יותר מהקיים בו! אנחנו משנים את הערך שלו, ובהתאמה גם משנים את החץ המוביל אליו, להיות החץ היוצא מ-A.

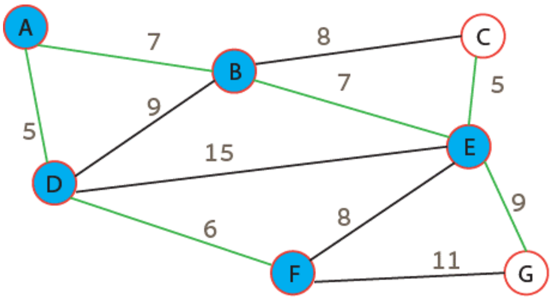
כנ"ל heapify, וממשיכים הלאה:

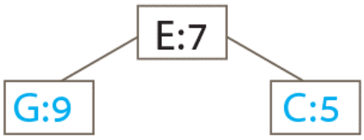


בודקים בעת את קדקוד F. ישר אנחנו רואים, שהערך של E צונח באופן דרסטי מ15 ל-8, ומעדכנים את החץ המתאים. כמו כן, אנחנו נתקלים בקדקוד חדש G. מוציאים את F, מערבבים וממשיכים:



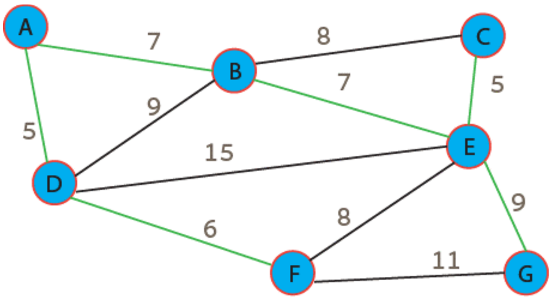
בודקים כעת את הקדקוד B, שבאורח פלא מעדכן שוב את E שיורד להיות בערך 7. בנוסף, נתקלים בקדקוד C, ועכשיו יש לנו ערכים אמיתיים בכל הקדקודים. מערבבים שוב את הקלפים:



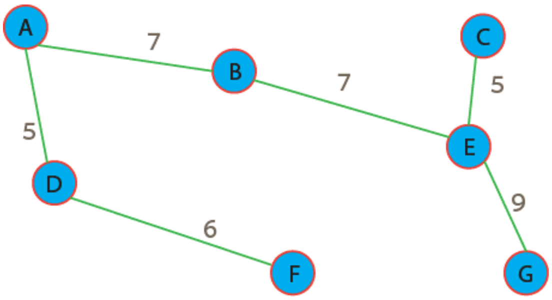


הקדקוד E מעדכן לנו פה גם את C וגם את G, ומוריד להם את הערך.

תכלס עכשיו, אנחנו כבר סיימנו. יש לנו 6 צלעות. גם אם נבדוק את שני הקדקודים שנשארו לנו, שלא בדקנו באופן פרטני, לא נשנה את התוצאות, אבל נבדוק בכל אופן על הסיבוי הקלוש שיקרה משהו:



אפשר לעשות פרצוף מופתע. אין שינוי. מה שנשאר עכשיו הוא לנקות את כל הצלעות המיותרות ולקבל את העץ הבא:



שהוא בדיוק אותו העץ שקיבלנו גם עם קרוסקל.

עכשיו נבדוק את זמן הריצה, שגם כאן מתחלק לשלושה חלקים:

1. קביעת הערכים הראשוניים – $O(V)$
2. הוצאת מינימום וסידור הערימה – $O(\log V)$
3. בדיקת שכנים וחזרה E פעמים – $O(E\log V)$

סה"כ סדר גודל $O(E\log V)$

בעת נשאלת השאלה המתבקשת – מה יותר טוב? קרוסקל $O(E\log E)$ או פרים $O(E\log V)$.

אמנם, אנחנו שואפים להגיע לעץ שצלעותיו יהיו תמיד פחותים יותר מסך קדקודיו, ולכן אנחנו נוטים לומר שקרוסקל בסופו של דבר יעיל יותר. אבל, זה יהיה רק במקרה הטוב או קרוב לו, שמספר הצלעות הוא נמוך יחסית. במקרים אחרים, בהם מספר הצלעות יהיה מרובה, אזי דווקא פרים הוא זה שייתן סדר גודל מתאים יותר ביעילותו.

שאלות הרחבה

על גרף לא מכוון, קשיר ומשוקלל G=(V,E) הורץ אלגוריתם למציאת עפ"מ והוחזר העץ T. בעת הסירו מהגרף קשת (u,v) והתקבל גרף חדש 'G. עלינו למצוא עפ"מ חדש 'T, ביעילות מירבית.

על פי דברי אבירם זילברמן, שאלה זו היתה באחד המבחנים בשנים קודמות. חלק גדול מהתלמידים ענו על שאלה זאת עם התשובה הקלאסית "נריץ על העץ החדש את האלגורית של פרים". מה שהוביל אותם לקבל 0, ואותנו לשיעור החשוב – אל תענה את התשובה הראשונה שעולה לך לראש!

אמנם האלגוריתם של פרים הוכח כיעיל יחסית לקרוסקל, אבל זה לא אומר שום דבר בקשר למקרה הזה. מאחר וכבר יש לנו עץ שאנחנו יודעים שהוא מינימלי, הורדה של צלע מהגרף המקורי, אולי תשנה את הגרף במקצת, אך בטוח לא באופן משמעותי, כי יש לנו קדקודים שלא קרובים לאיזור ההשמטה שבטוח לא יושפעו מזה. יותר מזה – אם הצלע שהורד לא היה חלק מהעפ"מ T המקורי, לא יהיה בו שום שינוי.

אם כן – איך נתקדם פה?

נעבוד באופן הבא – נבחן את הצלע שהוצאה (u,v) – עבורה יש לנו 2 אפשרויות:

1. היא לא חלק מהעץ T המקורי – שום דבר רע לא קרה ואפשר להמשיך הלאה בסבבה.
2. היא חלק מהעץ T המקורי.
   במקרה כזה, אנחנו יוצאים מנקודת הנחה שהעץ T התחלק לשני עצים חדשים. בנוסף אנחנו יכולים להבין שכל קדקוד מהצלע שייך לחלק עץ אחר.
   מה שנעשה – נריץ על כל חלק של העץ BFS, כאשר בT_1 נצבע את הקדקודים שאנחנו מגיעים אליהם בכחול, וב T_2 נצבע את הקדקודים באדום.
   לאחר שעברנו על שני העצים, נעבור על כל הצלעות – הצלעות יחולקו ל-3 אפשרויות – צלע שכולה כחולה, צלע שכולה אדומה – בשני המקרים האלו, אין לנו מה לעשות ואלו צלעות אינטגרליות של העץ T, וכל הצלעות שנותרו בעצם יחברו בין שני החלקים של העץ וכל קצה יהיה צבוע בצבע אחר. נבחר מתוך הצלעות האלו את הצלע המינימלית שתחבר לנו את שני חלקי העץ, והצלחנו.

כמובן, שהתגובה הראויה לשאלה בזאת היא "למה נראה לכם שאני עכשיו אחשוב במבחן על BFS?" והתשובה היא "שיהיה לנו בהצלחה!".

נתון גרף G(V,E) לא מכוון, ממושקל וקשיר.
צ"ל: עץ פורש *מקסימלי*

פתרון: נציע גרסה שונה של קרוסקל-

1. נמיין את הקשתות ב-G בסדר יורד לפי משקלי הקשתות.

2. בכל איטרציה, במקום לבחור את הקשת בעלת המשקל הנמוך ביותר (כמו באלגוריתם המקורי), נבחר את הקשת בעלת המשקל הגבוה יותר.

הפתרון לכאורה, נדמה להיות די אינטואיטיבי, במנהגם של אלגוריתמים חמדניים, אך ברגע שאנחנו נצטרך להוכיח את נכונות האלגוריתם אנחנו נתחיל להסתבך. אנחנו לא יכולים להסתמך באופן פשוט על כך ש"קרוסקל עובד, אז גם זה יעבוד", כי למעשה השינויים שעשינו באלגוריתם הם רחבים מכדי לבטל אותם.

מבחינת ההוכחה, כמובן שמאחר ואנחנו למעשה משתמשים בשיקול חמדני, נוכל להשתמש בדרך ההוכחה המתאימה, אך אנחנו רוצים להוכיח על בסיס הקרוסקל, אבל בצורה נכונה, וללא נפנופי ידיים.

הוכחה: נניח בלי הנחת הכלליות, שכל המשקלים הינם חיוביים (מטעמי נוחות גרידא. אם באמת יש משקלים שליליים, נשנה את הכל בהתאמה, והופ! יש לנו חיוביים. די לשאול שאלות!). כעת, נבנה גרף חדש ונבנה אותו G^- שזהה לחלוטין מבחינת המבנה שלו לגרף G. ההבדל היחיד בין הגרפים, הוא שבעת נגדיר את המשקלים להיות בדיוק הפוכים מאשר המשקל המקורי שלהם. כל המשקלים יהיו בעת שליליים.

עכשיו, נריץ את אלגוריתם קרוסקל על הגרף G^-.

טענה: קבוצת הקשתות שתיתן ריצת האלגוריתם על הגרף G^-, תהיה העץ פורש מקסימום של גרף G אם"ם, קבוצת הקשתות היא העץ הפורש מינימלי.

נוכיח את הטענה הזאת משני הכיוונים.

הוכחה: נניח בשלילה, כי קבוצת הקשתות המתקבלת ב-G^- היא בעלת משקל מינימלי, ונניח כמו כן, שקיימת קבוצת קשתות בG שהמשקל הכולל שלה הוא מקסימלי. אם כן, ננסה להפוך את הסימנים של משקלי הקשתות בקבוצה, ואזי נקבל קבוצת קשתות חדשה, שלמעשׂ משקלה קטן ממשקל קבוצת הקשתות שנמצאו על יד אלגוריתם קרוסקל. **סתירה.**

מצד שני – נתונה קבוצת קשרים בG שמשקלה מקסימלי. נניח בשלילה שקיימת קבוצת קשתות שונה, אשר בגרף G^-, נותנת עץ במשקל מינימלי קטן יותר מהמקביל שלו. ולכן קבוצת הקשתות עם המשקל המקסימלי בG היא קבוצת הקשתות המינמלית ב-G^- שאותה בחר אלגוריתם קרוסקל. כלומר מצאנו קבוצת קשתות בעלת משקל קטן יותר ממה שקרוסקל מצא, וזה הרי לא ייתכן. **סתירה.**

נתון גרף G(V,E) לא מכוון, קשיר וממושקל
צ"ל תת גרף פורש מינימלית

שוב, דבר שראינו כבר קודם – לא להיחפז ולענות תשובות. אוטומטית אנחנו יכולים לטעון, שתת גרף מינימלי בעצם שקול לעץ פורש מינימלי. אבל, עלינו לבחון את כל מרחב התשובות האפשרי. הטעות האינטואיטיבית שלנו היא, שאנחנו ישר חשבנו על כך שכל המשקלים חיוביים, במקרה כזה, אכן תת הגרף המינימלי יהיה העץ פורש מינימלי. אך במקרה בו חלק מהמקלים הם בעלי ערך שלילי, יכול להיות שהעץ לא יהיה הערך המינימלי של תתי הגרף האפשריים. למה? כי נוכל להוסיף קשתות בעלי משקל שלילי שייצרו גם עיגולים – אך אין שום חוק מונע מבעדנו לעשות באופן הזה.

אם כן, האלגוריתם יהיה כדלקמן:

1. ראשית, נבצע סריקת קרוסקל על הגרף.

2. לאחר מכן, נוסיף לתת הגרף את כל הקשתות בעלי המשקל השלילי שנשארו מחוץ לעץ.

באופן כזה נוכל להבטיח כי תת הגרף יהיה פורש (בעזרת הקרוסקל), וכן שיהיה מינימלי (על ידי שימוש בבכל הקשתות השליליות).

הוכחה:

1. בוודאי שתת הגרף הוא פורש.
2. צ"ל שתת הגרף הוא גם מינימלי.

נניח בשלילה כי קיים תת גרף G^- שמשקלו קטן יותר ממשקלו. הרי שיש קשת הנמצאת ב-G^- שלא נמצאת בG. משקל הקשת בוודאי לא שלילי – מאחר שחלק מהאלגוריתם הוגדר לקחת את כל הקשתות בעלי המשקל השלילי – הקשת 'e בוודאי בעלת משקל חיובי.

נתנק את הקשת 'e מהגרף הפורש 'G, אם קיבלנו תת גרף פורש- סתירה.

אחרת – יש תת גרף שאינו פורש המחולק לשני חלקי קשירות – על החתך שנוצר. אם 'e היא קשת יחידה על החתך, אזי היא חייבת להיות בכל עץ פורש של הגרף, ובוודאי גם על העץ הפורש, ולכן שייכת גם ל'G וזה סתירה.

אם יש קשת נוספת על החתך ''e, אם 'e''<e לא יכול להיות נכון. אם זה היה נכון, בוודאי כבר היה נכנס תת הגרף המקורי, ואז זה סתירה למינימליות של 'G.

אחרת – אם 'e''>e, אזי 'e חייבת להיות בעץ הפורש מינימום של G, והרי סתירה.

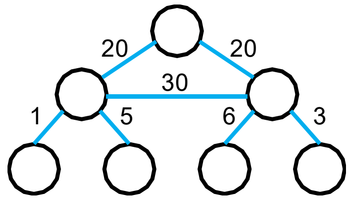
נתון גרף לא מכוון ומשוקלל G(V,E)
צ"ל עץ בעל משקל מינימלי המכיל 1-|V| קדקודים.
נתון האלגוריתם הבא:
1. מצא עפ"מ של G.
2. מצא עלה שהסרתו גורמת להקטנת המשקל של העץ באופן מקסימלי.
3. החזר את העץ ללא העלה הנ"ל.
הוכח או הפרך את האלגוריתם.

אנחנו רוצים לבדוק אפשרות של לקחת גרף, להוריד לו קדקוד אחד, ולמצוא את העפ"מ באותו רגע. כמובן שאנחנו יכולים לקחת כל קדקוד ולבדוק בכל רגע האם יש שינוי, ואת זה אנחנו בודקים.

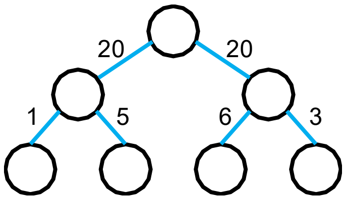
האלגוריתם המוצע, טען שאם ניקח את אותו עץ שהוחלט כמינימלי, ונוריד ממנו את הערך הגבוה ביותר הקיים בו, נקים את הדרוש. האם זה נכון?

לא.

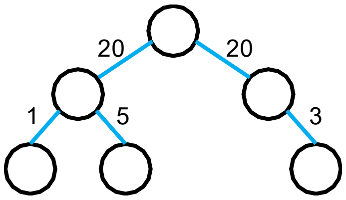
נפריך על ידי הדוגמה הבאה –



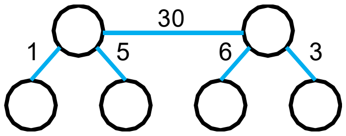
אם ניקח את העץ הפורש המינימלי, נישאר עם העץ הבא:



המשקל הכללי של העץ כרגע הוא 55. ואם עכשיו נוריד את העלה בעל המשקל הגדול ביותר נגיע ל49.



אבל למעשה, התשובה הנכונה יותר תהיה להוריד את הקדקוד ולהוריד את הערך הכללי ב-10:



שללנו את הטענה!

מסלולים קצרים ביותר

בעיית המסלולים הקצרים ביותר, הינה בעיה שכשמה כן היא, מחפשת את המסלולים הקצרים ביותר. היישומים של הבעיה נוגעים לכל אפליקציות הניווט שמשתמשים בהם היום[8], אך גם למסחר בבורסה, שילוח בינלאומי (UPS ודומיהן שצריכים להעביר את כל החבילות מאמזון עד אלינו בדרך המהירה ביותר) ועוד.

כמובן שבכל המקרים האלה, לקחת את כל הדרכים האפשריות ולבדוק את כל הקומבינציות האפשריות יהיה לא יעיל בעליל, 2^n בלי למצמץ בכלל. אנחנו כמובן צריכים למצוא את הדרך היעילה ביותר, עד כמה שאפשר, ולשמור על העסק שיישאר ברמה פולינומיאלית.

גרסאות שונות עבור הבעיה

דיברנו על "מסלולים קצרים ביותר", אך לא אמרנו מהיכן לאיפה. מסתבר שיש ארבע גרסאות שונות לבעיה, שכל אחת מהן נפתרת בשיטה מעט שונה. כמובן, שהכל מתבסס פחות או יותר על אותו פתרון, אבל היישום בפועל משתנה בהתאם לבעיה.

<u>מסלולים קצרים ביותר ממקור יחיד –</u> זה מקרה הבסיס איתו מתחילים. האלגוריתמים שנציג, כל אחד עם מגבלותיו ויכולתיו, פותר את הדרך למצוא את הדרך הקצרה ביותר מנקודה נתונה (שתיקרא בדרך כלל s), לכל שאר הקדקודים. כמו שכבר אמרנו, אנחנו מחפשים דרך קצת יותר מתוחכמת מפשוט לבדוק את כל האופציות.

<u>מסלולים קצרים ביותר מכל הקדקודים למקור יחיד –</u> בעיה זו אינה פשוט נפתרת על ידי בדיקה של כל קדקוד בנפרד, כמו שהיינו עלולים לחשוב, ונרחיב עליה בשאלות ההרחבה.

<u>מסלול קצר ביותר בין שני קדקודים –</u> אם נדרש למצוא את המסלול הקצר ביותר בין u ל-v, כל שיהיה עלינו לעשות, הוא להגדיר את u בנקודת ההתחלה, ולהריץ עליו את האלגוריתם הידוע, ופשוט לבחור את המסלול המתאים. כיום לא ידוע על שום פיתרון אחר שפשוט מוצא את המסלול המבוקש מאיתנו, שזמן הריצה שלו יהיה משמעותית נמוך יותר מהפתרון המלא.

<u>מסלולים קצרים ביותר בין כל זוגות הקדקודים –</u> בבעיה זו, עלינו למצוא את סך **כל** המסלולים עבור כל זוג קדקודים הקיימים בגרף. כמובן שגם פה השיטה לא תהיה להריץ את האלגוריתם הרגיל שוב ושוב, עד שנמצא את כל המסלולים האפשריים. עלינו לזכור, שמירב הבעיות פה מתייחסות לגרפים שהם מכוונים, כך שאם מצאנו מסלול ספציפי בין שתי נקודות, מצאנו רק כיוון אחד, ואין לנו שום הבטחה שהכיוון השני יהיה באותו מסלול אם בכלל.

הגדרות כלליות עבור הפרק

הגרפים שיוצגו לנו, יהיו תחת ההגדרה הרגילה של G(V,E) עם פונקציות משקל $w:E \rightarrow R$, שהמשקל כמובן יהיה קנה המידה בו נשתמש לבעיות השונות (מרחק, זמן נסיעה וכו').

[8] תיאור נחמד וקליל על השימוש באפליקציות ניווט – https://www.facebook.com/InnovationAuthority/photos/a.578866995492520/2008101869235685/?type=3&theater

המסלול אותו אנחנו עוברים יוגדר ב $P=<v_0,v_1..v_n>$ שהוא כמובן וקטור המורכב מהקשתות האפשריות.

המשקל הכללי שנקבל בסוף יוגדר ב $w(p)=\sum_{i=1}^{k} w(v_{i-1}-v_i)$.

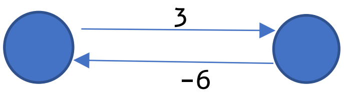
המסלול הקצר ביותר יוגדר בסוף האלגוריתם באופן הבא:

$$\delta(u,v)=\begin{cases}\min\{w(p):u\rightarrow\rightarrow^{[9]} v\} & \text{אם יש מסלול מ-u ל-v} \\ \infty & \text{אחרת}\end{cases}$$

עץ המסלולים הקצרים ביותר שנמצא בסוף כל אלגוריתם, שהוא אותו עץ המושרש ב-s, הוא תת-גרף מכוון G’(V’,E’), כאשר V’⊆V, E’⊆E, וכן יהיה חייב למלא את שלושת התנאים הבאים:

1. V’ היא קבוצת כל הקדקודים ב-G שניתן להגיע אליהם מ-s. (המצב האופטימלי הוא להגיע לכל הקדקודים, אבל יכול להיות שהגרף המקורי G מכיל מספר קדקודים שלא ניתן להגיע אליהם)
2. G’ יוצר עץ מושרש ששורשו s.
3. עבור כל v∈V’, המסלול הפשוט היחיד מ-s אליו בגרף G’ הוא המסלול הקצר ביותר אליו גם בגרף המקורי G. (זה נשמע טריוויאלי, אבל נוכיח את כל הדרוש בשביל להפוך את זה לרשמי).

הערה: בחלק מהגרפים עלולים להגיע לנו גם צלעות בעלי משקל שלילי, נתקלנו בדומים כאלה בעבר, אבל כאן כאשר אנחנו רואים צלע בזאת היא עלולה להוביל אותנו למצב בו יהיה לנו מעגל שלילי. מקרה כזה הוא קצת בעייתי, מאחר ובעצם הוא "הורס" לנו את החישוב ללפחות שני קדקודים שנמצאים באותו מעגל. אם למשל נתון לנו חלק הגרף הבא:



כל סיבוב שנעבור במסלול הזה, יוריד לנו את התוצאה הסופית ב-3. נעבור עליו אינסוף פעמים, והתוצאה תהיה ∞-. כך גם כל חלק מהמשך המסלול שיוצא מאחד משני הקדקודים האלה, ירד גם הוא ל∞-. אמנם זה נשמע מגניב להגיע ממקום למקום בזמן שהוא מינוס אינסופי (וזה גם כח-על לא רע בכלל!), אבל זה לא באמת ישים כשאנחנו מחפשים דרך ממקום למקום.

מבנה הפתרונות המוצעים

אנחנו נדבר על שלושה אלגוריתמים שונים, שעובדים כל אחד בדרך טיפה שונה. נציג את ההבדלים ביניהם בטבלה הבאה:

שם האלגוריתם	מבוסס על פתרון	משימה	אילוץ	סדר גודל
דייקסטרה	חמדני	ממקור יחיד לכל היעדים	משקלות לא שליליים	O(E+LogV)
בלמן-פורד	חמדני	ממקור יחיד לכל היעדים	משקלות כלשהם (גם שליליים)	O(VE)
פלויד-וורשאל	דינאמי	מכל המקורות לכל היעדים	משקלות כלשהם	$O(V^3)$

[9] הסימון הרשמי הוא חץ מסולסל, שמבטא מעבר של מסלול ולא רק חץ ישיר בין שני קדקודים. מאחר ולא מצאתי אפשרות כזו בוורד, אז בכל מקום בו יש שני חיצים צמודים אחד לשני, הכוונה היא למסלול.

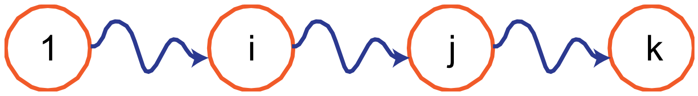
המסלול להבנת האלגוריתמים המוצעים, עובר במספר לֶמוֹת. גם חלק גדול מהלמות מרגישות מאוד אינטואיטיביות וטריוויאליות, אבל נוכיח אותן עד כמה שאפשר, על מנת שלא נצטרך לעשות קפיצות לוגיות, ולהכריז שהאלגוריתם "פשוט עובד"[10]. הלמות ימוספרו 25.x כל אחת על פי מקומה, על שם הפרק בספר של האוניברסיטה הפתוחה.

למות להוכחה -חלק א'

25.1 – תת מסלול של מסלול-קצר-ביותר הוא מסלול קצר ביותר.

בהינתן גרף מכוון ומשוקלל G(V,E) עם פונקציית משוקלל w:E→R, ויהי המסלול הקצר ביותר $p=<v_1,v_2...v_k>$, המסלול הקצר ביותר בין v_1 ל-v_k. אזי עבור כל i, j המקיימים $1\leq i\leq j\leq k$, יהי $p_{ij}=<p_i..p_j>$ המסלול הקצר ביותר בין שתי הנקודות v_i, v_j.

<u>הוכחה:</u> ברור שזה נראה מאוד טריוויאלי, אבל גם כשלמדנו את הלמה הזאת במבנה נתונים ב' עבור BFS נדרשנו להוכיח אותה ולא להסתפק רק במה שנראה לנו במבט ראשון. גם פה אנחנו נשתמש באותה צורת הוכחה שהשתמשנו אז, על מנת להחיל את הקביעה הזאת גם כאן, בגרף שהוא מכוון ומשוקלל.



נסתכל על הגרף הבא שמתאר לנו את המסלול הכללי p החל מ-1 ועד ל-k, המקיים לנו מסלול קצר ביותר, או באופן רשמי יותר δ(1,k). על פי הגדרת הטענה, נוכל לראות שבעצם אנחנו יכולים לחלק את המסלול לשלושה חלקים – $p = <v_1,v_i> + <v_i,v_j> + <v_j,v_k>$. נניח בעת בשלילה, שלתת המסלול $p_{ij}<p_i..p_b>$ קיים מסלול, שנקרא לו p'_{ij} שהוא קצר יותר מהנתון. אזי, המסלול p אינו קצר ביותר בין שתי הנקודות! כי אם ניקח את $p' = <v_1,v_i> + p'_{ij} + <v_j,v_k>$ יתקיים לנו כי $p < p'$. ויש פה סתירה לטענה, ומש"ל.

25.2 המשפט הזה אינו למה, אלא מסקנה מהלמה הקודמת.

יהי גרף G(V,E) מכוון, ממושקל וכו'. ונתון לנו מסלול קצר ביותר בין s ל-v, הניתן לפירוק באופן הבא: s→→u→v (הכוונה היא, שאנחנו יודעים שבאותו מסלול, הקדקוד הלפני-אחרון הוא u. אזי אנחנו יכולים לדעת בוודאות, כי המסלול בין s→→u הוא המסלול הקצר ביותר.

<u>הוכחה:</u> כן. גם את זה צריך להוכיח. אמנם הוכחנו ב25.1 את התבונה הכללית הזאת, אבל פה אנחנו רוצים להוכיח שהטענה שאנחנו מביאים עומדת בתנאי הלמה הקודמת. בשביל להוכיח זאת, אנחנו נשתמש בפירוק המתמטי הבא:

$$\delta(s, v) = w(p) \rightarrow w(p') + w(u,v) \rightarrow \delta(s,u) + w(u,v)$$

מה שאנחנו אומרים כאן באופן פשוט הוא הדבר הבא: אנחנו יודעים שהמסלול הקצר ביותר הוא p. אנחנו גם יודעים שהמסלול הקצר ביותר הנ"ל, הוא בעצם מורכב מהפירוק של תת-מסלול כלשהו, באיחוד עם הקשת בין u ל-v. עכשיו אנחנו משתמשים בלמה, וטוענים שתת המסלול p' הוא בוודאי תת מסלול קצר ביותר, וזה מה שרצינו להוכיח. מש"ל.

---

[10] על פי דברי פרופ' קרנר, אין צורך לדעת את הוכחת כל הלמות עצמן למבחן, אך ראוי להכיר אותן כשלומדים את האלגוריתם.

25.3 יהי גרף G(V,E) מכוון וממושקל וכו'. אזי עבור כל קשת (u,v) ∈ E, מתקיים $\delta(s,v)\leq\delta(s,u)+w(u,v)$.

<u>הוכחה:</u> זה לא מה שהוכחנו ב25.2? כן. עכשיו נגדיר זאת כתכונה עקרונית:

משקל המסלול הקצר ביותר מ-s ל-v, הוא בוודאי אינו גבוה יותר מכל מסלול אחר שעובר בין שני הקדקודים (אמנם יכול להיות שהוא שווה למסלול אחר, אבל זה לא מעלה ולא מוריד). בפרט, המשקל של אותו מסלול, גם אינו עולה על כל מסלול אחר העובר עד הקדקוד הסמוך לו, ואז עובר אליו (זה כמובן, על פי מה שאמרנו מקודם). מש"ל

טכניקת ההקלה Relaxation

כל האלגוריתמים שנביא בהמשך משתמשים בטכניקה שנקראת "הקלה". אנחנו מחפשים בידוע את הדלתא δ המייצגת את המשקל הנמוך ביותר האפשרי בין שני קדקודים, משקל זה הוא כמובן קבוע מראש ואנחנו מחפשים לחשוף אותו. על מנת להגיע לדלתא האופטימלית, אנחנו נשמור עבור כל קדקוד ערך d שייצג את המרחק המינימלי אותו אנחנו יכולים לחשב עד לאותו רגע, בשאיפה שבסופו של דבר הערך הנתון לנו ב-d, אכן יהיה ה-δ. בנוסף, אנחנו נשמור גם את ערך "הקודם" π בו נשמור את הקדקוד המינימלי דרכו הגענו לקדקוד הנוכחי.

על מנת שהרעיון יעבוד בצורה נוחה, בתחילת כל אלגוריתם, אנחנו נאתחל את השדות להיות אופטימליים עבורנו, ונשים בהם ערכים שבוודאי ישתנו בעת המעבר על הגרף. פונקציית האתחול מוגדרת כך:

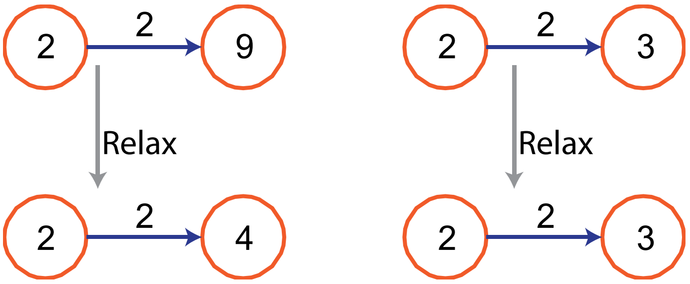
```
Initialize-Single-Source(G,s)        // הפונקציה מקבלת את הגרף ואת נקודת ההתחלה
  for each vertex v ∈ V[G]           // עבור כל הקדקודים בגרף:
      d[v] = ∞                       // אתחול מרחק ההגעה מקדקוד ההתחלה לאינסוף
      π= NIL                         // אתחול הקודם לערך ריק (מאחר ואנחנו לא יודעים מי מגיע אליו)
  d[s] = 0                           // אתחול המרחק הראשוני של נקודת ההחלה ל-0
```

ניתן לראות, כי ערך הקודם של נקודת ההתחלה s תישאר ריקה, וזאת באמת מאחר ואין לו קדקוד קודם. תהליך ההקלה עצמו, בודק עבור כל שני קדקודים הנשלחים לפונקציה, האם הוספת המסלול ביניהם תשפר לנו את המרחק או לא. כמובן, שבמעבר הראשון שמגיעים לכל קדקוד, אנחנו כבר נתחיל לעבוד ולהוריד את האינסוף לאיזשהו ערך קבוע, אבל אנחנו עלולים לחזור לכל קדקוד מספר פעמים ולא תמיד תנאי ההקלה יתקיים, ולכן גם לא תמיד נשפר.

האלגוריתם מוצג באופן הבא:

```
Relax(u,v,w)                         // מקבלים שני קדקודים ואת משקל הקשת ביניהם
  if d[v] > d[u] + w(u,v)            // בדיקה האם המרחק יקטן בעקבות שימוש בקדקוד הקודם
      d[v] = d[u] + w(u,v)           // במידה וכן – 1. נעדכן את המרחק
      π[v]= u                        // 2. נעדכן את הקודם
```

לאחר ביצוע שני האלגוריתמים האלה, כל אחד לפי השימוש שלו במקום שהוא יתפוס באלגוריתמים, אנחנו נקבל בסוף את המרחק המינימלי עבור כל קדקוד, וכן את המסלול שנוכל לעקוב אחריו בעזרת ה-π.



ניתן לראות דוגמא של פעולת ההקלה עבור שתי אפשרויות שונות. נניח כי שני הקדקודים כבר מאותחלים בערך מסוים. כעת, כאשר אנחנו בודקים בגרף השמאלי, אנחנו יכולים לראות שהשימוש בקדקוד u עם המשקל 2 יניב לנו תוצאה הרבה יותר אופטימלית עבור v ולכן נעדכן אותו. מה שאין כן בגרף הימני, בוא הערך הקיים הוא נמוך יותר ומשאירים הכל שיעמוד על מקומו.

למות להוכחה – חלק ב'

המשך הלמות קשורות באופן ישיר לפעולות האתחול וההקלה, כאשר רובן מתייחסות לגרף מכוון משוקלל לאחר אתחול.

25.4

יהי גרף G(V,E) גרף מכוון ומשוקלל. ובתוכו קשת $(u,v) \in E$. אזי, מייד לאחר פעולת ההקלה על אותה הקשת, מתקיים $d[v] \leq d[u] + w(u,v)$.

<u>הוכחה:</u> בפשטות, זה מה שאנחנו עושים בפעולת ההקלה, אך נוכיח זאת באופן פורמלי: אם לפני בדיקת ההקלה, המצב הקיים היה $d[v] > d[u] + w(u,v)$, אז בוודאי שנקיים את התנאי ובעת $d[v] = d[u] + w(u,v)$. אם עוד לפני פעולת ההקלה, התקיים כי $d[v] \leq d[u] + w(u,v)$, אז בכלל לא נכנסו לפעולה והמצב הקודם מקיים את הדרוש (כמובן, שזה כולל את האופציה בה הגענו ל-v ממסלול אחר והוא שווה למסלול הכולל את u).

25.5

יהי גרף G(V,E) מכוון ומשוקלל לאחר פעולת אתחול. אזי עבור כל קדקוד $v \in V$, מתקיים כי $d[v] \geq \delta(s,v)$. ואינוואריאנטה (אי-שינוי) זה מתקיים לאורך כל סדרה של צעדי הקלה בכל אלגוריתם שלא יהיה. יותר מזה, אם ייווצר מצב בו $d[v] = \delta(s,v)$, אזי הערך של d[v] לא ישתנה יותר.

<u>הוכחה:</u> נוכיח עבור כל המצבים האפשריים, כי אכן דבר זה נכון:

עבור התחלת הפעולות – לאחר שאתחלנו את כל המערכת, כל המרחקים בקדקודים שאינם s יאותחלו לאינסוף, ובוודאי שזה יהיה גדול/שווה לתוצאה הסופית – אם לא נוכל להגיע לקדקוד מ-s זה יישאר כך, ואם לא כל דבר אחר קטן מאינסוף. לעניין s – אנחנו מאתחלים אותו ב-0. מה שבכלל הנראה יהיה נכון. הסיבה היחידה שדבר זה עלול להשתנות, זה אם נגלה מעגל שלילי שמוביל בחזרה ל-s. במקרה כזה $\delta(s,s) = -\infty$, שגם הוא ידוע בתור קטן מ-0.

לגבי המשך הצעדים – ניתן להוכיח זאת בשלילה.

נצא מנקודת הנחה שאכן האינווריאנטה נשמרת. אך לאחר שהגענו לקדקוד הראשון ועשינו עליו הקלה הגענו למצב בו $d[v] < \delta(s,v)$. במקרה כזה, הגענו לתוצאה המתמטית הבאה:

$d[u] + w(u,v) = d[v] < \delta(s,v)$

וידוע לנו, כי לאחר פעולת ההקלה מתקיים כי $d[v] \leq \delta(s,u) + w(u,v)$ על פי למה 25.3 שהוכחנו קודם.

אך אם מה שהנחנו עד כה נכון, אז גם $d[u] < \delta(s,u)$ אבל זה לא ייתכן, כי פעולת ההקלה איננה משנה את u אלא רק את v, ואנחנו מדברים על הקדקוד הראשון שאנחנו בוחרים לשנות, ויש כאן סתירה!

השלב הסופי – אנחנו רוצים להוכיח שברגע ש $d[v] = \delta(s,v)$ אזי $d[v]$ לא מתעדכן יותר. זה עתה ראינו שלא יכול להיות מצב בו $d[v] < \delta(s,v)$ תנאי זה חל באופן קבוע, כמו שהוכחנו. ולכן גם $d[v]$ אינו יכול יותר להתעדכן לעולם. מש"ל.

25.6

יהי גרף G(V,E) מכוון, ממושקל ומאותחל. אזי, אם יש קדקוד v שאין שום מסלול המחבר אליו את s, אנו מקבלים אוטומטית כי $d[v] = \delta(s,v)$ והשוויון הזה נשמר באופן קבוע לכל אורך הפעולות שנעשה על הגרף.

<u>הוכחה:</u> על פי מה שהוכחנו בלמה 25.5, תמיד יתקיים כי $d[v] \geq \delta(s,v) = \infty$, ובוודאי ש-$d[v]$ לא יכול להית גדול יותר מזה, ולכן $d[v]=\infty = \delta(s,v)$.

25.7

יהי גרף G(V,E) מכוון, ממושקל ולאחר אתחול. ויהי s→→u→v מסלול קצר ביותר בG. אם לאחר מספר צעדי הקלה הגענו למצב בו $d[v] = \delta(s,v)$, לפני קריאה חדשה עבור פונקציית ההקלה, אזי גם $d[u]=\delta(s,u)$ מתקיים בהתאם לפני הקריאה.

<u>הוכחה:</u> על פי הלמה 25.5, מרגע שהגענו למצב האופטימלי, המצב הזה כבר לא משתנה.

על פי למה 25.4, $d[v] \leq d[u] + w(u,v)$

על פי המסקנה 25.2 $d[v] = \delta(s,u) + w(u,v) \rightarrow \delta(s,v)$

ובחזרה ללמה 25.5, גם $d[u]$ מפסיק להתעדכן מאותו רגע, וזה נשאר באופן קבוע שווה לדלתא.

עצי מסלולים קצרים ביותר

שתי הלמות הבאות באות ללא הוכחה, כי זה ארוך מאוד ומתיש (יאיי!). ומתייחסות לעצי המסלולים הקצרים ביותר. בהגדרות של הפרק דיברנו על כך שבסופו של דבר אנחנו מנסים ליצור עץ שבסיסו יהיה הקדקוד s, ויכיל את כל המסלולים הקצרים ביותר מאותו קדקוד לכל קדקוד אחר בגרף. על מנת לראות שאכן כך הדבר, באות שתי הלמרות (האחרונות) וסוגרות לנו את הפינה.

25.8

יהי גרף G(V,E) מכוון ומשוקלל, בעל קדקוד מקור s, וללא מעגלים בעלי משקל שלילי. אזי, לאחר אתחול הגרף, תת גרף הקודמים G_π יוצר עץ מושרש ב-s. וכל סדרה של צעדי הקלה תשמר תכונה זו באינוואריאנטה.

הסבר: מאחר ואנחנו מתחילים את כל האלגוריתם בקדקוד s, ומאחר וכל הקדקודים מאותחלים ב-π להיות NIL, אז כל צעד הקלה יכול לעדכן את הקדקודים השונים רק בלפי ה-s. לא משנה כמה נתקדם באלגוריתם, בשנחזור אחורה נגיע תמיד לאותו מקום.
מסיבה זאת דרשנו שלא יהיה מעגלים שליליים, שאנחנו יכולים להסתבך בהם ולא להגיע לשום מקום.

25.9

יהי גרף G(V,E) מכוון ומשוקלל, בעל קדקוד מקור s, וללא מעגלים בעלי משקל שלילי. אזי, לאחר אתחול הגרף, וביצוע מספר צעדי הקלה כשהגענו למצב בו לכל קדקוד בגרף $d[v] = \delta(s,v)$, אזי תת הגרף G_π הוא עץ מסלולים קצרים ביותר המושרש ב-*s*.

אין פה עוד הרבה מה להוכיח, אבל נסתכל על 3 התבונות שציינו קודם, ונראה שאכן הכל מתקיים:

1. V' היא קבוצת כל הקדקודים ב-G שניתן להגיע אליהם מ-s – כל קדקוד שיכיל ערך סופי כלשהו ב[d[v, אזי ברור לנו שניתן להגיע איו, כי אכן הגענו אליו. אם יש קדקוד שנשאר ב $d[v] = \infty$, אז כנראה שלא ניתן להגיע אליו ישירות מ-s.
2. G' יוצר עץ מושרש ששורשו s – התחלנו מ-s ולכן בוודאי שהוא יהיה מושרש בו – על פי למה 25.8.
3. עבור כל $v \in V'$, המסלול הפשוט היחיד מ-s אליו בגרף G' הוא המסלול הקצר ביותר אליו גם בגרף המקורי G – על פי כל התבונות שראינו עד עכשיו, המסלול שנבחר יהיה הקצר ביותר ויקיים את כל האמור.

עכשיו אחרי שאנחנו יודעים את כל הלמות, ניתן לעבור לאלגוריתמים עצמם.

האלגוריתם של דייקסטרה

אלגוריתם זה בוסס במאמרו של אדסחר דייקסטרה (זה לא טעות, הוא הולנדי) בשנת 1959 ומהווה בסיס למציאת המסלולים הקצרים ביותר.

כמו שציינו כבר קודם, האלגוריתם עובד אך ורק על גרפים בעלי משקל חיובי, ולכן מרחב השימושים שלו מצומצם יותר, אבל מאחר וזמן הריצה שלו יותר טוב מהאחרים, כדאי להשתמש בו כשאפשר.

הסבר האלגוריתם

יש לנו מספר קבוצות אותם אנחנו מנהלים תוך כדי מימוש האלגוריתם: S – קבוצת הקדקודים בהם כבר קבענו את המסלולים הקצרים יותר. הווה אומר, עבוד כל קדקוד $v \in S$ המשמעות היא שהגענו כבר לתוצאה המקיימת את הנתון הדרוש לנו $d[v] = \delta(s,v)$. בנוסף V יכיל כרגיל את כל הקדקודים בגרף, ונפעיל גם תור קדימויות Q, שיכיל את כל הקדקודים שעוד לא טיפלנו בהם, כלומר V-S. סדר הקדימות יוגדר להיות כמובן, הקדקוד בעל המשקל הנמוך ביותר מבין הנוכחים בתור.

האלגוריתם לוקח את הקדקוד בראש תור הקדימויות, מעביר אותו ל-S, ואז מבצע הקלה עבור כל הקדקודים הקשורים אליו (כל זאת בהנחה המוקדמת שאכן יש לנו טבלת סמיכויות מתאימה).

נראה את המימוש של האלגוריתם בצורה מפורטת:

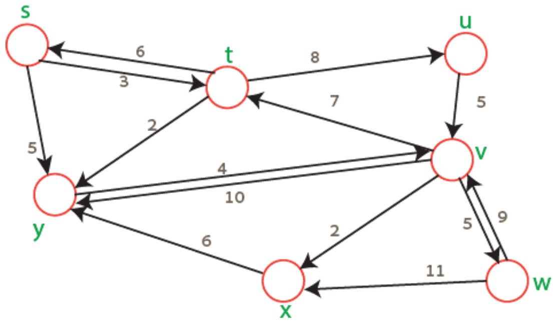
```
Dijakstra (G, w, s)                    // מקבלים רשימה, משקלות, נקודת התחלה
  Initialize-Single-Source(G,s)        // אתחול הגרף
  S = ϕ                                //  אתחול הקדקודים המטופלים בקבוצה ריקה
  Q = V[G]                             // אתחול התור בהכנסת כל קדקודי הגרף
  while Q ≠ ϕ                          // לולאה רצה עד שתור הקדימויות ריק
```

```
u = Extract-Min(Q)              // הוצאת ראש התור
S = S ∪{u}                      // צירוף ראש התור למטופלים
for each vertex v ∈Adj[u]       // מעבר על כל השכנים של הקדקוד
  Relax(u,v,w)                  // ביצוע הקלה
```

שימו לב, שפעולת ההקלה לא נועלת את הקדקוד בפני שינויים. דבר זה נותן לנו את האפשרות לחזור לאותו קדקוד עם פעולת הקלה של שכנים נוספים, מה שייתן לו את האופציה להגיע בסופו של דבר לערך האופטימלי של $d[v] = \delta(s,v)$.

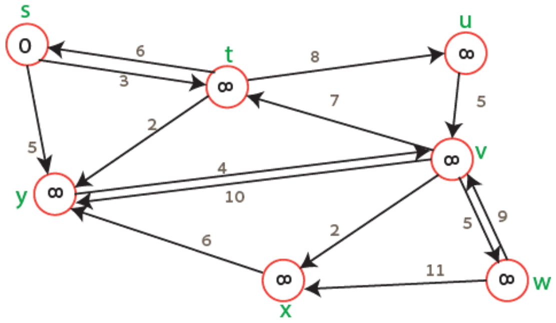
נריץ בעת דוגמה של האלגוריתם ונראה את סדר הפעולות:

ניקח את הגרף שהרצנו עליו את פרים וקרוסקל, ונעשה לו התאמות בשביל שיתאים לשימוש כאן:

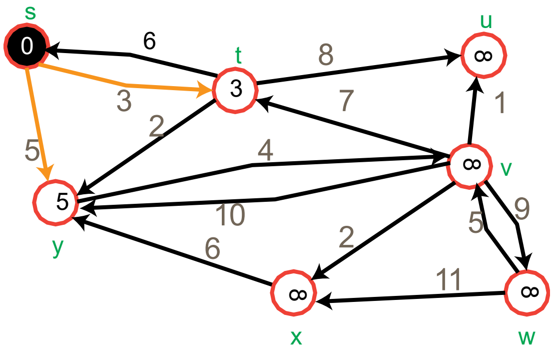


אנחנו לוקחים גרף שהוא מכוון וממושקל, כאשר כל האיברים חיוביים, ויש לנו כמה מעגלים, סתם בשביל הספורט.

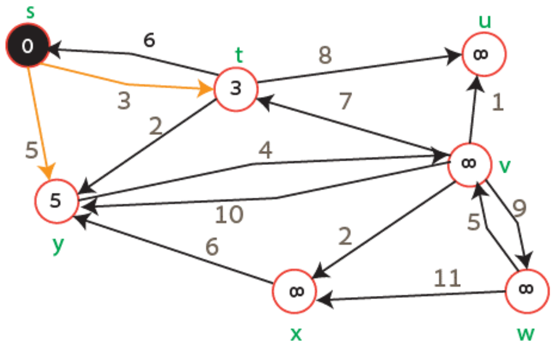
קודם כל – נעשה אתחול לכל הגרף:



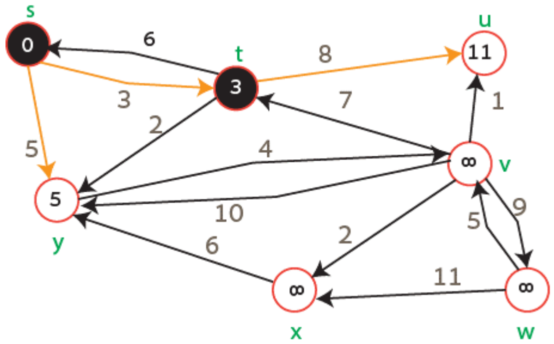
קדקוד s יקבל את הd להיות 0, וכל השאר הם במרחק אינסוף. תור הקדימויות שייבנה, כמובן יהיה מאוד פשוט, מאחר שיש רק ערך אחד שהוא קבוע ואמיתי, וכל השאר די לא רלוונטיים. הבדיקה הראשונה של תור הקדימויות בוודאי תניב שהוא לא ריק ונוכל להתחיל –



לוקחים כמובן את הקדקוד s, צובעים אותו בשחור, ומתחילים לעבור על השכנים שלו ומעדכנים אותם על ידי פעולות הקלה.

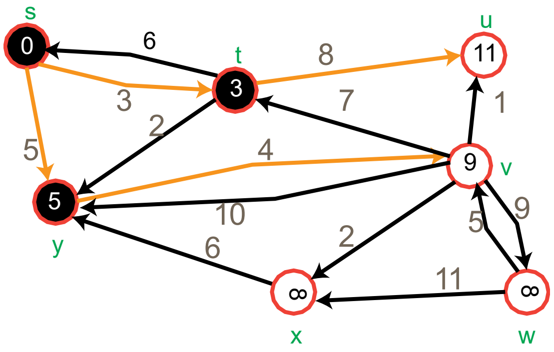


השלב הבא הוא לקחת את הקדקוד t, מאחר ו d[t] = 3.

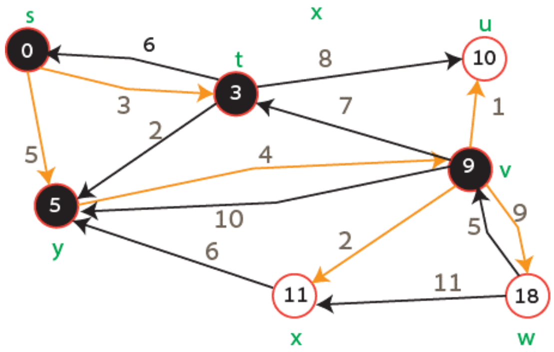


ראשית, אנחנו מעדכנים את u, שמים בו ערך 11. ועכשיו בודקים את השכן השני של t, הלוא הוא y. אממה? s→y = s→t→y = 5. מה יקרה עכשיו? על פי התנאי בהקלה המחפש רק אפשרויות של קטן ממש, הערכים לא ישתנו וכאן הכל יישאר אותו דבר.

הלאה –



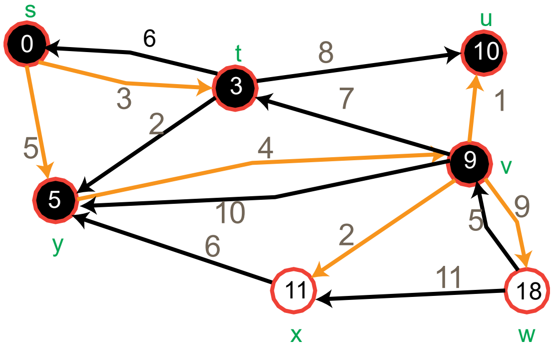
לוקחים את קדקוד y, ומעדכנים את השכן היחיד שלו v. שום דבר לא חשוד פה.



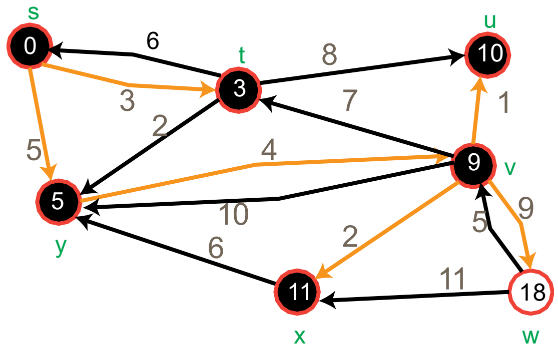
הקדקוד הבא בגודלו הוא v, d[v] = 9. והוא עושה מספר דברים – קודם כל, הוא מעדכן את u. מסתבר שלעבור שלושה קדקודים יוצא מהר יותר מאשר שניים, כולנו נדהמים אבל לא מאבדים עשתונות ומעדכנים את שני הקדקודים הנותרים.

האם סיימנו? לא!

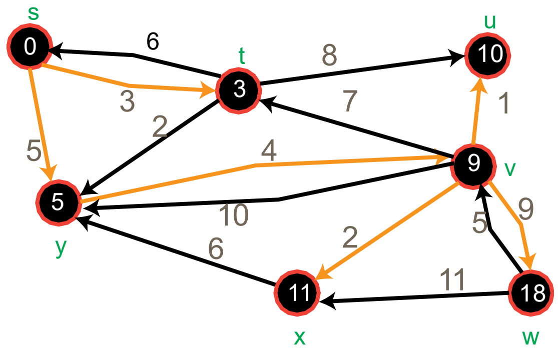
עד שלא נעבור על כל הקדקודים בפעולות הקלה, לא נוכל להיות בטוחים שהגענו לדלתא של כל הקדקודים. ולכן נמשיך הלאה ונבדוק את הקדקוד הבא u.



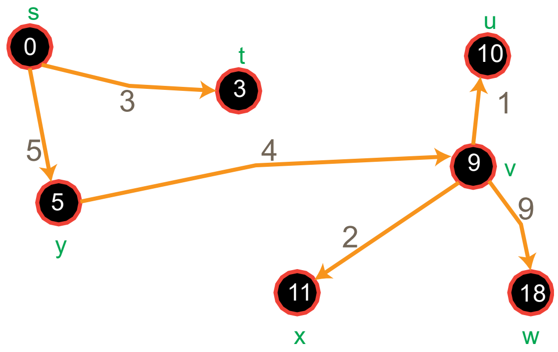
זה היה קצר. שום מסלול לא ממשיך ממנו.



גם קדקוד x לא מעדכן אף אחד, כי 11+6 לא ממש קטן מ-5.



עכשיו סיימנו לעבור על כל הקדקודים, תור הקדימויות ריק, ואנחנו יודעים בוודאות שאנחנו יכולים למצוא את המסלולים הקצרים ביותר מ-s.



סיימנו.

הוכחת נכונות האלגוריתם

טענה (25.10 על פי הספר):

אם מריצים את האלגוריתם של דייקסטרה על גרף מכוון ומשוקלל G(V,E) עם פונקציית משקל אי-שלילית w ומקור s, אזי עם עצירת האלגוריתם, $d[u] = \delta(s,u)$ עבור כל הקדקודים $u \in V$

הוכחה: יוכח בהמשך.

מסקנה (25.11):

אם מריצים את האלגוריתם של דייקסטרה על גרף מכוון משוקלל G(V,E) עם פונקציית משקל אי-שלילית w ומקור s, אזי עם סיום ההרצה, תת-גרף הקודמים G_π הוא עץ מסלולים קצרים ביותר המושרש ב-s.

הוכחה: כנ"ל

ניתוח זמן ריצה

אמרנו שזמן הריצה של דייקסטרה הוא יחסית מהיר. אך עד כמה? זה תלוי בעיקר במימוש של תור הקדימויות. כמובן שאם נחזיק סתם מערך לינארי ונעבור עליו בכל פעם לא נקבל תוצאות טובות (בהנחה שהוא לא ממוין, כל הוצאה תהיה $O(V)$, ומאחר שיש לנו V פעמים שאנחנו עושים את כל הסיבוב הזה, נקבל $O(V^2)$.

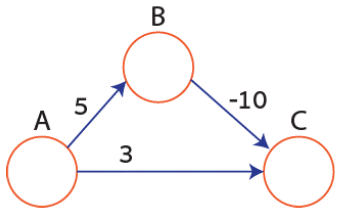
ננסה לעשות את זה באמצעות ערימת-מינימום שנחשבת יעילה ופשוטה, ואז כל הוצאה של קדקוד תהיה $O(LogV)$. וכמובן שגם פה נעשה את זה |V| פעמים. בנוסף, יש לנו את בניית הערימה שתהיה $O(V)$ – יש לזכור שהבניה של כל הערכים שהם אינסוף מלבד s שהוא 0, לא ממש אמור להיות מסובך. עכשיו נותר רק לעשות את פעולת ההקלה. ההקלה אינה רק פעולה פשוטה של השמה שנעשית ב$O(1)$, אלא יש לנו פה פעולה של פעפוע של הערימה שנעשה ב$O(LogV)$, פעולה זו נעשית לכל היותר |E| פעמים – במקרה הגרוע בו בכל קשת נצטרך לעדכן את הקדקודים. אם כן יש לנו שתי אפשרויות שניתן לצמצם תחת $O((E+V)LogV)$, שהוא בדרך כלל $O(ELogV)$ מאחר ובדרך כלל יש יותר קשתות מקדקודים.

ניתן גם לשפר את הזמן הזה על ידי ערימת פיבונאצ'י, שהוא הרבה יותר מסובך וארוך ולא נלמד אותו, אבל אסימפטוטית הוא מביא זמן ריצה יותר טוב.

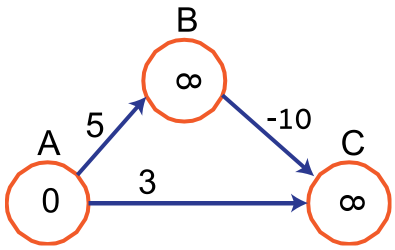
בעיית המשקלים השליליים

הזכרנו את זה קודם, וזה בטח עוד יעלה, אבל למה דייקסטרה לא מסוגל להתמודד עם משקלים שליליים?

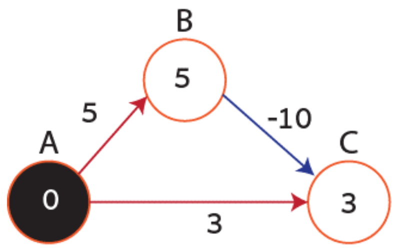
האלגוריתםם של דייקסטרה, נוגע פעם אחת בכל קדקוד. ראינו אלגוריתם, שעוד לפני שאנחנו מתחילים לעשות משהו, אנחנו מעבירים את הקדקוד המטופל מרשימת הממתינים לרשימה של הקדקודים השחורים-המטופלים. כך שאם פתאום נגלה דרך טובה יותר לאחד הקדקודים שכבר טיפלנו בו, לא תהיה לנו שום דרך לעדכן את הקדקוד. נראה דוגמא שתסביר את זה טוב יותר:



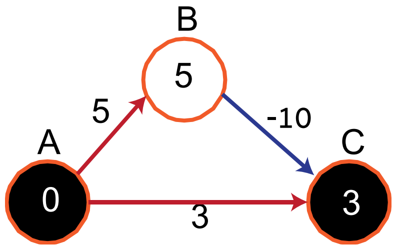
נריץ את האלגוריתם על הגרף הבא באופן הרגיל. ראשית, נאתחל את כל הקדקודים בגרף על פי הדרישה – 0 לקדקוד הראשון, ואינסוף לכל השאר:



עכשיו, אנחנו יכולים להתחיל לעבוד. נבחר את הקדקוד, בעל המסלול הקצר ביותר, ונעדכן את הקדקודים הסמוכים אליו:



בשלב הזה, אם היינו הולכים לקדקוד B, היינו יכולים להמשיך על הקשת היוצאת ממנו, ולסמן בקדקוד C את הערך של המסלול להיות 5-. אבל – אל פי האלגוריתם, אנחנו בוחרים דווקא במסלול שמוביל ל-C, מאחר שהערך שלו נמוך יותר. נעבור אליו –



קדקוד C לא מוציא ממנו שום קשת, ולכאורה היינו עוברים לקדקוד B, אבל מאחר ועברנו כבר שני קדקודים, שהם במקרה n-1 קדקודי בסך הכל, אנחנו עוצרים את האלגוריתם ומחזירים את הגרף איך שהוא עכשיו. כמובן שזה טעות, ואם היינו ממשיכים לבדוק, היינו מגלים את הטעות, אבל בשביל זה לא משלמים לנו.

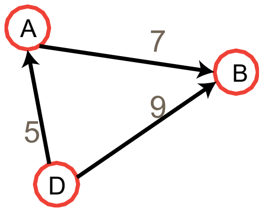
הבדלים בין דייקסטרה לפרים

במעבר ראשון על האלגוריתם של דייקסטרה, ישר קופץ לנו לעין שזה מאוד דומה לאלגוריתם של פרים. שניהם עושים אתחול לאינסוף, שניהם משתמשים בתור קדימויות וגם הריצה שלהם בסך הכל די דומה. אך יש לשם לב למספר הבדלים.

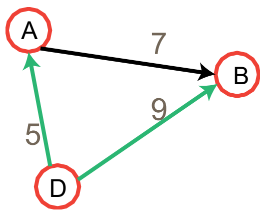
האלגוריתם של פרים מיועד בעיקרו, ואכן מבצע זאת בהצלחה, למציאת עץ פורש מינימלי. הכוונה היא שאנחנו מחפשים פה עץ שיחבר את כל הנקודות ומשקלו הכולל יהיה מינימלי. יש לזה שימושים רבים אם נרצה למצוא דרך לפרוש קווי חשמל באופן שיהיה יעיל ביותר תחת איזור מסוים, אך אין לנו שום ערובה לכך שבין שתי נקודות בלשהן אנחנו נמצא את הדרך הקצרה ביותר.

האלגוריתם של דייקסטרה מחפש את המסלולים הקצרים ביותר בין כל הקדקודים לקדקוד מסוים שיוגדר. השימוש של זה יעיל לניווט, או בניית דרכים מנקודה מסוימת, יכול להיות ואף הגיוני שהגרף שנקבל לא יהיה בעל המשקל המינימלי, אך זו נקודת הסתכלות שלא מעניינת אותנו. אנחנו מחפשים פה את הפתח לאופטימיזציה עבור קדקוד בודד.

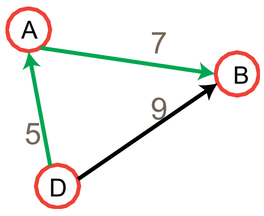
נסתכל לדוגמא, על הגרפונצ'יק הבא:



אם נרצה למצוא את המסלולים הקצרים ביותר של הקדקוד D, אנחנו נקבל את העץ הבא-



אך האם זה העץ הפורש המינימלי בגרף? בוודאי שלא, העפ"מ יהיה דווקא זה –



שוב, המטרה שאנחנו מגדירים לנו בגרף תביא לנו את התוצאה הסופית, ומלבד זה שבסוף יהיה לנו ע בשני המקרים, אף אחד לא מבטיח לנו שמדובר על אותו עץ, או שיהיו בו תכונות דומות.

האלגוריתם של בלמן-פורד

האלגוריתם של בלמן-פורד פותר את הבעיה הכללית יותר של המסלולים הקצרים ביותר. בעוד שדייקסטרה הוא יחסית מהיר, חלות עליו ההגבלות של ערכים לא-שליליים, ואם אנחנו נרצה להכניס באלה למשוואה, נגיד אם נכניס לחישוב גם פרמטר של גובה שיכול להיות שלילי, דייקסטרה עלול לא להוציא לנו תשובה הגיונית, כמו שכבר ראינו.

דבר נוסף שמייחד את בלמן-פורד הוא, שהאלגוריתם בודק ומחזיר תשובה בוליאנית האם יש לנו מעגל שלילי שמחזיר לנו בעצם תשובה לא מדויקת. הוא לא פותר את הבעיה או מציין איפה היא קיימת (למרות שזה משהו שאפשר לבדוק), אלא רק אומר האם זה קיים בגרף הנתון.

הסבר האלגוריתם

בשונה מדייקסטרה שעושה הקלה רק לשכנים של הקדקוד ולוקח בכל פעם את הקדקוד בעל המשקל הנמוך ביותר, בלמן-פורד פשוט רץ על כל קדקוד, ובתוכו נכנס לכל הקדקודים המקושרים אליו בסדר לקסיקוגרפי ומבצע הקלה, באופן של לולאה מקוננת.

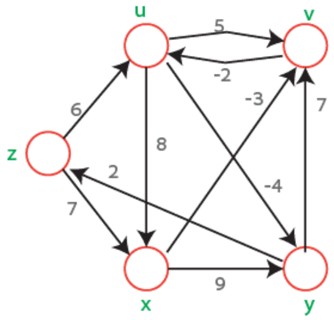
לאחר הסבב הזה, מתבצע סבב נוסף של בדיקה האם יש מעגל שלילי אינסופי, אם כן אז מחזירים FALSE.

והנה האלגוריתם:

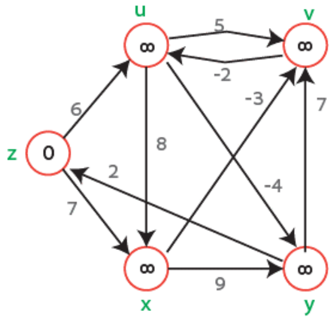
```
Bellman-Ford(G,w,s)                          // מקבלים את הגרף, רשימת המשקלות ונקודת המקור
  Initialize-Single-Source(G,s)              // אתחול הגרף
  for i = 1 to |V[G]|-1                      // ריצה במעגל עד הקדקוד האחרון
        for each edge (u,v) ∈ E[G]           // מעבר על כל הקדקודים המקושרים
          Relax(u,v,w)                       // הקלה
  for each edge (u,v) ∈ E[G]                 // בדיקה של כל הצלעות
        if d[v]>d[u]+w(u,v)                  // אם תנאי זה מתקיים – יש מעגל שלילי אינסופי
```

```
        return FALSE
return TRUE                      // במידה וכל המעבר חוזר ללא שגיאות מחזירים אמת
```

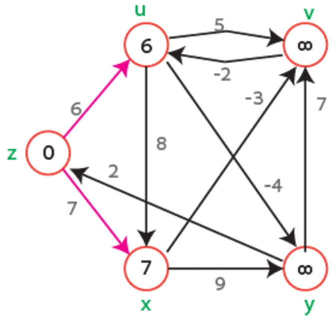
עבור הדוגמה נשתמש באותה אחת שנעשתה גם בספר:



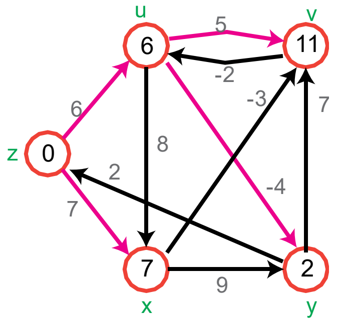
זה הגרף הבסיסי איתו אנחנו מתחילים, ניתן לראות שיש בו מעגלים, ויש בו משקלים שליליים, אך אין בו מעגל שלילי. דבר ראשון – נאתחל את הערכים:



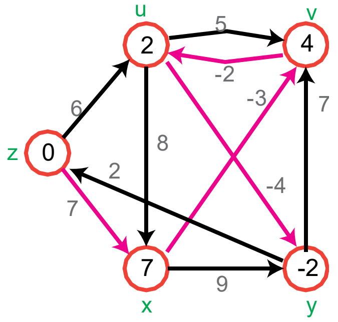
מתחילים את הריצה מהקדקוד ההתחלתי, צובעים את הקשתות הרלוונטיות, מעדכנים את הערכים וממשיכים הלאה על פי סדר אלפבתי.



עבור הקדקוד u, נבדוק את צמד הקשתות היוצאות ממנו – (u,v)(u,y). זה יעדכן לנו הערכים בקדקודים האלה.

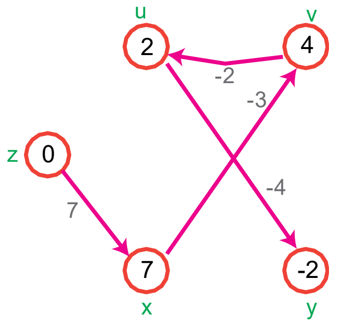


למעשה עכשיו אנחנו כבר עוברים הלאה לקדקוד v. הקשת היוצאת ממנו (v,u) בודקת האם 2-11> 6 . מאחר ולא, לא נעשה שום שינוי וממשיכים הלאה.



בשלב זה, שהוא השלב הסופי, מתרחשים לא מעט אירועים. אנחנו בודקים את הקדקוד x. ואת הצלעות היוצאות ממנו (x,y)(x,v). הקשת (x,v) שהמשקל שלה שלילי גורם לנו לעדכן את הקדקוד v להיות 4. עכשיו אנחנו צריכים לבדוק שוב את הצלעות היוצאות מ-v. אם עדכנו אותו, הרי יכול להיות שזה ישפיע על קדקודים אחרים שמקושרים אליו. ואכן, (v,u) שגם הוא בעל משקל שלילי מעדכן את u, ומשנה אותו מ-11 ל-2. הקדקוד y, על אף הערך השלילי הקיים בו, לא באמת מעדכן שום ערך – לו הקשת (y,z) היתה מעדכנת את z להיות נמוכה יותר מ-0, זה היה גורם לכל הגרף להיות בעל משקלים אינסופיים שליליים.

בסופו של דבר, אנחנו מסיימים את ההרצה ומקבלים את העץ הבא:



שבוייה.

ניתוח זמן ריצה

האתחול עצמו מתבצע בזמן $\Theta(V)$, פשוט עוברים על כל הקדקודים.

כל אחד מהמעברים על הקשתות נעשה תחת חסם עליון של גישה לכל הקדקודים – $O(VE)$

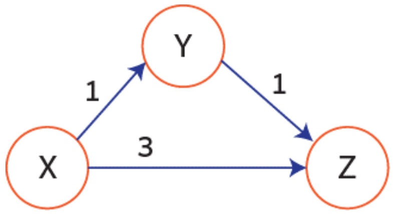
הוכחת נכונות האלגוריתם

לא הגענו לזה בשיעור.

שאלות הרחבה

נתון גרף G(V,E) מכוון וממושקל.
טענה: אם נוסיף קבוע חיובי k>0 לכל משקל קשת בגרף, עץ המרחקים הקצרים מקדקוד כלשהו בגרף לא ישתנה. הוכח או הפרך.

לא נכון.
אמנם ראינו תרגיל בשיעורי הבית של עץ פורש מינימלי, בו הוספנו משקל אחיד אך התוצאה לא השתנתה. אך כאן ניתן לראות באמת את ההבדל בדרישה בין שני האלגוריתמים. בעוד שבמציאת עפ"מ, ברגע שמעלים את כל הערכים הכל ישתנה באותו ביוון, כאן הוספה של ערכים יכולה להיות הרסנית. מדוע? אם יש לנו מסלול שעובר בין כמה קדקודים שונים, כאשר יש מסלול ישיר, אך יקר יותר, הוספת המשקל עלולה לגרום לתיעדוף של אותה קשת על פני המסלול האחר.



ניקח לדוגמה את הגרף הבא. ברור שנעדיף ללכת מהמסלול שעושה סיבוב, אך שוקל רק 2, לעומת הישיר שעולה 3 (דרך קצרה שהיא ארוכה וכל זה), אבל אם נוסיף k=2, משקל הקשת (x,z) יהיה 5, לעומת המסלול x→y→z שישקול 6.

נתון גרף G(V,E) מכוון וממושקל.
טענה: אם נוסיף קבוע שלילי k<0 לכל משקל קשת בגרף, עץ המרחקים הקצרים מקדקוד כלשהו בגרף לא ישתנה. הוכח או הפרך.

למעשה, מדובר במעט באותה שאלה כמו קודם, רק בביוון אחר. ברור שברגע שאנחנו מכניסים פה משהו שליךי, אנחנו שוללים את האפשרות להשתמש באלגוריתם דייקסטרה. אם ננסה להשתמש באלגוריתם דייקסטרה על גרף שיש בו קשתות במשקל שלילי, הוא יחזיר תשובות לא נכונות.

אבל גם אם נישאר בערכים חיוביים, אם נסתכל בדיוק על אותו גרף מהשאלה הקודמת, אבל לאחר הוספת הk=2. נקבל מסלולים קצרים חדשים, ואם נוריד את מה שהוספנו (k= -2), אז נחזור לתשובה אחרת, ולכן גם זה לא מתקיים.

נתון: גרף מכוון וממושקל G(V,E) עם פונקצית משקל w→e∈E. כמו כן, נתון קדקוד s∈V. כתוב אלגוריתם יעיל בכל האפשר שמוצא מסלולים קצרים ביותר מכל קדקוד בגרף ל-s.

כשהתחלנו לדבר על המסלולים הקצרים ביותר, העלינו את הגרסאות השונות לבעיה, ובעיה זאת היתה

אחת מהן. כמובן שהפתרון לבעיה הוא יחסית פשוט – יש לנו אלגוריתם שעושה בדיוק את זה, רק בכיוון ההפוך. המשימה שלנו היא, להגדיר את השינויים הנדרשים על מנת שיצליח לפתור את הבעיה הזאת, והחלק החשוב והלא-פחות-מסובך, להוכיח את הנכונות של מה שעשינו.

הרעיון של האלגוריתם שנציע הוא כזה – אם אנחנו רוצים למצוא את כל הדרכים המובילות לקדקוד אחד, כמובן שלא נבדוק את הקדקודים עצמם – אם הם מגיעים ומה הדרך הקצרה ביותר, כי זה ייקח לנו יותר מידי זמן. מה שנעשה הוא בעצם השיטה הישראלית לנסיעה ברחוב של "אין כניסה". ניסע ברברס. או במילים אחרות –

1. נהפוך את כל כיווני הקשתות – מה שייצור לנו מצב, בו הקדקוד s, במקום להכניס אליו את המסלולים יוציא אותם לעבר הכיוון ההפוך.
2. נריץ אלגוריתם למציאת מסלולים קצרים ביותר מ-s. (דייקסטרה או בלמן פורד, על פי תנאי הגרף)
3. ניקח את העץ שהתקבל, ונהפוך את כל הכיוונים בחזרה לכיוון המקורי (שזה אלגוריתם בפני עצמו, אבל נתעלם מזה כרגע).

כל שנותר לנו עכשיו, הוא רק להוכיח את הנכונות של מה שעשינו. לצערינו, נפנופי ידיים לא ממש מתקבלים בהבנה בקורס, ואמירות של "נו, זה עובד" לא עובדות.

מה שננסה הוא להוכיח בשלילה (כמה מפתיע) שזה לא נבון, ויש מסלולים קצרים יותר. ונראה שאם באמת הטענה מתקיימת סתרנו את נכונות האלגוריתם בו השתמשנו.

הוכחה:

נניח בשלילה שהאלגוריתם שהצענו לא נכון, ולא מביא את המסלולים הקצרים ביותר מכל קדקוד לקדקוד s.

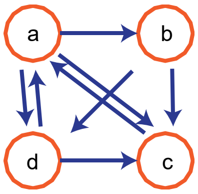
יהיה $P_1(v,s)$ מסלול מהקדקוד $v\in V$ אל s שהאלגוריתם שלנו חישב, והוא באמת אינו הקצר ביותר. ויהי P_2 המסלול הקצר ביותר האמיתי מ-s ל-v.

נהפוך את כיווני הקשתות של $P_1(v,s)$ ונקבל את $P_1(s,v)$. נהפוך את כיווני הקשתות ב-P_2, ונקבל את P'.

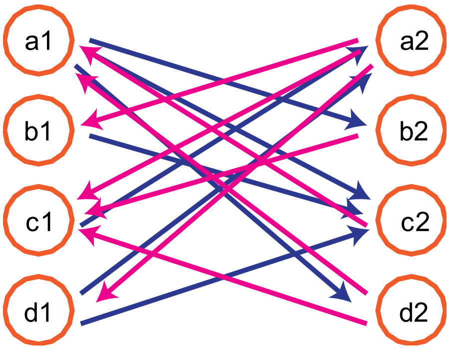
בעת נאמר ש-P' מסלול קל יותר במשקלו מהמסלול $P_1(s,v)$, כי שינינו רק כיווני מסלול ולא משקלים, ואם האלגוריתם שכתבנו הביא תוצאה לא נכונה, אז לא מדובר על אותו מסלול. ואם כך, אז יש מסלול $P_n(s,v)$ אחר יותר קל, וזה הרי לא ייתכן, כי אנחנו עבנו עם האלגוריתם של בלמן-פורד/דייקסטרה, וידוע לנו שהם עובדים. ויש פה סתירה. מש"ל.

נתון גרף $G(V,E)$ מכוון וממושקל ונתון קדקוד s בגרף.
עלינו למצוא מסלולים קלים ביותר מקדקוד s לכל קדקוד אחר בגרף, כך שהמסלולים יהיו בעלי אורך זוגי (בהתייחס למספר הקשתות שבדרך).

על מנת לפתור את הבעייה הזאת, נשתמש בבניית עזר. נכפיל את קדקודי הגרף, ונשתמש בדיוק באותם קשתות, כאשר מה שאנחנו צריכים לדאוג הוא שבתוך כל חלק של הכפלה לא יהיו קשתות פנימיות, אלא כל קשת תעבור תמיד מצד לצד, באופן הבא:



אם זה היה הגרף המקורי שלנו, אנחנו מפרידים לשני קצוות, ומעבירים את כל הקשתות **פעמיים**, פעם אחת מכל צד לכל קשת. זה ייראה ככה (קצת בלאגן, אבל הרעיון אמור להיות מובן):

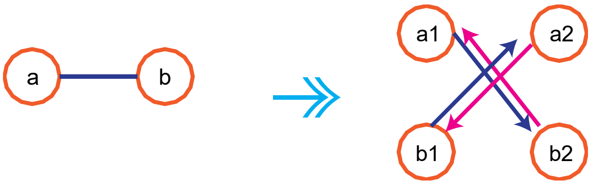


עכשיו, נריץ את האלגוריתם שהחלטנו להשתמש בו, ונקבל בתור מסלולים רצויים רק את אלה שמסתיימים בצד השמאלי (לצורך העניין, אם a יוגדר להיות נקודת המקור, אז אנחנו מחפשים מסלולים שמתחילים בקדקוד a1 ומסתיימים באותו צד). מסלולים שכאלה בוודאי יהיו בעלי מספר קשתות זוגי, מאחר ואין לנו קשתות פנימיות בתוך הקבוצות, אלא רק קשתות חוצות. נשאר לנו רק להראות שבאמת כל המסלולים ב-G נמצאים ב-G', ולהיפך, להראות שההכלה פה היא מלאה ולא הוספנו ב-G' מסלולים שלא היו קיימים ב-G.

מה שאנחנו צריכים להוכיח בעצם, זה שיש לנו את כל המסלולים באורך של קשת אחת, ואת כל המסלולים בעלי 2 קשתות (כמובן, שברגע שנוכיח את זה, אז כל המסלולים יכולים להיות מחולקים באותו רגע לזוגי ואי-זוגי).

עבור על קשת ב-G, יצרנו 2 קשתות – אחת $G'_{1\to2}$ והשניה $G'_{2\to1}$ כך שבעצם כיסינו את כל המסלולים באורך 1. עבור כל מסלול ארוך יותר – מאחר שכיסינו את כל המסלולים באורך 1, הרי שמסלול באורך 2 הוא פשוט הרכבה של שני מסלולים באורך 1. ואם יש לנו את כל המסלולים באורך 1, אז גם המסלולים האלה מכוסים. וכן על זה הדרך ושלום על ישראל.

הערה: ניתן להריץ את אותו רעיון גם על גרף לא-מכוון. ההסתכלות על כל קשת לא-מכוונת בודדה תהיה פשוט כשתי קשתות מכוונות בכיוון מנוגד. לצורך הדוגמה:



בעצם את הקשת הלא-מכוונת שכפלנו 4 פעמים, 2 כיוונים * 2 חלקי גרף.

נתון גרף G(V,E) מכוון וממושקל, עם קדקוד s, ונתונה לנו קבוצת קדקודים V'⊆V (קדקודים "מיוחסים").

כתוב אלגוריתם יעיל ביותר שמוצא מסלולים קצרים ביותר היוצאים מ-s לכל הקדקודים, ועוברים לפחות על קדקוד אחד מ-V'

הדרך הפשוטה לחשוב על זה, זה ניווט למקום מסוים, עם דקירה של נקודות בדרך – לנסוע לאולם חתונות ולעבור בדרך בבספומט/ דוכן פלאפל (תלוי כמובן לאיזה אולם נוסעים).

במובן מסוים, השאלה הזאת מאוד דומה לשאלה שהיה גם קודם. ואכן, דרך הפתרון מאוד דומה, אך יש לעמוד על ההבדלים והאפשרויות השונות. גם כאן אנחנו נשתמש באותה הכפלה אך בסגנון שונה – כאן אין לנו בעיה בקשתות פנימיות בתוך שתי הקבוצות השונות, מה שמשנה לנו כאן, זה רק המעבר לקדקוד ששייך ל"קולים". לכן נגדיר את הקשתות באופן הבא – הקשתות החוצות בין שתי הקבוצות יהיו רק כאלה שעוברות לקדקוד מיוחס. המסלול שיתקבל הוא רק מסלול שנמצא בקבוצה השניה, מאחר וברור לנו שהוא נגע בלפחות אחד מהקדקודים הדרושים, ומה שקורה אחר כך כבר פחות חשוב לנו. כמובן, שיכול להיות שאנחנו נמצא מסלול קצר שלא ייכנס לתוך הקבוצה, אך הוא לא עומד על כל הדרישות שלנו.

הערה: גם את השאלה הזאת ניתן להרחיב – אם נדרש למצוא מסלול שעובר בקדקודים האלה אך לא מסיים בהם, נכפיל את הגרף 3 פעמים, כך שהקישור בין הגרף השני לשלישי יהיה כל קשת שיוצאת מאחד הקדקודים המיוחסים לעבר קדקוד אחר, ובחזרה.

הרחבה נוספת שראינו בתרגול – אם יש דרישה למינימום קשתות במספר מסוים, וגם שיהיה מספר קשתות זוגי – מכפילים את הגרף למספר הדרוש בשביל המינימום 1+ (כי כל הכפלה מביאה לנו רק קשת אחת, אז אם רוצים שש קשתות באמצע, אנחנו נצטרך שיעברו בין שבעה קדקודים שונים). לאחר מכן מוסיפים עוד הכפלה אחר כך (או מחזירים לאחת ההכפלות הקודמות) על מנת שייתן לנו את האפשרות למעברים זוגיים הלוך וחזור.

ואידך, זיל גמור.

נתון: גרף לא מכוון G(V,E) ותת קבוצה w ⊆ V. נתונים שני קדקודים s,t ∈V.
תאר אלגוריתם יעיל ביותר למציאת מסלול p(s,t), המבקר במספר מינימלי בכל האפשר של צמתים מ-w.

דבר ראשון, נשים לב שהגרף לא ממושקל. כך שאין לנו איזה מסלול מסוים שכבר עכשיו נקבע לנו ואנחנו צריכים לבדוק אם הוא הקל ביותר.

מה שנעשה הוא להגדיר את המשקל בעצמנו. באופן דומה למה שראינו בתרגילים הקודמים, אנחנו נמיין את הקדקודים לשחור (קדקוד בטוח – לא עובר בקדקודים שמקושרים ל-w), ואדום (קדקוד לא בטוח – שייך לקבוצה w), ואת הקשתות ל-4 סוגים שונים:

1. קשת בטוחה – שחור לשחור – אותו נמשקל ב-0.
2. קשת לא בטוחה – אדום לאדום – יקבל 2.
3. קשת בעייתית 1 – משחור לאדום (כניסה לאיזור w) – יקבלו משקל 1.
4. קשת בעייתית 2 – מאדום לשחור (יציאה מ-w) – בדומה לקודם, יקבל משקל 1.

בעת נריץ את האלגוריתם לזיהוי מסלולים קצרים ונקבל את הדרוש.

הוכחת נכונות: למעשה, המשקלים שהבאנו לכל קשת, יתנו לנו בסופו של דבר את מספר הקדקודים האדומים בהם נעבור, ולכן, המסלול הקל ביותר יהיה בהכרח גם המסלול בעל הקדקודים האדומים הנמוך ביותר. משל.

אלגוריתם פלויד וורשאל

האלגוריתם של פלויד-וורשאל הוצג בשנת 1962 על ידי רוברט פלויד וסטיבן וורשאל (כל אחד בנפרד. מוחות גדולים וכו'). וכמו שנאמר כבר מקודם, הוא מבוסס על התכנון הדינאמי, ומצליח למצוא מסלול קצר ביותר מכל קדקוד לכל קדקוד אחר, והוא מצליח להתמודד בצורה לא רעה גם עם משקלות שליליים.
המיוחד בו לעומת האלגוריתמים שהצענו עד עכשיו, הוא שפלויד-וורשאל מקבל לא את הגרף, אלא את טבלת הסמיכויות של הגרף – מטריצה בגודל n*n המתארת באופן פשוט את המעברים מקדקוד לקדקוד.

הסבר האלגוריתם

עבור הטיפול באלגוריתם אנחנו מכינים לנו שתי מטריצות, בגודל n*n כל אחת. מטריצה אחת תכונה D ותביל בסופו של דבר את המשקל מכל קדקוד לחברו, והמטריצה השנייה תיקרא $\prod$ (פאי) ותביל את האיבר הקודם במעבר בין שני קדקודים.

המקרה הבסיסי בו נמלא את המטריצות, הוא המעבר הישיר בין כל שני קדקודים – השורות יבטאו את המוצא של הקשת והעמודות את היעד. אם לא נמצא מסלול שכזה, נשאיר את הערך בין שני הקדקודים כ-∞ (כמובן, שאם אין דרך באמת להגיע אל אותו קדקוד אנחנו גם נישאר באינסוף), ונסמן גם את המרחק בן כל קדקוד לעצמו בתור 0. בהתאמה נמלא גם את מטריצת הקודמים $\prod$, ונכתוב על הקשר בין שני קדקודים את הקודם שלו – בשורה הראשונה נכתוב כל קדקוד שיש קשת היוצאת אליו מ-1 ובן הלאה כל שורה (שימו לב: המספור הוא מ-1 ולא מ-0).ובכל הקדקודים שאין לנו מעבר מקדקוד מסוים, נסמן את הקודם בתור NIL, כמובן שהאלכסון של כל צלע לעצמה תהיה – NIL.

לאחר שהגדרנו את השלב הראשוני, כל איטרציה תעבור על האפשרויות הקיימות בין שני קדקודים או יותר, כאשר בכל פעם נוסיף רק קדקוד אחד למעגל הבדיקות, אך נבדוק את כל האופציות שמתווספות לנו על ידי זה – כלומר, אם אני עכשיו בודק את D^3, אני לא בודק רק את קדקוד 3, אלא את כל המעברים הנוגעים בו בישירות, או בעזרת קבוצת הקדקודים {1,2,3}. במידה ואנחנו מוצאים איזשהו קשר בעזרת הקדקוד הנוסף, שהוא נמוך יותר ממה שכתוב לנו כרגע (בתור התחלה, הכל יהיה נמוך יותר מ-∞), וכמו כן, נעדכן את מטריצת הקודמים, ונכתוב שבמסלול (i,j) אנחנו עוברים בקדקוד k כלשהו.

למעשה, אם אנחנו לוקחים את קבוצת הקדקודים {1,2...n} =V, נבחר איזה תת קבוצה של קדקודים {1..k} בהם ידוע לנו שעובר המסלול p, שהוא המסלול הקצר ביותר בין (i,j) כלשהם בגרף. עכשיו נבחר בכל פעם את הקדקוד ה-k, ונבדוק האם הוא חלק מהמסלול הקצר ביותר אותו אנחנו מחפשים (קצת בדומה למה שעשינו בתכנון הדינאמי של בעיית תיק הגב). עבור השאלה הזאת, האם k שייך למסלול, יש לנו בדיוק שתי אפשרויות, כן ולא:

- אם k לא שייך למסלול הקצר ביותר, אז בעצם המסלול הקצר ביותר לא שייך רק לתת הקבוצה (1,2..k}, אלא גם ל {1,2...k-1}, כי הרי k לא חייב להיות חלק מהקבוצה.
- k שייך למסלול הקצר – נחלק את המסלול שמביל את k לשני חלקים p1,p2 כאשר k עומד בתווך. שוב ניתן לראות, שגם p1 וגם p2 קיימים בקבוצה של {1,2..k-1}.

הנוסחה הרקורסיבית מוגדרת בצורה הבאה:

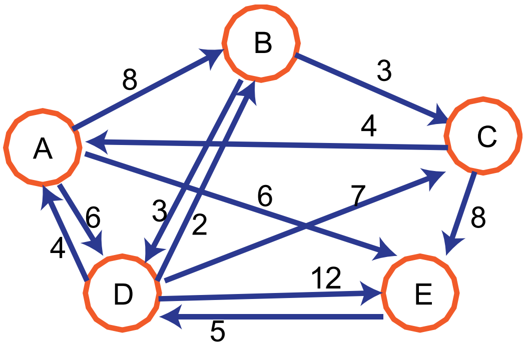
$$d^{(k)}_{ij} = \begin{cases} w_{i,j} & \text{אם } k = 0 \\ \mathrm{Min}(d^{(k-1)}_{ij},\ d^{(k-1)}_{ik} + d^{(k-1)}_{kj}) & \text{אם } k \geq 1 \end{cases}$$

// מקרה הבסיס
// כל מקרה אחר, בודקים אם אנחנו
// עושים שימוש בקדקוד k

נסתכל בעת על הקוד:

```
Floyd-Warshall(w)                                         // אנחנו מקבלים את טבלת הסמיכויות
  n = rows[W]                                             // מגדירים את המעגן למספר השורות
  D^(0) = W                                               // הגדרת אתחול הבסיס
  for k = 1 to n                                          // לולאה ראשית – הוספת כל קדקוד לבדיקה
      for i = 1 to n                                      // לולאות מקוננות לבדיקת המסלול
         for j = 1 to n
            d^(k)_ij = min (d^(k-1)_ij, d^(k-1)_ik + d^(k-1)_kj)   // הנוסחה הרקורסיבית שהגדרנו
  return D^(n)
```

נדגים את הריצה של האלגוריתם על הגרף שמובא לנו בשיעורי הבית (כי יש גבול, ואם כבר אני מריץ לפחות שיצא לי משהו מזה).



נתון הגרף הבא, אין בו משקלות שליליים, אבל זה לא אומר שהאלגוריתם לא יעיל פה. קודם כל – נבנה את שתי המטריצות:

$$D^{(0)} = \begin{bmatrix} 0 & 8 & \infty & 6 & 6 \\ \infty & 0 & 3 & 3 & \infty \\ 4 & \infty & 0 & \infty & 8 \\ 4 & 2 & 7 & 0 & 12 \\ \infty & \infty & \infty & 5 & 0 \end{bmatrix} \qquad \pi^{(0)} = \begin{bmatrix} NIL & 1 & NIL & 1 & 1 \\ NIL & NIL & 2 & 2 & NIL \\ 3 & NIL & NIL & NIL & 3 \\ 4 & 4 & 4 & NIL & 4 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

החלק הזה היה פשוט קריאה מהגרף (או טבלת סמיכויות, במידה ויש לנו), והכנסת הערכים במטריצה המתאימה – ניתן לראות את ההתאמה, בכל תא ב-D שיש ערך השונה מאינסוף, יש תא מתאים ב-π, המכיל את השורה שאנחנו נמצאים בה.

נעבור ל$D^{(1)}$:

$$D^{(1)} = \begin{bmatrix} 0 & 8 & \infty & 6 & 6 \\ \infty & 0 & 3 & 3 & \infty \\ 4 & 12 & 0 & 10 & 8 \\ 4 & 2 & 7 & 0 & 10 \\ \infty & \infty & \infty & 5 & 0 \end{bmatrix} \qquad \pi^{(1)} = \begin{bmatrix} NIL & 1 & NIL & 1 & 1 \\ NIL & NIL & 2 & 2 & NIL \\ 3 & 1 & NIL & 1 & 3 \\ 4 & 4 & 4 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

סימנתי את האיברים אותם ניתן לעדכן – שניים מתוכם פשוט הכנסנו להם ערך שהוא לא אינסוף, אולם (4,5) ירד מ-12 ל-10. כבר משהו יותר טוב. איך נדע לזהות על איזה מסלולים וקדקודים אנחנו צריכים להסתכל בשביל לראות האם הם השתנו? העמודות מציינות את הקדקודים אליהם ניתן להגיע מכל קדקוד – למשל, העמודה הראשונה מראה לנו שיש מסלול ישיר מ-3 ומ-4.

נמשיך ל$D^{(2)}$:

$$D^{(2)} = \begin{bmatrix} 0 & 8 & 11 & 6 & 6 \\ \infty & 0 & 3 & 3 & \infty \\ 4 & 12 & 0 & 10 & 8 \\ 4 & 2 & 5 & 0 & 10 \\ \infty & \infty & \infty & 5 & 0 \end{bmatrix} \qquad \pi^{(2)} = \begin{bmatrix} NIL & 1 & 2 & 1 & 1 \\ NIL & NIL & 2 & 2 & NIL \\ 3 & 1 & NIL & 1 & 3 \\ 4 & 4 & 2 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

לא היו פה הרבה עדכונים, אבל הורדנו אחד מהמסלולים בקצת.

נמשיך ל-$D^{(3)}$:

$$D^{(3)} = \begin{bmatrix} 0 & 8 & 11 & 6 & 6 \\ 7 & 0 & 3 & 3 & 11 \\ 4 & 12 & 0 & 10 & 8 \\ 4 & 2 & 5 & 0 & 10 \\ \infty & \infty & \infty & 5 & 0 \end{bmatrix} \qquad \pi^{(3)} = \begin{bmatrix} NIL & 1 & 2 & 1 & 1 \\ 3 & NIL & 2 & 2 & 3 \\ 3 & 1 & NIL & 1 & 3 \\ 4 & 4 & 2 & NIL & 1 \\ NIL & NIL & NIL & 5 & NIL \end{bmatrix}$$

אין צורך באמת לפרט כל שלב. אם היה לנו פו משקלות שליליים, יש סיכוי שהיו פה קצת יותר עדכונים, מוזמנים לנסות בזמנכם החופשי.

נעבור ל-$D^{(4)}$:

$$D^{(4)} = \begin{bmatrix} 0 & 8 & 11 & 6 & 6 \\ 7 & 0 & 3 & 3 & 11 \\ 4 & 12 & 0 & 10 & 8 \\ 4 & 2 & 5 & 0 & 10 \\ 9 & 7 & 10 & 5 & 0 \end{bmatrix} \qquad \pi^{(4)} = \begin{bmatrix} NIL & 1 & 2 & 1 & 1 \\ 3 & NIL & 2 & 2 & 3 \\ 3 & 1 & NIL & 1 & 3 \\ 4 & 4 & 2 & NIL & 1 \\ 4 & 4 & 4 & 5 & NIL \end{bmatrix}$$

למעשה, אין פה הרבה עדכונים, כי בחלק גדול מהמקרים אנחנו מגיעים לתוצאה שהיא שווה בדיוק למה שיש לנו – למשל (c,b) שעבר קודם $c \to a \to b$ ויכול עכשיו לעבור דרך $c \to a \to d \to b$ ולקבל את אותו משקל, לא רלוונטי לנו כי אנחנו מחפשים קטן ממש.

נבדוק אחרון את $D^{(5)}$:

$$D^{(5)} = \begin{bmatrix} 0 & 8 & 11 & 6 & 6 \\ 7 & 0 & 3 & 3 & 11 \\ 4 & 12 & 0 & 10 & 8 \\ 4 & 2 & 5 & 0 & 10 \\ 9 & 7 & 10 & 5 & 0 \end{bmatrix} \qquad \pi^{(5)} = \begin{bmatrix} NIL & 1 & 2 & 1 & 1 \\ 3 & NIL & 2 & 2 & 3 \\ 3 & 1 & NIL & 1 & 3 \\ 4 & 4 & 2 & NIL & 1 \\ 4 & 4 & 4 & 5 & NIL \end{bmatrix}$$

קצת מפתיע, אבל אין פה שום עדכון לאף תא. הווה אומר סיימנו את התהליך, ויש לנו את המטריצות הסופיות. עכשיו מה?

רק נזכיר – אנחנו חיפשנו את המסלולים הקצרים מכל קדקוד לכל קדקוד. שתי המטריצות שקיבלנו למעשה מחזיקות לנו את כל המידע הדרוש, ואנחנו צריכים רק לדעת כיצד להוציא אותו. לצורך כך נשתמש באלגוריתם רקורסיבי להדפסת מסלול בין שתי נקודות:

```
Print-All-Pairs-Shortest-Path(∏, i, j)      // הפונקציה מקבלת את מטריצת הקודמים והנקודות הדרושות
  if i=j                                     // אפשרות א' - כאשר מדובר בהכנסה של אותו קדקוד
    print i                                  // מדפיסים רק את הקדקוד
  else if πij = NIL                          // אפשרות ב' – לא קיים מסלול בין הקדקודים
     print "no path from" i "to" j "exists"
  else PAPSP(∏, i, πij)                      // אחרת – מכניסים רקורסיבית צעד קודם של המסלול
       print j                               // בסיום הריצה – מדפיסים את הקדקוד הסופי
```

נריץ על יבש את האלגוריתם, ונחפש את המסלול הקצר ביותר של (4,3)
(4,3) כניסה – הקדקודים לא שווים, ועל פי המטריצה יש לנו מסלול ביניהם. ולכן מכניסים לאלגוריתם את הקודם שרשום לנו במטריצה – 2

(4,2) כניסה – הקדקודים לא שווים (וברור שקיים לנו מסלול, אחרת לא היינו מגיעים לפה מלכתחילה), נכנסים שוב עם הקודם של (4,2) – 4

(4,4) כניסה – הקדקודים שווים! הגענו לתחילת המסלול! מדפיסים את i ← 4
חוזרים לקריאה הקודמת –

(4,2) המשך – מדפיסים את j ← 2
חוזרים לקריאה הקודמת –

(4,3) המשך – מדפיסים את j ← 3
יציאה.

המסלול שהודפס הוא 4→2→3.
סיימנו.

הערה: כיצד נדע האם יש לנו מעגלים שליליים בתוך הגרף? אמרנו שיש לנו שני אלכסונים חשובים – האלכסון של המטריצה D בו כל קדקוד לעצמו אמור להגיע במשקל 0 – הדרך היחידה בה דבר זה ישנה, הוא אם יש לנו מעגל שלילי, מה שייצור לנו מצב בו מעבר מקדקוד לעצמו, לא יהיה 0 אלא יהיה סך המשקל השלילי של המעגל.
בנוסף, מטריצת הקודמים אמורה להישאר בNIL באלכסון המרכזי, אם נראה שיש לנו מעבר מאותו קדקוד לעצמו, דרך קדקוד אחר, ככל הנראה מדובר בו על מעגל שלילי, אחרת אין סיבה לעדכן את הנתון הזה.

ניתוח זמן ריצה

את האלגוריתם הזה די קל לנתח – יש לנו 3 לולאות מקוננות, הן רצות במקסימום על כל הקדקודים בכל פעם – $n*n*n = n^3$. זמן הריצה של פלויד וורשל הוא $\Theta(n^3)$.

ההדפסה עצמה לוקחת לנו במקרה הכי גרוע n, אם יש לנו משהו מנוון לגמרי שאנחנו צריכים להכנס לכל הקדקודים בדרך. אם נרצה להדפיס ממש את כל המסלולים הקיימים, נצטרך שוב לעבור n איברים בכל פעם, שגם זה יוצא לנו n^3 ($n_{\text{זמן ריצה של האלגוריתם}}*n_{\text{איברים של מוצא}}*n_{\text{איברי יעד}}$), מה שמשאיר את זמן הריצה באותו סדר גודל מבחינה אלגוריתמית.

תפוסת מקום בזיכרון

האלגוריתם המקורי תופס לנו לא מעט מקום בזיכרון – יש לנו שתי מטריצות, כל אחת בסדר גודל של n*n, וזה עבור רק שלב אחד, ויש לנו n שלבים באלה, וזה לא מעט מקום. ניתן לשפר זאת אם נשתמש בכל פעם רק בשתי איטרציות של מטריצות, ובכל פעם שאנחנו מתקדמים לדאוג למחיקה של מידע מיותר (אם אנחנו ב$D^{(3)}$ אין לנו צורך במטריצות של $D^{(1)}$. יותר מזה, על ידי חישוב נכון, ניתן לשכתב בכל פעם על אותם מטריצות את הנתונים וכך לחסוך לנו בניה ומחיקה של מטריצות בכל פעם.

רשתות זרימה

נושא רשתות הזרימה נראה דומה מאוד לגרפים. קדקודים, צלעות ומה שביניהם, אך השימושים של רשתות הזרימה שונים לגמרי.
עד עכשיו, המשקלות שקבלנו והשתמשנו בהם בגרפים, היוו ערך שלם ממנו אנחנו לא יכולים לרדת או להשתמש בחלקו, ואילו בעת ברשתות הזרימה, אנחנו מתייחסים למשקל בערך עליון. מה הכוונה? רשת זרימה, כשמה כן היא, מבטאת לנו את היכולת להעביר "משהו" ממקור s(source) ליעד t (termainl\sink), כאשר כל מה שאנחנו מעבירים זורם דרך קדקודים שמקשרים ביניהם. לצורך הדגמה לעולם האמיתי, בדרך כלל מדברים על אפשרויות להעביר מוצרים ממפעל שנמצא בעיר אחת, לבין המחסן שנמצא בעיר אחרת. אך מאחר שלמפעל אין דרך לעבור ישירות מהמקור ליעד, הוא עובר דרך תחנות ביניים. הבעיה מתחילה ברגע שהדרכים בין התחנות לא יכולות להיות באותה איכות. למשל- המפעל יכול להוציא 50 ארגזים ביום, אבל בדרך היוצאת ממנו הוא יכול להעביר רק 30 ארגזים ביום – בוודאי שאין צורך שהמפעל יעבוד ויכין ארגזים שסתם ייערמו לו. באופן דומה, תכנון מערכת מים או תקשורת, המתבססים על צינורות והעברה בקיבולים שונים, אנחנו תמיד חסומים מלמעלה על ידי מעבר הקיבול הנמוך ביותר. מה שחשוב לשים לב, וגם נחזור עליו בהמשך, שהקדקודים שאנחנו משתמשים בהם, אין בהם שום קיבול. כל מה שנכנס חייב גם לצאת באותו רגע, ואנחנו לא יכולים להגיד לקדקוד שישמור לנו על מה שהגיע אליו.

המטרה שלנו ברשתות הזרימה – למצוא אלגוריתם שמביא לנו את הדרך האופטימלית למעבר מהמקור ליעד, וכל זאת תחת חסם פולינומי.

מושגים כלליים לרשתות זרימה

<u>רשתות זרימה</u> – גרף G=(V,E) מכוון שבו כל קשת (u,v) היא בעלת קיבול c (capacity) לא-שלילי ושונה מ-0. אם c(u,v) ∉ E , אז אנחנו קובעים כי c(u,v) = 0, כלומר אין קיבולת (אין זרימה) בין שני הקדקודים האלה. כמו שאמרנו קודם, אנחנו מגדירים את נקודת ההתחלה "המקור" ב-s, ואת הסיום "בור" ב-t. בדרך כלל, מדובר על כך שכל הקדקודים על אותו הגרף מחוברים ביניהם שכולם קשורים, או יכולים להיות קשורים למסלול העובר בין s ל-t, מה שכמובן אומר שהגרף קשיר, ובתאמה, יחס הקדקודים צלעות יקיים את הכלל |V-1| ≤ |E|.

<u>קיבולת נוכחית</u> – בתחילת התרגילים, אנחנו נקבל גרף עליו מסומנים הקיבולות השונות במצב המקסימלי שלהם. ברגע שנתחיל לעבוד "ולהזרים" בקשתות השונות, נצטרך לסמן זאת על הקשתות. הסימון יופרד בקו אלכסוני במתבונות הבאה – מקס' קיבולת/זרימה נוכחית. לדוגמא 7/11 מבטא שיש לנו אפשרות להזרים עוד 4 יחידות באותה הקשת.

<u>זרימה</u> – העניין לשמו התכנסנו הערב. זרימה היא פונקציה ממשית ברשת שהוגדרה למעלה f: VxV→R (f במשמעות flow), שיכולה לעבור בין כל קדקוד וקדקוד בגרף, ועל מנת שתחשב זרימה, היא חייבת לקיים שלוש תכונות חשובות –

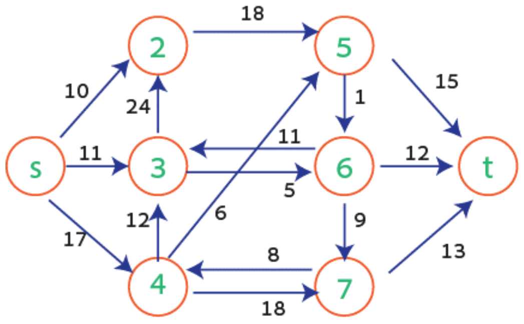
1. **אילוצי קיבול** (Capacity Constrains) – לכל u,v ∈ V, f(u,v) ≤ c(u,v) – החלק הזה די אינטואיטיבי. אף קשת לא יכולה שיזרום אצלה יותר מהקיבול המוגדר למקסימום.
2. **סימטריה נגדית** (Skew Symmetry) – לכל u,v ∈ V, f(u,v) = -f(v,u). זה כבר פחות אינטואיטיבי, ומשתמשים בזה לא מעט, אז באסה. אם יש לנו זרימה (חיובית) בין שני קדקודים (u,v), ואנחנו רוצים עכשיו להעביר הפוך בין (v,u), אנחנו יכולים פשוט להוריד את זה מהזרימה

ולתת את זה לצד השני. לפעמים השימוש באפשרות הזאת, חורג מהגדרת המקסימום, אבל זה תקין מאחר שאם אתה לא מזרים 2 יחידות לקדקוד אחד, אתה חייב להעביר אותם לאנשהו, ואז אתה יכול "להגדיל" את הקיבולת. (ולמי שמחפש דוגמא מהחיים – נגיד שאשה, או גבר, אנחנו לא שוביניסטיים, מוצאת בגד שהיא רוצה לקנות במחיר יותר זול ממה שהיא ציפתה, אז היא בעצם חסכה את הכסף הזה שהיא לא הוציאה על הבגד, וזה לגיטימי לקנות נעליים).

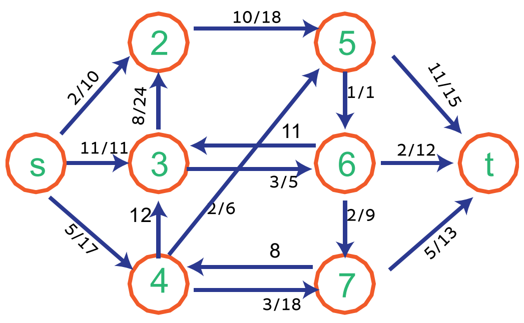
3. **שימור הזרימה** (Flow Conservation) – לכל $u \in V - \{s,t\}$, אנחנו דורשים כי $\sum_{v \in V} f(u,v) = 0$. דבר זה נגזר בעצם מכל מה שתיארנו קודם. מאחר ולקדקוד אין קיבול, ואנחנו מתייחסים לכל זרימה יוצאת בזרימה הפוכה לנכנסת, עלינו לוודא שסך כל הזרימות המקושרות לקדקוד מסויים ייסכמו ל-0. מכלל זה אנחנו מוציאים את קדקודי המקור והיעד, מאחר והם זורמים רק לכיוון אחד – המקור רק מוציא זרימה, והיעד רק מקבל.

זרימה נטו – $|f| = \sum_{v \in V} f(u,v)$ מוגדרת כסך כל הזרימות בין שני קדקודים מסוימים, כאשר בדרך כלל את אחד הערכים יחליף המקור או היעד, ויבטא את סך הזרימה עד מקום מסוים, או ממקום מסוים לבור. בנוסף, קיימת גם הזרימה נטו חיובית, המתייחסת כמובן, רק לזרימה החיובית בין הקדקודים, ללא התחשבות בקיזוזים וזרימות סימטריות.

ניקח בדוגמא את הרשת הבאה:



רשת הזרימה הזאת היא הרשת המקורית לפני שהתחלנו להעביר בה. קדקוד s מוציא זרימה בשלוש דרכים, וקדקוד t מקבל זרימה בשלוש דרכים שונות. נניח שהתחלנו להעביר בו את הזרימה, לאחר מספר איטרציות הרשת עלולה להיראות באופן הבא:



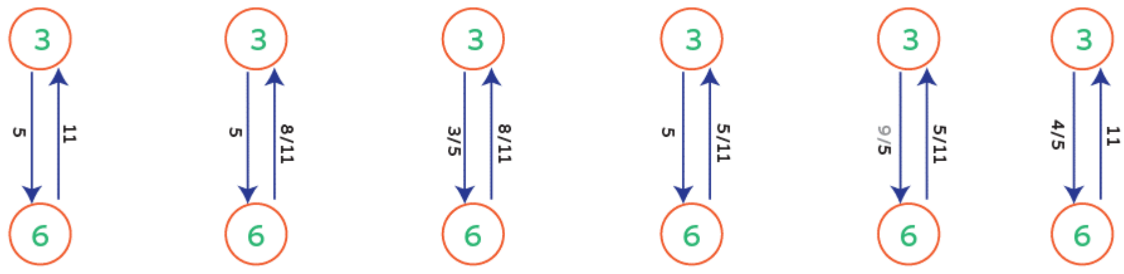
כמובן, שזה רק חלק מהמעבר, ויש פה עוד אפשרויות לשפר, אבל ניתן כבר לראות שאנחנו מקיימים את החוקים הנדרשים – אף קדקוד לא מעביר יותר מהקיבולת שלו. חוק שימור הזרימה נשמר אף הוא, מאחר וכל קדקוד מוציא בדיוק מה שהוא מקבל – ניקח לדוגמא את קדקוד 5.

לקדקוד 5 נכנס זרימה משני מקורות שונים – (2,5) מעביר אליו 10, ו-(4,5) מעביר אליו 2. סה"כ 12.

קדקוד 5 גם מוציא זרימה לשני קדקודים שונים – (5,6) מוציא 1, (5,t) מוציא 11. סה"כ 12.

מה שנכנס הוא שיוצא, ואין קיבול תחת הקדקוד.

<u>קיזוזים</u> – קיזוזי זרימה, זה המהלך שאנחנו יוצרים על בסיס הסימטריה הנגדית ושימור הזרימה. אם יש לנו מעבר דו-כיווני בין שני קדקודים, וזרימה באחד או יותר מהקדקודים האלה, יש לנו כמה אפשרויות לייצג את הזרימה הדו כיוונית הזאת. ניקח לדוגמה זרימה בין שני קדקודים באופן הבא – הקשתות בין הקדקודים 3, ל-6, מכילות אפשרויות לזרימה דו כיוונית. ואכן, ברשת הזרימה שתיארנו, יש זרימה בקשת (3,6). מה יקרה אם נרצה להוסיף זרימה גם בכיוון השני? נראה את התיאור הבא –



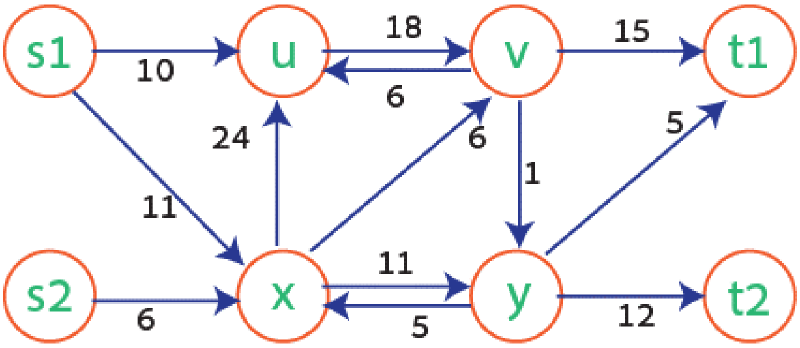
נתחיל משמאל לימין, ונציג את הקיזוזים האפשריים. משמאל נתחיל במצב הרגיל, יש לנו על כל קשת רישום של הקיבול המקסימלי שלה. בעת יש לנו זרימה בקשת (6,3) של 8. כמובן שאין עם זה בעיה ובכלום לא אמור להשתנות. בשלב השלישי, אנחנו מזהים אפשרות לקיזוז – יש לנו מעבר מצד אחד של 8 ומצד שני של 3. כמובן שדבר כזה קצת מיותר, מאחר ואנחנו מדברים על מעברים שקורים לכאורה בו זמנית. אין שום סיבה שתוציא החוצה מקדקוד 3 את מה שנכנס אליו, ותחזיר בחזרה לאותו קדקוד. מה שנעשה, הוא דבר די אינטואיטיבי ופשוט נפחית מקשת (6,3) את הזרימה ההפוכה לה, וכך נוכל לקזז את שתי הזרימות בשביל לקבל מצב של זרימה רק בכיוון אחד. השלב הבא, הוא החידוש היותר גדול – נגיד ואנחנו רוצים עכשיו להעביר בקשת (3,6) 9 יחידות. אוטומטית אנחנו ישר קופצים (או לפחות אמורים לקפוץ) ולהגיד שזה סותר את החוק הראשון של הזרימות – לא מדברים על זרימות שהן גדולות יותר מקיבולים! איך נוכל לעשות זאת בכל אופן? על ידי שימוש בחוק השני. אנחנו רואים שיש לנו זרימה של 5 ב(6,3), ועל פי מה שלמדנו, אנחנו יכולים "להחזיר" את הזרימה אחורה וכך לשנות את הזרימה הזו ל-0, ובצד השני להעביר רק 4/5, ועדיין זה ייחשב לנו כאילו אנחנו מעבירים 9! (יש פה עוד איזה קיזוזון ביניים שלא הכנסתי, תעשו את זה בראש).

חשוב לזכור- דבר שלא מסתכלים עליו פה בדוגמא שהבאתי – אנחנו לא יכולים פשוט לקזז כמה שבא לנו. כל קיזוז שאנחנו עושים, משנה את הזרימה הנכנסת והיוצאת לקדקודים אחרים, ולכן כל קיזוז שיתבצע צריך לעבור בכל הסובב אותו כדי לוודא שאנחנו שומרים על חוק שימור הזרימה.

מקורות ובורות מרובים

עד עכשיו דיברנו על מקור ובור יחיד. אך מה קורה כאשר יש לנו יותר מקורות או יותר בורות, או יותר משניהם? בלי פאניקה.

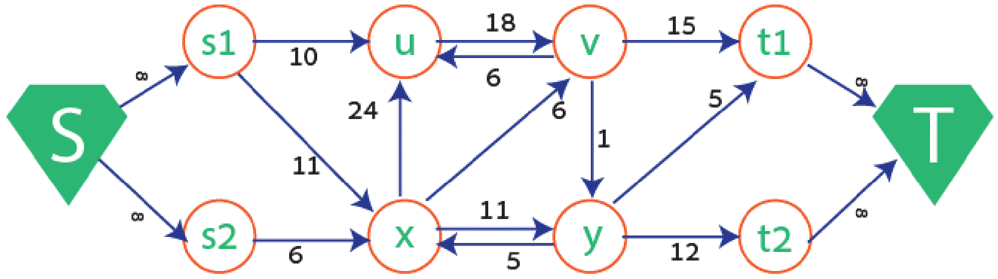
נגיד שיוצא לנו הגרף הבא:



ברור שדרך הפתרון שהצענו עד עכשיו, לא רלוונטית פה, מאחר ואין לנו דרך להזרים רק ממקור אחד. אנחנו צריכים שהכל יעבוד במקביל ובצורה מסונכרנת. איך נעשה את זה? בניית עזר!

נוסיף מקור חדש "מקור-על" (SuperSource), בח-העל שלו יהיה להזרים לכל המקורות את המקסימום האפשרי. נוסיף גם בור חדש "בור-על" (SuperSink), שבח-העל שלו יהיה בהתאמה לקבל את המקסימום. ביחד הם יילחמו בפשע, כי אולי הם לא הקדקודים שיש לנו, אבל הם בהחלט הקדקודים שאנחנו צריכים. נמשיך.

מכל אחד מהצמד החדש נעביר קשת למקורות או מהבורות אל הבור-על, ונגדיר את הזרימה להיות אינסוף באופן הבא –



עכשיו כאשר נזרים את המקסימום מ-S, הוא יוכל לספק לנו את שני המקורות המקוריים במקסימום שהם יכולים לספוג, שזה בדיוק מה שהם מוציאים. ואותו דבר יקרה גם בצד השני, כל מה שלא יגיע לבורות המקוריים יוכל להמשיך הלאה לבור-העל, אבל כמובן שהוא לא יוכל להעביר יותר מאשר הקשתות מובילות אל הבורות. ועכשיו כולם רגועים. כתוביות. סצנת אפטר-קרדיטס.

עבודה עם זרימות

בגדול, כאשר מסתכלים על זרימות, אנחנו מדברים על חתכים שונים של הרשת (נרחיב בהמשך), המחלקת אותה לשני חלקים שבדרך כלל מכונים X,Y. כאשר מתייחסים לקדקוד אחד מתוך הקבוצות, מדברים על x,y ששייכים כמובן כל אחד בהתאמה לקבוצה הגדולה שלו. הזרימה של כל קבוצה, היא סך הזרימות של כל הקדקודים תחת הקבוצה ההיא, כך שאם נחפש את הזרימה של שתי הקבוצות נגדיר זאת באופן הבא: $f(X,Y) = \sum_{x\in X}\sum_{y\in Y} f(x,y)$, שזה בעצם אומר, שאנחנו לוקחים כל קדקוד בכל קבוצה ובודקים את סך הזרימות המשותפות ביניהן.

באופן דומה ניתן גם להשתמש בסימון של קבוצה גדולה כדי לציין ריבוי מסלולים, למשל f(u,V) מציין את כל המסלולים היוצאים מ-u לקבוצה מסוימת.

ישנן מספר "זהויות" ביחסים בין הזרימות, אותם חשוב להבין על מנת שנוכיח איתם את האלגוריתמים והשיטות לשימוש ברשתות זרימה.

למה 27.1

תהי $G=(V,E)$ רשת זרימה, ותהי f זרימה ב-G.

אזי **עבור $X\subseteq V$ מתקיים $f(X,X)=0$**. – מאחר שאמרנו שיש לנו את שימור הזרימה, הדורש שבכל קדקוד הזרימה לעצמו תהיה 0, אזי ברור לנו שגם אם ניקח קבוצת קדקודים, מה שיווצר לנו זה חיבור ענק של אפסים, וכמו שאנחנו יודעים מהבחירות, שום דבר חיובי לא יוצא מחיבור אפסים. בלי פוליטיקה.

עבור $X,Y\subseteq V$ מתקיים $f(Y,X)=-f(X,Y)$ שוב בדומה למעבר על קדקוד בודד, גם כאן קבוצת קדקודים שיתהפכו, יביאו בסופו של דבר את הנגדי של אותה זרימה.

עבור $X,Y,Z\subseteq V$ כאשר $X\cap Y=\phi$ מתקיים $f(X\cup Y,Z)=f(X,Z)+f(Y,Z)$, $f(Z,X\cup Y)=f(Z,X)+F(Z,Y)$. כלומר, אם יש לנו שתי קבוצות קדקודים זרות, אז הזרימה המשותפת שלהם לקדקוד אחר ולהיפך, שווה לחיבור של כל מחלקת זרימות בנפרד. דבר זה מתקיים כמובן רק בקבוצות קדקודים זרות, כי אם היה חלק משותף, אז האיזור המשותף באיחוד היה מבטל אחד את השני על חלק מסוים והתוצאה היתה שונה.

שיטת פורד פולקרסון

בהינתן לנו רשת זרימה כלשהי, אנחנו שואפים למצוא את הזרימה המקסימלית האפשרית בה. השיטה של פורד פולקרסון, לפחות בגרסה המקורית שלה, היא אחת מאלגוריתמי ה"טא-דא!" שבולנו כל כך אוהבים. הרעיון של השיטה, הוא לעבור בשיטה איטרטיבית על מסלולים ולשפר בכל פעם קצת עד שמגיעים לתוצאה הטובה ביותר. כן – תעשו עד שזה בסדר. ובצורה קצת יותר מקודדת:

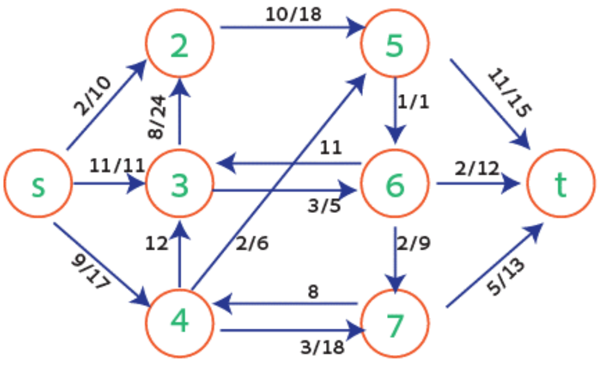
```
Ford-Fulkerson-Method(G,s,t)              // מקבלים רשת עם נקודת התחלה וסיום
  initialize flow to 0                    // הזרימה הראשונית היא כמובן 0
  while there exists an augmenting path p // כל עוד קיימת דרך שמשפרת את הזרימה
        augment flow f along p            // השתמש בדרך, ושפר את הזרימה
  return f
```

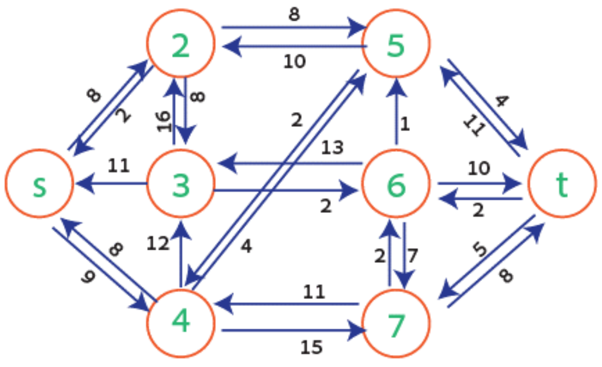
השיטה הזאת כל כך כללית, ולא אומרת כיצד לבצע באמת את הדרך, עד שמקפידים לכנות אותה "שיטה" ולא "אלגוריתם" שמבטא משהו מסודר ועקבי. על מנת להבין בצורה מלאה איך אנחנו עובדים, נצטרך להסביר את מושג ה"שיור".

<u>רשתות שיוריות</u> – עבור כל רשת זרימה שנקבל, המכילה את הזרימה ביחס לקיבול באותו רגע, קיימת לנו רשת מראתית, המבטאת כמה עוד אנחנו יכולים להזרים בין שני הקדקודים. איך בונים את הרשת השיורית? נסתכל קודם על <u>קשת שיורית</u> – עבור כל שני קדקודים החולקים ביניהם זרימה כלשהי, אנחנו יכולים לקבוע שתי קשתות שמבטאות את ההזרמה המלאה בה אנחנו יכולים להשתמש – ראשית, הזרימה לאותו כיוון עם השארית בה עוד לא השתמשנו, ולכיוון השני, אנחנו יכולים לשלוח את הזרימה השלילית בה אנחנו משתמשים. לדוגמא:

הזרימה בקשת (2,5) מוגדרת כ10/18. השארית של הזרימה ב-(2,5) היא האפשרות להזרים עוד 8 באותו כיוון. אך מעבר לזה, על פי חוק הזרימה הסימטרית, אנחנו יכולים גם להעביר זרימה ב-(2,5) בכמות שווה בגודלה למה שזורם - 10. ולכן הקשת מתפצלת לשניים (הכנס כאן תמונה מהמערכון של אסי וגורי) ומתוכה יוצא זרימה ענקית לשני הכיוונים.

בהתאמה – הרשת השיורית, תהיה מימוש של **כל** הקשתות והאפשרויות להזרמה **בכל** הגרף. נסתכל שוב על הגרף הקודם, ועל הרשת השיורית השייכת לו:

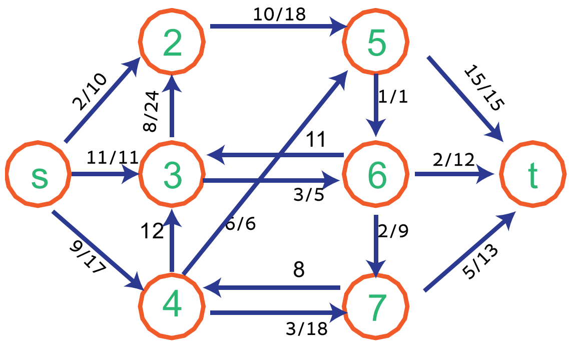


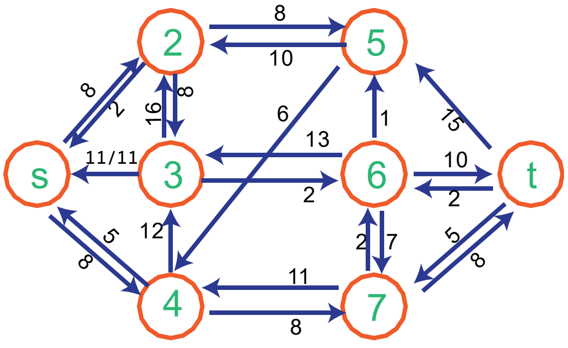


יש כמה דברים שחשוב לראות ברשת השיורית – הקשת השיורית הרגילה אותה ראינו בדוגמה, כמו למשל (2,5) שהתפצלו לשאריות לכל כיוון. מה שמעניין יותר, הוא היחס לזרימה ריקה, ולזרימה רוויה. הקשת (5,6) מכיל אפשרות לזרימה של יחידה בודדת, שבאה לידי ביטוי בזרימה, כך שהקשת "רוויה", בהתאמה, הקשת היחידה המופיעה ברשת השיורית של אותו מקום, היא רק האפשרות להחזיר את הזרימה לקדקוד המקורי.

מצד שני, אם נסתכל על (4,3) נראה שהקיבול הוא 12, אך לא עובר שום דבר באותו צינור. אז אם אין לנו זרימה בפועל, כמובן שאין לנו מה להחזיר לקדקוד השני, ולכן הקשת נשארת בדיוק כמו שהיא.

אחרי שבנינו לנו את הרשת השיורית, אנחנו מחפשים איזה מסלול p שיוביל לנו בין s ל-t, ויכול להוסיף לנו לזרימה עוד קצת, מסלול כזה נקרא <u>מסלול שיפור</u>. למשל אם ניקח את המסלול הבא – s→4→5→t אנחנו יכולים לשחק עליו קצת ולהוסיף לזרימה. כמה יכול להתווסף לנו על המסלול הזה? נבדוק את הזרימה של כל הקשתות במסלול, ווייצא לנו: 12,4,4. מכאן אנחנו מסיקים, שאם נבחר במסלול הזה, אנחנו יכולים לשפר את הזרימה ב-4 יחידות. נשנה את הרשת המקורית בהתאם וננסה שוב:





ברשת השמאלית אנחנו רואים את הזרימה שהוספנו למסלול, וברשת השיורית, אנחנו מגלים בעת, שלמעשה "חסמנו" עכשיו את הדרך ליעד מכיוון אחד, מה שאומר, שבכל המשך השיפורים יכולים להיות רק

משתי הקשתות האחרות. ברגע שנגיע למצב בו כבר לא ניתן להוסיף לזרימה הנכנסת ל-t, נדע שסיימנו לשפר והגענו לזרימה המקסימלית.

את הבחירה של מסלולי השיפור, אנחנו עושים בעזרת חתכים. באופן דומה למה שהגדרנו בגרפים, גם כאן אנחנו מגדירים חתך כאוסף קדקודים, כאשר הם מוגדרים בדרך כלל כ-S,T כאשר T=V-S, כך שבוודאי שתי הקבוצות זרות זו לזו. החלוקה מתבצעת כמובן על סמך המקור והיעד של הזרימה שעל פיהם יוחלט איזה חתך יהיה S, ואיזה יהיה T. כמו כן, בהתאמה לגרפים, אנחנו מבררים גם כאן על הקשתות החוצות, רק שברשת הזרימה, מה שיעניין אותנו הוא דווקא הזרימה העוברת בין שני החתכים. אנחנו סוכמים את כל הזרימות הקיימות בקשתות, כאשר אם יש לנו זרימה מנוגדת, אנחנו מחשבים אותה כזרימה שלילית, והתוצאה תהיה הזרימה נטו של החתך.

זרימה מקסימלית חתך מינימלי

השאיפה שלנו במושג החתכים, היא למצוא את החתך שמביא לנו את הזרימה המקסימלית. איך נמצא אחד כזה? נחפש את החתך המינימלי. מה ההיגיון? מאחר שאמרנו כבר שאנחנו מדברים פה על זרימות וכלי קיבול, אנחנו יכולים להבין בצורה הפשוטה ביותר, שגם אם יש לנו המון קשתות בעלי זרימה גבוהה, וכמה בודדות עם זרימה נמוכה, אם נהיה חייבים ללכת דרך קשת בעלת זרימה נמוכה, ייווצר לנו צוואר בקבוק, וכמות הזרימה תרד.

משפט הזמח"מ אומר כך:

אם f היא זרימה ברשת זרימה G(V,E) עם מקור s ובור t, אזי התנאים הבאים שקולים זה לזה:

1. **f היא זרימה מקסימלית ב-G.**
2. **הרשת השיורית G_f אינה מכילה שום מסלולי שיפור.**
3. **קיים חתך (S,T) ב-G שעבורו |f| = c(S,T).**

עד כאן המשפט. עכשיו מה הוא אומר? דבר ראשון – התנאים שקולים! אנחנו לוקחים כל תנאי ומוכיחים בעזרתו את התנאי הבא, ועושים זאת בצורה מעגלית, כלומר, אם אחד מהתנאים הללו לא מתקיים, אף אחד מהתנאים לא מתקיים.

עכשיו, נתחיל עם התנאי הראשון, שהוא דבר שעומד בפני עצמו – **יש לנו f שהוא זרימה מקסימלית ב-G.** הזרימה של הרשת מוגדרת לנו ואנחנו יוצאים מנקודת הנחה שהזרימה היא אכן מקסימלית, ועל ידי נתון זה נוכיח את התנאי השני –

$1 \Rightarrow 2$ – נניח בדרך השלילה, שאכן f מבטא את הזרימה המקסימלית ברשת, אבל ב-G_f קיים מסלול שיפור כלשהו p, אזי, סכום הזרימות יגדל וכבר לא יהיה f אלא יהיה f+f' (כאשר f' הוא מסלול הזרימה הנוסף). ובמקרה כזה, יש לנו סתירה להנחת היסוד.

$2 \Rightarrow 3$ – נניח כי אכן אין שום מסלולי שיפור ב-G_f, הכוונה היא שאין לנו בעצם מסלול שמוביל מ-s ל-t. אם היה לנו מסלול כזה, היינו ממשיכים לשפר עליו. עכשיו נגדיר את שתי הקבוצות באופן הבא – S יהיה כל הקדקודים אליהם ניתן להגיע מ-s, ו-T יהיה כל השאר, שכמובן מתחבר גם ל-t. מה שזה ייתן לנו שהזרימה בין שני החתכים, יכולים להיות רק אלו בעלי הזרימה המקסימלית. למה? כי כל מעבר אחר בתוך הקבוצות הללו סגור בתוך עצמו, ומה שעובר זה רק הזרימה שיוצאת מ-s, כי זה מה שהגדרנו, וברגע שהוא עובר ל-T, הוא יכול להמשיך הלאה עד הבור.

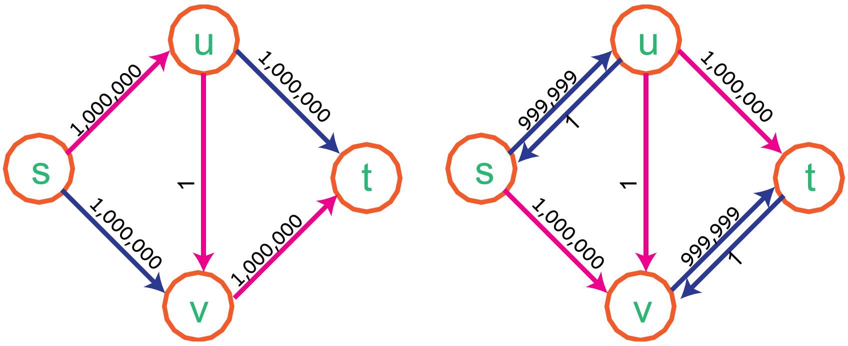
$1 \Rightarrow 3$ – ברגע שהגדרנו ב-3 חתך שהוא מוגדר כחתך המינימלי, אנחנו נשענים על טענה שאומרת שהזרימה המקסימלית חסומה מלמעלה על ידי כל החתכים שיהיו. וברגע שמצאנו חתך שחוסם עוד קצת אנחנו נשענים עליו להיות הזרימה המקסימלית.

אלגוריתם פורד פולקרסון הבסיסי

האלגוריתם הזה, הוא בעצם הרחבה של השיטה שראינו קודם – כלומר, זה בדיוק אותו דבר רק כתוב טיפה יותר פורמלי. המשמעות של האלגוריתם הוא להמשיך ולקחת עוד מסלולים, על מנת למצוא את החתך המינימלי של הרשת ולהגיע על ידי זה לזרימה המקסימלית:

```
Ford-Fulkerson(G,s,t)
  for each edge (u,v) ∈ E[G]                         // איפוס הזרימה ברשת
        f[u,v] = 0
        f[v,u] = 0
  while there exists a path p from s to t in the residual network Gf
        cf(p) = min {cf(u,v) : (u,v) is in p}        // מגדירים את הזרימה של המסלול לפי הקשת המינימלית
        for each edge (u,v) in p
          f[u,v] = f[u,v] + cf(p)                    // הוספת הזרימה מהמסלול לזרימה נטו
          f[f,u] = -f[u,v]
```

הבעיה של האלגוריתם הזה, למרות היותו מפורט יותר מהקודם, הוא בכך שלא מספיק שלא כתוב לנו כיצד למצוא את המסלול אותו אנחנו משפרים, אין בו קביעה ברורה של כמה לשפר בכל פעם. עד כמה זה קריטי? נגיד ויש לנו גרף שמסודר בצורה כמעט מושלמת, אבל יש בו קשת אחת שגורמת לזרימה לצנוח בצורה מטורפת. בחירה לא נכונה בקשת הזאת, תוביל לזרימה שהיא לא פחות ממבאסת. נדגים את העניין עם הגרף הבא:



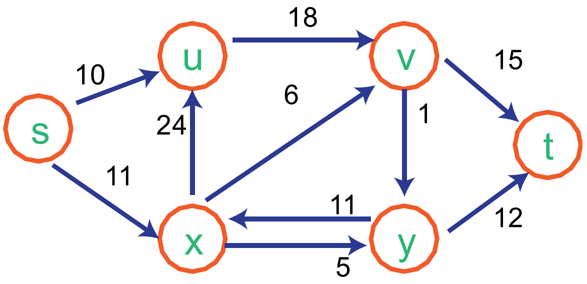
מימין יש לנו גרף באופן שתיארנו. בלי לחשוב יותר מידי, אנחנו יכולים לראות שאנחנו מעבירים פה 2 מיליון בקלות. אבל נגיד הגיע מישהו והחליט לבדוק מה קורה אם הוא מעביר מהצלע האמצעית. ואז ברשת השיורית, הוא מבין שהוא יכול להחזיר עוד פעם 1 על אותה קשת. וככה הוא מעביר 2 מיליון יחידות ב-2 מיליון איטרציות ומבזבז לכולנו את הזמן. ולכן יש לנו פה בעיה עם זמן הריצה שחסום על ידי $O(|f^*|)$ – שזה אומר שאנחנו חסומים על ידי כמות הזרימה שאנחנו אמורים להעביר. עכשיו, אמנם זה קבוע, אבל אם אתה יכול להרביץ את הכל בשתי איטרציות, ובמקום זה עושה 2 מיליון, אז עדיף לפרוש לדוקים.

אז הבנו שצריך למצוא שיטה יעילה ועקבית למעבר על הזרימות. מה יהיה יעיל?

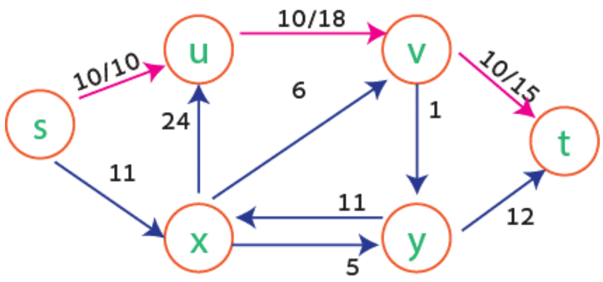
אלגוריתם אדמונדס-קארפ

למעשה אין שום הבדל בין פורד-פולקרסון לאדמונדס-קארפ, מלבד הערה קטנה. את שורת הלולאה while, לא עוברים "לפי העין", אלא מריצים על הגרף חיפוש לרוחב (BFS), אותו למדנו במבנה נתונים ב', שיוציא לנו את המסלולים הנפרשים מהנקודה s לכיוון t (כי הרי אמרנו, שכל קדקוד יגיע בסוף לבור), ונבחר את המסלולים שנוצרו החל מהדרגה הנמוכה ביותר. אם יש לנו מסלול שעושה את הדרך עם שתי קשתות בלבד, נשתמש קודם בו, ואחרי זה נתמודד עם השאר בסדר עולה. כמובן, שאין זה מחייב שהמסלול הקצר ביותר יהיה גם היעיל ביותר, אבל זה בהחלט יוריד את הסבירות לפגוש פתאום את הקשת המבאסת בסיבוב שמורידה את מסלול הזרימה ל-1.

עכשיו אפשר לעשות דוגמת הרצה על הגרף –

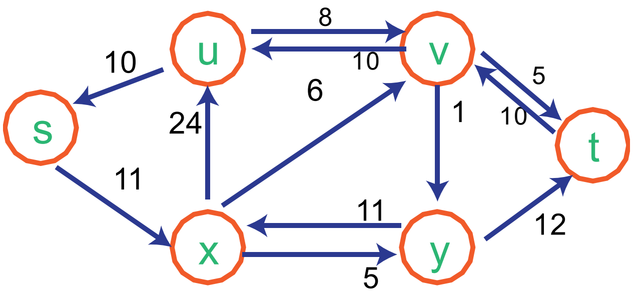


זה גרף שהוא טיפה שונה ממה שהצגנו בהתחלה, אבל הוא מספק את הסחורה. ראשית נריץ עליו BFS, ונוציא את כל המסלולים השונים שאנחנו יכולים להוציא ממנו. עכשיו אנחנו מתחילים להריץ את האלגוריתם של פורד פולקרסון על פי סדר הדרגה ובאופן לקסיקוגרפי. כמובן שכאשר אנחנו מקבלים את הגרף כמו שהוא, אין לנו צורך ברשת שיורית, מאחר וכל עוד אין זרימה, כל הגרף נשאר בדיוק אותו דבר. נתחיל:

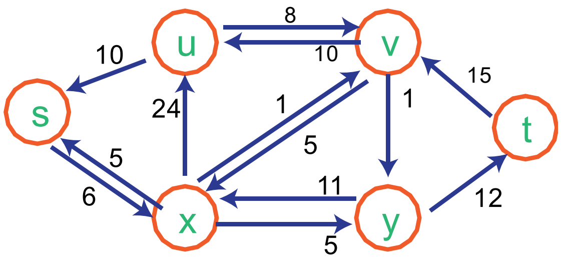


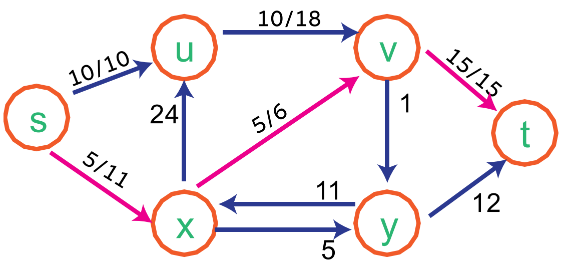
המסלול הראשון שנבחר(s→u→v→t) מזרים לנו 10, מאחר והמינימום של המסלול הוא ב(s,u). הזרימה הזאת ממשיכה גם הלאה עד שמגיעים לבור. עכשיו אנחנו צריכים לעבור ולעדכן את הרשת השיורית. נבפיל כל קדקוד שעברנו עליו ולא רווי עד הסוף, ונסדר את הערכים, ואת הקשת (s,u) נשאיר ב-10 מאחר והיא רוויה (מוזרמת עד הסוף), אבל נשנה לה את הכיוון, כי עכשיו אנחנו יכולים להשתמש רק בזרימה הסימטרית שלו.

הרשת השיורית תיראה עכשיו כבכה:



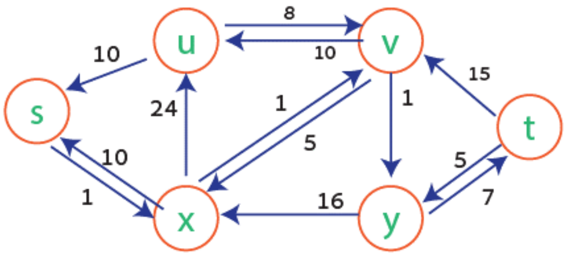
ברגע יש לנו עוד שני מסלולים באורך 3 קשתות, נבדוק את הראשונה שבהן:

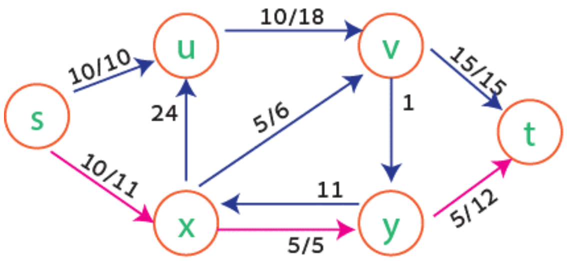




משמאל מופיעה לנו רשת הזרימה, בה בחרנו את המסלול s→x→v→t. מאחר וקודם הזרמנו ב-(v,t) 10/15, עכשיו הוא זה שמגביל לנו את הזרימה, ולא (x,v) שהקיבול שלו יחסית נמוך. נזרים כמה שאפשר, ונעדכן את הרשת השיורית. עכשיו נראה שאת הקשת (v,t) אנחנו מעדכנים באופן מיוחד. מאחר שהעברנו את המקסימום האפשרי לאותה קשת, אנחנו משתמשים ברשת השיורית בקשת החוזרת בלבד, מה שבעצם מסמן לנו שיש לנו התקדמות – ברגע שברשת השיורית לא יהיו קשתות נכנסות, נדע שסיימנו את האלגוריתם.

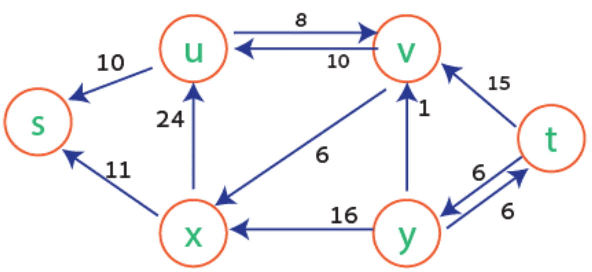
נעבור הלאה:

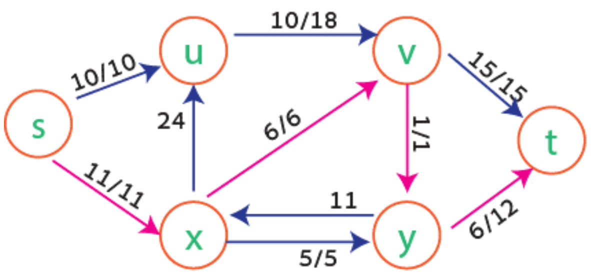




למעשה עכשיו, סיימנו את המסלולים בדרגת 3.

מבחינת הקשתות הנכנסות ל-t, אנחנו יכולים למצוא את (y,t) שיש לו עוד 7 יחידות להעביר, והוא גם היה עושה זאת בשמחה, הבעיה היא שלקדקוד y, נגמרו כמעט כל המקורות, מלבד (v,y) שגם בכה הקיבו שלו הוא 1. אבל אל ייאוש, אפילו אם אנחנו יכולים להעביר רק אחד, אנחנו נעשה זאת! השאלה היא באיה דרך. אנחנו מזהים הזדמנות מעולה, שהיא לגמרי במקרה גם מה שאנחנו צריכים לעשות לפי האלגוריתם, ועושים את הדבר הבא –





תבלס, גם אם היינו רוצים, אין לנו לאן להתקדם, העברנו מהמקור ליעד בדיוק 21 יחידות של מה שזה לא יהיה שרצינו. ברגע שהגענו למצב כמו ברשת השיורית הנוכחית, בו החיצים מופנים לאותו כיוון, ואין לנו יכולת להמשיך עבוד עליה, לטוב או לרע – עלינו לעצור.

לעיתים נישאל על החתך המינימלי של רשת לאחר ביצוע ריצה של אדמונדס-קארפ. עד כמה שזכור לי, זה לא נאמר באופן מפורש בכיתה, אבל כך מקובלנו (בשם שרה פרידמן), שעל מנת למצוא את החתך, אנחנו מריצים BFS על הרשת השיורית הסופית החל מ-s. כל קדקוד שאנחנו עוברים עליו שייך לקבוצה S, וקדקודים שאנחנו לא מסוגלים להגיע אליהם, יהיו שייכים לקבוצה T. בדוגמה שלנו, זה די קל – אי אפשר להמשיך מהקדקוד s, ואם נסתכל הזרימה היוצאת מהקדקוד היא במשקל 21, שזה באמת הזרימה שנכנסת גם ל-t, ולכן החתך יוגדר להיות כך: S = {s}, T = {u,v,x,y,t}. כמובן ששאלות אחרות יהיו יותר מסובבות, אבל זו השיטה לעבוד.

ניתוח זמן ריצה

אלגוריתם אדמונדס-קארפ משתמש בBFS שחסום מלמעלה על ידי O(VE) – במקרה בו יש לנו גרף עמוס, נצטרך לעבור על כל הקשתות שעל כל הקדקודים. בנוסף, הוא עובר אחר כך על כל הקשתות מה שמכפיל את זמן הריצה בעוד V, ומעלה את הכל ל – $O(VE^2)$.

בעיית הזיווג המקסימלי בגרף דו צדדי

אחד הדברים המעניינים בבעיות רשת זרימה, הוא שעל אף שהבעיה נדמית כמשהו יחסית מאוד מצומצם, ניתן להשליך אותו על הרבה סוגי בעיות אחרות בתחום הקומבינטוריקה, כל שאנחנו צריכים הוא לעשות מעין המרה (רדוקציה) לבעיות וליצור אותם מחדש כבעיית רשת זרימה, ועל ידי זה להצליח לפתור אותם בקלות. יחסית.

מציאת הזיווג המקסימלי היא בעיה קלאסית שמשתמשים בה- נתון לנו גרף לא-מכוון דו צדדי G(V,E), כל חלק של הגרף מסומן בתור L לשמאל ו-R לימין, ככה סתם רנדומלי. כל הקשתות בגרף עוברות רק מצד אחד לשני ללא כל קשתות פנימיות בתוך הקבוצות עצמן. כמובן, שיכול להיות שכל קדקוד מחובר לצד השני עם יותר מקשת אחת (המכונה אצלנו many to many). אנחנו צריכים למצוא מהו הזיווג המקסימלי – כלומר, מה מקסימום החיבורים הבודדים שאנחנו יכולים לעשות בין שני הצדדים. ה"זיווג" M (Matching) שאנחנו דורשים, הוא שברגע שיש שני קדקודים שמוגדרים כמחוברים, אין אף קדקוד אחר שיכול להתחבר אליהם שוב.

היישום של בעיה זו הוא לא רק זיווגים, אלא בעצם כל חלוקה כלשהי של משימות למכונות וכדו', אנחנו צריכים לבדוק קודם כל מה המקסימום שאנחנו יכולים להגיע.

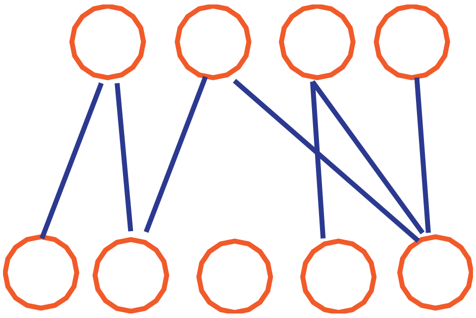
לכאורה, זה דבר שנדמה כאילו אינו קשור לרשתות זרימה, אבל אם נתעמק בזה, נשאל שאלה אחרת – האם רשת זרימה חייבת לעבור ממקור ליעד יחידים? לכאורה כן, אבל לא הגדרנו את זה באופן מוחלט. מה קורה אם יש לנו יותר ממקור אחד או יותר מיעד אחד, האם אנחנו עדיין מסוגלים להתמודד עם זה?

התשובה היא כן, והראינו את זה קודם במקורות ובורות מרובים, דרך הפיתרון היא אחת שכבר פגשנו בחלק מהקורסים השונים – בניית עזר.

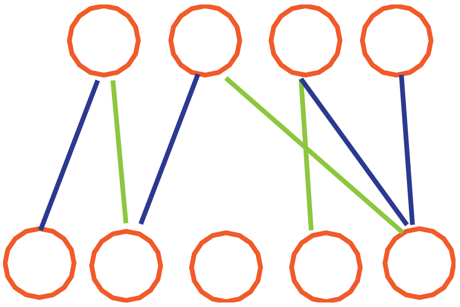
אם נתונים לנו מספר מקורות שונים, כמובן שלא נוכל פשוט להזרים הכל במכה ולראות מה קורה, מאחר ואנחנו נתנגש ונסבך את עצמנו יותר על המידה, אבל אם נגדיר קדקוד מקור דמיוני 's, נוכל לשים אותו לפני כל שאר המקורות, ואז לחבר אותו אל כולם, ולהתחיל את הזרימה ממקור יחיד. באופן דומה אנחנו יכולים גם ליצור בור דמיוני 't, ולהזרים אליו את כל המקורות השונים.

לאחר שחיברנו את הקדקודים של בניית העזר, אנחנו צריכים להגדיר גם איך אנחנו פותרים בעזרתם את השאלה. בשביל זה אנחנו "נקבל" (מלשון "למשקל") את הרשת באופן הבא – כל קשת מקורית העוברת בין שני צדדי הגרף הדו-צדדי תקבל קיבול מקסימלי של 1. כל קשת העוברת לבניית העזר, תקבל גם היא ערך 1. איך זה יעזור לנו? ברגע שקשת תיתפס על ידי שני קדקודים, לא יהיה אפשרי יותר להשתמש בה מאחר והיא רוויה. הקשתות החדשות של בניית העזר, ברגע שהם יוגדרו כ-1, יוכלו להזרים רק יחידה 1 בכל פעם, וכך להביא לנו בעצם את הדרוש לנו (הוכחה לבעיה דומה לזה, נמצאת בשאלות ההרחבה, וכדאי להסתכל למקרה שידחקו אתכם לפינה עם הוכחה כזו).

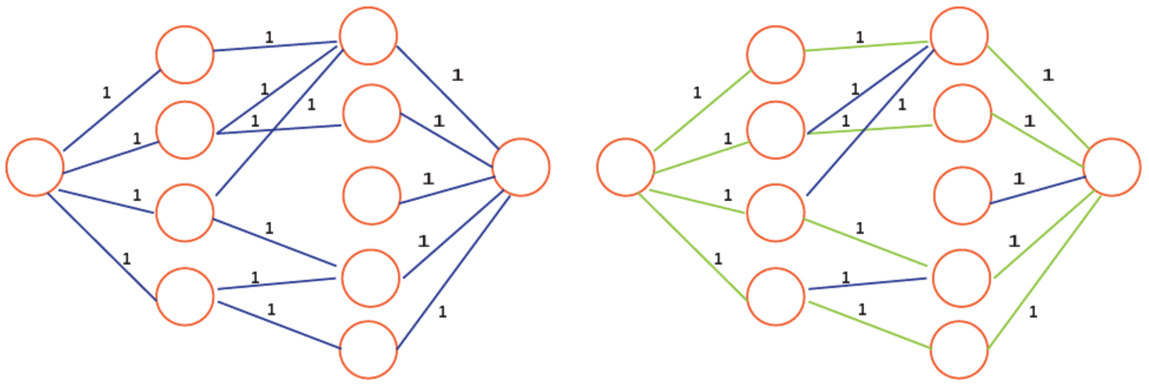
נתסכל על תצורת הגרף הבאה:



לא קשה לראות פה שהזיווג המקסימלי הוא 4, אבל אם יבוא אותו אחד שהזרים 2 מיליון יחידות בבודדת, הוא עוד עלול לייצר זיווג פחות מזה באופן הבא:



על ידי בחירה מוטעית, נגיע למקומות לא רצויים. את הבנייה נראה באופן הבא (אני מסובב את זה לצורך הנוחות)-



רשת הזרימה נתנה לנו את האפשרות לפתור את זה בצורה יעילה ומוכחת.

שאלות הרחבה

נתונה רשת זרימה G(V,E) שבה הקיבולים על הקשתות הם מספרים שלמים וזוגיים, מלבד קשת אחת (u,v) שהקיבול שלה אי-זוגי. כמו כן, נתון לנו שהזרימה המקסימלית ברשת היא אי-זוגית.

האם הקשת (u,v) רוויה?

באופן אינטואיטיבי, ברור לנו שהמשפט הזה נכון. אם יש לנו זרימה שעוברת רק בקיבולים זוגיים, אין לנו שום סיבה לחשוב שפתאום בסוף יקפוץ לנו מספר אי-זוגי. אבל איך מוכיחים את זה בצורה פורמלית?

כמו שאמרנו קודם, אנחנו יכולים להתבסס על הכלל של "זרימה מקסימלית – חתך מינימלי". לכל גרף יש מספר חתכים שונים. כל חתך משפר במקצת את הזרימה או לא משפיע עליה בכלל. בחתך המינימלי של הרשת ברור לנו שאנחנו נקבל שכל הקשתות הינם רוויות, ואנחנו כבר יכולים להגיד שבכבל הנראה גם הקשת האי זוגית נמצאת שם. אבל אנחנו מחפשים משהו שהוא יותר מ"ככל הנראה". מה שנעשה הוא הוכחה בשלילה – נניח שהקשת המדוברת, האי זוגית, אינה רוויה, ונוריד את ערך הקיבול שלה ב-1. אם אכן הקשת לא נמצאת על החתך המינימלי אין סיבה שנרגיש את זה. עכשיו, אם הורדנו ערך אי-זוגי ב-1, הוא נהפך להיות זוגי. עכשיו, ברגע ששינינו את הקשת היחידה שאינה זוגית להיות זוגית, כל הקשתות אמורות להיות זוגיות.

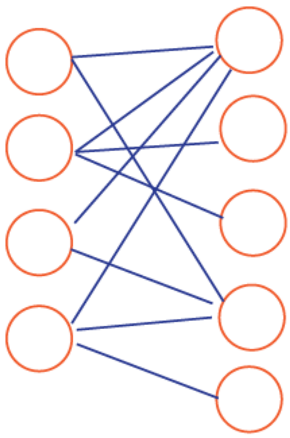
ידוע כי הזרימה אמורה להיות מתאימה לחתך, אבל אם החתך הוא בוודאות זוגי, גם הזרימה היא זוגית, ויש כאן סתירה.

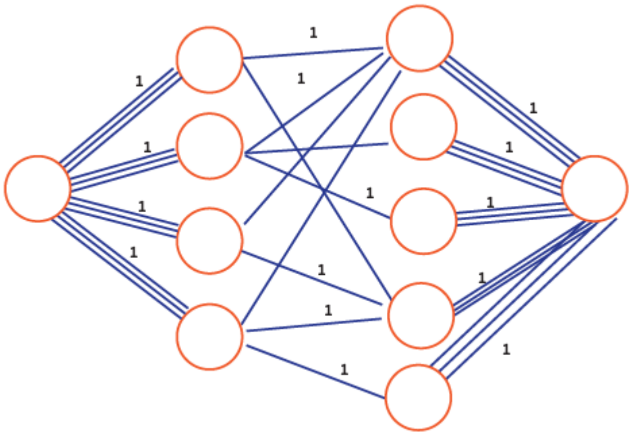
נתון גרף לא מכוון דו-צדדי G(V,E).
תאר אלגוריתם יעיל ככל האפשר המוצא ב-G תת גרף בעל מספר מירבי של קשתות, שבו לכל קדקוד דרגה לכל היותר ברמה 3.

כמובן שאנחנו מדברים פה על בעיה שהיא דומה לזיווגים. ישנו ענף בתורת הגרפים הנקרא Bipartite Graphs, הגרפים הדו-צדדים. דיברנו עליהם בלוגיקה, והזכרנו גם את העניין, אי שם בפרק. הרעיון הוא למתוח את הגרף בכבה שהקשתות יעברו רק בין שתי הקבוצות, הימנית והשמאלית, ולא בתוך הקבוצות עצמם. בבעיית הזיווגים הרגילה, רצינו שתעבור קשת בודדת בין שני קדקודים, ועכשיו אנחנו דורשים שיהיו מקסימום קשתות עד לרמה 3.

לבעיה זאת אנחנו עושים רדוקציה, ומתאימים אותה לבעיה איתה אנחנו יכולים להתמודד. את כל זה כבר עשינו קודם, וגם פה נעשה בדרך דומה, אבל נצטרך לדאוג גם להוכיח את נכונות הטענה שאנחנו פותרים את הבעיה בעזרת הגרפים.

איך נהפוך את הגרף הדו-צדדי לרשת זרימה? נגדיר קדקודים דמיוניים בבניית עזר s,t ומכל אחד מהם נעביר שלוש קשתות לכל קדקוד מהגרף, ולכל אחת מהן נגדיר קיבול 1, וגם לקשתות הפנימיות נגדיר קיבול של 1, באופן הבא:





בעת מתחילים את השגרה של הזרמה ברשת, ומה שייצבע אכן יקיים את התנאים.

הערה: אם יש שאלה כזאת במבחן, אין צורך לכתוב פסאודו אלגוריתם, אלא לתאר מה עושים, ולצייר בצורה כללית איזה דוגמא. מבחינת ההוכחה, זה כבר צריך להיעשות בצורה מלאה כמו שנעשה תיכף.

בעת, יש לנו להוכיח שני דברים שנגזרים מהבעיה שהוצעה לנו: 1. מדובר בתת גרף בעל מספר מירבי של קשתות. 2. הדרגה המקסימלית של הקדקודים היא 3.

נתחיל בלהוכיח שאכן <u>דרגת המקסימום היא 3</u>.

קודם כל אינטואיטיבית – כשאנחנו מזרימים זרימה מקסימלית ברשת שיצרנו, מה המקסימום שיכול להכנס לכל קדקוד? 3. ולפי כל החוקים שאמרנו, זה גם הקיבול שיכול לצאת. ומאחר ויש לנו שלוש קשתות בדיוק שמזרימות, אם ימצאו שלוש קשתות להזרים אליהם הקדקוד זה יספק, ואם לא, אז גם הזרימה הנכנסת לקדקוד תהיה רק כמה שנצליח להרים הלאה, אבל בכל אופן, לא יכול להיות שנזרים דרך הקדקוד הנ"ל יותר מ-3.

עכשיו פורמלית – נוכיח שדרגת המקסימום היא 3. לכל קדקוד ב-L (החלק השמאלי, כאמור) יש 3 קשתות נכנסות עם קיבול כולל של 3. ולכן הזרם המקסימלי שיכול לצאת הוא 3, כלומר **לכל היותר** 3 קשתות אדומות. הווה אומר – דרגת המקסימום ב-L היא 3.

לכל קדקוד ב-R יש 3 קשתות יוצאות עם קיבול כולל של 3. ולכן הרימה המקסימלית היוצאת היא 3, ומאחר שגם בזרימה המקסימלית יכול להיות לנו לא יותר מ-3, אז דרגת המקסימום הוא 3. מש"ל

הוכחנו שהמקסימום הוא 3. אבל איך נוכיח <u>שאכן מדובר בתת גרף בעל מספר מירבי של קשתות</u>.

במובן שנוכיח זאת בשלילה.

נניח בשלילה כי יש פתרון אחר המכיל יותר קשתות (כלומר קיבענו את המקסימום, אבל לא את המינימום זרימה שתהיה). במקרה כזה, הכוונה היא שיש לנו חתך מינימלי שהוא גדול יותר, והזרימה המקסימלית עוברת בו. וזה סתירה לכך שמצאנו את הזרימה המקסימלית ברשת. (כלומר, אם קיבענו את המקסימום זרימה באופן שהוא חד משמעי, אז בוודאי שהזרימה שנקבל היא בדיוק מה שרצינו – זרימה מירבית עד דרגה 3) מש"ל.

הערה: את בניית העזר לא חייבים לעשות עם שלוש קשתות של קיבול 1, אלא ניתן להעביר קשת בודדת עם קיבול 3. הסיבה היחידה שאבירם עשה כך, כי הוא אמר שיותר נוח להבין ככה כשרואים את הבניה. ברמת המבחן – אם עושים את זה עם קשת אחת זה לגמרי עונה, ואם רוצים לענות בצורה יותר ברורה ולנפנף קצת בידיים, אפשר לעשות 3 קשתות.

מחלקות סיבוכיות

חלק זה של הקורס, הוא לכאורה נפרד מכל מה שלמדנו עד עכשיו, אך במבט מעמיק יותר הוא תלוי בכל מה שלמדנו, ונשתמש בכל הידע שרכשנו בחלקי ניתוח האלגוריתמים השונים. בכל אלגוריתם שהצענו, בדקנו וניסינו לשכלל את זמני הריצה, כך שירדו מזמני ריצה מעריכיים לזמן ריצה פולינומיים. בעוד שבקורסים קודמים כמו מבנה נתונים, חיפשנו זמני ריצה בקירוב ליניארי, ראינו שאלגוריתמים שפועלים על קלטים ושאלות מסובבות יותר, ניתנות להכרעה בזמן של n^2 או אפילו n^3, ומה שהיה נחשב לזמן ריצה גבוה מידי, הפך לזמן סביר.

(נזכיר רק דבר שדיברנו עליו בלא-מעט קורסים קודמים – עד כה מחשבים משתכללים במרוצת השנים ונעשים מהירים יותר, ואין מה להשוות מחשב של היום אל מול מחשב של לפני 20 שנה, כל השינויים האלו נעשו בסדר גודל, ואם ניקח אלגוריתם שהוא לא יעיל, גם המחשבים המהירים ביותר יקרטעו בדיוק באותו אופן כמו הישנים)

בעת אנחנו מסווגים את האלגוריתמים השונים למחלקות סיבוכיות. כל מחלקה מייצגת מספר (אינסופי) של אלגוריתמים, שהמשותף להם הוא זמן הריצה ביחס לקלט. כלומר, אם יש לנו קלט באורך של n ביטים, נרצה שזמן הריצה יהיה למשל n^2. מחלקה זו נקראת מחלקת הסיבוכיות P.

כמובן, שהכל היה יותר נוח, אם היינו יכולים למצוא אלגוריתם פולינומי עבור כל בעיה שתצוץ. הבעיה היא, שאין לנו, או לפחות עד היום לחלק מהבעיות לא נמצא בכלל אלגוריתם פולינומי. למשל, "בעיית הסוכן הנוסע" המוכרת היא בעיה מהסוג NP-קשה (שאנחנו בכלל לא יודעים מה זה, אבל זה לא נשמע מבטיח). עיקר הבעיה היא – האם סוכן מכירות שצריך לנסוע למספר נקודות שונות על המפה, שחלקן מקושרות אחת לשניה (גרף ממושקל), יכול למצוא מסלול אופטימלי שיעבור בכל הערים ויחזור לעיר המקור בזמן כמה שיותר קצר. אנחנו יודעים להגיד שהבעיה אינה פולינומית, אבל אנחנו לא מסוגלים אפילו לקבוע חסם תחתון כלשהו שאנחנו יכולים להגיד בוודאות שהאלגוריתם לא יעבור אותו, החסם שאנחנו מחפשים למעשה ייתן לנו את התשובה לשאלה האם $P \neq NP$, שרוב החוקרים סבורים שזה אכן נכון אך ללא חסם תחתון שיבריע שהם אכן לא שווים, לא ניתן לקחת את זה בקביעה מוחלטת.

ישנם גם אלגוריתמים כמו למשל "בעיית העצירה" של אלן טיורינג מ-1936 המציעה את הבעייה הבאה – האם ניתן לבנות מכונה M, שבעבור **כל** תוכנית, **ובכל** קלט שלא נכניס ביחד למכונה, נוכל לקבל תשובה האם התכנית תעצור או לא. נגיד שנריץ את המכונה על מספר תוכניות, יכל להיות שבהתחלה זה יעצור, אבל הוא יתחיל להתקע עבור תוכנית וקלט מסוימים, עד שנחליט לעצור אותו בעצמנו. עד כמה שזה יישמע מגוחך, האם אנחנו יכולים להגיד בוודאות שאנחנו עצרנו את התכנית, או שמא בדיוק באותו רגע המכונה נעצרה מעצמה? בגדול, טיורינג הציג באופן מתמטי ובצורה שלא משתמעת לשני פנים שזו בעיה שלא ניתנת לחישוב.

בפרק זה אנחנו מתעסקים בעיקר בבעיות NP שלמות, שמעמדן אינו ידוע. מה הכוונה "אינו ידוע"? מעבר לשאלת $P \neq NP$, כל האלגוריתמים המוגדרים כשייכים למחלקה ה-NP שלמה, קשורים אחד לשני באופן, שאם מישהו ימצא אפילו לאחד מהאלגוריתמים האלה פתרון פולינומי, ישר כל מה שמקושר למחלקה הזו יוגדר באלגוריתם פולינומי, ולהיפך – אם נמצא חסם תחתון לאחד מהאלגוריתמים שהוא אינו פולינומי – הכל יוכר בלא-שייך לP. ונחזור לזה בהמשך.

הערה קטנה שחשובה להפנמה – אמנם אנחנו מדברים על זמנים פולינומים שהרבה יותר מוצלחים ממעריכיים, אבל אם ניקח את n^{100} ולעומתו את 2^n אנחנו יכולים להציב ש $n=3$, ואז דווקא המעריכי יהיה

הרבה יותר סביר? אבל כמו שכבר אמרנו, האלגוריתמים הפולינומיים הגדולים ביותר שראינו הם n^4, ולדבר על חזקה של 100, או מספר גבוה אחר, זה בעצם עשרות לולאות שתקועות אחת בשניה, וברור שאף אחד שפוי בדעתו לא יעשה דבר כזה (ככה אנחנו מקווים לפחות), ולכן אנחנו מתייחסים למקרים הסטנדרטיים האלה.

נגדיר עכשיו כמה מושגים פורמלים, על מנת שנוכל לתאר את הבעיות השונות:

בעיה מופשטת (Q): תיאור של בעיה בצורת קשר בינארי של מופעים I (Instances) ופתרונות S (Solutions). למשל, עבור בעית מסלולים קצרים שראינו בעבר, נגדיר את SHORTEST-PATH כבעיה מופשטת. עבור המופעים השונים של הבעיה, יש לנו אינסוף גרפים על פי הגדרתם G(V,E), כל גרף שמגיע עם סט מסוים של קדקודים וקשתות שונות מגדיר לנו מופע חדש לבעיה, וכל שני קדקודים מהווים העלאת השאלה מחדש. הפתרון עבור כל מופע שכזה, יהיה רצף קדקודים/קשתות שיגדירו לנו מסלול. כמובן, שיכולים להיות עבור מופע אחד של גרפים שני קשרים שונים או יותר, במידה ויש לנו יותר מפיתרון אחד. בהתאמה, יכול להיות שנקבל קבוצה ריקה – במידה ואין מסלול קצר בין הקדקודים הרצויים. הגדרה של בעיה בזאת היא מאוד רחבה, וקצת קשה לקבל עליה מושג או לעבוד איתה.

בעיית הכרעה: בלל הבעיות שהתשובה שלה מסתכמת בבן/לא.
הבעיות המופשטות הביאו לנו עבור כל שאלה נתונים שהם הרבה יותר רחבים מאשר מה שאנחנו דורשים פה. כמובן, שעבור קבוצה ריקה, שתי הבעיות יוכרעו אותו דבר, אבל אם אני מחפש רק תשובה לשאלה "האם יש מסלול בין שני קדקודים?", לא מעניינים אותי כל המופעים השונים, וכל הקדקודים האפשריים, אני מחפש תשובה בוליאנית, יש מסלול? תעצור, תגיד "כן" נמשיך הלאה. מבחינה פורמלית, אנחנו אומרים שאנחנו ממפים את כל הפתרונות לקבוצה {0,1}. מבחינת תשובה שתהיה לי קונקרטית יותר לבעיה המופשטת של מציאת מסלול קצר, אנחנו יכולים להמיר את השאלה לצורה הכרעתית – נגדיר את המופע של הבעיה כ $i = (G, u, v, k)$, כאשר k יהווה את החסם העליון שעליו אנחנו נשאל את השאלה "האם יש מסלול קצר יותר מ-k?", כאשר על מנת למצוא את המסלול הקצר ביותר, נשאל את השאלה הזאת שוב ושוב ונצמצם את הk למינימום האפשרי.

בעיית אופטימיזציה: בעיות שמחפשות מקסימום או מינימום של טווח מסוים. למעשה, הניסוח של המסלול הקצר ביותר, הוא בעייה שכזו, ודרך הטיפול בה היא כמו שתיארנו.

מבחינת מחלקת השלמות ב-NP, אין הבדל מהותי מבחינתנו בין הבעיות השונות, אך נעדיף תמיד להגיע לבעיות הכרעה שהתשובה שלהם היא תמיד כן/לא, ואם נוכל לפתור בעיית הכרעה כלשהי, המרחק בינה לבין פתרון אופטימיזציה, הוא פשוט כמה איטרציות (כלומר, המינימום יהיה פשוט לצמצם עד שנגיע למספר הנמוך ביותר שאנחנו מכריעים שהוא מקיים, וכן להיפך במקסימום).

קידוד

באשר מחשב מקבל בעייה, הוא לא מקבל אותה באופן מילולי. אם ננסה להכניס "יהי גרף שושקה" זה לא אומר כלום למכונה. הדיבור מול מחשב הוא בשפה בינארית {0,1}, וכל הבעיות מגיעות מותאמות במחרוזת בינארית איתה ניתן לעבוד. המעבר מהבעיה לאופן מוכר למחשב (לאוו דווקא בינארית), נקרא קידוד e (Encoding), והוא יוצר לנו את הבעייה הקונקרטית מולה אנחנו מתמודדים. כמובן שישנם קידודים שהם לא יעילים, כמו למשל קידוד אונארי, שמבטא כל מספר במחרוזת אחדות מתאימה (1111111111 = 10) מה שהופך את הקידוד ללא יעיל בהחלט, אבל אנחנו מדברים על קידודים של בינארי, HEX ובדו' שהמעבר מאחד לשני הוא יחסית פשוט.
באשר מחשב מקבל מופע i של בעיה, אנחנו מגדירים את ה-n עליו אנחנו מדברים בתור $|i| = n$, וזמן

הפיתרון מגדיר לנו את המחלקה אליה אנחנו משתייכים. כלומר, בעיה שמוגדרת כסיבוכיות פולינומית, תפתור את הבעייה בזמן $O(n^k)$ ביחס לאורך הקלט.

כמובן, שהבעיה בתלות בקידודים, הוא ששימוש בקידוד לא נכון (כמו האונארי) עלולה להעביר לנו בעיה בצורה שרק המעבר עליה יהיה ארוך ולא יעיל – אם בעיה אונארית מוצגת לנו ב-n תווים, בעיה באורך דומה אך בייצוג בינארי תוכל לייצג לנו בעיה שהיא הרבה יותר ארוכה, למעשה בעיה שהיא $\Theta(2^n)$ [11]. אבל אנחנו יוצאים מנקודת הנחה שאנחנו מדברים על ייצוג בינארי (או דומה לו), שהוא יעיל מבחינת הריצה עליו.

עכשיו אנחנו יכולים להגדיר את המחלקה P באופן הבא – הפונקציה הבינארית $f:\{0,1\}^* \rightarrow \{0,1\}^*$ המוגדרת, ניתנת לחישוב בזמן פולינומיאלי, אם קיים אלגוריתם זמן-פולינומיאלי A, אשר בהינתן קלט $x\in\{0,1\}^*$ מופק פלט $f(x)$. כמו כן, אנחנו מגדירים קשירות פולינומיאלית בין שתי פונקציות באופן הבא, אם יש לנו שתי פונקציות המסוגלות לעבד קלט מסוים ולהמיר אותו לתצורה אחרת וחזור (כלומר f_{12}, f_{21}, שממירות מאחת לשניה) תחת זמן פולינומיאלי, ונתונות לנו שתי בעיות בשני קידודים שונים כך שאם נפעיל את הפונקציות עליהם, נוכל לעבור מקידוד אחד למשנהו, כלומר אם נפעיל על e1 את הפונקציה f_{12} נגיע ל-e2 ($f_{12}(e1) = e2$) וכנ"ל בחזרה, אזי הפונקציות קשורות פולינומיאליות. מה זה משנה לנו? מאחר ואם אנחנו מסוגלים לפתור את אחת הבעיות עצמן תחת זמן פולינומיאלי, אנחנו יכולים לומר שגם הבעיה השניה נפתרת בזמן פולינומיאלי (גם אם אין לנו את הפתרון עצמו).

למה 36.1

תהי Q בעיית הכרעה מופשטת על קבוצת מופעים I, ויהיו e_1, e_2 קידודים קשורים פולינומיאלית על I. אזי $e_1(Q)\in P$ אם"ם $e_2(Q)\in P$.

כלומר, הלמה הזאת מנסה לומר שאין לנו הבדל מהותי בין שני קידודים שונים של אותה בעיה, כל עוד אפשר לקשור אותה בצורה פולינומיאלית – אם ניתן לעבור מקידוד אחד לשני בזמן פולינומיאלי, אז כל קידוד שלא יהיה לבעיה הולך ביחד, כלומר, לא יכול להיות שבבסיס 10 נמצא פתרון, אבל בבסיס 2 לא נצליח למצוא פתרון.

<u>הוכחה</u>: נוכיח רק כיוון אחד, מכיוון שאנחנו מסתמכים על הסימטריות בין שני הקידודים השונים. נניח שניתן לפתור בעיה e1(Q) (בעיה Q כלשהי, תחת קידוד מסוים) תחת זמן של $O(n^k)$, עבור איזשהו קבוע k. נוסף על כך, נניח שכל מופע i של הבעיה ניתן להמיר מהקידוד e2(i) ישירות לe1(i) תחת החסם $O(n^c)$, שגם c הוא קבוע כלשהו. אזי על מנת לפתור את הבעיה e2(Q) כל שנצטרך הוא להמיר לe1 ואז לפתור בקידוד הנ"ל. הזמן הכולל של זה יהיה $O((n^c)^k)$, ומאחר ששתי החזקות הם קבועים, ההרכבה שלהם תישאר פולינומיאלית. מש"ל.

מסגרת של שפות פורמליות

תבלס, עשינו את זה כבר קודם, בלי ממש להכריז על זה. אבל מאחר ואנחנו מתייחסים בעיקר לבעיות הכרעה של כן/לא, מאוד נוח לנו לעבוד עם שפות פורמליות. נגדיר שפה, נתאר מה נכנס אליה, ואז נוכל לדעת האם מילים (בעיות) מסוימות נכנסות תחת השפה או לא.

---

[11] מאחר ואורך ייצוג בינארי של מספר n כלשהו שווה n = logk (בקירוב), אז ההמרה של הרצף לצד השני היא הרבה יותר ארוכה.

נעבור מהר על המסגרת שנשתמש בה, בהנחה שכולנו כבר מכירים את הסיפור הזה מאוטומטים:

∑- אלפבית – קבוצה סופית של סימנים – לדוגמא: הא"ב העברי, השפה הבינארית {0,1} וכו'.

L – שפה – קבוצה כלשהי של מחרוזות מתחת שפה מסויימת, למשל – {...L = {10, 101, 11 הם שפה תחת האלפבית הבינארי.

ε – מחרוזת ריקה.

ϕ - שפה ריקה.

∑* - סימון כל המחרוזות האפשריות תחת הא"ב המוגדר.

על השפות ניתן לבצע פעולות של איחוד, חיתוך, משלים (כל מחרוזות הא"ב שלא נמצאות בשפה) ועוד.

מבחינת בעיות ההכרעה שהצגנו, נגדיר את כולן בשייכות לא"ב הבינארי {1 ,0} = ∑, ונגדיר את השפה ככל המופעים השייכים לא"ב המחזירים 1 – {L = {x∈∑* : Q(x) = 1. למשל, השפה המקיימת את בעיית המסלול הקצר תתואר באופן הבא:

PATH({<G,u,v,k>: G(V,E) הוא גרף בלתי מכוון,
u,v ∈ V
k ≥ 0 וכן הוא מספר שלם
k קיים מסלול בגרף בין שני הקדקודים שאורכו לכל היותר}

ההגדרה של השפה בצורה הזאת מגדירה לנו באופן הרבה יותר מדויק מה נכנס לשפה ומה לא, כאשר מה שמוגדר כמקבל, הוא כל הבעיות שבעבורן האלגוריתם A יוציא לנו כפלט תוצאה 1 – כלומר, יש לנו מסלול בגרף שקטן מאורך k המוגדר.

מציאה של הכרעה כזו היא בוודאות תחת זמן פולינומיאלי – מריצים BFS על הגרף, בודקים את שני הקדקודים, ואם אנחנו מוצאים שזה מתחת לרף שהצבנו, אנחנו מאושרים. ואם לא, אז כמו בחיים נוריד את הרף עוד קצת או משהו בסגנון.

תחת המסגרת של השפות הפורמליות, אנחנו יכולים עכשיו להגדיר גם את המחלקות באופן דומה:

עבור מחלקת הסיבוכיות P הפולינומיאלית, נגדיר אותה כשפה המתקבלת תחת זמן פלינומיאלי, באופן הבא:

P ={ L ⊆ (0,1)* : קיים אלגוריתם המכריע את שפת הבעייה תחת זמן פולינומיאלי}

כדאי לשים לב שהשפה L המוגדרת פה, היא בדומה לשפה PATH שהגדרנו, ומייצגת אוסף מופעי בעיות.

אימות פולינומיאלי

אלגוריתם אימות בהגדרתו, הוא קבלה של בעיה x שאיננו יודעים מה התוצאה שלה, ומחרוזת y, שמראה לנו עבור קלט מסוים את הפלט הרצוי לבעיה. לנו נותר רק לאמת שאכן, זה מתקיים. מה הכוונה? אם נקבל מופע של PATH עם כל ההגדרה המלאה שלו, כולל גרף וקדקודים, ואת ה-k הרצוי, אבל לפני שאנחנו מתחילים לחפש את המסלול, ניגש אלינו בחור חשוד ופשוט מביא לנו רצף של קדקודים. אנחנו יכולים בעת לבדוק פשוט את הרצף שהוא הביא לנו, ועל ידי זה לראות אם זה נכון.

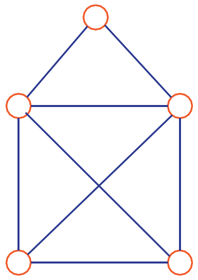
מה יקרה אם נגלה שהאימות שקיבלנו לא נכון? האם זה אומר משהו על הבעייה? לא. יכול להיות שפשוט קילבנו אימות לא נכון, ולכן חשוב לשים לב – אלגוריתמי אימות יכולים להגיד לנו בוודאות על בעיה שהיא **שייכת ל-NP**, אך אין לנו שום דרך לשלול שייכות ל-NP אם לא הצלחנו.

כמובן, שעבור האלגוריתם של מציאת מסלול קצר, אנחנו לא ממש תרמנו הרבה, מאחר שגם ככה הוא מוגדר כשפה פולינומית. זה כן מתחיל לעניין אותנו כאשר מדובר על אלגוריתם שאנחנו לא יודעים איך והאם בכלל אפשר לפתור אותו תחת זמן פולינומי. לדוגמא-

מעגלים המילטוניים

גם בבעיית הגרפים ההמילטוניים נתקלנו בעבר, אבל נזכיר את זה בקצרה – בהינתן גרף G(V,E) האם אפשר למצוא מסלול שיעבור בכל הקדקודים רק פעם אחת, ואז יחזור לקדקוד המקור?

עבור גרפים קטנים קל יותר למצוא מעגל המילטוני[12], אחד המוכרים שבהם הוא בית עם גג ואיקס. כזה-



מי שלא יודע לצייר את זה, שיחזור לכיתה ה'. כמובן שכל שיש יותר קדקודים מצלעות הבעיה נהיית הרבה יותר מסובכת. אם יתנו לנו גרף בלי שום ידע מוקדם, איך נמצא מעגל המילטוני? נבחר קדקוד, ונעבור לשכן שלו, ונחפש מסלול מתאים. ברגע שנתקעים חוזרים אחורה ומנסים שוב. כמה זמן זה ייקח לנו? המון! O(n!) בלי למצמץ אפילו.

לצורך המשך הדיון, נגדיר את בעיית המעגל ההימלטוני בצורה הבאה:

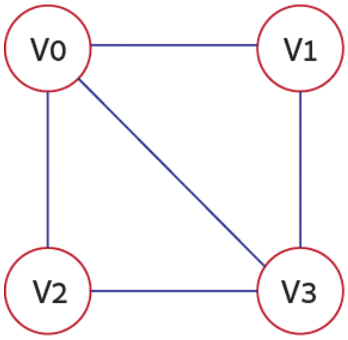
HAM-CYCLE = {<G> : G הוא מעגל המילטוני}

אבל, נגיד שירד נביא מהשמיים – אותו נביא מתת הסדרה המשותפת של התכנון הדינאמי, עכשיו אנחנו כבר יודעים שהוא נביא אמת והכל סבבה, ובקול סמכותי הוא אומר לנו רצף קדקודים שלכאורה פותר לנו את הבעיה. כמובן שאנחנו בודקים זאת ישר – האלגוריתם שנבדוק עבור האימות יבדוק קודם כל שאכן כל הקדקודים שרשומים לנו, אכן מכסים את כל הקדקודים בגרף, ובהמשך נבדוק עבור כל מעבר בין קדקודים אם אכן יש ביניהם קשת מחברת. סך הכל האימות ייקח לנו זמן של $O(n^2)$ – צריך לזכור שאין פה ריצה בעיניים על גרף מצויר, אלא על מימוש בצורה של מטריצת סמיכויות או רשימות מקושרות, ולכן הריצה מתארכת. כך או כך, מצאנו מופע של HAM-CYCLE שהצלחנו לאמת שהוא אכן שייך לשפה תחת זמן פולינומיאלי. הידד.

נפרט כאן קצת יותר, על מנת לראות את משמעות האימות בזמן פולינומיאלי.

[12] שימו לב, מעגל מסתיים באותה נקודה שהתחלנו, ומסלול המילטוני אינו חייב לסיים באותה נקודה ובלבד שיעבור בכל המסלולים. (כמובן שזה קצת קל יותר, אבל לא בהרבה)

בעבור הדוגמא ניקח את הגרף הבא:



לא קשה לראות שמדובר בגרף המילטוני, ולמצוא מסלול מתאים שסוגר פה מעגל. מה שנעשה עכשיו, הוא להגדיר לכל קדקוד בגרף ייצוג בינארי, באופן הבא:

V_0	00
V_1	01
V_2	10
V_3	11

אם נרצה עכשיו לבטא תיאור של קשת מסוימת, נוכל פשוט להשתמש בקידוד של שני הקדקודים ברצף, וזה ייתן לנו את הקישור של שני הקדקודים בערך הקשת. למשל, הקשת (V_2,V_3) יכולה להיות מתוארת ב 1011. אם נרצה עכשיו, נוכל ליצור לנו מחרוזת מוסכמת שהחלק הראשון שלה יכיל את מספר הקדקודים הקיימים בגרף, וזנבה מתאר ברצף את כל הקשתות הקיימות בו. המחרוזת תיראה כך: 0100000100100011011111011. בכוונה צבעתי את החלק הראשון, בשביל שיופרד לנו בראש משאר המחרוזת, למרות שבאמת אין בזה צורך.

בעזרת השיטה שתיארנו עכשיו, אנחנו יכולים ליצור את כל הגרפים הקיימים בכלל, ובפרט את כל הגרפים ההמילטוניים. כל זה טוב ויפה, אבל כל זה היה מופע בודד (ליתר דיוק – מילה x) של גרף המילטוני, איך ממשיכים מפה הלאה? בשביל לאמת את המילה שתתקבל לנו, נגדיר את המחרוזת הבאה: 00011110. שתגדיר לנו את רצף הקדקודים שעלינו לעבור. אמנם זה נראה קצת מוזר במבט ראשון – איך אתה אומר שהאימות הוא פולינומיאלי ביחס לקלט המילה שלנו, אם בעצם האימות שקיבלנו היה כל כך קטן? מאחר ועכשיו אנחנו מגדירים מה אנחנו צריכים לעשות – קודם כל אנחנו בודקים שהקדקודים מספקים אותנו – הווה אומר יש לנו בדיוק את אותם קדקודים שמתוארים לנו במילה. בנוסף כל צמד קדקודים אמור להוות תת-מסלול. אנחנו צריכים לבדוק כל צמד אם אכן יש קשת כזו. איך נעשה את זה? נבדוק במילה של הגרף האם יש לנו צמד קדקודים מתאים, בכל פעם נעבור על כל המילה, בסופו של דבר נגיע לזמן פולינומי של מעבר על המילה בשביל לאמת אותה.

מחלקת הסיבוכיות NP

לאור מה שראינו עם המעגלים ההמילטוניים, אנחנו יכולים בעת להגדיר את המחלקה הבאה, מחלקת NP (Nondeterministic Polynomial time) בתור מחלקת השפות אותן ניתן לאמת תחת זמן פולינומיאלי. אנחנו לא יודעים כמה זמן באמת ייקח לנו אלגוריתם אופטימלי בשביל לפתור את הבעיות האלה, בכל הנראה תמיד זה יהיה אלגוריתם מעריכי כלשהו, אבל בשביל **לאמת**, אני חוזר – **לאמת** ולא **להכריע**, אנחנו מסוגלים לעשות זאת תחת זמן פולינומיאלי.

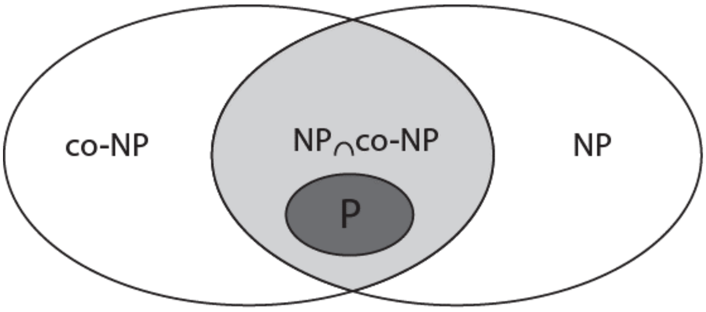
מבחינה פורמלית, נגדיר את השפה כבזו שמקבלת בעיית הכרעה כלשהי, ומחרוזת אימות, ובעזרת אלגוריתם אימות A, אנחנו יכולים לקבוע האם מדובר בבעיה ששייכת ל-NP או לא. ובעוד יותר פורמליות –

L = {x∈{0,1}* : A(x,y) =1 כך ש- |y| = O(|x|^c) קיים אישור y המקיים}

עד פה הצלחנו להוכיח באותות ובמופתים, שהשפה HAM-CYCLE שייכת ל-NP. אפשר גם לומר בוודאות, שאם יש לנו שפה L∈P, אז בוודאי שהיא שייכת גם ל-NP. כי אם ניתן **להכריע** אותה תחת זמן פולינומיאלי, זה למעשה אלגוריתם האימות שלנו עבור כל השפות ב-P. כל אלגוריתם אימות יוגדר להיות האלגוריתם שיפתור את הבעיה בעצמה.

אם כן, אנחנו יודעים בוודאות ש P⊆NP, אך עומדת לנו השאלה, שכבר 50 שנה נשאלת ועדיין לא נענתה – האם P=NP? ככל הנראה לא מדובר בשוויון מלא, אלא רק בהכלה, וזה דבר שדי מוסכם, גם אם לא מוכח בין כל החוקרים כי P ≠ NP. מי שיצליח להוכיח לכל אחד מהכיוונים יזכה לתהילת עולם ו-100 בקורס.

בגדול חלוקת מחלקות הסיבוכיות בעצמה, לא מוסכמת לגמרי על כל החוקרים, אך מה שדי מקובל על רובם (ופרופ' קרנר בכללם) הוא הרישום הבא:



המחלקה P כלולה בתוך הNP. בתוך המחלקה NP יש לנו גם את המחלקה co-NP, המשלימה שלה, שמוכלת בה חלקית – כלומר, אנחנו יודעים להגיד על חלק מהבעיות, שהבעיות המשלימות שלהן בוודאי NP, אך אנחנו לא יודעים להגיד את זה באופן מספיק גורף. כמובן, שאין מה להסתכל על יחס של גודל, כי בעצם ההגדרה של השפות, כל אחת מהמחלקות היא אינסופית, אבל לפחות זה משהו שיכול לתת לנו איזה כיוון ראשוני של מה מוכל בתוך מה.

אם הגדרנו את NP להיות מחלקה לה יש לנו אפשרות לאמת שייכות למחלקה, אבל לא לשלול, אז המחלקה המשלימה עושה בדיוק את ההיפך – יש לנו אפשרות גם לשלול שייכות לשפה. לפי הבנה זו, קל לראות שמחלקה P מוכלת גם ב-NP וגם במשלים, ובשניהם מאותה סיבה – אם אנחנו יכולים להכריע תחת זמן פולינומיאלי, אז בוודאי זה האימות שאנחנו צריכים.

לדוגמא: אם נגדיר את השפה המשלימה ל-SAT, נבקש לקבל רק פסוקיות שתמיד יהיו false. קל מאוד לשלול קיימות של מילה בשפה, ברגע שנקבל השמה שהיא אמת, כבר המילה אינה בשפה. (כמובן שגם טאוטולוגיה – פסוק שהוא תמיד אמת – ניתן לשלול את השייכות לשפה באותו אופן).

שאלות הרחבה

הוכח כי אם $L_1 \in P$, וגם $L_2 \in P$, אזי $L_1 \cup L_2 \in P$

מה שמבקשים מאיתנו להוכיח, זה דבר שאמרנו קודם בדרך אגב, וכבר נדרשנו להרבה הוכחות דומות בקורסים קודמים והוא – סגירות לאיחוד. אנחנו נדרשים להוכיח שאם נאחד שתי שפות השייכות למחלקה P, אז גם השפה שתיווצר לנו, שייכת למחלקה P.

<u>הוכחה:</u> נתון לנו כי $L_1 \in P$, כלומר על פי ההגדרה, קיים אלגוריתם פולינומי A_1 המכריע את השפה L_1. כלומר לכל מילה $x\in \{0, 1\}^*$, A_1 יכול להגיד האם $x \in L_1$ וכל זה תחת זמן פולינומיאלי.

ואותו כנ"ל – $L_2 \in P$, כלומר על פי ההגדרה, קיים אלגוריתם פולינומי A_2 המכריע את השפה L_2. כלומר לכל מילה $x\in \{0, 1\}^*$, A_2 יכול להגיד האם $x \in L_2$ וכל זה תחת זמן פולינומיאלי.

בשאלות מסוג זה, אנחנו צריכים לספק שלושה תוצרים: 1. **<u>L_3</u>** – הגדרת שפה חדשה בצורה ברורה, כולל האלגוריתם המתאים שיפתור אותה. 2. **<u>הוכחת נכונות</u>** – הוכחה שהאלגוריתם שהצענו אכן עומד בדרישות. 3. **<u>הוכחת זמני ריצה</u>** – על מנת לוודא שעדיין נשארנו תחת זמן פולינומיאלי.

1. כעת נגדיר שפה חדשה $L_1 \cup L_2$ = **<u>L_3</u>**.

בעבור השפה החדשה, נגדיר אלגוריתם חדש A_3, שעובד באופן הבא:

בהינתן מילה $x \in \{0,1\}^*$

1. A_3 מריץ את אלגוריתם A_1 על x.
   אם A_1 מחזיר 1, נחזיר 1 ונסיים.
2. אחרת, נריץ את A_2 על x.

   אם A_2 מחזיר 1, נחזיר 1 ונסיים.
3. אחרת – נחזיר 0.

2. **<u>הוכחת נכונות</u>** – נניח ש $x \in L_3$, כלומר $x \in L_1 \cup L_2$, כלומר $x \in L_1$ או $x\in L_2$.

אם כך, יוצא לנו שאם $x \in L_1$ אזי בשורה הראשונה באלגוריתם, אנחנו נחזיר 1 (אמת). ואם $x \in L_2$ אז בשורה השניה באלגוריתם, אנחנו נחזיר 1.

עכשיו נראה מה קורה אם $x \notin L_3$. במקרה כזה, מובן לנו שהוא גם לא משתייך לאיחוד השפות, או לכל אחת מהן בנפרד. במקרה זה בשתי השורות הראשונות כלום לא יקרה, אך ברגע שנגיע לשורה השלישית , לא נותרו לנו כבר בדיקות לעשות, ונחזיר את הערך false.

3. **<u>זמני ריצה</u>** – נתון לנו שזמני הריצה של שתי השפות L_1, L_2 הם פולינומיאליות ושייכות למחלקה P. כך שגם אם נצטרך להריץ את שני האלגוריתמים אחד אחרי השני, זה יצא לנו זמן פולינומי כפול 2. מה שעדיין מגדיר לנו את זה בזמן פולינומיאלי, מש"ל.

הערה: ניתן להוכיח בדיוק באותו אופן גם את הסגירות לחיתוך של המחלקה P. כל שעלינו לעשות, הוא לקחת בדיוק את אותה הוכחה שכתבנו עכשיו ולהתאים לסימונים מתאימים לחיתוך – אם נרצה להוכיח שמילה כלשהי שייכת לחיתוך השפות L_1 ו-L_2, נכתוב את האלגוריתם באופן שיבדוק בשתי השורות הראשונות שהמילה משתייכת לכל שפה בפני עצמה, ורק במידה ויש לנו את שני האישורים נוכל להחזיר אמת עבור המילה. כל יתר ההוכחה תשתנה בהתאם בשינויים קלים. ובזה באנו על מקומנו בשלום.

הוכח כי אם L∈P, אז $\overline{L}$∈P

פה אנחנו נדרשים להוכיח סגירות למשלים תחת זמן הכרעה פולינומי. ההוכחה הנ"ל הינה יחסית קלה, כי מאחר ואנחנו יכולים להריץ את האלגוריתם של השפה בזמן פולינומיאלי, ולהכריע שהמילה שקיבלנו אכן שייכת לשפה, אז המרחק בין זה לבין להוכיח שהיא לא קיימת בשפה (כלומר קיימת בשפת המשלים) הוא בדיוק אותו זמן פולינומיאלי, פלוס שורה אחת – החזרה הפוכה של התוצאה.

מבחינת הנכונות – אם המילה קיימת בשפה המקורית, כלומר x∈L, אזי ההחזרה ההפוכה תוציא לנו שהמילה לא קיימת במשלים, ואם נקבל שהמילה לא קיימת בשפה L, אזי בהכרח היא קיימת בשפה המשלימה.

זמן הריצה – כמו שאמרנו יהיה גם הוא פולינומי, בדיוק כמו השפה המקורית.

הוכח כי אם $L_1 \in NP$, וגם $L_2 \in NP$, אזי $L_1 \cup L_2 \in NP$

בשאלה זו לא מבקשים להוכיח לנו את P, שהוא יחסית פשוט, אלא איחוד של NP. מה השוני הגדול? ההגדרה של אלגוריתם אימות ב-NP, הוא כזה שאני מקבל אלגוריתם בודד, ובעבור זה מחזיר את מחרוזת האימות המתאימה. אך כאן יש לי שני אלגוריתמים ושתי מחרוזות אימות. כמובן שצריך לעשות פה איזה שינוי שהוא קצת יותר גדול.

נתחיל בנתונים:

$L_1 \in NP$, כלומר קיים איזה אלגוריתם זמן-פולינומיאלי A_1 שבעבור כל מילה $x \in \{0,1\}^*$, קיים אישור y_1, כך ש $|y_1| = Poly(|x|)$, כך ש $x \in L_1$ אם"ם $A_1(x,y_1) = 1$.

וכן $L_2 \in NP$, כלומר קיים איזה אלגוריתם זמן-פולינומיאלי A_2 שבעבור כל מילה $x \in \{0,1\}^*$, קיים אישור y_2, כך ש $|y_2| = Poly(|x|)$, כך ש $x \in L_2$ אם"ם $A_2(x,y_2) = 1$.

בעת נציע אלגוריתם A_3 למימוש השפה L_3. אנחנו צריכים להראות שיש אלגוריתם שבעבורו לכל מילה x נוכל להגיד האם המילה היא בשפה או לא. מה שנאמר הוא כדלקמן – אם $x \in L_2$ אז זה אומר שהוא נמצא ב-L_1 או ב-L_2. כלומר קיים אישור כלשהו למילה שהוא $y_1 \backslash y_2$. דבר זה הוא נתון, ולכן לא ניתן לשינויים. באותו אופן אנחנו יכולים לבדוק עבור כל מילה את שתי האופציות, ואם המילה שייכת לאחת משתי השפות על ידי האימותים המתאימים – אזי המילה שייכת ל-L_3.

1. **<u>L_3</u>** – בהינתן מילה $x \in L_3$, נריץ את A_1 עם האישור המתאים y_1. אם $A_1(x, y_1) = 1$ – מה טוב. נחזיר 1 ונסיים. אחרת, נריץ את A_2 עם האימות y_2. אם $A_2(x,y_2)= 1$ נחזיר 1. אחרת, נחזיר 0 ונסיים. עד כאן תיאור האלגוריתם.

2. **<u>הוכחת נכונות</u>** – הנכונות של זה בדיוק כמו בהוכחה הקודמת של איחוד השפות הפולינומיאליות, או שנקבל אימות בעבור אחת מהשפות, או שלא נקבל אימות משום שפה.

3. **<u>זמן ריצה</u>** – מאחר ואנחנו עושים פה אימותים על אלגוריתמים שהם זמן-פולינומיאלים, נישאר תחת אותה הגדרת זמן, ולא נעבור למסגרת אחרת שהיא מעריכית.

הוכח שלכל $L \in NP$ קיים אלגוריתם מעריכי שמכריע אותו

אנחנו יודעים שלכל שפה NP קיימת לכל מילה x איזה אלגוריתם אימות, התלוי ב-y, שרץ בסך הכל בזמן פולינומיאלי, כלומר $|y| = Poly(x)$. אם נניח ש $|x| = 10$ אז אלגוריתם האימות יכול להיות $|y| = 10^2$. לא תמיד מדובר ביחסים האלה, לפעמים זה יותר ולפעמים פחות, אך זה מספיק לצורך הדוגמה.

המשמעות של זה היא, שבהינתן מילה x, אותה אנחנו צריכים להכריע ללא שום ידע מוקדם, ואנחנו אלו שנחליט אם המילה שייכת לשפה או לא, אז אנחנו יכולים לראות שיש לו חסם מינימלי על הגודל והוא זמן הריצה של אלגוריתם האימות. אבל אם צריכים לחפש את אותו מסלול, או כל אימות אחר בעצמנו, היינו צריכים לחפש את כל האופציות הקיימות, כלומר $2^{|y| = Poly(|x|)}$. וזה בהכרח אלגוריתם בזמן ריצה מעריכי. מש"ל

שלמות ב-NP ורדוקציות

דיברנו עכשיו על שתי המחלקות שמהוות לכאורה את שתי הקצוות של הסיבוכיות – P, NP. אמנם אנחנו לא יכולים לקבוע בוודאות מוחלטת שמדובר על שתי מחלקות שונות, אבל נזרום על הטענה לפיה $P \neq NP$. עכשיו אנחנו נתעסק בשתי מחלקות נוספות, שהם הבשר האמיתי של הסיבוכיות. והמבחן.

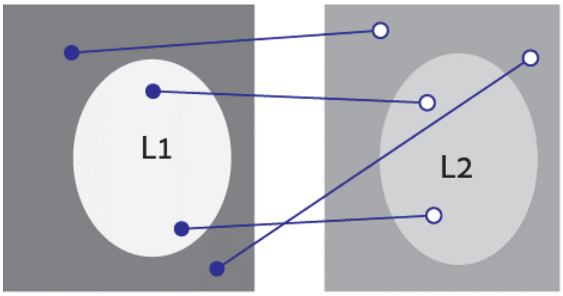
בדיאגרמה שהבאנו קודם, חילקנו את אחד מהעיגולים לחלק שנמצא בתוך המחלקה NP, ולחלק שנמצא מחוץ לו. ההבדל בין שתי המחלקות מבחינה פורמלית, הוא שהמחלקה של ה-NPH זה הבעיות הקשות שנמצאות מחוץ לNP, וה-NP השלמות נמצאות בתוך ה-NP. ההגדרה הזאת נראית די מתחמקת כי פשוט כתבתי את אותו הדבר פעמיים, אבל יש לה חשיבות.

שתי המחלקות הללו נחשבות לבעיות "קשות" להכרעה, אך לכל הבעיות וואלגוריתמים המוכלים בהם, ניתן למצוא קישור שתופר את הכל ביחד. דבר זה יוצר לנו מצב מיוחד בבעיות ה-NP שלמות, שלא ברור לנו בדיוק האם הם יותר שייכות ל-NP או ל-P. על פי ההגדרה, מאחר וכל הבעיות קשורות אחת בשניה, וכמובן שאנחנו מדברים פה על אינסוף בעיות, אם נצליח להוכיח ולו לאחת מהבעיות שניתן לפתור אותן בזמן פולינומי, אזי כל המחלקה ה-NP שלמה עוברת במכה אחת להיות מוגדרת תחת מחלקת P. יותר מזה – זה גם יפתור לנו את שאלת החיים, היקום וכל השאר – אם $P=NP$? אם נעביר את הNP השלמות, נוכל לומר בצורה ברורה שגם $P=NP$.

את קישור הבעיות למחלקה, נעשה בדרך שנקראת רדוקציות, שנסביר עכשיו.

רדוקציות

רדוקציה, היא ניסוח מחדש של פונקציה, באופן שמסייע לנו להגדיר בעיה שלא היינו מסוגלים להגדיר אותה לפני כן. מה הכוונה? נניח ויש לנו שתי שפות L_1, L_2, ואנחנו יודעים להגדיר רק את L_1 (בדרך כלל, פשוט מדובר על זה שאנחנו יודעים שהשפה שייכת ל-NP) אבל לא מצליחים להגדיר את L_2. אם נצליח למצוא איזו המרה (רדוקציה) שעובדת בזמן פולינומיאלי באופן מלא – כלומר כל דבר שיתקבל ב-L_1 גם יהווה הכרעה לבעיה המתאימה ב-L_2, וכל בעיה שמוחזרת שאינה שייכת לשפה L_1 גם לא תהיה שייכת ל-L_2, אז אנחנו יכולים לסמן כי $L_1 \leq_p L_2$ חשוב לשים לב פה לכיוונים והכל – מה שאנחנו אומרים פה שהשפה L_2 שמכילה את אינסוף הבעיות שלה היא "קשה" יותר וגדולה יותר מ-L_1, כלומר, במקרה הכי טוב נצליח ממש להוכיח שאם נבדוק אותה שפה ההכלה היא הכלה מלאה, אבל בשונה משקילות קידודים שראינו קודם, אין לנו דרישה של הכלה דו כיוונית, אלא אם יש לנו הכלה מלאה מצד אחד לשני זה כבר מספיק לנו.



האיור מדגים לנו את הדרך שאנחנו צריכים ממש להראות ש**לכל** פונקציה ב-L_2 קיימת הפונקציה המתאימה לה בתוך או מחוץ לשפה.

דבר נוסף שבדאי לשים לב – הסימון שאנחנו עובדים איתו, הוא לא סתם גדול/שווה. כאשר אנחנו עושים רדוקציה משפה מסויימת L_1, אנחנו אומרים שיש לנו אפשרות להעביר את **כל** הבעיות ששיכות לשפה (בעזרת שינויים קטנים וכיוונונים), להיות מתאימים לשפה L_2. למרות שמבחינת מרחב הבעיות ב-L_2 עצמה, אנחנו נוגעים רק במרחב בעיות יחסית מצומם – אנחנו נראה בהמשך רדוקציות שהן מובילות אותנו לבעיות שהן מאוד ספציפיות, אבל זה מספיק לנו מבחינת ההוכחה, כי כמו שכבר אמרנו – אנחנו מוכיחים שהשפה החדשה שקיבלנו היא **קשה** יותר מהשפה ממנה עשינו את הרדוקציה – הדרך בה זה עוזר לנו, זה הקל-וחומר שאנחנו עושים ואומרים שאם הבעיה הקשה יותר תיפתר תחת זמן פולינומיאלי זה יפתור לנו גם את הבעיה הקלה. אך כמובן שזה לא יעבוד באופן הפוך – פתירת הבעיה הקלה לא תפתור כלום מהקשה.

לדוגמה: מישהו מסתבך ביצד לפתור את הנוסחה הבאה $bx+c = 0$. אבל אחרי הרבה זמן, הוא מחליט פשוט להשתמש בנוסחה למציאת X ממעלה שניה – $ax^2+bx+c=0$, כי הוא זוכר את הנוסחה בעל פה. ועל ידי זה לפתור את הבעיה הזאת. כמובן שההמרה הזאת מאוד בעייתית, למה? כי בשביל שהנוסחה תוציא לו אותו דבר, הוא צריך להציב פה $a=0$, ואם הוא ילך על הנוסחה הרגילה למציאת ה-x הוא יצטרך לחלק ב-0. אז זה לא הפתרון. פתרון שכן יהווה רדוקציה, הוא לפתור את זה באופן הבא – $x = -c/b$. כאן בטוח לא תהיה בעיה, כי אם ה $b=0$ גם ה-x נעלם מהנוסחה המקורית, והצלחנו לספק את הרדוקציה.

מבחינה פורמלית, אנחנו מגדירים את הרדוקציה באופן הבא – השפה L_1 ניתנת לרדוקציה בזמן פולינומיאלי לשפה L_2 (הביטוי שרשמנו מקודם, $L_1 \leq_p L_2$ מבטא את בניית הרדוקציה) אם קיימת פונקציה $f:\{0,1\}^* \rightarrow \{0,1\}^*$ כך שעבור כל מילה $x \in \{0,1\}^*$ מתקיים:

$x \in L_1$ אם ורק אם $f(x) \in L_2$[13].

כלומר הפונקציה של רדוקציה צריכה להיות כל כך מוחלטת, עד כדי שנוכל לבדוק מילה שתיכנס אליה, ואם הפלט לא יצא שייך לפה L_2 הוא גם לא יהיה שייך לשפה L_1.

למה 36.3

אם $L_1, L_2 \subseteq \{0, 1\}^*$ הן שפות המקיימות $L_1 \leq_p L_2$, אזי $L_2 \in P$ גורר $L_1 \in P$.

הסבר: הלמה הזאת מניחה לנו את הקשר-זמן-ריצה בין הפונקציות לאחר הרדוקציה. אנחנו טוענים שאם המעבר בזמן פולינומיאלי, העביר לנו שפה אחת, לא ידועה, לשפה אחרת שהיא ידועה כפולינומית, אז גם השפה הראשונה היא שפה פולינומית. שימו לב! אנחנו מדברים על גרירה חד ביוונית. כלומר, אם ידוע לנו שהשפה L_2 אינה פולינומית, או לחילופין ידוע לנו שהשפה L_1 שיצאנו ממנה היא בעצמה פולינומית, זה לא אומר שום דבר לגבי השפה השניה.

ההוכחה כמובן, די פשוטה, אם אנחנו מגדירים רדוקציה זמן פולינומיאלית, והאלגוריתם שאנחנו מגיעים אליו גם הוא פולינומי, אז יש לנו פה פשוט הרכבה, ואנחנו יכולים להגדיר את השפה המקורית כבעלת פיתרון פולינומיאלי. ובשפה פורמלית יותר:

<u>הוכחה:</u> יהי A_2 אלגוריתם זמן-פולינומיאלי המכריע את L_2, ויהי F אלגוריתם רדוקציה שרץ בזמן פולינומיאי ומחשב את פונקצית הרדוקציה f (עד כאן זה הבסיס הנתון לנו).

---

[13] זה תנאי לנכונות הרדוקציה, ונשתמש בו בהמשך – נקרא לו פשוט 36.1.

אנו נבנה אלגוריתם זמן-פולינומיאלי A_1 המכריע את L_1 (תכלס, נגדיר "אלגוריתם" שפשוט מבצע את כל הפעולות של הרדוקציה + ההכרעה מA_2). נכונות האלגוריתם נובעת מהתנאי הקודם על שייכות המילים לשפות ולרדוקציות, באופן שמכריח אותנו שהכל ירוץ תחת זמן פולינומיאלי, והרי לנו חיבור פולינום. מש"ל.

שלמות ב-NP

למעשה, מה שהרדוקציה נותנת לנו, הוא קישור בין פונקציות שונות ברמה שאינו משנה משמעותית את זמן הריצה – מאחר והמחלקה היעילה ביותר שאנחנו מדברים עליה היא פולינומית, אז הקישור של רדוקציה לכל המחלקות האחרות לא מעלה ולא מוריד. מה שהוא כן עושה, זה לקשור את כל הפונקציות ביחד לחבילה אחת, או בניסוח אחר, להראות שבעיה מסוימת קשה לפחות כמו בעיה אחרת, אבל רק ברמה פולינומיאלית, אז תכלס שניהם אותו דבר.

את מחלקת הבעיות השלמות ב-NP, נגדיר בעזרת האיחוד הנ"ל, באופן הבא:

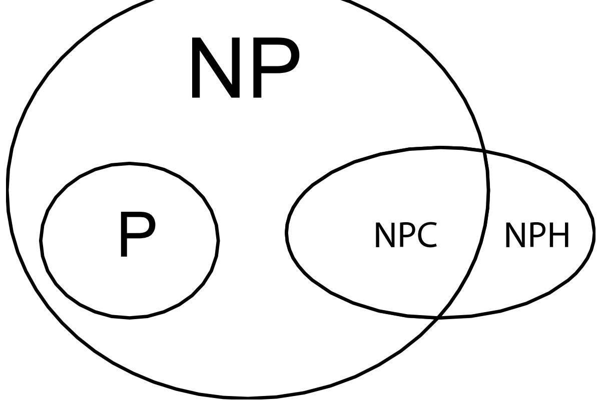
שפה $L \subseteq \{0,1\}^*$ היא שלמה ב-NP, אם מתקיימים שני התנאים הנ"ל:

1. $L \in NP$
2. $L' \leq_p L$ עבור כל $L' \in NP$.

נתחיל מהתנאי השני. מאחר ודיברנו על כל עניין הרדוקציה, ואנחנו בעצם כובלים את הבעיות אחת לשניה, אז הבעיות השלמות הן אלה שאפשר לכבול עליהם את כל מה שנמצא ב-NP. כדאי לזכור, שבמחלקת ה-NP קימים גם כל הבעיות של P, מה שכמובן גם מסביר את הקשר בין הNP שלמות לשאלת הP=NP. אם כל הבעיות מתחברות לNP שלמות בעזרת הרדוקציה, אז הכרעה מוחלטת שתראה ניתן להעביר הכל לזמן פולינומי, תעביר גם את אלו שהיו קשורות, אך היו מוגדרות בתור NP. בגלל הקשר ההדוק הזה, רוב המחקר נע על הציר של בעיות אלו, ולא על ניסיון להראות אם יש/אין שיוויון.

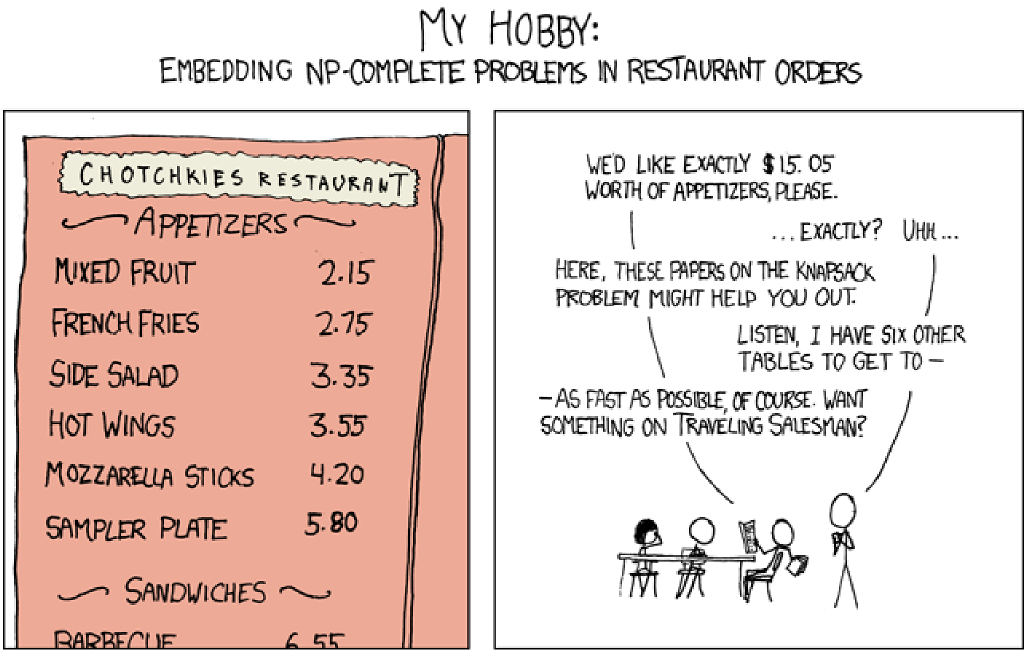
התנאי הראשון מגדיר לנו את הקבוצה כמוכלת בתורך הקבוצה NP. יכול להיות מצב שנמצא בעייה שניתן לעשות אליה רדוקציות שמקיימות את התנאי השני, אבל אם לא ניתן לבעיות אלו אלגוריתם אימות, הם ייצאו מ-NP ויעבור למחלקה של ה-NPH (הקשות ב-NP).

הגדרה זו, מעדכנת לנו את תרשים המחלקות שראינו קודם, ואם נסתכל על המחלקה NP באופן יותר ממוקד, תרשים המחלקות יהיה כזה –



כלומר המחלקה השלימה ב-NP כמובן תהיה מוכלת בה, אך היא תהיה צמודה ל-NPH שהן הבעיות הקשות שאפילו אין להן אלגוריתם שייאמת אותן בזמן פולינומיאלי.

עד עכשיו הסברנו מה זה בעיית NP שלמה, והבהרנו שאנחנו צריכים ליצור את כל הבעיות בנדבך אחד על גבי השני. אבל עכשיו נתקענו בבעיית ה"Kit-Kat". חטיף הקיט-קט (שהוא בעצם כיף-כף, אבל לפני שעלית גנבו אותו לעברית), הוא חטיף וופל מצופה שוקולד. שהוופל שלו עשוי מחטיפי קיט-קט גרוסים. שהם היו עשויים מחטיפי קיט-קט גרוסים ובן הלאה (או בצורה פשוטה יותר – רקורסיה). ככל הנראה הקיט-קט הראשון היה עשוי באמת מוופל, ואנחנו עכשיו מחפשים את הקיט-קט הקדמון, המקור והאב הקדמון לכל חטיפי הוופל המצופים שוקולד. נתחיל עם אלגוריתם אחד שמוכח שהוא קשור להיות NP-complete, ואת שאר הבעיות נגדיר על גביו.



http://xkcd.com/287/

ספיקות מעגלים

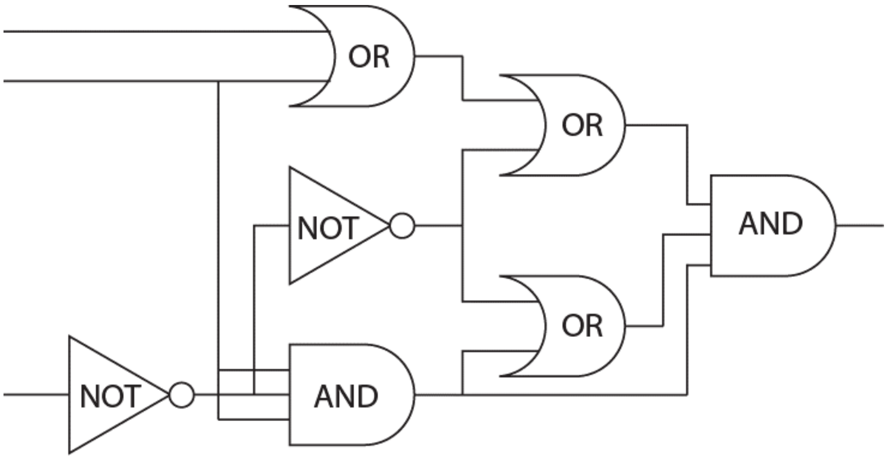
בעיה זו הינה הבעיה הראשונה שהוכחה כשייכת למחלקת השלמות ב-NP, על ידי סטפן קוק ב-1971, ולכן נהוג לקרוא את זה על שמו "משפט קוק". את הבעייה עצמה נציג בצורה פשוטה, אבל לא ניכנס לעומק ההוכחה, כי היא ארוכה, מסובבת ופחות רלוונטית

בעיית ספיקות המעגלים, מתייחסת למעגלים לוגיים באופן שלמדנו עליו ב"ספרתיות". במובן, שהיישום של זה נוגע לאלקטרוניקה ועוד, ובהמשך ניתלה על הבעיה הזאת וניצור עוד בעיות שיתקיימו תחת המטריה של NPC.

הבעיה מוגדרת כך: בהינתן מעגל צירופי בוליאני, המורכב משערי AND, OR, ו-NOT. האם המעגל ספיק?

מבחינה פורמלית, נתחשב בקידוד המוכר של שערים לוגיים שלמדנו בספרתיות. עבור כל קבוצת שערים לוגיים המחוברים ביניהם, יש לנו שלוש אפשרויות לתוצאה: 1. הפלט יהיה תמיד T, 2. הפלט יהיה תמיד F, 3. הפלט יהיה תלוי בהשמת הערכים בחוטים.

נראה דוגמא למעגלים לפי האפשרויות השונות-



בהינתן לנו מעגל לוגי, כמו בדוגמא למעלה, איך נוכל למצוא את התשובה לבעייה? נצטרך להתחיל להציב ולבדוק מה התוצאה. כמובן, שאם אנחנו מציבים וממשיכם לקבל שהתוצאה היא T, זה לא אומר שהמעגל תמיד ספיק – תמיד יכול להיות שדווקא הבדיקה האחרונה היא זו שתניב לנו תוצאת F. כמובן שבדיקה של כל האפשרויות היא מעריכית – $O(2^n)$ ביחס לכמות השערים. מצד שני, גם אם נבדוק את כל הפרמוטציות האפשריות ונקבל F, אנחנו חייבים להגיע עד הסוף בשביל לוודא שהמעגל לא מסופק. למה זה רלוונטי? כי אם יש מעגל שבכל הזמן מוציא 0/1 אז אפשר פשוט לבטל את כולו ולהחליף בחוט בודד באותו הערך.

מבחינת בעיית הסיפוקים, מה שמעניין אותנו בסוף זו רק השאלה הבולייאנית של האם המעגל ספיק או לא-ספיק. התשובה לא צריכה להיות "הוא ספיק בשלוש מתוך חמש מאות מקרים", אלא אם מצאנו השמה שפועלת, אפשר לעצור.

את השפה הפורמלית, נגדיר בצורה הבאה: {C הוא מעגל צירופי בוליאני ספיק: <C>} = CIRCUIT-SAT

על מנת להוכיח שהשפה קיימת ב-NP שלמה, נוכיח את שני התנאים:

1. בעיית ספיקות המעגלים שייכת למחלקה NP (למה 36.5) – זה שהאלגוריתם שפותר את הבעיה הוא מעריכי, ברור לנו כבר באופן די מוחלט. השאלה היא האם קיים אלגוריתם אימות, שבעבורו נוכל לאמת בזמן פולינומיאלי את ספיקות המעגל. כמובן, שהתשובה היא כן, אם נקבל את המעגל C וביחד איתו קוד אימות שמתאר לנו את ההשמה המתאימה, ברור שנוכל לאמת את זה במהירות פולינומיאלית, ואולי אפילו לינארית, אם נצליח לעבור בצורה פשוטה. נזכיר רק, שאלגוריתם האימות לא יכול לשלול לנו קיום, אלא רק להוכיח נכונות של המעגל תחת השפה של ספיקות המעגלים.
2. בעיית ספיקות המעגלים היא NP-קשה (למה 36.6) – התנאי השני, האומר שניתן לעשות **לכל** בעייה ב-NP רדוקציה לבעייה הנתונה, מגדיר לנו את הבעיה (במקרה שלנו, ספיקות המעגלים) כבעייה "קשה". נראה אחר כך מה המשמעות של זה, אבל מבחינתנו, יכולות להיות בעיות קשת שהם לא ב-NP, כלומר יכול להיות בעיות שאין לנו אפשרות אפילו לאמת אותן בזמן פולינומי. רק הקשירה של שתי התנאים, הופכת את הבעיה ל-NP שלמה. ולעניין שלשמו התכנסנו, יש הוכחה ארוכה ומייגעת עליה דלגנו בשיעור, אז רק תסמכו על קורמן (ומרן פרופ' קוק) שזה נכון.

הנובע משתי הלמות הנתונות לנו, הוא המשפט החשוב-עד-למאוד 36.7 – בעיית הספיקות של מעגלים היא NP-שלמה.

הוכחות שלמות ב-NP

הוכחנו בצורה ברורה (לכאורה) שכל בעיה ב-NP יכולה לעבור רדוקציה $L \leq_p$ CIRCUIT-SAT. הוכחנו את זה כל כך לכאורה, שקשה מאוד לחשוב שאנחנו צריכים להוכיח את זה מחדש עבור כל בעיה. השיטה שלנו להוכחת הנכונות עבור שאר הבעיות, תהיה בעזרת הלמה הבאה:

למה 36.8

אם L היא שפה כך ש- $L' \leq_p L$ עבור איזושהי $L' \in NPC$, אזי L היא NP-קשה.
יותר על כן, אם $L \in NP$, אזי $L \in NPC$.

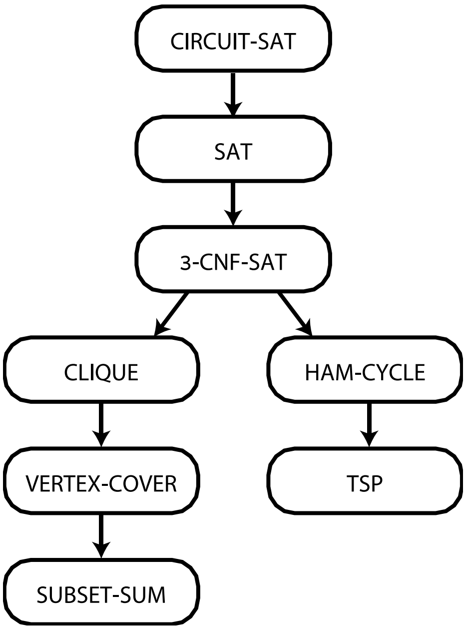
הלמה הזאת תוקפת את הוכחת השלמות ב-NP מזוית שונה. אנחנו אמרנו שהרדוקציה היא חד ערכית, כלומר את כל אינסוף הרדוקציות שעשינו (או לפחות הוכחנו שאנחנו מסוגלים לעשות), עבור ההוכחה הראשונה, אין להם שום משמעות מבחינת בעיות המקור. אבל אם נצליח לעשות רדוקציה משפה שהיא שלמה לשפה אחרת, אז אנחנו קושרים את השפות במרחק של מעבר פולינומיאלי. אבל אנחנו מקבלים עם זה בונוס בצורה טרנזטיבית. אם כל השפות ב-NP יכולות לעבור באינדוקציה לשפה 'L כלשהי, תחת זמן פולינומיאלי, ואם אנחנו יכולים להוסיף לזמן הזה עוד מעברון קליל לבעיה אחרת – אז גם לשפה L אנחנו מסוגלים להגיע תחת רדוקציה זמן-פולינומיאלית! והנה הגדרנו את השפה כ-NP קשה. אם נצליח למצוא גם אלגוריתם אימות, ולמקם את הבעיה ב-NP, קיבלנו פה NP שלמה לתפארת.

נסדר עכשיו את השיטה בצורה מסודרת. על מנת למצוא שפה שהיא NP-שלמה, עקוב אחר השלבים הבאים:

1. הוכח כי $L \in NP$. (אם נעשה את כל השלבים על שפה שהיא בכלל לא ב-NP, אנחנו סתם עובדים על ריק, ולכן, זאת תהיה הבדיקה הראשונה שלנו.
2. בחר שפה NP-שלמה ידועה 'L. (למשל ספיקות מעגלים)
3. תאר אלגוריתם המחשב פונקציה f הממפה כל מופע של 'L למופע של L.
4. הוכח שהפונקציה f מקיימת $x \in L'$ אם ורק אם $f(x) \in L$, עבור כל $x \in \{0,1\}^*$. שזה תנאי הרדוקציה 36.1 שהגדרנו במקומו (במילים אחרות – הוכחת נכונות).
5. הוכח שהאלגוריתם המחשב את f רץ במן פולינומיאלי (הוכחת זמן ריצה).

עכשיו אנחנו נתחיל לבנות תילי-תילים של הוכחות לבעיות שכולם נשענות על ספיקות המעגלים. בכל פעם, ניקח איזה פן אחר של הבעיה ונתמודד איתו בעזרת השיטה שהגדרנו.

עץ ההוכחות שנעבור נראה כך:



כאשר כל הוכחת בעייה ב-NPC תוכל להוביל אותנו אל עבר הבעיה הבאה, כך שיהיה קשר הדוק בים כל ההוכחות.

ספיקות נוסחאות

השלב הבא שנתמודד איתו הוא ספיקות נוסחאות. ברמת הרעיון הפשוט קל לראות את הקשר ספיקות מעגלים לנוסחאות, אנחנו מדברים על אותו עולם דיון רחב. אבל מאחר שבנוסחאות הלוגיות, אנחנו משתמשים בסימנים יותר מורכבים, אנחנו צריכים להוכיח באופן ברור את כל הרדוקציה.

עבור הגדרת הבעיה SAT נשתמש בהגדרות הבאות -ראשית נגדיר את המופע של SAT בתור ϕ המרכב מהסימנים הבאים:

1. משתנים בוליאנים : $x_1, x_2...$
2. קשרים בוליאניים: כל הקשרים שאנחנו רגילים אליהם ומכירים מהקורס בלוגיקה - $\vee$ (OR), $\wedge$ (AND), $\neg$ (NOT), $\rightarrow$ (גרירה לוגית), $\leftrightarrow$ (אם ורק אם)
3. () סוגריים.

בדומה לספיקות המעגלים שראינו קודם, אנחנו מחפשים גם פה בעבור נוסחה נתונה, האם ניתן לקבוע איזה הצבה שהנוסחה תוציא ערך אמת. נגדיר בצורת שפה פורמלית באופן הבא:

SAT = {<ϕ> : ϕ היא נוסחה בוליאנית ספיקה}

כמובן שגם פה הדרך הנאיבית לפתור נוסחה שכזו הוא פשוט להציב בה השמות עד שנגיע לתוצאה הרצויה. וכמובן שגם פה 2^n וכל הסיפור הידוע והמוכר. לא צריך לכתוב את זה שוב.

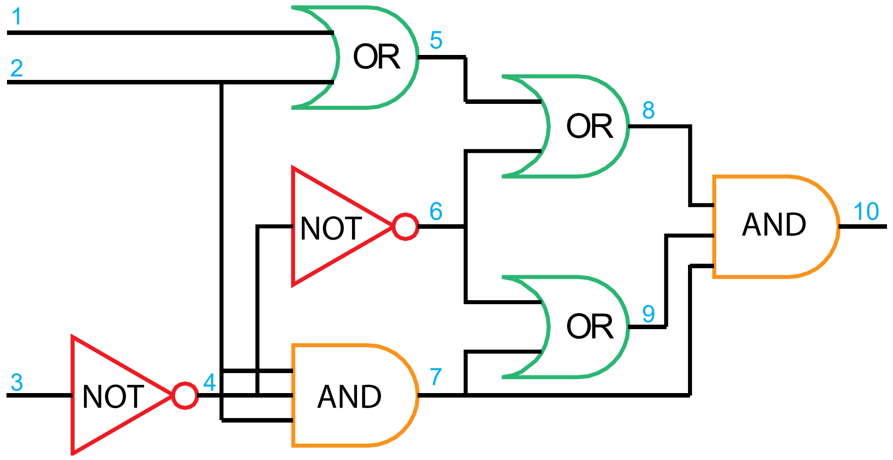
<u>**משפט 36.9:**</u> ספיקות נוסחאות בוליאניות היא NP שלמה.

על מנת להוכיח את זה, נרצה להוכיח שני דברים (חשוב לזכור את זה כי כל שאלה שתגיע על הוכחת נוסחה כלשהי נצטרך לעבוד באותו אופן) הדבר הראשון – $SAT \in NP$. כמו שהזכרנו מקודם, אם זה לא ב-NP זה בטוח לא NP שלמה, ואין לנו מה לעבוד לחינם. הדבר השני הוא (וכאן מתחיל החידוש) נראה כי

כלומר, נעשה את הרדוקציה מבעית המעגלים לבעית ספיקת הנוסחאות, $\text{CIRCUIT-SAT} \leq_p \text{SAT}$ נוודא שהרדוקציה עובדת בזמן-פולינומיאלי, ועל ידי זה נוכל לומר שהבעיה שלנו היא NP שלמה.

נתחיל עם השייכות ל-NP. כל שאנחנו צריכים להוכיח הוא שבהינתן אישור שאנחנו יכולים לאמת – במקרה הזה, השמה של כל המשתנים באופן שיוביל אותנו לקבל ערך "אמת", ביצוע יהיה בזמן פולינומיאלי. ובכן, מה שנצטרך לעשות הוא לעבור על כל הנוסחה, ולהחליף כל משתנה בערך המתאים לו מהמילה שקיבלנו לאימות. לאחר שנחליף את כל המשתנים, נצטרך לבדוק שאכן ההשמה היתה מוצלחת וקיבלנו 1. כל זה יכול להתבצע בזמן פולינומיאלי, ולכן $\text{SAT} \in \text{NP}$.

עכשיו נדבר על הרדוקציה $\text{CIRCUIT-SAT} \leq_p \text{SAT}$ – אנחנו רוצים ליצור מעבר שנוכל לבטא על ידו את בעיית ספיקת המעגלים, בצורה של ספיקת נוסחאות בוליאניות. עקרונית ניתן לבנות כל שער בצורת אינדוקציה אט אט ולכסות את כל המעגל הקיים לנו, הבעיה שאם נעשה ככה, אנחנו עלולים להתקע במעגלים ארוכים ומסובבים. לכן, הדרך שנעשה את זה, הוא במקום לנסות וליצור את הנוסחה המדויקת ביותר, שעולולה לסבך אותנו, נפרק פשוט כל חלק לתת-נוסחה בוליאנית. לדוגמא נסתכל על המעגל שלקחנו כדוגמא בחלק הקודם, ונמספר את הכניסות ואת היציאות של כל מעגל הספיקות באופן הבא:



צבעתי אותו גם יפה, שיהיה לנו קצת יותר נוח להסתכל על כל השערים. עכשיו, אופן הרדוקציה תהיה בדלקמן – ניקח שער מסוים, לצורך העניין, נתחיל מהקצה, ונבדוק איך אנחנו יכולים להמיר את זה לנוסחה לוגית. יש לשים לב, תוצאה של 1 שתתקיים במוצא השער תלויה באופן מובהק במה שנכנס. וכן להיפך, ולכן את הקשר בין כל שני צדדים של שער נבטא לכל אורך הרדוקציה בתור קשר $\leftrightarrow$ אם ורק אם. ולכן, נחזור לשער הפלט, ונגדיר את הנוסחה שלו באופן הבא: $(x_7 \wedge x_8 \wedge x_9) \leftrightarrow x_{10}$. את כל הנוסחה הזאת נסגור בסוגריים, ונעבור הלאה לשער הבא, כאשר כל חלק וחלק נחבר עם קשר AND מאחר ואנחנו בהחלט דורשים שכל התנאים האלה יתקבלו בכל חלקי הנוסחה. לאחר שנפרק את כל המעגל לנוסחאות נקבל את המופע הבא:

$$\begin{aligned}
\phi = x_{10} &\wedge (x_4 \leftrightarrow \neg x_3) \\
&\wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\
&\wedge (x_6 \leftrightarrow \neg x_4) \\
&\wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\
&\wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\
&\wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\
&\wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9))
\end{aligned}$$

אחרי שהבנו את הרעיון של בניית הנוסחה מתוך בעיית ספיקת המעגל, אנחנו יכולים להבין שהבנייה של דבר בזה היא לא יותר מזמן פולינומיאלי, נניח ויש לנו n שערים ו-m חוטים, גם אם נגיד שהם איכשהו מחוברים ומסובבים אחד בשני כמו אוזניות חוטיות שמכניסים לכיס, עדיין לא נצטרך לעבור יותר מ-n^2.

עכשיו אנחנו צריכים לוודא שהקשר של הרדוקציה הוא אכן אמין – כלומר, שאם המעגל C המקורי ספיק, גם הנוסחה ϕ ספיקה. נאמר כי מאחר ויש לנו השמה טובה עבור C, אז כל תיל מתוך בפני עצמו מושם גם הוא בצורה איכותית, ולכן גם החיבור בין כולם נשאר מספק באותו אופן. וגם להיפך, אם הנוסחה שכתבנו מספקת לנו תוצאה 1, אז ההחזרה של הנוסחה לצורה של מעגלי ספיקה בוודאי שיתואר באופן שיחזיר גם הוא אחד. מש"ל.

(תכלס ההוכחה הזאת די צולעת, כי אנחנו פשוט אומרים שאם עשינו את השלבים נכון, הכל עובד לכל הכיוונים. מה שאנחנו צריכים לשים לב באמת, זה שאם נגדיר את הרדוקציה בצורה נכונה ולא יהיו לנו חורים לא מטופלים, ההוכחה של הנכונות של הרדוקציה לא צריך להיות באמת הרבה יותר מזה.)

ספיקת נוסחאות 3-CNF

אחרי שהוכחנו בתופים ובמחולות שהשפה $SAT \in NPC$, אנחנו יכולים להתפנות ולהתחיל לבנות עליה הוכחות נוספות. אנחנו רוצים לבנות איזה נוסחה שיהיה קל לראות עליה האם היא ספיקה, ולכן משתמשים בשפות CNF, למי שלא זוכר – CNF זה בדיוק ההיפך מ-DNF. ומי שזה עדיין רק מצלצל מוכר, נסביר שה-CNF הוא צורת שרשור של פסוקיות (תתי נוסחאות לוגיות) המקיימות לאורך כל הנוסחה את התצורה האחידה של חיבור פסוקיות שבתוכם כל הליטרלים מחוברים בקשר OR, ושרשור של מספר פסוקיות עם קשר AND ביניהם. לדוגמא: $(X\vee Y)\wedge(Z\vee W)$היא צורת CNF מסדר שני. הסיבה שאנחנו מחפשים דווקא CNF מסדר שלישי (המכונה אצלנו 3-CNF), היא מאחר שCNF מדרגה ראשונה ושניה שייכים למחלקה P, ורק מ 3-CNF ומעלה, אנחנו מתחילים להגיע לנוסחאות שהן NP-שלמות.

מה היא בעצם תצורה של 3-CNF? הוספת מגבלה נוספת על הפסוקיות, שכל אחת מהן תכיל בדיוק שלושה ליטרלים. לא משנה כמה פסוקיות יש לנו בנוסחה הסופית, כל עוד אנחנו שומרים על הכלל הזה, אנחנו בסדר.

ננסה להוכיח שבעיית ספיקות הנוסחאות הבוליאניות 3-CNF היא NP-שלמה.

אין צורך אפילו לכתוב את ההוכחה שבעיה זו שייכת ל-NP, מאחר וזה בדיוק אותה הוכחה של SAT, בסך הכל אנחנו מדברים על מקרה פרטי של SAT. אבל בכל זאת, מה שכן נדרש עלינו הוא להוכיח רדוקציה $SAT \leq_P$ 3-CNF ועל ידי זה נקבע את הבעיה בתור NP-שלמה.

את אלגוריתם הרדוקציה, נבצע באופן שונה למה שעשינו, אך הוא מתחבר גם לצורת ספיקת המעגל (במובן מסוים).

סדר הרדוקציה יהיה כזה:

1. **עץ פיסוק בינארי** – בשלב הראשון, נבנה עץ "פיסוק" בינארי, בו הקשרים יהיו קדקודי העץ, והליטרלים יהיו העלים, כאשר כל כניסה לסוגריים בין קשרים תפריד אותנו לשני תתי-עץ.
2. **רדוקציה של ספיקת נוסחאות** – לאחר שיש לנו את העץ, אנחנו עושים לו רדוקציה דומה לזו שעשינו עם ספיקות הקשרים, וניצור לנו את כל הפסוקיות הקיימות.

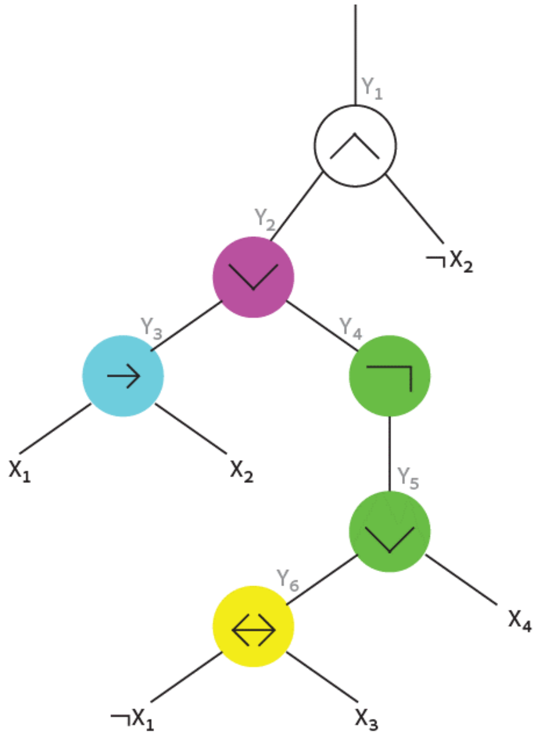
3. **טבלת אמת** – השלב הבא הוא קצת שחור ולא כיפי, עבור כל פסוקית נבנה טבלת אמת, ונבדוק מתי התוצאה של הפסוקית הנוכחית הוא ספיק ומביא לנו 1, ומתי הוא מביא לנו 0.
4. **בניית DNF** – עכשיו, על מנת ליצור צורת CNF של הפסוקית, ניקח את כל תוצאות ה-0 של הטבלה ונרשום אותם בצורת DNF – כלומר, הליטרלים יהיו מאוחדים עם AND, והפסוקיות יקושרו ב-OR. דבר זה ייתן לנו את הנוסחה ההופכית ל-CNF
5. **דה מורגן** - המרה של כל הנוסחאות לצורת CNF.
6. **וידוא שלישיות** – שינוי הפסוקיות שקיבלנו לצורה שתתאים לנו לדרגה שלישית של CNF.

על מנת לראות את כל השלבים, גם אם לא במלואם, נראה את הדוגמאה הבאה מתוך הספר –

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

עץ פיסוק בינארי

נתחיל בפירור הנוסחה לעץ בינארי. נזכיר – הקשרים יהיו לקדקודים, והליטרלים לעלים. מבחינת חלוקת הפיצול של העץ, אנחנו יכולים לראות שבסופו של דבר, יש לנו פה שני חלקים עיקריים לנוסחה – הסוגריים הסגולות, שמכילות תתי-נוסחאות, והצד השני שהוא מכיל רק ליטרל בודד. נפרק את כל המרכיבים ונקבל את העץ הבא –



הקדקודים צבועים כמובן לפי הסוגריים שתוחמות כל חלק מהנוסחה, ככה קצת יותר נוח לקרוא ולהבין את ההמרה.

רדוקציה של ספיקת נוסחאות

עכשיו אנחנו עובדים בדיוק באותה צורה שעשינו בספיקת המעגלים, כל מוצא יסומן בגרף באות המתאימה (למעלה זה האותיות y בשביל שנבדיל ביניהם ובין הקלט של הקדקוד), ונוציא מכל קדקוד את הנוסחה בצורה של אם"ם ביחס הקלט-פלט. למשל, עבור הקדקוד הסופי, יוצא ממנו y1, שתלוי בפלט y2 וגם

בהופכי של x2, ולכן יתורגם ל- $(y_2\wedge\neg x_2)\leftrightarrow y_1$. נמשיך ונעשה את זה עבור כל הקדקודים השונים, ע שנקבל את הנוסחה ϕ הבאה –

$$\phi = y \wedge (y_1 \leftrightarrow (y_2\wedge\neg x_2))$$
$$\wedge (y_2 \leftrightarrow (y_3\vee y_4))$$
$$\wedge (y_3 \leftrightarrow (x_1\rightarrow x_2))$$
$$\wedge (y_4 \leftrightarrow \neg y_5))$$
$$\wedge (y_5 \leftrightarrow (y_6\vee x_4))$$
$$\wedge (y_6 \leftrightarrow (\neg x_1\leftrightarrow x_3))$$

עכשיו מגיע החלק הכיפי, שקורמן החליט שאין צורך להראות את כולו-

טבלת אמת

עבור כל פסוקית בנוסחה ϕ שמצאנו אנחנו בונים טבלת אמת. קצת מייגע, אבל זאת הדרך שלנו לחלץ את הגרירות השונות לתצורה של DNF. כמובן שאפשר להשתמש בכל חוקי ההמרה הלוגיים, אבל אני לא ממש בטוח שזה יעשה את העבודה יותר קלה. בכל אופן, בואו נתחיל, ונראה איפה נישבר-

y_1	y_2	x_2	$y_1\leftrightarrow (y_2\wedge \neg x_2)$
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

y_2	y_3	y_4	$y_2\leftrightarrow (y_3 \vee y_4)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

y_3	x_1	x_2	$y_3 \leftrightarrow (x_1\rightarrow x_2)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

y_4	y_5	$y_4 \leftrightarrow \neg y_5$
0	0	0
0	1	1
1	0	1
1	1	0

y_5	y_6	x_4	$y_5 \leftrightarrow (y_6\vee x_4)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

y_6	x_1	x_3	$y_6 \leftrightarrow (\neg x_1\leftrightarrow x_3)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

עכשיו סיימנו לעשות טבלת אמת עבור רק שש פסוקיות. השלב הבא הוא זה שמתחיל לעשות את הנוסחה הסופית ארוך ומייגע.

בניית DNF

עבור כל טבלה, ניקח את כל התוצאות 0, ונרכיב מהם את פסוקיות הDNF הדרושות לנו – כלומר, כל טבלה מכונה בשם הנוסחה, למשל $y_5 \leftrightarrow (y_6 \vee x_4)$, טבלת האמת שלה זה ϕ'_5. על מנת ליצור CNF שיקיים אותו, נצטרך לבנות את הDNF ההופכי של הטבלה, כלומר את $\neg\phi'_5$.

y_5	y_6	x_4	$y_5 \leftrightarrow (y_6 \vee x_4)$
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0

עבור זה, נוציא את הנוסחה הבאה (נזכור שאנחנו מבצעים פה היפוך, וכל 1 שמופיע אנחנו ממירים אותו ל-0):

$$(y_5 \wedge y_6 \wedge \neg x_4) \vee (y_5 \wedge \neg y_6 \wedge x_4) \vee (y_5 \wedge \neg y_6 \wedge \neg x_4) \vee (\neg y_5 \wedge y_6 \wedge x_4)$$

דה מורגן

מה שנותר לנו להגיע לצורת CNF היא ההמרה של דה-מורגן שמשנה את כל הסימנים וכל הקשרים, ונקבל את התצורה הבאה:

$$CNF\text{-}\phi''_5 = (\neg y_5 \vee \neg y_6 \vee x_4) \wedge (\neg y_5 \vee y_6 \vee \neg x_4) \wedge (\neg y_5 \vee y_6 \vee x_4) \wedge (y_5 \vee \neg y_6 \vee \neg x_4)$$

הידד לנו! הוצאנו נוסחה אחת מתוך הטבלה, תרשו לי לעצור פה ולא לבצע את כולם, הרעיון מובן.

נותר לנו השלב האחרון ברדוקציה-

וידוא שלישיות

אמנם השגנו CNF, ובמקרה הוא יצא לנו גם 3-CNF. אבל תמיד יכולים להיות מקרים בהם יהיה לנו פסוקיות בגודל קטן יותר (לא יכול להיות גדול יותר, מאחר ועבדנו על עץ בינארי, אז יכול להיות לנו מקסימום שני קלטים ופלט אחד, גאוני!).

על מנת לטפל במקרים השונים, מה שנעשה הוא כדלקמן:

- אם יש לנו פסוקית של שני ליטרלים (כמו למשל ϕ'_4 בדוגמה שלנו), לאחר שנוציא את הCNF נכפיל כל פסוקית פעמיים, ונוסיף לכל אחד מהן ליטרל שהוא מחוץ למסגרת – למשל p, ובכל הכפלה נשים אותו פעם בתור עצמו ופעם בתור הופכי, ובך נשמור על התצורה. למשל אם יצאה לנו הנוסחה $(\neg y_5 \vee y_6)$, נעשה לה המרה ולצורה הבאה –$(\neg y_5 \vee y_6 \vee \neg p) \wedge (\neg y_5 \vee y_6 \vee p)$, כך שלא משנה מה נגדיר את p, זה שקול לטאוטולוגיה.
- אם יש לנו ליטרל בודד, אז עושים בדיוק את אותו רעיון, רק שמכפילים את הליטרל ארבעה פעמים, ובכל פעם מוסיפים שני ליטרלים בצורה מותאמת על מנת למלא את החלל בפסוקיות אמת.

אחרי כל זה – סיימנו עם הרדוקציה, עכשיו רק נותר לוודא שהיא אכן מקיימת.

בשלבי הבנייה השונים, דאגנו להשאיר תמיד את כל הנוסחאות וכל המעגלים שיהיו ספיקים בצורה שקולה. כלומר, בניית הטבלה היא בוודאי שקולה לנוסחה המקורית (אותה כבר הוכחנו שהיא מקיימת רדוקציה). בניית הDNF מהטבלה היא שקילות בוודאי כי זה רק קריאת נתונים, והפעלת דה מורגן, בוודאי שמתקיימת, ולכן נשארנו באותו מקום – הנוסחה הסופית שנייצר ספיקה אם ורק אם הנוסחה המקורית ספיקה.

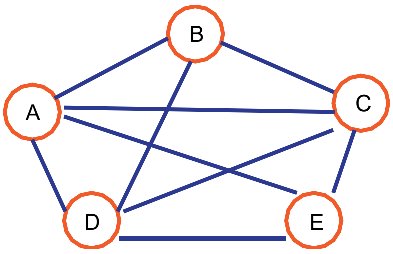
מבחינת חישוב בזמן פולינומיאלי, במקרה הגרוע ביותר, נשים בסופו של דבר 4 ליטרלים על כל ליטרל מקורי של הנוסחה (וגם זה רק בהנחה שאיך שהוא יצאו לנו רק ליטרלים בודדים בפסוקיות), וגם אם זה קרה, עדיין הוספת הליטרלים נשמרת תחת זמן פולינומי ולא מעריכי.

מש"ל.

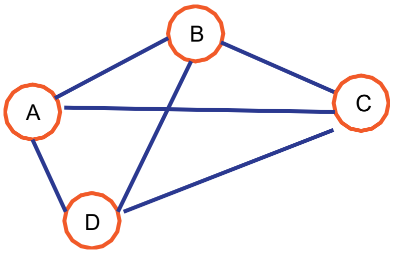
בעיית הקליקה Clique

בעיית הקליקה מתחילה להראות לנו את השימושים המטורפים שעושים בעזרת רדוקציות. אם עד עכשיו עברנו על דברים שנראים אותו דבר, ברעיון גם אם לא בפועל, כאן אנחנו עושים דילוג עצום לעבר בעיות חדשות בעזרת בעיות שהוכחנו קודם.

בעיית הקליקה מוגדרת כך – עבור גרף בלתי מכוון $G(V,E)$, $V' \subseteq V$ היא קבוצת קדקודים בה כל זוג קדקודים מחובר על ידי קשת ב-E, כלומר יש פה תת גרף שלם (שלם מבחינת ההגדרה – יש קשתות בין כל הקדקודים) של G. דרגה של קליקה מוגדרת כמספר הקדקודים המחוברים ביניהם. לדוגמא, נסתכל על הגרף הבא:



עבור גרף זה, קיימת לנו קליקה בדרגה 4, נסתכל על תת הגרף המתאים:



ניתן לראות שכל קדקוד מחובר לכל שאר האחרים בגרף. חדי העין, ישימו לב שיש פה עוד קליקה בדרגה 4, $\{A,C,D,E\}$, אבל אנחנו כידוע מחפשים אם יש פתרון, ולא כמה פתרונות יש.

בעיית אופטימיזציה תהיה "מה הקליקה הגדולה ביותר בגרף?", בדיקה של דבר כזה היא כמובן מעריכית, ואנחנו מחפשים את בעיית ההכרעה המתאימה – בהינתן גרף ומספר קבוע, האם יש בגרף קליקה בגודל הדרוש? ובצורה פורמלית:

CLIQUE = { <G,k> : G הוא גרף המכיל קליקה בגודל k }

האלגוריתם הנאיבי לפתירת הבעיה יעבוד באופן הבא – נסדר רשימה של כל התת-קבוצות האפשריות בגרף, ונבדוק עבור כל אחת מהן האם תת הקבוצה הנתונה היא קליקה. אם ננסה להסתכל על זמן הריצה של אלגוריתם נאיבי כזה, אז יש לנו קודם כל חלוקה של תת הקבוצות שזה $\binom{|v|}{k}$, ועבור כל תת קבוצה (בהנחה המוצלחת יותר שיש לנו מטריצת סמיכויות) נצטרך לרוץ k^2. כלומר סה"כ - $\Omega(k^2\binom{|v|}{k})$.

עכשיו, אם מדובר על k שהוא גבוה מאוד עד כדי שהוא מתקרב ל-|V|, מספר התת קבוצות יהיה יחסית קטן, באופן דומה אך שונה, אם k יהיה קטן מאוד ושואף ל-0, אז אמנם יהיו מלא תת קבוצות, אבל זמן הריצה שלהם יהיה מאוד קצר. בשני המקרים האלה אנחנו מצליחים לשמור על האלגוריתם הזה תחת זמן פולינומי, אבל בכל מקרה אחר, שיש לנו גם הרבה תתי-קבוצות, וגם מעבר בדיקה ארוך עליהם, זמן הריצה קופץ להיות מעריכי.

אך אל דאגה! אנחנו טוענים כי בעיית הקליקה היא NP שלמה (זה לא שאין דאגה, רק טיפה פחות).

בשביל להוכיח זאת, נעבוד בסדר הפעולות הידוע:
ראשית, נוכיח שהבעיה בכלל נמצאת ב-NP, אחרת היא סתם קשה. אנחנו מחפשים קשה עם טוויסט.

עבור גרף נתון G וקבוע k כלשהו, אנחנו יכולים לקבל מחרוזת אימות y שתהיה סדרה של קדקודים, ונצטרך לבדוק אותה תחת זמן פולינומי. איך נעשה זאת?

1. קודם כל נספור את הקדקודים בשביל לוודא שכמות הקדקודים מתאימה ושווה ל-k זה כמובן יהיה $O(k)$.
2. נתפוס כל קדקוד בנפרד, ונוודא שיש לו קשתות עבור כל שאר הקדקודים $O(k^2)$.

אם שתי הבדיקות עברו בהצלחה, אז $CLIQUE \in NP$.

עכשיו החלק היותר מסובך – על מנת להוכיח כי בעיית הקליקה היא בעייה קשה ב-NP, נוכיח כי $3\text{-}CNF\text{-}SAT \leq_p CLIQUE$ – כלומר, נעשה רדוקציה מסיפוק תלת-פסוקיות לבעייה הזו! זה נראה לא קשור בכלל אחד לשני, אבל נבנה את הגרף בצורה מאוד מוגדרת וברורה על מנת שהרדוקציה תהיה מספקת.

על מנת לבנות את הגרף באופן מתאים, אנחנו ניקח נוסחה העומדת בתנאי 3-CNF – שרשור של פסוקיות CNF, שכל אחת מהן מורכבת משלושה ליטרלים. ניקח את הנוסחה $\phi = C_1 \wedge C_2 \wedge .. \wedge C_k$ ונבנה עבורה גרף, שבתוכו יהיה קליקה בגודל k הרצוי.

כל ליטרל שנדבר עליו, יסומן באופן הבא – l_1^r, כך ש-l מייצג ליטרל (literal), המספר העליון r ממספר את הפסוקית המתאימה עליה אנחנו מדברים, כך ש- $1 \leq r \leq k$. והמספר התחתון, הוא מיקום הליטרל בתוך הפסוקית בין 1 ל-3.

אופן הבנייה יהיה כזה –

- עבור כל פסוקית C_r, נכניס את שלושת הליטרלים בקדקודים, ונסמן על כל אחד מהם את ערך הליטרל.
- את הקשתות נמתח בין כל הקדקודים תחת שני תנאים:

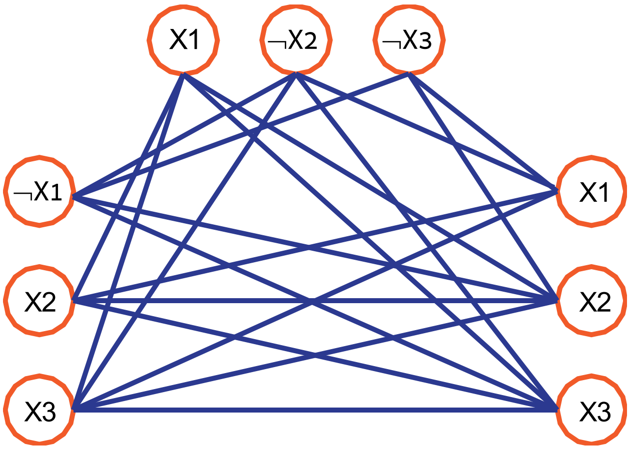
1. הקשתות יהיה רק בין כל קדקוד לכל שאר הקדקודים מחוץ לפסוקית. כלומר, בין הקדקודים באותה שלישיה, לא יהיו קשתות.

2. מעבר לקדקודים באותה פסוקית, לא נמתח גם קשתות בין שני ליטרלים שסותרים זה את זה. כלומר, אם $l_2^3 = x_2$, $l_3^4 = \neg x_2$ – לא תעבור ביניהם קשת.

נבנה גרף כזה בשביל להבין את צורת הבנייה, ואז נראה כיצד ממשיכים. עבור הבניה ניקח את נוסחת 3CNF הבאה:

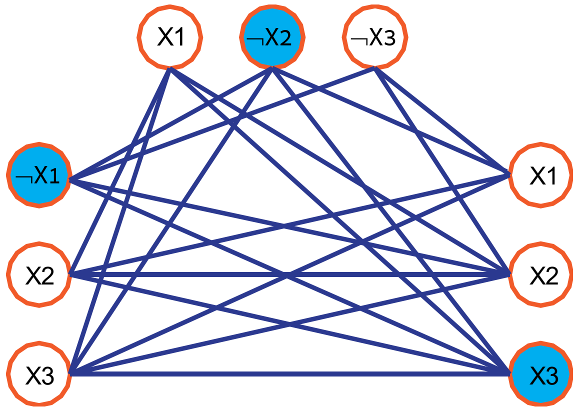
$$\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge(\neg x_1 \vee x_2 \vee x_3) \wedge(x_1 \vee x_2 \vee x_3)$$

ועבור הנוסחה הזאת, נבנה את הגרף על פי התנאים שאמרנו, ונקבל את הגרף המופלא הבא, בו אנחנו טוענים שתהיה קליקה מסדר 3 (שזה כמובן כמות הפסוקיות הנתונות לנו בנוסחה):



השלישיה הראשונה היא C_1, השמאלית היא C_2, והימנית היא C_3, האם יש לנו כאן קליקה מסדר 3? קל לראות שכן, ואפילו יותר. אבל עם קל-לראות לא הולכים למכולת, ולכן נסביר.

קודם כל נראה את הרדוקציה שהיא מספקת את הדרישות. נניח כי ϕ היא נוסחה שניתן לספק עבור השמה מסוימת, כלומר עבור הנוסחה שנתנו, אם ניקח את ההשמה הבאה $<x_1 = 0, x_2 = 0, x_3 = 1>$, אז התוצאה של הנוסחה תהיה 1. עכשיו נראה איך זה מתקיים בגרף שיצרנו.



צבענו את הפסוקיות שמקבלות את ההשמה המתאימה, וניתן לראות כי בנינו קליקה מתאימה.

מאחר וכל הפסוקיות מחוברות ביניהם עם AND, ובתוך הפסוקיות הליטרלים מחוברים עם OR, זה מכריח אותנו שלכל פסוקית תהיה לפחות השמה אחת של 1. אם יהיה כך, כל פסוקית בנפרד תפיק לנו 1, והנוסחה תסופק (כמובן שיכול להיות יותר, אבל זה מספיק לנו). כמובן שכל השמה היא חח"ע, ולא יכול להיות מצב בו x_i מסוים הוא בו זמנית גם 0 וגם 1. אם נסתכל בגרף ובחוקי הבניה שלו, גם לא יכול להיות שיש לנו

קשת מליטרל מסוים לשלילה שלו, ולכן עבור כל $l_i^r, l_j^s \in V$, כאשר $r \neq s$, וכאשר ברור לנו מחוקי הבניה שלא מדובר על שלילה אחד של השני, יש לנו קשת ששייכת לקליקה.

בכיוון ההפוך, לו נתון לנו גרף המכיל השייך לבעיית הקליקה, ובנוי בצורה של שלישיות, אז אם יימצא שבגרף זה אכן יש קליקה, ניתן לקחת כל קדקוד ששייך לקליקה ולתת לו ערך 1 בנוסחה שנבנה, ואין לנו צורך לחשוש מהשמה כפולה של משתנים, מאחר והשלילה של הליטרל איננה שייכת לקליקה, ואנחנו מסוגלים לספק גם את ϕ. הוכחנו כי CLICQUE ∈ NPH.

ומאחר וקיימנו את שני התנאים, אזי הוכחנו כי CLICQUE ∈ NPC

הערה: ישאל השואל (ובצדק) שהבניה שעשינו היא מאוד ספציפית ומקרה פרטי, אבל מה נעשה אם כל שאר המקרים הקיימים? דבר ראשון, אנחנו לא צריכים להוכיח את כל המקרים, אלא רק להראות שיש דרך לעבור מסיפוק הנוסחאות ליצירת קליקה. דבר שני, אם נרצה לממש עבור כל גרף, נוכל לעשות רדוקציה מהבעיה הנוכחית ולעבור לבעיות אחרות, ושיהיה לנו בהצלחה.

בעיית כיסוי הקדקודים Vertex-Cover

בעיית כיסוי הקדקודים, היא בעיה שממשיכה את הקו של בעיית הקליקה (והרדוקציה גם נעשית מהקליקה), ומתוארת כך – עבור גרף נתון G(V,E), קיימת תת-קבוצה V'⊆V, כך שבעבור כל קשת (u,v) ∈E, u ∈ V' או v ∈ V'. כלומר, אנחנו מחפשים תת-קבוצה של קדקודים V', שכל קשת שקיימת בגרף, מחוברת באחד מהקצוות שלה לקדקוד ב-V'.
נדגיש שוב, אנחנו לא מדברים על בעיית אופטימיזציה, אלא על בעיית הכרעה – בהינתן גרף, האם יש לנו קבוצה של k קדקודים שמכסה את כל הקשתות בגרף. נשאל זאת שוב בצורה קצת יותר פורמלית:

VERTEX-COVER = { <G,k> : לגרף G יש כיסוי-על-ידי-קדקודים שגודלו k}

אנחנו ננסה להוכיח, כמובן בהצלחה, כי VERTEX-COVER ∈ NPC.

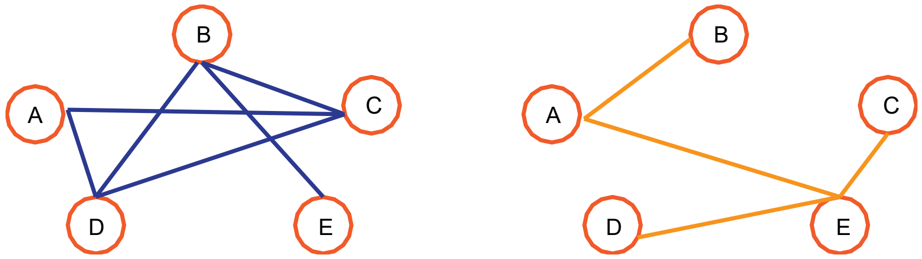
כנהוג, נתחיל בהוכחה כי VC ∈ NP. האימות תחת זמן פולינומיאלי, כמובן יהיה בדומה לקליקה – נקבל רשימת קדקודים, נוודא שזה מתאים ל-k הדרוש, ונבדוק שקבוצת הקדקודים מכילה את כל הקשתות מתוך E. דבר כזה כמובן לא יעלה הרבה מעבר לפולינומיאלי.

עכשיו נוכיח כי VC ∈ NPH, על ידי רדוקציה $\text{CLIQUE} \leq_p \text{VERTEX-COVER}$.

מאחר ושתי הבעיות מגיעות מעולם הגרפים, המעבר מאחר לשני לא מפתיע אף אחד (לפחות לא כמו מנוסחאות CNF לגרפים), אבל האופן בו הרדוקציה נעשית, זה כבר משהו לא פחות ממדהים.

את בסיס הרדוקציה אנחנו נעשה על בסיס גרף המשלים, ונראה כי לגרף יש כיסוי על ידי קבוצה של קדקודים, אם"ם בגרף המשלים של הגרף הנתון, יש לנו קליקה בגודל V-k על ידי הקדקודים שאינם בכיסוי. הבנייה של הגרף המשלים תהיה כמובן זמן פולינומיאלי, בו נבנה מטריצת סמיכויות חדשה, ובכל מקום בו אין קשת, שפוט נוסיף אחת במטריצה החדשה.

קחו רגע לעכל את זה, ונראה את זה עם דוגמא:



הגרף השמאלי G הוא גרף בעל חמישה קדקודים עם קליקה בדרגה 3. ניקח את G, ונבדוק מהו הגרף המשלים שלו $\bar{G}$. בגרף הימני המשלים, ניתן לראות שמתוך חמשת הקדקודים, שניים מתוכם (שזה בדיוק 5-3!!! מי היה מאמין??) מקושרים לכל הקשתות הקיימות בו. אפשר לבדוק את זה עם עוד גרפים, גם כאלה לא קשירים וזה תמיד יהיה נכון. מדהים. עכשיו נוכיח את זה.

נתחיל מהקליקה לכיסוי-קדקודים.
נניח כי נתון לנו גרף G שמכיל קליקה V', כאשר $|V'| = k$.
טענה: V-V' מהווה כיסוי קדקודים עבור $\bar{G}$.
הוכחה: עבור כל קשת $(u,v) \in \bar{E}$, מתקיים כי $(u,v) \notin E$. למשל בדוגמה למעלה, הקשת (a,b) נמצאת רק בגרף המשלים.
כלומר – לפחות אחד משני הקדקודים u או v **לא שייכים** לקליקה V'. כי אם שניהם היו בקליקה, אז היתה ביניהם קשת. הקדקוד b שייך לקליקה {b, c, d}, אליה הקדקוד a לא שייך.
באופן שקול – לפחות אחד משני הקדקודים u או v **שייכים** ל-V-V', כי הם חייבים להיות איפשהו. הקדקוד a שאינו בקליקה עליה דיברנו, שייך לקבוצת הכיסוי {a,e} שהיא בדיוק המשלים של קדקודי הקליקה.
כלומר הקשת (u,v) מכוסה על ידי V-V'. כמובן על ידי הגבלת הכלליות, וכל השקרים שאנחנו אומרים בשביל להגיד שזה מתקיים לכל הקדקודים ולכל הקשתות. ומכאן – הקבוצה V-V' שגודלה $|V| - k$ מהווה כיסוי-על-ידי-קדקודים עבור $\bar{G}$.

בכיוון ההפוך, נניח כי ל-$\bar{G}$ יש כיסוי-קדקודים על ידי $V' \subseteq V$, כאשר $|V'| = |V| - k$.
לפיכך – עבור כל זוג קדקודים $u, v \in V$, אם $(u, v) \in \bar{E}$, אזי $u \in V'$ או $v \in V'$ או שניהם. ניקח למשל את {c, e} בגרף הכיסוי. הקשת $(c, e) \in V'$. ואכן הקדקוד $e \in \bar{E}$.
ולהיפך ניתן לראות, כי לכל $u, v \in V$, אם $u \notin V'$ וגם $v \notin V'$ אז $(u, v) \in E$ והצלע הזאת היא חלק מהקליקה V-V', שגודלה $k = |V| - |V'|$. כלומר אם ניקח את הקדקודים {d,c} אף אחד מהם אינו שייך ל-V', אבל הקשת $(d, c) \in E$ והיא חלק מהקליקה.

הוכחנו שהרדוקציה מתקיימת, כלומר VERTEX-COVER ∈ NPH.

ומשילוב שני ההוכחות אנחנו יכולים לומר כי VERTEX-COVER ∈ NP. מש"ל

בעיית סכום התת-קבוצה Subset-Sum

בעיית סכום התת-קבוצה, היא בעיה אריתמטית, ומוגדרת כך – בהינתן סדרת מספרים S, וקבוע t (הפתעה! זה לא k), האם ניתן למצוא תת סדרה S', שסכום האיברים שלה הוא t.

אנו מוכיחים כי SUBSET-SUM ∈ NPC.
אימות של תת קבוצה שכזה, הוא בוודאי על זמן פולינומי. מקבלים תת סדרה, ובודקים אם היא קיימת ב-S ואם סכומה שווה t.

עבור הוכחת הקיימות של SUS ב-NPH, עושים זאת על ידי בניית רדוקציה $\leq_p$ VERTEX-COVER SUBSET-SUM. אבל כאן יש לנו בשורות טובות – ההוכחה והבנייה לא חשובות למבחן (לחיים אולי כן, אבל לא למבחן). מי שממש רוצה זה עמוד 510-512 בכרך ב' של הפתוחה. תהנו!

בעיית המעגל ההמילטוני Hamilton's Cycle

מי שיסתכל בעץ שנעשה על הרדוקציות, יכול לראות שסכום התת-קבוצה מסיים לנו ענף אחד של ההוכחות, והענף השני מתחיל בבעיית המעגל ההמילטוני.
קל להוכיח את הבעיה בשייכת לNP (מקבלים את המסלול המעגלי, בודקים שהמסלול קיים ועובר בכל הקדקודים, בודקים שהוא עובר רק פעם אחת בכל קדקוד)
את הרדוקציה אנחנו עושים על בסיס ה 3-CNF-SAT $\leq_p$ HAM-CYCLE, ובונים את הגרף עם המעגל באופן שהוא דומה קצת לקליקה, אבל הרבה יותר מסובך. כמה יותר מסובך? לא צריך לדעת אותו, אלא רק לדעת שהוא קיים. מש"ל.

בעיית הסוכן הנוסע The Traveling Salesman

הבעיה האחרונה עבור הרדוקציות האלה, כל השאר אנחנו צריכים לעשות לבד. בעיה זאת קשורה מאוד לבעיית המעגלים ההמילטוניים, ואכן הרדוקציה נעשית ישירות ביניהם. הבעיה מוגדרת כך – סוכן מכירות רוצה לעבור בכמה שיותר ערים נתונות. לצורך העניין, הוא מתכנן לעבור ב-n ערים שונות ולסיים בחזרה בעיר הראשונה, באשר הגרף המכיל את רשת הערים, הוא גרף שלם – כלומר, יש דרך להגיע מכל עיר לכל עיר אחרת, אך כל מסלול בין שני ערים נתון עם משקל שונה. בעיית אופטימיזציה תהיה כמובן, מה הדרך המהירה ביותר שהוא יכול למצוא, אך הסיבוכיות של מציאת מסלול כזה הוא בערך !n. אנחנו מחפשים בבעיית הכרעה את השאלה "האם ניתן לעבור את המסלול הנ"ל תחת k מסוים?". האופן הפורמלי לשפה של בעיית ההכרעה הוא:

TSP = { <G, c, k > : G(V,E) הוא גרף שלם,
c היא פונקציה מ- V*V → Z,
$k \in Z$,
G מכיל נתיב עבור הסוכן הנוסע שעלותו לכל היותר k }

נוכיח כי בעיית הסוכן הנוסע היא NP שלמה.

נראה כי TSP ∈ NP.
בהינתן מופע של הבעיה ומחרוזת אישור של n קדקודי המסלול לפי סדר-
1. נאמת כי הסדרה מכילה את כל הקדקודים בגרף.
2. שהיא מכילה אותם רק פעם אחת.
3. שיש מסלול מעגלי בין כל הקדקודים הנ"ל.
4. שהוא אינו עולה על k.
כל זה ניתן לעשות תחת זמן פולינומי. נמשיך.

נראה כי TSP ∈ NPH על ידי הרדוקציה HAM-CYCLE $\leq_p$ TSP.

על מנת לבנות את הרדוקציה, ניקח גרף G(V,E) מתוך השפה של המעגלים ההמילטוניים, ונעתיק אותו אלינו אל גרף שלם חדש G'(V', E'), תוך כדי שאנחנו ממשקלים אותו באופן הבא:

אם הקשת $(i, j) \in E$ נגדיר את המשקל 0.
אם הקשת $(i, j) \notin E$ נגדיר את המשקל 1.

כלומר, נבנה גרף שלם (שזה הדרישה של בעיית הסוכן הנוסע, מעגל המילטוני לא חייב להיות שלם), שבל הצלעות שנלקחו מהמילטון, ימושקלו ב- 0, ובל השאר ב-1. בנייה של גרף היא כמובן תחת זמן פולינומיאלי, ולכן זה מקיים את הדרישה לזמן-פולינומיאלי.
מה שזה ייתן לנו, הוא שאם אכן יתברר כי יש מעגל המילטוני בגרף, אז המסלול הקצר ביותר לסוכן הנוסע הוא במשקל 0. ובביוון ההפוך – אם יש לסוכן הנוסע איזשהו מסלול שמשקלו הוא 0, אזי אנחנו יודעים שאנחנו יכולים לקחת בחזרה את הגרף G מתוך G', ויהיה לנו בו מעגל המילטוני. ומכאן הרדוקציה מתקיימת, מה שגורר כי TSP ∈ NPH.

ובשילוב שתי ההוכחות, הוכחנו כי TSP ∈ NPC. מש"ל.

מחלקת coNPC – המשלימה ל-NPC

ראינו בתרשים המחלקות הגדול, שישנה מחלקה שהיא המשלימה לNP, כלומר בעיות שניתן למצוא להם אלגוריתם **שלילה** תחת זמן פולינומיאלי. גם למחלקה NPC יש את המחלקה המשלימה, שלא מדברים עליה הרבה, אך חשוב להבין אותה לחיים, ולא פחות חשוב – למבחן.

הרעיון העומד מאחורי מחלקה זו, הינן הבעיות שהן השלילה של המחלקה NPC. הבעיה המוכרת ביותר השייכת למחלקה זו הינה בעיית הטאוטולוגיה המכונה TAUT. נסתכל על הבעיה, ועל דרך הבנייה, ונקווה להבין מזה.

שאלות שיופיעו בהקשר זה בדרך כלל יהיו בעיות שאנחנו מכירים בתור NPC, כאשר עם שינוי הנוסח אנחנו נהפוך אותן בדיוק לקבל את כל הדברים ההפוכים מהשפה המקורית.

בעיית הטאוטולגיה TAUT

בהינתן נוסחה לוגית ϕ, האם הנוסחה הנתונה היא טאוטולוגיה?

בזכור משיעורי לוגיקה, נוסחה טאוטולוגית, הינה נוסחה שעבור כל השמה שלא נציב בה, התשובה תהיה True. כמובן, שדבר כה קשה מאוד להוכיח – ככל שגודל הנוסחה עולה, וכמות המשתנים עולה, אזי הקומבינציות האפשריות עבור אותה בעיה גדל בהתאם בצורה מעריכית. ביצד נוכל להוכיח שבעיה היא טאוטולוגיה? מסתבר שאי אפשר. אבל מה שאנחנו כן יכולים להוכיח זה, אבור נוסחה מסויימת האם היא לא טאוטולוגיה. איך עושים את זה? פשוט! מוצאים השמה ששוללת את הנוסחה. כלומר אם נגדיר את השפה NOT-TAUT, אזי זה ייראה באופן הבא –

NOT-TAUT = {ϕ | כאשר ישנה השמה עבורה הנוסחה אינה ספקה}

כלומר – המילים המתקבלות בשפה, הן דווקא אלו **שניתן להוכיח בהן שלילה**. זה קצת מבלבל, אבל אם קוראים את זה לאט, יש בזה היגיון.

אם כן – לאן שייכת הבעיה NOT-TAUT? נוכיח כי NOT-TAUT ∈ NPC.

בנוהל הרגיל – נוכיח כי הבעייה שייכת ל-NP, ואחרי זה נוכיח שהוא גם NPH. לא נעשה את כל ההוכחה בשלמותה, כי הדבר החשוב פה הוא ה"בן-טאוטולוגיה", ולא הצד השני.

1. NOT-TAUT ∈ NP-

עבור האימות, נוכל לקבל מופע x של הבעיה, ורצף אימות y, שיכיל השמה עבור הנוסחה x שבעבורה נוציא false. כמובן שזה יספיק לנו בשביל לשייך את המופע לשפה, וקל לעשות את זה תחת זמן פולינומיאלי.

2. NOT-TAUT ∈ NPH

נעשה רדוקציה $SAT \leq_P$ NOT-TAUT

כזכור, בעיית SAT רוצה למצוא האם יש השמת **אמת** לנוסחה נתונה, על מנת לעשות את הרדוקציה, פשוט נתחום את כל הנוסחה בסוגריים, ופשוט נוסיף בחוץ NOT. מה זה ייתן לנו? לו נצליח לפתור בצורה כלשהי את הנוסחאות בSAT, באופן סימולטני זה יכניס לנו גם את אותן נוסחאות (בשלילה) לשפה NOT-TAUT.

עד כאן הוכחנו שהשפה NOT-TAUT ∈ NPC.

כלומר הבעיה שמוצאת את **כל הנוסחאות שאינן טאוטולגיה**, היא NPC, אז השלילה של השפה הזאת – כלומר **כל הנוסחאות שהן כן טאוטולוגיה** היא השפה המשלימה ולכן שייכת ל-coNPC.

מש"ל.

שאלות הרחבה

בעיית החלוקה (Partition)

נתונה בעיית ההכרעה המוגדרת כך:

$$PAR_{\text{בעיית הכרעה}} = \{ S = \{a_1..a_n\} |\ \exists S_1,S_2 \subseteq S,\ S_1 \cap S_2 = \phi,\ S_1 \cup S_2 = S,\ \sum_{a_i \in S1} a_i = \sum_{a_j \in S2} a_j \}$$

צ"ל – PAR ∈ NP-Complete

ראשית, נגדיר את הבעיה בצורה מובנת יותר – אנחנו מחפשים עבור סדרה נתונה של מספרים, האם ניתן לחלק את הסדרה לשני תתי-סדרות בחלוקה ממצה – הווה אומר, שתי תתי הסדרות הן זרות זו לזו, ואיחודם יביא לנו את הסדרה S, כך שסכום שתי תת-הסדרות יהיה שווה. שם הבעיה PAR נגזר כמובן מהמילה PARTITION.

לדוגמה, אם יביאו לנו $S = \{1,2,4,5\}$, אז נוכל לחלק את S באופן הבא: $S_1 = \{1,5\}$, $S_2 = \{2,4\}$. וכל התנאים הנ"ל יתקיימו.

ראשית, נגדיר את השפה, שבמקרה זה היא מאוד דומה להגדרת הבעיה. ההבדל הוא, שבעיית ההכרעה שזה קבוצות של הסדרות שניתן לחלק אותן תחת כל הנ"ל, והשפה היא כבר קבוצת המחרוזות הבינאריות תחת כל הכללים. את השפה PAR נגדיר-

$$PAR_{\text{שפה}} = \{ S = \{a_1..a_n\} \in \{0,1\}^* |\ \exists S_1,S_2 \subseteq S,\ S_1 \cap S_2 = \phi,\ S_1 \cup S_2 = S,\ \sum_{a_i \in S1} a_i = \sum_{a_j \in S2} a_j \}$$

כאשר באמת ההבדל הוא רק השיוך לקידוד הבינארי.

על מנת להוכיח ש PAR ∈ NPC, מוטל עלינו להוכיח שני דברים:

1. PAR ∈ NP
2. PAR ∈ NPH

(הכלל השני, שאנחנו מבקשים רדוקציה מתאימה מכל שפה, בעצם מגדיר לנו את PAR כ-NPH, ועל מנת להכליל את השפה תחת תת הקבוצה NPC, אנחנו צריכים לוודא שזה אכן תחת NP)

בכדי להראות את התנאי הראשון, נראה כי עבור כל מחרוזת שנקבל, אם נקבל עבורה אישור מתאים, נוכל לבדוק את אפשרות החלוקה תחת זמן-פולינומיאלי. את אלגוריתם האימות נגדיר בצורה הבאה – נציע עבור כל סדרת קלט $x \in \{a_1..a_n\}$, את הקלט y שיהווה אימות, כך ש $y = \{b_1..b_k\},\{b_{k+1}..b_n\}$. כלומר, האימות שנקבל לא יהיה רק סדרה בודדת כמו בבעיות מסלולים ודומיהם, אלא עבור סדרה מסוימת, נקבל את האימות המתאים של דרך החלוקה הממצה. אלגוריתם האימות שנספק עכשיו יהיה בדלקמן:

1. נבדוק ש-y הוא קלט תקין, כלומר הוא קידוד מתאים של סדרת מספרים.
2. נבדוק ש $x \supseteq \{b_1..b_k\}$, $x \supseteq \{b_{k+1}..b_n\}$
   נבדוק $x_1 \cap x_2 = \phi$
   נבדוק $x_1 \cup x_2 = x$
3. נחשב $t_1 = \sum x_1$, $t_2 = \sum x_2$
4. אם $t_1 = t_2$ נחזיר True
   אחרת, נחזיר False.

<u>הוכחת נכונות:</u> אם $x \in PAR$, ו-y אישור מתאים, אזי – שורות (1),(2) מצליחות, שורה (3) גורמת לשורה (4) להחזיר True.

<u>הוכחת זמן ריצה:</u> בדיקת הקלט תהיה תחת O(n), אימות תתי-הסדרות יהיה $O(n^2)$, וחישוב הסכום יהיה O(n) – סה"כ זמן פולינומיאלי $O(n^2)$.

ומכאן $PAR \in NP$. מש"ל.

בעת נוכיח כי $PAR \in NPH$. מאחר שיהיה לנו מסובך להוכיח מחדש עבור כל שפה את הקיימות שלה בNPH, השיטה שאנו עובדים עליה היא לקחת שפה שכבר הוכחנו, ולקשור אותה לשפה הנתונה ברדוקציה. עבור הרדוקציה ניקח את השפה SUBSET-SUM (להלן SUS), המוגדרת באופן הבא:

$$SUS = \{<S=\{a_1..a_n\}>, k \mid \exists S' \subseteq S, \sum S' = k\}$$

בלומר עבור סדרה נתונה S, וערך קבוע k, השפה מקבלת רק מילים המכילים תת-סדרה שסכומה הוא k. למשל $S=\{2,4,6,8\}$, $k=14$, שייכת לשפה SUS.

עכשיו נוכיח כי $SUS \leq_p PAR$. ועל ידי זה יוכח $PAR \in NPH$.

רעיון הרדוקציה – בהינתן קלט מתאים עבור השפה SUS וערך k, נבנה מחרוזת חדשה שתכיל את כל המחרוזת המקורית, שאנו יודעים שעלול להיות בה תת-סדרה שסכומה הוא k. ונוסיף לסדרה הזאת את הדרוש, על מנת שתת-הסדרה השניה, יהיה שווה גם הוא k.

נראה דוגמה שתסביר את הרדוקציה, ואז נמשיך בפורמליות-

נניח קיבלנו כי $k=452$, $S=\{137, 42, 271, 103, 154, 16\ 3\}$.

עבור הסדרה הנתונה, שסכומה הכולל הוא 726, קיימת תת הסדרה $S' = \{137,42, 103, 154, 16\}$ שסכומה הוא k, מה שמשאיר את $S'' = \{271,3\}$ בצד. הבניה שנעשה היא כזאת – נגדיר את $x = \sum S - 2k$ מה שייתן לנו את ההפרש (אנחנו לוקחים את סכום הקבוצה במלואה, מחסירים ממנו k שזה המספר אליו אנחנו

שואפים, ועכשיו מוצאים את ההפרש בין הנותרים ל-k – לכן מחסירים 2k), ואת המספר הנוסף, אנחנו מוסיפים לסדרה החדשה.

עכשיו הסדרה הקיימת לנו מכילה 3 אלמנטים עיקריים – 1. תת הסדרה שערכה k.
2. הנותר מהסדרה המקורית.
3. איבר שישלים את הנותר להיות שווה k.

עכשיו נותר לנו להוכיח את תנאי הרדוקציה $w \in SUS \leftrightarrow f(w) \in PAR$.
כלומר, עלינו להוכיח באופן דו כיווני כי כל מילה מSUS שעוברת רדוקציה שייכת ל-PAR.

כיוון 1 - $w \in SUS \rightarrow f(w) \in PAR$

נתון $w \in SUS$. ‏w: {S,k}.

מאחר הוא שייך לSUS, אז יש שם איזה תת סדרה ששווה ל-k, ותת סדרה שהגרנו בתור הנותרים. נוסיף לסדרה את x, כמו שהגדרנו למעלה, ובעת גם הנותרים שווים ל-k. כלומר יש לנו שני תתי-סדרות ששווים בסכומם, ומקיימים את כל שאר התנאים שביקשנו.

כיוון 2 – $w \in PAR \rightarrow f^{-1}(w) \in SUS$.

נתון $w \in PAR$.

כלומר מאחר שביצענו את הרדוקציה, יש לנו בעצם מילה שהיא במקור שייכת ל-SUS, כך שסכום כל האיברים ב-f(w) שווה ל-x+$\sum S$.
פה זה מתחיל להיות קצת מבולגן, אז אני אפריד לשורות שיהיה קצת יותר ברור-
אם נפרק את x, נקבל כי $\sum f(w) = \sum s + \sum s - 2k$.
כלומר $\sum f(w) = 2\sum s - 2k$,
את זה ניתן לחלק לשני חלקים שווים בגודלם, שכל אחד מהם הוא $\sum s - k$.
אם ניקח את אחד החלקים, ונפצל אותו שוב, נקבל - $\sum s - 2k + k$.
נפריד את זה לשני איברים שהגדרנו בעבר, ונקבל x, k. כלומר – הקבוע אליו אנחנו צריכים להגיע, בתוספת ההפרש לנותר. וככה חזרנו למילה המקורית שהתקבלה ל-SUS.

מש"ל.

בעיית תרמיל הגב בשלמים 0-1 (Knapsack)

נתונה קבוצת פריטים $S = \{a_1..a_n\}$. כמו כן, נתונה לנו פונקציית משקל w{i} ומחיר v(i). כמו כן, נתון לנו שק בתבולה מקסימלית של W.
צריך למצוא – תת קבוצה של פריטים $S' \subseteq S$, שסכום מחירי הפריטים בו הוא מקסימלי, ומשקל הפריטים לא עולה על W.

אמרנו כבר קודם שאת האלגוריתם הזה, אנחנו פותרים בתכנון דינאמי, כלומר שהוא שייך ל-P. אז אם אנחנו מוכיחים שהוא בעיית NP-שלמה, אנחנו יכולים להוכיח כי P=NP ולזכות בתהילת עולם!!!

אז לא.

כל מה שדיברנו עד עכשיו היה פסאודו-אלגוריתם ופסאודו-ריצה, אבל אם אנחנו מדברים על קלט של מחרוזת בינארית, זמן הריצה קופץ בצורה מטורפת לרמה המעריכית. ולכן אפשר לשים לב, שהבעיה

מוגדרת כ-0-1. כלומר, בהתייחסות לגישה לבעייה ברמה הבינארית שלה (למעשה חלק גדול מהבעיות שלמדנו בצורה "פשוטה" עלולות ליפול ל-NPC ברגע שנעביר אותן לבינארי).

נשים לב, שהבעיה המוצעת היא בעיית אופטימיזציה, אנחנו מחפשים את מקסימום האיברים שיוכלו להיות בתיק. אבל הבעיות שאנחנו מחפשים באמת, הן בעיות הכרעה. עבור בעיות אופטימיזציה נתונה, אנחנו יכולים להוציא מספר בעיות הכרעה שונות:

בהינתן קבוצת פריטים $S = \{a_1..a_n\}$, פונקציית משקלים w(i), פונקציית עלויות v(i), משקל שק W, ומספר קבוע k -

- האם קיימת תת-קבוצה של פריטים S' ⊆ S, כך שסך המחירים הוא **לפחות** k, תחת האילוץ שסכום המשקלים הוא לכל היותר W.
- האם קיימת תת-קבוצה של פריטים S' ⊆ S, כך שסך המחירים הוא **שווה** k, תחת האילוץ שסכום המשקלים הוא לכל היותר W.
- האם קיימת תת-קבוצה של פריטים S' ⊆ S, כך שסך המחירים הוא **לא-יותר מ**-k, תחת האילוץ שסכום המשקלים הוא לכל היותר W.

בעצם, יש לנו 3 בעיות שונות (ואם ממש רוצים, אפשר למצוא עוד), שכולם מתבססים על בעיות אופטימיזציה, ואנחנו לוקחים אחת מהבעיות ומנסים לעבוד עליה. אנחנו נתמקד בבעיה הראשונה – למעשה היא זאת המובילה לבעיית אופטימיזציית המקסימום – נעלה את ה-k בכל פעם עוד קצת, עד שנמצא את המקסימלי.

עכשיו נכתוב את הבעיה בצורה פורמלית-

$$KNAPSACK0\text{-}1 = \{S=\{a_1..a_n\}, w(i), v(i), W,K \mid \exists\, S' \subseteq S, \sum_{a_i\in S} v(i) \ge k, \sum_{a_i\in S} w(i) \le W\}$$

אם ישאלו אותנו, מה השפה המתאימה, עלינו רק להוסיף את הגדרת הבעייה ב"מילה" כלומר לתחום אותו כך –

$$KNAPSACK0\text{-}1 = \{<S=\{a_1..a_n\}, w(i), v(i), W,K > \mid \exists\, S' \subseteq S, \sum_{a_i\in S} v(i) \ge k, \sum_{a_i\in S} w(i) \le W\}$$

אם ממש רוצים אפשר גם לשייך את זה תחת שפה בינארית $\{0,1\}^*$. עכשיו אחרי שהגדרנו את השפה סאופן פורמלי, אנחנו יכולים למעשה למחוק את כל חלק השאלה למעלה, ולהגדיר את השאלה באופן הבא: הוכח שהשפה KNAPSACK0-1 ∈ NPC

נוהל ההוכחה מוכר – 1. נוכיח כי KNAPSACK0-1 ∈ NP

<u>הוכחה –</u> נוכיח כי בהינתן מחרוזת קלט כמו שאמרנו $x \in \{0,1\}^*$, ומחרוזת אימות y, שתהיה פשוט תת קבוצה של פריטים, נוכל להוכיח את הנכונות תחת זמן פולינומי.

ראשית, יש לשים לב, שברור שאורך קלט האימות הוא ביחס פולינומי ל-S, מאחר שמדובר בתת סדרה. עכשיו נבנה את האלגוריתם של האימות, ונכתוב בצד כל שורה גם את זמן הריצה הנכון.

1. בדיקת תקינות הקלט (קידוד מתאים ונכון) – O(n)
2. בדיקה שאכן S' ⊆ S – $O(n^2)$
3. חישוב $\sum_{a_i\in S} w(i)$ - O(n)
4. אם W < w' נחזיר false – O(1)
5. חישוב $\sum_{a_i\in S} v(i)$ -O(n)

6. אם $v' < k$ נחזיר false – O(1)
7. נחזיר true – O(1)

אם תת הקבוצה 'S אכן היא תת הקבוצה הנכונה, כל התנאים יתקיימו ואכן נגיע עד הסוף ונחזיר true. מבחינת זמן ריצה – הזמן המקסימלי יהיה $O(n^2)$. חשוב לציין שוב, מאחר שמדובר באלגוריתם אימות, תשובה חיובית תגיד לנו בוודאי שהבעייה שיכת ל-NP, אבל אם תחזור לנו תשובה שלילית, זה לא אומר שהבעיה לא ב-NP אלא שאנחנו לא יכולים לקבוע אם כן או לא, יכול להיות שסתם קיבלנו קלט אימות לא נכון, ואין לנו שום דבר שנוכל לעשות, מלבד לחכות לקלט נכון.

עכשיו נוכיח KNAPSACK0-1 ∈ NPH

<u>הוכחה:</u> נראה כי קיימת רדוקציה $\text{SUSBSET-SET} \leq_p \text{KNAPSACK0-1}$.

ברור לנו מאוד למה לקחנו את השפה הזאת, ואכן בהרבה מאוד מקרים, השפה שנבחר לעשות ממנה רדוקציה, תהיה דומה יחסית לשפה שאנחנו מחפשים. ניקח לדוגמה את הבעיה הנוכחית, יש לנו k מסויים ואנחנו צריכים לסכום את האיברים על מנת להגיע לאותו k. במקרים כאלה עיקר הרדוקציה, תהיה להראות איך אנחנו עושים את המעבר בין שיטה אחת לאחרת.

צריך לזכור! הרדוקציה היא חד כיוונית. כלומר, כל המעבר שלנו צריך להוכיח, שעבור כל מופע בבעיה המקורית שאנחנו יוצאים ממנה, אנחנו יכולים להגיע לפונקציה איפשהו בבעיית היעד. אנחנו לא צריכים לכסות את כל המופעים של תיק הגב (במקרה כמו שלנו), אלא להראות שכל בעית סכום תת קבוצה, ניתן להעביר לתצורה של תיק גב.

אופן הרדוקציה – עבור כל סדרה שנקבל ב-SUS, נגדיר את הבעיה בצורת תיק הגב באופן הבא – כל פריט a_i, יוכפל להיות גם w(i), וגם v(i) בעלי אותו ערך (אין סיבה שמופע כזה לא יהיה אפשרי). את ה-k המוצע לנו ב-SUS, נכפיל גם כן – פעם אחת בתור k, ופעם אחת בתור W. ועכשיו נוכל לפתור את KS0-1 תחת התנאים המתאימים.

נחזור שוב על מה שעשינו – בעוד שבבעיית SUS יש לנו רק סדרה אחת של מספרים, בבעיית KS0-1, יש לנו שתי סדרות עם מידע רלוונטי, סדרה של משקל וסדרה של מחיר. באופן דומה, כנגד המספר הקבוע (הבודד) שנמצא ב-SUS, קיימים לנו שני מספרים שתוחמים את הסדרות לפי העלות והמשקל. ההגדרה המחודשת שלנו בעצם גורמת לנו לעבור פעמיים על אותה סדרה, כאשר בכל פעם אנחנו צריכים להיות שווים או גדולים/קטנים (לפי הסדרה) מאותו מספר. אם נניח שב-SUS רצינו להיות שווים בדיוק ל-k, אז כאן אנחנו רוצים להיות גם גדולים/שווים וגם קטנים/שוים מ-k, מה שנותן לנו בעצם את האופציה היחידנית להיות שוים ל-k. ובזה כולם מרוצים.

<u>הוכחת נכונות</u>: צ"ל כי $x \in \text{SUBSET-SUM} \leftrightarrow f(x) \in \text{KNAPSACK0-1}$

כיוון ראשון – נתון כי x ∈ SUS. צריך להראות שהרדוקציה על המילה מביא אותנו לבעייה מתאימה של תיק הגב, ומאחר שקיימת תת קבוצה $S' \subseteq S$, כך שסכומה שווה ל-k. אז סכום המשקלים/העלויות שלה שווה גם הוא בשקלול ל-k.

כיוון שני – נתון f(x) ∈ KS-1. צ"ל שהמילה שיצאנו ממנה שייכת ל-SUS.

נתון לנו שקיימת קבוצת פריטים שסכום משקלם לא יותר מ-W, וסכום המחירים לא פחות מ-k. כנגד כל קבוצת פריטים כזאת, קיימת סדרת מספרים השווה בדיוק ל-k.

הרדוקציה בוודאי עובדת בזמן פולינומי – מדובר על השמה מחדש של הסדרה – O(n).

ומשילוב שתי ההוכחות ראינו כי KNAPSACK0-1 ∈ NPC.

בעיית אריזות המיכלים (Bin Packing)

נתונה קבוצת פריטים $A = \{a_1.. a_n\}$, כך שלכל פריט a_i, יש גודל s_i כך ש $0 \le s_i \le 1$. נתונה קבוצה אינסופי של מיכלים בגודל 1 כל אחד.

עלינו לסדר את הפריטים במיכלים כך שנשתמש במינימום מיכלים. נדגיש - עלינו לקחת את <u>כל</u> הפריטים, אבל בכמה שפחות מיכלים.

<u>בעיית ההכרעה</u> – האם ניתן לסדר את כל הפריטים לכל היותר ב-k מיכלים?

צ"ל BIN-PACKING ∈ NPC

יש לנו המון פריטים שהמשקל של כל אחד מהם הוא בין 0-1, כלומר שבר כלשהו. אנחנו רוצים למלא מיכלים בתבולה של 1, בצורה אופטימלית, תחת הגבלה של מספר מיכלים שנקבע. כמובן, שיכול להיות שנבקש שייכנס ל-100 מיכלים, ונסגור את כל הסיפור ב-3 מיכלים בלבד, אבל מן הסתם, זה לא תחום המופעים לבעיות שיגיעו אלינו ושאיתם נצטרך להתמודד.

בשלב הזה של הרדוקציות, אנחנו יכולים כבר להתחיל לזהות דפוסים של הרדוקציה, ושלבי העבודה יהיו קלים יותר.

ראשית, נוכיח כי BIN-PACKING ∈ NP

<u>הוכחה</u> – בהינתן $x \in \{0,1\}^*$, אישור y יהיה סידור של כל הפריטים ברצף שיהיה תואם למילוי k מיכלים. האישור כמובן יראה לנו שהאיברים מתחלקים למיכלים בצורה נכונה.

אלגוריתם האימות:

1. בדיקת תקינות הקלט – $O(n)$
2. בדיקה שכל הפריטים ב-y נמצאים ב-A – $O(n^2)$
3. נבדוק שהסידור חוקי:
   a. סך הגדלים בקבוצה לא יותר מ-1 – $O(n)$
   b. השתמשנו ב-k מיכלים לכל היותר. – $O(n)$

זמן הריצה הוא כמובן $O(n^2)$.

נוכיח כעת כי PARTITION $\le_p$ BIN-PACKING

למה לקחנו דווקא את החלוקה לקבוצות? אנחנו מחפשים בעיות שיהיו להם מוטיבים דומים – במקרה שלנו אנחנו מחפשים להגיע לחלקים שווים, אם מבין סדרה של מספרים (PAR), ואם מדובר בסדרה של פריטים (BP). מה שנותר לנו הוא לגשר על הפער, ולמצוא כיצד אנחנו הופכים את PARTITION להיות בעיה של BIN_PACKING. מה שנעשה הוא כזה – בעיית הPARTITION מבקשת לחלק את הסדרה שני חלקים, לכן אנחנו נגדיר את ה-k של ה-BP להיות 2. כלומר אנחנו רוצים להכניס את כל הפריטים לשני מיכלים שווים בגודל 1.

עכשיו, אנחנו יוצאים למעשה מהשפה PARTITION, כלומר עבור כל מילה שנקבל, אם היא תהיה שייכת לשפה, זה אומר שנוכל לקבל איזה סדרה שסכומה הכולל יכול להתחלק לשניים באופן מסודר. כך שאם

נרצה להתאים את הבעיה למיכלים בגודל 1, נצטרך לחלק כל מספר בחצי מהסכום הכולל (כלומר, אם אנחנו סוכמים את האיברים {1, 2, 3, 4}, נחלק כל איבר ב-5), ובך סכום איברי תתי-הקבוצות יהיה שווה למיכלים בגודל אחד.
כמובן שאם המילה שנקבל לא באמת שייכת לשפה של PAR, שום חלוקה לא תעזור לנו, וגם לא נוכל להכניס את זה בתוך מיכלים מתאימים (לדוגמא – אם יש לנו {2, 12} ונחלק כל אחד מהם ב-7, לא באמת נוכל להכניס את זה לתוך שני מיכלים).

<u>הוכחת נכונות:</u> עלינו להוכיח כי $x \in PARTITION \leftrightarrow f(x) \in BIN\text{-}PACKING$

כיוון ראשון – נתון כי $x \in PARTITION$, כלומר יש לנו שתי תתי-סדרות השוות אחת לשניה. כלומר לאחר הרדוקציה יהיה לנו שתי סדרות של פריטים שניתן להכניס לשני מיכלים ולקיים את כל הדרוש לבעיית BP.

כיוון שני – נתונה לנו מילה $f(x) \in BIN\text{-}PACKING$, כלומר שהיא שייכת לתחום הבעיות בו k=2, וכל הפריטים נכנסים למיכלים. אם נפרק את הרדוקציה בחזרה, נקבל סדרה של מספרים שונים, שניתן לחלק אותם בהתאמה לשתי תתי סדרות שיהיו שווים בסכומם אחד לשני.

מש"ל. ומכאן ניתן לומר כי $BIN\text{-}PACKING \in NPC$.

אלגוריתמי קירוב

ראינו לא מעט בעיות שהן NP שלמות, ודי ברור לנו שהאלגוריתמים המנסים לפתור אותם אינם יעילים בכלל. אבל לצערנו, לעומת הפתרונות שהם לא ריאליים כל כך, הבעיות עצמן דווקא כן, ונוגעות בלא מעט תחומים בחיי היום-יום. הבעיה כמובן מובנת – אם יש לנו בעיות שהם לא ממש פתירות, אבל אנחנו חייבים לעשות משהו, מה נעשה? הפרק הזה נועד לתת מענה כלשהו לשאלה זאת. כמובן שהדרך הנוחה ביותר, היא פשוט לוותר ולעזוב את הבעיות האלו לנפשן. אבל אנחנו קצת יותר חזקים מזה, ולכן אנחנו ננסה למצוא אלגוריתמי קירוב.

מה הרעיון העומד מאחורי זה? דיברנו במשך כל הקורס, על כך שאנחנו משתדלים להמנע בכל האפשר מאלגוריתמיים שרצים בזמן מעריכי, ואכן אם יהיה לנו זמן ריצה של n^2 זה לא ממש להיט, אבל אם נתייחס לעולם האמיתי של הבעיות שעומדות לפנינו בדרך כלל, ה-n שאנחנו מדברים עליו לא באמת שואף לאינסוף. נכון, הוא יכול להיות מאוד גדול ומבאס, אבל גם אם יש לנו n=1000, אז זמן ריצה של מיליון, הוא אמנם לא כיף בעליל, אבל אפשר להגיע אליו מתישהו. באופן דומה, אם אנחנו מנסים להעביר שדר בוידאו, אחד מכל מיליוני הפיקסלים נאבד, או אפילו אלף כאלה, אנחנו נצליח לראות את הוידאו הזה וזה לא באמת יפריע לנו, כי אנחנו מסוגלים להתמודד עם חוסר שלמות.

עכשיו, ברגע שהוכחנו שאלגוריתם מסוים שייך ל-NPC, אנחנו נצא מנקודת הנחה שאין לנו פתרון פולינומי (כלומר $P \neq NP$), וחבל שנבזבז את הזמן שלנו בלנסות למצוא אחד כזה. מה שאנחנו יכולים להשקיע בו את הזמן וזה דווקא כן יהיה פרודוקטיבי, הוא למצוא אלגוריתם קירוב מספיק טוב.

האלגוריתמים השונים יתחלקו לשניים – אלגוריתמי אופטימיזציה והיוריסטיקה.

<u>אופטימיזציה</u> – מציאת פיתרון בצורה קירובית. כלומר, ניסיון למצוא אלגוריתם שיביא לנו פתרון "סביר" לבעיה.

<u>היוריסטיקה</u> – שיטה לחפש פתרון בצורה מקורבת.

אמנם זה נראה שאין הרבה הבדל בין שני אלה, אבל היוריסטיקה מתבססת יותר על ניסוי וטעיה, ומתרכזת בחקירה על מנת למצוא איזשהו פתרון שהוא יהיה סביר. האופטימיזציה לא מחפשת, אלא ישר מביאים איזה אלגוריתם שמביא פיתרון מקורב כלשהו, ועליו אנחנו רצים.

חסמים על ביצועי אלגוריתמי קירוב

אם אכן נמצא פתרון שהוא לא אופטימאלי, אבל עובד בצורה מספיק טובה כדי לפתור את הבעיה, אנחנו נצטרך גם להגדיר עד כמה התפשרנו באלגוריתם שהבאנו. אמנם זה נראה קצת מוזר שאנחנו אומרים "אין לנו מושג מה הפיתרון, אבל קח פתרון שהוא גרוע פי 2 ממנו!", כי אם אנחנו לא יודעים מה הפתרון, איך אנחנו בכלל יכולים להגיד על משהו שהוא קרוב אליו? אבל בעצם, אם נסתכל על זה, אנחנו מדברים על אימות של אלגוריתם תחת זמן פולינומי, כלומר אנחנו יכולים לקבל איזה מילה שתהווה פיתרון. אין לנו שמץ של מושג מה המילה, אבל בחלק גדול מהבעיות, אנחנו יכולים להגיד מה סדר הגודל של האימות הזה, ולפי זה אנחנו מחשבים.

את הפתרון האמיתי והאופטימלי, אנחנו מגדירים בתור C^*. את הפתרון המקורב, אנחנו מגדירים C. אם עד עכשיו בשאלות התמקדנו בבעיות הכרעה שהתעסקו עם איזה k מסוים, כאן אנחנו מדברים על בעיות

אופטימיזציה של "הגדול ביותר" או "הקטן ביותר". כלומר, אנחנו לא שואפים להגיע הכי קרוב למספר, אלא הכי קרוב לפתרון המוצלח ביותר.

אנחנו נגדיר שני סימונים עבור החסמים האלו, שכל אחד מהם יסמן דבר טיפה שונה –

ρ(n) – (קוראים לזה "רו", למרות שזה נראה כמו "פי"). מבטא את **חסם היחס**, כלומר פי כמה הפתרון שהבאנו רע יותר מהפתרון המקורי. הוא מוגדר באופן הבא $\max\left(\frac{C}{C*},\frac{C*}{C}\right)\leq \rho(n)$. כאשר ההגדרה מתאימה גם לבעיות מינימום וגם לבעיות מקסימום.
איך זה עובד? מאחר ואנחנו מחפשים את המקסימום בין שני השברים האלה, אז התוצאה שתתקבל היא תמיד תהיה ה"גרועה" יותר – לדוגמא: עבור בעיית מקסימום – נגיד והפתרון האמיתי הוא C*=10, ואנחנו הגענו ל-8, אם נכניס את הערכים לפונקציה, נקבל $\max\left(\frac{10}{8},\frac{8}{10}\right)$, וכמובן שהתוצאה המקסימלית היא 1.8.
אבל עבור בעיית מינימום, אם התנאים יהיו בדיוק הפוכים, עדיין נקבל חסם יחס של 1.8 (מוזמנים לבדוק אותי, זה עובד!). כמובן, שחסם יחס לא יכול להיות בשום אופן קטן מ-1, כי אז בעצם יתברר לנו שמצאנו פתרון טוב יותר מהאופטימלי, וזה לא יכול להיות כי (כמו שאנחנו זוכרים מהפתרונות החמדניים) אם היינו מוצאים פתרון טוב יותר מהאופטימלי, אז כבר הוא היה אופטימלי בעצמו.

ε(n) – **חסם השגיאה היחסית**. יכול להביא לנו סדר גודל של הטעות עצמה. כלומר, לא עד כמה אנחנו קרובים יחסית לפתרון האופטימלי, אלא כמה אנחנו רחוקים ממנו, ועד כמה פספסנו אותו (מה ההפרש). חישוב השגיאה היחסית מחושב ב- $\frac{|C*-C|}{C*}\leq \varepsilon(n)$.

עבור חלק מהבעיות, הצליחו למצוא איזה פתרון שרץ בצורה מקורבת בלי קשר לגודל הקלט. במקרה כזה, סימוני יחסים יהיו ללא ה-n. במקרים אחרים יכול להיות, שעבור קלט מסוים היחס ישתנה ולכם צריך לציין את גודל הקלט. את כל זה נראה בהמשך.

נעבור עכשיו על מספר אלגוריתמי קירוב, ונראה איך הם פותרים בעיות שהסתבכנו איתם, ומה חסם היחס של הטעות.

אלגוריתם קירוב Vertex-Cover

את בעיית כיסוי הקדקודים הכרנו כבר קודם, וכעת נציג אלגוריתם קירוב שפותר אותו, אבל קודם נזכיר שוב את הבעיה-

נתון לנו גרף בלתי-מכוון G(V,E). כיסוי על ידי קדקודים יהיה קבוצה V' ⊆ V, כך שאם (u,v), היא קשת ב-G, אזי u ∈ V' או v ∈ V' (או שניהם). גודלו של כיסוי על ידי קדקודים הוא מספר הקדקודים בכיסוי.

בעיית האופטימיזציה המתאימה, היא מציאת מינימום הקדקודים שיכסו את כל הצלעות הקיימות בגרף.

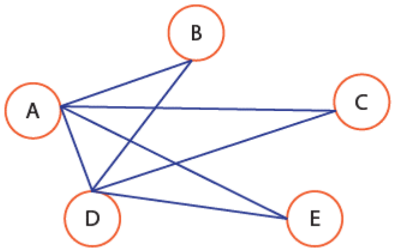
האלגוריתם הבא שנציע, פותר את הבעיה ומציע לנו כיסוי קדקודים טוב, אבל בדרך כלל הוא לא מוצא את הכיסוי האופטימאלי.

```
Approax-Vertex-Cover (G)
  C ← ϕ                                          // מתחילים עם קבוצה ריקה, שלתוכה נכניס את הפתרון
  E’ ← E[G]                                      // קבוצה זאת תהווה את קבוצת הצלעות עליה עובדים
  while E’ ≠ ϕ                                   // הלולאה רצה עד שנכסה את כל הצלעות
        let (u,v) be an arbitrary edge of E’     // בוחרים שני קדקודים להם יש קשת בקבוצה
        C ← C ∪ {u, v}                           // מוסיפים את הקדקודים לקבוצת הכיסוי
```

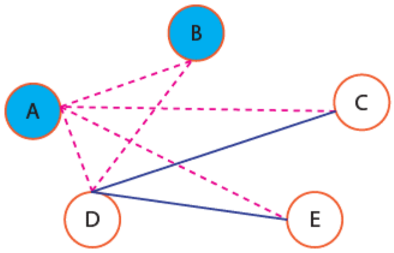
```
        remove from E' every edge          // מורידים את כל הקשתות המחוברות לשני הקדקודים
        incident on either u or v
return C
```

בפשטות – אנחנו בוחרים בכל פעם שני קדקודים (וכמובן גם את הקשת ביניהם), ומורידים את כל הקשתות המקושרות לאחד משני הקדקודים. באופן הזה, אנחנו מנסים לכסות כמה שיותר קדקודים בכמה שפחות איטרציות על הלולאה.

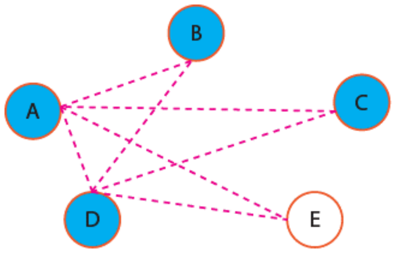
נראה דוגמת ריצה פשוטה לאלגוריתם:



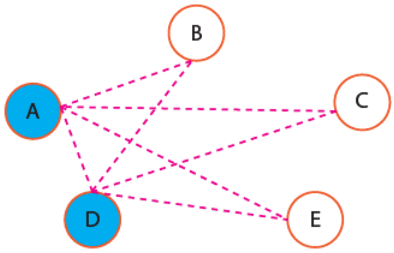
ניקח את הגרף הפשוט הזה. על פי האלגוריתם נבחר שני קדקודים, ובאופן טבעי ניקח את {a,b} כי אנחנו אוהבים ללכת בסדר לקסיקוגרפי, ונסמן את הקשתות השייכות אליהם –



לא כיסינו הכל. עכשיו נעבור הלאה ל- {c, d}. נבדוק את הקשתות המתאימות –



הידד לנו! כיסינו את כל הקשתות. אבל, מי שיסתכל טוב יותר, יוכל לראות שאת אותו הכיסוי בדיוק היינו יכולים לעשות עם שני קדקודים בדיוק –



עכשיו אנחנו יכולים להבין איך אנחנו יכולים להתחיל להעריך את החסמים של השגיאה/קירוב שלנו. אלגוריתם אופטימלי באמת לא היה בוחר באופן אוטומטי שני קדקודים סמוכים אחד לשני. למה? כי אין

שום היגיון בזה. למרות שבתוצאה הסופית זה אבן מה שקרה, יש היגיון יותר נכון בלחפש קדקודים אחרים שלא מחוברים, אחרת אנחנו "מבזבזים" פה חלק מהכיסוי. כמובן שצריך לזכור שגרפים יהיו בדרך כלל קלט גדול יותר מחמישה קדקודים, אבל כבר מפה אפשר לנתח שאם הפתרון האופטימלי הוא $C^*=2$, אז פתרון מקורב יכול להיות $C=4$. כלומר יש לנו איזה חסם של קירוב פי 2 מהתוצאה האופטימלית. כמובן, שיכול להיות שתוצאה תהיה ביחס טיפה שונה על קלטים שונים. למשל, מאחר ופה יצא לנו שני קדקודים סמוכים, אם רק היינו עובדים בסדר אחר, היינו יכולים במכה אחת להצליח ולכסות את כל הקשתות. ולכן אנחנו צריכים לזכור שהחסם של ρ הוא רק חסם עליון לבעיה ולא מדד קבוע שאומר שבכל פעם אנחנו הולכים לטעות בדיוק פי 2.

הוכחת זמן ריצה –

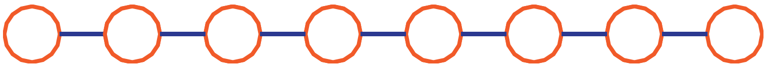
אתחול הגרף נעשה ב$O(n)$. הלולאה כמובן יכולה לרוץ עד $O(n^2)$. תלוי כמובן עד כמה הקשתות מחוברות בין הקדקודים.

הוכחת נכונות –

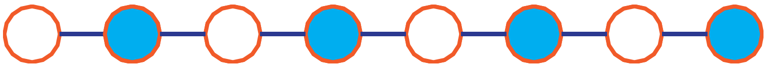
ראשית, נוכיח שהאלגוריתם לכשעצמו נכון. מאחר והלולאה רצה עד שמאגר הקשתות שלנו ריק, אז בוודאי שאנחנו מכסים את כל הקדקודים, ומקיימים את הדרישה הבסיסית של הבעיה.

הוכחת חסם –

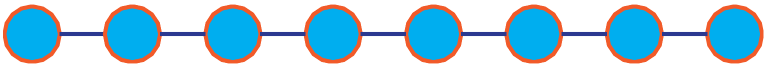
במקרה הגרוע ביותר (שלמעשה תיארנו למעלה), בו אין חפיפה בין הקשתות, אנו נכסה כל קשת על ידי שני קדקודים, לעומת האלגוריתם האופטימלי שיכסה רק קדקוד אחד. ובכל איטרציה שכזו, אנחנו מוסיפים עוד שניים. מה שיוצא בסוף, הוא שאם היה לנו גרף כאמור, אנחנו נהיה מוכרחים לקחת פי 2 מהאופטימלי.



שוב, לצורך הדוגמה, הגרף הזה שאין בו חפיפה, היה אמור להיות בצורה אופטימלית משהו כזה-



ועל ידי ארבעה קדקודים לכסות הכל. אבל מאחר ואנחנו בוחרים זוגות, מה שיצא לנו הוא דווקא כזה –



וזה כמובן פי 2 מהאופטימלי, ולא מוצלח בכלל.

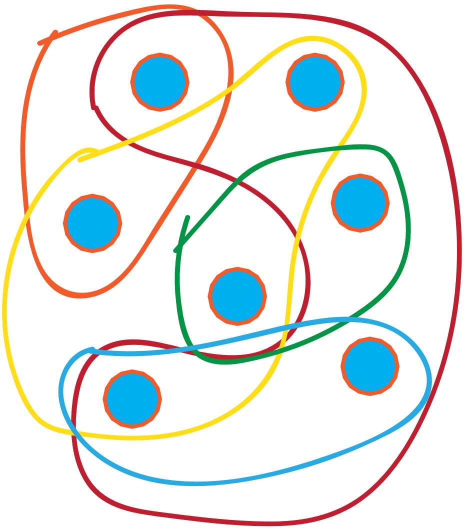
בעיית כיסוי הקבוצה

בעיית כיסוי הקבוצה, הינה בעייה שעדיין לא נתקלנו בה, אבל היא לא מסובכת מידי (כאילו, היא כן קשה, אבל לא קשה להבין אותה). היא מוגדרת בצורה הבאה – נתונה לנו קבוצה X, ומשפחה F שבעבורה יש לנו מספר תתי קבוצות $S \in F$, המקיימים $x = \bigcup_{S\in F} S$, כלומר איחוד כל תתי הקבוצות יביא לנו את הקבוצה הגדולה X. עלינו למצוא C (פתרון) מסוים שיהווה כיסוי של קבוצות מינימלי ל-X. ובצורה של בעיית אופטימיזציה – מה מספר הקבוצות המינימלי שנדרש עלינו לקחת בשביל לקבל את כל X?

נראה דוגמה פשוטה של הבעיה – $S_1 = \{1, 2\}, S_2 = \{2, 3\}, S_3 = \{3,4\}$ $X = \{1, 2, 3, 4\}$. יש לנו 3 תתי קבוצות שאיחוד שלהם יביא לנו את הקבוצה X, אבל קל לראות שאנחנו לא צריכים באמת את שלושת תתי הקבוצה, אלא רק את S_1, S_3 והם לבדם יכסו לנו הכל.

הבעיה הזאת שייכת ל-NPC, על ידי רדוקציה מבעיית כיסוי הקדקודים, שלא מפורטת[14]. מי שזה ממש מעניין אותו, אז הבעיה הזו היא אחת מ-21 הבעיות שהגדיר ריצ'ארד קארפ (ההוא מאדמנודס-קארפ).

בצורה ויזואלית ניתן לראות את הבעיה כמספר איברים התחומים בקבוצות, ואנחנו מחפשים כמה שפחות איחודי קבוצות, באופן הבא:

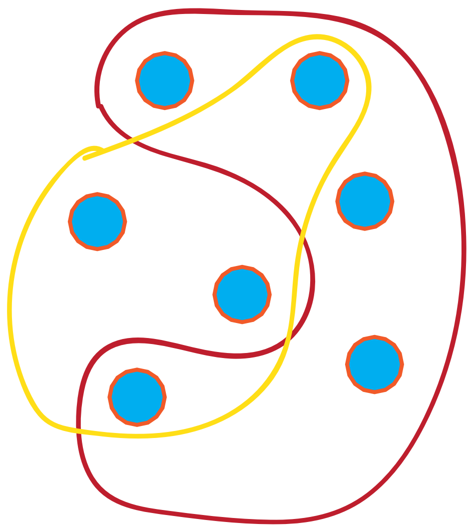


נתונים לנו הקבוצות השונות, כאשר חלקן מוכלות אחת בשניה, וחלקן מוכלות באיחוד שתי קבוצות וכדו'. והמשימה שלנו היא למצוא את האופטימום של מינימום איחוד קבוצות שיכסה את כל האיברים.
האופטימום ייראה כמובן כך:

---

[14] הצעה שלי לרדוקציה (שלא וידאתי לגמרי את הנכונות שלה, אבל אני אנסה בכל זאת):
סדר הרדוקציה-
1. כל קשת $(u, v) \in E$ תוגדר כאיבר שיש לכסות, ונשנה את השם ל $\{e_1...e_n\}$.
2. כל קדקוד $v \in V$ יוגדר כמסגרת קבוצה מתוך $\{S_1... S_n\}$.
3. נחזיר את הקבוצות הנתונות תחת V' (כיסוי הקדקודים) בכיסוי הקבוצות האופטימלי F.

לאחר הרדוקציה, הקבוצות השונות יהיו נתונות כל אחת במסגרת על פי קישור הקדקודים שהיה בגרף המקורי עבורו אנחנו יודעים את הכיסוי. הטענה היא: הקבוצות המושרות על ידי כיסוי הקדקודים יהווה גם כן את כיסוי הקבוצות.
הוכחת נכונות:
כיוון ראשון: מאחר ונתונה לנו קבוצת הקדקודים V', אזי עבור כל קשת (u, v), $u \in V'$ או $v \in V'$ (או שניהם).
כלומר לאחר הרדוקציה, האיבר (u, v) יהיה מוכל תחת אחת הקבוצות השייכות ל-S' (קבוצת כיסוי הקבוצות).
כיוון שני: עבור כל איבר $ei \in X$ המוכל בתוך קבוצה השייכת ל-F, נוכל להחזיר אותו אחורה להיות קשת המחוברת לקדקוד. כך שאם הקבוצה $Si \in F$, אזי גם $vi \in V'$.

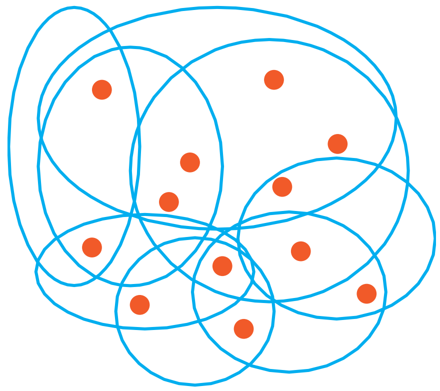


ועכשיו נראה את אלגוריתם הקירוב, הפועל באופן חמדני שיביא לנו תוצאה סבירה, באופן הבא:

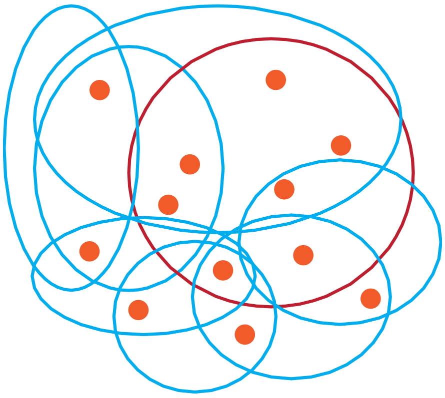
```
Greedy-Set-Cover (G, F)
  U ← X                                          // הקבוצה הזו מכילה את כל האיברים שטרם כוסו
  C ← ϕ                                          //מכיל את הכיסוי הנבנה לפי הקבוצות
  while U ≠ ϕ                                    // הריצה היא בלולאה כל עוד לא כיסינו את כל האיברים
        select an S ∈ F that maximizes | S ∩ U |  // בוחרים את הקבוצה שתוסיף כמה שיותר איברים לכיסוי
        U ← U- S                                 // מורידים את האיברים שכיסינו מקבוצת אלו שלא טופלו
        C ← C ∪ {S}                              //  מוסיפים את האיברים לקבוצת המטופלים
  return C                                       // בסיום הלולאה מחזירים את הקבוצה של הכיסוי המינימלי
```

נשים לב, התנאי לא אומר שבכל פעם אנחנו לוקחים את הקבוצה הגדולה ביותר, אלא את הקבוצה שמספר האיברים שהיא מוסיפה הוא המקסימלי. יכול להיות שיש קבוצה ענקית, אבל היא תוסיף לנו איבר בודד, ומצד שני קבוצה שמכילה רק שני איברים – במקרה כזה אכן ניקח את הקבוצה הקטנה יותר, שיעילה לנו בחשבון הכללי.

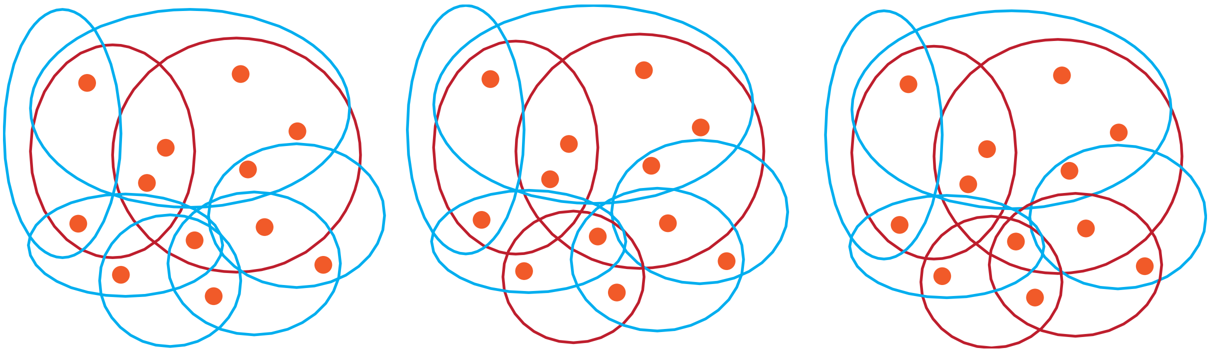
נריץ בזריזות את האלגוריתם על דוגמא, ונראה עד כמה זה מוצלח. או שלא. נתון לנו הבלאגן הבא:



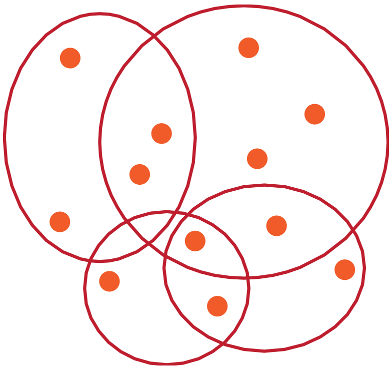
במבט מלמעלה קשה לראות כמה קבוצות יש, ומה שייך למה, אבל אנחנו פשוט נעבוד בצורה החמדנית שהוצעה לנו. מאחר שהקבוצה של ה"מטופלים" ריקה, בשלב הראשון אנחנו פשוט בוחרים את הקבוצה בעלת מספר האיברים הגדול ביותר-



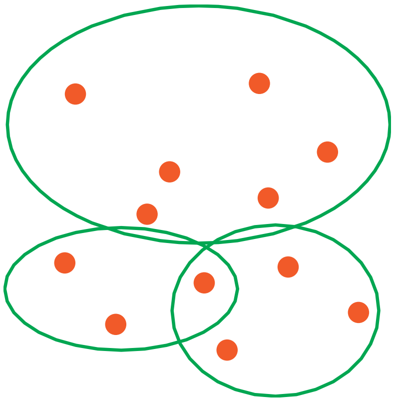
מוזמנים לבדוק אותי. בקבוצה זו יש 7 איברים, וזה המספר המקסימלי. עכשיו נותרו לנו בחוץ חמישה איברים, שמחולקים פחות או יותר בקבוצות של 2 איברים בכל אחת, ולכן אין זה משנה הסדר בו נבחר את הקבוצות השונות –



וכיסינו את כל האיברים, הצלחנו לעשות זאת בארבע קבוצות ואנחנו מבסוטים כי קיבלנו את התוצאה הבאה:



אממה? זה מאוד נחמד ארבע קבוצות, אבל התוצאה האופטימלית היא 3!



למעשה, כבר בבחירה הראשונה שנראתה לנו טובה עשינו את הפאדיחה! אבל צריך לזכור, שמרווח הטעות הזה הוא לגמרי סביר. זה לא שעשינו קבוצה נפרדת לכל איבר, אלא רק הוצאנו קבוצה אחת יותר מידי.

עכשיו ננתח את הריצה של האלגוריתם –

אנחנו מגדירים את חסם היחס כפונקציה הרמונית התלויה במספר האיברים הגדול ביותר של תתי הקבוצות השוונות, כלומר – H (max {|S| : S ∈ F}).

הפונקציה ההרמונית מוגדרת בסכימה הבאה – $H(d) = \sum_{i=1}^{d} \frac{1}{d}$. כלומר, איחוד השברים עד למספר האיברים בתת הקבוצה הגדולה ביותר. בדוגמה שהבאנו חסם היחס מוגדר 2.6 (בקירוב) שזה אומר שלכאורה היינו יכולים לשמונה קבוצות בשביל לכסות את כל האיברים. כמובן שככל שהקבוצה הראשית גדולה יותר, ככה מרווח הטעויות גדול. אין הוכחה נדרשת לזה ואכמ"ל.

בעיית הסוכן הנוסע

שוב בעיה מוכרת כבר. נתון לנו גרף לא מכוון G(V,E), מלא (יש קשת בין כל שני קדקודים), כאשר כל קשת ממושקלת c ≥ 0, ועלינו למצוא מעגל המילטוני בעל עלות מינימלית בגרף. כלומר, עלינו למצוא את הדרך הזולה ביותר שהסוכן יוכל לעבור בכל הערים פעם אחת בלבד, ויסיים בעיר המוצא.

הבעיה כמובן מוגדרת NP שלמה, למעוניינים יש הוכחה לזה שם למעלה. אם כן, ברור לנו שאנחנו לא יכולים למצוא אלגוריתם יעיל שיפתור את הבעיה, אבל מאחר שההשלכה של הבעיה הזאת על המציאות היא מאוד גדולה (תחשבו על שליח שיש לו מספר תחנות עליו לעבור), אנחנו מחפשים אלגוריתם קירוב, שאולי לא יהיה הכי אופטימלי, אבל יהיה יעיל וסביר מבחינת התוצאה. עבור אלגוריתם הקירוב נצטרך להגדיר הגדרה אחת נוספת שתהיה תנאי חשוב באפשרות בכלל להביא פיתרון מקורב-

אי שיוויון המשולש

במושג הזה כבר נתקלנו בעבר, ואנחנו עושים לו השלכה לבעיות אלה. אנחנו יוצאים מנקודת הנחה, שעבור כל מעבר בין שני קדקודים u, v ∈ V הדרך בעלת העלות הנמוכה ביותר, תהיה הדרך הישירה בין שני הקדקודים. כלומר – אם יש לנו משולש (u, v, w) אזי, c(u,v) ≤ c(v, w) + c(u, w). במקרה הטוב ביותר, דרך שעוברת בין הקדקודים בצורה לא ישירה, תהיה אולי שווה לדרך הישירה אך לעולם לא תהיה בעלת עלות נמוכה יותר מהדרך הישירה (u, v).

עבור בעיית הסוכן הנוסע, אנחנו דורשים שהתנאי הזה, שהוא הגיוני בסך הכל ונטוע במציאות, יתקיים.

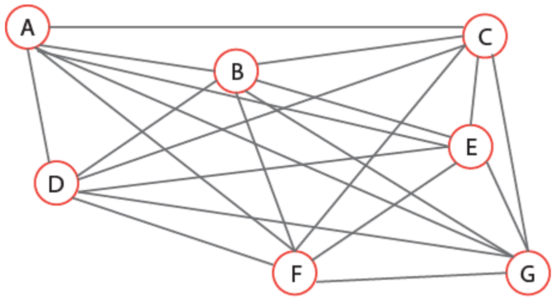
אלגוריתם הקירוב:

Approx-TSP-Tour (G, c)
    select a vertex r ∈ V[G] to be a "root" vertex
    grow a minimum spanning tree T for G from root r usint MST-Prim(G, c, r)
    let L be the list of vertices visited in a preorder tree walk of T
    return the Hamiltonian H that visits the vertices in the order L

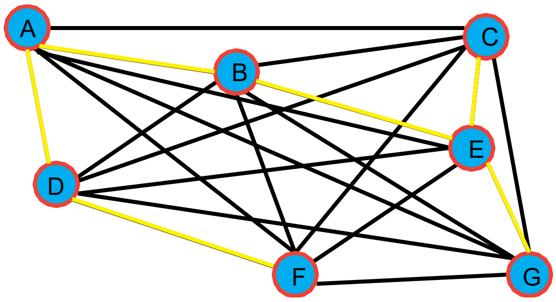
כלומר – קודם כל אנחנו בוחרים את קדקוד המוצא. לאחר שהגדרנו מה יהיה מוצא, אנחנו מתחילים להריץ את אלגוריתם "פרים" על הגרף, בשביל לקבל את העץ הפורש המינימלי. לאחר מכן, אנחנו עוברים בסריקת Pre-Order (אנחנו לא מחויבים דווקא לסריקה תחילית, אלא בעיקר חשוב שנשמור על סדר קבוע) על כל הקדקודים שהתקבלו, החל מקדקוד המוצא, ואז אנחנו נחזיר את סדר ביקור הקדקודים. כמובן, שבכל קדקוד אנחנו נבקר עד פעמיים, ולכן אנחנו מתחילים לצמצם בצורה שלא רשומה באלגוריתם.

עבור כל מעבר כפול בין קדקודים, אנחנו נבדוק מה הקדקוד הבא אליו אנחנו מגיעים בפעם הראשונה, וניצור קיצור דרך ישירות אליו, דבר שאנחנו מסוגלים מאחר והגרף מלא. למשל, אם סדר הסריקה יביא לנו את הרצף הבא – v, w, u, w (שזה כמובן אבא ושני בנים) במקום לחזור מ-u ל-w (שאנחנו יכולים לחשוב בו שהוא בעייתי כי הוא בצבע אדום), אנחנו נבחר את הקשת (u, v) ונכניס אותה לגרף. מאחר ואנחנו מתבססים על אי-שיוון המשולש, יצירת הקיצור הזה אמורה לבחור לנו קשת שהיא או שווה למסלול הקודם, או (בתקווה) אפילו טובה יותר ממנו.

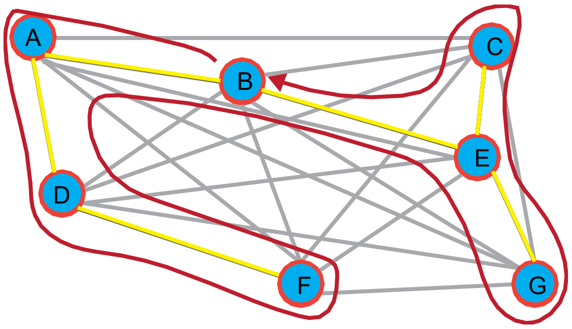
נריץ דוגמא, ונראה מה יוצא לנו-



ניקח את הגרף הבא, שהוא בדיוק הגרף עליו הרצנו את פרים בפרק למעלה, רק שהוספתי לו מספיק קשתות על מנת שיהיה מלא (והורדתי את המשקלים, כי זה היה עושה בלאגן). ונצא מנקודת הנחה שמתקיים פה אי-שיוויון המשולש. נריץ עליו את פרים, ונקבל את תת העץ הבא-

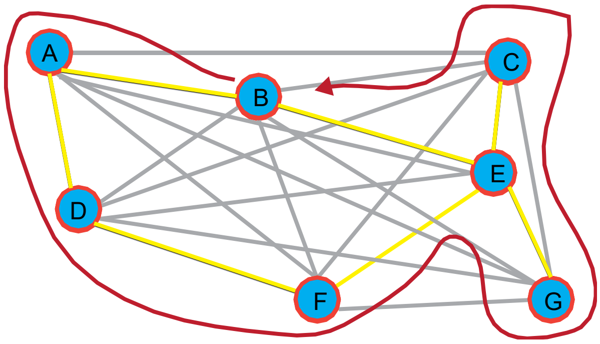


לצורך העניין, נניח שהתחלנו מ-B, ואנחנו רצים בסדר תחילי (תזכורת – אמצע, שמאל, ימין). סדר המעבר על הקדקודים יהיה: b, e, c, e, g, e b, a, d, f, d, a, b. ובצורה יותר ויזואלית –

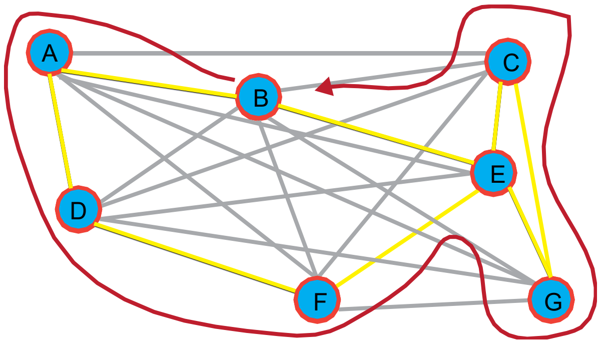


עכשיו נעבור על המסלול, ונבדוק איפה יש לנו חזרה אחורה - b, a, d, f, d, a, b, e ,g ,e, c, e, b.

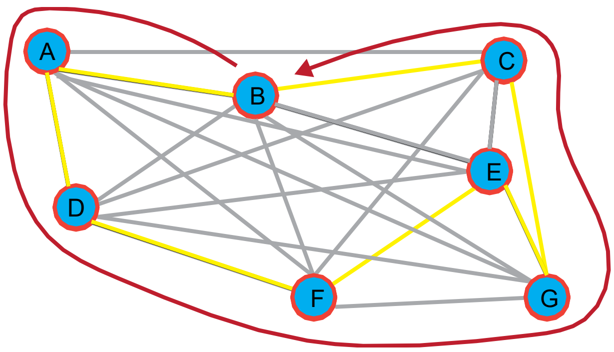
כשאנחו מגיעים לקדקוד f, אנחנו חוזרים על שלושה קדקודים בהם כבר ביקרנו. לכן, אנחנו יכולים לעשות קיצור דרך, ולעבור ישירות מ-f ל-e.



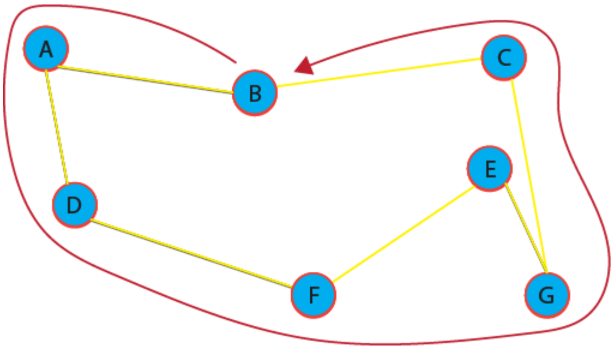
אנחנו יכולים להיות די בטוחים, שהמסלול (האדום) קצר יותר מקודם או לפחות לא יותר ארוך (בינינו, חתכנו פה שלושה קדקודים, קשה להאמין שהוא יהיה שווה). ועכשיו נעבור לחזרה הבאה - b, a, d, f, e ,g ,e, c, e, b. ונביא את הקיצור הנוסף –



ועכשיו, לחזרה האחרונה - b, a, d, f, e ,g, c, e, b. –



ולמעשה סיימנו את הרצת האלגוריתם. עכשיו מובטח לנו שיש לנו מסלול לא רע, וזמן הריצה שלו היה סביר.



כל שנותר לנו עכשיו הוא להוכיח את זמן הריצה שהוא בהחלט סביר, ואת חסם היחס של הקירוב.

<u>הוכחת זמן ריצה:</u> בחירת הקדקוד היא כמובן פעולה יחידה. מאחר הגרף שלנו הוא מלא, הריצה של פרים עליו תהיה בסדר גודל של V^2, שאר המעברים פחות דומיננטיים, כי הם בסדר של $|V|$. אז זמן הריצה של הקירוב, הוא בהחלט סביר.

<u>הוכחת חסם יחס:</u> אנחנו טוענים כי יחס הקירוב, הוא פי 2 מתוצאה האופטימלית. כלומר אם נגדיר את H להיות המסלול שמצאנו, ואת H^* להיות האופטימלי, אזי חסם היחס הוא $c(H) \leq 2c(H^*)$.

נאמר כך – אם נתון לנו H^* שהוא המעגל ההמילטוני המינימלי, אזי כל מחיקה של אחת מהקשתות תיתן לנו עץ פורש בגרף. ואם יש לנו את T שהוא העץ הפורש המינימלי בגרף, אנחנו יכולים להסיק כי $c(T) \leq c(H^*)$, ולו רק מהסיבה שבמעגל ההמילטוני יש לנו קשת נוספת שלא קיימת בעץ.

עכשיו, מבחינת הסריקה על קדקודי העץ (שנקרא לה W), מאחר ואנחנו עוברים על כל קדקוד גם בהלוך וגם בחזור, התוצאה היא שהמשקל של המסלול הוא פי 2 מהמשקל האמיתי של העץ – $c(W) = 2c(T)$.

משילוב השוויון הזה, ואי השוויון הקודם, אנחנו יכולים עכשיו לומר כי מאחר וראינו שמתקיים כי $c(T) \leq c(H^*)$, וגם $c(W) = 2c(T)$, אז אפשר לומר כי $c(W) \leq 2\, c(H^*)$. לאמור, עלות הסריקה אינה עולה על משקל המסלול האופטימלי פי 2.

עכשיו, המסלול שנוצר לנו W אינו המילטוני, כי בהגדרה אנחנו עוברים בכל מסלול יותר מפעם אחת. אבל מאחר ואנחנו שיפרנו את המסלול על ידיד הורדת החלק הכפול, אנחנו יודעים שעכשיו יש לנו מסלול המילטוני שהוא לא יותר גרוע מהסריקה W (וכאמור, בשאיפה אפילו יותר טוב). כלומר $c(H) \leq c(W)$, המסלול שמצאנו לסוכן, הוא חסום מלמעלה על ידי הסריקה שעשינו. והרי (לעשות עת זה בניגון של לימוד), אמרנו שהמסלול הזה בעצמו חסום מלמעלה על ידי $2c(H^*)$! אז נעשה את הדבר הטרנזטיבי המתבקש, ונגדיר כי $c(H) \leq 2c(H^*)$. מש"ל.

בעיית הסוכן הנוסע הכללית

דיברנו על בעיית הסוכן הנוסע, תחת הגדרה מאוד ספציפית – קיום אי שיוויון המשולש. אבל מה קורה באשר אי-השוויון לא מתקיים? נגיד מדובר בסוכן נוסע על קורקינט, ובחלק מהדרכים בין הערים יש ירידה ארוכה וכיפית, כך שנחסך ממנו מאמץ והוא פשוט יכול לגלוש לו בכיף לעיר הבאה. במקרה כזה אולי נמשקל את הדרך בציון נמוך יותר מאשר המרחק האמיתי. לו יקרה דבר כזה, כל בסיס העבודה שלנו על שיפור הדרכים עלול להיהרס – יכול להיות שהוא גולש לו בכיף בירידה לכיוון עיר מסוימת, השיער שלו

(זה שמחוץ לקסדה לפחות) מתנפנף לו ברוח, הוא נהנה מהבריזה, ואז הוא מגלה שהכרחנו אותו ללכת במסלול שיש בו עליה רק כי הוא לכאורה קצר יותר.

מסתבר, שאכן לבעיה הכללית, אין לנו פתרון. יותר מזה – אנחנו טוענים, שאין בכלל פתרון אלגוריתמי לבעיה זאת, אלא אם P=NP. כלומר, אם היינו מוצאים בעיית קירוב לבעייה הזאת, בהנחה שאנחנו מדברים על זמן פולינומי, אז אנחנו כבר מעבירים את הבעיה למחלקה הפולינומית.

<u>הוכחה:</u> נוכיח זאת בדרך השלילה. נניח שיש אלגוריתם קירוב פולינומי, וכמו כן P≠NP. אנחנו נראה כי מצב כזה גורר כי בעיית הסוכן הנוסע היא P, מה שגורר בתורו כי P=NP.

נניח כי יש לנו אלגוריתם קירוב לבעיה הכללית A, בעל חסם יחס $\rho \geq 1$ כלשהו, כאשר אנחנו יוצאים מנקודת הנחה ש-ρ הוא מספר שלם. במקרה כזה, ניקח מופע כלשהו של בעיית המעגל ההמילטוני (שכזכור, אנחנו עושים רדוקציה מהמעגל ההמילטוני לסוכן הנוסע), ונעביר אותו ברדוקציה מתאימה לבעיית הסוכן הנוסע – אנחנו צריכים ליצור כעת גרף שלם, ובשביל זה אנחנו נצטרך להוסיף קשתות. מה שנעשה הוא ככה – עבור כל קשת שהייתה קיימת כבר בגרף נעביר אותה כמו שהיא, ונמשקל אותה ב-1 (כאשר אנחנו יודעים שאי שם אמור להסתתר לו גם מעגל המילטוני), את כל שאר הקשתות שנוסיף נמשקל בעלות של חסם היחס של אלגוריתם הקירוב שנמצא +1 -כלומר יהיה לנו משקל של לפחות 2 עבור כל קשת כזו.

עכשיו, אם אכן היה מעגל המילטוני בגרף המקורי G, אז גם בגרף החדש G' יהיה מסלול מתאים לסוכן בעלות של |V|, כי הרי כל הקשתות המקוריות ממושקלות כ-1. אם אין מעגל המילטוני ב-G, אז הריצה של אלגוריתם הקירוב, יביא תוצאה שהיא גדולה ממש מחסם היחס (כי משקלנו את הקשתות בגדולות יותר מהחסם).עכשיו נשים לב, דבר כזה אפשרי רק אם אי השיוויון לא מתקיים – כי אז יכול להיות שקשת ישירה תהיה יקרה יותר מאשר מעבר דרך מספר קשתות שונות.

אבל, אם אנחנו אומרים שאלגוריתם הקירוב מביא תוצאה שתחת חסם היחס, דבר כזה לא יתבן, כי כל שימוש בקשת שאינה בגרף המקורי מקפיצה לנו את משקל המסלול לכזה שהוא מעבר לחסם, ועוד עשינו את זה בזמן פולינומי, ודבר כזה כמובן לא ייתבן, ויש פה סתירה. מש"ל.

בעיית אריזות המיכלים (Bin Packing)

את בעיית אריזות המיכלים הצגנו בפרק הקודם, וכעת נציע אלגוריתם קירוב הפותר אותו. נזכיר רק בקיצור – אנחנו צריכים למצוא דרך להכניס מספר פריטים בעלי משקל c, כאשר $1 \geq c \geq 0$, כלומר הוא שבר כלשהו, לתוך מינימום מיכלים בעלי קיבולת 1.

על מנת לעשות זאת, יש לנו אפשרות להשתמש באלגוריתם Best-Fit (בדיוק כמו במערכות הפעלה), בו נכניס את האיברים לתוך המיכל שייתן לנו את המרווח הקטן ביותר (לאחר ההכנסה). כמובן שנוכל גם למיין קודם את הפריטים וכך לחסוך עוד קצת זמן.

אפשרות נוספת, שעליה נרחיב, היא להשתמש באלגוריתם First-Fit (כן, כמו במערכות הפעלה!) בו אנחנו מכניסים את הפריט למיכל הראשון שאנחנו רואים שיש בו מספיק מקום. אם לא נמצא מיכל שנוכל להכניס אליו את הפריט הדרוש, ניקח מיכל חדש ונשתמש בו.

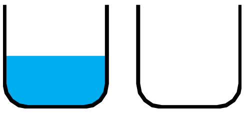
כמובן שאם נעשה ככה, לא נגיע לרמה הכי אופטימלית, מסיבה פשוטה – ושוב, נשתמש במושגים של מערכות הפעלה – אם נמלא את המיכלים בצורה כזאת, יווצר לנו שברור פנימי, המכונה פרגמנטציה.

כלומר, אם באחד המיכלים יש 0.2 מקום פנוי, וכל הפריטים שלנו גדולים מזה, בסופו של דבר יישאר לנו שטח לא מנוצל. נשאלת השאלה במובן – מהו חסם היחס?

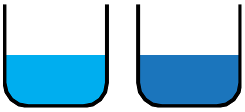
אנו טוענים כי חסם היחס (לפחות עבור האלגוריתם First-Fit) הינו 2, ונוכיח זאת:

עבור כל שני מיכלים נתונים, לא יכול להיות שהמצב הסופי של שניהם יהיה חצי. מדוע? מהסיבה הפשוטה – אם היה מצב כזה הם היו אמורים להצטמצם! נראה מה הכוונה:

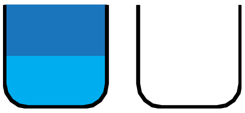
נניח ונתונים לנו שני מיכלים, כאשר אחד מהם מלא עד חציו (או ריק, תלוי כמה אתם שבוזים מהחומר):



ועכשיו ייכנס לנו עוד פריט שיהיה בדיוק חצי. במובן שאין שום סיבה שהוא ייכנס למיכל השמאלי בתצורה הבאה:



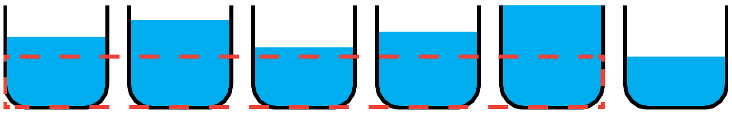
אלא, הוא ימלא את המיכל הראשון בדיוק עד שפתו –



ורק בשלב הבא, נתחיל למלא את המיכל השני. זה דבר שלא צריך ממש להוכיח, אלא מובן מאליו.

כעת, ניתן לומר כי בסוף הריצה, לו השתמשנו ב-m מיכלים, בוודאי m-1 מיכלים מלאים ביותר מחצי (כי אם היו שניים כאלה, הם היו מצטמצמים). עכשיו, על מנת לחשב את חסם היחס, נעשה דבר כזה – קודם כל, נסמן את מה שבטוח יש לנו – אם השתמשנו במיכלים באופן הבא:

נשים לב שמתוך m=6 מיכלים בהם השתמשנו, ישנם חמישה שמלאים מעבר לחציים. ובשביל החישוב הסופי, נראה את הדבר הבא – מספר המיכלים האופטימלי, במובן חסום על ידי סכימה של כל הפריטים. כלומר $OPT \geq \sum_{i=1}^{n} a_i$. למה הוא גדול מכל זה? מאחר והמספר של המיכלים הוא מספר שלם. אנחנו לא נשתמש ב-6.45 מיכלים, אנחנו נשתמש ב-7 מיכלים, אבל תמיד יכול להיות שהכל ייסכם יפה ויהיה בדיוק מספר שלם. כמה מיכלים זה יצא? אם ניקח את הדוגמה של מילוי המיכלים שראינו עכשיו, אנחנו יכולים לומר בוודאות שנצטרך יותר מאשר $\frac{m-1}{2}$ המיכלים שיש לנו. איך הגענו לחישוב המופלא הזה? ככה –



השטח המקווקו בכתום (למי שהדפיס בשחור-לבן, אז באפור בלשהו שאני מקווה ששונה מספיק מהאפור השני), הוא סימון של m-1 המיכלים שמלאים בוודאות עד חציים. כלומר, אם יש לנו פה חמישה מיכלים באלה, בטוח שנצטרך לפחות 3 מיכלים (שוב, אנחנו מעגלים למעלה), ורק לאחר שיהיה לנו את מספר

המיכלים הזה, נוכל להתחיל ולהכניס את שאר הפריטים עד שנקבל המספר האופטימלי האמיתי. נאחד את כל מה שראינו לכדי נוסחה אחת –

$$OPT \geq \sum_{i=1}^{n} a_i > \frac{m-1}{2}$$

נסביר שוב בשביל לוודא שזה מובן – מספר המכלים האופטימלי חסום על ידי סכימה של כל הפריטים, בעיגול כלפי מספר שלם, שהוא גדול ממש מחצי כמות המיכלים בחיסור אחד מהם (גדול ממש, מאחר ויש לנו את המיכל האחרון שאנחנו טוענים שיש בו משהו, ותכלס לא משנה לנו כמה בדיוק, וכמו כן, בכל המיכלים אנחנו מניחי שיש קצת יותר מחצי, ובחישוב הכולל בטוח שזה יצא לפחות מיכל אחד יותר מהדרוש).

עכשיו, על מנת לראות מה השיטה שהבאנו נותנת לנו (להזכירכם – First Fit), שאותו אנחנו חוסמים עם השבר מימין, נכפיל את הכל ב-2 בשביל שלא יהיה לנו שבר, ויצא לנו – $2OPT > m-1$, ואם ננסה לצמצם את החיסור של המיכל האחרון, נוכל לומר כי האופטימום חסום על ידי $2OPT \geq m$.

מש"ל.

בהצלחה!

נספח א'
סיכום אלגוריתמים על גרפים[15]

אלגוריתמים כלליים

שם האלגוריתם	מטרה	קלט	תוצאה	פעולת האלגוריתם	זמן (מבנה נתונים)	הערות
חיפוש לרוחב (BFS)	מרחק מינימלי מהשורש, מספר צלעות	גרף מכוון / לא מכוון	עץ רוחב – כל הקדקודים אליהם ניתן להגיע מקדקוד המקור (s), בתוספת המרחק של כל קדקוד.	מכניסים את השורש לתור, מוציא מראש התור קדקוד, ומכניס את כל שכניו לסוף התור, תוך כדי עדכון המרחק מהשורש	Θ(V+E) – הגדול ביניהם (רשימה)	פרים ודייקסטרה מתבססים על הרעיון הזה.
חיפוש לעומק (DFS)		גרף מכוון / לא מכוון	יער עצי עומק, זמני גילוי וסיום.	עוברים על כל הקדקודים שלא סיימו טיפול, מטפלים בכל שכניו וקובעים בהם את הפרמטרים הרלוונטיים	Θ(V+E) (רשימה)	
גרף דו צדדי	בדיקה האם גרף נתון הוא דו צדדי	גרף לא מכוון	כן/לא	בוחרים קדקוד מקור, ומריצים BFS. לאחר הריצה בודקים אם יש קשת בין קדקוד במרחק זוגי לזוגי אחר, או בין אי-זוגי לאי-זוגי אחר.	Θ(V+E) (רשימה)	שימוש בBFS, אפשר גם עם DFS וגם עם גרף מכוון
מיון טופולוגי	סידור ליניארי של כל קדקודי הגרף בכיוון מסוים	גמ"ל	פרמוטציה של הקדקודים (על פי סדר ההופעה)	מבצעים DFS. ממינים את זמני הסיום בסדר יורד.	Θ(V+E) (רשימה)	שימוש בDFS
רכיבים קשירים היטב	קבוצות הקדקודים בהם ניתן להגיע מכל קדקוד לכל קדקוד	גרף מכוון	קבוצות קדקודים	חישוב DFS, חישוב הגרף המשוחלף, ואז חישוב DFS על הגרף G^T כאשר הקדקודים ממויינים בסדר יורד של זמני הסיום בגרף המקורי.	Θ(V+E) (רשימה)	

[15] מעובד מתוך הקובץ שבמודל

אלגוריתמים לעצים פורשים מינימליים

שם האלגוריתם	מטרה	קלט	תוצאה	פעולת האלגוריתם	זמן	הערות
עץ פורש כללי	עץ פורש	גרף קשיר עם משקולות	אוסף צלעות המחברות בין כל קדקודי הגרף, כך שנוצר עץ.	יוצרים קבוצת צלעות ריקה, ובבל פעם מוסיפים קשת "בטוחה" (שלא פוגעת בתכונות העץ) לקבוצה.	אלגוריתם כללי תלוי במימוש	
קרוסקל	עץ פורש מינימלי	גרף קשיר עם משקולות	עץ פורש בעל משקל קשתות מינימלי בגרף.	מיון הקשתות בסדר עולה, הוספת כל קשת שלא סוגרת מעגל לתוך העץ.	Θ(ELog(E+V)) תלוי ביחס קדקודים/קשתות	שימוש באלגוריתמים של מבני נתונים לקבוצות זרות.
פרים	עץ פורש מינימלי	גרף קשיר עם משקולות	עץ פורש בעל משקל קשתות מינימלי בגרף.	מתחילים בקדקוד לפי בחירה, ובבל פעם מוסיפים את הקשת הקלה ביותר המחוברת לאחד הקדקודים	Θ(ELog))	

אלגוריתמים למציאת מסלולים קצרים ביותר

שם האלגוריתם	מטרה	קלט	תוצאה	פעולת האלגוריתם	זמן	הערות
דייקסטרה	מציאת מסלול קצר ביותר מקדקוד יחיד	גרף מכוון, משקולות אי שליליים	עץ מסלולים קצרים ביותר, מושרש בקדקוד המקור	איתחול המרחקים ו"הקודמים" של כל הקדקודים. מוציאים את קדקוד המקור מהתור, בכל איטרציה מוציאים את הקדקוד בעל המרחק הנמוך ביותר ומבצעים הקלה אל הקדקודים האחרים	Θ(ELogV)	
בלמן-פורד	מציאת מסלול קצר ביותר מקדקוד יחיד	גרף מכוון עם משקולות	עץ מסלולים קצרים ביותר מהמקור + ערך בוליאני האם יש בגרף מעגל שלילי	מבצעים V-1 הקלות על כל הקשתות, ופעם נוספת על מנת לבדוק האם יש שינוי במרחקים.	O(VE)	
פלויד-ווארשל	מציאת המסלולים קצרים ביותר מכל קדקוד לכל קדקוד	גרף מכוון עם משקולות	מטריצת מרחקים בין הקדקודים, וכן מטריצה של האבות של כל קדקוד	בבל איטרציה בודקים אם יש שינוי במרחקי הקדקודים בהתחשבות בקדקוד נוסף החל מ-1 ועד n.	$\Theta(V^3)$	אלגוריתם דינאמי

אלגוריתמים לרשת זרימה

שם האלגוריתם	מטרה	קלט	תוצאה	פעולת האלגוריתם	זמן	הערות
פורד-פולקרסון	מציאת זרימה מקסימלית אפשרית	גרף מכוון, עם נקודת התחלה וסיום.	זרימה מקסימלית ברשת	בבל מציאת אפשרות לשיפור הזרימה, נשתמש במסלול שנמצא ונעדכן את הזרימה	לא קבוע	יותר "שיטה" מאלגוריתם. עלול להגיע לזמן ריצה ארוך מאוד
אדמונדס-קארפ	מציאת זרימה מקסימלית אפשרית	גרף מכוון, עם נקודת התחלה וסיום.	זרימה מקסימלית ברשת	נריץ BFS על רשת הזרימה, ונעבור על המשלולים החל מהמסלול הקצר ביותר, ועד שאין לנו אפשרות להזרים יותר.	$\Theta(VE^2)$	

נספח ב'
סדר הוכחה לשייכות לNPC

הוכחת שייכות ל-NP

עלינו לכתוב אלגוריתם אימות A(x,y), כאשר x זה מופע של הבעיה, ו-y זה קלט שאמור לאמת אותו. האימות אמור להיות בזמן פולינומי.

עבור הקלט שמתקבל אנחנו צריכים להריץ סדרת בדיקות –

- קודם כל – עלינו לוודא שהוא בכלל מתאים לבעיה
- וידוא שהקלט **יכול** לפתור את הבעיה –מכיל את מספר האיברים המתאים לפתרון הבעיה, איברים לא חוזרים על עצמם בלי צורך וכו'
- וידוא שהקלט **פותר את הבעיה** – הרצה של בדיקה עבור הבעיה הנתונה – סכימה של האיברים, מעבר על מסלול וכו

מומלץ להוסיף תנאי יציאה כשאפשר – אם הקלט לא בגודל הגיוני ביחס לתשובה המצופה, להחזיר ישר false ולצאת מהבעיה.

הוכחת נכונות האימות – להוכיח שהאלגוריתם אכן מאמת לנועם הקלט, שהמופע שייך לבעיה, אם האלגוריתם נכתב נכון, אז אפשר לומר "אם הקלט היה קצר/ארוך מידי, היינו נופלים בשורה 1 וכו"

הוכחת זמן ריצה – אנחנו דורשים זמן פולינומי, ולכן אנחנו עוברים שורה שורה ורושמים את זמן הריצה (ברוב המקרים זה סביב n^2, בגלל שצריך לאמת שאין כפילויות וכאלה, אבל אל תכתבו בלי נימוק), בסוף כותבים את זמן הריצה הכולל בסדר גודל.

אם כל זה נכון – הוכחנו שהשפה שייכת לNP.

הוכחת שייכות ל-NPH

עלינו להוכיח שניתן לעשות רדוקציה בזמן פולינומי מכל בעיה בNP לבעיה הנתונה, את זה אנחנו עושים על ידי רדוקציה מבעיה שאנחנו מכירים כבר שהיא בNPH.

תזכורת – הסימון הוא כזה – $L_1 \leq_p L_2$, כאשר L_2 זה השפה שצריכה להכנס לNPH, ו-L_1, היא השפה ממנה עושים את הרדוקציה.

סדר בניית הרדוקציה-

פירוט של השינויים הנצרכים ב-L_1, על מנת שיהיה מתאים לשפה L_2. (הוספת משקלות, קשתות, מספרים, מה שלא יהיה)

הוכחת זמן ריצה – השינויים שאנחנו עושים ברדוקציה צריכים להיות גם הם תחת זמן פולינומי, לכן יש לפרט את זמן הריצה עבור השינויים (הוספת קשתוות לגרף שלם – $O(n^2)$, משקול קשתות – $O(|E|)$, ובכן על זה הדרך)

הוכחת נכונות הרדוקציה – $x \in L_1 \leftrightarrow f(x) \in L_2$

אנחנו צריכים להוכיח את שני הכיוונים, לפעמים זה מאוד פשוט, לפעמים צריך לפרט יותר.

$x \in L_1 \rightarrow f(x) \in L_2$

אנחנו צריכים להראות כיצד **כל** מילה ב-L_1, עוברת אחרי הרדוקציה להיות מופע של L_2, כלומר עלינו להוכיח שכל תנאי הבעיה L_2 מתקיימים אחרי הרדוקציה.

$x \in L_1 \leftarrow f(x) \in L_2$

אם עשינו את הרדוקציה נכון, אז פירוק של הבעיות יחזיר אותנו לשפה המקורית. צריך לזכור – לא מדובר על כל הבעיות במופע החדש, אלא רק על אלו שמקיימות את התנאים של הרדוקציה שציינו -כלומר אם החלטנו שאנחנו ממקשלים את הכל 1, אז אנחנו צריכים להוכיח שעל כל מופע של הגרפים שממושקלים כולם ב-1, אנחנו יכולים להוריד את המשקול בחזרה, ולהיות במופע של השפה המקורית ממנה יצאנו לרדוקציה.