סיכום

מבנה נתונים ב'

סוכם משיעוריו של הרה"ח

אריה ויזן

בתוספת הערות מחכימות מהמצגות של הרבנית החשובה

אלישבע בנש"ק-דוקוב

ושל רשכבה"ג עוקר הרים הרד"ר

אלכסנדר חייט

עם פירוש

"רי"ח טוב"

נערך בחסדי ה'

מאיתי הצב"י יוחנן חאיק

שנת 1011010010010

"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**
סיבוביות – חזרה **4**
עצי החלטה **6**
מבני נתונים לייצוג קבוצות זרות **7**
קבוצות זרות 7
פעולות על קבוצות זרות 7
מימוש על ידי גרפים 8
מימוש על ידי רשימה מקושרת 9
מימוש ב"יער של עצים" 11
איחוד על פי דרגה 12
כיווץ המסלולים 12
סיבוכיות הפעולות 14
פונקציית אקרמן 14
דוגמה לפונקציית אקרמן 15
קוד האפמן **17**
הוכחה שעץ האפמן חייב להיות מלא 18
השלבים לבניית עץ מייצג של קוד האפמן 19
בניית עץ האפמן לדוגמא 19
פסאודו-קוד לבניית עץ בקוד האפמן 20
עלות עץ בקוד האפמן 20
אלגוריתמים חמדניים 21
תבונות עצים אופטימליים 21
עצי חיפוש מאוזנים **23**
עצי 2-3 .... **24**
הכנסת איבר לעצי 2-3 25
דוגמה לבניית עץ 2-3 26
מחיקת איבר מעץ 2-3 27
דוגמה למחיקת עץ 2-3 29
עצי B **32**
תבונות עצי B 32
פעולות על עץ B 33

בניית עץ B 33
בניית עץ לדוגמא 34
פתרון נוסחאות נסיגה 38
צורה כללית של נוסחת נסיגה 39
שיטת ההצבה 41
שיטת האיטרציה 42
עצי ריקורסיה 44
משפט האב (מאסטר) 45
מציאת האיבר ה-i 48
אלגוריתם דטרמיניסטי 49
טבלאות גיבוב 52
מיעון ישיר 52
מיעון מחושב (גיבוב) 52
הנחת הגיבוב האחיד הפשוט 53
מקדם העומס 53
זמן ממוצע של חיפוש 54
פונקציות גיבוב 54
פונקציית הערך התחתון 54
פונקציית החילוק 54
פונקציית הקיפול 55
פונקציית אמצע הריבוע 55
פונקציית הכפל 56
מיעון פתוח 56
סריקה ליניארית 56
בדיקה ליניארית 58
בדיקה ריבועית 59
גיבוב כפול 60
גיבוב אוניברסלי 63
בניית פונקציית גיבוב אוניברסלית 64
גרפים 66
הגדרות רשמיות 66
ייצוגים של גרף 69
חיפוש לרוחב 70

נבונות BFS (למציאת מסלולים קצרים ביותר) 70
תת גרף הקודמים 72
עץ רוחב 73
חיפוש לעומק Depth First Search 73
משפט הסוגריים 74
סיווג הקשתות בDFS 75
משפט המסלול הלבן 76
גמ"ל – גרף מכוון ללא מעגלים 77
מיון טופולגי 77
רכיב קשיר חזק 79
שיחלוף של גרף מכוון (Transpose) 80
אלגוריתם למציאת SCC 80

סיבוכיות – חזרה

כפי שלמדנו בסמסטרים הקודמים, כאשר אנחנו רוצים לחשב מהירות של תוכנה וזמן ריצה שלה, מהירות של המחשב עליו התוכנה רצה הוא נתון פחות משמעותי ביחס לזמן הריצה והסיבוכיות של האלגוריתם.

כמו שראינו כבר בקורס מבנה המחשב, תנופת הפיתוח במרוצת השנים גרם לכך שאין לנו בעיה של איחסון מידע רב יחסית תחת מקום מוגבל, ניתן היום לשמור אפילו כמות של טרה-בייט (1000 גיגה) על דיסק און קי בודד. מה עוד, שהעולם הולך ומתקדם לעבר הBig Data, כמויות אדירות של מידע הנשמרות ואמורות להיות נוחות לגישה – ניתן לקחת לדוגמה רשת חברתית כמו "פייסבוק" המחזיקה מידע אודות מיליוני משתמשים, ותחת כל אחד מהם יש כמות מאוד גדולה של מידע, תמונות, קשרים למשתמשים אחרים ועוד. האתגר הגדול העומד בפני החברה הוא לא איפה ואיך לאחסן את כל המידע הזה, מאחר ולזה כבר יש פתרונות, אלא איך לגשת לכל נתון בצורה מהירה ומדויקת. דבר זה נעשה על ידי מיון נכון של מידע ודאגה לזמני ריצה מהירים ונגישים.

נחזור על הסימנים המוסכמים עבור מדידה של זמני הריצה, מאחר שאנחנו ממשיכים להשתמש בהם גם בקורס הזה כמדדים שונים לזמני ריצה של האלגוריתמים המוצעים.

ישנם שלושה (שהם חמישה) מדדים לסיבוכיות:

1. O – **חסם אסימפטוטי עליון** (גדול וקטן)

הגדרה: $f(n)=O(g(n))$ אם קיימים קבועים n_0, c כך ש- $f(n) \le cg(n)$ לכל $n \ge n_0$.
הגדרה שקולה: $f(n)=O(g(n))$ אם קיים $c > 0$ כך ש- $\lim_{n\to\infty}(f(n)/g(n)) \le c$.

על מנת שפונקציה כלשהי $g(n)$ תחשב חסם עליון של פונקציה אחרת $f(n)$, אנחנו צריכים לוודא שעבור כל מספר גדול מ-0 שייבחר (ואפילו שברים), החל מנקודה מסוימת הפונקציה $g(n)$ תהיה גדולה יותר משמעותית מהפונקציה האחרת $f(n)$. בהשאלה ניתן לומר זאת גם אחרת – אם נשאי, לאינסוף את שתי הפונקציות, כאשר: $f(n)/g(n)$ אנחנו נקבל השאפה ל-0. זה כמובן מאחר ואם הפונקציה התחתונה משמעותית גדולה יותר, השאפה שלה לאינסוף תיצור אינסוף במכנה, מה שיוביל את שתי הפונקציות לשאוף לכיוון ה-0.

2. Ω– **חסם אסימפטוטי תחתון** (גדול וקטן)

הגדרה: $f(n)=\Omega(g(n))$ אם קיימים קבועים n_0, c כך ש- $f(n) \ge cg(n)$ לכל $n \ge n_0$.
הגדרה שקולה: $f(n)=\Omega(g(n))$ אם קיים $c > 0$ כך ש- $\lim_{n\to\infty}(g(n)/f(n)) \le c$.

במקביל לחסם העליון, קיים גם החסם התחתון, המציין כי לא משנה כמה נגדיל את הפונקציה $g(n)$ עדיין תמיד היא תהיה קטנה יותר באופן משמעותי מהפונקציה $f(n)$. ובשאיפה לאינסוף מדובר על אותו דבר, רק שבמקרה זה אנחנו מניחים את הפונקציה $f(n)$ במכנה וככה בהשאפה לאינסוף אנחנו עדיין מקבלים מספר שהוא 0.

יש לזכור עבור שני המקרים של חסם עליון ותחתון, שאנחנו מחפשים את הפיתרון עבור מספרים גדולים ושאיפה לאינסוף, ולא רק עבור מקרים פרטיים, ולכן גם אם נמצא טווח שבו זה לא מתקיים, גדול ככל שיהיה, כל עוד בשאיפה לאינסוף הכללים עדיין מתקיימים יש ללכת לפיהם.

3. Θ – **חסם אסימפטוטי הדוק.**

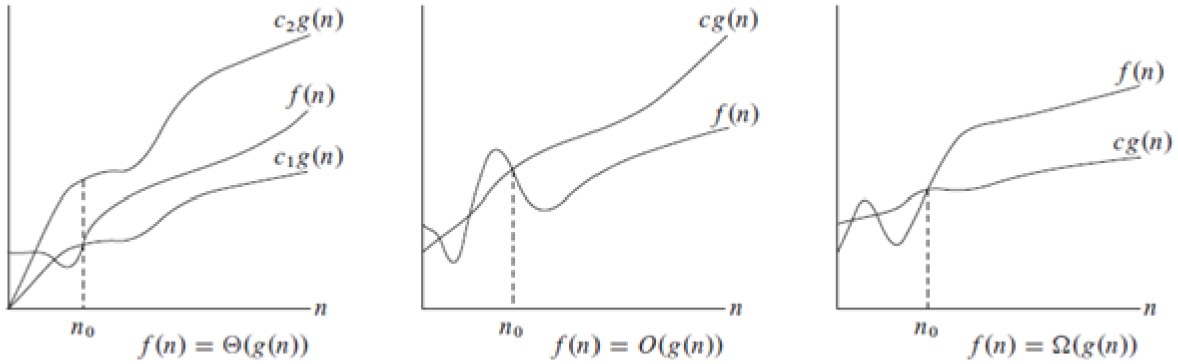
הגדרה: $f(n)=\Theta(g(n))$ אם קיימים קבועים n_0, c_1, c_2 כך ש- $c_1 g(n)\le f(n) \le c_2 g(n)$ לכל $n \ge n_0$.
הגדרה שקולה: $f(n)= \Theta(g(n))$ אם קיים $c > 0$ כך ש- $\lim_{n\to\infty}(f(n)/g(n)) = c$.

הגדרת החסם ההדוק הוא למעשה שילוב של שני החסמים – העליון והתחתון. בתיאור הפשוט – אנחנו מנסים לתחום את הפונקציה הנתונה לנו. איננו יכולים לקבוע בוודאות לאן היא שואפת, אך אם נוכל לתחום אותה מלמעלה ומלמטה תחת אותה פונקציה הנתונה לשינויים קטנים, אנחנו נוכל ללמוד מזה על ההתנהגות הכללית של הפונקציה. לשם כך עלינו למצוא שני מספרים קבועים שונים זה מזה, עבורם אותה פונקציה g(n) תהיה פעם אחת גדולה יותר משמעותית מ f(n)– חסם עליון, ופעם אחת, עבור קבוע אחר קטנה יותר משמעותית – חסם תחתון.

כאשר נשאיף את שתי הפונקציות האלו לאינסוף, לא נקבל אינסוף או 0, אלא מאחר ושניהם קרובות יחסית אחת לשניה ומאותו סדר גודל אנחנו נקבל בסוף מספר שהוא קבוע ושונה מאפס.

בדרך כלל אנחנו מחפשים את הגבול העליון שהוא יותר רלוונטי לבדיקת המקסימום ריצה של האלגוריתם. ומאחר והוא מוגדר כחסם עליון אנחנו יודעים שעבורם גם אין שינוי, אך בחלק גדול מהמקרים אנחנו נגיע רק לחסם הדוק מאחר ויש הרבה משתנים שמשפיעים על זמן הריצה של הפונקציות השונות.

בנוסף, לחסם העליון והתחתון, מלבד מה שהזכרנו קיימים גם השיוכים הקטנים לחסמים שהם f(n)=o(g(n)), ובן fn=ω(g(n)) המתייחסים לחסמים קטנים או גדולים **ממש** אך בקורס זה אנחנו לא משתמשים בהם.



$f(n) = \Theta(g(n))$ (a) $f(n) = O(g(n))$ (b) $f(n) = \Omega(g(n))$ (c)

עצי החלטה

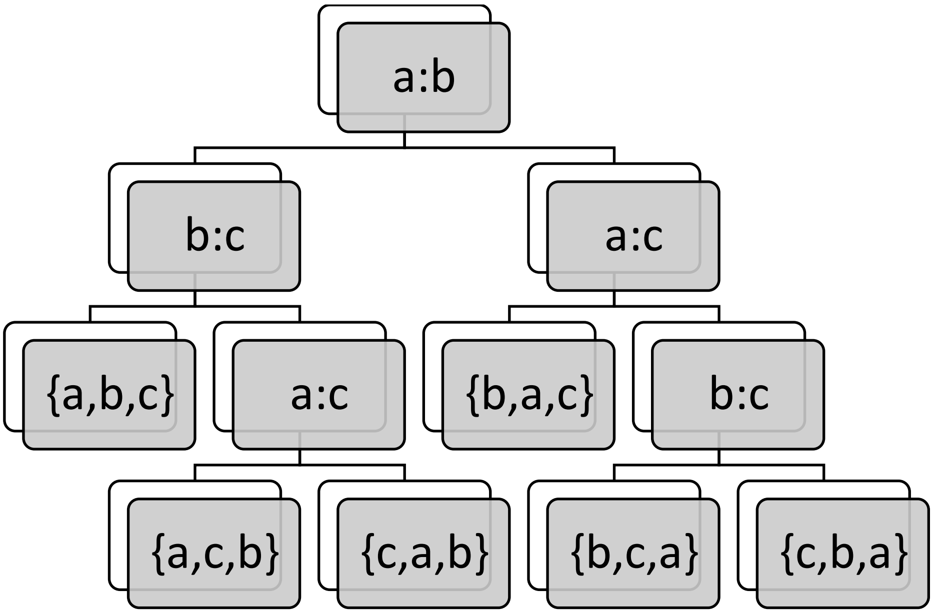
"עץ החלטה הוא עץ בינארי[1] מלא המורכב מצמתי החלטה שבכל אחד מהם נבדק תנאי מסוים על מאפיין מסוים של התצפיות ועלים המכילים את הערך החזוי עבור התצפית המתאימה למסלול שמוביל אליהם בעץ". (ויקיפדיה)

עץ החלטה נותן לנו חיווי ויזואלי על סדר פעולות ואפשרויות המובילות מאחת לאחרת. בכל צומת בעץ ההחלטה ישנו ערך שהוא למעשה שאלה/החלטה שצריך להחליט, וממנו מתפצלות מספר תשובות למקומות שונים, שם מגיעים לצמתים אחרות.

למעשה זה דומה לכך שיש לנו תפריט טלפוני, בו מציגים מספר אפשרויות עד שמגיעים לפעולה מסויימת. או סדרה של שאלות שבעצם מובילות לכמה אפשרויות שמוסיף מידע אחד על גבי השני ומוביל לתוצאה הסופית.

עץ החלטה בנוי מ"שורש" – השאלה הראשונית, ולה מספר תשובות אופציונאליות (לאו דווקא בצורה בינארית), כאשר כל צומת בנוי מהשאלה/החלטה המקומית של אותו צומת, ובנוסף יהיה רשימה של תשובות, שמוביל כל אחד עם פוינטר לשאלה הבאה. כאשר אם מדובר על החלטה סופית, ה"ליסט" הוא ריק ולא מוביל לשום מקום (אפשרות נוספת, להכניס משתנה בוליאני שאומר אם מדובר על עלה או לא).

דוגמא פשוטה לעץ החלטה, עליה הסברנו בחלק א' של הקורס הוא עץ החלטה עבור מיון בועות. ניקח קבוצה קטנה של מספרים: {a,b,c}. נתונים לנו שלושה מספרים שונים , שאנחנו א יודעים את הסדר שלהם, ועל ידי מספר השוואות אנחנו יכולים לקבוע בעצמנו מה הסדר הנכון של המספרים. בדרך כלל את הערכים אותם אנחנו נ=משווים נפריד על ידי סימון של נקודתיים ":", כאשר הפניה ימינה או שמאלה במורד העץ תהיה לכיוון של ה"החלטה" אותה אנחנו קובעים מראש. בעץ זה הנקודתיים מוגדרות בתור השוואת גודל, בה אנחנו הולכים לכיוון הערך הגבוה ביותר מבין השניים. כך שאם אנחנו רואים ש a>b נפנה שמאלה, ולאחר מכן נבדוק את b>c ובמקרה זה נוכל לקבוע תוך שתי השוואות בלבד את הסדר הנכון ב {a,b,c}.



[1] על פי המתרגל דוד כהן: לא חייב להיות דווקא עץ בינארי

מבני נתונים לייצוג קבוצות זרות

קבוצות זרות

קבוצות זרות הינם קבוצות **דינאמיות** של אובייקטים, זאת אומרת שהקבוצה מאפשרת: 1. הוספה, 2. חיפוש ו-3. מחיקה. איברי הקבוצה יכולים להיות אובייקטים כלשהם ללא תלות מסויימת מסוג הערכים אותם האובייקטים מחזיקים.

התכונה העיקרית של קבוצות זרות, הינו שחיתוך של שתי קבוצות יניב "קבוצה ריקה", כלומר, כל פריט שייך אך ורק לקבוצה אחת בדיוק, לא יותר ולא פחות.

למשל: אם תלמיד=object, ניתן לדבר על קבוצה של תלמידים.

כיצד מוודאים שאין אף אלמנטים משותפים בין הקבוצות השונות? כל קבוצה מיוצגת על ידי "**נציג**[2]". הנציג הוא חלק אינטגרלי מהקבוצה ומכיל את כל המידע המוכל בתוך כל אובייקט אחר בתוך הקבוצה ושווה לכל שאר האיברים, אך יש בו תוספת מיוחדת מהגדרתו כנציג-

הנציג מציין את ההשתייכות של כל האיברים תחת אותה קבוצה. אין בו ערך מסוים בפני עצמו והוא לא נבחר דווקא בתור האיבר הגדול ביותר או תחת פרמטר כזה או אחר, אלא הוא פשוט מוגדר כנציג, וכל איבר אחר שייכנס לאותה הקבוצה מיד יקבל ערך המצביע לכיוון הנציג. כך שאם מעוניינים לבחון את אחד מהאיברים, ניתן לבדוק מי הנציג שלו וכך לשייך אותו לקבוצה המתאימה.

כמובן, ששני איברים המיוצגים על ידי שני איברי נציג שונים, שייכים לשתי קבוצות שונות, ואין שום אפשרות שהם שייכים לאתה קבוצה. אם רואים שיש שני נציגים המכילים אותו ערך, הווה אומר - הם שווים, כנראה שמדובר באותו נציג ומאותה קבוצה, ולא בשתי נציגים מקבוצות שונות שבמקרה הם תואמים. ובעזרת הנציג ניתן גם לבדוק האם שני איברים שייכים לאותה קבוצה – במידה והנציג שלהם שווה – שניהם שייכים לאותה הקבוצה.

כל עוד הקבוצה קיימת בקבוצה עצמאית ולא נעשה בה שום שינוי מלבד שינויים בודדים של הכנסה והוצאה (כמובן שאין הגבלה להכנסות והוצאות מתוך הקבוצה), הנציג שלה נשאר ללא שינוי.

מלבד הפעולות המתבצעות בתוך הקבוצה עצמה, קיימות גם הפעולות המתבצעות על קבוצות, הפעולות על הקבוצות הן - "איחוד", "חיתוך", "משלים" ו"הפרש", כאשר, אם יתבצע איחוד בין שתי קבוצות, רק אחד מהנציגים יהיה הנציג של הקבוצה החדשה-המאוחדת, והשני ייעלם, כאשר אנו נלמד בהמשך מי מהם נעלם ומי ממשיך לייצג את שני הקבוצות, ומה הפרוצדורה של המעבר לנציג חדש.

פעולות על קבוצות זרות

1. **Make-Set(x)** – יצירת קבוצה.
פעולה זו מתבצעת רק פעם אחת, בדיוק כמו בנאי שלמדנו בC++. 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, אלא לנציג בלבד. (אם בשני זימונים שונים על שני איברים שונים הערך המוחזר

[2] representative

מהפונקציה תהיה זהה, אנו מבינים שמדובר על כך שהאיברים נמצאות 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.
אך מה קורה אם מדובר בשני איברים מאותה קבוצה? במקרה זה כלום לא מתבצע, מאחר ואיחוד של אותה קבוצה עם עצמה, מחזיר את עצמו. כמו שכבר אמרנו, כאשר מתקבלת פקודת האיחוד, דבר ראשון, אנו קוראים לפונקציה findSet עבור כל אחד מהאיברים בנפרד, ואז בודקים האם מדובר בקבוצה שווה (על ידי בדיקת שוויון בין הנציגים), ובמידה שיש שוני בין הנציגים האיחוד מתבצע על ידי פונקציה הנקראת Link(r1,r2). הפונקציה Link מקבלת שני נציגים (representive), כאשר ידוע לנו שהם שונים, ואז היא מאחדת בין הקבוצות. אחד משני הנציגים הופך להיות הנציג של הקבוצה המשותפת – הבחירה באיזה נציג מבין השניים מתבצע באפונים שונים, כאשר ישנם שני אלגוריתמים עבור מבני נתונים שונים, כאשר ניתן לאחד רק אותו סוג של מבני נתונים ללא הבלאות, שבודקים את גודל הקבצות ופרמטרים נוספים ומחליטים על "הכתרת" הנציג החדש.

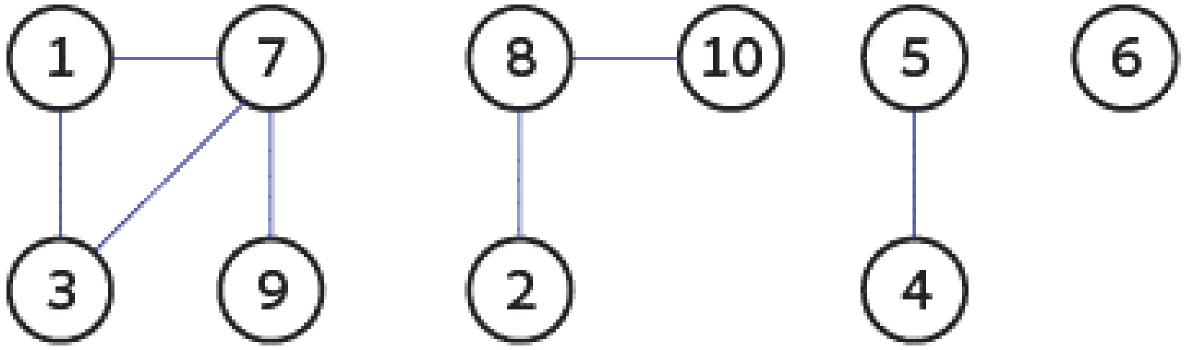
זמן הריצה המחושב עבור הפעולות, הוא לא זמן הריצה המדויק עבור כל פעולה בנפרד, אלא אנו מחפשים את הזמן המוערך עבור כל הפעולות המבוצעות על מבנה הנתונים. מהאחר וחלק מהפעולות מבוצעות בתדירות נמוכה יותר אך עולים בזמן ריצה גבוה, וישנן פעולות המבוצעות בתדירות גבוהה אך בזמן ריצה נמוך.

מימוש על ידי גרפים

כאשר מסתכלים על גרפים, אנו בוחנים את הקדקודים של כל גרף, ואנו מחפשים לבדוק האם הגרפים קשורים אחד לשני. על מנת לבדוק זאת אנו צריכים לפרק אותם ל"רכיבי קשירות"[3].

אנחנו לוקחים את כל הקדקודים ומפעילים על כל אחד מהם makeSet, כאשר ברור שיתקבלו לנו קבוצות זרות, ואז אנו נעבור על הקשתות בין הקדקודים, ונתחיל לאחד בין הקבוצות (=הקדקודים) השונים לכדי קבוצה אחת גדולה. כאשר נסיים לעבור על הקבוצות, ונראה שיש איברים שלא שייכים לאותה קבוצה, נבין שהם שייכים כל אחד לגרף אחר.

לדוגמא, נבחן את הגרפים הבאים:



[3] כל נושא הגרפים ו"רכיבי קשירות" יוסבר במשך. אך על רגל אחת – רכיב קשירות הוא קבוצה של קדקודים וצלעות המרכיב ביחד גרף מקסימלי בו ניתן להגיע לכל הקדקודים. באיור הדוגמא – הקדקודים {1,3,7,9} הם רכיב קשירות, וכן כל אחד מהגרפים הקטנים יותר הוא רכיב קשירות, כאשר הקדקוד הבודד נחשב גם הוא גרף בפני עצמו, ורכיב קשירות בפני עצמו.

בתחילה הפונקציה עוברת בלולאה על כל הקדקודים במספר האיברים הקיימים בגרף, ומפרידה כל אחת מהן לקבוצה בפני עצמה, ואז בודקת האם קיימות קשתות מחברות בין הקדקודים. הבדיקה על החיבורים הקיימים מתבצעת בסדר לקסיקוגרפי, כאשר הבדיקות מתבצעות על הצלעות הקיימות, ולא סתם באויר – למשל, הבדיקות עבור הגרף המתואר פה, יתחיל בקדקוד {1}, ויבדוק עבור כל הקדקודים האחרים לאן הוא מתחבר, עד שיגיע לחיבור עם קדקוד {3}, ברגע זה הוא יאחד בין הקבוצות וייצור את הקבוצה {1,3}, וימשיך בבדיקה עד שיגיע ויאחד את הקבוצה הגדולה {1,3,7}. כאשר האיחוד הסופי עם קדקוד {9}, יתבצע רק בקריאה לפונקציה של קדקוד {7} הבודק את האיברים הסמוכים לו, ורק אז הקבוצה תגיע לצורתה הסופית והנכונה של {1,3,7,9}, הפונקציה ממשיכה ורצה על כל הקדקודים עד שהיא מסיימת את כל הגרף, כאשר לאחר שהגרף מאוחד כבר ויש נציג לכל קבוצה, אנחנו יכולים לקבוע בוודאות בעזרת הפונקציה SameComponent האם מדובר באיברים מאותה הקבוצה או לא:

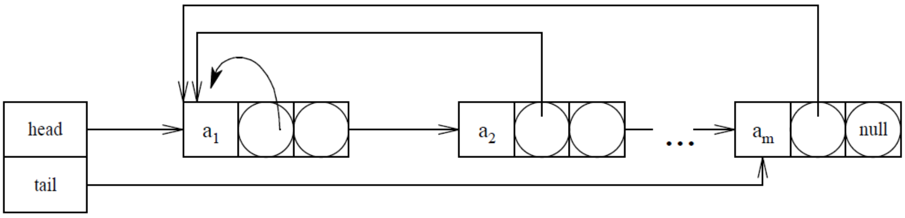
```
ConnectedComponents(V,E,|V|,|E|)
FOR i:=1 TO |V| DO
   MakeSet (i)
ENDFOR
FOR i:=1 TO |V|-1 DO
  FOR j:=i+1 TO |V| DO
    IF {i,j} ∈E THEN
                  x:=FindSet(i); y:=FindSet(j)
                  IF x≠y THEN Union(x,y)
ENDFOR
```

```
SameComponent(i,j)
IF FindSet(i)= FindSet(j) THEN RETURN T
ELSE RETURN F
```

פעולה	תוצאה
הפרדה לקבוצות	{10}{9}{8}{7}{6}{5}{4}{3}{2}{1}
{1,3}	{10}{9}{8}{7}{6}{5}{4}{2}{1,3}
{1,7}	{10}{9}{8}{6}{5}{4}{2}{1,3,7}
{2,8}	{10}{9}{6}{5}{4}{2,8}{1,3,7}
{3,7}	אין שינוי
{4,5}	{10}{9}{6}{4,5}{2,8}{1,3,7}
{7,9}	{10}{6}{4,5}{2,8}{1,3,7,9}
{8,10}	{6}{4,5}{2,8,10}{1,3,7,9}

מימוש על ידי רשימה מקושרת

באשר ממשים ברשימה מקושרת הנציג יהיה ראש הרשימה, כאשר מבנה כל חוליה בתוך הרשימה יהיה: האיבר עצמו – האובייקט אותו אנחנו מחלקים לקבוצות זרות, שכמובן יכול להכיל תתי נתונים), מצביע לאיבר הבא, מצביע לנציג. בצורה הבאה[4]:



[4] מתוך סיכומי השיעורים של ד"ר חייט

בנוסף, יהיה בכל רשימה איבר שמצביע לראש הרשימה ולזנבה, שבמידת הצורך שניהם יעודכנו לפי המצב הקיים.

המצביע לזנב יעזור לנו בהמשך, על מנת להוסיף איבר בודד לרשימה בזמן ריצה של O(1).

הפונקציות יתממשו ברשימה בצורה הבאה:

Make set – יצירת רשימה חדשה, כאשר האיבר המוכנס שהוא היחיד ברשימה יהפוך להיות הנציג של אותה רשימה. זמן ריצה – O(1)

Find-set – מציאת הנציג על ידי בדיקה של המצביע לראש הרשימה. כמובן שגם פה מדובר על זמן ריצה של O(1).

Union – איחוד הרשימות מתבצע בשני שלבים:

- שרשור רשימה אחת אחרי השניה – כאשר ראש רשימה נכנס לאחר זנב הרשימה שתהיה ראשונה.
- לאחר השרשור, יש לעבור על כל האיברים של השימה המתווספת ולעדכן את המצביע של איבר הראש, כך שיצביע לנציג של הקבוצה הקודמת.

מאחר שאנחנו צריכים לעבור על כל איבר בנפרד ברשימה ולשנות את המצביע של כל אחד שיצביע לעבר אחד אחר, אנחנו עלולים להגיע לזמן ריצה ארוך מהצפוי ($\Theta(n^2)$). מאחר ואם ניקח רשימה קצרה ונוסיף לה בכל פעם רשימה ארוכה יותר ממה שקיים בה אותו רגע, אנחנו רצים כל פעם על מספר איברים גדול מאוד.

על מנת לשפר את הביצוע הזה, לפחות במקרה הממוצע (באוסף של פעולות), משתמשים בהיוריסטיקה[5]. הרעיון הכללי לפיו נלך הוא להוסיף את הרשימה הקצרה יותר בסוף הרשימה הארוכה. כאשר במקרה הגרוע ביותר בו שתי הקבוצות שוות, זמן הריצה במעבר על הרשימה החדשה הוא n/2.

אם נמשיך וסכום את כלל האיחודים נגיע לזמן ריצה של :[6] – O(m+nlog(n))

הוכחת זמן הריצה של איחוד היוריסטי:

- ראשית – נתון לנו שזמן ריצה של find-set ו-make-set הוא O(1).
- יהי x איבר כלשהו ברשימה (יש לזכור שMake-set יוצר לנו רשימה עם איבר בודד)
  נחשב את מספר הפעמים בהם עודכן המצביע שלו:

מס' עדכון של x	אורך הרשימה ≤
1	2
2	4
3	8
.....	
$\lceil \lg(k) \rceil$	k

(בכל פעם שאנחנו נאלצים לעדכן את המצביע, ברור לנו שהרשימה גדלה פי 2 מהפעם הקודמת, ולכן אנחנו מדברים על פונקציה לוגריתמית בבסיס 2)

[5] "**היוריסטיקה (Heuristic)** היא כלל חשיבה פשוט, מעין כלל אצבע המבוסס על הגיון פשוט או אינטואיציה, המציע דרך קלה ומהירה לקבלת החלטות ופתרון בעיות, ללא התעמקות ובמחיר דיוק נמוך." (ויקיפדיה)

[6] סימן החיבור אומר פה שמתייחסים לנתון הגדול מבין השניים, כאשר תמיד m>= nlogn.

ובאופן כללי נוכל לומר שבל איבר ברשימה עודכן מקסימום $\lfloor \lg(n) \rfloor$ פעמים.

- סה"כ הזמן של הוספה ועדכון של n איברים הוא o(nlg(n))
- נגדיר את m להיות איחוד של שלושת הפעולות של מציאת הסט והוספה.
- סה"כ הזמן של כל הפעולות הוא O(m+nlog(n))

כאשר אם m=Θ(n) אז זמן הריצה של פעולות הm יהיה כלול בזמן הריצה, והזמן הכולל יהיה O(nlg(n))

מימוש ב"יער של עצים"

הגדרת עץ[7]: עץ הוא גרף מכוון (כל צלע בין שני קדקודים גם מוגדר עם חץ שהוא חד כיווני, ומתאר מי מיוחס למי – במקרה זה אנחנו מדברים על כך שכל הקדקודים מצביעים לכיוון האבא/השורש), קשיר (כל הקדקודים המיוחסים לעץ מחוברים ביניהם באופן כלשהו), וללא מעגלים.

הגדרת יער: אוסף של עצים.

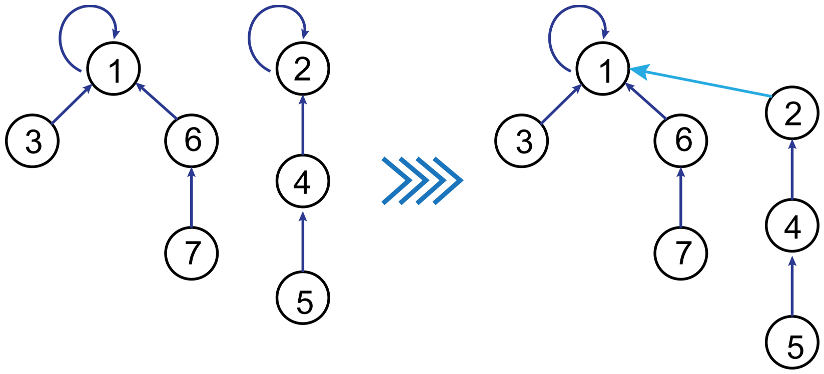
הנציג יהיה תמיד האיבר בשורש העץ, כאשר בניגוד לרשימה מקושרת, אין לכל איבר מצביע ישיר לנציג, אלא בשביך לגלות מיהו הנציג עולים בשרשור בכל פעם לכיוון האב, עד שמגיעים לשורש בו יש לולאה בה הוא מצביע לעצמו.

מימוש הפונקציות ביער של עצים:

Make-Set(x) – יצירת עץ עם איבר-קדקוד המצביע לעצמו. זמן ריצה: O(1).

find-Set(x) – אם מקבלים איבר שרואים שהוא לא הנציג, עוברים על הצלע המקשרת אותו לכיוון האבא וממשיכים על המצביעים, עד שמגיעים לאיבר שמצביע לעצמו (בשונה מעץ רגיל, עליו למדנו במבנ"ת א' המצביע הוא לא לעבר הבן, אלא לכיוון האבא). אם ניקח את האיחוד של כל הגרפים הצורה פשוטה וללא איחודים, אז זמן הריצה של החיפוש נציג יהיה O(n)

Union(x , y) – קבלת שני העצים, ואחד מהנציגים של שני הקבוצות, יצביע על הנציג של הקבוצה השניה, בה הנציג יישאר בתפקידו (כמובן שהעצים לא חייבים להיות בינאריים).



זמן הריצה של פעולת האיחוד לבדה יהיה O(1) – שינוי המצביע בשורש מהצבעה לעצמו להצבעה לשורש לידו. אך כאשר אנחנו מקבלים שתי קבוצות לאיחוד, איננו מקבלים את שני השורשים אלא איבר בודד מכל קבוצה, ויש לחפש את הנציג של אותה קבוצה לפני האיחוד עצמו. בעיה שעלולה לצוץ, היא במקרה שהעץ יהיה מנוון, זמן הריצה יהיה ארוך יחסית, ועלול להגיע עד ל O(m+n)כמו שכבר ציינו

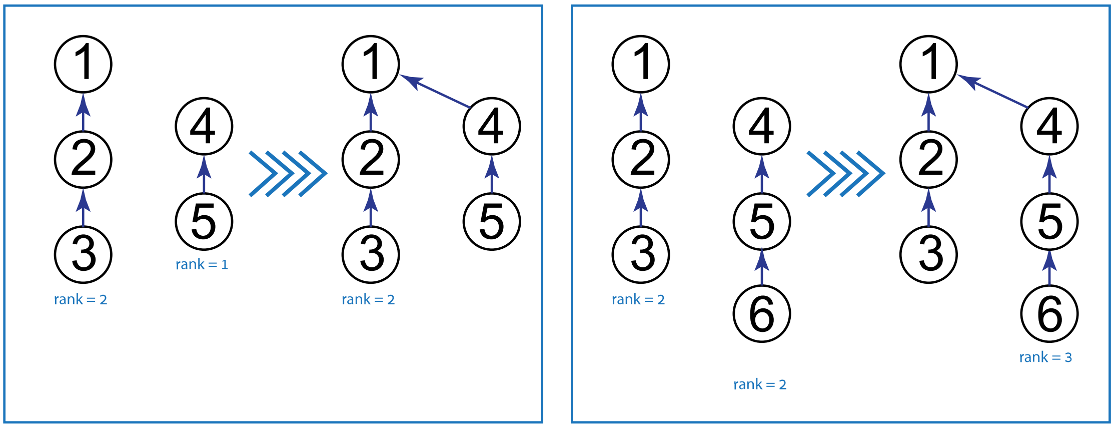
[7] בהמשך הקורס יופיעו עוד הגדרות לסוגי עצים שונים, וניתן שהם יהיו לא מכוונים, אך כאן מתייחסים למקרה ספציפי הזה.

מקודם. על מנת למנוע מצב של זמן ריצה ארוך לאחר איחודים, נקבעו שתי היוריסטיקות המסייעות בפעולה:

<u>איחוד על פי דרגה</u>[8] – האלגוריתם בודק אם הדרגה בין שני העצים שווה או שונה. במידה והדרגה של שני העצים שונה, העץ עם הדרגה הגבוהה "בולע" את הדרגה הנמוכה יותר, כך שגם לאחר הוספת העץ החדש הדרגה של העץ לא תשתנה. במידה ודרגת העצים שווה, בוחרים את אחד מהעצים שיצביע על הנציג של העץ השני והדרגה החדשה גדלה ב1 בלבד (למשל: כאשר מאחדים שני עצים המכילים כל אחד רק איבר בודד, העץ החדש יהיה בגובה/דרגה של 2).

בסופו של דבר, איחוד של כל העצים תחת איחוד על פי דרגה יוציא לנו עץ שגובהו המקסימלי הוא log(n), וזה יהיה גם זמן החיפוש של הנציג בעץ.

איחוד עצים בעלי דרגה שונה איחוד עצים בעלי דרגה שווה



<u>כיווץ המסלולים</u>[9] – איחוד על ידי כיווץ מסלולים למעשה שולח את שני העצים (או האיברים שנשלחו מתוך העץ), מבצע על שניהם את פעולת כיווץ המסלולים ואז מאחד אותם לעץ בודד.

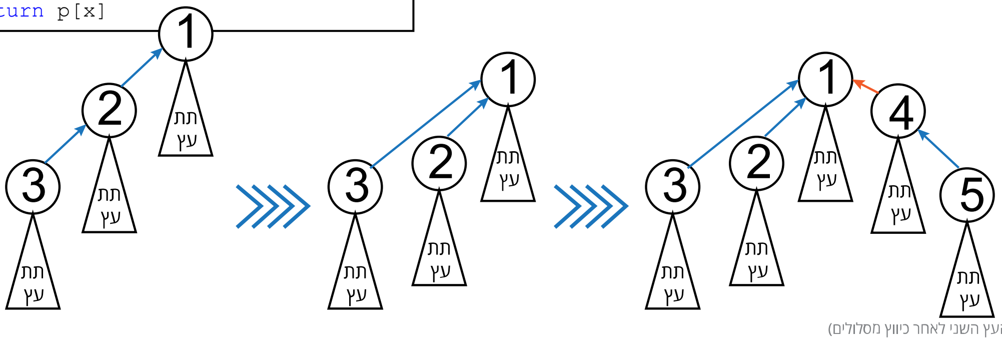
הריצה של כיווץ המסלולים עובדת בזמן הFindSet שמחפשת את הנציג של העץ, כאשר בזמן הריצה הוא לא רק עובר על המסלולים, אלא גם שומר את המסלול של כל הבנים, ומעביר את כל מצביעי האב (הנציגים של תתי העצים) להצביע ישירות על הנציג של העץ אליו מאחדים את המסלול.

```
MAKE-SET(x)
   p[x] ← x
   rank[x] ← 0

UNION(x, y)
  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
   if rank[x] > rank[y]
      then p[y] ← x
      else p[x] ← y
           if rank[x] = rank[y]
         then rank[y] ← rank[y] + 1

FIND-SET(x)
   if x ≠ p[x]
      then p[x] ← FIND-SET(p[x])
   return p[x]
```



---
[8] Union by Rank
[9] Path compression

האלגוריתם עובד בצורה שמפריד קודם כל את איברי העץ בעזרת MakeSet, והאיחוד union מפעיל את הפונקציה שמקשרת את הלינק של האב כך שבמקום להצביע אל ההורה שלו, יצביע לנציג של העץ, על ידי בדיקה של דרגות העץ איזו מהדרגות נמוכה יותר, ושרשור האיבר כך שיהיה בן של הנציגה, וכך הפונקציה עוברת בצורה רקורסיבית על כל האיברים, עד שבסוף ההרצה, כל הנציגים הישנים יצביעו לכיוון הנציג החדש.

הבעיה הנוצרת עם כיווץ מסלולים, היא שלא ברור לנו באמת מה הדרגה והגובה הסופי של העץ לאחר כל הכיווצים והאיחוד. כאשר כתוצאה מהכיווצים הרבים שעולים לקרות אחד בתוך השני (יש לזכור שמדובר בפונקציה רקורסיבית) דרגת העץ יכולה להשתנות בצורה משמעותית מהדרגה המקורית ולנוע בין 1 לlog(n). דבר זה לא משתנה גם כאשר ידוע לנו כמה איברים יש בעץ, ומה הדרגה המקורית.

<u>הוכחת זמן ריצה לכיווץ מסלולים</u>: גם FindSet וגם Union עובדים באותו זמן ריצה, מאחר והאיחוד מתבצע על ידי קריאה כפולה לFindSet, ובחישוב הסיבוכיות בהתעלמות מקבועים מדובר על אותו זמן ריצה. במקרה הגרוע של זמן ריצה האיחוד מתבצע בכל פעם על קבוצות עם גובה שווה (logN) אך אנו לא מתעניינים בזמן של פעולה אחת ויחידה, גם כאשר היא משפרת במשהו את זמן הריצה של המשך הסדרה, ואנו מסתבלים על הזמן הכולל. כך שמינימום פעולות שנבצע יוגדר בm, כך שn=<m, ובסך הכל O(mLog*n)

משפט: במימוש באמצעות אוסף עצים של n איברים, סיבוכיות זמן בצוע m פעולות כיווץ איברים (union/find/makeSet) הוא O(mα(n)), כאשר הפונקציה α(n) מוגדרת להיות הפונקציה ההפוכה לפונקציית אקרמן, אסביר על מה מדובר בהמשך, אך ברגע עלינו לדעת שמדובר על פונקציה שהעליה שלה כל כך מונוטונית ואיטית, עד שהיא מוגדרת להיות כ"במעט ליניארית"

לפיכך הזמן המשוערך (time amortized) לכל פעולה יחידנית חסום ע"י $O(\log^* n)$, כאשר הפונקציה $\log^* n$ מוגדרת בדלהלן:

$$\log^{(i)}(n) = \underbrace{\log_2\log_2 \ldots \log_2}_{i} n$$

$$\log^*(n) = \min\{i \ge 0 \mid \log^{(i)}(n) \le 1$$

ובמילים: log*n הוא מספר הפעמים שצריך לקחת log כדי לקבל מספר קטן או שווה לאחד .זוהי פונקציה מונוטונית העולה בקצב מאד איטי. לכל מספר n פרקטי $\text{Log}^* n \geq 5$, כלומר פעולת find לוקחת זמן משוערך שהוא קבוע מבחינה מעשית (מאחר וגם המספרים הגבוהים שיוצאים הם עדיין מספרים קבועים), כאשר:

$$\text{Log}^*(1) = 0$$

$$\text{Log}^*(2) = 1$$

$$\text{Log}^*(2^2) = 2$$

$$\text{Log}^*(2^{2^2}) = 3$$

$$\text{Log}^*(2^{2^{2^2}}) = 4$$

$$\text{Log}^*(2^{2^{2^{2^2}}}) = \log^*(2^{65,536}) = 5$$

סיבוכיות הפעולות

סיכום סיבוכיות זמן הריצה עבור מימושים שונים:

הפעולה	גרפים	רשימה מקושרת	יער של עצים
Make-Set	O(1)	O(1)	O(1)
Union (פעולה בודדת)	O(1)	O(logN)	O(1)
Union (איחוד הקבוצה)	תלוי במימוש	O(NlogN)	O(NlogN)
Find-set	O(log(n))	O(1)	O(logN)

פונקציית אקרמן[10]

פונקציית אקרמן היא פונקציה רקורסיבית המקבלת שני ערכים שלמים k,j, כאשר k≥0, j≥1.
ומוגדרת בצורה הבאה:

$$A_k(j) = \begin{cases} j+1, & k=0 \\ \underbrace{A_{k-1}(A_{k-1}\ldots(A_{k-1}(j)\ldots)}_{(j+1)\text{ times}}, & k\geq 1 \end{cases}$$

באופן כללי, כאשר k מקבל ערך חיובי ממש (k≥1) המשתנה נקובע את מספר האיטרציות שהפונקציה מבצעת. כאשר הבסיס הוא לבצע בכל פעם את j+1.

המיוחד בפונקציה הזאת, הוא שתוך מספר איטרציות מועט ביותר, התוצאה המתקבלת היא גבוהה וגדלה בצורה מאוד משמעותית. נחשב את האיטרציות הראשונות עבור j=2.

באיטרציה הלפני-ראשונה, אנחנו פשוט מקבלים את ערך הj, ובמקרה שלנו – 2.

לאחר מכן, אנחנו מכפילים את ה-J, ומקבלים 4, ובפעם השנייה עושים זאת עוד j פעמים ומקבלים 8.

בפעם השלישית למעשה אנחנו מבצעים $A_3(2)=A_2(A_2(2))$. אמרנו כבר קודם ש $A_2(2)=8$, ועכשיו אנחנו לאחר החישוב הזה עושים שוב A_2, אך הפעם לערך שהתקבל כבר , כך ש-

$A_3(2) = A_2(A_2(3))=A_2(8) = 2^3*2^8 = 2^{11}=2048$.

באופן כללי כל איטרציה מבצעת $j*2^j$ ובכל שיותר מתקדמים הקפיצה היא משמעותי מ-8 ל-2048. עליה זאת היא קפיצה הרבה יותר משמעותי מכל גידול מעריכי שראינו עד עכשיו, ואנחנו רק בדרגה השלישית שלו.

בהמשך אנחנו יכולים לראות שאנחנו ממשיכים להעמיס מגדל של חזקות על גבי ה-2 שמשמש לנו בתור בסיס, כך שמוסיפים למספר בכל פעם חזקה בגודל מה שהתקבל באיטרציה הקודמת. אם ננסה לחשב מספרית את $A_4(2) = 2^{2048}$, נגיע מספר שהוא מוגדר הרבה יותר גדול מ10^{600}, מה

[10] סרטון מעולה שמסביר את הפונקציה (באנגלית) של אוניברסיטת סטנפורד נמצא בלינק הבא – https://goo.gl/pCbuZy.
מומלץ לצפות בו בשביל להבין את זה יותר לעומק. זה לא חושב לדעתי ברמת המבחן, אבל זה מעניין עד כמה שאפשר.

שאפילו בלי לנסות ולחשב או לכתוב את המספר הזה ניתן לקבוע די בוודאות שאמנם לא מדובר על אינסוף, אבל בהחלט מדובר על מספר שיכסה את כל האפשרויות שנצטרך.

הפונקציה ההפוכה לאקרמן מתייחסת באופן קבוע ל j=2, ומוגדרת באופן הבא:

$$\alpha(m,n) = min\{i \geq 1 : A(i, \lfloor m/n \rfloor) > log\, n\}$$

אנחנו מחפשים את הרמה הנמוכה ביותר עבורה $A_k(1) \geq n$. מהחישוב של פונקציית אקרמן, ניתן להבים כי עבור כל מספר ממשי שנקבל $n \geq 4$.

עכשיו גם אם נסתבל על קצב הגידול של $\log^*$ שהוסבר קודם, ניתן לראות שעד כמה שהפונקציה הזאת יחסית מונוטוני, היא עדיין חסומה ביחס לפונקציית אקרמן, וזמן הפעולות עדיין נחשב ליניארי.

<u>דוגמה לפונקציית אקרמן</u>

הדוגמה הבאה לקוחה מתוך מבחן במבנה נתונים של שנת תשע"ז סמס' א, מועד א'.

כדאי לעבור על זה גם מאחר וזה שאלה רלוונטית מהמבחן, ולא פחות חשוב בשביל להבין את סדר הפעולות על הפונקציה ומה מצופה מאיתנו לעשות.

פונקציית אקרמן מוגדרת כך:

$j \geq 1$ $A(1, j) = 2^j$

$i \geq 2$ $A(i, 1) = A(i-1,2)$

$i, j \geq 2$ $A(i,j) = A(i-1, A(i, j-1))$

$A(2, 1) = A(1, 2) = 2^2 = 4$

מהו הערך של $A(2, 3)$?

הראו את אופן החישוב

אנחנו מקבלים את השאלה מחולקת לשלושה חלקים:

1. הצורה הכללית של פונקציית אקרמן.
2. התוצאה של שתיים מהאפשרויות (שלמעשה לא מדובר בתוצאות מיוחדות עבור השאלה, אלא "חוסכים" לנו זמן חישוב מיותר)
3. הערך אותו אנחנו נדרשים לקבל.

על מנת לפתור את השאלה נצטרך להתחיל להציב עד שנגיע לקצה הרקורסיה:

ראשית מאחר ושני הערכים הדרושים הם גדולים/שווים ל-2, נציב בנוסחה השלישית-

$A(2,3) = A(2\text{-}1, A(2, 3\text{-}1)) = A(1, A(2,2))$

נוכל לראות בעת, שהפונקציה הפנימית עדיין גדולה/שווה ל-2 בשני חלקיה, ולכן נציב בה שוב -

$A(1, A(2,2)) = A(1, A(2-1, A(2, 2-1))) = A(1, A(1, A(2, 1)))$

למעשה, יש לנו עדיין נוסחה בפנים, אך את התשובה קיבלנו כבר בנתון – $A(2,1) = 2^2 = 4$. ועכשיו אפשר להתחיל לעבוד עם התוצאה הזאת.

$A(1, A(1, 4))$

עכשיו את הנוסחה הפנימית אנחנו יכולים לפתור בעזרת הנוסחה הראשונה –

$A(1, A(1, 4)) = A(1, 2^4)$

ושוב נשתמש בנוסחה הראשונה על מנת לפתור לגמרי את השאלה –

$A(1, 2^4) = A(1, 16) = 2^{16}$

סיימנו. הרווחנו 5 נקודות.

קוד האפמן

קוד האפמן הוא שיטה לקידוד סימנים בגון תווי טקסט ללא אובדן נתונים ובאופטימיזציה מרבית. הרעיון הכללי של השיטה מתבסס על הקצאת אורך קוד משתנה לכל תו, על פי השכיחות של התו ברצף הטקסט המוצע, כך שככל שהסימן יותר נפוץ, כך הוא יוצג בעזרת פחות תווים. שימוש בשיטה זאת חוסך בין 20% ל90% בכמות המידע[11].

קוד האפמן שייך לקבוצת אלגוריתמיים המכונה "אלגוריתמים חמדניים[12]", הנותנים על פי הערכה היוריסטית את הדרך המהירה והפשוטה ביותר לפתירת בעיות שונות. במקרה זה, כאמור, מתבססים על שכיחות התווים.

בתחילה מונים את השכיחות של כל תו, ולאחר מכן בונים קוד יעיל עם קידוד מסוים לכל תו על פי השכיחות שלו, ובעזרת זה כמות הטקסט קטנה משמעותית, סדר הבנייה יופיע בהמשך בצורה מפורטת.

לדוג', אם אנחנו מדברים על טקסט שמורכב רק משישה תווים שונים המרכיבים טקסט של 100,000 תווים בשכיחויות שונות כמצוין, הקוד יופיע בצורה הבאה:

	F	E	D	C	B	A
שכיחות (באלפים)	5	9	16	12	13	45
קידוד לא דחוס	101	100	011	010	001	000
קידוד האפמן דחוס	1100	1101	111	100	101	0

במקרה הרגיל, על מנת לייצג 6 תווים שונים אנחנו צריכים 3 ביטים (בינאריים), שיחולקו לפי המיקום שלהם והסדר הכרונולוגי. אך לעומת זאת, בעזרת קידוד האפמן, נוכל לתת לאיבר השכיח ביותר (A) ערך של תו בודד, ולעלות בכמות התווים בכל פעם לפי השכיחות, עד שהאיברים בעלי השכיחות הנמוכה ביותר יקבלו ערך של 4 תווים, האפמן הוכיח שגם במקרה בזה כמות התווים הכוללת של הקוד תהיה קטנה יותר בצורה משמעותית (224 לעומת 300 בקידוד רגיל – חיסכון של 24%, וזוהי רק דוגמא פשוטה).

אחת התכונות שעוזרת לקוד האפמן להיות יעיל, הוא השיוך שלו לקבוצת הקודים הנקראת "קודים תחיליים[13]", צורת קוד בה חלוקת התווים לסימנים נעשית בצורה בה אין שני אותיות שהתווים שלהם יתחילו באותו אופן, מה שמסייע לפיענון מהיר, מאחר והאפשרות לקריאה ופענוח בנוי בצורה חח"ע. אם ניקח את הקידוד הלא דחוס בדוגמא בטבלה, ונתחיל לקרוא קוד שמתחיל בספרה 0, יהיו לנו 4 אופציות בחירה, וכשנעבור לתו הבא (0/1) האפשרויות יצטמצמו לנו ל-2, ורק בתו השלישי נוכל לדעת בוודאות על איזה תו מדובר. יותר מזה, אם נקבל קוד שלא ידוע לנו מה אורך המילה, לא נוכל לדעת בוודאות אם אנחנו קוראים אותו בצורה נכונה – 001011 יכול להיקרא שונה לגמרי אם נחלק את הקוד לשני מילים של 3 תוים (001 011) או לשלושה מילים של שני תוים (00 10 11). אך לעומת זאת בקוד האפמן, הקוד: 001011101 יכול להיקרא אך ורק בצורה של aabd, ולא בשום אופן אחר.

כאשר מתארים עץ המייצג קידוד (כמובן בצורה בינארית), מייצגים את ה-0 כפנייה שמאלה, ואת ה-1 כפנייה ימינה. בעלים מציינים את התו אותו קידדנו ובצמוד אליו, ומופרד עם נקודתיים (:) מציינים את

---

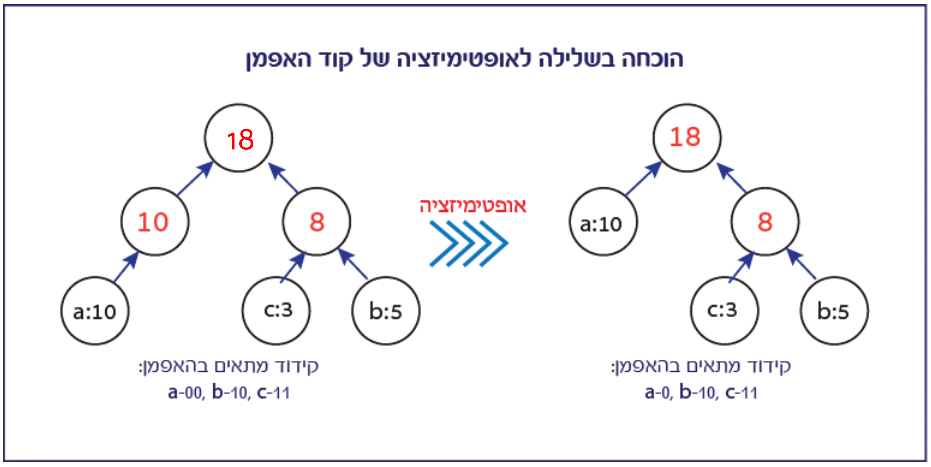
[11] ברמה האופטימלית, אך מעשית רק עד 68%.
[12] "אלגוריתם חמדן (Greedy Algorithm) :הוא אלגוריתם המתבסס על היוריסטיקה לפיה בוחרים את האפשרות הטובה ביותר הנראית לעין בשלב הנוכחי, מבלי לקחת בחשבון את ההשפעה של צעד זה על המשך הפתרון. אלגוריתמים חמדנים נפוצים בפתרון בעיות מיטוב, בהן מנסים למצוא את הפתרון הטוב ביותר." (ויקיפדיה)
[13] Prefix Code

השכיחות של התו בטקסט (לדוגמא – A:45). בנוסף בצמתים שאינם עלים כותבים את השכיחות המצטברת של תתי העצים שמתחת לאותו קדקוד (גם מהימין וגם מהשמאל). כך שהבחירה איזה תו יהיה מקודד באותו רגע הוא פשוט גלישה ימינה ושמאלה לכיוון השכיחות שלו ובחירה בתו לפי הכיוון.

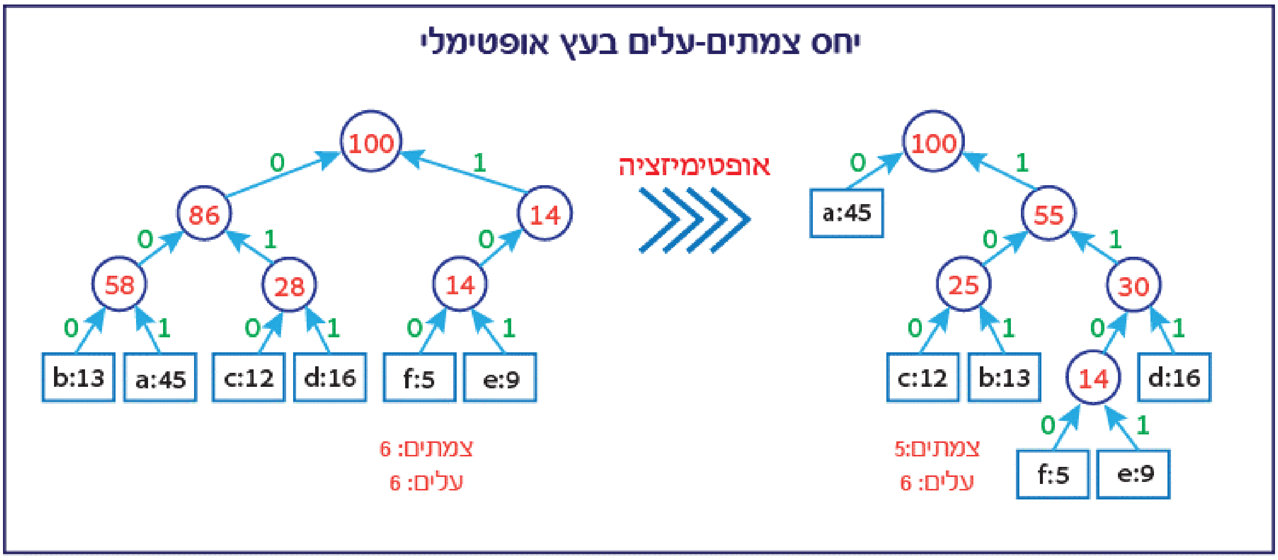
תבונה נוספת לקוד זה, עץ המייצג קידוד של תווים בצורה אופטימלית בדומה לקוד האפמן, חייב להיות עץ מלא – לכל צומת **חייב** להיות שני בנים(אלא אם מדובר בעלה).

<u>הוכחה שעץ האפמן חייב להיות מלא</u>

הוכחה בשלילה - נניח שקיים עץ T המייצג קוד תחיליות אופטימלי שאינו מלא.
הווה אומר - יש קדקוד אחד המכיל רק בן יחיד. אזי אנחנו יכולים לייעל את הקוד, בצורה שנחליף את הקדקוד שיש לו בן אחד בלבד בבן (היחיד) שלו, וכך אנחנו מייעלים את הקוד על ידי קיצור הקידוד של תת העץ בביט אחד.



תבונה נוספת לקוד האפמן, שלמעשה נגזרת מהוכחת האופטימליות – מספר הצמתים בעץ יהיה נמוך ב-1 ממספר העלים המייצגים את התווים (שמתייחסים אליהם ב- |C| ומספר הצמתים יהיה |C|-1[14]) לדוג' המתאימה לטבלה למעלה:



[14] למעשה, הזמן שלוקח לבנות את העץ משלב תור הקדימויות הוא |C|-1, כאשר בכל פעם לוקחים שני איברים מהתור ומחזירים אחד.

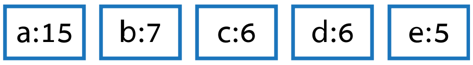
העץ השמאלי מייצג את העץ הלא יעיל והימני את קוד האפמן, כאשר כל "ירידה" בעץ לכיוון האות מוסיף תו 0/1 על פי הכיוון של הירידה.

השלבים לבניית עץ מייצג של קוד האפמן

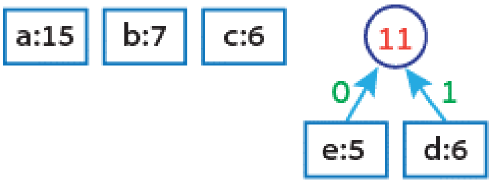
1. יצירת "תור קדימויות" כך שבכל שמספר המופעים של האות קטן יותר, כך הקדימות שלו תהיה יותר גבוהה
2. כל עוד בתור יש יותר מאיבר אחד:
   a. יוצרים צומת חדש.
   b. מוציאים את שני האיברים עם הקדימות הגבוהה ביותר ושמים אותם כבנים של הצומת, כאשר האיבר בעל המופעים הגבוה יותר יהיה מימין, והנמוך יותר משמאל.
   c. בצומת שנוצרה, מכניסים את הערך של שתי השכיחויות של הבנים של הצומת (בדוגמא שלפנינו מוציאים את e:9 ו-f:5 ובצומת מכניסים את הערך "14").
   d. מחזירים את הצומת בעל הערך החדש לתור הקדימויות[15].
3. ברגע שנשאר בתור רק איבר אחד, הוא שורש העץ.
4. קובעים את הקוד עבור כל תו על פי המסלול בין הקשתות – ימין 1 שמאל 0.

בניית עץ האפמן לדוגמא

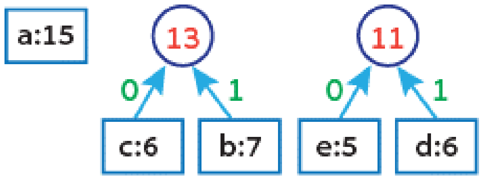
שלב ראשון: מסדרים את כל התווים בתור קדימויות על פי סדר השביחות:



שלב שני: בוחרים את שני המספרים בעלי התדירות הנמוכה ביותר, ומעלים אותם תחת קדקוד יחיד, (כמובן שאם יש לנו כמו במקרה הזה שני קדקודים בעלי אותה תדירות, אין חשיבות לסדר הלקיחה):

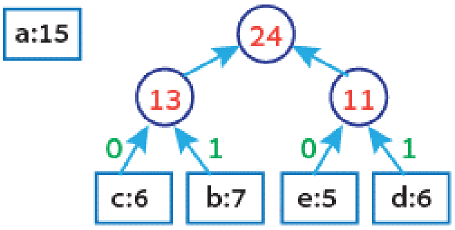


שלב שלישי: ממשיכים הלאה בלקיחת שתי השכיחויות הנמוכות ביותר, כאשר יש לשים לב שבעת המספרים הנמוכים יכולים להיות שניים אחרים לגמרי ולא אלו שכבר התעסקנו איתם בשלב הקודם:

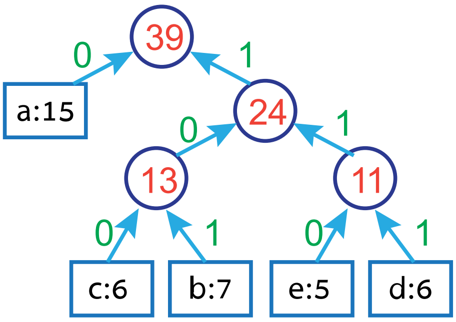


[15] למעשה, המיוחד בעץ הזה, הוא הבניה שלו מלמטה למעלה. קודם כל לוקחים את העלים התחתונים ביותר ולאחר שבונים אותו כתת-עץ מחזירים אותו לתוך התור עד שצריך להשתמש בו שוב.

שלב רביעי: מאחדים קדקודים בעלי תדירות נמוכה:



שלב חמישי ואחרון: סוגרים את הקדקוד עם העלה שנותר, ויש לנו עץ האפמן לתפארת!



<u>פסאודו-קוד לבניית עץ בקוד האפמן</u>:

מציבים ב-n את מספר האיברים אותו אנחנו הולכים למיין, וב-Q (תור הקדימויות) את כל התוים, כמובן על פי סדר הקדימויות (שורה 1-2).

עוברים בלולאה כמספר התוים פחות 1:

-בונים קדקוד חדש.

- מכניסים את האיבר הקטן ביותר קודם לשמאל, ואחר כך את הקטן ביותר (הנובחי) לימין.

```
Huffman(C)

n ← |C|;
Q ←C;
for i ← 1 to n – 1 do
      allocate a new node z
            z.left ← x ← Extract-Min(Q);
            z.right ← y ← Extract-Min(Q);
            z.freq ← x.freq + y.freq;
            Insert(Q, z);
end for
return Extract-Min(Q); {return the root of the tree}
```

- מחברים את שניהם לתוך הקדקוד שיצרנו, ומכניסים את הקדקוד עצמו לתוך התור.

לאחר המעבר כל התוים נוציא את האיבר היחיד שנשאר בתור, והוא יהיה העץ הדרוש

<u>עלות עץ בקוד האפמן</u>

כאשר מדברים או "מודדים" עץ הופמן, אנחנו מתייחסים על מכלול העץ בתור "עלות" או "מחיר" של העץ. מחיר עץ הוא למעשה נגזרת של האופטימליות של העץ - ככל שמחיר העץ נמוך יותר, כך הוא מתאים יותר ונחשב יותר "נכון".

הנוסחה לחישוב עלות היא - $B(T) = \sum_{c \in C} f(c) d_T(c)$,

כאשר f(c) – תדירות התו (frequency)

$D_T(c)$ – עומק התו ביחס לעץ, באשר ככל שהעומק יותר גדול, כך למעשה אורך קידוד התו ארוך יותר.

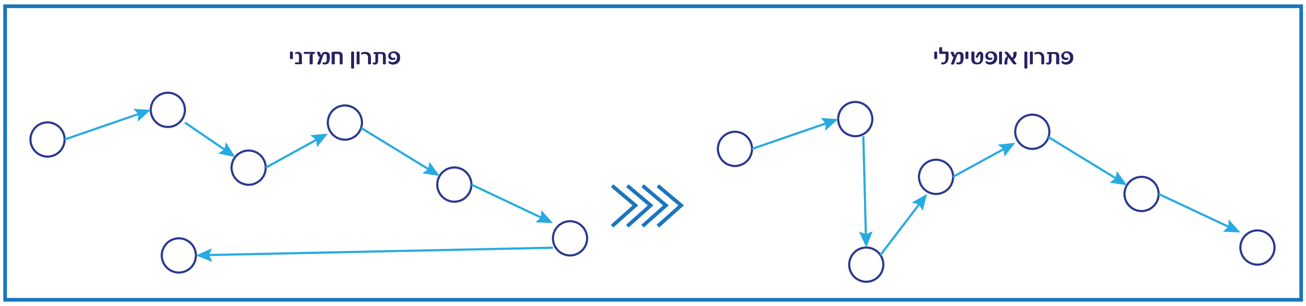
עוברים בסכימה על כל הקדקודים ומכפילים את שכיחות התא בעומק שלו (אורך המסלול), מחברים את התוצאות ומקבלים את המחיר של העץ – B(T).

אם סורקים את העץ בסריקת in-order למעשה ניתן לקבל את התו המצוין ואת המסלול המוביל אליו בצורה מסודרת, אך לא ממוינת! אם רוצים שיהיה ממוין ניתן לעבור עליו שוב לאחר שיצאו התוים.

אלגוריתמים חמדניים

הזכרנו קודם שקוד הופמן הוא אלגוריתם חמדן, ויש צורך טיפה להרחיב עליו. אלגוריתמים חמדניים, הם כינוי למגוון אלגוריתמים שברגע שניתנת להם אפשרות, הם בוחרים באופציה הנראית כטובה ביותר באותו רגע, וללא התחשבות במה שיקרה בהמשך.

למשל: "בעיית הסוכן הנוסע" - סוכן מכירות רוצה לעבור במספר יישובים כדי למכור את הסחורה שלו. המטרה היא למצוא את המסלול הקצר ביותר שיעבור דרך כל היישובים. על פי שיטת האלגוריתם החמדן, הסוכן הנוסע צריך להסתכל בכל פעם במפה ולנסוע ליישוב הקרוב ביותר בו לא ביקר עדיין. לכאורה, שיטה זו נראית טוב, אך יכול להיות שמאחר והוא מתקדם בכל פעם ליישוב הקרוב אליו, ייצא שהוא מפספס יישוב שהיה במקום השני והוא יצטרך לחזור אליו בסוף הסיבוב. הפיתרון הנכון יותר הוא לוודא שאין איזה יישוב שעלול להתפספס בסיבוב, כמתואר באיור:



במקרה של קוד האפמן, האלגוריתם מכונה חמדני, מאחר והפשטות שלו סובבת סביב הוצאה פשוטה בכל פעם של השכיחות הנמוכה ביותר בתור הקדימויות.

הבעיה בשיקול החמדני הוא, שהוא לא באמת תמיד נותן פיתרון אופטימלי ולפעמים הוא מביא דווקא את האופציה הפחות טובה מבין האופציות. למשל, כמו שתיארנו בבעיית הסוכן הנוסע – יכול להיות שתהיה דרך טובה יותר להוציא את המסלול הנכון.

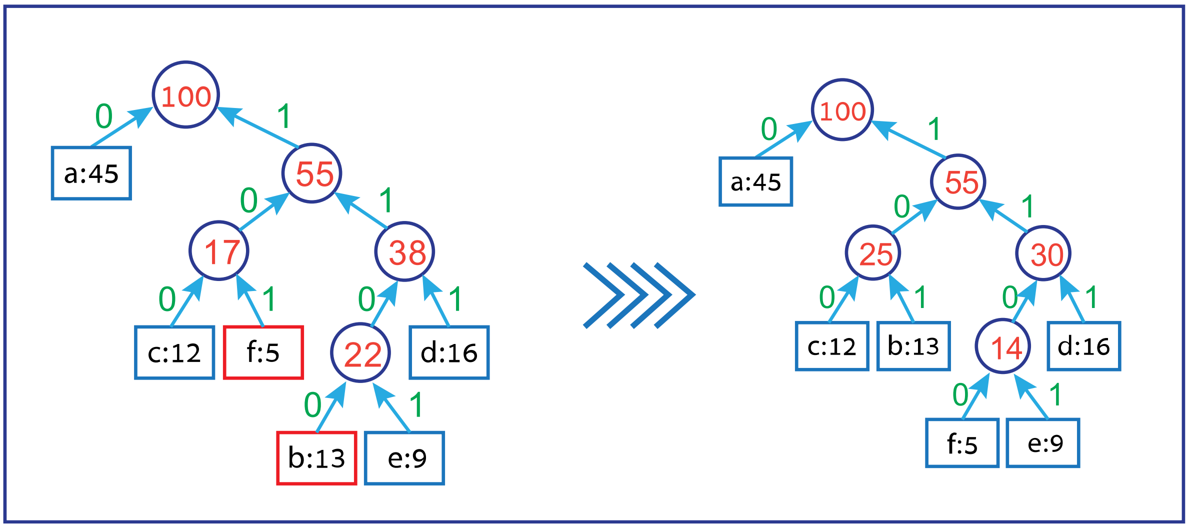
ניתן להוכיח באינדוקציה יעילות של אלגוריתם חמדני בקוד האפמן, על ידי בדיקת אופטימליות של הרמה התחתונה ביותר, ועלייה עם העץ כלפי מעלה עד שמוכיחים שכל העץ החמדני הוא אופטימלי.

כאשר ניקח עץ בעל n איברים ונבחר את השניים בעלי השכיחות הנמוכה ביותר ונחליף אותם באיבר אחר z→(x,y), אז לפי ההנחות יסוד שלנו אנחנו שמרנו על אופטימליות של העץ מאחר ואנחנו לא משנים את העלות הכוללת של העץ, ואם נמשיך ונעלה באינדוקציה נראה שגם כל העץ סמוך על אופטימליות.

תבונות עצים אופטימליים

1. ברמה התחתונה ביותר יש לפחות שני קדקודים -העץ המייצג את קוד האפמן הוא עץ מלא– הוכחנו קודם, שבמידה והעץ לא מלא ניתן לייעל את הקוד ולצמצם את העלה הבודד כלפי מעלה, מבלי לפגוע במבנה העץ, וכך לשפר את האופטימליות בקיצור המסלול.

2. שני העלים בעלי התדירות המינימלית חייבים להימצא ברמה התחתונה ביותר – בדומה להוכחה הקודמת, אם יווצר מצב בו הם אינם ברמה התחתונה ביותר, ניתן לייעל את האופטימליות של העץ על ידי החלפה של העלים למקומות הנכונים שלהם. לדוגמא האות f שמופיעה רק 5 פעמים, והאות b שמופיעה 13 פעמים חייבות להופיע בסידור אחר מהעץ השמאלי, אחרת הערך בעל התדירות היותר גבוהה מבי השניים יקבל גם קוד ארוך יותר בתו אחד בודד, ויוריד משמעותי את יעילות הקוד.

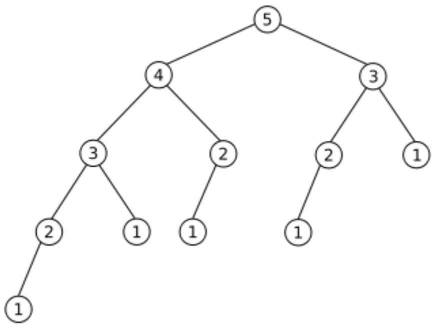


3. שני העלים בעלי התדירות המינימלית יכולים להיות אחים – הוכחנו כבר שהם ברמה התחתונה ביותר, אזי אם הם אינם אחים, ניתן להחליף להם מקומות בלי לשנות את האופטימליות של העץ, כך שהם יהיו אחים.

בתרגולים ובמבחן ניתנו שאלות של בניית עץ, חישוב העלות שלו, בדיקת יחס הייעול ($\frac{\text{קוד לאחר אופטימיזציה}}{\text{קוד מקורי}}$) וכו'. למשל: בניית עץ המכיל שביחוית בסדר פיבונאצ'י וחישוב יחס הדחיסה – 162 ביטים לפני הדחיסה, ו-132 ביטים לאחר הדחיסה כך ש: $0.168 = \frac{132}{162}$ (יחס הזהב)

<u>עצי חיפוש מאוזנים</u>

למדנו בעבר על עצי חיפוש בינאריים, המכילים רק עד שני בנים לכל צומת. על מנת שהעצים הבינאריים יהיו כמה שיותר יעילים עליהם להיות <u>מאוזנים</u> (בדומה לעץ AVL) וכך נוכל לחסום את הפעולות המבוצעות בעץ ב-O(lgn) ואם לא יהיו מאוזנים, עלול להיווצר מצב של עץ מנוון שהיעילות שלו נמוכה מאוד. אך גם בעצי AVL ישנו שוני בין העלים השונים בהפרשים מאוד גבוהים. לדוגמא – עץ פיבונאצ'י, הבנוי בצורה שהיא מאוזנת, אך ההפרשים בין כמות הבנים ברמות השונות הם הפרשים גדולים מאוד.



על מנת להרחיב את האופציות לעצים שיחזיקו יותר מידע קיימים מבני-נתונים של עצי חיפוש מאוזנים שאינם בינאריים, שהתכונה החשובה שבהם היא שהם תמיד שלמים- כל העצים נגמרים בצורה אחידה באותה רמה.

ישנם שני סוגים עיקריים לעצים באלה:

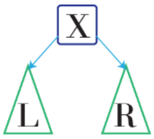
1. עצי 2-3
2. עצי B

המבנה הכללי שלהם וסדר הפעולות דומה בשניהם, אך עצי B מכילים כמות מידע הרבה יותר גדולה ולכן המימוש בו דורש קצת יותר תשומת לב.

עצי 2-3

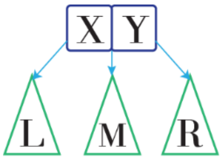
לעצי 2-3 ישנם 3 סוגי קדקודים:

1. **קדקוד NULL** – לא מכילים מפתחות ומהווים רק קדקודים שמציינים את סיום המסלולים מהשורש.
2. **קדקוד 2** – לקדקודים מהסוג הזה יהיו שני בנים כמו שאנו מכירים בעץ בינארי רגיל.



- דרישות עבור קדקוד מסוג 2: כל הערכים מתת-העץ משמאל יהיו קטנים (או שווים) מהקדקוד, וכל הערכים בתת העץ מימין יהיו גדולים (או שווים – בדרך כלל מתייחסים בעצים כאלה לערכים חח"ע, כך שלא יהיו ערכים שווים בתוך העץ)

3. **קדקוד 3** – מכיל 3 בנים: L – שמאל, M- אמצע ו-R – ימין.



- דרישות עבור קדקוד מסוג 3: בקדקוד העליון שומרים שני מפתחות X|Y, כאשר היחס של הערכים בתת-העץ של הבנים לקדקוד עצמו יהיה בדלקמן:
  - תת העץ במפתח השמאלי – קטן יותר מX
  - תת העץ של המפתח האמצעי - בין X ל-Y
  - תת העץ הימני - יהיה גדול יותר מ-Y.

דרישה עבור כל סוגי הקדקודים: אורך המסלול מכל קדקוד עד לכל קדקודNULL בתת-העצים שלו חייב להיות שווה.

גובה של עצי 2-3 בעל n קדקודים הוא לכל היותר log(n+1)-1, בתור הגדרתם בעצים שלמים.

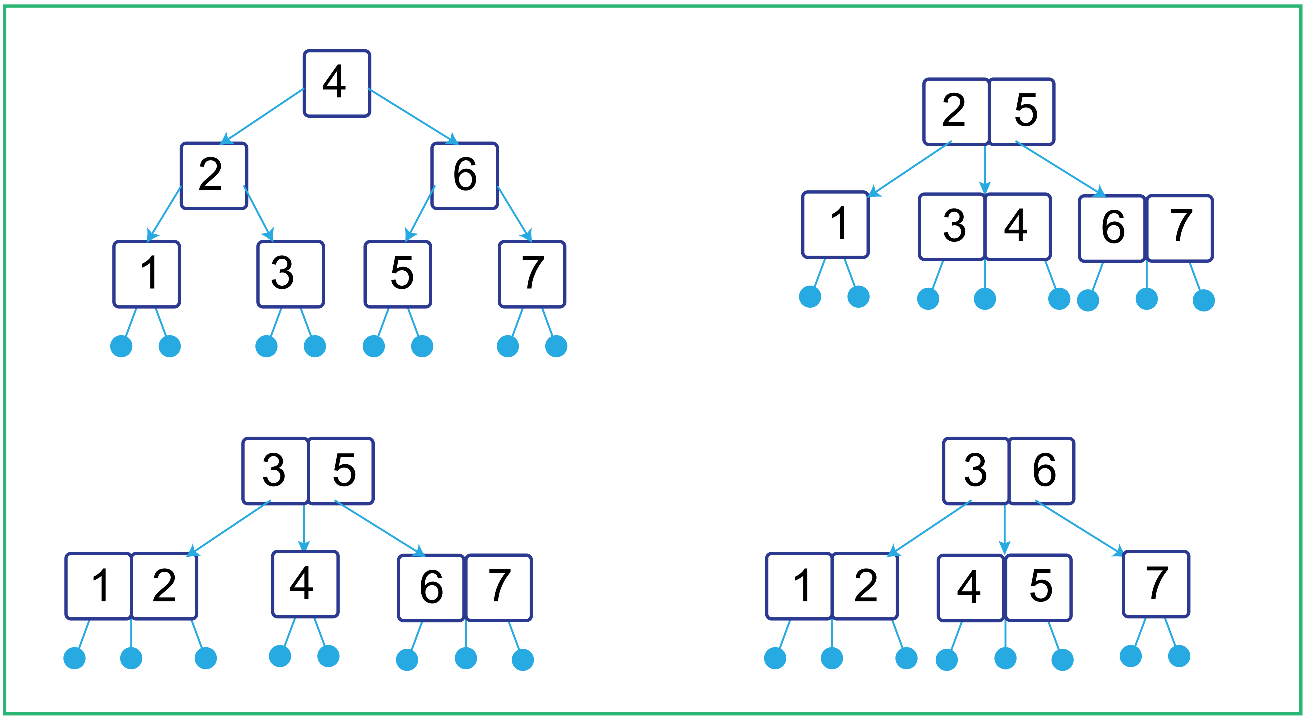
מספר הצמתים המינימלי בעץ - 2^{h-1} – עץ שלם שבכל קדקודיו מסוג 2

מספר צמתים מקסימלי בעץ - $\frac{3^{h-1}}{2}$ – עץ שבכל קדקודיו מסוג 3

מספר המפתחות בעץ- $2^{h-1} \leq n \leq 3h-1$ – טווח המפתחות האפשרי, כאשר המקסימום הוא מספר הצמתים המקסימלי, בו לכל אחד מהצמתים יש שני מפתחות.

כל התנאים המרכיבים את העץ (מאוזן ושלם) גורמים לכך שהפעולות הבסיסיות על העץ (חיפוש, הוספה, מחיקה) יבולות להתבצע בזמן לוגריתמי, התרגילים השונים עוסקים בצורה של ביצוע הפעולות בצורה יעילה שתשמור על תבונות העץ.

למעשה, עקב התבונות של העץ, עבור n איברים יכולים להתקיים מספר עצים שונה שיהיו שווים בנכונותם. למשל, עבור רצף המספרים 1-7, יכולים להתקיים כל העצים הבאים:



קדקוד סופי יוגדר להיות כזה אשר כל בניו הם קדקודי NULL, וברוב המקרים לא נצייר את בני הNULL, ונתייחס לקדקוד עצמו כ**סופי**.

הכנסת איבר לעצי 2-3

תהליך הכנסת האיבר:

1. קודם כל, יש לרדת בעץ ולמצוא את מקום ההכנסה הנכון עבור האיבר החדש.
2. כאשר מגיעים לסיום המסלול- קדקוד סופי – בודקים האם מדובר בקדקוד 2 או 3.
   a. קדקוד 2 – מרחיבים את הקדקוד להיות קדקוד 3, ומכניסים לכל בניו ערכי NULL.
   b. קדקוד 3 – מפצלים את הקדקוד לשניים, כאשר השמאלי יהיה הערך הקטן יותר מבין השלושה (x,y,new) הימני יהיה הגדול יותר והערך האמצעי יהפוך להיות הקדקוד שמעליהם.
3. לאחר השינוי, יכול להיות שהופר האיזון הכלל של העץ, ולכן בודקים את הקדקוד שמעל השינוי האחרון ומתקנים כלפי מעלה על פי סוג הקדקוד.
   a. קדקוד 2 – מקבל את הערך של האמצעי מבין מה שעלה, ועל מנת למור על העץ שלם, הופך את עצמו להיות קדקוד 3, על ידי סידור כל הערכים באופן מתאים על פי הערך של כל איבר.
   b. קדקוד 3 – מאחר ולא ניתן לצרף עוד איברים לאיבר 3, שוב מפצלים את הקדקוד לשניים, וחוזרים להציף את הבעיה עד שהיא מתוקנת לגמרי.

סיום ההכנסה יהיה באחד משני מקרים: מציאת מקום מתאים, או במקרה בו עברנו את שורש העץ. במקרה שבזה, יוצרים קדקוד 2 חדש ומכניסים שם את שורש העץ.

דוגמה לבניית עץ 2-3

ניקח בדוגמה את המילה "אלגוריתם" ונעבור אות-אות, עד שנבנה את העץ.

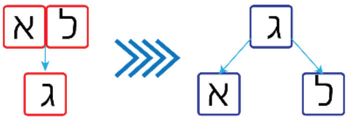
1. שלב ראשון – 'א'

2. שלב שני – 'ל'

האות 'ל' גדולה יותר מהאות 'א', ולכן תהיה מימינו. מאחר והעץ צריך תמיד להישאר שלם, הקדקוד מסוג 2, יהפוך להיות קדקוד מסוג 3.

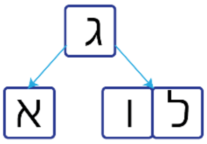
3. שלב שלישי – 'ג'

האות 'ג' נמצאת בין שני האותיות 'א' ו'ל', דבר בעייתי מאחר וקדקוד 3 יכול להכיל אך ורק שלושה בנים, והאיזון פה מופר[16]. במקרה כזה כמו שתיארנו, הערך האמצעי מבין השלושה הופך להיות קדקוד מסוג 2, ושני האיברים האחרים הופכים להיות הבנים שלו.



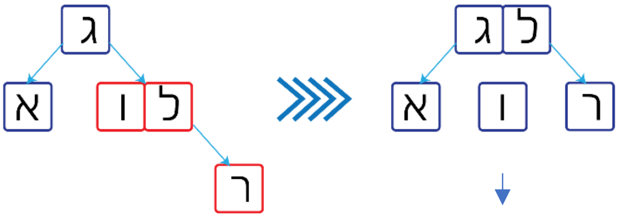
4. שלב רביעי – 'ו'

מבניסים את האות 'ו'. מיקומה הוא בין 'ג' ל'ל'. מאחר ולא ניתן להכניס את האות לשורש (כי זה יהפוך את השורש להיות קדקוד מסוג 3 שלא יכול להכיל 2 בנים, נכניס אותו בצמוד לאות 'ל', כמובן משמאל, ונהפוך את הקדקוד הזה לקדקוד מסוג 3.



5. שלב חמישי – 'ר'

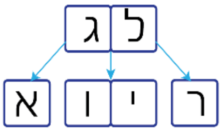
מורידים את האות עד למקום הנכון בו היא אמורה להיות – ברגע מאחר והיא הגדולה ביותר היא נמצאת מימין במקום הכי קיצוני. מאחר ויש באותו מקום כבר קדקוד מסוג 3, ואין אפשרות



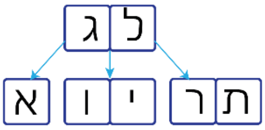
[16] את הפרת האיזון סימנתי במסגרת האדומה סביב הריבוע. כאן ובכל ההדגמות בהמשך.

להכניס אותו באותה רמה, בודקים את הרמות שמעל. שורש העץ קדקוד מסוג 2, ולכן מה שניתן לעשות הוא להעלות את הערך האמצעי מבין השלושה בהם האיזון מופר -ל- ולצרף אותו לקדקוד 3 בשורש, את שאר הבנים שנותרו באותה רמה מכניסים כל אחד במקומו הנכון.

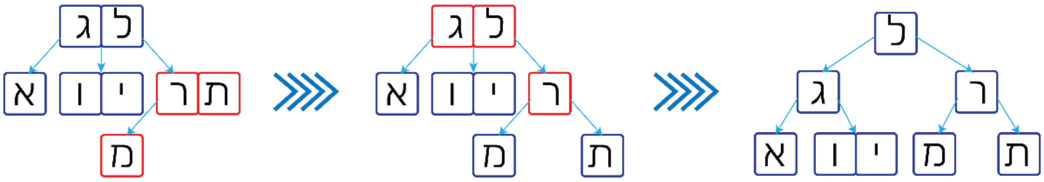
6. שלב שישי – 'י'
השלב הזה פשוט יחסית, 'י' נכנס בין ה'ל' ל'ג', ונצמדת לאות 'ו' והופכת אותו לקדקוד מסוג 3.



7. שלב שביעי – 'ת'
באופן דומה לשלב הקודם, מכניסים את ה'ת' ליד ה'ר'.



8. שלב שמיני – 'ם'[17]
בתחילה, האות מ' יורדת עד למקום הנכון שלה, משמאל לאות 'ר'. כמובן שהאיזון פה מופר, ומאחר שמדובר בקדקוד 3 שלא ניתן להוסיף לו עוד ערכים, צריך לפצל את הקדקוד לשניים, והערך האמצעי עולה להיות השורש של תת-העץ.
בשלב השני, האיזון של העץ עדיין מופר, ואנו בודקים כלפי מעלה. האבא של הערכים הבעייתים (השורש של העץ במקרה הזה) הוא קדקוד מסוג 3, ולכן בדומה לשלב הקודם, אנחנו מפצלים את השורש ומעלים למעלה את 'ל', כאשר 'ג' שהיה החלק השמאלי של השורש, הופך להיות שורש של תת עץ חדש, והאיזון נשמר.



מחיקת איבר מעץ 2-3

מפעילים את החיפוש ויורדים בעץ עד שמוצאים את המפתח אותו צריך למחוק. במידה ואיבר המבוקש אינו ערך סופי (עלה), מחליפים את המיקום שלו עם הערך העוקב או הקודם לו, כך שיהיה עלה, ואחר כך פועלים על פי האלגוריתם הבא:

1. אם מדובר על קדקוד 3, המכיל 2 מפתחות, ניתן פשוט למחוק את הערך הדרוש מאחר והאיזון לא יופר
2. בקדקוד מסוג 2, מוחקים את הקדקוד, ואז מתקנים על פי האב:
(שיטה מועילה שתסייע לזכור איך מטפלים בחריגות – בודקים כמה בנים יש ברמה האחרונה, ויוצאים מנקודת הנחה שהדרגה האחרונה מכילה רק עלים. אם מוצאים שיש שלושה עלים, אנחנו מחפשים איך לאחד אותם תחת קדקוד מסוג 3, ואם מוצאים 4, מחפשים להכניס אותם תחת שני

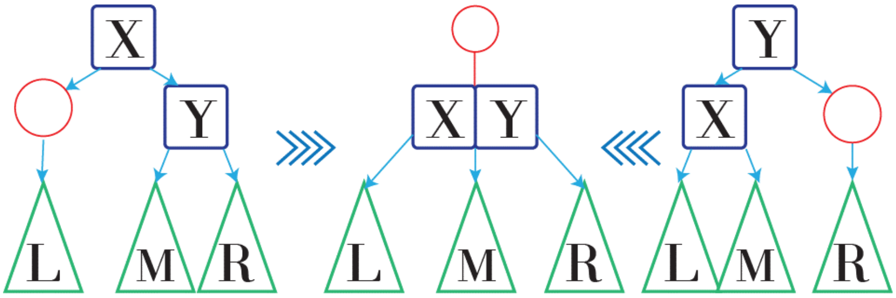
[17] אני מתייחס אליו בתור אות מ' רגילה

אבות מסוג 2. כמובן שאם יש כמויות גדולות של עלים שניתן לחלק בצורות שונות, זה עלול להיות שונה, אבל אפשר להסתייע בזה כ"כלל אצבע")

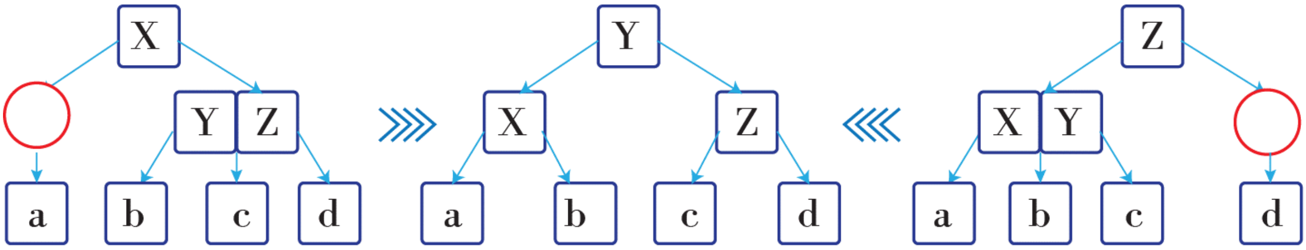
א. אם האב הוא מסוג 3, חייבים שיהיו לו 3 בנים, ולא ניתן להשאיר מצב של קדקוד 3 שאינו מלא, במקרה כזה, מה שעושים הוא "פעפוע" והצפה של הבעיה כלפי מעלה עד שהחור יתוקן, או עד שיגיע לשורש ומשם יתפוגג –

<u>שלב עליה:</u>

1. <u>לחור יש הורה מסוג 2 ואח מסוג 2</u> –
בשני המקרים המתוארים, מאחדים את האח להיות קדקוד 3. ה"חור" שמופיע בשורש, למעשה נבלע ונעלם והשורש יהפוך להיות קדקוד 3 בעל הערכים XY.

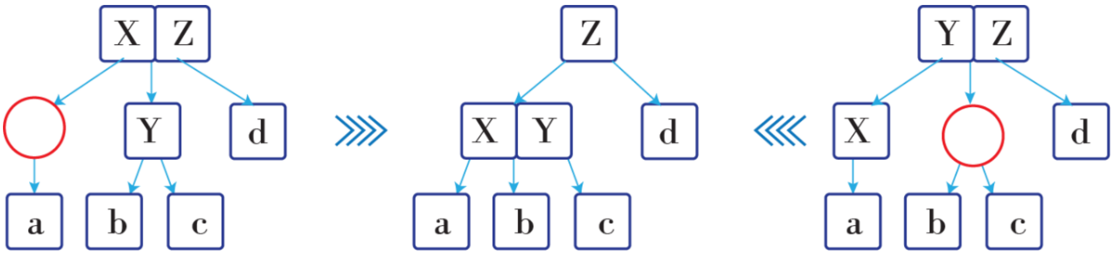


2. <u>לחור יש הורה מסוג 2 ואח מסוג 3</u> –
ברמה הקיימת הנוצרת, יש לנו 2 ערכים קיימים שיבולים להיות אב, ו4 ערכים-עלים, שניתן לחלק בצורה שווה בין האבות החדשים. לכן הדבר היעיל ביותר במקרה כזה, הוא פשוט לפצל את האב הכפול להיות אבות מסוג 2, והקרוב ביותר לחור "מאמץ" את הבן הנותר.



3. <u>לחור יש הורה מסוג 3 ואח מסוג 2</u>[18]:

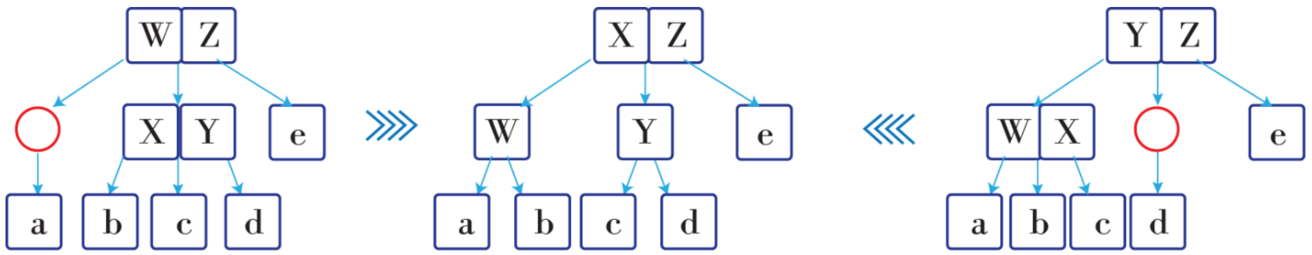
באופן דומה, יש לנו 3 ילדים באותה רמה, ואנו שואפים לאחד אותם תחת אותה מסגרת. ולכן את האב מסוג2 נהפוך להיות לאב מסוג 3, על ידי שנוריד אליו את



[18] שימו לב שהאיור הנ"ל מתייחס רק לתיקון של החלק בו יש חור, ולצורך העניין ניתן להניח שתחת הקדקוד D יש תת עץ הממשיך להיות ברמה אחידה עם שאר הקדקודים, על מנת לקיים עץ שלם.

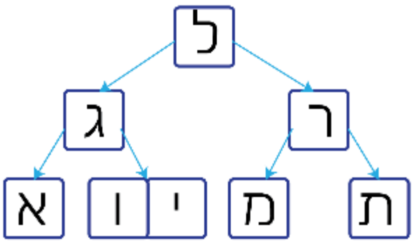
אחד הערכים מהרמה שמעל, מה שיהפוך אותו לקדקוד מסוג 2, ברמה מתחת נאחד את כל העלים להיות בנים של הקדקוד 3 החדש ביותר.

4.<u>לחור יש הורה מסוג 3 ואח מסוג 3:</u>
דומה למקרה ה2, בו לא נדרש שינוי בשורש העץ, מפצלים את הקדקוד האמצעי, ממצב בו הוא קדקוד מסוג 3, לשני קדקודים מסוג 2, שיבילו את כל העלים הנותרים.

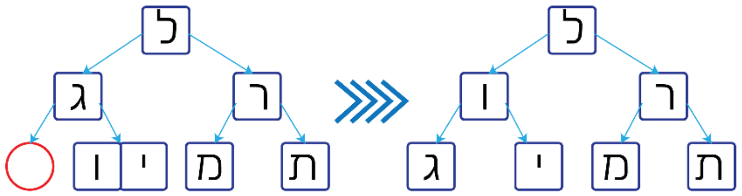


<u>דוגמה למחיקת עץ 2-3</u>

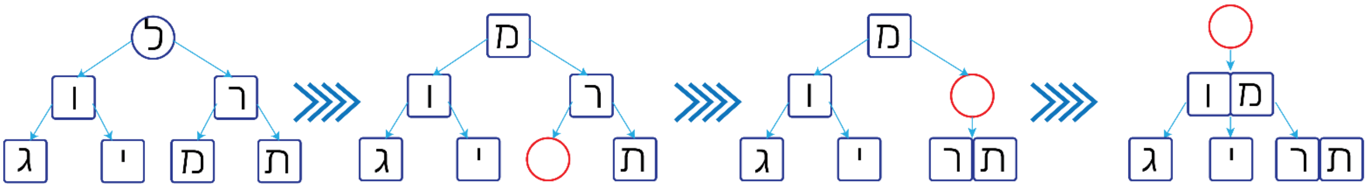
נמחק את העץ "אלגוריתם" שבנינו בחלק הקודם. הוא לא יכיל את כל אפשרויות המחיקה הקיימות, אבל ייתן כיוון כללי.



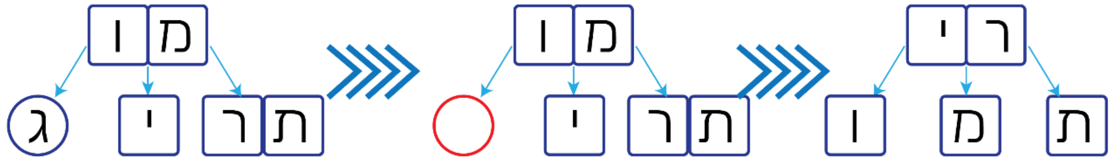
1. מחיקת 'א':
'א' הוא עלה, האב שלו הוא קדקוד מסוג 2, והאח שלו קדקוד מסוג 3. לוקחים את שלושת הערכים והופכים אותם לאחר המחיקה לתת-עץ מסוג 2, כאשר הערך האמצעי מבין השלושה – 'ו' – יהיה שורש תת העץ.



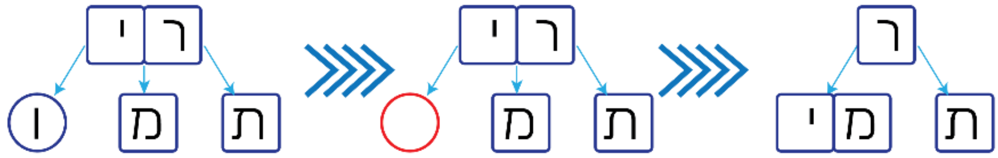
2. מחיקת 'ל':
מאחר וה'ל' נמצא בשורש העץ, נעביר אותו להיות במקום העוקב או קודם שלו. יש לשים לב שבל בחירה כזאת תשנה את צורת העץ להמשך, אך אין בדרך כלל עדיפות לבחור באפשרות מסוימת. החלפנו אותו עם ה'מ', ואיחדנו את 'ר' ו'ת' להיות עלים מסוג 3, את החור שנשאר לנו אנחנו מפעפעים כלפי מעלה, מה שגורם לאיחוד של השורש הנוכחי עם הבן שלו, והופך את כל העץ להיות עץ מסוג 3.



3. מחיקת 'ג':
הערך 'ג' הוא עלה בתחתית העץ, כך שאין לנו צורך להזיז אותו ממקומו. נמחק אותו וכך יישארו לנו מעבר לחור שנוצר, חמישה ערכים המסודרים כבר במבנה כמעט קלאסי של קדקוד מסוג 3 – או בפירוט המקרים שציינו כבר, מדובר במקרה הרביעי בו יש לנו קדקוד מסוג 3, ואח מסוג 3. מה שעלינו לעשות הוא לפצל את קדקוד האח כך שיהיה קדקוד מסוג 2, ולסדר את כל הערכים בצורה שתהיה תואמת לדרישות הלקסיקוגרפיות.

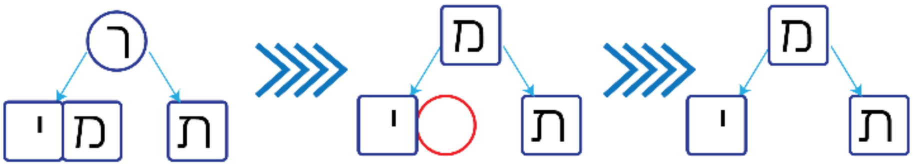


4. מחיקת 'ו':
מוחקים את ה-'ו' ממקומה בקצה העץ, ומאחר והאיזון הופר, וישנם רק 4 ערכים, ניתן לבנות מזה רק עץ מסוג 2. כמובן שהאיבר הכפול יהיה ברמה התחתונה יותר בעץ ולא בשורש, ולכן מורידים

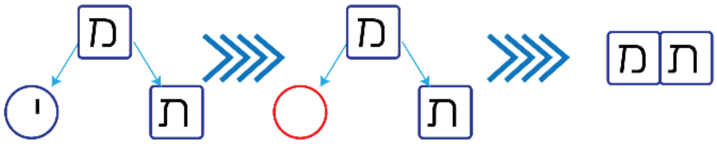


את ה-'י' ביחד ומשמאל ל-'מ'

5. מחיקת 'ר':
מורידים את ה-'ר' ומחליפים עם הקודם 'מ' (מאחר ואנחנו רואים שיש לנו שם קדקוד מסוג 3, הרבה יותר נכון לבחור אותו על מנת "לצמצם נזקים"), לאחר מכן מוחקים את הערך וממשיכים.



6. מחיקת 'י':
מוחקים את 'י', את שני האיברים הנותרים מאחדים תחת איבר יחיד מסוג 3.



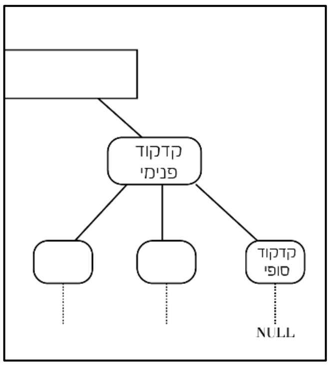
7. מחיקת 'ת' - סיום:
מוחקים את ה-'ת', נשאר איבר בודד.

עצי B

עצי B פועלים בשיטת מיון דומה לעצים בינאריים (וכן לעצי 2-3) אך מכילים כמות הרבה יותר גדולה של נתונים. שיטת המיון תורמת לכך שסיבוכיות הזמן העבודה בפונקציות השונות תמיד יהיה זמן לוגריתמי. היתרון החשוב ביותר של עצי B, הוא שניתן להכניס בו כמות מאוד גדולה של נתונים, וכך באותו זמן לוגריתמי אנחנו יכולים להגיע להרבה יותר מידע, דבר שרלוונטי מאוד ומסייע בגישה לכמויות גדולות של מידע כאשר זמן הגישה עצמו הוא איטי. בעצי B סיבוכיות הזמן היעילה מפצה גם על הגישה האיטית.

תכונות עצי B

עבור כל עץ B יוגדר ערך הנקרא "רמת מקסימום", אותו נהוג לסמן באות m, המתייחס למקסימום הערכים שניתן להכניס בצומת.



סוגי קדקודים אליהם אנחנו מתייחסים בעצי B:

1. **קדקוד פנימי** – קדקוד הנמצא בתוך העץ המכיל ערכים בכמות המתאימה להגדרה, ומכיל גם בנים מתחתיו. מספר הבנים יהיה n+1 למספר הערכים הקיימים בו (כאשר אין זה אומר כלום לגבי כמות הערכים הקיימת בבנים)
2. **קדקוד סופי** – יש לו ערכים ובניו כולם NULL. קדקודים אלו יהיו ברמה הנמוכה ביותר של העץ.
3. **קדקוד NULL** – לא מכיל בתוכו ערכים[19].

כמות המפתחות המותרת בבל צומת:

1. המפתחות בבל **קדקוד פנימי / סופי** (לא שורש) מונחים בתוך מערך ממוין. כמות המפתחות-ערכים בכל קדקוד יכול לנוע במקרה הגדול ביותר ל-m-1 ערכים בתוך הצומת. כאשר, על מנת לשמור על העץ בתור עץ שהחיפוש בו יתבצע בסיבוכיות יעילה ולוגריתמית, מינימום הערכים שניתן להכניס בכל צומת הוא (m/2)-1
יש הדואגים לדאוג לסמן גם את רמת המינימום שתהיה חצי מה-m, אך אנו לא התייחסנו לזה.
2. **לשורש** יכול להיות בין 1 ל m-1 מפתחות. המקסימום כמו בכל קדקוד אחר, והמינימום יכול להגיע לאחר מספר הכנסות וסידורים של העץ, שיגיע לפיצול שבזה.

מספר הבנים של כל צומת:

כמו שכבר צוין בחלק הקדקודים, לכל צומת יש מספר בנים שונה, על פי כמות המפתחות המאוכסנים אצלו. נראה תיכף שסידור הערכים מתבצע כמו שלמדנו לגבי עצי 2-3 בקדקוד מסוג 3. קדקוד זה מכיל את הבן האמצעי של תת העץ, כאשר הערך שלו יהיה בין שני האבות שלו. באופן דומה, כמות הבנים גם פה תהיה גדולה באחד מכמות הערכים, כך שהבנים יוכלו להתפרש בצורה נאותה בין כל המפתחות באב. בהתאם לזה נגדיר את האפשרויות השונות-

1. של **קדקוד פנימי** – לפחות m/2 ומקסימום m.
2. של **שורש** - לפחות 2 ומקסימום m.
3. כל הקדקודים הסופיים נמצאים באותה רמה – צריך לשמור על איזון מושלם של העץ.

[19] נתעלם מהם כמו בעצי 2-3.

בנוסף, על מנת לשמור על הנתונים ולדעת איך להגיע אליהם בצורה נוחה, בכל קדקוד שומרים מספר נתונים נוספים:

Nkeys – מספר הערכים השמורים באותו קדקוד.

[]KEY– מערך ממוין בסדר עולה (או לא יורד, במידה ומותר כפילויות), המכיל את כל המפתחות המאוכסנים בתוך המערך.

[]Son – רשימה של מצביעים לבנים, כאשר במידה ואין בנים שמוגדרים תחת הצומת הרשימה יכולה להיות ריקה – יש לשים לב, גם בעלים התחתונים ביותר יהיו מצביעים לבנים שהם NULL (מאחר ויבולים התמלא בהמשך), ורק בקדקודי NULL הרשימה תהיה ריקה.

נתונים אופציונליים אותם ניתן לשמור בכל קדקוד:

()isLeaf – משתנה בוליאני המכיל ערך אמת -1- במידה והגענו לעלה, ו-0 במידה ואנחנו נבוך צומת פנימית/שורש. הבדיקה במובן היא פשוטה – בודקים האם יש לו מצביעים ואם הם מצביעים על ערכי NULL המייצגים סוף רשימה

Parent – מצביע להורה של הקדקוד. מסייע במקרים של הכנסה והוצאה

[]Info – מידע של הקדקוד בנוסף למפתח.

פעולות על עץ B

ניתן לשים לב די בקלות, שכל הפרטים והתנאים שצוינו לעיל, הם תנאים שמתקיימים גם בעצי 2-3, ואכן, עצי 2-3 הם מקרה פרטי של עצי B. כך שאת כל הפעולות הבסיסיות של העץ אנחנו אמורים להכיר, רק שכאן עושים טיפה שינויים והגבלות על מנת שיתאים למקרה הכללי של עצי B.

חיפוש – בדומה לעץ חיפוש, גולשים בעץ עד שמגיעים לרמה המתאימה ואז בודקים האם הערך נמצא במערך.

הכנסה – מורידים את הערך עד למקום המתאים לו, ומתקנים כלפי מעלה. התיקון יבצע בצורה של פיצול והזזה של הערכים והצמתים תוך כדי שמירה על האיזון

מחיקה – מוחקים את הערך באשר הופכים אותו לעלה על יד החלפתו עם העוקב או הקודם שלו, ומפעפעים את התיקון כלפי מעלה.

מאחר שהכל מתבסס על עצי חיפוש, כמו שכבר נאמר, ניתן לומר בצורה כללית שזמני הריצה יהיו לוגריתמיים.

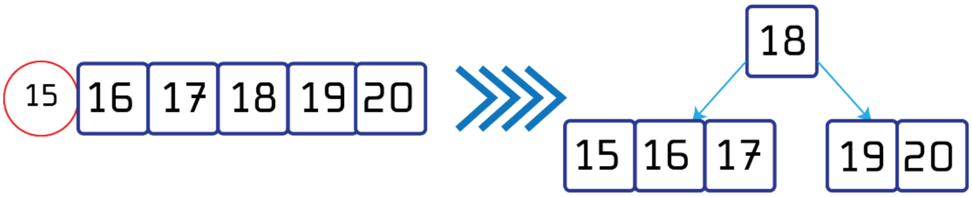
בניית עץ B

כאשר מקבלים את הערכים ואת הדרגה המקסימלית של העץ, פועלים בסדר הבא:

1. מכניסים את האיברים הראשונים בשרשרת עד שמתקבל בשורש m-1 ערכים.
2. האיבר הבא שמוכנס גורם לפיצול. את הפיצול מבצעים בצורה הבאה:

א. אם מספר הבנים הוא זוגי – הווה אומר, הי לנו מערך עם מספר איברים זוגי, ואנחנו מכניסים ערך נוסף גורם לפיצול, פשוט נחלק את האיברים כאשר הערך האמצעי יהיה השורש, ושאר הבנים יתחלקו בצורה מתאימה מימין ומשמאל.

ב. כאשר נדרש מאתנו לפצל קדקוד המכיל m ערכים כאשר ה-m הוא זוגי, מפצלים את הערכים לחצי, כאשר המחצית הראשונה תכנס להיות הבן השמאלי, והערך השמאלי ביותר מהמחצית השניה יהפוך להיות שורש תת העץ, והשאר יהיה הבן הימני, לדוגמה:
אנחנו רוצים להכניס את הערך 15 לקדקוד המכיל כבר 5 ערכים, וה-m המוגדר הוא 6. אנחנו מפצלים את הקדקוד בצורה הבאה:



3. ממשיכים להכניס את האיברים בעלים עד שממלאים בהם את המקום, על פי הגדרות העץ.
4. בבל מקרה בו מגיעים למקסימום ערכים אפשריים, מפצלים את הקדקודים, ומסדרים את הערכים על פי הצורך.

בניית עץ לדוגמא

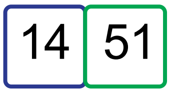
נדרש לבנות עץ B ממערך המספרים הבא: 14, 51, 7, 40, 118, 13, 21, 9, 16, 312,101 (משמאל לימין)

דרגת מקסימום 4 = (m)

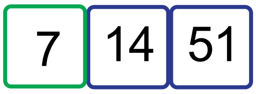
שלב 1:

כאמור, בשלבים הראשונים אנחנו פשוט דוחפים את האיברים למערך השורש, עד שנגיע לכמות שדורשת פיצול

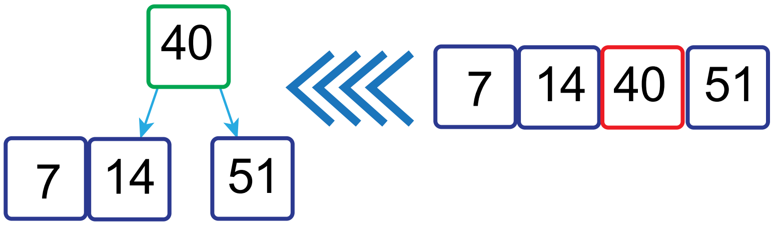
שלב 2:



שלב 3:

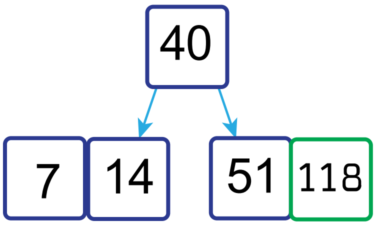


שלב 4:



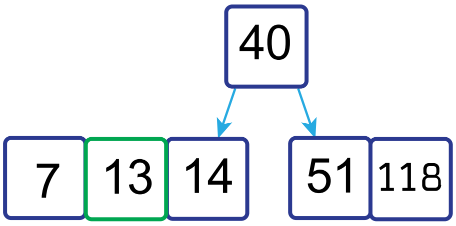
ההכנסה של האיבר הרביעי גורמת לכך שמגיעים למקסימום האיברים האפשרי בשורש, ולכן מפצלים את הקדקוד. כזכור, ברגע שמספר הקדקודים המפוצלים הוא זוגי, דואגים לכך שהבן השמאלי יהיה גדול יותר ב-1 מהימני.

שלב 5:

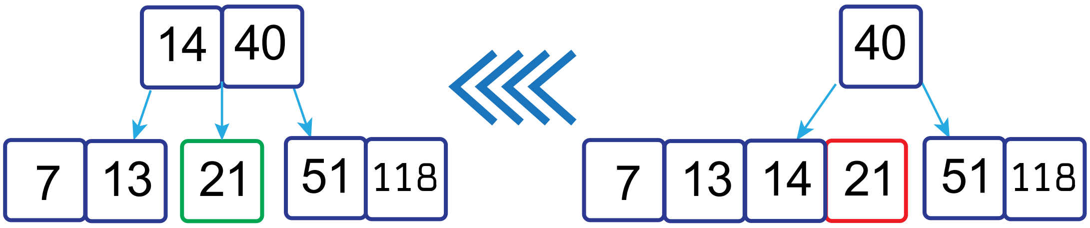


לאחר הפיצול הראשון, ממשיכים להכניס איברים במערך עד שמגיעים לשלב בו צריך לבצע פיצול נוסף באחד מהבנים.

שלב 6:



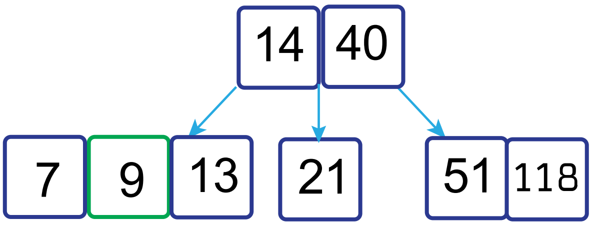
שלב 7:



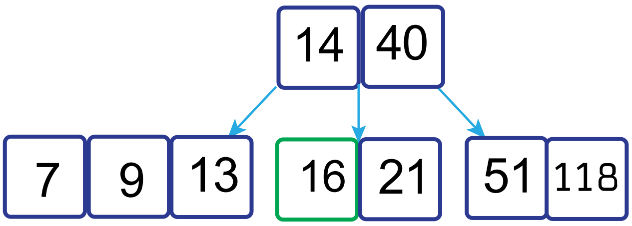
הפיצול כאן הוא קצת דומה לפיצול בעץ 2-3. כמובן שלא ניתן לפצל רק את הבן הנוכחי, מאחר ואחת הדרישות החשובות בעץ B היא שהעץ יישאר שלם ללא יוצא דופן. ולכן אנחנו מרימים את האיבר ה"אמצעי" מתוך המערך, כמובן בהתחשב בזוגיות המערך ושמירה על הכלל שהחלק השמאלי יהיה בעל יותר איברים. את האיבר האמצעי אנחנו מכניסים בקדקוד הראש, בצמוד לאיבר שהיה בו קודם.

את חלוקת הבנים אנחנו מסדרים לפי הכלל שבאשר בקדקוד יש n איברים, חייב שיהיו לו n+1 בנים. ולכן אנחנו דואגים שיהיו לו 3 בנים. החלק המפוצל יתחלק לבן שמאלי ואמצעי (שיהיה כמובן בין שני איברים בשורש), והחלק הימני יישאר במקומו ללא שינוי משמעותי.

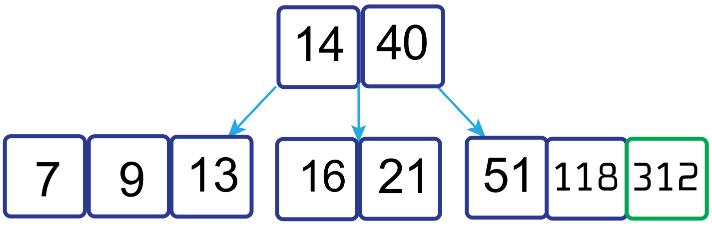
שלב 8:



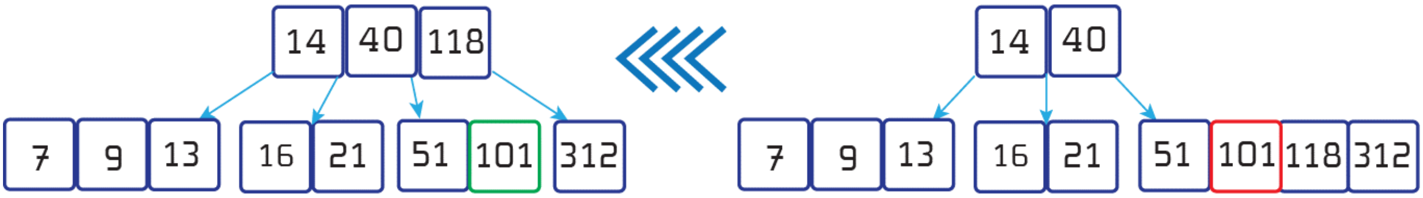
שלב 9:



שלב 10:



שלב 11:



עד השלב האחרון הכל הלך יחסית חלק, וגם כאן ההכנסה היא יחסית פשוטה – בדומה לפיצול הקודם שעשינו, האיבר האמצעי עולה לשורש, והקדקוד עצמו מתחלק בין האיברים בצורה המוגדרת – האיבר האמצעי עולה לראש, והשאר מתחלקים ביחס 2:1 והעץ מובן.

פתרון נוסחאות נסיגה

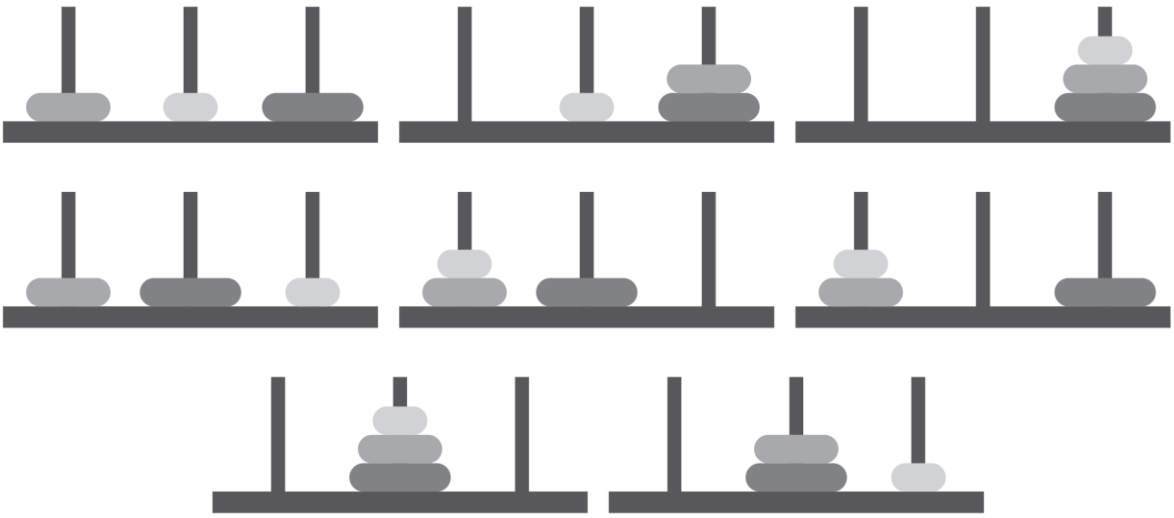
בקורס הקודם, מבנה נתונים א', דיברנו על ניתוח זמני ריצה.

מונחת לפנינו פונקציה, ועל ידי קריאת הקוד, והבנת המהלך של התכנית, אנחנו יכולים לעריך את זמן הריצה של הפונקציה. למשל, אם יש לנו לולאה כלשהי, אנחנו יכולים להסתכל על תנאי ההתחלה והסיום של הלולאה, ובכן על אופי האיטרציות, ולא משנה אם מדובר בלולאת while או for, אנחנו יכולים להגיד אם מדובר בזמן של $\Theta(n)$, יותר או פחות.

בנוסף דיברנו בקורס הקודם, וגם בקורסים אחרים על אלגוריתמים רקורסיביים. מונחת לפנינו בעיה גדולה שאין לנו יכולת להתמודד מולה, ודרך הפתרון שאנו נוקטים, הוא לפרק את הבעיה לתתי-בעיות קטנות, עד שאנחנו מגיעים למצב אטומי-ראשוני אותו אנחנו יכולים לפתור בצורה פשוטה.

דוגמא מוכרת מאוד היא סדרת פיבונאצ'י – סדרה בה כל איבר מורכב מחיבור שני האיברים שלפניו. בהגדרה הפשוטה ביותר של סדרת פיבונאצ'י, או בהסבר של הסדרה, אנחנו מתחילים להסביר את סדר האיברים – 1,1,2,3 וכו' ועל ידי הראייה של האיברים הראשונים אנחנו מבינים את הכלל ומכאן ועד להגיע מספר מסוים בסדרה זה פשוט לעקוב אחרי ההתקדמות שלה.

דוגמא נוספת שראינו היא מגדלי האנוי – משחק מתמטי בו צריך להעביר ערימת דסקיות בין מוט שעליה הן מושחלות לאחד משני מוטות אחרים. החוקים העיקריים הם שמותר להעביר רק דסקית אחת בכל פעם, ושאסור להניח דסקית על גבי דסקית שקטנה ממנה. דרך הפתרון למשחק הזה מגיע על ידי אינדוקציה: מתחילים בערימה המכילה רק דסקית אחת – קל לראות שניתן להצליח במשימה. נתקדם הלאה ל-2 דסקיות – מעבירים אחת למוט השלישי, אחת למוט השני, ועליו מערימים את הדסקית הראשונה שהזזנו – והמשימה הושלמה. הפתרון האמיתי מתחיל להגיע עבור מגדל של 3 דסקיות – עושים את השלבים הראשונים כמו ברמת 2 הדסקיות, אך לאחר שעושים ערימה של 2 דסקיות, מעבירים את דסקית הבסיס למוט שלישי, מפצלים את שתי הדסקיות ומערימים הכל בחזרה. מצורף תיאור גרפי להמחשה:



ברגע שהבנו את השלב הזה, ניתן להשליך על השלב הבא – עושים בדיוק את אותם השלבים, מעבירים את הדסקית האחרונה למוט השלישי, ואז אנחנו מעבירים שוב את המגדל הקטן יותר ב n+1, ואת זה הרי כבר לא צריך להסביר לנו איך לעשות ☺.

גישה זו לפתירת הרקורסיה מכונה גישת "הפרד ומשול", מפרידים את הבעיה הגדולה, עד שהיא קטנה מספיק שנוכל לטפל בה/"למשול" עליה, ואז מצרפים את הכל לכדי הפיתרון השלם.

כאשר נרצה לנתח זמן ריצה של פונקציית רקורסיה ניתקל בבעיה. אם עד עכשיו דיברנו על על פונקציות שעובדות בצורה שיטתית ומסודרת, פונקציות הריקורסיה עובדות עם גודל משתנה של נתונים בכל איטרציה. אם ניקח את מגדלי האנוי, נוכל לשים לב שכמו שכבר צוין, אנחנו מתעסקים בכל פעם עם n-1 איברים, כך שקשה להכריז באופן ברור כמה הפונקציה רצה ומה היא עושה עכשיו.

צורה כללית של נוסחת נסיגה

אם נרצה לחשב כמה זמן ייקח לנו לבצע פעולת ריקורסיה - הזמן הכללי של שיטת "הפרד ומשול" – נצטרך לקחת כל מרכיב בנפרד ולהבין מה הזמן הדרוש לביצועו. ניקח בחשבון שחלוקת זמן-הביצוע היא לאו דווקא חלוקה שווה של a חלקים שגודל כל אחת מהן הוא $\frac{1}{a}$ מגודל הבעיה הכללית, אלא שייתכן מצב של חלוקה טיפה שונה ובן תתי-חלוקות, כך שמגדירים את החלוקה כ-$\frac{1}{b}$.

החישוב הכללי יהיה כדלקמן:

<u>זמן חלוקת הבעיה:</u> D(n) – הפרד (divide).

<u>זמן הפתרון של תתי הבעיות:</u> aT(n/b) - משול (time) – פתרון תתי-הבעיות הוא בעצם הנתון האמיתי שלוקח לנו לפתור את זמן הריצה של האלגוריתם, מאחר שחלוקת הבעיה והצרוף, שניהם מספרים קבועים, ואילו פתרון תתי הבעיה ייתן לנו את אמת המידה האמיתית.

זמן צרוף הפתרונות - C(n) (צרף) – (connect)

סה"כ:
$$T(n) = \begin{cases} \Theta(1) & : n \le 1 \\ aT\left(\frac{n}{b}\right) + D(n) + C(n) & : else \end{cases}$$

נוסחה כזו מכונה "נוסחת נסיגה" והיא למעשה ביטוי של זמן הריצה בצורה של נוסחה מתמטית. אנחנו נדרשים לקחת את הנוסחה הזו, להפוך אותה מבפנים ולהגיע לזמן הריצה האמיתי.

עכשיו אחרי שהבנו את "מגדלי האנוי", ואיך בונים נוסחת נסיגה, ננסה לבנות את נוסחת הנסיגה של מגדלי האנוי, ונסתכל על האלגוריתם המפעיל אותו:

```
hanoi(n,A,C,B)
If (n=1)
   move one ring from A to C
Else
   hanoi(n-1,A,B,C)
   move one ring from A to C
    hanoi(n-1,B,C,A)
```

הפונקציה מקבלת "מערך" של טבעות, ואת שלושת העמודים – כאשר העמוד האחרון מוגדר באמצע ולא בסוף.

תנאי העצירה שלה הוא כשנותרה לנו רק טבעת אחת בעמוד המקורי, הטבעת האחרונה, ואנחנו מעבירים אותה לטבעת הרצויה.

בכל מקרה אחר, אנחנו מפעילים את האלגוריתם הרקורסיבי – מורידים איבר אחד מהמערך הראשי, ושוב על זה הדרך כאשר יש בכל פעם שינוי של סדר העמודים, על מנת לעמוד בתנאים של מגדלי האנוי, ולאחר שמגיעים ל n=1 מתחילים להזיז את הטבעות על פי הסדר עד שהערימה עוברת לעמוד הבא.

בעת נותר לנו להבין כיצד אנחנו מוציאים מהקוד ומההבנה שלו את נוסחת הריצה.

נגדיר: מספר ההזזות שצריך לבצע על מנת להעביר את כל n הדסקיות ממוט אחד לאחר יוגדר בתור T(n).

על פי הניתוח המילולי שעשינו, על מנת להעביר את n הדסקיות, דרושים לנו שלושה שלבים:

1. העברת n-1 דסקיות למוט אחד (T(n-1)) – הפרדה.
2. העברת הדסקית האחרונה למוט השלישי (1) – פתרון.
3. העברת n-1 הדסקיות שיהיו מעל הדסקית האחרונה T(n-1) – צירוף.

נחבר את שלושת החלקים לכדי נוסחה אחת ונקבל: T(n) = T(n-1)+1+T(n-1)

ובצורה הפורמלית יותר: $T(n) = \begin{cases} \Theta(1) & : n = 1 \\ 2(T(n-1)+1 & : else \end{cases}$

לסיכום – על מנת להוציא מהקוד לפנינו את נוסחת הנסיגה, עלינו לנתח את חלקי הריקורסיה השונים, להבין כמה חלקים מתוך הפונקציה המקורית נכנסים אליה, מתי היא מסתיימת, ומה היא מחזירה – לפעמים מדובר בנתון בודד, ולפעמים מדובר בהחזרתמערך של n מספרים ואפילו יותר.

דוגמה מוכרת שמקיימת את נוסחת הנסיגה שתיארנו: מיון מיזוג–

```
]Merge-Sort(A,p,r)
    if p<r
       then q <- (p+r)/2
        Merge-Sort (A,p,q)
        Merge-Sort (A,q+1,r)
        Merge (A,p,q,r)
```

לוקחים מערך גדול לא-ממוין ושני איברים שיהוו מצביעים לגבולות המערך הדרוש, ורוצים להפוך אותו למערך ממוין. מאחר ואין לנו דרך למיין ישירות את המערך (או לפחות לא בזמן ריצה יעיל), אנחנו מתחילים בחלוקה של המערך לחצי. כל תת מערך שכזה, אנחנו מחלקים שוב לחצי, עד שמגיעים למצב בו יש לנו רק איבר אחד במערך. בשלב זה מתחילים למזג – לפתור – את הבעיה, תוך כדי שדואגים לצרף את החלק הפתור לחלק שמעליו, עד שמגיעים למערך השלם הממוין בצורה נכונה.

מאחר שיש לנו שתי חלוקות שוות לחצי וטיפול בשני החלקים, נכניס בנוסחה את זמן הריצה $T\left(\frac{n}{2}\right)$ פעמיים – אחד בנגד כל חלוקה. ומכיוון שעלינו להחזיר את כל המערך לאחר שהוא מויין, נוסיף על כך Θ(n).

נוסחת הנסיגה המתאימה, אם כן, למיון המיזוג:

$$T(n) = \begin{cases} \Theta(1) & : n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & : n > 1 \end{cases}$$

לפני שנמשיך בחישוב נוסחת הנסיגה, חשוב להזכיר שלוש נוסחאות נסיגה חשובות:[20]

[20] את חשיבות הזיהוי של נוסחת הנסיגה נראה בהמשך, רק נציין, שאם מכירים נוסחה ומזהים משהו דומה לה, זה יכול לתת לנו כיוון לאן הפונקציה אמורה להגיע.

מיון הכנסה - $T(n) = \begin{cases} 1 & : n \le 1 \\ T(n-1)+n & : else \end{cases}$

באופן די גורף נוכל לומר שאלגוריתמים של מיו מחזירים לנו את המערך הממויין, ולכן זמן הפתרון שלה הוא n. במיון הכנסה, כזכור, בודקים כל איבר אל מול כל אלו שקדמו לו כבר – לכן אם נסתכל באופן הפוך על הפעולה – הפעולה האחרונה היא לקחת את האיבר ה-n במערך ולהשוות אותו אל מול אלו שקדמו לו ב n-1 התאים האחרים במערך.

חיפוש ליניארי ברשימה מקושרת - $T(n) = \begin{cases} 1 & : n \le 1 \\ T(n-1)+1 & : else \end{cases}$

כאשר מחפשים ברשימה, מתחילים מאיבר הראש ומתקדמים איבר אחד בכל פעם עד שמוצאים (או לא מוצאים) את האיבר הדרוש. בדומה למיון ההכנסה, מאחר ואנחנו בודקים בכל פעם חלקים קטנים יותר של המערך, אנחנו מתייחסים אליו בתור n-1, ומאחר שבסוף עלינו להחזר רק את האיבר הדרוש, זמן הפיתרון הוא 1.

חיפוש בינארי - $T(n) = \begin{cases} 1 & : n \le 1 \\ T\left(\frac{n}{2}\right)+1 & : else \end{cases}$

ראשית נשים לב שמאחר ומדובר בפונקציית חיפוש שמחזירה איבר בודד, זמן הפתרון הוא 1. השוני פה משני הפונקציות הקודמות הוא, ששתי הפונקציות הקודמות מורידות בכל פעם איבר אחד מתוך אלו שהתקבלו לפעול עליהן. את בחיפוש בינארי אנחנו מכניסים בפונקציה בכל פעם רק חצי מכמות המערך הקודם, ולכן אנחנו מסמנים אותו ב $T\left(\frac{n}{2}\right)$.

לאחר שניתנו את הקוד והתקבלה לנו נוסחת נסיגה מתאימה, עלינו לחשב את זמן הריצה המתאים עבור הפונקציה. על מנת לעשות זאת ישנן ארבע דרכים עיקריות:

שיטת ההצבה

שיטת ההצבה מסתמכת על כך שיש לנו בסיס ראשוני איתו אנחנו יודעים לעבוד. לדוגמא, אם אנחנו מכירים את נוסחת הנסיגה של מיון-מיזוג, ואנחנו רואים נוסחה דומה אליה יחסית, נוכל לנחש שגם זאת באופן דומה למיון מיזוג חסומה על ידי O(nlogn), כמובן שיכול להיות שיהיו שינויים נדרשים, אך נגיע גם לזה.

לאחר שמצאנו את הנוסחה המתאימה שאנחנו מגדירים כחסם עליון, נוכיח את החסם בצורה פורמאלית בדיוק כמו שלמדנו בחלק א' של הקורס -

למשל, אם נקבל את נוסחת הנסיגה הבאה: $T(n) = 2T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + n$, נוכל לנחש שאנחנו מדברים על פונקציה של O(nLogn), ונוכיח זאת באינדוקציה.

הוכחה:

עלינו להוכיח שקיימים קבועים חיוביים c, n_0 כך שלכל $n>n_0$ יתקיים $T(n)>c(n\log n)$.

1. נתחיל עם ההנחה שהחסם נכון עבור $\left\lfloor\frac{n}{2}\right\rfloor$ ולכן – $T(\left\lfloor\frac{n}{2}\right\rfloor)\leq c\left\lfloor\frac{n}{2}\right\rfloor\log\left\lfloor\frac{n}{2}\right\rfloor$. להוכיח את זה קל, מאחר והפונקציה הימנית היא לא-יורדת בצורה מונוטנית. בעת נוכל לומר שעבור T(n) (למעשה מכפילים את הפונקציה ב2), מתקיים כי $T(n) = 2T(n/2)+n\leq 2(c\left\lfloor\frac{n}{2}\right\rfloor\log\left\lfloor\frac{n}{2}\right\rfloor)$
2. נצמצם את המשוואה הימנית:
$2(c\left\lfloor\frac{n}{2}\right\rfloor\log\left\lfloor\frac{n}{2}\right\rfloor) = c2\frac{n}{2}logn - log2 + n$ = cn(logn-1)+n=<u>cnlogn-cn+n</u>.
3. אחרי שמצאנו את הנוסחה, נשווה אותה עם הפונקציה אליה אנחנו מכוונים כחסם אסימפטוטי:
$T(n)\leq cnlogn-cn+n\leq cnlogn$
$-cn+n\leq 0$
$n\leq cn$
$1\leq c$
4. לאחר שקיבלנו את ה-c המתאים אנחנו נבדוק אותו בפונקציה ונוודא נכונות[21]

כלומר, עבור כל $c\geq 1$, ניתן לומר שהוא מקיים אחר כלל הנסיגה שציינו.

הקושי העיקרי הבולט בשיטה זאת, היא ההתבססות על הניחוש שהוא נכון וקרוב מספיק לנוסחה שאנחנו רוצים. אך כיצד ניתן לקבל מושג אילו פונקציות אנחנו צריכים "לנחש"?

עבור ההחלטה הזאת, ישנם מספר היורסיטיקות שניתן להשתמש בהם:

1. אם הנוסחה דומה לנוסחה רקורסיבית שראינו בעבר, אז מנחשים פתרון דומה – למשל, אם הנוסחה המתקבלת היא $T(n) = 2T(\frac{n}{2}+17)+n$, נוכל להתעלם מהקבוע 17, ולהתקדם לכיוון נוסחת המיון-מיזוג.
2. נחש באופן "פרוע" חסם עליון ותחתון וננסה לצמצם את התווך. – אם ניקח את הפונקציה $T\left(\frac{n}{2}\right)+n$ נוכל באופן די מיידי להציב שהנוסחה הזאת נמצאת אי שם בטווח של $n<T(n)<n^2$, ואז לנסות ולצמצם את האפשרויות בעזרת בדיקה פורמאלית עד שמגיעים לטווח שהוא קרוב יותר לאמת.
3. מבצעים מניפולציה אלגברית על נוסחת הנסיגה, בדוגמת הצבה של משתנה ופותחים עד שמגיעים לפונקציה מוכרת יותר – למשל, עבור הפונקציה:$T(n) = 2T(\sqrt{n})+logn$ נציב כי m=logn. $T(2^m)=2T(2^{m/2})+m$. בהמשך לזה, נציב שוב: $S(m) = T(2^m)$, ונקבל: $S(m) = 2S(m/2)+m$, שאנחנו יודעים כבר שמדובר על mlogm. נמיר בחזרה את כל הדרך ונקבל:
$T(2^m)=O(mlogm)$
$T(n) = O(log(n)log(log(n)))$

<u>שיטת האיטרציה</u>

הרעיון הכללי של שיטת האיטרציה, הוא לקחת את הנוסחה אותה אנחנו מקבלים בנקודת פתיחה, ומתחילים להציב אותה בעצמה, עד שמקבלים מושג על המבנה הכללי של הסדרה. לאחר מכן מוכיחים באינדוקציה את הנוסחה (בדרך כלל עבור (n=1)), ולבסוף מוצאים את הפתרון הסופי.

עבור הבנת השיטה נסתכל על שתי נוסחאות שונות ודרכי הפיתרון:

דוגמא ראשונה:

[21] ספציפית במקרה הזה יש לשים לב שהטענה לא בדיוק נכונה, מאחר שהיא מתקיימת רק מ$c\geq 2n$.

$T(n) = 2T(n-1)+1$
$T(1) = 1$

<u>שלב ראשון:</u> הצבת הנוסחה בעצמה

(על מנת לעקוב בצורה נוחה יותר אחרי השינויים, כל חלק צבוע בצורה אחרת)

מהנתון הראשוני $T(n) = 2T(n-1)+1$, אנחנו מנסים להציב איטרציה אחת אחורה:

$2T(n-2)+1 = T(n-1)$. את הדפוס של (n-1) אנחנו יכולים לזהות בקיים בתוך הנוסחה המקורית, ולכן אנחנו יכולים להציב אותו בחזרה בתוך הנוסחה.

$T(n)= 2T(n-1)+1 = 2(2T(n-2)+1)+1$. נפתח את הסוגריים ונקבל:

$T(n) = 4T(n-2)+(2+1)$

הרעיון הכללי הוא, שעכשיו אנחנו בעצם יכולים להתחיל ללכת אחורה ל$T(n-2)$ וכו', עד שנגיע לאיטרציה הראשונה, כאשר נתון לנו ש$T(1)=1$. על מנת להמשיך את הכיוון הזה, נציב את $T(n-2)$ בנוסחה **המקורית** (עניין של נוחות מאחר והיא מגיעה בדרך כלל בצורה נוחה יותר), ואז נוכל להציב את התוצאה בנוסחה שפיתחנו, בה מופיע המשתנה של $T(n-2)$:

$\mathbf{T(n-2)} = 2T(n-3)+1$

$T(n) = 4\mathbf{T(n-2)}+(2+1)$

$T(n)=4(2T(n-3)+1)+(2+1)$. נפתח את הסוגריים ונקבל:

$T(n)=8T(n-3)+(4+2+1)$. בשלב זה כבר אפשר לזהות שיש פה איזה שהיא סדרתיות – המקדם של הT משתנה לפי חזקות של 2, וגם ה"שרוך" שמצטרף בסוף הוא בחזקות של 2. מכאן אפשר לעבור לשלב הבא:

<u>שלב שני:</u> ניחוש נוסחה כללית

כפי שבבר הערנו, אנחנו מצליחים לזהות את הסדרתיות, ועכשיו עלינו להמיר אותה לצורת נוסחה.

אם נפרק מה שקיבלנו, נוכל לראות שעבור **המקדם** של T, אנחנו מתחילים ב-2 ועולים בצורה הנדסית, ולכן קל לקבוע בצורה כללית שמדובר על 2^i.

החלק השני, נראה כמו סדרה חשבונית רגילה המתחילה מ1 ובכל פעם מוסיפים מספרים שהם חזקות של 2, ובצורה כללית יותר: $(2^{n-1}+2^{n-2}+\ldots+1)$, שסדרה זו היא המקבילה של הסכימה $\sum_{k=0}^{\infty} 2^k$, $2^i-1 =$.

נחבר הכל, ונאמר כי:

$T(n) = 2^iT(n-i)+2^i-1$.

בעת נותר לנו להוכיח את הניחוש שלנו.

<u>שלב שלישי:</u> הוכחת הנוסחה והוצאת נוסחה סופית

ראשית נבדוק עבור $i=1$, אנחנו יודעים לאן אנחנו אמורים להגיע, ואכן אנחנו מקבלים את הנוסחה הראשונית שלנו:

$T(n)=2T(n-1)+1$.

ובעת נבחר עבור $i=n-1$, ונמצא כי:

$T(n)=2^{n-1}T(n-(n-1))+2^{n-1}-1$

$T(n) = 2^{n-1}T(1)+2^{n-1}-1$

ידוע לנו כי $T(1)=1$, ולכן:

$T(n)=2^{n-1}+2^{n-1}-1$

$T(n)=2^{n}-1$

וזאת הנוסחה הכללית.

עצי ריקורסיה

מתוך עצי הריקורסיה ניתן להבין בהמשך את נוסחת האב.

נתחיל עם דוגמה:

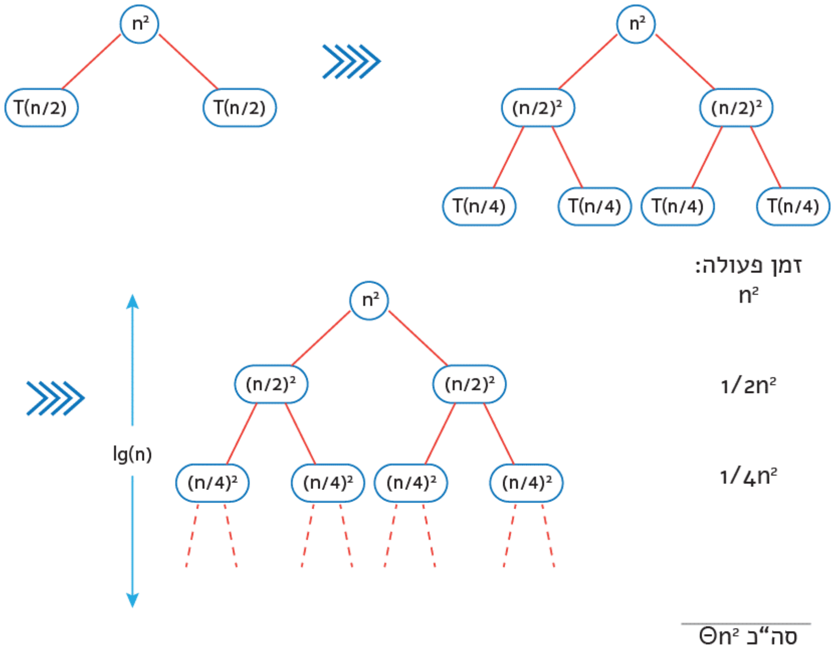
$T(n)=2T(n/2)+n^2$

נשרטט עץ, כאשר נתחיל האיבר שהוא לא רקורסיבי. כאן בדוגמה יש לנו את החלק הרקורסיבי שהוא $T(n/2)$, והחלק של N^2 חלק שהוא קבוע. ואז נפצל את הבעיות החלקיות כתתי עץ.

ואז אנחנו ממשיכים ומחשבים את הרקורסיות לתוך עצמם, וממשיכים עד שהמבנה יגדל ויגיע להיות שווה למונה ונקבל תוצאת שבר של 1. כאן ניתן לראות שהגורם המשותף לכל התוצאות הוא הn^2 וברגע שנוציא אותו מכל איחוד הזמנים נקבל $n^2(2)$, כאשר זה לא בדיוק 2, אלא טור שחסום על ידי 2.

בסופו של דבר נוכל לקבוע שמדובר בזמן ריצה Θn^2.

יש לשים לב, שלא תמיד העץ שנקבל יהיה בינארי. אלא הוא פונקציה של המקדם של הT. כאן הוא היה 2T, ולכן הוא בינארי, ואם יהיה יותר, מספר הבנים והפיצולים יהיה בהתאם.

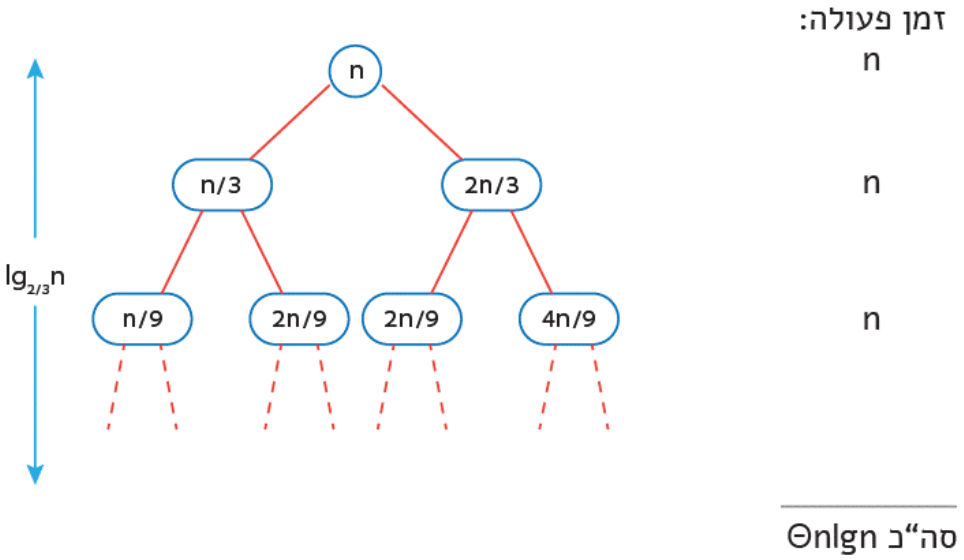


כמובן שהציור של עצי הריקורסיה הוא לא הוכחה רשמית, אלא מעין קיצור דרך שיכול לתת לנו מושג של הביוון אליו אנו שואפים.

דוגמא נוספת:

T(n)=T(n/3)+t(2n/3)+n

בדוגמה הזו רואים מקרה בו יש שני חלקים רקורסיביים לכל איטרציה. אבל מאחר שבעצם יש לנו 2T, גם אם הוא כתוב בצורה טיפה אחרת, עדיין מציירים את העץ בצורה בינארית, כאשר כל צד ייצא לנו אחת



מהנוסחאות השונות, ושאר העבודה נראה בדיוק אותו דבר כמו הקודם. הגובה הסופי של העץ יהיה למעשה LogN, מאחר ומדובר בעץ שלם, ובכל איטרציה אנחנו מגיעים בעזרת איחוד החלקים השונים לn. אם נשקלל את שתי התוצאות האלו, אנחנו מקבלים בצורה פשוטה n*logn

מה אנחנו למדים מהנוסחאות הללו?

האיברים הרקורסיביים הולכים וקטנים בכל שמקדמים, מעצם הגדרתם כריקורסיה, והענף הכי ארוך בעץ יסתיים ב-loga על הבסיס הכי קטן בו אנחנו מחלקים. בדוגמה השניה חלק אחד קטן פי 3, והשני פי 1.5, ולכן הזמן הרלוונטי הוא החלק הקטן יותר ששואף להקטנה איטית יותר.

בנוסף, אם נסתכל על הדוגמא הראשונה, נמצא מערכת רקורסיבית שלמעשה, בכלל לא משפיעה על זמן הריצה של הפונקציה. וזאת מאחר שבעוד שקצב הגידול של הבנים גדל פי 2 בכל איטרציה, גודל כל חלק בפני עצמו קטן בצורה מעריכית ובקצב שהוא הרבה יותר משמעותי מגידול האיברים, ולכן זה ייתן תוצאה שהיא זניחה יחסית לעומת האיבר החופשי.

משפט האב (מאסטר)

מתוך עצי הריקורסיה ניתן ללמוד צורה כללית של הגדרה.

.T(n)=aT(n/b)+f(n)

המשתנים a ו-b קובעים לנו את קצב הגידול של הרקורסיה, כאשר a הוא מספר הבנים בכל איטרציה, ו-b מגדיר את השינוי עצמו בn.

לנוסחת המאסטר ישנם שלושה מקרים בהם אפשר להציב במשוואות תחת אילוצים שונים. יש לשים לב, על אף השם המפוצץ של נוסחת "מאסטר", היא לא באמת מתאימה לכל הנוסחאות האפשריות.

חלוקת המקרים מתבצעת בצורה הבאה:

$$T(n)=\begin{cases}\Theta\left(n^{\log_b a}\right) & f(n)=O\left(n^{\log_b a-\varepsilon}\right)\\ \Theta\left(n^{\log_b a}\log n\right) & f(n)=\Theta\left(n^{\log_b a}\right)\\ \Theta\left(f(n)\right) & f(n)=\Omega\left(n^{\log_b a+\varepsilon}\right)\text{AND}\\ & af(n/b)<cf(n)\ \text{ for large } n\end{cases}\quad \begin{matrix}\varepsilon>0\\ c<1\end{matrix}$$

הרעיון הכללי של שיטת המאסטר – יש לנו $a^{\log_b n}$ עלים בעץ. מעבר על כל אחד מהם הוא בעלות של $\Theta(1)$. מה שנותן לנו בסך הכללי $\Theta(a^{\log_b n})$, שעל פי חוקי לוגים, שווה בעצמו גם ל $\Theta(b^{\log_a n})$.

המקרה הראשון הוא הפשוט יותר, ומקביל לדוגמה הראשונה בעצי הריקורסיה – החלוקה בין הבנים מתבצעת בצורה שווה, כך שהפונקציה גדלה בצורה איטית יותר באופן משמעותי מ $n^{\log_b a}$ ולכן ניתן לקבוע את הנוסחה להיות $\Theta n^{\log_b a}$.

במקרה השני - אם יינתן לנו מקרה בו שני החלקים אינם שווים, כמו בדוגמא האחרונה, נוכל לעשות הערכה מלמעלה ולהשוות את שני החלקים על מנת לקבל חלקים שהם שווים. למשל אצלנו, החלק הקטן הוא b=1/3, ועל מנת להגדיל אותו נכפיל אותו ב2. כך שהעלות של כל איטרציה תהיה $(a/b)^i$ אם fn חסומה מלמעלה על ידי $(n^{\log_b a}/n^{\varepsilon})$ובכל עוד האפסילון יהיה גדול מ0 זמן הריצה יהיה $\Theta n^{\log_b a}$ שלמעשה תהיה פונקציה פולינמיואלית שתקבע את קצב הגדילה.

אך אם ניקח את ה$f(n)= \Theta n^{\log_b a}$, נוכל לקבוע את זמן הריצה לסכום של $\Theta n^{\log_b a} * \log N$, המבטא את הגודל הסופי של העץ.

במקרה השלישי - אם נוסיף לנוסחה את האפסילון ולא נחסיר, נכפיל את הנוסחה בחזקה ששואפת לאינסוף שקצב הגידול שלה גדול יותר בצורה משמעותית עד שלמעשה זה יגרום ללוג להעלם ולהשאיר לנו רק את הפונקציה. אבל כל זה רק אם מתקיים תנאי הרגולריות: $\{af(n/b)\leq cf(n) \mid c<1\}$

דוגמאות:

$T(n)=9T(n/3)+n$

נציב: $a=9, b=3, f(n)=n$.

לאחר הצבה נקבל $n^{\log_b a}=n^{\log_3 9}$ וזה מתאים למקרה הראשון של שיטת האב, מה שפותר את הנוסחה להיות $\Theta(n^2)$.

$T(n)=T(2n/3)+1$

נציב: $a=1, b=3/2, f(n)=1$.

מתאים למקרה השני, מאחר ו $\log_b a = 0$, כך ש $n^{\log_{1.5} 1}$ שווה לפונקציה. ולכן הפתרון לנוסחה זו הוא Θlgn.

(כמובן שהוא מוכפל ב$n^{\log_{1.5} 1}$, אך מאחר ומדובר ב-1, הוא לא נוכח בפונקציה)

$T(n)=3T(n/4)+nlgn$

נציב: $a=3, b=4, f(n) = nlgn$

כאשר נציב בחזקת הלוג המתאים, נקבל: $n^{\log_b a}=n^{\log_4 3}=n^{0.79}$.

הפונקציה $f(n)$ חסומה מלמטה עבור n^1 ולכן נגדיר $f(n) = \Omega(n^{\log_4 3+\varepsilon})$ כאשר $\Omega=0.2$.

בעת נותר להוכיח את משפט הרגולריות, ואז נוכל לקבוע בוודאות שהנוסחה מתאימה למקרה מס' 3.

עבור n גדול דיו מתקיים:

$af(n/b) = 3(n/4)lg(n/4) \leq (3/4)nlgn = cf(n)$. עבור $c=3/4$. ולכן מתקיים פה המקרה השלישי, והפיתרון לנוסחת הנסיגה הוא $T(n) = \Theta(nlgn)$

$T(n) = 2T(n/2)+nlgn$

נציב: $a=2, b=2, f(n)= nlgn$.

לכאורה ניתן להגיד ש $n^{\log_b a} = n$, ולכן הנוסחה מתאימה למקרה השלישי, מאחר ו-nlogn בוודאי גדול משמעותי מ-n, אך קצב הגידול אינו פולינומיאלי. עבור על קבוע חיובי ℇ, היחס $f(n)/n^{\log_b a} = (nlogn)/n = lgn$ והוא קטן אסימפטוטית מ n^{ε} עבור כל קבוע חיובי ℇ. ולכן הנוסחה הזאת נופלת בין הכסאות – היא כמעט מתאימה למקרה השני וכמעט למקרה השלישי, אך בסופו של יום לא עומדת בתנאים המתאימים.

$T(n) = T(n/4)+T(n/2)+n^2$

במקרה זה יש לנו נוסחה שמורכבת משתי נוסחאות שונות, אותם אין לנו אפשרות לפתור ביחד. אך אם נתבונן, נוכל לתחום את הנוסחה הזאת סביב שתי נוסחאות אחרות, המכילות כל אחת את הT, ואת ה-(n) -

$2T(n/4)+n^2 \leq \mathbf{T(n/4)+T(n/2)+n^2} \leq 2T(n/2)+n^2$

את הנוסחה הימנית אנחנו יכולים לפתור בעזרת המקרה השלישי של נוסחאות הנסיגה ולקבל $\Theta(n^2)$.

את הנוסחה הימנית אנחנו יכולים לפתור בעזרת המקרה השלישי וגם הוא שווה ל-$\Theta(n^2)$. ומאחר שהנוסחה שקיבלנו תחומה בין שניהם, ניתן להשתמש במשפט הסנדביץ', ולקבוע שהנוסחה הזאת גם היא שווה ל-$\Theta(n^2)$.

מציאת האיבר ה-i

מציאת האיבר ה-i, הינו אלגוריתם רקורסיבי בו אנו מנסים לפתור בעיה המוצגת בסגנון "מצא את האיבר הרביעי בגודלו במערך".

כמובן, שבמקרה שמדובר באיבר הגדול ביותר או הקטן ביותר, המציאה שלו היא יחסית פשוטה, ואנחנו דנים על איבר במרחק מסוים.

בבעיה המוצגת אנחנו מקבלים מערך A בגודל n המכיל מספרים לא ממוינים ואנחנו נדרשים למצוא את האיבר ה-i בגדלו (כאשר $1\leq i\leq n$). ניתן לגשת לבעיה בצורה שפשוט ממיינים את איברי המערך, ואז ניתן בקלות יחסית למצוא את האיבר הדרוש. הבעיה היא שמיון שכזה באופן המהיר ביותר יתבצע תחת זמן של $\Omega nlogn$, ונותן לנו פיתרון אחיד לכל המספרים במערך. אך אנו מחפשים שהפתרון המוצע יהיה כזה שלא משנה את המערך, אלא משיג את התשובה, ושהפיתרון יהיה, עד כמה שאפשר, מהיר ויעיל -אנחנו מחפשים שיטה בה נוכל להגיע לזמן של $O(n)$. הזמן אליו אנחנו שואפים נגזר ממציאת איברי הקצוות – אם נרצה למצוא את האיבר הגדול ביותר או הקטן ביותר, פשוט נעבור על כל איברי המערך עם דגל שיונח על האיבר המתאים (באותו רגע) עד שבסיום הריצה נדע בוודאות שעברנו על הכל, והאיבר שהגענו הוא המתאים לפי ההגדרה.

האלגוריתם המוצע עבור מציאת האיבר הינו אלגוריתם רקורסיבי, ולכן מתקבל בצורה של נוסחת נסיגה, ומתבסס על אלגוריתם המיון-המהיר (QuickSort):

- מחלקים שרירותית את המערך לשניים.
  - בוחרים איבר ציר (pivot) סביבו נחלק את האיברים
  - את שאר האיברים מעבירים מימינן ומשמאל לאיבר הציר על פי הגודל.
- על כל צד מהאיברים הכמעט ממויינים מפעילים שוב בצורה רקורסיבית את האלגוריתם וממשיכים כך עד שנשארים רק שני איברים למיין.
- מחברים את כל חלקי המערך ומקבלים מערך ממויין.

ההבדל העיקרי באלגוריתם למציאת האיבר, הוא שבעוד במיון המהיר אנחנו ממשיכים את המיון על שני חלקי המערך, באלגוריתם שלנו אנחנו מפעילים את המשך המיון רק על החלק שאנחנו מעריכים שהאיבר הדרוש לנו נמצא בו. מה שכמובן מוריד את זמן הריצה של האלגוריתם.

מבחינת הזמנים – מאחר ואת איבר הציר אנחנו בוחרים באופן אקראי, יש לחלק את המקרים של הריצה למקרים טובים וגרועים.

במקרה הטוב – אנחנו בוחרים בכל פעם את החציון של מערך האיברים כך שבסופו של דבר אם נסכום את הטור של חצאי-החצאים אם המערך השלים נקבל טור ששואף ל-2n. כמובן שבאופן רשמי נגדיר את זמן הריצה כ $\Theta(n)$.

במקרה הגרוע – המערך יתחלק בכל פעם לשני חלקים – האחד איבר בודד, והשני יהיה n-1. כמובן שבמקרה כזה אנחנו בכל פעם נעבור כמעט על כל האיברים n פעמים. מה שיגרור זמן הריצה להיות $\Theta(n^2)$.

במקרה הממוצע- חלוקת האיברים מתבצעת בצורה אקראית בין כל האפשרויות:

איבר הציר	**הסתברות**	**חלק עליון** (הכולל את איבר הציר)	**חלק תחתון**
ראשון	1/n	n-1	1
שני	1/n	n-1	1
שלישי	1/n	n-2	2
...	...	...	...
n	1/n	1	n-1

כאשר עושים ממוצע של כל האפשרויות מגיעים בסופו של דבר למסקנה שזמן הריצה במקרה הממוצע מגיע גם הוא ל O(n).

בגדול, זמן הריצה של האלגוריתם הנוכחי הוא לא רע – במקרה הטוב או הממוצע מגיעים ל-O(n), אך אנחנו שואפים להגיע שגם במקרה הרע נהיה תחת אותו זמן ריצה.

אלגוריתם דטרמיניסטי

(חציון = הממוצע של המספר בתחום)

כאמור, הבעיה באלגוריתם הPartition, היא שבמקרה הגרוע ביותר האיבר הנבחר עלול להיות בקצה, מה שמעלה משמעותית את זמן הריצה של הפונקציה להיות n^2.

האלגוריתם הדטרמניסטי[22] מבטיח את החלוקה של הPartittion בצורה חכמה, על ידי בחירה מתאימה של איבר הציר. מציאת החציון המתאים ובחירתו כאיבר ציר. למעשה, מדובר בדיוק על אותו אלגוריתם שראינו קודם שפועל לפי המיון המהיר, אך ההבדל הגדול הוא בצורה בה בוחרים את איברי הציר.

סדר פעולות של אלגוריתם דטרמיניסטי:

1. חלוקת הקלט מהמערך A למחרוזות באורך 5 תווים כל אחת.
2. מציאת החציון של כל חמישיה, והכנסת כל איברי החציון למערך נפרד B (שבמובן גדלו יהיה n/5).
3. הפעלה רקורסיבית של האלגוריתם מציאת איבר החציון בתוך המערך B.
4. חלוקת המערך המקורי A סביב איבר הציר
5. בחירה של החלק המתאים מתוך המערך.

למשל עבור הדוגמה הבאה – נתון לנו מערך של לא ממוין של 100 מספרים ועלינו למיין אותו. נבדוק את החציון בכל אחת מעמודות:

39	37	35	19	32	20	17	1	40	44	21	3	7	0	5	9	2	11	15	12
79	72	56	43	34	51	25	52	46	49	27	4	33	6	26	10	8	14	16	13
83	81	67	66	65	63	60	58	55	50	47	42	36	31	30	29	28	24	23	17
87	85	76	70	80	64	78	71	57	59	48	54	82	62	41	61	53	38	45	22
91	98	93	77	90	75	84	99	88	74	92	73	97	68	69	89	86	94	95	96

[22] "**אלגוריתם דטרמיניסטי** במדעי המחשב הוא אלגוריתם המתנהג בצורה צפויה וניתנת לניבוי. כלומר, בהינתן קלט מסוים, המכונה עליה רץ האלגוריתם לעולם תבצע את אותם צעדים והפלט הסופי לעולם יהיה אותו פלט." (ויקיפדיה)

בעת נשים לב שעבור המספר 47, אם נחלק את המערך יתקבלו לנו חלקים שווים מימין ומשמאל. נבחר את המספר 47 בתור החציון, ונחלק שוב בכל עמודה את האיברים לפי אילו שגדולים יותר מהחציון ואלו שקטנים ממנו:

12	15	11	2	9	5	0	7	3	21	44	40	1	17	20	32	19	35	37	39
13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

לאחר שחילקנו הכל ניתן לראות שיש עליהם ניתן לומר באופן גורף כי כל האיברים הנמצאים תחת אותו שטח בוודאות גדולים מאיבר החציון או קטנים ממנו:

12	15	11	2	9	5	0	7	3	21	44	40	1	17	20	32	19	35	37	39
13	16	14	8	10	26	6	33	4	27	49	46	52	25	51	34	43	56	72	79
17	23	24	28	29	30	31	36	42	47	50	55	58	60	63	65	66	67	81	83
22	45	38	53	61	41	62	82	54	48	59	57	71	78	64	80	70	76	85	87
96	95	94	86	89	69	68	97	73	92	74	88	99	84	75	90	77	93	98	91

אם נבחן בעת את הקבוצות שנוצרו ואת גודלן, נוכל להגיע למספר מסקנות:

- יש $\lceil n/5 \rceil$ קבוצות, ובכל קבוצה חציון, כך שיש לנו גם $\lceil n/5 \rceil$ חציונים, ש-47 הוא איבר החציון שלהם.
- $\lceil n/5 \rceil * \lceil 1/2 \rceil$ מהחציונים קטנים או שווים לאיבר החציון (47)
- בחצי מהקבוצות המחולקות לחמישה איברים, יש לפחות 3 איברים הגדולים מהחציון, ובחצי השני לפחות 3 הקטנים מהחציון.

אם נחשב בדיוק נראה שיש לנו לפחות n/10 קבוצות (מהם נוריד את הקבוצה של הpivot עצמו, ואת הקבוצה בה יכול להיות פחות מחמישה איברים) שבכל אחת מהן יש לפחות שלוש איברים שקטנים יותר מאיבר הציר שלנו. ובאופן פורמלי יותר:

$$(\lceil n/5 \rceil * \left\lceil \frac{1}{2} \right\rceil - 2) * 3 \geq (3/10)n - 6$$

ובאופן דומה נוכל לומר שמספר האיברים הגדול מאיבר הציר הוא לכל היותר (7/10)n-6.

בעת נוכל לקחת את סדר הפעולות ולהצמיד לכל אחת מהן את זמן הריצה המתאים לו:

1. חלוקת הקלט מהמערך A למחרוזות באורך 5 תווים כל אחת. **O(n)**
2. מציאת החציון של כל חמישיה, והכנסת כל איברי החציון למערך נפרד B (שכמובן גדלו יהיה n/5). **O(n)**
3. הפעלה רקורסיבית של האלגוריתם מציאת איבר החציון בתוך המערך B. **T(n/5) גודל הקלט קטן פה משמעותית מאשר ה-n המקורי ולכן ניתן להשתמש ברקורסיה הזו תחת עלות לא גבוהה ומוגדרת ליניארית**

4. חלוקת המערך המקורי A סביב איבר הציר **O(n)**
5. בחירה של החלק המתאים מתוך המערך. **T(7n/10+6)**

ובסך הכל נקבל: **T(n)≤T(n/5)+T(7n/10+6)+O(n)**,וניתן להוכיח שזמן הריצה של הנוסחה הזאת מתאים ל-O(n), בלי הגבלות של הקלט וחלוקה למקרים טובים או רעים, וביצענו את המוטל.

טבלאות גיבוב

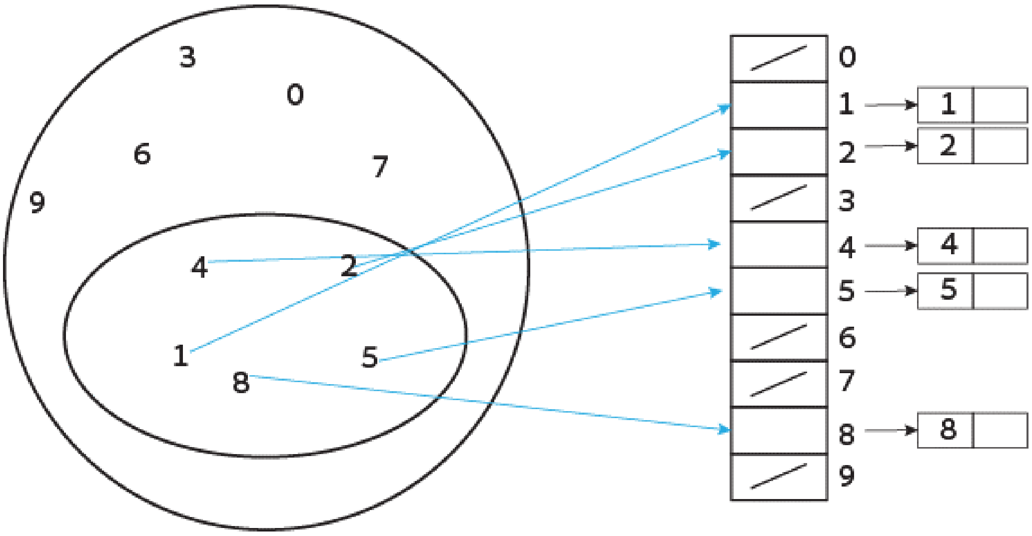
טבלת גיבוב, היא מבנה-נתונים מילוני, הנותן גישה לרשומה באמצעות מפתח מתאים.

נתון לנו מילון, בו כל מילה היא חד-ערכית, ואנחנו רוצים בהינתן המפתח, למצוא את כל המידע הקשור למפתח.

לכאורה זו בעיה שדומה לחיפוש איבר במערך לפי אינדקס, אותו אנחנו מבצעים ב O(1). הפיתוח של האינדקס הנתון במערך, הייתה התקדמות עצומה בגישה לנתונים – Random Access Memory (RAM), אך מציאת מילה במילון הוא שונה במקצת ממציאת איבר לפי המיקום שלו באינדקס. מאחר ותחום הערכים האפשריים של המילים הוא הרבה יותר גדול – אם ניקח מילה בת 15 אותיות, מרחב האפשרויות שלנו ליצירת מילים הוא 22^{15} ולכן איבר אינדקסיאלי הוא הרבה פחות אפשרי לגישה. שלא לדבר על כך שמתוך כל השילובים האפשריים חלק גדול מהם בכלל לא רלוונטי וחסר-משמעות.

מיעון ישיר

בתחילה חיפשו שיטה במיעון ישיר, האינדקס מוביל לתוך תא, ובו יישמר הערך. במקרה כזה כל הכנסה או הוצאה יהיו O(1), אך אם יהיה לנו מספר גדול של אובייקטים שאנחנו לא משתמשים בבולם, אנחנו נבזבז הרבה מקום בזיכרון, למפתחות לא מנוצלים.



בגלל בעיית הזיכרון הלא-ממומש, שיטת המיעון הישיר מתאימה לכמות מידע לא גדולה שניתן לנצלה בצורה אופטימלית וללא בזבוז.

מיעון מחושב (גיבוב)

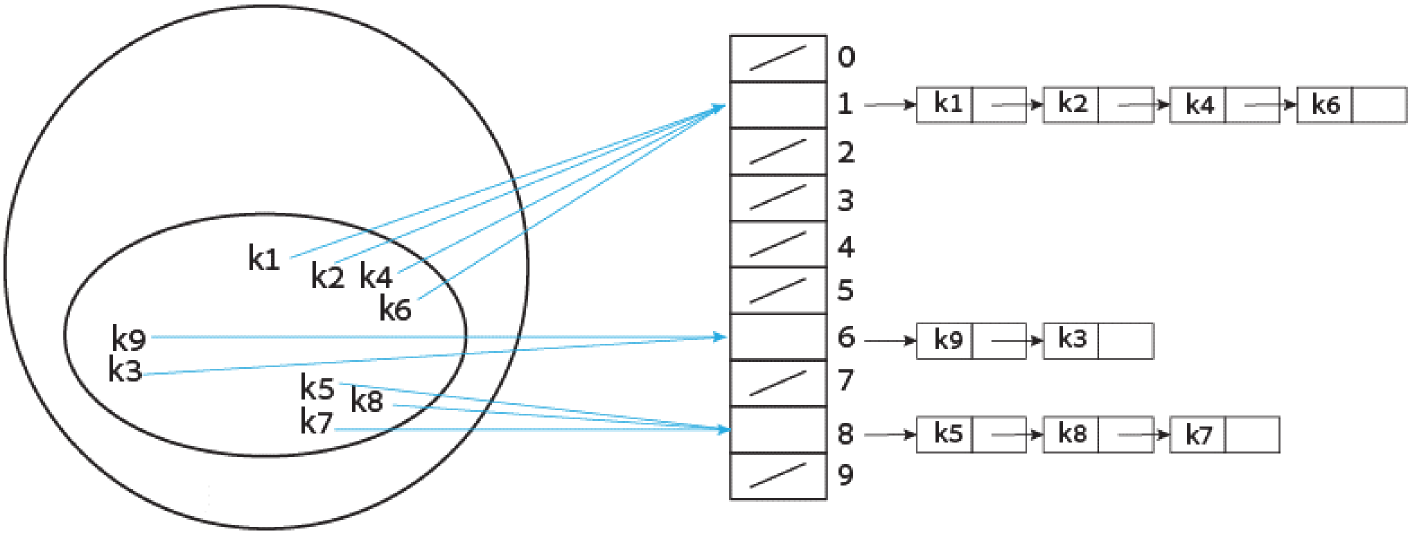
על מנת לחסוך את הבעיה של מרחב זיכרון גדול יחסית ללא מימוש, פותחה שיטת ה"גיבוב" או ה"מיעון המחושב".

רעיון השיטה בקצרה: נגדיר מערך מצביעים, בו כל תא לא-ריק יצביע על רשומה, ותא ריק יכיל Null.
כמו כן, נגדיר **פונקציית גיבוב** – אלגוריתם המקבל מפתח (key[x]), ומגדיר את התא המתאים למפתח על פי הערך המתקבל מההכנסה לפונקציה (למשל mod 10 -מיון כל האיברים ל10 תאים ממוספרים 0-9) ובכל שהשיטה של המיעון יותר מתוחכמת מקטינים את ההסתברות לקבל שני תאים עבור אותו ערך. למה הכוונה? אם ניקח את הפונקציה של mod 10 ניתקל בבעיה שלא משנה כמה המספר גדול ומרובה

מספרים, בסופו של דבר הספרה היחידה שתקבע לאן ייכנס הערך תהיה הספרה הימנית ביותר שלו. באופן דומה אם ניקח עבור ערך בינארי מודולו 2 ניתקל באותה בעיה, ולכן עלינו למצוא פונקציה שתביא פיזור כמה שיותר גדול.

יש לציין, שעד כמה שאנו ננסה להגדיל את הפיזור, עדיין ייתכן מצב ששני איברים יגיעו לאותו מקום, ולכן על מנת לטפל בהתנגשויות – בכל תא נגדיר רשימה מקושרת, כך שבכל התנגשות שתהיה עבור תאים, המפתח ייכנס לתוך הרשימה. כמובן שעל מנת לשמור על זמן הכנסה של O(1), כל איבר חדש ייכנס ישירות לתחילת הרשימה.

כאשר הבעיה שנוצרת היא שבעת חיפוש/מחיקה של ערך, אנחנו נקבל במקרה הגרוע ביותר Θ(list.length), מאחר ויכול להיות שנעבור על כל הרשימה בשביל לקבל את הערך המבוקש.



הנחת הגיבוב האחיד הפשוט

על מנת שהגיבוב יתבצע בצורה טובה עלינו למצוא פונקציות גיבוב טובה, כזו שנוכל לסמוך על כך שהאיברים יתפזרו בצורה כמה שיותר אחידה. ההגדרה האופטימלית לפונקצייה כזו, היא שלכל שני מפתחות a,b השונים זה מזה, ההסתברות שיצא ערך שווה לאחר הגיבוב בין שני הערכים הוא P(h(a))=P(h(b))=1/m

מקדם העומס

מקדם העומס מסומן על ידי האות היוונית α (אלפא), ומבטא את הצפיפות היחסית בתוך הטבלה.

על מנת להשיג את מקדם העומס, נגדיר את n להיות מספר האיברים במערך אותו אנחנו רוצים להכניס לפונקציית הגיבוב, ואת m לגודל הטבלה אליה מכניסים. מקדם העומס (Load factor) מוגדר על ידי $\alpha=\frac{n}{m}$.
יש לזכור שאנחנו מחפשים דרך לצמצם את מרחב ההתנגשויות ולא לבטל אותן לגמרי, מאחר וברגע שגודל הטבלה קטן יותר ממספר האיברים, חייב להיות התנגשויות בסופו של דבר, וככל שמקדם העומס יהיה גדול יותר כך הסיכוי להתנגשויות גדל.

זמן ממוצע של חיפוש

בהנחה ש α הוא האורך ממוצע של כל תא, חיפשו מוצלח יתבצע בזמן ממוצע של $\Theta(1+\alpha/2)$ וזמן חיפוש כושל יהיה $\Theta(1+\alpha)$. הסיבה שאנחנו לא מורידים את הקבוע 1 מתוך הנוסחא, הוא בגלל שאנחנו בן מחפשים אורך זמן שהוא לא אינסופי.

הוכחה לחיפוש כושל היא די טריוויאלית - כאשר יש לנוα איברים ברשימה (הממוצע על פי מקדם העומס), עלינו לעבור על כולה על מנת לבדוק האם האיבר נמצא בתוך הרשימה או לא, ואם לא נמצא ניאלץ לעבור על כל הרשימה +1 של הגעה לתוך הרשימה הנכונה.

הוכחת זמן חיפוש מוצלח – $\Theta(1+\alpha/2)$

- נחפש את האיבר ה-i שהוכנס לרשימה. כאשר נגיע לרשימה נזכור שיש n-i איברים שנכנסו לאחר האיבר אותו אנחנו מחפשים.
- תחת הנחת הגיבוב האחיד, מתוך כל האיברים שהוכנסו לאחר האיבר ה-i, $\frac{n-i}{m}$ איברים הוכנסו לאותה רשימה, וכמובן שלפני האיבר הנדרש.
- מכאן שזמן החיפוש ברשימה הוא $1+\frac{n-i}{m}$.
- את זה נמצע תחת כל האפשרות של i. ונקבל:

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{n-i}{m}\right)$$
$$=1+\frac{1}{nm}\sum_{i=1}^{n}(n-i)=1+\frac{1}{nm}\left(\sum_{i=1}^{n}n-\sum_{i=1}^{n}i\right)$$
$$=1+\frac{1}{nm}\left(n^2-\frac{n(n+1)}{2}\right)=1+\frac{n-1}{2m}$$

מסקנה: אם ניקח לטבלה מספר שהוא בסדר גודל של כל האיברים שאמורים להיכנס אליו, אנחנו נקבל טבלה עם זמן חיפוש קבוע. למשל: אם מפתחות נלקחים מתוך $U=\{1,...10^6\}$, ומספר האיברים בפועל הוא $n=2100$, אזי אם נבחר להשתמש בטבלה בגודל $m=700$, נקבל שאורך כל רשימה הוא בממוצע 3, וזמני החיפוש יהיו בהתאם.

פונקציות גיבוב

המיעון הפתוח הוא שיטת פיתרון נוספת לטיפול בהתנגשויות, כאשר בשונה משיטת השרשור, כאן מקדם העומס יהיה קטן מ1, הווה אומר, המערך יהיה תמיד גדול יותר ממספר האיברים הקיימים.

למעשה, המיעון הפתוח, אינו שיטה אחת אלא אוסף של שיטות העוזרות לגבב את הערכים. נציין מספר שיטות, כאשר למעשה, השיטות בהם נוקטים הם שילובים של מספר שיטות ביחד. שילוב של השיטות יוצר גיבוב שמתחלק בצורה יותר טובה מאחר ולוקח את היתרונות מכל שיטה.

<u>פונקציית הערך התחתון</u> – כאשר המפתחות מתפלגים בצורה אחידה בין $0\leq k<1$ תתקיים לנו הנחת הגיבוב האחיד הפשוט.

<u>פונקציית החילוק</u> – h(k) = k mod m

עושים חלוקת מודולו (המחזירה שארית) על הערכים , כאשר m מייצג כמובן את כוגל הטבלה איה אנחנו מכניסים את המפתחות, ונבחר את התא על פי התוצאה המתקבלת. כמו שכבר צויין, אם יש לנו הרבה

מספרים עגולים ונבחר מודולו 10, אנחנו לא נגיע רחוק. ובאופן דומה עבור מספרים בינאריים, אם נחלק ב2^p, נקבל ערכים לפי הLSB מה שיהיה בוודאות הרבה פחות יעיל.

מסיבות אלו נהוג לבחור מספר ראשוני רחוק בכל האפשר מחזקה של 2 (למשל עבור 2000 ערכים, ניקח את 701), מאחר ומספר ראשוני לא יכיל מחלקים שעלולים לגרום לאיחוד של מספרים (אם נבחר את 6 למשל, הוא כולל בתוכו גם את כל הכפולות שלו, ובן גם את המחלקים שלו 2 ו-3, מה שהופך אותו לפחו אפקטיבי, אך מספר כמו 701, מבטיח לנו 701 אפשרויות שיש פחות סיכוי שתיווצר בהם התנגשות, מה שייתן לנו את אורך הרשימות הרבה יותר ניתן לשליטה – אם יהיה לנו 200 ערכים, יהיה לנו מקסימום 3 איברים שמשורשרים לאותו תא).

__פונקציית הקיפול__ – הרעיון הכללי של השיטה הוא לקחת מספר ולהמיר אותו למספר בינארי, אותו נחלק לשני חלקים, נעשה ביניהם את פעולת הXOR, ואת הערך נכניס בתוך האיבר שמספר כפי הערך הסופי. מספר הפעולות הגבוה יחסית, והבלתי-ניתן לצפייה נותן את היכולת לערבול מוצלח. (לא ניתן לדעת בוודאות מראש לאן ייפול המספר.הערך).

לדוגמא, עלינו להכניס לטבלה את המספר 2966.

נמיר אותו למספר בינארי – 101110010110

נחלק את המספר לשני חלקים ונעשה ביניהם XOR – 010110

__101110__

56_{10} = 111000

__פונקציית אמצע הריבוע__ – מעלים בריבוע את ערך המפתח, ממה שתקבל לוקחים מספר מסוים של הספרות האמצעיות בתור המיקום בטבלה. נקודה חשובה לגודל הטבלה היא, שבמידה ואנחנו עובדים על בסיס עשרוני, גודל הטבלה צריך להיות חזקה של עשר, ובן כל בסיס מספרי אחר – גודל הטבלה צריכה להיות חזקת הבסיס המסוים.

בשיטה זו בוחרים דווקא את הספרות האמצעיות שיבריזו על המיקום בטבלה מאחר והם יחסית משתנות הרבה – הספרות הימניות משתנות יחסית מעט (בהתחשב בסוג החזקה וכו') והספרות השמאליות גם הן, צריכות "להתמלא" בכל שהמספר גדול יותר. למשל – עבור המספר 23654. נבחר טבלה בגודל 10,000 (10^4)

23654^2 = 559,511,716 –התא המתאים הוא 95,117

באשר אם היינו בוחרים את הספרות השמאליות יותר, עבור המספר 23655 (559,559,025) ו-23654 (559,511,716) היינו מקבלי את אותו התא (5595).

ואם היינו בוחרים את הספרות הימניות ביותר, 555 (308,025) ו-5555 (30,858,025) היו מגיעים לאותו תא (8025).

במובן שלקיחת האמצע לא בהכרח מביאה לנו תוצאה שהיא תמיד מושלמת, ומעצם הגדרת טבלת הגיבוב אנחנו מוכנים לקבל שרשור מינלי של איברים, אך לקיחת המספרים האמצעיים, מורידה מהאפשרויות מציאה מהירה מידי של חוקיות במספרים עוקבים (ספרות שמאליות ביותר), או דומים מידי (ספרות ימניות ביותר).

<u>פונקציית הכפל</u> – $h(k) = \lfloor m(kA \bmod 1) \rfloor$

ראשית, מגדירים A. כאשר 0<A<1. יש לשים לב, שבשיטה זאת הA אותו אנחנו מחפשים הוא A כמה שפחות רציונלי. בהשוואה לפונקציית החילוק (שאשר לומר שיש בה בחירה של A רציונלי) אם נבחר מספר שהוא "קל" מידי אנחנו נחלק את כל הערכים למספר תאים קבוע – אם A=1/2, כל האפשרויות יהיה סביב שני תאים בלבד. נהוג לקחת עבור הערך A את מספר חיתוך הזהב – 0.618... שהוכח על ידי פרופ' [דונלד קנות](#)', בתור מספר אי-רציונלי אופטימלי עבור גיבוב.

את ה-A המוגדר מכפילים בערך אותו רוצים להכניס לטבלה.

את תוצאת הכפל מחלקים במודולו 1 – לוקחים את החלק שאחרי הנקודה העשרונית.

את החלק הזה מכפילים בm. בדרך כלל בוחרים ערך m המקיים כלל $m=2^p$, שעוזר לממש את הפונקציה ביתר קלות.

מספר התא הוא ערך הרצפה של התוצאה הסופית – עבור 326314.21354894 ניקח את 326314, ושם נכניס את המפתח.

לדוגמה – ערך **<u>k</u>** – 123456 **<u>A</u>** – 0.6810339 **<u>m</u>** – 2^{16} = 16384

$h(k) = \lfloor 16384 * (123456 * 0.6810339\%1) \rfloor$

$h(k) = \lfloor 16384 * (84{,}077.7211584\%1) \rfloor$

$h(k) = \lfloor 16384 * (0.7211584) \rfloor$

$h(k) = \lfloor 11{,}815.459226 \rfloor$

$h(k) = 11{,}815$

<u>מיעון פתוח</u>

בשיטת השרשור, הגדרנו "שרשור". במידה והיה התנגשות בין שני איברים שיגיעו לאותו תא במערך, ניתן פשוט לשרשר אחד אחרי השני, כמובן תחת מגבלה של זמן ריצה קבוע, וכך יכולנו שהמערך יהיה תמיד בגודל של m≥n. אך בשיט המיעון הפתוח, אנחנו יוצאים מנקודה בהאנחנו לא רוצים להמשיך ולשרשר לתוך התאים, אלא כל איבר במערך יכיל איבר אחד בלבד.

כמובן, שמאחר ואין לנו תאים שיכולים להכיל יותר מאיבר אחד, מספר האיברים במערך המגובב חייב להיות m≤n. על מנת להקל על ההבנה, יש לציין, שאנו בעצם כבר למדנו על השיטה הזאת בשיעורי "מבנה המחשב".

עבור קיום השיטה, יש להגדיר את מקדם העומס α≤1. וכן, בשונה מהמיעון הישיר, יכול להיווצר מצב בו הטבלה כבר מלאה.

כאשר אנחנו מקבלים ערך בלשהו להכנסה, מלבד הערך עצמו אנחנו מקבלים גם "מונה" מאופס, ובכל ניסיון הכנסה בושל, אנחנו מעלים את המונה ב-1. מסמנים את הערך תחת k(h,i), כאשר i מסמן את מספר נסיונות ההכנסה. עבור כל סוג שונה של פונקציית הכנסה מקבלים למעשה מספר תמורות אפשריות לכל הכנסה, ואם יש צורך בודקים את כולם על מנת למצוא מקום נאות.

נציין מספר שיטות הכנסה למיעון הפתוח.

<u>סריקה ליניארית</u> – $h(k,i)=(h'(k)+i) \bmod m$

ההכנסה נעשית תחת פונקצית חילוק במודולו m, ובמידה שהמקום המיועד מלא, עוברים פשוט למיקום הבא במערך. כמובן שיכול להיווצר מצב, בו לאחר ניסיון הכנסה כושל, נעביר את הערך איבר הבא ברשימה, מה שיחסום את אותו איבר מלקבל את המידע אליו הוא מיועד על פי פונקציית המודולו. אך גם פה, נמשיך באותו אופן ופשוט נמלא את התוא הבא במערך.

לדוגמא, עבור המספרים הבאים : 57,12,37,19,17,62,53, (משמאל לימין) ותחת מערך בגודל של m=10, יתקיים הסדר הבא:

9	8	7	6	5	4	3	2	1	0
		57					12		

עבור 37 נקבל גם כן את תא 7, אך מאחר והוא כבר תפוס, נשנה את i לו 1 ונעביר אותו לתא 8, ונמשיך

9	8	7	6	5	4	3	2	1	0
19	37	57					12		

17 ימשיך לנדוד עד לתא 1 (i=3)

9	8	7	6	5	4	3	2	1	0
19	37	57					12		17

62 יעבור לתא 3, ו-53 לתא 4

9	8	7	6	5	4	3	2	1	0
19	37	57			53	62	12		17

כמובן שהיינו יכולים להמשיך ולהכניס עוד 3 מספרים, שבהסתברות די גבוהה לא היו נופלים במקו ם הנכון להם מבחינת פונקציית החילוק, ואז הטבלה היתה מתמלאת לגמרי.

מבחינת ההכנסה והחיפוש, האלגוריתמים די פשוטים הנה, ועובדים באופן דומה – מגיעים לתא הדרוש, אם הוא מלא סובמקרה של חיפוש – זה לא האיבר הדרוש), ממשיכים הלאה לתא הבא.

מבחינת מחיקה אנחנו נתקלים בבעיה, אם נמחק לדוגמא את הערך "12" שהכנסנו לפי הסדר למעלה:

9	8	7	6	5	4	3	2	1	0
19	37	57			53	62	12		17

עלול להיווצר לנו מצב בעייתי – הערך 112 הוכנס למקומו הנכון, אך 62 שגם הוא מקיים את מודולו 10, נמצא בתא שלאחריו. ואם נחפש למחוק את האיבר 62 (או אפילו אם סתם נחפש בשביל לבדוק אם הוא קיים), תוחזר לנו הודעה שהמספר לא קיים במערך.

על מנת לפתור מצב זה, הוחלט שהאיברים שנמחקים מהמערך לא יסומנו ב-Null, אלא ב-Delete, כך שאם יגיע לאותו תא אלגוריתם חיפוש כלשהו, הוא יידע שהיה כאן ערך שנמחק, ולא סתם תא ריק. דבר

זה אגב, גורם לכך שזמן הריצה של פונקציות החיפוש לא רוצה בזמן של מציאת איבר 1, אלא תהיה תלויה בבמות האיברים שנכנסו עד כה.

הנחת הגיבוב האחיד בהתאמה לשיטת המיעון הפתוח, גורסת שיש הסתברות שווה לכל אחת מ-m! האפשרויות השונות לסידור איברי המערך. (לכאורה היינו סופרים את האפשרויות השונות כ-n!, כמספר האיברים שאנחנו רוצים להכניס למערך, אך יש לזכור שמאחר וm עלול להיות גדול יותר מ-n, אנחנו מכניסים גם את כל הרווחים האפשריים שייווצרו במערך הסופי בתור "תו").

<u>**בדיקה ליניארית**</u> – h(k,i)=(h'(k)+i)mod m

מקבלים את הערך שרוצים להכניס, ואת ערך i המאופס. בוחרים פונקציית גיבוב רגילה ומסמנים אותה בתור h', הפונקציה הזאת קובעת את המקום הראשוני ממנו אנחנו מתחילים לסרוק את המערך. במידה והפונקציה h' לא מוצאת מקום מיידי, מגדילים את ה-i ובודקים את האיבר הבא ברשימה על פי ה-i עד שמוצאים מקום פנוי. במידה והרשימה מלאה (לצורך העניין m=25, ואנחנו הגענו ל26, פעולת המודולו m נכנסת לפעולה כך שאנחנו משיכים תמיד לחפש בצורה מעגלית בתוך המער עד שאנחנו מוצאים מקום מתאים.

דוגמא: 0.873, 0.23, 0.92, 0.49, 0.24, 0.87, 0.33, 0.75, 0 (משמאל לימין) **M**=10 **h'(k)** = $\lfloor km \rfloor$

פונקציית גיבוב h(k,i)= (h'(k)+i)mod m

הערכים הראשונים נכנסים ללא בעיה מיוחדת:

9	8	7	6	5	4	3	2	1	0
0.92	0.873				0.49		0.23		

כאשר מגיע 0.24, הוא אמור לקבל את תא מס, 2 שכבר מאוכלס, ולכן, תחת האילוצים, יעבור לתא 3

9	8	7	6	5	4	3	2	1	0
0.92	0.873				0.49	0.24	0.23		

לאחר מכן הערך 0.87, אמור להיכנס בתא 8, ועובר אחר כך ל9, ובאשר גם שם מלא, הוא נמצא כבר ב10 (I=2) ולכן עובר לתא 0

9	8	7	6	5	4	3	2	1	0
0.92	0.873				0.49	0.24	0.23		0.87

0.33 יעבור במובן לתא 5, ו-0.75 יוכל להישאר במקום המקורי בו הוא צריך להיות

9	8	7	6	5	4	3	2	1	0
0.92	0.873	0.75		0.33	0.49	0.24	0.23		0.87

המספר האחרון 0 , יגדל ב-1 ויעבור לתא 1.

9	8	7	6	5	4	3	2	1	0
0.92	0.873	0.75		0.33	0.49	0.24	0.23		0.87

יתרונות במימוש השיטה הליניארית: קלה למימוש.

חסרונות: הטבלה מתמלאת יחסית במהירות וגורמת לאיברים לנדוד יותר מידי עש שמוצאים את המקום.

כפועל יוצא מהחיסרון הקודם – אפשרויות המימוש הולכות ומצטמצמות בכל רגע עד לכדי m, ולא הפוטנציאל המלא של !m.

<u>בדיקה ריבועית</u> $h(k,i)=(h'(k)+c_1\cdot i+c_2 i^2) \bmod m$

הבדיקה עצמה דומה לבדיקה הליניארית מהסעיף הקודם. לוקחים את הפונקציה המוגדרת h'(k), ומרבבים לפיה, אך כאשר מגיעים לאיבר במערך שכבר תפוס, המעבר לא מתבצע אוטומטית לתא הסמוך, אלא עובר מיני-ערבול קטן למקום החדש. כאשר האיברים c_1 ו-c_2 המוכפלים ב-i, כאשר איברי ה-c לא חייבים להיות מספרים שלמים, ותלויים בגודל הטבלה, כך שכל התמורות ישארו במרחב האפשרויות הקיים.

- בודקים בהתחלה את התא שמקיים h'(k)
- אם הוא מלא, בודקים את $h'k+c_1+c_2$
- אם יש צורך עוברים ל $h'(k)+2c_1+4c_2$
- ובכה ממשיכים עד ששמגיעים למספר מאפס את המודולו ומתחילים מתחילת המערך.

דוגמא: נכניס את האיברים: 0.873, 0.23, 0.92, 0.49, 0.24, 0.87, 0.33, 0.75 (משמאל לימין)

m=8 **c_1**=1/2 **c_2**=1/2 **h'(k)** =$\lfloor km \rfloor$

ארבעת המספרים הראשונים נכנסים בלי בעיה למקום הנכון (כזכרו, בתחילת האלגוריתם l=0, ולכן הקבועים c לא רלוונטיים).

7	6	5	4	3	2	1	0
0.92	0.873			0.49		0.28	

כשמגיעים ל-0.24*8 מתקבל 1.92, ובתא מס' 1 כבר קיים ערך, ולכן מנסים את התא הבא ומכניסים את האיבר בתא 2.

7	6	5	4	3	2	1	0
0.92	0.873			0.49	0.24	0.28	

האיבר הבא – 6.09 = 8*0.87.

בתא 6 כבר קיים ערך, וגם בתא 7 (i=1), כבר קיים, ובודקים עבור (i=2).

$h(0.87,2) = 6.09+(2*0.5)+(2^2*0.5)$

$h(0.87,2)=6+1+2=9 \bmod 8=1$

גם בתא 1 יש ערך קיים ובודקים עבור (i=3)

$h(0.87,2) = 6.09+(3*0.5)+(3^2*0.5)$

$h(0.87,2) = 6.09+(1.5)+(4.5)=12mod8=4$

7	6	5	4	3	2	1	0
0.92	0.873		0.87	0.49	0.24	0.28	

מספר הבא 2.64 = 8*0.33

האינדקס 2 כבר מלא, וגם 3, והדילוג הבא הוא לתא 5 שבו ניתן להכניס (כבר כאן אפשר לראות איזה בעייתיות של סדרתיות יחסית גבוהה בקפיצות)

7	6	5	4	3	2	1	0
0.92	0.873	0.33	0.87	0.49	0.24	0.28	

0.75, במובן ייכנס לתא באינדקס 0. השאלה היא רק תוך כמה איטרציות הוא יגיע לשם.

6=8*0.75

(i=1)=7

(i=2)=9mod8=1

(i=3) 12mod8 = 4

$(i=4)=6+(4*0.5)+(4^2*0.5) = 6+2+8 = 16mod\ 8 = 0$

7	6	5	4	3	2	1	0
0.92	0.873	0.33	0.87	0.49	0.24	0.28	0.75

יתרונות בשיטה הריבועית: קפיצה גדולה יותר במרחק בין תאים מלאים.

חסרונות: סדר הבדיקות הוא קבוע יחסית, כך שאם יצא שכמה מספרים נופלים לאותו תא, המרחק יהיה קבוע והבדיקה הולכת וחוזרת בכל פעם על אותו רצף.

בדומה לבדיקה הליניארית, גם פה יש אפשרות רק ל-m אפשרויות מתוך ה-!m.

<u>גיבוב כפול</u> $h(k,i)=(h_1(k)+i\cdot h_2(k))mod\ m$

בגיבוב זה אנחנו מקבלים **שתי** פונקציות גיבוב, כאשר את השניה אנחנו מכפילים ב-i, ועל ידי זה הגיבוב מתבצע בצורה שהיא הרבה יותר טובה. על מנת שסדרת הבדיקות תהיה יעילה, יש לוודא שגודל המערך (m) יהיה ערך שהוא זר לתוצאה מ$h_2(k)$, ולא יתחלקו אחד בשני, אחרת פונקציית המודולו לא תהיה יעילה מספיק. ביצד ניתן לעשות זאת בצורה ודאית? שתי אופציות – 1. לוודא שאחד יהיה זוגי ואחד אי-זוגי, למשל – לבחור עבור ה-m חזקה של 2 (למשל 2^m), ואת הפונקציה לוודא שתקבל ערך אי-זוגי (למשל h*2+1 ייתן תוצאה שהיא תמיד אי-זוגית). 2. בוחרים ל-m מספר ראשוני, ולוודא ש h_2 תפיק ערבים קטנים מm.

דוגמא: הכנסת המספרים 3, 17, 6, 9, 15, 13, 10 (משמאל לימין)

m=7 $h_2(k) = k \bmod m$ [23]$h_1(k) = 1+k \bmod 4$

איבר ראשון – 3

h(3,0)= [(1+3mod4)+(0*3mod7)]mod7=(4+0)%7 = 4

6	5	4	3	2	1	0
		3				

איבר שני – 17

h(17,0) = [(1+17mod4)+(0*17mod7)]mod7 = (2+0)%7 = 2

6	5	4	3	2	1	0
		3		17		

איבר שלישי – 6

h(6,0)= [(1+6mod4)+(0*6mod7)]mod7=(3+0)%7 = 3

6	5	4	3	2	1	0
		3	6	17		

איבר רביעי –9

h(9,0)= [(1+9mod4)+(0*9mod7)]mod7=(2+0)%7 = 2

h(9,1)= [(1+9mod4)+(1*9mod7)]mod7=(2+2)%7 =4

h(9,2)= [(1+9mod4)+(2*9mod7)]mod7=(2+4)%7 =6

6	5	4	3	2	1	0
9		3	6	17		

איבר חמישי – 15

h(15,0)= [(1+15mod4)+(0*15mod7)]mod7=(4+0)%7 = 4

h(15,1)= [(1+15mod4)+(1*15mod7)]mod7=(4+1)%7 = 5

[23] המצגת בשיעור היתה עם סידור שונה בין הפונקציות עצמן, מה שגם נכון יותר מבחינת ההגדרות שדרשנו (זוגי אי-זוגי)

6	5	4	3	2	1	0
9	15	3	6	17		

איבר שישי – 13

h(13,0)= [(1+13mod4)+(0*13mod7)]mod7=(2+0)%7 = 2

h(13,1)= [(1+13mod4)+(1*13mod7)]mod7=(2+6)%7 = 1

6	5	4	3	2	1	0
9	15	3	6	17	13	

איבר שביעי – 10

h(10,0)= [(1+10mod4)+(0*10mod7)]mod7=(2+0)%7 = 2

h(10,1)= [(1+10mod4)+(1*10mod7)]mod7=(2+3)%7 = 5

h(10,2)= [(1+10mod4)+(2*10mod7)]mod7=(2+6)%7 = 1

h(10,3)= [(1+10mod4)+(3*10mod7)]mod7=(2+9)%7 = 2

h(10,4)= [(1+10mod4)+(4*10mod7)]mod7=(2+12)%7 = 0

6	5	4	3	2	1	0
9	15	3	6	17	13	10

יתרונות השיטה: ניתן לקבל m(m-1) סדרות שונות. עדיין לא עומד בדרישה של !m, אך משמעותית הרבה יותר טוב מהפונקציות הקודמות שסקרנו.

במידה ואנחנו סוקרים את טבלת הגיבוב שלנו ומגלים שיש הרבה התנגשויות בכל כניסה, ניתן להבין כי כנראה יש בעיה בגיבוב. מאחר וכל רעיון הגיבוב שואף למעט את מספר ההתנגשויות, עלינו לבדוק מספר כיוונים על מנת לטפל בבעיה:

- ראשית יש לבדוק האם התפוסה של המערך כבר גדולה–
    - במידה וכן, יש להגדיל את ה–m, במידה ועושים כך, יש לבצע שינוי בפונקצית הגיבוב המקורית סעל מנת תגיע לחלקים יותר רחבים בתוך הטבלה), או לחילופין, למחוק את כל הרשומות בטבלה המוגדרות כמחוקות. כמובן שכדאי לבצע את שני הדרים יחד.
- לאחר מכן יש לבדוק את צורת הפיזור של פונקציות הגיבוב השונות. ולוודא שהם מפזרות את הנתונים בצורה שהיא אחידה או לפחות קרובה לאידה.
- לאחר מכן, יש לוודא שיש שוני משמעותי בין שתי פונקציות הגיבוב – יכול להיות שלא שמנו לב והגדרנו שתי פונקציות שמגיעות לאותם מקומות כל הזמן, ויש לשנות זאת.
- יש לוודא שגודל m מקיים את הכלל של מספר ראשוני הגדול מספיק מהאיברים הדרושים.

<u>סיכום עבור שיטת המיעון הפתוח:</u>

כאשר בודקים סופו של דבר, עד כמה המיעון הפתוח יעיל, מגלים שעבור קלט של ערכים גדולים מגיעים להתנגשויות גבוהות גם בגיבוב הליניארי וגם בגיבוב הכפול, מה שעלול לגרום להאטה משמעותית בזמן החיפוש של איברים.

כאשר נידרש לבחור בין שרשור לבין מיעון פתוח ישנם מספר פרמטרים שיעזרו לנו להחליט איזה אחד מבין השניים מתאים יותר. <u>המיעון הפתוח</u>- מתאים למצבים בהם לא מבצעים מחיקות. אם רוצים להחזיק כמות גדולה של מידע שהבסיס שלה לא אמור להימחק (כמו למשל מידע לפי מס' תעודות זהות) המיעון הפתוח יתאים אך אם מדובר על מידע שהוא יותר דינאמי, אז כמו שנאמר כבר קודם, זמן החיפוש עולה באופן משמעותי. <u>שיטת השרשור</u> – מתמודדת בצורה נאותה גם עם נתונים הצריכים להימחק, אך מסורבלת יותר בגלל כל המצביעים אותם היא צריכה לשמור מה שגורר הרבה זיכרון שצריך לטפל במצביעים.

גיבוב אוניברסלי

אחת הבעיות הנוצרות עם כל פונקציית גיבוב כזו או אחרת, היא שבסופו של דבר יכול להגיע קלט שימוען כולו לאותו תא. דבר זה יכול להיות סתם במקרה, או במקרה הרע יותר, יכול להיות מכוון על מנת להעמיס על המערכת. כמובן שאנחנו רוצים להימנע בכל האפשר ממציאות כזאת, מאחר ואם המידע לא יכול להיות מאוחסן בצורה נאותה, וגישה אליו אחר כך יהיה מאוד מסורבל וארוך, וכל התוכנית תהיה הרבה פחות יעילה ממה שתוכננה להיות. בעיה זאת מכונה "מספרים דביקים".

הפיתרון אותה שיטת הגיבוב האוניברסלי מציגה, הוא פשוט לבחור את פונקציית הגיבוב עצמה בצורה אקראית. בזמן הפעלת היישום, תבחר פונקציה אחת או יותר מבין מערך הפונקציות הקיים לנו, וכל הגישה לנתונים תהיה דרך הפונקציה הזאת. כמובן, שברגע שנבחרה לתוכנית פונקציה, עלינו להמשיך ולגשת אל הנתונים מאותה פונקציה ולא להתחיל להחליף פונקציות בכל פעם. אם נשנה את הפונקציות ללא בקרה לא נוכל להגיע לנתונים שהוכנסו בדרך שהיא שונה ממה שאנחנו משתמשים בה עכשיו.

<u>הגדרה:</u>

יהי H אוסף סופי של פונקציות גיבוב $H=\{h_i:U\rightarrow \{0,\ldots,m-1\} \mid i=1,2,\ldots,t\}$.

H תקרא **אוסף אוניברסלי** אם עבור כל זוג מפתחות שונים זה מזה, $x,y\in U$ מספר פונקציות הגיבוב $h\in H$ שעבורן יש התנגשות, כלומר, $h(x)=h(y)$, הוא בדיוק $|H|/m$.

ובצורה פשוטה יותר – אנחנו מחזיקים אוסף של פונקציות גיבוב. הדרישה מפונקציות הגיבוב היא שאם ניקח את אותם מספרים ונכניס אותם פעמים תחת הזדמנויות שונות, נקבל שיבוץ שונה בתוך המערך, כאשר ההסתברות לכך ששני פונקציות יכניסו את אותם שני ערכים לאותו תא אחרי הגיבוב יהיה 1/m.

כך שאם יהיו לנו d פונקציות ערבול, ולהם יהיו שייכים b פונקציות שמכניסות שני איברים שונים לאותו תא, הדרישה היא שההסתברות תקיים:

$$\frac{b}{d}\le\frac{1}{m}\Rightarrow b\le\frac{d}{m}$$

כאשר יש צורך ש- d(מספר הפונקציות) יהיה לפחות שווה או גדול מגול המערך m, ובן יש לשים לב שגודל ה-b עלול להיות שונה בין כל שני מספרים.

ניקח דוגמא פשוטה עבור קבוצה אוניברסלית:

נגדיר מערך בגודל m=2.

קבוצת המפתחות U={0,1,2}

קבוצת הפונקציות H={h1,h2,h3}

הממענות את הערכים בסדר הבא:

h3	h2	h1	Values
1	0	0	0
0	1	0	1
1	0	0	2

בקבוצה U קיימים זוגות המספרים הבאים: {0,1},{0,2},{1,2}.

עבור כל אחד מהם קיימת רק פונקציה אחת שמתאימה להם את אותו ערך (h1). מאחר ויש לנו 3 פונקציות, ההסתברות להתנגשות תהיה 1/3>1/m. ולכן הקבוצה של הפונקציות H מהווה קבוצה אוניברסלית.

<u>בניית פונקציית גיבוב אוניברסלית</u>

פונקציית הגיבוב האוניברסלית עליה אנחנו מדברים, מתייחסת לנוסחה הבאה:

$h_{a,b}(k)=((ak+b)(modp))(modm)$

כאשר אם ניקח את קבוצת הערכים U, ה-m הוא קטן ממש בקנה מידה מאשר |U| (לדוגמה - אם $|U|=n^2$, אז m=n), ו-p הוא מספר ראשוני המקיים $p\geq|U|$.

כמו בן, נגדיר 2 קבוצות של מספרים טבעיים, $Z_s=\{0,1,2....s-1\}$, $Z^*_s=\{1,2,3...s-1\}$. ועבורם נגדיר את $a \in Z_S^*$

$b \in Z_S$ הנגזרות בגודלן ממספר האפשרויות של p. בכל פעם שיבקשו מאיתנו לבצע גיבוע אוניברסלי, יביאו לנו את הנתונים המתאימים עליהם נבצע גיבוב ואנו נצטרך לשבץ בפונקציית את כל הערכים על פי הדרישה.

לדוגמה -

נגדיר את p=17, m=6, ונחשב את $h_{3,4}(8)$

$H_{3,4}(8)=((3*8+4)mod17)mod6$

$h_{3,4}(8)=((28)mod17)mod6$

$h_{3,4}(8)=11mod6$

$h_{3,4}(8) = 5$

אם ננתח את האפשרויות העומדות בפנינו, נוכל לומר כי מאחר ויש לנו p אפשרויות עבור b, ו p-1 עבור a, מגדיל את סדר הגודל משמעותית עבור הפונקציות המתקבלות. למשל אם נרצה לגבב מספרי תעודות זהות, יש לנו עבור קבוצת המספרים האפשרי 10^8 (בהנחה שהספרה הראשונה היא שונה מ-0), אז קבוצת הפונקציות האפשריות תהיה בסדר גודל 10^{16}.

מאחר ויש לנו טווח אפשרויות רחב מאוד לבניית הפונקציה, ואנחנו מדברים על מודולו במספר ראשוני, שנותן הסתברות לכך שיהיו פחות אפשרויות להתנגשות, ויתקיים שההסתברות היא 1/m. יש לזה הוכחה ארוכה, ולא רלוונטית למבחן.

גרפים

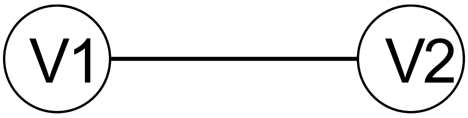
"בתורת הגרפים, גרף הוא ייצוג מופשט של קבוצה של אובייקטים, כאשר כל תת-קבוצה של אובייקטים בקבוצה עשויים להיות מקושרים זה לזה" (ויקיפדיה).
גרף הוא מושג שמבטא את הקשר בין דברים. "דברים", לצורך ההגדרה, הם אנשים, תופעות שונות וכל דבר אחר בזיקה בלשהי של בין אחד לשני. הרעיון של גרף עוזר לנו לעבוד עם התופעות האלה ולנצל את הקשרים הקיימים ביניהם על מנת לנתח מידע הקשור באובייקטים. "קשרים" זה כל דבר שיש לנו בעולם – קשר סיבתי, לוגי, פיזי, מסלולים בתחבורה, בחיים ועוד. כך שאי אפשר להמעיט מערך הגרף.

הגדרות רשמיות

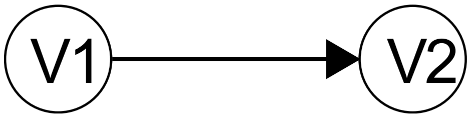
הסימון הרשמי של גרף הוא G=(V,E), כאשר האות G מייצגת את הגרף, והערכים שבסוגריים נותנים לנו מושג לגבי המבנה הכללי שלו:

<u>V – קבוצת קדקודים (Vertex)</u> – קדקוד הוא אובייקט בלשהו המייצג כל דבר שאנו רוצים להתייחס אליו – אדם, עצם או כל דבר הניתן ל"קשירה". בכל אובייקט שבזה יכול להיות המון מידע התלוי בסוג האובייקט שלו ומה שהוא מייצג, אך הרלוונטי לנו, בכל קדקוד יש שדה זיהוי המייצג מה ה"שם" שלו ביחס לגרף. הקדקודים מסומנים בדרך כלל בצורת עיגולים עם השם שלהם בתוכם. סימן מקובל לקדקודים יכול להופיע בתור אותיות או מספרים, לפי בחירת המיון. וכן שילוב של השניים (כמו בדוגמה הסמוכה) כאשר מספר הקדקודים מסומנים ב-n, והספירה שלהם מתבצעת מ1-n.

<u>E – קבוצת הקשתות (Edge) –</u> קבוצת זוגות של קדקודים (מכונים גם "צלעות"/"קשתות") כאשר E מייצג את החיבור בין שניהם. כל קשת מסומנת בתור הקדקודים ביניהם היא מקשרת, בדוגמא לפנינו, הקשת היא (v1,v2). מספר הקשתות בגרף הוא E≤(v*v)



<u>קשת לא מכוונת</u> – כאשר הקשת בין הקדקודים מופיע ללא חץ המציין מאיפה ולאן הוא הולך (מכונה גם: זוג בלתי סדור) המשמעות היא ש (v1,v2)=(v2,v1) וניתן לעבור ממקום למקום בחופשיות.

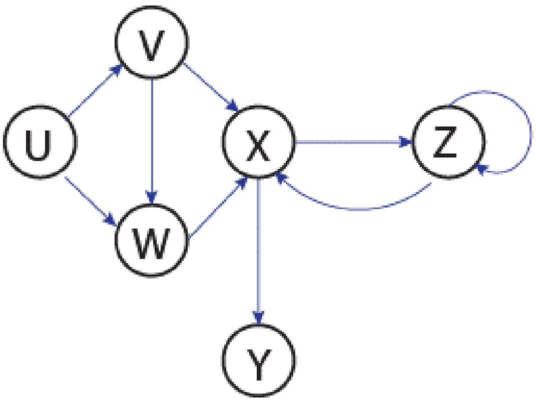


<u>קשת מכוונת</u> – מצוין על ידי חץ מקדקוד אחד לשני (זוג סדור), במקרה בזה, הכיוון הוא רק לכיוון אליו פונה החץ במקרה זה (v1,v2)≠(v2,v1)

<u>גרף מכוון</u> – גרף בעל צלעות מכוונות. כאשר ברגע שיש לנו אפילו קשת אחת שהיא מכוונת הוא הופך את כל הגרף לגרף מכוון, אפילו אם כל שאר הקשתות בגרף הן לא-מכוונות – אנחנו מתייחסים לקשתות הלא-מכוונות בשתי קשתות מכוונות הצמודות אחת לשניה.

<u>גרף לא מכוון</u> – גרף ש**כל** צלעותיו לא מכוונים. למעשה, גרף לא מכוון הוא למעשה מקרה פרטי של גרף מכוון שכל הדרכים בו הם דו סטריות.

יכול להיות גם שיהיו קדקודים שלא מחוברים לשום מקום והם לא משפיעים על הגדרת הגרף במכוון/לא-מכוון.



במידה ולא אומרים לנו מה הסדר שאנחנו עובדים לפיו, ישנה מוסכמה ללכת לפי סדר לקסיקוגרפי. כך שאת הגרף הבא נגדיר:

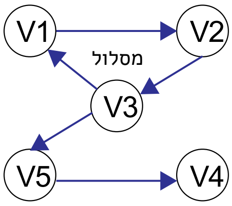
E={(u,v)(u,w)(v,w)(v,x)(w,x)(w,y)(x,y)(x,z)(z,x)(z,z)}

|v|=6 (מספר קדקודים בגרף)

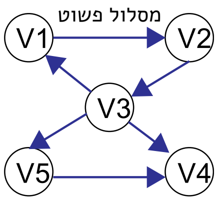
|E| = 10 (מספר קשתות בגרף)

דרגה – לכל קדקוד, יש בנוסף הגדרה של **דרגת כניסה ויציאה**, המציינים את מספר הקשתות הנכנסות אל הקדקוד או יוצאות ממנו. למשל: קדקוד X. דרגת כניסה 3. דרגת יציאה 2.
כמובן שדרגות אלו חלות רק בגרף מכוון. בגרף לא מכוון מדברים רק על **דרגה**, מאחר וכל דרגת כניסה היא גם יציאה.

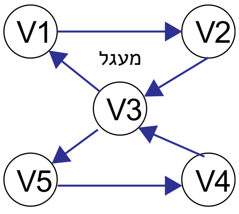
מסלול – סדרה של קדקודים בו כל קדקוד מחובר לזה שאחריו, כאשר אורך המסלול נגזר ממספר הקשתות במסלול. יכול להיות שקדקוד יופיע פעמיים באמצע המסלול



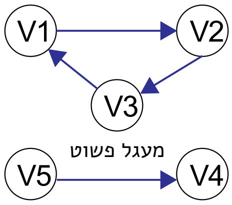
מסלול פשוט – מסלול שבכל קדקוד מופיע בו רק עם אחת.



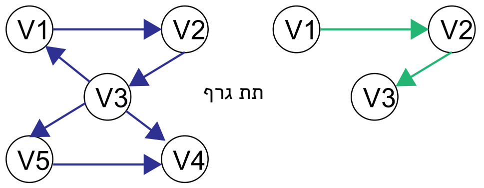
מעגל – מסלול שנקודת ההתחלה והסיום שלו שווים, אך עלול לעבור באחת הנקודות יותר מפעם אחת.



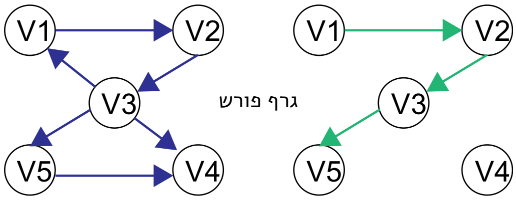
מעגל פשוט – מקרה פרטי של מעגל, בו מלבד הקדקוד שמוגדר בראשון/אחרון, אין חזרה על אף קדקוד.



<u>תת-גרף –</u> חלק מתוך גרף המוכל בתוך הגרף השלם. יכול להכיל חלק מהמסלול ומספר קדקודים (גם בנפרד. כמובן שההגדרה של גרף מכוון/לא מכוון שייך גם לתת הגרף שלו.



<u>גרף פורש –</u> תת-גרף ש"תפרש" על כל קדקודי הגרף המלא, אף אם לא בכל הקשתות



<u>רכיבים קשירים</u> –גרף נקרא "קשיר" רק אם יש מסלול בין כל שני קדקודים בG

<u>רכיב קשיר</u> – הוא תת גרף קשיר מקסימלי, אם ניתן להוסיף קדקוד מהגרף המקורי והתת גרף נשאר עדיין קשיר. אך אם הקדקוד המתווסף הופך את תת הגרף להיות לא קשיר, אזי מדובר כבר ברכיב קשיר.

ההגדרה של קשירות שייכת רק בגרף לא מכוון, כאשר בגרף מכוון, מדברים על "**רכיב קשיר חזק**". כאשר הקשירות החזקה מתבוונת לקשירות דו כיוונית בי כל שני קדקודים.

<u>יער</u> – גרף (לא מכוון) שאינו מכיל מעגלים

<u>עץ –</u> גרף (לא מכוון) קשיר שאינו מכיל מעגלים. למעשה עץ הוא מקרה פרטי של יער.

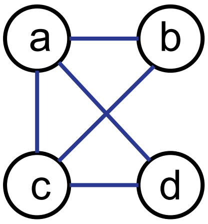
הגדרות שקולות לעץ: (כל ההגדרות האחרות אומרות אותו דבר אך משתמשים בהגדרות אחרות)

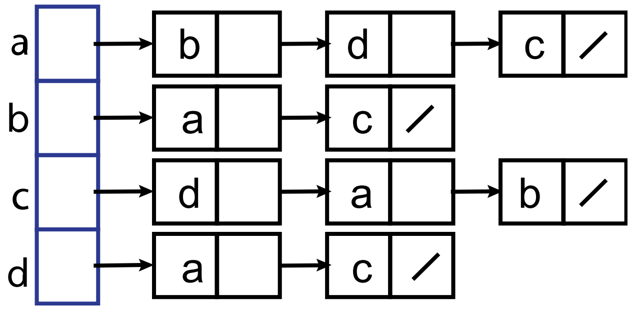
גרף G=(G,E) הוא עץ אם ורק אם:

- G קשיר ללא מעגלים
- G ללא מעגלים, ו |E|=|V|-1 (מספר הקדקודים גדול ב1 ממספר הצלעות)
- G קשיר ו |E|=|V|-1
- G קשיר והוצאת קשת אחת כלשהי מG גורמת לG להיות לא קשיר
- G אינו מכיל מעגל, והוספת קשת אחת כלשהי בG גורמת לG להכיל מעגל

ייצוגים של גרף

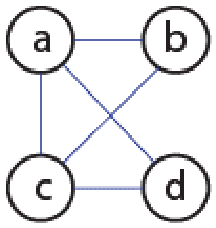
רשימת סמיכויות – רשימה של רשימות כאשר כל קדקוד מכיל מערך המכילה את הקדקוד אליהם ניתן לעבור. כאשר סדר הסמיכויות לא חשוב.





מטריצת סמיכויות – מטריצה של שורות ועמודות המכילות תוצאה של 0 ו-1 המסמנים בין כל שני קדקודים האם יש ביניהם חיבור כלשהו. כאשר המספר הוא בסדר לקסיקוגראפי.

	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0



איזה מבין הייצוגים מוצלח יותר?

ייצוג	מקום בזיכרון	זמן הוספת צלע	זמן חיפוש צלע	זמן מחיקת צלע
רשימת סמיכויות	Θ(V+E) בגרף לא מכוון כל קשת בעצם נספרת פעמיים גם בהלוך וגם בחזור. בגרף מכוון המקום בזיכרון יהיה בדיוק מספר הקשתות לפי הכיוונים שלהם	Θ(1) הוספת הקדקוד המקושר החדש בהתחלה	O(V) הזמן המקסימלי הוא כאשר אותו קדקוד מחובר לכל הקדקודים (גרף צפוף) וצריך לעבור את כולם על מנת לדעת אם יש צלע בין שני קדקודים.	Θ(V) ישש לגשת לקדקוד הראשון מסין השניים O(1), ואז למצוא את היבר אותו רוצים למחוק – זמן חיפוש O(V).
מטריצה	$\Theta(N^2)$ מאחר וצריך ייצוג מלא גם לקשתות הלא-קיימות בין כל הקדקודים הקיימים בגרף כאשר מדובר על גרף צפוף (כל קדקוד מחובר לכל האחרים)	Θ(1) שינוי 0 ל-1	Θ(1) גישה לנקודה והחזרה של תשובה.	Θ(1) שינוי מ1 ל0

חיפוש לרוחב

בדומה להתפשטות של אש בשדה קוצים. התהליך עובד בבמה מישורים – האחד – האזור שעוד לא נשרף – החלק עוד לא ביסינו בחיפוש. אזור האש – אזור שעוברים עליו באותו רגע (יכול להיות רחב). והאזור שכבר נסרק/נשרף.

את קדקוד המקור נסמן באות S (source)

בחיפוש לרוחב מוסיפים בכל קדקוד שני שדות חדשים – D (מרחק- distance) מציין את מספר הקשתות המינימלי מקדקוד המקור לאותו קדקוד במיקומו הנוכחי. במידה ולא ניתן להגיע לאותו קדקוד מסמנים אותו באינסוף – בעצם מגדירים בבריחת מחדל את כל הקדקודים כאינסוף, ובמידה שמגיעים אליו משנים את הD לערך הנכון.

הערך השני הוא π (פאי) המוגדר כמצביע לקדקוד הקודם לנוכחי במסלול הקצר ביותר מהמקור. אם רוצים למצוא את המסלול, ניתן להשתמש בפונקציה רקורסיבית שמגיעה עד סוף המסלול, ואז מחזירים את הדרך.

איחוד הקדקודים במסלולים הקצרים ביותר ייתן לנו תת גרף שהוא עץ, מאחר ומדובר על גרף שהוא (רכיב) קשיר- כולם מגיעים לS, ואותו העץ נקרא "**עץ רוחב**". במידה והיינו יכולים להוסיף קדקודים, היינו כבר מוסיפים אותם במהלך החיפוש לרוחב.

קוד חיפוש לרוחב – BFS(G,s)

```
1.   for each vertex u in V[G] – {s}
2          do color[u] ← white
3             d[u] ← ∝
4             π[u] ← NULL
5    color[s] ← gray
6    d[s] ← 0
7    π[s] ← NULL
8    Q ← Φ
9    enqueue(Q,s)
10   while Q ≠ Φ
11         do u ← dequeue(Q)
12               for each v in Adj[u]
13                     do if color[v] = white
14                           then color[v] ← gray
15                                d[v] ← d[u] + 1
16                                π[v] ← u
17                                enqueue(Q,v)
18               color[u] ← black
```

בנוסף, כל קדקוד יאותחל בשדה "צבע" – כאשר הצבע הראשוני יהיה לבן, במהלך הטיפול הוא יהיה צבע אפור, ולאחר שעברו עליו הוא יהיה בצבע חור.

הקוד עובר בהתחלה על כל התאים ומגדיר בהם את שלשת השדות החשובים – "צבע", D, π בערכי ברירת המחדל שלהם (שורות 1-4), ואז מכניסים את הבסיס למרחק 0 ומתחילים לטפל בבל האיברים (שורות 5-7).

מכניסים לתור את הקדקוד הראשון (שורה 8-9), ועוברים על כל הקדקודים ברשימת הסמיכות של הקדקוד שהוצא מהתור, על פי רשימת הסמיכויות שלו (שורות 10-11). מוודאים שהקדקוד של השכן הוא בצבע לבן – לא נגעו בו עדיין – ובמידה שהוא אבן חדש, צובעים אותו באפור, ומעדכנים את שדה המרחק ואת האב שלו להיות הקדקוד המקורי בו אנחנו מטפלים (שורות 12-17). וברגע שמסיימים לעבור על כל הסמיכויות של הקדקוד ניתן לצבוע אותו בשחור (שורה 18)

נכונות BFS **(למציאת מסלולים קצרים ביותר)**

נסמם ב"דלתא" את אורך המסלול הקצר ביותר בים שני קדקודים ונחפש האם הטענה שחיפוש לרוחב הוא הקצר ביותר. בשביל נשתמש בשלוש "למות" (טענות).

למה 1: $\forall(u,v)\in E, \delta(s,v) \le \delta(s,u) + 1$

לכל מסלול בין שני קדקודים, המסלול בין השורש לקדקוד הראשון, קטן או שווה למרחק בין השורש לקדקוד השני+1.

למה 2: בסיום BFS, מתקיים $\forall v\in V, d[v] \ge \delta(s,v)$.

בסיום הסריקה, מתקיים כי המרחק מהשורש עד לקדקוד v, גדול או שווה למסלול בין שני קדקודים סמוכים אחרים. בשדה d, מוגדר המרחק הסופי של הקדקוד מהשורש, מאחר וכבר סיימנו את הסריקה, ולכן ניתן להתייחס לנתונים הקיימים פ בנתונים סופיים עליהם ניתן לסמוך.

למה 3: אם בתור Q נמצאים הקדקודים $<v_1, .. v_r>$, כאשר v_1 ו-v_r הם הראש והזנב של Q, בהתאמה, אזי:

- $d[v_r] \le d[v_1] + 1$
- $\forall\ i=1,2,..,r-1,\ d[v_i] \le d[v_{i+1}]$

אם בזמן הסריקה הקדקודים נמצאים בתוך התור ומחכים לטיפול, ההפרש בין הקדקוד הראשון (הקרוב ביותר למקור), ובין הקדקוד הרחוק ביותר מהמקור(האחרון), יהיה לפחות אחד.

כמובן, הסדרה של התור היא סדרה עולה מונוטונית, כך שכל קדקוד גדול או שווה לזה שלפניו. וזאת מאחר, ששדה ה-d אף פעם לא נהיה קטן יותר אלא רק גדול יותר.

מסקנה: אם במהלך BFS קדקוד v_i הוכנס לתור לפני קדקוד v_j, אזי כאשר מוכנס v_j, מתקיים $d[v_i] \le d[v_j]$.

משפט:

1. במהלך הרצתו, BFS מבקר בכל קדקוד $v\in V$ שניתן להגעה מ-s. הקדקודים שלא ניתן להגיע אליהם מהמקור, גם לא מגיעים איהם בסריקה.
2. בסיום הרצת BFS, $d[v] = \delta(s,v)$. בשדה d של כל קדקוד נמצא המרחק הקצר ביותר מהמקור
3. לכל קדקוד v הניתן להגעה מ-s, מלבד s עצמו, אחד המסלולים הקצרים ביותר מ-s ל-v הוא המסלול הקצר ביותר מ-s ל-$\pi[v]$, ואחריו הצלע $(\pi[v],v)$. השדות פאי של כל קדוקד נותנים לנו את המסלולים הקצרים ביור מ המקור לכל קדקוד. המסלול הקצר ביותר תמיד יכול להיות לא יחד – יכול להיות שמ-s לקדקוד כל שהו V יכול להיות מסלול בארוך מסוים, אך אין מן הנמנע שיהיו שני מסלולים באותו אורך מינימלי שמגיעים מהמקור ועד הקדקוד, אך כמובן שאין מסלול נוסף שיהיה קצר יותר. כך שבוודאות המסלול העובר בין (u,v), יביא לנו את המסלול הקצר ביותר גם ביניהם.

ועכשיו להוכחה עצמה:

על דרך השלילה, נניח שקיימים קדקודים v שעבורם $d[v]\ne \delta(s,v)$, ונתבונן ב- v אחד מהם, עם $\delta(s,v)$ מינימלי. לפי **למה 2**, $d[v]>\delta(s,v)$.

1. $v \ne s$, כי $d[s]=0=\delta(s,s)$. (הנחנה ברורה)
2. v ניתן להגעה מ-s , כי אחרת $\delta(s,v)=\infty \ge d[v]$. בזכור – מה שלא מעודכן נשאר עם ערך אינסוף.

יהי u הקדקוד מיד לפניv במסלול הקצר ביותר מ-s ל-v.

- ➔ $\delta(s,u)<\delta(s,v)$ $\delta(s,v) = \delta(s,u)+1$ $d[u] = \delta(s,u)$ (מהמינימליות של $\delta(s,v)$)
- ⍰ $d[v]>\delta(s,v)=\delta(s,u)+1=d[u] + 1$ *

נשים לב מה קורה כאשר מוציאים מהתור את u. הקדקוד v הוא או לבן, או אפור, או שחור:

- v לבן: אז בשורה 15 (בקוד), $d[v]=d[u]+1$, בסתירה ל*.
- v אפור: קיים קדקוד אחר $w\in V$ שהוצא מהתור לפני u
  ⍰ $d[v]=d[w]+1$. אבל מהמסקנה, $d[w]\le d[u]$
  ⍰ $d[v]\le d[u]+1$, בסתירה ל*.
- v שחור: v כבר הוצא מהתור
  ⍰ (לפי המסקנה) $d[v] \le d[u]$, ולכן $d[v]<d[u]+1$, בסתירה ל*.

⍰$d[v] = \delta(s,v)$

3. אם $\pi[v]=u$ אזי $d[v]=d[u]+1$. לכן, המסלול הקצר ביותר מ-s ל-v יתקבל על ידי לקיחת המסלול הקצר ביותר מ-s ל-$\pi[v]$, והוספת הצלע $(\pi[v],v)$.

תת גרף הקודמים

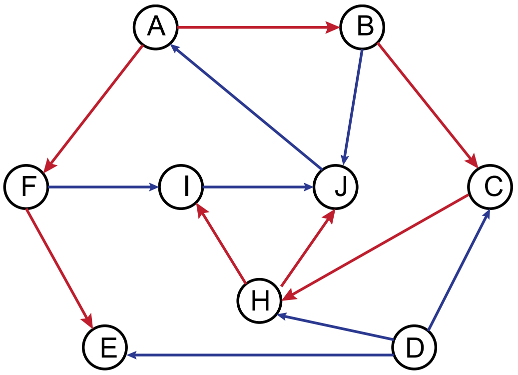
מה אנחנו מקבלים כתוצאה מהשדות פאי מכל קדקוד? קבוצת קדקודים של הגרף שהפאי שלהם שונה מNull, ובתוספת הקדקוד s. כאשר:

$$V_\pi=\{v\in V : \pi[v] \neq NULL\}\cup\{s\}$$

$$E_\pi=\{(\pi[v],v)\in E : v \in V_\pi - \{s\}\}$$

קבוצת הקשתות המתייחסות לעץ גרף הזה יוצר לנו עץ בו יש רק את הקשתות המתייחסות לגרף. שהוא במובן עץ המסלול הקצרים ביותר הניתנים להגעה מהמקור לכל קדקוד אחר.

במקרה לפנינו, טבלת הπ תתמלא בצורה הבאה:



J	I	H	F	E	D	C	B	A	v
H	H	C	A	F	\	B	A	\	**π(v)**

כאשר השורש s=A

קבוצת הצלעות של π: $V_\pi=\{A,B,C,E,F,I,H\}$

וקבוצת הצלעות: E_π={(A,B),(B,C),(F,E),(A,F),(C,H),(H,I),(H,J)}

עץ רוחב

תת-גרף הקודמים G_π הוא **עץ רוחב**, אם:

- V_π מורכב מכל הקדקודים הניתנים להגעה מהמקור s.
- לכל $v \in V_\pi$, קיים מסלול פשוט יחיד מ-s ל-v ב-G_π, שהינו גם מסלול קצר ביותר מ-s ל-v ב-G.

יש לציין שמדברים פה על תת-גרף **של תת גרף הקודמים** שחלקו ייחשב עץ רוחב וחלקו לא.

והצלעות של העץ נקרות צלעות עץ.

חיפוש לעומק Depth First Search

נניח שמישהו נמצא במבוך, והוא בתוך אחד החדרים המרכזיים ויש יציאה אחת או יותר מהמבוך. אם נגיד שהצמתים של המסדרונות הם הקדקודים והצלעות הן המסדרונות של המבוך. במקרה כזה חיפוש לרוחב ייתן לך את כל המסלולים לכל החדרים ולכל היציאות הקיימים. אך אנחנו עוברים על **כל** המבוך ועל כל השכנים של כל הקדקודים וכך מקבלים את כל רשימת הסמיכויות, אך זה לא יעיל.

במקרה כזה, יותר יעיל להגיע לצומת ההסתעפויות ולמספר את ההסתעפויות, בשביל שנוכל לעבור בהם לפי הסדרץ.פונים למעבר הראשון, ומגיעים לחדר השני בו יש הסתעפויות ושוב ממשיכים וחוזר חלילה. במקרה ואנחנו נתקרב לקדקוד אליו כבר הגענו אנחנו נדע שאנחנו לא ממשיכים לשם, אלא ממשיכים לחפש במקום חדש.

בשביל לחזור אחורה לנקודת המוצא אנחנו נשתמש במחסנית (וזה ההבדל הגדול מחיפוש רוחב) וחוזרים על פי שם התחנה עד שמגיעים להתחלה. אך בדומה לחיפוש רוחב, אנחנו נסמן כל קדקוד שעברנו עליו כבר בשחור.

בחיפוש לעומק יש לנו משתנה זמן שמתקדם באחד בכל פעם שאנחנו מגלים קדקוד חדש (לבן), או שאנחנו משחירים קדקוד. כך שלכל קדקוד ש שני שדת נוספים:d (זמן הגילוי discovery), תוך כמה זמן הגיעו אליו לראשונה, ו-f (זמן הסיום /השחרה finish).

בשמתחילים במקור ה-d=1, מתקדמים ובכל נועה מקדמים אותו ב-1, אם מגיעים לקדקוד שישר משחירים אותו – בדרך כלל אחד שנמצא בקצות העץ – אז המרחק בין ה-d-f הוא בפער של אחד. מבחינת הצבעים: לבן – קדקוד ששני השדות האלו ריקים. אפור – יש זמן הגעה אבל לא זמן סיום. שחור – שני השדות מלאים.

על יד האלגוריתם ניתן לראות גם האם יש מעגלים בתוך הגרף על ידי סימון של "קשת אפורה", ואם מגיעים לאחת כזאת אנחנו מבינים שהיו בגרף מעגלים.

קלט: גרף G = (V, E) (מבוון או לא מבוון).

פלט: לכל v ∈ V

- d[v]= **זמן הגילוי (Discovery)** של הקדקוד v (הזמן בו הקדקוד הופך מלבן לאפור).
- f[v]= **זמן הסיום (Finish)** של הקדקוד v (הזמן בו הקדקוד הופך מאפור לשחור).

- $\pi[v]$= קדקוד u שהוא קודם של הקדקוד v, כך ש-הπ התגלה תוך סריקת רשימת הסמיכויות של u.
- בניית **יער עצי עומק**.

תת-גרף הקודמים של DFS מוגדר קצת אחרת, והוא יוצר **יער עצי עומק**:

- $G_{\pi} = (V, E_{\pi})$
- $E_{\pi} = \{(\pi[v],v), v \in V \text{ and } \pi[v] \neq NULL\}$

האלגוריתם של החיפוש לעומק מורכב משתי אלגוריתמים. התבנית הראשית ופונקציית עזר DFS-visit.

```
1. for each vertex u ∈ V[G]
2.     do color[u] ← WHITE
3.        π[u] ← NULL
4. time ← 0
5. for each vertex u ∈ V[G]
6.     do if color[u] = WHITE
7.           then DFS-Visit(u)
```

DFS-Visit(*u*)

```
color[u] ← GRAY
time ← time + 1
d[u] ← time
for each v ∈ Adj[u]
    do if color[v] = WHITE
          then π[v] ← u
               DFS-Visit(v)
color[u] ← BLACK
f[u] ← time ← time + 1
```

הפונקציה הראשית בודקת האם מדובר בקדקוד בצבע לבן. במידה וכן, שולחים אותו לפונקציה, וכך עוברים בלולאה על כל הקדקודים.

בפונקציית העזר קודם כל צובעים את הקדקוד בצבע אפור, ומקפיצים את משתנה הגילוי ב-1, ומכניסים לשדה. לאחר מכן, עוברים על כל הקדקודים הקשורים אליו בצורה רקורסיבית, וברגע שמסיימים את כל השני, צובעים את הקדקוד בשחור, ומעדכנים את הזמן שחלף במשתנה הסיום. למעשה, בסיום הסריקה, איבר האחרון שנסרק יחזיק בתא הזמן-סיום שלו ערך השווה לפי 2 ממספר הקדקודים.

למעשה, בכל פעם שאנחנו מסיימים קריאה בDFS-visit מהפונקציה הראשית (שורה 7) אנחנו מסיימים קריאה בתת-עץ מסוים ששרשו הוא בקדקוד המקור, וחוזרים לבדוק האם יש עוד קדקודים לבנים בשאר העץ (שורה 5). בסופו של דבר יישאר לנו יער של עצי עומק – שוב, בשונה מחיפוש רוחב, בו אנחנו הולכים רק על המקושרים, כאן אנחנו ישר בודקים בצורה לקסיקוגרפית אם קיים קדקוד שעדיין לבן – גם אם לא מקושר לאותו קדקוד – ולכן יכול להיווצר לנו מצב של יער ולא רק עץ אחד בודד.

אם יש לנו קשת שחוזרת על עצמה היא נקראת "קשת אחורה", והיא בעצם מסומנת באפור, ומגדירה לנו האם יש מעגלים בתוך הגרף. במידה ויש 'שתות באלה זה אומר שיש מעגלים בגרף. יש לציין, הקשתות אחורה לא יופיעו בתת הגרף הקודמים.

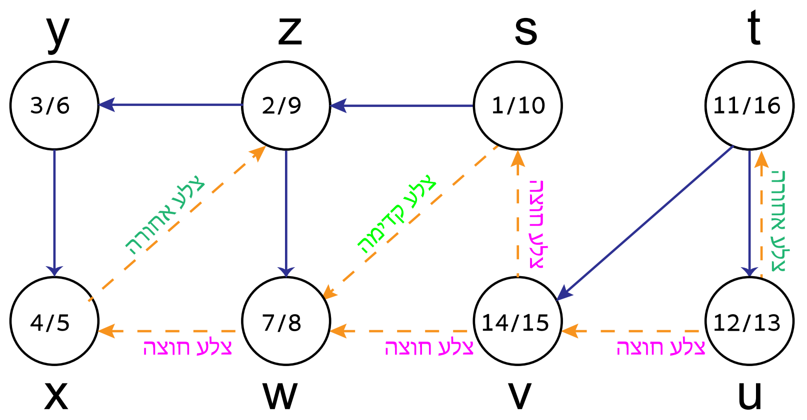
<u>משפט הסוגריים</u>

אם נתבונן אחרי הרצת האלגוריתם על שני קדקודים כלשהם (u,v), (כאשר ברגע לא חשוב סדר ההגעה ואם הם היו סמוכים), ובכל תא מסומן כמובן זמן ההגעה והסיום הרלוונטים להם. היחס סין הערכים יהיה אחד משלושה מצבים:

1. f [v]>d[v]>f [u]>d[u] או f [u]>d[u]>f [v]>d[v], ו-u אינו צאצא של v, וכן להפך. אם כל המעבר על הגילוי והסיום לא קשור אחד לשני, הוה אומר, כל אחד טופל בנפרד ואין שום יחס בין שני הקדקודים.
2. f [u]>f [v]>d[v]>d[u], ו-v הוא צאצא של u. הטיפול של v נמצא בתוך הזמן של u, אפשר להבין מזה שיש איזשהו קשר בי הקדקודים, ומאחר ו v הוא הפנימי יותר, ניתן להבין מזה, שהוא צאצא של u.
3. f [v]>f [u]>d[u]>d[v], ו-u הוא צאצא של v.

חוק זה נקרא בהשאלה "חוק הסוגריים", כאשר זמן הגילוי והסיום של התא מוגדר כ פתיחה וסגירה של הסוגריים, כך שאם נבחר שני זוגות של סוגריים אקראיים, נוכל לראות אם אחד מהם מוכל בתוך השני על ידי בדיקה של סדר התוים.

למשל, עבור הגרף הבא:



נתחיל בקדקוד s, ונתקדם על פי הסדר הבא – (s (z (y (x x) y) (w w) z) s) (t (u u) (v v) t).

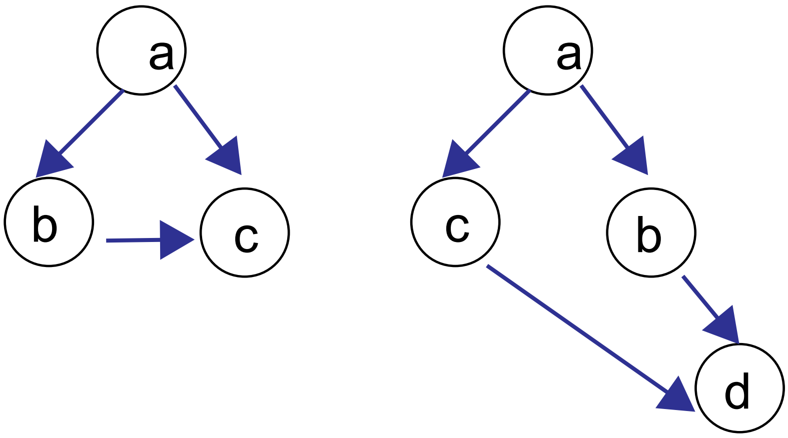
<u>סיווג הקשתות ב</u><u>DFS</u>

הקשתות בחיפוש עומק ממוינות על פי ארבעה סוגים שונים, שכולם הופיעו בגרף הקודם, ולכן בדאי להתעכב עליהם רגע.

- **צלע עץ (Tree edge)** צלע (u,v) אשר דרכה נתגלה הקדקוד v (כלומר, צלע ב- G_π). צלע זו מגלה קדקודים לבנים. למעשה, מדובר בצלע הסטנדרטית שתרכיב לנו אחר כך את תת-עץ הקודמים. אלו החיצים המסומנים באיור בצבע כחול.
- **צלע אחורה (Back edge)** צלע (u,v) כאשר u הוא צאצא של v ב-G_π. צלע היוצאת מקדקוד אחד בו אנו נמצאים על פי המסלול, ומצביע לעבר קדקוד שכבר נסרק. באיור לפנינו הצלע הראשונה העונה להגדרה זו היא הצלע (x,z), מאחר והקדקוד z כבר התגלה, אי אפשר לעבור אליו שוב. קיום קשת אחורה אחת או יותר, שקול לקיום המעגלים בגרף. קשת זו מגלה לנו קדקודים אפורים. נראה בהמשך את החשיבות של זה.
- **צלע קדימה (Forward edge)** צלע (u,v) כאשר v הוא צאצא של u, אך אינו ב-G_π. בדוגמא שלנו קיימת רק צלע אחת כזו – (s,w) – למעשה w היא צאצא של s שהוא הקדקוד, אך מכיוון שהקדקוד הזה הופיע כבר קודם במהלך סריקה תת העץ של הקדקד הקודם, כשהסתיים סבב הבדיקה והצביעה, הקדקוד הזה לא רק שהתגלה (כמו בצלע אחורה), אלא הטיפול בו כבר סוים, ולכן הוא אף בצבע שחור.

- **צלע חוצה (Cross edge)** כל צלע אחרת (יכולה לחבר בין קדקודים באותו עץ עומק, או בין קדקודים בעצי עומק שונים). לכאורה, זה נראה מאוד דומה לצלע קדימה – שניהם מגלים קדקודים שחורים שהטיפול בהם כבר סוים, אך יש הבדל ביניהם. בעוד שצלע קדימה מתייחסת לצאצאים של אותו קדקוד ובאותה דרגה, צלע חוצה מתייחס לקדקודים שאין ביניהם קשר ישיר – למשל קדקודים הנמצאים בתוך עץ עומק, או שני קדקודים המקשרים בעומק בין שני עצים שונים.

להבנה נוספת בהבדל בין שני סוגי הצלעות ניתן לראות את הציור מטה – הגרף השמאלי הוא צלע קדימה. הצאצאים באותה דרגה אך הקדקוד c כבר "נשרף" על ידי b. הגרף הימני מדבר על שני קדקודים בעומק של העץ, המקושרים ביניהם ולכן מדובר בצלע שהיא "חוצה" דרגות וענפים, אך כבר מושחרת.



בהרצת DFS על גרף **לא מכוון**, מתקבלות רק צלעות עץ וצלעות אחורה. לא מתקבלות צלעות קדימה או צלעות חוצות. אם ניקח לדוגמה את הגרף הימני, ונניח שהוא לא מכוון, אזי לאחר שאנחנו מגיעים לקדקוד d, אנחנו ממשיכים ל-c. לאחר מכן, הוא כבר מצביע בחזרה לשורש a, אך היא נסרקה כבר והצלע המקשרת (c,a) היא עכשיו על תקן צלע אחורה.

משפט המסלול הלבן

הגדרנו קודם במשפט הסוגריים, שיכול להיות ש-v הוא צאצא של u[24], משפט המסלול הלבן מוסיף על כך ואומר ש-

v הוא צאצא של u אם"ם בזמן d[u] (זמן הגילוי של הקדקוד u), קיים מסלול (u,v) המורכב כולו מקדקודים לבנים. הכוונה, לא רק שהצאצא לא התגלה עדיין, אלא כל הקדקודים הנמצאים ביניהם, גם הם לבנים – לא התגלו.

[24] d[u]<d[v]<f [v]<f [u] (המקרה השני)

הוכחה:

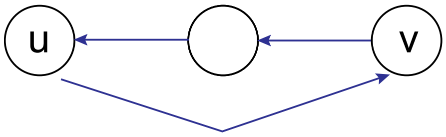
- אם v=u, אז בוודאי שהקדקודים שהם אב קדמון וצאצא לעצמו, בוודאי שכל מה שביניהם הוא לבן, או לפחות לא כל דבר אחר.
- אם v הוא צאצא ממש של u, כלומר אינו שווה לו – על פי הקוד של האלגוריתם, אז אנחנו זימנו את הפונקציה הרקורסיבית של v, מתוך מחסנית הרקורסיה השייכת לU, וזה אומר על פי התנאי לכניסה לרקורסיה, שכל מה שנכנס הוא בצבע לבן.
- בכיוון הפוך – נניח שיש מסלול לבן המוביל מ-v (הצאצא) ל-u (האב) בזמן d[u] (זמן הגילוי של u), אבל v לא צאצא שלו. ניתן להניח שכל קדקוד בדרך בין v ל-u יהיה צאצא של u. (אחרת, מדובר על קדקודים סמוכים). נגדיר גם את הקדקוד w שנמצא בין שני האיברים, כך שיהיה צאצא של u (או u עצמו), ובנוסף w מוגדר להיות גם אב קדמון של v (או הוא עצמו). כלומר, w חייב להיות בין שניהם, ועל פי משפט הסוגריים יוצא לנו כי (u(w(v v)w)u), ולכן v הוא צאצא של u.

גמ"ל – גרף מכוון ללא מעגלים

למה (הנחת מוצא) - גרף מכוון G הוא ללא מעגלים אם"ם הרצת DFS על G לא מניבה צלעות אחורה.

גמ"ל (או DAG – Directed Acyclic Graph) הוא מקרה פרטי של גר, המקיים את התנאים הפשוטים המצויינים בשמו: 1. הוא מכוון 2. אין מעגלים בגרף. הלמה בה אנחנו משתמשים גורסת כי הרצת חיפוש עומק על גרף כזה לא וציא צלעות אחורה. בהבנה פשוטה, ציינו את זה כבר בסיווג הצלעות, שצלעות אחורה למעשה מבשרים על הימצאות מעגלים בגרף, ולכן המשפט הזה מובן, אך יש צורך להוכיח אותו באופן רשמי.

הוכחה:



שתי ההוכחות הבאות מתייחסות לשני הנתונים של הגמ"ל ומוכיחה בשלילה משני הכיוונים שהם מקיימים אחד את השני:

- בדרך השלילה, נניח שבכן קיימת צלע אחורה (u, v). אזי v הוא אב קדמון של u ביער עצי העומק. לכן, ישנו מסלול (u,v), כך ש (v,u,v)הינו מעגל, בסתירה לנתון.

מצד הצלעות המכוונות לא יכול להיות שיהיה צלע אחורה, אחרת זה מניב לנו מעגלים – מה שאמרנו שלא יכול להיות, ולכן בשלילה זה מקיים.

- בדרך השלילה, נניח שבכן קיים מעגל c בגרף G. נסמן ב-v את הקדקוד הראשון שנתגלה במעגל c, כאשר (u,v) היא הצלע שלפני קדקוד זה במעגל. בזמן d[v], הקדקודים של c יוצרים מסלול לבן (v,u). לפי משפט המסלול הלבן, u הוא צאצא של v ביער עצי העומק. לכן, (u,v) היא צלע אחורה, בסתירה לנתון.

מצד האפשרות לקיום מעגל בגרף שהוא גמ"ל – נניח שיש באמת מעגל בגרף. הקדקוד הראשון שמתגלה יוצר מסלול עד הקצה שלו, אך דבר זה לא מתקיים, מאחר ויש לו צלע אחורה שלא תהיה מסלול לבן, ולכן בשלילה זה לא מתקיים.

מיון טופולוגי

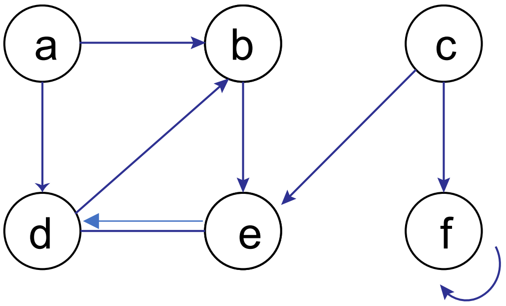
מהו מיון טופולוגי? סידור של קבוצות איברים התלויים אחד בשני, כך שאף איבר לא יופיע לפני איבר אחר בו הוא תלוי. בעיות נפוצות שנפתרות על ידי מיון טופולוגי – סדר פעולות של מעקב אחרי הוראות:

ביצוע הוראה לפני זמנה, עלול לגרום לכל הפעולות לא לעבוד – שימוש בנתון בתכנות לפני שהוא הוגדר גורם לתוכנה לקרוס. או לחילופין – יצירת מעקב אחרי קורסים של סטודנט – מעבר של קורס אחד הוא תנאי הכרחי למעבר לקורס הבא, כך שיש בעצם איזה גרף שהוא מכוון שמוביל מנקודה אחת לאחרת או לכמה. אי אפשר לגשת לקורס מבנה נתונים ב', ללא מבנה נתונים א'.

על מנת לעשות מיון טופולוגי משתמשים בגמ"ל, כך שמגדירים יחס סדר על הקדקודים באופן הבא: אם (u,v)∈E, אז u<v.

כאשר יכול להיות שהיחס מתקיים גם עבור שני קדקודים (u,v) שלא מתקיים ביניהם קשר כלשהו בגרף, ועדיין ניתן לקבוע מי "גדול יותר" ממי, וזאת עקב תכונת הטרנזטיביות.

בהנחה שמדובר בגרף ללא מעגלים (מעצם ההגדרה), אנחנו רוצים לכתוב את סדר הקדקודים באופן שיהווה תיאור של כל הקדקודים בגרף שילכו רק "קדימה", מקדקוד שנמצא בראש הגרף לאילו שנמצאים מאוחר יותר בציר.

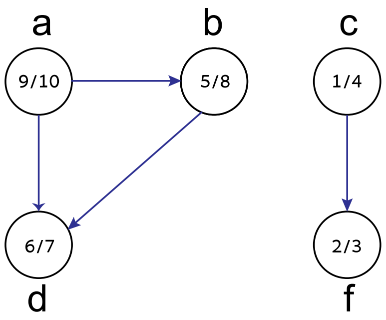


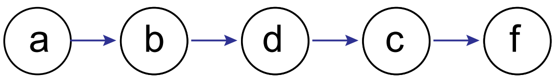
למשל עבור הגרף המתואר ניתן למיין את הגרף בצורה טופולוגית בכמה אופנים:

A,b,e,d,c,f
A,d,b,e,c,f

אין בעיה שיהיו מספר מיונים שונים עבור כל גרף, ואפילו בהגדרה יש לכל גרף לפחות מיון טופולוגי אחד. מאחר והסידור מדבר על המיון והיחס בין שני קדקודים ספציפית. בדוגמא כאן, הקדקוד d קטן יותר מa בכל אופן, ואין שום יחס בינו לבין c, כך שלא משנה מי מהם מופיע קודם ברשימה.

במקרה כמו זה הקיים כאן, הגרף מוגדר **יחס סדר חלקי**, מאחר שיש איברים שאין ביניהם שום יחס – (c,a), כאשר המטרה של מיון טופולוגי הוא להגיע למצב בו יהיה יחס סדר מלא, בו יהיה יחס קיים בין כל הקדקודים בצורה ברורה.





על מנת להשלים את זה ליחס סדר מלא, צריך שלא יהיו מעגלים בגרף.

האלגוריתם עובד בשני שלבים:

1. ריצה של DFS
2. הכנסת הקדקוד בזמן הסיום הגבוה ביותר שיהיה בראש רשימה מקושרת, ומה שטופל ראשון יהיה בזנב.
3. החזרת הרשימה.

הוכחת נכונות – מיון טופולוגי

צריך להוכיח:
שהאלגוריתם המוצע מחזיר מיון טופולוגי, כלומר, שאם (u, v)∈ E אזי f [u] < f [v].

בהנחה שאכן מתקיים ש v>u, כלומר v מופיע אחרי u בסדר הגרף, אזי זמן הסיום של u יהיה אחרי זמן הסיום של v, וזאת מאחר שזמן הסיום "חוזר אחורה" בגרף אל עבר המקור.

הוכחה:
כאשר מטיילים על הצלע (u, v), הצבע של u הוא אפור (הגענו לזמן הגילוי שלו, אך לא סיימו את הטיפול הקשור בו). מה לגבי v?

- האם v אפור?

אם כן, זאת אומרת ש-u צאצא של v. (מאחר ומי שמתגלה מוקדם יותר, מוגדר להיות קטן יותר).
ולכן,
(u, v) צלע אחורה.

זאת בסתירה לכך שבDAG אין צלעות אחורה. וזה לא ייתכן.

- האם v לבן?

אם כן, זאת אומרת ש-v צאצא של u.

לכן, לפי משפט הסוגריים, d[u] < d[v] < f [v] < f [u].

- האם v שחור?

אם כן, זאת אומרת ש-v סיים, אבל u לא סיים.

ולכן, f [v] < f [u].

מ.ש.ל.

רביב קשיר חזק

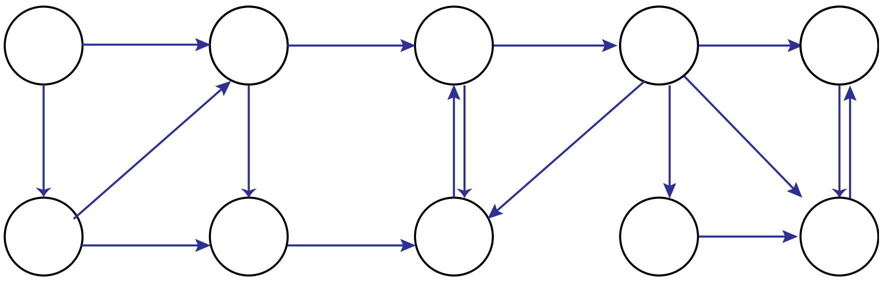
הגדרה: גרף מכוון G=(V,E) הוא **קשיר חזק**, אם לכל u,v∈V קיימים מסלולים מ-u ל-v ולהפך.

דיברנו על רכיבי קשירות בגרף בלתי-מכוון, וכבר אז ציינו שבגרף מכוון אנחנו לא מדברים כל רכיב קשירות, אלא על רכיב קשירות חזק. על פי ההגדרה לרכיב קשירות, אנחנו מחפשים את מספר הקדקודים המקסימלי בגרף, עבורם ניתן להגיע מכל קדקוד אחד לאחר. בגרף בלתי מכוון, זה היה יותר קל, מאחר וכל חיבור בין קדקוד כלל מעבר לשני הכיוונים, אך בגרף מכוון אנחנו אמורים לבדוק את המסלולים והמעגים המרכיבים את רכיב הקשירות.

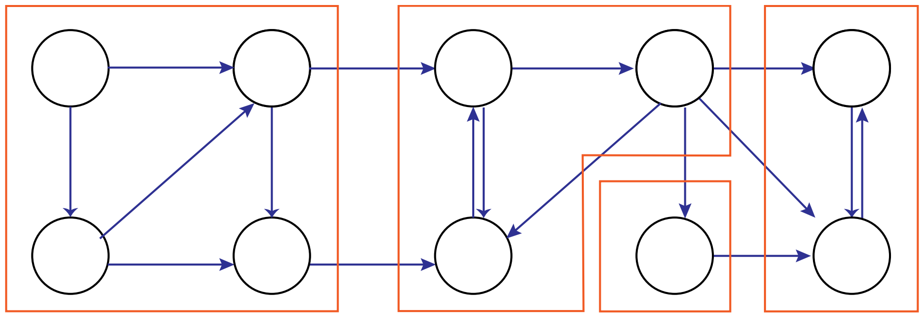
רכיב קשיר חזק– strongly connected component (SCC)– של גרף G הוא קבוצה מקסימלית של קדקודים C⊆V, כך שלכל u,v∈C קיימים מסלולים מ-u ל-v ולהפך.

חוזקה של רכיב קשירות בגרף מכוון מוגדר בכזה שעבור כל קדקוד בקבוצה החלקית הנגזאת מהגרף G, תהיה אפשרות להגיע מקדקוד אחד לשני ובחזרה.

ניקח לדוגמא את הגרף הבא:

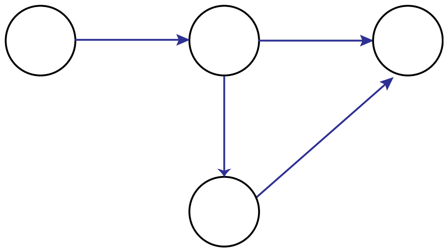


אם נבחן את רכיבי הקשירות החזקים שלו, נוכל לראות שהם מתחלקים לקבוצות באופן הבא:



אפשר לשים לב שכל קבוצה תחומה, מחוברת רק בתוכה, ועוברת לקבוצה הבאה רק באופן חד כיווני.

מצב זה יוצר לנו הגדרה חדשה של **גרף רכיבים** – $G^{SCC}=(V^{SCC},E^{SCC})$ (SCC – Strongly Connected Component), המציין את הייצוג של קבוצת הרכיבים הקשירים. כל קבוצת רכיבים מוגדר על ידי קדקוד V^{SCC}, והצלע המקשרת בין הרכיבים מוגדרת כ-E^{SCC}. שוב, אם ניקח את הדוגמא שהשתמשנו בה, יש לנו ארבעה קבוצות המסודרות ביניהם באופן הבא, בצורה של גרף רכיבים:



גרף הרכיבים-קשירים-חזק, הוא כמובן גרף גמ"ל.

למה: יהיו C ו- C′ שני SCC שונים ב-G, ויהיו $u,v\in C,\ u',v'\in C'$, כך שיש מסלול מ-u ל-u′ ב-G. אזי לא ייתכן שקיים גם מסלול מ-v′ ל-v ב-G.
ניקח שני רכיבים קשירים שונים. בין שני הרכיבים יש צלע מקשרת בין שני הקדקודים v והקדקוד מהרכיב השני 'v. המסלול הקיים ביניהם הוא יחיד, ואין עוד צלע חוזרת מ-'v ל-v.

ניתן להוכיח זאת בשלילה:

נניח כי קיים מסלול החוזר מ-'v ל-v. במקרה כזה, זה אומר שיש דרך להגיע מהקדקוד u לקדקוד 'v ואף לחזור. אך אם מצב זה היה נכון, שני הקדקודים היות תחת אותו רכיב קשיר (מעצם הגדרת רכיב קשיר בקבוצת קדקודים שניתן להגיע מכל קדקוד לקדקוד אחר וחזור), ומאחר והם שני רכיבים שונים, דבר זה עומד בסתירה לאפשרות ולכן לא נכון.

שיחלוף של גרף מכוון (Transpose)

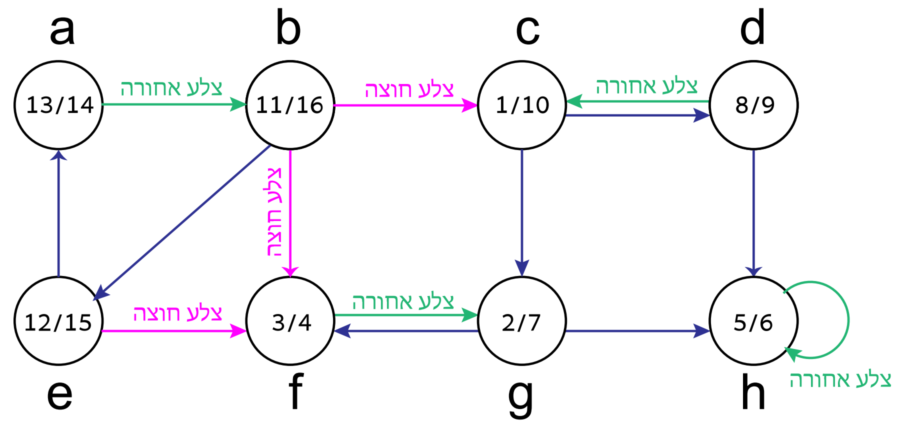
הגדרה: $G^T=(V,E^T)$ הוא ה-**transpose** של גרף מכוון G, $E^T = \{(u, v) : (v, u) \in E\}$.

גרף משוחלף G^T הוא בעצם אותו גרף מבחינת סדר הקדקודים, רק שינוי של כיוון הצלעות שיעברו הפוך בסדר מהסוף להתחלה. במובן שלגרף המקורי ולמשוחלף יש אותם רכיבים קשירים, מאחר שהדבר היחיד שמשתנה הוא הכיוון של הצלעות , ואם לא היה מעגל בגרף זה משהו שלא משתנה על ידי השיחלוף.

אלגוריתם למציאת SCC

1. הרצת DFS(G) לחישוב זמני הסיום f[u] לכל קדקוד u.

מריצים את החיפוש עומק, ומוציאים רק את הנתונים של זמני הסיום של הקדקודים.

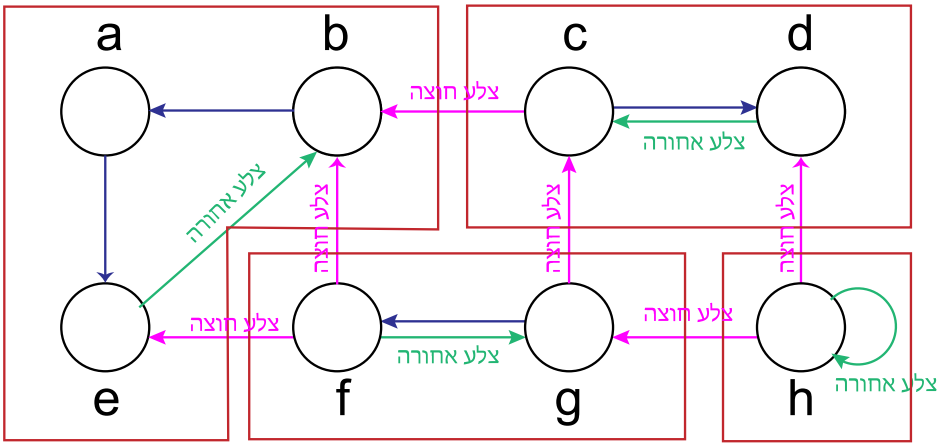


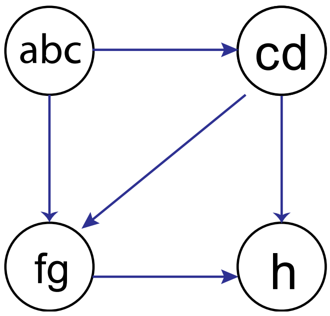
לוקחים את הגרף G ועושים עליו חיפוש עומק. סדר הקדקודים היוצא לאחר החיפוש הוא: c,g,f,h,d,b,e,a.

ניתן כבר עכשיו לראות את הרכיבים הקשירים, אך עדיין לא נעשה בהם שימוש.

2. חישוב G^T. (שיחלוף של הגרף)
בצורה פשוטה- מכניסים את זמן הסיום הפוך, כך שזמן הסיום המאוחר ביותר יהיה ראשון.

3. הרצת $DFS(G^T)$, כשסדר הקדקודים בלולאה החיצונית הוא בסדר יורד של f[u], שחושב לעיל.



לאחר הרצה חוזרת על הגרף המשוחלף ניתן כבר לראות את הרכיבים הקשירים המופרדים על ידי הצלעות החוצות. סדר הקדקודים כרגע הוא: b,a,e,d,c,g,f,h

4. החזרת הקדקודים בכל עץ ביער הנוצר בהרצה השניה, כSCC נפרד.