מבני נתונים

מרצים

ד"ר עמית דניאלי
ד"ר גיא כץ

דניאל דייצ'ב
דויד קיסר שמידט

deychev.com

מבני נתונים | 67109

מרצים - ד״ר עמית דניאלי, ד״ר גיא כץ

הרצאות

דניאל דייצ׳ב
דויד קיסר שמידט

12 באוגוסט 2022

מצאתם שגיאה? ספרו לנו! שלחו לנו מייל לאחת מהכתובות שכאן:

daniel.deychev@mail.huji.ac.il

david.keisarschm@mail.huji.ac.il

תוכן העניינים

I הרצאה I - מושגי יסוד, סיבוכיות זמן ריצה 10

1 אלגוריתמים לפתרון בעיות 13

1.1 מדדי יעילות 17

2 סיבוכיות 20

2.1 חסמים אסימפטוטים 25

II הרצאה II - שיטת האיטרציה, שיטת ההצבה ומשפט האב 28

3 פתרון נוסחות נסיגה 28

3.1 פתרון נוסחות רקורסיביות 31

4 משפט האב $Master\ Theorem$ 33

III הרצאה III - מיון מהיר ומיון מבוסס השוואה 40

5 בעית המיון 40

5.1 מיון מהיר $Quick\ Sort$ 41

6 מיון מבוסס השוואה - חסם תחתון 51

6.1 עץ בחירה 52

IV הרצאה IV - טבלות גישה ישירה, טבלות גיבוב, שרשור, גישה פתוחה, פונקציות גיבוב 56

7 טבלות גיבוב 57

7.1 שיטות גיבוב 60

7.2 מימושים לפעולות $Insert, Delete, Search$ 66

8 פונקציות גיבוב 72

V הרצאה $\mathbb{V}$ - גיבוב אוניברסלי, משפחות אוניברסליות וגיבוב מושלם 75

9 גיבוב אוניברסלי 75

9.1 חיפוש מחיקה והוספה בשיטות שונות 76

9.1.1 חיפוש מחיקה והוספה בשרשור 76

9.1.2 חיפוש מחיקה והוספה במיעון פתוח 77

10 בניית משפחה אוניברסלית 79

11 גיבוב מושלם 80

VI הרצאה $\mathbb{VI}$ - ערימות ותורי עדיפויות $(Heaps\ and\ ProrityQueues)$ 83

11.1 תור עדיפויות (PriorityQueue) 83

12 ערמה (Heap) 84

12.1 יצוג ערמה כמערך 86

12.2 תכונות של עצים בינאריים 87

12.3 שמירה על תכונת הערמה 88

12.4 $Max-Heapify\,(A,i)$ 89

12.4.1 סיבוכיות 90

12.5 בניית ערמה 92

12.5.1 סיבוכיות זמן ריצה 96

13 מיון ערימה $HeapSort$ 96

14 פעולות שימושיות בערמות 97

14.1 $Heap-Max$ 98

14.2 $Heap - Extract - Max$ . . . 98
14.3 $heap - increase - key$ . . . 99
14.4 $Heap - Insert$ . . . 100

VII הרצאה $\mathbb{VII}$ - עצי חיפוש בינאריים $(Binary\ Search\ Trees)$ 102

15 מבוא 102

16 עצי חיפוש בינאריים 102

VIII הרצאה $\mathbb{VIII}$ - גרפים 117

17 מושגים בסיסיים 117
17.1 גרפים מכוונים, לא מכוונים ותכונותיהם . . . 117
17.2 עצים ותכונותיהם . . . 119

18 יצוג גרפים 119

19 בעית המסילה הקצרה ביותר 120
19.1 אלגוריתם חיפוש לרוחב $BFS - Breadth\ First\ Search$ . . . 120

IX הרצאה $\mathbb{IX}$ - אלגוריתם $DFS - Depth\ \ First\ \ Search$, מיון טופולוגי ומציאת רכיבי קשירות חזקים 124

20 DFS 124
20.1 מוטיבציה . . . 124
20.2 פעולות האלגוריתם . . . 124
20.3 סיבוכיות . . . 127

21 מיון טופולוגי 129

22 מציאת רכיבי קשירות חזקים 130

22.1 הוכחת נכונות 133

X הרצאה $\mathbb{X}$ - עצים פורשים מינימלים 134

23 עצים פורשים מינימלים 134

23.1 מוטביציה 134

23.2 זהויות של עצים פורשים מינימלים 134

23.2.1 אלגוריתם גנרי למציאת עץ פורש מינימלי 135

23.3 האלגוריתם של קרוסקל $(Kruskal)$ 137

23.3.1 ניתוח האלגוריתם 138

23.4 האלגוריתם של פרים $(Prim)$ 138

23.4.1 ניתוח האלגוריתם 140

XI הרצאה $\mathbb{XI}$ - המסלול הקצר ביותר לגרפים ממושקלים 140

24 מבוא 140

24.1 מובטיבציה 140

24.2 הגדרות ותכונות 141

25 אלגוריתמים לפתרון הבעיה 141

25.1 המשותף בין האלגוריתמים 142

25.2 האלגוריתם של דיקסטרה $(Dijkstra)$ 144

25.2.1 ניתוח זמן ריצה 147

25.3 האלגוריתם של בלמן פורד $(Bellmann - Ford)$ 147

25.3.1 צלעות בעלות משקל שלילי 147

25.3.2 אופן פעולת האלגוריתם 148

25.3.3 ניתוח האלגוריתם 150

XII הרצאה XII - בעיית כל המסלולים הקצרים ביותר, "כפל מטריצות", האלגוריתם של פלויד וורשל 152

26 מבוא 152

27 זהויות והגדרות 153

27.0.1 קבלת המסלולים הקצרים ביותר בהנתן Π 154

28 הכפלת מטריצות (Matrix-Multiplication-Algorithm) 155

28.1 מבוא 155

28.2 האלגוריתם 156

29 האלגוריתם של פלויד-וורשל (Floyd-Warshall) 159

29.1 מבוא 159

29.2 האלגוריתם 160

XIII הרצאה XIII - קבוצות זרות $(Disjoint\ Sets - Union\ Find)$ 164

30 מבוא 165

30.1 הגדרת הבעיה 165

30.2 מוטיביציה - מציאת רכיבי קשירות 166

31 פתרונות לבעיה 168

31.1 פתרון ראשון - רשימות מקושרות 168

31.1.1 מימוש הפעולות 169

31.2 יער של קבוצות זרות $(Disjont - Set\ Forest)$ 171

31.2.1 איחוד לפי גובה $(Union - By - Rank)$ 172

31.2.2 דחיסת מסילות $(Path - Compression)$ 173

31.2.3 מימוש הפעולות 173

31.3 זמן ריצה 175

31.4 סיכום 176

רשימת אלגוריתמים

1 העברת האיבר המקסימלי במערך לסופו . . . 11
2 מיון הכנסה . . . 22
3 מיון מהיר - $Quick\ Sort\,(A, l, r)$ . . . 42
4 $Partition\,(A, l, r)$ . . . 45
5 הוספת איבר לטבלת גיבוב - $Hash - Insert\,(T, k)$ . . . 66
6 חיפוש איבר בטבלת גיבוב - $Hash - Search\,(T, k)$ . . . 67
7 $Max - Heapify\,(A, i)$ . . . 90
8 $Bulid - Max - Heap\,(A)$ . . . 92
9 $Heap - Sort\,(A)$ . . . 97
10 $Heap - Max\,(A)$ . . . 98
11 $Heap - Extract - Max\,(A)$ . . . 99
12 $Heap - Increase - Key\,(A, i, key)$ . . . 100
13 $Heap - Increase - Key\,(A, i, key)$ . . . 101
14 $Tree - Search\,(x, k)$ . . . 104
15 $Tree - Search\,(x, k)$ . . . 105
17 $Tree - Maximum\,(x)$ . . . 105
16 $Tree - Minimum\,(x)$ . . . 105
18 $Inorder - Tree - Walk\,(x)$ . . . 106
20 $Prerder - Tree - Walk\,(x)$ . . . 106
19 $Postorder - Tree - Walk\,(x)$ . . . 106
21 $Tree - Successor\,(x)$ . . . 108
22 $Tree - Insert\,(T, z)$ . . . 109
23 $Transplant\,(T, u, v)$ . . . 111
24 $Tree - Delete\,(T, z)$ . . . 112
25 $BFS\,(G, s)$ . . . 121
26 $DFS\,(G, s)$ . . . 126
27 $DFS - Visit\,(u)$ . . . 126
28 $Topological - Sort\,(G)$ . . . 130
29 $\mathrm{SCC}\,(G)$ . . . 132

רשימת אלגוריתמים

30 $MST - Kruskal\,(G)$ 138
31 $MST - Prim\,(G)$ 139
32 Relax (u, v) 142
33 Dijkstra $(G = (V, E)\,, s)$ 144
34 Bellmann-Ford (G, s) 149
35 Print-All-Pairs-Shortest-Path (Π, i, j) 155
36 Extend-Shortest-Paths (A, B) 157
37 Square-Matrix-Multiply (A, B) 158
38 Faster-All-shortest-Paths (W) 158
39 Floyd-Warshall (W) 162
40 $MST - Kruskal\,(G)$ 165
41 $Connected - Components\,(G)$ 167
42 $MST - Kruskal\,(G)$ 168
43 $Make - Set\,(x)$ 173
44 $Find - Set\,(x)$ 174
45 $Link\,(x, y)$ 174
46 $Union\,(x, y)$ 174

חלק I

הרצאה $\mathbb{I}$ - מושגי יסוד, סיבוכיות זמן ריצה

לפני שנגלוש לעומק החומר עלינו להכיר מספר מושגי יסוד.

אלגוריתם

אלגוריתם הוא מתכון להשגת מטרה, מתאר את הצעדים שיש לעשות כדי להגיע מהמצב ההתחלתי למטרה. לדוגמה, כאשר יש לנו מספר ירקות נוכל לעקוב אחר מתכון כדי להכין מהירקות סלט.

באלגוריתם של מחשב הקלט $(Input)$ יהיה **מידע** כלשהו, לדוגמה: מערך מספרים, רשימת שמות ומספרי טלפון, תצפיות, מפות, מידע מחיישנים וכו'.

נרצה להפעיל אלגוריתם על הקלט בשביל להגיע ל**מטרה** כלשהי.

מטרות נפוצות

מיון מערכי מספרים, **חיפוש** מספר טלפון השייך לאדם מסויים, **מציאת** המסלול הקצר ביותר בין מספר ערים (בלי לחזור על אותה עיר פעמיים) וכו'.

אלגוריתם הוא המתכון שמעביר אותנו מהמידע למטרה.

$Algorithms\ Vs.\ Programs$

- **אלגוריתם** הוא רעיון אבסטרקטי $(high\ \ level)$, בכתיבת אלגוריתם אנו נגיד דברים כמו "מצא את האיבר הגדול ביותר במערך והעבר אותו לסוף", זאת בניגוד ל**תוכנה**, בה נמצא את המימוש המדויק $(low\ \ level)$, עם הוראות שמחשב יכול להבין).

- **תוכנית** $low\ level$, הוראות ברורות. למשל, תכנית למציאת האיבר המקסימלי במערך והעברתו לסוף.

Algorithm 1 העברת האיבר המקסימלי במערך לסופו

$maxIndex := 0$

$max := A[0]$

for $i := 1$ **to** 3 do**:**

    if $(A[i] > max)$ **then** $maxIndex := i$**:**

$temp = A[3]$

$A[3] = A[maxIndex]$

$A[i] = temp$

במקרה זה נקבל כי $\begin{bmatrix} 1 & 7 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 1 & 4 & 2 & 7 \end{bmatrix} = Arr$.

נעיר כי אנו נתמקד באלגוריתמים.

מבני נתונים

מבנה נתונים הוא דרך לאחסון נתונים במחשב.

בשביל להשיג מטרה באמצעות אלגוריתם, הדרך בה אנו מארגנים ומנהלים את המידע הוא קריטי. על כן, מבני הנתונים ילכו יד ביד עם האלגוריתמים ולכן נדבר כאן הרבה על אלגוריתמים (הם אלה שיקבעו לנו באילו מבני נתונים להשתמש).

חשוב לנו שאלגוריתם ירוץ ביעילות משתי בחינות:

1. **מהירות** - נרצה שהמחשב ישלים פעולות במהירות, ולא נצטרך לחכות זמן רב כדי שיסיים.
2. **שימוש בזכרון** - נרצה שהאלגוריתם שלנו ידרוש כמה שפחות זכרון.

בחירת מבנה נתונים מתאים תוכל לגרום לאלגוריתם להיות יעיל משתי הבחינות הנ"ל.

דוגמה. דוגמה נרצה לראות כיצד משפיעים מבני נתונים על אלגוריתם למציאת המספר המקסימלי מבין אוסף של מספרים. אם האוסף שמור:

(i) ברשימה ממוינת - הפתרון טריוויאלי, האלגוריתם הוא לגשת לאיבר האחרון וזהו.

(ii) ברשימה לא ממוינת - ברור שנצטרך להשתמש בפתרון יותר מסובך, יהיה עלינו לעבור על כל איבר במערך ולמצוא את הגדול ביותר.

שאלה נתונים לנו שני פולינומים, שנסמנם P,Q, ונרצה לכתוב אלגוריתם המחשב את מכפלתם, $P\cdot Q$. איזה מבין מבני הנתונים הבאים יהיה יעיל יותר לאחסון הפולינומים?

(i) מבנה נתונים בו הפולינומים מתוארים כמכפלות של גורמים ליניאריים, לדוגמה: $\underbrace{(x-1)(x+1)(x+2)}_{P}\cdot\underbrace{(x+3)}_{Q}$

(ii) מבנה נתונים בו הפולינומים מתוארים כסכומים, לדוגמה: $\underbrace{(x+3)}_{Q}\cdot\underbrace{(x^3+2x^2-x-2)}_{P}$

$(\star)$ שימו לב שאכן $P=x^3+2x^2-x-2=(x-1)(x+1)(x+2)$.

פתרון. פתרון עבור מבנה (i) קל מאוד לחשב את התוצאה, שכן אנו מתארים את הפולינומים כמכפלות, והתשובה שנחזיר גם היא תיוצג כמכפלות של גורמים ליניאריים.

למשל עבור כפל של שני הפולינומים P,Q, ברור שהתשובה היא $P\cdot Q=(x-1)(x+1)(x+2)(x+3)$.

עבור מבנה (ii), יותר קשה לחשב את התוצאה, שכן בשביל לחשב את $P\cdot Q$ כסכום עלינו לכפול כל איבר ב-P עם כל איבר ב-Q, ואז נקבל $P\cdot Q=x^4+5x^3+5x^2-5x-6$.

שימו לב כיצד הדרך הראשונה הייתה טריוויאלית, ויכלנו לראות אותה בעין, בעוד שבדרך השנייה לא יכלנו לראות מיידית את התוצאה, והיה עלינו להפעיל מספר פעולות. על כן, מבנה (i) יהיה יותר יעיל למטרתנו.

הערה. מסיבה דומה, אם במקום לחשב מכפלה של שני פולינומים היינו רוצים לחשב סכום, מבנה (ii) היה הרבה יותר יעיל. חשבו, באילו עוד דרכים עדיפה הדרך הראשונה על השנייה?

מסקנה. מבני נתונים שונים מתאימים למטרות שונות ויתרונם נקבע ביחס למטרה.

מבנה נתונים מופשט $(ADT - Abstract\ Data\ Structure)$

הגדרה. מבנה נתונים **מופשט** (ADT) הוא תיאור אבסטרקטי של ממשק מבנה נתונים המספר מה מבנה הנתונים צריך לספק.

דוגמה. כשנדבר על ADT נוכל להגיד שאנו משתמשים במבנה נתונים המאפשר למצוא מספר מקסימלי במהירות (למשל, הרשימה הממויינת ממקודם), או שקל למחוק ממנו איברים אך קשה להכניס לתוכו איברים חדשים.

הגדרה. מבנה נתונים **קונקרטי** הוא מימוש של ADT.

כאמור, אנו נתעסק פחות בקוד ובמימוש של מבני נתונים, אלא נתעסק ברמה הרעיונית $(high\ level)$ לכן נשמע את המושג ADT הרבה (לכל ADT יכולים להיות מימושים רבים).

מטרות

- כאן נתעסק בעיקר באלגוריתמי **חיפוש, מיון ומִפתוחַ**.
- אנו ננתח את האלגוריתמים, נמדוד מתמטית את האלגוריתמים ונסיק מי מהם טובים יותר או פחות וננסה לראות האם אלגוריתם שמצאנו הוא הטוב ביותר האפשרי.
- המידות בהן נשתמש להערכת יעילות האלגוריתמים הם **זמן וזכרון**.

דוגמה. דוגמה לבעיה שאין לה אלגוריתם כזה היא בעיית העצירה (לא נרחיב עליה בפורום זה).

נרצה לענות גם על שאלות נוספות כמו האם תמיד קיים פתרון **יעיל** לבעיה כלשהי (עבור בעיית האריזה לא קיימת פתרון יעיל).

1 אלגוריתמים לפתרון בעיות

כדי לפתור בעיה כלשהי נשאל את השאלות הבאות:

- האם **קיים** אלגוריתם הפותר אותה?
  - נדע שלא תמיד, למשל בעית העצירה עליה נלמד בהמשך.
- האם יש אלגוריתם **יעיל** שפותר אותה?
  - נדע שלא בהכרח, אך בהחלט יתכן. דוגמה ידועה היא בעית המיון והאלגוריתם $Bubble\ Sort$ שאינו אופטימלי.
- האם האלגוריתם הוא **היעיל ביותר**?
  - למשל בבעיית המיון נראה שהפתרון היעיל ביותר הוא $\Omega\left(n\log n\right)$ לזמן הריצה ו-$\Omega\left(n\right)$ לזכרון.

◁ הסימונים $\Omega,\Theta,\mathcal{O}$ יוגדרו בהמשך.

בעיות קלות וקשות

נוכל לסדר את הבעיות בהן ניתקל בסקלה לפי קושי הבעיה. להל"ן סקלה כזו עם מספר בעיות מפורסמות (שעל חלקן נדבר בהמשך):

איור 1: סקלה של קושי עבור בעיות מפורסמות

את קושי הבעיה אנו מודדים לפי **מודל חישובי** מסוים.

המודל הסדרתי $(Sequential\ Computational\ Model)$

המודל בו אנו נעסוק הוא המודל הסדרתי, בו כל פעולה באלגוריתם שלנו תתבצע שורה אחר שורה, פעולה אחר שורה, אנו לא נדבר על מצב של מקביליות (קרי כשמספר פעולות רצות במקביל, דבר זה יכול לזרז מאוד אלגוריתמים).

מחלקות שקילות $(Equivalence\ Classes)$

ננסה לאפיין את הבעיות לפי מחלקות שקילות. להלן דוגמה לסיווג למחלקות שקילות (את הסימונים הנ"ל נבין בהמשך):

- $\Omega\left(n\right)$, בעיות חיפוש.
- $\Omega\left(n\log n\right)$, בעיות מיון.
- $\Omega\left(2^{n}\right)$ בעיית האריזה.
- בעיות לא פתורות, עצירה.

בעיות קלות

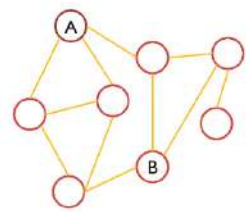
בעיית הנתיב הקצר ביותר

בבעיה זו מטרתנו היא למצוא את הדרך הקצרה ביותר בין נקודה A לנקודה B המייצגות שתי תחנות רכבת כמתואר באיור:

איור 2: שתי תחנות רכבת במפה

- את אורך המסלול נמדוד במספר התחנות שיש בין אותן שתי תחנות.
- במקרה זה, מספר הצעדים לפתרון פרופורציוני למספר כל התחנות.

מבנה הנתונים בו נשתמש כדי לפתור את הבעיה הזו נקרא **גרף**. את מפת התחנות נייצג כגרף כאשר כל תחנה היא צומת בגרף, ובין כל שתי תחנות סמוכות תעבור קשת כמתואר באיור:



איור 3: גרף המייצג את הבעיה

מבנה נתונים זה יעיל ומאפשר לפתור את הבעיה בסיבוכיות של $\mathcal{O}\left(n\right)$ כתלות במספר הקשתות.

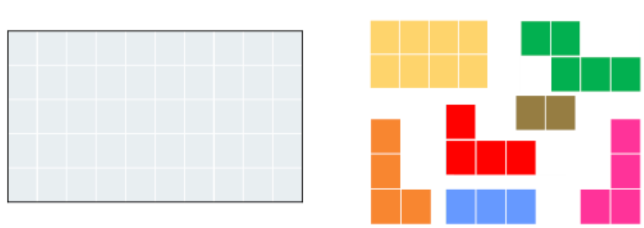
בעיות קשות

בעית האריזה

נרצה לארוז מספר חפצים בתוך ארגז, כך שהחפצים יתפסו כמה שפחות מקום.

נתון:

- הלוח הוא 5×10.
- יש 7 חלקים וסך הכל 30 קוביות המרכיבות אותם.
- החלקים מתוארים באיור יחד עם הלוח:



איור 4: צורת החלקים והלוח

(ב) ניסיון שני לאריזת החלקים

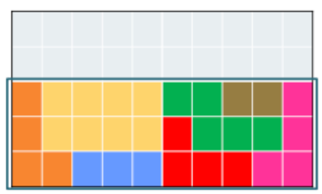
(א) ניסיון ראשון לאריזת החלקים

איור 5 : דרכים אפשריות לאריזה

ננסה לפתור את הבעיה.

- נסיון ראשון : האריזה דורשת קופסה של 40 ריבועים.
- נסיון שני : האריזה דורשת רק 36 ריבועים.

בעיה זו נחשבת לבעיה קשה, מיד נבין מדוע. בינתיים נביט **בפתרון האופטימלי** שדורש רק 30 ריבועים :



איור 6 : אריזה אופטימלית

עתה נרצה לנסות לנסח אלגוריתם לפתרון הבעיה. הדבר הראשון שעולה לנו בראש הוא האלגוריתם הנאיבי הבא :

1. מצא את **כל** האפשרויות.
2. חשב את השטח התפוס.
3. שמור את הקומבניציה הכי טובה.

שאלה כמה קונפיגורציות יש?

תשובה נשים לב שלכל צורה, יש 4 סיבובים שצריך להתחשב בהן, אז צריך לפחות 4^7 אפשרויות (כי יש 7 צורות). באופן כללי יש לפחות 4^n אפשרויות עבור n צורות (שימו לב שלא התחשבנו במיקום של כל צורה, כך שיש אף יותר מ-4^n).
לפונקציה כזו, מהצורה $f(n) = a^n$, אנו קוראים פונקציה אקספוננציאלית.

הערה. לבעיה זו אין פתרון טוב יותר למקרה הגרוע ביותר. במקרה הטוב ביותר כל הצורות יהיו ריבועים בגודל 1, ואז הפתרון קל מאוד. עם זאת, בדר"כ לא נתמקד במקרה הטוב ביותר (שכן לרוב הוא טריוויאלי), אלא במקרה הממוצע/הגרוע ביותר.

1.1 מדדי יעילות

אנו בדרך כלל מתעניינים בסיבוכיות הזמן והזכרון.

על מנת למצוא אותן נניח כי הקלט הוא אחד מהמקרים הבאים:

- המקרה הטוב ביותר.
- המקרה הרע ביותר.
- המקרה הממוצע.

כשנדבר על סיבוכיות נתייחס לסיבוכיות האלגוריתם ואף נתייחס לסיבוכיות הבעיה.

אנו ננסה למצוא חסמים עליונים וחסמים תחתונים ליעילות האלגוריתם ונשאל, מה הכי טוב שנוכל לעשות?

בעיית המיון

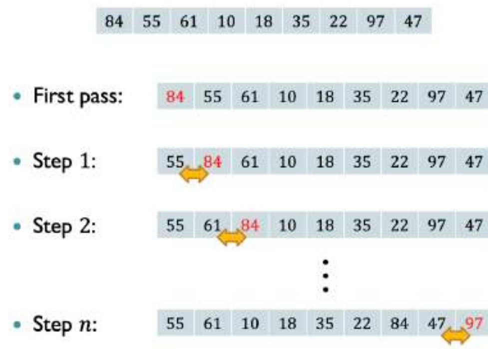
בעיה בהנתן מערך של 4 מספרים נרצה למיין אותם בסדר עולה.

פתרון נאיבי - מיון בועות $(Bubble\ Sort)$

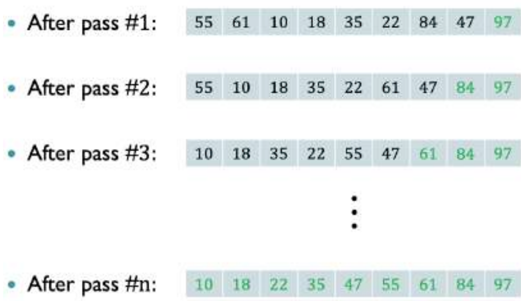
מיון בועות עובד באופן הבא:

- אם הזוג הראשון לא מסודר, תהפוך אותו. תמשיך ככה עד סוף המערך.
  - בסוף מעבר ראשון זה, האיבר המקסימלי הגיע לסוף המערך, למשל: $\begin{bmatrix} 1 & 7 & 2 & 4 \end{bmatrix} \to \begin{bmatrix} 1 & 2 & 4 & 7 \end{bmatrix}$.
- חזור על הפעולה רק שהפעם אל תתייחס לאיבר האחרון במערך.
  - בסוף מעבר שני זה, האיבר המקסימלי במערך "החדש" יהיה במקום האחד לפני האחרון.
- חזור על התהליך עד שתשאר עם מערך בגודל אחד.
  - בסוף שלב זה המערך כבר יהיה ממוין.

ננתח את האלגוריתם על הדוגמה הבאה שמוצגת באיור:



(א) מעבר אחד של האלגוריתם



(ב) שלב שלב באלגוריתם

איור 7: דוגמה לריצת אלגוריתם מיון בועות

לאחר שהבנו ויזואלית מה קורה בכל מעבר, ננתח את המעברים הבאים עד המעבר n (כמתואר באיור ב').

כפי שצוין לעיל, בכל מעבר, האיבר המקסימלי במערך הנותר (תת המערך שלא מכיל את האיברים המקסימלים מהשלבים הקודמים), מפועפע למיקומו האחרון.

נשאלת השאלה מהי סיבוכיות זמן הריצה שלו? ננתח את השלבים:

- מספר המעברים על המערך n פעמים.
- מספר הצעדים בכל מעבר n (למען האמת מספר הצעדים הוא $n, n-1, \ldots, 1$, אבל נתעלם מזה לעת עתה).
- לכן מספר הפעולות הוא n^2 שזו גם סיבוכיות זמן הריצה.
  - מניתוח זה עולה כי סיבוכיות זמן הריצה היא מספר הפעולות שנעשות סך הכל על ידי האלגוריתם.

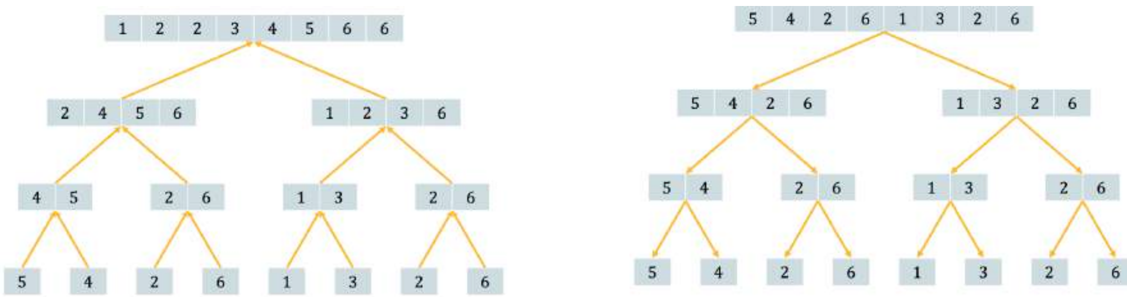
פתרון "יעיל" יותר - מיון מיזוג $(Merge\ Sort)$

באלגוריתם מיון זה אנו ממיינים את המערך על ידי מיון כל חצי ממנו ואיחוד החצאים.

הוראות האלגוריתם הן כדלקמן:

- נפצל את המערך לשני מערכים וכל אחד מהם לשני מערכים וכן הלאה... עד שנקבל n מערכים בגודל אחד (איור א').
- נמיין כל זוג מערכים ונמזג אותם. נחזור על התהליך עם המערכים שהתקבלו עד שנקבל מערך אחד גדול וממויין (איור ב'). $(\star)$

$(\star)$ בשביל למיין כל זוג תתי-מערכים, ניגש לאיבר הראשון בכל מערך ונשאל מי מבין שניהם יותר קטן. את הקטן מביניהם נשים ראשון במערך החדש ונמשיך בתהליך באותו האופן עד שנעבור על כל האיברים.



(א) פיצול המערך		(ב) איחוד תתי-המערכים

איור 8: המחשה לאלגוריתם מיון מיזוג

שאלה גם כאן נשאלת השאלה, מהי סיבוכיות זמן הריצה של האלגוריתם?

תשובה נספור את מספר הפעולות שהוא מבצע. נספור קודם כל כמה רמות יש בעץ (המופיע באיור הנ"ל).

מכיוון שברמה התחתונה של העץ יש מערכים עם איבר אחד בלבד, ובמעבר מרמה לרמה מספר האיברים בכל מערך קטן פי 2, נקבל כי מספר הרמות בעץ, שנסמנו l, הוא המספר שמקיים את התנאי $2^l = n$, כלומר $l = \log_2 n$.

עתה נחשב את מספר הפעולות שנעשות בכל רמה. בכל רמה אנו מאחדים מערכים שמספר איבריהם הכולל הוא n, כלומר בכל רמה יש סה"כ n פעולות.

מכאן נסיק כי סיבוכיות זמן הריצה היא $n \log n$ (את סיבוכיות הזכרון נחשב בהמשך והיא $\mathcal{O}\left(n \log n\right)$).

השוואת מיון בועות למיון מיזוג

נרשום את הנתונים בטבלה (את רובם לא הוכחנו)

אלגוריתם	זכרון	זמן (טוב ביותר)	זמן (גרוע ביותר)	זמן (ממוצע)
מיון בועות	n	n מעבר יחיד	n^2 n מעברים	$\frac{n^2}{2}$ $\frac{n}{2}$ מעברים
מיון מיזוג	$n\log n$	$n\log n$	$n\log n$	$n\log n$

איור 9: השוואה בין מיון בועות למיון מיזוג

נשאלת השאלה איזה אלגוריתם טוב יותר? התשובה היא שמיון מיזוג, שכן $n \log n < n^2$ כאשר n גדול מספיק. נאמר זאת עם סייג, מכיוון שסיבוכיות הזכרון של מיון בועות נמוכה יותר (נעיר כי ניתן לממש את מיון מיזוג בצורה כזו שסיבוכיות

הזכרון שלו תהיה $\mathcal{O}\left(n\right)$ ולא $\mathcal{O}\left(n\log n\right)$, כפי שנראה בהמשך).

נקודה חשובה העולה מכאן, היא שהתשובה לשאלה "איזה אלגוריתם יותר טוב" היא איננה שחור או לבן, צריך להחליט מה יותר חשוב: סיבוכיות זמן? סיבוכיות מקום? האם יותר חשוב לנו להתייחס למקרה הממוצע או הגרוע? וכו'.

שאלות להמשך

- איך סיבוכיות של אלגוריתם מתנהגת באופן אסימפטוטי?
  - איך אלגוריתמים רקורסיבים מתנהגים? האם יש שיטות לטפל בהם?
- האם יש דרך למצוא חסמים עליונים ותחתונים לסיבוכיות האלגוריתם? לבעיה?

נושאי הקורס

- מיון: מיזוג, מהיר, סופר, בסיס, דליים.
- ערימות, תורים עם עדיפות, עצים בינארי, עצי AVL
- טבלות גיבוב, פונקציות גיבוב, קוד $Huffman$.
- אלגוריתמים בגרפים.

כל האלגוריתמים יתוארו בפסודו קוד.

2 סיבוכיות

כשרוצים לנתח אלגוריתם ורוצים לדעת כמה זמן לוקח לו לרוץ נשאלת השאלה, איך מודדים זאת?

הפתרון הנאיבי הוא למדוד זאת באמצעות שעון.

פתרון זה מקומי, שכן מחשבים שונים פועלים במהירות שונה ואם יצא מחשב חדש הזמן יהיה מהיר יותר, כלומר הסיבוכיות תהיה תלויה במכשיר.

כדי לעשות זאת בצורה טובה, נמדוד את היחידות הבסיסיות לזכרון וזמן ריצה. מנייה כזו היא בלתי-תלויה במכשיר.

יתר על כן, הסיבוכיות תכתב כפונקציה של אורך הקלט n ולא של ערך ספציפי, שכן יתכן כי עבור ערך כנ"ל, האלגוריתם ירוץ הרבה יותר מהר/לאט ממקרים אחרים עם אורך קלט באותו סדר גודל.

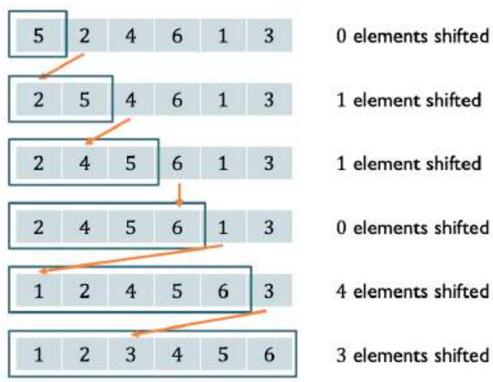
על כן, נבדוק מה קורה כאשר n גדל ונשים לב ש-n-ים קטנים לא מאוד מעניינים.

מיון הכנסה $Insertion\ Sort$

נביט במערך $[5, 2, 4, 6, 1, 3]$. מיון הכנסה ימיין אותה לפי הצעדים הבאים :

- נגדיר רישא בגודל אחד. בתוך הרישא נכניס את האיבר האחרון למיקום המתאים לו ונדחוף את שאר האיברים בהתאם.
- נגדיל את הרישא באחד ונחזור לשלב הראשון.
- נעצור כאשר סיימנו לעבור על המערך וקיבלנו רישא הכוללת את כל המערך.

נביט בדוגמה למיון זה :



איור 10 : צעדי פעולות האלגוריתם מיון הכנסה

נסתכל על כל אחד מהצעדים וננתח אותם :

1. איבר אחד, הוא נשאר במקומו.
2. $2 > 5$ לכן יש הכנסה של 2 למיקום הראשון ברישא ודחיפה של 5 קדימה.
3. $4 > 5$ לכן 4 מוכנס למיקום של 5 ו-5 נדחף קדימה.
4. $6 < 5$ לכן אין שינוי.
5. $1 > 2 > 6$ לכן 1 מוכנס למיקום של 2 וכל השאר נדחפים קדימה (סך הכל 4 דחיפות)
6. $3 < 6$ לכן 3 מוכנס למיקום המתאים לו שזה המיקום של 4 ולכן $4, 5, 6$ נדחפים קדימה.

לאחר שהבנו מה האלגוריתם מבצע, נביט בפסודו קוד שלו:

Algorithm 2 מיון הכנסה

```
1 : for j ← 2 to A.length do
2 :     key ← A[j]
3 :     // insert A[j] into A[1...j − 1]
4 :     i ← j − 1
5 :     while i > 0 and A[i] > key do
6 :         A[i + 1] ← A[i]
7 :         i ← i − 1
8 :     A[i + 1] ← key
```

שאלה מהי סיבוכיות זמן הריצה של האלגוריתם?

תשובה נסמן את הסיבוכיות של כל שורה ב-$c_1, \ldots, c_8$ ונמצא את מספר הפעמים שהיא מתבצעת:

	Insertion-Sort(A) // Sort an array A of n integers	Cost	Times
1	for $j \leftarrow 2$ to A.length do	c_1	n
2	$key \leftarrow A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into $A[1..j-1]$		
4	$i \leftarrow j-1$	c_4	$n-1$
5	while $i > 0$ and $A[i] > key$ do	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] \leftarrow key$	c_8	$n-1$

איור 11: סיבוכיות מיון הכנסה

כאשר t_j מסמן את מספר הפעמים שהשורה בוצעה באיטרציה ה-j (t_j תלוי באיברי הקלט עצמם, לכן הוא מסומן כנעלם).

נעבור על כל שורה ונבין מדוע מספר הפעמים נכון:

1. הלולאה רצה $n-1$ פעמים, אך השורה מתבצעת n פעמים, שכן בפעם האחרונה היא עוצרת.
2. הלולאה רצה $n-1$ פעמים לכן השורה השנייה תתבצע $n-1$ פעמים.
3. הערה, אין לה סיבוכיות.
4. כמו שורה 2.

5. הביטוי $\sum_{j=2}^{n} t_j$ הוא מספר הפעמים ששורת התנאי של הלולאה רצה .
6. הביטוי $\sum_{j=2}^{n} (t_j - 1)$ מכיל בפנים את $t_j - 1$ מכיוון שהלולאה רצה $t_j - 1$ פעמים.
7. כמו שורה 6.
8. כמו שורה 2.

סיבוכיות זמן הריצה היא הסכום של כל זמני הריצה של כל ההוראות.

כל שורה עולה c_i יחידות זמן ומספר הפעמים שהיא רצה הוא t_i לכן זמן הריצה הכולל של האלגוריתם הוא $T(n) = \sum_{i=1}^{k} c_i t_i$ (במקרה שלנו $k = 8$). לכן

$$T(n) = c_1 n + c_2 (n-1) + c_4 (n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$$

נבחן את זמן הריצה, ונתרכז באיברים האדומים שכן הם אלה שתלויים בקלט :

- **המקרה הטוב היותר:** נבין כי המקרה הטוב ביותר הוא כאשר המערך ממוין. במקרה זה :
  - צעדים $6,7$ לא מבוצעים (שכן אם המערך ממוין לא נצטרך להזיז איברים, ואז לא נכנסים לתוך ה-$while$ ולא מבצעים את פעולות $6,7$).
  - צעד 5 מבוצע פעם אחת בכל לולאה ולכן מתבצע $n-1$ פעמים סה"כ.

  לכן נקבל כי זמן הריצה הכולל במקרה זה הוא

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8) n - (c_2 + c_4 + c_5 + c_8)$$

קיבלנו פונקציה **לינארית**.

- **המקרה הגרוע היותר:** נבין כי במקרה זה המערך ממוין בסדר הפוך. במקרה זה :
  - $t_j = j$.
  - צעד 5 מתבצע j פעמים בכל איטרציה ולכן לוקח סך הכל $\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$.
  - צעדים $6,7$ יהיו $\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$ כל אחד.

לכן סה"כ קיבלנו פונקציה **ריבועית**:

$$T\left(n\right)=\frac{c_1+c_6+c_7}{2}n^2+\left(\frac{c_1+c_2+c_4+c_8+c_5-c_6-c_7}{2}\right)n-\left(c_2+c_4+c_5+c_8\right)$$

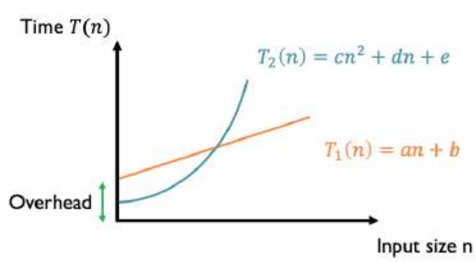
ניתוח אסימפטוטי

מה שמעניין אותנו הוא ניתוח אסימפטוטי, כלומר נרשום $T\left(n\right)=f\left(n\right)$ כאשר f כוללת קבועים, ואנו נרצה למצוא התנהגות אסימפטוטית שלה.

נשאל, כיצד פונקציה $f\left(n\right)$ מתנהגת באופן אסימפטוטי?

- עבור קלטים קטנים, זמן הריצה קטן.
- נרצה לדעת מה קורה עבור קלטים גדולים.

נשווה בין שני המקרים באלגוריתם מיון הכנסה באמצעות שרטוט הגרפים של שני המקרים כתלות ב-n:



איור 12: השוואה בין המקרה הטוב ביותר למקרה הגרוע ביותר במיון הכנסה

- נסיק כי עבור n **גדול** מתקיים כי $T_2\left(n\right)>T_1\left(n\right)$ ללא תלות בגודל הקבועים.

נסכם את מה שראינו עד עכשיו.

- כאשר מדברים על יעילות של אלגוריתם מה שמעניין אותנו הוא התנהגות אסימפטוטית כתלות באורך הקלט.
- אנו מתעניינים בסיבוכיות זמן ומקום.
- אנו מתעניינים במקרה **הגרוע ביותר**, במקרה **הטוב ביותר** ובמקרה **הממוצע**. (למרות שרוב הזמן נתמקד במקרה הרע והממוצע)

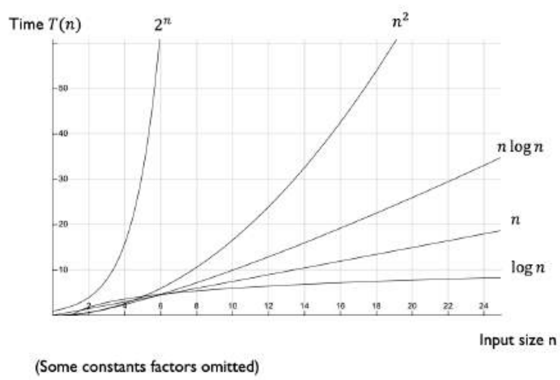
סיבוכיות זמן ריצה וזיכרון

- נסמן ב-$T\left(n\right)$ את סיבוכיות זמן הריצה.
  - מספר הביצוע של פעולות.
  - n הוא גודל הקלט.

- נסמן ב-$S\left(n\right)$ את סיבוכיות המקום.
  - גודל תאי הזכרון הדרושים.
  - n הוא גודל הקלט.

נרצה לדעת מהו **קצב הגידול** של S,T. קרי, כמה מהר $T\left(n\right),S\left(n\right)$ גדלים?

עתה נראה השוואה בין סדרי גודל שונים של סיבוכיות:



איור 13: השוואה בין פונקציות שונות של סיבוכיות

2.1 חסמים אסימפטוטים

- חסם מלעל $\mathcal{O}\left(f\left(n\right)\right)$: לכל היותר $f\left(n\right)$ פעולות.
- חסם מלרע $\Omega\left(g\left(n\right)\right)$: לכל הפחות $g\left(n\right)$ פעולות.
- חסם הדוק $\Theta\left(h\left(n\right)\right)$: בדיוק $h\left(n\right)$ פעולות.

חסם מלעל $\mathcal{O}$ ("או" גדול)

הגדרה. עבור $g:\mathbb{N}\to\mathbb{R}^{+}$ נגדיר $\mathcal{O}\left(g\left(n\right)\right)=\left\{ f\left(n\right):\mathbb{N}\to\mathbb{R}^{+}\mid\exists c>0,n_{0}>1:\ \forall n\geq n_{0},\ f\left(n\right)\leq c\cdot g\left(n\right)\right\}$

כלומר אם $f=\mathcal{O}\left(g\left(n\right)\right)$ (f היא $\mathcal{O}$ של g) אז החל ממקום מסוים f קטנה יותר מ-g כפול קבוע.

חסם מלרע Ω (אומגה גדולה)

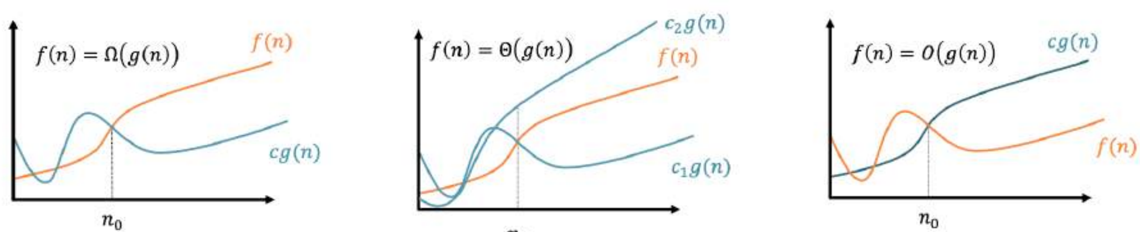
הגדרה. עבור $g:\mathbb{N}\to\mathbb{R}^{+}$ נגדיר $\Omega\left(g\left(n\right)\right)=\left\{ f\left(n\right):\mathbb{N}\to\mathbb{R}^{+}|\,\exists c>0,n_{0}>1:\ \forall n\geq n_{0},\ f\left(n\right)\geq c\cdot g\left(n\right)\right\}$

כלומר אם $f=\Omega\left(g\left(n\right)\right)$ (f היא Ω של g) אז החל ממקום מסוים f גדולה יותר מ-g כפול קבוע.

חסם הדוק Θ (תטא גדולה)

הגדרה. עבור $g:\mathbb{N}\to\mathbb{R}^{+}$ נגדיר $\Theta\left(g\left(n\right)\right)=\left\{ f\left(n\right):\mathbb{N}\to\mathbb{R}^{+}|\,\exists c_{1},c_{2}>0,n_{0}>1:\ \forall n\geq n_{0},\ c_{1}\cdot g\left(n\right)\leq f\left(n\right)\leq c_{2}\cdot g\left(n\right)\right\}$

כלומר אם $f=\Theta\left(g\left(n\right)\right)$ (f היא Θ של g) אז החל ממקום מסוים f קטנה יותר מ-g כפול קבוע, אך גדולה יותר מ-g כפול קבוע אחר. קרי, f מתנהג בערך כמו g אסימפטוטית.



(א) המחשה ל-$\mathcal{O}$ (ב) המחשה ל-Θ (ג) המחשה ל-Ω

איור 14: חסמים אסימפטוטיים

תכונות של חסמים אסימפטוטים

משפט. $f\left(n\right)=\mathcal{O}\left(g\left(n\right)\right)$ וגם $f\left(n\right)=\Omega\left(g\left(n\right)\right)$ אם"ם $f\left(n\right)=\Theta\left(g\left(n\right)\right)$.

- **רפלקסיביות** $f\left(n\right)=\Theta\left(f\left(n\right)\right);f\left(n\right)=\Omega\left(f\left(n\right)\right);f\left(n\right)=\mathcal{O}\left(f\left(n\right)\right)$.
- **סימטריות** $f\left(n\right)=\Theta\left(g\left(n\right)\right)$ אם"ם $g\left(n\right)=\Theta\left(f\left(n\right)\right)$.
- **טרנזיטיביות**
  - אם $f\left(n\right)=\mathcal{O}\left(g\left(n\right)\right)$ וגם $g\left(n\right)=\mathcal{O}\left(h\left(n\right)\right)$ אז $f=\mathcal{O}\left(h\left(n\right)\right)$.
  - אם $f\left(n\right)=\Omega\left(g\left(n\right)\right)$ וגם $g\left(n\right)=\Omega\left(h\left(n\right)\right)$ אז $f=\Omega\left(h\left(n\right)\right)$.
  - אם $f\left(n\right)=\Theta\left(g\left(n\right)\right)$ וגם $g\left(n\right)=\Theta\left(h\left(n\right)\right)$ אז $f=\Theta\left(h\left(n\right)\right)$.
- **לינאריות** $\mathcal{O}\left(f\left(n\right)+g\left(n\right)\right)=\mathcal{O}\left(f\left(n\right)\right)+\mathcal{O}\left(g\left(n\right)\right)$
- **כפל** $\mathcal{O}\left(f\left(n\right)\cdot g\left(n\right)\right)=\mathcal{O}\left(f\left(n\right)\right)\cdot\mathcal{O}\left(g\left(n\right)\right)$.

תכונות נוספות

- $\mathcal{O}\left(\mathcal{O}\left(f\left(n\right)\right)\right)=\mathcal{O}\left(f\left(n\right)\right)$.
- $\mathcal{O}\left(f\left(n\right)+g\left(n\right)\right)=\mathcal{O}\left(f\left(n\right)\right)+\mathcal{O}\left(g\left(n\right)\right)$.
- $1\neq\forall b>0,\ \mathcal{O}\left(\log_{b}n\right)=\mathcal{O}\left(\lg n\right)$
  כאשר $\lg=\log_{2}$.
- פולינומים: $\sum_{i=1}^{k}a_{i}n^{i}=\Theta\left(n^{k}\right)$ כאשר $a_{k}>0$.

◁ דוגמה:

$$n^{4}+2n^{2}-5n+7=\begin{array}{l}\mathcal{O}\left(n^{5}\right)\\ \mathcal{O}\left(n^{4}\right)\\ \Theta\left(n^{4}\right)\\ \Omega\left(n^{3}\right)\end{array}.$$

הערה. מטעמי פשטות אנו נוהגים לכתוב $f\left(n\right)=\mathcal{O}\left(g\left(n\right)\right)$ כאשר הכוונה היא שמתקיים $f\left(n\right)\in\mathcal{O}\left(g\left(n\right)\right)$. כך גם עבור Ω,Θ.

שימוש בסיבוכיות

המטרה שלנו היא להסיק את סיבוכיות זמן הריצה באמצעות שימוש בפונקציות אסימפטוטיות.

אנו נראה שלמצוא חסמים תחתונים זאת משימה קשה יותר ממציאת חסמים עליונים ומכאן - חסם הדוק זאת המשימה הקשה ביותר.

שאלה האם ניתן לחסום כל אלגוריתם כדי לפתור בעיה?

תשובה בשביל זה צריך חסמים על בעיות שזה דבר שנראה רק בהמשך.

חלק II

הרצאה $\mathbb{III}$ - שיטת האיטרציה, שיטת ההצבה ומשפט האב

3 פתרון נוסחות נסיגה

נרצה לתת מענה לבעיות הבאות:

- כתיבת **ביטוי** לזמן הריצה של אלגוריתם $T\left(n\right)$.
- **למצוא** פונקציה $f\left(n\right)$ שתקיים $T\left(n\right)=\Theta\left(f\left(n\right)\right)$.
- **פתירת** נוסחות נסיגה רקורסיביות לזמן ריצה (הפרד ומשול).

ניתוח זמן ריצה של אלגוריתם

ראינו כבר שניתן לנתח אלגוריתם על ידי **מחיר** של כל פעולה c_j **ומספר הפעמים** שהיא מבוצעת t_j ושזמן הריצה הכולל הוא $T\left(n\right)=\sum c_jt_j$, למשל עבור לולאות. אבל ביטוי זה מעט מסובך עבור אלגוריתמים מורכבים יותר.

נוחסאות רקורסיביות

בהנתן נוסחה רקורסיבית של זמן ריצה, למשל מהצורה $T\left(n\right)=T\left(n-1\right)+c$ מתקיים כי:

- $T\left(1\right)=\Theta\left(1\right)$ הוא קבוע.
- הפתרון הוא $T\left(n\right)=\mathcal{O}\left(n\right)$.

פתרון זה מאוד פשוט להוכחה, אמנם מה בדבר נוסחות כמו $T\left(n\right)=2T\left(\frac{n}{2}\right)+\mathcal{O}\left(n\right)$ בה נתקלנו במיון מיזוג? מיד נראה.

לשם פשטות נניח כי הקלטים אינם יוצרים **מקרי קצה**:

- n גדול כדי להימנע ממקרי קצה לא מייצגים.
- $T\left(1\right)=\Theta\left(1\right)$ לשם פשטות.
- אם אנו מבצעים את הפעולה $\frac{n}{2}$ באופן סדרתי נניח כי $n=2^k$, אם אם זה קורה פעם אחת נניח כי $n=2k$.

נעיר כי למרות שבעבר אמרנו שאפשר להתעלם מקבועים, כאן זה לא המצב בהכרח. למשל, בנוסחה $T\left(n\right)=T\left(n-1\right)+c$, אם נתעלם מ-$c$ נקבל כי $T\left(n\right)=T\left(1\right)$ וזה לא המצב כמובן.

הפרד ומשול

בהנתן אלגוריתם רקורסיבי, נשאל את עצמנו את השאלות הבאות:

1. לכמה **חלקים** הבעיה מחולקת בכל פעם?
2. מה **גודל** הבעיה החדשה בכל חלק?
3. כמה **עבודה** נעשית כדי **לאחד** את תוצאות תת הבעיות לכדי תוצאה אחת?

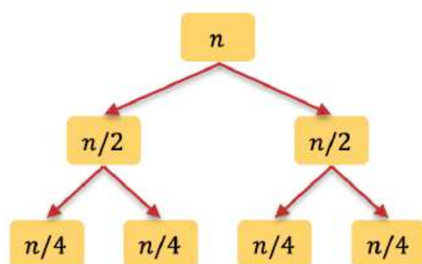
שאלה מספר 1 תוביל אותנו ליצירת "עץ רקורסיה" שכל קודקוד בו ייצג תת בעיה והענפים ממנו מטה ייצגו תתי בעיה שלו.

שאלה מספר 2 תוביל אותנו לקבוע מה הגודל של כל קודקוד בעץ.

שאלה מספר 3 תעזור לנו לקבוע מה העבודה הכוללת שנעשית.

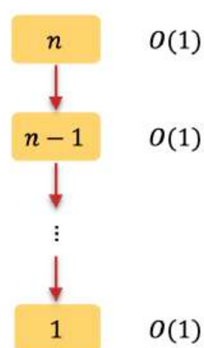
דוגמות

למשל, באלגוריתם מיון, ניתן למיין כל חצי מערך בנפרד (באופן רקורסיבי) ולקבל:



איור 15: עץ רקורסיה פשוט

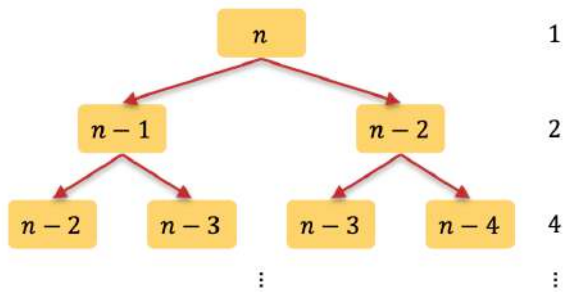
דוגמה. נביט למשל בפונקציה העצרת. מתקיים כי $n! = n(n-1)!$ ולכן מכיוון שכפל הוא בעל זמן ריצה קבוע, נקבל כי סיבוכיות זמן הריצה היא $T(n) = T(n-1) + \mathcal{O}(1)$. נוסחה זו תוביל אותנו לעץ הבא (שנראה כמו ענף ארוך)



איור 16: עץ רקורסיה לחישוב $n!$

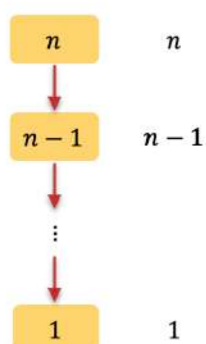
ואינטואיטיבית, נקבל כי $T\left(n\right)=\mathcal{O}\left(n\right)$. תוצאה זו נכונה גם לחיפוש סדרתי.

דוגמה. חישוב איבר פיבונצ'י ברמה ה-n ניתן לחישוב על ידי הנוסחה $fib\left(n\right)=fib\left(n-1\right)+fib\left(n-2\right)$ ולכן נתאים נוסחת נסיגה לזמן הריצה שלו $T\left(n\right)=T\left(n-1\right)+T\left(n-2\right)+\mathcal{O}\left(1\right)$. נוסחה זו תוביל אותנו לעץ מאוד רחב ועמוק של עלים שגדל כל פעם פי 2 בכל רמה. בקרוב נראה כי נוסחה זו תתן לנו $T\left(n\right)=\mathcal{O}\left(2^{n}\right)$.



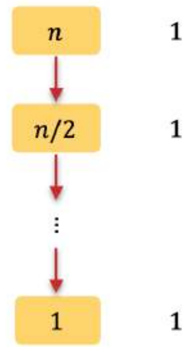
איור 17: עץ רקורסיה לסדרת פיבונצ'י

מיון הכנסה. מכיוון שבכל שלב אנו ממקמים איבר במקום הרצוי ופותרים את הבעיה עם שאר האיברים, נדרשת הזזה של איברים והיא $\mathcal{O}\left(n\right)$. לכן נקבל כי $T\left(n\right)=T\left(n-1\right)+\mathcal{O}\left(n\right)$. בשונה ממה שראינו עד עכשיו, איחוד תתי הבעיות אינו קבוע, וסך הכל נקבל כי זאת סדרה חשבונית: $1+2+\ldots+n=\frac{n(n+1)}{2}=\frac{1}{2}\left(n^{2}+n\right)$ ולכן $T\left(n\right)=\mathcal{O}\left(n^{2}\right)$. עץ הרקורסיה המתאים לבעיה זו הינו:



איור 18: עץ רקורסיה למיון הכנסה

דוגמה. חיפוש בינארי. בכל פעם אנו מחפשים בחצי מהמערך המקורי כאשר חיפוש הוא $\mathcal{O}\left(1\right)$ ולכן סך הכל $T\left(n\right)=T\left(\frac{n}{2}\right)+\mathcal{O}\left(1\right)$. הפתרון לנוסחה זו הוא $T\left(n\right)=\mathcal{O}\left(\log n\right)$ ועץ הרקורסיה המתאים לבעיה הינו:



איור 19: עץ רקורסיה לחיפוש בינארי

דוגמה. מיון מיזוג. במיון זה אנו מפצלים כל מערך לשניים ועושים עליו את הפעולה המקורית ואז אנו מאחדים בין התוצאות, איחוד שלוקח $\mathcal{O}\left(n\right)$. לכן נוסחת הנסיגה היא $T\left(n\right)=2T\left(\frac{n}{2}\right)+\mathcal{O}\left(n\right)$. מכיוון שבכל רמה יש n פעולות סך הכל ועומק העץ הוא $\log n$ נקבל כי $T\left(n\right)=\Theta\left(n\log n\right)$ (בקרוב נראה איך פותרים נוסחה זו באופן פורמלי).

3.1 פתרון נוסחות רקורסיביות

כפי שראינו עד כה, כדי לפתור נוסחות כנ"ל צריך:

- למצוא קודם כל את הנוסחא.
- לפתור אותה.

נציג כמה שיטות כדי לפתור את שתי בעיות אלה.

שיטת ההצבה $Substitution$

בשיטת ההצבה אנו מנחשים פתרון ואז מוכיחים באינדוקציה את נכונותו.

דוגמה. מיון מיזוג. ננחש שהפתרון הוא $\mathcal{O}\left(n\log n\right)$ ונוכיח באינדוקציה את נכונותו. אם כך, נוכיח כי קיים $c>0$ כך שהחל ממקום מסוים $T\left(n\right)\leq cn\log n$. נבחר $n_0\geq 2$ ונבחן אילו תנאים c צריך לקיים כדי שהוכחה שלנו תעבוד.

בסיס: $n=2$ מתקיים כי $T\left(2\right)=2T\left(1\right)+2=4\underbrace{\leq}_{\text{נדרוש}}c2\cdot\log 2=2c$ ולכן נבחר $c\geq 2$. (אם לא נאמר אחרת על $\log$ בסיס 2)

שלב: נניח שהטענה נכונה עבור $1,\ldots,n-1$ נוכיח עבור n. נציב (מכאן בא השם "שיטת ההצבה") את הניחוש בנוסחה באמצעות הנחת האינדוקציה:

$$T\left(n\right)=2T\left(\frac{n}{2}\right)+n\leq 2\left(c\frac{n}{2}\log\frac{n}{2}\right)+n=cn\log\frac{n}{2}+n=cn\log n-cn\log 2+n=cn\log n-cn+n\leq cn\log n$$

עבור $c>1$. אם כך הטענה נכונה לכל $c>1$ (שימו לב שגם $c\geq 2$).

הערה. שיטה זו עבודת בהנחה שיש לנו ניחוש או ביטוי נתון, אך מה אם אין לנו ניחוש?

שיטת האיטרציה

במקרה בו אנו רוצים למצוא "ניחוש מושכל" לנוסחת זמן ריצה, נבצע את הפעולות הבאות:

- נרשום כמה איברים של הנוסחא.
- נמצא תבנית לאיבר הכללי.
- נשתמש בתבנית כדי להגיע לתוצאה הסופית ללא הופעות רקורסיביות.

דוגמה. מיון הכנסה. ידוע כי $T(n)=T(n-1)+n$ לכן נקבל כי $\begin{array}{c} T(n-1)=T(n-2)+n-1 \\ T(n-2)=T(n-3)+n-2 \end{array}$. נציב תוצאות אלה בתוצאה הסופית ונקבל

$$\begin{aligned} T(n) &= T(n-1)+n \\ &= T(n-2)+(n-1)+n \\ &= T(n-3)+(n-2)+(n-1)+n \\ &\vdots \\ &= T(n-k)+\sum_{i=1}^{k}(n-i+1) \\ &= T(n-k)+nk-\frac{k(k-1)}{2} \end{aligned}$$

קיבלנו את התבנית $T(n)=T(n-k)+nk-\frac{k(k-1)}{2}$ עבור כל שלב $1\leq k<n$. אנו יכולים להוכיח כי התבנית נכונה באינדוקציה ואז להציב את השלב ה-$n-1$ כדי להגיע לאיטרציה הסופית ($T(n-k)=T(1)$ ולכן נקבל נוסחה מפורשת).

אך אפשר גם לקבל את הביטוי לאיטרציה הסופית ואז להוכיח את נכונותו בשיטת ההצבה. נציב ונקבל

$$T(n)=1+n^2-n-\frac{n^2}{2}+\frac{3n}{2}-1=\frac{n^2}{2}+\frac{n}{2}=\mathcal{O}\left(n^2\right)$$

הבהרה בהחלט **ניתן** להוכיח נכונות לתבנית באינדוקציה ואז להשתמש בה כדי למצוא ביטוי מפורש ל-$T(n)$ והוא יהיה נכון (כי התבנית נכונה).

הבהרה אם החלטנו להשתמש בתבנית כדי לנחש ביטוי אותו נוכיח באינדוקציה, אנו חוזרים לשיטת ההצבה. במילים אחרות, תהליך הפתרון הוא להל"ן

$$\boxed{\text{Iteration}} \rightarrow \boxed{\text{Substitution}}$$

4 משפט האב $Master\ Theorem$

עד כה מצאנו דרכים לפתרון נוסחות נסיגה רקורסיביות. הן עובדות, אך במקרים מסוימים הן מאוד מסובכות. נציג עתה שיטה ישירה לפתרון הנוסחה ללא ניחוש וללא הצבה.

בהנתן נוסחת נסיגה העונה על תנאים מסוימים, משפט האב מאפשר למצוא את הפתרון שלה באופן מיידי. נציג את המשפט ללא כל פרטיו, נפתחו, נציגו באופן מלא ונוכיח אותו.

לקבלת משפט האב הקהל מתבקש לקום.

משפט. (משפט האב) יהיו $a, b \geq 1, c \geq 0$ קבועים. נניח כי $T(n) = aT\left(\frac{n}{b}\right) + n^c$ וכי $T(1) = \Theta(1)$. אזי:

(i) אם $\log_b a < c$ מתקיים כי $T(n) = \Theta(n^c)$.

(ii) אם $\log_b a = c$ מתקיים כי $T(n) = \Theta(n^c \log_b n)$.

(iii) אם $\log_b a > c$ מתקיים כי $T(n) = \Theta\left(n^{\log_b a}\right)$.

נרצה למצוא $f(n)$ עבורה $T(n) = \Theta(f(n))$. נרצה להבין מה המשמעות של המשתנים a, b, c ומהי השפעתם על הפתרון? נבין כי:

- a הוא מספר החלוקות של כל בעיה.
- b הוא גודל תת הבעיה.
- n^c הוא העבודה הנדרשת לאיחוד כל תתי הבעיות לבעיה אחת (לכן מייצג את העלות של קודקוד בעץ).

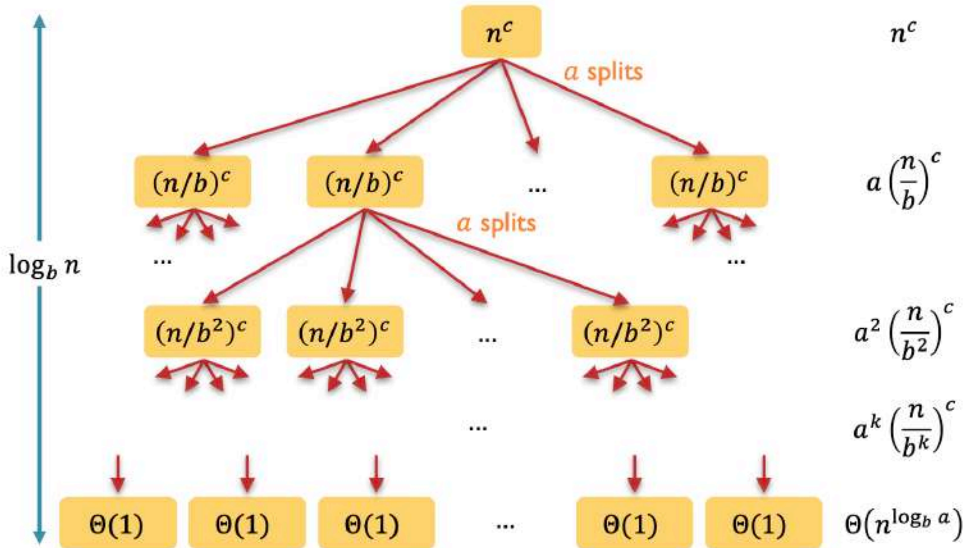
אינטואיטיבית, אם a מאוד גדול, יש המון עלים בעץ, ולכן נצפה שהוא יקבע את ההתנהגות של $T(n)$ ולא c. לעומת זאת, אם a קטן ביחס ל-c נצפה ש-c יהיה יותר דומיננטי, ואם הם באותו סדר גודל, נצפה ששניהם ישפיעו.

ניתן לראות זאת בדוגמות הבאות:

1. $T(n) = 60T\left(\frac{n}{2}\right) + n^1$. המון עלים, מעט עבודה לאיחוד תת בעיות.
2. $T(n) = 2T\left(\frac{n}{16}\right) + n^4$. הרבה עבודה, מעט עלים.
3. $T(n) = 2T\left(\frac{n}{2}\right) + n^1$. אותו סדר גודל של עלים ועבודה (המונח סדר גודל יובהר בהמשך).

הבנה אינטואיטיבית של משפט האב

כדי להבין את משפט האב נשתמש בשיטת האיטרציה. תחילה נשרטט עץ רקורסיה לבעיה:



איור 20: עץ רקורסיה לנוסחה $T\left(n\right)=aT\left(\frac{n}{b}\right)+n^c$

נשים לב כי נגיע ל-$T\left(1\right)$ לאחר $\log_b n$ רמות, וכי ברמה ה-k הסיבוכיות של כל קודקוד היא $\left(\frac{n}{b^k}\right)^c$ (היא הסיבוכיות של איחוד כל תתי הבעיות לכדי תוצאה אחת). נרצה לדעת מהו מספר העלים ברמה האחרונה. מכיוון שבכל רמה אנו מקבלים a כפול מספר כל העלים ברמה הקודמת, נקבל כי מספר העלים הוא בקירוב $\prod_{i=1}^{\log_b n} a=a^{\log_b n}$. אבל $\log_b n=\log_a n\cdot\log_b a$ ולכן $a^{\log_b n}=a^{\log_a n\log_b a}=n^{\log_b a}$. מכאן נסיק כי יש $\Theta\left(n^{\log_b a}\right)$ עלים בעץ. באמצעות גודל זה ותבנית לזמן הריצה ברמה ה-k (נסיק אותו מיד) נוכל לקבל נוסחה מלאה לנוסחת הנסיגה. נסכם את מה שמצאנו עד כה:

- מספר הרמות: $\log_b n$.
- עבודה בשורש לאיחוד כל תתי הבעיות שלו: n^c.
- עבודה כוללת ברמה ה-k: $a^k\left(\frac{n}{b^k}\right)^c$ שכן יש ברמה זו $\Theta\left(a^k\right)$ עלים.
- עבודה ברמה הנמוכה ביותר: $\Theta\left(n^{\log_b a}\right)$.

מנתונים אלה נסיק כי

$$\boxed{T\left(n\right)=\Theta\left(n^{\log_b a}\right)+\sum_{i=0}^{\log_b n-1}a^i\left(\frac{n}{b^i}\right)^c}$$

זאת הנוסחה שננחש כפתרון למשפט האב. בשלב זה סיימנו את שלב האיטרציה.

לפני שנציג את המשפט המלא, ניתן כמה דוגמות לאינטואיציה:

- העבודה בשורש דומיננטית: $T\left(n\right)=2T\left(\frac{n}{3}\right)+n^{700}$. במקרה זה $T\left(n\right)=\Theta\left(n^{c}\right)$.
- העלים דומיננטים: $T\left(n\right)=16T\left(\frac{n}{2}\right)+n^{2}$. במקרה זה $T\left(n\right)=\Theta\left(n^{\log_{b}a}\right)$.
- העלים והעבודה דומיננטים: $T\left(n\right)=2T\left(\frac{n}{2}\right)+n^{1}$. במקרה זה $T\left(n\right)=\Theta\left(n^{c}\log_{b}n\right)$ שכן יש $\log_{b}n$ רמות ו-n^{c} עבודה כוללת בכל רמה.

מסקנות ממשפט האב

1. **מיון מיזוג** $T\left(n\right)=2T\left(\frac{n}{2}\right)+n$. מתקיים כי $a=2,b=2,c=1$ ולכן $\log_{b}a=\log_{2}2=1=c$ ולכן $T\left(n\right)=\Theta\left(n\log_{2}n\right)$.
2. **חיפוש בינארי** $T\left(n\right)=T\left(\frac{n}{2}\right)+1$. מתקיים כי $a=1,b=2,c=0$ ולכן $\log_{b}a=\log_{2}1=0=c$ ולכן $T\left(n\right)=\Theta\left(\log_{2}n\right)$.
3. $T\left(n\right)=16T\left(\frac{n}{4}\right)+n^{3}$. מתקיים כי $a=16,b=4,c=3$ ולכן $\log_{b}a=\log_{4}16=2<3=c$ ולכן $T\left(n\right)=\Theta\left(n^{3}\right)$.

הוכחת משפט האב

נשתמש בשיטת האיטרציה ולאחר מכן בשיטת ההצבה.

ממה שראינו עד כה מתקיים כי

$$\begin{aligned}T\left(n\right)&=aT\left(\tfrac{n}{b}\right)+n^{c}\\T\left(\tfrac{n}{b}\right)&=aT\left(\tfrac{n}{b^{2}}\right)+\left(\tfrac{n}{b}\right)^{c}\\T\left(\tfrac{n}{b^{2}}\right)&=aT\left(\tfrac{n}{b^{3}}\right)+\left(\tfrac{n}{b^{2}}\right)^{c}\\&\vdots\end{aligned}$$

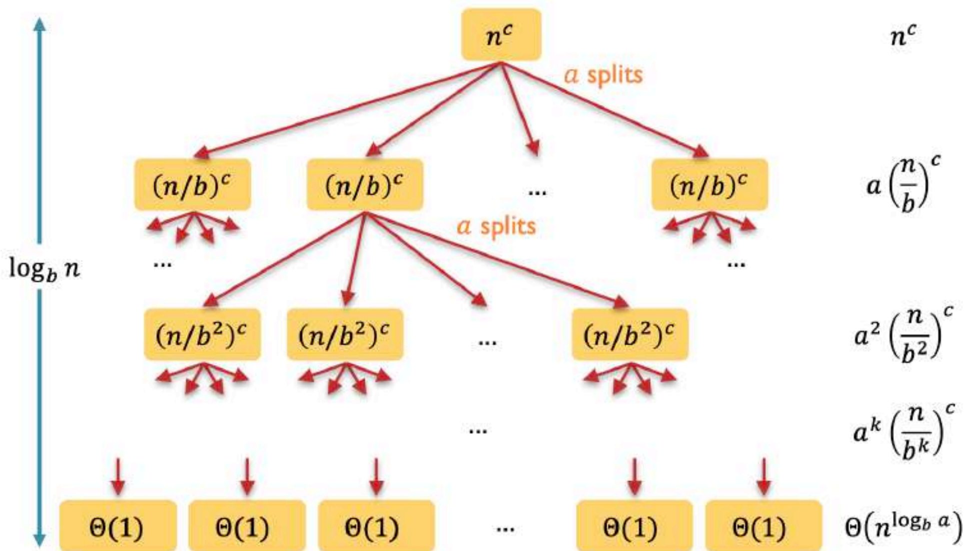
נעבור לשיטת ההצבה:

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{b}\right) + n^c \\
&= a\left(aT\left(\frac{n}{b^2}\right) + \left(\frac{n}{b}\right)^c\right) + n^c \\
&= a\left(a\left(aT\left(\frac{n}{b^3}\right) + \left(\frac{n}{b^2}\right)^c\right) + \left(\frac{n}{b}\right)^c\right) + n^c \\
&\vdots \\
&= a^kT\left(\frac{n}{b^k}\right) + n^c\left(1 + a\left(\frac{1}{b}\right)^c + a^2\left(\frac{1}{b^2}\right)^c + \ldots + a^{k-1}\left(\frac{1}{b^{k-1}}\right)^c\right)
\end{aligned}
$$

ועל ידי הוכחה באינדוקציה נסיק כי $T(n) = a^kT\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i\left(\frac{n}{b^i}\right)^c$.

הבהרה כאשר אנו משתמשים ב-. . . אנו מדלגים על שלב ההוכחה באינדוקציה ולכן צריך להיזהר, בקורס זה לא נוותר על הוכחה זו אלא רק בקורסים מתקדמים. אם כך, במידה ונשתמש בטרמינולוגיה כלשהי של . . . נדע שעלינו להוכיח זאת באינדוקציה.

נרצה לחשב את המחיר הכולל של כל העץ:



איור 21: $T(n) = a^kT\left(\frac{n}{b^k}\right) + \sum_{i=0}^{k-1} a^i\left(\frac{n}{b^i}\right)^c$

ראינו כי יש $k=\log_b n$ רמות בעץ, לכן מחיר כל העץ הנ"ל הוא

$$\begin{aligned}T(n)=&a^{\log_b n}T\left(\frac{n}{b^{\log_b n}}\right)+\sum_{i=0}^{\log_b n-1}a^i\left(\frac{n}{b^i}\right)^c\\=&n^{\log_b a}T\left(\frac{n}{n}\right)+\sum_{i=0}^{\log_b n-1}a^i\left(\frac{n}{b^i}\right)^c\\=&n^{\log_b a}T(1)+\sum_{i=0}^{\log_b n-1}a^i\left(\frac{n}{b^i}\right)^c\\=&\Theta\left(n^{\log_b a}\right)+\sum_{i=0}^{\log_b n-1}a^i\left(\frac{n}{b^i}\right)^c\\=&\Theta\left(n^{\log_b a}\right)+n^c\sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i\end{aligned}$$

נשים לשני דברים. דבר ראשון, $n^{\log_b a}$ לא תלוי ב-c וזה כבר עונה על האינטואיציה שלנו ממקודם. הסכום בתוצאה הסופית מזכיר סכום גאומטרי, מעניין. את נוסחה זו עלינו להוכיח באינדוקציה על n, אך לא נעשה זאת עכשיו.

נסכם בקצרה את הביטויים בנוסחה:

- $\Theta\left(n^{\log_b a}\right)$ מספר העלים.
- $n^c\sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i$ עלות בכל הרמות.

נשים לב כי $\sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i=\frac{\left(\frac{a}{b^c}\right)^{\log_b n}-1}{\frac{a}{b^c}-1}$ מהנוסחה לסכום סדרה גאומטרית. נחלק למקרים עבור $\frac{a}{b^c}$:

המקרה $\frac{a}{b^c}<1$

במקרה זה $a<b^c$ ולכן $\log_b a<\log_b b^c=c$ (כמו בתנאי למשפט האב) ונשים לב כי

$$\frac{\left(\frac{a}{b^c}\right)^{\log_b n}-1}{\frac{a}{b^c}-1}=\frac{1-\left(\frac{a}{b^c}\right)^{\log_b n}}{1-\frac{a}{b^c}}\le\frac{1}{1-\frac{1}{b^c}}=\Theta(1)$$

כלומר במקרה זה $n^c\sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i=\Theta(n^c)$ ומכיוון ש-$\log_b a<c$ נקבל כי

$$T(n)=\Theta\left(n^{\log_b a}\right)+\Theta(n^c)=\Theta(n^c)$$

כרצוי לפי תנאי המשפט.

המקרה $\frac{a}{b^c}=1$

במקרה זה $c = \log_b b^c = \log_b a$. נשים לב כי $\log_b n = \sum_{i=0}^{\log_b n-1} 1 = \sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i$ ולכן $= \sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i n^c$
$\Theta\left(n^c \log_b n\right)$ ומכיוון ש-$n^{\log_b a} \le n^c \log_b n$ נקבל כי

$$T(n)=\Theta\left(n^{\log_b a}\right)+\Theta\left(n^c\log_b n\right)=\Theta\left(n^c\log_b n\right)$$

המקרה $\frac{a}{b^c}>1$

במקרה זה $c = \log_b b^c < \log_b a$. נשים לב כי $\Theta\left(\left(\frac{a}{b^c}\right)^{\log_b n}\right) = \frac{\left(\frac{a}{b^c}\right)^{\log_b n}-1}{\frac{a}{b^c}-1}$ ולכן $\left(\frac{a}{b^c}\right)^{\log_b n}\right) = \Theta\left(n^c \left(\frac{a}{b^c}\right)^{\log_b n}\right) = \sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i n^c$.
נחשב

$$n^c\left(\frac{a}{b^c}\right)^{\log_b n}=\frac{n^c}{b^{c\log_b n}}\cdot a^{\log_b n}=\frac{n^c}{n^c}a^{\log_b n}=a^{\log_b n}=n^{\log_b a}$$

ולכן $\Theta\left(n^{\log_b a}\right) = \sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i n^c$. במקרה זה

$$T(n)=\Theta\left(n^{\log_b a}\right)+\Theta\left(n^{\log_b a}\right)=\Theta\left(n^{\log_b a}\right)$$

כאן למעשה סיימנו את הוכחת המשפט.

משפט האב - סיכום

בהנתן:

- $a,b\ge 1, c\ge 0$ קבועים.
- $T(n)=aT\left(\frac{n}{b}\right)+n^c$.
- $T(1)=\Theta(1)$.

נוכל לקבוע כי:

- $T(n)=\Theta\left(n^c\right)$ אם $\log_b a<c$.

- $T(n)=\Theta\left(n^c\log_b n\right)$ אם $\log_b a=c$.
- $T(n)=\Theta\left(n^{\log_b a}\right)$ אם $\log_b a>c$.

משפט האב הכללי

המקרה הכללי הוא עבור:

- $a,b\geq 1$ קבועים.
- $T(n)=aT\left(\frac{n}{b}\right)+f(n)$.
- $T(1)=\Theta(1)$.

נראה זאת במפורט בתרגול.

דוגמות חשובות

(i) $T(n)=T(n-1)+\mathcal{O}(1)$. במקרה זה אי אפשר להשתמש במשפט האב, אבל ראינו כי $T(n)=\mathcal{O}(n)$.

(ii) $T(n)=T(n-1)+\mathcal{O}(n)$. במקרה זה גם אי אפשר להשתמש במשפט האב אבל ראינו כי $T(n)=\mathcal{O}\left(n^2\right)$.

(iii) $T(n)=2T(n-1)+\mathcal{O}(1)$. אינטואיטיבית $T(n)=\mathcal{O}(2^n)$ (אפשר להוכיח בשיטת ההצבה).

(iv) $T(n)=T\left(\frac{n}{2}\right)+\mathcal{O}(1)$ ממשפט האב $T(n)=\Theta(\log_2 n)$.

(v) $T(n)=2T\left(\frac{n}{2}\right)+\mathcal{O}(1)$, ממשפט האב $T(n)=\Theta(n)$.

(vi) $T(n)=2T\left(\frac{n}{2}\right)+\mathcal{O}\left(\frac{n}{2}\right)$. ממשפט האב $T(n)=\Theta(n\log n)$.

חלק III

הרצאה $\mathbb{III}$ - מיון מהיר ומיון מבוסס השוואה

בהרצאה זו נלמד את הנושאים הבאים:

- אלגוריתם "מיון מהיר" $(Quick\ Sort)$ (בספר הקורס מופיע בפרק 7).
- ננסה לאפיין את האלגוריתם, ונראה כמה הוא יעיל.
  - נוכיח כי כל אלגוריתם מיון הוא בעל סיבוכיות זמן הריצה T שמקיימת $T\left(n\right)=\Omega\left(n\log n\right)$.

5 בעית המיון

בהינתן סדרה של n מספרים נרצה ליצור ממנה פרמוטציה בסדר עולה.

- זו בעיה **פנדמנטלית**.
- יש מגוון דרכים לגשת לבעיה זו ולכתוב לה אלגוריתם.
- אנו יודעים חסם **תחתון** אופטימלי לבעיית המיון.

סוגי מיונים

יש שני סוגי מיון עיקריים:

מיון מבוסס השוואה

- משתמשים באופרטור ההשוואה בין איברי המערך $a_i\leq a_j$, זהו אופרטור הבסיס. את שיטה זו ראינו במיון מיזוג, מיון בועות ומיון הכנסה.
- החסמים התחתונים בשיטה זו הם $T\left(n\right)=\Omega\left(n\log n\right),\ S\left(n\right)=\Omega\left(n\right)$ (נוכיח בהמשך ההרצאה).

מיון שאינו מבוסס השוואה

- לא משתמשים בהשוואות. בשביל מיון כזה צריך להניח הנחות נוספות (כפי שנראה בהמשך).
- החסמים התחתונים בשיטה זו הם $T\left(n\right)=\Omega\left(n\right),\ S\left(n\right)=\Omega\left(n\right)$.
- יש פה טרייד-אוף, אנחנו מוותרים על כלליות ומרוויחים יעילות זמן ריצה.

5.1 מיון מהיר $Quick\ Sort$

האלגוריתם מיון מהיר[1] הוא:

- מבוסס השוואה.
- מהיר מאוד בפועל.
- מבצע מיון במערך הקלט.

סיבוכיות

אלגוריתם זה משתייך למשפחה של אלגוריתם שנקראים $In-Place\ Sorting$ (מיון במקום).

$In-Place\ Sorting$ במקרה זה, האלגוריתם לא משתמש בזכרון נוסף באופן לא קבוע, אלא רק באיברי המערך ועוד כמות קבועה של זכרון. כלומר, האלגוריתם ממיין את האיברים בתוך מערך המקור. על כן, $S(n)=\Theta(1)+\Theta(n)=\Theta(n)$.

זמן הריצה של האלגוריתם הוא:

- המקרה הגרוע: $T(n)=\Theta\left(n^2\right)$.
- המקרה הממוצע: $T(n)=\Theta(n\log n)$.

את שתי תכונות אלה נוכיח היום.

שאלה זמן הריצה במקרה הגרוע ביותר הוא $\Theta\left(n^2\right)$, אם כך מדוע אנו אומרים שבפועל השיטה הזו מהירה מאוד?

תשובה במקרה הממוצע זמן הריצה הוא $\Theta(n\log n)$, ואכן במקרים מסוימים האלגוריתם מאוד לא יעיל, אבל אנו נראה שבמקרה הממוצע, ברוב המקרים, $T(n)=\Theta(n\log n)$ ולכן הוא מהיר.

הפרד ומשול

- **קלט**: מערך לא ממוין $A[i]$ בגודל n, $1\le i\le n$.
- **פלט:** מערך ממוין $A[i]\le A[i+1]$, $1\le\forall i\le n-1$.

[1] הומצא ב-$1961-1959$ על ידי $Tony\ Hoare$, זוכה פרס טיורינג.

הרעיון ההפרדה מאחורי האלגוריתם הוא הדבר הבא:

הפרד ניקח את המערך ונחלק אותו לשני חלקים באמצעות איבר שנכנו $pivot$[2]. מכך נקבל חלק ימני וחלק שמאלי. את החלק הימני והשמאלי נסמן ב-$r = right$, $l = left$ בהתאמה. נרצה כי $left \leq pivot < right$. כלומר על ידי חלוקת המערך לשני תתי מערכים נקבל שתי תתי בעיות חדשות.

לדוגמה, נביט במערך $[2, 1, 3, 4, 7, 6, 5, 8]$ עבור $pivot = 4$.

הרעיון מאחורי המשילות הוא הבא:

משול נרצה שה-$pivot$ כבר יהיה במקום הנכון ונמיין באופן רקורסיבי את צד ימין וצד שמאל. המיון נעשה **במקום**, בתוך המערך, לכן לא צריך לאחד מערכים.

דוגמה. נביט במערך $A = [\underset{\underset{l}{\uparrow}}{2}, 1, 3, \underset{\underset{m}{\uparrow}}{4}, 7, 6, 5, \underset{\underset{r}{\uparrow}}{8}]$. כאשר m מצביע על מיקום ה-$pivot$ ו-r, l מצביעים על סוף תת המערך הימני ותחילת תת המערך השמאלי בהתאמה.

נרשום אם כך פסידו-קוד לאלגוריתם באופן הבא:

Algorithm 3 $Quick\ Sort\,(A, l, r)$ - מיון מהיר

1: **if** $l < r$ **then do**
2: $\quad m \leftarrow Partition\,(A, l, r)$
3: $\quad Quick\ Sort\,(A, l, m-1)$
4: $\quad Quick\ Sort\,(A, m+1, r)$

נשים לב כי:

- אם $r = l$, הריצה תעצור, אנו צריכים למיין מערך אחד שזאת בעיה שפתרונה טריוויאלי ואינו דורש טיפול, וזהו למעשה מקרה הבסיס.
- $A\,[m]$ תמיד במקום שהוא אמור להיות בו בפרמוטציה הממוינת.
- m הוא ה-$pivot$.

באלגוריתם השתמשנו בשיטת עזר שנקראת $Partition$, נרצה לחקור אותה.

שיטת עזר $Partition\,(A, l, r)$

נשים לב כי:

[2] באלגברה ליניארית נהוג לכנותו "איבר מוביל".

- אנו יודעים כי השיטה תעבור על המערך פעם אחת אינטואיטיבית, כי היא צריכה רק להעביר איברים מצד אחד לצד אחר ביחס ל-$pivot$. מעבר כזה בעל סיבוכיות זמן ריצה $\Theta\left(n\right)$
- שיטה זו בוחרת ערך $pivot$ מהמערך. הבחירה שרירותית, למרות שהבחירה שנעשה תשפיע על הריצה.
- בסוף ריצת השיטה ה-$pivot$ ממוקם במיקום הסופי והנכון, המתאים לו בפרמוטציה הממוינת.
  - כל האיברים הקטנים מ-$pivot$ נמצאים משמאלו במערך.
  - כל האיברים הגדולים מ-$pivot$ נמצאים מימינו במערך.
  - המערך לא בהכרח ממוין בסוף ריצה של שיטה זו.

אינטואיציה

נרצה להבין אינטואיטיבית איך לממש אותה. נחלק את המערך לארבעה חלקים:

1. שמאל $\left(left\right)$ - האיברים שמשמאל ל-$pivot$ וקטנים ממנו.
2. ימין $\left(right\right)$ - האיברים שמימין ל-$pivot$ וגדולים ממנו.
3. לא ידוע - האיברים שעוד לא נקבע מיקומים (ימין או שמאל).
4. $pivot$.

בהתחלה, מתקיים כי:

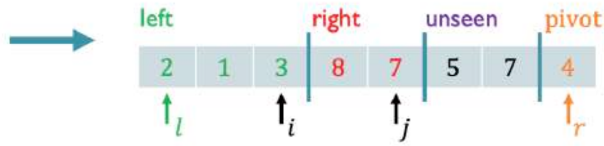
- $pivot$ נבחר להיות האיברי הימני ביותר במערך - נבחר זאת לשם פשטות.
- $left, right$ ריקים.
- לא ידוע הוא כל שאר המערך.

בסוף ריצת השיטה:

- לא ידוע הוא אזור ריק.
- כל איבר במערך מלבד $pivot$ נמצא ב-$left$ או ב-$right$.

מסקנה. בכל איטרציה בשיטה נטפל באיבר נוסף מהאזור הלא ידוע. לאחר כל האיטרציות נסיים למקם את כולם.

לצורך המחשה, נביט במערך הבא:



איור 22 : המחשה לחלוקה האזורים והגדרת משתני עזר

כדי להיות מסוגלים להבחין מהו האזור השמאלי ומהו האזור הימני, נגדיר שני משתני עזר i, j אשר יצביעו על סוף האזור הימני והשמאלי בהתאמה. למעשה, זהו כל הזכרון הנוסף הדרוש לאלגוריתם והוא קבוע.

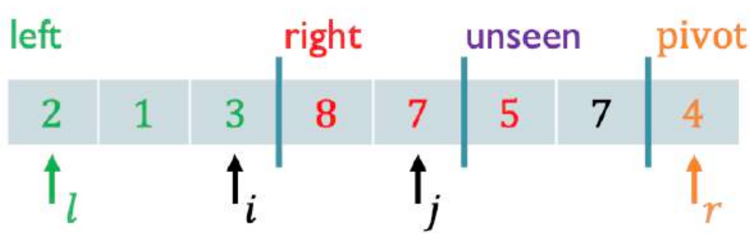
בנוסף, מתקיים כי $A\left[r\right]$ הוא האיבר הימני ביותר ומייצג את $pivot$ ומטרתנו היא לקבל כי :

- כל האיברים ב-$left \leq pivot$.
- כל האיברים ב-$right > pivot$.

על מנת לעבור על המערך, נגדיל בכל פעם את j ולמעשה $A\left[j+1\right]$ יושווה ל-$pivot$ במטרה לקבוע את מיקומו. על מנת להעבירו מאזור אחד לאזור אחר, נגדיל את i, j בהתאם (מיד נבין איך).

מקרה קל

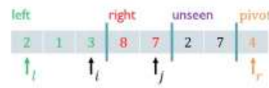
במקרה הפשוט, מתקיים כי $A\left[j+1\right] > pivot$ ולכן כל שעלינו לעשות הוא לכלול אותו באזור ה-$right$ ולכן כל שעלינו לעשות הוא להגדיל את j. ניתן להביט באיור הבא להמחשה.



איור 23 : המקרה הקל של החלפה, $A\left[j+1\right] = 5 > 4 = A\left[r\right]$

מקרה אחר

במקרה זה $A\left[j+1\right] \leq A\left[r\right]$ ולכן מה שעלינו לעשות הוא לכלול את $A\left[j+1\right]$ באזור ה-$left$, כדי לעשות זאת, נגדיל את i, j ונחליף בין $A\left[j\right]$ לבין $A\left[i\right]$ כלומר נחליף את האיבר האחרון באזור $right$ עם האיבר הימני ביותר באזור ה-$left$. האיור הבא מצורף להמחשה.



איור 24 : מקרה אחר של החלפה, $A\left[j+1\right]=2<4=A\left[r\right]$

משני דוגמות אלה, נסיק כי אך ורק במקרה $A\left[j+1\right]\leq A\left[r\right]$ נרצה לעשות החלפה, שכן במקרה הראשון כל שהיה עלינו לעשות הוא $j\leftarrow j+1$.

אם כך, נכתוב פסדואו קוד לשיטה זו :

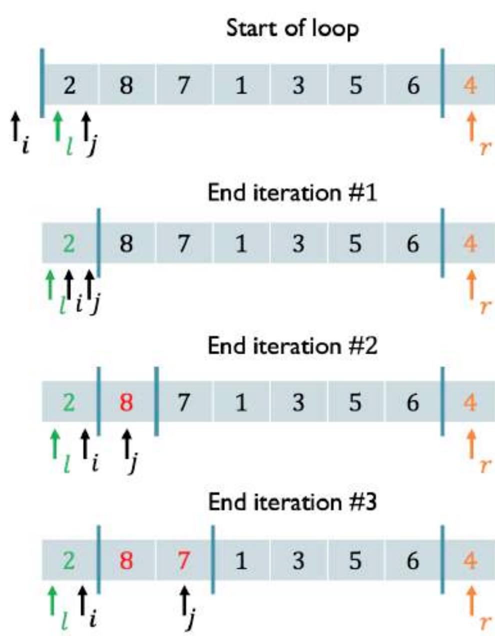
Algorithm 4 $Partition\left(A,l,r\right)$

$i\leftarrow l-1$ // **End of the left region**
for $j\leftarrow l$ **to** $r-1$ **do** // **All Elements are in** $left, right$ **or pivot.**
  if $\left(A\left[j\right]\leq A\left[r\right]\right)$ **then**
    $i\leftarrow i+1$
    $Exchange\left(A\left[i\right],A\left[j\right]\right)$
$Exchange\left(A\left[i+1\right],A\left[r\right]\right)$ // **Place pivot in final spot.**
return $i+1$ // **Return pivot's index**

נרצה להבין את האלגוריתם באמצעות דוגמת הרצה.

נביט במערך $[2,8,7,1,3,5,6,4]$. תחילה, i מחוץ למערך ו-l,j מצביעים על תחילתו. אכן מקרה זה מעט מוזר.

נביט בתרשים הבא :



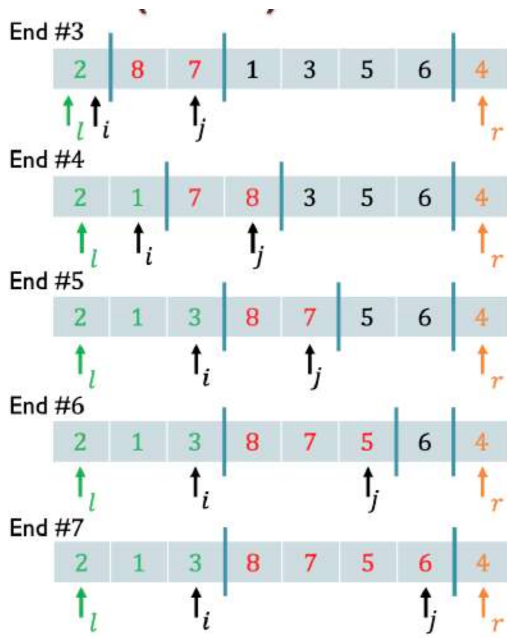
איור 25 : תרשים לריצה של $Partition$

באיטרציה הראשונה מצאנו כי $2 < pivot = 4$ ולכן החלפנו אותו עם עצמו והוא נשאר במקומו.

באיטרציה השנייה, מצאנו כי $8 > pivot$ ולכן הגדלנו את j כדי להכיל אותו באזור ה-$right$.

באיטרציה השלישית עשינו פעולה דומה עם 7 כמו באיטרציה השנייה.

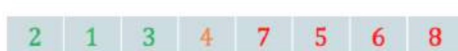
נביט בהמשך התהליך:



איור 26: המשך דוגמת הרצה של $Partition$

באיטרציה הרביעית היינו צריכים להחליף בין $3, 7$ שכן במקרה זה מתבצעת ההחלפה בלולאה. השלבים הבאים דומים לשלבים הקודמים.

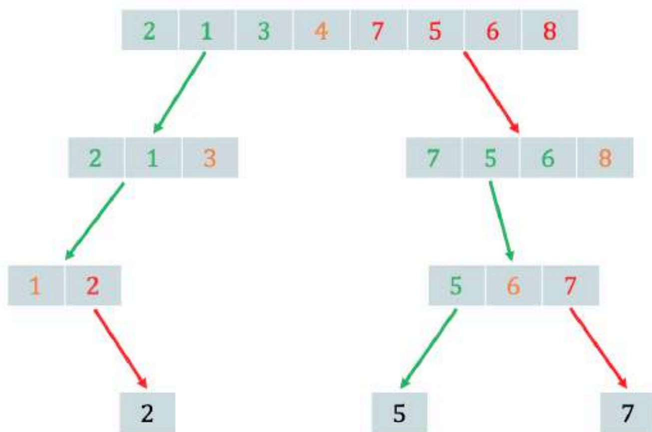
בסופו של דבר, בסיום ריצת הלולאה, מתבצעת הפעולה $Exchange\left(A\left[i+1\right], A\left[r\right]\right)$ ולכן מתקבל המערך:



איור 27: המערך בסיום ריצת $Partition$

נשים לב כי $pivot$ במיקומו הנכון, שכן כל האיברים מימינו גדולים ממנו וכל האיברים משמאלו קטנים ממנו, נותר לקרוא רקורסיבית לאלגוריתם $Quick\ Sort$ על $left, right$.

נמחיש את ריצת האלגוריתם $Quick\ Sort$ לפי העץ הבא:



איור 28: דוגמה לעץ רקורסיה עבור האלגוריתם $Quick\ Sort$

כלומר החלוקה ממשיכה להתבצע עד שמגיעים למערכים בגודל אחד שהם מקרה הבסיס.

סיבוכיות

נשים לב כי הסיבוכיות תלויה בבחירת ה-$pivot$ שכן בחירה זו יוצרת לנו את $left, right$. האם בכל בחירה $left, right$ מאוזנים?

- חלוקה לא מאוזנת: כל האיברים ב-$left$ או שכל האיברים ב-$right$. במקרה זה נקבל שתי תתי בעיות בגודל $0, n-1$.
- חלוקה מושלמת: $left, right$ באותו הגודל. במקרה זה נקבל שתי תתי בעיות בגודל $\frac{n}{2}$.
- חלוקה כללית: איפשהו באמצע בין שני המקרים הקודמים, כלומר קיים $0 \leq q \leq n-1$ עבורו נקבל שתי תתי בעיות בגודל $q, n-q-1$.

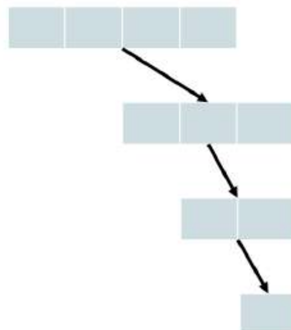
חלוקה לא מאוזנת

במקרה זה נקבל כי נוסחת הנסיגה היא $T(n) = \underbrace{T(0)}_{0} + T(n-1) + \Theta(n)$ ($\Theta(n)$ הוא הסיבוכיות של $Partition$).

פתרון נוסחא זו הוא באינדוקציה, ומשיטת האיטרציה נקבל כי

$$T(n) = T(n-1) + \Theta(n) = \sum_{k=1}^{n} \Theta(k) = \Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

ורקורסיבית זה יראה כך:

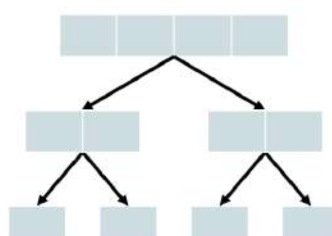


איור 29: המחשה לחלוקה לא מאוזנת

חלוקה מושלמת

במקרה זה תמיד נבחר את האיבר האמצעי במערך מבחינת גודל (לא מיקום). נקבל כי נוסחת הנסיגה היא $T\left(n\right)=2T\left(\frac{n}{2}\right)+$ $\Theta\left(n\right)$ שפתרונה ממשפט האב הוא $T\left(n\right)=\Theta\left(n\log n\right)$ (כמו מיון מיזוג).

רקורסיבית זה יראה כך:



איור 30: המחשה לחלוקה מושלמת

המקרה הכללי

עולה מהמקרים הקודמים כי $\Theta\left(n\log n\right)$ הוא המקרה הטוב ביותר ו-$\Theta\left(n^2\right)$ הוא המקרה הגרוע. נותר רק להוכיח זאת באופן פורמלי. נוכיח עבור המקרה הגרוע, שכן העובדה ש-$\Theta\left(n\log n\right)$ היא המקרה הטוב ביותר תנבע מעובדה שנראה בהמשך ומכך שהאלגוריתם מסוגל לבצע את המשימה ב-$\Theta\left(n\log n\right)$. האינטואיציה מאחורי המקרה הגרוע ביותר היא בחירה של $n-1,0$, אבל זאת לא הוכחה.

הוכחה: עבור חלוקה ספציפית עם $q, n-q-1$ נקבל כי

$$T(n) = T(q) + T(n-q-1) + \Theta(n)$$

ואילו במקרה הגרוע ביותר, נרצה כי $T(q) + T(n-q-1)$ יהיה מקסימלי, אז נסמן

$$T(n) = \max_{0\le q\le n-1}\{T(q) + T(n-q-1)\} + \Theta(n)$$

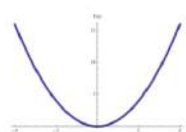
באמצעות שיטת ההצבה נוכיח כי $T(n) = \mathcal{O}(n^2)$. כמו שראינו בעבר, נבחר את c באופן מושכל לפי הצעד.

בסיס: $n=1$ במקרה זה $T(1) = 1 \le c\cdot 1^2$ לכל $c \ge 1$.

צעד: נניח שהטענה נכונה לכל $q \le n-1$ נוכיח שהיא נכונה עבור n.

$$\begin{aligned}T(n) &= \max_{0\le q\le n-1}\{T(q) + T(n-q-1)\} + \Theta(n)\\ &\overset{\text{ה״א}}{\le} \max_{0\le q\le n-1}\left\{cq^2 + c(n-q-1)^2\right\} + \Theta(n)\\ &= c\max_{0\le q\le n-1}\left\{q^2 + (n-q-1)^2\right\} + \Theta(n)\end{aligned}$$

נותר לקבוע מהו $\max_{0\le q\le n-1}\left\{q^2 + (n-q-1)^2\right\}$. נסתכל על $f(q) = q^2 + (n-q-1)^2$ כפונקציה פרבולית במשתנה ממשי q. נשים לב כי זו פרבולה מחייכת וכי ציר הסימטריה הוא ב-$q = \frac{n}{2}$. לכן **בקצוות** של הקטע $[0, n-1]$ נקבל כי ערכה **מקסימלי**. כלומר עבור $q \in \{n-1, 0\}$ נקבל את המקרה הגרוע ביותר ובו $f(q) = (n-1)^2$.



איור 31: הפרבולה $q^2 + (n-q-1)^2$ כאשר הציר האנכי הוא ב-$x = \frac{n}{2}$

אם כך נקבל כי (עבור c, n גדולים דיו)

$$\begin{aligned}T(n) &\le c \max_{0\le q\le n-1}\left\{q^2+(n-q-1)^2\right\}+\Theta(n)\\ &=c(n-1)^2+\Theta(n)\\ &=cn^2-c(2n-1)+\Theta(n)\\ &\le cn^2\end{aligned}$$

כאשר נבחר c מספיק גדול עבורו $-c(2n-1)+\Theta(n)<0$. על כן $T(n)=\mathcal{O}\left(n^2\right)$. ■

שאלה האם $T(n)=\Omega\left(n^2\right)$?

תשובה כשדיברנו על חלוקה לא מאוזנת עבור $Partition$ ראינו כי מקבלים סך הכל $T(n)=\Theta\left(n^2\right)$ ומכיוון שראינו בהוכחה כי $0, n-1$ הם המקרה הגרוע ביותר, נסיק כי $T(n)=\Omega\left(n^2\right)$ במקרה הגרוע ביותר.

המקרה הממוצע

מקרה זה הוא איפשהו באמצע בין שני המקרים.

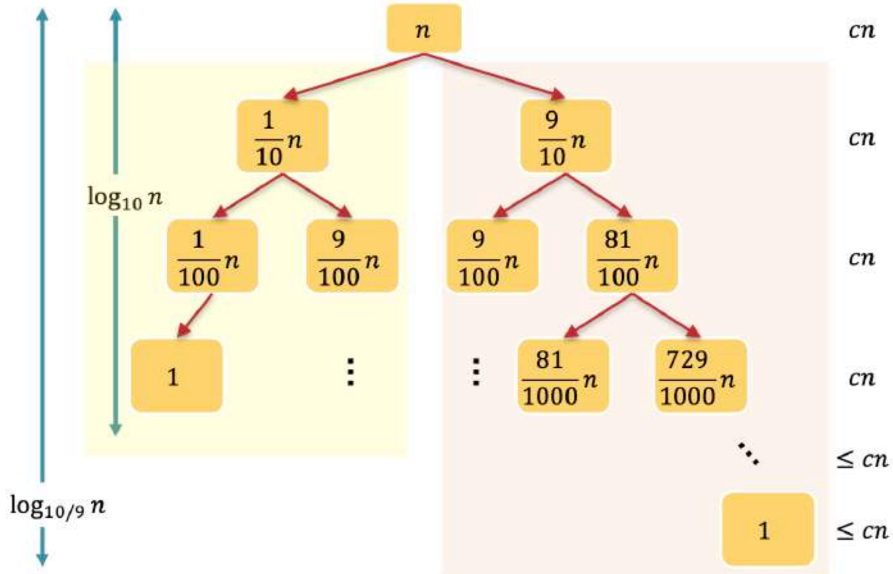
- לא יותר טוב מ-$\Theta(n\log n)$.
- לא גרוע יותר מ-$\Theta\left(n^2\right)$.

מיד נראה כי במקרה זה $T(n)=\Theta(n\log n)$.

כדי לצבור אינטואיציה, נביט בדוגמה הבאה.

נניח כי כל חלוקה היא ביחס של $1/9$ כלומר תחילה $\frac{9}{10}, \frac{1}{10}$ וכן הלאה...

במקרה זה נקבל עץ רקורסיה הנראה כך:



איור 32: המחשה למקרה הממוצע בחלוקה ביחס $1/9$

נשים לב כי הענף הקצר ביותר בעץ הוא בעומק $\log_{10} n$ ואילו הענף הארוך ביותר הוא בעומק $\log_{\frac{10}{9}} n$. בכל רמה בעץ אנו עושים פעולות בסדר גודל של $\Theta(n) = cn$.

לכן, במקרה הטוב ביותר, הגענו עד לרמה בגובה $\log_{10} n$ (הענף הקצר) ונקבל ש-$T(n) = \Omega(cn \log_{10} n) = \Omega(n \log n)$.
במקרה הגרוע ביותר אנו בענף הארוך ביותר ברמה בעומק $\log_{\frac{10}{9}} n$ונקבל כי $T(n) = \mathcal{O}\left(n \log_{\frac{10}{9}} n\right) = \mathcal{O}(n \log n)$. לכן $T(n) = \Theta(n \log n)$. זהו רק הסבר אינטואיטיבי ולא הוכחה, על מנת להוכיח שזה אכן המצב, נצטרך כלים הסתברותיים אותם נרכוש רק בהמשך הקורס.

נוסחת הנסיגה לזמן הריצה במקרה זה היא $T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$. שימו לב שבסיס ה-log לא משנה לזמן הריצה.

עתה נסכם ונאמר כי ראינו רק עבור מקרה מאוד ספציפי כי $T_{\text{Average}}(n) = \Theta(n \log n)$, נוכיח את הטענה במלואה בתרגול.

עתה נשים לב כי יתכן שמישהו עלה על אופן בחירת ה-*pivot* שלנו ויתן לנו בכוונה מערכים ממוינים או בסדר הפוך. על כן, נוכל לבחור את ה-*pivot* באופן רנדומלי ונבטיח כי $T(n) = \Theta(n \log n)$. כל דרך אחרת תוביל לפרצות.

6 מיון מבוסס השוואה - חסם תחתון

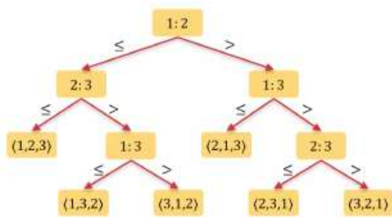
נרצה למצוא חסם תחתון לכל אלגוריתם מיון מבוסס השוואה.

הערה. נשנה את מחשבתנו למחשבה כללית ללא התמקדות באלגוריתם ספציפי.

- נשתמש בעץ בחירה על מנת למנות את כמות הפעולות בכל אלגוריתם כזה.
- נשתמש במודל זה על מנת לאפיין אלגוריתמי מיון מבוססי השוואה ונגלה כי לכל אלגוריתם כזה מתאים עץ בחירה.

6.1 עץ בחירה

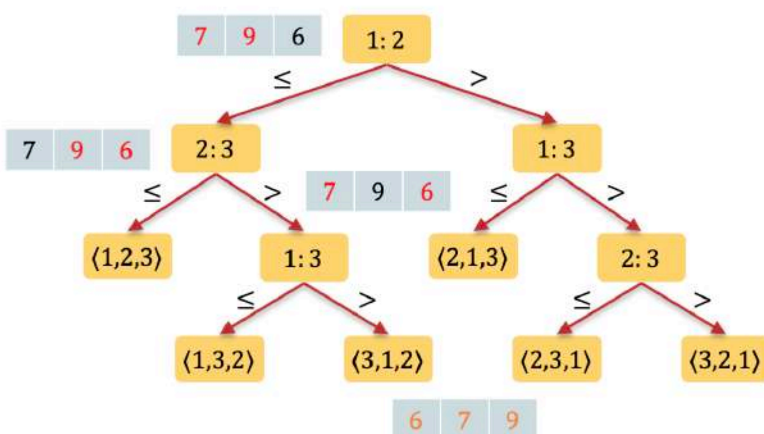
נביט במערך $[1,2,3]$ ונבנה עץ החלטה בניסיון למיין אותו. כל זוג בקודקודים בעץ מייצ זוג אינדקסים.



איור 33 : דוגמה לעץ בחירה

כל עלה הוא פרמוטציה של אינקסים שמייצגת את הגרסא של הממוינת של המערך לפי הנתיב לעלה.

באופו דומה מתבצע תהליך דומה עבור המערך $[7,9,6]$:



איור 34 : דוגמה נוספת לעץ בחירה

נבחין כי עצי בחירה הם עצים בינאריים. יתר על כן, הם יכולים לייצג השוואות בין איברים באלגוריתם ספציפי.

בנוסף, הקודקדים בעצים אלה הם כדלקמן :

- קודקוד פנימי: שני אינדקסי השוואה i, j.
- עלה: פרמוטציה $\langle\pi[1], ..., \pi[n]\rangle$. כאן π מסמל פרמוטציה.
- ענפים: תוצאות של השוואות $a_i \leq a_j$ משמאל או $a_i > a_j$ ומימין.

יתר על כן, בהנחה שהעץ מייצג אלגוריתם, כל ביצוע של האלגוריתם הוא נתיב לאורך העץ וכל קודקוד פנימי הוא שאלה של האלגוריתם עבור שני קלטים.

נוסף על כך, העלים הם התוצאה של האלגוריתם ובהם π, הפרמוטציה.

חישוב באמצעות עצי בחירה

נחשב את הנתיב המתאים לקלט לפי השלבים הבאים:

- נתחיל בשורש.
- נרד כל פעם רמה אחת מטה לפי תוצאת ההשוואה.
- נסיים את התהליך כאשר הגענו לעלה.

טענה. כל אלגוריתם מיון נכון חייב להכיל נתיבים לכל אחת מהפרמוטציות.

נשים לב כי יש $n!$ פרמוטציות כאלה עבור קלט באורך n.

משפט. אלגוריתם מיון מבוסס השוואה יכול להיות מתואר על ידי עץ החלטה.

הוכחה: לא כרגע. ■

מסקנה. לעץ ההחלטה של האלגוריתם יש לפחות $n!$ עלים.

שאלה מדוע רק לפחות ולא בדיוק?

תשובה יתכן והאלגוריתם לא יעיל במיוחד וחוזר על עצמו. למשל, ברגע שמצא פרמוטציה הוא מוצא פרמוטציה נוספת שיש לה כבר נתיב אחר.

שאלה מהו העומק המינימלי בעץ עם $n!$ עלים?

נשים לב כי

- במקרה הגרוע ביותר של ההשוואות, מקבלים את האורך של הנתיב המקסימלי לעלה בעץ.

- חסם תחתון למקרה הגרוע שצוין לעיל יתן חסם תחתון למספר צעדי האלגוריתם הדרושים.

מסקנה. חסם תחתון על האורך של הנתיב המקסימלי לעלה בעץ חוסם מלמטה את הסיבוכיות הגרועה ביותר של אלגוריתמי מיון מבוססי השוואה.

נחשב את החסם על עומק זה. נסמן עומק מינימלי זה ב-d, אזי מכיוון שזה עץ בינארי יש בו לכל היותר 2^d עלים (וודאו שאתם מבינים למה) וממה שראינו קודם, יש בו לפחות $n!$ עלים, כלומר $n! \leq 2^d$ ולכן $\log n! \leq d$ כלומר אורך הנתיב המינימלי הוא לכל הפחות $\log n!$.

טענה. $\log n! = \Theta(n \log n)$.

הוכחה: ראינו כי $n! \leq n^n$ ולכן $\log n! \leq \log n^n = n \log n$ על כן $\log n! = \mathcal{O}(n \log n)$.

עתה נוכיח כי $n! \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$.

מתקיים כי

$$n! = \prod_{i=1}^{n} i = \left(\prod_{i=1}^{\left[\frac{n}{2}\right]-1} i\right) \cdot \left(\prod_{i=\left[\frac{n}{2}\right]}^{n} i\right) \overset{\text{הוכחה באינדוקציה}}{\geq} \prod_{i=\left[\frac{n}{2}\right]}^{n} i$$
$$\geq \prod_{i=\left[\frac{n}{2}\right]}^{n} \left[\frac{n}{2}\right] \geq \prod_{i=\left[\frac{n}{2}\right]}^{n} \frac{n}{2} \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

ולכן $\mathcal{O}(n!) = \left(\frac{n}{2}\right)^{\frac{n}{2}}$ ומכיוון ש-$\left(\frac{n}{2}\right)^{\frac{n}{2}} \to \infty$ נקבל כי מטענה שראינו בתרגיל הבית כי $\log\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \mathcal{O}(\log n!)$ אם"ם $\log n! = \Omega\left(\log\left(\frac{n}{2}\right)^{\frac{n}{2}}\right) = \Omega\left(\frac{n}{2}\log\frac{n}{2}\right)$ ולכן $\log n! = \Omega(n \log n)$. ■ נסכם את מה שראינו עד כה באלגוריתמי מיון מבוססי השוואות:

Algorithm	Space	Worst Case	Best Case	Average Case
Bubble Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Insertion Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n)$	$\Theta(n^2)$
Merge Sort	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$
Quick Sort	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n \log n)$	$\Theta(n \log n)$

איור 35 : סיבוכיות זמן ריצה וזכרון של אלגוריתמי מיון מבוססי השוואות

ונסכם כי החסם התחתון על אלגוריתמי מיון אלה הוא $S\left(n\right)=\Omega\left(n\right)$, $T\left(n\right)=\Omega\left(n\log n\right)$.

הערה. הסיבה ש-$S\left(n\right)=\Omega\left(n\right)$ מכיוון שאנו שומרים מערך בגודל n לכן צורכים זכרון בסדר גודל של n לפחות. מכך שקיים אלגוריתם מיון שצורך בדיוק $\Theta\left(n\right)$ זכרון נסיק כי $\Omega\left(n\right)$ חסם תחתון הדוק.

חלק IV

הרצאה $\mathbb{IV}$ - טבלות גישה ישירה, טבלות גיבוב, שרשור, גישה פתוחה, פונקציות גיבוב

מוטיבציה

נרצה מבנה נתונים שממש פעולות על טבלה. המטרות שלנו הן:

- **גישה** לאיברי הטבלה: זמן ממוצע $\mathcal{O}\left(1\right)$.
- המפתחות **לא ממוינים**.

דוגמה. לימוד בחדרים באוניברסיטה - אין משמעות למיקום החדרים אלא רק לגישה אליהם.

דוגמה. לקוחות בנק - מספר זהות, מה שחשוב הוא המספר ולא מיונו (הלקוחות לא ממוינים במערך).

הפעולות שנרצה לממש הן:

- $search$ - חיפוש איבר קיים: $\mathcal{O}\left(1\right)$ בממוצע.
- $insert$ - הכנסת איבר חדש: $\mathcal{O}\left(1\right)$ בממוצע.
- $delete$ - מחיקת איבר קיים: $\mathcal{O}\left(1\right)$ בממוצע.

לשם השוואה, $search$ בעץ הוא $\mathcal{O}\left(\log n\right)$, הרבה יותר ממה שאנחנו רוצים. למעשה הפתרון לבעיה זו היא טבלת גיבוב.
נביט עתה במספר פתרונות לא אידיאליים לבעיה שלנו, ונראה כיצד נוכל לפתורה.

שימוש במערכים

נרצה לנסות לקבל את מטרותינו באמצעות מערכים. ידוע לנו כי גישה לאיבר במערך מתבצעת בזמן ריצה קבוע $\mathcal{O}\left(1\right)$.

בעיות

- צריך מערכים מאוד גדולים. למשל תעודות זהות - כל תעודה בעלת 9 ספרות וללא אפסים, לכן כדי לקבל חיפוש מהיר נצטרך מערך בגודל 10^9 (מיליארד) תאים. נסו להקצות כזו כמות זאת במחשב שלכם (הוא לא יאהב את זה...).
- גם אם היינו יכולים להקצות כזו כמות של זכרון, איננו צריכים אותה בפועל, בסופו של דבר לא יהיו לנו באמת מילארד אובייקטים, לכן רוב התאים יהיו מיותרים. על כן פתרון זה לא אידאלי.

השראה

באותה דוגמה של תעודות הזהות, אם נצפה רק ל-1000 תאים, נוכל לנסות למקם את האובייקטים שלנו במקומות מתאימים ובכך להשיג את הדרוש.

אם כך, נרצה למצוא דרך לתת לכל אובייקט אינדקס במערך. נעשה זאת באמצעות פונקצייה $h\left(k\right)$ שתקבל כקלט תעודת זהות k ותחזיר את האינדקס במערך שמתאים לו $h\left(k\right)$.

למעשה, מערכים קלאסיים משתמשים בפונקציית הזהות כפונקציית מיון $h\left(k\right)=k$. אך האם יש דרך חכמה יותר?

למשל, ניקח את שני המספרים האחרונים בכל תעודת זהות במערך. כך נקבל לדוגמה, $h\left(283475218\right)=18$. היתרון בשיטה זו הוא ש-100 תאים יספיקו. החסרון הוא שיתכנו התנגשויות, מי אמר שרק תעודת זהות אחת מסתיימת ב-18, למשל $h\left(182734118\right)=18$ גם כן. כלומר $\exists k_1,k_2$ כך ש-$k_2\neq k_1$ אבל גם $h\left(k_1\right)=h\left(k_2\right)$, קרי h איננה חח"ע.

7 טבלות גיבוב

רקע היסטורי

הומצאו בסביבות 1953 ב-IBM. עדיין משתמשים בהם המון - $dictionaries$ ב-$Python$ למשל.

מאפיינים

טבלות גיבוב מאופיינות באופן כללי כך:

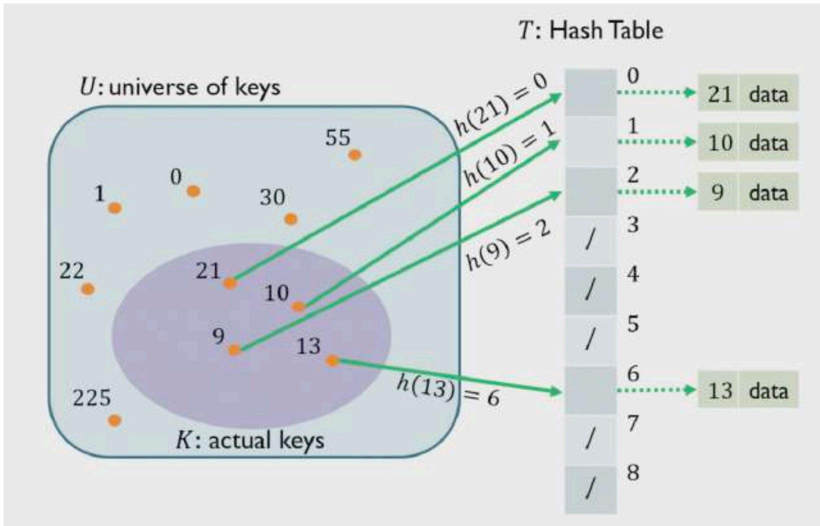
- מבנה נתונים שהוא הכללה של מערכים $A\left[0,\ldots,m-1\right]$.
- במקום להשתמש ב-k כאינדקס נשתמש בפונקציית גיבוב $h\left(k\right)$. כלומר $A\left[k\right]\rightarrow A\left[h\left(k\right)\right]$.
- גודל הטבלה יהיה פרופורציוני למספר האיברים בטבלה ולא הטווח של הערכים שלהם.
- $h\left(k\right)$ היא לא חח"ע.
  - התנגשויות - יתכנו $k_2\neq k_1$ כך ש-$h\left(k_1\right)=h\left(k_2\right)$.
  - נרצה למצוא דרך להתמודד עם התנגשויות.

נביט בדוגמה הבאה שבאיור למיפוי האובייקטים לטבלת גיבוב.

נתונה לנו קבוצת כל המפתחות האפשריים למיפוי בטבלה שמסומנת ב-U. בתוכה הקבוצה K שמכילה את המפתחות שנמפה בפועל, שכן לצורך ההמחשה לא כל האנשים בעולם רשומים בבנק, אלא רק חלק. על פי המידע שיש ב-K נמפה את האיברים

לטבלה T.

סימן ה-$/$ בתוך איברי הטבלה מסמן תא ריק ונקרא NIL.



איור 36 : המחשה למיפוי מפתחות לטבלת גיבוב

דוגמות

מלבד בנק ותעודות זהות אפשר להסתכל על המקרים הבאים (ועוד רבים) :

- חנייה למכוניות לפי לוחית רישוי. מקרה זה מאופיין בכך שאין סדר למכוניות שמגיעות ולא ניתן לצפות את לוחיות הרישוי שיגיעו. בנוסף, נצטרך להעריך את המספר שיגיע, למשל 1000, ועל פיו לבנות את h. עבור לוחית רישוי k ניתן להגדיר $h\left(k\right)=k \mod 1000$ ובכך להשיג מספר קטן מ-1000 (שזה מספר המכוניות שאנו מצפים לו) וגדול מאפס.
- שם משתמש - יש המון אפשרויות לשמות משתמשים, אך הגעתם דינאמית ולא ממוינת באופן אלפאבתי. במקרה זה ננסה למצוא פתרון דומה לבעיה הקודמת.

פורמליזציה

סימונים והנחות :

- U הוא יקום כל המפתחות, גודלו יסומנו $|U|$.
  - לשם פשטות נניח כי $U=\{0,1,\ldots,|U|-1\}$.

- K היא קבוצת כל המפתחות שבהם נשתמש בפועל. כלומר $K \subseteq U$ ונסמן $|K| = n$.
- T היא טבלת הגיבוב בגודל $|T| = m$ כאשר $m \leq |U|$.
- פונקציית הגיבוב $h\left(k\right)$ היא העתקה $h: U \rightarrow \{0, \ldots, m-1\}$ הממפה ערכי איברים מ-U לאינדקס ב-T.
- $h\left(k\right)$ מחושבת בזמן קבוע - $\mathcal{O}\left(|k|\right) = \mathcal{O}\left(1\right)$. למשל אם יש לקלט 9 ספרות - החישוב הוא קבוע.
- עבור $0 \leq i \leq m-1$ לאיברי $T\left[i\right]$ יש גישה בסיבוכיות זמן ריצה קבועה $\mathcal{O}\left(1\right)$.
- למקומות ריקים בטבלה נסמן $T\left[i\right] = NIL\ (\text{or}\ /)$. האיבר NIL הוא איבר מיוחד שאינו חלק מ-U.

בעיות במפתחות

נשאלת השאלה, מהן פונקציות $hash$ טובות ורעות? מהם הקריטריונים להערכת פונק' $hash$?

דוגמה. עבור תעודות זהות, נבחר את הספרה הראשונה כאינדקס. זהו פתרון לא טוב, כי יהיו לנו המון התנגשויות.

השיקולים בבדיקת פונקציית כאלה הם:

- כמה התנגשויות?
- כמה טוב האיברים מפוזרים במערך?

דוגמה. עבור תעודות זהות, נבחר ספרה זוגית מהמספר. הבעיה בפתרון זה מלבד התנגשויות הוא שאנו לא משתמשים מחצי מהמקומות בטבלה - המקומות בעלי אינדקס אי-זוגי.

נשאל אם כך את השאלות הבאות:

- מה נעשה עם התנגשויות?
- איך נבטיח זמן ריצה קבוע $\mathcal{O}\left(1\right)$ בממוצע?

בקרוב נראה שבממוצע משימה זו אפשרית (הכוונה בממוצע שאין איזשהו יריב שמנסה לתת לנו אך ורק מקרי קצה שיצרו התנגשויות במכוון).

שאלה מדוע אי אפשר ליצור פונקציה h חח"ע

תשובה אם יש לנו 10^9 מפתחות אבל רק 10^3 תאים בטבלה, מעקרון שובך היונים, נקבל התנגשויות. כלומר, יש יותר מפתחות מתאים בטבלה.

טיפול בבעית המפתחות

כדי להתמודד עם בעיית המפתחות כמו התנגשויות, נניח את ההנחה הבאה:

הנחת הגיבוב האחיד והפשוט - גא"פ ההסתברות שמפתח רנדומלי חדש ימופה לאיזשהו תא m בלתי תלויה במיפוי האיברים הקודמיםץ אנו מניחים שהסיכוי שאיזשהו אובייקט יכנס לתא m הוא אותו סיכוי של אותו אובייקט להיות ממופה לכל תא אחר.

הנחה זו היא אידאליסטית, שתעזור לנו לנתח את פונקציות הגיבוב.

הטיפול בבעיה ללא הנחה זו דורשת שנדע את ההתפלגות ההסתברותית של מיפוי המפתחות, שלרוב לא ידועה לנו.

7.1 שיטות גיבוב

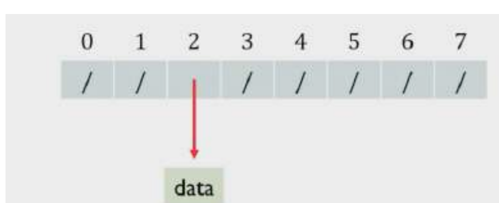
נציג עתה שלוש שיטות גיבוב שונות.

מיון ישיר

שיטה זו היא למעשה מימוש של רעיון השימוש במערכים. פונק' הגיבוב היא פונק' הזהות. כאן נממש את הפעולות באופן הבא:

- $Search\left(T,k\right)$: נחזיר את $T\left[k\right]$, כמו במערך, כלומר אם נרצה למצוא איש עם תעודת זהות נקרא ל-T [תעודת זהות].
- $Insert\left(T,x\right)$: כדי להוסיף איבר, פשוט נגדיר $T\left[x.key\right]=x$ כלומר איש $=$ [תעודת זהות] T.
- $Delete\left(T,x\right)$: נגדיר $T\left[x.key\right]=NIL$.

זה יראה כך:



איור 37: המחשה למיון ישיר

מקרה זה יעבוד כאשר כמות המפתחות להם אנו מצפים בפועל היא קטנה, יש כמות מספיקה זכרון ו-המפתח הגדול ביותר הוא קטן.

לדוגמה, אם נדע שיש 1000 תעודות זהות עם 3 ספרות, זה יהיה פתרון הגיוני.

לשיטה זו מספר יתרונות אך גם כמה בעיות

יתרונות:

- **אין** התנגשויות - מפתוח ישיר.
- זמן ריצה: בגישה לאיברים $\mathcal{O}(1)$ במקרה הגרוע ביותר.

בעיות:

- כאשר הערכים של האובייקטים גדולים, הפתרון מאוד בזבזני מבחינת זכרון.
- לא פרקטי עבור m גדול.

באופן כללי, נרצה פונקציה לא חח"ע $h(k)$ שממפה את k לאינדקס ב-T.

- נניח כי קבוצת כל המפתחות שנשתמש בהם בפועל יכולה להיות מאוד קטנה ביחס לכל שאר המפתחות $|U| >> m$.
- נפתור את בעית ההתנגשויות באמצעות:
  - שרשור.
  - מיון פתוח (Adressing Open).

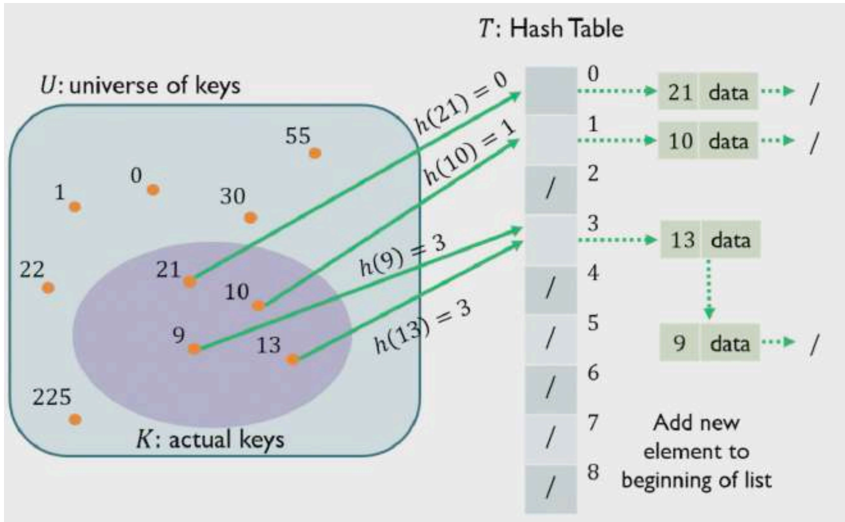
שרשור

בשרשור אנו פותרים את בעיית ההתנגשות באופן הבא:

- איברים עם מפתחות מתנגשים ממוקמים ברשימה מקושרת.
- הטבלה T תכיל בתא המתאים מצביע לראש הרשימה המקושרת.
- $Insert$: איבר חדש מתווסף לראש הרשימה המקושרת - במקרה הגרוע $\mathcal{O}(1)$.
- $Search$: מעבר על הרשימה המקושרת - במקרה הגרוע ביותר, $\mathcal{O}$ (אורך הרשימה הארוכה ביותר).

נביט בדוגמה הבאה להמחשה:

שימו לב שהמפתח שהוכנס ראשון לתא 3 הוא 9, לאחר פעולת $insert$ הוכנס המפתח 13 לראש הרשימה המקושרת בתא 3.



איור 38 : המחשה לשרשור

היתרונות בשיטה זו הם:

- הטבלה לא יכולה להתמלא.

החסרון:

- צריך לטייל על הרשימות המקושרות כדי להשיג איבר - במקרה הגרוע ביותר $\mathcal{O}\left(n\right)$ (במקרה הגרוע ביותר כל האיברים מופו לאותו תא, וכדי למצוא את האיבר האחרון ברשימה המקושרת נצטרך לעבור על כל איבריה - כלומר $\mathcal{O}\left(n\right)$).

שאלה איך ניפטר מחסרון זה?

תשובה כדי להיפטר מבעיה זו נרצה להבטיח שאורכי הרשימות המקושרות הן באותו אורך. איזו הנחה יכולה לעזור בזה? הנחת הגיבוב האחיד והפשוט!

נזכיר כי משמעות הנחה זו היא:

- מפתח רנדומלי בעל הסתברות זהה למיפוי לתאים שונים במערך.
- מיפוי מפתח חדש בלתי תלוי באיברים אחרים.

נגדיר מקדם עומס ע"י $\alpha = \frac{n}{m}$ כאשר:

- m הוא מספר התאים בטבלה.

- n מספר האיברים המאוחסנים בטבלה.

מקדם העומס למעשה אומר לנו כמה הטבלה עמוסה ביחסה לגודל שלה.

נביט למשל בדוגמה הבאה:

0	1	2	3	4	5	6	7
/	4	/	/	/	17	20	3

איור 39: דוגמה למערך לחישוב מקדם עומס

במקרה זה $n=4$, $m=8$ ולכן $\alpha=\frac{1}{2}$.

נשים לב שיתכן מצב בו $\alpha>1$, שכן לא הגבלנו את מספר האיברים בטבלה עם שרשור (יכולים להגיע עוד ועוד איברים ו-α רק יגדל). α חסום רק על ידי $\frac{|U|}{m}$.

בנוסף, מקדם העומס לא מתאר לנו את ההתפלגות של האיברים - אלא רק את **העומס הכולל**.

סיבוכיות זמן ריצה תחת הנחת הגיבוב הפשוט והאחיד

מבחינת זמן ריצה, נקבל כי:

- במקרה הגרוע ביותר: כל האיברים באותו התא (באותה רשימה מקושרת) ולכן סיבוכיות החיפוש היא $\Theta\left(n\right)$.
- בהנחה שיש לנו גיבוב אחיד ופשוט, הסיכוי שאיבר יגיע לתא מסוים שווה לסיכוי שיגיע לתא אחר ללא תלות באיברים אחרים ולכן האורך הממוצע של רשימה מקושרת בטבלה הוא מקדם העומס $\alpha=\frac{n}{m}$, שכן לכל אחד מ-n האיברים היה הסיכוי להגיע לרשימה מסוימת $\frac{1}{m}$ ולכן מכיוון שאורכי הרשימות שווים, אורך הרשימה הוא $\frac{n}{m}$.

משפט. (תחת הנחה הגיבוב והפשוט) סיבוכיות זמן הריצה לחיפוש בטבלת גיבוב המשתמשת בשרשור היא $\Theta\left(1+\alpha\right)$ כאשר α מקדם העומס.

מתקיים כי:

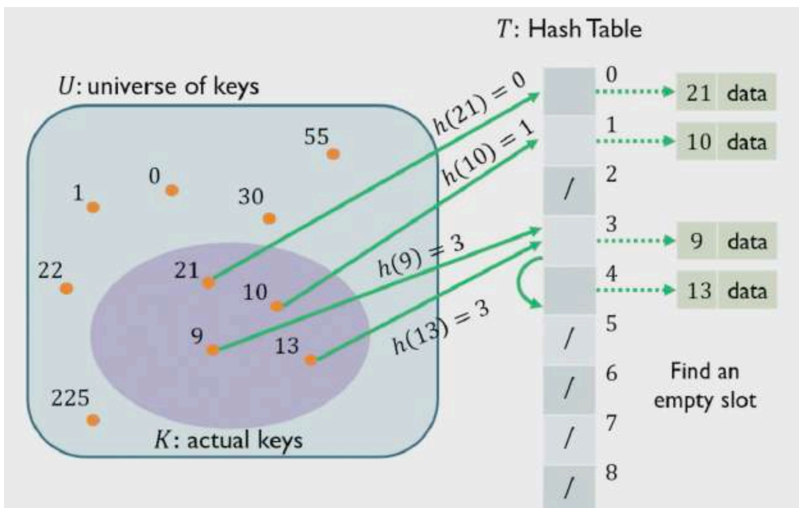
- התוספת של 1 היא הסיבוכיות לחישוב $h\left(k\right)$.
- אם $n=\Theta\left(m\right)$ אזי $\alpha=\Theta\left(1\right)$ ולכן סיבוכיות זמן הריצה של הפעולה $Search$ היא $\Theta\left(1\right)$.
- ההגיון הוא שאנו מוצאים את האינדקס בטבלה באמצעות h בסיבוכיות $\Theta\left(1\right)$ ואז מוצאים את האיבר ברשימה המקושרת באמצעות $\Theta\left(\alpha\right)$ ולכן סך הכל הסיבוכיות היא $\Theta\left(1+\alpha\right)$.

מיון פתוח

בעיה. אם אין לנו הרבה זכרון, זה יהיה מאוד בעייתי מבחינתנו אם במקום למלא את הטבלה, נמלא רשימה מקושרת בכל תא.

שאלה איך נפתור את זה?

תשובה למשל, במקום ליצור רשימה מקושרת, נדחוף כל איבר שמתנגש עם איבר אחד לתא אחר בטבלה, כפי שמוצג באיור הבאת בו מפתח 13 ממופה לתא 3 שכבר תפוס, לכן הוא מועבר לתא הפנוי הבא, תא 4:



איור 40: דוגמה להמנעות מיצירת רשימה מקושרת בטבלת גיבוב

כאשר אם התא הנ"ל תפוס, נעבור לתא הבא. כאן תמיד מתקיים $\alpha = \frac{n}{m} < 1$, הטבלה עלולה להתמלא.

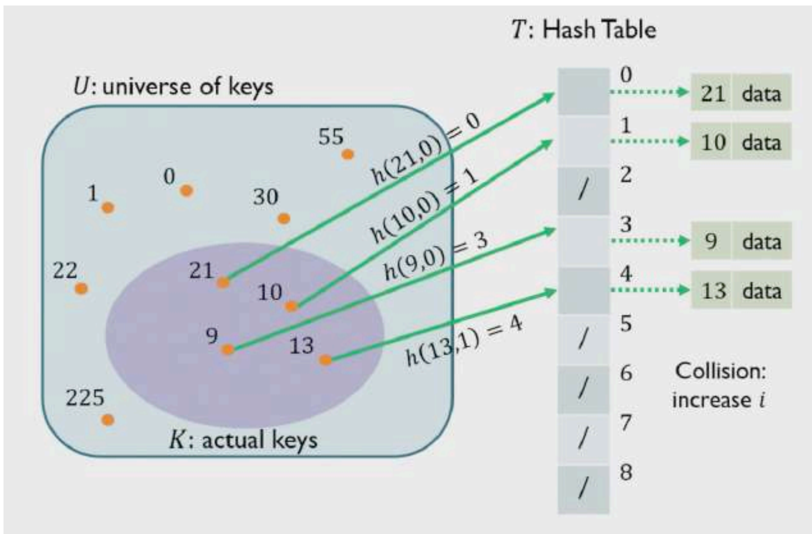
הערה. נשאלת השאלה איך נמצא את האיבר? כדי לעשות זאת, נצטרך לחשב את הגיבוב שלו, אם הוא שם, ניקח אותו, אחרת לא נדע אם הוא שם או הוזז, נצטרך לחפש את התא עד שנגיע או לאיבר או לסוף הטבלה. נוצרת כאן סיבוכיות שהיא בוודאי לא קבועה. לכן נצטרך למצוא מנגנון חדש.

על כן, במיון פתוח נשנה מעט את פונקציית הגיבוב:

- מפתחות מתנגשים ממופים לתא הפנוי הבא.
- חיפוש האיבר הפנוי הבא נקרא $probing$ ומשמעותו "חיפוש" או "גישוש".
- כל התאים מכילים איבר כלשהו או NIL.

- פונקציית הגיבוב מקבלת כארגומנט איבר נוסף: $h\left(k,i\right):U\times\{0,\ldots,m-1\}\rightarrow\{0,\ldots,m-1\}$, כאשר i הוא מספר צעד החיפוש שאנו מבצעים עכשיו. כלומר משמעותו היא "כמה קפיצות עשיתי בטבלה בגלל תאים מלאים" ולמעשה כל תא יאופיין עבור הפונקציה על ידי הזוג (k,i).

נביט בדוגמה הבאה להמחשה:



איור 41: דוגמה למיון פתוח

כאן למעשה אנו ממפים כל איבר בצורה רגילה כאשר $i=0$. לפתע 13 מתנגש עם 9 ולכן, במקום לבדוק האם התא $(13,0)$ פנוי, נבדוק האם התא $(13,1)$ הפנוי וכן הלאה... באופן הזה אנו יודעים למעשה מהו צעד החיפוש שצריך לעשות כדי למצוא את 13. במקרה בו הטבלה מתמלאת אנו מפסיקים להוסיף איברים.

פעולת הוספה $Insert$

גישוש עד שנמצא תא ריק, כלומר נגדיל את i עד שנקבל ש-$h\left(k,i\right)=NIL$.

- תלוי במפתח.
- אם אין תא ריק לאחר m חיפושים, נסיק כי הטבלה מלאה.

פעולת חיפוש $Search$

גישוש בדיוק כמו ב-$Insert$.

- אם מפתח נמצא - נחזיר את הערך שלו.
- אם קיבלנו NIL, כלומר לא הצלחנו להשיג את הערך, נסיק כי המפתח לא קיים בטבלה.

פעולת מחיקה $Delete$

מציאת המפתח בטבלה והשמת NIL בתא המתאים כך שנוכל להשתמש בו שוב.

תכף נראה שהוא מעט מקשה עלינו.

7.2 מימושים לפעולות $Insert, Delete, Search$

מימוש הוספה $Insert$

Algorithm 5 $Hash - Insert\left(T, k\right)$ - הוספת איבר לטבלת גיבוב

```
i = 0 // The search index
repeat
    j = h(k, i)
    if T[j] == NIL // Check if the cell is free
        return j
    else i = i + 1 // Increment search index
until i == m
error "overflow"
```

הפעולה המתבצעת היא למעשה בדיקת $T\left[h\left(k,i\right)\right]$, אם הוא NIL כלומר פנוי, נחזיר את $h\left(k,i\right)$. אחרת, נבדוק את $h\left(k,i+1\right)$ עד שנגיע ל-$h\left(k,m\right)$, אם הגענו למצב זה נחזיר שגיאה.

שאלה למה צריך להחזיר את j?

תשובה מהותית זה לא נחוץ, אמנם נראה בהמשך מדוע זה עוזר.

מימוש חיפוש $Search$

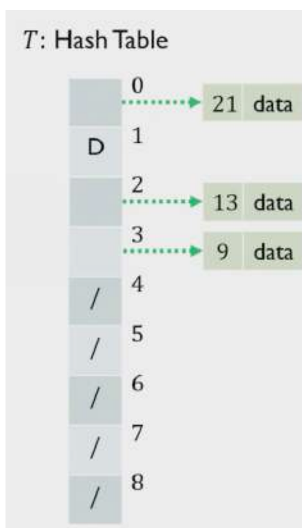
Algorithm 6 $Hash - Search\,(T, k)$ - חיפוש איבר בטבלת גיבוב

```
i = 0 // The search index
repeat
    j = h(k, i)
    if T[j] == k // Check if the cell contains the value
        return j
    else i = i + 1 // Increment search index
until T[j] == NIL or i == m
return NIL
```

כאן אנו מבצעים תהליך דומה כמו ב-$Insert$ רק שהעצירה מתבצעת כאשר הגענו לסוף הטבלה או כאשר הגענו לאיבר פנוי - אם הגענו לאיבר פנוי סימן שאין עוד סיבה לחפש כי אחרת האיבר היה בו.

מימוש מחיקה $Delete$

הבעיה בפעולת המחיקה שאין אנו יכולים פשוט למצוא את המקום של הערך ואז לשים בו NIL. למשל, בדוגמה הבאה:



איור 42: המחשה לבעיית המחיקה

נניח כי בתא במקום ה-1 בטבלה יש איבר וגם במקום השני. נרצה להוסיף את 9 ומשום מה קיבלנו התנגשות בתא ה-1. מהדרך בה אנו מוסיפים איברים, נקבל כי מיקומו של 9 הוא בתא ה-3. נמחק את האיבר בתא האחד ונרצה למחוק את 9, אמנם מכיוון שבתא ה-1 יש NIL, הפעולה $Search$ תחשוב ש-9 לא נמצא בטבלה. כדי להתמודד עם בעיה זו במקום לשים בתא NIL נשים בו D וכך נדע שהיה שם איבר נמחק.

אם כך, צריך לעדכן את $Search, Insert$ - לא נעשה זאת כרגע, ניתן לקרוא על כך בספר.

סיבוכיות הפעולות

- כל הפעולות $Search, Insert, Delete$ מגששות דרך האיברים.
- סיבוכיות זמן הריצה שלהן היא אורך הגישוש הרצוף הארוך ביותר.

שאלה האם ניתן לחסום אורך זה?

תשובה תחת הנחת הגיבוב הפשוט והאחיד נוכל לקבוע כי אורך זה הוא α.

חיפוש

נציג שלוש טכניקות חיפוש.

נעיר כי אף אחת מהשיטות להל"ן אינה מספיק טובה כדי שנקבל את הנחת הגיבוב הפשוט והאחיד. נראה בהמשך שיטות טובות יותר.

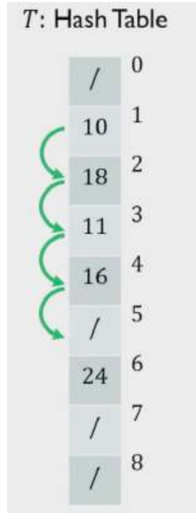
חיפוש לינארי

בחיפוש זה, נגדיר $h(k,i) = (h'(k) + i) \mod m$, מאוד דומה למה שעשינו עד כה.

- המודולו דואג לכך שלא נעבור את גבולות הטבלה.
- h' היא פונקציית עזר שהגיבוב שלה אינו תלוי בחיפוש.
- בהנתן מפתח k הגישוש יתבצע כך: $T[h'(k)], T[h'(k)+1], \ldots, T[h'(k)+m-1]$.

נשים לב כי ברגע שבחרנו מפתח k, כל רצף החיפוש נקבע.

נביט בדוגמה הבאה:



איור 43: המחשה לחיפוש לינארי

נרצה להוסיף את 9. כלומר לבצע את $Insert\left(9\right)$.

נניח כי $h'\left(9\right)=1$. אזי:

- $h\left(9,0\right)=1, h\left(9,1\right)=2, \ldots, h\left(9,4\right)=5$.

שאלה האם שיטה זו טובה?

תשובה ראשית, הפעולה היא בסיבוכיות זמן ריצה $\mathcal{O}\left(n\right)$, אמנם ככה גם שאר הפעולות. הבעיה עיקרית שהיא מובילה להצטברות איברים ולא לפיזור אחיד, שכן ברגע שמצאנו מקום פנוי נשתמש בו.

בעיה. הצטברות של רצפים מלאים. אם האיברים מגיעים בצורה אחידה ויש לנו רצף של תאים בגודל i, זה אומר שהתא ה-$i+1$ סביר מאוד להתמלא. דבר זה מעלה את הזמן הממוצע של הריצה.

חיפוש ריבועי

כדי להתמודד עם בעיית ההצטברות, הוצעה שיטת החיפוש הריבועי. בשיטה זו $h\left(k,i\right)=\left(h'\left(k\right)+c_1i+c_2i^2\right)\mod m$. גיבוב כזה יגרום לפיזור הרבה יותר.

לצורך המחשה, נניח כי $h\left(k,i\right)=\left(h'\left(k\right)+i+i^2\right)\mod 9$ כאשר הטבלה היא:

איור 44: המחשה לחיפוש ריבועי

במקרה זה נקבל כי אם $h'(9)=1$ אז החיפוש יהיה: $h(9,0)=1, h(9,1)=3, h(9,2)=7$.

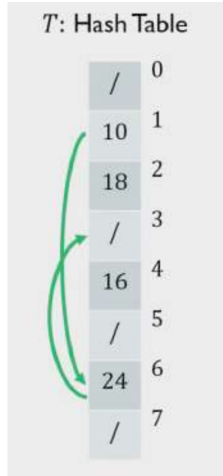
פתרון זה מונע הצטברות כמו בבעיה הקודמת אך יוצר בעיה חדשה של הצטברות משנית שבה תאים מתמלאים באותו הסדר.

חיפוש כפול

בשיטה זו נגדיר $h(k,i)=(h_1(k)+i\cdot h_2(k)) \mod m$ כאשר h_1, h_2 שתי פונקציות גיבוב.

- קודם כאשר התא הבסיסי היה נקבע, כל הרצף היה קבוע. כאן בגלל שיש שתי פונקציות גיבוב, רצף הפעולות יהיה אחר.
- אם שני תאים מופו בהתחלה לאותו התא, רצף החיפוש לא יהיה בהכרח זהה עבורם, כי יש שתי פונקציות גיבוב. כלומר לאיברים שונים, פיזורים שונים.

לדוגמה, נגדיר $h(k,i)=(h_1(k)+i\cdot h_2(k)) \mod 8$ ונביט ב-



איור 45: המחשה לחיפוש כפול

נניח כי $h_1(9) = 1, h_2(9) = 5$ אזי $h(9,0) = 1, h(9,0) = 6, h(9,2) = 3$. כאן המודולו מחזיר אותנו מסוף הטבלה לתא ה-3.

באופן כללי, נרצה פונקציות פיזור שיבטיחו פיזורים שונים לאיברים שונים.

ניתוח חיפוש פתוח

נשים לב כי שני החיפושים - הלינארי והריבועי היו דומים מאוד המותית שכן

- בחיפוש הלינארי, היו לנו m רצפים כי כל תא קבע רצף משלו.
- בחיפוש הריבועי היו לנו m רצפים כי כל תא קבע רצף משלו.

בחיפוש הכפול לעומת זאת, היו לנו m^2 רצפים כי בכל תא היו m אפשרויות לכל פונקציית $hash$.

תחת הנחת הגיבוב הפשוט והאחיד, כל $m!$ האפשרויות לרצפים יכולים לקרות לכל איבר וכשמגיע איבר אקראי חדש, הסיכוי שיקבל רצף מסוים זהה לכל רצף, שכן סך הכל $m!$ אלא רצפים בתאי הטבלה ולאיבר חדש יש את אותו סיכוי לקבל כל אחד מהרצפים האלה.

משפט. (תחת הנחת הגיבוב האחיד והפשוט) בהנתן טבלת גיבוב עם חיפוש פתוח ומקדם עומס $\alpha = \frac{n}{m} < 1$ רצף החיפושים הממוצע בחיפוש לא מוצלח (לא מצאנו איבר) הינו $\frac{1}{1-\alpha}$.

דוגמה. נניח שהטבלה מלאה עד מחציתה $\alpha = \frac{1}{2} \to \frac{1}{1-\alpha} = 2$ כלומר מספר הצעדים שנצטרך לעשות כדי להכריע שהאיבר אינו בטבלה הוא 2 בממוצע, **תחת הנחת הגיבוב הפשוט והאחיד!**

דוגמה. נניח כי $10 = \frac{1}{1-\alpha} \rightarrow \alpha = \frac{9}{10}$ במקרה זה נצטרך לעשות 10 צעדי חיפוש בממוצע.

שתי דוגמות אלה מתאימות לחיפוש לא מוצלח.

משפט. (תחת הנחת הגיבוב האחיד והפשוט) בהנתן טבלת גיבוב עם חיפוש פתוח ומקדם עומס $\alpha = \frac{n}{m} < 1$ רצף החיפושים הממוצע בחיפוש מוצלח (מצאנו את האיבר) הינו $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$.

נדבר על משפט זה יותר בשבוע הבא.

נסיק כי:

- אם אין לנו הרבה זכרון - עדיף לעבור עם חיפוש פתוח.
- אם יש לנו מספיק זכרון עדיף לעבור עם שרשור, כי במקרה זה $\alpha \approx 1$ ואז $\frac{1}{1-\alpha}$ מאוד מאוד גדול.

8 פונקציות גיבוב

שאלה מהי פונקציית גיבוב טובה ורעה? מה התכונות הרצויות?

- אנו רוצים שהיא תהיה כמה שיותר קרובה להנחת הגיבוב הפשוט והאחיד.
- ביצועי הפונקציה תלויים במקור המפתחות.
  - עבור תעודות זהות - אם הן לא אחידות זה משפיע על הפונקציה.
- פיזור טוב למפתחות רבים - לא רוצים רצפים!

הנחת הגיבוב הפשוט והאחיד

שאלה האם הנחה זו חיונית?

ראשית, נאמר כי הנחה זו לא סבירה ומהווה אידאל. זה נובע מכך ש-

- אנו לא יודעים את הקלטים מראש.
- אם בחרנו פונקציה h, לא יתכן ש-h תתן התפלגות אחידה על קבוצת כל המפתחות U.

על כן, h לא יכולה להיות תמיד פונקציה טובה לכל קבוצת ערכי מפתח K.

בניית פונקציית גיבוב

נרצה להשיג שני דברים:

1. לתכנן פונקציה שתעבוד טוב "רוב הזמן".
   - נשתמש בקירובים.
2. רנדומליזציה: נבחר פונקציות באופן רנדומלי בשביל ביצוע טוב ומניעת מקרי קיצון. כלומר נרצה שאדם חיצוני לא יוכל לתת קלט שייצור בעיות.

הערה. בבחירת פונקציה נרצה שהיא תהיה רנדומלית. אם למשל בחרנו שעבור תעודת זהות היא תחזיר את שתי הספרות האחרונות, ונדע שהתעודות מגיעות בצורה רנדומלית - נוכל לומר שהיא טובה. אמנם בפועל הן לא מגיעות כך.

פונקציות גיבוב מקורבות

אלה פונקציות שעונות על העקרונות הבאים:

- נרצה שהפונקציה תעבוד טוב ברוב המפתחות שמשתמשים בהם בפועל, ולא בהכרח על מפתחות שנבחרו בכוונות זדון.
- קבוצת המפתחות K לא רנדומלית ובעלת תבנית מסוימת.

המטרה שלנו למצוא פונקציות שמחלקות את U בצורה כזו שיהיו מעט מאוד התנגשויות.

הערה. היריסטיקה - כלל שבדרך כלל עובד.

שיטת החלוקה

נניח כי $U=\mathbb{N}=\{0,1,2,\ldots\}$. ונגדיר $h(k)=k \mod m$.

נבחין כי בחירה של $m=2^p$ אינה טובה, כי זו בחירה של p הביטים הראשונים בלבד של המספר k השמור במחשב.

אם כך, נרצה כי m יהיה שונה מחזקה של 2.

הערה. המחשב בנוי על בסיס 2 ולכן אם יש חוקיות בספרות הראשונות, נאבד את הרנדומליזציה. על כן ככל שהמספר רחוק יותר מחזקה של 2 ככה ניקח בחשבון יותר מהביטים שלו.

נניח למשל ש-$|U|=n=2000$ ונרצה למצוא $c\leq 3$ התנגשויות לכל תא. מה נעשה עם m?

מתקיים כי $\frac{n}{c}=\frac{2000}{3}=666$ מספר שלם. נמצא מספר ראשוני קרוב אליו שאינו חזקה של 2, $m=701$.

מכאן קיבלנו את הפונקציה $h(k)=k \mod 701$.

נשים לב שבמקרה זה $0,701,1402$ השלשה היחידה שמגיעה לתא ה-0.

שיטת המכפלה

נמפה את k לאחד מ-m התאים באופן הבא:

1. נכפיל את k בקבוע $0 < A < 1$.
2. נבחר את החלק השבור של התוצאה כלומר את $(kA - \lfloor kA \rfloor)$.
3. נכפיל ב-m וניקח את הערך השלם.

כלומר $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$.

דוגמה. למשל עבור $k = 1234, A = 0.4, m = 100$. נקבל כי $h(k) = 60$.

מסתבר ששיטה זו פחות רגישה לערך של m. הפרמטר A נוטה לערבב טוב את הערכים וליצור התפלגות סבירה, זו הוריסטיקה ולא הוכחה.

אם נבחר למשל m כחזקה של 2 ו- $A = \frac{\sqrt{5}-1}{2}$ נקבל היריסטיקה טובה. הבחירה של m נותנת חישוב מהיר יותר.

גיבוב אוניברסלי

(נראה בהמשך)

- נבחר פונקציית גיבוב באופן רנדומלי ללא תלות במפתחות.
- בחירה זה תספיק כדי להוכיח שהמערכת עובדת טוב בממוצע, כאילו שיש לנו את הנחת הגיבוב האחיד והפשוט.

בעיה. מאילו פונקציות נבחר?

פתרון נבחר ממספר סופי של אוסף אוניברסלי של פונקציות גיבוב.

חלק V

הרצאה $\mathbb{V}$ - גיבוב אוניברסלי, משפחות אוניברסליות וגיבוב מושלם

היכונו למתמטיקה!

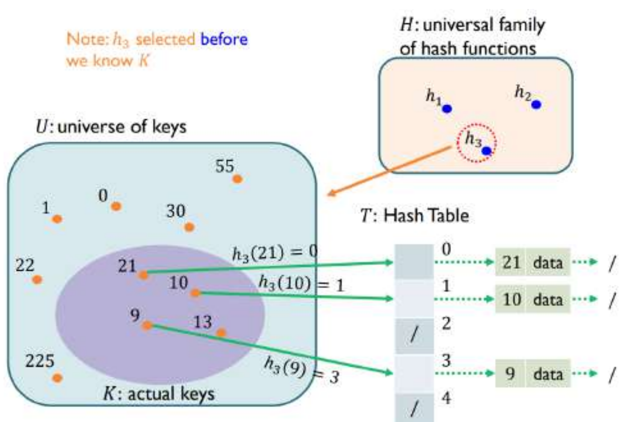
נזכור שטבלת גיבוב היא טובה כמו פונק' גיבוב שלה - בחירה גרועה של פונק' גיבוב תפגע מאוד ביעילות הטבלה: אם אין לנו מזל (וניתקל במקרה הגרוע ביותר), או שפונק' הגיבוב לא טובה, יהיו לנו המון התנגשויות.

על כן, לא נדבר היום על המקרה הגרוע ביותר כפי שצוין לעיל.

9 גיבוב אוניברסלי

נרצה למצוא שיטה שתאפשר לנו להתקל רב הזמן במקרה בממוצע, ולא הגרוע ביותר. רעיון אפשרי הוא בחירת פונק' גיבוב רנדומלית והחלפתה מדי זמן מה (לברר את המנגנון המדויק, לא צוין בהרצאה).

נצטרך להחזיק מבחר סופי, אוניברסלי, של פונק' גיבוב אותן נוכל לבחור ולהשתמש בהן.



איור 46: שימוש במאגר פונק' גיבוב

כפי שניתן לראות באיור, טרם ידועים לנו המפתחות שעלינו למפות, K, אנו בוחרים פונק' גיבוב מהמאגר שלנו, במקרה זה h_3.

נרצה לענות על השאלה: כיצד ניצור מאגר כזה של פונקציות, כך שבממוצע תתקיים הנחת הגיבוב האחיד והפשוט:

לכל זוג מפתחות k_1, k_2 ותאים y_1, y_2, ההסתברות ש-$h(k_1) = h(k_1)$ הוא $\frac{1}{m}$.

הגדרה. H יקרא אוניברסלי אם לכל $k_1 \neq k_2$, מספר פונק' הגיבוב $h \in H$ עבורם $h(k_1) = h(k_1)$ הוא לכל היותר $\frac{|H|}{m}$.

דוגמה. $|H| = 6, m = 3$. אז עבור $k_1 = 17, k_2 = 32$ לכל היותר 2 פונקציות יגרמו להתנגשות.

שאלה למה דווקא $\frac{|H|}{m}$? זאת כי אז ההסתברות להתנגשות במקרה זה היא $\frac{1}{m}$.

h נבחר רנדומלית, כך שהסיכוי לבחירה גרועה הוא קטן מ-$\frac{1}{m} = \frac{\frac{|H|}{m}}{|H|}$. בדיוק כמו הנחת גא"פ.

9.1 חיפוש מחיקה והוספה בשיטות שונות

9.1.1 חיפוש מחיקה והוספה בשרשור

משפט. (סיבוכיות זמן החיפוש הצפוי) תהי h פונק' מתוך קבוצת פונק' הגיבוב האוניברסלית המשמשת לגיבוב n מפתחות לטבלה T בגודל m (באמצעות שרשור) ויהי $\alpha = \frac{n}{m}$ מקדם העומס. אזי סיבוכיות החיפוש הצפויה היא:

אם $k \notin T$ אזי לכל היותר α (אורך הרשימה בממוצע).

אם $k \in T$ אזי לכל היותר $1 + \alpha$ (אורך הרשימה בממוצע).

זה נקרא האורך הצפוי של הרשמה כאשר k מגובב. התוצאה כאן היא ממש כמו בגא"פ! (זה המקרה הממוצע)

k לא בטבלה, אז $\frac{1}{m}$ מהמפתחות יגובבו לתא ה-k. על כן הרשימה תהיה $\alpha = n \cdot \frac{1}{m}$ בממוצע.

אם k כן בטבלה, אז $\frac{1}{m}$ מתוך ה-$n-1$ איברים שנותרו יגובבו לתא של k ולכן אורך הרשימה יהיה $1+\alpha = 1+\frac{n}{m} \leq 1+\frac{n-1}{m}$.

הוכחה: נסמן ב-n_i את אורך הרשימה שמחכה בתא ה-i. אזי

$$E[n_i] = \text{i-אורך הרשימה הצפוי בתא ה}$$

נסמן $X_{kl} = I[h(k) = h[l]]$ כלומר משתנה מקרי אינדיקטור להתנגשות בין המפתחות k, l.

מתכונת הגיבוב האוניברסלי, מתקיים כי $P_r\{h(k) = h\{l\}\} = P_r\{X_{kl} = 1\} \leq \frac{1}{m}$. עזי

$$E[X_{kl}] = P_r(X_{kl} = 1) = P_r\{h(k) = h(l)\} \leq \frac{1}{m}$$

נגדיר Y_k להיות משתנה מקרי מוגדר להיות $Y_k = \sum\limits_{l \neq k} X_{kl}$ כלומר סופר לנו את מספר המפתחות שהתנגשו עם k וזה אורך הרשימה בתא ה-k. נשים לב כי $Y = \sum\limits_{k=1}^{n} Y_k$ סופר את מספר ההתנגשויות בכל הטבלה. נחשב את התוחלת של Y_k.

$$E\left[Y_k\right] = E\left[\sum_{l \neq k} X_{kl}\right] = \sum_{l \neq k} E\left[X_{kl}\right] \leq \sum_{l \neq k} \frac{1}{m}$$

עתה נותר לחלק למקרים.

נניח כי לקחנו את k וחישבנו את $h(k)$, בהנחה שהוא לא באמת טבלה

אם $k \notin T$ אזי $n_{h(k)} = Y_k$ וכן $|\{l : l \in T \wedge l \neq k\}| = n$ וגם $E\left[n_{h(k)}\right] = E\left[Y_k\right] \leq \sum\limits_{l \neq k} \frac{1}{m} = \frac{n}{m} = \alpha$

עתה נניח כי אותו k בטבלה, אזי $n_{h(k)} = Y_k + 1$ וגם $|\{l : l \in T \wedge l \neq k\}| = n - 1$ ולכן

$$T\left[n_{h(k)}\right] = E\left[Y_k + 1\right] \leq E\left[1\right] + E\left[Y_k\right] \leq 1 + \frac{n-1}{m} = 1 + \alpha - \frac{1}{m} \leq 1 + \alpha$$

■

מסקנה. לטבלת גיבוב עם m תאים, על ידי שימוש בגיבוב אוניברסלי ושרשור, סדרת הוספות/חיפושים/מחיקות באורך n בעלת סיבוכיות זמן ריצה של $\Theta(n)$ כאשר אנו מניחים שמספר ההוספות הוא $\mathcal{O}(m)$.

הוכחה: אם יש לנו $\mathcal{O}(m)$ הוספות אז $\alpha = \mathcal{O}(1)$ ולכן כל חיפוש הוא $\Theta(1)$ לכן סדרת הוספות, חיפושים, מחיקות באורך n נקבל $n \cdot \Theta(1) = \Theta(n)$. ■

הערה. אנו בוחרים פונקציית גיבוב באופן ראנדומי כדי שתוקף לא יוכל לזהות את תבנית הגיבוב שלנו.

9.1.2 חיפוש מחיקה והוספה במיעון פתוח

משפט. (סיבוכיות החיפוש הממוצעת) תהי h פונקציה שנבחרה מאוסף אוניברסלי של פונקציות $hash$. נניח כי הפונקציה מגבבת n מפתחות לטבלה T בגודל m. נניח כי $\alpha = \frac{n}{m} < 1$ מקדם העומס. אז סיבוכיות החיפוש הממוצעת היא:

(i): אם $k \notin T$ היא לכל היותר $\frac{1}{1-\alpha}$.

(ii): אם $k \in T$ היא לכל היותר $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.

נציג עתה את עיקר ההוכחה של המשפט.

הוכחה: נגדיר מספר החיפושים בחיפוש לא מוצלח $= X$. נשאל מתי $X \geq i$?

$(i):(k \notin T)$ ניסינו לחפש את המפתח ולא מצאנו עד שהגענו לתא ה-i, כלומר כל התאים היו מלאים. אם כך

$$P_r\left\{X \geq i\right\} = \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdot \ldots \cdot \frac{n-i+2}{m-i+2} \leq \left(\frac{n}{m}\right)^{i-1} = \alpha^{i-1}$$

כאשר מכפלה זו נובעת מכך שההסתברות שהתא הראשון תפוס היא $\frac{n}{m}$ כי יש n איברים שיכולים להגיע לתא הראשון כאשר לכל איבר ההסתברות $\frac{1}{m}$, בהנחה שזה נכון, ההסתברות שהתא השני תפוס היא $\frac{n-1}{m-1}$ וכן הלאה. שימו לב שהסתברויות אלה הן הסתברויות <u>תלויות</u>. נחשב את התוחלת של X. מתקיים מנוסחת הזנב

$$\begin{aligned} E\left[X\right] &= \sum_{i=1}^{\infty} P_r\left\{X \geq i\right\} = \sum_{i=1}^{m} P_r\left\{X \geq i\right\} + \sum_{i=m+1}^{\infty} P_r\left\{X \geq i\right\} \\ &= \sum_{i=1}^{m} P_r\left\{X \geq i\right\} + 0 \leq \sum_{i=1}^{m} \alpha^{i-1} \leq \sum_{i=1}^{\infty} \alpha^{i-1} \\ &= \sum_{i=0}^{\infty} \alpha^{i} = \frac{1}{1-\alpha} \end{aligned}$$

$(ii):(k \in T)$. ההוכחה הבאה מבוססת על ההוכחה בספרו של קורמן.

נשים לב כי מספר הצעדים למצוא את k שקולים למספר הצעדים הדרושים להוסיף את k כלומר למצוא לו מקום בטבלה. פעולה זו שקולה גם כן למציאת איבר ריק בטבלה. ממקרה (i) נקבל כי אם k הוא האיבר ה-$i+1$ אז דרושים בממוצע $\frac{1}{1-\frac{i}{m}} = \frac{m}{m-i}$ צעדים כדי למצוא לו מקום. לכן אם נחשב את הממוצע על כל המקרים של k נקבל

$$\begin{aligned} \frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} &= \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}\sum_{j=m-n+1}^{m}\frac{1}{j} \\ &\leq \frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}\,\mathrm{dx} = \frac{1}{\alpha}\left(\ln\left(\frac{m}{m-n}\right)\right) \\ &= \frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right) \end{aligned}$$

כרצוי. ■ על כן ראינו שבשרשור, במקרה הגרוע יש רשימה מקושר ובמקרה הממוצע $\alpha, 1+\alpha$. עבור מיעון פתוח - $\frac{1}{1-\alpha}, \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$.

10 בניית משפחה אוניברסלית

נבחר $p > m$ ראשוני כך ש-$\forall u \in U : p > u$.

נגדיר $\mathbb{Z}_p = \{0, ..., p-1\}$ כאשר $\forall a, b \in \mathbb{Z}_p, a \neq 0$ נגדיר $h_{a,b}(k) = ((ak+b) \mod p) \mod m$ והמשפחה היא

$$H_{p,m} = \{h_{a,b} \mid a, b \in \mathbb{Z} \wedge a \neq 0\}$$

כלומר נבחר פונקציה באופן רנדומי, נבחר $a, b \in \mathbb{Z}, a \neq 0$ באופן רנדומי. יש לכך $p(p-1)$ אפשריות.

דוגמה. $p = 17, m = 6$ נבחר $h_{3,4}(k) = ((3k+4) \mod 17) \mod 6$ ולכן $h_{3,4}(8) = ((3 \cdot 8 + 4) \mod 17)$ $\mod 6 = 5$.

טענה. משפחה זו היא משפחה אוניברסלית.

הוכחה: יהיו $k_1 \neq k_2$ ונגדיר $r = (ak_1 + b) \mod p, s = (ak_2 + b) \mod p$. נוכיח כי $r \neq s$ זה נובע מכך ש-p ראשוני ומתורת המספרים. מכאן נסיק כי אין התנגשויות עד שנעשה את $\mod m$. יתר על כן, כל בחירת $\langle a, b \rangle$ נותנת זוג $\langle r, s \rangle$ אחר. דבר זה מבטיח פיזור אחיד ושוויוני עבור פונקציות הגיבוב, כלומר זוגות של פונקציות לא יכולות לתת את אותם זוגות של מפתחות.

אם כך, אם נבחר באופן אקראי ואחיד אז גם ערכי הגיבוב שנקבל הם אקראיים ואחידים.

דוגמא $p = 5, a = 2, b = 3$ נבחר $k_1 = 4, k_2 = 3$ אזי

$$(2k_1 + 3) \mod 5 = 11 \mod 5 = 1 \neq (2k_2 + 3) \mod 5 = 9 \mod 5 = 4$$

זה לא מקרי, זה תמיד יהיה כך.

נשאל, מהו הסיכוי ש-$r \equiv s \pmod m$? מתקיים כי

$$\Pr\{h_{a,b}(k_1) = h_{a,b}(k_2)\} \leq \frac{1}{m}$$

לא נוכיח טענה זו, ונזמין את הקורא להביט בהוכחה שבספר הקורס. ■ לכן קיבלנו $\mathcal{O}(1)$ בממוצע לכל אוסף של מפתחות כאשר אנו מניחים שפונקציית הגיבוב נבחרת באופן ראנדומי וגם a, b.

ולמרות שהמקרה הגרוע שונה, הסיכוי לקבל אותו נמוך.

יחד עם זאת, עבור אוסף משתנה לא נוכל לדעת מה יהיה הביצוע.

11 גיבוב מושלם

גיבוב אוניברסלי מבטיח ביצוע מצוין לכל אוסף מפתחות, ומבטיח שהסיכוי לביצוע לא טוב הוא נמוך מאוד,

אבל במקרים מסוימים, אפשר לעשות יותר טוב.

בגיבו מושלם, אנו פותרים בעיה אחרת, עבור אוסף סטאטי של מפתחות כלומר אנו יודעים מראש את אוסף המפתחות. תחת הגבלה זו, נקבל כי סיבוכיות החיפוש היא $\mathcal{O}(1)$ עבור המקרה הגרוע ביותר.

דוגמה. מילים שמורות בשפות תכנות, רשימת ערים במדינה.

להמחשה, ידוע לנו כי המפתחות הם $21, 10, 13, 9$. נניח כי נתונות הפונקצות h_1, h_2, h_3, נבחר את h_3.

בגיבוב אוניברסלי סדר הפעולות היה באופן הבא:

- נבחר פונקציה מאוסף אוניברסלי של פונקציות hash.
- נגלה את קבוצת המפתחות K.

לעומת זאת, בגיבוב מושלם, סדר הפעולות הוא כדלקמן:

- נקבל את קבוצת המפתחות K.
- נבחר פונקציית hash מתאימה שתתן תוצאות טובות עבור K - לא יהיו בה התנגשויות בכלל.

משפט. נניח כי יש לנו אוסף של n מפתחות ונרצה למפות אותם לטבלת גיבוב בגודל $m = n^2$. נניח כי אנו משתמשים בפונקציית hash שנבחרה באופן ראנדומי ממשפחה אוניברסלית. אזי ההסתברות לאיזשהי התנגשות בין מפתחות קטנה מ-$\frac{1}{2}$.

הוכחה: יש לנו $\frac{n(n-1)}{2} = \binom{n}{2}$ זוגות שיכולים להתנגש אחד עם השני וכל זוג מתנגש עם הסתברות $\frac{1}{m}$ שכן פונקציית הגיבוב היא מאוסף אוניברסלי. יהי X מספר ההתנגשויות ובפרט יהי X_{ij} אינדיקטור למקרה בו המפתח ה-i התנגש עם המפתח ה-j, אזי $X = \sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}$ עבור $m = n^2$ נקבל כי

$$E[X] = E\left[\sum_{i=1}^{n}\sum_{j=i+1}^{n} X_{ij}\right] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} E[X_{ij}] = \sum_{i=1}^{n}\sum_{j=i+1}^{n} \frac{1}{m} = \binom{n}{2}\frac{1}{m}$$
$$= \frac{n(n-1)}{2} \cdot \frac{1}{n^2} = \frac{n^2-n}{2n^2} = \frac{1}{2} - \frac{1}{2n} < \frac{1}{2}$$

כרצוי. ■

מסקנה. נבחר פונקציית hash ונבדוק אותה על המפתחות, אם יש התנגשות נבחר אחת אחרת. מכיוון שההסתברות לבחירת פונקציה לא טובה היא פחות מ-$\frac{1}{2}$, נדע שנוכל למצוא פונקציה טובה די מהר.

החסרון בשיטה זו הוא שדרוש $\Theta\left(n^2\right)$ זכרון. למרות זאת, גיבוב מושלם מאפשר לעשות זאת עם $\Theta\left(n\right)$!

כדי לעשות זאת, נרצה למקם את המפתחות המתנגשים באותו תא, אך לא באמצעות רשימות מקושרות, אלא באמצעות טבלת גיבוב נוספת שנסמנה S_j.

בתוך טבלה זו נוודא שאין התנגשויות בין המפתחות שהתנגשו בטבלה המקורית.

- טבלת הגיבוב המקורית: $h\left(k\right) = \left(\left(ak+b\right) \mod p\right) \mod m$.
- טבלת הגיבוב בתא ה-j: $h_j\left(k\right) = \left(\left(a_jk+b_j\right) \mod p\right) \mod m_j$.

תהליך המיפוי יהיה כך:

- נחשב את $h\left(k\right)$.
- נחשב את $h_{h(k)}\left(k\right)$ כדי למפות את k לטבלה ה-$h\left(k\right)$. למעשה $h\left(k\right)$ הוא אינדקס תת הטבלה בטבלה המלאה.

גודל כל טבלת פנימית היא ריבוע מספר המפתחות שבה, על מנת להבטיח שלא יהיו התנגשויות כמו במשפט שהוכחנו.

מהמשפט שראינו, נובע כי אין התנגשות בתוך הטבלות הפנימיות. נרצה לנתח את סיבוכיות הזכרון. יש לנו n תאים בטבלה המקורית ועוד $\sum\limits_{j=1}^{n} n_j^2$ תאים בכל הטבלות ביחד. לכן $S\left(n\right) = n + \sum\limits_{j=1}^{n} n_j^2$. נרצה לחשב את $E\left[S\left(n\right)\right]$. אינטואיטיבית, במקרה הרע, אם כל המפתחות ימופו לאותו תא, נקבל כי $S\left(n\right) = \mathcal{O}\left(n^2\right)$. במקרה הטוב, נקבל כי כל התאים נתפסו בטבלה המקורית ואז $S\left(n\right) = \mathcal{O}\left(n\right)$.

משפט. נניח כי אנו רוצים למפות n מפתחות לטבלת גיבוב בגודל $m = n$ ויש לנו פונקציית גיבוב ממשפחה אוניברסלית שנבחרה באופן ראנדומי h. אזי $E\left[\sum\limits_{j=0}^{m-1} n_j^2\right] < 2n$.

הוכחה: נשתמש בזהות $a^2 = a + 2\binom{n}{2}$ אזי

$$E\left[\sum_{j=0}^{m-1} n_j^2\right] = E\left[\sum_{j=0}^{m-1}\left(n_j + 2\binom{n_j}{2}\right)\right] = E\left[\sum_{j=0}^{m-1} n_j\right] + 2E\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right]$$
$$= E\left[n\right] + 2E\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right] = n + 2E\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right]$$

אבל מהי $E\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right]$? נבחין כי $\sum_{j=0}^{m-1}\binom{n_j}{2}$ זהו מספר כל האפשרויות להתנגשויות שיש בכל תתי הטבלות יחד, לכן אלה כל האפשרויות להתנגשויות בתוך הטבלה, שהם כמו קודם $\binom{n}{2}$, מכיוון ש-h היא ממשפחת אוניברסלית, לכל התנגשות הסתברות $\frac{1}{m}$ ולכן,

$$E\left[\sum_{j=0}^{m-1}\binom{n_j}{2}\right]=\binom{n}{2}\frac{1}{m}=\frac{n\left(n-1\right)}{2n}=\frac{n-1}{2}$$

ולכן $E\left[\sum_{j=0}^{m-1}n_j^2\right]=n+2\frac{n-1}{2}=2n-1<2n$. ■

מסקנה. כמות הזכרות הנדרשת בממוצע בטבלות הפנימיות היא קטנה מ-$2n$. על ידי בחירת פונקציית גיבוב מתאימה, נמצא אחת טובה.

לסיכום, קיבלנו שבגיבוב מושלם חיפוש הוא $\mathcal{O}\left(1\right)$ במקרה הגרוע, וכמות הזכרון בממוצע לקבוצת מפתחות דינאמית היא $\mathcal{O}\left(n\right)$. התנגשויות נפתרות על ידי מיעון פתוח או שרשור. גיבוב אוניברסלי מבטיח תכונה זו בממוצע.

בקצרה, גיבוב מושלם מבטיח חיפוש בזמן קבוע לקבוצת מפתחות נתונה.

חלק VI

הרצאה $\mathbb{VI}$ - ערימות ותורי עדיפויות

$(Heaps\ and\ ProrityQueues)$

מבוא

בעיה אנו מארגנים מכרז כדי למכור מוצר. מגוון לקוחות מציעים מחירים, משנים את ההצעות שלהם או מבטלים אותן.

מטרתנו היא לזכור מי הציע את ההצעה המקסימלית בכל רגע נתון. באיזה **מבנה נתונים** כדאי להשתמש?

הערה באופן סימטרי ניתן להציג את אותה הבעיה (עם פתרון אנלוגי) בו נרצה לזכור מי הציע את ההצעה המינימלית.

מבנה נתונים אחד שנוכל לבחור בו הוא טבלת גיבוב. עם זאת, לטבלות גיבוב אין סדר באיברם, לכן יהיה קשה להבין מי האיבר המקסימלי בכל רגע נתון, שכן אם ההצעה המקסימלית ברגע מסוים נמחקה, יהיה קשה לנו למצוא את ההצעה המקסימלית החדשה.

מבנה נתונים אחר שנוכל לבחור בו הוא מערך ממוין. תמיד נדע מי המקסימום, זהו פשוט האיבר האחרון. אנו יודעים כיצד למחוק איברים או להוסיף איברים. עם זאת, מערך ממוין הוא “overkill” כאן. אנו לא צריכים לזכור עבור כל ההצעות מי יותר גדול ממי ולטרוח למיין את כל המערך.
כידוע לבנות מערך ממוין עולה לנו $n\log n$, מבנה הנתונים שנציע עתה, תור עדיפויות, יעלה לנו רק n, שכן נוותר על המידע המיותר.

11.1 תור עדיפויות (PriorityQueue)

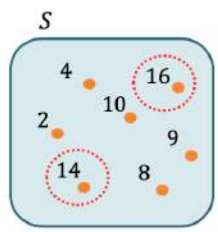
פעולות תור עדיפויות/קדימויות S תומך בפעולות הבאות:

1. $Max\left(S\right)$ - מחזיר את האיבר המקסימלי.
2. $Insert\left(x,S\right)$ - מכניס לתור את האיבר x.
3. $Extract-Max\left(S\right)$ - מסיר ומחזיר את האיבר המקסימלי.
4. $Increase-Key\left(x,k,S\right)$ - מגדיל את איבר x ב-k.

דוגמה. נניח למשל שאנו מבצעים את הפעולות:

- $Insert\left(9,S\right)$.
- $Max\left(S\right)\rightarrow16$.
- $Extract-Max\left(S\right)\rightarrow16$.
- $Max\left(S\rightarrow14\right)$.
- $Increase-Key\left(8,9,S\right)$ הופך את 8 ל-17.

התור עם האיברים $\left(4,10,16,2,9,8,14\right)$. ראש התור יהיה 16 והאיבר לפניו יהיה 14. שאר האיברים מפוזרים בשאר התור בסדר חלש יותר, אותו נראה בהמשך. ניתן לקבל המחשה באיור הבא:



איור 47: איברים בתור עדיפויות

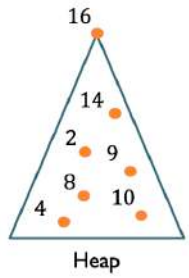
שימו לב כיצד $Extract-Max\left(S\right)$ מחזיר 16 ובגלל שאחרי הפעולה הזו, 16 כבר לא נכלל בתור, $Max\left(S\right)$ החזיר לנו 14. בנוסף, $Increase-Key\left(8,9,S\right)$ הפך את 8 ל-17.

12 ערמה (Heap)

בשביל לממש תור עדיפויות נשתמש במבנה נתונים הנקרא "ערמה". ערמה היא אוסף איברים, כאשר מעניין אותנו רק מי האיבר שנמצא בראש ובמקרה זה נדאג שהאיבר המקסימלי יהיה בראש, שכן הוא זה שמעניין אותנו.

הערה אנו אומרים שאנו יוצרים תור קדימויות באמצעות ערמה, בעוד ששניהם מבני נתונים מופשטים (שנממש באמצעות מערך). אנו עושים זאת משיקולים היסטוריים, למרות שעבורנו הם יהוו את אותו הדבר.

אז מה היא ערימה? למשל, עבור הדוגמא קודמת, הערימה המתאימה תראה כך:



איור 48: יצוג תור הקדימויות מהדוגמא הקודמת באמצעות ערימה

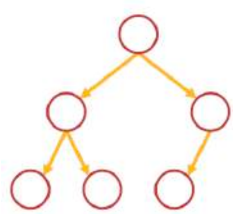
באיור מתוארת הדוגמה הקודמת, כפי שהיא נראית בערמה. שימו לב כיצד משתנה האיבר בקודקוד בהתאם למי הגדול ביותר, וכי הסדר של שאר האיברים לא באמת חשוב לנו, אלא רק היחס בין הילדים לאבא שלהם.

תכונות על הערמה לתמוך באותן פעולות כמו תור עדיפויות.

נבחר לממש ערמה באמצעות מערך, עם מבנה של עץ בעל כמה תכונות מיוחדות.

- ערמה היא עץ בינארי שלם. כלומר, כל הרמות של העץ מלאות מלבד הרמה האחרונה. ברמה האחרונה האיברים יהיו מלאים משמאל לימין, כלומר אם ברמה האחרונה חסרים איברים, אז הם יהיו מצד ימין. האיברים הקיימים יהיו לחוצים בצד שמאל.

באיור הבא יש המחשה לתכונה זו.



איור 49: עץ בינארי שלם

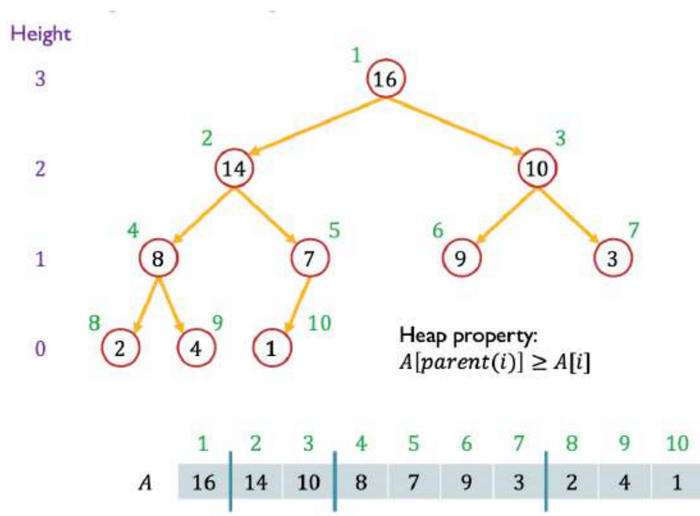
נשים לב שלכל עץ בינארי שלם בעל n איברים, יש בדיוק דרך אחת בו הוא יכול להראות. כמובן, נוכל לסדר בתוכו את האיברים כרצוננו, אך כמבנה המכיל אותם הוא יחיד. מכאן נבין כי ייצוג ערימה כעץ הוא יצוג יחודי מבחינת מבנה.
באיור הנ"ל יש 6 איברים, כל עץ בינארי שלם עם 6 איברים יהיה בעל המבנה הזה, למרות שהערכים בכל צומת בעץ יכולים להיות שונים, אנו מדברים כאן רק על איך נראה המבנה.

יחד עם זאת, אנו לא נממש את הערמה כעץ במובן שיהיו לנו מצביעים לאיברים וכדומה, אלא באמצעות מערך, למרות שיהיה לנו נוח לחשוב על המבנה בתור עץ. למעשה, כשנדבר על ערימות במובן הויזואלי - נדבר על עצים וכשנדבר על ערימות בכתיבת אלגוריתמים נדבר אך ורק על מערכים.

תכונת הערמה הערך השמור בצומת ילד $\geq$ הערך השמור בצומת אב ⊳ כלומר $\forall i > 1,\ A\left[parent\left(i\right)\right] \geq A\left[i\right]$

תכונה זו מבטיחה לנו ששורש העץ, $i = 1$, הוא האיבר המקסימלי. בנוסף, כל תת-עץ של הערמה, הוא ערמה בפני עצמו.

גובה גובה של רמה הוא המרחק של צמתי הרמה מהרמה האחרונה. כל פעם שאנו עולים רמה אחת למעלה (אל עבר הקודקוד) הגובה גדל ב-1.



איור 50 : יצוג ערימה באמצעות עץ

באיור יש עץ בינארי שלם, המקיים את תכונת הערמה כי הערך בכל צומת קטן מערך האב, ולכן מדובר בערמה.

באופן דומה קל לראות שגם כל תת-עץ הוא אכן ערמה.

הערה. הערה נשים לב שיתכן מצב בו איבר a_1 נמצא ברמה יותר נמוכה מ-a_2, ועדיין $a_1 > a_2$. לדוגמה, אם באיור במקום 8 היינו שמים את האיבר 12, היינו מקבלים שתכונת הערמה מתקיימת, גם אם 12 נמצא ברמה יותר נמוכה מ-10.

כלומר, אם שני צמתים הם **אינם** צאצאים אחד של השני, אין ביניהם קשר **ולא** נוכל להסיק לגביהם כמעט כל מסקנה.

12.1 יצוג ערמה כמערך

עתה נבין כיצד מייצגים ערמה כמערך. אנו עוברים מראש העץ ומטה משמאל לימין ומוסיף איברים למערך. באיור אנו מתחילים ברמה הראשונה ומקבלים את המערך $[16]$. לאחר מכן הגענו לרמה השנייה וקיבלנו את המערך $[16 \mid 14, 10 \mid]$ וכן

הלאה עד שקיבלנו את המערך הכתוב באיור. דבר זה קריטי, שכן היינו רוצים לחשוב שעל ידי הורדת ראש הערמה מהמערך, היינו מקבלים שתי ערמות חדשות שמיוצגות באופן תקין על ידי מערך, אבל לפי הבנייה שלנו אנו מקבלים רק את המשך הערמה המקורית כמערך ולא שני חצאים בהם הערמה הימנית והשמאלית. דבר זה נראה בעייתי, אך תכף נראה שהוא דווקא מועיל.

נשאלת השאלה, כיצד משתמשים במידע זה כדי לגשת לאב של קודקוד או ליד של קודקוד? אם זה היה עץ, היינו עושים זאת בקלות. אבל מה עכשיו?

נבין כי בהנתן אינדקס i שמיצג קודקוד הנמצא בתחילת הרמה h, הילד השמאלי של i יהיה האיבר הראשון ברמה הבאה. כמה אינדקסים עלינו לעבור עד שנגיע לילד זה ברמה ה-$h+1$? $\underbrace{1+1+\ldots+1}_{2^h}=2^h$ ומהו האינדקס ה-i? הוא

$$\underbrace{1+2+2^2+...+2^{h-1}}_{\text{מספר הקודקודים עד הרמה ה-h-1 כולל}}+\underbrace{1}_{\text{עוד אחד כדי להגיע לרמה הבאה}}=\frac{2^h-1}{2-1}=2^h-1=2^h$$

ולכן האינדקס של הילד השמאלי הוא $2\cdot i=2^h+2^h$. עבור הילד הימני נצטרך לזוז עוד פעם אחת ולכן נקבל $2^h+2^h+1=2\cdot i+1$. מכאן נסיק כי עבור ילד i, אינדקס האב שלו j יקיים את הקשר $i=\begin{cases}2j+1 & \text{ימני}\\ 2j & \text{שמאלי}\end{cases}$ ולכן $j=\left\lfloor\frac{i}{2}\right\rfloor$.

אם כך, בשביל לעבור בקלות בין איברים בעץ יש לנו מספר פונק' המחזירות לכל i את הערכים הבאים:

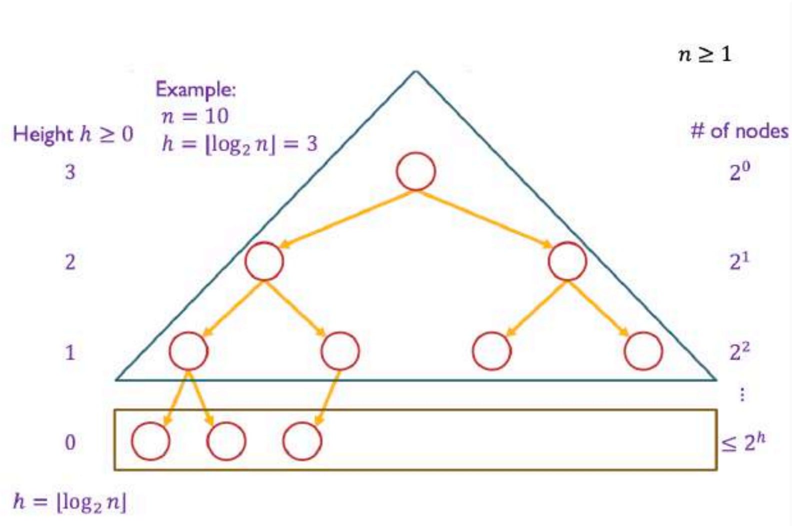
(i) $parent\,(i)=\left\lfloor\frac{i}{2}\right\rfloor$.

(ii) $left-child=2i$.

(iii) $right-child\,(i)=2i+1$.

12.2 תכונות של עצים בינאריים

נביט באיור הבא להמחשה:



איור 51 : עץ בינארי שלם

לעצים בינארים שלמים התכונות הבאות :

1. מספר הצמתים בעץ הוא $2^h \leq n \leq \underbrace{2^{h+1}-1}_{\text{מספר הצמתים בעץ בינארי מושלם}}$ כאשר h הוא גובה העץ. נשים לב כי $1 \leq n$.
2. מספר הצמתים הפנימיים (ללא הרמה התחתונה) הוא $\sum_{i=0}^{h-1} 2^i = 2^h - 1$. בכל רמה i יש 2^i צמתים.
3. מספר הצמתים ברמה האחרונה הוא $1 \leq i \leq 2^h$.
4. מספר הצמתים בעץ מושלם (בו הרמה התחתונה מלאה) הוא $\sum_{i=0}^{h} 2^i = 2^{h+1} - 1$.
5. גובה העץ הוא $h \geq 0$, $h = \lfloor \log_2 n \rfloor$. רמות העץ נעות בין 0 ל-$h$.

12.3 שמירה על תכונת הערמה

נרצה לממש מספר פעולות, תוך שמירה מתמדת על תכונת הערמה $\forall i > 1 : A[parent[i]] \geq A[i]$:

(i) מציאת האיבר המקסימלי - $\mathcal{O}(1)$. (זהו פשוט $A[1]$)

(ii) הכנסת/מחיקת/עדכון איבר - $\mathcal{O}(\log n)$.

(iii) $Max - Heapify$ - מתודת עזר שתעזור לשמור על תכונת הערמה.

כאשר אנו מכניסים/מעדכנים איבר נשתמש ברעיון הבא :

(i) נדחוף את האיבר למטה כל עוד הוא קטן מאחד מצמתי הבן שלו.

(ii) נחליף מקומות עם צומת הבן בעל הערך הגדול ביותר.

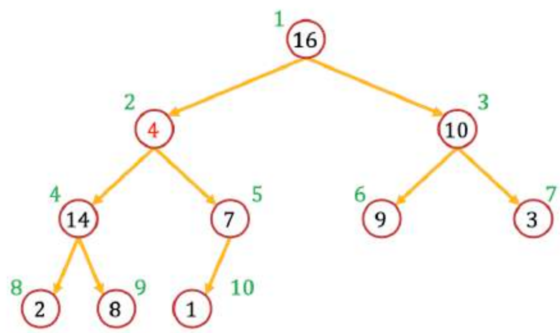
(iii) נמשיך בתהליך באופן רקורסיבי.

היות שלעץ יש $h=\lfloor\log n\rfloor$ רמות ובמקרה הגרוע אנו עוברים בכל רמה, תהליך זה יעלה $\mathcal{O}\left(\log n\right)$.

12.4 $Max-Heapify\left(A,i\right)$

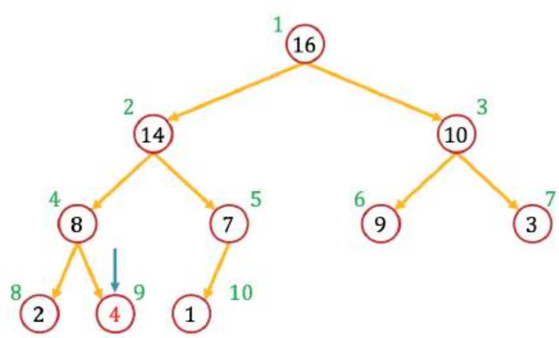
$Max-Heapify\left(A,i\right)$ מבצעת תיקון הפרה של תכונת הערמה בהנחה שכל תת-עץ הוא כבר ערמה. היא מבצעת זו לפי רעיון הדחיפה שצוין לעיל.

דוגמה. נביט בערמה שבה 16 באינדקס 2 ו-4 באינדקס 1. הפעולה $Max-Heapify\left(A,1\right)$ תחליף את 16 עם 4 ונקבל את הערמה הבאה:



איור 52: המחשה לתהליך הוספת איבר

לאחר מכן, היא תחליף את 4 עם 14 ולאחר מכן תחליף את 4 עם 8 ככה שמתקבלת הערמה:



איור 53: הערמה לאחר הרצה $Max-Heapify\left(A,1\right)$

נביט עתה באלגוריתם $Max-Heapify\left(A,i\right)$ וננתח אותו.

Algorithm 7 $Max - Heapify\left(A, i\right)$

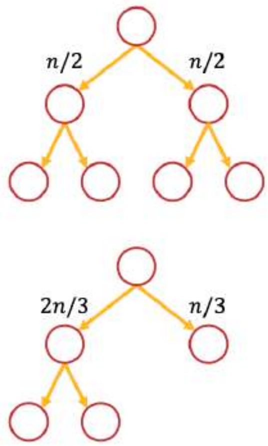
$l \leftarrow left\left(i\right)$
$r \leftarrow right\left(i\right)$
if $l \leq heapsize\left(A\right)$ **and** $A\left[l\right] > A\left[i\right]$
  then $largest \leftarrow l$
  else $largest \leftarrow i$
if $r \leq heapsize\left(A\right)$ **and** $A\left[r\right] > A\left[largest\right]$
  then $largest \leftarrow r$
if $largest \neq i$
  then $Exchange\left(A\left[i\right], A\left[largest\right]\right)$
    $Max - Heapify\left(A, largest\right)$

Identify $largest$

12.4.1 סיבוכיות

נשים לב כי המתודה תלויה בדברים הבאים:

- זמן הנדרש לטיפול בתת-עץ בגודל n עם שורש i:
  - $\Theta\left(1\right)$ לתיקון הבעיה ב-$A\left[i\right]$.
  - קריאה רקורסיבית ל-$Max - heapify$ על התת-עץ של כל אחד מהילדים של $A\left[i\right]$.
- גודל תת העץ:
  - עץ מאוזן - $\frac{n}{2}$.
  - עץ לא מאוזן - $\frac{2n}{3}$, כאשר התת-עץ האחרון חצי מלא.



איור 54 : היחס בין תתי העצים בעץ לא מאוזן

על כן נוסחת הנסיגה שנקבל מקיימת

$$T\left(\frac{n}{2}\right)+\Theta\left(1\right)\leq T\left(n\right)\leq T\left(\frac{2n}{3}\right)+\Theta\left(1\right)$$

חסם תחתון - $\log n\leq T\left(n\right)$, כמו בחיפוש בינארי.

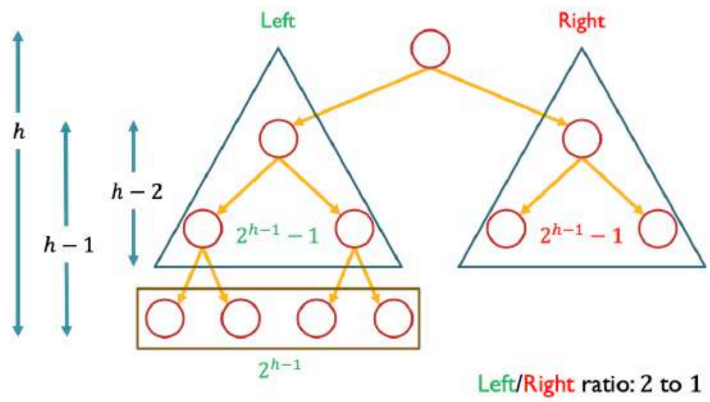
חסם עליון- $T\left(n\right)=\mathcal{O}\left(\log n\right)$, משתמשים במקרה 2 במשפט האב, כאשר $a=1,b=\frac{3}{2},c=0$.

על כן, $T\left(n\right)=\Theta\left(\log n\right)$.

שאלה מדוע המקרה הלא-מאוזן הוא $\frac{2n}{3}$?

תשובה נזכור כי מספר הצמתים בעץ מושלם בגובה t הוא $2^{t+1}-1$. נשים לב שכמתואר באיור נקבל יחס $right/left$ של 2 ל-1.

לכן עבור עץ בגודל n נקבל כי $\frac{3}{2}n=\frac{\frac{3}{2}n}{n}=\frac{\frac{1}{2}n+n}{n}=\frac{right+left}{left}$ כרצוי. ניתן לקבל לכך המחשה באיור הבא :



איור 55 : המחשה ליחס בין גדלי תתי העצים בעץ לא מאוזן

12.5 בניית ערמה

לשם בניית ערמה נשתמש ב-$Max-heapify$ באופן רקורסיבי מלמטה למעלה.

האיברים ברמה האחרונה נמצאים כבר במקום, כלומר $A\left[\left(\lfloor\frac{n}{2}\rfloor+1\right),\ldots,n\right]$ במקום. נרצה לבנות ערמה מקסימלית.

נעשה זאת באופן הבא :

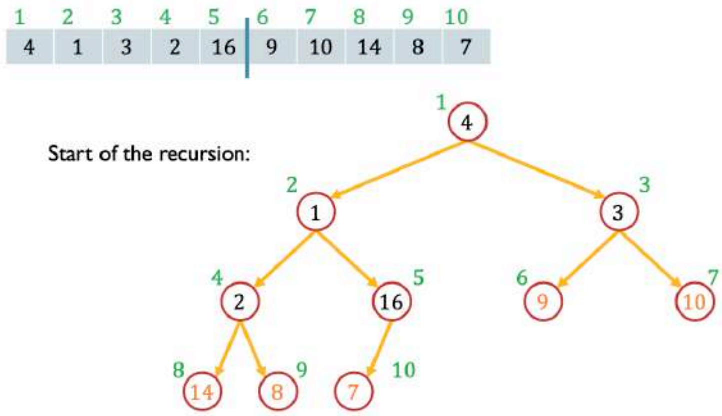
- עבור על הצמתים שנשארו מלמטה למעלה.
- תבצע $max-heapify$ על כל אחד מהם.

נרשום זאת כפסואדו קוד :

Algorithm 8 $Bulid-Max-Heap\left(A\right)$

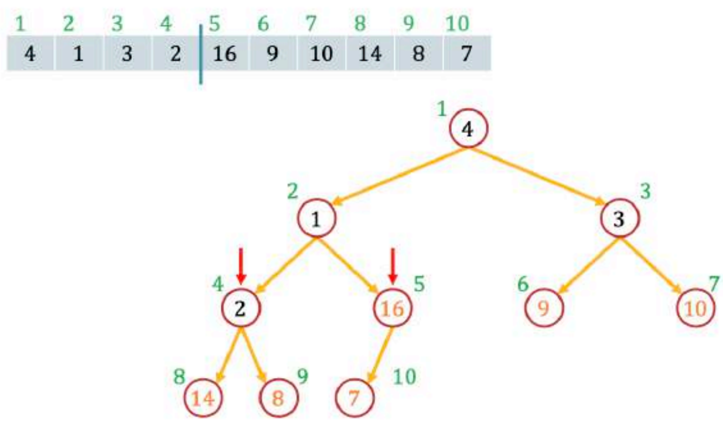
for $i\leftarrow\left\lfloor\frac{A.length}{2}\right\rfloor$ **down to** 1
  do $Max-Heapify\left(A,i\right)$

לפני שננתח את האלגוריתם נריץ אותו על הערמה הבאה :



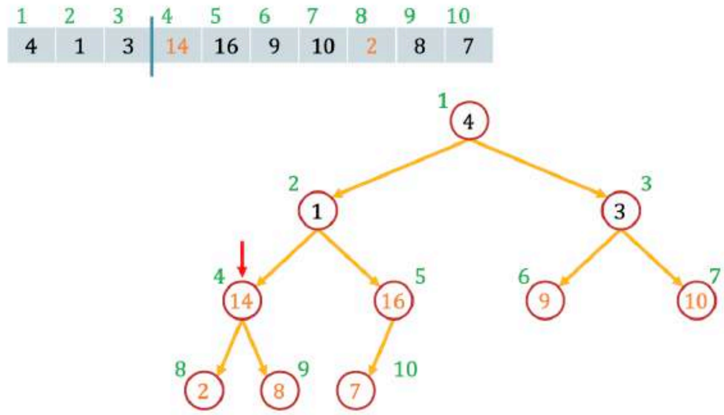
איור 56 : דוגמת הרצה לבניית ערמה

לאחר הרצה של מתודת העזר $Max - Heapify$ על 16 נקבל את הערמה :



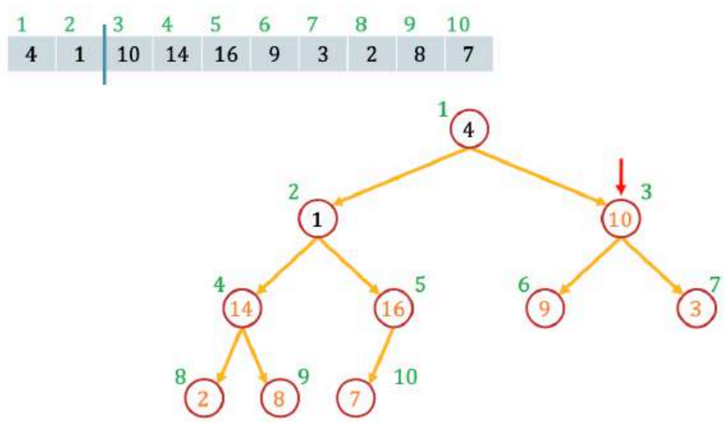
איור 57 : דוגמת הרצה לבניית ערמה

לא השתנה שום דבר. עתה נריץ על 2 ונקבל :



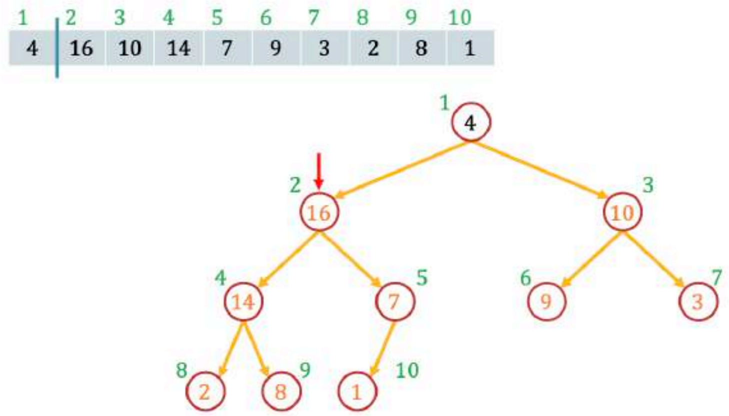
איור 58 : דוגמת הרצה לבניית ערמה

כאן 14 התחלף עם 2. עתה נריץ על 3 ונקבל:



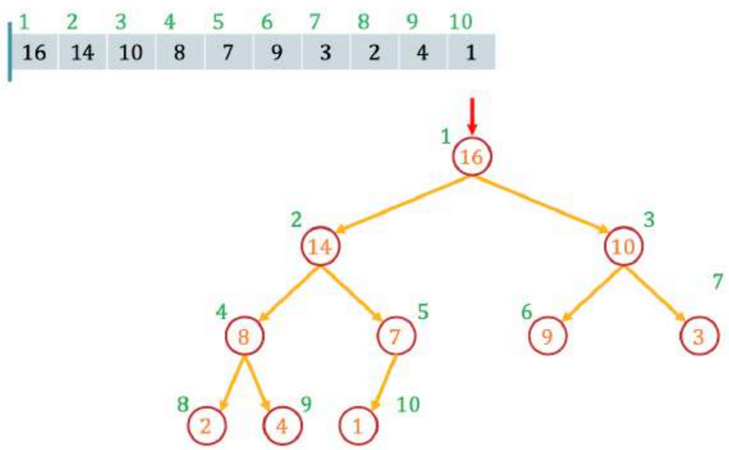
איור 59 : דוגמת הרצה לבניית ערמה

כאן 10 התחלף עם 3. נריץ על 1 ונקבל את הערמה:



איור 60: דוגמת הרצה לבניית ערמה

ולבסוף נריץ על 4 ונקבל את הערמה המקסימלית:



איור 61: דוגמת הרצה לבניית ערמה

לאחר שצברנו אינטואיציה, נרצה להוכיח את נכונות האלגוריתם.

הוכחה: כדי להוכיח את האלגוריתם נשתמש באנוריאנט. נשאלת השאלה מהו? נשים לב כי באיטרציה ה-i כל האיברים מימין ל-i כלומר $i+1,\dots,n$ הם ראשים של ערימה מקסימלית, זהו האינוריאנט שלנו. נוכיח באינדוקציה את האינוריאנט ולבסוף נסתכל על האיטרציה האחרונה.

אתחול: באיטרציה הראשונה $i=\left\lfloor \frac{n}{2} \right\rfloor$ וכל $i' > i$ הוא עלה ולכן הוא ראש של ערמה מקסימלית בגודל 1.

איטרציות: נניח כי האינוריאנט נכון עבור איטרציה ונוכיח עבור האיטרציה הבאה. נשים לב כי כל הילדים של $A[i]$ הם ראשים של ערמה מקסימלית ולאחר $Max-Heapify$ נקבל כי $A[i]$ יהיה גדול יותר מכל הילדים שלו. עתה, כל הילדים שלו עדיין ראשים של ערמה מקסימלית ולכן האינוריאנט נכון.

קביעה: הלולאה מסתיימת כאשר $i=1$ ואז $A[1]$ הוא שורש של ערמה מקסימלית. ■

12.5.1 סיבוכיות זמן ריצה

נשים לב כי מריצים את $Max-Heapify$ על $\frac{n}{2}$ צמתים וכל הפעלה שלו היא $\mathcal{O}(n)$. לכן הסיבוכיות היא לכל היותר $\mathcal{O}(n\log n)$.

יחד עם זאת, זהו לא חסם הדוק, שכן בפועל אנו מפעילים את $Max-Heapify$ על ערמות קטנות יותר מ-n. אם כך, נביט בטענה הבאה שתעזור לנו לנתח זמן ריצה זה.

טענה. לכל רמה $0\le h\le\lfloor\log_2 n\rfloor$ מתקיים כי מספר הקודקודים בעץ ברמה זו הוא לכל היותר $\left\lceil\frac{n}{2^{h+1}}\right\rceil$.

הוכחה: באינדוקציה. מושארת לקורא הנאמן כתרגיל. ■ אם כך, העבודה שנעשית ברמה ה-h היא $\mathcal{O}(h)\cdot\left\lceil\frac{n}{2^{h+1}}\right\rceil$. נסכום עבור כל הרמות ונקבל כי

$$T(n)\le\sum_{n=0}^{\lfloor\log n\rfloor}\left\lceil\frac{n}{2^{h+1}}\right\rceil\cdot\mathcal{O}(h)=\mathcal{O}\left(n\sum_{n=0}^{\lfloor\log n\rfloor}\frac{h}{2^{h+1}}\right)$$

נשתמש בכך שלכל $|x|<1$ מתקיים כי $\sum_{k=0}^{\infty}kx^k=\frac{x}{(1-x)^2}$ ולכן עבור $k=h$ ו-$x=\frac{1}{2}$ נקבל כי

$$\sum_{n=0}^{\lfloor\log n\rfloor}\frac{h}{2^{h+1}}\le\sum_{n=0}^{\infty}\frac{h}{2^{h+1}}=\frac{\frac{1}{2}}{\left(1-\frac{1}{2}\right)^2}=2$$

ולכן

$$T(n)=\mathcal{O}\left(n\sum_{n=0}^{\lfloor\log n\rfloor}\frac{h}{2^{h+1}}\right)=\mathcal{O}(2n)=\mathcal{O}(n)$$

כלומר הערמה נבנית בסיבוכיות $\mathcal{O}(n)$! שזה יותר טוב ממערך ממוין שנבנה ב-$\mathcal{O}(n\log n)$.

13 מיון ערימה $HeapSort$

נרצה להשתמש בערמה כדי למיין מערך $in-place$. כדי לעשות זאת נבצע את התהליך הבא:

- $.Extrat - max$
- נחליף בין האיבר המקסמלי לבין האיבר בסוף המערך.
- נבצע $.Max - Heapify\left(A,1\right)$
- נחזור על התהליך עם הערמה החדשה ללא האיבר האחרון.

נרשום פסידו קוד לאלגוריתם:

Algorithm 9 $Heap - Sort\left(A\right)$

```
Build-Max-Heap(A)
A.heapsize ← A.length
for i ← A.length downto 2
    do Exchange(A[1], A[i])
    do A.heapsize ← A.heapsize − 1
    do Max-heapify(A, i)
```

נחשב את סיבוכיות זמן הריצה של האלגוריתם.

- בניית הערמה ההתחלתית היא $.\mathcal{O}\left(n\right)$
- שימוש ב-$max - heapify$ נקרא n פעמים ובעל סיבוכיות $.\mathcal{O}\left(\log n\right)$

סך הכל $.n + n\log n = \mathcal{O}\left(n\log n\right)$ זהו מיון מבוסס השוואות ולכן $T\left(n\right) = \Omega\left(n\log n\right)$ ומכאן

$$T\left(n\right) = \Theta\left(n\log n\right)$$

סיבוכיות המקום היא $\mathcal{O}\left(n\right)$. זהו המיון הראשון שראינו ש-$T\left(n\right) = \Theta\left(n\log n\right)$ והוא ממיין $.in - place$. נעיר כי ניתן לממש את $.in - place\ MergeSort$

14 פעולות שימושיות בערמות

ניזכר כי רצינו לממש את הפעולות הבאות עבור תורי עדיפויות:

- $Heap-Max\left(S\right)$ - מחזיר את האיבר המקסימלי בתור.
- $Heap-Extract-Max\left(S\right)$ - מחזיר את האיבר המקסימלי ומוציא אותו מהתור.
- $Heap-Increase-Key\left(x,k,S\right)$ - מגדיל את x ב-k.
- $Heap-Insert\left(x,S\right)$ - מכניס את x ל-S.

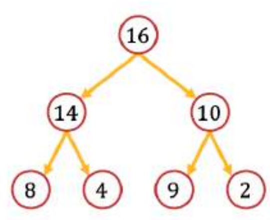
14.1 $Heap-Max$

נכתוב אותו באופן הבא:

Algorithm 10 $Heap-Max\left(A\right)$

1: **return** $A\left[1\right]$

נביט בדוגמא הבאה להמחשה:



16	14	10	8	4	9	2

איור 62: המחשה למתודה

סיבוכיות זמן הריצה של פעולה זו היא $\mathcal{O}\left(1\right)$.

14.2 $Heap-Extract-Max$

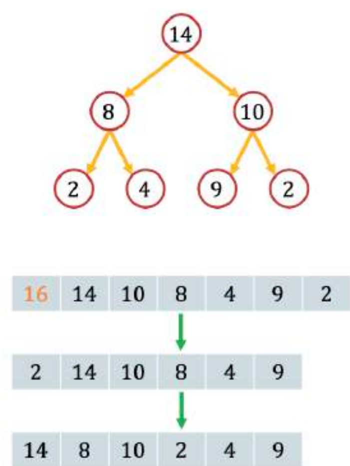
נבצע אותו באופן הבא:

Algorithm 11 $Heap-Extract-Max\,(A)$

if heapsize$(A) < 1$**:**
  then throw "heap underflow"
max$\leftarrow A\,[1]$
$A\,[1] \leftarrow A\,[heapsize\,(A)]$
$heapsize\,(A) \leftarrow heapsize\,(A) - 1$
Max-Heapify$(A, 1)$
return max

כלומר אנו שומרים את ערכו של המקסימלי ושמים בתא הראשון את הערך שבתא האחרון. ככה נוכל להפעיל את $max-heapify$ ולקבל שוב ערמה מקסימלית.

ניתן להביט באיור הבא להמחשה:



איור 63: דוגמא לריצת המתודה

מכיוון שמלבד $max-heapify$ כל הפעולות בעלות סיבוכיות קבועה, נקבל כי הסיבוכיות היא $\mathcal{O}\,(1)+T_{max-heapify}\,(n) = \mathcal{O}\,(1) + \mathcal{O}\,(\log n) = \mathcal{O}\,(\log n)$.

14.3 $heap-increase-key$

פעולה זו מזכירה את $max-heapify$ אך בכל זאת שונה. במקום לרשת מלמעלה למטה עד לעלים, נעלה מהקודקוד מעלה.

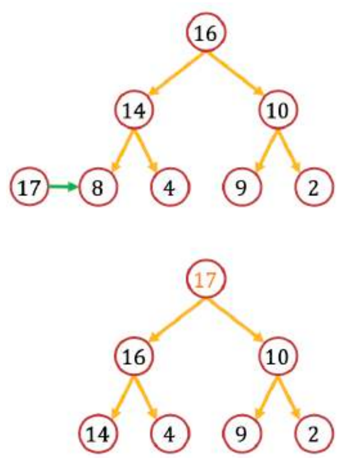
נרשום פסידו קוד באופן הבא:

Algorithm 12 $Heap - Increase - Key\left(A, i, key\right)$

1 : **if** $key < A\left[i\right]$
2 : **then throw "new key is smaller"**
3 : $A\left[i\right] \leftarrow key$
4 : **while** $i > 1$ **and** $A\left[parent\left(i\right)\right] < A\left[i\right]$
5 : **do** $Exchange\left(A\left[i\right], parent\left(A\left[i\right]\right)\right)$
6 : **do** $i \leftarrow parent\left(i\right)$

כאן עולים במעלה העץ ולכן במקרה הגרוע $T\left(n\right) = \mathcal{O}\left(\log n\right)$.

נביט בדוגמא הבאה להמחשה:



איור 64: דוגמא להרצת המתודה

14.4 $Heap - Insert$

הרעיון הוא כמו קודם, נוסיף איבר לסוף המערך. נשים בו ערך מינימלי ונוסיף לו את הערך הרצוי.

נרשום את הפסידו קוד הבא:

Algorithm 13 $Heap - Increase - Key\,(A, i, key)$

$heapsize\,(A) \leftarrow heapsize\,(A) + 1$

$A\,[heapsize\,(A)] \leftarrow -\infty$

$Heap - Incease - Key\,(A, heapsize\,(A)\,, key)$

הסיבוכיות היא של $Heap - Increase - Key$ ולכן $\mathcal{O}\,(n)$. נביט בדוגמת הרצה:

16	14	10	8	4	9	2	
16	14	10	8	4	9	2	$-\infty$
16	14	10	8	4	9	2	5

$\vdots$

איור 65: דוגמא להרצת המתודה

לסיכום,

- ערמה היא עץ בינארי שלם.
  - מציאת המינימום/מקסימום היא $\mathcal{O}\,(1)$.
  - כל פעולה דינאמית היא $\mathcal{O}\,(\log n)$.
- כל הפעולות משמרות את מבנה הערמה.
- בניית העץ נעשית במקום הקיים.
- מיון ערמה - $T\,(n) = \Theta\,(n \log n)$ והמקום הוא $\Theta\,(n)$.

חלק VII

הרצאה $\mathbb{VIII}$ - עצי חיפוש בינאריים

$(Binary\ Search\ Trees)$

15 מבוא

נניח כי ברשותנו סט מידע דינמי, כלומר כזה שעלול להשתנות עם הזמן, בו חשוב סדר האיברים (מפתחות). לדוגמה,

$$S=(2,3,4,5,6,9,13,15,17,1,20)$$

חשוב לנו לדעת אילו מספרים נמצאים בכל רגע במבנה הנתונים ומה הסדר היחסי בין האיברים.

המידע דינמי - אנו מוסיפים ומורידים איברים, לכן לא נרצה להשתמש במערך. זאת כי אם לדוגמה נרצה להכניס איבר לאמצע המערך עלינו להזיז את כל האיברים שמופיעים אחריו, ופעולה זו לוקחת $\mathcal{O}(n)$.

הסדר משנה - לכן לא נרצה להשתמש בטבלות גיבוב, שכן בהן אין סדר יחסי.

הערה שימושים ידועים של מבנה עם דרישות כאלה הוא עצי הופמן לדחיסת מידע ועצי ניתוב לניתוב במידע ברשת מחשבים

16 עצי חיפוש בינאריים

עץ חיפוש בינארי (BST) הוא מבנה נתונים בו המפתחות מסודרים. המבנה תומך ביעילות בפעולות $(\star)$:

(i) חיפוש איבר.

(ii) מציאת איבר מינימלי/מקסימלי.

(iii) מציאת איברים עוקבים.

(iv) הכנסת ומחיקת איברים.

האיברים נשמרים בתור עץ בינארי.

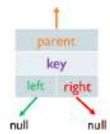
$(\star)$ כל הפעולות לוקחות $\Theta(h)$ כאשר h הוא גובה העץ. אם העץ מאוזן אז נקבל $\Theta(\log n)$.

הערה כל המגבלות שהיו לנו על ערימות אינן קיימות עבור עצים בינאריים. לדוגמה, בניגוד לערימה, יתכן שבעץ בינארי רמה פנימית לא תהיה מלאה.

לעץ חיפוש בינארי יש **שורש, צמתים פנימיים עם לכל היותר שני ילדים ועלים.**

לכל צומת x ישנם 4 שדות:

ההורה ($parent$), המפתח (key) , בן שמאלי ($left$) ובן ימני ($right$).



איור 66: צומת בעץ

אם לצומת אין ילד ימני/שמאלי או הורה אז אותו שדה יכיל בתוכו את הערך $null$. נשים לב שרק לשורש אין הורה.

מלבד השדות הללו ניתן לשמור גם מידע נוסף, אך 4 השדות הללו הם השדות הבסיסיים למימוש עץ חיפוש בינארי.

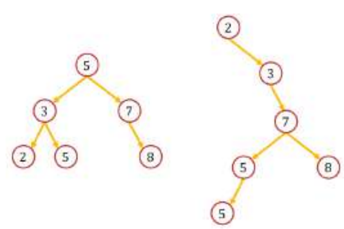
תכונת ה- BST יהי x צומת. אזי

(i) לכל צומת l בתת-עץ השמאלי של x מתקיים $l.key \leq x.key$.

(ii) לכל צומת r בתת-עץ הימני של x מתקיים $x.key \leq r.key$.

הערה שימו לב כי תכונה (i) מתקיימת לכל צומת בתת-עץ השמאלי של x, ולא רק לבן של x. כך גם עבור (ii).

הערה כל **תת-עץ** של BST הוא BST גם כן.



איור 67: איזון עצים

דוגמה עבור $S = (2, 3, 5, 57, 8)$ נוכל לבנות את העצים באיור הנ"ל. שימו לב כיצד העץ הימני יותר מאוזן (פחות גבוה ופרוש באחידות) מהשמאלי.

הערה מכאן אנו יכולים להבין שיש יותר מדרך אחת לבנות עץ בינארי. זאת בניגוד לערמה, בה המבנה היה יחיד.

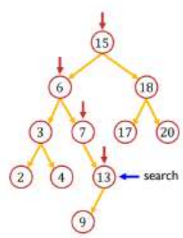
חיפוש בעץ בינארי

להלן אלגוריתם רקורסיבי לחיפוש איבר בעץ בינארי. אנו מנצלים את העובדה שכל האיברים מימין ל-x גדולים ממנו והאיברים משמאל קטנים ממנו, ואת העובדה שכל תת-עץ בעץ החיפוש הבינארי הוא עץ חיפוש בינארי גם כן.

Algorithm 14 $Tree-Search\,(x,k)$

Tree-Search(x,k)
if $x = null$ **or** $k = x.key$
    return x
if $k < x.key$
    return $Tree - Search\,(x.left, k)$
else
    return $Tree - Search\,(x.right, k)$

בדוגמה שבאיור למטה אנו רוצים למצוא את 13. היות ש-13 קטן מ-15 נפנה לבן השמאלי של 15. 13 גדול מ-6 לכן נפנה לבן הימני. 13 גדול מ-7 לכן נפנה לבן הימני וכך מצאנו את 13.



איור 68 : חיפוש איבר בעץ

סיבוכיות האלגוריתם היא $\mathcal{O}\,(h)$ כאשר h הוא הגובה שכן אנו מבצעים לכל היותר h "פניות" בעץ - עד העלה העמוק ביותר.

את האלגוריתם הנ"ל ניתן לממש איטרטיבית ללא רקורסיה באותה סיבוכיות :

Algorithm 15 $Tree - Search\,(x,k)$

Iterative-Tree-Search(x,k)

while $x \neq null$ **and** $k \neq x.key$

  if $k < x.key$

    $x \leftarrow x.left$

  else

    $x \leftarrow x.right$

return x

מציאת איבר מינימלי ומקסימלי

נשים לב כי מתכונת ה-BST האיבר השמאלי ביותר הוא המינימלית והימני ביותר הוא המקסימלי.

זאת כי אם לדוגמה x הוא האיבר המינימלי אז תמיד נבצע פניות שמאלה (אם ביצענו פנייה ימינה זה אומר שהוא גדול מאיבר אחר בסתירה למינימליות). כך נקבל את הפסידו-קודים הבאים:

אלגוריתם 16 $Tree - Minimum\,(x)$

Tree-Minimum(x)

while $x.left \neq null$

  $x \leftarrow x.left$

return x

אלגוריתם 17 $Tree - Maximum\,(x)$

Tree-Minimum(x)

while $x.right \neq null$

  $x \leftarrow x.right$

return x

הערה שימו לב! BST הוא לא קיצור ל-$BULLSHIT$.

מעבר על עץ

נרצה לעבור על כל איברי העץ. יש לנו מספר דרכים לעשות זאת.

הדרך הראשונה היא $Inorder - Tree - Walk$ - נדפיס את כל העץ משמאל לימין (כלומר בסדר עולה, ממוין):

Algorithm 18 $Inorder-Tree-Walk\left(x\right)$

1: **Inorder-Tree-Walk**(x)
2: **if** $x \neq null$
3: $Inorder-Tree-Walk\left(x.left\right)$
4: $print\left(x\right)$
5: $Inorder-Tree-Walk\left(x.right\right)$

ניתן לראות בקוד כיצד קודם אנו עושים קריאה רקורסיבית שמאלה, רק אז מדפיסים את האיבר ממנו יצאנו, ולבסוף עושים קריאה רקורסיבית ימינה. לכן באמת נקבל שהאיברים יודפסו בסדר ממוין.

סיבוכיות אינטואיטיבית מובן לנו שזמן הריצה הוא $\Theta\left(n\right)$ שכן אנו מבצעים עבודה קבועה לכל צומת. פורמלית ניתן לפתור את נוסחת הנסיגה $T\left(n\right)\leq T\left(k\right)+T\left(n-k-1\right)+d$ ולקבל את אותה התוצאה.

שתי דרכים נוספות לעבור על עץ הן $preorder$ בו קודם נדפיס את שורש העץ ורק אז נבצע קריאות רקורסיביות, ו-$postorder$ העושה את ההפך.

אלגוריתם 19 $Postorder-Tree-Walk\left(x\right)$

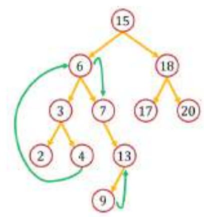
1: **Postorder-Tree-Walk**(x)
2: **if** $x \neq null$
3: $Postorder-Tree-Walk\left(x.right\right)$
4: $Postorder-Tree-Walk\left(x.left\right)$
5: $print\left(x\right)$

אלגוריתם 20 $Prerder-Tree-Walk\left(x\right)$

1: **Preorder-Tree-Walk**(x)
2: **if** $x \neq null$
3: $print\left(x\right)$
4: $Preorder-Tree-Walk\left(x.left\right)$
5: $Preorder-Tree-Walk\left(x.right\right)$

מציאת עוקב

לכל מפתח x נרצה למצוא את העוקב שלו: המפתח y הקטן ביותר עבורו $y.key\geq x.key$.



איור 69 : מציאת עוקב

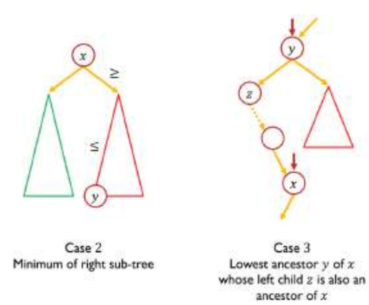
נשים לב כי בעץ חיפוש בינארי העוקב של מספר יכול להופיע במגוון מקומות : מעל האיבר (כמו עבור 9), מתחתיו (כמו עבור 6) או בכלל רחוק ממנו (כמו עבור 4).

יש לנו שלושה מקרים בהם נטפל :

1. $null$ - אם x הוא הצומת המקסימלית.
2. המינימום של התת-עץ הימני של x - אם ל-x יש ילד ימני. $(\star)$
3. האב הקדמון הנמוך ביותר של x שהבן השמאלי שלו הוא אב קדמון של x או x עצמו. (אם אין ל-x ילד ימני ואינו מקסימלי) $(\star\star)$

$(\star)$ המינימום יהיה האיבר השמאלי ביותר בתת-העץ הימני.

$(\star\star)$ בתיאור שקול, במקרה 3 אנחנו מחזירים את ההורה הראשון (הנמוך ביותר) שעבורו פנינו שמאלה כדי להגיע ל-x.
בתיאור שקול נוסף, אם נצא מ-x אנו נפנה שמאלה לכיוון מעלה ($\nwarrow$) כל עוד אי אפשר לפנות ימינה ($\nearrow$). ברגע שתהיה לנו פניה ימינה נחזיר את ההורה הקדום אליו הגענו.



איור 70 : תיאור מקרים $2, 3$ במציאת עוקב

מכאן נקבל את הפסידו-קוד הבא

Algorithm 21 $Tree - Successor\,(x)$

Tree-Successor(x)
if $x.right \neq null$
  return $Tree - Minimum\,(x.right)$ } **case** 2
$y \leftarrow x.parent$
while $y \neq null$ **and** $x = y.right$
  $x \leftarrow y$
  $y \leftarrow y.parent$ } **case**3 $(and\ 1)$
return y

הערה אם יש לנו כפילויות, לדוגמה אם 5 מופיע שלוש פעמים ו-6 נמצא בעץ, אז ל-2 עותקים של המספר 5 העוקב יהיה 5 אחר, ועבור ה-5 השלישי העוקב יהיה 6.

נוכיח את נכונות האלגוריתם עבור מקרה 3.

טענה אם ל-x אין בן ימני אז העוקב של x הוא האב הקדום הראשון של x שבנו השמאלי הוא גם אב קדום של x (או x עצמו). אם אין אב קדום כזה, x מקסימלי ולכן העוקב הוא $null$.

הוכחה כדי להראות כי $y = successor\,(x)$ (y הוא העוקב של x) נצטרך להראות כי $y.key \geq x.key$ וכן x הוא המקסימום מבין כל האיברים הקטנים מ-y.

באלגוריתם שלנו אנו מתחילים מ-x ומטפסים במעלה העץ לאורך בנים ימניים. נניח כי עצרנו בצומת z ונסמן $y = z.parent$.

נשים לב כי x הוא האיבר המקסימלי בתת-עץ ששורשו הוא z. זאת כי אם נצא מ-z ונלך לאורך האיברים הימניים כל הזמן, נגיע ל-x, שכן אנו בטוח נגיע ל-x כי פונים ימינה כל הזמן, אבל בגלל שאנו במקרה 3 ול-x אין בן ימני אז לא נוכל לפנות יותר, לכן x הוא המקסימום.

נראה ש-y הוא העוקב של x. מתקיים $y.key \geq x.key$ כי z הוא הבן השמאלי של y, לכן x נמצא בתת-עץ השמאלי של y. הראינו ש-x הוא המקסימום בתת-עץ ששורשו הוא z, לכן x הוא האיבר המקסימלי שקטן מ-y.

הכנסת איבר

מסתבר שתמיד ניתן להכניס איבר חדש בתור עלה, ואין צורך לשחק עם מבנה העץ ולהזיז איברים. בעקבות הפעולה הזו יתכן שהעץ לא יהיה מאוזן, אך לא נתייחס לבעיה זו לעת עתה.

Algorithm 22 $Tree - Insert\,(T, z)$

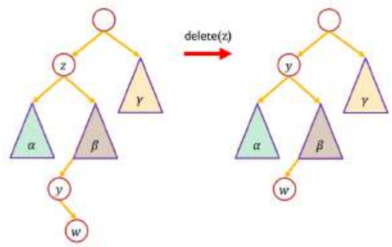
Tree-Insert(T, z)
$y \leftarrow null$
$x \leftarrow T.root$
while $x \neq null$
    $y \leftarrow x$
    if $z.key < x.key$
        $x \leftarrow x.left$
    else
        $x \leftarrow x.right$
$z.parent \leftarrow y$
if $y = null$ **then** $T.root \leftarrow z$
else if $z.key < y.key$
    $y.left \leftarrow z$
else
    $y.right \leftarrow z$

מחיקת איבר

מחיקת איבר z הינה פעולה מסובכת יותר מאשר הכנסה. נטפל בשלושה מקרים:

1. ל-z אין ילדים - נמחק את z, ונעדכן את ההורה שלו כך שבמקום המצביע ל-z יהיה $null$.
2. ל-z יש ילד אחד - נמחק את z, נחבר את ההורה של z עם הילד של z.
3. ל-z יש שני ילדים - יותר מסובך, לא ניתן סתם לבחור ילד כמו במקרה 2 - לשם כך נשתמש **בעוקב**.

מקרה 3



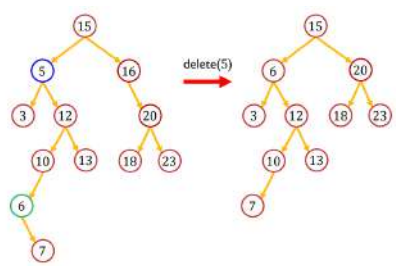
איור 71: מחיקת איבר - מקרה 3

עבור הדוגמה של מקרה 3 אנו נשתמש בעוקב של z, שנסמנו ב-y (נשים לב כי y הוא האיבר השמאלי ביותר בתת-עץ של הילד הימני של z).

נשים את y במקום z, ונקבל את העץ המופיע מימין. נשים לב כי התת-עץ α קטן מ-z, אבל z קטן מ-y לכן התת-עץ α קטן מ-y, כלומר צד שמאל תקין. ב-β כולם גדולים מ-z, אבל y הוא האיבר הקטן ביותר שעדיין גדול מ-z, לכן עדיין נקבל כי ב-β כולם גדולים מ-y, ולכן צד ימין תקין גם הוא.

הערה אם y הוא העוקב של z, ול-z יש שני ילדים, אזי אין ל-y שני ילדים (יש לכל היותר אחד)!

הסבר אם ל-y היו שני ילדים, אז יש לו בן ימני, לכן העוקב שלו הוא המינימום בתת-עץ של הבן הימני. המינימום הזה הוא הצומת השמאלי ביותר בתת-עץ של הבן הימני, לכן לאותו צומת אין בן שמאלי (אחרת הוא לא היה השמאלי ביותר).



איור 72: לדוגמה למחיקת איבר - מקרה 3

השתלה $(Transplant)$

הפסידו-קוד של המקרה השלישי הינו מעט מסובך. לכן נשתמש בפונק' עזר בשם $Transplant$.

פונק' זו לוקחת עץ ומחליפה אותו בעץ אחר: אם יש לנו קודקוד u שמושרש בו עץ, נוכל להחליף את u והעץ ב-v שמושרש בו עץ אחר.

Algorithm 23 $Transplant\ (T, u, v)$

Transplant(T, u, v)
if $u.parent = null$
    $T.root \leftarrow v$
else if $u = u.parent.left$
    $u.parent.left \leftarrow v$
else
    $u.parent.right \leftarrow v$
if $v \neq null$
    $v.parent \leftarrow u.parent$

קל לראות שאם ל-u אין הורה אז (כלומר הוא השורש) אז פשוט מחליפים את השורש של T להיות v. אם u בן שמאלי/ימני שמים את v בתור בן שמאלי/ימני בהתאמה.

עתה נביט בפסידו-קוד למחיקת איבר מעץ:

Algorithm 24 $Tree-Delete\left(T,z\right)$

```
Tree-Delete(T, z)
if z.left = null
    Transplant(T, z, z.right)
else if z.right = null
    Transplant(T, z, z.left)
else
    y ← Tree − Minimum(z.right)
    if y.parent ≠ z
        Transplant(T, y, y.right)
         y.right ← z.right
         y.right.parent ← y
         Transplant(T, z, y)
         y.left ← z.left
         y.left.right ← y
```

הערה קוד סימטרי ניתן לכתוב שמשתמש באיבר הקודם ולא באיבר העוקב של z.

סיבוכיות במקרים $1,2$ אנו רק מחליפים מצביעים : $\Theta\left(1\right)$. במקרה 3 אנו משתמשים ב-$Tree-Minimum$, שעולה לנו $\Theta\left(h\right)$.

לסיכום, כל הפונק' $Successor, Predecessor, Insert, Search$ עולות לנו $\Theta\left(h\right)$.

במקרה הלא-מאוזן (הגרוע) אנו עלולים לקבל $h=n$ ואז הסיבוכיות תהא $\Theta\left(n\right)$.

עצי חיפוש בינאריים בנויים רנדומלית

בהינתן n מפתחות נוכל להכניס בסדר **אקראי** לתוך BST (שבמקור היה ריק). הכוונה באקראי היא שכל פרמוטציה של המפתחות תופיע בהסתברות שווה. לעץ כזה נקרא BST בנוי רנדומלית.

משפט הגובה הממוצע של BST בנוי רנדומלית הוא $\Theta\left(\log n\right)$.

הוכחה הושארה כתרגיל לסטודנט המשקיע החכם היפה והחסון. (ההוכחה מופיעה גם בספר הקורס)

עצי AVL

ראינו קודם שלאותה קבוצת מפתחות יכולים להיות כמה BST-ים. ראינו כי לפעמים אותם עצים יוצאים מאוד לא מאוזנים. נרצה לשנות את פעולות ה-$insert$ ו-$delete$ כך שהעצים יהיו תמיד מאוזנים.

יש מספר דרכים להגדיר מהו עץ "מאוזן". מסתבר שלשמור על העץ מאוזן לחלוטין זה מאוד יקר (ולא נחוץ) לכן נרצה לשמור על הפרשי הגבהים בין התת-עץ הימני לשמאלי קטנים ככל האפשר.

את המטרה שלנו ניתן להשיג בכמה דרכים, ביניהן:

- עצי AVL.
- עצים אדומים שחורים.
- עצי $2-3$.
- עצי B.

אנו נתמקד בעצי AVL.

שיטה נאזן את העץ בכל פעם שנראה שהוא יוצא מאיזון. את פעולת התיקון ניתן להשיג ב-$\Theta\left(\log n\right)$.

הגדרה עץ חיפוש בינארי T יקרא עץ AVL אם מתקיימות התכונות הבאות:

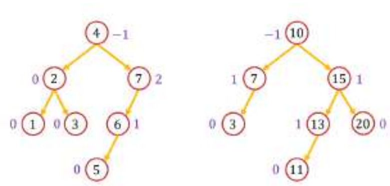
(i) T לא ריק.

(ii) התתי-עצים הימני והשמאלי של T הם גם עצי AVL.

(iii) הפרשי הגבהים בין התתי-עצים של הבן הימני והשמאלי הוא לכל היותר 1. כלומר $|left\ height - right\ height| \leq 1$.

דרישה (iii), כך מסתבר, מבטיחה לנו כי $h = \lfloor \log_2 n \rfloor$. כך נקבל שהפעולות שהגדרנו יעבדו ב-$\Theta\left(\log n\right)$.

הערה אין דרישה בעצי AVL שהעץ יהיה מלא.



איור 73: עץ AVL ועץ שאינו AVL

העץ השמאלי באיור אינו עץ AVL שכן תכונה (iii) אינה מתקיימת עבור הצומת 7 : הגובה של התת-עץ השמאלי הוא -1 $(\star)$ והגובה של התת-עץ הימני הוא 1, לכן ההפרש הוא 2 וזה גדול מ-1.

העץ בימני באיור הוא אכן AVL, בכל מקום הפרשי הגבהים הם לכל היותר 1.

$(\star)$ הגובה של עץ ריק מוגדר להיות -1.

הגדרה לכל צומת נגדיר את הפרש הגובה $hd(x)$ להיות $hd(x) = \begin{cases} 0 & \text{עלה } x \\ left\ height - right\ height & \text{אחרת} \end{cases}$ כאשר הגובה של עץ ריק הוא -1.

דוגמה באיור הקודם הפרשי הגבהים מופיעים בסגול.

משפט אם T עץ AVL אזי הגובה שלו הוא $\Theta(\log n)$, ולמעשה הגובה המדויק הוא לכל היותר $\log_{\varphi}\left(\sqrt{5}(n+2)\right) - 2 \approx 1.44 \cdot \log_2(n+2) - 0.328$, כאשר φ הוא יחס הזהב.

הוכחה הושארה כתרגיל לסטודנט המשקיע החכם היפה והחסון.

הכנסה ומחיקה

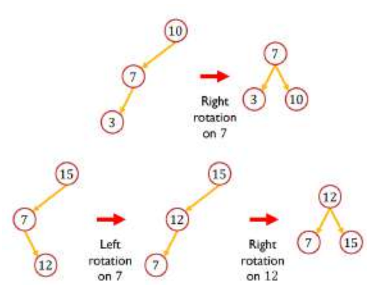
נבצע הכנסה ומחיקה של איברים כפי שביצענו ב-BST, ואז בגלל שהפעולות האלה עלולות לפגוע בהפרשי הגבהים, בתום פעולת ההכנסה/מחיקה נבצע תיקון ונאזן את העץ.

שאלה על כמה תתי-עצים אנו משפיעים בכל פעולה?

תשובה בהכנסה - אנו משפיעים על כל הצמתים מהעלה עד השורש, $\mathcal{O}(h)$. לכן אנו נעלה מהעלה לשורש ונחפש איפה התקלקלה תכונת ה-AVL ונתקן.

במחיקה - עבור מקרים $1, 2$ אנו משפיעים על כל העלים מהצומת שמחקנו עד השורש, $\mathcal{O}(h)$. במקרה 3 אנו משפיעים בנוס על התת-עץ הימני והשמאלי.

בעת הכנסה אנו משנים את הפרשי הגבהים בלכל היותר 1. נחשב את הפרשי הגבהים של הצמתים שהושפעו : $\mathcal{O}(h)$. $|hd(x)| \leq 1$ אז סיימנו. אחרת, נאזן את העץ באמצעות פעולת סיבוב ($rotation$).



איור 74 : סיבוב עצי AVL

דוגמה באיור הנ"ל בעץ העליון, לאחר שהכנסנו את האיבר 3, העץ יצא מאיזון. אנו מפעילים על העץ מעין פעולת סיבוב כך ש-10 יורד למטה, ו-$7, 3$ עלו למעלה. זהו סיבוב לימין על 7.

בעץ התחתון אנו נפעיל סיבוב לשמאל על 7, ולאחר מכן סיבוב לימין על 12 וכך נקבל שהעץ מאוזן.

סיבוב מחליף את התפקידים של האב והילד, תוך שימור סדר ה-BST. כתוצאה מהפעולה תכונת ה-AVL משתקמת.

הכנסה

אם אנחנו רוצים להכניס איבר z לעץ ששורשו x, יתקיים אחד מהבאים:

1. z הוכנס לתת-עץ השמאלי של הילד השמאלי של x.
2. z הוכנס לתת-עץ הימני של הילד השמאלי של x.
3. z הוכנס לתת-עץ השמאלי של הילד הימני של x.
4. z הוכנס לתת-עץ הימני של הילד הימני של x.

עבור מקרים $1, 4$ (הם סימטריים) מספיק סיבוב אחד. עבור מקרים $2, 3$ יתכן שנצטרך שני סיבובים.

(א) מקרים $1, 4$

(ב) מקרים $2, 3$

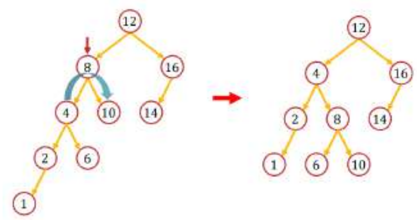
איור 75: סיבובים שונים בעת הכנסה לעץ AVL

מקרה 1 דרוש סיבוב לימין.

מקרה 4 דרוש סיבוב לשמאל.

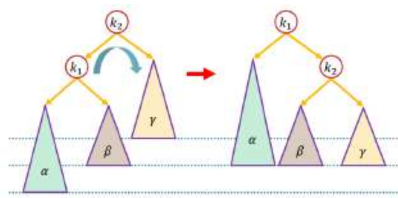
מקרה 2 דרוש סיבוב לשמאל ואז סיבוב לימין.

מקרה 3 דרוש סיבוב לימין ואז סיבוב לשמאל.



איור 76 : דוגמה להכנסה

דוגמה באיור הנ"ל לאחר הכנסת האיבר 1, אנו נעבור על כל הצמתים שעוברים במסלול מהעלה 1 עד השורש. נשים לב שב-8 תכונת ה-AVL נפגעה, הואיל ומדובר על מקרה 1 נבצע סיבוב לימין וכך תישמר תכונת ה-AVL.



איור 77 : תיאור כללי של מקרה 1

נציג אינטואיציה עבור מקרה 1 (אך לא נוכיח את כל ארבעת המקרים).

באיור הנ"ל יש לנו עץ אליו הכנסנו איבר שנכנס לתת-עץ α. עתה יש הפרש בגודל 2 בין α ל-γ. לכן תכונת ה-AVL נפגעה ב-k_2. לאחר הפעלת הסיבוב לימין ניתן לראות שהפרשי הגבהים קטנו, ועתה העץ מאוזן.
העץ נשאר BST שכן α, β, γ בפנים לא השתנו. α היה ילד שמאלי של k_1 וכך נשאר, וכך גם עבור γ ו-k_2, לכן גם כאן אין בעיה. β היה בתת-עץ השמאלי של k_2 והוא עדיין בתת-עץ השמאלי של k_2, לכן גם כאן הכל תקין. על כן, העץ נשאר BST.

הערה לאחר הכנסה וסיבוב יחיד, הגובה של העץ החדש הוא אותו גובה כמו של העץ לפני ההכנסה.
רוטציה לוקחת $\Theta\left(1\right)$, אנו עוברים על כל המסלול מעלה לשורש לכן סה"כ לוקח לנו $\Theta\left(\log n\right)$.

מחיקה

מקרה זה יותר מסובך, ויתכן שנזקק ל-$\Theta\left(\log n\right)$ רוטציות במסלול מעלה לשורש (שוב, יש לנו כ-$\log n$ צמתים במסלול ורוטציה לוקחת $\Theta\left(1\right)$).

חלק VIII

הרצאה $\mathbb{VIII}$ - גרפים

17 מושגים בסיסיים

הרבה פעמים יהיה נח לייצג את הנתונים שלנו בתור גרף - אוסף קודקודים שמחוברים ביניהם (לא כולם בהכרח מחוברים לכולם) באמצעות צלעות. אנו נראה שלמעשה גרף הוא הכללה של עץ.

למשל, רשת חברתית - הקודקודים הם המשתמשים והקשתות מחברות בין חברים. דוגמא נוספת היא המדינה, הקודקודים בה הם הערים והצלעות הן הדרכים המחברות ביניהן.

קיימים כמה יחסים בינאריים בין קודקודים בגרף:

- סימטרי - אם a מחובר ל-b אז b מחובר ל-a. לדוגמה יחס חברות (אם אני חברך אתה חברי).
- א-סמטרי - אם a מחובר ל-b אז b לא בהכרח מחובר ל-a. לדוגמה אם לינק אחד מכיל קישור ללינק אחר, הלינק האחר לא חייב לכלול קישור ללינק הראשון.

כאשר היחס א-סימטרי נשתמש בגרף מכוון.

17.1 גרפים מכוונים, לא מכוונים ותכונותיהם

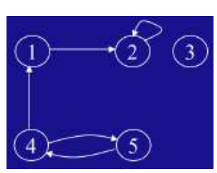
הגדרה. הגדרה גרף מכוון $G=(V,E)$ הוא זוג בו $V=\{v_1,\ldots,v_n\}$ היא קבוצת הקודקודים ו-$E=\{e_1,\ldots,e_m\}$ היא קבוצת הצלעות.

צלע $e_k=e_{ij}=(v_i,v_j)$ היא זוג סדור המייצג את החיבור בין הקודקוד v_i ל-v_j.

הגדרה. הגדרה גרף לא מכוון $G=(V,E)$ הוא זוג בו $V=\{v_1,\ldots,v_n\}$ היא קבוצת הקודקודים ו-$E=\{e_1,\ldots,e_m\}$ היא קבוצת הצלעות.

צלע $e_k=e_{ij}=\{v_i,v_j\}$ היא קבוצה המייצגת את החיבור בין הקודקוד v_i ל-v_j ולהפך.

דוגמה. $E=\{(4,5),(5,4),(4,1),(1,2),(2,2)\}$, $V=[5]$. הגרף $G=(V,E)$ יראה כבאיור הנ"ל.



איור 78: דוגמה לגרף מכוון

הערה בגרף לא מכוון לא נסכים ש-(v_i,v_i) יהיה צלע, כלומר לא יהיו צלעות מקודקוד לעצמו.

הגדרה. (בגרף לא מכוון) הדרגה של קודקוד v היא מספר הצלעות שמחוברות אליו. נסמן הדרגה $d\left(v\right)$, כלומר $d\left(v\right)=\left|\left\{e\in E\mid v\in e\right\}\right|$.

משפט. (לחיצות הידיים) $\sum\limits_{v\in V}d\left(v\right)=2\left|E\right|$.

מסקנה. $\left|E\right|\leq\left|V\right|\left(\left|V\right|-1\right)=\mathcal{O}\left(\left|V\right|^{2}\right)$.

מסקנה זו נכונה כי במשפט לחיצות הידיים אנו סוכמים על כמות הקודקודים, ויש בדיוק $\left|V\right|$ כאלה. הדרגה של כל קודקוד היא לכל היותר $\left|V\right|-1$ ומכאן הדרוש.

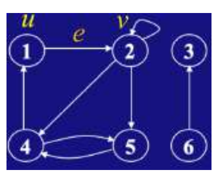
הגדרה. (בגרף מכוון) דרגת הכניסה של $v\in V$ היא מספר הצלעות שנכנסות לקודקוד. נסמן דרגת הכניסה $d_{in}\left(v\right)=\left|\left\{e\in E\mid\exists u\in V:e=\left(u,v\right)\right\}\right|$.

הגדרה. הגדרה (בגרף מכוון) דרגת היציאה של $v\in V$ היא מספר הצלעות שיוצאות מהקודקוד. נסמן דרגת היציאה $d_{out}\left(v\right)=\left|\left\{e\in E\mid\exists u\in V:e=\left(v,u\right)\right\}\right|$.

הערה נשים לב כי $\sum\limits_{v\in V}d_{out}\left(v\right)=\sum\limits_{v\in V}d_{in}\left(v\right)$ שכן כל צלע שיוצאת מקודקוד כלשהו חייבת להכנס לקודקוד כלשהו ולהפך.

דוגמה בגרף הנ"ל, $d_{in}\left(2\right)=2,\ d_{out}\left(3\right)=0,\ d_{in}\left(3\right)=1$ (לא נרשום לכל הקודקודים).

אכן גם $\sum\limits_{v\in V}d_{out}\left(v\right)=\sum\limits_{v\in V}d_{in}\left(v\right)=8$.



איור 79: דוגמה לגרף מכוון

הגדרה מסילה מקודקוד u ל-v בגרף $G=\left(V,E\right)$ היא סדרה $\left\langle v_0,\ldots,v_k\right\rangle$ כאשר $v_0=u,\ v_k=v$ ו-$\forall i\in\left[k\right],\ \left(v_{i-1},v_i\right)\in E$. אורך המסילה הוא k.

המרחק בין u ל-v, $\delta\left(u,v\right)$, הוא אורך המסילה הקצרה ביותר מ-u ל-v.

הערה לא חייבת להיות מסילה יחידה מ-u ל-v (ולא חייבת להיות מסילה כזו בכלל).

הגדרה מעגל הוא מסלול מקודקוד לעצמו בגודל גדול או שווה ל-1.

הגדרה גרף לא מכוון $G=(V,E)$ יקרא קשיר אם קיימת מסילה בין כל שני קודקודים ב-V.

הגדרה גרף מכוון $G=(V,E)$ יקרא קשיר-חזק אם $\forall v,u\in V$ קיימת מסילה מ-v ל-u ומ-u ל-v.

הגדרה יהי $G=(V,E)$ גרף. גרף $G'=(V',E')$ יקרא תת-גרף של G אם $V'\subseteq V,\ E'\subseteq E$.

הגדרה יהי $G=(V,E)$ גרף. רכיבי הקשירות $G_1,G_2,\ldots$ של G הם התתי-גרפים הקשירים-חזק המקסימליים של G.

17.2 עצים ותכונותיהם

הגדרה עץ הוא גרף קשיר ללא מעגלים.

טענה לעץ יש בדיוק $|E|=|V|-1$ צלעות. בנוסף, לגרף לא מכוון G התנאים הבאים שקולים:

(i) G עץ.

(ii) G חסר מעגלים מקסימלי. קרי, G חסר מעגלים ואם נוסיף צלע אז יווצר מעגל ב-G.

(iii) G קשיר מינימלי. קרי, G קשיר ואם נמחק צלע אז G לא יהיה קשיר.

(iv) G חסר מעגלים וקשיר.

הגדרה גרף משוקלל הוא גרף בו לכל צלע מתאימה משקולת $w(v_i,v_j)>0$.

ניתן לחשוב על זוג קודקודים שאינם מחוברים בצלע כבעלי משקולת ∞.

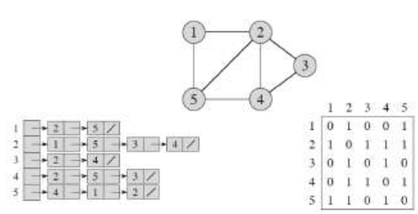
הגדרה משקל של מסלול $\langle v_0,\ldots,v_k\rangle$ הוא סכום המשקולות על הצלעות: $\sum_{i=1}^{k} w(v_{i-1},v_i)$.

18 יצוג גרפים

יש שתי דרכים סטנדרטיות ליצוג גרפים:

1. רשימת סמיכויות - לכל קודקוד v נשמור רשימה מקושרת L_v המכילה את שכניו. סיבוכיות המקום של יצוג זה היא $\Theta(|V|+|E|)$.

2. מטריצת סמיכויות - מטריצה A בגודל $|V|\times|V|$ בה יש $A_{ij}=1$ אם קיימת צלע v_i,v_j ו-$A_{ij}=0$ אחרת. סיבוכיות המקום של יצוג זה היא $\Theta\left(|V|^2\right)$.



איור 80: דוגמה ליצוג גרף

הערה במטריצת סמיכויות, כאשר הגרף מכוון, אם $A_{ij}=1$ לא בהכרח $A_{ji}=1$, כלומר לא בהכרח $A^t=A$.

הערה אם הגרף שלנו דליל, כלומר כמות הצלעות בו קטנה, נעדיף להשתמש ברשימת סמיכויות. מטריצת סמיכויות יעילה כאשר יש הרבה צלעות. נבחין כי במקרה בו $|E|=\Theta\left(|V|^2\right)$ להשתמש במטריצת סמיכויות עדיף גם מבחינת זכרון על רשימת סמיכויות.

מסקנה כל עוד $|E|$ מספיק קטן ביחס ל-$|V|^2$, נעדיף להשתמש ברשימת סמיכויות.

הערה במקרה בו הגרף הוא גרף ממושקל, מטריצת הסמיכויות לא תכיל 1 אם הקודקוד ה-i שכן של הקודקוד ה-j, אלא את $w\left(v_i,v_j\right)$ ואז נקבל מטריצה שמייצגת גרף ממושקל.

19 בעית המסילה הקצרה ביותר

דוגמה כאשר רכב רוצה להגיע מנק' אחת לאחרת, נרצה לספק לו את המסלול הקצר ביותר מהקודקוד בו הוא נמצא לקודקוד היעד. הצלעות ככל הנראה יהיו הכבישים, ומשקל כל צלע יכול להיות תלוי באורך הכביש, בעומס התנועה בו וכו'.

בעית המסלול הקצר ביותר מתחלקת לכמה בעיות:

1. מסלול יחיד - נתונים קודקודים s,t ועלינו למצוא את המסלול הקצר ביותר מ-s ל-t ואת אורכו.
2. מקור יחיד - נתון קודקוד s ועלינו למצוא את המסלול הקצר ביותר מ-s לכל קודקוד אחר.
3. כל הצמדים - עלינו למצוא את המסלול הקצר ביותר בין כל שני קודקודים.

נתחיל עם בעיה 2, שכן בסיסה הוא המקור לפתרון הבעיות האחרות, והגרף שלנו יהיה לא ממושקל, לכן אורך המסלול הוא מספר הקודקודים במסלול.

19.1 אלגוריתם חיפוש לרוחב $BFS-Breadth\ First\ Search$

ברעיון ה-BFS, אנו נתחיל מקודקוד s ונסרוק את הגרף כך שהקודקודים שאנו סורקים מתרחקים מ-s. האלגוריתם יעשה את הדבר הבא:

- נתחיל מקודקוד s ונסמן את הרמה שלו ב-0.
- נסרוק את כל הקודקודים שיש מ-s צלע אליהם. נסמן את רמתם ב-1.

- באופן כללי, נסמן את רמת השכנים, שלא סרקנו, של הקודקודים ברמה ה-i ב-$i+1$. נבהיר שנית שאם קודקוד כבר סומן, לא נסמן אותו שוב.

נפריד בין כמה סוגים קודקודים. קודקודים:

1. שביקרנו בהם - קודקוד שביקרנו בו וסרקנו את כל שכניו.

2. נוכחיים - קודקוד שאנו סורקים כרגע אך עוד לא סרקנו את כל השכנים שלו.

3. שלא ביקרנו בהם - הקודקוד עוד לא נסרק.

האלגוריתם ישמור את הקודקודים הנוכחיים בתוך תור.

לכל קודקוד u נשמור 3 שדות נוספים:

$\underline{u.label}$ שישמור את סוג הקודקוד. $\underline{u.dist}$ שישמור את המרחק מ-s ל-u. $\underline{u.\pi}$ שהוא הקודם של u בעץ.

הקודם, $u.\pi$, יאפשר לנו למצוא את המסלול מ-u ל-s.

Algorithm 25 $BFS\,(G,s)$

$\textbf{BSF}(G,s)$

$s.label \leftarrow current; \quad s.dist \leftarrow 0; \quad s.\pi \leftarrow null$

$\textbf{for } u \textbf{ in } V \setminus \{s\}$

$\qquad u.label \leftarrow not\_visited; \quad u.dist \leftarrow \infty; \quad u.\pi \leftarrow null$

$EnQueue\,(Q,s)$

$\textbf{while } Q \textbf{ is not empty}$

$\qquad u \leftarrow DeQueue\,(Q)$

$\qquad \textbf{for each } v \textbf{ that is a neighbor of } u \textbf{ do}$

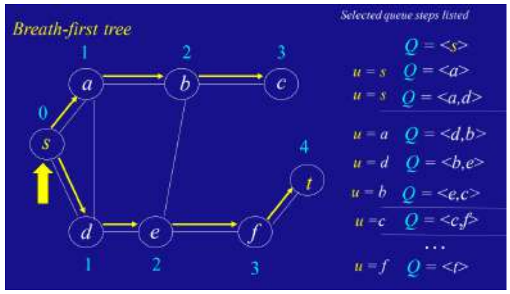
$\qquad\qquad \textbf{if } v.label = not\_visited$

$\qquad\qquad\qquad v.label \leftarrow current; \quad v.dist \leftarrow u.dist+1; \quad v.\pi \leftarrow u$

$\qquad\qquad\qquad EnQueue\,(Q,v)$

$\qquad u.label \leftarrow visited$

להלן דוגמה לריצת האלגוריתם



איור 81: דוגמה לריצת BFS

ננתח את סיבוכיות האלגוריתם ונוכיח את נכונותו.

נשים לב שהאלגוריתם מסיר כל קודקוד מהתור בדיוק פעם אחת.

באתחול אנו עוברים על כל הקודקודים לכן נקבל גם $\mathcal{O}(|V|)$.

לכל קודקוד, אנו עוברים על כל שכניו ומבצעים $\mathcal{O}(1)$ פעולות. לכן מספר הפעולות הוא קבוע ($\mathcal{O}(1)$) כפול מספר השכנים של כל קודקוד: $\sum_{v\in V} d(v) = 2|E|$.

כלומר סה"כ הסיבוכיות היא $\mathcal{O}(|V|)+\mathcal{O}(|E|)=\mathcal{O}(|V|+|E|)$ שזה לכל היותר $\mathcal{O}(|V^2|)$.

נשים לב ש-$\mathcal{O}(|V|+|E|)$ הוא ביטוי לינארי, והחסם העליון $\mathcal{O}(|V^2|)$ הוא ריבועי.

משפט. (נכונות BFS) בסיום ריצת האלגוריתם, $\forall v\in V$ מתקיים:

(i) $v.dist = d(s,v)$.

(ii) אם $d(s,v)<\infty$ אזי קיים מסלול קצר ביותר מ-s ל-v שהקודקוד לפני האחרון שלו הוא $v.\pi$.

למה 1 $\forall (u,v)\in E,\ d(s,v)\le d(s,u)+1$.

הוכחה: נסמן $k=d(s,u)$. לכן קיימת מסילה $\langle s=v_0,\dots,v_k=u\rangle$ מ-s ל-u. מכאן $\langle v_0,\dots,v_k,v\rangle$ היא מסילה באורך $k+1$ מ-s ל-v. לכן $d(s,v)\le k+1=d(s,u)+1$. ■

למה 2 בסיום ריצת האלגוריתם מתקיים $\forall v\in V,\ v.dist\ge d(s,v)$.

הוכחה: באינדוקציה על מספר עדכוני השדה $dist$.

<u>בסיס</u>: לאחר האתחול מתקיים עבור $v=s$, $v.dist=0=d(s,v)=d(s,s)$.

עבור $v\neq s$ מתקיים $v.dist=\infty\ge d(s,v)$.

<u>צעד</u>: נניח שמעדכנים את $v\in V$. מה"א האי-שוויון עדיין מתקיים ליתר הקודקודים. נראה שהוא עדיין מתקיים עבור v.

לאחר העדכון מתקיים $v.dist=u.dist+1\overset{\text{ה"א}}{\ge} d(s,u)+1\overset{\text{למה 1}}{\ge} d(s,v)$. ■

למה 3 נניח כי במהלך ריצת האלגוריתם $Q = \langle v_1, \ldots, v_r \rangle$ (v_1 הוא ראש התור). אזי:

(i) $\forall i \in [r-1]\,,\ v_i.dist \leq v_{i+1}.dist$.

(ii) $v_r.dist \leq v_1.dist + 1$.

הוכחה באינדוקציה (לא נוכיח כאן)

הערה לעיתים מסמנים את המרחק הקצר ביותר בין u ל-v ב-$\delta(s,v)$.

הוכחת המשפט נוכיח באינדוקציה על $\delta(s,v)$.

<u>בסיס</u>: $\delta(s,v) = 0$. נובע כי $v = s$, לכן הטענה ברורה.

<u>צעד</u>: יהי s קודקוד המקור ו-v קודקוד כלשהו.

יהי M אחד המסלולים הקצרים ביותר מ-s ל-v (קיים מסלול קצר ביותר כי $|M| = \delta(s,v) < \infty$ אך הוא לא בהכרח יחיד). נסתכל על הקודקוד שמגיע לפני v ונסמנו u. מרחקו מ-s הוא $|M| - 1$. מה"א מתקיים $u.dist = |M| - 1$.

נביט על ריצת האלגוריתם כאשר u מוצא מהתור Q. נחלק למקרים:

(i) <u>אם ביקרנו את v</u>: מלמה (3) מתקיים $v.dist \leq u.dist + 1 = \delta(s,v)$ אבל לפי למה (2) $\delta(s,v) \leq v.dist$ כלומר סה"כ $\delta(s,v) = v.dist$ כנדרש.

(ii) <u>אם לא ביקרנו את v</u>: נעדכן לפי הקוד $v.dist = u.dist + 1 (= \delta(s,v))$. כלומר $v.dist = \delta(s,v)$ כנדרש.

החלק השני של המשפט נובע בקלות (ii) הקטע $[u.d, u.f]$ **מוכל** ב-$[v.d, v.f]$ ו-u הוא צאצא של v ב-G_π.מהחלק הראשון.

חלק IX

הרצאה $\mathbb{IX}$ - אלגוריתם $DFS - Depth\ First\ Search$, מיון טופולוגי ומציאת רכיבי קשירות חזקים

בעוד שאלגוריתם BFS בא לפתור את בעיית המסלול הקצר ביותר, DFS לא בא לפתור אף בעיה טבעית, אך נראה שבכל זאת נמצאו לו שימושים כגון:

- מיון טופולגי.
- מציאת התת-עץ הפורש קטן ביותר.
- מציאת רכיבי קשירות חזקים.

20 DFS

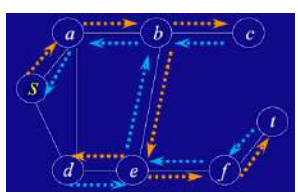
20.1 מוטיבציה

אנו נקבל גרף חסר מעלים שכל קודקוד בו מייצג משימה, הצלעות יסמנו עדיפות של המשימה כלומר $a \to b$ משמעו שצריך לבצע את a לפני b. נרצה לסדר את המשימות כך שלא נפר את הסדר. עם DFS אפשר לעשות זאת זמן לינארי.

20.2 פעולות האלגוריתם

אז מה בדיוק DFS עושה? בפשטות, הוא סורק את הגרף לעומק ופועל באופן הבא:

התחל מקודקוד s והתקדם עמוק ככל הניתן, עד שלא נשארו עוד קודקודים שלא בקרנו בהם. כשניתקע במקום ללא קודקודים נחזור קודקוד אחורה ונבדוק האם הוא מצביע לקודקודים שלא בוקרו, אם כן נסרוק אותם באותו אופן, אם לא נמשיך לחזור אחורה $(backtracking)$.

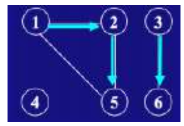


איור 82: דוגמה לריצת DFS

אם בסוף הריצה נותרו קודקודים שלא בוקרו, נחזור על האלגוריתם כאשר קודקוד המקור s הוא קודקוד שלא ביקרנו בו.

נחזור על התהליך עד שנבקר בכל הקודקודים.

דבר כזה יקרה רק כאשר הגרף אינו קשיר-חזק.



איור 83: דוגמה לריצת DFS על גרף שאינו קשיר-חזק

הערה בשביל לראות אילו קודקודים בוקרו ואילו לא, מספיק להוסיף לכל צומת בגרף שדה שיגיד לנו האם הקודקוד נסרק כבר או לא.

הערה התוצאה הסופית של ריצת האלגוריתם היא $depth - first\ forest$:

$$G_\pi = (V, E_\pi)\,,\ E_\pi = \{\,(v.\pi, v)|\, v \in V \wedge v.\pi \neq null\}$$

כאשר $v.\pi$ הוא הקודם של v בעץ החיפוש.

הערה G_π מייצג את היער הרקורסיבי: $DFS - vist\,(v)$ נקרא מ-$DFS - visit\,(u)$ אם"ם (u, v) צלע ב-G_π (שימו לב ש-(u, v) הוא זוג **סדור**).

כמו ב-BFS, גם ב-DFS יהיו לנו צמתים מכמה סוגים:

1. שסרקנו
2. שאנו סורקים כרגע, "קודקודים פתוחים"
3. שלא סרקנו.

לכל קודקוד כמה שדות:

1. $u.d$ - מתי גילינו את הקודקוד לראשונה.
2. $u.f$ - מתי סיימנו עם הקודקוד, כלומר מתי סיימנו עם כל שכניו.

Algorithm 26 $DFS\,(G, s)$

```
DFS(G, s)
for each u ∈ V do
    u.d ← null; u.f ← null; u.π ← null; u.label ← not_visited;
time ← 1
DFS − Visit (s)
for each u ∈ V do
    if u.label = not_visited then DFS − Visit (u)
```

Algorithm 27 $DFS - Visit\,(u)$

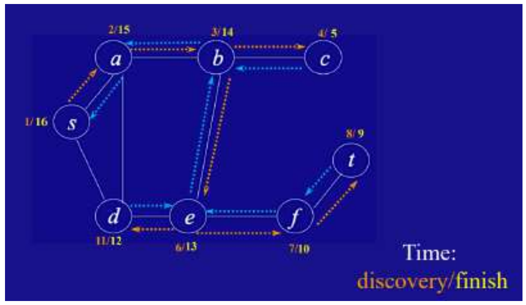
```
DFS-Visit(u)
u.label ← current; u.d ← time; time ← time + 1;
for each v ∈ neighbors (u) do
    if v.label = not_visited
        v.π ← u
        DFS − Visit (v)
u.label ← current; u.f ← time; time ← time + 1;
```

הערה נשים לב כי $time$ הוא משתנה $Integer$ גלובלי. $time$ מתעדכן רק כשמתעדכן אחד השדות $u.d$ או $u.f$, כלומר רק כשאנו מגלים קודקוד או מסיימים עם קודקוד.

הערה במקרה של גרף לא קשיר-חזק, אם נתחיל מקודקוד s ובסוף הריצה לא סרקנו את s', אז ברגע שנריץ את האלגוריתם על s' אנחנו לא נאפס את $time$, אלא נמשיך מהערך האחרון שהיה לו בריצה על s.

מהאלגוריתם (ומההערה הנ"ל) אנו מסיקים כי יער הפלט תלוי בסדר בחירת הקודקודים שאנו עוברים עליהם קודם (וכן בבחירת קודקוד ההתחלה).

הערה במקום רקורסיה, ניתן להשתמש בתור $LIFO$ (באופן אנלוגי לתור ה-$FIFO$ ב-BFS).



איור 84 : דוגמה לשדות הזמן בריצת DFS

20.3 סיבוכיות

- האלגוריתם סורק כל קודקוד $v \in V$ - $\Theta\left(|V|\right)$.
- DFS נקרא פעם אחת $\forall v \in V$ שכן הפעולה הראשונה היא לסמן קודקוד שלא נסרק בתור "נוכחי" (נסרק כרגע) - $\Theta\left(|V|\right)$.
- מספר הפעולות ב-$DFS - Visit\left(v\right)$ לא כולל קריאות רקורסיביות הוא קבוע כפול מספר הצלעות היוצאות מ-v. סה"כ על כל הקודקודים נקבל $\Theta\left(|E|\right) = \sum_{v \in V} |neighbors\left(v\right)|$.

סה"כ $\Theta\left(|V| + |E|\right) = \Theta\left(|V|\right) + \Theta\left(|E|\right)$, כלומר זמן לינארי ביחס לגודל הגרף.

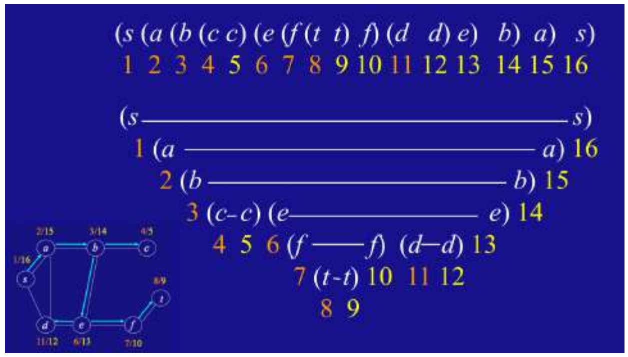
הערה $\Theta\left(|V| + |E|\right)$ הוא גם $\mathcal{O}\left(|V|^2\right)$.

תכונת הסוגריים

נניח שכל פעם שאנו מגלים קודקוד אנו פותחים סוגריים, וברגע שאנו מסיימים עמו אנחנו סוגרים סוגריים.

אנו טוענים כי מבנה הסוגריים הוא "חוקי" (לדוגמה $(())()$ הוא רצף חוקי, אך $)($ אינו חוקי, כי אנו סוגרים סוגריים לפני שהם נפתחו בכלל).

באופן שקול לטענה שמבנה הסוגרים חוקי, ניתן לומר כי מקטעי הזמן בהם קודקודים היו פתוחים (כלומר בתהליך סריקה) מקיימים יחס הלכה או יחס זרות.



איור 85: המחשה לתכונת הסוגריים ב-DFS

משפט (הסוגריים) בגרף DFS, $G=(V,E)$, לכל ששני קודקודים $u,v\in V$, מתקיים אחד ורק אחד מהבאים:

(i) הקטעים $[u.d,u.f]$, $[v.d,v.f]$ **זרים** ו-u,v אינם צאצאים אחד של השני ב-G_π.

(ii) הקטע $[u.d,u.f]$ **מוכל** ב-$[v.d,v.f]$ ו-u הוא צאצא של v ב-G_π.

(iii) הקטע $[v.d,v.f]$ **מוכל** ב-$[u.d,u.f]$ ו-v הוא צאצא של u ב-G_π.

נכונות המשפט (ומכך גם הנכונות שמבנה הסוגריים חוקי) נובעת מכך שאם פתחנו קריאה רקורסיבית ב-v, וירדנו אל u, לא נוכל לעלות חזרה ל-v ("ולסגור את הסוגריים") לפני שסיימנו עם u.

הוכחה: נניח ש-$u.d<v.d$ שכן המקרה ההפוך סימטרי. נחלק למקרים.

1. אם $u.f<v.f$ אזי $DFS-VISIT(v)$ נקרא כאשר $DFS-VISIT(u)$ היה פתוח ולכן v הוא בהכרח צאצא של u ו-$[v.d,v.f]\subseteq[u.d,u.f]$.

2. נניח ש-$u.f<v.f$ אזי $u.f<v.d$ ולכן $DFS-VISIT(v)$ נקרא לאחר $DFS-VISIT(u)$ ולכן v הוא לא צאצא של u וגם v לא צאצא של v ו-$[u.d,u.f]$, $[v.d,v.f]$ זרים.

∎

משפט. (המסלול הסרוק) ב-$depth-first\ forest$ של גרף $G=(V,E)$, קודקוד v הוא צאצא של u אם"ם בזמן $u.d$, קרי כש-u התגלה, יש מסלול מ-u ל-v המורכב מקודקודים שלא סרקנו.

הערה שימו לב שהפלט של DFS (היער) הוא גרף מכוון, לכן המושג צאצא מוגדר היטב.

הוכחה: $\Leftarrow$: ממשפט הסוגריים לכל קודקוד w במסילה מ-u ל-v מתקיים כי $u.d\le w.d$ כי הוא צאצא של u ולכן אם $u\neq w$ אזי $u.d<w.d$ ומכאן $w.label=not\_visited$.

$\Leftarrow$: נניח בשלילה ש-v לא צאצא של u. נניח בה"כ ש-v הינו הראשון במסילה שאינו צאצא של u. נביט ב-w שלפני v על המסילה, אזי w צאצא של u ולכן ממשפט הסוגריים

$$u.d \leq w.d \leq w.f \leq u.f$$

ומההנחה $u.f < v.d \leq v.f$ כי v לא צאצא של u ממשפט הסוגריים. נביט על הקריאה של $DFS - VISIT\ (w)$ בה v לא נסרק כלומר $v.label = not\_visited$, מתקיים כי $v.\pi = w$ כלומר v צאצא של w שצאצא של u ולכן v צאצא של u. סתירה. ■

21 מיון טופולוגי

יהי $G = (V, E)$ גרף מכוון. נחשוב על הקודקודים של G בתור משימות שאנו רוצים למצוא סדר לביצוע שלהן. החצים בין הקודקודים מייצגים את הקדימויות: אם יש חץ בין קודקוד u ל-v, משמע משימה u צריכה להתבצע לפני v.

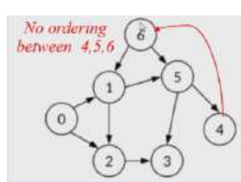
נרצה לסדר את הקודקודים שלנו בצורה שלא תפר את הקדימויות: אם $(u, v) \in E$ אז u יופיע לפני v ברשימת הפלט שלנו.

הגדרה. גרף מכוון א-ציקלי ($DAG - Directed\ Acyclic\ Graph$) הוא גרף מכוון חסר מעגלים.

הערה. אם ב-G **קיים** מעגל, לא נוכל לפתור את הבעיה. קרי, אין סידור לינארי לקודקודים באופן שתיארנו.
אם בגרף **אין** מעגלים, כלומר הגרף הוא גרף DAG, תמיד נוכל לפתור את הבעיה. זאת כי תמיד יש לנו איבר מינימלי (שלא נכנסים אליו חצים), שכן אין מעגלים. לכן נוכל לשים את האיבר המינימלי במקום הראשון ולחזור על התהליך עם הגרף שנותר (ללא האיבר הזה). כך תתקבל רשימה ממוינת טופולוגית.

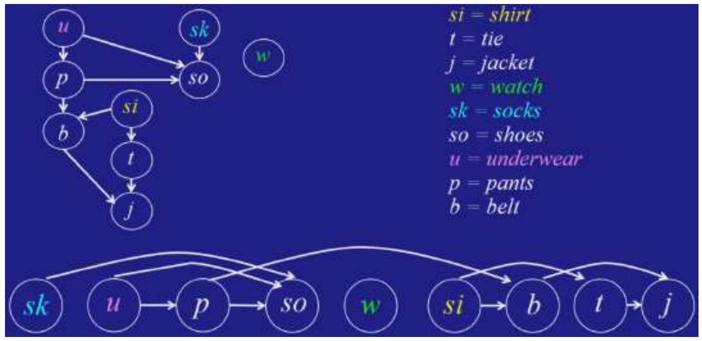
הערה האלגוריתם שתיארנו למציאת מיון טופולוגי לגרף DAG אינו יעיל, לכן נשתמש ב-DFS כדי למצוא פתרון טוב יותר.

דוגמה. בגרף הנ"ל, יש מעגל בין $4, 5, 6$. על משימה 6 להתבצע לפני 5, 5 לפני 4 ו-4 לפני 6, וזה לא הגיוני, לכן אין פתרון לבעיה.



איור 86: גרף שלא ניתן למצוא לו מיון טופולוגי

דוגמה לגרף הבא, המתאר סדר לבישת בגדים, קיים מיון טופולוגי כמתואר בתחתית האיור.



איור 87 : גרף שניתן למצוא לו מיון טופולוגי

נרצה לחשוב על פתרונות לבעיה. הפתרון הכי נאיבי שאפשר לחשוב עליו הוא לעבור על כל האפשרויות ולבחור את האפשרות שעונה על התנאים. אבל זה לא יעיל באופן קיצוני. פתרון אחר שאפשר לחשוב עליו הוא בהנתן קודקוד, ללכת אחורה עד שנגיע לקודקוד שלא יוצאים ממנו חצים ואז לשים אותו במקום הראשון ברשימה ולחזור על התהליך, זה יעבוד כי אין מעגלים בגרף. הבעיה בתהליך זה שיתכן שהוא בסיבוכיות ריבועית. מיון טופולוגי מאפשר לעשות זאת באופן לינארי.

להלן אלגוריתם למציאת מיון טופולוגי.

Algorithm 28 $Topological-Sort\left(G\right)$

1 : **Topological-Sort**(G)

2 : $DFS\left(G\right)$ // **Call** $DFS\left(G\right)$ **to compute finishing times**

3 : **return the list of vertices** $v_1,\ldots,v_n$ **s.t.** $v_1.f>\ldots>v_n.f$

משפט. $Topological-Sort\left(G\right)$ מחזיר רשימה ממוינת טופולוגית של הגרף G בזמן לינארי $\Theta\left(|V|+|E|\right)$.

הוכחה: נקבע (מלשון קיבוע) (u,v). נוכיח כי $v.f>u.f$. נראה ש-u מסתיים מאוחר יותר. אם גילינו את u קודם, ממשפט קודם מתקיים הדרוש. אחרת, גילינו את v קודם ואז מהיות u לא צאצא של v כי אין מעגלים ולכן אין מסלול של $not\_visited$ מ-v ל-u וגם עם קודקודים אחרים, מתקיים כי $[u.d,u.f]$, $[v.d,v.f]$ זרים. אבל $v.d<u.d$ ולכן $u.f,v.f$.
■

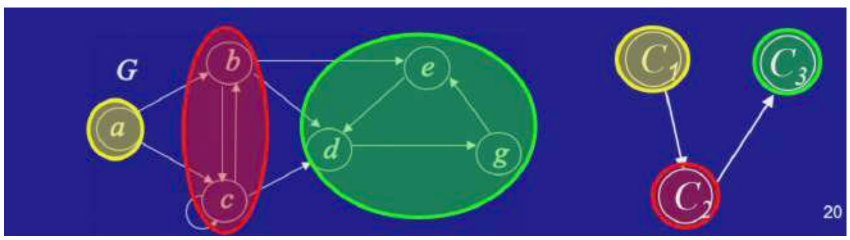
22 מציאת רכיבי קשירות חזקים

הגדרה. רכיבי קשירות חזקים של גרף הם תתי גרפים זרים ומקסימלים $C_1,\ldots,C_k$ כלומר אין להם קודקודים משותפים או צלעות משותפות ובתוכם יש מסלול בין כל שני קודקודים. הכוונה במקסימליות היא שאין אנו יכולים להוסיף קודקוד והתכונה תישמר.

נבחין כי מתקיימות התכונות הבאות:

- חלוקת הקודקודים: $C_i \cap C_j = \emptyset$ וגם $V = [] \bigcup \cdot_i C_i$ שכן אחרת נקבל סתירה למקסימליות.
- יש לו מבנה של גרף מכוון חסר מעגלים (DAG):
  - נוח לחשוב על כל רכיב קשירות כקודקוד ולחבר בין רכיבי הקשירות אם יש צלע בין קודקוד באחד לקדוקוד באחר. כלומר על גרף $\tilde{G}$ שבו יש צלע בין שני קודקודים C_i, C_j אם יש צלע בין קודקוד ב-C_j לקודקוד ב-C_j.
  - $\tilde{G}$ חסר מעגלים. אם היה מעגל, כל רכיבי הקשירות שנמצאים במעגל היו רכיב קשירות אחד.

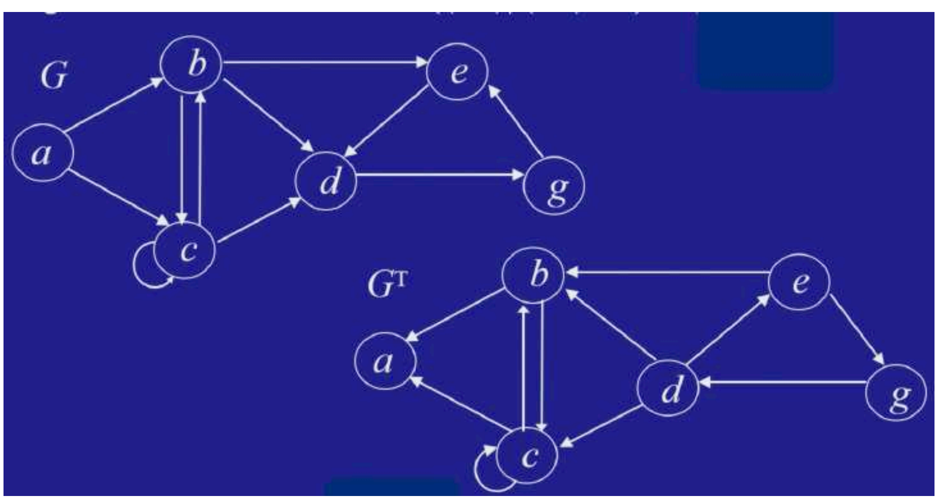
דוגמה. נביט בגרף הבא:



איור 88: דוגמא לרכיבי קשירות חזקים בגרף

המטרה שלנו היא למצוא אלגוריתם יעיל שבהנתן גרף מכוון, יתן לו חלוקה שלו לרכיבי קשירות חזקים.

הגדרה. הגרף המשוחלף $G^T = (V, E^T)$ הוא גרף שמתקבל מגרף $G = (V, E)$ על ידי היפוך הצלעות, כלומר $E^T = \{(u, v) \mid (v, u) \in E\}$. סימן ה-$Transpose$ נובע מכך שבמטריצת שכנויות נחשב את G^T לפי A^T.



איור 89: המחשה ל-G^T

הערה. שימוש אפשרי לרכיבי קשירות חזקים הוא רשתות תקשורת, בהן נרצה ליצור רשתות אוטונומיות לפי מודל של רכיבי קשירות חזקים.

על מנת למצוא רכיבי קשירות חזקים נבצע את התהליך הבא:

Algorithm 29 SCC (G)

1: **Strongly-Conected-Components**(G)

2: **DFS** (G) // **Call $DFS(G)$ to compute finishing times**

3: **compute** G^T

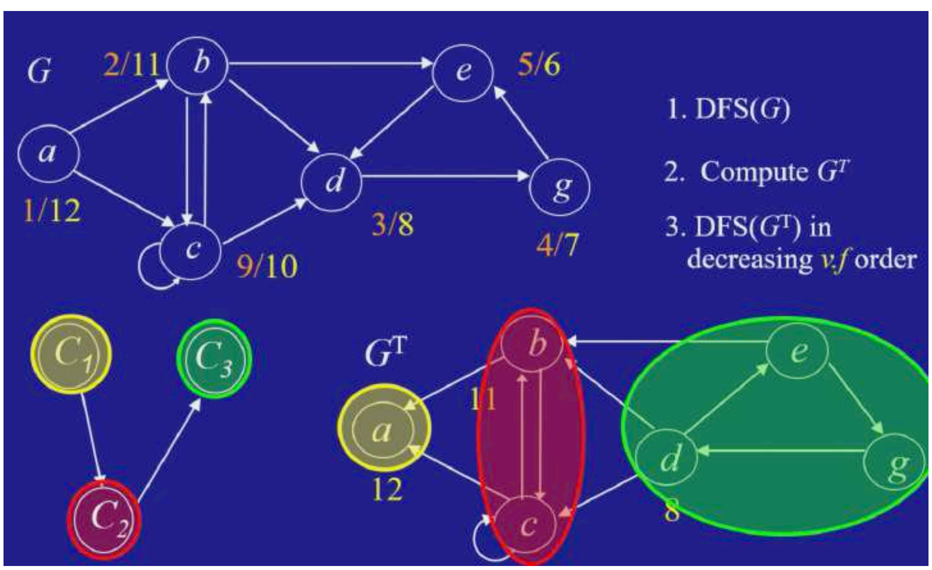
4: DFS $\left(G^T\right)$ such that in the outer loop in DFS run in the order of decreasing $v.f$

5: OUTPUT: vertices of each DFS tree from step 4 as a separate SCC

אנו למעשים מבצעים את השלבים הבאים:

- אנחנו מריצים את DFS ומקבלים יער עצי עומק DFS.
- לאחר מכן, אנו רצים על הקודקודים בעצים בעלי זמן הסיום הגדול ביותר ב-DFS על G^T עד שעברנו על כולם.
- מכיוון שהוא כל פעם רץ על עץ חדש, אבל עם היפוך הקשתות, וזה (ככל הנראה) יחזיר רכיב קשירות כי עכשיו עוברים על הצלעות בכיוון ההפוך.

דוגמה. נתן דוגמת הרצה.



איור 90: דוגמת הרצה ל-SCC

כאן אנו רואים למעשה שבהרצה הראשונה, DFS ממיין טופולוגית את הגרף לפי עומקם של רכיבי הקשירות. לאחר חישוב ה-$Transpose$, הסדר מתהפך, כלומר אנו מתחילים מהעמוק ביותר. לאחר שגילינו את רכיב הקשירות הראשון, עוברים לרכיב

הבא ואנו יודעים שברכיב זה חייב להיות מסלול בין כל שני קודקודים, לכן ה-$Transpose$ לא באמת משנה את הרכיב, כי אם אין מסלול כזה, זה לא רכיב קשירות חזק. אנו ממשיכים לעומק רכיבי הקשירות עד שעברנו על כל הקודקודים.

22.1 הוכחת נכונות

למה. יהיו C, C' שני רכיבי קשירות חזקים של G. אזי אם C' עמוק יותר, הוא יתגלה מוקדם יותר, כלומר אם $(u,v) \in E$ וגם $u \in C, v \in C'$ אזי $f(C) > f(C')$, כאשר $f(C) = \max\limits_{u \in C}(u.f)$ עבור $C \subseteq G$. כלומר לכל רכיבי הקשירות יש זמן גילוי ואם רכיב קשירות עמוק יותר, נסיים לעבור עליו מוקדם יותר.

הוכחה: כמו בהוכחה של מיון טופולוגי, רק אנלוגית על $\tilde{G}$. ■

מסקנה. לכל $(u,v) \in E^T$ ושמקיימים $u \in C, v \in C'$ מתקיים $f(C) < f(C')$ כאשר הסיום הוא לפי ריצת ה-DFS הראשונה ולא השנייה.

משפט. האלגוריתם מחשב נכונה את רכיבי הקשירות החזקים של הגרף המכוון G.

הוכחה: נוכיח באינדוקציה על k, כלומר ש-k העצים הראשונים שמצאנו באלגוריתם הם SCC.

בסיס: $k=0$, מקרה טריוויאלי, מתקיים באופן ריק.

שלב: נניח כי מצאנו k עצים שהם רכיבי קשירות חזקים. נוכיח כי העץ ה-$k+1$ הוא רכיב קשירות חזק.

נניח שהתחלנו להצמיח את העץ מהקודקוד u. נוכיח כי העץ הוא רכיב קשירות חזק C, כלומר שכל הקודקודים צאצאים של u ושאין ל-u עוד צאצאים שלא גילינו עד כה.

בזמן שגילינו את u לא התגלה שום קודקוד ברכיב הקשירות הזה, מצד שני הרכיב כולל מסלולים מ-u לכל הקודקודים. מכאן, כל קודקודים אלה הם צאצאים של u בעץ ($visited\ path\ theorem$). נוכיח כי אין עוד צאצאים.

מהנחת האינדוקציה, לכל רכיב קשירות C' שלא התגלה עד כה, מתקיים מהלמה כי $f(C') < f(C)$. מהמסקנה, אין צלע ב-G^T מ-C ל-C'. לכן, לכל צלע ב-G^T שיוצאת מ-C מתקיים כי היא מגיעה ל-SCC שכבר גילינו, אחרת היינו מקבלים ש-$f(C) < f(C')$. לכן, אין קודקוד ברכיב קשירות שונה מ-C שלא גילינו שהוא צאצא של u בריצת ה-DFS על G^T.

מכאן, אלה הקודקודים של עץ ה-DFS ב-G^T שמושרש מ-u בדיוק ברכיב קשירות אחד. נבין מדוע מדובר ברכיב קשירות חזק. ראשית, יש מסלול מ-u לכל קודקוד בעץ המושרש מריצת ה-DFS הראשונה שעליו אנו עוברים גם בפעם השנייה. עתה, מכיוון שאנו עוברים על G^T נשריש עץ חדש שמורכב אך ורק מהקודקודים שמהם ניתן להגיע ל-u ולכן התוצאה הסופית תתן רכיב קשירות בו מכל קודקוד ניתן להגיע לכל קודקוד אחר דרך u למשל. ■

חלק X

הרצאה $\mathbb{X}$ - עצים פורשים מינימלים

23 עצים פורשים מינימלים

23.1 מוטביציה

נרצה לפתור את הבעיה הבאה. נקבל כקלט גרף לא מכוון, קשיר וממושקל $G=(V,E)$. נרצה למצוא $T\subseteq E$ כך ש:

1. קבוצת הצלעות T מחברת בין כל הקודקודים בגרף.
2. הסכום של משקלי הצלעות הוא מינימלי.

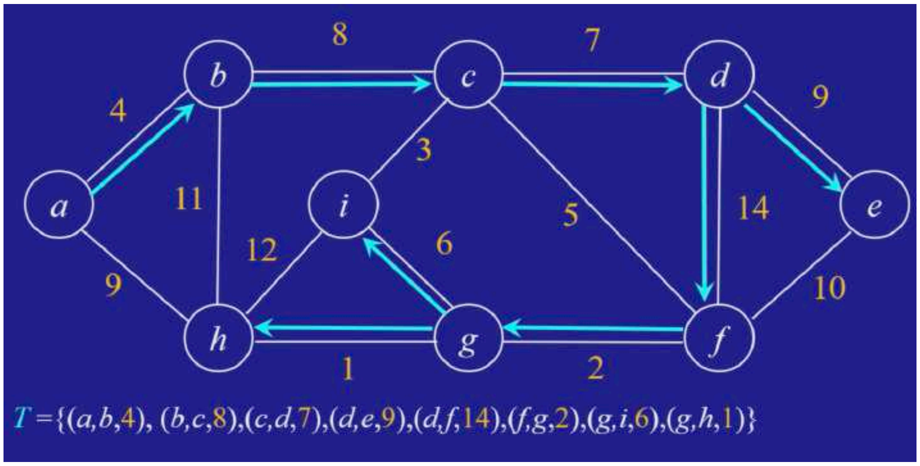
- מתקיים כי הגרף $G=(V,T)$ הוא עץ, והוא נקרא "העץ הפורש המינימלי" $(Minimum\ Spanning\ Tree)$ או בקיצור MST.
- התהליך מאוד דומה למציאת עצים ב-BFS וב-DFS רק הפעם עם גרפים ממושקלים וסכום משקלים מינימלי.

דוגמה. נניח שיש לנו אוסף אתרים הנמצאים במקומות גאוגרפים שונים, ואנו רוצים למתוח קווי תקשורת ביניהם כך שבין כל שני אתרים יהיה מסלול של קווי תקשורת. כל אתר יהיה קודקוד וכל קוו תקשורת יהיה צלע. המשקל יהיה המרחק מאתר א. לאתר ב. נרצה למצוא עץ פורש מינימלי שיקשר בין כל זוג קודקודים שמשקל קבוצת הצלעות שלו קטן ככל האפשרי, כלומר, נשתמש בכמה שפחות קווי תקשורת.

23.2 זהויות של עצים פורשים מינימלים

- גרף ממושקל הוא גרף בו לכל צלע יש משקל (מחיר), $w(v_i,v_j)>0$.
- גרף רגיל הוא מקרה פרטי של גרף ממשקל פשוט עם משקל 1 של כל צלע.
- עבור שני קודקודים v_i,v_j בלי צלע ביניהם נרשום $w(v_i,v_j)=\infty$.
- המחיר של נתיב בין u,v הוא סכום המשקלים של הצלעות בנתיב, כלומר $w(path(u,v))=\sum\limits_{i=1}^{k}w(v_{i-1},v_i)$ עבור $u=v_0,v_k=v$.
- המשקל של קבוצת צלעות T הוא סכום משקלי הצלעות, כלומר $w(T)=\sum\limits_{e\in T}w(e)$.

דוגמה. נביט בעץ הפורש בגרף הממושקל הבא:



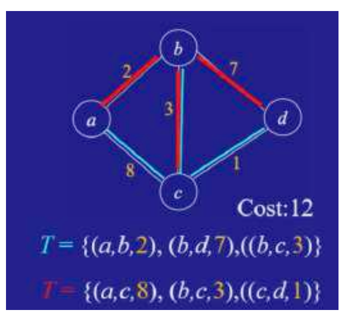
איור 91: דוגמת הרצה לעץ פורש בגרף ממושקל עם מחיר 51

ניתן לקבל עץ פורש טוב יותר עם משקל 39.

הגדרה. יהי $G=(V,E)$ גרף ממושקל, קשיר ולא מכוון. עץ פורש של G הוא תת קבוצה $T\subseteq E$ של צלעות מכוונות כך שתת הגרף $G'=(V,T)$ הוא קשיר וחסר מעגלים.

הגדרה. עץ פורש מינימלי (MST) היא עץ פורש עם משקל מינימלי: $w(T)=\sum_{(u,v)\in T} w(u,v)$. כאן אין צורך לדרוש שהוא אכן חסר מעגלים כי אם היה מעגל היינו יכולים להסיר צלעות ולקבל עץ עם משקל קטן יותר.

הערה יתכן שלגרף יהיה יותר מעץ פורש אחד. למשל, בגרף הבא:

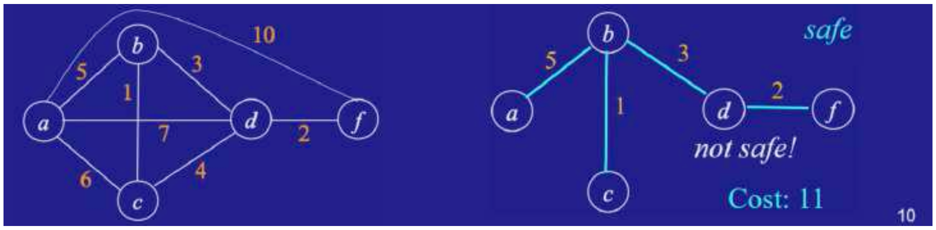


איור 92: גרף עם כמה עצים פורשים מינימלים

23.2.1 אלגוריתם גנרי למציאת עץ פורש מינימלי

האסטרטגיה שלנו היא חמדנית, כלומר נרוץ לפי התוצאה הטובה ביותר שנראית לעין. נתחיל עם $T=\emptyset$. נוסיף צלע אחת e בפעם אחת כך ש-$T\cup\{e\}$ תת קבוצה של עץ פורש מינימלי כלשהו, צלע כזו נקראת צלע בטוחה. התהליך יעצר כאשר הגענו לעץ פורש מינימלי. למעשה, לב העניין הוא למצוא צלע בטוחה, וכמובן איננו יודעים מה העץ שמכיל את T, אחרת לא היה צורך ב-T.

האסטרטגיה הנ"ל תהווה את הבסיס לאלגוריתמים הבאים שנראה. ניתן לקבל המחשה באיור הבא:



איור 93: דוגמת לאלגוריתם החמדן שתיארנו

כדי למצוא צלע בטוחה, נצטרך קצת טרמינולוגיה.

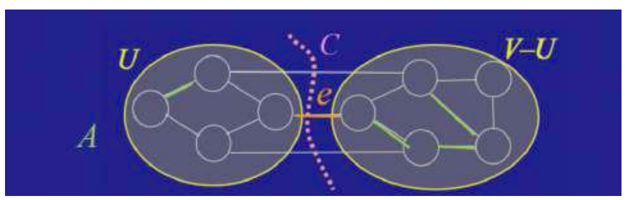
הגדרה. חתך $C=(U,V-U)$ של גרף לא מכוון $G=(V,E)$ הוא חלוקה של V לשתי קבוצות זרות שאיחודן הוא V.

הגדרה. צלע $e=(u,v)$ חותכת את החתך $C=(U,V-U)$ אם $u\in U$ וגם $v\in V-U$.

הגדרה. חתך $C=(U,V-U)$ מכבד את הקבוצה $A\subseteq E$ אם אין צלע ב-A שחותכת את C.

הגדרה. נאמר שצלע e היא צלע קלה אם היא הצלע עם המשקל המינימלי של כל צלע חותכת אחרת.

דוגמה. ניתן לקבל המחשה לחתך שמכבד קבוצה ולצלע חותכת באיור הבא:



איור 94: דוגמא לחתך ששומר על קבוצת הצלעות הירוקות. הצלע הכתומה חותכת את החתך והחתך לא מכבד את הקבוצה שמכילה אותה

משפט. יהי גרף $G=(V,E)$ גרף קשיר, ממושקל ולא מכוון. תהי $A\subseteq E$ שמוכלת בתוך עץ פורש מינימלי T של G. יהי חתך $C=(U,V-U)$ שמכבד את A. תהי $e\in(u,v)$ צלע קלה, אזי e היא צלע בטוחה, כלומר $A\cup\{e\}$ תת קבוצה של עץ פורש מינימלי.

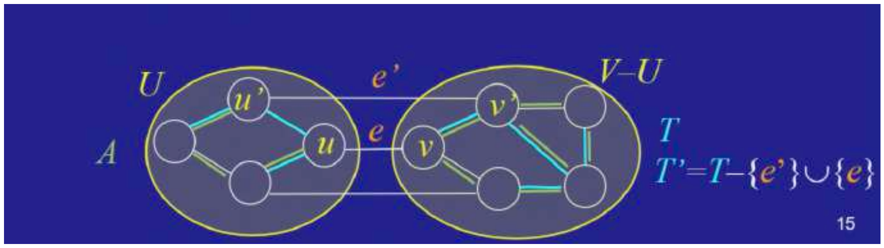
הוכחה: יהי T עץ פורש מינימלי שמכיל את A ונניח כי T לא מכיל את הצלע הקלה $e=(u,v)$, שכן אם הוא כן, אז היא צלע בטוחה. נמצא עץ פורש מינימלי T' שמכיל את $A\cup e$. מהיות u,v בצדדים שונים של החתך, יש לפחות צלע אחת $e'=(u',v')$ ב-T בנתיב p שחותך את החתך. הצלע e' היא לא ב-A כי החתך מכבד את A. על כן, A מוכלת ב-$T'=(T-\{e'\})\cup\{e\}$. על כן מספיק להוכיח ש-T' עץ פורש מינימלי. נוכיח כי הוא קשיר.

מהיות e' על מסלול יחיד מ-u ל-v ב-T, הורדה שלה שוברת את T לשני רכיבים. הוספת $e=(u,v)$ מחברת מחדש את שני הרכיבים ויוצרת עץ פורש חדש $T'=(T-\{e'\})\cup\{e\}$, לכן T' עץ. נוכיח כי הוא מינימלי. מהיות $e=(u,v)$ צלע קלה החותכת את $(U,V-U)$ ו-$e'=(u',v')$ גם חותכת את החתך, בהכרח $w(u,v)\le w(u',v')$ ולכן

$$w(T')=w(T)-w(u,v)+w(u',v')\le w(T)$$

אבל מההנחה, T עץ פורש מינימלי ומהיות $w\left(T'\right)\leq w\left(T\right)$ נקבל כי $w\left(T'\right)=w\left(T\right)$ ולכן T' הוא גם עץ פורש מינימלי.

ניתן להביט באיור להמחשה גרפית:



איור 95: המחשה לצעדי ההוכחה

ובזאת סיימנו את ההוכחה. ■

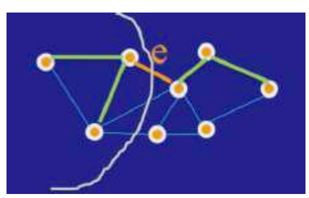
שאלה איך נמצא צלע בטוחה ב-A ביעילות?

תשובה יש לכך שני אלגוריתמים שמוצגים בהמשך.

23.3 האלגוריתם של קרוסקל $(Kruskal)$

הקבוצה A היא יער והצלע הבטוחה שמוסיפים היא הצלע עם המשקל המינימלי שמחברת שני עצים של A. כלומר,

- נתחיל עם קבוצה של עצים.
- נחבר כל פעם זוג עצים עם צלע בעלת משקל מינימלי.



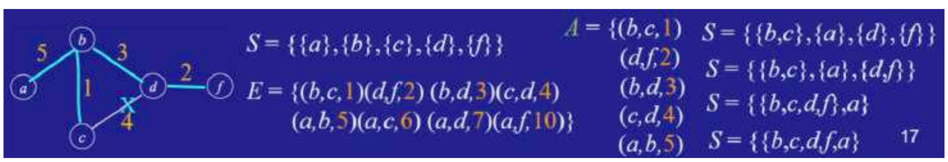
איור 96: המחשה לצעדי האלגוריתם

להל"ן, פסאודו קוד לאלגוריתם:

Algorithm 30 $MST - Kruskal\,(G)$

MST-Kruskal(G)
$A \leftarrow \emptyset$
for each vertex $v \in V$ **do: Make-Set**(v) // Saves collections of vertices. Each set represents a tree
Sort edges $e \in E$ **in increasing order**
for each edge $e = (u, v) \in E$ **do:**
    if Find-Set$(u) \neq$**Find-Set**(v) **then:** // The trees are distinct
        $A \leftarrow A \cup (e)$ // Add e to the tree
        Union(u, v) // Combine the trees

נביט בדוגמת ההרצה הבאה:



איור 97: דוגמת הרצה לאלגוריתם

23.3.1 ניתוח האלגוריתם

- נכונות: נובעת ישירות ממשפט הצלע הבטוחה.
- זמן ריצה: תלוי במימוש הפעולות על הקבוצה. מימוש נאיבי יקח $O\left(|V| \cdot |E|\right)$ (מושאר כתרגיל לקורא הנאמן והשקדן). יחד עם זאת, יש גם מימוש יעיל:
  - מיון הצלעות לוקח $O\left(|E| \log |E|\right)$.
  - לולאת ה-for עוברת על כל צלע ומבצעת שתי פעולות $Find - Set$ ופעולת $Union$ אחת. פעולות אלה ניתן לממש ב-$O\left(1\right)$ (נראה איך עושים זאת בהמשך הקורס).
  - זמן הריצה הכולל במימוש זה הוא $O\left(|E| \log |V|\right) = O\left(|E| \log |E|\right)$.

23.4 האלגוריתם של פרים $(Prim)$

הקבוצה A היא עץ והצלע הבטוחה שמוסיפים היא צלע עם משקל מינימלי שמחברת את A לקודקוד שלא ב-A. כלומר,

- נתחיל עם קודקוד אחד
- נצרף בכל פעם צלע עם משקל מינימלי שמחברת את A לקודקוד חדש.

איור 98 : המחשה לצעדי האלגוריתם

צעדי האלגוריתם המלאים הם כלדקמן :

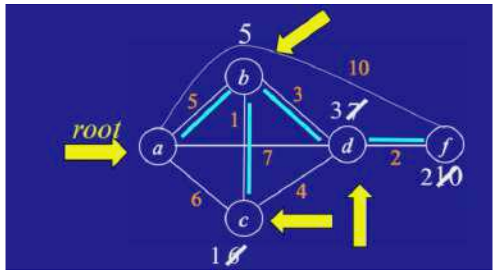
- נחזיק ערימת מינימום Q שתשמור את כל הקודקודים שלא הוכנסו לעץ.
- המפתח של כל קודקוד v ב-Q הוא המחיר המינימלי של צלע שמחבר בין v לעץ.
- נחזיק שדה נוסף $v.\pi$ לכל קודקוד כך ש-$(v.\pi, v)$ היא הצלע הקלה ביותר מ-v לעץ. כלומר נחבר את הקודקוד בערימה לעץ באמצעות $v.\pi$.

להל"ן, פסאודו קוד לאלגוריתם :

Algorithm 31 $MST - Prim\,(G)$

MST-Prim$(G = (V, E)\,, root)$
$Q \leftarrow V \setminus \{root\}$
$A \leftarrow \emptyset$
for each vertex $v \in Q$ **do:**
    $v.key \leftarrow w\,(root, v)$
    $v.\pi \leftarrow root$
while Q **is not empty do:**
    $u \leftarrow$**Extract-Min**(Q)
    Add$(u.\pi, u)$ **to** A
    for each v **that is a neighbor of** u **do:** // Update the most efficient way to connect v to A
        if $v \in Q$ **and** $w\,(u, v) < v.key$ **then:** // Save the heap updated
            $v.\pi \leftarrow u$
            $v.key \leftarrow w\,(u, v)$
return A

דוגמה. נביט בדוגמת ההרצה הבאה:



איור 99: דוגמת הרצה לאלגוריתם

23.4.1 ניתוח האלגוריתם

- נכונות: נובע ישירות ממשפט שהוכחנו.
- זמן ריצה: תלוי במימוש הפנימי של הערימה. באמצעות ערימת מינימום בינארית:
  - בניית הערימה היא $O\left(|V|\right)$.
  - $ExtarctMin$ דורשת $O\left(\log|V|\right)$ לכל קודקוד, לכן $O\left(|V|\log|V|\right)$ סך הכל.
  - הלולאה מתבצעת $|E|$ פעמים ממשפט לחיצות הידיים ובכל פעם, בדיקת שייכות היא $O\left(1\right)$, ושינוי ערכו של מפתח הוא $\log|V|$.
  - לכן זמן הריצה הכולל הוא $O\left(|V|\log|V|+|V|+|E|\log|V|\right)=O\left(|E|\log|V|\right)$.

חלק XI

הרצאה $\mathbb{XI}$ - המסלול הקצר ביותר לגרפים ממושקלים

24 מבוא

24.1 מובטיבציה

נזכיר כי יש לנו שלוש מטרות:

1. בעית המסילה הקצרה ביותר - בהנתן שני קודקודים s,t, נרצה למצוא את המסילה הקצרה ביותר בין s,t ואת אורכה.
2. בעיית המקור היחיד - בהנתן קודקוד s, נמצא עבור כל קודקוד אחר בגרף מסלול קצר ביותר מ-s אליו.

3. בעיית כל הזוגות - נמצא את המסלול הקצר ביותר בין כל הזוגות (s,t).

ראינו כבר פתרון ל-$1,2$ למקרה של גרפים לא ממושקלים, באמצעות BFS. עתה נרצה לפתור את הבעיה עבור גרפים ממושקלים.

דוגמא בהנתן מפה ושתי נקודות A,B נרצה למצוא את הדרך הקצרה ביותר בין A,B כאשר המשקל בין כל שני קודקודים הוא המרחק ביניהם.

24.2 הגדרות ותכונות

נזכיר כמה הגדרות.

- עבורנו, לכל זוג (v_i,v_j) יש משקל $w(v_i,v_j)>0$ חיובי.
- לשני קודקודים ללא צלע או מסלול ביניהם נגדיר $w(v_i,v_j)=\infty$.
- המשקל של מסלול $p(u,v)$ מסומן ב-$u\xrightarrow{p}v$ הוא סכום המשקולות של הצלעות במסלול: $w(p(u,v))=\sum_{i=0}^{k-1}w(v_i,v_{i+1})$.
- משקלו של המסלול הקצר ביותר מ-u ל-v מוגדר באופן הבא: $\delta(u,v)=\min\left\{w(p):u\xrightarrow{p}v\right\}$.

ניזכר בכמה תכונות.

- למסילה קצרה ביותר בין שני קודקודים אין מעגלים, כלומר היא מסילה פשוטה. דבר זה נובע מכך שאם יש מעגל, אפשר לעבור מיד להמשך המסלול לאחר המעגל ולקבל מסילה קלה יותר.
- תת מסילה של מסילה קצרה ביותר היא קצרה ביותר, כלומר אם $u\xrightarrow{p}v$ היא מסילה קצרה ביותר ו-x,y במסילה, אזי אם יש מסילה קצרה יותר בין x,y נקבל כי p איננה מסילה קצרה ביותר.

25 אלגוריתמים לפתרון הבעיה

כדי לפתור את בעיית המסלול הקצר ביותר עבור גרפים ממושקלים נסתכל על שני האלגוריתמים הבאים וננתח אותם:

1. דיקסטרה $(Dijkstra)$ - בסיבוכיות זמן ריצה של $\mathcal{O}(|E|\log|V|)$ וסיבוכיות מקום של $\mathcal{O}(|E|+|V|)$, עבור משקלים חיוביים.
2. בלמן פורד $(Bellmann-Ford)$ - בסיבוכיות זמן ריצה של $\mathcal{O}(|E|\cdot|V|)$ וסיבוכיות מקום של $\mathcal{O}(|E|+|V|)$, כאשר הוא עובד גם עבור משקלים שליליים.

לפני שננתח את שני האלגוריתמים, נרצה ללמוד על המשותף ביניהם.

25.1 המשותף בין האלגוריתמים

הגדרה. עץ שמייצג מסלול קצר מינימלי מהשורש s, הוא עץ פורש המושרש ב-s, כך שהנתיב בעץ בין s לכל קודקוד אחר הוא המסלול הקצר ביותר בגרף המקורי.

שאלה האם תמיד ניתן לייצג את כל המסלולים הקצרים ביותר באמצעות עץ?

תשובה לא. אם אנו מעוניינים במסלולים הקצרים ביותר בין כל הזוגות, זה כבר לא אפשרי. למשל, ניקח גרף משולש.

בשני האלגוריתמים נבצע את הדברים הבאים:

- נחזיר עצים כאלה, כמו שב-BFS החזרנו את $v.\pi$. כלומר, האלגוריתמים יחזירו עץ שמייצג מסלול קצר ביותר שמיוצג על ידי $v.\pi$. בסוף הריצה, העץ G_π הנתון על ידי $V_\pi = \{s\} \cup \{v : v.\pi \neq null\}$ ו-$E_\pi = \{(v.\pi, v) : v \in V_\pi\}$ יהיה עץ שמייצג מסלול קצר ביותר.
- נשמור שדה $v.dist$ שיהווה חסם מלעל על $\delta(s,v)$ ובסוף הריצה יתקיים השוויון $\delta(s,v) = v.dist$.
- **אתחול:** נאתחל לכל v נבצע $v.\pi = null$ ואם $s \neq v$ נגדיר $v.dist = \infty$. נגדיר $s.dist = 0$. כלומר העץ מאותחל להיות העץ הריק.
- לאחר האתחול, $v.\pi, v.dist$ משתנים רק על ידי הפונקציה $Relax(u,v)$ שפועלת על צלע.

הפונקציה $Relax$ מבצעת את הפעולות הבאות:

Algorithm 32 Relax (u,v)

```
Relax(u, v)
if v.dist > u.dist + w(u, v) then:
    v.dist ← u.dist + w(u, v)
    v.π ← u
```

כלומר אנו בודקים האם ההערכה הנוכחית של המרחק של u משפרת את ההערכה הנוכחית של המרחק של v מ-s. למשל אם $v.dist = 10, u.dist = 5, w(u,v) = 2$ יתקיים כי $v.dist = 10 > 7 + 2 = u.dist + w(u,v)$ ולכן נעדכן $v.dist = u.dist + w(u,v)$.

תכונות

- אם $v.\pi \neq null$ אזי $(v.\pi, v) \in E$, שכן אם הוא לא $null$ אז קראנו ל-$relax$ שעדכנה את u $v.\pi =$ כאשר $(u, v) \in E$.
- $v.dist \geq \delta(s, v)$, שכן לאחר ש-$v.dist$ מעודכן מתקיים כי $v.dist = u.dist + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)$.
- אם אין מסלול בין s, v אזי $v.dist = \delta(s, v) = \infty$.
- $v.dist$ רק יורד, שכן אם שינינו את $v.dist$, אזי לפי פעולות הפונקציה $Relax$, בהכרח הקטנו את הערך שלו. יתר על כן, אם $v.dist = \delta(s, v)$ בשלב מסוים, הוא לא יגדל כי הוא תמיד קטן והוא לא יקטן כי הוא מינימלי.

משפט. (תכונת ההתכנסות) תהי $s \xrightarrow{p} u \rightarrow v$ מסילה קצרה ביותר ב-G עבור $u, v \in V$. נניח כי קראנו ל-$Relax(u, v)$ והערך של u הוא הערך הנכון כלומר $u.dist = \delta(s, u)$. אזי $v.dist = \delta(s, v)$ בכל הזמנים לאחר מכן.

הוכחה: לאחר ש-v עודכן מתקיים כי

$$
\begin{aligned}
v.dist &\overset{Relax}{=} u, dist + w(u, v) \\
&= \delta(s, u) + w(u, v) \\
&= \delta(s, v)
\end{aligned}
$$

כאשר המעבר האחרון נובע מכך ש-p הוא מסלול קצר ביותר בין s ל-v. מכאן v לא קטן לאחר מכן. ■

מסקנה. אם $s \rightarrow v \rightarrow u \rightarrow w$ אז לאחר שנריץ $Relax(s, v)$ יתקיים כי $v.dist = \delta(s, v)$ ונקבל כי $Relax(v, u), Relax(u, w)$ יעדכנו את u, w למרחק מינימלי.

משפט. (תכונת הגרף הקודם) אם לכל $v \in V$ מתקיים כי $v.dist = \delta(s, v)$, אז לכל v מתקיים כי $v.\pi$ הוא קודקוד אחד לפני הקודקוד האחרון במסלול הקצר ביותר בין s ל-v. כלומר G_π הוא עץ שמייצג מסלול קצר ביותר.

הוכחה: (נראה בתרגול). ■

מסקנה. כדי להוכיח את נכונות האלגוריתמים, מספיק להוכיח שהם מחשבים את המרחקים נכונה.

הערה. הכוונה במסלול קצר ביותר היא למסלול בעל משקל מינימלי.

עד כה, כל התכונות היו משותפות לשני האלגוריתמים. נעבור לדבר על כל אחד מהם בנפרד.

25.2 האלגוריתם של דיקסטרה $(Dijkstra)$

האלגוריתמים מבצע את הצעדים הבאים:

1. נחזיק תור קדימויות מינימלי Q עם מפתחות $v.dist$ שיכיל את כל הקודקודים שלא ביקרנו בהם.
2. כאשר קודקוד יוצא מערימת המינימום Q, אנו מבצעים $Relax$ על כל הצלעות היוצאות ממנו.

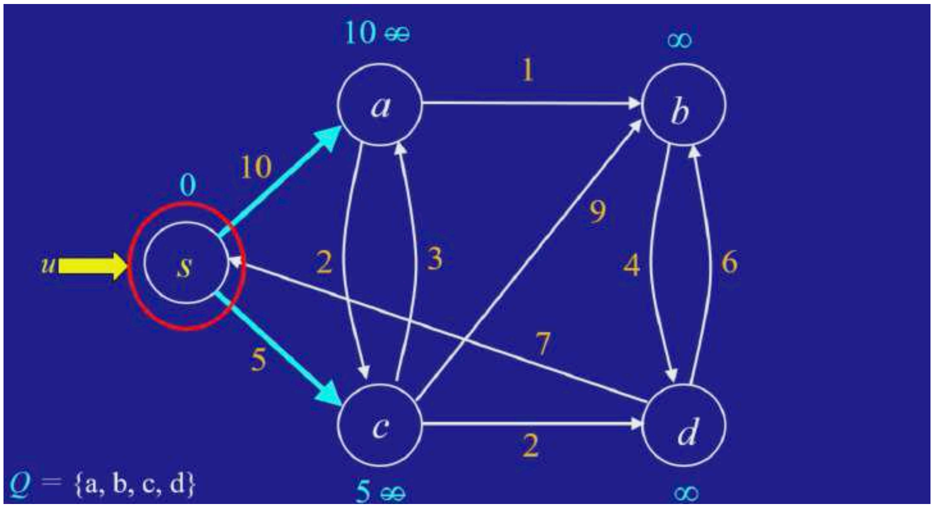
נרשום פסאודו קוד לאלגוריתם:

Algorithm 33 Dijkstra $(G = (V, E), s)$

Dijkstra$(G = (V, E), s)$
$s.dist \leftarrow 0;\ s.\pi \leftarrow null$
for all vertices u **in** $V - \{s\}$ **do:**
  $u.dist \leftarrow \infty;\ u.\pi = null$
$S = \emptyset$ // **stores the vertices the went out of the heap**
$Q \leftarrow V$ // **Adds all the vertices into the min heap**
while $Q \neq \emptyset$:
  $u \leftarrow$**Extract-Min**(Q)
  $S \leftarrow S \cup \{u\}$
  for each neitghbor v **of** u **do:**
    Relax(u, v)
    $S \leftarrow S \cup \{u\}$

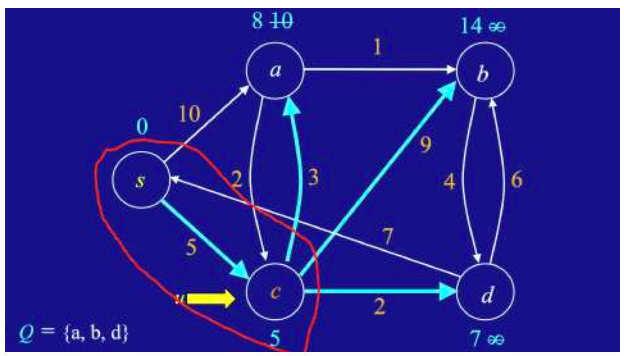
הקבוצה S היא רק לצורך ניתוח האלגוריתם ואין לה משמעות בפתרון הבעיה. נרצה לתת דוגמת הרצה לאלגוריתם:

בתחילת הריצה זה יראה כך:



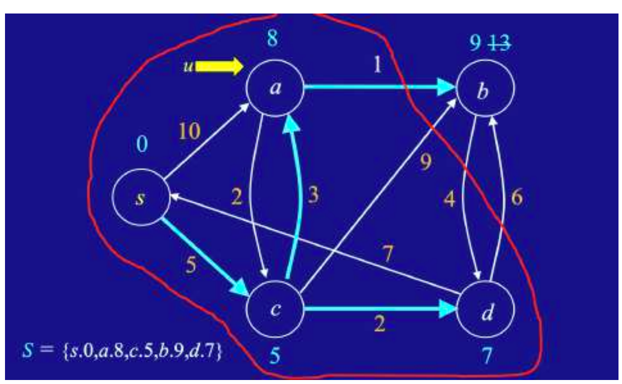
איור 100 : דוגמת הרצה לאלגוריתם, חלק 1

נבצע $Relax$ לכל הקודקודים. לאחר מכן נעבור לקודקוד c כי המשקל שלו קטן יותר מ-a :



איור 101 : דוגמת הרצה לאלגוריתם חלק 2

נחזור על התהליך עם d שכן הוא בעל משקל מינימלי, וכן הלאה... (מושאר לקורא כתרגיל) עד שנגיע לקודקוד האחרון a ונסיים :



איור 102 : דוגמת הרצה לאלגוריתם חלק אחרון

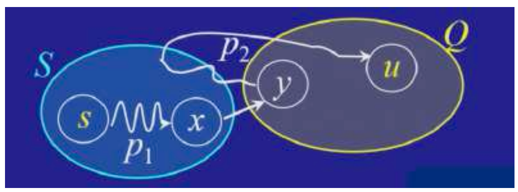
משפט. בסוף ריצת האלגוריתם של $Dijkstra$, מתקיים כי $u.dist = \delta(s, u)$ לכל $u \in V$.

מסקנה. העץ שהאלגוריתם מחשב הוא $Shortest - Path - Tree$.

הוכחה: נוכיח באינדוקציה על מספר ההוצאות מהתור Q. כלומר נראה שבכל הוצאה של קודקוד u מהתור, בהכרח יתקיים כי $u.dist = \delta(s,u)$.

<u>בסיס:</u> כאשר s יוצא מהתור מתקיים כי $s.dist = 0$ ולכן זה מתקיים.

<u>שלב:</u> נניח כי u יוצא מהתור Q. נוכיח כי לאחר שהוצאנו אותו יתקיים כי $u.dist = \delta(s,u)$. נסתכל על מסילה קצרה ביותר מ-s אל u. נחלק את הגרף לשני חלקים, S ו-Q. רגע לפני שהוצאנו את u נסתכל על החלוקה. בשלב זה הגרף נראה באופן כללי כך:



איור 103: המחשה לגרף רגע לפני שהוצאנו את u מ-Q

תהי p מסילה קצרה ביותר בין s ל-u. בשלב מסוים המסילה עוזבת את S. נסמן ב-y את הקודקוד הראשון במסילה שלא נמצא ב-S, ונסמן ב-x את הקודקוד הקודם ל-y במסילה. כלומר המסילה היא כזו: $s \xrightarrow{p_1} x \to y \xrightarrow{p_2} u$.

נוכיח כי $y.dist = \delta(s,y)$ כאשר u יוצא. נסתכל על הזמן שבו x הוצא מתוך Q. מהיות $x \in S$, מהנחת האינדוקציה, $x.dist = \delta(s,x)$. לאחר שהוצאנו את x נבצע $Relax$ לכל הצלעות שיוצאות מ-x ובפרט לצלע (x,y). כלומר ביצענו את הפעולה $Relax(x,y)$, אבל x לפני y במסלול קצר ביותר וגם $x.dist = \delta(s,x)$ לפני שמבצעים את הפעולה $Relax(x,y)$, ולכן מתכונת ההתכנסות, לאחר הקריאה $y.dist = \delta(s,y)$.

נוכיח כי $u.dist = \delta(s,u)$. מהיות y על מסילה קצרה ביותר מ-s ל-u ומתקיים כי y נמצא לפני u, מתקיים כי $\delta(s,y) \le \delta(s,u)$ לכן

$$y.dist = \delta(s,y) \le \delta(s,u) \le u.dist$$

אבל מהיות y ו-u בתור Q כאשר u יוצא, מתקיים כי $u.dist \le y.dist$ כי u נמצא מעליו בערימת המינימום, לכן

$$y.dist = \delta(s,y) \le \delta(s,u) \le u.dist \le y.dist$$

ולכן $u.dist = \delta(s,u)$. למעשה, y הוא בדיוק הקודקוד u, כי הוא נמצא באותו מסלול קצר ביותר ומתקיים כי המרחק שלו מ-s שווה למרחק של u מ-s. ■

25.2.1 ניתוח זמן ריצה

- האתחול לוקח $\mathcal{O}(|V|)$ כי בניית ערימה בגודל n היא $\mathcal{O}(n)$.
- לאחר מכן, שליפת כל קודקוד בערימה היא $\mathcal{O}(|\log|V||)$ ויש לנו בדיוק $|V|$ שליפות לכן סך הכל $\mathcal{O}(|V|\log|V|)$.
- הלולאה הפנימית רצה פעם אחת על כל צלע ובתוכה מבצעים $Relax$ שהיא לא בהכרח $\mathcal{O}(1)$, שכן אם עדכנו מפתח צריך לעדכן את הערימה ב-$\mathcal{O}(\log|V|)$. אנו רצים על כל הצלעות פעם אחת ולכן נקבל כי יש לכל היותר $|E|$ עדכונים, ולכן סך הכל $\mathcal{O}(|E|\log|V|)$.
- סך בכל נקבל $\mathcal{O}((|E|+|V|)\log|V|)$.

נבחין כי התוצאה תלויה במימוש התור. אם $|E|=\Omega\left(|V|^2\right)$, התוצאה לא אופטימילית, כי יש יותר פעולות של $Decrease\_key$ מאשר פעולות $Extract-Min$.

לכן כדי לפתור את הבעיה במקרה זה, נממש את תור העדיפויות כמערך ולא כעץ, כלומר לכל קודקוד v נחזיק את $v.dist$ במערך ונשמור שדה בוליאני שיאמר לנו אם v נמצא ב-Q או לא. כדי לאתחל, זה עדיין יקח $\mathcal{O}(|V|)$ כי צריך להגדיר את המשתנים עבור כל קודקוד.

עדכון ערך של קודקוד יהיה ב-$\mathcal{O}(1)$ כי צריך רק להכניס את הערך החדש למערך.

השליפה $Extract-Min$ תהיה יקרה, כי נצטרך לשלוף את המינימום מבין $|V|$ קודקודים. יחד עם זאת, יש רק $|V|$ שליפות לכך סך הכל $\mathcal{O}\left(|V|^2\right)$.

נבחין כי $|E|=\mathcal{O}\left(|V|^2\right)$ ולכן הלולאה הפנימית תיקח סך הכל $\mathcal{O}(|E|)$ כי בפנים יש פעולות שהן $\mathcal{O}(1)$. לכן סך הכל קיבלנו $\mathcal{O}(|V|)+\mathcal{O}\left(|V|^2\right)+\mathcal{O}(|E|)=\mathcal{O}\left(|V|^2\right)$ שזה יותר טוב כאשר $|E|=\Omega\left(\frac{|V|^2}{\log|V|}\right)$. כאשר $|E|$ קטן יותר עדיף להשתמש בשיטה הקודמת.

25.3 האלגוריתם של בלמן פורד $(Bellmann - Ford)$

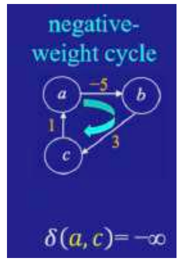
25.3.1 צלעות בעלות משקל שלילי

נאפשר לזוג v_i,v_j להיות בעל משקל $w(v_i,v_j)<0$. למשל יש לנו מדינות שמעבר ביניהן הוא פעולה והמשקולות הן מחירי הפעולוץ. מחיר חיובי הוא הפסד ומחיר שלילי הוא רווח.

מה משתבש כאשר מדברים על משקולות שליליות?

- משקל של נתיבים נשאר זהה.
- מסילה קצרה ביותר - אם יש מעגל שלילי בגרף, עבור זוגות מסוימים של קודקודים לא נוכל למצוא מסילה קצרה ביותר, אלא נגיע ל-$-\infty$. כלומר אין מינימום לאורך המסלול.

ניתן לקבל המחשה באיור המצורף:



איור 104: גרף עם מעגל של צלעות עם משקולות שליליות

תמיד נוכל להמשיך לטייל במעגל כדי להקטין את עלות המסלול.

- אם אין מעגלים שליליים, בין כל זוג קודקודים, אם נסתכל על מסילה בין הקודקודים האלה שיש בה מעגל (חיובי), נוכל למחוק את כל הקודקודים במעגל ולקבל מסילה קלה יותר. כלומר מספיק להסתכל על כל המסלולים ללא מעגלים, מכאן $-\infty < \min\left\{w(p) : u \overset{p}{\to} v \; with \; no \; cycles\right\} = \min\left\{w(p) : u \overset{p}{\to} v\right\} = \delta(u,v)$.
- מסלול קצר ביותר מוגדר היטב כי $-\infty < \delta(u,v)$ ויש נתיב שהמשקל שלו הוא $\delta(u,v)$.
- תתי מסילות של מסילות קצרות ביותר הן עדיין מסילות קצרות ביותר, מאותו נימוק.
- מסלול קצר ביותר הוא ללא מעגלים - מלבד מסילות עם מעגלים ששוקלים אפס, אך גם במקרה הזה ניתן לקבל מסילה קצרה ביותר ללא מעגלים.

עתה נבחין במה שלא נשמר:

- אם u מופיע לפני v במסלול קצר ביותר מ-s, אזי לא בהכרח $\delta(s,u) \leq \delta(s,v)$, למשל, אם נבחר משקל שלילי בין u ל-v נקבל שהטענה לא מתקיימת. כלומר $Dijkstra$ לא עובד יותר!

25.3.2 אופן פעולת האלגוריתם

באופן כללי, הוא דומה לדיקסטרה:

- אנו עדיין משתמשים ב-$Relax(u,v)$ כדי שלנות את המשקולות ועדיין מבטיחים כי:
  - $v.dist \geq \delta(s,v)$.
  - $v.dist$ ממשיך לקטון.
  - תכונת ההתכנסות עדיין מתקיימת.
  - תכונת הגרף הקודם עדיין מתקיימת - G_π הוא עץ שמייצג מסלול מינימלי.

- האלגוריתם יבצע $|V|-1$ איטרציות (שכן המסלול בין כל שני קודקודים הוא לכל היותר באורך $|V|-1$).
- בנוסף, הוא יגלה את כל המעגלים השליליים בגרף.

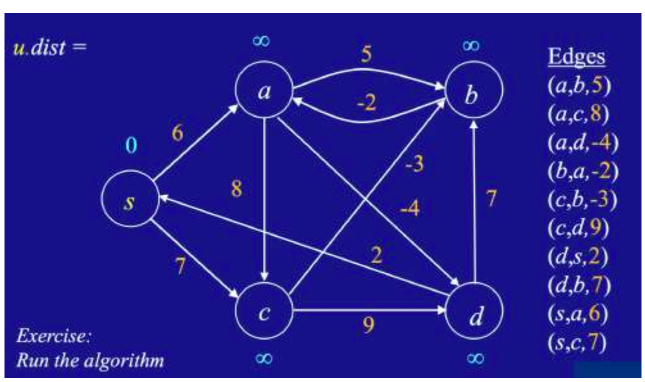
נרשום פסאודו קוד לאלגוריתם:

Algorithm 34 Bellmann-Ford (G, s)

Bellmann-Ford(G, s)
  Initilize(G, s) // **define** $v.dist$ **and** $v.\pi$
  for $i \leftarrow 1$ **to** $|V|-1$ **do:**
    for each edge$(u, v) \in E$ **do:**
      Relax(u, v)
  for each edge$(u, v) \in E$ **do:**
    if $v.dist >$ $u.dist + w(u, v)$ **then:** // **If we do relax, will it improve anything? It shouldn't!**
      return "Negative Cycle"
  return "No Negative Cycles"

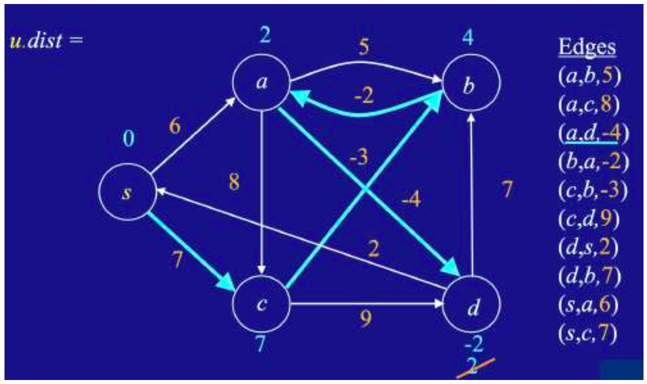
אם אין מעגלים שליליים האלגוריתם ימצא ימצא מסלולים קצרים ביותר. אם יש מעגלים האלגוריתם ימצא מעגל שלילי וידווח על זה.

ניתן דוגמת הרצה לאלגוריתם ונבחין כי בכל איטרציה, מה שרלוונטי הוא הצלעות שיוצאות מקודקודים שהערך שלהם הוא לא אינסוף:



איור 105: דוגמת הרצה לאלגוריתם חלק 1

המעבר לחלק הבא מושאר כתרגיל לקורא הנאמן:



איור 106: דוגמת הרצה חלק 2

כן, זה מייגע.

25.3.3 ניתוח האלגוריתם

משפט. נניח כי אין מעגלים שליליים ויהי v קודקוד. יהי $v_0, \ldots, v_k$ מסלול קצר ביותר מ-s ל-v שלא מכילה מעגל. בפרט $k \leq |V|-1$, אחרת היה קודקוד שמופיע פעמיים ולכן היה במסילה מעגל. נוכיח את נכונותה אלגוריתם באמצעות אינדוקציה על האינווריאנטה הבאה: לאחר האיטרציה ה-i, המרחק של הקודקוד ה-i הוא המרחק הנכון מ-s.

הוכחה: נוכיח באינדוקציה.

<u>בסיס</u>: לאחר אפס מעברים על הצלעות, המרחק של $s.dist = \delta(s,s) = 0$ כי כך אתחלנו אותו. יתר על כן, לאחר שעברנו על כל הצלעות בפעם הראשונה נקבל כי הקודקוד ה-1 גם יכיל את המרחק הנכון מתכונת ההתכנסות. יתר על כן, לאחר מעבר על כל הצלעות בפעם השנייה הקודקוד השני יכיל את המרחק הנכון מתכונת ההתכנסות וכן הלאה וכן הלאה.

<u>צעד</u>: באופן פורמלי, נניח כי הטענה נכונה עבור $i-1$ ונוכיח עבור האיטרציה ה-i. מהנחת האינדוקציה $v_{i-1}.dist = \delta(s, v_{i-1})$, לכן מתכונת ההתכנסות, לאחר שמבצעים $Relax(v_{i-1}, v_i)$ נקבל כי $v_i.dist = \delta(s, v_i)$.

בפרט, לאחר האיטרציה ה-k יתקיים כי $v.dist = \delta(s,v)$. עתה, מתכונת הגרף הקודם, G_π הוא עץ שמייצג מסלול קצר ביותר, ומושרש ב-s. ■

משפט. אם נריץ את האלגוריתם של בלמן-פורד על גרף ממושקל, מכוון $G=(V,E)$ עם מקור s, אזי:

(i): אם בגרף אין מעגלים שליליים שניתן להגיע אליהם מ-s, אז האלגוריתם מחזיר "$NoNegativeCycles$" ומתקיים $v.dist = \delta(s,v)$ לכל $v \in V$ ו-G_π הוא עץ שמייצג מסלול קצר ביותר מ-s.

(ii): אם יש מעגל שלילי שניתן להגיע אליו מ-s, האלגוריתם מחזיר "$NegativeCycle$".

הוכחה: (i) : נובע ישירות מהאינוריאנטה שהוכחנו.

(ii) : בתרגול נוכיח שאם היה מעגל שלילי, היינו בהכרח מגלים אותו. ■ נחשב את סיבוכיות זמן הריצה של האלגוריתם :

- בחלק הראשון, האלגוריתם רץ $|V|-1$ פעמים ובכך פעם הוא רץ $|E|$ פעמים שבכל פעם מתבצעות פעולות שהן $\mathcal{O}(1)$, לכן סך הכל $\mathcal{O}((|V|-1)\cdot|E|)=\mathcal{O}(|V|\cdot|E|)$.
- בחלק השני, האלגוריתם רץ $|E|$ פעמים ומבצע פעולות שהן $\mathcal{O}(1)$, ולכן סך הכל $\mathcal{O}(|E|)$.
- סך הכל $\mathcal{O}(|V|\cdot|E|)+\mathcal{O}(|E|)=\mathcal{O}(|V|\cdot|E|)$.
- סיבוכיות המקום היא פשוט הזכרון הדרוש לאחסון הגרף - $\mathcal{O}(|E|+|V|)$.

סיכום

פתרון בעיית המסילה הקצרה ביותר על גרפים ממושקלים נעשית באמצעות הפעלת $Relax$ על הצלעות. פעולה זו מתבססת על אי שוויון המשולש בנתיבים - אם $E \in e=(u,v)$ אזי $\delta(s,v)\leq\delta(s,u)+w(u,v)$.

על מנת לפתור את הבעיה ראינו שני אלגוריתמים :

- $Dijkstra$ - סיבוכיות זמן ריצה $\mathcal{O}((|V|+|E|)\log|V|)$ או $\mathcal{O}\left(|V|^2\right)$ אם $|E|=\Omega\left(\frac{|V|^2}{\log|V|}\right)$, סיבוכיות מקום $\mathcal{O}(|V|+|E|)$. האלגוריתם לא עובד על גרפים עם משקולות שליליות.
- $Bellmann - Ford$ - סיבוכיות זמן ריצה $\mathcal{O}(|V|\cdot|\mathrm{E}|)$, סיבוכיות מקום $\mathcal{O}(|V|+|E|)$. האלגוריתם עובד על גרפים עם משקולות שליליות.

חלק XII

הרצאה $\mathbb{XII}$ - בעיית כל המסלולים הקצרים ביותר, "כפל מטריצות", האלגוריתם של פלויד וורשל

26 מבוא

בהרצאה זו נלמד את הדברים הבאים:

- הגדרות וזהויות במציאת כל המסלולים הקצרים ביותר
- אלגוריתם פשוט לפתור הבעיה ב-$\mathcal{O}\left(|V|^4\right)$.
- אלגוריתם טוב יותר ב-$\mathcal{O}\left(|V|^3\log|V|\right)$.
- האלגוריתם של פלויד-וורשל ב-$\mathcal{O}\left(|V|^3\right)$.

עד כה, פתרנו את הבעיות הבאות:

1. בעיית המקור היחיד - פתרנו באמצעות BFS לגרפים לא ממושקלים, ובאמצעות $Dijkstra, Bellmann-Ford$ לגרפים ממושקלים.
2. מציאת רכיבי קשירות חזקים - DFS

היום נפתור את הבעיה הבאה:

בעיה. נרצה למצוא את הנתיב הקצר ביותר בין כל זוג (s,t).

דוגמה. $Waze, Google-Maps$. נמצא את כל המסלולים הקצרים ביותר ונשלוף אותם במהירות בעת הצורך.

פתרון נאיבי לבעיה יהיה לרוץ על כל אחד מהקודקודים ולהריץ את אלגוריתם ה-$single-source-shortest-path$ שלנו. ככה נקבל את המסלול הקצר ביותר בין כל זוג קודקודים אפשרי. מה יהיה זמן הריצה?

- אם בחרנו בבלמן פורד, נקבל $\mathcal{O}\left(|V|^4\right)=\mathcal{O}\left(|E|\cdot|V|^2\right)=\mathcal{O}\left(|E|\cdot|V|\right)\cdot\mathcal{O}\left(|V|\right)$.
- אם בחרנו בדיקסטרה, נקבל $\mathcal{O}\left(|V|^3\log|V|\right)=\mathcal{O}\left(|E|\log|V|\right)\cdot\mathcal{O}\left(|V|\right)$. אבל, אם יש משקולות שליליות, זה לא יעבוד. לכן הגישה הטובה ביותר כרגע היא הרצה של בלמן-פורד על כל הקודקודים.

נראה שני אלגוריתמים דינאמיים לפתרון הבעיה המלאה:

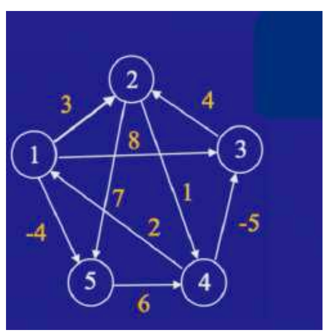
- האלגוריתמים הנ"ל דינאמיים, הם מסתמכים על תוצאה טובה מאיטרציה קודמת כדי להתקדם קדימה.
- אלגוריתם שמשתמש בכפל מטריצות "$Matrix - multiplication$" - $\mathcal{O}\left(|V|^3 \log |V|\right)$. באופן פרטי, בשלב ה-$i$, הוא ידע למצוא את המסלול הקצר ביותר שמכיל i צלעות.
- $Floyd - Warshall$ - ב-$\mathcal{O}\left(|V|^3\right)$. בכל שלב i הוא ימצא את המסלולים הקצרים ביותר שמותר להם להשתמש בקבוצה מוגבלת של קודקודים.

27 זהויות והגדרות

תיאור הגרף נשתמש במטריצת שכנויות כדי לתאר את הגרף שלנו $n \times n = |V| \times |V|$ כאשר $V = \{v_1, \ldots, v_n\}$.

הגדרה. <u>מטריצת הצלעות</u> היא מטריצה W שכל שורה i ועמודה j בתוכה מייצגות משקלים של צלעות מהקודקוד ה-i והקדוקוד ה-j בהתאמה. כלומר, $W(i,j)$ הוא משקל הצלע שיוצאת מ-i ל-j. כלומר, $W(i,j) = \begin{cases} 0 & i = j \\ w(i,j) & i \neq j \wedge (i,j) \in E \\ \infty & i \neq j \wedge (i,j) \in E \end{cases}$.

דוגמה. עבור הגרף הבא:



איור 107: גרף מכוון וממושקל

מתקיים כי

$$W = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 0 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

(וודאו שזה אכן כך).

הגדרה. מטריצת משקלי הנתיבים הקצרים ביותר היא מטריצה L שנסמנה $L = (l)_{ij}$ ומתקיים כי $(l)_{ij}$ הוא המשקל המינימלי של מסילה מ-i ל-j ואינסוף אם אין מסילה כזו.

דוגמה. בגרף מהדוגמא הקודמת מתקיים:

$$L = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

שוב וודאו שזה אכן כך.

נרצה גם למצוא את המסלולים הקצרים ביותר. ניתן לכל זוג לשמור רשימה מקושרת של המסלול הקצר ביותר, אך זה יקח במקרה הגרוע $\mathcal{O}\left(|V|^3\right)$ שזה מאוד בזבזני. לכן, נשתמש במבנה אחר כמו שעשינו ב-DFS.

הגדרה. מטריצת הקודמים היא מטריצה Π ומתקיים כי $[\Pi]_{ij}$ היא הקודם של j במסלול מ-i ל-j. או $null$ אם $i = j$.

הערה. נבחין כי Π מספיקה לחישוב המסלול הקצר ביותר בין כל זוג קודקודים, שכן עבור זוג (i, j) הקודם הוא $prev = \Pi\left[i, j\right]$ ונלך ל-$\Pi\left[i, prev\right]$ נחזור על התהליך עד שנגיע ל-i. המסלול שנקבל הוא בהכרח המסלול הקצר ביותר. תהליך זה הוא לינארי בגודל המסילה.

מסקנה. בהנתן W מספיק לחשב את L ו-Π כדי לקבל את המסלולים הקצרים ביותר.

27.0.1 קבלת המסלולים הקצרים ביותר בהנתן Π

נרשום פסאודו קוד לאלגוריתם שתיארנו קודם למציאת המסלול הקצר ביותר בהנתן Π:

Algorithm 35 Print-All-Pairs-Shortest-Path (Π, i, j)

```
Print-All-Pairs-Shortest-Path:
if i == j :
      print i
else if π_ij == NIL :
      print "no path from" i "to" j "exists"
else Print-All-Pairs-Shortest-Path(Π, i, π_ij)
      print j
```

זהו פשוט אותו תהליך שתיארנו רק בניסוח רקורסיבי ולא איטרטיבי.

28 הכפלת מטריצות (Matrix-Multiplication-Algorithm)

28.1 מבוא

הגדרה. <u>מטריצת המרחקים הקצרים ביותר המאולצת</u> מסומנת ב- $L^{(m)} = \left(l_{ij}^{(m)}\right)_{ij}$ ומתקיים כי $l_{ij}^{(m)}$ היא המשקל המינימלי של כל נתיב מ-i ל-j שיש לו לכל היותר m צלעות. מתקיים כי $l_{ij}^{0} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases}$ ואת L^0 נסיק בהתאם.

שאלה מהי L^1?

תשובה מתקיים כי $l_{ij}^{1} = \begin{cases} 0 & i = j \\ w(i,j) & i \neq j \wedge (i,j) \in E \\ \infty & i \neq j \wedge (i,j) \notin E \end{cases}$ ולכן $L^1 = W$.

שאלה מהי L^n?

תשובה לכל זוג קודקודים יש לכל היותר מסילה באורך n ולכן $L^n = L$.

שאלה מהי L^{n+1}?

תשובה באותו האופן $L^{n+1} = L$ ובאופן כללי לכל $k \geq n$ מתקיים כי $L^k = L$.

המטרה שלנו באלגוריתם זה היא למצוא את $L^{(q+r)}$ בהנתן $L^{(q)}, L^{(r)}$. ואנו נבצע זאת ב-$\mathcal{O}\left(|V|^3\right)$.

אנו רוצים לעשות זאת, מכיוון שברגע שנצליח לעשות זאת, נוכל לחשב את $L^{\left(2^k\right)}$ ב- $\mathcal{O}\left(k\,|V|^3\right)$. שכן, $L^{(1)} \to L^{(2)} \to L^{(4)} \to \ldots \to L^{\left(2^k\right)}$ ב-k צעדים ולכן זה $\mathcal{O}\left(k \cdot |V|^3\right)$. נוכל לבחור $k = \log_2 |V|$ ונקבל $2^k = |V|$ ולכן $L^{\left(2^k\right)} = L$ ומכאן נוכל למצוא את L ב-$\mathcal{O}\left(|V|^3 \log_2 |V|\right)$.

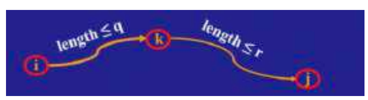
28.2 האלגוריתם

ניסוח האלגוריתם ונכונותו יתקבלו מהלמה הבאה:

למה. לכל $i, j \in V$ מתקיים כי $l_{ij}^{(q+r)} = \min_{k \in V}\left(l_{ik}^{(q)} + l_{kj}^{(r)}\right)$.

נבין מדוע זה נכון.

מתקיים כי כל מסילה באורך לכל היותר $q + r$ ניתן לפרק לשתי מסילות, האחת באורך **לכל היותר** q והשנייה באורך **לכל היותר** r. נסתכל על החיבור ביניהן k, ונמצא את k שנותן מסלול מינימלי. כלומר, נעבור על כל הקודקודים האפשריים k, נחשב את $l_{ik}^{(q)}, l_{kj}^{(r)}$, נחבר. נבחר את k שנותן מסלול מינימלי.



איור 108: המחשה ללמה

הוכחה: (הלמה) נוכיח אי שוויון דו כיווני.

ראשית, נוכיח כי $l_{ij}^{(q+r)} \leq \min_{k \in V}\left(l_{ik}^{(q)} + l_{kj}^{(r)}\right)$. מספיק להוכיח כי לכל $k \in V$ מתקיים כי $l_{ij}^{(q+r)} \leq l_{ik}^{(q)} + l_{kj}^{(r)}$. יהי $k \in V$.

אזי קיימות מסילות $p_1 \in P_{ik}^{(q)}, p_2 \in P_{kj}^{(r)}$ כך ש-$w(p_1) = l_{ik}^{(q)}, w(p_2) = l_{jk}^{(r)}$. נחבר את שתי המסילות p_1, p_2 ונקבל מסילה $p \in P_{ij}^{(q+r)}$ עם משקל $w(p) = w(p_1) + w(p_2) = l_{ik}^{(q)} + l_{jk}^{(r)}$ ולכן

$$l_{ij}^{(q+r)} = \min_{p \in P_{ij}^{(q+r)}} w(p) \leq l_{ik}^{(q)} + l_{jk}^{(r)}$$

כרצוי.

עתה נוכיח את אי השוויון $l_{ij}^{(q+r)} \geq \min_{k \in V}\left(l_{ik}^{(q)} + l_{kj}^{(r)}\right)$.

יהי $p = (v_0, \ldots, v_s)$ מסלול מ-i ל-j עם משקל $w(p) = l_{ij}^{(q+r)}$ ו-$s \leq q + r$. יהי $t = \min(q, s)$. נפרק את p לשני

מסלולים:

$$p_1 = (v_0, \ldots, v_t)$$
$$p_2 = (v_t, \ldots, v_s)$$

עם אורכים $t \leq q$, $s - t \leq q + r - t = \begin{cases} q + r - q & q < s \\ q + r - s & q \geq s \end{cases}$. לכן

$$l_{ij}^{(q+r)} = w(p) = w(p_1) + w(p_2) \geq l_{iv_t}^{(q)} + l_{v_tj}^{(r)} \geq \min_{k \in V} \left(l_{ik}^{(q)} + l_{kj}^{(r)} \right)$$

כרצוי. ■ נרשום פסאודו-קוד לאלגוריתם שבהנתן $A = L^{(q)}, B = L^{(r)}$ יתן את $L^{(q+r)}$:

Algorithm 36 Extend-Shortest-Paths (A, B)

Extend-Shortest-Paths(A, B)
$n \leftarrow A.rows$
let $C = (c_{ij})$ **be an** $n \times n$ **matrix**
for $i \leftarrow 1$ **to** n **do:**
  for $j \leftarrow 1$ **to** n **do:**
    $c_{ij} \leftarrow \min_k (a_{ik} + b_{kj})$
return C

קל לראות שזמן הריצה הוא $\Theta\left(|V|^3\right)$, כי רצים n^2 פעמים ובכל אחת מהן מחפשים מינימום ב-$O(n)$.

נשאלת השאלה, מאיפה מגיע השם מכפלת מטריצות? מכיוון שהוא מאוד דומה לאלגוריתם לכפל מטריצות:

Algorithm 37 Square-Matrix-Multiply (A, B)

```
1 : Square-Matrix-Multiply(A, B)
2 : n ← A.rows
3 : let C = (c_ij) be an n × n matrix
4 : for i ← 1 to n do:
5 :     for j ← 1 to n do:
6 :         c_ij ← Σ_{k=1}^{n} a_ik b_kj, min_k (a_ik + b_kj)
7 : return C // returns A × B
```

$$c_{ij} \leftarrow \sum_{k=1}^{n} a_{ik}b_{kj},\ \cancel{\min_{k}\left(a_{ik}+b_{kj}\right)}$$

ההבל היחיד הוא בשורה 6, זה גם הבדל מאוד מהותי. במקום לכפול את שני האיברים ולסכום את כל התוצאות, אנו עוברים על כל זוגות האיברים ומוצאים מביניהם את המינימום, כלומר עושים פעולה על השורה והעמודה כמו בכפל מטריצות רק עושים פעולה אחרת.

מכאן נסיק את האלגוריתם הבא למציאת כל המסלולים הקצרים ביותר:

Algorithm 38 Faster-All-shortest-Paths (W)

```
Faster-All-Shortest-Paths(A, B)
n ← A.rows
L^1 ← W
m ← 1
while m < n − 1 do:
    L^(2m) ← Extend-Shortest-Paths(L^(m), L^(m))
    m ← 2m
return L^m
```

מהלמה שהוכחנו, נובעת נכונות האלגוריתם ומתקיים כי הסיבוכיות שלו היא $\Theta\left(|V|^3\log_2|V|\right)$, שכן אנו רצים $\Theta\left(\log_2|V|\right)$ פעמים ובכל פעם עושים פעולה של $\Theta\left(|V|^3\right)$.

נעיר רק כי $\ldots = L^{(n-1)} = L^{(n)} = L^{(n+1)}$.

נרצה למצוא את Π. יש שתי גישות לחישוב. אפשר לחשבו אותו ב-$\mathcal{O}\left(|V|^3\right)$ מ-L.

נבחין כי π_{ij} הוא הקודקוד ה-k שהופך את $L_{ik}+W_{kj}$ למינימלי יש $|V|$ אפשרויות כאלה ויש $|V|^2$ זוגות לכן נקבל $\mathcal{O}\left(|V|^3\right)$. האם אפשר לעשות זאת באופן יותר יעיל.

מכאן נקבל כי חישוב L,Π הוא ב-$\mathcal{O}\left(|V|^3\log|V|\right)$. דרך אחרת היא לחשב את Π באמצעות חישוב L, כלומר במהלך ריצת האלגוריתם.

29 האלגוריתם של פלויד-וורשל (Floyd-Warshall)

29.1 מבוא

גם אלגוריתם זה משתמש במטריצת משקלים מוגבלת.

- נניח כי $V=\{1,\dots,n\}$.
- בהנתן מסילה $p=(v_0,\dots,v_r)$, קודקוד פנימי במסיל הוא כל קודקוד במסילה p שהוא לא v_0,v_r.
- מטריצת המרחקים של המסלולים הקצרים ביותר מסומנת ב-$D^{(k)}=\left(d_{ij}^{(k)}\right)_{ij}$ ומקיימת כי-$d_{ij}^{(k)}$ היא המשקל המינימלי של כל מסלול מ-i ל-j עם קודקודים פנימיים $\{1,2,\dots,k\}$.
  - k הוא לא רק כמות הקודקודים הפנימיים, הוא גם מגדיר בדיוק באילו קודקודים מותר להשתמש והם $\{1,\dots,k\}$.

שאלה מהי $D^{(0)}$?

תשובה $W_{ij}=\begin{cases}w(i,j) & i\neq j\\ 0 & i=j\end{cases}=d_{ij}^{(0)}$ ולכן $D^{(0)}=W$.

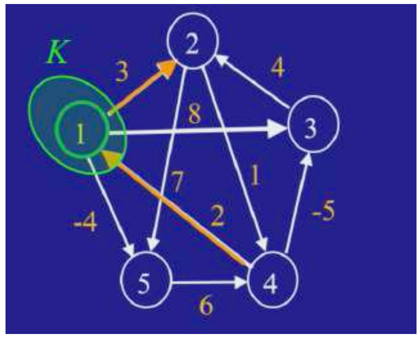
שאלה מהי $D^{(n)}$?

תשובה מתקיים כי $D^{(n)}=L$ שכן יש לכל היותר n קודקודים פנימיים.

בשונה מהאלגוריתם הקודם, אנו נחשב את $D^{(k)}$ מתוך $D^{(k-1)}$, בזמן הרבה יותר יעיל, ב-$\mathcal{O}\left(|V|^2\right)$.

מכאן נקבל כי $L^1\to L^2\to\dots\to L^n=L$ ב-n צעדים ולכן $\mathcal{O}\left(|V|^3\right)$, וכמו שראינו קודם נוכל לחשב את Π ב-$\mathcal{O}\left(|V|^3\right)$ וסך הכל נחשב את הכל ב-$\mathcal{O}\left(|V|^3\right)$.

דוגמה. נביט בגרף הבא:



איור 109: דוגמא לחלק מריצת האלגוריתם

מתקיים כי

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \boxed{5} & -5 & 0 & \boxed{-2} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

נבחין כי אם נאפשר להשתמש בקודקוד הפנימי 1, נוכל לקבל מסלולים קצרים יותר.

29.2 האלגוריתם

הלמה הבאה תעזור לנו לנסח את האלגוריתם ולהוכיח את נכונותו:

למה. $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$.

נסביר מה ההגיון מאחורי הלמה:

- יש שני סוגי מסילות, כאלה שעוברות דרך הקודקוד k וכאלה שלא.
  - מסילות שלא עוברות דרכו נכללות ב-$d_{ij}^{(k-1)}$.
  - מסילות שכוללת אותו, מגיעות ל-k מ-i ומשם ממשיכות ל-j ולכן אורך המסילה המינימלית מביניהן הוא המינימום להגעה מ-i ל-k ועוד המינימום להגעה מ-k ל-j כלומר $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

- מכאן, תמיד $d_{ij}^{(k)} = \min\left(\text{מינימום ממסלולים שלא עוברים דרך k}, \text{מינימום למסלולים שעוברים דרך k}\right) = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$.

הוכחה: נסמן ב-$P_{ij}^{(k)}$ את קבוצת כל הנתיבים מ-i ל-j עם קודקודים פנימיים שמוכלים בקבוצה $\{1, \ldots, k\}$.

מכאן, נקבל כי

$$\begin{aligned} d_{ij}^{(k)} &= \min_{p \in P_{ij}^{(k)}} w\left(p\right) \\ &= \min\left(\min_{p \in P_{ij}^{(k-1)}} w\left(p\right), \min_{p \in P_{ij}^{(k)} \setminus P_{ij}^{(k-1)}} w\left(p\right)\right) \\ &= \min\left(d_{ij}^{(k-1)}, \min_{p \in P_{ij}^{(k)} \setminus P_{ij}^{(k-1)}} w\left(p\right)\right) \end{aligned}$$

נבחין כי המסילות ב-$P_{ij}^{(k)}$ שלא נמצאות ב-$P_{ij}^{(k-1)}$ בהכרח עוברות דרך k. נראה זאת באופן פורמלי.

תהי p_{ij} מסילה עם משקל מינימלי ב-$P_{ij}^{(k)} \setminus P_{ij}^{(k-1)}$. אזי p_{ij} בעלת משקל מינימלי מקרב כל הנתיבים מ-i ל-j עם קודקודים פנימיים $\{1, 2, \ldots, k\}$ ובהכרח עם k כקודקוד פנימי. נוכיח כי $w\left(p_{ij}\right) = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

נשבור את p_{ij} לשני נתיבים מ-i ל-k ומ-k ל-j ונסמנם p_{ik}, p_{kj}.

הנתיב p_{ik} הוא נתיב בעל משקל מינימלי מ-i ל-k עם הקודקודים הפנימיים $\{1, 2, \ldots, k-1\}$. באופן דומה p_{kj} נתיב בעל משקל מינימלי מ-j ל-k עם הקודקודים הפנימיים $\{1, 2, \ldots, k-1\}$. לא יתכן שהם לא מינימלים, אחרת נקבל סתירה למינימליות p_{ij}. מכאן

$$w\left(p_{ij}\right) = w\left(p_{ik}\right) + w\left(p_{kj}\right) = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$$

כרצוי. ■

נציג פסאודו קוד לאלגוריתם:

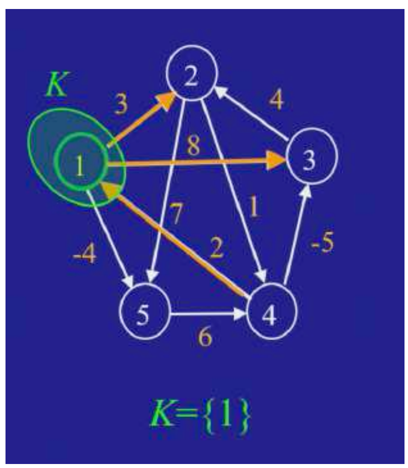
Algorithm 39 Floyd-Warshall (W)

Floyd-Warshall(W)
$n \leftarrow W.rows$
$D^{(0)} \leftarrow W$
for $k \leftarrow 1$ **to** n **do:**
    for $i \leftarrow 1$ **to** n **do:**
        for $j \leftarrow 1$ **to** n **do:**
            $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
return $D^{(n)}$

נכונות האלגוריתם נובעת ישירות מהלמה.

נרצה לראות דוגמת הרצה לאלגוריתם.

דוגמה. תחילה נתון הגרף הבא:



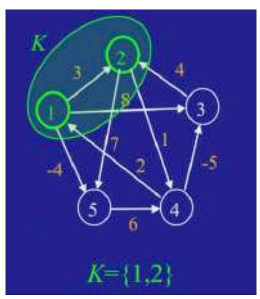
איור 110: דוגמת הרצה לאלגוריתם חלק 1

ונקבל את המטריצות הבאות:

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \boxed{\infty} & -5 & 0 & \boxed{\infty} \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

נעבור לאיטרציה הבאה:

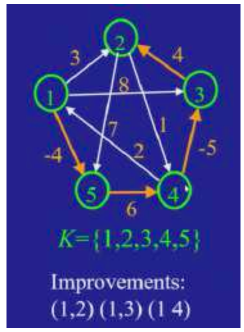


איור 111: דוגמת הרצה לאלגוריתם חלק 2

נבחין כי $d^{(2)}_{14} = \min\left(\infty, d^{(1)}_{12} + d^{(1)}_{24}\right) = \min(\infty, 3+1) = 4$ ולכן נקבל את המטריצה:

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

נחזור על התהליך ונקבל לבסוף:



איור 112: דוגמת הרצה לאלגוריתם, סוף

ואת המטריצה:

$$L = D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

וזהו הפלט הסופי.

סיכום מה שראינו עד כה

ראינו שני אלגוריתמים:

- כפל מטריצות: $\mathcal{O}\left(|V|^3 \cdot \log|V|\right)$. אילצנו את האלגוריתם להסתכל באיטרציה ה-i על מסלול באורך i.
- האלוגריתם של פלויד מרשל: $\mathcal{O}\left(|V|^3\right)$. אילצנו את האלגוריתם בכל צעד k להסתכל על הקודקודים הפנימיים בטווח $\{1, \ldots, k\}$.
- שני האלגוריתמים הם אלגוריתמים דינאמיים, אלגוריתמים שבכל צעד מקלים על האילוצים ומשתמשים בתוצאה מאילוץ גבוה יותר. ככה בצעד הסופי מגיעים לתוצאה הרצויה.

חלק XIII

הרצאה $\mathbb{XIII}$ - קבוצות זרות

$(Disjoint\ Sets - Union\ Find)$

30 מבוא

כשדיברנו על עצים פורשים ראינו את האלגוריתם $Kruskal$ בשתי צורות. בצורה הראשונה המימוש שלו לקח $\mathcal{O}\left(|E|\cdot|V|\right)$ ובצורה השנייה $\mathcal{O}\left(|E|\log|V|\right)$:

Algorithm 40 $MST-Kruskal\left(G\right)$

```
1: MST-Kruskal(G)
2: A ← ∅
3: for each vertex v ∈ V do: Make-
   Set(v) // Saves collections of vertices. Each set represents a tree
4: Sort edges e ∈ E in increasing order
5: for each edge e = (u, v) ∈ E do:
6:     if Find-Set(u) ≠ Find-Set(v) then: // The trees are distinct
7:         A ← A ∪ (e) // Add e to the tree
8:         Union(u, v) // Combine the trees
```

בצורה זו, הנחנו שמבנה הנתונים $Union-Find$ נותן לנו זמן ריצה $\mathcal{O}\left(1\right)$ בשורות $6,8$. היום נלמד בדיוק איך לממש אותו בצורה זו.

30.1 הגדרת הבעיה

בעיה. בהנתן n איברים שונים $X=\{x_1,\ldots,x_n\}$, נרצה לתחזק אוסף $\mathcal{S}=\{S_1,\ldots,S_k\}$ של קבוצות זרות שיכילו את איברי X כך ש-$S_i\cap S_j=\emptyset$ לכל $i\neq j$.

נרצה לאפשר את הפעולות הבאות:

- פעולות מפתח:
  - $Make-Set\left(x\right)$ - ניצור קבוצה S_x שהאיבר היחיד בה יהיה x. נניח כי x לא נמצא באף קבוצה אחרת.
  - $Union\left(x_i,x_j\right)$ - מחליף את שתי הקבוצות S_i,S_j באיחוד שלהן שיכיל אך ורק את x_i,x_j. שוב נניח כי x_i,x_j בקבוצות שונות.
  - $Find-Set\left(x\right)$ - נמצא ונחזיר נציג של הקבוצה S_x שמכילה את x, הנציג הוא יחיד, כלומר אם x,y נמצאים באותה קבוצה, אזי בהכרח $Find-Set\left(x\right)==Find-Set\left(y\right)$.

מטרתנו היא לתחזק מבנה נתונים אבסטרקטי שתומך ב-m קריאות, בכל פעם לפונקציה מסוימת מבין השלוש, כך שזה יעשה באופן יעיל. אפשר לנתח מה זמן הריצה הגרוע לכל פונקציה ומה סיבוכיות זמן הריצה של m הפעולות.

כלומר, נקבל

- קבוצה X כך ש-$n=|X|$.
- סדרה של שאילתות: $M-S\left(x_1\right),F-S\left(x_2\right),U\left(x_3,x_2\right),\ldots,M-S\left(x_n\right)$.

ונרצה לדעת

- מה זמן הריצה של כל שאילתה.
- מה זמן הריצה הכולל של כל שאילתות.
  - המוטיבציה מאחורי חישוב זה, הוא שבאלגוריתם קרוסקל אנחנו מתעניינים בזמן הריצה הכולל של השאילתות $Find-Set,Union$.

הפתרון הנאיבי לבעיה הוא להחזיק רשימות מקושרות כקבוצות. $make-set$ תהיה $\mathcal{O}\left(1\right)$ כי יוצרים רשימה מקושרת חדשה. בהנחה שהנציג הוא האיבר הראשון ברשימה, $find-set$ תהיה $O\left(n\right)$ כי נצטרך לעבור על כל הרשימות עד שנמצא את הנציג. הפעולה $Union$ היא $\mathcal{O}\left(n\right)$ כי בהנתן שני איברים נצטרך למצוא שתי רשימות ב-$\mathcal{O}\left(n\right)$ ואז נצטרך לאחד אותן ב-$\mathcal{O}\left(1\right)$. מכאן, כדי לבצע m שאילתות של $Find-Set,Union$ נקבל $\mathcal{O}\left(m\cdot n\right)$.

היות נראה שניתן לממש זאת בזמן קרוב ל-$\mathcal{O}\left(n+m\right)$ כלומר ב-$\mathcal{O}\left(1\right)$ בממוצע.

30.2 מוטיביציה - מציאת רכיבי קשירות

באמצעות מבנה נתונים זה נוכל למצוא את רכיבי הקשירות של גרף.

Algorithm 41 $Connected - Components\,(G)$

Connected-Components$(G=(V,E))$

for each vertex $v \in V$ **do: Make-**

Set(v) // **Saves collections of vertices. Each set represents a tree**

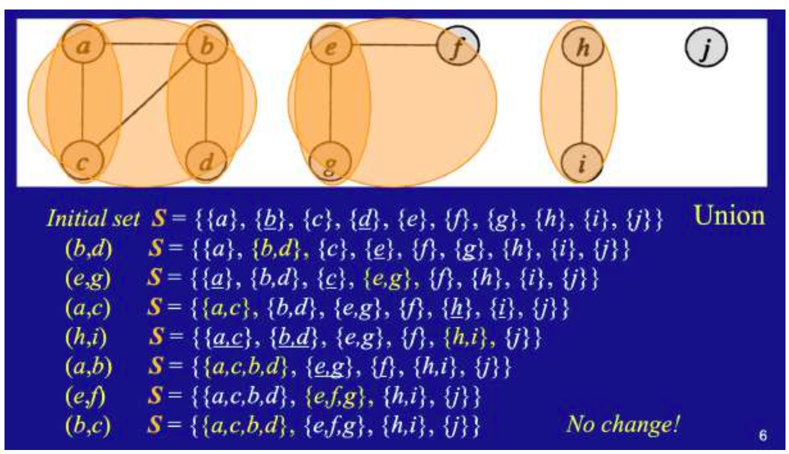
for each edge $e=(u,v)\in E$ **do:**

  if Find-Set(u) $\neq$**Find-Set**(v) **then:** // **The trees are distinct**

    Union(u,v) // **Combine the trees**

קל להשתכנע שהאלגוריתם עובד. נבחין כי באלגוריתם אנו מבצעים $|V|$ שאילתות של $Make-Set$ ו-$|E|+|E|+|E|$ שאילתות של $Find-Set$ כי באיחוד מבצעים בדיוק שתיים, ולכן סך הכל $|V|+3|E|$ שאילתות, על $|V|$ קבוצות ומכאן זמן הריצה שלו הוא $\mathcal{O}\left(f\left(|V|,3|E|+|V|\right)\right)$ כאשר $f\left(n,m\right)$ היא זמן הריצה של m פעולות על n איברים ב-$Union-Find$.

נראה דוגמת הרצה:



איור 113: דוגמת הרצה לאלגוריתם

מכאן גם נבין שאם זמן הריצה הוא $n+m$ נקבל שהאלגוריתם רץ בזמן לינארי $\mathcal{O}\left(|V|+|E|\right)$.

עתה ננתח את זמן הריצה של קרוסקל עם מבנה זה:

Algorithm 42 $MST-Kruskal\left(G\right)$

1 : **MST-Kruskal**(G)

2 : $A \leftarrow \emptyset$

3 : **for each vertex** $v \in V$ **do: Make-Set**(v) // Saves collections of vertices. Each set represents a tree

4 : **Sort edges** $e \in E$ **in increasing order**

5 : **for each edge** $e = (u, v) \in E$ **do:**

6 : **if Find-Set**(u) $\neq$**Find-Set**(v) **then:** // The trees are distinct

7 : $A \leftarrow A \cup (e)$ // Add e to the tree

8 : **Union**(u, v) // Combine the trees

יש לנו $|V|$ שאילתות של $M-S$, $3\left|E\right|$ שאילתות של $F-S$ ולכן $f\left(\left|V\right|, 3\left|E\right|+\left|V\right|\right)$ וסך הכל $\mathcal{O}\left(\left|E\right|\log\left|V\right|+f\left(\left|V\right|, 3\left|E\right|+\left|V\right|\right)\right)$ ואם f אכן לינארית נקבל $\mathcal{O}\left(\left|E\right|\log\left|V\right|\right)$.

31 פתרונות לבעיה

במימוש הנאיבי ראינו כי $f\left(n, m\right) = n \cdot m$. נראה שלושה מימושים אפשריים שיתנו זמן ריצה טוב יותר:

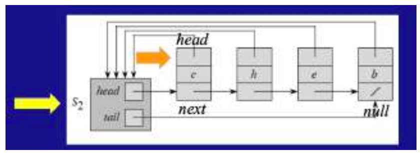
1. וריאציה על הפתרון הנאיבי - באמצעות רשימות מקושרות לפי משקל (אורך הרשימה). נראה שזה יתן $\mathcal{O}\left(m+n\log n\right)$.
2. באמצעות יער של קבוצות זרות עם הוריסטיקה אחת לאיזון גובהי העץ - $\mathcal{O}\left(m\log n\right)$.
3. באמצעות יער, אבל עם 2 הוריסטיקות לאיזון גובהי העצים - $\mathcal{O}\left(m\alpha\left(n\right)\right)$ עבור $\alpha\left(n\right)$ שרצה לאט יותר מ-$\log^{*}\left(n\right) = \log\left(\log\left(\log\ldots\left(n\right)\right)\right)$.

31.1 פתרון ראשון - רשימות מקושרות

- נשמור רשימה מקושרת לכל קבוצה L_i של איברים. הנציג של הרשימה יהיה הראש שלה. לרשימה יהיו שתי שדות:
  - נשמור מצביע לראש הראשונה $head$.
  - נשמור מצביע לסוף הרשימה $tail$.
- לכל איבר $x \in X$ יהיו שתי שדות:

– $x.next$ - מצביע לאיבר הבא ברשימה המקושרת שלו L_i, או $null$ אם x לא שייך לרשימה או נמצא בסוף רשימה.

– $x.head$ - מצביע לראש הרשימה L_i, או $null$ אם x לא באף רשימה. כלומר נוכל להגיע לנציג הרשימה, ראש, הרשימה וזנב הרשימה בהנתן איבר x ב-$\mathcal{O}\left(1\right)$.

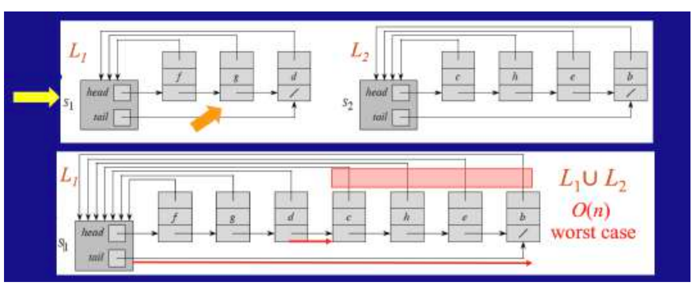


איור 114 : המחשה למימוש המבנה

אנו לא נקצה זכרון בכל פעולת הוספה, אלא נקצה לכל איבר x מראש את השדות $next, head$ כך ש-$next$ מצביע לערך האיבר הבא ברשימה (אם האיבר הבא הוא 7 $x.next = 7$) , ו-$head$ מצביע כן מצביע למקום בזכרון שמכיל את ראש הרשימה וזנב הרשימה.

31.1.1 מימוש הפעולות

- $Make - Set$ - ניצור רשימה חדשה ב-$\mathcal{O}\left(1\right)$.
- $Find - Set\left(x\right)$ - ניגש ל-$x.head$ ונקבל גישה לנציג של הרשימה של x.
- $Union\left(x_1, x_2\right)$ - נמצא את הרשימה של x_1 שהיא L_1, ואת של x_2 שהיא L_2. עד כה $\mathcal{O}\left(1\right)$. נאחד את L_1, L_2 כדי לעשות זאת נשים את L_2 לאחר L_1. נביט באיור הבא להמחשה :



איור 115 : המחשה לפעולה

נעדכן את ההתחלה ש-L_2 להיות ההתחלה של L_1 ב-$\mathcal{O}\left(1\right)$. לאחר מכן נעדכן את ה-$next$ של d להיות c ואת ה-$tail$ של הרשימה L_1 להיות b. אבל אז נצטרך לעדכן את כל האיברים ב-L_1 שיכילו את ה-$head$ של רשימה L_1 וזה במקרה הגרוע $\mathcal{O}\left(n\right)$.

יחד עם זאת, זה רק חסם עליון על פעולה אחת, נוכל לקוות שנקבל זמן ריצה טוב יותר ב-m פעולות.

דוגמה. נניח כי יש לנו קבוצה של n איברים מהם ניצור n קבוצות של יחידונים באמצעות $Make-Set$ ב-$\mathcal{O}(1)$:

$$\{x_1\},\{x_2\},\ldots,\{x_{n-1}\},\{x_n\}$$

נבצע $n-1$ פעולות של $Union$:

מספר הפעולות	הפעולה
1	$Union(x_n,x_{n-1})\to\{x_n,x_{n-1}\}$
2	$Union(x_{n-2},x_{n-1},x_n)$
3	$Union(x_{n-3},\ldots,x_n)$
	$\vdots$
$n-1$	$Union(x_1,\ldots,x_n)$

וקיבלנו סך הכל $1+2+\ldots+n-1=\frac{n(n-1)}{2}=\mathcal{O}(n^2)$ כלומר מצאנו סדרה של n פעולות שהיא עדיין בסיבוכיות $\mathcal{O}(n^2)$.

שאלה האם אפשר לשפר מבנה זה?

נבחין כי בכל הפעולות שעשינו בדוגמא הקודמת עדכנו את כל האיברים בקבוצה הגדולה, כאשר יכולנו לעדכן אך ורק את הרשימה הקצרה. כלומר נשמור שדה נוסף לכל רשימה והוא יהיה האורך שלה, אותו נעדכן בכל איחוד של רשימה להיות סכום האורכים ב-$\mathcal{O}(1)$.

למרות שעדיין מתקיים כי זמן הריצה של $Union$ הוא $\Omega(n)$, אם נבצע את הפעולות שעשינו קודם נקבל $\mathcal{O}(n)$ ולא $\mathcal{O}(n^2)$.

יתר על כן, ניתן להראות שבסדרה של m פעולות, נקבל שהעלות הכוללת היא $\mathcal{O}(m+n\log n)$ ולא $\mathcal{O}(n\cdot m)$. נוכיח זאת!

הרעיון הוא שאם נתון איבר x, אזי אם עדכנו את x הוא אוחד עם רשימה באורך גדול יותר מהרשימה שלו ולכן אורך הרשימה שלו מוכפל לכל הפחות ומספר ההכפלות הוא לכל היותר $\log n$ ולכן האיחוד הכולל הוא $n\log n$ וזה עבור מעבר על m איברים לכן $m+n\log n$ עבור כל התהליך.

טענה. לאחר m שאילתות על קבוצה בת n איברים נקבל זמן ריצה $\mathcal{O}(m+n\log n)$.

הוכחה: עבור הפעולות $Make-set, Find-set$ נקבל סך הכל $\mathcal{O}(1)$ ולכן עבור m פעמים נקבל $\mathcal{O}(m)$ כרצוי. אם כך מספיק להוכיח את הטענה עבור $Union$.

נבחין כי יש לנו לכל היותר $n-1$ פעולות איחוד. כל פעולותיה של $Union$ הן $\mathcal{O}(1)$ מלבד העדכון של $x.head$ עבור האיברים ברשימה החדשה. מכאן מספיק להוכיח שמספר העדכונים של $x.head$ הוא חסום על ידי $n\log n$.

יהי $x \in X$. נוכיח כי מספק העדכונים של $x.head$ הוא לכל היותר $\log n$, שכן מכאן נסיק כי עבור n איברים מספר העדכונים הוא $n \log n$.

אם כך, נבחין כי בכל עדכון של x, נובע שרשימה שלו מאוחדת עם רשימה באורך גדול יותר, ולכן גודל הקבוצה החדשה הוא לכל הפחות הכפלה של גודל הקבוצה של x, לכן יתכן לכל היותר $\log n$ עדכונים שלו, כי אז קיבלנו קבוצה בגודל לכל הפחות $2^{\log n}=n$ כלומר לא יתכנו יותר עדכונים. נבחין כי יתכן כי $x.head$ עודכן פחות מ-$\log n$ פעמים כי לעיתים האיחוד הוא עם קבוצה קטנה יותר, אבל בכל מקרה, לא יתכן כי עדכנו את הערך שלו יותר מ-$\log n$.

מכאן נסיק כי מספר העדכונים הכולל של כל האיברים הוא $\sum\limits_{x\in X}\log n=|X|\log n=n\log n$. ■

31.2 יער של קבוצות זרות $(Disjont-Set\ Forest)$

הייצוג יתבסס על עצים:

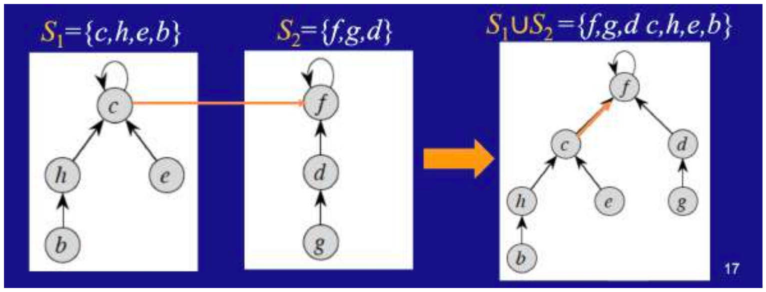
- כל קבוצה תיוצג על ידי עץ והאיברים בה יהיו קודקודי העץ.
- באופן טבעי, הנציג של הקבוצה יהיה שורש העץ.
- העצים לא בהכרח יהיו עצים בינאריים.

כדי לעשות זאת בצורה יעילה:

- לכל $x\in X$ נחזיק שדה $x.parent$ שיצביע על שורש העץ שהוא נציג הקבוצה.
  - אם x הוא שורש העץ, $x.parent=x$, כלומר הוא מצביע על עצמו. מכאן x הוא שורש העץ אם"ם $x.parent=x$.
  - אם x לא נמצא באף קבוצה נגדיר $x.parent=null$.

את הפעולות נבצע באופן הבא:

- $Make-Set$ - ניצור עץ חדש ב-$\mathcal{O}(1)$.
- $Find-Set(x)$ - נמצא את השורש של x ב-$\mathcal{O}(h)$ באמצעות $x.parent$, כאשר h הוא המרחק מ-x לשורש.
- $Union(x_1,x_2)$ - נמצא את השורשים של x_1,x_2 שנסמנם ב-S_1,S_2 בהתאמה. נהפוך את אחד העצים לתת עץ של העץ השני. זמן הריצה הוא $\mathcal{O}(2h+1)=\mathcal{O}(h)$ שכן יש שתי שאילתות של $Find-Set$ ועוד איחוד העצים באמצעות קישור עם פוינטר חדש ב-$\mathcal{O}(1)$.



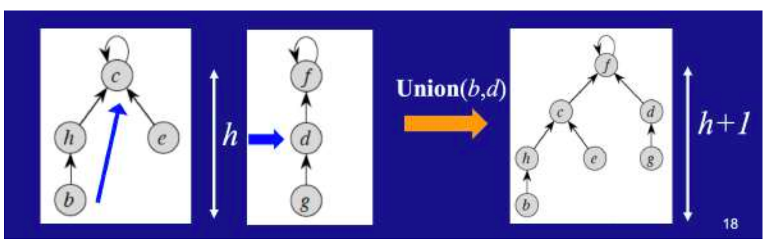
איור 116 : המחשה לפעולה

אם נאחד את העצים באופן נאיבי, יתכן שנקבל שזמן הריצה הוא $\mathcal{O}\left(n\right)$ בכל פעולה. למשל, אם נאחד את העצים ככה שתווצר רשימה מקושרת ארוכה, נקבל שחיפוש הוא $\mathcal{O}\left(n\right)$. לכן, נעדיף תמיד לחבר עץ קטן לעץ גדול.

נראה שתי דרכים כדי לשמור על העצים נמוכים.

31.2.1 **איחוד לפי גובה** $(Union - By - Rank)$

נמצא את שני השורשים של x_1, x_2. נאחד את השורש של העץ הנמוך היותר עם השורש של העץ הגבוה היותר. אם כך, לכל קודקוד x נשמור שדה נוסף $x.rank$ שישמור את גובה העץ שנמצא מתחתיו. עדכון השדה יהיה זול. אם אנו מאחדים עץ אם גובה קטן לעץ עם גובה גדול יותר הגובה לא משתנה, ולכן אין צורך לעדכן. אבל אם הגבהים זהים הגובה של העץ החדש יגדל באחד ולכן נצטרך לעדכן את ה-$rank$ של שורש העץ שאליו חיברנו את השורש השני. בשני המקרים זה $\mathcal{O}\left(1\right)$.



איור 117 : המחשה לאיחוד. במקרה זה נעדכן $f.rank = f.rank + 1$

בנוסף, נבחין כי מתקיים שהפעולה $root.rank + +$ מתבצעת אם"ם אנו מאחדים שני עצים באותו גובה.

טענה. הגובה המקסימלי של העץ על ידי שימוש באיחוד לפי גובה הוא $\mathcal{O}\left(\log n\right)$.

הוכחה: נוכיח באינדוקציה על מספר פעולות האיחוד ליצירת העץ, שמספר הקודקודים בעץ בגובה h בעל לכל היותר 2^h קודקודים.

בסיס: אם לא עשינו פעולות איחוד, יש לנו עץ עם קודקוד אחד וגובה 0 ולכן 2^0 קודקודים.

שלב: נניח את נכונות הטענה עבור עצים שמרכיבים עת בגובה h. נוכיח כי עבור עץ כנ"ל בגובה h יש לכל הפחות 2^h קודקודים.

נחלק למקרים.

אם העץ שלנו נוצר מאיחוד בין שני עצים בגבהים שונים, אז הגובה h של העץ החדש הוא אותו גובה של העץ הגדול ולכן מהנחת האינדוקציה מספר הקודקודים בעץ הוא לכל הפחות 2^h.

אם גובה עץ גדל, כלומר גובהי העצים שיצרו אותו זהים ולכן הגובה של העץ החדש הוא $h+1$ כאשר h הוא גובה העצים שיצרו אותו. לכן מהנחת האינדוקציה מספר הקודקודים בכל אחד מהעצים הוא לפחות 2^h ולכן ביחד זה $2^h + 2^h = 2^{h+1}$ לפחות. ■

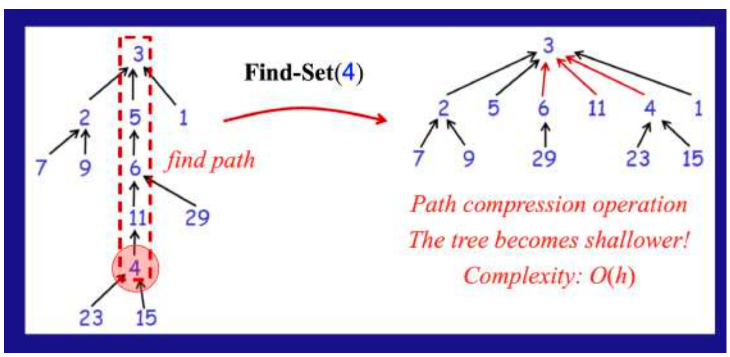
מסקנה. זמן הריצה המקסימלי של m פעולות מכל אחת מהפעולות הוא $\mathcal{O}(m \log n)$.

הוכחה: $Union$ הוא $\mathcal{O}(\log n)$ מהטענה, וכך גם $Find - Set$. $Make - Set$ היא $\mathcal{O}(1)$ ולכן עבור m פעולות נקבל $\mathcal{O}(m \log n)$. ■

31.2.2 דחיסת מסילות $(Path - Compression)$

במהלך פעולה של $Find - Set$, לכל קודקוד בנתיב החיפוש, נעדכן את המצביע שלו להצביע על שורש העץ.

נביט בדוגמא הבאה להמחשה:



איור 118: המחשה לעדכון, עבור הפעולה $Find - Set\ (4)$.

נבחין כי לא כל הילדים של 6 מצביעים על 3, כי לא כולם בנתיב החיפוש. לאחר הפעולה הקטנו משמעותית את גובה העץ.

31.2.3 מימוש הפעולות

שתי ההוריסטיקות שראינו נותנות ביחד את הפסאודו קודים הבאים:

Algorithm 43 $Make - Set\ (x)$

1: **Make-Set**(x)

2: $x.parent = x$

3: $x.rank = 0$

Algorithm 44 $Find - Set\,(x)$

```
Find-Set(x)
if x ≠ x.parent then x.parent ←Find-Set(x.parent)
freturn x.parent
```

Algorithm 45 $Link\,(x, y)$

```
Link(x, y)
if x.rank > y.rank then:
      y.parent ← x
if x.rank < y.rank then:
      x.parent ← y
else:
      x.parent ← y
      y.rank ← y.rank + 1
```

Algorithm 46 $Union\,(x, y)$

```
Union(x, y)
Link(Find − Set (x) , Find − Set (y))
```

נותר לנו רק לחשב את סיבוכיות זמן הריצה של האלגוריתמים.

משפט. באמצעות שימוש בדחיסת מסלולים, זמן הריצה הגרוע ביותר של m פעולות על קבוצה התחלתית בגודל n, נקבל $\mathcal{O}\left(m \log n\right)$.

הוכחה: (ראו בספר). ■

שאלה מה יקרה אם נשתמש בשתי השיטות ביחד?

משפט. על ידי ייצוג של יער קבוצות זרות עם דחיסת מסילות ואיחוד על ידי גובה, סדרה של m פעולות על קבוצה התחלתית בגודל n היא בעלת זמן ריצה במקרה הגרוע ביותר של $\mathcal{O}\left(m\cdot\alpha\left(n\right)\right)$ כאשר $\alpha\left(n\right)$ היא הפונקציה ההפוכה של פונקציית $Ackerman$.

- כאשר $\alpha\left(n\right)\leq 4$ לכל $n\leq 2^{2^{2^{65536}}}-3$ וניתן להחשיב אותה כקבוע.
- מתקיים כי $\alpha\left(n,m\right)=\min\left\{i\geq 1:A\left(i,\left\lfloor\frac{m}{n}\right\rfloor\right)\geq\log_2 n\right\}$ כאשר

$$A\left(n,m\right)=\begin{cases}n+1 & m=0\\ A\left(m-1,1\right) & m>0\land n=0\\ A\left(m-1,A\left(m,n-1\right)\right) & m>0\land n>0\end{cases}$$

הוכחה: ההוכחה מורכבת ונראה אותה בקורסים מתקדמים יותר. בספר הקורס ניתן לראות ורסיה קלה יותר. ■

31.3 זמן ריצה

עתה נרצה לנתח את זמני הריצה.

- במציאת רכיבי הקשירות החזקים, מתקיים שהסיבוכיות היא

$$\mathcal{O}\left(\left|V\right|+f\left(\left|V\right|,3\left|E\right|+\left|V\right|\right)\right)$$

שזה סך הכל

$$\mathcal{O}\left(\left|V\right|+\left|E\right|\cdot\alpha\left(\left|V\right|\right)\right)$$

שזה לינארי עבור כל מטרה פרקטית.

- באלגוריתם של קרוסקל, מתקיים שהסיבוכיות היא

$$\mathcal{O}\left(\left|E\right|\log\left|E\right|+\left|V\right|+f\left(\left|V\right|,3\left|E\right|+\left|V\right|\right)\right)$$

שזה סך הכל

$$\mathcal{O}\left(\left|E\right|\cdot\log\left|V\right|\right)$$

לינארי עבור כל מטרה פרקטית.

31.4 סיכום

- עבור סדרה של m פעולות המורכבת מ-$Make-Set, Union, Find-Set$. זמן הריצה הגרוע ביותר הוא:
  - מימוש באמצעות רשימות מקושרות: $\mathcal{O}\left(m+n\log n\right)$. אם $m>>n$ נקבל שזה $\mathcal{O}\left(m\right)$, לכן במקרה זה נעדיף להשתמש ברשימות מקושרות. במקרים אחרים נשתמש בשיטות הבאות:
  - מימוש באמצעות יער קבוצות זרות עם הוריסטיקה אחת - $\mathcal{O}\left(m\log n\right)$.
  - מימוש באמצעות יער קבוצות זרות עם שתי ההוריסטיקות של דחיסת מסילות ואיחוד על פי גובה - $\mathcal{O}\left(m\cdot\alpha\left(n\right)\right)$ כאשר $\alpha\left(n\right)$ היא $\mathcal{O}\left(1\right)$ פרקטית.
- המבנה שימושי עבור שני האלגוריתמים לגרפים:
  - $MST-Kruskal$ - $\mathcal{O}\left(|V|+|E|\log|V|\right)$.
  - $Connected-Components$ - $\mathcal{O}\left(|V|+|E|\,\alpha\left(|V|\right)\right)$.

הערה. בזאת סיימנו את חומר הקורס. בשיעורים הבאים נדבר על המבחן.