מבני נתונים - פרופ' ליאו יוסקוביץ - סמסטר א' 2021

מסכם: יחיאל מרצבך

תוכן העניינים

הקדמה כללית לקורס

שיעור מס' 1: **יום רביעי** 21.10.20

מבני נתונים ואלגוריתמים.

בקורס זה נעסוק בעיקר באספקט התיאורטי של האלגוריתמים. מדוע קוראים לקורס "מבני נתונים"? בהמשך נבחין כי למעשה שני המושגים נעים במקביל. בעקבות כך, בלתי אפשר לנתק את שני המושגים אחד מהשני.

סוגי מידע אבסטרקטיים

סוגי מידע אבסטרקטי, או בכינויו ADT– Abstract Data Types הוא למעשה דרך לאחסון מידע עם סט מסוים של פעולות.

מבנה נתונים הוא למעשה מימוש של דבר זה. בקורס נלמד כיצד להוכיח דברים הקשורים למבני הנתונים וכמה זמן ומקום הם דורשים.

בקצרה, נוכל לומר כי נתעסק בעיקר בנושאים המרכזיים במדעי המחשב: חיפוש, מיון, אינדקסים ועוד. נרצה למשל להבחין בין סוגי אלגוריתמים שונים - ולשאול האם האחד טוב מהשני ובאילו נסיבות.

אלגוריתמים ופיתרון.

כאשר יש לנו בעיה חישובית שאנו באים להתמודד איתה, נרצה לשאול מספר שאלות:

- האם תמיד יש פיתרון (אלגוריתם) לבעיה?
  - לא תמיד! כך למשל, ניתן להוכיח בצורה פורמלית כי אין אלגוריתם ל"בעיית העצירה". כלומר, לא קיימת דרך לבדוק האם האלגוריתם עוצר.
- האם תמיד קיים פיתרון יעיל לפתור את הבעיה ?
  - לא תמיד! למשל, בעיית האריזה.
- האם האלגוריתם שמצאנו הוא הטוב ביותר?
  - לא בהכרח! למשל, אם נמצא מיון ב־ $O\left(n^2\right)$, זהו לא זמן הריצה היעיל ביותר.
- מהו האלגוריתם הטוב ביותר שנוכל לפתח?
  - נעיר כי לגבי מיון למשל, ניתן להוכיח כי אין אלגוריתם הקטן מזמן ריצה של $\Omega\left(n\log n\right)$ ומזיכרון של $\Omega\left(n\right)$.

בעיות קשות וקלות.

אם כך, במסגרת פיתרון "הבעיות", נוכל לדבר מחד על 'בעיות קשות', או 'בעיות קלות' מאידך. לדוגמא, 'בעיית מיון' נחשבת לבעייה קלה, ואילו 'בעיית האריזה' עליה דיברנו לעיל נחשבת לבעייה קשה, כיוון שבסופו של דבר הפיתרון שלה הוא בזמן ריצה אקספוננציאלי. [1]

אם נרצה, נוכל להסתכל על כך בתור 'מחלקות שקילות' של סיבוכיות, דהיינו כל הבעיות שיש להן את אותה סיבוכיות:

(1) $\Omega\left(n\right)$ - זמן ליניארי.

(2) $\Omega\left(n\log n\right)$.

(3) $\Omega\left(n^2\right)$ - זמן ריבועי.

(4) $\Omega\left(n^k\right)$ - זמן פולינימואלי.

(5) $\Omega\left(2^n\right)$ - זמן אקספוננציאלי.

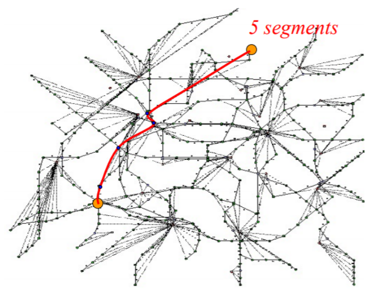
(6) $\Omega\left(2^{2^n}\right)$ - זמן אקספוננציאלי כפול.

[1] במהלך הקורס נדון בכלים הפורמליים לאבחן מה הופך בעייה מסוימת לבעיה קשה יותר.

(7) בעיות שלא ניתנות לפיתרון מבחינה חישובית.

חישוב מרחק בדרך הקצרה ביותר

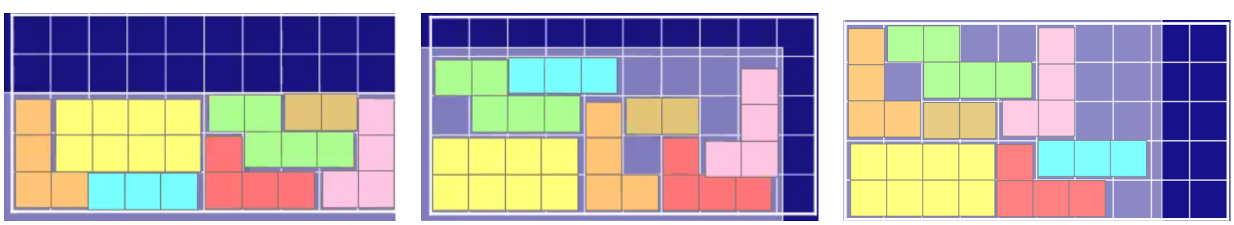
נתבונן בבעיה פשוטה: אלגוריתם המוצא את הדרך הקצרה ביותר מנקודה x לנקודה y:



ראשית, נצטרך למצוא דרך לייצוג הנקודות - אפשר לייצג כל צומת באמצעות קודקוד בגרף. באמצעות מספר הצמתים נוכל למצוא את מספר מסלולים. כעת, נרצה למצוא אלגוריתם המוצא את המסלול הקצר ביותר. גודל הגרף הרצוי יהיה מספר ה'סגמנטים' - הסכום של המרחק בין הסגמנטים האלו, זה הפיתרון.
נרצה גם לשאול - כמה זמן ייקח לנו לחשב מסלול כזה? נראה שדבר זה לוקח בהמשך $O\left(n\right)$ לפי n הסגמנטים.

בעיית סידור הקופסאות

נתבונן כעת בבעיה עליה דיברנו קודם לכן, בעית סידור הקופסאות. בהינתן קופסאות בגדלים שונים, נרצה לסדר אותן בתוך ריבוע, עם כמה שפחות ריבועים ריקים.
נוכל להבחין כי ישנם מספר דרכים לפיתרון, כשהפיתרון השמאלי הוא למעשה הפיתרון הטוב ביותר:



אם כך, נרצה לשאול, כמה זמן ייקח לנו למצוא את זמן האריזה האופטימלי? מהו סדר האלגוריתם:

- בדיקה שכל הפעולות חוקיות.
- חישוב השטח שכיסינו.
- שמירת השטח הטוב ביותר.

עדיין לא ברור לנו כמה זמן ייקח לנו להגיע לפיתרון היעיל ביותר.
בעקבות העובדה שניתן להציב את הקופסא בארבע כיוונים בערך, נוכל לחשוב כי המקרה הגרוע ביותר הוא 4^n, כש-n זהו מספר הקופסאות. ייתכנו מקרים בהם הקופסאות באותו גודל ובאותה צורה, ואז זמן הריצה יהיה קטן יותר.
אך אמנם, ניתן להוכיח כי עבור הרכב מסוים של קופסאות (שאינם באותו גודל ובאותו צורה), אין זמן ריצה טוב יותר מאשר המקרה הגרוע ביותר (Worst case).

חישוביות, יעילות וזמני ריצה

הקדמה

למעשה, ייתכנו מספרי מקרי זמן ריצה:

(1) המקרה הטוב ביותר - Best case - הקלט שיגרום לאלגוריתם הכי פחות עבודה.
(2) המקרה הגרוע ביותר - Worst case - הקלט שיגרום לאלגוריתם הכי הרבה עבודה.
(3) המקרה הממוצע - Average case - הקלט שיגרום לאלגוריתם זמן ריצה ממוצע. בדרך כלל נבחן את האלגוריתים לפי הזמן הממוצע.

בהכללה, נרצה לשאול, לכל קלט אפשרי, מהו ה'חסם העליון' - דהיינו, מהו זמן הריצה היעיל ביותר, ו'חסם תחתון'- - מהו זמן הריצה הקצר ביותר.

יעילות: אלגוריתמי מיון

בהינתן מערך של n מספרים, נוכל למיין אותו במספר צורות.

מיון בועות - BubbleSort

ניקח כל פעם שני איברים מהמערך, ונבעבע אותם, עד שהאיבר הכי גדול יהיה ב'צד' ימין של המערך:

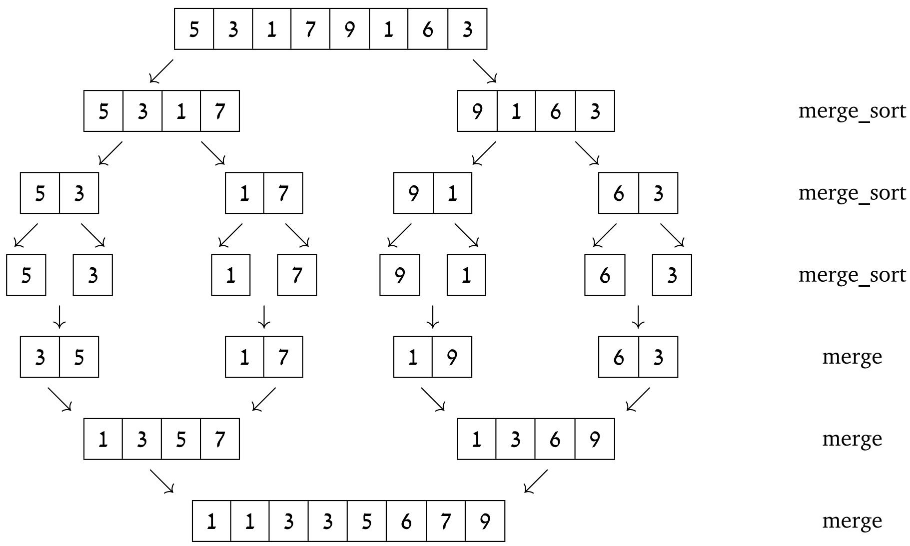
84	55	61	10	18	35	22	97	47	start
55	84	61	10	18	35	22	97	47	1^{st} itration
55	61	84	10	18	35	22	97	47	2^{nd} itration
...									...
55	61	10	18	35	22	84	47	97	n^{th} itration

זהו הסיבוב הראשון. כעת נבחין מה קורה בכל הסיבובים:

55	61	10	18	35	22	84	47	97	1^{st} pass
55	10	18	35	22	47	61	84	97	2^{nd} pass
10	18	35	22	55	47	61	84	97	3^{rd} pass
...									...
10	18	22	35	47	55	61	84	97	n^{th} pass

מיון מיזוג - MergeSort

נפצל בכל פעם את המערך, נמיין את החלק הרלוונטי, ונאחד אותו. כך, למשל:



מסתבר שהחישוביות בשני המקרים שונה. מספר הפעמים שנצטרך לפרק את הרשימה יהיה $\log n$.

לסיכום:

	SPACE	Time		
		Best	Worst	Average
Bubble Sort	n	n: one pass	n^2: n passes	$\frac{n^2}{2}$:$\frac{n^2}{2}$ passes
Merge Sort	$n\log n$	$n\log n$	$n\log n$	$n\log n$

נוכל לחשוב על מערך מיון נוסף - InsertionSort.

מיון הכנסה - InsertionSort

כעת, נחשוב על הפסאודו קוד שעושה את הפעולה:[2]

אלגוריתם 1 מיון הכנסה

---

(1) לכל $j \leftarrow 2$ עד לאורך המערך. (c1)

(א) $key \leftarrow A[j]$ (c2)

(ב) $i \leftarrow j-1$ (c4)

(ג) כל עוד $i > 0$ וגם $A[i] > key$ (c5)

(i) $A[i+1] \leftarrow A[i]$ (c6)

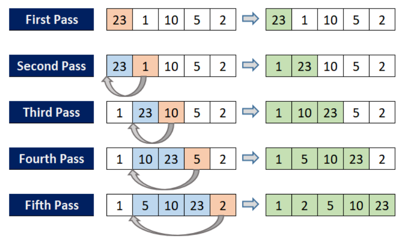
(ii) $i \leftarrow i-1$ (c7)

(ד) $A[i+1] \leftarrow key$ (c8)

---

[2] משמעות הסימון $a \leftarrow b$ בפסאודו קוד היא השמה של המשתנה a לתוך המשתנה b.

מבחינה ציורית, זה נראה כך:



אם נרצה, נוכל לסכם כי זמן הריצה יהיה סכום כל הפעולות הללו.

אמנם, אם שלב מסוים קורה c_i פעמים, כשבכל שלב מתרחשת t_i פעולות, נראה כי $T(n)=\sum\limits_{i=1}^{k} c_i t_i$, או אם נפתח זאת:

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

נוכל להבחין כי הזמן משתנה לפי הקלט:

במצב הטוב ביותר - BestCase - הלולאה עוצרת בבדיקה הראשונה (צעד 5 לוקח $\sum_{j=2}^{n} t_j=n$ פעולות) ואז סך הכל נקבל:

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

כלומר, בסך הכל מדובר בזמן ריצה ליניארי - $T(n)=an+b$.

במקרה הגרוע ביותר - WorstCase - צעד 5 לוקח $\sum\limits_{j=2}^{n} j=n(n+1)/2-1$ ואילו צעדים $6-7$ (שתמיד מתרחשים) לוקחים $\sum_{j=2}^{n}(j-1)=n(n-1)/2$, לכן, בסך הכל נקבל:

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

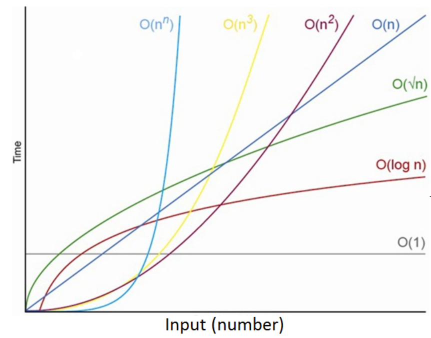
בסופו של דבר, קיבלנו ביטוי ריבועי מהצורה $T(n)=an^2+bn+c$.

נוכל להשוות כעת בין שתי הפונקציות האלו, ודבר זה נקרא **'ניתוח אסימפטוטי'**.

בהינתן ניתוח של ביטוי, נוכל להגדיר $f(n)=T(n)$.

הפונקציה יכולה לכלול קבועים, אך כיוון ש-n גדל, אין חשיבות לקבועים. למעשה, ההגדרה של 'ניתוח אסימפטוטי', היא מה מתרחש עבור n מספיק גדול.

נוכל לראות זאת גם בגרף הבא:



אם נסכם, הצורה הטובה ביותר לתיאור אבסטרקטי של אלגוריתם, היא לדבר על 'ההתנהגות האסימפטוטית שלו'. נרצה לתאר גם את 'זמן הריצה', וגם את 'המקום' הנדרש. בנוסף, נרצה לדבר גם על 'המקרה הגרוע ביותר', 'המקרה הטוב ביותר' ו'המקרה הממוצע'.

סיבוכיות זמן וזיכרון

הגדרות

חישוביות זמן ריצה: כמות הפעולות הנדרשות לביצוע הפעולה.

חישוביות מקום: מספר התאים הנדרשים לביצוע הפעולה.

קצב הגידול: כמה מהר הפונקציות גדלות.

כעת, בהינתן אלגוריתם המתואר בשפה אבסטרקטית, נרצה להבין את ההתנהגות שלו.

סימונים

חסם עליון (המקרה הגרוע ביותר) - מקסימום $O\left(f\left(n\right)\right)$ פעולות

חסם תחתון - (המקרה הטוב ביותר) - מינימום $\Omega\left(f\left(n\right)\right)$ פעולות.

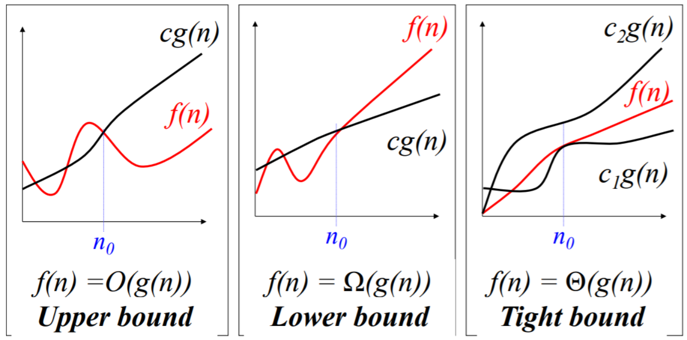
חסם הדוק - בכל המקרים נצטרך $\Theta\left(f\left(n\right)\right)$ פעולות.

נקודות מתמטיות.

הגדרות

- חסם עליון - בהינתן שתי פונקציות $f\left(n\right), g(n) : \mathbb{N} \to \mathbb{R}_+$, נאמר כי $f\left(n\right) = O\left(g\left(n\right)\right)$, כאשר קיים $c > 0$ ו־$n_0 > 1$ כך ש־ $f\left(n\right) \leq c\left(g\left(n\right)\right)$ לכל $n \geq n_0$.
- חסם תחתון - בהינתן שתי פונקציות $f\left(n\right), g(n) : \mathbb{N} \to \mathbb{R}_+$, נאמר כי $f\left(n\right) = \Omega\left(g\left(n\right)\right)$, כאשר קיים $c > 0$ ו־$n_0 > 1$ כך ש־ $f\left(n\right) \geq c\left(g\left(n\right)\right)$ לכל $n \geq n_0$.
- חסם הדוק - בהינתן שתי פונקציות $f\left(n\right), g(n) : \mathbb{N} \to \mathbb{R}_+$, נאמר כי $f\left(n\right) = \Theta\left(g\left(n\right)\right)$, כאשר קיימים $c_1, c_2 > 0$ ו־$n_0 > 1$ כך ש־ $0 \leq c_1\left(g\left(n\right)\right) \leq f\left(n\right) \leq c_2\left(g\left(n\right)\right)$ לכל $n \geq n_0$.

ולסיכום:



טענה

תהיינה $f(n), g(n) : \mathbb{N} \to \mathbb{R}_+$, נאמר כי $f(n) = \Theta(g(n))$ אם ורק אם $f(n) = O(g(n))$ וגם $f(n) = \Omega(g(n))$.

הוכחה

בתרגול.

טענה

תהיינה $f(n), g(n), h(n) : \mathbb{N} \to \mathbb{R}_+$. אזי החסמים Ω, Θ, O מקיימים את התכונות הבאות:

(1) רפלקסיביות:

(א) $f(n) = O(f(n))$.

(ב) $f(n) = \Omega(f(n))$.

(ג) $f(n) = \Theta(f(n))$

(2) סימטריות:

(א) $f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.

(3) טרנזיטיביות:

(א) $f(n) = O(g(n)) \quad \wedge \quad g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$.

(ב) $f(n) = \Omega(g(n)) \quad \wedge \quad g(n) = \Omega(h(n)) \Rightarrow f(n) = \Omega(h(n))$.

(ג) $f(n) = \Theta(g(n)) \quad \wedge \quad g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$.

הוכחה

בתרגול ובתרגיל.

תכונות של חסמים

תהיינה $f(n), g(n) : \mathbb{N} \to \mathbb{R}_+$ פונקציות. אזי החסמים Ω, Θ, O מקיימות את התכונות הבאות:

- $O(O(f(n))) = O(f(n))$.
- $O(f(n) + g(n)) = O(f(n)) + O(g(n))$.
- $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$
- $O(\log(n)) = O(\lg(n))$.[3]
- $O\left(\sum_{i=1}^{k} a_i n^i\right) = O\left(n^k\right)$.

[3] הביטוי lg הינו $\log_2$.

- $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$.
- $n! = O\left(n^{n+0.5}\right)$.
- $O\left(\log n!\right) = O\left(n \lg n\right)$.

הוכחה

בתרגול ובתרגיל.

לסיכום, בכל אלגוריתם של פסאודו קוד, נרצה לבדוק את ההתנהגות האסימפטוטית על מנת לגזור ממנה את הסיבוכיות שלה. הוכחת החסם ההדוק היא הקשה ביותר בדרך כלל.

נוסחאות נסיגה

שיעור מס' 2:
יום רביעי
28.10.20

ניתוח סיבוכיות של אלגוריתם.

ראינו קודם לכן כי נוכל לנתח סיבוכיות של אלגוריתם, באמצעות סכימת כל הפעולות - דהיינו $T(n) = \sum c_i t_i$. (באמצעות לולאות ניתן לחשב כמה פעולות מתרחשות).

אמנם, כעת נלמד שיטה נוספת - באמצעות חישוב רקורסיבי. דהיינו, למעשה נחשב את סך כל הפעולות בפונקציות רקורסיבית. נשים לב שבשיטה זו מדובר על הערכה מסוימת, ולא על חישוב מדויק.[4]

הנוסחה הרקורסיבית של מציאת מקסימום במערך הינה למשל, הינה :

$$\begin{cases} T(n-1)+c & n>1 \\ 1 & 1 \end{cases}.$$

נוכל לראות שבמקרה זה החישוב הינו $T(n) = O(n)$.

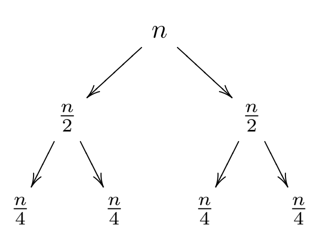
נבחין כי חייבים להתקיים מספר תנאים:

- n מספר מספיק גדול.
- במקרה של n מספיק קטן נחשב כי $T(1) = \Theta(1)$ - דהיינו כיוון שמדובר ב-n קטן, ישנו **מספר קבוע** של פעולות.
- עלינו לבחור n לפני התנאים במקרה הרצוי (למשל, ניתן לבחור n זוגי או n שהינו חזקה של 2).[5]

כמו כן, נצטרך:

(1) לכמה חלקים ניתחנו את הבעיה (לפעמים ניתוח בעיה לחלקים קטנים יותר הופך אותה לפשוטה יותר).
(2) מהו גודל תת הבעיה שאליה חילקנו.
(3) כיצד נוכל לחבר את תת הבעיות לבעיה הגדולה.

זהו מבנה עץ הרקורסיה הקלאסי:



דוגמאות

[4] בסופו של דבר אנחנו משמיטים את הקבועים ולכן אין משמעות לכך.

[5] יחד עם זאת, שימו לב לקבועים! $T\left(\frac{n}{2}\right) \neq T(n)$!

נתבונן בכמה דוגמאות פשוטות:

(1) מציאת $n!$: כפילת n ב-$(n-1)!$ ולכן נקבל כי:

$$T\left(n\right)=T\left(n-1\right)+O\left(1\right)\rightarrow O\left(n\right)$$

(2) נוסחת פיבונצ'י המוגדרת על ידי- $fibonacci\left(n-1\right)+fibonacci\left(n-2\right)$:

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

(3) חיפוש סקוונציאלי: חיפוש המקסימום או המינימום במערך. נוסחת הרקורסיה והחישוביות הינה:

$$T\left(n\right)=T\left(n-1\right)+O\left(1\right)\rightarrow O\left(n\right)$$

(4) מיון הכנסה (אותו ראינו בפרק הקודם). כעת נבחין כי בשונה מהבעיות הדומות הקודמות, נצטרך לחשב היכן להכניס את האיבר בכל פעם, ובסך הכל נקבל:

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

דהיינו, ההבדל הינו 'בחזרה מהרקורסיה'.

(5) חיפוש בינארי בעץ מאוזן: בחירה האם לבדוק בכל אחד מצדדי העץ. נצטרך לחפש רק $\frac{n}{2}$ איברים, ולכן נקבל:

$$T\left(n\right)=T\left(\frac{n}{2}\right)+O\left(1\right)\rightarrow O\left(\log n\right)$$

(6) מעבר על כל הקודקדים בעץ: בשונה מהחיפוש הקודם, נצטרך לבדוק גם בצד ימין וגם בצד שמאל. כלומר, נעשה את הפעולה הקודמת פעמיים:

$$T\left(n\right)=2T\left(\frac{n}{2}\right)+O\left(1\right)\rightarrow O\left(n\right)$$

(7) מיון מיזוג (אותו ראינו בפעם קודמת): מחד, אנחנו עוברים על כל האיברים בעץ (כמו החיפוש הקודם), מצד שני, דבר זה ייקח לנו $O\left(n\right)$ זמן (עד שנגיע ל-1). בהמשך, במיזוג, נקבל תוצאה דומה (ולכן זהו פעמים אותה העבודה), ולסיכום:

$$T\left(n\right)=2T\left(\frac{n}{2}\right)+O\left(n\right)\rightarrow O\left(n\log n\right)$$

פיתרון סיבוכיות רקורסיבית

ישנן שלוש צורות לפיתרון בעיות רקורסיביות:

(1) הצבה ושימוש באמצעים מתמטיים.
(2) הוכחה באמצעות עץ רקורסיבי.
(3) משפט האב - Master theroem - הנוסחה הינה: $T(n) = aT(n/b) + f(n)$.

שיטת ההצבה

נרצה להוכיח כי הפיתרון של הביטוי $T(n) = 2T\left(\frac{n}{2}\right) + n$ הינו $O(n \log n)$ לכל $n \geq 2$.

הוכחה

מקרה הבסיס, כאשר $n = 2$:

מתקיים:

$$T(2) \leq c(2 \log_2 2)$$

ולכן לכל $c \geq 1$ נקבל כי $T(2) \geq 2$.

צעד האינדוקציה:

נניח כי הטענה נכונה עבור $\frac{n}{2}$, ולכן לפי הנחת האינדוקציה נקבל:

$$T\left(\frac{n}{2}\right) \leq 2\left(c\frac{n}{2}\log\left(\frac{n}{2}\right)\right)$$

וכעת נציב:

$$\begin{aligned}
T(n) &\overset{\text{אינדקוציה}}{\leq} 2\left(\left(c \cdot \frac{n}{2}\log\left(\frac{n}{2}\right)\right)\right) + n \\
&\overset{\text{כללי לוגים}}{\leq} c \cdot n \log\left(\frac{n}{2}\right) + n \\
&\overset{\text{כללי לוגים}}{\leq} c \cdot n \log n - c \cdot n \log 2 + n \\
&\leq c \cdot n \log n - c \cdot n + n \\
&\leq c \cdot n \log n
\end{aligned}$$

דהיינו, הטענה נכונה $\forall c \geq 1$. כלומר, הנחנו עבור $n > k$ והוכחנו עבור n ולכן הטענה נכונה לכל n טבעי.

אמנם, נשים לב כי 'ניחשנו' לאיזה חסם נגיע.
על מנת לעזור לנו למצוא זאת, נוכל להיעזר בשלבים הבאים:

(1) נמצא מספר איברים ברקורסיה.
(2) ננסה למצוא נוסחה.
(3) נפתח זאת בצורה רקורסיבית ונוכיח באמצעות אינדוקציה.

נתבונן למשל בדוגמא הבא. נתחיל להציב את המספרים הבאים:

$$\begin{aligned}
T(n) &= T(n-1)+n \\
T(n-1) &= T(n-2)+(n-1) \\
T(n-2) &= T(n-3)+(n-2) \\
T(n-3) &= T(n-4)+(n-3)
\end{aligned}$$

כעת, נוכל להציב בצורה כללית:

$$\begin{aligned}
T(n) &= T(n-1)+n \\
&= [T(n-2)+(n-1)]+n \\
&= [[T(n-3)+(n-2)]+(n-1)]+n \\
&= [[[T(n-4)+(n-3)]+(n-2)]+(n-1)]+n \\
&= T(n-k)+\sum_{i=1}^{k}(n-i+1)
\end{aligned}$$

לאחר שהגענו לנוסחה המסוימת, נפתח את הצורה הרקורסיבית, ונקבל באמצעות נוסחה חשבונית:

$$T\left(n\right)=T\left(n-k\right)+nk-\frac{\left(\left(k-1\right)k\right)}{2}$$

בסוף הרקורסיה, נקבל כי $k=n-1$ וגם $T\left(1\right)=1$. כלומר, נקבל בסופו של דבר:

$$\begin{aligned}
T(n) &= 1+n^2-n-n^2/2-3n/2-1 \\
&= n^2/2-n/2 \\
&= O\left(n^2\right)
\end{aligned}$$

מצאנו כי הפיתרון $O\left(n^2\right)$ מתאים לנוסחה זאת.

משפט האב - Master theorem

יהיו $a,b,c\in\mathbb{R}$ $1\le$. ויהיו $f\left(n\right):\mathbb{N}\to\mathbb{R}_+$, וגם $T\left(n\right)$ פונקציה המוגדרת על $T\left(n\right)=aT\left(\frac{n}{b}\right)+n^c$ (a הינו מספר החלוקות, c הינה כמות העבודה בכל חלוקה, ו-b הוא הגודל שאליו נחלק).

אזי:

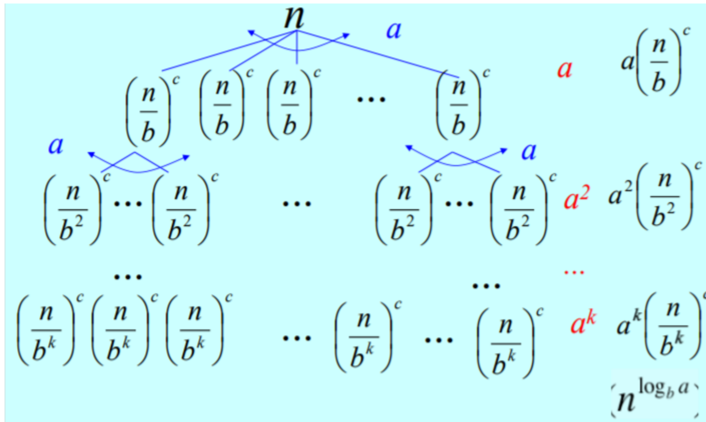
(1) $T\left(n\right)=\Theta\left(n^c\right)$ כאשר $\frac{a}{b^c}<1$ (כלומר $\log_b a<c$).

(2) $T\left(n\right)=\Theta\left(n^c\log_b n\right)$ כאשר $\frac{a}{b^c}=1$ (כלומר $\log_b a=c$).

(3) $T\left(n\right)=\Theta\left(n^{\log_b a}\right)$ כאשר $\frac{a}{b^c}>1$ (כלומר $\log_b a>c$).

מוטיבציה למשפט

נתבונן בשרטוט הבא:



נחלק את הבעיה ל-a חלקים, כשכל אחד בגודל $\left(\frac{n}{b}\right)$, ומתבצעים בכל אחד $\left(\frac{n}{b}\right)^c$ פעולות. דבר זה גורר כי $k=\log_b n$ הינו העומק של הרקורסיה.

הוכחה

נוכיח זאת באמצעות שיטת ההצבה. נבחין קודם כל בדפוס:

$$\begin{aligned}
&T(n)=aT(n/b)+n^c\\
&T(n/b)=aT\left(n/b^2\right)+(n/b)^c\\
&T\left(n/b^2\right)=aT\left(n/b^3\right)+\left(n/b^2\right)^c\\
&T\left(n/b^4\right)=aT\left(n/b^5\right)+\left(n/b^4\right)^c
\end{aligned}$$

כעת, נוכל למעשה לפתח את נוסחת הסכום, כאשר $k=\log_b n$:

$$\begin{aligned}
T(n)=&aT(n/b)+n^c\\
=&a\left[aT\left(n/b^2\right)+(n/b)^c\right]+n^c\\
=&a\left[a\left[aT\left(n/b^3\right)+\left(n/b^2\right)^c\right]+(n/b)^c\right]+n^c\\
=&a^kT\left(n/b^k\right)+n^c\left[1+a(1/b)^c+a^2\left(1/b^2\right)^c+\right.\\
&\left.+\ldots+a^{k-1}\left(1/b^{k-1}\right)^c\right]=\\
&a^kT\left(n/b^k\right)+\sum_{i=0}^{k-1}a^i\left(\frac{n}{b^i}\right)^c
\end{aligned}$$

הוכחת משפט האב

ננסה לאבחן מה מתרחש בכל שלב ברקורסיה:

- בשלב k ישנם למעשה $a^k\left(\frac{n}{b^k}\right)^c$ תתי בעיות.
- ישנם $k=\log_b n$ שלבים.

ננסה לפשט את הביטוי

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

בשלב זה נוכל להחליף את הביטויים לביטויים נוחים יותר:

$$T(n) = a^k T\left(n/b^k\right) + \sum_{i=0}^{k-1} a^i \left(\frac{n}{b^i}\right)^c \overset{k=\log_b a,(*)}{\Rightarrow}$$

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

כמו כן, נבחין כי מספר הפעולות הינו:

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

כשהעומק מושפע מהביטוי $\left(\frac{a}{b^c}\right)$.

כעת נשים לב כי ביטוי זה תלוי בשלושת המקרים:

- כאשר $\frac{a}{b^c} < 1$ נקבל

(בעקבות העובדה כי עבור $|x| < 1$ מתקיים $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$) ,

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

נוכל להראות כי הביטוי הימני חסום על ידי קבוע, ולכן נקבל כי $T(n) = \Theta(n^c)$.

- כאשר $\frac{a}{b^c} = 1$ נקבל כי $\sum_{i=0}^{\log_b n-1}\left(\frac{a}{b^c}\right)^i = \log_b n$ ולכן קיבלנו כי $T(n) = \Theta(n^c \log_b n)$.
- כאשר $\frac{a}{b^c} > 1$ נקבל:

$$\sum_{i=0}^{\log_b n-1}\left(\tfrac{a}{b^c}\right)^i = \Theta\left(\left(\tfrac{a}{b^c}\right)^{\log_b n}\right)$$
$$n^c\Theta\left(\left(\tfrac{a}{b^c}\right)^{\log_b n}\right) = \Theta\left(n^c\left(\tfrac{a}{b^c}\right)^{\log_b n}\right)$$
$$n^c\left(\tfrac{a}{b^c}\right)^{\log_b n} = \tfrac{n^c}{b^{c\log_b n}} \cdot a^{\log_b n} = \tfrac{n^c}{n^c} \cdot n^{\log_b a} = n^{\log_b a}$$

ולכן קיבלנו כי $T(n) = \Theta\left(n^{\log_b a}\right)$.

דוגמא - שימוש במיון מיזוג

במקרה כזה, נקבל כי $a = 2, b = 2, c = 1$. ולכן נצטרך להשתמש במצב $\frac{1}{b^c} = 1$. ולכן, הסיבוכיות היא בפשטות: $T(n) = \Theta\left(n^{\log_2 2}\log n\right) \Rightarrow T(n) = \Theta(n\log n)$.

דוגמא נוספת

מצאו את נוסחת הנסיגה - $T(n) = 9T\left(\frac{n}{3}\right)+n$. קיבלנו כי $a = 9, b = 3, f(n) = n$ ולכן $n^{\log_b a} = n^{\log_3 9} = n^2$.

משפט האב המוכלל

תהי נוסחת הנסיגה עבור $a \geq 1, b > 1$ עבור פונקציית האיחוד $f(n)$:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

מתקיים:

(1) אם $f(n) = O\left(n^{(\log_b a)-\varepsilon}\right)$ עבור $\varepsilon > 0$ אזי $T(n) = \Theta\left(n^{\log_b a}\right)$.

(2) אם $f(n) = O\left(n^{\log_b a}\right)$ עבור $\varepsilon > 0$ אזי $T(n) = \Theta\left(n^{\log_b a}\log n\right)$.

(3) אם $f(n) = \Omega\left(n^{(\log_b a)+\varepsilon}\right)$ עבור $\varepsilon > 0$ ואם עבור קבוע $c < 1$ לכל $n > n_0$ מתקיים $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ אזי $T(n) = \Theta(f(n))$.

מיון על בסיס השוואות

שיעור מס' 3:
יום רביעי
04.11.20

הגדרה

בהינתן סדרה של n מספרים - $A = (a_1, \ldots, a_n)$
סידורן מחדש, $A' = (a'_1, \ldots, a'_n)$ כך ש- $a'_1 \leq \ldots \leq a'_n$ נקרא **מיון**.

מדוע נצטרך מיון?

מדובר בבעיה בסיסית במדעי המחשב, כשהרבה אלגוריתמים משתמשים במיון בתור שיטת מפתח. בנוסף, המיונים משתמשים בהמון טכניקות. במהלך השיעור נמצא אף חסם תחתון לכל סוגי המיון וכך נוכל לאפיין מראש הרבה בעיות המסתמכות על מיון.

קיימים שני סוגי מיון:

(1) מיון באמצעות השוואה:
- מיון מיזוג, מיון הכנסה, מיון בעבוע.
- מיון מהיר.
- נוכיח כי החסם התחתון הינו $T(n) = \Omega(n \log n)$.

(2) מיון ללא השוואה. שיטת מיון זה דורשת הנחות על הקלט, וזמן הריצה המינימלי שלה הינה $O(n)$ - קצב גידול ליניארי.

מיון מהיר - QuickSort

נתבונן באלגוריתם המיון הבא - QuickSort - מיון מהיר.

התכונה המיוחדת של מיון זה, הינו שהמיון מתבצע על **המערך עצמו**, ללא שימוש בזיכרון נוסף. דהיינו, ישנו **מספר קבוע** של תאים נוספים. כעת, נתאר את האלגוריתם ונוכיח את נכונותו ואת זמני הריצה שלו.

הרעיון של המיון הינו חלוקה לשני תתי מערכים. אם נרצה, נוכל להתבונן בבעיה הבעיה:

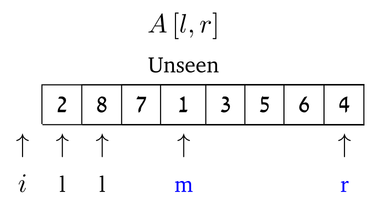
$A[l, r]$

Left				Right			
2	1	3	4	7	6	5	8
↑			↑				↑
l			m				r

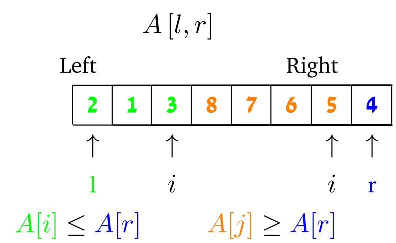
איזור שמאלי - בירוק, איזור ימני - בכתום, והציר של המערך - בכחול.

בחירת הציר תהיה כך - כל המספרים מימינו יהיו גדולים ממנו, וכל המספרים משמאלי יהיו קטנים ממנו.
החלוקה תתבצע למעשה בצורה רקורסיבית, אך צורת האלגוריתם תאפשר לנו למיין את המערך במקומו, ללא שימוש באיחוד.

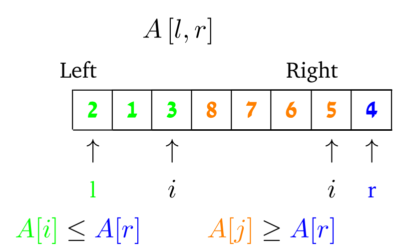
נוכל לראות דוגמה לחלוקה ומציאת הציר בכל פעם.
נניח שבחרנו בתור ציר את האיבר הימני:



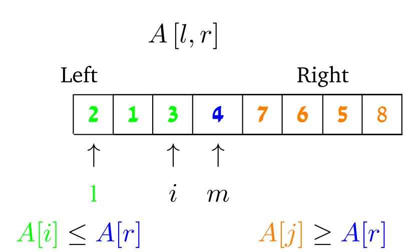
כעת, נרוץ על כל המערך, ונבדוק אילו איברים קטנים ממנו, ואילו איברים גדולים ממנו ונסדר אותם בהתאם.
דהיינו, אם איבר גדול ממנו, נחליף אותו.
במצב כזה, החלפנו בין $1 \leftrightarrow 8$ ובין $3 \leftrightarrow 7$ וכעת נקבל:



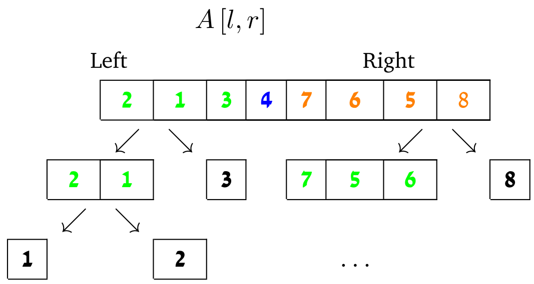
כעת, קיבלנו למעשה ציר בצד ימין, ואיברים אותם נוכל למיין בפני עצמם:



לבסוף, נחליף את האיבר האחרון ואת האיבר הראשון בסדרה ששמאלית לו $(i+1)$, כלומר $8 \leftrightarrow 4$:



וכעת, נפעל רקורסיבית על כל אחד מהאיברים, ונקבל:



וכן הלאה.

אלגוריתם 2 מיון מהיר

- פונקצית QuickSort
  (1) אם $l < r$:
    (א) $m \to partition\,(A, l, r)$
    (ב) תפעיל את QuickSort על $(A, l, m-1)$
    (ג) תפעיל את QuickSort על $(A, m{+}1, r)$
  (2) אם $l = r$:
    (א) אל תעשה כלום.
- מציאת הציר - Partition:
  (1) $i \to l - 1$
  (2) מ-l $j \to$ עד $r - 1$
    (א) אם $A\,[j] \leq A\,[r]$
      (i) $i \to i + 1$
      (ii) תחליף את $A\,[i]$ ואת $A\,[j]$
  (3) תחליף את $A\,[i+1]$ ואת $A\,[r]$
  (4) תחזיר את $i + 1$

מהי הסיבוכיות?

(1) <u>מקרה לא מאוזן בכלל</u> - אין שום איזון בין האיברים, ולכן נצטרך לפרק את הבעיה ל-$n-1$ בעיות.
(2) <u>מקרה אידיאלי</u> - החלוקה היא תמיד באמצע, ולכן נצטרך לחלק למקסימום $\frac{n}{2}$ בעיות.
(3) <u>מקרה מאוזן</u> - החלוקה תהיה תמיד במקום כלשהוא באמצע, ולכן נצטרך לחלק ל-$n-k$ ו-k תתי בעיות.

המקרה הלא המאוזן

בצורה הכי לא מאוזנת, יתבצעו למעשה $T\,(n) = \underbrace{T\,(0)}_{\text{בחירת הציר}} + T\,(n-1) + \Theta\,(n)$.

בצורה כזאת, נקבל כי הסיבוכיות הינה (כפי שראינו כבר):

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

במקרה האידיאלי

בצורה המושלמת, נקבל למעשה עץ בינארי:

$$T\,(n) = T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + \Theta\,(n)$$

ולכן, לפי מה שראינו בשיעור הקודם:

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

המקרה הכללי

במקרה הכללי, נחלק למעשה למקרים הבאים:

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

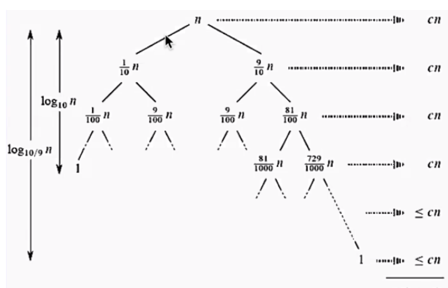
כעת קיבלנו חלק מאוזן, ולכן העומק יהיה בהכרח המקסימום מבין כל העלים בעץ:

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

אם כך, המקרה הממוצע יהיה בסך הכל:

$$\Theta\left(n\log n\right)\leq T\left(n\right)\leq\Theta\left(n^{2}\right)$$

לא נפתור את המקרה הכללי כרגע (בתרגול, באמצעות תוחלת והסתברות), אבל נבדוק זאת בדוגמא ספציפית, למשל, בתמונה הבאה:



כלומר, נקבל כי תמיד כל צד מתחלק בין 9 ו-1.

במצב כזה, נקבל כי הסיבוכיות הינה:

$$T(n)=T(n/10)+T(9n/10)+\Theta(n)$$

לפי זה, נעצור כאשר נגיע לעומק $\log_2 n$. בעקבות כך, הסיבוכיות הינה $T\left(n\right)=O\left(n\log n\right)$.

את המקרה הגרוע אפיינו כבר, וראינו כי $T\left(n\right)\leq cn^{2}$, דהיינו כי מתקיים $O\left(n^{2}\right)$.

נשים לב כי באלגוריתם השתמשנו בהשוואות. למעשה, נוכל לומר כי עד כה בכל האלגוריתמים שראינו היה שימוש בהשוואות. ראינו כי אלגוריתמים מסוימים טובים יותר או פחות במקרים שונים.

במצב כזה, נרצה לשאול, האם קיים חסום תחתון לכל אלגוריתמים באמצעות השוואה. ואכן, נוכל להוכיח כי החסם התחתון של אלגוריתם מיון באמצעות השוואה בין האיברים, הינו $\Omega\left(n\log n\right)$[6].

הגדרה

עץ החלטה הינו עץ בינארי **מלא** (תמיד ישנם שני צדדים), שאינו מאוזן.

הקודקודים בעץ הינן הזוגות הסדורים $1\leq i,j\leq n$ (i,j).

העלים בעץ ההחלטה הינן הפרמוטציות של הקלט.

מסלול בעץ הינו תוצאה של השוואה בין שני איברים.

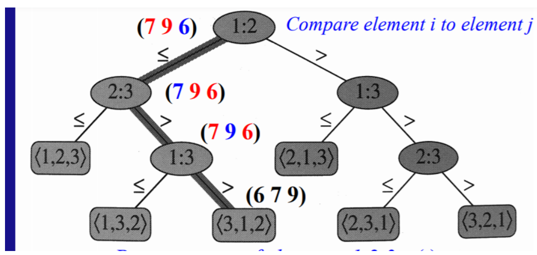
טענה

לא קיים אלגוריתם מיון באמצעות השוואה, שהחסם התחתון שלו קטן מ-$\Omega\left(n\log n\right)$.

הוכחה

באמצעות 'עץ החלטה', נוכל לחשב את מספר ההשוואות הנדרש במקרה הגרוע ביותר.

נתבונן למשל בהשוואה המתוארת ב'עץ ההחלטה' הבא:



דהיינו, הקודקודים הינם ההשוואות בין כל אלמנט i,j כאשר $1\leq i,j\leq n$. בנוסף, העלים בעץ הינן הפרמוטציות האפשריות במערך. קיבלנו כי לכל השוואה בין קלטים ישנו מסלול מסוים. יתר על כן, נוכל לומר כי הסיבוכיות הינה **המסלול הארוך ביותר בגרף**. כלומר, זהו **מספר ההשוואות המקסימלי** בעץ ההחלטה.

החישוב של עץ ההחלטה חייב להיות מלא, משורש העץ עד הקודקוד. בנוסף, לפי מה שתיארנו קודם לכן, מספר העלים בעץ צריך להיות $n!$ (סידור n איברים בשורה ללא חזרות).

בהינתן עץ החלטה עם $n!$ החלטות, מהו המסלול באותו העץ? המסלול הארוך ביותר בעץ צריך להיות בסך הכל $n!\leq2^{d}$ כאשר d הינו עומק העץ.

אם כך, נקבל בחישוב מהיר:

$n!\leq2^{d}\Rightarrow\log n!\leq\log\left(2^{d}\right)=d$.

נוכל גם לטעון כי $\log n!=\Theta\left(n\log n\right)$. נקבל:

$$\left(\frac{n}{2}\right)^{\frac{n}{2}}\leq n!\leq n^{n}\Rightarrow$$
$$\frac{n}{2}\log\frac{n}{2}\leq\log\left(n!\right)\leq n\log n$$

ולכן, הוכחנו כי $n\log n$ זהו החסם ההדוק של כל עץ החלטה, וממילא גם של כל מיון באמצעות השוואה.

[6] ייתכן אמנם מיון ב-$O\left(n\right)$, אבל במקרה כזה נצטרך להניח דברים על הקלט.

טבלאות גיבוב

שיעור מס' 4:
יום חמישי
11.11.20

הקדמה ומוטיבציה

בהינתן טבלה שמכילה נתונים ואיברים מסוימים, כשאין סדר ביניהם, (למשל, הכנסה של תעודות זהות, מספרי טלפונים ועוד) נרצה מבנה נתונים שמאפשר לנו לגשת לנתונים בזמן ריצה של $O\left(1\right)$.
שלושת הפעולות שנרצה במבנה הנתונים יהיו **חיפוש, הכנסה ומחיקה -** והן אמורות להיות כאמור בזמן ריצה קבוע.
לעומת זאת, במידה ונסדר אותם בעץ, נרצה כי זמן הריצה בחיפוש בעץ יהיה $\log\left(n\right)$.

נתבונן למשל במערך A הבא:

index	0	1	2	3	4	5	6	7
i		e_1		e_3				e_7

זהו מערך בגודל n, כך שלכל $0\leq i\leq n-1$ נוכל לסדר את המידע בטבלה על ידי הכנסתו ל-$A\left[i\right]$.
נבחין כי במקום זה ה**מפתח** הוא למעשה האינדקס. למשל, עבור e_7, נקבל כי המפתח הינו 7.

נרצה למצוא דרך לקשר בין ה**מידע** ובין **מפתח**. כלומר, בעצם נחפש פונקציה, $h\left(k\right)$ שתחזיר לנו את האינדקס הרצוי ב- $O\left(1\right)$ - זמן קבוע.
הפונקציה הפשוטה ביותר הינה $h\left(k\right)=k$. אם ניקח פונקציה שכזו, נקבל למעשה כי המפתח והאינדקסים הינם באותו טווח. אמנם, לעיתים דבר זה יכול לבזבז מקום - למשל, במקרה בו נשתמש במספר תעודת זהות בתור המפתח.
לכן, נרצה לחפש פונקציה אחרת - לדוגמא, בנוגע לתעודות זהות, נוכל לבחור תמיד את שתי הספרות האחרות.
אמנם, במקרה שכזה ייתכנו **התנגשויות -** שהרי ישנו סיכוי גבוה שלשתי תעודות זהות יהיו אותן שתי ספרות אחרונות.
לכן, נחפש פיתרון אחר.

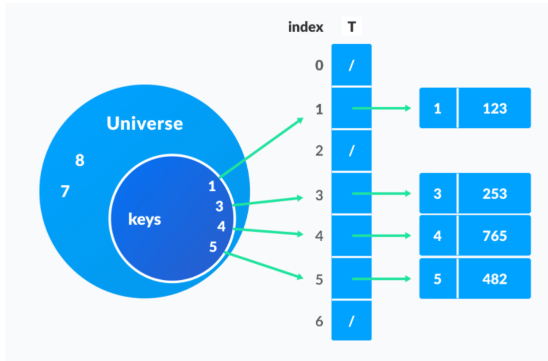
מבט כללי

טבלת גיבוב הינה הכללה של מערך בגודל $A\left[m-1\right]$, שמאפשרת לנו, באמצעות פונקצית גיבוב $h\left(k\right)$ לחשב את האינדקס במערך.
גודל טבלת הגיבוב הינו פרופרציונלי לגודל המערך, כך שמתקיים - $h\left(k\right)=k\mod m$.
פונקצית הגיבוב איננה חח"ע.
התנגשות מתרחשת כאשר מידע ממופה לאותו אינדקס - עלינו לחפש מנגנון שיאפשר לנו למנוע התנגשויות.

הגדרה

תהי U קבוצה של מפתחות, נסמן את גודלה ב-$\left|U\right|$.
תהיינה $K\subset U$ קבוצת המפתחות הנוכחיים, המקיימת כי $\left|K\right|=n$ ו-T טבלת הגיבוב, המקיימת $\left|T\right|=m\leq\left|U\right|$.
אזי נאמר כי $h\left(k\right):U\rightarrow\left[0,.,m-1\right]$ הינה פונקצית הגיבוב.
כלומר, $h\left(k\right)$ ממפה מפתחות מ-U לאינדקסים ב-T.
לכל $1\leq i\leq m-1$ מתקיים כי הגישה ל-$T\left[i\right]$ היא בזמן ריצה של $O\left(1\right)$.
ערך ריק בטבלת הגיבוב יסומן ב-NULL.
נניח כי $U=\left\{0,\ldots,N-1\right\}$

מבחינה ויזואלית, זה נראה כך:



כאשר המעגל הגדול הינו האוסף הכללי של המפתחות, ואילו המעגל הפנימי, שמכיל את "המפתחות הנוכחיים", שולח אותם לאינדקסים, באמצעות פונקציה מסוימת.
במהלך העיסוק בנושא זה, נבחן מספר סוגיות מרכזיות. למשל, מהן פונקציות "טובות" או "רעות"? מה קורה במידה ויש התנגשויות? כיצד נוכל להבטיח שנקבל זמן ריצה קבוע? מהו זמן הריצה הגרוע ביותר?
נתחיל לבחון שלוש צורות של פונקציות גיבוב:

(1) 'מיעון ישיר' - Direct-address.
(2) 'מיעון פתוח' - Open-addressing.
(3) 'מיעון סגור' - Closed-addressing או Chaining.

לפני שנתחיל לעסוק בכל אחת מהשיטות, נעיר כי השיטה הראשונה איננה מתמודדת עם בעיה של 'התנגשויות'. שתי השיטות האחרונות, בהן מספר המפתחות הנוכחי קטן באופן משמעותי מגודל המערך, מציגות גישות שונות להתמודדות עם התנגשויות אלו.
הגישה הראשונה הינה ליצור פונקציה, שבאמצעות נוסחה מתמטית עוברת לאינדקס אחר, במידה ומתרחשת התנגשות.
הגישה השנייה יוצרת רשימה מקושרת, ובכל פעם מכניסה את האיבר למיקום הראשון ברשימה - ומקשרת את השאר.
קודם לכן, נפתח במספר הגדרות ומונחים.

הגדרות

הנחות על פונקציית גיבוב - Simple uniform hashing assumption
בהינתן מערך כלשהוא, נוכל להניח כי ניתן למפות כל ערך לתוכו, ללא תלות בשאר האיברים.

מקדם עומס - load factor
בהינתן טבלת גיבוב T עם m חריצים, נגדיר את $\alpha = \frac{n}{m}$ להיות מקדם העומס - Factor Load כאשר n זהו מספר האלמנטים המשובצים.
בהכרח מתקיים כי $0 \leq \alpha \leq 1$.

טבלאות מיעון ישיר

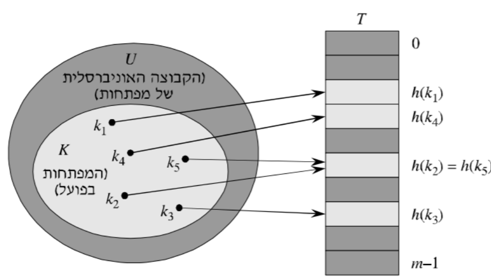
טבלת 'מיעון ישיר' היא הדרך הבנאלית והנוחה ביותר לסידור איברים.
גודל טבלת הגיבוב הינו למעשה גודל המערך, ופונקציית הגיבוב מקיימת כי $h(k)=k$.
במצב כזה, אין מצב להתנגשויות, וזמן הריצה הוא תמיד $O(1)$.
החסרונות המרכזיים של טבלה זו הינם כי גישה זו לעיתים קרובות מבזבזת מקום, וגם איננה יעילה בקלטים גדולים.

נתבונן בפסאודו קוד של חיפוש, הכנסה ומחיקה מטבלת הגיבוב T:

אלגוריתם 3 מיעון ישיר

- חיפוש של איבר k:
  (1) תחזיר את האיבר $T[k]$
- הכנסה של איבר x:
  (1) $T[x.key] \to x$
- מחיקה של איבר x:
  (1) $T[x.key] \to null$

דוגמה למיעון ישיר:



טבלאות מיעון פתוח

בשיטת המיעון הפתוח, נפעיל את פונקציית הגיבוב על הערך באינדקס, ובמידה ולא הצלחנו למצוא מקום פנוי והבדיקה לא צלחה, נמשיך לאיבר הבא על פי הנוסחה.
למעשה, ישנה פונקצית $h(k,i): U\times[0,\dots,m-1]\to[0,\dots,m-1]$, כך שמתבצעות בדיקות על פי אינדקס i שרץ.

נכתוב את פונקציית הגיבוב כך - $h(k,i): U\times[0,\dots,m-1]\to[0,\dots m-1]$.
נמשיך עם החיפוש, עד שנמצא מקום ריק. במידה ועשינו זאת m פעמים, הטבלה תתמלא.
כיצד מתבצעות הפעולות בשיטה זו?

- **הכנסה:** מפעילים את פונקציית הגיבוב, בודקים את הטבלה, עד שמוצאים איבר ריק.
- **חיפוש:** מפעילים את פונקצית הגיבוב, עד שמוצאים את האיבר (הצלחה) או איבר ריק (כישלון).
- **מחיקה:** על מנת שלא לפגוע בדרך פעולת הגיבוב, נסמן את האיבר אותו מחקנו כ"מחוק".

סיבוכיות זמן הריצה נקבעת למעשה על ידי מספר האיברים.

ישנם למעשה שלושה סוגים של מיעון פתוח:

(1) בדיקה ליניארית.

(2) בדיקה ריבועית.

(3) גיבוב כפול.

נתחיל להתעמק בכל אחת מהן.

בדיקה ליניארית.

במקרה הזה פונקציית הגיבוב הינה $h\left(k,i\right)=\left(h'\left(k\right)+i\right)\mod m$.

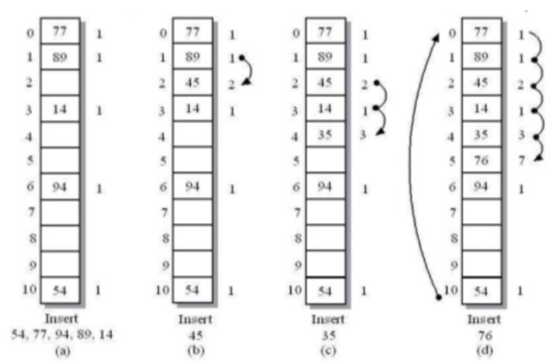
כאשר $h'\left(k\right)$ הינה פונקצית גיבוב הנקבעת מראש, ו-i הינו אינדקס שאנו רצים עליו.

כלומר בהינתן k מסוים, סדרת הבדיקות תהיה:

$$T\left[h'(k)\right],T\left[h'(k)+1\right],\ldots,T\left[h'(k)+m-1\right]$$

אם כך, סדרת הבדיקות תהיה תמיד סדרה מונטונית עולה. נשים לב כי אנחנו יכולים לייצר אך ורק m איברים בסדרה.

נתבונן בדוגמה של מיעון ליניארי:



נוכל לראות כי במקרה הגרוע ביותר נקבל שההסתברות לקבל מקום פנוי הולכת וקטנה ככל שהאינדקסים עולים. דהיינו, זמן החיפוש הולך ועולה עד שטבלת הגיבוב הופכת ללא יעילה.

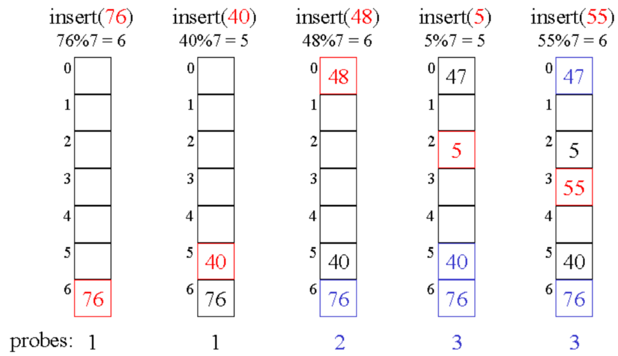
בדיקה ריבועית.

בצורת בדיקה זו, פונקצית הגיבוב הינה למעשה:

$$h(k,i)=\left(h'(k)+c_1i+c_2i_k^2\right)\mod m$$

כאשר c_1,c_2 הינם קבועים שונים מאפס ו-$h'\left(k\right)$ הינה פונקציית גיבוב כלשהיא ו-i כאמור הולך וגדל לפי גודל ההתנגשויות.

נוכל לראות זאת בדוגמה הבאה:



פונקצית הגיבוב בדרך זו, מונעת את הצפיפות של הבלוקים, אך אמנם ייתכנו רצפים של צפיפיות.
גם טבלת הגיבוב הזו יכולה להכיל עד m איברים.

גיבוב כפול.

כעת, במקום להתעסק עם בדיקות ליניאריות או ריבועיות, ננסה ליצור שתי פונקציות גיבוב.
כלומר, למשל:

$$h(k,i) = (h_1(k) + i \times h_2(k)) \mod m$$

כאשר $h_1(k)$ ו-$h_2(k)$ הינן שתי פונקציות גיבוב.
הבדיקה הראשונה הינה ב-$T\left[h_1\left(k\right)\right]$ ומכאן והלאה אנחנו ממשיכים ל-$h_2\left(k\right) \mod m$.
בשביל לוודא שאנחנו מגיעים לכל התאים בטבלה, על $h_2\left(k\right)$ להחזיר בהכרח **מספר ראשוני** (אחרת אנחנו נכנסים למעגליות בפונקציות הגיבוב). ניתן גם להוכיח כי מספר האפשרויות שאנו יכולים לקבל הוא m^2 ולכן מבחינה הסתברותית מדובר על פונקציה טובה יותר.

דוגמה

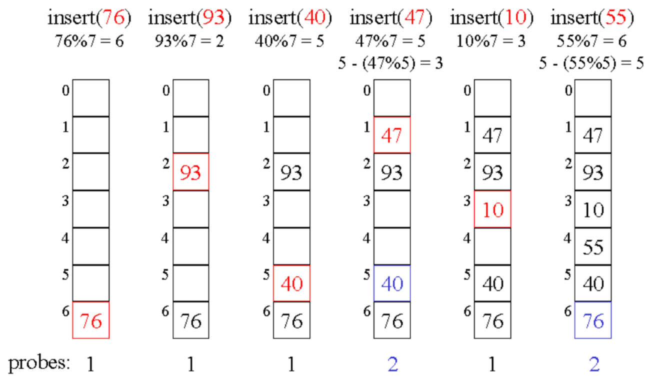
אם נרצה, נוכל להתבונן בדוגמה הבאה:

$$: h_I(k) = k \mod m$$

$$h_2(k) = 1 + (k \mod m') \quad m' < m$$

כאשר m הינו חזקה של 2 ואילו $m' < m$.

אפשר לראות דוגמה נוספת:



טענה

בהינתן טבלת גיבוב פתוחה עם מקדם עומס $\alpha = \frac{n}{m} < 1$, אזי מספר הבדיקות הצפוי בפונקצית גיבוב רגילה הינו:

מקסימום $\frac{1}{1-\alpha}$ בחיפוש לא מוצלח.

מקסימום $\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$ בחיפוש מוצלח.

כאשר α קבוע זמן החיפוש הינו $O\left(1\right)$.

טבלאות מיעון סגור

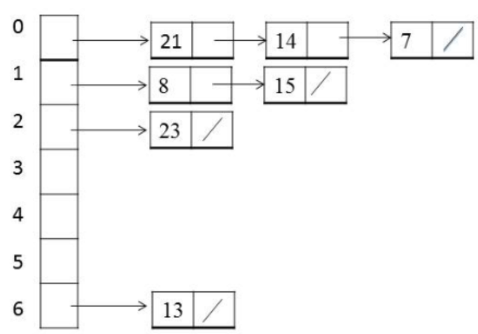
בצורת מיון זו, כאמור, נמיין את כל המפתחות ה-k_i לתוך $T\left[h\left(k_i\right)\right]$ כאשר T הינה טבלת הגיבוב. כאשר בכל אחד מהמיקומים במערך ישנו מצביע לתוך רשימה מקושרת.

הפעולות בצורה זו מתרחשות כך:

- **הכנסה:** מתבצעת תמיד לתוך ראש הרשימה המקושרת. במקרה הגרוע זהו זמן ריצה של $O\left(1\right)$.
- **חיפוש ומחיקה:** כניסה לתוך הרשימה המקושרת באמצעות פונקציית הגיבוב, וחיפוש ומחיקה עד למציאה.

כלומר, זמן הריצה במקרה הגרוע ביותר, פרופציונלי לאורך של הרשימה הארוכה ביותר.

דוגמה לשימוש במיעון סגור:



טענה

בהינתן טבלת מיעון סגור עם מקדם עומס $\alpha = \frac{n}{m} < 1$, כמות הבדיקות הצפויה לכל פונקציית גיבוב הינה - $\Theta\left(1+\alpha\right)$, הן לחיפוש מוצלח והן לחיפוש לא מוצלח.

כאשר $\alpha = O\left(1\right)$, זמן החיפוש הממוצע הינו $\Theta\left(1\right)$.

כאשר $n = O\left(m\right)$ ו-1 $= \alpha = O\left(\frac{m}{n}\right)$ אזי זמן החיפוש הינו גם $\Theta\left(1\right)$.

בניית טבלאות גיבוב

כיצד נוכל לבנות פונקציית גיבוב טובה? ראינו כי ישנן פונקציות 'טובות' ופונקציות 'גרועות', שתלויות למעשה בבניית הפונקציה, לכן חשוב שנבנה את הפונקציה בצורה המתאימה.
אנחנו מניחים כי פונקציית הגיבוב מכניסה את האיבר לתוך המערך, ללא תלות בשאר האיברים.[7]

כאמור, עלינו לתכנן פונקציית גיבוב. כעת, ישנן שתי דרכים לעשות זאת:

(1) **היוריסטיקה:** לפי כלל אצבע שנראה הגיוני, כיוון שאיננו יכולים להבטיח כי קיימת פונקציית גיבוב שמתנהלת בצורה יעילה בכל מקרה אפשרי.
(2) **בצורה רנדומלית:** בחירת פונקציות גיבוב בצורה רנדומלית, על מנת שההסתברות תהיה טובה יותר.

פונקציות היוריסטיות.
מהן הפונקציות שעובדות ברוב המקרים? הרעיון הינו כל מידע שנרצה לסדר בנוי על היגיון פנימי, וכלל האצבע משתמש בעובדה זו ובתכונות אלו (למשל, תעודות זהות לא נוצרו סתם). נרצה להוכיח, בהמשך, כי ההסתברות למקרה הגרוע קטנה ביותר.
נציג כאן שתי גישות מרכזיות.

שיטת החילוק
יהי $U = N = \{0, 1, 2 \ldots\}$ קבוצה של מספרים טבעיים.
נמפה k לתוך מערך בעל m איברים, על ידי לקיחת השארית של מספר המתחלק ב-m:

$$h(k) = k \mod m$$

על מנת שדבר זה יעבוד, עלינו להימנע מבחירת m שהינו חזקה של 2 (עלול לפגוע ביעילות האלגוריתם) - על m להיות מספר ראשוני.

יהי $|U| = n = 2000$.
אנחנו מעוניינים ב-c שיגרום לנו למקסימום 3 התנגשויות במיפוי.
נחשב: $\frac{n}{c} = \left\lfloor \frac{2000}{3} \right\rfloor = 666$.
נמצא את המספר הראשוני הקרוב ביותר ל-666 ושאיננו חזקה של 2 ונקבל את $m = 701$.
אם כך בהכרח $h(k) = k \mod 701$.
לדוגמא, הערכים $0, 701, 1402$ ימופו כולם ל-0.

שיטת הכפל
נמפה k מפתחות לתוך m תאים, בצורה הבאה:

(1) נכפול זאת ב-A כך ש-$0 < A < 1$.
(2) נקח את החלק השברי של kA (כלומר את $kA - \lfloor kA \rfloor$).
(3) ניקח את הערך השלם של התוצאה הנכפלת ב-m ונקבל:

[7] דילגנו בכיתה על ההוכחה הפורמלית של חלק זה.

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

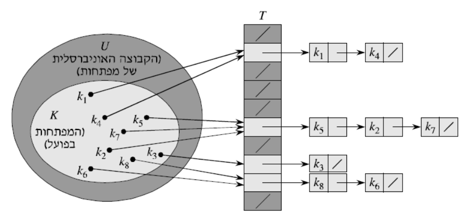
נשים לב כי שיטה זו 'רגישה' פחות לערכים של A, מכיוון שההתנהגות שלה רנדומלית הרבה יותר, שהרי לרוב המפתחות אין קורלציה עם A.

ניקח $k = 1234, A = 0.4, m = 100$.
נקבל כי $kA = 493.6$ וגם $(kA - \lfloor kA \rfloor) = 0.6$.
ובסך הכל $h(1234) = 60$.

שיעור מס' 5:
יום רביעי
18.11.20

מחלקות אוניברסליות וחסמי זמני ריצה

לשם הרענון, נציג דוגמה כללית של פתרון התנגשויות על ידי שרשור:



במקרה זה $|U| = 30^{15}$ ו-$|K| = n = 1000$וגם $|T| = m = 3000$.
קיבלנו כי גם $m = O(n)$, ומאידך, מקדם העומס $\alpha = \frac{n}{m} < 1$.
לכן מתקיים כי $\alpha = \frac{n}{m} = \frac{n}{O(1)} = O(1)$.
במקרה הגרוע ביותר, הוכנסו $O(n)$ מפתחות (כל n המפתחות מגובבת לאותו התא ויוצרות רשימה באורך n), ולכן זמן הריצה הארוך ביותר יהיה $O(n)$.
המקרה הטוב ביותר, יהיה כי נמצא את המפתח בזמן $O(1)$ (כאשר בכל תא ישנה רשימה של $O(1)$ ממפתחות) נרצה למצוא מהי הדרך להגיע לכך.

לאחר שדיברנו בתרגול על מושגי הסתברות ותוחלת, נוכל להשתמש בכלים אלו כעת על מנת לדבר על זמנים ממוצעים.
דיברנו על כך שהגיבוב אחיד ופשוט, ולכן ההסתברות שמפתח k, שאיננו מאוחסן בטבלה, יגובב לתא מסוים זהה עבור כל m התאים (ללא תלות בשאר התאים).
אנחנו מדברים על **זמן ממוצע** ולכן לא נתחשב **במקרה הגרוע** אלא במקרה ממוצע. דהיינו, הבחירה שלנו תהיה **רנדומלית.** אם מדובר בבחירה רנדומלית, נרצה להוכיח כי **בממוצע**, הביצועים יהיו טובים. כיצד נוכל להוכיח זאת? באמצעות הכלים שלמדנו, על תוחלת והסתברות.
קודם לכן, נוכל להכריע באמצעות שיטה הנקראת "גיבוב אוניברסלי" המשיגה ביצועים טובים בממוצע.

נזכיר קודם לכן את המושגים שדיברנו עליהם בתרגול:

תזכורת למונחי הסתברות:

מרחב מדגם

מרחב הסתברות הוא זוג $(\Omega, \mathbb{P})$.

Ω נקרא מרחב המדגם. זו קבוצה (לא ריקה). האיברים נקראים מאורעות אטומים ותתי קבוצות נקראים מאורעות.

$\mathbb{P}$ היא פונקציה $\mathbb{P}: 2^{\Omega} \to [0,1]$ שמקיימת את התנאים:

- $\sum\limits_{\omega \in \Omega} \mathbb{P}(\omega) = 1$. כלומר סכום ההסתברות של המאורעות האטומים הוא 1.
- לכל $A \subset \Omega$ מתקיים כי $\mathbb{P}(A) = \sum\limits_{\omega \in A} \mathbb{P}(\omega)$.

משתנה מקרי

משתנה מקרי על מרחב מדגם Ω הוא פונקציה $X: \Omega \to \mathbb{R}$. נסמן $\mathbb{P}(X = r) = \mathbb{P}(\{\omega \mid X(\omega)\} = r)$

תוחלת

עבור משתנה מקרי $X: \Omega \to \mathbb{R}$ התוחלת של X היא:

$$\mathbb{E}(X) = \sum_{\omega \in \Omega} X(\omega) \mathbb{P}(\omega) = \sum_{x \in \mathrm{Im} X} x \cdot \mathbb{P}(X = x)$$

כלומר, התוחלת היא ממוצע הערכים אותם צפוי המשתנה לקבל.

כעת, נעבור להגדרה הרלוונטית אלינו:

הגדרה

יהי H אוסף סופי של פונקציות גיבוב הממפות קבוצה אוניברסלית נתונה U של מפתחות אל התחום $\{0, 1 \ldots, m-1\}$.

אוסף כזה ייקרא **אוניברסלי**, אם עבור כל זוג מפתחות שונים זה מזה, $k, l \in U$ מספר פונקציות הגיבוב $h \in H$ המקיימות $h(k) = h(l)$ הוא לכל היותר $\frac{|H|}{m}$.

כלומר, אם ניקח באופן אקראי פונקצית גיבוב מתוך H. הסיכוי להתנגשות בין k ו-l קטן שווה מ-$\frac{1}{m}$.

משפט

יהי H **אוסף אוניברסלי** ותהי $h \in H$ פונקציית גיבוב, כך ש-h מגבבת n מפתחות לתוך טבלה T בגודל m.

נניח כי T היא טבלה הנפתרת באמצעות שרשור.

אם ניקח $\alpha = \frac{n}{m} < 1$, אזי (כיוון שהתוחלת משפיעה על זמן החיפוש):

- אם מפתח k כלשהוא לא נמצא בטבלה, אזי **התוחלת** $\mathbb{E}\left[n_{h(k)}\right]$ של אורך הרשימה שהמפתח k מגובב אליה, היא לכל היותר α.
- אם המפתח k נמצא בטבלה, אזי **התוחלת** $\mathbb{E}\left[n_{h(k)}\right]$ היא כל היותר $1 + \alpha$.

הוכחה

יהי n_i אורך רשימה מקושרת כלשהיא, בתא i .

יהי $\mathbb{E}[n_i]$ האורך המשוער של הרשימה.

נגדיר מאורע מקרי - המשתנה המציין - $X_{kl} = I\{h\{k\} = h\{l\}\}$.

כיוון שהנחנו שההסתברות של הפיזור שווה, שהרי מדובר ב**אוסף אוניברסלי**, אזי ההסתברות ששני מפתחות יתנגשו זה בזה, היא לכל היותר $\frac{1}{m}$, וממילא (לפי טענה שהייתה בתרגול - למה 5.1 בספר) גם **התוחלת** של X_{kl} תהיה קטנה מ-$\frac{1}{m}$. כלומר בסך הכל $E[X_{kl}] \leq \frac{1}{m}$.

נגדיר כעת לכל מפתח k משתנה מקרי נוסף – Y_k – שמונה את מספר המפתחות שנמצאות בתא ש-k נמצא בו, ושונים ממנו.
ובסך הכל:

$$Y_k = \sum_{\substack{l \in T \\ l \neq k}} X_{kl}$$

מליניאריות התוחלת, אותה הוכחנו בתרגול, עולה:

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

נחלק לשני מקרים:
במקרה בו $k \notin T$, כלומר k איננו נמצא בטבלה, אזי נקבל כי $n_{h(k)} = Y_k$, כלומר אורך הרשימה שווה למשתנה המקרי במצב זה.
כמו כן נבחין כי גודל הקבוצה $\left|\{l \in T \mid l \neq k\}\right| = n$.
לכן נקבל כי התוחלת למעשה שווה:

$$\mathbb{E}\left[h_{h(k)}\right] = \mathbb{E}\left[Y_k\right] \leq \sum_{\substack{l \in T \\ l \neq k}} \frac{1}{m} \overset{\text{גודל הקבוצה}}{=} \frac{n}{m} = \alpha$$

כפי שרצינו.
אם $k \in T$, כלומר המפתח k נמצא למעשה ברשימה הרצויה, אך אמנם הוא לא נמצא ב-Y_k מעצם ההגדרה של המשתנה המקרי, לכן נקבל כי $n_{h(k)} = Y_k + 1$.
כמו כן, נקבל כי גודל הקבוצה -1 $\left|\{l \in T \mid l \neq k\}\right| = n - 1$.
לכן בסך הכל נקבל:

$$\mathbb{E}\left[h_{h(k)}\right] = \mathbb{E}\left[Y_k\right] + 1 \leq \left(\sum_{\substack{l \in T \\ l \neq k}} \frac{1}{m}\right) + 1 \overset{\text{גודל הקבוצה}}{=} \frac{n-1}{m} + 1 = 1 + \alpha - \frac{1}{m} < 1 + \alpha$$

כנדרש.

מסקנה
אם מדובר בגיבוב אוניברסלי ופתרון בעיות על ידי שרשור בטבלה עם m תאים, זמן הריצה של n פעולות הכנסה, חיפוש או מחיקה, כאשר מספר התאים בטבלת הגיבוב הוא לפחות ביחס ישר למספר האיברים בטבלה, ($n = O\left(m\right)$), יהיה $\Theta\left(n\right)$.

הוכחה
נבחין כי כיוון שבוצעו $O\left(m\right)$ הכנסות, מתקיים כי $n = O\left(m\right)$, ולכן $\alpha = o\left(1\right)$.
לכן, ממה שראינו לגבי זמני פעולות של הכנסה וחיפוש, שהינן בזמן קבוע, נקבל בסך הכל כל סך הכל הפעולות יהיה $\Theta\left(n\right)$.

משפט

תהי טבלת גיבוב פתוחה עם מקדם עומס $\alpha = \frac{n}{m} < 1$, אזי מספר הקפיצות בהינתן גיבוב אוניברסלי, הינה:
$\frac{1}{1-\alpha}$ בחיפוש לא מוצלח, ו-$\frac{1}{\alpha}\ln\left(\frac{1}{1-\alpha}\right)$ בחיפוש מוצלח.

על מנת להבין זאת טוב יותר, נתבונן קודם כל בדוגמא. נניח שיש לנו טבלה חצי מלאה. דהיינו $\frac{n}{m} = \frac{1}{2}$, אזי לפי הטענה, יהיו במקסימום $\frac{1}{1-\frac{1}{2}} = 2$ בדיקות עד שנכריע שהמספר איננו נמצא. בצורה דומה, ההכרעה האם האיבר נמצא תהיה : $2 > \frac{1}{\frac{1}{2}}\ln\frac{1}{1-\frac{1}{2}}$.

לעומת זאת, אם $\frac{n}{m} = \frac{9}{10}$, כלומר הטבלה כמעט מלאה, אזי חיפוש מוצלח ייקח $\frac{1}{1-9/10}\ln\frac{1}{1-1/2} = 10 \times 2.31$ ואילו חיפוש לא מוצלח ייקח $10 = \frac{1}{1-9/10}$.

הוכחה

תחילה, נוכיח עבור חיפוש לא מוצלח.

מבחינה אינטואטיבית, בכל בדיקה, פרט לבדיקה האחרונה, אנחנו מגלים 'תא תפוס'. בבדיקה האחרונה מתגלה תא ריק. אנחנו נחשב את כל ההסתברויות הללו, דרכם נחשב את התוחלת, ואז נחסום אותה מלעיל, וסיימנו.

ברמה הכללית נעשה זאת כך.

נגדיר משתנה מקרי X, שיספור לנו את מספר הבדיקות בחיפוש כושל.

כמו כן, נגדיר A_i להיות המאורע בו נערכה בדיקה i והתגלה תא תפוס.

אם כך, המאורע $\{X \geq i\}$ הוא סך הכל חיתוך כל המאורעות $A_1 \cap A_2 \cap \ldots \cap A_{i-1}$.

ולכן מנוסחה מתמטית כלשהיא שלמדנו בהסתברות עולה:

$$\mathbb{P}\left(A_1 \cap A_2 \cap \ldots A_{i-1}\right) = \mathbb{P}\left\{A_1\right\} \cdot \mathbb{P}\left\{A_2 \mid A_1\right\} \ldots \mathbb{P}\left(A_{i-1} \mid A_1 \cap A_2 \cap \ldots A_{i-2}\right)$$

בתכל'ס, קיימים לנו n איברים ו-m תאים, לכן פשוט נקבל מבחינת כפל המאורעות:

$$\mathbb{P}\left(\{X \geq i\}\right) == \frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \frac{n-2}{m-2} \cdots \frac{n-i+2}{m-i+2} \leq \left(\frac{n}{m}\right)^{i-1}$$

נבצע 'רק כמה פעולות אריתמטיות'[8]:

$$\mathbb{E}[X] \overset{(*)}{=} \sum_{i=1}^{\infty} \mathbb{P}\{X \geq i\} \overset{(**)}{=} \sum_{i=1}^{\infty} \left(\frac{n}{m}\right)^{i-1} \overset{(***)}{=} \sum_{i=1}^{\infty} \alpha^{i-1} \overset{(****)}{=} \sum_{i=0}^{\infty} \alpha^{i} \overset{(*****)}{=} \frac{1}{1-\alpha}$$

$(*)$ נוסחה שלא למדנו.

$(**)$ טענה בשורה קודם.

$(***)$ הגדרה.

$(****)$ שינוי אינדקסים.

$(*****)$ סכום סדרה חשבונית.

כעת נוכיח את החיפוש המוצלח.

[8] בגדול, זה חסם האיחוד, או משהו כלשהוא שלא למדנו, או נלמד בקרוב בהסתברות.

קודם כל, נוכל להבחין כי הכנסת איבר כלשהוא למערך דורשת בממוצע $\frac{1}{1-\alpha}$ בדיקות. שכן, הכנסת איבר דורשת קודם כל חיפוש כושל!

כעת, אם נרצה למצוא איבר מסוים, נצטרך לעשות 'את אותה הדרך' שעשינו עבור הכנסתו.

לפי מה שראינו קודם לכן, הממוצע הינו $\frac{m}{m-i} = \frac{1}{1-\frac{i}{m}}$ (כאשר i זהו מספר הבדיקות שנדרשנו לבדוק לפני שהכנסנו את האיבר)

נשים לב כי עד כה התייחסנו למאורע ספציפי – הכנסה לתא מסוים. אמנם, עלינו 'לאחד' את כל הקבוצות בהם דבר זה מתקים, כלומר, על פני כל המפתחות:

$$\frac{1}{n}\sum_{i=0}^{n-1}\frac{m}{m-i} = \frac{m}{n}\sum_{i=0}^{n-1}\frac{1}{m-i} = \frac{1}{\alpha}\sum_{i=0}^{n-1}\frac{1}{m-i}$$

כעת ננסה לחסום מלעיל[9]:

$$\frac{1}{\alpha}\sum_{i=0}^{n-1}\frac{1}{m-i} \overset{(*)}{=} \frac{1}{\alpha}\sum_{k=m-n+1}^{m}\frac{1}{k} \overset{(**)}{\leq} \frac{1}{\alpha}\int_{m-n}^{m}\frac{1}{x}dx \overset{(**)}{\leq}$$
$$\frac{1}{\alpha}\ln\frac{m}{m-n} = \frac{1}{\alpha}\ln\frac{1}{1-\alpha}$$

כאשר $(*)$ נובע מטריק אלגברי, $(**)$ מבחן האינטגרל כנראה, $(***)$ חסימת האינטגרל ההרמוני. כנדרש.

תכנון מחלקה אוניברסלית.

נוכיח כי לא קשה לבנות מחלקה אוניברסלית, באמצעות שימוש במספרים ראשוניים ומודולו.

תחילה, נבחר מספר ראשוני גדול מספיק, כך שמתקיים $p > m$.

כעת, ניקח את הקבוצה $Z_p = \{0, 1 \ldots, p-1\}$ של המספרים הראשוניים עד p.

ניקח כעת $a, b \in Z_p$, ונגדיר לכל אחת מהן פונקציית גיבוב $h_{a,b}$ על ידי העתקה ליניארית ושימוש כפול במודולו:

$$h_{a,b}(k) = ((ak+b) \mod p) \mod m$$

מכך יתקיים כי אוסף כל פונקציית הגיבוב יהיה:

$$H_{p,m} = \{h_{a,b} \mid a, b \in \boldsymbol{Z}_p \text{ and } a \neq 0\}$$

על מנת לבחור פונקציה בצורה רנדומלית, נבחר רנדומלית את המספרים הראשוניים $a, b \in Z_p$.

[9] "האינטגרל? זהו משפט במתמטיקה"

טענה

יהיו k_1 ו-k_2 מפתחות שונים. לכל שני מספרים $x_1, x_2 \in Z_p$ הסיכוי ש-k_1 יגובב ל-x_1 ו-k_2 יגובב ל-x_2 יהיה $\frac{1}{p^2}$.
דבר זה גורר כי המשפחה $\boldsymbol{H}_{p,m} = \{h_{a,b} := a, b \in \boldsymbol{Z}_p\}$ הינה מחלקה אוניברסלית.

הוכחה

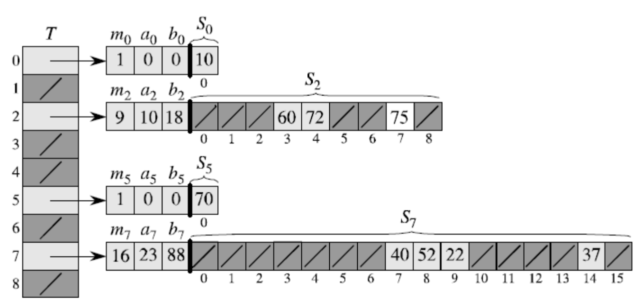
פה זה לא קורס בתורת המספרים.
אכן, לא הוכחנו את הטענה בכיתה, אבל הסתמכנו על כך שבהינתן שתי אי השוויונות הבאים:

$$ak_1 + b = x_1 \bmod p$$
$$ak_2 + b = x_2 \bmod p$$

תמיד יהיה פיתרון יחיד כאשר p ראשוני. לכן, הסיכוי לקבל את פונקצית ההסתברות הרצויה הינו למעשה הסיכוי לקבל גם את a וגם את b, ולמעשה הינו $\frac{1}{p^2}$.

גיבוב מושלם

הרבה פעמים משתמשים בטבלת גיבוב, בשל הזמן המוצלח של התוחלת שלה.
אבל לעיתים אפשר להשתמש אף בגיבוב טוב יותר. דבר זה יתקיים כאשר קבוצת המפתחות תהיה סטטית – למשל על קבוצה של אנשים שלא משתנה, מספר תעודות זהות שלא משתנה וכו'. אם כך, האינטואיציה מסבירה כי בעקבות כך נוכל למצוא ב- $O(1)$ גם במקרה הגרוע.
דבר זה נקרא **גיבוב מושלם.**
כיצד זה מתבצע? אנחנו נשתמש בגיבוב בשתי רמות.
ברמה הראשונה, נבצע גיבוב באמצעות שרשור. כלומר, נגבב n מפתחות ל-m תאים. אלא שבמקום רשימה מקושרת, ניצור **פונקציית גיבוב משנית.**
ברמה השנייה, אם כך, ניצור פונקציית גיבוב שנייה.
למשל, כמו בדוגמה הבאה:
פונקציית הגיבוב הראשונה – $h(k) = ((ak+b) \mod p) \mod m$
פונקציית הגיבוב השנייה – $h_i(k) = ((a_i k + b_i) \mod p) \mod m_i$.
ניקח למשל – $K = \{10, 22, 37, 40, 52, 60, 70, 72, 75\}$.
ואז $a = 3, b = 42, p = 101, m = 9$.
אם נחפש את 75, אבל אז נקבל כי $h(75) = 2$, וכעת נבדוק את 7 $h_2(75) =$.
הקפיצה הראשונה לוקחת זמן קבוע, והקפיצה השנייה גם היא לוקחת זמן קבוע.
ומבחינה ציורית:



כלומר, לכל תת קבוצה של המספרים יצרנו פונקציית גיבוב.

משפט
אם נאחסן n מפתחות בטבלת גיבוב שגודלה $m=n^2$, באמצעות פונקציית גיבוב $h\in H$ כאשר H הוא מחלקה אוניברסלית של פונקציות גיבוב, ההסתברות שתהיה לפחות התנגשות אחת, קטנה מ-$\frac{1}{2}$.
הוכחה
סך הכל, קיימים $\binom{n}{2}$ זוגות שעלולים להתנגש. כאמור, ההסתברות שזוג כלשהוא יתנגש הינה $\frac{1}{m}$, שהרי מדובר במחלקה אוניברסלית.
כעת, יהי X משתנה מקרי המונה את מספר ההתנגשויות.
נקבל, עבור התוחלת:

$$\mathbb{E}\left[X\right]=\binom{n}{2}\cdot\frac{1}{n^2}=\frac{n^2-n}{2}\cdot\frac{1}{n^2}<\frac{1}{2}$$

אם כך, דבר זה אומר שהסיכוי שתהיה התנגשות נמוך מהסיכוי שלא תהיה התנגשות, ולכן מובטח כי נוכל למצוא פונקציית גיבוב שתהיה בלי התנגשויות כלל.
הבעיה היא עם גיבוב מושלם הוא שלעיתים טבלת $m=n^2$ עלולה להיות גדולה מדי. לכן, ניתן להראות (ולא נעשה זאת), כי בממוצע המקום שנצטרך יהיה לינארי.

שיעור מס' 6:
יום חמישי
25.11.20

מיון ערימה

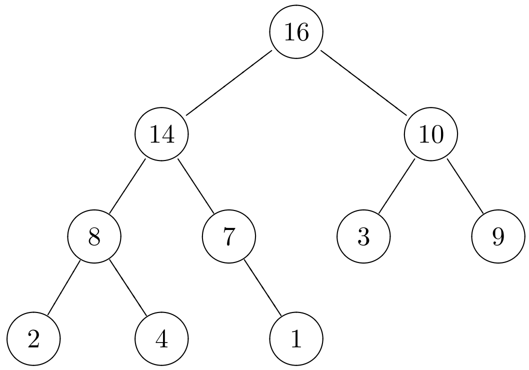
ערמות

במהלך השיעור, נתעסק במבני נתונים חדשים, שלא הכרנו עד כה.
נתחיל תחילה מדוגמא, שתיתן לנו מוטיבציה להבנה, ולאחר מכן נראה כיצד ניתן לממש בתור מבנה נתונים.

נניח שישנה חברת אינטרנט המוכרת מסכים, ומוציאה מכרזים לקנייתם. המסך, כמובן, הולך למציע המחיר הגבוה ביותר. עלינו להמציא אלגוריתם, שמאפשר את שמירת כל ההצעות, כך שנוכל להכניס, להוציא ולעדכן.
אנחנו מחפשים את הזמן היעיל ביותר, דרכו נוכל למצוא את ההצעה הגבוהה ביותר (הסדר בין הערימות האחרות איננו חשוב). אם כך, נרצה להגיע למצב בו נוכל להוציא ב-$O\left(1\right)$ את המקסימלי. נדגיש שוב שאכפת לנו **בעיקר** מהמקסימום, ולא מהסדר בין האיברים.
לסיכום, בצורה פורמלית, בהינתן x קבוצת מספרים דינימית, נרצה מבנה נתונים S, כך שנקבל את $\max\left(S\right)$ בזמן ריצה של $O\left(1\right)$, ושמאפשר הכנסה של איברים – $\text{insert}\left(x,S\right)$, הוצאת איבר מקסימלי, והעלאת הערך של איבר x ב-k.
S ייקרא Priorty.
על כמות המקום להיות ליניארית -$O\left(n\right)$ ושאר הפעולות אמורות להתבצע ב-$O\left(\log n\right)$.
נבצע דבר זה באמצעות ערימה.

ערימה ממומשת למעשה באמצעות עץ בינארי, כמעט מלא (מלבד השורה האחרונה). אינטואטיבית, זמן ריצה של עץ כזה יהיה $\log n$, אם נרצה להכניס לו n איברים.
כלומר, ניקח מערך A של n איברים.
תכונת הערימה הינה כי **האיבר המקסימלי** יהיה **בשורש העץ**. אם כך, מעצם ההגדרה, המקסימום נמצא בשורש.

לדוגמא:



נשים לב כי גובה העץ הינו 3, ומלבד השורה האחרונה, העץ עצמו **מלא**. נשים לב כי בכל תת עץ, שורש העץ הינו המקסימלי.

במערך, נוכל לסדר זאת כך:

נגדיר-

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

$left\left(i\right)=2i$

$right\left(i\right)=2i+1$

ולכן זה ייראה כך במערך:

1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

גובה העץ הינו $\log n$, כאשר **השורש** נקרא רמה 0. בכל רמה h ישנם 2^h קודקודים.

במידה והרמה האחרונה הייתה מלאה, אזי מספר הקודקודים הינו $\sum\limits_{i=0}^{h}2^{i}=2^{h+1}-1$.

מספר הקודקודים מלבד הרמה האחרונה, הינו בהכרח $\sum\limits_{i=0}^{h-1}2^{i}=2^{h}-1$.

מספר הקודקודים ברמה האחרונה יכול להיות $1\le i\le 2^{h}$. אם כך, מספר הקודקודים בעץ הינו בהכרח $2^{h}\le n\le 2^{h+1}-1$.

שמירה על תכונות הערימה

כיצד נוכל לעשות שינויים בעץ כזה? נתבונן בפעולות האפשריות בעץ.

נניח שישנה ערימה קיימת, ונרצה להכניס לתוכה איבר חדש, נצטרך לשמור על **תכונת הערימה, בכל תת מערך של העץ**.

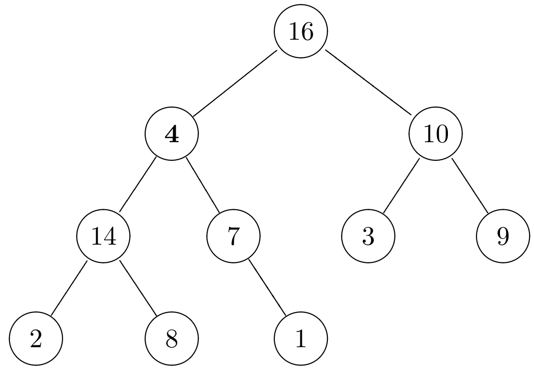
נרצה שהאיבר יהיה **בסוף המערך**, על מנת לשמור על מלאות העץ.

כיוון שאנחנו בכל פעם בודקים רק על צד אחד של עץ, הפעולות תיקח $\log n$, בלבד.

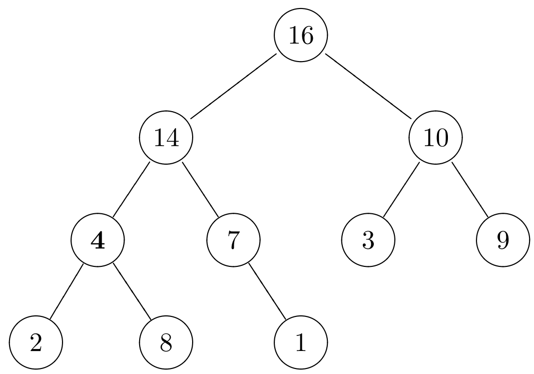
נדגים את רעיון האלגוריתם.

נרצה שבכל חלק של העץ, הוא ישמור על תכונת הערימה.

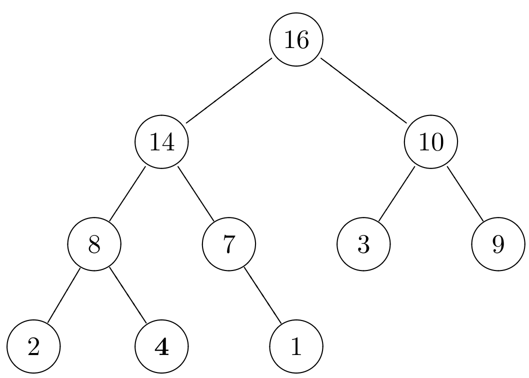
נניח, ניקח את העץ הקודם ונרצה לשמור על תכונת הערימה:



נשווה כעת כל קודקוד לשני בנים. תת העץ הימני שומר על תכונת הערימה, אולם כעת $14 < 4$, לכן בפרט הוא איננו שומר על תכונת הערימה, ונחליף אותו (באיבר המקסימלי):



כעת, שוב, $8 < 4$ ולכן נחליף, ונקבל:



בפרט, הרשימה ממוינת.

ובפסאודו קוד:

אלגוריתם 4 $Max-Heapify\left(A,i\right)$

(1) $l=Left\left(i\right)$
(2) $r=Right\left(i\right)$
(3) אם $l\leq heap-size\left[A\right]$ וגם $A\left[l\right]>A\left[i\right]$
(א) $largest=l$
(4) אחרת:
(א) $largest=i$
(5) אם $r\leq heapsize\left[A\right]$ וגם $A\left[r\right]>A\left[largest\right]$:
(א) $largest=r$
(6) אם $largest\neq i$
(א) תחליף את $A\left[i\right]$ ואת $A\left[largest\right]$
(ב) בצע $Max-Heapify\left(A,largest\right)$

ס

נשים לב כי הפעולה תסתיים עד $\log n$ שלבים, כשבכל שלב אנחנו עושים $\Theta\left(1\right)$ פעולות.

במקרה הטוב ביותר, יש לנו עץ מאוזן לגמרי, דהיינו כל תת עץ מאוזן.

במקרה הגרוע ביותר, בו העץ לא מאוזן וכמעט מלא, נקבל חלוקה ל$\frac{2}{3}n$ בצד אחד ו-$\frac{1}{3}n$ בצד השני.

מדוע? אם העץ בגובה h, נוכל להתבונן בכל אחד מתתי העצים (מלבד שורש העץ, והשורה התחתונה). כל אחד יהיה בגובה$h-2$.

לכן,כולל השורה התחתונה, תת העץ הימני יהיה $\left(2^{h-1}-1\right)$ ותת העץ השמאלי שהוא עץ מלא בגובה $\left(h-1\right)$, יהיה $\left(2^{h}-1\right)$.

כעת, נקבל:

$$\frac{\text{מספר בצד שמאל}}{\text{מספר כולל}}=\frac{2^{h}-1}{1+\left(2^{h}-1\right)+\left(2^{h-1}-1\right)}=$$
$$\frac{2^{h}-1}{2^{h-1}\left(2+1\right)-1}=\frac{2^{h}-1}{3\cdot2^{h-1}-1}$$

כעת, נשאיף את $h\rightarrow\infty$ ונקבל:

$$\lim_{h\rightarrow\infty}\frac{2^{h}-1}{3\cdot2^{h-1}-1}=\lim_{h\rightarrow\infty}\frac{2\cdot2^{h-1}-1}{3\cdot2^{h-1}-1}=\frac{2}{3}$$

לכן, במקרה הגרוע ביותר, צד ימין הוא $\frac{1}{3}$ וצד שמאל הוא $\frac{2}{3}$, כנדרש.

אם כך, זמן הריצה יהיה:

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

באמצעות משפט האב ניתן לגלות את הסיבוכיות של $T\left(\frac{2}{3}n\right)+\Theta\left(1\right)$.

$a=1,b=\frac{3}{2},c=0$ ולכן $\frac{a}{b^{c}=1}$. אם כך, מההגדרה של משפט 2, נקבל כי $T(n)=\Theta\left(n^{c}\log_{b}n\right)=\Theta(\log n)$.

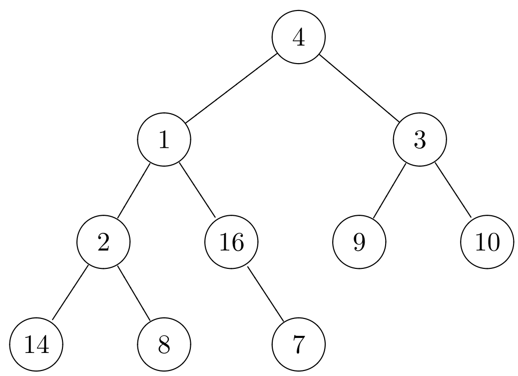
בניית ערמה

ראינו את הצורה הבסיסית של שמירת הערימה. כעת נרצה לבנות ערימה שכזאת.

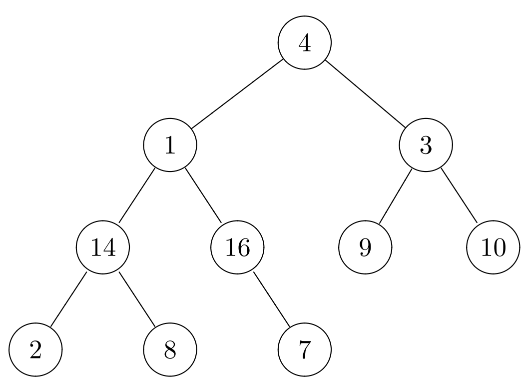
נשתמש באלוגריתם שראינו קודם לכן, על $A\left[1\ldots\left\lfloor\frac{n}{2}\right\rfloor\right]$.
נשים לב כי האיברים בתחילת התהליך הם ערימה בעלי איבר אחד. לכן, נתבונן בשאר האיברים במערך, ונוודא כי הם מקיימים את תכונת הערימה. נרוץ על כל אחד מהאיברים עד $\frac{n}{2}$. כשנסיים בשורש, תכונת הערימה תתקיים עבור כל העץ.
נתבונן למשל בדוגמא הבאה:
ניקח את המערך הבא:

1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

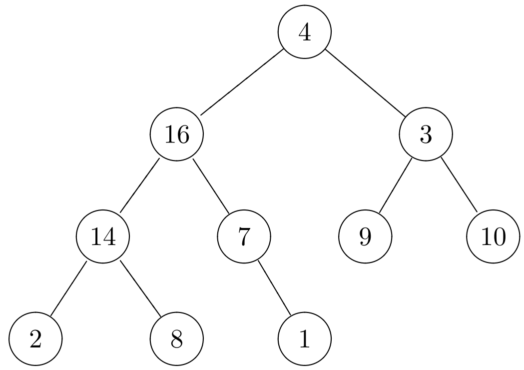
ונסדר אותו בעץ:



לאחר שנבדוק את כל העלים (שהם בהכרח משמרים את תכונת הערימה), נתקדם נגיד ל-2 (בתת העץ השמאלי. אנחנו יכולים לראות כי הוא לא משמר את תכונת הערימה, ולכן נחליף אותו:



וכן הלאה, נמשיך ל-1, ונעבוד רקרוסיבית, וכן הלאה:



אינטאוטיבית, הסברנו מדוע זה נכון. בכל פעם שנשתמש בשיטה הזו, תת העץ כבר מקיים את התכונה, ולכן ניתן לעבור לשלב הבא (מרכיבים דבר שכבר מקיים את התכונה).

מבחינת פסאודו קוד:

אלגוריתם 5 בניית ערימה

---

(1) $heap-size\left[A\right]\leftarrow length\left[A\right]$

(2) מ- $i\leftarrow\left\lfloor \frac{length[A]}{2}\right\rfloor$ עד 1:

(א) בצע $Max-Heapify\left(A,i\right)$

כעת, נצטרך להוכיח זאת פורמלית, באמצעות שמורת לולאה.

הוכחה

עלינו להוכיח כי שמורת הלולאה מתקיימת לפני הלולאה הראשונה, בכל איטרציה של הלולאה, ושעל פיה אפשר להראות נכונות את האלגוריתם ביציאה האחרונה מן הלולאה.

<u>אתחול</u>

לפני הלולאה הראשונה, נקבל כי $i=\left\lfloor \frac{n}{2}\right\rfloor$, ולכן בפרט כל אחד מהאיברים $\left\lfloor \frac{n}{2}\right\rfloor+1,\ldots,n$, הוא עלה, ובפרט הוא שורש של ערמת מקסימום (טריוואלית - בת איבר אחד).

<u>תחזוקה</u>

נשים לב כי כל הבנים של קודקוד i הם אינדקסים גדולים מ-i.

לכן, לפי שמורת הלולאה, הם שורשים על ערמת מקסימום. לכן הקריאה לאלגוריתם על i הופכת את צומת i לשורש של ערמת מקסימום.

בנוסף, הקריאה לאלגוריתם לא פוגעת בעובדה כי כל האינדקסים $i+1,i+2$ וכן הלאה, שורשים.

בכל פעם אנחנו מקטינים את i ב-1, ולכן הלולאה נשמרת גם באיטרציה הבאה.

<u>סיום</u>

בסיום התהליך, $i=1$ ולכן, לפי שמורת הלולאה, כל הצמתים $1,2,\ldots,n$ הם שורשים של ערימות מקסימום.

סיבוכיות ריצה של בניית ערמה

ברור לנו כי כל קריאה לאלגוריתם שמירת הערימה לוקחת $\log n$, והקריאה לוקחת $O(n)$ פעמים, לכן $n\log n$ הוא חסם עליון של האלגוריתם שלנו.
אבל אנחנו יכולים אפילו למצוא חסם הדוק יותר.
הגובה של ערימה בעץ הינו $\log n$, וכאמור לעיל, מספר הצמתים בכל גובה h הינו לכל היותר $\frac{n}{2^{n+1}}$.
על כל צומת מגובה עץ, הפעלת MAX-HEAPIFY לוקחת $O(h)$, ולכן נוכל לבטא את זמן הריצה באמצעות:

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

הסכום האחרון, הינו למעשה מהצורה $\frac{x}{(1-x)^2}=\sum_{k=0}^{\infty}kx^k$ לכל $|x|<1$.
ולכן מתקיים, כאשר $k=h$ ו-$x=\frac{1}{2}$:

$$\sum_{h=0}^{\infty}\left[\frac{h}{2^h}\right]=\frac{\frac{1}{2}}{(1-\frac{1}{2})^2}=2$$

ולכן, נקבל בסך הכל:

..

$$T(n)=O\left(n\sum_{h=0}^{\log n\rceil}\left[\frac{h}{2^{h+1}}\right]\right)=O\left(n\sum_{k=1}^{\infty}\left[\frac{h}{2^h}\right]\right)=O\left(n\cdot 2\right)=O\left(n\right)$$

אם כך, ניתן לבנות ערמת מקסימום ממערך בלתי ממוין, בזמן ליניארי.

האלגוריתם מיון ערימה

במידה ונוכל למצוא את המקסימום, נוכל למיין דרך האלגוריתם הזה. האיבר המקסימלי יהיה הראשון. נשים אותו בסוף המערך. לאחר מכן, נפעיל את האלגוריתם על $n-1$ איברים, ונעביר את המקסימלי למקום אחד לפני האחרון, וכן הלאה:

אלגוריתם 6 מיון ערימה

(1) $Build-Max-Heap\left(A\right)$
(2) לכל $i\to length\left[A\right]$ עד 2
  (א) תחליף את $A\left[1\right]$ ו-$A\left[i\right]$
  (ב) $heapsize\left[A\right]\to length\left(A\right)-1$
  (ג) $Max-Heapify\left(A,1\right)$

זמן הריצה של האלגוריתם מיון ערימה הוא $O\left(n\log n\right)$, כיוון שכל קריאה לבניית העץ המקסימלי היא $O\left(n\right)$ וכל אחת מ-$n-1$ הקריאות של מציאת שורש הערימה, היא $O\left(\log n\right)$.

תורי קדימויות

כיצד נוכל לפתור את הבעיה שהצגנו בתחילת השיעור? נוכל להשתמש בבניית ערימה.
נוכל לקחת את המקסימום בפשטות - נחזיר את האיבר $A\left[1\right]$.

מה אם נרצה לקחת את האיבר המקסימלי ולהקטין את הרשימה?

נצטרך לקחת את האיבר המקסימלי, לשמור אותו, ולנסות להקטין.

נשים את האיבר האחרון במקום האיבר הראשון, ונבצע את האלגוריתם של $Max-Heapify$, ולכן מדובר בזמן ריצה של $\log n$.

מבחינת פסאודו קוד:

אלגוריתם 7 $Heap-Extract-Max(A)$

---

(1) אם אורך הרשימה קטן מ-1

  (א) תחזיר שגיאה

(2) $max\rightarrow A\left[1\right]$

(3) $A\left[1\right]\rightarrow A\left[heapsize\left[A\right]\right]$

(4) $heapsize\left[A\right]\rightarrow heapsize\left[A-1\right]$

(5) בצע $Max-Heapify\left(A,1\right)$

(6) תחזיר את ה-max

מה אם נרצה להעלות איבר כלשהוא במספר? הגדלת המפתח עלולה להרוס את תכונת הערימה, ולכן נצטרך לסרוק מסלול מהצומת, על מנת למצוא מקום מתאים למפתח שהוגדל. אם מפתח שהוגדל קטן יותר משל אביו, לא נחליף אותו והתהליך ייעצר, ואם כן נחליף ונמשיך.

נעצור את האלוגריתם כשנגיע לשורש.

מבחינת פסאודו קוד:

אלגוריתם 8 $Heap-Increase-key(A,i,key)$

---

(1) אם $key<A\left[i\right]$

  (א) תחזיר שגיאה - המפתח החדש קטן מהנוכחי

(2) $A\left[i\right]\rightarrow key$

(3) כל עוד $i>1$ וגם $A\left[Parent\left(i\right)\right]<A\left[i\right]$

  (א) תחליף את $A\left[i\right]$ ואת $A\left[Parent\left(i\right)\right]$

  (ב) $i\rightarrow Parent\left(i\right)$

אם נרצה להכניס איבר חדש, נוסיף עלה חדש לערמה. המפתח החדש יוגדר בתור $-\infty$ (סימון זמני כיוון שאיננו יודעים עדיין מהו ערכו). נקרא ל-$Heap-Increase-key$ על פי המפתח הרצוי.

מבחינת פסאודו קוד:

אלגוריתם 9 $Heap-Insert(A,key)$

---

(1) $heapsize\left[A\right]\rightarrow heapsize\left[A\right]+1$

(2) $A\left[heapsize\left[A\right]\right]\rightarrow-\infty$

(3) הפעל את $Heap-Increase-key\left(A,heapsize\left[A\right],key\right)$

אם כך, ראינו את מימוש ארבעת הפעולות בצורה פשוטה יחסית, כפי שרצינו.

שיעור מס' 7:

יום רביעי

02.11.20

עצי חיפוש בינאריים ועצים מאוזנים

אלגוריתמים שונים בעץ בינארי

מדוע שנרצה עצי חיפוש בינאריים? נניח ונרצה לסדר רשומות (או מפתחות) בסדר עולה, ולבצע עליהם פעולות בסיסייות, כמו חיפוש, מציאת מקסימום, מינימום, האיבר הבא והאיבר הקודם. כיצד נוכל לעשות זאת ב-$\log(n)$ זמן? באמצעות שימוש בעץ בינארי!

נשים לב כי נראה שהעץ יהיה מאוזן על מנת לשמור על זמן הריצה הרצוי (אם העץ איננו מאוזן, ייתכן ונגיע ל-$O(n)$).

הגדרה

עץ בינארי הינו מבנה נתונים, בעל שורש, בן ימני ובן שמאלי, ועלים.

לכל קודקוד בעץ ישנם שלושה מצביעים: לאב, לבן הימני והבן השמאלי. במקרה בו אין אב או אחד מהבנים, הוא יצביע על null.

בהינתן קודקוד x, הגישה למצביעים תתבצע באמצעות $x.right, x.left, x.parent$.

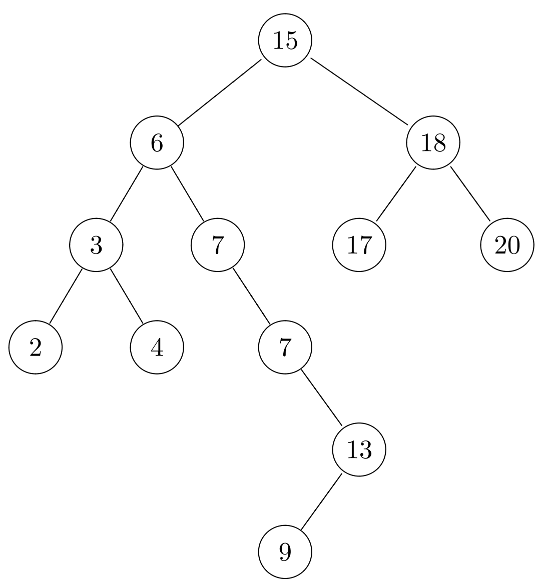
נדרוש כי כי האיברים בתת עץ **השמאלי** יהיו קטנים או שווים לשורש וכל האיברים בתת העץ **הימני** גדולים או שווים לשורש.

נוכל להתבונן בדוגמת העץ הבאה:

נניח שנתון המערך הבא :

2	3	4	6	7	9	13	15	17	18	20

נוכל לסדר אותו בעץ:



נשים לב כי ייתכנו גם עצים שונים.

נתבונן ראשית באלגוריתם **חיפוש איבר בעץ**:

אלגוריתם 10 $Tree - Serach\left(\mathrm{x}, k\right)$

(1) אם $x = null$ או $k = x.key$

(א) תחזיר את x

(2) אם $k < x.key$

(א) תחזיר את $Tree - Serach\left(x.left, k\right)$

(3) אחרת: תחזיר את $Tree - Serach\left(x.right, k\right)$

נוכל לכתוב גם גרסה ב**אמצעות לולאה**:

אלגוריתם 11 $IterTativeTree - Serach\left(k,k\right)$

(1) כל עוד $x \neq null$ או $k \neq x.key$:
  (א) אם $k < x,key$
    (i) $x \to x.left$
  (ב) אחרת: $x \to x.right$
(2) תחזיר את x

בשניהם, הסיבוכיות הינה ב-$O\left(h\right)$.

אלגוריתם למציאת **מינימום**:

אלגוריתם 12 $Tree - Minimum\left(x\right)$

(1) כל עוד $x.left \neq null$
  (א) $x \to x.left$
(2) תחזיר את x

כך גם לגבי מציאת **מקסימום**:

אלגוריתם 13 $Tree - Maximum\left(x\right)$

(1) כל עוד $x.right \neq null$
  (א) $x \to x.right$
(2) תחזיר את x

על מנת למצוא את האיברים בסדר עולה - **ביקור בעץ בסדר העולה** (In Order). ואז נקבל:

אלגוריתם 14 $InOrder - Tree - Walk\left(x\right)$

(1) אם $x \neq null$
  (א) תפעיל את $InOrder - Tree - Walk\left(x.left\right)$
  (ב) תדפיס את x
  (ג) תפעיל את $InOrder - Tree - Walk\left(x.right\right)$

מהו זמן הריצה?

$$T(0) = \Theta(1)$$
$$T(n) = T(k) + T(n-k-1) + \Theta(1)$$

לא קשה להראות כי מדובר ב-$O\left(n\right)$. אינטואטיבית, ברור מדוע זהו זמן הריצה, אנו מבקרים בכל העלים של העץ, וניזכר שמדובר במערך באורך n.

ייתכנו אפשרויות נוספות לביקור בעץ.

כיצד נוכל למצוא את האיבר הבא בעץ? כלומר, את האיבר הבא במערך. נשים לב שכיוון שמדובר בעץ בינארי מדובר בסיטואציה מורכבת יותר. מהי החוקיות כאן?

נחלק למקרים :

(1) למקסימום אין איבר הבא אחריו - ואז התשובה הינה $null$.
(2) מדובר במינימום של תת העץ הימני.
(3) אם אין תת עץ ימני, אז עלינו "לפנות ימינה" בעץ - פעם אחת. ולמצוא את המינימום (דרך "צד שמאל").

במימוש בקוד זה נראה כך - אלגוריתם למציאת **עוקב**:

אלגוריתם 15 $Tree-Successor\left(x\right)$

(1) אם $x.right\neq null$
(א) תחזיר את $Tree-Minimum\left(x.right\right)$
(2) אחרת:
(א) $y\to x.parent$
(ב) כל עוד $y\neq null$ ו-$x=y.right$
(i) $x\to y$
(ii) $y\to y.parent$
(ג) תחזיר את y

פורמלית:
אם x הוא המקסימום, אזי נחזיר $null$ (אין לו איבר עוקב).
אם תת העץ הימני של צומת x אינו ריק, אזי בהכרח האיבר העוקב הימני נמצא בתת העץ הימני, ולכן נפעיל את אלגוריתם המינימום על תת עץ זה.
במקרה השלישי, אם אין לו בן ימני, אזי עלינו לעלות עד לאב, לרדת לתת העץ הקרוב ומשם לאיבר המינימלי.

עד כה, הפעולות עצמן היו סטטיות וכולן לקחו $O\left(h\right)$. אך מה יקרה כאשר נרצה להכניס איבר? תחילה עלינו למצוא היכן למצוא את האיבר. רק לאחר מכן, נצטרך לאזן את העץ.
אם כן, נבצע חיפוש בעץ, עד שנגיע לקודקוד המתאים.

אלגוריתם 16 $Tree-Insert\left(T,z\right)$

(1) $y\to null$
(2) $x\to T.root$
(3) כל עוד $x\neq null$:
(א) $y\to x$
(ב) אם $z.key<x.key$
(i) אזי $x\to x.left$
(ג) אחרת:
(i) $x\to x.right$
(4) $z.parent\to y$
(5) אם $y=null$:
(א) $T.root\to z$
(6) אם $z.key<y.key$:
(א) $z\to y.left$
(7) אחרת:
(8) $z\to y.right$

נשים לב כי **מחיקת איבר** מורכבת יותר, שהרי אנחנו משאירים מצביעים 'יתומים'.
נחלק למקרים:

(1) אם לאיבר שנרצה להוציא אין ילדים, אין לנו כלל בעיה.
(2) אם יש לאיבר רק בן אחד, גם במקרה זה לא ניתקל בבעיה, שכל נוכל לעלות את הבן ימינה (ועדיין אנחנו משאירים את העץ תקין).
(3) אם יש לאיבר הרצוי שני בנים, דבר זה מורכב יותר. נרצה לחפש מהו המספר שיחליף את האב. נצטרך את האיבר "שמיד אחרי האיבר שנרצה למחוק". דבר זה נוכל למצוא באמצעות האלגוריתם למציאת האיבר הבא.

תחילה, נגדיר אלגוריתם המחליף שני תתי עצים:

אלגוריתם 17 $Transplant\left(T,u,v\right)$

(1) אם $u.parent=null$:
  (א) $v\rightarrow T.root$
(2) אם $u=u.parent.left$
  (א) אזי $v\rightarrow u.parent.left$
(3) אחרת: $v\rightarrow u.parent.right$.
(4) אם $v\neq null$:
  (א) $u.parent\rightarrow v.parent$

כעת נוכל לגשת לאלגוריתם המרכזי:

אלגוריתם 18 $Tree-Delete\left(T,z\right)$

(1) אם $z.left=null$:
  (א) תפעיל את $Transplant\left(T,z,z.right\right)$
(2) אם $z.right=null$:
  (א) תפעיל את $Transplant\left(T,z,z.left\right)$
(3) אחרת:
  (א) $Tree-Minimum\left(z.right\right)$
(4) אם $y.parent\neq z$:
  (א) אזי תפעיל את $Transplant\left(T,y,y.right\right)$
  (ב) $y.right\rightarrow z.right$
  (ג) $y.right.parent\rightarrow y$.
(5) תפעיל את $Transplant\left(T,z,y\right)$
(6) $y.left\rightarrow z.left$
(7) $y.left.parent\rightarrow y$.

מבחינת זמני ריצה, מקרים 1 ו-2 לוקחים זמן קבוע - $O\left(1\right)$. במקרה השלישי נקבל $O\left(h\right)$.

טענה

אם ניקח עץ בינארי אקראי (פרמוטציה כלשהיא של עץ בינארי) אזי הגובה הממוצע של העץ הוא $O\left(\log n\right)$. (ראה הוכחה בספר)

איזון עצים

נשים לב כי במידה והעץ איננו מאוזן, זמן הריצה עלול להיות גרוע. לכן נרצה למצוא שיטה לאזן את העץ.
כיצד נוכל לעשות זאת? נשתמש בעצי AVL. [10]
אם כך, כאשר העץ מתחיל לצאת מאיזון, נרצה כי הגובה של העץ לא יהיה גדול מ-$\lfloor log(n) \rfloor$. בנוסף, נדרוש כי פעולת האיזון תקח במקסימום $\log(n)$ זמן.
הכלל אצלנו יהיה כי אם קיים עץ כלשהוא, עלינו לוודא כי לא יהיו יהיה הפרשי גובה גדולים מ-1.
הפרש גובה הינו למעשה "ההפרש בין הצד הימני ובין הצד השמאלי".
ובצורה פורמלית:

הגדרה
הפרש גובה הינו $|x.right.height - x.left.height|$.

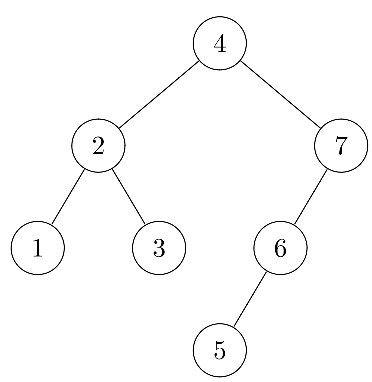
נראה כי אם נשמור על תכונת האיזון של העץ, אזי גובה העץ יהיה $h = \lfloor \log n \rfloor$, ולכן הפעולות כולן תיקחנה $\Theta(\log n)$ זמן.

אם כך, כיצד לאזן את העץ?
המקרה **הטוב ביותר**, הוא כאשר פעולת האיזון אורכת $\log n$.
פעולה אחת לוקחת מעט יותר מ-$\log n$ אבל שאר הפעולות פחות מ-$\log n$, ולכן בסך הכל דבר זה ייקח $\log n$ זמן.
נשים לב כי **איזון מושלם** ידרוש זמן פעולה יקר, ולכן נרצה גובה העץ שאיננו **אופטימלי**, אבל קרוב לזה.

מקור השם AVL הינו בשמם של מחברם - Adelson − Velskii ו-Landis.
נדרוש את התנאים הבאים:

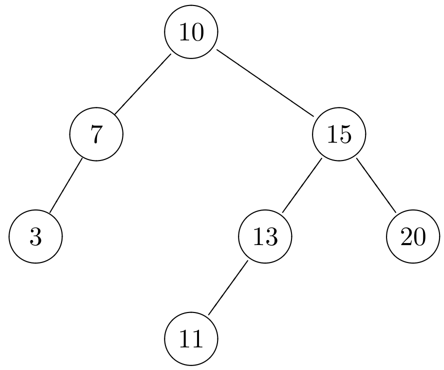
(1) העץ איננו ריק.
(2) תתי העצים יהיו גם הם עץ AVL.
(3) ההפרש בין כל תת עץ ימני ושמאלי יהיה במקסימום 1.

דוגמא לעץ שאיננו עץ AVL:



ועץ שהינו עץ AVL:

[10] בספר מופיעים עצי שחור- אדום, אך אנחנו נלמד על עצי AVL.



נשים לב כי נוכל להגדיר את ההבדל בין העצים באמצעות hd (לכל עלה יש הפרש 0).
מדובר בפעולה רקוריסיבית, שכן $hd\left(x\right)$ של עלה הוא 0, ומצד שני, ככל שעולים בעץ, הפרשי הגובה הינה ביחס לתתי העץ הימניים והשמאליים.

טענה

הגובה של עץ AVL עם $n>0$ קודקודים, הוא $\log n$.
למעשה, החסם של גובה עץ כזה הינו:
$\log_{\varphi}(\sqrt{5}(n+2))-2\approx1.44\log_{2}(n+2)-0.328$ כאשר φ זהו יחס הזהב.
אם כך, העץ הוא פי 1.5 יותר ארוך מאשר עץ מאוזן בצורה אופטימלית.

הוכחה

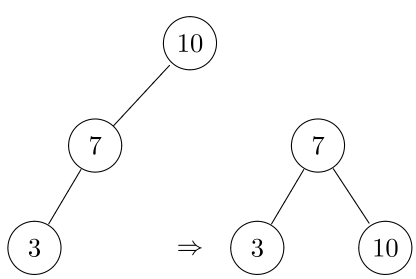
בתרגול.

מדוע? נסו למצוא את **הפרשי הגובה!**
נוכל להגדיר לכל קודקוד x את הפרשי הגובה, שנסמנו בתור $hd\left(x\right)$.

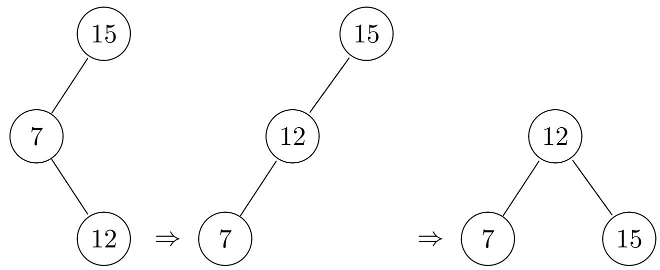
פעולת איזון מחדש.

נבצע את פעולת ההכנסה או המחיקה, ונצטרך לחשב מחדש את הפרשי הגובה.
פעולת האיזון מחדש נקראת "סיבוב", ונבין מדוע דבר זה נקרא כך.

אינטואטיבית, דבר זה מתקיים כך:



לעיתים לא נוכל לבצע זאת, ולכן נצטרך לעשות "סיבוב שמאלה"[11] ורק אז סיבוב ימינה:



כעת, אם יש לנו תת עץ ששורשו x , והפרש הגובה בצד שמאל. נוכל להוסיף קודקוד בארבעת האפשרויות הבאות:

(1) הכנסה לתת העץ השמאלי של הילד השמאלי של x.

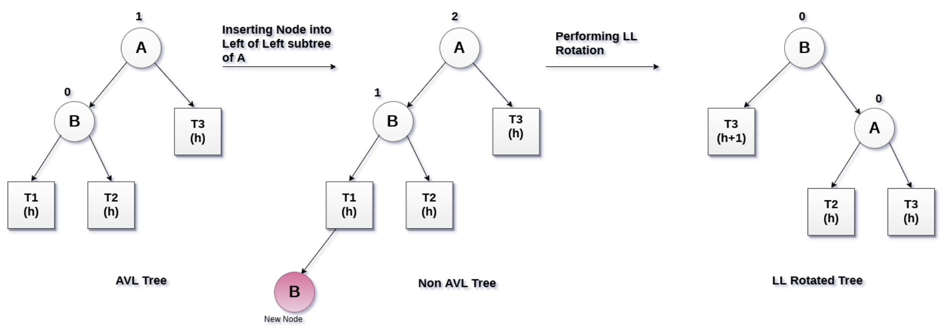
(2) הכנסה לתת העץ הימני של הילד השמאלי של x.

(3) הכנסה לתת העץ השמאלי של הילד הימני של x.

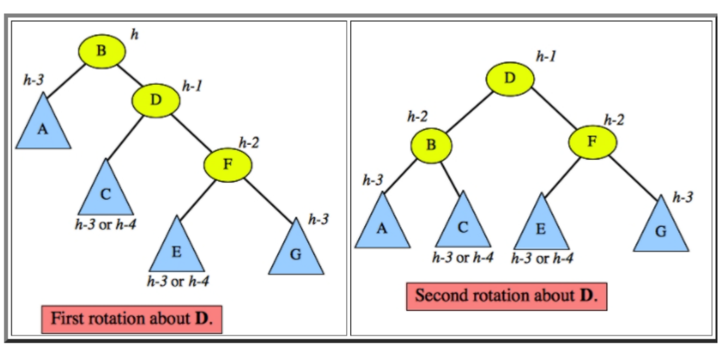
(4) הכנסה לתת העץ הימני של הילד הימני של x.

נשים לב כי מקרים 1 ו-4 הינם שקולים (דורשים סיבוב יחיד) וגם 2 ו-3 הינם שקולים (דורשים סיבוב כפול).

לבסוף, נוכל להכליל זאת למקרים גדולים יותר:



באילסוטרציה, זה נראה כך:



[11] תצטרכו דמיון רציני בשביל לראות שיש כאן סיבוב.

שיעור מס' 8:
יום רביעי
09.12.20

גרפים

הגדרות, מונחים וטרמינלוגיה

מוטיבציה

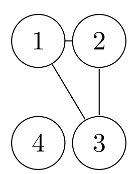
גרפים מאפשרים לנו לתאר קשרים בין קודקודים. היחס יכול להיות **סימטרי** (a מכיר את b וגם ההיפך) או לא סימטרי (a מכיר את b אבל b לא מכיר את a).
הגרף מתאר את היחסים **הבינאריים** בין היישויות האלו.
נרצה לשאול האם קיים מסלולים בין הקודקודים המתארים את יישויות אלו - כביכול האם הם 'מכירים' זה את זה.

הגדרות ומונחים

גרף G הוא זוג סדור (V,E) כאשר $V=\{v_1,\dots v_n\}$ הן קבוצות הקודקודים, ו-$E=\{e_1,\dots,e_m\}$ הן הצלעות.
כל $e_k\in E$ המוגדרת על ידי $e_k=e_{ij}$ מחברת שני קודקודים (v_i,v_j).
אם V וE הן קבוצות סופיות, אזי גם G סופי.

גודל הגרף הוא סכום הקודקודים והצלעות. כלומר $|G|=|V|+|E|$.

נוכל להתבונן למשל בדוגמא הבא:



מדובר בגרף שאיננו **מכוון**, כאשר הצלעות הינן $E=\{(1,2),(2,3),(3,1),\dots\}$.

שכנים בגרף

אם ישנה צלע בין u,v, אזי הם נקראים **שכנים**.
בגרף **מכוון** אין מעגלים. כלומר אין $e\in E$ כך ש-$e=(u,u)$.
שכנות של קודקודים היא **סימטרית.** כלומר, אם $e=(u,v)$ אזי u שכן של v ו-v שכן של u.
דרגה של כל קודקוד היא מספר ה**שכנים** של קודקוד.
סכום הקודקודים הוא $\sum d(v_i)=2|E|$.

בגרף **מכוון** ישנה כיווניות ולכן ייתכנו מעגלים. כיוון שהכיווניות חשובה, נגדיר **דרגת כניסה** $d_{in}(v)$ שמגדירה את מספר הצלעות ה'נכנסות', **ודרגת יציאה** שמגדירה את מספר הצלעות ה'יוצאות'. מתקיים למעשה כי $\Sigma d_{in}(v_i)=\Sigma d_{out}(v_i)=|E|$

מסלולים בגרף

יהיו $u, v \in V$. מסלול (u, v) הוא סדרה $(v_0, v_1, \dots v_k)$, כאשר $u = v_0$ ו-$v = v_k$, כאשר לכל $1 \le i \le k$ מתקיים כי $(v_{i-1}, v_i) \in E$.

זהו לא חייב להיות המסלול הקצר ביותר.

אורך המסלול הוא מספר הקודקודים בסדרה.

ייתכן ולא יהיה מסלול, ואז האורך לא מוגדר. בנוסף, ההגדרה תקפה לגרף מכוון וגם לגרף לא מכוון.

גרף משוקלל

נניח כי לכל צלע בגרף ישנו משקל, $w\left(v_i, v_j\right) > 0$.

נאמר כי גרף הינו משוקלל אם כל המשקלים של הצלעות הינן 1.

שני קודקודים בלי מסלול ביניהם, הינן עם משקל אינסופי.

המשקל של מסלול הוא סכום המשקלים של הצלעות:

$$w(path(u, v)) = \sum_{i=0}^{k} m\left(v_{i-1}, v_i\right)$$

כאשר $u = v_0$ ו-$v = v_k$.

סוגים של גרפים

מעגל הוא מסלול שמגיע מקודקוד לעצמו, באורך של לפחות 1.

קשירות בגרף היא העובדה שניתן להגיע מכל קודקוד לכל קודקוד.

גרף קשיר הוא גרף בו יש מסלול בין כל שני קודקודים בגרף.

תת גרף הוא תת קבוצה G' כך ש-(V', E'), כאשר $G' \subseteq G$ ו-$V' \subseteq V$ וגם $E' \subseteq E$.

ייתכן ותת גרף יהיה לא קשיר, למרות שהגרף יהיה קשיר.

רכיבי קשירות G_1, G_2 של גרף G, הם תתי הגרפים הקשירים הגדולים ביותר בגרף.

מספר הקודקודים בגרף

בגרף, מספר הקודקודים הוא במקסימום $|E| = O\left(V^2\right)$ קודקודים.

גרף בו יש $|E| = O\left(V^2\right)$ קודקודים ייקרא **קליקה**.

בנוסף, מתקיים כי לפחות $|E| \ge |V| - 1$ בגרף. (בעץ יש בדיוק $|E| = |V| - 1$צלעות)

גרף פלנרי הוא גרף בו אין שתי קשתות החוצות אחת את השנייה. ניתן להראות שהגרף האי פלנרי היחיד, הוא כאשר יש 5 צלעות.

קשר בין גרפים ועצים

לעץ יש בדיוק $|E| = |V| - 1$ צלעות.

בכל גרף G ארבעת התנאים הבאים שקולים:

(1) G הוא עץ.

(2) ל-G אין מעגלים. הוספת צלע מוסיפה מעגל.

(3) G קשיר, ומחיקת צלע מאבדת את הקשירות.

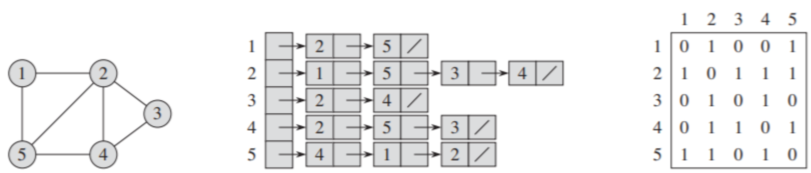
(4) G חסר לולאות עצמאיות ויש מסלול בין כל שני קודקודים.

מבני נתונים ואלגוריתמים נפוצים

מבני נתונים

ישנן שתי צורות להצגת גרף באמצעות מבני נתונים. באמצעות מטריצות וברשימות.
מבחינת רשימה, ישנה רשימה מקושרת בין כל השכנים בגרף. הסדר ברשימה איננו משנה (יש רק את רשימת השכנים),
גודל הייצוג הינו $\Theta\left(|V|+|E|\right)$ - ליניארי לפי מספר הקודקודים ומספר הקשתות.

נוכל לייצג זאת גם באמצעות מטריצה בגודל $|V|\times|V|$, כאשר כל צלע $e=(u,v)$ מיוצגת ע"י 1 אם היא קיימת.
ניתן לראות דוגמא לגרף שאיננו מכוון כאן:



(a) (b) (c)

הערה

במקרה של מעט קשתות, עדיף להשתמש בייצוג בתוך רשימה מקושרת, ואם יש הרבה קשתות עדיף להשתמש במטריצה.

בעיות גרפים נפוצות

ישנן שתי צורות נפוצות לחיפוש מסלול בגרף:
חיפוש **לרוחב של הגרף.** מקובל לקרוא לו BFS.
חיפוש **לעומק של הגרף.** מקובל לקרוא לו DFS.

קיים גם אלגוריתם למציאת **רכיבי הקשירות בגרף -** SCC **ולמציאת עץ פורש מינימלי** (רכיב קשירות אחד, כך שסכום המשקלים מינימלי).
כמו כן, ישנם אלגוריתמים למציאת מסלול מינימלי, אחד או יותר, או מציאת המסלולים הקצרים בין כל הקודקודים בגרף.
לפי סדר זה גם נלמד בשבועות הקרובים.

מציאת מסלולים בגרף

למעשה, אנחנו כבר מכירים בצורה אינטואטיבית את האלגוריתמים של חיפוש בגרף, באמצעות חיפוש מסלולים במבוך (התקדמות לפי הקודקודים, או חיפוש לפי הדלתות).
קיימים שלושה סוגי של בעיות מסלולים:

(1) מציאת מסלול בין שני קודקודים.
(2) מציאת כל המסלולים בין קודקוד אחד לשאר הקודקודים.
(3) מציאת כל המסלולים הקצרים בין כל שני קודקודים בגרף.

נתחיל מהבעיה הקלה, וראשית נמצא איך מוצאים מסלול, לאו דווקא הקצר ביותר.

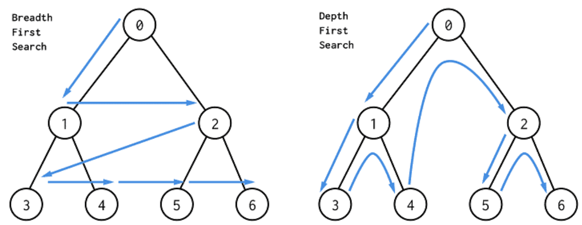
כיצד נוכל לחפש בגרף?

נתחיל ב'רמה הנמוכה' - אם מצאנו את הקודקוד - סיימנו. אחרת, נמשיך לשכנים שלו. למעשה, בכל רמה i, נחפש את הרמה ה-$i+1$. הרמה מוגדרת על ידי "הפעם הראשונה" שפגשנו את הקודקוד. אם מצאנו את הקודקוד, נסמן את כל "הדלתות" שדרכם נכנסו, ונסמן במסלול על ידי $(0,\ldots,i-2,i-1,i)$.

נוכל לראות כי שיטה זו שמדובר במסלול הקצר ביותר.

אפשר לחפש **לרוחב** - מעבר על כל הקודקודים ברמה ה-i, לפני הקודקודים ברמה ה-$i+1$, או **לעומק** - בדיקה של קודקוד ברמה ה-i, הגעה לשכן שלו, עד להגעה לעלה. חזרה רמה אחורה ושוב בדיקה עד העלה.

ניתן לראות הבדל בין שתי שיטות החיפוש כאן:



חיפוש לרוחב

נבצע את החיפוש בעץ, ללא בנייה שלו בפועל.

נוכל לחשוב על זה בתור "גלי הדף" - בכל פעם נתפשט עוד במורד הגרף.

נבנה את האלגוריתם אינטואטיבית. בהינתן "גל הדף" כלשהוא, נוכל לחלק לכמה סוגי קודקודים - קודקודים **שביקרתי** אותם ואת כל שכניהם (visited), קודקודים **שעדיין לא בדקתי** את כל שכניהם (current), וקודקודים **שלא ביקרנו** אותם (not-visited).לכל קודקוד נוסיף שדה המוודא באיזו סוג הוא נמצא.

אם כך, נשמור לכל קודקוד את השדה - של "האם ביקרתי" (label), "שדה של מרחק מהקודקוד ההתחלתי" - (dist) ושדה של ה"הקודקוד הקודם" (π).

נקבל את האלגוריתם הבא:

אלגוריתם 19 $BFS\left(G,s\right)$

---

(1) $s.label\rightarrow current$

(2) $s.dist\rightarrow0$

(3) $s.\pi\rightarrow null$

(4) לכל הקודקודים u ב-V מלבד s:

    (א) $u.label\rightarrow not\_visited$

    (ב) $u.dist\rightarrow\infty$

    (ג) $u.\pi\rightarrow null$

(5) $EnQueue\left(Q,s\right)$

(6) כל עוד Q איננו ריק:

    (א) $u\rightarrow DeQueue\left(Q\right)$

        (i) לכל v בשכנים של u, אם $v.label=not\_visited$:

            (א') $v.label\rightarrow current$

            (ב') $v.\pi\rightarrow u$

            (ג') $v.dist\rightarrow u.dist+1$

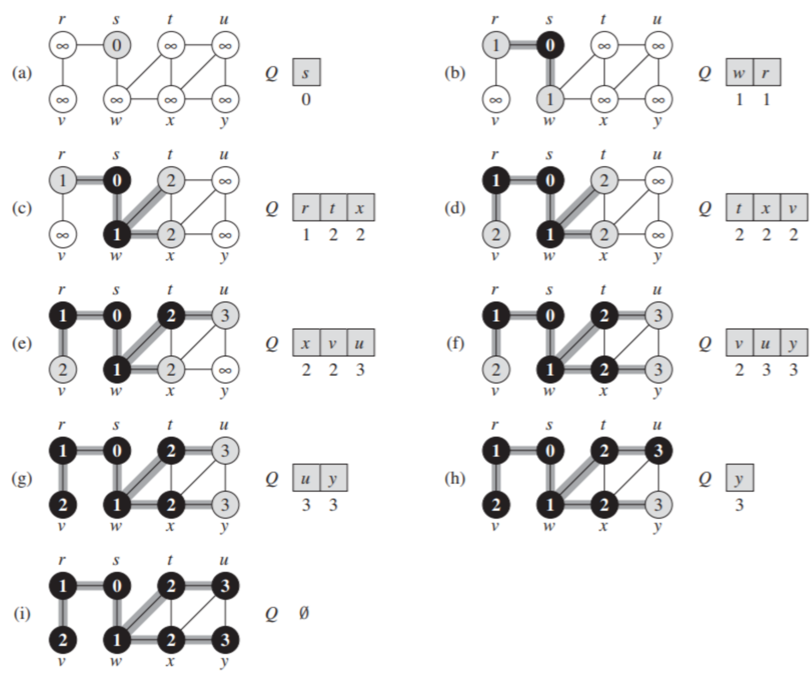
            (ד') $EnQueue\left(Q,s\right)$

        (ii) $u.label\rightarrow visited$

נבחין כי Q מכיל רק קודקודים שהם $current$. ברגע שקודקוד הופך ל-$current$ או $visited$, הוא לעולם לא יהפוך שוב ל-$not\_visited$. בנוסף, ברגע שכל שעברנו אצל כל השכנים של קודקוד, הוא הופך ל- $visited$. האלגוריתם בנוי כך שנפסיק אותו כשנגיע ל-t. למעשה, האלגוריתם בונה את את תת הגרף המוביל ל-t כעץ כך ש:

$$V_\pi = \{v \in V \mid v.\pi \neq null\}, E_\pi = \{v \in V \setminus \{s\}, (v.\pi, v)\}$$

מדובר בעץ, שכן לכל אחד מהקודקודים אב אחד והצלחנו להגיע ל-t כפי שרצינו.
באילסטורציה, זה נראה כך:



זמני ריצה

בכל פעם אנחנו מורידים קודקוד אחד - כלומר אין חזרות. אם כך, קיבלנו כי יש לכל היותר $|V|$ חזרות של $DeQueue$. עבור כל קודקוד הנחנו עוברים על כל השכנים ומבצעים מספר קבוע של פעולות - $O(E)$ (זהו זמן סריקת רשימות סמוכות).
העבודה על ה-if הפנימי הינה בזמן קבוע של מספר הצלעות היוצאות, לכן אנחנו מקבלים בכל הקודקודים בחלק זה את העבודה כמספר הצלעות.
לכן נקבל מבחינת זמני ריצה: $O(V+E)$.

נכונות האלגוריתם

כדי להוכיח נכונות, עלינו להשוות את המרחק שחישבנו ולגלות האם הוא המרחק הקצר ביותר.

יהי $G(V,E)$ ויהי $s \in V$. נגדיר $\delta(s,v)$ **כמסלול הקצר ביותר** מ-s ל-v. עלינו להראות כי האלגוריתם אכן מביא את מסלול זה.

למה

יהי $G\left(V,E\right)$ גרף מכוון או לא מכוון, ויהי $s\in V$. עבור כל קשת $\left(u,v\right)\in E$ מתקיים:

$$\delta\left(s,v\right)\leq\delta\left(s,u\right)+1$$

הוכחה

אם ניתן להגיע מ-s ל-u ניתן להגיע גם מ-s ל-v, אזי המסלול הקצר ביותר מ-s ל-v לא יכול להיות ארוך יותר מהמסלול הקצר ביותר מ-s ל-u ואחריו הקשת $\left(u,v\right)$.

למה

יהי $G\left(V,E\right)$ גרף מכוון או לא מכוון ונניח שהפעלנו את BFS על G מ-$s\in V$ כלשהוא. אזי, לאחר סיום האלגוריתם, עבור כל קודקוד $v\in V$ מתקיים כי $v.dist\geq\delta\left(s,v\right)$.

הוכחה

באינדוקציה על כמות הפעולות של הכנסה לתור.

<u>בסיס האינדוקציה</u>

כאשר הכנסנו את s לתור, מתקיים כי $s.dist=0=\delta\left(s,s\right)$, וגם לכל קודקוד $s\neq v\in V$ מתקיים כי $v.dist=\infty\geq\delta\left(s,v\right)$.

<u>צעד האינדוקציה</u>

נתבונן בקודקוד v שהינו $not\_visited$, שהגענו אליו מ-u. מהנחת האינדוקציה עולה כי $u.dist\geq\delta\left(s,u\right)$. כעת נקבל כי:

$$v.dist=u.dist+1\geq\delta\left(s,u\right)+1\geq\delta\left(s,v\right)+1$$

(בספר ישנן עוד מספר למות שלא הוכחנו).

משפט (נכונותו של חיפוש לרוחב)

יהי $G\left(V,E\right)$ גרף מכוון או בלתי מכוון, ונניח שהפעלנו את BFS על G מ-$s\in V$. אזי האלגוריתם מאפשר לגלות כל קודקוד $v\in V$ שאפשר להגיע אליו מ-s.
בסוף ביצוע האלגוריתם מתקיים כי $v.dist\geq\delta\left(s,v\right)$ לכל $v\in V$. מעבר לכך, לכל קודקוד $s\neq v$ שניתן להגיע אליו מ-s, מתקיים כי אחד המסלולים הקצרים ביותר הוא המסלול הקצר ביותר מ-s ל-$v.\pi$ ואחריו הקשת $\left(v.\pi,v\right)$.

הוכחה

תסתכלו בספר.

שיעור מס' 9: **יום רביעי** 16.12.20

חיפוש לעומק

מוטיבציה

נרצה להשתמש בחיפוש לעומק למספר שימושים. בהמשך השיעור נשתמש ב"מיון בתוך גרף" - יצירת סדר בין רכיבים בגרף מכוון. בנוסף, נשתמש בחיפוש לעומק על מנת למצוא את רכיבי הקשירות בתוך גרף. בשיעור הבא ניגע גם בעצים פורשים.

הרעיון של חיפוש לעומק הינו כזה - נמשיך בגרף לאורך השכנים, עד שנגיע לקודקוד שסיימנו לעבור על שכניו, ונחזור אחורה, נתקדם לקודקוד הבא, עד שנגיע למצב בו אין שכנים, ונחזור לקודקוד הראשון. בצורה כזאת, נבקר בכל הקודקודים ברכיב קשירות אחד. מדובר במצב של "עומק" כי אנחנו מתקדמים עם השכנים, עד שחוזרים אחורה.

בדומה ל-BFS, גם DFS משתמשים ב-$visited, current$ ו-$not\_visited$. נוסיף כעת גם שתי שדות נוספות, d ו-f, שמתארים את **הזמן בו הגענו אליו** (מתי שהתגלה -discoverd), **והזמן בו סיימנו לעבור על שכניו** (מתי שנגמר -finished), בהתאמה (הזמן נע מ-1 עד $2|V|$).
מטבע הדברים, בהינתן קודקוד u, יתקיים כי $u.d < u.f$ (אם אין לו שכן יתקיים כי $u.f = u.d + 1$).
בפסאודו קוד, זה נראה כך:

אלגוריתם 20 $DFS\,(G)$

---

(1) לכל קודקוד $u \in V$
  (א) אם $u.label = not\_visited$:
    (i) תבצע $DFS - Visit\,(u)$

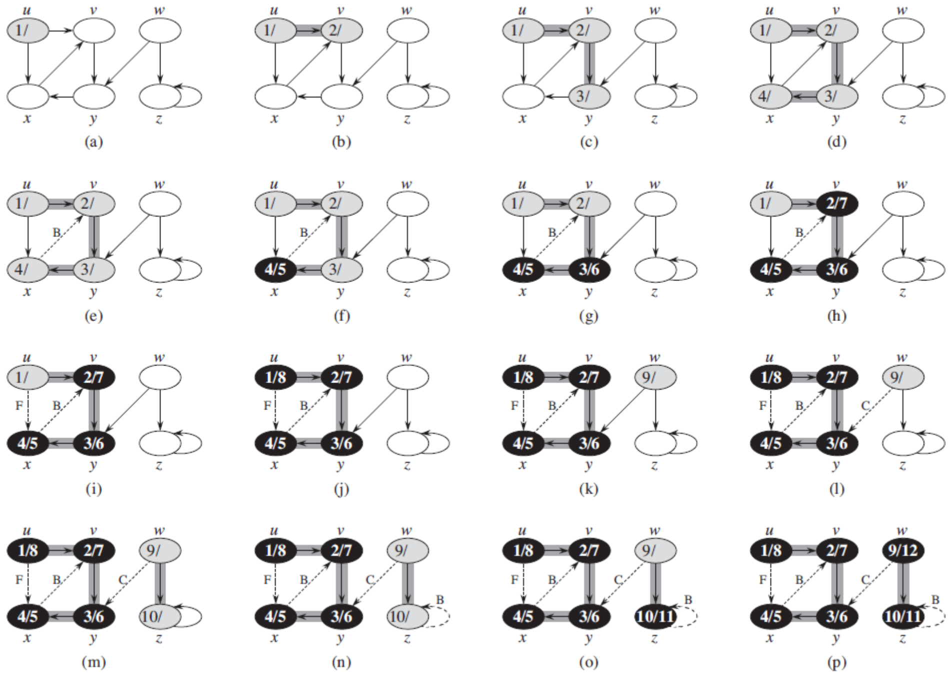
ו'פונקציית העזר':

אלגוריתם 21 $DFS - Visit\,(u)$

---

(1) $u.label \to current$.
(2) $time \to time + 1$.
(3) $u.d \to time$
(4) לכל קודקוד ב-v אם הוא שכן של u:
  (א) אם $v.label = not\_visited$:
    (i) $v.\pi \to u$
    (ii) $DFS - visit\,(v)$
  (ב) $u.label \to visited$
  (ג) $u.f \to time$
  (ד) $time \to time + 1$

ניתן לראות דוגמה לאלגוריתם כאן:



סיבוכיות

במהלך האלגוריתם, מבקרים בכל צלע פעמיים, כי כיוון שסיימנו עם קודקוד ושכניו, לא נחזור אליו. אם כך, הקריאה הרקורסיבית נעשית פעם אחת לכל צלע, ואם כך הסיבוכיות היא כמספר השכנים, דהיינו $\Theta\left(|E|\right)$.
אם כך, לסיכום, זמן הריצה הינו $\Theta\left(|V|\right)+\Theta\left(|E|\right)=\Theta\left(|V|+|E|\right)$.

תכונות

ישנם מספר סדרים אפשריים של עצים, כיוון שיש תלות באיזה שכן בחרנו.
בנוסף, אם נפעיל את האלגוריתם על כל הקודקדים, נוכל לקבל את כל רכיבי הקשירות ולמעשה "יער".

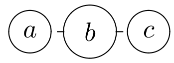
נוכל לשים לב כי נכנסנו לקודקוד אחד, קיבלנו כי:

$$\begin{array}{ccccc} & 1 & 2 & & \\ a & (a & a) & \rightarrow & () \end{array}$$

פתחנו סוגריים אחד וסגרנו.
בצורה דומה, אם ניכנס לשני קודקודים, נקבל:

$$\begin{array}{ccccccc} & 1 & 2 & 3 & 4 & & \\ a - b & (a & (b & b) & a) & \rightarrow & (()) \end{array}$$

ואם ניכנס לשלושה קודקודים:



$$\begin{array}{cccccccc} 1 & 2 & 3 & 4 & 5 & 6 & & \\ (a & (b & (c & c) & b) & a) & \rightarrow & ((())) \end{array}$$

כמו כן, ייתכנו דוגמאות נוספות (בלתי אפשרי לסגור סוגריים מבלי שסגרנו את הסוגריים הפנימיים יותר).
בצורה דומה, אם אין קשר בין הסוגריים, ישנם למעשה שני רכיבי קשירות.

משפט הסוגריים

בכל חיפוש לעומק של גרף $G=(V,E)$ - מכוון ולא מכוון, עבור כל שני קודקודים $u,v\in V$, מתקיימים אחד משלושת התנאים הבאים:

(1) הקטעים $[u.d,u.f]$ ו-$[v.d,v.f]$ זרים (כלומר, המסלולים שלהם לא מוכלים אחד בשני).
(2) הקטע $[u.d,u.f]$ מוכל בקטע $[v.d,v.f]$ (מסלול אחד מוכל בשני).
(3) הקטע $[v.d,v.f]$ מוכל בקטע $[u.d,u.f]$.

הוכחה

אם $u.d<v.d$ ישנן שתי אפשרויות:
המקרה הראשון - $v.d<u.f$. דבר זה אומר כי גילינו את u קודם, ולכן נובע כי v הוא צאצא של u. מעבר לכך, כיוון ש-v התגלה אחרי u, הטיפול בו מתקיים לפני שחזרנו ל-u. אם כך, נובע כי $[u,d,u.f]$ מוכל בקטע $[v,d,v.f]$.

במקרה השני - $u.f<v.d$ בהכרח מתקיים כי אמנם ישנו ייתכן קודקוד משותף, אבל שני הקטעים זרים.

המקרה השני $v.d<u.d$ סימטרי.

מסקנה

v הוא צאצא ממש של קודקוד u, אם ורק אם $u.d<v.d<u.f<u.f$.

טענה

בחיפש לעומק של גרף $G=(V,E)$, מתקיים כי v הוא צאצא של u אם ורק בזמן $u.d$ ניתן להגיע לקודקוד v דרך קודקודי $not\_visited$.

הוכחה

בכיוון הראשון, נניח כי v הוא צאצא של u. אזי בפרט מתקיים כי $u.d<v.d$ (גילינו אותו בשלב מאוחר). כלומר, יתקיים כי w היה $not\_visited$ בזמן $u.d$, כנדרש.

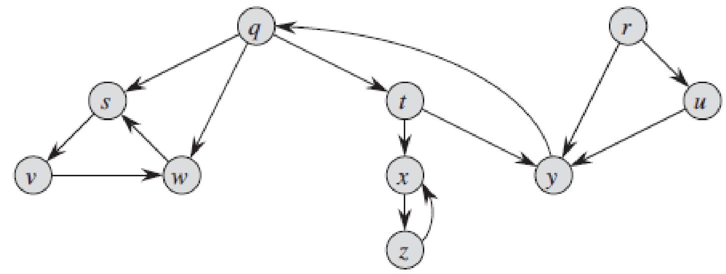
בכיוון השני, נניח בשלילה כי אין מסלול כפי שתיארנו במשפט. נניח כי הוא נמצא באמצע - דהיינו איבר w הוא האב של v, וצאצא של u.
אזי מתקיים כי $w.f\leq u.f$, ולכן מהטענה הקודמת מתקיים כי $u.d<v.d<w.f\leq u.f$. אבל מדובר בסתירה כי דבר זה גורר שהמסלולים שלהם מוכלים זה בזה - כלומר v הוא צאצא של u, בסתירה.

סיווג קשתות של DFS

ראשית, נבחין כי ניתן לסווג מספר סוגי צלעות:

(1) קשתות עץ - קשת רגילה בעת חיפוש לעומק.
(2) קשתות אחורה - קשתות שמחברות קוקדוד u לאב קדמון v.
(3) קשתות קדימה - קשתות שאינן קשתות עץ, שמחברות קדקוד u לקודקוד קדימה.
(4) קשתות חוצות - כל הקשתות האחרות. מקשרות קדקודים באותו עץ, כאשר אחד אינו צאצא של האחר. או קודקודים בעצי עומק שונים.

דוגמא:



משפט

בעת חיפוש לעומק של גרף מכוון $G\left(V,E\right)$, כל קשת קשת עץ היא קשת אחורה.

הוכחה

תהי $(u,v)\in E$ ונניח כי $u.d<v.d$. אזי בהכרח v התגלה והסתיים לפני ש-u הסתיים.
אם הלכנו בכיוון ההפוך, כלומר מ-v ל-u, אזי u היה $not\_visited$ ואם כך למעשה הצלע היא בהכרח קשת אחורה, כי בשלב בו הלכנו מ-v ל-u, u עדיין היה $open$.

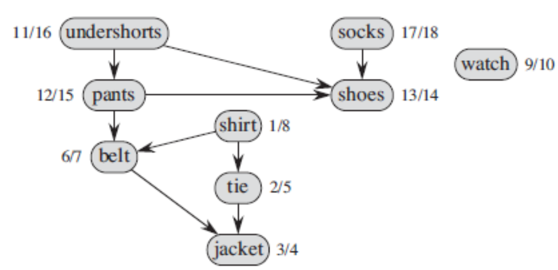
מיון טופולוגי.

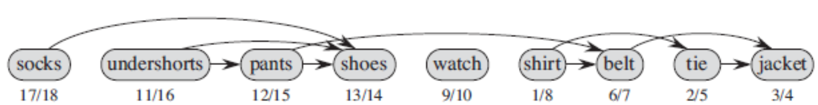
כעת נשתמש בחיפוש לעומק על מנת לבצע מיון טופולוגי של גרף מכוון ללא מעגלים.

הגדרה

מיון טופולוגי של גרף מכוון ללא מעגלים $G=(V,E)$ הוא סידור ליניארי של קודקודי G, כך שאם G מכיל קשת (u,v) אזי u מופיע לפני v בסידור זה.
(כאשר יש מעגלים, לא ניתן ליצור סדר ליניארי)

נוכל להתבונן בדוגמה הבאה:





כלומר, למשל, לא נוכל לעבור לשים חגורה, לפני ששמנו מכנסיים (זה מה שמסמל החץ "קדימה").
מבחינת אלגוריתם, זה נראה כך:

אלגוריתם 22 מיון טופולוגי

(1) $L \rightarrow null$.
(2) לכל $v \in V$:
 (א) תפעיל את $DFS\left(v\right)$ על מנת לחשב את $v.f$.
 (ב) אם v הינו $visited$:
  (i) תכניס אותו לראש הרשימה המקושרת.
 (ג) תחזיר את הרשימה המקושרת.

למה

גרף מכוון G אינו מכיל מעגלים, אם ורק אם לאחר חיפוש לעומק של G לא מצאנו קשתות אחורה.

הוכחה

בכיוון הראשון, נניח בשלילה שקיימת קשת אחורה $\left(u,v\right)$ אזי v הוא אב קדמון של v, ולכן מצאנו מסלול מ-v ל-u וגם מ-u ל-v ולכן מצאנו מעגל, בסתירה לכך שאין מעגלים.

בכיוון השני, נניח בשלילה כי G מכיל מעגל C. כעת יהי v הקודקוד הראשון המתגלה ב-C ותהי $\left(u,v\right)$ הקשת האחרונה ב-C.
כעת, בזמן $v.d$ מצאנו מסלול של $not\_visited$ מ-v ל-u. אם כך, u הוא צאצא של v, כלומר מצאנו קשת אחורה.

משפט

האלגוריתם של מיון טופולוגי, יוצר גרף מכוון ללא מעגלים.

הוכחה

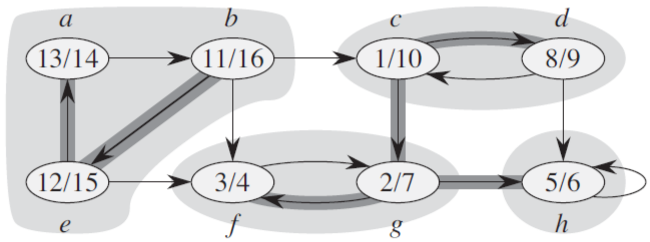
בתרגול.

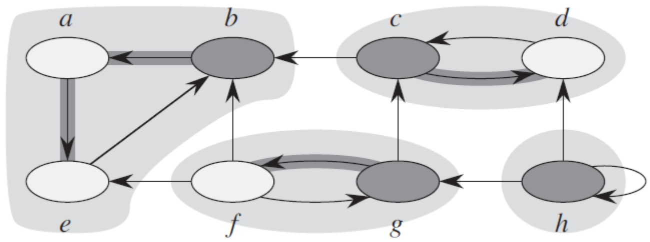
רכיבים קשירים היטב.

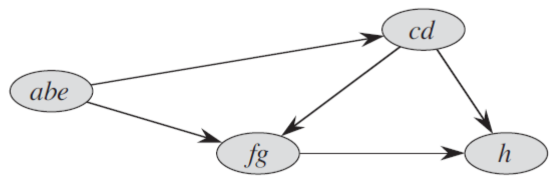
הגדרה

רכיבים קשירים היטב הם רכיבי הקשירות הגדולים ביותר בגרף מכוון $G=\left(V,E\right)$,, כך שלכל $u,v\in C_{i}$ ישנו מסלול בין u ל-v ובין v ל-u.

דוגמאות ניתן למצוא כאן:







הגדרה

הגרף המשוחלף $G^T = \left(V, E^T\right)$ של G הוא הגרף G עם כיוונים הפוכים. כלומר $E^T = \{(u,v) \mid (u,v) \in E\}$.

נרצה לחפש בזמן ליניארי את רכיבי הקשירות בגרף.

הרעיון של האלגוריתם הינו כזה. נבצע DFS בכיוון הראשון, ואז נבצע DFS על הגרף המשוחלף.

אלגוריתם 23 מציאת רכיבי קשירות בגרף

---

(1) תבצע DFS ל-G על מנת לחשב את $v.f$.
(2) תחשב את G^T.
(3) תבצע DFS ל-G^T בסדר יורד של $v.f$

למה 1

אם שני קודקודים נמצאים באותו רכיב קשיר היטב, אז כל מסלול ביניהם לא יוצא מהרכיב קשיר היטב.

למה 2

יהיו C ו-C' שני רכיבים קשירים היטב. כעת, יהיו $(u,v) \in E$ כך ש-$u \in C$ ו-$v \in C'$ אזי $f\left(C\right) > f\left(C'\right)$ כאשר $f\left(C\right) = \max_{u \in C}\left(u.f\right)$ עבור $C \subset G$.

(אינטואיטיבית, מהגדרת DFS, אנחנו מסיימים את C' לפני שאנחנו חוזרים ל-C, ולכן עבור קודקוד u כלשהוא ב-C, ה-f של u יהיה גדול יותר. הטענה לא נכונה עבור מינימום, כי ייתכנו "ענפים" שמסתיימים ב-C לפני שאנחנו מתחילים את C')

משפט

האלגוריתם של מציאת רכיבי קשירות בעץ נכון.

הוכחה

נוכיח זאת באינדוקציה על עצי העומק המתגלים בחיפוש לעומק של G^T.

בסיס האינדוקציה

$k=0$ - אין קודקודים ולכן דבר זה בהכרח נכון.

צעד האינדוקציה

נניח כי האלגוריתם נכון עבור n רכיבי קשירות ונוכיח עבור $n+1$.

לפי הלמה הקודמת, מתקיים כי $u.f=f\left(C\right)>f\left(C'\right)$, עבור רכיב קשירות שלא מצאנו (שהרי אנחנו מחפשים בסדר יורד).

כעת, מהנחת האינדוקציה, כאשר מצאנו את u, כל הקדקודים ב-v הינן $not\_visited$. לכן, לא פגענו באף רכיבי הקשירות הקודמים שמצאנו, והוספנו רק רכיב קשירות אחד, ולכן מצאנו בדיוק את הרכיב הנוסף, ומצאנו את כל רכיבי הקשירות בגרף.

שיעור מס' 10:

יום רביעי

23.12.20

עצים פורשים מינימליים

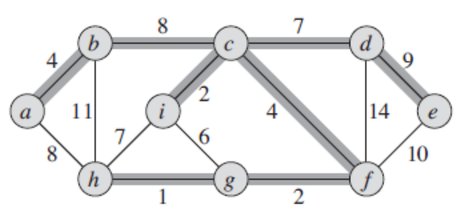
הגדרות ומונחים.

במהלך הנושא נעסוק בגרף לא מכוון $G=(V,E)$ שהינו קשיר, עם משקל לצלעות.

אנחנו מחפשים תת קבוצה של הצלעות, $T\subseteq E$ כך שסכום המשקלים יהיה מינימלי, והצלעות מהוות עץ - ללא מעגלים.

במצב כזה, נקבל גרף חדש $G'=(V,T)$.

למשל, נוכל להתבונן בדוגמא:



אם נקבל עץ עם סכום משקלים מינימלי, נקבל **עץ פורש מינימלי**.

מוטיביציה לעץ כזה יכולה להיות מרחק מינימלי בין כל אחת מערים במפה.

ניזכר בהגדרה של גרף ממשוקל.

במצב כזה, אנחנו מסמנים, לכל v_i,v_j משקל $w\left(v_i,v_j\right)>0$.

כמו כן, ניתן להפוך גרף שאינו ממשוקל לממושקל על ידי הוספת משקל בגודל 1 לכל אחת מהקשתות.

במידה ואין צלע בין שני קודקודים, נוכל לומר כי המשקל ביניהם הינו ∞.

הגדרנו גם משקל של מסלול, על ידי סכום המשקלים, כלומר:

$$w(path(u,v)) = \sum_{i=0}^{k} m\left(v_{i-1}, v_i\right) \qquad u = v_0, v_k = v$$

אם כן, עבור עץ T סכום המשקלים הינו: $w(T) = \sum_{i=1}^{|T|} m\left(e_i\right)$ לכל $e_i \in T$.

הגדרה

בהינתן גרף ממשוקל, נגדיר את ה**עץ הפורש** להיות תת קבוצה $T \subseteq E$ כך ש- $G' = (V,T)$ היא קשיר וללא מעגלים.

עץ פורש מינימלי הינו **עץ פורש** שמשיג מינימום של סכום המשקלים.

כלומר $w(T) = \sum_{(u,v)\in T} w(u,v)$ הינו המינימלי.

כיצד ניתן למצוא עץ פורש מינימלי? ראינו כי DFS משיג עץ פורש, ולכאורה נוכל למצוא את כל העצים הפורשים האפשריים ולמצוא את המינימלי מביניהם.

אך דבר זה איננו יעיל.

האלגוריתמים הכלליים.

לפני שנמצא את הפיתרון, נבצע "תרגיל חימום", עבור בעייה פשוטה יותר.

נניח כי קיימת $A = \{a_1, \ldots, a_n\}$ - קבוצה של n מספרים שלמים גדולים ביותר, נרצה למצוא תת קבוצה $S \subseteq A$ של k איברים, כך שסכום האיברים הינו מינימלי.

למשל, עבור הקבוצה $A = \{2, 25, 6, 10, 3, 11, 7, 32\}$. נקבל כי $n = 8$ ו-$k = 4$, ולכן $S = \{2, 3, 6, 7\}$ והסכום המינימלי הינו 18.

כלומר, קיבלנו כי המינימום הוא הסכום של k המספרים הקטנים ביותר.

דבר זה ידרוש קודם כל מיון של A, שייקח לנו $O\left(n \log n\right)$ זמן, ואז בחירת k המספרים הראשונים.

הגדרה

אלגוריתם חמדן הינו אלגוריתם הבוחר את האפשרות הטובה ביותר הנראית לעין, מבלי לקחת בחשבון את ההשפעה של צעד זה על המשך הפיתרון.

פיתרון חמדני הינו למצוא את המספר המינימלי, להוציא אותו ולחזור על פעולה זו k פעמים. על מנת לעשות דבר זה נצטרך לבנות ערימה, ולהוציא בכל פעם את המינימום (k פעמים). אם כך, הסיבוכיות תהיה $O\left(n\right) + k \cdot O\left(\log n\right) = O\left(n + k \log n\right)$. כלומר, כאשר $k < n$, הסיבוכיות בדרך זו תהיה טובה יותר.

במקרה שלנו, עלינו למצוא בכל פעם צלע שתעמוד בתנאים הדרושים לנו (משקל מינימלי, וללא מעגלים).

נגדיר **צלע בטוחה**, להיות צלע בעלת משקל מינימלי וגם ללא מעגלים.

נכתוב את האלגוריתם בפסאודו קוד גנרי:

אלגוריתם 24 $Generic-Mst\left(G\right)$

(1) $T = \emptyset$
(2) כל עוד T איננו עץ פורש של G
 (א) תמצוא צלע בטוחה $e = (v,u) \in E$
 (ב) $T = T \cup \{e\}$
(3) תחזיר את T

בשמורת הלולאה נרצה להוכיח כי בכל חלק שאנחנו מוסיפים אנחנו בהכרח מוסיפים צלע של עץ הפורש המינימלי הסופי.

ברור כי כאשר הגרף ריק הטענה נכונה, ובהמשך נוכיח כי זה נכון גם עבור כל הוספת צלע.

לפי השיטה הראשונה, נמצא בכל פעם את המינימום בגרף באמצעות ערימה, ועל מנת לוודא שאין מעגלים, נבדוק האם שכניו "סוגרים מעגלים".

לפי שיטה זו, ייתכן ומדובר ביער. כלומר, אנחנו לא בהכרח בוחרים צלעות שמקושרות לצלע השנייה.

נוכל לעשות זאת גם בשיטה אחרת, באמצעות מציאת הקשת עם המשקל הנמוך ביותר המקושרת אל הגרף הקיים, ולבחון את כל השכנים הקשורים לקשת זו (שיטה הדומה לBFS) - בשיטה זו יש בהכרח עץ לכל אורך הדרך.

נוכל להסביר את ההבדל בין שתי השיטות בצורה הבאה.

ניקח את הגרף המקורי ולאחר כל שלב נסמן את הקודקודים והצלעות שמצאנו "ונשים אותם" בצד שמאל - בקבוצה U, ואת כל הקודקודים שלא גילינו נסמן בתור $V \setminus U$ ו"נשים בצד ימין". הצלעות שנשארו ללא קבוצה הן הצלעות המחברות בין U ובין $V \setminus U$.

בשלב הבא, יש לנו שתי אפשרויות:

א. לבחור את הקשת המינימלית ב-$V \setminus U$ (וגם בקשת המחברת בין U ובין $V \setminus U$) - האפשרות הראשונה שהצענו קודם לכן.

ב.. לבחור את הקשת המינימלית המחברת בין U ובין $V \setminus U$ - האפשרות השנייה שהצענו, שמכריחה מצב של קשירות.

כאשר $U = V$ מובטח לנו שיש עץ פורש מינימלי.

הגדרות

חתך $C = (U, V \setminus U)$ של גרף שאינו מכוון $G = (V, E)$ הוא **חלוקה** של צלעות V.

נאמר כי קשת $e = (u, v)$ חוצה את החתך C אם ורק אם $u \in U$ ו-$v \in V \setminus U$.

קשת תיקרא **קשת קלה** אם ורק אם משקלה הוא המינימלי מבין הקשתות החוצות את החתך C.

משפט

תהי $U \subseteq V$ ותהי $e\left(u, v\right)$ קשת קלה החוצה את C (כלומר $u \in U$ ו-$v \in U \setminus v$), אז קיים עץ פורש מינימלי T, כאשר $e \in T$.

הוכחה

נניח כי יש עץ לנו עץ פורש T. אם $e \notin T$ ונרצה להוסיף אותו ל-T יווצר מעגל (כיוון שמדובר בעץ), ולכן בהכרח קיים $e' = (u', v')$ כאשר $u \in U$ ו-$v \in U \setminus V$.

אמנם, נבחין כי $w\left(e\right) \leq w\left(e'\right)$ כיוון שמדובר בקשת קלה.

לכן, נוכל למחוק את e' ולהוסיף את e. במצב כזה, מתקיים כי $T' = \{T \setminus \{e'\}\} \cup \{e\}$, כך ש-$w\left(T'\right) \leq w\left(T\right)$, ולכן זהו עץ פורש מינימלי.

משפט

יהי $G = (V, E)$ גרף בלתי מכוון קשיר עם פונקציית משקל w המוגדרת על E ומקבלת ערכים ממשיים. תהי A תת קבוצה של E המוכלת בעץ פורש מינימלי כלשהוא של G.

יהי $C = (U, V \setminus U)$ חתך כלשהוא של G המכבד את A (כלומר, אין צלע של A שחוצה את C) ותהי $e = (u, v)$ קשת קלה החוצה את C, אזי הקשת e הינה בטוחה עבור A.

הוכחה

יהי T עץ פורש המינימלי המכיל את A ואיננו מכיל את e. כלומר $e \notin T$.(אם הוא מכיל אותה, סיימנו).
נבנה כעת עץ אחר, על ידי הוספת e, ולכן בהכרח יצרנו מעגל (כי מדובר בעץ). לכן, נוכיח להסיר צלע אחת e', כשעדיין הגרף יישאר קשיר.
נראה כי T הוא עץ פורש מינימלי, שהרי e היא קשת קלה החוצה את C וגם e' חוצה את C, אבל $w\left(e\right) \leq w\left(e'\right)$.

הצלע e היא קשת בטוחה עבור A, שכן מכך ש- $A \subseteq T$ ו-$e \notin A$ ו-$A \cup \{e\} \subset T'$ עולה כי $A \subseteq T'$. ולכן בפרט T' הוא עץ פורש מינימלי, וגם e בטוחה עבור A.

כעת עלינו להוכיח שהוספת הצלעות הינה בטוחה, בצורה יעילה.
ישנם שני דרכים לעשות זאת, המתאימות לדרכים שתיארנו קודם לכן:
1. האלגוריתם של קרוסקל.
2. האלגוריתם של פרים.

האלגוריתם של קרוסקל.

נציג כעת תיאור של האלגוריתם של קרסקל, המתאים לשיטה הראשונה שתיארנו קודם לכן:

אלגוריתם 25 האלגוריתם של קרוסקל

---

(1) $A=\emptyset$

(2) לכל $v\in V$ תיצור קבוצה v.

(3) תמיין את הצלעות לפי משקלים עולים.

(4) לכל צלע $e=(u,v)\in E$:

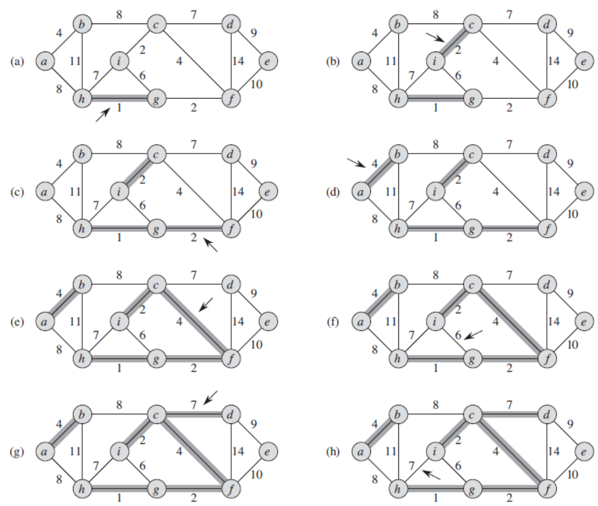
 (א) אם u לא נמצא בקבוצה של v (שייכים לאותו עץ):

  (i) $A\to A\cup\{e\}$

  (ii) תאחד את $\{u,v\}$

(5) תחזיר את A

נוכל לראות דוגמה לכך כאן:



הנכונות של האלגוריתם נובעת מהמשפט **הקודם** שהוכחנו קודם לכן.

הסיבוכיות הינה כזאת: ראשית, נצטרך למיין ודבר זה ייקח $|E|\log|E|$. לאחר מכן, נעבור כל הקודקודים, ונצטרך לבדוק עבור כל הצלעות, ולכן דבר זה ייקח $|E|\cdot|V|$ במימוש נאיבי שמבצע איחוד של קבוצות.

אמנם, באמצעות אלגוריתם שנראה בהמשך, נוכל לבצע איחוד של קבוצות ב-$O(1)$ ולכן זמן הריצה יירד ל-$O(|V|)$.

אם כך, זמן הריצה הינו $O(|E|\log|E|)$.

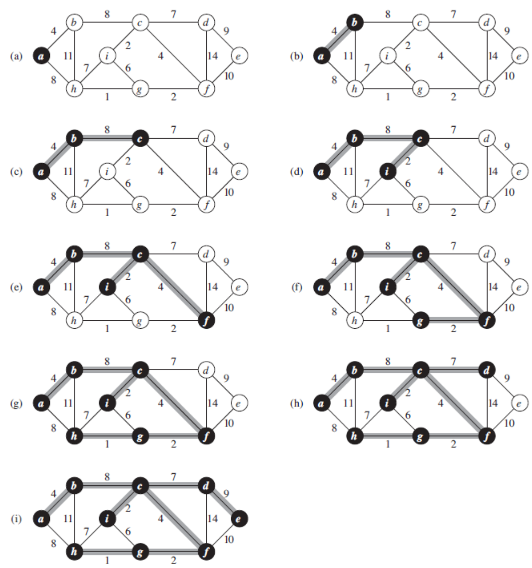
האלגוריתם של פרים.

כעת, נתבונן באלגוריתם של פרים:

אלגוריתם 26 האלגוריתם של פרים

(1) לכל $v \in V$
(א) $v.key \to \infty$
(ב) $v.\pi \to null$
(ג) $root.key \to 0$
(ד) $Q \to V$
(2) כל עוד התור Q איננו ריק:
(א) $u \to extract - min\left(Q\right)$
(ב) לכל v אם הוא שכן של u:
(i) אם $v \in Q$ ו-$w\left(u,v\right) < v.key$
(א') $v.\pi \to u$
(ב') $v.key \to w\left(u,v\right)$

נוכל לראות דוגמה לכך כאן:



הנכונות נובעת מהמשפט הראשון שהוכחנו קודם לכן.

הסיבוכיות תלויה במימוש של הערימה.

בניית הערימה אורכת $O\left(|V|\right)$, ובנוסף **הוצאת המינימלי** לוקחת $O\left(\log V\right)$ לכל קודקוד, ולכן $O\left(|V|\log|V|\right)$.

לולאת ה-for שאנחנו משתמשים בה מתבצעת $O\left(E\right)$ פעמים שכן סכום אורכי כל רשימות הסמיכות הוא $2|E|$.

בדיקת השייכות לתור ניתנת לביצוע בזמן קבוע, השמת הערך בשורה האחרונה שקולה ל-$decrease - key$ של ערימה, שלוקחת $O\left(\log|V|\right)$.

ולכן, סך הכל הזמנים הינם $O\left(|V|\log|V| + |E|\log|V|\right) = O\left(|E|\log|V|\right)$.

שיעור מס' 11: מסלולים קצרים ביותר בגרף

יום רביעי
30.12.20

הקדמות ומונחים.

ישנם שלושה סוגים של מציאת מסלולים קצרים ביותר:

(1) בהינתן שני קודקודים, כל המסלולים הקצרים ביניהם.
(2) בהינתן קודקוד מסוים, כל המסלולים ממנו לשאר הקודקודים.
(3) כל המסלולים הקצרים בין כל זוגות של קודקודים.

פתרנו את 1 ו-2 באמצעות BFS, עבור גרפים שאינם מכוונים ואיננם ממושקלים.
כעת נדבר על בעיות אלו בגרפים מכוונים וממושקלים.

נוכל למצוא מוטיבציה לכך, כשנרצה למצוא מסלול בין שתי נקודות, בהינתן קריטריונים נוספים.

הגדרות

גרף ממושקל הוא גרף לו יש משקל $w\left(v_iv_j\right)$- משקל יכול להיות גם שלילי.
אם אין מסלול בין שני קודקודים, המשקל ביניהם הוא ∞.
המשקל של **מסלול** הוא סכום המשקלים של **המסלולים**, כלומר $w(p(u,v))=\sum_{i=0}^{k} m\left(v_{i-1},v_i\right)$.
המשקל של המסלול הקצר ביותר בין שני קודקודים בהכרח קיים. נסמנו ב-$\delta\left(u,v\right)$ כאשר u ו-v הם שני קודקודים.
מתקיים:

$$\delta(u,v)=\begin{cases}\min(w(p):u\xrightarrow{p}v) & exist\ path\\ \infty & else\end{cases}$$

נוכל גם להתבונן במספר סימונים וטרמינלוגיות.

סימונים וטרמינלוגיות

בהינתן גרף $G=(V,E)$, כך ש-$V=\{v_1\dots,v_k\}$ ו-$E=\{(e_{ij},w\left(e_{ij}\right))\}$, אזי $p_{1k}=\langle v_1,\dots,v_k\rangle$ יסומן בתור המסלול הקצר ביותר בין v_1 ו-v_k.
כמו כן, נסמן ב-p_{ij} את תת המסלול של p_{1k} כאשר $1\le i,j\le k$.
כש- $v_i=v_j$ בתת מסלול כלשהוא, נאמר כי יש מעגל בגרף.
משקל שלילי בגרף הינו משקל בו "מחסירים" מהמשקל, והוא מוגדר היטב גם במקרה זה.
גרף שבו יש מעגל שלילי, משקלו הינו $-\infty$ ולכן נרצה לפסול אותם.
בהינתן מעגל עם מסלול 0, נרצה לפסול אותו גם.

תכונה 1
המסלול הקצר ביותר בין שני קודקודים, הינו חסר מעגלים חיוביים.
הוכחה
אם נניח בשלילה שיש מעגל חיובי בין שני קודקודים , אזי ברור שלא מדובר במסלול הקצר ביותר.
ולכן נוכל למחוק את המעגל החיובי.

תכונה 2
תתי מסלולים של מסלולים קצרים ביותר, גם הם קצרים ביותר.
הוכחה
נניח בשלילה שלא. נניח כי p_{ij} הינו תת מסלול של מסלול קצר ביותר p_{1k}, והוא לא המסלול הקצר ביותר בין v_i ו-v_j.
אזי קיים q_{ij} שהינו קצר ביותר ונוכל להחליף את p_{ij} , בסתירה לכך ש-p_{1k} הוא המסלול הקצר ביותר.

מסלולים קצרים ביותר מ-s לשאר הקודקודים, הינם עץ רוחב - BFS.
כלומר, מדובר ב-$G_\pi = (V_\pi, E_\pi)$, כך ש-$V_\pi = \{v \in V : v.\pi \neq null\} \cup \{s\}$
וגם $E_\pi = \{(v.\pi, v) : v \in V - \{s\}\}$.

ניזכר כי אם ניקח גרף לא ממושקל שאינו מכוון, נוכל להסתכל על כך בתור עץ חיפוש. דהיינו, נוכל להסתכל על הגרף כבעל רמות שונות, אשר תלויות בשכני הקודקודים.
אנו יודעים כי במצב זה, הפעם הראשונה שבה אנחנו מגלים קודקוד, הינה למעשה המסלול הקצר ביותר.
כעת, אם נשים משקלים, לא נוכל להתמודד בצורה דומה, כי המשקל לא אחיד לפי הרמות.
גם לא נוכל להשתמש ברעיון של $current$ ו-$not\_visited$, כי נוכל לחזור אחורה.
חשוב גם לזכור שבחישוב עץ רוחב הרגיל השתמשנו בשדות של "קודם" ו"מרחקים", וכעת כפי שראינו ייתכן שנצטרך לעדכן זאת "אחורה".
במהלך מציאת עצי רוחב, התחלנו במרחק של ∞ בין הקודקוד הרצוי לשאר הקודקודים, ושינינו אותו בהתאמה.
נשתמש ברעיון דומה גם במשקלים.

כיצד נוכל לעשות זאת בגרף ממושקל?
אנחנו כבר מבינים ש"השכן" איננו בהכרח המסלול הקצר ביותר (שכן ייתכן מסלול 'ארוך' יותר עם משקל קטן יותר).
הפעולה הבסיסית בה נשתמש בה היא "הקלה", ומדובר גם פה באלגוריתם חמדני, שכן העדכון מתבצעת בצורה חמדנית.

הגדרה
הקלה (relaxation) היא שיטה בה מקטינים בכל פעם חסם עליון של משקלו של כל קודקוד, עד שמגיעים למסלול הקצר ביותר.

פסאודו קוד פשוט **להקלה**:

אלגוריתם 27 הקלה

(1) אם $v.dist > u.dist + w(u,v)$
 (א) $v,dist \to u.dist + w(u,v)$
 (ב) $v.\pi \to u$

ההקלה מקיימת מספר תכונות:
(1) **תכונת החסם העליון** - לכל $v \in V$ מתקיים כי $v.dist \geq \delta(s,v)$. כמו כן $v.dist = \delta(s,v)$ לעולם איננו משתנה.
(2) **תכונת "אין דרך - מסלול אינסופי"** - אם אין מסלול בין s ל-v אזי $v.dist = \delta(s,v) = \infty$.
(3) **מונטוניות יורדת -** בסוף התהליך, $v.dist = \delta(s,v)$.

(4) **אי שוויון המשולש** - לכל $e=(u,v)\in E$ מתקיים כי $\delta(s,v)\leq\delta(s,u)+w(u,v)$
(5) **תכונת ההתכנסות** - אם יש מסלול בין s ו-v, כאשר $u.dist=\delta\left(s,u\right)$, בשלב המקדים להקלת הצלע (u,v) אזי $v.dist=\delta\left(s,v\right)$, לאחר מכן.
(6) **תכונת הקלת המסלול -** נניח כי $p_{1k}=\langle v_1,\ldots,v_k\rangle$ הוא מסלול קצר ביותר בין v_1 ו-v_j כאשר הצלעות הינן $\left(v_l,v_2\right),\left(v_2,v_3\right),\ldots\left(v_{k-1},v_k\right)$. אזי $v_k.dist=\delta\left(s,v_k\right)$.
(7) **תכונת הקודם של תת גרף -** אם $v.dist=\delta\left(s,v\right)$ אזי לכל $v\in V$ מתקיים כי עץ הקודמים, הוא עץ רוחב של מסלולים קצרים ביותר, ששורשו ב-s.

אלגוריתמים.

במהלך השיעור, נראה שני אלגוריתם.
האלגוריתם של דייקסטרה, שהוא הכללה של BFS. סיבוכיות הזמן של אלגוריתם זה הינה $O\left(|E|\log|V|\right)$ וסיבוכיות המקום הינה $O\left(|V|+|E|\right)$.
אלגוריתם זה מניח כי פונקציית המשקל הינה אי שלילית.

בנוסף נראה את האלגוריתם של בלמן-פורד, שמבצע במהלך $|V|$ איטרציות הקלות על כל הצלעות. לכן, זמן הריצה שלו הוא $O\left(|E|\,|V|\right)$, וסיבוכיות המקום הינה $O\left(|V|+|E|\right)$.
היתרון של אלגוריתם זה הינו שהוא מסוגל לזהות מעגלים שליליים בגרף.

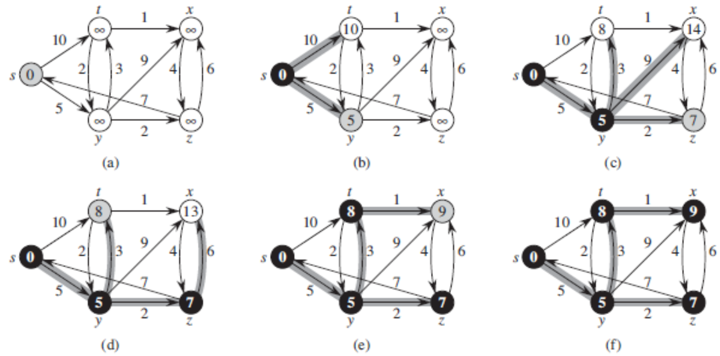
שני אלגוריתמים האלו הינם חמדניים, ומשפרים בכל פעם את המרחק.
נתחיל באלגוריתם של דייקסטרה. כשנוסיף את הצלע, נוסיף גם את **המשקל**. בנוסף, נרצה להתקדם לפי השכן בעל **המשקל המינימלי**.
כלומר, כאשר נגיע לקודקוד, נתבונן בכל השכנים שלו, נמצא את הקודקוד בעל **המשקל המינימלי ונבצע עליו הקלה**.
נתבונן באלגוריתם:

אלגוריתם 28 $Dijkstra\left(G,s\right)$

---

(1) $s.dist\rightarrow0$
(2) $s.\pi\rightarrow null$
(3) לכל $u\in V\setminus\{s\}$
   (א) $u.dist\rightarrow\infty$
   (ב) $u.\pi=null$
(4) $S=\emptyset$
(5) $Q\rightarrow V$
(6) כל עוד $Q\neq\emptyset$:
(7) $u\rightarrow Extract-Min\left(Q\right)$
(8) $S\rightarrow S\cup\{u\}$
(9) לכל שכן v של u:
   (א) תבצע $Relax\left(u,v,w\right)$

ניתן לראות דוגמה לכך בתמונה הבאה:



משפט (נכונתו של האלגוריתם של דייקסטרה)

אם מריצים את האלגוריתם של דייקסטרה על גרף מכוון ומשוקלל, $G=(V,E)$ עם פונקציית משקל אי שלילית w אזי $u.dist=\delta(s,u)$ לכל $u\in V$.

הוכחה

נוכיח את שמורת הלולאה.

אכן, בתחילת האיטרציה הראשונה, מתקיים כי $u.dist=\delta(s,u)$ לכל $u\in S$.

כעת, עלינו להראות כי לכל $u\in V$ מתקיים כי $u.dist=\delta(s,u)$ כאשר הוספנו את u ל-S.

בתחילה, מתקיים כי S הינו ריק, ולכן בפרט הטענה נכונה.

באיטרציה האחרונה, מתקיים כי $S=V$ והתור Q הינו ריק. לכן עלינו להתבונן רק באמצע התהליך.

נשתמש בהנחה בשלילה.

נבחר את הקודקוד הראשון בו תכונה זו איננה מתקיים, ונסמנו ב-u. כלומר, פורמלית, זהו הקודקוד הראשון בו $u.dist\neq\delta(s,u)$.

בפרט מתקיים כי $s\neq u$ שהרי $s.dist=\delta(s,s)=0$.

מאחר ש-$s\neq u$ מתקיים כי $S\neq\emptyset$ גם לפני הכנסתו של u.

חייבת להיות דרך מ-s ל-u, שכן אחרת $u.dist=\delta(s,u)=\infty$.

נפצל את המסלול לשלושה מסלולים, כלומר למסלול (s,x) ו-(x,y) וגם (y,u), כאשר $x\in S$.

בפרט, לפי ההנחה $x.dist=\delta(s,x)$.

כיוון שביצענו הקלה לצלע (x,y), לכן בפרט יתקיים כי $y.dist=\delta(s,y)$.

כל שנותר לנו הוא להראות כי $u.dist=\delta(s,u)$.

כיוון ש-y מופיע לפני u וכיוון שהמשקלות אי שליליים, אזי מתקיים בפרט כי $\delta(s,y)\leq\delta(s,u)$.

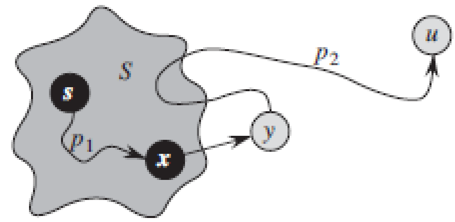
כעת, נוכל לשים לב כי מתקיים:

$$y.dist=\delta(s,y)\leq\delta(s,u)\leq u.dist$$

כיוון ש-$u,y\in V\setminus S$ כאשר u נבחר, מתקיים בפרט מפעולת האלגוריתם, כי $u.dist\leq y.dist$ ולכן $\delta(s,u)=u.dist$ (כיוון שיש שני אי שיוויונית חלשים הפוכים, יש בפרט שוויון).

קיבלנו כי $\delta(s,u)=u.dist$, בסתירה להנחה.

תמונה להמחשת ההוכחה:



זמני ריצה

מבחינת זמני ריצה, כאמור האלגוריתם מבצע $|V|$ פעולות של הכנסה לתור והוצאת מינימום, ו-$|E|$ פעולות של $Decrease-Key$ ולכן זמן הריצה היינו:

$$O(|V|\log|V|)+O(|E|\log|V|)=O(|E|\log|V|)$$

כאשר $|E|=O\left(|V|^2\right)$ זה לא אופטימילי, שהרי מתבצעות יותר פעולות של $Decrease-Key$, מאשר $Extract-Min$.

הפיתרון הוא לממש בתור מערך, ואז כל פעולת $Extract-Min$ לוקחת $O\left(|V|\right)$ ומתבצעות $|V|$ פעולות כאלו.

מאידך, כל פעולה של הכנסה או $Extract-Min$ לוקחת $O\left(1\right)$.

לכן במקרה זה זמן הריצה הינו בסך הכל $O\left(|V|^2\right)+O\left(|E|\right)=O\left(|V|^2\right)$.

האלגוריתם של בלמן-פורד

נתבונן רק ברעיון.

נרצה לבדוק מהי הדרך הקצרה ביותר מכל קודקוד לקודקוד אחר. מספר הצלעות המקסימלי במסלול ללא מעגלים הוא $|V|-1$.

לכן, מספיק שנבדוק $|V|-1$ צלעות. אם לאחר $|V|$ צלעות המשקל ירד, סימן שיש מעגל שלילי בגרף (הגרף לא חסום מלרע, נוכל תמיד להסתובב בו ולרדת יותר ויותר).

נתבונן באלגוריתם:

אלגוריתם 29 $Bellman-Ford\left(G,w,s\right)$

(1) $initialize-single-source\left(G,s\right)$

(2) לכל $i=1$ עד $|V|-1$

  (א) תבצע $Relax\left(x,v,w\right)$

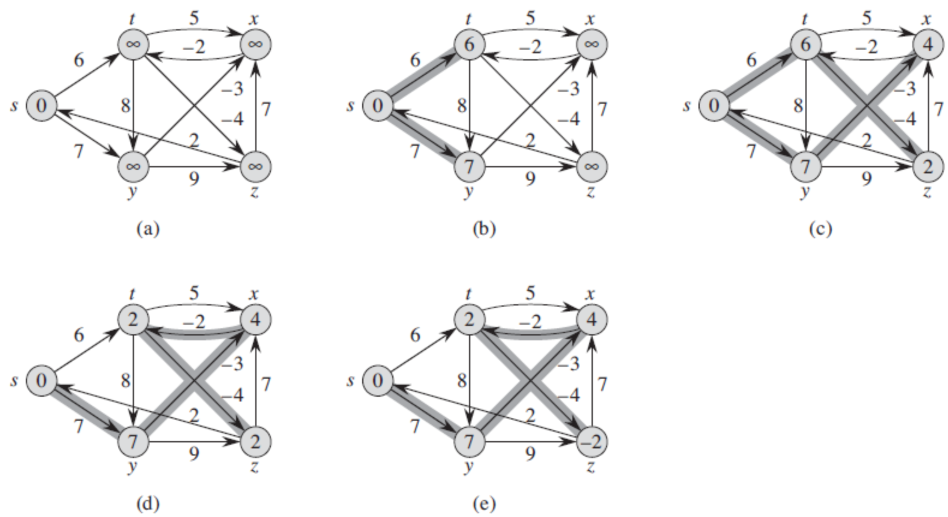
(3) לכל צלע $\left(u,v\right)$

  (א) אם $v.dist>u.dist+w\left(u,v\right)$

    (i) תחזיר "יש מעגל שלילי"

(4) תחזיר $true$

נוכל לראות דוגמה לכך כאן:



שיעור מס' 12:

יום רביעי

06.01.21

מסלולים קצרים בין כל הזוגות

מוטיבציה

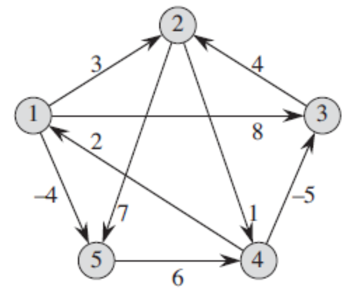
בהינתן גרף, נרצה לחשב מראש את **כל המסלולים הקצרים** בין **כל הקודקודים**.
לעתים, גם אם לא נרצה לחשב את כל המסלולים הקצרים, נרצה לחשב כמה שיותר מהם.

הפיתרון הפשוט הוא לכאורה להריץ את האלגוריתם למציאת המסלול הקצר בין קודקוד ספציפי לשאר הקודקודים, על כל הקודוקדים.
אם ניקח למשל את **בלמן פורד, נקבל כי** $O\left(|E|\cdot|V|\right)$ וכיוון ש-$|V|=O\left(V^2\right)$ נקבל $O\left(|V|^4\right)$.
אם ניקח את **דייקסטרה**, נקבל כי $O(|E|\cdot\log|V|\cdot|V|)=O\left(|V|^3\log|V|\right)$

אך האם אנחנו יכולים להשיג זמן מהיר יותר? אינטואיטיבית, נראה שלא. אמנם, נוכל לחשוב על תתי המסלולים, שהם אופטימליים, ולהשתמש בהם שוב ושוב ללא חישוב מחדש.
ואכן, נראה כי אפשר למצוא את כל המסלולים הקצרים בזמן של $O\left(|V|^3\right)$.

הרעיון הבסיסי

בשונה מהאלגוריתמים שראינו עד כה, שרובם משתמשים ברשימות סמיכות, באלגוריתמים כאן נשתמש בייצוג על ידי **מטריצת סמיכויות**.
ניקח מטריצות בגודל $|V|\times|V|$.
ניקח את הדוגמה הבאה:



מטריצה אחת שהינה **מטריצת המשקלים** המיוצגת על ידי:

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ w(i,j) & i \neq j \text{ and } (i,j) \in E \\ \infty & i \neq j \text{ and } (i,j) \notin E \end{cases}$$

במקרה זה, נבחין כי הכיוונית חשובה. בדוגמה שראינו לעיל, למשל, המטריצה הינה:

$$\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

מטריצה נוספת היא **מטריצת המרחקים הקצרים** ביותר:

$$\begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

(יש רק 0-ים על האלכסון כי מרחק מכל קודקוד לעצמו הוא 0).
מטריצה זו תסומן ב- L, כאשר תא (i,j) מציג את המרחק הקצר ביותר i ו-j.

המטריצה השלישית היא **מטריצת הקודמים** והיא מייצרת לנו את $G_{\pi,i} = (V_{\pi,i}, E_{\pi,i})$. נסמנה ב-$\prod$.
מטריצה זאת נראית כך:

$$\begin{pmatrix} null & 3 & 4 & 5 & 1 \\ 4 & null & 4 & 2 & 1 \\ 4 & 3 & null & 2 & 1 \\ 4 & 3 & 4 & null & 1 \\ 4 & 3 & 4 & 5 & null \end{pmatrix}$$

(האלכסונים מלאים ב-$null$, כיוון שהקודם של "מסלול קצר לעצמו", הוא $null$).

מבחינת אלגוריתם:

אלגוריתם 30 הדפסת כל הזוגות הקצרים

(1) אם $i=j$
  (א) תדפיס את i
(2) אם $\pi_{ij}=null$:
  (א) תדפיס שאין דרך
(3) תפעיל את האלגוריתם רקורסיבית על (π,i,π_{ij})
(4) תדפיס את j

אם כך, כל שנותר לנו הוא לבנות את מבנה הנתונים הללו בצורה יעילה.

ניזכר כי אמרנו **שתתי מסלולים** של מסלול קצר ביותר גם הם **קצרים ביותר**.
כמו כן, ניזכר כי אם המסלול הקצר ביותר בין v_j ל-v_j אורך במקסימום m צלעות, אזי המסלול הקצר ביותר הוא אחד משתי האפשרויות הבאות:
(א) במקסימום עם $m-1$ צלעות.
(ב) עם $m-1$ צלעות ועוד צלע נוספת (כלומר מפצלים למסלול עד קודקוד v_k כלשהוא + הצלע שמחברת בין v_k ל-v_j).
בנוסף, במצב כזה נקבל כי:

$$l_{ij}^{(m)}=\min\left(l_{ij}^{(m-1)},\min_{1\le k\le n}\left\{l_{ik}^{(m-1)}+w_{kj}\right\}\right)$$

כלומר, מדובר בסוג של א"ש משולש, שמוודא מהו המשקל המינימלי.

הפיתרון שנראה כעת בנוי על שתי ההערות שהזכרנו כעת.
זהו פיתרון שמבוסס על **תכנות דינמי**. כלומר, מתבצעים השלבים הבאים:
(א) אפיון המבנה האופטימלי
(ב) הגדרה רקורסיבית של ערכו של פיתרון אופטימלי
(ג) חישוב ערכו של הפיתרון האופטימלי (חישוב המינימום אצלנו).
(ד) בניית פתרון אופטימלי מתוך המידע שחושב (המסלולים הקצרים ביותר).

כעת, ניגש למטריצה L שראינו קודם לכן. יהי $l_{i,j}^{(m)}$ המשקל המינימלי בין כל קודקודי i ו-j, עם מקסימום $m=0$.
כעת, אם $m=0$ נקבל כי $l_{ij}^{(0)}=\begin{cases}0 & i=j\\ \infty & i\neq j\end{cases}$.
עבור $m\ge 1$ נגדיר את $l_{i,j}^{(m-1)}$ להיות הדרך הקצרה ביותר, כך שמתקיים:

$$l_{ij}^{(m)}=\min\left(l_{ij}^{(m-1)},\min_{1\le k\le n}\left(l_{ik}^{(m-1)}+w_{kj}\right)\right)=\min_{1\le k\le n}\left(l^{(m-1)}+w_{kj}\right)$$

כאשר למעשה אי השוויון האחרון נובע מהעובדה כי כאשר $k=j$ אזי נקבל 0.

בדוגמה שלנו, בתחילה נגדיר את המטריצה כך:

$$L^{(0)} = \begin{pmatrix} 0 & \infty & \infty & \cdots & \infty \\ \infty & 0 & \infty & \cdots & \infty \\ \infty & \infty & 0 & \cdots & \infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \infty & \infty & \infty & \cdots & 0 \end{pmatrix}$$

לאחר מכן, עבור $L^{(1)}$ נקבל למעשה את הגרף G עצמו:

$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$

למעשה, מדובר בסוג שונה של א"ש המשולש. מדובר בתופעה דומה להשוואה שעשינו כשדיברנו על "הקלות".

כעת, ניגש לאלגוריתם.
לפי מה שראינו עד כה, עלינו למצוא סדרה של מטריצות $L^{(1)}, L^{(2)}, \ldots, L^{(n-1)}$.
נזכיר כי $L^{(1)} = G$, כלומר הגרף המקורי.

אלגוריתם 31 $Extend - Shortest - Paths\,(L)$

(1) $n \to L.rows$
(2) נגדיר מטריצה $L' = \left(l'_{ij}\right)$ מטריצה $n \times n$.
(3) לכל $i \to 1$ עד n:
  (א) לכל $j \to 1$ עד n:
    (i) $l'_{ij} \to \infty$.
    (ii) לכל $k \to 1$ עד n:
      (א') $l'_{ij} \to \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$.
(4) תחזיר את L'

לא קשה לראות כי זמן הריצה הוא $\Theta\left(|V|^3\right)$.
נבחין כי הפעולה הזאת דומה מאוד ל**כפל מטריצות**.
הפעולה הבסיסית של מכפלת מטריצות היא כפל של השורה ה-i בעמודה ה-j כשלמעשה כופלים כל איבר מתאים.
מבחינה אלגוריתמית:

אלגוריתם 32 כפל מטריצות

(1) $n \to A.rows$
(2) ניקח C מטריצה בגודל $n \times n$
(3) לכל $i \to 1$ עד n:
  (א) לכל $j \to 1$ עד n :
  (ב) $c_{ij} \to 0$
    (i) לכל $k \to 1$ עד n;:
      (א') $c_{ik} \to c_{ij} + a_{ik} \cdot b_{kj}$
(4) תחזיר את C

גם כאן הסיבוכיות היא n^3.
מדובר בדבר דומה לכפל מטריצות, כיוון שבמצב שלנו נחליף את $c_{ij} \to 0$ ב-$l'_{ij} \to \infty$.
כמו כן, נחליף את $c_{ik} \to c_{ij} + a_{ik} \cdot b_{kj}$ ב-$l'_{ij} \to \min\left(l'_{ij}, l_{ik} + w_{kj}\right)$.

אם כך, מהו האלגוריתם למציאת כל המסלולים הקצרים ביותר?

אלגוריתם 33 כל המסלולים הקצרים ביותר בזמן נורא ארוך (W)

---

(1) $n \to W.rows$
(2) $L^{(1)} \to W$
(3) לכל $m \to 2$ עד $n-1$:
(א) $L^{(m)} \to$ Extend-Shortest-Paths
(4) $return\ \ L^{(m)}$

קיבלנו $|V|^4 = |V| \cdot \left|V^3\right|$. זמן לא יעיל יותר מדי. [12]

ננסה אם כן להיעזר בדרך זו על מנת להגיע לזמן ריצה יעיל יותר.
על מנת לעשות זאת, נתבונן בהעלאות רגילות בחזקה.

הדגמה קטנה על חזקות בצורה רקורסיבית
יהי $a \in \mathbb{R}$. נניח כי a^n כאשר n הינו חזקה של 2.
נקבל כי $a^n = a^{\frac{n}{2}} \cdot a^{\frac{n}{2}}$ וגם $a^{\frac{n}{2}} = a^{\frac{n}{4}} \cdot a^{\frac{n}{4}}$. אם נבצע זאת רקורסיבית, נוכל לבצע סך הכל $\log_2 n$ פעולות.
כאשר n איננו חזקה של 2, אזי בעץ הרקורסיה יש בסך הכל $n \geq 2^{\lceil \log_2(n) \rceil}$ עלים, ולכן מספר המכפלות הינו בסך הכל $\lceil \log_2(n) \rceil$.
דבר זה נקרא "העלאה בריבוע באמצעות רקורסיה".

במקרה שלנו, איננו צריכים לחשב את $D^{(n-1)}$ בלבד כאשר $m \geq n-1$. ניתן לחשב את $D^{(n-1)}$ בצורה דומה למה שראינו בכפל מטריצות.
כלומר, במקרה שלנו:

$$
\begin{aligned}
D^{(1)} &= W \\
D^{(2)} &= W^2 && = W \cdot W \\
D^{(4)} &= W^4 && = W^2 \cdot W^2 \\
D^{(8)} &= W^8 && = W^4 \cdot W^4
\end{aligned}
$$

לכן, נקבל בסך הכל:

$$
L^{\left(2^{\lceil \log(n-1) \rceil}\right)} = W^{\left(2^{\lceil \log_2(n-1) \rceil}\right)} = W^{\left(2^{\lceil \log_2(n-1) \rceil - 1}\right)} \cdot W^{\left(2^{\lceil \log_2(n-1) \rceil - 1}\right)}
$$

אם כך, נוכל להוריד את מספר האיטרציות ל-$O\left(\log|V|\right)$.
אלגוריתם חדש:

---

[12] ליאו: אז אומרים "באסה".

אלגוריתם 34 כל המסלולים הקצרים ביותר בזמן יותר סבבה (W)

(1) $n \rightarrow W.rows$
(2) $L^{(1)} \rightarrow W$
(3) כל עוד $m < n-1$:
(א) $L^{(2m)} \rightarrow \text{Extend-Shortest-Paths}\left(L^{(m)}, L^{(m)}\right)$
(ב) $m \rightarrow 2m$
(4) $return\ \ L^{(m)}$

נקבל אם כך זמן ריצה של $\Theta\left(|V|^3 \log |V|\right)$, גם במשקלות שליליים.

האם נוכל למצוא זמן מהיר יותר? מסתבר שכן.

האלגוריתם של פלוייד ורשל.

נניח שאין מסלולים קצרים שליליים בגרף, נוכל למצוא אלגוריתם שרץ בזמן של $O\left(|V|^3\right)$

הגדרה

קודקוד ביניים v_k של מסלול $p = \langle v_i, \ldots, v_k \rangle$ הוא קודקוד שנמצא בין v_i ו-v_j - כלומר $v_k \neq v_i \neq v_j$.

תהי $K = \{v_1, v_2, \ldots, v_k\}$ תת קבוצה של $V = \{v_1, \ldots, v_n\}$ עבור $1 \leq k \leq n$.
לכל זוג $v_i, v_j \in V$ נתבונן בכל המסלולים מ-v_i ל-v_j עם קודקודי ביניים ששייכים ל-K.
כעת, תהי p_{ij} הדרך המינימלית בין v_i ו-v_j.

כעת ישנן שתי אפשרויות:

(א) v_k **איננו קודקוד ביניים של** p_{ij}:
אזי כל קודקודי הביניים במסלול p_{ij} נמצאים בקבוצה $\{1, 2 \ldots, k-1\}$
אם כך, מסלול קצר ביותר מ-i ל-j שכל קודקודי הביניים שלו בקבוצה $\{1, \ldots, k-1\}$ הוא גם מסלול ביניים בין i ל-j, כאשר כל קודקודי הביניים שיכיים לקבוצה $\{1, 2, \ldots, k\}$.

(ב) **אם** v_k **הוא קודקוד ביניים של** p_{ij}:
נחלק את p_{ij} לשני מסלולי. מסלול p_{ik} מ-v_i ל-v_k ומסלול p_{kj} מ-v_k ל-v_j.
מסלול p_{ik} הוא מסלול קצר מ-v_i-לv_k וכן p_{kj} .

בהתבסס על מה שראינו, נציג הגדרה רקרוסיבית של אומדני מסלולים קצרים ביותר.
יהי $d_{ij}^{(k)}$ משקל של מסלול קצר ביותר מ-קודקוד v_i ל-v_j, שכל קודקודי הביניים שלו שייכים לקבוצה $\{1, 2 \ldots, k\}$.
אם $k = 0$ המסלול אינו מכילים קודקודי ביניים, ולכן מורכב מקשת אחת (אם $i \neq j$) ולכן $d_{ij}^{(0)} = w_{ij} /$
אחרת, אם $k \geq 1$ נוכל להתבונן בהגדרה הרקורסיבית - $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$.
ואם נרצה פסואדו קוד:

אלגוריתם 35 פלויד ורשל

(1) $n \rightarrow W.rows$
(2) $D^{(0)} \rightarrow W$
(3) לכל $k \rightarrow 1$ עד n:
(א) לכל $i \rightarrow 1$ עד n:
(i) לכל $j \rightarrow 1$ עד n:
(א') $d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$
(4) תחזיר את $D^{(n)}$.

נקבל אם כך זמן ריצה של $\Theta\left(|V|^3\right)$, גם במשקלות שליליים (כי חסכנו לולאה אחת).
המטריצה המתקבלת הינה המטריצה $D^{(n)} = (d_{ij})^{(n)}$ אשר מקיימת כי $d_{ij}^{(n)} = \delta\left(i,j\right)$ עבור כל $i,j \in V$, כיוון שכל קודקודי הביניים שיכיים ל-$\{1,2,\dots n\}$.

בדוגמה שראינו קודם לכן (קצת לפני מטריצת הסמיכויות).
המטריצות המתקבלות הינן:

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NLL} & 1 \\ \text{NIL} & \text{NL} & \text{NL} & 2 & 2 \\ \text{NIL} & 3 & \text{NL} & \text{NL} & \text{NL} \\ 4 & 1 & 4 & \text{NL} & 1 \\ \text{NL} & \text{NL} & \text{NL} & 5 & \text{NL} \end{pmatrix}$$

ובנוסף:

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

כאשר ישנן שתי דרכים לבנות מסלולים קצרים:

לבנות את המסלולים הקצרים לחשב את $\prod$ מטריצת ה"קודמים".
לחשב את מטירצת הקודמים, כאשר מחשבים את $D^{(k)}$, באמצעות החישוב:

$$\pi_{ij}^{(0)}=\begin{cases}\text{NIL} & \text{if } i=j \text{ or } w_{ij}=\infty\\ i & \text{if } i\neq j \text{ and } w_{ij}<\infty\end{cases}$$

וגם:

$$\pi_{ij}^{(k)}=\begin{cases}\pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)}\leq d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)}> d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\end{cases}$$

מידע מפורט יותר קיים בתרגול.

שיעור מס' 13:

יום רביעי

13.01.21

מבני נתונים של קבוצות זרות

מוטיבציה

לעיתים נרצה ליצור סטים - קבוצות בהן כל איבר מופיע פעם אחת, והחיתוך ביניהן הינו ריק.
נרצה שמבני הנתונים יתמוך בפעולות הבאות:
- $\text{Make-Set}\left(x\right)$: פעולה שיוצרת קבוצה S_x, בה x הוא האיבר היחיד.
- $\text{Union}\left(x,y\right)$: פעולה שמאחדת קבוצות S_x ו-S_y והופכת אותן לקבוצה אחת.
- $\text{Find-Set}\left(x\right)$: פעולה שמחפשת באיזו קבוצה נמצא x, ומחזירה אותו.

במהלך השיעור, נתייחס ל-n בתור **מספר הפעולות של יצירת Make-Set** (מדובר למעשה ב**מספר האיברים הכולל** ולכן במספר הקבוצות הנוצרות מלכתחילה) ול-$m>n$ בתור **מספר הפעולות הכולל של Make-Set, Union, Find-Set.**
כרגיל, נרצה מבנה אלגוריתם יעיל , אך הקושי שלנו מתחיל הוא ב-Union - באיחוד קבוצות. למעשה, יש לנו מלכתחילה n קבוצות, ו-m פעולות, ולכן הפיתרון הפשוט ייקח זמן של $O\left(n\cdot m\right)$.
המקרה הטוב ביותר שנוכל לחשוב עליו, הוא חיפוש בזמן של $O\left(n+m\right)$, כיוון שזמן יצירת הקבוצות הבסיסי הוא n, ולאחר מכן מתבצעות בהכרח m פעולות (נרצה שיהיה קשר ליניארי ולא כפלי).

למעשה, כבר פגשנו בעיה זו בשני אלגוריתמים בגרפים.
(1) בהינתן גרף בלי מכוון ולא ממושקל, רצינו למצוא את **מספר רכיבי הקשירות**. נוכל להסתכל על כל אחד מרכיבי הקשירות בתור קבוצה, ולבחור לכל קבוצה נציג.
(2) בהינתן גרף קשיר, ממושקל ולא מכוון, רצינו למצוא את **העץ הפורש המינימלי**. באלגוריתם החמדן של קרוסקל, איחדנו בכל פעם את הקודקודים, ולכן יש כאן יישום של Union-Find.

נתמקד קודם כל במציאת רכיבי הקשירות, וניזכר באלגוריתם שראינו:

אלגוריתם 36 מציאת רכיבי קשירות בגרף

(1) לכל קודקוד $v \in G$:
  (א) תבצע $\text{Make-Set}\left(v\right)$
(2) לכל צלע $\left(u,v\right) \in E$:
  (א) אם $\text{Find-Set}\left(u\right) \neq \text{Find-Set}\left(v\right)$:
    (i) תבצע $\text{Union}\left(u,v\right)$

למעשה, יש כאן שתי לולאות, הראשונה לוקחת $O\left(|V|\right)$ והשנייה תלויה ב-Find-Set ו-Union.
על מנת לחפש האם שני נציגים נמצאים באותו רכיב קשירות, נוכל להשתמש ב-Find-set, בצורה פשוטה.

לגבי האלגוריתם של קרוסקל, ראינו כי המטרה היא למצוא עץ פורש מינימלי.
נזכיר את האלוגריתם:

אלגוריתם 37 האלגוריתם של קרוסקל

(1) $A = \emptyset$
(2) לכל $v \in V$ תיצור קבוצה v.
(3) תמיין את הצלעות לפי משקלים עולים.
(4) לכל צלע $e = \left(u,v\right) \in E$:
  (א) אם SetFind- $\left(v\right)$ $\neq$ SetFind- $\left(u\right)$ (לא שייכים לאותו עץ) :
    (i) $A \to A \cup \{e\}$
    (ii) תבצע $\text{Union}\left\{u,v\right\}$
(5) תחזיר את A

ראינו כי בהתחלה הסיבוכיות היא $O\left(|V|\right)$. ובהמשך, מתבצע מיון של הצלעות, ולכן זמן הריצה ייקח$|E| \cdot \log|E|$.
לבסוף, ישנו זמן ריצה של $|E| \cdot f\left(|V|\right)$, כאשר f הינו זמן הריצה התלוי במימוש של איחוד קבוצות, שאנחנו מחפשים.

באופן כללי, עבור שני האלגוריתמים, הפעולה Make-Set לוקחת $\Theta\left(1\right)$ לכל קודקוד, ולכן בסך הכל $O\left(|V|\right)$.
זמן הריצה של m פעולות $\text{Union}\left(u,v\right)$ ו-Find-Set$\left(v\right)$ תלוי במימוש.
כל פעולה עלולה לקחת $O\left(i\right)$ זמן, כאשר i הוא מספר הקודקודים בקבוצה מסוימת, ולכן דבר זה יכול לקחת $O\left(m \cdot n\right)$.

אנחנו מעוניינים לשפר את $O\left(n \cdot m\right)$ - כאמור, אנחנו יודעים שהחסם התחתון הוא $O\left(n + m\right)$.

לא נחפש מצב בו כל פעולה תיקח זמן קצר ביותר, אלא אנחנו מעוניינים בסך כללי של פעולות שיהיה קצר ביותר.
כעת, ישנם שני פתרונות אפשריים:
(1) שימוש ברשימות מקושרות - באמצעות חישוב של אורך הרשימה. מימוש בזמן $O\left(m + n\log n\right)$.
(2) שימוש בעצים - דבר שלוקח $O\left(m \cdot \alpha n\right)$, כאשר $\alpha\left(n\right)$ היא פונקציה שגדלה באופן איטי מאוד של $\left(n^n\right) =$ $\log(\log(\log\ldots(n))$ - זהו זמן כמעט ליניארי.

פיתרון באמצעות רשימות מקושרות.
כל קבוצה ממומשת באמצות רשימה מקושרת L_i.
לרשימה ישנם שני שדות - מצביע **לאיבר הראשון** ומצביע ל**איבר האחרון**.
הנציג של הקבוצה יהיה **ראש הרשימה -** דבר שיאפשר מציאת נציג ב-$O\left(1\right)$.

כל אחד מהאיברים ברשימה המקושרת יכללו שלוש שדות: הערך של האיבר, מצביע לאיבר הבא ומצביע לאיבר הראשון ברשימה.

Make-Set ייקח בפשטות $O\left(1\right)$.

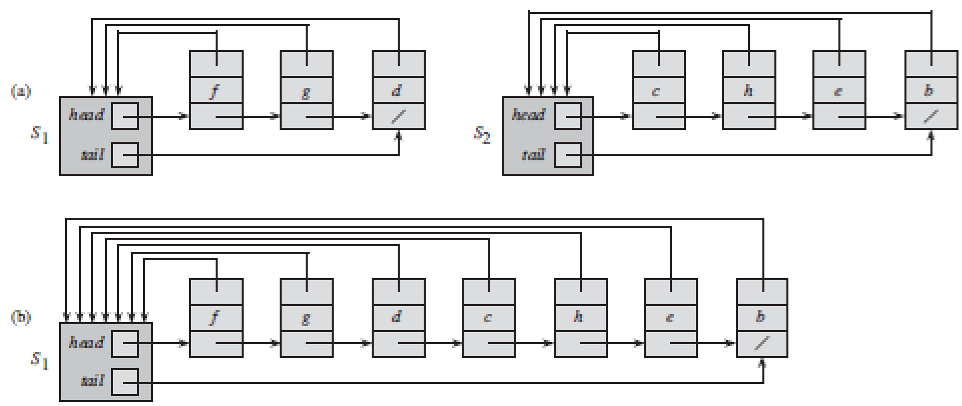
כיצד נוכל לעשות Find-Set? נחפש האם קיים איבר ברשימה, דבר שייקח $O\left(n\right)$ במקרה הגרוע.

כיצד נוכל לעשות Union?

נחבר את **הזנב** של L_1 **לראש** L_2 ונעדכן את כל האיברים ב-L_2 שיצביעו על **ראש** L_1.

אם כך, פעולה זו תיקח $O\left(n\right)$.

בקצרה, זה נראה כך:



בקלות נוכל לבנות דוגמא שתיקח לנו $O\left(n\cdot m\right)$, כפי שאנחנו לא מעוניינים.

נניח כי ישנם $x_1,\ldots,x_n$ איברים. אנחנו יוצרים n קבוצות עבור כל אחד מהאיברים, ולאחר מכן מבצעים $n-1$ פעולות איחוד. בכל פעם שאנחנו מאחדים קבוצה, אנחנו מבצעים $O\left(i\right)$ פעולות עדכון, כאשר i הינו גודל הקבוצה המתאחדת.

במצב כזה, נקבל כי פעולות העדכון הולכות ועולות.

למעשה נקבל בסך הכל:

$$\sum_{i=1}^{n} i=\Theta\left(n^2\right)$$

אם כך, מספר הפעולות הכולל הוא $m=2n-1$, ולמעשה קיבלנו שוב זמן ריצה של $O\left(m\cdot n\right)$, בשונה ממה שרצינו.

אמנם, ניקח את כל הפעולות, נוכל למצוא כי בממוצע, כל פעולה לוקחת $O\left(n\right)$ (כי יש זמן ריצה של $m\cdot n$ בסך הכל ויש m פעולות), ולכן **הזמן לשיעורין** הינו $\Theta\left(n\right)$.

היוריסטיקה של איחוד משוקלל

ישנה דרך לייעל את זמן הריצה. בכל פעם נאחד את הרשימה הקצרה עם הרשימה הארוכה, כיוון שיהיו לנו פחות מצביעים לעדכון.

אם כך, נוסיף שדה לראש הרשימה, שנקרא **גודל הרשימה**, באתחול הרשימה הוא יהיה 1, ובכל איחוד של קבוצה נעלה אותו ב-1.

בכל פעם ניקח את הרשימה הקטנה יותר, ונאחד אותה עם הגדולה

גם כאן יכולה להיות בעייה, כי פעולה בודדת עלולה לקחת $\Omega(n)$ זמן, אם כל אחת משתי הקבוצות מכילה $\Omega(n)$ איברים.

אמנם, אנחנו מתבוננים **בסך הכולל של הפעולות**. נרצה להראות שמספר הפעולות שאנחנו מעדכנים מצביע מסוים, חסום על ידי $\log n$ פעמים - כלומר, בלתי אפשרי שאיבר מסוים יהיה כל פעם בקבוצה הקטנה יותר.

משפט

כאשר משתמשים בייצוג קבוצות זרות על ידי רשימות מקושרות ובהיוריסטיקה של איחוד משוקלל, סדרה שלm פעולות Make-Set,Union,Find-Set אשר n פעולות מהן הן פעולות Make-Set מתבצעת בזמן $O(m+n\log n)$.

הוכחה

ראשית, נתבונן בקבוצה S בגודל n. עבור כל $x\in S$ ונמצא חסם עליון על מספר הפעמים שעודכן x לנציג בקבוצה אחרת.

יהי $x\in S_1$.

בעקבות השימוש בהיוריסטיקת האיחוד המשוקלל, בהכרח עדכון x גורר כי S_1 הייתה הקבוצה הקטנה יותר. אם כך, בפעם הראשונה שעודכן מצביע מ-x לנציג בקבוצה S_2 , הקבוצה S_2 מכילה 2 איברים לפחות. נסמן $S_2=S_3$.

כעת, $x\in S_3$, ולכן בפעם הבאה ש-x עודכן לקבוצה S_4, הקבוצה S_4 מכילה בהכרח 4 איברים.

אם נמשיך כך, עבור כל $k\leq n$, אחרי שעוכדן המצביע מ-x לנציג $\lceil\log k\rceil$, הקבוצה שהתקבלה הכילה לפחות k איברים.

כאמור, הקבוצה הגדולה ביותר מכילה n איברים, ולכן בכל עצם, המבציע לנציג עודכן $\lceil\log n\rceil$ פעמים לכל היותר.

כלומר, סך כל הפעולות העדכון הינו $O(n\log n)$.

מצאנו את זמן הריצה של האיחוד. כל פעולת יצירת קבוצה וחיפוש קבוצה לוקחת $O(1)$, ויש לנו $O(m)$ כאלו (תשימו לב שזה לא לגמרי מדויק, כי יש למעשה מתוכם n פעולות של יצירת קבוצות, וה-$n\log n$ הוא גם חלק מה-m).

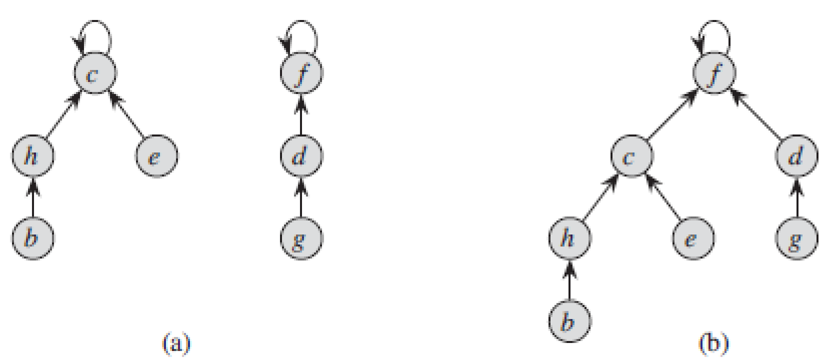
לכן נקבל בסוף הכל זמן ריצה של $O(m+n\log n)$.

יערות של קבוצות זרות.

נניח כי כל קבוצה הינה עץ.

כל איבר בקבוצה מצביע על ה"אב" שלו בעץ. השורש עצמו מצביע לעצמו.

על מנת לעשות איחוד בין שתי קבוצות, נרצה כי עץ אחד יהיה תת עץ של העץ השני.



על מנת לבצע Make-Set, פשוט ניצור עץ ב-$O\left(1\right)$. על מנת לעשות Find-Set - נעבור על העץ, עד שהאב של הקודקוד יהיה הקודקוד עצמו (ובכך הגענו לשורש העץ). דבר זה יכול לקחת $O\left(h\right)$- כאשר h הוא גובה העץ.
על מנת לבצע Union- למשל של b ו-d - נחפש את b ב-$O\left(h\right)$ של עץ, נמצא את השורש שלו ובצורה דומה גם עבור d. לבסוף, נאחד בין השורשים של עצי b ו-d.
בצורה כזאת, אם גובה העץ המקורי הוא h, כאשר נוסיף לו איבר הוא יהיה כעת $h+1$.

בהכללה, נוכל לומר כי העצים גדלים לפי סדרת הפעולות.
נבחין כי עד כה לא שיפרנו את הביצועים ביחס למימוש שעשינו באמצעות רשימות מקושרות (אפילו למקרה הטרוויאלי).
על מנת לפתור זאת נשתמש בשתי שיטות שיאפשרו לנו לצמצם את זמני הריצה.
בדומה לרשימות המקושרות, גם במקרה שלנו נוכל לצרף את הקטן יותר לגדול - **איחוד על פי הדרגה**.
השיטה השנייה בה נשתמש נקראת "כיווץ מסלולים".

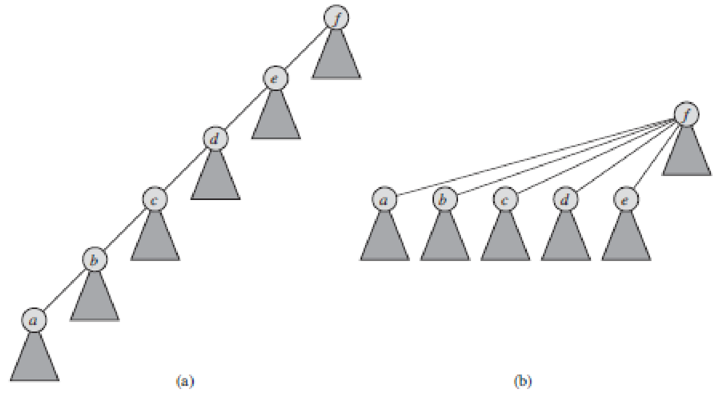
באיחוד על פי הדרגה, נשתמש בצורה דומה לרשימות המקושרות. נאתחל את העץ עם גובה דרגה של 1, ובכל פעם שנבצע איחוד, נעלה את הדרגה לפי גודל העץ שנוסף.

טענה
הגובה המקסימלי של עץ באמצעות שימוש באיחוד על פי הדרגה, הוא $O\left(\log n\right)$.
הוכחה
אינדוקציה על מספר פעולות האיחוד שהשתמשנו על מנת לבנות את העץ.
לפי הנחת האינדוקציה, כאשר גובה העץ הינו h, מספר העלים האפשריים הוא 2^h.
<u>בסיס האינדוקציה כאשר $h=1$:</u>
הטענה נכונה עבור האיחוד הראשון, כיוון שלעץ המתקבל יש בדיוק שני קודקודים, ולכן $\log 2=1$.

<u>הנחת האינדוקציה:</u>
נניח כי הטענה נכונה עבור עץ בגובה h.

<u>צעד האינדוקציה:</u>
כעת עבור האיחוד הבא, ייתכנו שני מקרים:
במקרה בו גובה העץ המצורף קטן יותר - צירוף איבר איננו מגדיל את הגובה.
במקרה בו שני גבהי העץ שווים - לפי הנחת האינדוקציה, לכל עץ יש 2^h עלים, ולכן הוספת 2^h עלים נוספים שווה ל-2^{h+1}.
כיוון שגובה העץ עולה גם גם הוא ב-1,הוכחנו את צעד האינדוקציה.
אנחנו מבצעים m פעולות כאלו, ולכן דבר זה ייקח $O\left(m\log n\right)$.

כעת נתבונן בשיטה השניה - כיווץ מסלולים.



הרעיון הינו כזה. במקום לחבר את האיברים אחד לשני, נוכל לחבר את העץ החדש ישירות לשורש המקורי. במצב זה, איננו צריכים לעדכן את הערכים של העצים שאנו מוסיפים, רק לעדכן את **השורש**.
על מנת לאחד בין שתי קבוצות, עלינו לבדוק באיזה קבוצה כל איבר נמצא - זהו הדבר שייקח יותר זמן. לכן, במהלך חיפוש הקבוצה, נוכל לבצע פעולה של כיווץ מסלול ולמעשה כשנעבור על כל קודקוד, נוכל לאחד אותו רקורסיבית עם האב.

נוכל למצוא פסאודו קוד לשיטה זו:

אלגוריתם 38 אלגוריתמים של יערות של קבוצות זרות

- $Make-Set\left(x\right)$
  (1) $x.parent \to x$
  (2) $x.rank \to 0$
- $Find-Set\left(x\right)$
  (1) אם $x \neq x.parent$ אזי $x.parent \to Find-Set\left(x.parent\right)$
- $Link\left(x,y\right)$
  (1) אם $x.rank > y.rank$:
    (א) $y.parent \to x$
  (2) אחרת:
    (א) $x.parent \to x$
    (ב) אם $x.rank = y.rank$:
      (i) $y.rank \to y.rank + 1$
- $Union\left(x,y\right)$
  (1) $Link\left(Find-Set\left(x\right), Find-Set\left(y\right)\right)$

טענה

זמן הריצה הגרוע ביותר של n פעולות Make-Set ו-$c \leq m$ פעולות Find-Set הוא:

$$\Theta\left(n + c\left(\log_{2+\frac{c}{n}} n\right)\right)$$

בעקבות כך, הסיבוכיות של כיווץ מסלולים הינה $O\left(n + m\log n\right)$.

הוכחה

בספר.

מה קורה בשילוב שתי השיטות?

משפט

בשיטת "יערות של קבוצות זרות", ובשימוש משולב של איחוד על פי דרגה וכיווץ מסלולים, סדרת $m > n$ פעולות של Find-Set,Make-Set ו-Union, כאשר n מתוכם הם פעולות Make-Set, תיקח במקרה הגרוע בזמן $O\left(m \cdot \alpha\left(n\right)\right)$, כאשר $\alpha\left(n\right)$ היא הפונקציה ההופכית לפונקצית אקרמן.
$\alpha\left(n\right)$ היא פונקציה הגדולה באופן איטי מאוד:

$$\alpha(n) < \log^*(n) = \log(\log(\log\ldots(n)\ldots)$$

למעשה, פונקציה זו נעה כל כך לאט, כך ש-$\alpha\left(n\right) \leq 5$ עבור n מספיק גדול.
הוכחה
לא במסגרת הקורס.[13]

שימוש באלגוריתמים שראינו

באלגוריתם של מציאת רכיבי קשירות, הסיבוכיות תהיה:

$$O\left(|V| + |E| \cdot \alpha\left|V\right|\right)$$

מדובר בזמן כמעט ליניארי - לצרכים פרקטיים מדובר בזמן ליניארי.
בצורה דומה, עבור קרוסקל נקבל $O\left(|V| + |E| \log|V|\right)$.

[13] ההוכחה מתבססת בעיקר על כך שהניתוח לשיעורין של Make-Set, Link ו-Find-Set הוא בהתאמה $O\left(\alpha\left(n\right)\right), O\left(1\right)$ ו-$O\left(\alpha\left(n\right)\right)$, ולכן הזמן הכולל הוא $O\left(m \cdot \alpha\left(n\right)\right)$.