מבני נתונים (67109) - סיכום הרצאות

משה קול (moshe.kol@mail.huji.ac.il)

סיכום הרצאות בקורס מבני נתונים (67109) שהועברו באוניברסיטה העברית תשע״ח 2018.

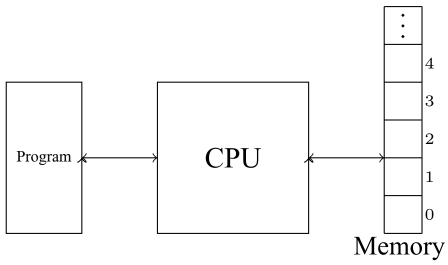
תוכן העניינים

1 שבוע 1 - 18.03.18 - פרופ׳ מיכאל בן-אור

1.1 מודל החישוב

מודל חישוב הוא מודל המתאר פעולות חישוב מותרות ואת העלות שלהן. לפני שנוכל לנתח אלגוריתמים מבחינת זמן ריצה ומקום, עלינו להסכים על מודל חישוב מסוים. מודל החישוב שילווה אותנו הנו מודל RAM (Random-Access Machine).
מודל RAM מורכב מהמרכיבים הבאים:

- מעבד (CPU)
- זיכרון המשמש לשמירת התוכנית
- זיכרון "גישה-אקראית"
- פעולות בסיסיות: קריאה וכתיבה לזיכרון, חיבור/חיסור, כפל/חילוק, השוואה



איור 1.1: תיאור מודל RAM

במודל RAM, מחיר "אחיד" (Unit Cost) עבור הפעולות הבסיסיות.
חשוב להסביר כי מודל חישוב זה מפשט את המימוש בפועל של מחשבים. לדוגמה, במודל זה אין התייחסות להיררכיית זיכרון (Cache Memory). יחד עם זאת, הניסיון מלמד שניתוח אלגוריתמים במודל RAM מנבא בקירוב טוב את ביצועי האלגוריתם במציאות.
במודל RAM ניתן לשכן בקלות יחסית מבני נתונים. לדוגמה:

- מערך (Array)

1	2	3	$\cdots$	n
7	10	139	$\cdots$	71

איור 1.2: תיאור מערך במודל RAM

- רשימה מקושרת (Linked List)

$$\boxed{7} \leftrightarrows \boxed{10} \leftrightarrows \boxed{139} \leftrightarrows \cdots \leftrightarrows \boxed{71}$$

איור 1.3: תיאור רשימה מקושרת דו-כיוונית במודל RAM

1.2 תיאור סדרי גודל

גבול אסימפטוטי עליון (Big-O Notation)

הגדרה 1.1 (גבול אסימפטוטי עליון)**.** תהיינה $\mathbb{R} \to \mathbb{N} : f, g$ פונקציות מהטבעיים לממשיים. בה"כ $f(n), g(n) \geq 0$. נאמר כי הפונקציה f היא O-גדול (Big-O) של g אם קיים $c \in \mathbb{R}$ קבוע ו-$n_0 \in \mathbb{N}$ כך שלכל $n \geq n_0$ מתקיים: $f(n) \leq c \cdot g(n)$. במקרה כזה מסמנים: $f(n) = \mathrm{O}(g(n))$.

הערה 1.2. יש להתייחס לסימון $f(n) = \mathrm{O}(g(n))$ במידה רבה של זהירות. באגף ימין מצויה למעשה קבוצה של פונקציות.

גבול אסימפטוטי תחתון (Ω Notation)

הגדרה 1.3 (גבול אסימפטוטי תחתון)**.** תהיינה $f, g : \mathbb{N} \to \mathbb{R}$. נאמר כי f היא Ω של g אם"ם קיים $c \in \mathbb{R}$ קבוע ו-$n_0 \in \mathbb{N}$ כך שלכל $n \geq n_0$ מתקיים: $f(n) \geq c \cdot g(n)$. מסמנים: $f(n) = \Omega(g(n))$.

גבול אסימפטוטי הדוק (Θ Notation)

הגדרה 1.4 (גבול אסימפטוטי הדוק)**.** תהיינה $f, g : \mathbb{N} \to \mathbb{R}$. נאמר כי f היא Θ של g אם"ם $f(n) = \mathrm{O}(g(n))$ וגם $f(n) = \Omega(g(n))$. במקרה כזה מסמנים: $f(n) = \Theta(g(n))$.

דוגמה 1.5. נתונות הפונקציות $f, g : \mathbb{N} \to \mathbb{R}$ המוגדרות באופן הבא:

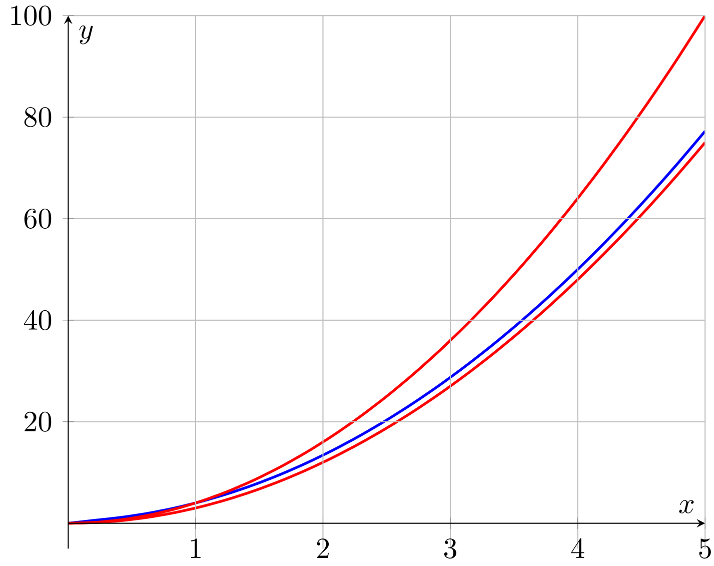
$$g(n) = n^2, \quad f(n) = 3n^2 + \sqrt{n}$$

נבחין כי מתקיים לכל n טבעי:

$$3n^2 < 3n^2 + \sqrt{n} \leq 4n^2$$

ולכן מתקיים ע"פ הגדרה:

$$f(n) = \Theta(g(n))$$



איור 1.4: הפונקציה $3x^2 + \sqrt{x}$ "כלואה" בין $4x^2$ ל-$3x^2$ לכל $x \in \mathbb{N}$. המחשה לגבול אסימפטוטי הדוק.

1.3 בעיית המילון

בעיה 1.6 (בעיית המילון)**.** נתונה קבוצה בעלת n איברים $S = \{a_1, a_2, \dots, a_n\}$. תהא U הקבוצה ממנה נלקחו איברי S, זאת אומרת, $S \subseteq U$. בהינתן $x \in U$, האם $x \in S$?

הערה 1.7. U יכולה להיות קבוצת כל המספרים עם 64 ביט, שמות כל הסטודנטים וכדומה.

נניח תחילה שהקבוצה S קבועה. באמצעות n פעולות ניתן לעבור על הרשימה כפי שהיא נתונה. נוכל לבצע מיון של איברי S בסדר עולה, ולשמור את התוצאה במערך A. נניח כי אחרי המיון, איברי הקבוצה S הם

$$a_1 \le a_2 \le \cdots \le a_n$$

$$A : \boxed{\begin{array}{c|c|c|c} a_1 & a_2 & \dots & a_n \end{array}}$$

נציג כעת אלגוריתם יעיל יותר לפתרון הבעיה.

חיפוש בינארי

תיאור האלגוריתם בהינתן $x \in U$ נחשב את האינדקס של האיבר האמצעי $k = \left\lfloor \frac{n}{2} \right\rfloor$.
אם x שווה לאיבר האמצעי a_k סיימנו.
אחרת, אם $x > a_k$ נמשיך לחפש את x בחצי העליון של המערך.
אם $x < a_k$ נמשיך לחפש את x בחצי התחתון של המערך.
אם הרשימה קצרה מ-1 אז התשובה שלילית.

שמורה של האלגוריתם התכונה הנשמרת לכל אורך האלגוריתם היא:
"אם x נמצא במערך הממויין, אזי הוא נמצא בחלק שנותר לטפל בו."

הוכחת נכונות האלגוריתם נוכיח באינדוקציה על n - גודל הקלט.
עבור $n = 1$ התוכנית פועלת נכון שכן חישוב האינדקס האמצעי הוא 0 (האינדקס היחיד).
נניח שהתוכנית פועלת נכון עבור מערכים בגודל קטן ממש מ-n ונוכיח שהתוכנית פועלת נכון עבור n.
ביצוע צעד באלגוריתם שומר על השמורה של האלגוריתם. הקריאה הרקורסיבית פועלת נכון תחת הנחת האינדוקציה, ולכן התשובה שתחזור מן הקריאה הרקורסיבית היא נכונה.

ניתוח זמן ריצה נסמן ב-$T(n)$ את מספר הצעדים שמתבצעים בתוכנית עבור הקלט הגרוע ביותר בגודל n. מתקיים:

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

כאשר $c \in \mathbb{R}$ קבוע כלשהו.
נפתח את כלל הנסיגה:

$$\begin{aligned} T(n) &= T\left(\frac{n}{2}\right) + c \\ &= \left(T\left(\frac{n}{4}\right) + c\right) + c \\ &= \dots \\ &\underbrace{=}_{l=\log_2 n} T\left(\frac{n}{2^l}\right) + \underbrace{(c + c + \cdots + c)}_{l \text{ times}} \\ &= \underbrace{c + c + \cdots + c}_{l+1 \text{ times}} \\ &= (l+1)c \\ &= (\log_2(n) + 1)c \end{aligned}$$

על כן מתקיים:

$$T(n) = \mathrm{O}(\log_2 n)$$

בעיית המילון - קבוצה דינאמית

כעת אנו מעוניינים לתמוך בפעולות של הוספת איבר לקבוצה S או הסרת איבר מקבוצה זו.
במערך ממויין, הוספה/הורדה של איבר דורשת $\mathrm{O}\left(n\right)$ פעולות.
לעומת זאת, הוספה/הורדה של איבר מרשימה מקושרת דורשת $\mathrm{O}\left(1\right)$ פעולות אם יודעים את המיקום של האיבר אותו רוצים לשנות. אולם, ברשימה מקושרת חיפוש דורש $\mathrm{O}\left(n\right)$ פעולות.
בהמשך הקורס נראה כיצד ניתן לפתור את בעיית המילון בעזרת מבני נתונים יחסית פשוטים כך שהמחיר יהיה $\mathrm{O}\left(\log n\right)$.

2 שבוע 2 - 26.03.18 - פרופ׳ מיכאל בן-אור

2.1 מיון מיזוג (Merge Sort)

בשבוע 1 תיארנו אלגוריתם יעיל לחיפוש איבר ברשימה - חיפוש בינארי. נזכור כי יש צורך למיין את האיברים לפני הפעלת האלגוריתם. כעת נציג אלגוריתם יעיל למיון הנקרא Merge Sort (״מיון באמצעות מיזוגים״).

תיאור אלגוריתם מיזוג

נניח שנתונות לנו שתי רשימות A ו-B ממויינות בסדר עולה, בגדלים n ו-m בהתאמה. זאת אומרת:

$$A[1] \leq A[2] \leq \cdots \leq A[n]$$
$$B[1] \leq B[2] \leq \cdots \leq B[m]$$

נרצה למזג את שתי הרשימות לרשימה ממויינת C בסדר עולה, באורך $n+m$.

דוגמה 2.1. אם $A=[1,3,5]$ ו-$B=[2,9]$ אז $C=[1,2,3,5,9]$ לאחר הרצת אלגוריתם המיזוג.

אלגוריתם

נשמור מצביע לאיבר הראשון שלא טיפלנו בו בכל רשימה.
נשווה בין האיברים בשתי הרשימות, ונכניס לרשימה C את האיבר הקטן יותר, בתור האיבר הבא ברשימה.
נקדם את המצביע ברשימה שהכילה את האיבר הקטן יותר.

שמורה

כמות ההשוואות באלגוריתם המוצע הוא לכל היותר $n+m-1$.
השמורה המתאימה היא: הרשימה החלקית C ממויינת בסדר עולה, וכל האיברים בה קטנים או שווים מכל האיברים שלא טיפלנו בהם עד כה.

הערה 2.2. לא מספיק להוכיח את השמורה: ״הרשימה החלקית C ממויינת בסדר עולה״ בלבד, למרות שמדובר בשמורה נכונה. הסיבה לכך היא שהעובדה שהרשימה החלקית C ממויינת בסדר עולה לא גוררת שהצעד הבא באלגוריתם אכן שומר על הרשימה C ממויינת. מעת לעת ניתקל בשמורות נכונות, אך לא מספיק ״חזקות״ כדי להוכיח נכונות של האלגוריתם.

תיאור אלגוריתם מיון

נתונה רשימה A לא ממויינת, באורך n.
ייתכנו שני מקרים:

- אם $n=1$ אז הרשימה ממויינת וסיימנו.
- אחרת, נחלק את הרשימה לשתי רשימות באורך $\frac{n}{2}$ (אם n אי-זוגי, אז רשימה אחת באורך $\left\lfloor \frac{n}{2} \right\rfloor$ והשניה באורך $\left\lceil \frac{n}{2} \right\rceil$).

  נמיין את הרשימות ע״י קריאה לאותו אלגוריתם (ברקורסיה).

  נמזג את שתי הרשימות החלקיות לרשימה אחת ממויינת באורך n.

ניתוח זמן ריצה
נסמן ב-$T(n)$ את מספר ההשוואות המקסימלי שהאלגוריתם Merge Sort מבצע במיון רשימה באורך n.
נניח לשם פשטות כי $n=2^k$ עבור $k\in\mathbb{N}$.
מתקיים עבור $T(n)$ כי: $T(n)\leq 2T\left(\frac{n}{2}\right)+n$. הסיבה לכך היא שאנו פעמיים מחלקים את הרשימה לשתי רשימות קטנות יותר באורך $\frac{n}{2}$, ואלגוריתם המיזוג שתיארנו מבצע $\left(\frac{n}{2}+\frac{n}{2}-1\right)=n-1$ השוואות. אנו מתעניינים בחסם מלמעלה, ולכן על מנת להקל על החישוב השמטנו את (-1).

נפתח את הנוסחה הרקורסיבית ונקבל:

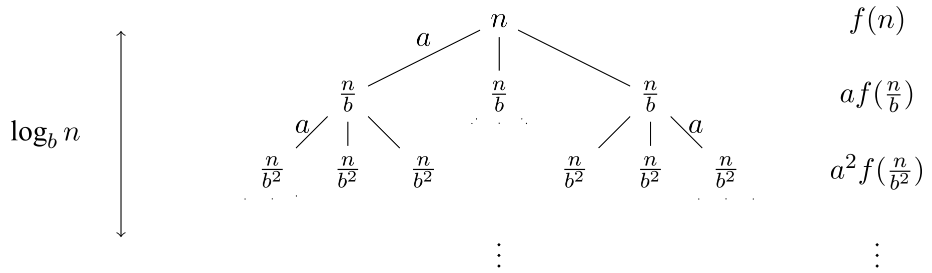
$$\begin{aligned}
T(n) &\le 2T\left(\frac{n}{2}\right)+n \\
&\le 2\left(2T\left(\frac{n}{4}\right)+\frac{n}{2}\right)+n \\
&= 4T\left(\frac{n}{4}\right)+n+n \\
&\vdots \\
&\le 2^k \underbrace{T\left(\frac{n}{2^k}\right)}_{=T(1)=0}+\underbrace{(n+n+\cdots+n)}_{k\text{ times}} \\
&= nk \\
&= n\log_2(n)
\end{aligned}$$

לכן זמן הריצה של האלגוריתם Merge Sort הוא $O(n\log(n))$.

2.2 משפט האב

נתעניין בדרך מוכללת לפתור נוסחאות רקורסיביות מהצורה:

$$T(n)=aT\left(\frac{n}{b}\right)+f(n); \quad T(1)=1$$



איור 2.1: הדגמה לעץ רקורסיבי שאנו מעוניינים לפתור

המשפט הבא הוא משפט חשוב בניתוח זמן ריצה של אלגוריתמים רקורסיביים.

משפט 2.3 (משפט האב)**.** יהיו $a\ge 1$, $b>1$ קבועים ותהי $f:\mathbb{N}\to\mathbb{R}^+$. נתונה פונקציה $T:\mathbb{N}\to\mathbb{R}^+$ המוגדרת באופן הבא:

$$T(n)=aT\left(\frac{n}{b}\right)+f(n); \quad T(1)=1$$

נסמן $p=\log_b(a)$. אזי $T(n)$ מתנהגת באופן אסימפטוטי כך:

- אם קיים $\varepsilon>0$ כך ש-$f(n)=O(n^{p-\varepsilon})$ אז $T(n)=\Theta(n^p)$.
- אם $f(n)=\Theta(n^p)$ אז $T(n)=\Theta(n^p\log(n))$.
- אם קיים $\varepsilon>0$ כך ש-$f(n)=\Omega(n^{p+\varepsilon})$ וגם קיים קבוע $0<c<1$ כך שמתקיים:

$$af\left(\frac{n}{b}\right)\le cf(n)$$

אז $T(n)=\Theta(f(n))$.

הערה 2.4. שימו לב כי במקרה השלישי במשפט האב התווסף תנאי על $f(n)$. נראה שתנאי זה מתקיים רק מהדרישה $f(n)=\Omega\left(n^{p+\varepsilon}\right)$, במקרה בו $f(n)=n^d$ (כלומר, $f(n)$ היא פונקציה פולינומיאלית).
מכך ש-$f(n)=\Omega\left(n^{p+\varepsilon}\right)$ אנו מקבלים $d>p$. קיים $q\in\mathbb{R}$ כך ש-$p-d\le q<0$.
נסמן $c=b^q$. אזי $0<c<1$ כי $q<0$ ובנוסף $q=\log_b(c)$.
נזכור כי $p=\log_b(a)$ ולכן מתקיים:

$$\begin{aligned}&\log_b(a)-d\le\log_b(c)\\ \Leftrightarrow&\log_b\left(\frac{a}{b^d}\right)\le\log_b(c)\\ \Leftrightarrow&\frac{a}{b^d}\le c\\ \Leftrightarrow&a\left(\frac{n}{b}\right)^d\le cn^d\\ \Leftrightarrow&af\left(\frac{n}{b}\right)\le cf(n)\end{aligned}$$

כפי שרצינו.

דוגמה 2.5. נתבונן בדוגמאות הבאות:

1. נתון: $T(n)=2T\left(\frac{n}{2}\right)+n$.
מתקיים $p=\log_2 2=1$ ולכן $f(n)=n=\Theta\left(n^1\right)$.
אנו נמצאים במקרה השני של משפט האב, ולכן $T(n)=\Theta(n\log(n))$.
2. נתון: $T(n)=T\left(\frac{n}{2}\right)+n^2$.
מתקיים $p=\log_2 1=0$. למשל עבור $\varepsilon=1$ נקבל כי $f(n)=n^2=\Omega\left(n^{0+1}\right)$.
עבור $c=\frac{1}{2}$ נקבל כי:

$$1\cdot f\left(\frac{n}{2}\right)=\frac{n^2}{4}\le\frac{1}{2}n^2=cf(n)$$

על כן מתקיימים תנאי המקרה השלישי של משפט האב, ולכן $T(n)=\Theta\left(n^2\right)$.

הוכחה. (הוכחה חלקית למשפט האב) נוכיח רק במקרה שבו $n=b^k$ כאשר b שלם, n חזקה שלמה של b.
נוכיח את המקרה השני של משפט האב באינדוקציה על n.
נבחר $c>0$ כך שמתקיים $f(n)\le cn^p$ לכל $n\in\mathbb{N}$. (הערה: בפועל נתון כי $f(n)=\Theta(n^p)$, לפי הגדרת נוטציית Θ אנו מקבלים כי החל ממקום מסוים קיים קבוע $c>0$ עם $f(n)\le cn^p$. אולם, יש מספר סופי של איברים שלא מקיימים את התנאי הנ"ל וע"י הגדלת הקבוע c נוכל לקבל חסם לכל $n\in\mathbb{N}$).
יתר על כן, נדאג גם שיתקיים $T(n)\le cn^p\log_b(n)$ עבור $n\le b$.
בסיס: לכל $n\le b$ מתקיים $T(n)\le cn^p\log_b(n)$.
צעד: נניח שלכל $m<n$ מתקיים $T(m)\le cm^p\log_b(m)$ ונוכיח עבור n (נזכור כי n הוא חזקה שלמה של b).

$$\begin{aligned}T(n)&=aT\left(\frac{n}{b}\right)+f(n)\\ &\le a\underbrace{c\left(\frac{n}{b}\right)^p\log_b\left(\frac{n}{b}\right)}_{\text{induction hypothesis}}+cn^p\\ &\underbrace{\le}_{a=b^p}cn^p\left(\log_b(n)-1\right)+cn^p\\ &=cn^p\log_b(n)\end{aligned}$$

ולכן $T(n)=O\left(n^p\log_b(n)\right)$.
חוזרים על טיעון דומה עבור חסם תחתון.
נוכיח את המקרה השלישי של משפט האב:

לפי הגדרת הרקורסיה מתקיים כי:

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

ולכן $T(n) = \Omega(f(n))$.
נוכיח באינדוקציה על n כי מתקיים: $T(n) = O(f(n))$.
נזכור כי נתון במקרה זה $af\left(\frac{n}{b}\right) \leq cf(n)$, כאשר $0 < c < 1$.
נבחר r_1 כך ש-$T(n) \leq rf(n)$ עבור $n \leq b$. נבחר r_2 כך ש-$r_2 \geq \frac{1}{1-c}$. נגדיר: $r = \max\{r_1, r_2\}$.
עלינו להראות שקיים $r \in \mathbb{R}$ כך ש-$T(n) \leq rf(n)$ (נוכיח באינדוקציה רק על חזקות של b, לשם פשטות).
בסיס: לכל $n \leq b$ מתקיים $T(n) \leq rf(n)$.
צעד: נניח נכונות הטענה לכל $m < n$ ונוכיח עבור n (n חזקה שלמה של b).

$$\begin{aligned}
T(n) &= af\left(\frac{n}{b}\right) + f(n) \\
&\leq a\left(rf\left(\frac{n}{b}\right)\right) + f(n) \\
&= r\left(af\left(\frac{n}{b}\right)\right) + f(n) \\
&\leq r(cf(n)) + f(n) \\
&= (rc+1) f(n) \\
&\leq rf(n)
\end{aligned}$$

בזאת הסתיים צעד האינדוקציה, ומתקיים $T(n) = O(f(n))$.
לפיכך, בשילוב עם $T(n) = \Omega(f(n))$ מקבלים $T(n) = \Theta(f(n))$, כנדרש. $\square$

3 שבוע 3 - 15.04.18 - פרופ׳ מיכאל בן-אור

3.1 השלמת הוכחת משפט האב

מבוא אינטואיטיבי

ראשית, ניתן מבוא אינטואיטיבי למשפט האב. נחזור ונתבונן באיור 2.1 משבוע 2.
אנו מתבוננים בבעיה בגודל n, ומפרקים אותה ל-a בעיות שכל אחת מהן בגודל $\frac{n}{b}$. כל בעיה בגודל $\frac{n}{b}$ אנו מפרקים ל-a בעיות נוספות שכל אחת מהן בגודל $\frac{n}{b^2}$, וכן הלאה. ״התשלום״ עבור שורש העץ הנו $f\left(n\right)$, ״התשלום״ עבור כל הקודקודים במרחק 1 מן השורש הנו $af\left(\frac{n}{b}\right)$, ובאופן כללי ״התשלום״ עבור כל הקודקודים במרחק i מן השורש הנו $a^if\left(\frac{n}{b^i}\right)$. עלותו של כל עלה היא $T\left(1\right)=1$, וכל עלה נמצא במרחק של $\log_b\left(n\right)$ מן השורש. מספר העלים הנו $a^{\log_b\left(n\right)}$.
לפי חוקי לוגריתמים מתקיים:

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

לפי ניתוח זה, מספר הפעולות $T\left(n\right)$ הוא הסכום

$$\underbrace{n^{\log_b\left(a\right)}}_{\text{מספר העלים}}+\underbrace{\sum_{i=0}^{\log_b\left(n\right)-1}a^if\left(\frac{n}{b^i}\right)}_{\text{סיכום העלויות של כל אחת מהרמות בעץ}}=f\left(n\right)+af\left(\frac{n}{b}\right)+a^2f\left(\frac{n}{b^2}\right)+\cdots+n^{\log_b\left(a\right)}$$

.
אם הפונקציה f גדלה לאט יחסית ל-$n^{\log_b\left(a\right)}$ אז כל הסכום $f\left(n\right)+af\left(\frac{n}{b}\right)+a^2f\left(\frac{n}{b^2}\right)+\cdots+n^{\log_b\left(a\right)}$ מתנהג ״כמו״ $n^{\log_b\left(a\right)}$. זהו המקרה הראשון במשפט האב שלמעשה גורס שהגורם הדומיננטי בעלות הכוללת של העץ הוא עלותם של העלים.

הערה 3.1. חשוב להבהיר כי משפט האב לא מכסה את כל המקרים של נוסחאות הרקורסיה מהצורה $T\left(n\right)=aT\left(\frac{n}{b}\right)+f\left(n\right)$. במקרה הראשון של משפט האב, הפונקציה $f\left(n\right)$ חייבת להיות קטנה <u>פולינומיאלית</u> מ-$n^{\log_b\left(a\right)}$. כלומר, $f\left(n\right)$ צריכה להיות קטנה מ-$n^{\log_b\left(a\right)}$ פי הגורם n^{ε} (עבור $\varepsilon>0$).
לדוגמה: $T\left(n\right)=2T\left(\frac{n}{2}\right)+n\log\left(n\right)$, לא ניתן לפתור את נוסחת הנסיגה באמצעות משפט האב

הוכחת המקרה הראשון של משפט האב

נסמן מעתה $p:=\log_b\left(a\right)$.
נניח לשם פשטות כי n חזקה של b (אולם המשפט נכון גם כאשר n אינו חזקה של b), ונוכיח באינדוקציה על n.
נבחר קבוע c_1 כך שעבור כל $n\in\mathbb{N}$ מתקיים $f\left(n\right)\le c_1n^{p-\varepsilon}$ (נובע מכך שמתקיים $f\left(n\right)=O\left(n^{p-\varepsilon}\right)$).
<u>בסיס</u>: לכל $d>0$ אפשר לבחור c_2 מספיק גדול כך שעבור n-ים קטנים ($n\le b$) יתקיים $T\left(n\right)\le c_2n^p-dn^{p-\frac{\varepsilon}{2}}$.
(החסרת הגורם $dn^{p-\varepsilon/2}$ חשובה, אחרת ניתקל במבוי סתום בצעד האינדוקציה).
<u>צעד</u>: נניח שמתקיים $T\left(m\right)\le c_2m^p-dm^{p-\frac{\varepsilon}{2}}$ לכל $m<n$ ונוכיח עבור n:

$$\begin{aligned}T\left(n\right)&=aT\left(\frac{n}{b}\right)+f\left(n\right)\\ \text{הנחת האינדוקציה}&\le a\left[c_2\left(\frac{n}{b}\right)^p-d\left(\frac{n}{b}\right)^{p-\frac{\varepsilon}{2}}\right]+f\left(n\right)\\ f\left(n\right)\ \text{תנאי עבור}&\le a\left[c_2\left(\frac{n}{b}\right)^p-d\left(\frac{n}{b}\right)^{p-\frac{\varepsilon}{2}}\right]+c_1n^{p-\varepsilon}\\ &\le c_2n^p-a\cdot d\frac{n^{p-\frac{\varepsilon}{2}}}{b^{p-\frac{\varepsilon}{2}}}+c_1n^{p-\varepsilon}\\ &=c_2n^p-d\frac{n^pb^{\frac{\varepsilon}{2}}}{n^{\frac{\varepsilon}{2}}}+c_1n^{p-\varepsilon}\\ &\le c_2n^p-n^{p-\frac{\varepsilon}{2}}\left[d\cdot b^{\frac{\varepsilon}{2}}-\frac{c_1}{n^{\frac{\varepsilon}{2}}}\right]\end{aligned}$$

אם $d \cdot b^{\frac{\varepsilon}{2}} - \frac{c_1}{n^{\frac{\varepsilon}{2}}} \geq d$ אז סיימנו. נזכור כי $b > 1$ ולכן $b^{\frac{\varepsilon}{2}} > 1$. הביטוי $\frac{c_1}{n^{\frac{\varepsilon}{2}}}$ שואף לאפס כאשר $n \to \infty$ ולכן אם נבחר d מספיק גדול נקבל כי

$$c_2 n^p - n^{p-\frac{\varepsilon}{2}} \left[d \cdot b^{\frac{\varepsilon}{2}} - \frac{c_1}{n^{\frac{\varepsilon}{2}}} \right] \leq c_2 n^p - d n^{p-\frac{\varepsilon}{2}}$$

בזאת הוכחנו כי $T(n) = O(n^p)$. במשפט האב יש גם חסם תחתון על $T(n)$, מניתוח העץ הרקורסיבי שביצענו במבוא האינטואיטיבי עולה כי

$$T(n) \geq n^p T(1)$$

כאשר $n^p T(1)$ היא התרומה של העלים ברקורסיה ואנו מוסיפים רק איברים אי-שליליים.
מכאן נובע: $T(n) = \Theta(n^p)$, כנדרש. ■

הערה 3.2. בהוכחת המקרה הראשון החסרנו את הביטוי $dn^{p-\varepsilon/2}$. אילו לא היינו מחסירים את הביטוי הזה, ההוכחה באינדוקציה הייתה משתבשת. טעות נפוצה בהוכחת חסמים אסימפטוטיים באינדוקציה היא שינוי קבועים במהלך ההוכחה.
למשל, בהוכחת המקרה הראשון של משפט האב שזה עתה סיימנו, אם לא היינו מחסירים את הביטוי $dn^{p-\varepsilon/2}$ אז היינו מקבלים בסיום כי $T(n) \leq c_2 n^p - c_1 n^{p-\varepsilon}$, וניתן לטעון כי מתקיים $c_2 n^p - c_1 n^{p-\varepsilon} \leq c_3 n^p$. זה אמנם נכון **אבל לא מוכיח** כי $T(n) \leq c_2 n^p$. אל לנו לשנות קבועים במהלך ההוכחה. במקרה זה, "הטריק" הוא להחסיר גורם מסדר נמוך.
להלן הוכחה שגויה באינדוקציה "בשיטת שינוי הקבועים". נתונה פונקציה $f(n) = 2^n$ ונרצה "להוכיח" כי מתקיים $f(n) = O(1)$. עבור $n = 1, 2$ מתקיים $f(n) \leq 4 = c$. נניח עבור n "ונוכיח" עבור $n+1$:

$$f(n) = 2^n = 2f(n-1) = f(n-1) + f(n-1) \leq O(1) + O(1) = O(1)$$

זה בוודאי שגוי כי למעשה צעד האינדוקציה מוליד $f(n) \leq 2c$ והרי לא מדובר באותו קבוע c!.

3.2 ניהול תור קדימויות

תור קדימויות (Priority Queue) הוא מבנה נתונים לטיפול בקבוצה של איברים. תור קדימויות תומך בפעולות הבאות:

- Delete-Max - מציאת האיבר המקסימלי והוצאתו מן הקבוצה (אם יש מספר איברים בגודל מקסימלי, הוצאת אחד מהם).
- בניית תור קדימויות - בהינתן n מספרים נרצה להכניס אותם יחד למבנה נתונים ריק.
- הוספת איבר אחד נוסף למבנה קיים (Insert (x)) - לא נטפל בפעולה זאת בקורס.

אנלוגיה: נחשוב על מזכיר רפואי אשר מוטלת עליו המשימה לנהל כניסת מטופלים לרופאים. בהתאם לבעיה הרפואית של כל מטופל, מסדר המזכיר את תור כניסת החולים לרופאים. אם מצבו של חולה מסוים קשה יותר משל האחר, הוא ייכנס לרופא לפניו. המזכיר הרפואי מנהל למעשה תור קדימויות.

3.3 ניהול תור קדימויות בעזרת ערימה

ניתן להשתמש בערימה (Heap) למימוש תור קדימויות. לפני שנמשיך בהצגת הערימה, נזדקק למספר הגדרות בסיסיות.

הגדרה 3.3. **עץ בינארי** הינו עץ שדרגת צומתיו היא לכל היותר 2.

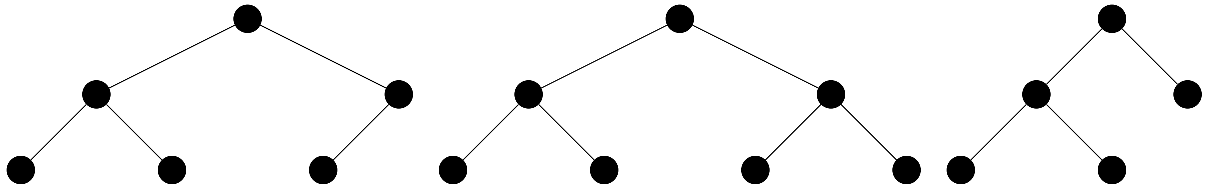
הגדרה 3.4. **עץ בינארי מלא** הנו עץ שבו לכל צומת שאינה עלה יש בדיוק שני בנים.

הגדרה 3.5. **עץ בינארי שלם** הנו עץ בינארי מלא שבו כל העלים נמצאים באותו עומק.

הגדרה 3.6. **עץ בינארי כמעט שלם** הוא עץ בינארי מלא בכל רמותיו, פרט אולי לאחרונה, המלאה משמאל ועד לנקודה מסוימת.

הגדרה 3.7. **העומק** (depth) של קודקוד הוא מרחק הקודקוד מן השורש. העומק של השורש הוא 0.

הגדרה 3.8. **הגובה** של קודקוד הוא המרחק הגדול ביותר לעלה מן הקודקוד. הגובה של העץ הוא גובה השורש. לעץ בעל צומת יחיד (השורש) גובה 0.

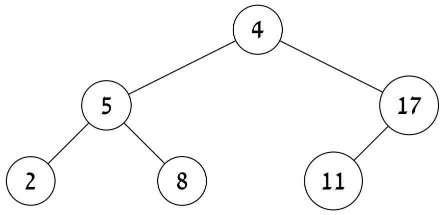


(א) עץ בינארי מלא (ב) עץ בינארי שלם (ג) עץ בינארי כמעט שלם

איור 3.1: דוגמאות לעצים בינאריים

מספר הקודקודים בעץ בינארי שלם בגובה h הנו: $1 + 2 + 2^2 + \cdots + 2^h = 2^{h+1} - 1$ (כאשר h הנו מספר הצלעות במסלול הפשוט הארוך ביותר היורד מצומת לעלה. באיור שלעיל גובה העץ השלם הנו $h = 2$ ומספר הקודקודים בעץ הנו $2^{2+1} - 1 = 7$).

שיכון של עץ בינארי כמעט שלם במערך ניתן לשכן עץ בינארי כמעט שלם במערך, כאשר לכל צומת בעץ מתאים איבר במערך שבו מאוחסן הערך שמכיל הצומת. נראה דוגמה באיור הבא.



(א) עץ בינארי כמעט שלם

4	5	17	2	8	11

(ב) המערך שמתאים לעץ הבינארי מימין

איור 3.2: דוגמה לשיכון עץ בינארי במערך

בהינתן אינדקס k במערך ניתן לחשב את האינדקסים של הבנים:

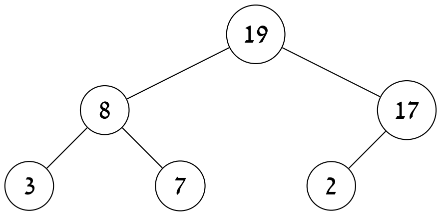
$$\text{left}\,(k) = 2k; \quad \text{right}\,(k) = 2k + 1$$

ואת אינדקס האב:

$$\text{parent}\,(k) = \left\lfloor \frac{k}{2} \right\rfloor$$

אם $\text{left}\,(k)$ או $\text{right}\,(k)$ גדולים מ-n (גודל המערך) אז לא קיים בן כזה. אם $\text{parent}\,(k)$ קטן מ-1 אז אנחנו נמצאים בשורש.

תכונת הערימה במבנה נתונים של ערימת מקסימום (Max Heap), המספרים מאוחסנים במערך בגודל n כך שאם מתבוננים על המערך כעץ בינארי כמעט שלם, מתקיימת תכונת הערימה והיא: המספר שנמצא בקודקוד גדול או שווה למספרים שנמצאים בבנים שלו, אם הם קיימים. בצורה שקולה נאמר: כל קודקוד גדול מכל צאצאיו.



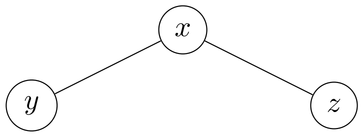
איור 3.3 : דוגמה לעץ בינארי שמקיים את תכונת הערימה

הוצאת מקסימום

נוריד את האיבר באינדקס 1 במערך ובמקומו נשים את האיבר שנמצא במקום ה-nי (נשנה את גודל המערך מ-n ל-$n-1$).
משהחלפנו בין האיבר הראשון לאחרון, ייתכן שפגענו בתכונת הערימה ולכן צריך לתקנה. נשים לב שבעץ שקיבלנו מתקיימת תכונת הערימה בכל קודקוד פרט אולי מאשר בשורש.
פעולת התיקון $\text{Heapify}\left(T\right)$ מקבלת עץ בינארי כמעט שלם המשוכן במערך, T, כאשר ההנחה היא שתכונת הערימה מתקיימת עבור כל קודקוד חוץ אולי מאשר בשורש, ובכך השורש מפר את תכונת הערימה. פעולת התיקון מחזירה ערימה תקנית.
אלגוריתם $\text{Heapify}(T)$:

- אם גודל העץ הוא 1, סיימנו.
- אחרת, נשווה בין שני הבנים (אם יש רק אחד אז נבחר אותו), ונשווה את הגדול מביניהם לשורש.
  - אם השורש גדול יותר, סיימנו.
  - אחרת, נחליף את הערך של הגדול עם השורש ונבצע Heapify על תת-העץ שאליו עבר הערך מן השורש.

מספר הצעדים לתיקון הערימה בעזרת $\text{Heapify}\left(T\right)$ הוא לכל היותר $O(\log n)$.



איור 3.4 : דוגמה להחלפת השורש באחד מבניו. נניח כי x הוא השורש. אם בה"כ $y < z$ וגם $x < z$ אז נחליף בין z ל-x.

בניית ערימה

נתאר את הפונקציה Build-Heap. נתון מערך לא ממויין בגודל n ורוצים לבנות ממנו ערימת מקסימום.

- נתחיל מהשורש ונריץ את Build-Heap על הבנים הימניים והשמאליים (אם יש כאלה).
- נבצע Heapify על השורש.

הערכת הזמן :

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

מתקיים $\log\left(n\right)=O\left(n^{\log_2(2)-0.5}\right)$ ולכן אנו במקרה הראשון של משפט האב, מכאן נובע $T\left(n\right)=\Theta\left(n\right)$.

הערה 3.9. בנינו ערימה ב-$O\left(n\right)$ ולא ב-$O\left(n\log\left(n\right)\right)$, זו למעשה הייתה המוטיבציה מאחורי ניתוח הערימה כמבנה נתונים. תכונה זו תתברר כשימושית בהמשך.

4 שבוע 4 - 22.04.18 - פרופ' מיכאל בן-אור

4.1 השלמת הערכת זמן ריצה - Build-Heap

נשלים את הערכת מספר הפעולות הדרוש לבניית ערימה ממערך לא ממויין - השגרה Build-Heap.
באופן כללי, מבנה הערימה אינו בהכרח "מאוזן" - כזכור, מדובר בעץ בינארי כמעט-שלם, וייתכן כי הרמה התחתונה לא מלאה.
על כן, מספר האיברים בתת-העץ השמאלי אינו שווה למספר האיברים בתת-העץ הימני. לכן אם $T(n)$ מייצג את מספר הפעולות שהשגרה Build-Heap מבצעת על מערך לא ממויין בגודל n, אנו מקבלים כי באופן כללי $T(n) \neq 2T\left(\frac{n}{2}\right) + \log(n)$.
נבצע את האנליזה של זמן הריצה ע"י הוספת קודקודים לערימה כך שנמלא את הרמה התחתונה בעץ. תוספת הקודקודים מייצרת בשבילנו עץ שלם, עבורו כלל הנסיגה יהיה מדויק. נבחין כי הזמן האסימפטוטי יהיה חסום כי לאחר הוספת הקודקודים ייתכן כי מבוצעות פעולות נוספות, אך במקרה הכללי הרקורסיה תיעצר קודם.
בערימה בגודל n ועומק h מתקיים:

$$2^h \leq n \leq 2^{h+1} - 1$$

כאשר משלימים את הערימה ע"י תוספת קודקודים, לכל היותר אנו מכפילים את n פי 2.
נסמן ב-m את מספר הקודקודים החדש בערימה שהיא עץ בינארי שלם. אזי מתקיים:

$$n \leq m \leq 2n$$

כעת נוכל לרשום את כלל הנסיגה בצורה מדויקת:

$$T(m) = 2T\left(\frac{m}{2}\right) + \log_2(m)$$

על-סמך משפט המאסטר 2.3 מקרה ראשון, אנו מקבלים כי $T(m) = \Theta(m)$.
כיוון ש-$n \leq m \leq 2n$ מתקיים $m = \Theta(n)$ ולכן $\Theta(m) = \Theta(n)$.

דרך נוספת נגדיר $S(h)$ - מספר הפעולות המקסימלי ש-Build-Heap מבצעת על עץ בינארי כמעט-שלם בגובה h.
מתקיים:

$$S(h) \leq 2S(h-1) + h$$

נבצר מאיתנו להשתמש במשפט המאסטר במקרה זה. לא נאמר נואש - קיימת טכניקה של החלפת משתנים שתעזור לנו לפתור גם את כלל הנסיגה הזה. נגדיר $m := 2^h$, $T(2^h) := S(h)$. אז מתקיים $T(m) = T(2^h)$ וגם $h = \log_2(m)$.
בסימון החדש אנו מקבלים:

$$T(m) \leq 2T\left(\frac{m}{2}\right) + \log_2(m)$$

ממשפט המאסטר מקבלים: $T(m) = \Theta(m)$. מאחר שמתקיים $2^h \leq n \leq 2^{h+1} - 1$ אנו מקבלים:

$$m \leq n \leq 2m - 1$$

ולכן $\Theta(m) = \Theta(n)$.

4.2 עצי חיפוש בינאריים - BST

בעיה 4.1. נתונה קבוצה S בת n איברים. בהינתן x איבר כלשהו, נרצה לבצע את הפעולות הבאות ביעילות (זמן ריצה לוגריתמי):

1. $\text{Member}(S,x)$ - נרצה לבדוק האם $x \in S$ או $x \notin S$.
2. $\text{Insert}(S,x), \text{Delete}(S,x)$ - הוספה או הורדה של x מ-S.

3. $\text{Successor}\left(S,x\right)$ - (כאשר מוגדר יחס סדר על איברי S) בהינתן $x\in S$ נרצה למצוא את האיבר העוקב ל-x.

4. (כאשר מוגדר יחס סדר על איברי S) הדפסת איברי S בסדר עולה.

הערה 4.2. ניתן לפתור את הבעיה ע"י שימוש במערך לא ממויין - אולם זמן הריצה של הפעולות שתיארנו הוא $\Theta\left(n\right)$.

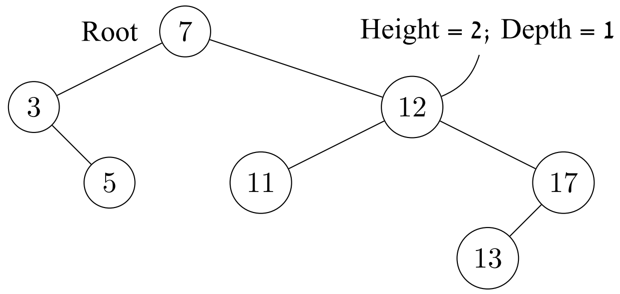
הערה 4.3. אם מוותרים על יחס הסדר ניתן לפתור את הבעיה "בזמן ממוצע" של $O\left(1\right)$ - עוד על כך בהמשך הקורס.

הערה 4.4. עצי חיפוש בינאריים פותרים את הבעיה, אך כפי שנראה מיד, לא בצורה יעילה מספיק.

הגדרה 4.5 (עץ חיפוש בינארי). **עץ חיפוש בינארי** הוא עץ בינארי שמקיים את **תכונת עץ החיפוש הבינארי**:

> לכל קודקוד v בעץ בחיפוש בינארי: האיבר ב-v גדול מכל האיברים בתת-העץ השמאלי שלו וקטן מכל האיברים בתת-העץ הימני שלו.

ניתן לייצג עץ בינארי באמצעות מבנה נתונים מקושר. זאת אומרת שלכל קודקוד יש מצביע ("פוינטר") לבן השמאלי, לבן הימני ולקודקוד האב.



איור 4.1: עץ חיפוש בינארי לדוגמה

פעולת החיפוש - Member

נתון עץ חיפוש בינארי T וערך כלשהו x. אם x קיים ב-T נחזיר מצביע לצומת שבו x מאוחסן, אחרת נחזיר מצביע ריק.
האלגוריתם:

> נסמן ב-a את האיבר שנמצא בשורש.
>
> 1. אם $x=a$ מצאנו את מבוקשנו, נחזיר מצביע למקום שבו האיבר נמצא.
> 2. אם $x>a$ נמשיך ונחפש ברקורסיה את x בתת-העץ הימני. אם x לא קיים בתת-העץ הימני אז x לא קיים ב-T.
> 3. אם $x<a$ נמשיך ונחפש ברקורסיה את x בתת-העץ השמאלי. אם x לא קיים בתת-העץ השמאלי אז x לא קיים ב-T.

אם T הוא בגובה h (ראה הגדרה 3.8), אז מספר הפעולות המקסימלי המבוצעת ע"י שגרת $\text{Member}\left(T,x\right)$ הוא $\Theta\left(h\right)$.

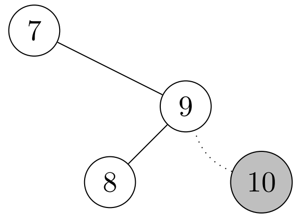
תרגיל. הוכיחו באינדוקציה על h את נכונות האלגוריתם.

פעולת הכנסה - Insert

נתון עץ חיפוש בינארי T וערך כלשהו x. אנו מעוניינים להוסיף את x ל-T.
האלגוריתם:

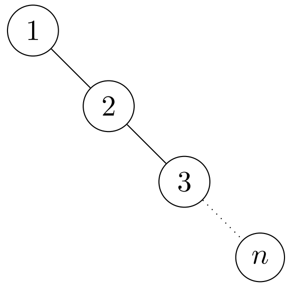
> מבצעים פעולת $\text{Member}\left(T,x\right)$. אם x נמצא בעץ, סיימנו.
>
> אחרת, החיפוש מסתיים בקודקוד שאין לו בן בכיוון של x, נוסיף את x בתור בן של הקודקוד הזה.

למעשה, הקודקוד שבו הסתיימה פעולת החיפוש נמצא במיקום שבו x היה אמור להיות, אך איננו - לכן נוסיף את x שם.



איור 4.2: מעוניינים להוסיף את הערך 10 לעץ הבינארי. החיפוש מסתיים בקודקוד 9, נוסיף את 10 בתור בן ימני כי הוא גדול מ-9.

מספר הפעולות של Insert הוא $O\left(h\right)$ כאשר h הוא גובה העץ.



איור 4.3: אם נכניס את $1,2,3,\ldots,n$ לפי הסדר באמצעות פעולת Insert נתקבל עץ חיפוש בינארי לא מאוזן, שהוא בפועל יותר מזכיר עבודה עם רשימה מקושרת מאשר עם עץ חיפוש בינארי מבחינת ביצועים. בהמשך נראה כי זו המוטיבציה מאחורי עצי AVL.

מינימום ומקסימום

יהי T עץ חיפוש בינארי.

1. $\text{Max}\left(T\right)$ הוא האיבר הגדול ביותר בעץ.
2. $\text{Min}\left(T\right)$ הוא האיבר הקטן ביותר בעץ.

מציאת מקסימום כדי למצוא את האיבר הגדול ביותר בעץ חיפוש בינארי T, נעקוב אחר המצביעים הימניים עד שניתקל בקודקוד בלי בן ימני.

מציאת מינימום כדי למצוא את האיבר הקטן ביותר בעץ חיפוש בינארי T, נעקוב אחר המצביעים השמאליים עד שניתקל בקודקוד בלי בן שמאלי.

מחיר הפעולות של מציאת מינימום ומקסימום הוא $O\left(h\right)$ כאשר h הינו גובה העץ T.

מציאת האיבר העוקב - $\textbf{Successor}\left(T,x\right)$

בהינתן צומת x בעץ חיפוש בינארי T, נרצה למצוא את העוקב לצומת.
האלגוריתם:

אם ל-x יש בן ימני, נחזיר את האיבר המינימלי בתת-העץ הימני ע"י קריאה לשגרה $\text{Min}\left(\text{Right}\left(x\right)\right)$.

אם ל-x אין בן ימני, האיבר העוקב של x הוא האב הקדמון הנמוך ביותר של x אשר הבן השמאלי שלו גם הוא אב קדמון של x. בניסוח מרושל אך נוח: עולים מהקודקוד לכיוון השורש וממשיכים עד שהעלייה היא לכיוון ימין וזה האיבר העוקב ל-x (אם לא קיימת פניה ימינה אז x הוא המקסימום ואין לו עוקב).

מחיר פעולת ה-Successor היא $O(h)$ כאשר h הנו גובה העץ כיוון שאנו מחפשים את העוקב במורד העץ או במעלה העץ.

דוגמה. נחזור ונתבונן באיור 4.1. האיבר העוקב של 12 הוא האיבר המינימלי בתת-העץ הימני - 13. האיבר העוקב של 5 הוא 7 (עולים שמאלה במעלה העץ עד שפנינו ימינה). לאיבר 17 אין איבר עוקב.

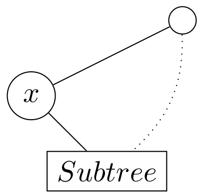
מחיקה - Delete

בהינתן ערך x בעץ חיפוש בינארי T, נרצה למחוק את הצומת שבו x מאוחסן.
האלגוריתם:

מחפשים את x - אם x לא שם, סיימנו. אחרת, נטפל בשלושה מקרים:

1. x עלה - נסיר אותו.
2. ל-x יש רק בן אחד - יוצרים קשר חדש בין $\text{Parent}(x)$ לבין הבן של x.
3. ל-x יש שני בנים - נמצא בתת-העץ הימני את האיבר העוקב ל-x. זה האיבר המינימלי בתת-העץ הימני ולכן אין לו בן שמאלי (בהכרח, אם היה לו בן שמאלי הוא לא היה העוקב, אלא בנו). נסיר את האיבר העוקב כמתואר במקרה 1 או 2, ונשים אותו במקום x.

זמן הריצה של השגרה Delete על עץ חיפוש בינארי בגובה h הוא $O(h)$ - פעולת המחיקה עצמה כרוכה בשינוי מצביעים - מספר קבוע של פעולות.



איור 4.4: המחשה למקרה השני בשגרה Delete - ל-x יש בן יחיד. במקרה זה נסיר את הקשר בין x לקודקוד האב ולתת-העץ שמחובר אליו, ובמקומו ניצור קשר חדש בין קודקוד האב לתת-העץ.

הדפסת האיברים בסדר עולה

בהינתן עץ חיפוש בינארי T נרצה להדפיס את כל האיברים בעץ בסדר עולה.
האלגוריתם:

1. נדפיס את איברי תת-העץ השמאלי של T ברקורסיה, אם הוא קיים.
2. נדפיס את הערך שמאוחסן בשורש.
3. נדפיס את איברי תת-העץ הימני של T ברקורסיה, אם הוא קיים.

הדפסת האיברים של עץ חיפוש בינארי בעל n איברים מתבצעת בזמן $\Theta(n)$. על מנת להשתכנע בכך, נספור כמה פעמים אנו מבקרים בכל צלע בעץ. מהרקורסיה נובע שאנו מבקרים בכל צלע פעמיים - פעם אחת במורד העץ ופעם נוספת במעלה העץ. מספר הצלעות בעץ בינארי על n קודקודים הוא $\Theta(n)$ ולכן זה זמן הריצה של השגרה.

4.3 עצי AVL

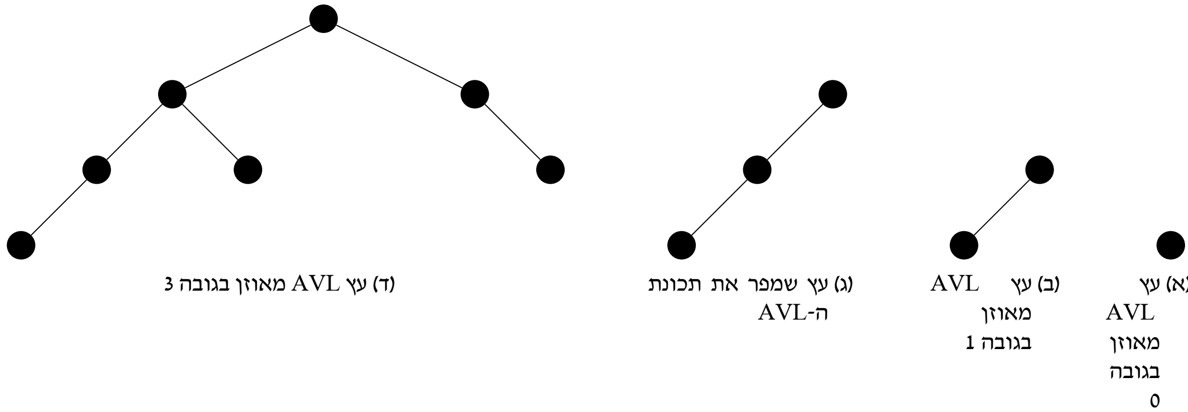
ראינו כי בעץ חיפוש בינארי הפעולות המרכזיות הן בזמן ריצה $\Theta(h)$ כאשר h הוא גובה העץ. אם העץ גבוה, ייתכן כי הביצועים לא יהיו יותר טובים מאשר שימוש במבנה נתונים כמו רשימה מקושרת. עתה, נראה מבנה נתונים מסוג עץ חיפוש בינארי שהוא "מאוזן". זה מבטיח כי הפעולות הבסיסיות יתבצעו בזמן לוגריתמי.

הגדרה 4.6 (עץ AVL)**.** עץ AVL הוא מבנה נתונים מסוג עץ חיפוש בינארי עם אותם פעולות, כך שבכל קודקוד מתקיימת תכונת ה-AVL:

אם h_L - גובה תת-העץ השמאלי של הקודקוד ; h_R - גובה תת-העץ הימני של הקודקוד, אז :

$$|h_L - h_R| \leq 1$$

הערה 4.7. נגדיר את גובה תת-עץ ריק להיות -1.



איור 4.5 : דוגמאות לעצי AVL

בשיעור הבא נוכיח שהגובה h של עץ AVL על n קודקודים מקיים $h = \Theta\left(\log\left(n\right)\right)$.

5 שבוע 5 - 23.04.18 - פרופ׳ מיכאל בן-אור

5.1 המשך טיפול בעצי AVL

טענה 5.1. אם T עץ AVL על n קודקודים, אז הגובה של T, h, מקיים $h = \Theta\left(\log\left(n\right)\right)$.

הוכחה. נסמן h גובה העץ, n מספר הקודקודים; אזי אנו יודעים כי מתקיים לכל עץ בינארי בגובה h:

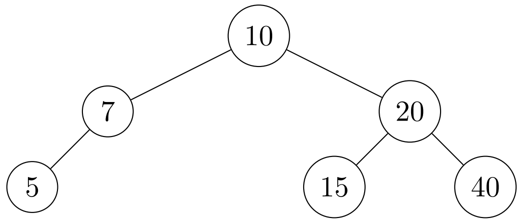
$$2^{h+1} - 1 \geq n$$

נרצה להוכיח כי עבור עצי AVL מתקיים $n \geq \left(\frac{3}{2}\right)^h$ (למעשה ניתן להוכיח חסם תחתון יותר ״חזק״, אבל זה מספיק טוב).
אם נוכיח את זה, אז ע״י הוצאת log מאגפי האי שוויון נקבל $h + 1 > \log_2\left(n\right) \geq h \log_2\left(\frac{3}{2}\right)$ ומכאן נובע $h = \Theta\left(\log\left(n\right)\right)$.
נוכיח את החסם התחתון באינדוקציה h.
נסמן ב-n_h את מספר הקודקודים המינימליים בעץ AVL בגובה h. נרצה להוכיח $n_h \geq \left(\frac{3}{2}\right)^h$.
בסיס: $h = 0 \Leftarrow n_0 = 1 \Leftarrow 1 = \left(\frac{3}{2}\right)^0 \geq 1$; $h = 1 \Leftarrow n_0 = 2 \Leftarrow 2 \geq \left(\frac{3}{2}\right)^1$.
צעד: נשים לב כי $n_h > n_{h-1}$ - כדי להשתכנע, נתבונן בעץ AVL בגובה h. לשורש יש לפחות בן אחד שהוא בגובה $h-1$, ולכן המספר המינימלי של קודקודים בגובה h גדל. בסה״כ קיבלנו סדרה מונוטונית עולה ממש (n_h).
כדי לקבל את עץ AVL בגובה h עם מספר קודקודים מינימלי, נתלה בבן השמאלי עץ AVL מינימלי בגובה $h-1$, וכדי לא להפר את תכונת ה-AVL, נתלה בבן הימני עץ AVL מינימלי בגובה $h-2$.
סה״כ אנו מקבלים את הרקורסיה הבאה: $n_h = n_{h-1} + n_{h-2} + 1$.
נניח שהטענה נכונה לכל גובה קטן מ-h ונוכיח ל-h.

$$\begin{aligned}
n_h &= n_{h-1} + n_{h-2} + 1 \\
&\geq \left(\frac{3}{2}\right)^{h-1} + \left(\frac{3}{2}\right)^{h-2} + 1 \\
&> \left(\frac{3}{2}\right)^{h-1} + \left(\frac{3}{2}\right)^{h-2} \\
&= \left(\frac{3}{2}\right)^{h-2}\left[\frac{3}{2} + 1\right] \\
&= \left(\frac{3}{2}\right)^{h-2}\left(\frac{10}{4}\right) \\
&> \left(\frac{3}{2}\right)^{h-2}\left(\frac{9}{4}\right) \\
&= \left(\frac{3}{2}\right)^{h-2}\left(\frac{3}{2}\right)^2 \\
&= \left(\frac{3}{2}\right)^h
\end{aligned}$$

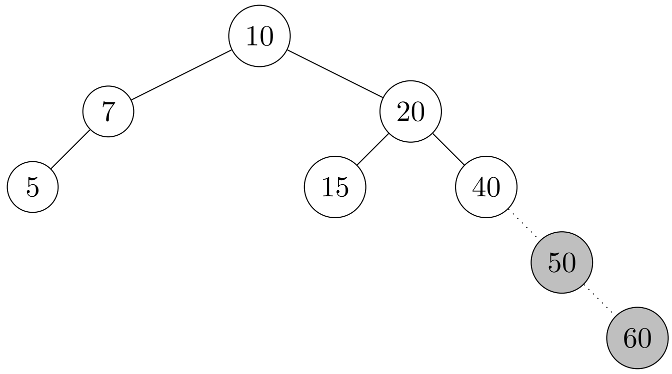
בזאת הסתיים צעד האינדוקציה. □

הוספת איבר לעץ AVL הוספת איבר לעץ AVL דומה להוספת איבר ב-BST, רק שבחלק מהמקרים ניאלץ לבצע תיקון, כי לאחר הוספת האיבר החדש ייתכן כי הפרנו את תכונת ה-AVL.
נתבונן בעץ הבא:



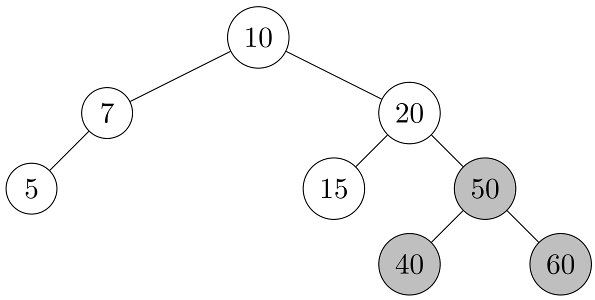
איור 5.1 : דוגמה לעץ AVL

נניח כי אנחנו מעוניינים להוסיף את האיבר 50 ולאחריו 60. אז העץ ייראה כך :



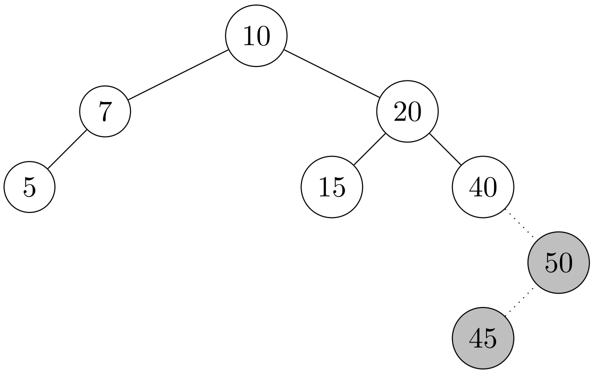
איור 5.2 : עץ AVL מאיור קודם, לאחר שהוספנו לו את הקודקודים 50 ו-60 (הפרת RR)

תכונת ה-AVL מופרת בקודקוד 40. נוכל לרשום את תת-העץ המושרש בקודקוד 40 גם כך :



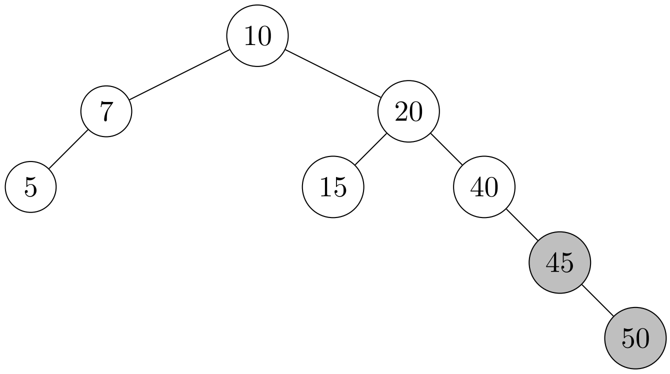
איור 5.3 : שינוי צורת העץ כך שיהיה מאוזן (תיקון הפרת RR)

נחזור ונתבונן באיור המקורי, אם נוסיף 50 ואז 45 אז נקבל עץ שייראה כך :



איור 5.4: הוספת 50 ו-45 לעץ AVL המקורי (הפרת RL)

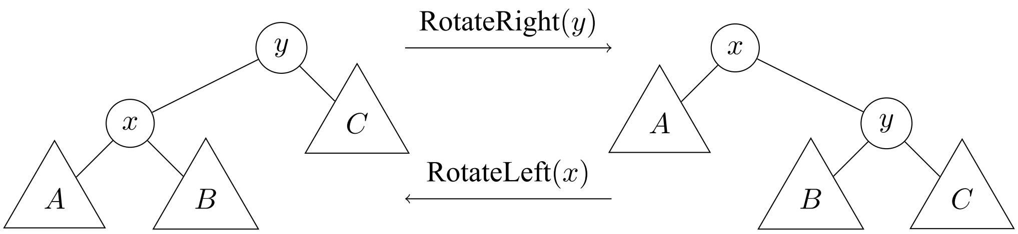
גם כאן יש הפרה בתכונת ה-AVL בקודקוד 40. הפעם, כדי לתקן נבצע סיבוב כך שיתקיים:



איור 5.5: תיקון חלקי לעץ AVL

לאחר מכן, נתקן כפי שעשינו במקרה הקודם.

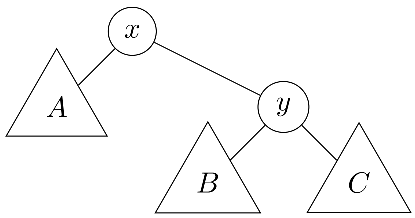
פעולות סיבוב נגדיר פעולות סיבוב ("רוטציות") על עצי חיפוש בינאריים. פעולות אלה מסייעות בתיקון העץ בפעולות כמו Insert ו-Delete והן שומרות על תכונת עץ החיפוש הבינארי.



איור 5.6: רוטציה שמאלית ורוטציה ימנית בעץ חיפוש בינארי

פעולות הסיבוב מותרות על עץ חיפוש בינארי.
כדי לטפל בעץ AVL נשמור בכל קודקוד את מצב האיזון שלו.

סוגי הפרות נתבונן באיור:



איור 5.7 : עץ AVL כללי במצב "מאוזן", וקודקוד x מקיים שגורם האיזון שלו הוא -1 (כלומר, גובה תת-העץ הימני גדול מגובה תת-העץ השמאלי).

- הפרת RR - אם הוספנו קודקוד ב-C כך שלאחר ההוספה נוצרה הפרה בקודקוד x ; אז נקרא להפרה זו RR.
- הפרת RL - אם הוספנו קודקוד ב-B כך שלאחר ההוספה נוצרה הפרה בקודקוד x ; אז נקרא להפרה זו RL.

הערה 5.2. הפרות LL ו-LR מוגדרות באופן דומה.

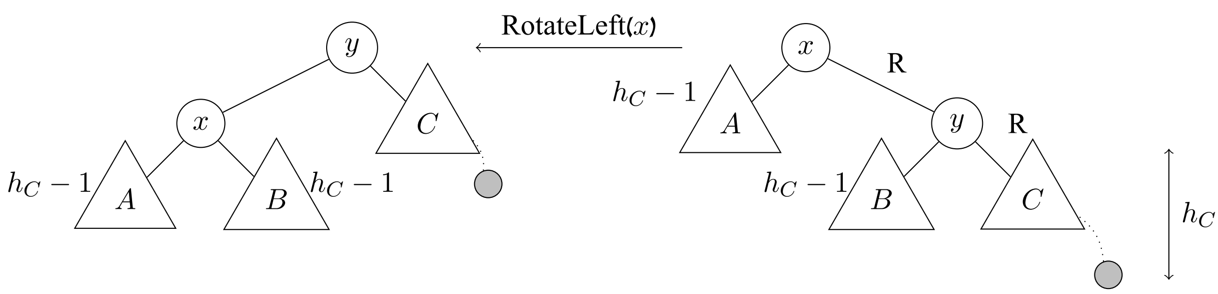
5.2 תיקון תכונת ה-AVL לאחר פעולת Insert

לאחר שהוספנו את הקודקוד הנוסף, עולים חזרה לכיוון השורש, ומעדכנים את גורמי האיזון. נתקדם עד לקודקוד הראשון שבו נוצרה הפרה. במקרה זה, מבצעים תיקון בהתאם לסוג ההפרה וממשיכים.

תיקון הפרת RR התיקון מתבצע ע"י ביצוע רוטציה שמאלה של x כמתואר באיור **??**.
נניח כי הוספנו קודקוד ל-C ובקודקוד x נוצרה הפרה ראשונה. נסמן ב-h_C את גובה תת-העץ של C לאחר ההוספה.
אזי מתקיים : $h_B = h_C - 1$. מדוע? אם $h_B = h_C - 2$ אז ההפרה הראשונה הייתה בקודקוד y.
אם $h_B = h_C$ או $h_B = h_C + 1$ אז לא הייתה הפרה בקודקוד x. בנוסף, מתקיים $h_B = h_C - 1$.
לאחר ביצוע רוטציה שמאלה, אנו מקבלים ש-B הוא בן ימני של x. מאחר ש-$h_A = h_B$ אנו מקבלים $h_x = h_C$.
הבן הימני של y הוא C ולכן הפרש הגבהים בין הבן השמאלי של y (שהוא הקודקוד x) לבין הבן הימני של y :

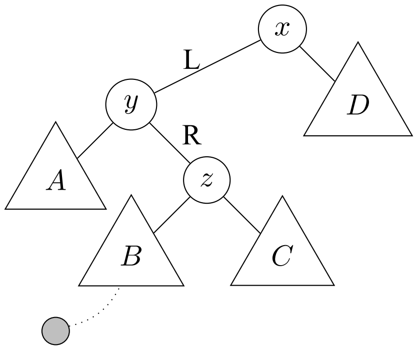
$$h_x - h_C = h_C - h_C = 0$$

ולכן העץ מאוזן לאחר ביצוע רוטציה שמאלה, והתיקון הסתיים. להלן המחשה :



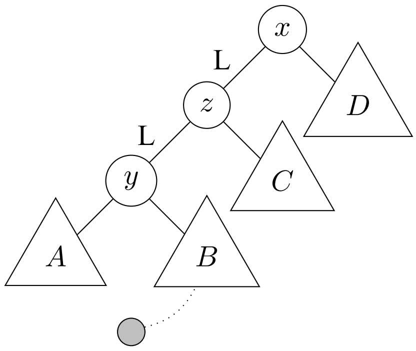
איור 5.8 : המחשה לתיקון הפרת RR

תיקון הפרת LR נתבונן בעץ הבא - הוספנו ב-B קודקוד שגרם להפרה ראשונה בקודקוד x :



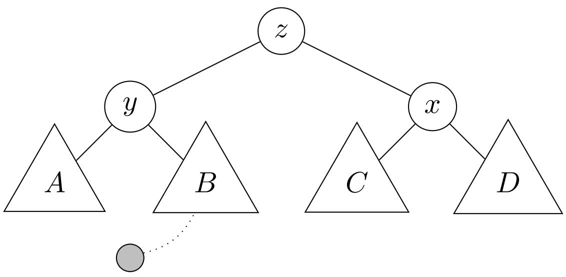
איור 5.9 : דוגמה להפרת LR

התיקון מתבצע ע"י RotateLeft (y) ואז RotateRight (x).
לאחר RotateLeft (y) העץ ייראה כך :



איור 5.10 : העץ לאחר RotateLeft (y) - כעת יש הפרת LL

לאחר RotateRight (x) העץ ייראה כך :



איור 5.11 : העץ לאחר RotateRight (x) - כעת העץ מאוזן

נסמן ב-h_B את הגובה של B לאחר ההוספה.
במצב ההתחלתי (הפרת LR) מתקיים : $h_z = h_B + 1$; $h_y = h_B + 2$; $h_x = h_B + 3$; $h_D = h_B$; $h_C = h_B - 1$; $h_A = h_B$.
לאחר ביצוע התיקון מתקיים : $h_y = h_B + 1$, $h_x = h_B + 1$, $h_z = h_B + 2$ ולכן העץ במצב מאוזן.

5.3 פעולת ה-Delete בעץ AVL

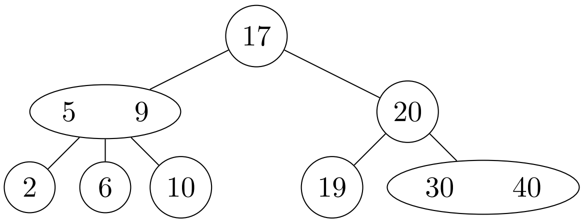
אחרי הורדת קודקוד (פעולה זו מבוצעת בדומה לפעולת ה-Delete ב-BST), עולים חזרה בעץ ואם מגלים הפרה, בודקים בצד "הכבד" (הגבוה יותר) באיזה סוג הפרה מדובר, ומתקנים בהתאם. אם לאחר התיקון הגובה יורד צריך להמשיך ולבדוק הלאה במעלה העץ.

ניתן להביא דוגמאות שבהן נדרש לבצע $O\left(\log\left(n\right)\right)$ סיבובים.

5.4 עצי $2-3$

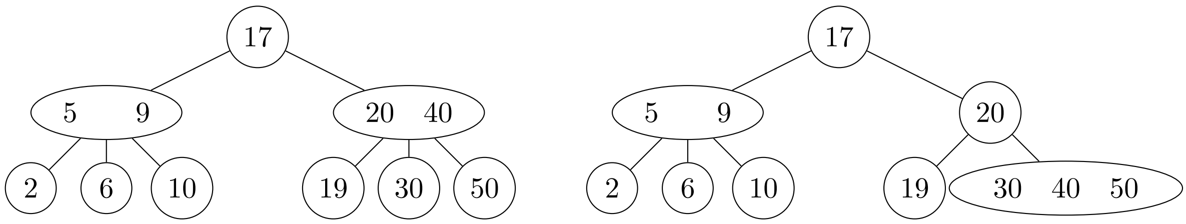
נעבור בקצרה על פתרון נוסף לעצי חיפוש מאוזנים.

הגדרה 5.3. בעץ $2-3$ הוא עץ חיפוש מאוזן בו לכל קודקוד יש 0 או 2 או 3 בנים. בנוסף, כל העלים נמצאים באותו עומק (מרחק מן השורש). בכל קודקוד נמצא איבר אחד או שניים.
אם בצומת מסוים יש 2 ערכים (נקרא גם "צומת-3") - נסמנם x,y - ונניח כי $x<y$. האיברים בתת-העץ השמאלי של קודקוד זה הם קטנים מ-x. האיברים בתת-העץ האמצעי הם בין x ל-y. האיברים בתת-העץ הימני הם גדולים מ-y.



איור 5.12: דוגמה לעץ $2-3$

בעץ $2-3$ על n קודקודים, הגובה h מקיים $h=\Theta\left(\log\left(n\right)\right)$.
דוגמה להכנסת איבר בעץ $2-3$



(א) שלב ראשון בהוספת הערך 50   (ב) שלב שני בהוספת הערך 50

איור 5.13: הוספת 50 לעץ $2-3$ מהאיור הקודם

6 שבוע 6 - 30.04.18 - פרופ׳ מיכאל בן-אור

6.1 הגדרות בסיסיות בהסתברות

הגדרה 6.1 (מרחב הסתברות)**.** מרחב הסתברות הוא זוג (Ω, P) כאשר Ω היא מרחב המדגם (קבוצת המאורעות) ו-P היא פונקציית ההסתברות $P : \Omega \to [0,1]$ שמקיימת $\sum\limits_{\omega\in\Omega} P(\omega) = 1$.

דוגמה 6.2. לעתים נטפל בקבוצה Ω שהיא קבוצת כל הסדרות על $0,1$ באורך n; $\Omega = \{0,1\}^n$ (יש 2^n סדרות כאלה). קבוצת המאורעות יכולה להיות גם כל המספרים הטבעיים - ז״א $\Omega = \{1,2,3,\dots\}$.

הערה 6.3. כאשר Ω קבוצה אינסופית אנו מקבלים כי הדרישה מפונקציית ההסתברות, $\sum\limits_{\omega\in\Omega} P(\omega) = 1$, היא בעצם טור אינסופי שצריך להתכנס ל-1. ניתן להגדיר את תורת ההסתברות מעל הישר הממשי, $\mathbb{R}$, אך זה מחוץ לתחום העיסוק של קורס זה; אנו נעסוק בתורת ההסתברות הבדידה.

הגדרה 6.4 (מאורע)**.** מאורע A הוא תת-קבוצה של מרחב המדגם Ω. נגדיר את ההסתברות של המאורע כסכום:

$$P(A) = \sum_{\omega\in A} P(\omega)$$

הגדרה 6.5 (מאורע משלים)**.** בהינתן מאורע $A \subseteq \Omega$, נגדיר את מאורע המשלים $\overline{A}$ בתור $\overline{A} = \Omega \setminus A$.

הערה 6.6. מההגדרות נובע כי $P(A) + P\left(\overline{A}\right) = 1$.

הגדרה 6.7 (התפלגות אחידה)**.** בהינתן מרחב מדגם סופי Ω, נאמר כי התפלגות ההסתברות אחידה על Ω אם״ם ההסתברות של כל מאורע אטומי $\omega \in \Omega$ היא $P(\omega) = \frac{1}{|\Omega|}$.

הגדרה 6.8 (מאורעות בלתי תלויים)**.** A ו-B מאורעות ב-Ω. נאמר כי A ו-B מאורעות בלתי תלויים אם״ם

$$P(A\cap B) = P(A)\cdot P(B)$$

באופן דומה נכליל את ההגדרה עבור n מאורעות. $A_1, A_2, \dots, A_n$ מאורעות בלתי תלויים אם״ם לכל $2 \le k \le n$ ולכל k אינדקסים $1 \le i_1 \le i_2 \le \dots \le i_k \le n$ מתקיים:

$$P(A_{i_1}\cap A_{i_2}\cap\dots\cap A_{i_k}) = P(A_{i_1})\cdot P(A_{i_2})\cdots P(A_{i_k})$$

הגדרה 6.9 (מאורעות k-בלתי תלויים)**.** נאמר שהמאורעות $A_1,\dots,A_n$ הם k-בלתי תלויים אם כל קבוצה של k מאורעות מתוכם הם בלתי תלויים.

הערה 6.10. למאורעות שהם 2-בלתי תלויים קוראים גם ״בלתי תלויים בזוגות״.

דוגמה 6.11. $\Omega = \{000, 110, 101, 011\}$ - אוסף כל הסדרות על $0,1$ באורך 3 שיש בהם מספר זוגי של 1. נגדיר מאורעות $A_i = \{(\omega_1,\omega_2,\omega_3)\in\Omega | \omega_i = 0\}$ לכל $i = 1,2,3$. התפלגות ההסתברות אחידה על Ω. נקבל כי

$$P(A_i) = \frac{1}{2} \qquad i = 1,2,3$$

ההסתברות לקבל 0 בקואורדינטה הראשונה ו-0 בקואורדינטה השניה היא:

$$P(A_1\cap A_2) = P(A_1)\cdot P(A_2) = \frac{1}{2}\cdot\frac{1}{2} = \frac{1}{4}$$

באופן כללי לכל $i \ne j$ נקבל $P(A_i\cap A_j) = \frac{1}{4}$. לכן המאורעות בלתי תלויים בזוגות. אולם המאורעות A_1, A_2, A_3 אינם בלתי תלויים בשלשות, שכן מתקיים $P(A_1\cap A_2\cap A_3) = \frac{1}{4} \ne \frac{1}{2}\cdot\frac{1}{2}\cdot\frac{1}{2}$.
אם מגדירים את $\Omega = \{0,1\}^3$ ו-A_i מוגדר כמו קודם אז מקבלים כי המאורעות A_1, A_2, A_3 בלתי תלויים.

6.2 הסתברות מותנית

הגדרה 6.12 (הסתברות מותנית). נתון מרחב הסתברות (Ω, P) ומאורע A כך ש-$P(A) > 0$. נגדיר את ההסתברות של מאורע כלשהו $B \subseteq \Omega$ תחת ההנחה ש-A קרה בתור:

$$P(B|A) \coloneqq \frac{P(A \cap B)}{P(A)}$$

הערה 6.13. אם A ו-B מאורעות בלתי תלויים אז

$$P(B|A) = \frac{P(A \cap B)}{P(A)} \overbrace{=}^{\text{אי תלות}} \frac{P(A) \cdot P(B)}{P(A)} = P(B)$$

דוגמה 6.14. דוגמה נוספת למרחב הסתברות (התפלגות גיאומטרית). מטילים מטבע מוטה כך שהתוצאה 1 בהסתברות p ו-0 בהסתברות $1-p$. הטלות מטבע שונות בלתי תלויות זו בזו, וממשיכים עד שמקבלים 1. ההסתברות לעצור אחרי k פעמים היא $(1-p)^{k-1} \cdot p$. נוודא כי פונקציית ההסתברות מקיימת את הדרישה שבהגדרה:

$$\begin{aligned}\sum_{k=1}^{\infty} p(1-p)^{k-1} &= p\sum_{k=0}^{\infty}(1-p)^k \\ &= p \cdot \frac{1}{1-(1-p)} \\ &= 1\end{aligned}$$

סדרת ההסתברויות היא סדרה גיאומטרית ומכאן שמה של ההתפלגות.

6.3 משתנים מקריים

הגדרה 6.15 (משתנה מקרי). משתנה מקרי הוא פונקציה $X : \Omega \to \mathbb{R}$.

הגדרה 6.16 (תוחלת). התוחלת של משתנה מקרי על מרחב הסתברות (Ω, P) ומשתנה מקרי X מוגדרת באמצעות:

$$\mathbb{E}[X] \coloneqq \sum_{\omega \in \Omega} P(\omega) X(\omega)$$

הערה 6.17. אם Ω סופי ו-P התפלגות אחידה אזי $\mathbb{E}[x] = \frac{\sum_{\omega\in\Omega} X(\omega)}{|\Omega|}$.
באופן כללי, התוחלת היא ממוצע משוקלל של כל הערכים האפשריים, כשכל ערך משוקלל בהסתברות שלו.

הערה 6.18. לכל $a \in \mathbb{R}$ ומשתנה מקרי X נתבונן במאורע "$X = a$" המוגדר ע"י $\{\omega \in \Omega | X(\omega) = a\}$. מתקיים בבירור (שינוי סדר סכימה):

$$\mathbb{E}[X] = \sum_{a \in \mathbb{R}} a \cdot P(X = a)$$

אמנם הסכימה היא על כל ה-a הממשיים, אולם ברוב המקרים ההסתברות $P(X = a)$ תהיה שווה 0, ולכן נסכום על כל ה-a האפשריים.

הגדרה 6.19 (חיבור משתנים מקריים). יהיו $X : \Omega \to \mathbb{R}$ ו-$Y : \Omega \to \mathbb{R}$ שני משתנים מקריים. נגדיר $Z : \Omega \to \mathbb{R}$, $Z = X + Y$ ע"י $Z(\omega) = X(\omega) + Y(\omega)$.

טענה 6.20 (לינאריות התוחלת). יהיו $X, Y : \Omega \to \mathbb{R}$ משתנים מקריים. אם $\mathbb{E}[X]$, $\mathbb{E}[Y]$ קיימים ומוגדרים היטב אז

$$\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$$

הוכחה. לפי הגדרת התוחלת וחיבור משתנים מקריים מתקיים:

$$\begin{aligned}\mathbb{E}[X+Y] &= \sum_{\omega\in\Omega}(X+Y)(\omega)P(\omega)\\ &= \sum_{\omega\in\Omega}(X(\omega)+Y(\omega))P(\omega)\\ &= \sum_{\omega\in\Omega}X(\omega)P(\omega)+\sum_{\omega\in\Omega}Y(\omega)P(\omega)\\ &= \mathbb{E}[X]+\mathbb{E}[Y]\end{aligned}$$

כנדרש. □

דוגמה 6.21. ניקח $\Omega=\{0,1\}^n$ עם התפלגות אחידה. נגדיר משתנה מקרי $X_k:\Omega\to\mathbb{R}$ באופן הבא:

$$X_k((a_1,\ldots,a_n))=a_k$$

נגדיר $Y:\Omega\to\mathbb{R}$ משתנה מקרי להיות מספר ה-1-ים בסדרה $\omega\in\Omega$; פורמלית:

$$Y((a_1,\ldots,a_n))=\sum_{k=1}^{n}a_k$$

נחשב את התוחלת של X_k:

$$\mathbb{E}[X_k]=\frac{1}{2}$$

כי מחצית מן הסדרות על $0,1$ באורך n מכילות 1 בקואורדינטה ה-k, ומחצית מכילות 0 בקואורדינטה ה-k.
נחשב את התוחלת של Y:

$$\mathbb{E}[Y]=\frac{1}{2^n}\sum_{k=0}^{n}k\binom{n}{k}$$

ההתפלגות אחידה על Ω, ומאחר שיש 2^n סדרות ב-Ω אנו מקבלים כי ההסתברות לקבל סדרה אחת היא $\frac{1}{2^n}$. את זה נכפול בסכום של מספר הסדרות שיש להם k אחדים $\binom{n}{k}$ כפול המספר k שהוא למעשה הערך של המשתנה המקרי Y עבור סדרה שיש בה k אחדים.
נבחין כי מתקיים:

$$Y=X_1+X_2+\cdots+X_n$$

ומלינאריות התוחלת מקבלים:

$$\begin{aligned}\mathbb{E}[Y] &= \mathbb{E}[X_1+X_2+\cdots+X_n]\\ &= \sum_{k=1}^{n}\mathbb{E}[X_k]\\ &= \sum_{k=1}^{n}\frac{1}{2}\\ &= \frac{n}{2}\end{aligned}$$

נשים לב שבעזרת מושג התוחלת הצלחנו לקבל את השוויון $\frac{1}{2^n}\sum_{k=0}^{n}k\binom{n}{k}=\frac{n}{2}$ שהיינו עלולים לעבוד קשה עבורו אם היינו רוצים להוכיח אותו באמצעים אחרים.

6.4 ניתוח אלגוריתם הסתברותי - מיון מהיר

מבוא: במהלך קורס מבני נתונים אנו נעסוק בניתוח זמן הריצה של אלגוריתמים הנעזרים במהלך הריצה בבחירות מקריות. אלגוריתם הסתברותי כזה מגדיר מרחב מדגם של כל הריצות האפשריות של האלגוריתם. בהינתן קלט נקבל התפלגות על הריצות האפשריות של האלגוריתם. זמן הריצה של אלגוריתם הסתברותי הוא משתנה מקרי, ויעילות האלגוריתם נמדדת ע״פ תוחלת זמן הריצה שלו.

תיאור האלגוריתם: נתון מערך לא ממויין בגודל n (נניח שכל המספרים שונים). אם $n=1$, סיימנו.
בוחרים איבר מקרי שנקרא Pivot. משווים את ה-Pivot לכל האיברים. את הקטנים שמים לפניו ואת הגדולים מאחוריו.
ממשיכים ברקורסיה על תת-המערך שערכיו קטנים מה-Pivot ועל תת-המערך שערכיו גדולים מה-Pivot.
למשל, עבור מערך $A = [7,9,2,1,13,15,3,19]$ ובחירה מקרית של Pivot שווה ל-13 נקבל אחרי ריצה אחת של האלגוריתם את המערך $A' = [7,9,2,1,3,\underline{13},15,19]$, ונמשיך את הריצה על שני תתי-המערך $[15,19]$ ו-$[7,9,2,1,3]$.

תוחלת מספר ההשוואות ב-QuickSort: יהי Y המשתנה המקרי הנותן את מספר ההשוואות שהאלגוריתם מבצע עבור סדרת בחירות מקריות מסוימת. רוצים לחשב את $\mathbb{E}[Y]$. בהינתן קלט בגודל n של מספרים שונים, נסמן את הסידור של האיברים במערך בסדר עולה $z_1 < z_2 < \cdots < z_n$.
נגדיר משתנים מקריים לכל $1 \le i < j \le n$ באופן הבא:

$$X_{ij}(\omega) = \begin{cases} 1 & \text{The algorithm compares } z_i \text{ and } z_j \\ 0 & \text{אחרת} \end{cases}$$

לפי הגדרת האלגוריתם מתקיים כי אם z_i ו-z_j הושוו פעם אחת, הם לא יושוו פעם נוספת.
על כן, מתקבל מיד השוויון הבא (שוויון בין פונקציות על הריצות האפשרויות):

$$Y = \sum_{1 \le i < j \le n} X_{ij}$$

נתעניין בגודל $\mathbb{E}[X_{ij}]$. נתבונן בתת-המערך שמתחיל ב-z_i ומסתיים ב-z_j: $[z_i, z_{i+1}, \ldots, z_j]$; כל בחירה של איבר Pivot שנבחר מחוץ לתת-המערך הזה לא קובע השוואה בין z_i ל-z_j. בנוסף, כל בחירה של איבר Pivot בתוך תת-המערך הזה (אבל לא בקצוות) קובע כי z_i לא יושווה עם z_j. בחירה של איבר Pivot של קצות תת-המערך גוררת השוואה בין z_i ל-z_j. לכן התוחלת היא:

$$\mathbb{E}[X_{ij}] = \frac{2}{j-i+1}$$

מכאן נקבל:

$$\mathbb{E}[Y] = \mathbb{E}\left[\sum_{1 \le i < j \le n} X_{ij}\right] = \sum_{1 \le i < j \le n} \mathbb{E}[X_{ij}] = \sum_{1 \le i < j \le n} \frac{2}{j-i+1} = O(n \log(n))$$

(ההוכחות הפורמליות יבוצעו בהרצאה הבאה).

7 שבוע 7 - 13.05.18 - פרופ׳ מיכאל בן-אור

7.1 השלמות בהסתברות

בעיה 7.1. ברשותכם מטבע לא מוטה; $\Omega = \{0,1\}$ ו-$P(0) = P(1) = \frac{1}{2}$. אם נטיל שני מטבעות מרחב המדגם יהיה $\Omega_4 = \{0,1\}^2$ - כאשר כל איבר במרחב מתקבל בהסתברות $\frac{1}{4}$ (התפלגות אחידה). כיצד ניתן לדגום ב-$\Omega_3 =$ $\{00, 01, 10\}$ בהתפלגות אחידה, כלומר $P(00) = P(01) = P(10) = \frac{1}{3}$?

לא ניתן לקרב ע״י מספר הטלות מסוים ℓ, משום שעבור ℓ הטלות אנו מקבלים את המרחב $\Omega_{2^\ell} = \{0,1\}^\ell$ וההסתברות של כל מאורע במרחב היא מהצורה $\frac{k}{2^\ell}$ כאשר k מספר שלם; והרי $\frac{1}{3} \neq \frac{k}{2^\ell}$ לכל בחירה של k ו-ℓ.

פתרון. מטילים שני מטבעות ודוגמים באופן אחיד את Ω_4. אם התוצאה היא אחת מ-$\{00, 01, 10\}$ אז עוצרים, אחרת מנסים שוב.

באופן כללי יותר, ננסח את הבעיה כך: נתונה קבוצה Ω והתפלגות הסתברות אחידה על Ω (כלומר, $P(\omega) = \frac{1}{|\Omega|}$ לכל $\omega \in \Omega$). נתונה קבוצה חלקית $S \subseteq \Omega$ כאשר $P(S) = p$. דוגמים איבר x מתוך Ω; אם קרה המאורע $A =$ "$x \in S$" אז נעצור, אחרת, נחזור על הניסוי עד להצלחה.

- האם בסוף בחרנו איבר בהתפלגות אחידה מתוך S?
- מה תוחלת מספר הבחירות שצריך לבצע עד להצלחה?

לפי הסימונים שקבענו מתקיים: $p = \frac{|S|}{|\Omega|} \overbrace{=}^{\text{התפלגות אחידה}} P(A)$. יהי $s \in S$ ונתבונן במאורע $B =$ "$x = s$". מאחר שמדובר בהתפלגות אחידה:

$$P(B) = \frac{1}{|\Omega|}$$

אנו מעוניינים בהסתברות $P(B|A)$ - כלומר, מה ההסתברות שיצא $x = s$ בהינתן שהניסוי הצליח. ע״פ הגדרת הסתברות מותנית:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$

המאורע $B \subseteq A$ (כי $s \in S$) ולכן:

$$P(B|A) = \frac{P(B)}{P(A)} = \frac{\frac{1}{|\Omega|}}{\frac{|S|}{|\Omega|}} = \frac{1}{|S|}$$

לכן בסוף בחרנו איבר בהתפלגות אחידה מתוך S.
תוחלת מספר הבחירות ב-Ω היא $\frac{1}{p}$ מכיוון שמדובר במשתנה גיאומטרי.

7.2 המשך ניתוח תוחלת מספר ההשוואות במיון מהיר

נמשיך בניתוח תוחלת מספר ההשוואות מהנקודה שבה עצרנו שיעור שעבר. אנחנו רוצים לחשב את $\mathbb{E}[X_{ij}]$ עבור $1 \leq i < j \leq n$. מהי ההסתברות ש-z_i ו-z_j יושוו ישירות?
נתבונן בתת-המערך $[z_i, z_{i+1}, \ldots, z_j]$. בחירות של איבר ציר מחוץ לתת-המערך הזה לא קובעות אם z_i ו-z_j יושוו ביניהם (נזכור כי $z_1 < z_2 < \cdots < z_{n-1} < z_n$ - כלומר, זה המערך בסדר הממויין שלו. לכן בחירה של איבר ציר מחוץ לתת-המערך הזה "מעבירה" את כל תת-המערך הזה לצד מסוים, והאלגוריתם יפעל שוב רקורסיבית על תת-מערך, בפוטנציה גדול יותר, שיכיל את תת-המערך שלנו). רק בחירה של איבר ציר בתת-המערך $[z_i, \ldots, z_j]$ קובעת האם z_i ו-z_j יושוו ישירות.
מאחר שיש $j - i + 1$ איברים בתת-המערך הזה, ו-z_i יושווה עם z_j אם ורק אם z_i או z_j נבחרו כאיבר ציר (Pivot), נקבל כי ההסתברות p_{ij} ש-z_i ו-z_j יושוו ישירות הינה: $p_{ij} = \frac{2}{j-i+1}$. מכאן נובע:

$$\mathbb{E}[X_{ij}] = p_{ij} \cdot 1 + (1 - p_{ij}) \cdot 0 = \frac{2}{j-i+1}$$

בתור הערה, נשים לב כי $\mathbb{E}\left[X_{23}\right]=1$ משום שבכל אלגוריתם מיון מבוסס השוואות, איברים סמוכים חייבים להיות מושווים ישירות, אחרת יכולנו להחליף את הסדר ביניהם.
עתה נחשב את $\mathbb{E}\left[Y\right]$ תוחלת מספר ההשוואות (תזכורת $Y=\sum\limits_{1\leq i<j\leq n}X_{ij}$).

$$\mathbb{E}\left[Y\right]=\mathbb{E}\left[\sum_{1\leq i<j\leq n}X_{ij}\right]=\sum_{1\leq i<j\leq n}\mathbb{E}\left[X_{ij}\right]=\sum_{1\leq i<j\leq n}\frac{2}{j-i+1}$$

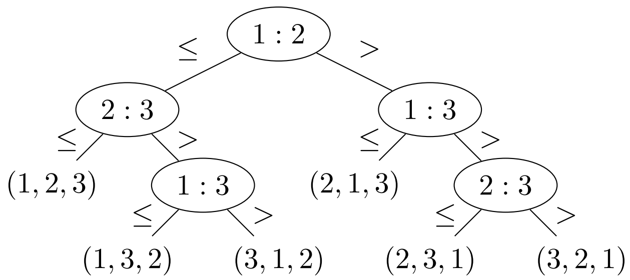
נפתח את הסכום:

$$\begin{aligned}
\sum_{1\leq i<j\leq n}\frac{2}{j-i+1} &= \sum_{\substack{j=1\\(i=1)}}^{n}\frac{2}{j}+\sum_{\substack{j=3\\(i=2)}}^{n}\frac{2}{j-1}+\cdots+\sum_{\substack{j=n\\(i=n-1)}}^{n}\frac{2}{j-n+2}\\
&= 2\cdot\left(\sum_{j=2}^{n}\frac{1}{j}+\sum_{j=2}^{n-1}\frac{1}{j}+\cdots+\sum_{j=2}^{2}\frac{1}{j}\right)\\
&\leq 2\cdot n\left(\sum_{j=2}^{n}\frac{1}{j}\right)\\
&\leq 2n\int_{1}^{n}\frac{1}{x}dx\\
&= 2n\left[\ln\left(x\right)\right]_{1}^{n}\\
&= 2n\ln\left(n\right)\\
&= 2n\ln\left(2\right)\log_{2}\left(n\right)\\
&\approx 1.38n\log_{2}\left(n\right)
\end{aligned}$$

נשים לב שחסמנו את הסכום $\sum\limits_{j=2}^{n}\frac{1}{j}$ באמצעות אינטגרציה.

7.3 חסמים תחתונים למיון

בהינתן אלגוריתם דטרמיניסטי (לא הסתברותי) למיון n מספרים בעזרת השוואות, נוכל לתאר את מהלך האלגוריתם באמצעות עץ ההשוואות שלו. במיון השוואות, כדי לקבל מידע על סדר האיברים משתמשים רק בהשוואות $<,>,\leq,\geq,=$.
נתון מערך לא ממויין $A=\left[a_1,\ldots,a_n\right]$ אשר יהווה את הקלט לאלגוריתם המיון, איברי המערך A שונים זה מזה.
על מנת להוכיח חסם תחתון, נרצה להוכיח שיש קלט שעליו האלגוריתם מבצע "הרבה" השוואות.
בלי הגבלת הכלליות, ניתן להניח שאין בעץ "שאלות" (=צמתים) שכאשר מגיעים אליהן התשובה כבר ידועה. אם כך, לכל מסלול בעץ מהשורש לעלים, יש קלט שעובר בדיוק במסלול הזה.



איור 7.1: דוגמה לעץ השוואות

בעץ השוואות של אלגוריתם מיון יש $n!$ עלים (כמספר הפרמוטציות של n מספרים), ואורך מסלול בעץ הוא בדיוק מספר ההשוואות שהאלגוריתם מבצע על קלט שמגיע לעלה הזה.

למה. בעץ בינארי עם M עלים יש מסלול באורך $\log_2(M)$ מהשורש לעלה.

נימוק: בכל צומת פונים לצד שמכיל יותר עלים; בכל פעם מספר העלים בתת-העץ קטן בפקטור של לא יותר מ-2 ולכן אפשר להמשיך לפחות $\log_2(M)$ צעדים.
מכאן נובע שיש מסלול בעץ ההשוואות של אלגוריתם מיון בגודל $\log_2(n!)$. כאשר מתקיים:

$$\log_2(n!) = 1 \cdot n \log_2(n) - O(n)$$

(הקבוע 1 מתקבל ע"י שימוש בקירוב סטירלינג). ניתן להוכיח חסם תחתון כך:

$$n! \geq n(n-1)\cdots\left(\frac{n}{2}+1\right) \geq \left(\frac{n}{2}\right)^{n/2}$$

לכן $\log_2(n!) \geq \frac{n}{2}\log_2\left(\frac{n}{2}\right) = \Omega(n\log(n))$. קיבלנו חסם תחתון על מספר ההשוואות שאלגוריתם מיון מבוסס השוואות מבצע.

8 שבוע 8 - 27.05.18 - פרופ׳ גיא קינדלר

8.1 עומק ממוצע של עלים בעץ בינארי

בעיה. בהינתן עץ בינארי T כלשהו, נרצה לדעת מהו העומק הממוצע של עלה בעץ.

ראשית נתעניין בעומק הממוצע של עלה בעץ בינארי כמעט שלם (ערימה).
יהי T עץ בינארי כמעט שלם עם ℓ עלים. אז אם הרמה התחתונה של T היא מלאה, נקבל ש-$\ell = 2^k$. במקרה זה כל העלים בעץ נמצאים בעומק k ולכן העומק הממוצע של עלה שווה ל-$k = \log_2(\ell)$.
במקרה הכללי, אם $2^k \le \ell < 2^{k+1}$ אז העומק הממוצע של עלה, נסמנו α, מקיים $\lfloor \log_2(\ell) \rfloor \le \alpha \le \lfloor \log_2(\ell) \rfloor + 1$ (תרגיל).

טענה 8.1. יהא T עץ בינארי עם ℓ עלים. אזי העומק הממוצע שלה עלה ב-T הוא לפחות $\lfloor \log_2(\ell) \rfloor$.

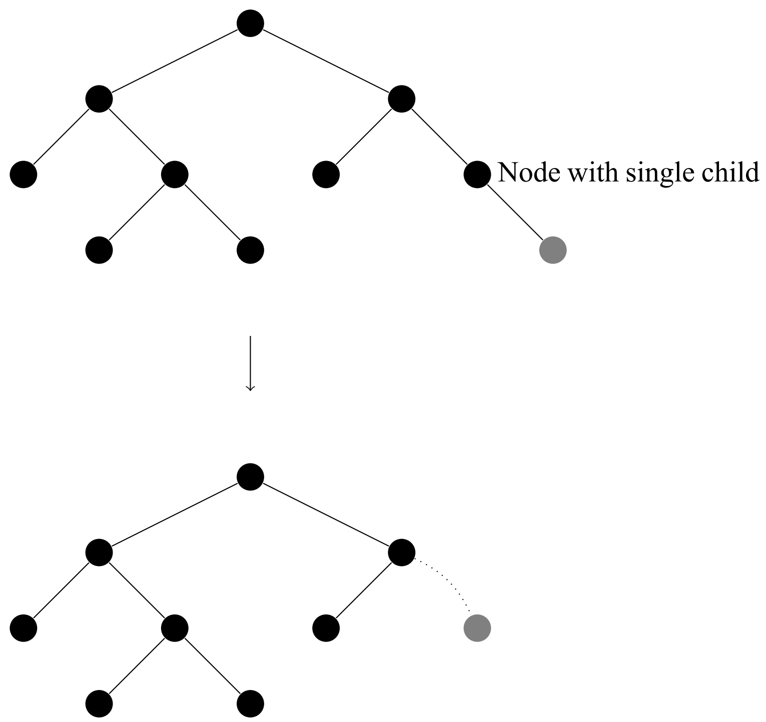
הוכחה. (רעיון ההוכחה: נבנה סדרה של פעולות על T שתעביר את T ל-T' עץ בינארי כמעט שלם, כך שלאחר כל פעולה מספר העלים יישאר זהה אבל העומק הממוצע של עלה יפחת.)
נבנה סדרת עצים $T = T_0, T_1, T_2, \ldots, T_m$ כאשר T_m עץ בינארי כמעט שלם בעל ℓ עלים, כך שהעומק הממוצע של עלה ב-T_i גדול או שווה מהעומק הממוצע של עלה ב-T_{i+1}.
תיאור הפעולות שמעבירות את T ל-T_m:
אם T_i הוא עץ בינארי כמעט שלם אז סיימנו. אחרת נבצע את הפעולות הבאות:

1. מחיקת צומת עם בן יחיד - ניצור קישור חדש בין קודקוד האב של הצומת לבין קודקוד הבן (היחיד). בדרך זו שמרנו על מספר העלים והקטנו את העומק הממוצע של עלה בעץ.

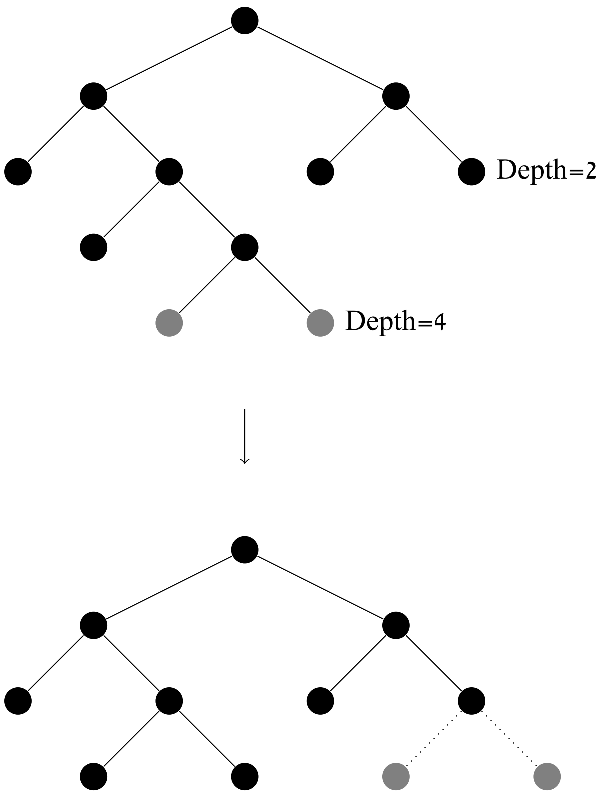
2. אם יש עלים בהפרש עומקים גדול או שווה מ-2, נתלה אחים עמוקים על צומת פחות עמוק. הרעיון מאחורי פעולה זו היא שבערימה כל שני עלים מקיימים שהם באותו עומק או שהפרש העומקים שלהם שווה 1. נשים לב שבפעולה זו מספר העלים לא השתנה, והעומק הממוצע ירד (הוספנו עלה עם עומק גדול יותר והפחתנו עומק של שני עלים).

3. יישור לשמאל של עלים.

בתום סדרת הפעולות נגיע לעץ T_m שהוא עץ ערימה ואז העומק הממוצע של עלה ב-T גדול או שווה מהעומק הממוצע של עלה ב-T_m וזה גדול או שווה מ-$\log_2(\ell)$, כנדרש. □



איור 8.1: דוגמה לפעולות שמעבירות עץ בינארי לעץ בינארי כמעט-שלם: מחיקת צומת עם בן יחיד



איור 8.2: דוגמה לפעולות שמעבירות עץ בינארי לעץ בינארי כמעט-שלם: הפרש עומקים גדול מ-2. באיור מתוארים שני עלים בעומק 4 שלאחר הפעולה נמצאים בעומק 3. שימו לב שמספר העלים נותר זהה והעומק הממוצע של עלה ירד לאחר הפעולה.

8.2 חסמים תחתונים למיון - מקרה ממוצע וטיפול במיון רנדומי

משפט 8.2. יהא $\mathcal{A}$ אלגוריתם מיון מבוסס השוואות, ויהא $T_{\mathcal{A}}(\sigma)$ מספר ההשוואות ש-$\mathcal{A}$ מבצע על פרמוטציה σ. אם n גודל הקלט, אזי $\frac{1}{n!}\sum_{\sigma\in S_n} T_{\mathcal{A}}(\sigma)\geq cn\log_2(n)$.

הערה 8.3. נשים לב שהמשפט דלעיל גורר את המשפט שהוכחנו בשיעור הקודם. זאת משום שאם הממוצע של $n!$ איברים גדול מ-$cn\log_2(n)$ אז לא ייתכן שלכל $\sigma\in S_n$ מתקיים $T_{\mathcal{A}}(\sigma)<cn\log_2(n)$. ז״א קיים $\sigma\in S_n$ עבורו $T_{\mathcal{A}}(\sigma)\geq cn\log_2(n)$. במודל ההשוואות, מספר ההשוואות שמבצע אלגוריתם קובע את זמן הריצה שלו, ולכן מיון n איברים חסום מלרע ע״י $\Omega(n\log(n))$.

הוכחה. נקבע n ויהא T העץ הבינארי שמייצג את ריצת האלגוריתם $\mathcal{A}$ (עץ ההשוואות של $\mathcal{A}$); אז ל-T יש $n!$ עלים. ע״פ טענה 8.1 נקבל שהעומק הממוצע של עלה ב-T הוא לפחות $\lfloor\log_2(n!)\rfloor$ וראינו כבר שמתקיים $\log_2(n!)\approx cn\log_2(n)$. נשים לב ש-$T_{\mathcal{A}}(\sigma)$ זה מס׳ ההשוואות שהאלגוריתם $\mathcal{A}$ מבצע ולכן מדובר בעומק של העלה σ בעץ T. מכאן נובעת מסקנת המשפט. □

משפט 8.4. יהא $\mathcal{A}$ אלגוריתם מיון רנדומי מבוסס השוואות. נסמן ב-$T_{\mathcal{A}}(\sigma)$ את תוחלת זמן הריצה של $\mathcal{A}$ על σ עם הביטים המקריים. אם n גודל הקלט, אזי קיימת פרמוטציה σ כך ש-$T_{\mathcal{A}}(\sigma)\geq cn\log_2(n)$.

הוכחה. עבור קביעה מסוימת $\mathcal{R}$ של הביטים המקריים של $\mathcal{A}$, נסמן ב-$\mathcal{A}_{\mathcal{R}}$ את ריצת האלגוריתם יחד עם הביטים

הללו. אזי $\mathcal{A}_{\mathcal{R}}$ הוא אלגוריתם מיון דטרמיניסטי ולכן ע"פ משפט 8.2 מתקיים:

$$\begin{aligned}
&\frac{1}{n!}\sum_{\sigma\in S_n} T_{\mathcal{A}_{\mathcal{R}}}(\sigma)\geq cn\log_2(n)\\
\Rightarrow&\mathbb{E}_{\mathcal{R}}\left[\frac{1}{n!}\sum_{\sigma\in S_n} T_{\mathcal{A}_{\mathcal{R}}}(\sigma)\right]\geq cn\log_2(n)\\
\Rightarrow&\frac{1}{n!}\sum_{\sigma\in S_n}\mathbb{E}_{\mathcal{R}}\left[T_{\mathcal{A}_{\mathcal{R}}}(\sigma)\right]\geq cn\log_2(n)\\
\Rightarrow&\frac{1}{n!}\sum_{\sigma\in S_n} T_{\mathcal{A}}(\sigma)\geq cn\log_2(n)\\
\Rightarrow&\exists\sigma\in S_n,\quad T_{\mathcal{A}}(\sigma)\geq cn\log_2(n)
\end{aligned}$$

כנדרש. $\square$

8.3 מילון

הגדרה 8.5 (מילון). מילון הוא מבנה נתונים אבסטרקטי שאינו מכיל כפילויות, ותומך בפעולות הבאות:

- $\text{Init}(S)$ - אתחול המילון עם קבוצת איברים ייחודית S.
- $\text{Member}(x)$ - בודק האם x נמצא במילון.
- $\text{Insert}(x)$, $\text{Delete}(x)$ - מוסיף/מסיר את x מהמילון. פעולות אלה נחשבות אופציונליות בתלות אם המילון דינמי או סטטי.

הערה 8.6. חרף שמו של מבנה הנתונים, במונחי Python, המבנה שמתאים להגדרה הוא set.

ניסיון נאיבי למימוש של מילון היינו יכולים להיעזר בעץ חיפוש בינארי מאוזן על מנת לממש מילון; במקרה כזה פעולת האתחול (Init) דורשת $\Theta(n\log(n))$. פעולות ה-Member, Insert, Delete דורשות $\Theta(\log(n))$. אולם, עץ חיפוש בינארי תומך בפעולות נוספות (למשל, מציאת האיבר העוקב). נקווה לקבל זמני ריצה טובים יותר אם נעבוד במבנה נתונים אחר.

8.4 טבלת גיבוב

נסמן ב-U את הקבוצה האוניברסלית - אוסף איברי המילון האפשריים. נניח כי U קבוצה סופית.
תהא $S\subseteq U$ תת-קבוצה; יהא מערך בעל m תאים שייקרא מעתה "טבלת גיבוב".

דוגמה. U יכולה להיות קבוצת כל המספרים בעלי 9 ספרות ו-S קבוצת תעודות הזהות.

הגדרה 8.7. **פונקציית גיבוב** היא פונקציה $h:U\to\{0,1,\ldots,m-1\}$ הממפה איבר מן הקבוצה האוניברסלית אל התא המתאים בטבלת הגיבוב.

כאשר רוצים להכניס איבר $x\in U$ למבנה, מחשבים את $h(x)$ ומקבלים מספר שלם בין 0 ל-$m-1$ שמהווה את האינדקס של התא המתאים ל-x בטבלת הגיבוב.
ייתכנו מקרים שבהם שני איברים $x,y\in U$ ממופים לאותו התא בטבלת הגיבוב. ז"א $h(x)=h(y)$ אבל $x\neq y$. במקרה כזה נאמר שיש **התנגשות**. דרך אחת לפתרון התנגשויות הוא שימוש בשרשור (Chaining) (לעתים נקרא גם "טבלה פתוחה" או באנגלית: Open Addressing).

פתרון התנגשויות ע"י Chaining בכל תא בטבלת הגיבוב יש מצביע לרשימה מקושרת. אם $x,y\in U$ וביצענו פעולת $\text{Insert}(x)$ ואז $\text{Insert}(y)$ וגם $h(x)=h(y)$ (כלומר, יש התנגשות) אז נוסיף את y לרשימה המקושרת שבה נמצא גם x. מאחר שאסורות כפילויות, נעבור על איברי הרשימה המקושרת בטרם נכניס את y. כדי לתמוך בפעולת $\text{Member}(x)$, נחפש את x ברשימה המקושרת שנמצאת בתא $h(x)$.
מבחינת סיבוכיות זמן ריצה, פעולת $\text{Init}(S)$ דורשת $O(n)$ כאשר $n=|S|$ (מניחים שאיברי S שונים).
פעולת ה-$\text{Member}(x)$ דורשת $O(n)$ במקרה הגרוע (כל n האיברים של S מופו לאותו תא).

גיבוב אוניברסלי נסמן ב-H את קבוצת כל פונקציות הגיבוב; הרעיון הוא לבחור $h \in H$ באופן אקראי ואז להשתמש בה לאחסון האיברים. לנותן הקלט אין מידע בדבר הפונקציה שבה השתמשנו, ובכך אנו מקטינים את האפשרות שיקרה המקרה הגרוע בו כל איברי S מופו לאותו התא.

הגדרה 8.8. H נקרא אוסף פונקציות גיבוב **אוניברסלי** אם"ם מתקיים

$$\forall x, y \in U, x \neq y, \quad \Pr_{h \in H}\left[h\left(x\right) = h\left(y\right)\right] \leq \frac{1}{m}$$

יהא $x \in U$ ו-H אוסף פונקציות גיבוב אוניברסלי. אם $|S| = n$ וגם $m \geq n$ אז תוחלת מספר ההתנגשויות של x עם איברי S קטנה מ-1, שכן מתקיים:

$$\begin{aligned}\mathbb{E}_{h \in H}\left[\text{Number of elements } y \text{ such that } h\left(y\right) = h\left(x\right)\right] &= \sum_{y \in S} Pr\left[h\left(y\right) = h\left(x\right)\right] \\ &\leq \frac{n}{m} \\ &\leq 1\end{aligned}$$

9 שבוע 9 - 03.06.18 - פרופ' גיא קינדלר

9.1 המשך גיבוב אוניברסלי

דוגמה 9.1 (דוגמה לאוסף פונקציות גיבוב אוניברסלי). יהא p מספר ראשוני. U, אוסף המפתחות, הוא $(\mathbb{F}_p)^d$ - ז"א וקטורים באורך d מעל השדה $\mathbb{F}_p$.
האינדקסים בטבלת הגיבוב יהיו $M = \{0, 1, \ldots, p-1\}$. נרצה לבחור p ראשוני כך ש-$p = O(n)$ כאשר n מייצג את מספר האיברים בקבוצה $S \subset U$ שבהם אנו צריכים לטפל.
לכל וקטור $a = (a_0, a_1, \ldots, a_d) \in (\mathbb{F}_p)^{d+1}$ נגדיר פונקציית גיבוב h_a באופן הבא:

$$\forall x \in U, \quad h_a(x) = \left(a_0 + \sum_{i=1}^{d} a_i x_i\right) \mod p$$

נגדיר את $\mathcal{H}$, אוסף פונקציות הגיבוב, להיות $\mathcal{H} = \left\{h_a : U \to M | a \in (\mathbb{F}_d)^{d+1}\right\}$.

טענה 9.2. לכל $x, y \in U$ שונים זה מזה, ולכל $\alpha, \beta \in \mathbb{F}_p$ מתקיים $\Pr_{h_a \in \mathcal{H}}[h_a(x) = \alpha \wedge h_a(y) = \beta] = \frac{1}{p^2}$.
(לא נוכיח טענה זו).

הגדרה 9.3. אוסף $\mathcal{H}$ של פונקציות גיבוב מ-U ל-$\{0, \ldots, m-1\}$ נקרא **k-אוניברסלי** אם"ם לכל $x_1, \ldots, x_k \in U$ שונים זה מזה, ולכל $\alpha_1, \ldots, \alpha_k \in \{0, \ldots, m-1\}$ מתקיים $\Pr_{h \in \mathcal{H}}\left[\bigwedge_{i=1}^{k} h(x_i) = \alpha_i\right] = \frac{1}{m^k}$.

עובדה 9.4. אם $\mathcal{H}$ אוסף פונקציות גיבוב 2-אוניברסלי אז הוא אוניברסלי. (תרגיל)

הערה 9.5. יחד עם ההגדרה והעובדה נקבל שהאוסף $\mathcal{H}$ שהוגדר בדוגמה דלעיל הוא אוסף פונקציות גיבוב 2-אוניברסלי ולכן אוניברסלי.

9.2 מיפוי מושלם

נרצה שפעולת $\text{Member}(x)$ תבוצע בזמן $O(1)$ במקרה הגרוע, ולא רק במקרה הממוצע.
לשם כך נוכל להתפשר על:

1. גודל הטבלה;
2. דינאמיות המבנה, ז"א, המבנה יהיה סטטי - לא נתמוך בפעולות Insert ו-Delete.

נתאר אלגוריתם לביצוע $\text{Init}(S)$ כאשר $S \subset U$ בת n איברים, ו-m, גודל הטבלה, הוא $m = cn^2$ (c חיובי).

1. נבחר $h \in \mathcal{H}$ אקראית כאשר $\mathcal{H}$ אוסף פונקציות גיבוב אוניברסלי מ-U ל-$\{0, 1, \ldots, cn^2 - 1\}$.
2. לכל $x \in S$ נכניס את x לטבלה בתא שהאינדקס שלו הוא $h(x)$.
3. אם יש התנגשות, חזור לשלב מספר 1, עד להצלחה (ללא התנגשויות).

טענה 9.6. אם $c > 3$ אז כל איטרציה של האלגוריתם $\text{Init}(S)$ מסתיימת בהצלחה בסיכוי לפחות $\frac{1}{2}$.

הוכחה. איטרציה של האלגוריתם $\text{Init}(S)$ מסתיימת בהצלחה אם אין התנגשויות. לכן, נתעניין בתוחלת מספר ההתנגשויות. נגדיר משתנה מקרי $C_{x,y}(h) = \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{Otherwise} \end{cases}$. כלומר $C_{x,y}(h) = 1$ אם h ממפה את x

ואת y לאותו התא, ז״א, יש התנגשות. ואז נקבל:

$$\begin{aligned}
\mathbb{E}_{h\in\mathcal{H}}\left[\sum_{\substack{x,y\in S\\x\neq y}} C_{x,y}(h)\right] &= \sum_{\substack{x,y\in S\\x\neq y}} \mathbb{E}_{h\in\mathcal{H}}\left[C_{x,y}(h)\right]\\
(\mathcal{H}\text{ is universal}) &\leq \sum_{\substack{x,y\in S\\x\neq y}} \frac{1}{cn^2}\\
&\leq \frac{1}{cn^2}\cdot n(n-1)\\
&\leq \frac{1}{c}\\
&< \frac{1}{3}
\end{aligned}$$

מכאן נובע $\frac{1}{3} > \Pr_{h\in\mathcal{H}}\left[\left(\sum_{\substack{x,y\in S\\x\neq y}} C_{x,y}(h)\right)\geq 1\right]$, ולכן $\Pr_{h\in\mathcal{H}}[\text{No collisions}] \geq \frac{2}{3} > \frac{1}{2}$, כנדרש. □

מסקנה 9.7. תוחלת מספר החזרות עד להצלחה של האלגוריתם $\text{Init}(S)$ קטנה או שווה מ-2.

הערה 9.8. זמן הריצה של $\text{Init}(S)$ הוא $\text{O}(n)$ בתוחלת.

מיפוי דו-שלבי

הרעיון: נסמן טבלה M בגודל n.
השלב הראשון הוא גיבוב האיברים לתוך הטבלה באמצעות פונקציית גיבוב $h\in\mathcal{H}$ כאשר $\mathcal{H}$ אוסף פונקציות גיבוב. ברור שבמצב כזה ייתכנו התנגשויות. הפעם, במקום להשתמש ברשימות מקושרות, ניעזר בטבלת גיבוב נוספת - כאשר נבטיח שתהיה ללא התנגשויות.
השלב השני הוא שלכל תא $i=0,1,\ldots,n-1$ בטבלה שלנו M נבחר פונקציית גיבוב $h_i\in\mathcal{H}$. אם יש k_i איברים שמתאימים לתא ה-i (ז״א הפונקציה h ממפה k_i איברים לאותו התא), נבנה טבלת גיבוב משנית בגודל $|M_i|=3k_i^2$, שאליה נצביע מהתא ה-i. בחרנו את גודל הטבלה להיות ריבועי ביחס למספר האיברים שישוכנו שם כיוון שזה יאפשר לנו לבצע Perfect Hashing שם.

תמיכה בפעולת Member: בתכנון הזה, פעולת Member תבוצע ע״י חישוב דו-שלבי. אם $x\in U$ אז תחילה נחשב $i:=h(x)$ ואז נחשב $j:=h_i(x)$, ונבדוק בטבלת הגיבוב M_i באינדקס ה-j - אם $M_i[j]=x$ אז x נמצא; אחרת, x לא נמצא במבנה.

צריכת זיכרון: נסמן כמקודם ב-k_i את מספר האיברים שמופו לתא ה-i בטבלה. סה״כ צריכת הזיכרון נתונה ע״י:

$$\text{O}(n)+\text{O}\left(\sum_{i=1}^{n}k_i^2\right)$$

טענה 9.9. אם $\mathcal{H}$ הוא אוסף פונקציות גיבוב 2-אוניברסלי אז $\mathbb{E}_{h\in\mathcal{H}}\left[\sum_{i=1}^{n}k_i^2\right]\leq 4n$.

(ההוכחה תושלם בשבוע הבא).

10 שבוע 10 - 10.06.18 - פרופ׳ גיא קינדלר

10.1 מיפוי מושלם - השלמה

נוכיח את טענה 9.9.

הוכחה. נגדיר $C_{x,y,i}(h)=\begin{cases}1 & h(x)=h(y)=i\\ 0 & \text{Otherwise}\end{cases}$, כלומר $C_{x,y,i}(h)$ הוא 1 אס״ם x ו-y מופו לתא ה-i ע״י h.
האבחנה המרכזית היא שלכל $i=1,\ldots,n$ מתקיים $\sum_{x,y\in S} C_{x,y,i}(h)$ = k_i^2 - מספר האיברים בריבוע שמופו לתא ה-i בטבלה שווה למספר הזוגות הסדורים של $x,y\in S$ שעליהם $C_{x,y,i}(h)$ נותן 1.
נקבע $i=1,\ldots,n$ ונעריך את $\mathbb{E}_{h\in\mathcal{H}}\left[k_i^2\right]$ באופן הבא:

$$\begin{aligned}\mathbb{E}_{h\in\mathcal{H}}\left[k_i^2\right] &= \mathbb{E}_{h\in\mathcal{H}}\left[\sum_{x,y\in S} C_{x,y,i}(h)\right]\\ (\text{לינאריות התוחלת}) &= \sum_{x,y\in S}\mathbb{E}_{h\in\mathcal{H}}\left[C_{x,y,i}(h)\right]\\ (\text{תכונות של משתנה אינדיקטור}) &= \sum_{x,y\in S}\Pr_{h\in\mathcal{H}}\left[h(x)=h(y)=i\right]\end{aligned}$$

בשלב הזה ניאלץ לפצל את הסכום לזוגות הסדורים שבהם $y\neq x$ ולאלה שבהם $x=y$. הסכום ייראה כך:

$$\sum_{\substack{x,y\in S\\ x\neq y}}\Pr_{h\in\mathcal{H}}\left[h(x)=h(y)=i\right]+\sum_{x\in S}\Pr_{h\in\mathcal{H}}\left[h(x)=i\right]=\sum_{\substack{x,y\in S\\ x\neq y}}\frac{1}{m^2}+\sum_{x\in S}\frac{1}{m}$$

כאשר השוויון נובע מכך ש-$\mathcal{H}$ הוא 2-אוניברסלי, וזה גורר ש-$\mathcal{H}$ הוא גם 1-אוניברסלי. מכאן נקבל:

$$\sum_{\substack{x,y\in S\\ x\neq y}}\frac{1}{m^2}+\sum_{x\in S}\frac{1}{m}\le\frac{n^2}{m^2}+\frac{n}{m}$$

זכרו שהטבלה שלנו בגודל n, ולכן $m=n$ וקיבלנו $\mathbb{E}_{h\in\mathcal{H}}\left[k_i^2\right]\le 2$ שזה מספר קבוע.
עתה נעריך את הזיכרון שנקצה לטבלאות המשניות:

$$\mathbb{E}_{h\in\mathcal{H}}\left[\sum_{i=1}^{n}k_i^2\right]=\sum_{i=1}^{n}\mathbb{E}_{h\in\mathcal{H}}\left[k_i^2\right]\le 2n$$

לכן $\mathbb{E}_{h\in\mathcal{H}}\left[\sum_{i=1}^{n}k_i^2\right]=\mathrm{O}(n)$, כנדרש. □

הערה 10.1. המסקנה מהטענה האחרונה היא שבתוחלת צריכת הזיכרון תהיה לינארית. אם נרצה להבטיח צריכת זיכרון לינארית, ננסה הקצאה לטבלה עם פונקציית גיבוב מסוימת עד שנגיע למצב שבו צריכת הזיכרון לינארית. מספר החזרות בתוחלת עד להצלחה יהיה קבוע.

10.2 מבנה נתונים Union-Find

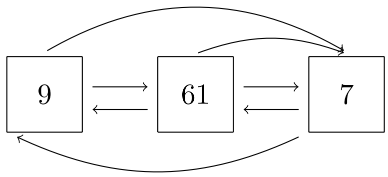
Union-Find הוא מבנה נתונים שמחזיק אוסף של קבוצות זרות, ותומך בפעולות הבאות (x מייצג אוביקט):

- $\text{Make-Set}(x)$: יוצרת יחידון עם האיבר x. דורשים ש-x לא נמצא כבר במבנה (הקבוצות שנמצאות במבנה צריכות להיות זרות).
- $\text{Union}(x,y)$: מאחדת את הקבוצה ש-x מוכל בה עם הקבוצה ש-y מוכל בה.
- $\text{Find}(x)$: מחזיר איבר מייצג של הקבוצה ש-x נמצא בה. x ו-y באותה קבוצה אס״ם $\text{Find}(x)=\text{Find}(y)$.

למבנה נתונים זה קיימים מימושים רבים; נתאר מימוש הנסמך על רשימות מקושרות.

10.3 מימוש Union-Find באמצעות רשימות מקושרות

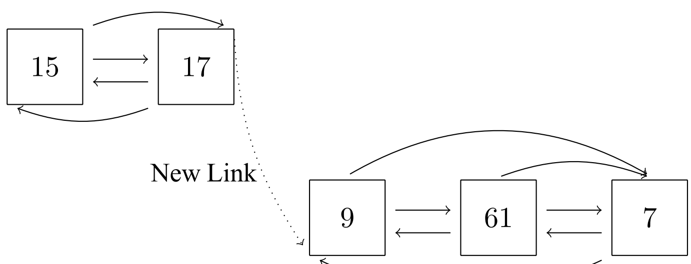
במימוש זה, קבוצה במבנה מיוצגת באמצעות רשימה מקושרת דו-כיוונית. כל איבר ברשימה המקושרת מכיל מצביע לראש הרשימה (בנוסף למצביעים $next$ ו-$tail$ הרגילים). ראש הרשימה מכיל מצביע לסוף הרשימה. בנוסף, נשמור מונה של מספר האיברים בקבוצה.



איור 10.1 : רשימה מקושרת שמייצגת את הקבוצה $\{9, 61, 7\}$. כאן 7 הוא ראש הרשימה.

מימוש הפעולות:

- Make-Set (x) : נבנה רשימה מקושרת בעלת איבר יחיד ובו הערך x ; ראש הרשימה יהיה x ; נציג הקבוצה הוא x ומונה הקבוצה יהיה 1.
- Find (x) : נחזיר את ראש הרשימה שבה נמצא x ע״י מעקב אחרי המצביע מ-x לראש רשימתו. (כאן x הוא אוביקט ולא ערך ; מניחים ש-x נמצא במבנה).
- Union (x, y) : נחבר את הרשימה הקצרה לסוף הרשימה הארוכה, נעדכן מצביעים ככל שנדרש ונעדכן את מונה הקבוצה.



איור 10.2 : עבור הפעולה Union $(61, 15)$, נמצא את האיבר האחרון ברשימה של 61 (הארוכה יותר) ונחבר את ראש הרשימה הקצרה, 17, לאיבר 9. כמובן שניאלץ לעדכן מצביעים של הרשימה הקצרה, עכשיו המצביע לראש הרשימה עבור 17 ו-15 יתעדכן להיות 7 (ראש הרשימה החדש). המצביע לסוף הרשימה של האיבר 7 יתעדכן גם הוא להיות 15.

זמן ריצה: הפעולות Make-Set ו-Find לוקחות $\mathrm{O}(1)$.
נסמן ב-m את מספר פעולות ה-Make-Set, Union ו-Find שביצענו על המבנה ;
נסמן ב-n את מספר פעולות ה-Make-Set שביצענו (זה אומר שבמבנה יש n איברים, וגם $m \geq n$).
כל פעולת Union מאחדת שתי קבוצות זרות ולכן ניתן לבצע $n-1$ פעולות Union לכל היותר.
נקבע אוביקט x. הפוינטר של x לראש הרשימה מתעדכן אס״ם הקבוצה שבה x מוכל מתמזגת עם קבוצה גדולה ממנה. אם ℓ הוא גודל הקבוצה שבה x מוכל, זה אומר שלאחר האיחוד, הקבוצה החדשה תכיל לפחות 2ℓ איברים. מאחר שיש לכל היותר n איברים במבנה (ולכן הקבוצה הגדולה ביותר היא בגודל n), נקבל חסם על מספר הפעמים שהמצביע של x מעודכן - $\mathrm{O}(\log(n))$. לאחר $n-1$ פעולות Union נקבל שסה״כ הזמן לעדכן את המצביעים לוקח $\mathrm{O}(n\log(n))$. יש לכל היותר m פעולות Make-Set ו-Find שכל אחת מהן לוקחת $\mathrm{O}(1)$, ולכן נקבל שזמן הריצה לכל רצף הפעולות הוא $\mathrm{O}(m + n\log(n))$.

10.4 עץ פורש מינימלי

בעיה 10.2. נתון גרף לא מכוון $G=(V,E)$, $|V|=n$, $|E|=m$; בנוסף נתונה פונקציית משקל $w:E\to\mathbb{R}^+$ שמקצה לכל צלע "משקל" חיובי. נרצה למצוא תת-גרף T של G בעל משקל מינימלי, כך ש-T יהיה גרף חסר מעגלים וקשיר.

הגדרה 10.3. $G=(V,E)$ גרף לא-מכוון. **יער על** V הוא תת-גרף שקבוצת הקודקודים שלו היא V וקבוצת הצלעות שלו לא יוצרת מעגל. יער על V שבו קיים מסלול בין כל שני צמתים ב-V נקרא **עץ פורש**.

הערה 10.4. בעץ פורש על n קודקודים יש לפחות $n-1$ צלעות, כי אם התחלנו עם n רכיבי קשירות שונים, תוספת צלע מורידה את מספר רכיבי הקשירות לכל היותר ב-1. בעץ פורש יש בדיוק $n-1$ צלעות, כי אם יש n צלעות אז נסגור מעגל.

אלגוריתם חמדני למציאת עץ פורש מינימלי (האלגוריתם של Kruskal):

1. אתחל מבנה נתונים Union-Find. בצע $\text{Make-Set}(v)$ לכל $v\in V$ קודקוד ב-G.
2. מיין את הצלעות ע"פ משקלן (מהנמוך לגבוה);
3. עבור על הצלעות לפי סדר משקל עולה, ולכל צלע e בצע:
   - א) נסמן $e=\{u,v\}$. נבדוק האם $\text{Find}(u)=\text{Find}(v)$?
     - i. אם כן, e סוגרת מעגל עם היער הקיים, ונעבור לצלע הבאה;
     - ii. אם לא, נבצע $\text{Union}(u,v)$.

ניתוח זמן ריצה: שלב אתחול מבנה הנתונים Union-Find דורש n פעולות Make-Set, כאשר n מספר הקודקודים; מיון הצלעות לוקח $O(m\log(m))$ כאשר m מספר הצלעות.
מספר הפעולות Make-Set, Find ו-Union הוא $O(m)$ (פעולת Find מבוצעת פעמיים עבור כל צלע, ופעולת Union מבוצעת לכל היותר $m>n-1$ פעמים). לפיכך, לפי ניתוח זמן ריצה של מבנה Union-Find, נקבל שזמן הריצה של האלגוריתם הינו:

$$O\left(\underbrace{m\log(m)}_{\text{מיון}}+\underbrace{m+n\log(n)}_{\text{Union-Find}}\right)=O(m\log(m))$$

כאשר השוויון נובע מכך ש-$m\geq n$ כי הנחת העבודה היא שהגרף G קשיר.

נכונות האלגוריתם:

למה 10.5 (למת החתך)**.** יהי H תת-יער של G, תהי $C\subseteq V$ תת-קבוצה של קודקודים ונניח ש-H לא מכיל צלעות בין קודקוד של C לקודקוד של $V\setminus C$. אם H מוכל בעץ פורש מינימלי ומוסיפים ל-H צלע e המקשרת בין C ל-$V\setminus C$ שהיא בעלת משקל מינימלי, אזי $H\cup\{e\}$ היא תת-קבוצה של עץ פורש מינימלי.

טענה 10.6. בכל איטרציה של האלגוריתם, היער שהאלגוריתם מחזיק הוא תת-גרף של עץ פורש מינימלי כלשהו.

טענה 10.7. אם G קשיר אז האלגוריתם ימצא בו עץ פורש מינימלי.

הוכחת הנכונות תושלם בשבוע הבא.

11 שבוע 11 - 17.06.18 - פרופ׳ גיא קינדלר

11.1 השלמת הוכחת נכונות האלגוריתם של Kruskal

נפתח בהגדרה שתסייע להבנת למת החתך.

הגדרה 11.1. $G=(V,E)$ גרף לא מכוון, $C\subseteq V$ תת-קבוצה של קודקודים. **החתך המוגדר ע״י** C, שמסומן $(C,\bar{C})$, הוא אוסף הצלעות $e=\{u,v\}$ כך ש-$u\in C$ ו-$v\in\bar{C}$ (או ההפך: $v\in C$ ו-$u\in\bar{C}$).

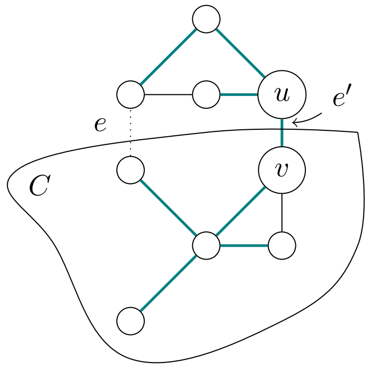
בעזרת ההגדרה ניתן ניסוח חלופי ללמת החתך (למה 10.5):

למה 11.2 (למת החתך*)**.** יהי $G=(V,E)$ גרף לא מכוון בעל משקלים על הצלעות, מוגדרים ע״י הפונקציה $w:E\to\mathbb{R}^+$. יהי $H\subseteq E$ ו-$T\subseteq E$ כך ש-T הוא עץ פורש מינימלי ו-$H\subseteq T$. תהי $C\subseteq V$ קבוצת קודקודים בגרף ותהא צלע $e\in(C,\bar{C})$.
אם H לא מכיל צלע ב-$(C,\bar{C})$ וגם e היא בעלת משקל מינימלי בחתך $(C,\bar{C})$, אזי $H\cup\{e\}$ מוכל בעץ פורש מינימלי.

הוכחה. אם $e\in T$ אז $H\cup\{e\}\subseteq T$, כנדרש. אחרת, $e\notin T \Leftarrow$ אם נוסיף את e ל-T נקבל מעגל. זה אומר שקיימת צלע $e'\in T$ כך ש-$e'\in(C,\bar{C})$, כלומר e' מקשרת בין קודקוד ב-C ל-$\bar{C}$, וגם שייכת ל-T שהוא עץ פורשת מינימלי.
נחליף את הצלע e' בצלע e ונקבל עץ פורש $T'=(T\setminus\{e'\})\cup\{e\}$ (T' קשיר כי בהסרת הצלע e' חילקנו את T לשני רכיבי קשירות, שאותם חיברנו עם הצלע החדשה e).
נותר להראות ש-T' הוא עץ פורש מינימלי. ואכן, מכך ש-e בעלת משקל מינימלי בחתך נקבל כי:

$$w(T')=w(T)-w(e')+w(e)\underbrace{\leq}_{w(e)\leq w(e')}w(T)$$

העץ T הוא עץ פורש מינימלי ומכך ש-$w(T')\leq w(T)$ נקבל כי T' הוא עץ פורש מינימלי.
מתקיים $H\subseteq T'$ משום ש-$H\not\ni e'$ כי לפי הנתון H לא מכיל צלע בחתך המוגדר ע״י C. לכן $H\cup\{e\}\subseteq T'$, כלומר $H\cup\{e\}$ מוכל בעץ פורש מינימלי, כנדרש. □



איור 11.1: המחשה להוכחת למת החתך; הצלעות שמסומנות בירוק הן הצלעות של T.

למת החתך גוררת את נכונות האלגוריתם של Kruskal (טענה 10.6).

הוכחה. שלב האתחול נכון באופן ריק, שהרי בהתחלה היער לא מכיל צלעות כלל. נניח שבאיטרציה מסוימת של האלגוריתם היער שהאלגוריתם מחזיק מוכל בעץ פורש מינימלי כלשהו. באיטרציה הבאה האלגוריתם בוחר צלע $e=\{u,v\}$ שאינה סוגרת מעגל עם היער הקיים, ובעלת משקל מינימלי ביחס לכל הצלעות שלא סוגרות מעגל. נגדיר חתך C בתור רכיב הקשירות של u ביער (טיעון זה תקף גם אם היינו לוקחים את C בתור רכיב הקשירות של v).
היער שהאלגוריתם מחזיק מקיים את תנאי למת החתך: u ו-v נמצאים ברכיבי קשירות שונים, ולכן לא קיים מסלול ביניהם (אחרת, הצלע e הייתה סוגרת מעגל עם היער הקיים); בנוסף, e היא צלע בחתך המוגדר ע״י C והיא בעלת משקל מינימלי ביחס לכל הצלעות שלא סוגרות מעגל - בפרט זה נכון עבור הצלעות בחתך המוגדר ע״י C. יתר על כן, מהנחת האינדוקציה, היער הקיים מוכל בעץ פורש מינימלי כלשהו. נובע מכאן שאם נוסיף את e ליער אז נקבל תת-גרף של עץ פורש מינימלי (לאו דווקא אותו העץ הפורש), כנדרש. □

11.2 מרחקים קצרים בין כל הזוגות - All-Pairs Shortest-Paths

בעיה 11.3. יהי $G = (V, E)$ גרף מכוון. תהי $\ell : E \to \mathbb{R}^+$ פונקציה המתאימה לכל צלע "אורך", אם $(x, y) \notin E$ אז $\ell(x, y)$ מוגדר להיות ∞. אם $P = (u_0 = u, u_1, \ldots, u_k = v)$ מסלול בין u ל-v אז $\ell(P)$ מציין את אורך המסלול ומוגדר כך: $\ell(P) := \sum_{i=0}^{k-1} \ell(u_i, u_{i+1})$; המסלול הקצר ביותר בין u ל-v מוגדר כך:

$$d(u, v) := \min\{\ell(P) : P \text{ is a path from } u \text{ to } v\}$$

לכל זוג של קודקודים $u, v \in V$ נרצה למצוא את המסלול הקצר ביותר ביניהם, כלומר את $d(u, v)$. במילים אחרות, נרצה למצוא את איברי המטריצה D כאשר $D_{ij} = d(u_i, u_j)$. המטריצה D לעתים נקראת: "מטריצת המרחקים".

בתור מוטיבציה נחשוב על חברת תעופה, שמעוניינת לחשב את המחיר הנמוך ביותר שנוסע ייאלץ לשלם כדי להגיע מעיר א' לעיר ב'. ניתן למדל את הבעיה הזאת בתור גרף מכוון, שבו הערים הם הקודקודים וקיימת צלע בין עיר א' לעיר ב' אם ורק אם קיימת טיסה הממריאה מעיר א' ונוחתת בעיר ב'. בנוסף, קיימת "פונקציית אורך", ℓ, המתאימה לכל צלע את מחיר הטיסה.

אלגוריתם תכנון דינאמי:

לטובת פתרון הבעיה נזדקק להגדרות נוספות.

הגדרה 11.4. המסלול הקצר ביותר בין u ל-v שיש בו לכל היותר k צלעות ("דילוגים") מוגדר ע"י:

$$d^{\le k}(u, v) := \min\{\ell(P) : P \text{ is a path from } u \text{ to } v \text{ with at most } k \text{ edges}\}$$

מטריצת המרחקים עם לכל היותר k דילוגים, מסומנת $D^{\le k}$, ומוגדרת ע"י $D_{ij}^{\le k} := d^{\le k}(u_i, u_j)$.

עובדה 11.5. את המטריצה $D^{\le 1}$ אפשר לחשב בקלות, שהרי $D_{ij}^{\le 1} = \ell(u_i, u_j)$. בנוסף, $D_{ij} = D_{ij}^{\le k}$ עבור $k \ge n - 1$ (כאשר n הוא מספר הקודקודים). הסיבה לכך היא שאם יש מסלול עם יותר מ-$n - 1$ דילוגים אז הוא בהכרח מכיל מעגל, כלומר, ניתן למצוא מסלול "זול" יותר.

טענה 11.6. תחת הגדרות בעיה 11.3, מתקיימות התכונות הבאות:

1. $d(u, v) \le d(u, w) + d(w, v)$, לכל $u, v, w \in V$;
2. $d^{\le(k+l)}(u, v) \le d^{\le k}(u, w) + d^{\le l}(w, v)$, לכל $u, v, w \in V$;
3. לכל $u, v \in V$ קיים $w \in V$ עבורו מתקיים: $d^{\le(k+l)}(u, v) = d^{\le k}(u, w) + d^{\le l}(w, v)$.

הוכחה. הוכחת תכונות 1 ו-2 מושארות כתרגיל. נוכיח את תכונה 3.
יהא $P = (u_0 = u, u_1, \ldots, u_m = v)$ המסלול הקצר ביותר בין u ל-v עם מספר צעדים קטן מ-$k + l$.
אזי מתקיים $m \le k + l$, נגדיר $i := \min\{m, k\}$, $w := u_i$ (שימו לב שאם $m < k$ אז w מתלכד עם v).
נסמן $P_1 = (u_0 = u, \ldots, u_i = w)$ ו-$P_2 = (u_i = w, \ldots, u_m = v)$.
P_1 הוא מסלול בין u ל-w עם פחות מ-k דילוגים $\Leftarrow$ $\ell(P_1) \ge d^{\le k}(u, w)$.
בדומה, P_2 הוא מסלול בין w ל-v עם פחות מ-l דילוגים $\Leftarrow$ $\ell(P_2) \ge d^{\le l}(w, v)$.
סה"כ קיבלנו:

$$d^{\le(k+l)}(u, v) = \ell(P) = \ell(P_1) + \ell(P_2) \ge d^{\le k}(u, w) + d^{\le l}(w, v)$$

מתכונה 2 של הטענה מתקיים באופן כללי, $d^{\le(k+l)}(u, v) \le d^{\le k}(u, w) + d^{\le l}(w, v)$, ומכאן נובע השוויון הדרוש.
□

מסקנה 11.7. לכל $u, v \in V$ מתקיים $d^{\le(k+l)}(u, v) = \min_{w \in V}\left\{d^{\le k}(u, w) + d^{\le l}(w, v)\right\}$.
בלשון המטריצות ננסח זאת כך: $D_{ij}^{\le(k+l)} = \min_{1 \le t \le n}\left\{D_{i,t}^{\le k} + D_{t,j}^{\le l}\right\}$, כאשר n מייצג את מספר הקודקודים.

תיאור האלגוריתם:

1. חשב את $D^{\leq 1}$.
2. לכל $k = 1, 2, 3, \ldots, \lceil \log_2 (n) \rceil$ (כאשר n מייצג את מספר הקודקודים), חשב את $D^{\leq 2^k}$ ע״י:

$$D_{ij}^{\leq 2^k} = \min_{1 \leq t \leq n} \left\{ D_{i,t}^{\leq 2^{k-1}} + D_{t,j}^{\leq 2^{k-1}} \right\}$$

3. הדפס כפלט את המטריצה האחרונה שחושבה.

זמן ריצה: מספר האיטרציות הוא $\log_2 (n)$. בכל איטרציה אנחנו מחשבים את $D_{ij}^{\leq 2^k}$ לכל זוג אינדקסים (i, j), ויש n^2 זוגות כאלו. זמן החישוב של קואורדינטה יחידה במטריצה הוא $\mathrm{O}(n)$ (הזמן שלוקח לחשב את ה-min, בשלב 2 של האלגוריתם). סה״כ אנו מקבלים:

$$\mathrm{O}\left(\log (n) \cdot n^2 \cdot n\right) = \mathrm{O}\left(n^3 \log (n)\right)$$

12 שבוע 12 - 24.06.18 - פרופ' גיא קינדלר

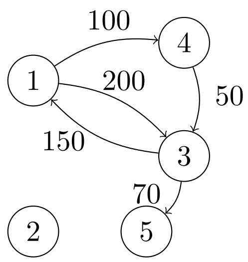
12.1 אלגוריתם Flowd-Warshall

נראה כיצד ניתן לפתור את בעיה 11.3 (בעיית ה-APSP) באמצעות אלגוריתם תכנון דינאמי אחר שפותר את הבעיה בדרך יעילה יותר. ההבדל בין האלגוריתם של Flowd-Warshall לאלגוריתם שהוצג בשבוע 11 הוא באיפיון המבנה של תת-הבעיה.

הגדרה 12.1. יהי גרף מכוון $G=(V,E)$ כאשר $V=\{1,\ldots,n\}$. נגדיר את המטריצה $F^{(k)}$ ע"י

$$F_{ij}^{(k)}=\min\left\{\ell\left(P\right):P\text{ is a path from }i\text{ to }j\text{, using intermediate vertices }\{1,2,\ldots,k\}\text{ only}\right\}$$

הערה 12.2. הכוונה ב-"intermediate vertices" היא ל"צמתי ביניים". כלומר, אנו מתעניינים במסלולים $i\overset{P}{\rightsquigarrow}j$ כך שאם $P=(x_0=i,x_1,\ldots,x_{m-1},x_m=j)$ אז $x_1,\ldots,x_{m-1}\in\{1,\ldots,k\}$. במילים אחרות, כל הקודקודים ב-P לקוחים מהקבוצה $\{1,2,\ldots,k\}$ פרט, אולי, לקודקוד ההתחלה והסיום.



איור 12.1: בגרף המתואר באיור מתקיים $F_{1,3}^{(5)}=150$. לעומת זאת, $F_{1,3}^{(3)}=200$, כי לא ניתן להשתמש בקודקוד מספר 4. $F_{1,3}^{(0)}=F_{1,3}^{(1)}=F_{1,3}^{(2)}=200$, שהרי קודקוד 3 לא נחשב "צומת ביניים".

כדי לפתור את בעיית ה-APSP, נרצה לחשב את המטריצה $F:=F^{(n)}$ - מטריצת המרחקים הקצרים ביותר.

עובדה 12.3. חישוב המטריצה $F^{(0)}$ הוא ישיר שהרי

$$F_{ij}^{(0)}=\begin{cases}\ell\left(i,j\right) & (i,j)\in E\\ 0 & i=j\\ \infty & (i,j)\notin E\end{cases}$$

, ℓ היא פונקציית האורך שהוגדרה בבעיה 11.3.

נסמן ב-P את המסלול הקצר ביותר בין i ל-j שמשתמש בצמתי ביניים $\{1,2,\ldots,k\}$. ייתכנו שתי אפשרויות:

- הקודקוד k לא מופיע בתור צומת ביניים ב-P, במקרה זה $F_{ij}^{(k)}=F_{ij}^{(k-1)}$;
- הקודקוד k מופיע בתור צומת ביניים ב-P, במקרה הזה $F_{ij}^{(k)}=F_{i,k}^{(k-1)}+F_{k,j}^{(k-1)}$.

בהינתן שחישבנו את $F^{(k-1)}$ נחשב את $F^{(k)}$ באמצעות: $F_{ij}^{(k)}=\min\left\{F_{ij}^{(k-1)},F_{i,k}^{(k-1)}+F_{k,j}^{(k-1)}\right\}$.

תיאור האלגוריתם:

1. חשב את $F^{(0)}$;
2. לכל $k=1,\ldots,n$ חשב את $F^{(k)}$ בעזרת $F^{(k-1)}$;
3. החזר כפלט את $F^{(n)}$.

זמן ריצה: נסמן ב-n את מספר הקודקודים, ז"א $n:=|V|$.
הזמן שלוקח כדי לחשב את $F^{(0)}$ הוא $O\left(n^2\right)$ כי המטריצה היא מסדר $n\times n$ וכל Entry בה מחושב ב-$O\left(1\right)$.
האלגוריתם מבצע n איטרציות, ובכל איטרציה מחשב n^2 קואורדינטות שכל אחת מהן ניתנת לחישוב בזמן $O\left(1\right)$.
בסה"כ מקבלים: $O\left(n^2+n\cdot n^2\right)=O\left(n^3\right)$.

דרך נוספת לפתרון בעיית APSP: נעיר שניתן לפתור את בעיית APSP באמצעות אלגוריתם Dijkstra. כזכור, אלגוריתם Dijkstra מקבל כקלט קודקוד מסוים, s, ומוצא את כל המרחקים הקצרים מ-s אל שאר הקודקודים בגרף בזמן $O\left(\left(n+m\right)\log\left(n\right)\right)$, כאשר n מייצג את מספר הקודקודים ו-m מייצג את מספר הצלעות. אם נפעיל את אלגוריתם Dijkstra על כל אחד מהקודקודים שלנו, הרי שנפתור את בעיית ה-APSP בזמן $O\left(n\left(n+m\right)\log\left(n\right)\right)=$ $O\left(nm\log\left(n\right)\right)$. נבחין כי אם $m=\Theta\left(n^2\right)$ (למשל, במקרה של גרף שלם) אז נקבל אלגוריתם פחות טוב מזה של Flowd-Warshall.

12.2 אלגוריתמי שטף - Streaming Algorithms

אלגוריתמי שטף (באנגלית: Streaming Algorithms, או Data Stream Algorithms) הם האלגוריתמים שמאפשרים לעבד סדרה של איברים (Stream), במעבר אחד ובכמות מוגבלת של זיכרון. כלומר, אם נתון שטף $s_1, s_2, \dots, s_t$, אז נדרוש מהאלגוריתם להשתמש בכמות זיכרון שאינה תלויה ב-t (חושבים על t כעל מספר גדול מאוד).
אלגוריתמי שטף מספקים מענה לשאלות כמו "כמה איברים שונים היו ב-Stream?", "מי הם האיברים שמופיעים בתדירות גבוהה ב-Stream?", וכיו"ב. מכיוון שאין אפשרות לשמור את שטף הנתונים במלואו, לא ניתן לספק תשובות מדויקות לשאלות הללו; יחד עם זאת, ניתן לספק הערכה לתשובה הנכונה.
לאלגוריתמי שטף יש מספר שימושים, לדוגמה:

- ניטור רשתות תקשורת - דרך Router או Switch עוברת כמות נתונים גדולה מאוד בכל שניה. נוכל להתעניין, למשל, במספר ה-flows השונים שעברו ברכיב התקשורת.
- זחלן רשת ("Web Crawler") שתפקידו למפתח (index) את ה-Public Web, עשוי להשתמש באלגוריתמי שטף כדי לענות על שאילתות שונות.

הגדרה 12.4. אלגוריתם רנדומי הוא (M, ε) קירוב לפונקציה f של הקלט אם"ם הפלט שלו, p, מקיים

$$\frac{f\left(\text{Input}\right)}{M}<p<Mf\left(\text{Input}\right)$$

בהסתברות לפחות $1-\varepsilon$.

בעיה 12.5. בהינתן שטף $s_1, s_2, \dots, s_t$ עם איברים השייכים לקבוצה אוניברסילית U, נתעניין בקירוב מספר האיברים השונים שהיו בשטף.

פתרון. יהי $\mathcal{H}$ אוסף פונקציות גיבוב מ-U ל-$\{1,2,\dots,m\}$ כאשר $m=10^6\cdot t$ (לא נצטרך לשמור את הטבלה בזיכרון). נבחר $h\in\mathcal{H}$ מקרית ונפעיל עליה את האלגוריתם הבא:

1. נגדיר $a:=m$;
2. לכל $i=1,2,\dots,t$ קבע: $a=\min\left\{a, h\left(s_i\right)\right\}$;
3. החזר $\left(\frac{m}{a}-1\right)$.

כלומר, בכל איטרציה אנחנו שומרים את האינדקס הכי קטן ש-h ממפה איבר משטף הנתונים לטבלה.

טענה 12.6. יהיו $y_1, y_2, \dots, y_k$ איברים שונים ונניח כי הם ממופים למקומות רנדומיים בטבלה $\{1,\dots,m\}$. כל איבר ממופה לתא שנבחר בצורה בלתי-תלויה. אם נסמן ב-s את התא המינימלי שמאוחסן ע"י אחד האיברים, אזי

$$\mathbb{E}\left[s\right]=\frac{m}{k+1}$$

הערה 12.7. הטענה לא נכונה כאשר $\mathcal{H}$ הוא 2-אוניברסלי (כי המיפוי לתאים לאו דווקא מתבצע בצורה בלתי-תלויה); לשם נוחיות נניח כי $\mathcal{H}$ מכילה את אוסף כל פונקציות הגיבוב מ-U ל-m. הטענה מובאת ללא הוכחה.

ניתוח האלגוריתם: נניח שיש k איברים שונים ברצף. מהו הסיכוי ש-$a < \frac{m}{2k}$?

נניח כי האיברים השונים הם $y_1, y_2, \ldots, y_k$; נגדיר משתנה מקרה אינדיקטור $C_j = \begin{cases} 1 & h(y_j) < \frac{1}{2} \cdot \frac{m}{k} \\ 0 & \text{Otherwise} \end{cases}$.

אזי $\mathbb{E}[C_j] = \frac{1}{2} \cdot \frac{m}{k} \cdot \frac{1}{m} = \frac{1}{2k}$ כי יש $\frac{m}{2k}$ תאים שונים, וההסתברות של y_j להיות ממופה לאחד מהתאים היא $\frac{1}{m}$ (אם מניחים ש-$\mathcal{H}$ הוא אוסף 2-אוניברסלי אז הוא גם 1-אוניברסלי, בפרט נכון עם ההנחה ש-$\mathcal{H}$ הוא אוסף כל הפונקציות).

מכאן אנו מקבלים $\mathbb{E}\left[\sum_{j=1}^{k} C_j\right] = \sum_{j=1}^{k} \mathbb{E}[C_j] = \frac{1}{2} \Leftarrow \Pr\left[\forall 1 \le j \le k, \quad h(y_j) \ge \frac{m}{2k}\right] \ge \frac{1}{2}$.

כלומר, האלגוריתם מחזיר מספר שהוא לכל היותר $2k$ בסיכוי לפחות $\frac{1}{2}$ (במילים אחרות, מחזיר מספר שהוא לפחות $2k$ בסיכוי לכל היותר $\frac{1}{2}$).

מהו הסיכוי ש-$a > \frac{m}{k/2}$?

הסיכוי שהאיבר הראשון ממופה לאינדקס בטבלה שגדול יותר מ-$\frac{2m}{k}$ הוא $\left(m - \frac{2m}{k}\right) \cdot \frac{1}{m} = 1 - \frac{2}{k}$.

הסיכוי שכל k האיברים ממופים לאינדקס שגדול מ-$\frac{2m}{k}$ בטבלה הוא $\left(1 - \frac{2}{k}\right)^k$, כאן השתמשנו בכך ש-$\mathcal{H}$ הוא אוסף כל הפונקציות ולכן ההסתברות שמיפינו איברים לטבלה באינדקס שגדול מ-$\frac{2m}{k}$ בלתי תלויה במיפויים קודמים. מכאן נקבל:

$$\left(\left(1 - \frac{1}{k/2}\right)^{k/2}\right)^2 \le \left(\frac{1}{e}\right)^2 < \frac{1}{4}$$

כלומר, האלגוריתם מחזיר מספר קטן מ-$\frac{k}{2}$ בסיכוי לכל היותר $1/4$.

מכאן נסיק שהאלגוריתם מחזיר מספר שהוא בין $k/2$ ל-$2k$ בסיכוי לפחות $1 - \left(\frac{1}{4} + \frac{1}{2}\right) = \frac{1}{4}$. במילים אחרות, הוא קירוב $(2, 3/4)$ לבעיה 12.5.