האוניברסיטה העברית בירושלים | The Hebrew University of Jerusalem

סיכום הרצאות קורס מבני נתונים
Data Structures (67109)

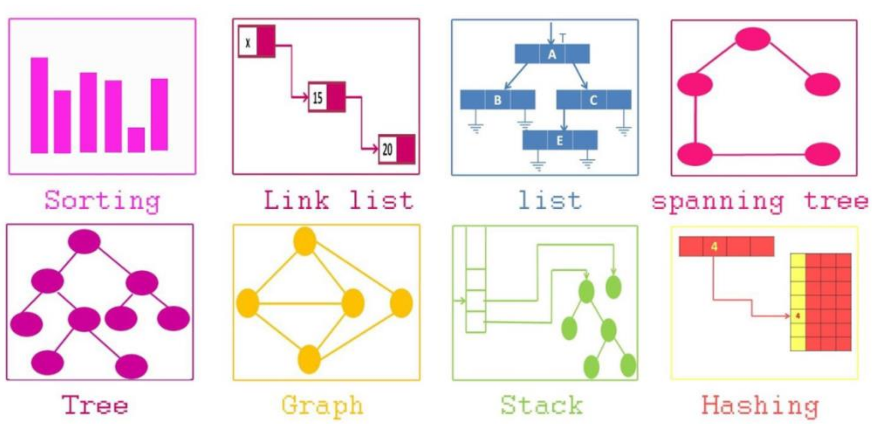
בעקבות הרצאותיה של פרופ' דורית אהרונוב

תשע"ט // 2019 // סמסטר ב'

סוכם על־ידי רועי שטוירמן, בהליך כתיבה ארוך שדרש דם, יזע ובעיקר דמעות (כתוצאה מהבהייה הממושכת במסך). לתלונות, הערות קטנוניות, מחמאות, בדיחות מתכנתים, סיפורים קצרים, ביצועים טובים של המתאוס פסיון ושאלות פילוסופיות עמוקות:

roy.shtoyerman@mail.huji.ac.il

המוצא את כל הלינקים המוחבאים במסמך זה זוכה בשלוש אלפיות נקודה, כבונוס לציון הסופי.



תוכן העניינים (לחיצה על התאריך בכותרת ההרצאות תחזיר לתוכן העניינים.)

גרסה זו של הסיכום מוקדשת לירדן ואן דן ברינק. עודכן לאחרונה ביום שישי 25 אוקטובר 2019.

הרצאה 1: מבוא כללי, מבוא לניתוח זמני ריצה ופתיחת ארבעת הפרשים 10/3/2019

א. מבוא כללי לקורס

ברוכים הבאים לקורס מבני נתונים! מה יהיה בקורס?

א. מבני נתונים (כמובן): ייצוג אוספי נתונים - רשימות, מערכים, גרפים, עצים ועוד מכל טוב.
ב. אלגוריתמים: פונקציות גיבוב, מיון מערכים, חיפוש בהם והוספת נתונים אליהם, מציאת מסלולים בגרפים...
ג. זמני ריצה: ניתוח סיבוכיות של אלגוריתמים שונים.

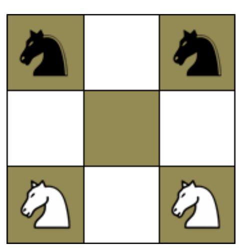
לפירוט נוסף, ראו את הסילבוס המלא באתר.

אופן חישוב הציון

ראו פירוט מלא ומדויק כאן, בקובץ נהלי הקורס.
בקורס זה קיימת חובת הגשה (ומעבר) של כל התרגילים (כ־14) מלבד שלושה. התרגילים יפורסמו בימי רביעי ויוגשו ברביעי שבוע לאחר מכן. כל תרגיל מזכה בציון עובר או לא עובר, ובנוסף ציון על איכות התרגיל (שאינו נכנס לממוצע).
בנוסף עליהם כל סטודנט יעבור שלושה ריאיונות במהלך הסמסטר. על כל מקבץ תרגילים חובה לגשת לריאיון אחד, לכל אחד מהם משקל של 5 נקודות (15% מהציון הסופי). בריאיונות תישאלנה שאלות דומות לאלו שבתרגילים. שאר הנקודות הן של ממוצע המבחנים.
באמצע הסמסטר יתקיים מבחן אמצע (מגן), שיהווה 15% או 0% מציון המבחנים (שמהווה 85% מהציון הסופי), כלומר כ־13% של ציון מגן. יתר הציון הוא כמובן של המבחן הסופי.
בונוס של חצי נקודה לציון הסופי יינתן למי שיגיש לפחות חצי מתרגיליו מוקלדים, בונוס של נקודה למי שיגיש כך את כולם. נוסף על כך יינתן בונוס לארבעת מסכמי ההרצאות הטובים ביותר, ולמי שהשתתף בהן באופן פעיל.

תרגיל מוטיבציה: פתיחת ארבעת הפרשים

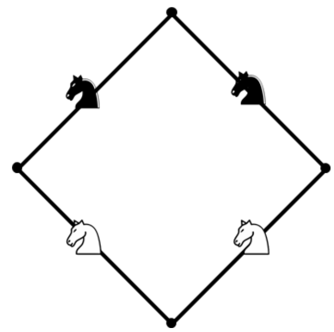
נניח שנתון לוח שחמט בגודל 3×3, שבקצהו האחד שני פרשים שחורים ובצִדו האחר שניים לבנים (ראו שרטוט 1.1). בכמה מהלכי פרש ניתן להחליף את מיקומי הפרשים הלבנים והשחורים? נזכיר כי מסע פרש מורכב משני צעדים בכיוון אחד (אנכי או אופקי), ואחד בכיוון האחר.



שרטוט 1.1: ארבעת הפרשים.

להרחבה נוספת בנושא פתיחה זו, ראו: Halloween Gambit

פתרון: נייצג את הלוח באמצעות גרף, שכל משבצת בו היא קדקוד (מלבד האמצעית, אליה לא ניתן להגיע), וקיימת צלע בין כל שני קדקודים שמסע פרש מקשר ביניהן. גרף כזה ייראה כך:



שרטוט 1.2: ייצוג גרפי של חידת ארבעת הפרשים.

כעת ניתן לראות באופן מידי את שיטת הפתרון: נזיז את הפרשים במעגל ציקלי, וכך נוכל תוך שישה־עשר מהלכים להחליף בין מיקומי הפרשים השחורים והלבנים. כך למעשה אנו רואים ששינוי מבנה ייצוג הנתונים עשוי להשפיע מהותית על קלות פתרון הבעיות. כעת כאשר אנו מלאי מוטיבציה, נוכל להתחיל ללמוד.

ב. מבוא לניתוח אלגוריתמים

מציאת פסגה במערך חד־ממדי

נתחיל עם בעיה פשוטה. נניח ש־A הוא מערך חד־ממדי.[1] את ערכיו נסמן כך:

$$A = [a_1, \dots, a_n] = \big[A[1], A[2], \dots, A[n]\big]$$

שימו לב שבקורס זה אנו ממספרים אינדקסים החל מ־1 ולא מ־0 כנהוג בשפות תוכנה.

נגדיר **"פסגה"** כך:

(i) אם $n = 1$, אז $A[1]$ פסגה.

(ii) $A[n]$ פסגה אם $A[n] \geq A[n-1]$.

(iii) $A[1]$ פסגה אם $A[1] \geq A[2]$.

(iv) $A[i]$ (עבור $1 < i < n$) פסגה אם $A[i-1] \leq A[i]$ וגם $A[i+1] \leq A[i]$.

בניית אלגוריתם נאיבי[2]

לפי הגדרה זו, במערך של n איברים זהים תהיינה n פסגות. כיצד נמצא פסגה? להלן פסודו־קוד[3] לאלגוריתם נאיבי:

---

[1] למוטרדים בענייני לשון עברית, כותבים "ממד" ולא "מימד": בדרך כלל אין כותבים יו"ד לסימון התנועה e. המילים מֵרַב, דֵּקָן ומֵמַד נכתבות ללא היו"ד גם בכתיב חסר הניקוד, כמפורט כאן.

[2] אלגוריתם נאיבי הוא לרוב הפתרון הברור והפשוט ביותר, ולעיתים גם הטיפש ביותר. הוא אלגוריתם שאינו יעיל במיוחד, אך פותר את הבעיה, ובצורה שאיננה מתוחכמת.

[3] קוד שאינו לחלוטין בשפת תוכנה, אך מסביר את עיקר הקוד באמצעות מושגים מן התחום.

```
def find_peak_1(A):
    i = 1
    found = False
    while found == False and i ≤ n:
        if A[i] is a peak:
            return i
        else:
            i += 1
```

האם הקוד עובד? נראה שהאלגוריתם `find_peak_1` מוצא פסגה לכל מערך (קלט) A.

הוכחת נכונות האלגוריתם הנאיבי

אם האלגוריתם עצר בשלב $i < n$, אז $A[i]$ פסגה (זאת מאחר שאנו עוצרים כאשר יש פסגה.) אחרת, הערכים $A[1]$ עד $A[n-1]$ אינם פסגות. נראה שבמקרה זה $A[n]$ פסגה. נשתמש בטענת עזר לשם כך.

<u>טענה 1.1</u>: אם $\forall k < n$ מתקיים שהערכים $A[1], \ldots, A[k]$ אינם פסגות, אז:

$$A[1] < A[2] < \cdots < A[k] < A[k+1]$$

<u>הוכחת טענה 1.1</u>: נראה באינדוקציה.

<u>בסיס האינדוקציה</u>: עבור $k = 1$: אם $A[1]$ איננו פסגה אז $A[1] < A[2]$.

<u>שלב האינדוקציה</u>: נניח נכונות עבור k, נרצה להוכיח עבור $k+1$: אם $A[1], \ldots, A[k+1]$ אינן פסגות, אז מהנחת האינדוקציה:

$$A[1] < A[2] < \cdots < A[k] < A[k+1]$$

וספציפית $A[k] < A[k+1]$, ומאחר ש־ $A[k+1]$ איננו פסגה מתקיים $A[k+1] < A[k+2]$, ובזאת קיבלנו את שרצינו להוכיח. $\mathcal{Q.E.D}$[4]

מטענת עזר זו נוכל להסיק שאם הערכים $A[1]$ עד $A[n-1]$ אינם פסגות, בהכרח $A[n]$ פסגה, כנדרש, אז **בכל מערך חד־ממדי יש פסגה**.

האלגוריתם הנאיבי רץ בזמן של $O(n)$. לא נכתוב כאן הוכחה פורמלית לכך, אבל בנפנופי ידיים: כל שורה בלולאה רצה בזמן $O(1)$, והלולאה רצה n פעמים.

זמן ריצה של אלגוריתם על קלט מסוים A - נסמנו ב־ $T(A)$. אנו עוסקים בסיבוכיות המקרה <u>הגרוע ביותר</u>:

[4] נשתמש בקובץ סיכומים זה בסימון Q. E. D (בלטינית: Quod Erat Demonstrandum, 'מה שהיה להראות'), בהשפעתו של שפינוזה, כתחליף לריבוע השחור הרשע והמפורסם ■ שלעיתים עלול להופיע בסיומה של הוכחה. מועמד מוביל נוסף היה W^5 (כקיצור ל־ Which Was What We Wanted).

$$T(n) = \max\left\{T(A) \;\middle|\; |A| = n\right\}$$

כעת נרצה להגדיר באופן רשמי, פורמלי ומתמטי את החסמים לזמני הריצה.

ג. הגדרת הסימונים האסימפטוטיים

כעת ניתן הגדרות מדויקות יותר למה שנלמד בקורס מבוא למדעי המחשב (67101).

1. סימון O-גדול – Big-O Notation: חסם אסימפטוטי עליון

הגדרה 1.2: תהיינה שתי פונקציות $f, g: \mathbb{N} \to \mathbb{R}^+$. נאמר ש־ $f(n) = O\big(g(n)\big)$ אם קיימים $\exists c > 0$ ו־ $\exists n_0 \in \mathbb{N}$ כך ש־ $\forall n > n_0:\ f(n) \le c \cdot g(n)$.

מה ההגדרה אומרת? אם "אחרי" מספר מסוים n_0, קיימת כפולה של הפונקציה $g(n)$ שגדולה או שווה לערך של $f(n)$, אז $f(n) = O\big(g(n)\big)$.[5]

דוגמה:

$$g(n) = n, \qquad f(n) = 100n$$

האם $f(n) = O\big(g(n)\big)$?

תשובה: **כן.** ניקח $c = 100$, ואז לכל $n \in \mathbb{N}$ מתקיים $f(n) \le 100g(n)$.

דוגמה נוספת:

$$g(n) = n^2 - 100, \qquad f(n) = n$$

האם $f(n) = O\big(g(n)\big)$?

תשובה: **כן.** עלינו להראות שקיים n_0 שעבורו $\forall n > n_0:\ n + 100 < n^2$. כיצד נעשה זאת? נוכל להראות שהפרבולה $n^2 - n - 100$ חיובית לאחר n מסוים; נוכל להראות שהגבול שלה כאשר $n \to \infty$ הוא ∞ באמצעות גבולות כמו שלמדנו באינפי 1, או בכל שיטה אחרת שמוצאת חן בעיניכם! לא ניתן פתרון מפורש בהרצאה, אך תעבדו על הוכחות מורכבות מזו בתרגיל הבית.

2. סימון Ω – Big Omega Notation: חסם אסימפטוטי תחתון

הגדרה 1.3: כמו קודם, תהיינה $f, g: \mathbb{N} \to \mathbb{R}^+$. נאמר ש־ $f(n) = \Omega\big(g(n)\big)$ (f "אומגה של" g) אם קיימים $c > 0$ ו־$n_0 \in \mathbb{N}$ כך שעבורם $\forall n > n_0:\ f(n) \ge c \cdot g(n)$.

מה ההגדרה הזו אומרת? בדיוק את ההיפך מ־O-גדול: כעת אנחנו חוסמים אסימפטוטית מלמטה.

[5] אנו משתמשים בסימון השוויון, אך גם נפוץ (ומדויק יותר) הסימון $f(n) \in O\big(g(n)\big)$, כלומר ש־$f(n)$ שייכת לקבוצת הפונקציות ש־$g(n)$ מהווה להן חסם O-גדול.

<u>דוגמה</u>: האם מתקיים:

$$g(n) = \Omega\big(f(n)\big) \Leftrightarrow f(n) = O\big(g(n)\big)$$

<u>תשובה</u>: **כן.** להלן הוכחה מהירה (הלקוחה מתרגיל הבית):

$$f(n) = O\big(g(n)\big) \Leftrightarrow$$
$$\exists c > 0, n_0 \in \mathbb{N}:\ \forall n > n_0:\ f(n) < c \cdot g(n) \Leftrightarrow$$
$$\forall n > n_0:\ \ g(n) > \frac{1}{c} f(n) \Leftrightarrow$$
$$g(n) = \Omega\big(f(n)\big)$$

Q.E.D

3. סימון Θ – Theta Notation: חסם הדוק

בקורס מבוא למדעי המחשב (67101), כאשר נשאלנו מה זמן הריצה של פונקציה, היינו משיבים תשובות מהסוג $O(n)$, אך למעשה זהו רק חסם עליון. כאשר אנו מדברים על $\Theta\big(f(n)\big)$ אנו מדברים על חסם עליון ותחתון גם יחד, אשר מהווה חסם הדוק לזמן הריצה של הפונקציה.

<u>הגדרה 1.4</u>: נאמר ש־$f = \Theta\big(g(n)\big)$ אם מתקיים $f(n) = \Omega\big(g(n)\big)$ וגם $f(n) = O\big(g(n)\big)$.

למה כל ההגדרות האלו מועילות לנו? בעזרתן נוכל לנתח את זמן הריצה של אלגוריתמים פשוטים בהרבה מהאלגוריתמים הממשיים, שאנו יודעים שלהם זמן ריצה דומה.

ד. <u>חזרה לבעיית מציאת פסגה במערך חד־ממדי</u>

כעת נחזור לאלגוריתם `find_peak_1`, וננסה לשפרו:

```
def find_peak_2(A, n):
    i = ⌈n/2⌉
    if A[i] is peak:
        return i
    else:
        if n ≥ i+1 find_peak_2(A[i+1,…,n],n-i)
```

הוכחת נכונות האלגוריתם `find_peak_2`

נוכיח באינדוקציה על n.

<u>בסיס האינדוקציה</u>: עבור $n = 1$ נקבל ש־$A[1]$ הוא פסגה כנדרש.

<u>צעד האינדוקציה</u>: נניח נכונות לקלטים שקטנים מ־n, ונוכיח עבור n.

עבור $n > 1$: או שמצאנו פסגה בבדיקה הראשונה, ואז סיימנו. אחרת קראנו לאלגוריתם על מערך קטן יותר, ומהנחת האינדוקציה נמצא שם פסגה, מש"ל.

ההוכחה הנ"ל כמובן שגויה, והאלגוריתם לא עובד. נניח שיש לנו מערך שרק יורד, הקוד יחזיר את הנקודה הנמוכה ביותר, ולא את הפסגה.

מה הטעות? השתמשנו במילה "פסגה" באופן שגוי: הוא תלוי במיקום שבו אנו נמצאים. במערך המתואר, שיורד כל הזמן, הנקודה שנחשבת פסגה במערך קטן יותר, אינה פסגה במערך כולו!

ועכשיו באמת: ננסה שוב

אז כיצד נוכל לשפר את האלגוריתם זאת? היזכרו במבחן מועד א' באינטרו של הסמסטר הקודם... עלינו לבדוק מה הצד שמכיל את הערך הגדול יותר מנקודת האמצע, ואז לבחון את המקטע לשמאלה או לימינה של הנקודה i, בהתאם. נתקן הקוד בזריזות ובנחישות:

```
def find_peak_3(A, n):
    i = ⌈n/2⌉
    if A[i] is peak:
        return i
    else if A[i+1] > A[i]:
        return find_peak_3(A[i+1,…,n],n-i)
    else:
        return find_peak_3(A[1,…,i-1],i-1)
```

אנו מצפים לקבל זמן ריצה לוגריתמי. נוכיח את הסיבוכיות.

ניתוח סיבוכיות

__**טענה 1.5**__: נטען שלכל A, כאשר $|A| = n$, `find_peak_3(A, n)` רץ בזמן לוגריתמי:

$$T(n) = O(\log n)$$

__**הוכחת טענה 1.5**__: נוכיח באינדוקציה: $T(n) \leq 10\log(n) + 10$, לכל $n \geq 1$.

<u>בסיס האינדוקציה</u>: עבור מקרה בסיס $n = 1$: המקרה ברור - זמן הריצה הוא $O(1)$.

<u>צעד האינדוקציה</u>: עבור $n > 1$: נניח שמתקיים עבור כל $n' < n$ ונוכיח עבור n.

$$T(n) \leq 10 + T\left(\left\lfloor\frac{n}{2}\right\rfloor\right) \underset{\text{הנחת האינדוקציה}}{\leq} 10 + 10\log\left(\left\lfloor\frac{n}{2}\right\rfloor\right) + 10 \leq 20 + 10\log_2\left(\frac{n}{2}\right)$$

$$= 20 + 10\log_2(n) - 10 = 10 + 10\log_2(n)$$

Q.E.D

למה בזאת סיימנו את ההוכחה? נרצה להוכיח $T(n) \leq 20\log n$. ניתן להוכיח ישירות, אך ניתן גם להראות $T(n) = O(\log n)$, לפי העיקרון שנוכיח בהרצאה הבאה - שקובע מתי ניתן להזניח פונקציות בעת חישוב החסמים האסימפטוטיים.

<u>הרצאה 1 הגיעה לסיומה!</u>

הרצאה 2: סימון o-קטן ו־ω-קטנה, אלגוריתמים ממיינים, שיטת האב

17/3/2019

א. המשך הסימונים האסימפטוטיים

לצד הסימונים היפהפיים O, Θ, Ω, יש גם שימוש בחסמים o ו־ω, אותם נגדיר עתה.

4. סימון o-קטן – Little-o Notation: חסם עליון ממש

תזכורת: **גם לכיף יש גבול!**

נאמר ש־ $\lim\limits_{n\to\infty} h(n) = 0$ אם לכל $\varepsilon > 0$ קיים n_0 כך ש־ $\forall n > n_0\colon\ |h(n)| < \varepsilon$.

הגדרה 2.1: תהיינה שתי פונקציות, $f, g\colon \mathbb{N} \to \mathbb{R}^+$. נאמר ש־ $f(n) = o\big(g(n)\big)$ אם **לכל** $c > 0$ קיים $n_0 \in \mathbb{N}$, כך שלכל $n > n_0$ מתקיים $f(n) < c \cdot g(n)$. או באופן שקול:

$$\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$$

המשמעות היא ש־$\boldsymbol{f(n)}$ **זניחה ביחס ל־**$\boldsymbol{g(n)}$ (עבור n גדול).

דוגמה:

$$g(n) = 10\log n\,, \qquad f(n) = 10$$
$$\Downarrow$$
$$f(n) = o\big(g(n)\big)$$

טענה 2.2:

$$f(n) = O\big(h(n) + g(n)\big), \qquad h(n)\ \text{חיובית}, \qquad g(n) = o\big(h(n)\big)$$
$$\Downarrow$$
$$f(n) = O\big(h(n)\big)$$

במילים: כל פונקציה שמוסיפים שזניחה לפונקציה "עיקרית", ניתן להתעלם ממנה עבור n גדול. אותה הטענה תקפה גם עבור Θ ו־Ω. כך נוכל לראות בקלות מה הסיבוכיות של אלגוריתמים מסוימים, באמצעות הזנחה של הפונקציות ה"שוליות".

הוכחת טענה 2.2:

נקצר מעט את ההליכים, ובמקום להסתבך עם אפסילונים, דלתאות ושאר חייזרים נכתוב בקיצור נמרץ:[6]

$$f(n) \underset{\substack{\text{החל} \\ n\text{־מ} \\ \text{מסוים}}}{\leq} c\big(h(n)+g(n)\big) \underset{\substack{\text{החל} \\ n\text{־מ} \\ \text{מסוים}}}{\leq} 2ch(n)$$

Q.E.D

5. סימון ω-קטנה – Little-ω Notation: חסם תחתון ממש

באופן דומה והפוך ל־o-קטן, נוכל להגדיר חסם תחתון ממש (לא הוגדר בהרצאה).

__**הגדרה 2.3**__**:** תהיינה $f, g: \mathbb{N} \to \mathbb{R}$ שתי פונקציות. נאמר ש־$f(n) = \omega\big(g(n)\big)$ אם **לכל** $c > 0$ קיים $n_0 \in \mathbb{N}$ שעבורו $f(n) > c \cdot g(n)$. באופן שקול:

$$\lim_{n\to\infty}\left|\frac{f(n)}{g(n)}\right| = \infty$$

ובמקרה זה, **$g(n)$ זניחה יחסית ל־$f(n)$.**

__**דוגמה:**__

$$f(n) = 2^n, \qquad g(n) = n^2$$
$$\Downarrow$$
$$f(n) = \omega\big(g(n)\big)$$

__**טענה**__**:** (למעשה, טענתיים[7]):

$$f(n) = o\big(g(n)\big) \Rightarrow f(n) = O\big(g(n)\big) \tag{2.4}$$
$$f(n) = \omega\big(g(n)\big) \Rightarrow f(n) = \Omega\big(g(n)\big) \tag{2.5}$$

הטענות נובעות מידית מן ההגדרה. ההיפך כמובן אינו נכון (נראה בתרגיל).

__**לסיכום חמשת החסמים האסימפטוטיים**__:

1. **O-גדול**: חסם אסימפטוטי עליון על זמן הריצה.
2. **Ω-גדולה**: חסם אסימפטוטי תחתון על זמן הריצה.
3. **Θ-גדולה**: חסם אסימפטוטי הדוק על זמן הריצה.
4. **o-קטן**: חסם אסימפטוטי עליון ממש: הפונקציה לא יכולה לרוץ בסיבוכיות הזהה לחסם העליון ה־o-קטן שלה.
5. **ω-קטנה**: חסם אסימפטוטי תחתון ממש: הפונקציה לא תרוץ בסיבוכיות הזהה לחסם התחתון ω-קטנה שלה.

---

[6] למקרה שחשדתם, הביטוי "בקיצור נמרץ" לקוח, כמובן, מהז'רגון העשיר והנפלא של ד"ר איב גודין.

[7] רגע נוסף של עברית: צורת הזוגי בעברית (כמו בערבית) נבנית על היחיד, לכן "טענתיים" ולא "טענותיים", כמו נְקֻדָּתַיִם ולא נְקֻדּוֹתַיִם, מיטת קוֹמָתיים ולא מיטת קומותיים.

שימו לב שאין הבדל בין θ-קטנה ו־Θ-גדולה - בניגוד לשמלות, בחסמים אסימפטומטיים אין הבדל בין הדוק והדוק ממש! אם חסם הוא הדוק, אז הוא בהכרח שווה לסיבוכיות הריצה של הפונקציה שהוא חוסם!

ב. אלגוריתמים ממיינים

(i) מיון בועות (Bubble Sort)

ראינו אלגוריתם מיון זה כבר במבוא למדעי המחשב.
נכתוב פסודו־קוד של פונקציה שמבצעת "בעבוע" אחד, כלומר עוברת פעם אחת על המערך ומפעפעת את הערך הגדול ביותר ימינה:

```
def bubble(A[1, ..., n]):
    for i = 1, … ,n-1:
        if A[i] > A[i+1]:
            swap them
```

הפונקציה bubble רצה בסיבוכיות של $\Theta(n)$ - לא משנה מה גודל המערך ומה בתוכו, היא תמיד תרוץ מתחילתו עד סופו. כעת נכתוב את הפונקציה הממיינת כולה:

```
def bubble_sort(A[1,...,n]):
    bubble(A[1,...,n])
    bubble_sort(A[1,...,n-1])
```

הוכחת נכונות האלגוריתם מיון בועות (Bubble Sort)

נוכיח נכונות עבור bubble, כלומר נוכיח את הטענה הזו:

<u>טענה 2.6</u>: בסוף ריצת bubble, התא ה־n מכיל ערך מקסימלי (במערך).

<u>הוכחת הטענה 2.6</u>: נוכיח באינדוקציה, או ליתר דיוק באמצעות שמורת לולאה. כלומר, נראה שיש איזושהי למה[8] שנשמרת לאורך כל הלולאה, לאחר כל מספר של צעדים בה. שמורת הלולאה במקרה זה אומרת שאחרי i צעדים בלולאה מתקיים $A[j] \leq A[i+1]$ לכל $j < i+1$. עתה נוכיח אותה.

<u>בסיס האינדוקציה</u>: עבור $i = 1$, אם $A[1] > A[2]$ אז מתבצעת החלפה.

<u>שלב האינדוקציה</u>: נניח נכונות עבור i, ונגרור מכך את הנכונות עבור $i+1$. אחרי i צעדי לולאה, $A[i+1]$ גדול או שווה לכל קודמיו. בשלב ה־$i+1$, אם $A[i+1] > A[i+2]$ אזי מחליפים ביניהם, כיוון שבתא ה־$i+2$ נמצא האיבר המקסימלי. Q.E.D

[8] דגש חשוב: יש להבחין בין לָמָה ובין לֶמָה. כאשר יש לכם לָמָה שנשמרת לכל אורך ההוכחה, המצב רע מאוד, כי בהמות דרום אמריקאיות לרוב אינן מבינות הרבה במדעי המחשב. מאידך גיסא, שמירה על לֶמָה לאורך ההוכחה עשויה להועיל מאוד בהוכחת הנכונות. לשיקולכם!

<u>טענה 2.7</u>: נטען על־פי זאת שהאלגוריתם `bubble_sort` נכון, כלומר שבסיומו המערך ממוין.

<u>הוכחת טענה 2.7</u>: נוכיח באמצעות שמורת הלולאה הזו:

<u>טענה 2.8: שמורת הלולאה:</u> אחרי i הרצות של הלולאה, מתקיים:

א. כל i האיברים האחרונים שבמערך ממוינים בסדר עולה (חלש).

ב. כל $n-i$ האיברים השמאליים במערך קטנים או שווים ל־ $A[n-i+1]$.

נשים לב שאם שמורת הלולאה נכונה, אז כאשר $i=n$, המערך ממוין כנדרש.

<u>הוכחת טענה 2.8:</u>

<u>בסיס האינדוקציה</u>: עבור $i=1$: נובע ישירות מהוכחת הנכונות של `bubble`.

<u>שלב האינדוקציה</u>: נניח נכונות עבור i, ונראה שבהכרח הטענה נכונה עבור $i+1$. נראה שאחרי הריצה ה־$i+1$, כל $n-i-1$ האיברים האחרונים ממוינים וגדולים או שווים לכל אלו שלשמאלם.

א. נובע מכך שהוספנו איבר קטן או שווה לאלו הממוינים כבר, מצד שמאל.

ב. נובע מנכונות של `bubble` ומהנחת האינדוקציה (שמורת הלולאה). *Q.E.D*

שרטוט 2.1: הדגמה לשמורת הלולאה במערך של שמונה איברים.

ניתוח סיבוכיות האלגוריתם מיון בועות (Bubble Sort)

<u>טענה 2.9</u>: סיבוכיות זמן הריצה של Bubble Sort היא $\Theta(n^2)$.

<u>הוכחת טענה 2.9</u>: נניח שזמן הריצה של האלגוריתם על מערך שלם שגודלו n אורך זמן $T(n)$. בכל שלב אנחנו מבצעים את `bubble` שאורך זמן של $\Theta(n)$, ואז מבצעים את T על מערך קטן ב־1. מכאן נקבל את נוסחת הנסיגה (נוסחת רקורסיה) עבור האלגוריתם:

$$T(n)=\Theta(n)+T(n-1),\qquad T(1)=\Theta(1)$$

נכניס קבועים, על־פי $\Theta(n)$, כדי לפשט את הנוסחה:

$$T(n)\le cn+T(n-1)\le cn+c(n-1)+T(n-2)\le$$

$$\le cn+c(n-1)+c(n-2)+\cdots+c\cdot 3+c\cdot 2+T(1)\le$$

$$\underset{(1)}{\leq} c\frac{n(n+1)}{2} = \frac{c(n^2+n)}{2} \underset{(2)}{=} \Theta(n^2)$$

שימו לב שבמעבר (1) השתמשנו בנוסחת הסכום של סדרה חשבונית:

$$1+2+3+\cdots+(n-1)+n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2} \tag{2.10}$$

וסימן השוויון האחרון (2) נובע מ־(2.1), לפיו ניתן להזניח את החזקות הנמוכות של n (ההוכחה הפורמלית ברגיל הבית). שימו לב, הראינו $T(n) \leq f(n) = \Theta(n^2)$, כלומר $T(n) = O(n^2)$. נוכל באותו האופן לחסום את $T(n)$ מלמטה ולהגיע לכך ש־$T(n) \geq \Theta(n^2)$:

$$T(n) \geq c'n + T(n-1) \geq c'n + c'(n-1) + T(n-2) \geq \cdots \underset{(2.8)}{\geq} \frac{c'(n^2+n)}{2} = \Theta(n^2)$$

ומכאן:

$$T(n) = \Omega(n^2)$$
$$\Downarrow$$
$$T(n) = \Theta(n^2)$$

Q.E.D

סכנה!

זהירות לא לטעות פה! נניח שישנה נוסחת רקורסיה כזו:

$$T(n) = T(n-1) + \Theta(n), \qquad T(1) = 1$$

נוכיח (את הטענה השגויה) $T(n) = \Theta(n)$. נניח באינדוקציה:

$T(n-1) = \Theta(n-1)$, ונוכיח עבור n:

$$T(n) = T(n-1) + \Theta(n) = \Theta(n-1) + \Theta(n) = \boldsymbol{\Theta(n)}$$

איפה הטעות? עלינו להשתמש בקבועים ולא בסימונים אסימפטוטיים.

(ii) מיון מיזוג (Merge Sort)

נניח שנתון מערך A בגודל שהוא חזקה שלמה של 2: $|A| = 2^m = n$, מטעמי נוחות.

הפסודו־קוד עבור אלגוריתם ממיין זה הוא כדלהלן:

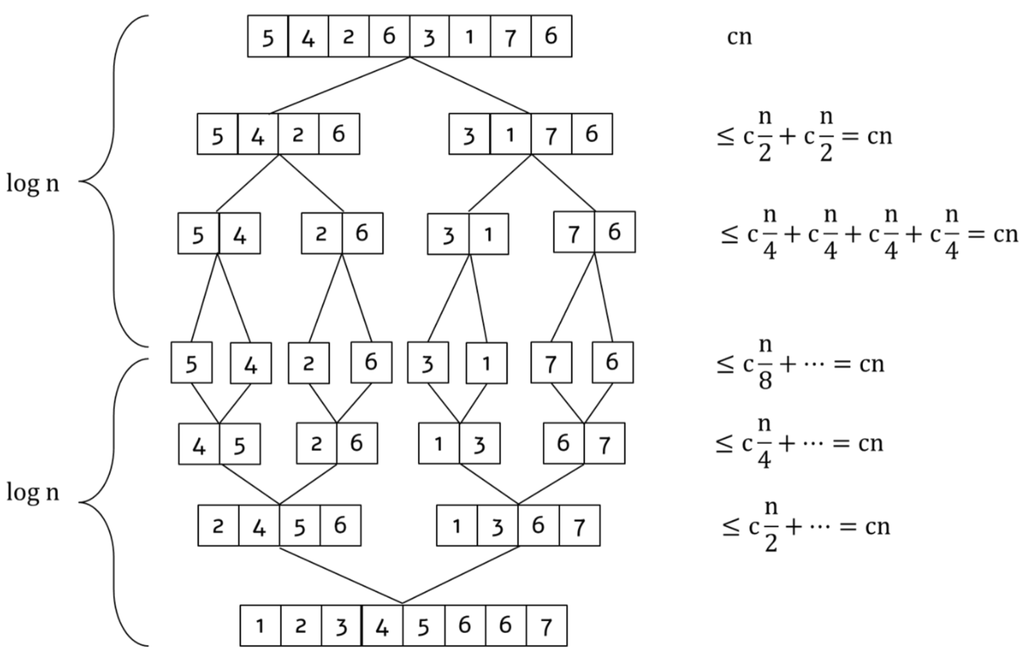
```
def merge_sort(A[1,....,n]):
    if n > 1:
```

```
        merge_sort(A[1,...,n/2])
        merge_sort(A[n/2 + 1,...,n])
        merge(A[1,...,n/2], A[n/2 + 1,...,n])

def merge(A, B, C, n):
    l = 1
    i = 1
    j = 1
    while i <= n/2 or j <= n/2:
        C[l] = min{A[i], B[j]}
        if C[l] == A[i]:
            i += 1
        else:
            j += 1

      l += 1
```

<u>הוכחת נכונות</u>: משאירים לתרגיל.

<u>טענה 2.12: ניתוח סיבוכיות</u>: האלגוריתם Merge Sort פועל בסיבוכיות זמן של $\Theta(n\log n)$.

<u>הוכחת טענה 2.12</u>: נצייר לאלגוריתם את עץ הרקורסיה:



שרטוט 2.2: עץ הרקורסיה של Merge Sort.

במקרה זה, האלגוריתם מבצע עבודה שווה בכל שלב: מספר איברי הרשימה נשאר זהה בכל שלב. מאחר שיש $2\log n$ שלבים ובכל שלב העבודה היא מסיבוכיות $\Theta(n)$, האלגוריתם רץ בסיבוכיות זמן של $\Theta(n\log n)$.

האלגוריתם הממיין הזה הוא דוגמה למקרה כללי של אלגוריתמים שפועלים באמצעות חילוק הבעיה לבעיות פשוטות יותר, ומיזוג של הפתרונות הקטנים לכל התת־בעיות לכדי פתרון שלם. נבחן את המקרה של אלגוריתם "הפרד ומשול" שכזה.

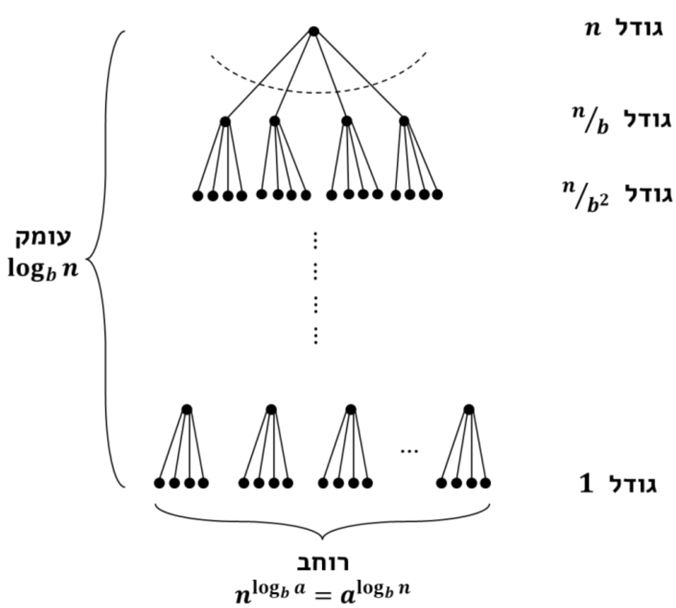
ג. משפט האב (Master Theorem)

נניח שיש לנו אלגוריתם "הפרד ומשול" כללי, שמקבל קלט מגודל n. נניח גם הפעם, מטעמי נוחות, $n=b^m$ מטעמי נוחות. האלגוריתם מייצר a בעיות קטנות יותר, כשגודל הבעיות החדשות הוא $\frac{n}{b}$ כל אחת,[9] ושכמות העבודה בלחבר את כל הבעיות הללו היא $\Theta(n^k)$. נוסחת הרקורסיה המתאימה היא:

$$T(n) = \underbrace{a\,T\left({}^{n}/_{b}\right)}_{\text{כמות העבודה בגלל הרקורסיה}} + \underbrace{\Theta(n^k)}_{\text{העבודה בכל שלב}}$$

$$T(1) = \Theta(1)$$

נצייר עץ רקורסיה חדש באותו האופן:



שרטוט 2.3: עץ רקורסיה של אלגוריתם "הפרד ומשול" כללי

נסכום את העבודה הכוללת:

[9] אין הכרח ש־ $b=a$. ייתכן גודל הבעיה קטן בכל שלב בפקטור b, בלי קשר למספר הבעיות.

$$cn^k + cn^k\left(\frac{a}{b^k}\right) + cn^k\left(\frac{a}{b^k}\right)^2 + \cdots + cn^k\left(\frac{a}{b^k}\right)^m$$

כלומר, קיבלנו חסם של סדרה הנדסית, ומשפט זה נקרא משפט המאסטר, משפט האב:

$$\boldsymbol{T(n) \le cn^k \sum_{j=0}^{m} \left(\frac{a}{b^k}\right)^j} \qquad \textbf{(2.13):}$$

שלושת המקרים של משפט האב

<u>טענה 2.14: משפט האב</u>: נסמן $q \overset{\text{def}}{=} \frac{a}{b^k}$, אז פתרון נוסחת הנסיגה $T(n) = a\,T\left({}^{n}/_{b}\right) + \Theta\left(n^k\right)$ מתחלק לשלושה מקרים:

<u>1.</u> $\boldsymbol{q = 1}$**:**

זהו המצב, לדוגמה, במקרה של האלגוריתם merge sort, כאשר כמות העבודה בכל קריאה (בכל עומק/שורה בעץ הרקורסיה) קבועה. זה המקרה הקל:

$$T(n) \le cn^k(\log n + 1)$$

ובאופן כללי:

$$\boldsymbol{T(n) = \Theta\left(n^k \log_b n\right)}$$

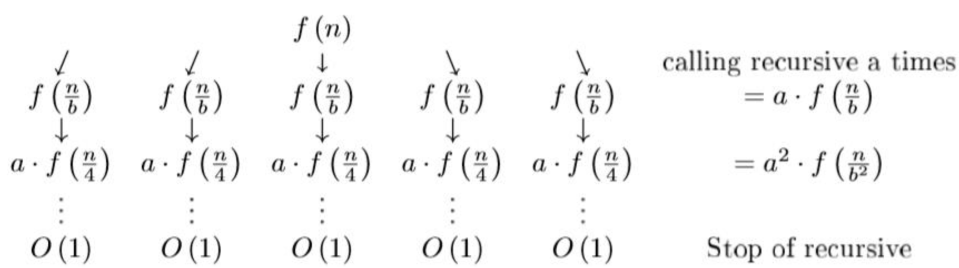
<u>2.</u> $\boldsymbol{q > 1}$**:**

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

<u>3.</u> $\boldsymbol{q < 1}$**:**

$$\boldsymbol{T(n) = \Theta\left(n^k\right)}$$

הרחבה נוספת בעניין משפט זה - משפט האב המורחב - נראה בתרגול.



שרטוט 2.4: עץ רקורסיה נוסף של אלגוריתם "הפרד ומשול" כללי.

<u>הרצאה 2 הגיעה לסיומה!</u>

הרצאה 3: מיון מהיר (Quick Sort), אלגוריתמים אקראיים ופתיח לעולם ההסתברות

24/3/2019

א. חזרה על משפט האב

נזכיר את משפט האב: הוא עסק באלגוריתמים מסוג "הפרד ומשול", בעל נוסחת רקורסיה כזו:

$$T(n) = aT\left({}^{n}\!/_{b}\right) + \Theta\left(n^k\right)$$

וחילקנו לשלושה מקרים, לפי כמות העבודה היחסית, כאשר הגדרנו $q \stackrel{\text{def}}{=} \frac{a}{b^k}$:

1. **אם $q < 1$**, כלומר כמות העבודה היחסית קטנה בכל שלב (אזי השלב הראשון הוא המשמעותי ביותר):

$$\boldsymbol{T(n) = \Theta\left(n^k\right)}$$

2. **אם $q = 1$**, כמו ב־Merge Sort, כלומר שכמות העבודה קבועה שכל שלב:

$$\boldsymbol{T(n) = \Theta\left(n^k \log_b n\right)}$$

3. **אם $q > 1$**, כלומר העבודה היחסית גדלה בכל שלב:

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

ב. האלגוריתם Quick Sort

ראינו בקורס המבוא את הגרסה האקראית של האלגוריתם quick sort. הרעיון של אלגוריתמים אקראיים היה לא פחות מפורץ דרך במדעי המחשב, והתחיל באלגוריתם של מיכאל רבין למציאת מספרים ראשוניים.

כיצד quick sort עובד? הוא בוחר ציר שרירותי,[10] נקרא לו בשם המקורי והיפהפה pivot (או, איבר ציר). לאחר מכן הוא מסדר את כל האיברים הגדולים מ־pivot בצד ימין, ואת כל האיברים הקטנים ממנו בצד שמאל. לאחר מכן האלגוריתם קורא לעצמו רקורסיבית על המערך של הערכים מימין ומשמאל.

מטעמי פשטות, נניח שכל הערכים במערך שונים. להלן פסודו־קוד לאלגוריתם (A הוא המערך, R מקצר "Right", L מקצר "Left"):

---

[10] שרירותי: שאין שום סיבה לבחור **דווקא** אותו, אך אין זה אומר שהוא נבחר אקראית! כלומר, איננו מוגרל. נניח, ניתן לבחור תמיד את האמצע, או במקרה זה, את הקצה הימני.

```
def Quicksort(A, L, R):
    if L < R: # sort only in case of two or more values
       m = partition(A, L, R)
       Quicksort(A, L, m-1)
       Quicksort(A, m+1, R)

def partition(A, L, R):
    pivot = A[R]    # זו בחירה שרירותית: הקצה הימני
    i = L - 1
    j = R
    while true:
       while A[i] < pivot:
             i++
       while A[j] > pivot:
             j--
       if i ≥ j:
          break loop
       else:
          swap A[i] and A[j]
    swap A[i] and A[R]
    return i
```

מה הסיבוכיות של partition? לא פורמלית מדי: $\Theta(n)$, מכיוון שאנו רצים על כל המערך.

הוכחת הנכונות של Quick Sort

נרצה להוכיח את נכונותו של המיון המהיר. נרצה לבחור שמורת לולאה, אך נתקשה לעשות זאת מאחר שאין פה לולאה, אלא רקורסיה. נשתמש במקום בטענה זו: ניתן לראות (ולא נוכיח כעת) שבסיום partition מתקיים:

$$\{A[1], \dots, A[m-1]\} \leq A[m] \leq \{A[m+1], \dots, A[n]\} \tag{3.1}$$

<u>טענה 3.2: נכונות המיון המהיר</u>: נראה ש-Quick Sort אכן מחזיר מערך ממוין. נוכיח באינדוקציה מלאה.

<u>בסיס האינדוקציה</u>: טריוויאלי - עבור $n=1$ המערך כבר ממוין.

<u>שלב האינדוקציה</u>: נניח ש-quick sort ממיין כהלכה לכל מערך בגודל m שקטן מ-n, ונוכיח עבור n.

לפני הנחת האינדוקציה: $A[1], \dots, A[m-1]$ ממוין, כלומר:

$$A[1] \leq A[2] \leq \cdots \leq A[m-1]$$

וכן:

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

מנכונות partition (משוואה 3.1):

$$A[1], \ldots, A[m-1] \leq A[m]$$
$$A[m] \leq A[m+1], \ldots, A[n]$$

אזי:

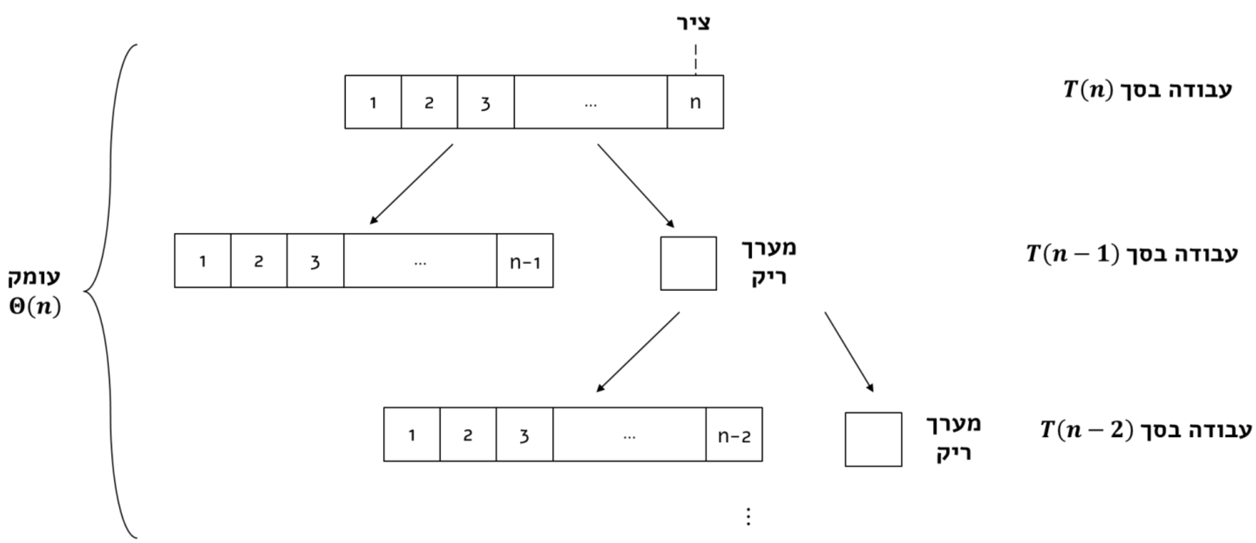
$$A[1] \leq A[2] \leq \cdots \leq A[n]$$

Q.E.D

סיבוכיות אלגוריתם המיון המהיר (Quick Sort)

<u>המקרה הגרוע ביותר</u>

נצייר עץ רקורסיה למקרה הגרוע ביותר (עוד נדבר על הסיבות לכך שהוא הגרוע ביותר, וגם כיצד לפתור את זה). נניח שיש מערך שכבר ממוין (בסדר עולה):



שרטוט 2.5: עץ רקורסיה של Quick Sort של המקרה הגרוע ביותר.

(מדוע אנו מניחים תמיד שהקלט הוא הגרוע ביותר? ייתכן שאנשי מדעי המחשב הם סתם פסימיסטיים.

ובהקשר זה נצטט את האבות הרוחניים של Python: Always look on the bright side of life!)

למה הוא מחלק מערך ממוין כך? כי partition לוקח את המערך, מסדר מצד שמאל את כל הערכים הקטנים מהאיבר הימני, ומצד ימין את כל הגדולים ממנו (ואת האיבר עצמו שומר כשלעצמו): במקרה זה הוא פשוט יוציא החוצה את n, ויקרא ל־quick sort על המערך השמאלי בלי הערך n. כך שעומק עץ הרקורסיה הוא $\Theta(n)$, ובשלב ה־i, שבו גודל המערך הוא $k = n - i$ הוא מבצע עבודה בסך $T(k)$. כלומר במקרה הגרוע במיוחד הזה, נוסחת הרקורסיה היא:

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

וכבר פגשנו נוסחת רקורסיה שכזו עבור bubble sort, מבט מהיר במשפט האב ייתן שמתקיים:

$$T(n) = \Theta(n^2)$$

וזה כבר לא כל כך מהיר כפי שמציעה הכותרת.

<u>המקרה הטוב ביותר</u>

ננחש מה יהיה המקרה הטוב ביותר (לא נוכיח זאת כעת - למעשה רק בעוד שתי הרצאות). ניחוש טוב לסיבוכיות של T יהיה:

$$T(n) = O(n^2)$$
$$T(n) = \Omega(n \log n)$$

ניתן לראות שבכל שלב (בכל "קומה") סך כל העבודה היא לכל היותר $O(n)$, ומספר הקומות קטן או שווה ל־n, ולכן אם נסכום עבור כל המערך נקבל

$$T(n) = O(n^2)$$

אנו רוצים להראות שבאופן אופייני נקבל שבמערך אקראי האלגוריתם ירוץ בזמן הרבה יותר קצר מזה. נצטרך לבצע אנליזה סטטיסטית כלשהי. נניח שמובטח שתמיד החלוקה תיעשה לכדי עשירית ותשע עשיריות. במקרה זה נוסחת הנסיגה המתאימה היא:

$$T(n) = T\left(\frac{n}{10}\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

כיצד פותרים נוסחת נסיגה כזו? זו לא צורה של משפט האב, אז ננסה להניח ש־T מונוטונית ונעריך בצורה גסה:

$$\dots \le 2T\left(\frac{9n}{10}\right) + \Theta(n)$$

במקרה זה $b = \frac{10}{9}$, $a = 2$, $k = 1$, אז $q = \frac{18}{10}$. מכאן לפי משפט האב:

$$T(n) = O\left(n^{\log_{\frac{10}{9}} 2}\right) \approx O(n^{6.58})$$

אך חסם זה **ממש גרוע!** נסתכל על העץ: גובה העץ חסום על־ידי $\log_{\frac{10}{9}} n$, מאחר שבכל פעם אנו מחלקים את הקלט באורך n לשני חלקים ביחס 1:9. ולכן:

$$c \cdot n \log_2 n = n \log_{\frac{10}{9}} n \ge T(n) \Longrightarrow$$
$$T(n) = O(n \log n)$$

שימו לב שקיים הבדל גדול בין דיון במערך מקרי, שבו יש הסתברות גבוהה מאוד שהחלוקה תהיה יחסית מאוזנת (נניח, ביחס 1:3 או 1:4), לבין דיון באלגוריתם שיעבוד ב־$\Theta(n \log n)$ לכל קלט. וגם כזה אלגוריתם אפשרי! לשם כך עלינו להיכנס מעט לעולם ההסתברות.

ג. <u>פתיח להסתברות ותהליכים אקראיים</u>

<u>שאלה:</u> נניח שנתון מערך A שגודלו n, ומובטח שמתקיימים אחד משני המקרים לגביו:

(מקרה ראשון) $\forall i: A[i] = 1$

או:

(מקרה שני) $\begin{cases} A[i] = 0, & \text{בחצי מהמקרים} \\ A[i] = 1, & \text{בחצי מהמקרים} \end{cases}$

לאיזה משני המקרים המערך A שייך? כיצד נוכל לקבוע זאת מהר?

בכל מקרה, ניתן למצוא מערך כך ש-$\frac{n}{2}$ הערכים הנבדוק יהיו 1, והערך $\frac{n}{2}+1$ יהיה 0, כך שתמיד במקרה הגרוע ביותר נאלץ לבצע $\frac{n}{2}+1$ בדיקות.

אולם נרצה להביט בעניין באופן סטטיסטי. נגריל משבצת באופן אקראי (כלומר, כך שלכל משבצת יש סיכוי שווה להיבחר). אם אנו במקרה השני, בחצי מהמקרים נבחר 0, כלומר בהסתברות של $\frac{1}{2}$. אם נגריל שוב, ההסתברות היא $\frac{1}{4}$ לבחור 0. וכך אם נגריל 1000 פעמים ולא נמצא 0, ההסתברות לכך שנו במקרה השני היא $\frac{1}{2^{1000}}$, כלומר סיכוי נמוך מאוד, מאוד, מאוד (2^{1000} גדול בהרבה מאוד סדרי גודל מכמות החלקיקים המוערכת ביקום).

__הרצאה 3 הגיעה לסיומה!__

הרצאה 4: פתיח לעולם ההסתברות, ניתוח Quick Sort האקראי, הטור ההרמוני והרמוניה למתחילים 31/3/2019

א. תרגול התרגול: חזרה קצרה על המושגים מהתרגול

מרחב הסתברות הוא זוג (Ω, P), כאשר Ω היא קבוצת המאורעות. מאורע בה נקרא **מאורע אטומי.**

מאורע הוא תת־קבוצה של Ω.

משתנה מקרי הוא פונקציה $f: \Omega \to \mathbb{R}$.

לדוגמה: משתנה אינדיקטור, המקבל ערך 1 או 0 בהתאם להתרחשותו או אי־התרחשותו של מאורע מסוים. נניח, במרחב ההסתברות של הטלת שתי קוביות, האינדיקטור למקרה שבו סכום ההטלות הוא 4 הוא:

$$f = \begin{cases} 1 & p \in \{(1,3),(2,2),(3,1)\} \\ 0 & \text{else} \end{cases}$$

תוחלת של משתנה מקרי X היא ממוצע הערכים אותם צפוי המשתנה לקבל, משוקלל על־פי ההסתברויות לקבלת הערכים השונים. פורמלית, התוחלת של משתנה מקרי X מוגדרת כך:

$$\mathbb{E}(X) = \sum_{x \in \operatorname{Im} X} \Pr(X = x) \cdot x$$

מאורעות בלתי תלויים $\boldsymbol{Z, Y}$ הם מאורעות שעבורם:

$$\Pr(Z = z \wedge Y = y) = \Pr(Z = z) \cdot \Pr(Y = y)$$

ההסתברות של $\boldsymbol{A}$ בהינתן $\boldsymbol{B}$ מוגרת כך:

$$P(A \mid B) \stackrel{\text{def}}{=} \frac{P(A \cap B)}{P(B)}$$

נשים לב שעבור מאורעות A ו־B בלתי־תלויים נקבל $P(A \mid B) = P(A)$.

ב. ניתוח הגרסה האקראית של מיון מהיר (Quick Sort)

ננתח עתה את סיבוכיות זמן הריצה של מיון מהיר כאשר הציר הנבחר **אקראי** (כלומר, מספר המוגרל בהסתברות אחידה). שימו לב ש־partition הוא מיקום הציר לאחר מיון אחד.

__טענה 4.1__: אם בחרתי איבר אקראי במערך כציר, אזי m (המיקום שבו הציר נמצא לאחר partition) הוא מספר אקראי שמפולג באופן אחיד בין 1 ל־n.

```
m = partition(A)
```

__הוכחת טענה 4.1__: מבחינה אינטואיטיבית: נניח שהיינו מסדרים את המערך בצורה ממוינת. מאחר שהמספרים מעורבבים באופן חד־חד־ערכי (כלומר כל מספר מגיע למקום יחיד, ולכל מיקום יש מספר

יחיד שהגיע אליו) ההסתברות שנבחר המיקום ה־i שקולה להסתברות שנבחר המספר שהיה קודם לכן במשבצת שתגיע לאחר partition.

חישוב התוחלת של מיון מהיר (Quick Sort)

בהינתן שה־pivot נחת במקום ה־m, נוסחת הרקורסיה היא:

$$T(n \mid m) = T(m-1) + T(n-m) + \Theta(n) \tag{4.2}$$

אבל איננו יודעים באיזה מיקום ה־pivot נחת, לכן נרצה לכתוב נוסחת נסיגה עבור התוחלת, כמעין ממוצע על הערכים השונים ש־m יכול לקבל. נסמן $\mathbb{E}\big(T(n)\big) = R(n)$. אז מלינאריות התוחלת:

$$\mathbb{E}\big(T(n)\big) = R(n) = \sum_{m=1}^{n} \frac{1}{n} \cdot \big(R(m-1) + R(n-m)\big) + \Theta(n)$$

מבחינה אינטואיטיבית, כל אחד מהערכים יוצא בהסתברות $\frac{1}{n}$, לכן אם נסכום לכל n את הסיבוכיות המתאימה לו, נקבל את התוחלת. כעת בצורה רגורוזית יותר. נגדיר משתנים מציינים (אינדיקטורים) $Z_1, \dots, Z_n$:

$$Z_i = \begin{cases} 1 & m = i \\ 0 & else \end{cases}$$

ואז נקבל:

$$T(n) = \sum_{i=1}^{n} Z_i \cdot T(n \mid m = i)$$

בסימון קצת אחר, כאשר M הוא משתנה מקרי, שמקבל את הערך m אם יצא שה־pivot מתמקם ב־m.

$$T(n \mid M = m) \equiv \text{בהינתן ש־}M\text{ יצא }m$$

מדוע זה נכון? אם יצא $i = m$ מסוים, אז כולם יוצאים אפסים מלבד ה־$i = m$ המסוים, ונקבל את נוסחה 4.2 - שהיא נוסחת הנסיגה בהינתן m מסוים. מהתכונה של לינאריות התוחלת שראינו בתרגול, במקום לבצע תוחלת על כל הסכום, נוכל לסכום את התוחלות השונות:

$$\mathbb{E}\big(T(n)\big) = \sum_{m=1}^{n} \mathbb{E}\big(Z_m \cdot T(n \mid M = m)\big)$$

ומאחר שכל ה־Z_m אינם משתנים תלויים[11] (אינם תלויים זה בזה ובהגרלות שבאות לאחריהן, כלומר $\mathbb{E}\big(Z_i Z_j\big) = \mathbb{E}(Z_i)\mathbb{E}\big(Z_j\big)$, ובמקרה זה נשתמש בכך שאינם תלויים ב־$T(n \mid M = m)$), נקבל:

$$\mathbb{E}\big(T(n)\big) = \sum_{m=1}^{n} \mathbb{E}(Z_m)\mathbb{E}\big(T(n \mid M = m)\big)$$

$$= \sum_{m=1}^{n} \underbrace{\mathbb{E}(Z_m)}_{=\frac{1}{n}} \cdot \left[\underbrace{\mathbb{E}\big(T(m-1)\big)}_{R(m-1)} + \underbrace{\mathbb{E}\big(T(n-m)\big)}_{R(n-m)} + \Theta(n) \right]$$

[11] לא נוכיח זאת. הבינו לבדכם מדוע זה נכון.

נשים לב שהתוחלת של $\Theta(n)$ היא פשוט $\Theta(n)$, ושהתוחלת $E(Z_i)$ היא $\frac{1}{n}$, ומכאן נקבל את הדרוש:

$$R(n) = \sum_{m=1}^{n} \frac{1}{n} \cdot \left(R(m-1) + R(n-m)\right) + \Theta(n) \tag{4.3}$$

קיבלנו נוסחת רקורסיה לא פשוטה בכלל! כיצד ניגש לפתוח אותה? נפתח אותה מעט:

$$R(n) = \Theta(n) + \frac{1}{n}\sum_{m=0}^{n-1} R(m) + \frac{1}{n}\sum_{m=1}^{n} R(n-m)$$

ומאחר שאנחנו למעשה סוכמים את אותו הדבר פעמיים, אם נשנה את סדר הסכימה נקבל:

$$R(n) = \Theta(n) + \frac{2}{n}\sum_{m=0}^{n-1} R(m)$$

אבל זה עדיין לא מועיל מדי. נעשה טריק קטן ונגדיר פונקציה חדשה כדי להיפטר מה־$\Theta(n)$, ומאוחר יותר נראה שהיא חסם עליון ל־$R(n)$:

$$U(n) = cn + \frac{2}{n}\sum_{m=0}^{n-1} U(m)$$

וכעת מגיע תורו של הטריק השני להיום: נפתח את נוסחת הרקורסיה המלאה לנוסחה שמשתמשת בכל פעם רק באיבר אחד אחורנית. נכפול ב־n:

$$(\star): \qquad nU(n) = cn^2 + 2\sum_{m=0}^{n-1} U(m) \tag{4.4}$$

נסכום עד איבר אחד יותר, כלומר נציב $n+1$ במקום n:

$$(\star\star): \qquad (n+1)U(n+1) = c(n+1)^2 + 2\sum_{m=0}^{n} U(m) \tag{4.5}$$

נחסר את $(\star)$ מ־$(\star\star)$:

$$(n+1)U(n+1) - nU(n) = c(n+1)^2 - cn^2 + 2U(n)$$
$$(n+1)U(n+1) = 2cn + c + (n+2)U(n)$$

ונקבל נוסחת נסיגה של $U(n+1)$ באמצעות $U(n)$:

$$U(n+1) = \frac{2cn+c}{n+1} + \frac{n+2}{n+1}U(n)$$

נשים לב שמתקיים:

$$U(n+1) \le 2c + \frac{n+2}{n+1}U(n)$$

נניח קודם כל, $U(1) \le c'$, וניקח $d \ge 2c, c'$ קבוע שגדול מ־$2c$ ומ־c'. נקבל את נוסחת הרקורסיה הזו:

$$U(n+1) \le d + \frac{n+2}{n+1}U(n), \qquad U(1) \le d \tag{4.6}$$

נפתור בשיטת האיטרציה:

$$U(n+1) \le d + \frac{n+2}{n+1} \cdot \left(d + \frac{n+1}{n} U(n-1)\right) = \cdots$$

נשים לב שהמקדם של $U(n-1)$ הוא $\frac{n+2}{n}$, כי $n+1$ מצטמצם מהמונה והמכנה (וימשיך לעשות כן בצורה טלסקופית):

$$\ldots = d + \frac{n+2}{n+1} d + \frac{n+2}{n} U(n-1) \le d + \frac{n+2}{n+1} d + \frac{n+2}{n} \cdot \left(d + \frac{n}{n-1} U(n-2)\right) \le$$

$$\le d + \frac{n+2}{n+1} d + \frac{n+2}{n} d + \frac{n+2}{n-1}\left(d + \frac{n-1}{n-2} U(n-3)\right) =$$

אפשר להמשיך לפתוח כך עד אינסוף, וזה בדיוק מה שנעשה, רק נחסוך לעצמנו עבודה אינסופית. נחשוב מה יהיה האיבר האחרון:

$$= (n+2) \cdot \left[d\left(\frac{1}{n+2} + \frac{1}{n+1} + \cdots + \frac{1}{i} + \cdots + \frac{1}{2}\right) + U(1)\right]$$

זה הטור ההרמוני! איזו אלגנטיות! מה טור יפה כמוך עושה בנוסחת נסיגה מכוערת שכזו? כמובן עם תוספת של קבוע $U(1)$. נקבל, כאשר $H(n)$ מסמן את הסכום של הטור ההרמוני עד n:

$$R(n) \le (n+2)d(H(n+2) + \text{const}) \tag{4.7}$$

ג. __הערת ביניים מאינפי: על הטור ההרמוני וקבוע אוילר־מסקרוני__

הסדרה ההרמונית $H(n)$ מוגדרת כך:

$$H(n) = \sum_{i=1}^{n} \frac{1}{i}$$

כלומר: $H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}$. הוא נקרא כך בגלל תופעה שכל מוזיקאי וכל פיזיקאי מכיר: אוברטונים. זהו טור ש**אינו** מתכנס,[12] אלא גדל עד אינסוף, אולם הגדילה של הטור איטית מאוד. אם תנסו לפתח לסכום רימן את האינטגרל $\int_1^n \frac{dt}{t}$ תראו שיש דמיון רב ללוגריתם, ולא בכדי.

__טענה 4.8:__

$$H(n) = \Theta(\log n)$$

אם משעמם לכם והתגעגעתם להוכחות באינפי, תוכלו להראות שהסדרה $H_n - \ln n$ יורדת מונוטונית אך חסומה מלמטה, ולכן מתכנסת, כאשר לגבול הזה:

[12] זאת בניגוד לבן דודו, $\sum_{i=1}^{\infty} \frac{1}{i^2} = 1 + \frac{1}{4} + \frac{1}{9} + \cdots$, שמתכנס לערך המפתיע $\frac{\pi^2}{6}$, כפי שהראה אוילר בפתרונו עבור בעיית בזל. כפי שראיתם בתרגיל הבית, הטור $\sum_{i=1}^{\infty} \frac{1}{i^n}$ מתכנס עבור $n \ge 2$, ע"ע פונקציית זטא של רימן.

$$\gamma = \lim_{n\to\infty}\left(\left(1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}\right)-\ln n\right)\approx 0.5772\ldots \qquad (4.9)$$

שמסמנים ב־γ, קוראים קבוע אוילר־מסקרוני. קבוע זה מעניין מאוד בפני עצמו: למשל, איננו יודעים אם הוא רציונאלי או לא.

אנחנו נוכיח את טענה 4.7 ללא שימוש בקבוע זה. נחלק את $H(n)$ ל"קופסאות" שכל אחת מהן חסומה מלמעלה על־ידי 1 (למה 1? כי האיבר הראשון הוא $\frac{1}{2^k}$, והטור יורד כמובן):

$$H(n) = \underbrace{1}_{\le 1} + \underbrace{\frac{1}{2}+\frac{1}{3}}_{\le 1} + \underbrace{\frac{1}{4}+\frac{1}{5}+\frac{1}{6}+\frac{1}{7}}_{\le 1} + \underbrace{\frac{1}{8}+\cdots+}_{\le 1}\frac{1}{n}$$

נניח כעת $n = 2^k - 1$ מטעמי פשטות, ולכן $k = \log_2(n-1)$. מספר ה"קופסאות" הוא k, ולכן:

$$H(n) = O(\log n) \qquad (4.8)$$

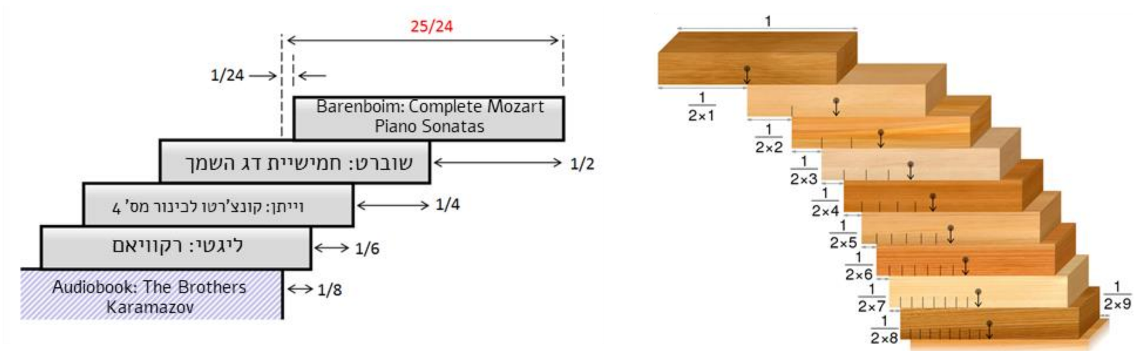
Q.E.D

תוכלו לנסות להוכיח גם עבור n שאינו מהסוג $2^k - 1$.

ד. הרמוניה א' וקונטרפונקט למתחילים: מיצירות באך ושוסטקוביץ'

דוגמה לטור ההרמוני מעולם המוזיקה

ניקח מכלול דיסקים מצוינים של יאשה חייפץ, דניאל ברנבוים, יהודי מנוחין וחברים נוספים. נרצה לאזן את כל הדיסקים בקצה השולחן - בתיאוריה, מספר אינסופי שלהם, אך לצערנו חייפץ נפטר בשנת 87' (ועל־כן גם הפסיק להקליט). כיצד נוכל לאזן אותם? והאם ניתן להתקדם איתם עד אינסוף? התשובה המפתיעה היא שכן! אם נחשב בכל הזזה את מרכז המסה החדש, נראה שנוכל להטות את הדיסקים באופן הבא בלי להפיל אותם:



שרטוט 4.1: מימין: אוסף הדיסקים של דורית אהרונוב, בקצה השולחן, נצבע מחדש (2019)
משמאל: אוסף הדיסקים של דורית אהרונוב, בקצה השולחן, סכמטי (2019)

לבעיה זו קוראים Block-stacking problem, והיא פופולרית מאוד במוזיאוני מדע ברחבי העולם.

הרצאה 4 הגיעה לסיומה!

הרצאה 5: אי־שוויון מרקוב, חסם תחתון על אלגוריתמים ממיינים ועצי חיפוש בינאריים

7/4/2019

א. <u>המשך וסיום ניתוח של אלגוריתמים ממיינים הסתברותיים</u>

אי־שוויון מרקוב

מדוע התוחלת בכלל מעניינת אותנו? למה העובדה שהתוחלת היא $\mathbb{E}(x) = n$ אומרת שזמן הריצה אינו יכול להיות $100n$?

<u>טענה 5.1</u>: יהי x משתנה מקרי אי־שלילי שמקיים $\mathbb{E}(x) = \mu$, ויהי $\lambda > 1$. אז $\Pr(X > \lambda\mu) \leq \frac{1}{\lambda}$.

לדוגמה, אם תוחלת זמן הריצה היא $3n$, אז ההסתברות שזמן ריצה יהיה $10n$ קטנה או שווה ל־$\frac{3}{10}$.

<u>הוכחת טענה 5.1</u>:

$$\mu = \mathbb{E}(x) = \sum_x \Pr(X = x) \cdot x =$$

נפצל לסכום המקרים השונים, כאשר $x \geq \lambda\mu$ ולהיפך:

$$= \underbrace{\sum_{x, x \leq \lambda\mu} \Pr(X = x)\, x}_{\geq 0} + \sum_{x, x \geq \lambda\mu} \Pr(X = x) \cdot x \geq \lambda\mu \sum_{x, x \geq \lambda\mu} \Pr(X = x)$$

וקיבלנו כך:

$$\mu \geq \lambda\mu \cdot \Pr(X \geq \lambda\mu)$$

$$\frac{1}{\lambda} \geq \Pr(X \geq \lambda\mu)$$

וקיבלנו את הדרוש.

כיצד להטות את הסטטיסטיקה לטובתך

<u>כמה הערות על סטטיסטיקה לפני שממשיכים</u>:

א. סטטיסטיקה זה תחום מבלבל ומטעה, ולעיתים קרובות ניתן לספר עובדות סטטיסטיות מטעות מאוד. פרדוקס מוכר בנושא הוא <u>פרדוקס מונטי־הול</u>, אבל ניקח לדוגמה פרדוקס אחר, פרדוקס סימפסון:

אם 40% מניסיונותיו של שחקן הכדורסל הנודע טארק היו קליעות, לעומת 50% מקליעותיו של ברדה שהיו מוצלחות, ובמחצית השנייה טארק צלח ב־80% מקליעותיו, לעומת 90% מניסיונותיו של ברדה שצלחו. האם ייתכן שבאחוזים כוללים מתוך ניסיונות הקליעה, טארק עקף את ברדה? כן! להלן דוגמה:

	טארק	**ברדה**
מחצית ראשונה	40% (מתוך 10 ניסיונות: 4 קליעות)	50% (מתוך 20 ניסיונות: 10 קליעות)
מחצית שנייה	80% (מתוך 20 ניסיונות: 16 קליעות)	90% (מתוך 10 ניסיונות: 9 קליעות)
סך הכול	20 קליעות מתוך 30	19 קליעות מתוך 30

ב. שאלות מסוג זה, כפי שנראו קודם לכן בתרגול, דומות מאוד, אך על־אף דמיון זה התשובה להן לעיתים שונה מהותית! ניקח לדוגמה את שתי הבעיות האלו:

(1) נניח שישנן שתי מעטפות, ויש לך 50% סיכוי לבחור באחת שמכילה יותר כסף. בחרת במעטפה א', האם כדאי להחליף?

(2) נניח שאתה אולמרט והעבירו לך את מעטפה א'. מציעים לך אחת נוספת, מעטפה ב'. ידוע לך שיש 50% שתקבל במעטפה ב' סכום כפול משיש בא', ו־50% שהסכום יהיה מחצית מזה שיש בא'. האם שווה לך להחליף?

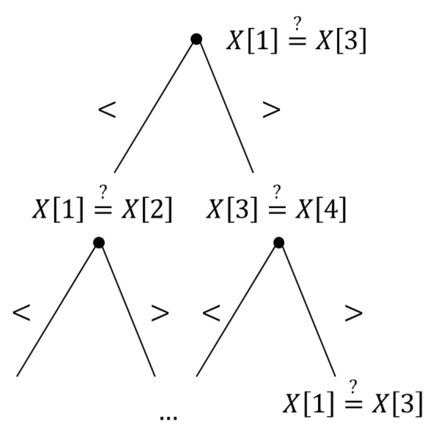
ב. <u>חסם תחתון על זמן הריצה של אלגוריתם מיון מבוסס השוואות</u>

ראינו אלגוריתמים ממיינים שרצים בסיבוכיות זמן ריצה של $O(n\log n)$, אבל בתרגול נתקלנו גם בכאלו הרצים ב־$O(n)$ - אם כי להם היו הנחות על הקלט. אנו רוצים כעת להוכיח שכל אלגוריתם ממיין שאינו מניח הנחות על המערך שקיבל, ירוץ בזמן שהוא לפחות $O(n\log n)$. כיצד נוכיח דבר כזה?

עץ הכרעה

קודם כול, אל־נא תבלבלו מונח זה עם עץ רקורסיה. זהו מבנה מתמטי המאפשר ניתוח של האלגוריתם.

נניח מטעמי פשטות שהאיברים הממוינים שונים זה מזה. בכל קדקוד של העץ נשרטט את ההשוואה שהאלגוריתם מבצע.

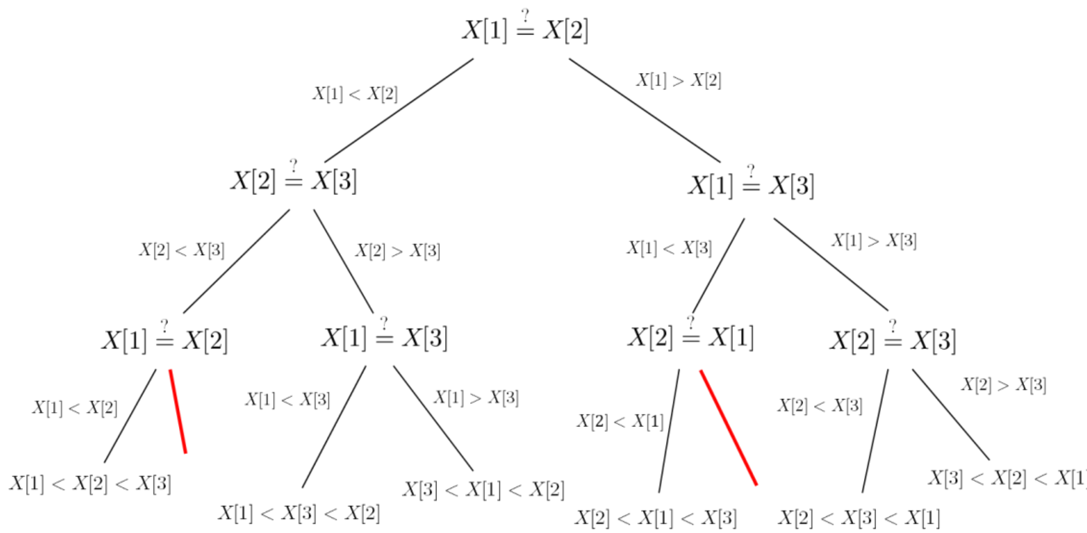


שרטוט 5.1: דוגמה לעץ הכרעה של אלגוריתם ממיין מבוסס השוואות כלשהו. בכל שלב בעץ הוא בוחר את פעולותיו והשוואותיו הבאות לפי תוצאות ההשוואה הקודמת.

ריצה של אלגוריתם על קלט מסוים מתאימה לירידה בעץ לאורך מסלול מסוים. סיבוכיות זמן הריצה בריצה זו גדול או שווה לאורך המסלול.
לאחר כל ההשוואות האלו מתקבלת איזושהי תמורה על האיברים, כלומר סידור מחדש שלהם.

__שאלה:__ כיצד ייראה עץ ההכרעה עבור אלגוריתם מיון בועות עם מערך בגודל של $n = 3$?

__פתרון:__



שרטוט 5.2: עץ הכרעה של אלגוריתם bubble sort למקרה של גודל מערך $n = 3$.
הקווים האדומים מייצגים שגיאה.

נגדיר גובה עץ בתור אורך המסלול המקסימלי מהשורש עד לעלה כלשהו, ונסמנו ב־h. שימו לב, כאשר אנו מדברים על עץ הכרעה אנחנו מתייחסים רק למסלולים האפשריים - כל ענף בעץ מתאים לקלט אפשרי כלשהו, אין מסלול שבלתי־אפשרי להגיע אליו.

__טענה 5.2:__ h הוא חסם תחתון על זמן הריצה של האלגוריתם. לא נוכיח זאת, אך הטענה נובעת מידית מעצם ההגדרה של עץ הכרעה.

__טענה 5.3:__ כל תמורה על n חייבת להופיע לפחות בעלה אחד בעץ ההכרעה ולכן יש לפחות $n!$ עלים בעץ. זאת מאחר שייתכן קלט בכל סידור שהוא (מתוך $n!$ הסידורים האפשריים), ולכל סידור כזה מתאימה תמורה אחת ויחידה שתמיינו לפי הסדר.

נשים לב שלעץ בינארי בגובה h יש לכל היותר 2^h עלים (טענה שהוכחה בקורס במתמטיקה דיסקרטית ותוכיחו שוב בתרגיל). אז (5.2) ו־(5.3) אומרות ש־ $2^h \geq n!$, ולכן:

$$h = \log(2^h) \geq \text{מספר העלים} \geq \log(n!) \tag{5.4}$$

ננסה לחסום את $\log(n!)$, לפי נוסחת סטירלינג:

$$\log(n!) = \log(n(n-1)(n-2)\dots) = \log(n) + \log(n-1) + \dots = \sum_{i=1}^{n} \log i \geq$$

ניתן לקבל קירוב לסכום זה על־ידי סכום רימן, כלומר באמצעות קירוב עם אינטגרל:

$$\sum_{x=1}^{n} \log x \approx \int_{1}^{n} \log x \, dx = n \ln n - n + 1$$

אך בהרצאה הסתפקנו בהוכחה ש־ $\log(n!) = \Omega(n \log n)$, שהוכחנו באופן שלהלן. מאחר ש־$\log n$ היא פונקציה מונוטונית:

$$\dots \geq \sum_{i=\frac{n}{2}}^{n} \log i \geq \sum_{i=\frac{n}{2}}^{n} \log\left(\frac{n}{2}\right) = \left(\frac{n}{2}+1\right) \log\left(\frac{n}{2}\right) = \Omega(n \log n)$$

בזאת סיימנו את החלק הראשון של הקורס, שבו הכרנו כלים רגורוזיים לניתוח אלגוריתמים למיניהם. נגיע כעת לחלק עיקרי יותר של מבני נתונים.

ג. עצי חיפוש בינאריים (BST)

הידד! הגענו למבנה הנתונים הראשון שלנו, עץ חיפוש בינארי!

כפי שכבר ראינו באינטרו, אלו הן התכונות שנרצה לממש במבנה נתונים זה:

א. **Insert** - הכנסת איבר לעץ.
ב. **Delete** - הסרת איבר מהעץ.
ג. **Search** - חיפוש איבר.
ד. **Max** - מציאת מקסימום בעץ.
ה. **Min** - מציאת מינימום בעץ.
ו. **Successor** - מציאת העוקב של קדקוד בעץ, כלומר האיבר המינימלי מבין אלו שגדולים ממנו.
ז. **Predecessor** - מציאת קודם לאיבר בעץ, כלומר מציאת האיבר הגדול מבין אלו הקטנים ממנו.

בהרצאה הבאה נראה כיצד מממשים תכונות אלו.

הרצאה 5 הגיעה לסיומה!

הרצאה 6: עצי חיפוש בינאריים (BST) ופעולות עליהם

14/4/2019

א. פעולות בעצי חיפוש בינאריים: חיפוש, הכנסת ערך, מינימום ומקסימום

נזכיר מאינטרו: לכל קדקוד בעץ יש סימון x,y, ולכל אחד יש מפתח $\text{key}(x)$ לפיו אנחנו מחפשים. לכל קדקוד x מוגדר $\text{left}(x)$ ו־$\text{right}(x)$, וכן $\text{parent}(x)$. לעץ כולו T מוגדר $\text{root}(T)$. עתה נזכיר את ההגדרה ואת התכונות של עץ חיפוש בינארי.

תכונות עץ חיפוש בינארי: BST

אם y בַּתת־עץ[13] השמאלי של x, אז:

$$\text{key}(y) \leq \text{key}(x)$$

אם x בתת־עץ הימני של x, אז:

$$\text{key}(y) > \text{key}(x)$$

ונזכיר כמה מהפעולות שנרצה לבצע על עץ בינארי:

```
Search(x, k)
Tree_min(x)
Tree_max(x)
```

כאשר x הוא שורש העץ, לאו דווקא שורש העץ הראשי (אולי שורש של תת־עץ שלו). שימו לב שהעלים מצביעים על הערך null.

להלן פסודו־קוד לפונקציות הללו:

```
def Tree_search(x, k):
    if x == null:
        return "NOT_FOUND_ERROR"
    else if key(x) == k:
        return x
    else if key(x) < k:
        return Tree_search(right(x), k)
    else:
        return Tree_search(left(x), k)
```

[13] למוטרדים בענייני לשון יפה, "התת־עץ" היא הצורה התקנית, ולא "תת־העץ". באותו אופן, הריבוי של "תת־קבוצה" הוא "תת־קבוצות" ולא "תתי־קבוצות", וכן הריבוי של "מד־מהירות" (ליתר דיוק, מַדְמְהִירוּת) הוא הצורה המפתיעה "מַדְמְהִירוּתִים". מדוע? ראו פירוט כאן.

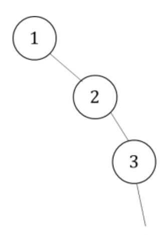
שימו לב שאנו מוצאים עד קדקוד **אחד** שערכו k, אבל ייתכנו עוד. באשר למציאת מינימום - זה קל מאוד, פשוט הולכים שמאלה כל הזמן עד שנתקלים ב־null:

```
def Tree_min(x):
    while left(x) ≠ null:
        x = left(x)
    return x
```

ועבור Tree_max, החליפו את המילה left ב־right[14] ותקבלו את מבוקשכם. כעת נרצה לממש את הפונקציה insert, אשר מכניסה לעץ T את הערך z. להלן פסודו־קוד מתאים:

```
def insert(T, z):
    y = null
    x = root(T)
    while x ≠ null:
        y = x
        if key(z) ≤ key(x):
            x = left(x)
        else:
            x = right(x)
    parent(z) = y
    if y == null: # in case of empty tree
            root(T) = z
    else if key(z) ≤ key(y):
            left(y) = z
    else:
        right(y) = z
```

מה זמן הריצה של הדבר הזה? כולם לוקחים כגובה העץ, ובמקרה הגרוע ביותר הוא יכול להיות n, כמספר הערכים שבתוך העץ (ראו שרטוט 6.1).



שרטוט 6.1: עץ חיפוש בינארי ממש גרוע (מבחינת סיבוכיות זמן הפעולות עליו), אך תקין. נדבר בהמשך על עצי חיפוש מאוזנים.

[14] למה זה נכון? כי אין יותר ימין ושמאל.

עולה צורך לחלק מן הקפדנים שבקהל להוכיח את הנכונות של insert, אבל אין סיבה כי זה טריוויאלי ולכן נשאיר זאת כתרגיל לקורא.

ב. מציאת עוקב וקודם בעצי חיפוש בינאריים

אנו רוצים כעת לתכנן פונקציה שתמצא את העוקב של ערך כלשהו. מהו עוקב של ערך $x = \text{key}(y)$? הערך הקטן ביותר במערך שגדול ממש מ- x. אנחנו נניח מטעמי פשטות שאין ערכים זהים במערך.

מכלול טענות 6.1: שלושה מקרים אפשריים (שלוש טענות להוכחה) בקשר לאלגוריתם למציאת עוקב:

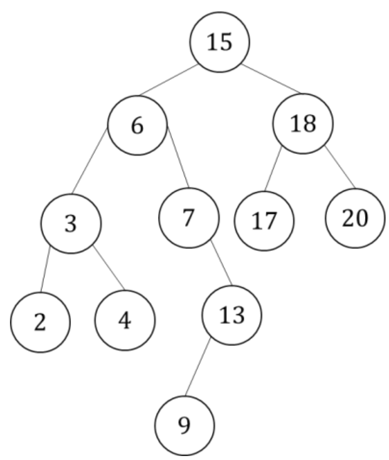
א. אם ל־x יש בן ימני w, העוקב הוא $\min(T(w))$, כלומר המינימום של העץ ש־w שורשו.

ב. אם ל־x אין בן ימני, אז:

i. הוא המקסימום בעץ.

ii. אחרת, העוקב y יהיה האב הקדמון הכי נמוך של x שמקיים שגם ילדו השמאלי הוא אב קדמון של x, או x בעצמו.

ניקח עץ לדוגמה:



שרטוט 6.2: עץ חיפוש בינארי לדוגמה. אם ננסה למצוא את העוקב ל־13 לפי האלגוריתם המתואר, מאחר שאין לו בן ימני, נחפש במעלה העץ, ונמצא שהבן השמאלי של 15 הוא אב קדמון של 13, ולכן הוא עוקבו.

אם הטענות נכונות, אז קל מאוד לבנות אלגוריתם שיפתור את הבעיה:

```
def Successor(node):
    if node.right is not null:
        return Tree_min(node.right)
    parent = node.parent
    while (parent is not null) and (node == parent.right):
        node = parent
```

```
        parent = node.parent
    return parent
```

כעת ננסה להוכיח את הטענות, מטלה שאינה כל כך פשוטה למעשה. ניעזר בטענת עזר חשובה:

<u>טענה 6.2: טענת הטווח</u>: נתבונן בתת־עץ $T(x)$ ששורשו x. אם z איבר מחוץ ל־$T(x)$, אז:

$$\text{key}(z) \le \text{Tree\_min}(x)$$

או:

$$\text{key}(z) \ge \text{Tree\_max}(x)$$

<u>הוכחת טענה 6.2</u>:

נוכיח את הטענה עבור עץ חיפוש בינארי כללי, כלומר שייתכנו ערכים זהים. ייתכנו שלושה מקרים:

1. **הקדקוד z הוא אב קדמון של x.** במקרה זה, x נמצא בתת־עץ השמאלי או בתת־עץ הימני של z, ומתכונת עץ החיפוש הבינארי (BST), נקבל ש־z קטן או גדול (במובן החלש) בהתאמה מכל ערכי התת־עץ שלו, ולכן הטענה מתקיימת.
2. **הקדקוד z נמצא בתת־עץ ימני של אב קדמון w של x.** במקרה זה $\text{key}(w) < \text{key}(z)$, ולפי סעיף 1 בטענה זו (6.2.1) מתקיים ש־$\max\big(T(x)\big) \underset{(BST)}{\ge} \text{key}(w)$.
3. **אחרת: הקדקוד z נמצא בתת־עץ שמאלי של אב קדמון של x.** אלו הם כל שלושת המקרים: נעלה בעץ מ־z עד השורש, ומ־x עד השורש, ונניח ששני המסלולים נפגשים לראשונה בקדקוד w, אז מאחר ש־z קדקוד מחוץ ל־$T(x)$, אחד משלושת המקרים המצוינים חייבים לקרות; במקרה זה מתקיים $\text{key}(z) \le \text{key}(w) \le \min\big(T(x)\big)$.

בכל במקרים $\text{key}(z)$ לא נמצא בטווח המחובר.

Q.E.D

<u>הוכחת טענה 6.1</u>: ניעזר בטענת הטווח על־מנת להוכיח את טענות 6.1.

א. נתבונן ב־$T(x)$. בתוך $T(x)$ קל לראות ש־$\min\big(T(w)\big)$, כאשר w הבן הימני של x, הוא העוקב. זה נובע ישירות מטענת הטווח, שכן $\min\big(T(w)\big)$ הוא המינימלי מבין כל הערכים הגדולים מ־$\text{key}(x)$ בתוך $T(w)$, וכל שאר האיברים ב־$T(x)$ קטנים מ־$\text{key}(x)$.

ב. נעלה עד הפנייה הראשונה ימינה ונסמן קדקוד זה ב־w. נתבונן ב־$T(w)$. x נמצא בתת־עץ שמאלי של w, והוא המקסימום של תת־עץ זה בתחום $T(w)$, ולכן בתוך $T(w)$, w הוא העוקב של x.

Q.E.D

ואיך מוצאים קודם? תענו על שאלו זו, כמובן, בתרגיל.

<u>הרצאה 6 הגיעה לסיומה!</u>

הרצאה 7: עצי חיפוש מאוזנים ופעולות עליהם, עצי AVL, שבלולי פיבונאצ'י וצנוברים 28/4/2019

א. עצי AVL: הגדרה והוכחת איזון

הגדרה

השם AVL לקוח משמות הממציאים - **A**delson, **V**elsky and **L**andis.

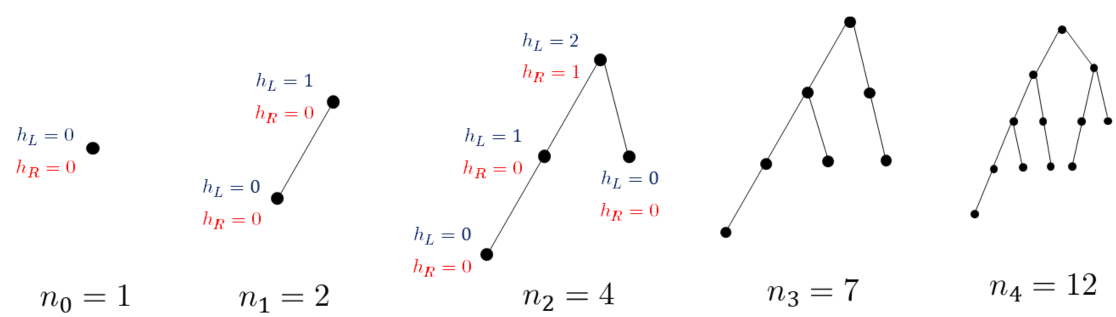
התכונה המגדירה את העצים הללו היא שבכל קדקוד x, ההבדל בין גובה התת־עץ הימני לגובהו של התת־עץ השמאלי הוא 1 או 0. כלומר, שלכל קדקוד מתקיים:

$$|h(T_L) - h(T_R)| \leq 1 \qquad (7.1)$$

כאשר $h(T)$ הוא הגובה של העץ T.[15] כרגיל, L מקצר $Left$ ו־R מקצר $Right$. נרצה כעת להוכיח שעץ המקיים תנאי זה הוא בהכרח עץ מאוזן, כלומר שגובהו גדל לוגריתמית עם מספר הקדקודים.

הוכחת איזון

נגדיר סדרה n_k, היא מספר הקדקודים **המינימלי** בעץ AVL בגובה k. הערכים הראשונים של סדרה זו הם:



*שרטוט 7.1: העצים המתאימים לחמשת הערכים הראשונים בסדרה n_k, המייצגת את מספר הקדקודים **המינימלי** בעץ AVL בגובה k. נסו לחשוב על נוסחת נסיגה ליצירת העצים הללו. רמז: מהם תתי־העצים, הימני והשמאלי?*

<u>טענה 7.2</u>: $n_k \geq ac^k$ עבור $c > 1$ ו־$a > 0$ כלשהם (כלומר, $n_k = \Omega(c^k)$. למעשה, נראה עוד מעט ש־$n_k = \Theta(\varphi^k)$, או בדיוק: $n_k = 2F_k + L_k - 1$, כש־F_k מספר פיבונאצ'י ה־k, L_k מספר לוקאס ה־k.

אם טענה 7.2 נכונה, נוכל להסיק שעץ AVL הוא אכן מאוזן. נניח ש־T עץ AVL בגובה k ועם n קדקודים. אז:

$$n \geq n_k \geq ac^k \Rightarrow n \geq ac^k$$

נוציא לוגריתם משני הצדדים:

$$\log n \geq \log a + k \log_2 c$$

---

[15] כלומר, מספר הצלעות המקסימלי עד לעלה.

$$k \leq \frac{\log n - \log a}{\log_2 c} \Longrightarrow k = O(\log n)$$

<u>טענה 7.3</u>: n_k מונוטוני עולה **ממש** ב־k.

<u>הוכחת טענה 7.3</u>: נסתכל על עץ AVL בגובה $k+1$, עם מספר קדקודים מינימלי n_{k+1}. מאחר שגובה העץ הכולל הוא $k+1$, גובה אחד מהתת־עצים (הימני והשמאלי) הוא k, נניח בלי הגבלת הכלליות שמדובר בעץ השמאלי. ממינימליות העץ כולו, נובע שבתת־עץ השמאלי יש n_k קדקודים, לכן בעץ כולו יש לפחות $n_k + 1$ קדקודים, ומכאן ש־$n_{k+1} > n_k$, ולכן n_k מונוטוני עולה ממש. Q.E.D

<u>הוכחת טענה 7.2</u>: ניעזר ב־(7.3) לשם הוכחת (7.2). נמצא נוסחת נסיגה ל־n_k. נניח שוב בלי הגבלת הכלליות שגובה העץ השמאלי $k-1$, אז כפי הסברנו בהוכחת (7.3), בתת־עץ השמאלי יש n_{k-1} קדקודים. מאחר שמדובר בעץ AVL, העץ מקיים את תנאי (7.1), לכן גובה התת־עץ הימני הוא $k-1$ או $k-2$. מאחר שמדובר בעץ עם מספר קדקודים מינימלי, ומתקיים $n_{k-1} < n_{k-2}$ לפי טענה (7.3), אז גובה העץ השמאלי $k-2$, וממינימליות העץ הכולל, בתת־עץ הימני יש n_{k-2} קדקודים. אם נסכום את n_{k-1} קדקודי התת־עץ השמאלי, n_{k-2} קדקודי התת־עץ הימני ואת השורש נקבל את נוסחת הנסיגה הזו עבור n_k:

$$\boldsymbol{n_k = n_{k-1} + n_{k-2} + 1} \tag{7.4}$$

נוסחה זו מזכירה מעט את מספרי פיבונאצ'י. ניתן להוכיח באינדוקציה (בתרגיל) שהסדרה הזו:

$$a_1 = 1, \qquad a_n = a_{n-1} + a_{n-2} + C$$

מקיימת:

$$\boldsymbol{a_n = (C+1)F_n - C}$$

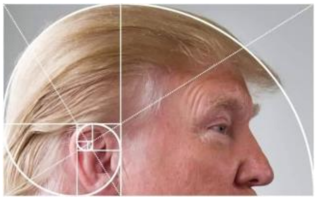
כאשר F_n הוא [מספר פיבונאצ'י](#) ה־n, וקיימת גם [נוסחה כללית](#) הנעזרת ב־L_n, מספר [לוקאס](#) ה־n. עם זאת, נפתור את נוסחת הנסיגה (7.4) באופן אחר. נגדיר $g_k = 1 + n_k$, אז:

$$g_k - 1 = g_{k-1} - 1 + g_{k-2} - 1 + 1$$
$$g_k = g_{k-1} + g_{k-2}$$

ומי שלא יודע איך פותרים את נוסחת הנסיגה הזו, יכול לנסות [ללכסן את המטריצה](#) $A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}$ המקיימת: $\begin{bmatrix} F_{n+1} \\ F_n \end{bmatrix} = A \begin{bmatrix} F_n \\ F_{n-1} \end{bmatrix} = A^n \begin{bmatrix} F_1 \\ F_0 \end{bmatrix}$, ובבסיס המלוכסן לחשב באופן סגור את A^n. או שאפשר לנחש:

$$g_k = ac^k \Longrightarrow ac^k = ac^{k-1} + ac^{k-2}$$

קיבלנו את המשוואה $c^k = c^{k-1} + c^{k-2}$, משוואה שפתרונותיה הם φ, ψ. נזכיר: $\varphi = \frac{1+\sqrt{5}}{2}$ הוא יחס הזהב, אחד הפתרונות של המשוואה $x^2 = x + 1$, הפתרון השני הוא $\psi = \frac{1-\sqrt{5}}{2}$. שני הפתרונות הם גם פתרונות של המשוואה הכללית יותר, $x^k = x^{k-1} + x^{k-2}$. מי שלא מכיר את הספירלות ושלל



שרטוט 7.2: נשיא ארה"ב מדגים את יחס

התופעות הקשורות למספרי פיבונאצ'י, שילך לאסוף כמה צנוברים מאצטרובלים, שיצפה בהכתבות קצב הודיות, או שיתבונן במיני מְחְקִים (בלעז: memes) על ספירלות של יחס הזהב (שרטוט 7.2).

נוכל לבדוק שמתקיים:

$$a\varphi^k + b\psi^k = \underbrace{a\varphi^{k-1} + a\varphi^{k-2}}_{a\varphi^k} + \underbrace{b\psi^{k-1} + b\psi^{k-2}}_{b\psi^k} \Longrightarrow$$

$$\boldsymbol{g_k = a\varphi^k + b\psi^k = g_{k-1} + g_{k-2}}$$

ובהצבת תנאי התחלה ($a + b = 2 = g_0$, $a\varphi + b\psi = 3 = g_1$) מקבלים $g_k = F_{k-3} = \frac{\varphi^{k-3} - \psi^{k-3}}{\sqrt{5}}$,

ומכאן ש־ $n_k = \Theta(F_n) = \Theta(\varphi^k)$, ובזאת סיימנו להוכיח את (7.2). $\mathfrak{Q.E.D}$

ב. הכנסת איבר לעצי AVL

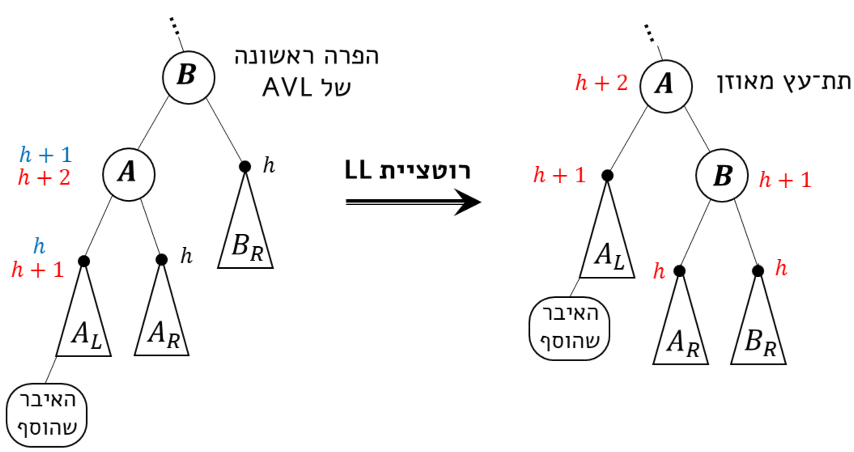
<u>הגדרה 7.3: גורם האיזון</u>: נגדיר את גורם האיזון כך: $h(T_L) - h(T_R)$.

בעצי AVL הוא צריך להיות ± 1 או 0.

נרצה למצוא דרך להכניס קדקוד חדש לעץ AVL בלי לשנות את תנאי (7.1), כלומר תוך שאנו משאירים אותו עץ AVL. אחרי הכנסת איבר, נעלה לשורש ובכל קדקוד נתקן את הגובה. כמו כן נבדוק את גורם האיזון, ובהפרה הראשונה נעצור.

<u>מקרה ראשון, LL</u>: נניח שהכנסנו קדקוד בקצה של A_L והתחלנו לעלות אל השורש, ועצרנו בקדקוד B שמפר לראשונה את גורם האיזון. נסמן את הגובה של B_R ב־h, אז הגובה של התת־עץ A_R הוא h (מדוע? נסו לחשוב מה היה קורה אילו הוא היה $h+1$ או $h-1$ - אז היינו מקבלים סתירה לכך ש־B הוא ההפרה הראשונה של AVL, או סתירה לכך שהעץ היה מאוזן קודם).

נרצה לתקן את השגיאה הזו. במקרה זה נוכל לבצע **רוטציית LL**: נשנה את המצביעים של הקדקודים לקבלת העץ הזה:



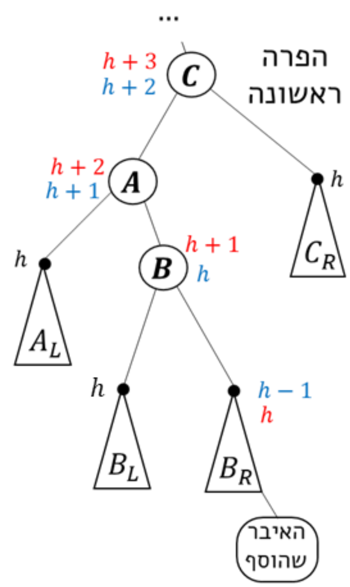
שרטוט 7.3: המקרה של LL, כלומר המקרה שבו האיבר שהוסף הוא משמאל לעץ השמאלי של ההפרה הראשונה. משמאל: העץ לאחר הכנסת האיבר הנוסף, לפני איזונו. מימין: לאחר רוטציית LL.בשרטוט משמאל, מופיע בכחול: גובה התת־עץ לפני הכנסת איבר נוסף, באדום: גובה התת־עץ לאחר ההוספה. בשחור: תת־עץ שגובהו לא השתנה לפני ואחרי ההוספה.

נשים לב שביצוע פעולה זו של רוטציית LL דורשת סיבוכיות זמן של $O(1)$, כיוון שאנו משנים את המצביעים למספר קבוע של קדקודים. נשים לב כמו כן שמתקיימים התנאים האלו:

1. אחרי הרוטציה תוקנה ההפרה.
2. העץ לאחר הרוטציה נותר עץ חיפוש בינארי.
3. סיבוכיות כל ההכנסה היא כגובה העץ, כלומר $O(\log n)$ כאשר n מספר הקדקודים.

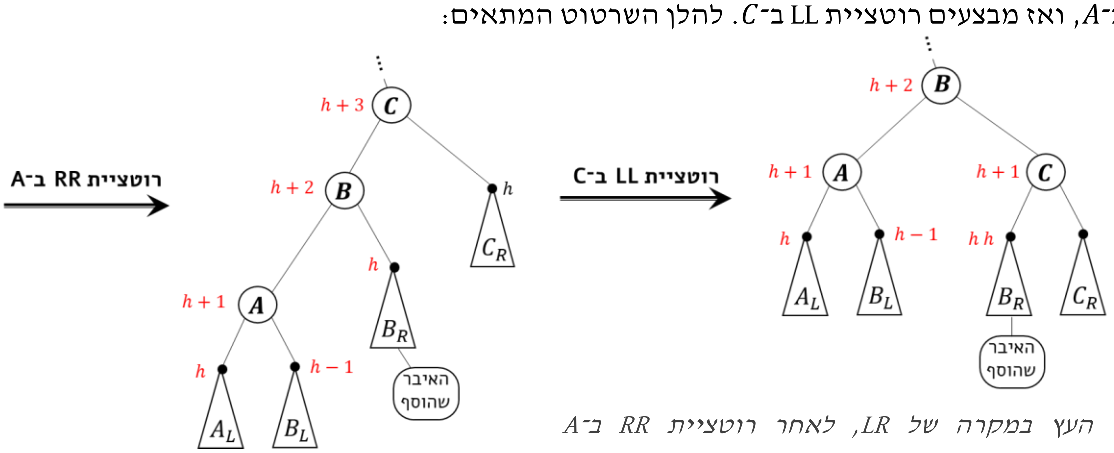
<u>מקרה שני, RR</u>: אותו הדבר כמו LL אבל בהיפוך מראה.

<u>מקרה שלישי, LR</u>: נניח שההפרה הראשונה היא בקדקוד C:



שרטוט 7.4: העץ במקרה של LR, לפני הרוטציה. בכחול: גובה התת־עץ לפני הכנסת איבר נוסף, באדום: גובה התת־עץ לאחר ההוספה. בשחור: גובה העץ אם לא השתנה לפני ואחרי ההוספה.

ובמקרה זה הרוטציה קצת יותר מסובכת. מפרקים אותה לשתי רוטציות: ראשית מבצעים רוטציית RR ב־A, ואז מבצעים רוטציית LL ב־C. להלן השרטוט המתאים:



שרטוט 7.5: העץ במקרה של LR, לאחר רוטציית RR ב־A (משמאל), ובמצבו המאוזן לאחר רוטציית LL ב־C (מימין).

נשים לב שגם במקרה זה מתקיימות שלוש התכונות מקודם (העץ מאוזן, נותר עץ חיפוש בינארי והסיבוכיות הכוללת היא $O(\log n)$). מי שהשרטוטים לא ברורים לו, האנימציה הזו מסבירה אותם היטב. בהמשך נראה כיצד מסירים קדקודים מעצי AVL.

עשרת האיברים הראשונים ב־n_k, סדרת מספר הקדקודים המינימלי בעץ AVL בגובה k הם:

$$(n_k)_{k=0}^{\infty} = (1, 2, 4, 7, 12, 20, 33, 54, 88, 143, \dots)$$

(מומלץ לשנן בעל־פה לקראת בחני אמצע, כך שבניגוד לכותב סיכום זה לא תטעו בחיבור ותחשבו את הסכום $12 + 20 + 33 = 64$, וכתוצאה מכך תרדנה לכם 20 נקודות)

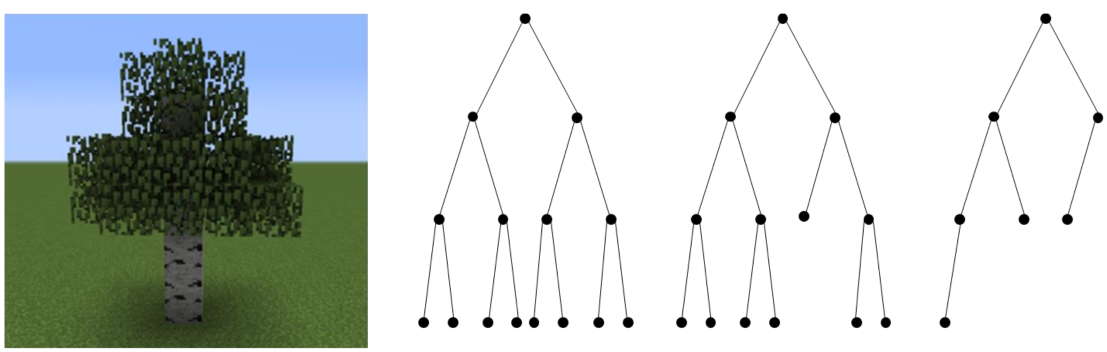
<u>הערה נוספת על עצים בינאריים</u>: עץ **מאוזן**, עץ **מלא** ועץ **שלם** אינם אותו הדבר!

עץ מלא הוא עץ בו לכל צומת שאינו עלה יש בדיוק שני בנים - לכל קדקוד יש בדיוק 0 או 2 בנים.

עץ מאוזן (AVL) הוא עץ שבו ההפרש בין עומק שני התת־עצים של כל אחד מקדקודיו הוא 0 או 1.

עץ שלם / מושלם הוא עץ <u>מלא</u> שבו כל העלים נמצאים באותה הרמה.

עץ שַׁדָּר, או לִבְנֶה (שם מדעי: Betula) הוא סוג עץ יער ממשפחת השדרים. רוב מיני השדר נפוצים באזורים צפוניים של אקלים ממוזג: צפון אסיה, אמריקה הצפונית, ויערות באירופה.



עץ שדר | עץ שלם | עץ מלא | עץ מאוזן

שרטוט 7.6: הדוגמה לארבעת סוגי העצים.

כאן נגמר החומר לבוחן האמצע.

<u>הרצאה 7 הגיעה לסיומה!</u>

הרצאה 8: ערימות מקסימום: הגדרה, תכונות ופעולות עליהן

5/5/2019

א. <u>ערימת מקסימום, ערימת מינימום וערימת ילדים</u>

(i) תור קדימויות

נניח שיש לנו 150 משימות לבצע בחודש הקרוב, ביניהן: 8 תרגילים להגיש ב־9 קורסים שונים, דוקטורט לסיים לכתוב עד מחר, מילוא טופס בקשה לקבלת אישור להגשת טופס בקשה וכן דו"ח מעבדה שעלינו לשרוף מוקדם ככל האפשר.

נרצה למצוא דרך לתעדף אותן, כך שנוכל בכל שלב לגשת לתור ולקחת את המשימה הקרובה, הראשונה בחשיבותה. מהו ה־ADT[16] המתאים? אלו הפונקציות שנרצה לממש:

API for the ADT:

```
max(A)
extract_ max(A)
insert(A, x)
increase_key(A, x, key)
```

הצעה אחת למימוש הפונקציות האלו היא באמצעות עץ חיפוש בינארי, אך לא נעסוק באופציה זו. אנו נממש את המתודות הללו באמצעות מבנה נתונים שנקרא **ערימת מקסימום (Max Heap)**, שהוא למעשה **עץ בינארי** (לא עץ <u>חיפוש</u> בינארי!) שעבורו מתקיימות שתי תכונות:

1. בכל קדקוד שיש לו אב או בן מתקיימת תכונת הערימה:

$$\text{key}(\text{parent}(x)) \geq \text{key}(x)$$

(המפתח הוא הקדימות.)

2. העץ הינו **עץ בינארי כמעט שלם**.

עץ בינארי כמעט שלם הוא עץ בינארי שכל הרמות שלו מלאות, חוץ (אולי)[17] מהרמה התחתונה, שהיא מלאה **משמאל** עד נקודה מסוימת, וימינה ממנה היא ריקה.

(ii) תכונות עץ בינארי כמעט שלם

בעץ בינארי **שלם** שגובהו d יש 2^d עלים. יש לו $2^0+2^1+\cdots+2^{d-1}=2^d-1$ קדקודים פנימיים (שאינם עלים), ו־$n=\sum_{i=0}^{d} 2^i=2^{d+1}-1$ קדקודים בסך הכול.[18]

בעץ בינארי **כמעט שלם** בגובה d ובעל n קדקודים (סך הכול, כולל העלים והקדקודים הפנימיים), מתקיים:

---

[16] Abstract Data Type.

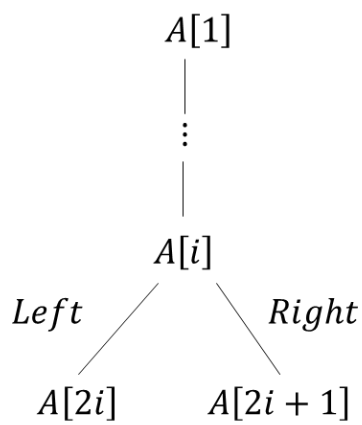
[17] גם עץ שלם הוא בפרט עץ **כמעט שלם**.

[18] שימו לב, עץ בעל קדקוד יחיד הוא בעל 0 רמות, כי העומק נמדד על־פי מספר **הצלעות**.

$$2^d \leq n < 2^{d+1} \tag{8.1}$$

מאחר שלעץ שלם בגובה d יש $2^d - 1$ קדקודים **פנימיים**, עליהם נוסיף לפחות קדקוד אחד בגובה d.
<u>הערה</u>: השוויון $n = 2^d$ מתקיים אם כך כאשר לעץ יש עלה יחיד בעומק d[19] וכפי שציינו, אם מדובר בעץ שלם, מתקיים $n = 2^{d+1} - 1$.

נניח ש-A הוא המערך השומר את המצביעים לכל אחד מהקדקודים. אז מתקיימים הקשרים הללו בין קדקודים:



שרטוט 8.1: קשר חשוב בין בנים והורים בעץ כמעט שלם.
לשם חיזוק הקשר בין בנים והורים, חוקרים אמריקניים ממליצים על דַּיִג משותף. מטעמם של [הדגים](#) נמסר שהם מעדיפים שלא.

כלומר, בהינתן מערך $A = [1, \dots, n]$, מתקיימים הקשרים הבאים עבור קדקוד $x = A[i]$:

$$A\left\lfloor {}^{i}\!/_{2} \right\rfloor = \text{parent}(x) \tag{8.2}$$

$$A[2i] = x.\text{left}() \tag{8.3}$$

$$A[2i+1] = x.\text{right}() \tag{8.4}$$

נסמן:

```
heapsize(A) = n
```

נרצה לממש את המתודות שתיארנו:

[19] נזכיר שגובה סופרים מלמטה למעלה, ועומק סופרים מלמעלה למטה.

```
def max(A):
    return A[1]
```

נממש פונקציית עזר:

```
def max_heapify(A, i):
   ℓ = left(i)   # the index of left(i)
   r = right(i)  # the index of right(i)
   if ℓ ≤ n: # check if the array contains only one node
       if r > n: # check if i has a right son
           r = n
       largest = index of max{A[ℓ], A[i], A[r]}
       if largest ≠ i:
           swap A[i], A[largest]
           max_heapify(A, largest)
```

מה הפונקציה `max_heapify` עושה? היא מתקנת ערימה החל מהאינדקס i ועד n, בהתבסס על כך ששני התת־עצים ששורשיהם $\mathrm{right}(i)$ ו־$\mathrm{left}(i)$ כבר מסודרים בערימה תקינה. נבחין שהפונקציה רצה בזמן לוגריתמי, $O(\log n)$, כיוון שמקיימת את נוסחת הנסיגה $T(n) = T\left(\frac{2}{3}n\right) + c$. זאת מאחר שלפי (8.1) היחס המקסימלי בחלוקת n קדקודים של תת־עץ ששורשו x בין שני תתי־העצים (הימני והשמאלי) של x הוא 1:2, כפי שנראה בתרגיל 7, שאלה 1.

וכעת נוכל לממש בקלות את `extract_max`:

```
def extract_max(A):
    max = A[1]
    A[1] = A[n]
    heapsize(A) -= 1
    max_heapify(A, 1)    # O(log(n))
    return max
```

(iii) הוכחת נכונות של max_heapify

ניעזר בכמה טענות עזר לשם הוכחת הנכונות של האלגוריתם `max_heapify(A, i)`.

<u>טענה 8.5</u>: כל קדקוד בעץ כמעט שלם הוא שורש של עץ כמעט שלם. (ההוכחה טריוויאלית ולכן לא נוכיח כאן).

טענה 8.6: כל קדקוד בערימת מקסימום הוא שורש של ערימת מקסימום.
הוכחת טענה 8.6: בקצרה, טענה זאת נובעת מכך שכל קדקוד בערימת מקסימום מקיים את תנאי הערימה, ומטענה 8.5.

טענה 8.7: אם נפעיל את max_heapify על קדקוד x בעץ כמעט שלם, כך ששני בניו של x שורשי ערימת מקסימום, אז אחרי max_heapify, x יהיה שורש ערימת מקסימום.

הוכחת טענה 8.7: באינדוקציה שלמה על הגובה h של התת־עץ של x.

בסיס האינדוקציה: $h = 0$, הטענה נכונה באופן ריק כי x עלה.

שלב האינדוקציה: נניח נכונות עבור כל $k \leq h$, ונוכיח עבור $h + 1$. נשים לב שאם ל־x (1) מתקיימת תכונת הערימה, (2) שני ילדיו שורשי ערימות מקסימום וכן (3) x שורש של עץ כמעט שלם, אז x שורש של ערימת מקסימום. (1) נובע מכך שאחרי max_heapify, x מכיל את המקסימום בתת־עץ, (2) נובע מהנחת האינדוקציה, ו־(3) נובע מטענה 8.5, ובזאת הוכחנו את נכונות max_heapify.

Q.E.D

כעת נרצה לבנות מתודה שמעלה את הקדימות של ערך מסוים:

```
def increase_key(A, i, key):
    if key < A[i]:
        raise error
    A[i] = key
    while i > 1 and A[parent(i)] < A[i]:
        swap A[i], A[parent(i)]
        i = parent(i)
```

הוכחת הנכונות של אלגוריתם זה תיעשה בתרגיל, והיא דומה מאוד להוכחת הנכונות עבור max_heapify. נממש את המתודה insert, המכניסה איבר חדש לערימה:

```
def insert(A, x):
    heapsize(A) += 1
    A[heapsize] = -∞ # לא באמת מינוס אינסוף, אבל מספר שלילי מאוד
    increase_key(A, heapsize(A), key)
```

הנכונות של insert נובעת מהנכונות של increase_key.

עד כה לא נראה שהפקנו תועלת ממבנה הנתונים החדש, ושעץ חיפוש בינארי יעיל יותר. ובכן, איפה הקאץ'?[20] על־מנת לבנות עץ חיפוש בינארי עם n קדקודים, נדרש זמן ריצה של $O(n\log n)$, אולם בערימת מקסימום ניתן לבצע זאת בזמן לינארי:

```
def Buildheap(A):
    for i = ⌊n/2⌋ to 1: # מתחילים מהקדקוד הראשון שאינו עלה ועולים לשורש
        max_heapify(A, i)
```

מדוע פונקציה זו רצה בסיבוכיות זמן $O(n)$? היינו מצפים לסיבוכיות זמן ריצה של $O(n\log n)$. אחרוג כאן מתפקידי ואשתמש בסיכום התרגול על־מנת להסביר נקודה עדינה זו:

> נשים לב לשתי נקודות חשובות שהאלגוריתם מתבסס עליהן ומהן נובעת נכונותו. בניגוד לאינטואיציה איננו רצים על כל הקדקודים, וכמו־כן איננו מבצעים איטרציה בסדר עולה. מדוע? נתחיל מהחלק הפשוט המתבסס על הפונקציה `max_heapify(A, i)` - פונקציה זו יוצרת ערימה תקינה החל מהאינדקס ה־i והלאה, בהתבסס על כך שעבור הקדקודים שהאינדקס שלהם גדול ממש מ־i מהווים כבר ערימת מקסימום (אחרת לא נקבל בהכרח ערימה.) עובדה זו מחייבת אותנו להתחיל מן הסוף ולעלות במעלה העץ בחזרה להתחלה, כשאנחנו יודעים שהקדקודים שתחתינו מקיימים את תכונת הערימה. כמובן שעבור כל קלט שנקבל, אנו יודעים בוודאות שהעלים מקיימים את תכונת הערימה. כאן מסתתרת הנקודה השנייה - אין צורך להפעיל את `max_heapify(A, i)` על כל הקדקודים, אלא רק על קדקודים שהילדים שלהם מקיימים את תכונת הערימה. לכן מספיק להריץ אותה החל מהאבא של העלה האחרון ומעלה. פעולה זו תבטיח לנו שבכל שלב כל הקדקודים שכבר עברנו עליהם מקיימים את תכונת הערימה. מהו האינדקס של האבא הראשון? בדיוק `⌊n/2⌋` , לפי (8.2).

כעת נוכיח טענה זו באופן פורמלי, ותופתעו מפשטות ההוכחה.

טענה 8.8: `Buildheap` רץ בסיבוכיות זמן של $O(n)$.[21]

הוכחת טענה 8.8: בעץ בעל n קדקודים יש לכל היותר $\frac{n}{2^d}$ קדקודים בגובה d. נסכום את הסיבוכיות לפי כל הגבהים. אנו נראה **שאפילו אם הפונקציה הייתה רצה על כל** המערך, זמן הריצה עדיין היה לוגריתמי (c קבוע כלשהו):

$$\sum_{d=0}^{\log n}\underbrace{\frac{n}{2^d}}_{\text{מספר קדקודים}}\cdot\underbrace{cd}_{\text{תרומה לסיבוכיות}}=cn\sum_{d=0}^{\log n}\frac{d}{2^d}\leq cn\sum_{d=0}^{\infty}d\left(\frac{1}{2}\right)^d=\cdots$$

---

[20] הוא נמצא גם כאן.

[21] כמה קישורים מועילים בנושא: פירוט נוסף על ערימות מקסימום תוכלו למצוא כאן וכאן.

נשתמש, כרגיל, בטריק מאינפי,[22] לפיו גזירה של טור שקולה לסכימת הנגזרות (עבור טור מתכנס):

$$\sum_{d=0}^{\infty} dx^d = x\sum_{d=0}^{\infty} dx^{d-1} = x \cdot \frac{d}{dx}\left(\sum_{d=0}^{\infty} x^d\right) = x \cdot \frac{d}{dx}\left(\frac{1}{1-x}\right) = \frac{x}{(1-x)^2}$$

ובמקרה זה, $x = \frac{1}{2}$:

$$\ldots = cn \cdot \frac{{}^{1}/_{2}}{\left(1-{}^{1}/_{2}\right)^2} = 2cn = O(n)$$

$\mathcal{Q.E.D}$

הרצאה 8 הגיעה לסיומה!

[22] חפשו כאן, או לחילופין, בקשו מפיזיקאי מחמד להראות לכם את פרקים 5 ו־6 של מת"פ 2. הנאה מובטחת!

הרצאה 9: קוד האפמן ופתיח לפונקציות גיבוב

12/5/2019

א. קוד האפמן: שימוש בערימות מינימום לשם דחיסת מידע

ערימת מינימום היא כמו ערימת מקסימום, אלא שהפעם כל קדקוד קטן מבניו וגדול מאביו (אם יש כאלו). נראה שימוש מעניין (ושימושי) בערימות מינימום, הקשור לתחום הקידוד - התחום שעוסק בשכתוב תוכן, הכתוב באלפבית אחד, לאלפבית אחר (או לעיתים לאותו אלפבית, אך באופן שונה). במדעי המחשב מדובר לרוב, כמובן, בקוד בינארי - כלומר האלפבית הוא $\{0,1\}$.

דחיסת מידע על־ידי קוד האפמן

נניח שיש לנו אלפבית עם 22 אותיות. נוכל אם כן לקדד כל אות בקוד בינארי בן 5 ספרות:

א - 00000

ב - 00001

ג - 00010

וכן הלאה. נוכל לקודד את קובץ סיכומים זה באמצעות רצף בינארי המורכב ממקטעים בני חמש ספרות, אך הבעיה היא שצורת קידוד זו אינה יעילה: הרי האותיות ו' וי' מופיעות בתדירות גבוהה הרבה יותר מאשר אותיות כמו צד"י וקו"ף.

נרצה להגדיר מדד ליעילות הקידוד. נגדיר את התוחלת כאורך הכולל של הקידוד, חלקי מספר האותיות באלפבית, ששקול לביטוי זה:

$$L(\text{קידוד}) = \sum_{\sigma \in \Sigma} \ell(\sigma) \cdot f(\sigma)$$

כאשר Σ מייצג את האלפבית,[23] $f(\sigma)$ היא שכיחות האות בשפה (ליתר דיוק: שכיחותה בקוד כולו), $\ell(\sigma)$ אורך הקידוד של האות. קוד האפמן שייך למשפחת קידודים חסרי תחיליות (חסרי רישא): אין אף אות שקידודה מהווה התחלה של קידוד של אות אחרת, ולכן הקוד הוא בר פיענוח. לדוגמה:

A – 0

B – 101

C – 100

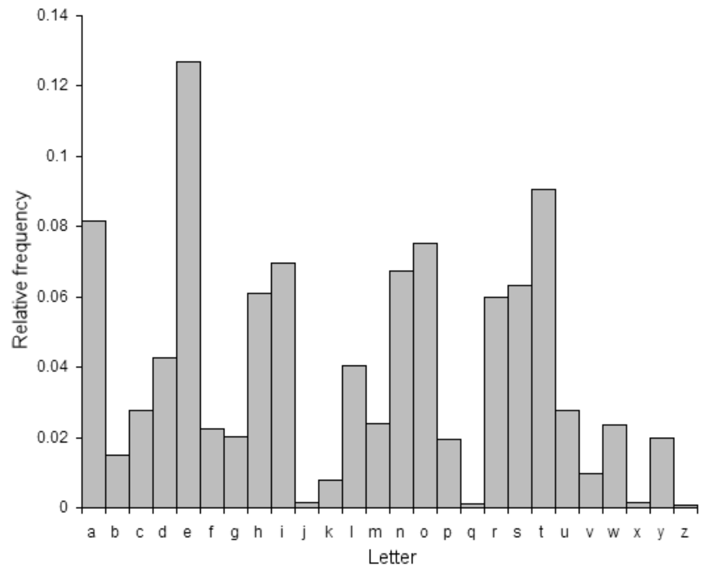
D – 111

E – 1101

F – 1100

---

[23] למי שהתבלבל מעודף סיגמות, שיזכור שהתוחלת של הקידוד היא הסיגמה על הסיגמה שבסיגמה של ℓ של סיגמה כפול f של סיגמה.



שרטוט 9.1: התפלגות האותיות באנגלית.
האות e היא כמובן הנפוצה ביותר, z הנדירה ביותר.

נוכל להתאים לקידוד חסר תחיליות עץ בינארי, שבעליו יש אותיות (ראו שרטוט 9.2). נרצה להתאים לאותיות הנפוצות יותר קידוד קצר יותר, כך שהקוד עצמו יהיה חסכוני יותר. נניח שנתונה שפה בעלת שש אותיות, $A, \dots, F$ עם שכיחויות כדלקמן:

A – 45%

B – 13%

C – 12%

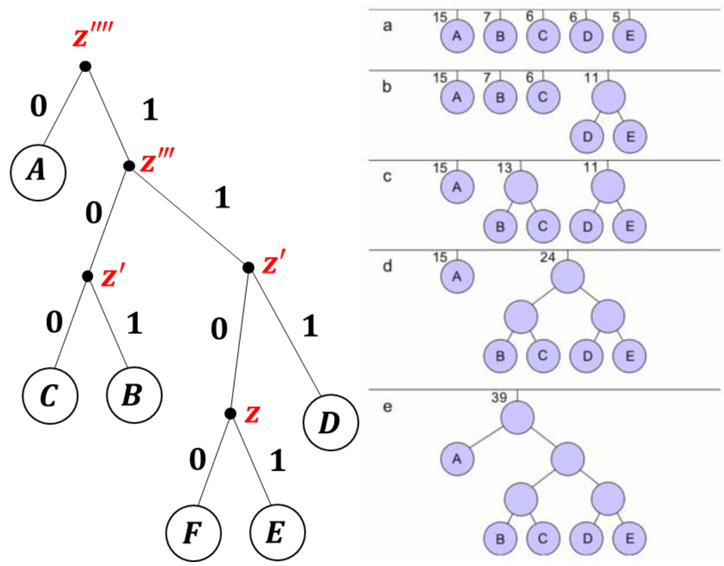
D – 16%

E – 9%

F – 5%

נקודד לפי דרישות אלו באמצעות ערימת מינימום:

```
def Huffman(C):
    n = |C| # size of the alphabet
    Q = build_min_heap({c in C})
    for i = 1 to n - 1: # We add n - 1 nodes for n letters (Why?)
        allocate a new node z
        left(z) = extract_min(Q)
        right(z) = extract_min(Q)
        key(z) = key(left(z)) + key(right(z))
        insert(Q, z)
    return min(Q)
```



שרטוט 9.2: משמאל: דוגמה לעץ קידוד בינארי, עבור המקרה של שש האותיות המתוארות לעיל.
מימין: דוגמה נוספת לבניית עץ האפמן ([ויקיפדיה](#)).

<u>**טענה 9.1:**</u> קוד האפמן הוא קוד ללא תחיליות אופטימלי. כלומר עבור אלף־בית C, אם T העץ המיוצר על־ידי אלגוריתם האפמן, אזי לכל T' מתקיים $L(T') \geq L(T)$. לשם הוכחת טענה זו ניעזר בשלוש טענות עזר.

<u>**טענה 9.2:**</u> אם T קוד ללא תחיליות אופטימלי, אזי לכל a, b מתקיים:

$$f(a) < f(b) \Longrightarrow \ell(a) \geq \ell(b)$$

זו טענה אינטואיטיבית מאוד, אבל ההוכחה של טענה זו תיעשה בתרגול.

<u>**הגדרה 9.3:**</u> עץ בינארי ייקרא **עץ מלא** אם לכל קדקוד יש בדיוק 0 או בדיוק 2 בנים.

<u>**טענה 9.4:**</u> אם $|C| \geq 2$ (זאת אומרת, אם יש יותר מאות אחת באלף־בית), אז קיים T אופטימלי שבו שתי האותיות בעלות השכיחות הנמוכה ביותר הן עלים בעץ ובנים לאותו הקדקוד, ברמה התחתונה.
<u>**הוכחת טענה 9.4:**</u> (א) בקוד אופטימלי, T עץ מלא. (ב) מטענה 9.2 ומסעיף א' נובע ששתי האותיות הללו חייבות להיות ברמה התחתונה. (ג) אם אינן אחים, נחליף אותן כך שיהיו. Q.E.D

<u>**טענה 9.5:**</u> יהי C אלף־בית המקודד על־ידי T. נניח ששתי האותיות הנדירות ביותר, x, y, אחים בעץ ועלים. נגדיר את T' להיות העץ ממנו מסירים את x, y. זהו עץ שמקודד את C', שבו החלפנו את x, y ב־z המוגדר $f(z) = f(x) + f(y)$. אזי $L(T) = L(T') + f(z)$.
<u>**הוכחת טענה 9.5:**</u> ההוכחה למעשה פשוטה יותר מניסוח הטענה:

$$\begin{aligned} L(T) &= \sum_{c\in C\setminus\{x,y\}} f(c)\ell(c) + f(x)\ell_T(x) + f(y)\ell_T(y) \\ &= \sum_{c\in C\setminus\{x,y\}} f(c)\ell(c) + f(x)(\ell_T(z)+1) + f(y)(\ell_T(z)+1) \\ &= \underbrace{\sum_{c\in C\setminus\{x,y\}} f(c)\ell(C) + \big(f(x)+f(y)\big)\ell_T(z)}_{L(T')} + \underbrace{f(x)+f(y)}_{=f(z)} \end{aligned}$$

$\mathcal{Q.E.D}$

כעת נוכיח את הטענה המרכזית.

__הוכחת טענה 9.1__: באינדוקציה על $|C|$.

__מקרה בסיס__: אם $|C| = 2$, קוד האפמן ייתן 0 ו־1, והוא אופטימלי.

__שלב האינדוקציה__: נניח שהאלגוריתם מייצר קוד אופטימלי עבור $|C| < k$, ונוכיח עבור k. יהי T קידוד האפמן, ונניח בשלילה שקיים W שעבורו $L(W) < L(T)$ ו־W אופטימלי. מ־(9.4) נקבל ששתי האותיות הנדירות x, y אחים ברמה התחתונה של W. לפי הבנייה של קוד האפמן, x, y גם הם אחים בעץ. לפי (9.5) נקבל $L(W) = L(W') + f(z)$, וכן $L(T) = L(T') + f(z)$, כאשר W', T' הוא העץ לאחר שמסירים ממנו את הקדקודים x, y. נסיק מהנחת השלילה ש־$L(W') < L(T')$, וקיבלנו סתירה להנחת האינדוקציה, מאחר ש־T' עץ האפמן.[24] $\mathcal{Q.E.D}$

ב. __פתיח לפונקציות גיבוב (Hash Functions)__

__הגדרה 9.6__: **פונקציות גיבוב** (או: **טבלאות גיבוב**, **פונקציות ערבול**, **פונקציות תמצות** ואף **פונקציות טחינה**) מוגדרות בתור נס קסום ופלאי. הן מצליחות לעשות הכול בזמן ממוצע של $O(1)$, ולמרבה ההפתעה, שלא כמו חדי קרן [ומערכות אקסיומות עקביות ושלמות](), הן גם קיימות במציאות.

אנו רוצים לממש את המתודות הללו ב־$O(1)$:

```
insert(H, k)
search(H, k)
delete(H, k)
```

__הגדרה פורמלית יותר__: נניח שיש עולם מפתחות עצום שנסמן ב־U. פונקציית גיבוב היא פונקציה שממירה כל מפתח לקלט קצר בהרבה, באורך קבוע, כמו ממספרת את הערכים הרלוונטיים בְּמערך. גישה אחת לטבלאות גיבוב היא יצירת מערך בגודל של U, כך שהאינדקס ה־k הוא האיבר המתאים, אך פתרון זה מבזבז כמות עצומה של זיכרון.

[24] מדוע הוא עץ האפמן? הביטו שוב באלגוריתם המתואר ותבינו.

שיטת החלוקה

כעיקרון נרצה שפונקציית גיבוב תיתן פלט זהה לקלטים שונים בהסתברות נמוכה מאוד. אנו נראה פתרון פשוט (וגרוע) להגדרת פונקציית הגיבוב $h: U \to A$, השולחת מפתח כללי למיקום במערך. שיטת החלוקה היא פונקציה $h(k) = k \pmod n$ עבור מערך בגודל n. הבעיה במקרה זה היא התנגשות: $h(k) = h(k+n)$.[25] ניתן לקחת קלט של כפולות שלמות של n, ונקבל שהפונקציה אינה יעילה במקרה זה.

שיטת ההכפלה

פתרון אחר לבעיה זו הוא:

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

כאשר $A = \varphi = \frac{1+\sqrt{5}}{2}$ לדוגמה, או מספר אי־רציונאלי אחר. זה אולי נראה יותר טוב, אך גם במקרה זה תהיינה התנגשויות. למעשה, בכל מקרה תהיינה:

<u>**טענה 9.7:**</u> לכל n, $h: U \to \{0, \dots, m-1\}$, $|U| > Nm$, קיימת קבוצת מפתחות בגודל לפחות N שנשלחת לתא אחד בטבלה.

<u>**הוכחת טענה 9.7:**</u> משובך היונים זה נובע מיד. אילו לא היו קיימים N ערכים כאלו, היינו מקבלים שלכל תא יש פחות מ־$N-1$ מקומות ולכן $|U| \leq m(N-1)$ בסתירה. $\mathcal{Q.E.D}$

<u>*הרצאה 9 הגיעה לסיומה!*</u>

[25] הפתרון להתנגשות כזו עשויה להיות רשימה מקושרת, כפי שנראה בתרגיל 4 ב־OOP וכפי שנראה בתרגול, וכמובן שנראה בהמשך פתרונות נוספים.

הרצאה 10: פונקציות גיבוב

19/5/2019

א. גיבוב פשוט אחיד

<u>הגדרה 10.1</u>: נאמר על פונקציה $h: U \to \{0, \dots, m-1\}$ שהיא מקיימת את הנחת **גיבוב פשוט אחיד** אם h שולחת כל מפתח בקלט $S \subseteq U$ כלשהו למקום כלשהו בטבלה בהסתברות אחידה, כלומר: $\forall x \in S\colon\ \Pr[h(x) = i] = \frac{1}{m}$ לכל $i \in \{0, \dots, m-1\}$.

<u>טענה 10.2</u>: תחת הנחת גיבוב אחיד ופשוט, מתקיים שלכל $x \in S$, תוחלת אורך הרשימה שבה x נמצא היא $\mathbb{E}\big(\ell(x)\big) \le 1 + \frac{N}{m}$.

<u>הוכחת טענה 10.2</u>: ניעזר במשתנים מציינים (אינדיקטורים). לכל $x, y \in S$ נגדיר את האינדיקטור הבא, המציין אם הייתה התנגשות בין x ל־y:

$$C_{xy} = \begin{cases} 1 & h(x) = h(y) \\ 0 & h(x) \neq h(y) \end{cases}$$

אז אורך הרשימה שבה x נמצא הוא:

$$\ell(x) = \underbrace{1}_{x\text{ ברשימה}} + \underbrace{\sum_{y \in S, y \neq x} C_{xy}}_{\substack{\text{ה־}y\text{ שנשלחים} \\ \text{לאותה רשימה}}}$$

ומכאן נחשב את תוחלת אורך הרשימה שבה x נמצא באמצעות אינדיקטורים אלו:

$$\mathbb{E}\big(\ell(x)\big) = \mathbb{E}(1) + \sum_{y \in S, y \neq x} \mathbb{E}\big(C_{xy}\big) = 1 + (N-1) \cdot \frac{1}{m} \le 1 + \frac{N}{m}$$

ל-$\frac{N}{m}$ נקרא **פקטור העומס**.

ב. משפחה אוניברסלית של פונקציות גיבוב

לפתירת הבעיה המתוארת, מאחר שלא קל למצוא h מסוים שמקיים את תכונת הגיבוב הפשוט והאחיד, ניקח משפחה אוניברסלית של פונקציות גיבוב (אך לא גדולה מדי, לשם מניעת בזבוז זיכרון).

<u>הגדרה 10.3</u>: הקבוצה:

$$\mathcal{H} = \big\{h: U \to \{0, \dots, m-1\}\big\}$$

תיקרא **משפחה אוניברסלית של פונקציות גיבוב** אם לכל $x \neq y$, $x, y \in U$ מתקיים, בהגרלת פונקציה $h \in_R \mathcal{H}$ באקראי:

$$\Pr_{h \in_R \mathcal{H}}\big(h(x) = h(y)\big) \leq \frac{1}{m}$$

__טענה 10.4__: אם $\mathcal{H}$ משפחה אוניברסלית של פונקציות גיבוב, אז מתקיים שלכל $x \in U$ תוחלת אורך הרשימה בה x נמצא הינה $\mathbb{E}\big(\ell(x)\big) \leq 1 + \frac{N}{m}$, כאשר בחרנו $h \in \mathcal{H}$.

__הוכחת טענה 10.4__: ההוכחה למעשה זהה להוכחה מקודם.

דוגמה למשפחה אוניברסלית

אם התגעגעתם לאלגברה לינארית, עתה תפסיקו להתגעגע. נניח $|U| = 2^\ell$, $m = 2^r$ וכן $\ell > r$. נגדיר את המשפחה כאוסף המטריצות $\mathcal{H} = \mathcal{M}_{r \times \ell}(\mathbb{Z}_2)$, כלומר אוסף המטריצות הבינאריות בגודל r על ℓ, לדוגמה:

$$\begin{pmatrix} 0 & 0 & \cdots & 1 & 0 \\ 1 & 1 & \cdots & 1 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 1 \end{pmatrix} \in \mathcal{H}$$

כיצד מתרגמים זאת לפונקציה? ניקח מפתח ב־U המתואר באמצעות ℓ ביטים של 0 ו־1, ונכפול את המטריצה בווקטור על־מנת לקבל את מיקום המפתח בטבלה, כלומר:

$$h \in \mathcal{H}\colon\ \{0,1\}^\ell \to \{0,1\}^r$$

מוגדרת באמצעות כפל של הווקטור המייצג את $u \in \{0,1\}^\ell$ במטריצה M_h המתאימה ל־h.

__טענה 10.5__: המשפחה $\mathcal{H}$ של מטריצות אלו מעל $\mathbb{Z}_2$[26] היא משפחה אוניברסלית של פונקציות גיבוב.

__הוכחת טענה 10.5__: נרצה להראות שאם נקבע $x, y \in U = \{0,1\}^\ell$ שני מפתחות שונים, ונבדוק מהי ההסתברות שיישלחו לאותה רשימה, נקבל שהיא פחותה מ־$\frac{1}{m}$. בכתיב מתמטי, עלינו להוכיח:

$$\forall x, y \in U \quad x \neq y \Longrightarrow \Pr\big(h(x) = h(y)\big) \leq \frac{1}{m}$$

נזכור שמתקיים:

$$M_n(x) = M_n(y) \Leftrightarrow M_n(x - y) = 0$$

ומכאן נסיק שבהינתן $0 \neq z \in \{0,1\}^\ell$, מספיק להראות שמתקיים:

$$\Pr_{h \in \mathcal{H}}\big(M_n(z) = 0\big) \leq \frac{1}{m}$$

נסמן את העמודות של M_n כ־h_1, h_2 עד h_ℓ. אם z היה שווה 0 בכל מקום פרט לקואורדינטה ה־j, אז נקבל ש־ $M_n z = h_j$:

[26] נזכיר שבשדה זה מתקיים $1 + 1 = 0$, ובכלליות $x + y = \mathrm{xor}(x, y) = x + y \pmod 2$, והכפל כמו בשלמים.

$$M_n z = \begin{pmatrix} | & | & & | \\ h_1 & h_2 & \cdots & h_\ell \\ | & | & & | \end{pmatrix} \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix} = h_j$$

אז למעשה נוכל לכתוב את המכפלה כך:

$$M_n z = \sum_{i \mid z_i \neq 0} h_i = h_{j_0} + \sum_{\substack{i \mid z_i \neq 0 \\ i \neq j_0}} h_i = 0$$

נשאל: מהי ההסתברות $\Pr(h_{j_0} = u)$, כאשר u וקטור כלשהו באורך r?

ההסתברות היא בדיוק $\left(\frac{1}{2}\right)^r = \frac{\mathbf{1}}{\boldsymbol{m}}$, כי אנחנו צריכים שהקואורדינטות ישתוו בכל אחת מהקואורדינטות, כאשר בכל אחת מהן יש הסתברות $\frac{1}{2}$ שהן תהיינה זהות, וכמובן שיש r קואורדינטות כאלו, ובזאת סיימנו את ההוכחה. $\mathcal{Q.E.D}$

נראה דוגמות נוספות למשפחות אוניברסליות בתרגול.

> שימו לב: אנו רוצים להוכיח שבהסתברות טובה מאוד, **לכל קלט** נקבל זמן ריצה טוב. זה שונה מאוד מלומר ש**רוב הקלטים** נקבל בתוחלת זמן ריצה טוב. זוהי נקודה עדינה שמזכירה את הדיון בהבדל בין Quick Sort האקראי והדטרמיניסטי.

ג. גיבוב מושלם

ציטוט לפתיחת פרק זה:

"אף אחד אינו מושלם. חוץ מגיבוב."

- אריסטו

אבל לשלמות יש מחיר, ונראה אותו בהמשך.

גיבוב מושלם

מהו אותו גיבוב מושלם? נרצה שלא תהיינה התנגשויות **בכלל**. לעצמנו לשנות את h לפי הקלט.

הגדרה 10.6: פונקציה h תיקרא **גיבוב מושלם** אם עבור קלט S של N של מפתחות מ־U, נקבל שעבור כל $x \neq y$, $x, y \in S$, מתקיים $h(x) \neq h(y)$.

בתור התחלה, נאפשר ל־m להיות ריבועי ב־N. נבחר $N^2 \leq m < 2N^2$, שיש עבורו משפחה אוניברסלית $\mathcal{H} = \{h \mid h: U \to \{0, \dots, m-1\}\}$. נבחר באקראי $h \in_R \mathcal{H}$.[27]

[27] הסימון $\in_R$ פירושו בחירה אקראית (בהסתברות אחידה) מתוך הקבוצה.

<u>הגדרה 10.7</u>: נגיד ש־h פונקציית גיבוב "**טובה**" אם אין בה התנגשויות:

$$\forall x, y \in S, x \neq y:\ h(x) \neq h(y)$$

כאשר $|S| = N$.

<u>טענה 10.8</u>: ההסתברות ש־h טובה גדולה או שווה ל־$\frac{1}{2}$.

<u>הוכחת טענה 10.8</u>: הטענה שקולה לכך שלכל היותר $\frac{1}{2}$ מהפונקציות $h \in \mathcal{H}$ לא טובות. ישנם $\binom{N}{2} = \frac{N(N-1)}{2}$ זוגות של קלטים x, y. יש לכל היותר $\frac{1}{m}$ פונקציות שגורמות להתנגשות של x ו־y. נפסול באופן זה עוד פונקציות, ונקבל שלכל היותר $\binom{N}{2} \cdot \frac{1}{m}$ אינן טובות, ומאחר ש־$m \geq N^2$:

$$\frac{N(N-1)}{2m} = \binom{N}{2} \cdot \frac{1}{m} \underset{m \geq N^2}{\leq} \frac{N(N-1)}{2N^2} \leq \frac{1}{2}$$

תיאור אלגוריתם גיבוב מושלם במקום ריבועי

אז מה האלגוריתם המתאים? נגריל את h ונבדוק ב־$O(n)$ אם היא טובה. מאחר ש־h טובה בהסתברות שגדולה מ־$\frac{1}{2}$, תוחלת מספר ההגרלות היא 2:

$$\sum k \cdot \Pr(\#\text{ההגרלות} = k) = \sum_{k=1}^{\infty} k \left(\frac{1}{2}\right)^k = 2$$

כאשר השוויון האחרון זהה לזה שב־(8.8) (טריק מת"פי לסכום של טור באמצעות גזירה איבר־איבר).

<u>הרצאה 10 הגיעה לסיומה!</u>

הרצאה 11: גיבוב מושלם במקום לינארי ומבוא לגרפים

26/5/2019

א. טבלאות גיבוב: פינאלה וקודה – גיבוב מושלם במקום לינארי

תזכורת מההרצאה הקודמת: גיבוב מושלם במקום ריבועי

נזכיר: גיבוב מושלם במקום ריבועי הוא גיבוב בעל שתי שאיפות: הראשונה היא שאין שום התנגשות בין שני קלטים (ועל כן חיפוש מתבצע ב-$O(1)$ **תמיד**, בהינתן הקלט), והשנייה היא שגודל הטבלה צורך זיכרון מסדר גודל $\Theta(N^2)$, כאשר N מספר המפתחות.

ראינו שחיפוש ב-$O(1)$ לכל מפתח בקלט אפשרי ומובטח, אך לשם כך יש להתאים את h לקלט. ראינו גיבוב מושלם במקום ריבועי, כלומר $N^2 \leq m < 2N^2$, ושאם בוחרים באקראי פונקציה $h \in_R \mathcal{H}$ מתוך משפחה אוניברסלית $\mathcal{H}$, ההסתברות ש-h "טובה" עבור הקלט הספציפי (בעל n המפתחות) גדולה או שווה ל-$\frac{1}{2}$, לכן התוחלת של מספר הניסיונות היא 2. כמו כן ראינו בתרגול 10 שזמן הבנייה לינארי, כלומר שהזמן הנדרש לאתחול מבנה נתונים זה הוא $O(n)$. ננסה כעת לבחון את המקרה של גיבוב לינארי.

גיבוב מושלם במקום לינארי

על-מנת ליישם גיבוב זה נחלק את התהליך לשני שלבים. נניח שנתון עולם U ותת-קבוצה של N מפתחות מ-U. נגדיר תחילה את טבלת הגיבוב בגודל $m = N$ עבור הטבלה הראשונית (צריכת מקום לינארית. בהמשך ניקח ערך אחר, $m < 2N$). נבחר באקראי $h \in_R \mathcal{H}$ פונקציית גיבוב מתוך משפחה אוניברסלית $\mathcal{H}$, ונגדיר את הגודל n_j כמספר האיברים ש-h שולחת לתא ה-j. כעת על כל תא j נגדיר גיבוב מושלם במקום ריבועי. הרעיון הוא שאפילו שהגיבוב המושלם הוא במקום ריבועי, מאחר שהוא פועל על מספר קטן של איברים, הגיבוב הכולל יהיה במקום לינארי. נוכיח זאת בהמשך. לכל j נבנה טבלת גיבוב שגודלה m_j, בשיטת גיבוב מושלם במקום ריבועי, כלומר $n_j^2 \leq m_j < 2n_j^2$. אנו למעשה בונים מערך במקום לינארי של טבלאות גיבוב במקום ריבועי.

__טענה 11.1__: זמן החיפוש של איבר במבנה שהוגדר לעיל הוא $O(1)$.

__הוכחת טענה 11.1__: נסמן את פונקציות הגיבוב המושלם לכל אחת מהטבלאות שבמערך הראשי כ-$h_0, \ldots, h_{m-1}$. בתא ה-j בטבלה שמורה טבלת הגיבוב שגודלה m_j, וכן פונקציית גיבוב h_j המחפשת במערך ה-j בסיבוכיות זמן של $O(1)$. הפונקציה h מפנה כל איבר לתא המתאים במערך בזמן $O(1)$ לפי אופן בניית המערך, ולכן הזמן הכולל של חיפוש האיבר הוא $O(1)$. Q.E.D

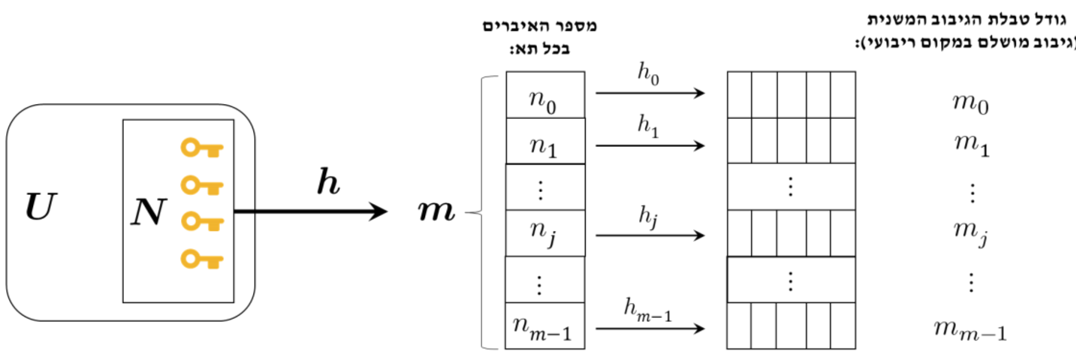
נשים לב שאם בוחרים $h \in_R \mathcal{H}$ באקראי, כאשר $\mathcal{H}$ היא משפחה אוניברסלית של פונקציות $h: U \to \{0, \dots, m-1\}$, אז המקום שיידרש למבנה הנתונים המתואר הוא (ראו שרטוט 11.1):

$$S(N) = \underbrace{m}_{\text{גודל הטבלה ההתחלתית}} + \underbrace{\sum_{j=0}^{m-1} m_j}_{\text{גודל כל אחת מהטבלאות הקטנות}}$$

ומאחר שכל אחת מהטבלאות המשניות היא טבלת גיבוב מושלם במקום ריבועי, מתקיים $m_j < 2n_j^2$.

אנו לוקחים עתה את גודל הטבלה המקורית, m, כך ש-$N \le m < 2N$, ולכן:

$$S(N) \le 2N + \sum_{j=0}^{m-1} 2n_j^2$$



שרטוט 11.1: עזר ויזואלי לאופן הפעולה של גיבוב מושלם במקום לינארי.

תיאור אלגוריתם הגיבוב המושלם במקום לינארי

נתאר עתה את האלגוריתם במלואו:

1. נבחר $h \in_R \mathcal{H}$ באקראי ממשפחה אוניברסלית $\mathcal{H}$ מ-U ל-$\{0, \dots, m-1\}$. נזכיר שאנו עתה בוחרים את m כך ש-$N \le m < 2N$.
2. נבדוק אם גודל הזיכרון הדרוש למבנה שתיארנו, $S(N) \le N + \sum_{j=0}^{m-1} 2m_j^2$, קטן מ-$12N$, חסם לינארי שרירותי. לפי מרקוב ובהתבסס על (11.2), יש סבירות שגדולה מ-$\frac{1}{2}$ שסעיף זה מתקיים. הבדיקה לוקחת $O(N)$ (נראה בתרגול, זה לא מסובך). נקבל גיבוב מושלם במקום $S(N)$, שבהכרח קטן מ-$12N$, ולכן לינארי. אם אכן עבור h מסוימת מתקיים $S(N) \le 12N$, נאמר ש-h, כמו תשחצים ושחמט עיוור, **טובה לזיכרון**.
3. ברגע ש-h מסוימת טובה לזיכרון, נדבק בה. נפעיל גיבוב מושלם במקום ריבועי לכל אחת מהטבלאות המשניות - בכל טבלה ננסה h_j עד שנצליח למצוא פונקציה מתאימה.

את הוכחת תוחלת **זמן הריצה** הלינארית של אלגוריתם זה נראה בתרגול. כעת נראה שתוחלת **המקום** לינארית.

<u>טענה 11.2</u>: $\mathbb{E}\big(S(N)\big) = O(N)$

<u>הוכחת טענה 11.2</u>: נגדיר C_{xy} משתנה מציין:

$$C_{xy} = \begin{cases} 1 & h(x) = h(y) \\ 0 & h(x) \neq h(y) \end{cases}$$

אז מספר הזוגות (כולל כפילויות, כי סוכמים את C_{ij} וגם את C_{ji}) שיש בהם התנגשות הוא:

$$\sum_{x,y} C_{xy} = \sum_{j=0}^{m-1} \begin{pmatrix} \text{מספר הזוגות} \\ \text{שמתנגשים} \\ \text{בתא ה־}j \end{pmatrix} = \sum_{j=0}^{m-1} n_j^2$$

ומכאן נסיק:

$$\mathbb{E}\left(\sum_{x,y} C_{xy}\right) = \cdots$$

נפצל את התוחלת לסכום התוחלות על האיברים $x = y$ ו־$x \neq y$. יש N איברים, ועבור כל אחד מהם $C_{ii} = 1$, אז נקבל סך הכול N זוגות של איבר־עם־עצמו שמתנגשים. נזכור שההסתברות שפונקציה h במשפחה אוניברסלית תשלח שני ערכים שונים לאותו ה־Hash היא פחות מ־$\frac{1}{m}$, ולכן מאחר שישנם $N(N-1)$ זוגות $x \neq y$:

$$\ldots = N + \sum_{\substack{x,y \\ x \neq y}} \underbrace{\mathbb{E}\big(C_{xy}\big)}_{\leq \frac{1}{m}} \leq N + \frac{N(N-1)}{m} \underset{m \geq N}{\leq} 2N$$

קיבלנו שהתוחלת $\mathbb{E}\big(\sum n_j^2\big) \leq 2N$, ולכן:

$$\mathbb{E}\big(S(N)\big) \leq 2N + 2\mathbb{E}\big(\sum n_j^2\big) \leq 6N$$

Q.E.D

ב. <u>מבוא לגרפים</u>

גרפים הוא מבנה הנתונים הכי מעניין במדעי המחשב (לפי דורית) והוא אמור לעניין אתכם כי הוא אחד הנושאים הכי נשאלים במבחנים, והשאלות לגבי מבנה זה לעיתים אינן כל כך פשוטות.

הגדרת הגרף הלא מכוון והמכוון

ראיתם כבר בדיסקרטית את ההגדרה, אבל הנה היא שוב.

<u>הגדרה 11.3</u>: תהי V קבוצה ו־$E \subseteq \big\{\{v,u\} \mid v,u \in V\big\}$ קבוצה סימטרית של זוגות קדקודים, כלומר (v,u) ו־(u,v) הן אותה הצלע (פורמלית, הצלע היא הקבוצה $\{v,u\}$, אך נדבוק בסימון (v,u)). הזוג הסדור $G = (V,E)$ ייקרא **גרף לא מכוון**.

<u>הגדרה 11.4</u>: תהי V קבוצה ו־$E \subseteq V \times V$. הזוג הסדור $G = (V, E)$ ייקרא **גרף מכוון**. זאת אומרת, כעת מסירים את ההגבלה על הסימטריות של E. נשים לב ש־E תת־קבוצה של זוגות **סדורים**, בניגוד לגרף לא מכוון, שם E קבוצה של קבוצות קדקודים, כלומר (v, u) ו־(u, v) הן **לא** אותה הצלע.

מסמנים לרוב: $|V| = n$, $|E| = m$, ובגרף פשוט לא מכוון (גרף שבו אוסרים על צלעות כפולות ולולאות), מתקיים $m \leq \binom{n}{2}$.

<u>הגדרה 11.5</u>: מטריצת שכנויות של גרף היא מטריצה בגודל $n \times n$: $A = [C_{ij}]$, כאשר $C_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{else.} \end{cases}$, כלומר מטריצה שמציגה 1 אם בין הקדקודים קיימת צלע, ו־0 אם לא. בגרף לא מכוון המטריצה הזו היא כמובן סימטרית, כי $(i,j) \in E \Leftrightarrow (j,i) \in E$.

<u>הגדרה 11.6</u>: רשימת שכנויות של קדקוד היא רשימה מקושרת של כל שכניו.

ננסה כעת לכתוב אלגוריתם שייבחן אם גרף הוא קשיר. נתחיל בגרפים לא מכוונים. האלגוריתם הראשון שנעסוק בו הוא DFS (Depth First Search). נגדיר את הפונקציה `Explore`:

```
def Explore(v):
    visited[v] = 1
    for all w so that (v, w) ∈ E:
        if visited[w] == 0:
            Explore(w)
```

<u>טענה 11.7</u>: `Explore` מגיע לכל הקדקודים בגרף אליהם ניתן להגיע מ־v (תקף גם למסלול מכוון).

<u>הוכחת טענה 11.8</u>: נניח שקיים מסלול בגרף $v \to v_1 \to v_2 \to \cdots \to v_f \to z$. אם `Explore` לא הגיע ל־$z$, אז קיים קדקוד v_j במסלול שלא המשיך ל־v_{j+1}, בסתירה לכך ש־`Explore` סוקר את כל שכני v_j. $\mathfrak{Q.E.D}$

נכליל את האלגוריתם `Explore` לאלגוריתם האב DFS:

```
def DFS(G):
    for all v∈V:
            visited[v] = 0
    for all v∈V:
```

```
if visited[v] == 0:
    explore(v)
```

__הרצאה 11 הגיעה לסיומה!__

הרצאה 12: שימושים לאלגוריתם ה־DFS: מציאת רכיבי קשירות

2/6/2019

א. מציאת רכיבי קשירות בגרף לא מכוון (CC – Connected Components)

הגדרה 12.1: נגדיר **רכיב קשירות** בגרף לא מכוון להיות קבוצת קדקודים מקסימלית[28] כך שקיים מסלול המחבר בין כל זוג קדקודים בה. במילים אחרות, רכיב קשירות בגרף מכוון $G = (V, E)$ הוא תת־גרף **קשיר מקסימלי**, כלומר $G_1 = (V_1, E_1)$ כך שלכל $v_1, v_2 \in V_1$ קיים מסלול $C: v_1 \to \cdots \to v_2$.

לשם מימוש אלגוריתם למציאת מספר רכיבי הקשירות בגרף, נשתמש - כפי שהכותרת מציעה - באלגוריתם ה־DFS, עם שינוי קל כך:

```
def Explore2(v):
    visited[v] = 1
    previsit(v)
    for all w so that (v, w) ∈ E:
        if visited[w] == 0:
            Explore2(w)
    postvisit(v)

def DFS2(G):
    for all v in V:
        visited[v] = 0
    for all v in V:
        if visited[v] == 0:
            CC_num += 1
            Explore2(v)

def previsit(v):
    CC[v] = CC_num

def postvisit(v):   # For now, it does nothing.
```

[28] כלומר, שלא ניתן להוסיף לקבוצה קדקוד נוסף מן הגרף כך שהיא תקיים את תכונת רכיב הקשירות.

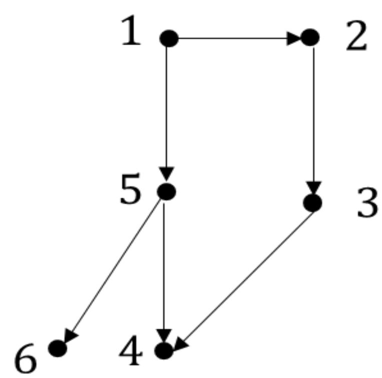
```
    return

def CC_main_algorithm(G):
    CC_num = 0
    DFS2(G)
```

מה הסיבוכיות? הוא עובר על כל הקדקודים ולכן היא לפחות $O(n)$, ועוברים על כל הצלעות לפחות פעם אחת (בגרף לא מכוון - בדיוק פעם אחת), אז הסיבוכיות היא $\Theta(n+m)$, כאשר $n=|V|$, ו־ $m=|E|$.

ב. <u>מציאת רכיבי קשירות בגרף מכוון (SCC – Strongly Connected Components)</u>

<u>הגדרה 12.2</u>: נאמר שקבוצת קדקודים בגרף מכוון היא **רכיב קשירות** אם זו קבוצה מקסימלית כך שבין כל שני קדקודים בה קיים מסלול.

זו למעשה אותה הגדרה בדיוק כמו קודם, אלא שהפעם המשמעות היא שנדרש שיהיה מסלול מ־v ל־u ולהיפך, מה שנובע מאליו בגרף לא מכוון. לדוגמה, בגרף שלהלן יש שישה רכיבי קשירות.



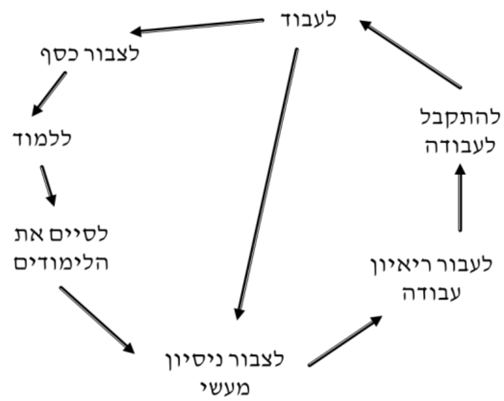
שרטוט 12.1: דוגמה לגרף מכוון עם שישה רכיבי קשירות: מכל קדקד בגרף יש מסלול דו־כיווני אך ורק לקדקוד בעצמו (מסלול באורך 0).

מיון טופולוגי

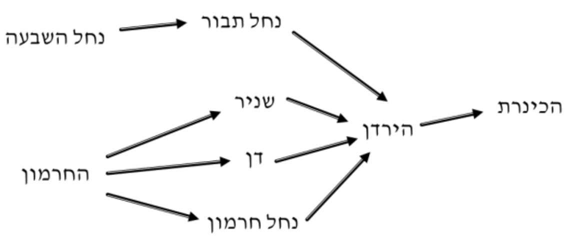
נניח שיש לנו רשימה של משימות לבצע, כך שיש בין חלקן תלות: נניח, עלינו לגשת לדואר ולקחת חבילה, אך לשם ביצוע משימה זו עלינו לחדש את הדרכון, אך לשם כך עלינו להצטלם, ולשם כך עלינו להסתפר וכן הלאה.[29] נרצה לתת הגדרה עבור סידור שיאפשר לנו לבצע את המשימה.

<u>הגדרה 12.3</u>: מיון טופולוגי של גרף מכוון $G=(V,E)$ הוא סידור של הקדקודים כך שלכל צלע (u,v) מתקיים ש־u נמצא לפני v בסידור.

[29] ראו חבילה הגיעה.



שרטוט 12.2: גרף מכוון שלא ניתן להגדיר עליו מיון טופולוגי



שרטוט 12.3: גרף מכוון שניתן להגדיר עליו מיון טופולוגי.
<u>הערה</u>: "כל-הנחלים הולכים אל-הים, והים איננו מלא; אל-מקום, שהנחלים הולכים--שם הם שבים, ללכת."

לדוגמה, בגרף 12.3 נוכל להגדיר מיון טופולוגי כך:

[הכינרת → הירדן → נחל תבור → נחל החרמון → דן → שניר → נחל השבעה → החרמון]

אולם בגרף 12.2 לא ניתן להגדיר מיון טופולוגי, שכן צריך ניסיון מעשי כדי להתקבל לעבודה, אבל צריך לעבוד כדי לצבור ניסיון מעשי, ולכן תואר במדמ"ח אינו באמת שימושי.

נבצע שינוי באלגוריתם: נגדיר משתנה חדש, clock, שבאמצעותו אנו ממספרים את הכניסות והיציאות לכל קדקוד. אנו שומרים ומעדכנים את ערך המשתנה clock, כך שהוא ממלא מעין פונקציה הקוראת ל"שעה" הנוכחית, כלומר ממספרת את הדברים לפי סדר התרחשותם. הפונקציות המעודכנות, `previsit` ו־ `postvisit` נראות כך:

```python
def previsit(v):
    pre[v] = clock
    clock += 1


def postvisit(v):
    post[v] = clock
    clock += 1
```

נשים לב שזמן הריצה האסימפטוטי לא השתנה ועודנו $O(|V|+|E|)$.

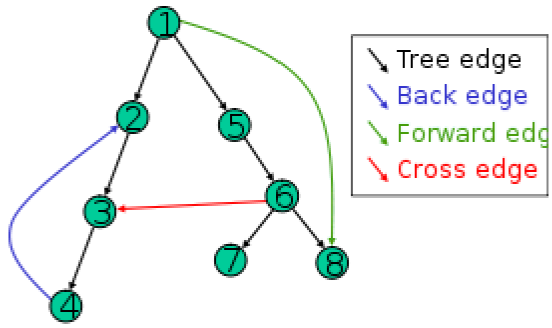
<u>טענה 12.4</u>: לגרף מכוון שיש בו מעגל, לא קיים מיון טופולוגי.

<u>הוכחת טענה 12.4</u>: נניח שקיים מעגל $v_0 \to \cdots \to v_k \to v_0$, ונניח בשלילה שקיים מיון טופולוגי. אז v_0 מופיע במיון הטופולוגי, ולפי הגדרה v_k מופיע אחרי v_0, אבל נקבל בסתירה ש־v_0 מופיע אחרי v_0.

Q.E.D

נזכיר את ההגדרות מהתרגול - בריצת ה־DFS יש ארבעה סוגים של צלעות:

1. צלע של העץ, השייכת ליער הנוצר לאחר ריצת ה־DFS.
2. צלע קדימה, שמצביעה לקדקוד שמופיע בעץ ה־DFS כצאצא, אך אינה שייכת ליער ה־DFS.
3. צלע אחורה, שסוגרת מעגל, כלומר מצביעה לאב קדמון בעץ ה־DFS.
4. צלע חוצה, שאינה אף אחת מהקודמות.



שרטוט 12.4: דוגמה לארבעת סוגי הצלעות בריצת DFS

<u>הגדרה 12.5</u>: כיור (בור) הוא קדקוד בגרף מכוון שאין ממנו צלעות יוצאות (כלומר, שנכנסות אליו צלעות בלבד). לדוגמה, הכינרת היא בור בגרף 12.3.

<u>הגדרה 12.6</u>: מקור הוא קדקוד בגרף מכוון אליו לא נכנסות צלעות (שיש לו צלעות יוצאות בלבד). לדוגמה, החרמון הוא מקור בגרף 12.3.

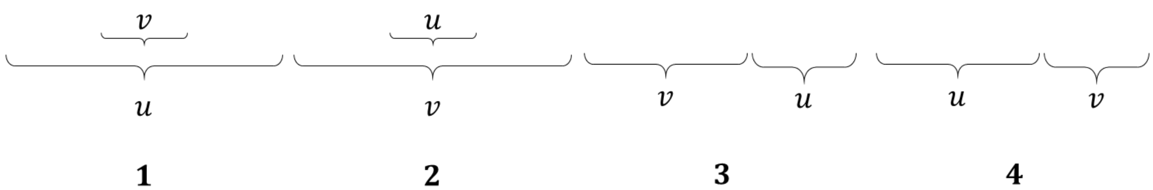
ייתכנו מספר כיורים ובורות בגרף.

<u>טענה 12.7</u>: בגרף מכוון יש מעגל **אם ורק אם** יש בו צלע אחורה.

<u>הוכחת טענה 12.7</u>: בכיוון הראשון, אם יש בגרף צלע אחורה (v,w), מתוך הגדרתה יש מסלול בעץ מ־w ל־v, כלומר היא סוגרת מעגל. בכיוון השני, אם בגרף המכוון יש מעגל $v_0 \to \cdots \to v_k \to v_0$, נניח ש־$v_j$ הקדקוד הראשון במעגל ש־DFS הגיע אליו. מכך ש־Explore מגיע לכל הקדקודים שיש אליהם מסלול מ־v_j, נסיק שכל הקדקודים $v_{j+1}, \dots, v_k, v_0, v_1, \dots$ צאצאים בעץ ה־DFS ולכן (v_{j-1}, v_j) צלע אחורה. $\mathfrak{Q.E.D}$

<u>טענה 12.8</u>: משפט הסוגריים: משפט זה דומה לטענת הטווח. בהינתן גרף $G=(V,E)$, אז לגבי כל שני קדקודים $u,v \in V$ מתקיים אחד מהארבעה:

1. $u.pre < v.pre < v.post < u.post$. במקרה זה $e=(v,u)$ היא **צלע עץ או צלע קדימה.**
2. $v.pre < u.pre < u.post < v.post$. במקרה זה $e=(v,u)$ היא **צלע אחורה.**
3. $v.pre < v.post < u.pre < u.post$. במקרה זה $e=(v,u)$ היא **צלע חוצה.**
4. $u.pre < u.post < v.pre < v.post$. במקרה זה $e=(v,u)$ היא צלע ?

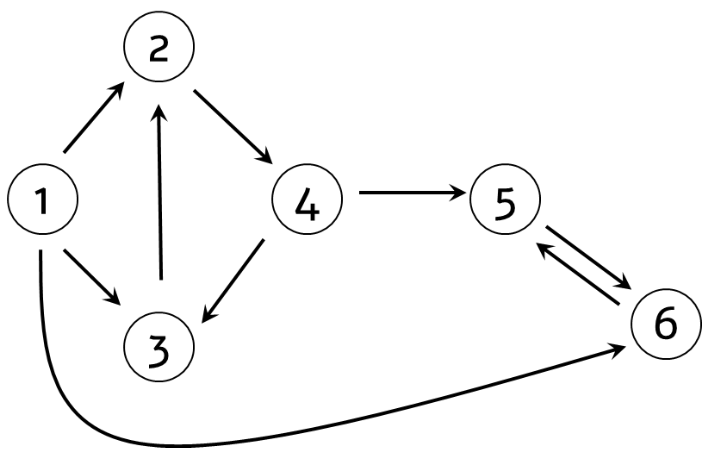


שרטוט 12.5: ארבעת המקרים של משפט הסוגריים, ממוספרים.

נבחין ש־(u,v) צלע אחורה כאשר היא קדקודיה מקיימים את מקרה 2 של משפט הסוגריים.

<u>הגדרה 12.9</u>: גרף רכיבי קשירות חזקה $\boldsymbol{G'=(V',E')}$ של גרף מכוון G הוא ייצוג גרפי של רכיבי הקשירות של G: לכל רכיב קשירות ב־G קיים קדקוד $v'\in V'$, וקיימת צלע $(v_1',v_2')\in E'$ בגרף רכיבי הקשירות אם ורק אם קיימת צלע המחברת בין רכיבי הקשירות המיוצגים על־פי v_1' ו־v_2' בהתאם (זאת אומרת, שקיים מסלול בין כל קדקוד ברכיב המתאים ל־v_1' לבין כל קדקוד ברכיב המתאים ל־v_2').

נבחין שמתוך הגדרת רכיבי הקשירות החזקה, גרף רכיבי קשירות חזקה הוא **תמיד DAG**.



שרטוט 12.6: משמאל: גרף כלשהו, ומימין: גרף רכיבי הקשירות שלו. נשים לב שהקדקוד 1 הוא מקור בגרף רכיבי הקשירות, והקדקוד (5,6) הוא בור.

נרצה למצוא בגרף מיון טופולוגי. לשם כך נוכיח טענה קריטית.

<u>טענה 12.10</u>: אם G הוא DAG, כלומר גרף חסר מעגלים, סידור הקדקודים בסדר יורד לפי ערכי ה־post שלהם מהווה מיון טופולוגי. נסו להריץ את אלגוריתם ה־DFS החדש על־מנת להבין את האינטואיציה למשפט זה.

<u>הוכחת טענה 12.10</u>: נניח שיש צלע (u,v). אם הגענו ל־u קודם, אז לפי מקרה 1 של משפט הסוגריים, $post[u]>post[v]$. אם הגענו ל־v קודם, אז יצאנו מ־v לפני שנכנסו ל־u (כי אין מעגלים ב־DAG) ולכן המקרה המתאים ממשפט הסוגריים הוא מקרה 3, וגם כאן $post[u]>post[v]$. Q.E.D

<u>טענה 12.11</u>: נניח שישנה צלע מ־C_1 ל־C_2, שני רכיבי קשירות חזקים. אזי ערך ה־post המקסימלי ב־C_1 גדול מערך ה־post המקסימלי ב־C_2.

<u>הוכחת טענה 12.11</u>: מדוע זה נכון? בדיוק כמו ההוכחה של (12.10).

ננסח כעת את האלגוריתם המתאים:

```
def SCC(G):
    DFS2(Gr)   # Gr is the graph of flipped edges. Run with clock!
    order them in descending order according to post values
    CC(G sorted by post)
```

כאשר DFS2 הוא האלגוריתם DFS הנעזר בחותמות הזמן.[30] נשים לב שאם A מטריצת השכנויות של הגרף $G=(V,E)$, אז הגרף G^r, הגרף שבו הופכים את כיווני הצלעות, הוא הגרף המתאים למטריצת השכנויות המשוחלפת A^t:

$$G^r=(V_r,E_r),\qquad V_r=V,\qquad E_r=E^t=\{\,(v_1,v_2)\mid (v_2,v_1)\in E\,\}$$

מדוע אנחנו עוברים על הגרף המשוחלף? כי נרצה להתחיל את ריצת ה־DFS ברכיב קשירות שמהווה **בור**. למה? נסביר עם בדיחה עצובה. היה פעם איש שנפל לבור, וזהו, הוא נשאר שם. מבחינה אינטואיטיבית, כשמתחילים מבור, אי אפשר לצאת ממנו - כך לא נעבור בטעות לרכיב קשירות אחר טרם סיימנו עם הנוכחי. אנו למעשה מריצים את ה־DFS במיון טופולוגי הפוך על גרף רכיבי הקשירות.

שרטוט 12.9: הזמן post

[חמש שריקות]
"מה זה?"
"השעון."
"השעה חמש?"
"זמן לישון. "
- פדריקו גארסיה לורקה,
מתוך "ככלות חמש שנים"
תקראו לורקה. הוא טוב.

שרטוט 12.8: הזמן pre

<u>הרצאה 12 הגיעה לסיומה!</u>

[30] "מהו הזמן? אם לא שואלים אותי אני יודע, אבל אם מבקשים אותי להסביר אינני יודע."
- אוגוסטינוס

הרצאה 13: אלגוריתם BFS, אלגוריתם Dijkstra ובעיית הסוכן הנוסע 16/6/2019

א. אלגוריתם Breadth-First Search – BFS

תזכורת מדיסקרטית: הגדרת המרחק בגרף מכוון

<u>הגדרה 13.1</u>: יהי $G = (V, E)$ גרף מכוון, ויהיו $v_1, v_2 \in V$. נגדיר את **המרחק** בין v_1 ל-v_2 להיות אורך המסלול המינימלי (מספר הצלעות בו) מ-v_1 ל-v_2. מסמנים ב-$d(v_1, v_2)$.

<u>הערות להגדרה 13.1</u>:

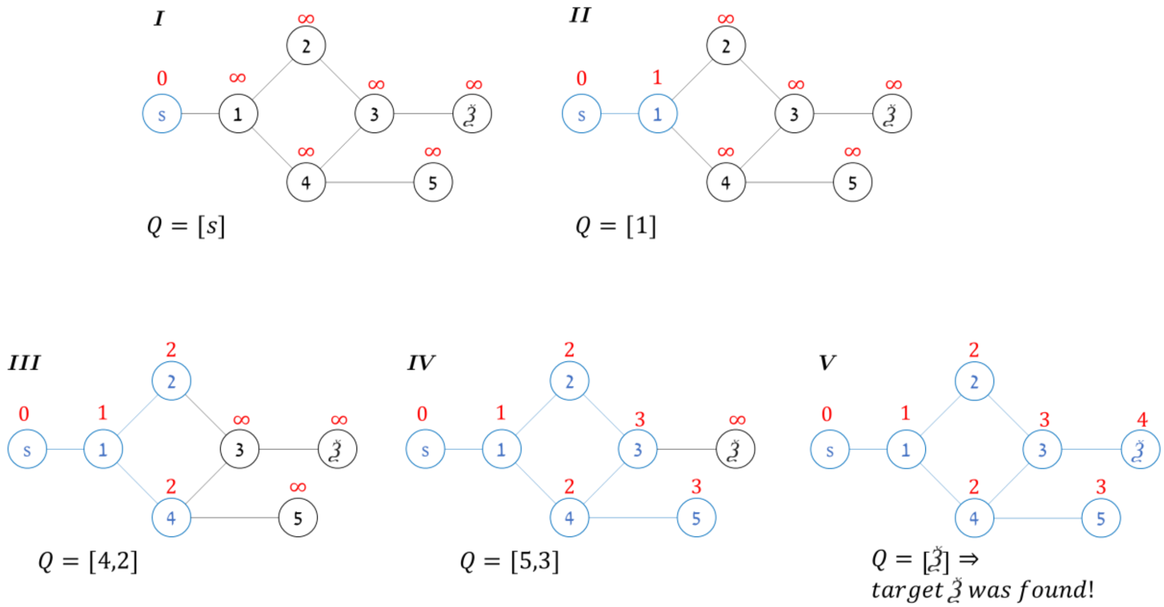
1. בגרף לא מכוון $d(u, v) = d(v, u)$.
2. אם לא קיים מסלול מ-u ל-v, מסמנים $d(u, v) = \infty$ (כמינימום של קבוצה ריקה).
3. $(v_1, v_2) = \min\{ d(v_1, v) \mid (v, v_2) \in E \} + 1$. נשתמש בתכונה זו עבור אלגוריתם חישוב המרחקים.

אלגוריתם חיפוש רוחבי – Breadth-first search (BFS)

נכתוב אלגוריתם חדש ואיטרטיבי שעובר על כל צלעות גרף. הוא אלגוריתם סורק שמחפש לרוחב הגרף (כלומר, סורק אותו לפי מרחק הולך וגדל מקדקוד הבסיס). בתרגיל ראיתם מימוש חלופי ל-DFS בלי רקורסיה, עם שימוש במחסנית, ונשאלתם מה היה קורה אילו היה משתמש בתור (Queue) ולא במחסנית. ובכן, האלגוריתם BFS, כפי שניתן היה לחשוד, משתמש בתור. להלן פסודו-קוד:

```
def BFS(G, start_vert, goal (optional)): # start_vert ∈ V
    Q = new queue
    for v in V:
        dist[v] = ∞ # mark as unvisited. No known path to it yet!
    dist[start_vert] = 0
    Q.enqueue(start_vert)
    while Q is not empty:
        u = Q.dequeue()
        if u is the goal: # the algorithm may search for some vertex.
            return u
        for every w so that (u, w) in E:
            if dist[w] == ∞: # if it is unvisited
```

```
Q.enqueue(w)
dist[w] = dist[u] + 1
```



שרטוט 13.1: דוגמה לריצת ה־BFS, משמאל למעלה לימין למטה. האלגוריתם מתחיל מהקדקוד s ומחפש אחר קדקוד המטרה ѯ, האות הקירילית העתיקה Ksi. באדום: המרחק לכל קדקוד בכל איטרציה. נשים לב שלתור מכניסים איברים משמאל ומוציאים מימין (וכן לפי סדר נומרי). קדקודים שבוקרו מסומנים בכחול.

נשים לב שהאלגוריתם BFS עובר בשלב הראשון על כל $v \in V$, ועל כל הצלעות בגרף פעמיים, ומבצע פעולות בסיבוכיות $O(1)$ עליהן, ומכאן שזמן הריצה שלו הוא $O(|V| + |E|)$.

הוכחת הנכונות של BFS

__**טענה 13.2:**__ האלגוריתם BFS מבקר בכל הצמתים ברכיב הקשירות של s (s מקצר `start_vert`).

__**הוכחת טענה 13.2:**__ יהא C מסלול מ־s ל־v: $C: s = v_0 \to v_1 \to \cdots \to v_k = v$. נראה באינדוקציה על i שהאלגוריתם מבקר ב־v_i.

__בסיס האינדוקציה__: עבור v_0 הטענה ברורה (נובע ישירות מהשורה `dist[s] = 0`).

__צעד האינדוקציה__: אילו ביקרנו ב־v_i, סיימנו. אחרת, v_i לא מבוקר, ומאחר שאנו מבקרים ב־v_{i-1} מהנחת האינדוקציה, נקבל שהאלגוריתם יבצע את השורה `dist[`v_i`] = dist[`v_{i-1}`] + 1`, כלומר יבקר ב־v_i. $\mathfrak{Q.E.D}$

נוכיח עתה את נכונות האלגוריתם.

__**טענה 13.3:**__ לכל $i \in \mathbb{N}$, האלגוריתם:

1. יכניס לתור את כל הקדקודים במרחק i ויכתוב $\text{dist}[v] = i$ עבורם.

2. כל הקדקודים במרחק i מ־s ייכנסו לתור **אחרי** כל הקדקודים שבמרחק $i-1$ **ולפני** כל הצמתים במרחק גדול מ־i.

<u>**הוכחת טענה 13.3:**</u> באינדוקציה על i כמובן.

<u>בסיס האינדוקציה</u>: עבור $i=0$ הטענה ברורה, יש רק קדקוד אחד שבמרחק 0 מ־s והוא s עצמו, והוא הקדקוד הראשון שנכנס.

<u>צעד האינדוקציה</u>: נניח שהטענה נכונה עבור $k \leq i-1$ ונוכיח עבור i. כל הקדקודים במרחק i ייכנסו לתור בשלב זה או אחר לפי (13.1), בהנחה שהגרף קשיר. אם v קדקוד במרחק i מ־s, יש לו שכן במרחק $i-1$, והוא יכניס אותו לתור. מהנחת האינדוקציה, שדה ה־dist של שכן זה מורה על $i-1$, ומכאן שהמרחק של הקדקוד יירשם כ־i.

מדוע אחרי כל הצמתים במרחק $i-1$? חלק זה נובע ישירות מהנחת האינדוקציה עבור $i-1$, מאחר שאין לו שכנים ממרחק של פחות מ־$i-1$.

מדוע לפני כל הצמתים במרחק גדול מ־i? כי צומת במרחק $i+1$ ייכנס על־ידי צומת במרחק i בלבד, שנמצא תמיד בתור אחרי כל הצמתים במרחק $i-1$. כלומר, מהנחת האינדוקציה, לא נטפל בקדקוד ממרחק i לפני כל הקדקודים ממרחק $i-1$, ואלה יכניסו לתור את כל הצמתים ממרחק i. $\mathfrak{Q.E.D}$

<u>ב. מציאת מסלול אידיאלי בגרף ממושקל</u>

בעיית הסוכן הנוסע Travelling Salesman Problem

<u>בעיית הסוכן הנוסע</u> (וכמוה גם <u>בעיית הדוור הסיני</u>) היא אחת מהבעיות המפורסמות בתורת הגרפים ובתורת הסיבוכיות, אחת מבעיות ה־NP-hardness, ובעיה מרכזית בתחום האופטימיזציה. תיאור הבעיה הוא כדלהלן: נניח שנכשלנו במבחן הסופי בדאסט ועל־כן החלטנו שאין לנו עתיד במקצוע, ולכן פרשנו מהתואר והתחלנו את דרכנו כסוכני מכירות העוברים מעיר לעיר. יש בידינו רשימה של ערים ושל המרחקים ביניהן, ואנו רוצים לעבור בכולן פעם אחת ולחזור לבסוף לעיר בה התחלנו. מהי הדרך הקצרה ביותר שבה ניתן לעשות כן?

גרפים ממושקלים

נעסוק בכמה בעיות שקשורות לבעיית הסוכן. נרצה קודם כל להגדיר מהו מרחק בין שתי ערים - שכן הגדרת המרחק בין שני קדקודים בגרף אינה מספקת עבור בעיה זו (ראו שרטוט 13.2 להסבר מדוע).

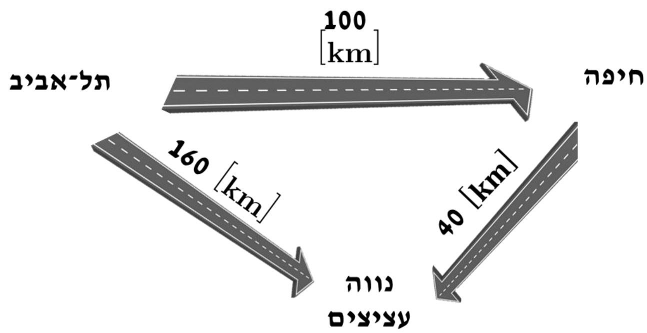
<u>**הגדרה 13.4:**</u> **גרף ממושקל** הוא גרף $G=(V,E)$ עליו מוגדרת פונקציית משקלים: $w: E \to \mathbb{R}^+$, כלומר פונקציה שנותנת לכל צלע את ה"משקל" (החיובי לבינתיים) שלה.

<u>הערה</u>: נגדיר מחדש את המושג **מרחק** בין u ל־v בגרף ממושקל כסכום כל המשקלים של הקשתות במסלול המינימלי בין u ו־v (מעתה כשנתייחס למושג "מסלול קצר ביותר" או "מינימלי", הכוונה למסלול שסכום משקלי קשתותיו מינימלי.)

<u>טענה 13.5</u>: בגרף ממושקל $G(V,E)$ עם פונקציית משקלים w, לכל $v_1, v_2 \in V$ מתקיים:

$$d(v_1, v_2) = \min\{\, d(v_1, v) + w(v, v_2) \mid (v, v_2) \in E \,\}$$

טענה זו נובעת מידית מההגדרה.



שרטוט 13.2: גרף ממושקל לדוגמה. נבחין שהמרחק מתל אביב לנווה עציצים אינו 160 מבחינתנו, אלא 140 - כי ניתן לקצר דרך חיפה. כמובן שבגרף שאינו ממושקל היינו מחשיבים את המרחק בין כל שני קדקודים כ-1 במקרה זה.

לא נפתור בדיוק את בעיית הסוכן הנוסע (על אף שהאלגוריתם שנציג עשוי לפתור ניסוחים מסוימים שלה), אלא נרצה למצוא את המסלול הקצר ביותר בגרף ממושקל. ניעזר בתור קדימויות: כלומר, בערימת מינימום.

אלגוריתם Dijkstra

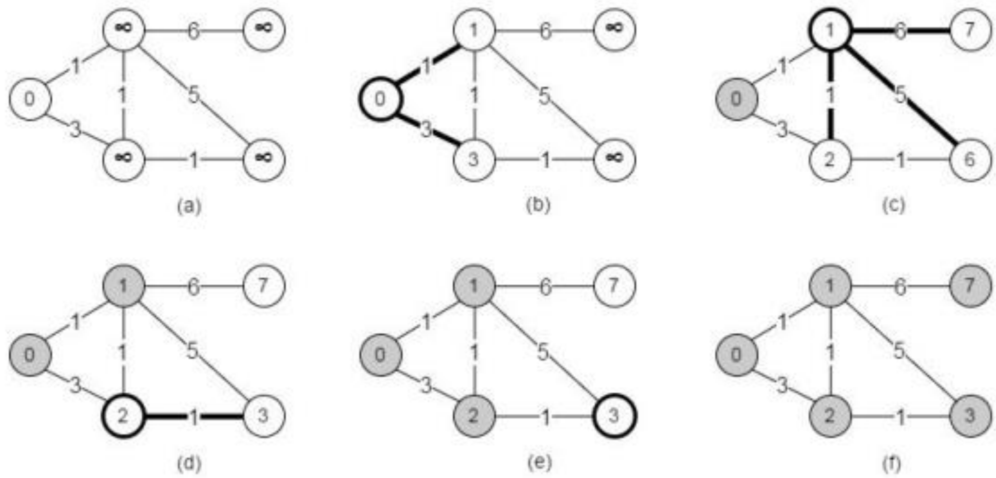
```
def Dijkstra(G, s, w):
    for v in V: # O(|V|)
        dist[v] = ∞
    dist[s] = 0
    Q = build_min_heap(V)  # O(|V|). We use the distances as key values.
    while Q is not empty:   # enters the loop |V| times
        v = Q.extract_min()   # O(log(|V|))
        for every u so that (v, u) in E: # overall: |E|
            if dist[u] > dist[v] + w(v, u):
                dist[u] = dist[v] + w(v, u)
                Q.decrease_key(u)   # O(log(|V|))
```

<u>טענה 13.6</u>: האלגוריתם של Dijkstra רץ בסיבוכיות זמן $O\big((|V|+|E|)\log|V|\big)$.

<u>הערה</u>: ניתן גם להוריד את זמן הריצה באמצעות שימוש [בערימת פיבונאצ'י](#) (לא צריך לדעת).

<u>הוכחת טענה 13.6</u>: באופן לא פורמלי מדי, סכימת כל הפעולות תיתן סיבוכיות זמן ריצה של $O(|V| + |V|\log|V| + |E|\log|V|)$. $\mathfrak{Q.E.D}$

נוכיח את נכונות האלגוריתם.



שרטוט 13.3: הרצת אלגוריתם דייקסטרה. בדוגמה: s הקדקוד השמאלי. במודגש: הצלעות בהן מתבוננים באיטרציה הנוכחית, באפור: הקדקודים שסיימנו לטפל בהם.
מומלץ לצפות באנימציה של האלגוריתם, [כאן](#).

<u>טענה 13.7</u>: נסמן ב־R את קבוצת הקדקודים שהוצאנו מ־Q. אזי לאחר הוצאת k קדקודים מהערימה, כשהאחרון שהוצאנו הוא v:

1. לכל קדקוד $w \in R$, $\text{dist}[w]$ מכיל את המרחק המתאים. (מרחק זה נשאר קבוע מהרגע ש־w יוצא מהערימה.)
2. טיפלנו בכל הקדקודים u שמרחקם מ־s קטן מהמרחק של v מ־s, $d(s,v)$.

<u>הערה</u>: טענה (13.7.1) מספיקה להוכחת נכונות האלגוריתם.

<u>הוכחת טענה 13.7</u>: באינדוקציה על k כמובן.

<u>בסיס האינדוקציה</u>: עבור $k = 0$, הקבוצה ריקה, הטענה נכונה באופן ריק.

<u>צעד האינדוקציה</u>: אם קדקוד v בראש התור, מהנחת האינדוקציה טיפלנו בכל הקדקודים v_i עבורם $\text{dist}[v_i] < \text{dist}[v]$, כלומר הם כולם יצאו מהתור. בפרט ל־v יש שכן u כך שמרחקו מ־s מינימלי, והוא בהכרח המרחק הנכון מהנחת האינדוקציה, ומכאן (13.4) מוכיח עבור v. $\mathfrak{Q.E.D}$

<u>הוכחה נוספת</u> לנכונות האלגוריתם ניתן למצוא [כאן](#).

<u>הרצאה 13 הגיעה לסיומה!</u>

הרצאה 14: עצים פורשים מינימליים, אלגוריתם Kruskal, תכונות חתך בגרף, קבוצה-זרה 23/6/2019

א. עצים פורשים מינימליים

<u>הגדרה 14.1</u>: עוד תזכורת מדיסקרטית: עץ הוא גרף קשיר וחסר מעגלים.

<u>טענה 14.2</u>: בעץ עם $n \geq 1$ קדקודים יש $n-1$ צלעות.

<u>הוכחת טענה 14.2</u>: באינדוקציה על מספר הקדקודים. עבור $n=1$ יש 0 צלעות. ניקח עץ $T=(V,E)$ עם $|V|=n$ קדקודים, ונרצה לחבר לו קדקוד נוסף v. הוספה של צלע (u,v) לכל $u \in V$ באופן ברור לא תגדיל את מספר רכיבי הקשירות (היה מסלול קודם לכן בין כל $u_1,u_2 \in V$, וכעת יש מסלול גם ל-v), ולא תוסיף מעגלים (שכן הוספנו צלע אחת בלבד (u,v) ואם נסירה נקבל שהיה מעגל בגרף המקורי). כמובן שהוספה של 0 צלעות לא אפשרית שכן הגרף יהיה לא קשיר, והוספה של שתי צלעות: (u_1,v) ו-(u_2,v) תוביל לקיום מעגל, שכן T קשיר ולכן קיים מסלול נוסף בין $u_1 \to \cdots \to u_2$ לפני ההוספה. מכאן שניתן להוסיף אך ורק צלע אחת בדיוק, וכך נקבל עץ עם n צלעות ו-$n+1$ קדקודים.
$\mathcal{Q.E.D}$

<u>טענה 14.3</u>: גרף קשיר עם n קדקודים ו-$n-1$ צלעות הוא עץ.

<u>הוכחת טענה 14.3</u>: נניח בשלילה שקיים T כזה שאינו עץ. הוא בפירוש קשיר, כלומר קיימים בו מעגלים. נוריד מספר מינימלי של צלעות עד אשר אין מעגלים (כאשר אנו שומרים על תכונת הקשירות לאורך הדרך - כל עוד יש מעגלים, נוכל להסיר צלע בלי לפגום בתכונה זו). כעת קיבלנו עץ עם פחות מ-$n-1$ צלעות, בסתירה ל-(14.1). $\mathcal{Q.E.D}$

הגדרת MST – עץ פורש מינימלי (Minimum Spanning Tree)

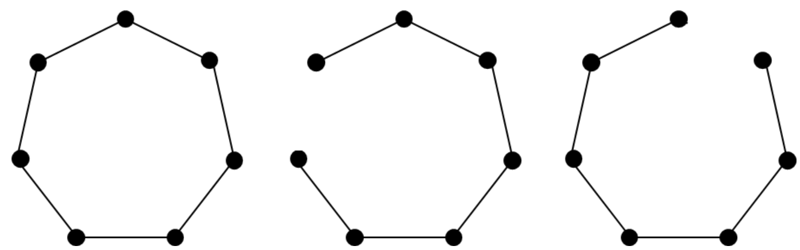
<u>הגדרה 14.4</u>: יהי $G=(V,E)$ גרף לא מכוון. נאמר ש-$T=(V',E')$ **תת-עץ פורש** של G אם T הוא עץ ו-$V=V'$ וגם $E' \subseteq E$.

<u>הגדרה 14.5</u>: יהיה $G=(V,E)$ גרף ממושקל ולא מכוון, עם פונקציית משקלים $w: E \to \mathbb{R}^+$. תת-עץ פורש $T=(V,E')$ של G ייקרא **עץ פורש מינימלי** אם לכל תת-עץ אחר T' מתקיים:

$$w(T) \leq w(T')$$

נזכיר, $w(T)=\sum_{e\in E} w(e)$.

<u>דוגמה</u>: בגרף **מעגלי** עם n קדקודים (ולכן גם n צלעות), יש n עצים פורשים (ראו שרטוט 14.1).



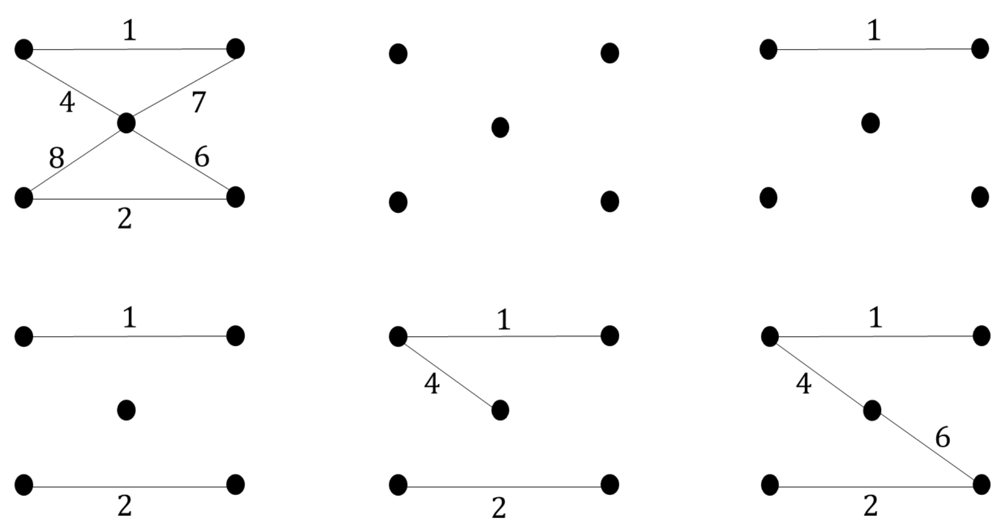
שרטוט 14.1: דוגמה לגרף מעגל עם $n = 7$ קדקודים (משמאל), ושני עצים פורשים שלו (מימין ובאמצע). הסרת כל אחת מהצלעות תיצור עץ פורש.

<u>טענה 14.6</u>: בגרף שבו כל צלעותיו בעלות משקל שונה, יש עץ פורש מינימלי יחיד.

<u>הוכחת טענה 14.6</u>: נניח שישנם T_1, T_2 עצים פורשים מינימליים שונים, כלומר קיימת צלע $e = (v, u)$ **<u>מינימלית</u>** כך ש- $e \in T_1$ אבל $e \notin T_2$. עם זאת ב-T_2 קיים מסלול $C = v \to v_1 \to \cdots \to v_k \to u$ (שאינו $v \to u$), כלומר הוספה של e ל-T_2 תיצור מעגל. באותו אופן הוספה של $(v, v_1), (v_1, v_2), \dots, (v_k, u)$ ל-T_1 תיצור מעגל ב-T_1. לכן בהכרח המסלול $C \subseteq T_2$ כולל איזושהי צלע $f \notin T_1$, (וכמו כן $f \in T_2$). מהגדרת $e \in T_1$, בהכרח מתקיים $w(f) \geq w(e)$ ומאחר שמשקלי הצלעות שונים נקבל $w(f) > w(e)$. למעשה, אם נחליף ב-T_2 את הצלע $f \in T_2$ בצלע $e \notin T_2$, נקבל עץ פורש בעלות נמוכה יותר, וזאת בסתירה להיותו של T_2 עץ פורש מינימלי. Q.E.D.

אלגוריתם Kruskal למציאת עץ פורש מינימלי בגרף ממושקל

נרצה אלגוריתם המוצא עץ מינימלי פורש בגרף ממושקל $G = (V, E)$. באופן לא פורמלי, נבנה גרף X נטול צלעות שמכיל את הקדקודים V של G. נסדר את צלעות G בסדר עולה לפי $w(v_i)$, ולפי סדר זה, לכל הצלעות $e \in E$: (א) אם e סוגרת מעגל, נדלג עליה. (ב) אם e איננה סוגרת מעגל, נוסיף אותה ל-X.



שרטוט 14.2: דוגמה להרצת אלגוריתם קרוסקל. משמאל לימין, מלמעלה למטה: הגרף המקורי, הגרף נטול הצלעות, והוספה של הצלעות מהקטנה לגדולה, כאשר שתי הצלעות האחרונות אינן מתווספות כי הן סוגרות מעגל.

זהו אלגוריתם חמדן, שבכל פעם מוסיף לגרף את הצלע שהמשקל שלה מינימלי. בשרטוט (14.2) מובאת דוגמה להרצה של האלגוריתם.

ב. תכונות החתך

על־מנת להוכיח את נכונות האלגוריתם של קרוסקל, ניעזר בכמה למות והגדרות.

<u>הגדרה 14.7</u>: יהי $G=(V,E)$ גרף לא מכוון.
חתך (V_1,V_2) של G הוא חלוקה של קדקודי V: שתי קבוצות הקדקודים $V_1,V_2\subseteq V$ הן זרות ואינן ריקות $V_1,V_2\neq\emptyset,\ V_1\cap V_2=\emptyset$.
צלע בחתך היא צלע $e\in E$ כך ש־$e=(v_1,v_2)$ עבור $v_1\in V_1$ ו־$v_2\in V_2$.

<u>טענה 14.8: למת / תכונת החתך</u>: בהינתן $G=(V,E)$ גרף לא מכוון עם n קדקודים ו־T עץ פורש מינימלי של G, ניקח תת־גרף $X=(V,E')$ של G, כך ש־$X\subseteq T$. נניח שקיים חתך $(V_1,V\backslash V_1)$ בגרף כך שאף צלע ב־X אינה צלע בחתך. אם e צלע **מינימלית בחתך** (מבחינת משקלה), אזי $X\cup\{e\}$ עדיין תת־קבוצה של MST כלשהו T'.

<u>הוכחת טענה 14.8</u>: אם $e\in T$ אז $X\cup\{e\}\subseteq T$ וסיימנו. נניח ש־$e\notin T$, אז $T\cup\{e\}$ גרף שיש בו מעגל. מכאן שב־T יש צלע נוספת בחתך, ונסמנה e'. נגדיר $T'=(T\cup\{e\})\backslash\{e'\}$, ומתקיים $X\cup\{e\}\subseteq T'$. נראה שהוא MST.
<u>עץ פורש</u>: יש בו n קדקודים ו־$n-1$ צלעות, והוא קשיר שכן T עץ והסרנו ממנו את המעגל היחיד שהוספת הצלע e יצרה. על־כן (14.3) יראה שהוא עץ.
<u>מינימלי</u>: מהנתון: $w(e)\leq w(e')$. נשלול את האופציה $w(e)<w(e')$, שכן אז T לא מינימלי, כלומר $w(e)=w(e')$, ומכאן $w(T)=w(T')$ ולכן T' מינימלי. $\mathcal{Q.E.D}$

<u>טענה 14.9: הוכחת נכונות עבור אלגוריתם קרוסקל</u>: בהינתן גרף $G=(V,E)$, k הצלעות הראשונות שאלגוריתם קרוסקל מוסיף מוכלות בעץ פורש מינימלי כלשהו של G.
<u>הוכחת טענה 14.9</u>: נסמן ב־X את אוסף הצלעות שנבחרו עד הצעד (האיטרציה) ה־$k-1$ באלגוריתם. נניח ש־e_k הצלע שהאלגוריתם מוסיף באיטרציה זו, האיטרציה ה־k. נרצה להראות ש־e_k מקיימת $X\cup\{e_k\}\subseteq MST$ ל־MST כלשהו. נשים לב ש־e_k מינימלית מבין הצלעות שאינן ב־X, ובהכרח אינה סוגרת מעגל. כלומר, קרוסקל מוסיף את הצלע e_k המינימלית שמחברת בין שני רכיבי קשירות T_1,T_2 (זרים) ב־X. נגדיר חתך כך: $(T_1,V\backslash T_1)$. e_k צלע בחתך, ומלמת החתך נובע ש־$X\cup\{e_k\}$. $\mathcal{Q.E.D}$
בפרט, אחרי האיטרציה ה־$|E|$ נקבל עץ פורש מינימלי ל־G.

ג. מימוש האלגוריתם של Kruskal ושל Prim

פורמלית יותר, נוכל לבצע זאת באמצעות מבנה הנתונים **קבוצה-זרה** שדיברנו עליו בתרגול.

תזכורת מתרגול 12: קבוצה זרה הוא מבנה נתונים המתחזק $S = \{S_1, \dots, S_n\}$ קבוצות דינאמיות, שבתחילת העבודה איתן הקבוצות זרות: $S_i \cap S_j = \emptyset$ לכל $i, j = 1, \dots, n$, $i \neq j$. כל קבוצה מיוצגת על־ידי אחד האיברים בה, נציג כלשהו, ומוגדרות על מבנה נתונים זה הפעולות:

1. MAKE-SET(v): יוצר קבוצה חדשה שמכילה את האיבר v.
2. FIND-SET(u): מחזיר את הנציג של הקבוצה שמכילה את האיבר u.
3. UNION(u, v) : מאחד את שתי הקבוצות שמכילות את האיברים v ,u. אם האיברים זהים, נעלה שגיאה.

להלן פסודו־קוד לקרוסקל (לקוח מוויקיפדיה):

```
def KRUSKAL(G, weight):
    A = Ø
    for every v ∈ V:
    MAKE-SET(v)
    foreach (u, v) ∈ E, ordered by weight(u, v), increasing:
    if FIND-SET(u) ≠ FIND-SET(v):
        A = A ∪ {(u, v)}
        UNION(u, v)
    return A
```

אלגוריתם Prim למציאת עץ פורש מינימלי בגרף ממושקל

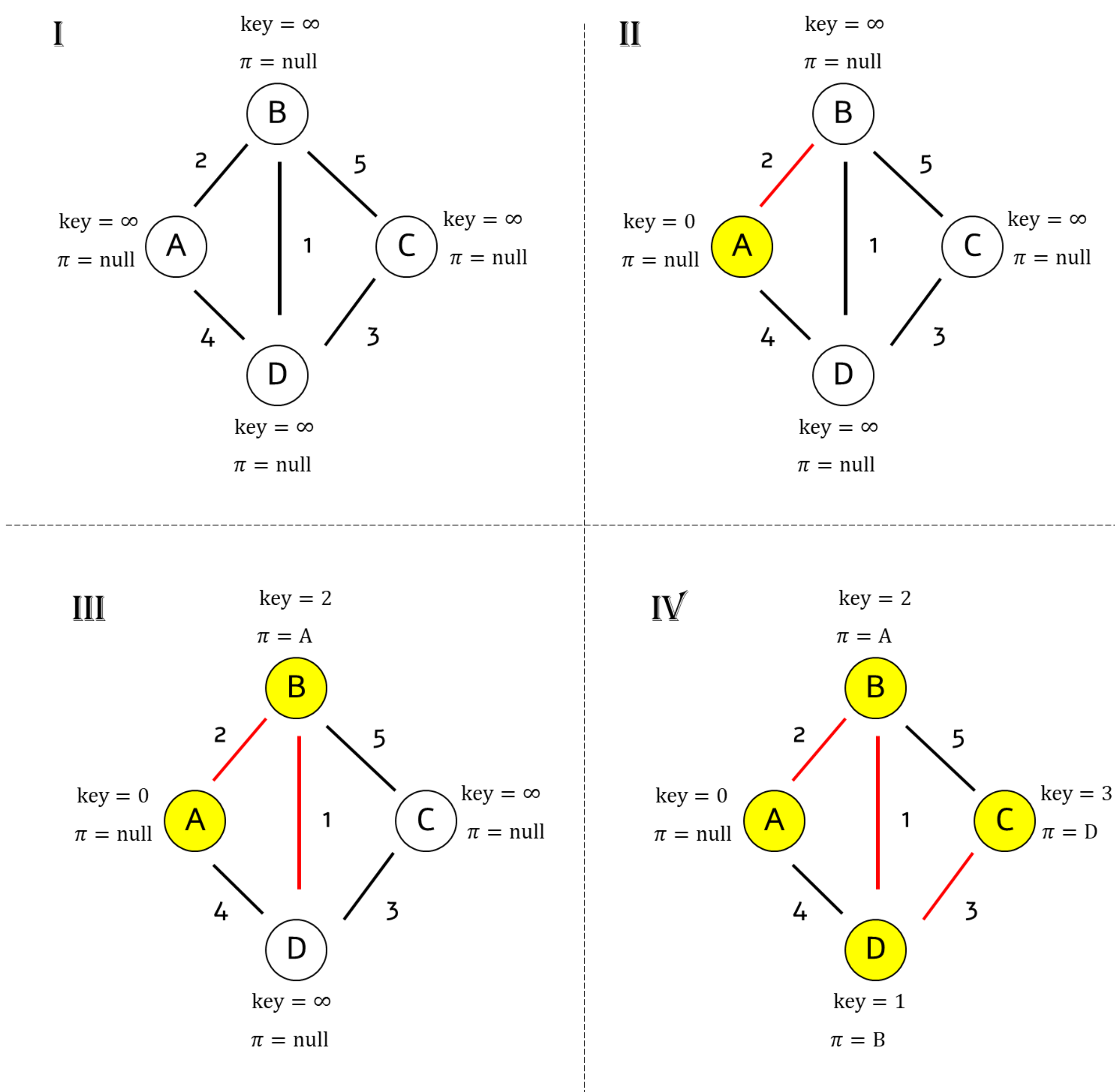
למרות שחלק זה לא היה בהרצאה אלא בתרגול 13 (ובתרגילים 13, 14), מאחר שהוא חשוב, אוסיף אותו לסיכום זה גם כן.

האלגוריתם של פרים (Prim) עובד באופן דומה מאוד לזה של קרוסקל, גם הוא אלגוריתם חמדן. ההבדל הוא שבמקום למצוא את הצלע המינימלית מתוך <u>כל</u> הצלעות שנותרו (שאינן סוגרות מעגל), האלגוריתם מחפש את הצלע המינימלית <u>מתוך אלו הכוללת את אחד הקדקודים שכבר נבחרו</u>. כלומר, הוא בונה עץ פורש מינימלי שנשאר קשיר לכל אורך הריצה.

הסבר אינטואיטיבי, קצת יותר מפורט מזה שמופיע בתרגול (ראו שרטוט 14.3):

1. מגדירים את הערכים $\text{key}(v)$ ו־$\pi(v)$ לכל הקדקודים $v \in V$. הערך $\pi(v)$ הוא מצביע לקדקוד שהתווסף לעץ הפורש **לפני הקדקוד v**, כלומר הקדקוד u שאת הצלע ממנו ל־v הוספנו ל־MST. $\text{key}(v)$ ייצג את משקל הצלע המחברת אותו לאותו הקדקוד.
2. מאתחלים את $\text{key}(v) = \infty$ ו־$\pi(v) = \text{null}$ לכל הקדקודים $v \in V$.
3. מתחילים מקדקוד אקראי s, ומסמנים $\text{key}(s) = 0$.
4. מבין הצלעות היוצאות מקבוצת הקדקודים שנבחרו עד כה, אל קדקודים שטרם נבחרו:
   א. בוחרים את המינימלית במשקלה.
   ב. מוסיפים הקדקוד לקבוצת הקדקודים.

ג. מוסיפים הצלע לעץ הפורש המינימלי.
ד. אם הושלם עץ, נחזיר אותו, אחרת נמשיך.



שרטוט 14.3: דוגמה פשוטה לריצת האלגוריתם של Prim. באדום נצבעות הצלעות שמוסיפים ל־MST, בצהוב הקדקודים, ולצד כל קדקוד מצוין הערכים π ו־key שלו בכל איטרציה.
כיצד נראית הערימה Q בכל איטרציה?

מעשית, נשתמש בערימת מינימום כדי למצוא את הצלע המינימלית מבין אלו שלא נבחרו. כאשר המשקלים בטווח ידוע נוכל להשתמש ברשימה מקושרת כפולה (Doubly-Linked List), כפי שמתואר בתרגיל 13, וכמו כן בערימת פיבונאצ'י, כפי שהזכרנו בהקשר של דייקסטרה.

להלן פסודו־קוד:

```
def Prim(G, s, w):
    for v ∈ V:    # O(|V|)
        key[v] = ∞
        π[v] = null

    key[s] = 0
    Q = Build_min_heap(V, key)   # O(|V|)
    while Q ≠ ∅:   # enters the loop |V| times
        v = Q.extract_min()   # O(log|V|)
        R = R ∪ {π(v), v}
        for every u so that (u, v) ∈ E:   # overall: 2|E| סך כל
            if u in Q and w(v, u) < key[u]:
                π[u] = v
                key[u] = w(v, u)   # O(log|V|)
    return R
```

__הוכחת נכונות:__ נובעת ישירות מלמת החתך.

__זמן הריצה:__ העלות היא $O\big((|V|+|E|)\log|V|\big)$: אתחול מערכים וערימה ב־$O(|V|)$, כניסה ללולאה העיקרית ערימה בדיוק $|V|$ פעמים ובכל פעם הוצאת מינימום ב־$O(\log|V|)$, ואז לולאה פנימית שבסך כל ריצותיה עוברת על כל הצלעות פעמיים (אחת בכל כיוון), ובכל צלע מבוצע עדכון מפתח בסיבוכיות של $O(\log|V|)$

__הרצאה 14 הגיעה לסיומה!__

סוף קורס מבני נתונים!

תודה לכם על כל התגובות האוהדות על הסיכום, ומקווה שנהניתם מהבדיחות (שלעיתים דרשו השקעה רבה יותר מפרקים אחדים) ומקישורי המוזיקה המוחבאים ברחבי הסיכום. תודה מיוחדת לכל מי ששלח לי סיכומי הרצאות.

מאחר שהרבה אנשים שאלו - השרטוטים נוצרו ב־PowerPoint והסיכום נכתב ב־Word בעזרת ה־Equation Editor.[31] באותו הנושא, אני ממליץ להשתמש בפונט הזה עבור עורך המשוואות - הוא הרבה יותר יפה מהפונט Cambria Math הדיפולטיבי שהשתמשתי בו בסיכום זה. הרחבה בנושא ניתן למצוא כאן וכאן. הפונט Latin Modern Math יוצר כמה בעיות במעבר ל־PDF, ולעומתו הפונט Asana Math אינו בעייתי, אך פחות יפה.

לקראת עונת המבחנים ובאופן כללי, אודה לכל מי שיסב את תשומת ליבי **לכל טעות קטנה באשר היא.** כל טעות ואי־דיוק בחומר חשובים לי. מעבר לשגיאות, יועילו הצעות לתיקון של עניינים כגון סימון לא קונסיסטנטי, פסקה שחסרה וכדאית בה דוגמה, הוכחה בלשון מסורבלת שצריכה שכתוב וכן טעויות בעיצוב ובעברית:[32] רווחים שהושמטו בשגגה או כאלו מיותרים, מספור לא עקבי של שרטוטים או משוואות, הפנייה למספרי עמודים שגויים, שגיאות כתיב, משפטים שניסוחם מטעה, שינוי פונט פתאומי, בדיחה לא מצחיקה (או הצעה לאחת מתאימה ומוצלחת) וכן הלאה. אל־נא תהססו לשלוח מסרון מהנשמע (כלומר, ווטסאפ, למספר 0542288485) או דואר אלקטרוני לכתובת roy.shtoyerman@mail.huji.ac.il. כל הערה תעזור לכולנו.

---

[31] זאת מאחר שהכתיבה ב־Lyx ו־Latex לוקחת לי זמן רב מדי, ואני מעדיף את הפתרון הפשוט והקצת יותר מכוער במקרה זה.

[32] כפי שבוודאי שמתם לב לאור הערות השוליים הרבות בנושאים אלו, כותב הסיכום אכן מעדיף לשון ובלשנות על־פני מבני נתונים.

רשימת אלגוריתמים

רשימת נושאים הכלולים בקורס וסיכום זה אינו מכסה

יש כמה נושאים שההרצאות אינן מכסות (או שאינן מכסות בשלמותם), והוצגו בתרגולים ובתרגילים בלבד. להלן רשימה <u>חלקית</u> של נושאים אלו (מוזמנים לשלוח לי נושאים שפספסתי):

1. הסתברות: הגדרות ודוגמות (קרב רב, הטלות מטבע וכו'...)
2. האלגוריתם Quick Select
3. מיונים (מובאים בתרגולים 4 ו־5) :
  א. מיון יציב: הגדרה וניתוח
  ב. מיון דליים (Bucket Sort)
  ג. מיון ספירה (Count Sort)
  ד. מיון בסיס (Radix Sort)
  ה. מיון ערימה (Heap Sort) (תרגול 7)
4. עצי אדום־שחור (תרגיל 6)
5. מחיקה בעץ AVL
6. גיבוב (תרגולים 8 ו־9):
  א. גיבוב לינארי וריבועי (Linear and Quadratic Probing)
  ב. גיבוב כפול (Double Hashing)
  ג. מסנן בלום (Bloom Filter)
  ד. הוכחת זמן בנייה של טבלת גיבוב מושלם (תרגול 10)
7. גרפים:
  א. יער ה־DFS וסוגי הצלעות בריצת ה־DFS (תרגול 10)
  ב. משפט הסוגריים (תרגול 11)
  ג. יצירת גרף רכיבי קשירות (תרגול 11)
  ד. אלגוריתם בלמן-פורד (Bellman-Ford) (תרגול 13)
  ה. ניתוח זמן ריצה של אלגוריתם קרוסקל (Kruskal) (תרגול 14)
8. היוריסטיקות של מבנה הנתונים קבוצה-זרה (תרגול 12)
9. ניתוח סיבוכיות לשיעורין (Amortized Complexity Analysis) (תרגול 12)

למה?