יסודות האלגוריתמים והסיבוכיות

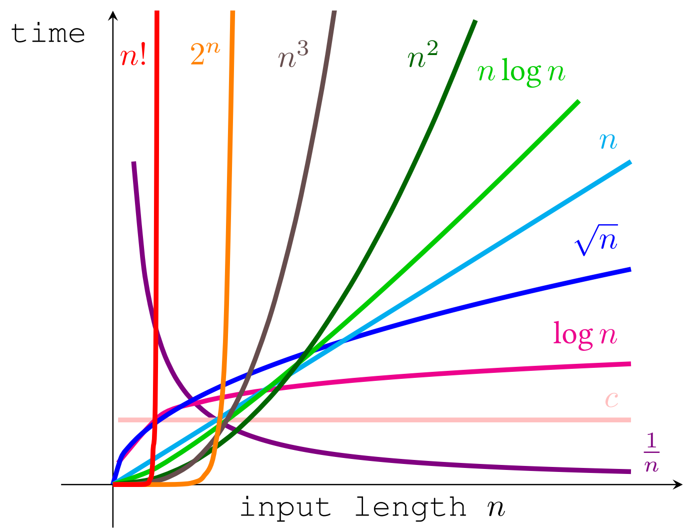
תרגולים

תוכן העניינים

1 ניתוח אסימפטוטי

אלגוריתם רצף צעדים המוגדרים היטב לפתרון של משימה או בעיה.[1]

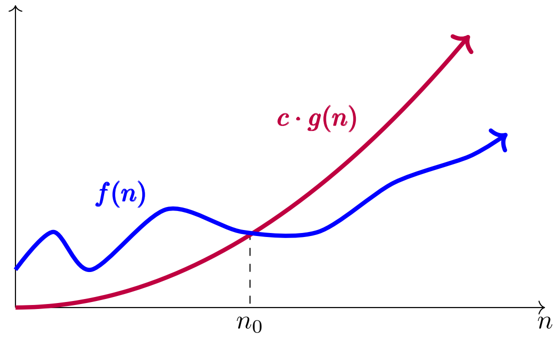
סיבוכיות זמן – מספר הפעולות *הבסיסיות* שהאלגוריתם מבצע כתלות בגודל הקלט, עד כדי קבוע שתלוי בחומרה ובתוכנה.



איור 1: פונקציות סיבוכיות נפוצות עבור קלט באורך n.

חסמים כשנתנח זמני ריצה נשתמש ב*חסמים*, פונקציות שמקבלות את אורך הקלט n ומחזירות את $f(n)$ שמתארת את זמן הריצה של האלגוריתם. נגיד ש $f(n)$ היא חסם עליון אם לא ייתכן שאלגוריתם שלנו ירוץ בזמן ארוך ממנה (Worst Case), כמו כן נגיד שהיא חסם תחתון אם לא ייתכן שזמן הריצה שלנו קצר ממנה.

[1] המונח אלגוריתם מבוסס על שמו של המתמטיקאי הפרסי אל-חוואריזמי שתיאר כבר במאה השמינית שיטה לפתירת משוואות ריבועיות – יותר *ממילניום* לפני המצאת המחשב. https://ioinformatics.org/journal/v11si_2017_71_74.pdf



איור 2: הפונקציה g היא חסם עליון של f כי היא גדולה ממנה (עד כדי קבוע c) לכל ערך של n הגדול מהקבוע n_0.

כדי להזניח קבועים תוך התחשבות בערכי קלט גדולים, נשתמש במתמטיקה אסימפטוטית. נבחן את הגבולות של הפונקציות כשהקלט שואף לאינסוף. נוכל להשתמש באחת משתי הגדרות שקולות.

סימון	**הגדרה 1**	**הגדרה 2**
$f \in \mathcal{O}(g)$	$\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} < \infty$	$f(n) \leq c \cdot g(n)$
$f \in o(g)$	$\lim\limits_{n\to\infty} \frac{f(n)}{g(n)} = 0$	$f(n) < c \cdot g(n)$
$f \in \Omega(g)$	$\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} < \infty$	$f(n) \geq c \cdot g(n)$
$f \in \omega(g)$	$\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = 0$	$f(n) > c \cdot g(n)$
$f \in \Theta(g)$	$f \in \mathcal{O}(g)$ וגם $g \in \mathcal{O}(f)$	$c \cdot g(n) \leq f(n) \leq c' \cdot g(n)$

ההגדרה השנייה מחייבת ש $c \in \mathbb{Q}^+, n_0 \in \mathbb{N}$ והמשוואה או אי-השוויון מתקיימים לכל $n > n_0$.

הסימון $\mathcal{O}$, "או גדול" (big O), יהיה הכלי העיקרי שלנו בניתוח זמני ריצה של אלגוריתמים. הוא והסימונים האחרים מבטאים יחסי גודל אסימפטוטיים (בדומה לסימונים $\leq$, $<$ ו $=$). כשאנחנו כותבים $f(n) \in \mathcal{O}(g(n))$ אנחנו מתכוונים ש g חוסם את f עד כדי קבוע כש n שואף לאינסוף.

בעזרת ההגדרות נוכל לראות ש$\mathcal{O}$ הוא **טרנזיטיבי**, כלומר אם $f \in \mathcal{O}(g)$ ו $g \in \mathcal{O}(h)$ אז $f \in \mathcal{O}(h)$ (אתם תוכיחו את זה במטלת הבית).

נפוץ מאד לכתוב $f = \mathcal{O}(g)$ במקום $f \in \mathcal{O}(g)$ אבל אל תטעו! בניגוד למשמעות הרגילה של סימון השוויון, $\mathcal{O}$ הוא **לא סימטרי**. כלומר, אם $f = \mathcal{O}(g)$ לא בהכרח מתקיים $g = \mathcal{O}(f)$. עוד מעט נוכיח את זה בעצמנו עבור $f(n) = n$ ו $g(n) = n^2$.

כמו כן, תוכיחו בתרגיל הבית ש $\mathcal{O}$ **אדיש לחיבור** כלומר אם $f \in \mathcal{O}(g)$ אז גם $f + g \in \mathcal{O}(g)$. תכונת האדישות לחיבור אינה מתקיימת עבור o ו ω, מצליחים לראות למה?

תרגיל 1

נתונה הפונקציה $f(n) = 10n^2 + 5n$. הראו כי $f \in \mathcal{O}(n^2)$ בעזרת קבועים c ו n_0 מתאימים.

פתרון:

ע"פ ההגדרה הראשונה:

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{10n^2+5n}{n^2} \\
&= \lim_{n\to\infty} \frac{n^2(10+\frac{5}{n})}{n^2} \\
&= \lim_{n\to\infty} 10 + \frac{5}{n} \\
&= 10 < \infty
\end{aligned}$$

ע"פ ההגדרה השנייה:

$$f(n) = 10n^2 + 5n \quad \leq \quad 10n^2 + 5n^2 \quad = \quad 15n^2 = 15 \cdot g(n)$$

ולכן אי השוויון מתקיים עבור כל n (ניקח למשל $n_0 = 1$) כש $c = 15$.

תרגיל 2

הוכיחו את הטענה $n^2 \notin \mathcal{O}(n)$.

פתרון:

ע"פ ההגדרה הראשונה:

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{n^2}{n} \\
&= \lim_{n\to\infty} n \\
&= \infty
\end{aligned}$$

ע"פ ההגדרה השנייה:

$$\begin{aligned}
\forall n > n_0 \quad : \quad f(n) &\leq c \cdot g(n) \\
n^2 &\leq c \cdot n \qquad | \text{ נחלק את שני הצדדים ב } n \\
n &\leq c
\end{aligned}$$

זוהי כמובן סתירה משום ש n אינו מוגבל ואילו c קבוע. גם אם אי השוויון מתקיים עבור n_0 כלשהו, הוא לא מתקיים עבור $n = n_0 + c$ לכן $n^2 \notin \mathcal{O}(n)$.

תרגיל 3

הראו כי $20\log(n) + 55\log_{20}(n) \in \Omega(\lg(n))$.

זכרו! $\log_a b = \frac{\log_c b}{\log_c a}$

פתרון:

נתחיל עם חישוב עזר קצר:

$$20\log(n)+55\log_{20}(n)=\frac{20\lg n}{\lg 10}+\frac{55\lg n}{\lg 20}=(\frac{20}{\lg 10}+\frac{55}{\lg 20})\lg n$$

נגדיר אם כך $c'=\frac{20}{\lg 10}+\frac{55}{\lg 20}$ (הביטוי קבוע ואינו תלוי ב n).

ע"פ ההגדרה הראשונה:

$$\lim_{n\to\infty}\frac{g(n)}{f(n)}=\lim_{n\to\infty}\frac{\lg(n)}{20\log(n)+55\log_{20}(n)}$$
$$=\lim_{n\to\infty}\frac{\lg(n)}{c'\cdot\lg(n)}$$
$$=\frac{1}{c'}<\infty$$

ע"פ ההגדרה השנייה:

$$\forall n>n_0\quad:\quad f(n)\ =c'\lg n=c'\cdot g(n)$$

אם ניקח את $c=c'$ אז $f(n)=c\cdot g(n)$ לכל n. בכך הוכחנו ש $f(n)\in\Omega(g(n))$ וגם ש $g(n)\in\Omega(f(n))$. למעשה הוכחנו משהו הרבה יותר חזק : $f(n)\in\Theta(g(n))$.

שימו לב שההוכחה תקפה כל עוד בסיס הלוגריתם קבוע ולכן $\log_a n\in\Theta(\log_b n)$ לכל a וb קבועים.

תרגיל 4

נתונה הפונקציה $f(n)=n\log(n^5)+6n$. הראו כי $f\in\mathcal{O}(n\log(n))$.

זכרו! $\log(n^c)=c\log(n)$.

פתרון:

ע"פ ההגדרה הראשונה:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{n\log(n^5)+6n}{n\log(n)}$$
$$=\lim_{n\to\infty}\frac{5n\log(n)}{n\log(n)}+\frac{6n}{n\log(n)}$$
$$=\lim_{n\to\infty}5+\frac{6}{\log(n)}$$
$$=5<\infty$$

ע"פ ההגדרה השנייה:

$$\begin{aligned} n\log(n^5)+6n &\le c\cdot n\log(n) \\ 5n\log(n)+6n-c\cdot n\log(n) &\le 0 \\ n(5\log(n)+6-c\cdot\log(n)) &\le 0 \qquad \mid \text{נניח ש } n_0>0 \\ 5\log(n)+6-c\cdot\log(n) &\le 0 \\ \log(n)(5-c)+6 &\le 0 \end{aligned}$$

כדי לפשט את אי השוויון נוכל לחפש c שיקיים אותו בהנחה ש $n_0=2$ (אנחנו יודעים שפונקציית הלוגריתם היא מונוטונית, כלומר לא יורדת).

$$\begin{aligned} \log(n)(5-c)+6 &\le 0 \quad \mid \log(2)\cong 0.301 \\ 24.93 &\le c \end{aligned}$$

לכן אי השוויון מתקיים עבור $c=25$ ו $n_0=2$.

שימו לב שהפעולות שלנו על אי השוויון יקיימו אותו לכל $n>n_0$. הנה דרך נוספת להוכיח את החסם.

$$\begin{aligned} n\log(n^5)+6n &= 5n\log(n)+6n\cdot 1 \\ &\le 5n\log(n)+6n\log(n) \quad \mid \text{אנו יודעים ש } \log n\ge 1 \text{ בעבור } n_0>10 \\ &= 11n\log(n) \end{aligned}$$

לכן נוכל לקחת כקבועים גם את $n_0=10$ ואת $c=11$.

תרגיל 5

נתונה הפונקציה הבאה:

$$f(n)=3n^2+5$$

הראו כי $f\in\Theta(n^2)$ מצאו n_0 ו c מתאימים.

פתרון:

ע"פ ההגדרה הראשונה:

$$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\lim_{n\to\infty}\frac{3n^2+5}{n^2}=\lim_{n\to\infty}\frac{3n^2}{n^2}+\frac{5}{n^2}=3<\infty$$

$$\lim_{n\to\infty}\frac{g(n)}{f(n)}=\lim_{n\to\infty}\frac{n^2}{3n^2+5}=\frac{1}{3}<\infty$$

ע"פ ההגדרה השנייה:

$$\begin{aligned} c_1\cdot g(n)\le\ & f(n) && \le c_2\cdot g(n) \\ c_1\cdot n^2\le\ & 3n^2+5 && \le c_2\cdot n^2 \\ 1\cdot n^2\le\ & 3n^2+5 && \le 8\cdot n^2 \end{aligned}$$

ניתן לראות כי התנאים מתקיימים לכל $n>n_0=1$ עבור הקבועים $c_1=1$ ו $c_2=8$.

תרגיל 6

הוכיחו באינדוקציה כי $\log(n) \in \mathcal{O}(n)$.

זכרו! $\log(xy) = \log(x) + \log(y)$.

פתרון:

נראה כי עבור $n = 1$ מתקיים $\log(n) \leq n$. (הנחת האינדוקציה):

$$\log(1) = 0 < 1$$

כעת נוכיח זאת עבור $n = n + 1$ (צעד האינדוקציה).

$$\begin{aligned} \log(n+1) &\leq \log(10n) \\ &= \log(10) + \log(n) \\ &= 1 + \log(n) \\ &\leq 1 + n && | \text{ נשתמש בהנחת האינדוקציה} \\ &\leq 2n \end{aligned}$$

ראינו שהנחת האינדוקציה מתקיימת עבור 1, ועל פי צעד האינדוקציה היא מתקיימת לכל n טבעי. אם כך $\log(n) \leq 2n$ עבור כל n טבעי ולכן $\log(n) \in \mathcal{O}(n)$ עבור $n_0 = c = 2$.

2 מיון מנייה

בעיית המיון בעיית המיון היא אחת הבעיות הבסיסיות ביותר בתחום האלגוריתמיקה. בבעיית המיון אנו מקבלים מערך A בו n איברים ואנו מתבקשים לבנות ממנו מערך ממויין.

האלגוריתם הנאיבי כשאנחנו אומרים "אלגוריתם נאיבי" אנחנו בדרך כלל מתכוונים לאלגוריתם "פשוט ולא מתוחכם". לדוגמא: אפשר למיין את מערך A על ידי חיפוש האיבר המינימלי במערך בסיבוכיות $\mathcal{O}(n)$, בכל פעם נוציא את האיבר המינמלי מ A ונוסיף אותו למערך הפלט שלנו. אחרי n חזרות על הפעולה הזאת נקבל מערך ממויין בסדר *לא יורד*[ב].

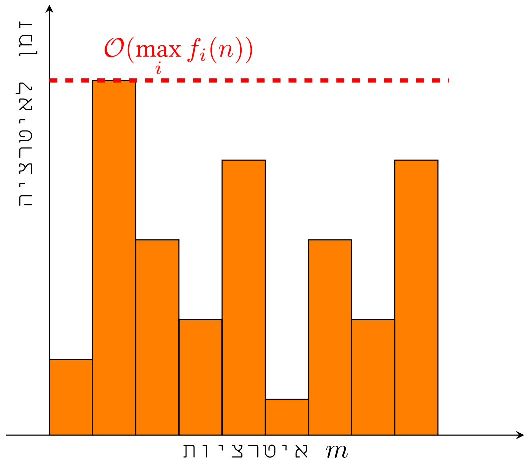
מאחר שחזרנו n פעמים על פעולה $\mathcal{O}(n)$ זמן, קיבלנו שזמן הריצה של המיון הנאיבי הוא $\mathcal{O}(n^2)$.

ניתוח גלובלי ולוקאלי לפעמים ניתן להשתמש בטכניקות ניתוח מעט יותר מתוחכמות כדי להראות שאלגוריתם מסויים רץ בזמן קצר יותר משטכניקות אחרות מראות. שימו לב! האלגוריתם הוא אותו אלגוריתם, רק שיטת הניתוח שונה. בתרגול הזה נלמד להבחין בין שתי טכניקות לניתוח זמני ריצה, ניתוח זמן לוקאלי וניתוח זמן גלובלי.

ב**ניתוח לוקאלי** נחפש חסם עליון על זמן הריצה של איטרציה אחת. אם זמן הריצה של האיטרציה הi הוא $\mathcal{O}(f_i(n))$ בבירור אין איטרציה ארוכה מ $\mathcal{O}(\max_i f_i(n))$. לאחר מכן נכפיל את מספר האיטרציות באותו חסם לאיטרציה בודדת. כלומר, אם היו לנו m איטרציות וכולן היו ארוכות כמו האיטרציה הארוכה ביותר זמן הריצה הכולל לא יחרוג מ $\mathcal{O}(m \cdot \max_i f_i(n))$.

ב**ניתוח גלובלי** נסתכל על כל האיטרציות כמכלול וננסה לחסום את זמן הריצה של *סכומן* באמצעות טיעון לוגי מחוכם. מיון מנייה הוא דוגמא מעולה לכך שבאמצעות מעט מחשבה נוכל להוכיח שזמני הריצה שלנו טובים יותר ממה שניתוח נאיבי יעלה.

[ב] כל איבר במערך קטן או שווה לאיבר הבא.



איור 3: בניתוח זמן לוקאלי נחסום את האיטרציה הארוכה ביותר ונניח שכולן ארוכות כמוה, בניתוח זמן גלובלי נחסום את סכום הזמנים של כל האיטרציות.

מערך מערך A הוא בעצם n איברים סדורים הנשמרים באופן *קבוע* בזכרון המחשב. מסיבה זו אי אפשר להגדיל מערך באופן יעיל, כדי להוסיף איברים למערך נאלץ להקצות זיכרון למערך חדש שיכיל בנוסף את כל האיברים החדשים, לכן הוספת איבר למערך בן n איברים תיקח $\mathcal{O}(n)$ זמן. לעומת זאת, ניתן לגשת לאיברים במערך בזמן קבוע $\mathcal{O}(1)$ בהינתן כתובת (אינדקס).

$$A = [a_1, \dots, a_n]$$

רשימה מקושרת בשונה ממערך, רשימה מקושרת L היא מבנה נתונים *דינמי*, אפשר להוסיף לה ולהסיר ממנה איברים מבלי לבנות את כולה מחדש. כשמאתחלים רשימה מקושרת בעצם מאתחלים מצביע p (POINTER), כשהרשימה ריקה המצביע אינו מצביע על דבר (למעשה הוא מצביע על $\varnothing$). כשנרצה להוסיף ערכים לראשית הרשימה המקושרת, נוסיף חוליה המכילה ערך a ומצביע p'. כעת נשנה את כתובת המצביע p כך שיצביע על החוליה (a, p') ואת הכתובת של המצביע p' נשנה כך שיצביע על מה שעד עתה הצביע p. קל לראות שהוספת חוליה לראשית הרשימה תתבצע בזמן קבוע $\mathcal{O}(1)$. כדי להוסיף חוליה לסוף הרשימה נאלץ להגיע עד המצביע האחרון ברשימה, זה שמצביע על $\varnothing$ ולכן זה יקח לנו $\mathcal{O}(n)$ זמן לרשימה בגודל n איברים. מאותה סיבה חיפוש ברשימה יקח $\mathcal{O}(n)$ זמן, נאלץ לעבור במקרה הגרוע ביותר על כל האיברים כדי לענות על השאלה "האם x ברשימה?".

$$L = p_0 \to (a_1, p_1) \to (a_2, p_2) \to \cdots \to (a_n, p_n) \to \varnothing$$

תרגיל 1

נתון המערך $[7, 3, 4, 1, 5, 8, 2, 7, 2]$ פרטו את שלבי מיון מניה.

```
CountSort(A[1,...,n], k)
Create Array C[1,..., k]
Create Array B[1,..., n]
i = 1
while i ≤ n do
    C[A[i]] = C[A[i]] + 1
    i = i + 1
i = 1 and j = 1
while j ≤ k do
    while C[j] > 0 do
        B[i] = j
        C[j] = C[j] − 1
        i = i + 1
    j = j + 1
return B[1,...,n]
```

פתרון:

1. בניית מערך מנייה C בגודל $k=8$ (טווח הערכים), כמו כן בניית מערך פלט B בגודל n.
2. נסרוק את מערך A ונגדיל ב1 את התא הj במערך המנייה C בכל פעם שנתקל בערך j במערך A.
3. נעבור על מערך המנייה מהקטן לגדול (בלי אובדן הכלליות[ג]) ונפלוט אותו למערך הפלט. עבור כל j מ $\{1,\dots,k\}$ נפלוט $C[j]$ פעמים את j למערך הפלט.

תרגיל 2

נתונה קבוצה של n מספרים שלמים בקטע $[1,k]$[ד].

(א) בצעו על הקבוצה פעולות בזמן $\mathcal{O}(n+k)$ כך שיהיה ניתן לענות בזמן $\mathcal{O}(1)$ על השאלה "כמה מספרים בקבוצה הנתונה הם בתחום $(a,b]$[ה]?"

(ב) כתבו מה החישוב שיתבצע במקרה בו a או b (או שניהם) אינם בתחום $[1,k]$.

פתרון:

(א) נשתמש באלגוריתם מיון מנייה עד שלב בניית מערך המנייה C. לאחר מכן נהפוך את C ל B "מערך צבירה" - מערך בו הערך בתא הi הוא מספר האיברים הקטנים

[ג] הכוונה היא שאנחנו יכולים לעבור גם מהגדול לקטן ע"פ צורך.
[ד] $x\in[a,b]\Rightarrow a\le x\le b$
[ה] $x\in(a,b]\Rightarrow a<x\le b$

מ i המופיעים בA. ניתן לעשות כך ע"י מעבר על C תוך סכימת האיברים שלו. עבור כל $i \in \{2,\ldots,k\}$ נעדכן $B[i] = B[i-1] + C[i]$.

כל עוד a ו b נמצאים בתחום $[1,k]$ נוכל להחזיר בזמן קבוע $B[b] - B[a]$.

(ב) כעת אנחנו מוכנים לשאילתות החריגות. נחלק אותן לארבעה מקרים:

(1) אם $a < 1$ וגם $b > k$ נחזיר $B[k]$ או n.
(2) אם $a < 1$ נחזיר $B[b]$.
(3) אם $b > k$ נחזיר $B[k] - B[a]$.
(4) אם $a, b > k$ או $a, b < 1$ נחזיר 0.

קל לראות שעבור כל זוג a ו b נענה על השאילתה בזמן קבוע. יש לנו $4 \in \mathcal{O}(1)$ מקרים ולכל מקרה מספר קבוע של פעולות.

תרגיל 3

נתונה קבוצה של n מספרים (שלמים, לא בהכרח שונים ולא בהכרח ממוינים) בטווח של בין 1 ל- 5000. כיצד ניתן לקבוע כמה פעמים נמצא ערך X כלשהו בקבוצה הנתונה? פרטו את שלבי האלגוריתם ונתחו את סיבוכיות זמן הריצה.

פתרון:

נכון, ניתן פשוט לעבור על המערך ולספור את מספר הפעמים שX מופיע, זה יקח לנו זמן לינארי בגודל המערך. אבל מה אם נשאל את אותה שאלה שוב ושוב עבור הרבה Xים שונים?

נשתמש בתכונות מיון מניה על מנת לפתור את הבעיה.

(1) נבנה מערך A בגודל n ונכניס אליו את ערכי n המספרים. $\mathcal{O}(n)$

(2) נבנה מערך עזר בגודל 5000. $\mathcal{O}(1)$

(3) עבור כל תא במערך A נגדיל את התא המתאים במערך B באופן הבא: $B[A[i]] = B[A[i]] + 1$. $\mathcal{O}(n)$

(4) נחזיר את $B[X]$. $\mathcal{O}(1)$

ניתוח זמן ריצה - סך הכול $\mathcal{O}(n) = \mathcal{O}(1) + \mathcal{O}(n) + \mathcal{O}(1) + \mathcal{O}(n)$.

תרגיל 4

נתון מערך של n מספרים שלמים בתחום 1 עד k. כל מספר עשוי להופיע מספר פעמים במערך. תארו אלגוריתם הממיין את המספרים במערך לפי מספר הופעותיהם, מספרים עם מספר הופעות זהה יופיעו במערך הפלט בסדר עולה ע"פ ערכם. כתבו את כל שלבי האלגוריתם ונתחו את זמן ריצת האלגוריתם.

פתרון:

כדי לבצע את המשימה נבנה מערך מנייה בזמן $\mathcal{O}(n+k)$, בנוסף נבנה מערך של רשימות מקושרות באורך k. כעת נעבור בסדר יורד על מערך המנייה C, עבור התא ה j של C נוסיף את האיבר j לראשית הרשימה ה $C[j]$ ב D ($D[C[j]]$.append(j)) - ניתן להוסיף איבר לראשית רשימה מקושרת ב $\mathcal{O}(1)$ זמן. עברנו בשלב זה על k תאים, בכל פעם הוספנו איבר אחד לראשית רשימה מקושרת, עשינו זאת ב $\mathcal{O}(k)$.

כעת מערך D מכיל את סוגי האיברים של A לפי מספר הופעותיהם. נותר לנו רק לפלוט אותם ממערך D, כך שהאיבר השמור בתא הj של D יפלט j פעמים למערך הפלט B. מאחר ואנחנו פולטים מתוך n תאים בסה"כ n איברים שלב הפליטה ייקח $\mathcal{O}(n)$ זמן.

```
OccurrenceSort(A[1,...,n], k)

Create Array C[1,..., k]
Create Array B[1,..., n]
Create List Array D[1,..., n]
i = 1
while i ≤ n do
    C[A[i]] = C[A[i]] + 1
    i = i + 1
i = 1
while i ≤ k do
    D[C[i]].append(i)
    i = i + 1
j = 1 and ℓ = 1
while j ≤ n do
    while D[j] is not empty do
        Let a be the next element of D[j]
        Remove a from D[j]
        i = 1
        while i ≤ j do
            B[ℓ] = a
            ℓ = ℓ + 1
            i = i + 1
    j = j + 1
return B[1,...,n]
```

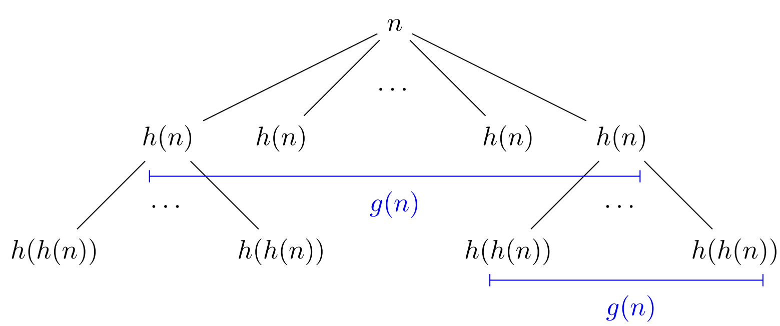
3 מיון מיזוג

אלגוריתם רקורסיבי הוא אלגוריתם הקורא לעצמו במהלך ריצתו. רקורסיה היא טכניקה יעילה מאד כאשר ניתן לחלק בעיה למספר תתי בעיות בעלות דפוס דומה לבעיה הגדולה. נחזור על החלוקה שוב ושוב עד שנגיע למקרה פשוט (כשנגיע *לתנאי העצירה*), אותו נפתור בקלות ואז נשתמש בפתרונות שמצאנו כדי להרכיב פתרון לבעייה הגדולה.

ננתח זמני ריצה של אלגוריתמים רקורסיביים באמצעות נוסחת נסיגה:

$$T(n) = g(n) \cdot T(h(n)) + f(n)$$

כש $T(n)$ הוא זמן הריצה הכולל של האלגוריתם על קלט בגודל n והוא תלוי בזמן הריצה של הקריאה הבאה ברקורסיה $T(h(n))$, במספר הקריאות שלה $g(n)$ ובזמן הריצה של הקריאה הנוכחית $f(n)$. כמובן ש $h(n)$ קטן מ n. קצב הנסיגה של n בין הקריאות יסייע לנו לנתח את זמן הריצה. **עץ**[1] **הרקורסיה** ממחיש את האופן בו מתנהג אלגוריתם רקורסיבי. נקרא לקריאה הראשונה *שורש העץ*, כל הקריאות שמבוצעות בשורש נקראות *הילדים* של השורש. שורש העץ מתאפיין בקלט האלגוריתם, כל צומת בעץ הרקורסיה מתאפיין בקלט של הקריאה שלו.



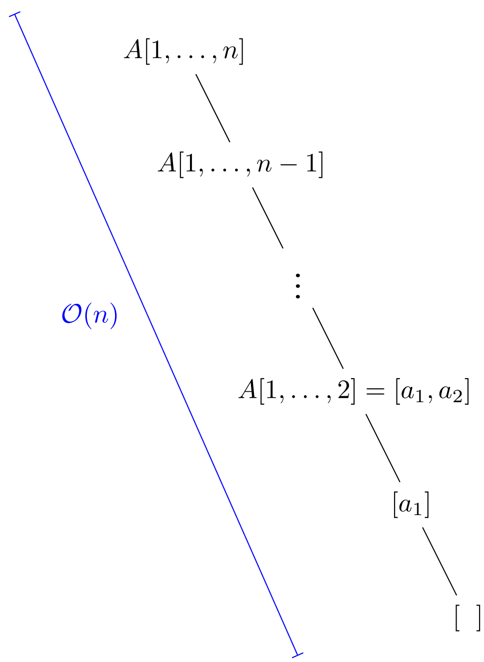
איור 4: המחשה של מהלך אלגוריתם רקורסיבי, כל קריאה עם קלט בגודל n מייצרת עד $\mathcal{O}(g(n))$ קריאות נוספות עם קלט בגודל $h(n)$. זמן הריצה של כל קריאה (ללא זמן הקריאות הרקורסיביות) הוא $f(n)$.

נדגים עם דוגמא פשוטה (כל כך פשוטה שזה כואב), נתון לנו מערך A ואנו מבקשים למצוא בו איבר x. נשתמש באלגוריתם הרקורסיבי הבא:

[1] עץ הוא מבנה נתונים המורכב מצמתים ובינהם קשתות (המבטאות יחס) כך שלכל צומת יש צומת אב אחד בלבד. מסיבה זו עץ הוא גרף ללא מעגלים – עוד על גרפים בהמשך!

```
Find(A[1,...,n], x)
if A is empty then
    return NO
else if x = A[n] then
    return YES
else
    return Find(A[1,...,n-1], x)
```

יפה, בואו ננתח את זמן הריצה שלו בצורה רקורסיבית. האלגוריתם מקבל מערך בגודל n, בודק אם המערך ריק או שהאיבר האחרון הוא האיבר הנדרש (תנאי העצירה), אם לא, הוא קורא לעצמו עם הקלט $A[1,\dots,n-1]$ שארית המערך.
קל לראות שקריאה אחת של $\mathtt{Find}(A[1,\dots,n],x)$ לוקחת זמן קבוע $f(n)\in\mathcal{O}(1)$, כמו כן עבור כל קריאה מתבצעת רק קריאה אחת נוספת במקרה הגרוע ביותר $g(n)=1\in\mathcal{O}(1)$. בין קריאה לקריאה אורך הקלט מצטמק באופן קבוע (כלומר $h(n)=n-\mathcal{O}(1)$) ותנאי העצירה מגיע כש $h(n)=0$, לכן לעץ הרקורסיה תהיינה $\mathcal{O}(n)$ רמות.



איור 5: עץ הרקורסיה של אלגוריתם $\mathtt{Find}(A[1,\dots,n],x)$ ניתן לראות שיש בו $\mathcal{O}(n)$ רמות.

כעת ניתן לנתח את זמן הריצה של $\mathtt{Find}(A[1,\dots,n],x)$:

$$
\begin{aligned}
T(n) &= \mathcal{O}(1) + T(n-1) && \mid\ T(x) = T(x-1) + \mathcal{O}(1) \text{ נשתמש בזהות} \\
&= \mathcal{O}(1) + \mathcal{O}(1) + T(n-2) \\
&= 2 \cdot \mathcal{O}(1) + T(n-2) \\
&= 3 \cdot \mathcal{O}(1) + T(n-3) \\
&\ \vdots \\
&= n \cdot \mathcal{O}(1) + T(0) && \mid\ T(0) = \mathcal{O}(1) \\
&= (n+1) \cdot \mathcal{O}(1) \\
&= \mathcal{O}(n)
\end{aligned}
$$

ניתחנו שזמן הריצה של אלגוריתם החיפוש שלנו הוא לינארי בגודל הקלט (הפתעה גדולה לא?), עכשיו בואו נשתמש בעקרונות שלמדנו כדי לנתח את זמן הריצה של אלגוריתם מתוחכם יותר - מיון מיזוג. בתרגול הזה נעבור על שלבי האלגוריתם, נוכיח את זמני הריצה שלהם ונלמד איך אפשר לבצע שינויים מקומיים באלגוריתם כדי לפתור בעיות שונות.

MergeSort($A[1,\dots,n]$)

```
if  n = 1  then
    return A
B1 = MergeSort(A[1,...,n/2]).
B2 = MergeSort(A[n/2+1,...,n]).
return Merge(B1, B2).
```

Merge($B_1[1,\dots,n]$,$B_2[1,\dots,n]$)

```
i = i1 = i2 = 1.
while i1 ≤ n/2  and  i2 ≤ n/2 do
    if  B1[i1] ≤ B2[i2]  then
        B[i] = B1[i1].
        i1 = i1 + 1.
    else
        B[i] = B2[i2].
        i2 = i2 + 1.
    i = i + 1.
if  i1 > n/2  then
    B[i,...,n] = B2[i2,...,n/2].
if  i2 > n/2  then
    B[i,...,n] = B1[i1,...,n/2].
return B
```

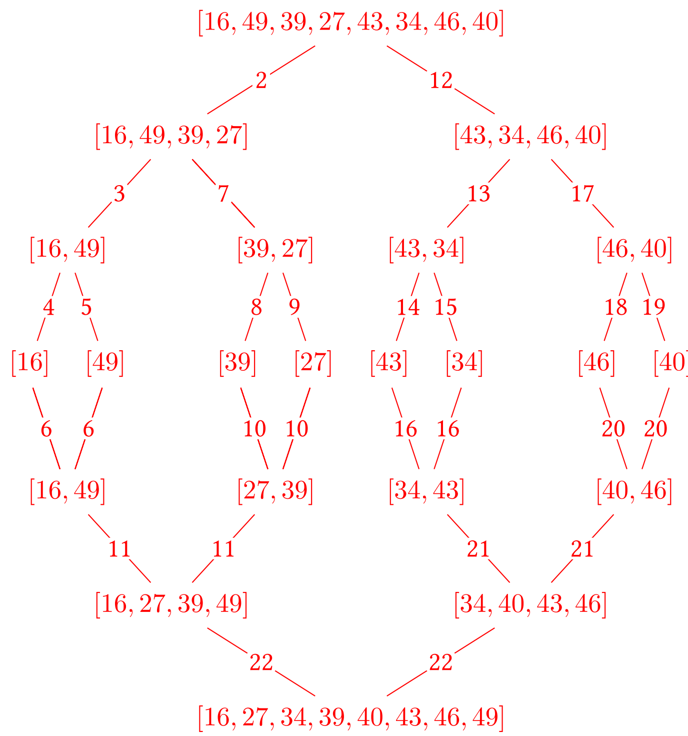
חובבי ריקודי עם טרנסילבניים-סקסונים לבטח יהנו מהסרטון בקישור המציג ההמחזה של אלגוריתם מיון מיזוג.[1]

תרגיל 1

נתון המערך $[16, 49, 39, 27, 43, 34, 46, 40]$
פרטו את שלבי מיון המיזוג.

[1] https://www.youtube.com/watch?v=XaqR3G_NVoo

פתרון:



איור 6: תרשים הזרימה של אלגוריתם מיון מיזוג. האלגוריתם מבקר בתתי המערכים ע"פ המספרים בקשתות, כשהמערך הראשון הוא מערך הקלט.

תרגיל 2

כעת נתמקד בשלב המיזוג, נשנה את אלגוריתם מיון מיזוג באופן הבא:
בכל שלב, במקום לחלק את המערך ל 2 תתי מערכים, נחלק את המערך ל k תתי מערכים בגודל זהה. כלומר כל תת מערך בגודל $\frac{n}{k}$ ($k > 1$ שלם). לאחר החלוקה נפעיל את האלגוריתם באופן רקורסיבי על כל אחד מתתי המערכים. אנו יודעים כי ניתן למיין בזמן קבוע מערך שגודלו חסום בקבוע. אם כך נותר לנו למזג את k תתי המערכים הממויינים.
תארו בקצרה איך למזג k תתי מערכים ממויינים לתוך מערך ממויין אחד. הניחו שנתונים k מערכים ממויינים, כל המערכים ביחד מכילים n איברים. מהו זמן הריצה של האלגוריתם? תנו תשובה כפונקציה התלויה ב n ו k.

פתרון:

נתונים לנו k מערכים ממויינים, כל אחד בגודל $\frac{n}{k}$ ועלינו לבנות מהם מערך ממויין אחד.

1. ניצור מערך עזר בגודל k.
2. ניצור k מצביעים, אחד עבור כל מערך. עבור כל מצביע i מ kn המצביעים, נכוון את i כך שיצביע על האיבר המינימלי במערך i (מאחר וכל אחד מהמערכים ממויינים, המצביע יצביע על האיבר הראשון).
3. נעתיק את k האיברים עליהם מצביעים המצבעים לתוך מערך העזר.
4. כל עוד קיים מצביע שנמצא בתוך אחד המערכים:
   (א) נמצא את האיבר המינימלי מתוך מערך העזר ונעתיק אותו למערך הפלט.
   (ב) נקדם את המצביע שהצביע עליו לאיבר הבא במערך.
   (ג) נעתיק את האיבר החדש במקום האיבר המינימלי.

שורות 1-3 מתבצעות ב $\mathcal{O}(k)$ זמן. שורה 4 תתבצע n פעמים, עד שעברנו על כל האיברים של כל המערכים. מציאת מינימום במערך בגודל k מתבצעת ב $\mathcal{O}(k)$ זמן. שאר הפעולות בלולאה מתבצעות בזמן קבוע. לכן סך הזמן של הלולאה בשורה 4 יהיה $\mathcal{O}(n \cdot k)$.

זמן הריצה הכולל של האלגוריתם יהיה איפה $\mathcal{O}(n \cdot k)$.

תרגיל 3

להלן אלגוריתם מיון מיזוג חדש. האלגוריתם מקבל מערך באורך n כך ש- n היא חזקה של 3, מפצל את המערך לשלשה מערכים באורך שווה, ממיין רקורסיבית את שלשת המערכים, ואז משתמש באלגוריתם של שאלה 2 כדי למזג את שלשת המערכים למערך ממוין אחד. מהו זמן הריצה של האלגוריתם הנ"ל? נמקו.

פתרון:

ראשית נבחן כמה רמות יש בעץ שבו מפצלים את המערך ל-3:
מכיוון שבכל פעם אנו מחלקים את המערך ב-3, בכל שלב יש $\frac{n}{3^k}$ איברים בכל מערך (כאשר k הוא מספר הרמות). התהליך מסתיים כשיש בכל מערך איבר אחד בלבד, לכן נדרוש: $\frac{n}{3^k} = 1$ כלומר $n = 3^k$. נוציא לוג משני הצדדים ונקבל שמספר הרמות הוא $k = \log_3(n)$.
לפי שאלה 2 מיזוג של k מערכים בגודל $\frac{n}{k}$ רק בזמן $\mathcal{O}(3n) = \mathcal{O}(n)$ עבור $k = 3$. מכאן שעלות סך פעולות המיזוג בכל רמה הוא $\mathcal{O}(n)$ (ניתוח גלובלי - בכל רמה i ממזגים k^i מערכים בגודל $\frac{n}{k^i}$). הראנו שישנן $\mathcal{O}(\log_3(n))$ רמות בעץ ולכן סה"כ סיבוכיות זמן ריצת האלגוריתם יהיה $\mathcal{O}(n \cdot \log_3(n))$.

תרגיל 4

נתונה קבוצה של n שמות משפחה. בנוסף, נתון אלגוריתם בשם LYXO היודע להשוות שתי מילים ולהחזיר ב- $\mathcal{O}(\lg(n))$ זמן, מי משני שמות המשפחה קטן לפי הסדר הלקסיקוגרפי. רשמו אלגוריתם המשתמש באלגוריתם הנ"ל על מנת למיין את כל המילים הנתונות בזמן $\mathcal{O}(n \lg^2(n))$.

פתרון:

שורה 3 של LYXOMerge מתבצעת $\mathcal{O}(n)$ פעמים בקריאה אחת.

LYXOMerge$(B_1[1,\ldots,n), B_2[1,\ldots,n])$

1: $i = i_1 = i_2 = 1$.
2: **while** $i_1 \le n/2$ and $i_2 \le n/2$ **do**
3:     **if** LYXO$(B_1[i_1], B_2[i_2]) = B_1[i_1]$ **then**   ▷ $\mathcal{O}(1)$ במקום $\mathcal{O}(\log n)$
4:         $B[i] = B_1[i_1]$.
5:         $i_1 = i_1 + 1$.
6:     **else**
7:         $B[i] = B_2[i_2]$.
8:         $i_2 = i_2 + 1$.
9:     $i = i + 1$.
10: **if** $i_1 > n/2$ **then**
11:     $B[i, \ldots, n] = B_2[i_2, \ldots, n/2]$.
12: **if** $i_2 > n/2$ **then**
13:     $B[i, \ldots, n] = B_1[i_1, \ldots, n/2]$.
    return B

בכל רמה הסיבוכיות של המיזוג היא $\mathcal{O}(n \cdot \lg(n))$ (השוואה של שתי מילים לוקחת $\lg n$ ובכל רמה זה קורה n פעמים). יש $\mathcal{O}(\lg n)$ רמות אז הסיבוכיות של כל האלגוריתם היא $\mathcal{O}(\frac{n}{2}\lg n) + \mathcal{O}(n\lg n)\cdot\mathcal{O}(\lg n) = \mathcal{O}(n\lg^2 n)$.

אפשר להסביר את אותו הדבר באמצעות נוסחת נסיגה:

$$T(n) = g(n) + T(h(n)) + f(n)$$

כמה קריאות רקורסיביות יש בקריאה אחת? 2 ($g(n) = 2$). בכל קריאה גודל הקלט מצטמצם בחצי ($h(n) = \frac{n}{2}$). זמן הריצה של אלגוריתם המיזוג הוא $\mathcal{O}(n\log n)$ עבור מיזוג של שני מערכים ממויינים בגודל $\mathcal{O}(n)$ ($f(n) = \mathcal{O}(n\log n)$). על כן ניתן לתאר את זמן הריצה של LYXOMerge ע"פ הנוסחא הבאה.

$$T(n) = 2\cdot T(\frac{n}{2}) + n\cdot\log n$$

הערך של $T(n)$ תלוי אם כך בערך של $T(\frac{n}{2})$, בואו נחשב מה הוא:

$$T(\frac{n}{2}) = 2\cdot T(\frac{n}{4}) + \frac{n}{2}\cdot\log\frac{n}{2}$$

אבל עכשיו הערך של $T(\frac{n}{2})$ תלוי בערך $T(\frac{n}{4})$. בואו נחשב אותו:

$$T(\frac{n}{4}) = 2\cdot T(\frac{n}{8}) + \frac{n}{4}\cdot\log\frac{n}{4}$$

אבל עכשיו הערך של $T(\frac{n}{4})$ תלוי בערך $T(\frac{n}{8})$. בואו נחשב אותו:

סתם זה לא ייגמר, אפשר להמשיך ככה עד אינסוף.

במקום זאת בואו ננסה להבין מה הדפוס. נציב את מה שקיבלנו בינתיים.

$$\begin{aligned} T(n) &= 2 \cdot T(\frac{n}{2}) + n \cdot \log n \\ &= 2 \cdot (2 \cdot T(\frac{n}{4}) + \frac{n}{2} \cdot \log \frac{n}{2}) + n \cdot \log n \\ &= 2 \cdot (2 \cdot (2 \cdot T(\frac{n}{8}) + \frac{n}{4} \cdot \log \frac{n}{4}) + \frac{n}{2} \cdot \log \frac{n}{2}) + n \cdot \log n \\ &= n \cdot \log n + 2 \cdot \frac{n}{2} \cdot \log \frac{n}{2} + 2 \cdot 2 \cdot \frac{n}{4} \cdot \log \frac{n}{4} + 2 \cdot 2 \cdot 2 \cdot T(\frac{n}{8}) \end{aligned}$$

כשפותחים את הסוגריים ומסדרים קצת את האיברים ניתן לראות שהנוסחא היא סכום של איברים מהצורה:

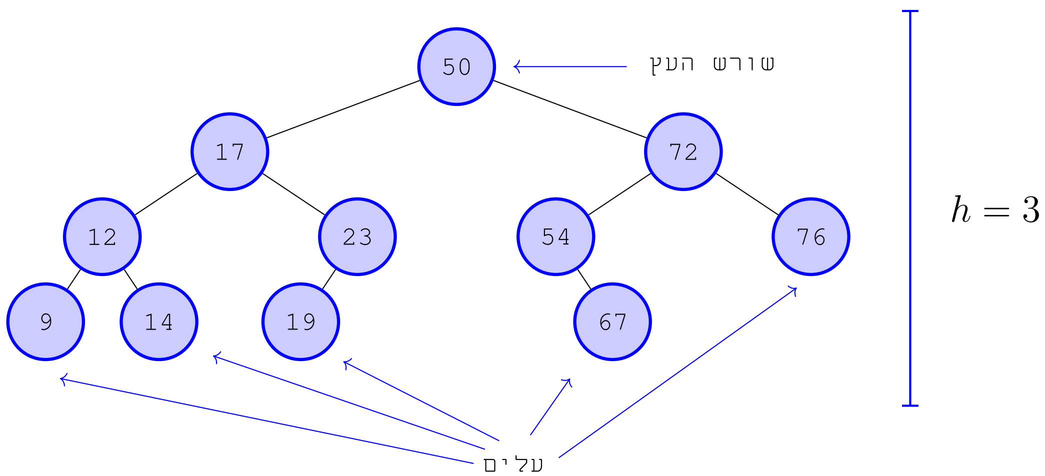
$$\begin{aligned} T(n) &= \sum_i 2^i \frac{n}{2^i} \log \frac{n}{2^i} \\ &= \sum_i n \log \frac{n}{2^i} \end{aligned}$$

כש i מתחיל ב0 ונגמר כשהרקורסיה נגמרת בעומק $\mathcal{O}(\log n)$.

$$\begin{aligned} T(n) &= \sum_i^{\log n} n \log \frac{n}{2^i} \\ &\leq n \sum_i^{\log n} \log n \\ &= n \log^2 n \end{aligned}$$

4 עץ חיפוש בינארי

בחלק זה של הקורס נתמקד במבני נתונים, מבנים שישמשו אותנו כדי לענות על שאילתות במהירות, שאילתות כמו "האם האיבר x קיים במבנה?", "מה האיבר המינימלי במבנה?" וכו'. הכרנו את המערך הממויין שבנייתו לוקחת $\mathcal{O}(n\log n)$ ומאפשר חיפוש בזמן לוגריתמי בגודל המערך. לעומת זאת פעולות של הכנסה והוצאת איברים יקרות יותר ולוקחות $\mathcal{O}(n)$ זמן, לכן מערך ממויין יהיה בחירה יקרה יותר כשנרצה לעדכן את מספר האיברים במבנה שלנו לעיתים תכופות. מנגד רשימה מקושרת מאפשר הוספה מהירה של איברים וחיפוש בזמן ארוך של $\mathcal{O}(n)$. עץ חיפוש בינארי מפשר בין הזמנים של פעולות אלה.



עץ בינארי הוא קצת כמו רשימה מקושרת, במובן שהאיברים בו מכילים מצביעים לאיברים אחרים, הוא גם קצת כמו מערך ממויין משום שהאיברים בו מסודרים ע"פ הערכים שלהם. עץ חיפוש בינארי הוא סוג של גרף[ה], לכן לאיברים בו נקרא גם *צמתים*, אם צומת מצביע על צומת אחר נגיד שיש בינהם *קשת*. כל צומת בגרף יחזיק עד שלושה מצביעים, אחד לצומת אב ושניים לילדים. לכל הצמתים למעט צומת אחד יהיה צומת אב, לצומת זה נקרא *שורש* העץ. לצמתים ללא ילדים נקרא *עלים*. נגיד שהעומק של צומת מסויים הוא מספר האבות הקדמונים שלו, כך שהשורש בעומק 0. נגדיר שגובה העץ הוא מספר האבות הקדמונים של הצומת העמוק ביותר. באופן שקול הגובה הוא מספר הקשתות במסלול הכי ארוך מהשורש לעלה.

עבור כל צומת v **בעץ חיפוש בינארי** מתקיים; המפתח של v קטן מהמפתחות של כל הצמתים בתת העץ הימני שלו וגדול מכל המפתחות בתת העץ השמאלי. בעצים בינארים מאוזנים הגובה (height) חסום ב$\mathcal{O}(\lg n)$, בשלב זה אתם כבר יכולים לראות למה זה נכון.

[ה]נכיר גרפים ביחידה השישית של הקורס.

	מערך ממויין	רשימה מקושרת	עץ חיפוש בינארי
בנייה	$\mathcal{O}(n \log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \times \text{height})$
הוספת איבר	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(\text{height})$
הוצאת איבר	$\mathcal{O}(n)$	$\mathcal{O}(1)$ בהינתן מצביע לאיבר $\mathcal{O}(n)$ אחרת	$\mathcal{O}(\text{height})$
חיפוש	$\mathcal{O}(\log n)$	$\mathcal{O}(n)$	$\mathcal{O}(\text{height})$

כל איבר בעץ מחזיק מצביעים לילדים שלו ומצביע לאב שלו, כך שניתן לנווט בעץ באמצעותם. ניתן למשל לסרוק את איברי עץ החיפוש ע"פ סדר בזמן לינארי, למדנו על ארבע סריקות:

1. In-Order:
תת עץ שמאל ⇐ שורש ⇐ תת עץ ימין

InOrder(x):
  if x.left $\neq \varnothing$ **then** InOrder(x.left)
  Print x.key
  if x.right $\neq \varnothing$ **then** InOrder(x.right)

2. Out-Order:
תת עץ ימין ⇐ שורש ⇐ תת עץ שמאל
(הפוך מ-In-Order)

OutOrder(x):
  if x.right $\neq \varnothing$ **then** OutOrder(x.right)
  Print x.key
  if x.left $\neq \varnothing$ **then** OutOrder(x.left)

3. Pre-Order
שורש ⇐ תת עץ שמאל ⇐ תת עץ ימין

PreOrder(x):
  Print x.key
  if x.left $\neq \varnothing$ **then** PreOrder(x.left)
  if x.right $\neq \varnothing$ **then** PreOrder(x.right)

Post-Order
תת עץ שמאל ⇐ תת עץ ימין ⇐ שורש

PostOrder(x):
  if x.left $\neq \varnothing$ **then** PostOrder(x.left)
  if x.right $\neq \varnothing$ **then** PostOrder(x.right)
  Print x.key

תרגיל 1

כתבו תוכנית המחזירה מפתח מינימאלי בעץ בינארי.

פתרון:

על מנת למצוא את המפתח המינימלי צריך למצוא את האיבר הכי שמאלי.

Min(x):
  while x.left $\neq \varnothing$ **do**
    $x = x$.left
  return x.key

אפשר לעשות זאת גם בצורה רקורסיבית עם:

MinRecursive(x):
  if x.left $\neq \varnothing$ **then**
    return x.key
  return MinRecursive(x.left)

זמן הריצה של שני האלגוריתמים חסום בגובה העץ $\mathcal{O}(\text{height})$.

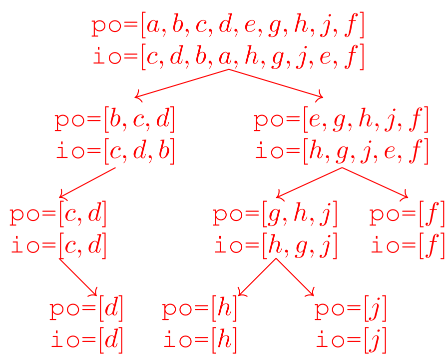
תרגיל 2

להלן שתי סריקות של עץ בינארי מסויים. שחזרו את העץ המקורי.
pre-order: $a, b, c, d, e, g, h, j, f$.
in-order: $c, d, b, a, h, g, j, e, f$.

רמז: העזרו ברקורסיה.

פתרון:

נתחיל במציאת השורש של העץ, אנחנו יודעים שהוא a כי הוא מופיע בראש סריקת pre-order שנתונה לנו. כעת נבחן את סריקת ה in-order, ע"פ התכונות של עץ בינארי אנחנו יודעים שכל מה שמופיע לפני a חייב להיות בתת העץ השמאלי של a כמו כן מה שמופיע מימינו חייב להיות בתת העץ הימני של a.



איור 7: עץ הרקורסיה של האלגוריתם, אם נחליף את הקלט באיבר הראשון של הpre-order נקבל את העץ המשוחזר.

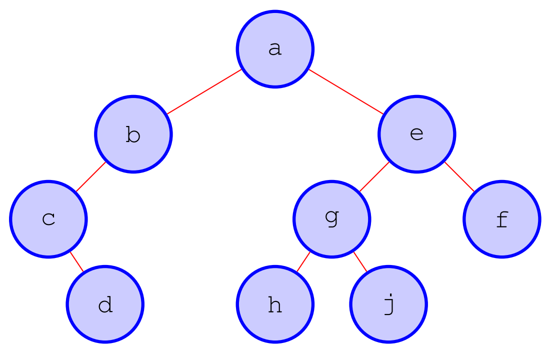
למעשה חילקנו את הבעייה שלנו לשתי בעיות קטנות, שחזור תת העץ הימני ושחזור תת העץ השמאלי. מה הוא הקלט של כל תת בעיה? אנחנו יודעים ש in-order של תת עץ שמאל היא c, d, b כי הם חייבים גם כאן להיות בסדר עולה. אם כך נוכל לראות שסריקת pre-order היא b, c, d (פשוט נוריד מהסריקה המקורית את כל מה שלא בתת העץ השמאלי). כפי שמצאנו את השורש a מייד ניתן לראות שהשורש של תת העץ השמאלי (או הבן השמאלי של a) הוא c. שוב ניתן לחלק את תת הבעיה לשני חלקים ולפתור אותם בצורה רקורסיבית עד ששחזרנו את כל העץ.

ReconstructTree(PO, IO):

Set root $PO[0]$ of T
Let i be the index of $PO[0]$ in IO
Set right sub-tree **ReconstructTree**$(PO[IO[i:]], IO[i:])$
Set left sub-tree **ReconstructTree**$(PO[IO[:i]], IO[:i])$
return T

איור 8: האלגוריתם לשחזור העץ, בהנתן מערך A נכתוב בקיצור $A[i:]$ לתת המערך שבא אחרי האיבר ה-i, $A[:i]$ נכתוב בקיצור לתת המערך שבא לפני. $A[B]$ נכתוב כקיצור למערך A אחרי שהורדנו ממנו את כל מה שלא במערך B.

כל השחזור ייקח $\mathcal{O}(n \times \text{height})$ זמן, ניתן לראות כי בכל קריאה האלגוריתם עובר על שתי הסריקות ומחלק אותן לשתיים $\mathcal{O}(n)$ עבור שתי סריקות בגודל $\mathcal{O}(n)$. בכל רמה סך אורכי הסריקות הוא לכל היותר $\mathcal{O}(n)$ מכאן שכל רמה מחושבת בזמן ליניארי. העץ המשוחזר:



תרגיל 3

הוכיחו / הפריכו את הטענה כי ניתן לשחזר עץ בהינתן סריקות pre-order ו post-order.

פתרון:

הטענה לא נכונה, נפריך באמצעות דוגמה.
בהנתן הסריקות pre-order = a,b ו post-order = b,a
לא ניתן לדעת אם b הוא בן ימני או בן שמאלי של a. לפיכך קיימים שני עצים שונים שיתאימו לאותן סריקות ולא ניתן להבחין בינהם בעזרתן.

תרגיל 4

נתון עץ בינארי אשר בכל צומת שלו מתועד גם מפתח האב (p). כתבו תוכנית המקבלת עץ וצומת m המחזירה את הצומת העוקב (הצומת בעל המפתח הקטן ביותר הגדול מהמפתח

של m). מה הסיבוכיות של התוכנית?

פתרון:

נחלק לשני מקרים:

1. אם יש לצומת בן ימני, הצומת העוקב יהיה הקטן ביותר בתת העץ הימני. בתרגיל הראשון כתבנו תוכנית למציאת האיבר המינימלי בעץ. $\mathcal{O}(\text{height})$

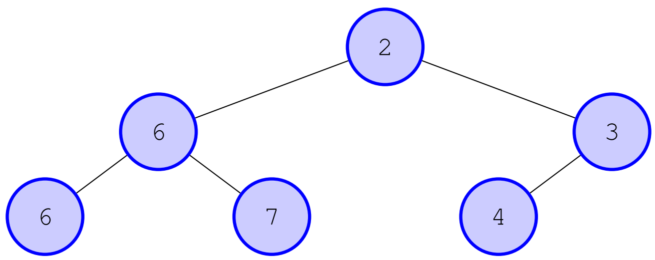
2. אם אין לצומת בן ימני אז הצומת העוקב יהיה האב הקדמון הצעיר ביותר של m שהבן השמאלי שלו הוא גם אב קדמון של m. נעלה אם כן במעלה העץ עד שהתנאי יתקיים. אם הגענו לשורש אך התנאי אינו מתקיים לבטח התחלנו באיבר המקסימלי ולכן אין לא עוקב, נחזיר $\varnothing$. $\mathcal{O}(\text{height})$

```
Successor(m):
  if x.right ≠ ∅ then                    ▷ אם יש לו בן ימני
      return Min(x.right)
  else                                   ▷ אם אין לו בן ימני
      y = m.p
      while y ≠ ∅ and m = y.right do
          m = y                          ▷ נתקדם במעלה העץ
          y = m.p
      if m = y.left then                 ▷ אם הגענו לשורש משמאל
          return y.key
      else                               ▷ אם הגענו לשורש מימין
          return ∅                       ▷ אין עוקב
```

5 ערימה בינארית

בתרגול הזה נתמקד בערימות בינאריות, מבנה נתונים יעיל ליישום של תורי קדימויות. ערימה בינארית היא עץ (קבוצת צמתים המחוברים בקשתות ללא מעגלים) המקיים שתי תכונות נוספות:

1. כל הרמות בערימה מלאות למעט האחרונה.
2. הבנים של כל צומת גדולים ממנו או שווים לו.



איור 9: דוגמה לערימה בינארית.

היתרון הגדול בערימה בינארית הוא שניתן לבנות אותה בזמן לינארי $\mathcal{O}(n)$ בעזרת האלגוריתם של Floyd, המשתמש בפעולת פעפוע Heapify שנלמדה בהרצאה.

Floyd($A[1,\ldots,n]$)

for $i \in \{n,\ldots,1\}$ **do**
 HeapifyDown(i)

פעולת הפעפוע קורית בזמן ביחס ישיר לגובה הערימה (בתרגול הזה נוכיח שהוא $\mathcal{O}(\lg n)$), כשמספר האיטרציות הוא כמספר האיברים בערימה הוא n ע"פ ניתוח לוקאלי זמן הריצה של הבנייה חסום ב $\mathcal{O}(n \lg n)$, אך באמצעות ניתוח גלובלי ניתן למצוא חסם יותר הדוק. נשים לב שהאלגוריתם של פלויד מפעפע מטה את כל איברי הערימה החל מהרמות התחתונות, בהן הפעפוע מטה קורה בזמן קבוע. לצומת בגובה i ייקח i זמן לפעפע מטה. לכן סך הפעולות של אלגוריתם חסומות ע"י $\mathcal{O}(\sum_{i\in\{1,\ldots,h\}} i \cdot \frac{n}{2^i})$. נראה שמספר הפעולות הינו טור מתכנס.

$$\begin{aligned}
\sum_{i\in\{1,\dots,h\}} i\cdot\frac{n}{2^i} &= n\cdot\sum_{i\in\{1,\dots,h\}}\frac{i}{2^i} && \mid\ i < 1.5^i \text{ מספר טבעי ולכן } i \\
&< n\cdot\sum_{i\in\{1,\dots,h\}}\frac{1.5^i}{2^i} \\
&= n\cdot\sum_{i\in\{1,\dots,h\}}(\frac{3}{4})^i \\
&< 4n && \mid\ \lim_{h\to\infty}\sum_{i\in\{1,\dots,h\}}(\frac{3}{4})^i < 4
\end{aligned}$$

בתרגול זה נעבור על מהלך האלגוריתם ויישומו. כמו כן נשתמש במספר האיברים בערימה n כדי לחסום את גובה הערימה h ולהפך.

תרגיל 1

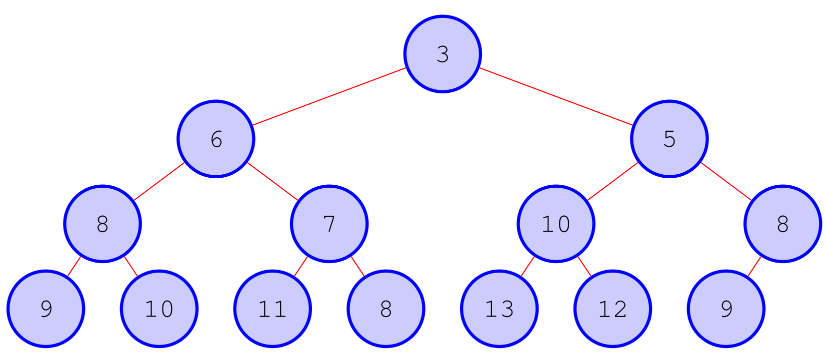
להלן מערך:

$$[3, 10, 5, 6, 7, 13, 8, 9, 8, 11, 8, 10, 12, 9]$$

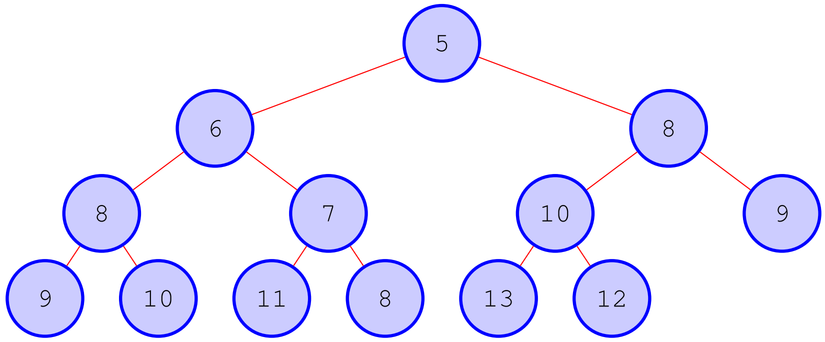
(א) בנו את הערימה המתאימה ע"פ האלגוריתם של פלויד.

(ב) הוציאו את המינימום ולאחר מכן הכניסו את האיבר 5.

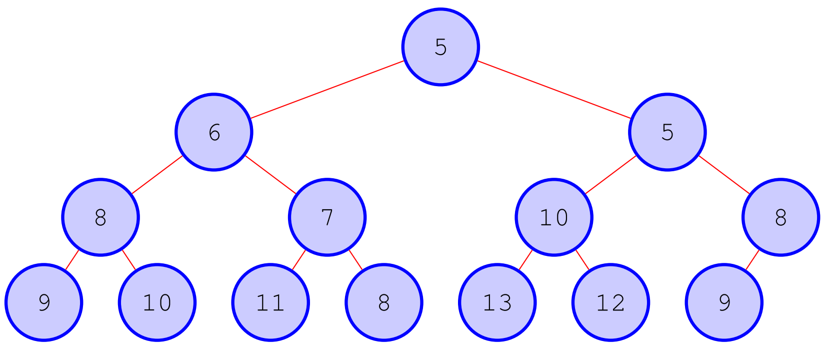
פתרון:



איור 10: בסוף האלגוריתם של פלויד נקבל את הערימה הזאת.



איור 11: לאחר הוצאת השורש נקבל את הערימה הזאת.



איור 12: לאחר הכנסת האיבר 5 נקבל את הערימה הזאת.

תרגיל 2

נתונות שתי ערימות בינאריות H_1 ו H_2 כל אחת בת n איברים. הציעו אלגוריתם המקבל את שתי הערימות ומחזיר ערימה חדשה עם כל $2n$ האיברים בסיבוכיות $\mathcal{O}(n)$.

פתרון:

אם ננסה להכניס את האיברים מערימה אחת לשנייה בזה אחר זה, נכניס n איברים בסיבוכיות $\mathcal{O}(\lg n)$ לפעולת הכנסה, סה"כ $\mathcal{O}(n \lg n)$.

יש דרך יותר יעילה, נכניס את כל האיברים בערימות למערך אחד בגודל $2n$ איברים ונבנה ממנו ערימה בעזרת האלגוריתם של פלויד בזמן $\mathcal{O}(2n) = \mathcal{O}(n)$.

תרגיל 3

ידוע כי גובה ערימה מסויימת בת n איברים הוא h.

מהו טווח מספר האיברים של הערימה? הציעו חסם עליון וחסם תחתון על n המסתכמים על h בלבד.

פתרון:

בערימה בעלת גובה h יש $h+1$ רמות (מהשורש שהוא רמה 0 עד העלים) כאשר מלבד הרמה האחרונה, כל הרמות מלאות, כלומר בעלות 2^i איברים ברמה הi.
כדי לחשב את החסם העליון נניח שהרמה האחרונה מלאה. מכאן שסך האיברים ברמות המלאות הוא $\sum_{i\in\{0,1,\ldots h\}} 2^i = 2^{h+1}-1$ ראו הסבר על הנוסחה בהמשך. אם כך הרי ש $n \in \mathcal{O}(2^{h+1}-1)$.
כדי לחשב את החסם התחתון נניח שהרמה האחרונה היא קטנה ככל האפשר, כלומר בעלת איבר אחד. מזל שממש הרגע מצאנו נוסחא למספר האיברים בערימה עם $h+1$ רמות מלאות, נציב h בנוסחה ונקבל את החסם התחתון.
אם כך הרי שסך האיברים בערימה כזו הוא $2^h-1+1=2^h$, השגנו איפה חסם תחתון על מספר האיברים $n \in \Omega(2^h)$.
לסיכום:

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

פירוט של הנוסחה לחישוב סכום של טור הנדסי.

$$\begin{aligned}
\sum_{i\in\{0,1,\ldots h\}} 2^i &= 2^0+2^1+2^2+\cdots+2^h \\
&= 1\cdot(2^0+2^1+2^2+\cdots+2^h) \\
&= (2-1)\cdot(2^0+2^1+2^2+\cdots+2^h) \\
&= 2\cdot(2^0+2^1+2^2+\cdots+2^h) - 1\cdot(2^0+2^1+2^2+\cdots+2^h) \\
&= 2^1+2^2+2^3+\cdots+2^h+2^{h+1}) - 2^0-2^1-2^2-\cdots-2^h) \\
&= 2^{h+1}-2^0 \\
&= 2^{h+1}-1
\end{aligned}$$

בצורה דומה אפשר להראות שסיבוכיות האלגוריתם של פלויד מתכנסת ל $\mathcal{O}(n)$:

$$\begin{aligned}
\sum_{i\in\{0,1,\ldots \log n\}} (\frac{3}{4})^i &= (\frac{3}{4})^0+(\frac{3}{4})^1+(\frac{3}{4})^2+\cdots+(\frac{3}{4})^{\log n} \\
&= \frac{4}{4}\Big((\frac{3}{4})^0+(\frac{3}{4})^1+(\frac{3}{4})^2+\cdots+(\frac{3}{4})^{\log n}\Big) \\
&= 4(1-\frac{3}{4})\Big((\frac{3}{4})^0+(\frac{3}{4})^1+(\frac{3}{4})^2+\cdots+(\frac{3}{4})^{\log n}\Big) \\
&= 4\Big((\frac{3}{4})^0+(\frac{3}{4})^1+(\frac{3}{4})^2+\cdots+(\frac{3}{4})^{\log n}) - \frac{3}{4}((\frac{3}{4})^0+(\frac{3}{4})^1+(\frac{3}{4})^2+\cdots+(\frac{3}{4})^{\log n})\Big) \\
&= 4\Big((\frac{3}{4})^0-(\frac{3}{4})^{\log n+1})\Big) \\
&< 4\Big((\frac{3}{4})^0\Big) \\
&= 4
\end{aligned}$$

שימו לב שהחסם קבוע ללא תלות ב n בכלל.

תרגיל 4

הראו שהגובה של ערימה בת n איברים הוא $\lfloor \lg n \rfloor$.

פתרון:

נשתמש בחסמים שמצאנו בתרגיל הקודם, ראינו שבערימה בגובה h מספר האיברים n חסום ב:

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

יש לנו שני אי שוויונים, נפתור אותם כדי לבטא את h באמצעות n. נוציא $\lg$ מאי השוויון התחתון כדי לקבל $h \leq \lg n$.
קצת קשה להוציא $\lg$ מאי השוויון העליון, אבל אל חשש! אפשר להחליף אותו בחסם עליון יותר נוח. הרי זה ברור ש $2^{h+1} - 1 < 2^{h+1}$ נכון? אז קיבלנו מאי השוויון הזה חסם טיפה פחות הדוק $\lg n < h + 1$ אבל הוא יספיק כדי לענות על השאלה.
כדי להוכיח שהגובה הוא בדיוק $\lfloor \lg n \rfloor$ נתבונן בתחום שקיבלנו:

$$\begin{array}{rcccl} h \leq & \lg n & < & h+1 \\ \lg n - 1 < & h & \leq & \lg n \end{array}$$

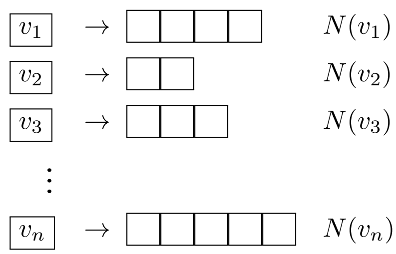
אנחנו יודעים ש h הוא מספר שלם ומכיוון שבתחום $(\lg n - 1, \lg n]$ קיים רק מספר שלם אחד, בהכרח $h = \lfloor \lg n \rfloor$.

6 גרפים – ייצוג וחיפוש

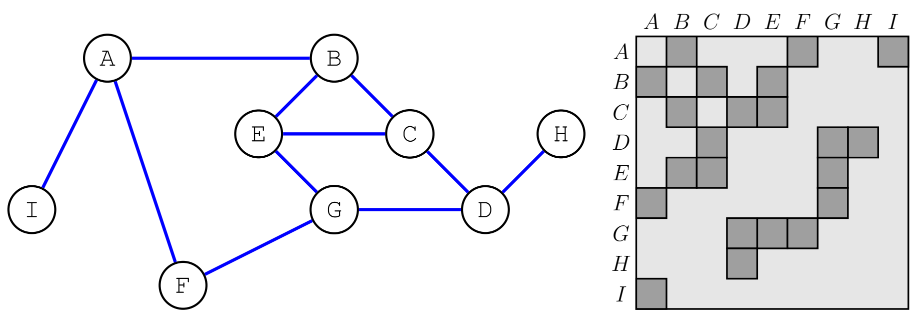
גרפים הם אובייקטים מתמטים בעזרתם ניתן למדל *יחסים בינארים*[ט] בין אובייקטים אחרים להם נקרא *צמתים*. את קבוצת הצמתים נסמן ב V ואת היחס הבינארי נבטא עם קשתות $E \subseteq V \times V$. עבור $v,u \in V$ נסמן קשת $e \in E$ כקשת *לא-מכוונת* ע"י $e = \{v,u\}$ וכקשת מכוונת ע"י (v,u) כש e יוצאת מ v ונכנסת ל u. נקרא לכל הצמתים הסמוכים ל v *השכנים* של v ולקבוצה המכילה $N(v)$ את כולם *השכונה* של v. לגודל השכונה $|N(v)|$ אנו קוראים *הדרגה* של v.

נתקלנו בתרגולים הקודמים בעצים, גרפים ללא מעגלים, בהמשך הקורס נלמד על כמה בעיות שניתן לבטא אותן באמצעות גרפים, נלמד גם לנצל תכונות של סוגים שונים של גרפים כדי לפתור אותן בצורה יעילה. בתרגול הזה נתמקד בייצוג של גרפים, איך אנחנו שומרים אותם בזכרון המחשב, כמו גם שני אלגוריתמי חיפוש שחובה להכיר – BFS ו DFS.

רשימת שכנויות היא למעשה מערך של רשימות שכנויות כאשר בתא i במערך נשמור את כל השכנים של $v_i \in V$ ברשימה מקושרת $N(v_i)$.



מטריצת שכנויות היא מטריצה בינארית $A \in \{0,1\}^{n \times n}$ המתארת גרף כש $A[i,j] = 1$ אם ורק אם $(i,j) \in E$. כמובן שאם A היא מטריצת שכנויות של גרף לא מכוון המטריצה תהיה *סימטרית*.



לרוב נסמן את עוצמת קבוצה הצמתים $|V|$ עם n ואת עוצמת קבוצת הקשתות ב m. קל לראות שמספר הקשתות בגרף מכוון חסום ב n^2. בתרגיל בית 2 הוכחתם שמספר הקשתות בגרף לא-מכוון חסום ב $\binom{n}{2}$[`].

[ט] יחס בינארי בין קבוצה A לקבוצה B הוא קבוצה של זוגות סדורים (a,b) כש $a \in A$ ו $b \in B$.
[`] זוכרים את התרגיל עם מספר ההיפוכים? איך זה מתקשר לגרפים לא מכוונים?

אחד הדברים הבסיסיים שנרצה לדעת עבור גרפים הוא האם הם *קשירים* גרף קשיר הוא גרף בו עובר *מסלול* בין כל זוג צמתים. כשמסלול הוא סדרת צמתים כך שבין כך שיש קשת בין כל זוג צמתים עוקבים. *מעגל* הוא מסלול המתחיל ומסתיים באותו צומת, נגיד שמסלול הוא *פשוט* אם אין בו מעגלים.

נכיר כעת שני אלגוריתמים לבדיקת קשירות BFS (חיפוש לרוחב) ו DFS (חיפוש לעומק).

DFS$(G=(V,E),v)$

```
Mark i as visited.
for u ∈ N(v) do
    if u is not visited then
        DFS(G, u)
```

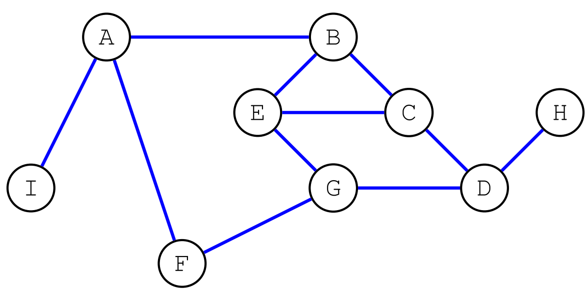
BFS $(G=(V,E),v)$

```
Create empty list L.
Add v to the tail of L.
while L ≠ ∅ do
    Let u be the head of L.
    Mark u visited and remove it from L.
    for w ∈ N(u) do
        if w is non-visited then
            Add w to the tail of L.
```

הסיבוכיות של שניהם היא $\mathcal{O}(|V|+|E|)$, באיזו טכניקה שלמדנו אפשר להשתמש כדי להוכיח את זה?

תרגיל 1

נתון הגרף G הבא:



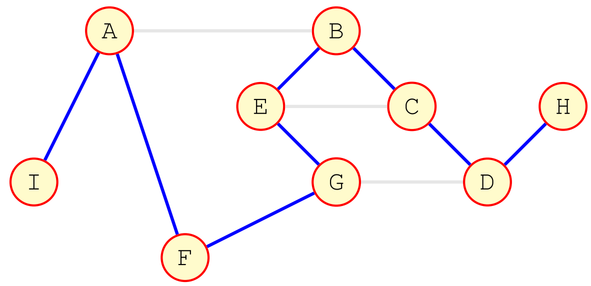
(א) כתבו את גרף ה DFS המתקבל כאשר מתחילים בצומת A.

(ב) כתבו את גרף ה BFS המתקבל כאשר מתחילים בצומת I.

(ג) מהו סוג הגרפים? האם הם יחודיים ל G?

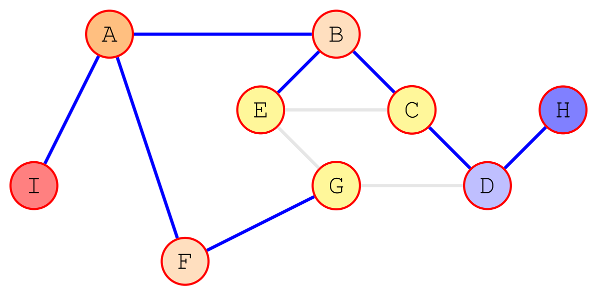
פתרון:

(א) נריץ את אלגוריתם DFS ונשמור בגרף רק את הקשתות עליהן עברנו.



איור 13: עץ DFS המתקבל בסריקה של G המתחילה בA.

(ב) כמו כן נריץ את אלגוריתם BFS ונשמור בגרף רק את הקשתות עליהן עברנו.



איור 14: עץ BFS המתקבל בסריקה של G המתחילה בI.

(ג) ניתן לראות כי שני הגרפים הם עצים. זה כמובן לא מקרה, לא שמרנו בגרף קשתות בין צמתים בהם ביקרנו ולכן לא יצרנו מעגלים.

שני העצים אינם ייחודיים לגרף זה, מספר עצי החיפוש האפשריים תלוי במספר המעגלים בגרף.

תרגיל 2

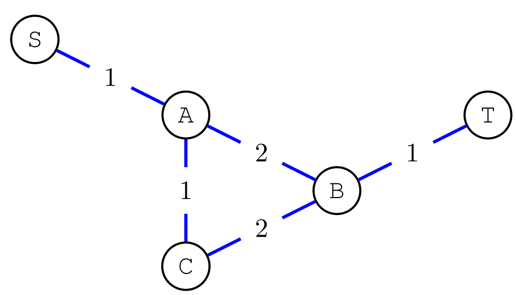
יש n ערים ו m כבישים. הניחו שכל דרך היא דו-כיוונית. הציעו אלגוריתם לעיבוד המידע (preprocessing) בזמן $\mathcal{O}(n+m)$ כך שלאחר מכן אפשר לענות בזמן $\mathcal{O}(1)$ על שאלה "האם ניתן להגיע מעיר a לעיר b".

פתרון:

אם ניתן להגיע מעיר a לעיר b הרי ששתיהן באותו רכיב קשירות. כדי לענות על השאלה בזמן קבוע נצטרך לדעת עבור כל עיר מה הוא רכיב הקשירות שלה.
נבנה מערך A בגודל n ונריץ מצומת אקראית (עיר אקראית) את אחד מהאלגוריתמים BFS או DFS (שניהם יכולים לחשב רכיבי קשירות בזמן $\mathcal{O}(n+m)$). כל הצמתים שהגלנו $V' \subseteq V$ שייכים לרכיב קשירות אחד, לכן נסמן עבור כל $v \in V'$ בתא המתאים במערך $A[v] = i$. כעת נמחק את כל הצמתים שגילינו מהגרף, נגדיל את i ב1 ונחזור על הפעולה עד שגילינו את כל רכיבי הקשירות בגרף.
בניתוח גלובלי נבחין שלא משנה מה הוא מספר רכיבי הקשירות בגרף, זמן הריצה של כל החזרות יהיה בסה"כ $\mathcal{O}(n+m)$ כי אנחנו עוברים על כל צומת פעם אחת בלבד, כמו גם על כל קשת.
כעת אנחנו מוכנים לשאילתות, במידה והערים a ו b נמצאות באותו רכיב קשירות נחזיר כן אחרת נחזיר לא. ניתן לראות האם מתקיים $A[a] = A[b]$ בזמן $\mathcal{O}(1)$.

תרגיל 3

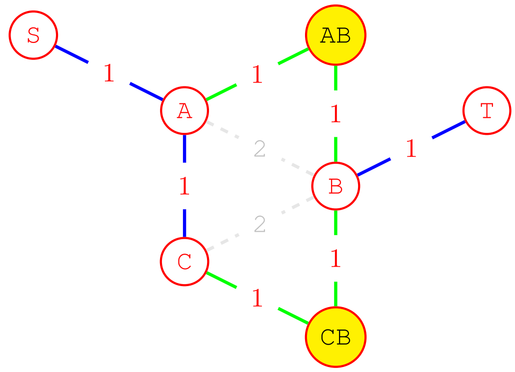
נתון גרף לא מכוון ממושם ברשימת שכנויות. לכל צלע יש משקל $w : E \to \{1,2\}$.
לדוגמא:



הציעו אלגוריתם שמוצא מסלול הכי קל מקודקוד v לקודקוד u בזמן $\mathcal{O}(|V| + |E|)$.

פתרון:

ידוע לנו שBFS מוצא את המסלול הכי קצר מקודקוד כלשהו v לשאר קודקודי הגרף רק אם הגרף לא ממושקל.
נוכל לנצל את העובדה הזו כדי להמיר את הגרף הנתון לגרף לא ממושקל G', כך שהמסלול הכי קצר בגרף החדש יתורגם למסלול הכי *קל* בגרף המקורי.
כדי לעשות זאת, ראשית נעתיק ל G' את כל הקשתות והצמתים של G. נעבור על כל קשתות הגרף G, עבור כל קשת $\{v,u\} \in E$ שמקיימת $w(\{v,u\}) = 2$, נסיר את $\{v,u\}$ מ G' ונוסיף צומת vu כמו גם שתי קשתות $\{v, vu\}$ ו $\{vu, u\}$. כשנסיים לגרף G' יהיו $\mathcal{O}(n+m)$ צמתים ו $\mathcal{O}(2m)$ קשתות.

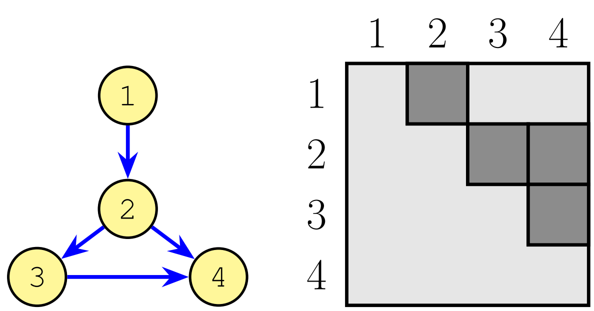


איור 15: הבנייה של G' מ־G.

כעת נוכל להריץ BFS על G' בזמן $\mathcal{O}(n'+m')=\mathcal{O}(n+m+2m)=\mathcal{O}(n+m)$. קל להפוך כל מסלול קצר ב־G' למסלול קל ב־G, פשוט נמחק את הצמתים החדשים.

תרגיל 4

מיון טופולוגי של גרף מכוון וקשיר $G=(V,E)$ הוא סידור לנארי של V כך שאם קיימת קשת (v_1,v_2) אז v_1 יהיה לפני v_2 (כלומר כל צאצא תמיד יבוא אחרי האב שלו). מיון טופולוגי קיים רק בגרף ללא מעגלים. למיון טופולוגי יש חשיבות להרבה משימות כגון ניהול פרויקטים (איזה משימה קודמת לאחרת), סדר תהליכים במחשב ועוד.



איור 16: הסדר $1,2,3,4$ הוא מיון טופולוגי בגרף.

אלגוריתם למציאת מיון טופולוגי בגרף:

1. אתחול: $k=0$
2. בנה מערך עזר בגודל $|V|$

3. כל עוד קיימים מקורות (קודקודים להם אין קשתות נכנסות) בצע:
   (א) מצא מקור v, וודא שהוא לא נמצא במערך העזר.
   (ב) קדם את k ב-1, תן ל-v מספר k.
   (ג) הכנס את v למערך העזר
   (ד) מחק את v ואת הקשתות היוצאות ממנו.
4. אם $k=|V|$ אז קיים סדר טופולגי, אחרת בגרף יש מעגל מכוון.

הגרף ממומש במטריצת שכנויות. מה תהיה סבוכיות האלגוריתם במקרה זה?

פתרון:

זמן הריצה יהיה $\mathcal{O}(|V|^3)$ ראו ניתוח:

1. אתחול: $k=0$ $\mathcal{O}(1)$
2. בנה מערך עזר בגודל $|V|$ $\mathcal{O}(|V|)$
3. כל עוד קיימים מקורות (קודקודים להם אין קשתות נכנסות) בצע: $\mathcal{O}(|V|)$
   א. מצא מקור v וודא שהוא לא נמצא במערך העזר. $\mathcal{O}(|V|^2)$
   כדי לבדוק שצומת הוא מקור נצטרך לבדוק במטריצת השכנויות את הטור שלו ולוודא שכולו מלא ב0.
   ב. קדם את k ב-1, תן ל-v מספר k. $\mathcal{O}(1)$
   ג. הכנס את v למערך העזר $\mathcal{O}(1)$
   ד. מחק את v ואת הקשתות היוצאות ממנו. $\mathcal{O}(|V|)$
   אפס את השורה של v.
4. אם $k=|V|$ אז קיים סדר טופולוגי, אחרת בגרף יש מעגל מכוון. $\mathcal{O}(1)$

7 עץ פורש מינימלי

למדנו בתרגול הקודם על גרפים ובדיקות קשירות, ראינו שהאלגוריתמים BFS ו DFS מוצאים עץ הנפרש בין כל צמתי רכיב הקשירות בו הם מופעלים. נבחין כי עץ הוא הגרף עם מספר הקשתות הקטן ביותר היכול לחבר n צמתים, מצליחים לראות למה זה נכון? כידוע, גרפים מתאימים למידול של רשתות (לוגיסטיקה, תקשורת, תחבורה) בהן נשאף לחבר בין מספר אובייקטים בעלות מינימלית, כשעלויות החיבור בין כל שני אובייקטים נתונות. מתעוררת השאלה הטבעית: בהנתן גרף ממושקל (עם משקלים על הקשתות) מה הוא העץ הפורש המינימלי של הגרף?

אפשר לנסח את בעיית העץ המינימלי כ *בעיית הכרעה*[א]:

בעיית העץ הפורש

קלט: גרף ממושקל G ומספר ממשי W.

שאלה: האם קיים בG עץ פורש במשקל כולל של W או פחות?

דרך פשוטה למדי להכריע את בעיית העץ הפורש היא למצוא עץ פורש במשקל נמוך מW. דרך יותר אלגנטית תהיה למצוא עץ פורש עם משקל מינימלי (לא קיים בגרף עץ פורש עם משקל נמוך משלו) ולבדוק האם משקלו נמוך מ W. נאיבית נוכל לעבור על כל קבוצה של $n-1$ קשתות, לבדוק אם הן בונות עץ פורש ואם משקלו הוא הקטן ביותר שמצאנו עד כה, אך זה יכול לקחת זמן אקספוננציאלי $\mathcal{O}(\binom{m}{n-1})$ בגרפים כלליים. למזלנו יש אלגוריתם המוצא עץ פורש מינימלי בזמן יעיל הרבה יותר!

```
Prim(G = (V, E))
T = ∅
Mark arbitrary vertex as visited.
while there exists a non-visited vertex in G do
    Find minimum weighted edge between a visited
     vertex v and a non-visited vertex u.
    Mark u as visited and add {v,u} to T
return T
```

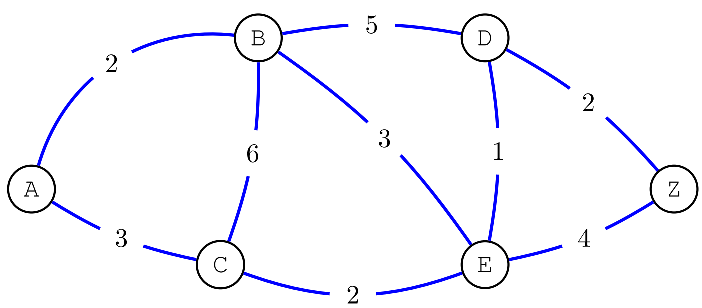
בתרגול זה נעבור על מהלך האלגוריתם של פרים, נבין מה הוא מחשב (וגם מה הוא **לא** מחשב) ולמה הוא נכון.

[א] בעיית הכרעה היא שאלה של כן/לא וקלט, כאשר כל סוגי הקלטים האפשריים מתחלקים לקלטים של "כן" וקלטים של "לא". עוד על בעיות הכרעה בהמשך!

```
HeapPrim(G = (V, E))
1: T = ∅
2: Create an empty heap H
3: Mark arbitrary vertex i as visited.
4: Add all edges {i, j} ∈ E to H.
5: while there exists a non-visited vertex in G do
6:     {i, j} = H.popRoot().
7:     if i and j are both visited then Go To 6
8:     Let x be the non-visited vertex of {i, j} .
9:     Mark x as visited and add {i, j} to T.
10:    Add all edges {x, y} ∈ E to H.
11: return T
```

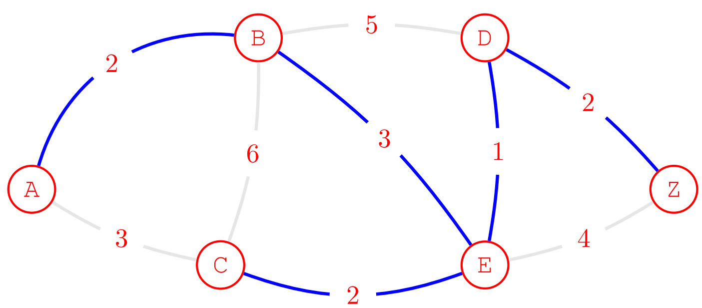
תרגיל 1

מצאו עץ פורש מינימאלי בגרף הבא:



פתרון:

העץ הפורש המינימלי:



שימו לב שהוא לב שהוא לא העץ הפורש המינימלי היחיד של הגרף, אנחנו יכולים להחליף בין BE ל AC כדי לקבל עץ פורש שקול.

תרגיל 2

נתון גרף לא מכוון קשיר וממושקל בעל n צמתים. תארו אלגוריתם בזמן $\mathcal{O}(n)$ המוצא קשת ספציפית שהורדת משקלה ב-1 יוריד את משקל העץ הפורש המינימאלי של הגרף.

פתרון:

נבחר צומת v כלשהו, נעבור על כל הקשתות היוצאות ממנו נמצא את הקשת המינימלית $\{v,u\}$ ונבחר אותה. בחירתנו נכונה מאותה סיבה שהאלגוריתם של פרים מחשב עץ פורש מינימלי. בכל איטרציה i של פרים העץ הזמני T_i שמתקבל ע"י מחיקת כל הצמתים שאינם ב T_i. זה נכון בברור לצומת שבחרנו באקראי (כי עדיין אין לעץ קשתות), וקל לראות שזה נכון עבור כל צומת שאנו מוסיפים לעץ. ההוכחה המלאה בהרצאה.

ברור לנו שכל צומת בגרף מתווסף לפרים באיזושהי איטרציה. מאחר שראינו שפרים נכון, אם פרים בוחר את $\{v,u\}$ זה אומר שהיא בעץ פורש מינימלי, לכן אם נוריד את משקלה קל לראות שהוא ירד משקלו. נבחן את המקרים האפשריים:

1. הצומת התווסף באקראי להיות הראשון – במקרה זה פרים יבחר את הקשת המינימלי שיוצאת ממנו.
2. הצומת התווסף אחרון – כלומר כל השכנים של הצומת כבר התווספו ולכן הקשת המינימלית מוכרת לפרים והוא יבחר בה.
3. הצומת נבחר באמצע, נחלק שוב לשני מקרים:
   (א) הקשת המינימלית של v היא הקשת המינימלית שמוכרת לפרים באיטרציה i – במקרה זה פרים יבחר בה.
   (ב) נניח במקרה הגרוע שפרים לא בוחר בה עד שהוא מסיים, כלומר קיים עץ פורש מינימלי T' שלא מכיל את $\{v,u\}$. אבל ראינו שפרים נכון לכן T' חייב להיות שקול לעץ שמכיל את $\{v,u\}$. כלומר אם הורדנו את משקלה בהכרח הורדנו את משקל העץ הפורש המינימלי.

הסיבוכיות של האלגוריתם היא $\mathcal{O}(n)$ משום שבמקרה הגרוע נעבור על $n-1$ שכנים של v.

תרגיל 3

נתון גרף לא מכוון קשיר וממושקל, שכל המשקלים בו הם 0 או 1. תארו אלגוריתם הבודק אם לגרף זה יש עץ פורש שסכום המשקלים שלו הוא 1.

פתרון:

למעשה אנחנו מחפשים עץ פורש שבו כל הקשתות חוץ מאחת במשקל 0.

1. נמחק מ G את כל הקשתות במשקל 1 ונמצא את רכיבי הקשירות של הגרף החדש בזמן $\mathcal{O}(n+m)$.

2. אם מצאנו יותר משני רכיבי קשירות נחזיר "לא" משום שנצטרך לפחות שתי קשתות במשקל 1 כדי לחבר את כל הרכיבי לעץ אחד.

3. אם נוצרו שני רכיבי קשירות נחזיר "כן" משום שנתון לנו שG קשיר ולכן בהכרח קיימת ביניהם קשת במשקל 1.

4. אם נוצר רכיב קשירות אחד נחזיר "כן" אם ורק אם קיימת בG קשת במשקל 1, בדיקה של זה יכולה להתבצע ב$\mathcal{O}(m)$ זמן.

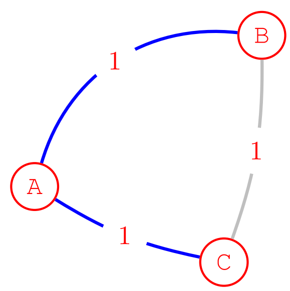
תרגיל 4

הוכיחו או הפריכו את הטענה:

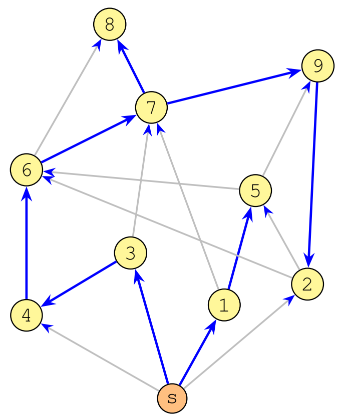
בהינתן עץ פורש מינימלי T בגרף $G = (V, E)$ ושני צמתים $v, u \in V$, המסלול הקצר ביותר בין v לu כלול בעץ T.

פתרון:

הטענה לא נכונה. ניקח כדוגמת נגד את הגרף שהוא מעגל ממשוקל אחיד על שלושה צמתים (הגרף המלא על שלושה צמתים). כל צמד קשתות בגרף זה הוא עץ פורש מינימלי אך אינו מכיל את המסלול הקצר ביותר בין שני העלים של העץ, אחרת היה בו מעגל והוא לא היה עץ.



8 מסלולים קצרים



ראינו שאלגוריתם BFS יכול למצוא את המסלול הקצר ביותר בגרפים לא ממושקלים. אך כשהגרפים ממושקלים לרוב נקרא "קצר" למסלול הקל ביותר – המסלול עם סך משקלי הקשתות הנמוך ביותר מאלו שמחברים בין שני צמתים. מציאת מסלול קצר בין שני צמתים היא משימה שמופיעה באופן טבעי בתחומים רבים אחרים מעבר למערכות ניווט ותקשורת. ניתן לנסח בעיות רבות כמו מציאת מילים דומות או קטעי *DNA* דומים בתור בעיית מציאת מסלולים קצרים בגרפים.

בעיית המסלול הקצר

קלט: גרף ממושקל G שני צמתים s ו t ומספר ממשי W.

שאלה: האם קיים בG מסלול קצר בין s ל t במשקל כולל של W או פחות?

בתרגול זה נכיר שני אלגוריתמים המכריעים בריצה אחת את בעיית המסלול הקצר מצומת כלשהו s לכל שאר צמתי הגרף:

BellmanFord$(G=(V,E),s)$

for $v \in V$ **do**
  father$(v) = \varnothing$
  dist$(v) = \infty$.
dist$(s) = 0$
counter $= 1$
while counter $< n$ **do**
  for $(v,u) \in E$ **do**
    if dist$(v) + w(v,u) <$ dist(u) **then**
      dist$(u) =$ dist$(v) + w(v,u)$
      father$(u) = v$
  counter $=$ counter $+ 1$

```
Dijkstra(G = (V, E), s)
for v ∈ V do
    Mark v as not-visited
    father(v) = ∅
    dist(v) = ∞.
dist(s) = 0
Create Heap H with all vertices sorted according to dist()
while There exists a not-visited vertex  do
    v = H.PopRoot()
    Mark v as visited
    for not-visited u ∈ N(v) do
        if dist(v) + w(v, u) < dist(u) then
            dist(u) = dist(v) + w(v, u)
            H.UpdateKey(u) = dist(u)
            father(u) = v
```

האלגוריתם של דייקסטרה מניח שמשקלי הקשתות אינם שליליים ומנצל הנחה זאת למציאת מסלולים קצרים בזמן $\mathcal{O}(n\lg n + m\lg n)$ בגרפים, בין אם הם מכוונים ובין אם לא. לעומתו אלגוריתם בלמן-פורד מסוגל למצוא בזמן $\mathcal{O}(m \cdot n)$ מסלולים קצרים גם אם קיימים משקלים שליליים בגרף, אך אם הגרף לא מכוון נוכל להניח שאין בו משקלים שליליים. הסיבה היא שבגרפים לא מכוונים קיומה של קשת עם משקל שלילי מעיד על קיומו של *מעגל שלילי* המעקר ממשמעות את המושג מסלול קצר. אלגוריתם בלמן פורד מסוגל לזהות מעגלים שליליים.

בתרגול זה נבחן אילו פעולות נוכל להפעיל על פונקציית המשקל מבלי לשנות את יחסי המסלולים בגרף. כמו כן איך לבצע התאמות לדייקסטרה ולבלמן-פורד כך שיפתרו בעיות קרובות לבעיית המסלול הקצר ומתי לבחור באחד לעומת האחר.

קצת *Fun Facts* – אלגוריתם בלמן-פורד פותח באופן עצמאי ע"י ריצ'ארד בלמן ולסטר פורד, השני פרסם את הטכניקה שלו כשנתיים לפני בלמן.
זוכרים ערימות בינאריות? אז דייקסטרה[יב] לא השתמש בהן בתיאור האלגוריתם שלו שרץ בזמן $\mathcal{O}(n^2)$. שימוש בערימה בינארית מאיץ את האלגוריתם של דייקסטרה עבור גרפים דלילים (יותר מדוייק לגרפים בעלי $o(\frac{n^2}{\lg n})$ קשתות), עבור גרפים מלאים דייקסטרה ללא ערימה יהיה מהיר יותר ע"פ ניתוח המקרה הגרוע ביותר.
הערה: בשנת 1984 פיתחו המדענים רוברט טרג'ן ומיכאל פלדמן ערימה שמתגברת על הקושי של ערימה בינארית בהאצת דייקסטרה על גרפים צפופים. הערימה נקראת ערימת פיבונאצ'י ובאמצעותה זמן הריצה של דייקסטרה ירד ל $O(m + n\lg n)$ [יג], תוצאה שלא השתפרה עד היום.

[יב] https://www.cs.yale.edu/homes/lans/readings/routing/dijkstra-routing-1959.pdf באותו מאמר דייקסטרה הציג את האלגוריתם שלו לבעיית המסלול הקצר, וגם...! לעוד בעיה שלמדנו בקורס! מזהים איזו?
[יג] https://www.youtube.com/watch?v=6JxvKfSV9Ns&t=344s הערימה מבצעת את פעולות ההכנסה והעדכון בזמן קבוע באמצעות טכניקה מתוחכמת למיזוג ערימות קטנות, הסקרנים מבינכם יהנו מהאנימציה של האלגוריתם בסרטון המצורף.

תרגיל 1

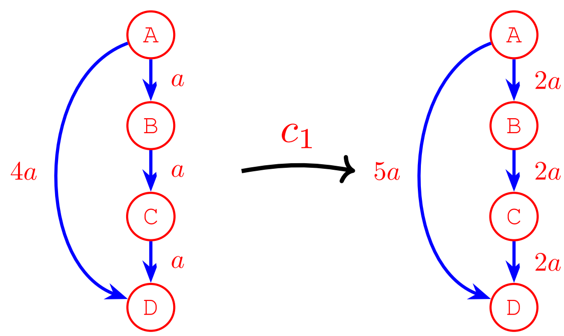
נתון גרף קשיר ולא מכוון $G=(V,E)$ ופונקציית משקל w על הקשתות, לפיה כל הקשתות בעלות משקל חיובי. הוכיחו/הפריכו את הטענות הבאות:

(א) נגדיר פונקציית משקל חדשה c_1 כך ש: $c_1(e)=a+w(e)$ (a חיובי). אם P הוא המסלול הקצר ביותר בין זוג קודקודים לפי w, הוא גם הקצר ביותר לפי c_1.

(ב) כנ"ל אם פונקציית המשקל היא $c_2(e)=a\cdot w(e)$ (a חיובי).

פתרון:

(א) הטענה לא נכונה, נפריך באמצעות דוגמת נגד:



ניתן לראות כי המסלול ABCD קצר מהמסלול AD ע"פ w אך לא ע"פ c_1.

(ב) הטענה נכונה, נוכיח אותה עבור זוג צמתים אקראי בגרף. יהי P המסלול הקצר ביותר בין s ל t ע"פ פונקציית המשקל w, כלומר עבור כל מסלול Q מתקיים $\sum_{e\in P} w(e) \le \sum_{e\in Q} w(e)$. נראה ש$P$ קצר או שווה לכל מסלול אחר Q גם ע"פ c_2.

$$\begin{aligned}
\sum_{e\in P} c_2(e) &= \sum_{e\in P} a\cdot w(e) && \text{| נוציא גורם משותף.}\\
&= a\cdot \sum_{e\in P} w(e) && \text{| ידוע כי } \sum_{e\in P} w(e) \le \sum_{e\in Q} w(e)\\
&\le a\cdot \sum_{e\in Q} w(e) \\
&= \sum_{e\in Q} a\cdot w(e) \quad = \sum_{e\in Q} c_2(e)
\end{aligned}$$

קיבלנו ש $\sum_{e\in P} c_2(e) \le \sum_{e\in Q} c_2(e)$, מש"ל.

תרגיל 2

נתון גרף לא ממושקל, מכוון $G=(V,E)$ ותת קבוצה $W\subseteq V$ ונתונים שני צמתים $s,t\in V$. תארו אלגוריתם יעיל למציאת מסלול מ-s ל-t המבקר במספר מינימלי של צמתים מ- W.

פתרון:

אנחנו רוצים לעודד את האלגוריתם שלנו לעבור בכמה שפחות קשתות שמובילות אותו לצמתים מקבוצה W. נקבע פונקציית משקל על קשתות הגרף $w: E \to \{0,1\}$ ש"קונסת" את האלגוריתם על כל קשת שמובילה לצומת לא רצוי:

$$w((v,u)) = \begin{cases} 0, & \texttt{if } u \notin W \\ 1, & \texttt{otherwise} \end{cases}$$

נפעיל את דייקסטרה על צומת s ונעצור כשהגענו ל t. מאחר ודייקסטרה מוצא את המסלול הקצר ביותר הוא ימזער את מספר הקשתות שהובילו אותו לצמתים ב u.

תרגיל 3

נתון גרף מכוון $G=(V,E)$ עם פונקציית משקל $w: E \to \mathbb{R}$, וקודקוד $s \in V$. נגדיר אורך של מסלול להיות משקל הקשת הכבדה ביותר במסלול. הציעו אלגוריתם יעיל למציאת את המסלולים ארוכים ביותר מ- s ל- $v \in V$ לפי הגדרה זו וכתוב מה סיבוכיות זמן הריצה שלה.

פתרון:

מאחר וייתכנו משקלים שליליים נשתמש באלגוריתם בלמן-פורד עם עדכון קל, נשנה את פונקציית המרחק $\texttt{dist}(i)$ כך שתשמור עבור כל צומת את הקשת הכבדה ביותר במסלול שהוביל אליו. החלפנו באלגוריתם פעולות שמתבצעות בזמן קבוע באחרות שמתבצעות בזמן קבוע גם כן, לכן סיבוכיות הזמן זהה לסיבוכיות של בלמן-פורד המקורי $\mathcal{O}(nm)$.

$\texttt{BellmanFord}(G=(V,E),s)$

for $v \in V$ **do**
  $\texttt{father}(v) = \varnothing$
  $\texttt{dist}(v) = -\infty$.
$\texttt{dist}(s) = 0$
$\texttt{counter} = 1$
while $\texttt{counter} < n$ **do**
  for $(v,u) \in E$ **do**
    if $\texttt{dist}(u) < \max\{\texttt{dist}(v),\ w(v,u)\}$ **then**
      $\texttt{dist}(u) = \max\{\texttt{dist}(v),\ w(v,u)\}$
      $\texttt{father}(u) = v$
  $\texttt{counter} = \texttt{counter} + 1$

תרגיל 4

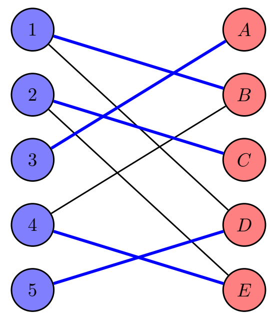
יהי $G=(V,E)$ גרף מכוון עם משקלים אי שליליים על הקשתות, פרט לקשת אחת $e=(u,v)$ בעלת משקל שלילי. ונתון קודקוד s. מצא את המרחק מ s ליתר הקודקודים בגרף ע"י שימוש באלגוריתם דייקסטרה. ניתן להניח שאין ב G מעגלים שליליים.

פתרון:

נוריד את הקשת (u,v) ונשתמש בדייקסטרה כדי למצוא את המסלולים הקצרים מs לכל צמתי הגרף, נשמור עבור צומת את המרחק במשתנה d_1. כעת נפעיל את דייקסטרה מצומת v ונשמור עבור כל צומת את המרחק במשתנה d_2. עבור כל צומת x נחשב את המרחק המינימלי ע"י $d(x) = \min\{d_1(x),\ \ d_1(u) + w(u,v) + d_2(x)\}$.

מאחר והפעלנו את האלגוריתם של דייקסטרה מספר קבוע של פעמים (פעמיים) סיבוכיות הזמן זהה לדייקסטרה עם ערימה $\mathcal{O}(n \log n + m \log n)$ או בלי ערימה $\mathcal{O}(n^2)$.

9 שידוך מקסימלי



בתרגול זה נכיר את בעיית השידוך וגם אלגוריתם שפותר מקרה פרטי שלה המוכר בבעיית ההשמה (*Assignment Problem*). בבעיית ההשמה נתונות לנו n מכונות ו-n משימות ואנו מתבקשים *לשדך* משימות לכמה שיותר מכונות כך שכל מכונה תהיה לכל היותר משימה אחת. הבעייה היא שלא כל המכונות זהות, לכל מכונה יש קבוצת משימות אותן היא מסוגלת לבצע.

ניתן למדל את היחס בין מכונות למשימות באמצעות גרף G בו קבוצת הצמתים הוא האיחוד של המכונות והמשימות וקבוצת הקשתות מוגדרת כך שיש קשת בין מכונה למשימה אם ורק אם המכונה מסוגלת לבצע את המשימה. נשים לב שבגרף G אין קשתות בין משימות למשימות או בין מכונות למכונות, הגבלה זו משייכת את G למחלקת *הגרפים הדו-צדדיים*.

גרף דו-צדדי: גרף בו כל צמתי הגרף מחולקים לשתי קבוצות *זרות*, כך שאין קשת בגרף המחברת שני צמתים מאותה הקבוצה.

השמה חוקית של מכונות למשימות בבעיית ההשמה שקולה *לשידוך* בגרף G.

שידוך: אוסף של קשתות שלא מכילות אף צומת משותף. השידוך M נקרא מושלם אם כל קודקודי הגרף משתתפים בשידוך.

באופן טבעי נבקש לשדך כמה שיותר משימות למכונות, כלומר אנו מחפשים את השידוך המקסימלי בגרף.

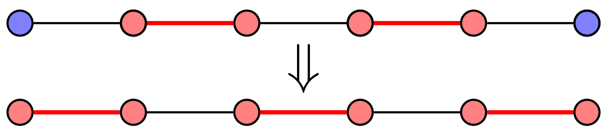
שידוך מקסימלי: שידוך M הוא מקסימלי אם ורק אם, לא קיים בגרף שידוך N כך ש $|M| < |N|$.

בבעיית ההשמה לעיתים נתונה לנו גם פונקציית משקל על הקשתות, אם מכונה מסויימת מבצעת משימה מסויימת בצורה יותר יעילה ניתן לקשת בינהן משקל גדול המנבא את הרווח מביצוע המשימה. כעת נגדיר את בעיית השידוך:

בעיית השידוד
קלט: גרף ממושקל G ומספר ממשי k.
שאלה: האם קיים בG שידוך בגודל k או יותר?

בתרגול זה נתמקד באלגוריתם של הרולד קון (הידוע גם כאלגוריתם ההונגרי) שמוצא שידוך מקסימלי בגרף דו-צדדי בזמן $\mathcal{O}(nm+n^2)$, בהנתן גרף דו-צדדי ממושקל זמן הריצה יהיה $\mathcal{O}(n^2m)$.
האלגוריתם ההונגרי מבוסס שני משפטים של המתמטיקאים ההונגרים דנס קוניג ויאנו איגרוואריי. באמצעותן הצליח קון להוכיח את האופטימליות של האלגוריתם שלו. למזלנו קיימת הוכחה פשוטה הרבה יותר לנכונות האלגוריתם, אבחנה שהוכחה כמה שנים מאוחר יותר על ידי המתמטיקאי הצרפתי קלוד ברז'. כדי להבין אותה נכיר ראשית את המושג *מסלול שיפור* ביחס לשידוך.

מסלול שיפור ביחס ל- M : מסלול פשוט בעל מספר קשתות אי-זוגי בין שני צמתים לא משודכים ב- M כך שכל קשת במיקום זוגי במסלול שייכת ל- M .



איור 17: למעלה: מסלול שיפור P ביחס לשידוך M, הקשתות של M באדום.
למטה באדום: ההפרש הסימטרי $M\Delta P$ בין מסלול השיפור לשידוך.

באמצעות ההגדרה של שידוך מקסימלי וההגדרה של מסלול שיפור ברז' הוכיח את הלמה הבאה:

הלמה של ברז'[יד]: M הוא שידוך מקסימלי אם ורק אם, בגרף אין אף מסלול שיפור ביחס ל- M.

באמצעות הלמה של ברז' ניתן בקלות לראות כיצד האלגוריתם מוצא את השידוך המקסימלי בגרף לא ממושקל. ברז' הוכיח אותה ב1957, כשנתיים לאחר פרסומו של האלגוריתם ההונגרי. באמצעותה הצליח ג'אק אדמונדס להציג ב1961 אלגוריתם בזמן $\mathcal{O}(mn^2)$ לפתירת בעיית השידוך על גרפים *כלליים*. האלגוריתם של אדמונדס מכונה גם אלגוריתם הפריחה (Blossom Algorithm) ונחשב עד היום לאחד האלגוריתמים החשובים בתולדות האלגוריתמים הקומבינטוריים[טו].

כעת נתבונן בפסואדו קוד של האלגוריתם של קון על גרף לא ממושקל.

[יד] ברז' קורא למסלול שיפור alternating-chain https://www.pnas.org/doi/pdf/10.1073/pnas.43.9.842
[טו] הנפשה של אלגוריתם הפריחה https://www.youtube.com/watch?v=3roPs1Bvg1Q

```
BipartiteMatching(G = (A ∪ B, E))
M = ∅
while True do
    Construct H(M)
    Compute path P from s to t.
    if P = ∅ then
        return M
    Remove from P the first and last edges.
    M = MΔP
```

האלגוריתם מתחיל עם שידוך ריק $M=\{\}$, בונה גרף $H(M)$ ובכל פעם מגדיל אותו לכל הפחות בקשת אחת ע"י מציאת מסלול P בין שני צמתים לא משודכים ב- $H(M)$ ולקיחת ההפרש הסימטרי[טז] $M\Delta P$. הבנייה $H(M)$ היא אלגוריתם המקבל שידוך M וגרף דו צדדי G ובונה מהם גרף מכוון $H(M)$ כך שאם הגרף קשיר אמ"ם קיים מסלול שיפור ב G ביחס ל M. נוסיף לגרף G שני צמתים נוספים s ו t, נוסיף כמוכן קשתות בין s לצמתים של צד אחד ובין t לצד השני שלא מופיעים ב-$V(M)$ (הצמתים בשידוך[יז]). נתון פסודו-קוד של הבנייה $H(M)$:

```
H(G, M)
Create empty graph H = (V_H, E_H).
Add s and t to V_H.
Add all vertices of G to V_H.
Add (s, v) for v ∈ A \ V(M)
Add (v, t) for v ∈ B \ V(M)
for {v, u} ∈ E do
    if {v, u} ∈ M then
        Add (u, v) to E_H.
    else
        Add (v, u) to E_H.
return H
```

ניתן לראות שהבנייה של הגרף $H(G,M)$ לוקחת זמן לינארי בגודל הגרף G. מאחר והשידוך המקסימלי בגרף חסום ב $\frac{n}{2}$ ומשום שחיפוש מסלול כלשהו בגרף לוקח גם כן זמן לינארי בגודל הגרף נקבל שהאלגוריתם ימצא שידוך מקסימלי בזמן $\mathcal{O}(n^2+mn)$.
בפיתוח האלגוריתם התבסס הרולד קון על מחקרם של המתמטקאים ההונגרים, דנס קוניג ויאנו איגרווארי, שהתאבדו בנסיבות טרגיות. לימים התברר שהבעיה נפתרה כמאה שנים מוקדם יותר ע"י המתמטקאי הגרמני קרל גוסטב יעקב יעקובי[יח].

[טז] $A\Delta B = (A\cup B)\setminus(A\cap B)$

[יז] $V(M)=\bigcup_{\{v,u\}\in M}\{v,u\}$

[יח] בקישור הסיפור המלא של האלגוריתם כפי שהוא מסופר ע"י הרולד קון בעצמו https://www.math.utoronto.ca/~mccann/1855/KuhnEJOR12.pdf

תרגיל 1

באפליקציית הכרויות נסיונית חדשה מבצעים התאמה בין גברים לנשים. לצורך ניסוי האפליקציה בחרו 6 נשים ו–6 גברים ובדקו את ההתאמה הזוגית בינהם ע"י מספר קריטריונים שונים.
להלן התוצאות: (נשים =אותיות, גברים = מספרים)

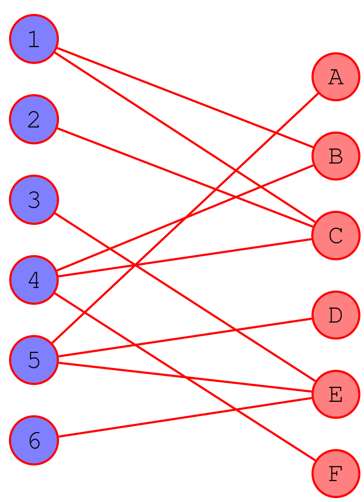
	1	2	3	4	5	6
A					✓	
B	✓			✓		
C	✓	✓		✓		
D					✓	
E			✓		✓	✓
F				✓		

(א) שרטטו את נתוני הטבלה בגרף.

(ב) מצאו שידוך מקסימלי.

(ג) הסבירו מדוע אין אפשרות להגיע לשידוך מושלם.

פתרון:



(א)

(ב) $M = \{A5, B1, C2, E6, F4\}$

(ג) שידוך מושלם משדך בין כל צמתי הגרף ובמקרה זה לא ניתן לשדך בו זמנית את A ואת D משום שתיהן מתאימות אך ורק ל־5, והרי קשתות של שידוך חייבות להיות זרות בצמתים.

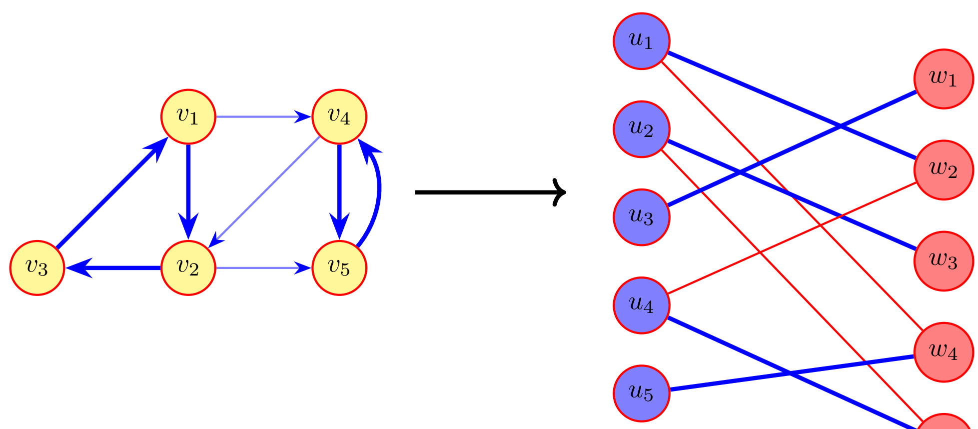
תרגיל 2

נתון גרף (לא בהכרח דו־צדדי) מכוון $G = (V, E)$ עם n צמתים. תארו אלגוריתם המוצא קבוצת קשתות $E' \subseteq E$ כך שלכל צומת i קיימת קשת אחת ב־ E' מסוג (j, i) (קשת נכנסת ל־ i) וקשת אחת מסוג (i, j) (קשת יוצאת מ־ i). אם אין קבוצת קשתות E' כנ"ל על האלגוריתם לדווח זאת.

פתרון:

- נבנה גרף דו צדדי $G^* = (U \cup W, E^*)$ כך ש $V = U = W$.
- נסמן $V = \{v_1, \ldots, v_n\}$, $U = \{v_1, \ldots, v_n\}$ ו $W = \{v_1, \ldots, v_n\}$.
- עבור כל קשת (v_i, v_j) ניצור קשת (u_i, w_j).
- נבדוק באמצעות BipartiteMatching אם בגרף יש שידוך מקסימלי בגודל $\frac{n}{2}$ (שידוך מושלם).
- אם מצאנו שידוך כנ"ל, נחזיר את קבוצת הקשתות המתאימות לקשתות השידוך ב־G^*.
- אחרת נודיע שלא קיים קבוצת קשתות כזאת.

סיבוכיות האלגוריתם $\mathcal{O}(mn + n^2)$.



הוכחת נכונות:

- נשים לב שכל קשת היוצאת מ v_i בG מתאימה לקשת היוצאת מ u_i ב G^*.
- על כן, זוג קשתות ב G, אחת הנכנסת ל v_i ואחת היוצאת מ v_i מתאימות לזוג קשתות זרות בG^*.
- לכן קבוצת קשתות E' כמתבקש בשאלה מתאימה לשידוך מושלם ב G^*.

תרגיל 3

נתונים שני שידוכים M ו-N בגרף $G=(V,E)$.

(א) אילו רכיבי קשירות אפשריים קיימים בגרף $G'=(V,M\cup N)$? הוכיחו.

(ב) נניח כי $|M|>|N|$, הוכיחו כי קיים ב G' מסלול שיפור ביחס ל M.

פתרון:

(א) הדרגה המקסימלית בגרף G' היא 2, זאת משום שכל צומת ב-G' משודך לכל היותר גם ב M וגם ב N.

רכיבי הקשירות האפשריים הם צמתים בודדים, מסלולים פשוטים ומעגלים באורך זוגי. אם לצומת מסויים דרגה 0, אז הוא רכיב קשירות בפני עצמו. כעת נראה כי הרכיבים האפשריים הם מסלול פשוט ומעגל.

יהי $P=[\{v_1,v_2\},\{v_2,v_3\},\dots\{v_{k-1},v_k\}]$ מסלול מקסימלי בגרף G', כלומר לא קיים מסלול P' המכיל את P. מאחר והדרגה המקסימלית של כל צומת היא 2, נסיק שלצומת v_i אין שכנים לבד מ v_{i+1} ו v_{i-1} עבור כל $1<i<k$. אנו יודעים שצמתים v_1 ו v_k שכנים של v_2 ו v_{k-1} בהתאמה, לכן לשניהם יכול להיות עוד שכן אחד שאינו אחד מהצמתים האמצעיים. משום ש P מקסימלי v_1 אינו שכן של אף צומת מחוץ למסלול, כלומר השכן הנוסף שלו יכול להיות רק v_k. כלומר הצמתים של P מהווים רכיב קשירות שיכול להיות או מסלול פשוט או מעגל במידה והקשת $\{v_1,v_k\}$ קיימת בגרף.

כעת נשאר להראות שאם k אי-זוגי אזי הצמתים של P אינם מעגל. נגיד, בלי אובדן הכלליות, שצומת v_1 משודך לv_2 בM. כלומר צומת v_2 חייב להיות משודך ל v_3 ב N, אחרת נקבל סתירה להגדרתם כשידוכים. מכאן ש $\{v_i,v_{i+1}\}$ משודך ב M אם ורק אם i אי-זוגי. כלומר אם k זוגי הוא משודך ב N ל v_{k-1}. נניח שקיימת קשת $\{v_1,v_k\}$ בגרף אז k חייב להיות זוגי אחרת צומת אחת תופיע פעמיים באחד השידוכים.

(ב) נתבונן ברכיבי הקשירות של G', בכל רכיבי הקשירות מסוג צומת בודד מספר הקשתות של M שווה למספר הקשתות של N משום שלשניהם 0 קשתות ברכיב. גם ברכיבי קשירות מסוג מעגל מספר הקשתות לכל שידוך שווה, משום שמעגל ב G' חייב להיות באורך זוגי.

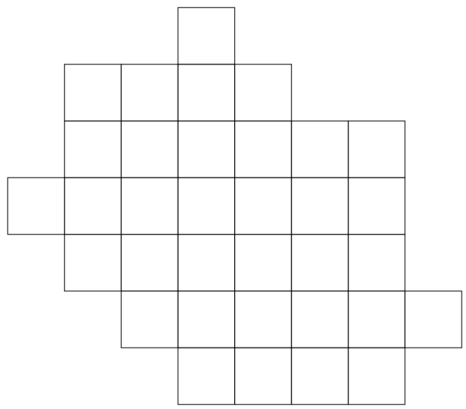
משום ש $|N|>|M|$ חייב להיות קיים לפחות רכיב קשירות אחד ב G' בו מספר הקשתות של N גדול ממספר הקשתות של M. ראינו שרכיב קשירות זה אינו צומת בודד או מעגל באורך זוגי, מכאן שהוא חייב להיות מסלול פשוט. ראינו שלא יכולות להיות שתי קשתות עוקבות באותו שידוך, לכן כל הקשתות במיקום הזוגי חייבות להיות שייכות לשידוך אחד ושאר הקשתות לשידוך השני. מספר הקשתות במסלול חייב להיות אי-זוגי אחרת ברכיב קשירות זה מספר הקשתות שווה.

תרגיל 4

נגדיר *שטח משובץ* כקבוצה רצופה של משבצות, כל משבצת חולקת צלע עם לפחות משבצת אחת. נגדיר שלשטח משובץ יש *ריצוף דומינו* אם ורק אם ניתן לרצף אותו במלבנים בגודל שתי משבצות כך שכל השטח מכוסה ואין שני מלבנים חופפים.

(א) תנו אלגוריתם המוצא ריצוף דומינו בהינתן שטח משובץ.

(ב) נתון השטח המשובץ הבא:

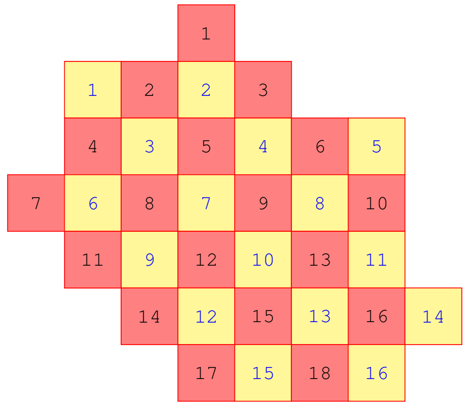


מצאו עבורו ריצוף דומינו או נמקו אם ריצוף כזה לא קיים.

פתרון:

(א) נהפוך את השטח המשובץ לגרף דו צדדי באופן הבא: נצבע את המרצפות כמו לוח שחמט. ניצור עבור כל משבצת צומת ונחבר בקשת כל זוג משבצות סמוכות. נחלק את קבוצות הצמתים לשתי קבוצות, צמתים אדומים וצמתים צהובים. קל לראות שאין זוג צמתים שכנים עם אותו צבע, לכן הגרף דו צדדי. נשתמש באלגוריתם למציאת שידוך בגרף דו-צדדי בזמן $\mathcal{O}(n^2+nm)$. אם מצאנו שידוך מושלם ניתן לרצף את השטח באמצעות דומינו.

(ב) נתבונן בצביעה מהסעיף הקודם. נשים לב כי כל אבן דומינו חייבת לכסות משבצת אחת אדומה ומשבצת אחת צהובה.



מספירה של המשבצות עולה כי הקבוצות אינן זהות במספר ולכן לא ייתכן שקיים ריצוף דומינו לשטח המשובץ.

10 זרימה מקסימלית

רשת זרימה היא גרף מכוון וממושקל (עם פונקציית משקלים c) בעל שני צמתים מיוחדים אותם אנו מכנים מקור s ויעד t.

זרימה זרימה ברשת זרימה בעלת n צמתים היא מיפוי $f : V \times V \to \mathbb{N}$ המקיים שני סוגי אילוצים:

1. אילוצי קיבולת – הזרימה בכל קשת אינה חורגת מהקיבולת שלה:

$$\forall i,j \in V \quad : \quad f_{i,j} \leq c_{i,j}$$

2. אילוצי שימור זרימה – עבור כל צומת למעט המקור והיעד, סך הזרימה הנכנסת שווה לסך הזרימה היוצאת:

$$\forall i \in V : \quad \sum_{j}^{n} f_{i,j} = \sum_{j}^{n} f_{j,i}$$

נגדיר את **ערך הזרימה** $|f|$ כסך הזרימה היוצאת מהמקור (באופן שקול סך הזרימה הנכנס ליעד):

$$|f| = \sum_{j}^{n} f_{s,j} = \sum_{j}^{n} f_{j,t}$$

נגיד שזרימה f היא **מקסימלית** אם לא קיימת זרימה f^* כך ש $|f^*| > |f|$. כעת אנו מכירים את כל המושגים הנדרשים להבנת בעיית הזרימה:

בעיית הזרימה
קלט: רשת זרימה G ומספר שלם וחיובי F.
שאלה: האם קיימת עבור G זרימה חוקית f כך ש $|f| \geq F$?

בתרגול זה נתמקד באלגוריתם של פורד-פולקרסון [יט] המוצא זרימה מקסימלית ברשת זרימה בזמן $\mathcal{O}(|f|(m+n))$ כש f היא הזרימה המקסימלית של הרשת. נעבור על מהלך האלגוריתם, נשתמש בו כדי לפתור את בעית השידוך בגרף דו צדדי וכדי למצוא את החתך המינימלי.

```
MaximumFlow(G = (V,E), c : E → ℕ)
Initialize f to have zero flow on all edges of G.
while True do
   Construct G_f.
   Compute path P in G_f from source to target.
   if P = ∅ then
      return f
   Update f according to P.
```

[יט] פורד זה אותו לסטר פורד מבלמן-פורד שפיתח את האלגוריתם ביחד עם עמיתו דלברט פולקרסון: https://www.cambridge.org/core/services/aop-cambridge-core/content/view/5D6E55D3B06C4F7B1043BC1D82D40764/S0008414X00036890a.pdf/maximal-flow-through-a-network.pdf .

בהינתן רשת זרימה $G=(V,E)$ וזרימה f נסמן ב $G_f=(V_f,E_f)$ את **הרשת השיורית** (Residual Network) של G בהנתן f. כאשר הרשת השיורית מכילה את כל הצמתים של רשת הזרימה $V_f=V$, כמו כן היא מכילה עבור כל קשת $(i,j)\in E$ את שתי הקשתות $(i,j),(j,i)$, פורמלית $E_f=\{(i,j),(j,i)|(i,j)\in E\}$. משקל הקשתות, או *הקיבולת* שלהן, ברשת השיורית יחושב ע"פ $c_{i,j}-f_{i,j}+f_{j,i}$, במידה והקיבולת של קשת היא 0 לא נכלול אותה ברשת השיורית.

האלגוריתם של פורד-פולקרסון מתחיל עם f זרימת האפס באיטרציה הראשונה ואז מגדיל אותה בכל פעם ע"י מציאת מסלול ברשת השיורית והגדלת הזרימה f בהתאם לצוואר הבקבוק של המסלול (הקשת בעל הקיבולת הקטנה במסלול). לבסוף כשברשת השיורית של f אין מסלול עם צוואר בקבוק גדול מ0, האלגוריתם מחזיר את f.

אבל למה שבסוף התהליך הזה f תהיה הזרימה המקסימלית? קל לראות שאם היה קיים מסלול P ברשת השיורית אז הייתה קיימת $|f^*|=|f|+\texttt{bottle\_neck}(P)$ ואז f לא הייתה מקסימלית. אבל הטיעון הזה לא מסביר את הכיוון ההפוך - למה *העדרו* של מסלול מעיד על כך שהזרימה בידינו מקסימלית? זהו הכיוון החשוב שהבטיח את האופטימליות של פורד-פולקרסון על פני "טכניקת ההצפה"[c]. כדי להבין למה הטכניקה של פורד-פולקרסון מבטיחה לנו את מציאת הזרימה המקסימלית נכיר את המושג *חתך*:

חתך ברשת זרימה הוא חלוקה של הצמתים לשתי קבוצות זרות S ו T כך שהמקור נמצא באחת והיעד בשנייה.

$$S\cup T=V,\qquad S\cap T=\varnothing,\qquad s\in S,\qquad t\in T$$

נסמן את קבוצות הקשתות החוצות את החתך ב $E(S,T)$.

$$E(S,T)=\{(i,j)\in E|i\in S,j\in T\}$$

את **ערך החתך** נסמן ב $C(S,T)$ והוא מוגדר להיות סך הקיבולות של הקשתות החוצות:

$$C(S,T)=\sum_{(i,j)\in E(S,T)}c(i,j)$$

הנכונות של האלגוריתם פורד-פולקרסון נובעת מהנכונות של שני המשפטים הבאים:

משפט 1. *f היא זרימה מקסימלית אם ורק אם ברשת השיורית של f אין אף מסלול מהמקור ליעד עם צוואר בקבוק גדול מ0.*

משפט 2. *בכל גרף מתקיים שערך הזרימה המקסימלית שווה לערך החתך המינימלי.*

וודאו שאתם מבינים את ההוכחה שלהם, הם המפתח כדי להבין למה הטכניקה של פורד ופולקרסון עדיפה על טכניקת ההצפה בה השתמשו ראשית.

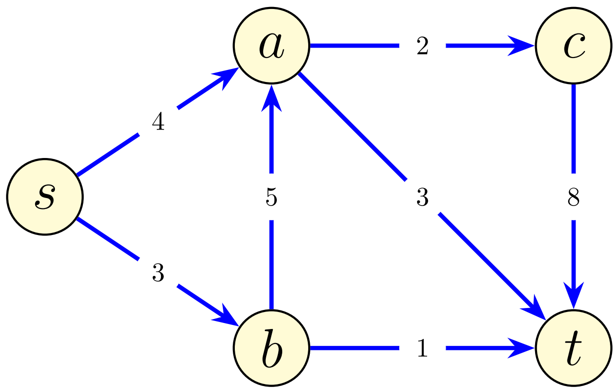
נקודה לסיום, האלגוריתם של פורד-ופולקרסון פורסם ב1956, שימו לב כי זמן הריצה שלו תלוי בערך הזרימה המקסימלית של הרשת, ערך זה אינו חסום בגודל הרשת.

[c] טכניקת ההצפה הייתה הטכניקה הראשונית באמצעותה ניסו לפתור את בעיית הזרימה המקסימלית, במהלכה "הוצפה" רשת הזרימה עד אשר הושגה זרימה מהמקור ליעד. למתעניינים פירוט על טכניקת ההצפה בנוסף של הגדרת בעיית הזרימה המקסימלית שעלתה בנסיון הערכת היכולת של רשת הרכבות הסובייטית בזמן המלחמה הקרה. https://ntlrepository.blob.core.windows.net/lib/13000/13200/13238/AD093458.pdf

אדמונדס וקארפ פרסמו ב 1972 שיפור לאלגוריתם בו הצליחו להפטר מ $|f|$ בזמן הריצה ע"י מציאת המסלול הקצר בקשתות בכל איטרציה. בכך הראו שניתן למצוא את הזרימה המקסימלית בגרף בזמן $\mathcal{O}(nm^2)$[כא].

תרגיל 1

נתון הגרף הבא:



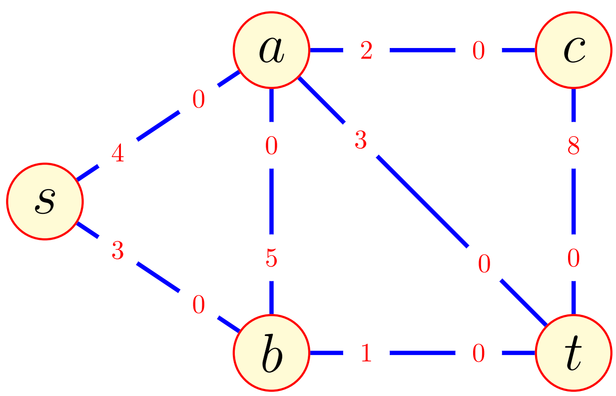
(א) בנו את הרשת השיורית בהינתן זרימה $f_{i,j} = 0$ לכל $i, j \in V$.

(ב) מצאו את הזרימה המקסימלית באמצעות האלגוריתם של פורד–פולקרסון.

(ג) מצאו את החתך המינימלי.

פתרון:

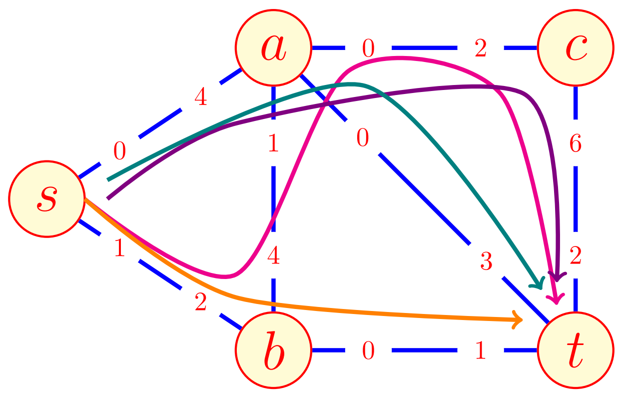
(א) הרשת השיורית המתקבלת עבור זרימה 0.



[כא] המאמר של אדמונדס וקארפ https://link.springer.com/content/pdf/10.1007/3-540-36478-1_4.pdf, דוברי הרוסית מבינכם יוכלו להשוות עם המאמר של דיניץ מ1970 ולאשר אם הוא אכן הקדים אותם לשיפור https://www.mathnet.ru/links/9dae42f9afaaf275c6e2baf02c6936c7/dan35701.pdf.

(ב) הזרימה המקסימלית שהתקבלה היא 6. את הזרימה המקסימלית ניתן לפרק למסלולים:

- $f_1 = 1$
- $f_2 = 1$
- $f_3 = 3$
- $f_4 = 1$



(ג) החתך המינימלי הוא $\{(a,c),(a,t),(b,t)\}$, בתרגיל הבא נראה איך מוצאים אותו.

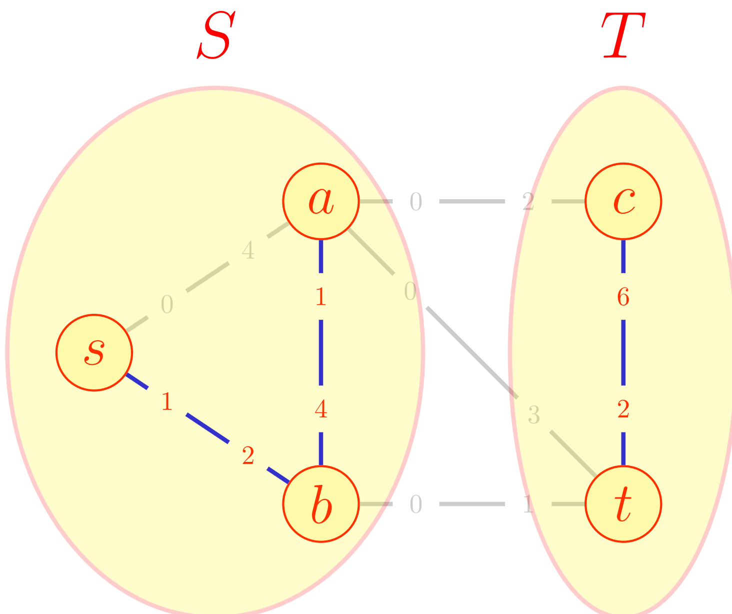
תרגיל 2

נתונה רשת זרימה G וזרימה מקסימלית f בעבורה. תארו אלגוריתם למציאת חתך מינימלי של הרשת, נתחו את זמן הריצה.

פתרון:

זכרו שחתך מינימלי מחלק את הגרף לשתי קבוצות של צמתים, כאלו שאפשר להגיע אליהם מ s וכאלה שלא.
ברשת השיורית של הזרימה המקסימלית, קשתות עם קיבולת 0 הן קשתות אשר ניצלו את כל הקיבולת שלהם ולכן לא ניתן להעביר בהן עוד זרימה. לכן משום שהזרימה f שמצאנו היא מקסימלית, לא ניתן יהיה להזרים $a > 0$ מהמקור ליעד ברשת השיורית של f שכן אז תהיה קיימת $f^* = f + a$ גדולה מ f וזה בא בסתירה לכך שf מקסימלית.[כב]

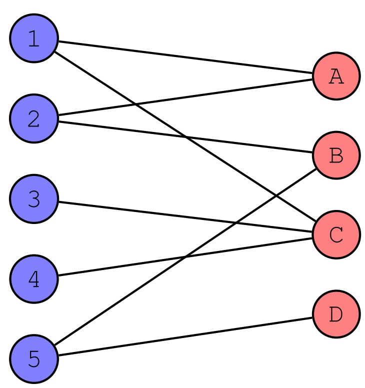
[כב] שימו לב! זו לא הוכחה לכך שf היא מקסימלית. ההוכחה לכך בהרצאה.



1. נבנה את הרשת השיורית המתקבלת בהינתן הזרימה המקסימלית f. זה ייקח לנו $\mathcal{O}(n+m)$ זמן משום שמדובר במעבר על הגרף והוספת m קשתות לכל היותר.
2. כעת נפעיל את אלגוריתם BFS לבדיקת קשירות ב$\mathcal{O}(n+m)$ זמן (גם DFS יעבוד) מצומת המקור ברשת השיורית G_f. נוסיף לקבוצה S את כל הצמתים שמצא אלגוריתם החיפוש. שאר הקודקודים יהיו בקבוצה T.
3. החלוקה S וT היא החתך המינימלי.

תרגיל 3

נתון גרף דו צדדי:



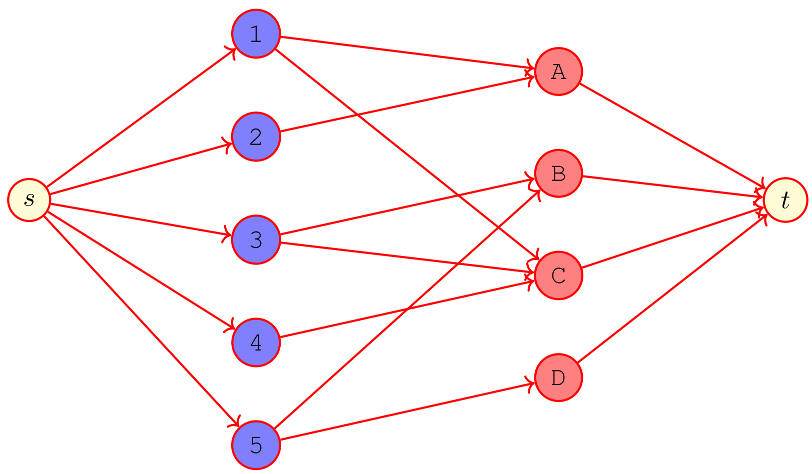
תארו אלגוריתם למציאת שידוך מקסימלי המשתמש באלגוריתם של פורד-פולקרסון.

פתרון:

נהפוך את הגרף $G=(U\cup W,E)$ לרשת זרימה:

1. נכוון כל קשת בגרף מקבוצה U לקבוצה W.
2. נוסיף צומת מקור s ונחבר אותה לכל הצמתים של U.
3. נוסיף צומת יעד t ונחבר אותה לכל הצמתים של W.
4. ניתן לכל הקשתות משקל 1.

נחשב את הזרימה המקסימלית באמצעות האלגוריתם של פורד פולקרסון. הזרימה המקסימלית ברשת הזרימה מתאימה לשידוך מינימלי שכן לכל צומת המחוברת למקור מגיעה יחידת זרימה אחת וצוואר הבקבוק של כל צומת המחוברת ליעד גם הוא בגודל יחידה אחת. כלומר הזרימה המקסימלית בוחרת את המספר המקסימלי של קשתות זרות בין שתי הקבוצות.
זמן הריצה $\mathcal{O}(f(n+m))=\mathcal{O}(n(n+m))$ משום שהשידוך המקסימלי חסום ב$\mathcal{O}(n)$.



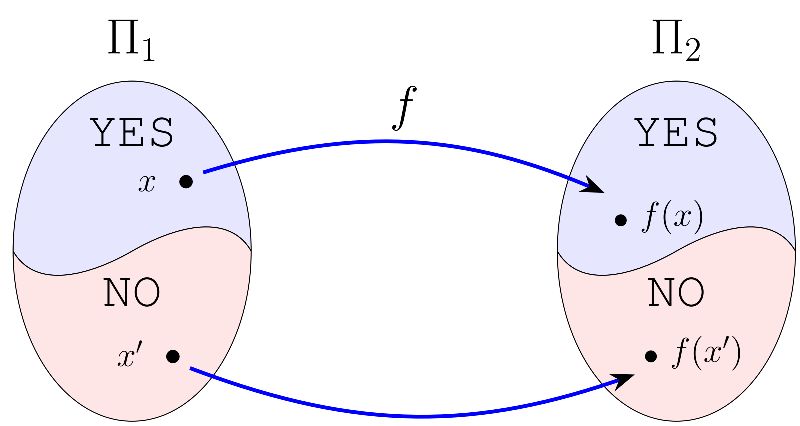
טיפ של אלופים - הראנו שניתן למצוא שידוך מקסימלי בגרף דו צדדי באמצעות האלגוריתם של פורד-פולקרסון שמוצא זרימה מקסימלית ברשת זרימה. בעצם עשינו **רדוקציה** מבעיית השידוך המקסימלי בגרף דו צדדי לבעיית הזרימה המקסימלית. רדוקציה היא מיפוי של בעייה א' לבעייה ב' כך שפתרון לב' יכול להיות "מתורגם" לפתרון של בעיה א'. במקרה שלנו זרימה מקסימלית מתורגמת לשידוך מקסימלי. נראה הגדרה פורמלית לרדוקציה ביחידה הבאה.

רדוקציה היא טכניקה נפוצה בעזרתה ניתן למצוא לבעייה אלגוריתם יעיל ע"י שימוש באלגוריתם מוכר שפותר בעייה אחרת. עוד על רדוקציות בחלקו האחרון של הקורס.

11 רדוקציות

עד חלק זה של הקורס ראינו בעיות שניתן לפתור בזמן ריצה פולינומי[כג] , או לפחות – בעיות שניתן *להכריע* בזמן פולינומי. כל בעיות ההכרעה שראינו עד כה שייכות למחלקה P, מחלקת הבעיות שניתנות להכרעה בזמן פולינומי. בעיית הכרעה היא שאלה של כן או לא, הנשאלת על סוג מסויים של קלט. במקרה של בעיית השידוך סוג הקלט הוא הצמד (G,k) כש G הוא גרף לא מכוון ו k הוא מספר טבעי. נגיד ש (G,k) הוא קלט "כן" אם קיים ב G שידוך בגודל k או יותר, אחרת נגיד שהוא "לא".

ראינו שניתן להכריע את בעיית השידוך באמצעות האלגוריתם של פורד–פולקרסון, למעשה עשינו *רדוקציה* מבעיית השידוך לבעיית הזרימה. **רדוקציה** היא אלגוריתם הרץ בזמן פולינומי הממיר קלט x של בעיה אחת, בעייה Π_1, לקלט של בעייה אחרת, בעיה Π_2. למעשה האלגוריתם מחשב את הפונקציה f כך ש x הוא קלט "כן" של בעייה Π_1 אם ורק אם $f(x)$ הוא קלט "כן" של בעייה Π_2. פורמלית נכתוב $x \in \Pi_1 \Leftrightarrow f(x) \in \Pi_2$ כשאנחנו מתכוונים ב $a \in \Pi$ ש a הוא קלט "כן" של Π. אם קיימת רדוקציה מ Π_1 לבעייה Π_2 נסמן זאת כך $\Pi_1 \leq_p \Pi_2$.



איור 18: הפונקציה f היא רדוקציה מבעיה Π_1 לבעייה Π_2, אם אפשר לחשב אותה בזמן פולינומי והיא ממפה קלטים של "כן" לקלטים של "כן" וקלטים של "לא" לקלטים של "לא".

בדוגמא של בעיית השידוך ראינו שניתן לבנות מהגרף רשת זרימה כך שקיימת זרימה בגודל k אם קיים בגרף G שידוך בגודל k. למעשה רדוקציה יכולה לעזור לנו לסווג בעיית הכרעה לא מוכרת Π_1 כבעיה ב P ע"י שימוש באלגוריתם A המכריע בעייה אחרת Π_2. בהינתן קלט x ורדוקציה $f : \Pi_1 \rightarrow \Pi_2$ נוכל לחשב $f(x) = y$ ואז להפעיל את אלגוריתם A כדי להכריע בזמן פולינומי אם y הוא קלט "כן" או לא. הרדוקציה מחושבת בזמן פולינומי $\mathcal{O}(|x|^c)$ עבור קבוע c, כמו כן $A(y)$ מחושב בזמן $\mathcal{O}(|y|^{c'})$ עבור קבוע c'. מאחר ו y הוא פלט שנכתב בזמן פולינומי, אנחנו יודעים ש $|y| \in \mathcal{O}(|x|^c)$. בעצם קיבלנו אלגוריתם פולינומי B המכריע את Π_1.

[כג] זמן פולינומי: $\mathcal{O}(n^c)$ עבור c קבוע.

```
B(x):
  Compute y = f(x).      ▷ O(|x|^c)
  Compute A(y).          ▷ O(|y|^{c'}) = O((|x|^c)^{c'}) = O(|x|^{c'·c})
  if A(y) is YES then
     return YES
  else return NO
```

יפה, אז אם פגשנו בעיה לא מוכרת נוכל לסווג אותה כבעייה ב P ע"י מציאת אלגוריתם פולינומי המכריע אותה או מציאת רדוקציה לבעייה אחרת ב P. אבל מה אם לא מצאנו אלגוריתם או רדוקציה כאלה? האם אין לנו מה לעשות חוץ מלהמשיך לחפש? למזלנו, כמו שניתן להשתמש ברדוקציה כדי להראות שלבעיות מסויימות יש אלגוריתמים פולינומים, נוכל להשתמש ברדוקציה כדי להראות שלבעיות אחרות "אין"[כד] אלגוריתם פולינומי. על השימוש הזה נלמד בחלקו האחרון של הקורס, לפני כן נצטרך להכיר עוד מחלקה של בעיות, אחת מהן היא בעיית הספיקות Satisfiability או בקיצור SAT.

בעיית הספיקות SAT

קלט: נוסחת CNF בוליאנית Φ.

שאלה: האם יש ל Φ השמה מספקת ?

נוסחת CNF (Conjunctive Normal Form) היא פסוק המורכב מאוסף פסוקיות המחוברות בינהן בקוניוקצה (האופרטור $\wedge$ "וגם") $\Phi = C_1 \wedge C_2 \wedge ... \wedge C_m$. כל פסוקית $C = (\alpha_1 \vee \alpha_2 \vee ... \vee \alpha_k)$ היא נוסחא בוליאנית בה המשתנים מחוברים בדיסיונקציה (האופרטור "או" $\vee$) ויכולים להופיע בשלילה. כדי לספק פסוקית די לספק את אחד המשתנים בה. השמה לנוסחא בוליאנית היא מיפוי של כל אחד מהמשתנים לערך TRUE או FALSE . נגיש שהשמה β היא השמה מספקת אם Φ הוא פסוק אמת כשנציב בכל משתנה את ערכו ע"פ β.

הגיע הזמן להכיר את מחלקת הבעיות NP – אנו מגדירים אותה בתור מחלקת הבעיות הניתנות לצמצום לSAT, כלומר קיימת רדוקציה פולינומית מהן ל SAT. ניתן להגיד ש NP היא מחלקת הבעיות הניתנות להכרעה בזמן פולינומי בהנתן עד y, משום שאם ניתן להמיר לSAT ניתן להכריע אותן בזמן פולינומי בהנתן ההשמה המספקת בתור עד. אינטואיטיבית אפשר להגיד שבעייה ב NP ניתנת *לבדיקה* בזמן פולינומי בהינתן פתרון y. כלומר, אם Π היא בעייה ב NP, אז קיים אלגוריתם פולינומי A כך ש $A(x, y) = YES$ אם ורק אם $x \in \Pi$. כבר מההגדרה זו ניתן לראות שכל בעיה ב P היא גם בעיה ב NP. חשבו, מה יכול להחשב בתור עד לבעייה ב P?

ייתכן שישנן בעיות ב NP שאינן מוכלות ב P. כלומר, ניתן לבדוק עבורן פתרונות ביעילות (בזמן פולינומי) אך לא ניתן *להכריע* אותן בזמן פולינומי.

בעיית הספיקות היא בעיה ב NP(באופן טבעי כשהרדוקציה לSAT היא פונקציית הזהות), גם לפי האינטואציה משום שבהינתן השמה מספקת נוכל לבדוק אותה בזמן פולינומי (אתם תוכיחו את זה במטלת הבית), כמו כן ניתן להכריע כל בעיה ב NP בזמן $\mathcal{O}(2^{\text{poly}(n)})$, מצליחים לראות למה? אל דאגה, נראה לכך דוגמא בתרגול וגם תוכיחו את זה במטלת הבית.

[כד] או שיש, אבל אז נגלה שאנחנו חיים בעולם מוזר מעבר לכל דמיון.

נשאלת השאלה הטבעית, האם ניתן להכריע את בעיית הספיקות בזמן פולינומי ללא עד? האם ניתן להכריע כל בעיה ב NP בזמן פולינומי? האם P=NP?
ביחידה האחרונה של הקורס נלמד על קבוצה מיוחדת של בעיות המוכלות ב NP, בעיות NP-שלמות. עד היום לא הצלחנו למצוא עבור בעיות אלה אלגוריתמים פולינומים, מדענים רבים מאמינים שלא קיימים אלגוריתמים פולינומים עבורן, אבל מדוע? חסרי הסבלנות מבינכם יכולים להנות כבר עכשיו מההסבר ב סרטון[כה] על שאלת P vs. NP.

תרגיל 1

נתונה נוסחת ה-SAT הבאה:

$$\Phi = (x \vee y \vee \neg z) \wedge (\neg x \vee z) \wedge (\neg y \vee z) \wedge (\neg x \vee \neg y \vee z) \wedge (x \vee \neg y \vee z)$$

כמה השמות שונות מספקות את הנוסחה?

פתרון:

לנוסחה עם n משתנים יש 2^n השמות אפשריות. במקרה שלנו $2^3 = 8$, מתוכן ארבע השמות מספקות:

x	y	z	Φ
1	1	1	T
1	1	0	F
1	0	1	T
1	0	0	F
0	1	1	T
0	1	0	F
0	0	1	F
0	0	0	T

תרגיל 2

בעיית הקליקה

קלט: גרף לא מכוון G ומספר טבעי k.

שאלה: האם יש ב G תת גרף מלא בגודל k צמתים?

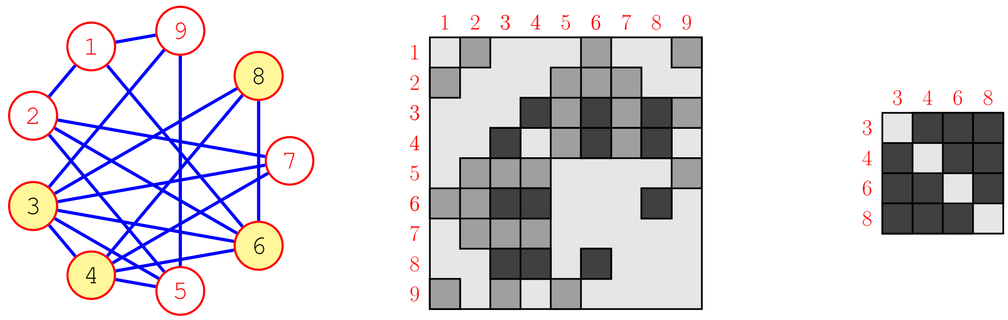
תארו אלגוריתם נאיבי להכרעת בעיית הקליקה. נתון גרף $G = (V, E)$ הממומש במטריצת שכנויות. מה תהיה סיבוכיות האלגוריתם במונחי $|E|$, $|V|$ ו-k? מה התנאי עבורו האלגוריתם ירוץ בזמן פולינומי?

פתרון:

באופן נאיבי ניתן לעבור על כל תתי הקבוצות בנות k צמתים ולבדוק אם אחת מהן היא קליקה. יש $\binom{n}{k}$ קבוצות כאלה משום שאנחנו בוחרים k צמתים מתוך קבוצה של n צמתים. המקדם הבינומי $\binom{n}{k}$ חסום ב $\mathcal{O}(n^k)$, כלומר קיימות $\mathcal{O}(n^k)$ אפשרויות עבור $V' \subseteq V$ כך שמתקיים $|V'| = k$.

[כה] https://www.youtube.com/watch?v=YX40hbAHx3s

כעת נותר לנו לראות בכמה זמן נבדוק אם קבוצת צמתים V' היא קליקה. בהינתן V' נבדוק עבור כל צומת v אם הוא שכן של כל צומת $u \in V' \setminus \{v\}$. ניתן לבדוק זאת בזמן קבוע עבור כל צמד (u,v). יש $\mathcal{O}(k^2)$ צמדים כאלו ב $V' \times V'$.
בדקנו $\mathcal{O}(n^k)$ קבוצות בזמן $\mathcal{O}(k^2)$ לקבוצה לכן האלגוריתם שלנו ירוץ בזמן $\mathcal{O}(n^k k^2)$.



איור 19: משמאל – הגרף G, במרכז – מטריצת השכנויות של G, מימין – תת המטריצה המתאימה לקליקה המקסימלית $V' = \{3,4,6,8\}$.

ביחידה האחרונה של הקורס נלמד למה האלגוריתם הנאיבי הוא גם האלגוריתם המהיר ביותר (המוכר למדע) שמכריע את בעיית הקליקה, ולמה אנחנו מאמינים שלא קיים אלגוריתם מהיר יותר.

הסבר עבור $\binom{n}{k} \in \mathcal{O}(n^k)$:

$$\begin{aligned}\binom{n}{k} &= \frac{n!}{(n-k)!k!} \\ &= \frac{1 \times 2 \times \ldots \times n}{(1 \times 2 \times \ldots \times n-k)1 \times 2 \times \ldots \times k} \\ &= \frac{(n-k+1) \times \ldots \times n}{1 \times 2 \times \ldots \times k} \\ &\leq \frac{n^k}{1 \times 2 \times \ldots \times k} \leq n^k\end{aligned}$$

תרגיל 3

בעיית המסלול הארוך

קלט: גרף $G=(V,E)$, שני צמתים $s,t \in V$ ומספר טבעי k.
שאלה: האם יש ב G מסלול באורך לפחות k בין s ל t?

מצאו רדוקציה מגרסת ההכרעה של בעיית השידוך המקסימלי בגרפים כלליים (מצא שידוך לפחות בגודל k) לבעיית המסלול הארוך.

פתרון:

שימו לב, בעיית המסלול הארוך היא בעיה NP-קשה[כ] בעוד בעיית השידוך המקסימלי בגרף דו צדדי היא בעיה ב P. רדוקציה בכיוון הזה היא אפשרית ואפילו פשוטה

[כ] עוד על בעיות NP-קשות ביחידה האחרונה של הקורס

מאד! אנחנו רוצים להמיר בזמן פולינומי את הצמד (G,k) לרביעייה (G',s,t,k') כך שאם ב G שידוך גדול מ k יהיה ב G' מסלול באורך לפחות k'.

הרדוקציה תחשב את G' כדקלמן: ראשית תחשב את השידוך המקסימלי ב G, אנחנו יודעים שניתן לעשות זאת בזמן פולינומי באמצעות אלגוריתם הפריחה של אדמונדס. אם נמצא שידוך גדול שווה k היא תחזיר גרף G' בן שני צמתים (s,t) שכנים. אם השידוך קטן מ k בגרף G' לא תהיה קשת. נגדיר $k'=1$.

הרדוקציה מחושבת בזמן פולינומי, כמו כן, קל לראות ש (G,k) הוא מופע "כן" של בעיית השידוך אם ורק אם $(G',s,t,1)$ הוא מופע "כן" של בעיית המסלול הארוך.

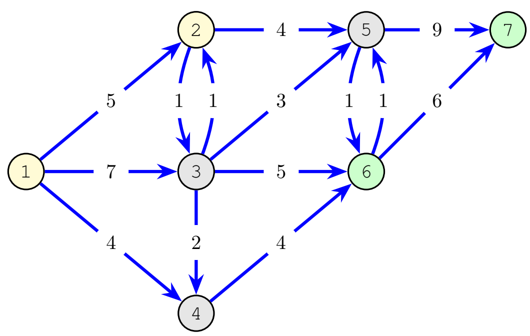
תרגיל 4

בעיית הזרימה מרובת המקורות והיעדים

קלט: רשת זרימה G עם מספר מקורות $s_1,s_2,...,s_\ell$ ומספר יעדים $t_1,t_2,...,t_{\ell'}$.

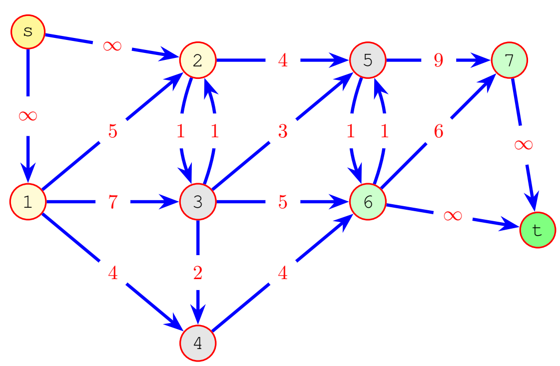
שאלה: האם קיימת זרימה חוקית f עבור G כך ש $|f|\geq k$?

הראו רדוקציה מבעיה זו לבעיית הזרימה. להלן דוגמא לרשת עם שני מקורות ושני יעדים:



פתרון:

נוסיף מקור-על ונחבר אותו לכל המקורות בקשתות עם קיבולת k, נוסיף כמו כן צומת יעד המחוברת לכל היעדים עם קשתות עם קיבלות k. זמן הבנייה $\mathcal{O}(\ell+\ell')$.

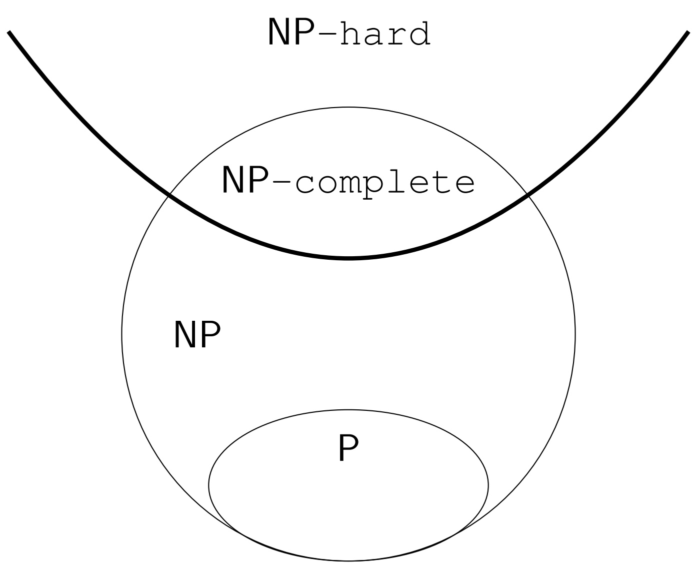


אם קיימת ב G זרימה f גדולה מ k קיימת זרימה f' כך ש $f'_{i,j}=f_{i,j}$ עבור כל i ו j וכך ש $f'_{v,j}=\sum_{j\in V}|f_{v,j}|$ לכל $v\in\{s_1,...,s_\ell,t_1,...,t_{\ell'}\}$. הכיוון השני, אם קיימת זרימה

f' גדולה מ k אז קיימת זרימה f גדולה מ k, גם כן פשוט. נוכל לקבל את f ע"י מחיקת הצמתים החדשים.

12 בעיות NP-קשות

הכל מתחיל בטרנזיטיביות של רדוקציות פולינומיות. בהינתן רדוקציה f מבעיה Π_1 לבעיה Π_2 ורדוקציה g מבעיה Π_2 לבעיה Π_3 נוכל להראות ש $h = g \circ f$ היא רדוקציה. אנחנו יודעים ש h ממפה קלט "כן" של Π_1 לקלט "כן" של Π_3 משום שידוע לנו ש $x \in \Pi_1 \Leftrightarrow f(x) \in \Pi_2$ וגם ש $x \in \Pi_2 \Leftrightarrow g(x) \in \Pi_3$ אז $x \in \Pi_1 \Leftrightarrow f(x) \in \Pi_2 \Leftrightarrow g(f(x)) \in \Pi_3$. חשוב גם להראות איך אנחנו יודעים ש h פולינומית. העזרו בהסבר על רדוקציות לבעיות ב P מהתרגול הקודם. ראינו שרדוקציה מ Π לבעיה ב P יכולה להראות לנו ש Π מוכלת P. חשבו, מה מראה לנו רדוקציה מבעיה ב NP לבעייה Π? נראה בתרגול זה!



איור 20: באיור היחסים בין המחלקות P ו NP. בעיה Π היא NP-קשה אם קיימת רדוקציה מ SAT ל Π. בעיה Π היא NP-שלמה אם היא NP-קשה והיא נמצאת ב NP.

סטפן-קוק הוכיח שאפשר לפתור את בעיית הספיקות (וגם כל בעיה אחרת ב NP) בזמן פולינומי באמצעות מודל חישובי הנקרא *מכונת טיורינג לא-דטרמיניסטית*[כז]. אפשר לחשוב על המודל הזה כעל אלגוריתם המסוגל "לנחש" השמה מספקת לבעיית הספיקות במידה ואחת כזאת קיימת. מכונות כאלו כמובן לא קיימות במציאות, אך ההוכחה של קוק חשובה משום שהיא סיווגה לראשונה את בעיית הספיקות כבעיה ה NP-שלמה הראשונה.

אנחנו אומרים שבעייה Π היא NP-קשה אם היא *קשה לפחות כמו SAT*, כלומר אם ניתן להכריע את Π בזמן T כלשהו, ניתן להכריע את SAT בזמן פולינומי ב T. אנחנו אומרים שבעייה Π היא NP-שלמה אם היא גם NP-קשה וגם נמצאת במחלקה NP (ניתנת להכרעה בזמן פולינומי בהינתן עד).

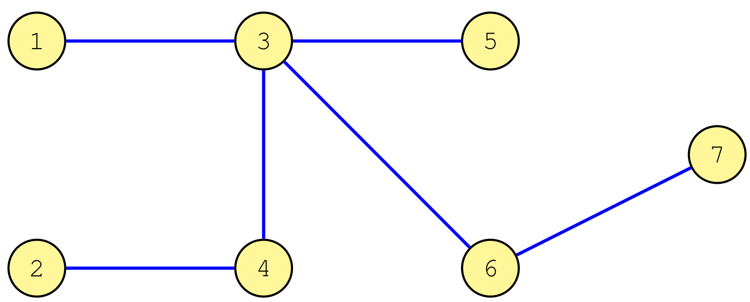
בזכות הטרנזיטיביות של הרדוקציות ניתן לסווג בעיה Π כבעיה NP-קשה אם מצאנו רדוקציה מבעיה NP-קשה אחרת ל Π. זוכרים את ריצ'ארד קארפ? זה בדיוק מה שהוא

[כז] לכן המחלקה נקראת NP - **N**on-deterministic **P**olynomial time.

עשה. להנאתכם ראיון אצל לקס פרידמן[כח] בו הוא מדבר על פרוייקט[כט] בו סיווג 21 בעיות NP–שלמות, בינהן בעיית הקליקה, כיסוי צמתים ועוד.
למעשה הוכחת קושי של בעיה Π מהווה *חסם תחתון* לזמן הריצה של אלגוריתם המכריע אותה. רדוקציה מ SAT ל Π מראה שאם היה לנו אלגוריתם פולינומי ל Π אז היה לנו אלגוריתם פולינומי ל SAT, מאחר ואנחנו מאמינים שאין אלגוריתם פולינומי ל SAT – אנחנו מאמינים שאין אלגוריתם פולינומי ל Π.

תרגיל 1

נגדיר שתי בעיות על גרפים לא מכוונים:



בעיית הקבוצה המרוחקת
קלט: גרף לא מכוון G ומספר טבעי k.
שאלה: האם יש ב G קבוצת צמתים בגודל k כך שהמרחק בין כל זוג צמתים בקבוצה גדול מ 2?

-

בעיית הקבוצה הבלתי תלויה
קלט: גרף לא מכוון G ומספר טבעי k.
שאלה: האם יש ב G קבוצת צמתים בגודל k כך שאין זוג צמתים שכנים בקבוצה?

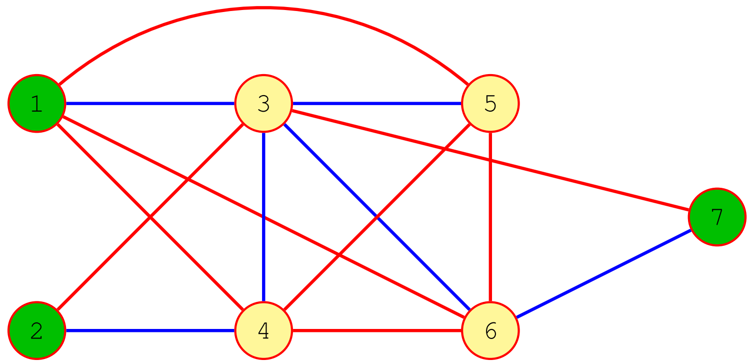
(א) בצעו רדוקציה מבעיית הקבוצה המרוחקת לבעיית הקבוצה הבלתי תלויה.

(ב) בצעו רדוקציה מבעיית הקבוצה הבלתי תלויה לבעיית הקבוצה המרוחקת.

פתרון:

(א) נבנה מהגרף $G=(V,E)$ גרף $G'=(V',E')$ כך ש $V'=V$, ו $E'=E\cup E^*$ כך ש $E^*=\{\{v,u\} \mid \text{distance}(v,u)\le 2\}$. למעשה נוסיף קשת בין כל שני צמתים אשר המרחק בינהם הוא עד 2. עבור כל צומת ניתן למצוא את כל השכנים במרחק 2 באמצעות שתי איטרציות של אלגוריתם BFS ולכן ניתן לחשב את $G'=(V',E')$ בזמן פולינומי.

[כח] https://www.youtube.com/watch?v=KllCrlfLuzs
[כט] https://cgi.di.uoa.gr/~sgk/teaching/grad/handouts/karp.pdf

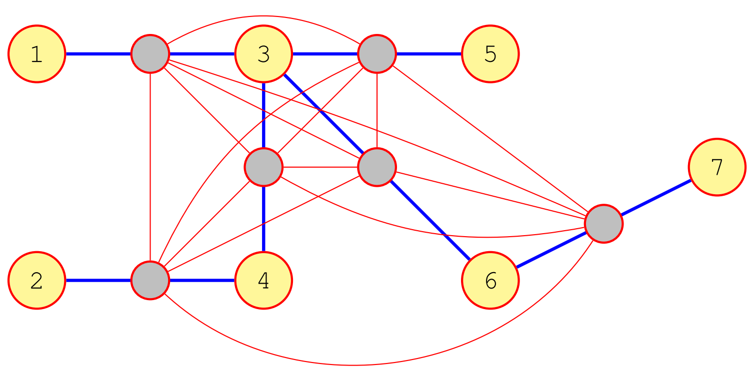


ב G יש קבוצה מרוחקת בגודל k אם ורק אם ב G' יש קבוצה בלתי תלויה בגודל k.

($\Leftarrow$) תהי V' קבוצה מרוחקת בגודל k, כלומר אין מסלול קצר מ2 בין כל שני $v,u \in V'$ ולכן ע"פ ההגדרה של E' אין קשת בין אף $v,u \in V'$ ב G'. לפיכך V' היא קבוצה בלתי תלויה ב G'.

($\Rightarrow$) אם אין קבוצה מרוחקת בגודל k, נניח בשלילה ש V' היא קבוצה בלתי תלויה ב G' כך ש $|V'| = k$. זו כמובן סתירה משום שעבור כל זוג צמתים מ V' מתקיים $\{v,u\} \notin E'$. ע"פ הבנייה של E' זה אומר שבין כל זוג צמתים אין מסלול באורך 2 או פחות ב G. לפיכך V' היא גם קבוצה מרוחקת ב G.

(ב) נבקש לבנות מ $G = (V,E)$ גרף $G' = (V',E')$ כך שבין כל זוג צמתים שאינם שכנים יפריד מסלול בגודל מ2. נעשה זאת ע"י החלפת כל קשת במסלול פשוט באורך 2, לצורך כך נוסיף ל V צומת עבור כל קשת, סה"כ m קשתות. כדי למנוע פתרונות המכילים את הצמתים החדשים נהפוך את כל m הצמתים החדשים לקליקה. ע"פ הבנייה $V' = V \cup \{v_e | e \in E\}$, ו $E' = \{\{v_{v,u}, v\}, \{v_{v,u}, u\} | \{v,u\} \in E\} \cup \{\{v_e, v_{e'}\} | e,e' \in E\}$.



ב G יש קבוצה בלתי תלויה בגודל k אם ורק אם ב G' יש קבוצה מרוחקת בגודל k.

($\Leftarrow$) אם $V'' \subseteq V'$ היא קבוצה בלתי תלויה ב G היא גם קבוצה מרוחקת ב G'. ראשית נסיק כי V'' לא מכילה צמתים חדשים משום שהיא קבוצה בלתי תלויה ב G. על פי הבנייה של G', כל זוג צמתים שאינם סמוכים ב G אינם סמוכים גם ב G', כלומר יש ביניהם לפחות שתי קשתות ב G ומשום שכל קשת מוחלפת במסלול באורך 2 אורך מסלול זה ב G' הוא לפחות 4. חיברנו את כל הצמתים החדשים ב G' לקליקה, לפיכך המסלול הקצר ביותר ביניהם ב G' יהיה באורך 3 לכל הפחות – קשת לתוך הקליקה החדשה, קשת בתוך הקליקה החדשה וקשת היוצאת מהקליקה החדשה.

($\Rightarrow$) אם $V'' \subseteq V'$ היא קבוצה מרוחקת ב G' היא גם קבוצה בלתי תלויה ב G. נניח בשלילה ש V'' אינה קבוצה בלתי תלויה ב G', כלומר קיימת קשת $\{v,u\} \in E'$ בין שני צמתים $v,u \in V''$. אנחנו יודעים שהצמתים החדשים אינם קיימים ב V לכן $v,u \in V$. קיבלנו סתירה משום שע"פ הבנייה כל קשת הוחלפה במסלול באורך 2, אם $\{v,u\} \in E$ אז V'' שמכילה את v,u אינה קבוצה מרוחקת. $\square$

תרגיל 2

הראו רדוקציה מבעיית SAT לבעיית 3SAT.

בעיית 3SAT
קלט: נוסחת CNF Φ כך שבכל פסוקית ב Φ בדיוק שלושה משתנים.
שאלה: האם יש Φ השמה מספקת?

הדגימו את הרדוקציה באמצעות נוסחת ה-SAT הבאה:

$$\Phi = (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5)$$

פתרון:

נבחין ש נוסחת SAT היא ספיקה כאשר כל אחת מהפסוקיות בה ספיקות, נשתמש באבחנה זו כדי לבנות מכל פסוקית SAT אוסף פסוקיות 3SAT כך שאם הפסוקית אינה ספיקה, לא תהיה השמה מספקת לאוסף הפסוקיות. כעת נוכל להתמקד בכל פעם בפסוקית אחת, נתייחס לשלושה מקרים:

1. פסוקיות עם שלושה משתנים נשארות ללא שינוי.

$$(x_1 \vee x_2 \vee \neg x_3) \rightarrow (x_1 \vee x_2 \vee \neg x_3)$$

2. נחליף כל פסוקית עם פחות משני משתנים בפסוקית עם שלושה באמצעות חזרה על המשתנים הקיימים.

$$(\neg x_1 \vee x_3) \rightarrow (\neg x_1 \vee \neg x_1 \vee x_3)$$

3. נחליף כל פסוקית עם יותר משלושה משתנים בשרשרת של פסוקיות "המקושרות" במשתני עזר.

$$(\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5) \rightarrow (\neg x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee y_2) \wedge (\neg y_2 \vee \neg x_4 \vee x_5)$$

קל לראות שכל השמה המספקת פסוקיות בגודל 2 או 3 של SAT יסופקו ע"י אותה השמה בנוסח 3SAT שבנינו. נשים לב שאם השמה של אחד המשתנים מספקת את פסוקית בגודל 4 ומעלה, ניתן לספק את שרשרת הפסוקיות משום שמשתני העזר "חופשיים" לספק את השרשרת.

y_1	y_2	$(y_1) \wedge (\neg y_1 \vee y_2) \wedge (\neg y_2)$
1	1	F
1	0	F
0	1	F
0	0	F

לא ניתן לספק את שרשרת הפסוקיות רק באמצעות השמה של משתני העזר, ניתן לראות זאת באמצעות טבלת האמת של השרשרת, כל השמת אמת של משתנה y_1 תכפה השמת שקר של משתנה y_ℓ בפסוקית האחרונה. □

תרגיל 3

הראו רדוקציה מבעיית SAT לבעיית כיסוי הצמתים.

בעיית כיסוי הצמתים
קלט: גרף לא מכוון G ומספר טבעי k.
שאלה: האם יש ב G קבוצת צמתים בגודל k כך שמחיקת הקבוצה תותיר את G ללא קשתות?

פתרון:

נדגים באמצעות נוסחת ה-SAT הבאה:

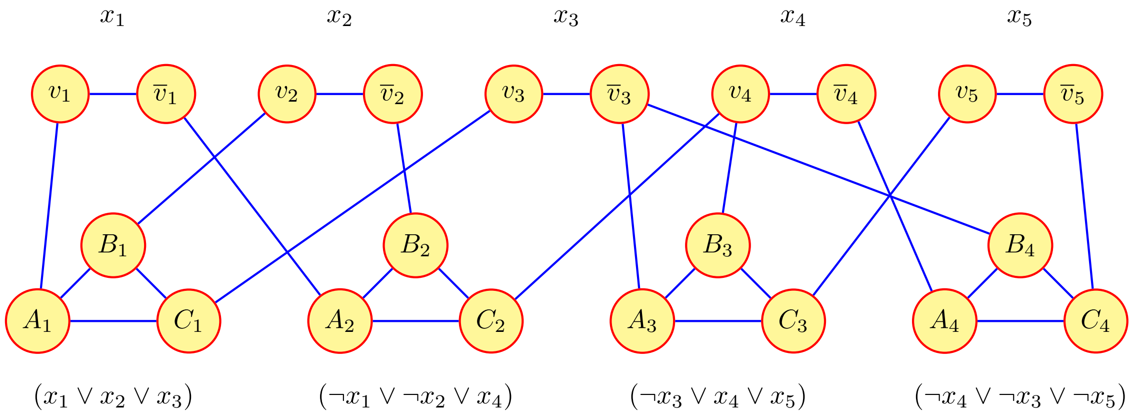
$$\Phi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_3 \vee x_4 \vee \neg x_4 \vee \neg x_3)$$

נמיר אותה לנוסחת 3SAT ע"פ הרדוקציה של תרגיל 2:

$$\Phi' = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_3 \vee x_4 \vee x_5) \wedge (\neg x_4 \vee \neg x_3 \vee \neg x_5)$$

נבנה מ Φ' גרף G כך:

1. עבור כל משתנה x_i ניצור זוג *צמתי השמה* v_i ו $\overline{v}_i$ וקשת $\{v_i, \overline{v}_i\}$ לה נקרא "קשת השמה".
2. עבור כל פסוקית:
   (א) ניצור קליקה של שלושה *צמתי פסוקית* A_j, B_j ו C_j. נקרא לקשתות בינהם "קשתות פסוקית".
   (ב) נחבר כל צומת פסוקית ב"קשת בקרה" למשתנה המתאים לו ע"פ הפסוקית.



בבעיית כיסוי הקודקודים נבקש לכסות כמה שיותר קשתות עם כמה של פחות צמתים, ניתן לחשוב על $2m+n$ כמעין "תקציב".

בגרף G קיים כיסוי קודקודים בגודל $k=2m+n$ אם ורק אם Φ ספיקה.

$(\Leftarrow)$ אם Φ ספיקה קיימת עבורה השמה מספקת β. ניקח לכיסוי הצמתים את הצומת v_i אם בהשמה β המשתנה $x_i=1$ אחרת, אם $x_i=0$, ניקח את $\overline{v}_i$. כך נעשה עבור כל אחד מהמשתנים, בכך כיסינו את כל קשתות ההשמה. אנחנו יודעים ש β היא השמה מספקת ולכן עבור כל פסוקית אחת מקשתות הבקרה מכוסה. עבור כל פסוקית, נבדוק איזה משתנה מהפסוקית מספק אותה וניקח אל הפתרון את שני צמתי הפסוקית הנותרים. בכך כיסינו את כל קשתות הפסוקית ואת שתי קשתות הבקרה הנותרות. כיסינו את כל הקשתות בגרף באמצעות $2m+n$ צמתים.

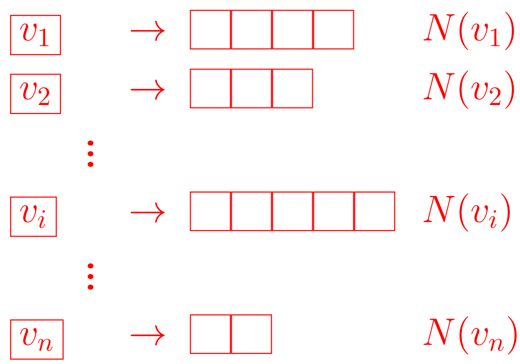
$(\Rightarrow)$ אם קיים כיסוי קודקודים בגודל $2m+n$ אז Φ ספיקה. אם Φ לא ספיקה אז לא קיים כיסוי קודקודים בגודל האמור. נבחין בין שלושה סוגי קשתות: קשתות השמה, קשתות פסוקית וקשתות בקרה.

ניתן לראות שעבור כל משתנה x_i נאלץ לבחור ב v_i או ב $\overline{v}_i$ על מנת לכסות את הקשת $\{v_i, \overline{v}_i\}$. לפיכך כל כיסוי צמתים "יבזבז" n מהתקציב כדי לכסות את קשתות ההשמה. את קשתות הפסוקית ניתן לכסות אך ורק ע"י לקיחת שני צמתים מתוך צמתי הפסוקית לכיסוי הצמתים, אם ניקח לפתרון רק צומת אחד, תישאר קשת אחת לא מכוסה בפסוקית ולכן לא יהיה לנו כיסוי צמתים. כלומר כל כיסוי צמתים יכיל n צמתי השמה ו$2m$ צמתי פסוקית. שני צמתים הפסוקית שבחרנו יכסו שתיים מתוך שלוש קשתות הבקרה. הקשת הנותרת מכוסה אך ורק אם בחרנו במשתנה המתאים לצומת הפסוקית הנותרת. אך הנחנו ש Φ אינה ספיקה ולכן אם קשת הבקרה השלישית מכוסה עבור כל פסוקית מתקבלת סתירה, משום שאז נקבל השמה מספקת עבור Φ – פשוט נציב $x_i=1$ עבור כל צומת השמה בפתרון. $\square$

13 חזרה למבחן

תרגיל 1

נתון גרף הממומש ברשימת שכנויות. תארו אלגוריתם יעיל ככל האפשר למיון הצמתים לפי הדרגה שלהם. אם עשיתם שימוש באלגוריתם ידוע כלשהו נמקו מדוע עשיתם זאת. מה הסיבוכיות של האלגוריתם?



פתרון:

1. $\mathcal{O}(n)$ נוסיף לכל צומת תא נוסף שיכיל את דרגת הצומת $\mathcal{O}(n)$
2. $\mathcal{O}(m+n)$ נחשב את הדרגה של כל צומת: מעבר על כל הצמתים וספירת הקשתות היוצאות מהם.
3. $\mathcal{O}(n+n)$ מאחר והמספרים שלמים והמספר המקסימלי חסום ב $\mathcal{O}(n)$ נשתמש במיון מניה:
   (א) נמצא את הדרגה המקסימלית בזמן $\mathcal{O}(n)$.
   (ב) ניצור מערך מניה בגודל הדרגה המקסימלית. בכל תא במערך תהיה רשימה מקושרת (כדי שנוכל לקשר את הצמתים).
   (ג) נעבור על מערך הצמתים והדרגות. ונצרף לרשימה המקושרת שבמערך המניה כל צומת לפי הדרגה שלו.
   (ד) נעבור תא תא במערך המניה ונדפיס את האיברים שברשימות המקושרות שבתוכו לפי הסדר.

סה"כ זמן הריצה הוא $\mathcal{O}(n+m)$.

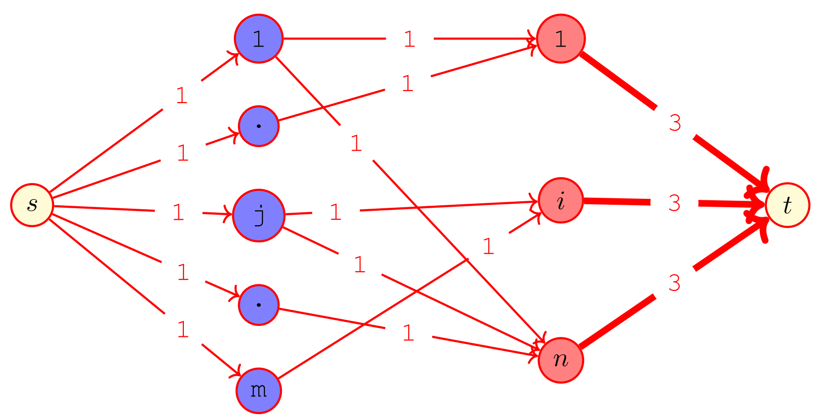
תרגיל 2

באוניברסיטה m סטודנטים ו-n כיתות (סטודנט יכול להשתייך ליותר מכיתה אחת). הצע אלגוריתם שעונה על השאלה האם ניתן להציג עבור כל כיתה ועד של 3 סטודנטים כך שהועדים יהיו זרים (כלומר שסטודנט לא ייצג יותר מכיתה אחת). כתבו מהי סיבוכיות האלגוריתם.

פתרון:

נבנה רשת זרימה באופן הבא: ניצור צומת עבור כל סטודנט וכל כיתה וכן צומת מקור ויעד. נעביר קשת בין צומת המקור לכל צמתי הסטודנטים עם קיבולת 1. נעביר קשת בין כל צומת סטודנט לצמתי הכיתות אליהם הוא משתייך עם קיבול של 1. נעביר קשת בין כל צומת כיתה לצומת היעד עם קיבול של 3. נריץ אלגוריתם פורד פולקרסון למציאת הזרימה המקסימלית

אם ערך הזרימה המקסימלית שווה למספר הכיתות כפול 3 אז ניתן להקצות ועדים זרים של 3 סטודנטים לכל כיתה. אחרת לא ניתן.



סיבוכיות: בניית הרשת $\mathcal{O}(mn)$ במקרה שכל סטודנט שייך לכל הכיתות הרצת אלגוריתם פורד פולקרסון בסיבוכיות $\mathcal{O}(F^*(|E|+|V|))$ כאשר הזרימה המקסימלית (F^*) יכולה להגיע לכל היותר ל- $3n$. נשים לב ש $|V| = n+m$ ו $|E| \in \mathcal{O}(nm)$ מכאן שהסיבוכיות הכוללת היא $\mathcal{O}(mn^2)$.

תרגיל 3

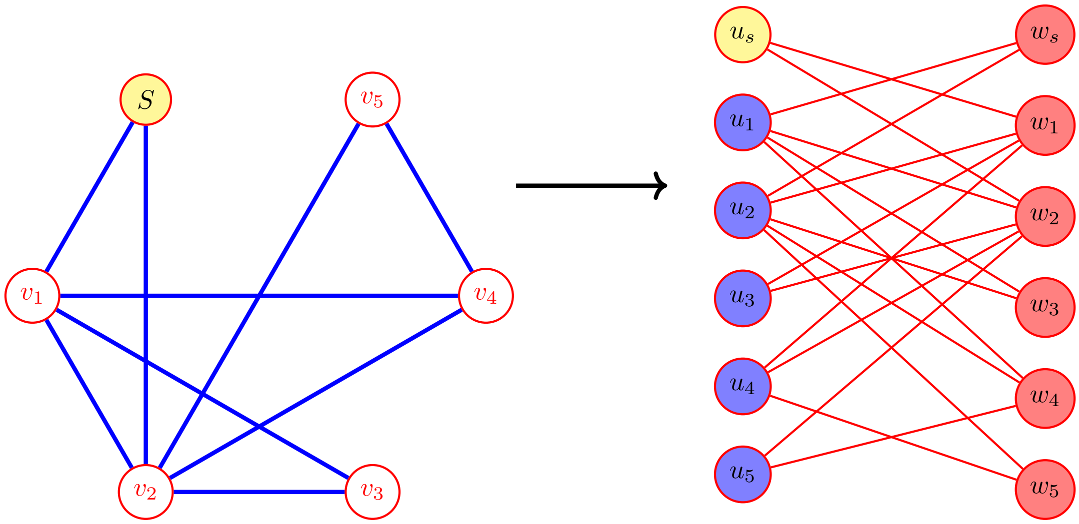
כתבו אלגוריתם שמקבל גרף לא מכוון $G=(V,E)$ וצומת s ומחשב לכל צומת v את המסלול (לאו דווקא פשוט) באורך זוגי הקצר ביותר.

רמז: יש להעזר ברדוקציה.

פתרון:

נבנה מהגרף הנתון גרף $G'=(V',E')$ באופן הבא, נשכפל את קבוצת הצמתים כך שעבור כל $v_i \in V$ יהיו שני צמתים $u_i, w_i \in V'$. עבור כל קשת $\{v_i, v_j\} \in E$ תהיינה שתי קשתות $\{u_i, w_j\}, \{u_j, w_i\} \in E$. בניית הגרף לוקחת $\mathcal{O}(|V|+|E|)$ זמן.

נריץ אלגוריתם BFS כדי לחשב את המסלולים הקצרים ביותר מ u_s לשאר צמתי הגרף. אפשר להמיר בקלות כל מסלול לצומת u_i בG' למסלול באורך זוגי בין s ל v_i ב G. הגרף G' ליניארי בגודל של G לכן BFS ייקח גם כן $\mathcal{O}(|V|+|E|)$. סה"כ $\mathcal{O}(|V|+|E|)$ זמן.


S
v5
v1
v4
v2
v3
us
ws
u1
w1
u2
w2
u3
w3
u4
w4
u5
w5