30.12.20

הרצאה 11 - Data Structures

מסלולים קצרים ביותר

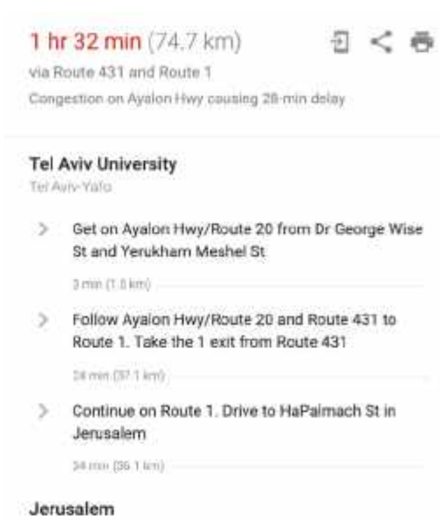
Shortest Paths Algorithms

ישנם 3 סוגים של בעיות מסלולים קצרים ביותר:

1. מסלול קצר ביותר בין שני קודקודים (s ו-t)
2. מסלולים קצרים ביותר מקודקוד אחד לכל האחרים (מ-s לכל $v \in V$)
3. מסלולים קצרים ביותר בין כל זוג קודקודים בגרף (מכל $s \in V$ לכל $t \in V$)

ראינו פתרונות ל-1 ו-2 ע"י שימוש ב-BFS בגרפים לא מכוונים ללא משקל.
היום נרצה לפתור את הבעיות האלו עבור גרפים מכוונים עם משקל.

נשים לב שבגרף לא מכוון ללא משקלים המסלול הקצר ביותר מוגדר כך שהוא כולל מספר מינימלי של צלעות, ואילו עבור גרפים מכוונים עם משקל לא בהכרח הכי קצר מבחינת צלעות, אלא מתחשב במינימליות המשקל.



דוגמה:

קודקודים: ערים וצמתים

צלעות: כבישים

משקל: מרחק, זמן, השילוב בין השניים

הגדרות ותכונות עבור מסלולים קצרים ביותר

תכונות:

גרף ממושקל

* גרף ממושקל הוא גרף שבו לצלעות הן מעלות משקל/מחיר ו- $w(v_i,v_j)\in\mathbb{R}$.
  חידוש
* גרף הוא גרף ממושקל שבו משקל כל צלע הוא 1.
* אם אין אף מסלול בין שני קודקודים אז המחיר/משקל הוא ∞.

המשקל של מסלול $p(u,v)$ / $u\overset{p}{\Rightarrow}v$ מוגדר כסכום המשקלים של הצלעות בו:

$w(Path(u,v))=\sum_{i=0}^{k}w(v_{i-1},v_i)$ כאשר $u=v_0$ ו- $v_k=v$.

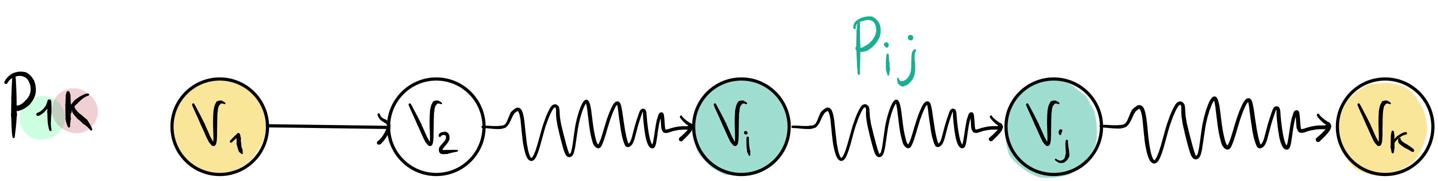
מסלול קצר ביותר עפ"י משקל:

$$\delta(u,v)=\begin{cases}\min\{w(p):u\overset{p}{\Rightarrow}v\}\\ \infty & \text{כאשר אין מסלול}\end{cases}$$

ננית ש- $w\neq-\infty$, $w\neq\infty$ וגם $w+(-\infty)=(-\infty)$, $w+\infty=\infty$.

* גרף $G=(V,E)$ והיה גרף מכוון ממושקל כאשר $V=\{v_1,\dots,v_n\}$ הם קודקודים ו- $E=\{(e_{ij},w(e_{ij}))\}$ הן הצלעות בגרף ומשקלן.

* נסמן מסלול $P_{1k}=\langle v_1,v_2,\dots,v_k\rangle$ כמסלול הקצר ביותר בין v_1 ל- v_k.
ונסמן מסלול $P_{ij}=\langle v_i,\dots,v_j\rangle$ כתת-מסלול של P_{1k}, כך ש- $1\le i,j\le k$
כאשר $v_i=v_j$ מדובר במסלול עם מעגל.



צלעות בעלות משקל שלילי

צלעות בעלות משקל שלילי מעידות על "רווח". במקרה כזה אנחנו "מחסירים" מהמשקל במסלול.

$$s_1 \xrightarrow[2]{pay} s_2 \xrightarrow[-3]{gain} s_3$$

גרף שהוא בעל מעגל שלילי, משקלו הוא $-\infty$, כלומר "רווח" אינסופי ולכן נפסול אותם, נפסול גם את המעגלים בעלי משקל 0, כי הם מוסיפים צלעות בצורה מיותרת.

המסלול הקצר ביותר מוגדר כל עוד אין מעגלים שליליים.

הערה: נתיר צלעות בעלות משקל שלילי אבל נאסור על מעגלים בעלי משקל שלילי.

תכונה 1: המסלול הקצר ביותר בין כל שני קודקודים לא כולל מעגלים חיוביים.

הוכחה: נניח בשלילה שקיים מעגל חיובי בין שני קודקודים ב- P_{1k}, אז המסלול הוא לא הקצר ביותר, בסתירה לאופן שבו הגדרנו אותו.

תכונה 2: כל תת-מסלול של מסלול קצר ביותר, גם הוא קצר ביותר.

הוכחה: נניח בשלילה שתת-המסלול P_{ij} של P_{1k} אינו הקצר ביותר.
אם כך קיים מסלול q_{ij} שקצר ממנו. אם כך נוכל להחליף את P_{ij} ב- q_{ij} ונקבל את q שקצר מ-P_{1k}, בסתירה לכך שהוא כבר הקצר ביותר.

ייצוג המסלולים הקצרים ביותר

כל המסלולים הקצרים ביותר מ-s לכל אחד מהקודקודים הוא עץ רוחב BFS:

$G_\pi = (V_\pi, E_\pi)$ such that

① $V_\pi = \{v \in V \mid v.\pi \neq null\} \cup \{s\}$

② $E_\pi = \{(v.\pi, v) \mid v \in V / \{s\}\}$

נשים לב שהעץ הזה לא חייב לכלול את כל הקודקודים, ולא יכלול את אלו שלא נגישים לשורש.

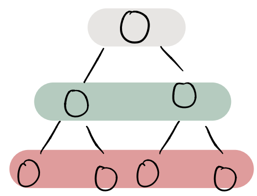
תזכורת BFS

מחפשים את כל הקודקודים ברמה ה-i לפני שעוברים לקודקודים ברמה ה-i+1.
באלגוריתם BFS שלמדנו בגרף לא-מכוון לא ממושקל, המסלול הקצר ביותר היה
המסלול מתחילת הריצה ועד הרגע שבו פגשנו קודקוד לראשונה.
אם נוסיף משקלים לגרף, לא נוכל להתמודד שוב באופן כזה, לכן נצטרך לבצע התאמה.

נזכר בנוסף בהפרדה הבאה לצבור קודקודי הגרף:

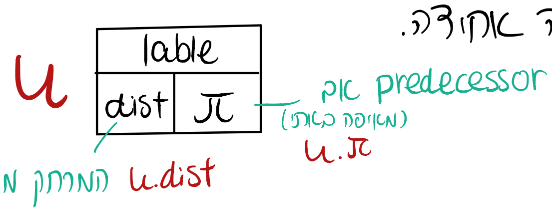
(1) כאלו שביקרנו בהם Visited
(2) הנוכחיים Current - ישמרו בתור Q
(3) לא ביקרנו בהם not-visited

בגרף:



lable

ונשים לב שגם במקרה כזה השדות הנ"ל כבר לא רלוונטים, כי על מנת למצוא את המסלול
הקצר ביותר נצטרך לבצע חיפושים ולחזור אחורה לקודקודים שכבר ביקרנו בהם.
התמונה תשאר דומה, אך ההתקדמות לא תהיה אחידה.
נזכר גם בשדות dist ו-π של כל קודקוד:



במהלך BFS השתמשנו ב-∞ כערך התחלתי למרחק בין קודקוד s לשאר הקודקודים
ונשתמש בשיטה זו גם הפעם.

איך נבצע התאמות לגרף ממושקל?

בגרף ממושקל, כפי שציינו, השדה v.dist יכול להשתפר עד מציאת מסלול משתלם יותר.
ולכן נשתמש בשיטה חדשה הנקראת הקלה (Relaxation) שתשפר את המסלול
למסלול האופטימלי, תוך כדי הסריקה של הגרף, ועד סופה.

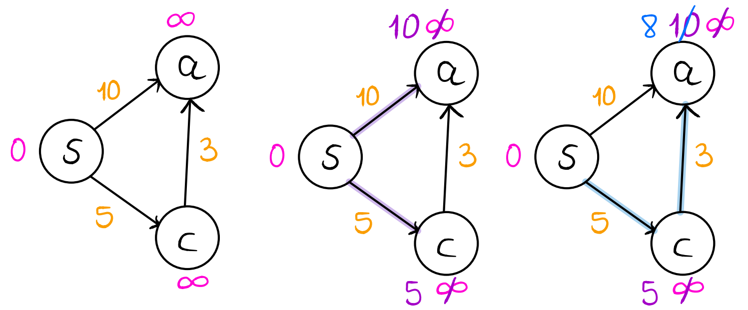
הקלה היא שיטה בה משפרים את המסלול הקצר ביותר מ-s ל-v עד כה, כלומר, $v.dist$ הולך וקטן, עד שמגיעים למסלול הקצר ביותר $\delta(s,v)$.

מדובר באלגוריתם חמדני

האלגוריתם:

```
RELAX(u, v, w)
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
    v.π = u
```

דוגמה:



נוסיף פעולה זו ל-BFS הקיים כדי להתאים למשקלים.

תכונות של מסלולים קצרים ביותר והקלה

1. **חסם עליון** לכל $v \in V$ מתקיים $\underset{\text{נוכחי}}{v.dist} \geq \underset{\text{מינימלי}}{\delta(s,v)}$ במהלך כל הריצה, ברגע שיתקיים שוויון $v.dist$ לא ישתנה.
2. **ללא מסלול** אם אין מסלול בין s ל-v נשאר במצב בו $v.dist = \delta(s,v) = \infty$.
3. **מונוטוניות יורדת** $v.dist$ הולך ויורד (מתחיל ב-∞ ומשתפר בכל הקלה)
4. **סיום התהליך** בסופו של התהליך $v.dist = \delta(s,v)$
5. **אי-שוויון המשולש** לכל $E \ni e = (u,v)$ מתקיים $\delta(s,v) \leq \delta(s,u) + \delta(u,v)$
6. **התכנסות** אם יש מסלול בין s ל-u וכעת $u.dist = \delta(s,u)$, אז במהלך ההקלה של צלע (u,v) יתקיים: $v.dist = \delta(s,v)$.
7. **הקלת מסלול** יהי $P_{1k} = \langle v_1, v_2, \dots, v_k \rangle$ המסלול הקצר ביותר בין v_1 ל-v_k כאשר הצלעות הן: $(v_1,v_2),(v_2,v_3),\dots,(v_{k-1},v_k)$. אז מתקיים $v_k.dist = \delta(s,v_k)$
8. **הקודם של תת-הגרף** כאשר $v.dist = \delta(s,v)$ אז לכל $v \in V$ מתקיים כי תת על הקודמים הוא עץ BFS של מסלולים קצרים ביותר, ששורשו s.

אלגוריתמי מסלול קצר ביותר

1. אלגוריתם דייקסטרה Dijkstra Algorithm

* הכללה של BFS
* מניח שאין משקלים שליליים
* $O(|V|\log|E|)$ בזמן ו-$O(|V|+|E|)$ מקום

2. אלגוריתם בלמן-פורד Bellmann-Ford Algorithm

* מבצע $|V|$ איטרציות, מבצע הקלות על כל הצלעות ב-E בכל איטרציה.
* מותר משקלים שליליים
* $O(|E||V|)$ בזמן ו-$O(|V|+|E|)$ מקום

שני האלגוריתמים חמדניים ומשפרים את המרחק.
ההבדל בהקלה הוא שבדייקסטרה עושים רק לשכנים ובבלמן-פורד לכל הצלעות.

אלגוריתם דייקסטרה Dijkstra Algorithm

ביצוע כמו BFS בגרף לא ממושקל עם 3 עדכונים:

1. נשתמש במשקלים כפונקציית מרחק בין קדקודים (במקום 1).
2. נשתמש בערימת מינימום במקום בתור פשוט כדי לבחור שכן על בסס ה-dist המינימלי. ולכן גל ההתפשטות הולך להיות לפי הקודקוד בעל ה-v.dist המינימלי.
3. ברגע שהגיע לקודקוד הנבחר על פי 2, נבצע הקלה על כל שכניו-כל מי שיוצא ממנו.

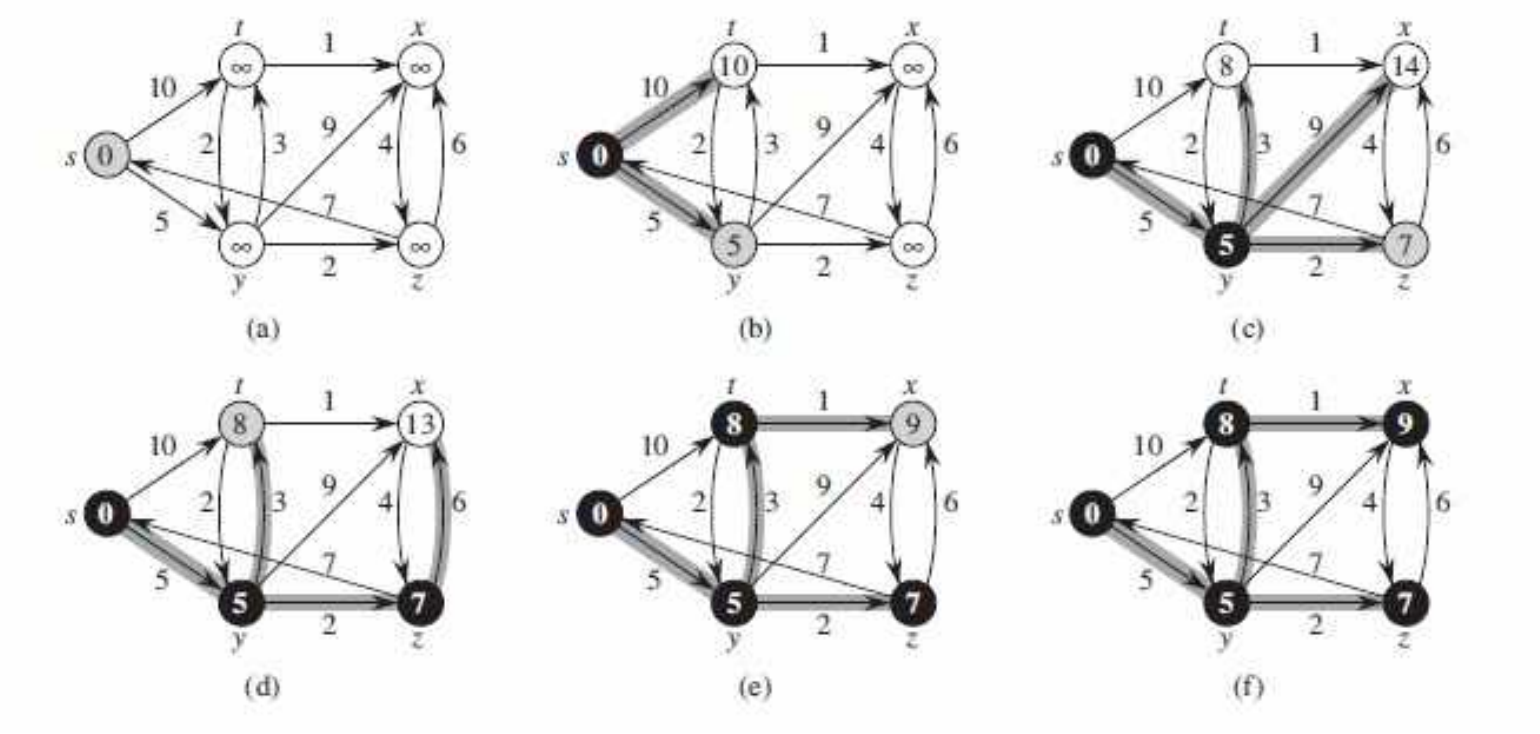
האלגוריתם:

החלק הראשון הוא אתחול.

```
DIJKSTRA(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
S = Ø
Q = G.V
while Q ≠ Ø
    u = EXTRACT-MIN(Q)
    S = S ∪ {u}
    for each vertex v ∈ G.Adj[u]
        RELAX(u, v, w)
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

דוגמה להרצה:


(a) (b) (c) (d) (e) (f)

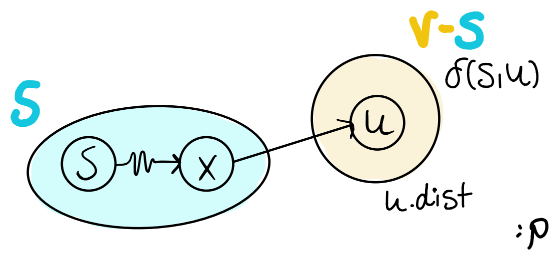
נכונות אלגוריתם דייקסטרה

משפט: בסוף ריצת אלגוריתם דייקסטרה $v.dist = \delta(s,v)$ עבור כל קודקוד $v \in V$.

הוכחה: נוכיח את שמורת הלולאה. נרצה להוכיח שבתחילת כל איטרציה של לולאת ה-while התכונה $v.dist = \delta(s,v)$ נשמרת לכל הקודקודים ב-S.

בהתחלה $S=\phi$ נכון באופן טריוויאלי. סוג של אינדוקציה

ובסוף $S=V$ ו-$Q=\phi$ אז לפי השלב הקודם בהוכחה נקבל שגם עבור V זה יהיה נכון.



לכן השלב המאתגר בהוכחה הוא בכל שאר המקרים:

נניח בשלילה ש-u הקודקוד הראשון שבו $u.dist \neq \delta(s,u)$ כאשר u נוסף ל-S

(בצורה אחרת, המקרה הראשון בו התכונה הנ"ל מופרת).

לכן נקבל ש-$u \neq s$ כי $s.dist = \delta(s,s) = 0$ והתכונה לא מופרת.

לכן גם $S \neq \emptyset$ (כי s כבר שם).

חייב להיות מסלול מ-s ל-u, אחרת $u.dist = \delta(s,u) = \infty$ והתכונה לא מופרת.

נסמנו p, ונפצל את p ל-3 תתי-מסלולים: $s \xRightarrow{p_1} x$, $\overset{p_2}{(x,y)}$, $y \xRightarrow{p_3} u$

כאשר $x \in S$.

טענה: $y.dist = \delta(s,y)$ כאשר u נוסף ל-S.

הוכחה:

מכיוון ש-x נוסף ל-S לפני u (הראשון שמפר את התכונה) לכן עפ"י הנחה בשלילה $x.dist = \delta(s,x)$ מכיוון שביצענו הקלה לצלע (x,y) אז $y.dist = \delta(s,y)$ (עפ"י תכונה 4.)

בנוסף, בגלל ש-y מופיע לפני u והמשקלים הם אי-שליליים, אז $\delta(s,y) \le \delta(s,u)$ (עפ"י תכונה 6.) ובנוסף נשים לב שמתקיים:

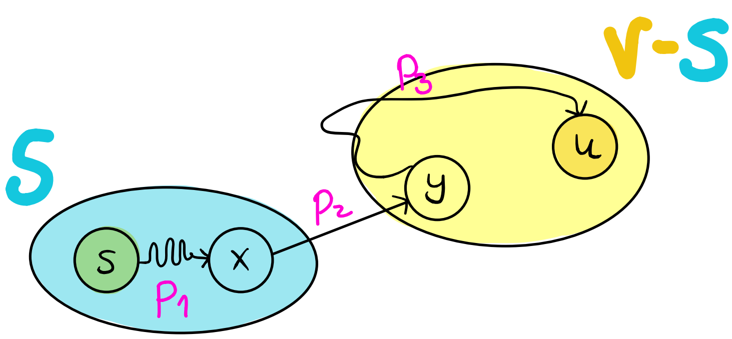
⊛ $y.dist = \delta(s,y) \le \delta(s,u) \le u.dist$ (תכונה 1.)

מכיוון ש-$y,u \in V/S$ אז $u.dist \le y.dist$ ובגלל עפ"י ⊛ מתקיים ש: $\delta(s,u) = u.dist$ בסתירה להנחה.

אינטואיציה להוכחה:

יש מסלול מ-S ל-u, אם נחשוב שבאיזשהו שלב במסלול הזה הוספתי קטע
שלא מקיים את התכונה ($u.dist \neq \delta(s,u)$)
לא ייתכן כי זה סותר (תכונת החסם העליון ותכונת ההתכנסות, 1. ו-6.)

המחשה להוכחה:



$s \xrightarrow{P_1} x,\ (x,y),\ y \xrightarrow{P_3} u$

סיבוכיות זמן הריצה של אלגוריתם דייקסטרה:

האלגוריתם מבצע $|V|$ פעולות insert-Queue ו-Extract-Min $(\log(V))$
ו-$|E|$ פעולות Decrease-Key. ולכן סה"כ נקבל:

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

אם הגרף מאוד צפוף - לא כדאי להשתמש בערימת מינימום. נסביר:
כאשר תור הקדימויות ממומש כערמת מינימום ו-$|E|=O(V^2)$ המימוש אינו אופטימלי
כי מתבצעות יותר פעולות Decrease-Key מאשר Extract-Min.
הפתרון הוא לממש כמערך, לאתר מינימום $O(|V|)$ אבל insert ו-decrease ייקח
$O(1)$ ובמקרה כזה נקבל: $O(|V^2|)+O(|E|)=O(|V^2|)$ במקום $O(|V^2|\log|V|)$

אלגוריתם בלמן-פורד Bellmann-Ford Algorithm

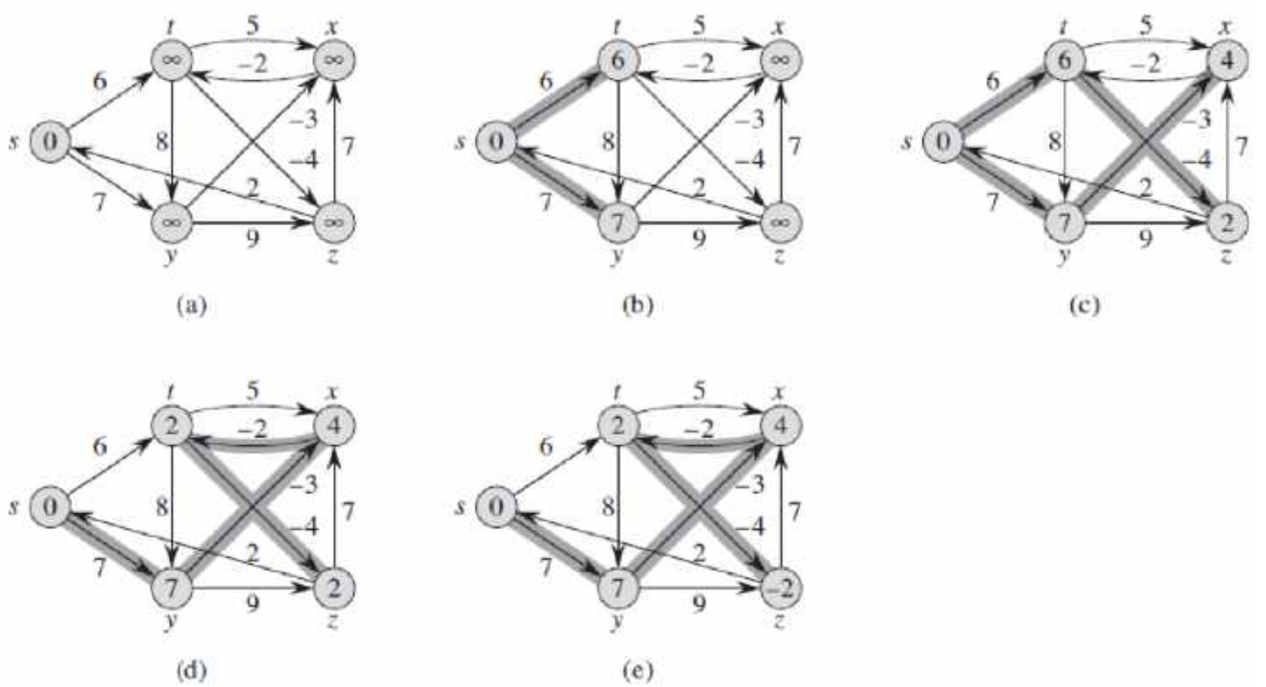
רעיון כללי: נרצה למצוא את המסלול הקצר ביותר מ-s לכל קודקוד $v \in V$.
מספר הצלעות המקסימלי במסלול ללא מעגלים הוא $|V|-1$ כאשר הוא עובר ב-$|V|$ קודקודים. לכן מספיק לבדוק $|V|-1$ צלעות לכל היותר.
אם נריץ $|V|$ פעמים (שלב נוסף) ונראה שהמשקל ירד - עתראות המעגל השלי!

האלגוריתם:

```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| - 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE        קיים מעגל שלילי
return TRUE
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

דוגמה להרצה:



(a) (b) (c) (d) (e)

6.1.21

Data Structures - הרצאה 12
מסלולים קצרים בין כל הקודקודים
All Shortest Paths Algorithms

נרצה לחשב את כל המסלולים הקצרים ביותר מכל קודקוד לכל קודקוד.

$|E|=O(|V|^2)$

פתרון פשוט:

נקח את אלגוריתם בלמן-פורד ונפעיל אותו |V| פעמים $O(|E|\cdot|V|\cdot|V|)=O(|V|^4)$

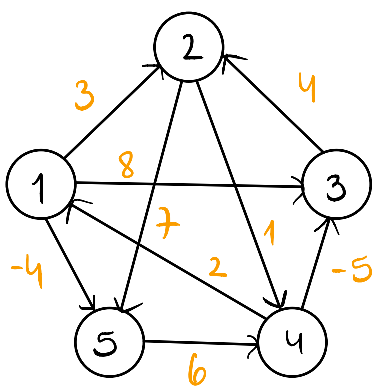
נקח את אלגוריתם דייקסטרה ונפעיל אותו |V| פעמים $O(|E|\cdot\log|V|\cdot|V|)=O(|V|^3\cdot\log|V|)$

על מנת לשפר את זמן הריצה, נוכל לנסות להשתמש בתתי המסלולים כתוב מסלולים קצרים ביותר.

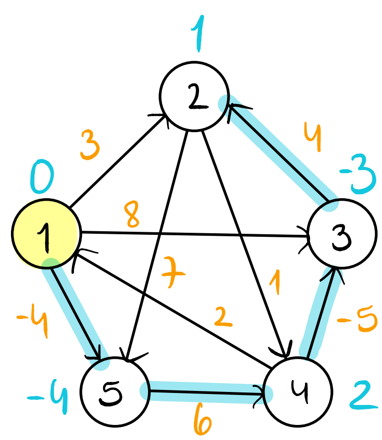
דוגמה:

נקבל וקטור מרחקים מ-1 לכל v

וקטור קודמים בדרך מ-1 לכל v

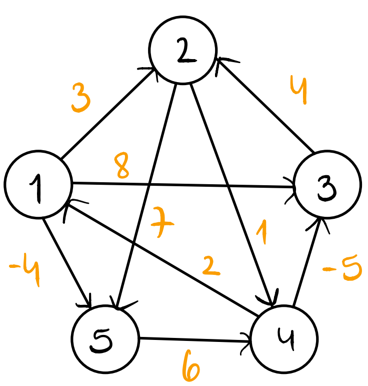


	1	2	3	4	5
π	∅	3	4	5	1

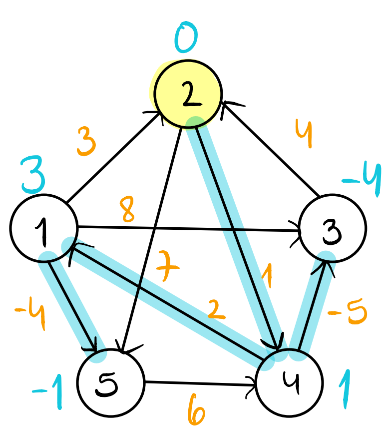


	1	2	3	4	5
dist	0	1	-3	2	-4

דוגמה:



	1	2	3	4	5
π	4	∅	4	2	1



	1	2	3	4	5
dist	3	0	-4	1	-1

ייצוג מטריציוני

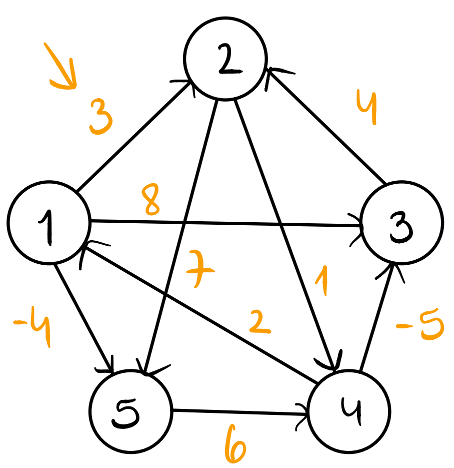
אם נרצה ייצוג קומפקטי של כל המסלולים, נשתמש ב-n וקטורים בגודל n,
כאשר n הוא מס' הקודקודים.

ע"י מטריצה $n\times n = |V|\times|V|$ נייצג את:
1. הגרף - מבחינת משקלים W
2. אורכי המסלולים הקצרים ביותר L
3. תתי גרף הקודמים π

1. מטריצת משקלי צלעות W מוגדרת כשהמקום ה-(i,j)-W הוא משקל הצלע $(i,j)=e$.

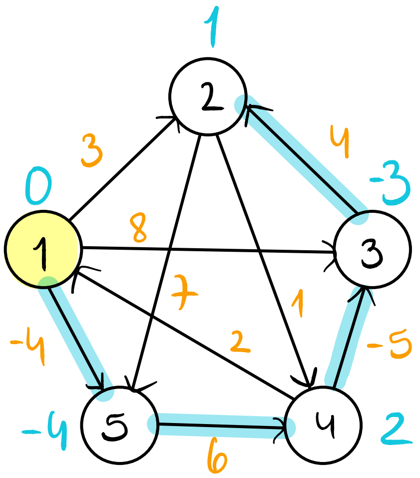
$$w_{ij}=\begin{cases} 0 & \text{if } i=j \\ w(i,j) & i\neq j \text{ and } (i,j)\in E \\ \infty & i\neq j \text{ and } (i,j)\notin E \end{cases}$$

$$\begin{array}{c} j\rightarrow \\ i\downarrow \end{array} \quad W=\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}$$



$$\begin{array}{c|c} & j\rightarrow \text{TO} \\ & \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \hline \begin{array}{c} i\downarrow \text{FROM} \\ 1\\2\\3\\4\\5 \end{array} & \begin{pmatrix} \cancel{0} & 3 & 8 & \infty & -4 \\ \infty & \cancel{0} & \infty & 1 & 7 \\ \infty & 4 & \cancel{0} & \infty & \infty \\ 2 & \infty & -5 & \cancel{0} & \infty \\ \infty & \infty & \infty & 6 & \cancel{0} \end{pmatrix} \end{array}$$

2. מטריצת אורכי מסלולים קצרים ביותר L מוגדרת כש-$L(i,j)$ מכילה את
אורך המסלול הקצר ביותר בין i ל-j.
כך מתחילות $u.dist$ שראינו באלגוריתמים קודמים.



1 2 3 4 5

L_1 (0 1 -3 2 -4)

j → TO

i ↓ FROM	1	2	3	4	5
1	0	1	−3	2	−4
2	3	0	−4	1	−1
3	7	4	0	5	3
4	2	−1	−5	0	−2
5	8	5	1	6	0

האלכסון "ישאר 0 כי המרחק מקודקוד לעצמו הוא 0.

הגרפים $G_{\pi,i}=(V_{\pi,i},E_{\pi,i})$ הם גרפי המסלולים הקצרים ביותר ששורשם i

באופן הבא ←

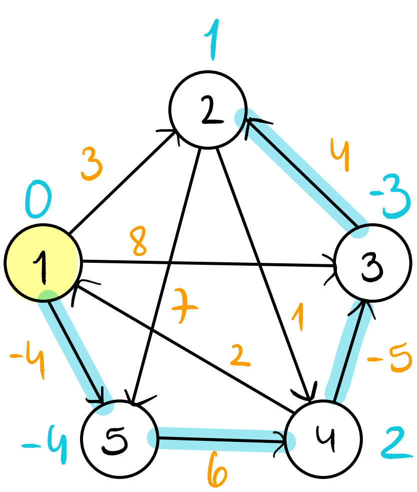
$$V_{\pi,i}=\{j\in V:\pi_{ij}\neq null\}\cup\{i\}$$
$$E_{\pi,i}=\{(\pi_{ij},j):j\in V_{\pi,i}-\{i\}\}$$

וכעת נוכל להגדיר:

3. מטריצת תת גרף הקודמים π מוגדרת כש-$\pi(i,j)$ הוא הקודם של j במסלול הקצר ביותר ל-i.

כאשר j=i או כשאין מסלול-נקבל null.

כדי שנוכל לשחזר את המסלולים



1 2 3 4 5

π (0 3 4 5 1)

j → TO

i ↓ FROM	1	2	3	4	5
1	null	3	4	5	1
2	4	null	4	2	1
3	4	3	null	2	1
4	4	3	4	null	1
5	4	3	4	5	null

איך נחלץ את המסלולים עצמן?

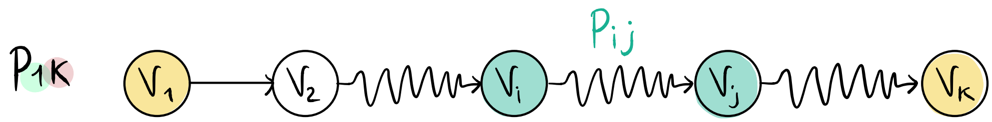
ניתן לעשות דוג' הרצה ע' השורה ה-I במטריצה.

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
if i == j
    print i
elseif π_ij == NIL
    print "no path from" i "to" j "exists"
else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
    print j
```

אלגוריתם זה ידפיס את כל המסלולים שבשורה ה-i.
אם נרצה להציג את כל המסלולים באופן כללי, נוסיף לולאה.

תזכורת: תכונות המסלולים הקצרים ביותר

1. כל תת-מסלול P_{ij} של מסלול קצר ביותר P_{1k}, גם הוא קצר ביותר.

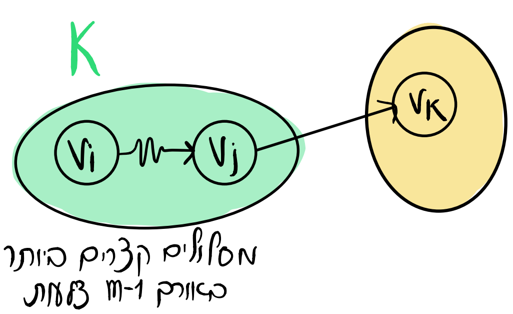


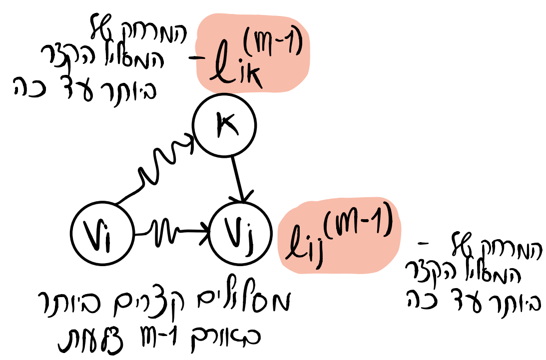
מבחינת משקל

2. המסלול הקצר ביותר מ-v_i ל-v_j באורך m צלעות לכל היותר, הוא אחד מהשניים:

א. המסלול הקצר ביותר באורך m-1 צלעות לכל היותר (ואז אין שיפור).

ב. המסלול הקצר ביותר באורך m-1 צלעות + המשקל של הצלע מקודקוד מתוך m-1 הקודקודים לקודקוד נוסף (שיפור), מסלול חדש עם m צלעות. נבחר את הסכום הקטן יותר.





כלומר, בסה"כ יוצא:

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

תכנות דינאמי

הצעדים בתכנות דינאמי הם:

1. אפיון המבנה של הפתרון האופטימלי.
2. הגדרה רקורסיבית של הפתרון האופטימלי.
3. חישוב הפתרון (במקרה שלנו: חישוב המינימום)
4. בניית הפתרון (במקרה שלנו: המסלולים הקצרים ביותר)

פתרון רקורסיבי של המסלולים הקצרים ביותר

יהי $l_{i,j}^{(m)}$ המשקל המינימלי של כל מסלול מקודקוד i לקודקוד j שיש לו לכל היותר m צלעות (מחליף את u.dist).

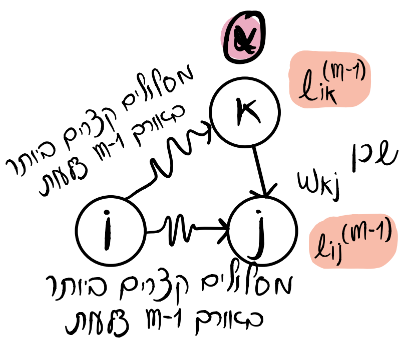
כאשר m=0:

$$l_{ij}^{(0)} = \begin{cases} 0 & i = j \\ \infty & i \neq j \end{cases}$$

מסלולים עצמיים

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

כאשר $m \geq 1$: $l_{i,j}^{(m)}$ הוא המינימום בין $l_{i,j}^{(m-1)}$ למסלול הקצר ביותר בהוספה של שכני הקודקוד:



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

$$= \min_{1\leq k\leq n}\{l_{ik}^{(m-1)} + w_{kj}\} \quad \textit{k = j included!}$$

$$\begin{array}{cc} & \begin{array}{ccccc} & & \text{Kj} & & \end{array} \\ (m-1) \quad L^{(1)} = & \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & 5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \begin{array}{c} \\ \\ \\ \text{iK} \\ \\ \end{array} \end{array}$$

($i \downarrow$, $j \rightarrow$)

נקשר את זה לפעולת ה-Relax:

```
RELAX(u, v, w)
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
    v.π = u
```

$\Longleftrightarrow$ $\min(v.dist,\ u.dist + w(u,v))$

זה דומה באופן שבו אנחנו מחליטים אם להשאיר עם המרחק שהיה לנו, או שנחליף אותו במסלול שעובר דרך k בציור.

פתרון: מציאת כל המסלולים הקצרים ביותר

תהי W מטריצת המשקלים ו-L מטריצת אורכי המרחקים הקצרים ביותר שחושבו עד כה.

נחשב סדרת מטריצות $(L^{(1)}, L^{(2)}, \dots, L^{(n-1)})$ עבור $m=1,\dots,n-1$.

כאשר $L^{(m)} = \ell_{ij}^{(m)}$ היא מטריצה של כל אורכי המסלולים הקצרים ביותר עם לכל היותר m צלעות.

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

בהתחלה, $W = L^{(1)}$. ובסוף $L^{(n-1)}$ כוללת את כל אורכי המסלולים הקצרים ביותר.

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

הפעולה הבסיסית היא לחשב את $L^{(m)}$ מ-$L^{(m-1)}$ ו-W. נראה כעת:

הרחבת כל המסלולים הקצרים ביותר בצלע אחת מ-$L^{(m-1)}$ ל-$L^{(m)}$

האלגוריתם:

```
EXTEND-SHORTEST-PATHS(L, W)
n = L.rows
let L' = (l'_ij) be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        l'_ij = ∞
        for k = 1 to n
            l'_ij = min(l'_ij, l_ik + w_kj)
return L'
```

סה"כ מוצא מינימום מבין כל האורכים של המסלולים

קריאה: Extand-Shortest-Paths($L^{(m-1)}$, W) returns $L^{(m)}$

סיבוכיות זמן ריצה: $O(|V|^3)$

חישוב המסלולים הקצרים ביותר דומה לכפל מטריצות

אופי המעבר של אורכי המסלולים דומה לכפל מטריצות, המעבר על העמודות והשורות במקביל, והצבירה של הסכום מתחלפת במינימום.

```
SQUARE-MATRIX-MULTIPLY(A, B)
n = A.rows
let C be a new n × n matrix
for i = 1 to n
    for j = 1 to n
        c_ij = 0
        for k = 1 to n
            c_ij = c_ij + a_ik · b_kj
return C
```

אם נחליף את $A=L, B=W, C=L'$

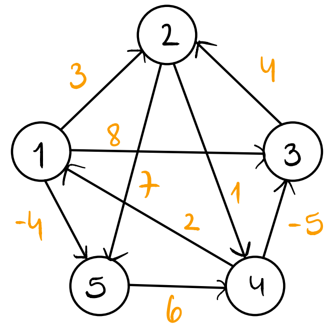
$l'_{ij}=\infty$

$l'_{ij}=\min(l'_{ij}, l_{ij}+w_{ij})$

$$\underbrace{\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix}}_{B} = \underbrace{\begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}}_{C=A\times B}$$

מסלולים עם לכל היותר 2 צלעות

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

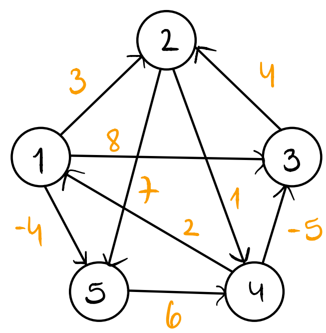


$$L^{(2)}_{ij} = \min\left(l^{(1)}_{ij}, l^{(1)}_{ik} + W_{kj}\right)$$

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

מסלולים עם לכל היותר 3 צלעות

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

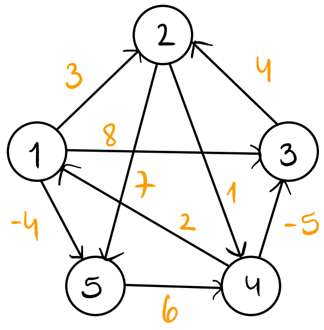


$$L^{(3)}_{ij} = \min\left(l^{(2)}_{ij}, l^{(2)}_{ik} + W_{kj}\right)$$

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

מסלולים עם לכל היותר 4 צלעות

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

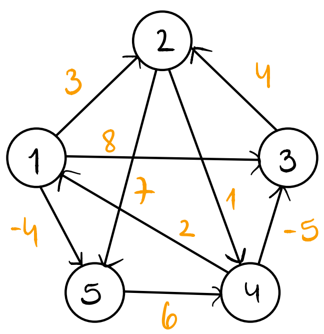


$$L^{(4)}_{ij} = \min(l^{(3)}_{ij}, l^{(3)}_{ik} + W_{kj})$$

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

מסלולים עם לכל היותר 5 צלעות

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



$$L^{(4)}_{ij} = \min(l^{(3)}_{ij}, l^{(3)}_{ik} + W_{kj})$$

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

נשים לב שלא התבצע שיפור אחרי n-1 איטרציות.

אלגוריתם All-Shortest-Paths

```
All-Shortest-Paths(W)
n = W.rows
L^(1) = W
for m = 2 to n-1 do:
    L^(m) = Extend-Shortest-Paths(L^(m-1), W)
return L^(m)
```

סיבוכיות: $O(|V|^4)$ - באסה.

לא נדרשת הוכחת נכונות כי האלגוריתם מסתמך על 2 התכונות הקודמות שכבר הוכחנו.

שיפור אלגוריתם למסלולים הקצרים ביותר

אבחנה:

נניח לחשב את a^n כאשר n הוא חזקה של 2 כך: $a^n = a^{\frac{n}{2}} \cdot a^{\frac{n}{2}}$

אם נפרק שוב: סה"כ $\log_2 n$ הכפלות .... $a^1 = a$, $a^2 = a \cdot a$, $a^4 = a^2 \cdot a^2$

כאשר n אינו חזקה של 2, נחשב את המכפלה ע"י פרוק החזקה לחזקות של 2.

לכן נקבל את התוצאה בלכל היותר ב- $\lceil \log_2(n) \rceil$ מכפלות (או קריאות רקורסיביות) בעל רקורסיה, שמכיון $2^{\lceil \log_2 n \rceil} \geq n$ קודקודים).

ובאופן דומה גם בכפל מטריצות:

נרצה לחשב את $L^{(n-1)}$ הסופי, אך לא את כל $L^{(m)}$, ונוכל להמנע מכך ע"י:

$L^{(1)} = W$, $L^{(2)} = W \cdot W$, $L^{(4)} = W^4 = W^2 \cdot W^2$

וכך זה ממשיך באופן חזרתי, העלאה בריבוע בעזרת רקורסיה עבור $n-1$.

סה"כ אם נמשיך נקבל:

$$L^{(2^{\lceil\log(n-1)\rceil})} = W^{(2^{\lceil\log(n-1)\rceil})} = W^{(2^{\lceil\log(n-1)\rceil}-1)}.W^{(2^{\lceil\log(n-1)\rceil}-1)}$$

מכיוון ש-$2^{\lceil\log_2(n-1)\rceil} \geq n-1$ ההכפלה האחרונה תהיה שווה ל-$L^{(n-1)}$ כאשר "ידרשו רק $\lceil\log_2(n-1)\rceil$ איטרציות.

וכעת נוכל להשיג אלגוריתם יעיל יותר, Faster-All-Shortest-Paths.
כאשר לולאת ה-while תתבצע $\log|V|$ פעמים במקום $|V|$.

```
Faster-All-Shortest-Paths(W)
   n = W.rows
   L(1) = W

   while m=n-1 do
        L(2m) = Extend-Shortest-Paths(L(m), L(m))
        m = 2m
return L(m)
```

ובעצם החלפנו את:

```
for m = 2 to n-1 do:
      L(m) = Extend-Shortest-Paths(L(m-1), W)
```

ב-

```
while m=n-1 do
   L(2m) = Extend-Shortest-Paths(L(m), L(m))
   m=2m
```

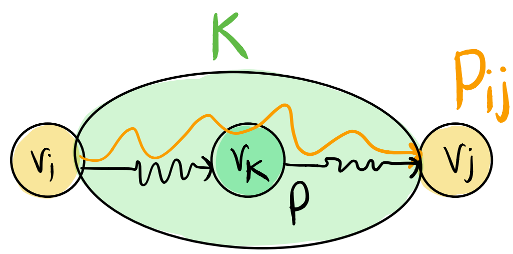
ונקבל שעלות הסיבוכיות היא: $\Theta(|V|^3 \cdot \log|V|)$ - עדיין מאוסה...

אלגוריתם פלויד - ורשל | Floyd-Warshall Algorithm

מניחים שאין מעגלים שליליים, האלגוריתם רץ ב-$\Theta(|V|^3)$.
משתמש בתכונות של קודקודי ביניים (ניצול של העובדה שתתי מסלולים הם קצרים ביותר)

הגדרה: קודקוד ביניים v_k, במסלול $p=<v_i,...,v_j>$ הוא כל קודקוד במסלול מלבד
$v_k \neq v_i$ ו- $v_k \neq v_j$.
לא חייב לכלול את v_i v_{j-1}

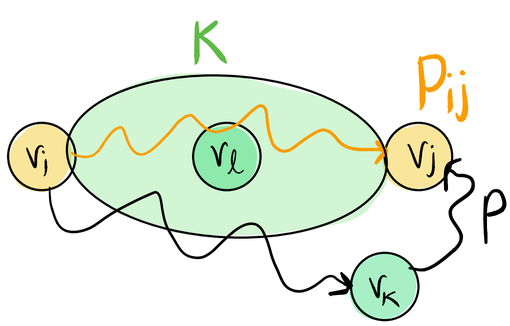
תהי $K=\{v_1,v_2,...,v_k\}$ תת-הקבוצה של $V=\{v_1,v_2,...,v_n\}$ כאשר $1\le k\le n$.
לכל זוג קודקודים v_i,v_j ב-V, נשקול את כל המסלולים מ-v_i ל-v_j עם קודקודי ביניים מ-K.
נגדיר את p_{ij} כמסלול הקצר ביותר (משקלית) בין v_i ל-v_j.



וישנם שני מקרים:

1. v_k הוא לא קודקוד ביניים במסלול p_{ij}

כל קודקודי המסלול p_{ij} הם בקבוצה $K=\{v_1,v_2,...,v_{k-1}\}$

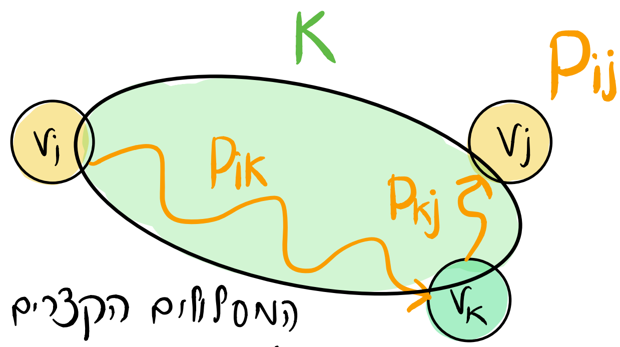


המסלול הקצר ביותר מ-v_i ל-v_j עם כל קודקודי K
כקודקודי ביניים הוא גם המסלול הקצר ביותר עם
קודקודי $K\cup\{v_k\}$ כקודקודי ביניים.
טומר ההוספה של v_k לא תועלת.

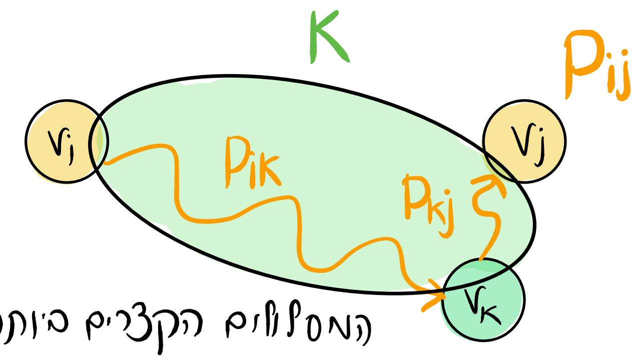
2. v_k הוא קודקוד ביניים במסלול p_{ij}

נפצל את p_{ij} לשני חלקים: ① p_{ik} מ-v_i ל-v_k ② p_{kj} מ-v_k ל-v_j } שני המסלולים הם הקצרים ביותר.

ובשני המסלולים קודקודי $K=\{v_1,v_2,...,v_k\}$ הם קודקודי ביניים.



המסלולים הקצרים ביותר עם $v_1,..,v_k$



המסלולים הקצרים ביותר עם $v_1,..,v_{k-1}$

האלגוריתם:

```
FLOYD-WARSHALL(W)
n ← rows[W]
D^(0) ← W
for k ← 1 to n
    do for i ← 1 to n
        do for j ← 1 to n
            d_ij^(k) ← min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
return D^(n)
```

* L מכילה את המסלול הקל ביותר שיש לו m צלעות (באלגוריתם הקודם).
* D תכיל את המסלול הקל ביותר שעובר דרך קודקודים $v_1,...,v_n$

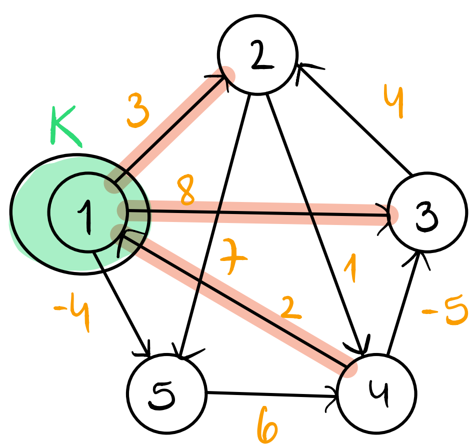
חסכנו לולאה אחת ולכן נקבל $\Theta(|V^3|)$

באלגוריתם הקודם כל פעם הוספנו צלע, והפעם אנחנו בודקים האם הוספת קודקוד תשפר את המסלול, ועוד כמה קודקודים (כלומר v_1, ועוד v_1,v_2 וכן הלאה).

דוגמא:

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

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



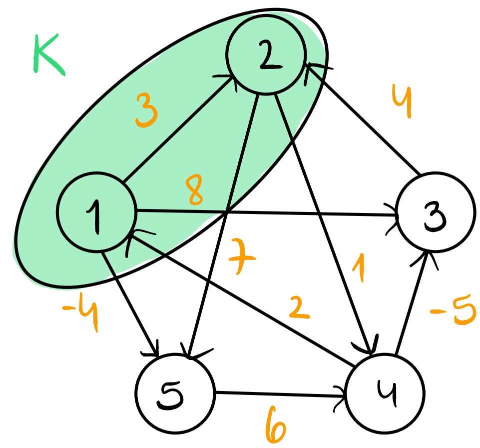
$K=\{1\}$

שיפורים:

(4,2), (4,5)

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

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



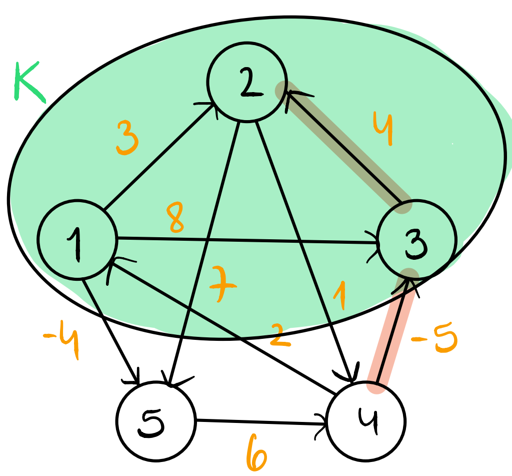
$K=\{1,2\}$

שיפורים:

(1,4), (3,4), (3,5)

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

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



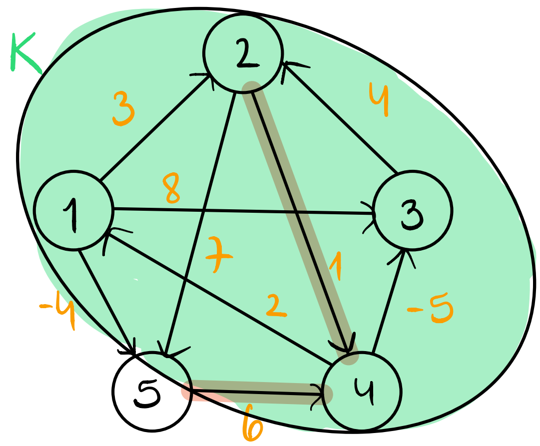
$K=\{1,2,3\}$

שיפורים:

(4,2)

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

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



$K=\{1,2,3,4\}$

שיפורים:

(1,3),(1,4),(2,1)

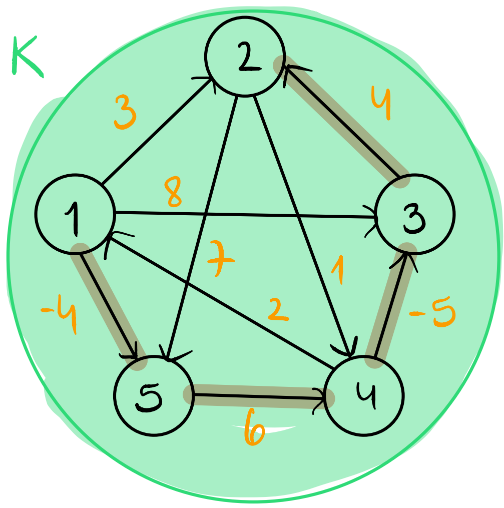
(3,1),(3,5),(5,1)

(5,2),(5,3)

הוספת 4 עשתה הרבה שינויים.

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

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



$K=\{1,2,3,4,5\}$

שיפורים:

(1,2),(1,3),(1,4)

13.1.21

הרצאה 13 - Data Structures
איחוד קבוצות זרות ואיחוד-חיפוש
Disjoint Sets Union-Find

מוטיבציה:

בעיה: אחסקת קבוצות של n אלמנטים שונים $X=\{x_1,\dots,x_n\}$ כך ש-$x_i\neq x_j$ עתיק אוסף של k קבוצות זרות $S=\{S_1,\dots,S_k\}$ כך ש- $S_i\cap S_j=\emptyset$ לכל $i\neq j$.

פעולות חשובות:

יצירת קבוצה, Make-Set (x): יצירת קבוצה חדשה S_x שהחבר היחיד בה הוא x. תוך הנחה ש-x אינו בשום קבוצה בשלב זה.

מציאת נציג, Find-Set (x): החזרת הנציג של הקבוצה S_x שמכילה את x.

איחוד קבוצות, Union (x_i,x_j): החלפת שתי קבוצות זרות S_i ו-S_j שמיוצגים ע"י x ו-y ע"י קבוצה חדשה שהיא איחודם.

עבור n אלמנטים נבצע n Make-Set מתוך סה"כ אפשרויות m פעולות על הקבוצות (Union, Find) ולכן $m>n$.

מטרה: הגדרת מבנה נתונים שיתמוך בפעולות הנ"ל על קבוצות זרות בצורה יעילה.

פתרון פשוט: עבור n אלמנטים, כאשר נבצע Union נשתמש ב-Find-set, אם נבצע m פעולות על קבוצות ובמקרה הגרוע קבוצה תכיל n אלמנטים נקבל סיבוכיות של $O(m\cdot n)$ מסה"כ.

m - כמות ה-Find ו-Union שנבצע
n - כמות האלמנטים

נרצה להשיג זמן לינארי שקרוב ל- $O(m+n)$.

דוגמאות לשימושי Union-Find?

שני אלגוריתמי גרפים משתמשים בפעולות Union-Find:

1. מציאת רכיבי קשירות - Connected Components
2. אלגוריתם קרוסקל - Minimum Spanning Tree

נגדיר עבור סיבוכיות זמן הריצה:
○-איטרציות
○-עבודה של הפעולות

עבור כל שני אלגוריתמים אלו:

1. Connected Components

משתמשים ב-Find-Set, Make-Set וב-Union.

__קלט:__ גרף לא מכוון לא ממושקל G. __פלט:__ קבוצות רכיבי הקשירות של G.

__האלגוריתם:__

```
CONNECTED-COMPONENTS(G)
for each vertex v ∈ G.V
    MAKE-SET(v)
for each edge (u, v) ∈ G.E
    if FIND-SET(u) ≠ FIND-SET(v)
        UNION(u, v)
```

$MAKE\text{-}SET(v) \rightarrow O(1)$

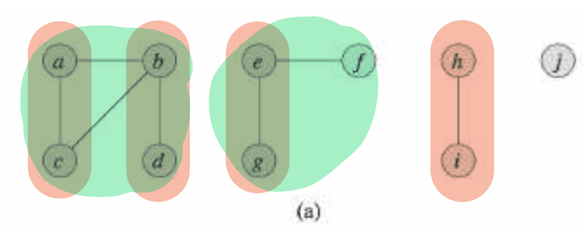
$O(\underbrace{|V|}_{\text{איטרציות}} \cdot \underbrace{1}_{\text{סיבוכיות Make-Set}}) = O(|V|)$

אם הנציגים שונים, נעשה union

$O(\underbrace{|E|}_{\text{איטרציות}} \cdot \underbrace{f(|V|)}_{\text{סיבוכיות של Find-Set ו-Union}})$

לכן סה"כ נקבל: $O(|V|+|E|\cdot f(|V|))$

__דוגמה:__



(a)

Edge processed	Collection of disjoint sets									
initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

(b)

נגדיר עבור הסיבוכיות:
- מיון
- איטרציות
- עבודה של הפעולות

2. Minimum Spanning Tree (אלגוריתם קרוסקל)

משתמשים ב-Find-Set וב-Union.

<u>קלט:</u> גרף לא מכוון ממושקל G.

<u>פלט:</u> עץ T שמחברות כל קודקודי G ע"י סכום משקלי צלעות מינימלי.

```
MST-KRUSKAL(G, w)
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)   //שני הקודקודים בקבוצות שונות
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

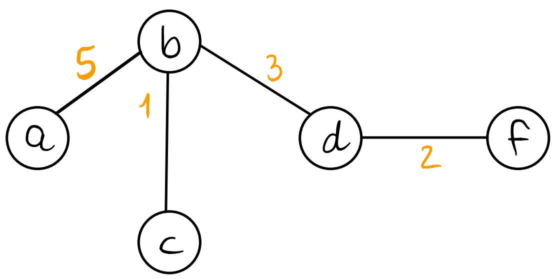
שורות 2–3: $O(|V|\cdot 1)=O(|V|)$

שורה 4: $\rightarrow O(|E|\cdot\log|E|)$

שורות 5–8: $O(|E|\cdot f(|V|))$

אכן בסה"כ נקבל $O(|V|+|E|\log|E|+|E|\cdot f(|V|))$

<u>דוגמה:</u>



$S=\{\{a\},\{b\},\{c\},\{d\},\{f\}\}$

$E=\{(b,c,1),(d,f,2),(b,d,3),(c,d,4),$
$(a,b,5),(a,c,6),(a,d,7),(a,f,10)\}$

$A\cup(b,c,1)\rightarrow S=\{\{b,c\},\{a\},\{d\},\{f\}\}$

$A\cup(d,f,2)\rightarrow S=\{\{b,c\},\{a\},\{d,f\}\}$

$A\cup(b,d,3)\rightarrow S=\{\{b,c,d,f\},\{a\}\}$

$A\cup(c,d,4)\rightarrow S=\{\{a,b,c,d,f\}\}$

ומכאן הוספת כל צלע תשאיר את הקבוצה כמו שהיא.

סיבוכיות זמן ריצה עבור שני האלגוריתמים

* פעולת Make-Set (x) לוקחת $\Theta(1)$ לכן עבור |V| קודקודים סה"כ $\Theta(|V|)$.
חח

* סיבוכיות המעבר על הצלעות, כאשר נעבור על m צלעות, תלויה במימוש פעולות Find-Set (x) ו- Union (x_i, x_j).

* כל פעולה תקח $O(1)$ עבור i איברים בכל קבוצה ולכן נקבל סה"כ $O(m \cdot n)$.

האיבר הדומיננטי בסיבוכיות הוא $O(|E| \cdot f(|V|))$.

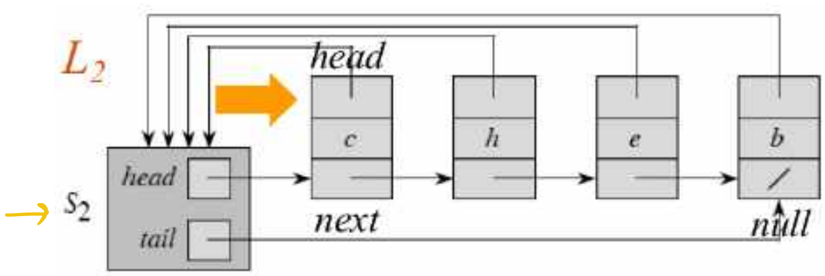
אנו רוצים לייצר מבנה נתונים שהפעולות Find-Set, Make-Set ו- Union ירוצו בפחות מ- $O(m \cdot n)$, נרצה להגיע קרוב ל- $O(m+n)$.

יש לנו שתי פתרונות: 1) רשימות מקושרות - נקבל $O(m+n\log n)$
2) יערות עצים של קבוצות זרות - נקבל $O(m \cdot \alpha(n))$
פונ' שגדלה בקצב מאוד איטי

1) רשימות מקושרות

* כל קבוצה תוצג באמצעות רשימה מקושרת L_i של אלמנטים.
* ברשימה יש שתי שדות: 1. מצביע לראש הרשימה $L_i \leftarrow$ Head
2. מצביע לסוף הרשימה $L_i \leftarrow$ Tail

* הנציג של L_i הוא בראש הרשימה.
* לאלמנטים ב- L_i ישנם 3 שדות:
1. ערך value
2. מצביע לאלמנט הבא next
3. מצביע לראש הרשימה head



פעולות: Make-Set(x) - יצירת רשימה חדשה - $O(1)$.

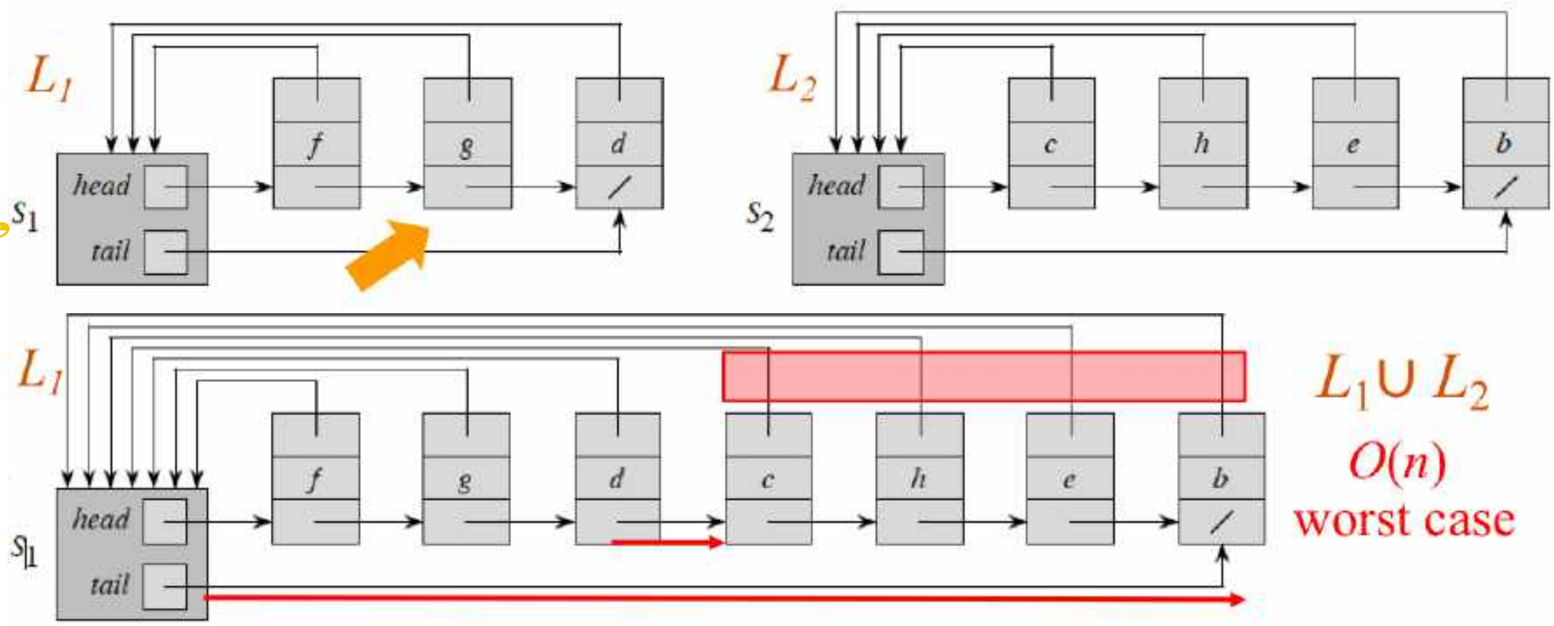
Find-Set(x) - חיפוש האלמנט במורד הרשימה במקרה הגרוע - $O(n)$.

Union(x_i, x_j) - חיבור הזנב של L_1 לראש הרשימה L_2,

לוקח יותר זמן

ועדכון שדה המצביע לראש הרשימה של L_1, עבור כל אחד מאיברי L_2.

לכן במקרה הגרוע (בגלל עדכון השדות) נקבל $O(n)$.



בסה"כ נקבל $\Theta(n^2)$ מכיוון שנוצר של n פעולות Make-Set(x)

שמלוות ב- n-1 פעולות Union יקחו סה"כ $n+\sum_{i=1}^{n-1} i = \Theta(n^2)$

(כאשר i מייצג פעולת Union שלוקחת בממוצע $\Theta(n)$).

דוגמה: סדרה של m פעולות שצורכת זמן של $O(m^2)$.

Operation	Number of objects updated
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_2, x_1)	1
UNION(x_3, x_2)	2
UNION(x_4, x_3)	3
⋮	⋮
UNION(x_n, x_{n-1})	$n-1$

נחלק את סך הפעולות בכמות הפעולות:

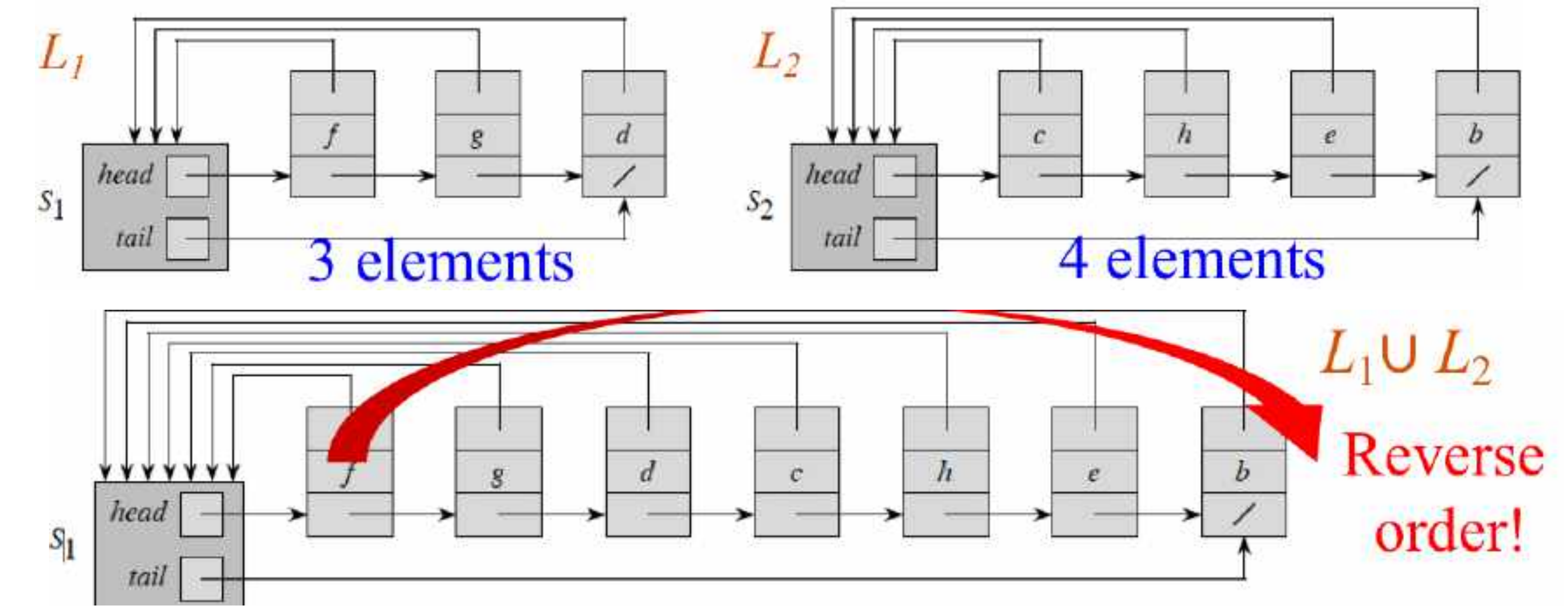
$$\frac{\Theta(n^2)}{2n-1} = \Theta(n)$$

ובכך נקבל $\Theta(n)$ זמן לפעולה בממוצע במימוש זה.

ולכן "הזמן לשיעורין" של פעולה הוא $\Theta(n)$.

יוריסטיקת האיחוד הממושקל

* כאשר נאחד שתי רשימות, נוסיף את הרשימה ה<u>קצרה</u> לרשימה הארוכה ובכך נחסוך בכמות המצביעים שנצטרך לעדכן.
* נוסיף שדה שאורך הרשימה אצל ה נציג ״יתעדכן בכל Union.
* פעולת Union אחת עדיין יכולה לקחת $\Omega(n)$ אם בשתי הקבוצות $\Omega(n)$ איברים.



מיד נראה שהשינוי שהתבצע באמצעות יוריסטיקה זו, והפכה את הסיבוכיות ל- $O(m+n\log n)$ כאשר ישנם m פעולות של Union, Make-Set ו-Find כאשר מתוכן n פעולות הן Make-Set.

<u>משפט</u>: כאשר נשתמש בייצוג של רשימות מקושרות לאיחוד קבוצות זרות, וביוריסטיקת האיחוד הממושקל, סדרה של m פעולות של Union, Make-Set ו-Find כאשר מתוכן n פעולות הן Make-Set, מתבצעת ב- $O(m+n\log n)$.

<u>אינטואיציה להוכחה</u>: אלמנט x מסוים יכול לעדכנות המצביע שלו לכל היותר $\lceil \log n \rceil$ פעמים, כי לא ייתכן שתמיד ״ימצא ברשימה הקצרה (זו שמעדכנים את המצביעים שלה בכל פעם).

הוכחה: נראה ש- $\lceil \log n \rceil$ הוא חסם עליון עבור כמות עדכוני המצביע של אלמנט x נתון כלשהו.
אנו יודעים שבכל פעם ש- x מתעדכן, הוא השתייך לפני האיחוד לרשימה הקצרה יותר (לפי היוריסטיקה - ה"טריק").
לכן, בפעם הראשונה שמתעדכן המצביע של x, האיחוד הניב קבוצה של לפחות 2 איברים.
בדומה, בפעם השנייה שמתעדכן המצביע של x, האיחוד הניב קבוצה של לפחות 4 איברים.

הערה:
הסיבה לכך היא שמספר האיברים ברשימה ש-x היה בה צריכה להיות קטנה או שווה לרשימה השנייה על מנת שיתבצע העדכון.

נמשיך באותו אופן עבור כל $k \leq n$ נקבל שבפעם ה- $\lceil \log k \rceil$ שמתעדכן המצביע נקבל קבוצה שמכילה $k = 2^{\lceil \log_2 k \rceil}$ איברים לפחות.
הקבוצה הגדולה ביותר שתתקבל ע"י איחוד תהיה בגודל n ולכן נוכל להסיק שהתבצע העדכון $\lceil \log n \rceil$ פעמים לכל היותר.

הזמן הכולל שצריך לעדכונים של n האיברים הוא $O(n\log n)$ כי הרי שלאיבר בודד העדכון הוא $O(\lceil \log n \rceil)$ ועדכון מצביעי ה-tail ואורכי הרשימות הם $O(1)$.

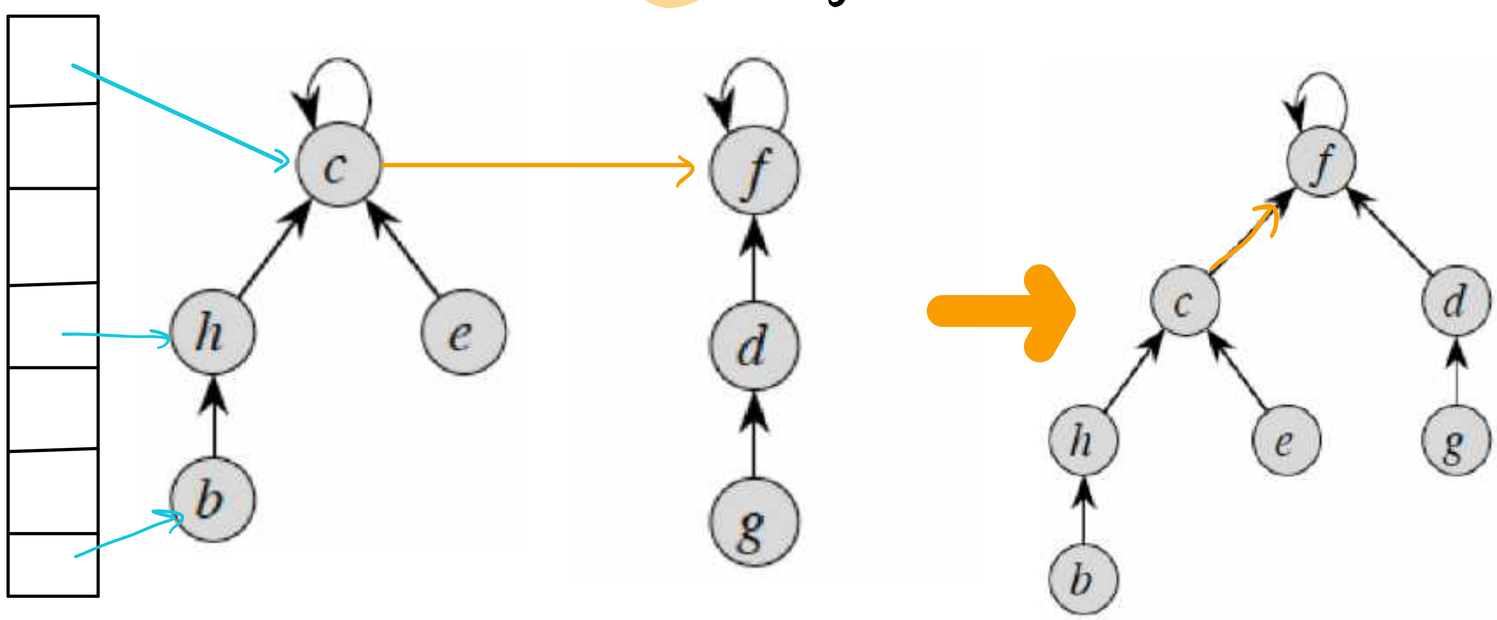
הזמן לביצוע סדרת m הפעולות: כל פעולת Union ו-Find-Set לוקחת $O(1)$ וישנן $O(m)$ כאלה ולכן סה"כ נקבל: $O(m + n\log n)$, כנדרש.

2) יערות של קבוצות זרות

* כל קבוצה היא עץ שמיוצג ע"י השורש שלו.
* כל אלמנט מצביע לאב שלו בעץ.
* השורש מצביע לעצמו, כלומר, אנו יוצרים מעגל בסופה (ושמש אותנו ב-Find וב-Union).
* ישנו מערך A של מצביעי הקודקודים כך שנוכל להגיע אליהם ב-$O(1)$.
* איחוד: נקח עץ אחד ונהפוך אותו לתת עץ של השני.

A

$S_1 = \{c,h,e,b\}$ $S_2 = \{f,g,d\}$



h = עומק העץ

פעולות: Make-Set(x) - יצירת השורש - $O(1)$.

Find-Set(x) - מעבר על המצביעים עד שמגיעים לשורש - $O(h)$.

Union(x_i,x_j) - נמצא את שני השורשים ב-$O(h)$ ואז נבחר מי מהשניים יצביע לשני ב-$O(1)$.

סיבוכיות זמן ריצה:

סיבוכיות זמן הריצה תלויה בסדרת הפעולות שיוך העצים מתיחלקים.
במקרה הגרוע נקבל $O(m\cdot n)$ כי h יכול לגדול עד ש- $h=n$.
⇐ לכן נרצה לשמור את העצים עם h מינימלי, כדי לקבל סיבוכיות טובה יותר.

<u>שתי יוריסטיקות:</u> 1. איחוד על פי דרגה: נרצה לצרף את העץ הנמוך לגבוה העץ החדש יהיה ה-h המקסימלי $+1$.
2. דחיסת מסלולים: נצרף את כל המצביעים ישירות לשורש.

עבור כל שתיהן:

1. איחוד על פי דרגה

* נקבע את השורש של העץ הנמוך יותר להיות הילד של העץ הגבוה יותר בדיוק כמו שעשינו ברשימות.
* נגדיר את שדה ה-rank של קודקוד כחסם עליון על גובה של הקדקוד.
* באיחוד, נעדכן את ה-rank של השורש.

סיבוכיות Find-Set

<u>טענה:</u> הגובה המקסימלי של עץ כאשר משתמשים באיחוד עפ"י דרגה יהיה $O(\log n)$.

<u>הוכחה:</u> באינדוקציה על מספר פעולות ה-Union שהתבצעו על מנת ליצור את העץ.
כאשר גובה העץ הוא h, מספר הקודקודים הוא לכל הפחות 2^h.
<u>בסיס:</u> עבור $h=1$, פעולת Union אחת, בעץ יהיו 2 קדקודים.
<u>צעד האינדוקציה:</u> נניח נכונות עבור עצים בגובה h, נוכיח עבור $h+1$.

נפצל לשני מקרים עבור פעולת Union הבאה:

1) גובה העץ לא גדל - אחד העצים היה קצר יותר, לכן מס' הקודקודים גדל
ו- h נשאר כמו שהוא.

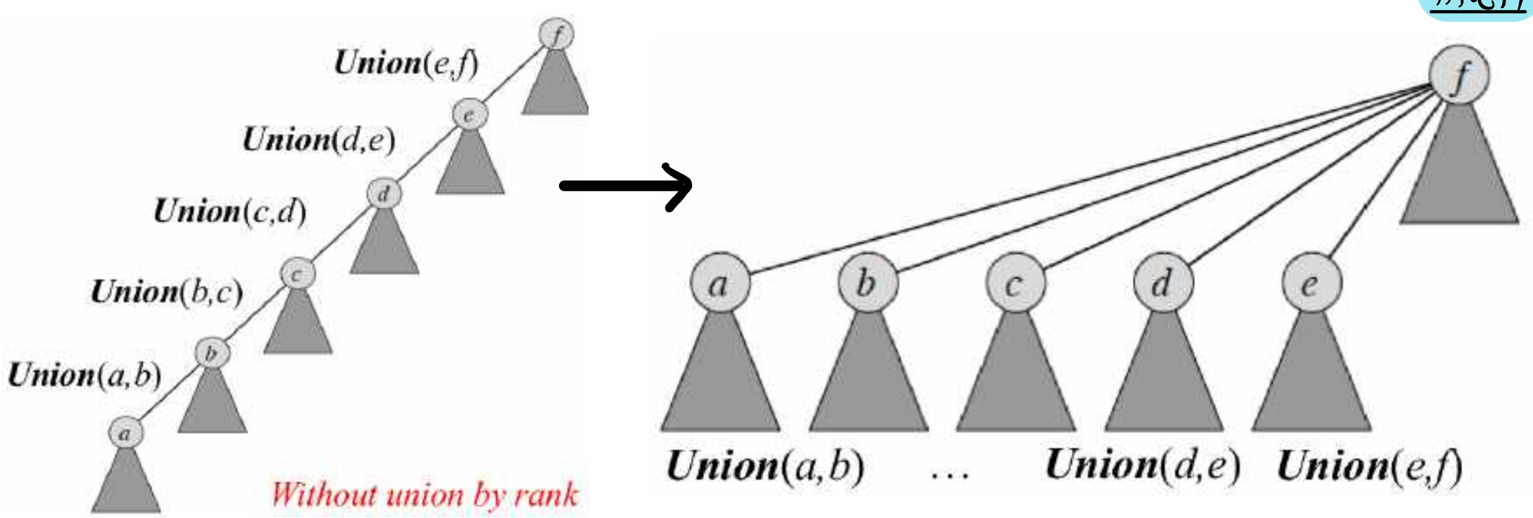
2) גובה העץ גדל - אז שני העצים היו באותו גודל ולכן לפי ההנחה, כל
תת עץ בעל 2^h קודקודים לפחות אז לעץ החדש יש לפחות $2^{h+1} = 2\cdot 2^h$ קודקודים.
כיוון שגובה העץ עלה גם הוא ב-1 נקבל את הנדרש.

מכיוון שאנו מבצעים סה"כ m פעולות כאלו, נקבל סה"כ שבאיחוד עפ"י דרגה
זמן הריצה הכולל יהיה $O(m\log n)$ במקום $O(m\cdot n)$ שהיה לנו קודם.

2. דחיסת מסלולים

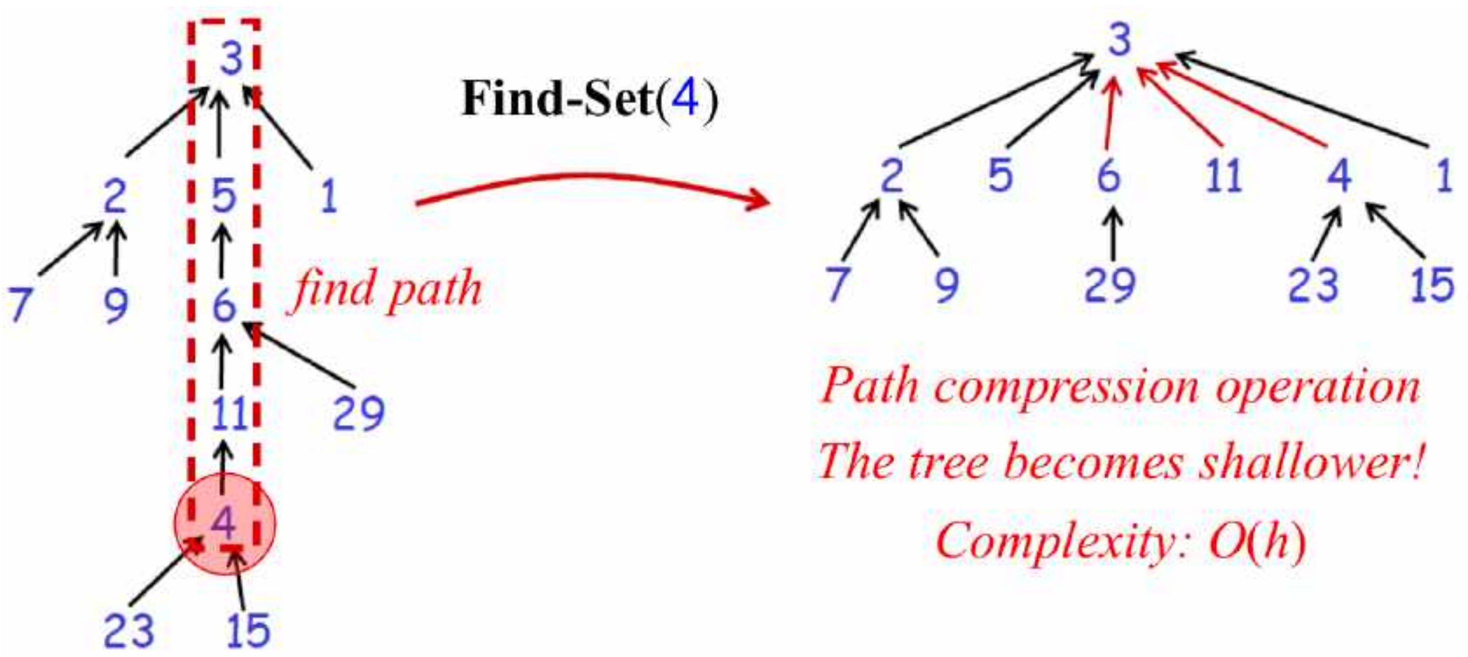
*במקום לחבר את האיברים אחד לשני, ננסה לחבר את העץ החדש ישירות לשורש
ובכך לא נצטרך לעדכן את ה-ranks של תתי העצים, אלא רק של השורש.

דוגמה:



* מה שיגדיל את הסיבוכיות במקרה הזה של איחוד קבוצות הוא בדיקת הקבוצה בה כל איבר נמצא (Find-set)

* הפתרון לכך יהיה שבמהלך חיפוש הקבוצה נבצע דחיסת מסלול ובמעבר על הקודקודים, נאחד אותם רקורסיבית אל השורש.



אלגוריתמי הפעולות עבור יערות של קבוצות זרות

```
MAKE-SET(x)
x.p = x
x.rank = 0
```

```
UNION(x, y)
LINK(FIND-SET(x), FIND-SET(y))
```

```
LINK(x, y)
if x.rank > y.rank
    y.p = x
else x.p = y
    if x.rank == y.rank
        y.rank = y.rank + 1
```

```
FIND-SET(x)
if x ≠ x.p
    x.p = FIND-SET(x.p)
return x.p
```

טענה: זמן הריצה הגרוע ביותר של n פעולות Make-Set ו- $c\le m$ פעולות Find-Set הוא: $\Theta(n+c(\log_{2+\frac{c}{n}}(n)))$

הוכחה: ללא הוכחת הטענה בקורס.

מסקנה: הסיבוכיות של דחיסת מסלולים היא $O(m+m\log n)$

כאשר נשלב בין שתי השיטות הנ"ל נוכל לקבל לשפור הסיבוכיות:

משפט: בשיטת יערות של קבוצות זרות, בשימוש משולב באיחוד לפי דרגה ובדחיסת מסלולים, סדרת $m>n$ פעולות של Union, Find-set ו-Make-Set כאשר n מתוכן הן Make-Set, סיבוכיות זמן הריצה תהיה: $O(m\cdot\alpha(n))$.

הסבר לאינטואיציה:

$\alpha(n)$ היא הפונקציה ההופכית לפונקציית אוקרמן.

בנוסף - $\alpha(n) < \log^*(n) = \log(\log(\log\cdots(n)\cdots))$

פונקציית $\alpha(n)$ עולה בקצב כה איטי כך שעבור n גדול מספיק $\alpha(n)<5$.

פונקציית אוקרמן $\rightarrow$

$$A(m,n)=\begin{cases} n+1 & \text{if } m=0\\ A(m-1,1) & \text{if } m>0 \text{ and } n=0\\ A(m-1,A(m,n-1)) & \text{if } m>0 \text{ and } n>0.\end{cases}$$

פונקציית α $\rightarrow$

$$\alpha(m,n)=\min\{i\ge 1 : A(i,\lfloor m/n\rfloor)\ge \log_2 n\}$$

הוכחה: לפי נכונות המשפט בקורס.

אם נחבר לאלגוריתמים שראינו קודם את מבחינת זמני ריצה נקבל:

Make-Set(x)	$O(1)$
Find-Set(x)	$O(\alpha(n))$
Link(x_i, x_j)	$O(\alpha(n))$

ולכן בסה"כ עבור m פעולות מסט לה נקבל: $O(m\cdot\alpha(n))$

נחבר לאלגוריתמים שראינו בתחילת השיעור ומבחינת זמני ריצה נקבל:

Connected-Components - $O(|V|+|E|\cdot f(|V|))$ והסקנו ש- $f(|V|)=O(\alpha(|V|))$

$\Leftarrow$

בסה"כ $O(|V|+|E|\cdot\alpha(|V|))$

Kruskal's Algorithm - $O(|V|+|E|\cdot\log(|E|)+|E|\cdot f(|V|))$

$\Leftarrow$ $O(|V|+|E|\cdot\log(|V|))$