הרצאה 8 המסלול הכולל הקצר ביותר

<u>אלגוריתם פלויד וורשל Floyd Warshall</u>

קלט: מטריצת משקלים כאשר $w(i,j)=\begin{cases}\infty, & (i,j)\notin E\\ 0, & i=j\\ w(i,j), & (i,j)\in E\end{cases}$

פלט: מטריצת משקלים הקלה ביותר

פתרון ע"י תכנות דינמי

בעיה: למצוא את המסלול הקצר ביותר בין כל 2 קודקודים בגרף

$\boldsymbol{F.W(G)}$:

$for\ k=1\ to\ |V|\ do$:

$for\ i=1\ to\ |V|\ do$:

$for\ j=1\ to\ |V| do$:

$D^k[i][j]\leftarrow min(D^{k-1}[i][j], d^{k-1}D[i][k]+d^{k-1}[k][j])$

Return D^k

<u>סיבוכיות</u>: $O(|V|^3)$

<u>אלגוריתם גונסון Johnson</u>

רעיון: בגרף **<u>דליל</u>** נרצה ליעל את זמן הריצה של $F.W$ ולכן נרצה להשתמש בדיקסטרה $|V|$ פעמיים, אולם הבעיה היא שדיקסטרה עובד אך ורק על גרף בעל צלעות ממושקלות אי שלליות ולכן נרצה למצוא פונקציה $h: V\to\mathbb{R}^+$

כך ש $w'(u,v)=w(u,v)+h(u)-h(v)$

<u>באופן שבו 2 התנאים הבאים יתקיימו:</u>

1. עבור כל צלע $w'(u,v)\geq 0$
2. $w'(p)=\delta'(v_1,v_k)\Leftrightarrow w(p)=\delta(v_1,v_k)$

הרעיון של גונסון:

1. להוסיף קודקוד מקור s וצלעות (s,v) עבור כל $v\in V$ כך ש $w(s,v)=0$
2. להריץ בלמן פורד על קודקוד המקור s
3. להגדיר $h(v)=\delta(s,v)$ ולהריץ דיקסטרה $|V|$ פעמיים כאשר כל פעם נשים קודקוד מקור אחר

<u>הוכחה של 1:</u>

<u>נשים לב כי:</u>

$$w'(u,v) + w(u,v) + h(u) - h(v) =$$

$$= w(u,v) + \delta(s,u) + \delta(s,v)$$

לפי אי שיוון המשולש ידוע כי $\delta(s,v) \leq \delta(s,u) + w(u,v)$

ולכן אם נעביר אגפים נקבל כי $w'(u,v) \geq 0$

<u>הוכחה של 2:</u>

$$w'(p) = \delta'(v_1, v_k) \Leftrightarrow w(p) = \delta(v_1, v_k)$$

עבור $w'(u,v) = w(u,v) + h(u) + h(v)$

נשים לב כי:

$$w(p) = w(v_1, v_2) + w(v_2 + v_3) + \cdots + w(v_{k-1}, v_k)$$

כעת נשים לב כי :

$$w'(p) = w'(v_1, v_2) + w'(v_2 + v_3) + \cdots + w'(v_{k-1}, v_k) =$$

$$= w(v_1, v_2) + h(v_1) - h(v_2) + w(v_2, v_3) + h(v_2) - h(v_3) + \cdots + w(v_{k-1}, v_k) + h(v_{k-1}) + h(v_k) =$$

$$w(p) + h(v_1) - h(v_k)$$

נניח בשלילה ש p הינו המסלול הקצר ביותר בין v_1 ל v_k ב w' ולכן קיים מסלול q בין v_1 ל v_k כך ש $w'(q) < w'(p)$ ולכן:

$$w'(q) = w(q) + h(v_1) - h(v_k) < w'(p) = w(p) + h(v_1) - h(v_k)$$

ולכן $w(q) < w(p)$ בסתירה לאופטליות של p

האלגוריתם של גונסון:

1. נגדיר G' גרף חדש כך ש : $\begin{matrix} V' = V \cup \{s\} \\ E' = E \cup (s,v)\ for\ all\ v \in V \end{matrix}$
2. נריץ בלמן פורד על קודקוד s
3. עבור כל $(u,v) \in E'$ קבע $w'(u,v) = w(u,v) + h(u) - h(v)$
4. עבור כל קודוד $v \in V$

4.1 הרץ דיקסטרה עבור v קודקוד מקור על מנת למצוא את $\delta'(u,v)$

4.2 קבע $\delta(u,v) = \delta'(u,v) - h(v) + h(u)$

<u>סיבוכיות:</u> $O(|V||E| + |V|(|E| + |V|)log(|V|) = O(|V|^2 log|V| + |V||E|)$

טבלת סיכום:

מציאת המסלול הקצר ביותר מקודקוד מקור לקודקוד יעד

סוג של גרף	**אלגוריתם**	**זמן ריצה**	**יתרון**
גרף לא ממושקל	BFS	$O(\|V\|+\|E\|)$	
גרף ממושקל משקל אי שלילי **צפוף**	$Dijkstra$	$O(\|V\|^2) - LL$	זמן ריצה טובה, אבל לא מטפל בגרפים בעלי משקל שלילי
גרף ממושקל משקל אי שלילי **דליל**	$Dijkstra$	$O(\|E\|log\|V\|) - Heap$	
גרף ממושקל	$Belman - Ford$	$O(\|V\|\|E\|)$	נשתמש רק אם יש משקל **שלילי**

מציאת המסלול הקצר ביותר עבור <u>כל</u> זוג קודקודים

סוג של גרף	**אלגוריתם**	**זמן ריצה**	**יתרון**
גרף לא ממושקל	BFS $\|V\|$ פעמיים	$O(\|V\|(\|V\|+\|E\|))$	
גרף ממושקל משקל אי שלילי **צפוף**	$Dijkstra$ $\|V\|$ פעמיים	$O(\|V\|^3) - LL$	זמן ריצה טובה, אבל לא מטפל בגרפים בעלי משקל שלילי
גרף ממושקל משקל אי שלילי **דליל**	$Dijkstra$	$O(\|E\|log\|V\|^2) - Heap$	
גרף ממושקל	$Belman - Ford$	$O(\|V^2\|\|E\|)$	נשתמש רק אם יש משקל שלילי
גרף ממשוקל	$Floyed\ Warshall$	$O(\|V\|^3)$	**גרף צפוף**
גרף ממושקל	$Johnson$	$O(\|V\|^2log\|V\|+\|V\|\|E\|)$	**גרף דליל**

<u>הרצאה 7 המסלול הקצר ביותר בגרף</u>

בעיה: נתון גרף לא מכוון ממושקל ונקודת מוצא ויעד

המטרה: למצוא את המסלול הקצר ביותר מנקודות המוצא אל היעד

הגדרה: נגדיר $\delta(u,v)$ להיות אורך המסלול הקצר ביותר בין נקודה u לנקודה v

המסלול הקצר ביותר מקיים **את אי שיוון המשולש** :

$x \in V$ עבור כל $\delta(u,v) \leq \delta(u,x) + \delta(x,v)$

כאשר השיוון מתקיים כאשר x נמצא במסלול $\delta(u,v)$

בגרפים בעלי מעגל שלילי נגדיר כי $\delta(u,v) = -\infty$ היות וככל שנעשה יותר סיבובים בגרפים נקבל מסלול קצר יותר.

<u>הקלה *Relaxation*</u>

הקלה על קודקוד $v \in V$ היא בדיקה האם ניתן לשפר את הערך $d[v]$ שלו בהינתן קשת וקודקוד, לדוגמא:

$\boldsymbol{Relax(u,v,w)}$

$if\ \big(\ d[v] > d[u] + w(u,v)\big)\ then:$

$d[v] = d[u] + w(u,v)$

<u>Dijkstra's Algorithm</u>

קלט: גרף ממושקל <u>**אי שלילי**</u>

- איפוס כל ערכי d לאינסוף ואת ערכי π ל $null$
- איפוס ערך ה d של המקור ל-0
- עבור הקודקוד בעל ערך ה d **המינמלי** שעוד **לא** נבחר:
בצע הקלה על כל **השכנים** של הקודקוד שלא נבחרו

פסאדו קוד:

$\boldsymbol{Dijkstra(G,s)}$

$for\ each\ v \in V$

$d[v] = \infty, \pi[v] = null$ — $O(V)$

$d[s] = 0\ , S = \emptyset, Q = V, \pi[s] = null$

$while\ Q \neq \emptyset$ — $O(V)$

$\quad u \leftarrow ExtractMin(Q)$

$\quad S = S \cup \{u\}$ — $O(logV)$

$\quad for\ each\ v \in Adj[u]$ — $O(E)$

$\quad\quad Relaxation(u,v,w(u,v))$ — $O(logV)\ Dec-key$

סיבוכיות: $O\big(V + V(\log(v)) + E\log(v)\big) = O((V+E)logV)$ **במימוש ערמה**

$O(V^2+E)$ **במימוש רשימה מקושרת**

ולכן במידה והגרף צפוף נעדיף מימוש ע"י רשימה מקושרת ואילו אם הגרף דליל נעשה שימוש בערמה

הערה לגבי הסיבוכיות: נשים לב כי הסיבוכיות של מימוש **בערמה** היא $O(|E|\log|V|)$ היות ובגלל שאנחנו רצים על כל הקודקודים בלולאה החיצונים ואז בפנימית נרוץ על כל השכנים נקבל כי אנחנו מבצעים $\deg(v_1)+\deg(v_2)+\cdots+\deg(v_n)=O(|E|)$ ולכן סה"כ $O(|V|\log|V|+|E|\log|V|)$

__**Bellman - Ford Algorithm**__

קלט: גרף ממושקל

- איפוס כל ערכי d לאינסוף ואת ערכי π ל $null$
- איפוס ערך ה d של המקור ל-0
- חזור $|V|-1$ פעמיים על הפעולה:

ביצוע הקלה על **כל** הקשתות בגרף

- בשביל לוודא שאין מעגלים שליליים:
  בדוק אם ניתן לבצע הקלה על אחת הקשתות, אם כן ← אזי יש מעגל שלילי.

פסאדו קוד:

$\boldsymbol{Bellman - Ford\ (G,s)}$

$for\ each\ v\in V$

$d[v]=\infty,\pi[v]=null$

$d[s]=0$

$\boxed{O(V)}$

$for\ i=1\ to\ |V|-1$

$\quad for\ each\ (u,v)\in E$ $\boxed{O(E)}$

$\quad\quad Relaxation(u,v,w(u,v))$ $\boxed{O(1)}$

$\boxed{O(V)}$

$for\ each\ v\in Adj[u]$

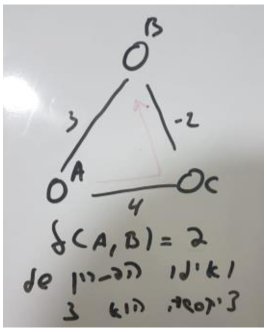
$\quad\quad \text{if}(\ \text{d[v]}>d[u]+w(u,v))$

$\quad\quad\quad \text{return "}No\ Solution\text{"}$

סיבוכיות: $O(|V|\cdot|E|)$

בגלל שבלמדן פורד הוא לא חמדני, נעדכן את כל המסלולים בגרף בכל איטרציה של הלולאה

נשים לב כי בגלל שהגרף לא חסר מעגלים נצטרך לרוץ $|V|-1$ פעמיים כי לא בהכרח נרוץ על המסלול הנכון, אולם אם הגרף הוא __חסר__ מעגלים ניתן לעשות עליו מיון טופולוגי ב $O(|V|+|E|)$ ולאחר מכן לרוץ לפי הסדר הנכון (על הצלעות) ולבצע הקלה ב $O(1)$ ולכן בגרף חסר מעגלים נוכל לחשב מסלול קצר ביותר בין 2 קודקודים בזמן ליניארי של גודל הקלט.



<u>שאלה:</u> למה משקולות שליילים זה בעיה עבור דיקסטרה ולא עבור בלמן-פורד

<u>פתרון:</u> קודם כל נשים לב כי דיקסטרה הוא אלגוריתם חמדני ואילו בלמן פורד הוא לא חמדני
בנוסף אנחנו משתמשים בערמת מינימום, כאשר אנחנו מוציאים קודקוד אז אנחנו מניחים שהוא במרחק המינמלי ביותר מקודקוד המקור, היות ואם נבחר קודקוד בעל מרחק גדול יותר, אז אנחנו מוסיפים מרחק אבל אם יש משקלים שליליים אז כבר לא ניתן להניח זאת היות ויכול להיות שקיים מסלול עקיף שיכול להוריד את המשקל עבור אותו קודקוד.

<u>שאלה:</u> למה צריך לחזור על פלמדן פורד בדיוק $|V| - 1$ פעמיים

<u>פתרון:</u> יש לכל היותר $|V| - 1$ צלעות במסלול הארוך הארוך ביותר מבין קודקוד כלשהו בגרף לקודקוד אחר

נשים לי האינווראיטנה הבאה נשמרת:

לאחר האיטרציה ה $k -$ של הלולאה החיצונית, כל קודקוד עם מרחק מקודקוד המקור באורך לכל היותר k צלעות מכיל בתוכו את ערך המרחק הנכון.

הרצאה 6 Minimum Spanning Trees (MST)

המטרה: עץ פורש מינמלי של הגרף

עץ פורש מינמלי – תת גרף מסוג עץ שמכיל את כל הקודקודים של הגרף, כאשר עץ פורש מינמלי הוא עץ שמבין כל העצים הפורשים של הגרף, העץ הוא בעל משקל $w(T) = \sum_{e\in T} w(e)$ מינמלי.

- Kruskal's algorithm: $O(e \log e)$
- Prim's algorithm: $O(e \log v)$

האלגוריתם של קרוסקל *Kurskal*

קלט: גרף ממושקל לא מכוון

1. הגדר $A \leftarrow \emptyset$ $\quad O(1)$
2. חלק את כל הקודקודים לקבוצות: כל קודקוד בקבוצה $\quad O(V)$
3. מיין את הקשתות לפי משקל: **מהקטן לגדול** $\quad O(ElogE)$
4. עבור על כל קשת לפי המשקל הקטן לגדול $\quad O(E)$

4.1 אם הקשת המחברת בין 2 קודקודים מקבוצות שונות:

4.1.1 אחד את קבוצת הקודקודים

4.1.2 הוסף את הקשת לקבוצה A

4.2 אם A מכילה $|V|$ קודקודים – סיים את האלגוריתם

$Union(x,y)$

$find(x)$

$O(a(|V|))$ כאשר יש שימוש ב$Union - Find$. $a(|V|)$ הינה f^{-1} של אקרמן.

סיבוכיות האלגוריתם מושפעת בעיקר מהצורך למיין את הקשתות בתחילת הפעלתו. בגרף עם V קשתות ו E קשתות, הסיבוכיות חסומה ע"י $O(|E| \cdot \log|E|)$

האלגוריתם של פרים *Prim*

קלט: גרף ממושקל לא מכוון

1. אתחל $Q \leftarrow V$ $\quad O(|V|) - Build\ Min - Heap$
2. עבור כל קודקוד $u \in Q$ אתחל את $key(u) \leftarrow \infty$, $\pi(u) = NULL$ $\quad O(|V|)$
3. בחר קודקוד רנדומלי $s \in Q$ ואתחל את $key(s) = 0, \pi(s) = NULL$
4. כל עוד Q לא ריק $\quad O(|V|)$

4.1 $u \leftarrow DeleteMin(Q)$ $\quad O(\log|E|)$

4.2 עבור כל קודקוד שכן של $v \in Adj(u)$ $\quad O(|E|)$

4.2.1 אם $v \in Q$ ו $w(u,v) < key(v)$

4.2.1.1 $\pi[v] \leftarrow u$

4.2.1.2 $key[v] \leftarrow w(u,v)$

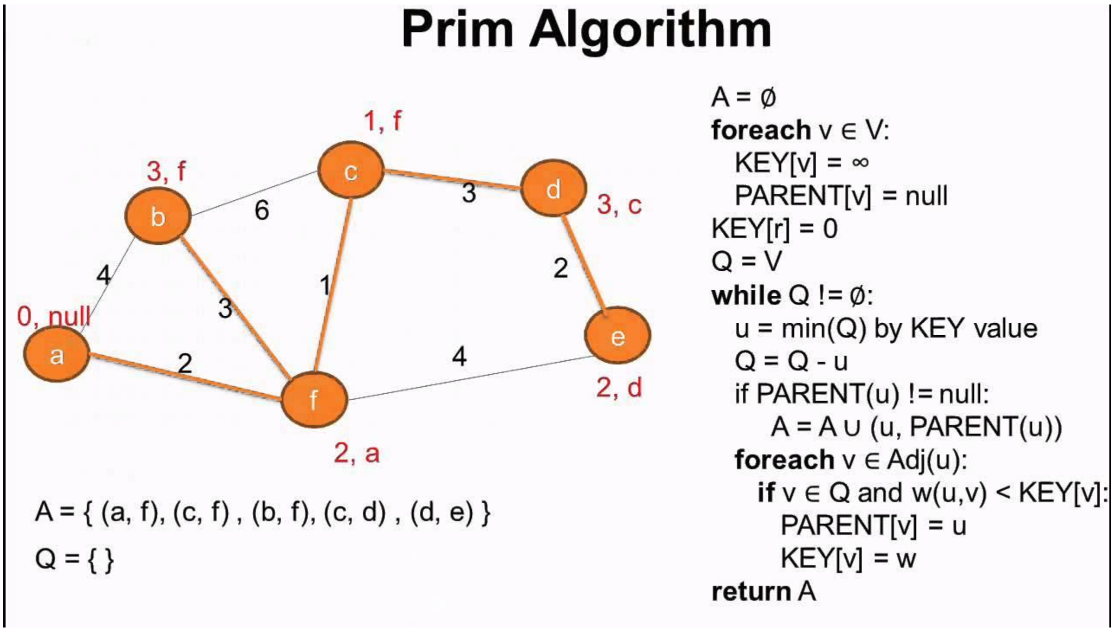
$DECREASE\_KEY$

$O(\log|V|)$

סיבוכיות האלגוריתם של פרים ללא שימוש בערמה: $O(E+V^2)$

בעת מימוש האלגוריתם נעשה שימוש בערימה שמתוכה מוציאים בכל פעם את הצלע המינימלית. אם משתמשים בערימה בינארית סיבוכיות האלגוריתם תהיה $O(ElogV + VlogV)$ (כאשר E הוא מספר הקשתות ו-V הוא מספר הקודקודים). ניתן לשפרה מעט באמצעות שימוש בערימת פיבונאצ'י ולהגיע ל-$O(E + V \log V)$.

באופן כללי היעילות של האלגוריתם של פרים טובה מזו של האלגוריתם של קרוסקל. למרות זאת, אם הקלט כבר ממויין לפי משקלי הקשתות או כאשר ניתן למיין אותם בזמן ליניארי, אזי האלגוריתם של קרוסקל יהיה מהיר יותר עם זמן ריצה של $O(E\,\alpha(E,V))$.

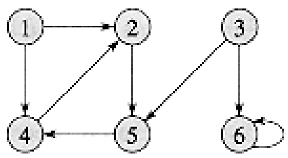


האלגוריתם של קרוסקלר עובד ע"י יערות אל תוך עץ פורש מינמלי

האלגוריתם של פרים עובד ע"י בניית עץ שבסוף נהפך להיות עץ פורש מינמלי

הרצאה 5 – $DFS, Topological\ Sort, SCC, BFS$

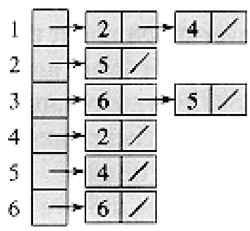
יצוג של גרפים



(a)

רשימת שכנויות

Adjacency List



מערך של רשימות מקושרות

סיבוכיות מקום: $O(V+E)$

יתרונות:

- להוסיף קודקוד קל יותר ע"י יצירת תא חדש במערך

חסרונות:

- על מנת לראות אם יש קשת בין 2 קודקודים נצטרך לרוץ על כל הרשימה המקושרת סה"כ $O(E)$

מטריצת שכניות

Adjacency Matrix

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

סיבוכיות מקום: $O(V^2)$

בגרף לא מכוון ניתן להפעיל רק על החצי העליון של המטריצה היות והמטריצה סימטרית

יתרונות:

- קל ליצוג
- ניתן לבדוק צלע ב $O(1)$
- הוספה והסרת צלע ב $O(1)$

חסרונות:

- זכרון $O(V^2)$
- להוסיף קודקוד לוקח $O(V^2)$

אם יש הוספה של קודקודים אז נעדיף

היות וצריך לעשות $resize$ למטריצה

עדיף אם יש שליפה מהזכירון בלי הוספה של קודקודים, בגרף **צפוף** נעדיף מטריצה היות ואין בזבוז של "אפסים" במטריצה

גרף צפוף – גרף שכמות הצלעות היא $\theta(V^2)$

אלגוריתם DFS – סריקה לעומק	**אלגוריתם BFS – סריקה לרוחב**
קלט: גרף מכוון / לא מכוון **שימושים:** • בדיקת קשירות של גרף נתון • מציאת המסלול העמוק ביותר **אלגוריתם:** $\boldsymbol{DFS(G)}$ 1. $for\ each\ v \in V$ 2. $\text{if}\ (\text{color}[v] = white)$ 3. $\pi[v] = null$ $\boxed{O(V)}$ 4. $time \leftarrow 0$ 5. $for\ each\ v \in V$ $\boxed{O(V)}$ 6. $if\ color[v] = white$ 7. $\text{do DFS} - \text{VISIT}(v)$ $\boldsymbol{DFS - VISIT(v)}$ $\boxed{O(E)}$ 1. $color[v] = grey$ 2. $time \leftarrow time + 1$ 3. $d[v] = time$ 4. $for\ each\ u \in Adj(v)$ 5. $if\ color[u] = white$ 6. $\pi[u] = v$ 7. $DFS - VISIT(u)$ 8. $color[v] = black$ 9. $f[u] = time = time + 1$ סיבוכיות זמן ריצה: $O(V+E)$	**קלט:** גרף מכוון / לא מכוון וקודקוד מקור s 1. אתחל תור Q 2. הכנס את s לתוך התור $color[n] = white$ 3. אתחל מערך $\pi[n] = null$ $dist[n] = \infty$ $color[s] = grey$ 4. אתחל $\pi[s] = null$ $dist[s] = 0$ 5. כל עוד Q לא ריק בצע: 6. $v \leftarrow Q.pull()$ 7. עבור כל שכן u של v 8. אם $color[u] = white$ 9. קבע $color[u] = gray$ 10. קבע $\pi[u] = v$ 11. קבע $dist[u] = dist[v] + 1$ 12. הכנס את u ל Q
ההבדל בין DFS ל BFS זה ש DFS משתמש במחסנית ואילו BFS משתמש בתור	

<u>DFS – מההרצאה של דנה</u>

<u>משפט הסוגריים:</u>

לכל $u, v \in V$ אחת התכונות הבאות מתקיימות:

1. $d(u) < f(u) < d(v) < f(v)$ או להפך כלומר $\underset{uuvv}{()()}$ – מציין שהקודקודים u, v זרים, אין יחס צאצא – אב
2. $d(u) < d(v) < f(v) < f(u)$ או להיפך, כלומר u הוא אב של v $\underset{uvvu}{(())}$

<u>משפט המסלול הלבן:</u>

v הוא צאצא של u $\Leftrightarrow$ בזמן הגילוי של u קיים מסלול $v \rightsquigarrow u$ שמורכב רק מצמתים לבנים

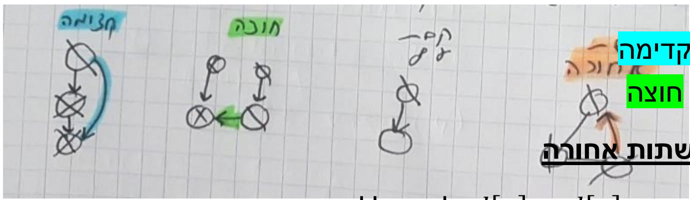
<u>סיווג קשתות בזמן ריצה עבור קשת (u, v)</u>

אם v הוא לבן אזי (u, v) היא קשת עץ

אם v אפור אזי (u, v) היא קשת אחורה

אם v שחור אזי $\begin{matrix} d[u] < d[v] \\ d[u] > d[v] \end{matrix}$ אז נקבל כי זוהי קשת $\begin{matrix} \text{קדימה} \\ \text{חוצה} \end{matrix}$

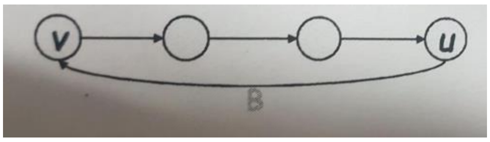
רק בגרף מכוון



<u>משפט: בגרף לא מכוון יש אך ורק קשתות עץ או קשתות אחורה</u>

<u>הוכחה</u>: יהיה $(u, v) \in E$ קשת כלשהי בגרף, בה"כ נניח כי $d[u] < d[v]$ ולכן בגלל שאנחנו בסריקת עומק יש קודם לטפל ב– v, ולכן אם הקשת (u, v) התגלתה לפני בכיוון $v \rightarrow u$ אזי v עדיין לבן ולכן זוהי קשת עץ. אם הקשת התגלתה בכיוון $u \rightarrow v$ אזי u כבר אפור ולכן זוהי קשת אחורה (מעגל)

<u>משפט: גרף G חסר מעגל $\Leftrightarrow$ אין ב G קשתות אחורה</u>



<u>הוכחה $\Leftarrow$:</u> נניח כי G חסר מעגלים ונוכיח כי ב G אין קשתות אחורה, נניח בשלילה כי בG יש קשת אחורה (u, v) ולכן v הוא צאצא של u בעץ העומק, ולכן קיים מסלול $v \rightarrow u \rightsquigarrow v$ אשר סוגרת מעגל

<u>הוכחה $\Rightarrow$:</u> נניח כי ב G אין קשתות אחורה ונוכיח כי G חסר מעגלים,

- נניח בשלילה כי ב G יש מעגל c כלשהו
- יהי v הקודקוד הראשון שהתגלה במעגל c
- בזמן הגילוי של v לפי משפט המסלול הלבן יש מסלול שמורכב רק מקודקודים לבנים במעגל c $u \rightsquigarrow v$ (ניתן לראות בציור למעלה)
- לפי משפט המסלול הלבן, u הוא צאצא של v
- ולכן (u, v) היא קשת אחורה – סתירה להנחה.

<u>מיון טופולוגי (G)</u>

1. תקרא $DFS(G)$ על מנת לחשב את זמני הסיום
2. כל קודקוד שסיים, הכנס את הקודקוד לתוך ההתחלה של הרשימה מקושרת
3. החזרת את הרשימה מקושרת (זמן הסיום הגדול ביותר יהיה הראש וזמן הסיום בקטן ביותר יהיה הזנב)

<u>סיבוכיות:</u> $O(V + E)$ היות ומשתמשים ב DFS

גרף קשיר היטב - *SCC -Strongly Connected Components*

אלגוריתם $SCC(G)$

1. תקרא ל$DFS(G)$ על מנת לחשב את כל זמני הסיום של כל קודקוד
2. חשב את G^T
3. תקרא $DFS(G^T)$ לפי סדר של זמני **סיום יורד** (מהגדול לקטן)
4. קבוצת כל הצלעות ביער העומק הוא רכיב קשירות נפרד

הרצאה 4 תכנון דינמי

רעיון: הרעיון הכללי מאחורי האלגוריתם בקטגורייה תכנון דינמי זה לפרק את הבעיה למספר תת בעיות קטן ככל שאפשר ולפתור כל תת בעיה באופן אפטמלי, וחישוב $Bottom - UP$ בעזרת מטריצה.

המטרה היא להמנע מחישובים חוזרים כמו שקורה ברקרוסיה ולכן כאשר אנחנו בונים מטריצה אנחנו מסתמכים על ערכים שכבר חישבנו בעבר ולכן נמנעים מחישובים שנעשים יותר מפעם אחת.

שיטות שלמדנו עד עכשיו:

- הפרד ומשול
- אלגוריתם חמדני
- חיפוש שלם (Brute Force)
- תכנון דינמי

בעיה 1: רצף פיבונאצי

סדרת פיבונאצי מוגדרת באופן הבא: $f_0 = 0, f_1 = 1, f_n = f_{n-1} + f_{n-2}$

נשים לב כי על מנת לחשב את f_{100} נבצע הרבה חישובים חזורים ולכן נפתור את הבעיה ע"י תכנון דינמי.

$\boldsymbol{f(n)}$

1. נאתחל $f[100]$
2. נרוץ מ2 ועד 100

נחשב $f[i] = f[i-1] + f[i-2]$

נשים לב כי אין חישובים חוזרים – **סיבוכיות** $O(n)$

בעיה 2: מקדם בינומי

מקדם בינומי מוגדר רקורסיבית באופן הבא : $\binom{n}{k} = \binom{n-1}{k} + \binom{n-1}{k-1}$

פתרון ע"י תכנון דינמי:

נבנה מטריצה $A_{(n+1)\cdot(k+1)}$ אשר תייצג את הפתרון הנכון עבור מקדם הבינום עבור (n,k)

תנאי עצירה:

לכל $A(0,j)$ כאשר $j \in [1,k+1]$ הגדר $A(0,j)=1$ היות ויש רק אפשרות אחת לבחור j אנשים מקבוצה של 0 – לא לבחור אף אחד.

לכל $A(i,0)$ כאשר $i \in [1,n+1]$ הגדר $A(i,0)=1$ היות ויש רק אפשרות אחת לבחור j אנשים מקבוצה של i – לא לבחור אף אחד.

עבור כל תא $A(i,j)$ נגדיר $A(i,j) = A(i-1,j) + A(i-1,j-1)$

<u>סיבוכיות:</u> $O(n\cdot k)$ במקום אספוננציאלית היות ו $T(n)_{req} = 2T(n-1) = \cdots = 2^k T(n-k) =$

$$=_{for\ k=n} 2^n T(1) = 2^n$$

<u>בעיה 3: הטוב מבין ה2</u>

<u>הגדרת הבעיה:</u> נניח כי יש 2 קבוצות שמשחקות, קבוצה A וקבוצה B, נרצה לדעת מי הראשון שינצח n משחקים כאשר אם קבוצה אחת ניצחה $\left\lfloor \frac{n}{2} \right\rfloor + 1$ יש לה ניצחון אוטוטי.

<u>הנחה:</u> לקבוצה A ולקבוצה B יש הסתברות זהה לניצחון כל משחק - 50%

נגדיר $P(i,j)$ להיות ההסתברות ש A תנצח כאשר A צריכה i משחקים לנח וB צריכה j משחקים לנצח.

תנאי עצירה:

$$P(0,j) = 1 : \forall j > 0$$

$$\mathrm{P(i,0)} = 0 : \forall i > 0$$

נוסחת נסיגה:

נסתכל על המשחק האחרון

$$P(i,j) = \frac{1}{2}P(i-1,j) + \frac{1}{2}p(i,j-1)$$

כפולה אריתמטית $O(1)$

סיבוכיות:

$$\begin{aligned} T(n,m) &\ge T(n-1,m) + T(n,m-1) \\ &= T(n-2,m) + T(n-1,m-1) + T(n-1,m-1) + T(n,m-2) \\ &> 2^2 T(n-2,m-2) > \cdots > 2^k T(n-k,n-k) =_{for\ k=\min(n,m)} 2^{\min(n,m)} \\ &= O\left(\pi\left(2^{\min(n,m)}\right)\right) \end{aligned}$$

<u>נפתור בעיה זו בעזרת תכנון דינמי:</u>

נבנה מטריצה $P_{(n+1)\cdot(k+1)}$ כאשר התא $P(n,m)$ תייצג את הפתרון הנכון עבור ההסתברות ש A תנצח כאשר A צריכה n משחקים לנח וB צריכה m משחקים לנצח.

תנאי עצירה:

לכל $P(0,j)$ כאשר $j\in[1,m+1]$ הגדר $P(0,j)=1$

לכל $P(i,0)$ כאשר $i\in[1,n+1]$ הגדר $P(i,0)=0$

עבור כל תא $P(i,j)$ נגדיר $P(i,j)=\frac{1}{2}P(i-1,j)+\frac{1}{2}P(i,j-1)$

סיבוכיות: $O(nm)$

<u>בעיה 4: מיקום הסוגריים האופטמלי לכפל שרשרת של מטריצות</u>

הבעיה: כפל מטריצות הוא אסוציאטיבי, כלומר ניתן לשנות את סדר הקדמיות של פעולות הכפל בינן לבין עצמן מבלי שהתוצאה תשתנה. מספר הפעולות, לעומת זאת, כן משתנה (בהנחה שלא כל המטריצות מאותו סדר גודל) והבעיה היא כיצד ניתן למצוא את מיקום הסוגריים במכפלה של n מטריצות, שבו מספר הפעולות האירתמטיות יהיה מינמלי.

ביהנתן n מטריצות $M_1,M_2,\dots,M_n$ נחשב את כמות הפעוילות הכפל המינצליות שצריך לבצע על המכפלה כאשר למטריצה ה i מתקיים שהיא מסדר $d_{i-1}\times d_i$

לדוגמא:

$$M_1=2\times 5$$

$$\mathrm{M}_2=5\times 3$$

$$\mathrm{M}_3=3\times 4$$

$(M_1\times M_2)\times M_3$ $(2\times 4)\ \sum 54$ $2\times 3\ \sum 30$	$M_1\times(M_2\times M_3)$ $(5\times 4)\ \sum 60$ $2\times 4\ \sum 100$

נגדיר $M(i,j)$ להיות מספר הפעולות האריתמטיות שיש לבצע על מנת להכפיל את המטריצות $M_iM_{i+1}\dots M_j$.

נגדיר את נוסחת הנסיגה עבור פתרון רקרוסיבי:

$$\boldsymbol{M(i,j)} = \begin{cases} \boldsymbol{0}, & \boldsymbol{i=j} \\ \boldsymbol{min_{k\in[1,j-1]}\big(\, m(i,k) + m(k+1,j) + d_{i-1}d_k d_j\,\big)}, & \boldsymbol{i<j} \end{cases}$$

בגלל שכפל מטריצות A מסדר $p*q$ ומטריצה B עם מימד $q*r$ אז המכפלה לוקחת $O(pqr)$

כי $1 \le i < p, 1 \le j \le r$:$(AB)_{i,j} = \sum_{l=1}^{q}(a_{i,l})(B_{jl})$

אנחנו מתעניין ב $M(1,n)$

סיבוכיות:

$$T(n) = \sum_{k=1}^{n-1}(T(k) + T(n-k) + 1)$$

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

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

$$T(n) - T(n-1) = 2T(n-1) + 1$$

$$\mathrm{T}(n) = 3T(n-1) + 1 > 3T(n-1) > \cdots > 3^k T(n-k) = O(\pi(3^n))$$

פתרון ע"י תכנון דינמי:

נבנה מטריצה בגודל M_{n*n} ונרצה להגיע ל$M(1,n)$

נסתכל רק על המשולש העליון של המטריצה היות וכל $j > i$ הינו $\emptyset$

על מנת לשחר איפה לשים את הסוגריים, נבנה מטריצה חדשה S כאשר כל פעם שנעדכן לתוכו את k

סיבוכיות מקום: $O(2n^2)$ **לבדוק אם נכון**

סיביכיות זמן ריצה: $O(n^3)$

הרצאה 3 – קוד הופמן

מטרה: על מנת לצמצם בשטח אחסון, ניתן קוד לכל תו בטקסט. נבדוק את השכיחות של כל תו בטקס, וכך ניתן ליצור קוד קצר יותר לתווים שכיחים יותר וקוד ארוך יותר לתויים נדירים יותר. הביצוע נעשה בעזרת בניית עץ $Prefix$ שבו כל תו **נמצא בעלה** והקוד של התו הוא המסלול מהשורש ועד אליו כאשר כל פנייה לבן שמאלי מסומן ע"י 0 ופנייה לבן ימיני מסומן ע"י 1 היות המחשב עובד בצורה בינארית.

קוד חסר רישות $\begin{matrix}\boldsymbol{prefix}\\ \boldsymbol{free}\\ \boldsymbol{code}\end{matrix}$ **:** קוד שבו אף מילת קוד אינה רישע של מילת קוד אחרת

לדוגמא רישות של מילת קוד: $0111\begin{cases}\epsilon\\ 0\\ 01\\ 011\\ 0111\end{cases}$

מיוצג ע"י עץ בינארי, כל צלע מיוצגת ב'0' או ב'1', כל עלה בעץ הינו תו כלשהו, כאשר המסלול בין שורש העץ לבית העלה זה היוצג של אותו התו, אורך המסלול מסומן ב (l_i)

הגדרה: $Uniquely\ Decipherable(UD)$ הוא קוד שניתן לפענח בצורה יחידה

יתרונות לקוד חסר רישות:

- קל לקידוד ופיענוח
- ניתן לפיענון בצורה יחודית UD
- ניתן להוכיח כי כל דחיסת קוד אופטמלית אשר ניתן ע"י קוד לא חסר רישות אזי ניתן תמיד לדחוס בצורה זהה ע"י קוד חסר רישות ולכן ניתן להתמקד בקוד חסר רישות.

כל קוד חסר רישות הוא UD ($Prefix\ free \Rightarrow$ UD)

דוגמא: בהינתן המחרוזת $abcde$

עבור מילות הקוד $\begin{matrix} a=1 \\ b=01 \\ c=001 \\ d=0001 \\ e=00001 \end{matrix}$ נקבל את הקוד הבא: $1|01|001|0001|00001$

אבל לא כל UD הוא חסר רישות ($\text{Prefix} \nLeftarrow \text{UD}$)

- **דוגמא:** בהינתן המחרוזת $abcde$
  עבור מילות הקוד: $\begin{matrix} a=1 \\ b=10 \\ c=100 \\ d=1000 \\ e=10000 \end{matrix}$, קוד לא חסר רישות אבל UD: $1|10|100|1000|10000$

כדי **להוכיח** שקוד כלשהו הוא **לא** UD יש צורך לתת מחרוזת בינארית שיש לה שתי פירושים שונים

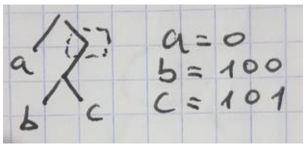
לדוגמא: עבור הקוד: $\begin{matrix} a=0 \\ b=101 \\ c=100 \\ d=111 \\ e=110 \\ f=1100 \end{matrix}$ אזי הקוד $\begin{matrix} 1100=1100=\text{"}f\text{"} \\ 1100=110+0=\text{"}ea\text{"} \end{matrix}$

- עבור דחיסה, קוד אופטמלי חייב להיות מיוצג ע"י עץ מלא

  עץ מלא: עץ שלכל צומת יש 2 בנים

  עץ שלם: עץ מלא שבו כל העלים באותו עומק

- לא כל קוד חסר רישות הוא עץ מלא אבל קוד חסר רישות **אופטמלי** הוא עץ **מלא**
  לדוגמא:



- בעץ בינארי מלא אז אם יש n עלים אז יש $n-1$ צמתים פנמיים
  הוכחה: צ"ל שכמות העלים בעץ גדולה ב1 מכמות הצמים הפנימיים, באינדוקציה על כמות הצמתים הפנמיים.

l_i - אורך היצוג בביטים
w_i – שכיחות בטקסט

קוד הופמן

הרעיון הכללי: על מנת לצמצם בשטח אחסון, כלומר נרצה להשיג $\sum_{i=0}^{\Sigma} l_i w_i$ מינמלי, ניתן קוד לכל תו בטקסט. נבדוק את השכיחות של אותו התו ובכך ניתן קוד קצר יותר המתאים לתווים שכיחים יותר וקוד ארוך יותר לתווים נדירים יותר, הביצוע נעשה בעזרת עץ חסר רישות,שבו כל תו נמצא בעלה והקוד של התו הוא המסלול מהשורש ועד לעלה כאשר כל פנייה שמאלה מסומנת ב'0' וכל פנייה ימינה מסומנת ב'1'.

<u>$Huffman\ Algo$</u>

אלגוריתם חמדני.

רעיון: להתחיל לבנות את העץ מלמטה, ובכך העלים שנמצאים ברמה הגדולה ביותר יהיו השכיחים ביותר עם l_i מקסמלי ואילו האותיות הנפוצות ביותר יהיו ברמה הנמוכה ביותר עם l_i מינמלי.

$\boldsymbol{Huffman}(\Sigma)$

$n \leftarrow |\Sigma|$ $\boxed{O(1)}$

$Q \leftarrow \Sigma$

$for\ i = 1\ to\ n-1\ do:$

$do\ new\ node\ z$

$left(z) \leftarrow x \leftarrow Extract - Min(Q)$

$right(z) \leftarrow y \leftarrow Extract - Min(Q)$

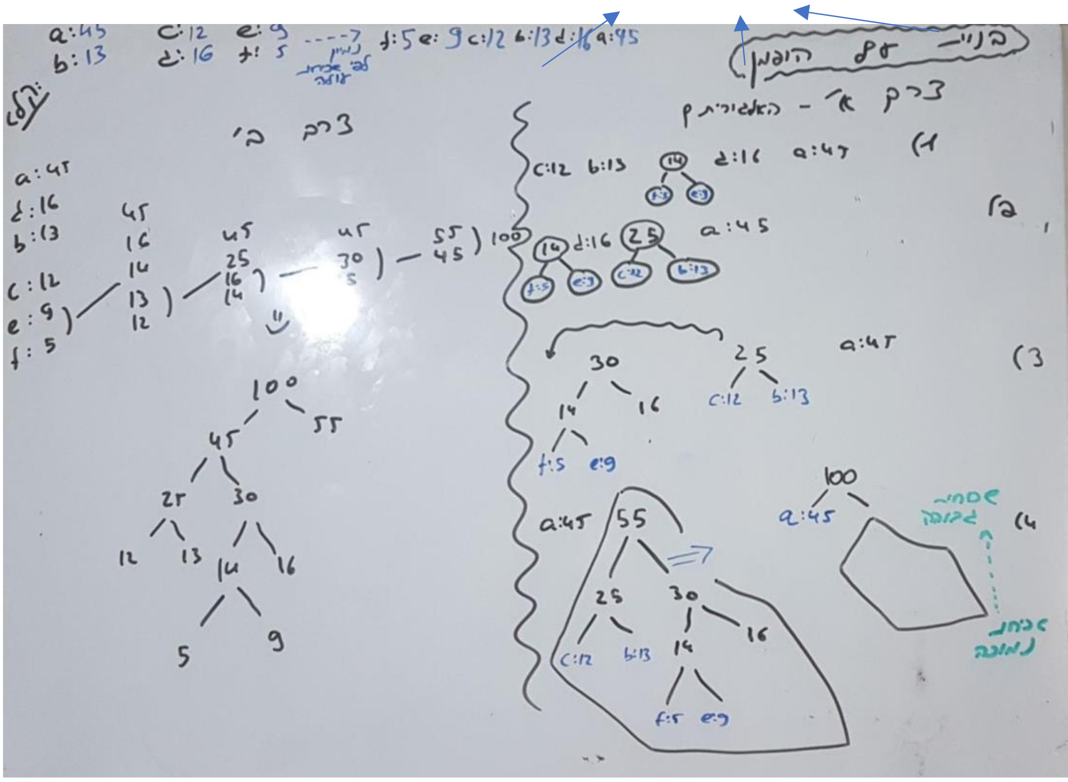
$w(z) = w(x) + w(y)$

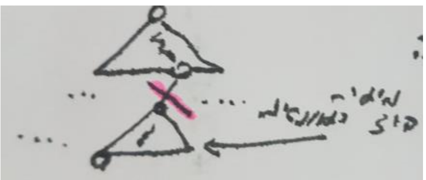
$insert(Q, z)$

$return\ Extract - Min(Q)$

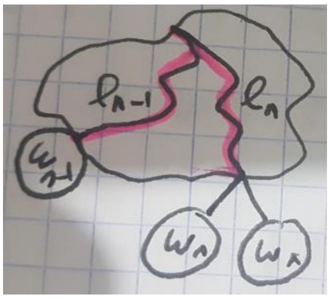
סיבוכיות: $\boldsymbol{nlog(n) + n + 1 = O(nlogn)}$

Min





למה 1 : עץ אופטמלי הוא עץ מלא
הוכחה: נניח בשלילה כי קיים עץ אופטמלי שהוא לא עץ מלא
נבנה קוד חדש כאשר נשמיט את הסיבית המתאימה לצומת עם הבן היחיד מכל המילות הרלוונטיות ונקבל קוד טוב יותר, בסתירה לאופטמליות של הקוד ההתחלתי.



למה 2 : העץ אופטמלי, שני המשקלים הנמוכים ביותר w_n, w_{n-1} נמצאים ברמה התחתונה של העץ

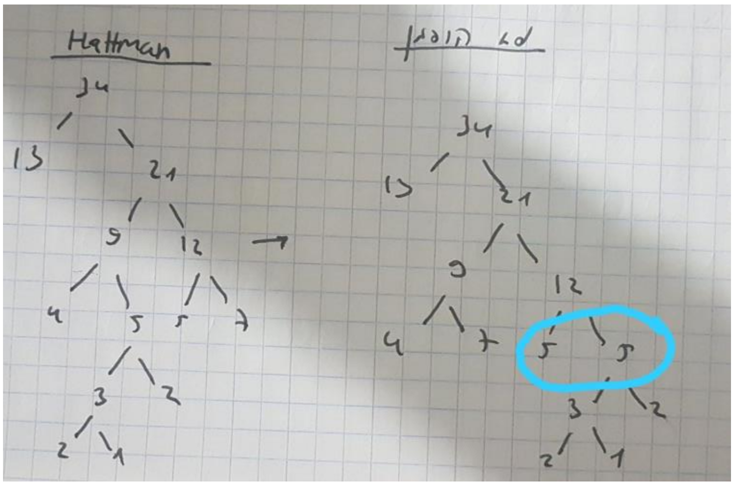
הוכחה:
נניח בשלילה כי קיים עץ אופטמלי שבו w_n, w_{n-1} הם לא ברמה התחתונה של העץ, ידוע כי ל w_n יש אח (למה 1) ולכן נבצע החלפה בין w_x לבין w_{n-1} בעץ ונראה כי הקוד החדש קצר יותר, כלומר צ"ל כי:

$$l_{n-1}w_{n-1} + l_n w_x > l_n w_{n-1} + l_{n-1} w_x$$

נשים לב כי $(l_n - l_{n-1})(w_x - w_{n-1}) > 0$
כי כל איבר במכפלה גדול מ0 ולכן אם נפתח סוגריים נקבל את מה שרצינו. סתירה לכך שהעץ אופטמלי

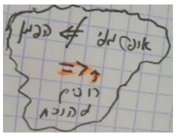
למה 3 : בעץ אופטמלי w_n, w_{n-1} יכולים להיות אחים:
הוכחה: על לפי למה 2, w_n, w_{n-1} נמצאים ברמה התחתונה ונניח כי w_n ו w_{n-1} אינם אחים, עפ"י למה 1, עץ אופטמלי הוא עץ מלא ולכן לכל אחד מהם יש אח, בהחלפת x ו w_{n-1}, $\sum w_i l_i$ אינו משתנה ולכן העץ החדש גם אופטמלי

דוגמא לעץ אופטמלי שאינו הופמן:



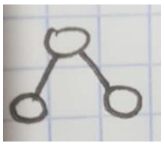
<u>משפט : עץ הופמן הוא עץ אופטמלי</u>

בהינתן משקולות $w_1, \dots, w_n$ אלגוריתם הופמן בונה מילות קוד עם אורכים $l_1, \dots, l_n$ כך ש $\sum_{i=1}^{n} w_i l_i$ הוא מינמלי.



הוכחה:

בהאינדוקציה על מספר העלים n

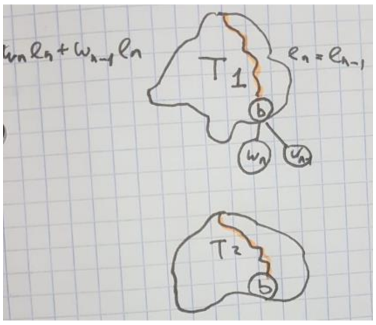


בסיס: עבור $n = 2$, נצא מעץ אופטמלי עם 2 עלים נקבל ,

עץ זה הוא עץ אופטמלי יחיד עם 2 העלים וזה עץ הזהה לעץ הופמן המתקבל ע"י 2 עלים

צעד: נוכיח עבור n עלים

יהי T_1 <u>עץ אופטמלי</u> עבור משקולות $w_1, \dots, w_n$ עם מילות קוד עם אורכים $l_1, \dots, l_n$ כך ש $\sum_{i=1}^{n} w_i l_i$ הוא מינמלי.



נבנה עץ T_2 מ T_1 ע"י השמטת w_n, w_{n-1}

נטען כי T_2 הינו עץ אופטמלי עבור $n-1$

עלים עם המשקולות $\{w_1, \dots, w_b\}$

כאשר $w_b = w_{n-1} + w_n$

נוכיח טענה זו.

נשים לב כי $M_2 = M_1 - (w_{n-1} + w_n) * 1$ (ה $1 *$ זה בגלל הסיבית שקוצצה)

נניח בשלילה כי T_2 לא אופטמלי ולכן קיים T_3 עבור אותם משקולות $\{w_1, \dots, w_b\}$ כך ש $M_3 < M_2$

נבנה עץ T_4 אשר הינו עץ זהה, עם משקולות w_n, w_{n-1} נקבל את M_4.

ידוע כי $M_3 < M_2$ לפי ההנחה ולכן

$$M_4 = M_3 + (w_{n-1} + w_n) < M_2 + (w_{n-1} + w_n) = M_1$$

בסתירה לאופטמליות של M_1 ולכן T_2 הוא עץ אופטמלי, נוסיף ל T_1 2 עלים ע"י הופמן ונקבל כי T_1 הוא עץ הופמן

הרצאה 2 – אלגוריתמים חמדניים

אלגוריתם חמדני פותר בעיית אופטמזציה

הרעיון הכללי מאחורי האלגוריתמים בקטגורייה זו היא ביצוע סדררת החלטות שכל אחת מהן היא אופטמלית בפני עצמה, ובמיחד הן מביאות לפתרון גלובלי.

בעיה 1 : בעיית המטבעות – אלגוריתם חמדני פותר קבוצת מטבעות מסויימת

בעיה 2 : בעיית הגנב השביר – אלגוריתם חמדני נותן פתרון אופטמלי לבעיה

קלט: בהינתן $G_1, \dots, G_n$ עצמים בעלי משקל w_i וערך v_i נחפש $\sum_{i=1}^{n} f_i v_i$ כאשר $f_i \in [0,1]$ להיות החלק של G_i כאשר $\sum_{i=1}^{n} f_i v_i < C$ כאשר C היא קבולת התיק.

```
S ← set of all vi/wi
while C>0
  i ← index of maximum value in S
  S ← S-{vi/wi}
  if (wi<C)
     print('wi Kilos of item i were taken')
     C ← C - wi
  else
     print('C Kilos of item i were taken')
     C ← 0
```

בעיה 3 : בעיית הגנב הבדיד – בעיית NP **אין** פתרון ע"י אלגוריתם חמדני

בעיה 4 : בעיית הפעיליות – יש פתרון ע"י חמדני

קלט: קבוצה S אשר מכילה את כל הפעילות ממויינת בסדר עולה לפי זמני הסיום.

```
Greedy-Activity-Selector(s,
n ← length(s)
A ← {1}
j ← 1
for i = 2 to n
  if s_i ≥ f_j
      then A ← A ∪ {i}
              j ← i
Return A
```

בעיה 5: כיסוי צמתים בגרף – **הסבר** בהמשך.

<u>סדר פעולות להוכחה אלגוריתם חמדני</u>

יש צורך להוכיח 2 דברים

1. נכונות בחירות האלגוריתם – צריך להוכיח כי הבחירה בכל שלב נמצאת גם כן בפתרון אופטמלי כלשהו

2. נכונות תת המבנה האופטמלי – בהינתן קבוצה S אשר מכיל את הבחירה של החמדני, אזי $S-\{x\}$ הוא גם כן פתרון אופטמלי לבעיה.

<u>על מנת להוכיח את 1 :</u>

אפשרות א':

להניח בשלילה כי לא קיים פתרון אופטמלי S אשר מכיל את הבחירה של החמדני, יהי O פתרון אופטמלי, נראה כי O נכיל את הפתרון של החמדני ולכן קיים S אשר מכיל את הבחירה של האלגוריתם החמדני

אפשרות ב':

להגדיר A קבוצה של כל האפשריות. להגדיר S פתרון אופטמלי כך ש $A \subseteq S$. להגדיר a_1 פתרון של החמדני ולהראות כי $a_1 \in S$.

אם $S \ni a_1$ אז סיימנו

אם $a_1 \notin S$ אז נמצא פתרון אופטמלי S' כך ש $a_1 \in S'$

על מנת להוכיח את 2.

להגדיר S פתרון אופטמלי לבעיה, להגדיר $S' = S - \{x\}$ כאשר x זה בחירה האלגוריתם החמדני, להראות כי S' הוא פתרון אופטמלי לתת הבעיה.

להניח כי S' הוא לא פתרון אופטמלי לתת המבנה ולהגיע לסתירה ע"י $|S|$.

<u>הוכחת נכונות לבעיית הפעליות:</u>

צ"ל:

1. נכונות הבחירה החמדנית

2. נכונות תת המבנה האופטמלי

הוכחה 1: יהי $A = \{a_1, a_2, \dots, a_n\}$ קב' כל הפעליות הנתונה בסדר עולה לפי זמני הסיום

יהי $A \subseteq S$ פתרון אופטמלי כלשהו לבעיה

יהי a_1 בחירת האלגוריתם החמדני באיטרציה הראשונה

אם $a_1 \in A$ סיימנו

אם $a_1 \notin A$ אזי נסמן את הפעילות הראשונה בפתרון האופטמלי S ב a_k כאשר $k \in [2, n]$

נסתכל על $S' = S - \{a_k\} \cup \{a_1\}$, נשים לב כי גודל הקבוצה S' זהה לגודל הקבוצה S ולכן נותר להוכיח כי בקבוצה S' חוקית (כלומר אין פעילות אשר לא מתיישבות אחת עם השנייה), נשים לב כי $f_1 < f_k$ היות ובחירת האלגוריתם החמדני בוחרת לפי זמן הסיום הקטן ביותר, ולכן אם כל הפעילות ב S מתיישבות עם f_k אז בוודאי שהם התיישבו עם f_1 ולכן S' היא קבוצה חוקית.

הוכחה 2: יהי S פתרון אופטמלי כלשהו לבעיה, ויהי a_n בחירת האלגוריתם החמדני. נוכיח כי $S' = S - \{a_n\}$ הוא תת מבנה אופטמלי לבעייה.

נניח בשלילה כי S' הוא לא תת מבנה אופטמלי לבעיה ולכן קיים S'' כך ש $|S''| > |S'|$

ולכן $|S''| > |S'| = |S| - 1$ ולכן אם נוסיף לS'' את a_n נקבל כי קיימת קבוצה $S''' = S'' \cup \{a_n\}$

כך ש $|S'''| > |S|$ בסתירה לאופטמליות של S.

<u>בעיה: אלגוריתם לכיסוי צמתים בעץ</u>

מטרה: ליצור קבוצה מינמלית S מכילה את כל צליות הגרף בעזרת הקודקודים

כיסוי צמתים בגרף – קבוצה של כל הקודקודים S כך שלכל צלע $\{u, v\}$ בגרף מקיימת ש $u \in S$ או $v \in S$ (עבור גרף לא עץ לא קיים פתרון)

פלט: כיסוי הצמתים המינמלי בG

אלגוריתם חמדני:

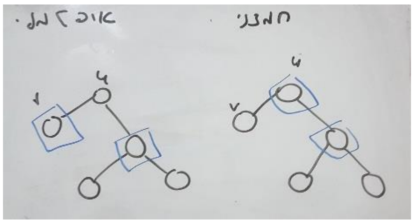
1. $u \leftarrow \emptyset$
2. כל עוד קיים עלה v בגרף
   1. הוסף את u, השכן של v בגרף לקבוצה
   2. הסר את כל הקשתות החלות על u

<u>הוכחה:</u>

צ"ל:

1. נכונות הבחירה החמדנית

2. נכונות תת המבנה האופטמלי



הוכחה 1: יהי O פתרון אופטמלי לבעיה, ויהי $u \in V$ בחירת האלגוריתם

החמדני באיטרציה הראשונה אשר שכן של v בגרף, אם $u \in O$ סיימנו.

אם $u \notin O$ אז בהכרח $v \in O$ היות ו O פתרון אופטמלי אשר מכסה את <u>כלל</u> הצלעות בגרף, אבל מכוון ש $O - \{u\} \cup \{v\}$ כיסוי תקין באותו גודל המכיל את v נקבל כי קיים פתרון אופטמלי אשר מכיל את u וזוהי סתירה.

הוכחה 2: נכונות תת המבנה האופטמלי. יהי S פתרון אופטמלי לבעיה, ויהי u בחירת האלגוריתם החמדני, נסתכל על $S' = S - \{u\}$ ונוכיח כי S' הוא פתרון אופטמלי לגרף

$$G' = G - \{u\} - \{(u, v) \in E: v \in V\}$$

נניח בשלילה כי S' הוא לא פתרון לתת הגרף G' ולכן קיים פתרון אופטמלי S'' כך ש $|S''| < |S'|$

ולכן $|S''| < |S'| = |S| - 1$ ולכן אם נוסיף ל S'' את u נקבל $S''' = S'' \cup \{u\}$ ולכן

$|S'''| < |S|$ בסתירה לאופטמליות של S

הרצאה 1 – הפרד ומשול

במדעי המחשב, הפרד ומשול היא פרדיגמת תכנון אלגוריתמים חשובה. היא מבוססת על שבירה רקורסיבית של הבעיה לשתיים או יותר תת-בעיות מאותה הצורה (או צורה דומה לה), עד שהבעיות הופכות לפשוטות דיין כדי שניתן יהיה לפתור אותן ישירות. לאחר מכן הפתרונות לתת הבעיות משולבים יחד כדי לתת פתרון לבעיה המקורית.

<u>בעיית $Min\ Max$</u>

בעיה: מציאת מנימום ומקסימום במערך

פתרון נאיבי 1: לשמור 2 משתנים $\frac{min}{max}$ ולעשות מעבר אחד על המערך

סיבוכיות: $O(2n-2) = O(n)$ היות ויש $n-1$ השוואות עבור כל משתנה.

פתרון 2 – הפרד ומשול: נפריד את הבעיה ל2, נמצא $min - max$ בחצי הראשון של המערך ונשמור ב Min_1, Max_1 ואז , נמצא $min - max$ בחצי השני של המערך ונשמור ב Min_2, Max_2 ואז נבצע 2 השוואות (נשאל Max_1, Max_2 ו Min_1, Min_2)

min Max: נתון מערך A יש למצוא את האיבר הגדול ביותר והקטן ביותר

first index, last index

```
min Max (i, j)
if (i = j) return (A[i], A[i])   // only 1 in array
if (j = i+1)
    if (A[i] < A[j]) then return (A[i], A[j])   // 2 in array
    else return (A[j], A[i])
else {
    k ← ⌊(i+j)/2⌋
    (m1, M1) = min Max (i, k)
    (m2, M2) = min Max (k+1, j)
    return (min(m1, m2), max(M1, M2))
```

נאיבי: לרוץ n-1 עבור max על המערך ועוד n-1 עבור המינימום

רעיון:

$m_1\ M_1 \quad m_2\ M_2$

השוואה בין m_2, m_1
השוואה בין M_1, M_2
זה 2

$$T(n) = \begin{cases} 0 & n=1 \leftarrow \text{תנאי עצירה} \\ 1 & n=2 \\ 2T(n/2)+2 & n>2 \end{cases}$$

מספר השוואות

$$T(n) = 2^k T\left(\frac{n}{2^k}\right) + \sum_{i=1}^{k} 2^i \qquad // \ \frac{n}{2^k} = 2 \rightarrow k = \log_2 \ldots$$

$$= \frac{n}{2} T(2) + 2^{\log_2 n/2} - 1 - 1$$

מתחילים מ 1 ולא מ 0 ולכן צריך להוריד את 0. $(1=2^0)$

$$= 2n/2 + n/2 - 2 = 3/2\, n - 2 \cong \Theta(3/2\, n)$$

$$\left(\frac{3}{2}n\right) < (2n-2)$$

בעיית Max-Max

max max: נתון מערך ורוצים שהחזיר את המקסימום הראשון והשני בגודלו

פתרון נאיבי: מצא את המקס הגדול ביותר ושמור ב max_1 ושנה אותו ל $-\infty$, שאחר מכן מצא את max_2 במערך החדש.

סיבוכיות: זמן ריצה: $O(n)$
כמות השוואות: $2n-2$

פתרון שני: מצא 2 מקסימום בכל אחד מחצי המערך ואז

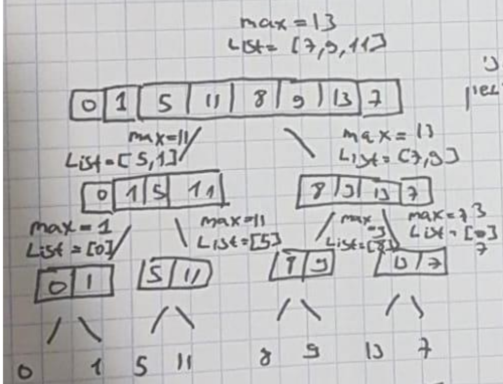
$$\begin{matrix} max1_2 < max1_1 \\ \wedge \quad\quad \vee \\ max2_2 < max2_1 \end{matrix}$$

וזה צורך ב-2 השוואות.

$$T(n) = \begin{cases} 1 & n = 2 \\ 2T(n/2)+2 & n > 2 \end{cases}$$

$$\Rightarrow \Theta(n \cdot 3/2)$$ כמו ב $\begin{Bmatrix} min \\ max \end{Bmatrix}$

ראינו: נשים לב כי יש תכונה חשובה שלא הי"ה בשאלת min-max והיא שאין טעם השוואה בין 2 איברים אם אחד נהפך שחור מקום והשני מקום 2.



פתרון שלישי: מצא מקום 1 מהחצי הראשון ומקום 2 מהחצי השני, לכל אחד מהם תהיה רשימת מועמדים למקום 2. (List מכיל את list המנצח)

אם מקום 1 מהחצי הראשון הוא המנצח אז נקח את List של max_1 ונוסיף את max_2 לרשימה ונחפש שם. את ה max_2

$$T(n) = \begin{cases} 1 & 2 \\ 2T(n/2)+1 & n > 2 \end{cases}$$

מציאת מקום 1 בכל

$$T(n) = n-1$$

$$F(n) = n-1+\log_2 n-1$$

$$= n+\log_2 n-2.$$

$$F(n) = T(n) + \underbrace{\log_2 n - 1}_{\text{רוצה רק}}$$

רשימה בגודל $|\log_2 n|$ זה $\log_2 n - 1$.

נכתב ע"י צבי מינץ

<u>בעיית כפל מחרוזות בוליאניות</u>

מכפלת מחרוזות בוליאניות {Boolean multiplication}

הבעיה הכללית: נתונות 2 מחרוזות בוליאניות באורך n כל אחת (נניח) בצורה ישירה חישוב המחרוזות יהיה בסיבוכיות $\Theta(n^2)$

נראה אלגוריתם שמשפר את זמן הריצה.

n ביטים

X: | x_1 | x_2 |

y: | y_1 | y_2 |

רעיון ראשון:

$$x = x_2 + x_1 \cdot 2^{n/2}$$

$$y = y_2 + y_1 \cdot 2^{n/2}$$

הבסיס (אם זה היה דצימאלי זה היה בבסיס 10 $4975 = 75+49\cdot 10^2$)

ולכן

$$xy = x_2y_2 + 2^{n/2}(x_1y_2 + x_2y_1) + 2^n(x_1y_1)$$

נשים לב כי יש 4 פעולות כפל ולכן

ולכן אם נפתור נקבל:

$$T(n) = \begin{cases} 1 & n=1 \\ 4T\left(\frac{n}{2}\right) + cn & n>1 \end{cases}$$

סדר פעולות החיבור

* והמכפלה נעשה ע"י פעולת לחזקת 2 Shift בזמן קבוע.

$$T(n) = 4T\left(\frac{n}{2}\right) + cn =$$
$$4\left[4T\left(\frac{n}{4}\right) + \frac{cn}{2}\right] + cn =$$
$$4^2T\left(\frac{n}{2^2}\right) + 2cn + cn =$$
$$4^2\left(4T\left(\frac{n}{2^3}\right) + \frac{cn}{2^2}\right) + 2cn + cn$$
$$= 4^3T\left(\frac{n}{2^3}\right) + 4cn + 2cn + cn$$
$$= 4^kT\left(\frac{n}{2^k}\right) + cn\sum_{i=0}^{k-1} 2^i$$

עבור $\frac{n}{2^k} = 1 \Leftarrow k = \log_2 n$ נקבל

$$T(n) = 4^{\log_2 n}T(1) + cn\sum_{i=0}^{\log_2 n - 1} 2^i =$$
$$n^2 + cn\left(2^{\log_2 n} - 1\right) = n^2 + cn^2 - 1\cdot cn$$

עדיין $T(n) = \Theta(n^2)$!

נשפר פתרון זה.

$$\boxed{\sum_{i=0}^{k} 2^i = 2^{k+1} - 1}$$

נוכל לחסוך פעולת כפל ע"י שימוש ב-3 במקום ב-4 מכפלות מסדר $n/2$.

כשמכפילים מספר עם n ספרות ב-m ספרות נקבל לכל היותר $n+m$

כאשר מחברים מספר עם $n/2$ ספרות עם עוד מספר של $n/2$ נקבל לכל היותר $n+1$ עבור carry

$$X = X_1 X_2$$
$$Y = Y_1 Y_2$$

נגדיר

$$A = X_1 Y_1$$
$$B = X_2 Y_2$$
$$C = (X_1 + X_2)(Y_1 + Y_2)$$

עדיין מכפלה של 2 מספרים מ-$n/2$ (מתעלמים מ-1 ה-Carry)

זאת במקום שהיה

$$xy = x_2 y_2 + 2^{n/2}(x_1 y_2 + x_2 y_1) + 2^n (x_1 y_1)$$

נקבל

$$xy = A + 2^{n/2}(C - A - B) + 2^n \cdot B$$

נשים לב כי יש 3 פעולות כפל של $n/2$

ולכן

$$T(n) = \begin{cases} 1 & n = 1 \\ 3T\left(\frac{n}{2}\right) + cn & n > 1 \end{cases}$$

לאחר k שלבים

$$T(n) = 3^k T\left(\frac{n}{2^k}\right) + cn \sum_{i=0}^{k-1} \left(\frac{3}{2}\right)^i$$

$$T(n) = 3^{\log_2 n} T(1) + cn\left(\left(\frac{3}{2}\right)^{\log_2 n} - 1\right)$$

$$= n^{\log_2 3} + cn\left(\frac{n^{\log_3 2}}{n} - 1\right)$$

$$= n^{\overset{1.584}{\log_2 3}} + c\left(n^{\log_2 3} - n\right) =$$

$$n^{\log_2 3}(1 + c) - cn \approx \Theta\left(n^{1.584}\right)$$

* זמן המכפלה של קלט של n הוא 3 פעמים מכפלה של חצי קלט ועוד פעולות חיבור ו-shift בזמן לינארי

$$3^{\log_2 n} = 3^{\frac{\log_3 n}{\log_2 3}} = n^{\log_2 3}$$

$$a^{\log_b n} = n^{\log_b a}$$

סילבוס

הרצאה 1 **הפרד ומשול**	**הרצאה 2** **אלגורתמים חמדניים**	**הרצאה 3** **קוד הופמן**	**הרצאה 4** **תכנון דינמי**
• Min Max • Max Max • Boolean Multiply	• בעיית פריטת המטבעות • בעיית הגנב השביר • בעיית גנב הבדיד • בעיית הפעליות • בעיית כיסוי צמתים	• קוד חסר רישות • אלגוריתם הופמן	• פיבונאצי • מקדם בינומי • הטוב מבין השניים • מיקום סוגריים בכפל מטריצות
הרצאה 5 **גרפים**	**הרצאה 6 (MST)**	**הרצאה 7 המסלול הקצר ביותר**	**הרצאה 8 המסלול הכולל הקצר ביותר**
• DFS • BFS • Topology Sort • SCC	• Prim • Kruskal	• Dijkstra • Bellman-Ford	• Floyd Warshall • Johnson