עצי חיפוש בינארי מאוזנים — Balanced search trees

מוטיבציה לעצי חיפוש בינארי מאוזנים:

פעולות של עצי חיפוש בינאריים לוקחות $O(h)$ וחלקן עלולות לגרום לעצים להיות לא מאוזנים, מה שיכול לגרור מצב ש- $h=O(n)$ ואז נקבל ביצועים ב- $O(n)$.

* המטרה: לאזן את העץ כך ש- $h=O(\log(n))$.

ישנן מספר דרכים לשמור על עצים מאוזנים כמו: AVL, Red-Black, 2-3 Trees, B-Trees. היום נדבר על AVL.

עצי חיפוש בינארי מאוזנים (BST)

ההפרש גובה בין כל תת עץ הוא לכל היותר 1.

$$|\text{left height} - \text{right height}| \le 1$$

זה מבטיח ש- $h=\lfloor \log_2(n) \rfloor$ כך שפעולות יקחו $\Theta(\log(n))$ בכל המקרים.

נרצה לאזן את העץ מחדש ע"י פעולות איזון כאשר צריך.

במקרה הטוב איזון מחדש יקח $\Theta(\log n)$ במקרה הגרוע.

במקרה הטוב אחריו: גם יקח $\Theta(\log n)$ במקרה הגרוע כאשר:

1) פעולה בודדת תקח $\Theta(\log n)$

2) כל n פעולות יקחו לא יותר מ-$\Theta(\log n)$ בממוצע ביחד.

נשים לב שעץ BST אומנם מאוזן אך אין הכרח שיהיה מלא.

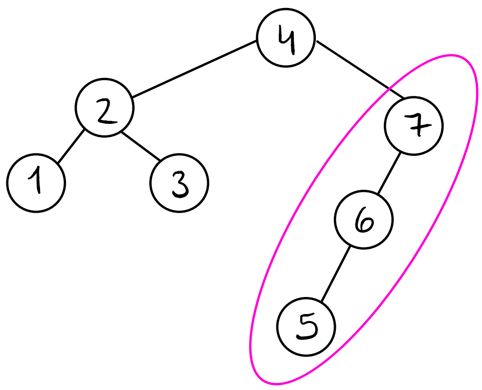
AVL trees

הגדרה:

1. העץ אינו ריק.
2. כל תת עץ מקיים את תכונת ה-AVL.
3. הפרש הגבהים של תתי העצים השמאליים והימניים יהיה 1 לכל היותר.

$$|\text{left height} - \text{right height}| \le 1$$

דוגמה לעץ שאינו עץ AVL:



Height diff = 2 > 1

$hd(x)$

עצי AVL - מקדם איזון Balance Factor:

* לכל צומת בעץ, x, יש תכונה בשם הפרש גבהים height difference.
* הוא מוגדר בצורה רקורסיבית:

1. הפרש הגבהים של עלה הוא 0.
2. הפרש הגבהים של צומת פנימית הוא הפרש הגבהים בין תת העץ השמאלי לתת העץ הימני.

$$hd(x) = \text{left height} - \text{right height}$$

* מקדמי האיזון של כל הצמתים ב-AVL הם: $hd(x) \in \{-1,0,1\}$

לא נוכח בהרצאה

משפט: גובה של עץ AVL עם $n>0$ צמתים הוא $\Theta(\log(n))$.

עץ AVL: הכנסות ומחיקות

הכנסה ומחיקה מתבצעות באופן זהה לעץ חיפוש רגילים.

* פעולות אלו עלולות לשנות את הפרשי הגבהים (hd(x)) של תתי עצים.
* הפרשי גבהים אלו יכולים להתעדכן רקורסיבית במהלך/אחרי התרחשות הפעולה.

* כמה הפרשי גבהים יכולים להשתנות? כמות תתי העצים המושפעים מהפעולה.

← בהכנסה: כמות הצמתים מהעלה שהוכנס ועד השורש ← $O(h)$

← בהוצאה מקרים 1-2: כמות הצמתים מהצומת שנמחקה ועד השורש ← $O(h)$

← בהוצאה מקרה 3: מתווסף גובה תת עץ ימני/שמאלי ← $O(h)$

עץ AVL: פעולת איזון מחדש

* הכנסה/מחיקה בודדת של צומת יכולה לשנות את הפרש הגבהים של תתי עצים ב-AVL אף יותר ב-1.
* עדכון הפרש הגבהים בפעולות אלו יקח $O(h)$.
* אם הפרשי הגבהים המעודכנים הם $1,0,-1$ אז העץ הוא AVL ולא נדרשות פעולות נוספות, אחרת- נצטרך לבצע פעולות איזון מחדש.

פעולת איזון לדוגמה היא: "סיבוב" (rotation) של תת עץ סביב צומת ספציפית.

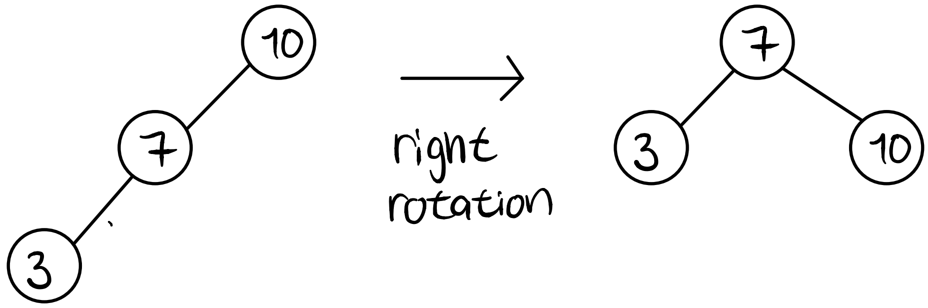
פעולת איזון מחדש - סיבוב
AVL trees rebalancing-Rotation

האיזון מחדש בשיטה זו מתבצע ע"י "סיבוב" של תתי עצים.

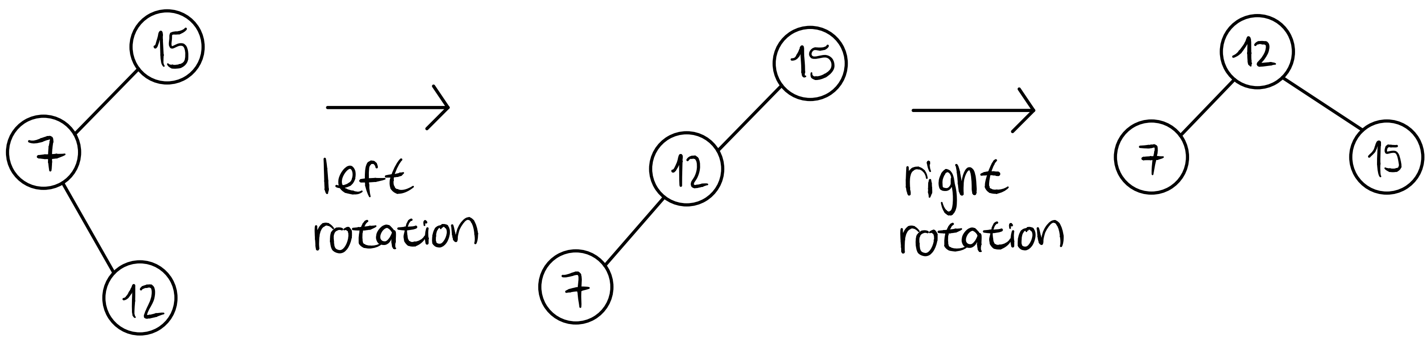
נבין כי חוסר איזון מתרחש כאשר בוצעה הוצאה או הכנסה.

דוגמה פעולות סיבוב:

כאשר פעולת סיבוב אחת מספיקה:



כאשר נדרשות שתי פעולות סיבוב:



איזון מחודש של AVL ע"י סיבוב: הכנסת איבר חדש

תהא x צומת שדורשת איזון מחדש להכנסה בודדת של איבר חדש.
ישנם 4 מקרים:

1. הכנסה לתת עץ השמאלי של הילד השמאלי של x.
2. הכנסה לתת עץ הימני של הילד השמאלי של x.
3. הכנסה לתת עץ השמאלי של הילד הימני של x.
4. הכנסה לתת עץ הימני של הילד הימני של x.

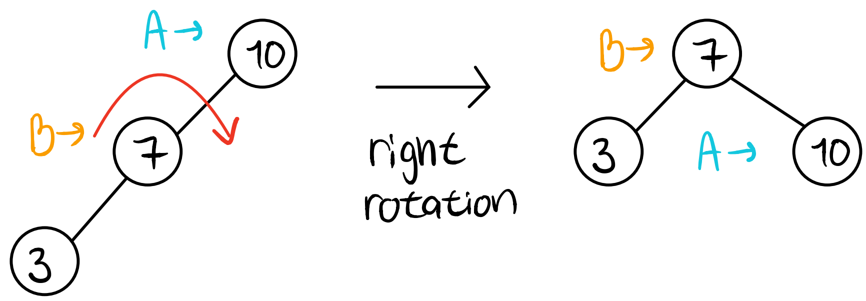
מקרים 1 ו-4 הם סימטריים - דורשים סיבוב יחיד.

מקרים 2 ו-3 הם סימטריים - דורשים סיבוב כפול.

סיבוב יחיד נסתכל על מקרים 1 ו-4.

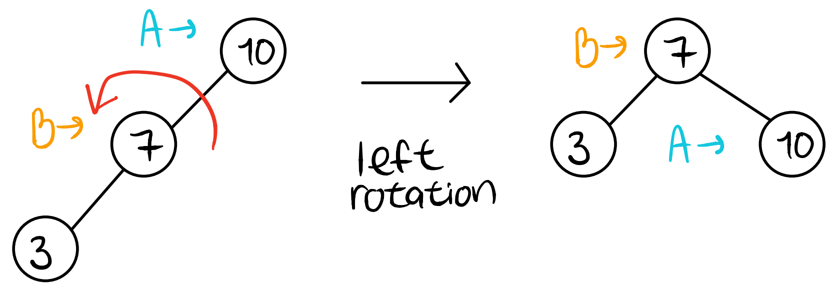
* מקרה 1. הכנסה לתת עץ השמאלי של הילד השמאלי של x.

insert 3



* מקרה 4. הכנסה לתת עץ הימני של הילד הימני של x.

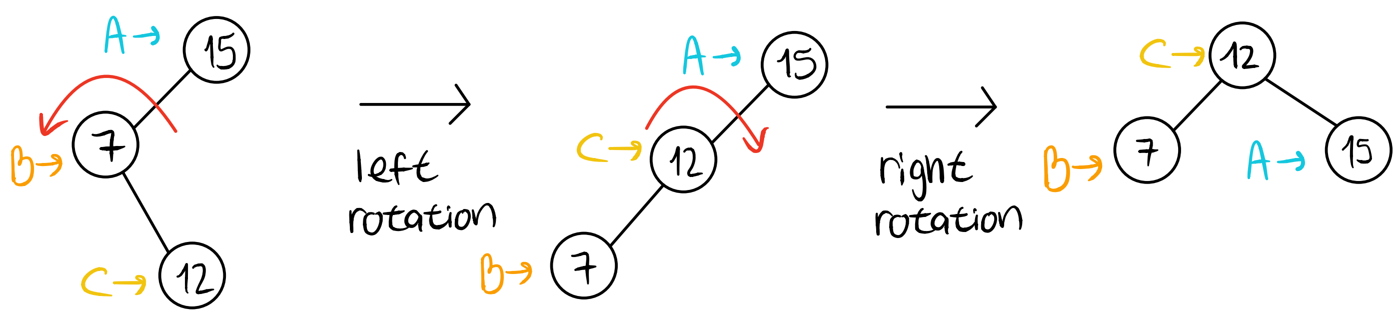
insert 10



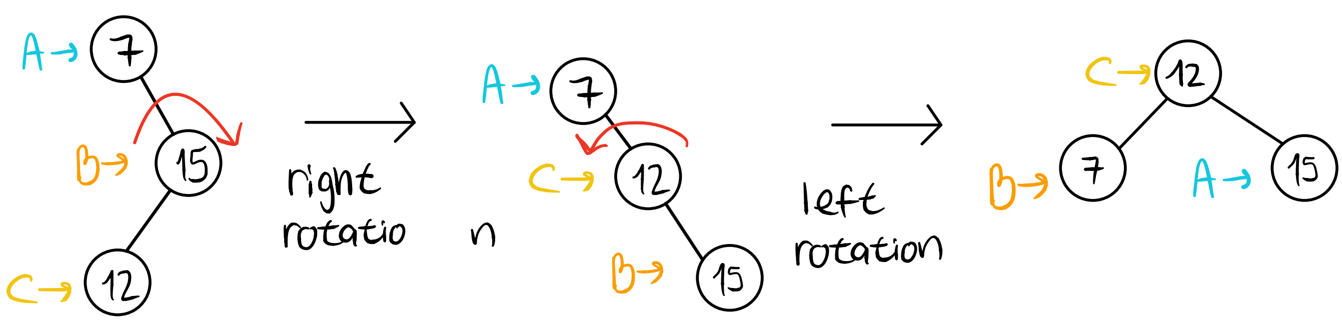
insert 12

סיבוב כפול: נסתכל על מקרים 2 ו-3.

* מקרה 2. הכנסה לתת עץ הימני של הילד השמאלי של x.



* מקרה 3. הכנסה לתת עץ השמאלי של הילד הימני של x.



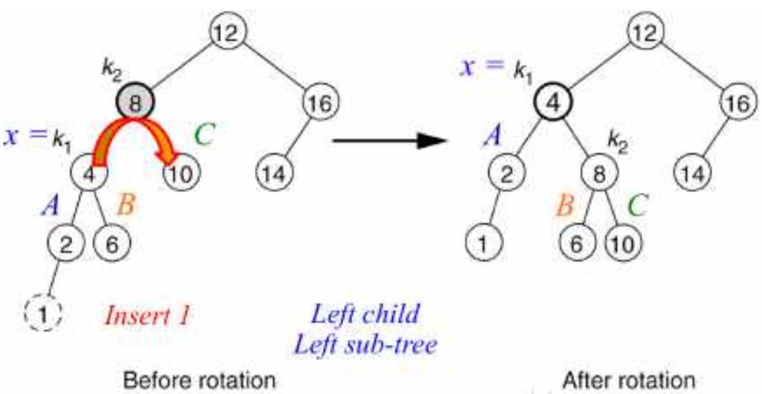
סיבוב יחיד

* החלפת התפקידים בין האב לילד של X, כך שנשמר סדר החיפוש.
* החלפה בין צומת לילד שלו:
  1. הילד הופך לאב.
  2. והאב הופך להיות הילד השמאלי/ימני.

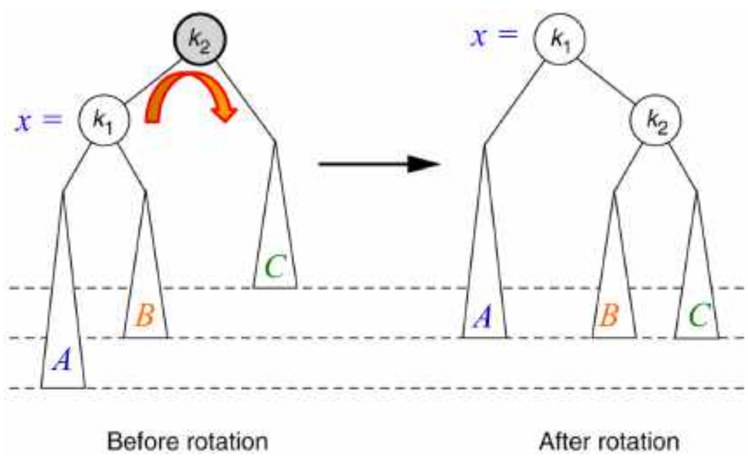
והעץ המאוזן שקיבלנו עומד בדרישות תכונת ה-AVL.

המחשה:

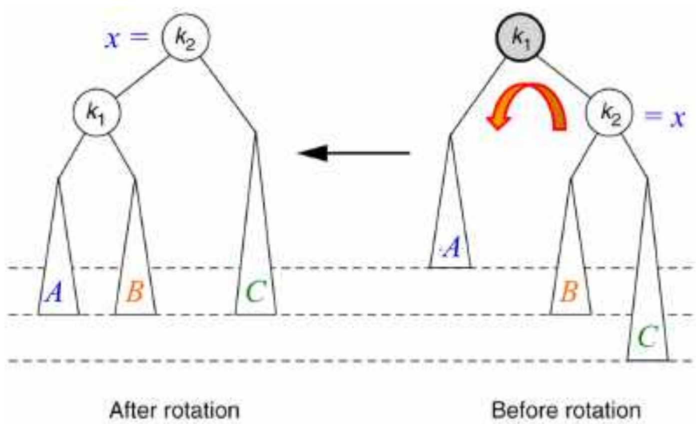
סיבוב יחיד - הכנסה:



סיבוב יחיד - ימינה:



סיבוב יחיד - שמאלה:



9/12/20

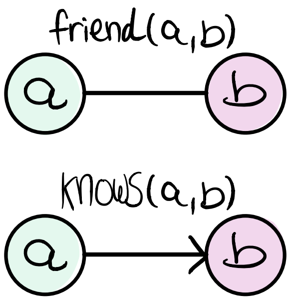
Data Structures - 8 הרצאה
Graphs and basic graph search algorithms
גרפים ואלגוריתמי חיפוש בגרפים

מוטיבציה לשימוש בגרפים:

* הרבה יחסים יכולים להיות מוצגים ביחס בינארי
* ישנם יחסים סימטריים או א-סימטריים (דפי web וכו')
* גרף הוא מבנה אבסטרקטי שמתאר יחסים בינאריים
* גרף הוא הכללה של עץ.
* הרבה בעיות יכולות להיפתר ע"י "ייצוג" של גרפים

דוגמה ליחס בינארי

דוגמה ליחס א-סימטרי

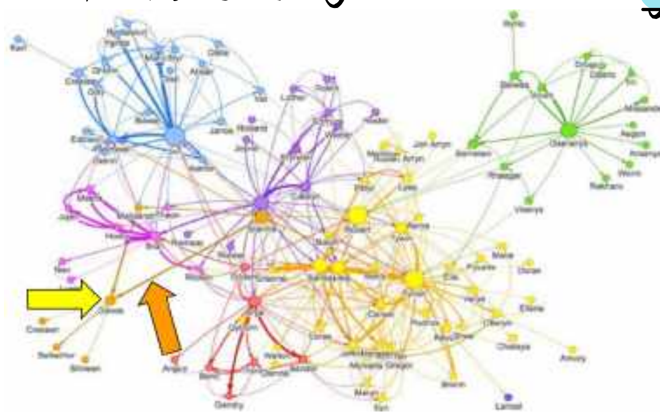


דוגמאות:

רשת חברתית

קודקודים: משתמשים

צלעות: קשר בין משתמשים



רכבת תחתית

קודקודים: תחנות

צלעות: דרכים

גרפים

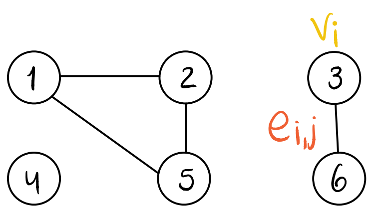
הגדרה: גרף $G=(V,E)$ הוא זוג כאשר

• $V=\{v_1,\dots,v_n\}$ הם הקודקודים

• $E=\{e_1,\dots,e_m\}$ הן צלעות

* צלע $e_k=e_{i,j}=(v_i,v_j)$ מקשרת בין שני קודקודים, v_i ו-v_j.
* צלעות יכולות להיות מכוונות או לא מכוונות

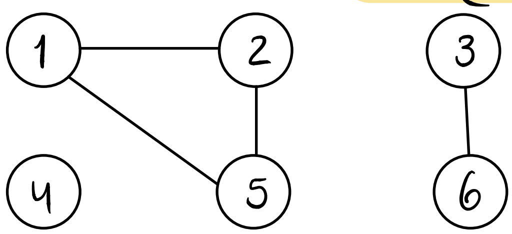
$e_{i,j}: v_i \to v_j$ או $e_{j,i}: v_j \to v_i$



* הגרף G הוא סופי כאשר V ו-E שתיהן קבוצות סופיות.
* גודל הגרף הוא סכום הקודקודים והצלעות: $|G|=|E|+|V|$

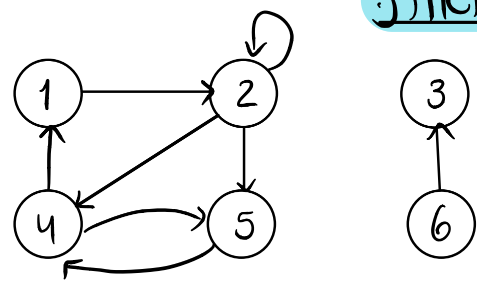
דוגמאות:

$V = \{1,2,3,4,5,6\}$



$E = \{(1,2),(2,5),(5,1)..\}$

גרף לא-מכוון



$E = \{(1,2),(2,2),(4,1)..\}$

גרף מכוון

גרפים לא מכוונים

* אם יש קשר בין u ו-v כך ש- $e=(u,v)$ נקרא להם שכנים.
* לא קיימות לולאות כך ש- $e=(u,u)$ (אין צלע מ-u ל-u)
* ישנו יחס סימטרי $e(u,v)$ - אם u שכן של v אז v שכן של u.
* דרגת הקודקוד $d(v)$ היא סך כל הצלעות שנוגעות בקודקוד הזה.
* סכום כל הדרגות הוא בדיוק פעמיים כמות הצלעות $\sum d(v_i) = 2|E|$.

גרפים מכוונים

* צלע $e=(u,v)$ יוצאת מ-u ל-v. v סמוך ל-u.
* קיימות לולאות כך ש- $e=(u,u)$ (יכולה להיות צלע מ-u ל-u)
* דרגות: נסמן ב-$d_{in}(v)$ את כמות הצלעות הנכנסות ל-v.
* ונסמן ב-$d_{out}(v)$ את כמות הצלעות היוצאות מ-v.
* סכום כל הדרגות הולך כך: $\sum d_{in}(v_i) = \sum d_{out}(v_i) = |E|$.

מסלולים:

הגדרה: מסלול מ-u ל-v בגרף G הוא סדרה של קודקודים:

$$path(u,v) = (v_0, v_1, \ldots, v_k)$$

עבור $u=v_0$ ו- $v_k=v$ כך שמתקיים: $i\in[|V|]$ ו- $(v_{i-1},v_i)\in E$.

* המסלולים מקודקוד לקודקוד מגוונים, יכולים לחלוק צלעות ולא חייבים להיות הקצרים ביותר.
* אורך המסלול הוא מספר הצלעות מקודקוד u ל- v.

גרף ממושקל

* גרף ממושקל הוא גרף שבו צלעות הן בעלות משקל/ מחיר ו- $w(v_i,v_j)>0$.
* גרף הוא גרף ממושקל שבו משקל כל צלע הוא 1.
* אם אין לי מסלול בין שני קודקודים אז המחיר/משקל הוא ∞.

המשקל של מסלול $path(u,v)$ 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- $v_k=v$.

טרמינולוגיית סוגי גרפים

מעגל: מסלול שמגיע מקודקוד לקודקוד עצמו חזרה עם גודל גדול/שווה ל-1.

גרף קשיר: גרף בלתי מכוון הוא קשיר כשקיים מסלול בין כל שני קודקודים בגרף.

גרף קשיר היטב: גרף מכוון הוא קשיר היטב כשקיים בו מסלול מכוון בין כל שני קודקודים בגרף.

תת גרף: הקבוצות $V'\subseteq V$, $E'\subseteq E$ יוצרות את $G'\subseteq G$, תת-גרף כך:

$G'=(V',E')$.

רכיב קשירות: תת הקבוצה הגדולה ביותר שממנה יש מסלול בין כל שני קודקודים בגרף.

גדלי גרפים

* בגרף ישנן לכל היותר $|E| = O(|V|^2)$ צלעות.
* **קליקה:** גרף שבו $|E| = |V|^2$.

הוכחה באינדוקציה על גודל V

* ישנן לפחות $|E| \ge |V|-1$ בגרף קשיר - זה בדיוק המקרה של עץ.
* **גרף פלנרי:** אף צלע לא חוצה צלע אחרת, ומתקיים: $|E| = O(|V|)$.

גרפים ועצים

עץ הוא גרף קשיר ללא מעגלים. לעץ יש בדיוק $|E| = |V|-1$ צלעות.

בגרף לא מכוון G, 4 התנאים הבאים שקולים:

1) G הוא עץ.
2) G הוא חסר מעגלים, הוספת צלע תוסיף מעגל.
3) G הוא קשיר, מחיקת צלע הורסת את הקשירות.
4) G הוא חסר לולאות עצמיות ויש מסלול בין כל שני קודקודים.

ייצוג גרפים:

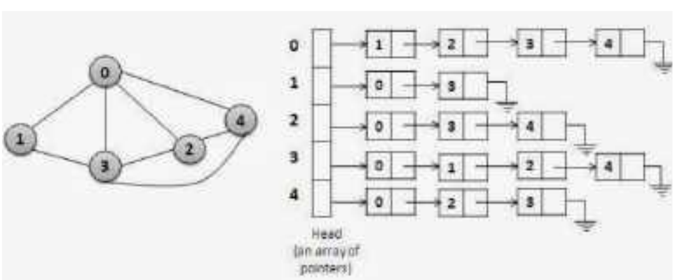
ישנן 2 דרכים לייצג גרפים:

Adjacency List

1) **רשימת השכנים:** לכל צלע v ישנה רשימה מקושרת L_v המכילה את שכניו בגרף.

גודל הייצוג הוא: $O(|V|+|E|)$

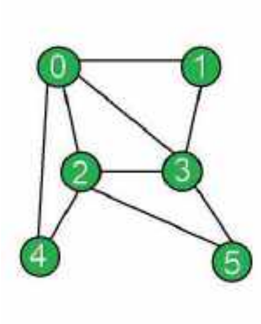
דוגמה:



Adjacency Matrix

2) **מטריצת ה"שכנים":** מטריצה שהיא $|V| \times |V|$ שבה בתא $e=(u,v)$ מיוצגת ע"י 1 אם קיימת. גודל הייצוג הוא: $\Theta(|V|^2)$.

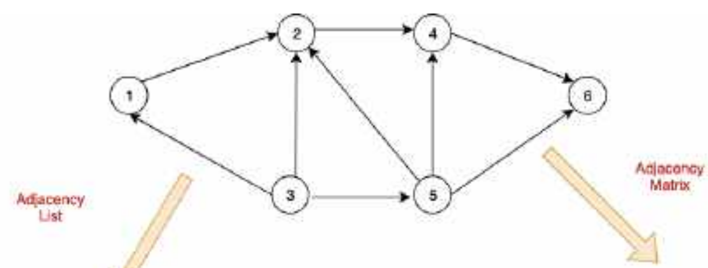
דוגמה:

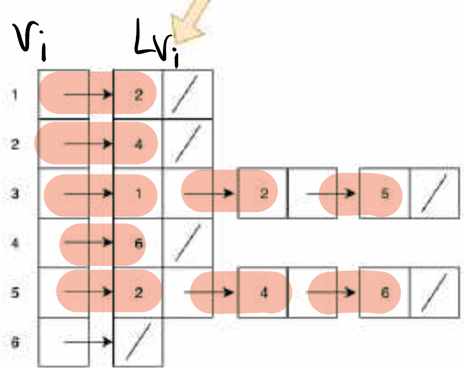


	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	0	1	0	0
2	1	0	0	1	1	1
3	1	1	1	0	0	1
4	1	0	1	0	0	0
5	0	0	1	1	0	0

הערה: אם יש מעט קשתות עדיף את הרשימה.
ואם יש הרבה קשתות עדיף את המטריצה.

דוגמה לשני הייצוגים עבור אותו גרף:





A

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

בעיות גרפים ואלגוריתמים

4) **מעבר על גרפים:**

Breath-First Search – BFS רוחב

Depth-First Search – DFS עומק

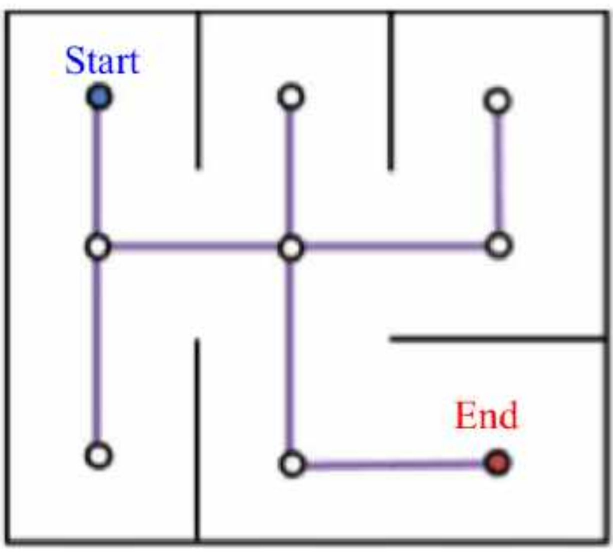
2) מציאת רכיבי קשירות: Strongly Connected Components SCC

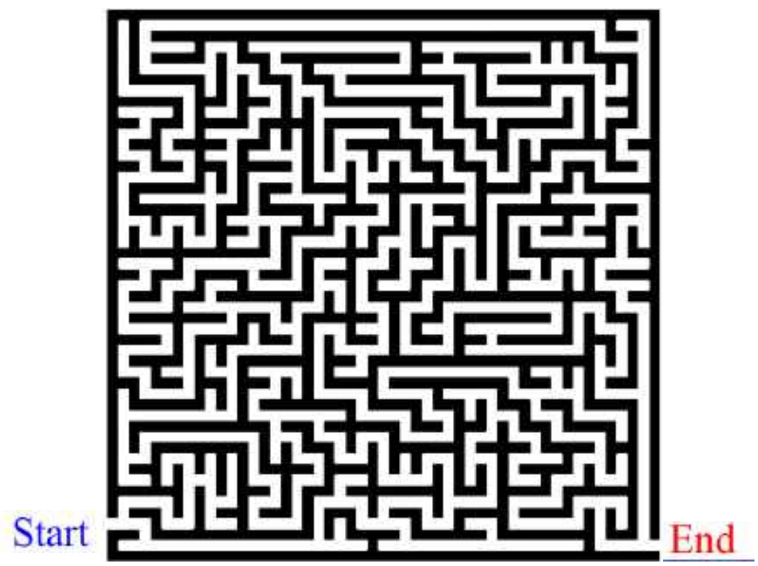
3) עץ פורש מינימום: רכיב קשירות אחד כך שסכום המשקלים מינימלי:
Minimum Spanning Trees MST

4) המסלולים הקצרים ביותר:
* Single path
* Single source shortest path
* All-pairs shortest path

מיד נדבר עליהם.

דוגמה - מציאת הדרך במבוך (חיפוש לעומק וחיפוש לרוחב)





בעיות מסלול קצר ביותר

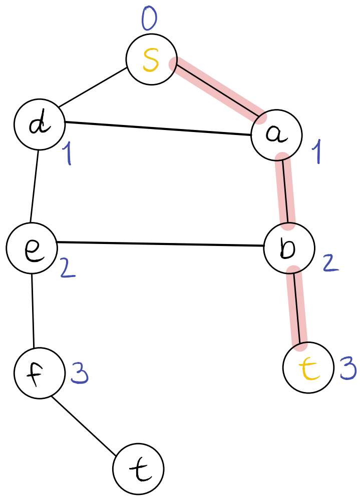
1) Single path - בהינתן 2 קודקודים s ו-t, נמצא את המסלול הקצר ביותר מ-s ל-t (ואת אורכה).

2) Single Source - בהינתן קודקוד s, נמצא את המסלול הקצר ביותר לכל שאר הקודקודים.

3) All pairs - נמצא את המסלול הקצר ביותר בין כל זוג קודקודים (s,t).

נתחיל ב- 1) Single path ונמצא מסלול:

אינטואיציה: איך נחפש בגרף?



נתחיל מקודקוד s שאותו נסמן ב-0.

← אם t הוא שכן של s, סיימנו.

← אחרת, נסמן את רמותיהם של כל שכניו ב-1.

וכעת נכליל:

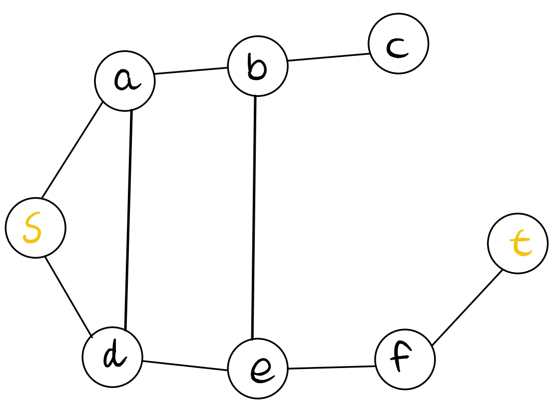
← אם t הוא שכן של קודקוד ברמה ה-i, סיימנו.

← אחרת, נסמן את רמותיהם של כל <u>שכני</u> הקודקודים מדרגה i, בדרגה i+1.

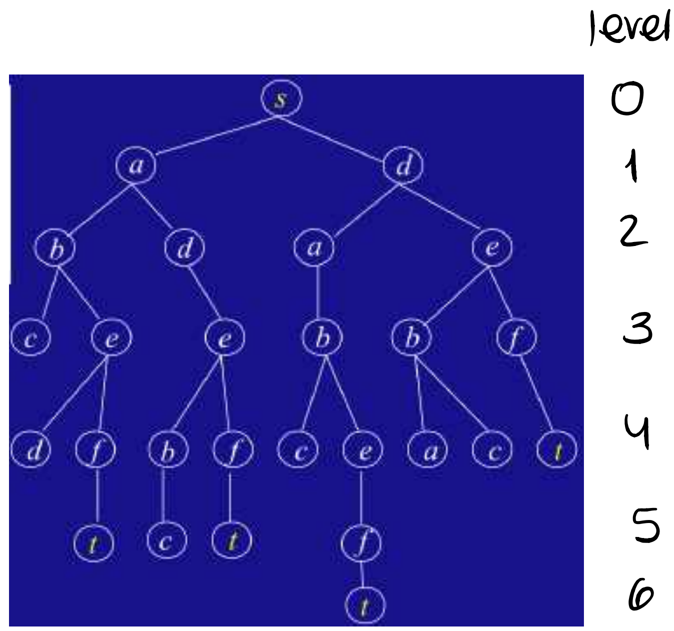
כאשר נמצא את t, נאתר את המסלול אחורה ע"י מעבר בקודקודים i, i-1, i-2, ..., 0.

השיטה הזו (חיפוש לרוחב) מבטיחה שהמסלול הכה קצר ביותר.

נהפוך את זה לבעיית חיפוש בעץ:



$\Rightarrow$



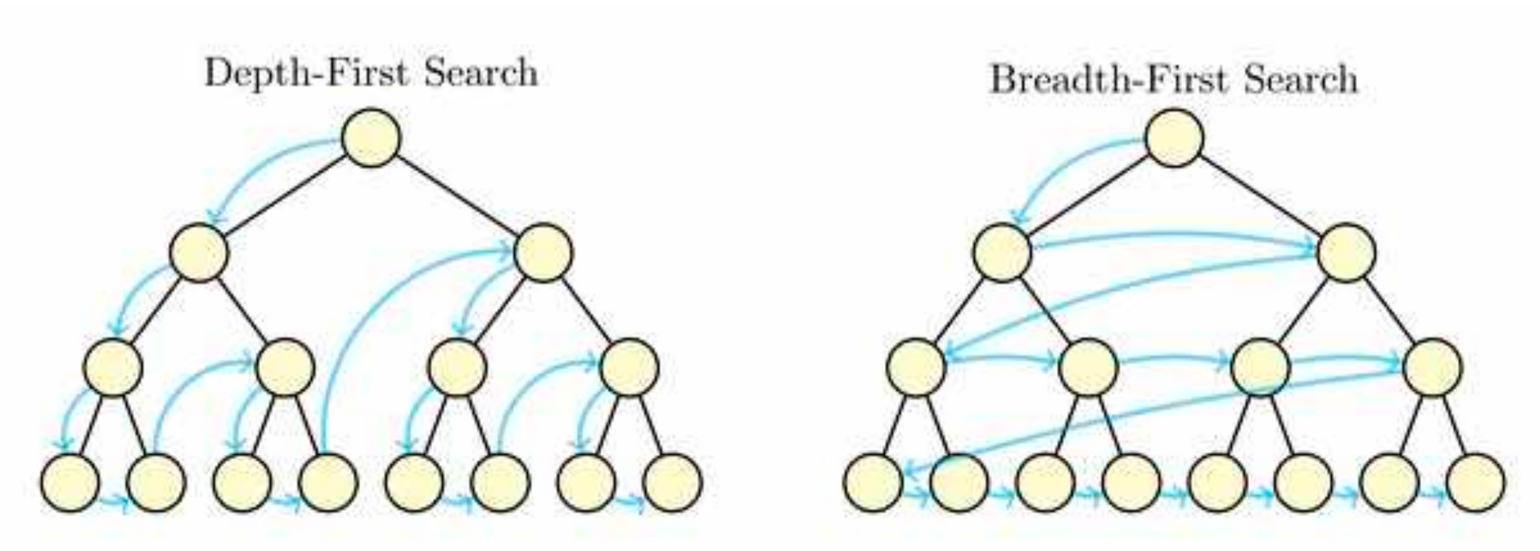

Paths from s to t:
<s,a,b,e,f,t>
<s,a,d,e,f,t>
<s,d,a,b,e,f,t>
<s,d,e,f,t>

אפשר לחפש בעץ לרוחב: BFS

נבדוק את כל הקודקודים ברמה ה-i לפני המעבר על כל הקודקודים ברמה ה-i+1.

אפשר לחפש בעץ לאורך: DFS

נבדוק קודקוד אחד ברמה ה-i ואז מיד לשכן שלו (השמאלי) ואז כאשר נגיע לעלה, נחזור רמה אחת קודם, נבדוק את השכן השני (שלא נבדק) ונמשיך באותו התהליך עד שנמצא.



אלגוריתם BFS - חיפוש לרוחב

חיפוש בגרף לפי רמות יורדות כאשר מתחילים מ-S. ניתן לחשוב על כך כגלי הדף.

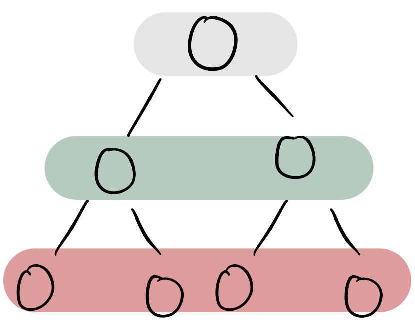
בכל רגע נתון ישנן חזיתות של הגל, ושם נפריד בין 3 סוגים של קודקודים:

1) כאלו שביקרנו בהם Visited

2) הנוכחיים Current - ישמרו בתור Q.

3) לא ביקרנו בהם not-visited

בגרף:



נייצג אותו כך:

u

lable	
dist	π

u.lable — אחד מ-3 הסוגים

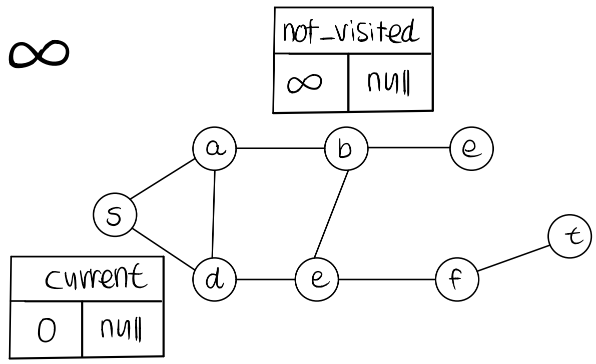
u.π — אבא predecessor (מאופיה בשותי)

u.dist — המרחק מ-S

האלגוריתם BFS

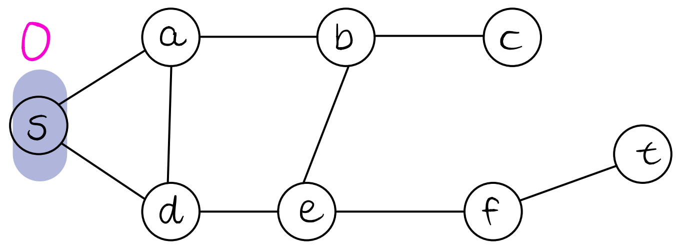
BFS(G,s):

1) s.label = current, s.dist = 0, s.π = null
2) for all vertices u in V / {s} do:
3) u.label = not_visited , u.dist = ∞
4) u.π = null
5) EnQueue (Q,s)
6) while Q is not empty do:
7) u = DeQueue (Q)
8) for each v that is neighbor of u do:
9) if v.lable = not_visited then:
10) { v.label = current
11) v.dist = u.dist + 1 , v.π = u
12) EnQueue (Q,v) }
13) u.lable = visited

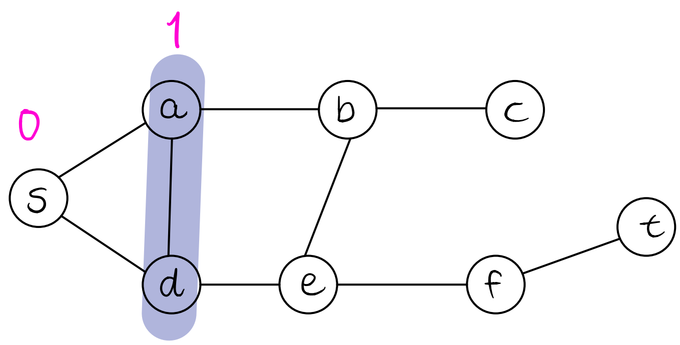


ניתן להוסיף כאן בדיקה: האם הגענו ל-t ואם כן, נעצור.

דוגמה: אלגוריתם BFS



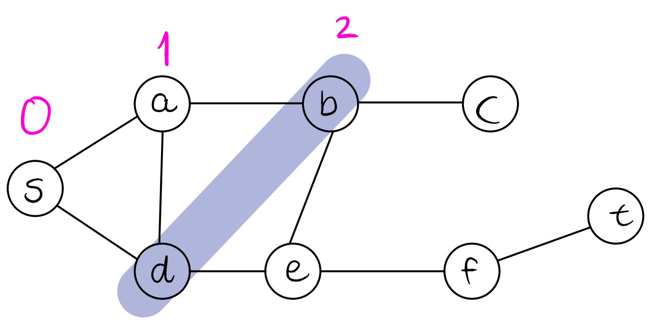
$Q = \langle s \rangle$



$u = s$

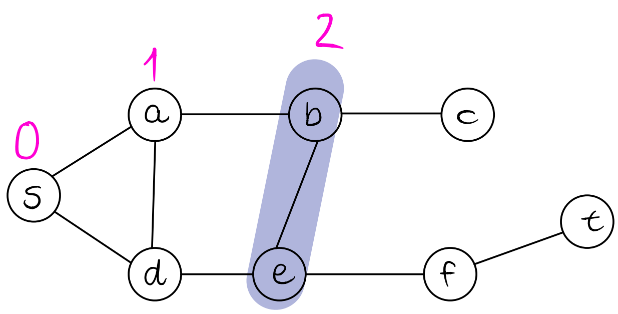
$Q = \langle a \rangle$

$Q = \langle a, d \rangle$



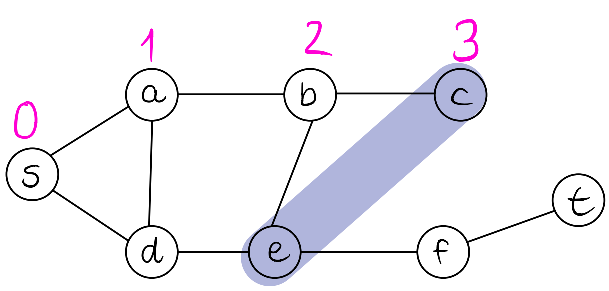
$u = a$

$Q = \langle d, b \rangle$



$u = d$

$Q = \langle b, e \rangle$



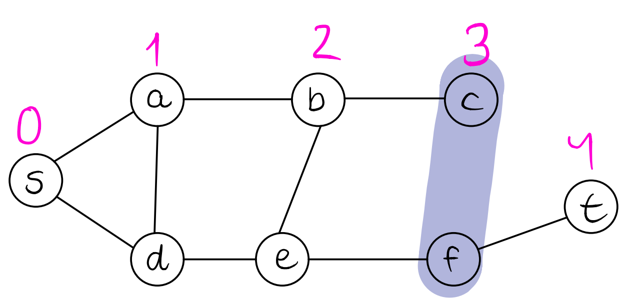
$u = b$

$Q = \langle e, c \rangle$

וכך הלאה...

עד ש- $u = f$

ו- $Q = \langle t \rangle$



$u = c$

$Q = \langle c, f \rangle$

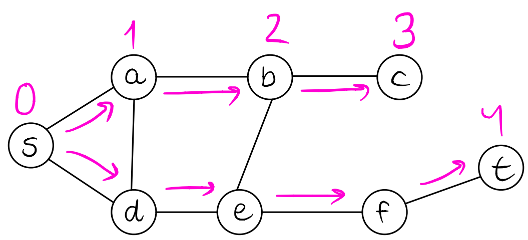
תכונות אלגוריתם BFS

* Q כולל רק קודקודים נוכחיים current
* ברגע שקודקוד הופך ל-current או visited הוא לעולם לא יחזור להיות not-visited.
* ברגע שסיימנו עם כל השכנים, הקודקוד הופך ל-visited.
* ניתן להכניס עוד if על מנת לעצור את האלגוריתם כאשר t נמצא, או לא נמצא כלל, כמו ב- .
* האלגוריתם בונה את תת הגרף המוביל ל-t כעץ BF כך ש:

$G_\pi = (V_\pi, E_\pi)$ כאשר $V_\pi = \{v \in V \mid v.\pi \neq null\} \cup \{s\}$

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

למה זה עץ? כל אחד מהקודקודים אבא אחד והצלחנו להגיע ל-t לאחר שעברנו בכל קודקוד בדרך. הוא כולל רכיב קשירות אחד ומתאר את כל המסלולים מ-s ל-t.



סיבוכיות זמן הריצה של BFS

* האלגוריתם מוריד כל קודקוד פעם אחת בדיוק, כלומר, אין חזרות.
ולכן לכל היותר יש |V| פעולות לכן |V| חזרות של DeQueue. $O(|V|)$
* לכל קודקוד, נעבור על כל השכנים ומבצע מספר קבוע של פעולות. $O(|E|)$
* העבודה על כל הזוג ב-if שבתוך ה-while הוא זמן קבוע של מספר הצלעות היוצאות לכן נקבל מכל הקודקודים בתוך הזה את העבודה כמס' הצלעות $|E|$. $O(|E|)$

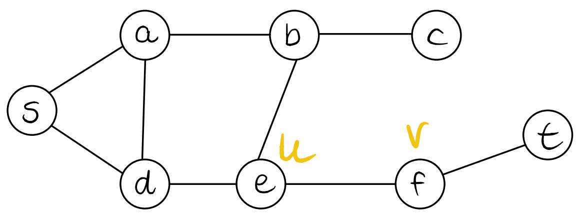
לכן סה"כ $O(|V|) + O(|E|) = O(|V| + |E|) \leq O(|V|^2)$

נכונות BFS

יהי $G=(V,E)$ גרף ו- $s\in V$ הקודקוד ממנו נצא.

נגדיר פונקציה $\delta(s,u)$ שהיא המרחק הקצר ביותר מ-s ל-u (כמות הצלעות המינימלית):

$\delta(u,u)=0$ ו- $\delta(s,u)=\infty$ כאשר אין דרך.



להציב את u ימינה

למה I: לכל צלע $(u,v)\in E$ מתקיים: $\delta(s,v)\le\delta(s,u)+1$

הוכחה: המסלול הקצר ביותר מ-s ל-v לא יכול להיות ארוך מהמסלול הקצר מ-s ל-u ועוד צלע אחת (u,v).

למה II: בסוף ביצוע האלגוריתם נקבל ש: $v.dist\ge\delta(s,v)$ $\forall v\in V$

הוכחה: אינדוקציה על כמות פעולות הכנסה לתור - EnQueue.

בסיס: כאשר s הוא הראשון שהוכנס, נסתכל על מסלולים באורך 0.

נקבל ש- $s.dist=\delta(s,s)=0$ ו- $v.dist=\infty\ge\delta(s,v)$ $\forall v\in V/\{s\}$

צעד: נניח שבצלע לקודקוד v שהוא not-visited שהתגלה במהלך החיפוש מ-u. נקבל ש- $u.dist\ge\delta(s,u)$ לכן נשים לב כי:

$v.dist=u.dist+1\ge\delta(s,u)+1\ge\delta(s,v)$

והשדה $v.dist$ לא ישתנה מכאן.

ישנן עוד למות שלא הוכחנו ובעזרת ארבעת הלמות ניתן להוכיח את השוויון

משפט: נגדיר את B המסלולים מ-s ל-v. בסוף האלגוריתם $v.dist=\delta(s,v)$

אחד המסלולים הקצרים בין s לכל אחד מהקודקודים $v\in V/\{s\}$ הוא המסלול הקצר ביותר מ-s ל- $v.\pi$ שמוצמד לו הצלע $(v.\pi,v)$.

לא הוכחנו

16/12/20

הרצאה 9 - Data Structures

DFS, מיון טופולוגי, רכיבי קשירות חזקים

אלגוריתם DFS - שימוש בעיקר כדי להוציא מידע מעניין מגרף נתון.

שלושה שימושים שונים ל- DFS בהם נעסוק:

* מיון טופולוגי
* רכיבי קשירות

} היום

* עצים מינ' פורשים } שבוע הבא

מהלך אלגוריתם DFS

* נתחיל מהקודקוד s נתון ונשיך לעמוק כל עוד ניתן: נטייל בקודקוד במסלול הנוכחי!
* נלך חזרה בשיטת ה-back tracking לקודקוד הקודם ונבחר בקודקוד השכן שטרם עדיין not_visited.
* אם אכן נמצאו הקודקודים שהם עדיין בסטטוס not_visited, נבחר אחד מהם כ-s ונחזור על התהליך.
* התוצאה היא יער של עצי עומק

$G_\pi = (V, E_\pi)$, $E_\pi = \{ (v.\pi, v) \mid v \in V \text{ and } v.\pi \neq null \}$

כאשר $v.\pi$ הוא הקודם של v בעץ החיפוש.

* נסמן את הקודקודים ע"י 3 lables: visited, current, not_visited
* נוסיף עוד שני שדות חשובים המסמנים חתימת זמן (כניסה ויציאה):

u.d - discoverd בזמן הכניסה ו-1- u.f - finished בזמן היציאה.

באותו רגע הופך ל- current   באותו רגע הופך ל- visited

כמובן ש- $u.f > u.d$. והזמן הזה ינוע בין 1 לבין 2|V|.

* עקרון ה-backtracking ממומש באמצעות הרקורסיה.

האלגוריתם עצמו:

נדבר בסימוני הקודקודים מהרצאה 8:

1) כאלו שביקרנו בהם Visited
2) הנוכחיים Current
3) לא ביקרנו בהם not-visited

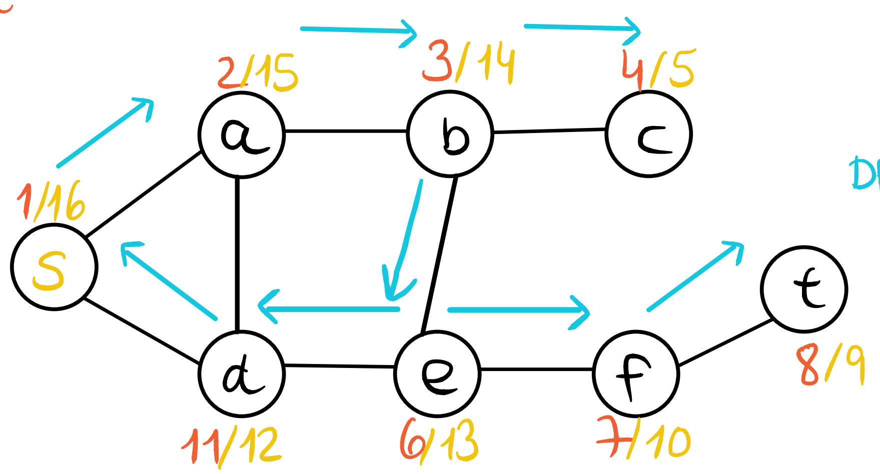
```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE  // not_visited
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE  // not_visited
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1              // white vertex u has just been discovered
u.d = time
u.color = GRAY  // current
for each v ∈ G.Adj[u]        // explore edge (u, v)
    if v.color == WHITE  // not_visited
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK  // visited  // blacken u; it is finished
time = time + 1
u.f = time
```

דוגמה:

u.d זמן כניסה

u.f זמן יציאה



צעדות בעץ ה-DFS

עובד כ- LIFO, ניתן לראות ע"י צורת התחומות הזמנים

סיבוכיות DFS:

בכל צעד בגרף נבקר פעמיים, לאחר שסימנו את קודקוד כלשהו ואת שכניו, לא נחזור אליו. הקריאה הרקורסיבית נעשית פעם אחת עבור כל צעד בגרף ולכן הסיבוכיות היא לכל היותר מס' השכנים (כמות הצלעות) ולכן $\Theta(E)$.
בסך הכל נקבל $\Theta(|V|)+\Theta(|E|)=\Theta(|V|+|E|)$

'יש וריאציות שונות. "יתכנו מצבים שבהם $|E|=0$ ואז $\Theta(V)$
או ש- $|E|=|V|^2$ ואז $\Theta(V^2)$

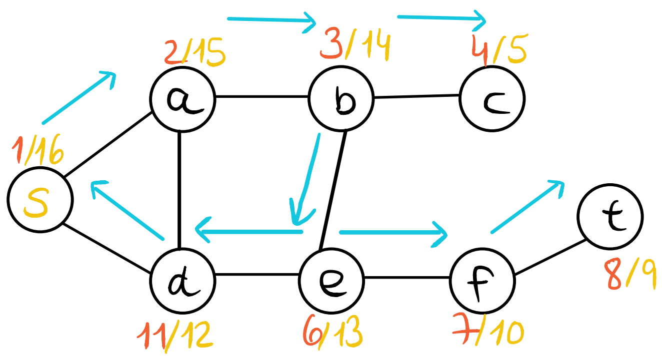
מאפייני DFS

הסדר של הטיול הוא תלוי בבחירה קודקוד בחרנו ובחירה שכנים ובחירה סדר.

כלומר, מכאן ניתן יכולים לצאת מס' טיולים שונים.

נוכל לעדכן את DFS כך שהחיפוש יתבצע מקודקוד המוצא s ולמצוא את המסלול הקצר ביותר מ-s ל-t.

כעת ננסה לחקור מהי החוקיות בין הזמנים האלו (מעבר לכך ש- $u.d < u.f$)



ננסה לחשוב על זה כחוקיות של פתיחת וסגירת סוגריים.

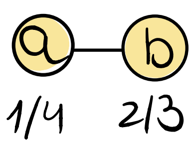
① נתחיל ב-DFS של קודקוד יחיד:

$u.d=1$, אין שכנים אז נחזור ל-a ונסגור:

$\overset{1}{(a}\ \overset{2}{a)} \rightarrow ()$

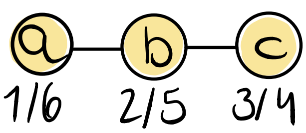
נתרגם את הסדרה הזו לפתיחה וסגירה של סוגריים.

② כעת נבנה ב-DFS של שני קודקודים:



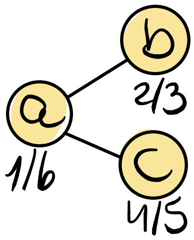
$\overset{1}{(a}\ \overset{2}{(b}\ \overset{3}{b)}\ \overset{4}{a)} \rightarrow (())$

③ כעת נבנה ב-DFS של שלושה קודקודים:



$\underset{1}{(a}\ \underset{2}{(b}\ \underset{3}{(c}\ \underset{4}{c)}\ \underset{5}{b)}\ \underset{6}{a)} \rightarrow ((()))$

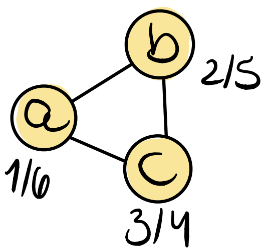
נשים לב לחוקיות של פתיחת וסגירת סוגריים בגרפים מורכבים יותר:



(a (b b) (c c) a)
1 2 3 4 5 6

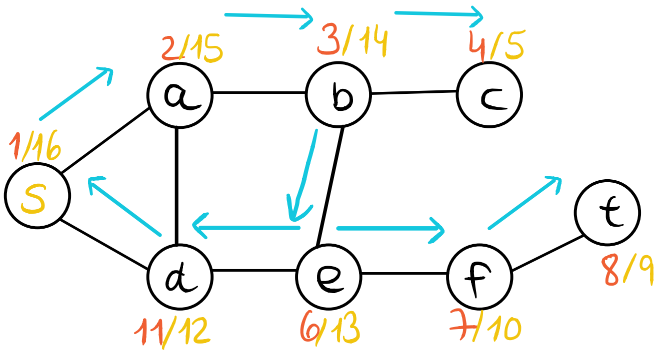
(a (c c) (b b) a)

(( ) ( ))



(a (b (c c) b) a) → ((( )))
1 2 3 4 5 6

נשים לב שבמודה ואין קשר בין הסוגריים, ושקולה למס' רכיבי קשירות:

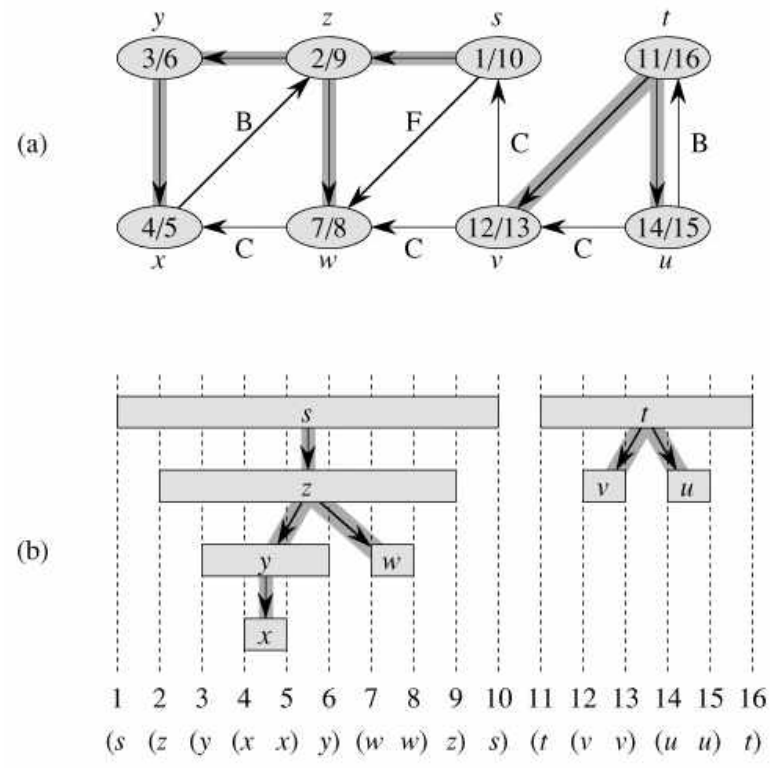


Discover	open (	push
Finish	close)	pop

(s (a (b (c c) (e (f (t t) f) (d d) e) b) a) s)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

ועוד צורה:

(s ———————————————— s)
1 (a ———————————— a) 16
2 (b ——————————— b) 15
3 (c-c) (e ——————— e) 14
4 5 6 (f ——— f) (d — d) 13
7 (t-t) 10 11 12
8 9



דוגמה נוספת להמחשה:

משפט הסוגריים:

ב-DFS של גרף $G=(V,E)$, לכל שני קודקודים u ו-v, אחד מהתנאים הבאים יתקיים:

(1) הקטעים $[u.d, u.f]$ ו-$[v.d, v.f]$ זרים, אין ביניהם חיתוך.

(2) $[u.d, u.f] \subseteq [v.d, v.f]$ ו-u צאצא של v.

(3) $[u.d, u.f] \supseteq [v.d, v.f]$ ו-u אב של v.

תזכורת: מדובר על זמני כניסה ויציאה

הוכחה:

נניח בה"כ ש-$u.d<v.d$ (המקרה בו $v.d<u.d$ סימטרי). ייתכנו אחד מהשניים:

(1 $v.d<u.f$, כלומר, u התגלה לראשונה כאשר v עדיין היה current.

אם גיליתי את u קודם, לפני v, אז הטיפול בו מתקיים לפני שנחזור ל-v.

$\Longleftarrow$ v צאצא של u, $[u.d, u.f] \supseteq [v.d, v.f]$

(2 $u.f<v.d$, ונזכור ש-$u.d<v.d$ ולכן $[v.d,v.f]$ ו-$[u.d,u.f]$ זרים לגמרי (מקרה (1) במשפט). במילים, אף קודקוד לא התגלה בזמן ששני היה ב-current אז לא מתקיימים יחסים ביניהם.

מסקנה: v הוא צאצא של קודקוד u, אמ"מ $u.d < v.d < v.f < u.f$

זמן הכניסה של u
זמן היציאה של v
זמן הכניסה של v
זמן היציאה של u

משפט Visited Path Theorem

בעץ DFS של גרף G, מתקיים כי v צאצא של u אמ"מ ניתן להגיע ל-v דרך קודקודי not-visited ואם כך יש מסלול של חיפוש לעומק ל-v.

הוכחה:

בכיוון הראשון $\Leftarrow$ (מניחים ש-v צאצא של u) מתקיים ש- $u.d < w.d$ (w צאצא של u) ולכן w הוא קודקוד not_visited בזמן u.d (עוד לא התגלה בזמן זה).

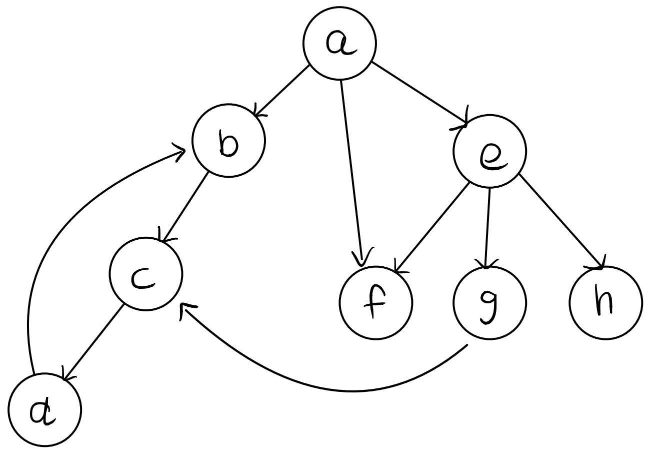
בכיוון השני $\Rightarrow$ נניח בשלילה שאין מסלול כזה ו- v לא צאצא של u.
נניח שקיים w שהוא אב של v וצאצא של u. אם כך מתקיים $w.f \leq u.f$
ולכן מהמסקנה של המשפט הקודם: $u.d < v.d < w.f \leq u.f$. וזו סתירה
כי אז $[v.d, v.f] \subseteq [u.d, u.f]$ ולכן u צאצא של v.

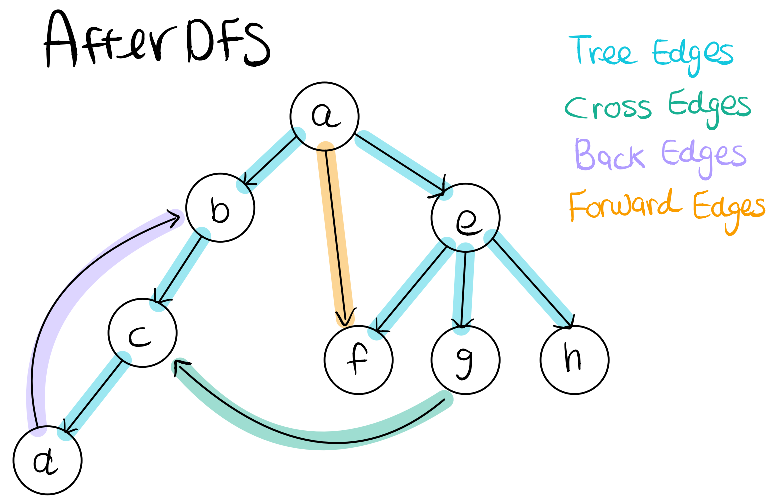
גרף מכוון

סיווג הצלעות בעץ DFS לפי 4 קטגוריות

1. צלעות עץ — Tree Edges - צלעות רגילות בעץ חיפוש לעומק.
2. צלעות אחורה — Back Edges - צלעות שמחברות u לאב קדמון v.
3. צלעות קדימה — Forward Edges - צלעות שאינן קשתות בעץ, שמחברות קודקוד u לקודקוד קדימה בעץ.
4. צלעות חוצות — Cross Edges - כל שאר הצלעות. מקשרות קודקודים שאין ביניהם יחסים של צאצא ואב.

דוגמה לצלעות DFS מכוון





משפט: ב-DFS של גרף לא-מכוון $G(V,E)$,
כל צלע ב-E היא צלע עץ או צלע אחורה.

הוכחה: תהי $e=(u,v)\in E$ ונניח ש- $v.d<u.d$. אז v התגלה והסתיים
לפני ש-u הסתיים, מכיוון ש-v הוא ברשימת השכנים של V.
אם e התגלה מ-u-ל-V אז u הוא not_visited עד אותו זמן.
אחרת, זו צלע אחורה כי u עדיין current כאשר הצלע עדיין נחקרת.

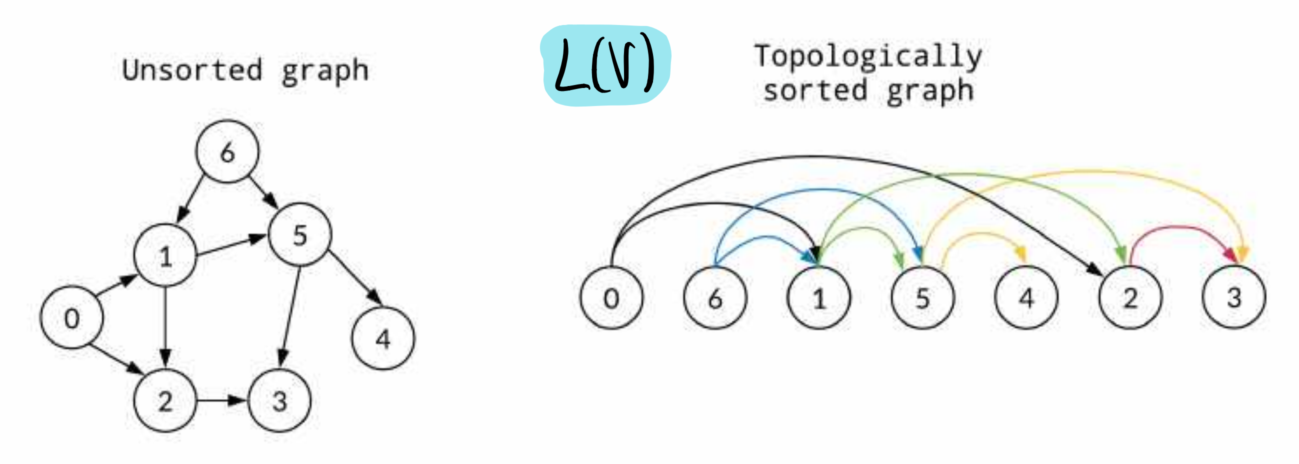
רלוונטי רק לגרף מכוון ללא מעגלים - DAG

מיון טופולוגי

מטרה: לקחת גרף ולהוציא סדר חלקי מיוחד.

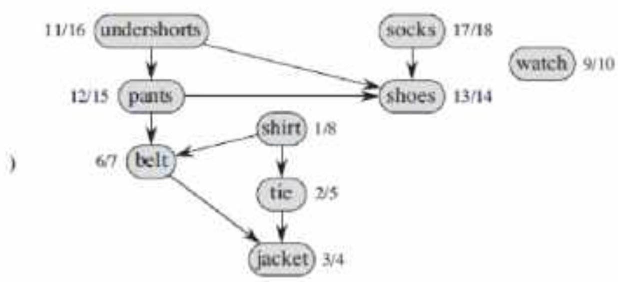
הגדרה: מיון טופולוגי של גרף מכוון ללא מעגלים $G=(V,E)$ הוא סידור
ליניארי של קודקודי G, רשימה ע"פ סדר $L(V)$ של קודקודי V,
כך שאם G מכיל צלע (u,v) אז u מופיע לפני v בסידור $L(V)$.

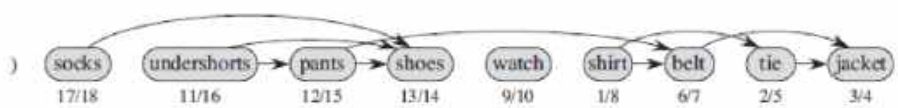
דוגמה:



דוגמה מהחיים (מהספר):

סדר פעולות
בהתלבשות





אלגוריתם מיון טופולוגי

Topological-Sort (G):

1. L = null
2. for each $v \in V$:
3. call DFS(v) to compute v.f
4. if v == visited then:
5. insert v in the front of L
6. return L

חידוד: הסימון נשמר מה-DFS ולכן השדות "ישמרו"
(visited וכו')

טענה: גרף מכוון G אינו מכיל מעגלים אם"ם לאחר DFS ב-G לא מצאנו צלעות אחורה.

הוכחה: כיוון ראשון ($\Leftarrow$): נניח בשלילה שכן קיימת צלע אחורה $e=(u,v)$ אז v הוא קדמון של u, לכן מצאנו מסלול מ-v ל-u ולכן G מכיל מעגל.

כיוון שני ($\Rightarrow$): נניח בשלילה שב-G יש מעגל, נסמנו C. ויהי v הקודקוד הראשון שמתגלה ב-C ($v.d$) ותהי (u,v) הצלע הקודמת ב-C.

בזמן $v.d$ מצאנו מסלול not_visited מ-v ל-u. אם כך, u צאצא של v, כלומר, ישנה צלע אחורה.

נכונות בתרגול.

משפט: האלגוריתם מיון-טופולוגי יודע גרף מכוון ללא מעגלים.

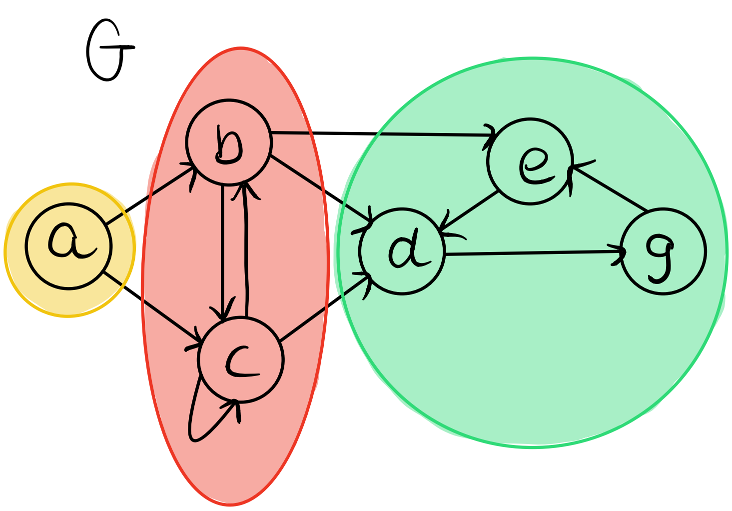
אחד מהשימושים של DFS הוא מציאת ה-SCC בגרף

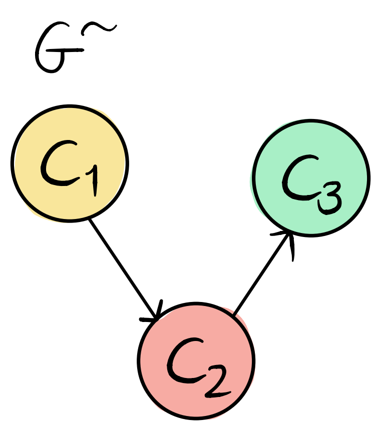
רכיבי קשירות בגרף מכוון

הגדרה: רכיבים קשירים היטב (SCC) המסומנים $C_1,C_2,\dots,C_k$ בגרף G הם תתי הגרפים השונים הגדולים ביותר כך שלכל שני קודקודים u ו-v ב-C_i, ישנו מסלול מ-u ל-v ומ-v ל-u.

$\tilde{G}$ גרף מכוון ללא מעגלים.

דוגמה:





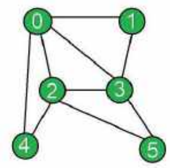
גרף משוחלף Transpose Graph

<u>הגדרה:</u> הגרף המשוחלף $G^T=(V,E^T)$ של G הוא הגרף G עם כיוונים הפוכים.

ובאופן פורמלי: $E^T=\{(u,v) \mid (v,u)\in E\}$

נזכר במטריצת השכניות מהרצאה 8:

מדובר גם בשחלוף שלאותה מטריצה.



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

נרצה לחשב בזמן לינארי את רכיבי הקשירות בגרף, נעשה זאת ע"י:

SCC Algorithm

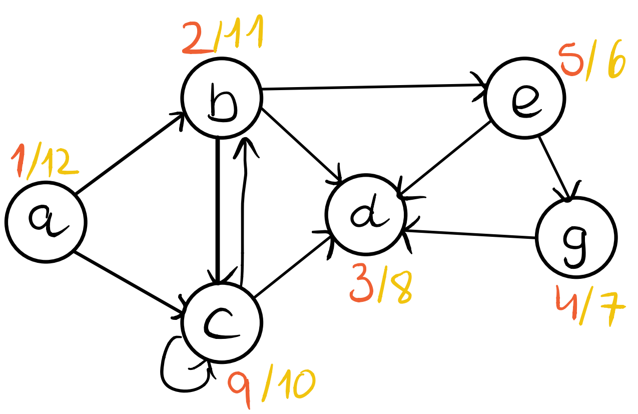
<u>מהלך:</u>
DFS בכיוון אחד על G
וכן DFS על הגרף המשוחלף G^T.

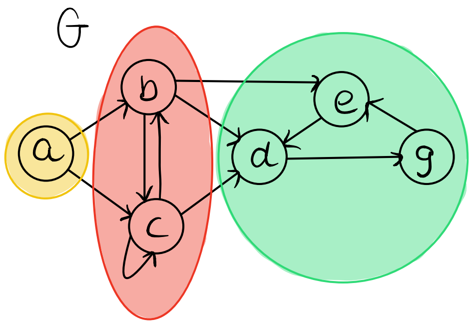
SCC-Algorithm (G):

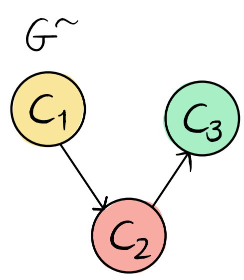
1. call DFS(G) to compute $v.f$ $(\forall v\in V)$
2. compute G^T
3. call DFS(G^T) in the order of decreasing $v.f$

הפלט יהיה קודקודי $\tilde{V}$ של G לפי DFS כרכיבי קשירות נפרדים,
$\tilde{E}$ יהיו צלעות שלא ברכיבי הקשירות.

<u>דוגמה:</u>







הוכחת נכונות SCC

למה 1: אם שני קודקודים, u ו-v שייכים לאותו SCC, נגיד C, כל מסלול ביניהם נשאר בתוך C.

למה 2: יהיו C_1 ו-C_2 שני SCC נפרדים בגרף G נתון.
יהיו $e=(u,v)$ כאשר $u\in C_1$ ו-$v\in C_2$ אז מתקיים:
$f(C_1)>f(C_2)$ כאשר $f(C)=\min_{u\in C}(f(C))$ עבור $C\subseteq G$.

משפט: האלגוריתם SCC אכן מוצא את הרכיבים הקשורים היטב בגרף מכוון נתון G.

הוכחה: נוכיח באינדוקציה על מספר עצי ה-DFS שנמצאו במהלך ה-DFS בגרף G^T.

בסיס: $n=0$ אין עצים לכן הדבר נכון באופן טריוויאלי.

צעד: $k<n$ העצים הראשונים מופקים ע"י האלגוריתם SCC.
נוכיח עבור $n+1$ עצים.

לפי **למה 2** מתקיים כי $u.f=f(C_1)>f(C_2)$, עבור רכיב קשירות שלא מצאנו, C_2 כלשהו.
לפי ההנחה, כאשר מצאנו את קודקוד u, כל הקודקודים מתוך רכיב הקשירות של עץ שלא נמצאו, מסומנים כ-not_visited.
לכן הקודקודים האלו צאצאים של u ברכיב C, ואלו יהיו לא צאצאים ברכיב אחר.
לא פגענו ברכיבי הקשירות הקיימים שנמצאו קודם לכן (n הרכיבים)
הוספנו את הרכיב ה-$n+1$, רכיב קשירות בודד שלא מחובר לרכיבים הקיימים.

23/12/20

הרצאה 10 - Data Structures
עץ פורש מינימלי
Minimum Spanning Tree (MST)

<u>מוטיבציה:</u> בהינתן גרף קשיר ממושקל לא מכוון $G=(V,E)$, נרצה למצוא את תת הקבוצה T של צלעות

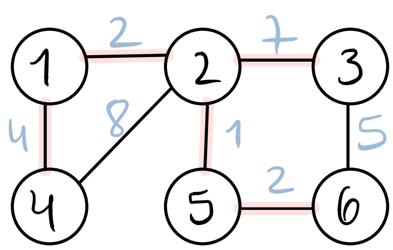
כך ש: 1. הצלעות $T\subseteq E$ מחברים את כל הצלעות <u>וגם</u>

2. הסכום של משקלי הצלעות הוא הקטן ביותר

במצב שבו 1. ו- 2. מתקיימים נקבל גרף חדש $G'=(V,E)$

<u>דוגמה:</u> גרף G וגרף G', משקלים

T = כל הצלעות של G'.



סכום: 12

כאשר סכום המשקלים מינימלי, נקבל עץ פורש מינימלי.

<u>גרפים ממושקלים:</u>

<u>הגדרה:</u> גרף ממושקל הוא גרף שבו הצלעות הן בעלות משקל $weight(v_i,v_j)>0$.

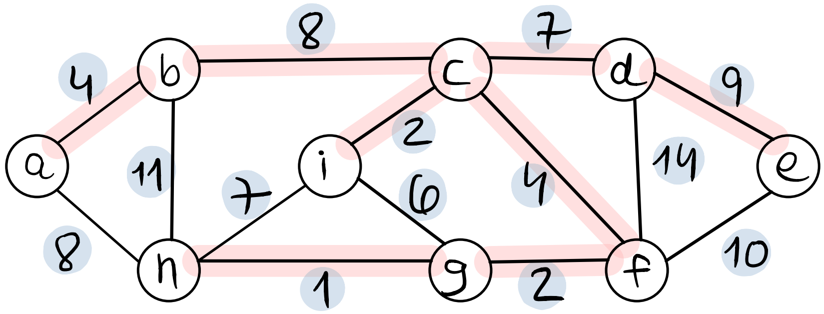
* ניתן להפוך כל גרף לא ממושקל לממושקל ע"י הוספת משקל כגון 1 לכל צלע.
* המשקל (הצלעות) בין שני קודקודים שאין ביניהם צלע הוא ∞ (שימושי לאלגוריתם).
* המשקל של מסלול $path(u,v)$ יוגדר להיות סכום המשקלים של הצלעות:

$$weight(path(u,v)) = \sum_{i=0}^{K} weight(v_{i-1},v_i) \qquad (u=v_0,\ v=v_K)$$

ובנוסף המשקל של תת קבוצה T היא סכום המשקלים של הצלעות:

$$weight(T) = \sum_{i=0}^{|T|} weight(e_i) \qquad (e_i\in T)$$

דוגמה לעץ פורש:



כאשר נחליף בין הצלעות נקבל משקלים אחרים.

* יכולים להיות כמה עצים פורשים מינימליים.

עץ פורש מינימלי Minimum Spanning Tree (MST)

הגדרה: יהי $G=(V,E)$ גרף לא מכוון ממושקל קשור. נגדיר את העץ הפורש להיות תת קבוצה $T\subseteq E$ כך ש- $G'=(V,T)$ קשיר ללא מעגלים.

עץ פורש מינימלי הוא עץ פורש שסכום משקלי T הוא המינימלי.

כלומר, $weight(T)=\sum_{(u,v)\in T} weight(u,v)$ הוא המינימלי.

* נרצה למצוא אלגוריתם שימצא עץ פורש מינימלי. נתחיל בהקדמה לפתרון.

"חימום"

סכום מינימלי של K מספרים שלמים

בהינתן סט $A=\{a_1,...,a_n\}$ של n מספרים שלמים ו- $a_i>0$ לכל i. נרצה למצוא $S\subseteq A$ של $K\le n$ איברים כך שסכומם הוא מינימלי: $min\_sum\_K=\min_{a_i\in A}\left(\sum_{i=1}^{K} a_i\right)$

דוגמה: עבור $A=\{2,25,6,10,3,11,7,32\}$ נקבל ש- $n=8$ ו- $K=4$ ולכן $S=\{2,3,6,7\}$ והסכום המינימלי הוא 18.

כלומר, קיבלנו שהמינימום הוא הסכום של K המספרים הקטנים ביותר.

* רעיון אחד לבעיה זו: מיון. לא מתוחכם להשיג, נצטרך למיין את A תחילה מה שייקח לכל הפחות $O(n\log n)$ ואז להוציא K האיברים הראשונים.

* רעיון נוסף לבעיה זו: גישה חמדנית. נמצא את המינימום, נוציא אותו מ-A, נוסיף ל-S, ונחזור על זה k פעמים. בעצם נשתמש ב-Min-Heap.
קודם כל נצטרך לבנות ערימת מינימום פעם אחת ואז לחלץ את המינימום k פעמים ובסוף נקבל - $O(n+k\log n) = O(n)+k\cdot O(\log n)$.

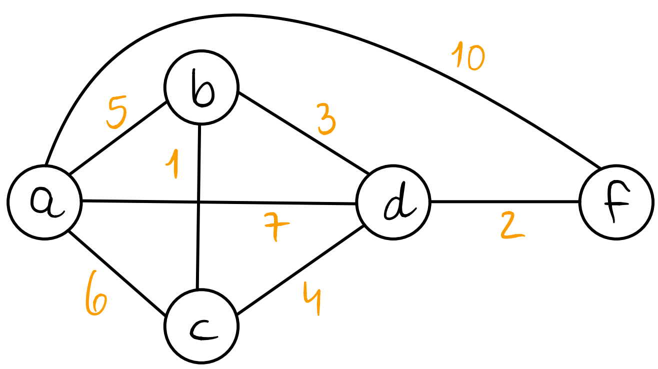
מושג חשוב שנדבר עליו בהרצאה זו הוא: אלגוריתם חמדן.

הגדרה: <u>אלגוריתם חמדן</u> הוא אלגוריתם שבוחר את האפשרות הטובה ביותר בהינתן המצב הנוכחי, מבלי לחשב השלכות עתידיות.

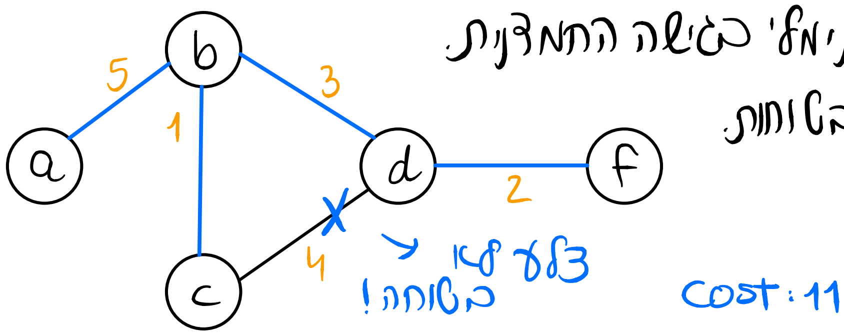
לא תמיד אופטימלי

מהקשר שלנו *, נרצה למצוא כל פעם צלע יחידה שתעמוד בשני התנאים:
1) מבנה של עץ - ללא מעגלים. 2) משקל צלע מינימלי (אין צלע "קלה" יותר)
כל צלע שתעמוד בתנאים אלו תיקרא: צלע בטוחה.
נרצה להוסיף צלעות בצורה כזו עד שכל הקודקודים יהיו מחוברים.

מתוך הגרף הבא:



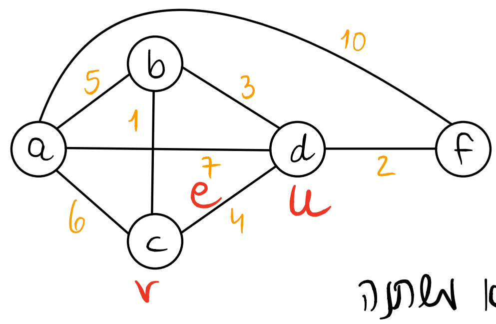
נדבר על פתרון מינימלי בגישה החמדנית.
נסמן ב- צלעות בטוחות



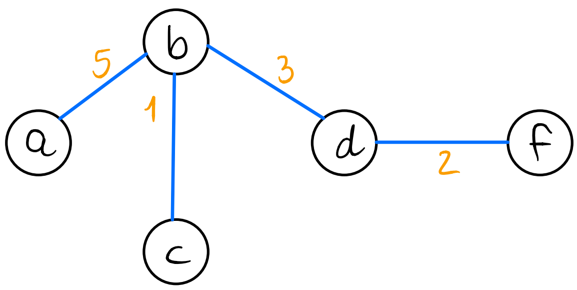
האלגוריתם עצמו:

```
GENERIC-MST(G, w)
A = ∅
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    A = A ∪ {(u, v)}
return A
```

קלט:



פלט:



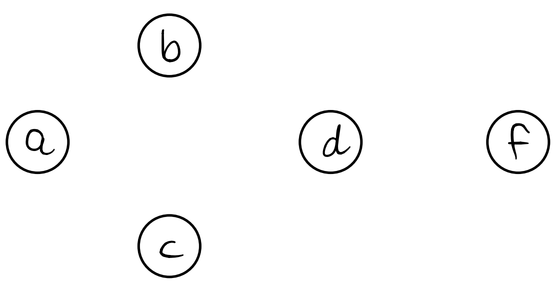
תכונה שלא משתנה

האינווריאנטה באלגוריתם היא שהצלעות ב-T הוא תת קבוצה של העץ פורש מינימלי בסופו של דבר.

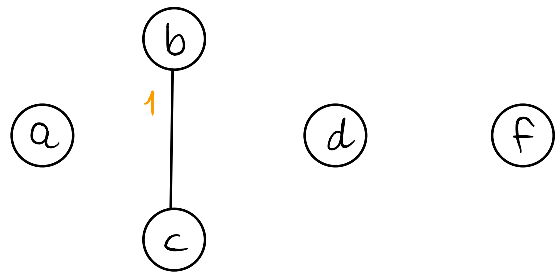
נציג שתי שיטות להוספת הצלעות בדרך שהצגנו.

שיטה ראשונה: נמצא את המינימום בכל פעם מהצלעות הקיימות, וע"מ לוודא שאין מעגלים נבדוק לפי שכנים האם סוגרים מעגל.
בצורה זו T יכול להיות יער ולא בהכרח שהצלעות יתחברו.

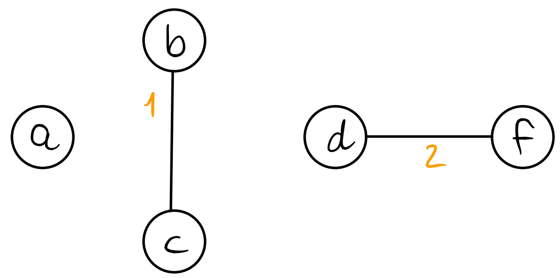
נראה באמצעות **דוגמה:**



Start: $T = \emptyset$

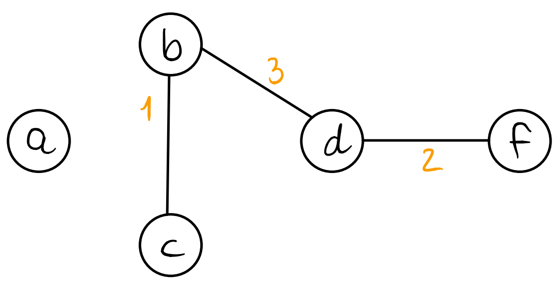


step 1: $T = \{(b,c,1)\}$

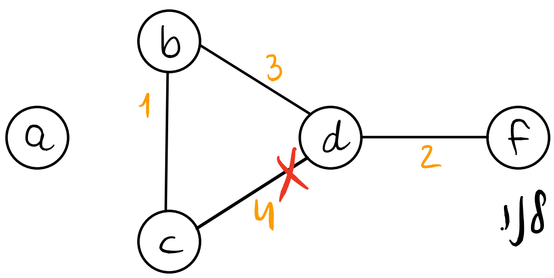


step 2: $T = \{(b,c,1),(a,f,2)\}$

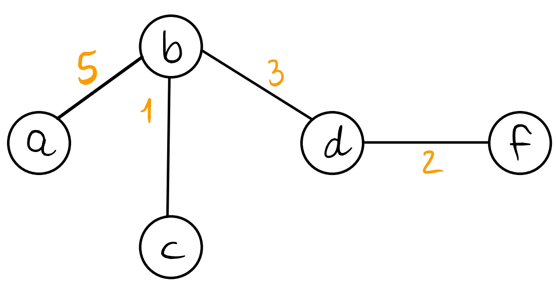
נשים לב שכרגע T הוא אכן יער
(מורכב משני עצים נפרדים)



step 3: $T = \{(b,c,1),(a,f,2),(b,d,3)\}$



אם נחשוב להוסיף את 4 (משקל מינימלי)
נשים לב ש - (c,d,4) כבר סוגר מעגל - לא טוב לנו!
c,d כבר נמצאים ברשימה.



step 4: $T = \{(b,c,1),(a,f,2),(b,d,3),(a,b,5)\}$

ומכאן והלאה כל צלע שנמצאו כבר תסגור מעגל ותפר את התנאים, לכן סיימנו!

נסמן ב- U = { כל הקודקודים שתפסנו } = קודקודים ב- T

וב- $V-U$ = { כל הקודקודים שעוד לא תפסנו } = קודקודים שלא ב- T

וכל השאר הן צלעות שנמתחות בין U ל- $V-U$.

בסוף האלגוריתם U יכיל את כל הקודקודים V, ו- $V-U$ המשלים יהיה ריק.

ולכן המטרה היא להעביר את כל הקודקודים מ- $V-U$ ל- U.

איך נעשה זאת?

1. נמתח צלע ראשונה.
2. נבחר אחת מהאופציות:

א. נבחר צלע מינימלית שמחברת בין U ל- $V-U$ שלא יוצרת מעגל. ← האלגוריתם של קרוסקל

ב. נבחר צלע מינימלית מ- $V-U$ (שחוצה את החתך) ← האלגוריתם של פרים

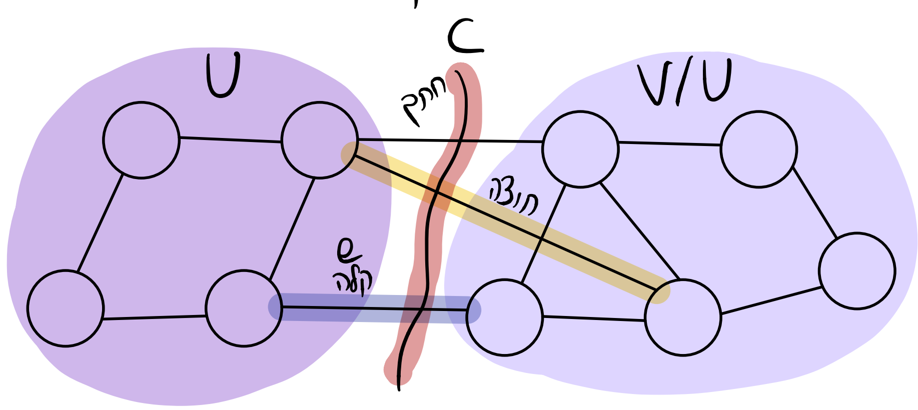
3. נעצור כאשר $V=U$.

בשני מקרים אלו נשמור על התכונות (ללא מעגל, מינימום)

הגדרות וטרמינולוגיה

* חתך $C=(U,V/U)$ של גרף לא מכוון $G=(V,E)$ הוא חלוקה של צלעות V.
* צלע $e=(v,u)$ חוצה את החתך C אמ"מ $u\in U$ ו- $v\in V/U$.
* צלע קלה e היא קשת שחוצה את החתך ומשקלה מינימלי!
* חתך C מכבד קבוצת צלעות $A\subseteq E$ אמ"מ אין צלע ב- A שחוצה את C.

לפי הצבעים של ההגדרות



צלע קלה

משפט 1: תהי $U\subseteq V$ ותהי $e=(u,v)$ הצלע עם המשקל המינימלי כאשר הקודקוד אחת הוא מ-U וקודקוד שני הוא מ-$V-U$. אזי קיים עץ פורש מינימלי T כך ש-$e\in T$.

כלומר, אם e חוצה את החתך, הוא בהכרח חלק מעץ פורש מינימלי.

הוכחה: יהי T עץ פורש מינימלי. אם $e\notin T$ ונוסיף אותו ל-T אזי ייווצר מעגל (כרגע T עץ)
ולכן בהכרח קיימת $e'=(v',u')$ כאשר $u\in U$ ו-$v\in V/U$.
נבחין כי $weight(e)<weight(e')$ מעצם הגדרת e (צלע קלה).
ומכאן שנוכל למחוק את e' ולהוסיף את e. במצב כזה $T'=\{T/\{e'\}\}\cup\{e\}$
כך ש-$weight(T')<weight(T)$ ולכן גם T' עץ פורש מינימלי, ו-$e\in T'$.

משפט 2: יהי $G=(V,E)$ גרף לא מכוון קשיר.
תהי הקבוצה A תת קבוצת צלעות של E שכלולה בעץ פורש מינימלי כלשהו T של G.
יהי $C=(U,V-U)$ החתך שמכבד את A (אין צלע של A שחוצה את C)
ותהי $e=(u,v)$ הצלע עם המשקל המינימלי כאשר הקודקוד אחת הוא מ-U והקודקוד שני הוא מ-$V-U$.
אזי e היא צלע בטוחה ל-A (הוא משמר את מבנה העץ של T (ללא מעגלים) ואת תנאי המשקל המינימלי)

אם כן - היא בטוחה

הוכחה: יהי T עץ פורש מינימלי, תהי e צלע קלה שחוצה את C. בנוסף, $A\subseteq T$ ו-$e\notin T$.
נבנה כעת עץ אחר T', ע"י הוספת e כך: $T'=T\cup\{e\}$. כעת נוצר מעגל.
לכן נוכל להסיר את מהמעגל ועדיין נשאר עם עץ פורש מינימלי, נוכל להסיר צלע e'
כאשר שחוצה את החתך C. $T'=T\cup\{e\}/\{e'\}$.
כעת נראה כי T' עץ פורש מינימלי, מכיוון ש-e צלע קלה שחוצה את C ו-e'
צלע שחוצה את e ו-$weight(e)<weight(e')$. הצלע e היא צלע בטוחה עבור A,

מכך ש- $A \subseteq T$ ו- $e \notin A$, $A \cup \{e\} \subset T'$ וגם $A \subseteq T'$. בעצם קיבלנו שמתקיים:

$$weight(T') = weight(T) - weight(e) + weight(e') \leq weight(T)$$

לכן בפרט T עץ פורש מינימלי ו- e בטוחה עבור A.

כעת נרצה למצוא את הצלעות הבטוחות ב- A בצורה יעילה.

נוכל לעשות זאת באמצעות אחד משני האלגוריתמים הבאים:

1. אלגוריתם קרוסקל Kruskal Algorithm

משפט 2

הקבוצה A יכולה להיות יער והצלע הבטוחה שנוסיף תמיד תהיה מינימלית.
בזמן התהליך (בסוף, לא)

2. אלגוריתם פרים Prim Algorithm

משפט 1

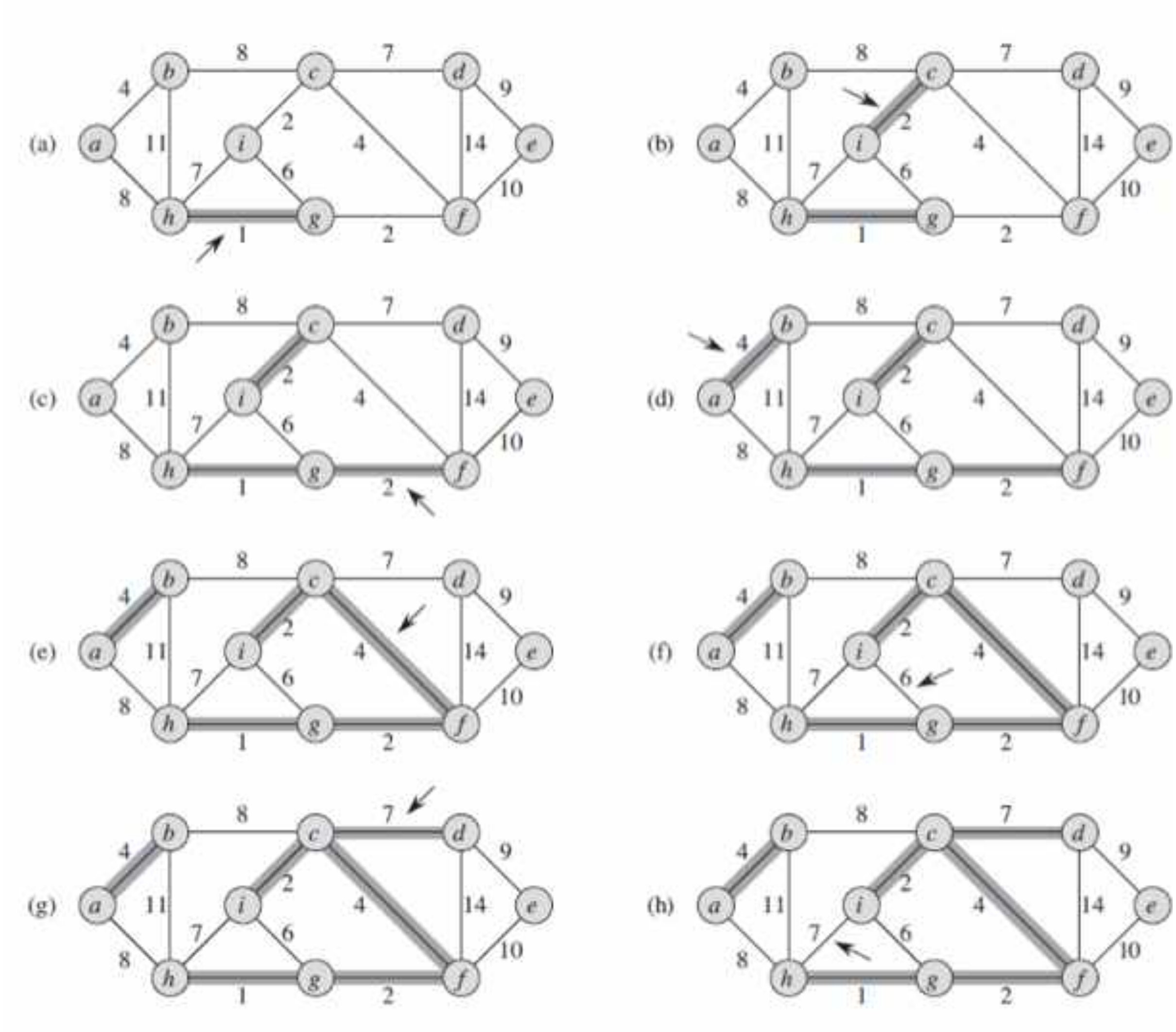
הקבוצה A היא עץ והצלע הבטוחה שנוסיף תהיה המינימלית שמחברת בין קודקוד ב- A לקודקוד שלא ב- A.

האלגוריתמים:

1. אלגוריתם קרוסקל Kruskal Algorithm

```
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
```

דוגמה:



נכונות אלגוריתם קרוסקל נובעת ממשפט 2 שהוכחנו.

סיבוכיות אלגוריתם קרוסקל:

תחילה נבצע מיון, מה שייקח $|E|\log|E|$.

לאחר מכן, נעבור על כל הקודקודים ונצטרך לבדוק את כל הצלעות ב-E, מה שייקח $|V|\cdot|E|$ ע"י איחוד קבוצות. בהמשך נראה אלגוריתם יעיל לאיחוד קבוצות ב-$O(1)$

ואז זמן הריצה עבור שלב זה יהיה $O(|E|)$.

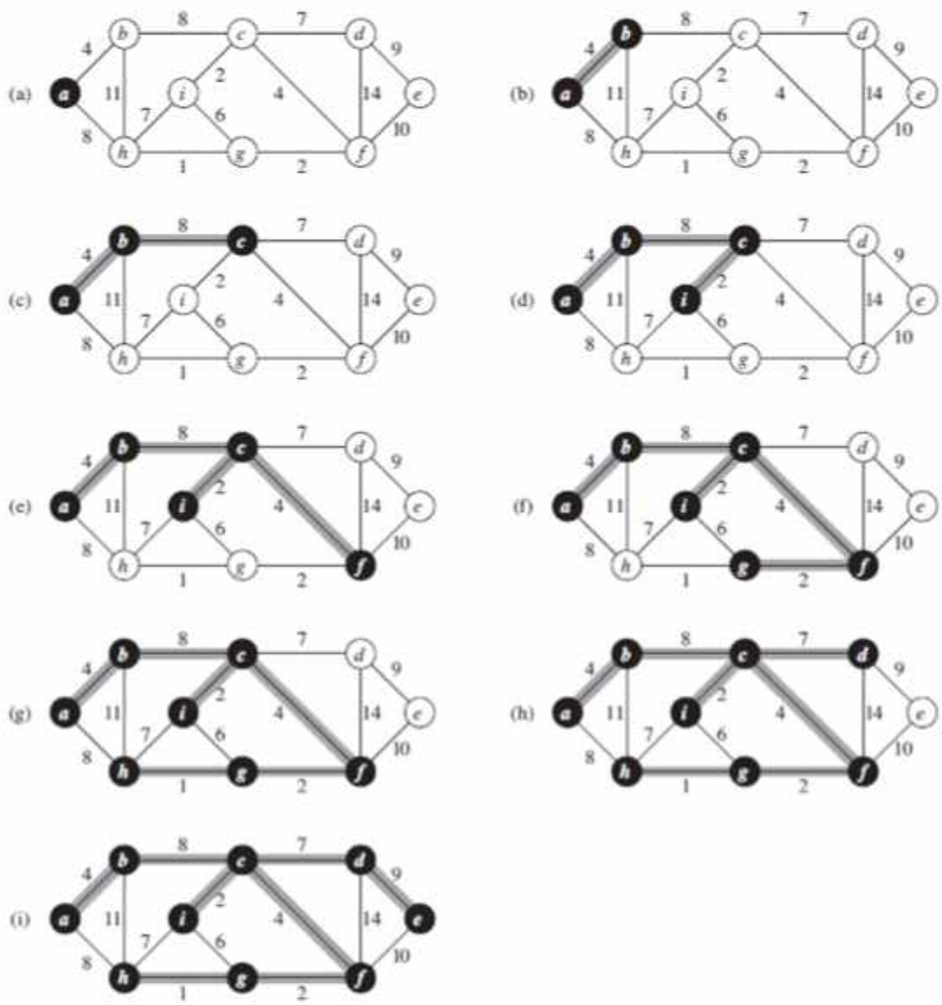
ומכאן נקבל סה"כ: $O(|E|\log|E|)$

2. אלגוריתם פרים Prim Algorithm

```
MST-PRIM(G, w, r)
for each u ∈ G.V
    u.key = ∞
    u.π = NIL
r.key = 0
Q = G.V                         // תור Queue
while Q ≠ ∅
    u = EXTRACT-MIN(Q)                              נבחר רק את השכנים
    for each v ∈ G.Adj[u]       → neighbor
        if v ∈ Q and w(u, v) < v.key
            v.π = u
            v.key = w(u, v)
```

דוגמה:

* מזכירות BFS



נכונות אלגוריתם פרים נובעת ממשפט 1 שהוכחנו.

סיבוכיות אלגוריתם פרים:

באופן כללי, הסיבוכיות תלויה במימוש הערימה.

בנייתה אורכת כ-$O(|V|)$ ובנוסף הוצאת המינימום תקח $O(\log|V|)$ לכל קודקוד ולכן נקבל בסה"כ: $O(|V|\log|V|)$.

לולאת ה-for מתבצעת $O(|E|)$ פעמים כי סכום אורכי רשימת השכניות הוא $2|E|$.

בדיקת השייכות לתור Q היא $O(1)$, והשמת הערך ב-key בשורה האחרונה שקולה ל-Decrease-Key של ערימה שראינו שלוקחת $O(\log|V|)$.

ולכן בסה"כ נקבל: $O(|V|\log|V| + |E|\log|V|) = O(|E|\log|V|)$