© 1995 A. Itai

גרפים לא מכוונים

• גרף $G = (V,E)$ ייקרא <u>לא-מכוון</u> אם

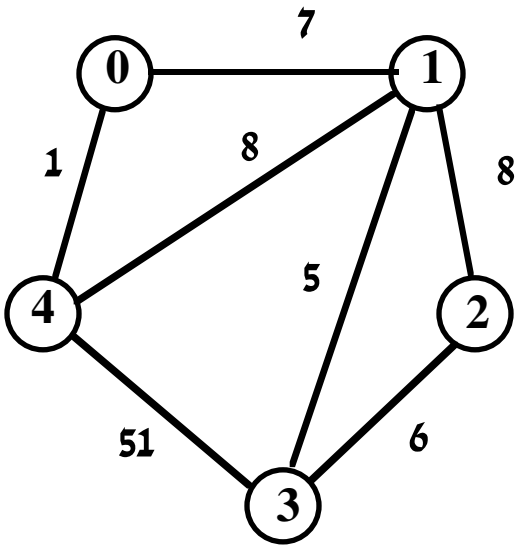
$$(u,v) \in E \quad \Leftrightarrow \quad (v,u) \in E$$

• גרף ייקרא <u>ממושקל</u> (weighted) אם לכל קשת $e \in E$ קיים משקל $w(e) > 0$.

ייצוג גרפים לא מכוונים ממשוקלים

א. מטריצת סמיכויות סימטרית

$$A[i,j] = \begin{cases} 0 & (i,j) \notin E \\ w(i,j) & \text{אחרת} \end{cases}$$

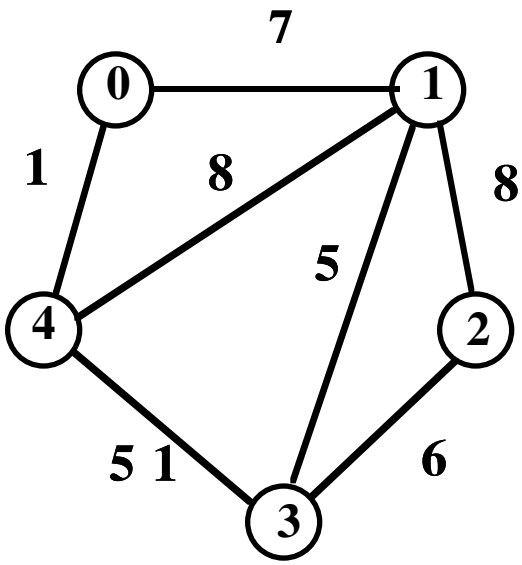


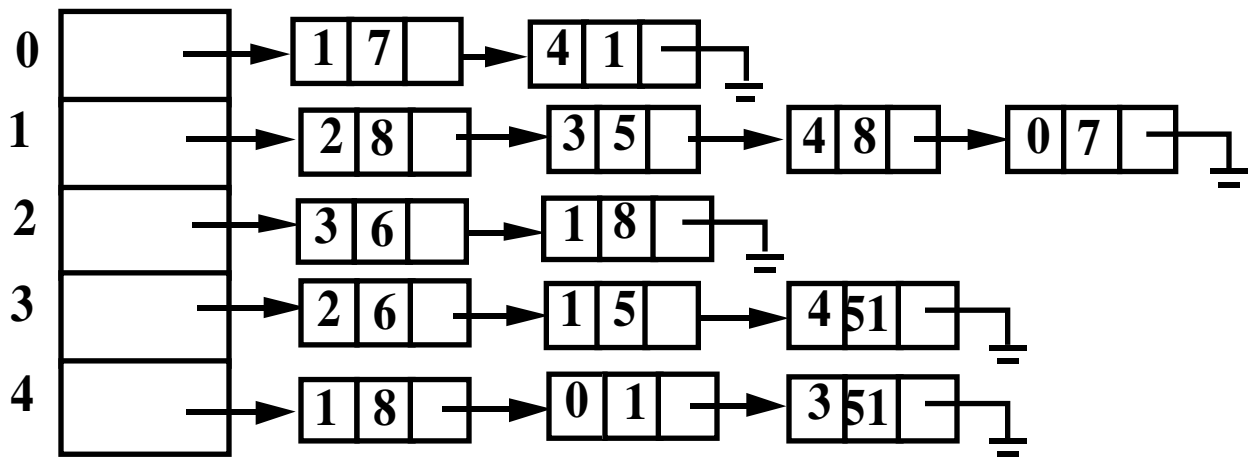
	0	1	2	3	4
0		7	0	0	1
1			8	5	8
2				6	0
3					51
4					

כמה איברים לשמור?
איך נמצא את ערך $w(i,j)$?

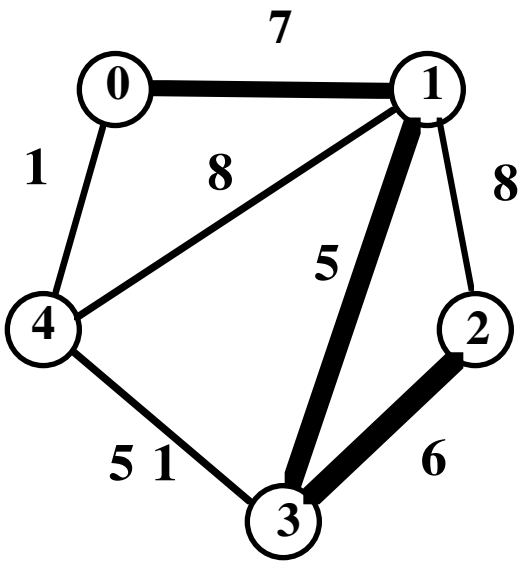
ב. רשימות סמיכויות

```
struct {
        int weight;
        int head;
        struct node  *next;
}  node;
```





מסלול סדרת צמתים $v_0,v_1,\ldots,v_k$ **כאשר** $(v_i,v_{i+1}) \in E$
$i \neq j \Rightarrow v_i \neq v_j$

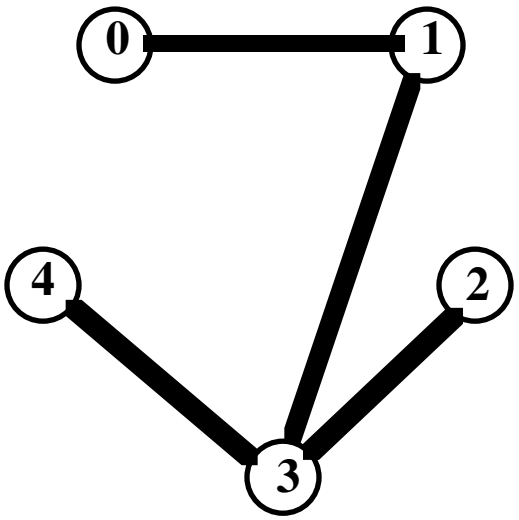


המסלול

(0, 1, 3, 2)

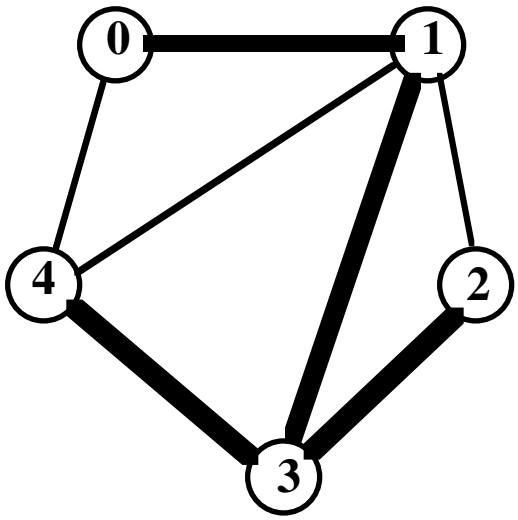
הגדרות

- **<u>מעגל</u> מסלול שבו** $v_0 = v_k$
- **<u>גרף קשיר</u> - גרף שבו בין כל שני צמתים קיים מסלול.**
- **<u>עץ</u> - גרף קשיר חסר מעגלים**

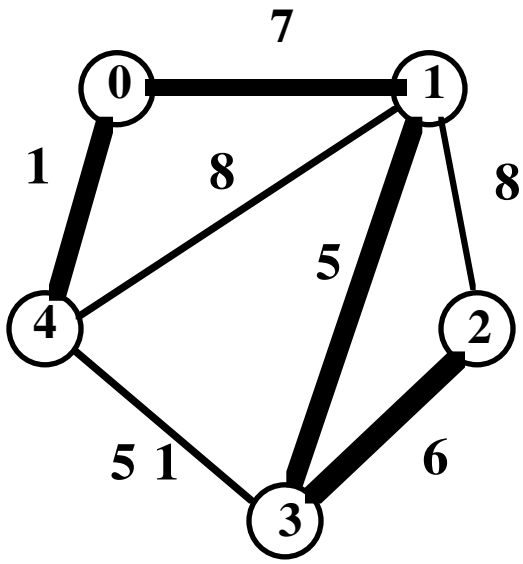


• עץ $T = (V_T, E_T)$ הוא <u>עץ פורש</u> של גרף $G=(V,E)$ אם

$$V = V_T \qquad E_T \subseteq E$$



• יהי G גרף ממושקל אז T <u>עץ פורש מינימום</u> אם הוא עץ פורש ומשקלו (סכום משקלות קשתותיו) הוא מינימלי.



אלגוריתם למציאת עץ פורש מינימום (Prim)

1. **<u>איתחול</u>: יהי** $s \in V$, $V_T = \{s\}$ $E_T = \varnothing$

2. **<u>כל עוד</u> קיימות קשתות שיוצאות מ- s:**

   2.1 **בחר את הקשת e הזולה ביותר שיוצאת מ- s**

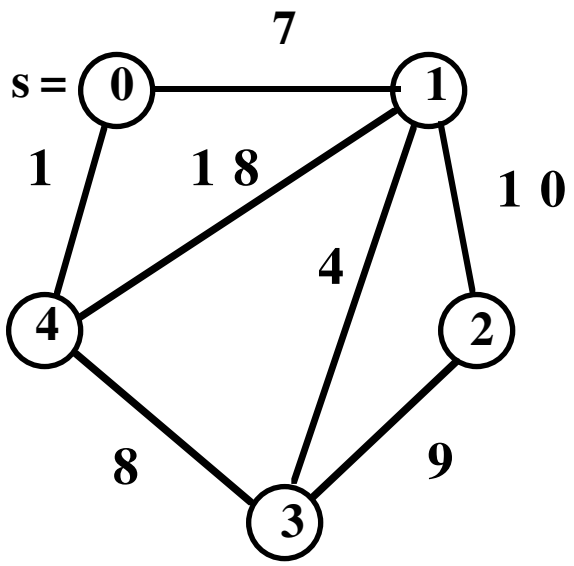
   2.2 **<u>אם</u>** $e = (s, j)$ **הוסף אותה (עם הקצוות המקוריים שלה) ואת הצומת j לעץ.**

   2.3 **עדכן את G:**

   כלומר, סלק את j וכל הקשתות הנוגעות בה מ- G,
   ולכל קשת (j, i) $s \neq i$ **בצע**
   <u>אם</u> הקשת (s, i) **לא נמצאת ב-G**
   הוסף אותה עם משקל $w(j, i)$;
   <u>אם</u> (s, i) **נמצאת ב-G ו-** $w(j, i) < w(s,i)$
   הקטן את משקלה: $w(s,i) = w(j, i)$.

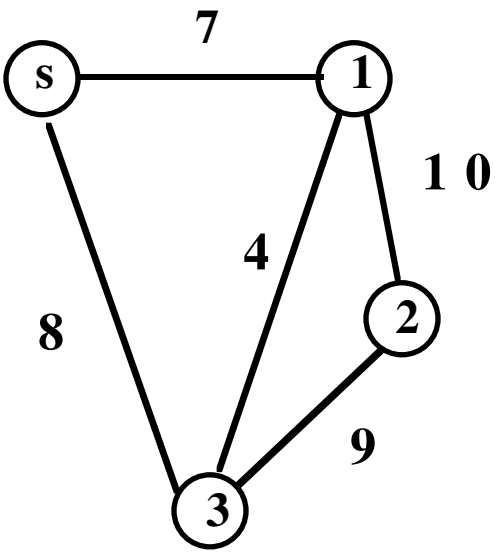
<u>שאלה</u>: בסעיף 2.2, כיצד נדע מהם הקצוות המקוריים של הקשת?

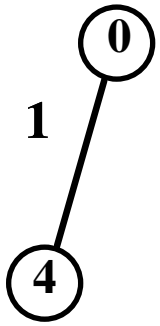
G

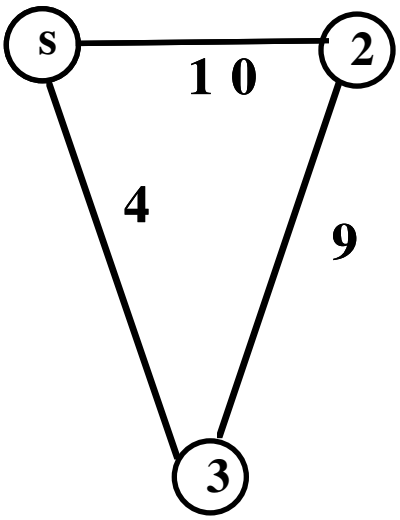


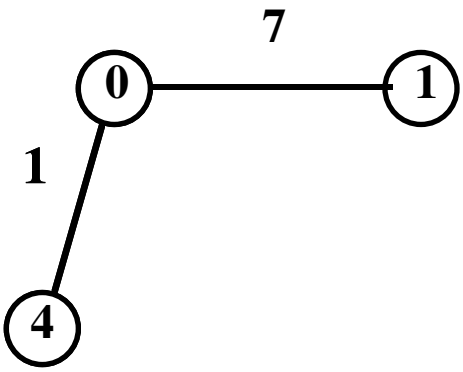
T

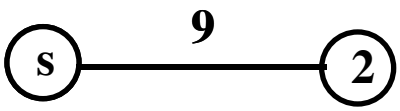
דוגמא

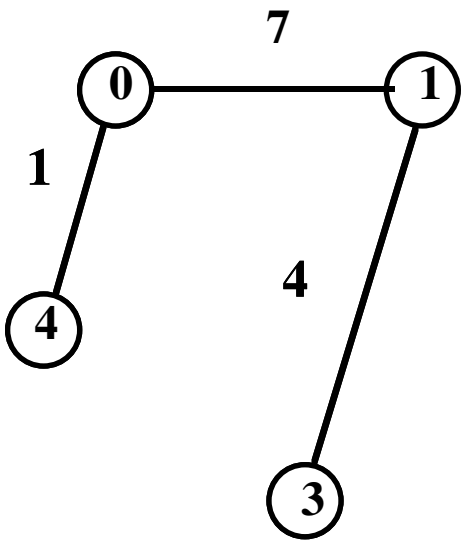




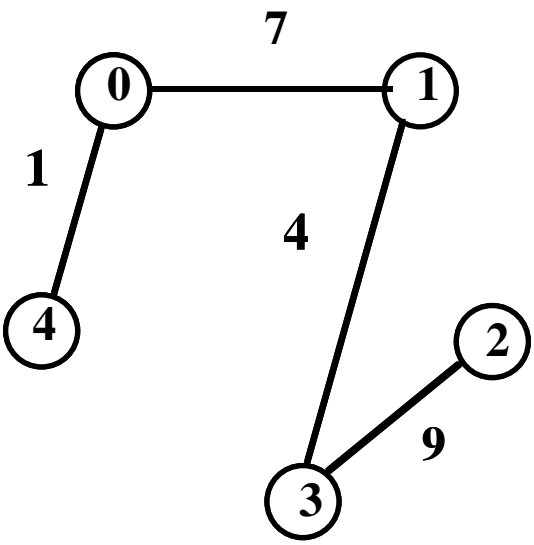

s
2
1 0
4
9
3


7
0
1
1
4


9
s
2


7
0
1
1
4
4
3

s


7
0
1
1
4
4
2
9
3

מבני נתונים

<u>G במטריצת סמיכויות</u>: + ייצוג לצמתי T.
כדי למצוא את הקשת הזולה ביותר שיוצאת מ-s
נעבור על השורה ה-s -- זמן O(n).
סילוק צומת i: נמזג את השורה ה-i והשורה ה-s -- זמן O(n)
כל האלגוריתם: זמן $O(n^2)$.

	0	1	2	3	4
0	-	7	-	-	1
1	7	-	10	4	18
2	-	10	-	9	-
3	-	4	9	-	8
4	1	18	-	8	-

origin:	0	-	-	-	0

	0	1	2	3
0	-	7	-	8
1	7	-	10	4
2	-	10	-	9
3	-	4	9	-

origin:	0	-	4	

	0	2	3
0	-	10	4
2	10	-	9
3	4	9	-

origin:	0	-	4

<u>G ברשימת סמיכויות:</u>

סיבוכיות צעד:
כדי למצוא את הקשת הזולה ביותר
-- נעבור על כל רשימת הסמיכויות של s .

זמן deg(s)=O(n)

כל שאר הפעולות -- זמן O(n).

כל האלגוריתם: $O(n^2 + m) = O(n^2)$.

שיפור

אם נחזיק את רשימת הסמיכויות של s במערך
– A[i] = w(s,i)

בהוצאת צומת j אפשר לעדכן מערך זה בזמן deg(j).
עכשו צוואר הבקבוק הוא מציאת המינימום -- O(n) .

לכן נשמור את משקולות הקשתות שיוצאות מ- s <u>בערימה</u>.

סיכום המבנה:

מבנה המחזיק את הקשתות:

```
typedef struct { float weight;
                 int side1, side2} EDGE;
EDGE edges[m];
```

רשימות הקרויות (רשימות הקשתות של צומת):

```
typedef list_node struct { EDGE *edge;
                           list_node *next} LIST_NODE;
LIST_NODE incidence_list[n];
```

הערימה (מחוונים ל-edges):

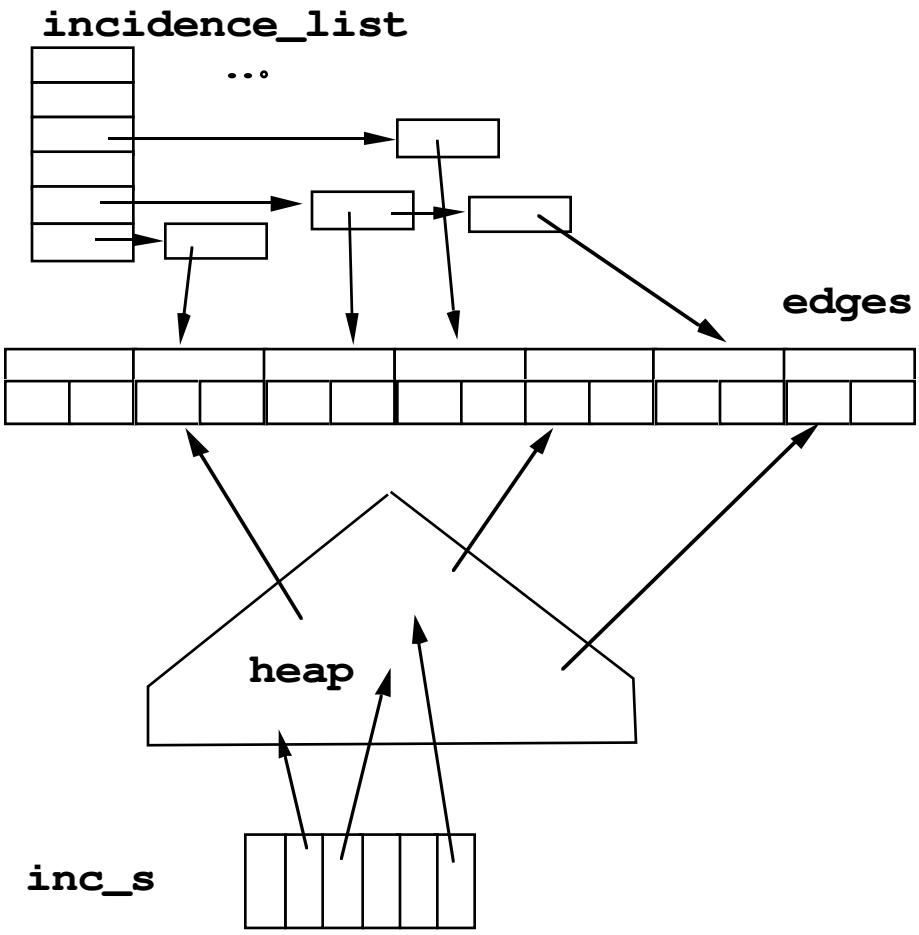
```
EDGE *heap[n];
```

שכני s (מחוונים ל-heap):

```
EDGE **inc_s[n];
```

רשימות הסמיכויות של העץ
(מקבל ערך NULL עבור צומת שאינו בעץ)

```
LIST_NODE tree[n];
```



הפעולות:

מציאת קשת מינימלית שיוצאת מ-s : min	**זמן**	**O(1)**
הוצאת קשת del_min	**זמן**	**O(log n)**
הוספת קשת	**זמן**	**O(log n)**
הקטנת משקל קשת: decrease ע"י	**sift_up**	**O(log n)**

שימו לב:
פעולות הערימה מחייבות עדכון inc_s בזמן החלפת צומת באביו.

כיון שכל קשת עלולה להכנס לערימה או לגרום לעדכון לכל היותר פעם אחת,
הזמן הכולל O(m log n)

- **מתי השיפור משפר?**

<u>הערה</u>: אפשר להוריד את זמן האלגוריתם ל- O(m +n log n)
ואפילו פחות, ע"י שימוש במבנה נתונים מתוחכם יותר.
(מבני נתונים 2).