

# תכנון וניתוח אלגוריתמים המשך הרצאה 12

---

מפרט נוסף

לייצוג רב מקושר של גרף





❖ השיטה הרביעית לייצוג גרף באמצעות מבנה רב מקושר

❖ עד כה הצענו שלוש חלופות לייצוג גרף ובכל חלופה הנחנו שמספר הקודקודים בגרף ידוע מראש.

❖ בשיטה חלופית זו נניח שאיננו יודעים מראש מספר הקודקודים בגרף וגם מספר הקשתות בגרף.

❖ לאור זאת עלינו להשתמש במבנה מקושר והקצאת הצמתים ושחרורם תתיבצע לפי הצורך ועלינו לבנות מבנה רב מקושר .



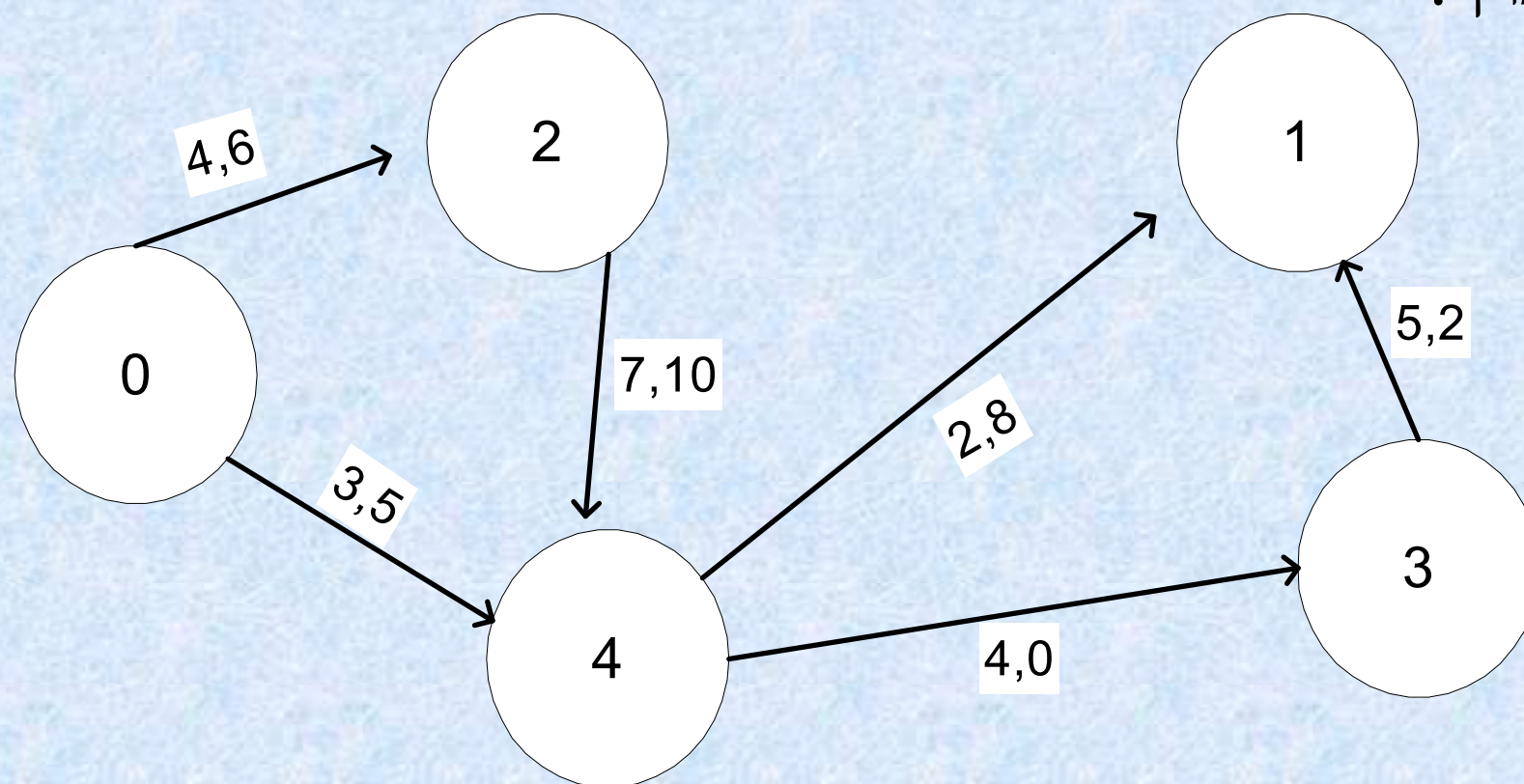


❖ בהמשך לחלופה השלישית, מאחר שאנו לא יודעים מראש את מספר הקדקודים בגרף נמיר את המערך, אשר היה מייצג את קודקודי הגרף, לרשימה מקושרת של קדקודי הגרף.

❖ באמצעות הגרף הבא ננסה לתאר ולייצג את הגרף בעזרת מבנה רב מקושר.



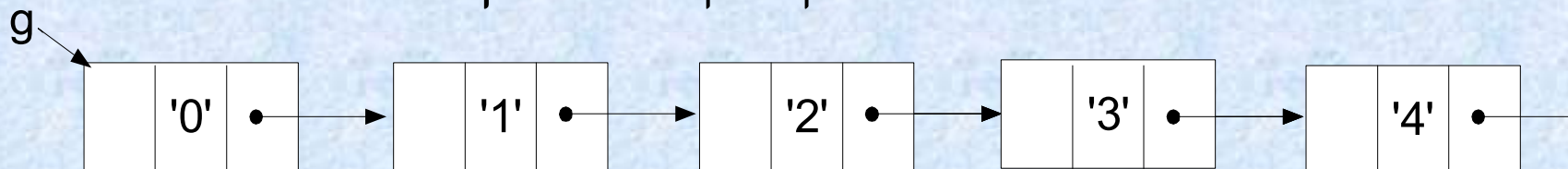
הגרף: 







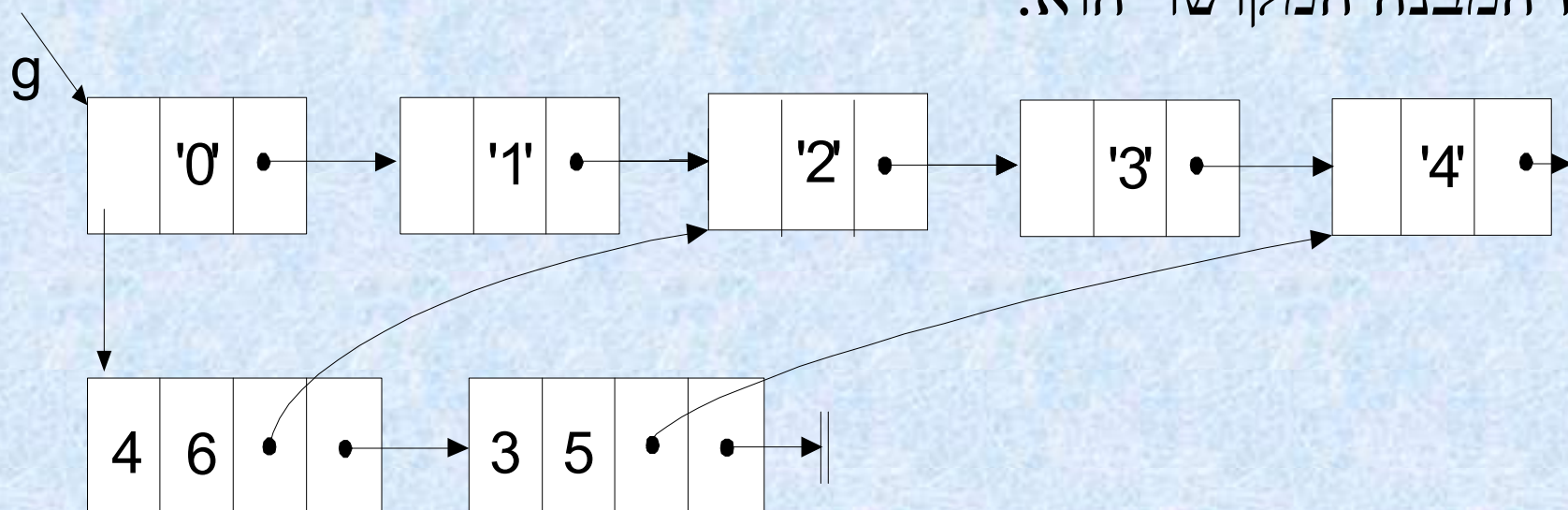
- ◆ גם בחלופה זו לא מייחסים מידע לקודקודים
- ◆ אך כן מאפשרים לייחס משקולות לקשתות.
- ◆ באיור רואים שלכל קשת מייחסים שתי משקולות  
( בשלב זה לא חשוב מה משמעותן ).
- ◆ כעת נציג בפרוטרוט את מבנהו המקושר של הגרף.
- ◆ הרשימה הבאה מייצגת את קודקודי הגרף :





מאחר שמקודקוד 0 יוצאות שתי קשתות  $\langle 0,2 \rangle$  ,  $\langle 0,4 \rangle$  ♦

אז המבנה המקושר הוא: ♦



צמתים המייצגים ♦

קשתות היוצאות מקדקוד 0. ♦





❖ כל צומת המייצג קשת  $\langle x, y \rangle$  מכיל את :

❖ המידע שמיוחס לקשת.

❖ מצביע לצומת המייצג צומת  $y$  בגרף בו נכנסת הקשת.

❖ מצביע לצומת המייצג את הקשת הבאה היוצאת מצומת  $x$ .

❖ באיור האחרון מקודקוד 0 יוצאות שתי קשתות: אחת

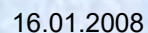
מהן נכנסת לקודקוד 2 והשנייה לקודקוד 4.



מקודקוד 1 לא יוצאות קשתות.

מקודקוד 2 יוצאת קשת לצומת 4.

לכן המבנה המקושר יראה כך:





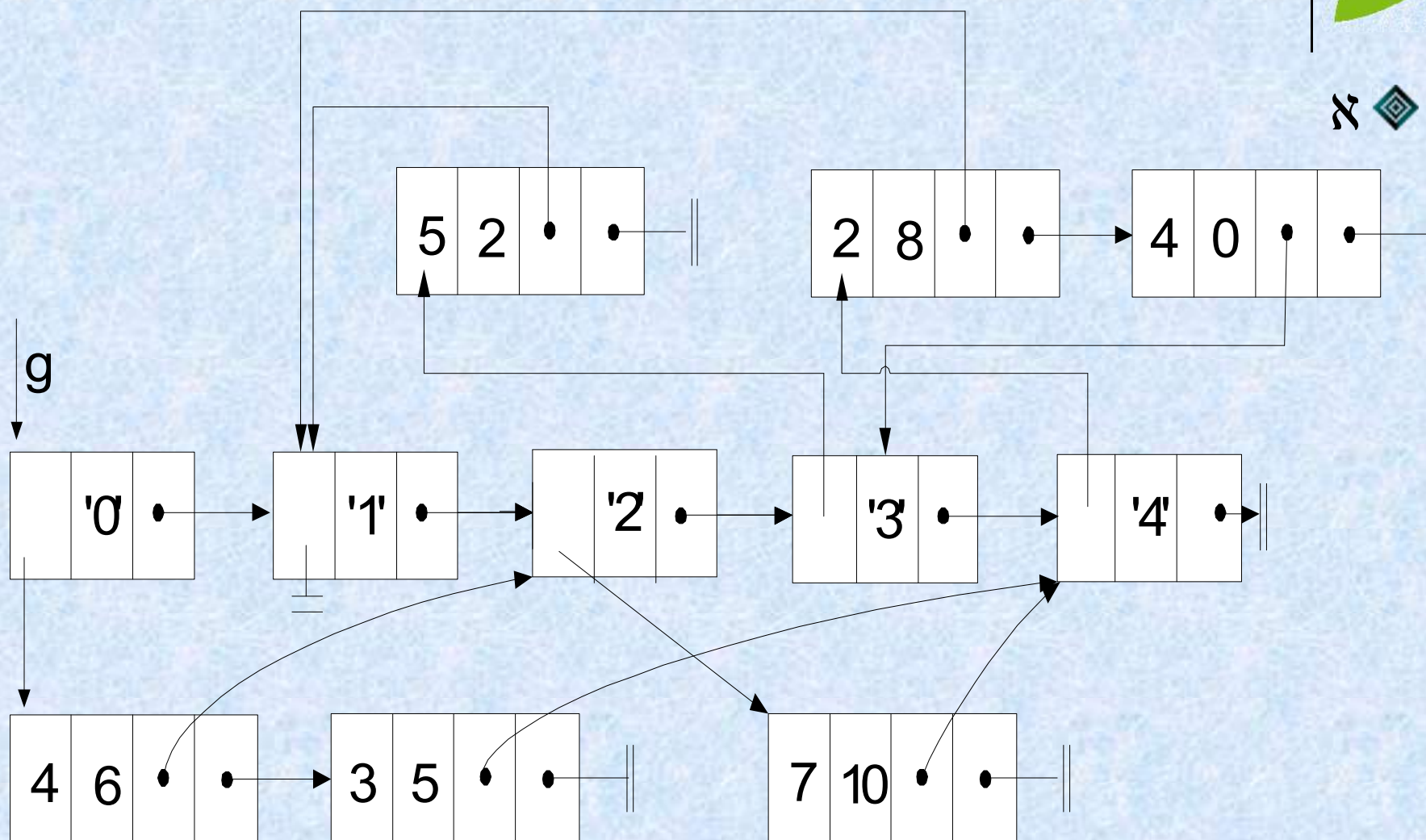


◆ עתה נצרף למבנה המקושר את המידע אודות יתר הקשתות.

◆ מקודקוד 4 יוצאות 2 קשתות ל-1 ול-3

◆ ומקודקוד 3 לקודקוד 1.

◆ לכן המבנה המקושר יראה כך:







❖ בשיטה חלופית זו קבוצת צמתי הגרף מיוצגת על ידי רשימה ליניארית מקושרת וכל צומת ברשימה מקושרת זו מייצג צומת בגרף.

❖ קבוצת הקשתות מיוצגת על ידי רשימת-שכנות (סמיכות / קשירות), וכל צומת ברשימת סמיכות מייצג קשת בגרף.



במבנה נתונים זה לכל צומת  $x$  המייצג קודקוד בגרף  
שלושה שדות ואלו הם:

info - שדה מידע ( למשל מספרו של קודקוד בגרף ).

Verticelink – שדה מצביע לצומת המייצג קודקוד  
בגרף, אם יש קודקוד כזה בגרף.

Arcpoint – שדה מצביע לרשימת צמתים המייצגת  
את הקשתות היוצאות מקודקוד מסוים  $x$  בגרף.





◆ לכל צומת, המייצג קשת כלשהי  $\langle x, y \rangle$  בגרף, שלושה שדות ואלו הם:

◆ Arcinfo – שדה המשקולות המיוחדות לקשת (שדה זה הינו רשומה).

◆ Verticepoint – שדה מצביע לצומת המייצג קודקוד  $y$  בגרף

◆ Arclink – מצביע לצומת המייצג את הקשת הבאה

◆ היוצאת מקודקוד  $x$  בגרף, אם בכלל יש קשת כזו.



יש לשים לב ששדה המידע arcinfo של קשת לא בהכרח  
זהה למבנה של שדה המידע info של צומת בגרף.

מכאן ניתן להסיק שבמבנה רב מקושר יש לנו שני סוגי  
צמתים:

סוג אחד של צומת שמייצג קודקוד בגרף ,

וסוג שני של צומת שמייצג קשת בגרף.

להלן הצהרות הדרושות :





/\*טיפוס רשומת המידע של קודקוד בגרף\*/

typedef struct vert

{

:

} vert\_info;

/\*טיפוס רשומת המידע של קשת בגרף\*/

typedef struct

{

:

} arc\_info;



/\* טיפוס רשומת צומת המייצג קודקוד בגרף \*/

```
typedef struct vertice  
{  
    vert_info info ;  
    struct vertice *verticelink ;  
    struct arc *arcpoint ;  
}node_vertice , *ptr_vertice ;
```





**/\* טיפוס רשומת צומת המייצג קשת בגרף \*/** ◆

◆ **typedef struct arc**

◆ **{**

◆ **arc\_info arcinfo ;**

◆ **struct vertice \*verticepoint ;**

◆ **struct arc \*arclink ;**

◆ **}node\_arc , \*ptr\_arc ;**



◆ ptr\_vertice g ; /\* כרזה על גרף \*/

◆ g = NULL; /\* יצירת גרף ריק \*/

◆ לאור ההחלטה לממש את הגרף על ידי שימוש במבנה רב מקושר עתה נציג את הפונקציות /שגרות למימוש הפעולות הבסיסיות שהזכרנו במפרט.

◆ נבצע את המימוש ישירות בסביבת העבודה (שפת C) תוך שימוש בייצוג המקושר.





◆ פונקציה/שגרה לפעולה – הוספת קשת בגרף g מקודקוד x לקודקוד y.

◆ void join(ptr\_vertice x, ptr\_vertice y, ptr\_vertice g);

◆ { ptr\_arc p , q ;

◆ המטרה לרדת לרשימת הקשתות היוצאות מקודקוד x.

◆ אם ברשימה זו קיימת קשת מקודקוד x ל-y אזי לא מוסיפים קשת חדשה,

◆ אחרת נוסיף את הקשת החדשה  $\langle x, y \rangle$ .



❖ מאחר שהמטרה להוסיף אלמנט חדש לרשימה – כידוע  
"לפי כלל ברזל" נטייל על פני רשימת הקשתות בעזרת  
זוג מצביעים הנעים במקביל על פני הרשימה.

❖ for

❖  $(q = \text{NULL}, \quad p = x \rightarrow \text{arcpoint};$   
 $p \&\&(p \rightarrow \text{verticepoint} \neq y);)$

❖ {  $q = p ; \quad p = p \rightarrow \text{arclink};$

❖ }

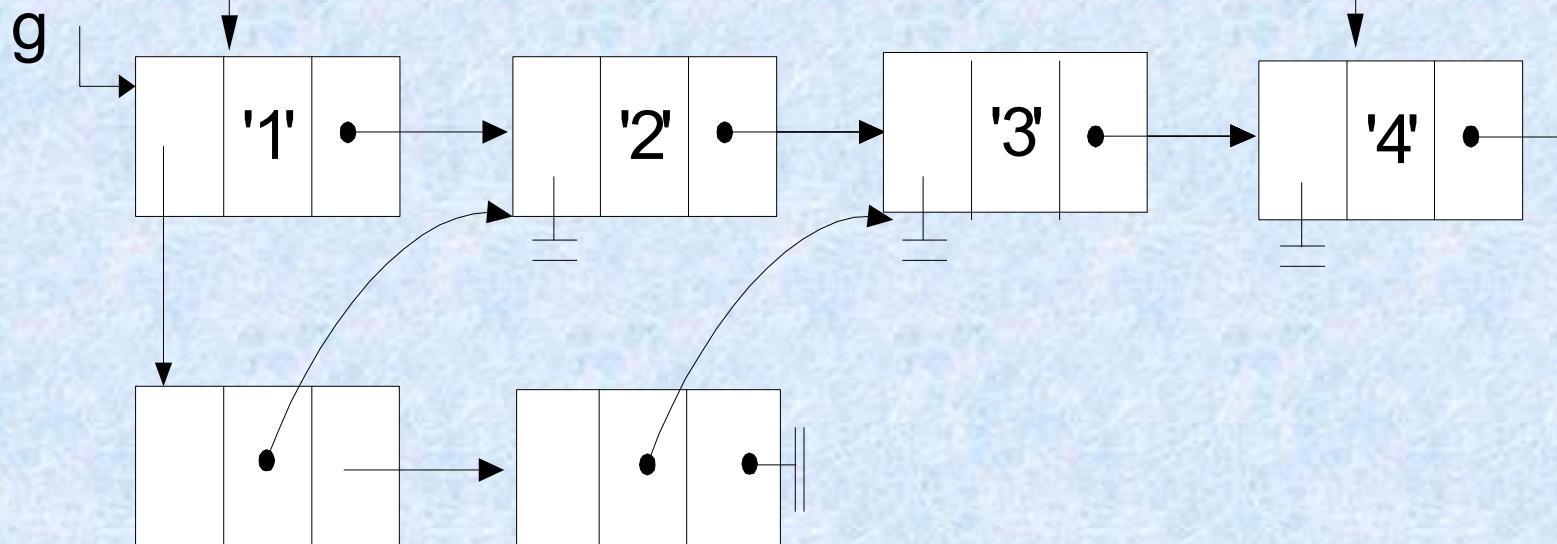




- ◆ if ( $p == \text{NULL}$ ) /\* לא קיימת הקשת המבוקשת \*/
- ◆ {  $p = \text{malloc}(\text{sizeof}(\text{struct arc}))$ ;
- ◆  $p \rightarrow \text{vertexpoint} = y$ ;  $p \rightarrow \text{arclink} = \text{NULL}$ ;
- ◆ האם הקשת החדשה  $\langle x, y \rangle$  מתווספת if ( $q == \text{NULL}$ )
- ◆ לראש רשימת הקשתות היוצאות מ- $x$   $x \rightarrow \text{arcpoint} = p$ ;
- ◆ else  $q \rightarrow \text{arclink} = p$ ;
- ◆ }



♦ (\*) עתה נראה את שלבי האלגוריתם בעזרת הגרף הבא:



♦ קל לראות שבגרף זה קימות רק שתי קשתות והן :

♦  $\langle 1, 3 \rangle$  ו  $\langle 1, 2 \rangle$ .

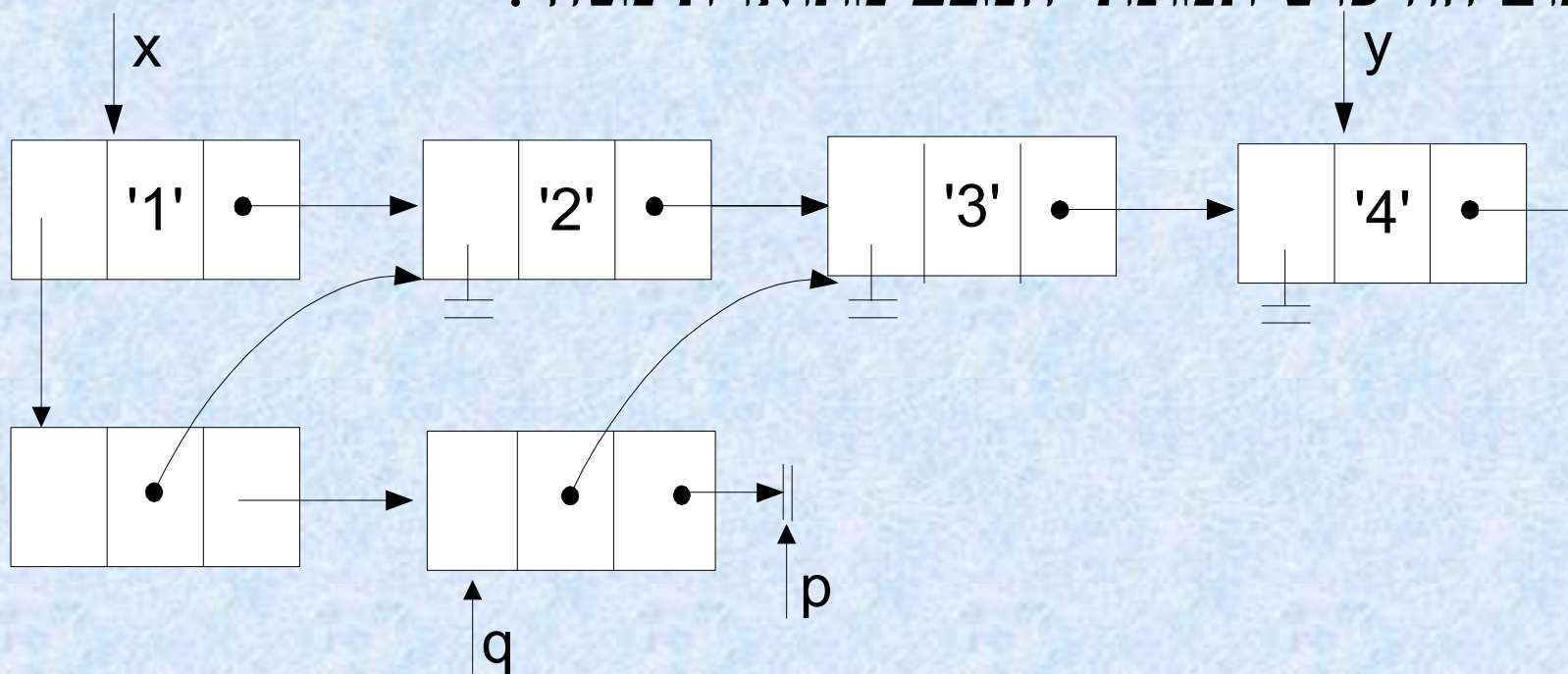






ב.תהליך חיפוש: האם קיימת קשת  $< 1, 4 >$  בגרף ?

בתום החיפוש תמונת המצב מתוארת מטה :



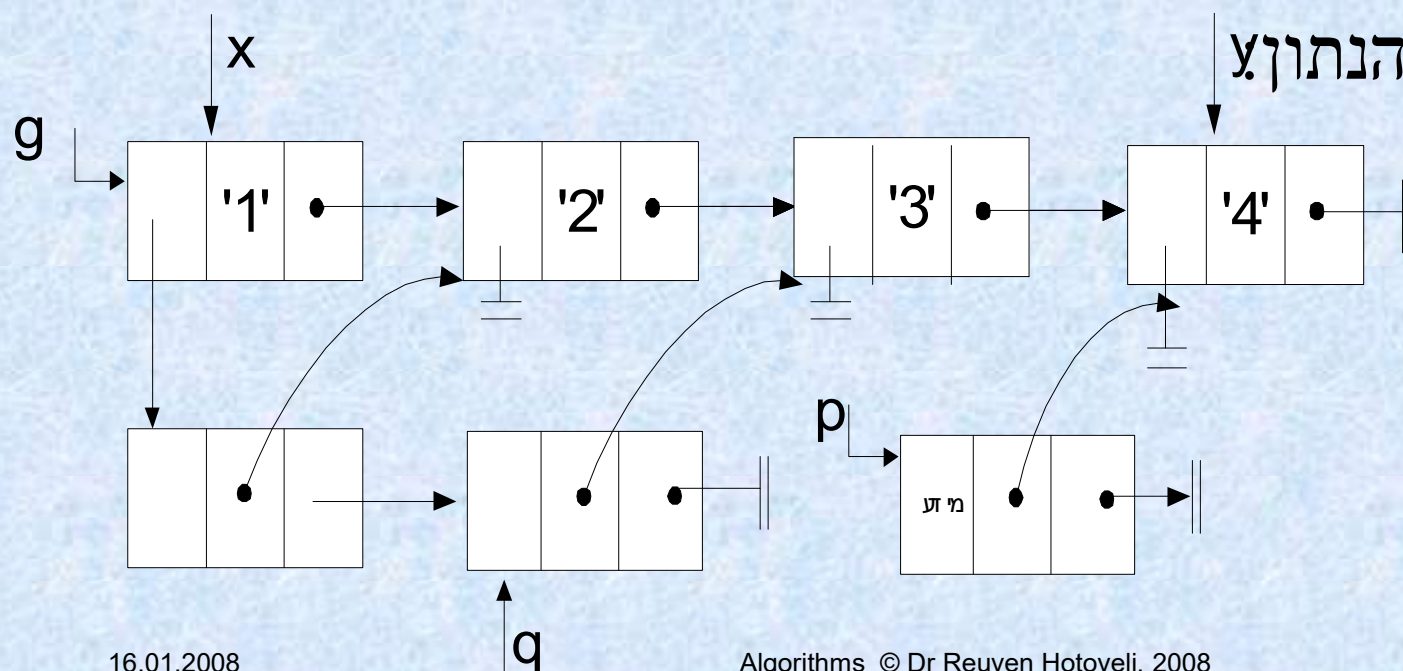




ג. בתום החיפוש מסתבר שקשת  $< 1, 4 >$  לא קיימת. לכן נרצה להוסיף אותה.

תחילה נבצע שלושה צעדים ראשוניים המתוארים

באלגוריתם הנתון:

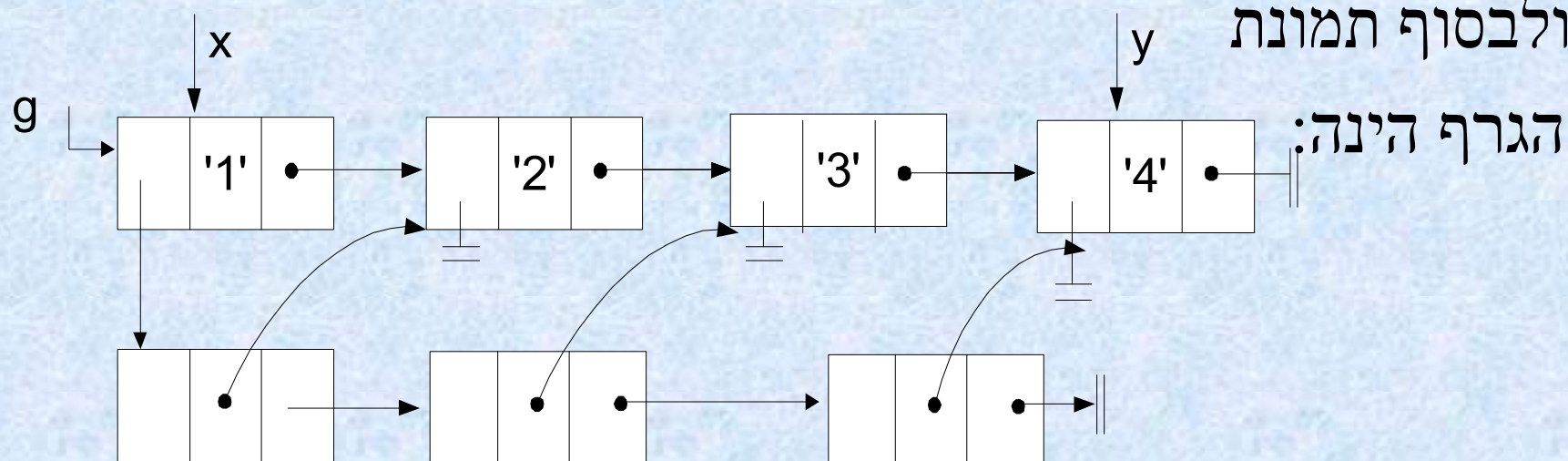




ד. בסוף צריך לחבר את הקשת  $< 1, 4 >$  המוצבעת על ידי  $p$  לרשימת הקשתות היוצאות מקודקוד 1.

מאחר ו-  $q \neq \text{NULL}$  נבצע :  $q \rightarrow \text{arclink} = p$  ;

ולבסוף תמונת



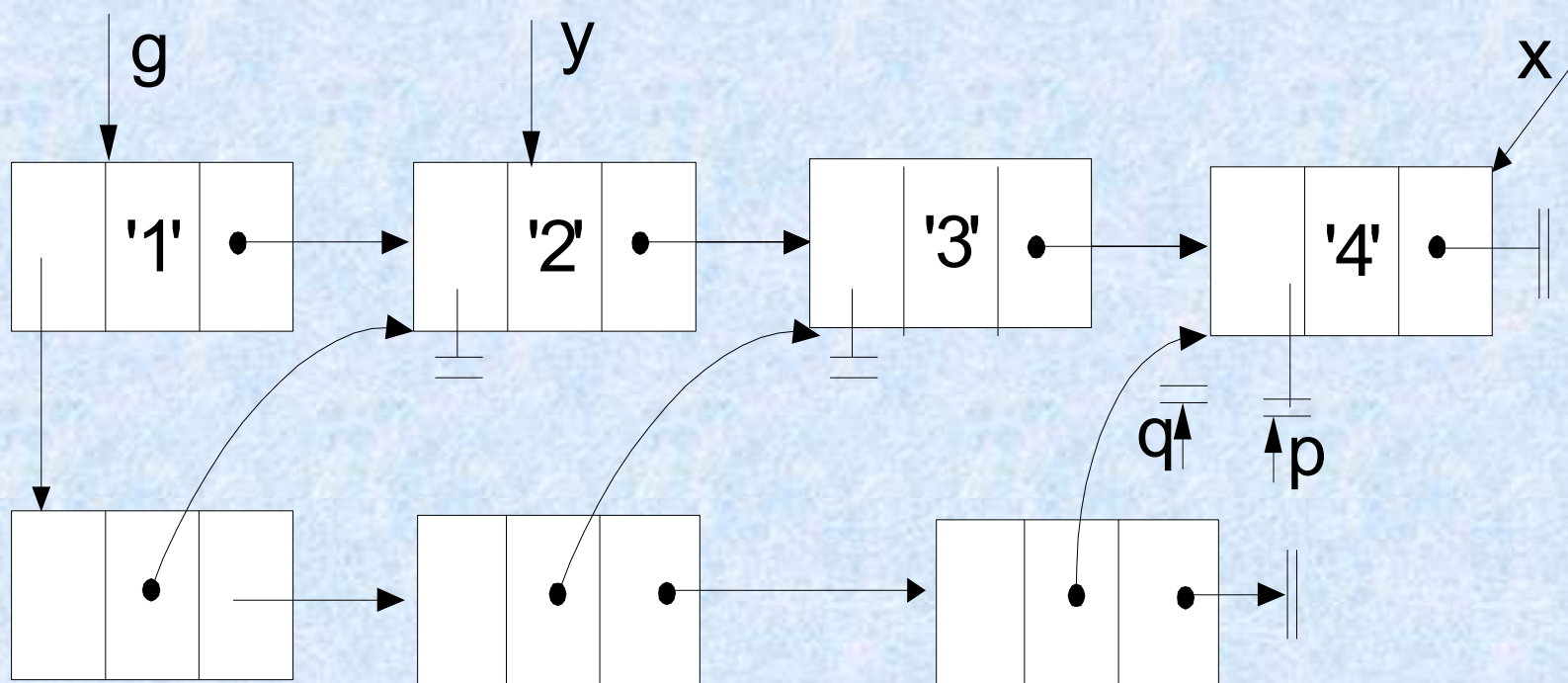




◆ (\*) עתה נראה את שלבי האלגוריתם, כאשר נרצה להוסיף קשת מקודקוד 4 לקודקוד 2 בגרף שקיבלנו לאחר הוספת הקשת  $\langle 1, 4 \rangle$ .

◆ כאמור מטרתנו להוסיף קשת  $\langle 4, 2 \rangle$ .

◆ א.תחילה  $p$  ו  $q$  יצביעו כנדרש לרשימת הקשתות היוצאות מקודקוד 4 כמתואר מטה:





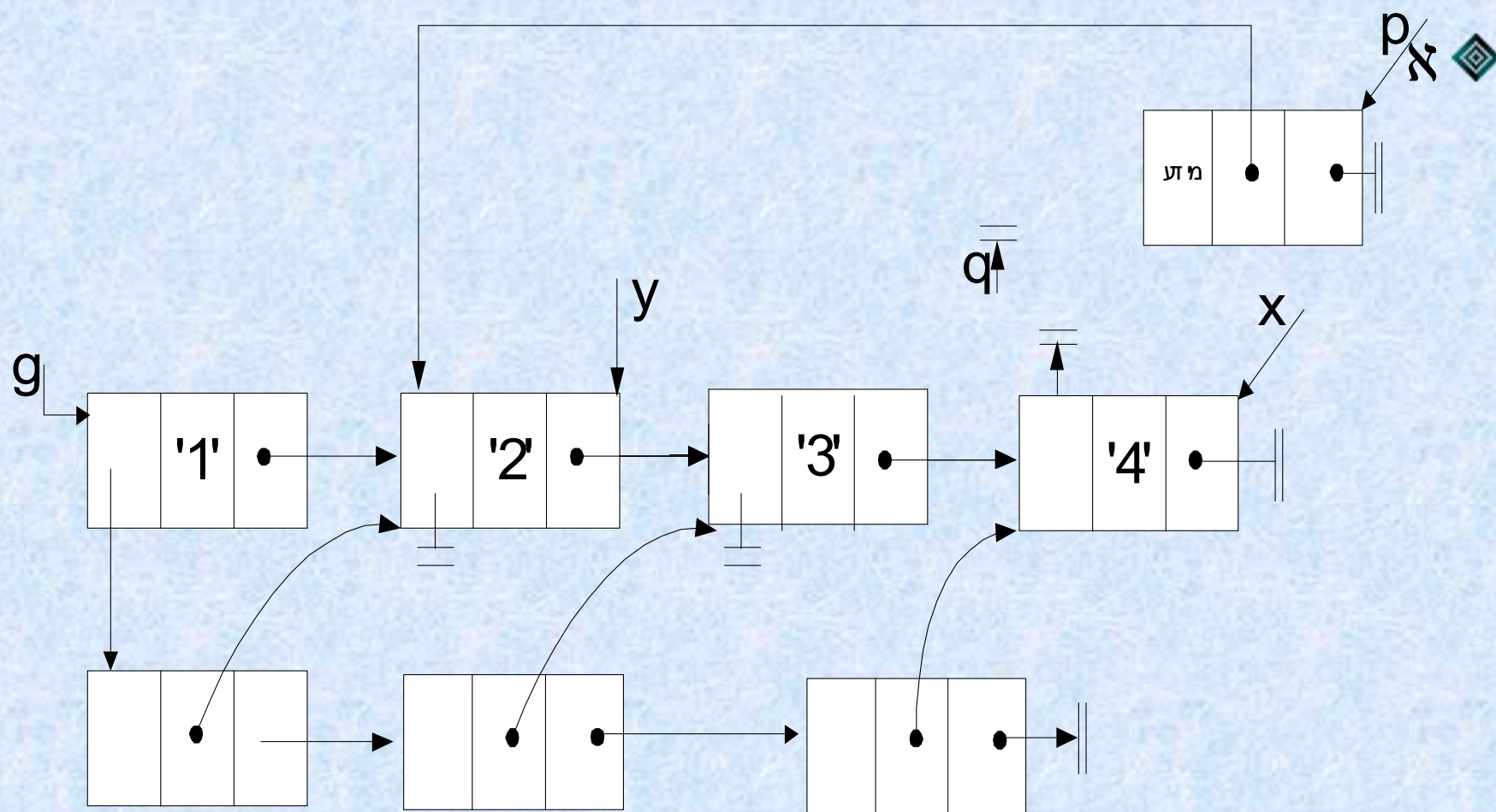


ב. בתום תהליך החיפוש אחר הקשת  $\langle 2, 4 \rangle$  תמונת המעקב תהיה אותה תמונה כמו ב- א'.

ג. בתום החיפוש מסתבר שקשת  $\langle 2, 4 \rangle$  לא קיימת לכן נרצה להוסיף אותה.

תחילה נבצע שלושה צעדים ראשוניים המתוארים באלגוריתם הנתון. לכן נקבל:

# Algorithm for the construction of a Huffman tree





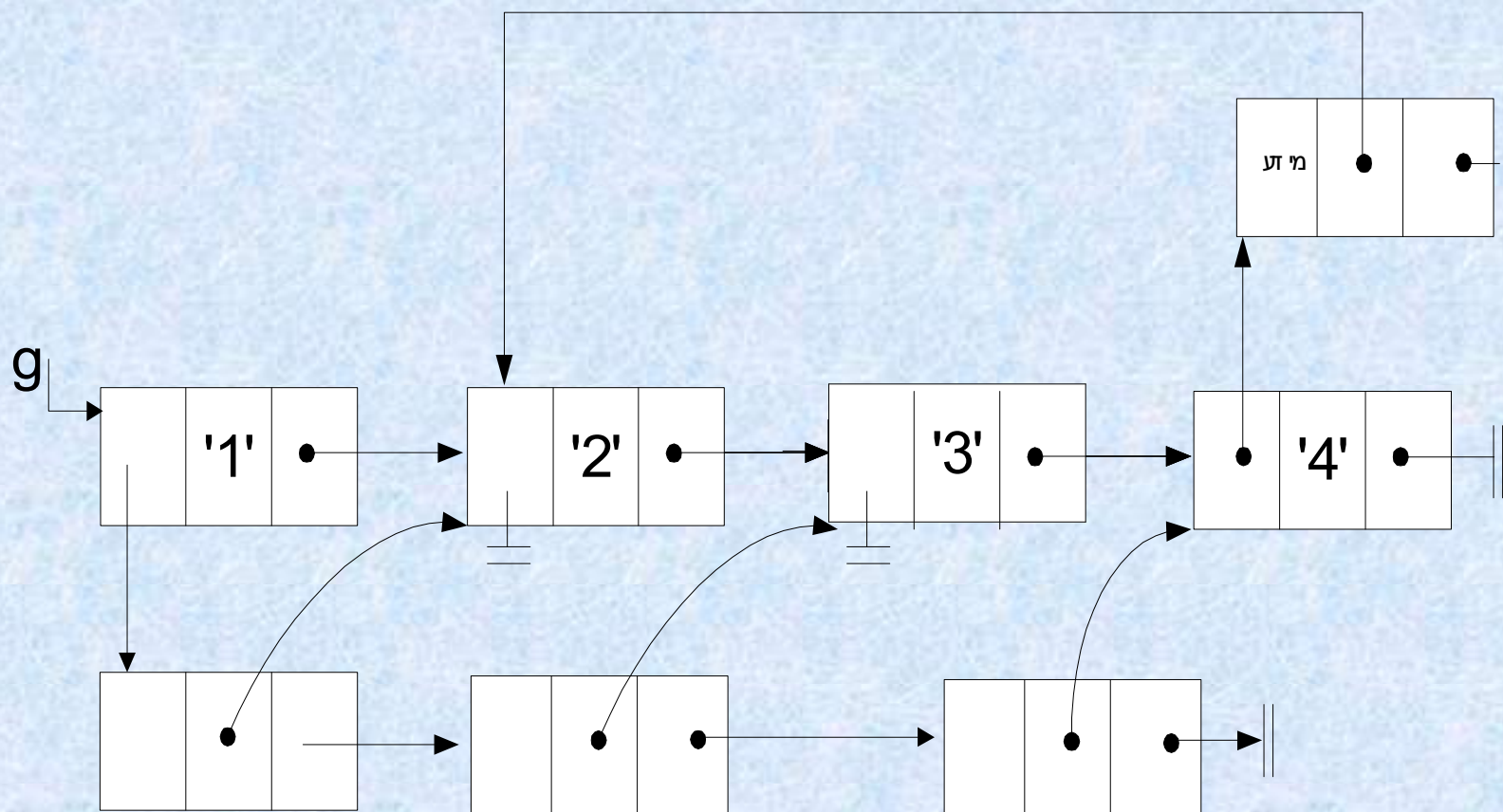


ד. בסוף צריך לחבר את הקשת  $< 4, 2 >$   
המוצבעת על ידי  $p$  לרשימת הקשתות היוצאות  
מקודקוד 4.

מאחר ש-  $q \Rightarrow \text{NULL}$  נבצע את המשפט הבא:

$x \quad \text{arcpoint} = p ;$

ובסוף תמונת הגרף הינה:







◆ עֵתָה נִמְמַשׁ אֶת הַפְּעוּלָה הוֹסַפְתָּ קֶשֶׁת מִשׁוּקְלָלָת

◆ נִכְתֵּב שִׁגְרָה לְהוֹסֵפֶת קֶשֶׁת מִשׁוּקְלָלָת ( עִם מִשְׁקַל  $w$  )  
מִקוּדוּד  $x$  לְקוּדוּד  $y$  בְּגֵרֶף  $g$  .

void joinw(ptr\_vertice x ,ptr\_vertice y,arc\_info ◆  
w,  
ptr\_vertice g). ◆

◆ /\* אִם קִיַּיֵּמַת קֶשֶׁת כִּזֹּו מִשְׁקָלָה יִהְיֶה מִשְׁקַל חֲדָשׁ  $w$  \*/

◆ /\* נִצָּא מִתּוֹךְ הַנְּחָה שֶׁמִּחוּץ לְפּוֹנִקְצִיָּה מוֹגְדָרִים עֲרֻכִים

בּוֹלִיאָנִים כְּדִלְקָמֶן : ◆



◆ #define TRUE 1

◆ #define FALSE 0

◆ נשתמש במשתנה לוגי (בוליאני) אשר יקבל את הערך הלוגי

"אמת" (TRUE) כאשר קיימת הקשת  $< x, y >$ .

◆ אחרת, המשתנה יקבל את הערך "שקר" (FALSE)

◆ וכזכור מאחר ומטרתנו להוסיף קשת משוקללת חדשה,

◆ נטייל על פני רשימת הקשתות בעזרת זוג מצביעים  $p$  ו  $q$ .





```

{
  int found = FALSE ;
  /* < x , y > קשת לא מצאנו כמובן לא חיפשנו עוד כל */
  ptr_arc p ,q;
  for (q=NULL, p=x→arcpoint ;
        (p!=NULL)&&(!found);)
    /* אם זיהינו קשת < x , y > עדכן את המשקל וצריך */
    /* לסיים את החיפוש */
    if (p→vertexpoint == y)

```



```

{
    p → arcinfo = w ;    found = TRUE ;
}

/*    < x , y >    אם לא זהינו עד כה את הקשת */
else {
    /*    קדם את הזוג p ו q    */
    q = p ;    p = p → arclink;
}
```





/\* בתום החיפוש נשאל- האם לא מצאנו הקשת  $\langle x, y \rangle$  \*/  
/\*?

/\* אם התשובה חיובית – נוסיף אותה \*/

- ◆ if (!found )
- ◆ {p= malloc(sizeof(struct arc));
- ◆ p → arcinfo = w ;
- ◆ p → arclink = NULL;
- ◆ p → verticepoint = y ;



♦ כעת נבדוק האם הוספת הקשת מתבצעת בראש רשימת /\*

♦ \*/ הקשתות היוצאות מקודקוד x ?

```
♦ if ( q == NULL )  
♦     x → arcpoint = p ;  
♦ else  
♦     q → arclink = p ;  
♦ }  
♦ }
```





## ◆ עֵתָה נִמְמַשׁ אֶת הַפְּעוּלָה הַסֵּרֶת הַקֶּשֶׁת

◆ `void remv(ptr_vertice x , ptr_vertice y ,  
ptr_vertice g)`

◆ המטרה לרדת לרשימת הקשתות היוצאות מקודקוד x בגרף .

◆ אם ברשימה זו קיימת קשת היוצאת מקודקוד x ל- y נבטל אותה .

◆ מאחר שהמטרה להסיר קשת-כידוע "לפי כלל הברזל" נטייל על פני רשימת הקשתות בעזרת זוג מצביעים הנעים במקביל על פני הרשימה .



```
◆ {  
◆ ptr_arc q ,p;  
◆ for(q=NULL,p=x→arcpoint;p&&  
◆ (p→verticepoint!=y);)  
◆ { q = p ;  
◆ p = p→arclink ;  
◆ }  
◆
```





◆ if ( $p \neq \text{NULL}$ )

◆ אם עצרנו את לולואת ה for בגלל שקיימת קשת  $\langle x, y \rangle$  /\*

◆ , כלומר  $p \rightarrow \text{vertexpoint}$  שווה ל- $y$ , אז נבטל את הקשת  $\langle x, y \rangle$ .

◆ /\* קודם נבדוק האם פעולת הביטול תתבצע בראש הרשימה ? /\*



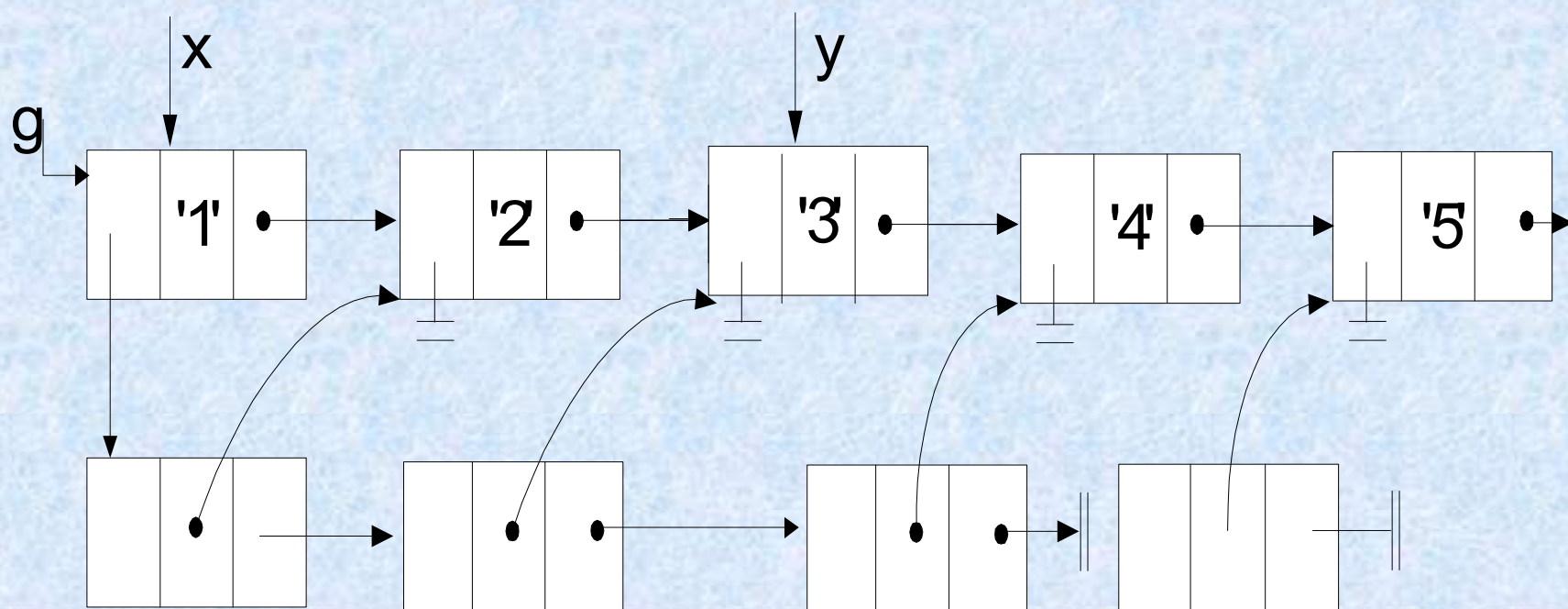
```

◆ {
◆   if (q==NULL)
◆     x → arcpoint = p → arclink;
◆   else
◆     q → arclink = p → arclink;
◆   free(p)
◆ }
◆ }
```





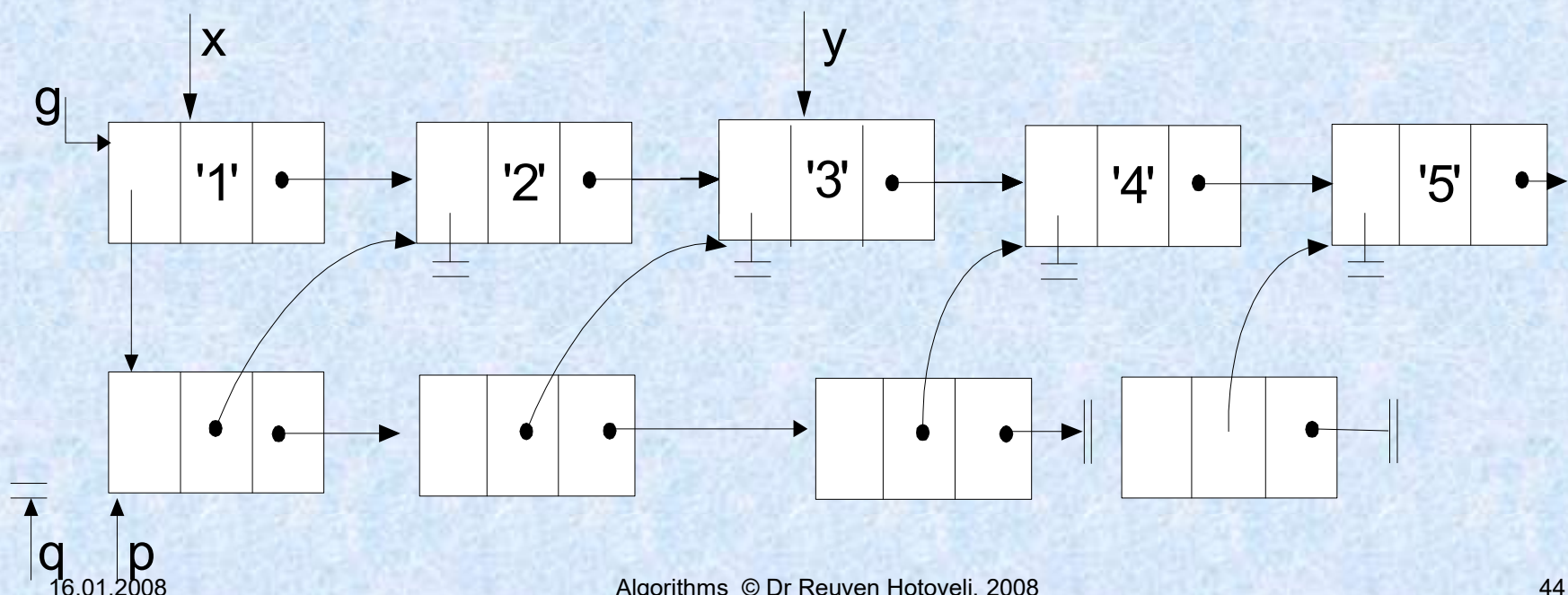
◆ עתה נראה את שלבי האלגוריתם בעזרת הגרף הבא:





♦ מטרינו לבטל קשת  $< 1, 3 >$  בגרף הנתון.

♦ א. תחילה  $p$  ו  $q$  יצביעו כנדרש לרשימת הקשתות היוצאות מקודקוד 1 כמתואר מטה :

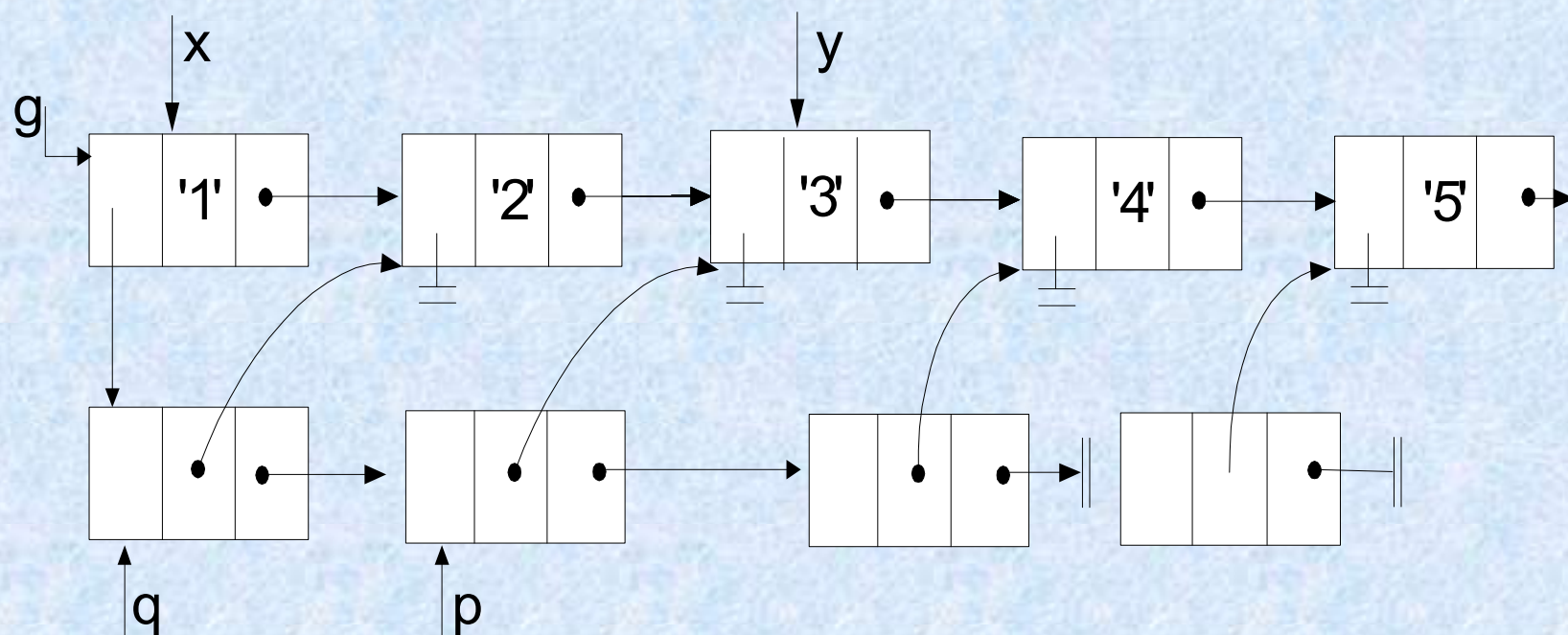






ב.תהליך החיפוש : האם קיימת קשת  $< 1, 3 >$  בגרף ?

בתום החיפוש תמונת המצב מתוארת מטה :





ג. החיפוש נעצר, מאחר והקשת  $< 1, 3 >$  נמצאת בגרף.

לכן נסיר אותה כנדרש.

קודם נבדוק האם הקשת  $< 1, 3 >$  נמצאת בראש רשימת

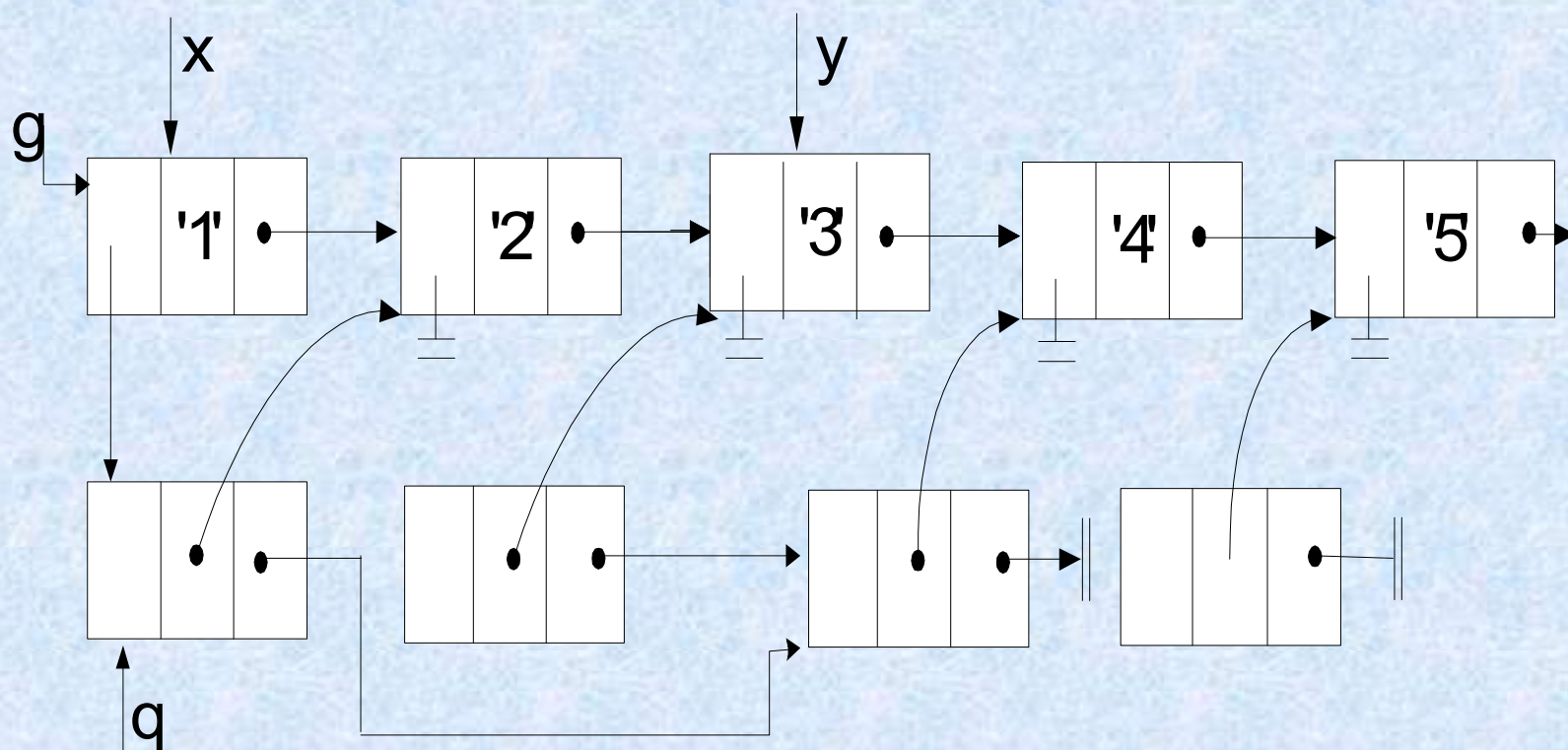
הקשתות היוצאות מקודקוד 1 ? ( כלומר האם המצביע q

שווה ל-NULL ? )

במקרה שלנו התשובה שלילית. לכן נבצע סדרת צעדים

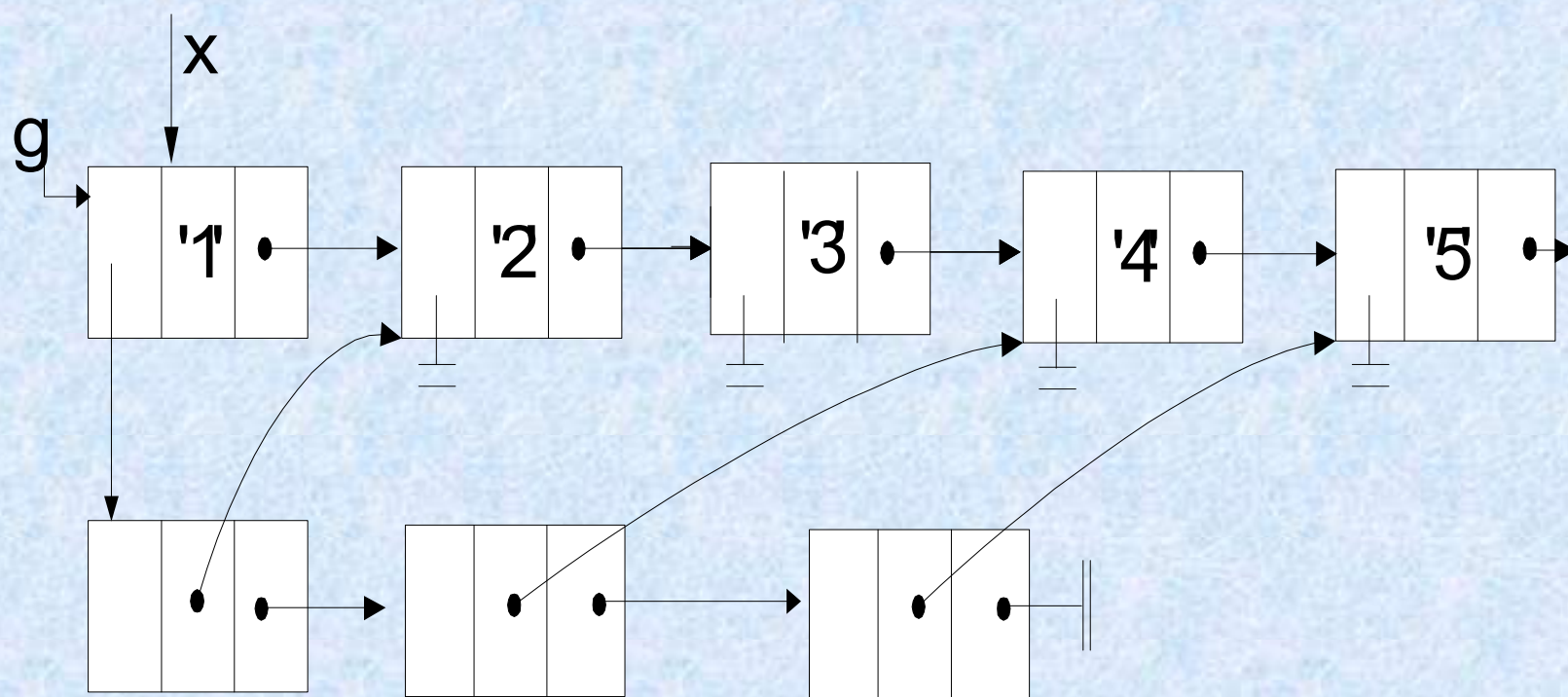
המתוארים באלגוריתם הנתון ונקבל:







בסוף נקבל את הגרף הבא :

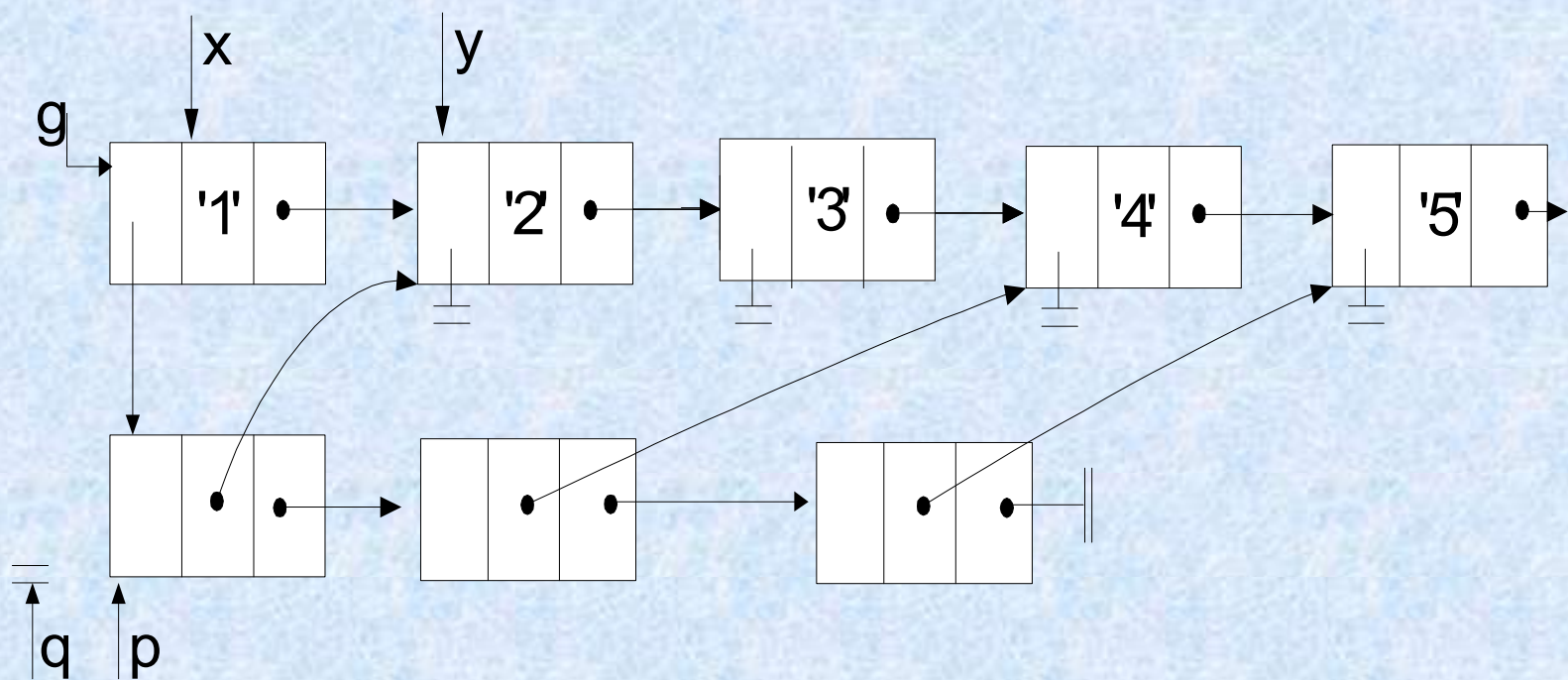






◆ עתה נראה את שלבי האלגוריתם כאשר נרצה להסיר קשת מקודקוד 1 לקודקוד 2 בגרף שקיבלנו לאחר הסרת הקשת  $\langle 1, 3 \rangle$ .

◆ תחילה  $p, q$  יצביעו כנדרש לרשימת הקשתות היוצאות מקודקוד 1 כמתואר מטה:







ב. בתום תהליך החיפוש אחר הקשת  $< 1, 2 >$  תמונת

המעקב תהיה אותה תמונה כמו ב-א'.

ג. בתום החיפוש אחר הקשת  $< 1, 2 >$  מאחר ו  $q = \text{NULL}$

(כלומר נצטרך להסיר קשת שנמצאת בראש רשימת

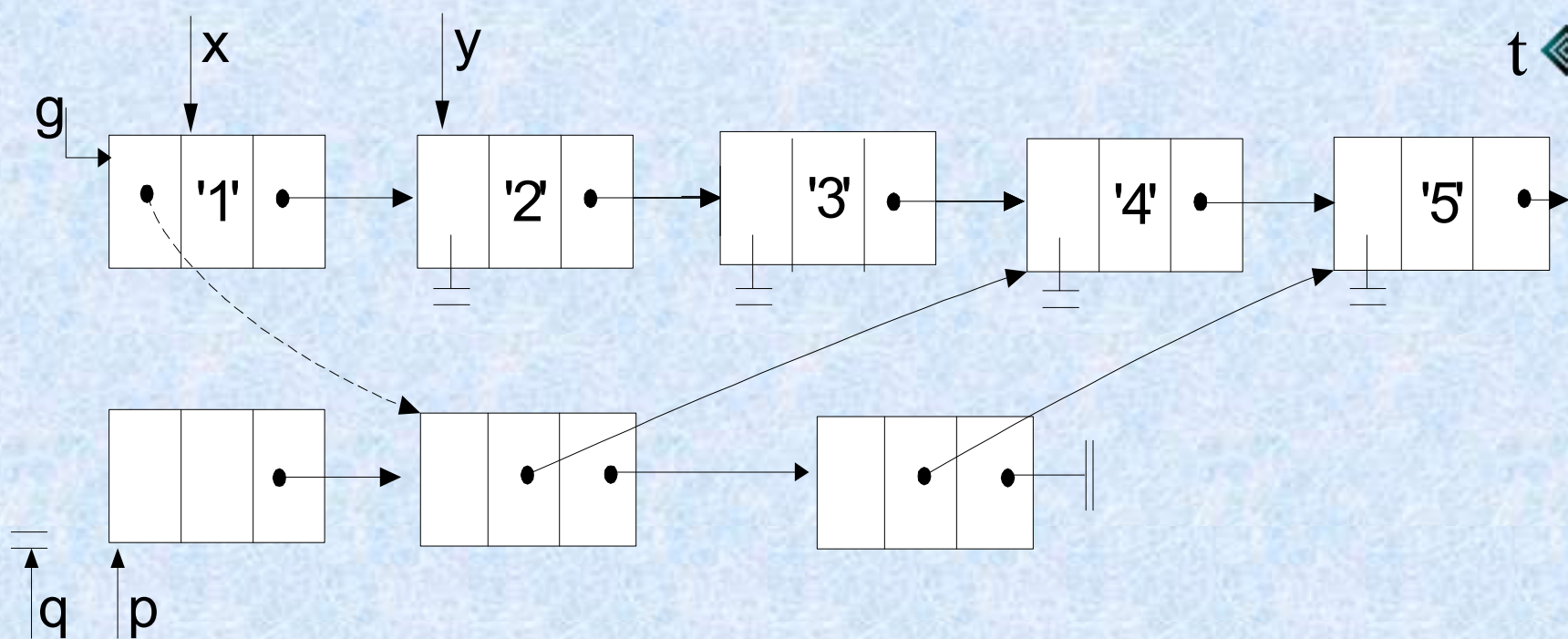
הקשתות היוצאות מקודקוד 1) נבצע את המשפט :

◆  $x \longrightarrow \text{arcpoint} = p \longrightarrow \text{arclink}$

◆ ולבסוף תמונת הגרף הינה:



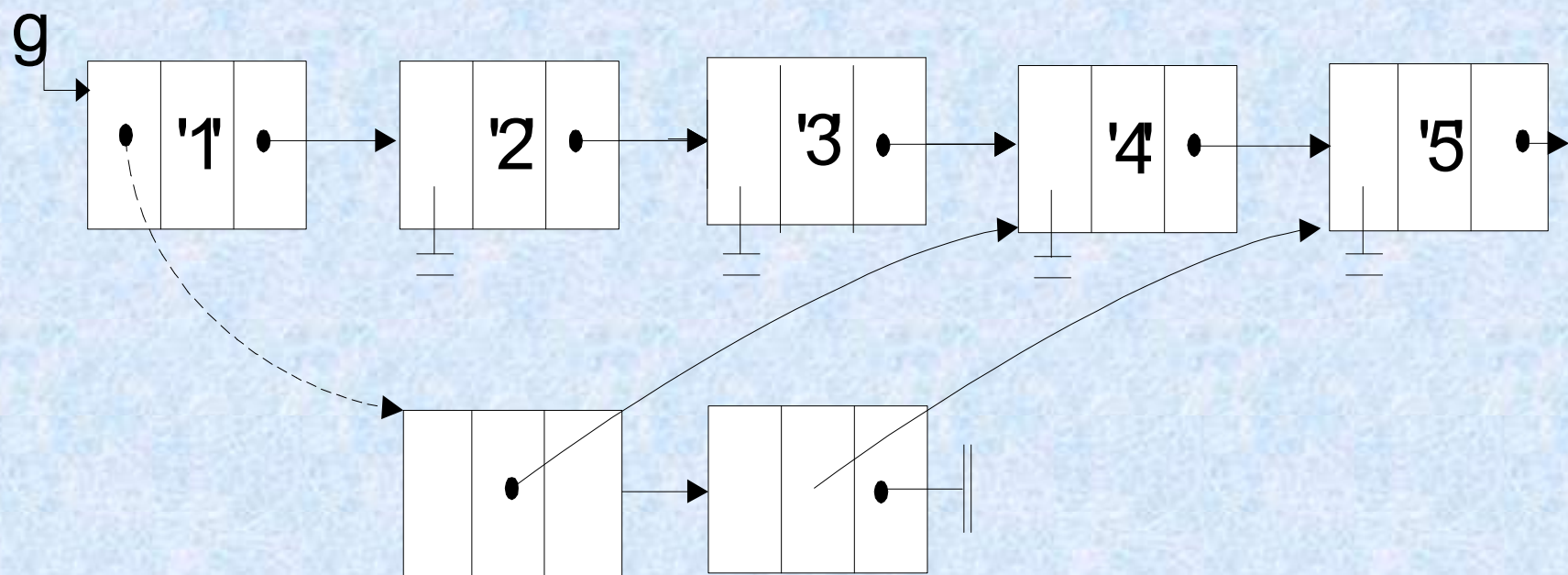
t 







❖ ולאחר שחרור הצומת המוצבע על ידי  $p$  נקבל :





הערות: ♦

♦ קל לראות שאם הקשת  $< x, y >$  לא קיימת בגרף  
, לא תתבצע פעולת הסרת הקשת (בדוק לבד !).  
♦ פעולת הסרת קשת משוקללת נשאר לביצוע  
כתרגול לקורא.





◆ עֵתָה נִמְמַשׁ אֶת הַפְּעוּלָה – בְּדִיקַת שְׁכָנוֹת

◆ נכתוב פונקציה אשר מחזירה ערך "אמת" ( TRUE ) אם קודקוד y שכן (סמוך) לקודקוד x , אחרת מחזירה ערך "שקר" ( FALSE ).

◆ `int adjacent (ptr_vertice x , ptr_vertice y ,  
ptr_vertice g)`

◆ `{`

◆ `ptr_arc p ;`

◆ `/* p יצביע לראש רשימת הקשתות היוצאות מקודקוד x */`



```
◆ for(p = x→arcpoint ;p&&(p→verticepoint! = y) ;  
      p = p→arclink) ;
```

```
/* ? <x,y> האם קיימת קשת */ ◆
```

```
◆ if (p! = NULL)    return (TRUE) ;
```

```
◆ else              return (FALSE) ;
```

```
◆ }
```





## עֵתָה נִמְמֵשׁ אֶת הַפְּעוּלָה – בְּדִיקַת גֵּרָף רִיק?

◆ Graph\_is\_empty (ptr\_vertice g)

◆ {

◆ if (g == NULL)

◆     return (TRUE) ;

◆ else

◆     return (FALSE) ;

◆ }



עֵתָה נִמְמַשׁ פְּעוּלָה – יִצְרֵת גֵּרָף רִיק – g ♦

♦ void empty\_graph (ptr\_vrtice \*g)

♦ {

♦ \*g = NULL ;

♦ }






הוספת קודקוד שמספרו  $x$  בגרף ומחזירה מצביע לצומת זה.

```
◆ ptr_vertice addnode (ptr_vertice *g , vert_info x)
◆ {
    מאחר ולא חשוב הסדר בין הקודקודים של הגרף, יעיל
    יותר להוסיף צומת חדש (עבור  $x$ ) לראש רשימת צמתי הגרף
    ◆ ptr_vertice p ;
    ◆ p = malloc (sizeof(struct vertice));
    ◆ p → info = x ;          p → arcpoint = NULL ;
    ◆ p → verticelink = *g ;   *g = p ;
    ◆ return (p) ;
    ◆ }
```



## בדיקת קיום קודקוד בגרף

 עתה נכתוב פונקציה אשר תחזיר מצביע לצומת המייצג קודקוד בגרף שמספרו  $x$ , אם קיים קודקוד כזה בגרף  $g$ , אם לא הפונקציה תחזיר את הערך `NULL`.





```
◆ ptr_vertice findelement (ptr_vertice g , vert_info x)
◆ { ptr_vertice p ;
◆   for (p = g ; p && (p->info != x); p = p->
verticelink);
◆   if (p != NULL) →
◆       /* (p->info == x) כלומר */
◆       return (p) ;
◆   else return (NULL) ;
◆ }
```



◆ עתה נכתוב פעולה-

◆ הסרת צומת שמייצג קודקוד בגרף שמספרו x

```
◆ Void remvnode (ptr_vertice *g , vert_info x)
◆ {
◆ ptr_vertice  p ,q ;
◆ for(q=NULL,p=*g; p&&(p→info!=x);
      q=p,p=p→verticelink);
```







◆ if ( $p \neq \text{NULL}$ )

◆ /\* כלומר  $(p \rightarrow \text{info} == x)$  \*/

◆ /\* נבדוק: האם הצומת שמסירים אותו נמצא בראש הרשימה ? \*/

◆ if ( $q == \text{NULL}$ )

◆ { free (\*g); \*g = p ;

◆ }

◆ else

◆ {  $q \rightarrow \text{verticelink} = p \rightarrow \text{verticelink}$  ;

◆ free (p) ;

◆ }

◆ }



◆ נדמה שהאלגוריתם האחרון להסרת צומת עובד כנדרש.

◆ צריך לשים לב כאשר מסירים צומת כלשהו  $x$  מהגרף חובה עלינו להסיר כל הקשתות היוצאות מצומת  $x$  וגם חובה להסיר את כל הקשתות הנכנסות בו.

◆ במבנה נתונים כפי שהצגנו לא ניתן להסיר צומת מגרף בצורה נכונה ויעילה כיוון שאי אפשר לדעת לפי המבנה נתונים אילו קשתות נכנסות לצומת זו.





- ◆ פתרון אפשרי לבעיה זו
- ◆ לכל צומת המייצג קודקוד של גרף נחזיק ארבעה שדות:
- ◆ info – שדה מידע.
- ◆ verticelink – שדה מצביע לצומת המייצג קודקוד בגרף,
- ◆ אם יש קודקוד כזה בגרף.
- ◆ arcpoint\_out – שדה מצביע לרשימת צמתים המייצגת את
- ◆ הקשתות היוצאות מקודקוד מסוים בגרף.
- ◆ arcpoint\_in – שדה מצביע לרשימת צמתים המייצגת את
- ◆ הקשתות הנכנסות לקודקוד מסוים בגרף.

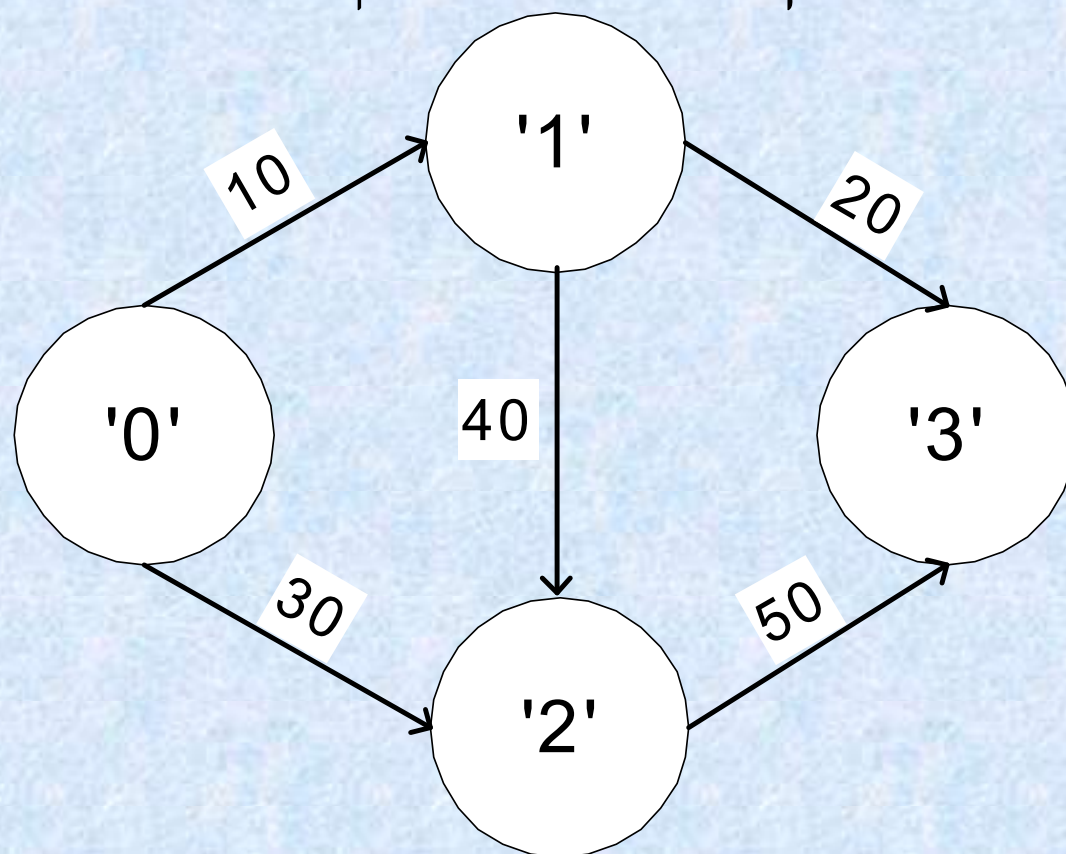


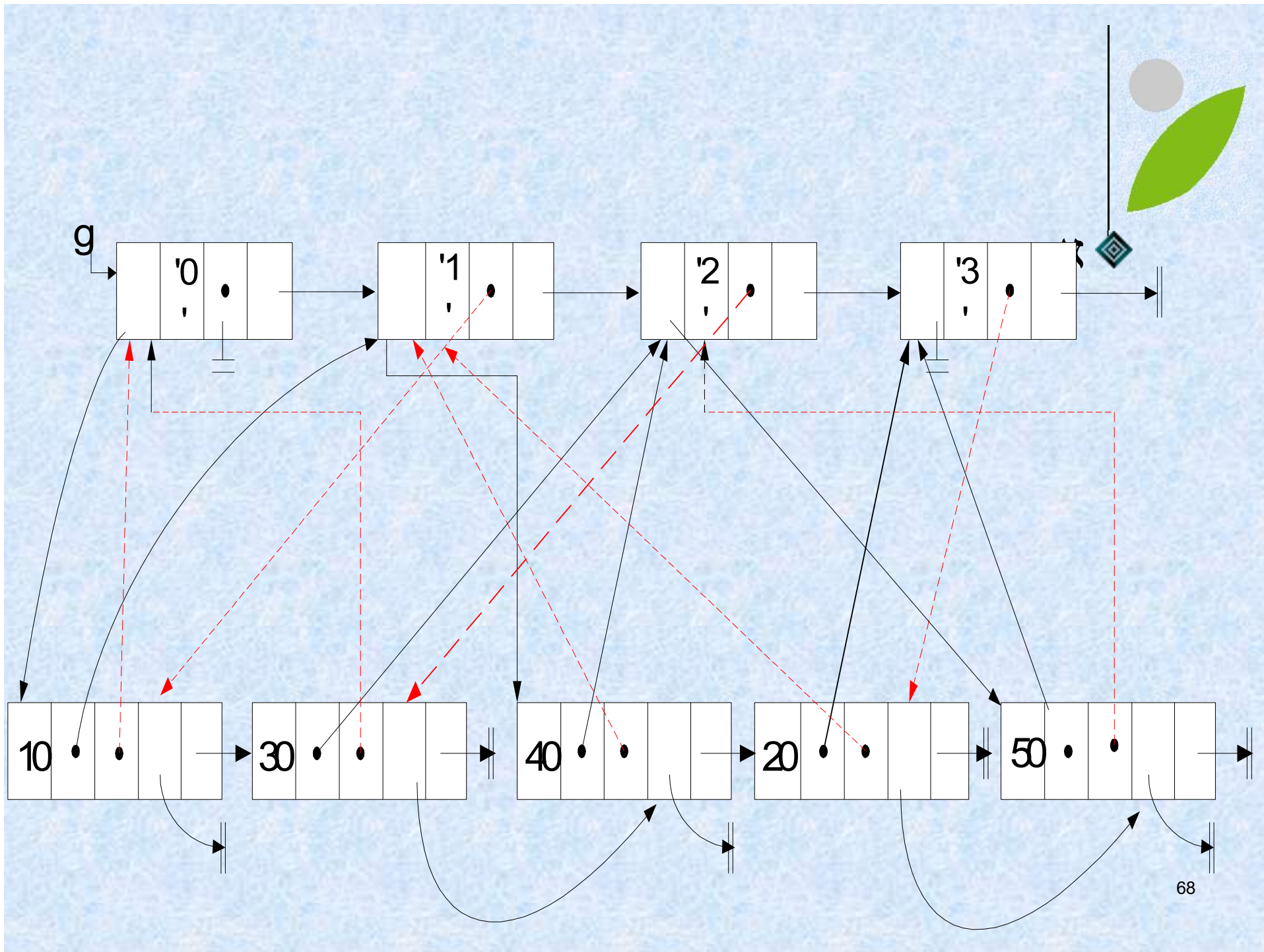
- ◆ לכל צומת המייצג קשת כלשהי  $\langle x, y \rangle$  בגרף 5 שדות.
- ◆ Arcinfo – שדה המשקלות המיוחסות לקשת  $\langle x, y \rangle$ .
- ◆ arclinkout – מצביע לקשת הבאה היוצאת מקודקוד x בגרף, אם בכלל יש קשת כזו.
- ◆ vertexpoint\_in – מצביע לצומת המייצג קודקוד בגרף בו נכנסת הקשת  $\langle x, y \rangle$ .
- ◆ arclink\_in – מצביע לקשת הבאה הנכנסת באותו צומת.
- ◆ vertexpoint\_out – מצביע לצומת המייצג קודקוד בגרף ממנו הקשת יוצאת (כלומר קודקוד x).





דוגמא: עבור הגרף הבא להלן המבנה הרב מקושר:









- ◆ תוך שימוש בייצוג החדש של גרף בייצוג רב מקושר ניתן לשכתב את הפונקציות/שגרות שכתבנו קודם.
- ◆ נשאיר את השכתוב כתרגיל לקורא.



❖ השיטה החמישית לייצוג גרף משתמשת במטריצת קשתות.

❖ בשיטה זו מספר הקודקודים בגרף ידוע מראש-נניח  $n$  וכל צומת בגרף מיוצג על ידי מספר שלם בין  $0$  ל-  $n-1$ .

❖ נניח שאין כל מידע המיוחס לקודקודים ולקשתות, כלומר נרצה לדעת רק על קיום קשתות הגרף ולא במידע המיוחס להן.



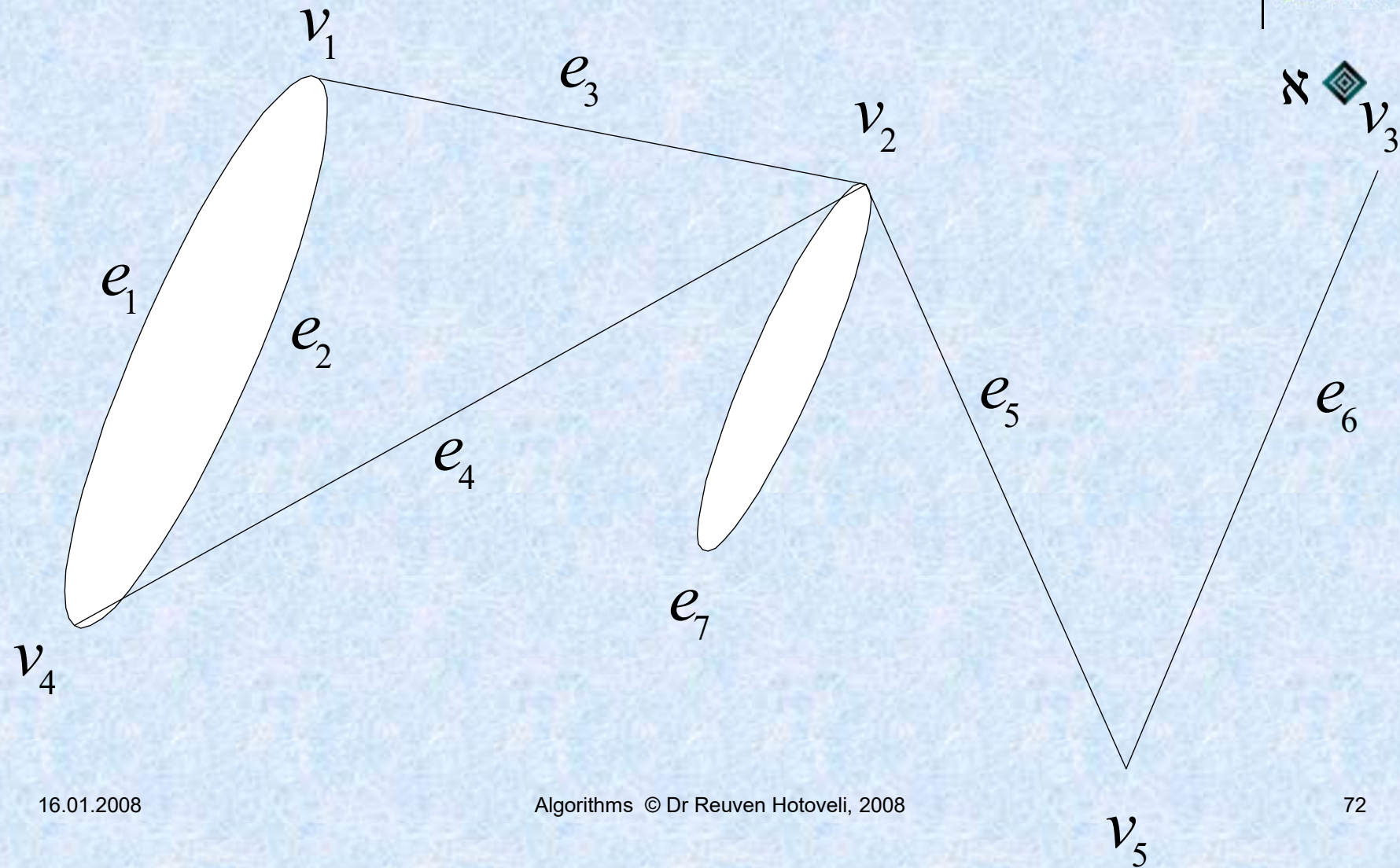


◆ להצגת הגרף נחזיק מערך דו-מימדי  $n \times m$  כאשר

$$|E| = m \quad \text{ו-} \quad |V| = n.$$

מערך דו-מימדי מייצג את כל הקשתות הנוגעות לכל צומת.

נבהיר את השיטה בעזרת הגרף הבא:







◆ נייצג אותו בעזרת המטריצה הבאה:

$$\begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{pmatrix} e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$



- ◆ השורות במטריצה הן קודקודים מהן יוצאות הקשתות.
- ◆ העמודות הן הקשתות הנוגעות לקודקוד בגרף.
- ◆ לדוגמא קודקוד  $v_1$  נוגע לקשתות  $e_1, e_2$  ו-  $e_3$ .
- ◆ לכן בשורת  $v_1$  במטריצה מופיעים "אחדים"
- ◆ (TRUES) לעמודות של הקשתות  $e_1, e_2$  ו-  $e_3$ .





◆ לאור האמור לעיל נוכל להגדיר את הגרף, בעל 100 קודקודים, כדלהלן:

◆ # define n 100

◆ # define m 10000

◆ int g [n] [m]

◆ כאמור כל צומת בגרף מיוצג על ידי מספר שלם בין 0 ל- 99 וכל קשת בגרף מיוצגת על ידי מספר שלם בין 0 ל- 9999.



אם  $g[i, j] = 1$ , אזי ניתן לומר שהקשת שמספרה  $j$  נוגעת לקודקוד שמספרו  $i$ , אחרת הקשת  $j$  אינה נוגעת לקודקוד שמספרו  $i$ .

בהמשך למבנה נתונים שהגדרנו כעת כתוב את הפונקציות/שגרות למימוש הפעולות הבסיסיות שהזכרנו במפרט. (כתוב את המימוש בסביבת העבודה – שפת C).





יש לציין שאם בגרף  $n$  קודקודים אזי קיימות בו  $n^2$  קשתות.

למרות שאנו יודעים מראש מספר הצמתים בגרף, איננו יודעים מראש דבר על מספר הקשתות.

אם לגרף מעט מאוד קשתות אזי מטריצת הקשתות תהיה דלילה.

במטרה למנוע בזבוז מקום במקרה זה רצוי להשתמש בחלופה השלישית לייצוג גרף באמצעות מבנה מקושר.