

## דוגמא ראשונה

```
void main(){
  for (i=0; i < 5; i++)
    insert(i);
  for (i=5; i<1000; i++) {
    insert(i);
    delete(i-5);
  }
}
```

כמה זיכרון צורכת התוכנית הבאה?

```
void insert(int x){
  NODE* p = malloc(sizeof(NODE));
  p -> value = x;
  p -> next = s;
  s = p;
}
```

```
void delete(int x){
  for (p = s; p -> next -> value != x; p = p -> next);
  p -> next = p -> next -> next;
}
```



## איסוף אשפה (Garbage Collection)

נותרת חילופית: ניהול זיכרון דינמי (Dynamic memory allocation)

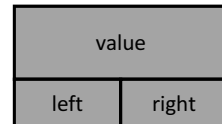
### חומר קריאה לשיעור זה

"Garbage Collection: Algorithms for automatic dynamic memory management", Richard Jones and Rafael Lins, Wiley, 1999.

## מודל פשוט לשימוש בזיכרון

• הזכרון מאורגן ממערך mem[m] שבו מוקצים צמתים (אובייקטים) מהצורה:

```
typedef struct node {
  DATA value;
  node *left *right;
} NODE;
```



(צמתים מורכבים יותר ושונים גודל אפשריים, אך אנו נתרכז במודל פשוט).

• קיימת רשימה AVAIL של צמתים לא מנוצלים. כל פעם שעושים malloc מקבלים צומת מרשימה זו וכשעושים free מחזירים צומת לרשימה AVAIL.

• הזיכרון מחולק פונקציונלית ל"ערימת אובייקטים" לשמירת אובייקטים המיוצרים באופן דינמי ול"מחסנית ריצה" המכילה את "משתני התוכנית".

## איסוף אשפה

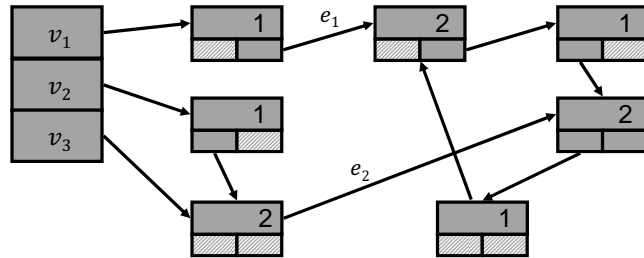
הבעיה: איך לשחרר צמתים שהמתכנת לא שחרר באמצעות פקודת free.

### חשיבות הבעיה

- בתוכניות גדולות עם הרבה מודולים והרבה אובייקטים (בניגוד לתוכניות שראינו בקורס) קשה למתכנתים לדעת מתי באמת משתחרר אובייקט.
- כאשר מיוצרת אשפה (Memory leak) קשה לאתר את התקלה, כלומר ה-debugging קשה.

## פתרון ע"י ספירה (reference counting)

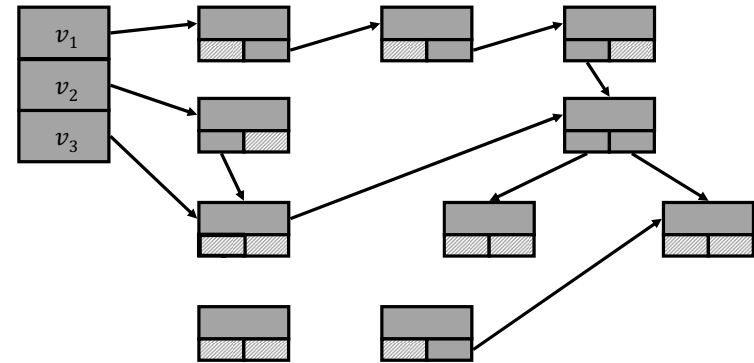
הוסף לכל צומת  $V$  שדה  $count$  השומר את מספר המצביעים אליו. עדכן את  $V.count$  בכל פעם שמשנים מצביע לצומת  $V$ . כאשר עבור צומת  $V$  מתקבל  $V.count=0$  הוסף את  $V$  לרשימת הצמתים הלא-מוצלים AVAIL ועדכן את  $count$  לצמתים אליהם  $V$  מצביע.



**בעיה:** מעגלים תמיד מכילים צמתים עם  $count$  חיובי אפילו אם אין גישה למעגל כולו. למשל אם נבטל קשת  $e_1$  וקשת  $e_2$  נקבל מעגל ללא גישה אליו, כלומר אשפה.

## מודל לשימוש בזיכרון (המשך)

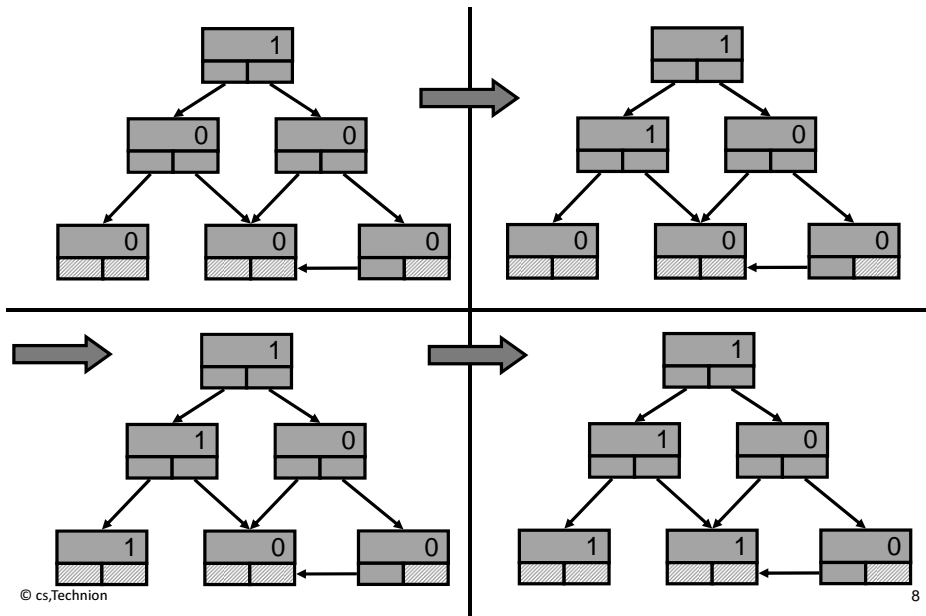
ניתן לגשת לצמתים בזכרון רק דרך "משתני התוכנית"  $v_1, \dots, v_n$  הנמצאים במחסנית הריצה.



**אשפה:** צמתים מנותקים, כאלה שאין אליהם מסלול מאף  $v_i$ . כיצד נוצרת אשפה?

נתאר שלושה פתרונות לבעיית האשפה: ספירה, סימון, ודחיסה.

## דוגמא על גרף כללי



## פתרון ע"י סימון (tracing collector)

**הרעיון:** כאשר הרשימה AVAIL מתרוקנת, עוברים על הזיכרון כולו ומסמנים את כל הצמתים אליהם יש גישה ממשתני התוכנית. עוברים שנית על הזיכרון ומכניסים לרשימה AVAIL את כל הצמתים שלא סומנו. כדי לאפשר את סימון הצמתים יש להוסיף שדה בוליאני  $label$  לכל צומת.

### האלגוריתם

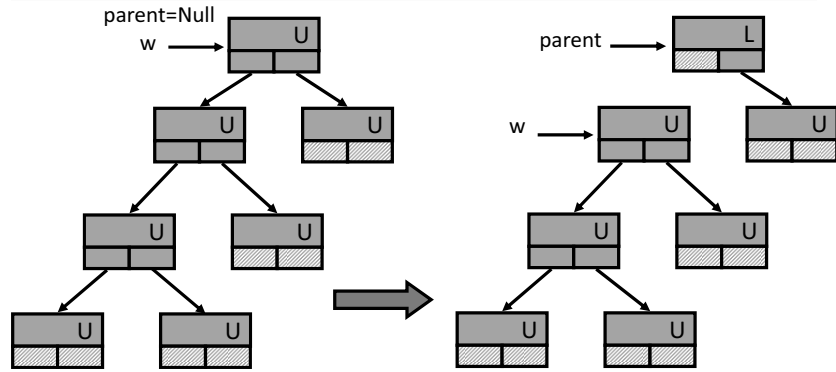
1. עבור על כל צמתי הזיכרון וקבע  $label=0$ .
2. סמן את הצמתים הקשורים ישירות למשתני התוכנית ע"י  $label = 1$ .
3. אם סימנת צומת  $w$  שלא היה מסומן, המשך רקורסיבית עם הבן השמאלי והימני כדלקמן:

```
void mark(NODE *w){
    if(w!=NULL && w->label == 0) {
        w->label = 1;
        mark(w->left);
        mark(w->right);
    }
}
```

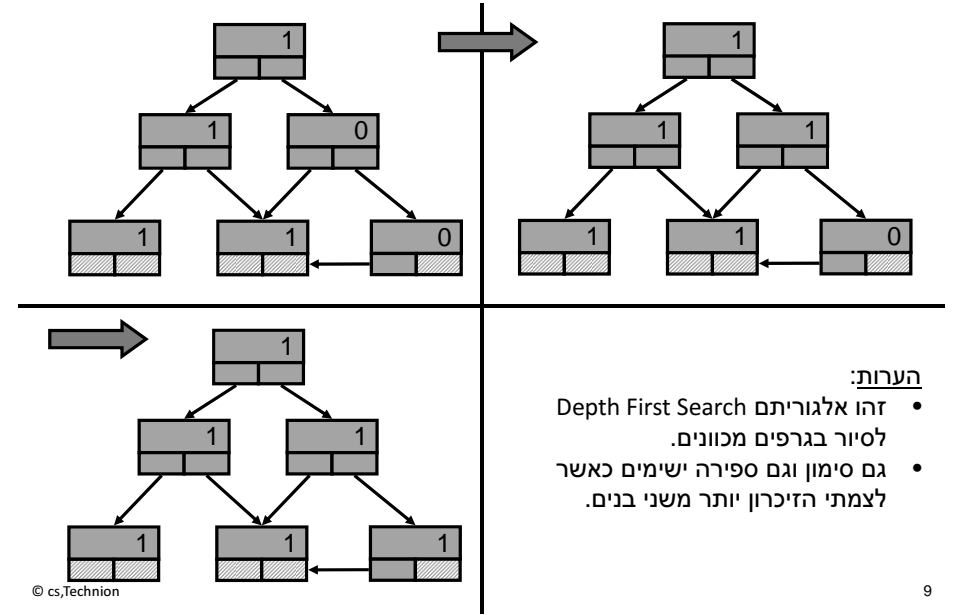
**הערה:** זהו אלגוריתם preorder עבור עצים. אבל הוא מסמן גרפים מכוונים כלשהם.

## סימון ללא רקורסיה

**הרעיון:** כאשר נבקר בצומת, נשמור את המכוון להורה שלו בשדה left או right כך שנוכל לחזור להורה בסוף הביקור. לתווית של כל צומת יהיו שלושה ערכים: {U,R,L} שיאפשרו לדעת איזה מכוון מצביע להורה – L אם המכוון השמאלי מצביע, R אם הימני מצביע, U אם אף אחד מהם אינו מצביע. נשתמש במשתנים הבאים: w עבור הצומת הנוכחי (מאותחל להיות השורש) ו-parent עבור אב הצומת הנוכחי (מאותחל ל-Null). השטה נקראת הפוך מצביעים (Pointer Reversal): פרק 4.3 בספר (Garbage Collection, Jones & Lins, 96).



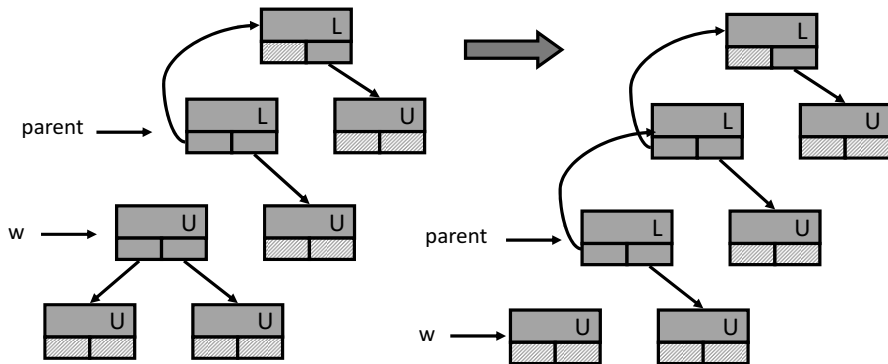
## דוגמא על גרף כללי



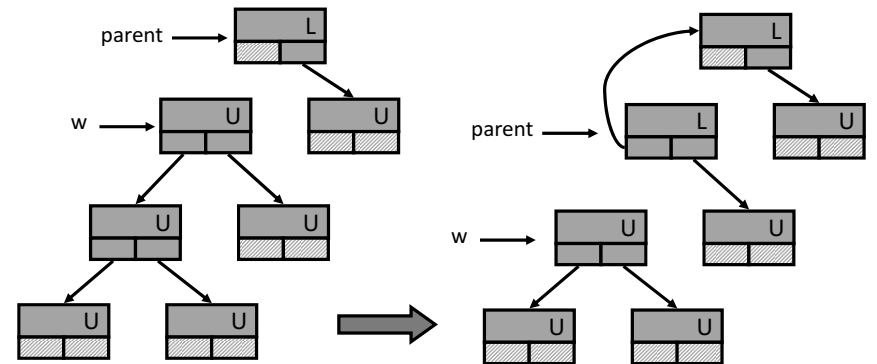
**הערות:**

- זהו אלגוריתם Depth First Search
- לסיור בגרפים מכוונים.
- גם סימון וגם ספירה ישימים כאשר לצמתי הזיכרון יותר משני בנים.

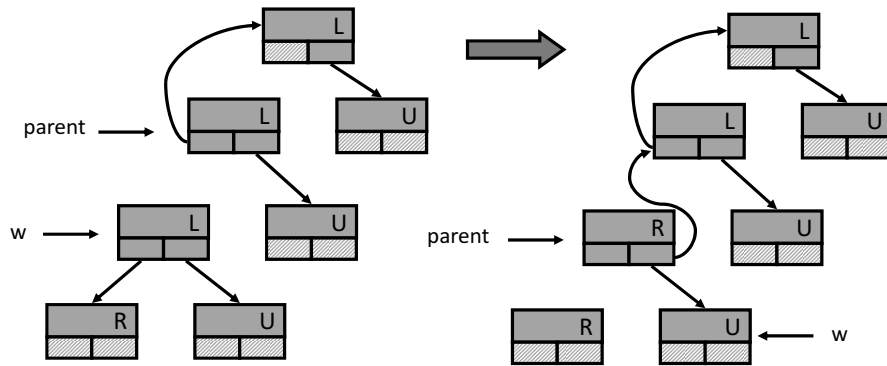
## סימון ללא רקורסיה (המשך)



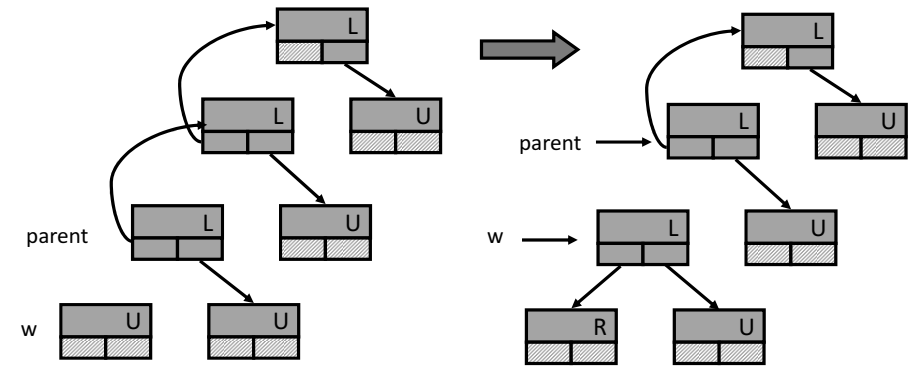
## סימון ללא רקורסיה (המשך)



## סימון ללא רקורסיה (המשך)



## סימון ללא רקורסיה (המשך)



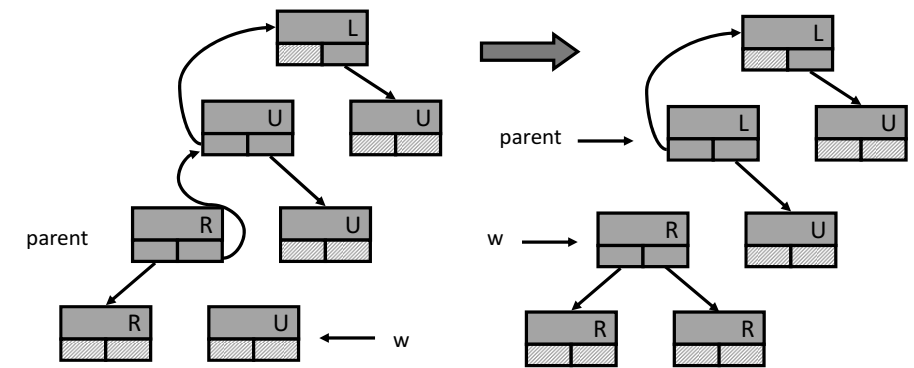
## שלד התוכנית

```

Void mark (NODE root) {
    NODE *w = root ;
    NODE *parent = NULL;
    NODE *temp
while(w != NULL){
    if ((w -> label == U) && (w -> left != NULL) && (w -> left -> label == U)){
        GO-LEFT : w -> label = L;
        temp = w -> left; /* הורדת מצביעים שמאלה תוך שימוש בתא עזר */
        w -> left = parent;
        parent = w;
        w = temp;
    }
    if (w -> label != R) && (w -> right != NULL) && (w -> right -> label == U){
        GO-RIGHT : w -> label = R;
        המשך באופן דומה...
    }
    else { /* בקרנו כבר את הילדים של w */
        GO-UP : w -> label = R;
        המשך באופן דומה...
    }
}

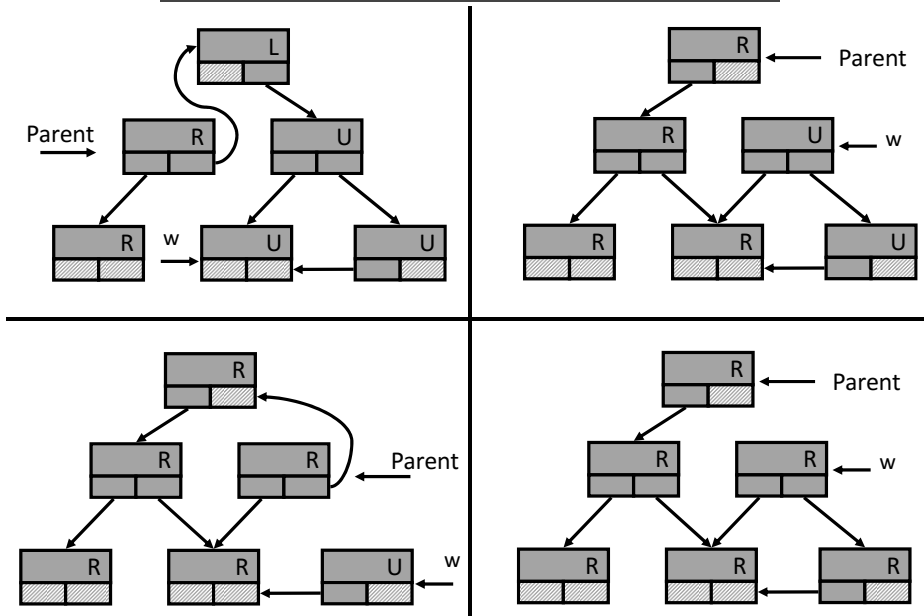
```

## סימון ללא רקורסיה (המשך)

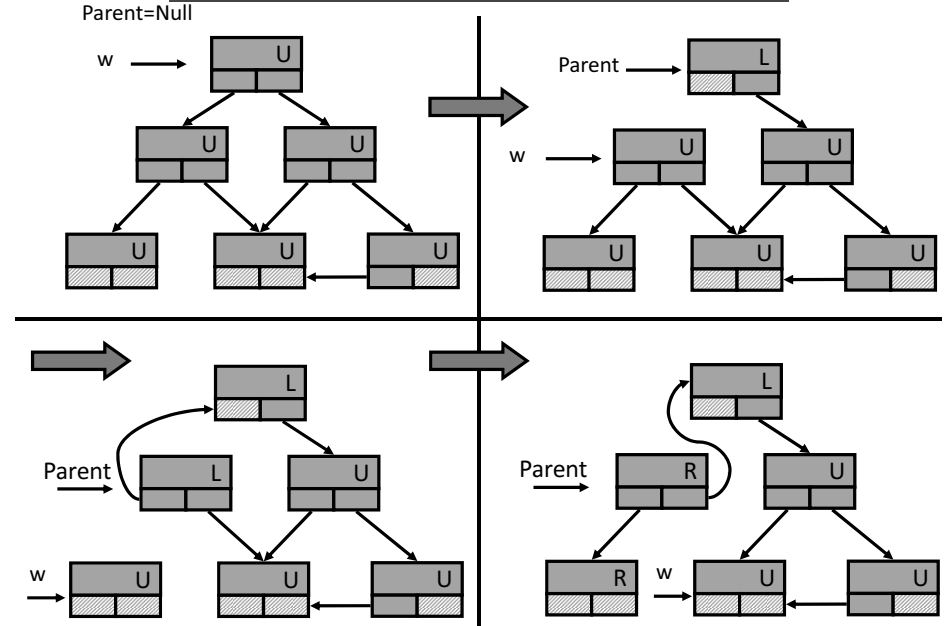


בסוף ריצת הפרוצדורה כל התוויות מסומנות .R

## האלגוריתם עובד גם על גרף כללי

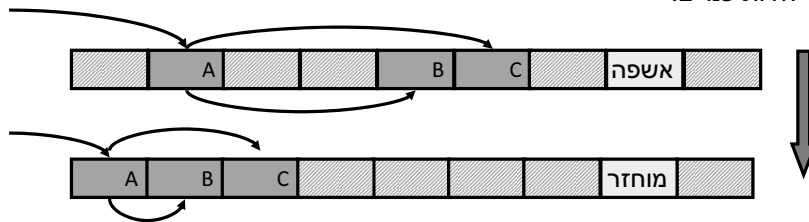


## האלגוריתם עובד גם על גרף כללי



## איסוף אשפה ע"י דחיסה (Compaction)

מטרה: ריכוז כל הצמתים הקשורים בתחילת הזיכרון – כל שאר הצמתים הופכים להיות פנויים.



נדרשת זהירות בעדכון המצביעים: העתקה של צמתים למקומם החדש אינה מספיקה. להלן פתרון ראשוני לבעיה:

- לכל צומת נוסף שדה new-addr שישימש אותנו בזמן איסוף האשפה.
- נבצע סימון.
- נחשב לכל צומת את הכתובת החדשה, ונשים אותה בשדה new-addr.
- נעבור על הצמתים הקשורים ונעדכן מצביעים.
- נעתיק את הצמתים.

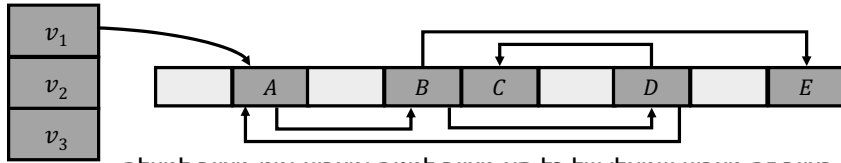
## הערות

רצוי למנוע מצב שמדי פעם נעשית פעולה ארוכה של איסוף אשפה מזיכרון שבו נערמה אשפה רבה.

### פתרון

- שילוב בן ספירה וסימון. ספירה משחררת צמתים בזמן שהם מתגלים. בד"כ אין מעגלים רבים וספירה מונעת הצטברות אשפה. ניתן להגביל את גודל השדה count לשלושה-ארבע ביטים מפני שרק לעיתים רחוקות כמות המצביעים על צומת גדולה מ-10. מצב נדיר זה מטופל ע"י הסימון.
- איסוף אשפה גם כאשר רשימת AVAIL אינה ריקה (on the fly), למשל כאשר יש הפוגה בעומס על מערכת ההפעלה.

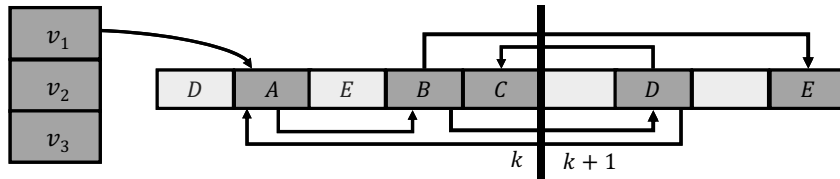
## דחיסה ללא שדה נוסף



בצירור זה מצביע שמאלי של כל תא מצויר למטה ומצביע ימני מצויר למעלה.

הרעיון: נניח שישנם  $k$  צמתים נגישים. בדוגמא  $k = 5$ .

כל הצמתים בכתובות  $1 \dots k$  יושארו במקומם. כל הצמתים הקשורים הנמצאים במקומות  $k + 1 \dots m$  יועתקו למרווחים שנמצאים ב- $k$  הכתובות הראשונות.



בזמן העתקת צומת  $V$  לכתובת החדשה, נשמור במקומו הישן (נאמר בשדה left) מצביע לכתובת החדשה. מצביע זה ישמש לעדכון המכוונים המצביעים אל  $V$ . בדוגמא נעדכן את שני המצביעים של צומת  $B$  לכתובות החדשות של  $D$  ושל  $E$ .

## פרוט התוכנית

```

1. סימון: For (i = 0; i < n; i++) mark(vi);

2. חישוב כתובת חדשה: p=0;
for (j=0; j < m; j++){
    if(mem[j].label != U)
        mem[j].new_addr = p++;
}

3. עדכון:
מצביעים:
for (j=0; j < m; j++){
    if(mem[j].label != U){
        if (mem[j].left != NULL)
            mem[j].left = mem[j].left → new_addr;
        if (mem[j].right != NULL)
            mem[j].right = mem[j].right → new_addr;
    }
}

4. העתקה:
for (j=0; j < m; j++){
    if(mem[j].label != U){
        mem[mem[j].new_addr] = mem[j];
    } /* שימו לב שמתקיים: mem[j].new_addr < j */
}
    
```

## הערות לסיכום

- בשיטת הספירה משתמשים לדוגמא במערכות ה- unix: perl, awk וכן במערכות חומרה קטנות כמו בקרים בהם פעולת reset מנקה אשפה.
- בשפת MODULA 2+ משתמשים בשיטת ספירה עם backup בעזרת שיטת הסימון.
- בספר "Garbage Collection: Algorithms for automatic dynamic memory management", Richard Jones and Rafael Lins, Wiley, 1999. יש פרקים על איסוף אשפה בשפת C++ ובשפות רבות נוספות.
- כל האלגוריתמים בהם דנו ניתנים להרחבה לזיכרון המכיל אובייקטים מסוגים וגדלים שונים.

## האלגוריתם לדחיסה

