© Itai +Katz April 14, 1996

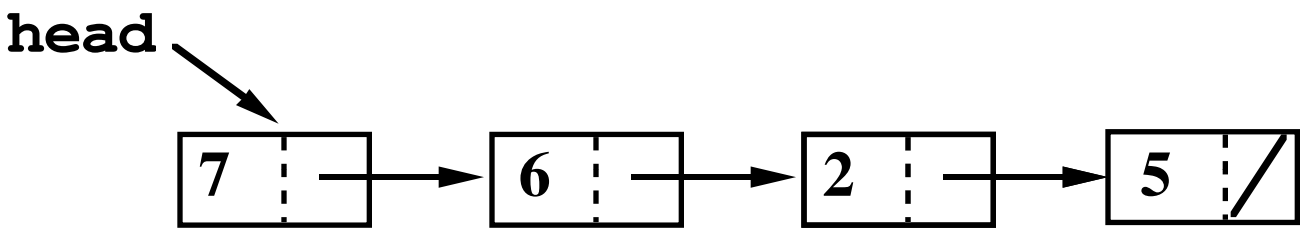
Linear Lists רשימות מקושרות

<u>רשימה_ליניארית היא מבנה נתונים בעל התכונות:</u>

- **סריקה סדרתית בזמן ליניארי.**
- **הכנסה/הוצאה בזמן קבוע (כשידוע מקום השינוי).**
- **גודל המבנה משתנה באופן דינמי במהלך התכנית (אין חסם על מספר האיברים).**
- **אין צורך להקצות מקום רצוף למבנה.**

<u>חסרונות</u>

- **אין גישה אקראית לפי מציין.**
- **אפשר לאבד מבנה שלם ...**
- **מימוש יותר מורכב.**



<u>הפעולות:</u>

איתחול--יצירת רשימה ריקה,

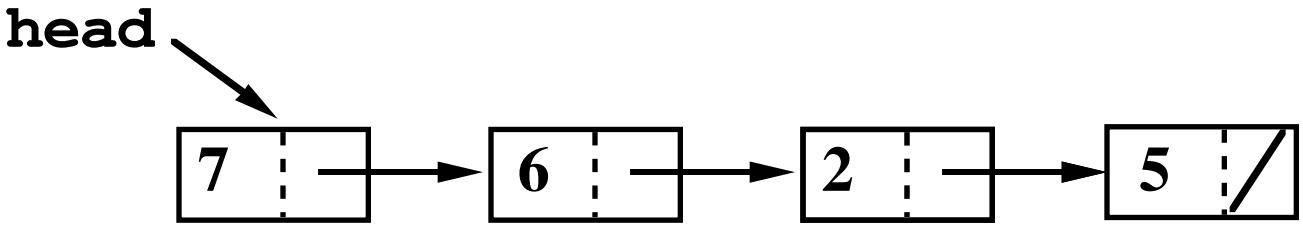
חיפוש --מציאת צומת ששדה ה-`info` שלו הוא `x`,

הכנסה --הוספת צומת,

הוצאה --סילוק צומת.

מימושים לרשימות מקושרות

1. על הדף:



2. בעזרת מערך : מציינים במקום מחוונים.

	info	next
0	6	4
1	5	8
2	7	0
3	88	3
4	2	5
5	5	-1

head = 2

(שימוש זה נפוץ בשפות שאין להן מחוונים, כגון Fortran).

3. מחוונים: במקום מציין נרשום את כתובת הצומת הבא.
בהנחה ש- int או מחוון תופסים 4 בייטים:

1000	6	1032
1008	5	2008
1016	7	1000
1024	88	1024
1032	2	1040
1040	5	0

ב-C: נשתמש בצמתים מטיפוס NODE

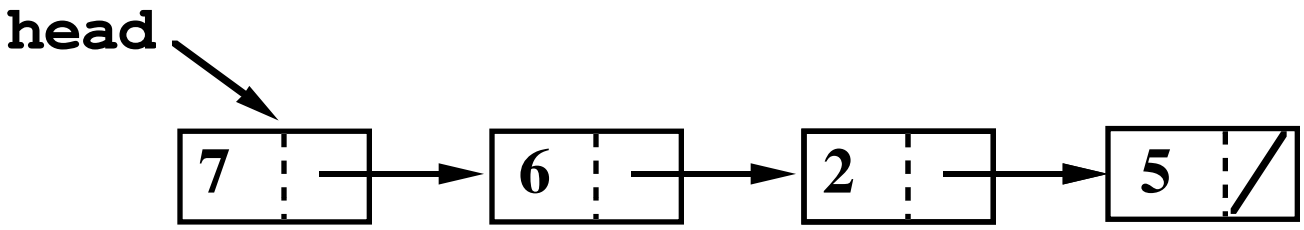
```
#define NULL 0
typedef struct node {
   int info;
   struct node *next;
} NODE;

NODE *h;
```

מימוש הפעולות ב-C בעזרת מחוונים:

איתחול--יצירת רשימה ריקה

```
head = NULL;
```

חיפוש צומת ששדה ה-info שלו הוא x :

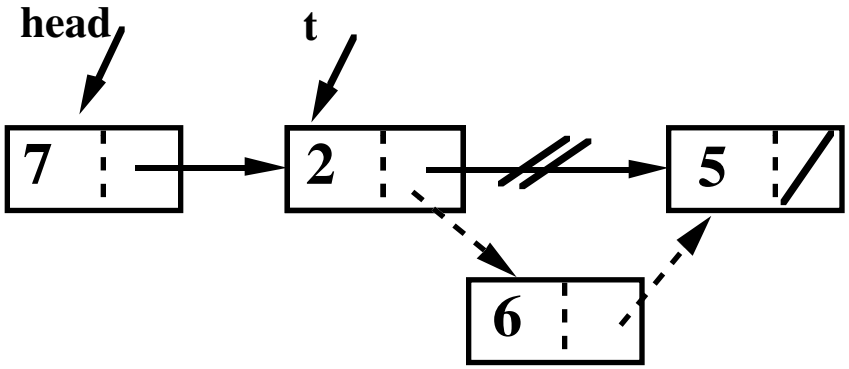



```
NODE *find(int x, NODE *head){
   NODE *t;
   for(t=head; t≠NULL && t->info≠x;
                  t = t->next)
    ;
   return t;
```

מה קורה אם x לא נמצא ברשימה?

הכנסה--

1. הוספת צומת אחרי הצומת ש-t מצביע עליו:




```
int insert(int x, NODE *t){
   NODE *p;
   if(t==NULL)
        return 0;
   p = (NODE *)malloc(sizeof(NODE));
   p->info = x;
   p->next = t->next;
   t->next = p;
   return 1;
}
```

2. הוספת צומת לראש הרשימה:

```
int insert_first(int x,NODE **p_head){
   NODE *p;
   p = (NODE *)malloc(sizeof(NODE));
   p->info = x;
   p->next = *p_head;
   *p_head = p;
   return 1;
}
```

מתי עוד יכולות הפרוצדורות האלו להכשל?

הוצאה

הוצאת הצומת שאחרי t:

```
delete(NODE *t){
   t->next = t->next->next;
}
```

מה קורה עם הצומת t->next?

הוצאת הצומת הראשון:

```
int delete_first(NODE **p_head){
   if(*p_head == NULL)
      return 0;
   *p_head = (*p_head)->next;
   return 1;
}
```

יתרונות וחסרונות של רשימות מקושרות

יתרונות

* הכנסה והוצאה בזמן $O(1)$.
* אין צורך בפתיחת/סגירת חורים.
* אין צורך להקצות איזור רצוף בזכרון.

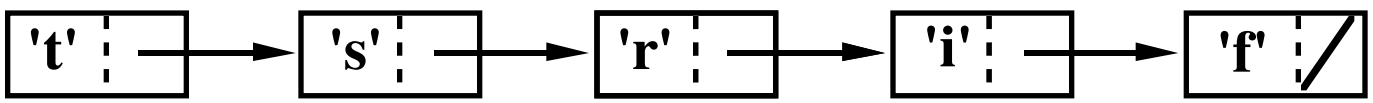
חסרונות

* חובה לעבור על הרשימה לפי הסדר,
  ואין גישה ישירה לפי מציין (אינדכס).

פעולות נוספות על רשימות

היפוך רשימה

```
NODE *reverse(NODE *t){
   NODE *rev=NULL, *temp;
   while(t≠NULL){
        temp = t->next;
        t->next = rev;
        rev = t;
        t = temp;
   }
   return rev;
}
```



temp חיוני כדי ליצור גישה להמשך הרשימה.

פרוצדורה רקורסיבית להיפוך רשימה

פרוצדורת עזר

```
NODE *last(NODE *h){
   if(h == NULL)
        return NULL;
   while(h->next ≠ NULL)
        h = h->next;
   return h;
}
NODE *rec_reverse(NODE *t){
   NODE  *temp, *l;
   if(t == NULL)
        return NULL;
   if(t->next == NULL)
        return t;
     temp = rec_reverse(t->next);
     l = last(temp);
     l->next = t;
     t->next = NULL;
     return temp;
}
```

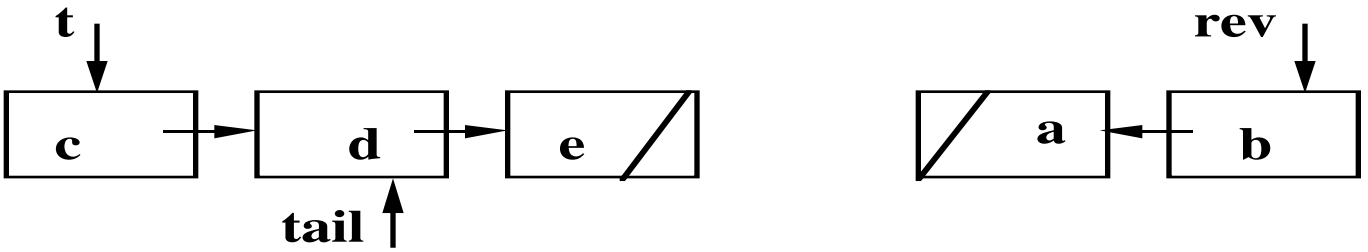
חישוב זמנים:

$$\begin{aligned} Time(n) &= Time(n-1) + c \cdot n \\ &= c \cdot n + c \cdot (n-1) + Time(n-2) \\ &= c \cdot n + c \cdot (n-1) + \ldots + c \cdot 1 + Time(0) \\ &= c\frac{n(n+1)}{2} + Time(0) \ = O(n^2). \end{aligned}$$

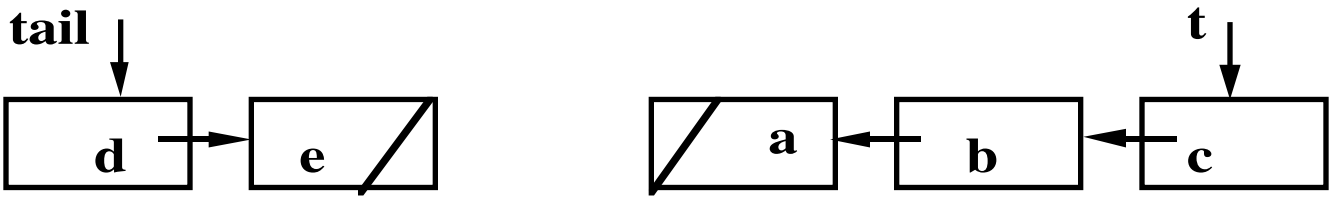
פרוצדורה רקורסיבית יעילה להיפוך רשימה
(efficient_rec_reverse)

פרוצדורת עזר:

```
NODE *rec_rev1(NODE *t, NODE *rev){
   NODE  *tail;
   if(t == NULL)
        return rev;
   tail = t->next;
   t->next = rev;
   return rec_rev1(tail, t);
}
```



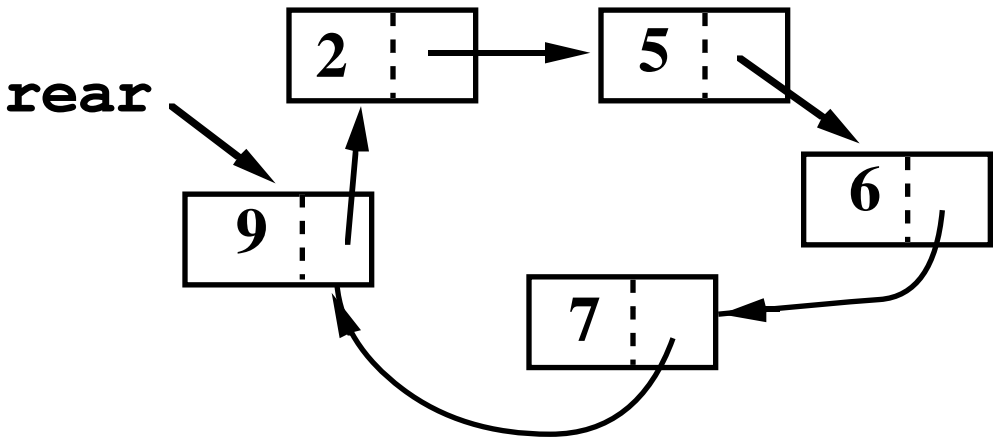
הקריאה הרקורסיבית:




```
NODE *efficient_rec_reverse(NODE *t){
   return rec_rev1(t, NULL);
}
```

וריאנטים

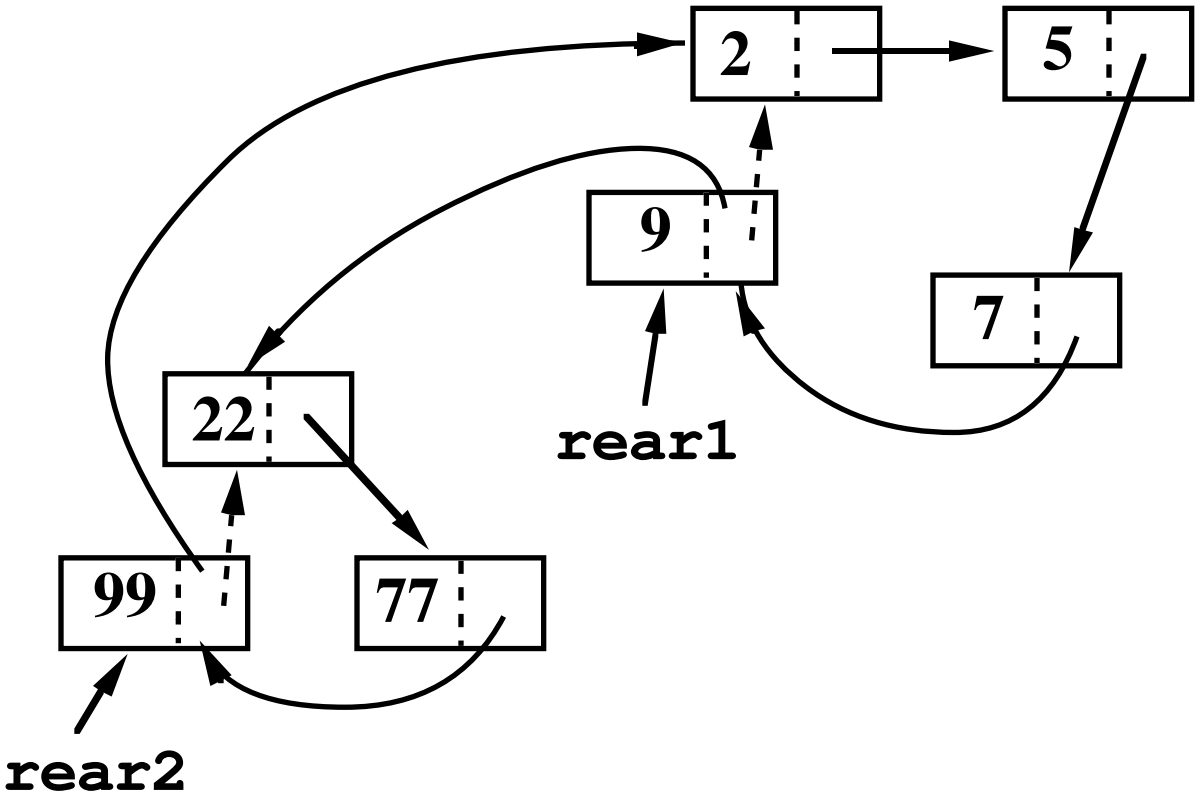
1. רשימה מעגלית



יתרונות:

אפשר להגיע מכל איבר לכל איבר

ניתן לשרשר רשימות מעגליות בזמן קבוע:

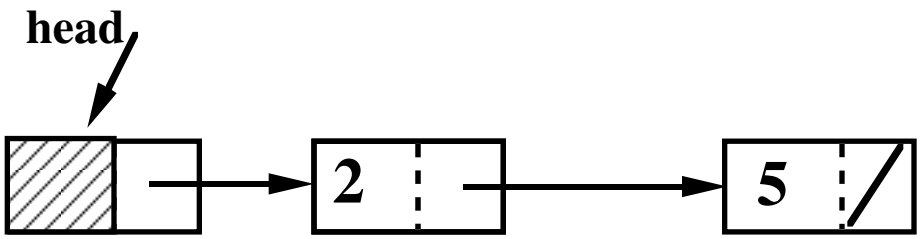


2. רשימות עם כותרת (header)

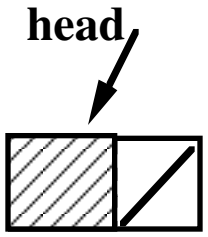
איבר "דמה" בתחילת הרשימה, החוסך טיפול מיוחד ברשימות ריקות, הוספות לפני האיבר הראשון וכו'.

בייצוג זה:

רשימה בת שני איברים

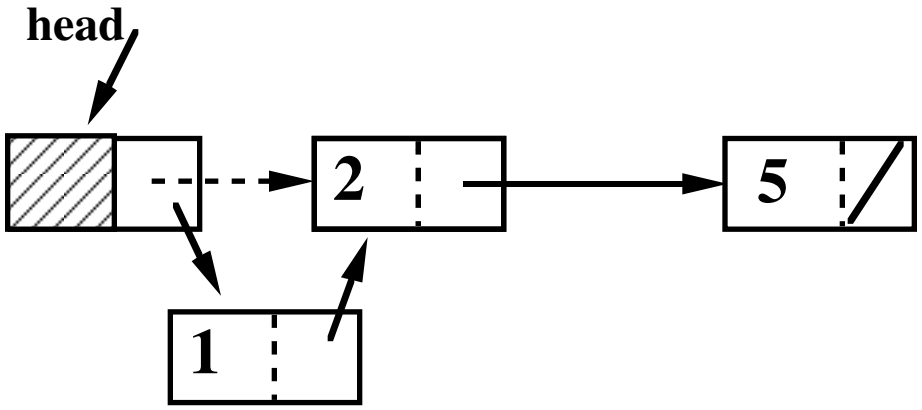


רשימה ריקה:

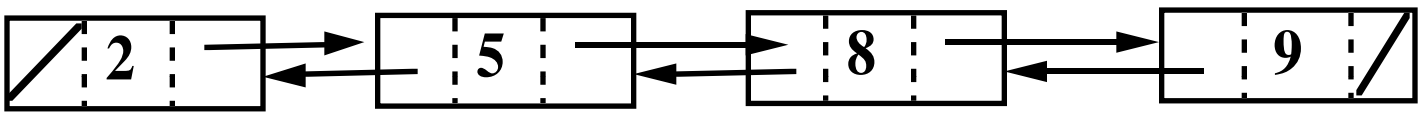


הוספת איבר ראשון:

```
insert(x, head)
```

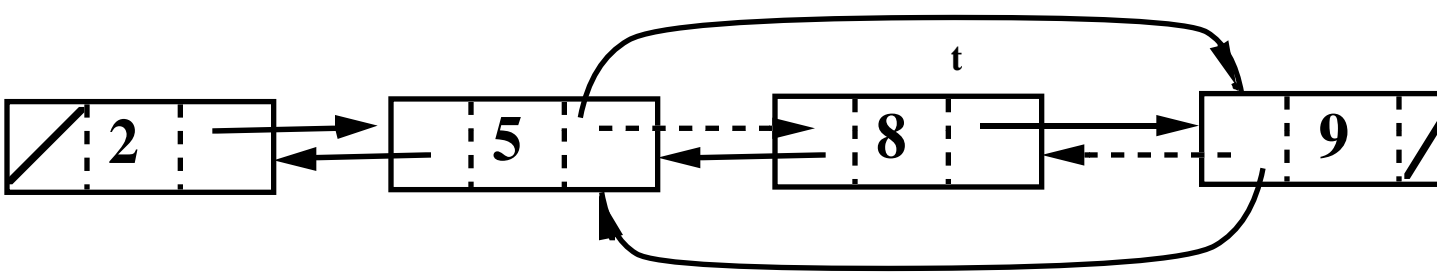


3. רשימה דו-כיוונית



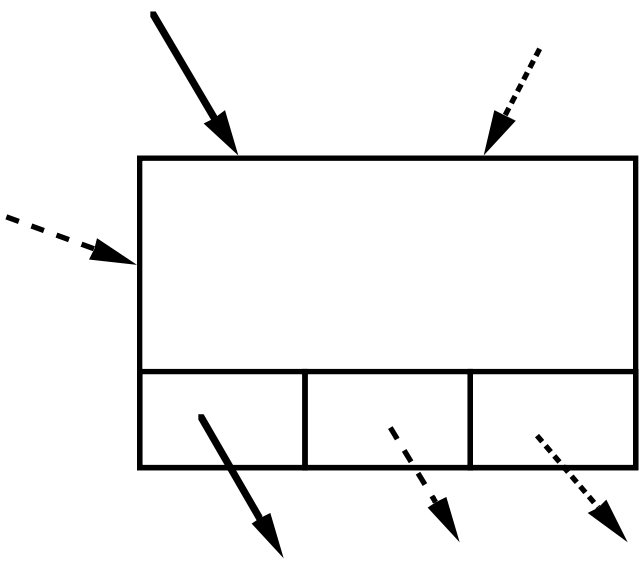
ייתרון: מאפשר להוציא איבר עצמו (ולא רק את זה שאחריו):

```
t->next->prev = t->prev;
t->prev->next = t->next;
```



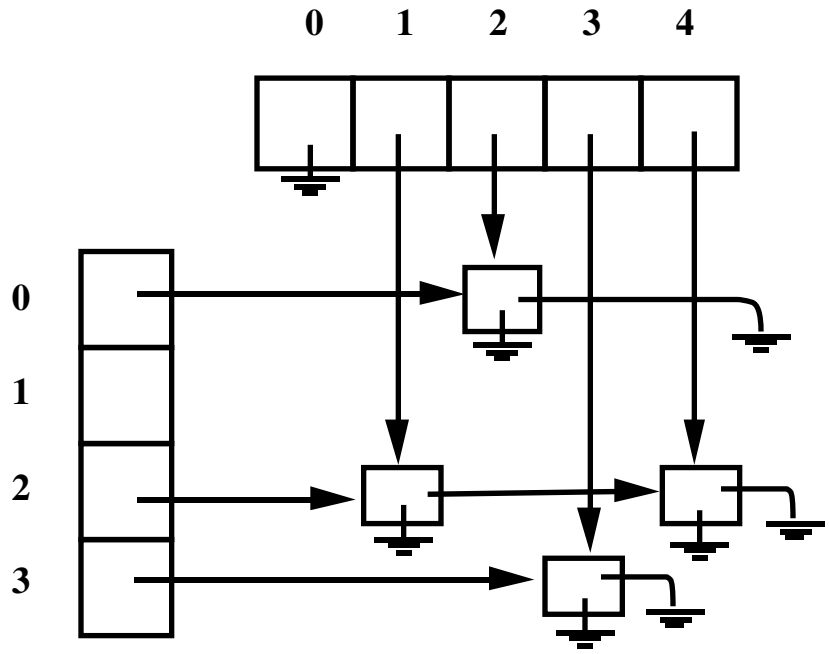
4. שיתוף רשימות

איברים משותפים למספר רשימות

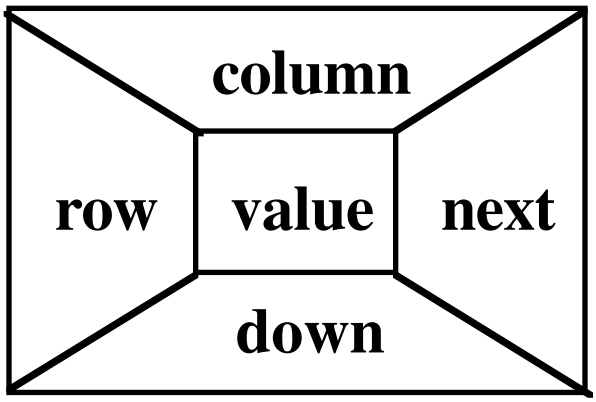


שימושים

מטריצה דלילה -- רוב אבריה = 0.



מבנה צומת:




```
typedef struct node {
   float value;
   struct node *next, *down;
   int row, column;
} NODE;
```

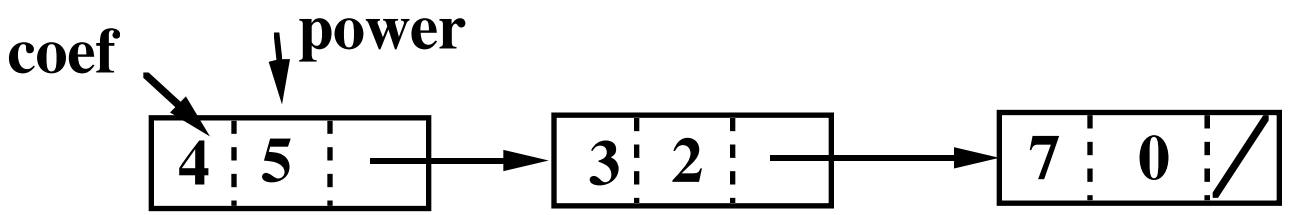
כפל מטריצות $C = A \cdot B$ -- חישוב $c_{i,j}$

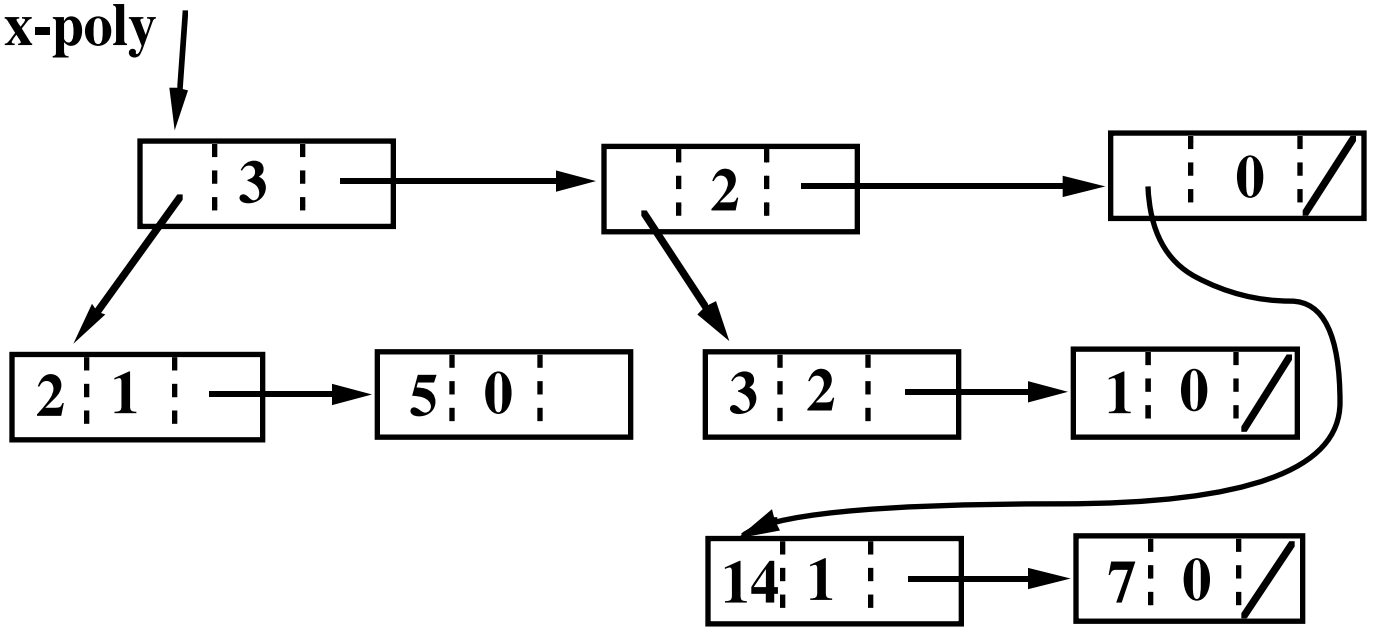
```
#define N_row 20
#define N_row 20

NODE *row_header[N_row],
     *col_header[N_col];
/* compute the value of c[i][j] */
float mult_row_col(int i, int j){
   float c = 0;          הערך המצטבר של c_{i,j}
   NODE *r = row_header[i],
        *k = col_header[j];

   while(r ≠ NULL && k ≠ NULL){
      if(r->column < k->row)
         r = r->next;
      else if(r->column > k->row)
         k = k->down;
      else /* r->column == k->row */{
         c += r->value * k->value;
         r = r->next;
         k = k->down;
   }
   return c;
}
```

מערכות לפולינומים

נעלם אחד

$:4x^5+3x^2+7$



שני נעלמים

$:(2y+5)x^3+\left(3y^2+1\right)x^2+(14y+7)$