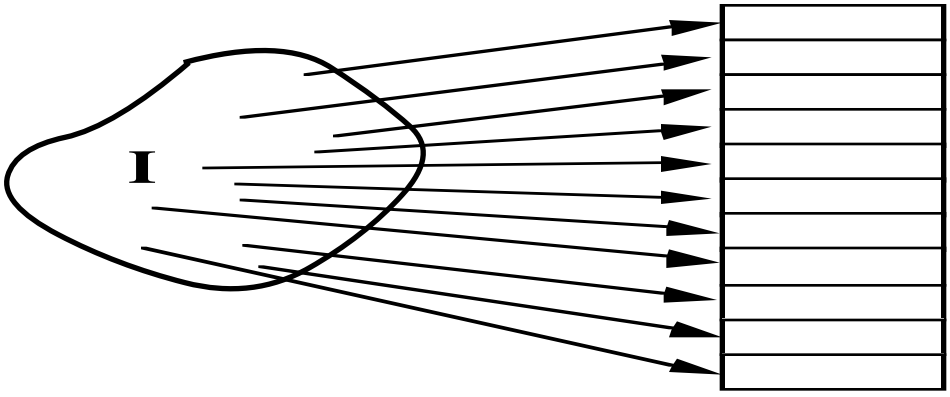
© Itai + Katz September 9, 1997

Arrays מערכים



תהי I קבוצה סופית (אינדכסים).
הפעולות על מערך a (i∈ I):
Get(a,i) (אחזר)

מחזיר את הערך של a לפי אינדכס i

Store(a,i,e) שמור

שמור את הערך הביטוי e ב-a לפי אינדכס i.

בשפת C

קבוצת האינדכסים `I={0,...,n-1}`

אחזר `a[i]`

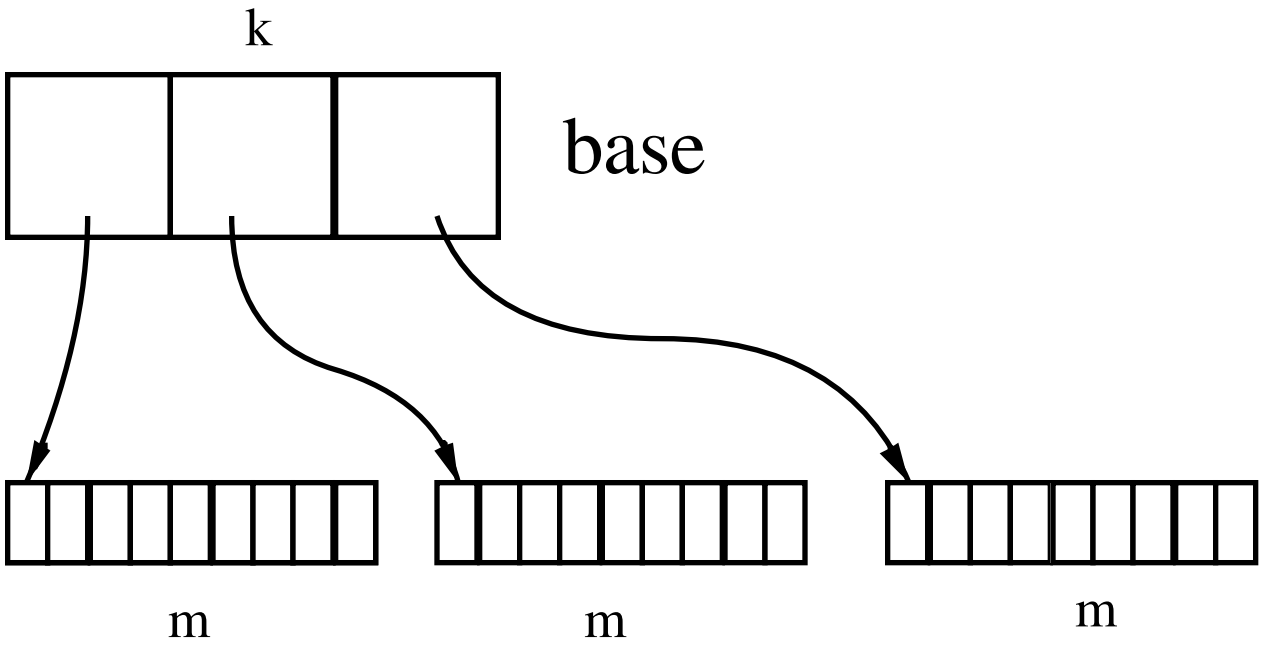
שמור `a[i] = e`

כלל המערך

- הערך המאוחזר לפי אינדכס `i`
  הינו הערך האחרון שנשמר עם אינדכס `i`.
- כל הערכים הם מאותו טיפוס ונקבע בהכרזה.

מימושים למערך

- איזור זכרון רצוף ⇐ שתי הפעולות בזמן O(1).
- מספר איזורים שלכל אחד גודל קבוע m:



כדי לממש מערך באורך n,
יהי $k=\lceil n/m \rceil$
a[i] נמצא בכתובת base[$\lfloor i/m \rfloor$] + i%m

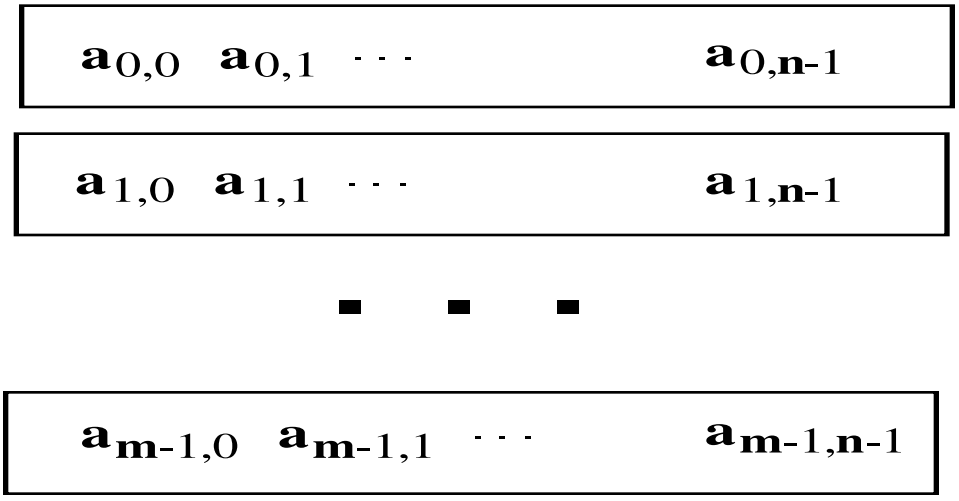
דוגמא:
יהי m=10, i=37:
אזי $\lfloor i/m \rfloor = \lfloor 37/10 \rfloor$
a[37] נמצא בכתובת base[3] + 7

מהי סיבוכיות הזמן של מימוש זה?

מערך דו-ממדי

`int a[m][n]`

כל שורה, `a[i]`, היא מערך חד-ממדי באורך n.



יהי base הכתובת של `a[0][0]`

הכתובת של `a[i]` **base + i·n**

הכתובת של `a[i][j]` **base + i·n + j**

מערך d-ממדי

`int a[n_d][n_{d-1}]...[n_1]`

הכתובת של `a[i_d][i_{d-1}]...[i_1]` :

$$base + (\cdots(i_d n_{d-1} + i_{d-1})n_{d-2} + i_{d-2})n_{d-3} + \cdots + i_2)n_1 + i_1$$

כלל הורנר לחישוב פולינום:

$$p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$$
$$= (\cdots(a_n x + a_{n-1})x + \cdots + a_1)x + a_0$$

מכאן התכנית הבאה תחשב פולינום:

```
p=a[n];
for(j=n-1; j>=0; j--)
  p = p*x + a[j];
```

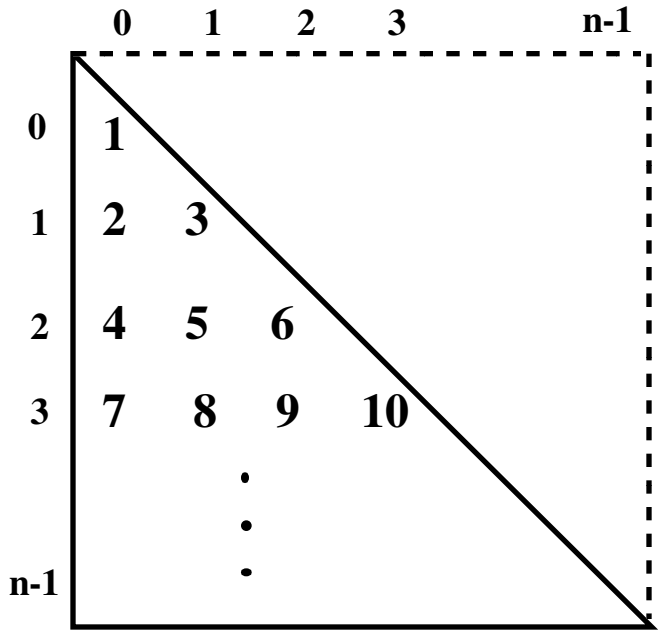
באופן אנאלוגי נחשב את הכתובת

```
addr = i[d];
for(j=d-1; j>=1; j--)
  addr = addr*n[j] + i[j];
```

ייצוג מטריצות מיוחדות

- **מטריצה (מערך דו-ממדי) סימטרית**

$$A[i,j] = A[j,i]$$



<u>הייצוג</u>:

<u>1 2 3 4 5 6 7 8 9 10</u>

איך נמצא את הכתובת של A[i,j] בייצוג זה?

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

$$\mathbf{loc(i,j)} = \begin{cases} \mathbf{base + i(i+1)/2 + j} & \mathbf{if\ \ i \geq j} \\ \mathbf{loc(j,i)} & \mathbf{otherwise} \end{cases}$$

מטריצה דלילה (sparse)

רוב (מוחלט של) האיברים שווים ל-0 (או קבוע c).
מהו רוב מוחלט?

$o(n^2)$ למשל n log n או אפילו O(n).

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

נוותר על גישה אקראית (O(1)), אך נדרוש גישה 'יעילה'.
– ייצוג בשלשות (i,j,v) בסדר לקסיקוגרפי

(0,2,3)

(0,5,1)

(3,0,5)

(4,3,2)

(6,1,4)

(6,5,3)

(6,8,6)

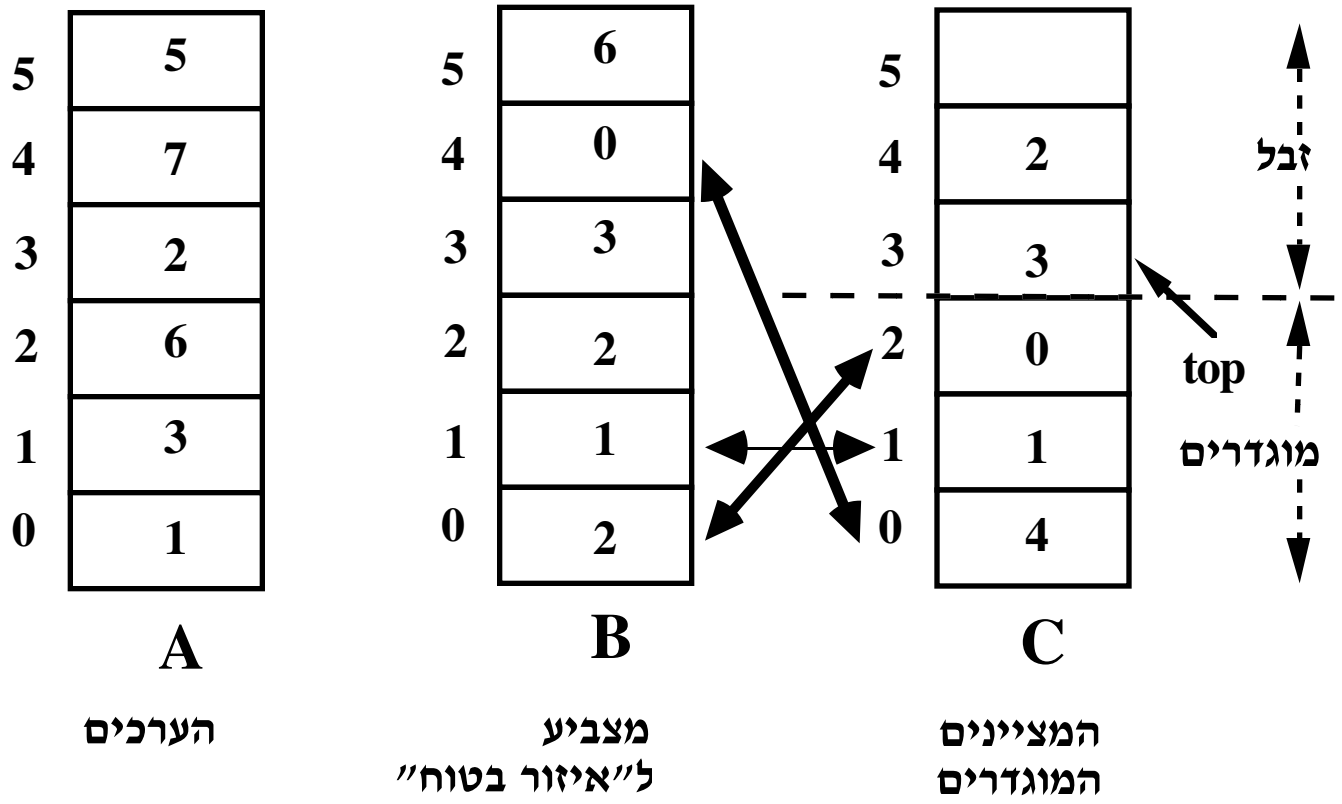
חישוב סכום שתי מטריצות: C = A + B

```
typedef NODE struct {
   float val;
   int row, col;
   }
NODE A[mA+1], B[mB+1], C[mA+mB+1];
int lex(NODE a, NODE b);
/*lexicographically compares a and b,
returns:
      -1  if (a.row,a.col) < (b.row,b.col)
       0  if (a.row,a.col) = (b.row,b.col)
      +1  if (a.row,a.col) > (b.row,b.col)   */

void Add(NODE *A, NODE *B, NODE *C, int *mC){
  int i=0, j=0, *mC=0;
  A[mA]=B[mB]={+∞, +∞, 0};
  while(i<mA || j<mB){
   switch lex(A[i],B[j]){
   case -1: C[*mC++] = A[i++]; break;
   case  1: C[*mC++] = B[j++]; break;
   case  0: C[*mC].row = A[i].row;
        C[*mC].col = A[i].col;
        C[*mC].val = A[i++].val + B[j++].val;
        if(C[*mC].val ≠ 0) *mC++;
   }
  }
}
```

איתחול בזמן O(1)

הבעיה: בייצוג הרגיל איפוס מערך דורש זמן O(n) .
נראה טריק שמאפשר איפוס בזמן קבוע.
זמן איתחול קבוע, על-חשבון מקום.



בדיקה אם במקום i יש זבל:

```
int is_garbage(int i){
  return !((B[i] < top)
           && C[B[i]] == i));
}
```

אתחול:

```
top = 0;
```

קריאה:

```
if(is_garbage(i))
  return GARBAGE;
else
  return A[i];
```

כתיבה:

```
if(is_garbage(i)){
  C[top] = i;
  B[i] = top++;
}
A[i] = v;
```

מערך -- סיכום

<u>יתרונות</u>:

- **גישה אקראית לפי מציין.**
- **זמן קבוע לכתיבה ולקריאה (עם קבוע קטן מאד).**
- **מימוש קל.**

<u>חסרונות</u>:

- **קשה להכניס איבר באמצע ($O(n)$).**
- **כנ"ל להוצאה (צריך לצמצם רווחים).**
- **גודל קבוע וידוע מראש.**
- **האיברים אחידים (הומוגניות).**