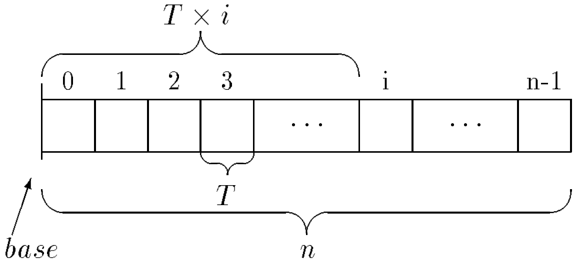
## תרגיל כיתה II

# מערכים ושימושיהם

## 1 מבנה של מערך

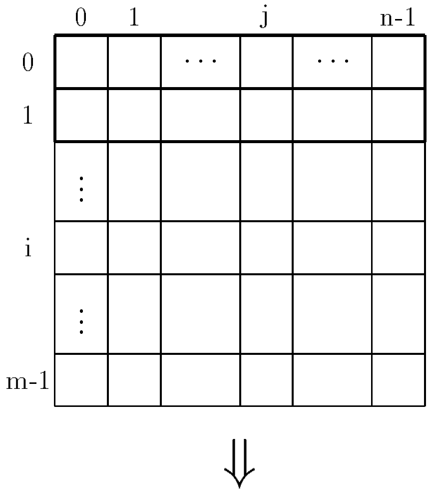
### 1.1 מערך חד-מימדי - וקטור

עבור מערך חד-מימדי באורך $n$ המכיל תאים בגודל $T$ (כולם זהים בגודלם) יוקצה בלוק זכרון בגודל $T \times n$. לדוגמא, למערך המוגדר ע"י הפקודה: `int A[10];` יוקצה בלוק זכרון בן 20 בתים (בהנחה שגודל משתנה מסוג `int` הינו 2 בתים). כלומר, סיבוכיות הזכרון היא $\boxed{O(n)}$.

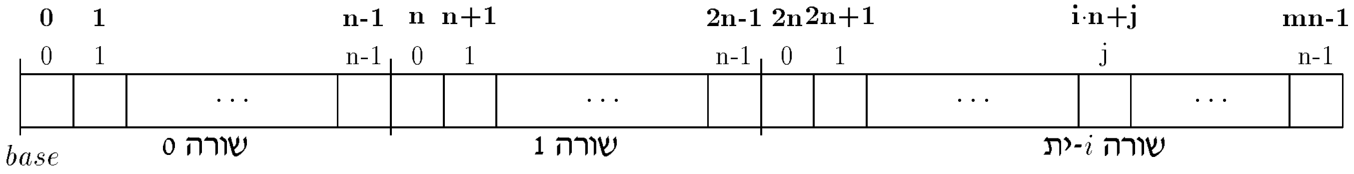



נניח שאינדקסציה במערך הינה החל מ-0. נסמן ב-$base$ את כתובת תחילת המערך בזכרון. אזי חישוב הכתובת של $A[i]$ הינו לפי הנוסחה: $\boxed{base + i \cdot T}$. מכיון שמספר הפעולות בנוסחה הוא קבוע ולא תלוי ב-$n$ או ב-$i$, זמן חישוב הנוסחה אף הוא קבוע. לכן סיבוכיות זמן של גישה לאיבר במערך היא $O(1)$.

### 1.2 מערך דו-מימדי - מטריצה




עבור מערך דו-מימדי באורך $n$ ובגובה $m$ המכיל תאים בגודל $T$ יוקצה בלוק זכרון בגודל $T \times n \times m$. מכאן, סיבוכיות הזכרון הינה $\boxed{O(mn)}$. אבל מבנה הזכרון הינו חד-מימדי, לכן בכדי לאחסן את המטריצה בזכרון "נפרוש" אותה לפי שורותיה ונשרשר שורה אחר שורה. נקח, לדוגמא, מטריצה $M_{m \times n}$. נניח, כמו קודם, שהאינדקסציה במערך הינה מ-0 ו-$base$ מסמנת את כתובת תחילת המערך החד-מימדי (הנוצר עקב שירשור השורות) בזכרון. אזי חישוב הכתובת של $M[i,j]$ הינו לפי הנוסחה: $\boxed{base + (i \cdot n + j)T}$. זמן חישוב הנוסחה עדיין קבוע. לכן סיבוכיות זמן של גישה לאברי במטריצה היא $O(1)$.



## 2 מטריצות מיוחדות

### 2.1 מטריצת אלכסונים (מטריצת טיופליץ)

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 5 & 8 & 4 \\ 1 & 7 & 1 & 5 & 8 \\ 2 & 5 & 7 & 1 & 5 \\ 3 & 6 & 5 & 7 & 1 \end{array}$$

<u>מטריצת אלכסונים</u> – מטריצה ריבועית שבה לאורך כל אלכסון כל האיברים זהים. (ראה בציור דוגמא של מטריצת אלכסונים $4 \times 4$).

דרוש להציע מימוש עבור מטריצת אלכסונים $M_{n\times n}$ המאפשר ביצוע הפעולות הבאות:

- get (i, j) – החזר את הערך של $M[i, j]$.
- put (i, j, x) – הצב את הערך $x$ ל-$M[i, j]$. יש לעדכן גם את שאר הערכים באותו אלכסון.

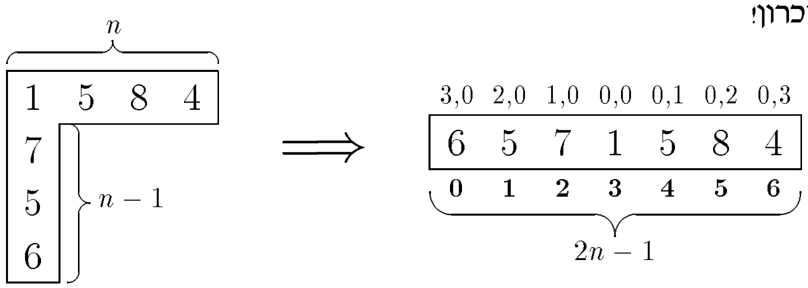
#### 2.1.1 מימוש סטנדרטי

במימוש הסטנדרטי של המטריצה כמערך דו-מימדי (שראינו קודם) סיבוכיות פעולת get(i, j) היא O(1) ואלו סיבוכיות פעולת put(i, j, x) היא $O(n)$ במקרה הגרוע. סיבוכיות המקום הינה $O(n^2)$.

#### 2.1.2 מימוש אחר

נשמור איבר אחד עבור כל אלכסון – יש לשמור רק $2n - 1$ איברים. כל אלכסון מאופיין ע"י ערך קבוע $j - i$ של האינדקסים השייכים לו. למשל, עבור האלכסון הראשי $j - i = 0$, ועבור האלכסון הימני ביותר $j - i = n - 1$.

נקצה מערך $A$ בגודל $2n - 1$. עבור האלכסון המאופיין ע"י $j - i$ הערך המייצג שלו יאוחסן בכניסה $(j - i) + (n - 1)$. לכן כדי לבצע פעולת get(i, j) מספיק לגשת ל-$A[j - i + (n - 1)]$. גם לביצוע פעולת put(i, j, x) מספיק לשנות את הערך של $A[j - i + (n - 1)]$ בלבד! מכיון שהחישוב הנ"ל דורש זמן קבוע, סיבוכיות הזמן של הפעולות היא O(1). סיבוכיות המקום הפעם הינה $O(n)$. כלומר, השגנו שיפור גם בזמן וגם בזכרון!




### 2.2 מטריצה תלת-אלכסונית

<u>מטריצה תלת-אלכסונית</u> – מטריצה ריבועית שבה כל האיברים שווים ל-0 מלבד (אולי) אברי האלכסון הראשי, זה שמעליו וזה שמתחתיו. (ראה בציור דוגמא של מטריצה תלת-אלכסונית $5 \times 5$).

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 2 & 0 & 0 & 0 \\ 1 & 5 & 4 & 1 & 0 & 0 \\ 2 & 0 & 0 & 3 & 0 & 0 \\ 3 & 0 & 0 & 1 & 2 & 3 \\ 4 & 0 & 0 & 0 & 7 & 9 \end{array} \Bigg\} n$$

#### 2.2.1 מימוש סטנדרטי

במימוש הסטנדרטי סיבוכיות הפעולות get(i, j) ו-put(i, j, x) היא O(1). סיבוכיות המקום הינה $O(n^2)$.

### 2.2.2 מימוש אחר

נשמור בוקטור $A$ רק את אברי שלושת האלכסונים ה"מעניינים" – יש לשמור רק $3n-2$ איברים. נשרשר את האלכסונים כך שקודם יופיעו אברי האלכסון העליון, אחריהם אברי האלכסון הראשי ולבסוף אברי האלכסון התחתון. בתא הראשון של המערך נשמור את הערך הנמצא בשאר המקומות במטריצה (במקרה שלנו זהו 0, אבל המימוש מאפשר לו להיות ערך כלשהו אחר).

| אחרים | 0,1 | 1,2 | 2,3 | 3,4 | 0,0 | 1,1 | 2,2 | 3,3 | 4,4 | 1,0 | 2,1 | 3,2 | 4,3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 1 | 0 | 3 | 1 | 4 | 3 | 2 | 9 | 5 | 0 | 1 | 7 |
| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** |

$\underbrace{1 \;\dots\; 13}_{3n-2}$

כעת נציג פונקציה, אשר בהינתן זוג $i, j$ – אינדקסים במטריצה המקורית $M$ "מתרגמת" אותו לאינדקס בוקטור הייצוג $A$:

```
int matrix2array (int i, int j) {
  if (i == j-1) return (i+1);        // האלכסון העליון
  if (i == j) return (i+n);          // האלכסון הראשי
  if (i == j+1) return (j+2*n);      // האלכסון התחתון
  return 0;                          // שאר האיברים
}
```

כעת ניתן לבצע פעולות $\mathrm{get(i, j)/put(i, j, x)}$ ע"י גישה ל-$A[matrix2array(i,j)]$. סיבוכיות הזמן של הפונקציה הנ"ל הינה $\mathrm{O}(1)$, וכך גם של הפעולות. סיבוכיות המקום היא $\mathrm{O}(n)$. כלומר, שמרנו על סיבוכיות הזמן המקורית וירדנו בדרישות הזכרון.

## 2.3 מטריצה דלילה

<u>מטריצה דלילה</u> – מטריצה שבה רוב האיברים שווים ל-0.
(ראה בציור דוגמא של מטריצה דלילה $4 \times 4$).

$$
\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}
\begin{array}{c}
\begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\underbrace{\left.\boxed{\begin{array}{cccc} 2 & 0 & 0 & 1 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 5 & 0 \end{array}}\right\} m}_{n}
\end{array}
$$

### 2.3.1 מימוש סטנדרטי

נסמן את המטריצה $M_{m \times n}$. במימוש הסטנדרטי סיבוכיות הפעולות $\mathrm{get}$ ו-$\mathrm{put}$ היא $\mathrm{O}(1)$. סיבוכיות המקום הינה $\mathrm{O}(mn)$.

### 2.3.2 מימוש אחר

נניח שידוע חסם $r$ על מספר האיברים השונים מ-0 במטריצה ומתקיים כי $r \ll m \cdot n$. אזי ניתן לממש את המטריצה כדלקמן. נקצה מערך חד-מימדי $A$ באורך $r$ ונשמור בו רק את האיברים השונים מ-0 בצורה הבאה: כל תא ב-$A$ יכיל זוג האינדקסים $i, j$ ואת הערך $M[i, j]$. המערך יהיה ממוין בסדר לקסיקוגרפי (כמו במילון) לפי המפתח $(i, j)$. למשל, המטריצה הנ"ל במימוש החדש תיראה כך:

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| row | 0 | 0 | 1 | 2 | 3 |
| col | 0 | 3 | 2 | 2 | 2 |
| val | 2 | 1 | 3 | 1 | 5 |

$\underbrace{\hspace{6em}}_{r}$

<u>הערה:</u> סדר לקסיקוגרפי מוגדר כך:

$$(a, b) > (c, d) \iff a > c \quad \text{or} \quad \Big(a = c \quad \text{and} \quad b > d\Big)$$

- מימוש פעולת $\mathrm{get(i,\ j)}$ יתבצע ע"י חיפוש בינארי של אינדקס $i,j$ במערך. אם האינדקס לא נמצא הרי שבמטריצה בכניסה $i,j$ מופיע 0. חיפוש בינארי במערך ממויין לוקח $\mathrm{O}(\log r)$ זמן. לכן גם סיבוכיות זמן של הפעולה הינה $\mathrm{O}(\log r)$.
- מימוש פעולת $\mathrm{put(i,\ j,\ x)}$ אף הוא יבצע קודם כל חיפוש בינארי של אינדקס $i,j$. אם האיבר נמצא יש לשנות את ערכו ל-$x$, אחרת נצטרך להכניס איבר חדש למבנה ולהזיז את כל אלה שאחריו. לכן סיבוכיות זמן של הפעולה במקרה הגרוע היא $\mathrm{O}(\log r) + \mathrm{O}(r) = \mathrm{O}(r)$.
- סיבוכיות המקום של המבנה הינה $\mathrm{O}(r)$.

כלומר, שיפרנו סיבוכיות המקום על חשבון הזמן. לכן לפני לבחור בין שני המימושים הנ"ל צריך קודם להחליט מה חשוב ומה כדאי יותר.

### 2.3.3 ביצוע transpose במימוש החדש

בכדי לקבל את ייצוג המטריצה המוחלפת $M^t$ מספיק בעצם להחליף שורות של $i$ ושל $j$ בינהן. אבל אז המערך יפסיק להיות ממויין. לכן בנוסף נצטרך למיין מחדש את המערך המייצג את $M^t$ (נקרא לו $A^t$) עפ"י המפתחות $(j,i)$ (הישנים).
נתבונן בשני אלגוריתמי מיון:

1. אלגוריתם פשוט יותר:
נחפש כל פעם את האיבר הקטן ביותר ב-$A$, נמחקו מ-$A$ ונוסיפו ל-$A^t$.
כך נבצע $r$ איטרציות (עד שנמחק את כל אברי $A$) ובכל איטרציה נבצע חיפוש הלוקח זמן $\mathrm{O}(r)$.
לכן סה"כ סיבוכיות האלגוריתם היא $\mathrm{O}(r^2)$.
2. אלגוריתם יעיל יותר:
נשתמש בעובדה שאחרי החלפת $i$ ו-$j$, איברים השייכים לאותה עמודה נשארים ממויינים בינהם. אזי נותר רק לקבץ אותם יחד. לשם כך, נספור כמה איברים (שונים מ-0, כמובן) נמצאים בכל עמודה ולפי הנתונים האלה נחלק את מערך התוצאה $A^t$ לקטעים שאורכם יהיו כאורכי העמודות המתאימות. לאחר מכן נעבור על המערך $A$ לפי הסדר ונעתיק איברים לקטעים המתאימים. להלן פירוט האלגוריתם והרצתו על המטריצה מהדוגמא:

   (א) נגדיר מערך $L$ בגודל $n$. ב-$L[i]$ נספור כמה אברי $A$ נמצאים בעמודה $i$-ית.
   סיבוכיות הספורה היא $\mathrm{O}(r+n)$ (אתחול $L$ ומעבר על $A$).

```
for (i=0; i<n; i++)   L[i] = 0;           // O(n) סיבוכיות
for (i=0; i<r; i++)   L[A[i].col]++;      // O(r) סיבוכיות
```

תוצאות ההרצה על המטריצה מהדוגמא:

$$L = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 1 & 0 & 3 & 1 \end{array}$$

   (ב) עבור כל עמודה $i$ נספור כמה איברים נמצאים בעמודות הקודמות לה (כך נדע באיזה אינדקס ב-$A^t$ מתחילה כל עמודה). את התוצאה נשמור במערך $S$. סיבוכיות פה היא $\mathrm{O}(n)$.

```
S[0] = 0;
for (i=0; i<n; i++)   S[i] = S[i-1] + L[i-1];        // O(n) סיבוכיות
```

תוצאות ההרצה על המטריצה מהדוגמא:

$$S = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 1 & 4 \end{array}$$

(ג) כעת ניתן לעבור על המערך $A$ ולשים כל איבר במקומו הסופי המוצבע ע"י האיבר המתאים מ-$S$. סיבוכיות הזמן היא $\mathrm{O}(r)$.

```
int for (i=0; i<r; i++) {           //  מעבר על A
  k = A[i].col;                     //  k – מספר העמודה
  A^t[S[k]].col = A[i].row;         //  S[k] – המיקום החדש
  A^t[S[k]].row = A[i].col;
  A^t[S[k]].val = A[i].val;
  S[k]++;                           //  קידום המצביע
}
```

הרצה על הדוגמא:

| | | | | |
|---|---|---|---|---|
| $S[0]=1$ | $,B[0]=(0,0,2)$ | $\Longleftarrow$ | $S[0]=0$ | $,(0,0,2)$ |
| $S[3]=5$ | $,B[0]=(3,0,1)$ | $\Longleftarrow$ | $S[3]=4$ | $,(0,3,1)$ |
| $S[2]=2$ | $,B[0]=(2,1,3)$ | $\Longleftarrow$ | $S[2]=1$ | $,(1,2,3)$ |
| $S[2]=3$ | $,B[0]=(2,2,1)$ | $\Longleftarrow$ | $S[2]=2$ | $,(2,2,1)$ |
| $S[2]=4$ | $,B[0]=(2,3,5)$ | $\Longleftarrow$ | $S[2]=3$ | $,(3,2,5)$ |

$$\mathrm{A}^t = \begin{array}{c|c|c|c|c|c|} & 0 & 1 & 2 & 3 & 4 \\ \hline \text{row} & 0 & 2 & 2 & 2 & 3 \\ \hline \text{col} & 0 & 1 & 2 & 3 & 0 \\ \hline \text{val} & 2 & 3 & 1 & 5 & 1 \\ \hline \end{array}$$

<u>הערה:</u> הסיבוכיות הכוללת של האלגוריתם הינה $\mathrm{O}(n+r)$. זה שיפור רציני לעומת האלגוריתם הקודם. אך יש לשים לב, כי במטריצה דלילה יתכן ש-$n < r^2$, ואז כדאי להשתמש באלגוריתם הראשון, שהוא גם יותר קל למימוש.