



## מבוא למדעי מחשב מ' / ח' (234117 / 234114)

### מסטר חורף תשס"ז

### פתרון מבחן מסכם מועד א', 15 פברואר 2007

<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
שם פרטי	שם משפחה	מספר סטודנט							

משך המבחן: 3 שעות.  
חומר עזר: אין להשתמש בכל חומר עזר בכתב, מודפס או אלקטרוני.

#### הנחיות והוראות:

- מלאו את הפרטים בראש דף זה.
- בדקו שיש 24 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- כתבו את התשובות על טופס המבחן בלבד, במקומות המיועדים לכך. שימו לב שהמקום המיועד לתשובה אינו מעיד בהכרח על אורך התשובה הנכונה.
- העמודים הזוגיים בבחינה ריקים. ניתן להשתמש בהם כדפי טיוטה וכן לכתוב תשובותיכם. סמנו טיוטות באופן ברור על מנת שהן לא תבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר. אין לכתוב הערות והסברים לתשובות אם לא נתבקשתם מפורשות לכך.
- בכל השאלות, הינכם רשאים להגדיר ולממש פונקציות עזר כרצונכם. לנוחיותכם, אין חשיבות לסדר מימוש הפונקציות, ובפרט ניתן לממש פונקציה לאחר הפונקציה שמשתמשת בה.
- אין להשתמש בפונקציות ספרייה או בפונקציות שמומשו בכיתה אלא אם צויין אחרת בשאלה.

**הערה:** לחלק מהפתרונות המובאים כאן מצורף הסבר ו/או תיעוד. הסברים אלו הינם לצרכי הבהרה של הפתרון בלבד, ואינם חלק מן התשובה הנדרשת במבחן.

צוות הקורס
<b>מרצים:</b> סאהר אסמיר, תמיר לוי, דר' מיכאל אלעד (מרצה אחראי).
<b>מתרגלים:</b> איתי שרון, אנדריי קלינגר, אסנת טל, אריה מצליח, ירון יורה, נוגה טל, עידו חניאל, רג'א גיריס, רון בגלייטר, סשה סקולוזוב, גיל בכר, צפריר קמלו, רן רובינשטיין (מתרגל אחראי).

שאלה	ערך	הישג	בודק
1	25		
2	25		
3	25		
4	25		
סה"כ	100		

**בהצלחה!**



## הטכניון, מכון טכנולוגי לישראל מבוא למדעי המחשב מ'ח'

- 2 -



## שאלה 1 (25 נקודות)

סעיף א

נתונה תוכנית ה-C הבאה, כאשר הפרמטר N הוא קבוע #define שלם וחייבי כלשהו:

```
int i = 2*N;

void strange(int *p)
{
    int j = *p;
    while (j) {
        printf("Backtracking is strange");
        j--;
    }
    *p = *p-1;
}

void charm()
{
    static int k = 8;
    int i;

    while (k) {
        printf("Arrays are charming");
        k = k/2;
    }
    i = 2*k;
}

int main ()
{
    while (i>0) {
        strange(&i);
        charm();
        i--;
    }
    return 0;
}
```

כתבו כמה פעמים תודפס כל אחת מן המחרוזות "Backtracking is strange" ו-"Arrays are charming!" כפונקציה של N (הכוונה למספר מדויק, לא לקירוב אסימפטוטי):

"Backtracking is strange": $N^2+N$	"Arrays are charming": 4
------------------------------------	--------------------------



הסבר:

**Backtracking is strange**: באיטרציה ה- $i$  השורה מודפסת  $i$  פעמים.  $i$  מתחיל מ- $2N$  ומופחת כל פעם בשתיים (פעם בתוך הלולאה ופעם בפונקציה `strange`, דרך המצביע  $p$ ) ולכן נקבל כסך הפעמים שהשורה מודפסת סדרה חשבונית:

$$2N + (2N-2) + (2N-4) + \dots + 2 = N^2 + N$$

הערה: במבחן ניתנו מלוא הנקודות גם למי שכתב את התוצאה כסכום ללא שימוש בנוסחת סכום סדרה חשבונית.

**Arrays are charming**: המשתנה  $k$  סטטי ולכן הוא מאותחל רק פעם אחת בתחילת ריצת התוכנית, וערכו נשמר בין הקריאות לפונקציה. יוצא שהשורה מודפסת רק 4 פעמים בקריאה הראשונה לפונקציה `charm`, ולאחר מכן כל קריאה נוספת אינה גורמת להדפסות נוספות.



## סעיף ב

בכל אחד מהסעיפים הבאים מופיעות מספר שורות קוד. לכל קטע קוד, הקיפו בעיגול את התיאור המתאים:

- א. **ללא שגיאות** – הקוד יתקמפל ללא כל שגיאה וירוך ללא תקלות.  
ב. **שגיאת זמן ריצה** – הקוד יתקמפל ללא שגיאות, אולם הוא עלול לבצע שגיאה בזמן הריצה.  
ג. **שגיאת קומפילציה** – הקוד לא יעבור קומפילציה.

1.

```
int i = 0;  
int *j = *i;
```

- א. ללא שגיאות  
ב. שגיאת זמן ריצה  
ג. שגיאת קומפילציה

**הסבר:** הפעלת האופרטור\* על משתנה שאיננו מצביע.

2.

```
int i[8] = {0,1,2,3,4,5};  
*(i + 2) = 8;
```

- א. ללא שגיאות  
ב. שגיאת זמן ריצה  
ג. שגיאת קומפילציה

3.

```
int *j;  
int *i = j;  
*i = 5;
```

- א. ללא שגיאות  
ב. שגיאת זמן ריצה  
ג. שגיאת קומפילציה

**הסבר:** כתיבה לזיכרון (\*i) שלא הוקצה.

4.

```
int *i, j, k;  
i = &(j+k);
```

- א. ללא שגיאות  
ב. שגיאת זמן ריצה  
ג. שגיאת קומפילציה

**הסבר:** הפעלת האופרטור & על ביטוי ולא על משתנה.

5.

```
char* s = "Hello World!";  
while (*s) {  
    *s = *s + 1;  
}
```

- א. ללא שגיאות  
ב. שגיאת זמן ריצה  
ג. שגיאת קומפילציה

**הסבר:** כתיבה לזיכרון שמוגדר כקריאה בלבד (אזור קבועים).



- 6 -



## שאלה 2 (25 נקודות)

נתון מערך `order` באורך קבוע של 26 (שימו לב להערה בהמשך). מערך זה מכיל את כל האותיות הקטנות בא"ב האנגלי, כל אות פעם אחת בדיוק. המערך מגדיר סדר חדש על אותיות הא"ב, כאשר על פי סדר זה, אות אחת תחשב לקטנה מאות אחרת אם ורק אם אות זו נמצאת במערך `order` לפני האות האחרת.

לדוגמה, עבור המערך `order` הבא מתקיים  $b < a$ , כיוון ש-`b` מופיע לפני `a` במערך. לעומת זאת, יתר האותיות במקרה זה הן בסדר לכסיקוגרפי רגיל.

`order[26] = { 'b','a','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z' } ;`

**הערה:** בשאלה זו, מספר האותיות בא"ב האנגלי (26) יכול להיחשב כקבוע לצורך חישובי סיבוכיות.

### סעיף א

השלימו את הפונקציה הבאה, המקבלת כקלט מחרוזת `s` המכילה אותיות אנגליות קטנות בלבד, את אורכה של המחרוזת `n` (לא כולל תו ה-`null`), ומערך `order` המתאר סדר של אותיות הא"ב. הפונקציה מחזירה 1 אם האותיות ב-`s` מסודרות על פי הסדר המוגדר במערך `order`, ו-0 אחרת (עבור מחרוזת ריקה החזירו 1). **יש לממש את הפונקציה בסיבוכיות זמן ומקום טובים ככל הניתן.** פתרון לא אופטימאלי יזכה בניקוד מופחת. כמו כן, השלימו את סיבוכיות האלגוריתם שלכם במקום המתאים למטה, כפונקציה של `n`.

```
int is_ordered(char *s, int n, char order[26]) {  
  
    int i=0, j=0;  
  
    while (j<n) {  
        while (i<26 && order[i] != s[j])  
            i++;  
        if (i==26)  
            return 0;  
        j++;  
    }  
    return 1;  
}
```

**הערה:** פתרון נפוץ אחר הינו להגדיר פונקצית עזר `compare(char a, char b, char order[26])` המשווה בין שני תווים `a` ו-`b` בהתאם לסדר המוגדר ב-`order`, ובאמצעות פונקציה זו לבצע סריקה של המחרוזת `s` ולוודא שכל שני תווים עוקבים בה מסודרים נכון.

סיבוכיות מקום נוסף:  $\theta(1)$

סיבוכיות זמן:  $\theta(n)$



- 8 -





## סעיף ב

בסעיף זה נממש פונקציה המבצעת חיפוש של אות נתונה במחרוזת ממוינת. השלימו את הפונקציה הבאה, המקבלת מחרוזת  $s$  הממוינת על פי סדר המוגדר ב- $order$ , את אורך המחרוזת  $n$  (לא כולל תו ה- $\text{null}$ ) וכן אות לחיפוש  $x$ . על הפונקציה להחזיר את האינדקס של המופע האחרון של  $x$  במחרוזת  $s$ , או -1 אם  $x$  אינה מופיעה במחרוזת.

לדוגמה, עבור הסדר  $order$  בדף הקודם והמחרוזת הבאה ( $n=9$ ):

```
char s[] = "bbacdddef"
```

על הפונקציה להחזיר 2 במקרה של חיפוש האות  $a$ , 1 במקרה של חיפוש האות  $b$ , 6 במקרה של חיפוש האות  $d$  ו-(-1) במקרה של חיפוש האות  $y$ .

יש לממש את הפונקציה בסיבוכיות זמן ומקום טובים ככל הניתן. פתרון לא אופטימאלי יזכה בניקוד מופחת. כתבו את סיבוכיות האלגוריתם שלכם במקום המתאים:

סיבוכיות מקום נוסף:  $\theta(1)$

סיבוכיות זמן:  $\theta(\log n)$

```
int search(char *s, int n, char order[26], char x) {  
  
    int low=0, high=n-1, mid, i;  
    char char_table[26];           // char_table[i] contains the index  
                                   // of the letter 'a'+i in order[]  
    for (i=0; i<26; ++i)  
        char_table[order[i] - 'a'] = i;  
  
    while (low <= high) {  
        int mid = (high+low)/2;  
        int compare = char_table[x-'a'] - char_table[s[mid]-'a'];  
  
        // if characters are equivalent and it is the last occurrence  
        if (compare == 0 && (mid==n-1 || s[mid+1]!=s[mid]))  
            return mid;  
        else if (compare < 0)  
            high = mid-1;  
        else  
            low = mid+1;  
    }  
    return -1;  
}
```



- 10 -



### שאלה 3 (25 נקודות)

נתונה מטריצה  $a$  בגודל  $N$  על  $N$ , המכילה מספרים שלמים. לכל איבר  $a[i][j]$  במטריצה זו, נגדיר את ארבע תת המטריצות הבאות שאיבר זה מהווה פינה שלהם:

תת-מטריצה שמאלית עליונה: האיברים  $a[0..i][0..j]$   
ימנית עליונה: האיברים  $a[0..i][j..N-1]$   
שמאלית תחתונה: האיברים  $a[i..N-1][0..j]$   
ימנית תחתונה: האיברים  $a[i..N-1][j..N-1]$

לדוגמה, עבור המטריצה הבאה ( $N=4$ ),

2	1	2	2
6	1	2	2
5	1	2	3
8	2	7	1

האיבר  $a[2][1]$  (המסומן באפור) מגדיר את ארבע תת-המטריצות:

<table><tr><td>2</td><td>1</td></tr><tr><td>6</td><td>1</td></tr><tr><td>5</td><td>1</td></tr></table>	2	1	6	1	5	1	<table><tr><td>1</td><td>2</td><td>2</td></tr><tr><td>1</td><td>2</td><td>2</td></tr><tr><td>1</td><td>2</td><td>3</td></tr></table>	1	2	2	1	2	2	1	2	3
2	1															
6	1															
5	1															
1	2	2														
1	2	2														
1	2	3														
<table><tr><td>5</td><td>1</td></tr><tr><td>8</td><td>2</td></tr></table>	5	1	8	2	<table><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>2</td><td>7</td><td>1</td></tr></table>	1	2	3	2	7	1					
5	1															
8	2															
1	2	3														
2	7	1														

אנו נאמר כי איבר  $a[i][j]$  במטריצה מאוזן אם סכום האיברים בכל אחת מארבע תת המטריצות שהוא מגדיר זהה. למשל, בדוגמה למעלה  $a[2][1]$  מאוזן, כיוון שסכום כל אחת מארבע תת המטריצות שהוא מגדיר שווה ל-16: שימו לב שעבור תת-המטריצה השמאלית העליונה מתקיים  $2+6+5+1+1+1=16$ , עבור תת המטריצה השמאלית התחתונה מתקיים  $5+1+8+2=16$ , וכך הלאה.

עליכם לממש (בדף הבא) פונקציה שמקבלת מטריצה המיוצגת כמערך דו-ממדי  $a[N][N]$  (מוגדר כ-`#define`). הפונקציה מחזירה 1 אם קיים במטריצה איבר מאוזן, ו-0 אחרת.

על הפונקציה לעבוד בסיבוכיות זמן  $O(N^2)$  וסיבוכיות מקום  $O(N^2)$ . שימו לב: פתרונות בסיבוכיות גרועה מזו יזכו בניקוד מופחת, בהתאם לסיבוכיות הזמן של הפתרון.



- 12 -



```
int equal_submatrix_sum(int a[N][N]) {  
  
    int i,j;  
    int ul[N][N] = { 0 };  
  
    init_ul_matrix(a, ul);  
  
    for (i=0; i<N; ++i) {  
        for (j=0; j<N; ++j) {  
  
            // sums of upper-left, upper-right, lower-left and  
            // lower-right sub-matrices  
            int ul_sum, ur_sum, ll_sum, lr_sum;  
  
            ul_sum = ul[i][j];  
            ur_sum = ul[i][N-1] - (j>0 ? ul[i][j-1] : 0);  
            ll_sum = ul[N-1][j] - (i>0 ? ul[i-1][j] : 0);  
            lr_sum = ul[N-1][N-1] - (i>0 ? ul[i-1][N-1] : 0)  
                - (j>0 ? ul[N-1][j-1] : 0)  
                + (i>0 && j>0 ? ul[i-1][j-1] : 0);  
  
            if ( ul_sum == ur_sum && ur_sum == ll_sum &&  
                ll_sum == lr_sum )  
                return 1;  
        }  
    }  
    return 0;  
}
```



```
void init_ul_matrix(int a[N][N], int ul[N][N])
```

```
{
```

```
    int i,j;
```

```
    for (i=0; i<N; ++i) {
```

```
        for (j=0; j<N; ++j) {
```

```
            ul[i][j] = a[i][j];
```

```
            if (i>0)
```

```
                ul[i][j] += ul[i-1][j];
```

```
            if (j>0)
```

```
                ul[i][j] += ul[i][j-1];
```

```
            if (i>0 && j>0)
```

```
                ul[i][j] -= ul[i-1][j-1];
```

```
        }
```

```
    }
```

```
}
```

**הסבר:** המערך `ul[]` מאוחל להכיל את הסכומים של תת-המטריצות השמאליות-עליונות:

```
ul[i][j] = sum { a[0..i][0..j] }
```

מערך זה ניתן לחישוב בזמן  $O(N^2)$ , כיוון שלכל  $j$ , מתקיים השיויון

```
ul[i][j] = a[i][j] + ul[i-1][j] + ul[i][j-1] - ul[i-1][j-1]
```

ולכן חישוב כל תא דורש מספר קבוע של פעולות.

מתוך מערך העזר `ul[]`, ניתן לחשב ביעילות לכל תא  $j$  ב- $a$  את סכום ארבע תת-המטריצות שסביבו.

לדוגמה, סכום תת המטריצה השמאלית-תחתונה מתקבל באמצעות הנוסחה

```
sum { a[i..N-1][0..j] } = ul[N-1][j] - ul[i-1][j]
```

וכן הלאה.



#### שאלה 4 (25 נקודות)

בשאלה זו נתכנת פונקציה שמנסה לסדר אבני דומינו על לוח. הלוח מתואר באמצעות מערך דו מימדי  $a$  שגודלו  $N$  על  $N$  (כש- $N$  קבוע  $\#define$ ). חלק ממשבצות הלוח ריקות, וחלקן מלאות. הפונקציה תקבל את המערך הדו מימדי  $a$  כאשר בכל משבצת ריקה ישנו 0, ובכל משבצת מלאה יש 1. עליכם לממש פונקציה שבודקת אם ניתן לרצף את משבצות הלוח הריקות בעזרת אבני דומינו בגודל 1 על 2, ובמידה וכן, מחזירה את הפתרון שמצאה.

אבני הדומינו יצוינו על ידי מספרים טבעיים: 1,2,3... (ניתן להניח שמספר אבני הדומינו אינו מוגבל). בתום ריצת הפונקציה, אם אכן ניתן לרצף את הלוח אזי התא  $a[i][j]$  יכיל את מספרה של אבן הדומינו שמוקמה בתא זה (אם לא ניתן לרצף אז אין חשיבות לתוכן  $a$  בסוף הריצה). כמו כן, הפונקציה תחזיר 1 במידה ומצאה פתרון, ו-0 אם אין פתרון. שימו לב שניתן למקם אבן דומינו על הלוח במאוזן או במאונך – ראו דוגמאות להלן.

הערה: אם יש יותר מאפשרות אחת לריצוף הלוח, ניתן להחזיר פתרון אחד כלשהו.

##### דוגמא ראשונה:

$N=3$

```
a[3][3]={ 0, 0, 0
           0, -1, 0,
           0, 0, 0 }
```

אחרי ריצת הפונקציה פתרון אפשרי הוא:

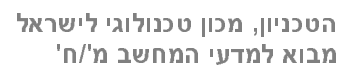
```
a[3][3]={ 1, 1, 2
           3, -1, 2,
           3, 4, 4 }
```

##### דוגמא שנייה:

$N=3$

```
a[3][3]={ 0, -1, 0
           0, 0, 0,
           0, 0, 0 }
```

לוח כזה אי אפשר לרצף.



- 16 -





לנוחותכם, ניתן להשתמש בפונקציות העזר הבאות:

פונקציה שמחזירה 1 אם התא  $(i, j)$  בתחום הלוח:

```
int on_board(int i, int j)
{
    if (i>=0 && i<N && j>=0 && j<N)
        return 1;
    return 0;
}
```

פונקציה שמקבלת זוג אינדקסים  $(i, j)$  של ריבוע בלוח, ומעדכנת אותם כך שיכילו את האינדקסים של הריבוע הבא בלוח. הפונקציה מתקדמת לריבוע הבא באותה השורה במידה ויש כזה, או לריבוע הראשון בשורה הבאה במידה והגענו לקצה השורה. הפונקציה מחזירה 0 אם הגענו לקצה הלוח:

```
int next_square(int* i, int* j)
{
    (*j)++;
    if (*j == N)
    {
        *j = 0;
        (*i)++;
        if (*i == N)
            return 0;
    }
    return 1;
}
```

פונקציה שמחזירה 1 אם התא  $(i, j)$  הינו ריבוע פנוי בלוח (0 אחרת):

```
int square_is_free(int i, int j, int a[N][N])
{
    if (on_board(i,j) && a[i][j]==0)
        return 1;
    return 0;
}
```

פונקציה שמחזירה 1 אם הלוח מלא (0 אחרת):

```
int board_is_full(int a[N][N])
{
    int i=0, j=0;
    do {
        if (square_is_free(i,j,a))
            return 0;
    } while (next_square(&i, &j));

    return 1;
}
```



- 18 -



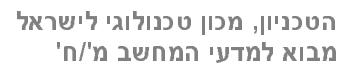
פונקציה שמחזירה 1 אם ניתן להניח לוחית דומינו בתא ה-(i, j) בלוח, ו-0 אחרת. עבור הפרמטר horizontal==0 יבדקו זוג הריבועים האופקיים (i, j), (i, j+1), ואילו עבור הפרמטר horizontal==1 יבדקו זוג הריבועים האנכיים (i, j), (i+1, j):

```
int can_put_tile(int i, int j, int a[N][N], int horizontal)
{
    if (!square_is_free(i, j, a))
        return 0;
    if (horizontal && square_is_free(i, j+1, a))
        return 1;
    if (!horizontal && square_is_free(i+1, j, a))
        return 1;
    return 0;
}
```

נתונה פונקציה המעטפת הבאה, המקבלת את הלוח ההתחלתי כפרמטר, וכותבת לתוכו את הפתרון (אם קיים כזה). הפונקציה מחזירה 1 אם אמנם נמצא פתרון, ו-0 אחרת. שימו לב שפונקציה זו עוטפת קריאה לפונקציה tile\_board\_aux. עליכם להשלים את הפרמטרים המתאימים בקריאה לפונקציה tile\_board\_aux, ולממש פונקציה זו בדף הבא.

**שימו לב:** ניתן לקבוע את זהות ומספר הפרמטרים של הפונקציה tile\_board\_aux כרצונכם.

```
int tile_board(int a[N][N])
{
    return tile_board_aux(a, 1);
}
```



- 20 -



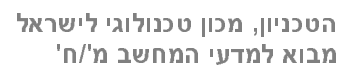
```
int tile_board_aux( int a[N][N], int index ) {  
  
    int i=0, j=0;  
  
    if (board_is_full(a))  
        return 1;  
  
    // check for all i,j on the board if tile can be placed  
    do {  
        if (can_put_tile(i,j,a,0)) {    // vertical positioning  
            a[i][j] = index;  
            a[i+1][j] = index;  
            if (tile_board_aux(a, index+1))  
                return 1;  
            a[i][j] = 0;  
            a[i+1][j] = 0;  
        }  
  
        if (can_put_tile(i,j,a,1)) {    // horizontal positioning  
            a[i][j+1] = index;  
            a[i][j] = index;  
            if (tile_board_aux(a, index+1))  
                return 1;  
            a[i][j+1] = 0;  
            a[i][j] = 0;  
        }  
  
    } while (next_square(&i, &j));  
  
    return 0;  
}
```



- 22 -



- 23 -



- 24 -