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

סמסטר חורף תשס"ח

מבחן מסכם מועד ב'-חדש, 28 מאי 2008

מספר סטודנט	*שם משפחה*	*שם פרטי*

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 22 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 לכתיבת תשובותיכם. סמנו טיוטות באופן ברור על מנת שהן לא תיבדקנה.
- 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.
- **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה אלא אם צוין אחרת בשאלה.**
- **פתרון שלא עומד בדרישות הסיבוכיות יקבל ניקוד חלקי בלבד.**

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

צוות הקורסים 234114/7
מרצים: פרופ'/ח' מיכאל אלעד (מרצה אחראי), סאהר אסמיר, ד"ר צחי קרני, רן רובינשטיין.
מתרגלים: אלדר אהרוני, גדי אלקסנדרוביץ', רון בגלייטר, שגיא בן-משה, אורי זבולון, מרק זילברשטיין, סשה סקולוזוב, אנדרי קלינגר (מתרגל אחראי), ולנטין קרבצוב, אייל רגב, אייל רוזנברג.

בהצלחה!

שאלה 1 (25 נקודות)

סעיף א

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

א. **ללא שגיאות** – הקוד יתקמפל ללא כל שגיאה וירוץ ללא תקלות.
ב. **שגיאת זמן ריצה** – הקוד יתקמפל ללא שגיאות, אולם עלול לגרום לשגיאה בזמן ריצתו (כלומר הפסקה מוקדמת של התוכנית ללא הגעה לסוף הפונקציה main)
ג. **שגיאת קומפילציה** – הקוד לא יעבור קומפילציה.

1.

```
int a;
int** b = 0;
*b = &a;
```

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

הסבר: ניסיון לכתוב לכתובת 0.

2.

```
char* s = "Hello";
s += 5;
*s = 0;
```

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

הסבר: כתיבה לתוך אזור הקבועים בשורה שלישית.

3.

```
void f(double a) {
  a /= 0;
}
int main() {
  double b=5;
  return f(b);
}
```

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

הסבר: פונקציה f מחזירה void ומנסים לעושות לו return.

4.

```
int a[];
a[0] = 3;
```

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

הסבר: הקצאת מערך ללא גודל וללא רשימת אתחול.

5.

```
char s[] = "Moed";
s[3] = 'C';
```

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

הסבר:

סעיף ב

נתון הקוד הבא:

```
void cool(int n)
{
     int k=n;
     if (n <= 1)
          return;
     while (k)
     {
          k = k/2;
     }
     cool(n/2)
}
```

מה סיבוכיות הזמן והמקום של **cool** כפונקציה של n?

סיבוכיות זמן: $\Theta(\ (\log n)^2\ )$

סיבוכיות מקום: $\Theta(\ \log n\ )$

הסבר:

$$
\begin{aligned}
T(n) &= \log(n) + T(n/2) = \\
&= \log(n) + \log(n/2) + T(n/4) = \\
&= \log(n) + \log(n)-1 + T(n/4) = \\
&= \log(n) + \log(n)-1 + \log(n)-2 + \ldots + 1 = \\
&= \Theta((\log(n)^2)
\end{aligned}
$$

שאלה 2 (25 נקודות)

סעיף א

כתבו פונקציה שבהינתן מספר שלם אי-שלילי n וספרה d בין 0 ל-9 תחזיר את מספר המופעים של הספרה d במספר n. לדוגמה, הקריאה:

```
count_digit(2881, 8);
```

תחזיר 2 כי המספר 2881 מכיל את הספרה 8 פעמיים.

על הפתרון לעמוד בסיבוכיות זמן O(logn) וסיבוכיות מקום נוסף O(1).

הערה: לצורך שאלה זו המספר 0 (אפס) אינו מכיל אף ספרה.

```
int count_digit(int n, int d)
{
  int count=0;
  while (n) {
    if (n%10 == d) count++;
    n /= 10;
  }
  return count;
}
```

סעיף ב

כתבו פונקציה שמקבלת זוג מספרים שלמים אי-שליליים n ו-k (k≤n) וכן ספרה d, ומחשבת את כל המספרים שאורכם עד n ספרות ושבהם הספרה d מופיעה בדיוק k פעמים. את המספרים יש לכתוב למערך הפלט []arr (אין חשיבות לסדר של המספרים). כמו כן על הפונקציה להחזיר את כמות המספרים שכתבה למערך.

<u>דוגמה</u>: עבור n=2, k=1, d=3 על הפונקציה לחשב את כל המספרים בעלי לכל היותר 2 ספרות ושבהם הספרה 3 מופיעה בדיוק פעם אחת. לפיכך, תוכן המערך []arr אחרי ריצת הפונקציה יהיה :

arr[] = { 3, 13, 23, 43, …, 93, 30, 31, 32 ,…, 39 }

(הסדר יכול להיות גם אחר) והפונקציה תחזיר 18.

<u>הערות</u>: בפתרונכם ניתן להיעזר בפונקציה שמימשתם בסעיף הקודם. כמו כן, ניתן להניח שהמערך []arr ארוך מספיק לאחסון כל המספרים. **דרישות סיבוכיות**: אין דרישה לפתור את השאלה בסיבוכיות זמן אופטימאלית, אולם יש לעמוד בסיבוכיות מקום O(1). את סיבוכיות הזמן של הפתרון שלכם יש להשלים במקום המיועד לכך:

סיבוכיות זמן: $\Theta(n*10^n)$

```
int fill_nums(int n, int k, int d, int arr[])
{
    int i, count=0, last=1;
    if (n==0 && k==0) {
        arr[0]=0;
        return 1;
    }
    for (i=0; i<n; i++)
        last *= 10;
    for (i=1; i<last; i++)
        if (count_digit(i,d)==k) {
            arr[count++] = i;
    return count;
}
```

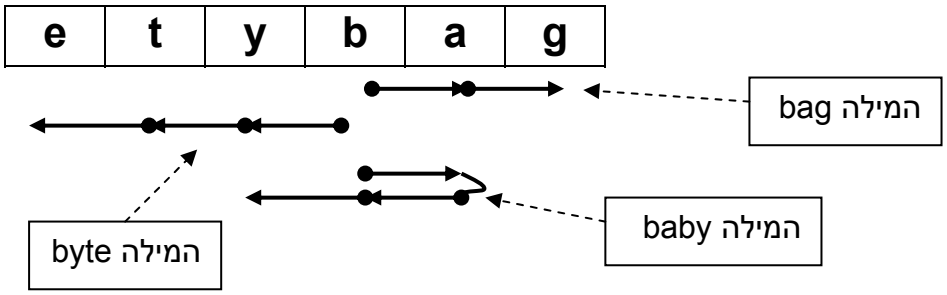
הסבר:
התנאי הראשון מטפל במקרה קצה שרוצים את כל המספרים בעלי לכל היותר 0 ספרות שמכילים איזושהי ספרה 0 פעמים. המספר "אפס" עונה על תנאי זה.

לאחר מכן מחשבים מה המספר הכי גדול בעל n ספרות ובודקים את כל המספרים עד אליו על ידי הפונקציה מסעיף א'.

שאלה 3 (25 נקודות)

שאלה זו עוסקת במשחק ה-boggle בגרסתו החד מימדית.

במשחק ה-boggle נתון לוח של אותיות קטנות באנגלית שגודלו n. על מנת להרכיב מילה, השחקן בוחר אות התחלתית בלוח, וממנה הוא ממשיך שמאלה וימינה לאותיות סמוכות עד שמתקבלת מילה חוקית באנגלית. לדוגמה, בלוח הבא ניתן להרכיב את המילים byte, bag ו-baby (ויתכן גם מילים נוספות):



שימו לב שאותה האות יכולה לשמש כמה פעמים באותה מילה, למשל האות b במילה baby למעלה.

כתבו פונקציה שבהינתן לוח משחק board ומילה לחיפוש word, מחזירה 1 אם המילה נמצאת בלוח ו-0 אחרת. הלוח מיוצג כמערך של char באורך n. שימו לב שהלוח אינו מחרוזת כיוון שהמערך מכיל n אותיות בדיוק ואינו מסתיים בתו null. המילה לחיפוש לעומת זאת מיוצגת כמחרוזת חוקית ב-C ומסתיימת בתו null.

הערות: בשאלה זו ניתן להניח **שהאותיות בלוח שונות זו מזו** (כלומר אף אות אינה מופיעה פעמיים). כמו כן, אפשר להניח שהלוח והמילה לחיפוש מכילים רק אותיות קטנות באנגלית.

דוגמאות נוספות: המחרוזות bababa ו- bababag נמצאות בלוח למעלה ואילו המחרוזות bababay ו- bagbagbag לא נמצאות בלוח.

סיבוכיות: יש לפתור את השאלה בסיבוכיות מקום נוסף O(1). אין דרישה על סיבוכיות הזמן של הפתרון, אולם יש לכתוב את סיבוכיות הזמן של המימוש שלכם במקום המתאים למטה, כפונקציה של n – גודל הלוח, ו-m – אורך המילה לחיפוש.

סיבוכיות זמן: (n+m)Θ

```
int search_word(char board[], int n, char* word) {
  int i=0;
  if (*word == '\0') return 1; // the empty word
  while (i<n && board[i] != *word) i++;
  if (i==n) return 0;
  word++; // found the first letter
  while (*word != '\0') {
    if (i>0 && *word == board[i-1]) {
      i--;
    }
    else if (i<n-1 && *word == board[i+1]) {
      i++;
    }
    else
      return 0;
    word++;
  }
  return 1;
}
```

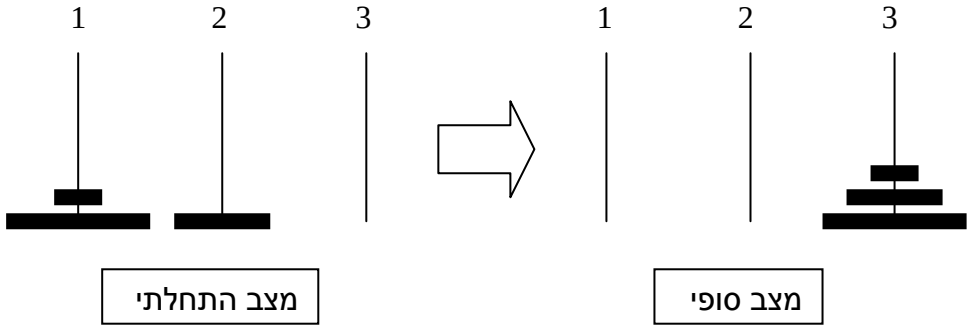
הסבר:
המילה הריקה נמצאת בכל לוח.
אם המילה לא ריקה, אז מוצאים את האות הראשונה ואז כל אות הבאה מחפשים מימין או משמאל לאות הקודמת.

שאלה 4 (25 נקודות)

שאלה זו עוסקת בבעיית מגדלי הנוי. כזכור, בבעיית מגדלי הנוי נתונות 3 עמודות של טבעות, כאשר במצב ההתחלתי הטבעות ממוקמות כולן על אחת העמודות מהגדולה (למטה) אל הקטנה. בשאלה זו נמספר את הטבעות מ-1 עד n, כאשר הקטנה מספרה 1 והגדולה מספרה n. הכלל הוא, כרגיל, שאין למקם טבעת גדולה מעל טבעת קטנה יותר, כלומר – אסור לשים טבעת i מעל לטבעת k במידה ו- k<i.

נגדיר **מגדל הנוי מלא** כמגדל של טבעות שמכיל את **כל** הטבעות מ-1 עד n (הגדולה למטה). באותו אופן, נגדיר **מגדל הנוי חלקי** כמגדל המכיל רק חלק מהטבעות הללו (הגדולה למטה).

בשאלה זו נכתוב פונקציה שמאחדת שני מגדלי הנוי חלקיים. הפונקציה מקבלת שתי עמודות, base1 ו-base2, שכל אחת מהן מכילה מגדל הנוי חלקי, וכך שבמשותף, שתי העמודות מכילות את כל הטבעות מ-1 עד n (כל טבעת מופיעה בדיוק פעם אחת). על הפונקציה להזיז את הטבעות בהתאם לכללים הרגילים, כך שבסוף התהליך נקבל מגדל הנוי **מלא** בעמודה השלישית. לדוגמה, עבור המצב ההתחלתי המופיע בשרטוט משמאל (n=3), המצב הסופי צריך להיות:



עליכם לממש את הפונקציה hanoi_unite() שמבצעת את פעולת האיחוד. את הזזת הטבעות יש לבצע באמצעות הפונקציה move שנתונה להלן:

```
void move(int from, int to);
```

פונקציה זו מזיזה את הטבעת העליונה מהעמודה from לעמודה to.

במידת הצורך ניתן להשתמש בפונקצית העזר הבאה, שפותרת את בעיית הנוי הרגילה, כלומר מעבירה מגדל מלא בגודל n מהעמודה from לעמודה to. קוד הפונקציה דומה לזה שנראה בכיתה, ונתון לכם כתזכורת:

```
void hanoi(int from, int to, int n)
{
  int via = 6-to-from;
  if (n==0)
    return;
  hanoi(from, via, n-1);
  move(from, to);
  hanoi(via, to, n-1);
}
```

הפונקציה hanoi_unite() מקבלת שישה פרמטרים: base1,base2 – האינדקסים של שתי העמודות שמכילות את המגדלים החלקיים (בין 1 ל-3). n1,n2 – כמות הטבעות בכל מגדל החלקי, בהתאמה (הערה: שימו לב שיש סה"כ בבעיה n=n1+n2 טבעות). rings1[], rings2[] – מערכים באורך n1 ו-n2, בהתאמה, שמפרטים אילו מהטבעות נמצאות בכל מגדל חלקי. כל מערך מכיל את רשימת הטבעות שנמצאות באותה עמודה, מהגדולה בתא ה-0 במערך, עד הקטנה במקום האחרון במערך.

למשל, עבור הדוגמה בעמוד הקודם, הפונקציה hanoi_unite() תיקרא כך:

```
int rings1[2] = {3,1};
int rings2[1] = {2};

int base1 = 1, base2 = 2;

hanoi_unite(base1, base2, rings1, 2, rings2, 1);
```

הערות נוספות:

- אסור לשים טבעת גדולה על טבעת קטנה בכל שלב.
- אפשר להניח שהמגדלים ההתחלתיים חוקיים.
- מותר לשנות את תוכן המערכים rings1[] ו-rings2[], ואפשר להניח שגודלם לפחות n (כלומר כל אחד מהם מסוגל להכיל את כל הטבעות)

```
void hanoi_unite(int base1, int base2, int rings1[], int n1,
                                       int rings2[], int n2)
{
  int i, j, last1, last2;
  int target = 6-base1-base2;

  if (n1==0 && n2==0)
    return;
  while (n1 != 0 && n2 != 0) {
    last1 = rings1[n1-1];
    last2 = rings2[n2-1];
    if (last1 > last2) {
      hanoi(base2, base1, last1-1);
      n2 -= last1-1;
    } else
      hanoi(base1, base2, last2-1);
      n1 -= last2-1;
    }
  }
  if (n1==0)
    hanoi(base2, target, rings2[0]);
  else // n2==0
    hanoi(base1, target, rings1[0]);
  }
```

בפתרון המוצג הפונקציה hanoi_unite() אינה רקורסיבית. אפשר היה גם לעשות פתרון רקורסיבי או פתרון שמעדכן את rings1, rings2 במהלך הריצה.

הסבר:
אם אין טבעות בכלל אז לא צריך לעשות כלום.

נרוץ בלולאה עד שכל הטבעות יתאספו על מגדל אחד:

נבדוק, על איזה מגדל הטבעת העליונה גדולה יותר. אם הטבעת הגדולה יותר היא על base1 וגודלה last1, זה אומר שהטבעות [last1-1…1] נמצאות על base2. נזיז אותן ל base1 על ידי קריאה ל hanoi().
נקטין את n2 ב-last1-1 – מספר הטבעות שהזזנו מ base2.

אם הטבעות העליונה גדולה יותר על base2 נעשה פעולות הפוכות.

בסופו של דבר כל הטבעות יהיו על מגדל אחד: או base1 או base2. אז נזיז אותן ל target על ידי hanoi().