מבוא למדעי המחשב מ'/ח' (234114 \ 234117)

סמסטר אביב תשע"ז

מבחן מסכם מועד ב', 28 לספטמבר 2017

מספר סטודנט: | | | | | | | | | | *רשום/ה לקורס:* | | | | | | 2 | 3 | 4 | 1 | 1 |

משך המבחן: 3 שעות.

חומר עזר: אין להשתמש בכל חומר עזר.

הנחיות כלליות:

- מלאו את הפרטים בראש דף זה ובדף השער המצורף, בעט בלבד.
- 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 תבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר. ניתן בהחלט להשתמש בעיפרון ומחק, פרט לדף השער אותו יש למלא בעט.
- בכל השאלות, הינכם רשאים להגדיר ולממש פונקציות עזר כרצונכם. לנוחיותכם, אין חשיבות לסדר מימוש הפונקציות בשאלה, ובפרט ניתן לממש פונקציה לאחר השימוש בה.
- אלא אם כן נאמר אחרת בשאלות, **אין להשתמש בפונקציות ספריה או בפונקציות שמומשו בכיתה**, למעט פונקציות קלט/פלט והקצאת זיכרון (`malloc, free`). ניתן להשתמש בטיפוס `bool` המוגדר ב-`stdbool.h`
- אין להשתמש במשתנים סטטיים וגלובאליים אלא אם נדרשתם לכך מפורשות.
- כשאתם נדרשים לכתוב קוד באילוצי סיבוכיות זמן/מקום נתונים, אם לא תעמדו באילוצים אלה תוכלו לקבל בחזרה מקצת הנקודות אם תחשבו נכון ותציינו את הסיבוכיות שהצלחתם להשיג.
- נוהל "לא יודע": אם תכתבו בצורה ברורה "לא יודע/ת" על שאלה (או סעיף) שבה אתם נדרשים לקודד, תקבלו 20% מהניקוד. דבר זה מומלץ אם אתם יודעים שאתם לא יודעים את התשובה.
- נוסחאות שימושיות:

$$1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n}=\Theta(\log n) \qquad 1+\frac{1}{4}+\frac{1}{9}+\frac{1}{16}+\frac{1}{25}+\ldots=\Theta(1)$$

$$1+2+\ldots+n=\Theta(n^2) \qquad 1+4+9+\ldots+n^2=\Theta(n^3) \qquad 1+8+27+\ldots+n^3=\Theta(n^4)$$

צוות הקורס 234114/7

מרצים: פרופ' מירלה בן-חן (מרצה אחראית), דר' יחיאל קמחי, גב' יעל ארז **מתרגלים:** גב' דניאל עזוז, גב' צופית פידלמן, מר תומר לנגה, מר יובל בנאי, דר' יוסי ויינשטיין, מר מוחמד טיבי, מר דמיטרי רבינוביץ', מר יאיר ריעאני, מר איתי הנדלר

בהצלחה!

שאלה 1 (25 נקודות):

א. (8 נקודות) חשבו את סיבוכיות הזמן והמקום של הפונקציה f1 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.
הניחו שסיבוכיות הזמן של malloc(n) היא $\Theta(1)$, וסיבוכיות המקום של malloc(n) היא $\Theta(n)$.

```
int f1(int n) {
    for (int j=0; j*j<n; j++)
    {
        int* p = malloc(j);
        free(p);
    }

    for (int i=1; i<n; i++)
        for (int j=0; j*i<n; j++)
            printf("0");

    return n;
}
```

סיבוכיות זמן: $\Theta(n \cdot log(n))$ סיבוכיות מקום: $\Theta(\sqrt{n})$

ב. (9 נקודות): חשבו את סיבוכיות הזמן והמקום של הפונקציה f2 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.

```
int g2(int n, int m)
{
     if(m > n)
          return 0;
     return 1 + g2(n, m * m);
}

int f2(int n)
{
     return g2(n,2);
}
```

סיבוכיות זמן: $\Theta(loglog(n))$ סיבוכיות מקום: $\Theta(loglog(n))$

ג. (8 נקודות): חשבו את סיבוכיות הזמן והמקום של הפונקציה f3 המוגדרת בקטע הקוד הבא, כפונקציה של n. אין צורך לפרט שיקוליכם. חובה לפשט את הביטוי ככל שניתן.
ניתן להניח שסיבוכיות הזמן והמקום של הפונקציה sqrt הן $\Theta(1)$.

```
int g3(int n)
{
      int sum = 0;
      for (int i = 1; i<n; i*=2)
            sum += i;
      return sum;
}

int f3(int n)
{
      int sum = 0;
      while(n > 1)
      {
            sum += g3(n);
            n = sqrt(n);
      }
      return sum;
}
```

סיבוכיות זמן: $\Theta(\log(n))$ סיבוכיות מקום: $\Theta(1)$

שאלה 2 (25 נק׳)

"אינדקס זהות" במערך arr מוגדר להיות אינדקס i עבורו מתקיים arr[i] = i.
בהינתן מערך arr ממוין בסדר עולה **ממש** (כל תא גדול ממש מהתא שמשמאלו), יש לממש את הפונקציה:

```
int identityIndex(int arr[], int n)
```

שמקבלת מערך arr ואת אורכו n, ומחזירה את האינדקס שהינו "אינדקס זהות" במידה וקיים כזה, או -1 במידה ולא קיים אינדקס זהות.
במידה וקיים יותר מאינדקס זהות אחד במערך, עליכם להחזיר אחד מהם.

דרישות:
סיבוכיות זמן $\Theta(\log n)$
סיבוכיות מקום נוסף $\Theta(1)$

אין לשנות את תוכן המערך arr.

אם לפי חישוביכם לא עמדתם בדרישות הסיבוכיות אנא ציינו כאן את הסיבוכיות שהגעתם אליה:
זמן _____ מקום נוסף _____

```
int identityIndex(int arr[], int n)
{
    int low = 0 ,high = n-1;

    while (high >= low)
    {
        int mid = (high - low)/2 + low;
        if (mid == arr[mid]) return mid;
        if (mid < arr[mid]) high = mid-1;
        else low = mid+1;
    }
    return -1;
}
```

שאלה 3 (25 נקודות) :

בהינתן מחרוזת, נגדיר "*נקודת איזון*" במחרוזת בתור אינדקס i שעבורו החלק השמאלי והחלק הימני של המחרוזת מכילים את אותם תווים (ללא חשיבות לסדר או לכמות התווים). החלק השמאלי, הוא כל האינדקסים שקטנים או שווים ל i, והחלק הימני הוא כל האינדקסים שגדולים מ i.

לדוגמא, למחרוזת: "feffef" יש שלוש נקודות איזון:

אינדקס (נקודת איזון)	*חלק ימני*	*חלק שמאלי*
i=1	ffef	fe
i=2	fef	fef
i=3	ef	feff

כיתבו פונקציה שבהינתן מחרוזת מחזירה את מספר נקודות האיזון במחרוזת:

```
int count_balance(char* str)
```

עבור המחרוזת "feffef" ערך ההחזרה יהיה 3, עבור המחרוזת "ffefeef" ערך ההחזרה יהיה 3 (נקודות האיזון הן אינדקסים 2,3,4), למחרוזת "abcde" אין נקודות איזון כלל ולכן ערך ההחזרה עבורה יהיה אפס. ערך ההחזרה עבור מחרוזת ריקה יהיה אפס.

ניתן להניח שכל התווים במחרוזת הם אותיות קטנות באנגלית.

דרישות:
סיבוכיות זמן $\Theta(n)$ כאשר n מסמל את אורך המחרוזת (אינו נתון ויש לחשבו במידת הצורך).
סיבוכיות מקום נוסף $\Theta(1)$.

אין לשנות את תוכן המחרוזת str.

אם לפי חישוביכם לא עמדתם בדרישות הסיבוכיות אנא ציינו כאן את הסיבוכיות שהגעתם אליה:
זמן _____ מקום נוסף _____

```
#define LETTERS 'z'-'a'+1

unsigned int strlen(const char *s) {
     unsigned int i;
    for (i = 0; s[i] != '\0'; i++) ;
    return i;
}
```

```
int check_if_balanced(int hist1[], int hist2[]) {
     int i;
     for (i=0 ; i < LETTERS ; i++) {
          if ((hist1[i] > 0 && hist2[i] == 0) ||
               (hist1[i] == 0 && hist2[i] > 0)) {
               return false;
          }
     }
     return true;
}

int count_balance(char* str) {
     int left_hist[LETTERS] = {0};
     int right_hist[LETTERS] = {0} ;
     int i, count = 0, n = strlen(str);
     for (i = 0 ; i < n ; i++)
         right_hist[str[i] - 'a']++;

     for (i = 0 ; i < n ; i++) {
         left_hist[str[i] - 'a']++;
         right_hist[str[i] - 'a']--;

         count +=check_if_balanced(left_hist,right_hist);
     }
     return count;
}
```

שאלה 4 (25 נקודות) :

בעקבות הצפה בגן החיות צריך לפנות את N החיות המתגוררות בגן בעזרת סירות הצלה. לסירות ההצלה יש מגבלת משקל, כך שלא ניתן להעלות לסירה אחת חיות שסכום משקלן עולה על M.
בנוסף, חיות מסוימות לא יכולות לשוט יחד באותה הסירה.

ממשו את הפונקציה:

```
int assign_boats(int A[][N], int weights[])
```

הפונקציה assign_boats מקבלת:

- מערך weights באורך N המכיל את משקלן של החיות.
- מערך דו-מימדי A בגודל NxN, שערכו 1 במקום ה i,j אם החיות i ו j יכולות לשוט ביחד, ו 0 אחרת.

על הפונקציה להחזיר את מספר הסירות המינימלי הדרוש כדי להציל את החיות. ניתן להניח שאפשר להציל את כל החיות, כלומר משקל כל חיה לא יהיה גדול מ – M.

לדוגמא, אם M = 10 ותוכן המערכים הוא:

weights = {3, 4, 10}; A =

1	0	1
0	1	1
1	1	1

מספר הסירות המינימלי הדרוש כדי להציל את החיות הוא 3, מכיוון שהחיה עם משקל 10 לא יכולה לשוט עם חיות אחרות כיוון שזהו המשקל המקסימלי, ושתי החיות הראשונות לא יכולות לשוט יחד.
לכן, הפונקציה צריכה להחזיר את הערך 3.

הערות:

- יש להשתמש בשיטת backtracking כפי שנלמדה בכיתה.
- ניתן להניח שהמטריצה weights סימטרית, כלומר A[i][j]==A[j][i].
- ניתן להניח ש –A[i][i] = 1, כלומר כל חיה יכולה לשוט עם עצמה.
- ניתן להניח ש – M, N הם קבועים אשר מוגדרים בעזרת define.
- אין לשנות את תוכן המערכים A ו – weights.
- בשאלה זו אין דרישות סיבוכיות, אולם כמקובל ב-backtracking יש לוודא שלא מתבצעות קריאות רקורסיביות מיותרות עם פתרונות שאינם חוקיים.
- ניתן ומומלץ להשתמש בפונק' עזר (ויש לממש את כולן).

```
int min(int a, int b)
{
    return a<b ? a : b;
}

bool is_legal_boat(int A[][N], int weights[],
              int assignment[], int animal, int boat)
{
    int weights_on_boat = 0;
    for (int i=0; i < animal; i++)
    {
        if (assignment[i] != boat)
            continue;
        if (!A[animal][i])
            return false;
        weights_on_boat += weights[i];
    }
    return (weights_on_boat + weights[animal]) <= M;
}
```

```
int assign_boats_aux(int A[][N], int weights[],
             int assignment[], int animal, int last_boat)
{
    if (animal == N)
        return last_boat;
    int res = N;
    for (int i=1; i<=last_boat; i++) {
        if (is_legal_boat(A, weights, assignment, animal,
                          i))
        {
            assignment[animal] = i;
            int curr_res = assign_boats_aux(A, weights,
assignment, animal+1, last_boat);
            res = min(res, curr_res);
        }
    }
    int new_boat = last_boat + 1;
    assignment[animal] = new_boat;
    int new_boat_res = assign_boats_aux(A, weights,
                        assignment, animal+1, new_boat);
    return min(res, new_boat_res);
}

int assign_boats(int A[][N], int weights[])
{
    int assignment[N] = {0};
    return assign_boats_aux(A, weights, assignment, 0,
                            0);
}
```