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

מועד ב' תשפ"ב

מבחן מסכם , 15.03.2022

מספר סטודנט: | | | | | | | | | | *רשום/ה לקורס:* | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| | | | | | | |

2	3	4	1	1	

משך המבחן: 3 שעות.

חומר עזר: אין להשתמש בכל חומר עזר.

הנחיות כלליות לכולם:

- בדקו שיש 22 עמודים (4 שאלות) במבחן, כולל עמוד זה.
- בכל השאלות, הנכם 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 וגלובליים אלא אם נדרשתם לכך מפורשות.
- אפשר להניח שקורא התוכנית ידע לפענח את רעיונותיכם אך ורק מקריאת התוכנית אך בנוסף התוכנית יכולה להכיל תיעוד קל להבנה.
- 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:

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

$$1^k+2^k+3^k+\ldots+n^k=\Theta(n^{k+1}) \qquad 1+k+k^2+k^3+\ldots+k^n=\Theta(k^n),\ k>1$$

הנחיות כלליות לנבחנים במתכונת הרגילה:

- מלאו את הפרטים בראש דף זה ובדף השער המצורף, בעט בלבד.
- כתבו את התשובות על טופס המבחן בלבד, במקומות המיועדים לכך. שימו לב שהמקום המיועד לתשובה אינו מעיד בהכרח על אורך התשובה הנכונה.
- אנא סמנו טיוטות באופן ברור על מנת שהן לא תיבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר, **ובעט בלבד**.

צוות הקורס 234114/7

מרצים: פרופ' תומר שלומי (מרצה אחראי), גב' יעל ארז, ד"ר יוסי ויינשטיין, מר' איהאב ותד;
מתרגלים: קטרין חדאד (מתרגלת אראית), מג'ד ח'ורי (מתרגל אחראי), רואי בנימין, מתן ממיסטבלוב, דמיטרי רבינוביץ', אור-אל אדיבי, מרוה מועלם, הראל וקנין, שקד ניסנוב, אדיר רחמים, יואב-מתן פרץ, דניאל ליברמן, צביקה לזר, סתיו רות.

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 את שיקוליכם. חובה לפשט את הביטוי ככל שניתן.

```
void f1(int n)
{
    int x = 0;
    for (int i =1; i < n; i++)
        for (int j =1; j < n; j*=2)
            x++;
}
```

סיבוכיות זמן: $\theta(nlog\ n)$ סיבוכיות מקום: $\theta(1)$

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

```
void f2(int n) {

       for(int i=1; aux2(2,i)<n; i++){
              for(int j=0; j< n*n; j+=i)
                     printf("YOU GOT THIS!\n");
       }
}

int aux2(int x, int n) {
       int res = 1;
       for (int i = 0; i < n; ++i)
              res = res * x;
       return res;
}
```

סיבוכיות זמן: $\theta(n^2 \log\log n)$ סיבוכיות מקום: $\theta(1)$

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. יש להניח כי הפונקציה sqrt(n) מחזירה את השורש הריבועי של הפרמטר שהיא מקבלת וכי סיבוכיות הזמן והמקום שלה היא: $\theta(1)$.

```
int f3(int n)
{
    if (n <= 2)
        return 1;
    f3(2*f3(sqrt(n)));
    return n/2;
}
```

סיבוכיות זמן: $\theta(\log n)$ סיבוכיות מקום: $\theta(\log\log n)$

שאלה 2 (25 נק׳)

יהי *arr* מערך של מספרים שלמים אי שליליים. **מערך סכום מצטבר** של המערך *arr* הוא מערך שבו האיבר באינדקס ה- i שווה לסכום כל האיברים במערך *arr* מאינדקס 0 עד האינדקס i (כולל).

דוגמא: עבור המערך הבא:

1	0	4	2	2	5

מערך הסכום המצבטר שלו:

1	1	5	7	9	14

עליכם לממש את הפונקציה הבאה:

```
void findSubArray(int* cumsum, int n, int sum, int indices[N]);
```

הפונקציה מקבלת מערך סכום מצטבר של *arr*, את גודלו *n*, סכום חיובי *sum* ומערך *indices* שבו 2 איברים. הפונקציה מחפשת רצף של איברים סמוכים במערך *arr* אשר סכומם הוא *sum*. אם קיים רצף כזה, הפונקציה תרשום באיבר הראשון ב *indices* את האינדקס השמאלי של הרצף ובאיבר השני את האינדקס הימני של הרצף. אם לא קיים רצף כזה, הפונקציה תרשום 1- ב-2 איברי המערך *indices*. אם קיים יותר מרצף אחד, ניתן להחזיר אחד מהם.

דוגמא: עבור מערך הסכום המצטבר לעיל וסכום רצוי 6, הפונקציה יכולה להחזיר האינדקסים 1 ו- 3 שכן האיברים באינדקסים 1 עד 3 במערך *arr* סכומם שווה ל- 6. הפונקציה גם יכולה להחזיר האינדקסים 2 ו- 3.

הערות ודגשים:

- שימו לב שגודל מערך הסכום המצטבר שווה לגודל המערך המקורי.
- המערך arr **<u>איננו</u>** נתון לפונקציה.
- ניתן להניח כי הקבוע N=2 מוגדר בעזרת #define.
- בפתרון שלכם, אסור לשנות את התוכן של המערך **cumsum** אפילו לא באופן רגעי.

דרישות סיבוכיות:

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

אם לדעתכם לא עמדתם בדרישות הסיבוכיות, אנא ציינו את הסיבוכיות שהגעתם אליה.

המשך השאלה בעמוד הבא...

```
#define N 2
void findSubArray(int cumsum[], int n, int sum, int indices[N]){
	int r = 0, l = 0, sum_from_left = 0;
	for(int i = 0; i < N; i++){
		indices[i] = -1;
	}
	while(r < n){
		if(cumsum[r] - sum_from_left > sum){
			sum_from_left = cumsum[l];
			l++;
		}
		else if(cumsum[r] - sum_from_left < sum) r++;
		else {
			indices[0] = l;
			indices[N-1] = r;
			return;
		}
	}
}
```

שאלה 3 (25 נקודות) :

הגדרה: מחרוזת *str* נקראת ***k* - בטוחה** אם היא מחרוזת שבה כל 2 אותיות זהות שומרות על מרחק של לפחות *k* אותיות זו מזו (כלומר, מפרידות ביניהן לפחות *k* אותיות). מחרוזת באורך *n* המורכבת מאותיות שונות זו מזו הינה *(n-1)*-בטוחה.

לדוגמא:

- המחרוזת *"abcabdca"* היא 2-בטוחה.
- המחרוזת *"abaebdca"* היא 1-בטוחה (שני המופעים של האות *a* נמצאים במקומות 0 ו-2 ולכן הם מופרדים ע"י אות אחת בלבד).
- המחרוזת *"aabcdfre"* היא 0-בטוחה כיוון ששני המופעים של האות *a* אינם מופרדים ע"י אף אות אחרת.

ממשו את הפונקציה :

int get_safety(char* str);

המקבלת מחרוזת *str* ומחזירה את *k* המקסימלי עבורו המחרוזת היא *k*-בטוחה. ניתן להניח כי המחרוזת המתקבלת מורכבת מאותיות אנגליות קטנות בלבד.

- **דרישות:**
  - סיבוכיות זמן: $\theta(n)$ כאשר n הוא אורך המחרוזת,
  - סיבוכיות מקום נוסף: $\theta(1)$
- אם לפי חישוביכם לא עמדתם בדרישות הסיבוכיות אנא ציינו כאן את הסיבוכיות שהגעתם אליה.

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

int get_safety(char* str){

int places[LETTERS];
int len = strlen(str);
// strlen is the function that we learned in
// class and you are supposed to implement it in the exam if you used it!!

int max_k = len-1;
    for(int i = 0; i < LETTERS; i++)
        places[i] = -1;

    for(int i = 0; i < len; i++){
        if(places[str[i]-'a'] != -1 && i - places[str[i]-'a'] <= max_k)
                max_k = i - places[str[i]-'a'] -1;
        places[str[i]-'a'] = i; //update latest position
       }
return max_k;
}
```

שאלה 4 (25 נקודות) :

נתונים N שחקנים שמספריהם $0, \dots, N-1$, וברצוננו לשבץ אותם לשתי קבוצות זרות של שחקנים *P* ו- *Q*. נתונה לכם מטריצה בינרית *A* בגודל *NxN* (מטריצה בינרית היא מטריצה שערך כל כניסה בה הוא או 0 או 1). המטריצה *A* מייצגת אילוצים על השיבוצים החוקיים כך שאם $A[i][j] = 1$ אז שני השחקנים *i* ו- *j* **יכולים לשחק באותה קבוצה**, אחרת, כלומר אם $A[i][j] = 0$ אזי השחקנים *i* ו- *j* **לא יכולים** לשחק באותה קבוצה. שימו לב שמתקיים כי $A[i][j] = A[j][i]$ לכל זוג שחקנים, וכי לכל *i* מתקיים $A[i][i] = 1$ לכל *i*.

שיבוץ חוקי של שחקנים למשחק הינו חלוקה של קבוצת השחקנים לשתי קבוצות זרות *P* ו- *Q* כך שכל אחת מהקבוצות עונה על האילוצים של המטריצה *A* . **כל** השחקנים חייבים להיות משובצים, וכל שחקן משובץ לקבוצה אחת בלבד. **שימו לב:** בניגוד למשחקים במציאות, בשאלה זו יתכנו שתי קבוצות עם גדלים שונים.

עליכם לכתוב את הפונקציה הבאה:

```
int find_num_partitions(int A[N][N]);
```

המקבלת מערך דו ממדי *A* המיצג את מטריצת האילוצים, ומחזירה את מספר השיבוצים החוקיים.

לדוגמה, עבור המטריצה *A* הבאה:

	0	1	2	3	4
0	1	1	0	0	1
1	1	1	1	1	1
2	0	1	1	1	0
3	0	1	1	1	0
4	1	1	0	0	1

הפונקציה תחזיר 2 עבור שני השיבוצים האפשריים הבאים:

- *P={0,1,4}, Q={2,3}*
- *P={0,4}, Q={1,2,3}*

שימו לב: השיבוץ *P={1,2,3}, Q={0,4}* זהה לשיבוץ ההפוך *P={0,4}, Q={1,2,3}* ולכן אינו נספר שוב כשיבוץ חוקי.

המשך השאלה בעמוד הבא...

הערות:

- יש להשתמש בשיטת backtracking כפי שנלמדה בכיתה.
- בשאלה זו אין דרישת סיבוכיות, אולם כמקובל ב- backtracking, יש לוודא שלא מתבצעות קריאות רקורסיביות מיותרות עם פתרונות שאינם חוקיים.
- ניתן ומומלץ להשתמש בפונקציות עזר (ויש לממש את כולן).
- הקבוע N מוגדר ב- #define.

```
#define N 5
#define TEAMS 2
#define CLEAN -1

int find_num_partition(int A[N][N])
{
    if (N==0) return 0;
    int teams[TEAMS][N], sizes[TEAMS]={0};
    for (int i =0; i < TEAMS; i++)
       {
           for (int j=0; j<N; j++)
            teams[i][j] = CLEAN;
       }
        teams[0][0] = 0;
        sizes [0] = 1;
        return find_num_partition_aux(A, teams, sizes, 1);

}

int find_num_partition_aux(int A[N][N], int teams[TEAMS][N], int
sizes[TEAMS],int  p){
    if (p == N)
        return 1;
    int part_count = 0;
    for (int i = 0; i < TEAMS; i++)
    {
        if (check_assignment(A, teams[i],  sizes[i],  p))
        {

        teams[i][sizes[i]]=p;
        sizes[i]++;
        part_count+= find_num_partition_aux(A, teams,sizes,  p+1);
          teams[i][sizes[i]] = CLEAN;
          sizes[i]--;
          }

     }
     return part_count;
}
bool check_assignment( int A[N][N],int team[N], int size, int p)
{
    if (size == 0)
        return true;
    if (p < team[size-1]) return false;
    for (int i =0; i < size; i++)
    {
```

```
        if(A[team[i]][p] == 0)
            return false;
    }
    return true;

}
```