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

סמסטר אביב תש"פ

מבחן מסכם מועד ב', 01 באוקטובר 2020

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 תיבדקנה.
- יש לכתוב באופן ברור, נקי ומסודר, **ובעט בלבד**.
- 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 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:

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

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

צוות הקורס 234114/7

מרצים: פרופ' ניר אילון(מרצה אחראי), גב' יעל ארז **מתרגלים:** עמית ברכה, דן ברקוביץ', גיא ברשצקי, עמר דהרי, קטרין חדאד, דמיטרי רבינוביץ' (מתרגל אחראי)

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.

```
int f1(int n)
{
    if (n <= 3) return 1;
    return f1(f1(n / 3) + f1(n / 3) + f1(n / 3)));
}
```

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

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. הניחו שסיבוכיות הזמן של malloc(n) ו-free היא $\Theta(1)$, וסיבוכיות המקום של malloc(n) היא $\Theta(n)$.

```
void f2(int n)
{
    int* temp;
    int k = 0;
    for (int i = 0; i < n; i += k)
    {
        ++k;
        temp = malloc(k * k);
        free(temp);
    }
}
```

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

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.

```
void f3(int n)
{
      int x = 0;
      for (int i = 1; i <= n / 2; ++i)
            for (int j = i; j <= n * n; j += i)
                  x++;
}
```

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

שאלה 2 (25 נק')

נתון מערך **בינארי** a שמכיל **רק** אפסים (0) ואחדים (1) ומספר שלם חיובי k. מצאו את האורך של תת סדרה רציפה ארוכה ביותר של a שכוללת לכל היותר k אפסים.

ממשו את הפונקציה LongestSequence שמוצאת את האורך של סדרה כזאת:

```
int LongestSequence (int a[], int n, int k)
```

לדוגמה: אם נתון המערך

```
{ 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0 }
```

עבור k = 0,

הפונקציה תחזיר 4 (הסדרה באורך 4 המתחילה ממקום 6 היא ארוכה ביותר עם k=0 אפסים)

עבור k = 1,

הפונקציה תחזיר 7 (הסדרה באורך 7 המתחילה מאינדקס 3 עם k=1 אפסים)

עבור k = 2,

הפונקציה תחזיר 10 (הסדרה באורך 10 המתחילה מאינדקס 0, מכילה בדיוק k=2 אפסים)

עבור k = 3,

הפונקציה תחזיר 11 (הסדרה באורך 11 המתחילה מאינדקס 0 מכילה מספר אפסים מכסימלי האפשרי 3)

דרישות:

- ממשו את הפונקציה בסיבוכיות זמן $O(n)$. אין הגבלה על סיבוכיות המקום.

אם לא עמדתם בדרישות הסיבוכיות, אנא ציינו כאן את הסיבוכיות שהגעתם אליה:

זמן _____ מקום נוסף _____

int LongestSequence (int a[], int n, int k)
{

```
int LongestSequence(int a[], int n, int k)
{
    int longest = 0, front = 0,
        back = 0, seen_zeros = 0;

    while (front < n)
    {
        if (a[front] == 0)
            if (++seen_zeros > k)
            {
                while (a[back] != 0) ++back;
                ++back;
                --seen_zeros;
            }

        if (front - back + 1 > longest)
            longest = front - back + 1;
        ++front;
    }

    return longest;
}
```

שאלה 3 (25 נקודות) :

נגדיר *גיוון פונטי* של המחרוזת `str` כקבוצה של התווים שמופיעים בה (ללא חזרות).

למשל, עבור "`collection`" = `str` הגיוון הפונטי הוא
{'c', 'e', 'i', 'l', 'n', 'o', 't'}.

ממשו את הפונקציה:

```
int ShortestSubstring(char *str)
```

הפונקציה תקבל מחרוזת `str` ותחזיר את האורך של תת-מחרוזת קצרה ביותר בעלת גיוון פונטי זהה לזה של `str`.

דוגמא:

- עבור המחרוזת "`aabcaadefacebaa`" = `str[]`

הגיוון הפונטי של הקלט str הוא {'a', 'b', 'c', 'd', 'e', 'f'}. הפונקציה תחזיר 7, כי זהו האורך של "`defaceb`" (תת המחרוזת המודגשת באפור) והגיוון הפונטי של תת-מחרוזת זו זהה לגיוון הפונטי של `str`.

הערות:

- ניתן להניח כי המילים מורכבות מאותיות אנגליות קטנות בלבד.
- במקרה שלא קיימת תת-מחרוזת ממש בעלת אותו גיוון פונטי, הפונקציה תחזיר את האורך של המחרוזת `str`.

דרישות:

- סיבוכיות זמן $O(n)$, כאשר `n` הינו אורך של המחרוזת `str`, סיבוכיות מקום $O(1)$.

אם לפי חישוביכם לא עמדתם בדרישות הסיבוכיות אנא ציינו כאן את הסיבוכיות שהגעתם אליה:

זמן _____ מקום נוסף _____

`int ShortestSubstring(char *str)`
`{`

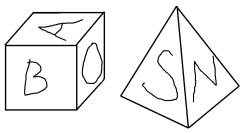
```
int ShortestSubstring(char* str)
{
    char* front = str, * back = str;
    int hist_full[ABC] = { 0 }, hist_substr[ABC] = { 0 };

    while (*str)
        ++hist_full[*str++ - 'a'];

    int shortest = str - front;
   while (*front)
   {
        ++hist_substr[*front++ - 'a'];
        while (Compare(hist_full, hist_substr))
        {
            if (front - back < shortest)
                shortest = front - back;
            --hist_substr[*back++ - 'a'];
        }
    }
    return shortest;
}
```

שאלה 4 (25 נקודות) :

נתונות n קוביות של אותיות. על כל פאה של כל קובייה מצוירת אות אנגלית **אחת**, אך בכל קובייה **יתכן** מספר **שונה** של פאות. בנוסף, לכל קוביה יש צבע, כאשר צבע הוא מספר בין 0 ל n-1 (כולל). שימו לב קוביות שונות יכולות להיות בעלות צבעים זהים, ויכולים להיות צבעים שאינם בשימוש.

כתבו פונקציה שמקבלת כקלט תיאור של n הקוביות, וכן מילת מטרה (באנגלית), ופולטת true אם ורק אם ניתן להעמיד את הקוביות (או את חלקן) זו לצד זו, כך שרצף האותיות המופיעות בפאות העליונות, לפי הסדר, זהה למילת המטרה, וכך שכל הקוביות בצבעים שונים. אחרת על הפונקציה להחזיר false.

הערה: לצורך השאלה, גם פירמידה (כבציור) היא סוג של "קוביה". יש להניח קיום של "קוביה" עם כל מספר טבעי של פאות (1, 2, 3, ...) , ויש להניח שעבור כל פאה של כל קוביה, ניתן מבחינה פיזית להעמיד את הקוביה כך שהפאה היא "עליונה" (אפילו במקרה של הפירמידה...).



להלן דוגמא של קלט (הערה: <u>הדוגמא לא קשורה לציור</u>). נשתמש ב-n=4 קוביות:

לקובייה הראשונה 3 פאות, ובהן האותיות **A, B, C**. צבע קוביה זו הוא 0.

לקובייה השנייה 3 פאות, ובהן האותיות **A, E, I**. צבע קוביה זו הוא 0 גם כן.

לקובייה השלישית 3 פאות, ובהן האותיות **E, O, U**. צבע קוביה זו הוא 1.

לקובייה הרביעית 5 פאות, ובהן האותיות **L, N, R, S, T**. צבע קוביה זו הוא 2.

למשל, עבור מילת מטרה "BONE"=target יש להחזיר false, מכיוון שיש לנו רק 4 קוביות, ושתיים מתוכן בעלות צבע זהה. (כך למעשה גם עבור כל מילה באורך לפחות 4).

גם עבור מילת המטרה "CAT"=target יש להחזיר false מאחר שהאות C מיוצגת רק בקוביה הראשונה, וזה מכריח אותנו לקחת את A מהקוביה השניה, ואנו מפרים את אילוץ הצבעים.

עבור מילת המטרה "BOT"=target יש להחזיר true, מאחר שניתן להשתמש בקוביה הראשונה בשביל B בשלישית בשביל O וברביעית בשביל T, תוך שימוש יחיד בכל אחד מהצבעים 0,1,2.

עבור המילה "SET"=target יש להחזיר false, מכיוון שכל אחת מהאותיות S,T מופיעה רק על פאה של קוביה מספר 4, ואין להשתמש באף קוביה פעמיים.

עבור המילה "TO" יש להחזיר `true`, מכיוון שניתן להעמיד את קוביה 4 בשביל `T` ואת קוביה 3 בשביל `O`, והצבעים שלהם שונים.

עבור המילה הריקה `target=""` יש להחזיר true מכיוון שאף אילוץ לא מופר אם פשוט לא נעמיד אף קוביה.

הפונקציה, בשם `scramble`, תקבל את ייצוג הקוביות כמערך של מחרוזות (כל קוביה כמחרוזת האותיות המופיעות על פאותיה, בסדר שרירותי), את מספר הקוביות `n`, את צבעי הקוביות כמערך של `int` באורך `n`, ואת מילת המטרה `target`.

הערות:

- יש להשתמש בשיטת `backtracking` כפי שנלמדה בכיתה.
- בשאלה זו אין דרישות 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).
- אין צורך לבדוק את תקינות הקלט.

```
bool Scramble(char *cubes[], int n, int colors[], char* target)
{
    int* pickedCubes = malloc(sizeof(int) * strlen(target));
    bool possible = Scramble_aux(cubes, n, colors, target, pickedCubes, 0);
    free(pickedCubes);
    return possible;
}
```

```
bool Scramble_aux(char* cubes[], int n, int colors[], char* target,
    int pickedCubes[], int letter_index)
{
    int k = strlen(target);

    if (letter_index == k)
        return UsedOnce(pickedCubes, k) &&
            UniqueColors(colors, pickedCubes, k);

    for(int letter = 0; letter < n; ++letter)
        if (LetterIsPresent(cubes[letter], target[letter_index]))
        {
            pickedCubes[letter_index] = letter;
            if (Scramble_aux(cubes, n, colors, target, pickedCubes,
                letter_index + 1))
                return true;
        }

    return false;
}

bool UsedOnce(int pickedCubes[], int k)
{
    for (int first = 0; first < k; ++first)
        for (int second = first + 1; second < k; ++second)
            if (pickedCubes[first] == pickedCubes[second])
                return false;
    return true;
}
```

```
bool LetterIsPresent(char* word, char letter)
{
    while (*word)
        if (*word++ == letter)
            return true;
    return false;
}


bool UniqueColors(int colors[], int pickedCubes[], int k)
{
    for (int first = 0; first < k; ++first)
        for (int second = first + 1; second < k; ++second)
            if (colors[pickedCubes[first]] == colors[pickedCubes[second]])
                return false;
    return true;
}
```