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

סמסטר אביב תשס"ז

מבחן מסכם מועד ב', 22 אוקטובר 2007

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 30 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: סאהר אסמיר, ד"ר אלי בן ששון (מרצה אחראי).
מתרגלים: מירי בן-חן, אלעד הרמתי, עידו חניאל, אסנת טל, סשה סקולוזוב, אנדרי קלינגר, איתי שרון (מתרגל אחראי).

בהצלחה!

שאלה 1

סעיף א - 10 נק', 4 ל-1 ו-3 לכל חישוב סיבוכיות

נתונה תוכנית ה-C הבאה:

```
#include <stdio.h>

int bar(int m, int d)
{
    if(d <= 1) return 1;
    if(!(m%d)) return 0;
    return(bar(m, d-1));
}

void foo(int n)
{
    if(!n)
        return;
    if(bar(n, n-1)) printf("%d\n", n);
    foo(n-1);
}

int main ()
{
   int n;
   scanf("%d", &n);
   foo(n);

   return 0;
}
```

1. מה מבצעת התכנית ומה יהיה הפלט עבור הקלט 10?
2. מהי סיבוכיות הזמן ומהי סיבוכיות המקום של התוכנית כפונקציה של **n**?

התוכנית מדפיסה את המספרים הראשוניים הקטנים מ או שווים ל-n.
עבור הקלט 10 תדפיס התוכנית 7, 5, 3, 2, 1

סיבוכיות זמן: התוכנית קוראת ל-bar() על כל המספרים מ-n ועד ל-1, הסיבוכיות של bar() היא d כי יש d קריאות רקורסיביות לפונקציה. מכיוון ש-foo() מתחילה תמיד עם d=n-1, מתקבל:

$Time(foo()) = (n-1)+(n-2)+\ldots+3+2+1 = O(n^2)$

מבחינת סיבוכיות מקום - עומק הרקורסיה המקסימלי הוא n (קריאה ל-foo() ועוד n-1 קריאות ל-bar()), ולכן

$Space(foo()) = O(n)$

סעיף ב - 15 נק', 3 לכל שאלה

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

א. **ללא שגיאות** – הקוד יתקמפל ללא כל שגיאה וירוץ 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.
```
char *p = "Oh, well";
p[2] = '!';
```

p מצביע לאזור בזכרון שיכול להיות read only, ולכן כתיבה אליו תגרום לשגיאת זמן ריצה.

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

2.
```
void foo(char bar[10]) {
  int i;
  for(i=0; i<10; i++)
    putchar(*(bar++));
}
```

אין בעיה – bar הוא מצביע, ולכן אפשר לשנות אותו.

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

3.
```
char *p = "%dx, %c%c";
printf(p, 10, 'c', 'u');
```

אין בעיה – printf מצפה לקבל מצביע מטיפוס *char וזה מה שאנחנו נותנים לו

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

4.
```
char s[5]={'H','e','l','l','o'};
int d=5.9999;

s[d] = 'O';
```

d יאותחל ל-5, אבל האינדקס 5 אינו חוקי במערך s שבו יש 5 איברים (אינדקסים 0 .. 4)

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

5.
```
int a=1, *b=&a, **c=&b;

a = (c==&&a)? 2 : **c;
```

האופרטור & נותן כתובת של משתנה, אבל &&a אינו כתיבה חוקית (אין כזה דבר "כתובת של כתובת", כתובת חייבת להיות של משתנה!)
למעשה, כמו שאתם יודעים, && הוא אופרטור ב-C (AND לוגי) והשימוש בו כאן אינו נכון.

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

שאלה 2

בבית ההשקעות "כפית הכסף" הצליחו לחזות במדיוק את מחיר מניית "דג הזהב" ב-n הימים הקרובים. הניחו כי המחירים העתידיים שמורים במערך **quotes** כאשר הכניסה הראשונה במערך מכילה את המחיר הנוכחי (היום הראשון). נרצה לכתוב פונקציה המחשבת את הרווח המקסימלי שניתן להשיג ע"י קניית המניה ומכירתה. שימו לב שיש לחשב את הרווח באחוזים. נוסחה כללית לחישוב הרווח נתונה ע"י: $revenue = \frac{sell\_price - buy\_price}{buy\_price}$

דוגמה: עבור סדרת המחירים הבאה

120	90	80	110	100

- קניית המניה ביום הראשון (מחיר 100) ומכירתה ביום האחרון (120) תניב רווח של $\frac{120-100}{100} = 0.2 = 20\%$
- קניית המניה ביום השלישי (מחיר 80) ומכירתה ביום האחרון (120) תניב רווח של $\frac{120-80}{80} = 0.5 = 50\%$
- קניית המניה ביום הראשון (מחיר 100) ומכירתה ביום השלישי (80) תניב הפסד של $\frac{80-100}{100} = -0.2 = -20\%$
- שילוב פעולת קנייה ביום הראשון (מחיר 100) ומכירה ביום השני (110) ואז קנייה מחדש ביום השלישי (מחיר 80) ומכירתה ביום האחרון (120) תניב רווח כולל של $\left(1+\frac{(110-100)}{100}\right)\cdot\left(1+\frac{(120-80)}{80}\right)-1 = 1.1*1.5-1 = 0.65 = 65\%$. במלים אחרות אילו היינו משקיעים 100 דולר במניות "דג הזהב" ומוכרים אותן אחרי יום, ולאחר מכן משקיעים את תמורתן במניות ביום השלישי ומוכרים אותן ביום החמישי הסכום אותו היינו מקבלים הוא 165 דולר, ולכן הרווח הוא 65%.

ניתן להניח את ההנחות הבאות:

- המחיר אינו משתנה במהלך היום
- תמיד ישנן מניות למכירה וקונים במחירים הנתונים
- קניה ומכירה אינה עולה כסף (עמלות, מיסים וכיו"ב)
- לא ניתן לקנות מניות לפני היום הראשון, לא ניתן למכור מניות אחרי היום האחרון
- ניתן לקנות חלקי מניות ללא הגבלה
- לחברת "כפית הכסף" יש סכום התחלתי להשקעה x, בכל פעולת קנייה נעשה שימוש בכל הכסף הזמין ובכל פעולת מכירה נמכרות כל המניות שניקנו

סעיף א – 2 נקודות

בהינתן מערך כנ"ל ואורכו, כתבו פונקציה יעילה ככל האפשר המחזירה את הרווח במידה וקונים את המניה ביום הראשון ומוכרים אותה ביום האחרון. למשל, עבור המערך לעיל הפונקציה תחזיר 0.2 (20%).

```
double buy_first_sell_last(double quotes[], int n)
{
  return (quotes[n-1]-quotes[0])/quotes[0];
}
```

סעיף ב – 9 נקודות

בהינתן מערך כנ"ל ואורכו, כתבו פונקציה יעילה ככל האפשר המחזירה את הרווח המירבי אותו ניתן להשיג ללא הגבלה של פעולות קניה ומכירה (כלומר מותר לקנות ולמכור מספר פעמים). למשל, עבור המערך לעיל הפונקציה תחזיר 0.65 (65%).
רמז: בהנחה שאתם מחזיקים במנייה כלשהי וידוע לכם שמכירה עומד לרדת, ובהנחה שעלות פעולות מכירה וקנייה היא 0 (כפי שנתון), מהו הדבר הנכון לעשות? מהו הדבר הנכון לעשות במידה וידוע לכם שמכיר המנייה עומד לעלות?

```
double buy_sell_nolimit(double quotes[], int n)
{
  double sum = 1;
  double last buy = 0;
  int i = 0;

  for(i = 0; i < n-1; i++) {
    if((quotes[i+1] < quotes[i]) && (last buy > 0)) {
      sum *= quotes[i]/last buy;
      last buy = 0;
    }
    else if((quotes[i+1] > quotes[i]) && (last buy == 0)) {
      last but = quotes[i];
    }
  }

  if(last buy)
    sum *= quotes[n-1]/last buy;

  return (sum - 1);
}
```

סעיף ג – 14 נקודות

בהינתן מערך כנ"ל ואורכו, כתבו פונקציה יעילה ככל האפשר המחזירה את הרווח המירבי אותה ניתן להשיג ע"י פעולת קניה אחת בלבד ופעולת מכירת אחת בלבד. למשל, עבור המערך לעיל הפונקציה תחזיר 0.5 (50%).
רמז: מרגע שקניתם מנייה, מתי הכי כדאי למכור אותה?

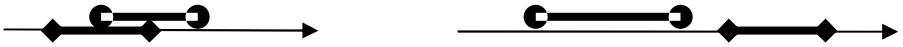
```
double buy_sell_once(double quotes[], int n)
{
  double max revenue = 0;
  double *buy, *sell;

  for(buy=sell=quotes; sell<(quotes+n); sell++) {
  {
    if(*buy > *sell)
      buy = sell;

    if(((*sell-*buy)/*buy-1) > max revenue)
      max revenue = ((*sell-*buy)/*buy-1);
  }
  return max revenue;
}
```

שאלה 3

נתונים אוסף של מקטעים על ציר X. כל מקטע מסומן ע"י שני מספרים שלמים **(x_start, x_end)** המציינים היכן מתחיל המקטע והיכן הוא מסתיים.

סעיף א - 15 נקודות

שני מקטעים הם **חופפים** אם החיתוך שלהם אינו ריק. לדוגמא: המקטעים $(1,3)$ ו $(2,4)$ חופפים, אך המקטעים $(3,4)$ ו-$(-1,2)$ אינם חופפים.



מקטעים חופפים מקטעים לא חופפים

כתבו פונקציה שמחשבת את המספר המירבי של מקטעים חופפים עבור קבוצת מקטעים נתונה. הפונקציה מקבלת מערך דו מימדי של מקטעים בשם **segments**, כאשר **segments[0][i]** מכיל את תחילת הסגמנט ה-i ו- **segments[1][i]** מכיל את סופו.

לדוגמא, עבור המקטעים: $(-2,4),(3,7),(5,10),(6,8),(9,14)$ המספר המירבי של מקטעים חופפים הוא 3, מכיוון שאת הקטע (6, 7) מכסים 3 מקטעים.

הנחות והנחיות נוספות:

- הניחו שמספר המקטעים מוגדר ב **#define** כ **N**.
- **חשוב:** הניחו שבכל נקודה מתחיל או מסתיים לכל היותר מקטע אחד.
- מותר להשתמש בפונקציה **msort(int arr[],int n)** המפעילה את אלגוריתם merge sort לצורך מיון מערך באורך n. מותר לשנות את תוכן המערך **segments**.
- הקפידו על סיבוכיות זמן ומקום נוסף קטנים ככל האפשר (אם יש סתירה בין השתיים – סיבוכיות זמן עדיפה על סיבוכיות מקום). פתרון נכון שאינו עומד בדרישה יזכה לניקוד חלקי.

```
int overlapping_segments(int segments[][N])
{
  int start = 0, end = 0, overlapping = 0, maxo = 0;
  msort(&(segments[0][0]), N);
  msort(&(segments[1][0]), N);

  while(start < N) {
    if(segments[0][start]<segments[1][end]) {
      start++;
      overlapping++;
    }
    else {// segments[0][start] > segments[1][end]
      end++;
      overlapping--;
```

```
  }
  maxo = (overlapping > maxo) ? overlapping : maxo;
}
return maxo;
```

סעיף ב - 10 נקודות

נקודה תיקרא **מכוסה ע"י מקטע** אם קיים מקטע המכיל אותה. למשל המקטע $[1,10]$ מכיל את הנקודה 7 אך אינו מכיל את הנקודה 11. אם הנקודה ממוקמת על גבול המקטע היא נחשבת מוכלת בו.
בסעיף זה נניח כי כל המקטעים הם באורך k.
כתבו פונקציה המקבלת מערך חד מימדי של התחלות מקטעים כאלו בשם **segments**, את אורך המקטעים k וכן נקודה x, ומחזירה 1 אם הנקודה x מכוסה ע"י מקטע כלשהו או אפס אחרת.
שימו לב שהמערך ממוין לפי נקודות ההתחלה, כלומר מקטע המתחיל לפני מקטע אחר יופיע לפניו.
לדוגמא, עבור המקטעים המתחילים ב- 10 וב- 15 **segments[] = {10, 15}** ו-**k=3**:

- עבור **x=12** תחזיר הפונקציה 1
- עבור **x=14** תחזיר הפונקציה 0

<u>הנחות והנחיות נוספות:</u>

- הניחו שמספר המקטעים מוגדר ב **#define** כ **N**.
- הניחו שאין שני מקטעים צמודים, כלומר שאין נקודה המשמשת כנקודת התחלה או סיום של שני סגמנטים שונים.
- הקפידו על סיבוכיות זמן ומקום נוסף קטנים ככל האפשר. פתרון נכון שאינו עומד בדרישה יזכה לניקוד חלקי בלבד.

```
int is_covered(int arr[N], int k, int x)
{
  int start = 0, end = N-1, curr;

  while (start <= end) {
    curr = (start + end)/2;
    if (x < arr[curr])
      end = curr-1;
    else if (x <= arr[curr]+k)
      return 1;
    else
      start = curr+1;
  }
  return 0;
}
```

שאלה 4

נוסחת CNF נקראת "k-ספיקה" אם קיימת הצבה למשתנים המספקת לפחות k ליטרלים בכל פסוקית. מטרת שאלה זו לכתוב פונקציה מסוג backtracking המכריעה אם נוסחת CNF היא k-ספיקה. לאורך השאלה m יסמן את מספר הפסוקיות, N יסמן את מספר המשתנים ו-k יסמן את רמת הספיקות (שלושת המספרים הללו הם מטיפוס unsigned int, הניחו ש-N מוגדר ע"י **`#define`**).

הגדרה: הצבה חלקית ***k-מספקת*** נוסחת CNF אם ורק אם בכל פסוקית בנוסחה יש לפחות k ליטרלים שונים אשר מקבלים ערך TRUE.

תזכורת: נוסחת CNF היא מהצורה C_0 AND C_1 AND … AND C_{M-1} כאשר:

- כל C_i נקראת **פסוקית**, והינה מהצורה: $C_i \equiv \left(L_j \text{ OR } L_k \text{ OR } ... \text{ OR } L_t\right)$
- כל L_j נקרא **ליטרל**, והינו משתנה בולאני (x_j) או ההופכי שלו ($!x_j$).

דוגמא לנוסחת CNF:

$$\left(x_1 \text{ OR } x_4\right) \text{ AND } \left(!x_2 \text{ OR } x_0 \text{ OR } !x_3\right) \text{ AND } \left(!x_4\right) \text{ AND } \left(!x_1 \text{ OR } x_3 \text{ OR } x_0 \text{ OR } !x_4\right)$$

הצבה היא קביעת ערך (TRUE או FALSE) למשתנים הבוליאניים. הצבה חלקית משמעותה קביעת ערך לחלק מן המשתנים. הצבה חלקית **מספקת** את הנוסחה אם ורק אם בכל פסוקית, יש לפחות ליטרל אחד אשר מקבל ערך TRUE.

לצורך שמירת נוסחת ה-CNF והצבות חלקיות נעשה שימוש בטיפוס **`truth`** המוגדר באופן הבא:

```
typedef enum {FALSE, UNSET, TRUE} truth;
```

הקלט לבעיה: מערך **`CNF[m][N]`** מטיפוס truth כפי שהוגדר בכיתה וקבוע ספיקות **`k`** מטיפוס **`unsigned int`**. בנוסף נעשה שימוש במערך **`assignment[N]`** מטיפוס **`truth`** לשמירת ההצבה החלקית (כפי שעשינו בכיתה).

הפלט לבעיה: אם קיימת הצבה חלקית k-מספקת אזי יש להדפיסה. (אם יש מספר הצבות כאלו, מספיק להדפיס אחת מהן.) אם לא קיימת הצבה k-מספקת יש להדפיס הודעה האומרת שהנוסחה איננה k-ספיקה.

הקוד של אלגוריתם DPLL שהוצג בכיתה מובא לנוחיותכם.

```
truth DPLLk(truth CNF[][N], truth assignment[], int m,
     int i, int k)
// i denotes the smallest unassigned variable. To use the
// function call DPLL(CNF, assignment, m, 0).
{
  truth current_truth;
  if (i==0 && k_literals_per_clause(CNF, m, k) == FALSE)
    return FALSE; // perform k_lit test only once, when i==0
  current_truth = CNF_SATk(CNF,assignment,m);
  if (current_truth==TRUE)
    return TRUE;
  if (current_truth==FALSE)
    return FALSE;
  assignment[i]=FALSE;
  if (DPLLk(CNF, assignment, m, i+1, k)==TRUE)
    return TRUE;
  assignment[i]=TRUE;
  if (DPLLk(CNF, assignment, m, i+1, k)==TRUE)
    return TRUE;
  assignment[i]=UNSET;
  return FALSE;
}
truth CNF_SATk (truth CNF[][N], truth assignment[], int m,
     int k)
{
  truth t=TRUE;
  int i;
  truth clause_truth;
  for (i=0;i<m;i++){
    clause_truth=clause_SATk(CNF[i], assignment, k);
    if (clause_truth==FALSE)
      return FALSE;
    if (clause_truth==UNSET)
      t=UNSET;
  }
  return t;
}
truth clause_sat (truth clause[], truth assignment[], int n)
{
  int j;
  truth t=FALSE;
  for (j=0;j<n;j++) {
    if (clause[j]!=0) {
      if (assignment[j]==clause[j])
        return TRUE;
      if (assignment[j]==UNSET)
        t=UNSET;
    }
  }
  return t;
}
```

סעיף א – 4 נקודות

הדפיסו את הפלט הרצוי על שני הקלטים הבאים עם קבוע ספיקות k=2:

```
1. CNF[3][3]={{TRUE, UNSET, UNSET}, {TRUE, TRUE, TRUE}, {TRUE, TRUE, FALSE}}
2. CNF[3][3]={{TRUE, FALSE, FALSE}, {TRUE, TRUE,UNSET},{TRUE, UNSET, FALSE}}
```

1. הנוסחה אינה 2-ספיקה מכיוון שבפסוקית הראשונה יש רק ליטרל אחד.

2. הנוסחה המיוצגת במערך היא

(x1v!x2v!x3)^(x1vx2)^(x1v!x3)

ההצבה x1=TRUE, x2=TRUE, x3=FALSE היא 2-מספקת את הנוסחה, ולכן הנוסחה היא 2-ספיקה.

סעיף ב – 6 נקודות

אם קיימת פסוקית ובה פחות מ-k ליטרלים, ברור שהנוסחה איננה k-ספיקה. כתבו קוד לפונקציה הבאה המחזירה ערך FALSE אם קיימת פסוקית בת פחות מ-k ליטרלים ומחזירה ערך TRUE אם בכל פסוקית יש לפחות k ליטרלים.

```
truth k_literals_per_clause(truth CNF[][N], int m, int k)
{
  int i, j;

  for(i=0; i<m; i++) {
    int num literals = 0;
    for(j=0; j<N; j++)
      num literals += (CNF[i][j] != UNSET);
    if(num literals < k)
      return FALSE;
  }

  return TRUE;
}
```

סעיף ג – 6 נקודות

כתבו קוד לפונקציה הבאה הקובעת האם ההצבה החלקית הנתונה במערך **assignment[]** הינה k-מספקת את האילוץ clause שהינו מערך בגודל n.

```
truth clause_SATk(truth clause[], truth assignment[], int k)
{
  int i;
  int num unset=0, num true=0;

  for(i=0; i<N; i++) {
    if(clause[i] == UNSET)
      continue;
    num unset += (assignment[i] != UNSET);
    num true += (clause[i]== assignment[i]);
  }

  if(num true>=k)
    return TRUE;

  return(((num true+num unset)>=k)? UNSET : FALSE);
}
```

סעיף ד – 9 נקודות

כתבו קוד לפתרון בעיית ה-k-ספיקות. יש לעשות שימוש יעיל ככל הניתן בפונקציה שבסעיף ב'. מותר להשתמש בפונקציות שהוגדרו בסעיפים קודמים גם אם לא עניתם על סעיפים אלו. מותר לכתוב את התשובה על גבי ותוך שימוש בקוד שהוצג בכיתה ונתון בעמוד 21.