שאלה 4 (25 נקודות)

נוסחת CNF נקראת *דלילה* אם כל משתנה מופיע בעשר פסוקיות שונות לכל היותר. מטרת שאלה זו היא לכתוב גרסא יעילה יותר של פונקצית DPLL המיועדת לפתור את בעיית הספיקות (אך ורק) על נוסחאות דלילות. לאורך השאלה, מספר המשתנים יסומן ב-N ומספר הפסוקיות ב-M (שני המספרים הם מספרים שלמים ומוגדרים ע"י `#define`). הגדרתה של נוסחת CNF מופיעה בתחתית העמוד, כתזכורת.

לצורך שמירת נוסחת ה-CNF והצבות חלקיות לנוסחה נשתמש בטיפוס `truth` שהוגדר בכיתה כך:

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

נוסחת ה-CNF נשמרת במערך דו-מימדי `truth CNF[m][N]` וכל שורה בו מייצגת פסוקית כך:

- `CNF[i][j]=TRUE` אם הליטרל x_j שייך לאילוץ ה-i
- `CNF[i][j]=FALSE` אם הליטרל x_j! שייך לאילוץ ה-i
- `CNF[i][j]=UNSET` כאשר גם x_j וגם x_j! אינם מופיעים כלל באילוץ ה-i

הצבה חלקית נשמרת במערך `truth assignment[N]`.

הקוד של אלגוריתם DPLL שהוצג בכיתה מובא לנוחיותכם.

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

- כל C_i נקראת **פסוקית**, והינה מהצורה: $C_i \equiv \left(L_j \text{ OR } L_k \text{ OR } \ldots \text{ OR } L_t\right)$
- 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.

לדוגמא: ההצבה החלקית $x_0=x_1=$TRUE, $x_2=x_3=$UNSET, $x_4=$FALSE מספקת את הנוסחה שלעיל, ולעומת זאת ההצבה החלקית $x_0=x_1=x_2=x_3=$UNSET, $x_4=$TRUE אינה מספקת הנוסחה, שכן הפסוקית השלישית מקבלת ערך FALSE תחת הצבה זו.

```
truth DPLL(truth CNF[][N], truth assignment[], int m, int i)
// i denotes the smallest unassigned variable. To use the
// function call DPLL(CNF, assignment, m, 0).
{
  truth current_truth;
  current_truth = CNF_sat(CNF,assignment,m);
  if (current_truth==TRUE)
    return TRUE;
  if (current_truth==FALSE)
    return FALSE;
  assignment[i]=FALSE;
  if (DPLL(CNF, assignment, m, i+1)==TRUE)
    return TRUE;
  assignment[i]=TRUE;
  if (DPLL(CNF, assignment, m, i+1)==TRUE)
    return TRUE;
  assignment[i]=UNSET;
  return FALSE;
}

truth CNF_sat (truth CNF[][N], truth assignment[], int m)
{
  truth t=TRUE;
  int i;
  truth clause_truth;
  for (i=0;i<m;i++){
    clause_truth=clause_sat(CNF[i], assignment, N);
    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'

לצורך שיפור זמן הריצה על נוסחאות דלילות, נעשה שימוש במערך `int appears[N][10]` אשר יציין לכל משתנה את מספריהן של הפסוקיות בהן הוא מופיע. כתבו קוד בעל סיבוכיות זמן ריצה קטנה ככל האפשר לפונקציה הבאה המאתחלת את המערך `appears`:

```
truth appears_init(truth CNF[][N], int m, int appears[][10])
```

לאחר היציאה מהפונקציה על השורה ה-i של המערך `appears` להכיל את מספריהן של עשר הפסוקיות בהן מופיע המשתנה ה-i. אם משתנה זה מופיע בפחות מעשר פסוקיות, על התאים הנותרים להכיל את הערך 1-. על הפונקציה להחזיר ערך `TRUE` אם הנוסחה היא דלילה. אחרת, עליה להחזיר ערך `FALSE`.

```
truth appears_init(truth CNF[][N], int m, int appears[][10])
{
```

סעיף ב'

עדכנו את הקוד של פונקצית `CNF_sat` שהוצגה בכיתה כך שהפונקציה תבדוק **רק** את הספיקות של הפסוקיות המכילות את המשתנה האחרון שהוצב. סיבוכיות זמן הריצה צריכה להיות O(N).
הפונקציה תחזיר ערך FALSE **אם ורק אם** אחת הפסוקיות הללו מקבלת ערך FALSE. בכל מקרה אחר הפונקציה תחזיר ערך UNSET. הפונקציה תקבל כקלטים נוספים את מספר המשתנה האחרון לו הוצב ערך (`last`) ואת המערך `appears` מסעיף א'.
ניתן להניח כי המערך מאותחל כמתואר בסעיף א', גם אם לא פתרתם את סעיף א'. מותר לעשות שימוש בקוד DPLL המוצג לעיל.

```
truth CNF_sat2(truth CNF[][N], truth assignment[], int m,
               int appears[][10], int last)
{
```

סעיף ג'

עדכנו את הקוד של פונקצית DPLL כך שסיבוכיות זמן הריצה על נוסחאות דלילות תהיה $O(2^N*N)$.
לצורך כך יש להשתמש בפונקציה המתוארת בסעיף ב', ומותר להשתמש בה אפילו אם לא פתרתם את סעיף ב'. שימו לב לתנאי העצירה של הרקורסיה ולנכונות הפלט של הפונקציה.

```
truth DPLL2(truth CNF[][N], truth assignment[], int m,
            int i, int appears[][10])
{
```