עבודת בית מספר 2

חלקים א' ו-ב'

מערכים, פונקציות ובעיית הספיקות

מבוא למדעי המחשב, סמסטר א' תשע"ט

המחלקה למדעי המחשב, אוניברסיטת בן-גוריון בנגב

מבוא למדעי המחשב – סמסטר א' תשע"ט

עבודת בית מספר 2: מערכים, פונקציות ובעיית הספיקות

צוות העבודה:

מרצה אחראי: פרופ' מיכאל קודיש

מתרגלים אחראים: אבי יצחקוב ונועה בן דוד

תאריך פרסום: 16/11/18

מועד אחרון להגשה: 25/11/18 בשעה 12:00 בצהריים

בעבודת בית זו נתרגל עבודה עם מערכים ופונקציות בג'אווה ונפגוש את בעיית הספיקות יחד עם כמה מושגים חשובים נוספים במדעי המחשב.

נכתוב תכנית לפתרון בעיית *הטיול הגדול*: בבעיה זו נתונים קבוצה של ערים וקווי תעופה ביניהן. יש לתכנן מסלול לטיול שיוצא מנמל הבית, עובר בכל שאר הערים וחוזר לנמל הבית, כך שכל עיר תופיע בדיוק פעם אחת במסלול וכן בסיומו נחזור לעיר המקור.

האלגוריתם שנממש לפתרון הבעיה מבוסס על רדוקציה ל"בעיית הספיקות", או באנגלית:
The Boolean Satisfiability Problem (SAT). את הנוסחה שנקבל מהרדוקציה נפתור בעזרת "פותרן של בעיית הספיקות" (SAT Solver).

לעבודה זו שני חלקים שמפורסמים בנפרד: בחלק א' נממש מספר פונקציות לצורך בדיקת תקינות של מופע נתון לבעיית הטיול הגדול וכן מספר פונקציות לווידוא נכונות של פתרון למופע נתון. בחלק ב', נממש את האלגוריתם לפתרון הבעיה שמבוסס על רדוקציה לבעיית הספיקות .

מרכיבי הציון לעבודה: חלק א' – 35 נקודות, חלק ב' – 65 נקודות.

מסמך זה כולל את שני חלקי העבודה. עליכם להגיש את הקוד של **כל המשימות של חלק א' וגם של חלק ב'** בקובץ Assignment2.java המסופק לכם. שימו לב, למרות שהגשתם כבר את חלק א', אתם רשאים לשנות את הקוד של משימות מחלק זה כרצונכם (לתקן/לייעל/לשנות את המימוש). שני החלקים בעבודה זו ייבדקו ביחד מהקובץ Assignment2.java שתגישו.

הוראות מקדימות:

הגשת עבודות בית

1. **קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה.** ודאו שאתם מבינים את כל המשימות. רמת הקושי של המשימות אינה אחידה: הפתרון של חלק מהמשימות קל יותר, ואחרות מצריכות חקירה מתמטית - שאותה תוכלו לבצע בספרייה או בעזרת מקורות דרך רשת האינטרנט. בתשובות שבהן אתם מסתמכים על עובדות מתמטיות שלא הוצגו בשיעורים, יש להוסיף כהערה במקום המתאים בקוד את ציטוט העובדה המתמטית ואת המקור (כגון ספר או אתר).
2. עבודה זו תוגש ביחידים. כדי להגיש את העבודה יש להירשם למערכת ההגשות (Submission System). את הרישום למערכת ההגשות מומלץ לבצע כבר עכשיו, טרם הגשת העבודה (קחו בחשבון כי הגשה באיחור אינה מתקבלת). את הגשת העבודה ניתן לבצע רק לאחר הרישום למערכת.
3. לעבודה מצורף קובץ Assignment2.java. עליכם לערוך קובץ זה בהתאם למפורט בתרגיל ולהגישו כפתרון, מכווץ כקובץ ZIP יחיד. שימו לב: עליכם להגיש רק את קובץ ה-Java. אין לשנות את שם הקובץ, ואין להגיש קבצים נוספים. שם קובץ ה-ZIP יכול להיות כרצונכם, אך באנגלית בלבד. בנוסף, הקובץ שתגישו יכול להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכד'..) לא יתקבל. בנוסף מצורפים: קובץ בדיקות Tests.java וקבצי דוגמאות לנוסחאות cnf: ExamplesSAT.java, ExamplesUNSAT.java. הקבצים הנ"ל הם קבצי עזר עבורכם והם אינם להגשה. חשוב להדגיש כי קובץ הטסטים שסופק לכם מכיל בדיקות חלקיות של חלק מהמשימות, אנו מעודדים אתכם להרחיב קובץ זה ולהשתמש בו על מנת לוודא את נכונות המימוש שלכם.
4. קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו. את קובץ ה-ZIP יש להגיש ב-Submission System. פרטים בעניין ההרשמה ואופן הגשת העבודה תוכלו למצוא באתר.

בדיקת עבודות הבית

5. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי.
6. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד ברור, על מתן שמות משמעותיים למשתנים, על הזחות (אינדנטציה), ועל הוספת הערות בקוד המסבירות את תפקידם של מקטעי הקוד השונים. אין צורך למלא את הקוד בהערות סתמיות, אך חשוב לכתוב הערות בנקודות קריטיות המסבירות קטעים חשובים בקוד. הערות יש לרשום אך ורק באנגלית. כתיבת קוד אשר אינה עומדת בדרישות אלו תגרור הפחתה בציון העבודה.

עזרה והנחיה

7. לכל עבודת בית בקורס יש צוות שאחראי לה. ניתן לפנות לצוות בשעות הקבלה. פירוט שמות האחראים לעבודה מופיע במסמך זה וכן באתר הקורס, כמו גם פירוט שעות הקבלה. בשאלות טכניות אפשר גם לגשת לשעות התגבורים, שבהן ניתנת עזרה במעבדה. כמו כן, אתם יכולים להיעזר בפורום ולפנות בשאלות לחבריכם לכיתה. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך.
8. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.

הערות ספציפיות לעבודת בית זו

9. בעבודה זו 20 משימות וסך הנקודות המקסימלי הוא 100. הניקוד לכל משימה שווה (5 נקודות)
10. בעבודה זו מותר להשתמש בידע שנלמד עד הרצאה 8 (כולל), וכן עד תרגול 4 (כולל).

יושר אקדמי

הימנעו מהעתקות! ההגשה היא ביחידים. אם מוגשות שתי עבודות עם קוד זהה או אפילו דומה - זוהי העתקה, אשר תדווח לאלתר לוועדת משמעת. אם טרם עיינתם בסילבוס הקורס, אנא עשו זאת כעת.

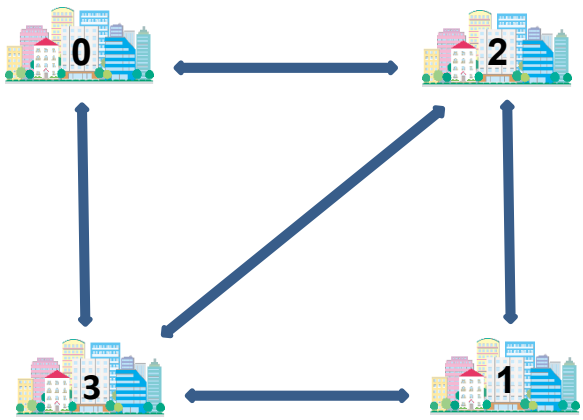
מומלץ לקרוא היטב את כל ההוראות המקדימות ורק לאחר מכן להתחיל בפתרון המשימות. ודאו שאתם יודעים לפתוח קבוצת הגשה (עבור עצמכם) במערכת ההגשות.

1. בעיית הטיול הגדול

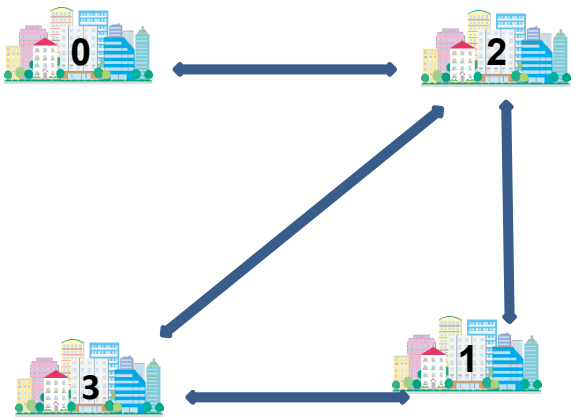
מבוא

בהינתן רשימה של קווי תעופה בין n ערים הממוספרות ב- $\{0 \dots n-1\}$, יש לתכנן מסלול המתחיל מנמל הבית הנמצא בעיר המקור 0, מבקר בכל אחת מן הערים האחרות $\{1 \dots n-1\}$ בדיוק פעם אחת ובסיומו חוזר לנמל הבית שבעיר המקור (מסלול מעגלי). קיומו של קו תעופה $\{i,j\}$ בין שני ערים שונות i,j , מציין שקיימת טיסה בשני הכיוונים $(i \to j,\ j \to i)$.

דוגמא 1: נניח שיש 4 ערים $\{0,1,2,3\}$ ושקווי התעופה הם: $\{1,3\},\{2,3\},\{1,2\},\{0,3\},\{0,2\}$, כפי שמוצג באיור מספר 1. ניתן לתכנן מסלול שפותר את בעיית הטיול הגדול. למשל המסלול: $0 \to 3 \to 1 \to 2 \to 0$.
דוגמא 2: נניח שיש 4 ערים $\{0,1,2,3\}$ ושקווי התעופה הם:$\{1,3\},\{2,3\},\{1,2\},\{0,2\}$, כפי שמוצג באיור מספר 2. בדוגמא זו לא ניתן למצוא מסלול שפותר את בעיית הטיול הגדול.



איור 1. הצגה גרפית של קווי התעופה מדוגמא 1



איור 2. הצגה גרפית של קווי התעופה מדוגמא 2

ייצוג מופע של בעיית הטיול הגדול ב-Java

מופע של בעיית *הטיול הגדול* עבור n ערים מיוצג באמצעות מטריצה בוליאנית flights בגודל $n \times n$, כאשר הערך בתא ה (i,j) במטריצה הוא true אם ורק אם קיים קו תעופה $\{i,j\}$.

הגדרה 1: מערך דו-ממדי flights מייצג מופע חוקי של בעיית הטיול הגדול אם הוא מטריצה בוליאנית שהיא:
א. ריבועית – מכילה n מערכים שכל אחד מהם באורך n.
ב. סימטרית - לכל $0 \le i \le j < n$ מתקיים flights[i][j] = flights[j][i].
ג. אנטי-רפלקסיבית – לכל $0 \le i < n$ מתקיים flights[i][i]=false.

דוגמה: המופע של בעיית הטיול הגדול שמוצג באיור מספר 1 ייוצג ב-Java באופן הבא:

```
boolean[][] flights = {{false, false, true,  true },
                       {false, false, true,  true },
                       {true,  true,  false, true },
                       {true,  true,  true,  false}};
```

ייצוג פתרון לבעיית הטיול הגדול ב *Java*

פתרון למופע של בעיית *הטיול הגדול* עבור n ערים מיוצג באמצעות מערך חד-ממדי בגודל n של מספרים שלמים, כאשר הערך בתא ה-i מציין את מספר העיר שמבקרים בה בשלב ה-i של הטיול. למשל, המסלול $0 \to 3 \to 1 \to 2 \to 0$ המהווה פתרון עבור המופע שמתואר בדוגמא 1, מיוצג באמצעות המערך [0,2,1,3]. נשים לב שהחזרה לעיר המקור (0) בסוף הטיול לא מיוצגת באופן מפורש במערך.

הגדרה 2: מערך חד-ממדי A של מספרים שלמים, מייצג פתרון למופע של בעיית הטיול הגדול על n ערים אם:

א. A הוא מערך באורך n המכיל את כל המספרים $0 \dots n-1$ (כל הערים מופיעות במסלול בדיוק פעם אחת).
ב. A[0]=0 - העיר הראשונה במסלול היא 0.
ג. לכל $0 \le i < n-1$, קיים קו תעופה בין A[i] ל-A[i+1] וכן קיים קו תעופה בין $A[n-1]$ ל-A[0].

וידוא תקינות קלט

במשימות הבאות נממש מספר פונקציות שיסייעו לנו לבדוק האם מופע נתון לבעיית הטיול הגדול הוא תקין.

משימה 1 (מטריצה ריבועית) (5 נקודות):

בהינתן מערך בוליאני דו-ממדי באורך כלשהו, נרצה לוודא כי המערך מייצג מטריצה ריבועית. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean isSquareMatrix (boolean[][] matrix)
```

על הפונקציה להחזיר ערך true אם ורק אם המערך matrix מכיל n מערכים באורך n ($n \ge 0$).

הנחות על הקלט וחריגות:

- אין להניח שום הנחות על הקלט.
- יש להחזיר ערך false אם הקלט אינו תקין.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
boolean[][] matrix1 = {{false,false},{true,true}} ;
System.out.println(isSquareMatrix(matrix1)); // true ;

int[][] matrix2 = {{true,false,true},{false,false}} ;
System.out.println(isSquareMatrix(matrix2)); // false ;

int[][] matrix3 = null;
System.out.println(isSquareMatrix(matrix3)); // false ;
```

משימה 2 (מטריצה סימטרית) (5 נקודות):

בהינתן מטריצה בוליאנית בגודל $n \times n$, נרצה לוודא כי המטריצה היא סימטרית. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean isSymmetricMatrix (boolean[][] matrix)
```

על הפונקציה להחזיר ערך true אם ורק אם לכל $0 \leq i < j < n$ מתקיים matrix[i][j]=matrix[j][i].

הנחות על הקלט וחריגות:
- הניחו כי matrix היא מטריצה ריבועית.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
boolean[][] matrix1 = {{false, false, true},
                       {false, false,  true},
                       {true,  true,  true}};
System.out.println(isSymmetricMatrix(matrix1)); // true
```

<u>משימה 3</u> <u>(מטריצה אנטי-רפלקסיבית) (5 נקודות):</u>

בהינתן מטריצה בוליאנית בגודל $n \times n$, נרצה לוודא כי המטריצה היא אנטי-רפלקסיבית. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean isAntiReflexiveMatrix (boolean[][] matrix)
```

הנחות על הקלט וחריגות:
- הניחו כי matrix היא מטריצה ריבועית.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
boolean[][] matrix1 = {{false, false},
                       {true,  false }};
System.out.println(isAntiReflexiveMatrix(matrix1)); // true

boolean[][] matrix2 = {{false, false},
                       {true,  true }};
System.out.println(isAntiReflexiveMatrix(matrix2)); // false
```

<u>משימה 4</u> <u>(מופע חוקי) (5 נקודות):</u>

בהינתן מערך דו-ממדי, נבדוק שהוא מייצג מופע תקין של בעיית הטיול הגדול. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean isLegalInstance (boolean[][] matrix)
```

הפונקציה תחזיר ערך true אם ורק אם המערך הדו-ממדי matrix מקיים את התנאים של מופע תקין לפי **הגדרה 1**.

הנחות על הקלט וחריגות:
- אין להניח שום הנחות על הקלט.
- יש להחזיר ערך false אם הקלט אינו תקין.
- פונקציה זו לא זורקת חריגות.

<u>וידוא פתרון לבעיית הטיול הגדול</u>

במשימות הבאות נממש מספר פונקציות לצורך וידוא פתרון של בעיית הטיול הגדול.

<u>משימה 5</u> <u>(המסלול עובר בכל הערים) (5 נקודות):</u>

הגדרה (פרמוטציה): מערך $array$ יקרא פרמוטציה אם הוא מכיל את על המספרים השלמים בין 0 ל- $array.length - 1$ כולל. כלומר, כל ערך בטווח הנ"ל יופיע <u>בדיוק פעם אחת</u>
דוגמאות: המערכים [0,2,1,3], [1,0] הם פרמוטציות ואילו המערכים [0,1,2,2], [0,2], [1,4,3,2] אינם פרמוטציות.

השלימו את הפונקציה הבאה בקובץ Assignment2.java::

```
public static boolean isPermutation (int[] array)
```

הפונקציה תחזיר את הערך true אם ורק אם המערך array הוא פרמוטציה.

הנחות על הקלט וחריגות:
- הניחו כי array אינו null.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
int[] array1 = {0,2,3,1};
System.out.println(isPermutation(array1)); //true

int[] array2 = {1,4,3,2};
System.out.println(isPermutation(array2)); //false
```

<u>משימה 6</u> <u>(כל הטיסות במסלול קיימות) (5 נקודות)</u>

בהינתן מערך דו ממדי בוליאני flights המייצג מופע של בעיית הטיול הגדול ומערך חד-ממדי של מספרים שלמים, tour, שמייצג מסלול, נבדוק שבין כל שני ערים עוקבות במסלול קיימת טיסה. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean hasLegalSteps (boolean[][] flights, int[] tour)
```

הפונקציה מקבלת מערך flights דו מימדי בגודל $n \times n$ ומערך *tour* באורך *n*. הפונקציה תחזיר ערך true אם ורק אם לכל $0 \leq i < n - 1$ יש קו תעופה בין tour[i] ל-tour[i+1] וגם יש קו תעופה חזרה מ-tour[n-1] ל- tour[0].

הנחות על הקלט וחריגות:
- הניחו שהמערך flights מייצג מופע תקין על n ערים.
- המערך tour הוא באורך n וערכיו הם מהתחום [0,n-1].
- פונקציה זו לא זורקת חריגות.

<u>משימה 7</u> <u>(פתרון חוקי) (5 נקודות):</u>

בהינתן מערך דו-ממדי flights המייצג מופע של בעיית הטיול הגדול ומערך חד-ממדי tour, נבדוק שהמערך מהווה פתרון למופע. השלימו את הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean isSolution(boolean[][] flights, int[] tour)
```

הפונקציה תחזיר ערך true אם ורק אם המערך tour מקיים את התנאים לפתרון לפי **הגדרה 2** עבור המופע flights.

הנחות על הקלט וחריגות:

- הניחו שהמערך flights מייצג מופע תקין על $n \geq 0$ ערים.
- אין להניח שום הנחות על המערך tour.
- פונקציה זו זורקת חריגה אם tour אינו מערך באורך n.

2. בעיית הספיקות

בחלק זה של העבודה נכיר את הייצוג של משתנים ופסוקיות CNF ב-Java וכן נלמד כיצד להשתמש ב"פותרן לבעיית הספיקות".

תזכורת לגבי תחשיב הפסוקים

נוסחה בוליאנית בצורת CNF היא קוניונקציה ("וגם") של פסוקיות. פסוקית היא דיסיונקציה ("או") של ליטרלים. ליטרל הוא משתנה בוליאני או שלילה של משתנה בוליאני. בכדי להימנע מבלבול בין המשתנים של ג'אווה לבין אלו של ה-CNF, למשתנים של ה-CNF נדייק ונקרא משתני CNF.

השמה היא פונקציה (מתמטית) אשר מתאימה לכל משתנה ערך true או false.

עבור נוסחה בוליאנית בצורת CNF, השמה היא מספקת אם היא מספקת את כל הפסוקיות. השמה מספקת פסוקית אם היא מספקת לפחות את אחד הליטרלים בפסוקית. השמה מספקת ליטרל אם: הליטרל הוא מהצורה x_i וההשמה מציבה ערך true למשתנה ה-CNF x_i, או שהליטרל הוא מהצורה $\neg x_i$ וההשמה מציבה ערך false למשתנה ה-CNF x_i.

בעיית הספיקות עוסקת בשאלה: בהינתן נוסחה בוליאנית, האם קיימת עבורה השמה מספקת?

תזכורת לגבי הייצוג של נוסחאות ב-Java:

בייצוג של ג'אווה, משתני ה-CNF תמיד יהיו ממוספרים ברצף מ-1 ועד n: $x_1, x_2, \dots, x_n$.

- את הליטרל x_i נייצג בג'אווה באמצעות המספר i, ואת הליטרל $\neg x_i$ נייצג באמצעות המספר $-i$.
- את הפסוקית $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_r$ נייצג בג'אווה באמצעות מערך המכיל את הייצוג של הליטרלים. למשל, את הפסוקית $(x_1 \vee x_4 \vee \neg x_{17} \vee x_6 \vee x_4 \vee x_{19} \vee \neg x_3)$ נייצג בג'אווה באמצעות המערך:

  int[] clause = {1,4,-17,6,4,19,-3}

- את נוסחת ה-CNF $c_1 \wedge c_2 \wedge \cdots \wedge c_n$ נייצג באמצעות מערך דו-ממדי. למשל, את הנוסחה $((x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee x_4 \vee \neg x_3) \wedge (\neg x_5 \vee x_8 \vee \neg x_{12}))$ נייצג בג'אווה באמצעות המערך הדו-ממדי:

  int[][] formula = {{1,3,5}, {2,4,-3},{-5,8,-12}}

תזכורת לגבי הייצוג של השמה ב-Java:

נייצג השמה למשתני ה-CNF $x_1, \dots, x_n$ באמצעות מערך בוליאני assignment באורך $n+1$, כאשר assignment[i] היא הערך של משתנה ה-CNF i תחת ההשמה הנתונה. שימו לב שבייצוג זה אין משמעות לאיבר assignment[0], הראשון במערך. למשל, את ההשמה $x_1 = false, x_2 = false, x_3 = true, x_4 = true$ נוכל לייצג בג'אווה באמצעות המערך boolean[] assignment = {true, false, false, true, true}. יש לשים לב שאין משמעות לערך באיבר הראשון.

נכתוב פונקציות ב-Java שמגדירות שלוש נוסחאות בוליאניות. כל אחת מהן תבטא אילוץ על קבוצה של משתני CNF. נוסחה מבטאת אילוץ על קבוצה של משתני CNF **אם קבוצת ההשמות המספקות שלה תואמות את כל האופנים שבהם ניתן לספק את האילוץ.** בהינתן אילוץ, נרצה להגדיר נוסחה שקבוצת ההשמות המספקות שלה תואמת בדיוק את האופנים שבהם ניתן לספק את האילוץ.

דוגמה: עבור משתנים בוליאנים x, y ואילוץ שאומר ש-$(x = y)$, נוסחאת ה-CNF: $(x \vee \neg y) \wedge (\neg x \vee y)$ מבטאת את האילוץ. זאת משום שלנוסחה לעיל ארבע השמות אפשריות, מתוכן רק שתי השמות מספקות, והן: $\{x = true, y = true\}$ ו- $\{x = false, y = false\}$.

<u>משימה 8:</u> (לפחות אחד)(5 נקודות)

במשימה זו נבנה נוסחת CNF אשר מבטאת אילוץ שאומר שלפחות משתנה CNF אחד מתוך קבוצה של משתנים נתונים מקבל את הערך true. השלימו את הגדרת הפונקציה בקובץ Assignment2.java::

```
public static int[][] atLeastOne(int[] vars)
```

הפונקציה מקבלת מערך של (שמות של) משתני CNF ומחזירה נוסחת CNF שקבוצת ההשמות המספקות שלה תואמת את כל האופנים שבהם ניתן לתת ערך true ללפחות אחד ממשתני ה-CNF שבמערך הקלט.

הדרכה: בהינתן קבוצת משתני CNF (בייצוג של ג'אווה) int[] vars = {2,5,7}, נוסחה שאומרת שלפחות אחד המשתנים מקבל ערך *true*, אומרת למעשה: או ש-x_2 מקבל ערך *true*, או ש-x_5 מקבל ערך *true*, או ש-x_7 מקבל את הערך *true*. רישמו נוסחה זו בצורת CNF והכלילו לקלט כלשהו.

הנחות על הקלט וחריגות:
- הניחו שהקלט תקין.
- פונקציה זו לא זורקת חריגות.

<u>משימה 9:</u> (לכל היותר אחד)(5 נקודות)

במשימה זו נבנה נוסחת CNF אשר מבטאת אילוץ שאומר שלכל היותר משתנה CNF אחד מתוך קבוצה של משתנים נתונים מקבל את הערך true. השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

```
public static int[][] atMostOne(int[] vars)
```

הפונקציה מקבלת מערך של (שמות של) משתני CNF ומחזירה נוסחת CNF שקבוצת ההשמות המספקות שלה תואמת את כל האופנים שבהם ניתן לתת ערך true ללכל היותר אחד ממשתני ה-CNF שבמערך הקלט.

הדרכה: בהינתן קבוצת משתני CNF (בייצוג של ג'אווה) int[] vars = {2,5,7}, נוסחה שאומרת שאחד המשתנים לכל היותר מקבל ערך true, אומרת למעשה: לא נכון שזוג המשתנים x_2, x_5 מקבלים שניהם את הערך true, וגם לא נכון שזוג המשתנים x_2, x_7 מקבלים שניהם את הערך true, וגם לא יתכן שזוג המשתנים x_5, x_7 מקבלים שניהם את הערך true. רישמו נוסחה זו בצורת CNF והכלילו לקלט כלשהו.

הנחות על הקלט וחריגות:
- הניחו שהקלט תקין.

- פונקציה זו לא זורקת חריגות.

<u>משימה 10:</u> (בדיוק אחד)(5 נקודות)

במשימה זו נבנה נוסחת CNF אשר מבטאת אילוץ שאומר שבדיוק משתנה CNF אחד מתוך קבוצה של משתנים נתונים מקבל את הערך true. השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

```
public static int[][] exactlyOne(int[] vars)
```

הפונקציה מקבלת מערך של (שמות של) משתני CNF ומחזירה נוסחת CNF שקבוצת ההשמות המספקות שלה תואמת את כל האופנים שבהם ניתן לתת ערך true לבדיוק אחד ממשתני ה-CNF שבמערך הקלט.

הדרכה: אם לפחות אחד המשתנים מקבל ערך *true*, וגם לכל היותר אחד המשתנים מקבל ערך *true*, אז בדיוק אחד המשתנים מקבל ערך *true*.

הנחות על הקלט וחריגות:
- הניחו שהקלט תקין.
- פונקציה זו לא זורקת חריגות.

<u>משימה 11:</u> (שימוש בפותרן)(5 נקודות)

במשימה זו נבנה נוסחת CNF שמאלץ קבוצות נתונות של משתני CNF לקיים את האילוץ "בדיוק אחד" (משימה 10). השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

```
public static boolean[]
     solveExactlyOneForEachSet(int[][] varSets)
```

הפונקציה מקבלת מערך של מערכים $A_1, \dots A_k$, כל מערך A_i מייצג קבוצה של משתני CNF. הפונקציה תייצר נוסחת CNF שמאלצת <u>כל</u> מערך A_i של משתני CNF לקיים את האילוץ "בדיוק אחד" (כלומר, בדיוק אחד ממשתני ה-CNF שב-A_i מקבל את הערך true). הפונקציה תוסיף את הנוסחה לפותרן[1] , תפעיל אותו על נוסחה זו ותחזיר את ההשמה המתקבלת (במידה ולא קיימת השמה מספקת, הפונקציה תחזיר מערך ריק). **שימו לב:** פונקציה זו משמשת אך ורק לתרגול שימוש בפותרן, ולא יהיה בה צורך במשימות הבאות.

הנחות על הקלט וחריגות:
- הניחו שהקלט תקין.
- יש לזרוק חריגה אם המופע היה בלתי פתיר עקב מגבלות זמן (timeout), פרטים נוספים על מגבלות זמן ניתן למצוא בנספח לעבודה זו.

דוגמאות:

```
int[][] varSets = {{1,2,4},{2,3},{3,4}};
boolean[] assignment = solveExactlyOneForEachSet(varSets);
// {false, true, false, true, false}
```

[1] הוראות בנוגע להתקנה ולשימוש בפותרן לבעיית הספיקות ניתן למצוא בנספח שבסוף העבודה. כמו כן, ניתן למצוא כמה דוגמאות לאילוץ exactlyOne בנספח ובקובץ Tests.java המצורף לקבצי העבודה.

3. רדוקציה לבעיית הספיקות

<u>תיאור הרדוקציה:</u>

בהינתן מופע של בעיית *הטיול הגדול על n ערים*, נייצג פתרון של הבעיה בעזרת מטריצה X של משתנים בוליאניים בגודל $n\ x\ n$: כל תא (i,j) במטריצה הוא משתנה בוליאני $X_{i,j}$ שמקבל את הערך $true$ אם ורק אם בצעד ה- i של הטיול מבקרים בעיר j.

דוגמא: עבור מופע של n=4 ערים, נגדיר את מטריצת המשתנים X הבאה:

$$\begin{bmatrix} X_{0,0}, X_{0,1}, X_{0,2}, X_{0,3} \\ X_{1,0}, X_{1,1}, X_{1,2}, X_{1,3} \\ X_{2,0}, X_{2,1}, X_{2,2}, X_{2,3} \\ X_{3,0}, X_{3,1}, X_{3,2}, X_{3,3} \end{bmatrix}$$

נרצה להגדיר נוסחה בוליאנית שכל השמה מספקת שלה נותנת ערכים למשתנים במטריצה X שמהווה פתרון של מופע נתון של בעיית הטיול הגדול. לשם כך נגדיר מספר אילוצים בוליאניים על המשתנים שבמטריצה. הניחו שהמטריצה X נתונה.

א. <u>בכל צעד מבקרים בעיר אחת</u> – בכל צעד במסלול שהוא פתרון מבקרים בדיוק בעיר אחת. כלומר, עבור כל שורה i במטריצה X בדיוק אחד מהמשתנים $X_{i,1}, \dots, X_{i,n}$ מקבל ערך true.
ב. <u>מבקרים בכל עיר רק פעם אחת</u> – מסלול שהוא פתרון יכלול כל עיר בדיוק פעם אחת. כלומר, עבור כל עמודה j במטריצה X, בדיוק אחד מהמשתנים $X_{1,j}, \dots, X_{n,j}$ מקבל את הערך true.
ג. <u>עיר המקור היא 0</u> –ההשמה של המשתנה $X_{0,0}$ היא true
ד. <u>אין צעדים לא חוקיים</u> – לכל שני ערים שונות j, k שאין ביניהם קו תעופה, אין מעבר מ-j ל-k בפתרון. כלומר, לכל i,j,k כך ש $j \neq k$, $flights[j][k] = false$ ו- $0 \leq i < n-1$, נדרוש כי לכל היותר אחד מן המשתנים $X_{i,j}, X_{i+1,k}$ יקבל ערך true. כמו כן, לכל עיר k שאין בינה לבין העיר 0 קו תעופה, נדרוש שלא ייתכן ש-k תהיה העיר האחרונה במסלול. כלומר $X_{(n-1),k} = false$

דוגמא: הנוסחה הבוליאנית הבאה (קוניונקציה של השורות, כל פסוקית מתוחמת בסוגריים) מבטאת את האילוצים הנדרשים עבור המופע הנתון בדוגמא 1.

$$\left.\begin{array}{l}
(X_{0,0} \vee X_{0,1} \vee X_{0,2} \vee X_{0,3}) \\
(\neg X_{0,0} \vee \neg X_{0,1}) \wedge (\neg X_{0,0} \vee \neg X_{0,2}) \wedge (\neg X_{0,0} \vee \neg X_{0,3}) \\
(\neg X_{0,1} \vee \neg X_{0,2}) \wedge (\neg X_{0,1} \vee \neg X_{0,3}) \wedge (\neg X_{0,2} \vee \neg X_{0,3}) \\
(X_{1,0} \vee X_{1,1} \vee X_{1,2} \vee X_{1,3}) \\
(\neg X_{1,0} \vee \neg X_{1,1}) \wedge (\neg X_{1,0} \vee \neg X_{1,2}) \wedge (\neg X_{1,0} \vee \neg X_{1,3}) \\
(\neg X_{1,1} \vee \neg X_{1,2}) \wedge (\neg X_{1,1} \vee \neg X_{1,3}) \wedge (\neg X_{1,2} \vee \neg X_{1,3}) \\
(X_{2,0} \vee X_{2,1} \vee X_{2,2} \vee X_{2,3}) \\
(\neg X_{2,0} \vee \neg X_{2,1}) \wedge (\neg X_{2,0} \vee \neg X_{2,2}) \wedge (\neg X_{2,0} \vee \neg X_{2,3}) \\
(\neg X_{2,1} \vee \neg X_{2,2}) \wedge (\neg X_{2,1} \vee \neg X_{2,3}) \wedge (\neg X_{2,2} \vee \neg X_{2,3}) \\
(X_{3,0} \vee X_{3,1} \vee X_{3,2} \vee X_{3,3}) \\
(\neg X_{3,0} \vee \neg X_{3,1}) \wedge (\neg X_{3,0} \vee \neg X_{3,2}) \wedge (\neg X_{3,0} \vee \neg X_{3,3}) \\
(\neg X_{3,1} \vee \neg X_{3,2}) \wedge (\neg X_{3,1} \vee \neg X_{3,3}) \wedge (\neg X_{3,2} \vee \neg X_{3,3})
\end{array}\right\} \boxed{\text{בכל צעד מבקרים בעיר אחת}}$$

$$
\begin{aligned}
&(X_{0,0} \vee X_{1,0} \vee X_{2,0} \vee X_{3,0}) \\
&(\neg X_{0,0} \vee \neg X_{1,0}) \wedge (\neg X_{0,0} \vee \neg X_{2,0}) \wedge (\neg X_{0,0} \vee \neg X_{3,0}) \\
&(\neg X_{1,0} \vee \neg X_{2,0}) \wedge (\neg X_{1,0} \vee \neg X_{3,0}) \wedge (\neg X_{2,0} \vee \neg X_{3,0}) \\
&(X_{0,1} \vee X_{1,1} \vee X_{2,1} \vee X_{3,1}) \\
&(\neg X_{0,1} \vee \neg X_{1,1}) \wedge (\neg X_{0,1} \vee \neg X_{2,1}) \wedge (\neg X_{0,1} \vee \neg X_{3,1}) \\
&(\neg X_{1,1} \vee \neg X_{2,1}) \wedge (\neg X_{1,1} \vee \neg X_{3,1}) \wedge (\neg X_{2,1} \vee \neg X_{3,1}) \\
&(X_{0,2} \vee X_{1,2} \vee X_{2,2} \vee X_{3,2}) \\
&(\neg X_{0,2} \vee \neg X_{1,2}) \wedge (\neg X_{0,2} \vee \neg X_{2,2}) \wedge (\neg X_{0,2} \vee \neg X_{3,2}) \\
&(\neg X_{1,2} \vee \neg X_{2,2}) \wedge (\neg X_{1,2} \vee \neg X_{3,2}) \wedge (\neg X_{2,2} \vee \neg X_{3,2}) \\
&(X_{0,3} \vee X_{1,3} \vee X_{2,3} \vee X_{3,3}) \\
&(\neg X_{0,3} \vee \neg X_{1,3}) \wedge (\neg X_{0,3} \vee \neg X_{2,3}) \wedge (\neg X_{0,3} \vee \neg X_{3,3}) \\
&(\neg X_{1,3} \vee \neg X_{2,3}) \wedge (\neg X_{1,3} \vee \neg X_{3,3}) \wedge (\neg X_{2,3} \vee \neg X_{3,3})
\end{aligned}
$$

מבקרים בכל עיר רק פעם אחת

$$
\begin{aligned}
&(\neg X_{0,0} \vee \neg X_{1,1}) \\
&(\neg X_{1,0} \vee \neg X_{2,1}) \\
&(\neg X_{2,0} \vee \neg X_{3,1}) \\
&(\neg X_{0,1} \vee \neg X_{1,0}) \\
&(\neg X_{1,1} \vee \neg X_{2,0}) \\
&(\neg X_{2,1} \vee \neg X_{3,0}) \\
&(\neg X_{3,1})
\end{aligned}
$$

אין צעדים לא חוקיים

$$(X_{0,0})$$

עיר המקור היא 0

בניית נוסחת CNF ב-Java

בסעיף הקודם ראינו כיצד להגדיר-את הנוסחה הבוליאנית עבור מופע נתון.
במשימות הבאות נממש תכנית ב-Java אשר מקבלת מופע של בעיית הטיול הגדול ומייצרת את נוסחת ה-CNF בייצוג של Java בדומה למה שעשינו במשימה 8.

משימה 12: (מיפוי המשתנים) (5 נקודות)

כזכור, משתנה CNF מיוצג ב-Java כמספר טבעי חיובי. נרצה לייצג בג'אווה את n^2 המשתנים של המטריצה X. נמפה את המשתנה $X_{i,j}$ למשתנה CNF לפי הנוסחה $i * n + j + 1$.

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static int[][] createVarsMap(int n)
```

הפונקציה מקבלת כקלט מספר n ומחזירה את טבלת המשתנים המתאימה למופע הכולל n ערים. הפונקציה מחזירה מערך דו-ממדי map בגודל $n \times n$ כך [i][j] map הוא שם משתנה ה-CNF המייצג את המשתנה $X_{i,j}$ לפי הנוסחה הנתונה לעיל.

הנחות על הקלט וחריגות:

- הניחו ש- n מספר חיובי.
- פונקציה זו אינה זורקת חריגות.

דוגמה:

```
int n = 4;
int[][] map = createVarsMap(n) ;

/*
 * map =
 * [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
 */
```

<u>משימה 13</u> (בכל צעד מבקרים בעיר אחת)(5 נקודות)

בהינתן מטריצת משתני CNF עבור מופע של בעיית הטיול הגדול, נבנה נוסחת CNF המתאימה לאילוץ "בכל צעד מבקרים בעיר אחת". השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

`public static int[][] oneCityInEachStep(int[][] map)`

הפונקציה מקבלת מטריצה map של (שמות של) משתני CNF המייצגים את המשתנים $X_{i,j}$, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לקיים את התנאי: בכל צעד במסלול שהוא פתרון מבקרים בדיוק בעיר אחת.

הנחות על הקלט וחריגות:

- הניחו ש-map מייצג מטריצת משתנים תקינה עבור מופע של n ערים.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 14</u> (עיר המקור היא 0)(5 נקודות)

בהינתן מטריצת משתני CNF עבור מופע של בעיית הטיול הגדול, נבנה נוסחת CNF המתאימה לאילוץ "עיר המקור היא 0 "

`public static int[][] fixSourceCity(int[][] map)`

הפונקציה מקבלת מטריצה map של (שמות של) משתני CNF המייצגים את המשתנים $X_{i,j}$, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לקיים את התנאי: ההשמה של המשתנה $X_{0,0}$ היא true

הנחות על הקלט וחריגות:

- הניחו ש-map מייצג מטריצת משתנים תקינה עבור מופע של n ערים.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 15</u> (מבקרים בכל עיר רק פעם אחת)(5 נקודות)

בהינתן מטריצת משתני CNF עבור מופע של בעיית הטיול הגדול, נבנה נוסחת CNF המתאימה לאילוץ "מבקרים בכל עיר רק פעם אחת". השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

`public static int[][] eachCityIsVisitedOnce(int[][] map)`

הפונקציה מקבלת מטריצה map של (שמות של) משתני CNF המייצגים את המשתנים $X_{i,j}$, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לקיים את התנאי: מסלול שהוא פתרון יכלול כל עיר בדיוק פעם אחת.

הנחות על הקלט וחריגות:
- הניחו ש-map מייצג מטריצת משתנים תקינה עבור מופע של n ערים.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 16</u>(אין צעדים לא חוקיים)(5 נקודות)

בהינתן מופע של בעיית הטיול הגדול ומטריצת משתני CNF המתאימה למופע זה, נבנה נוסחת CNF המתאימה לאילוץ "אין צעדים לא חוקיים". השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static int[][]
      noIllegalSteps(boolean[][] flights, int[][] map)
```

הפונקציה מקבלת מופע של הבעיה הנתון במערך הדו ממדי $flights$, וכן מטריצה של (שמות של) משתני CNF המייצגים את המשתנים $X_{i,j}$, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לקיים את התנאי: לכל שני ערים שונות j, k שאין ביניהם קו תעופה, אין מעבר מ j ל - k בפתרון (כולל החזרה מהעיר האחרונה לעיר המקור).

הנחות על הקלט וחריגות:
- הניחו ש- flights מייצג מופע תקין על n ערים.
- הניחו ש-map מייצג מטריצת משתנים תקינה עבור מופע של n ערים.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 17</u> (ממיר קלט)(5 נקודות)

בהינתן מופע של בעיית הטיול הגדול ומשתני CNF, נבנה נוסחת CNF שמקודדת את כל האילוצים, ונוסיף אותה לפותרן. השלימו את הגדרת הפונקציה הבאה בקובץ:

```
public static void encode(boolean[][] flights, int[][] map)
```

הפונקציה מקבלת מופע של הבעיה הנתון במערך הדו ממדי $flights$, וכן מטריצה map של (שמות של) משתני CNF המייצגים את המשתנים $X_{i,j}$, ומוסיפה לפותרן נוסחת CNF המקודדת את האילוצים של מופע נתון של בעיית הטיול הגדול:

1. בכל צעד מבקרים בעיר אחת.
2. מבקרים בכל עיר רק פעם אחת.
3. עיר המקור היא 0.
4. אין צעדים לא חוקיים.

הנחות על הקלט וחריגות:
- אין להניח שום הנחות על הקלט.
- יש לזרוק חריגה אם הקלט flights אינו מייצג מופע תקין לבעיית הטיול הגדול.
- יש לזרוק חריגה אם מטריצת המשתנים אינה מתאימה למופע.

<u>משימה 18</u> (ממיר פלט)(5 נקודות)

בהינתן השמה למשתני ה-CNF וטבלת המשתנים, ממיר הפלט (Decode) מפענח את ההשמה ומחשב פתרון למופע של בעיית הטיול הגדול.

השלימו את הפונקציה הבאה בקובץ:

```
public static int[] decode(boolean[] assignment, int[][] map)
```

הפונקציה מקבלת השמה assignment למשתנים בטבלת המשתנים map. על הפונקציה להחזיר מערך שמייצג פתרון למופע כך שהתא ה – i במערך התוצאה הוא מספר העיר שנבקר בה בצעד ה- i במסלול.

הנחות על הקלט וחריגות:

- הניחו ש map מייצג מטריצת משתנים תקינה עבור מופע של n ערים
- הניחו ש-assignment אינו null
- יש לזרוק חריגה אם assignment אינו מערך באורך $n^2 + 1$ (זכרו כי משתני ה-cnf ממוספרים מ-1 עד n^2 וההשמה למשתנה ה-0 אינה רלוונטית)

דוגמאות:

```
boolean[] assignment = {false,
    true, false, false, false,
    false, false, false, true,
    false, true, false, false,
    false, false, true, false};

int[][] map = {
    {1,2,3,4},
    {5,6,7,8},
    {9,10,11,12},
    {13,14,15,16}};

int[] tour = decode(assignment, map) // {0,3,1,2}
```

<u>משימה 19</u> (מציאת פתרון למופע) (5 נקודות)

לבסוף, אנו מוכנים לחבר את חלקי העבודה יחדיו ולכתוב פותרן לבעיית הטיול הגדול באמצעות רדוקציה לבעיית הספיקות הבוליאנית. השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static int[] solve(boolean[][] flights)
```

הפונקציה מקבלת כקלט מופע של בעיית הטיול הגדול. על הפונקציה:

- לייצר טבלת משתנים מתאימה למופע
- לאתחל פותרן לבעיית הספיקות
- לקודד את המופע, באמצעות טבלת המשתנים, לנוסחת CNF ולהוסיף את הפסוקיות לפותרן
- להפעיל את הפותרן
- אם מתקבלת השמה מספקת
  - יש לפענח את ההשמה המספקת לפתרון (נסמנו ב- s)
  - לוודא כי s הוא פתרון חוקי למופע שקיבלתם ולהחזיר תשובה בהתאם:
    - אם s הוא פתרון חוקי, יש להחזירו

- אחרת, יש לזרוק חריגה שמציינת שהפתרון אינו חוקי
- אם אין השמה מספקת יש להחזיר null

הנחות על הקלט וחריגות:
- אין להניח שום הנחות על הקלט
- יש לזרוק חריגה אם הקלט flights אינו מייצג מופע תקין לבעיית הטיול הגדול
- יש לזרוק חריגה אם המופע היה בלתי פתיר עקב מגבלות זמן (timeout), פרטים נוספים על מגבלות זמן ניתן למצוא בנספח לעבודה זו
- יש לזרוק חריגה במידה ויש השמה מספקת אך הפתרון המתקבל ממנה לא חוקי

<u>משימה 20</u> (קיום לפחות שני מסלולים) (5 נקודות)

בהינתן מופע של בעיית הטיול הגדול, נרצה לדעת האם קיימים למופע זה <u>שני</u> פתרונות שונים.

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean solve2(boolean[][] flights)
```

הפונקציה מקבלת כקלט מופע של בעיית הטיול הגדול flights, ומחזירה true אם ורק אם קיימים לפחות שני פתרונות שאינם שקולים לפי ההגדרה הבאה.

הגדרה 3 (שקילות פתרונות): עבור מופע של בעיית הטיול הגדול על n ערים , נאמר שפתרונות A, B שקולים אם $B = [A_0, A_{n-1}, A_{n-2}, \dots A_1]$.
דוגמא: הפתרונות [0,2,1,3] ו- [0,3,1,2] שקולים.

הנחות על הקלט וחריגות:
- אין להניח שום הנחות על הקלט
- יש לזרוק חריגה אם הקלט flights אינו מייצג מופע תקין לבעיית הטיול הגדול
- יש לזרוק חריגה אם המופע היה בלתי פתיר עקב מגבלות זמן (timeout), פרטים נוספים על מגבלות זמן ניתן למצוא בנספח לעבודה זו
- יש לזרוק חריגה במידה ויש השמה מספקת אך הפתרון המתקבל ממנה לא חוקי

בהצלחה!

נספח: שימוש בפותרן לבעיית הספיקות

יש להוריד את הקובץ SATSolver.java מאתר הקורס (בתיקייה של עבודת הבית מספר 2) ולמקמו במחשב באותה התיקייה יחד עם שאר קובצי הג'אווה של עבודת הבית. אין לשנות את הקובץ הזה ואין להגישו יחד עם קובצי המשימה. בקובץ נמצאים עיקרי הממשק לפותרן בעיית הסיפוק הבוליאני (SAT Solver). הפותרן מבוסס על פותרן שנקרא SAT4J. אם תרצו ללמוד יותר על פותרן זה, תוכלו להיעזר בגוגל.

כדי למצוא השמה מספקת לנוסחת CNF בעזרת הפותרן, יש לאתחל את הפותרן, להוסיף את הפסוקיות המהוות את הנוסחה ולבקש השמה מספקת (פתרון).

עיקרי הממשק של ה-SAT Solver:

- אתחול: יש לבצע קריאה לפונקציה SATSolver.init(int nVars) כאשר הערך במשתנה nVars מציין שמשתני ה-CNF שיופיעו בנוסחת ה-CNF יילקחו אך ורק מתוך הקבוצה $\{x_1, x_2, \dots, x_{nVars}\}$. למשל, לאחר אתחול הפותרן בקריאה: SATSolver.init(34), יהיה אפשר להתייחס רק למשתני CNF מתוך הקבוצה $\{x_1, \dots, x_{34}\}$.
- הוספת פסוקית: כדי להוסיף פסוקית בודדת לפותרן, יש לקרוא לפונקציה
  SATSolver.addClause(int[] clause)
  כאשר המערך clause מייצג פסוקית. למשל, שורות הקוד הבאות:

```
int[] clause = {5,2,-6,7,12};
SATSolver.addClause(clause);
```

  יוסיפו את הפסוקית $(x_5 \vee x_2 \vee \neg x_6 \vee x_7 \vee x_{12})$ לפותרן.

- הוספת פסוקיות: כדי להוסיף כמה פסוקיות לפותרן, יש לקרוא לפונקציה
  SATSolver.addClauses(int[][] clauses)
  כאשר המערך הדו-ממדי clauses מייצג את הפסוקיות. למשל, שורות הקוד הבאות:

```
int[][] clauses = { {5,-2,6}, {4,-17,99} };
SATSolver.addClauses(clauses);
```

  יוסיפו את הפסוקיות $(x_5 \vee \neg x_2 \vee x_6)$ ו- $(x_4 \vee \neg x_{17} \vee x_{99})$ לפותרן.

- מציאת השמה מספקת: כדי לפתור את נוסחת ה-CNF שהצטברה עד כה ב-SATSolver, יש לקרוא לפונקציה

```
SATSolver.getSolution()
```

פונקציה זו מחזירה ערך לפי אחת משלוש האפשרויות הבאות:

1. **מערך בוליאני שאינו ריק** - במידה שישנה השמה מספקת. אורך המערך יהיה כמספר המשתנים פלוס אחד. מערך זה מייצג השמה מספקת כפי שהוסבר במבוא לחלק 2 של העבודה, בסעיפי התזכורות.

2. **מערך בוליאני ריק** – במידה שהנוסחה אינה ספיקה (לא קיימת לה השמה מספקת).
3. **ערך null** – במידה שהפותרן לא מצא פתרון, עקב מגבלת זמן (timeout של 3 דקות).

דוגמאות:

1. התכנית הבאה מגדירה נוסחת CNF בעלת שלוש פסוקיות: $\left((x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3)\right)$,
מבקשת השמה מספקת מהפותרן, ומדפיסה פלט בהתאם לתוצאה: "SAT" אם הנוסחה מספקת, "TIMEOUT" אם הפותרן לא מצא פתרון עקב מגבלת זמן ו- "UNSAT" אם הנוסחא לא מספקת .

```
int nVars = 3;
SATSolver.init(nVars);

int[] clause = {1} ;
SATSolver.addClause(clause);

int[][] clauses = {{-1,-2}, {2,3}} ;
SATSolver.addClauses(clauses);

boolean[] assignment = SATSolver.getSolution() ;
if (assignment == null)
      System.out.println("TIMEOUT");

else if (assignment.length == nVars+1)
      System.out.println("SAT");
else
      System.out.println("UNSAT");
```

הפלט של תכנית זו הוא "SAT".

2. התכנית הבאה מגדירה נוסחת CNF בעלת ארבע פסוקיות:

$$\left((x_1) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_3)\right)$$

מבקשת השמה מספקת מהפותרן, ומדפיסה פלט בהתאם לתוצאה: "SAT" אם הנוסחה מספקת, "TIMEOUT" אם הפותרן לא מצא פתרון עקב מגבלת זמן ו- "UNSAT" אם הנוסחא לא מספקת .

```
int nVars = 3 ;
SATSolver.init(nVars);

int[] clause = {1} ;
SATSolver.addClause(clause);

int[][] clauses = {{-1,-2}, {2,3}, {-1,-3}} ;
SATSolver.addClauses(clauses);

boolean[] assignment = SATSolver.getSolution() ;
if (assignment == null)
      System.out.println("TIMEOUT");

else if (assignment.length == nVars+1)
      System.out.println("SAT");
else
      System.out.println("UNSAT");
```

הפלט הצפוי הוא "UNSAT".

3. הקובץ ExamplesSAT.java מכיל כמה דוגמאות נוספות של נוסחאות מסתפקות.

4. הקובץ ExamplesUNSAT.java מכיל כמה דוגמאות נוספות של נוסחאות שאינן מסתפקות.

כיצד לשלב את הפותרן בפרויקט אקליפס?

בתחילת העבודה מומלץ ליצור פרויקט java בסביבת אקליפס ולבצע את הפעולות הבאות:

1. להוסיף את כל קובצי הקוד המצורפים לעבודה לספריית הקוד של הפרויקט. ספריית הקוד בפרויקט אקליפס מקבלת את השם src כברירת מחדל.
2. שימו לב כי בקובצי הקוד שקיבלתם:
   a. ישנו קובץ שנקרא **org.sat4j.core.jar**. זהו הקובץ המכיל את הפותרן. אינכם צריכים לעבוד איתו ישירות, אבל צריך שיהיה בספריית הקוד שלכם.
   b. ישנו קובץ שנקרא **SATSolver.java**. זהו הקובץ בו נמצאות כל הפונקציות שאתם צריכים עבור העבודה עם הפותרן. פונקציות אלו מתוארות בסעיף הקודם "עיקרי הממשק של ה-SAT Solver".
3. כדי שיהיה אפשר לעבוד עם הפותרן, יש להוסיף אותו ל-Build Path של הפרויקט. למשל כך:
   a. באקליפס, לחצו עם המקש הימני של העכבר על הקובץ org.sat4j.core.jar שהוספתם לפרויקט.
   b. בחרו באפשרות "Build Path" ואז לחצו על האפשרות "Add to Build Path".
   c. כדי לוודא שהפותרן אכן משולב בפרויקט, תוכלו לכתוב פונקציית main עם קוד מאחת הדוגמאות שבסעיף הקודם ולוודא שהדוגמה אכן עובדת.