מבוא למדעי המחשב – סמסטר א' תשפ"ב

עבודת בית מספר 2: מערכים, פונקציות ובעיית הספיקות

צוות העבודה:

- מרצה אחראי: פרופ' מייק קודיש
- מתרגלים אחראים: אבי יצחקוב ולירן נחומסון

תאריך פרסום: 04/11/2021

מועד אחרון להגשה: 28/11/2021 בשעה 23:59

מה בתגבור: בתגבורים של 8-11/11/21 נפתור באופן מודרך את משימות: 2,8,10 ונדגים את השימוש בפותרן ה-SAT.

תקציר נושא העבודה:

בעבודת בית זו נתרגל עבודה עם מערכים ופונקציות ב-Java 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" (SAT).

בעבודה זו 20 משימות וסך הנקודות המקסימלי הוא 100. שימו לב שמספר הנקודות לכל משימה הוא אחיד (5 נקודות למשימה) ואינו מצביע על קושי המשימה.

הנחיות מקדימות

- קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. רמת הקושי של המשימות אינה אחידה: הפתרון של חלק מהמשימות קל יותר, ואחרות מצריכות מחשבה וחקירה מתמטית - שאותה תוכלו לבצע בעזרת מקורות דרך רשת האינטרנט .בתשובות שבהן אתם מסתמכים על עובדות מתמטיות שלא הוצגו בשיעורים או כל חומר אחר, יש להוסיף כהערה במקום המתאים בקוד את ציטוט העובדה המתמטית ואת המקור (כגון ספר או אתר).
- עבודה זו תוגש ביחידים במערכת המודל ניתן לצפות בסרטון הדרכה על אופן הגשת העבודה במערכת ה-VPL בלינק הבא: סרטון הדרכה.
- לצורך העבודה מסופקים לכם שלושה קבצים:
  1. קובץ Assignment2.java שכולל את התבניות להגדרת הפונקציות עבור המשימות בעבודה.

  קבצים לעבודה עם הפותרן (ניתן להוריד קבצים אלו מהמודל ליד קובץ תיאור העבודה):
  2. SATSolver.java – ממשק לעבודה עם פותרן SAT.
  3. org.sat4j.core.jar – ספרייה הממשת פותרן SAT.

המלצה על דרך העבודה- אנו ממליצים לפתוח פרויקט ב-eclipse בשם Assignment2.
הוסיפו את קובץ השלד Assignment2.java שהורדתם לפרויקט, בקובץ זה אתם תשלימו את הגדרת הפונקציות בהתאם למתואר במשימות. בנוסף יש לשלב את קבצי הפותרן בפרויקט בהתאם להסבר במסמך הנלווה "שימוש בפותרן SAT" הניתן להורדה במודל ליד קובץ תיאור העבודה.

הנחיות לכתיבת קוד והגשה

- בקבצי השלד המסופקים לכם קיים מימוש ברירת מחדל לכל פונקציה. יש למחוק את מימוש ברירת המחדל בגוף הפונקציות ולכתוב במקום זאת את המימוש שלכם לפי הנדרש בכל משימה.
- אין לשנות את החתימות של הפונקציות המופיעות בקבצי השלד.
- עבודות שלא יעברו קומפילציה במערכת יקבלו את **הציון 0** ללא אפשרות לערער על כך. אחריותכם לוודא שהעבודה שאתם מגישים עוברת תהליך קומפילציה במערכת (ולא רק ב-**eclipse**). להזכירכם, תוכלו לבדוק זאת ע"י לחיצה על כפתור ה-**Evaluate**.
- עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק.
- סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד יעיל, 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 מיותרות, 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.
- בעבודה זו ניתן להשתמש בידע שנרכש בקורס עד לתאריך פרסום העבודה. אין להשתמש בכל צורת קוד אחרת אשר לא נלמדה בכיתה.

עזרה והנחיה

- 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.
- ניתן להיעזר בפורום. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך. שימו לב, **אין לפרסם פתרונות בפורום.**
- בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.
- אנחנו ממליצים בחום להעלות פתרון למערכת המודל לאחר כל סעיף שפתרתם. הבדיקה תתבצע על הגרסה האחרונה שהועלתה (בלבד!).

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.

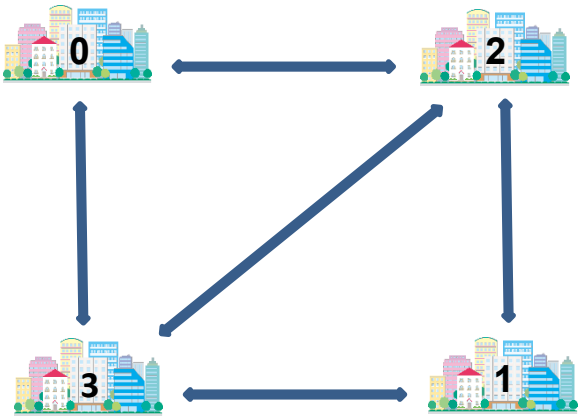
בעיית הטיול הגדול

מבוא

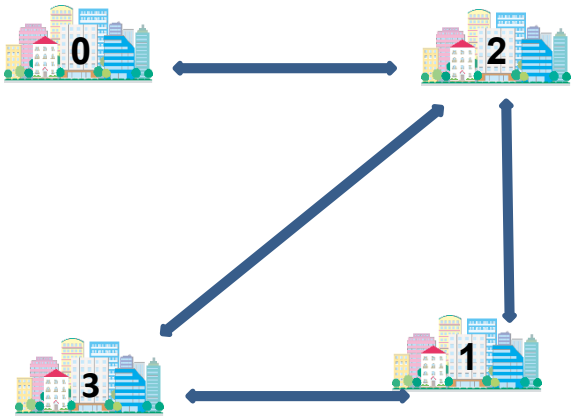
בהינתן רשימה של קווי תעופה בין n ערים $(n > 0)$ הממוספרות ב- $\{0 \dots n-1\}$, יש לתכנן מסלול המתחיל מנמל הבית הנמצא בעיר המקור 0, מבקר בכל אחת מן הערים האחרות $\{1 \dots n-1\}$ בדיוק פעם אחת ובסיומו חוזר לנמל הבית שבעיר המקור (מסלול מעגלי). קיומו של קו תעופה $\{i,j\}$ בין שני ערים שונות i,j , מציין 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: $0 \to 2 \to 1 \to 3 \to 0$.

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 $(n > 0)$ 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 על n ערים $(n > 0)$ 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 $(n > 0)$ 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 $(n > 0)$ אם:
א. A הוא מערך באורך n המכיל את כל המספרים 0…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.

<u>משימה 1:</u> <u>(מטריצה ריבועית) (5 נקודות)</u>

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת כקלט מערך בוליאני דו-ממדי באורך כלשהו ומוודאת כי המערך מייצג מטריצה ריבועית: על הפונקציה להחזיר ערך true אם ורק אם המערך matrix מכיל n מערכים באורך n $(n > 0)$.

הנחות על הקלט וחריגות:
- אין להניח שום הנחות על הקלט.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

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

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

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

<u>משימה 2:</u> <u>(מטריצה סימטרית) (5 נקודות)</u> [משימה זו תיפתר בתגבור השבועי].

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת כקלט מטריצה בוליאנית בגודל $n \times n$, ומוודאת כי המטריצה היא סימטרית: על הפונקציה להחזיר ערך true אם ורק אם לכל $0 \le i < j < n$ 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>

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת כקלט מטריצה בוליאנית בגודל $n \times n$, ומחזירה true אם ורק אם המטריצה היא אנטי-רפלקסיבית

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>

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

אשר מקבלת כקלט מערך דו-ממדי *matrix*, ובודקת האם הקלט הוא מופע תקין לבעיית הטיול הגדול 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 כל 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 ערך 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
```

משימה 6: (כל הטיסות במסלול קיימות) (5 נקודות)

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך flights דו-ממדי בגודל $n \times n$ המייצג מופע של בעיית הטיול הגדול ומערך *tour* באורך n המייצג מסלול. הפונקציה תחזיר ערך true אם ורק אם בין כל שני ערים עוקבות במסלול קיימת טיסה. באופן פורמלי, הפונקציה תחזיר ערך true 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 $n > 0$ ערים.
- המערך tour הוא באורך n וערכיו הם מהתחום $[0, n-1]$.
- פונקציה זו לא זורקת חריגות.

משימה 7: (פתרון חוקי) (5 נקודות)

השלימו את הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך flights דו-ממדי המייצג מופע של בעיית הטיול הגדול ומערך *tour*. הפונקציה תחזיר ערך true אם ורק אם המערך tour מקיים את התנאים לפתרון לפי **הגדרה 2** עבור המופע הנתון flights.

הנחות על הקלט וחריגות:

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

בעיית הספיקות

משימות 8-11: בדוגמאות של משימות 8-11 נתייחס להגדרות:

```
int[] lits1 = {1,2,3}; // {x1,x2,x3}
int[] lits2 = {-1,2,3}; // {¬x1,x2,x3}

boolean[] assign1 = {false, false, false, false};
boolean[] assign2 = {false, false, true, true};
boolean[] assign3 = {false, true, false, false};
```

משימה 8: (שיערוך נוסחת CNF בהינתן השמה) [משימה זו תיפתר בתגבור השבועי].

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

```
public static boolean evaluate(int[][] cnf, boolean[] assign)
```

הפונקציה מקבלת נוסחת CNF והשמה למשתנים שמופיעים בו, ומחזירה ערך `true` אם ורק אם ההשמה מספקת את הנוסחה.

הנחות על הקלט וחריגות:

- הניחו ש- cnf מייצג נוסחת CNF תקינה.
- הניחו שהמערך assign מייצג השמה תקינה.
- הניחו שעבור כל משתנה k שמופיע בנוסחה, assign[k] מייצג את ההשמה למשתנה זה.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
int[][] cnf = {{1,-2}, {-1, 2, 3}, {-3, 2}};

System.out.println(evaluate(cnf, assign1));  //true
System.out.println(evaluate(cnf, assign2));  //false
```

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

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מערך `lits` של ליטרלים (מספרים שלמים שמייצגים משתנים פסוקיים או שלילתם) ומחזירה נוסחת CNF שמאלצת את הליטרלים כך **שלפחות אחד** מהם מקבל ערך `true`. כלומר, ההשמות המספקות של הנוסחה המוחזרת הן בדיוק ההשמות שמספקות לפחות אחד מהליטרלים.

הנחות על הקלט וחריגות:

- הניחו שהמערך lits אינו null, אורכו גדול מ-0, ומכיל מספרים שלמים השונים מ-0.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
int[][] cnf1 = atLeastOne(lits1);
int[][] cnf2 = atLeastOne(lits2);

System.out.println(evaluate(cnf1, assign1)); // false
System.out.println(evaluate(cnf1, assign2)); // true
System.out.println(evaluate(cnf1, assign3)); // true
System.out.println(evaluate(cnf2, assign1)); // true
System.out.println(evaluate(cnf2, assign2)); // true
System.out.println(evaluate(cnf2, assign3)); // false
```

<u>משימה 10</u>: <u>(לכל היותר אחד)(5 נקודות)</u> [משימה זו תיפתר בתגבור השבועי].

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מערך `lits` של ליטרלים (מספרים שלמים שמייצגים משתנים פסוקיים או שלילתם) ומחזירה נוסחת CNF שמאלצת את הליטרלים כך **שלכל היותר אחד** מהם מקבל ערך `true`. כלומר, ההשמות המספקות של הנוסחה המוחזרת הן בדיוק ההשמות שמספקות לכל היותר אחד מהליטרלים.

הנחות על הקלט וחריגות:

- הניחו שהמערך lits אינו null, אורכו גדול מ-0, ומכיל מספרים שלמים השונים מ-0.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
int[][] cnf1 = atMostOne(lits1);
int[][] cnf2 = atMostOne(lits2);

System.out.println(evaluate(cnf1, assign1)); // true
System.out.println(evaluate(cnf1, assign2)); // false
System.out.println(evaluate(cnf1, assign3)); // true
System.out.println(evaluate(cnf2, assign1)); // true
System.out.println(evaluate(cnf2, assign2)); // false
System.out.println(evaluate(cnf2, assign3)); // true
```

משימה 11: (בדיוק אחד) (5 נקודות)

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מערך `lits` של ליטרלים (מספרים שלמים שמייצגים משתנים פסוקיים או שלילתם) ומחזירה נוסחת CNF שמאלצת את הליטרלים כך **שבדיוק אחד** מהם מקבל ערך `true`. כלומר, ההשמות המספקות של הנוסחה המוחזרת הן ההשמות שמספקות בדיוק אחד מהליטרלים.

הנחות על הקלט וחריגות:

- הניחו שהמערך lits אינו null, אורכו גדול מ-0, ומכיל מספרים שלמים השונים מ-0.
- פונקציה זו לא זורקת חריגות.

דוגמאות:

```
int[][] cnf1 = exactlyOne(lits1);
int[][] cnf2 = exactlyOne(lits2);

System.out.println(evaluate(cnf1, assign1)); // false
System.out.println(evaluate(cnf1, assign2)); // false
System.out.println(evaluate(cnf1, assign3)); // true
System.out.println(evaluate(cnf2, assign1)); // true
System.out.println(evaluate(cnf2, assign2)); // false
System.out.println(evaluate(cnf2, assign3)); // false
```

רדוקציה מבעיית הטיול הגדול לבעיית הספיקות

תזכורת: 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.

תיאור הרדוקציה - משתנים:

בהינתן מופע של בעיית הטיול הגדול על n ערים, נגדיר:

עבור כל זוג מספרים (i,j) כך ש-$0 \le i,j < n$ כאשר i מציין שלב בטיול ו- j מציין עיר, נגדיר משתנה פסוקי x_k שהערך שלו מציין האם בשלב ה-i של הטיול ביקרנו בעיר ה-j. בסך הכל עבור מופע על n ערים יהיו לנו n^2 משתנים פסוקיים $x_1, \dots, x_{n^2}$ המתאימים ל-n^2 זוגות המספרים (i,j) האפשריים.

לדוגמה כאשר $n = 3$, נוכל למפות את הזוגות באופן הבא:

$$(0,0) \mapsto 1, (0,1) \mapsto 2, (0,2) \mapsto 3, (1,0) \mapsto 4, (1,1) \mapsto 5, (\mathbf{1},\mathbf{2}) \mapsto \mathbf{6}, (2,0) \mapsto 7, (2,1) \mapsto 8, (2,2) \mapsto 9$$

לפי מיפוי זה, הערך שיקבל המשתנה הפסוקי x_6 מציין האם בשלב 1 במסלול ביקרנו בעיר 2.

בהגדרת הרדוקציה נצטרך למפות את זוגות המספרים $\{(i,j)|0 \le i,j < n\}$ למשתנים הפסוקיים $\{x_1, \dots x_{n^2}\}$ באופן חד-חד ערכי ועל. נעשה זאת בעזרת פונקציית מיפוי, $map : \{(i,j)|0 \le i,j < n\} \to \{1,2, \dots n^2\}$ כך שהערך $map(i,j) = k$ אם ורק אם המשתנה הפסוקי x_k מציין האם בשלב ה-i של הטיול ביקרנו בעיר ה-j.

<u>משימה 12:</u> <u>(מיפוי המשתנים) (5 נקודות)</u>

א. השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

```
public static int map(int i, int j, int n)
```

הפונקציה מקבלת זוג מספרים $0 \le i,j < n$ ואת מספר הערים n. הפונקציה מחזירה ערך ייחודי $1 \le k \le n^2$ כך שהמשתנה הפסוקי x_k מציין האם בצעד ה-i ביקרנו בעיר j.

הנחות על הקלט וחריגות:
- הניחו כי $0 \le i,j < n$, וכן ש $n > 0$.
- פונקציה זו לא זורקת חריגות.

ב. השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

```
public static int[] reverseMap(int k, int n)
```

הפונקציה מקבלת מספר k ואת מספר הערים n במופע. הפונקציה מחזירה את הזוג (i,j) המיוצג כמערך כך ש-$map(i,j) = k$. אם נבצע את השורות הבאות:

```
int i = 1, j = 2, n = 3;
int k = map(i,j,n);
int[] pair = reverseMap(k,n);
for(int index: pair)
     System.out.print(index + " "); // 1 2
System.out.println();
```

הנחות על הקלט וחריגות:
- הניחו כי $1 \le k \le n^2$ וכן ש $n > 0$.
- פונקציה זו לא זורקת חריגות.

<u>תיאור הרדוקציה - אילוצים:</u>

בהינתן מופע של בעיית הטיול הגדול על n ערים, נגדיר נוסחת CNF על המשתנים הפסוקיים $x_1, \dots, x_{n^2}$ כך שמכל השמה מספקת של הנוסחה נוכל לפענח טיול שהוא פתרון למופע. את נוסחת ה- CNF נבנה בחלקים כקוניונקציה של

אילוצים. בהמשך השאלה, לצורך נוחות, נכתוב $map(i,j,n)$ במקום המשתנה הפסוקי $x_{map(i,j,n)}$.
כעת נפרט את האילוצים על קבוצת המשתנים הפסוקיים $\{map(i,j,n) | 0 \leq i,j < n\}$ ונממש פונקציות שמייצרות נוסחאות CNF שמתארות כל אילוץ.

<u>משימה 13:</u> <u>(אילוץ א: בכל צעד של הטיול מבקרים בעיר אחת) (5 נקודות)</u>

תיאור האילוץ: לכל שלב i בטיול, בדיוק אחד מהמשתנים $map(i,0,n), \ \ldots, map(i,n-1,n)$ מקבל את הערך $true$. כלומר, לכל שלב i קיימת בדיוק עיר אחת j שעבורה המשתנה $map(i,j,n)$ מקבל את הערך $true$ בהשמה מספקת.

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת את מספר הערים במופע n, ומחזירה נוסחת CNF המבטאת את האילוץ.

הנחות על הקלט וחריגות:
- יש להניח ש - $n > 0$.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 14:</u> <u>(אילוץ ב: מבקרים בכל עיר רק פעם אחת) (5 נקודות)</u>

תיאור האילוץ: לכל עיר j בדיוק אחד מהמשתנים $map(0,j,n), \ldots, map(n-1,j,n)$ מקבל את הערך $true$.
כלומר, לכל עיר j קיים בדיוק שלב אחד i שעבורו המשתנה $map(i,j,n)$ מקבל את הערך $true$ בהשמה מספקת.

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת את מספר הערים במופע n, ומחזירה נוסחת CNF המבטאת את האילוץ.

הנחות על הקלט וחריגות:
- יש להניח ש - $n > 0$.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 15:</u> <u>(אילוץ ג: עיר המקור היא 0) (5 נקודות)</u>

תיאור האילוץ: המשתנה $map(0,0,n)$ מקבל את הערך $true$ בכל השמה מספקת.

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת את מספר הערים במופע n, ומחזירה נוסחת CNF המבטאת את אילוץ ג.

הנחות על הקלט וחריגות:
- יש להניח ש - $n > 0$.
- פונקציה זו אינה זורקת חריגות.

משימה 16: (אילוץ ד: אין מעברים לא חוקיים במסלול) (5 נקודות)

תיאור האילוץ: לכל שלב $0 \leq i < n$ במסלול, ושתי ערים שונות j, k שאין ביניהן קו תעופה, נדרוש שאם בשלב ה - i מבקרים בעיר j אז בשלב הבא לא מבקרים בעיר k. לכל שלב i, אם $i < n - 1$ אז השלב הבא הוא השלב ה- $i + 1$ ואם $i = n - 1$ אז השלב הבא הוא השלב ה- 0 (צעד חזרה לעיר המקור). כלומר, לכל היותר אחד משני המשתנים $map(i, j, n),\ map((i + 1)\%n, k, n)$ יכול לקבל ערך $true$.

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מופע של הבעיה הנתון במערך הדו-ממדי $flights$ ומחזירה נוסחת CNF המבטאת את האילוץ.
רמז: התמקדו בכל זוגות הערים השונות j, k שאין ביניהן קו תעופה על פי המופע הנתון.

הנחות על הקלט וחריגות:
- הניחו ש- $flights$ מייצג מופע תקין לפי הגדרה 1 ושמספר הערים $n > 0$.
- פונקציה זו אינה זורקת חריגות.

משימה 17: (ממיר קלט)(5 נקודות)

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מופע נתון של בעיית הטיול הגדול במערך הדו ממדי $flights$ ומחזירה נוסחת CNF שהיא קוניונקציה של כל האילוצים של בעיית הטיול הגדול עבור המופע הנתון $flights$:

- אילוץ א: בכל צעד מבקרים בעיר אחת.
- אילוץ ב: מבקרים בכל עיר רק פעם אחת.
- אילוץ ג: עיר המקור היא 0.
- אילוץ ד: אין צעדים לא חוקיים.

הנחות על הקלט וחריגות:
- הניחו שהמערך $flights$ מייצג מופע תקין לפי הגדרה 1.

משימה 18: (ממיר פלט)(5 נקודות)

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת מספר n ומערך בוליאני assignment באורך $n^2 + 1$ המייצג השמה לקבוצת המשתנים הפסוקיים $\{map(i, j, n) | 0 \leq i, j < n\}$. הפונקציה מחזירה מערך tour של מספרים שלמים באורך n המקיים את התנאי הבא:
לכל $0 \leq i < n$ אם הערך $tour[i] = j$ אז $assignment[map(i, j, n)] = true$.
שימו לב, אין דרישה שהמערך tour ייצג בהכרח מסלול שמבקר בכל הערים.

הנחות על הקלט וחריגות:
- הניחו ש $n > 0$.

- יש לזרוק חריגה אם assignment הוא null.
- יש לזרוק חריגה אם assignment אינו מערך באורך $n^2 + 1$.

דוגמא:

נניח ש $n = 3$ ושהפונקציה map ממפה את הזוגות (i, j) באופן הבא:

$$(0,0) \mapsto 1, (0,1) \mapsto 2, (0,2) \mapsto 3, (1,0) \mapsto 4, (1,1) \mapsto 5, (1,2) \mapsto 6, (2,0) \mapsto 7, (2,1) \mapsto 8, (2,2) \mapsto 9$$

בהינתן ההשמה הבאה:

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

הקריאה לפונקציה decode תחזיר את המערך שמצוין מימין.

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

במשימה הבאה נחבר את חלקי העבודה יחדיו לפתור את בעיית הטיול הגדול באמצעות רדוקציה לבעיית הספיקות הבוליאנית תוך שימוש בפותרן של SAT.

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

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

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

- לאתחל פותרן לבעיית הספיקות עם כמות המשתנים המתאימה למופע
- לקודד את המופע לנוסחת CNF בעזרת הפונקציה encode
- להוסיף את הנוסחה לפותרן
- להפעיל את הפותרן על מנת למצוא השמה מספקת לנוסחה
- אם מתקבלת השמה מספקת
  - יש לפענח השמה זו לפתרון (נסמנו ב- tour) בעזרת הפונקציה decode.
  - יש לוודא כי tour הוא פתרון חוקי למופע הנתון ב- $flights$ בעזרת הפונקציה isSolution ולהחזיר תשובה בהתאם:
    - אם tour הוא פתרון חוקי, יש להחזירו

    אחרת יש לזרוק חריגה שמציינת שהפתרון אינו חוקי

  אחרת (אין השמה מספקת) יש להחזיר null

הנחות על הקלט וחריגות:

- אין להניח שום הנחות על הקלט
- יש לזרוק חריגה אם $flights$ אינו מייצג מופע תקין לבעיית הטיול הגדול לפי הגדרה 1.
- 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 אם יש השמה מספקת אך הפתרון המתקבל ממנה לא חוקי

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

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

השלימו בקובץ Assignment2.java את הגדרת הפונקציה:

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

הפונקציה מקבלת כקלט מופע של בעיית הטיול הגדול $flights$, ומחזירה ערך $true$ אם ורק אם קיימים לפחות שני פתרונות לבעיית הטיול הגדול שמתחילים בעיר 0 ושאינם מכילים בדיוק את אותם הטיסות בכיוון הנגדי.

לדוגמא: המסלול [0,2,1,3] מכיל את אותן הטיסות כמו במסלול [0,3,1,2] בכיוון הנגדי.

למשל עבור המופע הנתון באיור 1, הפונקציה תחזיר ערך $false$.

הנחות על הקלט וחריגות:

- אין להניח שום הנחות על הקלט
- יש לזרוק חריגה אם $flights$ אינו מייצג מופע תקין לבעיית הטיול הגדול לפי הגדרה 1.
- יש לזרוק חריגה אם המופע היה בלתי פתיר עקב מגבלות זמן (timeout), פרטים נוספים על מגבלות זמן ניתן למצוא בנספח לעבודה זו

בהצלחה!