מבוא למדעי המחשב - סמסטר א' תש"פ
עבודת בית מספר 2

צוות התרגיל: לינור חזן, אבי יצחקוב, מייק קודיש
תאריך פרסום: 13.11.19
תאריך הגשה: 06.12.19, 12:00 בצהריים

הקדמה

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 המקשרים ביניהן. יש לתכנן מסלול לטיול שיוצא מעיר כלשהי, עובר בכל שאר הערים וחוזר לעיר המקור, כך שכל עיר תופיע במסלול בדיוק פעם אחת ובסיומו נחזור לעיר המקור.

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

בעבודה זו 16 משימות וסך הנקודות המקסימלי הוא 100. הניקוד לכל משימה מפורט במסמך.
בעבודה זו **מותר להשתמש בידע שנלמד עד הרצאה** 8 **(כולל), וכן עד תרגול** 4 **(כולל).**

הוראות מקדימות

הערות כלליות

1. **קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה**. ודאו שאתם מבינים את כל המשימות.
2. עבודה זו תוגש ביחידים. על מנת 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. בכל משימה מורכבת יש לשקול כיצד לחלק את המשימה לתתי-משימות ולהגדיר פונקציות עזר בהתאם.
4. בכל הסעיפים אפשר ומומלץ להשתמש בפונקציות מסעיפים קודמים.

קבצים

5. 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. נוסף על כך, הקובץ שתגישו יכול להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכדומה) לא יתקבל. הקפידו לא להשאיר בהגשה חלקי קוד שאינם חלק מהתכנית (לדוגמה, בדיקות שכתבתם עבור עצמכם).
6. בנוסף מצורפים:
   - קובץ בדיקות Tests.java
   - קבצי דוגמאות לנוסחאות cnf: ExamplesSAT.java , ExamplesUNSAT.java.
   - קובץ המכיל דוגמאות למופעים של בעיית הטיול הגדול: ExamplesFlights.java
   - שני קבצים: SATSolver.java ו – org.sat4j.core.jar שמאפשרים עבודה עם פותרן לבעיית הספיקות

הקבצים הנ"ל הם קובצי עזר עבורכם **<u>ואינם מיועדים להגשה</u>**. חשוב להדגיש כי קובץ הבדיקות Tests.java שסופק לכם מכיל בדיקות חלקיות של חלק מהמשימות. אנו מעודדים אתכם להרחיב קובץ זה ולהשתמש בו כדי לוודא את נכונות המימוש שלכם.

7. **קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו**. את קובץ ה־ZIP יש להגיש ב־ Submission System. פרטים לגבי ההרשמה ואיך להגיש את העבודה תוכלו למצוא באתר.

<u>בדיקת עבודות הבית</u>

8. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. הבדיקה האוטומטית מתייחסת אך ורק לפלט המוחזר מהפונקציות.
9. סגנון כתיבת הקוד ייבדק באופן ידני. יש להקפיד על כתיבת קוד יעיל וברור, 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. יש לתכנן את הקוד בצורה נכונה כך שמשימות מורכבות יחולקו לתתי משימות המבוצעות על ידי פונקציות עזר. כתיבת קוד שאינה עומדת בדרישות אלו תגרור הפחתה בציון העבודה.

<u>עזרה והנחיה</u>

10. 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. כמו כן, אתם יכולים להיעזר בפורום ולפנות בשאלות לחבריכם לכיתה. צוות הקורס עובר על השאלות ונותן מענה במקרה הצורך.
11. **בתגבור** של השבוע (20.11.19-17.11.19) נפתור באופן מודרך את משימות 1, 4, 5.
12. בכל בעיה **אישית** הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.
13. אנחנו ממליצים בחום להעלות פתרון למערכת ההגשה לאחר כל סעיף שפתרתם. הבדיקה תתבצע על הגרסה האחרונה שהועלתה (בלבד!).

<u>יושר אקדמי</u>

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 ב<u>סילבוס הקורס</u> 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**.

<u>משימה 0: הצהרה (0 נקודות)</u>

פתחו את הקובץ Assignment2.java **וכיתבו בראשו את שמכם ואת מספר תעודת הזהות שלכם**. משמעות פעולה זו היא שאתם מסכימים על הכתוב בו. דוגמה:

I, Israel Israeli (123456789), assert that the work I submitted is entirely my own.
I have not received any part from any other person, nor did I give parts of it for use to others.
I realize that if my work is found to contain code that is not originally my own, a formal complaint will be opened against me with the BGU disciplinary committee.

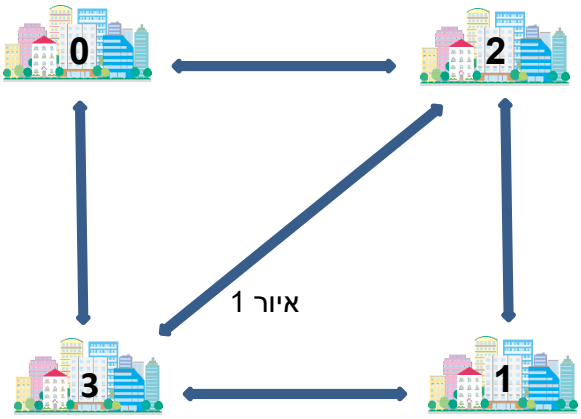
בעיית הטיול הגדול

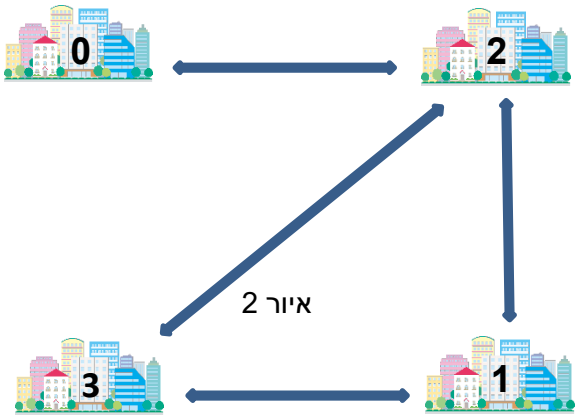
מבוא

בהינתן רשימה של קווי תעופה בין n ערים הממוספרות ב־$\{0,..,n-1\}$, יש לתכנן מסלול המתחיל באחת הערים (עיר המקור), מבקר בכל אחת מן הערים האחרות בדיוק פעם אחת ובסיומו חוזר לעיר המקור (מסלול מעגלי). קיומו של קו תעופה $\{i,j\}$ בין שתי ערים שונות i,j מציין שקיימת טיסה בשני הכיוונים ($i \to j, j \to i$).

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

דוגמה *2*: נניח שיש *4* ערים {0,1,2,3} ושקווי התעופה הם: $\{0,2\},\{1,2\},\{2,3\},\{1,3\}$, כפי שמוצג באיור מספר *2*. בדוגמה זו לא ניתן למצוא מסלול שפותר את בעיית הטיול הגדול.





ייצוג מופע של בעיית הטיול הגדול ב־Java

מופע של בעיית *הטיול הגדול* עבור n ערים מיוצג באמצעות מערך flights [][]int של n מערכים של מספרים, כאשר המערך flights[i] מציין את הערים אליהן ניתן לטוס מהעיר i.

הגדרה 1: מערך דו־ממדי flights מייצג מופע חוקי של בעיית הטיול הגדול על n ערים אם:

א. המערך flights הוא באורך n>1 ובפרט אינו null.
ב. כל מערך flights[i] מכיל ערכים בין 0 ל־n-1 ובפרט אינו null.
ג. לכל $0 \le i < j < n$, flights[i] מכיל את הערך j אם ורק אם המערך flights[j] מכיל את הערך i.
ד. לכל $0 \le i < n$, המערך flights[i] אינו מכיל את הערך i.

דוגמה: המופע של בעיית הטיול הגדול המוצג באיור מספר 1 ייוצג ב־Java באופן הבא:

```
int[][] flights = {{2,3}, {2,3}, {0,1,3}, {0,1,2}};
```

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.

דוגמה: המסלול $2 \to 0 \to 3 \to 1 \to 2$ המהווה פתרון עבור המופע המתואר באיור מספר 1, מיוצג באמצעות המערך

```
int[] solution = {2,1,3,0};
```

נשים לב שהחזרה לעיר המקור (העיר 2) בסוף הטיול אינה מיוצגת באופן מפורש במערך.

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

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

וידוא תקינות קלט

במשימות הבאות נממש כמה פונקציות שיסייעו לנו לבדוק אם מופע נתון לבעיית הטיול הגדול הוא תקין.

משימה 1 (קיום טיסה) (5 נקודות):

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean hasFlight(int[][] flights, int i, int j)
```

אשר בהינתן מופע int[][] flights של בעיית הטיול הגדול על n ערים ושתי ערים $0 \le i,j < n$ מחזירה ערך true אם קיימת טיסה בין העיר i לעיר j. אחרת הפונקציה מחזירה ערך false.
לדוגמה, קטע הקוד הבא

```
int[][] flights = {{2,3}, {2,3}, {0,1,3}, {0,1,2}};
System.out.println(hasFlight(flights,1,2));
System.out.println(hasFlight(flights,0,1));
System.out.println(hasFlight(flights,1,1));
```

ידפיס את שלוש השורות הבאות:

```
true
false
false
```

במשימה זו, יש להניח שהקלט תקין. כלומר: ש־flights הוא מופע חוקי לבעיית הטיול הגדול על n ערים (לפי הגדרה 1) וששתי הערים i,j הן ערכים $0 \le i,j < n$.

<u>משימה 2 (מופע חוקי) (10 נקודות):</u>

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

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

הפונקציה תחזיר ערך true אם ורק אם המערך הדו־ממדי flights מקיים את התנאים של מופע תקין לפי **הגדרה 1**.
לדוגמה, קטע הקוד הבא

```
int[][] flights1 = {{2,3}, {2,3}, {0,1,3}, {0,1,2}};
int[][] flights2 = {{2,3}, {2,3}, {0,3}, {0,1,2}};
int[][] flights3 = {{2,3}, {1,3}, {0,1,3}, {0,2}};
int[][] flights4 = {{2,3}, {1,2,3}, {0,1,3}, {0,1,2}};
System.out.println(isLegalInstance(flights1));
System.out.println(isLegalInstance(flights2));
System.out.println(isLegalInstance(flights3));
System.out.println(isLegalInstance(flights4));
```

ידפיס את ארבע השורות הבאות:

```
true
false
false
false
```

הנחות על הקלט וחריגות:

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

וידוא פתרון לבעיית הטיול הגדול

משימה 3 (פתרון חוקי) (10 נקודות):

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

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

הפונקציה תחזיר ערך true אם ורק אם המערך tour מקיים את התנאים לפתרון לפי **הגדרה 2** עבור המופע flights. לדוגמה, קטע הקוד הבא מתאר מספר קלטים אפשריים ואת התוצאה עבורם. חלק מהקלטים כוללים מערכים שאינם מייצגים טיול באופן חוקי (למשל מכילים עיר שאינה בטווח המתאים) ועבורם יש לזרוק חריגה.

```
int[][] flights = {{2,3}, {2,3}, {0,1,3}, {0,1,2}};
int[] tour1 = {2,0,3,1};
int[] tour2 = {1,0,4,3};
int[] tour3 = {1,3,2,0};
int[] tour4 = {1,3,2,0,4};
int[] tour5 = {0,3,2};
System.out.println(isSolution(flights, tour1));
System.out.println(isSolution(flights, tour2));
System.out.println(isSolution(flights, tour3));
System.out.println(isSolution(flights, tour4));
System.out.println(isSolution(flights, tour5));
```

כשנריץ את קטע הקוד יודפסו חמש השורות הבאות:

```
true
IllegalArgumentException
false
IllegalArgumentException
IllegalArgumentException
```

הנחות על הקלט וחריגות:

- הניחו שהמערך flights מייצג מופע תקין על $n > 1$ ערים.
- אין להניח שום הנחות על המערך tour.
- פונקציה זו זורקת חריגת IllegalArgumentException אם tour אינו מערך באורך n.
- פונקציה זו זורקת חריגת IllegalArgumentException אם tour מכיל מספרים מחוץ לתחום שבין 0 ל-n-1.

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. כדי להימנע מבלבול בין המשתנים של ג'אווה לבין אלו של ה־CNF, למשתנים של ה־CNF נקרא 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 $-i$.
- נייצג פסוקית 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 באמצעות המערך:

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

- נוסחת CNF תיוצג 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 תואמת 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- $\{x = false, y = false\}$.

ייצוג מספרים בעזרת משתנים בוליאנים

דרך נפוצה לייצג מספרים בעזרת משתנים בוליאנים היא כזו: מספר שלם בתחום $0 \dots n$ מיוצג בעזרת סדרה של $n + 1$ משתנים בוליאנים $< x_{i_0}, x_{i_1}, \dots, x_{i_n} >$ כאשר המשתנה ה-k בסדרה (המשתנה x_{i_k}) מקבל את הערך true אם ורק אם הערך של המספר המיוצג על ידי סדרת המשתנים הוא k. נשים לב שבייצוג זה עלינו להבטיח <u>שבדיוק אחד</u> מהמשתנים הבוליאנים בסדרה מקבל את הערך true.

דוגמה: מספר לא ידוע I בתחום 0..2 ניתן לייצג בעזרת סדרה של שלושה משתנים כלשהם, למשל $< x_{17}, x_9, x_{31} >$. כאשר I שווה ל-0 אם המתשנה הראשון בסדרה x_{17} מקבל את הערך true, I שווה ל1 אם המשתנה השני בסדרה x_9 מקבל את הערך true, ו-I שווה ל-2 אם המשתנה השלישי בסדרה x_{31} מקבל את הערך true. הנוסחה הבאה מאלצת כל השמה מספקת לתת ערך true לבדיוק אחד מבין שלושת המשתנים.

$x_{17} \vee x_9 \vee x_{31}$
$\neg x_{17} \vee \neg x_9$
$\neg x_{17} \vee \neg x_{31}$
$\neg x_9 \vee \neg x_{31}$

ההשמות המספקות לנוסחה הן:

$[x_{17} = T, x_9 = F, x_{31} = F]$
$[x_{17} = F, x_9 = T, x_{31} = F]$
$[x_{17} = F, x_9 = F, x_{31} = T]$

כל אחת מההשמות המספקות קובעת את הערך של המספר I באופן אחד ויחיד (לערכים 0,1,2 בהתאמה).

את המספר I בתחום 0..2 ואת הנוסחה שמאלצת את סדרת המשתנים $\langle x_{17}, x_9, x_{31} \rangle$ נייצג ב־Java באופן הבא:

```
int[] I = {17,9,31};
int[][] cnf = {{17,9,31}, {-17,-9}, {-17,-31}, {-9,-31}}
```

במשימות הבאות נכתוב מספר פונקציות ב־Java, אשר בהינתן מערך של משתני CNF, מחזירות נוסחה המאלצת כל השמה מספקת לתת ערך true ללפחות/ללכל היותר/לבדיוק אחד מהמשתנים.

<u>משימה 4 (לפחות אחד) (5 נקודות):</u>

השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך של משתני CNF, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לתת ערך true ללפחות אחד מהמשתנים מהמערך.

הדרכה: התבוננו בקלט int[] vars = {17,9,31}. הנוסחה שמאלצת לפחות אחד ממשתני הקלט לקבל ערך *true*, אומרת למעשה: או ש־x_{17} מקבל ערך *true*, או ש־x_9 מקבל ערך *true*, או ש־x_{31} מקבל את הערך *true*. רישמו אילוץ זה כנוסחה בצורת CNF. שימו לב שבנוסחה זו פסוקית יחידה, והכלילו לקלט כלשהו.

הנחות על הקלט וחריגות:

- הניחו שהמערך vars אינו null ומכיל שמות משתני CNF חוקיים (גדולים מ־0) ושונים זה מזה.
- פונקציה זו אינה זורקת חריגות.

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

השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך של משתני CNF, ומחזירה נוסחת CNF המאלצת כל השמה מספקת לתת ערך true לאחד מהמשתנים במערך לכל היותר.

הדרכה: התבוננו בקלט int[] vars = {17,9,31}. נוסחה שאומרת שאחד המשתנים לכל היותר מקבל ערך true, אומרת למעשה: לא נכון שזוג המשתנים x_{17}, x_9 מקבלים שניהם את הערך true, וגם לא נכון שזוג המשתנים x_{17}, x_{31} מקבלים שניהם את הערך true, וגם לא נכון שזוג המשתנים x_9, x_{31} מקבלים שניהם את הערך true. רישמו אילוץ זה כנוסחה בצורת CNF, והכלילו לקלט כלשהו.

הנחות על הקלט וחריגות:

- הניחו שהמערך vars אינו null ומכיל שמות משתני CNF חוקיים (גדולים מ־0) ושונים זה מזה.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 6 (שרשור מערכים דו־ממדיים) (5 נקודות):</u>

השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

```
public static int[][] append(int[][] arr1, int[][] arr2)
```

הפונקציה מקבלת שני מערכים דו־ממדיים arr1, arr2, ומחזירה מערך דו־ממדי שהוא השרשור שלהם. כלומר, הפלט הוא מערך המכיל מערכים הזהים בתוכנם למערכים מ-arr1 ומערכים הזהים בתוכנם למערכים מ-arr2, מסודרים לפי סדר הופעתם ב-arr1 וב-arr2 בהתאמה. תחילה יופיעו מערכי arr1 ולאחריהם יופיעו מערכי arr2.

פונקציה זו תשמש אתכם כאשר תצטרכו לשרשר נוסחאות CNF עבור אילוצים שונים.

דוגמה:

```
int[][] arr1 = {{1,2},{7,9,10}};
int[][] arr2 = {{3,5,7,9},{16,10,11}};
int[][] arr3 = append(arr1,arr2)
for (int i = 0; i < arr3.length; i=i+1){
     for (int j = 0; j < arr3[i].length; j=j+1){
          System.out.print(arr3[i][j] + " ");
     }
     System.out.println();
}
```

הפלט עבור קטע קוד זה הוא:

```
1 2
7 9 10
3 5 7 9
16 10 11
```

הנחות על הקלט וחריגות:

- אין להניח שום הנחות על הקלט.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 7 (בדיוק אחד) (5 נקודות):</u>

השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך של משתני CNF, ומחזירה נוסחת CNF שמאלצת כל השמה מספקת לתת ערך true <u>לבדיוק אחד</u> מהמשתנים שבמערך.

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 שהמערך vars אינו null ומכיל שמות משתני CNF חוקיים (גדולים מ־0) ושונים זה מזה.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 8 (שני מספרים שונים זה מזה) (10 נקודות):</u>

השלימו את הגדרת הפונקציה בקובץ Assignment2.java:

```
public static int[][] diff(int[] I1, int[] I2)
```

הפונקציה מקבלת שני מערכים באורך n של משתני CNF אשר מייצגים **מספרים** בתחום 0...n-1, ומחזירה נוסחת CNF המאלצת את המספרים המיוצגים להיות שונים זה מזה.

הנחות על הקלט וחריגות:

- הניחו שהמערכים I1,I2 אינם null, מכילים שמות משתני CNF חוקיים (גדולים מ־0) ואורכם זהה.
- הניחו בנוסף ש־I1,I2 מייצגים מספרים בתחום 0,..,n-1.
- פונקציה זו אינה זורקת חריגות.

דוגמה: קטע הקוד הבא מגדיר 2 מספרים בתחום 0..2 בעזרת משתני CNF ומאלץ את המספרים להיות שונים:

```
int[] I1 = {1,2,3};
int[] I2 = {4,5,6};
int[][] cnf1 = exactlyOne(I1);
int[][] cnf2 = exactlyOne(I2);
int[][] cnf3 = diff(I1,I2);
int[][] cnf = append(cnf1,append(cnf2,cnf3));
int nVars = 6;
SATSolver.init(nVars);
SATSolver.addClauses(nVars);
boolean[] assignment = SATSolver.getSolution();
```

ערך אפשרי למערך assignment (שמייצג השמה מספקת של הנוסחה) הוא:
{false, true, false, false, false, true, false} – עבור השמה זו I1 מייצג את המספר 0 ו־I2 את המספר 1.

רדוקציה מבעיית הטיול הגדול לבעיית הספיקות

<u>תיאור הרדוקציה:</u>

בהינתן מופע של בעיית *הטיול הגדול על n ערים*, נייצג פתרון של הבעיה בעזרת מערך map של n מערכים. המערך ה־i ב־map מייצג את מספר העיר שמבקרים בה בצעד ה־i של פתרון לבעיה.

דוגמא: עבור מופע של n=4 ערים, נגדיר מערך map של 4 מערכים. המערך ה־i (עבור $0 \leq i \leq 3$) ב־map מציין את משתני ה־CNF של מספר לא ידוע בתחום 0..3. קטע הקוד הבא מגדיר 4 מערכים $I1, I2, I3, I4$ שכל אחד מהם מכיל 4 משתני CNF בייצוג של Java. כל אחד מארבעת המערכים מייצג מספר בין 0 ל־3. בשורה האחרונה מוגדר המערך map, שמכיל את ארבעת המערכים.

```
//java representation of  the CNF variables
int [] I1 = {1,2,3,4};                    //  I1 = [x1,x2,x3,x4]
int [] I2 = {5,6,7,8};                    //  I2 = [x5,x6,x7,x8]
int [] I3 = {9,10,11,12};                 //  I3 = [x9,x10,x11,x12]
int [] I4 = {13,14,15,16};                //  I4 = [x13,x14,x15,x16]
int[][] map = {I1,I2,I3,I4};
```

נרצה להגדיר נוסחת CNF שכל השמה מספקת שלה נותנת ערכים למשתני ה־CNF ב־map, כך שהמספרים המיוצגים על ידי המערכים ב־map מהווים פתרון למופע נתון של בעיית הטיול הגדול. לשם כך, נגדיר כמה אילוצים בוליאנים על משתני ה־CNF.

בהינתן מופע על n ערים, האילוצים הבאים מייצגים את הדרישות מפתרון המיוצג על ידי map:

א. כל מערך ב־map מייצג מספר בתחום 0..n-1.
ב. כל הערים בפתרון המיוצג על ידי map שונות זו מזו. כלומר, לכל שני מערכים I_i, I_j $(i < j)$ ב־map, המספר המיוצג על ידי I_i שונה מהמספר המיוצג על ידי I_j.
ג. כל הצעדים בפתרון המיוצג על ידי map חוקיים.

<u>בניית נוסחת CNF לייצוג הדרישות ב־Java</u>

במשימות הבאות נממש תכנית ב־Java אשר מקבלת מופע של בעיית הטיול הגדול על n ערים ומייצרת את נוסחת ה־CNF המתאימה לה בייצוג של Java.

<u>משימה 9 (מיפוי המשתנים) (5 נקודות):</u>

במשימה זו נבנה את המערך map עבור מופע על n ערים.
השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת כקלט מספר n (מספר הערים במופע) ומחזירה מערך דו־ממדי המייצג את משתני ה־CNF המתאימים למופע הכולל n ערים.

דוגמה:

```
int n = 4;
int[][] map = createVarsMap(n) ;
for (int i = 0; i < n; i=i+1){
     for (int j = 0; j < n; j=j+1){
          System.out.print(map[i][j] + " ");
     }
     System.out.println();
}
```

הפלט עבור קטע קוד זה הוא:

```
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 16
```

<u>הדרכה חובה:</u> המשתנים במערך map ממוספרים באופן הבא: המשתנים במערך הראשון ממוספרים מ־1 עד n, המספרים במערך השני מ־n+1 ועד 2n, וכו'.

הנחות על הקלט וחריגות:

- הניחו ש־$n > 1$.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 10 (כל מערך ב־map מייצג מספר) (5 נקודות):</u>

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך map המייצג פתרון למופע על n ערים, ומחזירה נוסחת CNF המאלצת כל מערך של משתנים בוליאנים ב־map לייצג מספר בתחום 0,..,n-1.

הנחות על הקלט וחריגות:

- הניחו ש־map מייצג באופן תקין פתרון עבור מופע על n ערים כמתואר במשימה 12.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 11 (כל הערים בפתרון שונות) (5 נקודות):</u>

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת מערך map המייצג פתרון למופע על n ערים, ומחזירה נוסחת CNF המאלצת לכל $i < j$ את המערכים map[i] ו־map[j] לייצג מספרים השונים זה מזה.

הנחות על הקלט וחריגות:

- הניחו ש־map מייצג באופן תקין פתרון עבור מופע על n ערים כמתואר במשימה 12.
- פונקציה זו אינה זורקת חריגות.

הדרכה חובה: השתמשו בפונקציה diff שכתבתם במשימה x.

<u>משימה 12(כל הצעדים חוקיים) (5 נקודות):</u>

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static int[][] allStepsAreLegal(int[][] flights, int[][] map)
```

הפונקציה מקבלת מופע flights של בעיית הטיול הגדול על n ערים, ומערך map המייצג פתרון למופע על n ערים, כפי שתואר במשימה 12. הפונקציה מחזירה נוסחת CNF המאלצת כל פתרון להכיל טיסות חוקיות.

פתרון מכיל טיסות חוקיות אם רצף הטיסות שהוא מציין מורכב מטיסות קיימות במופע flights הנתון. באופן פורמלי, עבור כל $0 \leq i < n - 1$ נרצה לאלץ את הפתרון באופן הבא: אם העיר בשלב ה־i היא j, וגם - העיר בשלב ה־$i + 1$ היא k, אזי קיימת טיסה בין j ל־k במופע flights. כמו כן, נרצה לאלץ באופן דומה את הצעד האחרון (חזרה לעיר המקור): אם העיר בשלב ה־$n - 1$ היא j, וגם - העיר בשלב ה־0 היא k, אזי קיימת טיסה בין j ל־k במופע flights.

שימו לב שהעיר בשלב ה־s היא t אם משתנה ה־CNF $map[s][t]$ מקבל את הערך true בהשמה המספקת.

כעת נתאר את האילוצים כנוסחת CNF. עבור כל $0 \leq i < n - 1$ נרצה לבטא בעזרת פסוקית CNF את האילוץ:

$$map[i][j] \wedge map[i+1][k] \rightarrow hasFlight(flights, j, k)$$

כאשר $map[i][j]$ ו־ $map[i+1][k]$ הם משתני CNF ו־ $hasFlight(flights, j, k)$ הוא הערך true או false המוחזר מקריאה לפונקציה זו. אילוץ זה ניתן לרשום באופן שקול כפסוקית CNF באופן הבא:

$$\neg map[i][j] \vee \neg map[i+1][k] \vee hasFlight(flights, j, k)$$

נשים לב שאם הערך של $hasFlight(flights, j, k)$ הוא true, אזי הפסוקית מסתפקת ללא קשר להשמה למשתנים האחרים. לכן, אין צורך להוסיף אילוץ במקרה זה.
לעומת זאת, אם הערך של $hasFlight(flights, j, k)$ הוא false, אזי הפסוקית מסתפקת אם ורק אם $\neg map[i][j] \vee \neg map[i+1][k]$.

לסיכום, לכל i,j,k כך ש־ $hasFlight(flights,j,k) = false$, יש להוסיף את הפסוקית
$\neg map[i][j] \vee \neg map[i+1][k]$.

באופן דומה, נוסיף פסוקית שמבטיחה שהחזרה לעיר המקור היא גם צעד חוקי.
לכל j,k כך ש- $hasFlight(flights,j,k) = false$, יש להוסיף את הפסוקית
$\neg map[n-1][j] \vee \neg map[0][k]$

הנחות על הקלט וחריגות:
- הניחו ש־flights מייצג מופע תקין על n ערים.
- הניחו ש־map מייצג באופן תקין פתרון עבור מופע על n ערים כמתואר במשימה 12.
- פונקציה זו אינה זורקת חריגות.

<u>משימה 13 (ממיר קלט) (5 נקודות):</u>

בהינתן מופע של בעיית הטיול הגדול ומערך map המייצג פתרון למופע על n ערים, נבנה נוסחת CNF המקודדת את כל האילוצים ונוסיף אותה לפותרן. השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת מופע של הבעיה $flights$, וכן מערך map שמייצג פתרון למופע על n ערים, <u>ומוסיפה לפותרן</u> נוסחת CNF המקודדת את האילוצים של מופע נתון של בעיית הטיול הגדול. שימו לב שהפונקציה אינה מחזירה ערך.

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 map אינו מייצג פתרון למופע על n ערים באופן שתואר במשימה 12.

משימה 14 (ממיר פלט) (5 נקודות):

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

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

הפונקציה מקבלת השמה assignment למשתני ה־CNF המתוארים במערך map. על הפונקציה להחזיר מערך המהווה פתרון למופע כך שהערך בתא ה־i של המערך הוא מספר העיר שנבקר בה בצעד ה־i במסלול.

הנחות על הקלט וחריגות:

- הניחו ש־map מייצג באופן תקין פתרון עבור מופע על n ערים כמתואר במשימה 12.
- 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).

דוגמה:

```
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)

for (int i=0; i < tour.length; i=i+1) {
    System.out.print(tour[i] + " ");

}
```

הפלט עבור קטע קוד זה הוא: 0 3 1 2

משימה 15 (מציאת פתרון למופע) (10 נקודות):

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

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

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

- לייצר את המערך map.
- לאתחל פותרן לבעיית הספיקות.
- לקודד את המופע flights, בעזרת המערך map, 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 (מקרה זה ייתכן אם הנוסחה שמתארת את האילוצים הנדרשים מהפתרון אינה נכונה).
- אם אין השמה מספקת, יש להחזיר את הערך 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 שיש השמה מספקת אך הפתרון המתקבל ממנה אינו חוקי.

<u>משימה 16 (קיום של לפחות שני מסלולים) (5 נקודות):</u>

בהינתן מופע של בעיית הטיול הגדול, נרצה לדעת אם קיימים למופע זה <u>שני</u> פתרונות שונים כך שהעיר הראשונה במסלול היא s והעיר האחרונה (לפני החזרה ל־s) היא t.

השלימו את הגדרת הפונקציה הבאה בקובץ Assignment2.java:

```
public static boolean solve2(int[][] flights, int s, int t)
```

הפונקציה מקבלת כקלט מופע של בעיית הטיול הגדול flights, את מספר העיר שנתחיל ממנה את הטיול (s), ואת מספר העיר שנסיים בה את הטיול (t) (לפני החזרה ל־s). הפונקציה מחזירה true אם ורק אם קיימים לפחות שני פתרונות שונים כך שעיר המקור בכל אחד מן הטיולים היא s והעיר האחרונה (לפני החזרה ל־s) היא t.

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 שיש השמה מספקת אך הפתרון המתקבל ממנה אינו חוקי.

בהצלחה!

נספח: שימוש בפותרן לבעיית הספיקות

העבודה עם הפותרן מסתמכת על שני קבצים שנמצאים בקובץ ה -zip שסופק לכם לעבודה זו:
SATSolver.java ו – org.sat4j.core.jar.

הקובץ SATSolver.java מכיל עיקרי הממשק שתשתמשו בו לעבודה עם הפותרן.

הקובץ org.sat4j.core.jar מממש את הפותרן עצמו. יש להוסיף את הקובץ לפרוייקט כפי שמתואר בסוף נספח זה.

יש להוסיף את הקובץ SATSolver.java לאותה התיקייה יחד עם שאר קבצי הג'אווה של עבודת הבית. אין לשנות את קובץ זה, ואין להגישו יחד עם קבצי 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:

- <u>אתחול:</u> 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 $\{x_1, \dots, x_{34}\}$.
- <u>הוספת פסוקית:</u> 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.

- <u>הוספת פסוקיות:</u> 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.

- <u>מציאת השמה מספקת:</u> 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 מסתפקת, "TIMEOUT" אם הפותרן לא מצא פתרון עקב מגבלת זמן ו־"UNSAT" אם הנוסחה אינה מסתפקת.

```
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 מסתפקת, "TIMEOUT" אם הפותרן לא מצא פתרון עקב מגבלת זמן ו־"UNSAT" אם הנוסחה לא מסתפקת .

```
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 שבו 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.