עבודת בית מספר 2

מערכים, פונקציות ובעיית הסיפוק הבוליאני

מבוא למדעי המחשב – סמסטר ב' תשע"ט

עבודת בית מספר 2: מערכים, פונקציות ובעיית הסיפוק הבוליאני

מתרגלת אחראית: נועה בן-דוד

תאריך פרסום: יום ראשון, 31.3.2019

תאריך הגשה: יום שני, 15.4.2019 - עד השעה 14:00 בצהריים

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

נכתוב תכנית לשיבוץ מערכת בחינות. במערכת יש סטודנטים שלומדים בקורסים שונים. על המערכת להציע לוח בחינות אפשרי לקורסים אלה, המתחשב בכך שסטודנט אינו יכול להיבחן בשתי בחינות באותו היום.

בעיה זו נקראת בעיית שיבוץ הבחינות, או באנגלית: Exam Timetable Problem (ETP). בעבודה זו נממש אלגוריתם הפותר את הבעיה בעזרת רדוקציה ל"בעיית הסיפוק הבוליאני", או באנגלית: The Boolean Satisfaction Problem (SAT). נשתמש ב"פותרן של בעיית הסיפוק הבוליאני" (SAT Solver).

הוראות מקדימות

הגשת עבודות בית

1. קראו את העבודה מתחילתה ועד סופה לפני שאתם מתחילים לפתור אותה. ודאו שאתם מבינים את כל השאלות. רמת הקושי של המשימות אינה אחידה: הפתרון של חלקן קל יותר, ואחרות מצריכות חקירה מתמטית (שאותה תוכלו לבצע בספרייה או בעזרת מקורות דרך רשת האינטרנט). בתשובות שאתם מסתמכים בהן על עובדות מתמטיות שלא הוצגו בשיעורים, יש להוסיף כהערה במקום המתאים בקוד את ציטוט העובדה המתמטית ואת המקור (כגון ספר או אתר).
2. יש להגיש את העבודה בזוגות. עליכם להירשם כזוג להגשת העבודה במערכת ההגשות (Submission System) המחלקתית.
3. לעבודה מצורפים קובצי Java עם שמות כגון Task<n>.java. צרו תיקייה חדשה והעתיקו את קובצי ה-Java לתוכה. עליכם לערוך את הקבצים האלו בהתאם למפורט בתרגיל ולהגישם כפתרון, מכווצים כקובץ ZIP יחיד.
4. **עליכם להגיש רק את קובצי ה-Java הנקראים Task<n>.java.** אין לשנות את שמות הקבצים. אין להגיש קבצים נוספים.
5. שם קובץ ה-ZIP יכול להיות כרצונכם, אך באנגלית בלבד. בנוסף, הקבצים שתגישו יכולים להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכו') לא יתקבל.
6. קבצים שיוגשו שלא על פי הנחיות אלו לא ייבדקו.
7. את קובץ ה-ZIP יש להגיש ב-Submission System. פרטים על ההרשמה כזוג ועל אופן הגשת העבודה תוכלו למצוא באתר הקורס.

בדיקת עבודות הבית

8. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי. הבדיקה האוטומטית מתייחסת לפלט התכנית המודפס למסך. לכן, יש להקפיד על ההוראות ולבצע אותן במדויק. כל הדפסה אשר אינה עונה בדיוק על הדרישות המופיעות בעבודה (כולל שורות, רווחים, סימני פיסוק או כל תו אחר - מיותרים, חסרים או מופיעים בסדר שונה מהנדרש), לא תעבור את הבדיקה האוטומטית ולכן תגרור פגיעה בציון.
9. 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.
10. בכל פעם שאתם מתבקשים להדפיס למסך לפי הפורמט המתאים, עליכם להשתמש בפונקציה System.out.println - אשר מדפיסה למסך ויורדת שורה, או בפונקציה System.out.print - אשר מדפיסה למסך ואינה יורדת שורה. אין להדפיס למסך דברים מיותרים (כגון: "please enter an integer"). בכדי לקלוט נתונים מהמשתמש יש להשתמש ב-Scanner, כפי שנלמד בכיתה.

עזרה והנחיה

לכל עבודת בית בקורס יש מתרגל שאחראי לה. ניתן לפנות למתרגל בשעות הקבלה.

הערות ספציפיות לעבודת בית זו

11. לכל פונקציה הכתובה באחד מקובצי ה- Task<n>.java ניתן לקרוא גם מתוך קובצי משימות אחרים.
12. לעבודה זו מצורפים קבצים נוספים, המכילים דוגמאות (לקלט/פלט) ופונקציות אחרות שיעזרו לכם לבדוק את נכונות הקוד שלכם. הקובץ Example<n>.java מכיל דוגמאות רלוונטיות למשימה n. מצורפים גם קבצים נוספים, שיתוארו בהמשך. הבחירה אם להשתמש או לא להשתמש בקבצים אלו היא בידיכם. בכל מקרה, אין להגיש את הקבצים הללו.
13. בעבודה זו יש 11 משימות (כולל משימה 0). הניקוד על כל משימה הוא 10 נקודות, ולכן בסה"כ ניתן להגיע לציון מקסימלי של 110 נקודות. שימו לב כי חלק מהסעיפים מסתמכים על סעיפים קודמים. אם לא פתרתם סעיף מסוים, לא תוכלו לפתור סעיף שמסתמך עליו. יתרה מכך, אם הקוד שלכם בסעיף מסוים שגוי, יתכן שקוד (והתוצאה) של סעיף אחר שמסתמך על סעיף זה יהיה שגוי. עבדו בזהירות ובדקו היטב את הקוד שאתם כותבים.

יושר אקדמי

הימנעו מהעתקות! ההגשה היא בזוגות בלבד, ויש להגיש עותק יחיד של העבודה על ידי אחד מבני הזוג. במידה שמוגשות שתי עבודות עם קוד זהה, או אפילו דומה - ייווצר חשד להעתקה, אשר יגרור דיווח לאלתר לוועדת משמעת.

חלק א: בעיית שיבוץ הבחינות

מבוא

בהינתן n סטודנטים, אשר לומדים ונבחנים ב-m קורסים (כל סטודנט לומד בחלק מהקורסים, לאו דווקא בכולם), נרצה לבדוק אם (וכיצד) ניתן לשבץ את הבחינות של כל m הקורסים ב-k ימים שונים, מבלי שסטודנט יידרש להיבחן בשני קורסים שונים באותו היום.

נמספר את n הסטודנטים: $0,1,\dots,n-1$ ונתייחס למספור זה כאשר נאמר: "*הסטודנט שמספרו* $\boldsymbol{i}$". באותו אופן נמספר את m הקורסים: $0,1,\dots,m-1$ ונתייחס למספור זה כאשר נאמר: "*הקורס שמספרו* $\boldsymbol{i}$". נמספר גם את k הימים שבהם ניתן לקיים בחינות: $0,1,\dots,k-1$ ונתייחס למספור זה כאשר נאמר: "*יום הבחינות שמספרו* $\boldsymbol{i}$". שימו לב: כל הספירות הן החל מאפס.

בבעיית שיבוץ הבחינות נאפשר לשבץ שתי בחינות באותו היום רק אם הן אינן מתנגשות.

הגדרה #1: נאמר ששתי בחינות, בחינה של קורס **א** ובחינה של קורס **ב**, **מתנגשות** אם הקורסים **א** ו-**ב** שונים זה מזה וקיים תלמיד שלומד בשני הקורסים, **א** ו-**ב**.

שימו לב שלפי הגדרה #1 קורס אינו מתנגש עם עצמו.

ייצוג הבעיה

את הנתונים לבעיית שיבוץ הבחינות נייצג באמצעות ארבעה פרמטרים, אשר יחד ייקראו "מופע של בעיית שיבוץ הבחינות", או, בקיצור: "מופע".

- ייצוג הסטודנטים: את הסטודנטים נייצג באמצעות מערך של מחרוזות String[] students באורך n, כאשר students[i] הוא שמו של הסטודנט שמספרו i.
- ייצוג הקורסים: את הקורסים נייצג באמצעות מערך של מחרוזות String[] courses באורך m, כאשר courses[i] הוא שמו של הקורס שמספרו i.
- ייצוג מצב הרישום: את מצב הרישום של סטודנטים לקורסים נייצג באמצעות מערך דו-ממדי של מספרים שלמים int[][] studentCourses, כאשר studentCourses[i] הוא מערך המכיל את מספרי הקורסים שאליהם רשום הסטודנט שמספרו i.
- ייצוג הימים האפשריים: את מספר הימים שניתן לקיים בהם בחינות נייצג באמצעות מספר שלם חיובי, k>0.

הגדרה #2: *מופע* של בעיית שיבוץ הבחינות מורכב מארבעת הפרמטרים המתוארים לעיל.

משימה 1:

בהינתן מערך של מחרוזות באורך כלשהו, נרצה לבדוק כי כל המחרוזות במערך שונות זו מזו. השלימו את הפונקציה הבאה בקובץ Task0.java:

```
public static boolean uniqueStrings(String[] array)
```

על הפונקציה לוודא כי array לא ריק, כל ערכיו שונים זה מזה, ואף אחד מאבריו הוא לא null או מחרוזת ריקה. יש להניח כי array אינו null.

דוגמאות:

```
String[] array1 = {"Miri","Yosef","Eden","Tamar"};
System.out.println(uniqueStrings(array1)); // true ;

String[] array2 = {"Miri","Yosef","Eden","Miri","Tamar"};
System.out.println(uniqueStrings (array2)); // false ;

String[] array3 = {} ;
System.out.println(uniqueStrings(array3)); // false ;
```

משימה 2:

בהינתן מערך של מספרים באורך כלשהו, נרצה לבדוק כי כל ערכי המערך שונים זה מזה ונמצאים בטווח נתון.
השלימו את הפונקציה הבאה בקובץ Task0.java:

```
public static boolean uniqueIntsBetween(int[] array, int from,
int to)
```

על הפונקציה להחזיר ערך true אם ורק אם array הוא מערך לא ריק שכל איבריו שונים, גדולים שווים לערך from וקטנים שווים לערך to. יש להניח כי array אינו null.

דוגמאות:

```
int[] array1={};

System.out.println(uniqueIntsBetween(array1,1, 5)); // false

int[] array2={-1,5,9,7,8,4,0};
System.out.println(uniqueIntsBetween (array2,-2, 9)); // true

int[] array3={1,3,7,0,-2};
System.out.println(isMatrixBetween(array3,1,9)); // false
```

משימה 3:

בהינתן מערך דו-מימדי של מספרים, נרצה לבדוק שכל תא במימד הראשון של המערך מצביע על מערך שאינו null, אינו ריק, ושכל איבריו שונים זה מזה ונמצאים בטווח נתון. השלימו את הפונקציה הבאה בקובץ Task0.java:

```
public static boolean isMatrixBetween(int[][] matrix, int from,
int to)
```

על הפונקציה להחזיר ערך true אם ורק matrix היא מערך לא ריק של מערכים שכל אחד מהם שונה מ-null ולא ריק, שכל ערכיו שונים זה מזה, גדולים שווים לערך from וקטנים שווים לערך to. יש להניח כי matrix אינו null.

דוגמאות:

```
int[][] matrix1 = {{1,2,3},
                   {1,2,3},
                   {1,2,3}};
System.out.println(isMatrixBetween(matrix1,1,3)); // true

int[][] matrix2 = {{1,2,3},
                   null,
                   {1,2,3}};
System.out.println(isMatrixBetween(matrix2,1,3)); // false

int[][] matrix3 = {{1,2,3},
                   {2,2,3},
                   {1,2,3}};
System.out.println(isMatrixBetween(matrix3,1,3)); // false
```

משימה 4:

בהינתן מטריצת מספרים ומספר שלם m, נרצה לבדוק שהמטריצה מכילה כל אחד מהמספרים מ-0 עד m-1 לפחות פעם אחת. השלימו את הפונקציה הבאה בקובץ Task0.java:

```
public static boolean containsAll(int[][] matrix, int m)
```

על הפונקציה להחזיר true אם ורק אם matrix מכילה כל מספר מ-0 עד m-1 לפחות פעם אחת. ניתן להניח כי הקלט תקין.

דוגמאות:

```
int[][] matrix1 = {{0,1,2,3},
                   {1,2,3},
                   {1,2,3}};
System.out.println(containsAll(matrix1,4)); // true

int[][] matrix2 = {{1,2,3},
                   {1,2,3},
                   {1,2,3}};
System.out.println(containsAll (matrix2,4)); // false
```

משימה 5:

במשימה זו נכתוב פונקציה שמוודאת את תקינותו של מופע של בעיית שיבוץ הבחינות, כפי שתואר בהגדרה #2. מופע נתון נחשב תקין אם התנאים הבאים מתקיימים:

- המערכים students, courses אינם null ואינם ריקים.
- המערך studentCourses אינו null, אינו ריק, ואינו מכיל איברים ששווים ל-null.
- כל סטודנט לומד לפחות קורס אחד.
- בכל קורס לומד לפחות סטודנט אחד.
- שמות הסטודנטים במערך students שונים[1] זה מזה, אינם null, ואינם המחרוזת הריקה.
- שמות הקורסים במערך courses שונים זה מזה, אינם null, ואינם המחרוזת הריקה.
- אורך הממד הראשון של המערך studentCourses הוא כאורכו של המערך students.
- לכל אינדקס $0 \le i < \text{studentCourses.length}$, המערך studentCourses[i] אינו ריק ואיבריו הם ערכים שמייצגים מספר קורס מתוך $\{0,1,\dots,m-1\}$ ואינם מכילים כפילויות.
- המספר k הינו מספר טבעי הגדול מאפס.

השלימו את הגדרת הפונקציה בקובץ Task0.java:

```
public static boolean legalData(String[] students, String[]
courses, int[][] studentCourses, int k)
```

הפונקציה מקבלת מופע של בעיית שיבוץ הבחינות (לפי ההגדרה #2) ומחזירה true אם המופע הינו תקין לפי התנאים שתוארו במשימה זו. אחרת, הפונקציה מחזירה false. בכל מקרה, לפונקציה זו אסור לזרוק חריגות.

אין צורך לוודא ששמות הסטודנטים והקורסים הם רצפים הגיוניים של תווים. למשל, השם cb2xyz77 הוא שם אפשרי ותקין לחלוטין.

דוגמאות:

```
System.out.println(legalData(null,null,null,4)); // false


String[] students = {"student 1", null, "student 3"} ;
String[] courses = {"course1", "course2", "course3"} ;
int[][] studentCourses = {{0,1,2}, {1,2}, {0}} ;
int k = 2 ;
System.out.println(legalData(students, courses, studentCourses,
k)); // false


String[] students = {"student 1", "student2", "student 3"} ;
String[] courses = {"course1", "course2", "course3", "course4"}
;
int[][] studentCourses = {{0,2}, {1,2}, {0}} ;
int k = 2 ;
```

[1] השמות הם case sensitive; למשל, השמות Dan, DAN, dan וכדומה נחשבים שמות שונים.

```
System.out.println(legalData(students, courses, studentCourses,
k)); // false

String[] students =
               { "Yael Studentson",
                 "Moshe Classroom",
                 "Alice",
                 "Bob"
               } ;

String[] courses =
               { "Introduction to Computer Science",
                 "Algebra 1",
                 "Calculus 1",
                 "Introduction to Logic and Set Theory",
                 "English"
               } ;

int [][] studentCourses =
               { {0,1,3,4},
                 {1,2,3},
                 {0,1,2,3,4},
                 {2,3,1}
               } ;

int k = 3 ;
System.out.println(legalData(students, courses, studentCourses,
k)); // true
```

ייצוג הפתרון

בהינתן מופע של בעיית שיבוץ הבחינות (לפי הגדרה #2), פתרון (אם קיים) הוא שיבוץ של בחינות לימים. את השיבוץ נייצג באמצעות מערך schedule []int כאשר הערך schedule[i] מציין את יום הבחינה בקורס i. כלומר, היום שבו מתקיימת הבחינה של קורס i. אורכו של המערך schedule הוא m (מספר הקורסים).

משימה 6:

בהינתן מופע של בעיית שיבוץ הבחינות, אנו נאמר ששיבוץ (של בחינות לימים) הוא פתרון חוקי אם שני התנאים הבאים מתקיימים:

א. לכל קורס במערך הקורסים משובצת בחינה ביום חוקי (ערך בין אפס ל k-1, כולל)
ב. אין סטודנט במערך הסטודנטים שמשובץ לשתי בחינות באותו היום.

במשימה זו נכתוב שתי פונקציות, שמקבלות מופע נתון ולוח בחינות מוצע ומתייחסות לחוקיות הלוח ביחס למופע.

<u>משימה 6.1</u>: השלימו את הגדרת הפונקציה בקובץ Task6Verify.java:

```
public static int[] findConflictingStudent (int[] schedule
,String[] students, String[] courses, int[][] studentCourses,
int k)
```

הפונקציה מקבלת את השיבוץ ואת המופע ומחפשת סטודנט ששובץ להיבחן בשתי בחינות באותו היום. אם קיים סטודנט כזה, הפונקציה תחזיר מערך חד-ממדי בעל ארבעה איברים:

{dayNum, studentNum, examNum1, examNum2}

המפרט כי הסטודנט שמספרו studentNum נבחן ביום dayNum בשני קורסים שונים, שמספרם examNum1 ו-examNum2. אם קיימים כמה קונפליקטים, יש להחזיר רק אחד (לא חשוב איזה). אם לא קיים קונפליקט - הפונקציה תחזיר מערך ריק.

<u>משימה 6.2</u>: השלימו את הגדרת הפונקציה בקובץ Task6Verify.java:

```
public static boolean isLegalSchedule (int[] schedule ,String[]
students, String[] courses, int[][] studentCourses, int k)
```

הפונקציה מקבלת את השיבוץ ואת המופע ומחזירה true אם הלוח הוא חוקי עבור המופע. אחרת, היא מחזירה false.

חלק 2: רדוקציה לבעיית הסיפוק הבוליאני

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 הסיפוק הבוליאני 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: $1, \dots, n$

- 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.

משימה 7:

במשימה זו נגדיר ארבע 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 כזאת - שקבוצת ההשמות המספקות שלה תואמת בדיוק את האופנים שבהם ניתן לספק את האילוץ.

משימה 7.1: At Least One

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 Task7Cnf.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*. רשמו נוסחה זו בצורת CNF והכלילו לקלט כלשהו.

במשימה זו יש להניח שהקלט תקין.

משימה 7.2: At Most One

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 Task7Cnf.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. רשמו נוסחה זו בצורת CNF והכלילו לקלט כלשהו.

במשימה זו יש להניח שהקלט תקין.

משימה 7.3: Exactly One

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 Task7Cnf.java:

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

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*.

במשימה זו יש להניח שהקלט תקין.

משימה 7.4: Not Same Day

במשימה זו נבנה נוסחת CNF אשר מבטאת אילוץ שיתואר בהמשך. השלימו את הגדרת הפונקציה בקובץ Task7Cnf.java:

```
public static int[][] notBothTrue(int[] vars1, int[] vars2)
```

הפונקציה מקבלת שני מערכים vars1 ו-vars2 של (שמות של) של משתני CNF, שניהם באורך $n \geq 0$. הפונקציה מחזירה נוסחת CNF שקבוצת ההשמות המספקות שלה תואמת את כל האופנים שבהם ניתן לתת ערכים למשתנים כך שבכל אינקדס $0 \leq i < n$, לפחות אחד המשתנים vars1[i] ו- vars2[i] מקבל את הערך false.

הדרכה: נניח ש int[] vars1 = {2,5,7} ו- int[] vars2 = {3,4,8}. הנוסחה המבוקשת צריכה לציין עבור זוגות המשתנים (x_2, x_3), (x_5, x_4), ו- (x_7, x_8) שלפחות אחד המשתנים בכל זוג מקבל את הערך *false*. רשמו נוסחה זו בצורת CNF והכלילו לקלט כלשהו.

במשימה זו יש להניח שהקלט תקין.

בקובץ Example7.java ניתן למצוא כמה בדיקות בסיסיות שבעזרתן ניתן לבחון את נכונות הפונקציות שכתבתם במשימה זו. בדיקות אלו אינן מקיפות, והרצתן אינה מבטיחה את נכונות הפונקציות שכתבתם.

השימוש בפותרן לבעיית הסיפוק הבוליאני

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 זה יש פותרן לבעיית 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 נוסחאת ה-CNF שהצטברה עד כה ב-SATSolver יש לקרוא לפונקציה

  SATSolver.getSolution()

פונקציה זו מחזירה ערך לפי אחת משתי האפשרויות הבאות:

1. **מערך בוליאני שאינו ריק** - אם ישנה 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** – אם הנוסחה אינה ספיקה (לא קיימת לה השמה מספקת).

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, מדפיסה "SAT" עם הנוסחה מסתפקת, ומדפיסה "UNSAT" אם לא.

```
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.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, מדפיסה "SAT" עם הנוסחה מסתפקת, ומדפיסה "UNSAT" אם לא.

```
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.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. הקובץExamplesUNSAT.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 שמכיל את הפותרן. אינכם צריכים לעבוד איתו ישירות, אבל הוא חייב להיות 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 משולב בפרויקט, תוכלו לכתוב פונקציית main עם קוד מאחת הדוגמאות בסעיף הקודם ולוודא שהדוגמה אכן עובדת.

רדוקציה מבעיית שיבוץ הבחינות לבעיית הסיפוק הבוליאני

רדוקציה של בעיה א' (במקרה שלנו: בעיית שיבוץ הבחינות) לבעיה ב' (במקרה שלנו: בעיית הסיפוק הבוליאני) מוגדרת במונחים של ממיר קלט ושל ממיר פלט ומאפשרת לפתור מופעים של בעיה א' בהינתן אלגוריתם שפותר מופעים של בעיה ב'.

ייצוג הבעיה באמצעות משתנים בוליאניים

ממיר הקלט מקבל מופע של בעיית שיבוץ הבחינות וממיר אותו למופע של בעיית הסיפוק הבוליאני (נוסחת CNF). לתהליך ההמרה, שני מרכיבים: המרת המשתנים, והוספה של פסוקיות. תחילה נתמקד בהמרה של המשתנים (משתני המופע של בעיית השיבוץ למשתני ה-CNF).

בבעיית שיבוץ הבחינות אנו מעוניינים למצוא ערכים למשתנים $d_1, d_2, \dots, d_m$ המציינים את ימי הבחינה ב-m הקורסים. כל משתנה d_i מקבל ערך בין 0 (כולל) ל- k (לא כולל), שמציין באיזה יום תיערך הבחינה בקורס ה-i.

הגדרת משתני ה-CNF: לכל קורס i $(0 \le i < m)$, נבחר k משתני CNF שיציינו את יום השיבוץ של הקורס. אם המשתנה ה-j (מתוך k המשתנים) של הקורס ה-i יקבל ערך true, אזי הבחינה של הקורס ה-i תשובץ ביום ה-j (בספירה מאפס). שימו לב שבדיוק אחד מתוך k המשתנים האלו צריך לקבל את הערך true, כיוון שהבחינה צריכה להשתבץ ביום יחיד.

משימה 8:

במשימה זו נבחר שמות עבור משתני ה-CNF שישמשו אותנו לפתרון מופע של בעיית שיבוץ של m בחינות בלכל היותר k ימים. לכל אחד מ-m הקורסים נשתמש ב-k משתני CNF כדי לייצג את יום השיבוץ של הבחינה. בסך הכול נשתמש ב-$m \times k$ משתנים. השלימו את הגדרת הפונקציה בקובץ Task8Map.java:

```
public static int[][] variableTable (int m, int k)
```

הפונקציה מקבלת את מספר הקורסים m ואת מספר הימים k ומחזירה מערך דו-ממדי variableNames. אורך הממד הראשון של המערך המוחזר הוא m והאיבר variableNames[i] מכיל שמות של k משתני CNF. בנוסחת ה-CNF שנבנה בהמשך נדאג שמשתנה ה-CNF variableNames[i][j] יקבל את הערך true אם ורק אם הבחינה בקורס i משובצת ליום j.

הדרכה: יש להחזיר מערך דו-ממדי שמכיל את כל הערכים מ-1 ועד ל-$m \times k$. אפשר פשוט להציב את הערכים במערך. למשל, עבור ארבעה מבחנים המשובצים לשלושה ימים, אם נבצע את הפקודה

int[][]variableNames = variableTable(4,3)

אזי התוצאה היא ש-

variableNames={{1,2,3}, {4,5,6}, {7,8,9}, {10,11,12}}

שימו לב: הערך variableNames[2][1]=8 הוא הייצוג בג'אווה של משתנה ה-CNF x_8 שיקבל ערך true אם ורק אם הבחינה בקורס 2 תשובץ ליום 1. המערך variableNames[3]={10,11,12} מכיל את שמות משתני ה-CNF שמציינים לאיזה יום תשובץ הבחינה בקורס שמספרו 3. משתנה ה-CNF x_{10} יקבל ערך true אם הבחינה בקורס 3 תשובץ ביום 0, x_{11} יקבל ערך true אם הבחינה בקורס 3 תשובץ ביום 1 ו-x_{12} יקבל ערך true אם הבחינה בקורס 3 תשובץ ביום 2. במשימה זו יש להניח תקינות קלט; כלומר, יש להניח כי m ו-k גדולים מאפס.

משימה 9:

במשימה זו נממש ממיר קלט עבור בעיית שיבוץ הבחינות. בהינתן מופע של בעיית שיבוץ הבחינות, נייצר נוסחת CNF המבטאת את האילוצים של המופע של בעיית השיבוץ. כל אילוץ יתורגם לפסוקיות CNF שנוסיף לפותרן (כפי שהוסבר). נתאר כעת האילוצים שנרצה להמיר לנוסחאות CNF ולהוסיף לפותרן:

1. כל קורס ישובץ ליום אחד בדיוק
2. קורסים מתנגשים לא ישובצו לאותו יום, כלומר, בשני קורסים - i ו-j - שהבחינות בהם מתנגשות (לפי הגדרה #1) נרצה להוסיף פסוקיות CNF אשר מבטאות את האילוץ שהבחינה בקורס i אינה משובצת באותו היום עם הבחינה בקורס j. כך נעשה לכל זוג קורסים שבו הבחינות מתנגשות.

במשימה זו נכתוב שתי פונקציות, הנחוצות כדי להמיר את הקלט לנוסחת CNF.

<u>משימה 9.1:</u> השלימו את הגדרת הפונקציה בקובץ Task9Encode.java:

```
public static boolean[][]findExamConflicts(int m,int[][]
studentCourses)
```

הפונקציה מקבלת את מספר הקורסים ואת מצב הרישום לקורסים ומחזירה מערך בוליאני דו-ממדי examConflicts שגודלו $m \times m$. מערך זה מוגדר כך ש examConflicts[i][j] == true אם הקורס שמספרו i והקורס שמספרו j מתנגשים (לפי הגדרה #1); אחרת, examConflicts [i][j] == false.

שימו לב כי לפי ההגדרה, קורס אינו מתנגש עם עצמו, ולכן examConflicts[i][i] == false לכל i.

במשימה זו יש להניח כי הקלט תקין.

לדוגמה, בהנחה שישנם 6 קורסים ו-5 סטודנטים הרשומים לקורסים לפי המערך הבא:

```
int [][] studentCourses = { {0,1,3,4}, {1,2,3}, {5,1}, {0,1,2,3,4},
{2,3,1}};
```

על הפונקציה להחזיר את המערך הבוליאני הדו-ממדי הבא:

```
{{false,  true, true,  true,  true,  false},
 {true,  false, true,  true,  true,  true },
 {true,  true, false,  true,  true,  false},
 {true,  true, true,  false,  true,  false},
 {true,  true, true,  true,  false,  false},
 {false, true, false, false, false,  false}}
```

<u>משימה 9.2:</u> השלימו את הגדרת הפונקציה בקובץ Task9Encode.java:

```
public static void convertInput(int[][] variableNames, String[]
students,String[] courses, int[][] studentCourses, int k)
```

הפונקציה מקבלת טבלה של משתני CNF כפי שנבנה במשימה 8 ומופע של בעיית שיבוץ הבחינות. הפונקציה מוסיפה את פסוקיות CNF לפותרן כפי שהוסבר. יש להניח כי הפותרן כבר אותחל עם מספר המשתנים המתאים ($m \times k$). במשימה זו יש להניח תקינות קלט.

משימה 10:

במשימה זו נגדיר ממיר פלט מבעיית הסיפוק הבוליאני לבעיית שיבוץ הבחינות. בהינתן השמה ל-$m \times k$ המשתנים בטבלה (int[][] variableNames = variableTable(m,k), נרצה לבנות פתרון לבעיית שיבוץ הבחינות. כפי שהצגנו במשימות 2 ו-3, פתרון לבעיית שיבוץ הבחינות מיוצג באמצעות מערך int[] schedule. השלימו את הגדרת הפונקציה בקובץ Task10Decode.java:

```
public static int[] convertOutput(int[][] variableNames,
boolean[] assignment)
```

הפונקציה מקבלת כקלט את טבלת משתני CNF שיצרנו במשימה 8 ואת ההשמה שהתקבלה מהפותרן. הפונקציה מחזירה פתרון לבעיית שיבוץ הבחינות המיוצג ע"י מערך של מספרים, כאשר הערך באינדקס ה-i מציין את היום שבו משובצת הבחינה בקורס ה-i. שימו לב כי על השיבוץ המוחזר להיות עקבי עם ההשמה למשתני הבעיה.

במשימה זו יש להניח תקינות קלט. בפרט יש להניח כי assignment היא השמה מספקת למשתני ה-CNF ששמותיהם מופיעים במערך variableNames.

משימה 11:

במשימה זו נכתוב פונקציה הפותרת את בעיית שיבוץ הבחינות (או קובעת כי אין פתרון).

<u>משימה 11.1:</u> השלימו את הגדרת הפונקציה בקובץ Task11Solve.java :

```
public static int[] solveETP(String[] students, String[]
courses, int[][] studentCourses, int k)
```

הפונקציה מקבלת כקלט מופע לבעיית שיבוץ הבחינות ופועלת לפי השלבים הבאים:

1. מוודאת את תקינות המופע. אם המופע אינו תקין יש לזרוק חריגת IllegalArgumentException עם הסבר המציין כי המופע אינו תקין.
2. אחרת, מייצרת טבלת משתנים מתאימה למופע.
3. מאתחלת פותרן לבעיית הספיקות.
4. מקודדת את המופע, באמצעות טבלת המשתנים, לפסוקיות CNF ומוסיפה את הפסוקיות לפותרן
5. מפעילה את הפותרן[2]. אם אין השמה מספקת, על הפונקציה להחזיר מערך ריק כפלט.
6. אם יש השמה מספקת, על הפונקציה להמיר את ההשמה לפתרון עבור בעיית שיבוץ הבחינות. פתרון זה הוא הערך המוחזר כפלט מהפונקציה, במידה שבדיקת החוקיות בסעיף הבא עברה בהצלחה.
7. מוודאת שהשיבוץ חוקי. אם השיבוץ אינו חוקי יש לזרוק חריגת RuntimeErrorException, עם הודעה שמעידה כי אירעה שגיאה פנימית בתכנית והתקבל שיבוץ שאינו חוקי.

<u>משימה 11.2:</u> במשימה זו, בהינתן מופע של בעיית שיבוץ הבחינות **ללא הערך k**, נחפש את הערך של ה-k המינימלי כך שיהיה אפשר לשבץ את הבחינות ב-k ימים. השלימו את הגדרת הפונקציה בקובץ Task11Solve.java :

[2] כפי שהוסבר בסעיף "עיקרי הממשק של ה-SAT Solver", הפותרן מחזיר: א. השמה מספקת (בייצוג של מערך בוליאני) אם נמצאה כזאת; או ב. מערך ריק אם לא קיימת השמה מספקת. יש גם מצב שלישי: אם תוך שלוש דקות הפותרן אינו מצליח לפתור את המופע, הוא עוצר את החיפוש ומחזיר ערך null. אתם יכולים להתעלם מהמצב השלישי - לא נבדוק מקרים כאלו.

```
public static int[] solveMinDaysETP(String[] students,
String[] courses, int[][] studentCourses)
```

הפונקציה מקבלת מופע של בעיית שיבוץ הבחינות, ללא הפרמטר k. על הפונקציה להחזיר שיבוץ בחינות עם מספר ימים מינימלי k. בהרה: שיבוץ בחינות הוא עם מספר ימים מינימלי k אם ורק אם לא ניתן לשבץ את הבחינות ל- k-1 ימים. שימו לב כי תמיד יש פתרון לבעיית השיבוץ כאשר k=m, ולעולם לא יהיה פתרון כאשר k=0 (בהנחה ש- m>0).

על הפונקציה solveMinDaysETP לוודא את תקינות המופע כפי שהוגדר במשימה 0.

בהצלחה