עבודת בית מספר 2

מערכים, פונקציות ובעיית הסיפוק הבוליאני

מבוא למדעי המחשב, סמסטר א' תשע"ח

המחלקה למדעי המחשב, אוניברסיטת בן-גוריון בנגב

מבוא למדעי המחשב – סמסטר א' תשע"ח

עבודת בית מספר 2: מערכים, פונקציות ובעיית הסיפוק הבוליאני

צוות העבודה:

מרצה אחראי: מיכאל קודיש

מתרגלים אחראים: נועה בן דוד ומיכאל פרנק

תאריך פרסום: 9.11.2017

תאריך הגשה: 01.12.2017 בשעה 12:00 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 שפותרת מופעים של בעיית הסדוקו על-ידי 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 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 מצורפים קובצי Java עם שמות כגון Task<n><Description>.java, כאשר <n> מציין את מספר המשימה המתאימה לקובץ (לדוגמה, קובץ Java בשם Task5Verify.java מתאים למשימה מספר 5). בנוסף מצורף קובץ בשם TasksArrays.java למשימות 1-4. צרו תיקייה חדשה והעתיקו את קובצי ה-Java לתוכה. עליכם לערוך את הקבצים האלו בהתאם למפורט בתרגיל ולהגישם כפתרון, מכווצים כקובץ ZIP יחיד. שימו לב: עליכם להגיש רק את קובצי ה-Java. אין לשנות את שמות הקבצים. אין להגיש קבצים נוספים. 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 וכד'..) לא יתקבל.
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. 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.
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 11 משימות וסך הנקודות המקסימלי הוא 110. הניקוד לכל משימה שווה (10 נקודות) ואינו מצביע על קושי המשימה.
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.

חלק 1: מבוא לסודוקו

מופע של בעיית הסודוקו הקלאסית נתון כלוח בגודל 9×9 שבו חלק מהתאים מכילים מספרים שלמים שערכם בין 1 ל-9 והאחרים הם תאים ריקים. התאים שבהם נתונים מספרים נקראים רמזים. בלוח הקלאסי יש 9 שורות, 9 עמודות ו-9 בלוקים בגודל 3×3, כמתואר באיור בהמשך.

כדי לפתור מופע של בעיית הסודוקו יש למלא בכל התאים הריקים שבלוח מספרים שלמים שערכם בין 1 ל-9, כך שבכל שורה, בכל עמודה ובכל בלוק, כל הערכים יהיו שונים זה מזה.

לוח "מלא", שבו בכל תא נמצא ערך מספרי, נקרא פתרון לבעיית הסודוקו אם המספרים שבלוח מקיימים את התנאים הבאים (נקרא לתנאים אלו בהמשך תנאים א – ה):

א. כל הערכים בתאים הם שלמים וערכם בין 1 ל-9
ב. בכל שורה של הלוח, כל הערכים שונים זה מזה
ג. בכל עמודה של הלוח, כל הערכים שונים זה מזה
ד. בכל בלוק של הלוח, כל הערכים שונים זה מזה
ה. כל הרמזים הנתונים במופע של הבעיה מופיעים בתאים התואמים של הפתרון.

האיור הבא מציג מופע של בעיית הסודוקו בגודל 9×9 (משמאל) ופתרון של מופע זה (מימין). הרמזים מסומנים בשחור. הבלוקים מופרדים בקווים מודגשים.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

בעבודה זו נעסוק בבעיית הסודוקו הכללית: מופע של בעיית הסודוקו מסדר $\sqrt{n}$ נתון כלוח בגודל $n \times n$ שבו חלק מהתאים (הרמזים) מכילים ערכים שלמים בין *1* ל-*n*. בעיית הסודוקו הקלאסית היא מסדר 3 והלוח הקלאסי הוא בגודל 9×9. פתרון לבעיית הסודוקו מסדר $\sqrt{n}$ נדרש לקיים את התנאים א – ה, כאשר הערכים שבתאים הם בין 1 ל-n.

למשל, האיור הבא מציג מופע מסדר 2 (משמאל) ופתרון של מופע זה (מימין). הרמזים מסומנים בשחור והבלוקים מופרדים בקווים מודגשים.

2			
		1	
	2		
			4

2	1	4	3
3	4	1	2
4	2	3	1
1	3	2	4

<u>ייצוג מופע סודוקו בג'אווה</u>

<u>פתרון</u> של מופע לבעיית הסודוקו מסדר $\sqrt{n}$ ייוצג ב-Java באמצעות מערך דו-ממדי של מספרים שלמים `int[][] board` מגודל $n \times n$. בכל תא ב-board יופיע ערך בין 1 ל-n כך שתנאי הפתרון א-ה מתקיימים.

דוגמה: הפתרון של המופע באיור האחרון (מימין) ייוצג באופן הבא (ערכי הרמזים מודגשים):

```
int[][] board= {{2, 1, 4, 3},
                {3, 4, 1, 2},
                {4, 2, 3, 1},
                {1, 3, 2, 4}} ;
```

<u>מופע</u> של בעיית הסודוקו מסדר $\sqrt{n}$ ייוצג ב-Java באמצעות מספר שלם `int sqrtN`, בו נתון הערך של $\sqrt{n}$, ומערך דו-ממדי של מספרים שלמים `int[][] hints` שבו נתונים הרמזים. איברי המערך `hints` הם מערכים באורך שלוש, אשר מכילים את מיקום הרמז ואת ערכו. למשל,

אם `hints[k] = {i, j, value}` אז במיקום ה- (i, j) של הלוח מופיע הערך value. נוכל לתאר מופע ללא רמזים כאשר hints הוא המערך הריק.

דוגמה: המופע באיור האחרון (משמאל) ייוצג ב-Java באופן הבא:

```
int sqrtN = 2 ;
int[][] hints = {{0,0,2}, {1,2,1}, {2,1,2}, {3,3,4}}
```

כלומר, מדובר במופע מסדר 2, כך שבתא (0,0) מופיע הערך 2, בתא (1,2) מופיע הערך 1, בתא (2,1) מופיע הערך 2 ובתא (3,3) מופיע הערך 4. שאר התאים ריקים.

מטרת החלק הראשון של עבודה זו היא להגדיר ולממש כמה פונקציות שיסייעו לנו לבדוק אם מטריצה נתונה היא פתרון חוקי של מופע לבעיית הסודוקו.

<u>משימה 1:</u>

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

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

על הפונקציה לוודא כי כל ערכי המערך array הם מספרים שונים זה מזה. יש להניח כי array אינו null.

דוגמאות:

```
int[] array1 = {1,9,3,6} ;
System.out.println(isAllDiff (array1)); // true ;

int[] array2 = {1,4,3,5,1,6} ;
System.out.println(isAllDiff (array2)); // false ;

int[] array3 = {} ;
System.out.println(isAllDiff (array3)); // true ;
```

<u>משימה 2:</u>

בהינתן מערך דו-ממדי של מספרים שלמים, נרצה לוודא כי המערך מייצג מטריצה בגודל $n \times n$ המכילה רק מספרים בטווח נתון. השלימו את הפונקציה הבאה בקובץ TasksArrays.java:

public static boolean isMatrixBetween(int[][] matrix, int n, int min, int max)

על הפונקציה להחזיר ערך true אם ורק אם matrix הוא מערך המכיל n מערכים באורך n, וערכיו הם בין min ל-max (כולל). אין להניח דבר על תקינות הקלט, יש להחזיר ערך false אם הקלט אינו תקין.

דוגמאות:

```
System.out.println(isMatrixBetween(null, 4, 1, 5)); // false

int[][] matrix1 = {{1,2,3},
                   {1,2,3},
                   {1,2,3}};
System.out.println(isMatrixBetween(matrix1, 3, -2, 9)); // true

int[][] matrix2 = {{1,2,3},
                   null,
                   {1,2,3}};
```

```
System.out.println(isMatrixBetween(matrix2, 3, -2, 9)); // false

int[][] matrix3 = { {1,2,3},
                    {10,11,12},
                    {1,2,3}};
System.out.println(isMatrixBetween(matrix3, 3, -2, 9)); // false
```

<u>משימה 3:</u>

בהינתן מערך דו-ממדי של מספרים שלמים, שאיבריו הם ה"שורות" של מטריצה, נרצה לחשב מערך דו-ממדי תואם, שאיבריו הם ה"עמודות" של המטריצה. השלימו את הפונקציה הבאה בקובץ TasksArrays:

public static int[][] columns(int[][] matrix)

הפונקציה מקבלת מערך דו-ממדי matrix שממדו הראשון מייצג שורות של מטריצה, כלומר, `matrix[0]` מייצג את השורה העליונה של המטריצה, `matrix[1]` מייצג את השורה השנייה של המטריצה וכן הלאה. על הפונקציה להחזיר מערך דו-ממדי שאיבריו הם העמודות של אותה מטריצה, לפי הסדר שלהן משמאל לימין. כלומר, אם `matrixCols` הוא המערך שחוזר מקריאה לפונקציה `columns(matrix)` אזי `matrixCols[0]` היא העמודה השמאלית של `matrix`, `matrixCols[1]` היא העמודה השנייה משמאל של `matrix` וכן הלאה.
יש להניח כי הקלט `matrix` אינו null ומייצג מטריצה מגודל $matrix.length \times matrix.length$.

דוגמאות:

```
int[][] matrix1 =  {{1,2,3},
                    {4,5,6},
                    {7,8,9}};
int[][] matCols1 = columns (matrix1) ;
/*
 * matCols1 = {{1, 4, 7},
 *             {2, 5, 8},
 *             {3, 6, 9}}
 */
```

<u>משימה 4:</u>

בהינתן מערך דו-ממדי של מספרים שלמים, שהם השורות של מטריצה, נרצה לחשב מערך דו-ממדי של הבלוקים המתאימים במטריצה. השלימו את הפונקציה הבאה בקובץ TasksArrays:

public static int[][] blocks(int[][] matrix, int sqrtN)

הפונקציה מקבלת מערך דו-ממדי matrix אשר מייצג מטריצה מגודל $sqrtN^2 \times sqrtN^2$ ומספר sqrtN המייצג את הגודל של כל בלוק (כלומר, כל בלוק מגודל $sqrtN \times sqrtN$). על הפונקציה להחזיר מערך דו-ממדי שאיבריו הם הבלוקים של matrix מסודרים משמאל לימין ומלמעלה למטה. יש להניח כי $sqrtN \geq 2$, וכי matrix אינה null ומייצגת מטריצה מגודל $matrix.length \times matrix.length$.

דוגמה:

```
int[][] matrix1 =  {{11,12,13,14},
                    {15,16,17,18},
                    {19,20,21,22},
                    {23,24,25,26}} ;
int[][] matBlocks1 = blocks (matrix1, 2) ;
/*
 * matBlocks1 = {{11, 12, 15, 16},
 *               {13, 14, 17, 18},
 *               {19, 20, 23, 24},
 *               {21, 22, 25, 26}}
 */
```

<u>משימה 5:</u>

בהינתן פתרון למופע מסדר $\sqrt{n}$ של בעיית הסודוקו, נרצה לוודא כי הפתרון הוא חוקי; כלומר על הפתרון לקיים את תנאים א – ה. השלימו את הפונקציה הבאה בקובץ Task5Verify:

public static boolean isSolution(int sqrtN, int[][] hints, int[][] board)

הפונקציה מקבלת כקלט מופע של בעיית הסודוקו מסדר $\sqrt{n}$ המיוצג ע"י המספר sqrtN ומערך הרמזים hints. כמו כן, הפונקציה מקבלת פתרון, המיוצג ע"י המערך board. על הפונקציה להחזיר true אם board מייצג פתרון חוקי למופע המיוצג ע"י sqrtN ו-hints; כלומר, יש להחזיר true אם board מקיים את תנאים א – ה, אחרת הפונקציה תחזיר ערך false. יש להניח כי sqrtN הוא מספר אי-שלילי, וכי hints הוא מערך של שלשות של מספרים המייצגים אינדקסים וערכים חוקיים למופע הסודוקו. לא ניתן להניח דבר על תקינות board. במידה ש-board אינו תקין, כלומר, board אינו מייצג מטריצה מגודל $sartN^2 \times sqrtN^2$ עם ערכים בין 1 ל-$sqrtN^2$, יש לזרוק חריגה.

דוגמה:

```
int[][] hints1 = {{0,0,2}, {1,2,1}, {2,1,2}, {3,3,4}} ;
int[][] board1 = {{2, 1, 4, 3},
                  {3, 4, 1, 2},
                  {4, 2, 3, 1},
                  {1, 3, 2, 4}} ;
System.out.println(isSolution(2, hints1, board1)); // true
System.out.println(isSolution(3, hints1, board1)); // exception

int[][] hints2 = {{0,0,2}, {1,2,1}, {2,1,2} } ;
int[][] board2 = {{2, 1, 1, 3},
                  {3, 4, 1, 2},
                  {4, 3, 2, 1},
                  {1, 3, 2, 1}} ;
System.out.println(isSolution(2, hints2, board2)); // false
```

חלק 2: בעיית 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: $\boldsymbol{x_1, x_2, \dots, x_n}$.

- 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.

מטרת חלק 2 של עבודה זו היא לקודד כמה אילוצים ל-cnf ולהשתמש בפותרן לבעיית הספיקות.

<u>משימה 6:</u>

במשימה זו 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 6.1:</u> 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 Task6Cnf**:**

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.

במשימה זו יש להניח שהקלט תקין.

<u>משימה 6.2:</u> 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 Task6Cnf**:**

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.

במשימה זו יש להניח שהקלט תקין.

<u>משימה 6.3:</u> 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 Task6Cnf**:**

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.

<u>השימוש בפותרן לבעיית הסיפוק הבוליאני</u>

הוראות בנוגע להתקנה ולשימוש בפותרן לבעיית הסיפוק הבוליאנית ניתן למצוא בנספח שבסוף העבודה. כמו כן, ניתן למצוא כמה דוגמאות לאילוץ exactlyOne בנספח ובקובץ ExamplesTask6ExactlyOne.java המצורף לקבצי העבודה.

חלק 3: רדוקציה מבעיית הסודוקו לבעיית הסיפוק הבוליאני

הרדוקציה מבעיית הסודוקו לבעיית הספיקות הבוליאנית כוללת שלושה שלבים עיקריים:

1. <u>מיפוי (Mapping):</u> בהינתן מופע של בעיית הסודוקו מסדר $\sqrt{n}$, נרצה ליצור התאמה (מיפוי) בין משתני המופע לבין המשתנים שיופיעו בנוסחת ה-CNF אשר מקודדת את המופע.

<u>משתני המופע</u>: נגדיר את משתני המופע להיות מהצורה $x_{i,j,k}$ לכל $0 \leq i,j,k < n$, כאשר המשתנה $x_{i,j,k}$ יקבל את הערך true אם ורק אם בפתרון למופע הערך $k+1$ מופיע בתא ה-(i,j). לדוגמה: עבור לוח סודוקו מסדר 2 יהיו $4^3 = 64$ משתני מופע, ולמשל המשתנה $x_{0,2,3}$ יקבל ערך true אם ורק אם נמצא פתרון כך שבתא ה-(0,2) של לוח הסודוקו מופיע הערך 4.

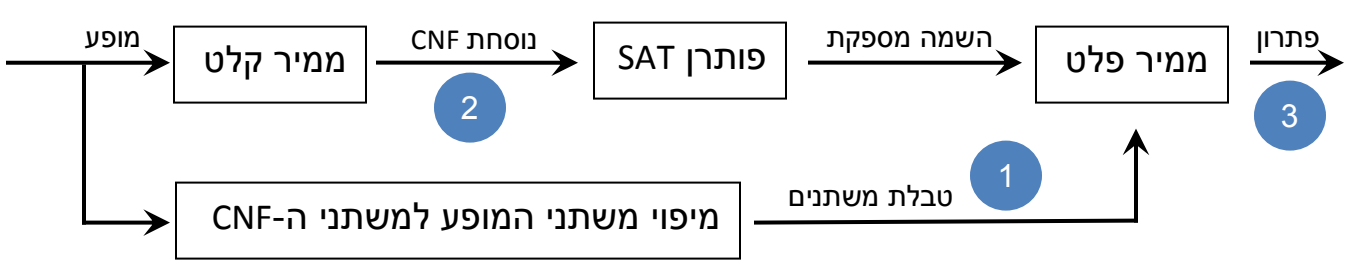
<u>משתני ה-CNF</u>: (שמות) משתני ה-CNF הם מספרים שלמים 1,2,3, ... לדוגמה: עבור לוח סודוקו מסדר 2, נגדיר $4^3 = 64$ משתנים בוליאניים שהשמות שלהם הם 1,2,3, ... ,64.

<u>מיפוי</u>: במשימת המיפוי נגדיר התאמה בין משתני המופע לבין משתני ה-CNF. את שמות המשתנים התואמים למשתני המופע $x_{i,j,k}$ נחשב ונתחזק במערך תלת-ממדי מגודל $n \times n \times n$, אשר ייקרא "טבלת המשתנים" (map).

2. <u>ממיר קלט (Encode):</u> לאחר מיפוי משתני המופע למשתני CNF יש להגדיר נוסחת CNF, כך שכל השמה מספקת של הנוסחה תתאים לפתרון חוקי של המופע. כלומר, יש לאלץ את משתנה ה-CNF התואם למשתנה $x_{i,j,k}$ להיות true בהשמה מספקת אם ורק אם בפתרון למופע מופיע הערך $k+1$ בתא ה-(i,j).

3. <u>ממיר פלט (Decode):</u> לאחר הפעלת פותרן לבעיית הספיקות על ה-CNF שנוצר בשלב 2, ייתכנו שני מצבים. (א) הפותרן מחזיר השמה מספקת, ויש "לתרגם" אותה בעזרת טבלת המשתנים לפתרון של המופע; (ב) אין השמה מספקת, כלומר, אין פתרון למופע זה. אם אין פתרון, יש לדווח שזה המצב ולסיים (אין מה לתרגם חזרה).

האיור הבא ממחיש את שלושת השלבים:



משימה 7: (שלב 1: מיפוי המשתנים)

משימה 7.1:

כזכור, שם משתנה CNF הוא מספר טבעי גדול ממש מ-0. בהינתן $0 \le i,j,k < n$ המייצגים משתנה $x_{i,j,k}$ של מופע של בעיית סודוקו מסדר $\sqrt{n}$, נרצה לחשב את שם משתנה ה-CNF המתאים לו ב-Java. שם המשתנה יהיה מספר בין 1 ל-n^3. במקרה של בעיית הסודוקו, שם המשתנה המתאים ל-i,j,k יחושב (בעבודה זו) לפי הנוסחה: $varName(i,j,k,n) = n^2 * i + n * j + k + 1$. השלימו את הגדרת הפונקציה הבאה בקובץ Task7Map:

```
public static int varName(int i, int j, int k, int n)
```

הפונקציה מקבלת כקלט שלשה של מספרים i,j,k ומספר n ומחזירה את שם המשתנה המתאים ל-i,j,k ו-n לפי הנוסחה המתוארת מעלה. יש להניח כי הקלט תקין, כלומר $0 \le i,j,k < n$.

משימה 7.2:

בהינתן שם משתנה CNF ומספר n נרצה לחשב את שלשת המספרים $0 \le i,j,k < n$ המגדירים את שם המשתנה. השלימו את הגדרת הפונקציה הבאה בקובץ Task7Map (זו הפונקציה ההופכית ל-varName):

```
public static int[] nameToIndex(int x, int n)
```

אשר מקבלת כקלט שם משתנה x ומספר n ומחזירה מערך בעל שלושה איברים {i,j,k} כך ש-`varName(i,j,k,n) == x`. יש להניח כי הקלט תקין, כלומר $0 < x \le n^3$.

דוגמה:

```
int n = 9;
int x = varName(3,4,5,n); // x== 9^2 * 3 + 9 * 4 + 5 + 1 == 285
int[] triplet = nameToIndex(x,9); // triplet == {3,4,5}
```

משימה 7.3:

השלימו את הגדרת הפונקציה הבאה בקובץ Task7Map:

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

אשר מקבלת כקלט מספר אי-שלילי n ומחזירה את טבלת המשתנים המתאימה למופע מסדר $\sqrt{n}$. זהו מערך תלת-ממדי מגודל $n \times n \times n$ כך שבמיקום ה-(i,j,k) של המערך מופיע שם משתנה ה-CNF המייצג את המשתנה $x_{i,j,k}$ לפי הנוסחה הנתונה בסעיף 7.1. יש להניח כי הקלט תקין. כלומר, כי $\sqrt{n}$ מספר שלם.

דוגמה:

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

```
/*
 * map =
 * [[[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]],
 *  [[17, 18, 19, 20], [21, 22, 23, 24], [25, 26, 27, 28], [29, 30, 31, 32]],
 *  [[33, 34, 35, 36], [37, 38, 39, 40], [41, 42, 43, 44], [45, 46, 47, 48]],
 *  [[49, 50, 51, 52], [53, 54, 55, 56], [57, 58, 59, 60], [61, 62, 63, 64]]]
 */
```

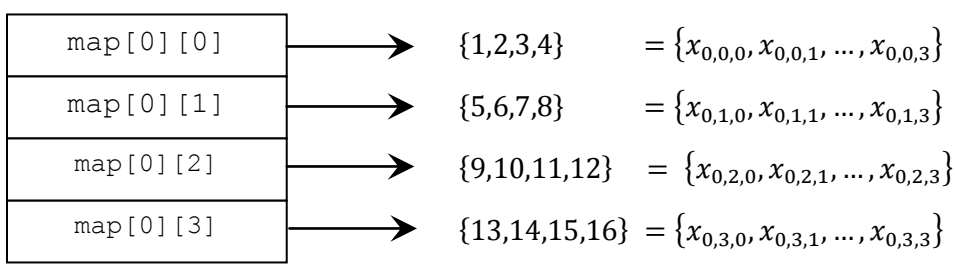
<u>משימה 8: (שלב 2: ממיר קלט)</u>

בהינתן מופע של בעיית הסודוקו מסדר $\sqrt{n}$ עם רמזים $hints$, נרצה לאלץ את המשתנה המופע $x_{i,j,k}$ באופן כזה שהמשתנה יציין אם בתא ה-(i,j) של הפתרון מופיע הערך $k+1$. כלומר, נרצה ש-$x_{i,j,k}$ יקבל ערך true בהשמה מספקת אם ורק אם בפתרון מופיע הערך $k+1$ בתא ה-(i, j). ואת כל זה נרצה לבטא ב-CNF, במונחים של משתני ה-CNF התואמים כפי שהוגדרו במשימה 7.

נתמקד במקרה בו $\sqrt{n} = 2$, כלומר $n = 4$, ונניח כי map היא טבלת המשתנים המתקבלת מקריאה לפונקציה varsMap(4).

<u>אילוצים על התאים</u>: נניח ש-`map[i][j] = {k1, k2, k3, k4}`. אלו שמות של 4 משתנים בוליאניים המציינים את הערך בתא (i,j). אם `k1` יקבל ערך true בהשמה מספקת, כלומר $x_{i,j,0} = true$, אז בתא (i,j) יופיע הערך 1. אם k2 יקבל ערך true, כלומר $x_{i,j,1} = true$, אז בתא (i,j) יופיע הערך 2, וכך הלאה. נרצה לבטא ב-CNF את העובדה שבדיוק אחד המשתנים ב-map[i][j] מקבל ערך true.

<u>אילוצים על השורות</u>: כיצד ייראה, למשל, האילוץ המגדיר כי ערכי השורה הראשונה של המטריצה שונים זה מזה? המשתנים המגדירים את ערכי השורה הראשונה נמצאים ב-map[0]. כאשר map[0][0] מגדיר את הערך הנמצא בתא הראשון של השורה הראשונה, map[0][1] מגדיר את הערך בתא השני של השורה הראשונה, וכך הלאה. באיור:



בצד שמאל באיור מופיעים התאים בטבלת המשתנים המתאימים לשורה הראשונה, באמצע מופיעים שמות משתני ה-CNF המתאימים להם, ומצד ימין - שמות משתני המופע המתאימים. כדי לבטא כי הערכים שיתקבלו בשורה הראשונה שונים זה מזה, יש לדרוש כי בדיוק אחד מהם יהיה שווה ל-1, בדיוק אחד מהם יהיה שווה ל-2, בדיוק אחד מהם יהיה שווה ל-3 ובדיוק אחד מהם יהיה שווה ל-4. כלומר, יש לדרוש, למשל, כי מבין $x_{0,0,0}, x_{0,1,0}, x_{0,2,0}, x_{0,3,0}$ יהיה בדיוק משתנה אחד אשר יקבל ערך true. יש להוסיף אילוצים כאלו למשתנים המתאימים לכל שורה של המטריצה.

שאר האילוצים: יש להוסיף אילוצים כאלו לכל שורה, עמודה ובלוק של טבלת המשתנים. בנוסף, יש להוסיף אילוצים המחייבים תאים בהם מופיע רמז לקבל את הערך של הרמז.

השלימו את הגדרת הפונקציה בקובץ Task8Encode:

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

הפונקציה מקבלת מופע לבעיית הסודוקו מסדר $sqrtN$ עם רמזים $hints$ וטבלת משתנים map. על הפונקציה להוסיף לפותרן לבעיית הספיקות (דרך הממשק המתואר בנספח) את הפסוקיות המתאימות, כך שהשמה מספקת למשתני map תתאים לפתרון חוקי של המופע הנתון ע"י sqrtN ו-hints. יש להניח כי הקלט תקין; כלומר, sqrtN הוא מספר טבעי גדול מ-1, hints מערך של שלשות (בדומה להנחות של משימה 5) ו-map טבלת משתנים שהתקבלה ע"י קריאה לפונקציה varsMap. מומלץ להגדיר פונקציות עזר (רמז: אפשר לקבל השראה מהפונקציות שהוגדרו בחלק 1).

משימה 9: (שלב 3: ממיר פלט)

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

משימה 9.1:

בשלב ראשון, נרצה לפענח ערך של תא אחד בלוח הסודוקו בהתאם להשמה מספקת. השלימו את הגדרת הפונקציה הבאה בקובץ Task9Decode:

```
public static int cellValue(int[][][] map, int i, int j, boolean[] assignment)
```

הפונקציה מקבלת את טבלת המשתנים map המתקבלת כתוצאה מקריאה לפונקציה varsMap, ובנוסף הפונקציה מקבלת מיקום בלוח המיוצג ע"י שני אינדקסים (i,j) והשמה assignment למשתנים ב-map. על הפונקציה להחזיר את הערך המתאים לתא ה-(i,j), כפי שיופיע בפתרון לפי טבלת המשתנים map וההשמה assignment. אם ההשמה ב- assignment אינה נותנת ערך true לאף אחד מהמשתנים התואמים למשתני המופע x_{ijk}, אז יש לזרוק חריגה.

ניתן להניח כי הקלט תקין; כלומר, כי i ו-j אינדקסים חוקיים, map התקבל ע"י קריאה ל-varsMap, ו-assignment מכילה השמה לכל המשתנים.

משימה 9.2:

כעת נרצה לתרגם השמה מספקת עבור מופע של בעיית הסודוקו ללוח הפתרון. השלימו את הגדרת הפונקציה הבאה בקובץ Task9Decode:

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

הפונקציה מקבלת את טבלת המשתנים map והשמה assignment למשתנים ב-map. על הפונקציה להחזיר מטריצה board בגודל $n \times n$ כך ש-board[i][j] מכיל את הערך המפוענח בתא (i,j). יש להניח כי הקלט תקין, כלומר n הוא מספר חיובי גדול מ-1, map הוא מערך שהתקבל כתוצאה מקריאה ל-varsMap(n) ו-assignment היא השמה מלאה למשתני ה-CNF המופיעים ב-map.

משימה 10: (קיים פתרון)

לבסוף, אנו מוכנים לחבר את חלקי העבודה יחדיו ולכתוב פותרן לבעיית הסודוקו באמצעות רדוקציה לבעיית הספיקות הבוליאנית.

השלימו את הגדרת הפונקציה הבאה בקובץ Task10Solve:

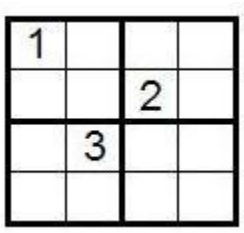
```
public static int[][] solve(int sqrtN, int[][] hints)
```

הפונקציה מקבלת כקלט מופע של בעיית הסודוקו מסדר $sqrtN$ עם רמזים $hints$. 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 board
  - יש לוודא כי board הוא פתרון חוקי למופע שקיבלתם (ע"י קריאה ל-isSolution ממשימה 5)
  - אם board הוא פתרון חוקי, יש להחזיר את board
  - אחרת, יש לזרוק חריגה שמציינת שהפתרון אינו חוקי
- אם אין השמה מספקת יש להחזיר null
- אם המופע היה בלתי פתיר עקב מגבלות זמן (timeout), יש לזרוק חריגה. פרטים נוספים על מגבלות זמן ניתן למצוא בנספח לעבודה זו.

משימה 11: (קיים פתרון יחיד)

מופע סודוקו נקרא מוגדר היטב אם קיים לו פתרון יחיד. למשל, ללוח הסודוקו באיור מטה ישנם 3 פתרונות אפשריים, ולכן הוא לא מוגדר היטב, וללוח הסודוקו באיור שבחלק 1 ישנו פתרון אפשרי יחיד.



השלימו את הגדרת הפונקציה הבאה בקובץ Task11Unique:

```
public static int[][] solveUnique(int sqrtN, int[][] hints)
```

הפונקציה מקבלת כקלט מופע של בעיית הסודוקו מסדר $sqrtN$ עם רמזים $hints$. על הפונקציה להחזיר פתרון למופע הנתון ע"י $sqrtN$ ו-$hints$ אם ורק אם יש לו פתרון יחיד. אם ישנם שני פתרונות, יש להחזיר null. אם חלה טעות בזמן אימות נכונות הפתרון, יש לזרוק חריגה, בדומה למשימה הקודמת.

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:

- <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 "SAT" אם הנוסחה מסתפקת, ו-"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:

מבקשת השמה מספקת מהפותרן, מדפיסה "SAT" אם הנוסחה מסתפקת, ו-"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. הקובץ ExamplesUNSAT.java מכיל כמה דוגמאות נוספות של נוסחאות שאינן מסתפקות.

5. הקובץ ExamplesTask6ExactlyOne.java מכיל כמה דוגמאות נוספות של שימוש באילוץ exactlyOne (המוגדר במשימה 6) בשילוב עם הפותרן.

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.

בהצלחה