מבוא למדעי המחשב – סמסטר א' תשע"ט

עבודת בית מספר 3 (רקורסיה, מבוא לתכנות מונחה עצמים)

צוות העבודה:

מרצה אחראי: חן קיסר

מתרגלים אחראים: ירין קופר, דינה סבטליצקי

תאריך פרסום: 22.11.2018

תאריך הגשה: 6.12.2018, עד השעה 12:00 בצהריים

הוראות מקדימות

הוראות מקדימות:

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).

הגשת עבודות בית

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. את קובץ ה-ZIP עם פתרון לעבודה יש להגיש ב-Submission System. פרטים בעניין ההרשמה ואופן הגשת העבודה תוכלו למצוא באתר.

בדיקת עבודות הבית

4. עבודות הבית נבדקות גם באופן ידני וגם באופן אוטומטי.
5. הבדיקה האוטומטית מתייחסת לערכי ההחזרה של הפונקציות או לפעולות אשר הן מבצעות וכן לפלט התכנית

המודפס למסך (אם קיים). 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.
7. לעבודה מצורפים קבצים KQueens.java, Bit.java. עליכם לערוך קבצים אלה בהתאם למפורט בתרגיל ולהגישם כפתרון, מכווצים כקובץ ZIP יחיד. שימו לב: עליכם להגיש רק את קבצי ה-Java. אין לשנות את שמות הקבצים, ואין להגיש קבצים נוספים. שם קובץ ה-ZIP יכול להיות כרצונכם, אך באנגלית בלבד. בנוסף, הקובץ שתגישו יכול להכיל טקסט המורכב מאותיות באנגלית, מספרים וסימני פיסוק בלבד. טקסט אשר יכיל תווים אחרים (אותיות בעברית, יוונית וכד'..) לא יתקבל.
8. בנוסף לקבצי ה-java, תיקיית ה-ZIP צריכה לכלול את קובץ readme.txt עם השם שלכם ומספר ת.ז. שלכם במקום המתאים בקובץ.
9. מותר להוסיף לכל קובץ java פונקציות עזר.
10. שימו לב כי המילה package לא מופיעה בקבצי ה-java שהגשתם.
11. **לעבודה זו מצורפים קבצי בדיקה שיעזרו לכם בבדיקת תקינות הקוד. מומלץ להוסיף בדיקות נוספות כרצונכם.**

עזרה והנחיה

12. 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.
13. בכל בעיה אישית הקשורה בעבודה (מילואים, אשפוז וכו'), אנא פנו אלינו דרך מערכת הפניות, כפי שמוסבר באתר הקורס.

הערות ספציפיות לעבודת בית זו

14. לעבודה זו מצורפים קבצי Java עם השמות הנדרשים כמפורט בכל משימה. 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 הנדרשים.
15. בעבודה זו ניתן להגדיר פונקציות (עזר) נוספות, לפי שיקולכם. פונקציות אלו ייכתבו בתוך קובצי המשימה הרלוונטיים.
16. בחלק א' של העבודה **אין להגדיר משתנים מחוץ לפונקציה** (משתנים גלובליים). משתני עזר יש להגדיר רק ב-scope של פונקציה.

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 א: בעיית k המלכות

בבעיה זו, נתונות k מלכות ונתון לוח משבצות אשר גודלו $n \times m$, כאשר $n \geq 1, m \geq 1, k \geq 1$. לוח המשבצות הנתון יכול להכיל משבצת אחת או יותר המסומנת כ**מחיצה**. הפרמטרים n, m ו-k ומיקומי המחיצות הם קלט לבעיה.

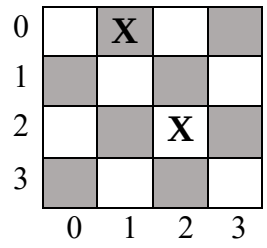
אנו אומרים שמלכה הממוקמת על גבי הלוח **מאוימת** על ידי מלכה אחרת הממוקמת על הלוח, אם שתיהן ממוקמות על אותה שורה, על אותה עמודה או על אותו אלכסון בלוח. יש למקם על גבי הלוח את k המלכות כך שאף מלכה לא תהיה מאוימת. כלומר, לוח שבו יש מלכה המאוימת על ידי מלכה אחת או יותר אינו חוקי.

משבצת בלוח הנתון יכולה להכיל **מחיצה**. מחיצה שנמצאת בין שתי מלכות, מונעת מהן לאיים אחת על השנייה. אי אפשר למקם מלכה במשבצת בה קיימת מחיצה.

עבור לוח משבצות בגודל 1×1 ועבור $k = 1$, לוח המשבצות כולל משבצת יחידה. אם משבצת זו מכילה מחיצה, לא ניתן למקם עליה מלכה ולכן אין פתרון. אם המשבצת ריקה, יש למקם מלכה אחת בלבד ולכן ישנו רק פתרון אחד אפשרי.

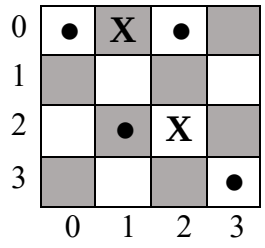
עבור $k = 4$ והלוח הנתון:

לוח א'

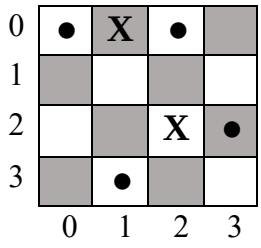


- הפתרונות הבאים הם **פתרונות חוקיים** לבעיה:

לוח א' – פתרון 1



לוח א' – פתרון 2

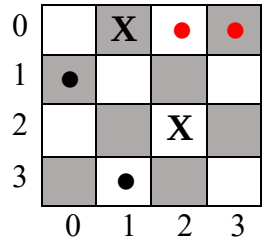


מלכה מיוצגת על ידי הסימן ●, מחיצה מיוצגת על ידי הסימן **X**. בכל לוח, כל אחת מהמלכות לא מאוימת כלל – בין כל שתי מלכות שנמצאות באותה שורה, עמודה או אלכסון מפרידה מחיצה.

לדוגמה: בלוח הימני (פתרון 1), המלכה במשבצת **[0,0]** והמלכה במשבצת **[3,3]** ממוקמות על אותו אלכסון, אך המחיצה במשבצת **[2,2]** מונעת מהן לאיים אחת על השנייה.

- הפתרון הבא **אינו חוקי**, מכיוון שהמלכות המסומנת באדום מאיימות אחת על השנייה – שתיהן נמצאות באותה שורה ואין מחיצה המפרידה ביניהן.

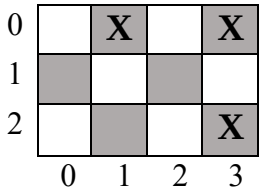
לוח א' – פתרון לא חוקי



שימו לב:

קיימים לוחות וערכי k עבורם אין פתרון אפשרי. למשל עבור $k = 4$ והלוח הנתון, כל ניסיון למקם **4** מלכות על גבי הלוח יביא למצב בו כל מלכה מאוימת על ידי מלכה אחרת.

לוח ב'



הקדמה: ייצוג לוח משבצות ופתרון מוצע עבור בעיית K המלכות

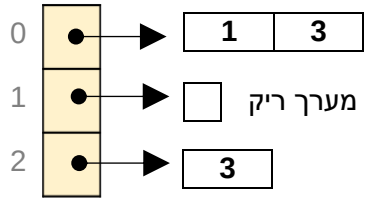
קלט: מספרים שלמים n, m, k ומיקומי מחיצות.

פלט: לוח משבצות בגודל $n \times m$ המכיל מחיצות במשבצות המתאימות (לפי המיקומים הנתונים) – בו ממוקמות k מלכות בצורה חוקית כך שאין שתי מלכות המאיימות אחת על השנייה. אם אין פתרון חוקי, הפלט יהיה לוח משבצות ריק (ללא משבצות).

ייצוג הקלט – מיקומי המחיצות

מיקומי המחיצות מיוצגים באמצעות מערך דו-מימדי של שלמים. עבור כל מחיצה, המערך יכיל מידע של מיקום המחיצה (השורה והעמודה של המשבצת עליה ממוקמת המחיצה). אינדקס התא במערך בממד הראשון מתייחס לשורה בלוח המשבצות (אינדקס 0 מתייחס לשורה 0, אינדקס 1 מתייחס לשורה 1 וכך הלאה). הערך בכל תא במערך בממד השני מתייחס לעמודה בלוח המשבצות.

לדוגמא, המערך הדו-מימדי הבא מייצג את המחיצות בלוח ב':



בשורה 0 בלוח המשבצות המתאים, קיימת מחיצה בעמודה 1 וקיימת מחיצה בעמודה 3. בשורה 1 אין מחיצות. בשורה 2 קיימת מחיצה בעמודה 3.

שימו לב: מספר התאים במערך בממד הראשון הוא כמספר השורות בלוח המשבצות. אם אין מחיצות בשורה מסוימת, התא המתאים במערך יכיל מערך ריק.

ייצוג לוח משבצות המכיל מחיצות

בהינתן מערך דו-מימדי של מחיצות, נרצה לבנות לוח משבצות ללא מלכות, המכיל מחיצות. את לוח המשבצות נייצג באמצעות מערך דו-ממדי בגודל $n \times m$ המכיל ערכים מטיפוס **int**.

ערך 0 מייצג תא ריק בלוח, וערך 1- מייצג תא בו ממוקמת מחיצה.

לדוגמא, עבור המחיצות הנתונות, נבנה את הלוח הבא (לוח ב' בדוגמה לעיל):

לוח ב'

	0	1	2	3
0	0	-1	0	-1
1	0	0	0	0
2	0	0	0	-1

ייצוג פתרון מוצע עבור בעיית k המלכות

על מנת לייצג פתרון מוצע עבור הבעיה נשתמש במערך דו-ממדי בגודל $n \times m$ המכיל ערכים מטיפוס **int**.

ערך 0 מייצג תא ריק בלוח, וערך 1 מייצג תא שבו ממוקמת מלכה, ערך 1- מייצג תא בו ממוקמת מחיצה.

לדוגמא, את שני הפתרונות בדוגמה עבור לוח א', נייצג על ידי המערכים הבאים:

לוח א' – פתרון 1

	0	1	2	3
0	1	-1	1	0
1	0	0	0	0
2	0	1	-1	0
3	0	0	0	1

לוח א' – פתרון 2

	0	1	2	3
0	1	-1	1	0
1	0	0	0	0
2	0	0	-1	1
3	0	1	0	0

שימו לב: עבור המקרה בו אין פתרון לבעיה, נייצג את הפתרון על ידי מערך ריק.

<u>הדרכת חובה:</u>

- במשימות הבאות תצטרכו לממש את הקוד של הפונקציות המתאימות במחלקה KQueens, בקובץ KQueens.java.
- אין לשנות את החתימות של הפונקציות הנתונות
- בתחילת הקוד במחלקה KQueens, בקובץ KQueens.java נתונים הקבועים הבאים:

```
final static int QUEEN = 1;
final static int WALL = -1;
final static int EMPTY = 0;
```

במימוש הפונקציות, יש להשתמש בקבועים אלה כאשר יש התייחסות לערכים של לוח המשבצות. קבועים אלה ישפרו את קריאות הקוד, ולא יהיה צורך לזכור את הערכים המספריים של מיקום מלכה/קיר/משבצת ריקה על הלוח.

דוגמה: בכל פונקציה בקובץ זה, תוכלו להשתמש במשתנים אלו.

```
1. int[][] board1 = {{EMPTY, WALL},{EMPTY, QUEEN}};
2. int[][] board2 = {{0, -1},{0, 1}};
```

שורות 1 ו-2 זהות מבחינת הפעולה המתבצעת (בשתי השורות נוצר מערך דו-ממדי עם אותם הערכים), אך בשורה 1 הקוד קריא יותר. בשורה 2 המשמעות של הערכים 1,0,1- אינה מובנת.

משימה 1: בדיקת תקינות הקלט (8 נק')

השלימו את הגדרת הפונקציה

```
public static boolean isValidInput(int k, int rows, int cols,
                                   int[][] walls)
```

במחלקה KQueens, בקובץ KQueens.java.

הפונקציה מקבלת 4 פרמטרים:

- k: מספר המלכות שיש למקם על לוח המשבצות
- rows: מספר השורות הדרוש בלוח המשבצות
- cols: מספר העמודות הדרוש בלוח המשבצות
- walls: מערך דו-מימדי המייצג מחיצות. הערך walls[i][j] קובע כי בלוח המשבצות בשורה i ובעמודה walls[i][j] קיימת מחיצה (ראו הסבר על מערך המחיצות)

פונקציה זו תחזיר את הערך הבוליאני true אמ"ם הפרמטרים הנ"ל תקינים, כלומר:

- $rows \geq 1$
- $cols \geq 1$
- $k \geq 1$
- walls אינו null
- walls מכיל $rows$ תאים
- ערך כל תא ב-walls אינו null
- walls מכיל ערכים למיקומי מחיצות שאינם חורגים מלוח המשבצות
- $k \leq$ מספר המשבצות הפנויות על הלוח (משבצות שאין עליהן מחיצות)

שימו לב: ניתן להניח שבכל מערך walls[i] לכל i, הערכים במערך שונים זה מזה (כלומר, אין צורך לבדוק זאת)

משימה 2: בניית לוח בגודל $n \times m$ המכיל מחיצות (8 נק')

השלימו את הגדרת הפונקציה

```
public static int[][] createBoard(int rows, int cols,
                                  int[][] walls)
```

במחלקה KQueens, בקובץ KQueens.java.

הפונקציה מקבלת 3 פרמטרים:

- rows: מספר השורות הדרוש בלוח המשבצות
- cols: מספר העמודות הדרוש בלוח המשבצות
- walls: מערך דו-מימדי המייצג מחיצות. הערך walls[i][j] קובע כי בלוח המשבצות בשורה i ובעמודה walls[i][j] קיימת מחיצה (ראו הסבר על מערך המחיצות)

על הפונקציה להחזיר מערך דו-מימדי שייצג לוח משבצות עם rows שורות ו-cols עמודות, המכיל מחיצות במיקומים המתאימים (לפי הערכים במערך walls). ראו הסבר על ייצוג לוח המשבצות.

שימו לב: במימוש הפונקציה, ניתן להניח שכל הפרמטרים תקינים. כלומר, לפני הקריאה לפונקציה זו, הייתה קריאה לפונקציה `isValidInput` עם הפרמטרים הנ"ל, ופונקציה זו החזירה את הערך true.

משימה 3: ייצוג פתרון מוצע והדפסתו (0 נק')

השלימו את הגדרת הפונקציה `public static void printBoard(int[][] board)` במחלקה KQueens, בקובץ KQueens.java.

הפונקציה מקבלת כקלט מערך דו-ממדי board של ערכים מטיפוס int המייצג פתרון מוצע כפי שמוסבר לעיל. על הפונקציה להדפיס את מיקומי המלכות והמחיצות על גבי הלוח בהתאם לייצוג במערך. לכל משבצת בלוח, יודפס התו 'Q' אם ישנה מלכה במשבצת, יודפס התו 'X' אם ישנה מחיצה במשבצת, והתו '*', אחרת.

לדוגמא, עבור הפתרון השמאלי של לוח א' (פתרון 2), הפונקציה תדפיס:

```
Q X Q *
* * * *
* * X Q
* Q * *
```

את ההדפסה יש לבצע בהתאם להנחיות הבאות:

- בתחילת כל שורה לא יופיע רווח.
- לאחר כל תו יופיע רווח יחיד.
- השורות תופענה ברצף אחת מתחת לשניה מבלי שתהיה שורת רווח בין זוג שורות.

שימו לב:

- במקרה שהפתרון המתקבל הוא מערך ריק על הפונקציה להדפיס את ההודעה הבאה, ללא רווחים בתחילתה או בסופה:

There is no solution

- במימוש הפונקציה, ניתן להניח שהפרמטר board תקין, כלומר:
  - board אינו null
  - כל תא ב-board אינו null
  - אם board אינו ריק, הוא מכיל ערכים תקינים

משימה 4: בדיקת חוקיות של פתרון מוצע (24 נק')

א. בדיקה האם הלוח הנתון מכיל מחיצות באותם המשבצות שניתנו כקלט. (8 נק')
המטרה של סעיף זה, הוא שתוכלו לוודא כי אחרי מיקום המלכות על הלוח, לא "נדרסו" בטעות תאים בהם היו מחיצות.
השלימו הגדרת הפונקציה הבאה במחלקה KQueens, בקובץ KQueens.java:

```
public static boolean validateWalls(int[][] walls, int[][]
board)
```

הפונקציה מקבלת 2 פרמטרים:

- walls: מייצג את מיקומי המחיצות בלוח המשבצות (ראו הסבר על מערך המחיצות).
- board: מייצג לוח משבצות עם מחיצות (ראו הסבר על ייצוג לוח המשבצות).

על הפונקציה להחזיר את הערך הבוליאני true אם:
- כל מיקום מחיצה ב-walls אכן נמצאת על הלוח board במיקום המתאים
- כל מיקום מחיצה בלוח board נמצא ב-walls

אחרת, עליה להחזיר false.

שימו לב:
- ניתן להניח כי walls הוא פרמטר חוקי. כלומר:
  - walls אינו null
  - כל תא ב-walls אינו מכיל ערך null
  - כל מערך walls[i] לכל i מכיל ערכים תקינים שאינם חורגים מגודל הלוח board, וכל הערכים שונים זה מזה.
- אין להניח דבר על board. אם board אינו תקין, על הפונקציה להחזיר false.

ב. בדיקה האם מלכה בתא ספציפי בלוח מאוימת על ידי מלכה אחרת על הלוח. (8 נק')

השלימו הגדרת הפונקציה הבאה במחלקה KQueens, בקובץ KQueens.java:

```
public static boolean isQueenThreatened(int[][] board, int row,
                                        int col)
```

הפונקציה מקבלת 3 פרמטרים:
- board: מייצג לוח דו-ממדי בו ממוקמת המלכה.
- row: מייצג את השורה בה ממוקמת המלכה.
- col: מייצג את העמודה בה ממוקמת המלכה.

על הפונקציה להחזיר את הערך הבוליאני true אם ורק אם המלכה בתא board[row][col] מאוימת על ידי מלכה אחרת על הלוח.

- במימוש הפונקציה, ניתן להניח כי:
  - board אינו null
  - כל תא ב-board אינו null
  - board אינו מערך ריק
  - board מכיל ערכים תקינים
  - board[row][col] מייצג תא שאינו חורג מהלוח
  - בתא board[row][col] אכן ממוקמת מלכה

דוגמה: אם נעביר לפונקציה את הלוח הבא כפרמטר:

0	1	-1	1	0
1	0	0	0	0
2	0	0	-1	1
3	1	0	0	0
	0	1	2	3

ואת הפרמטרים $row = 3, col = 0$:

הפונקציה תחזיר true מכיוון שהמלכה הממוקמת בשורה 3 ועמודה 0 מאוימת על ידי המלכה הממוקמת בשורה 0 ועמודה 0.

ואילו עבור אותו הלוח והפרמטרים $row = 0, col = 2$:
הפונקציה תחזיר false מכיוון שהמלכה הממוקמת בשורה 0 ועמודה 2 לא מאוימת ע"י אף מלכה.

ג. השלימו הגדרת הפונקציה הבאה במחלקה KQueens, בקובץ KQueens.java: (8 נק')

```
public static boolean isLegalSolution(int[][] board, int k,
                                      int rows, int cols, int[][] walls)
```

הפונקציה מקבלת 5 פרמטרים:

- board: מערך דו-ממדי בגודל $n \times m$ המייצג את הפתרון המוצע עבור $\boldsymbol{k}$ מלכות.
- k: מייצג את מספר המלכות שצריכות להיות ממוקמות בלוח.
- rows: מייצג את מספר השורות שהלוח צריך להכיל.
- cols: מייצג את מספר העמודות שהלוח צריך להכיל.
- walls: מערך דו מימדי שמייצג את המחיצות שצריכות להיות ממוקמות על הלוח.

על הפונקציה להחזיר את הערך הבוליאני true אם ורק אם הפתרון המוצע הוא חוקי. כלומר, board הוא לוח עם rows שורות ו-cols עמודות, יש k מלכות הממוקמות על הלוח, המחיצות על הלוח תואמות למחיצות ב-walls, ואין אף מלכה המאוימת על ידי מלכה אחרת על הלוח.

שימו לב:

- יש לבדוק כי הערכים ב-board תקינים (מייצגים תא ריק, מלכה או מחיצה), אחרת על הפונקציה להחזיר false.
- במימוש הפונקציה, ניתן להניח כי:
  - board אינו null
  - כל תא ב-board אינו null
  - board אינו מערך ריק
  - $k \geq 1$
  - rows≥ 1
  - cols≥ 1
  - walls אינו null
  - walls מכיל $rows$ תאים
  - ערך כל תא ב-walls אינו null
  - walls מכיל ערכים למיקומי מחיצות שאינם חורגים מלוח המשבצות

<u>הדרכת חובה</u>: על הפונקציה בסעיף זה לקרוא לפונקציות העזר המוגדרות בסעיפים א' ו-ב' לצורך ביצוע המשימה.

הערה: פונקציה זו תעזור לכם לבדוק את הפתרון לבעיית K המלכות שתקבלו כפלט אחרי שתממשו את משימה 6.

דוגמאות:

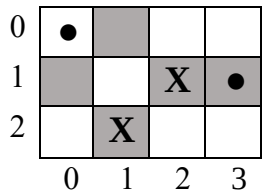
```
int [][] board = {{QUEEN,EMPTY,EMPTY},{EMPTY,EMPTY,QUEEN}};
```

```
int [][] walls = new int[2][0];
boolean isLegal = isLegalSolution(board, 2, 2, 3, walls);
System.out.println(isLegal);  // prints "true"

int [][] board2 = {{QUEEN,3,EMPTY},{EMPTY,EMPTY,QUEEN}};
isLegal = isLegalSolution(board2, 2, 2, 3, walls);
System.out.println(isLegal);  // prints "false"

int [][] board3 = {{QUEEN,WALL,QUEEN},{EMPTY,WALL,QUEEN}};
int [][] walls2 = {{1},{1}};
isLegal = isLegalSolution(board3, 3, 2, 3, walls2);
System.out.println(isLegal);  // prints "false"

int [][] board4 = {{QUEEN,WALL,QUEEN},{EMPTY,EMPTY,EMPTY}};
int [][] walls3 = {{1},{}};
isLegal = isLegalSolution(board4, 2, 2, 3, walls3);
System.out.println(isLegal);  // prints "true"

isLegal = isLegalSolution(board4, 3, 2, 3, walls3);
System.out.println(isLegal);  // prints "false"

isLegal = isLegalSolution(board4, 2, 2, 4, walls3);
System.out.println(isLegal);  // prints "false"
```

משימה 5: הוספת מלכה אל לוח נתון (8 נק')

הגדרה: **פתרון חלקי עבור $k' < k$ מלכות** הוא לוח בגודל $n \times m$ ובו מוצבות k' מלכות כך שהלוח מהווה פתרון עבור k' מלכות הממוקמות על הלוח הנתון. כלומר, כל מלכה המוצבת על הלוח אינה מאוימת על ידי מלכה אחרת המוצבת על הלוח.

שימו לב: לא ניתן בהכרח להשלים כל פתרון חלקי עבור $k' < k$ מלכות לפתרון מלא עבור k מלכות.
דוגמה לכך הוא הפתרון החלקי עבור $k' = 2$ מלכות המתואר על ידי הציור הבא. לא ניתן להוסיף מלכה ללוח ולקבל פתרון חוקי לבעיה עבור $k = 3$ מלכות.



השלימו את הגדרת הפונקציה הבאה במחלקה KQueens, בקובץ KQueens.java:

```
public static boolean addQueen(int[][] board, int row, int col)
```

הפונקציה מקבלת 3 פרמטרים:

- board: מערך דו-ממדי המייצג פתרון חלקי עבור k' מלכות ($k' < k$)
- row: מספר שלם, המייצג אינדקס שורה
- col: מספר שלם, המייצג אינדקס עמודה

על הפונקציה להוסיף לפתרון החלקי הקיים ב- board מלכה בשורה row ובעמודה col אך ורק אם הדבר אפשרי (כלומר רק אם הלוח המתקבל מהוספה זו הוא פתרון חוקי עבור k'+1 מלכות). על הפונקציה להחזיר את הערך true אם ניתן היה לבצע את ההוספה ואת הערך false אחרת.

שימו לב:

- הפונקציה משנה את המערך board
- במימוש הפונקציה ניתן להניח שהפתרון החלקי ב-board מהווה פתרון חלקי עבור k' מלכות
- במימוש הפונקציה, ניתן להניח שכל הפרמטרים תקינים. כלומר:
  - board אינו null
  - כל תא ב-board אינו null
  - board אינו מערך ריק
  - כל תא ב-board מכיל ערך חוקי
  - board[row][col] הוא תא שאינו חורג מגודל המערך
- הערך בתא board[row][col] יכול להיות ריק, להכיל מלכה או להכיל מחיצה

הדרכת חובה:

- **על הפונקציה בסעיף זה לקרוא לפונקציה `isQueenThreatened` המוגדרת במשימה 4ב.**
- **במימוש הפונקציה אסור לקרוא לפונקציה `isLegalSolution` המוגדרת במשימה 4ג.**

משימה 6: שימוש ברקורסיה לפתרון בעיית K המלכות (20 נק')

א. השלימו את הגדרת הפונקציה

```
public static int[][] kQueens(int k, int rows,
                              int cols, int[][] walls)
```

במחלקה KQueens, בקובץ KQueens.java:

הפונקציה מקבלת כפרמטרים:

- k : מספר שלם שמציין את מספר המלכות שיש למקם על הלוח
- rows: מספר שלם שמציין את מספר השורות של הלוח
- cols: מספר שלם שמציין את מספר העמודות של הלוח
- walls: מערך דו-מימדי של מיקומי המחיצות על הלוח

על הפונקציה להחזיר מערך דו-ממדי בגודל $\boldsymbol{rows} \times \boldsymbol{cols}$ עם מחיצות במקומות המתאימים, המהווה פתרון עבור בעיית $\boldsymbol{k}$ המלכות, אם קיים. אחרת, על הפונקציה להחזיר מערך ריק.

הדרכת חובה:

- **על הפונקציה בסעיף זה לקרוא לפונקציית העזר הרקורסיבית המוגדרת בסעיף הבא לצורך ביצוע המשימה. על הפונקציה בסעיף זה לבצע קריאה בודדת לפונקציה מהסעיף הבא.**
- **על הפונקציה בסעיף זה לקרוא לפונקציה isValidInput המוגדרת במשימה 1 כדי לוודא שהפרמטרים תקינים. אם הם אינם תקינים, על פונקציה זו להחזיר מערך ריק.**

ב. השלימו את הגדרת הפונקציה **הרקורסיבית** הבאה במחלקה KQueens, בקובץ KQueens.java:

```
private static boolean kQueens(int[][] board, int k,
                               int row, int col, int numOfQueens)
```

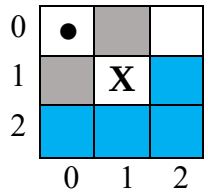
הפונקציה מקבלת 5 פרמטרים:

- board: מערך דו-ממדי המייצג פתרון חלקי עבור numOfQueens מלכות (numOfQueens≤ k)

- k: מספר שלם המייצג את מספר המלכות הסופי שיש למקם בלוח
- row: מספר שלם, המייצג אינדקס שורה
- col: מספר שלם, המייצג אינדקס עמודה
- numOfQueens: מספר המלכות הממוקמות על הלוח עד כה

על הפונקציה לבדוק האם ניתן להשלים את הפתרון החלקי ב- board לפתרון חוקי מלא על ידי:
- הצבת מלכות בתאים board[row][j] עבור j שמתקיים עבורו:
  - $col \leq \boldsymbol{j} \leq board[row].length - 1$
- או הצבת מלכות בתאים board[i][j] עבור i ו-j שמתקיים עבורם:
  - $0 \leq \boldsymbol{j} \leq board[row].length - 1$
  - $row + 1 \leq \boldsymbol{i} \leq board.length - 1$

כלומר, הצבת המלכות תיעשה בתאים הנמצאים מימין לתא board[row][col] כולל, או בשורה עם אינדקס גדול מ-row, כפי שמומחש בדוגמה הבאה.
בקריאה לפונקציה זו עם הלוח הבא והפרמטרים $\mathrm{k} = 2, \mathrm{row} = 1, \mathrm{col} = 2, \mathrm{numOfQueens} = 1$, תוכל להתווסף מלכה רק בשטח המסומן בכחול.



בזמן קריאה לפונקציה זו יש להניח כי:

1. בפתרון החלקי עבור row השורות העליונות 0,…,row-1 ב- board והתאים board[row][j] כאשר $0 \leq j \leq col - 1$ (בדוגמה – השטח שאינו מסומן בכחול), אין מלכה המאוימת על ידי מלכה אחת או יותר – דבר הנובע מעצם ההגדרה של פתרון חלקי.
2. **לא ניתן להשלים את הפתרון החלקי ב- board לפתרון מלא ע"י הצבת מלכה נוספת בשורות** 0,…,row-1 או בתאים board[row][j] כאשר $0 \leq j \leq col - 1$ (בדוגמה – השטח שאינו מסומן בכחול)**.** הנחה זו היא הנחה מנחה בפתרון פונקציה זו.

אם ניתן להשלים את הפתרון החלקי כמפורט לעיל:
- על הפונקציה להחזיר את הערך הבוליאני true ואחרת, false.
- על המערך board אשר התקבל כקלט להיות פתרון לבעיית k המלכות בסיום ריצת הפונקציה.

הדרכת חובה:

- **במימוש הפונקציה הרקורסיבית kQueens, אין להשתמש בלולאות, מלבד המקרים הבאים:**
  - **בפונקציה** `kQueens(int k, int rows, int cols, int[][] walls)` **ניתן להשתמש בלולאה עבור אתחול מערך**
  - **בפונקציה** `kQueens(int[][] board, int k, int row, int col, int numOfQueens)` **ניתן לקרוא לפונקציית עזר איטרטיבית (באופן ישיר או עקיף) המשתמשת בלולאות.**
- **במימוש פונקציה זו, יש לקרוא לפונקציית העזר addQueen שהוגדרה במשימה 5**
- **אין לקרוא לפונקציית isLegalSolution שהוגדרה במשימה 4ג**

חלק ב: המחלקה Bit

בתכנות מונחה עצמים (**Object Oriented Programming**) אנחנו מייצגים רעיונות באמצעות טיפוסי משתנים. בחלק זה של העבודה נגדיר את הטיפוס **Bit** המייצג את הרעיון של ספרה בינארית, שהערך שלה אחד או אפס. לשם כך ניצור מחלקה חדשה. בעבודת הבית הבאה נשתמש בעצמים ממחלקה זו כדי לייצג מספרים גדולים כרצוננו. המחלקה גם תכלול פונקציות סטטיות המבצעות פעולות בסיסיות על ספרות בינאריות. פונקציות אלו ישמשו בעבודה הבאה כאבני בניין לכתיבת פונקציות אריתמטיות.

למחלקה **Bit** יהיה שדה פרטי אחד, **value**, מהטיפוס **boolean** . הספרה **0** תיוצג על ידי עצם שהערך בשדה שלו הוא **false**. הספרה **1** תיוצג על ידי עצם שהערך בשדה שלו הוא **true** .

במשימות להלן תכתבו בנאי, שיטות ופונקציות סטטיות במחלקה **Bit**. אתם מקבלים קובץ בשם **Bit.java** שבו תבניות שאותן עליכם למלא.

<u>הנחייה:</u> מותר להוסיף שיטות ופונקציות נוספות ובלבד שהן תהיינה פרטיות.

```
public class Bit {
    private boolean value;
    public Bit(boolean value) { /* הקוד הזה יושלם במשימה 7 */}
    public String toString() { /* הקוד הזה יושלם במשימה 7 */}
    public boolean isOne() { /* הקוד הזה יושלם במשימה 7 */}
    public boolean lessThan(Bit digit) {/* הקוד הזה יושלם במשימה 8 */}
    public boolean lessEq(Bit digit) {/* הקוד הזה יושלם במשימה 8 */}
    public static Bit fullAdderSum(Bit A, Bit B, Bit Cin) {
        /* הקוד הזה יושלם במשימה 9 */
    }
    public static Bit fullAdderCarry(Bit A, Bit B, Bit Cin) {
        /* הקוד הזה יושלם במשימה 9 */
    }
}
```

משימה 7: כתיבת הבנאי והשיטות toString() ו- isOne() (4 נק')

א. השלימו את הקוד של הבנאי כך שאם הוא מקבל כארגומנט את הערך **false** , הוא יוצר עצם המייצג את הספרה **0**. אחרת, הוא יוצר עצם המייצג את הספרה **1**.

ב. השלימו את הקוד של השיטה **toString()** כך שאם העצם מייצג את הספרה **0**, השיטה תחזיר את המחרוזת **"0"**, אחרת, היא תחזיר את המחרוזת **"1"**. (**2** נק')

ג. השלימו את הקוד של השיטה **isOne()** כך שהיא תחזיר את הערך **true** אם העצם מייצג את הספרה **1**. אחרת יוחזר הערך **false**. (**2** נק')

לדוגמה, התוכנית

```
public static void main(String[] args) {
   Bit bit1 = new Bit(true);
   Bit bit0 = new Bit(false);
   System.out.println(bit0.toString()+" "+bit1.toString()+" "+
                      bit0.isOne()+" "+bit1.isOne());
}
```

תדפיס

0 1 false true

משימה 8: כתיבת השיטות `lessThan(Bit digit)` ו- `lessEq(Bit digit)` (12 נק')

בין שתי הספרות **0** ו- **1** יש יחס סדר. **0** קטנה מ- **1**.

השיטות `lessThan(Bit digit)` ו- `lessEq(Bit digit)` מייצגות יחס זה

א. השלימו את הקוד של השיטה `lessThan(Bit digit)` , המאפשרת לספרה להשוות את עצמה לספרה אחרת. השיטה מחזירה `true` אם ורק אם הארגומנט גדול מהספרה. (**6** נק')

ב. השלימו את הקוד של השיטה `lessEq(Bit digit)` . השיטה מחזירה `true` אם ורק אם ארגומנט גדול מהספרה או שווה לה. (**6** נק')

הנחיה: ניתן להניח שהקלט תקין, כלומר שאינו **null**.

לדוגמה, התוכנית

```
public static void main(String[] args) {
   Bit bit1 = new Bit(true);
   Bit bit0 = new Bit(false);
   System.out.println(bit0.lessEq(bit0)+" "+bit0.lessEq(bit1)+" "+
                      bit1.lessEq(bit0)+" "+bit1.lessEq(bit1));
   System.out.println(bit0.lessThan(bit0)+" "+bit0.lessThan(bit1)+" "+
                      bit1.lessThan(bit0)+" "+bit1.lessThan(bit1));
}
```

מדפיסה

```
true true false true
false true false false
```

משימה 9: כתיבת הפונקציות fullAdderSum ו- fullAdderCarry (16 נק')

חיבור של שלוש ספרות בינאריות, הוא פעולה אריתמטית בסיסית, שהפלט שלה הוא **זוג ספרות**. סכום (sum) ונשא (carry). בסך הכל ייתכנו שמונה שלישיות קלט. שלישיות אלו והפלט של פעולת החיבור שלהן מוצגים בטבלה.

רכיב אלקטרוני שמממש חיבור של שלוש ספרות נקרא full-adder. רכיבים כאלו הם מרכיבים עיקריים במערכות דיגיטאליות ובפרט במחשבים https://en.wikipedia.org/wiki/Adder_(electronics)

a	b	c_{in}	sum	carry
1	1	1	1	1
1	1	0	0	1
1	0	1	0	1
1	0	0	1	0
0	1	1	0	1
0	1	0	1	0
0	0	1	1	0
0	0	0	0	0

א. השלימו את הפונקציה **fullAdderSum(Bit A, Bit B, Bit Cin)** המקבלת שלוש ספרות ומחזירה עצם חדש, ספרת הסכום של חיבורן. (8 נק')

ב. השלימו את הפונקציה **fullAdderCarry(Bit A, Bit B, Bit Cin)** המקבלת שלוש ספרות ומחזירה עצם חדש, ספרת הנשא של חיבורן. (8 נק')

<u>הנחיה:</u> ניתן להניח שהקלט תקין, כלומר שאף אחד מן הפרמטרים אינו **null**.

<u>לדוגמה: התוכנית</u>

```
public static void main(String[] args) {
    Bit bit1 = new Bit(true);
    Bit bit0 = new Bit(false);
    Bit bitC = new Bit(false);
    System.out.println(Bit.fullAdderSum(bit0, bit1, bitC)+" "+
                       Bit.fullAdderCarry(bit0, bit1, bitC));
}
```

תדפיס

```
1 0
```

עבודה נעימה :)