מבני נתונים– תרגיל מספר 4 (מעשי)

תאריך פרסום: 19.5.21

תאריך הגשה: 23:59, 9.6.21 (נתון לשינוי בהתאם למצב הביטחוני)

מתרגל אחראי: נמרוד כהן

הנחיות:

- יש לקרוא תחילה את הנהלים להגשת עבודות במודל.
- שאלות לגבי העבודה יש לשאול בפורום באתר הקורס או בשעות קבלה של נמרוד כהן.
- עליכם להגיש קובץ zip יחיד בשם 'id1_id2.zip', כאשר id1 ו- id2 אלה תעודות הזהות של המגישים. בתוך קובץ ה- zip צריכה להיות תיקייה בשם 'hw4' ובה:
  - קבצי ה-java שהשלמתם: AVL.java, AVLNode.java, HashTable.java ו- StudentSolution.java.
  - קבצי ה-java הנוספים שיצרתם.
- הניחו שהקלט תקין.
- אין לשנות חתימות של מחלקות/פונקציות שאנו מספקים לכם.
- העבודה תיבדק באופן אוטומטי, ולכן אין לשנות את שמות הקבצים או את אופן ההגשה.
- **אין להשתמש בספריות הממשות AVL או Hash Table (כמו java.util.TreeMap ו- java.util.Hashtable), אלא יש לממש בעצמכם.**
- ניתן להשתמש בחבילות אחרות שאינן ממשות מבני נתונים שעליכם לממש בעצמכם.
- ניתן להוסיף שיטות ושדות בקבצים אותם אתם מגישים.
- **נא לקרוא את כל המסמך טרם תחילת העבודה – מומלץ יותר מפעם אחת.**
- **שימו לב – העבודה תיבדק למציאת העתקות (גם אל מול שנתונים של השנים האחרונות)! נא להימנע מכך.**

.

כללי

בעבודה זאת תתנסו בהבנה יסודית של מבני נתונים: עצי AVL (AVL Tree) וטבלאות גיבוב (Hash Table). אתם תשתמשו במבנים שלמדתם בקורס על מנת ליצור כלי ויזואלי שעונה על שאילתות גיאומטריות שנגדיר בעבודה, בדומה לכלי שיש בפלטפורמת Facebook.

בהינתן תמונה המכילה אלמנטים, כך שכל אלמנט מיוצג ע"י נקודה במישור, ובהינתן מלבן מקביל לצירים, מטרת האפליקציה היא לענות בצורה יעילה על השאלה – אלו מן האלמנטים נמצאים בתוך המלבן. לעבודה מצורפים איורים שממחישים את יכולות האפליקציה שתכתבו.

שימו לב שבכתיבת קוד נכונה עליכם לייצר מספר מחלקות שונות, כאשר כל אחת מהן מייצגת אובייקט נפרד. לכן מותר ומומלץ לייצר קבצים (מחלקות) נוספים.

חלק א' – מימוש תשתית על בסיס רעיון עץ AVL

בחלק זה עליכם לכתוב מימוש מלא ב-java של מבנה הנתונים שמבוסס על AVL הכולל הכנסה וחיפוש של ערכים. שימו לב שהמימוש צריך לעמוד בדרישות של החלקים הבאים בעבודה, לכן יש לקרוא את כל העבודה בטרם כתיבת הקוד.

שימו לב: עץ AVL שתממשו מוגדר כמבנה נתונים גנרי, כלומר, המבנה אינו מוגבל לאובייקטים מסוג מסוים. בהגדרת המחלקה יש את הסימון <T> , כאשר ניצור מופע חדש של עץ AVL, נגדיר לו את הסוג של האובייקטים שיקבל. לדוגמה:

```
AVL<Integer> integerAVL = new AVL<Integer>();
```

כעת אם נרצה לבצע הכנסה של איבר מסוג String, תהיה שגיאת קומפילציה בקוד.
ניתן לקרוא עוד על מבנים גנריים כאן:
https://www.tutorialspoint.com/java/java_generics.htm

דוגמה למבנה גנרי של רשימה מקושרת ניתן לראות כאן:
https://gist.github.com/lobster1234/5037064

מלאו את המחלקה AVL, יש להשתמש במחלקה AVLNode , המייצגת קודקוד בעץ.

הערכים שיאוחסנו בעץ יהיו הקואורדינטות של העצמים מהתמונה.
על אופן האחסון, השימוש המדויק והתוספות הדרושות עליכם לחשוב לבד.
הדרישות הן:

א. לממש את התשתית של הכנסה בהתאם לנלמד בכיתה.
ב. לממש את התשתית של חיפוש בהתאם לנלמד בכיתה.
ג. לממש את התוספת כדי לתמוך בדרישות התרגיל בצורה יעילה מבחינת זמן ריצה, יעילות ובהירות הקוד.

- **אין צורך לממש מחיקה**

חלק ב' – מימוש תשתית על בסיס רעיון HASH

בחלק זה עליכם לכתוב מימוש מלא ב-java של טבלת גיבוב שיתמוך בהכנסה בזמן של O(1) במקרה הגרוע וחיפוש בזמן O(1) במקרה הממוצע עפ"י שיטת גיבוב עם שרשור.
עליכם לממש חלק זה בקובץ HashTable.java.

פונקציות הגיבוב תהיה:

$$h(point) = (point.x + point.y) \bmod m$$

כאשר x,y הם קואורדינטות של הנקודה point.

- **אינכם נדרשים לממש את טבלת הגיבוב באופן גנרי.**

ניתן להניח כי אין שתי נקודות עם אותה קואורדינטה x ואין שתי נקודות עם אותה קואורדינטה y.

הממשק ObjectWithCoordinates:

ממשק זה מייצג נקודה במרחב הדו-מימדי בעל קואורדינטות x,y. על מנת לאפשר אחסון אובייקט המייצג נקודה, עליכם ליצור מחלקה המממשת את הממשק.

חלק ג' – מציאת אלגוריתמים לשאילתות טווח מלבני (דו מימדיות) בזמן יעיל

בחלק זה עליכם לכתוב את פעולת ההכנסת של האיברים אל מבנה הנתונים שלכם ו**שני מימושים** של שאילתות טווח דו-מימדיות (מלבניות).
את כל המימושים של סעיף זה יש לממש בקובץ StudentSolution.java במקומות המתאימים. מחלקה זו מממשת את הממשק MyInterface.
הממשק מכיל את שלושת השיטות הבאות:

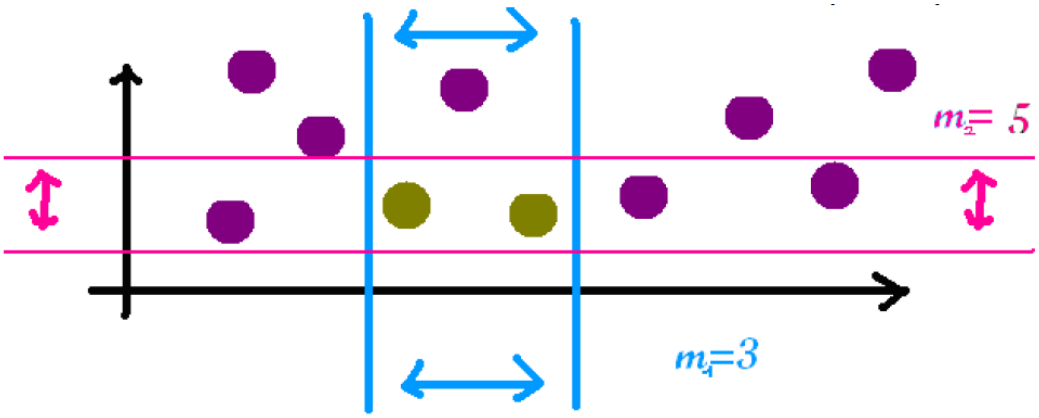
```
public interface MyInterface {

   public void insertDataFromDBFile(String objectName, int objectX, int objectY);

   public String[] firstSolution(int leftTopX, int leftTopY, int rightBottomX, int rightBottomY);

   public String[] secondSolution(int leftTopX, int leftTopY, int rightBottomX, int rightBottomY);

}
```

השיטה הראשונה בממשק היא הכנסת איבר למבנה הנתונים שלכם. שיטה זו נקראת מבחוץ ע"י קובץ הגרפיקה (הסבר בהמשך) בעת טעינת קובץ ה-DB שמכיל את הנקודות.
עבור כל שורה בקובץ ה-DB היא נקראת פעם אחת, כלומר היא מופעלת פעם אחת עבור כל אובייקט, עם הנתונים של אותו אובייקט. עליכם לשמור את הנקודות במבנה הנתונים שלכם כאשר השיטה נקראת. **זמן הריצה הדרוש לפעולה זו הוא O(logn).**

שתי השיטות האחרות מייצגות 2 מימושים לשאילתת טווח דו-מימדית:

נתונים n אובייקטים, לכל אובייקט יש 2 קואורדינטות (x,y), ושם ייחודי.
על מבנה הנתונים לתמוך בשאילתת טווח דו-מימדית, כך שבהינתן טווח של ערכי ציר X וטווח של ערכי ציר Y יש להחזיר את הנקודות שנמצאות במלבן שנוצר.

באיור למטה ניתן לראות דוגמה לפתרון שאילתת טווח דו-מימדית, כאשר הקווים האנכיים בצבע כחול מגדירים את הטווח של שיעורי ה-X של הנקודות, ואילו הקווים הוורודים מגדירים את הטווח של שיעורי ה-Y של הנקודות.
2 הנקודות בצבע חרדל מגדירות את הנקודות הרלוונטיות עבור השאילתה.



המימוש <u>הראשון (השיטה firstSolution)</u> צריך לעבוד לפי הרעיון הבא:

1. הפעל שאילתת טווח חד מימדית על פני ציר ה-x. כלומר מצא m_1 נקודות שנמצאות בטווח אופקי נתון.
2. הפעל שאילתת טווח חד מימדית על פני ציר ה-y. כלומר מצא m_2 נקודות שנמצאות בטווח אנכי נתון.
3. מצא את החיתוך – כלומר מצא את כל הנקודות שנמצאות במלבן הנוצר משתי השאילתות חד מימדיות. עליכם לחשוב כיצד לממש את החיתוך כדי לעמוד בזמני הריצה הדרושים (ראו טבלה מטה).

<u>רמז</u>: עבור שאילתת טווח חד מימדית, היזכרו בתרגול עצי חיפוש בינאריים כיצד אנחנו יכולים למצוא בצורה יעילה טווח של ערכים.

המימוש **<u>השני (השיטה secondSolution)</u>** לשאילתת טווח דו-מימדית צריך לעבוד לפי רעיון אחר עליו עליכם לחשוב לבד. (**רמז**: תבחנו את זמני הריצה הנדרשים עבור השיטות)

הערך המוחזר של שתי הפונקציות הוא מערך של מחרוזות של האובייקטים מסוג ObjectWithCoordinates (צריך להשתמש בשיטת toString על כל אובייקט המתאים לשאילתה).

במידה ולא קיימים אובייקטים, יש להחזיר מערך ריק, בגודל 0.

דרישות זמני ריצה וסיבוכיות מקום:

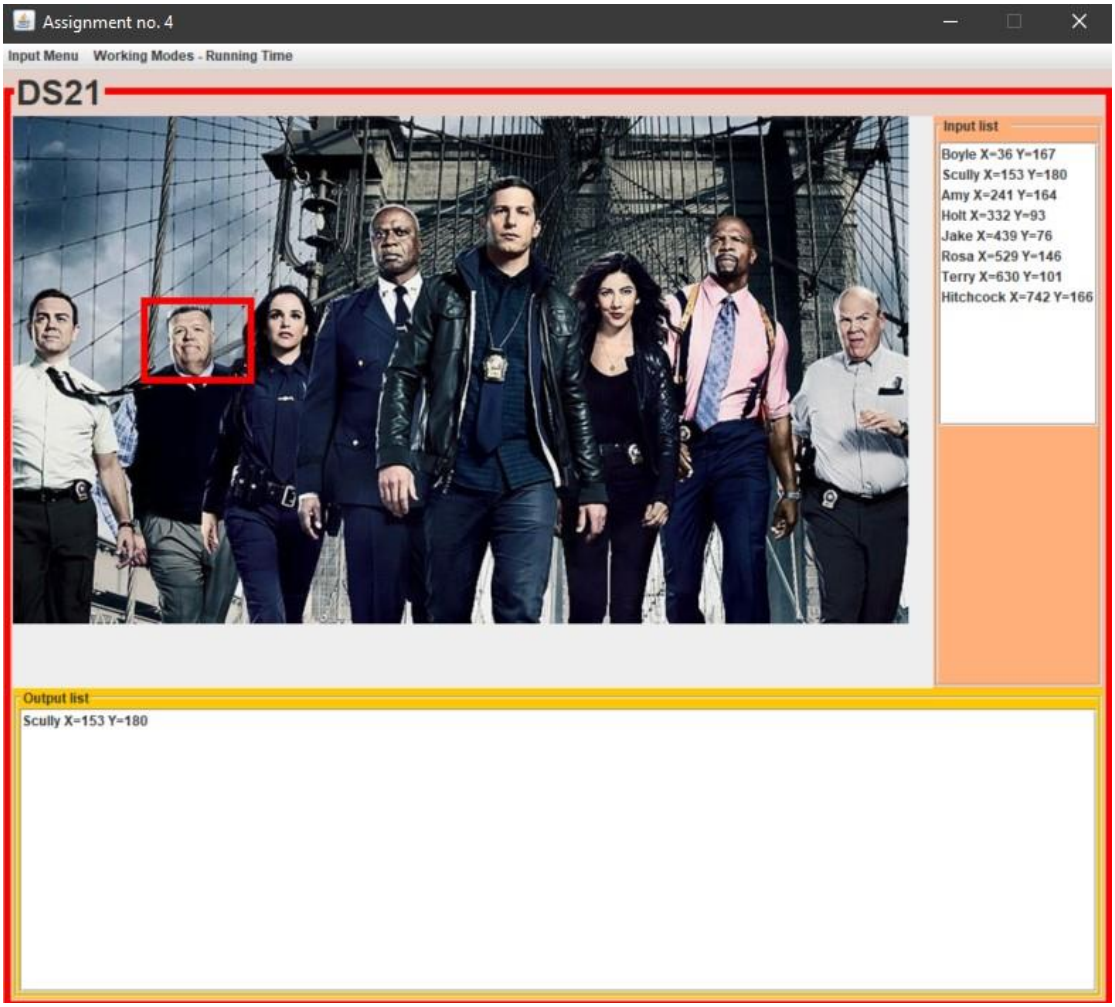
יש לממש עפ"י זמני הריצה הבאים:

מספר מימוש	זמן במקרה הממוצע	זמן במקרה הגרוע	צריכת מקום
1	$O(m_1 + m_2 + logn)$	$O(m_1 * m_2 + logn)$	$O(m_1 + m_2)$
2	$O(min(m_1, m_2) * log(max(m_1, m_2)) + m_1 + m_2 + logn)$		$O(m_1 + m_2)$

- כאשר n זה מספר האיברים הקיימים במבנה בעת ביצוע הפעולה
- יש לשים לב כי דרישת המקום מתייחסת למקום לזיכרון נוסף הדרוש במהלך ביצוע השאילתה (בנוסף למבנה עצמו).
- יצירת מבנה נתונים בגודל n ב-java דורשת זמן O(n) לטובת איפוס המבנה.

הממשק הגרפי:

ממשק זה מהווה את התוצר הסופי המבצע שימוש בחלקים הקודמים, אינכם מחויבים להשתמש בו, הוא מהווה את החלק הכיפי ומיועד לשימושכם.
בהינתן האפליקציה שיודעת לטעון תמונה וקובץ TXT, המייצג בסיס נתונים של העצמים בתמונה, יש לחבר את החלקים הקודמים וליצור את האפליקציה שיודעת להחזיר אילו עצמים מופיעים במלבן אותו אתם מסמנים.
האפליקציה מאוד פשוטה לתפעול. תחילה יש לטעון את התמונה וקובץ ה-TXT שנכתב בפורמט מסוים (ומתואר מטה) המתאים לתמונה מסוימת.
לאחר מכן יש לבחור מלבן באמצעות לחיצה על העכבר וגרירה על פני התמונה. לאחר עזיבת העכבר, הפלט יופיע למטה. ניתן להחליף בין שני האלגוריתמים באמצעות התפריט השני (working modes)



מצורפים לעבודה שני סטים של קבצים. כל סט מכיל תמונה (קובץ jpg) וקובץ DB על מיקומי אובייקטים (קובץ TXT). אתם יכולים להשתמש בהם כדי לבדוק את נכונות האפליקציה.
אתם יכולים לכתוב סטים של קבצים כאשר הפורמט של קובץ הטקסט צריך להיות:

NAME X=x_coord Y=y_coord

הקוד המסופק לכם כולל קובץ GUI.java, המכיל את ממשק המשתמש (קובץ הגרפיקה).
כדי להפעיל את האפליקציה יש להפעיל את הקובץ הזה ואין לבצע בו כל שינוי.

בהצלחה!!!