תכן ותכנות מונחה עצמים – 046271

תרגיל בית 2

מגישים:

שם	תעודת זהות
בנימין סרוסי	311314975
יואב כהן	203115373

שאלה 1

סעיף א' –

הפעולות השונות שאותן בחרנו עבור ההפשטה של גרף הן :

- Add_node
- Add_edge
- Get_nodes
- Get_children_nodes
- Get_graph_name

הפעולות הללו נבחרו אחרי קריאה של התרגיל וההבנה שאנחנו צריכים להיצמד להמשך התרגיל ועל מנת שנוכל למצוא את המסלול הקצר ביותר בגרף בצורה תקינה ונכונה.

אוסף פעולות זה אכן מספק מאחר ואין לנו צורך בפעולות נוספות כמו הסרת צמתים (כמובן למטרת התרגיל).

סעיף ב' –

החלטנו לממש את ההפשטה של גרף על ידי MAP שממפה בין הצמתים לבין סט של צמתים של ילדים של אותה צומת, כלומר במימוש אנחנו מבצעים מיפוי בין צומת לכל הצמתים כך שקיימת קשת ממנו אליהם. בנוסף אנחנו מחזיקים מחרוזת שמציינת את שם הגרף.

השיקול שלנו בבחירת מימוש המבנה הייתה החזרה של הבנים של אותה צומת בזמן קבוע , כך שלאחר מכן בעת המימוש של האלגוריתם למציאת המסלול הקצר ביותר נוכל לבצע זאת ביעילות שנדרש מאיתנו בתרגיל.

כעת נעבור על כל מימוש של כל מתודה ונסביר את בחירת המימוש :

- Add_node - אנחנו מוסיפים את הצומת שקיבלנו כפרמטר על ידי המתודה במיכל של MAP PutIfAbsent. על ידי ערך החזרה של מתודה זו אנו שולטים בניסיון של המשתמש להכניס את אותה צומת פעמיים. במידה והמשתמש אכן מנסה לעשות זאת תיזרק חריגה מתאימה.
- Add_edge - אנחנו מוסיפים קשת בין צומת האב לצומת הבן ,שקיבלנו כפרמטרים, על ידי הוספת צומת הבן לסט ששייך לצומת האב . במידה ואם אחד מהצמתים שקיבלנו לא שייך לגרף תזרק חריגה מתאימה. אם אותה קשת כבר נמצאת בגרף אז תיזרק גם כן חריגה מתאימה.
- Get_nodes – אנחנו מחזירים את סט הצמתים שמרכיב את הגרף. במקרה והגרף לא מכיל צמתים תיזרק חריגה מתאימה. בנוסף אנחנו מחזירים את סט הצמתים כסט unmodifiable כדי למנוע rep exposure.
- Get_children_nodes – המתודה מחזירה סט של צמתים , שמהווים את הבנים של הצומת parent_node שקיבלנו כפרמטר , כלומר את כל הצמתים שקיימת קשת מ-parent_node אל הצמתים המדוברים. במידה והצומת לא קיימת בגרף או שלצומת לא קיימים בנים, תיזרק חריגה מתאימה. גם כאן אנחנו מחזירים unmodifiableSet כדי למנוע rep exposure.
- Get_graph_name – מתודה המחזירה את שם הגרף.

את הגרף יכלנו לממש על ידי מבני נתונים אחרים . לדוגמא רשימות מקושרות תורים וכו'.

הסיבה שבחרנו לייצג את הגרף על ידי map של סטים היא הדרישה לסיבוכיות זמן קבועה בהחזרת הילדים של צומת.

במידה והיינו משתמשים במבני נתונים אחרים . היה עלינו לעבור טרם החזרת הילדים לעבור על רשימת הצמתים, מה שתגרור לסיבוכיות ליניארית בתלות במספר הילדים .

המימוש הספציפי שבחרנו משתמש ב-hashmap ו-hashset מה שנותן לנו גישה לכל צומת וכל איבר בסט בזמן משוערך קבוע.

סעיף ג' –

בבדיקות ה- black box רצינו לבדוק את המפרט שכתבנו על כל מתודה.

בדקנו שכל מתודה אכן מקיימת נכונות , כלומר כל מתודה אכן מבצעת את תפקידה . אפילו מצאנו באגים במימוש שלנו באחת מהבדיקות , אז באמת הבנו כמה החלק של הבדיקות הוא שימושי .

בנוסף בדקנו את כל מקרי הקצה שחשבנו עליהם . למשל במתודה add_node בדקנו מצב של הוספה של אותה צומת פעמיים. ובמקרה זה המתודה תזרוק חריגה. החלטנו שבמקרים כאלה לא ייכתב כלום לקובץ הפלט שלנו. כך הצלחנו להבדיל בקלות במערכת ה-Junit איפה יש שגיאה בין מה שכתבנו במפרט לבין מה שקורה בפועל.

בבדיקת ה- white box רצינו להגיע לpath coverage כמה שיותר גדול .כלומר, רצינו לכסות את כל המסלולים האפשריים בקוד. לכן יצרנו בדיקות שכיסו את כל משפטי התנאי והלולאות במתודות שלנו.

הבדיקות שביצענו אכן מספקות כי בדקנו שכל החריגות נזרקות ושכל הערכים שקיבלנו עומדים בציפיות שלנו בכל הבדיקות.

גם בבדיקות ה-black box ראינו שעמדנו במפרט שחשבנו עליו בכל אחד מהמקרים.

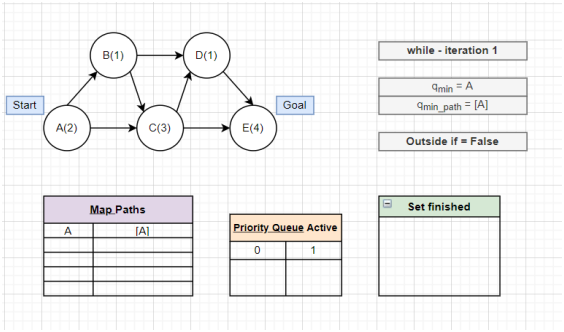
אך כמו שנאמר בתרגול ובהרצאה הבדיקות הן כלי למציאת שגיאות שאנו חושבים עליהן , אך לא כלי למציאת שגיאות שאנחנו לא יכולים לצפות אותן. לכן אף פעם לא ניתן לומר שהבדיקות מספקות אותנו באופן מושלם.

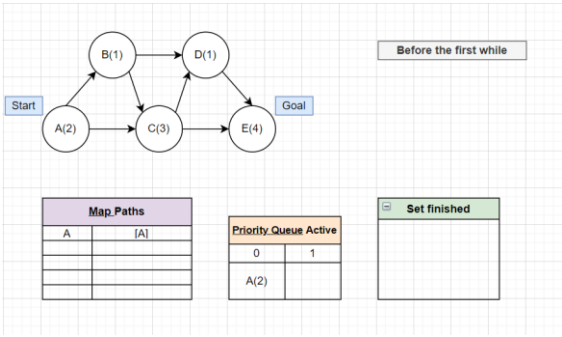
סעיף ד' –

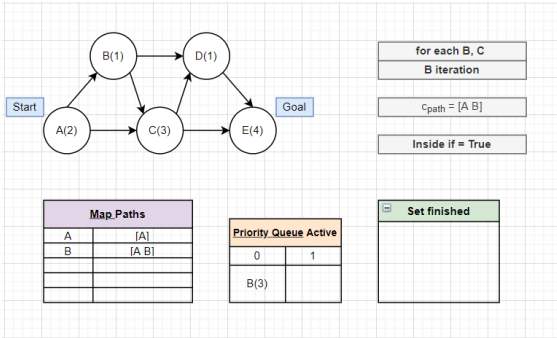
התשובה לשאלה היא שמכיוון שקיים ממשק שגם שמו הוא Path יכולות להיות התנגשויות בין הממשק לבין המחלקה של java עליה אנחנו מדברים בשאלה. לכן אנחנו חייבים לציין את שם המחלקה המלא.

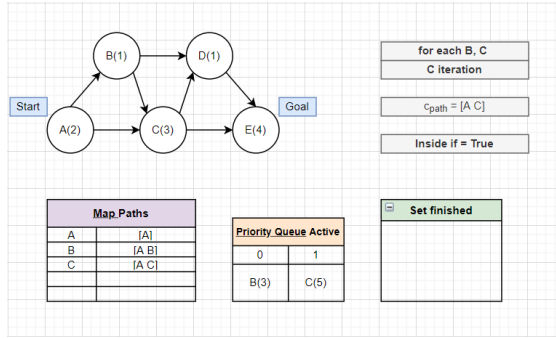
שאלה 2

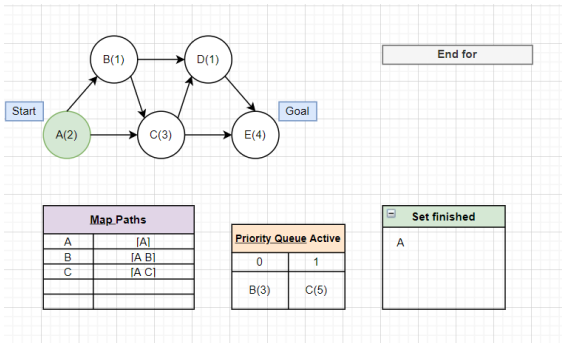
סעיף א' –

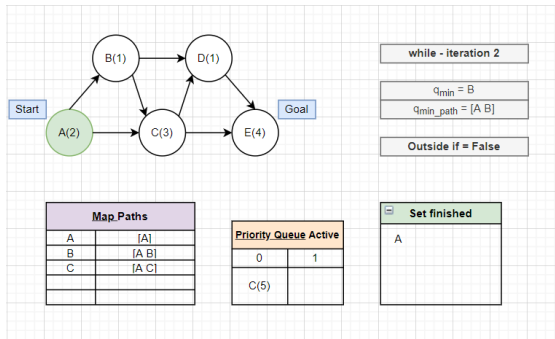


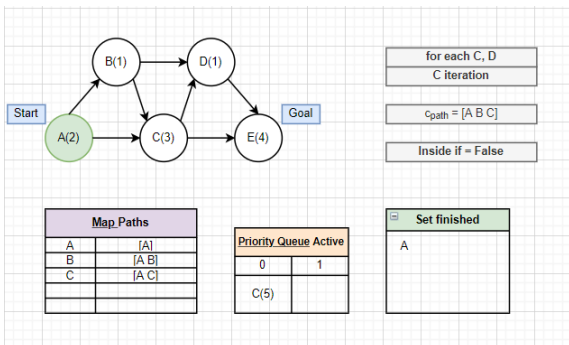


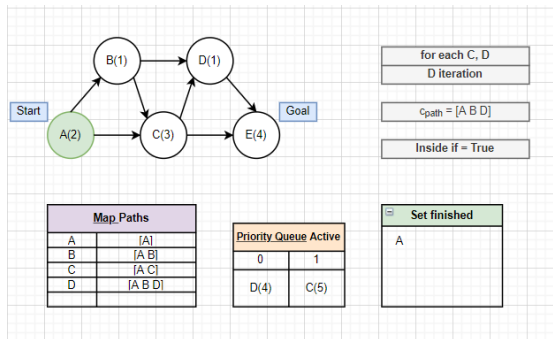

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each B, C
B iteration
c_{path} = [A B]
Inside if = True
Map Paths
A [A]
B [A B]
Priority Queue Active
0 1
B(3)
Set finished

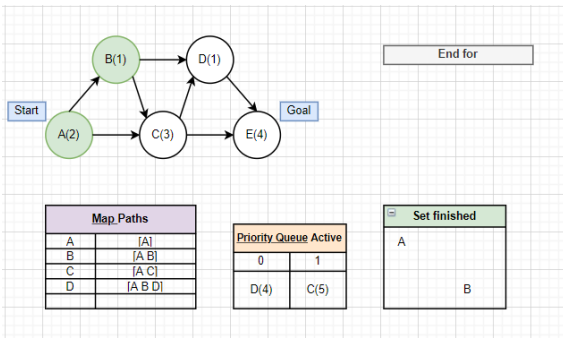

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each B, C
C iteration
c_{path} = [A C]
Inside if = True
Map Paths
A [A]
B [A B]
C [A C]
Priority Queue Active
0 1
B(3) C(5)
Set finished

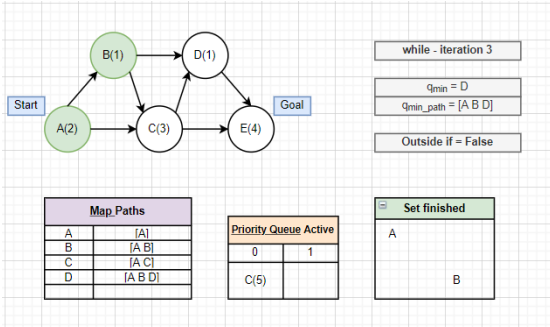

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
End for
Map Paths
A [A]
B [A B]
C [A C]
Priority Queue Active
0 1
B(3) C(5)
Set finished
A

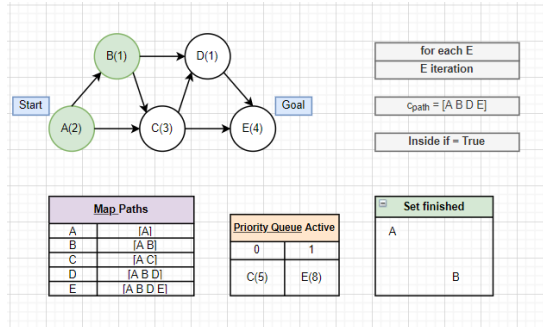

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
while - iteration 2
q_{min} = B
q_{min_path} = [A B]
Outside if = False
Map Paths
A [A]
B [A B]
C [A C]
Priority Queue Active
0 1
C(5)
Set finished
A

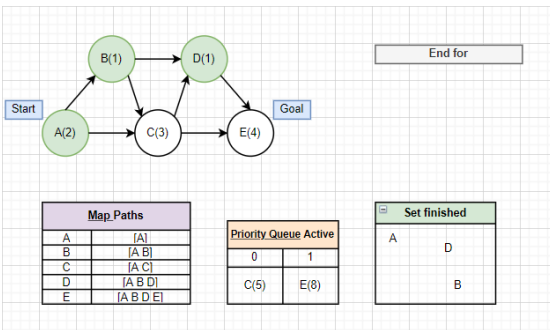

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each C, D
C iteration
c_{path} = [A B C]
Inside if = False
Map Paths
A [A]
B [A B]
C [A C]
Priority Queue Active
0 1
C(5)
Set finished
A

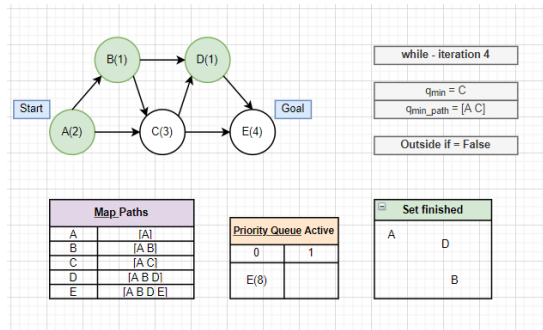

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each C, D
D iteration
c_{path} = [A B D]
Inside if = True
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
Priority Queue Active
0 1
D(4) C(5)
Set finished
A

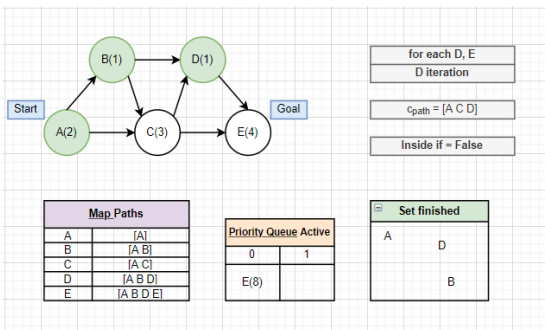

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
End for
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
Priority Queue Active
0 1
D(4) C(5)
Set finished
A
B

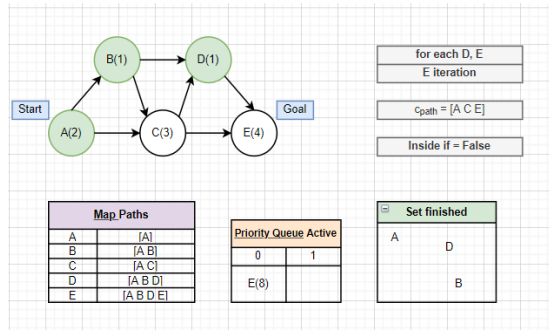

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
while - iteration 3
q_{min} = D
q_{min_path} = [A B D]
Outside if = False
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
Priority Queue Active
0 1
C(5)
Set finished
A
B

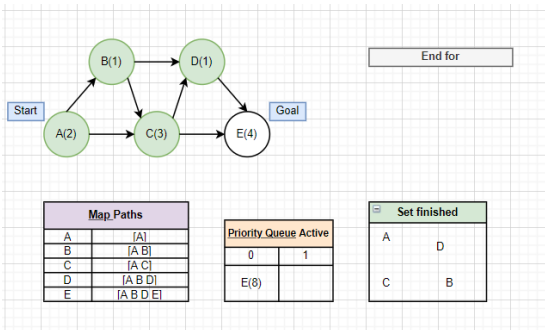

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each E
E iteration
c_{path} = [A B D E]
Inside if = True
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
C(5) E(8)
Set finished
A
B

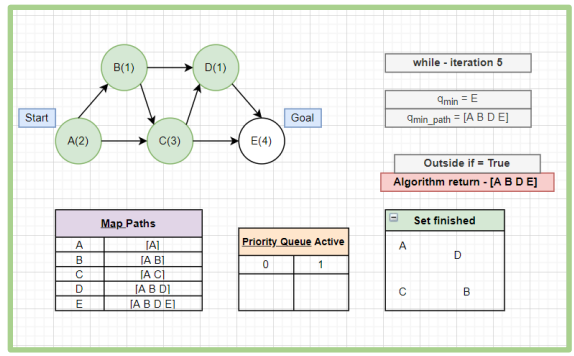

Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
End for
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
C(5) E(8)
Set finished
A
D
B


Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
while - iteration 4
q_{min} = C
q_{min_path} = [A C]
Outside if = False
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
E(8)
Set finished
A
D
B


Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each D, E
D iteration
c_{path} = [A C D]
Inside if = False
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
E(8)
Set finished
A
D
B


Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
for each D, E
E iteration
c_{path} = [A C E]
Inside if = False
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
E(8)
Set finished
A
D
B


Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
End for
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
E(8)
Set finished
A
D
C
B


Start
B(1)
D(1)
A(2)
C(3)
E(4)
Goal
while - iteration 5
q_{min} = E
q_{min_path} = [A B D E]
Outside if = True
Algorithm return - [A B D E]
Map Paths
A [A]
B [A B]
C [A C]
D [A B D]
E [A B D E]
Priority Queue Active
0 1
Set finished
A
D
C
B

שאלה 2

סעיף ג' –

כמו בבדיקות קופסא שחורה עבור גרף בדקנו את המפרט של המתודה find shortest path.

הבדיקות שביצענו

- מציאת מסלול בגרף ריק .
- מציאת מסלול מצומת לעצמו.
- מציאת מסלול כאשר צומת היעד לא בגרף.
- מציאת מסלול בין אב לבן.
- מציאת מסלול כאשר יש מספר מטרות ואחת מהן לא בגרף.
- מציאת מסלול כאשר יש כמה התחלות ואחת מהן לא בגרף.
- מציאת מסלול בגרף מלא(שאלה 2.1).
- מציאת מסלול בגרף מלא כאשר יש מספר התחלות ומטרות.

בדיקות אלה מכסות לפי שעתנו את המפרט של המחלקה כיוון שבודקות הרבה מקרים כולל מקרי קצה.

עבור בדיקות קופסא לבנה בדקנו ,בדומה לבדיקות שעשינו על גרף, שאכן כיסינו את כל המסלולים האפשריים במתודה.

שאלה 3

1. הבעיה היא שלא מוגדר על משתני המחלקה את סוג בקרת הגישה שלהם בשתי המחלקות, בגלל שב java בקרת הגישה הדיפולטית מוגדרת להיות package אז נוצר מצב שכל מי שמשתמש במחלקות אלו יכול לגשת למשתנים החברים מה שגורם ל- rep exposure.
בעיה זו נפתור על ידי הוספת המזהה private עבור המשתנים בכל מחלקה ומימוש מתודות observers ומתודות mutators במידת הצורך. אפשר למנוע rep exposure במתודה שתחזיר את שורש העץ על ידי החזרת עותק של השורש ולא את הצומת עצמו.

2. הסטודנט טועה, כאשר מממשים בנאי עם פרמטים ב java **ולא** מממשים בנאי חסר פרמטים לא יווצר בנאי כזה באופן דיפולטי. במקרה שלנו מחיקת הבנאי חסר הפרמטרים וקריאה אליו בקוד יגרור שגיאת קומפילציה.
3.

```
/**
  * @requires in case node != null then
  *              node != this.root && node not in this.root.left tree
  * @modifies this
  * @effects set the right son of this.root to the arg node
  *          notice that if there was a son the new node replace it
  *          if node == null then the right son is removed
  */
public void setRightSonOfRoot(Node node) {
    root.right = node;
}
```

4. יש לשנות את חתימת המחלקה כך שתתמוך בטיפוס גנרי -

$$class\ node < T > \{...$$

5. יהיה עלינו לשנות את החתימה של המחלקה כך שהטיפוס הגנרי המועבר אליה מממש את המתודה compareTo לכן החתימה תהיה -

$$class\ node < T\ extends\ Comparble <?\ super\ T >>$$

6.

```
// RepInvariant:
    //  this != this.right && this != this.left &&
    //  (this.right != null && this.left != null) -> this.right != this.left

// Abstraction Function:
    //  A node is a basic building block of a BinaryTree
    //  this.left is the left child || null
    //  this.right is the right child || null
    //  this.key is an integer which is the node key
```

```
// RepInvariant:
    //  this has no circles

// Abstraction Function:
    //  A binary tree is a data structure that has no circles
    //  each node has 0,1 or 2 nodes max
    //  this.root is the root || null if the tree is empty
```

7.

```
// RepInvariant:
    //  this.parent == null only if this is the root of the tree
    //  circles are allowed only between parent and child node

// Abstraction Function:
    //  A binary tree is a data structure that has no circles
    //  each node has 0,1 or 2 nodes max
    //  this.root is the root || null if the tree is empty
```