שאלה 1

**בדיקות קופסה שחורה - להוסיף עוד כמה**

**בדיקות קופסה לבנה - להוסיף**

**כתיבת החלק היבש.**

**בדיקת רווחים ושורות ריקות בפלט.**

בעיות אפשריות:

צומת עם אותו שם ומשקל שונה – הdriver לא נותן להוסיף את זה למערכת אבל הגרף כן (כי putifabsent בודק האם הצמתים שווים ולא בהכרח לפי השם).

הוספת קשת כאשר אחד הצמתים או שניהם לא קיים – אצלנו זה יוצר את הצמתים ומוסיף אותם לגרף אם הם לא נמצאים. אבל אין אינדקציה מבחוץ שזה קורה ומדפיסים רק על הוספת קשת בלי שהצמתים נוספו.

**שאלה 1**

1. ההפשטה עבור מחלקת גרף מכילה את הפעולות הבאות:

* הוספת צומת לגרף
* הוספת קשת לגרף
* החזרת רשימת צמתים
* החזרת רשימת ילדים של צומת מסוים

הפעולות נבחרו ע"י בחינת הנספח שצורף לתרגיל, לאחר הבנה מהן הפעולות הנתמכות על ידי הממשק של גרף.

פעולת יצירת גרף – לא נדרש בנאי במחלקה מכיוון שהגרף שנוצר מאותחל ללא צמתים וקשתות. לכן לא נדרש לאתחל אף שדה בתוך הבנאי. ולכן הבנאי הדיפולטי המסופק על ידי המחלקה מספיק.

הפעולה FindPath שנתמכת על ידי ממשק הבדיקות ממומשת במחלקה חיצונית לגרף ולכן לא נמצאת בתוך הממשק של גרף (אלא מקבלת אותו כפרמטר).

1. מחלקת graph היא Mutable. לכן, בהוספת קשת או צומת לגרף משתנה המופע של הגרף.  
   על מנת להגן על המבנה, בעת החזרת סט הצמתים וסט הילדים של צומת, השתמשנו במתודה Collections.unmodifiableSet שמחזירה גרסה לא ניתנת לשינוי של הסט. ולכן משתמש מבחוץ לא יוכל לשנות את הסטים.  
     
   מופע של מחלקת גרף מחזיק בתוכו מבנה מסוג HashMap הממפה בין צומת לבין סט של צמתים המייצגים את הבנים של אותו צומת.  
   את רשימת הילדים של הצמתים בחרנו לממש בתוך מבנה מתוך סט, מפני שכל צומת יכול להיות בן של צומת אחר פעם אחת בלבד, כלומר איבר לא יכול יכול להופיע במבנה פעמיים, והסט שומר על תכונה זו. בנוסף, אין חשיבות לסדר בין ילדים של צומת ולכן הסט מתאים לכאן.  
   את המבנה שמכיל את הצמתים בחרנו להחזיק במבנה hashMap כאשר הkey הוא הצומת והvalue הוא הסט של ילדי הצומת. כל צומת יכול להופיע בגרף פעם אחת, ולכן מבנה hashMap התאים כאן מפני שכל מפתח הוא ייחודי (לא יכול להופיע אותו מפתח פעמיים). בנוסף, בהינתן צומת נדרשנו לגשת בצורה יעילה ובזמן קבוע לסט של הילדים שלו. פעולת get של hashMap לוקחת o(1) ולכן המבנה hashMap ענה על הדרישה.

מימוש נוסף לדוגמא הוא ArrayList דו מימדי של כל הצמתים, כאשר לכל צומת ArrayList של בניו.  
יתרונות של מימוש זה: גם מימוש זה מספק גישה בזמן קבוע לכל איבר במבנה. מבנה ArrayList ניתן למיון ולכן לא נדרש להעתיקו לפני המיון (כמו שקורה במימוש שלנו מחוץ למחלקה).  
חסרונות של מימוש זה: ArrayList לא מונע כפילויות בתוך המבנה. לכן, בכל הוספה של קשת או צומת היינו צריכים לבדוק במבנה שהיא לא נמצאת בעצמנו.

1. השיקולים בבחירת בדיקות קופסא שחורה:  
   בדיקה של כל מתודה בנפרד והאפשרויות השונות עבורה על פי המפרט שלה.  
   בהוספת קשת לדוגמא, בדקנו גם מקרים שאמורים להצליח כמו: הוספה של קשת בצורה רגילה, הוספה קשת הפוכה וקשת עצמית. בדקנו גם מקרים שאמורים להחזיר את אחת השגיאות של הפונקציה (כפי שמפורט במפרט) למשל: הוספת קשת לצומת שלא נמצא בגרף או קשת שכבר קיימת בין אותם צמתים.  
   בנוסף בדקנו את תפקוד המחלקה וכל המתודות ביחד כאשר יש מספר גרפים במערכת, עם צמתים משותפים לכולם.

עבור מתודות שמשנות את המופע של Graph, בדקנו את השינויים על ידי הדפסת רשימת הצמתים והילדים.

**בדיקות קופסא לבנה:**

ד. **עוד לא בטוחה – לבדוק**   
 לא נעשה שימוש בשם המקוצר Path מפני שקיים בתוך הpackage גם ממשק של מחלקת Path ולכן זה עלול ליצור התנגשות ועלולה להיווצר שגיאה בזמן ריצה/קומפילציה.

שאלה 2

נשאר:  
כתיבת בדיקות קופסא לבנה ושחורה + תיעוד הבדיקות

**סעיף א**

**תיאור מילולי של אופן פעולת האלגוריתם:**

האלגוריתם הנתון מתחזק ערימת מינימום של צמתים, כאשר משקל הצומת הוא המסלול הקל ביותר עד אליו. בתחילת ריצת האלגוריתם בערימה יש את כל צמתי המקור, ומשקלן הוא משקל הצומת (המסלול הקצר ביותר הוא הצומת עצמה). בכל שלב מוציא את המינימום מהערימה, כלומר את הצומת שהמסלול אליו הוא בעל המשקל הנמוך ביותר. אם הצומת שהוציא נמצא ברשימת צמתי היעד, זה אומר שנמצא מסלול ולכן האלגוריתם מחזיר אותו (המסלול מוחזק בתוך map). לאחר מכן האלגוריתם עובר על כל הבנים של הצומת שהוציא. כל עוד הבן לא נמצא בערימת finished או active, הוא מחשב לכל בן מחשב את המסלול עד אליו כולל אותו. את המסלול מוסיף למבנה Map הממפה צומת למסלול מאחד מצמתי המקור לצומת הזה. לאחר סיום הטיפול בצומת, הוא נכנס למבנה finished. האלגוריתם ממשיך לרוץ כל עוד הערימה לא ריקה.

**הדגמת האלגוריתם ע"י מציאת מסלול מA לE:**

בתחילת האלגוריתם (הסימון מייצג שקיים מסלול אל הצומת N במשקל w):

שלב 1 – הוצאת איבר מactive ועדכון המשתנים:

שלב 2 – בלולאה עוברים על הילדים של הצומת A, כלומר על הצמתים B וC. מכניסים לactive את המסלולים אל צמתים אלה. מכניסים אל active את הצמתים B וC עם עדיפות השווה למשקל המסלול עד אליהם ומעדכנים את paths. בסיום האיטרציה:

שלב 3 – מוציאים שוב איבר מערימת active. בסיום השלב:

שלב 4 – בלולאה עוברים על הילדים של הצומת B, כלומר על הצמתים C ו D . מכניסים לactive את המסלולים אל צמתים אלה.מכיוון שC נמצא כבר בערימה active, מכניסים רק את הצומת D עם עדיפות השווה למשקל המסלול עד אליו ומעדכנים את paths עם המסלול עד אליו. בסיום האיטרציה:

שלב 5 – מוציאים שוב איבר מערימת active. בסיום השלב:

שלב 6– בלולאה עוברים על הילדים של הצומת D כלומר על הצמתים E . מכניסים לactive את המסלול אליו. מעדכים את active ואת paths. בסיום האיטרציה:

שלב 7 – מוציאים שוב איבר מערימת active. בסיום השלב:

שלב 8– בלולאה עוברים על הילדים של הצומת C כלומר על הצמתים E וD. מכיוון שD כבר נמצא ב finished וE כבר נמצא בactive לא מעדכנים אף מבנה.

שלב 9 - מוציאים שוב איבר מערימת active. בסיום השלב:

מכיוון שמתקיים התנאי

מחזירים את הpath המתאים:

התקבל מסלול במשקל 8 מצומת A לצומת E.

ב. בחרנו לממש את האלגוריתם בעזרת מחלקה עוטפת למחלקת הצומת, אשר מממשת את java.util.Comparable ולהכניס איברים מסוג זה אל ערימת המינימום. כמו כן, על מנת למנוע תלות בטיפוס הצמתים, המתודה מקבלת מסלולי התחלה ומסלולי סיום מנוונים (המכילים רק את הצומת עצמה ומשקלם הוא משקל הצומת). כך יכולנו להשתמש בממשק path שמאפשר את הפעולות getCost כדי לגשת למשקל הצומת וgetEnd על מנת לגשת לצומת עצמו.

1. שיקולים בבחירת בדיקות קופסא לבנה וקופסא שחורה ?

בדיקות קופסא שחורה: על פי המפרט של הפונקציה, בו קבענו מה הפונקציה צריכה להחזיר, בדקנו מספר אפשרויות לקלטים שיש ביניהם מסלול, קלטים שאין ביניהם מסלול, קלטים שיש ביניהם מספר מסלולים, מציאת מסלול מצומת לעצמו.

בדיקת המתודה עצמה – לבדוק שבאמת מחזירה מסלול נכון.

**שאלה 3**