# Object Oriented Programming and Design

## HW2

**Valeriya Khotuliov 328918966**

**Shay Bach 316147016**

**שאלה 1:**

א. קודם כל על ההפשטה להכיל את כל הפקודות המאפשרות ליצור גרף כללי המכיל בתוכו צמתים ומסלולים ביניהם, לכן יצרנו את הפעולות :

Graph(String name) - אשר יוצרת גרף חדש עם שם
addNode(T node) – מוסיפה צומת חדש שקיבלה לגרף
addEdge(T parent, T child) – מוסיפה קשת בין 2 צמתים בגרף שקיבלה, כאשר הקשר בין האב לבן הוא שהבן יתקיים ברשימת הילדים של האבא

לפתרון הבעיה אנו נצטרך לגשת לצמתים של הגרף על מנת לבדוק בהמשך מה הצומת הזול ביותר, כמו כן נצטרך גישה לבנים של כל צומת על מנת לבדוק את משקלם וכן להתקדם במסלול. לכן נוצרו הפעולות הבאות
listNodes() – הפעולה מחזירה את רשימת הצמתים הממוינת בגרף
listChildren(T node) - הפעולה את רשימת הבנים הממוינת של הצומת אותו קיבלה.

ב. בחרנו במימוש פשוט המאפשר ליצור גרף גנרי וכן מימוש המאפשר גישה לכל הצמתים בגרף וגם גישה לצמתים מהם ניתן להגיע מצומת נבחר.
על מנת להקל על חיפוש הצמתים בגרף, מימשנו את הגרף כHashMap על מנת לאפשר גישה מהירה לצומת, כמו כן, אין צומת המחובר בקשת כפולה לצומת אחר, לכן על מנת לשמור על חוסר קשתות כפולות, בחרנו להחזיק את רשימת הבנים של הצומת כSet
למרות שמדובר במימוש מופשט, בחרנו למיין את רשימות הצמתים אותן אנו מחזירים בפונקציות list, בחירה זו לא פוגעת בהפשטה, אך מייעלת את פתרון הבעיה המתוארת בתרגיל.
מימוש אחר שניתן היה להציע הוא שמירת צמתי הגרף גם במבנה נתונים של Set , וילדי הצמתים היו נשמרים גם במבנה Set כלומר היינו מקבלים רשימה המכילה רשימות. פתרון זה היה מצריך יותר בדיקות של מקרי קצה ובדיקות נוספות של מופעים שהם לא NULL.

ג. המימוש אמור לספק למשתמש יכולת ליצור גרף, כאשר לא ניתן להוסיף את אותו צומת פעמיים או יותר, וכן לא ניתן שיהיו כמה קשתות מצומת לצומת ( קשת בכיוון ההפוך מותרת). כמו כן, על המימוש לספק מסלול בין צומת מקור לצומת מטרה ואם לא קיים להדפיס הודעה בהתאם.
לכן הבדיקות שכתבנו הן גרף אשר כל הצמתים מצביעים לאותו צומת, וכן מצב בו צומת אחד מצביע על כולם בדיקת הדפסת מסלול כאשר לא קיים, בדיקות מציאת מסלול בין 2 צמתים וכן בדיקות אשר מראות כי לא נוסף צומת שכבר קיים בגרף ובדיקה דומה להוספה חוזרת של קשת.
בדיקות דומות מומשו גם בבדיקות קופסה לבנה, ונוספו בדיקות נוספות אשר בודקות את השגיאות המתקבלות בהוספת צומת שכבר קיים לגרף וכן הוספת קשת בין 2 צמתים אשר לא קיימים בגרף
בדיקות אלו מספקות כי הן בודקות את אופן היצירה, הקישור ותכונות הצמתים וכן בודקות את תכונות הגרף – חוסר שכפולי צמתים/קשתות. הבדיקות גם בודקות מקרי קצה כמו לולאה עצמית וחוסר מסלול בין 2 צמתים בגרף.

ד. בשני הקבצים הנ"ל אנו משתמשים גם במחלקה java.nio.file.Paths אם היינו מייבאים גם את java.nio.file.Path היה נוצר בלבול בין 2 המחלקות בעת כתיבת הקוד, לכן על מנת למנוע בלבול בין 2 השמות יש עדיפות להשתמש בשם המלא של המחלקה הנ"ל.

**שאלה 2:**

א. האלגוריתם הוא אלגוריתם מציאת המסלול הקצר ביותר בגרף בהינתן קבוצת מסלולי התחלה וקבוצת צמתי סיום. האלגוריתם מבצע חיפוש מסלול באופן חמדני בדומה לאלגוריתם של דייקסטרה. הוא מתחיל ביצירת מסלולים שמתוייגים לפי צמתי ההתחלה שלהם. לאחר מכן יוצר תור של כל מסלולי ההתחלה active ובאופן אירטיבי מוציא מהתור את הצומת עם המחיר הנמוך ביותר queueMin ובהתאם מוציא את המסלול שלו queueMinPath, בודק האם הצומת שהוצא הוא צומת הסיום, במידה וכן סיימנו ונחזיר את התור המתאים. במידה ולא נעבור על כל הילדים של הצומת האחרון שהוצא ובתנאי שאין לנו מסלול קיים שמתחיל מהצומת הילד וגם שלא בדקנו את הצומת הזה כבר ניצור מסלול חדש שמכיל את הצומת ונכניס את צומת הבן לתור עם מחיר לפי מחיר המסלול החדש, נמשיך כך לכל הבנים של הצומת queueMin, ונכניס את queueMin לרשימת הצמתים שבדקנו finished. במידה ומחסנית הצמתים ריקה לא מצאנו מסלול בגרף.

עבור הדוגמא במציאת המסלול הקצר ביותר מ-A ל-E :נכניס את A(2) למחסנית active, נשלוף אותו ונעבור על הבנים שלו – ניצור מסלולים A->B, A->C שנכניס ל-paths עם המפתחות C,B בהתאמה ונכניס את B למחסנית active עם מחיר 3 ואת C עם מחיר 5, נוסיף את A לרשימת finished.

נשלוף את B מהמחסנית, ניצור מסלולים ניצור מסלול A->B->D שנכניס ל-paths עם מפתח D, נכניס את D למחסנית עם מחיר 4 ונוסיף את B ל-finished.

נשלוף את D מהמחסנית, ניצור מסלול A->B->D->E שנכניס ל-paths עם מפתח E, נכניס את E למחסנית עם מחיר 8 ונוסיף את D ל-finished.

נשלוף את C מהמחסנית, כל הבנים שלו ב-active או ב-finished ולכן רק נוסיף אותו ל-finished.

נשלוף את E מהמחסנית ו-E הוא צומת הסיום ולכן נחזיר את המסלול A->B->D->E.

ג. בבדיקות כעת עלינו לבדוק מציאת המסלול הקצר בין 2 צמתים בגרף, וגם מציאת המסלול הקצר בין המסלולים האפשריים בין קבוצות של צמתי מקור וקבוצות של צמתי מטרה.

לכן יצרנו בדיקות אשר בודקות מצב בו יש קבוצה של צמתי מקור וצומת יעד יחידת צומת מקור אחד וקבוצה של צמתי מטרה וכן כאשר יש קבוצה של צמתי מקור וקבוצה של צמתי מטרה. בנוסף לבדיקות אלו נוצרה בדיקה אשר ישנם 2 מסלולים אפשריים בין 2 צמתים, אך אחד מהמסלולים יקר מהאחר והציפייה מהקוד היא למצוא את המסלול הזול ביותר.

בבדיקות קופסה לבנה בדקנו את מקרי הקצה בהן צומת המקור/המטרה הם NULL

בדיקות אלו מספקות כי הן בודקות את כל המצבים האפשריים מהם על הקוד למצוא את המסלול הזול ביותר וגם מקרים בהן יש לבדוק יותר מהתחלה/סוף יחידים.