מגישים:

אופיר קרנגל 302621305

רומן אברמזון 306359001

# שאלה 1

1. מחלקת Graph – מחלקה היוצרת גרף המורכב מצמתים בעלי טיפוס כלשהו. מכיוון שהגרף לא יודע (ולא צריך לדעת לצורך מימושו) מהו סוג הצומת, סוג הצומת מתקבל ע"י generic בשם Node. כל צומת יכול להופיע רק פעם אחת בגרף ולכן השדה הפנימי לתיאור רשימת הצמתים הוא מסוג HashMap המכיל מפתח מסוג string והערך הוא הצומת עצמו. בנוסף ,הגרף מחזיק תיאור של קשתות בין צמתים שונים וגם כאן אותה קשת לא יכולה להופיע פעמיים. לשם כך, יצרנו HashMap לתיאור הקשתות כאשר ה key הוא צומת האב והערך הוא רשימה ממויינת (PriorityQueue) של כל הבנים. זהו המימוש שנבחר מבין containers אפשרים בגלל הסיבוכיות הנמוכה שמתוארת בסעיף הבא.
2. סיבוכיות המתודות השונות:

מכיוון שרשימת הצמתים ושרשימת הקשתות מומשו באמצעות HashMap אז הסיבוכיות על הוספה (add) או האם האובייקט מוכל (contained) היא O(1) והיא הנמוכה מבין המימושים. הסיבוכיות של המתודה addEdge היא O(log n) מכיוון שמדובר בהוספת בן ל PriorityQueue. המתודה getChildren מחזירה את PriorityQueue הרלוונטי לפי צומת האב בסיבוכיות של O(1) (כי הקשתות מומשו ב HashMap). מתודה זו נקראת הרבה פעמים במהלך ביצוע אלגוריתם DFS ולכן נרצה לממש אותה בסיבוכיות הנמוכה ביותר. עלויות הסיבוכיות עבור כל אחד מ containers מפורטת בטבלה הבאה:

|  |
| --- |
| List | Add | Remove | Get | Contains | Next | Data Structure  ------------------------------------------------------------------------------ |
| ArrayList | O(1) | O(n) | O(1) | O(n) | O(1) | Array |
| LinkedList | O(1) | O(1) | O(n) | O(n) | O(1) | Linked List |
|  |
|  |
|  |
|  |
| Set | Add | Remove | Contains | Next | Size | Data Structure |
| ----------------------|----------|----------|----------|----------|------|------------------------- |
| HashSet | O(1) | O(1) | O(1) | O(h/n) | O(1) | Hash Table |
| LinkedHashSet | O(1) | O(1) | O(1) | O(1) | O(1) | Hash Table + Linked List |
| EnumSet | O(1) | O(1) | O(1) | O(1) | O(1) | Bit Vector |
| TreeSet | O(log n) | O(log n) | O(log n) | O(log n) | O(1) | Red-black tree |
|  |
|  |
|  |
|  |
|  |
| Queue | Offer | Peak | Poll | Remove | Size | Data Structure |
| ------------------------|----------|------|----------|--------|------|--------------- |
| PriorityQueue | O(log n) | O(1) | O(log n) | O(n) | O(1) | Priority Heap |
| LinkedList | O(1) | O(1) | O(1) | O(1) | O(1) | Array |
| ArrayDequeue | O(1) | O(1) | O(1) | O(n) | O(1) | Linked List |
|  |
| Map | Get | ContainsKey | Next | Data Structure |
| ----------------------|----------|-------------|----------|------------------------- |
| HashMap | O(1) | O(1) | O(h / n) | Hash Table |
| LinkedHashMap | O(1) | O(1) | O(1) | Hash Table + Linked List |
| EnumMap | O(1) | O(1) | O(1) | Array |
| TreeMap | O(log n) | O(log n) | O(log n) | Red-black tree |

1. לצורך בדיקות קופסה שחורה יצרנו שני טסטים: cycle, node\_error. הקובץ cycle יוצר גרף מעגלי ובמהלך הטסט נבדקות המתודות השונות של מחלקת Graph. לעומת זאת, node\_error בוחן את התנהגות גרף כאשר הקלט לא תקין (למשל הוספה של אותה הצומת פעמיים או אותה שפה פעמיים) ואז ההגנות של TestDriver פועלות.

בבדיקות קופסה לבנה הוספנו בדיקות שניגשות ישירות ל Graph ללא תיווך באמצע של TestDriver וכך בודקות נקודות קצה שתלויות במימוש הפנימי של המתודות השונות.

1. Path הינה מחלקה בחבילה homework2 המממשת מסלול של צמתים כך שיכול להיווצר conflict בין השתיים.

# שאלה 2

1. DFS – Depth-first search – הינו אלגוריתם לסריקת גרף. האלגוריתם מתחיל מצומת התחלה, בוחן את הילדים של הצומת כאשר הוא בוחר בכל פעם את הילד בעל המחיר הגבוה ביותר (ואם יש באותו מחיר אז בסדר אלפבתי נתוך יותר) וממשיך לסרוק ממנו. כאשר האלגוריתם מגיע לצומת ללא ילדים, הוא חוזר לצומת ממנו הגיע לילד וממשיך לילד הבא שאותו לא ביקר. האלגוריתם עוצר כאשר הוא מגיע לצומת סוף או כאשר הוא סיים לעבור על כל הצמתים בגרף.

נדגים את פעולתו על הגרף הבא כאשר נסרוק את המסלול מ A ל E:



לפני הרצת האלגוריתם מרוקנים את visited וצובעים את כל הצמתים ללבן.

שלב א - צובעים את A לאפור ומוסיפים אותו ל visited. מחלצים שהבנים של A שהם B, G, D ובוחרים להמשיך עם B.

שלב ב – צובעים את B באפור ומוסיפים אותו ל visited. מחלצים שהבנים הם A ו E וממשיכים ל E (כי ב A כבר היינו).

שלב ג – צובעים את E באפור ומוסיפים אותו ל visited. מכיוון ש E היא הצומת סוף אז מחזירים true. משם הערך true מחלחל האחורה לאב B ולאב A.

בסוף האלגוריתם צובעים את הצמתים חזרה ללבן.

בסוף התהליך המסלול המוחזר הוא visited = A, B, E.

הרצת האלגוריתם עבור A-> D:

לפני הרצת האלגוריתם מרוקנים את visited וצובעים את כל הצמתים ללבן.

שלב א - צובעים את A לאפור ומוסיפים אותו ל visited. מחלצים שהבנים של A שהם B, G, D ובוחרים להמשיך עם B.

שלב ב – צובעים את B באפור ומוסיפים אותו ל visited. מחלצים שהבנים הם A ו E וממשיכים ל E (כי ב A כבר היינו).

שלב ג – צובעים את E באפור ומוסיפים אותו ל visited. מכיוון ש E אין בנים אז צובעים לשחור ומחזירים false.

שלב ד – צובעים את B בשחור ומחזירים false.

שלב ה – עוברים ל G (הבן הבא של A). צובעים באפור ומוסיפים ל visited ומלחצים את הבנים R, D.

שלב ו – צובעים את R באפור ומוסיפים אותו ל visited. מכיוון ש R אין בנים אז צובעים לשחור ומחזירים false.

שלב ז – צובעים את D באפור ומוסיפים ל visited. מכיוון שזהו צומת הסוף, מחזירים true. משם הערך true מחלחל האחורה לאב G ולאב A.

בסוף האלגוריתם צובעים את הצמתים חזרה ללבן.

בסוף התהליך המסלול המוחזר הוא visited = A, B, E, G, R, D.

הרצת האלגוריתם עבור B:

לפני הרצת האלגוריתם מרוקנים את visited וצובעים את כל הצמתים ללבן.

שלב א - צובעים את B לאפור ומוסיפים אותו ל visited. מחלצים שהבנים של B שהם E, A ובוחרים להמשיך עם E.

שלב ג – צובעים את E באפור ומוסיפים אותו ל visited. מכיוון ש E אין בנים אז צובעים לשחור ומחזירים false.

שלב ד – עוברים ל A (הבן הבא של B). צובעים לאפור ומוסיפים ל visited. מחלצים שהבנים של A הם B, G, D. ממשיכים ל G (כי ב B כבר היינו).

שלב ה –צובעים את G באפור ומוסיפים ל visited ומלחצים את הבנים R, D.

שלב ו – צובעים את R באפור ומוסיפים אותו ל visited. מכיוון ש R אין בנים אז צובעים לשחור ומחזירים false.

שלב ז – צובעים את D באפור ומוסיפים ל visited. מכיוון ש D אין בנים אז צובעים לשחור ומחזירים false.

שלב ח – צובעים את G בשחור ומחזירים false.

שלב ט – עוברים לילד הבא של A שהוא D אך מכיוון שכבר היינו בו אז צובעים את A בשחור ומחזירים false.

בסוף האלגוריתם צובעים את הצמתים חזרה ללבן.

בסוף התהליך המסלול המוחזר הוא visited = B, E, A, G, R, D.

1. כדי לבדוק האם יש קשתות אחוריות בגרף, נבדוק בתחילת הפעלת המתודה DFS את צבעי הילדים. כל ילד בצבע אפור הוא סימן לאב קדמון (כי צומת בצבע אפור הוא צומת שממנו הגיעו לילדים). לכל צומת נוסיף שדה שמציין כמה קשתות אחוריות יש ונתחשב בזה ב NodeCountingPath.
2. לצורך בדיקות קופסה שחורה, cycle\_back מממש גרף עם קשתות אחוריות. הטסט מריץ את DFS וגם FindPath על הצמתים השונים בהתאם למפרט שהוגדר.

לצורך בדיקות קופסה לבנה, WhiteTestsDFS בודק את התנהגות DFS ו PathFinder במקרה קצה שונים (מסלול עם ובלי קשת אחורית, השוואה בין שני מסלולים שונים ומציאת המסלול הקצר ביותר, מקרים בהם ניגשים לצומת לא קיימת או מבקשים מסלול בין שתי צמתים לא מקושרים וכו..)