**תרגיל בית 2 בתכנות ותכן מונחה עצמים**

עמית ויס 305243719

מיכאל נוביצקי 311773915

**שאלה 1**

**סעיף א**

הפעולות שבחרנו לממש עבור ההפשטה מתחלקות ל-3 קבוצות:

1. פעולות פנימיות של המחלקה:

אלה הן מתודות עזר שמשמשות אותנו לביצוע פעולות על ההפשטה של הגרף. המתודות הן למשל מתודת בדיקה האם תווית מסוימת כבר משמשת צומת מסוים (isNodeWithNameExists) ומתודה שמוודאת את קיום ה-Representation Invariant (checkRep). כל המתודות האלה הן פרטיות ומיועדות לשימוש שלנו.

1. פעולות הדרושות לצורך הממשק של BipartiteGraphTestDriver:

מכיוון שנדרשנו לממש את כל פעולות המחלקה, השתמשנו בפעולות על ה-Driver כדי לבנות גם פעולות עבור מחלקת הגרף הכללית.

1. פעולות הנגזרות מתיאור הבעיה:

מההיכרות שלנו עם גרפים דו צדדיים, הבנו כי יידרשו פעולות מסוימות (שאינן מופיעות ב-Driver) לצורך שימוש יעיל ונוח בגרף דו צדדי. על כן הוספנו מתודות כמו:

getData, getLabel, getColor למחלקת ה-Node שלנו וכך נוכל, על ידי שימוש בגרף, לגשת לצומת ולקבל את הפרטים הרלוונטיים.

לכל פעולה שרצינו לבצע במהלך התרגיל מצאנו שהפעולות שמימשנו מספיקות ואף מאפשרות נוחות בעבודה עם הגרף ועל כן מימשנו אותן. נעיר כי ניתן לממש עוד פעולות רבות נוספות על גרף דו צדדי, אך בחרנו לממש את מה שבעינינו נדרש על מנת לאפשר שימוש פשוט אך נוח.

**סעיף ב**

בחרנו לממש את הגרף שלנו תוך העברת חלק גדול מהאחריות לביצוע המתודות אל הצמתים עצמם.

לפיכך, יצרנו טיפוס נתונים חדש, Node, אשר מכיל את השדות תווית, צבע, מידע (האובייקט שמאוחסן בצומת) ובנוסף מכיל שתי מפות ערבול – הראשונה שמכילה את כל בניו והשניה שמכילה את כל הוריו (כאשר המפתחות הם התוויות של הקשתות לצמתים אלה). בנוסף, בחרנו להחזיק לכל צומת שתי רשימות: של כל תוויות בניו ושל כל תוויות הוריו.

עבור הגרף עצמו – בחרנו לממשו כאובייקט שמחזיק שתי מפות ערבול כאשר המפתחות הם תוויות הצמתים והמידע הוא הצמתים עצמם. כל מפה מחזיקה צמתים מצבע מסוים – שחור או לבן. בנוסף אנחנו משמרים קבוצה (Set) של כל תוויות הקשתות השונות, לצורך ביצוע פעולות בהמשך.

פעולות מסוימות יבוצעו על ידי מחלקת הגרף עצמה, כמו למשל החזרת רשימת כל הצמתים הלבנים/ כל הצמתים השחורים, בעוד שפעולות אחרות יבוצעו על ידי קריאה למתודות של מחלקת הצומת כמו למשל החזרת רשימת כל ההורים/ כל הילדים.

השיקולים שהובילו אותנו לבחירה במימוש זה הם:

1. אחריות יותר "טבעית" למתודות מסוימות – נראה לנו הגיוני יותר שכל צומת יתחזק את הוריו ואת ילדיו ולא שהגרף ינהל זאת ללא ידיעה של הצומת. בצורה זו אנחנו מאפשרים, על ידי שימוש במבנה נתונים יעיל כמו מפת ערבול, גישה מהירה לבדיקות מסוימות.
2. התחזוקה של רשימות העזר שאנחנו מחזיקים עבור כל צומת (רשימת ההורים ורשימת הבנים) בעת הוספת בן/הורה לצומת, מאפשרת לנו להחזיר את רשימת כל בנים/רשימת כל ההורים של צומת מסוים בזמן קבוע.
3. פעולות שסביר שנידרש להן בגרף דו-צדדי יבוצעו ביעילות יחסית, כמו יצירת צומת (שמתבצעת על ידי שימוש ב-builder ומיד לאחריו הוספה למפת ערבול – כלומר באופן משוערך בזמן קבוע), הוספת קשת (שדורשת בדיקת תוויות לצמתים ביניהם מחברים), חיפוש צומת לפי תווית הקשת (שמתבצעת על ידי גישה למפת הערבול של הצמתים לפי צבעים ולאחר מכן למפת הערבול של ההורים/ בנים ולכן גם יחסית מהירה) וכו'.

**סעיף ג**

אפשרות נוספת עליה חשבנו בשלב ראשוני היא מימוש על ידי מטריצת סמיכויות. בצורה זו מקבלים מטריצה NxN כך שכל צומת i הוא בעל קשת לצומת j אמ"מ יש "1" בתא (i,j) במטריצה.

מימוש זה לוקה במספר בעיות לעומת המימוש שלנו:

1. נדרש מאיתנו לדעת סדר גודל של מספר הצמתים מראש – מה שלא נתון. גם אם נרצה ליצור מטריצה דינמית שגדלה וקטנה, עדיין ייתכן בזבוז משמעותי של זיכרון על שמירת מטריצה גדולה יותר מהנדרש או סיבוכיות זמן גבוהה במידה ונחזיק מטריצה קטנה מדי שנצטרך לשנות את גודלה לעיתים קרובות.
2. ההפרדה לצמתים לבנים ושחורים היא פחות טבעית. על מנת לדעת אם צומת הוא שחור או לבן נוכל לקבל את התשובה מהר (על ידי גישה בזמן קבוע לתא המתאים במטריצה ובדיקת ערך הצבע שלו), אבל כדי להחזיר את כל הצמתים השחורים או הלבנים נידרש לכל הפחות לתחזק מבנה נתונים נוסף.
3. החזרת כל הבנים או כל ההורים של צומת מסוים לוקחת פרק זמן שאינו קבוע – אלא תלוי במספר הצמתים (שכן אנחנו נדרשים לעבור על שורה/עמודה שלמה), בניגוד לנדרש בתרגיל.

נעיר כי יחד עם זאת קיימים גם יתרונות למטריצת סמיכויות – למשל הוספת קשת למטריצה נעשית בצורה פשוטה על ידי "הדלקת" תא מסוים במטריצה.

**סעיף ה**

דאגנו לכך שהבדיקות שביצענו במחלקה BipartiteGraphTest מכסות את כל מתודות הממשק של הגרף שנדרשנו לקיים. עבור כל מתודה עברנו על כל המקרים האפשריים שעליהם חשבנו הכוללים בדיקת תקינות ומקרי קצה (לרבות בדיקת זריקת חריגות).

פירוט:

testExample- בדיקה שניתנה על ידי צוות הקורס. בבדיקה הוספנו שני צמתים לגרף, חיברנו ביניהם קשת, ווידאנו את היחס בין הצמתים (צומת אב וצומת בן)

addNodesRegularTest – הוספנו 3 צמתים לבנים ו-3 צמתים שחורים ווידאנו שהם נמצאים בגרף בעזרת המתודות listBlackNodes, listWhiteNodes.

addNodesExceptionTest- וידאנו שלא ניתן להוסיף צמתים עם אותה התווית ושבמידה וננסה לעשות זאת נקבל חריגה מתאימה.

listNodesTest – הוספנו 6 צמתים שחורים ו-6 צמתים לבנים לגרף ווידאנו את תקינות המתודות listBlackNodes, listWhiteNodes.

addEdgeRegularTest- הוספנו צמתים וקשתות ווידאנו את תקינות המתודה להוספת קשתות לגרף.

addEdgeBetweenSameColorTest- בדקנו שהוספת קשת בין שני צמתים בעלי צבע זהה זורקת חריגה מתאימה.

addEdgeWithLabelThatAlreadyExistTest- בדקנו שלא ניתן להוסיף קשת עם תווית הזהה לתווית של קשת אחרת הנמצאת בגרף ושנזרקת חריגה מתאימה כדי להודיע על כך.

getNodesByEdgeLabelTest- הוספנו צמתים וקשתות לגרף ובדקנו את תקינות המתודות getChildByEdgeLabel, getParentByEdgeLabel

testGetEdgesLabels- בדקנו שהמתודה getEdgesLabels מחזירה סט נכון של התוויות של קשתות הגרף.

הבדיקות שבחרנו מספקות אותנו כיוון שהן מכסות את כל המתודות שמימשנו עבור הגרף. כל מתודה נבדקת בבדיקה ייעודית משלה ובנוסף משתמשים בה בבדיקות של מתודות אחרות. בנוסף, קיימת בדיקה כוללת של בניית גרף קטן ווידוא נכונותו. על ידי הבדיקות אנחנו יכולים להבטיח שכל מתודה מתנהגת כמצופה וכך שכל הגרף שלנו יתנהג כמצופה.