**תיעוד וסיבוכיות המתודות – תרגיל מעשי 2**

**מגישים:**

|  |  |  |
| --- | --- | --- |
|  | **מגיש א'** | **מגיש ב'** |
| **שם פרטי** | אדם | אסף |
| **שם משפחה** | טובי | מיכאלוביץ' |
| **תעודת זהות** | 215334822 | 200637270 |
| **שם משתמש אוניברסיטאי** | adamtuby | michaelovits |

**סיבוכיות ותיעוד:**

**public** **class** Graph

**private** **final** hashMap nodesHash

שדה זה מחזיק את טבלת ההאש המבוקשת בהנחיות התרגיל, שבה נשמור את הצמתים בגרף, כאשר המפתחות של האיברים בטבלת ההאש יהיו ה'מזההים' שלהם. כלומר הטבלה ממפה ממזהה קודקוד לייצוגו בגרף ובערימת המקסימום.

**private** **final** maxHeap nodesHeap

שדה זה מחזיק את ערימת המקסימום המבוקשת בהנחיות התרגיל, בה כל זוג מפתח-ערך הוא כך שהמפתח הוא סכום הסביבה של צומת כלשהו u, והערך הוא הצומת u.

**public** Graph(Node [] nodes)

הבנאי של המחלקה של התרגיל – Graph.  
הבנאי מקבל רשימה של צמתים (הצמתים הם בפורמט של צומת של גרף. פורמט זה הוא מחלקה סטטית Node של המחלקה Graph). הוא מגדיר את השדה nodesHash בכך שהוא שולח את מספר הצמתים שנשלחו, לבנאי של hashMap, שהיא מחלקה שמימשנו בתוך Graph, שממלאת את דרישת התרגיל שתהיה טבלת האש לגרף.  
הבנאי של hashMap לוקח את מספר הצמתים, ולוקח בהכרח את המספר כפקטור העומס הרצוי, ובונה לו את המערך לפי הצרכים שלו, כפי שראינו בכיתה. פירוט מעמיק על המחלקה בהמשך.  
נוסף על כך, הוא מגדיר את השדה nodesHeap בכך שהוא שולח גם את מספר הצמתים שנשלחו, לבנאי של maxHeap (שממש את ערימת המקסימום שבדרישת התרגיל) בשביל ש maxHeap יוכל לאתחל את ה priority-Queue שלו (כמערך). פירוט מעמיק על המחלקה maxHeap יבוא בהמשך.

לאחר אתחול השדות הללו, הוא מוסיף באמצעות לולאה את כל הצמתים nodes, אחד אחד, לטבלת ההאש nodesHash, ולערימת המקסימום nodesHeap, לפי הדרישות של המחלקות.

**סיבוכיות המתודה:** O(N) AMORTIZEDסיבוכיות המתודה היא , כאשר n הוא מספר הצמתים שהכנסנו לגרף.  
ובכן, הכנסה של n מפתחות לערימת מקסימום (או מינימום), כמו שראינו בכיתה, בעלת סיבוכיות של O(n) w.c., וסיבוכיות ההכנסה של n מפתחות לטבלת האש לוקח O(n) AMORTIZED בטבלת האש של Linked Lists (לפי מה שראינו בכיתה, כי דאגנו שפקטור העומס יהיה ), ולכן סה"כ סיבוכיות המתודה היא: O(n) AMORTIZED + O(n) = O(n) AMORTIZED.

**public** Node maxNeighborhoodWeight()

מתודה זו מחזירה את הצומת בגרף, שמשקל הסביבה שלו הוא הגדול ביותר בגרף. משקל הסביבה הוא סכום המשקלים של השכנים של הצומת + ועוד המשקל של הצומת עצמו. מתודה זו עושה זאת באמצעות החזרת הצומת שנמצא בראש ערימת המקסימום שאנחנו מתחזקים nodesHeap.

**סיבוכיות המתודה:** O(1)סיבוכיות המתודה היא O(1), שכן כל מה שנדרש זה לשלוף את האיבר הראשון במערך שאנחנו מתחזקים כ priority Queue של הערימה.

**public** **int** getNeighborhoodWeight(**int** node\_id)

מתודה זו מחזירה את משקל הסביבה של הצומת בגרף שבעל המזהה שמקבלת המתודה – node\_id.  
במידה ולא קיים צומת בגרף עם מזהה שכזה, נחזיר -1.  
נחפש את המזהה node\_id בטבלת ההאש, ובמידה ונמצא צומת שאכן עונה למזהה, נחזיר את משקל הסביבה שלו, אחרת נחזיר -1.  
נעשה זאת פשוט על ידי תחזוק שדה של משקל הסביבה עבור כל צומת (הרי במילא אנחנו צריכים לתחזק שדה זה עבור שלמות ערימת המקסימום שאנו צריכים לממש).  
ובכן, הגישה למשקל הסביבה תיקח לנו סיבוכיות של O(1), ולכן זניחה.

**סיבוכיות המתודה:** O(1) AMORTIZEDסיבוכיות המתודה היא O(1) AMORTIZED, שכן החיפוש של המזהה node\_id בטבלת האש במימוש Linked List, הינו מסיבוכיות O(1) AMORTIZED (נקבע לפי זמן החיפוש בתוחלת שעלינו לעבור).

**public** **int** getNumNodes()

מתודה זו מחזירה את כמות הצמתים שקיימת בבגרף ברגע הקריאה למתודה.   
נעשה זאת באמצעות השדה size שאנחנו מתחזקים בערימת המקסימום nodesHeap שלנו, שבעצם סופר כמה צמתים קיימים בערימה. תיעוד של שדה זה ותחזוקו בהמשך התיעוד של מחלקת הערימת מקסימום.  
ובכן, פשוט נקרא ל getter של השדה size של nodesHeap, ונחזיר את התוצאה.

**סיבוכיות המתודה:** O(1)סיבוכויות המתודה היא כמובן O(1) שכן פשוט ניגשנו למצביעים.

**public** **boolean** isEmpty()

מתודה זו תחזיר true אם בגרף אין צמתים כלל, אחרת false. היא עושה זאת באמצעות פנייה לגודל ערימת המקסימום nodesHeap, ובודקת האם היא 0. אם היא אכן 0, זה אומר שלא קיימים צמתים בכלל בעץ ולכן תחזיר true, אחרת תחזיר false.  
נשתמש במתודה זו בכדי לחסוך שכפול קוד במתודות אחרות רבות, שמצריכות התייחסות למקרה הפרטי שהגרף ריק.

**סיבוכיות המתודה:** O(1)  
ובכן, התבקשנו לבצע מספר קבוע של פעולות השוואה וגישה למצביעים ולכן הסיבוכיות היא קבועה.

**public** **int** getNumEdges()

מתודה זו מחזירה את כמות הקשתות הקיימות בגרף שלנו ברגע הקריאה למתודה.  
דבר זה נעשה בעצם באמצעות לולאה על כל הצמתים שנמצאים בגרף. ניתן לעשות את לולאה זאת על המערך של טבלת ההאש שמימשנו, או על המערך של ערימת המקסימום שמימשנו (שכן שני המימושים מכילים אפשרות לעבור אופן איטראטיבי על כל הצמתים בגרף).  
בחרנו בערימת המקסימום.  
ובכן, ניגש למערך של ערימת המסימום המייצג אותה, ונעבור צומת צומת, ונוסיף את כמות השכנים שיש לכל צומת לאיזשהו משתנה result אינטג'רי.  
ניתן לבדוק את כמות השכנים שיש לכל צומת על ידי קריאה לשדה length שיש לרשימה מקושרת דו-כיוונית שמימשנו.  
כעת, לאחר שהוספנו את כל כמויות השכנים ל result, בעצם נקבל את כמות הקשתות בגרף כפול 2, שכן כל קשת מיוצגת **בדיוק** פעמיים ברשימות השכנויות של הצמתים, ועל פי משפט ממבוא לקומבינטוריקה שאומר ש .  
ולכן, נחזיר את result/2.

**סיבוכיות המתודה:** O(n) w.c.  
סיבוכיות המתודה היא כמובן O(n) w.c., כאשר n הוא כמות הצמתים בגרף, שכן ביצענו n איטראציות כאשר כל אחת לקחה O(1).

**public** **boolean** deleteNode(**int** node\_id)

מתודה זו נועדה למחוק את הצומת בגרף שהמזהה שלו הוא node\_id.  
במידה ואין צומת כזה בגרף, תחזיר false, אחרת, תמחק את כל הקשתות המחוברות אל הצומת, ותשנה את כל המשקלים הסביבתיים של השכנים היו לצומת, ותחזיר true.

ובכן, נמצא את הצומת בטבלת ההאש לפי המזהה שקיבלנו.  
ראשית נמחק את הצומת לגמרי מטבלת ההאש (O(1) AMORTIZED) ונמחק את הצומת מערימת המקסימום (O(log n)).  
כעת, ניגש לרשימת השכנויות של הצומת.   
נעבור בלולאה על כל האיברים שנמצאים ברשימת השכנויות ונעשה כך עבור כל איבר:  
- עבור כל איבר, שמייצג קשת, ניגש לייצוג המקביל של הקשת באמצעות השדה parallel, שיהיה בעצם איבר בתוך רשימת השכנויות של השכן שנמצא בקשת זו ספציפית – O(1)  
- האיבר שקיבלנו בתוך רשימת השכנויות של השכן שנמצא בקשת זו ספציפית, יהיה בעצם הצומת שאנו רוצים למחוק, בתוך רשימת השכנויות של השכן שלו בקשת  
- נמחק את איבר זה מן רשימת השכנויות של השכן הנ"ל, באמצעות מחיקה נורמטיבית מרשימה דו-כיוונית מקושרת – O(1)  
- נעדכן את המשקל הסביבתי של השכן הנ"ל, בכך שנחסיר מהמשקל הסביבתי של השכן את המשקל של הצומת שאנו רוצים למחוק - O(1)  
- נבצע פעולת Heapify לשכן הנ"ל בערימת המקסימום, שכן המשקל הסביבתי שלו השתנה - O(log (n-1))=O(log n)  
  
**סיבוכיות המתודה:** , כאשר היא דרגתו של הצומת  
סך הכל, כאשר היא דרגתו של הצומת שאותו אנחנו רוצים למחוק מן הגרף, סיבוכיות המתודה היא:

**public** **boolean** addEdge(**int** node1\_id, **int** node2\_id)

מתודה זו נועדה להוסיף קשתות חדשות לגרף.  
המתודה מחזירה true אם הוספנו קשת בין הצמתים שמחזיקים את המזההים שקיבלנו, אחרת false (תחזיר false במידה ואחד מהמזההים שקיבלנו לא נמצא בגרף).  
אומנם אין תת מחלקה רשמית של Graph שמייצגת 'קשת' כמו שאר האובייקטים שייצרנו, אך ייצוג הקשת הינו דווקא בתוך מימוש המחלקה Node, שהיא מחלקה המממשת צומת בגרף.  
ובכן, נסביר בקצרה על הייצוג של הקשת שבחרנו בו, ונשאיר את פרטי הפרטים לתיעוד המחלקה Node.  
אכן, כל צומת מכיל בתוכו שדה שהינו תת-מחלקה של Graph שמימשנו ששמה DoublyLinkedList<T>, שהיא בעצם מימוש של רשימה מקושרת דו-כיוונית שהאיברים שלה הם ממחלקה T (נתעד אותה בהמשך), אך במקרה זה T יהיה Node, שכשמה כן הוא, רשימה מקושרת דו-כיוונית של איברים מסוג Node.  
אחריות רשימה זו, הוא להכיל את השכנים של הצומת (לפי סדר הכנסתם).  
האיברים ברשימה המקושרת הדו-כיוונית ממומשים על ידי תת המחלקה של DoublyLinkedList<T>, הנקראת: DoublyLinkedCell (תת-מחלקה לא סטטית, שכן נקבעת על פי T, ושכן אנו צריכים להשתמש בDoublyLinkedCell.this בכדי למחוק איברים בעתיד מרשימות שכנויות).  
אז קשת בין הצומת , תיוצג על ידי הנוכחות של הצומת ברשימת השכנויות של , ובדומה תיוצג על ידי הנוכחות של הצומת ברשימת השכנויות של .  
עם זאת, כיוון שנדרשנו שלכל קשת יהיה מצביע דו-כיווני לייצוג ההפוך שלה (כפי שראינו לקשת יש שני ייצוגים), לכל איבר ברשימה המקושרת של , שמייצג קשת יהיה שדה parallel, המכיל מצביע לאיבר אחר מסוג DoublyLinkedCell, שיהיה בעצם האיבר ברשימה המקושרת של , כלומר ייצוג של הקשת .   
  
ובכן, נעשה זאת באמצעות חיפוש המזההים בטבלת ההאש nodesHash של הצמתים, ובמידה ומצאנו שני צמתים שתואמים את המזההים (אחרת מחזירים false), נוסיף את שני הצמתים לרשימת השכנויות של אחד השנייה, ונדאג להוסיף את שדה ה parallel לאיברים החדשים ברשימת השכנויות.  
  
לאחר ההוספה של הקשת וההוספה של הצמתים לרשימת השכנויות של אחד השני, המשקל הסביבתי של שני הצמתים כנראה השתנה, ועל כן יש חשש ל2 פעולות Heapify שיתבצעו בערימה. כל פעולה כזאת תעלה לנו כפי שראינו בכיתה.

**סיבוכיות המתודה:**  O(log n)סיבוכיות המתודה היא O(log n), שכן התבקשנו למצוא 2 איברים עם מפתחות בטבלת האש, ובנוסף על כך לבצע פעולות בעלות סיבוכיות זמן קבועה כגון שינויי מצביעים וכו'.

**public** **static** **class** Node

**private** **final** **int** id - שדה זה מחזיק את המזהה של הצומת בגרף

**private** **final** **int** weight - שדה זה מחזיק את המשקל של הצומת בגרף

**private** **int** vicinityWeight - שדה זה מחזיק את המשקל הסביבתי של הצומת בגרף

**private** **final** DoublyLinkedList<Node> Neighbours - שדה זה מחזיק מצביע אל רשימת השכנויות של הצומת בגרף

**private** maxHeap.heapNode heapForm - שדה זה מחזיק מצביע אל הייצוג של הצומת בערימת המקסימום של הגרף. שדה זה יאותחל ברגע שנכניס את הצומת לערימת המקסימום.

**private** hashMap.hashCell hashForm - שדה זה מחזיק מצביע אל הייצוג של הצומת בטבלת ההאש של הגרף. שדה זה יאותחל ברגע שנכניס את הצומת לטבלת ההאש.

**public** Node(**int** id, **int** weight)

הבנאי של הצומת.  
הוא מאתחל את המזהה של הצומת להיות id, ומאתחל את המשקל של הצומת להיות weight, ומאתחל את המשקל הסביבתי של הצומת להיות 0, ומאתחל את רשימת השכנויות של הצומת להיות new DoublyLinkedList<Node>.

**סיבוכיות המתודה:** O(1)

**public** **int** getId() - ה getter של השדה id. מחזיר את המזהה של הצומת.

**public** **int** getWeight() - ה getter של השדה weight. מחזיר את המשקל של הצומת.

**public** **int** getVicinityWeight()- ה getter של השדה vicinityWeight. מחזיר את המשקל הסביבתי של הצומת.

**public** **void** UpdateVicinityWeight(**int** additionalWeight)

מתודה זו מעדכנת את המשקל הסביבתי של הצומת במידה ונוסף לו שכן.  
היא לוקחת את המשקל הסביבתי הנוכחי, ומוסיפה לו את המשקל additionalWeight (שאמור להיות המשקל של השכן שנוסף לצומת).  
בגלל שהמשקל הסביבתי השתנה, כעת יש אפשרות שצריך לבצע פעולת Heapify בערימת המקסימום, ואכן נעשה זאת.

**סיבוכיות המתודה:** O(log n)ובכן, התבקשנו לבצע פעולת Heapify, ועל כן סיבוכיות הפעולה תהיה O(log n), שכן יש n צמתים בערימת המקסימום.

**public** **void** addNeighbour(Node node)

מתודה זו מוסיפה את הצומת node להיות שכן של הצומת שלנו בגרף.  
מתודה זו נקראת מתוך מתודת הוספת קשת בגרף, כאשר הקשת מחברת בין node לבין הצומת של מופע המחלקה.  
אנו נוסיף את node לרשימת השכנויות של הצומת, ונוסיף למשקל הסביבתי של הצומת את המשקל של node (באמצעות UpdateVicinityWeight(node.getWeight())).

**סיבוכיות המתודה:** O(log n)  
ובכן, ההוספה של השכן לרשימת השכנויות הוא פשוט הוספה של איבר לסוף הרשימה המקושרת הדו-כיוונית, ועל כן פעולה זו בעלת סיבוכיות קבועה.  
עם זאת, עשינו גם שינוי במשקל הסביבתי, באמצעות המתודה UpdateVicinityWeight, וסיבוכיות מתודה זו היא O(log n), ולכן סיבוכיות המתודה addNeighbour תהיה O(1)+O(log n) = O(log n).

**public** **void** setHeapForm(maxHeap.heapNode heapForm)

מתודה נקראת פעם אחת בלבד במהלך כל המופע של Graph.  
תפקיד מתודה זו הוא לאתחל את המצביע heapForm של הצומת, ככה שנוכל לגשת באמצעות המופע של Node בלבד, למופעים של הצומת בתוך ערימת המקסימום ובתוך טבלת ההאש לשם שלמות המימוש ונגישות המימוש.  
מתודה זו נקראת רק מתוך הבנאי של המחלקה Graph, בה אנו מקבלים N צמתים ומוסיפים אותם לטבלת ההאש nodesHash ולערימת המקסימום nodesHeap.  
ובכן, בהינתן צומת כלשהו, אנחנו נוסיף אותו לטבלת ההאש ולערימת המקסימום באמצעות מתודות ההוספה שנמצאות במימוש hashMap ו maxHeap, ובתוך מתודות ההוספה האלה, אנו קוראים לבנאים של מחלקות התאים של טבלת ההאש וערימת המקסימום

**private** **static** **class** DoublyLinkedList<T>

**private** **class** DoublyLinkedCell

**private** **class** hashMap

**private** **class** hashCell

**private** **class** maxheap

**private** **class** heapNode