**מבני נתונים תרגיל מעשי 2**

מגישים:

יונתן רום – 314649286

אביב אברהם - 208295691

**מחלקת Graph:**

הגדרנו 4 שדות למחלקה:

1. ערימת מקסימום – maxheap - ערימת מקסימום שאיבריה מכילים מפתח שהוא משקל הסביבה של צומת (סכום משקלו ומשקל שכניו) וערך שהוא הצומת עצמו.
2. טבלת האש – HashTable - טבלת האש שמומשה בשיטת ה chaining כך שפונקציית האש הינה universal hashing קרי, נלקחה ממשפחת פונקציות המודולו באקראי כאשר הראשוני הרלוונטי הוא p = 10^9 + 9 .
3. מספר הצמתים – numNodes – שדה שסופר את כמות הצמתים שבגרף.
4. מספר הקשתות – numEdges – שדה שסופר את כמות הקשתות שבגרף.

הסבר על הפונקציות:

1. Graph(Node [] nodes) – constructor של גרף, מקבל מערך של צמתים מאורך N (כאשר כל אוביקט צומת מכיל את המזהה שלו ואת המשקל שלו) ומאתחלת את מבנה הנתונים כגרף עם N צמתים אלו וללא קשתות. סיבוכיות הזמן הינה , זאת מכיוון שלכל צומת אנו יוצרים צמתים חדשים מהסוג שלו לערימה ולטבלת האש, ומכניסים אותו ב בתוחלת, ומכיוון שאנו זאת לכל צומת, נעשה זאת N פעמים ולכן בתוחלת כנדרש.
2. addEdge(int node1\_id, int node2\_id) – מקבלת שני מזהים של צמתים. אם שני הצמתים נמצאים בגרף, הפעולה מוסיפה קשת ביניהם. אחרת, היא איננה עושה דבר.   
   הפונקציה מחזירה true אם נוספה קשת ואחרת מחזירה false.

אם הצמתים נמצאים, מכניסים כל צומת לרשימת שכנויות של הצומת השני, ומעדכנים בערימת המקסימום את ערכי המשקלים ומחזירים true .

סיבוכיות הזמן הינה זאת מכיוון שפעולת ההכנסה לערימת המקסימום עלתה .

(פעולת מציאת הצמתים ועדכונם עולה בתוחלת. )

1. maxNeighborhoodWeight() - מחזירה צומת כך שמשקל הסביבה שלו מקסימלי. אם בגרף אין צמתים, מחזירה null. מימוש הפונקצייה היא פשוט שליפה מהירה משדה של MaxHeap ולכן .
2. getNeighborhoodWeight(int node\_id) – בהינתן מזהה קודקוד node\_id, מחזיר את משקל הסביבה של הקודקוד בעל מזהה זה אם הוא בגרף. אם אין קודקוד בעל מזהה זה בגרף, מחזירה . הפונקצייה קוראת לפונקציית הhash ומחזירה את הצומת המתקבל מהפונקציה אם הוא קיים. סיבוכיות הזמן היא: .
3. deleteNode(int node\_id) – מוחקת את הקודקוד בעל המזהה node\_id (ואת כל הקשתות המערבות אותו) מהגרף, אם הוא קיים (אחרת, הפעולה לא מבצעת דבר).   
   הפונקציה מחזירה true אם נמחק צומת, ואחרת false.

תחילה אנו מוחקים את הצומת הנתון מטבלת האש ב בתוחלת, לאחר מכן אנו מוחקים אותו מערימת המקסימום ב , לאחר מכן, אנו עוברים בלולאה על כל שכניו של הקודקוד הנתון ומורידים את המפתח שלהם בערימת המקסימום במשקלו של הקודקוד אותו אנו מוחקים, דבר אשר לוקח , ומוחקים אותו מרשימת השכנויות המקושרת שלהם. כמובן שמעדכנים שדות ופינטרים בזמן קבוע. ולכן בסה"כ , כאשר היא דרגתו של הצומת (כלומר, מספר שכניו בגרף כרגע).

1. getNumNodes() מחזירה את מספר הצמתים שנמצאים בגרף כרגע. שמרנו את מספר זה בשדה ועדכנו אותו לפי הצורך, ולכן זהו פשוט שליפה מהירה. מכאן כנדרש.
2. getNumEdges() מחזירה את מספר הקשתות שנמצאות בגרף כרגע. שמרנו את מספר זה בשדה ועדכנו אותו לפי הצורך, ולכן זהו פשוט שליפה מהירה. מכאן כנדרש.

**מחלקת Node :**

הגדרנו שני שדות למחלקה:

1. תעודת זהות – id - מספר מזהה לצמתים, מספר שלם ואי שלילי, לכל צומת יש תעודת זהות מיוחדת.
2. משקל- weight – המשקל של הצומת בגרף, מספר שלם ואי שלילי.

הסבר על הפונקציות:

1. Node(int id, int weight) – בנאי אשר יוצר צומת של גרף עם מזהה id ומשקל weight. .
2. getId()– מחזיר את המזהה של הצומת, .
3. getWeight ()– מחזיר את המשקל של הצומת, .

**מחלקת ערימת המקסימום MaxHeap:**

הגדרנו שלוש שדות למחלקה זאת:

1. heapNode [] arr - מערך של צמתים מסוג heapnode (פירוט המחלקה שלהם בהמשך).
2. lastIndex - מציין את המיקום של האיבר האחרון במערך ערימת המקסימום.

הסבר על הפונקציות:

1. MaxHeap(heapNode [] arr) - בנאי למחלקה, מקבל מערך של צמתים מסוג heapnode, מכניס אותו זמנית לשדה מערך הצמתים, וקורא לפונקציית העזר שמתקנת את המערך לערימת מקסימום ב כפי שראינו בכתה. בנוסף מעדכנים את שדה האינדקס להיות האינדקס התקין כרגע. סה"כ .
2. DeleteMax() - מוחקת את האיבר המקסימלי מהערימה, מחליפה אותו באינדקס האחרון של הרשימה, מורידה את אורך הרשימה ב -1, וקוראת לפונקציית העזר heapify\_down על האיבר המוחלף. שרצה ב .
3. Max() - מחזירה את האיבר המקסימלי במערך, .
4. decrease\_key(int i, int sum) - משנה את ערך המפתח של צומת בערימת המינימום, וקוראת לאחת מפונקציות העזר heapify\_down/ heapify\_up שרצות ב.
5. Delete(int i) - בהנתן אינדקס(משדה המערך) למחיקה מוחקת אותו מהמערך באופן הבא- מחליפה אותו באינדקס האחרון של הרשימה, מורידה את אורך הרשימה ב -1, וקוראת לפונקציית העזר heapify\_down/ heapify\_up על האיבר המוחלף. שרצות ב . [בנוסף יש קריאה לפונקציות העזר switchNodes ו- getParentIndex(i) שרצות בזמן קבוע.]
6. heapify\_up(int i) - פעפוע כלפי מעלה ע"י שימור ערימת המקסימום כפי שראינו בכתה, קוראת ל switchNodes, ורצה בסה"כ ב.
7. heapify\_down(int i)- פעפוע כלפי מטה ע"י שימור ערימת המקסימום כפי שראינו בכתה, קוראת ל switchNodes, ורצה בסה"כ ב.
8. getLeftIndex(int i) - מחזירה את האינדקס של הבן השמאלי של צומת נתון (ע"י האינדקס שלו במערך), רצה ב .
9. getRightIndex(int i) - מחזירה את האינדקס של הבן השמאלי של צומת נתון (ע"י האינדקס שלו במערך), רצה ב .
10. getParentIndex(int i) - מחזירה את האינדקס של ההורה של צומת נתון (ע"י האינדקס שלו במערך),כאשר יוצאים מנקודת הנחה שההורה של האיבר המקסימלי זה הוא עצמו. רצה ב .
11. switchNodes(int i, int parent) פונקצייה זו מחליפה בין זוג צמתים בערימת המקסימום ע"י עדכון מצביעים, רצה ב .
12. array\_to\_MaxHeap() - פונקצייה זו הופכת מערך של צמתים לערימת מקסימום ע"י אלגוריתם שראינו בכתה, היא רצה בלולאה על הצמתים אשר ברמה הלפני האחרונה (עד לאחרונה) ומתקנת אותם ע"י שימוש בפונקציית העזר heapify\_down שרצה ב ב, הראנו בכתה שפונקצייה זו רצה ב כש N מספר הצמתים שבאתחול הערימה.

**מחלקת heapNode :**

הגדרנו 4 שדות למחלקה:

1. key – מייצג את סכום המשקלים של השכנים של הצומת (כולל הצומת עצמו) כלומר משקל הסביבה שלו.
2. Value – הid של הצומת בגרף
3. Index – זה האינדקס שלו במערך שמייצג את ערימת המקסימום
4. hashListNode – פוינטר לצומת המקביל לו בטבלת hash.

הסבר על הפונקציות:

1. heapNode(int sumWeights, int idOfTheNode) – בנאי אשר יוצר צומת של ערימת מקסימום עם מפתח וid .
2. getKey()– מחזיר את משקל הסביבה של הצומת .
3. setKey (int key)– מעדכן את המפתח להיות המפתח שקיבלנו בקריאה .

**מחלקת DoublyHashLinkedList:**

הגדרנו 4 שדות למחלקה:

1. first – מצביע לאיבר הראשון ברשימה.
2. last – מצביע לאיבר האחרון ברשימה.
3. length – כמות האיברים ברשימה

הסבר על הפונקציות:

1. Insert\_first(Graph.Node value,Grapth.heapNode heapNode)– מכניס את האיבר שקיבלנו לתחילת הרשימה המקושרת .
2. remove (HashListNode listNode)– הפונקציה מקבלת מצביע לאיבר ברשימה ומוחקת אותו מהרשימה .
3. Find(int node\_id) – הפונקציה מחפשת את האיבר עם הid שקיבלנו. כיוון שאנחנו עם משתמשים בhashing אז אורך הרשימה הוא בתוחלת וכך גם החיפוש.

ניתוח סיבוכיות: כיוון שזו רשימה מקושרת דו כיוונית ניתן למחוק איבר שקיבלנו מצביע אליו בזמן קבוע, הכנסה לרשימה מקושרת שיש לנו מצביע להתחלה פועלת בזמן קבוע.

**מחלקת hashListNode:**

הגדרנו 5 שדות למחלקה:

1. value – שומר את הצומת של ההקלאס graph שקיבלנו כאשר יצרנו את מבנה הנתנוים.
2. Next – האיבר הבא ברשימה.
3. Prev – האיבר הקודם הרשימה.
4. heapNode – פוינטר לאיבר המקביל שלו בערימת המקסימום.
5. Neighbors – פוינטר לרשימה מקושרת דו כיוונית שמכילה את השכנים של הצומת.

הסבר על הפונקציות:

1. HashListNode(Graph.Node value, heapNode heapNode)– בנאי אשר יוצר צומת של הרשימה המקושרת שנמצאת בטבלת hash .
2. getValue()– מחזיר צומת מסוג Graph.Node שמייצג את הצומת .
3. getHeapNode ()– מחזיר צומת מסוג heapNode שמייצג את הצומת בערימת המקסימום .

**מנשק hashable:**

מנשק עזר שהגדרנו לטובת פונקצית ה hash שתהיה דטרמיניסטית כפי שאנחנו רוצים.

**מחלקת HashTable:**

הגדרנו 3 שדות למחלקה:

1. hashfunction – פונקצית ה hash שמוגרלת ממשפחת הפונקציות המודולריות עם הראשוני הקבוע והפונקציה דטרמיניסטית לכל גרף.
2. DoublyHashLinkedList [] arr – מערך שבכל תא יש לנו רשימה מקושרת דו כיוונית שמטפלת בהתנגשויות
3. Prime number – מייצג את המספר הראשוני שמוגדר בתרגיל.

הסבר על הפונקציות:

1. HashTable(int size)– בנאי אשר מתאחל מערך עם רשימות מקושרות בכל תא וקורא לפונקציה שיוצרת את הטבלה .
2. createHashFunction(int size) – הפונקציה מקבל את גודל הטבלה ומגרילה פונקציתhash ממשפחת הפונקציות המודולריות עם הראשוני הנתון וקבועים a,b שהיא מגרילה .
3. find (int node\_id)– הפונקציה קוראת לפונקציה עזר שמחשבת את מיקום ההכנסה של הצומת לו היה בטבלה (הפונקציה רצה בזמן קבוע), לאחר מכן ניגשת לתא במערך ומחפשת את האיבר ברשימה המקושרת. בתוחלת.
4. Insert (HashListNode node)– הפונקציה קוראת לפונקציה עזר שמחשבת את מיקום ההכנסה של הצומת לו היה בטבלה (הפונקציה רצה בזמן קבוע), לאחר מכן ניגשת לתא במערך ומחפשת את האיבר ברשימה המקושרת, אם האיבר לא נמצא כבר אז מכניסה אותו לרשימה המקושרת בתוחלת.
5. Delete (HashListNode node)– הפונקציה קוראת לפונקציה עזר שמחשבת את מיקום ההכנסה של הצומת לו היה בטבלה (הפונקציה רצה בזמן קבוע), לאחר מכן ניגשת לתא במערך ומוחקת את האיבר ברשימה (בעזרת הפונקציה של הרשימה המקושרת שרצה בזמן קבוע).
6. IndexOfHash (int node\_id)– הפונקציה מבצעת חישוב של פונקצית ה hash עם node\_id ומחזירה את התוצאה .

**מחלקת DoublyLinkedList:**

הגדרנו 4 שדות למחלקה:

1. first – מצביע לאיבר הראשון ברשימה.
2. last – מצביע לאיבר האחרון ברשימה.
3. length – כמות האיברים ברשימה
4. hashListNode – מצביע לצומת שהמופע של הרשימה המקושרת מייצגת אותו.

הסבר על הפונקציות:

1. HashListNode(HashListNode node)– בנאי אשר יוצר צומת רשימה מקושרת חדשה .
2. Insert\_first(ListNode node)– מכניס את האיבר שקיבלנו לתחילת הרשימה המקושרת .
3. delete (ListNode listNode)– הפונקציה מקבלת מצביע לאיבר ברשימה ומוחקת אותו מהרשימה .

ניתוח סיבוכיות: כיוון שזו רשימה מקושרת דו כיוונית ניתן למחוק איבר שקיבלנו מצביע אליו בזמן קבוע, הכנסה לרשימה מקושרת שיש לנו מצביע להתחלה פועלת בזמן קבוע.

**מחלקת ListNode:**

הגדרנו 4 שדות למחלקה:

1. value – מצביע לצומת שיש לו קשת איתה
2. Next – האיבר הבא ברשימה.
3. Prev – האיבר הקודם הרשימה.
4. hashListNode – פוינטר לצומת ברשימה המקושרת של הטבלת hash שמייצגת את הצומת שהצומת הנוכחי נמצא ברשימת השכנים שלו.

הסבר על הפונקציות:

1. ListNode(ListNode value, HashListNode node)– בנאי אשר יוצר צומת של הרשימה המקושרת שמייצגת את רשימת השכנים של כל צומת .

**ניסוי:**

הכנסו כפי שהתבקשנו בשאלה בכל פעם את כמות הנדרשת של הצמתים. לכל צומת נתנו משקל 1, ככה הדרגה של הצומת שווה למשקל שלו פחות 1 (כי הוא עצמו נספר במשקל אך לא בדרגה).

|  |  |  |
| --- | --- | --- |
| **מספר סידורי i** | **N** | **דרגה מקסימלית בגרף** |
| 6 | 64 | 4 |
| 7 | 128 | 4 |
| 8 | 256 | 5 |
| 9 | 512 | 4 |
| 10 | 1024 | 5 |
| 11 | 2048 | 6 |
| 12 | 4096 | 5 |
| 13 | 8192 | 5 |
| 14 | 16384 | 6 |
| 15 | 32768 | 5 |
| 16 | 65536 | 7 |
| 17 | 131072 | 6 |
| 18 | 262144 | 5 |
| 19 | 524288 | 7 |
| 20 | 1048576 | 6 |
| 21 | 2097152 | 7 |

הסבר: הבעיה שמתוארת בשאלה היא מקבילה לבעיה של זריקה של n כדורים כאשר כל כדור נכנס לשני תאים שונים באקראי לתוך n תאים. לפי הערך בויקיפדיה, נראה שהמקרה שלנו דומה למקרה המתואר של זריקת m כדורים ל nתאים וכאשר m=n הביטוי מתאר בהסתברות גבוהה את החסם העליון לכמות הכדורים המקסימלית שתהיה בתא בודד, כלומר במצב שלנו כמות הקשתות שתהיה לצומת.

הביטוי שתיארנו גדל ככל ש n גדל ולכן גם נצפה לראות עלייה (מתונה מאוד) בדרגה המקסימלית עם העלייה בגודל הקלט. לפי התוצאות שקיבלנו ראינו עלייה מתונה בדרגה המקסימלית.