<u>מטלה מעשית במבני נתונים</u>

<u>שמות מגישים:</u>

שי גולדבורט – 207859706 , שם משתמש : goldbourt

אורן שפיגלר – 318638152 , שם משתמש : orenshpigler

<u>תיאור המחלקה LinkedList :</u> מחלקה סטטית המייצגת רשימה מקושרת של תאים שהם מטיפוס Cell (שיפורט בהמשך) .

השדות של כל רשימה מקושרת יהיו :

head – האיבר הראשון (ראש) ברשימה המקושרת .

last – האיבר האחרון ברשימה המקושרת.

size – גודל הרשימה המקושרת (כמות התאים) .

<u>הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות :</u> - LinkedList

<u>Insert_cell (cell c) :</u>

<u>הסבר:</u> מתודה זו מכניסה תא (cell) לרשימה המקושרת כאשר ההכנסה מתבצעת לסוף הרשימה המקושרת .

<u>סיבוכיות:</u> בצענו גישה לשדות שעולות O(1) , יתר הפעולות הינן פעולות קבועות שעולות O(1) , לכן סה"כ סיבוכיות זמן הריצה : O(1).

<u>delete_cell (cell c) :</u>

<u>הסבר:</u> מתודה זו מוחקת את התא c (cell) מהרשימה המקושרת כאשר המחיקה מתבצעת על ידי הפרדה למקרים (מחיקה מראש הרשימה המקושרת , מאמצע הרשימה המקושרת ומסוף הרשימה המקושרת) .

<u>סיבוכיות:</u> בצענו גישה לשדות שעולות O(1) , יתר הפעולות הינן פעולות קבועות שעולות O(1) , לכן סה"כ סיבוכיות זמן הריצה : O(1).

<u>is_in (int id) :</u>

<u>הסבר:</u> מתודה זו מחזירה את ערך המפתח של הצומת במידה והוא צומת אמיתי , אחרת זהו צומת וירטואלי ומחזירה 1- .

<u>סיבוכיות:</u> בצענו מעבר על אברי הרשימה המקושרת ובדיקה האם ערך הזהות של ה Node השייך לתא עליו אנו נמצאים שווה ל id, הפונקציה getId עולה O(1), המעבר על אברי הרשימה עולה O(d) כאשר d מציין את מספר התאים הנוכחי ברשימה המקושרת, לכן סה"כ סיבוכיות זמן הריצה הינה : O(d).

find (int id) :

הסבר: מתודה זו מחזירה את הצומת (Node) בעל הערך זהות id כאשר אנו מניחים לפני הפעלת הפונקציה כי קיים תא אליו שייך Node עם ערך זהות id .

סיבוכיות: בצענו מעבר על אברי הרשימה המקושרת ובדיקה האם ערך הזהות של ה Node השייך לתא עליו אנו נמצאים שווה ל id, לכן סה"כ סיבוכיות זמן הריצה הינה : O(d) , כאשר d מציין את מספר התאים הנוכחי ברשימה המקושרת .

insert_node(Node node) :

הסבר: מתודה זו מבצעת הכנסה של תא (Cell) אותו ניצור כאשר ה- Node יהיה שייך לו . ההכנסה תתבצע לסוף הרשימה המקושרת.

סיבוכיות: גישה לשדות ויצירת תא עולה O(1) , יתר הפעולות הינן פעולות קבועות שעולות O(1) , לכן סה"כ סיבוכיות זמן הריצה :

O(1).

delete_node(Node node):

הסבר: הנחת הפונקציה היא שקיים ברשימה המקושרת תא אשר node שייך לו.

מתודה זו מוחקת מהרשימה המקושרת תא (cell) אשר ה node שייך לו. כאשר תחילה נמצא את התא המתאים על ידי מעבר על אברי הרשימה המקושרת ולאחר מכן נבצע את המחיקה לפי המקרה המתאים (מחיקה מההתחלה , אמצע ומהסוף).

סיבוכיות: חיפוש האיבר על ידי מעבר על אברי הרשימה המקושרת יעלה O(d) , כאשר d מציין את מספר התאים הנוכחי ברשימה המקושרת והמחיקה עצמה תתבצע בזמן O(1) בה בצענו גישה לשדות . יתר הפעולות הינן פעולות קבועות שעולות O(1) , לכן סה"כ סיבוכיות זמן הריצה O(d) .

<u>תיאור המחלקה cell :</u> מחלקה סטטית המייצגת תא (אשר ברשימה מקושרת הוא מאפיין את אבריה) .

השדות של כל תא יהיו :

node – צומת שהוא מייצג .

next – מצביע לתא הבא .

prex – מצביע לתא הקודם .

Parallel_edge – מצביע לייצוג השני של אותה הקשת : בהינתן צומת v , אם cell נמצא ברשימת השכנים של v ומייצג צומת u ,

אז המצביע יהיה ל cell שמייצג את v ברשימת השכנים של u .

<u>הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות :</u> - cell

<u>Cell (Node node) :</u>

<u>הסבר:</u> הבנאי למחלקה , מתודה זו יוצרת תא כאשר הצומת אופן הוא מייצג יאותחל להיות node .

<u>סיבוכיות:</u> גישה לשדה מתבצעת בזמן O(1) , יתר הפעולות הינן קבועות שמתבצעות הזמן O(1) , לכן סה"כ סיבוכיות זמן הריצה

הינו : O(1) .

<u>תיאור המחלקה Node :</u> מחלקה סטטית המייצגת צומת (אשר בתא הוא מאפיין את אחד השדות שלו) .

<u>השדות של כל צומת יהיו :</u>

Index_in_heap – מציין את האינדקס של התא במערך (של ערימת מקסימום בינארית אותה אנו מתחזקים ותפורט בהמשך) אליו הצומת שייך .

weight – מציין את משקל הצומת .

id – מציין את המזהה של הצומת .

neighbors – מצביע לרשימה מקושרת שאבריה יהיו השכנים של הצומת .

neighborhood_sum – מציין את משקל הסביבה של הצומת .

<u>הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות :</u> - Node

<u>Node (int id , int weight):</u>

<u>הסבר:</u> בנאי של המחלקה , מתודה זו יוצרת צומת כאשר תאתחל את השדות id ו- weight של הצומת להיות שווים לאלו שקיבלה בקלט. בנוסף תאתחל את המצביע לרשימת השכנים (שהינה רשימה מקושרת) להצביע לרשימה מקושרת ריקה ואת משקל הסביבה להיות שווה למשקל הצומת .

<u>סיבוכיות:</u> בצענו גישה לשדות שעולות O(1) ,ויצרת רשימה מקושרת ריקה עולה גם כן O(1) . יתר הפעולות הינן קבועות שמתבצעות הזמן O(1) , לכן סה"כ סיבוכיות זמן הריצה הינה : O(1).

<u>getId() :</u>

<u>הסבר:</u> מתודה זו מחזירה את המזהה (id) של הצומת .

<u>סיבוכיות:</u> בצענו גישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות הריצה הינו : O(1) .

<u>getWeight() :</u>

<u>הסבר:</u> מתודה זו מחזירה את המשקל של הצומת .

<u>סיבוכיות:</u> בצענו גישה לשדה שעולה O(1) , לכן סה"כ סיבוכיות הריצה הינו : O(1) .

add_edge_to_node (cell c) :

הסבר: מתודה זו מעדכנת את רשימת השכנים השייכת לצומת עליה היא תופעל כאשר תכניס אליה את התא c על ידי שימוש בפונקציה insert_cell ולאחר מכן תבצע עדכון בערימת מקסימום הבינארית אותה אנו מתחזקים מאחר ומשקל הסביבה עשוי להשתנות על ידי הפונקציה increase_key . **נעיר** כי הפונקציה מיועדת לסייע בעת תהליך יצירת קשת בגרף.

סיבוכיות: סך הכל השתמשנו בפונקציה insert_cell שעולה O(1) ובפונקציה increase_key שעולה O(log n) , כאשר n מציין את מספר הצמתים הנוכחי בגרף . לכן סה"כ סיבוכיות זמן הריצה O(log n) .

delete_edge_from_node (cell c) :

הסבר: מתודה זו מעדכנת את רשימת השכנים השייכת לצומת עליה היא תופעל כאשר תמחק את התא c ממנה על ידי שימוש בפונקציה delete_cell ולאחר מכן תבצע עדכון בערימת מקסימום הבינארית אותה אנו מתחזקים מאחר ומשקל הסביבה עשוי להשתנות על ידי הפונקציה decrease_key . **נעיר** כי הפונקציה מיועדת לסייע בעת תהליך מחיקת קשת מהגרף.

סיבוכיות: סך הכל השתמשנו בפונקציה delete_cell שעולה O(1) ובפונקציה decrease_key שעולה O(log n) , כאשר n מציין את מספר הצמתים הנוכחי בגרף . לכן סה"כ סיבוכיות זמן הריצה O(log n) .

תיאור המחלקה max_heap: מחלקה המייצגת ערימת מקסימום בינארית (אשר המפתחות לפיהם יקבע המיקום של כל איבר בערימה יהיו ערכי משקל הסביבה של כל צומת) .

השדות של כל ערימת מקסימום בינארית יהיו :

arr – מערך של צמתים המייצג ערימת מקסימום הבינארית .

last_item – מציין את האינדקס במערך arr (ערימת מקסימום בינארית) של האיבר האחרון אשר נמצא בה .

הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות : - max_heap

Max_heap(int n) :

הסבר: בנאי של המחלקה , המחלקה יוצרת ערימת מקסימום מגודל n (כאשר בפועל יהיו n-1 איברים בה כי באינדקס 0 לא נבצע השמה לאיבר). בנוסף , נשמור בעזרת המשתנה last_item את המיקום של האחרון בערימה.

סיבוכיות: כפי שנאמר בכיתה לקיחת מערך מגודל n עולה O(1) שכן אנו "לוקחים" מערך מגודל n מבלי לבצע עליו השמות או פעולות כלשהן. בנוסף , בצענו גישה לתא במערך שעולה O(1) , לכן סה"כ סיבוכיות הריצה הינו : O(1) .

Max() :

הסבר: המתודה מחזירה את האיבר בעל ערך השדה neighborhood_sum המקסימלי . מצבעת זאת על ידי גישה לתא באינדקס 1 של המערך arr אשר מציין את "שורש" העץ (שמתאר את הערימץ מקסימום).

סיבוכיות: גישה לתא במערך עולה O(1) , לכן סה"כ סיבוכיות הזמן הינה : O(1) .

Insert(Node node) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מכניסה צומת מטיפוס Node לערימת מקסימום , ומבצעת זאת על ידי כך שמכניסה אותו למקום האחרון שחוקי להכניס בערימת מקסימום על ידי גישה למשתנה last_item ולאחר מכן מבצעת heapifyup על ידי הפונקציה heapify_up . לבסוף, מעדכנת את המיקום של המשתנה last_item .

סיבוכיות: בצענו גישה לשדה ושהשמה שעולה O(1) , לאחר מכן קראנו לפונקציה heapify_up שעולה O(logk) . יתר הפעולות הינן קבועות אשר עולות זמן קבוע , לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

Heapify_up(Node node) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מבצעת את הפעולה Heapity up , על ידי כך שבעזרת לולאה אנו עוברים על המסלול מהצומת Node לכיוון "שורש העץ" שמהווה את האיבר המקסימלי הערימה . נעצור כאשר נגיע למצב בו ערך ה neighborhood_sum של הצומת node קטן מערך ה neighborhood_sum של ההורה שלו. אחרת , אם הערך neighborhood_sum של הצומת node גדול יותר נחליף בין הצמתים הנ"ל כאשר נבצע זאת על ידי גישה לאינדקסים המתאימים במערך שמתאר את הערימת מקסימום. (כפי שראינו בכיתה).

סיבוכיות: בצענו מעבר על מסלול צמתים מהצומת node ועד לכל היותר לשורש העץ כאשר בכל איטרציה נבצע גישות לשדות ועידכונים מתאימים שעולים O(1) , לפיכך יעלה לנו O(logk) כיוון שגובה העץ הינו חסום על ידי O(logk). יתר הפעולות עולות זמן קבוע , לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

Heapify_down(Node node) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מבצעת את הפעולה Heapity down , על ידי כך שבעזרת לולאה אנו עוברים על המסלול מהצומת Node לכיוון "מטה" (העלים של העץ) כאשר נבדוק איזה מהבנים של הצומת node ערך ה neighborhood_sum שלו גדול יותר ונבדוק אם הוא גם גדול יותר מערך ה neighborhood_sum של node ונבצע זאת בעזרת הפונקציה Maximal_son ,אם אכן ערך ה neighborhood_sum של אחד מהבנים גדול יותר אז נבצע החלפה של הצמתים הנ"ל על ידי גישה לאינדקסים המתאימים במערך שמתאר את הערימת מקסימום . (כפי שראינו בכיתה). אם ערך ה neighborhood_sum של 2 הבנים של הצומת node קטנים יותר מערך ה neighborhood_sum של node אז נעצור אז נעצור.

סיבוכיות: בצענו מעבר על מסלול צמתים מהצומת node ועד לכל היותר לרמה התחתונה ביותר של העץ (בה העלה הנמוך שנמצא ברמה התחתונה ביותר נמצא) , כאשר בכל איטרציה נבצע גישות לשדות ,עידכונים מתאימים וקריאה לפונקציה maximal_son, שעולה זמן קבוע, וסה"כ יעלה לנו O(1) , לפיכך יעלה לנו O(logk) כיוון שגובה העץ הינו חסום על ידי O(logk). יתר הפעולות עולות זמן קבוע , לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

Maximal_son(Node node) :

הסבר: מתודה זו מקבלת צומת ובודקת למי מהבנים שלה (במידה ושניהם קיימים אחרת נבדוק עבור מי מהם שקיים במידה וקיים. אם אין לה בנים לא נבדוק) יש ערך neighborhood_sum גדול יותר ומחזירה אותו. אם קיים בן אחד אז תחזיר אותו.

סיבוכיות: בצענו גישה לשדה שעולה O(1),יתר הפעולות הינן קבועות אשר עולות זמן קבוע, לכן סה"כ סיבוכיות הריצה הינו : O(1).

delete(Node node) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מקבלת צומת אותו נרצה למחוק מהערימה ,ומוחקת אותו על ידי החלפת המקום של האיבר האחרון בערימת מקסימום עם האיבר node על ידי גישות לשדות ולאחר מכן נשים את הערך Null בתא של המיקום החדש של node (אשר יהיה האיבר הממוקם באינדקס של האיבר האחרון שקיים הערימה). ולאחר מכן מבצעת תיקון על ידי heapify_down לצומת שהוחלף עם node (כלומר הצומת שבתחילת התהליך האיבר האחרון שקיים במערך).

סיבוכיות: בצענו גישה לשדות שעולה O(1) , בנוסף השתמשנו בפונקציה heapify_up שעולה O(logk) . לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

Increase_key(Node node , int delta) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מקבלת צומת node ומספר delta אי שלילי אשר אותו נרצה להוסיף לערך של השדה neighborhood_sum של הצומת node (כלומר נגדיל את ערכו) ומבצעת זאת על ידי הוספת הערך delta לשדה neighborhood_sum של הצומת ולאחר מכן על מנת לשמור על תקינות העץ מבצעת heapify up על ידי הפונקציה heapify_up .

סיבוכיות: בצענו גישה לשדה שעולה O(1) , בנוסף בצענו קריאה לפונקציה heapify_up שעולה O(logk) , לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

decrease_key(Node node , int delta) :

הערה : K מציין את מספר האברים הנוכחי בערימת מקסימום.

הסבר: מתודה זו מקבלת צומת node ומספר delta אי שלילי אשר אותו נרצה להפחית מערך של השדה neighborhood_sum של הצומת node (כלומר נקטין את ערכו) ומבצעת זאת על ידי החסרת הערך delta מהשדה neighborhood_sum של הצומת ולאחר מכן על מנת לשמור על תקינות העץ מבצעת heapify down על ידי הפונקציה heapify_down .

סיבוכיות: בצענו גישה לשדה שעולה O(1) , בנוסף בצענו קריאה לפונקציה heapify_down שעולה O(logk) , לכן סה"כ סיבוכיות הריצה הינו : O(logk) .

תיאור המחלקה Hash_table : מחלקה המייצגת טבלת hash עם טיפול בהתנגשויות בעזרת chaining. לטבלה פונקציית hash ממשפחת הפונקציות האוניברסאלית מודולו , עם p=10^9+9.

השדות של כל טבלת hash יהיו :

LinkedList [ ] table – טבלה של רשימות מקושרות דו כיווניות .

Int a – המקדם של המשתנה בפונקציה ה - hash

Int b – הקבוע בפונקציה ה - hash

Int p – הראשוני בפונקציה ה - hash

הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות : - Hash_table

hash_table(int n) :

הסבר: בנאי עבור המחלקה Hash_table . הבנאי מקבל את גודל הטבלה הרצוי עבור טבלת ה- hash, ומאתחל אותה. בנוסף, הבנאי מגריל מספרים a ו-b עבור הפונקציה hash שתהיה ממשפחת הפונקציות האוניברסליות (שלמדנו בכיתה) .

סיבוכיות: הגרלת מספרים עולה O(1) , ובנוסף לקיחת מערך מגודל n ללא ביצוע פעולות עליו עולה זמן קבוע כפי שנאמר בכיתה , לפיכך סה"כ סיבוכיות זמן הריצה הינה : O(1) .

Insert(Node node) :

הסבר: המתודה מחשבת את ערך ה-hash של node (נסמנו ב-t) בעזרת הצבת ה-id שלו בפונקציה. לאחר מכן מתבצעת הכנסה של node לרשימה המקושרת שנמצאת במקום ה-t בטבלה.

סיבוכיות: טבלת ה-hash היא בגודל N, כאשר N מספר האיברים ההתחלתי בגרף. ראינו בשיעור שתוחלת הזמן להכנסה של איבר לטבלת hash (עם chaining) בגודל O(n) (n מספר האיברים בטבלה בזמן כל הכנסה) היא O(1). נזכר כי לא יהיה מצב שבטבלה יהיו יותר מ N אברים , ולכן סה"כ סיבוכיות הזמן הינה : O(1) .

Delete(Node node) :

הסבר: המתודה מחשבת את ערך ה-hash של node (נסמנו ב-t) בעזרת הצבת ה-id שלו בפונקציה. לאחר מכן מתבצעת מחיקה של node מהרשימה המקושרת במקום ה-t על ידי קריאה לפונקציה delete_node(node).

סיבוכיות: טבלת ה-hash היא בגודל N, כאשר N מספר האיברים ההתחלתי בגרף. ראינו בשיעור שתוחלת הזמן למחיקה של איבר מטבלת hash (עם chaining) בגודל O(n) (n מספר האיברים בטבלה בזמן כל מחיקה) היא O(1). לכן סה"כ סיבוכיות הזמן הינה : O(1) .

Is_in_table(int id) :

הסבר: המתודה מחשבת את ערך ה-hash של node (נסמנו ב-t) בעזרת הצבת ה-id שלו בפונקציה. לאחר מכן מתבצעת בדיקה האם ה-id שווה ל-id של אחד ה-nodes שנמצאים ברשימה המקושרת שבמקום ה-t בטבלה.

סיבוכיות: טבלת ה-hash היא בגודל N, כאשר N מספר האיברים ההתחלתי בגרף. ראינו בשיעור שתוחלת הזמן לחיפוש של איבר בטבלת hash (עם chaining) בגודל O(n) (n מספר האיברים בטבלה בזמן כל חיפוש) היא O(1). לכן סה"כ סיבוכיות הזמן הינה : O(1) .

Find_in_table(int id) :

הסבר: המתודה מחשבת את ערך ה-hash של node (נסמנו ב-t) בעזרת הצבת ה-id שלו בפונקציה. לאחר מכן מתבצע חיפוש ברשימה המקושרת שבמקום ה-t בטבלה של node בעל id זהה. צומת זה יוחזר. המתודה נקראת רק אם ידוע שהאיבר נמצא בטבלה.

סיבוכיות: הסיבוכיות היא O(1) בתוחלת, מאותן סיבות בדיוק כמו בפונקציה is_in_table .

<u>תיאור המחלקה Graph :</u>מחלקה המייצגת גרף פשוט לא מכוון (שמכיל קודקודים) כאשר לכל קודקוד יהיה מזהה id ומשקל wigth.

<u>השדות של כל Graph יהיו :</u>

hash - טבלת hash המכילה את קודקודי הגרף.

binary_max_heap – ערימת מקסימום שמכילה את קודקודי הגרף, ויחס הסדר בה הוא לפני משקל ה-"שכונה" של כל קודקוד.

NumNodes – מספר הצמתים בגרף.

NumEdges – מספר הקשתות בגרף.

<u>הסבר מפורט על מתודות המחלקה וניתוח הסיבוכיות :</u> - Graph

<u>Graph(Node [] nodes) :</u>

<u>הסבר :</u> בנאי למחלקה Graph. הבנאי מקבל רשימה של צמתים ומאתחל גרף המכיל אותם. ראשית, הבנאי מאתחל טבלת hash וערימת מקסימום בשדות המתאימים, בגודל שווה לגודל מערך הצמתים. הצמתים מוכנסים לטבלת ה-hash של הגרף, ומעודכן מספר הצמתים בגרף. לאחר מכן, הבנאי מכניס את הצמתים לערימת המקסימום (ללא סידור האיברים לאחר כל הכנסה, כמו שראינו בשיעור), ומתבצעות קריאות למתודה heapify_down עבור כל אחד מהצמתים.

<u>סיבוכיות:</u> הכנסה לטבלת hash בגודל O(N) בה יש לכל היותר O(N) צמתים לכל היותר לוקחת O(1) בתוחלת. לכן הסיבוכיות עבור אתחול הטבלה היא O(N). ראינו בשיעור שעל ידי הכנסת N איברים למערך המייצג ערימת מקסימום ללא סידור האיברים לאחר כל הכנסה, ולאחר מכן ביצוע פעולות heapify_down עבור כל איבר, ניתן לאתחל ערימת מקסימום ב-O(N). לכן הסיבוכיות הכוללת היא O(N).

<u>maxNeighborhoodWeight() :</u>

<u>הסבר :</u> המתודה מחזירה את האיבר המקסימאלי בערימת המקסימום של הגרף, כלומר את הצומת שסכום המשקל שלו ושל השכנים שלו הוא הגדול ביותר מבין כל הצמתים בגרף. אם אין בגרף צמתים, המתודה מחזירה null. הפעולה מתבצעת על ידי קריאה למתודה max שנמצאת במחלקה binary_heap.

<u>סיבוכיות:</u> המתודה max במחלקה binary_heap מתבצעת ב-O(1) לכן הסיבוכיות היא O(1).

<u>getNeighborhoodWeight(int node_id) :</u>

<u>הסבר:</u> המתודה מקבלת מזהה של צומת בגרף, ומחזירה את סכום המשקלים שלו ושל שכניו. אם אין צומת מתאים בגרף, תחזיר 1- . בדיקה האם קיים צומת עם מזהה זהה בגרף מתבצעת על ידי קריאה לפונקציה is_in_table שבודקת אם קיים צומת עם מזהה זהה בטבלת ה-hash של הגרף. אם קיים צומת כזה, המתודה קוראת למתודה find_in_table שמוצאת את הצומת בטבלה. לאחר מכן מוחזר השדה שמכיל את סכום המשקלים של הצומת ושכניו (שדה זה הוא שדה של המחלקה Node).

<u>סיבוכיות:</u> המתודות is_in_table ו- find_in_table מתבצעות בסיבוכיות O(1) בתוחלת, ולכן זו הסיבוכיות הכוללת.

<u>getNumNodes() :</u>

<u>הסבר:</u> המתודה מחזירה את מספר הצמתים בגרף על ידי החזרת הערך שבשדה NumNodes.

<u>סיבוכיות:</u> גישה לשדה עולה O(1) לכן סה"כ סיבוכיות הזמן הינה : O(1) .

: GetNumEdges()

הסבר: המתודה מחזירה את מספר הקשתות בגרף על ידי החזרת הערך שבשדה NumEdges.

סיבוכיות: גישה לשדה עולה O(1) לכן סה"כ סיבוכיות הזמן הינה : O(1) .

: addEdge (int node1_id, int node2_id)

הסבר: המתודה מקבלת מזהים של שני צמתים בגרף, ומוסיפה קשת ביניהם. אם אחד הצמתים לא נמצא בגרף, המתודה תחזיר false. אחרת, נוצרה קשת בין הצמתים ותחזיר true. חיפוש הצמתים המתאימים מתבצע על ידי המתודה find_in_table בטבלת ה-hash של הגרף. לאחר מכן, המתודה יוצרת שני שדות מסוג cell, כאשר בכל אחד יש שדה עם הצומת המתאים. מתבצעת השמה של השדה parallel_edge עבור כל אחד מה-cell-ים – להיות ה-cell השני. כך נוכל לעבור ביעילות משני הייצוגים של כל קשת במהלך מחיקה של צומת מהגרף. לבסוף מתבצעת קריאה ל-add_edge_to_node עבור כל אחת מהצמתים, כדי להוסיף את הצומת השני לרשימת השכנים שלו, ולעדכן את משקל הסביבה שלו.

סיבוכיות: חיפוש הצמתים בטבלת ה-hash לוקח זמן קבוע בתוחלת. המתודה add_edge_to_node מתבצעת ב-O(logn), ולכן זוהי הסיבוכיות הכוללת.

: deleteNode(int node_id)

הסבר: המתודה מקבלת מזהה של צומת בגרף ומוחקת אותו מהגרף. אם לא קיים צומת כזה יוחזר false. אם קיים – true. בדיקה האם הצומת נמצא בגרף מתבצעת על ידי חיפושו בטבלת ה-hash. המחיקה מתבצעת על ידי גישה לרשימת השכנים של הצומת שנרצה למחוק, ובעזרת גישה לשדה parallel_edge, מחיקת הצומת מרשימת השכנים של כל צומת שנמצא איתו בשכנות, על ידי קריאה למתודה delete_edge_from_node. לבסוף נמחוק את הצומת מערימת המקסימום של הגרף, ומטבלת ה-hash.

סיבוכיות: חיפוש הצומת המתאים בטבלת ה-hash לוקח זמן קבוע בתוחלת. המתודה delete_edge_from_node מתבצעת בסיבוכיות O(logn), עבור כל אחד משכניו של הצומת בנפרד. מחיקת הצומת מערימת המקסימום מתבצעת בO(logn), ומחיקתו מטבלת ה-hash מתבצעת בזמן קבוע בתוחלת. לכן הסיבוכיות הכוללת היא O((dv+1)*logn), כאשר dv הוא מספר השכנים (דרגה) של הצומת שנמחק.

מדידות :

1. ניסוי 1

מספר סידורי i	n	דרגה מקסימלית בגרף
6	64	6
7	128	7
8	258	8
9	512	8
10	1024	9
11	2048	9
12	4096	10
13	8192	10
14	16384	10
15	32768	11
16	65536	11
17	131072	11
18	262144	11
19	524288	11
20	1048576	12
21	2097152	12

הקשר בין הדרגה המקסימלית של קודקוד בגרף למשקל הסביבה שלו בכדי למצוא את הדרגה המקסימלית שלו :

כל קודקוד שהכנסנו לגרף אתחלנו את המשקל שלו להיות שווה לסיפרה 1.

על ידי כך, כאשר בסוף תהליך הניסוי (בפרט, אחרי ההכנסה של הקשתות) כאשר נרצה לדעת מהי הדרגה של הצומת המקסימלי, אחרי שנקרא לפונקציות maxNeighborhppdWeight ול – getNeighborhoodWeight הערך שנקבל פחות 1 יהיה הדרגה של הצומת המקסימלי (פחות 1 כי זהו המשקל של הצומת עצמו).

איך הדרגה המקסימלית גדלה ביחס ל n :

קצב הגדילה בין הדרגה המקסימלית ל n הינו איטי מאוד .

<u>הקשר ל- Balls into bins problem :</u>

הבעיה שלפנינו הינה הגרלת קשתות עבור 2 קודקודים שונים כאשר לא נוכל להרגיל קשת שהוגרלה פעם נוספת, בעיה זו דומה להבעיה Balls into bins problem מאחר ואם נעשה אנלוגיה מהבעיה שלנו לתחום הקומבינטורי. נקבל כי אנו רוצים להכניס 2n קשתות ל n קודקודים כאשר הקשתות יהוו עבורנו את הכדורים והקודקודים יהוו את התאים (בדומה לבעיית ה- Balls into bins problem).

נבחין כי במקרה שלנו מאחר ואנו מגרילים קשת עם הגבלות של אי חזרה על עצמה והגבלה נוספת שלא ייתכן שהיא תורכב מאותו הקודקוד אז היא לא תהיה זהה להבעית ה- Balls into bins problem . אך מכיוון וההסתברות שנבחר את אותו הקודקוד וההסתברות שנבחר קשת שכבר הוגרלה פעם נוספת זניח אז נוכל להסיק שאם נחשב את התוצאות שקבלנו ביחס לנוסחה שמוצגת בבעית ה - Balls into bins problem תהיה קרובה.

ואכן אם נשתמש בנוסחא המתאימה למקרה שלנו כאשר m = 2n ו- n' = n כאשר m , n' אלו הפרמטרים שמוצגים בבעית ה -Balls into bins problem והמקרה שבו m > nlogn עם הנוסחה המתאימה שהיא : $\frac{m}{n} + \# \left(\frac{mlogn}{n}\right)^{0.5}$

כאשר # מציין תטא .

נקבל תוצאות יחסית קרובות לאלו שקיבלנו (אשר מוצגים בטבלה לעיל).