בס"ד

חלק יבש:

תיאור כללי:

נשתמש ב-UnionFind על מנת להחזיק את כל החברות. כל חברה תחזיק עץ דרגות של עובדים עם שכר גדול מאפס, תחזיק hashTable של כל העובדים בחברה (כולל אלו עם שכר שווה ל-0), ותחזיק שני משתנים – אחד שיספור כמה עובדים יש בחברה עם שכר שווה ל-0, ועוד אחד שיספור את הדרגות של עובדים הללו.

כל עובד יחזיק את המזהה של החברה שהוא עובד בה, את המשכורת שלו, המזהה שלו ודרגה שלו.

במערכת שלנו יהיה לנו את החברות שלנו, עץ דרגות של כל העובדים במערכת (ששכרם גדול מאפס), hashTable של כל העובדים במערכת (כולל אלו ששכרם שווה ל-0) ושני משתנים בדומה למשתנים של החברה.

הגדרת מבני הנתונים שלנו:

1) System:

מבנה הנתונים המרכזי המכיל את כל המידע הנדרש לגבי העובדים ולגבי החברות במערכת, אשר מכיל את השדות הבאים:

- Size – משתנה מטיפוס שלם, מחזיק את מספר החברות הנמצאות במערכת.
- numEmpsWithSalZ – משתנה מטיפוס שלם, מחזיק את מספר העובדים במערכת שהמשכורת שלהם הינה 0.
- sumEmpswithSalZ – משתנה מטיפוס שלם, מחזיק את סכום הדרגות של העובדים במערכת שהמשכורת שלהם הינה 0.
- companies – מצביע מטיפוס UnionFind, המכיל את כל החברות במערכת ממוינות לפי ID (החברות הינם מטיפוס Company עליו נפרט בהמשך).
- allEmployees – מצביע לטבלת ערבול (hashTable) אשר מכילה את כל העובדים במערכת ממוינת לפי ID (עובדים מטיפוס Employee עליו נפרט בהמשך).
- salarySortedEmpNoZ – מצביע לעץ דרגות של כל העובדים במערכת שהשכר שלהם גדול מ-0 הממוין לפי משכורת כעדיפות ראשונה ו-ID בעדיפות השנייה. הצמתים בעץ הינם מטיפוס Employee עליו נפרט בהמשך.

2) Company:

מבנה נתונים המייצג חברה במערכת המכיל את השדות הבאים:

- Id – משתנה מטיפוס שלם המחזיק את ה-ID של החברה.
- Value – משתנה מטיפוס double המחזיק את שווי החברה.
- numEmpsWithSalZ – משתנה מטיפוס שלם, מחזיק את מספר העובדים בחברה שהמשכורת שלהם הינה 0.
- sumEmpswithSalZ – משתנה מטיפוס שלם, מחזיק את סכום הדרגות של העובדים בחברה שהמשכורת שלהם הינה 0.
- salarySortedEmpNoZ – מצביע לעץ דרגות של כל העובדים בחברה שהשכר שלהם גדול מ-0 הממוין לפי משכורת כעדיפות ראשונה ו-ID בעדיפות השנייה. הצמתים בעץ הינם מטיפוס Employee עליו נפרט בהמשך.
- compEmployees - מצביע לטבלת ערבול (hashTable) אשר מכילה את כל העובדים בחברה ממוינת לפי ID (עובדים מטיפוס Employee עליו נפרט בהמשך).

3) Employee:

מבנה נתונים המייצג עובד במערכת המכיל את השדות הבאים:

- Id – משתנה מטיפוס שלם המחזיק את ה-ID של העובד.
- Salary – משתנה מטיפוס שלם המחזיק את המשכורת של העובד.
- Grade – משתנה מטיפוס שלם המחזיק את דרגת העובד.
- companyID – מצביע למשתנה מטיפוס שלם המחזיק את ה-ID של החברה אליה נרצה שהעובד ישתייך.

4) hashTable:

מבנה נתונים המייצג טבלת ערבול כפי שלמדנו בכיתה המכיל את השדות הבאים:

- Arr – מערך של מצביעים מטיפוס Ht_item (עליו נפרט בהמשך) אשר יחזיק את הצמתים בטבלה.
- Counter – מונה מטיפוס שלם אשר יחזיק את גודל מספר האובייקטים בטבלה.
- size – משתנה מטיפוס שלם אשר יחזיק את גודל הטבלה.

5) Ht_item:

מבנה נתונים המייצג צומת בטבלת ערבול המכיל את השדות הבאים:

- Key – מצביע למשתנה גנרי היכיל מפתח של צומת.
- Data – מצביע למשתנה גנרי היכיל את המידע של הצומת.
- Next – מצביע ל-Ht_item המייצג את האיבר הנמצא אחרי האיבר הנוכחי.

6) rankedTree:

מבנה נתונים המייצג עץ דרגות כפי שלמדנו בביתה אשר מכיל את השדות הבאים:

- root – מצביע למשתנה מטיפוס Tnode (עליו נפרט בהמשך) אשר מייצג את שורש העץ.
- Counter – משתנה מטיפוס שלם המייצג את מספר הצמתים אשר נמצאים בעץ כרגע.
- Max_node – מצביע למשתנה מטיפוס גנרי אשר מייצג את ערך המפתח המקסימלי בעץ.

7) Tnode:

מבנה נתונים המייצג צומת בעץ הדרגות אשר מכיל את השדות הבאים:

- Key – מצביע למשתנה גנרי היכיל מפתח של צומת.
- Data – מצביע למשתנה גנרי היכיל את המידע של הצומת.
- Left – מצביע לבן השמאלי של הצומת.
- Right – מצביע לבן הימני של הצומת.
- Parent – מצביע לאבא של הצומת.
- Height – מכיל את גובה הצומת (כפי שלמדנו בביתה).
- Tree_nodes – מכיל את מספר הצמתים שיש בתת העץ של הצומת הזו.
- Sum_grades – מכיל את סכום הדרגות (אותו נגדיר בהתאם למבנה – בתרגיל זה הדרגה תהיה דרגת העובד) של כל הצמתים בתת העץ של צומת זו.

8) unionFind:

מבנה נתונים המייצג UnionFind (התומך בכיווץ מסלולים ומכיל עצים הפוכים) כפי שלמדנו בביתה אשר מכיל את השדות הבאים:

- Arr – מערך של מצביעים מטיפוס Unode (עליו נפרט בהמשך).
- arrSize – גודל המערך.

9) Unode:

מבנה נתונים המייצג צומת ב-UnionFind המכיל את השדות הבאים:

- Id – המספר המזהה של הצומת (שבתרגיל זה הוא יהיה המספר המזהה של החברה אותה הצומת מייצג).
- Sub_nodes – מספר הבנים שיש לצומת.
- Data – מצביע למשתנה מטיפוס חברה (החברה אותה הצומת מייצג).
- Parent – מצביע לאבא של הצומת.
- Root_flag – דגל המסמן האם הצומת הינו השורש של העץ ההפוך או לא.

סימונים: n – מספר העובדים במערכת, k – מספר החברות במערכת.

עץ דרגות: מימשנו עץ דרגות גנרי עם סכומים כפי שנלמד בהרצאות ובתרגולים ובנוסף מימשנו את הפונקציות הבאות:

mergeTree – פונקציה המקבלת עץ דרגות, וממזגת בינו לבין העץ דרגות אשר קראנו ממנו לפונקציה.

1. ראשית נהפוך את העצים למערך ממוין באמצעות סיור InOrder (בסה"כ 2 מערכים של הצמתים של העצים תחילה ומערך אחד של האיחוד שלהם לאחר מכן).
2. מיזוג המערכים באמצעות MergeSort למערך אחד ממוין.
3. מחיקת כל הצמתים מהעץ ממנו נקראה הפונקציה.
4. קריאה לפונקציה הרקורסיבית arrayToTree שמקבלת את המערכים הסופיים, אינדקסים של התחלה וסיום, ומצביע לסטטוס הפעולה, והופכת אותו לעץ באמצעות האלגוריתם הבא:
   1. הכנסת Array[middle] לשורש.
   2. קריאה לרקורסיבית לתת העץ השמאלי.
   3. קריאה רקורסיבית לתת העץ הימני.

סיבוכיות הזמן: הפיכת העצים למערך ממוין, מיזוג למערך ממוין, מחיקת כל הצמתים והמרה לעץ – $O(d+m)$ כאשר d,m מייצגים את מספר הצמתים בכל אחד מהעצים.

deleteAllNodes – פונקציה רקורסיבית אשר מוחקת את כל הצמתים בעץ.
סיבוכיות הזמן: $O(m)$ כאשר m הוא מספר הצמתים בעץ.
נשים לב שבמקרה הגרוע יהיו כ-$O(log\ m)$ קריאות במחסנית.

arrayToTree – פונקציה רקורסיבית שמקבלת מערך מצביעים לצמתים בעץ ומחזירה עץ דרגות הממוין לפי המפתחות של הצמתים אשר קיבלה. הפונקציה מחזירה מצביע לשורש של העץ שנוצר מהמערך.
סיבוכיות הזמן: $O(m)$ כאשר m הוא מספר האיברים במערך.

inOrder – פונקציה רקורסיבית המקבלת מערך ומכניסה את הצמתים בעץ לתא המתאים במערך לפי סיור InOrder אשר למדנו בביתה.
סיבוכיות הזמן: $O(m)$ כאשר m הוא מספר הצמתים בעץ.

specialInOrder – פונקציה המקבלת עץ, counter ומשתנה sum ומספר הצמתים שהפונקציה צריכה לסכום. הפונקציה מקבלת ב-sum את סכום כל הדרגות של העץ וב-counter את מספר כל הצמתים בעץ, והיא הולכת ימינה במורד העץ ומתחילה לחסר צמתים מתת העץ השמאלי בכל פעם עד שנגיע למצב בו counter==len (כלומר סכמנו את מספר הצמתים אשר היינו צריכים לסכום). בעצם, הפונקציה לכל היותר תעבור מסלול באורך גובה העץ.

סיבוכיות הזמן: $O(log\ m)$ כאשר m הוא מספר הצמתים בעץ (כגובה העץ).

UnionFind: מימשנו כפי שראינו בתרגול ובהרצאות, בנוסף השתמשנו בכיווץ מסלולים, בשיטת הארגזים (בדומה למה שראינו בתרגול) ובעצים הפוכים. פונקציות מיוחדות אשר הוספנו:

Shrink: פונקציה זו עוברת על מסלול מסוים מבן לשורש, ומחברת כל צומת במסלול לשורש ישירות.

סיבוכיות הזמן: $O(log\ m)$ כאשר m הוא מספר הצמתים הכללי במבנה.

HashTable: מימשנו בעזרת DynamicArray , אשר הוסבר בהרצאות והתרגולים.
הפונקציה שבחרנו לעבוד איתה היא : mod(size_of_DynamicArray) , כאשר גודל המערך בתחילת התוכנית מוגדר להיות גודל 10. פונקציות מיוחדות שהוספנו, מלבד find,insert,remove (סיבוכיות זמן של $O(1)$ בממוצע על הקלט) :

reHash: פונקציה אשר בודקת בעת הכנסה/הוצאה של עובד למערכת אם יש צורך להקטין את גודל המערך/להגדיל. אם יתבצע שינוי אשר או שהגודל של המערך יגדל פי 2 / יקטן פי 2.
סיבוכיות הזמן: כפי שלמדנו בהרצאה , ביצוע rehash היא פעולה יקרה אך נדירה. יקרה מכיוון שעלינו להעתיק את טבלת הערבול שלנו לטבלה גדולה/קטנה יותר איבר איבר – $O(n)$ כאשר n מייצג את מספר האובייקטים בטבלה. אך, מכיוון שהפעולה נדירה , סיבוכיות הזמן המשוערך היא $O(1)$.
סיבוכיות המקום: אם כעת גודל טבלת הערבול היא m ,ועלינו להקצאות מערך חדש בגודל המתאים, במקרה הגרוע אם נגדיל את טבלת הערבול נצטרך להקצאות טבלה הגדולה פי 2 מהמקורית: $2m$, ולכן סיבוכיות המקום היא: $O(m)$.

mergeHTs: היא פונקציה האחראית לשלב בין 2 טבלאות ערבול . הפונקציה עוברת איבר -איבר בטבלת הערבול של "הנרכשת" ומוסיפה אותם לטבלת הערבול של "הרוכשת". מבחינת סיבוכיות זמן , אם נסמן שעבור טבלה 1 מספר האובייקטים הוא n ובטבלה השניה היא p אז במקרה ונצטרך לבצע rehash אז
סיבוכיות הזמן : $O(n+p)$, ובמשוערך $O(1)$.
סיבוכיות המקום: נובעת בעקבות rehash ולכן : $O(n+p)$.

בס"ד

הפונקציות אשר נדרשנו לממש בתרגיל:

Void* init(int k): ראשית בודקת האם k קטן מ-1, כנדרש ולאחר מכן יוצרת מבנה נתונים מטיפוס System ומאתחלת את ה-UF של החברות שלו. במידה ואחד השלבים בתהליך נכשל, נחזיר nullptr.

סיבוכיות זמן: $O(k)$ – אתחול ה-UF עם K חברות כפי שראינו בהרצאה.

StatusType addEmployee(void *DS, int EmployeeID, int CompanyID, int Grade):

ראשית בודקת שהתנאים הנדרשים בתרגיל מתקיימים, ולאחר מכן נבדוק האם קיים ה-hashtable ב-system של כל העובדים (אליו נרצה להכניס את העובד), במידה ולא קיים ניצור אחד כזה ובמידה וקיים נמשיך הלאה (יצירת מערכת בגודל 10 ואתחול שלו יקח O(1). נעדכן את המשתנים של המערכת של מספר העובדים עם משכורת 0 ואת סכום הדרגות של העובדים עם משכורת 0 ולאחר מכן נכניס את העובד ל-hashTable של כל העובדים במערכת. נמשיך הלאה ונחפש את החברה של העובד ב-UF (בסיבוכיות זמן כפי שראינו בהרצאה) ולאחר שנמצא נעדכן את המשתנים כפי שעשינו עבור המערכת, ונבצע את אותן הפעולות עבור ה-hashTable של החברה.

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

הכנסה ל-hashTable של המערכת של כל העובדים (AllEmployees) – O(1) משוערך בממוצע על הקלט.

הכנסה ל-hashTable של החברה בה העובד עובד – O(1) משוערך בממוצע על הקלט.

חיפוש ב-UF של החברות (באמצעות find) – O(log*k).

עדכון משתנים – O(1). לכן סך הכל סיבוכיות הזמן היא: O(log*k) במשוערך בממוצע על הקלט.

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

StatusType removeEmployee(void *DS, int EmployeeID):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ונבדוק האם עובד קיים (על ידי חיפוש ב-hashTable של כל העובדים במערכת). אם הוא קיים, נמשיך הלאה ונמצא את החברה שהוא עובד בה (ב-O(1) מכיוון שאנו שומרים עבור כל עובד את החברה שלו העדכנית ולכן זו רק גישה למערך). נבדוק האם המשכורת שלו גדולה מ-0, אם לא אז נמחוק אותו רק מה-HT של המערכת ושל החברה שהוא עובד בה, אחרת: נמחוק את העובד מעץ ה-SalarySorted של המערכת ומעץ ה-SalarySorted של החברה וגם נבצע את המחיקות מה-HT של המערכת ושל החברה כפי שציינו מקודם.

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

חיפוש ב-HT של העובד – O(1) משוערך בממוצע על הקלט.

גישה למערך של ה-UF ושליפת החברה שלו – O(1).

מחיקה מ-HT – O(1) בממוצע על הקלט.

מחיקה מהעצים – O(log n). סך הכל סיבוכיות הזמן היא: O(log n) במשוערך בממוצע על הקלט.

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

StatusType acquireCompany(void *DS, int AcquirerID, int TargetID, double Factor):

ראשית נבדוק אם החברות קיימות והאם הפקטור גדול מ-0.
נחפש בUF את 2 החברות הרצויות . נבצע find לכל אחת מהם על מנת למצוא את החברה שרכשה את החברה הנ"ל כאשר בפונקציית find אנחנו גם מבצעים את shrink (כיווץ מסלולים) ב-$O(\log* k)$.
לאחר מכן נבצע איחוד של העצים בכל אחת מהחברות לעץ יחיד עבור החברה הרוכשת בעזרת mergeTrees : $O(n+m)$ כאשר n,m מייצגים את מספר העובדים בכל חברה. לאחר מכן נבצע איחוד של טבלאות הערבול שמחזיקה כל חברה לטבלת ערבול יחידה עבור החברה הרוכשת - $O(n+m)$ מתבצעת על ידי פונקציית mergeHTs .
בנוסף, נבצע את פונקציית Union אשר מבצעת שוב find: $O(\log* k)$, מקשרת בין 2 העצים ההפוכים - $O(1)$, ודואגת לשינוי flagRoot בהתאם- $O(1)$.

לבסוף, נשתמש בפונקציה adjustCompanyID (עליה נפרט בהמשך) אשר תשנה את הערך של המזהה של החברה של כל העובדים בחברה החדשה (לאחר הרכישה).
בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן: union,find ו- shrink מתבצעות ב $O(\log* k)$ משוערך. עבור r פעולות האיחוד של עצי הדרגה וטבלאות הערבול סיבוכיות זמן זהה : $O(n+m)$ משוערך. ולכן בסה"כ עבור r פעולות סיבוכיות הזמן המשוערכת היא: $O(\log^* k+n+m)$.

סיבוכיות מקום: איחוד של העצים וטבלאות הערבול מעלות את סיבוכיות המקום : $O(n+m)$ מכיוון שמקרה הגרוע , אנחנו מקצים בטבלאות הערבול בגודל שונה בעקבות reHash. אך נשים לב שעדיין אנו לא חורגים מן המותר בתרגיל.

פונקציית עזר שהגדרנו:

adjustcompanyID: הפונקציה מקבלת *hashTable* של עובדים, *ID* ישן, ו-*ID* חדש (אליו נרצה לשנות). כל עובד מכיל מצביע למשתנה מטיפוס שלם שמכיל את המזהה של החברה בה העובד עובד. הפונקציה *companyID* תעבור על כל ה-*hash* ותחליף לכל העובדים שם את ה-*ID* הישן לחדש במידת הצורך (אלא אם כבר יש לו את ה-*ID* החדש). בגלל שאנו משנים מצביע, זה גם יעדכן את הנתונים אוטומטית עבור העובדים הללו בעצים ובטבלת הערבול של המערכת.

סיבוכיות זמן: אנו עוברים על *hash*, ולכן כמספר האיברים ב-*hash*, לכן: $O(n_{acquire}+n_{target})$.

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

StatusType employeeSalaryIncrease(void *DS, int EmployeeID, int SalaryIncrease):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ונבדוק האם עובד קיים (על ידי חיפוש ב-hashTable של כל העובדים במערכת). אם הוא קיים, נמשיך הלאה ונמצא את החברה שהוא עובד בה (ב-O(1) מכיוון שאנו שומרים עבור כל עובד את החברה שלו העדכנית ולכן זו רק גישה למערך). נעדכן את המשכורת שלו ב-HT של המערכת וב-HT של החברה. אם המשכורת הקודמת שלו הייתה שווה ל-0, נוסיף אותו לעץ הדרגות של המערכת ממוין לפי משכורות, ולעץ הדרגות של החברה ממוין לפי משכורות ונעדכן את המשתנים של המערכת ושל החברה (של עובדים שעבורם המשכורת הינה 0). אחרת, ניגש לעצי הדרגות של המערכת ושל החברה ונעדכן עבור העובד בהם את המשכורת למשכורת העדכנית.

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

נשים לב שבסה"כ יש לנו חיפוש ב-HT (שיתקיים ב-O(1) משוערכת/ממוצע), חיפוש בעץ דרגות (שיתקיים ב-O(log n)) וגישה למערך של ה-UF (שיתקיים ב-O(1) כפי שציינו), ועדכון משתנים ב-O(1).

סך הכל סיבוכיות הזמן היא: O(log n).

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

StatusType promoteEmployee(void *DS, int EmployeeID, int BumpGrade):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ונבדוק האם עובד קיים (על ידי חיפוש ב-hashTable של כל העובדים במערכת). אם הוא קיים, נבדוק האם ה-bumpGrade גדול מ-0. אם לא, אז סיימנו. אם כן, נעלה את הדרגה של העובד ב-HT של המערכת, ב-HT של החברה שלו, ובמידה והמשכורת שלו גדולה מ-0 אז בנוסף בעצי הדרגות של המערכת ושל החברה בהתאמה. בנוסף, לאחר מציאת הצומת של העובד בעץ הדרגות, נחזור מעלה במעלה העץ ונעדכן את הסכומים של הדרגות בכל צומת בעץ.

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

נשים לב שבסה"כ יש לנו חיפוש ב-HT (שיתקיים ב-O(1) משוערכת/ממוצע), חיפוש בעץ דרגות (שיתקיים ב-O(log n)) וגישה למערך של ה-UF (שיתקיים ב-O(1) כפי שציינו), ועדכון משתנים ב-O(1). כפי שציינו, לאחר מציאת הצומת בעץ הדרגות, נחזור מעלה במעלה העץ על מנת לעדכן את הסכומים, מה שיקח גם O(log n).

סך הכל סיבוכיות הזמן היא: O(log n).

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

בס"ד

StatusType sumOfBumpGradeBetweenTopWorkersByGroup (void *DS, int CompanyID, int m):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ולאחר מכן נבדוק האם ה-companyID שווה ל-0. במידה וכן, נעבוד עם עץ הדרגות של המערכת, ואחרת נעבוד עם עץ הדרגות של החברה. נבדוק האם עץ איתו אנו עובדים ריק, אם כן נחזיר FAILURE (כנדרש), אחרת נמשיך בתהליך. כעת, נקרא לפונקציה specialInOrderAssist של העץ עם מצביע למשתנה sum, ונדפיס את ההודעה הנדרשת במקרה של הצלחה עם הערך המעודכן של sum לאחר השליחה לפונקציה. הפונקציה SpecialInOrderAssist תבצע קריאה לפונקציה specialInOrder (פירטנו על הפונקציה למעלה) עם מצביע לקאונטר num, ועם מצביע ל-sum אך לפני כן תעדכן את המצביע sum להיות ערך הדרגות של כל העובדים בעץ.

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

חיפוש החברה ב-UF – O(log*k) משוערך.

ביצוע של הפונקציה specialInOrderAssist – היא מבצעת קריאה ל-SpecialInOrder עם הפרמטרים שנדרשים ולכן סך הכל O(1) ועוד O(log n) של הפונקציה SpecialInOrder וזאת כפי שהוכחנו למעלה.

עדכון משתנים והשמת מצביעים – O(1).

סך הכל סיבוכיות הזמן היא: O(log*k +log n) משוערך.

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

StatusType averageBumpGradeBetweenSalaryByGroup (void *DS, int CompanyID, int lowerSalary, int higherSalary):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ולאחר מכן נבדוק האם ה-companyID שווה ל-0. במידה וכן, נעבוד עם עץ הדרגות של המערכת, ואחרת נעבוד עם עץ הדרגות של החברה (לאחר שנחפש אותה ב-UF). נבדוק האם עץ איתו אנו עובדים ריק, אם כן נחזיר FAILURE (כנדרש), אחרת נמשיך בתהליך. כעת, נקרא לשתי הפונקציות AverageBumpEmpsHigher, AverageBumpEmpsLower עם מצביעים למשתנים num,sum ועם higherSalary,lowerSalary בהתאמה. הפונקציות הללו בעצם בסוף ריצת שתיהן יסכמו לתוך sum ולתוך num את דרגות העובדים ואת מספר העובדים אשר לא מקיימים את הדרישות של השכר בתרגיל ולכן לא נרצה להתייחס אליהן (השכר שלהם גבוה מדי או נמוך מדי). לאחר מכן, נדפיס את ההודעה המתאימה כאשר סכום הדרגות יהיה סכום הדרגות בעץ פחות הערך שקיבלנו ב-sum (פחות הדרגות של העובדים הלא רלוונטים) ומספר העובדים יהיה מספר העובדים במערכת פחות הערך שקיבלנו ב-num (פחות מספר העובדים הלא רלוונטים) והערך שיודפס זה חילוק ביניהם (על מנת להדפיס את הממוצע).

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

סיבוכיות זמן:

עדכון משתנים והשמת מצביעים – O(1).

חיפוש החברה ב-UF במקרה הגרוע – O(log*k) משוערך.

קריאה לפונקציה AverageBumpEmpsLower – O(log n).

קריאה לפונקציה AverageBumpEmpsHigher – O(log n).

סך הכל סיבוכיות הזמן הינה O(log*k + log n).

סיבוכיות המקום אינה חורגת מהסיבוכיות המקום המותרת.

פונקציות העזר אשר מימשנו בשביל הפונקציה:

AverageBumpEmpsLower: הפונקציה מקבלת צומת של עובד (מתוך עץ דרגות ממוין לפי שכר), מצביע לקאונטר, מצביע למשתנה של סכום, וערך lower. הפונקציה בצורה רקורסיבית תבדוק האם המשכורת של העובד הנוכחי גדולה מ-lower, אם כן היא תמשיך לפנות שמאלה. אם לא, אז היא תסכום ל-sum את הדרגות של העובדים בתת העץ הנוכחי ותסכום לקאונטר את מספר העובדים בתת העץ הנוכחי. כעת, עבור מקרה זה בלבד, נבדוק האם יש תת עץ ימני, אם כן,

נחסר מהsum את הדרגות של תת העץ הימני ומקאונטר את מספר העובדים בתת העץ הימני ונקרא לפונקציה עם תת העץ הימני.

AverageBumpEmpsHigher: הפונקציה מקבלת צומת של עובד (מתוך עץ דרגות ממוין לפי שכר), מצביע לקאונטר, מצביע למשתנה של סכום, וערך higher. הפונקציה בצורה רקורסיבית תבדוק האם המשכורת של העובד הנוכחי קטנה מ-higher, אם כן היא תמשיך לפנות ימינה. אם לא, אז היא תסכום ל-sum את הדרגות של העובדים בתת העץ הנוכחי ותסכום לקאונטר את מספר העובדים בתת העץ הנוכחי. כעת, עבור מקרה זה בלבד, נבדוק האם יש תת עץ שמאלי, אם כן, נחסר מהsum את הדרגות של תת העץ השמאלי ומקאונטר את מספר העובדים בתת העץ השמאלי ונקרא לפונקציה עם תת העץ השמאלי.

סיבוכיות זמן: סיבוכיות הזמן של שתי הפעולות זהה (כי הן סימטריות) ונשים לב שאנו עוברים על מספר צמתים כגובה העץ במקרה הגרוע, ולכן סיבוכיות הזמן הינה $O(log\ n)$.

StatusType companyValue(void *DS, int CompanyID):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים,

בכל שלב של כשלון, נחזיר את הערך המתאים, אם אין כשלון נחזיר SUCCESS.

ניגש ב- $O(1)$ לחברה ב-UF, נאתחל מונה אשר סוכם בכל אחד מהשלבים הבאים את ערך value של Unode המייצג את החברה CompanyID (זאת כפי שראינו בתרגול, תחת השם "שיטת הארגזים"). נסכום כל value במסלול של העץ ההפוך אל עבר השורש בעזרת הפוינטר ל- Parent עד שנגיע לשורש העץ. הסכום במונה ייצג את ערך החברה (עובד על פי שיטת הארגזים שלמדנו בתרגול). כל value מייצג חלק מהערך שווי החברה, והמונה מייצג את ערך השווי האמיתי של החברה.

סיבוכיות זמן:

בפונקציית AcquireCompany אשר משתמשת בפונקציה union, מתבצעת הפעולה find לחברה הרוכשת ולחברה הנרכשת, משמע מתבצע כיווץ מסלולים בכל Union. עפ"י משפט שלמדנו בהרצאה ובתרגול : אם בעץ בעל k איברים, נשתמש באיחוד לפי גודל ובכיווץ מסלולים , סיבוכיות הזמן של m פעולות find,union הינה: $O(mlog^*k)$, כלומר בסיבוכיות משוערכת כל פעולה חסומה ע"י: $O(\log^* k)$. נשים לב שבפעולה זו אנו מבצעים עדכון משתנים ב-*O(1)*, ולאחר מכן עולים בעץ ההפוך מעלה (זהה לפעולת *find*) ולכן סך הכל סיבוכיות הזמן של הפעולה תהיה *O(log*k)* משוערך.

void Quit(void **DS):

ראשית נבדוק האם התנאים הנדרשים בתרגיל מתקיימים, ולאחר מכן נמחוק את המערכת. מחיקת המערכת תקרא לd'tor של ה-UF עבור החברות, לאחר מכן של ה-HT עבור העובדים, לאחר מכן של עץ הדרגות עבור ה-salarySortedEmp ועבור כל אחד מהם ימחקו השדות הפנימיים באמצעות d'tors של העצמים אשר הגדרנו.

סיבוכיות זמן:

הפעולה תמחוק את כל הצמתים מהעצי דרגות, מה-HT ומה-UF, סה"כ ישנם n עובדים ו-k חברות במערכת ולכן נקבל סיבוכיות זמן כוללת של: $O(n+k)$.

סיבוכיות מקום של התוכנית: נשים לב שסיבוכיות המקום של המבנה הינה $O(n+k)$ וזאת בעקבות עצי הדרגות, ה-HTs וה-UF שהגדרנו. במהלך התוכנית, הגדרנו לפעמים מערכי עזר, לכל היותר יתפסו O(n) מקום (מערכים אלו מוגדרים עבור העובדים בפונקציה inOrder לדוגמה ו-mergeTree כפי שציינו). כעת נסתכל על הפונקציות הרקורסיביות אשר הגדרנו: נשים לב שכפי שציינו עבור כל פונקציה מהסוג הנ"ל, אנו לא נתפוס יותר מ- $O(n)$ מקום במחסנית וזה רק עבור הפונקציה inOrder אשר עוברת כל העובדים במקרה הגרוע. פונקציות רקורסיביות מסוג אחר יתפסו כ- $O(log\ n)$ מקום במחסנית. בפעולות אשר מוגדרות עבור התרגיל, אנו משתמשים במבנים ובפונקציות אשר הגדרנו, וכפי שאמרנו למעלה, לא נחרוג בהם מסיבוכיות המקום הנדרשת ולכן בפרט לא נחרוג בפעולות אשר מוגדרות עבור התרגיל. לכן סך כל סיבוכיות המקום בתרגיל הינה $O(n+k)$.