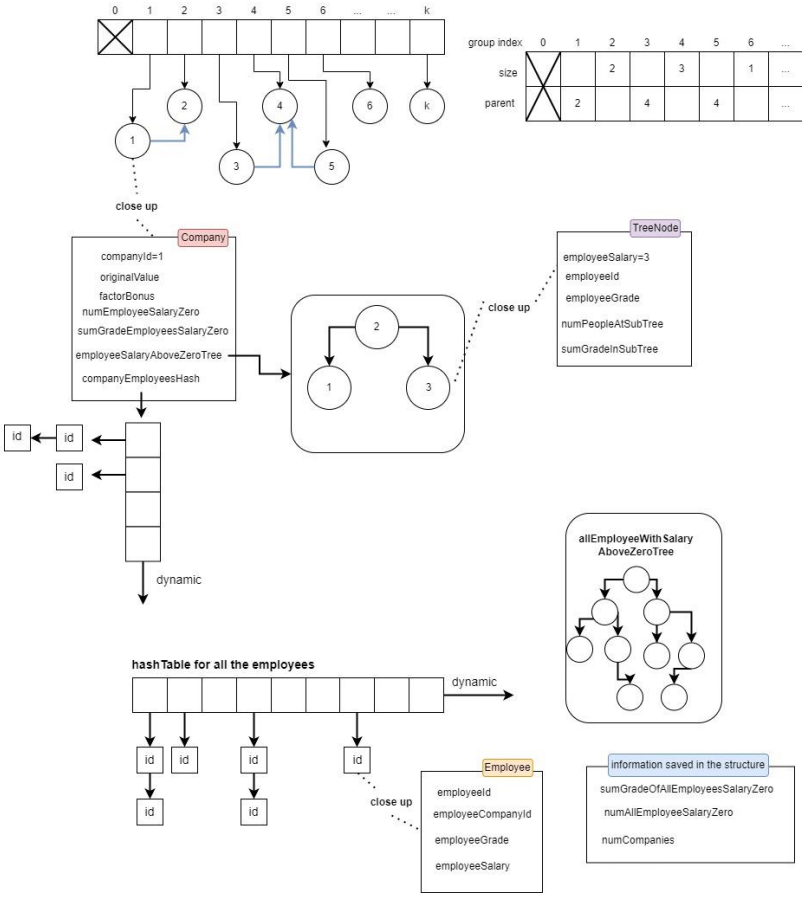
מבני נתונים- תרגיל רטוב שני- חלק יבש:

מגישות: אסתר (אתי) רווח -318477387, שילת קלינשטיין-207258120

תיאור הפתרון:



המבנה שלנו ישתמש בטיפוסי הנתונים הבאים:

1. ***Company***- טיפוס שיכיל: $companyId$ (את ה-id של הקבוצה), $originalValue$ (הערך ההתחלתי של החברה), $factorBonus$ (משתנה עזר שישמור תוספת לערך החברה, בו נשתמש בתהליך חישוב הערך הכולל של החברה, שלמשל יכול לעלות אחרי רכישת חברה- יפורט בהמשך), $numEmployeeSalaryZero$ (כמות העובדים בחברה שבעלי שכר 0), $sumGradeEmployeesSalaryZero$ (סכום הדרגות של עובדי החברה ששכרם 0), $employeeSalaryAboveZeroTree$ (עץ דרגות לעובדי החברה עם שכר גבוה מ-0), $companyEmployeeHash$ (טבלת ערבול בה נשמרים כל עובדי החברה)
2. ***TreeNode***- טיפוס בעץ הדרגות שיכיל: $employeeSalary$ (שכר העובד שה-$Node$ מייצג), $employeeId$ (הid של העובד), $employeeGrade$ (דרגת העובד), $numEmployeeAtSubTree$ (כמות אנשים בתת העץ-כולל הצומת עצמה), $sumGradeSubTree$ (סכום הדרגות בתת העץ- כולל הצומת עצמה)
   * בנוסף בכל צומת נשמור מצביע לאב, גובה ומצביעים לבנים הימני והשמאלי של הצומת
3. ***Employee***- טיפוס שיכיל: $employeeId$ (הid של העובד), $employeeCompanyId$ (החברה בה הוא עובד- מעודכן להיות הערך של החברה האחרונה שרכשה אותו בהתאם לפעולת האיחוד ב-UF כלומר שורש העץ

ההפוך של האיחוד האחרון, או החברה ההתחלתית במידה והיא לא נרכשה₩ רכשה חברה אחרת),
$employeeGrade$ (דרגת העובד), $employeeSalary$ (שכר העובד)

המבנה יכלול:

1. k חברות כאשר כל חברה מכילה שדות כמתואר בטיפוס $Company$
2. מבנה $union - Find$. המבנה יורכב ממערך בגודל $k + 1$ בו כל איבר מצביע לחברה המתאימה לאינדקס שלו (מקום 0 לא יצביע לכלום כי החברות מתחילות מ-1 עד k), ומעץ הפוך שבא לידי ביטוי במערך דו מימדי בגודל 2 שורות על $k + 1$ עמודות כפי שתואר בהרצאה (ימומש על ידי 2 מערכים- מערך 1 לשמירת $parent$ ומערך אחר לשמירת $size$ כפי שמוצג בתמונה למעלה)
3. עץ AVL שהוא עץ דרגות , אשר יכלול את כל עובדי המערכת ששכרם גבוה מ-0. העץ ימוין ראשית לפי שכר ושנית (במידה ויש עובדים עם שכר זהה) לפי id, כאשר כל איבר הוא מהצורה $TreeNode$ שתואר קודם
4. $HashTable$ (מטיפוס שרשראות) לכל עובדי המערכת, שבא לידי ביטוי במערך דינמי שכל איבר בו מצביע לרשימה מטיפוס $Employee$ שתואר קודם לכן. פונקציית הערבול תהיה: $index\ mod(size\_of\_arr)$ שמתאימה כדי ליצור פקטור עומס של 1 כפי שראינו בתרגול ובהרצאה.
5. המבנה ישמור את כמות החברות, ו2 משתנים: $numAllEmployeesSalaryZero$ (כמות האנשים מכל העובדים שיש להם שכר 0) ו-$sumGradeAllEmployeeSalaryZero$ (סכום דרגות העובדים עם שכר 0 מבין כל עובדי המערכת)

בנוסף נשתמש בעץ מת"ב רטוב 1, נהפוך אותו לכך שלא יהיה גנרי ונוסיף שדות לכמות אנשים בתת העץ וסכום דרגותיהם. ההכנסה וההוצאה מהעץ נעשית בצורה רקורסיבית ולכן נעדכן את השדות ואת באיברי העץ בצורה רקורסיבית, נגיע לאיבר ברמה התחתונה, ונעדכן מלמטה עד השורש. כאשר נצטרך לעשות גלגול נעדכן את המידע של הצומת ואז במקרה הצורך נעדכן גם את אביו. למשל בגלגול $LeftLeft$ נעדכן את המידע ב-$TreeNode$ עבור כמות האנשים וסכום דרגותיהם בתת העץ (כלומר נתקן שדות שהמידע שלהם תלוי בתת העץ שלהם) כך:

$$node.numPeopleAtSubTree = node \rightarrow right.numPeopleAtSubTree$$
$$+node \rightarrow left.numPeopleAtSubTree + 1$$
$$node.sumGradesAtSubTree = node \rightarrow right.sumGradesAtSubTree$$
$$+node \rightarrow left.sumGradesAtSubTree + node.employeeGrade$$

כאשר אם אין בן ימני או שמאלי יחובר 0 .

עבור כל עדכון של שדה, אם $node \rightarrow father! = nullptr$ נחזור על העדכונים גם עבורו (וכך נסיים את הגלגול עם מידע מעודכן). יש גישה למידע במצביעים ולכן גלגול עולה $O(1)$

תוספת למבנה $union - find$: נממש מבנה $union - find$ כפי שראינו בהרצאה, עם השינוי הבא: פונקציית איחוד הקבוצות תקבל 2 מזהים לקבוצה ועוד משתנה פקטור שמקבלים בפונקציית איחוד החברות. בכל שורש נשמור 2 שדות שישמשו כ"רשומה" (בהתחלה לכל חברה יש רשומה) וישמרו את $companyParentId$ (מספר החברה שרכשה את כל החברות שבעץ ההפוך- בהתחלה בכל חברה זה מספר החברה עצמה) ו-$companyParentBonus$-שדה שיעזור לחשב את הערך הכולל של $companyParentId$, מאותחל ב-0.

אם גודל החברה $acquire$ הרוכשת קטן₩שווה לגודל החברה $target$ הנרכשת, נאחד את החברה הרוכשת לחברה הנרכשת. לשם כך, ראשית נחשב את הערך שנרצה להוסיף לחברה הרוכשת:

$$extra = (target \rightarrow companyParentBonus + target \rightarrow companyParentId) * factor$$

נעדכן: $acquire \rightarrow factorBonus = acquire \rightarrow factorBonus + extra - target \rightarrow factorBonus$

(הפחתנו בסוף $target \rightarrow factorBonus$ כי זה מידע שנסכום כשנעלה לשורש בפונקציית חישוב ערך החברה וככה נימנע מספירה כפולה) וכעת נעדכן את הרשומה שבשורש:

$$target \rightarrow companyParentId = acquire \rightarrow companyParentId$$
$$target \rightarrow companyParentBonus = acquire \rightarrow companyParentBonus + extra$$

אחרת, גודל החברה הרוכשת גדול מגודל החברה הנרכשת, לכן נאחד את $target$ ל$acquire$. לשם כך נחשב:

$$extra = extra = (target \rightarrow companyParentBonus + target \rightarrow companyParentId) * factor$$

נעדכן את הרשומה: $acquire \rightarrow companyParentBonus += extra$

בנוסף נעדכן: $acquire \rightarrow factorBonus += extra$

$target \rightarrow factorBonus -= acquire \rightarrow factorBonus$ (הפחתנו את הערך שבשורש כדי שנעלה מ-$target$ ל-$acquire$ בפונקציית חישוב ערך של חברה, לא נספור מידע שלא שייך ל-$target$)

במהלך כיווץ המסלולים נבצע את שראינו בהרצאה, עם התוספת הבאה:

1. במהלך המעבר הראשון מהצומת עד לשורש העץ ההפוך, נשמור במשתנה sum את סכומי ה$factorBonus$ של החברות במסלול עד השורש (לא כולל השורש!)

2. נחזור על המסלול ונחזיק משתנה $sub = 0$ שישמור את סכומי ה-$factorBonus$ עד לצומת הנוכחי (לא כולל!), כשנגיע לצומת שיש לחבר לשורש נבצע $sum - sub$ ואת ערך זה נשים ב-$factorBonus$ של הצומת שנחבר לשורש, לפני זה נשמור את הערך הישן כדי שאחרי החיבור נעדכן: $sub+= v.oldFactorbonus$, וכך נמשיך במעלה המסלול.

שינוי זה נועד כדי שנוכל לבצע את $companyValue$ בכך שנגיע לחברה בעזרת המצביע ממערך הקבוצות שבתוך מבנה ה-$u - f$, נסכום את הסכום ההתחלתי שלה וכל ערך בונוס ($factorBonus$) עד לשורש העץ בו נמצאת החברה.

הסבר לפעולות:

הערה: עבור הפונקציות $addEmployee, companyValue, acquireCompany,$ $sumOfBumpGradeBetweenTopWorkersByGroup$, $averageBumpGradeBtweenSalaryByGroup$ בסוף כל אחת מהן כתובה הסיבוכיות המשוערכת ביחד עם פונקציות אלו, זו כיוון שבהינתן m פעולות מפונקציות אלו, ייתכן שבכולן נצטרך למצוא חברה בעזרת k פעולות $union - Find$, נראה שבהינתן שכל פעולה בפונקציה חסומה ב- $O(x)$ כלשהו (למשל ספירת כמות איברים בין 2 תחומי שכר בעץ דרגות במהלך הפונקציה $averageBumpGradeBtweenSalaryByGroup$ חסומה על ידי $O(logn)$), אז בהינתן m פעולות כאשר הפעולה **היקרה ביותר** על העובדים היא $O(x)$, יעלו $O(m \cdot x + m \cdot \log^* k)$ ולכן באופן משוערך $O(x + \log^* x)$ לפעולה (כשעושים פעולות על ה-$hashTable$ הפעולות הן בממוצע על הקלט ולכן הסיבוכיות תהיה משוערכת בממוצע של הקלט)

למשל, כשעושים m פעולות ואחת מהן כוללת את $acquireCompany$, עוד נראה שפעולה יקרה באיחוד ה-$hashTable$ של העובדים עולה $O(n)$ בממוצע על הקלט, ולכן בהינתן m פעולות כמו הוספת עובד, חישוב סכום בטווח מסוים וכו' מקבוצת הפונקציות המתוארת קודם, נקבל שהסיבוכיות היא $O(\log^* k + n)$ משוערכת בממוצע על הקלט כי הפעולה היקרה ביותר מתרחשת באיחוד הקבוצות והיא החסם שלנו. מפאת חוסר מקום פירטנו את הרעיון הכללי ולא תחת לכל פונקציה אבל אופן החישוב זהה.

עבור $removeEmployee$ שמשוערכת עם $addEmployee$. בהינתן n הכנסות של אנשים למבנה, הסרה תעלה במקרה הגרוע $O(logn)$ בממוצע על הקלט (כי יש להוציא גם מ$hashTable$)

Void* init(int k)- אם $k \leq 0$ נחזיר $null$, אחרת נבצע:

1. אתחול עץ דרגות ריק $allEmployeeWithSalaryAboveZeroTree$ ב-$O(1)$
2. שמירת מספר הקבוצות (k) במשתנה $numOfCompanies$, ואתחול השדות $numAllEmployeesSalaryZero$ ו- $sumGradeAllEmployeeSalaryZero$ שבמבנה ב-0 ב - $O(1)$
3. אתחול $hashTable$ לכל העובדים, תחילה על ידי מערך בגודל קבוע בו כל תא ריק-$O(1)$
4. אתחול מבנה ה-$union - find$ כפי שראינו בהרצאה. מקצים מספר סופי של מערכים שתלויים במספר החברות ב- $O(k)$, במהלך יצירת ה-$u - f$ נאתחל k קבוצות ריקות, כל אחת ב-$O(1)$, לכן סה"כ $O(k)$ (כל קבוצה מכילה באתחול משתנה לספירת כמות האנשים שבחברה עם שכר 0 ומשתנה לסכום דרגותיהם, שניהם מאותחלים ב-0, טבלת ערבול ריקה, עץ דרגות ריק, את ה-id שלה והערך ההתחלתי שלה (שזהה ל-id) ו$factorBonus = 0$, סה"כ $O(1)$)

אם הקצאות המקום הצליחו נחזיר מצביע למבנה הנתונים, אחרת נהרוס את המבני החלקי ונחזיר $null$. סה"כ הסיבוכיות של הפונקציה היא $O(k)$ כנדרש

הערה: במהלך כל אחת מהפונקציות הבאות, לאחר בדיקת התקינות (שמתרחשות תמיד ב--$O(1)$), אם במהלך הפונקציה יש בעיה בהקצאת הזיכרון נחזיר $ALOCATION\_ERROR$, אחרת (במידה ואין שגיאת $FAILURE$) נחזיר בסופה $SUCCESS$

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

אם$DS == NULL$ או $EmployeeID \leq 0$ או $CompanyID \leq 0$ או $CompamyID > k$ או $Grade < 0$ נחזיר $INVALID\_INPUT$, אחרת, נבדוק ב-$hashTable$ של כל העובדים האם העובד קיים כבר במערכת ב-$O(1)$ בממוצע על הקלט, ואם הוא קיים נחזיר $FAILURE$. אחרת, נמצא את חברת העובד ב-$O(\log^* k)$ משוערך כפי שראינו בהרצאה, ונוסיף את העובד ל-$hashTable$ של כל העובדים ב-$O(1)$ משוערך בממוצע על הקלט (כאשר ב-$conpamyId$ של העובד שמור ה-id של החברה שקיבלנו מפעולת ה-$find$ אשר אינה בהכרח זהה לפרמטר מהפונקציה), ונעדכן: $numAllEmployeesSalaryZero++$, $sumGradeAllEmployeeSalaryZero+= Grade$ ב-$O(1)$ (לא מכניסים אותו לעץ כי שכרו 0.

כעת,נוסיף את העובד ל-$hashTable$ של החברה שלו ב-$O(1)$ משוערך בממוצע על הקלט ונעדכן בחברה את השדות: $numEmployeeSalaryZero++$, $sumGradeEmployeesSalaryZero+= Grade$

לכן סה"כ סיבוכיות הפונקציה היא $O(\log^* k)$ משוערך בממוצע על הקלט

StatusType removeEmployee(void *DS, int EmployeeID)- אם $DS == NULL$ או $EmployeeID \leq 0$ נחזיר $INVALID\_INPUT$. נבדוק האם העובד קיים ב-$hashTable$ של כל העובדים במערכת ב-$O(1)$ בממוצע על הקלט, ואם הוא לא קיים נחזיר $FAILURE$. אחרת, נסיר את העובד מה-$hashTable$ של כל העובדים במבנה ב-$O(1)$ משוערך בממוצע על הקלט, לפני זה נשמור את ה-$companyId, salary, grade$ שלו ב-$O(1)$. אם השכר של העובד היה 0 נעדכן: $numAllEmployeesSalaryZero - -$,
$sumGradeAllEmployeeSalaryZero -= grade$ ב-$O(1)$
אחרת השכר של העובד היה גדול מ-0 לכן נסיר אותו מהעץ $allEmployeeWithSalaryAboveZeroTree$ ב-$O(logn)$. נשים לב שה-$companyId$ ששמור במידע של העובד זה הid של שורש העץ ההפוך בו החברה המקורית שלו נמצאת (מאופן הכנסת העובד ומאופן פעולת רכישת החברות שנתאר בהמשך, נדאג לשמור על מידע זה מעודכן ונכון), כלומר פעולת $find$ ב-$union - find$ תעלה $O(1)$. לפיכך, נמצא את חברת העובד ב-$O(1)$. אם שכר העובד היה גדול מ-0 נסיר אותו מהעץ של עובדי החברה עם שכר גדול מ-0 ב-$O(logn)$, אחרת נעדכן שכמות האנשים בחברה עם שכר 0 קטנה ב-1 וסכום דרגותיהם קטן ב-$grade$ ב-$O(1)$
סה"כ סיבוכיות הפונקציה היא $O(logn)$ משוערך בממוצע על הקלט

StatusType acquireCompany(void *DS, int AcquirerID, int TargetID, double Factor)- אם $DS == NULL$ או $AcquireID \leq 0$ או $AcquireID > k$ או $TargetID \leq 0$ או $TargetID > k$ או $TargetID ==$ $AcquireID$ או $Factor \leq 0.0$ נחזיר $INVALID\_INPUT$-$O(1)$. אם פעולת ה-$find$ על 2 החברות מחזירה את אותה החברה נחזיר $INVALID\_INPUT$ (אחת מהן רכשה אחרת או שתיהן נרכשו בעבר על ידי חברה אחרת, כלומר הן אותה החברה). -$O(\log^* k)$ משוערך. אם הקלט תקין נבצע:

1. נמצא את 2 החברות ונאחד לפי גודל בסיבוכיות $O(\log^* k)$ משוערך (תוך עדכון השדות $factorBonus$ כמתואר קודם), נוסיף במשתנה $numEmployeeSalaryZero$ של החברה אליה איחדנו את הערך ששמור בחברה שאוחדה אליה, ונאפס את הכמות בחברה שאוחדה לאחרת. וכן במשתנה $sumGradeEmployeesSalaryZero$בחברה אליה אוחדו החברות נוסיף את הערך ששמור במשתנה בחברה שאוחדה לאחרת, ונאפס אצל זאת שאוחדה את הערך ל-0 - $O(1)$
2. עבור עצי העובדים של 2 החברות, נרצה לאחד את העצים כך שהעובדים בהם יהיו בחברה אליה איחדנו לפי גודל, לכן נבצע:

2.1. נקרא לפונקציה שהכנו בעץ שיוצרת עץ חדש מ-2 עצים קיימים בעזרת איטרטור שהגדרנו בעץ שפועל לפי סיור $inorder$, ונכניס את איברי 2 העצים למערך ממוין (נוצר עותק) (נשתמש ב-2 איטרטורים ותנאים להשוואות ונכניס לפי סדר את איברי שני העצים מהקטן לגדול, כאשר במקרה של שוויון בשכר יוכנס קודם עובד עם id קטן יותר), זה יעלה $O(n_{AcquireID} + n_{TargetID})$

2.2. נרוקן את עצי 2 החברות ב-$O(n_{AcquireID} + n_{TargetID})$

2.3. נקרא לפונקציה שממלאת עץ לפי מערך וגודלו, וכך נמלא מחדש את העץ של החברה שגודלו גדול יותר- נעשה זאת בעזרת האלגוריתם שראינו בת"ב רטוב 1 שהיה כך:

- קבע: השורש יהיה האיבר האמצעי במערך
- באופן רקורסיבי תעשה לבן הימני והשמאלי:

1. השג את האמצע של חצי המערך השמאלי והפוך אותו לבן השמאלי של השורש שנקבע בפעולה הקודמת
2. השג את האמצע של חצי המערך הימני והפוך אותו לבן ימני של השורש שנקבע בפעולה הקודמת -תהיה חזרה לשלב בו נקבע בן כ"שורש" בתת עץ

ניתן לתאר את הסיבוכיות באופן הבא: $T(n) = 2T\left(\frac{n}{2}\right) + C$ כאשר $T(n)$ הזמן למילוי עץ על ידי **מערך ממוין** בגודל n ו-C קבוע (הזמן שלוקח למציאת האמצע במערך וכדומה). משיטת המאסטר לפי השיטה הראשונה ועבור בחירת $\epsilon = 1 > 0$ נקבל שסיבוכיות הפעולה היא $O(n)$ (c קבוע ולכן $f(n) = O(1)$)
הערה: במהלך הפונקציה יתכן שיהיו גלגולים וכן יהיו עדכוני שדות בצמתי העץ, אנחנו מכניסים ומעדכנים את השדות שתלויים בתת העץ רקורסיבית ולכן עומדים בסיבוכיות. וכמובן שנשחרר העתקים שיצרנו במהלך הפונקציה.

3. נאחד בין טבלאות הערבול של 2 החברות באופן הבא: נעבור על המערך הדינמי שהוא ה-$hashTable$ של הקבוצה הקטנה, בכל תא יש רשימה של העובדים שנמצאים בתא הנוכחי (אם קיימים), נעבור על איברי הרשימה ועבור כל עובד ברשימה, נבצע פעולת הכנסה אל ה-$hashTable$ של הקבוצה הגדולה (נמצא בעזרת פונקציית הערבול את התא המתאים לעובד במערך, ונוסיף ל**ראש הרשימה** שכרגע יש בו). נשים לב שייתכן שחלק מפעולות ההכנסה יגרמו להגדלת המערך הדינמי של החברה הגדולה. לבסוף נרוקן את ה-$hashTable$ של

החברה הקטנה. כשנסיים נעבור על ה-$hashTable$ של החברה הגדולה ונעדכן את ה-$companyID$ של כל העובדים בה לזה של החברה הגדולה. אם נסמן ב-n_1 את כמות העובדים בחברה הקטנה וב-n_2 את כמות העובדים בחברה הגדולה, נקבל שהמעבר על המערך הדינמי של החברה הקטנה עולה $O(n_1)$ כיוון שגודל המערך הוא גם $O(n_1)$ ויש n_1 אנשים לעבור עליהם ברשימות, פעולת ההכנסה אל ה-$hashTable$ של החברה השנייה עולה עבור כל עובד שמוכנס $O(1)$ משוערך בממוצע על הקלט, לכן סה"כ $O(n_1)$ משוערך בממוצע על הקלט. ריקון ה-$hashTable$ של החברה הקטנה עולה $O(n_1)$. המערך החדש של הקבוצה הגדולה הוא בסדר גודל $O(n_1 + n_2) = O(n_{AcquireID} + n_{TargetID})$ ולכן הסיבוכיות לעדכון ה-$companyID$ של כל העובדים היא $O(n_{AcquireID} + n_{TargetID})$ (בעזרת שימוש במצביעים חכמים לא נצטרך לעדכן גם ב-$hashTable$ של כלל העובדים, כי 2 הטבלאות יצביעו לאותו העובד), לכן סה"כ הסיבוכיות היא $O(n_{AcquireID} + n_{TargetID})$ משוערך בממוצע על הקלט.

לכן סה"כ סיבוכיות הפונקציה היא $O(\log^* k + n_{AcquireID} + n_{TargetID})$

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

אם $DS == NULL$ או $EmployeeID \leq 0$ או $SalaryIncrease \leq 0$ נחזיר $INVALID\_INPUT$.
נבדוק האם העובד קיים ב-$hashTable$ של כל העובדים ב-$O(1)$ בממוצע על הקלט, אם הוא לא קיים נחזיר $FAILURE$. אחרת מצאנו את העובד. נשמור את השכר, הדרגה וה-$companyID$ שלו במשתנים זמניים ב-$O(1)$. נעדכן את השכר של העובד ב-$hashTable$ ונוסיף לו את $SalaryIncrease$ (זה מעדכן גם בטבלת הערבול של החברה שלו לאור השימוש במצביעים חכמים)-$O(1)$. נמצא את חברת העובד ב-$O(1)$ (בגלל שה-$companyID$ ששמור במידע של העובד הוא ה-id של החברה שבשורש העץ ההפוך שבו נמצאת חברתו המקורית, נמצא את החברה בה הוא שמור כעת , שהיא שורש העץ ההפוך, ב-$O(1)$ בפעולת ה-$find$). אם השכר הישן של העובד היה 0 נעדכן:
$numAllEmployeesSalaryZero - -$, $sumGradeAllEmployeeSalaryZero -= employeeGrade$, וכן במשתנים שבחברה שלו: $numEmployeeSalaryZero - -$, $sumGradeEmployeesSalaryZero -=$ $employeeGrade$ ב-$O(1)$. אחרת, השכר הישן של העובד גדול מ-0, לכן נמצא אותו לפי ה-$(Oldsalary, id)$ שלו בעץ של החברה ובעץ של כלל העובדים, ונסיר אותו מהם ב-$O(logn)$
לבסוף נוסיף את העובד מחדש לעצים עם $(newSalary, id, employeeGrade)$ ב-$O(logn)$ כאשר $newSalary = OldSalary + SalaryIncrease$.
לכן סה"כ סיבוכיות הפונקציה היא $O(logn)$ בממוצע על הקלט.

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

אם $DS == NULL$ או $EmployeeID \leq 0$ נחזיר $INVALID\_INPUT$. נבדוק האם העובד קיים ב-$hashTable$ של כל העובדים ב-$O(1)$ בממוצע על הקלט, ואם הוא לא קיים נחזיר $FAILURE$.
נבדוק: אם $BumpGrade \leq 0$ לא נבצע דבר ונחזיר $SUCCESS$
אחרת, ניגש למידע של העובד שקיבלנו מה-$hashTable$ ונעלה לו את הדרגה ב-$BumpGrade$ ב-$O(1)$ (זה מעדכן גם בטבלת הערבול שבחברה שלו בזכות מצביעים חכמים), לפני זה נשמור את הדרגה הישנה שלו, את השכר ואת ה-$companyID$ שלו-$O(1)$. נמצא את החברה של העובד ב-$O(1)$ (כפי שנאמר קודם, ב-$companyID$ של העובד שמור ה-id של החברה שנמצאת בשורש העץ ההפוך שבו החברה המקורית שלו נמצאת, לכן פעולת ה-$find$ מתרחשת ב-$O(1)$). אם השכר של העובד 0, נעדכן במשתנה שבמבנה:

$$sumGradeAllEmployeeSalaryZero += BumpGrade$$

וכן במשתנה שבחברה שלו: $sumGradeEmployeesSalaryZero += BumpGrade$ ב-$O(1)$
אחרת, השכר שונה מ-0, לכן נמצא אותו לפי השכר וה-id שלו ונסיר את העובד מעץ העובדים של כלל העובדים ומעץ העובדים של החברה שלו ב-$O(logn)$ ואז נכניס אותו מחדש לעצים עם הדרגה החדשה שלו ב-$O(logn)$
לכן סה"כ סיבוכיות הפונקציה היא $O(logn)$ בממוצע על הקלט

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

-אם $DS == NULL$ או $CompanyID < 0$ או $CompanyID > k$ או $m \leq 0$ נחזיר $INVALID\_INPUT$. אם $CompanyID == 0$ נבצע את הפעולה הבאה על העץ $allEmployeeWithSalaryAboveZeroTree$ ואם $CompanyID > 0$ נמצא את החברה שמתאימה ל$CompanyID$ ב-$O(\log^* k)$ משוערך, ונפעיל את הפעולה על העץ שבה. השלבים עצמם זהים:

ניגש לעץ ב-$O(1)$ נמצא את כמות האנשים בו (ניגש לשורש ונסתכל על הערך ב-$numPeopleAtSubTree$ ששמור בו- זהו עץ דרגות ששומר כמות צמתים=כמות אנשים בתת העץ), אם מספר זה קטן מ-m נחזיר $FAILURE$. אחרת, נקרא לפונקציה עזר שתפעל על העץ לצורך הסכימה. פונקציה זו תקבל סכום התחלתי sum (בהתחלה 0), כמות אנשים שיש לסכום num (בהתחלה m), ואיבר ממנו נתחיל את הסכימה $node$ (בהתחלה השורש של העץ). האלגוריתם יפעל כך (בהמשך ניתוח הסיבוכיות שלו):

- בהינתן ה-$node$ שהתקבל בפונקציה, נבדוק האם יש לו בן ימני,
- אם יש בן ימני, נבדוק האם כמות האנשים בתת העץ ששורשו הבן הימני גדולה או שווה לnum (לפי שדה ששמור בצומת של כמות האנשים בתת העץ),
  - אם כן, נחזיר את הערך מהקריאה לפונקציה עם $sum, num, node \rightarrow right$
  - אם לא, נוסיף ל-sum את $node \rightarrow right.sumGradeAtSubTree$ ונעדכן $num -= node \rightarrow right.numPeopleAtSubTree$. כעת נבדוק האם $num == 1$, אם כן נוסיף לסכום את הדרגה ששמורה ב-$node$ ונחזיר את הסכום שהתקבל. אחרת נעדכן $sum += node.employeeGrade$, $num -= 1$ ונחזיר את הערך שהפונקציה תחזיר לאחר קריאה לה עם הפרמטרים $node \rightarrow left$ $\underbrace{sum, num,}_{the\ updated\ values}$. נשים לב שבהכרח קיים בן שמאלי לצומת, כי אנחנו יודעים שכמות האנשים בתת העץ גדולה₩שווה ל-m והכמות בתת העץ הימני ובצומת עצמו קטנה מכמות זו.
- אחרת, לא קיים בן ימני. נבדוק האם $num == 1$, אם כן נוסיף ל-sum את הדרגה ששמורה בצומת ונחזיר את הערך, אחרת נעדכן $sum += node.employeeGrade$ וכן $num -= 1$, ונחזיר את הערך שיתקבל בקריאה לפונקציה עם $node \rightarrow left$, $\underbrace{sum, num,}_{the\ updated\ values}$. גם פה ידוע שיש בן שמאלי אחרת בבדיקה בהתחלה הייתה חוזרת שגיאה, כי לא היו מספיק אנשים לסכימה מראש.

לבסוף נדפיס את התוצאה שקיבלנו.

<u>נכונות:</u> בסכימה אנחנו ראשית סוכמים את האיברים הימניים ביותר בעץ, ורק כשהכמות לא מספיקה עוברים לסכום מצד שמאל, לכן אנחנו ניתקל באנשים עם שכר גבוה יותר לפני שנתקבל באנשים עם שכר נמוך יותר, וכך במקרה של שווין בשכר נתקל קודם באנשים עם id גבוה יותר, לכן הפונקציה תבחר לסכום את הדרגות של האנשים עם המשכורות הכי גבוהות בחברה₩ בעץ הכללי של העובדים בהתאם למקרה לפי $CompanyID$.

<u>סיבוכיות:</u> במהלך האלגוריתם אנחנו מתקדמים ימינה או שמאלה בהתאם לכמות העובדים בתת העץ, בדומה לחיפוש, ולכן עושים מסלול מהשורש עד לכל היותר עלה. לפיכך אורך המסלול הארוך ביותר הוא $O(logn)$. בגלל השדות של כמות העובדים בתת העץ וסכום דרגות העובדים בתת העץ, אם אנחנו רואים שאי אפשר להשיג את הכמות המבוקשת מללכת ימינה, אנחנו מקבלים מידע על תת העץ הימני, כמות האנשים בו וסכום דרגותיהם ב-$O(1)$, ומתקדמים בחיפוש לצד השני, לכן לא נעבור על יותר מ-$O(logn)$ צמתים, לכן הסיבוכיות הכוללת היא $O(\log^* k + logn)$ משוערך (כאשר המקרה הגרוע זה המקרה בו צריך לחפש חברה)

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

-אם $DS == NULL$ או $lowerSalary < 0$ או $higherSalary < 0$ או $lowerSalary > higherSalary$ או $CompanyID < 0$ או $CompanyID < k$ נחזיר $INVALID\_INPUT$. אם $CompanyID == 0$ נבצע את הפעולה על המבנים של כלל העובדים, אחרת אם $CompanyID > 0$ נמצא את החברה שמתאימה למזהה $CompanyID$ שקיבלנו ב-$O(\log^* k)$ משוערך ונבצע את הפעולה על המבנים שבתוך החברה- הפעולות עצמן זהות:

1. נאתחל 2 משתנים: $num = 0$ (כמות האנשים בתחום) ו-$sum = 0$ (סכום דרגותיהם של האנשים שבטווח) ב-$O(1)$
2. נבדוק האם $lowerSalary == 0$, אם כן נוסיף ל-num את כמות האנשים ששמורה במבנה שהשכר שלהם 0 (לפי ה-$companyID$ נדע האם מדובר במידע של כלל העובדים או במידע ששמור בתוך החברה) ובאופן דומה את סכום דרגותיהם ל-sum, זה מידע שניתן להשיג מהמבנה המתאים ולחשב ב-$O(1)$
3. כעת נחלק למקרים:

3.1. אם $lowerSalary == higherSalary$:

1) ייתכן שיש כמה אנשים עם השכר הזה. נחפש צומת בעץ שבמבנה עם השכר $lowSalary$ ובעלת id מינימלי ששמור בה ב-$O(logn)$ -נבצע חיפוש כמו שראינו בהרצאה כדי למצוא צומת

ראשון עם שכר מתאים, אחרי שמצאנו אותו לא נעצור אלא נשמור מצביע אליו, ונמשיך לחפש בתת העץ השמאלי של הצומת (כי אם קיים צומת עם שכר זהה ו-id יותר קטן הוא יהיה משמאלו, ואם תת העץ השמאלי לא קיים נחזיר את מי שמצאנו), נמשיך שוב בחיפוש כמו שראינו בהרצאה, וכל פעם שנמצא צומת עם אותו שכר ו-id מינימלי נשמור אותו ונמשיך בחיפוש משמאלו, עד שבסוף נגיע לעלה, אנחנו כל פעם יורדים בחיפוש ולכן סה"כ זה יבוצע ב-$O(logn)$

2) אם לא מצאנו צומת כנדרש (הגענו לעלה והמצביע החזיר לנו $null$) סימן שאין אנשים במבנה עם השכר הרצוי, לכן נחזיר $FALSE$

3) אחרת, יש לפחות צומת אחת בעץ עם השכר הרצוי. באופן דומה נמצא צומת עם השכר הרצוי ועם id מקסימלי, נפעל באופן דומה לקודם רק הפעם כשנמצא צומת עם השכר הרצוי נשמור מצביע אליו ונמשיך בחיפוש מימינו, כי אם קיים צומת עם אותו שכר ו-id יותר גבוה הוא יהיה מימין מתכונות העץ.-$O(logn)$

4) אם 2 הצמתים שקיבלנו בחיפוש השכר עם ה-id המינימלי והמקסימלי הם אותה הצומת (השוואה ב-$O(1)$ לפי שדות) , יש רק אדם אחד שעונה על הדרישות, לכן נעדכן $num+=1$ ו-$sum+=node.employeeGrade$ (כאשר $node$ זו הצומת שמצאנו) ב-$O(1)$ ונעבור לשלב 4.

5) אחרת, יש לפחות 2 אנשים במבנה. נסמן את הצמתים שמצאנו כ-min, max ונמצא את האב המשותף של 2 הצמתים ב-$O(logn)$ –נשתמש בפונקציה רקורסיבית בעץ שמקבלת צומת התחלה, ו2 צמתים שעבורם יש למצוא אב משותף. נבצע חיפוש שמתחיל מהשורש ויבוצע כך: אם השכר שבשורש גדול מהשכר בmin וב-max (זה אותו השכר אבל הפונקציה הותאמה גם למקרים בהם השכר שונה), נבדוק האם הבן השמאלי קיים, אם לא- נחזיר את השורש, אחרת נקרא לפונקציה עם $min, max, root \rightarrow left$ ונתחיל חיפוש באותו אופן החל מהבן השמאלי אחרת, באופן דומה, אם השכר שבשורש קטן מהשכר ב-min, max, אם הבן הימני לא קיים נחזיר את השורש ואחרת נקרא לפונקציה עם $min, max, root \rightarrow right$
אחרת, השכר שבשורש זהה לשכר שב-min, max. במקרה זה, נבחר האם להמשיך בחיפוש בעץ הימני או השמאלי (אם קיים) לפי השוואות ה-id: אם ה-id שבשורש קטן מה-id שב-2 הצמתים האחרים נקרא לפונקציה עם $min, max, root \rightarrow right$, ואם ה-$id$ שבשורש גדול מה-id שב-2 הצמתים האחרים נקרא לפונקציה עם $min, max, root \rightarrow left$.
אחרת נחזיר את $root$
לצורך מציאת האב המשותף אנחנו כל הזמן פונים ימינה או שמאלה, כלומר יורדים בעץ. אנחנו מתחילים מהשורש ולכן אורך המסלול המקסימלי הוא כעומק העץ, לפיכך הסיבוכיות היא $O(logn)$

6) אחרי מציאת האב המשותף נסכום את כמות הצמתים בין min, max (כולל) החל מהאב המשותף ב-$O(logn)$- נשתמש בפונקציה רקורסיבית שמקבלת צומת התחלה שנסמנה $start$ (בהתחלה זה יהיה האב המסומן שמצאנו),את min, max ואת sum, num. היא תפעל כך: אם $start = null$ = תחזור, אחרת $num+=1$ ו-$sum+=start.employeeGrade$. עכשיו נבדוק:

- אם הצומת $start$ היא הצומת min, האב המשותף של הצמתים היה min, לכן כל האיברים משמאלו ולא בתחום, וmax נמצא מימינו. לכן נקרא לפונקציה שתסכום את כל הצמתים מימין לצומת $start$ שקטנים מ-max [נתחיל מהבן הימני של $start$, בכל פניה ימינה בחיפוש max מ$node$ כלשהו נעדכן:
$sum+=node \rightarrow left.sumGradesInSubTree + node.employeeGrade$
$num+=node \rightarrow left.numPeopleAtSubTree + 1$ (בחישובים במידה והצומת השמאלי לא קיים, נוסיף עבורו 0), ובכל פניה שמאלה לא נעדכן דבר. לבסוף כשנגיע ל-max, נעדכן: (אם אין לו בן שמאלי נקבל 0 בחלק שלו)
$sum+=max \rightarrow left.sumGradesInSubTree + max.employeeGrade$
$num+=max \rightarrow left.numPeopleInSubTree + 1$] -הסכימה עולה $O(logn)$ כי מתחילים באב המשותף (במקרה הגרוע זה השורש) וכל פעם פונים ימינה או שמאלה, כלומר יורדים ברמה. לכן עושים מסלול שעולה $O(logn)$ סה"כ. בגלל שהעץ עץ דרגות ברגע שצריך לפנות ימינה לצורך חיפוש max, אנחנו יכולים להשיג את הסכום של כל תת העץ השמאלי ב-$O(1)$ לכן כדי לסכום את האיברים שבטווח אין צורך להגיע אליהם בעץ אלא רק לבצע מסלול חיפוש ולסכום מידע לפי שדות ששומרים מידע על תת העץ. לכן הסיבוכיות היא כפי שציינו. כעת נעבור לשלב 4 לחישוב הממוצע

- אם הצומת $start$ היא הצומת max נפעל באופן סימטרי לסעיף הקודם ונעבור לשלב 4 - $O(logn)$
- אחרת בהכרח min נמצא משמאל לצומת $start$ וmax מימינה. ראשית נקרא לפונקציה שתסכום את כל האיברים בתת העץ הימני של הצומת שקטנים מ-max כפי שתיארנו קודם, והיא תעדכן את sum ו-num. אחר כך נסכום את האיברים בתת העץ השמאלי של הצומת שגדולים מ-min, ופונקציה זו תעדכן את sum , num. פעולות אלו יפעלו דומה לפעולות שתיארנו קודם ובסיבוכיות $O(logn)$. בסיום נעבור לשלב 4.

3.2. אם $lowerSalary \neq higherSalary$ נשים לב שמבדיקות התקינות בהכרח מתקיים ש- $lowerSalary < higherSalary$: במקרה זה, נאתחל משתנים $minIsFake = false$, $maxIsFake = false$ ו-$numFalseNodes = 0$. $O(1)$-

א. נחפש צומת עם שכר $lowSalary$ ו-id מינימלי כמו שתיארנו קודם-$O(logn)$, אם לא קיים כזה נוסיף צומת לעץ עם השכר המינימלי ו$id = 0, grade = 0$ ב-$O(logn)$ ונעדכן $minIsFake = true, numFalseNodes++$ - $O(1)$
ב. באופן דומה נחפש צומת עם שכר $highSalary$ ו-id מקסימלי ב-$O(logn)$ ואם לא קיים כזה נוסיף לעץ עם השכר המקסימלי ו$id = 0, grade = 0$ ב-$O(logn)$ ונעדכן $maxIsFake = true, numFalseNodes++$ - $O(1)$
ג. נמצא את האב המשותף של הצמתים שהכנסו ב-$O(logn)$ (כפי שתואר קודם בסעיף 5))
ד. נסכום את סכום הדרגות וכמות האנשים שנמצאים בטווח (באופן זהה למתואר בסעיף 6) קודם)
ה. כעת נסיר את הצמתים שהכנסנו ולא היו קיימים קודם (נדע את מי להסיר לפי המשתנים הבוליאנים) ונעדכן: $num -= numFalseNodes$ (אין צורך לעדכן את הסכום כי הכנסנו צמתים עם דרגה 0 שלא השפיעה על החישוב) , סה"כ -$O(logn)$

4. כשהגענו לשלב זה נותר רק לחשב את הממוצע. ב-sum שמור לנו סכום הדרגות של האנשים שבטווח וב-num את כמות האנשים שבטווח. אם $num == 0$ אין אנשים בטווח לכן נחזיר $FAILURE$, אחרת נחשב $avg = sum\backslash num$ ונדפיס את התוצאה -$O(1)$

סה"כ סיבוכיות הפונקציה היא $O(\log^* k + logn)$ (במקרה הגרוע $CompanyID > 0$ ויש למצוא את החברה)

StatusType companyValue(void *DS, int CompanyID)- אם $DS == NULL$ או $CompanyID \leq 0$ או $CompanyID > k$ נחזיר $INVALID\_INPUT$. אחרת, נקרא לפונקציה שממומשת במבנה ה-$union - find$ אשר תחשב לנו את ערך החברה:

הפונקציה תקבל את $CompanyID$, תיגש אליה ותשמור במשתנה $value$ את הערך $originalValue$ של החברה שה-id שלה הוא ממש $CompanyID$ (לא חברה שרכשה אותה וכדומה) ב-$O(1)$ בעזרת מערך הקבוצות שב-$union - find$. לאחר מכן היא תעלה מהחברה עד לשורש העץ ההפוך שבו היא נמצאת ב-$O(\log^* k)$ משוערך ותוסיף לסכום $value$ את כל ערכי ה-$factorBonus$ ששמורים בחברות לאורך המסלול מהצומת לשורש (בכל חברה גישה לשדה זה ב-$O(1)$), כולל הערך של השורש עצמו. לבסוף הפונקציה תדפיס את התוצאה שחושבה-$O(1)$.

סה"כ הסיבוכיות היא $O(\log^* k)$

void Quit(void **DS)- נהרוס את המבנים שיצרנו: כדי להרוס חברה נהרוס את העץ וה- $hashTable$ שבה בסיבוכיות $O(n_x)$ כאשר n_x זה כמות האנשים בחברה. נשים לב שחברה מכילה כמות מסוימת של עובדים, כך שכל החברות יחד מכילות $O(n)$ אנשים, לכן סה"כ סיבוכיות הריסת כל החברות היא $O(k + n)$. בנוסף נהרוס את העץ של כלל העובדים במערכת עם שכר מעל 0 ואת ה-$hashTable$ של כל העובדים ב-$O(n)$ ככמות העובדים, לכן סך סיבוכיות הפונקציה היא $O(k + n)$

סיבוכיות מקום: נשים לב שבכל הפונקציות הנדרשות לא עברנו את סיבוכיות המקום הנדרשת ממבני הנתונים ולכן בסה"כ מבנה הנתונים והפעולות המוגדרות עבורו עומדות בסיבוכיות המקום הנדרשת מאיתנו שהינה $O(n + k)$ כאשר k זה כמות החברות ו-n זה כמות העובדים בכל המערכת.