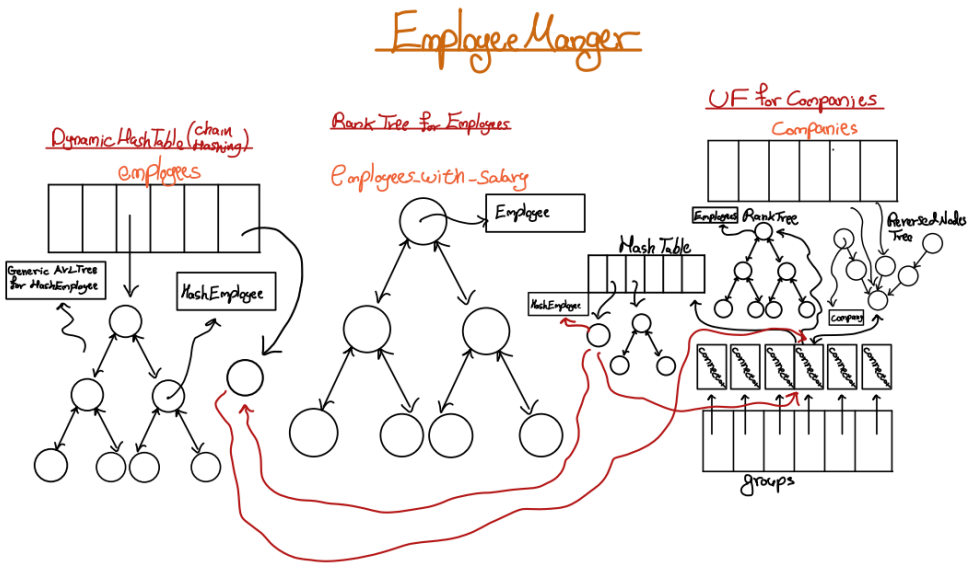
יבש – רטוב 2



תיאור מבנה הנתונים הראשי

מבנה הנתונים מכיל שלוש מחלקות ראשיות:

- **Dynamic Hash Table של עובדים (EmployeeHash)**
- **Ranked AVL Tree של עובדים (Employee)**
- **Union Find של חברות (CompanyUF)**

תיאור הפתרון :

על מנת לעמוד בסיבוכיות, תכננו את המבנה כך ש:

1. **נוכל להוסיף עובדים ללא שכר לטבלאות הערבול בסיבוכיות O(1) משוערך בממוצע על הקלט, ורק בשלב שהעובד מקבל העלאת משכורת, להוסיף אותו לעצי העובדים. מסיבה זו אנו מתחזקים בנוסף לעצים שדות המעידים על סכום הדרגות של העובדים ללא השכר ועל מספר העובדים ללא שכר.**
2. **נוכל לגשת לעובדים בHash table הכללי ודרכם להגיע לConnector (הקשר של כל קבוצה של חברות) המתאים לחברה המעסיקה, דרך הConnector אנו יכולים לבדוק נתונים לגבי החברה המעסיקה בגישה ישירה.**
3. **נוכל לגשת מעובד בטבלת ערבול של חברה לעותק שבטבלת הערבול הכללית ובמקרים בהם כבר היינו צריכים לגשת לחברה (ואלה המקרים בהם ייתכן ו-Connector של ישתנה) נשמור על תחזוק של השדות הנוספים המתאימים (נפרט בהמשך) .**
4. **באמצעות עץ הדרגות נוכל סכימה בצורה יעילה של כל הדרגות של העובדים .**

5. **הדרך בה אנחנו פועלים ב- UF (בדומה לתרגיל של הארגזים מהתרגול) מאפשרת לנו לחשב הערך של כל חברה ולעקוב אחרי הרכישות השונות ובנוסף להתייחס לקבוצה של חברות כך שעדיין כל חברה קיימת בזכות עצמה .**

פירוט על מבנה הנתונים הבסיסיים

Employee

מכיל את השדות:

Id – תעודת זהות של העובד.

Salary – משכורת של העובד.

Grade – דרגה של העובד.

CompanyId – תעודת זהות של החברה המעסיקה את העובד.

מבנה זה הוא המבנה שמיצג עובד ונמצא בעצי הדרגות שקיימים במבנה בנפרד מטבלת הערבול.

שאופרטור ההשוואה של Employee בנוי על השווה בין זוגות סדורים של משכורת ותעודות זהות לפי הסדר המילוני, כאשר ההשוואה היא קודם לפי משכורות ולאחר מכן לפי הID.

בנוסף הID 1- מוגדר כאינסוף ונחשב לפי אופרטור ההשוואה הID הכי גדול. (הקיום של הINF ID עוזר במימוש הפונקציה GradeAveragePerRange בעץ הדרגות).

HashEmployee

בעל אותן שדות כמו ה Employee אך מכיל גם שדות נוספים:

pMain – מצביע לעותק הראשי של ה Employee שנמצא בטבלת הערבול הראשית

Connector – מצביע לconnector המתאים לחברה השוכרת את העובד.

אופרטור ההשוואה של ה Hash Employee בניגוד לעובד הרגיל, עובד רק על השוואה בין ה ID של שני עובדים.

AVLTree

עץ AVL רגיל, זהה לעץ מרטוב 1. עצים מסוג זה יחוברו לטבלת הערבול במקום רשימה מקושרת.

תומך בפעולות הכנסה, הוצאה ומציאת איבר ב ((O(log(n.

EmployeeTree

עץ AVL זהה לעץ מרטוב 1, פרט להיותו עץ דרגות. עץ זה ינהל את עותקי העובדים שמחוץ לטבלת העירבול.

מכיל את השדות:

num_of_free_workers – שדה המכיל את מספר העובדים בעלי משכורת אפס (אם העץ הוא עץ של חברה אז המספר מכיל רק את העובדים בעלי משכורת אפס שגם עובדים בחברה, אם העץ הוא עץ העובדים הכללי אז המספר מכיל את כל העובדים בעלי משכורת אפס).

grade_sum_of_free_workers -שדה המכיל את סכום הgrade של העובדים בעלי משכורת אפס (ההערה לעיל נכונה גם כאן).

כל Node בעץ מכיל גם את השדות Grades וSize :

Size – מכיל את מספר הצמתים בתת העץ שהשורש שלו זה הNode בו אנו קוראים את המשתנה.

Grades - מכיל את סכום הgrades בתת העץ שהשורש שלו זה הNode בו אנו קוראים את המשתנה.

השדות size וgrade מתוחזקים באותה אופן בו מתחזקים עץ דרגות בהכנסה, והוצאה של איברים ובזמן ביצוע גלגולים.

העץ תומך בפעולות הוספה, מחיקה ומציאה של איברים בסיבוכיות (O(log(n)). בנוסף העץ תומך בפעולת איחוד עצים בסיבוכיות O(n1+n2)(בדומה לאיחוד העצים מתרגיל רטוב 1) . (כאשר n1 הוא מספר העובדים בעץ אחד ו2n מספר העובדים בעץ השני).

Company

מבנה נתונים המייצג חברה ונמצא בCompanyUF כNode שמחובר לעץ הפוך של חברות.

המבנה מכיל את השדות:

- **CompanyID** – תעודת זהות של החברה.
- **PartialVal** – סכום חלקי שעוזר לחשב את ה Value של החברה.
- **Parent** – מצביע לNode האב.
- **Connector** – מבנה המכיל את המידע על קבוצת החברות אליה החברה משויכת.

Connector

מבנה נתונים המכיל מידע על קבוצת חברות.

המבנה מכיל את השדות:

- **Num –** מספר החברות בקבוצת החברות.
- **mValue** – הערך של החברה הראשית של בקבוצה (הכוונה לחברה שרכשה את כל החברות האחרות "הרוכשת הראשית").
- **employeesTree** – עץ המכיל את כל העובדים של החברות ששייכות לקבוצה הנוכחית
- **employeeHash** – טבלת ערבול המכילה את כל העובדים של החברות ששייכות לconnector.

דרך מבנים מסוג זה מתחזקים את כל המידע של כל קבוצות החברות שבמערכת.

בנוסף לאורך כל המימוש דאגנו לתחזק מצביע ל-Connector על מנת לגשת מכל עובד לקבוצת החברות שהוא עובד בה .

CompanyUF

מבנה הנתונים מכיל את השדות הבאים:

- **companies** – מערך של מצביעים לחברות.
- **Groups** – מערך של מצביעים לconnector של כל החברות.

מבנה UF כאשר כל node מייצג חברה. המבנה פועל כמו מבנה הUF של הקופסאות מהתרגול. כלומר המבנה כולל משתנה נוסף בכל node, הנקרא partialSum שסכימתו מהnode הנוכחי עד לroot של העץ חברות אליו הNode מחובר, יניב את ה value של החברה המיוצגת על ידי ה node.

המבנה מאותחל בסיבוכיות O(k). המבנה תומך בפעולת איחוד בין קבוצות שמתבצעת ע"י איחוד עצי העובדים וטבלאות הערבול של העובדים וע"י איחוד החברות בעזרת איחוד עצים הפוכים לפי גודל וקיצור מסלולים ואיחוד הConnectorים של שני הקבוצות.

לכן האיחוד מתבצע בסיבוכיות:

הUF ממומש בעזרת קיצור מסלולים ובעזרת איחוד קבוצות לפי גודל מה שהופך את הסיבוכיות של האיחוד של נטו החברות (ללא האיחוד של המכלי עובדים) והחיפוש להיות O(log*(k)) משוערך.

EmployeeHash

Hash table של עובדים הבנוי בצורת Chain-hashing כך שבמקום רשימות מקושרות לכל תא במערך הדינאמי של ה hash table יש עץ AVL של עובדים. המערך בנוי כך שתמיד גודל הטבלה היא בגודל O(n) כאשר n הוא מספר העובדים בטבלה. כך מתאפשר פעולות שהן O(1) בממוצע על הקלט.

בעצם בכך שאנו דואגים שגודל הטבלה יהיה O(n), במומצע על הקלט, התפלגות העובדים בתאי הטבלה היא עובד לכל תא. כך בעצם כל גישה לאיברים בטבלה נשארת O(1) בממוצע על הקלט.

אנו משנים את גודל המערך בהוספה\מחיקה של עובדים רק כאשר מספר העובדים שווה למספר התאים במערך, או שמספר העובדים שווה לרבע ממספר התאים במערך. בכל שינוי גודל של המערך אנו או מגדילים אותו פי 2 או מקטינים אותו פי 2. בכל מקרה בין שני פעולות של שינוי גודל הטבלה, חייב להתבצע לפחות n/4 פעולות הוספה\החסרה של איברים. מכיוון שבין פעם אחת בה אנו משנים את גודל המערך לפעם הבאה בה אנו מבצעים זאת יש theta(n) פעולות הוספה ומחיקה של איברים, אנו יכולים להגיד כי פעולות הוספה והחסרה מתבצעות בסיבוכיות O(1) משוערך בממוצע על הקלט.

טבלת הערבול תומכת בפעולות הוספה, מחיקה ומציאת איברים ב O(1) משוערך בממוצע על הקלט, ובפעולת איחוד של שני טבלאות בסיבוכיות O(n1+n2) שפעולת באופן דומה להגדלת המערך רק שמבצעים Rehash של כל אחד מהמערכים למערך הגדול יותר (ולכן הסיבוכיות היא כגודל המערך החדש) .

הסבר על הפונקציות המשוערות :

באופן בוא אנחנו משתמשים במבנה ה-UF חיפוש קורה דרך ה-HashTable ולכן בכל חיפוש בחברה אנחנו עושים זאת בסיבוכיות של O(1) . בנוסף בחלק מהחיפושים (למשל בgetValue או ב-AddEmployee) מתבצע קיצוץ ענפים) ובפונקציית ה-Union אנחנו מאחדים קבוצות לפי גודל מה שהופך פעולות כבדות יותר להיות בסיבוכיות של O(log*k) .

ולכן בהתאם לפונקציה הרלוונטית בה נעשית הפעולה בדומה למימוש של מבנה UF רגיל הסיבוכיות יהיו O(1) או log*(k) בהתאמה .

פונקציות מבנה הנתונים הראשי:

Void* init(int k)

- מאתחל את המבנה הנתונים, ה EmployeeHash, וה EmployeeTree, מתאתחלים ב O(1) כמבנים רקים.
- ה CompanyUF מתאתחל ב O(k).

סך הכל בסיבוכיות O(k).

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

- מכניס את העובד לטבלאות הערבול בסיבוכיות O(1) משוערך בממוצע על הקלט.
- מחפש את החברה ב-UF ומוסיף את העובד לטבלת הערבול של החברה , בחיפוש זה אנחנו גם מעדכים את העובד שנכנס לחברה להצביע על העובד שנמצא בטבלת הערבול הכללית ולשניהם נעדכן את ה-Connector . O(1) משוערך על הקלט בגלל טבלת העירבול וO(log*(k) עבור חיפוש ב-UF .
- מעדכן את השדות ששומרים את מספר העובדים עם משכורת 0 בעץ העובדים הכללי ובעץ המתאים של החברה שמעסיקה (לאחר שחפשנו ב-UF הגישה היא גישה ישירה לעץ העובדים) O(1) .
- *בפונקציה זו אנחנו מבצעים חיפוש של חברה ב-O(log(k) .

סך הכל בסיבוכיות (O(log*(k)) משוערך בממוצע על הקלט.

StatusType removeEmployee(void *DS, int EmployeeID)

- גישה ישירה מהעובד בטבלת הערבול לקבלת ה-Connector , כך שבאמצעותו נוכל לגשת בגישה ישירה לעץ החברות ולטבלת העובדים של קבוצת שמעסיקה את העובד .
- מסירה את העובד מטבלאות הערבול O(1) בממוצע על הקלט משוערך.
- מסירה את העובד מהעצים המתאימים בסיבוכיות (O(log(n)).
- מעדכנת את השדות הרלוונטיים בסיבוכיות O(1)

*נשים לב שבמקרה בו לעובד יש שכר 0 , רק נוציא אותו מהטבלאות המתאימות ונעדכן את השדות המתאימים בעצים של עובדי החברה ושל העובדים .

*בפונקציה זו אנחנו מבצעים חיפוש של חברה ב-O(1) .

סך הכל O(log(n)) בממוצע על הקלט ומשוערך.

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

- מאחדת את החברות O(log*(k)) משוערך.
- מאחדת את העצי עובדים המתאימים תוך עידכון השדות הרלוונטיים על עובדים בשכר 0 O(n_acquirer+n_target).
- לאחר איחוד העצים נעדכן את השדות של מספר העובדים בכל צומת בעץ בעזרת חיפוש PostOrder O(n_acquirer+n_target)
- מאחדת את טבלאות העובדים המתאימות (כפי שתואר לעיל) ומעדכנת את השדות המתאימים לכל עובד בטבלאות העירבול של החברה ושכל כלל העובדים O(n_acquirer+n_target).
- מחשבת את הערך של קבוצת החברות המאוחד ומעדכן את השדות הרלוונטים O(log*(k)) משוערך.

סך הכול סיבוכיות O(log*(k)+n_acquirer+n_target) משוערך בממוצע על הקלט.

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

- גישה ישירה מהעובד בטבלת הערבול לקבלת ה-Connector , כך שבאמצעותו נוכל לגשת בגישה ישירה לעץ החברות ולטבלת העובדים של קבוצת שמעסיקה את העובד .
- מוסיפה את העובד לעצי העובדים במקרה הצורך (אם היה עם שכר 0 יש להכניסו ובמקרה בו יש לו שכר נצטרך לבצע הוספה והכנסה על מנת לשמור על איזור הערכים בעץ) O(log(n)).

בסך הכל סיבוכיות O(log(n)) בממוצע על הקלט.

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

- גישה ישירה מהעובד בטבלת הערבול לקבלת ה-Connector , כך שבאמצעותו נוכל לגשת בגישה ישירה לעץ החברות ולטבלת העובדים של קבוצת שמעסיקה את העובד .O(1)

  במקרה שלעובד שכר 0 נעדכן את השדות המתאימים בעצי העובדים הכללי והשל החברה .O(1)

- במקרה ובו יש לו שכר גדול מ-0 נצטרך לבצע הוספה והכנסה על מנת לשמור את התקינות של השדות הנוספים בעץ הדרגות O(log(n)).

סך הכל בסיבוכיות O(log(n)) בממוצע על הקלט.

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

מחשבת את סכום הgrade של m העובדים עם המשכורות הכי גבוהות.

הפונקציה פועלת כך שהיא מחפשת את האיבר ה m הכי גדול בעץ, ובזמן החיפוש סוכמת את סכום הgrade של כל העובדים שגדולים או שווים לאיבר הm הכי גדול בעץ.

הפונקציה מתחילה משורש העץ.

אם היא פונה לכיוון ימין בחיפוש האיבר הm הכי גדול בעץ, היא לא מוסיפה לסכום הgrade כי כל מי שמשמאל כולל השורש, הוא קטן ממש מm האיברים הכי גדולים בעץ.

אם היא פונה שמאלה אז היא תבדוק את הסכום של הgrade בתת העץ השמאלי, ותוסיף לו את הסכום של כל הgrade של האיברים מתת העץ הימני ושל השורש כי כולם גדולים מהאיבר הm הכי גדול.

כאשר האיבר הm הכי גדול נמצא, היא מוסיפה לסכום את סכום הgrade של תת העץ שהאיבר הm הוא השורש שלו.

פעולת הפונקציה חסומה על ידי מספר קבוע k כפול גובה העץ log(n).

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

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

הפונקציה קוראת לפעמיים לפונקציה calcNumOfSmallerElements, פונקציה אשר יורדת במורד העץ וסוכמת בדרך כמו איברים קטנים ממש מהאיבר שהוכנס. בדומה לפונקציה find, פונקציה זו פועלת בסיבוכיות O(log(n)).

לאחר מכן הפונקציה קוראת פעמיים לפונקציה topWorkersGradeSum אשר פועלת גם בסיבוכיות O(log(n)).

ובנוסף מבצעת פעולות נוספת ב O(1).

בסיבוכיות זמן O(log*(k)+log(n)) משוערך.

StatusType companyValue(void *DS, int CompanyID, void * standing)

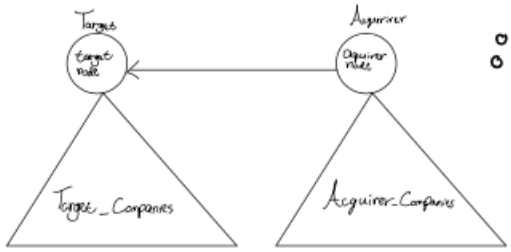
הציור ממחיש את הדרך בה אנחנו מתחזקים את השדות כאשר אנחנו רוכשים חברה (ובכך מראה את נכונות האלגוריתם) .

בזכות עדכון השדות של הסכום החלקי נגש לחברה בטבלה החברות של ה-UF .

ובדומה לחיפוש ב-UF נסכום את הסכום החלקי של ערך החברה עד לשורש (כפי שקורה בחיפוש ובנוסף נבצע גם קטיעת ענפים כפי שנלמד בתרגול בבעיית הארגזים) .

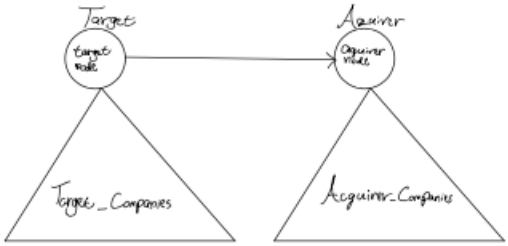
לאחר שנקבל את הסכום החלקי עד לשורש (לא כולל) החל מהצומת בה התחלנו את החיפוש נחבר את אותה לשורש ונעדכן את ה-partialVal שלה להיות הסכום הנוכחי ולאחר שעדכנו נוריד את הוואליו הישן שהיה לה מסכום זה ונמשיך לצומת הבאה .

בסה"כ החיפוש ב-UF משוערך ב-log*(k) .



: |Target| ≥ |Aquirer|

partialVal(target-node)old = partialVal(target-node)new
partialVal(aquirer-node)new -= partialVal(target-node)new
partialVal(aquirer-node)new += Factor*mValue(target)



: |Target| < |Aquirer|

partialVal(aquirer-node)new += Factor*mValue(target)
partialVal(target-node)new -= partialVal(aquirer-node)new

בסה"כ סיבוכיות זמן O(log*(k)) משוערך.

void Quit(void **DS)

במהלך ריצת התוכנית אנחנו מחזיקים עצי AVL ועצי דרגות כפעמיים מספר העובדים , טבלאות עירבול דינאמיות כגודל פעמיים מספר העובדים לכל הדיסטרקטורים של האובייקטים יעברו על כל האיברים על מנת לשחרר את הזכרון שהוקצה .

את מבנה ה-UF נעבור על המערכים בסה"כ יש 2 מערכים כגודל מספר החברות ונשחרר את הזיכרון שהוקצה .

לכן בסה"כ סיבוכיות המקום בזמן ריצת התוכנית הוא O(n+k) ולכן בפונקציה זו כאשר עלינו לשחרר את הזכרון עלינו לעבור על O(n+k) איברים וזו סיבוכיות הזמן של הריצה .