**מבני נתונים 1**

**234218**

**תרגיל רטוב 1**

**סטודנטים:**

בוראן סוויד – ת"ז 211516836.

סוראי סוויד ­­– ת"ז 322827239.

**תיאור המבנה:**

המבנה שלנו מכיל 4 עצי AVL מאוזנים הבאים:

1. **companiesTreeByID**- זהו עץ AVL שמכיל מידע על כל החברות שאנו מוסיפים למערכת, והוא ממוין לפי המספר המזהה של החברה.

כל צומת בעץ מכילה את המידע הבא:

1. companyID- זהו המספר המזהה של החברה.
2. Value- שווי החברה.
3. **activeCompaniesTree**- זהו עץ AVL שמכיל מידע על כל החברות אשר מעסיקות עובדים בלבד והוא ממוין לפי מספר מזהה של החברה.

כל צומת בעץ מכילה את המידע הבא על החברה:

1. company\_id- זהו הסמפר המזהה של החברה.
2. Employees\_number- מספר העובדים בחברה.
3. employeesBySalary- זהו עץ AVL ששומר את העובדים שמועסקים בחברה זאת, והוא ממוין ראשית לפי השכר ושנית לפי מספר מזהה של העובד.
4. highestSalary- שומר מצביע לצומת של העובד בעל השכר הכי גבוה בתוך העץ employeesBySalary של כל חברה.
5. employeesByID- שהו גם עץ AVL מאוזן שבו אנו שומרים את העובדים של אותה חברה, והוא ממוין לפי מספר מזהה של העובד.
6. **EmployeesTreeByID**- זהו עץ AVL אשר שומר את כל העובדים שהוספנו למערכת, והוא ממוין לפי מספר מזהה של העובדים.

כל צומת בעץ מכילה את המידע הבא:

1. Employee\_id- המספר המזהה של העובד.
2. Employer\_id- המספר המזהה של החברה שמעסיקה את העובד.
3. Salary- שכר של העובד.
4. Grade- הדרגה של העובד.
5. **EmployeesTreeBySalary**- עץ AVL אשר שומר מידע על כל העובדים הנמצאים במערכת, והוא ממוין ראשית לפי השכר של העובדים ושנית לפי מספר מזהה של העובדים.

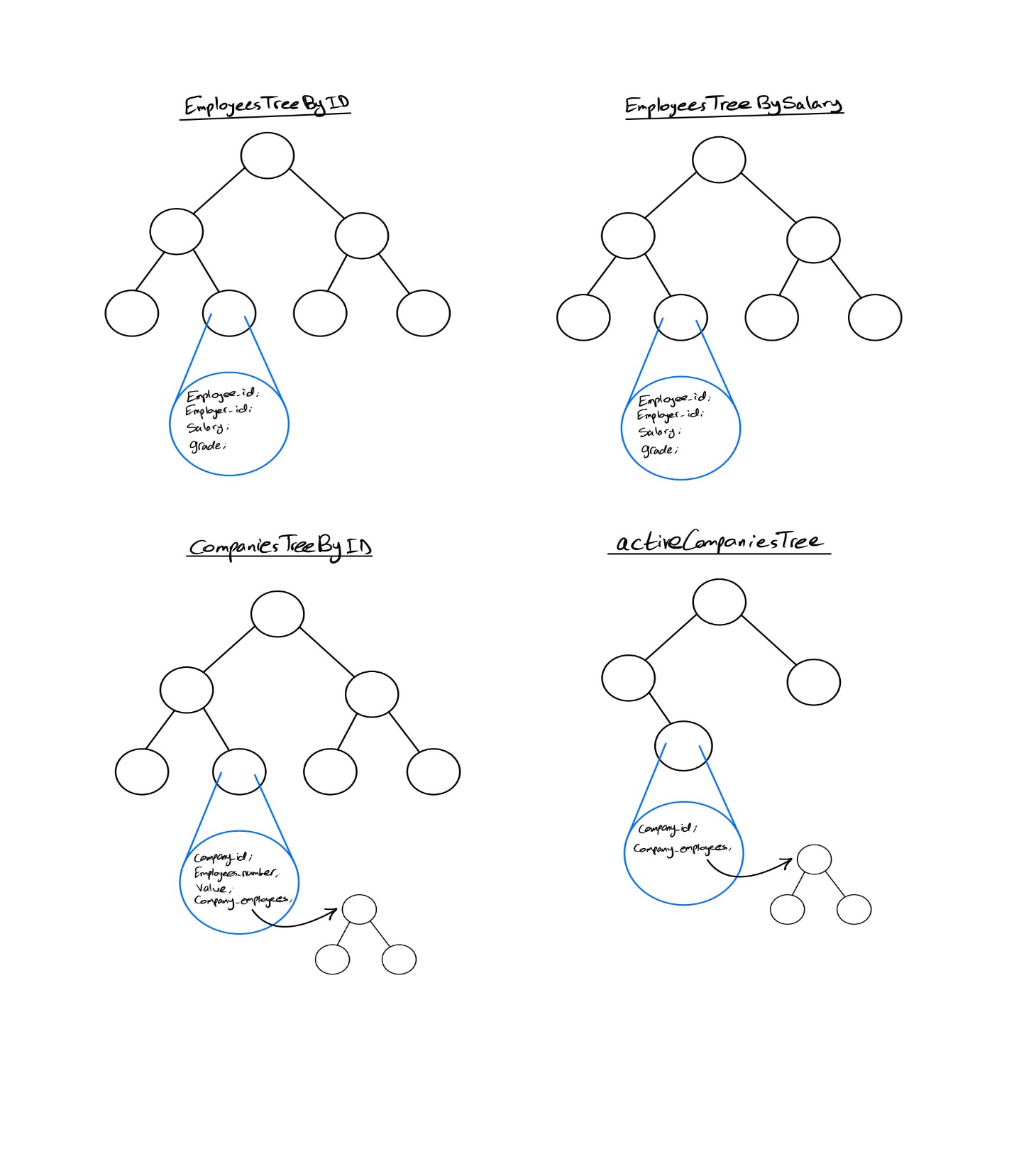
כל צומת בעץ מכילה את המידע הבא:

1. Employee\_id- המספר המזהה של העובד.
2. Employer\_id- המספר המזהה של החברה שמעסיקה את העובד.
3. Salary- שכר של העובד.
4. Grade- הדרגה של העובד.
5. highestSalaryAll- שומר מצביע לצומת של העובד בעל השכר הכי גבוה בתוך העץ **EmployeesTreeBySalary**.

בנוסף, שמרנו שני שדות למבנה numberOfCompanies, numberOfEmployees ששומרים את מספר החברות הכולל במערכת, ואת מספר העובדים הכולל במערכת בהתאמה.

(צריך להסיר אותם אם לא עשינו בהם שימוש ואולי להוסיף שדות)

**נצרף תיאור גרפי למבנה:**



**סיבוכיות המקום:**

* שמרנו שני עצים אשר מכילים מידע על העובדים, **EmployeesTreeByID** ו- **EmployeesTreeBySalary** ,כל עץ מכיל לכל היותר n צמתים (כאשר n הוא מספר העובדים הכולל במערכת), כאשר בכל צומת אנו שומרים 4 שדות, ולכן כל צומת לוקחת O(1) מקום, וסה"כ עבור כל העץ נקבל O(n) מקום, ולכן עבור שני העצים O(n) + O(n) = O(2n) = O(n).
* בשני העצים של החברות **companiesTreeByID** ו- **activeCompaniesTree**, בכל עץ יש לכל היותר k צמתים (כאשר k הוא מספר החברות הכולל במערכת), ובכל צומת אנו שומרים מספר קבוע של שדות בנוסף לשני עצים AVL עבור העובדים השייכים לאותה חברה בתוך העץ המכיל חברות מעסיקות עובדים בלבד, ולכן בכל צומת יש לנו

O(1) + O(ni) = O(ni) סיבוכיות מקום (כאשר ni הוא מספר העובדים עבור החברה הספציפית), עבור כל הצמתים ביחד נקבל סיבוכיות מקום עבור עץ חברות אחד:

נסביר את (\*):

סכום כל העובדים השייכים לחברות שווה ל-n שהוא מספר העובדים הכולל במערכת.

ולכן נקבל עבור עץ החברות activeCompanies O(n+2k) סיבוכיות מקום, ועבור העץ companiesByID O(n) , עבור שני העצים ביחד נקבל:

O(n)+O(n+2k) = O(2n+2k)=O(n+k).

לכן סיבוכיות מקום כוללת עבור כל המבנה:

וזה אכן עומד בסיבוכיות הדרושה.

**תיאור הפונקציות והוכחת סיבוכיות:**

(נציין שלאורך הוכחת הסיבוכיות עבור הפונקציות- k הוא מספר החברות הכולל במערכת ו- n הוא מספר העובדים הכולל במערכת).

**הערה: בפונקציות הבאות בדקנו את ערכי השגיאה ב- O(1) זמן במקרה הגרוע והחזרנו INVALID\_INPUT כנדרש ומסיימים את הפונקציה.**

**Void\* init():**

נאתחל מבנה נתונים ריק שמכיל:

* 4 עצי AVL ריקים ב- O(1) זמן לכל אחד.
* נאתחל את השדות numberOfCompanies ו- numberOfEmployees לערך 0 ב- O(1) זמן.
* נאתחל את השדה highestSalaryAll ל- NULL.

ולכן סה"כ סיבוכיות זמן- O(1) כנדרש.

**StatusType AddCompany(void \*DS, int CompanyID, int Value):**

קודם כל אנו בודקים אם companyID נמצא במערכת, ולכן אנו מחפשים בעץ companiesTreeById ואם הוא נמצא אז נחזיר FAILURE ונסיים, חיפוש בעץ AVL לוקח סיבוכיות זמן O(logk) כאשר k הוא מספר הצמתים בעץ כלומר מספר החברות במערכת.

אחרת, אנו מוסיפים צומת חדש עבור חברה זאת לעץ companiesTreeById עם companyID ו- value שאנו מקבלים כפרמטרים, ומאתחלים את שדה ה- Employees\_number ל- 0, ושני עצי עובדים ריקים. האתחולים לוקחים O(1) זמן, והוספה לעץ לוקחת O(logk) כאשר k הוא מספר הצמתים בעץ (כפי שנלמד).

לבסוף, אנו מוסיפים 1 לשדה numberOfCompanies, ומחזירים SUCCESS.

סה"כ סיבוכיות זמן עבור פונקציה זאת: O(logk)+O(logk)=O(2logk)=O(logk) כנדרש.

**StatusType AddEmployee(void \*DS, int EmployeeID, int CompanyID, int Salary, int Grade):**

קודם כל אנו בודקים אם העובד בעל מספר מזהה **EmployeeID** נמצא במערכת, ולכן נחפש אותו בעץ employeesTreeByID, זמן חיפוש בעץ זה הוא O(logn) כאשר n הוא מספר העובדים במערכת, בנוסף, נבדוק אם החברה בעלת מזהה **CompanyID** לא נמצאת במערכת ולכן נחפש אותה בעץ companiesTreeByID, זה לוקח O(logk) זמן כאשר k הוא מספר החברות המערכת.

אם לפחות אחד התנאים לעיל מתקיים, נחזיר FAILURE ונסיים.

אחרת, נוסיף עובד חדש למערכת באופן הבא:

* נוסיף אותו לעץ employeesTreeByID ולעץ employeesTreeBySalary כאשר נאתחל את השדות שלו כפי שקיבלנו אותם בפרמטרים של הפונקציה ונעדכן את השדה highestSalaryAll .

הוספת צומת חדש לכל עץ מהם לוקחת O(logn) סיבוכיות זמן כאשר n הוא מספר העובדים הכולל במערכת.

* נוסיף את העובד לעץ employeesByID ו- employeesBySalary השייך לחברה בעלת **CompanyID**, ולכן קודם כל נמצא את החברה בעץ החברות activeCompaniesTree כדי לגשת למידע שלה, זמן חיפוש לוקח O(logn), ואז נוסיף את העובד עם הנתונים שלו לשני העצים של העובדים של אותה חברה, סיבוכיות זמן הוספה לעץ O(log(ncompanyID)) כאשר ncompanyID הוא מספר העובדים השייכים לחברה הספציפית, ונעדכן את השדה highestSalary ב-( O(log ncompanyID
* לבסוף, לא נשכח להוסיף 1 למונה numberOfEmployees של המערכת, ולשדה Employees\_number של החברה בעלת מזהה companyID.

ולסיום נחזיר SUCCESS.

סה"כ סיבוכיות זמן:

כנדרש.

(\*) חסום ע"י n .

**StatusType RemoveEmployee(void \*DS, int EmployeeID):**

קודם נבדוק אם העובד בעל מספר מזהה EmployeeID קיים במערכת בזמן O(logn), אם הוא לא קיים אז נחזיר FAILURE.

אחרת, נסיר את העובד משני עצי העובדים הכללים ב- O(2logn) , ואז נחפש את החברה שמעסיקה עובד זה:

* נסיר אותו מעצי העובדים השמורים בתוך הצומת של החברה שלו מעץ activeCompanyTree ב- O(logn)
* נחסיר אחד מnumberOfEmployees השמור בתוך הצומת של חברה זו, אם הוא הגיע לאפס, אז נסיר חברה זו מעץ activeCompanyTree ב- O(logn).

(נשים לב כי מספר החברות בעץ activeCompany.

* נעדכן את המצביעים highestSalaryAll ו- highestSalary של החברה ב- O(logn)

נקבל סה"כ סיבוכיות O(3logn) = O(logn)

**StatusType RemoveCompany(void \*DS, int CompanyID):**

קודם כל נבדוק אם החברה בעלת companyID נמצאת במערכת ע"י חיפוש בעץ companiesTreeByID, בנוסף נבדוק אם היא נמצאת ב- activeCompanyTree כלומר יש לה עובדים, בדיקות אלו לוקחים O(logk) סיבוכיות זמן עבור חיפוש בעץ AVL.

אם אחד התנאים לעיל לא מתקיים אז נחזיר FAILURE ונסיים.

אחרת, נסיר את החברה בעלת companyID מהעץ companiesTreeByID, סיבוכיות זמן של הסרת צומת מעץ AVL היא O(logk) כפי שנלמד.

נשים לב שהחברה לא נמצאת בעץ **activeCompaniesTree** כי אנחנו צריכים לוודא שאין בה עובדים לפני הסרתה.

לבסוף נחזיר SUCCESS.

סה"כ סיבוכיות זמן O(logk) כנדרש.

**StatusType GetCompanyInfo(void \*DS, int CompanyID, int \*Value, int \*NumEmployees):**

קודם כל נבדוק אם החברה בעלת מזהה companyID נמצאת במערכת, ולכן נחפש אותה בעץ companiesTreeByID ב- O(logk) זמן, אם היא לא נמצאה אז נחזיר FAILURE .

אחרת, נבדוק אם החברה נמצאת בעץ activeCompanyTree, ניגש לdata של הצומת שלה בעץ ונשמור את שדה ה- value ואת Employees\_number שלה במצביעים המתאימים שקיבלנו כפרמטרים, אם היא לא נמצאת בעץ, אז ניגש אליה בעץ companiesTreeByID וניקח את ה- value השמור בצומת, ונציב 0 ל- NumEmployees.

לבסוף, נחזיר SUCCESS.

סה"כ סיבוכיות זמן : O(logk) כנדרש.

**StatusType GetEmployeeInfo(void \*DS, int EmployeeID, int \*EmployerID, int \*Salary, int \*Grade):**

קודם כל נבדוק אם העובד בעל מזהה employeeID נמצא במערכת, ולכן נחפש אותו בעץ employeesTreeByID בזמן O(logn) , אם הוא לא נמצא אז נחזיר FAILURE ונסיים.

אחרת, ניגש לצומת של עובד זה בעץ employeesTreeByID, ונשמור את השדות employeeID, salary ו- grade לתוך המצביעים שאנו מקבלים כפרמטרים לפונקציה.

ולסיום, נחזיר SUCCESS.

סה"כ סיבוכיות זמן: O(logn) כנדרש.