מבני נתונים 1

234218

תרגיל רטוב 1

סטודנטים:

בוראן סוויד – ת״ז 211516836.

סוראי סוויד – ת״ז 322827239.

<u>תיאור המבנה:</u>

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:
   i. companyID- זהו המספר המזהה של החברה.
   ii. Value- שווי החברה.
2. **activeCompaniesTree**- זהו עץ AVL שמכיל מידע על כל החברות אשר <u>מעסיקות עובדים</u> בלבד והוא ממוין לפי מספר מזהה של החברה.
   כל צומת בעץ מכילה את המידע הבא על החברה:
   i. company_id- זהו הסמפר המזהה של החברה.
   ii. Employees_number- מספר העובדים בחברה.
   iii. employeesBySalary- זהו עץ AVL ששומר את העובדים שמועסקים בחברה זאת, והוא ממוין ראשית לפי השכר ושנית לפי מספר מזהה של העובד.
   iv. highestSalary- שומר מצביע לצומת של העובד בעל השכר הכי גבוה בתוך העץ employeesBySalary של כל חברה.
   v. employeesByID- שהו גם עץ AVL מאוזן שבו אנו שומרים את העובדים של אותה חברה, והוא ממוין לפי מספר מזהה של העובד.
3. **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:
   i. Employee_id- המספר המזהה של העובד.
   ii. Employer_id- זהו מצביע משותף ששמור בתוכו המספר המזהה של החברה.
   iii. Salary- שכר של העובד.
   iv. Grade- הדרגה של העובד.
4. **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:
   i. Employee_id- המספר המזהה של העובד.
   ii. Employer_id- זהו מצביע משותף ששומר את המספר המזהה של החברה.
   iii. Salary- שכר של העובד.
   iv. 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.

<u>סיבוכיות המקום:</u>

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

$$O(n_1) + O(n_2) + \cdots + O(n_k) = \sum_{i=1}^{k} O(n_i) \underbrace{=}_{*} O(n)$$

נסביר את (*):
סכום כל העובדים השייכים לחברות שווה ל-n שהוא מספר העובדים הכולל במערכת.

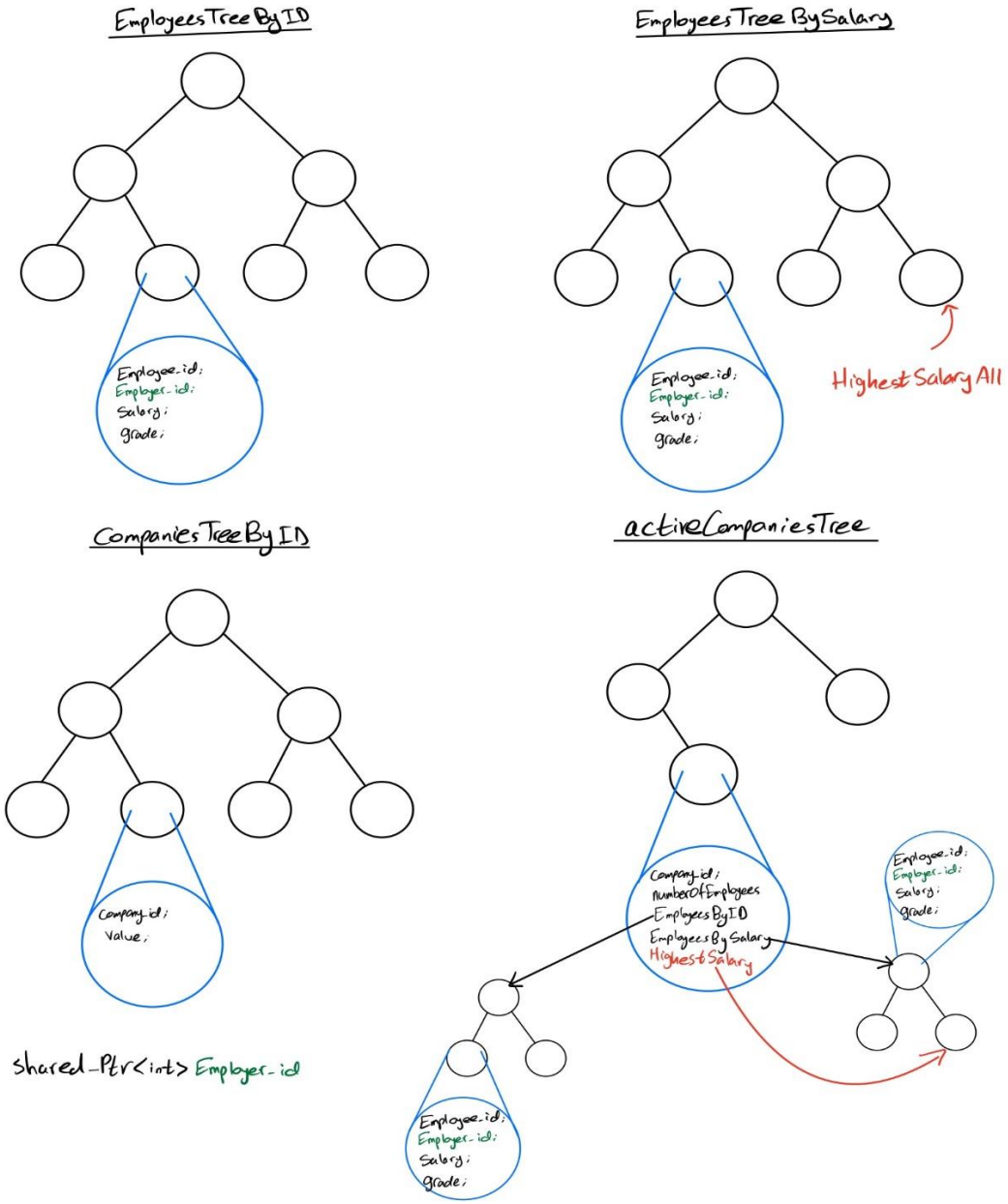
ולכן נקבל עבור עץ החברות activeCompanies O(n+2k) סיבוכיות מקום, ועבור העץ
O(n) companiesByID , עבור שני העצים ביחד נקבל:
O(n)+O(n+2k) = O(2n+2k)=O(n+k).

לכן סיבוכיות מקום כוללת עבור כל המבנה:

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

וזה אכן עומד בסיבוכיות הדרושה.

<u>נצרף תיאור גרפי למבנה:</u>



<u>תיאור הפונקציות והוכחת סיבוכיות:</u>

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

<u>הערה:</u> 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:

- נגדיר shared pointer עבור המספר המזהה של החברה כדי להוסיף אותו לעובד בכל עץ.
- 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($n_{companyID}$)) כאשר $n_{companyID}$ הוא מספר העובדים השייכים לחברה הספציפית, ונעדכן את השדה highestSalary ב- O(log $n_{companyID}$)

- לבסוף, לא נשכח להוסיף 1 למונה numberOfEmployees של המערכת, ולשדה Employees_number של החברה בעלת מזהה companyID.

ולסיום נחזיר SUCCESS.

סה״כ סיבוכיות זמן:

$O(logn) + O(logk) + 2O(logn_{companyID}) \underbrace{=}_{*} O(logn + logk)$ כנדרש.

(*) $n_{companyID}$ 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(logn) = O(3logn)

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.

StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease):

קודם כל נבדוק אם החברה נמצאת במערכת, ולכן נחפשה בעץ companiesTreeByID, בסיבוכיות זמן O(logk), אם היא לא נמצאת אז נחזיר FAILURE.

אחרת, נגדיל את השווי של החברה ב- valueIncrease ב- O(1) זמן, ונחזיר SUCCESS.

סה״כ סיבכויות זמן O(logk) כנדרש.

StatusType PromoteEmployee(void *DS, int EmployeeID, int SalaryIncrease, int BumpGrade):

קודם כל נבדוק אם העובד נמצא במערכת דרך חיפושו בעץ employeesTreeByID, בסיבוכיות זמן O(logn), אם הוא לא נמצא נחזיר FAILURE.

אחרת, נשמור את מספר המעסיק ואת השכר של העובד בצד כדי שנוכל לחפש אותו בעצים האחרים ונבצע :

- ניגש אליו בעץ employeesTreeByID ב-O(logn) נעדכן את השכר שלו ב- O(1) זמן.
- עבור העץ employeesTreeBySalary נמחק את הצומת של אותו עובד ב- O(logn) ונכניס אותו מחדש עם השכר המעודכן ב- O(logn).
- נעדכן את ההמצביע highestSalaryAll ב- O(logn).
- נחפש את החברה שלו בעץ activeCompanyTree ב- O(logk) זמן, ואז נעדכן אותו בשני העצים employeeTreeByID ו- employeeTreeBySalary כמו שעשינו בעצים הגדולים ב- 2O(logn) זמן.
- נעדכן את המצביע highestSalary של אותה חברה ב- O(logn) זמן.
- נשים לב כי אם BumpGrade היה חיובי אז נוסיף לכל עדכון של שכר שתואר למעלה, גם עדכון עבור הדרגה שהוא יוסיף O(1) סיבוכיות זמן.

נקבל סה״כ סיבוכיות זמן :

$$70(\log n) + O(\log k) \underset{*}{=} 80(\log n) = O(\log n)$$

הסבר ל- (*) : מספר החברות המעסיקות עובדים חסום על ידי מספר העובדים הכולל, הרי במקרה הכי גרוע הוא כאשר בכל חברה יש עובד אחד , ואז נקבל שמספר החברות המעסיקות עובדים הוא כמספר העובדים הכולל.

StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID):

קודם כל נבדוק אם העובד והחברה נמצאים במערכת בזמן O(logk) + O(logn) *זמן, אם לא אז נחזיר* FAILURE.

אחרת, נבצע את הבא:

- *קודם כל ניגש למידע של העובד בעץ* empliyeesTreeByID *ונשמור את השכר והדרגה שלו בצד.* O(logn)
- *נשתמש בפונקציה* removeEmployee() *שמימשנו קודם, ונסיר את העובד מהמערכת ב-* O(logn) *זמן.*
- *נשתמש בפונקציה* AddEmployee() *שמימשנו קודם ונכניס את העובד מחדש למערכת עם המספר המזהה של המעסיק החדש שלו בזמן* O(logk+logn).

נקבל סה"כ סיבוכיות זמן:

$$4O(logn) + 2O(logk) = O(logn + logk)$$

כנדרש.

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

קודם כל נבדוק אם שתי החברות נמצאות במערכת ב- *2O(logk)* זמן, אם לא אז נחזיר *FAILURE*, בנוסף נבדוק אם שווי החברה הרוכשת הוא לפחות פי 10 משווי החברה הנרכשת, אם זה לא מתקיים אז נחזיר *FAILURE* גם כן.

אחרת, נבצע את הבא:

- ניגש לחברה הרוכשת בעץ *companyTreeByID* בזמן *O(logk)* ונעדכן אם השווי החדש שלה ב- *O(1)*.
- נסיר את החברה הנרכשת מהעץ *companyTreeByID* ב- *O(logk)*.
- כעת, עבור עדכון העץ *activeCompanyTree* נחלק למקרים:
  1. שתי החברות הרוכשת והנרכשת לא מעסיקות עובדים, כלומר שתיהן לא נמצאות בעץ *activeCompanyTree*, ולכן לא נבצע כלום.
  2. אם החברה הרוכשת מעסיקה עובדים אך הנרכשת לא מעסיקה עובדים, אז לא נבצע כלום.
  3. החברה הרוכשת לא מעסיקה עובדים, אך לנרכשת יש עובדים, אז נוסיף את החברה הרוכשת לעץ ב-*O(logk)* ונעתיק את עצי העובדים של החברה הנרכשת לרוכשת ב-$O(n_{targetID})$, ונעדכן את מספר העובדים של החברה הרוכשת ב-*O(1)*, ונסיר את החברה הנרכשת מהעץ ב- *O(logk)*.
  4. אם שתי החברות מעסיקות עובדים, כלומר שתיהן נמצאות בעץ *activeCompanyTree*, אז קודם כל נאחד את עצי העובדים של שתי החברות ביחד ולכן נבצע פעולת *merge* שלוקחת סיבוכיות זמן $O(n_{acquireID}+n_{targetID})$, ונעדכן את מספר העובדים של החברה הנרכשת לסכום שנמצא בה ובנרכשת.
  ואז נסיר את החברה הנרכשת מהעץ *activeCompanyTree* ב- *O(logk)*.
- לבסוף נקטין את מספר החברות הכולל ב-1 ונחזיר *SUCCESS*.

נקבל סה"כ סיבוכיות זמן במקרה הגרוע:

$$O(logk) + O\left(n_{acqiureID} + n_{targetID}\right) = O\left(logk + n_{acqiureID} + n_{targetID}\right)$$

StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID):

1. אם *companyID < 0* אז נשמור את המספר המזהה של העובד השמור במצביע *highestSalaryAll* לתוך *EmployeeID*.
2. אם *companyID > 0* אז קודם נחפש את החברה בעלת מספר מזהה *companyID* בעץ *activeCompanyTree* ב- *O(logk)* זמן , אם היא לא נמצאת אז נחזיר *FAILURE*, אחרת נשמור

לתוך *EmployeeID* את מספר המזהה של העובד השמור במצביע *highestSalary* של אותה חברה ב- *O(1)*.

קיבלנו סה"כ סיבוכיות *O(logk)+O(1) = O(logk)* במקרה הגרוע כנדרש.

StatusType GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees):

1. אם *companyID < 0* :
   i. נשים את הערך של השדה *numberOfEmployees* לתוך המצביע *NumOfEmployees* ב- *O(1)*.
   ii. נקצה מערך בגודל *numberOfEmployees* שנקרא לו *all_employees*, ונעבור בסיור *inorder* על העץ *employeesTreeBySalary* ונמלא את המערך *all_employees* בצמתים שאנו עוברים בהם באופן מסודר, סיבוכיות זמן של סיור *inorder* היא *O(n)* כאשר *n* הוא מספר הצמתים שאנו עוברים בהם, כלומר מספר העובדים במערכת.
   iii. כיוון שדרוש להחזיר את המזהים של העובדים כך שיהיו מסודרים לפי שכר בסדר יורד, אז נהפוך את המערך ב- *O(n)*.
   iv. לבסוף נשמור מצביע למערך זה ב- *Employees* שקיבלנו כפרמטר.
2. אם *companyID > 0* :
   i. קודם כל נבדוק אם החברה בעלת מזהה *companyID* נמצאת במערכת כך שנחפש אותה בעץ *activeCompaniesTree* בזמן *O(logk)*, אם היא לא נמצאת כלומר אין לה עובדים אז נחזיר *FAILURE*, אחרת נבצע הבא.
   ii. נשים את הערך השמור בשדה *numberOfEmployees* השמור בצומת של חברה זו לתוך הפרמטר *NumOfEmployees* ב- *O(1)*
   iii. נקצה מערך בגודל *numberOfEmployees* שנקרא לו *employeesID_arr* ונעבור בסיור *inorder* על העץ *employeesTreeBySalary* השמור בתוך הצומת של חברה זאת, ונמלא את המערך שלנו בצמתים שאנו עוברים בהם, סיבוכיות של סיור *inorder* היא $O(n_{companiID})$ כאשר $n_{companyID}$ הוא מספר העובדים של אותה חברה.
   iv. כיוון שדרוש להחזיר את המזהים של העובדים כך שיהיו מסודרים לפי שכר בסדר יורד, אז נהפוך את המערך ב- $O(n_{companyID})$.
   v. לבסוף נשמור מצביע למערך זה ב- *Employees* שקיבלנו כפרמטר.

סה"כ סיבוכיות זמן במקרה גרוע : עבור *companyID < 0* - *O(logk) + O(n)*

ועבור *companyID > 0* - $O(logk) + O(n_{companyID})$.

StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCcompanies, int **Employees):

קודם נבדוק אם מספר החברות המעסיקות עובדים כלומר מספר הצמתים בעץ activeCompaniesTree *גדול שווה למספר* NumOfCompanies *ב-* O(1), *אם לא אז נחזיר* FAILURE.

אחרת, נקצה מערך בגודל NumOfCompanies *ונעבור בסיור* inorder *על* NumOfCompanies *החברות בעלות המזהה הכי קטן בסיבוכיות זמן*

O(log k +NumOfCompanies) *כאשר נשמור את הצמתים שאנו עוברים בהם במערך שהקצינו.*

אחר כך נקצה עוד מערך שבו נשמור את המספרים המזהים של העובדים שיוחזרו ולכן נעבור בלולאה על המערך של החברות, וניקח את המזהה של העובד השמור במצביע highestSalary *ב-* O(1) *ולכן סה"כ סיבוכיות עבור הלולאה הוא* O(NumOfCompanies) .

ולכן סה"כ סיבוכיות זמן : O(logk + NumOfCompanies) *כנדרש.*

StatusType GetNumEmployeesMatching(void *DS, int CompanyID, int MinEmployeeID, int MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees):

נחלק לשני מקרים :

1. CompanyID > 0 :
קודם כל נבדוק אם החברה בעלת מזהה companyID *נמצאת במערכת וגם מעסיקה עובדים ב-* O(logk) *זמן, אם לא אז נחזיר* FAILURE.
אחרת :

- ניגש למידע של החברה הזאת בעץ *activeCompaniesTree* ב-*O(logn)* , ונעבור בסיור *inorder* בין שני הצמתים המתאימים לעובדים בעלי מזהים *MinEmployeeID* ו- *MaxEmployeeID* ונספור את מספר הצמתים בניהם, זה לוקח $O(logn_{companyID}) + TotalNumOfEmployees$ סיבוכיות זמן.
- *נקצה מערך מסוג* EmployeeIdData *בגודל* TotalNumOfEmployees *ונעבור בסיור* inorder *בין* *MinEmployeeID* *ו-* *MaxEmployeeID* *בעץ* employeesTreeByID *של החברה ונמלא את המערך בצמתים שאנו עוברים עליהם.*(O(TotalNumOfEmployees).
- *בסוף נעבור על המערך שמילאנו אותו בסעיף הקודם בלולאה, ונבדוק כמה מהעובדים השמורים בו מקיימים שהשכר והדרגה שלהם הוא לפחות* MinSalary *ו-* MinGrade *בהתאמה, נשמור מספר זה ב-* NumOfEmployees. *סיבוכיות זמן* (O(TotalNumOdEmployees).
2. CompanyID < 0 :*נבצע את אותן הפעולות שביצענו במקרה הראשון, אך כעת אנו עוברים על העובדים הנמצאים בעץ* employeesTreeByID *של כל העובדים במע'.*

סה"כ סיבוכיות זמן : עבור CompanyID < 0 – TotalNumOfEmployees + O(logn)

CompanyID > 0 – TotalNumOfEmployees + $OOlogn_{companyID}$) + O(logk) *כנדרש.*

void Quit(void **DS):

*נקרא ל*destructor *של כל המבנים השמורים במערכת שלנו, כאשר אנו צריכים למחוק 4 עצים, שבהם אנו עוברים על כל הצמתים הנמצאים בהם ומבצעים להם* delete, *ולכן נקבל סיבוכיות זמן :*

1. *עבור שני עצי העובדים* employeesTreeByID *ו-* employeesTreeBySalary *הגדולים נקבל* 2O(n) *סיבוכיות זמן.*
2. *עבור העץ של החברות* companiesTreeById *נקבל* O(k).
3. *עבור העץ של החברות המעסיקות עובדים* activeCompaniesTree , *נקבל* 2O(n) *עבור מחיקת העובדים הנמצאים בכל החברות יחד, ולמחיקת החברות נקבל* O(k) *במקרה הגרוע.*

ולכן סה"כ סיבוכיות זמן : 2O(k) + 4O(n) = O(n + k) *כנדרש.*