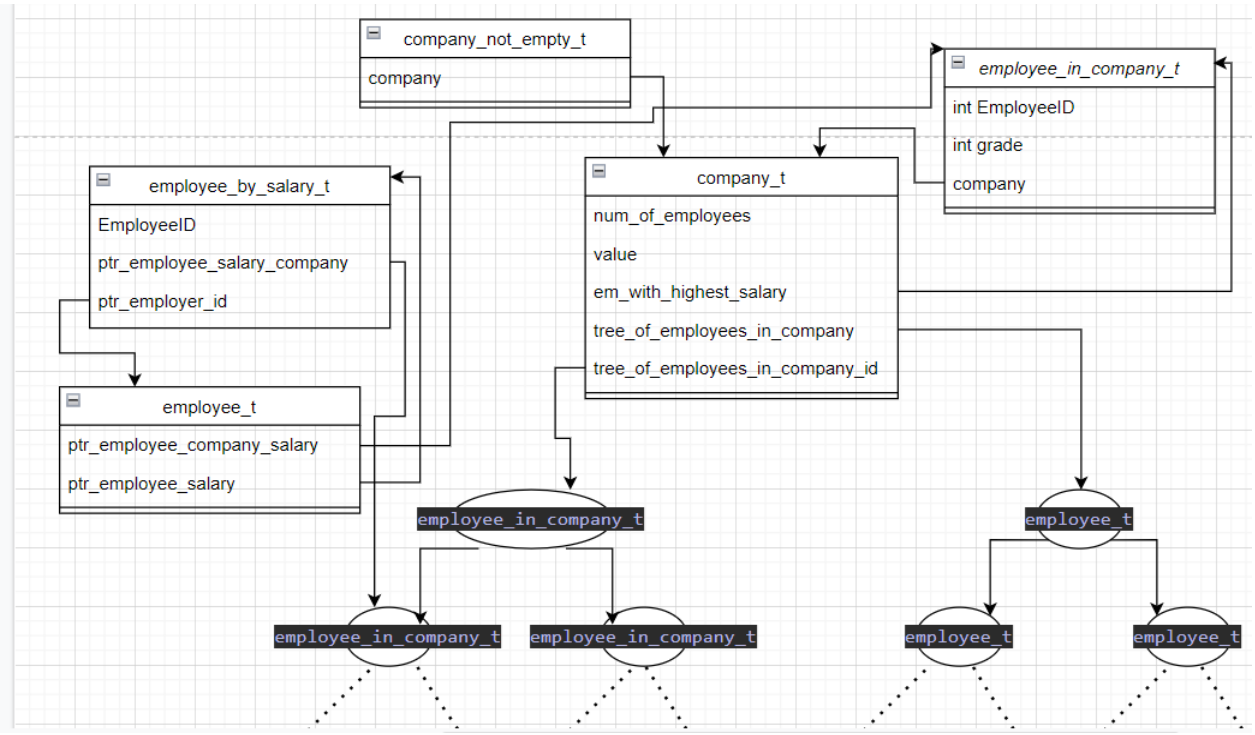
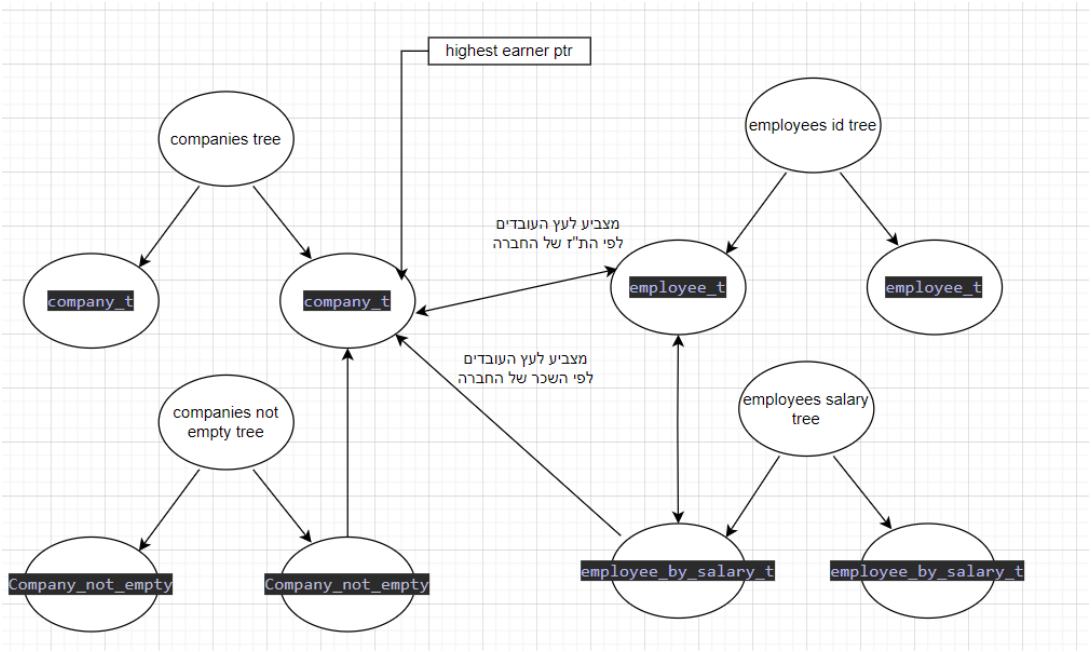
DRY OF WET 1

שרטוט של מבני נתונים שלנו:





Structures of the elements of the nodes in every tree:

Node for the tree of COMPANIES (the key is the ID of the company):

```
Stuct company_t {
    int num_of_employees;
    int value;
    Node<employee_in_company_t*>* em_with_highest_salary;
    Avl_tree<employee_in_company_t*>* tree_of_employees_in_company;
    Avl_tree<employee_t*>* tree_of_employees_in_company_id;
} *Company;
```

Node for the tree of EMPLOYEES_ID and EMPLOYEES_IN_COMPANY_ID (the key is the employee id):

```
Struct employee_t {
    Node<employee_in_company_t*>* ptr_employee_company_salary;
    Node<employee_by_salary_t*>* ptr_employee_salary;
} *Employee;
```

Node for the tree of EMPLOYEES_SALARY (the key is the salary of the employee):

```
struct  employee_by_salary_t {
    int EmployeeID;
    Node<employee_in_company_t*>*  ptr_employee_salary_company;
    Node<employee_t*>*  ptr_employer_id;
} Employee_by_salary* ;
```

Node for tree EMPLOYEE_IN_COMPANY (the key is the salary of the employee):

```
struct employee_in_company_t{
   int EmployeeID;
   int grade;
   Node<company_t*>* company ;
} Employee_in_company* ;
```

Node for tree COMPANY_NOT_EMPTY (the key is the company id):

```
struct company_not_empty_t {
   Node<company_t*>* company ;
} *Company_not_empty;
```

סיבוכיות מקום:

במהלך ריצת כל התוכנית, אנו שומרים במבנה הנתונים לכל חברה 2 עצי שחקנים אשר סיבוכיות המקום הכוללת עבור עצים אלו הינה: $2 \cdot O(n_1 + n_2 + \cdots + n_k) = O(n)$

בנוסף, 2 עצים כלליים המכילים את כל השחקנים - $O(2 \cdot n) = O(n)$

עצי החברות –$O(2 \cdot k)$

בנוסף, עבור כל אחת מהפונקציות, מקצים לכל היותר(קיימות פונקציות שכלל לא מקצים זיכרון נוסף התלוי בקלט) $O(n + k)$זיכרון נוסף שכן כל הקצאת זיכרון נוסף התלויה בקלט קורית באחד מהמקרים הבאים:

- הקצאת מערך בגודל n/k או יצירת עץ בעל לכל היותר O(n)/O(k) צמתים.
- רקורסיה בעצי AVL בגודל n/k צמתים. –$O(logk)/O(logn)$

כמו כן מספר ההקצאות המדוברות הוא מספר קבוע ולכן סך כל הזיכרון הנוסף הינו:

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

הפונקציות הנדרשות:

void* Init() :

- מאתחלים את מבנה הנתונים שלנו, כלומר אנו קוראים לבנאי שלCompanies_Managerעל ידי פעולת newשיוצר:
- עץ ריק EMPLOYEES_BY_SALARY, עץ ריק COMPANIES, עץ ריק EMPLOYEES_ID, עץ ריקCOMPANIES_NOT_EMPTY. נאתחל מצביע Highest_earner_in_DS לnull.
- במקרה של כישלון מחזירים NULL. במקרה של הצלחה נחזיר מצביע למבנה הנתונים שלנו

סיבוכיות: פונקציה זו דורשת מספר פעולות סופי ולכן סיבוכיות הזמן היא O(1) כנדרש.

StatusTypeAddCompany(void *DS, int CompanyID, int Value):

נוסיף את החברה החדשה למבנה שלנו בצורה הבאה

- נבדוק האם NULL==DS או אם CompanyID<=0 או Value<=0.במידה וכן, נחזיר את הערך INPUT_INVALID - .
- נחפש בעץ COMPANIESצומת בעל מפתחCompanyID. אם נמצא אותו,הפונקציה תחזיר FAILURE .
- אם לא קיים כזה, ניצור צומתCompanyחדש עם key ששווה לCompanyID, עם Value ששווה לValue של הפונקציה ועם מספר העובדים בעץ להיות 0 ונוסיף אותו לעץ COMPANIES– נבדוק הBF ונבצע גלגול במידה וצריך ונחזיר SUCCESS.

  סיבוכיות: לחפש בעץ AVL ולהוסיף צומת לעץ זה O(log k) כאשר k מספר הצמתים כלומר מספר החברות.

StatusTypeAddEmployee(void *DS, int EmployeeID, int CompanyID, int Salary, int Grade):

- נבדוק האם NULL==DS או אם CompanyID<=0 או EmployeeID<=0 או Salary<=0, או Grade<=0.במידה וכן, נחזיר את הערך INPUT_INVALID .
- נחפשבעץ EMPLOYEES_ID צומת בעל מפתח EmployeeID. אם נמצא אותהפונקציה תחזיר FAILURE . ונחפש בעץ COMPANIES צומת בעל מפתח CompanyID. אלא נמצא אותהפונקציה תחזירFAILURE.
- אחרת, נשווה את השכר של העובד החדש כלומר הפרמטר Salary עם השכר של העובד השמור בEmployee_with_highest_salary ונעדכן בהתאם את המצביע כלומר אם השכר של העובד החדש יותר גבוה (או שווה וID שלו יותר קטן) . נעדכן גם המצביע הכללי Highest_earner_in_DSלקבוצה החדשה לפי אותם תנאים.
- נוסיף את העובד החדש לעץ EMPLOYEES_IN_COMPANY לפי הSalary שלו ולעץ EMPLOYEES_IN_COMPANY_ID לפי הID שלו. נוסיף 1 לnum_employees ואם הוא שווה ל1 אז נוסיף את החברה לעץ COMPANIES_NOT_EMPTY והצומת החדש יצביע לצומת בCOMPANIES המתאים.
- נוסיף את העובד גם לעץ EMPLOYEES_BY_LEVEL לפי השכר של העובד וגם לעץ EMPLOYEE_ID לפי המזהה של העובד. ונחזיר ALLOCATION_ERROR במקרה של כישלון. נעדכן את המצביעים של כל צומת ונחזיר SUCCESS.

סיבוכיות: O(log k) כדי לחפש את החברה בעץ COMPANIES, וגם O(log k) כדי להוסיף לעץ COMPANIES_NOT_EMPTY במקרה וצריך – כי לכל היותר k חברות בעץ.O(log n) כדי לחפש/להוסיף לעץ של העובדים– וגם בתתי-עצים בחברה כי יש לכל היותר n עובדים בחברה- ולכן סהכ סיבוכיות O(log k + log n) כנדרש.

StatusTypeRemoveEmployee(void *DS, int EmployeeID):

- נבדוק האם NULL==DS או EmployeeID<=0. במידה וכן, נחזיר את הערך INPUT_INVALID .
- בעץ EMPLOYEES_ID נחפש את EmployeeID. אם לא נמצא נחזיר FAILURE.
- אחרת על ידי המצביע לצומת שלו בעץ EMPLOYEES_IN_COMPANY.
- על ידי המצביע לcompany נגיע לצומת שלהחברה ונבדוק האם זו החברה אליה מצביע Highest_earner_in_DS. אם כן בעץ EMPLOYEES_BY_SALARY נחפש העובד בעל השכר הכי גבוה אחרי העובד שמונסים למחוק – זה עץ לפי שכר ולכן סיבוכיות O(log n) - ונשים את הקבוצה אליה הוא שייך ב Highest_earner_in_DS על ידי המצביעים ב(O(1.
- בחברה אליה שייך העובד נבדוק אם המצביע Highest_earner_in_company מצביע אל העובד. אם כן נבצע חיפוש בעץ EMPLOYEES_IN_COMPANY של החברה הזו ונחפש את העובד בעל שכר הכי גבוה אחריו ונחליף המצביע אליו.העץ לפי Salary של העובדים ולכן זו סיבוכיות של O(log n). נמחוק את העובד מהעצים של החברה כלומר EMPLOYEES_IN_COMPANY וגם EMPLOYEES_IN_COMPANY_ID.
- נוריד 1 מnum_employeesבצומת של החברה ואם עכשיו הוא שווה ל 0, נחפש את החברה בעץ COMPANIES_NOT_EMPTYונמחוק את החברה מהעץ. בעץ הזה יש רק חברות שאינן ריקות לכן **קיים לפחות עובד אחד בכל חברה** ולכן יש לכל היותר n חברות כלומר סיבוכיות O(log n) כנדרש.
- נמחוק את העובד מעצים EMPLOYEES_ID וEMPLOYEES_BY_SALARY ונחזיר SUCCESS.

סיבוכיות: מספר קבוע של פעולות חיפוש/הסרה מעצים בעלי n צמתים לכל היותר בסיבוכיות O(log n) ולכן סיבוכיות O(log n).

StatusTypeRemoveCompany(void *DS, int CompanyID):

- נבדוק האם NULL==DS או אם CompanyID<=0 ונחזיר 0 אם כן.
- נחפש בעץ COMPANIESצומת בעל מפתח CompanyID. אם לא נמצא אותו,הפונקציה תחזיר FAILURE .
אחרת נבדוק אם num_employeesשווה ל0. אם לא אז יש עובדים בחברה והפונקציה תחזיר FAILURE.
- אחרת נמחוק את החברה מהעץ COMPANIES.

סיבוכיות: חיפוס/הסרה מעץ בעל k צמתים + מספר קבוע של פעולות זו סיבוכיות O(log k) כנדרש.

StatusTypeGetCompanyInfo(void *DS, int CompanyID, int *Value, int *NumEmployees):

- נבדוק האם NULL==DS או אם CompanyID<=0 או Value==NULL או אם NumEmployees=NULL. אם כן, נחזיר INVALID_INPUT.
- אחרת נחפש בעץ COMPANIESצומת בעל מפתח CompanyID. אם לא נמצא אותו,הפונקציה תחזיר FAILURE.
- אחרת נשים בValue את הValue של הCompany ששמרנו בצומת בעץ וב NumEmployeeנשמור את num_employees השמור בצומת גם ונחזיר SUCCESS.

סיבוכיות: חיפוס בעץ AVL עם k צמתים זו סיבוכיות O(log k).

StatusTypeGetEmployeeInfo(void *DS, int EmployeeID, int *EmployerID, int *Salary, int *Grade):

- נבדוק האם NULL==DS או אם EmployeeID<=0 או EmployerID == NULL או אם Salary==NULL או אם Grade==NULL. אם כן, נחזיר INVALID_INPUT.
- אחרת, בעץ EMPLOYEES_ID נחפש את EmployeeID. אם לא נמצא נחזיר FAILURE.
- אחרת, אחרת על ידי המצביע לצומת שלו בעץ EMPLOYEES_IN_COMPANYונשים את המשתנה Salary במצביע Salary של הפונקציה, והמשתנה Grade במצביע Grade של הפונקציה. כעט נגיש לחברה אליה שייך ונשים הkey של החברה במצביע EmployerID. ונחזיר SUCCESS.

סיבוכיות: חיפוס בעץ AVL עם n צמתים זו סיבוכיות O(log n).

StatusTypeIncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease):

- נבדוק האם NULL==DS או אם CompanyID<=0 או ValueIncrease<=0. אם כן, נחזיר INVALID_INPUT.
- אחרת נחפש בעץ COMPANIESצומת בעל מפתח CompanyID. אם לא נמצא אותו,הפונקציה תחזיר FAILURE.
- אחרת נוסיף ValueIncrease למשתנה Value.

סיבוכיות: חיפוש בעץ AVL של החברות עם k צמתים זו סיבוכיות O(log k).

StatusTypePromoteEmployee(void *DS, int EmployeeID, int SalaryIncrease, int BumpGrade):

- נבדוק האם NULL==DS או EmployeeID<=0 ואם SalaryIncrease<=0. במידה וכן, נחזיר את הערך INPUT_INVALID.
- בעץ EMPLOYEES_ID נחפש את EmployeeID. אם לא נמצא נחזיר FAILURE.
- אחרת על ידי המצביע ניגשלצומת שלו בעץ EMPLOYEES_IN_COMPANY. נשמור את הנתונים של הצומת. נמחוק את הצומת מהעץ ונוסיף אותו בחזרה עם אותם נתונים ושכר ששווה לשכר המקורי + SalaryIncrease. אם BumpGrade>0 נוסיף 1 לGrade של העובד.
- נבדוק האם השכר החדש שהתקבל גדול מזה השמור בHighest_earner_in_company. אם כן נחליף את המצביע אליו. ונבדוק גם עבור המצביע Highest_earner_In_DS כלומר אם השכר החדש גדול מזה של

העובד השמור במצביע– או אם השכר שווה וEmployerID של העובד הנוכחי קטן מזה של המצביע נחליף מצביע לחברה של העובד.

- נחליף מצביע אל הצומת החדש בצמתים של העץ EMPLOYEES_ID, וEMPLOYEES_BY_SALARY וגם EMPLOYEES_IN_COMPANY_ID.

נחזיר SUCCESS ונסיים.

סיבוכיות: חיפוש/הוספה/הסרה בעצים שמכילים n צמתים זו O(log n) כנדרש.

StatusTypeHireEmployee(void *DS, int EmployeeID, int NewCompanyID):

- נבדוק האם NULL==DS או EmployeeID<=0 ואם NewCompanyID<=0. במידה וכן, נחזיר את הערך INPUT_INVALID.
- בעץ COMPANIESנחפשצומת בעל מפתח NewCompanyID. אם לא נמצא אותו,הפונקציה תחזיר FAILURE.
בעץ EMPLOYEES_ID נחפש את EmployeeID. אם לא נמצא נחזיר FAILURE.
- אחרת על ידי המצביע ניגש לצומת של העובד בעץ ונחלץ משם את ערכי הsalary וgrade. לבסוף נסיר את העובד ממבנה הנתונים ע"י remove שהוזכר לעיל ונוסיף ע"י add.
סיבוכיות: מספר קבוע של פעולות + פעולת add + פעולת remove סה"כ כפי שהסברנו למעלה נקבל O(log n + log k) כנדרש.

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

- נבדוק האם NULL==DS או EmployeeID<=0 ואם NewCompanyID<=0. במידה וכן, נחזיר את הערך INPUT_INVALID.
- אחרת נחפש בעץ COMPANIESצומת בעל מפתח TargetID וצומת בעל AcquirerID. אם לא נמצא אחד מהם,הפונקציה תחזיר FAILURE .
- נבדוק שהValue של AcquirerID לפחות פי 10 גדול מהValue של TargetID.אם לא נחזיר FAILURE.
- אחרת, נבצע סיור Inorder2 העצים EMPLOYEES_IN_COMPANY של 2 החברות ונשים את הצמתים ב2 מערכים. נבצע merge לפי השכרים של העובדים בין המערכים ונקבל מערך ממוין לפי השכרים. ניצור עץ כמעט שלם בגודל המערך ונמלא אותו בסיור Inorder על ידי הצמתים במערך ממויין.
- נקבל עץ EMPLOYEES_IN_COMPANY עם כל העובדים של 2 החברות ונשים אותו כעץEMPLOYEES_IN_COMPANY של החברה AcquirerID.
- נבצע אותו דבר בין העצים EMPLOYEES_IN_COMPANY_ID אבל נבצע הmerge לפי הID של העובדים ונשים את העץ המתקבל כעץ EMPLOYEES_IN_COMPANY_ID של החברה AcquirerID.
- נבצע סכום בין 2 הValue של החברות ונכפיל בFactor ונשים התוצאה כValue של הAcquirerID. נסכום גם את מספר העובדים ונשים את התוצאה כnum_of_employees של AcquirerID.
- נמחוק את TargetID של העץ COMPANIE
- נחזיר הצלחה.

סיבוכיות: לחפש את החברות בעץ AVL זו סיבוכיות O(log k). לבצע סיור inorder בעצים זה $O(n_{AcquirerID} + n_{TargetID})$. לבצע Merge בין 2 המערכים הממוינים זה גם $O(n_{AcquirerID} + n_{TargetID})$. ליצור עץ כמעט שלם בגודל $n_{AcquirerID} + n_{TargetID}$ זה גם סיבוכיות $O(n_{AcquirerID} + n_{TargetID})$ כדי שנלמד בתרגול ולמלא אותו בסיור Inorder זה גם $O(n_{AcquirerID} + n_{TargetID})$. פעולות אריתמתיות זה O(1) ולכן סהכ סיבוכיות $O(\log k + n_{AcquirerID} + n_{TargetID})$.

StatusTypeGetHighestEarner(void *DS, int CompanyID, int *EmployeeID):

- נבדוק האם NULL==DS או EmployeeID==NULLאו אם CompanyID==0. במידה וכן, נחזיר את הערך INPUT_INVALID.
- אם CompanyID<0, נגיש למצביע Highest_earner_in_DS. אם NULL נחזיר FAILURE. אחרת,נגישלחברה אליה הוא מצביעובחברה זו נגיש לHighest_earner_in_company ונשים את הid של העובד אליו מצביע, במצביע של הפונקציהEmployeeID ונחזיר SUCCESS.
- אם CompanyID>0. נחפש את CompanyID בעץ COMPANIES. אם לא קיים נחזיר FAILURE. אחרת, נבדוק האם Highest_earner_in_company==NULL. אם כן נחזיר FAILURE. אחרת, נשים את הid של העובד במצביע של הפונקציהEmployeeID ונחזיר SUCCESS.

סיבוכיות: O(1) אם CompanyID<0 כי זה רק לגשת למצביע ולהחזיר מידע. (O(log k אם CompanyID>0 כי מחפשים בעץ AVL את החברה ב(O(log k ואז מגישים למצביע ומחזירים מידע ב(O(1.

StatusTypeGetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees):

ראשית נבצע בדיקות קלט.
אם $companyId > 0$: נבצע סיור למציאת הקבוצה הרצויה בעץ(אם לא קיימת או ריקה נחזיר שגיאה בהתאם).
וניגש לעץ השחקנים שלה הממוינים לפי שכר.
אחרת אם$companyId < 0$, ניגש ישירות לעץ השחקנים הכללי הממוין לפי שכר.
נקצה מערך בגודל מספר העובדים שבעץ הרצוי, וע"י סיור inorder נמקם את העובדים במערך. כפי שראינו בכיתה, סיור inorderיתן לנו את העובדים ממוינים במערך בסדר עולה.
נקצה מערך חדש באותו גודל ונעבור בלולאה ונמקם בסדר הפוך את העובדים.
כעת יש לנו במערך החדש את העובדים ממוינים בסדר יורד לפי השכר כנדרש.
סיבוכיות: אם $CompanyID < 0$: ביצוע מספר קבוע של פעולות פשוטות, סיור בעץ בגודל n, ולולאה למעבר על מערך בגודל n– סה"כ O(n) סיבוכיות זמן.

אם $CompanyID > 0$:סיור בעץ החברות -$O(logk)$. בנוסף סיור בעץ השחקנים של החברה ועל מערך בגודל מספר השחקנים שבחברה. סה"כ סיבוכיות מקום $O(logk + n_{company})$.

StatusTypeGetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees):

ראשית נבצע בדיקות קלט.
ניגש לעץ הקבוצות הלא ריקות ונבצע סיור inorderלמציאת מס החברות(NumOfCompanies)בעלות המזהה הנמוך ביותר עד שנגיע למספר החברות הרצוי ושם נעצור. במהלך הסיור, אנו ניגשים דרך המצביע של החברות בעלות לפחות עובד אחד, לעץ החברות הכללי, שם שמרנו מצביע נוסף אל העובד בעל השכר הגבוה ביותר בחברה. ניגש דרכו למזהה של העובד ונמקם אותו במערך הרצוי.
סיבוכיות: סיורinorder בעץ החברות בעלות לפחות עובד אחד –הגעה לצומת השמאלי ביותר בעץ מתבצעת לכל היותר ב $O(logk)$. משם נעבור על NumOfCompanies צמתים ולכן

סה"כ סיבוכיות $O(logk + \text{NumOfCompanies})$ כנדרש.

StatusTypeGetNumEmployeesMatching(void *DS, int CompanyID, intMinEmployeeID, int MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees) :

- נבדוק האם NULL==DS או TotalNumOfEmployees ==NULL או NumOfEmployees==NULL או אם CompanyID==0 או אם MinSalary<0 או אם MinGrade<0 או אם MinEmployeeID<0 או אם MaxEmployeeID<0 או אם MaxEmployeeID<MinEmployeeID. במידה וכן, נחזיר את הערך INPUT_INVALID.

נתאר אלגוריתם:

1. ניצור 2 counters: counter_total_employees וגם counter_em_matching.
2. ניצור משתנה first_employee מסוג Node<employee>.
3. אם MinEmployeeID==0 נחפש את העובד בעל ID הכי קטן <u>בעץ</u> ונשים אותו במשתנה first_employee. אחרת נחפש MinEmployeeID: אם לא מצאנו אותו אז נוסיף אותו <u>לעץ</u> עם מצביעים לnull ונחפש את הכי קטן אחריו ונשים אותו בfirst_employee. אחרת אם MinEmployeeID קיים בעץ אז נשים אותו כfirst_employee.
4. עכשיו נעבור inorder <u>בעץ</u> מהfirst_employee ועד שID של עובד בעץ גדול מMaxEmployeeID ונספור על כמה עובדים עוברים בסיור – כלומר עבור כל עובד עליו עוברים נוסיף 1 לcounter_total_employees. על ידי המצביע לEmployee_in_company בכל צומת, נבדוק עבור כל עובד עליו עוברים אם הGrade שלו גדול מMinGrade ואם הSalary שלו גדול מMinSalary ואם כן נוסיף 1 ל counter_em_matching.
5. TotalNumOfEmployees יצביע לcounter_total_employees וNumOfEmployees יצביע על counter_em_matching.
6. אם הוספנו MinEmployeeID לעץ אז נמחוק אותו.
7. נחזיר SUCCESS.

- אם CompanyID<0 ניגש לעץ EMPLOYEES_ID. במידה ואין עובדים בעץ נחזיר FAILURE. אחרת נבצע אלגוריתם שתיארנו על העץ EMPLOYEES_ID.
- אם CompanyID>0 נחפש אותה בעץ COMPANIES. אם לא קיים או אם num_employees=0 נחזיר FAILURE. אחרת ניגש לעץ EMPLOYEES_IN_COMPANY_ID ונבצע אלגוריתם הנ"ל.

סיבוכיות: אם CompanyID<0 אז חיפוש/הוספה אחת בעץ של העובדים זה O(log n) וסיור Inorder על TotalNumOfEmployees+1 זה O(TotalNumOfEmployees) כאשר עבור כל עובד עליו עוברים מבצעים פעולות ב(O(1– השוואות ופעולות אריתמטיות. ולכן סהכ סיבוכיות(O(log n + TotalNumOfEmployees כנדרש.

אם CompanyID>0 אז חיפוש בעץ של החברות זה O(log k) ואז חיפוש/הוספה אחת בעץ של העובדים שיש בו $n_{companyID}$ עובדים זה $O(\log n_{companyID})$. וסיור Inorder על TotalNumOfEmployees+1 זה O(TotalNumOfEmployees) כאשר עבור כל עובד עליו עוברים מבצעים פעולות ב(O(1– השוואות ופעולות אריתמטיות. ולכן סהכ סיבוכיות$O(\log k + \log n_{companyID} + TotalNumOfEmployees)$ כנדרש.

void Quit(void **DS):

ניגש לעץ החברות ונמחק מכל צומת בעץ את 2 עצי השחקנים שלה(לפי שכר ולפי id).
מחיקת העצים מתבצעת ע"י סיור Postorder שראינו בכיתה.
מתבצע בסיבוכיות: $2 \cdot O(n_1 + n_2 + \cdots + n_k) = O(n)$
בנוסף נמחק את עץ השחקנים הכללי לפי שכר ואת עץ השחקנים הכללי לפי id. - $O(2 \cdot n) = O(n)$

נמחק את עץ החברות הכללי ואת עץ החברות שאינן ריקות–$O(2 \cdot k)$

סה"כ **סיבוכיות**$O(n + k)$ $= O(n) + O(k) + O(2 \cdot k) \cdot 4$: כנדרש.