תיאור כללי של המבנה שלנו. ראשית נגדיר את המחלקות הבאות:

IDsalary מחלקה שתייצג דרך לזהות את העובדים כאשר נרצה לדרג אותם באמצעות משכורת קודם. זהו אובייקט בעל שתי מפתחות: מספר תעודת זהות ומשכורת. נדע להשוות בין שתי עובדים בעלי הפרטים האלה.

Employee: יאפיין עובד במערכת. לכל עובד יותאם מצביע משותף שיכיל את מזהה החברה אליה הוא שייך כרגע, המשכורת שלו, הדרגה שלו ומספר הזיהוי שלו.

Company- ייצג חברה כלשהי ויאופיין על ידי מזהה חברה ייחודי. כל אובייקט מהסוג הזה יכיל את השדות הבאים:

- מזהה חברה
- מספר העובדים העובדים בחברה כרגע
- עץ AVL של עובדים שיסודר לפי מספרי תעודות זהות של עובדים הקיימים בחברה
- עץ AVL של עובדים שיסודר לפי המשכורות של עובדים הרשומים במערכת ומספרי תעודת זהות כשובר שוויון. נעזר ב IDsalary כמפתח לעץ
- מצביע לעובד בעל המשתכר הגדול ביותר בחברה
- שווי החברה הנוכחי

המבנה Hitechs יכיל את השדות הבאים:

- מזהה חברה
- מספר העובדים רשומים במערכת
- עץ AVL של עובדים שיסודר לפי מספרי תעודות זהות של עובדים הרשומים במערכת
- עץ AVL של עובדים שיסודר לפי המשכורות של עובדים הרשומים במערכת ומספרי תעודת זהות כשובר שוויון. נעזר ב IDsalary כמפתח לעץ
- מצביע לעובד בעל המשתכר הגדול ביותר במערכת
- עץ AVL של חברות. יסודר על מספרי הזיהוי של החברה ושדה המידע יכיל מצביע משותף לאובייקט מסוג Company.
- עץ AVL של חברות שיש בהם לכל הפחות עובד פעיל אחד. יסודר על מספרי הזיהוי של החברה ושדה המידע יכיל מצביע משותף לאובייקט מסוג Company.
- מונה של מספר העובדים הרשומים במערכת
- מונה של המספר החברות שלהם יש עובד אחד לכל הפחות

Init()

נקצה מבנה של Hitches בשם hitechs, אשר ניתן לבצע זאת ב O(1) כך :

1) ניצור 4 עצי AVL ריקים כפי שראינו בהרצאה ב O(1).
2) נאתחל את כל 7 השדות של Hitches ב O(1) אשר בהתחלה המספרים מאותחלים לאפס ו highest_salary מאותחל לnullptr. ו נאתחל כל אחד אחד מהעצי AVL בעץ ריק שיצרנו למעלה.

ו בסוף נעשה cast ל *viod ונחזיר את GAME כנדרש ב O(1).

AddCompany(void* DS, int companyID)

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

קודם כל נחפש אם קיימת חברה בעלת מזהה companyID בתוך העץ : (DS->Companies)

אם קיימת כבר חברה בעלת מזהה זה נחזיר FAILURE. פעולת חיפוש זו לוקחת O(log(k)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL.

אם לא קיימת חברה כזו בעץ, נקצה חברה חדשה ו נוסיף אותה ל (DS-> Companies) פעולת הכנסה זו דורשת O(log(k)) כפי שראינו בהרצאה על עץ AVL. בנוסף ניצור 4 עצי AVL ריקים ב(O(1 ונגדיר את שאר השדות על פי הבנאי הדיפולטיבי שלהם. אם חלה בעיית זיכרון בהקצאת החברה החדשה נחזיר ALLOCATION_ERROR אחרת, נחזיר SUCCESS.

בסה"כ : ביצענו פעולת הוספת חברה ב O(log(k)) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נבדוק אם קיים שחקן בעל מזהה companyID בתוך העץ : (DS->employees_by_salary) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL.

נבדוק אם קיימת חברה בעלת מזהה companyID בתוך העץ : (DS->companies). פעולת חיפוש זו לוקחת O(log(k)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. אם קיימת החברה נסמנה : company_to_find.

אם : (לא קיימת חברה כזו) או (קיים שחקן כזה) נחזיר FAILURE.

אחרת : נקצה שחקן חדש שמאותחל בנתונים שקיבלנו מהפונקציה. אם חלה שגיאת זיכרון בניסיון ההקצאה נחזיר ALLOCATION_ERROR. אחרת, נוסיף השחקן לעצים הבאים :

1) (DS->employees_by_id) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הכנסה בעץ AVL.
2) (DS-> employees _by_salary) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הכנסה בעץ AVL.
3) (company_to_find-> employees_by_rank) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הכנסה בעץ. (כי יש בחברה מספר שחקנים קטן או שווה למספר השחקנים הכללי)
4) (company_to_find-> employees_by_id) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הכנסה בעץ. (כי יש בחברה מספר שחקנים קטן או שווה למספר השחקנים הכללי)

אחר כך, נבצע : ++(company_to_find->num_of_employees),++(DS->num_of_employees).

ונבדוק אם : (1 == (company_to_find)->num_of_employees)

אם כן : נוסיף את company_to_find ל (DS->companys_with_employees) פעולת הכנסה זו לוקחת O(log(k)) כפי שראינו בהרצאה על פעולת הכנסה בעץ AVL.

אחרת : לא נעשה כלום.

נחזיר SUCCESS בסוף.

בסה"כ : ביצענו הפעולה ב O(log(k) + log(n)) כנדרש.

StatusType RemoveCompany(void *DS, int CompanyID)
נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את החברה בעלת מזהה CompanyID בתוך העץ (DS->companies) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. ו נסמנה :company.

אם החברה לא קיימת ואם קיימת ויש בה עובדים פעילים אז נחזיר : FAILURE.

אחרכך, נמחק את company מהעץ הבא על ידי מספר הזיהוי שלה :

(DS->companies) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הוצאה מעץ AVL.

בסה"כ : ביצענו את הפעולות כולן ב O(log(n)) כנדרש.

RemoveEmployee(void *DS, int EmployeeID)

נבדוק שהנתונים שקיבלנו תקינים.אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את השחקן בעל מזהה EmployeeID בתוך העץ (DS->employees_by_id) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. ו נסמנו : Employee.

אם השחקן Employee לא קיים נחזיר : FAILURE.

אחרת: נחפש את החברה ש Employee שייך אליה בתוך העץ (DS->companies_with_employees) אשר נסמנה company. נקבל את המספר המזהה של החברה לצורך החיפוש מתוך (Employee->companyID) פעולת חיפוש זו דורשת O(log(n)) כי במקרה הגרוע ייתכן שיש n קבוצות שונות שמכילות n שחקנים שונים. ו משם ממשיכים עם פעולת החיפוש הרגילה שראינו בהרצאה.

אחרכך, נמחק את Employee מהעצים הבאים :

1) (DS->employees_by_id) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הוצאה מעץ AVL.
2) (DS->employees_by_salary) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הוצאה מעץ AVL.
3) (company->employees_by_rank) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הוצאה מעץ AVL.
4) (company->employees_by_id) פעולת זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת הוצאה מעץ AVL.

אחר כך, אם (Employees == company->highest_salary) נמצא את השחקן (החדש) בעל הדרגה הגבוהה ביותר (החדשה) אחרי שמחקנו את Employee מהעץ. ע"י כך שנלך ימין עד הסוף העץ AVL שזה דורש O(log(n)) ו נסמנו : highest_salary, ו נעדכן : (company->highest_salary = highest_salary).

ונבדוק גם, אם (DS-> highest_salary == Employee). אם כן : נמצא את השחקן בעל הדרגה הגבוהה ביותר (החדשה) כמו שעשינו למעלה, ונסמנו new_highest_salary,ו נעדכן :

(DS-> highest_salary = new_ highest_salary).

הפעולה באותה סיבוכיות כמו שמוסבר למעלה מאותם הסברים.

ונבצע : --(company->num_of_employees), --(DS->num_of_employees)

אם : (company->num_of_employees <= 0) נוציא את החברה company מעץ (DS-> companies_with_employees) כאשר פעולת הוצאה זו דורשת O(log(n)) כפי שהסברנו למעלה על הקשר בין מספר הקבוצות ו מספר השחקנים, וראינו בהרצאה על פעולת הוצאה בעץ AVL. בסוף נחזיר SUCCESS.

בסה"כ : ביצענו את הפעולות כולן ב O(log(n)) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את החברה בעלת מזהה CompanyID בתוך העץ (DS->companies) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. ו נסמנה company:.

אם החברה לא קיימת: FAILURE.

ניגש למידע על החברה כפי ששמור בצומת הרלוונטי לחברה ונעתיק למצביעים הרלוונטיים

בסה"כ : ביצענו את הפעולות כולן ב O(log(n)) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש עובד בעל מזהה EmployeeID בתוך העץ (DS-> employees_by_id) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. ו נסמנו employee: .

אם העובד לא קיימת: FAILURE.

ניגש למידע על החברה כפי ששמור בצומת הרלוונטי לעובד ונעתיק למצביעים הרלוונטיים

בסה"כ : ביצענו את הפעולות כולן ב O(log(n)) כנדרש.

StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease)
נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את החברה בעלת מזהה CompanyID בתוך העץ (DS->companies) פעולת חיפוש זו לוקחת O(log(n)) כפי שראינו בהרצאה על פעולת חיפוש בעץ AVL. ו נסמנה company:.

אם החברה לא קיימת: FAILURE.

ניגש למידע על החברה כפי ששמור בצומת הרלוונטי לחברה ונעדכן את שדה שווי החברה בהתאם להוראות.

בסה"כ : ביצענו את הפעולות כולן ב O(log(n)) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את העובד בעל מזהה EmployeeID בעץ (DS->employees_by_id) אשר פעולת חיפוש זה תדרוש O(log(n)) כפי שראינו בהרצאה. אם לא נמצא אותו, נחזיר :FAILURE .
אם נמצא אותו, נסמנו : Employee ו נסמן :
new_salary = (Employee->employee_salary) + SalaryIIncrease
אחר כך: נמצא את החברה של Employee שנקבל את מספר מ בעץ
(DS->companies_with_employees) אשר פעולת חיפוש זו תדרוש O(log(n)) כי יש לכל היותר n קבוצות שונות שיש בכל אחת שחקנים.
אחר כך: נוציא את Employee מהעץ (DS->employees_by_id) ו מהעץ
(DS->employees_by_salary) אשר פעולת הוצאה זו תדרוש O(log(n) כפי שראינו בהרצאה.
אחר כך, נבצע הפעולה : (Employee->employee_salary = newsalary).
נוסיף מחדש את Employee לעץ (DS->employees_by_salary) ולעץ DS->employees_by_id אשר פעולת הכנסה זו תדרוש O(log(n)) כפי שראינו בהרצאה. נבדוק אם Employee הפך להיות ה highest_salary של החברה ו נעדכן בהתאם.

נעשה אותו הדבר בדיוק עבור החברה שבה העובד רשום. נבדוק אם Employee הפך להיות ה highest_salary של DS ו נעדכן בהתאם.
אם חלה שגיאת זיכרון, תיזרק שגיאת bad_alloc אשר נתפוס אותה ו נחזיר ALLOCATION_ERROR. אחרת נחזיר SUCCESS.
בסה"כ : ביצענו את הפעולות ב O(log(n)) כנדרש.

StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID)
נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נמצא את העובד בעץ DS->employees_by_id . נקרא לפונקציית מחיקת עובד כפי שתוארה קודם עם המזהה של העובד ולאחר מכן נוסיף אותו מחדש לחברה בעלת המזהה המתאים. פעולת המציאה המחיקה וההוספה מחדש תיקח לנו O(log(n)). לכן בסה"כ : ביצענו את הפעולות ב O(log(n)) כנדרש.

GetHighestEarner(void *DS, int CompanyID, int *EmployeeID)
נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

נחפש את החברה בעץ (DS->companies_with_employees). אם אכן החברה קיימת ויש בה עובדים אז היא בוודאות תופיע בעץ הזה. אם לא נמצא אותו בעץ נחזיר שגיאה מסוג FAILURE . אם מצאנו את החברה אז ניגש לשדה המרוויח הגדול ביותר בחברה ונחזיר את המזהה של העובד הזה.
AcquireCompany(void *DS, int AcquirerID, int TargetID, double factor)

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

קודם כל : נחפש את הקבוצות בעלות המזהה CompanyID,ReplacementID בתוך העץ (DS->companys) ו נסמנם: company_to_remove ו replacement_company בהתאם לשמות כמובן. אשר פעולת חיפוש זו דורשת O(log(k)) כפי שראינו על פעולת חיפוש בעץ AVL בהרצאה.
אם אחת מהקבוצות לא קיימות בעץ נחזיר : FAILURE.
אחר כך, אם : (company_to_remove->num_of_employees > 0) נוציא את company_to_remove מתוך העץ (DS->companys_with_employees) כאשר זה דורש O(log(k)) כפי שראינו על פעולת הוצאה בעץ AVL בהרצאה.
ו נבצע הפעולה : --(DS->num_of_companies_with_employees)
אחרכך :

1) נוציא את איברי העץ (company_to_remove->employees_by_rank) למערך בעזרת סיור INORDER על העץ אשר זה דורש O(n- Acquirer).
2) נוציא את איברי העץ (replacement_company-> employees_by_rank) למערך בעזרת סיור INORDER על העץ אשר זה דורש O(n- Acquirer).
3) נמזג את המערכים למערך אחד בעזרת אלגוריתם merge שראינו במיון mergesort אשר זה ידרוש ממנו O(n- Acquirer + n- Target).
4) נעבור על המערך הממוזג ו במקרה הצורך נעדכן את ה companyID של כל שחקן במערך להיות של ReplacementID, אשר זה ידרוש ממנו O(n- Acquirer + n- Target) .
5) נבנה עץ כמעט שלם בשם new_tree עם (n- Acquirer + n- Target) nodes כפי שראינו בתרגול ב O(n- Acquirer + n-Replacement).
6) נעבור על new_tree בסיור INORDER ו נמלא בו את איברי המערך הממוזג, אשר זה ידרוש ממנו סיבוכיות O(n- Acquirer + n-Target).
7) נמחק את העץ הנמצא ב (replacement_company-> employees_by_rank) ו נשים במקומו את העץ new_tree.

8) נחזור על צעדים 1-7 וכעת העצים הרלוונטי יהיו company_to_remove-> employees_by_id ו replacement_company -> employees_by_id בהתאמה.
9) נבצע: (replacement_company->num_of_employees)+=(company_to_remove->num_of_employees).

אם חלה שגיאת זיכרון במקרים למעלה, תיזרק שגיאת bad_alloc אשר נתפוס אותה ו נחזיר ALLOCATION_ERROR.
אחרת נחזיר SUCCESS.
בסה"כ : ביצענו הפעולות ב O(n- Acquirer + n- Target) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים.אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

אם : (CompanyID < 0)
אם : (DS->num_of_employees == 0) אז : (*EmployeeID) = (-1).
אחרת : (*EmployeeID) = (DS->num_of_employees).
ו נחזיר SUCCESS.
אחרת :
נחפש חברה בעלת מזהה CompanyID בתוך (DS->companies) ו נסמנו company. אשר פעולת חיפוש זו תדרוש O(log(k)) בעץ AVL כפי שראינו בהרצאה. אם החברה לא קיימת, נחזיר FAILURE.
אם : (company->num_of_employees == 0) אז : (*EmployeeID) = (-1).
אם :(company->num_of_employees > 0) אז : (*EmployeeID) = (company->num_of_employees).
ו נחזיר SUCCESS.
בסה"כ : כאשר (CompanyID < 0) אנו מבצעים הפעולות ב O(1).אחרת, מבצעים הפעולות ב O(log(k)) כנדרש.

GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *numOfEmployees)

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

אם : (CompanyID < 0) ו גם (DS->num_of_employees) > 0) נקצה מערך בגודל (DS->num_of_employees) בתוך : (*Employees), אם יש שגיאת זיכרון נחזיר ALLOCATION_ERROR. ו נבצע את האלגוריתם המופיע ב **.
אם : (CompanyID > 0) נחפש חברה בעלת מזהה זו בעץ (DS->companies) אשר נסמנה company ו פעולה זו דורשת O(log(k)) כפי שראינו על פעולת חיפוש בעץ AVL. אם חברה זו לא קיימת, נחזיר FAILURE.
אם (company->num_of_employees) > 0)נקצה מערך בגודל (company->num_of_employees) בתוך : (*Employees), אם יש שגיאת זיכרון נחזיר ALLOCATION_ERROR. ו נבצע את האלגוריתם המופיע ב **.

<u>אלגוריתם המשותף בשתי המקרים:</u>**
אם (DS->num_of_employees == 0) או (company->num_of_ employees == 0) בהתאמה, אז : (*numOfEmployees = 0), (*Employees = NULL)
אחרת : נעבור על העץ המתאים (DS-> employees _by_salary) או (company-> employees _by_rank) בסיור INORDER בסדר יורד ונמלא את המערך שהקצינו שם. אשר זה יקח O(n) במקרה של (CompanyID < 0) ו O(n- Company) אחרת.
בסה"כ : ביצענו הפעולות ב O(n) כאשר (CompanyID < 0) ו ב O(log(k) + n-Group) אחרת כנדרש.

GetCompaniesHighestEarner(void *DS, int numOfCompanies, int **Employees)

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

אם : (DS->num_of_companies_with_employees < numOfCompanies) נחזיר FAILURE.
נקצה מערך בגודל numOfCompanies בתוך (Employees*) אם חלה שגיאת זיכרון נחזיר ALLOCATION_ERROR.
נבצע סיור INORDER על (DS->companies_with_employees) ו נשים במערך שהקצנו את המידע של ה highest_salary של הקבוצות כאשר נתחזק counter שיספור על כמה קבוצות עברנו בסיור אחרי שהגענו לחברה בעלת המזהה הכי קטן. אשר זה(להגיע להכי שמאלי) דורש O(log(k)) בעץ AVL. ו בעת שה counter יגיע ל numOfCompanies נפסיק את הסיור ו נחזיר SUCCESS.
בסה"כ : אנחנו מבצעים הפעולות ב O(log(k) + numOfCompanies) כנדרש.

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

נבדוק שהנתונים שקיבלנו תקינים. אחרת, נחזיר שגיאה מתאימה מסוג INVALID_INPUT.

אם : (CompanyID < 0) ניגש לעץ (DS->employees_by_id) > 0) ונלך שמאלה עד אשר נצא מהטווח התחתון. נשמור את הצומת הראשונה אשר נמצאת מחוץ לטווח (ייתכן וכל הצמתים בתוך הטווח ואז נשמור מצביע לnull. נעשה אותו הדבר עבור הצד השני ונשמור גם את המצביע הזה באפן דומה. נמשיך את האלגוריתם ב**
אם : (CompanyID > 0) נחפש חברה בעלת מזהה זו בעץ (DS->companies) אשר נסמנה company ו פעולה זו דורשת O(log(k)) כפי שראינו על פעולת חיפוש בעץ AVL. אם חברה זו לא קיימת, נחזיר FAILURE.
ניגש לעץ השחקנים המתאים לחברה (company->employees_by_id) > 0) ונלך שמאלה עד אשר נצא מהטווח התחתון. נשמור את הצומת הראשונה אשר נמצאת מחוץ לטווח (ייתכן וכל הצמתים בתוך הטווח ואז נשמור מצביע לnull).
נעשה את אותו הדבר בכך שנלך עד הסוף ימינה ונשמור גם את המצביע הזה באפן דומה. נמשיך האלגוריתם ב**.

המשך האלגוריתם **
במקרה בו המצביע שנשמר הוא null אז לא נעשה את החלק הזה. נספרו את כמות הצמתים המתאימים לטווח בתת העץ של הצומת הראשונה שנשמרה. נחלק למקרים זרים:

1. המפתח של הצומת נמצא בדיוק בטווח בין MinEmployeeID ו MaxEmployeeId. במצב הזה נוסיף 1 לTotalNumOfEmployees ונבדוק האם המשכורת והדרגה שלו גדולים מMinSalary MinGrade בהתאמה ונוסיף 1 ל NumOfEmployees בעת הצורך. כעת ייתכן ששתי הבנים שלו נמצאים גם בטווחים אז נבדוק גם אותם באופן רקורסיבי.
2. המפתח של הצומת נמצא מתחת ל MinEmployeeID. במצב הזה ייתכן ואחד הצאצאים בתת העץ הימני שלו כן נמצא בטווח. לכן נבדוק רקורסיבית את תת העץ הימני שלו.
3. המפתח של הצומת נמצא מעל ל MaxEmployeeId. במצב הזה ייתכן ואחד הצאצאים בתת העץ השמאלי שלו כן נמצא בטווח. לכן נבדוק רקורסיבית את תת העץ השמאלי שלו.

נעשה את אותו הדבר עבור הצומת השנייה ששמרנו.
נחשב את כמות הצמתים שבהם נעבור באלגוריתם הזה: נסתכל על מסלול חיפוש אפשרי כלשהו אחר צמתים רלוונטיים ונחלק ל2 מקרים אפשריים:

1. אם המפתח של הצומת נמצא בטווח אז נבדוק את 2 הילדים של הצומת אם הם נמצאים בטווח (נמשיך משם בעת הצורך). במקרה הכי גרוע 2 הילדים לא נמצאים בטווח ונאלץ לחזור ולכן נעשה מספר סופי של בדיקות. לכן יהיו בסך הכל עבור המקרים האלה O(TotalNumOfEmployees) בדיקות כאלה.

2. אם המפתח של הצומת הנוכחית לא נמצא בטווח אז בכל מקרה נמשיך לחפש רק בכיוון 1 (ימין או שמאל). נמשיך זאת עד שנמצא צומת כלשהי בטווח ואז נכנס למקרה 1. במקרה הכי גרוע נמשיך עד שנגיע לעלה הימני ביותר בתת העץ כלומר נבצע O(log(n-company)) בדיקות של צמתים שאינם בטווח.

כעת נספור את כל שאר הצמתים שאינם נמצאים ב2 תתי העצים שמצאנו קודם לכן. אנחנו יודעים בוודאות לפי תכונות עץ חיפוש בינארי שכולם נמצאים בטווח ולכן יספרו. נעשה חיפוש inorder כאשר השוני היחיד יהיה שברגע שנגיע לאחת הצמתים ששמרנו קודם, נחזור לאבא ונמשיך את החיפוש משם. לכן כל החיפוש הזה יקח לנו O(TotalNumOfEmployees).

סיבוכיות: עבור (CompanyID > 0) , חיפוש של החברה בעץ חיפוש בינארי O(logk) . משם נעבור לחיפוש בעץ החיפוש company->employees_by_id . החיפוש בתתי העצים יקח לנו
O(log(n-company)+ TotalNumOfEmployees).
והספירה בכל העץ שאינו שייך לתת העצים יקח לנוO(TotalNumOfEmployees) לכן בסך הכל, כל החיפוש יקח לנו O(log(n-company)+ TotalNumOfEmployees+logk).

עבור (CompanyID < 0) , נעבור ישר לחיפוש בעץ חיפוש DS->employees_by_id. החיפוש בתתי העצים יקח לנו O(logn+ TotalNumOfEmployees).
והספירה בכל העץ שאינו שייך לתת העצים יקח לנוO(TotalNumOfEmployees) לכן בסך הכל, כל החיפוש יקח לנו O(logn+ TotalNumOfEmployees).
לכן בבכל מקרה אנחנו נעמוד בסיבוכיות.

Quit(void** DS)

נעשה המרה ל (DS*) ל (SquidGame*) ו נשתמש ב delete כדי למחוק אותו (כמו שהשתמשנו ב new בתוך Init) ו נשים : (*DS) = NULL
סיבוכיות זמן: לכל חברה משויכת צומת כלשהי בעץ החברות ועץ מתאים (ואם יש בה עובדים אז צומת נוספת בעץ החברות עם עובדים). ולכל עובד משויכים סך הכל 4 צמתים בכל העצים (2 בעץ של החברה המתאימה ועוד 2 במבנה הכללי). תחת ההנחה ששחרור המידע על כל צומת הוא בזמן ריצה סופי, זמן הריצה הכולל של הפונקציה יהיה O(n+k).

סיבוכיות מקום: מאותם נימוקים כמו למעלה, לכל עובד ולכל חברה יש גודל חסום של מידע שמשויך לה ולכן סיבוכיות המקום תהיה O(n+k)