חלק יבש:

כמבנה נתונים מימשנו עץ AVL. העץ שמימשנו מאפשר את הפעולות הבאות:

הכנסה לעץ והסרה ממנו. זאת כמובן מבלי לפגוע בסדר הפנימי של העץ ולכן מומשו גלגולים כפי שנלמד בהרצאה.

סיור inOrder שמחזיר מערך של הצמתים בעץ, סיור inOrder שמחזיר את הdata השמורה בצמתים כמערך, סיור inOrder שמוגבל במספר צעדים, וסיור inOrder שמוגדרת לו נקודת התחלה ונקודת סיום.

merge לאיחוד שני עצים.

פונקציית מציאה של הצומת עם data נתונה, ופונקציית מציאת הצומת המינימלי.

העץ מומש עם data השמורה בshared_ptr וכן הצמתים שמורים בshared_ptr, לצורך שמירת צמתים גנריים.

מבנה הנתונים הלא גנרי שלנו, System, יאפשר מימוש כלל הפונקציות הנדרשות בתרגיל. יכיל את המידע הבא, תרשים מצורף לאחר הפירוט להלן:

1. AllCompaniesTree - עץ AVL שמכיל את כל החברות במערכת (Company).
2. NonEmptyCompaniesTree - עץ AVL שמכיל את החברות (Company) הלא ריקות במערכת משמע מעסיקות לפחות עובד אחד. הערה חשובה לגבי עץ החברות הלא ריקות- עץ החברות הלא ריקות יכול להכיל לכל היותר n איברים (חברות), כאשר n הוא מספרם הכולל של העובדים הרשומים במערכת. זאת מכיוון שבמקרה קצה ייתכן שבדיוק עובד אחד מועסק בכל חברה. כמו כן לכל היותר יכיל k איברים, כאשר k הוא מספר החברות, מפני שלא יתכנו יותר חברות לא ריקות מחברות בכלל. לכן חיפוש בעץ זה הוא לכל היותר בסיבוכיות המינימלית מבין log(n) ו.log(k)
3. id_highest_earner- מזהה העובד עם השכר הגבוה ביותר בכל המערכת מבין כלל העובדים במערכת
4. salary_of_highest_earner - ערך המשכורת של העובד עם השכר הגבוה ביותר במערכת.
5. num_all_employees - מספר העובדים הכולל בכל המערכת.
6. **עצים שיכילו מידע על כל עובדי המערכת:**
   - *AllEmployeesByIDTree*– עץ AVL שבתוכו שמורים העובדים במערכת לפי מפתח מזהה העובד (*EmployeeByID*).
   - *AllEmployeesBySalaryTree* - עץ AVL שבתוכו שמורים העובדים במערכת לפי מפתח משכורתם (*EmployeeBySalary*).
7. **עצים שיכילו מידע על עובדי חברה מסוימת והגישה אליהם היא מהחברה:**
   - *EmployeesByIDTree* - עץ AVL שבתוכו שמורים העובדים במערכת לפי מפתח מזהה העובד (*EmployeeByID*).
   - *EmployeesBySalary* - עץ AVL שבתוכו שמורים העובדים במערכת לפי מפתח משכורתם (*EmployeeBySalary*).
8. **פירוט הצמתים בעצים שיכילו מידע על העובדים:**

- *EmployeeByID*– צומת בעץ AllEmployeesByIDTree או בעץ של חברה EmployeesByIDTree
  - מפתח מספר מזהה של העובד.
  - מצביע ל- *EmployeeSlaiary* בעץ לפי משכורות המתאים.
  - מצביע ל- company החברה בה מועסק העובד. מצביע זה יאותחל רק בצומת בעץ כל העובדים במערכת.
- *EmployeeBySalary* - צומת בעץ AllEmployeeBySalaryTree או בעץ של חברה EmployeesBySalaryTree
  - מפתח משכורת העובד.
  - מזהה העובד.

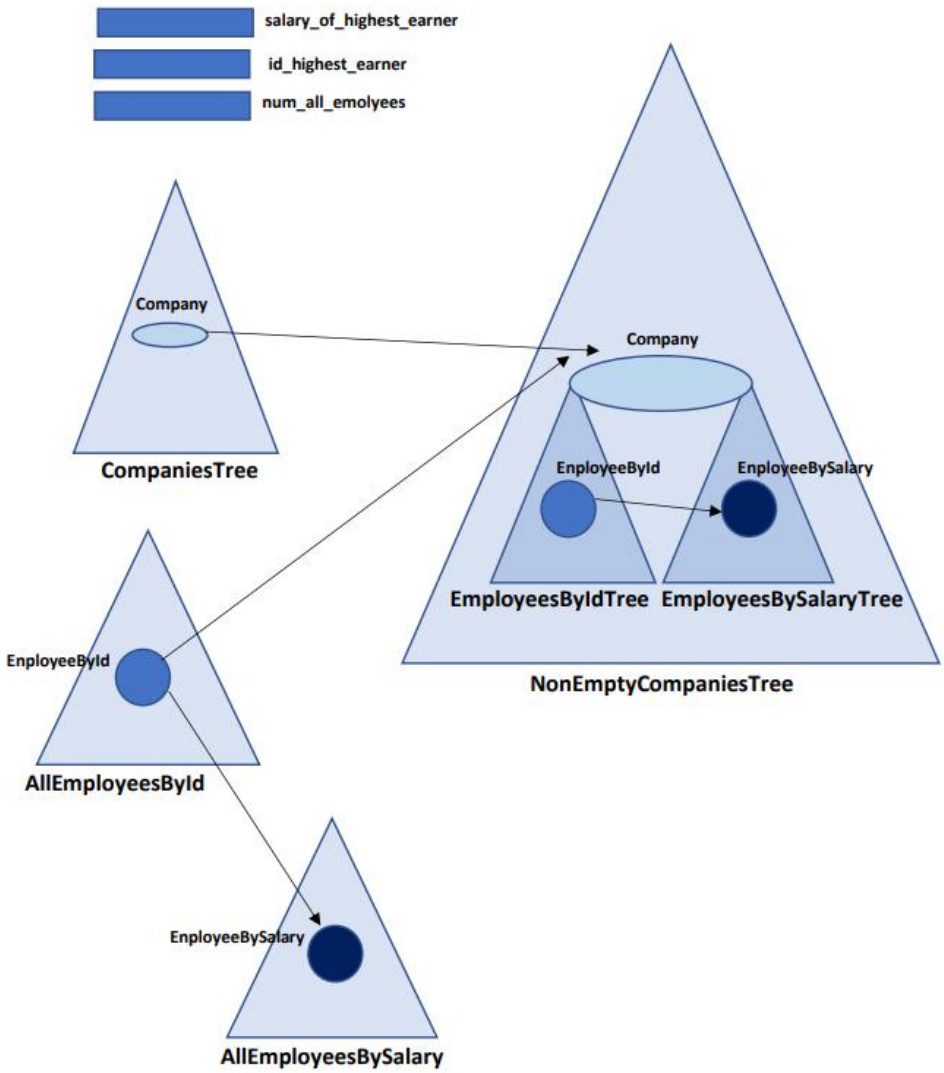
9. **פירוט צומת בעץ AllCompanies**

- company_id - מספר מזהה של החברה.
- value - ערך של החברה.
- *ptr_non_empty_company* מצביע לצומת החברה הלא ריקה בעץ החברות הלא ריקות.

10. **פירוט צומת בעץ NonEmptyCompanies**

- company_id - מספר מזהה של החברה.
- id_highest_earner - מספר מזהה של העובד עם השכר הגבוה ביותר בחברה.
- salary_of_highest_earner - ערך המשכורת של העובד עם השכר הגבוה ביותר בחברה.
- num_employees- מספר העובדים בחברה
- *ptr_employees_by_id* - מצביע לעץ *EmployeesByIDTree* שמכיל את כל עובדי החברה מסודרים לפי מספר מזהה.
- *ptr_employees_by_salary* מצביע לעץ *EmployeesBySalary* שמכיל את כל עובדי החברה מסודרים לפי משכורתם.

<u>סקיצת מבנה הנתונים</u>



<u>מימושים לפונקציות:</u>

1. **void* init()**

- ניצור מבנה נתונים חדש, יצירת העצים הריקים במערכת הכללית. יצירת מבנה זה בזמן קבוע. <u>לכן, סיבוכיות פונקציה זו O(1).</u>

2. **int value) ,int CompanyID ,StatusType AddCompany(void *DS**

- נחפש בעץ AllCompanies האם החברה קיימת אם כן, נחזיר FAILURE. סיבוכיות חיפוש בעץ בגודל k היא log(k).
- ניצור חברה *Company* ואתחול כל השדות. סיבוכיות אתחול משתנים O(1).נאתחל עצי העובדים בחברה לפי מזהה ושכר כעצים ריקים סיבוכיות האתחול בזמן קבוע O(1).
- נכניס את החברה לעץ AllCompaniesTree. סיבוכיות זמן של הכנסת איבר לעץ המכיל k חברות היא log(k).
- <u>ולכן, סיבוכיות פונקציה זו היא O(log(k)).</u>

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

- נחפש בעץ AllEmployeesByID האם העובד קיים. אם כן, נחזיר FAILURE. סיבוכיות חיפוש בעץ בגודל n היא log(n).
- נחפש בעץ AllCompaniesTree האם החברה קיימת. אם לא, נחזיר FAILURE. סיבוכיות חיפוש בעץ בגודל k היא log(k).
- נשתמש בפוינטר השמור בחברה לחברה בעץ NonEmptyCompaniesTree אם אין עדיין עובדים בחברה, נוסיף את הCompany לNonEmptyCompaniesTree סיבוכיות הכנסה זו כפי שהוסבר בפתיח לכל היותר log(k)
- ניצור את מבני הנתונים *EmployeeByID, EmplyeeBySalary, סיבוכיות אתחול O(1)*. נוסיף לעצים המתאימים עבור המערכת בעצי כל העובדים והחברה company בעץ NonEmptyCompaniesTree, סיבוכיות הוספה לעצים לכל היותר log(n).
- נבדוק האם לעובד החדש יש את השכר הגבוה ביותר בחברה/במערכת. בחברה, נשווה את שכר העובד החדש לשדה Salary_of_highest_earner, אם גבוה יותר נעדכן את Salary_of_highest_earner וכן את id_highest_earner. באופן שקול עבור המערכת. נעדכן את מספר העובדים הכולל במערכת. סיבוכיות O(1).
- <u>לכן הסיבוכיות המתאימה לפעולות היא O(log(k))+(log(n))</u>

4. **int EmployeeID) ,StatusType RemoveEmployee (void *DS**

- נחפש בעץ AllEmployeesByID האם העובד קיים. אם לא, נחזיר FAILURE. סיבוכיות החיפוש בעץ בגודל n היא log(n)
- נשתמש במצביע לחברה השמור בעובד, נפחית באחד בחברה את מספר העובדים. סיבוכיות O(1).
- נחפש בחברה בעץ EmployeesByIdTree את העובד למחיקה, סיבוכיות חיפוש לכל היותר O(log(n)).
- נבדוק האם המזהה של העובד זהה למזהה השמור בחברה ב id_highest_earner. אם כן, נשתמש במצביע של העובד ל EmployeesBySalaryTree, המרוויח ביותר אחריו. עדכון המרוויח ביותר גם עבור המערכת כולה במידת הצורך. נעדכן את מספר העובדים הכולל במערכת.
- נמחק את העובד מכלל העצים במערכת העצים סיבוכיות מחיקה מעץ מאוזן בגודל n עובדים לכל היותר היא log(n). מחיקת העובד מעצי העובדים בחברה תעמוד בסיבוכיות הדרושה כי אנו שומרים בעץ העובדים הכולל פוינטר לחברה בה מועסק ולכן מציאת החברה בה מועסק תהיה בסיבוכיות O(1).
- לכן <u>הסיבוכיות הגרועה ביותר היא O(log(n))</u>

5. **int CompanyID) ,StatusType RemoveCompanies(void *DS**

- נחפש בעץ CompaniesTree האם החברה קיימת. אם לא, נחזיר FAILURE. סיבוכיות חיפוש בעץ החברות O(log(k)).

- נבדוק האם בחברה יש עובדים בעזרת פוינטר לעץ החברות הלא ריקות. אם אינה ריקה, לא ניתן למחוק ונחזיר FAILURE.
- מחיקת החברה מעץ CompaniesTree. סיבוכיות מחיקת חברה מעץ בגודל k היא O(log(k)).
- <u>ולכן סיבוכיות פונקציה זו היא O(log(k)).</u>

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

- נחפש בעץ CompaniesTree את החברה. אם לא קיימת, נחזיר FAILURE. סיבוכיות זמן חיפוש בעץ מאוזן O(log(k)).
- ניגש לשדה num_Employees ונעדכן את ערך הפוינטר NumEmployees. סיבוכיות גישה לשדה O(1).
- ניגש לשדה value ונעדכן את ערך הפוינטר Value. סיבוכיות גישה לשדה O(1)
- <u>ולכן סיבוכיות פונקציה זו היא O(log(k)).</u>

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

- נחפש בעץ AllEmployeesById את העובד. אם לא קיים, נחזיר FAILURE. סיבוכיות זמן חיפוש בעץ מאוזן O(log(n)).
- ניגש למצביע ל- *EmployeeSalary* בעץ All*EmployeesSalary* *של העובד הנתון. נעדכן בפוינטר* Salary את שכרו.
- ניגש לשדה דרגתו של העובד ונעדכן בפוינטר Grade את דרגתו.
- ניגש למצביע ל- Company בעץ NonEmptyCompanies, בתוך החברה לשדה company_id ונעדכן את הפוינטר EmployeeID.
- סיבוכיות גישה לשדות דרך פוינטרים היא O(1).
- <u>ולכן סיבוכיות פונקציה זו היא O(log(n)).</u>

8. StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease)

- נחפש את החברה בעץ AllCompsnies, אם החברה לא קיימת נחזיר FAILURE. סיבוכיות חיפוש בעץ מאוזן היא O(log(k)).
- נעדכן בחברה את השדה value לערכו הקודם בתוספת ValueIncrease.
- <u>לכן, סיבוכיות פונקציה זו היא O(log(k)).</u>

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

- נחפש בעץ AllEmployeesById האם העובד קיים. אם לא, נחזיר FAILURE. סיבוכיות חיפוש O(log(n)).
- נשתמש בgetEmployeeInfo ונשמור את פרטי העובד. כמו כן, נשמור את הפוינטר לחברה בה העובד מועסק ששמור בצומת העובד לפי מזהה בעץ כל העובדים. O(log(n)). נשמור את המצביע לעובד בעץ העובדים לפי שכר.
- נמחק את העובד מעץ העובדים לפי שכר הכללי, ומעץ העובדים לפי שכר בתוך החברה עצמה.
- ניצור עובד זהה עם המשכורת החדשה.
- נוסיף את העובד החדש שיצרנו לעץ העובדים לפי שכר הכללי ולעץ העובדים לפי שכר של החברה (זאת באמצעות הפוינטרים השמורים בעובד, ולכן סיבוכיות ההוספה log(n).
- נבדוק האם המשכורת החדשה גדולה מהמשכורת המקסימלית השמורה, אם כן, נעדכן את השכר המקסימלי ואת המזהה של העובד. זאת הן במבנה הנתונים והן בחברה של העובד.
- אם BumpGrade>0 נעדכן את השדה Grade של העובד לפי BumpGrade. סיבוכיות גישה לשדה O(1).
- <u>לכן, סיבוכיות פונקציה זו במקרה הגרוע היא O(log(n)).</u>

10. StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID)

- נחפש את העובד בעץ AllEmployeesTree. אם העובד לא קיים, נחזיר FAILURE. סיבוכיות חיפוש O(log(n)).
- נחפש את החברה החדשה בה יועסק בעץ AllCompanies. אם לא קיים, נחזיר FAILURE. סיבוכיות חיפוש O(log(k)).

- נשתמש במצביע לחברה השמור בעובד, אם השדה company_id של החברה בה העובד מועסק זהה לNewCompanyID נחזיר FAILURE. סיבוכיות גישה לשדה דרך פוינטר .O(1)
- נקרא לפונקציה GetEmployeeInfo עם משתנים זמניים על מנת לשמור את המידע הרלוונטי על העובד. סיבוכיות הפונקציה O(log(n)).
- נמחק את העובד מהחברה בה עובד כרגע עי קריאה לפונקציה RemoveEmployee. סיבוכיות הפונקציה .O(log(n))
- נוסיף את העובד לחברה החדשה עי קריאה לפונקציה AddEmployee, נעביר לפונקציה את שדות המידע אודות העובד ששמרנו. סיבוכיות הפונקציה היא O(log(n))+.(log(k))
- <u>לכן, סיבוכיות פונקציה זו היא O(log(n))+.(log(k))</u>

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

- נחפש את החברה הרוכשת והחברה הנרכשת בעץ AllCompanies. אם אחת מהן לא נמצאה נחזיר FAILURE. החיפוש בסיבוכיות .O(log(k))
- נשתמש במשתנים זמניים אליהם נזין את הפרטים של החברה הרוכשת על ידי getCompanyInfo. נשמור את מספר העובדים של החברה הנרכשת גם כן. .O(log(k))
- נבדוק האם החברה הרוכשת עומדת בתנאי הרכישה על ידי הנוסחה הנתונה. אם לא, נחזיר FAILURE. אם כן, נשמור משתנה זמני של חישוב הvalue החדש.
- נאחד את עץ הID ועץ הSalary של שתי החברות:
  - ניצור מערכי עזר לכל אחת מהחברות בגודל השדה num_employees (בגדלים $n_{TargetID}$, $n_{acquireID}$) אליהם נעתיק בסיור inOrder את העובדים בחברה. לאחר ההעתקה מהעץ נמחק את העובד מהעץ הישן.
  - ניצור מערך עזר נוסף בגודל $n_{TargetID} + n_{acquireID}$ ונשתמש בmerge sort ליצור מערך ממויין של כל העובדים בחברה לאחר איחוד החברות. ניצור את העץ של החברה הממוזגת בסיבוכיות $n_{TargetID.} + n_{acquireID}$
- נמחק את החברה הרוכשת והחברה הנרכשת על ידי שימוש בRemoveCompany. נשים לב שהפעולה אפשרית מכיוון שמחקנו את העובדים מהחברות הישנות שלהם. סיבוכיות פונקצ מחיקת חברה היא .O(log(k))
- ניצור חברה חדשה עם פרטי החברה הרוכשת באמצעות המשתנים הזמניים. נעדכן במספר העובדים את מספר העובדים של החברה הרוכשת+החברה הנרכשת. נוסיף את עץ העובדים הממוזג תחתיה. נעדכן את העובד המרוויח ביותר בחברה.
- נעבור על העובדים בעץ הממוזג ונעדכן את המצביע לחברה שלהם להיות מצביע לחברה הממוזגת החדשה. זאת בסיבוכיות $n_{TargetID} + O(n_{acquireID})$. לכל עובד שמור פוינטר לחברה בה מועסק ואותו העובד שמור בעץ כל העובדים במערכת ובחברה ולכן הפוינטר יתעדכן בהתאמה בשני העצים.
- <u>לכן, סיבוכיות פונקציה זו היא O(log(k))+ $n_{acquireID}$ + .($n_{TargetID}$</u>

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

- אם CompanyID<0, נזין את id_highest_earner מהמבנה נתונים אל EmployeeID.
- אם CompanyID>0, נחפש את החברה בNonEmptyCompaniesTree, אם לא נמצא נחזיר FAILURE. החיפוש בסיבוכיות log(k).
- אם מצאנו, נעדכן את EmployeeID לערך השמור בחברה ב id_highest_earner.
- <u>לכן אם CompanyId<0 הסיבוכיות O(1) מספר פעולות קבוע, ואם CompanyID>0 הסיבוכיות O(log(k)) חיפוש בעץ החברות.</u>

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

- אם CompanyID<0, נבדוק האם במערכת יש עובדים בAllEmployeesByIDTree אם אין נחזיר FAILURE.
  - נהפוך את סיור InOrder כדי לשלוף את העובדים בסדר המבוקש על העץ AllEmployeesBySalaryTree על מנת למלא את המערך. נחזיר מערך של עובדים.
  - נקצה מערך להחזרה ואליו נזין את המספרים המזהים של העובדים.
  - נשחרר את מערך העובדים הזמני שהצקנו, ונחזיר את מערך המספרים המזהים.

  <u>מעבר על כלל העובדים במקרה זה הוא בסיבוכיות O(n).</u>
- אם CompanyID>0, נחפש את החברה בNonEmptyCompanies, אם לא נמצא נחזיר FAILURE. החיפוש בסיבוכיות log(k)
  - נהפוך את סיור InOrder כדי לשלוף את העובדים בחברה בסדר המבוקש על העץ EmployeesBySalaryTree על מנת למלא את המערך. נחזיר מערך של עובדים.
  - נקצה מערך להחזרה ואליו נזין את המספרים המזהים של העובדים.
  - נשחרר את מערך העובדים הזמני שהצקנו, ונחזיר את מערך המספרים המזהים.

  <u>מציאת החברה ומעבר על העובדים הוא בסיבוכיות זמן $O(\log(k))+(O(n_{companyID})$</u>

14. **StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees)**

- נעבור על עץ הNonEmptyCompaniesTree.
- נבדוק האם הגודל של NonEmptyCompaniesTree קטן מNumOfCompanies. אם כן נחזיר FAILURE.
- נמצא את החברה עם המזהה הנמוך ביותר. מציאת חברה זו בסיבוכיות O(log(k)
- ממנה נסייר בinOrder כמות צעדים NumOfCompanies. סיבוכיות מספר צעדים זה O(NumOfCompanies). נקבל מערך של numOfCompanies חברות.
- נקצה מערך להחזרה בגודל numOfCompanies
- נעבור על המערך ונשלוף את המספר מזהה של העובד המרוויח ביותר מהשדה id_highest_earner. נמלא ערך זה במערך להחזרה. סיבוכיות O(NumOfCompanies)

  מעבר על NumOfCompanies חברות וביצוע מספר קבוע של פעולות הוא בסיבוכיות O(NumOfCompanies)
- <u>לכן סיבוכיות הפונקציה $O(\log(k))+(NumOfCompanies)$</u>

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

  - אם CompanyID<0, נאפס את ערך הפוינטרים שקיבלנו. נעבור על העץ AllEmployeesByIDTree. נמצא את העובד בעל הid המינימלי הגדול שווה לMinEmployeeId וקטן שווה לMaxEmployeeId. אם לא נמצא נסיים. נעבור בסיור inOrder מעובד זה ועד לעובד בעל המזהה המקסימלי שקטן שווה ל MaxEmployeeId. מציאת העובד לתחילת סיור inOrder היא O(log(n). אורך הסיור הינו TotalNumOfEmployees ולכן הסיבוכיות היא O(TotalNumOfEmployees).

    עבור כל עובד:
    - נגדיל את ערכו של האינדקס, שבסוף התהליך יושווה ל . TotalNumOfEmployees
    - נבדוק האם שכרו של העובר גדול שווה מ MinSalary ודרגתו גדולה שווה מ MinGrade, אם כן, נגדיל את ערכו של NumOfEmployees באחד. סיבוכיות גישה לשדות של עובד היא O(1).
    - <u>לכן סיבוכיות פונקציה זו היא $O(\log(n))+(TotalNumOfEmployees)$</u>

- אם CompanyID>0, נעבור על עץ הNonEmptyCompaniesTree. נמצא את החברה לפי המזהה. מציאת חברה זו בסיבוכיות O(log(k)). אם לא נמצאה החברה מחזירים FAILURE.
  - נאפס את ערך הפוינטרים שקיבלנו.
  - נחפש את העובד בעל משכורת מינימלית שגדולה שווה מMinEmployeeID וקטנה שווה מMaxEmployeeId בEmployeesByIdTree, מציאת עובד כזה ברשימה זו היא בסיבוכיות $O(\log(n_{companyID})$ אם לא נמצא עובד סיימנו. נעבור בסיור inOrder מעובד זה ועד לעובד בעל המזהה המקסימלי שקטן שווה ל MaxEmployeeId. מציאת העובד לתחילת סיור inOrder היא $O(\log(\underline{n_{companyID}})$. אורך הסיור הינו TotalNumOfEmployees ולכן הסיבוכיות היא O(TotalNumOfEmployees). עבור כל עובד:
  - נגדיל את ערכו של האינדקס, שבסוף התהליך יושווה ל . TotalNumOfEmployees
  - נבדוק האם שכרו של העובר גדול שווה מ MinSalary ודרגתו גדולה שווה מ MinGrade, אם כן, נגדיל את ערכו של NumOfEmployees באחד. סיבוכיות גישה לשדות של עובד היא .O(1)
  - ולכן $\underline{O(\log(k)+\log(n_{companyID})+(TotalNumOfEmployees)}$

16. void Quit(void **DS)

- נעבור על כל העצים במבנה הנתונים ונמחוק את תוכנם.
- מחיקת העץ AllCompaniesTree בסיבוכיות של .O(k)
- מחיקת העץ NonEmptyCompaniesTree בסיבוכיות של לכל היותר .O(k)
- מחיקת העצים EmployeeByIdTree, AllEmployeeByIdTree, EmployeeBySalaryTree, AllEmployeeBySalaryTree בסיבוכיות של .O(n)

סיבוכיות מקום:

- במהלך התוכנית נקצה זכרון עבור k החברות ועבור n העובדים. נתחזק לכל היותר שישה סוגי עצים שיכילו shared_ptr לחברות ולעובדים בהתאם לצורך. לפיכך תחזוקת העצים לא דורשת הקצאת זיכרון מעבר לO(n+k). סיבוכיות כל עץ להלן:
- AllCompaniesTree - עץ המכיל את כל החברות. גודלו k מספר החברות.
- NonEmptyCompaniesTree - עץ המכיל את החברות הלא ריקות. גודלו לא עולה על k מספר החברות הכולל.
- AllEmployeesByIDTree - עץ המכיל את כל העובדים במערכת. גודלו לא עולה על n מספר העובדים הכולל.
- AllEmployeesSalaryTree - עץ המכיל את כל העובדים במערכת. גודלו לא עולה על n מספר העובדים הכולל.
- EmployeesByIDTree- עץ בחברה המכיל את העובדים בחברה. סכום העובדים בכל העצים בכל החברות לא עולה על n (מספר העובדים הכולל).
- EmployeesBySalary - עץ בחברה המכיל את העובדים בחברה. סכום העובדים בכל העצים בכל החברות לא עולה על n (מספר העובדים הכולל).
- <u>לכן הזיכרון המוקצה לכל העצים הוא בסיבוכיות O(n+k)</u>