מבני נתונים רטוב 1 - החלק היבש

חלק 1: הצגת מבנה הנתונים בצורה כללית:

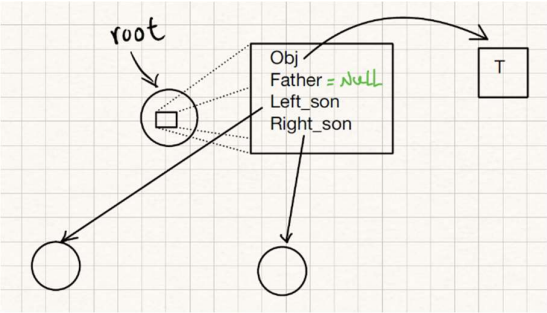
מבנה הנתונים הכללי המבוקש- EmployeeManager- נבנה באמצעות המחלקות הבאות:

1. **עץ AVL גנרי** – מבנה נתונים מסוג עץ חיפוש בינארי מאוזן, שבו עבור כל צומת הפרש הגבהים של שני תתי-העצים של הבנים של הצומת הוא לכל היותר 1.
   a. העץ ממומש כמחלקה המכילה:
      i. מצביע ל"שורש" העץ (מאותחל ב- C'tor ל- NULL).
      ii. פונקציית השוואה גנרית מטיפוס L (מתקבלת ב- C'tor).
      iii. מספר שלם המייצג את גודל העץ- כלומר מספר ה-NODE-ים שבו.
   b. תא בעץ: מוגדר כמבנה המכיל:
      i. מצביע לתוכן גנרי מטיפוס T.
      ii. מצביע לצומת "אב".
      iii. מצביעים לצמתים "בן שמאלי" ו"בן ימני".
      iv. מספר שלם המייצג את גובה הצומת בעץ.

<u>הגנריות של העץ:</u>

1. מחלקת השוואה גנרית- L . מחלקה זו משמשת להשוואה בין האובייקטים בעץ.
2. מחלקת Type גנרית- T והיא קובעת את סוג הטיפוס שעליו יצביע התא.

ציור להמחשה:



כדי לנהל את העץ מימשנו פונקציות רבות בהתאם לנלמד בהרצאות, לדוגמה: Insert, delete, פונקציות הגלגולים למיניהם,find, inorder, treeToArr ו-arrToTree.

2. **מחלקת Employee**

המחלקה מייצגת עובד במערכת ומכילה את כל המידע הרלוונטי לגביו:

a. מספרים שלמים המייצגים מספר זיהוי (ID), שכר (Salary) ודרגה (Grade) של העובד.
b. מצביע לחברה (p_company) שמעסיקה אותו.

לטובת ניהול המחלקה מימשו בנאי, הורס (דיפולטי), מתודות getters שיכולות לגשת לתוכן הפרטי ומתודות setters שיכולות לעדכן אותו.

כמו כן, מימשנו מחוץ למחלקת Employee את המחלקות להשוואת עובדים:

a. לפי מספר זיהוי CompEmployeeByID
b. לפי שכר CompEmployeeBySalary

3. **מחלקת Company**

המחלקה מייצגת חברה במערכת ומכילה את כל המידע הרלוונטי לגביה:

a. מספרים שלמים המייצגים מספר זיהוי (ID), שווי (Value) וגודל (מספר העובדים) של החברה.
b. מצביע ל-Node של העובד בעל השכר הגבוה ביותר בחברה (max_sal_emp).
c. עץ AVL המכיל את העובדים בחברה לפי מספרי זיהוי (ID)
d. עץ AVL המכיל את העובדים בחברה לפי שכר (Salary).
e. מחלקות להשוואת עובדים לפי מספר זיהוי ולהשוואה לפי שכר, לטובת בניית העצים הרלוונטיים.

לטובת ניהול המחלקה מימשו בנאי, הורס, מתודות getters שיכולות לגשת לתוכן הפרטי ומתודות setters שיכולות לעדכן אותו.

בנוסף, מימשנו פונקציות להוספה ולהסרה של עובדים ופונקציות נוספות שעתידות לסייע במימוש דרישות המערכת.

מימשנו מחוץ למחלקת Company את המחלקה להשוואה בין חברות לפי מספר זיהוי C CompEmployeeByID.

4. **מחלקת EmployeeManager**

המחלקה מייצגת את המערכת לניהול כלל העובדים והחברות וממומשת בשפת C++.

המחלקה מכילה:

- מספרים שלמים המייצגים את מספר העובדים במערכת ואת מספר החברות במערכת.
- מצביע לתא שמכיל את העובד בעל השכר הגבוה במערכת.
- מחלקות להשוואת עובדים לפי מספר זיהוי, עובדים לפי שכר, חברות לפי מספר זיהוי.
- ארבעה עצי AVL:
  - עץ המכיל עובדים ומסודר ע"י מספר זיהוי- employee_by_id
  - עץ המכיל עובדים ומסודר לפי שכר- employee_by_salary
  - עץ המכיל חברות ומסודר לפי מספר זיהוי
  - עץ המכיל את החברות שאינן ריקות (מסודר לפי מספר זיהוי)

המתודות במחלקה מקבילות לפונקציות אותן נדרשנו לממש במסגרת התרגיל ועליהן נרחיב בהמשך, בנוסף מימשנו פונקציות עזר נוספות.

בתוך המחלקה מחלקת שגיאות בשם EmExeption שיורשת מ-std::exeption ומורישה למחלקות נוספות בהתאם לשגיאות האפשריות במערכת.

5. **ספריית Library, הפעולות הנדרשות וחישובי סיבוכיות**

ספרייה זו מייצגת את המערכת לניהול כלל העובדים והחברות וממומשת בשפת C.

נפרט בהרחבה על הפעולות אותן נדרשנו לממש:

Init() - הפונקציה מאתחלת מבני נתונים ריק.

הפונקציה יוצרת מבנה EmployeeManage (נסמנו מעתה EM):

EM Ctor– אתחול ערכים קבועים (0), מצביע NULL לעובד המרוויח ביותר במבנה ובנייה של 4 עצי AVL.

Tree Ctor – אתחול של עץ ע"י השמת NULL בשורש, השמה של פוקנציית ההשוואה (המתקבלת כפרמטר לבנאי) ואיפוס size.

בעבור כל הפונקציות הללו סיבוכיות הזמן היא $O(1)$ ולכן הסיבוכיות הכללית של הפונקציה היא $O(1)$ כנדרש.

- AddCompany(void *DS, int CompanyID, int Value)

הפונקציה מוסיפה חברה חדשה עם המזהה CompanyID ושווי התחלתי Value.

1. בדיקה האם החברה כבר נמצאת במערכת, ע"י הפעלת פונקציית find על העץ Company_by_id.
   הפונקציה find משמשת לחיפוש של תא בעץ בהתאם לזו הנלמדה בכיתה.
   הסיבוכיות שלה היא כגובה העץ:
   זמן- $O(log\ k)$
   מקום- $O(log\ k)$ (יש שימוש ברקורסיה)
2. אם היא נמצאת במערכת- זריקה של שגיאה מסוג EmFailure – סיבוכיות זמן ומקום $O(log\ k)$.
3. אם היא לא נמצאת במערכת:
   a. יצירת חברה חדשה ע"י הבנאי של Company, עם המזהה CompanyID ושווי התחלתי Value.
      סיבוכיות זמן ומקום: O(1).
   b. הכנסה של החברה החדשה באמצעות פונקציית insert לעץ Company_by_id.
      זו מפעילה מתודת עזר רקורסיבית בשם addnode (בדומה לזו שנלמדה בכיתה).
      סיבוכיות כגובה העץ:
      זמן- $O(log\ k)$
      מקום- $O(log\ k)$ (יש שימוש ברקורסיה)

סה"כ סיבוכיות הזמן והמקום: $O(log\ k)$.

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

הפונקציה מוסיפה עובד חדש עם מזהה EmployeeID ששייך לחברה CompanyID עם שכר Salary ודרגה Grade.

1. חיפוש החברה בעלת מספר הזיהוי CompanyID בעזרת הפונקציה find בסיבוכיות $O(log\ k)$.
2. בדיקה האם העובד כבר קיים במערכת ע"י הפעלת הפונקציה find על העץ employee_by_id –
   סיבוכיות מקום וזמן $O(log\ n)$ (n מספר העובדים במערכת).
3. אם קיים- זריקת שגיאה מסוג Emfailure.
4. אם לא קיים:
   a. יצירה של עובד חדש עם הפרמטרים המתאימים- O(1).

b. הכנסת העובד לעצים המסודרים לפי מספר זיהוי ולפי משכורת בסיבוכיות $O(log\ n)$ לכל אחת מההכנסות.

c. הכנסה של העובד לשני העצים בתוך החברה שנמצאה ב- (1).

d. עדכון העובד בעל השכר הגבוה ביותר בחברה וב-EM בעזרת הפונקציה getmaxnode בסיבוכיות זמן ומקום $O(log\ n)$.

e. הכנסה של החברה שמצאנו ב- (1) לעץ של החברות הלא ריקות בסיבוכיות $O(log\ n)$, מאחר שיכולות להיות לכל היותר n חברות לא ריקות (כמספר העובדים במערכת, אם לכל חברה יש בדיוק עובד אחד).

סה"כ מסכימה קיבלנו :

סיבוכיות זמן - $O(log\ n + log\ k)$

סיבוכיות המקום – $O(\max\{log\ n, \log k\})$

StatusType RemoveEmployee(void *DS, int EmployeeID) –

הפונקציה מוחקת את העובד בעל המזהה EmployeeID מהמערכת עקב יציאה לפנסיה.

1. חיפוש העובד EmployeeID בעץ employee_by_id בסיבוכיות זמן ומקום $O(log\ n)$.
2. מחיקת העובד מתוך העץ employee_by_id בעזרת הפונקציה deletenode שבתורה קוראת לפונקציה הרקורסיבית removenode שמבצעת את משימתה בסיבוכיות זמן ומקום $O(log\ n)$ כפי שנלמד בכיתה.
3. מחיקת העובד מהעץ employee_by_salary בסיבוכיות זמן ומקום $O(log\ n)$.
4. הסרה של העובד משני העצים שבתוך החברה המעסיקה בסיבוכיות זמן ומקום $O(log\ n)$. הגישה לחברה המעסיקה באמצעות השדה p_company במבנה של employee, ולכן O(1).
5. עדכון העובד בעל השכר המקסימלי (בחברה וגם ב- EM) בעזרת הפונקציה getmaxnode בסיבוכיות $O(log\ n)$.
6. עדכון הגודל של החברה ומספר העובדים ב- EM בסיבוכיות O(1).
7. במידה והחברה המעסיקה התרוקנה לאחר הסרת העובד אז מסירים אותה מעץ של החברות הלא ריקות בסיבוכיות $O(log n)$ (בהתאם להסבר ב- AddEmployee.4.e).

סה"כ סיבוכיות הפונקציה היא $O(log\ n)$ כנדרש.

StatusType RemoveCompany (void *DS, int CompanyID) –

הפונקציה מוחקת את החברה בעלת המזהה CompanyID עקב פשיטת רגל. בעת המחיקה מהמערכת החברה חייבת להיות ריקה.

1. חיפוש החברה CompanyID בעץ company_by_id בסיבוכיות $O(log\ k)$.
2. וידוא כי החברה ריקה ע"י המתודה של company : getsize() בסיבוכיות O(1).
3. מחיקת החברה מהעץ company_by_id בסיבוכיות $O(log\ k)$.

סה"כ סיבוכיות המקום והזמן היא $O(log\ k)$ כנדרש.

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

הפונקציה מחזירה לתוך המצביעים את שווי החברה CompanyID ואת מספר העובדים בה.

1. חיפוש החברה בתוך העץ שמסודר לפי מזהים בסיבוכיות $O(log\ k)$
2. גישה לתוכן הפרטי בעזרת getters של המחלקה company והשמה במצביעים O(1).

סה"כ סיבוכיות הפונקציה היא $O(log\ k)$ כנדרש.

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

הפונקציה מחזירה לתוך המצביעים את מזהה המעסיק, השכר והדרגה של העובד EmployeeID.

1. חיפוש העובד בתוך העץ employee_by_id בסיבוכיות זמן ומקום $O(log n)$.
2. גישה לתוכן הפרטי בעזרת getters של המחלקה Employee והשמתם במצביעים. בסיבוכיות O(1).

סה"כ סיבוכיות הפונקציה היא $O(log\ n)$ כנדרש.

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

הפונקציה מעלה את שווי המניות של החברה CompanyID.

1. חיפוש החברה בעץ company_by_id בסיבוכיות $O(log\ k)$
2. עדכון התוכן הפרטי בעזרת מתודות setters של המחלקה company בסיבוכיות O(1).

סה"כ סיבוכיות הפונקציה היא $O(log\ k)$ כנדרש.

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

הפונקציה מעלה את השכר לעובד EmployeeID וגם מעלה אותו בדרגה בהתאם לערך של BumpGrade.

1. חיפוש העובד בעץ employee_by_id בסיבוכיות מקום וזמן $O(log\ n)$
2. עדכון התוכן הפרטי בעזרת מתודות setters של המחלקה Employee.
3. מחיקת העובד מהעץ employee_by_id והכנסתו מחדש, בסיבוכיות $O(log\ n)$.
4. גישה לחברה שבה העובד מועסק ע"י המצביע p_company בסיבוכיות O(1), מחיקת העובד מהעצים שנמצאים בתוך החברה והכנסתו מחדש- $O(log\ n)$.
   ההוצאה וההכנסה מחדש מוודאות את נכונות העצים.
5. עדכון העובד המרוויח ביותר ב-EM ע"י הפונקציה getMaxNode בסיבוכיות $O(log\ n)$.

סה"כ סיבוכיות המקום והזמן היא $O(log\ n)$ כנדרש.

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

הפונקציה מעבירה את העובד EmployeeID לחברה במזהה NewCompanyID.

1. חיפוש העובד בעץ employee_by_id בסיבוכיות זמן ומקום $O(log n)$.
2. חיפוש החברה NewCompanyID בעץ company_by_id בסיבוכיות $O(log\ k)$,
   גישה לחברה הנוכחית באמצעות המצביע p_company של employee שמצאנו ב-(1).
3. בדיקה האם ה-ID של שתי החברות זהה – O(1).
4. עדכון p_company בעזרת מתודה setCompany של המחלקה Employee ב- O(1).
5. הסרת העובד מהחברה הנוכחית בעזרת הפונקציה RemoveEmployee והוספה לחברה החדשה בעזרת הפונקציה AddEmployee שתיהן בסיבוכיות מקום וזמן $O(log\ n)$.
6. במידה והחברה הקודמת ריקה, מחיקה שלה מהעץ company_not_empty בסיבוכיות $O(log\ n)$.

סה"כ מסכימה, סיבוכיות המקום והזמן הן $O(log\ n + log\ k)$ כנדרש.

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

הפונקציה מטפלת ברכישה של החברה TargetID ע״י החברה AcquirerID.

1. חיפוש 2 החברות בעץ company_by_id בסיבוכיות $O(log\ k)$.
2. חישוב הערך והגודל החדשים של AcquirerID באמצעות מתודות getters וחישובים חשבוניים בסיבוכיות O(1).
3. הקצאת 6 מערכי עזר ממוינים המכילים מצביעים לטיפוס Employee.
   a. מערכים 2+1- יכילו מצביעים לעובדים ב-Target. (1) ממויין לפי ID ו-(2) ממויין לפי salary. סיבוכיות מקום היא $O\ (n_{\text{TargetID}})$.
   b. מערכים 4+3- יכילו מצביעים לעובדים ב-Acquire. (3) ממויין לפי ID ו-(4) ממויין לפי salary. סיבוכיות מקום היא $O\ (n_{\text{AcquirerID}})$.
   c. מערכים 6+5 - יכיל מצביעים לעובדים בשתי החברות. (5) ממויין לפי ID ו-(6) ממויין לפי salary. סיבוכיות מקום : $O(n_{\text{AcquirerID}}) + O\ (n_{\text{TargetID}})$.
4. קריאה לפונקציה treeToArray על כל אחד מהעצים.
   בפונקציה יש מעבר inorder על העץ והשמה של כל obj ב-node לתא במערך. סיבוכיות מקום וזמן $O(n_{\text{TargetID}} + n_{\text{AcquirerID}})$ (2 קריאות סה״כ). המערך שיתקבל יהיה ממויין (תכונה של inorder).
5. קריאה לפונקציה merge פעמיים (פעם אחת עבור ID ופעם שניה עבור salary), איחוד המערכים של 2 החברות למערך combined ע״י אלגורים מיזוג של 2 מערכים ממוינים בסיבוכיות $O(n_{\text{TargetID}} + n_{\text{AcquirerID}})$.
6. עדכון המצביע p_company של כל עובד לחברה החדשה באמצעות ריצה על המערך combinedID בסיבוכיות זמן $O(n_{\text{TargetID}} + n_{\text{AcquirerID}})$.
7. קריאה לפונקציה Arraytotree :
   a. חישוב גובה העץ החדש (combined) נסמנו height.
   b. יצירת עץ AVL שלם וריק בגובה height. סיבוכיות מקום וזמן $O(n_{\text{TargetID}} + n_{\text{AcquirerID}})$.
   c. חישוב מספר התאים שצריך להחסיר מהעץ החדש והסרתם בסיבוכיות O(1).
   d. העברה של שני המערכים המאוחדים לעץ החדש שיצרנו בריצת inorder ובסיבוכיות מקום וזמן
   e. ריקון העצים של 2 החברות לפני האיחוד מעובדים בסיבוכיות $O(n_{\text{AcquirerID}}) + O\ (n_{\text{TargetID}})$
   f. הסרה של 2 החברות מ-EM (מערך company_by_id ומערך company_not_empty) בעזרת RemoveCompany בסיבוכיות $O(log\ k)$.
   g. הכנסת את החברה החדשה לתוך עץ החברות לפי מזהה ואם צריך גם לעץ של החברות הלא ריקות. סיבוכיות $O(log\ k)$.
   h. שחרור מערכי העזר.

סה״כ סיבוכיות הזמן והמקום הן $O(logk + n_{\text{TargetID}} + n_{\text{AcquirerID}})$.

סיבוכיות המקום הנ״ל היא גם $O(\mathrm{k} + \mathrm{n})$ מהחישוב (עד כדי קבועים) :

$$logk + n_{\text{TargetID}} + n_{\text{AcquirerID}} \leq k + n$$

StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID) – הפונקציה מחזיר לתוך מצביע את העובד בעל השכר הגבוה ביותר בחברה או בכל המערכת.

1. אם CompanyID<0 אז החזרה של העובד שנמצא בשדה max_employee ב-EM. סיבוכיות $O(1)$.
2. אם CompanyID<0 :
   a. חיפוש החברה בתוך העץ company_by_id בסיבוכיות זמן ומקום $O(log\ k)$
   b. בדיקת מספר העובדים בחברה ע"י המתודה getSize(). סיבוכיות O(1).
   c. במידה ומספר העובדים גדול מ—0 , החזרת השדה max_sal_emp בחברה ב- O(1).

סה"כ סיבוכיות זמן ומקום $O(log\ k)$ במקרה הגרוע כנדרש.

StatusType GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees) – הפעולה מחזירה לתוך המערך Employees את כל העובדים, בחברה או במערכת, ממוינים בסדר יורד לפי המשכורות שלהם.

הפונקציה משתמשת בפונקציה בשם printtolist שמבצעת ריצת Inorder על העץ ומכניסה את האובייקטים שהוא מכיל לתוך מערך ממויין (תכונה של Inorder).

1. כאשר CompanyID<0 אז לעבור על העץ employee_by_salary שמכיל את כל העובדים במערכת, ולכן הפעולה מתבצעת בסיבוכיות O(n).
2. כאשר CompanyID>0 :
   a. מציאת החברה הרלוונטית בסיבוכיות $O(log\ k)$
   b. מעבר על העץ c_employee_by_salary בסיבוכיות $O(n_{companyID})$

סה"כ סיבוכיות זמן ומקום מקרה 1 : O(n) כנדרש.

סיבוכיות זמן ומקום מקרה 2 : $O(\log k + n_{companyID})$ כנדרש.

StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees) – הפונקציה מחזירה את העובד בעל השכר הגבוה ביותר עבור כל אחת מ-NumOfCompanies החברות בעלות המזהים הנמוכים ביותר.

1. הקצאת זיכרון עבור מערך int בגודל NumOfCompanies, כלומר סיבוכיות מקום : $O(NumOfCompanies)$.
2. בדיקה האם מספר החברות הלא-ריקות ב-EM <u>לא קטן</u> מ- NumOfCompanies– O(1).
3. קריאה לפונקציה getmax על העץ company_not_empty. זו פונקציה רקורסיבית שעוברת Inorder על העץ, בעלת counter מתעדכן עם כל הכנסה למערך.

   הפונקציה יורדת עד לענף השמאלי ביותר בעץ (החברה בעלת המזהה הקטן ביותר)- סיבוכיות זמן ומקום $O(log\ k)$.

   עם העלייה ב- Inorder הכנסה של העובד המרוויח ביותר בעזרת המתודה getMax של company בסיבוכיות O(1) ועדכון ה- counter. תנאי העצירה יגרום לפונקציה להיעצר כאשר counter>=n – כלומר המעבר הזה הוא בסיבוכיות $O(NumOfCompanies)$.

סה"כ מסכימה נקבל שסיבוכיות הזמן והמקום (כולל הקצאת המערך) : $O(\log k + NumOfCompanies)$

StatusType GetNumEmployeesMatching(void *DS, int CompanyID, int MinEmployeeID, int MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees) – הפונקציה תחזיר לתוך TotalNumOfEmployees את מספר העובדים, מתוך המערכת או חברה ספציפית, שהמזהה שלהם בין MinEmployeeID ל MaxEmployeeId ולתוך NumOfEmployees את מספר העובדים ששכרם לפחות MinSalary ודרגתם לפחות MinGrade.

1. אם CompanyID<0 :
   a. קריאה לפונקציה getmatch הרקורסיבית על העץ employees_by_id :
      i. בדיקת השורש – אם NULL אז עצירה.
      ii. בדיקת מפתח השורש – אם בטווח המבוקש הגדלה של הcounter המתאים וקריאה רקורסיבית לפונקציה על הבן השמאלי ולאחריו על הבן הימני. הפונקציה יורדת שמאלה עד לתא שמקיים employeeId<MinEmployeeID ועולה עד לתא שמקיים employeeId>MaxEmployeeID.
      iii. אם המפתח גדול מהתחום המבוקש – קריאה לפונקציה על הבן השמאלי.
      iv. אם המפתח קטן מהתחום המבוקש – קריאה לפונקציה על הבן הימני.

סה"כ הירידה למטה – $O(log\ \tilde{n})$ כאשר $\tilde{n} = n$ ומעבר על התאים פעם אחת – O(TotalNumOfEmployees) ומסכימה נקבל $O(log\ n + TotalNumOfEmployees)$.

2. אם CompanyID>0 :
   a. חיפוש החברה CompanyID בסיבוכיות $O(log\ k)$
   b. קריאה לפונקציה getmatch הרקורסיבית על העץ c_employee_by_id בדומה ל- 1.a .

סה"כ מסכימה סיבוכיות הזמן והמקום : $O(log\ n_{company} + \log k + TotalNumOfEmployees)$.

void Quit(void **DS) – הפונקציה משחררת את כל הזכרון אותו הקצנו עבור המבנה.

1. קריאה ל- D'TOR של המחלקה EmployeeManager
2. ה- D'TOR קורא לפונקציה DestroyTree (ראה הערה 2 עבור is_obj) עבור :
   a. Employee_by_id – עם is_obj = false
   b. Employee_by_salary – עם is_obj = false
   c. Company_not_empty – עם is_obj = false
   d. Company_by_id – עם is_obj=true.

Destroytree מבצעת ריצת postorder על העצים ומוחקת את כל ה-NODE-ים בסיבוכיות O(n) עבור עצים המכילים עובדים ו- O(k) עבור עצים המכילים חברות.

עבור d – יש קריאה גם להורס של האובייקט (שהוא company). ההורס של company קורא ל- D'tor עבור שני העצים שבו, ועבור העץ c_employee_by_salary מפעיל את ה- D'tor של employee (דיפולטי).

אין חריגה בסיבוכיות המקום משום שבמקרה הגרוע ביותר : $O(log\ n + \log k)$.

סה"כ סיבוכיות מקום וזמן : O(n+k).

הערות:

1. **הערה על סיבוכיות המקום** – בכל הפונקציות סיבוכיות המקום שתיארנו לא חורגת מסיבוכיות המקום הנדרשת שהיא $O(n + k)$.
2. **הערה על מחיקה של אובייקטים** – משום שבחרנו להשתמש במצביעים רגילים ולא ב- shared_ptr , נאלצנו לעבוד עם כל פונקציות המחיקה בזהירות.

הפונקציה	מחיקה של ה-NODE	מחיקה של האובייקט
RemoveEmployee RemoveCompany	V	V
HireEmployee	V	X
AcquireCompany	V	X
QUIT	V	V

כדי להבדיל בין המחיקות, הוספנו פרמטר לפונקציות המחיקה בעץ DeleteNode ו- DestroyTree בשם is_obj :

- True – למחוק את האובייקט ולהפוך את המצביע אליו ל-NULL, ואז מחיקת התא.
- False – למחוק את התא אך לא את האובייקט.