חלק יבש לרטוב 2

חלק 1: הצגת מבנה הנתונים:

1. המבנים בהם השתמשנו לפתרון התרגיל:

a. עץ דרגות – RankTree

i. העץ מכיל:
   1. מצביע לשורש העץ (מאותחל כ-NULL)
   2. פונקציית השוואה גנרית מסוג L (מתקבלת ב- C'tor)
   3. מספר שלם המייצג את גודל העץ.

ii. כל תא בעץ מכיל:
   1. obj – מצביע לתוכן גנרי מטיפוס T
   2. מצביע לצומת אב
   3. מצביע לצמתים $right\_son$ ו- $left\_son$.
   4. מספר שלם המייצג את גובה הצומת בעץ
   5. מספר שלם המייצג את הדרגה ($Grade$) של הצומת (מאותחל ע"י $obj \rightarrow getRank()$)
   6. מספר שלם המייצג את סכום הדרגות בתת העץ של הצומת - $rank\_subtree$
   7. מספר שלם המייצג את מספר הצמתים בתת העץ של הצומת – $size\_subtree$

iii. חישוב $index$ של צומת: נחפש את הצומת בעץ:
   1. $Index = 0$
   2. אם פנינו ימינה – נוסיף ל $index$ את גודל תת העץ ($size\_subtree$) של הבן השמאלי של הצומת ממנה פנינו פלוס 1 (עבור הצומת ממנה פנינו)
   3. אם פנינו שמאלה לא נוסיף.

iv. חישוב $Grade$ של כל הצמתים **עד לצומת הנוכחית כולל**:
   חישוב זה ישמש אותנו בפונקציות $SumOfBumpGradeBetweenTopWorkersByGroup$, $AverageBumpGradeBetweenSalaryByGroup$.
   1. נאתחל $sum = 0$ ונחפש את הצומת בעץ :
   2. אם פנינו ימינה- נוסיף ל sum את סכום הדרגות בתת העץ השמאלי של הצומת ממנה פנינו פלוס הדרגה של הצומת ממנה פנינו:
   $$sum += rank(root) + (root \rightarrow left_{son} \rightarrow rank\_subtree)$$
   3. אם פנינו שמאלה לא נוסיף.

v. הכנסה, הוצאה וגלגולים בעץ:
   1. לאחר ביצוע גלגול, יתבצע עדכון של rank_subtree ו- size_subtree של הצמתים אותם שינינו, בדומה לעדכון הגובה. זהו מספר סופי של עדכונים, כלומר O(1).

b. טבלת ערבול – HashTable

i. הטבלה מכילה:
   1. משתנה M – גודל המערך, משתנה N – מספר האובייקטים בטבלת הערבול.
   2. מערך של תאי Hnode, שמכילים בתוכם מצביע לעץ של עובדים ממוינים לפי ID (בחרנו להשתמש בעץ ולא ברשימה מקושרת משום שהעץ גנרי ועובד טוב, וגם הסיבוכיות במקרה הגרוע טובה יותר).
   3. פונקציית ערבול $h(x) = x \bmod M$.
   בדרישות התרגיל הסיבוכיות היא <u>ממוצע על הקלט</u> – כלומר פונקציית mod על גודל המערך טובה עבור הדרישה.
   4. פונקציית resize:
      a. אם $\frac{N}{M} < 0.5$ כלומר אם העומס קטן מ- 0.5, נבצע הקטנה של המערך.
      b. אם $\frac{N}{M} > 1.5$ כלומר אם העומס גדול מ- 1.5, נבצע הגדלה של המערך.

הסיבוכיות של resize היא $O(N)$ משום שיבוצע חישוב $hash$ מחדש על כל אחד מהאובייקטים במערך והכנסתו למערך המעודכן.

c. **מבנה UnionFind**

i. המבנה מכיל:

1. מערך objects – מכיל מצביעים למבנה Unode.
2. מערך sizes –מכיל את הגודל של הקבוצות במבנה.

ii. כל תא Unode מכיל:

1. obj – מצביע לתוכן גנרי מטיפוס T
2. מצביע לתא אב
3. דרגה של הצומת (מאותחל ע"י $obj \rightarrow getRank()$)
4. $leader$ – מצביע ל"מנהיג" של הקבוצה.

כפי שלמדנו, איחוד של קבוצות ב- UF תלוי אך ורק בגודל של הקבוצה.

לכן, כדי שנדע איזה חברה היא "הגבוהה" ביותר בהיררכיה של הקבוצה, נשמור מצביע $leader$ שיתעדכן בכל פעם שהקבוצה של חברה נרכשת גדולה מהקבוצה של החברה הרוכשת.

בכל פעם שהפונקציה $acquireCompany$ תיקרא, נבדוק את ה- $leader$ של השורש של הקבוצה של החברה הנרכשת:

- אם $leader = NULL$ - השורש של הקבוצה הוא זה שמכיל את החברה "הגבוהה" ביותר בהיררכיה.
- אם $leader! = NULL$ – השורש של הקבוצה הוא לא "הגבוה" ביותר ואילו $leader$ מצביע על החברה "הגבוהה" ביותר. אז נחשב את ה- $value$ של ה- $leader$, נכפול בפקטור ונבצע חישובים בהתאם (יפורט בהמשך).

iii. חישוב $Value$ של חברה:

1. הגעה לחברה ע"י: $company = objects[companyID] \rightarrow obj$
2. עלייה עד לשורש וסכימה על דרגות האיברים במסלול.

הסכום שיתקבל הוא השווי של החברה. (<u>פירוט נוסף במימוש של $CompanyValue()$ ושל $AcquireCompany()$</u>).

בכל חישוב של Value() ו- Find() נבצע rebase לחברה, ובכך הסיבוכיות המשוערכת היא $O(log * k)$:

- נעלה עד לשורש הקבוצה
- נחבר את כל התאים במסלול לשורש הקבוצה ונעדכן את ה- rank בהתאם.

כל מבני הנתונים הם גנריים.

2. **מחלקת Employee**

המחלקה מייצגת עובד במערכת ומכילה את כל המידע הרלוונטי לגביו:

- פרטי העובד: ID , $grade$, $salary$
- $p\_company$ – מצביע לחברה שמעסיקה את העובד

- בנאי, הורס, מתודות $getters$ לגישה לתוכן ומתודות $setters$ לעדכון התוכן.
- מחלקות להשוואת עובדים:
  - לפי מספר זיהוי $CompEmployeeByID$
  - לפי שכר ואז לפי מספר זיהוי $CompEmployeeBySalary$

3. **מחלקת Company**

המחלקה מייצגת חברה במערכת ומכילה את כל המידע הרלוונטי לגביה:

- מזהה החברה ID
- $num\_employees$ - מספר העובדים בחברה
- num_zero - מספר העובדים המתמחים (משכורת שווה ל-0) בחברה

- $grades\_zero$ – סכום דרגות העובדים המתמחים בחברה

- בנאי, הורס, מתודות $getters$ לגישה לתוכן ומתודות $setters$ לעדכון התוכן.
- מבנים של עובדים:
  - עץ דרגות לעובדים כמו זה שפורט ב- a. העץ ממויין לפי $CompEmployeeBySalary$.
  - טבלת ערבול לעובדים כמו זו שפורטה ב- b. העצים בטבלה ממוינים לפי $CompEmployeeByID$.

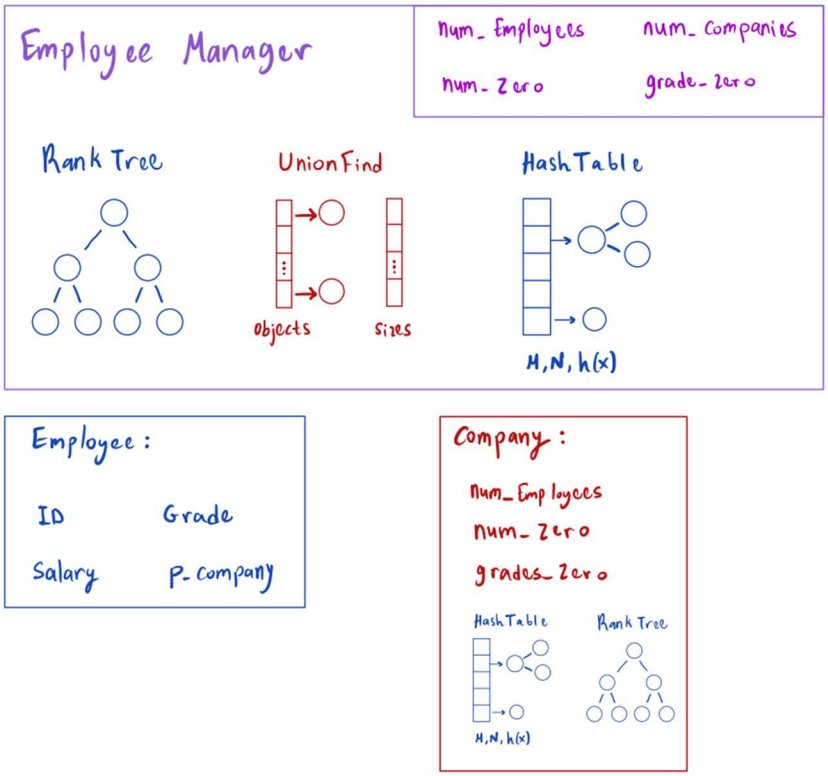
מחלקת EmployeeManager

המחלקה מייצגת את המערכת לניהול כלל העובדים והחברות.
המחלקה מכילה:

- Hash Table המכיל Employees, כאשר המפתח הוא ה-ID של העובדים.
- Rank Tree המכיל Employees, ממויין לפי Salary ואז ID.
- Union Find המכיל Companies.
- משתנה שמכיל את מספר העובדים ללא שכר, ומשתנה שמכיל את סכום דרגות העובדים ללא שכר.
- משתנה שמכיל את מספר החברות במבנה ומשתנה שמכיל את מספר העובדים במבנה.

המתודות במחלקה EmployeeManager מקבילות לפונקציות אותן נדרשנו לממש במסגרת התרגיל ועליהן נרחיב בהמשך.
בתוך המחלקה מימשנו מחלקת שגיאות EmException שיורשת מ- std::exception ומורישה למחלקות נוספות בהתאם לשגיאות האפשריות במערכת.

<u>שרטוט של המבנה:</u>



<u>חלק 2: ספריית Library2, הפעולות הנדרשות וחישובי סיבוכיות</u>

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

נפרט על הפעולות אותן נדרשנו לממש:

Init(int K)

הפוקנציה מאתחלת מבנה נתונים עם K חברות ריקות.

הפונקציה יוצרת מבנה EmployeeManager (נסמנו EM):

- EM – אתחול הקבועים ל-0. <u>סיבוכיות זמן</u>: $O(1)$.
- Union Find – יצירת מבנה UF בגודל K של חברות ריקות, ע"י יצירה של 2 מערכים בגודל K:
  - אתחול מערך objects במצביעים לחברות.
  - אתחול מערך sizes ב- 1.

  החברות באתחול הן סינגלטונים. <u>סיבוכיות זמן</u>: $O(k)$.
- RankTree – יצירת עץ דרגות ריק. <u>סיבוכיות זמן</u>: $O(1)$.

סיבוכיות זמן כוללת: $O(k)$.

AddEmployee(Void *DS, int EmployeeID, int CompanyID, int Grade)

הוספת עובד חדש עם מזהה $EmployeeID$ ששייך לחברה $CompanyID$ עם שכר 0 ודרגה $Grade$.

1. חיפוש העובד ב- HT של EM. אם הוא נמצא להחזיר שגיאה. סיבוכיות: $O(1)$ בממוצע על הקלט.
2. אם לא נמצא, הוספה של העובד ל- HT של EM. סיבוכיות: $O(1)$ משוערך.
3. חיפוש השורש של הקבוצה של החברה ב- UF. סיבוכיות: $O(log * k)$ משוערך.
4. הכנסה של העובד ל- HT של החברה. סיבוכיות: $O(1)$ בממוצע על הקלט.
5. עדכון הקאונטרים ב- EM ובחברה. סיבוכיות: $O(1)$.

סיבוכיות זמן כוללת: $O(log * k)$ משוערך.

removeEmployee(void *DS, int EmployeeID)

העובד בעל המזהה $EmployeeID$ יוצא לפנסיה, וצריך למחוק אותו מהמערכת.

1. חיפוש העובד ב- HT של EM. אם הוא לא נמצא להחזיר שגיאה. אם נמצא למחוק אותו מ-HT. סיבוכיות: $O(1)$ משוערך.
2. מחיקת העובד מהעץ ב- EM במידה והשכר שלו שונה מ-0. סיבוכיות: $O(logn)$.
3. גישה לשורש הקבוצה של החברה המעסיקה באמצעות המצביע p_company. סיבוכיות: $O(1)$.
4. מחיקת העובד מ-HT בחברה. סיבוכיות: $O(1)$ משוערך.
5. מחיקת העובד מהעץ בחברה במידה והשכר שלו שונה מ-0. סיבוכיות: $O(logn)$.
6. עדכון הקאונטרים ב- EM ובחברה. סיבוכיות: $O(1)$.

סיבוכיות זמן כוללת: $O(logn)$ משוערך.

<u>הסבר סיבוכיות O(1) משוערך עבור AddEmployee ו RemoveEmployee מ- HashTable:</u>

כאשר מוציאים / מכניסים אובייקט מ- HashTable, ייתכן ונצטרך לבצע resize() לטבלה, כלומר הקטנה/הגדלה של המערך.

לכן במקרה הגרוע, סיבוכיות הזמן היא $O(N)$ כאשר N הוא מספר האובייקטים בטבלה.

אולם בתרגול ראינו כי סיבוכיות הזמן <u>המשוערכת</u> היא $O(1)$.

לכן במקרה זה, סיבוכיות הזמן המשוערת של הכנסה, הוצאה (וחיפוש – בממוצע על הקלט) היא $O(1)$.

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

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

1. בדיקה האם החברות נמצאות תחת אותה קבוצה- אם כן החזר שגיאה.
2. מחפשים את שתי החברות ב-UF. תוצאת החיפוש תהיה שורש הקבוצה של כל אחת מהחברות- נסמנם ra שורש הקבוצה שמכילה את $Acquirer$ ו- rt שורש הקבוצה שמכילה את $Target$.
   סיבוכיות זמן: $O(log * k)$ משוערך עם CompanyValue (פונקציה דומה ל- FIND שלמדנו) כפי שלמדנו בתרגול.
3. בודקים איזה מהקבוצות גדולות יותר ומחשבים את $value$ (הערך שישמש להכפלה ב- factor של הקבוצה שמכילה את $target$) באמצעות המצביע $leader$ של rt.
4. מבצעים UNION:
   a. אם $sizes[ra] >= sizes[rt]$:
   מחברים מצביע מ- rt ל- ra ומחשבים:
   $$ra \rightarrow rank \mathrel{+}= factor * value$$
   $$rt \mathrel{-}= ra \rightarrow rank$$
   בחישוב ה- $value$ של rt נסכום עד לשורש (כפי שתואר ב- 1.c.iii). לכן, כדי לוודא שהחיבור של rt לשורש ra לא יפגע בחישוב ה- $value$, נחסיר את הערך החדש של ra מה- $rank$ של rt.
   b. אם $sizes[ra] < sizes[rt]$:
   $$ra \rightarrow rank \mathrel{+}= factor * value - rt \rightarrow rank$$

   ועדכון ה- $leader$ של rb להיות $ra \rightarrow leader$ - משום שהוא "באמת" ראש הקבוצה.

   סיבוכיות זמן: $O(log * k)$ משוערך עם CompanyValue.

5. איחוד עצי הדרגות של שתי החברות והכנסת העץ המאוחד לחברה *ra*.
   הסבר קצר על איחוד עצי דרגות (הוסבר בהרחבה בתרגיל רטוב 1):
   1. מעבר ב- *inorder* על כל אחד מהעצים והעתקת האיברים למערך.
   2. איחוד באמצעות הפונקציה *merge* למערך מאוחד.
   3. מעבר על המערך המאוחד והכנסה של האיברים לעץ כמעט שלם שיכיל את כל האיברים.
   4. מעבר ב- *PostOrder* על העץ המאוחד ועדכון *rank_subtree* ו*size_subtree*.

   סיבוכיות זמן: $O(n_{acquirer} + n_{target})$
6. איחוד *HASH TABLE* – האיחוד מתבצע ע"י העברת כל העובדים מ- *HashTable* של שורש הקבוצה הקטנה למערך, מעבר על המערך ועדכון המצביע *p_company* בכל אחד מהעובדים, ואז הכנסת העובדים לשורש הקבוצה הגדולה יותר.
   סיבוכיות זמן: $O(n_{acquirer}) \,||\, O(n_{target}) = O(n_{acquirer} + n_{target})$.
   בסיום התהליך, רק השורש של הקבוצה הגדולה יותר יכיל את המידע.

סיבוכיות זמן כוללת: $O(log * k + n_{acquirer} + n_{target})$.

employeeSalaryIncrease(void *DS, int EmployeeID, int SalaryIncrease)

העובד עם מזהה $EmployeeID$ מקבל העלאת שכר בגובה $SalaryIncrease$.

1. חיפוש העובד ב- *HASH TABLE* של כל העובדים במבנה.
   סיבוכיות זמן: כפי שלמדנו בהרצאה, חיפוש ב- *HashTable* הוא $O(1)$ בממוצע על הקלט.
2. עדכון השכר של העובד.
3. עדכון העץ במבנה *EM*, ע"י הוצאת העובד לפני העדכון והכנסתו לאחר העדכון. סיבוכיות זמן $O(logn)$.
4. גישה לשורש הקבוצה שמכילה את החברה שאליה שייך העובד באמצעות מצביע *p_company* ועדכון העץ בחברה ע"י הוצאת העובד לפני העדכון והכנסתו לאחר העדכון. סיבוכיות זמן $O(logn)$.
5. עדכון המשתנים *num_zero* ו- *sum_grades_zero* בחברה וב-*EM* אם לפני ההעלאה העובד היה בעל משכורת 0: $O(1)$.

נציין כי **העדכון הוא על העצים בלבד**, ואין הוצאה / הכנסה מה- *HASH TABLE* (הכנסה או הוצאה עלולים לגרום ל-$resize()$).

סיבוכיות זמן כוללת: $O(logn + 1) = O(logn)$ בממוצע על הקלט.

promoteEmployee(void *DS, int EmployeeID, int BumpGrade)

דרגת העובד עם מזהה $EmployeeID$ עולה בערך של $BumpGrade$.

1. חיפוש העובד ב- *HASH TABLE* של כל העובדים במבנה.
   סיבוכיות זמן: כפי שלמדנו בהרצאה, חיפוש ב- *HashTable* הוא *O(1)* בממוצע על הקלט.
2. עדכון הדרגה של העובד.
3. עדכון העץ במבנה *EM*, ע"י הוצאת העובד לפני העדכון והכנסתו לאחר העדכון. סיבוכיות זמן $O(logn)$.
4. גישה לחברה שאליה שייך העובד באמצעות מצביע *p_company* ועדכון העץ בחברה ע"י הוצאת העובד לפני העדכון והכנסתו לאחר העדכון. סיבוכיות זמן $O(logn)$.
5. עדכון המשתנה *sum_grades_zero* בחברה וב-*EM* אם העובד בעל משכורת 0: $O(1)$.

נציין כי **העדכון הוא על העצים בלבד**, ואין הוצאה / הכנסה מה- *HASH TABLE* (הכנסה או הוצאה עלולים לגרום ל-$resize()$).

סיבוכיות זמן כוללת: $O(logn + 1) = O(logn)$ בממוצע על הקלט.

sumOfBumpGradeBetweenTopWorkersByGroup (void *DS, int CompanyID, int m)

הפעולה מחשבת את סכום הדרגות $(Grades)$ של m העובדים עם המשכורות $(Salary)$ הגבוהות ביותר בחברה או במערכת, ללא עובדים עם משכורת $salary = 0$.

1. חיפוש החברה $companyID$ ב- *UF*. סיבוכיות זמן: $O(log * k)$ משוערך עם $companyValue$ כפי שלמדנו.
2. אבחנה: בשורש העץ *rank_subtree* מכיל את סכום הדרגות של כל התאים בעץ. נסמן:
   $$tmax = root \rightarrow rank\_subtree$$
3. נחשב את סכום הדרגות של כל התאים עד לתא בתא $size - m$ כמתואר ב- *1.a.iv*. נסמן ערך זה ב- *tmin*. סיבוכיות $O(logn)$ כגובה העץ.
4. נחסר $tmax - tmin$ וזהו סכום הדרגות של *m* העובדים בעלי המשכורות הגבוהות ביותר.

סיבוכיות זמן כוללת: $O(log * k + logn)$ משוערך.

averageBumpGradeBetweenSalaryByGroup (void *DS, int CompanyID, int lowerSalary, int higherSalary)

הפעולה מחשבת את ממוצע הדרגות ($Grades$) של העובדים בעלי משכורת שנמצאת בטווח של lowerSalary ו- higherSalary כולל הקצוות.

1. חיפוש החברה $companyID$ ב- *UF*. סיבוכיות זמן: $O(log * k)$ משוערך עם $companyValue$ כפי שלמדנו.
2. נחלק למקרים לפי הטווח שקיבלנו:
   a. אם הטווח שקיבלנו $[0,0]$:
   נשתמש במשתנים של מספר המתמחים וסכום דרגות המתמחים, ואת הממוצע נחשב כך: $avg = \frac{zero\_sum}{zero\_num}$.

   b. אם הטווח שקיבלנו $[0, higher]$:
      i. נחשב את האינדקס ואת סכום הדרגות של כל התאים עד לתא בעל ה- ID הגדול ביותר שמקיים: $salary == higher$ כפי שתואר ב- 1.a.iv , 1.a.iii.
      ii. נשתמש במשתנים של מספר המתמחים וסכום דרגות המתמחים, ואת הממוצע נחשב כך: $avg = \frac{high\_sum+zero\_sum}{index\_high+zero\_num}$.

   c. אם הטווח שקיבלנו$[lower, higher]$:
      i. נחשב את b.i.
      ii. נחשב את האינדקס ואת סכום הדרגות של כל התאים עד לתא בעל ה- ID הגדול ביותר שמקיים: $salary == lower - 1$ כפי שתואר ב- 1.a.iv , 1.a.iii.
      iii. נחשב את הממוצע כך: $avg = \frac{high\_sum+lower\_sum}{index\_high+index\_low}$

סיבוכיות הזמן בסעיף 2: $O(logn)$ (עבור מקרה i הסיבוכיות היא $O(1)$)

סיבוכיות זמן כוללת: $O(log * k + logn)$ משוערך.

companyValue(void *DS, int CompanyID)

1. גישה לחברה במבנה *UF* בסיבוכיות $O(1)$ כנלמד.
2. חישוב ה $value$ של החברה על ידי סכימה של כל ה-*rank*-ים של החברות בדרך אל שורש הקבוצה שבה החברה נמצאת.

(כפי שתואר ב- *AcquireCompany* , ה- *rank* של חברה מחושב בצורה שכאשר נרצה למצוא את ה-*value* שלה, נסכום את ה *rank*-ים של כל התאים עד לשורש).

סיבוכיות זמן כוללת: $O(log * k)$ משוערך.

void Quit(void **DS)

הפעולה משחררת את המבנה. בסוף השחרור מציבים ערך NULL ב-DS.

1. קריאה ל -*D'tor* של המחלקה *EmployeeManager*.
2. קריאה לפונקציה *DestroyTree* (ראה הערה 2 עבור *is_obj*).
3. קריאה לפונקציה *DestroyHashTable* (ראה הערה 2 עבור *is_obj*).
4. קריאה לפונקציה *DestroyUF*:
   a. הריסה של המבנה *UF*
   b. קריאה ל- *D'tor* של *Company*, שקורא בעצמו לסעיפים 2+3.

סיבוכיות זמן כוללת: $O(n+k)$ משוערך.

הערות:

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

הפונקציה	מחיקת תא ב-*RankTree*	מחיקה ב- *HashTable*	מחיקה ב- *UnionFind*	מחיקה של האובייקט
AddEmployee	*X*	במידה ויש *resize()* נעתיק את כל האובייקטים למערך חדש	*X*	*X*
RemoveEmployee	*V*	*V* במידה ויש *resize()* נעתיק את כל האובייקטים למערך חדש	*X*	*V*
AcquireCompany	*V*	מחיקת ה- *HASH* של החברה בקבוצה הקטנה	*X*	*X*
Quit	*V*	*V*	*V*	*V*

כדי להבדיל בין המחיקות הוספנו פרמטר *is_obj*:

- לפונקציות המחיקה בעץ- *DeleteNode()* ו- *DestroyTree()*.
- לפונקציות המחיקה ב- *HashTable* – *Remove()* ו- *DestroyHashTable()*.

משמעות הפרמטר *is_obj*:

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

מחיקה של אובייקט ב- *UnionFind* תתבצע רק בקריאה ל- *Quit()*, לכן נרצה למחוק גם את האובייקט (*Company*) עצמו ואת כל המבנים שבו (*HashTable* ועץ דרגות).