מבני נתונים 1

234218

תרגיל רטוב

1

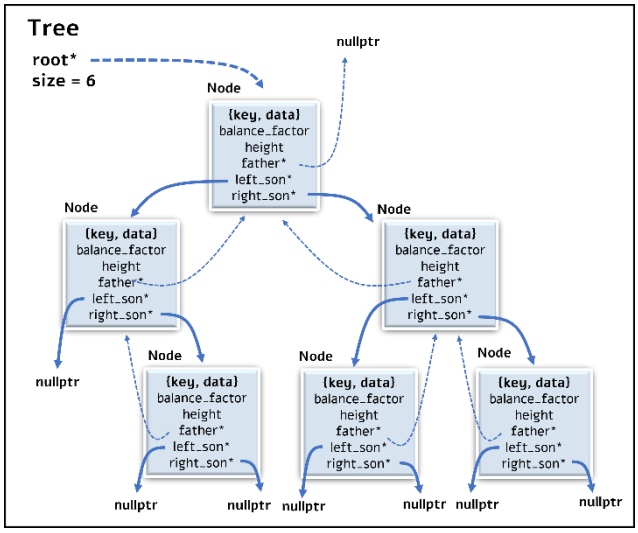
מספר

הוגש ע"י:

עדי צחורי	315374066
שם	מספר זהות
מתן צחורי	208936989
שם	מספר זהות

מבני נתונים 1 - 234218 - אביב 2022

גיליון רטוב 1



מבני הנתונים בהם נשתמש למימוש המבנה שלנו:

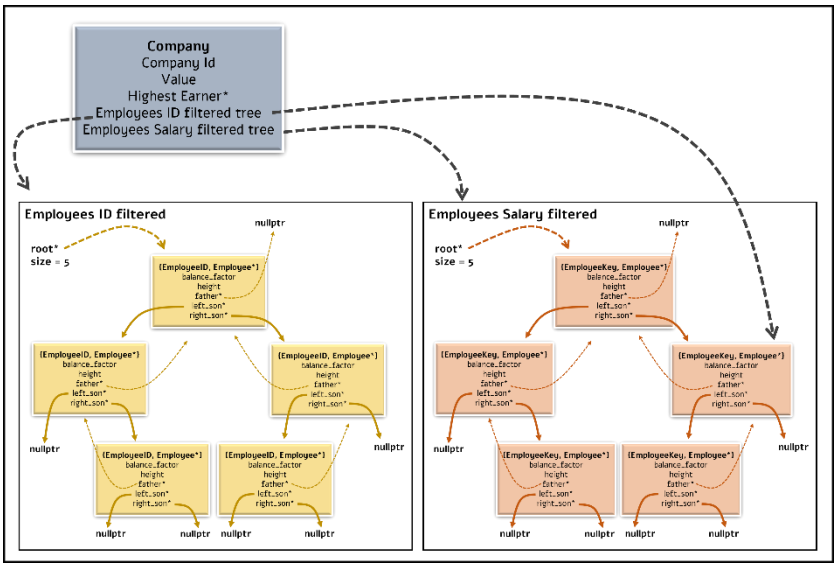
1. AVL_tree – עץ AVL כפי שנלמד בהרצאה ובתרגול. הטיפוס של הצמתים בעץ נקבעים לפי הטיפוס של $Node$. השדות של עץ:
   - $root*$ – מצביע אל האיבר הראשון בעץ.
   - $size$ – גודל העץ.

2. $Node = \{key, data\}$ – אובייקט המשמש לבניית צומת בעץ, המכיל את השדות:
   - $balance\_factor$ – גורם איזון
   - $height$ – גובה הצומת.
   - $left*, right*$ - מצביעים לבנים של הצומת.
   - $father*$ – מצביע לצומת האב שלו.

3. $Employee$ – אובייקט המשמש לייצוג עובד. השדות של $Employee$:
   - $Employee\ id$ – מזהה ייחודי של העובד.
   - $Employee\ Salary$ – המשכורת של העובד.
   - $Grade$ – דרגת העובד.
   - $Company*$ - מצביע לחברה בה עובד העובד.

4. $Employee\ key$ – אובייקט המשמש כמפתח לזיהוי עובד בעץ עובדים. השדות של $Employee\ key$:
   - $Employee\ id$ – מזהה ייחודי של העובד.
   - $Employee\ Salary$ – המשכורת של העובד.

5. $Company$ – אובייקט המשמש לייצוג חברה. השדות של $Company$:
   - $Company\ Id$ – מזהה ייחודי של החברה.
   - $Value$ – השווי של החברה.
   - $Highest\ earner*$ – מצביע לעובד בעל השכר הגבוה ביותר בחברה (והמזהה $Employee\ id$ הנמוך מבין העובדים בעלי שכר זהה).
   - $Salary\ filtered\ Employee\ tree$ – עץ עובדים בחברה, המסודר לפי $Employee\ key$, כך שהמשכורות מסודרות מהנמוך לגבוה, ועבור משכורות זהות, הסדר הוא לפי המזהים של העובדים מהגבוה לנמוך. כל צומת בעץ מכיל צמד של מפתח ($Employee\ key$) ומידע (מצביע לעובד המתאים).



   - $Id\ filtered\ Employee\ tree$ – עץ עובדים בחברה, המסודר לפי המזהה הייחודי של העובדים ($Employee\ id$ בלבד), מהנמוך לגבוה.
     כל צומת בעץ מכיל צמד של מפתח ($Employee\ Id$) ומידע (מצביע לעובד המתאים).

מבני נתונים 1 - 234218 - אביב 2022

גיליון רטוב 1

תיאור מבנה הנתונים שלנו:

מבנה הנתונים שלנו $\boldsymbol{Companies\ Manager}$ מורכב מ-5 שדות: 4 עצי AVL, ומצביע לעובד בעל המשכורת הגבוהה ביותר שהוכנס למבנה.

1. $*$ $\boldsymbol{Highest\ earner}$ – מצביע לעובד בעל השכר הגבוה ביותר מבין כל העובדים שהוכנסו למבנה (והמזהה $Employee\ id$ הנמוך מבין העובדים בעלי שכר זה).

2. $\boldsymbol{all\_employees\_salary\_filtered}$ – עץ AVL המכיל את כל העובדים שהוכנסו למבנה הנתונים. העץ ממויין לפי $Employee\ key$, כפי שתיארנו קודם לכן (סעיף 5 בעמוד הקודם, בעץ הממויין עם אותו מפתח).
עץ זה משמש עבור כל הפונקציות הדורשות מיון ראשי של העובדים על פי המשכורת שלהם, ומיון משני על פי ה-Id שלהם (כגון get all employees by salary).
בכל צומת בעץ מחזיקים מצביע אל העובד המתאים.

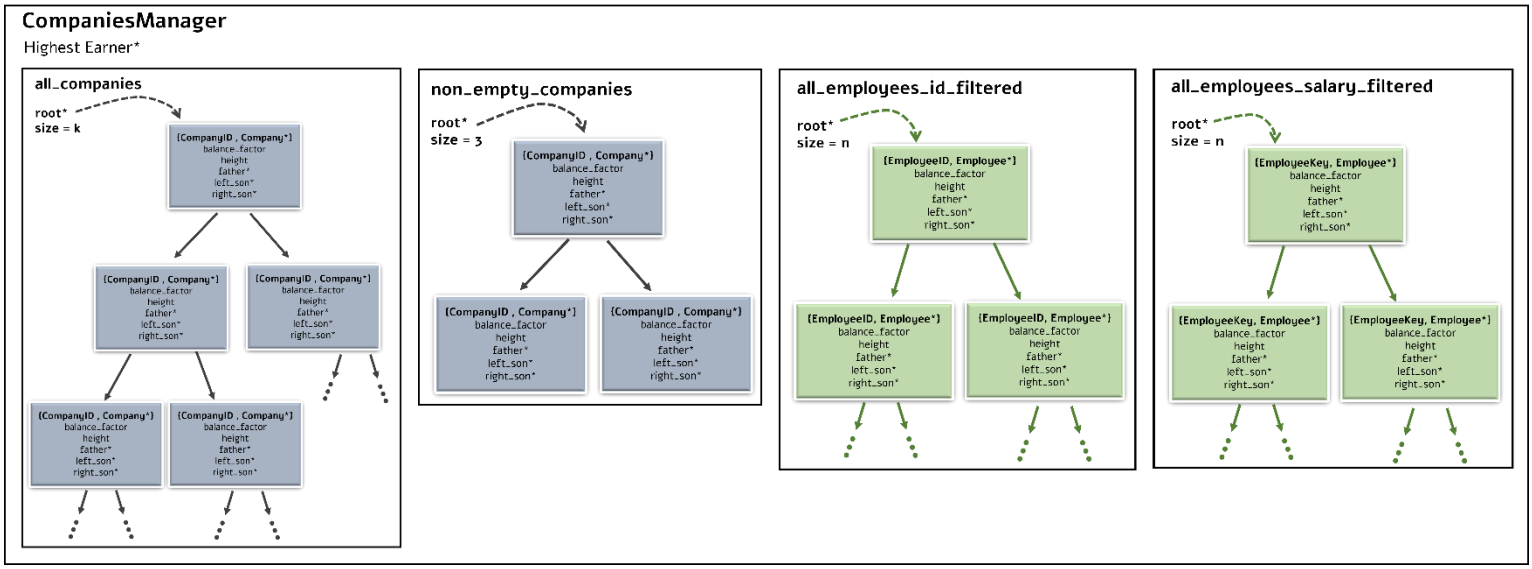
3. $\boldsymbol{all\_employees\_id\_filtered}$ – עץ AVL המכיל את כל העובדים שהוכנסו למבנה הנתונים. העץ ממויין לפי $Employee\ Id$ בלבד.
עץ זה משמש עבור כל הפונקציות הדורשות מציאת עובד על פי ה-Id שלו בלבד (כגון get num employees matching, ובדיקה האם עובד מסויים הוכנס למבנה או לא).
בכל צומת בעץ מחזיקים מצביע אל העובד המתאים.

4. $\boldsymbol{all\_companies}$ – עץ AVL שאיבריו הם חברות כפי שתואר מעלה, המכיל את כל k החברות שהתווספו למבנה. החברות בעץ ממוינות לפי ה-$Company\ Id$ מהקטן לגדול, כאשר צד שמאל יכיל את החברה בעלת ה-$Company\ Id$ הנמוך ביותר, וצד ימין יכיל את ה-$Company\ Id$ הגבוה ביותר.
בכל צומת בעץ מחזיקים מצביע אל החברה המתאימה.

5. $\boldsymbol{non\_empty\_companies}$ – עץ AVL שיכיל רק את החברות שאינן ריקות, כלומר חברות המכילות לפחות עובד אחד כל אחת. עץ זה ממוין לפי ה-$Company\ Id$ מהקטן לגדול.
בעץ זה יש לכל היותר n חברות, מאחר ש:
   - אם בחברה אין עובדים כלל, היא לא תופיע בעץ זה.
   - אם יש n עובדים, וכל אחד מהם בחברה אחרת, נקבל שיש בדיוק n חברות.
   - לכן לפי עיקרון שובך היונים, אם יש חברות בהן יש יותר מעובד אחד, אז נקבל שיש פחות מ-n חברות.

בכל צומת בעץ מחזיקים מצביע אל החברה המתאימה.

תרשים של מבנה הנתונים המלא להמחשה:



מבני נתונים 1 - 234218 - אביב 2022

גיליון רטוב 1

האלגוריתמים בהם השתמשנו:

1. **סיור** $\boldsymbol{pre\_order}$ / $\boldsymbol{in\_order}$ בעץ כפי שנלמד בכיתה – לטובת מציאת איבר בעץ.
סיבוכיות הזמן של סיור בעץ AVL היא $\log(h)$ כאשר h גובה העץ.
2. **הכנסה לעץ / הוצאה מעץ** – כפי שנלמד בכיתה, עם מימוש של גלגולים RR, LL, RL, LR.
ההוספה של איבר לעץ מתבצעת כפי שנלמד בכיתה, תוך שמירה על עץ מסודר.
סיבוכיות הזמן של הכנסה/הוצאה מעץ AVL היא $\log(h)$ כאשר h גובה העץ.
3. **מיזוג עצים** כפי שנלמד בכיתה – לטובת מימוש הפונקציה $AcquireCompany$, והעברת כל העובדים של החברה הנרכשת אל החברה הרוכשת.
מיזוג 2 עצי AVL שגודלם n_1, n_2 מתבצע בסיבוכיות זמן של $O(n_1+n_2)$, וגם בסיבוכיות מקום של $O(n_1+n_2)$.
בעת מיזוג שתי חברות, נקבל שיש בשתיהן ביחד לכל היותר n עובדים, לכן סיבוכיות זמן ומקום של לכל היותר $O(n)$.

הוכחת סיבוכיות זמן של פונקציות:

1. $\boldsymbol{Init}()$ – הפונקציה מאתחלת מבנה נתונים ריק מסוג $Companies\ Manager$ המכיל 4 עצים ריקים (2 עצי חברות ו-2 עצי עובדים, ומצביע המאותחל ל-$nullptr$).
עץ ריק מכיל מצביע לשורש שלו (שמאותחל להיות $nullptr$) וגודל העץ מאותחל להיות 0, לכן יצירת עץ ריק מתבצעת ב-$O(1)$, וכך גם יצירה של 4 עצים ריקים.
לכן בסה"כ - סיבוכיות הזמן של הפונקציה היא $O(1)$.

2. $\boldsymbol{AddCompany(void * DS,\ int\ CompanyID,\ int\ value)}$ –
תחילה מתבצע חיפוש בעץ כל החברות $all\_companies$ (המכיל לכל היותר k חברות), בהתאם לאלגוריתם חיפוש בעץ.
החיפוש מתבצע ב-$O(\log(k))$ במקרה הגרוע.
אם החברה לא קיימת בעץ, יוצרים אובייקט חדש עבור החברה עם 2 עצי עובדים ריקים, ומוסיפים אותה לעץ כל החברות $all\_companies$.
חברה מכילה את ה-$CompanyID$ שלה, 2 עצי עובדים ריקים, ומצביע לעובד שמאותחל להיות $nullptr$, לכן יצירת חברה חדשה מתבצעת ב-$O(1)$.
הוספה של החברה לעץ-כל-החברות $all\_companies$ מתבצעת ב- $O(\log(k))$ בהתאם לאלגוריתם ההכנסה לעץ. החברה מתחילה ללא עובדים כלל, ולכן לא מתווספת לעץ החברות הלא ריקות $non\_empty\_companies$.
לכן בסה"כ – סיבוכיות הזמן של הפונקציה היא $O(\log(k))$.

3. $\boldsymbol{AddEmployee(void * DS,\ int\ EmployeeID,\ int\ CompanyID,\ int\ Salary,\ int\ Grade)}$ –
תחילה מתבצע חיפוש בעץ כל העובדים הממויין לפי מזהה העובד $all\_employees\_id\_filtered$, על מנת לבדוק אם העובד כבר קיים במערכת. החיפוש מתבצע ב-$O(log(n))$.
אם העובד לא נמצא בעץ זה, אז מתבצע חיפוש בעץ כל החברות $all\_companies$ על מנת לבדוק האם החברה קיימת במבנה. החיפוש מתבצע ב-$O(log(k))$.
אם החברה אכן קיימת במבנה, אז יוצרים אובייקט חדש עבור העובד עם הפרמטרים שהתקבלו בפונקציה, וכן שומרים מצביע לחברה אליה הוא משתייך (הנוכחית שמצאנו). יצירת האובייקט מתבצעת ב-$O(1)$.
בנוסף, יוצרים אובייקט עזר $EmployeeKey$ עבור עץ העובדים הממויין לפי $EmployeeKey$, והוא מכיל את מזהה העובד והמשכורת שלו בלבד. יצירת האובייקט מתבצעת ב-$O(1)$.
מכניסים את העובד ל-2 העצים של כל העובדים במבנה, וכן ל-2 העצים של החברה בה הוא עובד. הכנסת העובד לארבעת העצים מתבצעת ב-$O(\log(n))$.
לאחר הוספת העובד לעצים, מתבצעת בדיקה האם הוא כעת העובד שמרוויח הכי הרבה בחברה/בכל המבנה, על ידי השוואה מול המרוויח הכי הרבה הנוכחי. זה מתבצע ב-$O(1)$.
במידה וזהו העובד הראשון בחברה, אז החברה מתווספת לעץ החברות שאינן ריקות. זה מתבצע ב-$O(log(k))$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(k))$, מספר קבוע של פעולות ב-$O(log(n))$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(n)+\log(k))$.

4. $-\boldsymbol{RemoveEmployee(void * DS, int\ EmployeeID)}$
תחילה מתבצע חיפוש בעץ כל העובדים הממויין לפי מזהה העובד $all\_$, על מנת לבדוק אם העובד כבר קיים במערכת.
החיפוש מתבצע ב-$O(log(n))$.
אם העובד נמצא בעץ זה, אז ניגשים לחברה בה הוא עובד, באמצעות המצביע לחברה השמור באובייקט של העובד. זה מתבצע ב-$O(1)$.
מסירים את העובד מ-2 העצים של החברה, ולאחר מכן מסירים אותו מ-2 העצים של כל-העובדים במבנה. הסרת העובד מכל אחד מהעצים מתבצעת ב-$O(\log(n))$.
לאחר הסרת העובד מכל העצים, אנו מוחקים את האובייקט של העובד (וגם את אובייקט העזר $EmployeeKey$), וזה מתבצע ב-$O(1)$.
לאחר מכן, מעדכנים את השדה של העובד בעל השכר הגבוה ביותר בחברה ממנה הוא הוסר, וגם את השדה הנ"ל במבנה הכללי ($Companies\ Manager$). העובד בעל השכר הגבוה ביותר נמצא בצומת הימני ביותר בעץ (על פי הדרך בה העובדים נשמרים בעץ), ולכן עדכון זה מתבצע על ידי שימוש בפונקציית עזר שמביאה את עובד זה, זה מתבצע ב-$O(\log(n))$.
במידה ולאחר הסרת העובד מהחברה, החברה נשארת ללא עובדים כלל, אז מסירים את החברה מהעץ של החברות שאינן ריקות. **נשים לב** שאם יש בעץ n עובדים, אז יש לנו לכל היותר n חברות לא ריקות, ולכן בעץ החברות הלא ריקות יש לכל היותר n צמתים, ולכן זה למעשה מתבצע ב-$O(\log(n))$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(n))$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(n))$.

5. $-\boldsymbol{RemoveCompany(void * DS, int\ CompanyID)}$
תחילה מתבצעת בדיקה בעץ כל החברות $all\_companies$ האם החברה קיימת במבנה. חיפוש החברה בעץ מתבצע ב-$O(log(k))$.
אם בחברה קיימים עובדים, לא ניתן להסיר אותה ומחזירים שגיאה בהתאם.
אם אין בחברה עובדים, אז היא נמצאת רק בעץ של כל החברות.
מסירים את החברה מעץ כל החברות, זה מתבצע ב-$O(log(k))$.
לאחר הסרת החברה מהעץ, אנו מוחקים את האובייקט של החברה. בשלב זה היא מכילה 2 עצים ריקים, ולכן זה מתבצע ב-$O(1)$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(k))$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(k))$.

6. $-\boldsymbol{GetCompanyInfo(void * DS,\ int\ CompanyID,\ int * Value,\ int * NumEmployees)}$
תחילה מתבצעת בדיקה בעץ כל החברות $all\_companies$ האם החברה קיימת במבנה. חיפוש החברה בעץ מתבצע ב-$O(log(k))$.
אם החברה קיימת, אנו מחזירים את הערכים הרלוונטיים בפרמטרים שהתקבלו בפונקציה:
בפרמטר $Value$ מחזירים את השווי הנוכחי של החברה.
בפרמטר $NumEmployees$ מחזירים את גודל העץ של החברה (אחד מהם, הם בעלי אותו גודל בכל מקרה).
זה מתבצע ב-$O(1)$.
סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(k))$.

7. $-\boldsymbol{GetEmployeeInfo(void * DS, int\ EmployeeID, int * EmployerID, int * Salary, int * Grade)}$
תחילה מתבצע חיפוש בעץ כל העובדים הממויין לפי מזהה העובד $all\_employees\_id\_filtered$, על מנת לבדוק אם העובד כבר קיים במערכת. החיפוש מתבצע ב-$O(log(n))$.
אם העובד קיים, אנו מחזירים את הערכים הרלוונטיים בפרמטרים שהתקבלו בפונקציה:
בפרמטר $EmployerID$ מחזירים את מזהה החברה, באמצעות המצביע לחברה ששמור באובייקט של העובד.
בפרמטר $Salary$ מחזירים את המשכורת הנוכחית של העובד.
בפרמטר $Grade$ מחזירים את הדרגה הנוכחית של העובד.

כל אלו מתבצעים ב-$O(1)$.
סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(n))$.

8. $-\ \boldsymbol{IncreaseCompanyValue(void\ *DS, int\ CompanyID, int\ ValueIncrease)}$
תחילה מתבצעת בדיקה בעץ כל החברות $all\_companies$ האם החברה קיימת במבנה. חיפוש החברה בעץ מתבצע ב-$O(log(k))$.
אם החברה קיימת, מעדכנים את השדה של שווי החברה. זה מתבצע ב-$O(1)$.
סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(k))$.

9. $-\ \boldsymbol{PromoteEmployee(void * DS, int\ EmployeeID, int\ SalaryIncrease, int\ BumpGrade)}$
תחילה מתבצע חיפוש בעץ כל העובדים הממויין לפי מזהה העובד $all\_employees\_id\_filtered$, על מנת לבדוק אם העובד כבר קיים במערכת. החיפוש מתבצע ב-$O(log(n))$.
אם העובד קיים, אז ניגשים לחברה בה הוא עובד באמצעות המצביע לחברה השמור באובייקט של העובד. זה מתבצע ב-$O(1)$.
מסירים את העובד מ-2 העצים של החברה, ומ-2 העצים של כל העובדים במבנה, זה מתבצע ב-$O(log(n))$.
מעדכנים את השדה של המשכורת של העובד, ובמידת הצורך מעדכנים גם את השדה של הדרגה שלו. זה מתבצע ב-$O(1)$.
(**נשים לב** שהאובייקט $EmployeeKey$ נמחק בעת הסרת העובד מהעצים, ולאחר עדכון השדות של העובד ניצור אובייקט חדש מתאים עם המשכורת המעודכנת ונכניס אותו מחדש לעצים).
כעת נכניס את העובד חזרה ל-2 העצים של כלל העובדים במבנה, ואל 2 העצים של החברה בה הוא עובד. זה מתבצע ב-$O(log(n))$.
לאחר הוספת העובד לעצים, מתבצעת בדיקה האם הוא כעת העובד שמרוויח הכי הרבה בחברה/בכל המבנה, על ידי השוואה מול המרוויח הכי הרבה הנוכחי. זה מתבצע ב-$O(1)$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(n))$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(n))$.

10. $-\ \boldsymbol{HireEmployee(void\ *DS, int\ EmployeeID, int\ NewCompanyID)}$
תחילה מתבצע חיפוש בעץ כל העובדים הממויין לפי מזהה העובד $all\_employees\_id\_filtered$, על מנת לבדוק אם העובד כבר קיים במערכת. החיפוש מתבצע ב-$O(log(n))$.
שולפים את מזהה החברה הנוכחית בה הוא עובד באמצעות המצביע השמור, ומשווים אותו למזהה של החברה החדשה $NewCompanyID$. זה מתבצע ב-$O(1)$.
במידה והם שונים, החברה החדשה יכולה להעסיק את העובד.
מחפשים בעץ כל החברות $all\_companies$ האם החברה החדשה קיימת במבנה. חיפוש החברה בעץ מתבצע ב-$O(log(k))$.
אם החברה קיימת, מסירים את העובד מ-2 העצים של החברה בה הוא עובד (ומעדכנים את השדה של המרוויח הכי הרבה בהתאם, כפי שתיארנו קודם לכן, וגם במידה והוא היה העובד היחיד מסירים את החברה מעץ החברות שאינן ריקות), ומוסיפים את העובד ל-2 העצים של החברה החדשה. במידת הצורך מעדכנים את השדה של העובד המרוויח הכי הרבה בחברה החדשה. בנוסף, מעדכנים את המצביע השמור באובייקט של העובד עם מצביע לחברה החדשה שלו.
הסרת העובד והוספתו לעצים מתבצעת ב-$O(log(n))$, עדכון השדות מתבצע ב-$O(1)$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(k))$, מספר קבוע של פעולות ב-$O(log(n))$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(log(n) + \log(k))$.

11. $-\ \boldsymbol{AcquireCompany(void\ *DS, int\ AcquirerID, int\ TargetID, double\ Factor)}$
תחילה מתבצע חיפוש בעץ כל החברות $all\_companies$ האם 2 החברות קיימות במבנה. חיפוש החברות בעץ מתבצע ב-$O(log(k))$.
אם 2 החברות קיימות, בודקים האם השווי של החברה הרוכשת $AcquirerID$ גדול פי 10 לפחות מהשווי של החברה הנרכשת $TargetID$. זה מתבצע ב-$O(1)$.

אם זה מתקיים, מחשבים את השווי החדש של החברה הרוכשת בהתאם לנוסחה שקיבלנו, ומעדכנים את השדה של השווי. זה מתבצע ב-$O(1)$.
משווים בין העובדים שמרוויחים הכי הרבה בכל חברה, ומעדכנים בחברה הרוכשת לפי הצורך. זה מתבצע ב-$O(1)$.
כעת ממזגים את העצים של 2 החברות (ממזגים את העצים של החברה הנרכשת לתוך העצים של החברה הרוכשת, כל עץ עם העץ שמתאים לו מבחינת סוג) בהתאם לאלגוריתם של מיזוג העצים שנלמד בכיתה. זה מתבצע ב-$O(n_{acquirer} + n_{target})$.
לאחר מיזוג העצים, עוברים על 2 עצי העובדים של החברה הרוכשת ומעדכנים לכל העובדים את המצביע לחברה כך שיכיל את החברה הרוכשת. זה מתבצע ב-$O(n_{acquirer} + n_{target})$.
מסירים את החברה הנרכשת מ-2 עצי החברות של המבנה, זה מתבצע ב-$O(log(k))$.
לבסוף מוחקים את האובייקט של החברה הנרכשת (בתהליך זה גם נמחקים העצים של החברה הנרכשת שמכילים מצביעים לעובדים, ואובייקטי עזר באמצעות ההורס של העצים). זה מתבצע ב-$O(n_{target})$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(k))$, מספר קבוע של פעולות ב-$O(1)$, ומספר קבוע של פעולות ב-$O(n_{acquirer} + n_{target})$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא:
$O(\log(k) + n_{acquirer} + n_{target})$.

12. $-\boldsymbol{GetHighestEarner(void * DS, int\ CompanyID, int * EmployeeID)}$
   - אם $CompanyID < 0$, מחזירים את השדה של העובד שמרוויח הכי הרבה השמור במבנה $Companies\ Manager$. זה מתבצע ב-$O(1)$.
   - אם $CompanyID > 0$, מחפשים את החברה בעץ כל החברות $all\_companies$, זה מתבצע ב-$O(log(k))$.
     לאחר מציאת החברה, מחזירים את השדה של העובד שמרוויח הכי הרבה בחברה, זה מתבצע ב-$O(1)$.
     בסה"כ סיבוכיות הזמן של הפונקציה במקרה זה היא $O(log(k))$.
   - אחרת, מחזירים שגיאה בהתאם.

13. $\boldsymbol{GetAllEmployeesBySalary(void * DS, int\ CompanyID, int ** Employees, int * -NumOfEmployees)}$
   - אם $CompanyID < 0$, מבצעים את התהליך שנתאר בהמשך על העץ של כל העובדים במבנה שממויין לפי משכורות העובדים $all\_employees\_salary\_filtered$.
   - אם $CompanyID > 0$, מחפשים את החברה בעץ כל החברות $all\_companies$, זה מתבצע ב-$O(log(k))$.
     לאחר מכן, מבצעים את התהליך שנתאר בהמשך על העץ של כל העובדים בחברה שממויין לפי משכורות העובדים $employees\_salary\_filtered$.
   - התהליך שמבצעים:
     תחילה מחזירים בפרמטר $NumOfEmployees$ את מספר העובדים בעץ (בכל המבנה או בחברה, לפי הצורך). זה מתבצע ב-$O(1)$.
     אם קיימים עובדים (לא 0) אז יוצרים מערך בגודל המתאים, ולאחר מכן מבצעים סיור $reverse\ in\ order$ על העץ (בכל המבנה או בחברה, לפי הצורך) במהלך הסיור, שומרים את המזהה $EmployeeId$ של העובד הנוכחי, בתא המתאים במערך שיצרנו. הסיור בעץ מתבצע ב-$O(n_{company})$ או $O(n)$ בהתאם לחברה מסוימת או לכל המבנה.
     נשים לב שבהתאם לכך שהסיור הוא $in\ order$ בסדר הפוך, אנו מקבלים לבסוף שהעובדים במערך מסודרים לפי סדר יורד במשכורות, וסדר עולה במזהה (במקרה של שכר זהה).
     בסה"כ מתבצע מספר קבוע של פעולות ב-$O(log(k))$ (רק במקרה של חברה מסוימת), מספר קבוע של פעולות ב-$O(1)$, ומספר קבוע של פעולות ב-$O(n_{company})$ או $O(n)$ בהתאם לחברה מסוימת או לכל המבנה, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא:
     עבור חברה מסוימת: $O(\log(k) + n_{company})$
     עבור כל המבנה: $O(n)$

14. $\boldsymbol{GetHighestEarnerInEachCompany(void * DS, int\ NumOfCompanies, int ** Employees)}$ –
תחילה בודקים האם הגודל של עץ החברות שאינן ריקות גדול יותר מהפרמטר שהתקבל $NumOfCompanies$. זה מתבצע ב-$O(1)$.
אם כן, יוצרים מערך בגודל $NumOfCompanies$.
מבצעים סיור $in\ order$ על עץ החברות שאינן ריקות, ובכל שלב בסיור שומרים את מזהה העובד שמרוויח הכי הרבה בחברה הנוכחית, בתא המתאים במערך שיצרנו.
הסיור מתחיל בחברה בעלת המזהה $CompanyID$ הנמוך ביותר, שנמצאת בעלה השמאלי ביותר בעץ, ולכן גישה לחברה זו מתקבלת ב-$O(\log(k))$.
נשים לב שעוברים בסיור על בדיוק $NumOfCompanies$ חברות בעץ, גם אם יש בו יותר חברות, ולכן הסיור מתבצע ב-$O(NumOfCompanies)$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O\big(log(k)\big)$, מספר קבוע של פעולות ב-$O(1)$, ומספר קבוע של פעולות ב-$O(NumOfCompanies)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא:
$O(log(k) + NumOfCompanies)$.

15. $\boldsymbol{GetNumEmployeesMatching(void * DS, int\ CompanyID, int\ MinEmployeeID, int\ MaxEmployeeId, int\ MinSalary, int\ MinGrade, int * TotalNumOfEmployees, int * NumOfEmployees)}$ –

- אם $CompanyID < 0$, מבצעים את התהליך שנתאר בהמשך על העץ של כל העובדים במבנה שממויין לפי המזהים של העובדים $all\_employees\_id\_filtered$.
- אם $CompanyID > 0$, מחפשים את החברה המתאימה בעץ כל החברות $all\_companies$, זה מתבצע ב-$O\big(log(k)\big)$.

לאחר מכן, מבצעים את התהליך שנתאר בהמשך על העץ של כל העובדים בחברה המסוימת, שממויין לפי המזהים של העובדים $employees\_id\_filtered$.

- התהליך שמבצעים:

תחילה מחפשים בעץ את העובד עם המזהה הקרוב ביותר ל-$MinEmployeeID$, נסמן אותו ב-min_emp. זה מתבצע ב-$O\big(log(n)\big)$ או $O\left(log\big(n_{company}\big)\right)$ בהתאם לחברה מסוימת/כל המבנה.
באותו אופן מחפשים בעץ את העובד עם המזהה הקרוב ביותר ל-$MaxEmployeeID$, נסמן אותו ב-max_emp. זה מתבצע ב-$O\big(log(n)\big)$ או $O\left(log\big(n_{company}\big)\right)$ בהתאם לחברה מסוימת/כל המבנה.
מאתחלים 2 מונים – אחד שסופר את כמות העובדים בטווח הנתון, ושני שסופר את כמות העובדים בטווח זה העומדים בתנאים של המשכורת והדרגה.
כעת מבצעים סיור $in\ order$ החל מעובד min_emp ועד לעובד max_emp, כך שבכל שלב בסיור בודקים האם המזהה של העובד נמצא בטווח שבין $MinEmployeeID$ ל-$MaxEmployeeID$ (לשם וידוא), ואם כן מגדילים את מונה העובדים בטווח. זה מתבצע ב-$O(1)$.
בנוסף, בודקים האם המשכורת של העובד גדולה מ-$MinSalary$, והאם הדרגה של העובד גדולה מ-$MinGrade$. אם כן, מגדילים את המונה של כמות העובדים העונים על התנאי לשכר/דרגה.
בדיקות אלו מתבצעות ב-$O(1)$.
בסיום הסיור, מחזירים בפרמטר $\boldsymbol{TotalNumOfEmployees}$ את מספר העובדים במונה הראשון, ובפרמטר $NumOfEmployees$ מחזירים את מספר העובדים במונה השני. זה מתבצע ב-$O(1)$.
נשים לב שמספר העובדים שנמצאים בטווח הנ"ל הוא $TotalNumOfEmployees$, ולכן הסיור מתבצע ב-$O(TotalNumOfEmployees)$.
בסה"כ מתבצע מספר קבוע של פעולות ב-$O\big(log(k)\big)$ (במקרה של חברה מסוימת), מספר קבוע של פעולות ב-$O\left(log\big(n_{company}\big)\right)$ (במקרה של חברה מסוימת) או ב-$O\big(log(n)\big)$ (במקרה של כל המבנה), מספר קבוע של פעולות ב-$O(1)$, ומספר קבוע של פעולות ב-$O(TotalNumOfEmployees)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא:
עבור חברה מסוימת: $O\big(log(k) + log\big(n_{company}\big) + TotalNumOfEmployees\big)$
עבור כל המבנה: $O(log(n) + TotalNumOfEmployees)$

16. $-\boldsymbol{Quit(void ** DS)}$

תחילה מבצעים סיור $in\ order$ על העץ של כל העובדים במבנה (אין חשיבות למי מהעצים בוחרים) ומוחקים את האובייקט של כל אחד מהעובדים, וכך כל המצביעים לעובדים בכל העצים של העובדים מחזיקים כעת ערך זבל. זה מתבצע ב-$O(n)$.
לאחר מכן מבצעים את אותו תהליך על עץ כל החברות ומוחקים את כל האובייקטים של החברות. סיור על עץ כל החברות מתבצע ב-$O(k)$.
נשים לב שבשלב זה, בעת מחיקת חברה אנו מוחקים גם את 2 העצים שלה, בכל החברות יחד יש n עובדים, ולכן בסה"כ מתבצעות $O(n+k)$ מחיקות.
נסביר מדוע: הריסה של כל חברה דורשת הריסה של 2 עצי העובדים שלה, שמכילים מצביעים לעובדים (ערך זבל). בכל חברה c_i $(1 \le i \le k)$ יש n_{c_i} עובדים, ומתקיים: $n_{c_1} + n_{c_2} + \cdots + n_{c_k} = n$ (עובד משתייך לחברה אחת בלבד) לכן בכל חברה נצטרך למחוק עץ שמכיל רק n_{c_i} צמתים, ובסה"כ בכל עצי העובדים של כל החברות נצטרך להסיר לכל היותר n צמתים.
מספר הפעולות הכולל בהריסת עץ כל החברות שווה לסכום של מספר הפעולות שיש לבצע עבור כל אחת מהחברות בנפרד:
מחיקה של צומת אחד מתבצעת ב-$O(1)$ ולכן הריסת עץ-עובדים בגודל n_{c_i} של החברה ה-i מתבצע ב-$O(n_{c_i})$, ואז הריסת החברה עצמה מתבצעת גם כן ב-$O(1)$, ולכן הריסת כל החברות דורשת בסה"כ $O\left(\sum_{i=1}^{k} n_{c_i}\right) = O(n)$ פעולות).
לבסוף, נשאר לנו למחוק את 4 העצים של המבנה (שכעת לא מכילים אובייקטים של עובד/חברה, אלא רק שדות של עץ).
מחיקת עץ עובדים בגודל n מתבצעת ב-$O(n)$, מחיקת עץ חברות בגודל k מתבצעת ב-$O(k)$, ולכן מחיקת 4 העצים מתבצעת ב-$O(n+k)$ (כי כעת בעצי החברות אין יותר מצביעים לחברות עצמן).
בסה"כ מתבצע מספר קבוע של פעולות ב-$O(n)$, מספר קבוע של פעולות ב-$O(k)$, ומספר קבוע של פעולות ב-$O(1)$, ולכן סיבוכיות הזמן הכוללת של הפונקציה היא $O(n+k)$.

<u>הוכחת סיבוכיות מקום:</u>

- כל אחד מהעובדים שאנו יוצרים מכיל מספר קבוע של שדות, ולכן כל עובד הוא בגודל קבוע, כלומר $O(1)$.
- כל חברה שאנו יוצרים מכילה 2 עצים בגודל לכל היותר n, ולכן גודל חברה הוא לכל היותר $O(n)$, ובכל החברות ביחד יש בסה"כ n עובדים, לכן אם קיימת חברה שמכילה n עובדים, אז כל היתר ריקות והן בגודל $O(1)$.
  כלומר כל החברות יחד תופסות $O(k+n)$ מקום.
- עץ החברות יכיל בכל שלב בתוכנית k צמתים בגודל קבוע (כי מצביע לחברה הוא בעל גודל קבוע) בהתאם ל-k החברות שהוכנסו עד אותו שלב, כלומר $O(k)$.
- עץ החברות הלא ריקות יכיל בכל שלב לכל היותר k צמתים בגודל קבוע, כי יש לכל היותר k חברות במבנה, ולכן גם לכל היותר k חברות לא ריקות, ולכן שוב $O(k)$.
- 2 העצים של כל העובדים במבנה (הממויין לפי מזהה ID, והממויין לפי $salary$) יכילו בכל שלב n צמתים בגודל קבוע, בהתאם ל-n העובדים שהתווספו למבנה עד אותו שלב, כלומר $O(n)$.
- עבור כל חברה, 2 עצי העובדים של החברה יכילו רק את העובדים השייכים אליה, וכל עובד שייך בדיוק לחברה אחת, לכן כל n העובדים שהתווספו למבנה "יתחלקו" בין עצי העובדים של כל החברות.

עבור כל חברה c_i, $1 \le i \le k$, עץ העובדים שלה יכיל n_{c_i} עובדים, ומתקיים:
$n_{c_1} + n_{c_2} + \cdots + n_{c_k} = n$.
כלומר: $O\left(n_{c_1} + n_{c_2} + \cdots + n_{c_k}\right) = O(n)$.
כלומר בסה"כ: $O(k+n)$.