תרגיל רטוב 1 - מבני נתונים 1 אביב 2022

מבנה הנתונים שלנו יכיל:

1) עץ AVL גנרי: עץ AVL שכל צומת בו יכיל מפתח לפיו נמיין את האיברים, ושדה נוסף Value שיכיל מידע נוסף עבור כל מפתח. בנוסף, לכל צומת נשמור מצביע לבנים שלו וגם להורה שלו. העץ וכל הפעולות שלו ימומשו בדיוק כפי שלמדנו בהרצאות ובתרגולים.
2) EmployeeKey: מבנה המכיל מספר זהות ושכר של עובד מסויים. מבנה זה יהיה מפתח בעצי העובדים הממויינים לפי השכר, ובמידה ושני עובדים הם בעלי אותו שכר אזי העובד שמספר זהותו נמוך יותר נחשב גדול יותר.
3) Employee: מבנה שמכיל את כל המידע ההכרחי לכל עובד. יהיה בו מספר זהות, שכר ודרגתו של העובד, ובנוסף נשמור גם מצביע למבנה החברה המעסיקה את עובד זה.
4) Company: מבנה של החברה המכיל המספר והערך שלה. והוא גם יכיל מצביע לעץ של העובדים שלה ממויין לפי מספרי הזהות שלהם, ומצביע לעוד עץ של העובדים שלה אך ממויין לפי השכר שלהם. ולבסוף נשמור כאן גם מצביע לצומת העובד המרוויח הכי יותר בחברה זו (במידה ויש יותר מאחד עם שכר זה נבחר את העובד בעל מספר הזהות הנמוך ביותר).
5) EmployeeManager: יכיל עץ AVL של החברות הריקות (כלומר החברות שלא מעסיקות אף עובד), עץ AVL של החברות שעובד בהן עובד אחד לפחות, ומצביע לצומת אחד של חברה שלא באמת קיימת (נקרא לה dummy) שתכיל את כל העובדים שנכניס למערכת.

EmployeeKey
int salary
int id

AVLTree<Key,Value, ptr>
shared_ptr<AVLNode<Key, Value, ptr>> root
int size

AVLNode<Key, Value, ptr>
Key key
Ptr<Value> value
int height
shared_ptr<AVLNode<Key, Value, ptr>> left
shared_ptr<AVLNode<Key, Value, ptr>> right
weak_ptr<AVLNode<Key, Value, ptr>> parent

Employee
int id
int salary
int grade
weak_ptr<Company> employer

Company
int id
private int value
shared_ptr<AVLTree<EmployeeKey, Employee, weak_ptr>> employees_by_id
shared_ptr<AVLTree<EmployeeKey, Employee, weak_ptr>> employees_by_salary
weak_ptr<Employee> highest_earner

EmployeeManager
uniqe_ptr<AVLTree<int, Company, shared_ptr>> empty_companies
uniqe_ptr<AVLTree<int, Company, shared_ptr>> non_empty_companies
shared_ptr<Company> dummy

סיבוכיות המקום:

המבנה שלנו יכיל שני עצי AVL של החברות כאשר כל חברה מיוצגת ע"י המבנה Company אשר סיבוכיות המקום שלו היא O(1). בהינתן שקיימות k חברות במערכת אזי שני עצים אלה יכילו 2k צמתים. בנוסף לכך, כל עובד שיוכנס למערכת יישמר כצומת ב-4 עצי AVL:

1. בחברה המתאימה, בעץ הממויין לפי מספרי הזהות של העובדים
2. בחברה המתאימה, בעץ הממויין לפי השכר של העובדים
3. בחברת dummy, בעץ הממויין לפי מספרי הזהות של העובדים
4. בחברת dummy, בעץ הממויין לפי השכר של העובדים

ולכן בהינתן כי הכנסנו למערכת n עובדים, אזי בסה"כ יהיו לנו 4n צמתים של עובדים, כאשר כל צומת של עובד מיוצג ע"י Employee ולעתים גם EmployeeKey כאשר סיבוכיות המקום של שניהם קבועה.

⇐ סיבוכיות המקום היא O(4n+2k) = O(n+k)

מימוש הפעולות:

בכל פונקציה נתחיל בלבדוק את הפרמטרים המועברים ואם הם חוקיים לפי הדרישות הנתונות (אם לא אזי נחזיר INVALID_INPUT ונסיים). כמו כן עבור הקצאה או הכנסה נבדוק שהפעולה התבצעה בהצלחה, אחרת נחזיר ALLOCATION_ERROR ונסיים).

Init:

נאתחל עץ AVL ריק (כפי שלמדנו בהרצאה) שמייצג את החברות שלא מכילות עובדים (empty_companies)- בו המפתח של כל צומת הוא מסוג int שמייצג מספר החברה, וה-Value הוא מסוג Company. ובאותו אופן נאתחל את non_empty_companies שהוא העץ של החברות שמעסיקות לפחות עובד אחד.

נרצה לאתחל גם את dummy שהוא מסוג Company. נאתחל בו את העץ של employees_by_id שיהיה עץ avl ריק בו כל מפתח של צומת יהיה int המייצג מספר זהות של כל עובד וה-Value הוא מסוג Employee. נאתחל ב-dummy גם העץ של employees_by_salary שיהיה עץ avl ריק בו כל מפתח של צומת יהיה מסוג EmployeeKey וה-Value הוא מסוג Employee. נאתחל את המצביע ל-highest_earner שיהיה ריק.

סיבוכיות הזמן: אתחלנו מספר קבוע של עצים ומצביעים ריקים ולכן סיבוכיות הזמן היא O(1).

AddCompany:

ראשית נחפש את החברה CompanyID בשני עצי החברות, אם מצאנו חברה כזאת נחזיר FAILURE מכיוון שהיא לא אמורה להיות קיימת במערכת. לאחר מכן ניצור Company שמכיל את המידע הנתון על החברה ונאתחל את העצים והמצביעים שבו להיות ריקים. נוסיף את Company כ-Value של צומת שמפתחו CompanyID לעץ empty_companies. אם כל הפעולות התבצעו בהצלחה, נחזיר SUCCESS.

סיבוכיות הזמן: בדיקת קיום החברה במערכת ע"י חיפושה לוקח O(logk) זמן (חיפוש צומת בעץ AVL מתבצע בסיבוכיות זמן לוגריתמית, כפי שלמדנו בהרצאה), והכנסת החברה לעץ גם כן תיקח O(logk) כפי שלמדנו בהרצאה. אתחול העצים/מצביעים יקח O(1) ולכן סיבוכיות הזמן הכוללת של הפעולה היא O(logk) כאשר k הוא מספר החברות הקיימות במערכת.

AddEmployee:

נחפש את העובד באחד עצי העובדים שנמצאים ב-dummy (ה-dummy מכיל את כל עובדי המערכת), במידה ומצאנו עובד עם מספר הזהות הנתון, נחזיר FAILURE ונסיים. לאחר מכן נחפש גם את החברה עם המספר CompanyID גם ב-empty_companies וגם ב-non_empty_companies, אם לא מצאנו אותה באף אחד מהעצים, נחזיר FAILURE. אחרת, נכניס לשני עצי העובדים שלה את EmployeeID והמידע

המתאים שלו (שכר, דרגה, וגם מצביע למבנה ה-Company שתעסיק אותו וכו') ונכניס את העובד גם לשני עצי העובדים של dummy (וכך גם מכניסים אותו למאגר כל העובדים שבמערכת).
לאחר מכן, עבור החברה התאימה נעדכן את השדה highest_earner שיכיל את העובד שהוא המקסימום של העץ employees_by_salary לאחר שהוספנו את העובד החדש. ובאותו אופן העדכן גם את ה-highest_earner של ה-dummy.
חיפוש והכנסה לעץ AVL מתבצעים בסיבוכיות זמן O(logn)- לפי המימוש שלמדנו בכיתה.
סיבוכיות הזמן: חיפשנו את העובד בעץ כל עובדי המערכת וגם הכנסנו אותו אליו שזה מתבצע ב-O(logn) וגם חיפשנו את החברה כדי להכניס את העובד אליה וזה מתבצע ב-O(logk), מציאת מקסימום בעץ היא ללכת לבן הימני עד שנגיע ל-nullptr ולכן סיבוכיות הזמן שלו היא כגובה העץ: O(logn)
ולכן סיבוכיות הזמן הכוללת היא O(logn+logk).

RemoveEmployee:

נחפש את העובד במערכת (כלומר בעץ employees_by_id שנמצא ב-dummy), אם הוא לא קיים, מחזיר FAILURE. אם הוא כן קיים, נלך לחברה שמעסיקה אותו - כאשר המצביע שלה נמצא במבנה של ה-Employee ולכן לא נצטרך לחפש אותה בעץ החברות, ונמחק את העובד משני עצי העובדים שלה. נבדוק אם אחרי מחיקת עובד זה נותרו בחברה עובדים, אם לא, נמחק אותה מהעץ non_empty_companies ונכניס אותה לעץ empty_companies. לאחר מכן נמחק את העובד משני עצי העובדים של dummy וכך נמחק אותו ממאגר כל העובדים שבמערכת.
נעדכן את ה-highest_earner של החברה שהעובד עבד בה לפני מחיקתו, ואת ה-highest_earner של dummy ע"י מציאת מקסימום של העץ המעודכן (בדיוק כפי שעשינו ב-AddEmployee)
סיבוכיות הזמן: חיפוש העובד ומחיקתו מעצי ה-avl מתבצעות בסיבוכיות זמן O(logn) (וגם מציאת המקסימום החדש בעץ) לפי המימוש שלמדנו בכיתה. ולכן סיבוכיות זמן הפעולה היא O(logn).

RemoveCompany:

נחפש את החברה לפי מספר הזהות שלה ב-empty_companies, אם היא לא נמצאת שם, נחזיר FAILURE מכיוון שגם אם היא נמצאת בעץ האחר של החברות שלא ריקות, זה אומר שיש לה עובדים ואסור למחוק חברה שמעסיקה עובדים. אחרת, נמחק את החברה מעץ ה-empty_companies ונסיים.
סיבוכיות הזמן: מחיקת איבר מעץ avl מתבצעת בסביכויות לוגריתמית. כאן חיפשנו את החברה ומחקנו אותה ולכן סיבוכיות הזמן הכוללת היא O(logk).

GetCompanyInfo:

נחפש את החברה בעצים empty_companies ו-non_empty_companies. אם לא מצאנו אותה באף אחד מהם נחזיר FAILURE. אם כן אזי כל האינפורמציה שלה נמצאת ב-value שבצומת שלה בעץ. נוציא מה-value שלה את השווי שלה ונשמור אותו במצביע המתאים. לאחר מכן ניקח את ה-size של אחד עצי העובדים שלה (יהיה להם אותו size כי אלה אותם עובדים רק מסודרים בצורה שונה בעץ) וגם את ה-size נשמור במצביע המתאים (שהוא מספר העובדים שלה).
סיבוכיות הזמן: סיבוכיות הזמן תהיה O(logk) מפני שחיפשנו את החברה בעצי ה-avl של המערכת. הוצאת המידע היא בסביוכיות זמן O(1) מפני שכל המידע נמצא במשתנים נגישים ולכן היא לא משפיעה על הסיבוכיות הכוללת של הפעולה.

GetEmployeeInfo:

נחפש את העובד בעץ dummy->employees_by_id ב-O(logn) ואם הוא לא קיים נחזיר FAILURE ונסיים. אם הוא כן קיים אזי נקבל את ה-value שלו מהעץ ושם נוציא את השכר והדרגה וגם את מספר החברה שהוא עובד בה (שקיים במצביע ל-Company שנמצא ב-value של כל עובד) ונשים את מספרים אלה במצביעים המתאימים.

סיבוכיות הזמן: חיפוש העובד בעץ avl מתבצע בסיבוכיות זמן O(logn) כפי שלמדנו בכיתה.

IncreaseCompanyValue:

תחילה נחפש את החברה בשני עצי החברות שבמערכת על מנת לוודא שהיא קיימת. נקבל את המצביע למידע הנוסף שנמצא בצומת שלה באחד העצים וניגש לשדה ה-Value ונשנה אותו.
סיבוכיות הזמן: חיפוש העובד בעץ avl מתבצע בסיבוכיות זמן O(logk) כפי שלמדנו בכיתה כאשר עדכון שדה הvalue הוא O(1).

PromoteEmployee:

נחפש את העובד ב-dummy->employees_by_id ואם הוא לא קיים נחזיר FAILURE.
נלך למצביע של החברה שנמצא בצומת של העובד (לאחר שמצאנו אותו בעץ) ונמחק מעצי העובדים שלו את העובד ונוסיף אותו מחדש אליהם עם ה-salary וה-grade החדש. בנוסף, עבור החברה שמצאנו, נעדכן את ה-highest_earner ע"י חיפוש המקסימום בעץ employees_by_salary המעודכן. נעשה את אותו דבר גם בעצי העובדים של ה-dummy וה-highest_earner שלו.
סיבוכיות הזמן: אנו מחפשים, מוציאים ומכניסים את העובד לחברה (שיש לנו מצביע אליה בצומת של העובד ולכן לא נצטרך לחפש אותה בעצי החברות) וגם ל-dummy שמכיל עצי כל העובדים שבמערכת. במקרה הכי גרוע, כל העובדים שבמערכת עובדים בחברה שהעובד שלנו נמצא בה, כלומר יש בה n עובדים ולכן כל הפעולות הנ"ל מתבצעות בסיבוכיות זמן O(logn) כפי שלמדנו בכיתה. גם אם לא כל העובדים עובדים בחברה הנ"ל, עצי ה-dummy יכילו את כל n עובדי המערכת ולכן סיבוכיות הזמן הכוללת של הפעולה היא O(logn).

HireEmployee:

נחפש את העובד והחברה החדשה בעצים המתאימים. אם הם לא קיימים נחזיר FAILURE. בנוסף, נחפש את העובד בעץ העובדים של החברה, אם הוא קיים שם נחזיק FAILURE גם.
נסיר את העובד מהמערכת בעזרת RemoveEmployee ונכניס אותו בחזרה בעזרת AddEmployee אך בתור CompanyID נעביר לפונקציה את מספר החברה החדשה שרוצה להעסיק אותו. שאר הפרמטרים ישארו זהים ונוציא אותם מהצומת שהעובד נמצא בו בעץ העובדים : dummy->employees_by_id.
סיבוכיות הזמן: חיפוש העובד בעץ העובדים: O(logn) וחיפוש החברה בעצי החברות: O(logk).
הפונקציה AddEmployee תתבצע בסיבוכיות הכי גדולה ולכן סיבוכיות הפעולה הכוללת היא O(logk+logn) - כסיבוכיות פעולות החיפוש, ההכנסה, וההוצאה שמתבצעות בפונקציות שהשתמשנו בהן.

AcquireCompany:

נחפש את שתי החברות בעצי החברות של המערכת ואם אחת מהן לא קיימת נחזיר FAILURE. לאחר מכן נמזג את עצי ה-employees_by_id של שתי החברות ע"י האלגוריתם שלמדנו בתרגול: בעזרת סיור inorder נעתיק כל עץ למערך ואז נמזג את מערכים אלו בעזרת merge כלומר עם שני איטרטורים (אחד לכל מערך) וכל פעם נכניס למערך הממוזג את האיבר הקטן יותר ונקדם את האיטרטורים המתאימים. לאחר מכן נרצה להפוך את המערך הממויין בחזרה לעץ. נתחיל בלשים את הערך שבתא שבאמצע המערך כשורש העץ, בבן הימני נשים את אמצע תת המערך הימני ואת הבן השמאלי נשים בו אמצע תת המערך השמאלי. וכך נמשיך באופן רקורסיבי על תתי המערכים. את העץ הממוזג נשמור בעץ החברה בעלת המספר AcquirerID. נבצע את אותה הפעולה גם עבור עצי ה-employees_by_salary של החברות.
אחר כך נמחק את עובדי החברה TargetID ע"י ביצוע הפעולה treeClear לשני עצי העובדים שלה.
לאחר מכן נעבור על המערכים הממוזגים של העובדים, ונשנה את שדה ה-employer של כל אחד מהם שיהיה החברה עם המספר AcquirerID.

סיבוכיות הזמן: חיפוש החברות : O(logk). מיזוג כל שני עצי עובדים לפי השיטה שלמדנו בתרגול : O(n(AcquirerID)+n(TargetID)) וזאת מכיוון שאנו עוברים על כל איברי שני העצים בעזרת סיור inorder, ולאחר מכן ממזגים את המערכים בעזרת merge sort שזה גם מתבצע בסיבוכיות זמן של סכום גודלי המערכים, ולבסוף שוב עוברים על כל איברי במערך הממוין ומכניסים אותם ל"עץ ריק". ולכן סיבוכיות הזמן הכוללת של הפעולה היא : O(logk + n(AcquirerID) + n(TargetID))

GetHighestEarner:

אם CompanyID>0, אזי נחפש אותה בעץ החברות הלא ריקות non_empty_companies, במידה והיא לא קיימת שם אזי נחזיר failure מכיוון שזה אומר או שהיא לא קיימת במערכת או שאין בה עובדים. אם היא כן קיימת אז נשמור במצביע הנתון את מספר הזהות של העובד המתאים לדרישות התרגיל שנמצא ב-highest_earner במידע הנוף של צומת החברה. **סיבוכיות הזמן** במקרה זה היא O(logk) מכיוון שחיפשנו את החברה ונסיים.
אם CompanyID<0 אז נבדוק אם גודל אחד עצי העובדים של dummy הוא אפס (נניח אם dummy->employees_by_id הוא עץ ריק) אם כן אז נחזיר FAILURE מכיוון שזה אומר שאין עובדים במערכת. אחרת נשמור במצביע הנתון את מספר הזהות של העובד הדרוש השמור ב-highest_earner ב-dummy ונסיים. **סיבוכיות הזמן** במקרה זה היא O(1) מפני שלא היינו לחפש אף עובד או חברה.

GetAllEmployeesBySalary:

אם CompanyID>0: נחפש את החברה בשני עצי החברות שבמערכת ואם היא לא קיימת נחזיר FAILURE.
נשמור במצביע NumOfEmployees את הגודל של עצי העובדים של החברה (יש להם את אותו גודל) ואחר מכן נקצה מקום למערך של int ים בגודל הsize של עצי העובדים של החברה.
קודם נמיר את העץ employees_by_salary של החברה למערך (נקצה מקום למערך כגודל העץ ונשתמש בפונקציה של עץ הavl שעוברת על העץ עם סיור inorder ומכניסה כל value של הצומת הנוכחי לאינדקס המתאים במערך). נעבור על המערך שקיבלנו מהסוף להתחלה ונשים את ה-id המתאים במערך Employees* מהתחלה לסוף - כלומר, אנו הופכים את המערך של העץ כדי לקבל מיון מהשכר הגבוה לנמוך.
במקרה זה **סיבוכיות הזמן** תהיה O(logk + n(CompanyID)) מכיוון שאנו קודם מחפשים את החברה, ולאחר מכן עוברים על כל צמתי עץ ה-employees_by_salary שלה כדי להמיר אותו למערך וגם עוברים על המערך, וישנם n(CompanyID) צמתים בעץ זה.

אם CompanyID<0 נבצע את אותן הפעולות אך על עצי העובדים של dummy.
במקרה זה **סיבוכיות הזמן** תהיה O(n) מכיוון שאנו עוברים על כל צמתי העובדים שבמערכת כדי להעתיק אותם למערך ואז עוברים על כל תאי מערך זה שהוא בגודל n.

GetHighestEarnerInEachCompany:

נבדוק שה-size של העץ non_empty_companies הוא לא קטן מ-NumOfCompanies, במידה וכן נחזיר FAILURE, בגלל שזאת אומרת שמספר החברות שמעסיקות לפחות עובד אחד קטן מ-NumOfCompanies. לאחר מכן בעזרת סיור inorder על העץ הנ"ל נשמור את המידע הנוסף של צמתי החברות במערך זמני, אך לא נעבור על כל צמתי העץ, אלא רק על NumOfCompanies צמתים, כלומר נשמור מונה וכשנגיע בסיור למונה ששווה לNumOfCompanies נעצור.
נקצה מקום ל-Employees* שיהיה בגודל NumOfCompanies. נעבור על המערך של העץ ונוציא מכל תא את מספר הזהות של ה-highest_earner שלו ונשמור אותו בתא המתאים ב-Employees* .
נשחרר את המערך הזמני שגודלו הוא(O(NumOfCompanies ולכן לא הייתה חריגה בסיבוכיות המקום.
סיבוכיות הזמן: תחילה אנו מתחילים סיור inorder מהצומת הקטן ביותר של העץ ועוברים בו על NumOfCompanies צמתים, ולאחר מכן אנו עוברים על המער ומעתיקים ממנו את מספרי העובדים למערך Employees, ולכן הסיבוכיות תהיה O(logk+NumOfCompanies).

GetNumEmployeesMatching:

אם CompanyID>0: נחפש את החברה ב-non_empty_companies ואם לא מצאנו אותה נחזיר FAILURE, מכיוון שזה אומר או שהיא לא נמצאת במערכת או היא נמצאת ב-empty_companies ואין בה עובדים. במידה וכן מצאנו אותה, נלך לעץ ה-employees_by_id שלה ונדפיס את הצמתים שהמפתח שלהם, כלומר מספר הזהות, גדול או שווה מ-MinEmployeeID וקטן או שווה ל-MaxEmployeeID.
נעבור על מערך זה ונספור כמה תאים מאותחלים (לא שווים לnull) יש בו, ותוך כדי נשמור ונעדכן מונה לעובדים שבמידע הנוסף שלהם השכר והדרגה שלהם גדולים מ-MinSalary and MinGrade.
אם CompanyID<0 נבצע את אותו דבר רק עבור dummy->employees_by_id.
סיבוכיות הזמן- אנו עוברים על TotalNumOfEmployees עובדים ומחפשים את הצמתים המינימלי והמקסימלי של העובדים כדי לא להדפיס צמתים שלא בטווח.
עבור המקרה של CompanyID>0: אנו גם צריכים לחפש את החברה ולכן: סיבוכיות הזמן היא כנדרש.

Quit:

נשחרר את כל העצים והצמתים שלהם. קיימים 2 עצי חברות וכל עובד מופיע כצומת ב-4 עצים (שני עצי המערכת הנמצאים ב-dummy ושני עצי החברה שמעסיקה את עובד זה) ולכן מספר הצמתים במערכת הוא O(n+k) כאשר שחרור כל אחד מהם מתבצע בסיבוכיות זמן קבועה ולכן סיבוכיות הזמן של הפעולה היא O(n+k).