מבני נתונים - תרגיל רטוב 1

30 באפריל 2022

1 תיאור מבנה הנתונים:

1.1 עץ AVL

מבנה הנתונים שמימשנו למען פתרון התרגיל הוא עץ AVL גנרי המשתמש בפעולת השוואה בעזרת פונקציית bool customCompare(T, T) אשר מסופקת בבנאי - או, בברירת המחדל, אופרטור השוואה $>$ המוגדר עבור אותו אובייקט גנרי. העץ מורכב מצביע לשורש שהוא מטיפוס Node גנרי ובו מצביעים לNode ימני ושמאלי - ומייצג בפועל את צמתי העץ. בנוסף מכיל שדה המציין את גודל העץ.
העץ תומך בפעולות הסטנדרטיות: הוספה, חיפוש, מחיקה ואיחוד בין עצים. בנוסף לאלו, ישנן פוקנציות עזר:

- פוקנציות עזר לפונקציית merge:
  - buildEmptyTree(int): פונקציה פנימית של העץ שיוצרת עץ בינארי ריק מלא בגובה h
  - trim(AVLTree<T>, int): פוקנציה פנימית של העץ שמקבלת עץ שלם והופכת אותה לעץ כמעט שלם על ידי הורדת מספר עלים.
  - mergeArrays: ממזג שני מערכים בעזרת mergesort.
- פונקציות עזר לאיזון העץ:
  - balance: פונקציה הממשת את האלגוריתם שהוצג בהרצאות לאיזון העץ, משתמשת בפונקציות rotateRight/rotateLeft שגם הן מימוש של האלגוריתם שהוצג בהרצאה לסיבובים.

1.2 המערכת

המערכת תנהל שני אובייקטים - עובדים וחברות. כל חברה תכיל (אולי) עובדים, וכל עובד יהיה שייך לחברה. כלל האובייקטים במערכת ינוהלו, יווצרו, וימחקו על ידי המערכת. היחסים בין עובדים לחברות ישמרו בתור התייחסויות עם מצביעים בלבד. (ראו נספח א)

למען עמידה בסיבוכיות הדרושה, המערכת תחזיק את החברות והעובדים בעצי AVL ממוינים, וישמרו ב**עץ חברות ראשי** (בו יהיו כל המצביעים לחברות במערכת ברגע נתון) ו**עץ עובדים ראשי** (בו יהיו כל המצביעים לעובדים במערכת ברגע נתון) - אלו יהיו ממוינים לפי מזהה. בנוסף לשני עצים אלו, יהיו תתי עצים - עד כדי יחס סדר שונה - שיכילו מצביעים אל אותם העובדים או חברות בעצים הראשיים בסדר או תת קבוצה רלוונטית למשימה כזו או אחרת - כל זאת על מנת לעמוד בסבוכיות הדרושה.

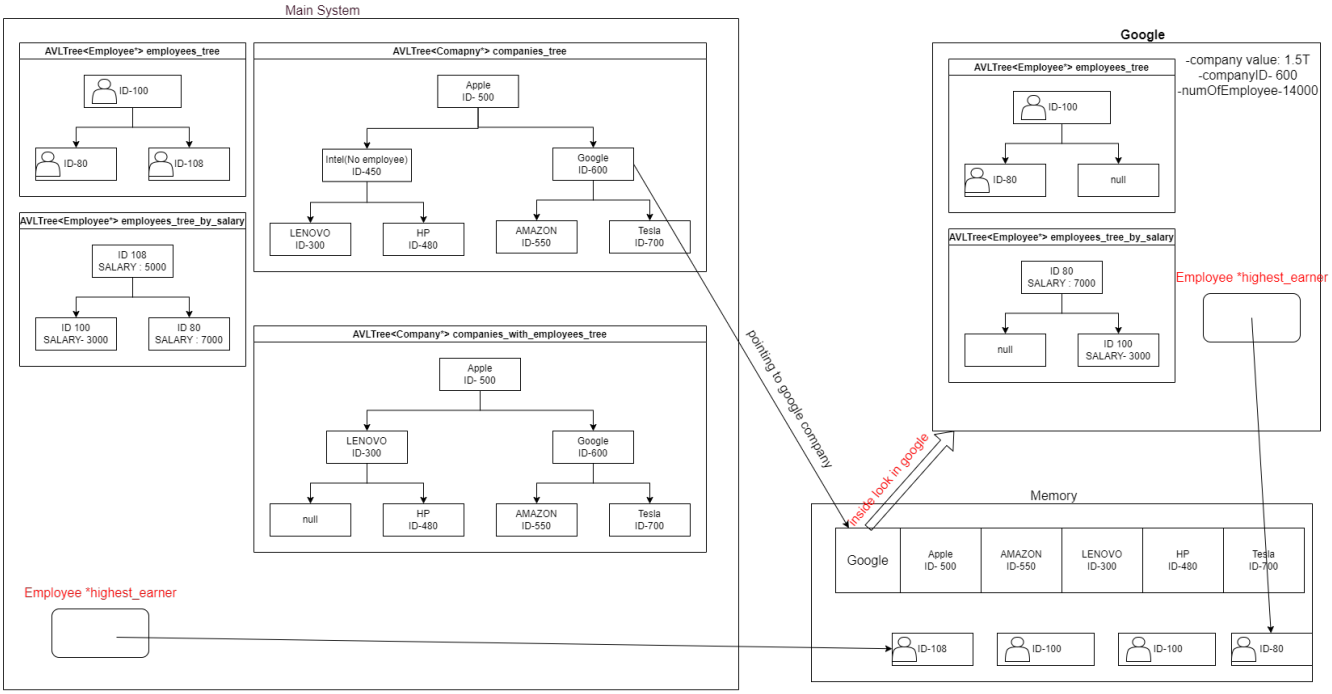
בפרט, המערכת תכיל בנוסף לעצים הראשיים עץ עובדים הממוינים לפי שכר-מזהה. עץ זה יכיל את כלל העובדים במערכת אך הם ימוינו בסדר עולה בשכר, ובמקרה והשכר שווה בסדר יורד במזהה. בנוסף, המערכת תכיל תת עץ חברות שיהיה ממוין לפי מזההים בו יהיו רק החברות בהן יש עובדים ברגע נתון.
בנוסף לעצים המתוארים לעיל יוחזק במערכת מצביע לעובד עם השכר הגבוהה ביותר במערכת כולה.
כל עובד בחברה יעבוד בחברה אחת בלבד.

1.2.1 חברה:

כל חברה תכיל שני עצי מצביעים לעובדים, המתייחסים לעובדים במערכת אליה החברה שייכת - האחד ממוין לפי מזהה, השני ממוין לפי שכר בדומה לעץ המקביל במערכת. החברה לא תמחק, תיצור או תשנה עובדים אלה רק תשמור מצביעים אליהם. בנוסף לאלו, תכיל שדה של מזהה, ערך החברה, מספר העובדים, ומצביע לעובד המרוויח הגדול מהעובדים בחברה.

1.2.2 עובד:

העובד יכיל שדה של מזהה, שכר, דרגה ומצביע לחברה בה הוא עובד.



1.3 ממשק

1.3.1 סיבוכיות זמן:

למען פשטות והקלת הקריאה עבור הקורא המסור, נציין שפעולות החיפוש,הסרה והוספה בעץ החברות הוא $\log k$ - כאשר k הוא מספר החברות בעץ, בנוסף כל חיפוש, הסרה,הוספה בעצים המכילים עובדים היא $\log n$ לכל היותר (מכיוון שמספר העובדים בעץ עובדים מסווים הוא בין 0 עובדים ולכל היותר n). אלגוריתם החיפוש עובד לפי שמורת החיפוש בעץ ולכן כפי שנלמד בהרצאה יקח $\log k/\log n$ (כתלות בעץ), אלגוריתם ההוספה עובד גם כפי הנלמד בהרצאה ולכן שומר על אותה סיבוכיות הזמן. במקרה ההסרה, במקרה הגרוע נבצע לאורך המסלול עד לשורש תיקונים לBF ולכן הסיבוכיות היא אותה הסיבוכיות זמן. בנוסף במהלך חלק מהפונקציות אנו מבצעים דפוס החוזר על עצמו : בדיקה האם האיבר בעץ, הכנסה / הוצאה , נציין שאם נבצע L פעמים חיפוש בעץ עדיין מתקיים שעבור J, L מספר טבעי : $J\log k + L\log n = O(\log k + \log n)$

1.3.2 סיבוכיות מקום:

במערכת כולה יוחזקו n עובדים וk חברות ־ בנוסף לאובייקטים עצמם השמורים בזכרון, יהיו העצים המוכלים במערכת, ובחברות, תלויים לינארית באותם משתנים. סכום כל הצמתים של עצי העובדים בתוך החברות הינו $2n$ מכיוון שזו חלוקה של העובדים (כל עובד יכול להיות בחברה אחת וישנם שני עצים בכל חברה), בנוסף ־ בתוך המערכת, העצים המכילים את החברות מחזיקים לכל היותר k צמתים והעצים של העובדים בדיוק n עובדים כל אחד. סך הכל, ברגע נתון יש $2n + 2n + 2k$ צמתים, בנוסף ל$n + k$ עובדים וחברות שהוקצו במערכת. כלומר ־ $O(n + k)$.

1. **init**:
הפעולה יוצרת ארבע עצים רקים : עץ של מצביעים לחברות, עץ שמכיל רק את החברות בעלות עובדים, עץ של מצביעים לעובדים הממויניים לפי ת`ז ועץ נוסף של מצביעים לעובדים הממוינים לפי המשכורות. בסהכ הסיבוכיות היא $O(1)$.

2. **AddCompany**:
הפעולה יוצרת חברה חדשה ריקה מעובדים, נבצע חיפוש בעץ החברות לראות האם החברה כבר קיימת - במידה וכן, נחזיר את ערך השגיאה המתאים. אם היא לא קיימת נבצע הכנסה של החברה בעזרת שמורת החיפוש של עץ החברות. בסהכ החיפוש וההכנסה $\log k + \log k = O(logK)$, כמו כן אם שאר ערכי הפרמטרים לא עומדים בתנאים אנו מחזירים את השגיאה המתאימה.

3. **AddEmployee**:
נבדוק את ערכי הפרמטרים, במידה ולא עומדים בתנאים נחזיר את השגיאה המתאימה. נבצע חיפוש למציאת החברה שעלינו להוסיף את העובד, אופן החיפוש הוא לפי שמורת החיפוש בעץ כפי שנלמד בהרצאה, במקרה הכי גרוע $\log k$, אם החברה אינה נמצאת - נחזיר את השגיאה המתאימה. ניצור את העובד, כעת נבדוק אם העובד קיים במערכת בסיבוכיות של $\log n$, אם העובד נמצא אנחנו מחזירים שגיאה שהעובד כבר קיים במערכת ונשחרר את העובד שיצרנו. אחרת, נצרף את העובד לעץ העובדים, לעץ העובדים שממוויין לפי המשכורות, ונוסיף אותו לעצים שבתוך החברה. אם לא היו עובדים בחברה נוסיף את החברה לעץ החברות בעלות עובדים. בסך הכל במקרה הגרוע $2\log k + 5\log n = O(\log k + \log n)$, בנוסף אנו בודקים אם העובד החדש הוא המרוויח הגדול מכולם, ודואגים לעדכן ב $O(1)$.

4. **RemoveEmployee**:
נבדוק את תקינות הפרמטרים, אם יש בעיה נחזיר את ערך ההחזרה המבוקש. נחפש את העובד לפי המזהה, ונשיג את החברה שבה הוא עובד ממנו ($O(1)$). נסיר את העובד מכלל העצים שבהם הוא נמצא, סך הכל $4\log n = O(\log n)$.

5. **RemoveCompany**:
נבדוק את תקינות הפרמטרים, אם לא נחזיר שגיאה מתאימה. במידה החברה לא קיימת נחזיר את השגיאה המתאימה לפי הדרישות. כעת נבצע בדיקה לראות מה מספר העובדים בחברה, מכיוון שקיים שדה המחזיק את מספיר העובדים בחברה פעולה זו היא $O(1)$. אם קיימים עובדים נחזיר את השגיאה המתאימה. אחרת, נסיר את החברה לפי האלגוריתם הנלמד בהרצאה שממומש בפונקציית ההסרה של העץ, סה`כ סיבוכיות ההסרה היא $\log(n)$.

6. **GetCompanyInfo**:
נבדוק את תקינות הפרמטרים, במידה ואין שגיאה נבצע חיפוש עבור החברה המבוקשת. במידה והיא לא קיימת נחזיר את השגיאה המתאימה, אחרת נשתמש בפונקציות הgetרים ־ פונקציות אלו פועלות בסיבוכיות $O(1)$ מכיוון שהערכים שמורים בשדות המתאימים בחברה. לאחר מכן, נשים את הערכים המתאים בvalue ובnumOfEmployees. סך הכל, במקרה הגרוע הסיבוכיות היא $O(\log k)$ בגלל החיפוש.

7. **GetEmployeeInfo**:
הפונקציה פועלת באופן דומה לפונקציה הקודמת ־ נבדוק את תקינות הפרמטרים, במידה ואין שגיאה נבצע חיפוש עבור העובד המבוקש בעץ העובדים הראשי. במידה והוא לא קיים נחזיר את השגיאה המתאימה, אחרת נשתמש בפונקציות הgetרים ־

פונקציות אלו פועלות בסיבוכיות $O(1)$ מכיוון שהערכים שמורים בשדות המתאימים בעובד. לאחר מכן, נשים את הערכים המתאים בכתובות המסופקות. סך הכל, במקרה הגרוע הסיבוכיות היא $O(\log n)$ בגלל החיפוש.

8. **IncreaseCompanyValue**:
נבדוק את תקינות הפרמטרים, במידה ואין שגיאה נבצע חיפוש עבור החברה המבוקשת בעץ החברות הראשי. אם נמצאה החברה, נשתמש בפונקציה Company::setValue(int value). סך הכל $O(\log k)$.

9. **PromoteEmployee**:
נבצע חיפוש בעץ העובדים ונמצא את העובד. נעדכן את השדות הרלוונטיים של העובד. בכדי לעדכן את מיקומו בעצי העובדים הממוינים לפי משכורת (בחברה ובמערכת) נוריד ונחזיר אותו ($4\log n$ במקרה הגרוע). החברה שבה עובד קיימת בשדה פנימי ולכן רק נצטרך למצוא את העובד ולבצע את העדכונים - סך הכל $O(\log n)$.

10. **HireEmployee**:
נמצא את העובד ואת החברה החדשה. נבדוק אם החברה החדשה שונה מהחברה הישנה, אם זו אותה חברה נחזיר שגיאה מתאימה. אחרת, נסיר את העובד מהחברה הישנה ונוסיף אותו לחברה החדשה. נעדכן את שדה החברה בעובד. נעדכן את מצב החברות בעץ החברות בעלות עובדים בהתאם למצבן אחרי העברת העובד. סה"כ, $O(\log n + \log k)$.

11. **AcquireCompany**:
נמצא את החברה הרוכשת והחברה הנרכשת. נבדוק שתנאי הרכישה מתקיימים לפי דרישות התרגיל, במידה והרכישה יכולה להתבצע נבצע מיזוג של החברות, כלומר מיזוג בין כלל העצים שבהם, עדכון הערך של החברה הרוכשת, ועדכון העובד המשתכר ביותר. לאחר מכן נסיר את החברה הישנה מהמערכת ונעדכן את עץ החברות בעלות העובדים בהתאם. סך הכל $O(\log k + n_{\text{Acquirer}} + n_{\text{Target}})$ בשל שימוש בפונקציית merge וחיפוש החברות המתאימות.

12. **GetHighestEarner**:
אם $\text{companyID} < 0$ נחזיר את העובד המתפנק ביותר מהשדה שמוחזק במערכת. אחרת, נמצא את החברה הנתונה ונחזיר מהשדה הפנימי שלה את המפונק. במידה והיא לא קיימת נחזיר שגיאה. במקרה בו חיפשנו את החברה, $O(\log k)$. אחרת, $O(1)$.

13. **GetAllEmployeesBySalary**:
נבחר את עץ העובדים הממוין לפי השכר המתאים עפ"י ה companyID, במידת הצורך על פי חיפוש בעץ החברות. נקבל מערך inOrder של העץ לפי השכר בשימוש בפונקציה של העץ, ממנו ניצור מערך של תעודות זהות ונחזיר את המערך הזה. סך הכל סיבוכיות $O(\log k + n_{company})$ כש$\text{companyId} > 0$, אחרת זהו מערך כל העובדים ולכן $O(n)$.

14. **GetHighestEarnerInEachCompany**:
נבדוק תחילה בעזרת עץ החברות בעלות עובדים אם יש מספיק חברות במערכת, אם ישנן - נקצה מערך int כגדול מספר החברות המבוקשות, לאחר מכן נקבל מערך inOrder מעץ החברות בעלות עובדים בגדול המתאים ונעבור עליו כדי לקבל את הId של כל עובד במערך שהקצנו. סך הכל, $O(NumOfCompanies + \log k)$ במקרה הגרוע בשימוש בפונקציית מערך inOrder על עץ החברות עם עובדים.

15. **GetNumEmployeesMatching**
נבצע בדיקה רקורסיבית על עץ העובדים המתאים - אותו נבחר על ידי מזהה החברה או אם קטן מ0 עץ העובדים הראשי. הרקורסיה תתבצע $inOrder$ ותתחיל מהעובד בעל מזהה בטווח הקטן ביותר. לכל עובד מעליו ביחס הסדר בעץ, תתבצע בדיקה של שאר התנאים. תנאי עצירה לרקורסיה הינו הגעה לסוף העץ או הגעה לעובד בעל מזהה גדול מהחסם העליון על המזהים - ותוך כדי הריצה יעודכנו שני מונים על התנאים שיוחזרו בסוף הריצה בעזרת המצביעים המסופקים. סך הכל, עד תחילת הרקורסיה לכל היותר $\log n$ שבה n הוא מספר העובדים בעץ העובדים המתאים, וריצת ה$inOrder$ עצמה תתבצע $TotalNumOfEmployees+1$ פעמים. סך הכל, אם $CompanyId < 0$ סיבוכיות $O(\log n+TotalNumOfEmployees)$, אחרת בגלל חיפוש החברה - $O(\log k + \log n_{company} + TotalNumOfEmployees)$.

16. **Quit**: תתבצע מחיקה של כלל העובדים במערכת לפי עץ העובדים הראשי, ומחיקה של כלל החברות במערכת לפי עץ החברות הראשי. כלל העצים מורכבים ממצביעים והינם תתי עצים של עצים אלו, לכן זו תהיה מחיקה של כל הזכרון שהוקצה דינאמית.