בס"ד

חלק יבש:

הגדרת מבני הנתונים שלנו:

1) System:

מבנה הנתונים המרכזי המכיל את כל המידע הנדרש לגבי העובדים ולגבי החברות במערכת, אשר מורכב מעצי AVL הבאים:

1. **Companies** – עץ AVL השומר את כל החברות במערכת. כל צומת מייצג חברה, כאשר המפתח של הצומת הוא המזהה של החברה והמידע שבצומת הוא השווי של החברה. העץ ממוין לפי המזהים של החברות.
2. **SettledCompanies** – עץ AVL השומר את כל החברות במערכת המכילות לפחות עובד אחד. כל צומת מייצג חברה, כאשר המפתח של הצומת הוא המזהה של החברה, והמידע שבצומת הוא אובייקט המייצג חברה (עליו נרחיב בהמשך). העץ ממוין לפי המזהים של החברות.
3. **SalarySortedEmp** – עץ AVL השומר את כל העובדים במערכת. כל צומת בעץ מייצג עובד, כאשר המפתח וגם המידע של הצומת הינם אובייקטים המייצגים עובד (על אובייקט זה נרחיב בהמשך). העץ ממוין לפי שכרם של העובדים, וכאשר נתקלים בשכר זהה, אז לפי המזהים של העובדים.
4. **IdSortedEmp** – עץ AVL השומר את כל העובדים במערכת. כל צומת בעץ מייצג עובד, כאשר המפתח של הצומת הינו המזהה של העובד, והמידע שבצומת הינו אובייקט מטיפוס עובד (עליו נרחיב בהמשך). העץ ממוין לפי המזהים של העובדים.

2) Company:

מבנה נתונים המכיל את פרטי החברה:

1. **Id** – מספר מזהה החברה.
2. **Value** – שווי החברה.
3. **salarySorted** – עץ AVL השומר את כלל העובדים בחברה – מסודרים לפי המשכורת שלהם בעדיפות ראשונה, ובעדיפות שניה על פי המזהה שלהם. המפתח והמידע של כל צומת הינם אובייקטים המייצגים עובד.
4. **IdSortedEmp** – עץ AVL השומר את כלל העובדים בחברה – מסודרים לפי המזהה שלהם. המפתח של כל צומת הינו המזהה של העובד, והמידע של כל צומת הינו אובייקט המייצג עובד.

3) Employee:

מבנה נתונים המכיל פרטים על עובד:

1. **Id**- מספר מזהה של העובד.
2. **Salary** – משכורת של העובד.
3. **Grade** – דרגה של העובד.
4. **CompanyID** – מצביע מסוג int אשר מצביע על מספר המזהה של החברה בה העובד עובד.

4) Tnode:

אובייקט (צומת) אשר נשתמש בו בעץ אשר מכיל בתוכו:

1. **Key** – מבציע מטיפוס KeyType שמייצג את המפתח של האובייקט.
2. **Data** – מצביע מטיפוס DataType שמייצג את המידע של האובייקט.
3. **Left** – מצביע לילד השמאלי.
4. **Right** – מצביע לילד הימני.
5. **Parent** – מצביע להורה של האובייקט.
6. **Height** – גובה של הצומת, מחושב על פי המרחק מהצומת אל העלה העמוק ביותר בתת העץ כאשר הצומת היא השורש של תת העץ (כפי שהגדרנו בהרצאה).

5) Tree: עץ AVL גנרי, אשר צמתיו הם אובייקטים Tnode המכיל:

1. **Root** – מצביע לצומת אשר מייצגת את שורש העץ.
2. **Counter** – מונה שתפקידו למנות כמה צמתים יש בעץ בכל זמן נתון.
3. **Max_node** – מצביע לצומת בעלת הkey הגדול ביותר בעץ בכל זמן נתון.

סימונים: n – מספר העובדים במערכת, k – מספר החברות במערכת.

בס"ד

עץ AVL ופונקציות עזר:

מימשנו עץ AVL גנרי כפי שנלמד בהרצאות ובתרגולים, ובנוסף מימשנו את הפונקציות הבאות:

1) mergeTree – פונקציה המקבלת עץ AVL, וממזגת בינו לבין העץ AVL אשר קראנו ממנו לפונקציה.
   1. ראשית נהפוך את העצים למערך ממוין באמצעות סיור InOrder (בסה"כ 2 מערכים של ה- KeyType של כל אחד מהעצים ועוד 2 מערכים עבור ה- DataType בסוף נקבל 2 מערכים עבור כל אחד מהעצים וסה"כ ארבעה מערכים).
   2. מיזוג המערכים באמצעות MergeSort למערך אחד ממוין.
   3. מחיקת כל הצמתים מהעץ ממנו נקראה הפונקציה.
   4. קריאה לפונקציה הרקורסיבית arrayToTree שמקבלת את המערכים הסופיים, אינדקסים של התחלה וסיום, ומצביע לסטטוס הפעולה, והופכת אותו לעץ באמצעות האלגוריתם הבא:
      1. הכנסת Array[middle] לשורש.
      2. קריאה לרקורסיבית לתת העץ השמאלי.
      3. קריאה רקורסיבית לתת העץ הימני.

סיבוכיות הזמן: הפיכת העצים למערך ממוין, מיזוג למערך ממוין, מחיקת כל הצמתים והמרה לעץ – O(d+m) כאשר d,m מייצגים את מספר הצמתים בכל אחד מהעצים.

נשים לב שבמקרה הגרוע יהיו כ-(n+m) קריאות במחסנית.

2) findMaxNode – פונקציה רקורסיבית שמקבלת אובייקט (node) ובודקת אם הערך של המפתח של האובייקט גדול יותר מהערך של max_node, אם כן ,מעדכנת את max_node ומבצעת קריאה רקורסיבית עם תת העץ הימני, עד שנגיע לסוף העץ.

   סיבוכיות הזמן: O(logm) כאשר m הוא מספר הצמתים בעץ (בעצם גובה עץ AVL).

3) **countCondNodes** – פונקציה רקורסיבית שמקבלת צומת מתוך העץ שהוא בעצם השורש של תת עץ כלשהו, ומקבלת טווח המינימלי וטווח המקסימלי מטיפוס KeyType ומחזירה את מספר האובייקטים שהמפתחות שלהם נמצאים בטווח.

   סיבוכיות הזמן: O(m) כאשר m הוא מספר הצמתים אשר המפתח שלהם נמצא בטווח.

4) specialcountCondNodes – פונקציה רקורסיבית שמקבלת מערך של מצביעים לטיפוס מסוג DataType, ובנוסף מקבלת אובייקטים מסוג KeyType שמייצגים טווח מינימלי ומקסימלי. הפונקציה מעדכנת את מערך המצביעים במצביעים ל-DataType של צמתים שהמפתחות שלהם נמצאים בטווח הנדרש. פונקציה זאת תעזור לנו לממש מתודות בתרגיל.

   סיבוכיות הזמן: O(m) כאשר m הוא מספר הצמתים אשר המפתח שלהם נמצא בטווח.

5) deleteAllNodes – פונקציה רקורסיבית אשר מוחקת את כל הצמתים בעץ.

   סיבוכיות הזמן: O(m) כאשר m הוא מספר הצמתים בעץ.

   נשים לב שבמקרה הגרוע יהיו כ-(log m) קריאות במחסנית.

6) arrayToTree – פונקציה רקורסיבית שמקבלת מערך מצביעים לאובייקטים מסוג KeyType ומערך מצביעים לאובייקטים מסוג DataType שמהם היא בונה עץ AVL שממוין לפי הKeyType. הפונקציה מחזירה מצביע לשורש של העץ שנוצר מהמערך.

   סיבוכיות הזמן: O(m) כאשר m הוא מספר האיברים במערך.

7) inOrder – פונקציה רקורסיבית המקבלת שני מערכים ומכניסה את המפתחות והמידע של הצמתים בעץ לכל אחד מהמערכים בהתאמה לפי סיור InOrder אשר למדנו בביתה.

   סיבוכיות הזמן: O(m) כאשר m הוא מספר הצמתים בעץ.

נשים לב שמבחינת סיבוכיות מקום אנו לא חורגים מסיבוכיות המקום הנדרשת בתרגיל.

בס"ד

פירוט הפונקציות בSystem:

1) void* Init() -
אתחול של המערכת System – כאשר בתוכה 4 עצי AVL ריקים שמצביעים ל- NULL.
סיבוכיות זמן : O(1).
סיבוכיות מקום נוכחית: O(1), המערכת ריקה.

2) StatusType AddCompany(void *DS, int CompanyID, int Value) -
פונקציה שמכניסה חברה חדשה למערכת.
1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה שלילי או שווי שלילי נזרקות שגיאות כנדרש.
2. בדיקה האם החברה כבר קיימת במערכת או לא, אם כן תיזרק שגיאה בהתאם - $O(logk)$ (חיפוש בעץ AVL).
3. יצירת אובייקט מטיפוס Company עם המזהה שהתקבל ויצירת העצים והשדות של האובייקט - $O(1)$.
4. הכנסת החברה לעץ Companies במערכת - $O(\log k)$ (הכנסת איבר לעץ AVL).

הפונקציה תחזיר SUCCESS במקרה וכל התהליך הסתיימו בצורה תקינה , אחרת תיזרק שגיאת זיכרון.

סיבוכיות זמן : O(log k).
סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

3) StatusType AddEmployee(void *DS, int EmployeeID, int CompanyID, int Salary, int Grade) -
פונקציה שמכניסה עובד חדש לחברה.
1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד / מזהה חברה / משכורת שליליים, נזרקות שגיאות.
2. בדיקה אם החברה לא קיימת במערכת, אם כן נזרקת שגיאה – O(logk).
3. בדיקה אם קיימים עובדים במערכת, אם לא נבצע את הפעולות הבאות:
   1. נקצה עץ עובדים למערכת – O(1). תיזרק שגיאה במקרה וההקצאה נכשלה.
   2. נקצה עץ משכורות למערכת – O(1). תיזרק שגיאה במקרה וההקצאה נכשלה.
4. בדיקה אם העובד נמצא במערכת, מתבצע על ידי מעבר בעץ IdSortedEmp , אם כן נזרקת שגיאה – O(logn).
5. חיפוש בעץ SettledCompanies ויצירת מצביע לData – O(logk), אם החברה לא קיימת , אז מדובר על עובד ראשון, אז נבצע את הפעולות הבאות:
   1. ניצור מצביע לחברה ע"י חיפוש בעץ companies - O(logk).
   2. ניצור אובייקט מטיפוס Company -O(1), נזרוק שגיאת זיכרון במקרה ויש תקלה.
   3. נקצה עץ עובדים לחברה ועץ משכורות לחברה - $O(1)$.
   4. נכניס את החברה לעץ בעזרת insert -O(logk), ולאחר מכן נעדכן את העצים של החברה להצביע לעצים שיצרנו. במידה ולא נצליח נחזיר את הקוד המתאים.
6. ניצור אובייקט Employee – O(1).
7. נכניס את ה-Employee ל-2 העצי החברה - $O(logm)$, כאשר m הוא מספר העובדים בחברה. מכיוון ש $m \leq n$ אז $O(logm) \leq O(logn)$.
8. נכניס ל-2 עצי העובדים במערכת - $O(\log n)$.

הפונקציה תחזיר SUCCESS במקרה וכל התהליך הסתיימו בצורה תקינה , אחרת תיזרק שגיאת זיכרון.

סיבוכיות זמן: סה"כ נקבל שסיבוכיות הזמן היא: O(log n+ log k).
סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

4) StatusType RemoveEmployee(void *DS, int EmployeeID) –
הפונקציה מוחקת את נתוני העובד מהמערכת.
1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד אי חיובי, נזרקות שגיאות.
2. בדיקה אם קיימות חברות במערכת - $O(1)$, בדיקה אם העובד קיים במאגר - $O(logn)$, במקרה ולא נזרקת שגיאה.
3. ניצור אובייקט מטיפוס Employee שיחזיק בתוכו את המידע של העובד – $O(logn)$ (שליפת מידע העובד מעץ העובדים).
4. נמחק מעץ של עובדי המערכת - $O(logn)$.
5. נמחק מעץ המשכורות של המערכת - $O(logn)$.
6. נמצא את החברה בה עובד העובד בעץ settledCompanies - $O(logn)$ (במקרה הגרוע וזאת מכיוון שישנם סה"כ n עובדים במערכת ולכן סה"כ n חברות במקרה הגרוע בעץ הזה מכיוון שעץ זה מכיל את החברות שיש בהן לפחות עובד אחד).

7. נסיר את העובד משני עצי החברה, נסמן m כמספר העובדים בחברה - $O(logm)$.
8. נבדוק אם החברה שבה עבד העובד ריקה מעובדים כעת - $O(logk)$, במידה ולא, נחזיר SUCCESS, במידה וכן נסיר את החברה מעץ החברות הפעילות (SettledCompanies) - $O(logk)$.
אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> לפי הנימוקים אשר הצגנו – O(log n).
<u>סיבוכיות מקום נוכחית:</u> לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

5) StatusType RemoveCompany(void *DS, int CompanyID) –

החברה פושטת רגל ויש למחוק אותה מהמערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה שלילי, נזרקות שגיאות.
2. נבדוק אם קיימות חברות במערכת - $O(1)$, במידה ולא , תיזרק שגיאה.
3. נבצע חיפוש בעץ Companies על מנת לבדוק אם החברה קיימת - $O(logk)$.
במידה ולא, תיזרק שגיאה.
4. נבצע חיפוש בעץ SettledCompanies על מנת לבדוק אם לחברה יש עובדים - $O(logk)$.
במידה ונמצא את החברה בעץ הזה סימן שיש עובדים בחברה, תיזרק שגיאה.
5. נבצע הסרה של החברה מהעץ companies - $O(logk)$.

<u>סיבוכיות זמן :</u> $\boldsymbol{O(logk)}$.
<u>סיבוכיות מקום נוכחית:</u> לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

6) StatusType GetCompanyInfo(void *DS, int CompanyID, int *Value, int *NumEmployees) –
מחזירה את שווי החברה ומספר העובדים בה.
1. אם נשלח מצביעים לא תקינים של המערכת/שווי חברה/כמות עובדים או מזהה חברה שלילי, נזרקות שגיאות.
2. נבדוק אם קיימות במערכת חברות - $O(1)$, אם לא, תיזרק שגיאה.
3. נבדוק אם החברה קיימת ע"י חיפוש בעץ companies - $O(logk)$, אחרת תיזרק שגיאה.
4. אם מצאנו נעדכן את המצביע לשווי החברה להיות הdata של החברה בעץ companies - $O(logk)$.
5. נבצע חיפוש בעץ SettledCompanies - $O(logk)$.
6. אם נמצא את החברה בחיפוש נעדכן את המצביע של NumEmployees לכמות העובדים בחברה ואם לא נאפס אותו (כי אין עובדים בחברה) $O(1)$.

אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> $\boldsymbol{O(logk)}$.
<u>סיבוכיות מקום נוכחית:</u> לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

7) StatusType GetEmployeeInfo(void *DS, int EmployeeID, int *EmployerID, int *Salary, int *Grade) -
פונקציה המחזירה את מעסיקו, שכרו והדרגה הנוכחית של העובד.
1. אם נשלח מצביעים לא תקינים של המערכת/שכר/דרגה או מזהה עובד אי חיובי, נזרקות שגיאות.
2. נבדוק ונשמור מצביע, אם קיים עובד במערכת על ידי חיפוש בעץ idSortedEmp - $O(logn)$, במקרה ולא תיזרק שגיאה.
3. השמה עבור כל המצביעים - $O(1)$.

אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן:</u> $\boldsymbol{O(logn)}$.
<u>סיבוכיות מקום נוכחית:</u> לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

8) StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease) –
פונקציה תשנה את ערך החברה.
1. אם נשלח מצביע לא תקין של המערכת או מזהה חברה/תוספת לשווי אי חיוביים, נזרקות שגיאות.
2. נחפש בעץ companies את החברה וניצור מצביע אל האובייקט, במקרה ולא נמצא את החברה תיזרק שגיאה - $O(logk)$.
3. נסתכל על הdata של החברה, כאשר באותו העץ הdata הוא שווי החברה, ונעדכן אותו בערך החדש - $O(1)$.

4. לאחר מכן נבדוק אם החברה קיימת בעץ של settledCompanies, במידה ולא, נסיים ונשלח SUCCESS, ואחרת, נעדכן את שווי החברה - $O(logk)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(logk)$.
סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

9) StatusType PromoteEmployee(void *DS, int EmployeeID, int SaloryIncrease, int BumpGrade) -
העובד מקבל קידום ועל הפונקציה לעדכן זאת במערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד/תוספת לשכר אי חיוביים, נזרקות שגיאות.
2. נבדוק אם העובד נמצא במערכת - $O(logn)$, אם לא תיזרק שגיאה ואם כן נשמור את המידע שלו.
3. לאחר מכן נעדכן את השכר החדש של העובד ואת דרגתו אם BumpGrade חיובי, אחרת, הדרגה לא תשתנה – O(1).
4. נחפש את העובד בעץ המשכורות של המערכת - $O(logn)$.
5. נמחק את האובייקט הקיים - $O(logn)$.
6. נקצה אובייקט מעודכן ונכניס אותו לעץ המשכורות של המערכת - $O(logn)$.
7. ניצור מצביע לחברה שבה עובד העובד על ידי חיפוש בsettledCompanies - $O(logk)$.
8. עבור עץ המשכורות של החברה נמחק את האובייקט הישן - $O(logm)$ כאשר m מייצג את מספר העובדים בחברה.
9. נקצה אובייקט מעודכן ונכניס אותו לעץ המשכורות של החברה- $O(logm)$ כאשר m מייצג את מספר העובדים בחברה.
10. נעדכן את פרטי העובד בעץ בעובדים של החברה הממוין לפי מספרים מזהים - $O(logm)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : $O(logn)$.
סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

10) StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID)
חברה מגייסת עובד מחברה אחרת, הפונקציה מבצעת את השינויים המתאימים במערכת.

1. אם נשלח מצביע לא תקין של המערכת או מזהה עובד/חברה אי חיוביים, נזרקות שגיאות.
2. נבצע 2 בדיקות: אם לא קיימים חברות במערכת ואם לא קיימים עובדים במערכת יזרקו שגיאות -$O(1)$.
3. ניצור מצביע לEmployee על ידי חיפושו בעץ העובדים של המערכת - $O(logn)$.
4. ניצור מצביע לחברה החדשה אליה עובר העובד O(logk) (חיפוש בעץ AVL) במקרה והמצביעים יצביעו על nullptr תיזרק שגיאה.
5. מקרה נוסף שתיזרק שגיאה: החברה אליה צריך לעבור העובד היא החברה שבה הוא עובד, הבדיקה O(1).
6. נשמור את הפרטים הרלוונטים - O(1).
7. מחיקת העובד מהמערכת – $O(logn)$
8. יצירת עובד חדש עם פרטים מעודכנים - $O(logn + logk)$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן: O(logn+logk).
סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

11) StatusType AcquireCompany(void *DS, int AcquirerID, int TargetID, double Factor) -
חברה אחת רוכשת את החברה השניה, הפונקציה תבצע את השינויים המתאימים ותעדכן את המערכת.

1. במקרה ונשלח מצביע לא תקין של המערכת / מספרים מזהים של החברות אי חיוביים/factor קטן מ-1, נזרקות שגיאות.
2. בדיקה אם קיימות חברות במערכת - $O(1)$.
3. ניצור מצביעים ונחפש את כל אחת מהחברות בעץ companies - $O(logk)$ לכל חיפוש, אם לא נמצא את אחת החברות תיזרק שגיאה.
4. נבצע בדיקה ששווי החברה הרוכשת גדול פי 10 לפחות משווי החברה הנרכשת, אם המצב אינו כך, תיזרק שגיאה - $O(1)$.
5. נעדכן את השווי החברה הרוכשת - $O(1)$.
6. באותו אופן, ניצור מצביעים לחברות ע"י הסריקות בעץ settledCompanies - $O(logk)$.
7. כעת ישנם כמה מצבים אפשרים:
   1. לחברה הנרכשת אין עובדים, במקרה זה נעדכן את שווי החברה הרוכשת בלבד - O(1) .
   2. לחברה הרוכשת אין עובדים, במקרה זה, נקצה אובייקט מטיפוס Company, ובנוסף נקצה עץ עובדי חברה ועץ משכורות בחברה - $O(1)$, ונשתמש במתודה mergeCompanies $O(n_{target})$ ונכניס את החברה לעץ החברות הפעילות.
   3. במקרה ולשני החברות יש עובדים, נבצע mergeCompanies שמשתמשת בmergeTree בעלת סיבוכיות זמן

$O(n_{target} + n_{acquirer})$.

8. נמחק את החברה שנרכשה בעץ settledCompanies במקרה והיו לה עובדים - $O(n_{target})$.
9. נמחק את החברה שנרכשה מעץ companies - $O(n_{target})$.
10. נשתמש בפונקציה עזר adjustCompId שהיא פונקציה רקורסיבית העוברת על כל העץ ומעדכן את הפוינטר של העובדים לחברה בה הם עובדים להצביע למזהה של החברה החדשה בה הם עובדים - $O(n_{target} + n_{acquirer})$.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$\boldsymbol{O(logk + n_{target} + n_{acquirer})}$$

סיבוכיות מקום: נשים לב שבכל רגע נתון בפונקציה זו לא נחרוג מסיבוכיות המקום הנדרשת, וזו על פי הגדרת פונקציה זו ועפ"י הגדרת mergeTree - הגדרת המערכים והעץ לא תחרוג מהסיבוכיות המקום הנדרש.

12) StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID) –

1. במקרה ונשלח מצביע לא תקין של המערכת / מצביע למספר מזהה של העובד או מספר מזהה של החברה אי חיובי נזרקות שגיאות. בנוסף אם לא קיימות חברות או עובדים במערכת אז יזרקו שגיאות - $O(1)$.
2. אם המזהה חברה חיובי, ניצור מצביע ונחפש את החברה בעץ companies - $\boldsymbol{O(logk)}$, במקרה ואינה שם תיזרק שגיאה.
   1. באותו אופן, ניצור מצביע ונחפש את החברה בעץ settledCompanies - $\boldsymbol{O(logk)}$, אם נמצא אותה בעץ המשמעות היא שיש לחברה עובדים, במקרה ולא נמצא תיזרק שגיאה.
   2. נשתמש בפוינטר שיש לכל עץ אשר מצביע לאובייקט המקסימלי בעץ (לפי איך שהעץ ממוין) – O(1) .
3. אם מזהה החברה שלילי:
   1. ניגש לעץ משכורות של כלל העובדים במערכת, ונשתמש בפוינטר על מנת לשלוף את העובד בעל המשכורת הגבוהה ביותר – O(1), אם עץ המשכורות של המערכת ריק, תיזרק שגיאה.

אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן :

$$O(logk) \ , if\ CompanyID > 0$$
$$O(1)\ , if\ companyID < 0$$

סיבוכיות מקום נוכחית: לא נחרוג מסיבוכיות המקום הנדרש בתוכנית.

13) StatusType GetAllEmployeesBySalary(void *DS, int CompanyID, int **Employees, int *NumOfEmployees) –

1. אם מזהה חברה חיובי, ניצור מצביע ונחפש את החברה בעץ companies - $\boldsymbol{O(logk)}$, במקרה ואינה שם תיזרק שגיאה.
   1. באותו אופן, ניצור מצביע ונחפש את החברה בעץ settledCompanies - $\boldsymbol{O(logk)}$, אם נמצא אותה בעץ המשמעות היא שיש לחברה עובדים, במקרה ולא נמצא תיזרק שגיאה.
   2. נקצה מערך בגודל מספר הצמתים בעץ המשכורות של החברה – O(1) .
   3. נבצע InOrder (הפונקציה שהגדרנו בעץ AVL) כאשר הפלט יכנס בעצם למערך – $O(\boldsymbol{n_{CompanyID}})$.
2. במקרה ומזהה החברה שלילי, נבצע תהליך דומה רק עבור עץ משכורות של כלל העובדים במערכת, ההבדל הוא שגודל המערך יהיה n, וסיבוכיות הזמן של inOrder תהיה O(n) .
3. כעת , נקצה מערך בגודל המערך ששלחנו לפונקציית inOrder.
4. נבצע היפוך למערך, מכיוון שעלינו להחזירו בצורת Reverse inOrder - $O(n)/O(\boldsymbol{n_{CompanyID}})$ (תלוי מקרה)
5. Employees יצביע למערך הסופי - O(1).

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

סיבוכיות זמן : במקרה הגרוע ביותר עבור כל אחד מהמקרים הבאים:

$$O(\log k + \boldsymbol{n_{CompanyID}})\ , if\ CompanyID > 0$$
$$O(n)\ , if\ companyID < 0$$

סיבוכיות מקום: במקרה הגרוע ביותר קיימים 2 מערכים כאשר כל אחד מהם בגודל n ובנוסף העצים של המערכת, ולא נחרוג מסיבוכיות המקום הנדרשת בתוכנית.

14) StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees) -

1. נבדוק מה גודל העץ settledCompanies, ואם במקרה הגודל הוא אפס, תיזרק שגיאה O(1).
2. נקצה מערך integers בגודל NumOfCompanies שיכיל את מזהי החברות, במקרה ונכשלה פעולת ההקצאה תיזרק שגיאה - $O(1)$.
3. נקצה מערך מצביעים לאובייקטים מטיפוס Company בגודל NumOfCompanies שיכיל את החברות עצמן, במקרה ונכשלה פעולת ההקצאה תיזרק שגיאה - $O(1)$.
4. נבצע InOrder לעץ settledCompanies, InOrder בנוי לרוץ רקורסיבית כגודל המערך ששלחו אליו, במקרה זה NumOfCompanies פעמים ולכן נקבל את המזהים שאנו רוצים לקבל ממוינים כנדרש - $O(NumOfCompanies)$.
5. נשחרר את מערך הintegers.
6. נקצה מחדש את מערך integers בגודל NumOfCompanies, אליו נכניס את הmax_node של עץ המשכורות של כל חברה. $O(NumOfCompanies)$.
7. נשחרר את מערך ה company – $O(NumOfCompanies)$
8. Employees יצביע אל המערך integers.

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> במקרה הגרוע ביותר עבור כל אחד מהמקרים שהצגנו לא נחרוג מ:

$$O(logk + NumOfCompanies)$$

<u>סיבוכיות מקום:</u> במקרה הגרוע ביותר קיימים 2 מערכים כאשר כל אחד מהם בגודל $NumOfCompanies$ ויתקיים שלא נחרוג מסיבוכיות המקום הנדרשת בתרגיל.

StatusType GetNumEmployeesMatching(void *DS, int CompanyID, int MinEmployeeID, int MaxEmployeeId, int MinSalary, int MinGrade, int *TotalNumOfEmployees, int *NumOfEmployees) – 15)

1. בדיקה אם אין חברות / עובדים במערכת, אם אין יזרקו שגיאות - $O(1)$.
2. יהיה לנו מונה של מספר עובדים שיאותחל ל-0, והוא יקרא numEmployees - $O(1)$.
3. אם ComanyID>0 אז נבצע את הפעולות הבאות:
   1. ניצור מצביע ונחפש את החברה בעץ companies של המערכת, אם לא נמצא את החברה תיזרק שגיאה $O(logk)$
   2. ניצור מצביע ונחפש את החברה בעץ settledCompanies של המערכת, אם לא נמצא את החברה תיזרק שגיאה $O(logk)$.
   3. כעת נבדוק כמה מתוך העובדים בחברה נמצאים בטווח של מזהי העובדים הרלוונטיים, פעולה זאת מתבצעת ע"י המתודה countCondNodesAss המפורטת בפונקציות עזר נוספות שמימשנו עבור מחלקת העץ, המתודה מחזירה את מספר העובדים שעומדים בתנאים – TotalNumOfEmployees - $O(TotalNumOfEmployees)$.
   4. אם מספר העובדים שווה ל-0 , אז $NumOfEmployees$ יצביע ל-0, המתודה תסתיים ותחזיר SUCCESS.
   אחרת, נקצה מערך מצביעים בגודל TotalNumOfEmployees,של מצביעים מטיפוס Employee. ונשתמש במתודה specialCondNodeAss שתמלא את מערך המצביעים שעומדים בתנאי הטווח - $O(TotalNumOfEmployees)$.
4. אם CompanyID<0 אז נבצע את הפעולות הבאות:
   1. אל מצביע TotalNumOfEmployees נבצע השמה בעזרת המתודה countCondNodesAss שתתבצע על עץ העובדים של כלל המערכת הממוין על פי EmpoleeID $O(TotalNumOfEmployees)$.
   2. אם TotalNumOfEmployees שווה ל-0 , נבצע השמה למצביע NumOfEmployees שיהיה שווה לאפס ואז תסתיים המתודה ויחזור SUCCESS.
   אחרת, נקצה מערך מצביעים בגודל TotalNumOfEmployees אשר מצביעים אל טיפוסי Employee. ונשתמש במתודה specialCondNodeAss שתמלא את מערך המצביעים שלEmployees שעומדים בתנאי הטווח $O(TotalNumOfEmployees)$.
5. כעת עבור 2 המקרים נבצע את הפעולות הבאות:
   1. נרוץ על מערך הEmployees ונעדכן את המונה numEmployees במקרה והEmployee עומד בדרישה של שכר וגם של הדרגה ← numEmployee++ -$O(TotalNumOfEmployees)$
6. NumOfEmployees=numEmployess*, ונשחרר את מערך המצביעים - $O(1)$.

חשוב להדגיש כי לאורך כל המתודה, בעת הקצאת מערכים/ אובייקטים , אם והקצאות נכשלו תיזרק שגיאה, אם כל התהליך צלח נחזיר SUCCESS.

<u>סיבוכיות זמן :</u> במקרה הגרוע ביותר עבור כל אחד מהמקרים הבאים:

$$O(logk + logn_{CompanyID} + TotalNumOfEmployees)\ ,if\ CompanyID > 0$$

חשוב לציין כי לפי ההסבר של הפונקציות *CountNodes* ו- *specialCountNodes* במקרה שבו בטווח מזהי העובדים נמצאים כל העובדים אזי $TotalNumOfEmployees = n_{CompanyID}$ ולכן מתקיים השיוויון למעלה.

$$\boldsymbol{O(\text{TotalNumOfEmployees})} \le O(logn + TotalNumOfEmployees)\quad ,if\ companyID < 0$$

<u>סיבוכיות מקום:</u> במקרה הגרוע ביותר קיימת הקצאה של מערך מצביעים בגודל TotalNumOfEmployees כאשר במקרה הגרוע ביותר אלה כלל העובדים במערכת $O(n)$, מתבצעות פונקציות רקורסיביות *CountNodes* ו- *specialCountNodes* , ולכן:

$$O(\text{TotalNumOfEmployees}) + O(\log(\text{TotalNumOfEmployees})) = \boldsymbol{O(log(\text{TotalNumOfEmployees}) + \text{TotalNumOfEmployees})}$$

16) void Quit(void **DS) –
משחררת את המבנה ואת כל הזיכרון שהוקצה , כל מחיקה של עץ מתבצעת ע"י המתודה deleteAllNodes ואז קריאה לדיסטרקטור,
סיבוכיות הזמן של deleteAllNodes היא $O(t)$, כאשר t הוא מספר הצמתים בעץ.

1. מחיקת עץ companies $O(k+n)$.
2. מחיקת עץ settledCompanies $O(k+n)$.
3. מחיקת עץ salarySortedEmp - $O(n)$.
4. מחיקת עץ idSortedEmp - $O(n)$.
5. שחרור המערכת - $O(1)$.
6. *DS=nullptr

<u>סיבוכיות זמן :</u> $O(n+k)$.
<u>סיבוכיות מקום:</u> deleteAllNodes היא פונקציה רקורסיבית וקוראים לה $O(logn + logk)$.

17) adjustCompanyID – פונקציית עזר אשר מימשנו שמטרתה היא לעבור על עץ של עובדים ולשנות את מזהה החברה בה הם עובדים.
<u>סיבוכיות זמן:</u> O(n) כאשר n הינו מספר הצמתים בעץ.
<u>סיבוכיות מקום:</u> עומדת בדרישות התרגיל.

<u>סיבוכיות מקום של התוכנית:</u>

הפונקציה עם סיבוכיות מקום הגרועה ביותר מכלל הפונקציות היא בעלת סיבוכיות מקום:
$\boldsymbol{O(log(\text{TotalNumOfEmployees}) + \text{TotalNumOfEmployees})}$
באשר במקרה הגרוע ביותר כל העובדי המערכת עובדים באותה חברה, ***TotalNumOfEmployees=n*** אז:
$\boldsymbol{O(logn + \text{n}) \leq O(2n) = O(n)}$.
סה"כ בכל רגע נתון המערכת מחזיקה: ***n*** עובדים ו-***k*** חברות, 4 עצים אשר מחזיקים:
$\boldsymbol{k + n + k + n + n = 3n + 2k = O(n+k)}$ אובייקטים.
ולכן המבנה עומד בדרישות סיבוכיות המקום.
חשוב לציין שהפונקציות הרקורסיביות אינם חורגות מסיבוכיות המקום של התוכנית ומחסנית הקריאות תעמוד בדרישות לאורך כל התוכנית.