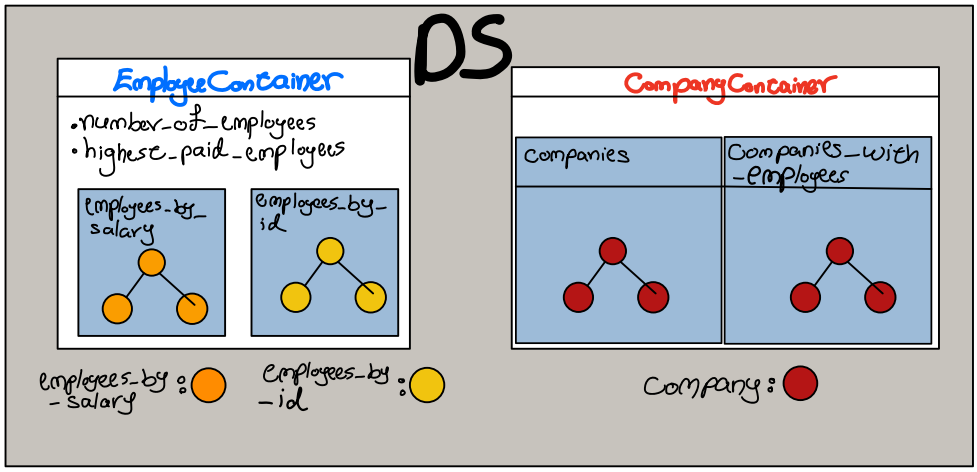
חלק יבש

אורי זהר - 057069502
לוי ראוכברגר - 218255602

1 תיאור מבנה הנתונים הראשי :



- מבנה הנתונים מכיל 2 מחלקות מרכזיות :
1) CompanyContainer: מבנה נתונים האחראי על ניהול המידע עבור החברות והעובדים בניהם . מבנה זה מכיל 2 עצי AVL אחד עבור כלל החברות והשני עבור חברות שמכילות עובדים (חברות לא ריקות) .
2) EmployeeContainer: מבנה נתונים שדואג לניהול המידע וטיפול בכלל העובדים
- מבנה זה מכיל 2 עצי AVL האחד דואג למיון של כל העובדים בהתאם לתעודת הזהות שלהם והשני לפי השוואת שכר (בצורה רגילה של int) ובמקרה בו השכר שווה אז עם השוואה הפוכה .
- באמצעות הבחירה במבנה הנתונים הנל נוכל לבצע את הפעולות הנדרשות גם על העובדים באופן כללי וגם בצורה פרטנית עבור החברות והעובדים השייכם להם .

2 פירוט המחלקות וטיפוסי הנתונים בהם השתמשנו:

2.1 *EmployeesContainer*

- מבנה הנתונים מכיל את השדות:
1) $EmployeesByID$: עץ AVL המכיל אובייקט של עובד לפי תעודת זהות .
2) $EmployeeBySalary$: עץ AVL המכיל אובייקט של עובד לפי שכר .
3) $highest\_paid\_employee$: מצביע לעובד עם השכר הכי גבוהה בעץ. העובדים הממיין לפי שכר .
4) $number\_of\_employees$: שדה השומר את מספר העובדים במבנה.
- מבנה נתונים זה מאפשר הוצאה והכנסה של עובדים ותוך כדי זה לתחזוק של השדה

של העובד הרווחי ביותר ומספר הכולל של העובדים . בנוסף נעזר בפעולות נוספות של עצי ה־AVL שיופרטו בהמשך .

2.1.1 מבנה נתונים $EmployeeBySalary/EmployeeById$:

־ מכיל את השדות :
1) id: תעודת הזהות של העובד .
2) $salary$: השכר של העובד .
3) $grade$: הדרגה של העובד .
4) $compny\_id$: מספר החברה בה העובד עובד .
5) $points\_main$: מצביע לאובייקט של העובד בעץ החברות הראשי .
־ עבורם יממש פונקציות של טיפול בעובד בגישה ישירה כמו העלאת שכר עידכון החברה בה הוא עובד וכו .
־ ההבדל בין שני סוגי העובדים הוא המימוש של אופרטורי ההשוואה כפי שתואר לעיל .
־ המטרה של $points\_main$ היא לאפשר גישה בין עובד הנמצא בחברה מסויימת לעובד הנמצא בעץ העובדים הכללי ללא חיפוש בכלל העובדים.

2.2 $CompanyContainer$

־ מבנה הנתונים מכיל 2 שדות :
1) $companies$: עץ AVL המכיל את כלל החברות במערכת .
1) $companies\_with\_employee$: עץ AVL המכיל חברות שיש בהם עובדים .
־ הסיבה להפרדה בין כלל החברות לחברות המכילות עובדים בלבד הוא שכאשר אנו רוצים לבצע
פעולה על עובד ונצטרך לחפש את החברה שלו נוכל לעשות זאת בלי
לעבור על כלל החברות כי מספר החברות המכילות עובדים בהכרח
קטן ממספר העובדים הכללי .

2.2.1 מבנה נתונים $Company$

־ מכיל את השדות
1) $company\_id$: מספר החברה .
2) $value$: הערך של החברה .
3) $main$: מצביע לחברה בעץ החברות הכללי
4) $employees$: טיפוס ממבנה הנתונים $EmployeesContainer$ האחראי לניהול העובדים בחברה .
־ מבנה נתונים המייצג חברה ומטרתו לאפשר פעולות של גישה וקידום עבור חברות ובנוסף מאפשר טיפול בעובדים .
־ המטרה של השדה $main$ היא שכאשר אנחנו אנו מוצאים חברה בעץ החברות עם עובדים נוכל לגשת לחברה בעץ החברות הכללי ללא חיפוש .

2.3 עץ AVL דו כיווני גנרי :

־ גנריות לפי טיפוס הנתונים אותו מאחסנים בעץ .
־ האיברים בעץ הם $AVLNode$ שגנריים לפי טיפוס ה־$data$ (T) ,
כל $Node$ יכיל את השדות :

1) $data$ - מצביע למידע אותו שומרים בעץ .
2) $height$ - הגובה של ה- $node$ הנוכחי .
3) $left, right, parent$ - מצביעים ל-$nodes$ המתאימים בעץ .
- העץ עצמו מכיל שני שדות :
1) $root$- מצביע לצומת המהווה השורש של העץ .
2) $number\_of\_elements$ - מספר האיברים בעץ.

2.3.1 מתודות :

1) $insertAVL$ -הכנסת איבר בסיבוכיות ותחזוק מספר האיברים בעץ בסיבוכיות זמן $O(log(n))$.
2) $deleteAVL$ - מחיקת איבר ותחזוק השדה של מספר האיברים בעץ בסיבוכיות זמן $O(log(n))$.
$findAVL$ - מציאת איבר בסיבוכיות זמן $O(log(n))$.
3) $getMax$ - שולח מצביע לאיבר המקסימאלי בסיבוכיות זמן $O(log(n))$.
** הפעולות הנל לא רקורסיביות ולכן סיבוכיות המקום היא $O(1)$.**
4) $getNumOfElements$ -שולח את מספר האיברים בעץ בסיבוכיות זמן ומקום של $O(1)$
.
5) $mergeTrees$- מיזוג שני עצים לעץ אחד .
- נכניס את איברי כל אחד מהעצים באמצעות סיור $postrder$ למערך
, $O(n+m)$ עבור הסיור לכל עץ והמעבר על המערך .
- שימוש במתודה $MergeSort$ למיזוג שני המערכים למערך אחד
, $O(n+m)$ כגודל כל אחד מהמערכים .
- שבירת המעגל עבור כל אחד מהעצים הישנים באמצעות סיור $inorder$
(לגרום לאב להצביע ל-$nullptr$) , $O(n+m)$.
בסהכ הסיבוכיות המתקבל היא $O(n+m)$.
- בניית עץ שלם חדש ממערך ממויין , הכנסה שיטתית של האיברים למיקומם הנכון בצורה רקורסיבית , $O(n+m)$.
בסה"כ סיבוכיות הזמן והמקום הכוללת היא $O(n+m)$.
6) $reversedElemToArr$- הכנסת איברי העץ בסדר הפוך ל-$inorder$.
- נעבור על איברי העץ בסיור $inorder$ ונכניסם למערך , $O(n)$.
- נכניס את האיברים למערך נוסף באמצעות מעבר על שני המערכים $O(n)$.
.בסהכ סיבוכיות הזמן והמקום המתקבלת היא $O(n)$.
7) $limitedIDataToTypeArray$ - פונקציה גנרית (טיפוס ופעולה) המכניסה את איברי העץ
למערך מתאים לאחר ביצוע הפעולה .
- ביצוע עם סיור $inorder$ תוך כדי ספירה של האיברים בסיור והוספת תנאי עצירה כשמגיעים למספר האיברים הרצוי .
בסהכ עבור הסיור ותנאי העצירה המגביל אותו סיבוכיות המקום והזמן היא $O(n_{elements})$
.
8) $RangeElemToSortedArray$ - הכנסת איברי העץ למערך בהתאם ל-2 איברים המהווים טווח והחזרת מספר איברים בטווח זה .
- ספירת האיברים באמצעות האלגוריתם , $O(log(n)+n_{elements})$.
- הקצאת מערך בגודל מספר האיברים שנמצא , $O(n_{elements})$
-הכנסת האיברים למערך באמצעות האלגוריתם , $O(log(n)+n_{elements})$.
לכן הסיבוכיות המקום והזמן של הפונקציה היא $O(log(n)+n_{elements})$.

האלגוריתם (רקורסיבי) :
- אם האיבר בתווך תתנהג כאילו אנחנו בסיור $inorder$ רגיל .
- אם האיבר קטן מהמינימום תלך ימינה .
- אם האיבר גדול מהמקסימום לך שמאלה .
כלומר אנחנו מבצעים חיפוש של איבר בטווח המתאים וברגע שנמצא סופרים את האיברים המתאמים
עד שמגיעים לאיבר לא מתאים ומבצעים שוב חיפוש לאיבר הבא שמתאים .
בסהכ הסיבוכיות הזמן והמקום של הפעולה היא $O(log(n) + n_{element})$ עבור החיפוש והמעבר על איברי הטווח .

3 מימושים של פעולות מבנה הנתונים :

3.1 void* init():

-הקצאת זכרון עבור ה-DS , לא מקצים עצמים באיתחול אלא רק מבנים ריקים .
בסהכ סיבוכיות הזמן והמקום היא $O(1)$.

3.2 StatusType AddCompany(void* DS, int CompanyID, int Value):

- איתחול אובייקט ריק מטיפוס $Company$, $O(1)$.
- הוספת האיבר לעץ החברות הכללי ב-$CompanyContainer$, $O(log(k))$.
בסהכ סיבוכיות הזמן היא $O(log(k))$.
*בסהכ סיבוכיות המקום $O(1)$ (ההוספה בעץ אין רקורסיבית) .

3.3 StatusType AddEmployee(void *DS, int EmployeeID, int CompanyID, int Salary, int Grade):

- חיפוש העובד בעץ העובדים הכללי , זמן - $O(log(n))$.
- חיפוש החברה בעץ החברות הכללי זמן - $O(log(k)$.
- הוספת העובד לכל אחד מ- 2 מעצי העובדים הכללי , $O(log(n))$.
- תחזוק אם העובד הוא הכי רווחי בחברה ועדכון השדה של המצביע לאובייקט בעץ העובדים הכללי , $O(1)$.
- הוספת העובד לעץ החברות של החברה בעץ החברות הכללי , $O(log(n))$
(מספר העובדים בחברה חסום על ידי כלל העובדים) .
- בדיקה אם מספר העובדים לאחר ההוספה הוא 1 , $O(1)$.
- במידה וכן הוספת החברה לעץ החברות עם עובדים ועדכון השדה של מצביע לחברה בעץ הכללי , $O(log(k))$.
*בסהכ סיבוכיות הזמן היא $O(log(n) + log(k))$ ומקום $O(1)$.

3.4 StatusType RemoveEmployee(void *DS, int EmployeeID):

- חיפוש העובד בעצי העובדים הכלליים , זמן - $O(log(n))$.
- גישה לשדה של מספר החברה אצל העובד וחיפוש בעץ

החברות המכילות לפחות עובד 1 , $O(log(n))$.
- קבלת מצביע לחברה בעץ החברות הכללי מהשדה של החברה שנמצאה $O(1)$.
-ביצוע מחיקה מכל עצי העובדים $O(log(n))$.
*בסהכ סיבוכיות זמן $O(log(n))$ ומקום $O(1)$.

3.5 StatusType RemoveCompany(void *DS, int CompanyID):

-חיפוש בעץ החברות המכילות לפחות עובד 1 , $O(log(k))$.
- במידה ונמצא נחזיר שגיאה .
- חיפוש בעץ החברות הכללי , $O(log(k))$.
- מחיקה מכל אחד מהעצים , $O(log(k))$.
*בסהכ סיבוכיות זמן $O(log(k))$ ומקום $O(1)$.

3.6 StatusType GetCompanyInfo(void *DS, int CompanyID, int *Value, int *NumEmployees)

- חיפוש בעץ החברות הכללי , $O(log(k))$.
- קבלתם המידע מהחברה בגישה ישירה ועדכון , $O(1)$.
*בסהכ סיבוכיות זמן $O(log(k))$ ומקום $O(1)$.

3.7 StatusType GetEmployeeInfo(void *DS, int EmployeeID, int *EmployerID, int *Salary, int *Grade):

- חיפוש בעץ העובדים הכללי , $O(log(n))$.
- קבלתם המידע מהעובד בגישה ישירה ועדכון , $O(1)$.
*בסהכ סיבוכיות זמן $O(log(n))$ ומקום $O(1)$.

3.8 StatusType IncreaseCompanyValue(void *DS, int CompanyID, int ValueIncrease):

- חיפוש בעץ החברות הכללי , $O(log(k))$.
- קבלתם המידע מהחברה בגישה ישירה ועדכון , $O(1)$.
*בסהכ סיבוכיות זמן $O(log(k))$ ומקום $O(1)$.

3.9 StatusType PromoteEmployee(void *DS, int EmployeeID, int SalaryIncrease, int BumpGrade):

- חיפוש העובד בעצי העובדים הכלליים , זמן - $O(log(n))$.
- גישה לשדה של מספר החברה אצל העובד וחיפוש בעץ
החברות המכילות לפחות עובד 1 , $O(log(n))$.
- קבלת מצביע לחברה בעץ החברות הכללי מהשדה של החברה שנמצאה $O(1)$.

- מחיקת העובד מכל אחד מהעצים לפי שכר , $O(log(n))$.
- עדכון השכר ועדכון הדרגה בכל העצים (במידה והתנאי מתקיים) בגישה ישירה , $O(1)$.
- הוספת העובד לכל אחד מהעצים ועדכון השדות של המצביעים גם של העובד וגם של החברות , $O(1)$.
- תחזוק שדה של העובד הרווחי ביותר , $O(log(n))$.
*בסהכ סיבוכיות הזמן של $O(log(n))$.

3.10 StatusType HireEmployee(void *DS, int EmployeeID, int NewCompanyID):

- חיפוש העובד בעצי העובדים , $O(log(n))$.
- חיפוש החברה החדשה בעץ החברות הכללי , $O(log(k))$.
- שימוש בפונקציה $DeleteEmployee$ עבור החברה הישנה , $O(log(n))$.
- שימוש בפונקציה $AddEmployee$ עבור העובד והחברה החדשה , $O(log(n) + log(k))$.
- תחזוק שדה העובד הרווחי ביותר , $O(log(n))$.
*בסהכ הסיבוכיות היא $O(log(n) + log(k))$.

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

- חיפוש החברות בעץ החברות הכללי , $O(log(k)$.
- בדיקה אם החברה יכולה להרכש , $O(1)$.
נפריד למקרים :
1. אם החברה יכולה להרכש ואין לה עובדים נעדכן את הערך של החברה הרוכשת .
2.נשתמש בפונקציה $mergeTree$ עבור העצים , $O(n_{comp1} + n_{comp2})$.
- אם לחברה הרוכשת לא היה עובדים נכניס אותה לעץ החברות עם עובדים , $O(log(k))$.
בסהכ : $O(log(k) + n_{comp1} + n_{comp2})$

3.12 StatusType GetHighestEarner(void *DS, int CompanyID, int *EmployeeID)

- אם $CompanyId > 0$ נבצע חיפוש בעץ החברות עם עובדים , $O(log(k))$.
- במקרה של $CompanyId > 0$ ניגש לעץ החברות הכללי $O(1)$.
- נקבל את עובד בגישה ישירה מתחזוק השדה $O(1)$.
*בסהכ במקרה החיובי , $O(log(k)$ ומקרה הכללי $O(1)$.

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

- אם $CompanyId > 0$ נבצע חיפוש בעץ החברות עם עובדים , $O(log(k))$.
- במקרה של $CompanyId > 0$ ניגש לעץ החברות הכללי $O(1)$.

־נשתמש ב־$reversedElemArray$ של העץ עבור עץ העובדים לפי שכר , $O(n)$.
־ נכניס את תעודת הזהות של העובדים למערך המבוקש , $O(n)$.
** נשים לב שבגלל הדרך בה מיינו את העובדים בעץ
לפי השכר נקבל אותם לפי הדרישה .**
־ נעדכן את הערך של מספר העובדים , $O(1)$.
*בסהכ במקרה הכללי $O(n)$ ובמקרה של חברה ספציפית $O(log(k)+n)$.

3.14 StatusType GetHighestEarnerInEachCompany(void *DS, int NumOfCompanies, int **Employees)

־ נקצה מערך בגודל המתאים , מקום ־ $O(NumOfCompanies)$.
־ נשתמש בפונקציה $LimitedDataTypeArray$ על עץ החברות עם עובדים
, $O(log(k)+NumOfCompanies)$.
*מעבירים לפונקציה אובייקט הממש אופרטור () המקבל $Company$ ומחזיר את תעודת הזהות של העובד הרווחי ביותר .
*סיבוכיות המקום והזמן הכוללת היא $O(log(k)+NumOfCompanies)$.

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

־ אם $CompanyId > 0$ נבצע חיפוש בעץ החברות עם עובדים , $O(log(k))$.
־ במקרה של $CompanyId > 0$ ניגש לעץ החברות הכללי $O(1)$.
־ ניצור אובייקטים של עובדים לפי תעודת הזהות שנתנו , $O(1)$.
** בשני המקרים נמשיך אותו דבר החל משלב זה **
־ נשתמש בפונקציה $RangeElemToSortedArray$ של עץ
העובדים לפי תזעל מנת לקבל את העובדים הנ"ל ואת מספרם
במערך , $O(log(n)+TotalNumOfEmployees)$
־ מעבר על האיברים במערך ,ספירתם ובדיקת התנאי עבור
הדרגה והשכר , $O(TotalNumOfEmployees)$.
*בסהכ במקרה שחיפשנו חברה $O(log(k)+O(log(n)+TotalNumOfEmployees)$
ובמקרה שלא $O(log(n)+TotalNumOfEmployees)$.

3.16 void Quit(void **DS)

־ בכל רגע נתון יש לנו בסה"כ $2k$חברות עבור 2 עצי החברות ו־$4n$ עובדים עבור
העצים של החברות והכליים של העובדים ולכן סיבוכיות המקום ברגע זה היא $O(n+k)$.
־ בשחרור זכרון נבצע מעבר רקורסיבי באמצעות $PostOrder$ ונשחרר את הזכרון
שהוקצה לעצמים בכל אחד מהעצים ושינוי מצביע האב לשבירת המעגל .
*בסהכ עברנו על כלל האיברים הקיימים באופן רקורסיבי ולכן סיבוכיות הזמן והמקום
היא $O(n+k)$.