**Wet 1 - Dry Part - Data Structure (234218)**

**מגישים:** איתמר ארביב - 206622979, דניאל ברנסקי - 204866149

**בנינו את מחלקות העזר הבאות:**

| Employee - עובד | | |
|---|---|---|
| Member Name | Member Type | משמעות המשתנה |
| id | int O(1) | מספרו הסידורי של העובד |
| salary | int O(1) | המשכורת של העובד |
| grade | int O(1) | הדרגה של העובד |
| company | Company* O(1) | מצביע לחברה בה מועסק העובד |

**סיבוכיות מקום של המבנה:** $O(1)$

| Company - חברה | | |
|---|---|---|
| Member Name | Member Type | משמעות המשתנה |
| id | int O(1) | מספרו הסידורי של החברה |
| value | int O(1) | ערך החברה |
| amount_of_employees | int O(1) | כמות העובדים בחברה |
| best_salary_employee | Employee* O(1) | מצביע לעובד המרוויח ביותר אשר שייך לחברה |
| company_employees | Map<Employee *, SalaryId> O(n) | Map המכיל מצביעים לעובדים של החברה, ממוינים לפי השכר הגבוה בעדיפות ראשונה ו- ID נמוך בעדיפות שנייה |
| employees_id | Map<Employee *, int> O(n) | Map המכיל מצביעים לעובדים של החברה, ממוינים לפי ה- ID הנמוך ביותר |

**סיבוכיות מקום של המבנה:** $O(n)$ , כאשר n = מספר העובדים בחברה.

| Pair צמד | | |
|---|---|---|
| Member Name | Member Type | משמעות המשתנה |
| element | T-template O(1) | האיבר ששמור בצמד |
| key | Key-key O(k) | המפתח של הצמד |

**סיבוכיות מקום של המבנה:** $O(k)$ (אך במקרה שלנו כל המפתחות יהיו מטיפוסים בעלי סיבוכיות של k=1, כלומר המשתמש יכול לספק מפתח בעל סיבוכיות מקום גבוהה יותר מ1)

**אופן החישוב:** $O(1 + k) = O(k = 1) = O(1)$

| צומת Node | | |
|---|---|---|
| Member Name | Member Type | משמעות המשתנה |
| h_left | int O(1) | גובה תת העץ השמאלי |
| h_right | int O(1) | גובה תת העץ הימני |
| balance_factor | int O(1) | מייצג את רמת האיזון של העץ |
| left | Node* O(1) | מצביע לבן שמאלי |
| right | Node* O(1) | מצביע לבן ימני |
| father | Node* O(1) | מצביע לאב |
| pair | Pair O(k) | מפתח ואיבר |

**סיבוכיות מקום של המבנה:** $O(1)$ (כאשר נתייחס לצומת עצמו ללא התחשבות בבנים) שוב כתלות בטיפוס המפתח נקבל שpair בכל סיבוכיות מקום של k אך במקרה שלנו כל המפתחות הם $O(1)$

**אופן החישוב:** $O(1 + 1 + 1 + 1 + 1 + k) = O(k = 1) = O(1)$

| מפה אשר ממומשת ע"י עץ Map - AVL | | |
|---|---|---|
| Member Name | Member Type | משמעות המשתנה |
| amount | int O(1) | כמות הצמתים בעץ |
| head | Node<T,Key> $O(n)$ | מצביע לצומת שהוא שורש העץ |

**סיבוכיות מקום של המבנה:** $O(n)$, כאשר n = כמות הצמתים בעץ.

| אוסף חברות ההייטק ואוסף העובדים - HighTech | | |
|---|---|---|
| Member Name | Member Type + Place | משמעות המשתנה |
| total_amount_of_employees | int O(1) | כמות העובדים הכוללת |
| amount_of_companies_with_at_least_one_employee | int O(1) | כמות החברות שיש בהן לפחות עובד אחד |
| amount_of_companies | int O(1) | כמות החברות הכולל |
| employees_sorted_by_id | Map<Employee *, int> O(n) | MAP המכיל מבציעים לעובדים של כלל החברות, ממוינים לפי המזהה שלהם |
| employees_sorted_by_salary | Map<Employee *, SalaryId> O(n) | Map המכיל מבציעים לעובדים של כלל החברות, ממוינים לפי השכר הגבוה בעדיפות ראשונה ו- ID נמוך בעדיפות שנייה |

| | | |
|---|---|---|
| best_earning_employees | Map<Employee *, EmployeeByCompanyId><br>O(n) | Map המכיל מצביעים של העובדים המרוויחים ביותר של כל החברות שיש בהן עובדים, ממוינים לפי המזהה של החברה בה הם עובדים |
| employee_with_best_salary | Employee*<br>O(1) | מצביע לעובד המרוויח ביותר מכלל החברות |
| companies | Map<Company *, int> O(m + n)<br>• ראה הערה | MAP המכיל מבציעים לחברות, ממוינים לפי המזהה שלהם |

**סיבוכיות מקום של המבנה:** $O(n + m)$, כאשר n = כמות העובדים הכולל ו- m כמות החברות הכולל.

**אופן החישוב:** $O(1 + 1 + 1 + n + n + n + 1 + m + n) = O(n + m)$

**הערה:** אמנם בעץ של החברות כל חברה מחזיקה עץ נוסף של העובדים שלה אבל מתקיים שסכום כל העובדים בכל החברות שווה בדיוק ל- n, מכיוון שכל עובד יכול לעבוד בחברה אחת בדיוק ולכן כמות הצמתים בסך כל העצים של העובדים בכל חברה שווה בדיוק ל- n. * באופן כללי, כל עובד יכול להיות במקרה הגרוע ביותר ב- 4 עצים, ולכן סיבוכיות המקום עבור n שחקנים זה $O(n)$.

## ניתוח סיבוכיות

**הפעולות במחלקה Map:**

כעת ננתח את סיבוכיות הזמן הפעולות של העץ אשר נתממשק עימו בהמשך.

פונקציות עזר:

GetNode(head,key)

פונקציית עזר אשר מקבלת את צומת ומפתח ומחזירה את הצומת בעל המפתח המבוקש בתת העץ הנתון. אם לא קיים איבר עם מפתח זה מחזירה NULL.

כפי שהוכח הרצאה זמן הריצה של מציאת איבר בעץ avl הוא $O(logn)$

ArrayFromTree()

מתודת עזר אשר מחזירה מערך המכיל את איברי העץ בצורה ממוינת (pairs).

רצים בInorder. כפי שהוכח בהרצאה מעבר על כל הצמתים בinorder הוא בעל

סיבוכיות זמן של $O(n)$

MergeSortedArrays(arr1,arr2)

פונקציית עזר המקבל שני מערכים ממוינים רצה על כל האיברים ומחזירה מערך ממוזג ממוין. פעולה זו רצה על שני המערכים במקביל כאשר לכל מערך אינדקס שלו המתקדם אם המערך מכיל את האיבר הקטן יותר. אנו עובדים על כל איברים המערכים ולכן סיבוכיות הזמן שלו הוא $O(n + m)$

TreeFromArray(father,arr,min_index,max_index)

מתודת עזר המקבלת מערך המכיל pairs ומחזירה Node אשר ראש העץ נוצר מהמערך. פעולה זו ממומשת בעזרת רקורסיה.

תחילה נחשב את ערכו של הmid_index אשר מכיל את ערך של ראש העץ וניצור node חדש שהאב שלו הוא father, כעת נגדיר את node.left להיות הערך המוחזר מ

TreeFromArray(node,min_index,mid_index-1)

ואת node.right לערך המוחזר מ

TreeFromArray(node,mid_index+1,max_index)

לבסוף נחזיר את Node. כך אנו מקצים את תת העץ הימני בכל פעם להיות חצי המערך השמאלי כנל לגבי צד ימין.

פעולה זו רצה על כל אברי שני המערכים ולכן סיבוכיות הזמן שלה היא $O(n + m)$

פעולות המחלקה:

Map()

בנאי של העץ, מגדיר את head להיות null וamount ל0 סיבוכיות זמן $O(1)$

bool does_exist(Key key)

מחזיר true אם קיים איבר עם המפתח במפה, נמצא את האיבר במפה בעזרת GetNode ונחזיר true אם הוא לא null אחרת false סיבוכיות זמן $O(logn)$

T find(Key key)

בהינתן מפתח מחזירה את בעץ בעל מפתח זה.

שוב נשתמש בGetNode ונחזיר Null אם לא קיים בעץ.

סיבוכיות זמן $O(logn)$

void insert(Key key,T element)

בהינתן מפתח ואיבר מכניסה אותו לתוך העץ avl. כפי שהוכח בהרצאה סיבוכיות זמן $O(logn)$

void remove(Key key)

בהינתן מפתח מסירה את האיבר בעל מפתח זה מהעץ שוב כפי שהוכח בהרצאה סיבוכיות זמן $O(logn)$

void merge(Map&)

מתודה המקבלת מפה אחרת וממזגת את האיברים שלה לתוך המפה עליה הופעל

תחילה נמיר את שני הערכים למערכים ArrayFromTree $O(n+m)$

לאחר מכן נמזג את שני המערכים MergeSortedArray $O(n+m)$

לבסוף נמיר את המערך הממוזג לעץ החדש TreeFromArray $O(n+m)$

ולכן בסך הכל פעולה זו בעלת סיבוכיות זמן של $O(n+m)$

GetMaxId()

מתודה המחזירה את האיבר עם המפתח הגדול ביותר, יורדת בעץ לצד ימין שפוגשת null, פעולה זו עוברת על לכל היותר כמות צמתים כגובה של העץ ולכן סיבוכיות הזמן שלה הינה $O(logn)$

GetFirstNum(num)

בהינתן מספר num מחזירה מערך המכיל את num האיברים הראשונים של המפה

פונקציה זו רצה בinorder (בצורה ממוינת) על העץ ומכניסה את האיברים למערך עם תנאי עצירה כאשר הindex מגיע ל-num1. פונקציה זו רצה על Num איברים בדיוק ולכן סיבוכיות הזמן שלה הינה $O(num)$

GetObjectFromKey(min_key,max_key)

מתודה המחזירה מערך של כל הצמתים בעץ אשר המפתח שלהם בין min_key לmax_key

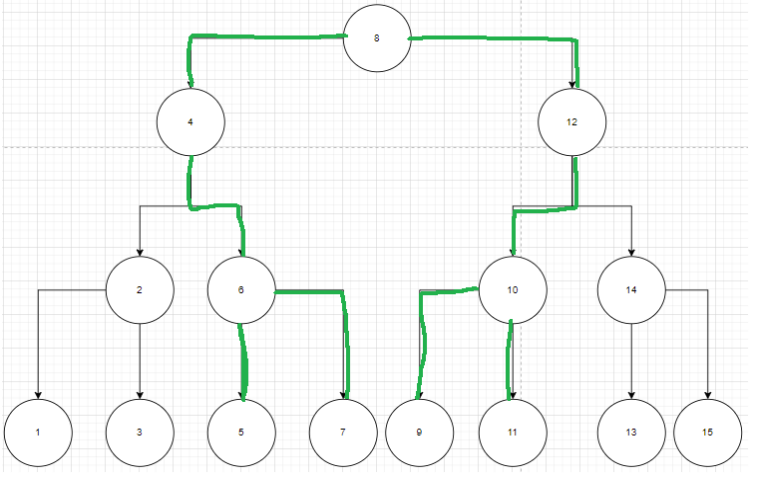
מתודה זו רצה ברקורסיה inorder בצורה הבאה:

אילו הצומת גדול ממש min_keyמ הרקורסיה יורדת שמאלה

אילו הצומת בטווח מוסיפים אותו למערך

אילו הצומת קטן ממש max_keyמ הרקורסיה יורדת ימינה

נתבונן בדוגמה הבאה קבלת כל המפתחות בין 5-11




תחילה 8 גדול מ5 ולכן נרד שמאלה, 4 אינו בטווח וקטן מ11 ולכן נרד ימינה, 6 גדול מ5 ולכן נרד שמאלה 5 הוא עלה וגם בטווח ולכן נכניס אותו למערך במקום הראשון, 6 בטווח ולכן נכניס אותו למערך וגם קטן מ11 ולכן נרד ל7 אשר נמצא בטווח ולכן נכניס אותו למערך וכן הלאה...

בסך הכל פעולה זו. נחשב את סיבוכיות הזמן שלה הפעולה, נסמן ב num כמות הצמתים אשר שייכים לטווח ובח את סך הצמתים בעץ. נתחיל ממציאת הצומת הראשון אשר שייך לטווח, בסך הכל כדי להגיע אליו במקרה הגרוע ביותר הוא כגובה העץ כלומר $O(logn)$

במקרה הקיצוני השני הוא ראש העץ, נתפצל ממנו ימינה ושמאלה ונספור ביקור ראשון בצומת בטווח,

כעת אילו בירידה ימינה קיים עוד איבר בטווח אזי ניתן לומר בוודאות שכל הצמתים שמשמאלו שייכים לטווח ולכן ביקור בהם נספר גם כן במניין הצמתים אשר בטווח, אחרת אם האיבר לא בטווח אז נמשיך לרדת בעץ, כך שבמקרה הגרוע ביותר נבקר בlogn צמתים אשר לא בטווח (בתת עץ הימני)

בצורה אנלוגית נבקר בתת עץ השמאלי, שוב אם קיים עוד צומת בטווח אזי ניתן לומר בוודאות שכל הצמתים הימנים שלו בטווח ונספרים במניין num וכל הצמתים שלא בטווח גם כגובה העץ.

לסיכום נבקר במקרה הגרוע ביותר בnum+2logn צמתים ולכן בסך הכל סיבוכיות הזמן של פעולה זו היא $O(logn+num)$

~Map()
הורס של המפה, רץ בצורת postorder ומשחרר את כל הצמתים שהוקצו באופן דינאמי.
בסך הכל מבקר בכל צומת פעם אחת וכפי שהוכח בהרצאה postorder הוא בעל סיבוכיות זמן של $O(logn)$

**הפעולות במחלקה HighTech:**

void* Init() = HighTech()
בנאי ללא פרמטרים של המחלקה, המאתחל את כל השדות של המחלקה, ומאתחלת את העובד המרוויח ביותר להיות NULL (מה שקורה בסיבוכיות זמן קבועה).
הערה: הראנו שאתחול MAP אשר ממומש ע"י עץ AVL לוקח O(1).
חישוב הסיבוכיות: $O(1 + 1 + 1 + 1 + 1 + 1 + 1 + 1) = O(1)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(1)$
void Quit(void** DS) = ~HighTech()
הורס ללא פרמטרים של המחלקה, המשחרר את כל הזיכרון אשר הוקצה ע"י המחלקה. ההורס עובר על כל הצמתים של העצים: של העובדים ושל החברות ומשחרר אחד אחד.
תחילה אנחנו עוברים כל אחד מהעצים ושמים את המידע במערך (באמצעות הפעולה GetFirstNum על שני העצים שממויינים לפי ID - במצב זה היא עוברת על כל הצמתים של 2 העצים) ואז עוברים על המערך המתקבל ומשחררים איבר איבר. שחרור איבר בעצים שלנו מתבצע בסיבוכות זמן ריצה קבוע, אך מכיוון שאנחנו עוברים בסה"כ על n + m איברים, נקבל סיבוכיות זמן של n + m.
ולכן: סה"כ סיבוכיות זמן ריצה: $O(n + m)$
מספר העובדים - n מספר החברות - m
void AddCompany(int CompanyId, int Value)
מוסיפה חברה חדשה (וריקה) למבנה הנתונים בתוך עץ החברות.
במידה והקלט אינו תקין, נזרקת חריגה מתאימה, בדיקת הקלט מתבצע בסיבוכיות קבועה.
לאחר מכן אנחנו משתמשים בפעולה insert שמימשנו (המבצעת הכנסה בסיבוכיות זמן ריצה לגורתימית בהתאם לגודל העץ כפי שציינו מעלה), על העץ של החברות.
במידה והחברה כבר קיימת נזרקת חריגה מתאימה, ובמידה ולא, היא נכנסת לעץ בהצלחה.
לאחר מכן אנחנו מעדכנים את כמות החברות בהתאם - זמן קבוע.
חישוב הסיבוכיות: $O(log\, m + 1) = O(log\, m)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(log\, m)$ m - מספר החברות
**פעולת עזר במחלקה HighTech**
void UpdateInCompany(Employee *employee, Company *company)
פעולת עזר המעדכנת את פרטי העובד בחברה שלו.
כלומר, הפעולה בודקת אם העובד הוא הראשון בחברה שלו, במידה וכן היא קובעת אותו להיות העובד המרוויח ביותר בחברה ומכניסה את עובד זה לעץ המכיל את העובד המרוויח ביותר מכל חברה (וכמובן מעדכנת את כמות החברות שיש בהן לפחות עובד אחד בהתאם).
לאחר מכן הפעולה בודקת האם העובד הוא המרוויח ביותר בחברה שלו, ובמידה וכן מעדכנת את העובד המרוויח ביותר בחברה להיות הוא ומעדכנת את עץ העובדים המרוויחים ביותר בהתאם (מוחקת באמצעות remove את העובד הקודם - מתבצע בסיבוכיות זמן ריצה לוגריתמית) ולאחר מכן מכניסה אותו באמצעות insert מה שמתבצע בסיבוכיות זמן ריצה לוגרתימית. (זהו המקרה הגרוע).

- פעולת ההשוואה בין שני עובדים (לגבי מי מרוויח יותר) פועלת כמובן בסיבוכיות זמן קבועה.

חישוב הסיבוכיות (במקרה הגרוע): $O(1 + log\, m + log\, m) = O(1)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(log\, m)$ m - מספר החברות
מכיוון שבעץ המכיל את העובד המרוויח ביותר, יש לכל היותר בעץ m איברים, כמספר החברות.
void AddEmployee(int EmployeeId, int CompanyID, int Salary, int Grade)
מוסיפה עובד חדש למבנה הנתונים בעצים המתאימים שיפורטו להלן:
במידה והקלט אינו תקין, נזרקת חריגה מתאימה, בדיקת הקלט מתבצע בסיבוכיות קבועה.
תחילה אנחנו מחפשים בעץ החברות את החברה שהמזהה שלה הוא CompanyId באמצעות הפעולה find - סיבוכיות זמן ריצה לוגרתימית. במידה והחברה לא קיימת, נזרגת חריגה מתאימה.
לאחר מכן (במידה והחברה כן קיימת בעץ החברות) אנחנו משתמשים בפעולה insert שמימשנו (המבצעת הכנסה בסיבוכיות זמן ריצה לגורתימית כפי שציינו מעלה), על העץ של העובדים אשר ממוין לפי ID.
במידה וקיים כבר עובד עם מזהה זה - נזרקת חריגה מתאימה, ובמידה ולא, העובד נכנס לעץ הנ"ל בהצלחה ולאחר מכן מכניסים את העובד לעץ של העובדים אשר ממוינים לפי השכר (ו- ID בעדיפות שנייה) - מה שמבתצע גם בסיבוכיות זמן ריצה לוגרתימית.
לאחר מכן אנחנו מכניסים את העובד לעץ של העובדים בתוך החברה (גם לעץ הממוין לפי ID וגם לעץ הממויין לפי SalaryID) שכל אחד מהם מתבצע בסיבוכיות זמן לוגריתמית, לאחר מכן אנחנו מגדילים את כמות

העובדים בחברה (בסיבוכיות זמן קבועה - באמצעות Set השייך למחלקה Company הניגש ישירות לתכונה).
לאחר מכן אנחנו משתמשים בפעולה העזר UpdateInCompany שפורטה מעלה, הפועלת בסיבוכיות זמן ריצה של (log m), ולאחר מכן אנחנו מעדכנים את כמות העובדים בכולל במבנה הנתונים שלנו ולבסוף קובעים את העובד המרוייח ביותר במבנה לפי העץ המכיל את העובד המרוויח ביותר בכל חברה, באמצעות הפעולה GetMaxId שפורטה מעלה ופועלת בסיבוכיות זמן ריצה של log m.
חישוב הסיבוכיות (במקרה הגרוע):
$O(\log m + \log n + \log n + \log n + \log n + \log m + 1 + 1 + \log n) = O(\log n + \log m)$
לכן סה"כ סיבוכיות זמן ריצה: $O(\log m + \log n)$ m - מספר החברות n - מספר העובדים

void RemoveEmployee(int employee_id)

מסירה עובד ממבנה הנתונים בעצים המתאימים שיפורטו להלן:
במידה והקלט אינו תקין, נזרקת חריגה מתאימה, בדיקת הקלט מתבצע בסיבוכיות קבועה.
לאחר מכן הפעולה מבצעת find לעובד בעץ העובדים, ובמידה ולא קיים זורקת חריגה.
במידה והעובד קיים, הפעולה מסירה את העובד משני העצים של העובדים המכילים את כלל העובדים ולאחר מכן מסירה אותו מהעצים השייכים לחברה בה הוא היה מועסק - העובד מכיל מצביע לחברה ולכן לא צריך לחפש את החברה בעץ העובדים - מציאת החברה של העובד מתקיימת בסיבוכיות זמן קבועה.
במידה והוא היה העובד המרוויח ביותר בחברה או בכלל, מתבצעים עדכונים בהתאם (כפי שבוצעו ב-AddEmployee).
לאחר מכן גם מעדכנים את כמות העובדים הכולל במבנה הנתונים שלנו ובמידה והחברה גם אינה מכילה כעת עובדים, אז מעדכנים את כמות החברות שמעסיקים עובדים בהתאם.
במקרה הגרוע ביותר, כמות העובדים בחברה שווה ל- n (כמות העובדים בכלל), ובמקרה גרוע אחר, כמות האיברים בעץ של המרוויחים ביותר הוא n (כאשר כל עובד שייך לחברה אחרת). ולכן:
חישוב הסיבוכיות (במקרה הגרוע):
$O(\log n + \log n + \log n + \log n + \log n + 1 + \log n + \log n + \log n) = O(\log n)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(\log n)$ n - מספר העובדים

void RemoveCompany(int company_id)

מסירה חברה ממבנה הנתונים בעצים המתאימים שיפורטו להלן:
במידה והקלט אינו תקין, נזרקת חריגה מתאימה, בדיקת הקלט מתבצע בסיבוכיות קבועה. לאחר מכן אנחנו משתמשים בפעולה remove שמימשנו (המבצעת הוצאה בסיבוכיות זמן ריצה לגורתימית כפי שציינו מעלה), על העץ של החברות אשר ממוין לפי ID.
במידה ולא קיימת חברה עם המזהה - נזרקת חריגה מתאימה, ובמידה ולא, ההסרה מתבצעת בהצלחה.
לאחר מכן אנחנו מעדכנים את כמות החברות - סיבוכיות זמן קבוע.
ולאחר מכן משחררים את החברה מהזיכרון - סיבוכיות זמן קבוע.
חישוב הסיבוכיות (במקרה הגרוע): $O(\log m + 1 + 1) = O(\log m)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(\log m)$ m - מספר החברות

void GetCompanyInfo(int company_id, int *Value, int *NumEmployees)

פעולה המחזירה את פרטי החברה (ערך + כמות העובדים בה) בעץ החברות.
במידה ואחד הקלטים אינו תקין נזרגת חריגה מתאימה. במידה ולא קיימת חברה כזו - נזרקת חריגה.
הפעולה מבצעת פעולת find על העץ של החברות הממוינים לפי ID, ובודקת האם הוא קיים בעץ.
כפי שהראנו מעלה, פעולת ה- find עובדת בסיבוכיות זמן ריצה של log m, כאשר m - מספר החברות הכולל, ולאחר מכן מוציאה את הפרטים הרצויים עליו, בסיבוכיות זמן ריצה קבוע (באמצעות Get של כל תכונה במחלקה Company, המתבצע בסיבוכיות זמן קבועה).
חישוב הסיבוכיות: $O(\log m + 1 + 1) = O(\log m)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(\log m)$ m - מספר החברות

void GetEmployeeInfo(int EmployeeId, int *EmployerID, int *Salary, int *Grade)

פעולה המחזירה את פרטי העובד (משכורת + שם החברה בה עובד + הדרגה) בעץ העובדים.
במידה ואחד הקלטים אינו תקין נזרגת חריגה מתאימה. במידה ולא קיים עובד כזה - נזרקת חריגה.
הפעולה מבצעת פעולת find על העץ של העובדים הממוינים לפי ID, ובודקת האם הוא קיים בעץ.
כפי שהראנו מעלה, פעולת ה- find עובדת בסיבוכיות זמן ריצה של log n, כאשר n - מספר העובדים הכולל. ולאחר מכן מוציאה את הפרטים הרצויים עליו, בסיבוכיות זמן ריצה קבוע (באמצעות Get של כל תכונה במחלקה Employee, המתבצע בסיבוכיות זמן קבועה).
חישוב הסיבוכיות: $O(\log n + 1 + 1 + 1) = O(\log n)$
ולכן סה"כ סיבוכיות זמן ריצה: $O(\log n)$ n - מספר העובדים

void IncreaseCompanyValue(int CompanyId, int ValueIncrease)

הפעולה מעדכנת את הערך של החברה שמספר המזהה שלה הוא companyId. במידה ואחד מהקלטים אינו תקין, נזרקת חריגה. הפעולה מחפשת בעץ של החברות את החברה המתאימה באמצעות find (בסיבוכיות של log m, כאשר m מספר החברות). במידה והחברה לא נמצאת נזרקת

חריגה מתאימה. לאחר מכן אנחנו מעדכנים את ערך החברה הנ"ל בהתאם, מה שמתבצע באמצעות סיבוכיות קבועה (באמצעות הפעולה IncreaseValue).
חישוב הסיבוכיות: $O(log\, m\ +\ 1)\ =\ O(log\, m)$
ולכן סה"כ סיבוכיות זמן הריצה: $O(log\, m)$ m - מספר החברות

void PromoteEmployee(int EmployeeID, int SalaryIncrease, int BumpGrade)
הפעולה מעדכנת את פרטי הדרגה והשכר של העובד שהמזהה שלו הוא EmployeeID.
במידה ואחד מהקלטים אינו תקין, נזרקת חריגה. הפעולה מחפשת בעץ של העובדים (הממויין לפי ID) אם העובד קיים באמצעות find (בסיבוכיות של log n). במידה והעובד לא נמצא נזרקת חריגה מתאימה. לאחר מכן אנחנו פונים לחברה (בסיבוכיות זמן קבוע כי העובד מצביע אליו) ומסירים אותו מהעץ הממויין לפי שכר (כדי שנעדכן אותו לאחר העדכון) ולאחר מכן אנחנו מעדכנים את פרטי העובד לפי התנאים המצויים בשאלה, מה שמתבצע באמצעות סיבוכיות קבועה (באמצעות הפעולות IncreaseValue & IncreaseBump).
מכניסים את העובד לעץ לפי ה- Salary של החברה - בסיבוכיות של log n במקרה הגרוע.
לאחר מכן מבצעים עדכון בחברה ובמבנה הכללי במידה ומדובר עכשיו בעובד המרוויח ביותר (מתבצעים בדיוק אותם פעולות הקשורות לעובד המרוויח ביותר כמו ב- AddEmployee הקורא בסיבוכיות של log n).
חישוב הסיבוכיות:
$$O(log\, n\ +\ log\, n\ +\ log\, n\ +\ logn\ +\ log\, n\ +\ 1\ +\ log\, n\ +\ 1)\ =\ O(log\, n)$$
ולכן סה"כ סיבוכיות זמן הריצה: $O(log\, n)$ מספר העובדים - n

void HireEmployee(int EmployeeID, int NewCompanyID)
הפעולה מעבירה את העובד שהמזהה שלו הוא EmployeeID, מהחברה שבה הוא נמצא לחברה שהמזהה שלה הוא NewCompanyId. במידה והעובד לא קיים או שלא קיימת חברה עם מזהה כזה, נזרקת חריגה מתאימה.
הפעולה מוחקת את העובד מהמבנה באמצעות RemoveEmployee - מה שמהתבצע בסיבוכיות של log n, ולאחר מכן משתמשת ב- AddEmployee לחברה הרצויה - מה שמתבצע בסיבוכיות של log n + log m.
חישוב הסיבוכיות: $O(log\, n\ +\ log\, n\ +\ log\, m)\ =\ O(log\, n\ +\ log\, m)$
לכן סה"כ סיבוכיות זמן ריצה: $O(log\, m\ +\ log\, n)$
מספר החברות - m
מספר העובדים - n

void AcquireCompany(int AcquireID, int TargetID, double Factor)
החברה AcquireID, מנסה לרכוש את החברה TargetID. במידה ולא קיימים חברות כאלה או שהחברה הרוכשת לא מקיימת את התנאי - נזרקת חריגה מתאימה. מציאת חברות לוקח log m (ראינו כבר).
הפעולה לוקחת את העובדים של שתי החברות ומבצעת ביניהם מיזוג (באמצעות הפעולה merge שהסברנו עליה, המבצעת מיזוג בזמן לינארי התלוי בכמות של הצמתים בשני העצים) - המיזוג מתבצע על 2 העצים הרלוונטים. לאחר מכן אנחנו מעדכנים את ערך החברה החדש - מבתצע בזמן קבוע ולאחר מכן אנחנו מסירים את החברה שנקנתה מעץ החברות, מה שלוקח log m (באמעצות RemoveCompany).
לאחר מכן אנחנו עוברים על כל העובדים בחברה החדשה בזמן ריצה לינארי, ומעדכנים לכל אחד את החברה ולבסוף גם מעדכנים את העץ של best_earning_employees - בסיבוכיות log m.
ולבסוף משחררים את הזיכרון שהוקצה עבור החברה.
חישוב הסיבוכיות: $O(2log\, m\ +\ 2log\, m\ +\ 2(n_{acquire}\ +\ n_{target}))\ =\ O(log\, m\ +n_{acquire}\ +\ n_{target})$
לכן סה"כ סיבוכיות זמן ריצה: $O(log\, m\ +n_{acquire}\ +\ n_{target})$
מספר החברות - m
מספר העובדים בחברה הרוכשת - $n_{acquire}$
מספר העובדים בחברה הנרכשת - $n_{target}$

void GetHighestEarner(int CompanyID, int *EmployeeID)
אם CompanyID > 0, הפעולה מחזירה את העובד המרוויח ביותר בחברה הנ"ל - log m, למצוא את החברה לוקח log m ולהוציא את העובד המרוויח ביותר ממנו - קבוע (memeber).
אם CompanyID < 0, הפעולה מחזירה את העובד המרוויח ביותר מכלל החברות - זמן קבוע (member).
במידה ואחד הקלטים אינו תקין (לדוגמה, אין חברה עם מזהה CompanyID) - נזרקת חריגה מתאימה.
חישוב הסיבוכיות (במקרה הראשון): $O(log\, m\ +\ 1)\ =\ O(log\, m\ )$
חישוב הסיבוכיות (במקרה השני): $O(1)\ =\ O(1\ )$
לכן סה"כ סיבוכיות זמן ריצה: $O(log\, m)$ עבור CompanyID > 0
מספר החברות - m
ובמקרה השני: $O(1)$ עבור CompanyID < 0

void GetAllEmployeesBySalary(int companyID, int **Employees, int *NumOfEmployees)

הפעולה מחזירה מערך המכיל את המזהים של העובדים לפי הסדר, מהמרוויח ביותר לפחות, בחברה שהמזהה שלה הוא CompanyID, ומחזירה את הכמות הנ"ל גם בהתאם.

אם CompanyID שלילי אז מחזירים אותו דבר רק עבור סך כל העובדים במערכת.

במידה ואחד הקלטים אינו תקין - מוחזרת חריגה בהתאם.

עבור CompanyID > 0: אנחנו תחילה מחפשים את החברה CompanyID - בסיבוכיות של log m (כפי שהסברנו).

במידה והחברה לא קיימת - נזרקת חריגה בהתאם.

הפעולה משתמשת בפעולה GetFirstNum על העץ של העובדים בחברה אשר מחזירה מערך של העובדים בעץ - העובד בסיבוכיות של $n_{company}$ כאשר $n_{company}$ הוא כמות העובדים בחברה.

הערך של $n_{company}$ שמור לנו בתכונות של המחלקה Company ולכן מוחזר בזמן קבוע.

עבור CompanyID < 0: הפעולה משתמשת בפעולה GetFirstNum על העץ של כלל העובדים אשר מחזירה מערך של כל העובדים בעץ הנ"ל - העובד בסיבוכיות של $n$ כאשר $n$ הוא כמות העובדים הכולל מכל החברות יחד.

הערך של $n$ שמור לנו בתכונות של המחלקה HighTech ולכן מוחזר בזמן קבוע.

חישוב הסיבוכיות (במקרה הראשון): $O(log\ m\ +\ n_{Company}\ +\ 1)\ =\ O(log\ m\ +\ n_{company})$

חישוב הסיבוכיות (במקרה השני): $O(n\ +\ 1)\ =\ O(n)$

לכן סה"כ סיבוכיות זמן ריצה: $O(log\ m\ +\ n_{company})$ עבור CompanyID > 0

m - מספר החברות ו- $n_{company}$ הוסבר מעלה.

ובמקרה השני: $O(n)$ עבור CompanyID < 0

n - מספר העובדים הכולל

void GetHighestEarnerInEachCompany(int NumOfCompanies, int **Employees)

הפעולה מחזירה מערך המכיל את המזהים של העובדים לפי הסדר, מהמרוויח ביותר לפחות, מכל חברה. במידה ואחד הקלטים אינו תקין - מוחזרת חריגה בהתאם.

הפעולה משתמשת בפעולה GetFirstNum בהתאם לערך של NumOfCompanies, ולכן סיבוכיות הפעולה הינה NumOfCompanies (הראנו זאת) המכיל את NumOfCompanies העובדים המרוויחים ביותר מכל חברה.

בעץ הנ"ל יש לכל היותר ערך השווה למינימום מבין m ו- n (הוסבר כבר) צמתים.

הפעולה מקצה מערך חדש בגודל המתאים - זמן ריצה קבוע ולאחר מכן עוברת על המערך שקיבלנו ומוציאה ממנו את ה- ID של כל עובד בעץ זה - סיבוכיות זמן של NumOfCompanies - כי זהו כמות העובדים שביקשנו (ולכן גם שווה לגודל המערך).

ולבסוף אנחנו משחררים את מערך העזר שהקצנו.

חישוב הסיבוכיות: $O(1\ +\ NumOfCompanies\ +\ 1)\ =\ O(NumOfCompanies)$

לכן סה"כ סיבוכיות זמן ריצה: $O(NumOfCompanies)$

$NumOfCompanies$ - מספר העובדים הרצוי

- פתרנו סעיף זה בסיבוכיות נמוכה יותר מהסיבוכיות שביקשו בשאלה.

void GetNumEmployeesMatching(int CompanyID, int MinEmployeeID, int MaxEmployeeID, int MinSalary, int MinGrade,
int *TotalNumOfEmployees, int *NumOfEmployees)

הפעולה מחזירה את מספר העובדים אשר המזהה שלהם בטווח בין min לmax ועומדים בתנאי השכר והדרגה. אילו ניתן מזהה חברה תקין נמצא את החברה בעץ החברות $O(logk)$

תחילה נשתמש על מפת העובדים של החברה בפעולה GetObjectFromKey אשר רצה בזמן $O(n_{company} + TotalNumOfEmployees)$ ומחזירה מערך של העובדים אשר בטווח

כעת נרוץ על המערך ונספור את כמות העובדים אשר עומדים בתנאי MinSalary וMinGrade

ריצה על מערך היא בסיבוכיות זמן של $O(TotalNumOfEmployees)$

ולכן סך הכל פעולה זו בעלת סיבוכיות זמן של: $O(logk\ +\ n_{company}\ +\ TotalNumOfEmployees)$

אילו מזהה החברה לא תקין נדלג על השלב הראשון ובמקום להחזיר את עובדי החברה נחזיר מערך של כל העובדים ולכן סיבוכיות הזמן שלה הוא: $O\ logn\ +\ TotalNumOfEmployees)$

במקרה של קלט לא תקין אחר הפעולה מחזירה שגיאה מתאימה