**רטוב 2 – הסבר ופירוט עבודה**

הסבר על מבני הנתונים הבסיסיים שהשתמשנו בהם:

Generic Union Find:

השימוש העיקרי באלגוריתם זה הוא ניצול העובדה שלפי המימוש הרביעי שראינו בהרצאה (איחוד לפי גודל וכיווץ מסלולים) הסיבוכיות המשוערכת לכל פעולה היא , מה שמאפשר לנו לקיים את דרישת סיבוכיות הזמן ברוב הפונקציות למשל בmerge נשתמש בUnion ונבצע זאת ב- או כשרוצים לחפש קבוצה מסוימת אז משתמשים בFind.

Hash Table:

טבלת ערבול דינאמית של כל השחקנים במערכת, שמכילה אובייקטים מסוג עובד (EmployeeByGrade/EmployeeBySalary תחת Employee.h). טבלת הערבול מאותחלת עם גודל קבוע מראש וגדלה/קטנה בהתאם לעומס בטבלה כפי שלמדנו בתרגול והרצאות. טבלת הערבול של השחקנים מערבלת ע"פ מזהה העובד וממומשת בשיטת double hashing שאינה צריכה לדעת את סדר גודל הקלט כי היא משנה את גודלה דינמית כפי שלמדנו בתרגול .

AVL Ranked Tree:

הסתמכנו על המימוש של ה-AVL מהתרגיל הקודם וניצלנו את זה שהחיפוש נעשה בסיבוכיות , אבל בנוסף עכשיו יש לנו את הsalary שנכנס לתמונה והשימוש בgrade עוזר לנו במימוש. כלומר הוספנו לעץ מידע נוסף שהוא סכום האיברים בתת העץ של הצומת הנוכחי, ובנוסף לכל צומת יש את ה-Rank שלו כלומר דבר זה מאפשר לחשב את הפעולות הבאות ב-Log(n).

הערה: יש לנו שני עצים לכל חברה ובנוסף שני עצים גלובליים.

1. מציאת מספר האיברים בין שני צמתים בעץ לבחירתנו.
2. מציאת סכום של קבוצה של איברים בין שני צמתים בעץ לבחירתנו.

Counters:

1. בכל חברה החזקנו שלושה מונים, הראשון הוא מספר העובדים הכללי בחברה זו, השני הוא מספר העובדים בחברה זו בעלי שכר 0, והשלישי הוא סכום הדרגות של אותם עובדים בעל שכר 0. את שלושת המונים האלה אנחנו מעדכנים בפונקציות הרלוונטיות ב-O(1) זמן.
2. ישנם עוד שלושת מונים דומים בדיוק לאלה שנמצאים בתת סעיף 1 לעיל, אך הם מיועדים למבנה הכללי, כלומר עבור כל החברות ביחד.

המבנה העיקרי:

1. Union Find אחד של הקבוצות:

נעשה על מערך של מספרים מ-1 ועד k שמייצגים את החברות שנמצאות אצלנו במערכת, בעבודה זו היה מספיק לעבוד עם מספרים (המזהים של החברות) ללא צורך במצביעים להם כיוון שהחזקנו במערך גלובלי נוסף שיש בו מצביעים לכל החברות שנמצאות במערכת. כל חברה במערך החברות מכילה את המספר המזהה שלה, מצביעם לשני העצים של העובדים שממוינים לפי salary&employeeID, ובנוסף החזקנו בעוד מספר מונים שמשמשים לחישובים כפי שנפרט בהמשך בתיאור הפונקציות.

ב-UF שלנו החזקנו שני מערכים של מידע נוסף (sum & data), המערך sum מתאר בתא i את הערך של תת העץ ההפוך על פי הכללים של קניה ומכירה של חברה בעבודה שלנו, והמערך data מכיל מספרים שמהווים תיקונים שחייבים להוסיף לכל צומת על מנת שחישוב ערך של חברה בסוף יצא נכון.

חישוב ערך של חברה x בעזרת UF מתבצע בעזרת מעבר על כל הצמתים מ-x ועד השורש ולאורך כל המסלול נסכום data[i]+sum[i] כאשר i מתאר צומת במסלול בין x לשורש (כולל). הרעיון של המידע הנוסף דומה ל-UF שלמדנו בתרגול (שאלת הקוביות סעיף אחרון), וכמו כן גם התיקונים (מערך ה-DATA).

1. Hash Table אחד של השחקנים:

טבלת ערבול דינמית שמכילה את כל האובייקטים של העובדים (כל אובייקט של עובד מכיל את המזהה של העובד, המזהה של החברה שהוא שייך לה, המשכורת והדרגה). מה שמאפשר לנו למצוא אם השחקן נמצא בשוק או לא במימוש הפונקציות (ב-O(1) זמן).

1. עצי AVL Ranked Table:

כל אחד מהעצים הבאים נמצא גם במבנה הכללי של השוק וגם בתוך טיפוס הקבוצה Company.

**\*\* כל הוספה/הסרה של אחד הצמתים בעצים אלה נעשית בסיבוכיות של כאשר n הוא מספר השחקנים בכל המשחק \*\***

1. **העץ הראשון:**

העץ הראשון מכיל את כל העובדים שה-salary שלהם הוא **לא** אפס, כלומר עבור כל עובד שנמצא בעץ המשכורת שלו אינה 0 והיא בהכרח חיובית. העץ הזה ממוין לפי קודם כל משכורת (salary) ולאחר מכן מיון משני לפי מזהה העובד (employeeID). בעזרת המיון הזה ניתן לחשב בסיבוכיות זמן הדרושה את הפונקציות שבתרגיל.

בכל צומת ה-KEY הוא העובד שלנו (טיפוס employee), וה-DATA הוא ה-grade של אותו עובד.

1. **העץ השני:**

מכיל את כל העובדים שה-salary שלהם הוא **לא** אפס, כלומר עבור כל עובד שנמצא בעץ המשכורת שלו אינה 0 והיא בהכרח חיובית. העץ הזה ממוין לפי קודם כל משכורת (salary) ולאחר מכן מיון משני לפי מזהה העובד (employeeID). **אך ההבדל בין עץ זה לעץ הראשון הוא ה-DATA והמידע הנוסף ששומרים בו**. בעץ זה, בכל צומת המפתח הוא העובד (טיפוס employee), וה-DATA היא פשוט המספר 1 (טיפוס int), מימוש זה מאפשר לנו לחשב את מספר העובדים בטווח [lowSalary, highSalary] בזמן O(log(n)) לעומת זאת העץ הראשון מאפשר לנו לדעת את סכום הדרגות של כל העובדים שנמצאים בין [lowSalary, highSalary].

**נפרט עבור כל פונקציה את הסיבוכיות של המימוש שלה :**

**\*\* כל הבדיקות של תקינות הקלט נעשית בסיבוכיות לכן לא נזכיר אותם בניתוח הסיבוכיות בפונקציות, כנ"ל לגבי עדכון מצביעים ואובייקטים בפונקציות \*\***

1. void\* init(int k):

אתחול של מצביע למבנה ריק שכולל עצים שמותחלים ב-O(1), hash ומערך של מצביעים לקבוצות לאלגוריתם ה-Union Find (בזמן O(k)).

אתחול של העצים וה-hash הוא בסיבוכיות של , מספר החברות הינו נתון וידוע k לכן בסה"כ האתחול הוא בסיבוכיות .

1. StatusType acquireCompany(void \*DS, int AcquirerID, int TargerID, double Factor):

מוצאים את החברות בפעולת Find לפי המזהה שלהן O(log\*k), ואז משתמשים בפעולת Union של הUnion Find כדי לאחד את שתי החברות O(1), שתי הפעולות האלה במימוש שלנו הן בסיבוכיות משוערכת של .   
לאחר שמצאנו את שתי החברות, לפי המימוש שלנו, יש בכל חברה שני עצים המכילים את כל העובדים בעל שכר חיובי, ולכן אנחנו חייבים להעביר את העובדים מעץ לעץ אך כפי שלמדנו בכיתה ניתן לבצע זאת ב-O(n1+n2) זמן כאשר n1 זהו מספר האיברים בעץ הראשון ו-n2 זהו מספר האיברים בעץ השני (בעצם ניתן לבצע זאת ע"י שני סיורים של inOrder על כל עץ בנפרד (O(n1+n2)) ולאחר מכן נוכל לבנות עץ כמעט שלם בעל מספר איברים מתאים ומספיק (n1+n2). לבסוף נפעיל אלגוריתם merge למיזוג שני מערכים בגודל n1+n2 בסיבוכיות O(n1+n2) וסיור inOrder אחרון על העץ כדי להחזיר את על האיברים לעץ – כלומר סה"כ קיבלנו שניתן למזג שני עצים בסיבוכיות ליניארית במספר הצמתים בשני העצים), את הפעולה הזאת נבצע על שני העצים שנמצאים בחברה הנרכשת ומעבירים את המידע לחברה הרוכשת.

המידע האחרון שנרצה לעדכן הוא המונים (counters) שיש לנו בהתחלה, נעדכן אותם בהתאם לכללים בעבודה, כלומר נעדכן את 3 המונים בחברה שרכשה ב-O(1) זמן.

סה"כ משוערך.

1. StatusType addEmployee(void \*DS, int EmployeeID, int CompanyID, int grade):

מוסיפים את העובד לטבלת הערבול של העובדים בסיבוכיות , לא מוסיפים את העובד לעצים בשלב הזה כדי לקיים את הסיבוכיות וכי העצים רק מכילים את העובדים בעלי שכר גדול ממש מ-0, אבל מעדכנים לחברה שהוא ישתייך אליה את ה-counter-ים ששמורים לנו ומתחזקים את העובדים בעל שכר 0. באותה דרך גם נעדכן את המונים הגלובליים.

לצורך עדכון ה-counter הזה מוצאים את החברה לפי פעולת Find בUnion Find בסיבוכיות של וזו היא הסיבוכיות הכוללת של הפונקציה כנדרש, כי העדכון של המונים נעשה ב-O(1) ולכן סה"כ O(1) זמן.

1. StatusType removeEmployee(void \*DS, int EmployeeID):

קודם נמצא את העובד לפי המזהה שלו בHash Table של העובדים בסיבוכיות בממוצע על הקלט, ואז מהעובד מקבלים את מזהה של החברה שהוא שייך לה (את החברה שהוא שייך לה אחרי כל הרכישות ולא את החברה המקורית שהוא היה שייך לה בהתחלה). מוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות משוערך, ובסוף מוצאים את העובד ומוחקים אותו מהעצים בסיבוכיות של **במקרה והשכר שלו הוא גדול ממש מאפס**, או פשוט מעדכנים את המונים שלנו בחברה וגם במבנה הכללי כלומר מחסירים את מספר העובדים באחד ומקטינים את סכום הדרגות בדרגה שלו.

בסה"כ מקבלים סיבוכיות של משוערך ובממוצע על הקלט במקרה שהעובד נמצא באיזשהו עץ. אחרת O(log\* k).

כלומר סה"כ O(log\*k + log n).

1. StatusType promoteEmployee(void \*DS, int EmployeeID, int BumpGrade):

קודם נמצא את העובד לפי המזהה שלו בHash Table של העובדים בסיבוכיות , ואז מהעובד מקבלים את מזהה של החברה שהוא שייך לה - (את החברה שהוא שייך לה אחרי כל הרכישות ולא את החברה המקורית שהוא היה שייך לה בהתחלה) - מוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות , ומכאן נחלק לשני מקרים:

1. העובד בעל EmployeeID יש לו שכר גדול מאפס:

במקרה הזה נמחק את העובד משני העצים הקיימים ונוסיף אותו בחזרה עם הדגרה המעודכנת. סיבוכיות זמן O(log n).

1. העובד בעל EmployeeID יש לו שכר ששווה לאפס:

במקרה זה מעדכנים רק את המונה ((counter שאחראי על סכום הדרגות של העובדים בעלי שכר אפס גם בחברה עצמה וגם במבנה הכללי.

סה"כ כנדרש.

1. StatusType employeeSalaryIncrease(void \*DS, int EmployeeID, int SalaryIncrease):

קודם נמצא את העובד לפי המזהה שלו בHash Table של העובדים בסיבוכיות , ואז מהעובד מקבלים את מזהה של החברה שהוא שייך לה ומוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות .

בדומה לקודם עתה נחלק למקרים:

1. השכר הקודם של העובד היה 0:

במקרה זה נעדכן את המונים גם בחברה עצמה וגם במבנה הכללי (נפחית את מספר העובדים בעלי שכר 0 ב-1 ונפחית את הדרגה שלו מסכום הדרגות של אותה חברה).

אותם שינויים נבצע למונים במבנה הכללי.

נוסיף את העובד עם השינוי החדש לשני העצים שנמצאים בחברה וגם לשני העצים שנמצאים במבנה הכללי. O(log n).

1. השכר הקודם של העובד לא היה 0:

מוחקים/מסירים את העובד מ-4 העצים שיש לנו במערכת (2 עצים לאותה חברה ששייך אליה העובד ועוד שני עצים במבנה הכללי), ובסוף מכניסים אותו בחזרה לכל 4 העצים עם השינוי הנדרש (שינוי בשכר). הסיבוכיות היא .

בסה"כ מקבלים סיבוכיות של .

1. StatusType sumOfBumpGradeBetweenTopWorkersByGroup (void \*DS, int CompanyID, int m)

נפצל למקרים: CompanyID = 0 או CompanyID != 0.

אך בשני המקרים החישוב דומה, אבל מבוצע בעצים שונים, כאשר מזהה החברה 0 נשתמש בעצים הגלובליים, וכאשר המזהה אינו 0 נשתמש בעצמים השמורים בחברה.

קודם כל נשתמש בעץ השני בכדי למצוא את העובד n-m בגודלו בעץ (כאשר n הוא מספר האיברים בעץ), לאחר מכן נחפש את אותו עובד בעץ הראשון, ולפי המידע הנוסף בעץ הראשון (עבור כל צומת – סכום הצמתים בתת העץ שלו – סכום הדרגות/grades בתת העץ שלו) נוכל לסכום רק את הצמתים/העובדים בעלי שכר גדולים שווים לו (אלה שנמצאים מימינו בעץ). החישוב יחזיר את התוצאה הרצויה, ובנוסף החישוב מבוצע אך ורק ע"י המידע שנמצא במסלול החישוב ואולי עבור כל צומת במסלול החישוב צומת אחד מימינו/שמאלה כלומר מספר הצמתים שאנחנו עוברים עליהם בחישוב הוא O(log n) כאשר n הוא מספר העובדים בעץ.

סה"כ O(log\* k + log n).

1. StatusType averageBumpGradeBetweenSalaryByGroup (void \*DS, int CompanyID, int lowerSalary, int higherSalary)

נפצל למקרים: CompanyID = 0 או CompanyID != 0.

אך בשני המקרים החישוב דומה, אבל מבוצע בעצים שונים, כאשר מזהה החברה 0 נשתמש בעצים הגלובליים, וכאשר המזהה אינו 0 נשתמש בעצמים השמורים בחברה.

בפונקציה הזו, נשתמש בשני העצים ובנוסך במונים ששמרנו ותחזקנו לאורך כל המבנה.

מחפשים את הצומת/עובד הקטן ביותר בעל lowerSalary וכמו כן מחפשים את העובד הגדול ביותר בעל higherSalary, לאחר מכן נמצא את האב המשותף שלהם בעץ, ונשתמש במידע הנוסף שהוא סכום האיברים בתת העץ בכדי לחשב את סכום הדרגות של העובדים בעלי lowerSalary ו-higherSalary, את השיטה הזו למדנו בתרגולים ולכן לא נפרט כאן את החישוב בדיוק, אך בתיאור קצר, כאשר עולים מהבן השמאלי ועד האב המשותף נסכום את כל האיברים מימין, וכאשר עולים מהצומת הימני לאב המשותף נסכום את כל האיברים משמאלו, נדאג לכך לא לסכום את אותו ערך פעמיים).

עתה אנחנו יודעים את סכום הדרגות של העובדים בעלי שכר גדול מאפס נסמנו sumGradesSalaryNonZero. עתה באמצעות העץ השני נחשב את מספר האיברים/עובדים בין lowerSalary ו-higherSalary בשיטה דומה מאוד לזו לעיל.

נסמן ערך זה ב-numEmployeesSalaryNonZero.

בנוסף נסמן numEmployeesSalaryZero בעובדים בעלי שכר 0 בחברה.

sumGradesSalaryZero סכום הדרגות לעובדים בעלי שכר 0 בחברה.

נחזיר את הערך הרצוי בהתאם ל-lowerSalary:

אם הוא אפס נחזיר:

אחרת נחזיר: *.*

*סיבוכיות זמן: כיוון ששמרנו את המידע הנוסף בשני העצים, כל החישובים בעצים מבוצעים אך ורק במסלול החיפוש, כפי שלמדנו בהרצאה / תרגולים ולכן סיבוכיות חישובים אלה היא O(log n).*

*בנוסף חישוב עם ה-מונים הוא ב-O(1) ולכן סה"כ הסיבוכיות היא:*

*O(log\* k + log n).*

1. StatusType companyValue(void \*DS, int companyID)

בעזרת המידע הנוסף ששמרנו ב-UF, אנחנו יכולים לחשב ערך של חברה פשוט ע"י קריאה ל-find וסכום הערכים לאורך המסלול בפעולת find יניב את הערך הנכון של החברה (כפי שפורט בהתחלה רעיון זה דומה לתרגיל שהועבר בתרגולים) השתמשנו בשני מערכים sum ו- data.

1. void Quit(void \*\*DS):

מאחר והקצנו (k חברות וn עובדים) מקום, עלינו לעבור על כל המקום ולשחרר אותו. לכן סיבוכיות הזמן היא בהתאם .