תרגיל רטוב 2 – מבני נתונים

אופיר אלישיב 318306289

רובן טימסיט 330083858

תיאור כללי של מבני הנתונים

- **UserArray –** מחלקה שמכילה בתוכה את הHashTable
המחלקה תחזיק מערך של עצים, ותשתמש בChain Hashing, כך שבכל אינדקס של המערך שלנו יש עץ של משתמשים על מנת למנוע התנגשויות.
העצים שנשתמש בהם יהיו עצי הAVL כמו הmembersTree לנוחות.
כל פעם שהמערך מגיע לגודל המקסימלי שלו, ניצור מערך דינמי חדש ונעתיק את כל המידע לתוכו, נדאג גם שלא נפגע בחיפוש על ידי זה שנכניס את האיברים למערך הגדול יותר שיצרנו מחדש.

- **UnionFind –** נחזיק 2 מערכים של AVLNODES כאשר אחד מהם מייצג את האיברים ואחד מהם מייצג את הסטים, בנוסף לכך נחזיק מערך setsSize שיחזיק את הגודל של כל אחד מהסטים.
נממש כפי שנלמד בהרצאה ובתרגול ע"י עצים הפוכים ומערכים, וגם כיווץ מסלולים ואיחוד לפי גודל של סט על מנת לעמוד בסיבוכיות.

- **AVLTree –** עץ דומה לעץ בתרגיל בית 1, אך כעת הוספנו לכל Node שדה נוסף Extra כדי שנוכל לתמוך בפעולות גם עבור הפרסים בmembersTree וגם עבור getPlace ב UnionFind

- **User –** מחלקה של משתמש, השדות שלו זה id, phone, isMember, expenses, prizeOffset
כאשר יש לו מתודות מתאימות לביצוע עדכון השדות שלו, השדה prizeOffset ישמש כדי לחשב את getExpense בצורה נכונה, הסבר מורחב בפונקציה עצמה.

- **Record –** מחלקה שמייצגת כל תקליט, מכילה מידע על כמות העותקים מכל תקליט וכמה פעמים התקליט נקנה עבור חישובים בפונק' אחרות כמו buyRecord ולעדכן את שדה הexpenses של המשתמש.

- **recordsCompany –** המחלקה של חברת התקליטים, השדות של המחלקה הינם:

  - user_array – אובייקט מסוג UserArray שמייצג את הHashTable על מנת שנוכל לגשת למשתמשים ולבצע את כל הפעולות הנדרשות בסיבוכיות המתאימה, נאתחל אותו עם מערך בגודל 2 כשהsize שלו מתחיל ב0.
  - records – מערך של מצביעים לאובייקטים מסוג Record, הולך להמחק ולהיווצר מחדש כל פעם שמשתמשים בnewMonth, בהתחלה יהיה nullptr, ברגע שנקרא לnewMonth ניצור מערך חדש ונעשה השמה לrecords.
  - membersTree – עץ AVL כפי שהוסבר למעלה, כאשר לNODES יש שדה נוסף.
עם כל makeMember תקין נוסיף את המשתמש המתאים לעץ, וכשנעשה insert לעץ נחשב את הprizeOffset ונכניס אותו למשתמש.
  - uf – מצביע למבנה הנתונים UnionFind, מתחיל כnullptr וברגע שקוראים לnewMonth נאתחל אותו כפי שיוסבר בהמשך, ישמש אותנו כדי לתמוך בפעולות getPlace,putOnTop
  - recordsNum – שדה int שנוכל לדעת מה כמות התקליטים הנוכחית, כל פעם שעושים newMonth נעדכן את השדה הזה בהתאם.

1) newMonth

הפונקציה מקבלת מצביע למערך כמות עותקים של תקליטים וגודל

במידה וכמות התקליטים קטנה מ0 נחזיר שגיאת קלט.

במידה והפונק' כבר נקראה בעבר אז נהרוס את מבנה הUF הקודם וגם את מערך התקליטים הקודם, לאחר מכן ניצור UF ומערך תקליטים חדש עם הערכים החדשים שקלטנו.

נעבור על העץ של המנויים ונאפס את ההוצאות והפרסים שלהם.

Complexity	Action
$O(n)$	איפוס הוצאות ופרסים
$O(m)$	הריסת הUF
$O(m)$	יצירת UF חדש

$$\boldsymbol{Total = O(m+n)}$$

2) ~RecordsCompany

הפונקציה מוחקת את מבנה הUF הקיים, ועוברת על מערך התקליטים ומשחררת את המקום שהוקצה לכל אחד מהם, ואז מוחקת את המקום שהוקצה למערך.

משחררת גם את המקום שהוקצה עבור הUserArray כולל כל העצים שבתוכו

Complexity	Action
$O(m)$	הריסת הUF
$O(n)$	הריסת הUserArray

$$\boldsymbol{Total = O(m+n)}$$

3) addCostumer

הפונקציה מקבלת ID של משתמש
ראשית נבדוק האם הקלט תקין והאם לא קיים משתמש עם ID זהה לזה של החדש, בכל אחד מהמקרים האלה נחזיר שגיאה בהתאם.

ברגע שנעבור את הבדיקות נכניס את המשתמש לעץ המשתמשים

Complexity	Action
$O(logn)$	חיפוש המשתמש בעץ המשתמשים
$O(logn)$	הוספה לעץ המשתמשים

$$\boldsymbol{Total = O(logn)}$$

4) getPhone

ראשית נוודא שהקלט תקין לפי ההגדרות.

ניגש אל הuserArray ונוציא מתוך מתודה של מחלקת המשתמש את מספר הטלפון שלו.

Complexity	Action
ממוצע(1)	חיפוש המשתמש בuserArray

$$\boldsymbol{Total = O(1)}$$ **ממוצע**

makeMember (5

ראשית נוודא שהקלט תקין לפי ההגדרות.

ניגש לuserArray כדי לקבל את המשתמש, נכניס אותו אל עץ הmembersTree כאשר אנחנו מחשבים את הoffset שלו ומכניסים את זה לשדה prizeOffset של המשתמש(כלומר את הפרס שאמור להיות לו במידה והיה בעץ מההתחלה על מנת שנוכל לדעת מה הoffset מהפרס ברגע שנחשב אותו בgetExpense)

בנוסף לזה גם נגדיר את המשתמש להיות חבר מועדון.

Complexity	Action
$O(logn)$	הוספה לmembersTree
$O(1)$ ממוצע	חיפוש בuserArray

$$\boldsymbol{Total = O(logn)}$$

isMember (6

ראשית נוודא שהקלט תקין לפי ההגדרות.

נחפש את המשתמש בuserArray וניגש למתודה של הקלאס של הuser ונבדוק האם הוא member, נחזיר בהתאם.

ברגע שנעבור את הבדיקות נכניס את הקבוצה לעץ הקבוצות כאשר סטטוס הVIP שלה הוא false

Complexity	Action
$O(1)$ משוערך	גישה לuserArray

$$\boldsymbol{Total = O(1)}$$ **משוערך**

buyRecord (7

ראשית נוודא שהקלט תקין לפי ההגדרות.

ניגש למשתמש דרך הuserArray ונוסיף להוצאות שלו כפי שנתבקשנו כשאנחנו ניגשים לכמות הרכישות שבוצעו דרך מערך הrecords במקום המתאים לid

לאחר שנבצע זאת נוסיף לתקליט רכישה אחת.

Complexity	Action
$O(\log^* n)$	חיפוש בuserArray
$O(1)$	חיפוש התקליט במערך תקליטים
$O(1)$	עדכון שדה בתקליט

$$\boldsymbol{Total = O(log^* n) < O(log n)}$$

8) addPrize

ראשית נוודא שהקלט תקין לפי ההגדרות.

לאחר מכן ניגש למתודה addPrize של העץ MEMBERS כאשר פעם אחת אנחנו מוסיפים לשדה הEXTRA של כל NODE בהתאם לאיך שראינו בתרגול, עם שינוי קטן כך שאנחנו לא מוסיפים לNODE עצמו אלא רק למי שקטן מהNODE

נקרא למתודה פעם אחת membersTree.addPrize(c_id2,prize) וכך נוסיף פרס לכל מי שקטן מc_id2, ולאחר מכן נקרא למתודה פעם נוספת membersTree.addPrize(c_id1,-prize) וכך נוריד את הפרס שהוספנו לכל מי שקטן מc_id1 **לא כולל** הוא עצמו, מכיוון שאנחנו רוצים להוסיף גם לו את הפרס.
כל פעולת addPrize היא כמו Find בעץ חיפוש ולכן הסיבוכיות O(logn)

Complexity	Action
$O(2logn)$	פעמיים addPrize בעץ

$$\boldsymbol{Total = O(log n)}$$

9) getExpenses

ראשית נוודא שהקלט תקין לפי ההגדרות.

נוודא שהמשתמש אכן member ע"י כך שניגש לhashTable ב O(log*n)
נחשב את הPRIZE של המשתמש בעזרת מתודת FIND הרגילה בעץ members, כאשר הוספנו לו אופציה להחזיר את הסכום של הEXTRA של כל NODE עד שמגיעים לNODE הרלוונטי.

לבסוף נחזיר את סה"כ ההוצאות של המשתמש פחות סה"כ הפרס שלו, ונוסיף את הOFFSET שהוספנו למשתמש ברגע שהוספנו אותו לעץ מה שיתן לנו בדיוק את ההוצאות שלו פחות הפרסים שהוא קיבל כשהיה בעץ.

Complexity	Action
$O(\log^* n)$	חיפוש בhashTable
$O(logn)$	חיפוש בעץ members
$O(1)$	פעולות חישוב

$$\boldsymbol{Total = O(log n)}$$

10) putOnTop

ראשית נוודא שהקלט תקין לפי ההגדרות.

ניגש למתודת Union כאשר היא לוקחת בחשבון שצריך לשים את id1 על id2 לפי הסדר.
בתוך Union אנחנו מחפשים את השורש של כל אחד מהתקליטים(השורש האמיתי ולא השורש

שאנחנו אמורים לראות) ונפעל בהתאם למה שראינו בתרגול, כאשר אנחנו מעדכנים בהתאם את השדה EXTRA של כל אחד.
נדאג לייצוג שבו גודל הערמה של id1 גדל בגודל הערמה של id2 ונגדיר שכעת השורש שאנחנו אמורים לראות עבור id1 יהיה השורש שאנחנו אמורים לראות של id2(גם אם אינו האמיתי).

בתוך הפונקציה אנחנו משתמשים מספר פעמים במתודת Find של UF, אך פונקציה זו עומדת בסיבוכיות משוערכת O(log*n) מכיוון שמתבצע כיווץ מסלולים תוך כדי Find, ואז בFind הבא אנחנו עוברים לכל היותר NODE אחד בכל חיפוש.

כל שאר הפעולות מתבצעות בO(1)

Complexity	Action
$O(\log^* n)$	חיפוש השורש של כל id
$O(1)$	גישה לאיברים במערך וחישובים

$$\boldsymbol{Total = O(log^* n)}$$

<u>getPlace (11</u>

ראשית נוודא שהקלט תקין לפי ההגדרות.

ניגש למתודת getCol של UF שמחפשת את השורש האמיתי של הID ואז ניגש לשורש המלאכותי של השורש האמיתי במידה ואיננו הוא, סה"כ נבצע חיפוש עם מתודת FIND של הUF

Complexity	Action
$O(\log^* n)$	חיפוש בUF

$$\boldsymbol{Total = O(log^* n)}$$