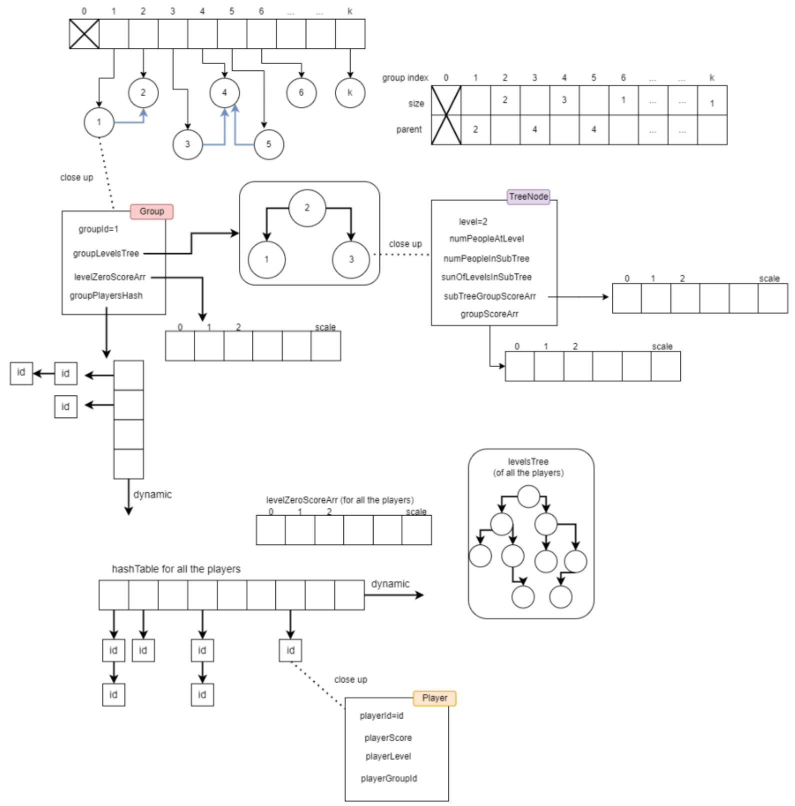
מבני- תרגיל רטוב שני- חלק יבש:

מגישות: אסתר (אתי) חיה רווח-318477387, עדי יוסף-318415684

תיאור הפתרון:

הערה: זה לא בא לידי ביטוי בציור, אבל במבנה נשמור את $scale$ ואת k (בשם $numOfGroups$)



טיפוסי נתונים בהם נשתמש:

- $Group$- טיפוס שיכיל: $groupId$ (את ה-id של הקבוצה), $groupLevelsTree$ (עץ דרגות AVL שכל איבר בו מטיפוס $TreeNode$ שיתואר בהמשך, עבור השלבים בהם נמצאים שחקני הקבוצה), $groupPlayersHash$ ($hashTable$ שבנוי בעזרת מערך דינמי שכל תא בו מכיל רשימה שאיבריה מטיפוס $Player$, שירכז את שחקני הקבוצה), $levelZeroScoreArr$ (מערך תוצאות שיפלג את כמות השחקנים בתוצאות כלשהן מתוך שחקני הקבוצה שבשלב 0 , כל תא יכיל את כמות השחקנים שבשלב 0 עם התוצאה $index$)
- $TreeNode$-טיפוס שיכיל: $level$ (מספר השלב), $nunPeopleAtLevel$ (כמות האנשים שבשלב הספציפי- אם עץ הדרגות בתוך הקבוצה, הכוונה לכמות האנשים מתוך הקבוצה שבשלב, אחרת מדובר בכמות האנשים שבשלב מתוך כלל שחקני המערכת), $numPeopleInSubTree$ (כמות האנשים בתת העץ), $sumOfLevelsInSubTree$ (סכום השלבים של כל השחקנים בתת העץ),
$groupScoreArr$ - מערך שמכיל בכל תא את את כמות השחקנים שבעלי התוצאה $index$ (אם מדובר בעץ השלבים של הקבוצה, מדובר במערך תוצאות של שחקני הקבוצה בלבד, אחרת מדובר במערך תוצאות של כלל השחקנים)- זהו מערך בגודל קבוע.
$subTreeGroupScoreArr$- מערך שמכיל בתא i את כמות השחקנים שבעלי תוצאה i מתוך כל השחקנים בתת העץ (אם זה עץ של קבוצה אז מתוך סך שחקני הקבוצה בתת העץ, אחרת מסך כל שחקני המערכת שבתת העץ)
[כל תא בעץ יכיל גם גובה, מצביע לאב, כמות $Node$ים בתת העץ]
- $Player$- טיפוס שיכיל: $playerId$ (ה-id של השחקן), $playerScore$ (תוצאת השחקן), $playerLevel$ (השלב של השחקן), $playerGroupId$ (ה-id של קבוצת השחקן)

המבנה שלנו יכלול:

- k קבוצות כאשר כל קבוצה מכילה שדות כמתואר בטיפוס $Group$
- מבנה $union-find$ שיורכב ממערך בגודל $k+1$ בו כל איבר מצביע לקבוצה המתאימה לו (המערך במקום ה-0 לא יצביע לכלום כי אינדקס של קבוצה הוא מ1 עד k), והעץ ההפוך שמתאים למבנה יתואר

בעזרת מערך דו מימדי של 2 שורות על $k+1$ עמודות כפי שתואר בהרצאה (דוגמה יש בתמונה לתיאור המבנה- בראש התמונה)

- עץ שלבים של כלל השחקנים במערכת, כאשר כל איבר בעץ הוא מטיפוס $TreeNode$ שתואר קודם לכן (עץ דרגות מסוג AVL) [כל איבר בעץ מצביע לאביו]
  הערה: מימשנו עץ דרגות אבל בסוף לא השתמשנו בפעולת ה-$select$... זה שזה עץ דרגות לא תרם בסופו של דבר לפתרון, אבל במימוש יש לנו עץ דרגות ולכן זה מצוין ביבש.
- $hashTable$ (מטיפוס שרשראות) לכלל השחקנים במערכת, שבא לידי ביטוי במערך דינמי שכל איבר בו מצביע לרשימה מטיפוס $Player$ שתואר קודם לכן (פונקציית הערבול תהיה:
  $index\ mod\ (size\_of\_array)$ שמתאימה כדי ליצור פקטור עומס של 1 כפי שראינו בתרגול ובהרצאה)
- $levelZeroScoreArr$- מערך שיפלג את כמות השחקנים שקיבלו תוצאות שמתאימות לאינדקסי המערך מתוך השחקנים שבשלב 0 במערכת

בנוסף, נשתמש בעץ שמימשנו בת"ב 1, נהפוך אותו שלא יהיה גנרי ונוסיף שדות לכל צומת: כמות שחקנים בתת עץ, סכום שלבים בתת העץ, כמות שחקנים בשלב, מערך תוצאות לשחקנים בשלב, מערך תוצאות לשחקנים בתת העץ וכמות צמתים בתת העץ (בנוסף לשדה גובה שהיה לנו). ההכנסה וההוצאה מהעץ נעשית בצורה רקורסיבית ולכן נעדכן את השדות באיברי העץ בצורה רקורסיבית, נגיע לאיבר ברמה התחתונה, ונעדכן מלמטה עד השורש. כאשר נצטרך לעשות גלגול נעדכן את המידע של הצומת שעובר גלגול ואז במקרה הצורך גם את אביו.

למשל בגלגול $LeftLeft$ נעדכן את המידע ב-$TreeNode$ עבור כמות האנשים בתת העץ כך:

$$node \rightarrow numPeopleAtSubTree = node \rightarrow right \rightarrow numPeopleAtSubTree + node \rightarrow left \rightarrow numPeopleAtSubTree + node \rightarrow numPeopleAtLevel$$

(אם אין בן ימני או שמאלי יחובר 0, כמות השחקנים בשלב לא משתנה עקב גלגול) ועבור, למשל, מערך התוצאות של תת העץ, נעדכן כך: נעבור בלולאה מ-$i=0$ עד $scale+1$ ונעדכן:

$$node \rightarrow subTreeScoreArr[i] = node \rightarrow right \rightarrow subTreeScoreArr[i] + node \rightarrow left \rightarrow subTreeScoreArr[i] + node \rightarrow scoreArr[i]$$

(אם אין בן ימני או שמאלי נחבר 0, כמות $node \rightarrow scoreArr[i]$ רלוונטית ספציפית לשלב ולא משתנה בגלגולים).
כלומר בגלגולים נעדכן רק שדות שקשורים לתת העץ (סכום דרגות בתת עץ, מערך תוצאות בתת עץ,...)
ועבור כל פעולה, אם $node \rightarrow father! = NULL$ נחזור על העדכונים גם עבורו (וכך נסיים את הגלגול עם מידע מעודכן) . כפי שניתן לראות יש לנו גישה למידע במצביעים ולכן גלגול נעשה ב-$O(1)$ [מפאת מקום לא הוסברו כל הפעולות שיש לבצע בגלגול, למשל תיקון סכום דרגות, אך זה נעשה בצורה דומה]

הסבר לפעולות:

הערה: בסוף כל פונקציה כתובה הסיבוכיות המשוערכת, זו כיוון שבהינתן m פעולות, ייתכן שבכולן נצטרך למצוא את הקבוצה בעזרת k פעולות $union-find$, נראה שכל פעולה בפונקציית חסומה ב-$O(x)$ כלשהו (למשל עדכון שלב לשחקן יהיה $(O(logn))$, ולכן m פעולות כאשר הפעולה **היקרה ביותר** על השחקנים היא $O(x)$, יעלו $O(m\cdot x+m\cdot$ $\log^* k)$ ולכן באופן משוערך $O(x+\log^* k)$, (שעושים פעולות על ה-$hashTable$ הפעולות הן בממוצע על הקלט ולכן הסיבוכיות תהיה משוערכת בממוצע על הקלט)

למשל כשעושים m פעולות ואחת מהן כוללת $mergeGroups$, עוד נראה שפעולה יקרה באיחוד ה-$hashTable$ של השחקנים עולה $O(n)$, ולכן בהינתן m פעולות כמו עדכון שלב, עדכון תוצאה לשחקן, קבלת אחוז השחקנים בתחום ועוד פונקציות שנספרות תחת הסיבוכיות המשוערכת לחלק שנוגע לקבוצות (כפי שמפורט בתרגיל תחת הנחיות לניתוח סיבוכיות), נקבל שהסיבוכיות המשוערכת היא $O(\log^* k+n)$ משוערכת בממוצע על הקלט, כי הפעולה היקרה ביותר מתרחשת באיחוד הקבוצות והיא החסם שלנו. מפאת חוסר מקום פירטנו את הרעיון הכללי ולא תחת כל פונקציה, אבל זה אופן חישוב זהה לכפי שראינו בהרצאה.

$void *\ init(int\ k, int\ scale)$:

נבדוק: אם $k\le 0$ או $scale \le 0$ או $scale > 200$ זו שגיאה ונחזיר $null$, אחרת נבצע:

- אתחול עץ דרגות ריק $levelsTree$ לכלל השחקנים בסיבוכיות $O(1)$
- שמירת $scale$ ו-k במבנה- $O(1)$
- אתחול $hashTable$ לכלל השחקנים, תחילה ע"י מערך בגודל קבוע שכל תא בו ריק-$O(1)$
- אתחול k קבוצות ריקות: בכל קבוצה יש אתחול של $hashTable$ שבהתחלה בגודל קבוע-$O(1)$, אתחול עץ AVL ריק לשלבים של שחקני הקבוצה-$O(1)$, אתחול מערך $levelZeroScoreArr$ בגודל קבוע-$O(1)$, והשמה של id למשתנה $groupId$-$O(1)$. מקצים מקום ל-k קבוצות ולכן הסיבוכיות הכוללת היא $O(k)$ [יצירת הקבוצות תתבצע בשניצור מבנה $union-find$]
- אתחול מבנה $union-find$ כפי שראינו בהרצאה. מקצים שני מערכים שתלויים בגודל של הקבוצות ולכן הסיבוכיות היא $O(k)$, מערך אחד ישמש כ"עץ הפוך" ומערך שני יצביע לקבוצות הריקות שהכנו

אם הקצאות המקום הצליחו נחזיר מצביע למבנה הנתונים, אחרת נהרוס את המבנה החלקי שנוצר ונחזיר $null$. סה"כ סיבוכיות $O(k)$

<u>**הערה:**</u> <u>במהלך</u> כל אחת מהפונקציות הבאות, לאחר בדיקת התקינות:

אם במהלך הפונקציה יש בעיה בהקצאת זיכרון, נחזיר $ALLOCATION\_ERROR$, אחרת (במידה ואין שגיאת $FAILURE$ להחזרה) נחזיר בסופה $SUCCESS$.

:StatusType mergeGroups(void $*$ DS, int GroupID1, int GroupID2)

נבדוק: אם $DS == NULL$ או $GroupId1 \leq 0$ או $GroupId1 > k$ או $GroupId2 \leq 0$ או $GroupId2 > k$ אז נחזיר $INVALID\_INPUT$, אחרת נבצע:

- נמצא את 2 הקבוצות ונאחד לפי גודל בסיבוכיות $O(\log^* k)$ באופן משוערך – כפי שראינו בהרצאה
- עבור עצי השלבים של הקבוצות, נרצה לאחד את העצים כך שיהיו בקבוצה הגדולה (אליה איחדנו) ויכילו מידע לגבי שחקני 2 הקבוצות, לכן נבצע:

1. נקרא לפונקציה שהכנו בעץ שיוצרת עץ חדש מ-2 עצים קיימים בעזרת איטרטור שהגדרנו בעץ שפועל לפי סיור $inorder$, ונכניס את איברי 2 העצים למערך ממויין (נוצר עותק). (נשתמש ב-2 איטרטורים ותנאים להשוואת ונכניס לפי סדר את איברי שני העצים מהקטן לגדול) , כשנעשה זאת: אם יש 2 שלבים זהים ב-$levelId$, נמזג אותם- נמזג את המערכים ואת ה-$hashTable$ שלהם, את גודלי כמות השחקנים בהם וכדומה, לכדי יצירת איבר חדש שמכיל את המידע של שני השלבים הזהים. אם n_{g1} זה כמות השחקנים של הקבוצה הקטנה ו-n_{g2} זה כמות השחקנים של הקבוצה האחרת, כך ש-$n_{g_1} + n_{g_2} = n$, נקבל שהסיבוכיות לפעולה זאת היא $O(n)$. הקצאנו מערך כגודל כמות האיברים בשני העצים ולכן נשמור משתנה שישמור לנו את גודל המערך האמיתי (כי ייתכן שחלקו יהיה ריק אם איחדנו שלבים)- $O(1)$
2. נרוקן את שני העצים את הקבוצות – סה"כ $O(n_{g_1}) + O(n_{g_2}) = O(n)$
3. נקרא לפונקציה שממלאת עץ לפי מערך וגודלו, וכך נמלא מחדש את העץ של הקבוצה הגדולה- נעשה זאת בעזרת האלגוריתם שראינו בת"ב רטוב 1, שהיה כך:

- קבע: השורש יהיה האיבר האמצעי המערך
- באופן רקורסיבי תעשה לבן הימני והשמאלי:

1) השג את האמצע של חצי המערך השמאלי והפוך אותו לבן השמאלי של השורש שנקבע בפעולה 1.
2) השג את האמצע של חצי המערך הימני והפוך אותו לבן הימני של השורש שנקבע בשלב 1.

-תהיה חזרה לשלב 1 בו בן הוא "שורש" בתת עץ

ניתן לתאר את הסיבוכיות באופן הבא: $T(n) = 2T\left(\frac{n}{2}\right) + C$

כאשר $T(n)$- הזמן למילוי עץ על ידי **מערך ממויין** בגודל n, ו-c קבוע (הזמן שלוקח למציאת האמצע במערך וכדומה). משיטת המאסטר <u>לפי השיטה הראשונה</u> ועבור בחירת $\epsilon = 1 > 0$ נקבל שסיבוכיות הפעולה היא $O(n)$ (c קבוע ולכן $f(n) = O(1)$)

הערה: במהלך הפונקציה ייתכן שיהיו גלגולים ועדכוני שדות בצמתי העץ, אנחנו מכניסים ומעדכנים רקורסיבית ולכן עומדים בסיבוכיות. וכמובן, נשחרר את כל ההעתקים שיצרנו במהלך הפונקציה.

- כעת נאחד בין 2 ה-$hashTable$ של 2 הקבוצות באופן הבא:

נעבור על המערך הדינמי שהוא ה-$hashTable$ של הקבוצה הקטנה, בכל תא יש רשימה של השחקנים שנמצאים בתא הנוכחי (אם קיימים), נעבור על איברי הרשימה ועבור כל שחקן ברשימה, נבצע פעולת $insert$ אל ה-$hashTable$ של הקבוצה הגדולה (נמצא בעזרת פונקציית הערבול את התא המתאים לשחקן במערך, ונוסיף **לראש הרשימה** שכרגע יש בו). נשים לב שייתכן שחלק מפעולות ההכנסה יגרמו להגדרת המערך הדינמי ב-$hashTable$ של הקבוצה הגדולה.

לבסוף נרוקן את ה-$hashTable$ של הקבוצה הקטנה. כשנסיים נעבור על ה-$hashTable$ של הקבוצה הגדולה ונעדכן את ה-$groupId$ של כל השחקנים בה לזה של הקבוצה הגדולה.

אם נסמן ב-n_1 את כמות השחקנים בקבוצה הראשונה, וב-n_2 את כמות השחקנים בקבוצה השנייה, כך ש-$n_1 + n_2 = n$ כאשר n סך השחקנים בשתי הקבוצות, נקבל:

המעבר על המערך הדינמי של הקבוצה הקטנה עולה $O(n_1)$ כיוון שגודל המערך הוא גם $O(n_1)$ ויש n_1 אנשים לעבור עליהם ברשימות, פעולת ההכנסה אל ה-$hashTable$ של הקבוצה השנייה עולה $O(1)$ באופן משוערך בממוצע על הקלט כפי שראינו בהרצאה, וריקון ה-$hashTable$ של הקבוצה הקטנה עולה $O(n_1)$.

המערך החדש של הקבוצה הגדולה הוא בסדר גודל $O(n)$, ולכן הסיבוכיות לעדכן את ה-$groupId$ של כל השחקנים בו תהיה $O(n)$

(בעזרת שימוש במצביעים חכמים לא נצטרך לעדכן גם ב-$hashTable$ של כלל השחקנים, כי 2 הטבלאות יצביעו לאותו השחקן) . לכן סה"כ הסיבוכיות היא $O(n)$ משוערך בממוצע על הקלט.

- נאחד בין מערכי $levelZeroScoreArr$ של הקבוצות, נעבור על המערך בקבוצה הקטנה ונעדכן כל תא במערך של הקבוצה הגדולה (נוסיף לסכום הקיים), לפי הערך בתא. נרוקן את המערך של הקבוצה הקטנה (יכיל אפסים). המערך בגודל קבוע ולכן הסיבוכיות הינה $O(1)$

לכן סה"כ הסיבוכיות היא $O(\log^* k + n)$ משוערך בממוצע על הקלט

$\text{StatusType addPlayer(void } * \text{DS, int PlayerID, int GroupID, int score)}$:

נבדוק: אם $DS == NULL$ או $GroupID > k$ או $GroupID \le 0$ או $PlayerID \le 0$ או $score > scale$ או $score \le 0$ אז נחזיר $INVALID\_INPUT$

נבדוק ב$hashTable$ של כלל השחקנים האם השחקן קיים- $O(1)$ משוערך בממוצע על הקלט, אם כן נחזיר $FAILURE$

אחרת: נוסיף את השחקן ל-$hashTable$ של כלל השחקנים ב-$O(1)$ משוערך בממוצע על הקלט. (שחקן יוכנס לעץ השלבים (של כלל השחקנים או של קבוצתו) רק אם השלב שלו מעל 0)

בנוסף, נעדכן $levelZeroScoreArr[score]++$ -$O(1)$

אח"כ נמצא את קבוצת השחקן ב-$O(\log^* k)$ באופן משוערך -כפי שראינו בהרצאה, ונוסיף את השחקן ל-$hashTable$ של שחקני קבוצתו ב-$O(1)$ משוערך בממוצע על הקלט. בנוסף נעדכן ב-$O(1)$:

$levelZeroScoreArr[score]++$

לכן סה"כ סיבוכיות הפונקציה היא $O(\log^* k)$ משוערך בממוצע על הקלט.

$\text{StatusType removePlayer(void } * \text{DS, int PlayerID)}$:

אם $DS == NULL$ או $PlayerID \le 0$ נחזיר $INVALID\_INPUT$, אחרת:

נבדוק אם השחקן קיים ב-$hashTable$ של כלל השחקנים ב-$O(1)$ משוערך בממוצע על הקלט, אם לא, נחזיר $FAILURE$. אחרת:

- נסיר את השחקן מה-$hashTable$ של כלל השחקנים ב-$O(1)$ משוערך בממוצע על הקלט. לפני זה נשמור את המידע לגביו: תוצאה, שלב ומספר קבוצה-$O(1)$
- **אם השלב של השחקן שונה מ-0:** נמצא בעץ השלבים של כלל השחקנים את השלב של השחקן ב-$O(logn)$ (כמות השלבים היא לכל היותר ככמות השחקנים במבנה) ונעדכן את המידע בשלב בהתאם: כמות השחקנים בתת העץ קטנה ב-1, כמות השחקנים בשלב קטנה ב-1, כמות השחקנים עם התוצאה של השחקן קטנה ב-1 במערך התוצאות וסכום השלבים בתת העץ קטן בהתאם, עדכונים אלו נעשים ב-$O(1)$ (נעדכן גם את השלבים מהשורש עד לשלב במהלך החיפוש- $O(logn)$ כשיתכנו גלגולים ב-$O(1)$)

  אם השלב של השחקן הוא 0, נעדכן: $levelZeroScoreArr[score]--$
- נמצא את הקבוצה של השחקן ב-$O(\log^* k)$ משוערך ונעשה את אותן הפעולות עבור עץ השלבים בקבוצה, מערך ה$levelZeroScoreArr$ וה-$hashTable$ שבה.

סה"כ $O(\log^* k + logn)$ משוערך בממוצע על הקלט

$\text{StatusType increasePlayerIDLevel(void } * \text{DS, int PlayerID, int LevelIncrease)}$:

אם $DS == NULL$ או $PlayerID \le 0$ או $LevelIncrease \le 0$ נחזיר $INVALID\_INPUT$

נבדוק אם השחקן קיים ב$hashTable$ של כלל השחקנים, אם הוא לא קיים נחזיר $FAILURE$-$O(1)$ משוערך בממוצע על הקלט.

אחרת, מצאנו את השחקן.

- נשמור במשתנה את השלב הנוכחי שלו-$past\_level$ והתוצאה, ונעדכן את השלב שלו ל-$past\_level + LevelIncrease$ -$O(1)$. (בזכות השימוש במצביעים חכמים זה מעדכן גם עבור האדם ב$hashTable$ של קבוצתו)
- **אם השלב הישן לא היה 0** המידע הרלוונטי על השחקן הוכנס לעץ השלבים לכן נבצע: ניגש לעץ השלבים של כלל השחקנים ונמצא את השלב $past\_level$ ב-$O(logn)$, אם $numOfPlayersAtLevel > 1$ ניצור העתק שלו -$O(1)$ כי כל המידע בעץ בגודל סופי שלא תלוי ב-n (מערכים, מצביעים, ושדות של מספרים)

  נסיר את השלב מהעץ. אם $numOfPlayersAtLevel > 1$ נעדכן את העותק שיצרנו ונכניס מחדש לעץ (הוא לא היה השחקן היחיד ולכן לא צריך להסיר את השלב)

נעדכן את המידע של ההעתק שלנו בהתאם (למשל כמות השחקנים בשלב זה קטנה ב-1, כמות השחקנים עם תוצאת השחקן בשלב זה קטנה ב-1...) -$O(1)$, ונכניס את ההעתק שיצרנו לעץ-$O(logn)$ (נעדכן את השלבים מהשורש עד לשלב במהלך החיפוש- $O(logn)$)

אם השלב היה 0 נעדכן: $levelZeroScoreArr[score]--$ -$O(1)$

- נמצא בעץ את השלב החדש של השחקן (אם קיים)-$O(logn)$, אם קיים היה כבר שחקן בשלב זה במשחק, לכן ניצור העתק של השלב ונעדכן מידע עבורו (למשל כמות השחקנים בשלב זה גדלה ב-1)-$O(1)$ ואז נסיר את המקורי מהעץ ב-$O(logn)$. אם השלב לא היה קיים ניצור חדש בעל שדות מעודכנים (כמות שחקנים בשלב היא 1,...)- $O(1)$
  נכניס את השלב שיצרנו (בעל השדות המעודכנים) לעץ-$O(logn)$
  (נעדכן את השלבים מהשורש עד לשלב במהלך החיפוש) –$O(logn)$
  נשים לך שכך רק שחקנים עם שלב גדול מ-0 "נמצאים" בעץ השלבים
- נמצא את הקבוצה של השחקן ב-$O(\log^* k)$ משוערך כפי שראינו בהרצאה
- נעדכן עבור עץ השלבים בקבוצה ואת המערך באותו האופן כמפורט מעלה-$O(logn)$

סה"כ הסיבוכיות היא $O(\log^* k + logn)$ משוערך בממוצע על הקלט

<u>בונוס (5 נק')</u>: תארו בחלק היבש כיצד ניתן היה לממש את הפעולה הנ"ל בסיבוכיות של $O(\log^*(k)+\log(n))$ משוערך (שימו לב שכאן הסיבוכיות אינה בממוצע על הקלט).

<u>תשובה:</u> הסיבוכיות נובעת מפקטור העומס ב-$hashTable$ שהוא $O(1)$ בממוצע על הקלט, במקרה בו הפילוג בטבלה מאוזן. במקרה הגרוע, יתכן תא ב- $hashTable$ שמכיל כמות גדולה של איברים ברשימה ולכן חיפוש האיבר ייקח זמן (כיוון שזה $hashTable$ מסוג שרשראות, התנגשויות נפתרות ע"י הוספת האיברים ברשימה, ולכן תיתכן רשימה מאוד ארוכה), כלומר אם n השחקנים "נפלו" על אותו התא, עלול לעלות לנו $O(n)$ למצוא איבר מסוים שנמצא בסוף הרשימה. ניתן לשנות את הסיבוכיות ל-$O(\log^* k + logn)$ משוערך, אם במקום שימוש ברשימה כשרשראות, כל תא יכיל עץ AVL מאוזן, לכן, פונקציית הערבול תוביל אותנו לתא המתאים במערך ב-$O(1)$ ומציאת האיבר המתאים בתא יעלה $O(logn)$ במקרה הרע (לא באופן ממוצע), בו כל השחקנים "נפלו" על אותו תא בעזרת פונקציית הערבול, כזמן שלוקח לחפש בעץ AVL עם n איברים. תהליך מציאת הקבוצה ושאר הפעולות בפונקציה מתבצעים ב-$O(log^*k + logn)$ משוערך סה"כ, לכן נקבל את הסיבוכיות הנדרשת. (אם נעשה m פעולות, ייתכן שכל פעולה דורשת את מציאת הקבוצה של השחקן, לאחר מציאת הקבוצה, מציאת השחקן נעשית ב-$O(logn)$ ופונקציות כמו עדכון שלב, עדכון תוצאה וכו' נעשה במקרה הגרוע ב-$O(logn)$ **בהינתן שכבר מצאנו את הקבוצה (לאחר לכל היותר k פעולות $union-find$)**, לכן אם יהיו לנו m פעולות על השחקנים נקבל במקרה הגרוע $O(m\cdot logn + m\cdot \log^* k)$ ולכן באופן משוערך $O(log^*k + logn)$, בפרט לפעולת העלאת השלב, ומכאן נקבל את הסיבוכיות המתבקשת)

$StatusType\ changePlayerIDScore(void\ *DS, int\ PlayerID, int\ NewScore)$:

אם $DS == NULL$ או $PlayerID \le 0$ או $NewScore > scale$ או $newScore \le 0$ נחזיר $INVALID\_INPUT$, אחרת: נבדוק אם השחקן קיים ב$hashTable$ של כלל השחקנים ב-$O(1)$ משוערך בממוצע על הקלט, אם הוא לא קיים נחזיר $FAILURE$, אחרת:

- נשמור במשתנה old_score את התוצאה הנוכחית של השחקן ונעדכן עבורו את התוצאה החדשה (יש מצביע לשחקן), וכן נשמור את השלב שלו ומספר קבוצתו-$O(1)$ (התוצאה מתעדכנת גם ב-$hashTable$ של קבוצתו)
- **אם השלב של השחקן שונה מ-0**, הוא קיים (השלב) בעץ השלבים ולכן נבצע:
- נמצא את השלב של השחקן בעץ השלבים, ניצור העתק שלו ונמחק את המקורי מהעץ ב-$O(logn)$ סה"כ
- נעדכן בהעתק במערך התוצאות באינדקס old_score שהכמות קטנה ב-1 -$O(1)$
- נעדכן בהעתק במערך התוצאות באינדקס new_score שהכמות גדלה ב-1 -$O(1)$
- נכניס ההעתק שיצרנו לעץ בסיבוכיות- $O(logn)$ (במהלך ההכנסה מעודכנים השדות בשלבים מהשלב החדש ועד השורש)

  אחרת השלב של השחקן הוא 0 לכן נבצע ב-$O(1)$:

  $levelZeroScoreArr[new\_score]++ \quad , \quad levelZeroScoreArr[old\_score]--$
- נמצא את הקבוצה של השחקן ב-$O(\log^* k)$ משוערך כפי שראינו בהרצאה
- **אם השלב של השחקן שונה מ-0** נמצא אותו בעץ השלבים של הקבוצה , ניצור העתק ונמחק את המקורי ב -$O(logn)$. נבצע עדכונים במערך התוצאות של העותק שיצרנו, בתוצאה הישנה של השחקן נפחית 1 בתא במערך, ובתוצאה החדשה נעלה ב-1 -$O(1)$ ונכניס מחדש לעץ-$O(logn)$

  אחרת, שלב השחקן הוא 0 ונבצע את העדכונים הבאים ב-$O(1)$ במערך שבקבוצתו:

  $levelZeroScoreArr[new\_score]++ \quad , \quad levelZeroScoreArr[old\_score]--$

סה"כ סיבוכיות $O(\log^* k + logn)$ משוערך בממוצע על הקלט

StatusType getPercentOfPlayersWithScoreInBounds
$(\text{void } *\text{DS, int GroupID, int score, int lowerLevel, int higherLevel, double } *\text{ players})$:
אם אחד המצביעים שווה ל-$NULL$ או $GroupID > k$ או $GroupID < 0$ יוחזר $INVALID\_INPUT$.
אם $GroupID == 0$ נבצע את הפעולות הבאות על המבנים של כלל השחקנים, אחרת אם $GroupID \neq 0$ נמצא את הקבוצה ב-$O(\log^* k)$ משוערך ונבצע את הפעולות הבאות על המבנים שבתוך הקבוצה- הפעולות עצמן זהות:

- נשמור 2 משתנים: $numPeopleAtLevelZero = 0$ ו-$numPeopleAtLevelZeroWithScore = 0$
- אם $lowerLevel \leq 0$, נשיג את כמות האנשים בשלב 0 ונוסיף ל-$numPeopleAtLevelZero$ (דרך ה-$hashTable$ נקבל את כמות סך השחקנים ב-$O(1)$- נתון זה נשמר במבנה, ודרך העץ נשיג את כמות השחקנים שבשלבים גדולים מ-0 ב-$O(1)$ – ניגש לשורש וניקח את כמות השחקנים בתת העץ. נחסר בין 2 אלו ונקבל את כמות השחקנים שבשלב 0)
- אם $lowerLevel \leq 0$ וגם $0 < score \leq scale$, ניגש למערך של תוצאות שחקני שלב 0, ונקבל את כמות השחקנים שבשלב 0 עם התוצאה $score$ ב-$O(1)$, נעדכן נתון זה ב-$numPeopleAtLevelZeroWithScore$
- נשמור משתנה $fakeLevels = 0$ למקרה שלא קיימים בעץ איברים עם השלבים של התחום (יתבהר בהמשך) -$O(1)$
- אם יש שחקנים בשלב 0, והשלב הגבוה בתחום גדול או שווה ל-0, יתכן (לא בהכרח) ששחקני שלב 0 יספרו בתחום, לכן נוסיף איבר לעץ עם שלב 0, עם כמות השחקנים שבשלב 0, ועם כמות השחקנים בשלב 0 עם התוצאה המבוקשת במערך התוצאות של הקבוצה. יצירתו והוספתו תעלה סה"כ $O(logn)$. אם אכן הכנסנו שלב 0 לעץ נסמן זאת לעצמנו ובסוף לפני החזרה מהפונקציה נוציא אותו מהעץ- $O(logn)$
- ניגש לעץ השלבים ונבצע פעולות $find$ כדי למצוא צמתים בעץ עם השלבים $lowerLevel$, $higherLevel$. -$O(logn)$, עבור כל שלב שלא נמצא, נעדכן: $fakeLevels++$, ניצור שלב חדש עם שחקן אחד **בעל תוצאה לא חוקית 0**, ונכניס לעץ-$O(logn)$. **לא נציין זאת בהמשך, אבל לפני החזרת התוצאה מהפונקציה נוציא את השלבים המזוייפים שיצרנו מהעץ** ב-$O(logn)$
- נשמור משתנים: $numOfPlayersBetweenBounds = 0$ ו-$numOfPlayersBetweenBoundsWithScore = 0$ - $O(1)$
  - אם $low == high$:
    1. אם $score < 0$ או $score > scale$ לא יכולים להיות אנשים עם התוצאה כיוון שהיא אינה חוקית וכמות האנשים בתחום היא כמות האנשים בשלב. נחשב ב-$O(1)$:
       $$numOfPlayersBetweenBounds = low \rightarrow numPeopleAtLevel - fakeLevels$$
       אם הכמות היא 0 נחזיר שגיאה -$O(1)$
       נחזיר: $*\text{ players} = 0$ (בלי קשר לאם הייתה שגיאה או לא) -$O(1)$
    2. אחרת $score$ בטווח התוצאות האפשריות ולכן ייתכן שיש אנשים בטווח עם התוצאה: נחשב:
       $$numOfPlayersBetweenBounds = low \rightarrow numPeopleAtLevel - fakeLevels$$
       $$numOfPlayersWithScore = low \rightarrow groupScoreArr[score]$$
       (כשהוספנו שלבים שלא קיימים הכנסו אנשים עם התוצאה 0 ולכן לא נחסיר את כמות השלבים המזויפים מהחישוב כי הם לא נספרו ממילא) – זה נעשה ב-$O(1)$
       נבדוק: אם כמות האנשים בטווח היא 0 נחזיר שגיאה וערך 0
       אחרת, לא נחזיר שגיאה, ונחזיר את הערך:
       $$100 \cdot \frac{numOfPlayersWithScore}{numOfPlayersBetweenBounds} : \text{-}O(1)$$
  - אם $low \neq high$:

נמצא את האב המשותף של 2 גבולות השלבים ב-$O(logn)$ באופן רקורסיבי [הוא בהכרח קיים כי יש לפחות 2 שלבים בעץ (אותם אולי הכנסנו בעצמנו) ובמקרה זה אחד מהם הוא האב המשותף]. איך נמצא? נבצע חיפוש במקביל, נתחיל מהשורש, אם 2 השלבים גדולים מהשלב בשורש, נמשיך לחפש מימינו, אם שניהם קטנים ממנו נחפש משמאלו, אחרת נחזיר את הצומת הנוכחי שאנחנו בו.אנחנו עוברים על העץ ויורדים כל פעם רמה בחיפוש, ולכן במקרה הגרוע יעלה $O(logn)$ כאורך המסלול הארוך ביותר. לאחר שמצאנו את האב, נחשב:

1. נשמור במשתנה את כמות האנשים בשלב, ואת כמות האנשים בשלב עם התוצאה המבוקשת $score$ -$O(1)$ בעזרת גישה לשדות המתאים בצומת ולמערך במקום $score$

2. נרד מהאב המשותף עד שנמצא את הצומת $highLevel$, וכל עוד השלב הגבוה הוא גם לא האב המשותף, אם השלב נמוך מהשלב הגבוה או שווה לו, נסכום את כמות האנשים בשלב וכמות האנשים עם התוצאה ונשמור ב-2 משתנים, אם נמשיך בחיפוש ימינה או אם הגענו לשלב הדרוש, נסכום גם את המידע **מתת העץ השמאלי** במידה והוא לא $null$ (ניקח את כמות האנשים **בתת העץ שלו** ואת כמות השחקנים עם התוצאה מהמערך של **תת העץ שלו**), אחרת נמשיך בחיפוש שמאלה ולא נסכום מידע מתת העץ הימני (כיוון שהשלב בו גבוה מדי). זה ייקח $O(logn)$ במקרה הרע, בו האב המשותף זה השורש והצומת עם השלב הגבוה היא עלה, כאשר המידע שצריך לסכום מתקבל ב-$O(1)$ בעזרת גישה לשדות השמורים בצמתים

3.באופן דומה נסכום את כמות השחקנים עד לשלב הנמוך בתחום, ואת כמות השחקנים עם התוצאה $score$ עד לשלב הנמוך, כאשר הפעם נסכום את תת העץ הימני במקרה בו נפנה שמאלה או הגענו לצומת שהוא השלב שלנו, ולא נסכום תת עץ שמאלי, כי בתת עץ שמאל יש איברים שקטנים מהחסם התחתון של התחום, וצד שמאל גדול מהשלב הנמוך וכן דורש סכימה -$O(logn)$

4.נחבר את התוצאות שקיבלנו בכל 1 מהשלבים 1-3 וכך נקבל את כמות השחקנים בתחום (ממנה נחסיר את כמות השלבים המזויפים) ואת כמות השחקנים עם התוצאה $score$ נסמן את המשתנים שהם שמורים בהם סה"כ ב-$numOfPeopleBetweenBounds$ ו- $numOfPlayersWithScore$

אם $numOfPeopleBetweenBounds == 0$ נחזיר 0 ו-$FAILURE$- $O(1)$

אחרת, נחשב ונחזיר את הערך $100 \cdot \frac{numOfPlayersWithScore}{numOfPlayersBetweenBounds}$ -פעולות אלו עולות $O(1)$

לכן סה"כ סיבוכיות $O(\log^* k + logn)$ משוערך .

StatusType averageHighestPlayerLevelByGroup (void * DS, int GroupID, int m, double * avgLevel):

אם אחד המצביעים שווה ל-$NULL$ או $GroupID > k$ או $GroupID < 0$ או $m \leq 0$ נחזיר $INVALID\_INPUT$. אם $GroupID == 0$ נבצע את הפונקציה על עץ השלבים של כלל שחקני המערכת, אחרת נמצא את הקבוצה ב-$O(\log^* k)$ משוערך ונבצע את הפעולה על עץ השלבים של הקבוצה- השלבים זהים:

- נבדוק: אם כמות השחקנים (מידע ששמור לנו וניתן לנו ב-$O(1)$ מה-$hashTable$) קטנה ממש מ-m, נחזיר שגיאה וערך 0. -$O(1)$
- בעת, אנחנו יודעים שסך כמות השחקנים שיש לנו, גדולה או שווה מ-m ולכן בהכרח נוכל לסכום m שלבים (אולי זהים) ולהחזיר הצלחה. נזכיר ששחקני שלב 0 נמצאים במערך משלהם, ורק שחקנים עם שלב גדול מ-0 נמצאים בעץ. לכן, נבדוק: אם כמות השחקנים בעץ היא 0 (העץ ריק, השורש $NULL$), נחזיר ערך 0 והצלחה. זו בדיקה שנעשית ב -$O(1)$
- בעת אנחנו יודעים שיש שחקנים בעץ. נגדיר: $m_{new} = \min\{m, root \rightarrow numPeopleAtSubTree\}$, כי אנחנו יכולים לסכום על לכל יותר בכמות השחקנים בעץ, ושאר השחקנים יהיו ברמה 0 ולא יתרמו לסכימה.-$O(1)$
- נפעיל אלגוריתם סכימה לסכימת m_{new} השחקנים ברמות הגבוהות ביותר, בעזרת פונקציה שתמומש בעץ ותקבל: סכום התחלתי (בהתחלה 0),כמות אנשים לסכימה(m_{new}), ואיבר בעץ ממנו להתחיל את הסכימה (בהתחלה השורש). האלגוריתם יפעל כך:
  - בהינתן ה-$node$ שהתקבל הפונקציה, האם יש לו בן ימני?
  - אם יש בן ימני:

האם כמות האנשים בתת העץ בבן הימני גדולה או שווה ל-m_{new}?

- אם כן, תחזיר את ערך החזרה מקריאה לפונקציה עם $node \rightarrow right, m_{new}, sum$ (בשורש $sum = 0$, אם זו קריאה לא מהשורש, יתקבל סכום כלשהו)
- אחרת, סכום: $sum += node \rightarrow right \rightarrow sumLevelsAtSubTree$

  וחשב: $m_{new} = m_{new} - node \rightarrow right \rightarrow numPeopleAtSubTree$

  (אם m_{new} הוא 0 בבדיקה הבאה יוחזר הערך)

  כעת בדוק: האם כמות האנשים ב-$node$ ב-$numPeopleAtLevel$ (כמות האנשים בשלב עצמו! לא בתת עץ) גדולה או שווה ל-m_{new}?

  - אם כן, הוסף $sum += m_{new} \cdot node \rightarrow level$ ותחזיר ערך זה.
  - אחרת, חשב: $sum += (node \rightarrow level) \cdot (node \rightarrow numPeopleAtLevel)$

  ועדכן: $m_{new} = m_{new} - node \rightarrow numPeopleAtLevel$

  והחזר את ערך הקריאה לפונקציה עם sum, m_{new} המעודכן ו-$node \rightarrow left$ (נשים לב שאיבר שמאל בהכרח קיים כי אנחנו יודעים שכמות השחקנים בתת העץ שאנחנו בו בכל שלב גדולה או שווה ל-m_{new} ועברנו על האיבר עצמו ועל הסכום בימינו)

- אם אין בן ימני: האם כמות השחקנים ב-$node$ הנוכחי ($node \rightarrow numPeopleAtLevel$) גדולה או שווה ל-$m_{new}$?
  - אם כן, סכום ותחזיר את הערך $sum += node \rightarrow level \cdot m_{new}$
  - אחרת, חשב: $sum += node \rightarrow level \cdot node \rightarrow numPeopleAtLevel$

    $m_{new} = m_{new} - node \rightarrow numPeopleAtLevel$

    והחזר את ערך ההחזרה מהקריאה לפונקציה עם $node \rightarrow left, m_{new}, sum$

    (כמו קודם, יודעים שהוא קיים, אחרת כבר בבדיקה בהתחלה הייתה חוזרת שגיאה)

לבסוף נכניס למצביע שקיבלנו את תוצאות החלוקה של הסכום שחישבנו ב-m המקורי ונחזיר הצלחה.
אם מספר הקבוצה היה 0, האלגוריתם יפעל על עץ השלבים של כלל השחקנים, אחרת נמצא את הקבוצה ב-$O(\log^* k)$ משוערך כמו שראינו בהרצאה ונפעיל את האלגוריתם על העץ שבה.
במהלך האלגוריתם אנחנו מתקדמים ימינה או שמאלה בהתאם לכמות השחקנים בתת העץ, בדומה לחיפוש, ולכן רק עושים מסלול מהשורש, עד לכל היותר עלה. לכן אורך המסלול הארוך ביותר הוא $O(logn)$ [בגלל השדות של כמות השחקנים בתת העץ, סכום שלבים בתת העץ וכדומה, אם אנחנו רואים שאי אפשר להשיג את הכמות המבוקשת מללכת ימינה או שמאלה, אנחנו מקבלים מידע על תת העץ הימני או השמאלי ב-$O(1)$ ומתקדמים בחיפוש בצד השני, לכן לא נעבור על יותר מ-$O(logn)$ צמתים],לכן הסיבוכיות הכוללת היא $O(\log^* k + logn)$ משוערך.

$\text{void Quit(void} \ ** \ \text{DS)}$:

נהרוס את המבנים שיצרנו: כל קבוצה נהרוס בסיבוכיות $O(k)$ וכל עץ או $hashTable$ בקבוצה בסיבוכיות $O(n_x)$ כאשר n_x זה מספר האיברים בעץ או בטבלה. נשים לב שכל קבוצה מכילה כמות של שלבים מסוימת וכמות של שחקנים מסוימת, כך שכל הקבוצות יחד מכילות $O(n)$ איברים (לכל היותר קבוע כפול n), לכן סה"כ סיבוכיות הריסת כל הקבוצות היא $O(k+n)$. בנוסף נהרוס את העץ של כלל השחקנים ואת ה-$hashTable$ של כלל השחקנים - בסיבוכיות $O(n)$ סה"כ כמספר השחקנים (עבור מערך התוצאות: כמות השלבים חסומה ע"י כמות השחקנים ומתפזרת בין תוצאות השחקנים. עבור עץ השלבים: יהיו לכל היותר כמות שלבים בכמות שחקנים, ועבור הטבלה , מכיוון שגודלה תלוי בכמות השחקנים גודלה $O(n)$)
לכן סה"כ סיבוכיות הפונקציה היא $O(k+n)$ במקרה הגרוע בו כמות השלבים שווה לכמות השחקנים

סיבוכיות מקום:

נשים לב שבכל הפונקציות הנדרשות לא עברנו את סיבוכיות המקום הנדרשת ממבני הנתונים ולכן בסה"כ מבנה הנתונים והפעולות המוגדרות עבורו עומדות בסיבוכיות המקום הנדרשת מאיתנו שהינה $O(n+k)$ כאשר k זה כמות הקבוצות ו-n זה כמות השחקנים בכלל המערכת.