רטוב 2:

הסבר על מבני הנתונים הבסיסיים שהשתמשנו בהם:

המבנה הבסיס:

Generic Union Find :

השימוש העיקרי באלגוריתם זה כדי לנצל את זה שלפי המימוש הרביעי שראינו בהרצאה (איחוד לפי גודל וכיווץ מסלולים) הסיבוכיות המשוערכת לכל פעולה היא $O(\log^* k)$, מה שמאפשר לנו לקיים את דרישת סיבוכיות הזמן ברוב הפונקציות למשל בmerge נשתמש בUnion או כשרוצים לחפש קבוצה מסויימת אז משתמשים בFind.

Hash Table :

טבלת ערבול דינאמית של כל השחקנים במערכת, שמכילה אובייקטים מסוג שחקן (PlayerByLevel/PlayerByScore תחת Player.h). טבלת הערבול מאותחלת עם גודל קבוע מראש וגד לה/קטנה בהתאם לעומס בטבלה. טבלת הערבול של השחקנים מערבלת ע"פ מזהי השחקן 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 הAVL מהתרגיל הקודם ו ניצלנו את זה שהחיפוש נעשה בסבוכניות $O(logn)$, אבל בנוסף עכשיו יש לנו את הscore שנכנס לתמונה והשימוש בrank עוזר לנו במימוש.

המבנה העיקרי:

1. Union Find אחד של הקבוצות:

נעשה על מערך של מצביעים ל k הקבוצות, כל קבוצה כזו מכילה את המספר המזהה שלה, מצביעם לשני העצים של

השחקנים ממויינים לפי הscore/level ומצביע לעץ הlevels כפי שיתואר בהמשך.

2. <u>Hash Table אחד של השחקנים:</u>

טבלת ערבול דינמית שמכילה את כל האבייקטים (שמכילים את המזהה של השחקן, המזהה של הקבוצה שהוא שייך לה, הרמה והדרגה) של כל השחקנים במשחק. מה שמאפשר לנו למצוא אם השחקן נמצא במשחק או לא במימוש הפונקציות.

3. <u>3 עצי AVL Ranked Table:</u>

כל אחד מהעצים הבאים נמצא גם במבנה הכללי של המשחק וגם בתוך טיפוס הקבוצה Group.

**** כל הוספה/הסרה של אחד הצמתים בעצים אלה נעשית בסיבוכיות של $O(logn)$ כאשר n הוא מספר השחקנים בכל המשחק ****

1. <u>השחקנים ממוינים לפי הlevel שלהם</u>

כל מפתח בעץ הוא מטיפוס PlayerByLevel שמכיל בתוכו את מזהה הקבוצה של השחקן, הscore והlevel של השחקן, וערך הdata של כל שחקן הוא המזהה שלו playerID, העץ מסודר בסדר יורד לפי הlevel של השחקן, ואם לשני שחקנים יש את אותו הlevel אז לפי הscore שלהם.

2. <u>השחקנים ממוינים לפי הscore שלהם</u>

כל מפתח בעץ הוא מטיפוס PlayerByScore שמכיל בתוכו את מזהה הקבוצה של השחקן, הscore והlevel של השחקן, וערך הdata של כל שחקן הוא המזהה שלו playerID, העץ מסודר בסדר יורד לפי הscore של השחקן, ואם לשני שחקנים יש את אותו הscore אז לפי הlevel שלהם.

3. <u>הLevels של השחקנים</u>

כל מפתח בעץ הוא הlevel של שחקן כאשר הdata מכיל את המזהה של השחקן הזה playerID.

<u>נפרט עבור כל פונקציה את הסיבוכיות של המימוש שלה:</u>

**** כל הבדיקות של תקינות הקלט נעשית בסיבוכיות $O(1)$ לכן לא נזכיר אותם בניתוח הסיבוכיות בפונקציות, כנ"ל לגבי עדכון מצביעים ואובייקטים בפונקציות ****

1. void* init(int k, int scale):

אתחול של מצביע למבנה ריק שכולל עצים, hash ומערך של מצביעים לקבוצות לאלגוריתם הUnion Find ריקים, האתחול של העצים והhash הוא בסיבוכיות של $O(1)$, מספר הקבוצות הינו נתון וידוע k לכן בסך הכל האתחול הוא בסיבוכיות $O(k)$.

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

מוצאים את הקבוצות בפעולת Find לפי המזהה שלהם, ואז משתמשים בפעול Union של הUnion Find כדי לאחד את שתי הקבוצות, שתי הפעולות האלה במימוש שלנו הן בסיבוכיות משוערכת של $O(\log^* k)$. וכי צריכים להפריד בסיבוכיות בין זאת של הקבוצות ושל השחקנים יש לנו סיבוכיות בשחקנים של n כמספר השחקנים בשתי הקבוצות לכן יש לנו בסך הכל $O(log^* k + n)$.

3. StatusType addPlayer(void *DS, int PlayerID, int GroupID, int score):

מוסיפים את השחקן לטבלת הערבול של השחקנים בסיבוכיות $O(1)$, לא מוסיפים את השחקן לעצים בשלב הזה כדי לקיים את הסיבוכיות, אבל שומרים לכל קבוצה counter ששומר לנו את השחקנים שעוד לא עודכן להם את הlevel לשימוש שנפרט בהמשך, לצורך עדכון הcounter הזה מוצאים את הקבוצה לפי פעולת Find בUnion Find בסיבוכיות של $O(\log^* k)$ וזו היא הסיבוכיות הכוללת של הפונקציה כנדרש.

4. StatusType removePlayer(void *DS, int PlayerID):

קודם נמצא את השחקן לפי המזהה שלו בHash Table של השחקנים בסיבוכיות $O(1)$, ואז מהשחקן מקבלים את מזהה של הקבוצה שהוא שייך לה ומוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות $O(log^*k)$, ובסוף מוצאים את השחקן ומוחקים אותו מהעצים בסיבוכיות של $O(logn)$. בסך הכל מקבלים סיבוכיות של $O(log^*k + logn)$.

5. StatusType increasePlayerIDLevel(void *DS, int PlayerID, int LevelIncrease):

קודם נמצא את השחקן לפי המזהה שלו בHash Table של השחקנים בסיבוכיות $O(1)$, ואז מהשחקן מקבלים את מזהה של הקבוצה שהוא שייך לה ומוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות $O(log^*k)$, ובסוף לפי קריאת הcounter שדיברנו עליה מקודם מוסיפים את השחקן או מעדכנים את הlevel שלו בעצים בסיבוכיות של $O(logn)$. בסך הכל מקבלים סיבוכיות של $O(log^*k + logn)$ כנדרש.

6. StatusType changePlayerIDScore(void *DS, int PlayerID, int NewScore):

קודם נמצא את השחקן לפי המזהה שלו בHash Table של השחקנים בסיבוכיות $O(1)$, ואז מהשחקן מקבלים את מזהה של הקבוצה שהוא שייך לה ומוצאים אותה באמצעות הFind של הUnion Find בסיבוכיות $O(log^*k)$, ובסוף מוצאים את ומעדכנים את הscore של השחקן בעצים בסיבוכיות של $O(logn)$. בסך הכל מקבלים סיבוכיות של $O(log^*k + logn)$.

7. StatusType getPercentOfPlayersWithScoreInBounds (void *DS, int GroupID, int score, int lowerLevel, int higherLevel, double * players):

קודם מוצאים את הקבוצה לפי המזהה שלה בUnion Find בסיבוכיות של $O(\log^* k)$, ואחרי זה מחפשים את החסמים של הlevels בעץ של Levels של הקבוצה, משום שכל צומת בעץ Ranked AVL מחזיק את מספר הצמתים שקטנים ממנו אז נקח את הפרש הרמות של שני החסמים. בסך הכל נקבל סיבוכיות של $O(log^*k + logn)$ כנדרש.

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

קודם נמצא את הקבוצה לפי המזהה שלה בUnion Find בסיבוכיות משעורכת של $O(log^*k)$, ואז נשתמש בעץ של הרמות כדי לקבל הסכום של m הרמות הגבוהות ביותר בסיבוכיות של $O(n)$ כי במקרה הגרוע כל השחקנים יכולים להיות בm הרמות הגבוהות ביותר, לכן הסיבוכיות בסך הכל היא $O(log^*k + logn)$ כנדרש.

9. void Quit(void **DS):

מאחר והקצנו $O(n + k)$ (k קבוצות וn שחקנים) מקום, עלינו לעבור על כל המקום ולשחרר אותו. לכן סיבוכיות הזמן היא בהתאם $O(n + k)$.