הסבר למבנה הנתונים:

מבנה הנתונים שלנו הולך להתבסס על עצי AVL.
לפני המימוש הכולל, נממש מחלקה שתיקרא $AVLTree$ וממנה ירשו $groupTree$ ו־$playerTree$ כפי שיוסבר בהמשך.

הסבר על המחלקות שמומשו:

1. מחלקת $BTreeNode$: תהיה גנרית, ולכן $template < T >$.

תכיל:

(א) $T^*\ value$ - מצביע לערך של ה־$node$.

(ב) $BTreeNode^* r\_child, l\_child$

(ג) $int\ height$

המתודות שנממש + סיבוכיות:

(א) $goLeft(), goRight()$ - סיבוכיות $O(1)$

(ב) $getValue()$ - סיבוכיות $O(1)$

(ג) $getHeight()$ - סיבוכיות $O(1)$

(ד) $setHeight()$ - סיבוכיות $O(1)$

(ה) $setValue()$ - סיבוכיות $O(1)$

2. מחלקת $AVLTree$ - תהיה גם היא גנרית ולכן $template < T >$. המחלקה היא $friend$ של $BTreeNode$.

תכיל:

(א) $BTreeNode^*\ source$ - מצביע לתחילת העץ.

המתודות שנממש+סיבוכיות:

(א) $max(int\ x, int\ y)$ - $private$ -מתודה פנימית למציאת הגובה המקסימלי של תתי העצים. ישמש למציאת גובה העץ. סיבוכיות $O(1)$

(ב) $RR(BTreeNode^*\ node)$ - $private$ - סיבוכיות $O(1)$

(ג) $LL(BTreeNode^*\ node)$ - $private$ -סיבוכיות $O(1)$

(ד) $getBalance(BTreeNode^*\ node)$ - $private$ -סיבוכיות $O(1)$

(ה) $insert(T^*\ value)$ - סיבוכיות $O(log(h))$ כאשר h הוא גובה העץ

(ו) $delete(T^*\ value)$ - סיבוכיות $O(log(h))$ כאשר h הוא גובה העץ

(ז) $find(T^*\ element)$ - אלגוריתם למציאת צומת בעץ. סיבוכיות $O(log(h))$ כאשר h הוא גובה העץ

(ח) $inOrder(), preOrder(), postOrder()$ - סיבוכיות $O(n)$

3. מחלקת $Player$ - תייצג שחקן שמשחק במשחק.

תכיל:

(א) key - int יאותחל לפי ID או לפי $LEVEL$ לפי הצורך

(ב) $PlayerID$ - int והוא ת"ז השחקן

(ג) $Level$ - int והוא רמת השחקן

המתודות שנממש+סיבוכיות:

(א) $operator <$ - תבצע השוואת key של שני שחקנים. סיבוכיות $O(1)$

(ב) $setKey(int\ key)$ - סיבוכיות $O(1)$

(ג) $setLevel(int\ level)$ - סיבוכיות $O(1)$

4. מחלקת $Group$ - תייצג קבוצת שחקנים במשחק.

תכיל:

(א) $groupID$ - ת"ז הקבוצה, int.

(ב) $players$ $AVLTree < player >$ - עץ המכיל את שחקני הקבוצה לפי סדר $playerID$.

(ג) $strongestPlayer$ - int של ת"ז של השחקן בעל ה-$level$ הגבוה ביותר בקבוצה.

(ד) $numOfPlayers$ - מספר השחקנים בקבוצה

המתודות שנממש+סיבוכיות:

(א) $operator <$ - תבצע השוואת $groupID$ של שני קבוצות. סיבוכיות $O(1)$

(ב) $setStrongestPlayer(Player\ player)$ - סיבוכיות $O(1)$

(ג) $getStrongestPlayer(Player\ player)$ - סיבוכיות $O(1)$

(ד) $getNumOfPlayers()$ - סיבוכיות $O(1)$

(ה) $setNumOfPlayers()$ - סיבוכיות $O(1)$

5. מחלקת $PlayersManager$ - תייצג את מבנה הנתונים שלנו

תכיל:

(א) $eGroup$ $AVLTree < group >$ - עץ המכיל את הקבוצות הריקות.

(ב) $fGroup$ $AVLTree < group >$ - עץ המכיל את הקבוצות המכילות לפחות שחקן 1.

(ג) $players$ $AVLTree < player >$ - עץ המכיל את שחקני הקבוצה לפי סדר $level$.

(ד) $strongestPlayer$ - int ישמר ה-ID של השחקן החזק ביותר מכולם.

הסברי סיבוכיות: (כל הפעולות בעץ חיפוש AVLהינם $O(log(h))$

- $Init()$ - $O(1)$ - ניצור 3 עצים ריקים.
- $AddGroup$ - $O(log(k))$ - עץ חיפוש AVL.
- $AddPlayer$ - $O(log(n) + log(k))$ - במקרה הכי גרוע נעבור קודם על העץ של הקבוצות המלאות וגם על העץ של הקבוצות הריקות,