מבני נתונים אבסטרקטיים

AVL Tree – מחלקה אבסטרקטית של עץ כאשר קיימים טמפלייט אחד למפתח, טמפלייט שני לערך, וטמפלייט שלישי לפונקציית השוואה בין מפתחות.

במחלקה מוגדרת מחלקה פנימית בשם node עם השדות הפנימיים הבאים:

- **מפתח**
- **ערך**
- **בן ימני**
- **בן שמאלי**
- **גובה**

למחלקה השדות הבאים:

- שורש המחזיק טיפוס מצביע חכם לטיפוס node
- מספר הצמתים בעץ

למחלקה המתודות הבאות:

1. **find**– מציאת צומת בעץ.
2. **insert** – הכנסת צומת חדש לעץ.
3. **remove**– הסרת צומת מהעץ.
4. **getSize**– החזרת מספר הצמתים בעץ.
5. **findMax** - החזרת הערך המתאים למפתח המקסימלי בעץ.
6. **merge** - מקבלת עץ אחר X ומעבירה את כל צמתי X אליו עד ש X ריק.
   a. העברת השחקנים מעצי שתי הקבוצות (באמצעות מעבר inorder) לשני מערכים ממוינים בהתאמה $O(n_{replacement} + n_{group})$.
   b. מיזוג שני המערכים הממוינים של השחקנים למערך יחיד $O(n_{replacement} + n_{group})$.
   c. העברת מערך השחקנים הממוזג לעץ שחקנים יחיד $O(n_{replacement} + n_{group})$.
7. **inOrder** – מקבלת פנקטור הפועל על ערך, ומפעילה אותו לפי סדר על כל הערכים בעץ.
8. **reverseInOrder** - מקבלת פנקטור הפועל על ערך, ומפעילה אותו לפי סדר הפוך על כל הערכים בעץ.

כמו כן למחלקה מחלקת (Function Object) TreeToArray פנימית (עזר) שמקבלת צומת ומיישמת אותו למקום מתאים במערך.

טיפוסים

PlayersManager – מחלקת האב של כלל מבנה הנתונים, עם השדות הפנימיים הבאים:

1. **Groups** – עץ שצומת בו מורכב ממספר הקבוצה (מפתח) ומטיפוס Group (מצביע shared_ptr לטיפוס קבוצה – זהו הערך).
2. **ActiveGroups** – עץ שצומת בו מורכב ממספר הקבוצה (מפתח)) ומטיפוס Group (מצביע shared_ptr לטיפוס קבוצה – זהו הערך) המונה שחקן אחד לכל הפחות.

3. **Players** – עץ שצומת בו מורכב ממספר השחקן (מפתח) ומטיפוס Player (מצביע shared_ptr לטיפוס שחקן – זהו הערך).
4. **Players_by_level** – עץ שצומת בו מורכב מטיפוס Player (מצביע shared_ptr לטיפוס שחקן – זהו הערך <u>והמפתח</u>).
5. **highest_level_player** – מצביע לשחקן שנמצא בשלב הגבוה ביותר <u>במערכת</u>.

Group - מחלקה עם השדות הפנימיים הבאים:

1. **id** – מספר הקבוצה.
2. **playersInGroup** – עץ שחקנים השייכים לקבוצה זו, הממוין לפי השלב של השחקן ובמקרה הצורך (כאשר יש שוויון בין שלבי השחקנים) לפי מספר השחקן.
3. **highest_level_player** – מצביע לשחקן שנמצא בשלב הגבוה ביותר <u>בקבוצה</u>.

למחלקה המתודות הבאות:

1. **insert** – הכנסת שחקן חדש לעץ players_in_group. הפונ' תעדכן את highest_level_player ואת Number_of_players.
2. **remove** – מחיקת שחקן קיים בעץ players_in_group. הפונ' תעדכן את highest_level_player ואת Number_of_players.
3. **getId** – החזרת מספר מזהה הקבוצה.
4. **replace** - מקבלת קבוצה אחרת אשר את שחקניה מספחת אליה. מעטפת למתודת המיזוג של העץ.
5. למחלקה מתודות נוספות שעוטפות את המתודות של העץ ומאפשרות גישה חיצונית למתודות העץ.

Player – מחלקה עם השדות הפנימיים הבאים:

1. **id** – מספר השחקן
2. **level** – שלב השחקן
3. **groupId** – מספר קבוצת השחקן.

למחלקה המתודות הבאות:

1. **getId** – החזרת את מספר השחקן.
2. **getLevel** – החזרת את שלב השחקן.
3. **אופרטור השוואה >** : משווה בין שני שחקנים ומחזיר אמת אם <u>השלב</u> של השחקן הימני גדול מהשמאלי. אם מתקיים שוויון מחזיר אמת אם <u>מספר מזהה</u> השחקן הימני גדול מהשמאלי.

PlayerToArray – מחלקת Functor עם השדות הבאים:

1. **ptr_array** – מערך של מצביעים.
2. **num_of_groups** – מספר הקבוצות מהן נרצה את השחקן הגבוה ביותר.
3. **Index** - משתנה עזר.

למחלקה המתודות הבאות:

1. **אופרטור ()** : המתודה מקבלת מצביע לשחקן/ קבוצה (באמצעות העמסת פונקציות) ואינדקס במערך, ומציבה את מספר השחקן/ השחקן הגבוה בקבוצה בתא זה.

<u>מימוש הפונקציות כולל ניתוח סיבוכיות</u>

****

*void** ***Init*** *()*

O (1) במקרה הגרוע.

- יצירת המבנה PlayerManager, ואתחול השדות על ידי :

  - יצירת עץ ריק Groups
  - עץ ריק ActiveGroups
  - עץ ריק Players
  - עץ ריק Players_by_level
  - מצביע highest_level_player המאותחל ל-NULL.

כל העצים מאותחלים ללא איברים ולכן מדובר במספר קבוע של פעולות לכל המבנה O(1)

****

StatusType ***AddGroup*** *(void *DS, int GroupID)*

O (log k) במקרה הגרוע, כאשר *k* הוא מספר הקבוצות.

- בדיקה האם מספר הקבוצה תקין O(1)
- חיפוש מספר הקבוצה בעץ הקבוצות כדי לבדוק אם כבר קיימת O(log k)
- יצירת אובייקט Group חדש, מפתח וצומת לעץ הקבוצות O(1)
- הכנסת הצומת לעץ הקבוצות O(log k)

****

StatusType ***AddPlayer*** *(void *DS, int PlayerID, int GroupID, int Level)*

O (log n + log k) במקרה הגרוע, כאשר *n* הוא מספר השחקנים ו-*k* הוא מספר הקבוצות.

- בדיקת המצביע למבנה הנתונים ותקינות מזהי השחקן,הקבוצה והשלב O(1)
- יצירת שחקן חדש O(1)
- הכנסת השחקן החדש לעץ השחקנים Players O(log n).
- הכנסת השחקן החדש לעץ השחקנים Players_by_level O(log n).
- הצבה של השחקן בעל השלב הגבוה ביותר בכל המשחק בתוך השדה Highest_level_player של מחלקת האב באמצעות מתודת findMax של העץ Players_by_level O(log n).
- מציאת הקבוצה בעץ הקבוצות O(log k)
- אם טרם הוכנסו אליה שחקנים, נוסיף את הקבוצה לעץ הקבוצות הפעילות O(log k)
- הכנסת השחקן לעץ השחקנים של הקבוצה players_in_group O(log n).
- הצבה של השחקן בעל השלב הגבוה ביותר בקבוצה בתוך השדה Highest_level_player של הקבוצה באמצעות מתודת max של העץ players_in_group O(log n).

****

****

StatusType ***RemovePlayer*** *(void *DS, int PlayerID)*

O (log n) במקרה הגרוע, כאשר n הוא מספר השחקנים.

- בדיקה של מזהה השחקן ושל מצביע מבנה הנתונים O(1)
- מציאת השחקן בעץ השחקנים Players O(log n)
  - אם קיים, שמירה השחקן כמשתנה זמני O(1)
  - אחרת, נחזיר שגיאה מתאימה O(1)
- הוצאת השחקן מעץ השחקנים Players O(log n).
- הוצאת השחקן מעץ השחקנים Players_by_level O(log n).
- הצבה של השחקן עם המפתח הגדול ביותר בעץ Players_by_level (נמצא אותו על ידי התקדמות ימינה בעץ ככל האפשר) בתוך השדה Highest_level_player של מחלקת האב PlayersManager. O(log n).
- מציאת הקבוצה של השחקן, והוצאת השחקן מעץ השחקנים players_in_group השמור באובייקט הקבוצה O(log n).
- אם העץ ריק, נציב null בתוך השדה Highest_level_player של הקבוצה ונוציא את הקבוצה מעץ הקבוצות הפעילות O(log n)
- הצבה של השחקן עם המפתח הגדול ביותר בעץ players_in_group (נמצא אותו על ידי התקדמות ימינה בעץ ככל האפשר) בתוך השדה Highest_level_player של הקבוצה. O(log n).

****

StatusType ***ReplaceGroup*** *(void *DS, int GroupID, int ReplacementID)*

O (log k + ngroup + nreplacement) במקרה הגרוע, כאשר *k* הוא מספר הקבוצות במערכת, *ngroup* מספר השחקנים של הקבוצה שפורקה ו-*nreplacement* הוא מספר השחקנים של הקבוצה המחליפה.

- חיפוש הקבוצה המיועדת למחיקה והקבוצה המחליפה בעץ הקבוצות Groups O(log k)
- הפעלת המתודה merge של העץ players_in_group המקבל את השחקנים, תוך העברת העץ players_in_group שנפטר מהשחקנים. O (ngroup + nreplacement)
- מחיקת הקבוצה הישנה מעץ הקבוצות ומעץ הקבוצות הפעילות O(log k)

****

****

StatusType ***IncreaseLevel*** *(void *DS, int PlayerID, int LevelIncrease)*

O (log n) במקרה הגרוע, כאשר *n* הוא מספר השחקנים.

- בדיקת מזהה השחקן, ערך העלאת השלב ומצביע מבנה הנתונים O(1)
- מציאת השחקן בעץ השחקנים Players O(log n)
- אם הוא לא קיים החזרת שגיאה O(1)
- הסרת השחקן באמצעות המתודה removePlayer והוספתו מחדש עם ערך השלב החדש, אך הפעם נשמור מצביע לקבוצה אליה נרצה להחזיר אותו O(log n)

****

StatusType ***GetHighestLevel*** *(void *DS, int GroupID, int *PlayerID)*

אם *GroupID*>0 אז ***O(1)*** במקרה הגרוע.

אחרת, ***O(log k)*** במקרה הגרוע. כאשר *k* הוא מספר הקבוצות.

- בדיקת מצביע מבנה הנתונים והמצביע לשחקן O(1)
- אם *GroupID*>0 נציב ב- *PlayerID** את מספר השחקן השמור בשדה Highest_level_player של מחלקת האב PlayersManager, אם המערכת ריקה, נחזיר כמספר שחקן את המספר 1- O(1)
- אחרת, נמצא את הקבוצה בעץ הקבוצות Groups O(log k)
  - אם היא לא קיימת, נחזיר שגיאה O(1)
  - אם קיימת וריקה, נחזיר כמספר שחקן את המספר 1-. O (1)
  - אחרת, נחזיר את מספר השחקן ששמור בשדה Highest_level_player של הקבוצה. O (1)

****

StatusType ***GetAllPlayersByLevel*** *(void *DS, int GroupID, int **Players, int *numOfPlayers)*

אם *GroupID*>0 אז ***O(n)*** במקרה הגרוע, כאשר *n* הוא מספר השחקנים במערכת.

אחרת, ***O(n_GroupID+log k)*** במקרה הגרוע, כאשר *n_GroupID* הוא מספר השחקנים ששייכים לקבוצה בעלת המזהה *GroupID* ו-*k* הוא מספר הקבוצות.

- בדיקת מצביע מבנה הנתונים ומצביע מערך השחקנים, ואם GroupID=0 O (1)
- אם GroupID<0 :
  - הקצאת זיכרון דינמי בגודל עץ השחקנים Players ובדיקת הצלחה O(1)
  - נאתחל אובייקט PlayerToArray עם המערך שהקצנו מעלה.

- נבצע מעבר על צמתי העץ Players_by_level באמצעות מתודת reverseInOrder ונעביר אליה את האובייקט מעלה. סה"כ O(numOfGroups+log k) .

- אחרת:
  - נמצא בעץ הקבוצות הפעילות את הקבוצה המתאימה.
  - הקצאת זיכרון דינמי בגודל players_in_group ובדיקת הצלחה O(1)
  - נאתחל אובייקט PlayerToPtrArray עם המערך שהקצנו מעלה.
  - נבצע מעבר על צמתי העץ players_in_group באמצעות מתודת reverseInOrder ונעביר אליה את האובייקט מעלה. סה"כ O(numOfGroups+log k) .

****

$StatusType$ **$GetGroupsHighestLevel$** $(void\ *DS, int\ numOfGroups, int\ **Players)$

O (numOfGroups+log k) במקרה הגרוע, כאשר *numOfGroups* הוא הפרמטר לפונקציה (כמות הקבוצות הלא ריקות להן יש להחזיר את השחקנים בשלב הכי גבוה) ו-*k* הוא מספר הקבוצות.

- בדיקת מצביע מבנה הנתונים ומצביע מערך השחקנים , וכי לא מתקיים numOfGroups<1 O(1)
- בדיקה כי - מספר הקבוצות הפעילות $numOfGroups <$ O(1)
- הקצאת זיכרון דינמי בגודל $numOfGroups$ ובדיקת הצלחה O(1)
- נאתחל אובייקט HighestPlayerToArray עם המערך שהקצנו מעלה. O(1)
- נשתמש במתודת inOrder כדי לעבור על צמתי העץ Active_Groups לפי סדר, ונעביר אליה את האובייקט שאתחלנו ואת $numOfGroups$. סה"כ O(numOfGroups+log k) .

****

void ***Quit*** *(void **DS)*

סיבוכיות מקום – ***O (n+k)*** במקרה הגרוע, כאשר *n* הוא מספר השחקנים ו-*k* הוא מספר הקבוצות

- נהרוס את כלל העצים, כל הרס של עץ מורכב ממעבר על כל הצמתים שלו ולכן סה"כ הסיבוכיות תהיה *O (n+k)*

****