מבני נתונים - תרגיל בית רטוב 1

נתיב מאור 319002911 , מירון דואר 305014243

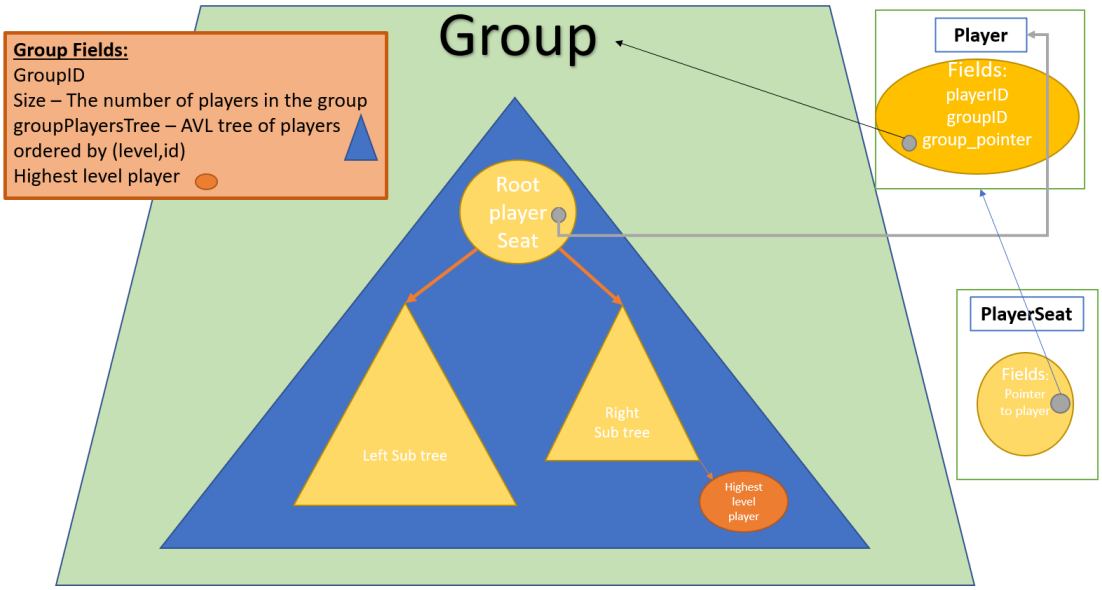
7 בדצמבר 2021

תיאור כללי:

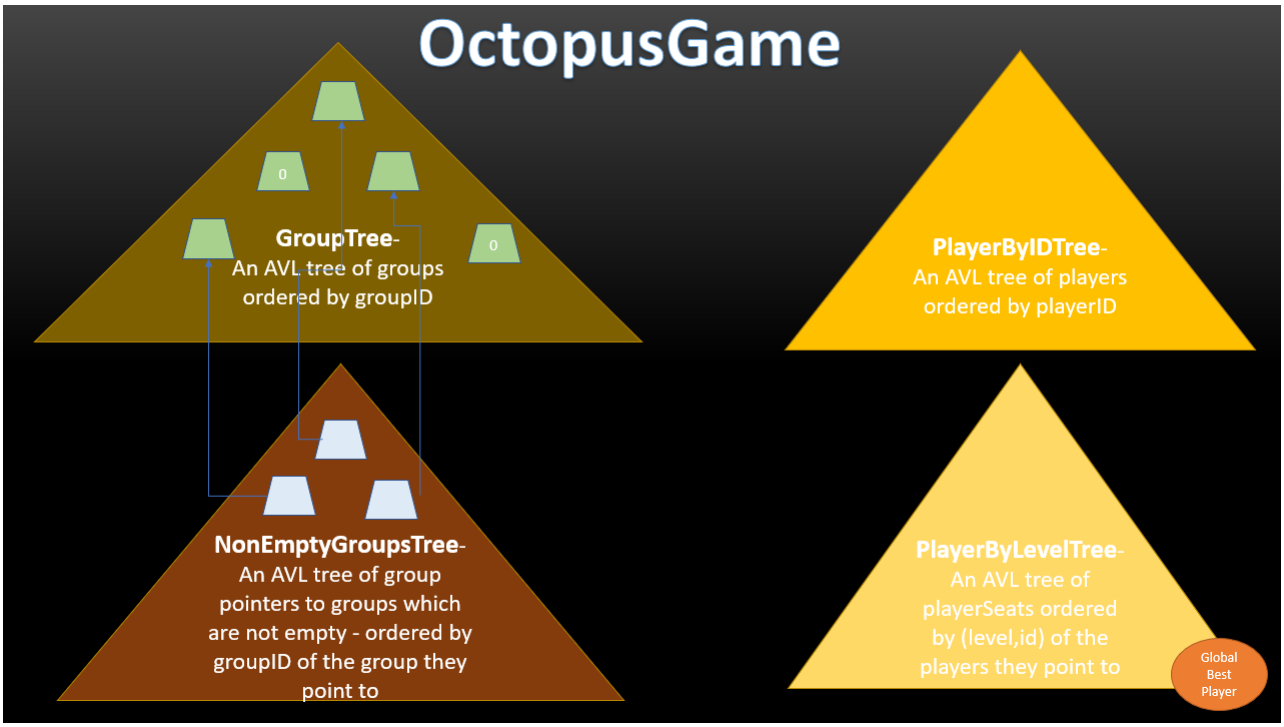
$OctopusGame$ - טיפוס המנהל את משחקי התמנון, מחזיק ארבעה עצים, אחד של השחקנים ממוינים לפי המספר המזהה שלהם, השני של השחקנים ממוינים לפי השלב בו הם נמצאים, השלישי של הקבוצות והרביעי של הקבוצת הלא-ריקות, כמו-כן שומר שדה של השחקן המוביל במשחקים.

תרשימים:

תרשים המתאר את המבנים Group, Player, PlayerSeat



תרשים המתאר את המבנה OctopusGame



מבני עזר:

$Group$ - טיפוס המתאר קבוצה של שחקנים בעל השדות הבאים:

- $GroupID$ - מספר מזהה לקבוצה - מספר שלם.
- $Size$ - גודל הקבוצה - מספר שלם.
- $groupPlayersTree$ - עץ AVL של השחקנים עבור הקבוצה, כאשר המפתחות הינם $PlayerSeat$ והערכים הינם $PlayerSeat$.
- $highestLevelPlayer$ -מצביע חכם לצומת בעץ בעל השלב הגבוה ביותר (אם בעל שלב זהה לשחקן אחר אז בעל ה-$PlayerID$ הנמוך ביותר).

$PlayerSeat$ -

טיפוס המכיל מצביע לשחקן המתאים בעץ השחקנים $PlayerByIDTree$ - נועד לשם מעטפת השוואה בין שלבים של שחקנים, כדי לאפשר לטיפוס להיות מפתח בעץ AVL הדורש יחס סדר.

$Player$ -טיפוס המתאר שחקן בעל השדות הבאים:

- $PlayerID$ - מספר מזהה עבור השחקן - מספר שלם.

- $GroupID$ - מספר מזהה עבור הקבוצה שהשחקן נמצא בה - מספר שלם.
- $Level$ - השלב בו נמצא השחקן - מספר שלם.
- $group$ - מצביע לקבוצה מטיפוס $Group$ בה נמצא השחקן.

עבור השחקן יוגדר היחס סדר הבא:

- נסמן שני שחקנים p_1, p_2:
  - $p_1 > p_2$ אם מתקיימים אחד מהתנאים הבאים
    - ($p_1.level > p_2.level$)
    - ($p_1.level = p_2.level$ וגם $p_1.PlayerID < p_2.PlayerID$)
- אופרטורי השוואה אחרים יוגדרו באופן דומה.

$AVL\_Tree < KEY, VAL >$ - עץ AVL :

גנרי דו כיווני (הורה מצביע לילד וילד מצביע להורה) כפי שנלמד בהרצאה.
הצמתים ממוינים לפי ערך המפתח ומכילים את המידע .
הפעולות המוגדרות וסיבוכיות הזמן והמקום של המבנה כפי שנלמדו בכיתה.

$List < T >$ - רשימה מקושרת דו כיוונית.

אברי הרשימה הם טיפוס $ListNode < T >$, המכיל פוינטר חכם ל ערך T ,מצביע לאיבר לפני ברשימה ומצביע לאיבר אחרי ברשימה.
הפעולות המוגדרות וסיבוכיות הזמן והמקום של המבנה כפי שנלמדו בכיתה.
השימוש במבנה הינו עבור פעולת איחוד העצים.

$Pair < Key, Value >$ - צמד.

נועד לשלב ביניים כדי לייצג צומת של עץ במעבר בין עץ למערך בפעולה של איחוד העצים.

$DynamicArray < T >$ - מערך.

מימוש המערך שנועד לשימוש כחלק מהפעולה של איחוד עצים, לניהול נוח של הזיכרון.
הפעולות המוגדרות וסיבוכיות הזמן והמקום של המבנה כפי שנלמדו בכיתה.
הערה: לא נעשה שימוש בחלק הדינאמי בתרגיל.

שדות המבנה $OctopusGame$:

- $PlayerByIDTree$ - עץ AVL שמכיל את השחקנים במשחק (טיפוס מסוג $Player$) ממוין לפי המספרים המזהים של השחקנים $PlayerID$.
- $GroupTree$ - עץ AVL שמכיל את הקבוצות במשחק (טיפוס מסוג $Group$) ממוין לפי המספרים המזהים של הקבוצות $GroupID$.

- $PlayerByLevelTree$ - עץ AVL שמכיל את מצביעים לשחקנים במשחק (טיפוס מסוג $PlayerSeat$) ממוין לפי זוג סדור של $(PlayerID, Level)$ המתאימים לשחקן המוצבע, כאשר המיון הוא קודם לפי $level$ בסדר עולה ולאחר מכן לפי $PlayerID$ בסדר יורד. כאשר הערך הכי ימני יישמר בשדה $GlobalBestPlayer$.
- $NonEmptyGroupsTree$ - עץ AVL שמכיל מצביעים לקבוצות **הלא ריקות** בעץ הקבוצות $GroupTree$, אשר ממוין לפי $GroupID$.
- $GlobalBestPlayer$ - מצביע לשחקן בעל ה-$Level$ הכי גבוה ובעל ה-$PlayerID$ הכי נמוך לאותו שלב.

מימוש פעולות:

$Init\ ()$:

- אתחול של השדות הבאים:
  - $PlayerByIDTree$ - אתחול של עץ ריק - סיבוכיות של $O(1)$
  - $GroupTree$ - אתחול של עץ ריק - סיבוכיות של $O(1)$
  - $PlayerByLevelTree$ - אתחול של עץ ריק - סיבוכיות של $O(1)$
  - $NonEmptyGroupsTree$ - אתחול של עץ ריק - סיבוכיות של $O(1)$
  - $GlobalBestPlayer$ - אתחול של משתנה $Player$ עם $Null$ - סיבוכיות של $O(1)$

סיבוכיות:
מפני שכל הפעולות הינן מסדר של $O(1)$ הרי שהסיבוכיות הכוללת הינה $O(1)$ כנדרש.

$AddGroup\ (void*\ DS, int\ GroupID)$:

- הוספת קבוצה ($Group$) ריקה בעלת מספר מזהה $GroupID$ לעץ הקבוצות ($GroupTree$) - סיבוכיות של $O(\log k)$

סיבוכיות:
הסיבוכיות הכוללת תהיה $O(\log k)$ כנדרש.

$AddPlayer\ (void*\ DS, int\ PlayerID, int\ GroupID, int\ Level)$:

- הוספת שחקן $Player$ חדש למערכת:
  - יוכנס שחקן $Player$ לעץ השחקנים $PlayerByIDTree$ - סיבוכיות $O(\log n)$.
  - יוכנס מצביע לשחקן $Player$ אשר ימוין בעץ $PlayerByLevelTree$ לפי זוג סדור $(PlayerID, Level)$ כאשר $Player.level$ יאותחל ל-$Level$ שהוכנס - סיבוכיות $O(\log n)$.
  - יוכנס $PlayerSeat$ לעץ הקבוצות $GroupTree$ כאשר קודם יהיה חיפוש של קבוצת השחקן לפי $GroupID$ (סיבוכיות $O(\log k)$) ועוד הכנסת ה-$PlayerSeat$ לפי $PlayerID$ לקבוצה $Group$ המתאימה (סיבוכיות $O(\log n)$) - סה"כ סיבוכיות $O(\log n + \log k)$.
  - אם הקבוצה שאליה הוכנס השחקן הייתה ריקה לפני הכנסתו:
    - נכניס את הקבוצה אליה הוא הוכנס לעץ $NonEmptyGroupsTree$ - סיבוכיות $O(\log k)$.
  - חיפוש השחקן בעל השלב הגבוה ביותר בעץ $GroupPlayersTree$ בקבוצה של השחקן (הצומת הכי ימני) ועדכון השדה $HighestLevelPlayer$ - סיבוכיות $O(\log n)$

- חיפוש השחקן בעל השלב הגבוה ביותר בעץ $PlayerByLevelTree$ (הצומת הכי ימני) ועדכון השדה $GlobalBestPlayer$- סיבוכיות $O(\log n)$

סיבוכיות:
הסיבוכיות הכי גבוהה הינה $O(\log n + \log k)$ ולכן גם הסיבוכיות הכוללת תהיה $O(\log n + \log k)$ כנדרש.

$RemovePlayer\ (void *\ DS, int\ PlayerID, int\ GroupID)$:

- הסר שחקן $Player$ מן המערכת
  - נחפש את השחקן בעץ השחקנים $PlayerByIDTree$ - סיבוכיות $O(\log n)$.
  - אם השחקן נמצא:
    - השחקן $Player$ יוסר מעץ השחקנים $PlayerByIDTree$ - סיבוכיות $O(\log n)$.
    - יוסר המצביע לשחקן $Player$ בעץ $PlayerByLevelTree$ ישירות ממצביע שקיים ב-$PlayerByIDTree$ - סיבוכיות $O(\log n)$.
    - יוסר $PlayerSeat$ מעץ השחקנים בקבוצה בעלת $GroupID$ המתאים שהינו $GroupPlayersTree$ לכך תהיה סיבוכיות של $O(\log n_{group})$ כאשר n_{group} הינו מספר השחקנים בקבוצה. - סה"כ סיבוכיות של $O(\log(n))$ (כי $n_{group} \leq n$).
    - אם הקבוצה $Group$ הפכה לריקה, נסיר אותה מ- $NonEmptyGroupsTree$ בסיבוכיות של $O(\log k_{non-empty})$ ולכן מפני שישנו לפחות שחקן אחד בכל קבוצה לא ריקה הרי ש-$k_{non-empty} \leq n$ ולכן נסיר אותה בסיבוכיות של $O(\log(n))$.
    - חיפוש השחקן בעל השלב הגבוה ביותר בעץ $GroupPlayersTree$ בקבוצה של השחקן (הצומת הכי ימני) ועדכון השדה $HighestLevelPlayer$ - סיבוכיות $O(\log n)$
    - חיפוש השחקן בעל השלב הגבוה ביותר בעץ $PlayerByLevelTree$ (הצומת הכי ימני) ועדכון השדה $GlobalBestPlayer$- סיבוכיות $O(\log n)$

סיבוכיות:
הסיבוכיות הכי גבוהה הינה $O(\log n)$ ולכן גם הסיבוכיות הכוללת תהיה $O(\log n)$ כנדרש.

$ReplaceGroup\ (void * DS, intGroupID, intReplacementID)$:

נסמן את הקבוצה אותה יש להסיר ב-g_{remove} ואת הקבוצה שאליה צריך להוסיף את השחקנים $g_{replacement}$

- חיפוש הקבוצה g_{remove} ב-$GroupTree$ - סיבוכיות $O(\log k)$
- שחרור עץ השחקנים של g_{remove} באמצעות סיור Post–Order - סיבוכיות $O(n_{group})$.
- תוך כדי שחרור העץ נבצע שמירת שחקנים ב- g_{remove} ברשימה זמנית - $O(n_{group})$.
- חיפוש $g_{replacement}$ ב-$GroupTree$ כאשר הסיבוכיות $O(\log k)$ ואז איחוד הקבוצה עם g_{remove} באמצעות התהליך של איחוד עצים אותו ראינו בתרגול 4 בו הופכים את העצים לרשימות ממוינות ואז מאחדים לעץ אחד בסיבוכיות $O(n_{group} + n_{replacment})$ - מכאן הסיבוכיות סה"כ תהיה $O(\log k + n_{group} + n_{replacment})$.

סיבוכיות:
הסיבוכיות הכי גבוהה הינה $O(\log k + n_{group} + n_{replacment})$ ולכן גם הסיבוכיות הכוללת תהיה $O(\log k + n_{group} + n_{replacment})$ כנדרש.

$IncreaseLevel\ (void * DS, intPlayerID, intLevelIncrease)$:

נסמן את השחקן אותו נרצה לעלות בדרגה ב-P_{level}.

- נחפשו בעץ השחקנים $PlayerByIDTree$ - סיבוכיות של $O(\log n)$.
- אם השחקן קיים בעץ (אחרת נחזיר שגיאה):
- בעץ $PlayerByLevelTree$:
  - בעזרת השחקן שמצאנו בעץ $PlayerByIDTree$ נמצא את הצומת המתאים בעץ עבור אותו שחקן (לשם החיפוש נדרש השלב של השחקן - $Level$) - סיבוכיות $O(\log n)$
  - נסיר את הצומת המתאים מן העץ תהיה סיבוכיות $O(\log n)$.
- מן המצביע של השחקן לעץ השחקנים של הקבוצה בה נמצא השחקן $GroupPlayersTree$
  - בעזרת השחקן שמצאנו בעץ $PlayerByIDTree$ נמצא את הצומת המתאים בעץ עבור אותו שחקן (לשם החיפוש נדרש השלב של השחקן - $Level$) - סיבוכיות $O(\log n)$
  - נסיר את הצומת המתאים מן העץ תהיה סיבוכיות $O(\log n)$ כי גודל הקבוצה קטן או שווה ל-n.
- נעדכן את השלב בו השחקן $Player$ המתאים נמצא - $Level + LevelIncrease$ - סיבוכיות $O(1)$.
- נוסיף את ה-$PlayerSeat$ המתאים ל-$PlayerByLevelTree$ - סיבוכיות $O(\log n)$.
- נוסיף את ה-$PlayerSeat$ המתאים ל-$GroupPlayersTree$ המתאים לקבוצה של השחקן- סיבוכיות $O(\log n)$.

סיבוכיות:
הסיבוכיות הכי גבוהה הינה $O(\log n)$ ולכן גם הסיבוכיות הכוללת תהיה $O(\log n)$ כנדרש.

$GetHighestLevel\ (void * DS, intGroupID, int * PlayerID)$:

- אם $GroupID < 0$:
  - להחזיר את ה-$PlayerID$ של השחקן בשדה $GlobalBestPlayer$.
- אחרת אם $GroupID > 0$
  - חיפוש הקבוצה בעלת ה-$GroupID$ המבוקש בעץ $GroupTree$ והחזרת מזהה השחקן ב-$highestLevelPlayer$ - סיבוכיות $O(\log k)$.

סיבוכיות:
הסיבוכיות הכי גבוהה הינה $O(\log k)$ ולכן גם הסיבוכיות הכוללת תהיה $O(\log k)$ כנדרש.

$GetAllPlayersByLevel\ (void * DS, int\ GroupID, int * *Players, int * numOfPlayers)$:

- אם $GroupID < 0$:
  - החזרת כל השחקנים בעץ $PlayerByLevelTree$ לפי הסדר של הזוג הסדור של השלבים לפי In–order. - סיבוכיות $O(n)$.
- אחרת: (מתקבלת סה"כ סיבוכיות של $O(n_{group} + \log k)$)

- חיפוש הקבוצה בעלת ה-$GroupID$ המבוקש בעץ $GroupTree$ - סיבוכיות $O(\log k)$.
- החזרת כל השחקנים השייכים ל-$groupPlayersTree$ של הקבוצה לפי הסדר של הזוג הסדור של השלבים לפי In–order ב**סדר הפוך** כלומר מימין לשמאל - סיבוכיות $O(n_{group})$.

סיבוכיות:
עבור המקרה של $GroupID < 0$ הסיבוכיות $O(n)$ כנדרש.
עבור המקרה של $GroupID > 0$ הסיבוכיות $O(n_{group} + \log k)$ כנדרש.

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

- נמצא את $numOfGroups$ בעלי השחקנים הכי גבוהים באופן הבא : (מתקבלת סה"כ סיבוכיות של $O(numOfGroups + \log k)$)
  - נמצא את הצומת הכי ימני בעץ $NonEmptyGroupsTree$ - סיבוכיות $O(\log k)$.
  - מבצעים סיור Reverse-Climb-In-Order (כמו In-Order רק מתחיל מהצומת הימני ומבצע סיור על העץ רק על $NumOfGroups$ הצמתים אותם אנו רוצים להחזיר, בתוספת של לכל היותר $O(logk)$ צמתים - כגובה העץ) כלומר מהצומת הכי ימני נבצע סיור לקבל את ה-$numOfGroups$ צמתים שבאים לפני הצומת הכי ימני בעץ $NonEmptyGroupsTree$ - סיבוכיות $O(numOfGroups)$

סיבוכיות:
הסיבוכיות הינה $O(numOfGroups + \log k)$ כנדרש.

$Quit\,(void **DS)$:

מתבצע ע"י ההורס של $OctopusGame$ שלשדות שלו הורסים מוגדרים.

- שחרור עץ $PlayerByIDTree$ על-ידי סיור Post-Order על העץ - סיבוכיות $O(n)$
- שחרור עץ $PlayerByLevelTree$ על-ידי סיור Post-Order על העץ - סיבוכיות $O(n)$
- שחרור עץ $GroupTree$ על-ידי סיור Post-Order על העץ וכל צומת בעץ נשחרר את $Group$ המתאים בשחרור העץ $groupPlayersTree$ המתאים לו (מתקיים $\sum_{i=1}^{k} n^i_{group} = n$) - סיבוכיות $O(k+n)$
- שחרור רשימת $NonEmptyGroupsList$ - סיבוכיות $O(k_{non-empty})$.

סיבוכיות:
מתקיים $n^i_{group} \leq n$ לכל i כי סכום סך האיברים בסך הקבוצות הינו n מכאן ששחרור כל הקבוצות $Group$ יהיה $O(n)$
ולכן שחרור העץ $GroupTree$ יהיה $O(n+k)$.
כמו כן $k_{non-empty} \leq k$ ולכן יהיה גם בסיבוכיות $O(k)$.
מכאן שהסיבוכיות הגבוהה ביותר הינה $O(n+k)$ ולכן הסיבוכיות הכוללת $O(n+k)$ כנדרש.

סיבוכיות מקום:

- $PlayerByIDTree$ - מכיל n צמתים כל צומת בגודל קבוע $O(1)$ - סיבוכיות מקום $O(n)$
- $GroupTree$ - מכיל k צמתים אשר כל צומת i מכיל עץ בעל n^i_{group} צמתים ולכן סה"כ $\sum_{i=1}^{k} n^i_{group} = n$ - סיבוכיות מקום $O(k+n)$

- $PlayerByLevelTree$ - מכיל n צמתים כל צומת בגודל קבוע $O(1)$ - סיבוכיות מקום $O(n)$
- $NonEmptyGroupsTree$ - מכיל לכל היותר k צמתים בגודל $O(1)$ (כל צומת מכיל מצביע) - סיבוכיות מקום $O(k)$
- $GlobalBestPlayer$ - מספר שלם - סיבוכיות מקום $O(1)$
- הערה: ישנן פעולות רקורסיביות בהן השתמשנו (כמו פעולות על העץ) אשר סיבוכיות המקום שלהן לא עולה על $O(k+n)$.

סיבוכיות המקום:
הסיבוכיות מקום הגבוהה ביותר הינה $O(k+n)$ ולכן הסיבוכיות מקום הכוללת הינה $O(k+n)$, כנדרש.