מבני נתונים

גיליון רטוב 2 - חלק יבש

יונתן רייכר	יונתן שטרנברג
214762718	214762338

מבני הנתונים:

לפני שנסביר את מבנה הנתונים הכולל נתאר את העץ המיוחד שנשתמש בו

עץ:
העצים שלנו יהיו עצי AVL ranked שיתמכו בכל הפעולות שראינו בהרצאות ובנוסף נבצע עליהם שני הרחבות.

1. שדות נוספים לצומת:
השדה ראשון יקרא addWins וישמש אותנו לניהול מספר הניצחונות בכל קבוצה.
השדה השני יקרא maxRank ויהיה הדירוג המקסימלי בתת עץ של הצומת ביחס ל-addWins.
(במבנה הנתונים, נשתמש בתכונות הללו רק בעץ המכיל קבוצות)
השדות עצמם מוסיפים מקום בגודל קבוע לכל צומת ולכן **לא משפיעים על סיבוכיות המקום** של העץ.
תחזוק addWins: בכל סיור בעץ נסכום את ערכי ה-addWins בענף (רקורסיבית). מספר הניצחונות של קבוצה מסוימת היא סכום ה-addWins שלה ושל הוריה, אז כאשר נוסיף קבוצה חדשה נאתחל את addWins שלה להיות מינוס מספר הנצחונות שסכמנו עבור האב שלה. כך החישוב של מספר הנצחונות שלה יתחיל מלהחזיר 0.
כאשר נוריד צומת מהעץ, נוסיף את ה-addWins לבניו.
בגלגולים, נחשב קודם לכן את מספר הנצחונות באותו אופן רקורסיבי *משורש הגלגול*. את הערכים הללו נשמור לכל צומת שמוערב בגלגול.
לאחר הביצוע של הגלגול, נבחר addWins לכל צומת שמעורב בגלגול ששווה לערך ששמרנו עבורו פחות addWins של ההורים שלו עד שורש הגלגול (ב-preorder). כך נשמור על גלגולים בסיבוכיות $O(1)$.
תחזוק maxRank: בכל שינוי של addWins בצומת X, או ניתוק או חיבור של בן של X או שינוי ב-maxRank של בן של X, או כאשר מוסיפים את X לעץ, ניתן מחדש ל-maxRank של X את הערך $addWins + max\{str, maxRank1, maxRank2\}$ כאשר str הוא הכוח השחקן החציוני בקבוצה (זמן קבוע לחישוב לפי השדה למטה) כפול מספר השחקנים בקבוצה, וה-maxRank1/2 הם הערכים בבנים של הצומת.
סיבוכיות:
התחזוקים של שני השדות האלו יקרו רק בסיורים מתי שכל המידע הנחוץ לחישובים זמין, והחישובים עצמם הם בזמן קבוע. לכן שני השדות **לא משפיעים על סיבוכיות** הזמן של העץ.

2. שדות נוספים לעץ:
maximum, minimum, middle יהיו שדות שישמרו את הצומת הימני ביותר, השמאלי ביותר, וזה שבאינדקס החציוני בעץ. נתחזק אותם בכך **שבכל פעם שנוסיף או נוציא** איבר מהעץ, נחפש את הצמתים הללו ונשמור אותם.
חיפושים אלו הם $O(\log n)$ סיבוכיות זמן והשדות עצמם הם בגודל קבוע ולכן הם $O(1)$ סיבוכיות מקום אז השדות האלו **לא ישנו את הסיבוכיות הזמן/מקום של העץ**.

ניצור גם פונ' עזר - $addWinsInRange(int\ i,\ int\ j)$**.**
פונ' זו מקבלת טווח [i:j] של אינדקסים של קבוצות* ומוסיפה להם ניצחון אחד (כולל הקצוות).
*אינדקס של קבוצה הוא מיקומה במערך הקבוצות עם היינו מסדרים אותן במערך ממויין.
נממשה באופן הבא:

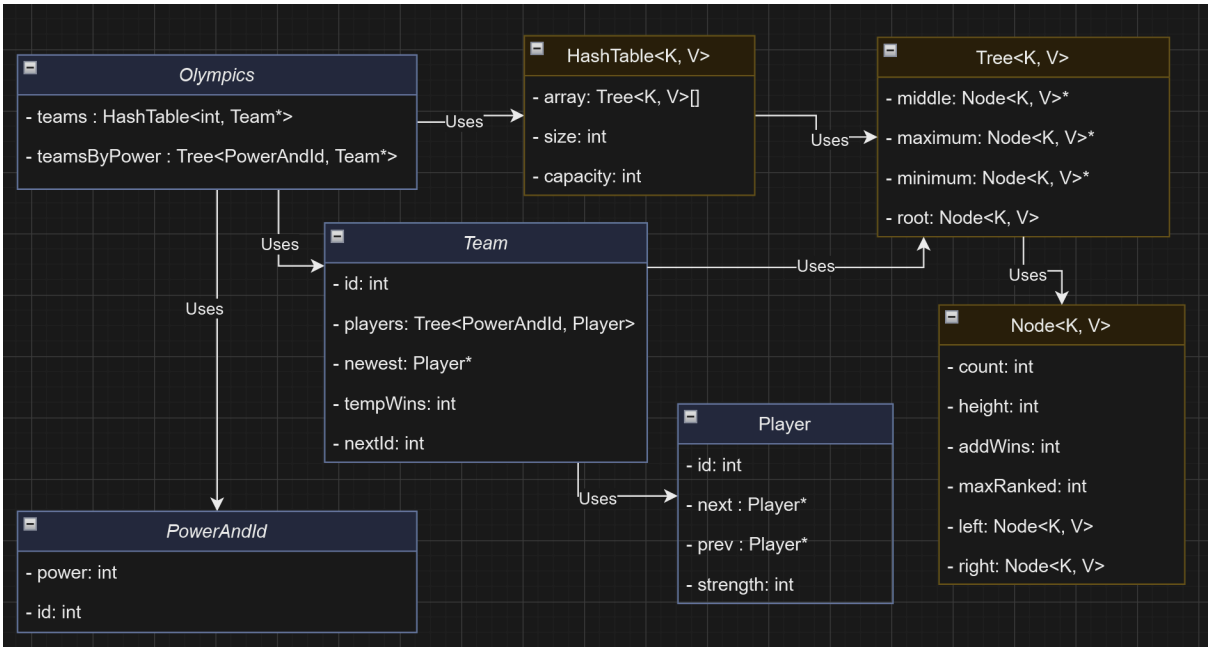
- ניצור עוד פונ' עזר $addWins(int\ i,\ int\ winsNum)$ אשר מוסיפה $winsNum$ ניצחונות לכל הקבוצות עד הקבוצה במקום ה-i. נעשה זאת ע"י סיור עד האיבר ה-i בעץ

ה-teamsByPower (בין אם קיים או לא) - $O(\log n)$. נגדיר דגל הוספה (שמתחיל כבוי). בכל צומת, אם i גדול ממנו או שווה לו ודגל ההוספה לא דולק, נוסיף לשדה הניצחונות של הצומת $winsNum$, נמשיך בסיור ונדליק את דגל ההוספה. אם i קטן מהצומת ודגל ההוספה דולק, נוריד $winsNum$ משדה הניצחונות של הצומת ונכבה את דגל ההוספה. כך נמשיך הלאה עד סוף הסיור.

- כעת, נפעיל $addWins(i-1,-1) + addWins(j,1)$ ונקבל שאכן הוספנו ניצחון 1 לקבוצות אם כוח X כך ש: $i-1 < X <= j$, כלומר $i <= X <= j$.

סה"כ ביצענו 2 פעולות בסיבוכיות $O(\log n)$, ולכן $addWinsInRange$ עצמה היא גם בסיבוכיות $O(\log n)$.

מבנה הנתונים:



<u>הסבר על PowerAndId ועל השוואת מפתחות:</u>

המבנה PowerAndId נועד בשביל ליצור מפתחות לשחקנים ולקבוצות שיעזרו לנו בסיבוכיות של play_tournament. השוואת PowerAndId הולכת לפי הדרישות ניצחון ב-play_match ו-play_tournament - כלומר, עבור שני אובייקטים a ו-b, הגדול מבינהם הוא זה עם power גדול יותר, ואם יש תיקו ב-power אז לפי id *קטן* יותר!

כך, ניצחון ב-play_tournament נקבע לפי מי ממוקם ימינה יותר בעץ teamsByPower מה שנותן לנו לבצע את הפעולה בסיבוכיות הנכונה על ידי שמירת כמות הנצחונות בתוך התכונה addWins שתיארנו מעלה.

בעץ teamsByPower יהיו ערכי הכח והמזהה של הקבוצה ובעץ players הם יהיו ה-strength של השחקן והמזהה שלו.

<u>פונקציות:</u>

<u>[1] - חישוב כוח קבוצה</u>

כוח של קבוצה מוגדר להיות מכפלת כוח השחקן החציוני (שדה middle של players) בכמות האנשים בעץ.

שם הפונקציה	מימוש	סיבוכיות זמן/מקום
olympics_t()	אתחול עץ ריק וטבלת ערבול ריקה.	$O(1)$
virtual ~ olympics_t()	מחיקת כל עצי השחקנים - $O(k_{players\ in\ team})$ לכל קבוצה סך הכל $O(k)$. מחיקת עץ הקבוצות - $O(n)$ מחיקת כל הקבוצות דרך מחיקת טבלת הקבוצות - $O(n)$	סך הכל: $O(n + k)$
StatusType add_team(int teamId)	אם teamId≤0 נחזיר INVALID_INPUT. נבדוק האם הקבוצה נמצאת בטבלת הקבוצות - $O(1)$ משוערך (חיפוש בטבלת HASH), אם כן נחזיר FAILURE. אחרת, ניצור קבוצה חדשה עם עץ ריק, מצביע ל-null ו-nextId של 1 ו-tempWins של 0 ונוסיף אותה לטבלת הקבוצות - $O(1)$ משוערך.	**זמן:** $O(1)$ משוערך **מקום:** יוצרים קבוצה - $O(1)$ הוספת איבר לטבלה - $O(log(n))$* * - במקרה הגרוע נעבור ברקורסיה על עץ עם n צמתים. סה"כ - $O(log(n))$
StatusType remove_team(int teamId)	אם teamId≤0 נחזיר INVALID_INPUT. נבדוק האם הקבוצה נמצאת בטבלת הקבוצות - $O(1)$ משוערך (חיפוש בטבלת HASH), אם לא נחזיר FAILURE. אחרת, נמחק את כל עץ המשתתפים - $O(k_{playersInTeam})$ ונמחק נמצא את הקבוצה בעץ הקבוצות (teamsByPower) ב - $O(log(n))$ (רק אם היא נמצאת בו). ונמחק אותה מטבלת הקבוצות - $O(1)$ משוערך.	**זמן:** $O(k_{players\ in\ team} + log(n))$ **מקום:** לא הקצנו מקום למשתנים חדשים. לכן $O(\log n)$ עבור הפעולות בעצים ובטבלה.
StatusType add_player(int teamId, int playerStrength)	נבדוק אם teamId או playerStrength אי חיוביים. אם לא, נחזיר INVALID_INPUT. נחפש את את הקבוצה במזהה teamId בהאשטאבל. אם הקבוצה לא נמצאה, נחזיר FAILURE. נניח כמו שרשום בתרגיל שהשחקן לא קיים בקבוצה. ניצור אובייקט שחקן חדש: נאתחל את prev שלו ל-newest ואת next ל-null. כוחו יהיה playerStrength. ה-id שלו יהיה nextId. נוסיף את השחקן תחת playerStrength ו-nextId לעץ players של הקבוצה שמצאנו. נוסיף +1 ל-nextId של הקבוצה. אם קיים newest, נעדכן את newest.next להיות השחקן החדש הנ"ל. נעדכן את newest להיות השחקן החדש. <u>כעת נעדכן גם את עץ הקבוצות:</u> אם הקבוצה נמצאת בעץ, נוציא אותה. (אם היא לא נמצאת בו, היא נמצאת בכל מקרה בטבלה וכבר יש לנו אותה ממקודם). כעת נחשב מחדש את הכוח של הקבוצה לפי [1]. נוסיף את הקבוצה אל עץ הקבוצות לפי הכוח החדש והמזהה teamId.	**זמן:** חיפוש בהאשטבל - מקרה הגרוע ביותר $O(\log n)$ הוצאה/הכנסה לעץ - $O(\log k)$ כל שאר הפעולות המתוארות הן בזמן קבוע לכן סך הכל: $O(\log n) + 2O(\log k) + O(1)$ $= O(\log n + \log k)$ **מקום:** כל שחקן נשמר פעם אחת בקבוצה שלו ואך ורק שם. בנוסף, הקבוצה שאנחנו מוסיפים לעץ

	אם כעת יש שחקן יחיד בקבוצה (כלומר היא הייתה ריקה לפני ההוספה) נעדכן את ה-addWins שלה כך שנקבל בחישוב הניצחונות שלה את הערך ששמרנו בשדה tempWins.	הקבוצות נשמרת בו אך ורק פעם אחת. עבור הפעולות בעצים ובטבלה - $O(\log n)$ סה"כ - $O(\log n)$.
StatusType remove_newest_player(int teamId)	נתחיל מלבדוק האם $teamId \leq 0$. אם כן, נחזיר INVALID_INPUT. נחפש את הקבוצה לפי המזהה בהאשטאבל. אם לא מצאנו, נחזיר או אם היא ריקה נחזיר FAILURE. נחפש את הקבוצה בעץ הקבוצות. נחשב את כמות הניצחונות שלה - נסכום את ערכי addWins של כל הצמתים שעברנו עליהם. נסמן את זה ב-$wins$. כעת הקבוצה לא ריקה אז נרצה להדיח את newest. נבצע הוצאה מהעץ players לפי newest.strength ו-newest.id. נוציא את הקבוצה מעץ הקבוצות. אם הקבוצה ריקה לאחר הוצאת השחקן, נעדכן: $newest \leftarrow nullptr$. אם היא לא ריקה, נחזיר אותה לעץ מחדש עם הכוח החדש. בנוסף, נבצע $newest \leftarrow newest.prev$ ואז לאחר מכן $newest.next \leftarrow nullptr$. אם הקבוצה כן ריקה לאחר ההוצאה, נבצע $tempWins \leftarrow wins$	**זמן:** חיפוש בהאשאבל (מקרה גרוע ביותר) - $O(\log n)$ הוצאה מעץ המשתתפים - $O(\log k)$ הוצאה מעץ הקבוצות - $O(\log n)$ שאר הפעולות הן בזמן קבוע לכן בסך הכל $O(\log n + \log k)$ **מקום:** הפונקציה הזו לא מקצה מקום כלל ומנקה שחקנים שיוצאים מהקבוצה - $O(1)$. עבור הפעולות בעצים ובטבלה - $O(\log n)$ סה"כ - $O(\log n)$.
output_t<int> play_match(int teamId1, int teamId2)	אם teamId1 = teamId2 או teamId1 ≤ 0,teamId2 ≤ 0 נחזיר INVALID_INPUT. נחפש את שתי הקבוצות בטבלת הקבוצות - $O(log(n))$ אם אחת מהן לא קיימת או שאחת מהן ריקה נחזיר FAILURE. נחשב את כוחות הקבוצות לפי [1]. נשווה את כוחות הקבוצות: הקבוצה המנצחת תהיה זו עם כח גדול יותר (או במקרה של תיקו, זו עם id קטן יותר). נמצא את האינדקס של הקבוצה בעץ הקבוצות. נבצע $addWinsInRange(i, i)$ כאשר i הוא האינדקס שמצאנו. נחזיר את אינדקס הקבוצה המנצחת!	**זמן:** $2 * log(n)$ - 2 חיפושים בעץ הקבוצות $log(n)$ - $addWinsInRange$. סה"כ - $log(n)$. **מקום:** לא הקצנו מקום למשתנים חדשים, ועבור הפעולות בעצים ובטבלה - $O(\log n)$ סה"כ - $O(\log n)$.
output_t<int> num_wins_for_team(int teamId)	אם teamId ≤ 0 נחזיר INVALID_INPUT. נחפש את הקבוצה בטבלה הקבוצות - $O(log(n))$ אם לא קיימת כזו, נחזיר FAILURE. אחרת, אם הקבוצה לא נמצאת בעץ, נחזיר tempWins. אחרת, נסייר על עץ ה-teamsByPower עם ערך הכוח של הקבוצה והמזהה שלה - $O(log(n))$ נסכום את כל ערכי שדות הניצחון של הצמתים במסלול ונקבל	**זמן:** חיפוש - $O(log(n))$ וסיור בעצי קבוצות **מקום:** לא הקצנו מקום למשתנים חדשים ועבור הפעולות בעצים ובטבלה

	את מספר הניצחונות של הקבוצה.	- $O(\log n)$ סה"כ - $O(\log n)$.
output_t<int> get_highest_ranked_team()	אם טבלת הקבוצות שלנו ריקה - אין לנו קבוצות במערכת ולכן נחזיר -1. אחרת, נבדוק האם יש איברים בעץ ה-teamsByPower, אם לא, אזי כל הקבוצות ריקות ולכן נחזיר 0. אחרת, נחזיר את שדה maxRank של השורש - זהו הדירוג הגבוהה ביותר בעץ.	**זמן:** כמות סופית של פעולות בזמן קבוע - $O(1)$ **מקום:** אין שימוש במשתנים כלל וכל הפעולות הם בזיכרון קבוע, לכן $O(1)$
StatusType unite_teams(int teamId1, int teamId2)	אם $teamId1 \le 0$, $teamId2 \le 0$ או $teamId1 = teamId2$ אז נחזיר INVALID_INPUT נחפש את שתי הקבוצות לפי המזהים שלהם בטבלת הערבול. אם אחת לא נמצאת, נחזיר FAILURE. נחשב את כח הקבוצה לפי [1] ונסמנו oldPower. <u>ניצור עץ שחקנים חדש עבור קבוצה 1:</u> 1. נכתוב את עץ קבוצה 1 למערך ממוין 2. נכתוב את עץ קבוצה 2 למערך ממוין 3. במערך השני, נוסיף את nextId של קבוצה 1 לכל המזהים של המשתתפים במערך קבוצה 2, ולמזהים ששמורים במפתחות שלהם. 4. נמזג את המערכים הממוינים לעץ אחד $O(k_{players\ in\ team1} + k_{players\ in\ team2})$ $+\ O(k_{players\ in\ team1})\ +\ O(k_{players\ in\ team2})$ זה יהיה העץ השחקנים החדש של קבוצה 1. נוסיף ל-nextId של קבוצה 1 ל-nextId של קבוצה 2. <u>בנוסף, נמזג את שני רשימות השחקנים כך:</u> אם newest1 מצביע ל-null אז $newest1 \leftarrow newest2$ אם newest2 מצביע ל-null אז לא צריך לעשות דבר *אחרת:* • $newest1.next \leftarrow oldest2$ • $oldest2.prev \leftarrow newest1$ • $newest1 \leftarrow newest2$ כאן newest1 זה שדה newest של הקבוצה הראשונה, newest2 זה שדה newest של הקבוצה השניה ו-oldest2 הוא השחקן הישן ביותר בקבוצה 2. על מנת למצוא את oldest2 נבצע סריקה לינארית מ-newest2 אחורית על ידי prev עד שנגיע לאיבר עם prev = nullptr - $O(k_{players\ in\ team2})$. נמחק את שני עצי השחקנים הישנים - $O(k_{players\ in\ team1})\ +\ O(k_{players\ in\ team2})$ נוציא את הקבוצה השניה מעץ הקבוצות ומהטבלה - $2O(\log n)$.	**זמן:** ביצענו כמות קבועה של פעולות הלוקחות סיבוכיות זמן של $O(k_{players\ in\ team1} + k_{players\ in\ team2} + log(n))$ או פחות, אז סך הכל הסיבוכיות זמן היא $O(k_{players\ in\ team1} + k_{players\ in\ team2} + log(n))$ **מקום:** הקצאת זיכרון עבור המערכים - $O(k_1 + k_2)$ עבור הפעולות בעצים ובטבלה - $O(\log n)$. סך הכל: $O(k_{players\ in\ team1} + k_{players\ in\ team2} + log(n))$

	נבדוק אם הקבוצה בעץ הקבוצה $O(\log n)$ לפי המזהה שלה ו-oldPower. אם כן, נוציא אותה מהקבוצה. נבדוק את אורך הקבוצה אינו 0, אז נכניס אותה מחדש לעץ הקבוצות לפי המזהה שלה והכח שלה (שנחשב מחדש לפי [1]).	
output_t<int> play_tournament(int lowPower, int highPower)	אם $lowPower \leq 0$ או $highPower \leq 0$ או $highPower \leq lowPower$ נחזיר כמובן INVALID_INPUT! בעץ הקבוצות, נמצא את האינדקס של האיבר הראשון עם מפתח גדול-שווה ל-$PowerAndId(lowPower, \infty)$ ואת האינדקס של האיבר האחרון עם מפתח קטן-שווה ל-$PowerAndId(highPower, -\infty)$ (נסמן אותם low ו-high) $(2O(\log n))$ במקרה שאחד מהאינדקסים האלו לא נמצאו, נחזיר FAILIURE. נבדוק האם $high - low + 1$ חזקה של 2 (יש לו ביט אחד בדיוק דלוק בייצוג unsigned). אם לא, נחזיר FAILURE. נסמן משתנה עזר $i \leftarrow high - low + 1$. **כל עוד** $i > 1$**:** 1. נבצע $addWinsInRange(high - \frac{1}{2}i + 1, high)$. $(O(\log n))$ 2. $i \leftarrow \frac{1}{2}i$ כך נוסיף את הניצחונות לקבוצות באופן שתואר בהנחיות. כעת ניגש לאיבר בעץ הקבוצות באינדקס high. $O(\log n)$ נחזיר את המזהה שלו.	**זמן:** נבצע את הלולאה שתיארנו $\log i$ פעמים (כאשר i הוא מספר הקבוצות בטווח). כל איטרציה מכילה פעולות בסיבוכיות זמן כוללת של $O(\log n)$, ושאר הפעולות הן בזמן $O(\log n)$ ולכן סה"כ נקבל סיבוכיות זמן: $O(\log i \cdot \log n)$ **מקום:** הפונקציה לא מקצה שום זיכרון באופן ישיר, אך פעולות העץ והפונקציה addWinsInRange פועלות בסיבוכיות זיכרון של $O(\log n)$. לכן סך הכל: $O(\log n)$

- **בכל מקרה של בעיה בהקצאה/שחרור זיכרון נחזיר ERROR_ALLOCATION**
- **בכל מקרה של הצלחה נחזיר SUCCESS**