מבני נתונים תרגיל רטוב שני

מגישים:

יצחק גרינבוים 318837317

קורן מעברי 207987314

מבנה הנתונים שלנו הוא כדלקמן:

- שני עצי דרגות מבוססי AVL כפי שנלמד בהרצאה: עץ ראשון שנקרא "teams" שממוין לפי key של int teamID ועץ שני שנקרא "PrioritizedTeams" שממוין לפי key של Team (ל־Team קיים אופרטור השוואה בהתאם לנתוני התרגיל).
- מבנה Union-Find כפי שנלמד בהרצאה.
- טבלת ערבול גנרית (Hash Table) הממומשת לפי שיטת Chain Hashing כפי שנלמד בהרצאה. ה־Data של כל Node ב־Linked Lists הוא מצביע על אובייקט PlayerInfo (הסבר על PlayerInfo מופיע בהמשך המסמך).
- Class Team.
- Class Player.
- Class TeamInfo.
- Class PlayerInfo.
- Class List אשר בשימוש בhashTable.
- Class intHash

כעת נסביר ביתר פירוט על כל מבנה ורכיביו:

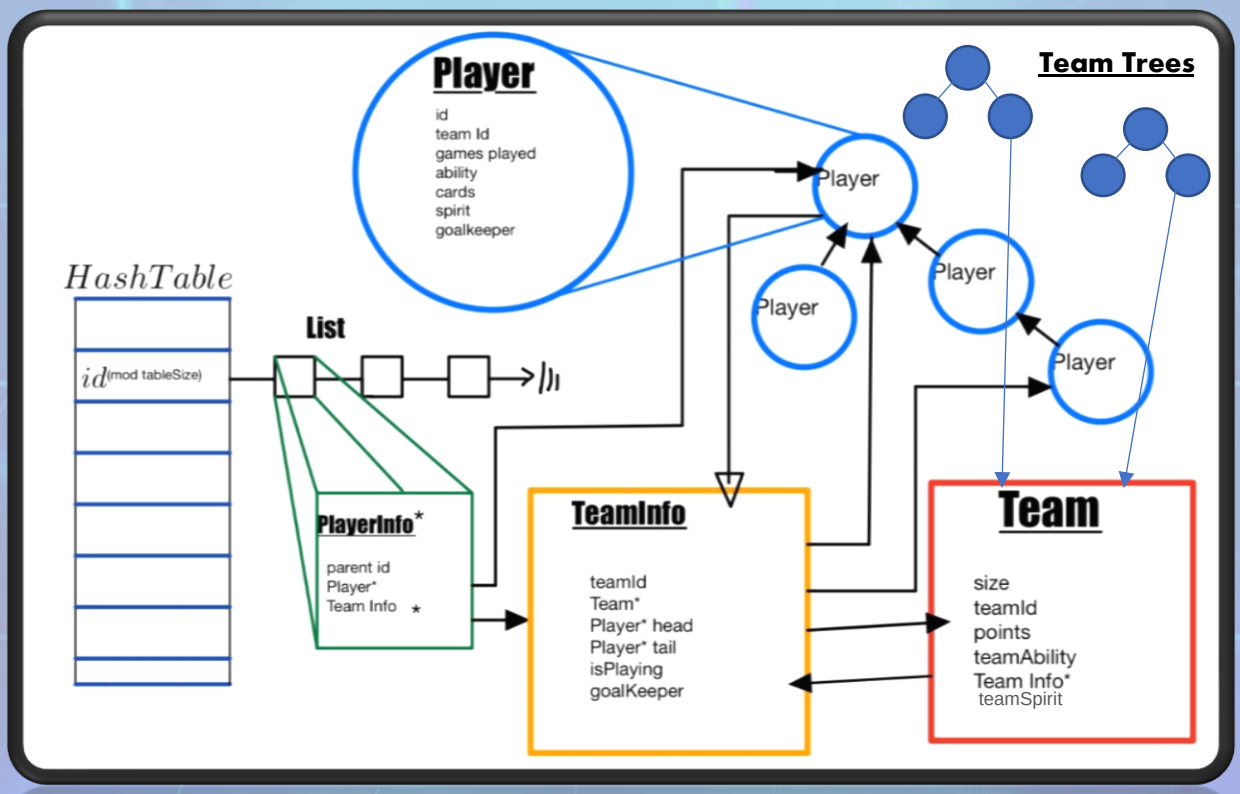
עץ "Teams":

ה־Nodes שלו מורכבים מ־key של int teamID ומ־Data של pointer ל־team. כך ניתן למצוא כל קבוצה בסיבוכיות O(log(k)).

עץ "PrioritizedTeams":

ה־Nodes שלו מורכבים מ־key של class Team ומ־Data של pointer ל־team.

תמונה להמחשה ויזואלית של מבנה הנתונים:



הסבר על האובייקטים:

מבנה ה **UnionFind** ממומש בשיטה דומה לזו שהוצגה בהרצאה על ידי שימוש במערך בו לכל איבר שמור ערך של האב שלו. אצלנו יש שימוש ב**HashTable** בו לכל שחקן שמור ***PlayerInfo** שמחזיק מידע על האב בעץ ההפוך (אין באמת עץ הפוך, זה רק עזר אבסטרקטי הנועד לעזור לנו להתנסח) בPlayerInfo יש מצביע לשחקן עצמו, מספר האב שלו, ומצביע לTeamInfo שיהיה רלוונטי רק עבור ראש קבוצה (שורש של העץ ההפוך) שייוחד על ידי כך שהוא אבא של עצמו.

intHash הוא functionObject בעל אופרטור סוגריים שמקבל מספר ומחזיר אותו לאחר פעולת מודולו לפי גודל הטבלה. האובייקט מתעדכן בכל פעם שיש צורך בשינוי גודל הטבלה.

בתוך **TeamInfo** שמורים מצביעים לPlayers: ראש הקבוצה(head) ול'זנב' הקבוצה(tail) (האחרון שהצטרף כרונולוגית). בכל פעם ששחקן מצטרף, הפרמוטציה שלו נכפלת בזו של האחרון שהיה לפניו (אם זה לא היה רק השורש) ומתווסף לקבוצה כמצביע לשורש כך שיש בו מכפלה של כל התמורות שהצטברו עד כה חוץ מזו של השורש והוא עכשיו הזנב החדש של הקבוצה. אחר כך על מנת להחזיר את הPartial_Spirit נעשה קריאה רקורסיבית שתחזיר מכפלה של כלל התמורות מן השורש ועד אל אותו שחקן (אנו מכניסים מצביע לפרמוטציית הזהות, ובחזרה מן הרקורסיה אנו עושים הכפלה מימין וכך מקבלים ביציאה מהפונקציה את המכפלה של כל התמורות עד אל אותו שחקן כולל), ובאותו הזמן תבצע קיצור מסלולים אם יש צורך. באופן דומה מתוחזק גם שדה של numOfGames. בנוסף שמורים משתנים בוליאנים על מנת לדעת אם הקבוצה עדיין משחקת ואם יש לה שוער.

בתוך **Team** שמורים Points ו teamAbility ו teamSpirit בשביל Play_Match. בכל פעולה בה משתנה זנב הקבוצה/מתבצע איחוד של קבוצות אנו מעדכנים את teamSpirit בהתאם.

Player מחזיק מידע המתוחזק במהלך שרשור השחקנים במבנה UnionFind וביצוע פעולות union/find כך שבכל זמן נתון, אם נבצע סכימה/הכפלה מראש הקבוצה עד לשחקן המבוקש נקבל את המידע המתאים עבור אותו שחקן. אמנם בכל **Player** שמור שדה של מצביע לTeamInfo אבל שדה זה רלוונטי רק כאשר השחקן הוא ראש קבוצה. רק כאשר נבצע Find על שחקן, ונגיע לראש הקבוצה שלו, אז ערך החזרה יהיה TeamInfo ששמור אצל אותו שחקן.

בפעולות: get_partial_spirit, num_games_played_for_player, add_player_cards, יש צורך בגישה לקבוצה שאליה משתייך השחקן ולכן נעשה שימוש בפעולת find ומתבצע כיווץ מסלולים כפי שנלמד בקורס. בפעולת buy_team יש שימוש בפעולת union. כאשר בunion: אם קבוצה גדולה קונה קבוצה קטנה, ראש הקבוצה הקטנה מוכפל משמאל במכפלה המצטברת של חברי הקבוצה הגדולה למעט הראש שלה, וראש הקבוצה הקטנה מצביע לראש הקבוצה הגדולה. ואם קבוצה קטנה קונה, עדיין הקבוצה הקטנה תצביע לגדולה על מנת לתמוך בסיבוכיות כפי שנלמד, אלא שכפי שהראו בשיטת הארגזים בתרגול, אנחנו מכפילים את ראש הקבוצה הגדולה בכל התמורה המצטברת של הקבוצה הקטנה, ולאחר מכן מכפילים את ראש הקבוצה הקטנה (אשר היא זו שקנתה) בהופכי של ראש הקבוצה הגדולה על מנת 'לבטל' את הערך שמגיע למעשה אחריה כרונולוגית. ובאופן דומה עבור numGamesPlayed.

עץ **RankAvl** שהסברנו עליו קודם PrioritizedTeams, הינו עץ דרגות הממוין על פי הקריטריון שהוגדר בתרגיל שמתעדף ability ולאחר מכן id. כפי שנלמד בתרגול, מימשנו פונקציית select(i) שמוצאת את האיבר בדרגה ה i בסיבוכיות $O(\log(k))$.

הסבר על פעולות ממשק world_cup_t:

בכל מקום בהסבר, $\boldsymbol{n}$ מתייחס לכמות כל השחקנים שהצטרפו למערכת מתחילת ריצת התכנית. ו $\boldsymbol{k}$ מתייחס לכמות הקבוצות הפעילות/הנמצאות בעצים. פרט לדיסטרקטור, בו $\boldsymbol{k}$ מתייחס לכלל הקבוצות שהיו מתחילת ריצת התכנית. בנוסף, כאשר כתוב $O(\log^* n)$ משוערך <u>בממוצע על הקלט</u>, הסיבה שמתייחסים לממוצע על הקלט היא בגלל ש-$\boldsymbol{find}$ משתמש כמובן ב$\boldsymbol{hashTable}$ שעובד ב*O(1)* בממוצע על הקלט.

---

world_cup_t

מאתחלת ברשימת אתחול את teams ואת prioritizedTeams להיות nullptr ואת numOfActivePlayers להיות 0. מכיוון שכמות הפעולות כאן היא סופית, אז הסיבוכיות היא $O(1)$.

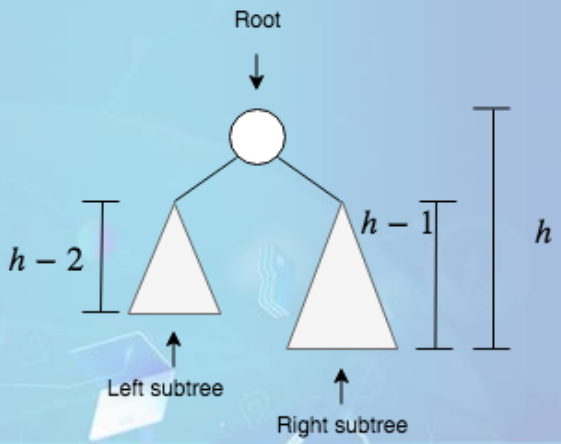
בסך הכל: $O(1)$ במקרה הגרוע.

~world_cup_t

כל Team, TeamInfo, ו־Player שאנחנו מקצים בשבילו זיכרון, אנחנו שומרים מצביע אליו ברשימה מקושרת. מחיקת הרשימה של ה־teams הוא ב־$O(k)$, מחיקת הרשימה של ה־TeamInfo's עולה $O(k)$, ומחיקת הרשימה של ה־players הוא ב־$O(n)$. מהסכום $O(k+k+n)$ נקבל בסה"כ: O(n+k).

בסך הכל: $O(n+k)$ במקרה הגרוע.

Add_team



מבצעים בדיקת תקינות קלט ב־$O(1)$, אחר כך בדיקת קיום הקבוצה בעץ AVL על ידי פונקציית find ב־$O(log(k))$. עושים new ל־TeamInfo, ל־Team, ולשני Nodes של RankAVL. אופן האתחול שלהם נכתב כך שכל אחד מהם קורה ב־$O(1)$. מוסיפים לראש רשימה מקושרת (שנועדה להורס של העולם) Node עם הפרמטרים של הקבוצה ב־$O(1)$. מוסיפים לשני עצי ה־AVL את הצומת הרלוונטי לכל עץ של קבוצות ב־$O(log(k))$. ולבסוף ומוסיפים 1 ל־counter של מספר הקבוצות בעולם ב־$O(1)$. יש לציין אם הקצאת הזיכרון נכשלת, אז הטיפול בכך לא ישפיע על הסיבוכיות שהרי כל delete כתוב כך שהוא קטן/שווה ל־new מבחינת סיבוכיות. עלות סך כל הפעולות היא: $O(1+log(k)+1+1+1+1+log(k)+log(k)+1)$, כלומר $O(3log(k)+6)$.

בסך הכל: $O(log(k))$ במקרה הגרוע.

---

remove_team

בדיקת תקינות קלט ב־O(1). חיפוש קיום הקבוצה ב־AVL של הקבוצות ב־O(log(k)). שינוי הפרמטר של הקבוצה isStillPlaying ל־false ב־O(1). מחיקת הצומת שלה משני עצי ה־AVL ב־O(log(k)). חיסור 1 מה־counter של מספר הקבוצות המשחקות במשחק ב־O(1). סכום הפעולות הוא O(1+log(k)+1+2log(k)+1), כלומר O(3log(k)+3).

בסך הכל: O(log(k)) במקרה הגרוע.

Add_player

בדיקת תקינות קלט ב־$O(1)$. בדיקת קיום הקבוצה ב־AVL של הקבוצות ב־$O(log(k))$ ובדיקת קיום השחקן ב־Union-Find על ידי שימוש ב־HashTable ששומר את המידע הרלוונטי ב־$O(1)$ משוערך בממוצע על הקלט בהתאם לנלמד בהרצאה על מבני הנתונים הללו. פעולת new ל־Player ב־$O(1)$. וטיפול באפשרות של שגיאת הקצאה בסיבוכיות שלא משפיעה על הסיבוכיות הכוללת. הוספת השחקן לראש הרשימה שמקושרת ב־$O(1)$. מציאת הצמתים של הקבוצות בעצי ה־AVL ב־$O(log(k))$. הוספת ה־Ability ב־$O(1)$. פעולת new לצומת חדש של הקבוצה של השחקן מעודכן עם השחקן החדש בתוכה ב־$O(1)$. עדכון ה־TeamInfo ב־$O(1)$. הסרת הצומת של הקבוצה הלא מעודכנת מהעץ prioritizedTeams ב־$O(log(k))$ והכנסת הצומת המעודכן ב־$O(log(k))$. הכנסה ל־HashTable ב־$O(1)$ משוערך בממוצע על הקלט. ביצוע find ב־UnionFind ב־$O(1)$ משוערך בממוצע על הקלט כדי להטיב את הסיבוכיות בהמשך. תיקון הפרמטר של numOfGamesPlayed ב־$O(1)$. עדכון ב־TeamInfo של ה"זנב" של אותו "שרוך" של שחקנים ב־$O(1)$. הוספת 1 לגודל הקבוצה ב־$O(1)$. גם כאן הטיפול בשגיאת ההקצאה לא מזיק לסיבוכיות. הסכום הוא O(1+log(k)+1+1+1+1+2log(k)+1+1+1+log(k)+log(k)+1+1+1+1+1), כלומר $O(5log(k)+12)$.

בסך הכול: $O(log(k))$ משוערך בממוצע על הקלט.

Play_match

בדיקת תקינות קלט ב־$O(1)$. מציאת שתי הקבוצות בעצי ה־AVL ב־$O(log(k))$. בדיקת קיום קבוצה ושוערים ב־$O(1)$. אם יש קבוצה מנצחת על ידי יכולת, אז חישוב מי מנצחת ב־$O(1)$ ולאחר מכן הוספת נקודות בהתאם ב־$O(1)$, אחרת, ביצוע השוואה של רוחות הקבוצה, תוספת נקודות מתאימות לקובצות ובחירת ערך חזרה מתאים ב$O(1)$ השוואת ב־$O(1)$. הוספת 1 ל־numOfGamesPlayed של ראשי הקבוצות (teamHead) עולה $O(1)$. הפעולות הללו עולות: O(1+1+1+1+1+2log(k)), כלומר O(2log(k)+6).

בסך הכל: $O(log(k))$ במקרה הגרוע.

Num_played_games_for_player

בדיקת קלט ב־$O(1)$. בדיקה ב־HashTable אם השחקן קיים ב־$O(1)$ משוערך בממוצע על הקלט. אתחול משתנה מקומי result=0 והעברת המצביע שלו לקריאה לfind ב־UnionFind ובכל צעד בחזרה מן הרקורסיה נוסיף ל result את הערך השמור בשחקן כפי שהסברנו לעיל. באמור בהרצאה ביצוע find עולה $O(\log^* n)$ משוערך בממוצע על הקלט. לבסוף נחזיר המשתנה המקומי result. פעולות אלו הן ב־O(1+1+log*(n)), כלומר O(log*(n)+2).

בסך הכל: $O(log^* n)$ משוערך בממוצע על הקלט.

Add_player_cards

ראשית אנו מבצעים בדיקת קלט $O(1)$. על מנת לבדוק אם הקבוצה עדיין פעילה אנו מבצעים את פעולת find על השחקן $O(\log^* n)$. אם הקבוצה פעילה נוסיף לו כרטיסים כמספר שהתקבל.

בסך הכל: $O(log^* n)$ משוערך, בממוצע על הקלט.

Get_player_cards

ראשית אנו מבצעים בדיקת קלט $O(1)$. לאחר מכן ניגשים לשחקן בHashTable ומחזירים את הערך השמור של cards אצל השחקן.

בסך הכל: $O(1)$ בממוצע על הקלט.

Get_team_points

ראשית אנו מבצעים בדיקת קלט $O(1)$. אם הקלט תקין, אנו ניגשים לקבוצה בעץ Teams $O(\log(k))$, ומחזירים את כמות הנקודות שלה.

בסך הכל: $O(\log(k))$ במקרה הגרוע.

Get_ith_pointless_ability

ראשית אנו מבצעים בדיקת קלט $O(1)$. לאחר מכן אנו מחפשים בעץ PrioritizedTeams את האיבר מאינדקס i (העץ ממוין לפי אופרטור ההשוואה שהוצג לעיל) שהתקבל בעזרת פעולת select(i). פעולה זו עולה לנו $O(\log(k))$.
בסך הכל: $O(\log(k))$ במקרה הגרוע.

Get_partial_spirit

ראשית אנו מבצעים בדיקת קלט $O(1)$. אנו מאתחלים פרמוטציית זהות וקוראים לפונקציית find עם id של השחקן המתאים ומעבירים לה מצביע של התמורה. כפי שהוסבר לעיל, פעולה זו עולה ל'שורש' של השחקנים בקבוצתו של השחקן, ומכפילה את התמורה בכל צומת בדרך. מאופן הבנייה מובטח לנו שזה יסתכם לכדי בדיוק מכפלת כל התמורות של השחקנים עד לאותו שחקן ברגע שהצטרף לקבוצה מבחינה כרונולוגית. פעולה זו עולה לנו $O(\log^* n)$ משוערך.

בסך הכל: $O(\log^* n)$ משוערך, בממוצע על הקלט.

Buy_team

ראשית אנו מבצעים בדיקת קלט $O(1)$. עבור המקרה בו הקבוצה הנקנית ריקה, אנו פשוט מוחקים אותה מעץ הקבוצות $O(log(k))$. אם הקבוצה הקונה היא ריקה, אנו מבצעים שינוי בקבוצה הנקנית ומשנים את ה'זהות' (id) שלה לזהות של הקבוצה הקונה ומסירים את הקבוצה הקונה הישנה $O(log(k))$. במקרה אחר בו שתי הקבוצות אינן ריקות, נבצע קריאה לפונקציית union ותתבצע כפי שהוסבר לעיל $O(\log^* n)$ משוערך.
כמובן שבכל מקרה מעדכנים את מיקום הקבוצות בעץ prioritized מפאת שינוי פוטנציאלי בteamAbility. $O(\log(k))$.

בסך הכל: $O(\log(k) + \log^* n)$ משוערך.
