

חלק יבש רטוב 2

מבנה הנתונים יכיל:

world_cup_t:

- **Avl<Team> all_teams_by_id** - עץ חיפוש בינארי מאוזן AVL שמכיל את כל הקבוצות במערכת וממיון לפי תז.
- **RankTree<Team>all_teams_by_abilities** - עץ דרגות (כל צומת מחזיק את מס' הצמתיים בתת העץ שהוא השורש) מבוסס AVL שמכיל את הקבוצות כאשר הן ממוינות לפי סך כל abilities של כל השחקנים בהן, אם שווה אז קבוצה קטנה יותר עם המזהה שלה קטן יותר.
- **UnionFindHash all_players** - טבלת ערבול של השחקנים ומבנה unionFind שממומש באמצעות עצים הפוכים, כיווץ מסלולים ואיחוד לפי גודל. אצל כל שחקן בטבלה יש מצביע לצומת שלו בעץ ההפוך של השחקנים בקבוצה שלו.
- **num_of_teams** - מספר הקבוצות.

:Team

- **teamId** - מזהה הקבוצה
- **num_of_players** - מספר השחקנים בקבוצה.
- **points** - מספר הנקודות של הקבוצה.
- **team_spirit** - מכפלת כל פרומטציות שחקני הקבוצה בסדר כרונולוגי - מתעדכנת בכל הוספת שחקן ובקניית קבוצה.
- **player_abilities** - סכום ability של כל השחקנים בקבוצה.
- **num_of_matches** - מספר המשחקים ששיחקה הקבוצה מאז שנכנסה למערכת.
- **byId** - משתנה בוליאני שעל פיו יודעים עם הקבוצה ממוינת לפי תז או לפי abilities שלה.
- **num_of_goal_keepers** - מספר השוערים בקבוצה. - שניתן יהיה לעקוב בקלות מתי קבוצה הופכת לכשירה ומתי כבר לא.
- **PlayerNode* root_of_player** - מצביע לשורש העץ ההפוך של שחקני הקבוצה.

:PlayerNode

- **player_id** - מזהה השחקן
- **playerNode* parent** - מצביע לאבא שלו בעץ ההפוך של השחקנים בקבוצה.
- **Team* team** - מצביע לטיפוס שמייצג את קבוצת השחקן. - כך נוכל לקבל את הפרטים של הקבוצה בגישה ישירה. אם המצביע הוא nullptr אצל השורש אז זה שחקן עבר.
- **games_played** - אם שחקן זה הוא השורש אז ערך זה יכיל את מס' המשחקים המעודכן ששיחק. אם הוא לא השורש אז סכום ערכים אלו במסלול עד לשורש כולל יכיל את הערך המעודכן לשחקן שהתחלנו ממנו.

- **partial spirit** - אם השחקן הזה הוא השורש, אז יכיל את כפל כל הפרומטציות של השחקנים בקבוצה עד אליו(כולל הוא). אחרת, כפל ערכים אלו (כל צומת שנעבור נכפול משמאל) במסלול עד לשורש כולל יכיל את ה- partial spirit של השחקן שהתחלנו ממנו.

Player:

- **player_id** - מזהה השחקן
- **ability** - היכולת של השחקן.
- **cards** - מספר הכרטיסים שקיבל השחקן. - מתעדכן בהתאם.
- **games_played** - מספר המשחקים ששיחק השחקן כשהצטרף למערכת.
- **bool goalKeeper** - אם השחקן הוא שוער יקבל true, אחרת false.
- *** himself PlayeNoder** - מצביע לnode שלו בעץ ההפוך של שחקנים בקבוצה.
- **-spirit** הפרמוטציה ההתחלתית שלו.

UnionFindHash:

- **players_table** - טבלת ערבול (בשיטת chain hashing) שמכילה את כל השחקנים, הטבלה מיוצגת ע"י מערך דינאמי שכל תא בו הוא עץ AVL.
- **num_of_players** - מספר השחקנים במערכת.

הערות על הסיבוכיות: (הוכחה מפורטת לאחר פירוט הפונקציות.)

- ראינו בהרצאה שהכנסה, הוצאה, חיפוש בעץ חיפוש מאוזן (AVL) ובפרט, גם בעץ דרגות מבוסס AVL, מתקיימים בסיבוכיות של $O(\log n)$ במקרה הגרוע כאשר n זה מספר הצמתים בעץ. לכן כל פעולה של הכנסה, הוצאה או חיפוש צומת בעץ בפונקציות יהיו בסיבוכיות זו.
- פונקציית הערבול בטבלת הערבול היא פונקציית המודולו על גודל המערך הנוכחי. פונקציית ערבול זו מקיימת את הנחת הפיזור האחיד הפשוט מפני שהיא ממפה את האיברים בצורה אחידה לכל אחד מהתאים. כשמספר השחקנים בטבלה מגיע לגודלה, נגדיל את המערך פי 2 וכך נשמור על פקטור עומס ב- $O(1)$. ראינו שבשיטת השרשראות ותחת הנחת הפיזור האחיד הפשוט פעולות ההוספה, חיפוש והסרה מתבצעות ב- $O(1)$ **בממוצע על הקלט**. בנוסף, ראינו **שעם הגדלת המערך**, סיבוכיות הזמן **המשוערכת** של פעולות אלו היא $O(1)$ בממוצע על הקלט.
- ראינו בהרצאה ובתרגול שאם נשתמש בunionFind שממומש באמצעות עצים הפוכים ונבצע איחוד לפי גודל וכיוון מסלולים, סיבוכיות הזמן המשוערכת של סדרת פעולות union, find היא $O(\log^* n)$ כאשר n הוא מספר האיברים במבנה.

סיבוכיות מקום: מבנה הנתונים מכיל עצים בגודל k קבוצות לכל היותר וטבלת ערבול בגודל n שחקנים ועצים הפוכים כאשר סך כל השחקנים בעצים שווה ל- n לכל היותר, בנוסף מספר שדות קבוע לכל שחקן ולכל קבוצה. ומספר שדות סופי בתוך world cup. לכן $O(n+k)$ מקום.

הערות על כל הפונקציות:

במידה ונשלח ערך אשר נחשב לא חוקי לפי הוראות התרגיל יוחזר invalid_input. במידה ותהיה בעיה בהקצאת זיכרון יוחזר ALLOCATION ERROR, ובמידה ולא נמצא שחקן או קבוצה שמחפשים יוחזר Failure, או אם רוצים להוסיף שחקן שכבר קיים או קבוצה שכבר קיימת יוחזר Failure גם כן.

הסבר הפונקציות:

add_team

מוסיפים את הקבוצה ל all_teams_by_id ול all_teams_by_abilities ומוסיפים 1 ל num_of_teams. סיבוכיות: הכנסות לעצי הקבוצות - $O(\log k)$.

remove_team

מוציאים את הקבוצה, אם הקבוצה לא ריקה, שמים אצל השחקן שהוא שורש העץ ההפוך שלה במצביע אליה nullptr וכך נוכל לדעת כשקבוצה של שחקן הוסרה. מסירים אותה מ all_teams_by_id ומ all_teams_by_abilities ומורדים 1 מ num_of_teams. סיבוכיות: חיפוש והוצאות מעצי הקבוצות - $O(\log k)$.

add_player

מוציאים את הקבוצה שאליה עתיד להתווסף השחקן, ב2 העצים. מוסיפים את ability שלו לשדה בקבוצה ומעדכנים את team_spirit של הקבוצה. יוצרים לו PlayerNode משלו, אם זה השחקן הראשון בקבוצה אז גם מעדכנים את המצביע לשורש אצל הקבוצה ואת המצביע לקבוצה אצלו. אחרת, מחברים אותו לשורש העץ ההפוך של הקיים כבר לקבוצה.

אם זה לא השחקן הראשון בקבוצה אז את partial_spirit אצלו נעדכן להיות:

(הרוח הקבוצתית כולל השחקן) team_spirit * inv -> partial_spirit players_root (ההופכי של השורש) ואת games_played אצלו נעדכן להיות:

gamesPlayed - players_root->games_played (ההתחלתי של השחקן, שהתקבל כפרמטר).

מכניסים את השחקן לטבלת הערבול (אם הגענו למס' שחקנים כגודל הטבלה, נגדיל אותה פי 2) ומגדילים את num_of_players שלה ב1 ואת מס' השוערים אם זה שוער.

סיבוכיות: מציאת הקבוצות - $O(\log k)$ כאשר k מס' הקבוצות במערכת, הכנסה לטבלת ערבול דינאמית - $O(1)$ בממוצע על הקלט משוער. לכן סה"כ קיבלנו: $O(\log k)$ משוער, בממוצע על הקלט.

play_match

מוציאים את הקבוצות בעצים ובודקים מי המנצחת לפי התנאים של התרגיל (אם לאחת מהן אין שוער יוחזר (FAILURE)).

מוסיפים 1 לgames_played של שורשי הקבוצות שהשתתפו בmatch. מסירים את הקבוצות מהעץ abilities ומחזירים אותם עם התוצאות המעודכנות כדי שהעץ המדורג יישאר ממורכז כמו שצריך.

סיבוכיות: מציאת הקבוצות, הסרתן והחזרתן לעץ. - $O(\log k)$ כאשר k מס' הקבוצות במערכת. כל שאר הפעולות ב $O(1)$ סה"כ $O(\log k)$.

num_played_games_for_player(int playerId)

מוציאים את השחקן בטבלת הערבוד.

נבצע (find(playerId) שבמהלכו אנו עולים מהצומת של השחקן בעץ ההפוך עד לשורש (מתקיים גם כיווץ מסלול ועדכון של השדות בהתאם לפי שיטת הארגזים שנלמדה בתרגול) ותוך כדי סוכמים את ערכי games_played של כל צומת במסלול כולל השורש והסכום שקיבלנו הוא games_played של השחקן ואותו נחזיר.

סיבוכיות: מציאת השחקן בטבלת הערבוד - $O(1)$ בממוצע על הקלט, הפעלת find עם כיווץ המסלולים, לכן סה"כ $O(\log n)$ משוערך, בממוצע על הקלט, כאשר n הוא מספר השחקנים בכל המערכת.

add_player_cards(int playerId, int cards)

מוציאים את השחקן בטבלת הערבוד ושומרים מצביע אליו.

נבצע (find(playerId) כדי לקבל את המצביע לקבוצת השחקן ולבדוק אם קבוצתו הוסרה או לא. אם הוא nullptr אז השחקן הוא שחקן עבר ונחזיר FAILURE. אחרת, נוסיף cards למס' הכרטיסים הקיים לשחקן.

סיבוכיות: מציאת השחקן בטבלת הערבוד - $O(1)$ בממוצע על הקלט, הפעלת find עם כיווץ המסלולים, הוספה בגישה ישירה דרך מצביע, לכן סה"כ $O(\log n)$ משוערך, בממוצע על הקלט, כאשר n הוא מספר השחקנים בכל המערכת.

get_player_cards

מוציאים את השחקן בטבלת ערבוד ומחזירים את מספר הכרטיסים שלו.

סיבוכיות: מציאת השחקן בטבלת הערבוד - $O(1)$ בממוצע על הקלט, החזרת מספר הכרטיסים - $O(1)$.

get_team_points

מוציאים את הקבוצה all_teams_by_id ומחזירים את מספר הנקודות שלה.

סיבוכיות: מציאת הקבוצה - $O(\log k)$ כאשר k מס' הקבוצות במערכת, החזרת מס' הנקודות ב- $O(1)$.

get_ith_pointless_ability(int i

העץ `all_teams_by_abilities` מכיל את הקבוצות כאשר הן ממוינות לפי סכום היכולות של השחקנים בה (אם זהה אז לפי מזהה), לכן, אם נפעיל פונקציית `select` על העץ עם האינדקס `i`, נקבל בדיוק את הקבוצה שנמצאת במקום ה-`i` בסדר הזה. (מתייחסים ל-`i` כאילו האינדקס הראשון הוא 0).
סיבוכיות: עץ דרגות מאפשר לבצע פונקציה זו בסיבוכיות $O(\log k)$ כאשר k הוא מספר האיברים בעץ, לכן אכן הסיבוכיות היא אכן $O(\log k)$ כאשר k הוא מס' הקבוצות במערכת.

get_partial_spirit(playerId)

מוצאים את השחקן בטבלת הערבול.

נבצע `find(playerId)` שבעזרתו אנו מגיעים לקבוצה של השחקן כדי לראות האם היא קיימת, במידה ולא יוחזר `Failure`. מתקיים גם כיווץ מסלול ועדכון של השדות בהתאם לפי שיטת הארגזים שנלמדה בתרגול. בנוסף במהלך העלייה מהצומת של השחקן בעץ ההפוך עד לשורש אנו מחשבים את מכפלת ערכי `partial_spirit` של כל צומת במסלול כולל השורש והמכפלה שקיבלנו הוא `partial_spirit` של השחקן ואותו נחזיר.
סיבוכיות: מציאת השחקן בטבלת הערבול - $O(1)$ בממוצע על הקלט, הפעלת `find` עם כיווץ המסלולים, לכן סה"כ $O(\log^* n)$ משוערך, בממוצע על הקלט, כאשר n הוא מספר השחקנים בכל המערכת.

buy_team

מוצאים את הקבוצות בעץ `all_teams_by_id`.

מבצעים `union` לעצים ההפוכים של הקבוצות. כאשר האיחוד מתבצע לפי גודל, כלומר שורש הקבוצה הקטנה יצביע על שורש הקבוצה הגדולה בסיום הפעולה.
בפעולת `union` מבצעים:

החסרת `games_played` של שורש הקבוצה הנקנית מ-`games_played` של שורש הקבוצה הקונה.
החסרת `games_played` של שורש הקבוצה הקונה מ-`games_played` של שורש הקבוצה הנקנית.

אם הקבוצה הקונה גדולה יותר:

מכפילים את `partial_spirit` הופכית של שורש הקבוצה הקונה ב-`team_spirit` של הקבוצה שלו ו-`partail_spirit` של שורש הקבוצה הנקנית **משמאל לימין**, ושמים ב-`partial_spirit` של שורש הקבוצה הנקנית.

אם הקבוצה הנקנית גדולה יותר:

מכפילים את `team_spirit` של הקבוצה הקונה ב-`partial_spirit` של שורש הקבוצה הנקנית **משמאל לימין**, ושמים ב-`partial_spirit` של שורש הקבוצה הנקנית.

מכפילים את ההופכית של `partial_spirit` של שורש הקבוצה הנקנית (שבדיוק התעדכן) ב-`partial_spirit` של שורש הקבוצה הקונה. **משמאל לימין**, ושמים ב-`partial_spirit` של שורש הקבוצה הקונה.

נסכום את הערכים הרלוונטיים (מס' שחקנים, שוערים, יכולות) ונעדכן אותם אל הקבוצה הקונה, וכמובן נעדכן גם את הרוח הקבוצתית שלה להיות `buyer->team_spirit * bought->team_spirit` בהתאם לדרישה כי השחקנים של הקבוצה הקונה נחשבים ותיקים יותר.

שמים `nullptr` במצביע לקבוצה אצל השורש של הקבוצה שנקנתה כדי לסמן שהיא הודחה וכעת כל השחקנים בעץ ההפוך שהוא השורש הם שחקני עבר.

מוחקים את הקבוצה שנקנתה משני עצי הקבוצות.

סיבוכיות: $O(\log k + \log^* n)$ משוערך, בממוצע על הקלט, כאשר n הוא מספר השחקנים בכל המערכת כולל שחקני עבר, k מספר הקבוצות המערכת.

***הערה על הסיבוכיות המשוערכת של הפונקציות מלבד `add_player`:**

בכל הפונקציות בהן הסיבוכיות היא משוערכת מלבד הפונקציה `add_player` יש קריאות לפונקציות `find`, `union` של מבנה הנתונים `union`, `find` בעל עצים הפוכים, איחוד לפי גודל וכיווץ מסלולים. לכן, לפי מה שנאמר בהערות בהתחלה על הסיבוכיות, הפונקציות האלו משוערכות יחד ככה שאכן הסיבוכיות המשוערכת שלהן תכיל $O(\log^* n)$.