

חלק יבש רטוֹב 2

מבנה הנתונים יכול:

world_cup_t:

- **Avl<Team> all_teams_by_id** •
וממויין לפי ת.ז.
- **RankTree<Team>all_teams_by_abilities** •
בתת העץ שהוא השורש) מבוסס AVL שמכיל את הקבוצות הממויניות לפי סדר של abilities של כל השחקנים בהן, אם שווה אז קבוצה קטנה יותר עם המזהה שלה קטן יותר.
- **UnionFindHash all_players** •
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

- מזהה הקבוצה **teamId** •
- מספר השחקנים בקבוצה **num_of_players** •
- מספר הנקודות של הקבוצה **points** •
- מכפלת כל פרוימטיציות שחקני הקבוצה בסדר כירונולוגי - מתעדכנת בכל הוספה שחקן ובקניית קבוצה. **team_spirit** •
- סכום העוצמה **player_abilities** •
מספר המשחקים ששיחקה הקבוצה מאז שנכנסה למערכת. **num_of_matches** •
- משתנה בוליאני שועל פיו יודעים עם הקבוצה ממוינת לפי ת.ז או לפי abilities שלה. **byId** •
- מספר השוערים בקבוצה. - שנייתן יהיה לעקב בקלות מתי קבוצה הופכת לכירה ומתי כבר לא. **num_of_goal_keepers** •
- מצביע לשורש העץ ההפוך של שחקני הקבוצה. **PlayerNode* root_of_player** •

:PlayerNode

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

- **partial spirit** - אם השחקן הזה הוא השורש, אז יכול את כפל כל הпромותציות של השחקנים בקבוצה עד אליו(כולל הוא). אחרת, כפל ערכיהם אלו (כל צומת שנעבור נכפול משמאלו) במסלול עד לשורש כולל יכול את partial spirit של השחקן שהתחלנו ממנו.

:Player

- **player_id** - מזהה השחקן
- **ability** - היכולת של השחקן.
- **cards** - מספר הCARTEISIM שקיבל השחקן. - מתעדכן בהתאם.
- **games_played** - מספר המשחקים ששיחק השחקן כשהצטראף למערכת.
- **bool goalKeeper** - אם השחקן הוא שוער יקבל true, אחרת false.
- **PlayeNoder*** himself - מצביע לnodeו של שחקנים בקבוצה.
- **spirit** - הпромותציה ההתחלתית שלו.

:UnionFindHash

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

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

- ראיינו בהרצאה שהכנסה, הוצאה, חיפוש בעץ חיפוש מאוזן (AVL) ובפרט, גם בעץ דרגות מבוסס AVL מתקיימים בסיבוכיות של $O(\log n)$ במקורה הגורע כאשר ח' 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 האחד הפשוט מפני שהיא מפנה את האיברים בצורה אחידה לכל אחד מה汰אים. כמספר השחקנים בטבלה מגיע לגודלה, נגדיל את המערך פי 2 וכך נשמר על פקטור עומס ב- (1) . ראיינו שבשיטת השרשאות ותחת הנחת הפיזור האחד הפשוט פעולות ההוסף, חיפוש והסרה מתבצעות ב- $O(\log n)$ בממוצע על הקולט. בנוספ', ראיינו שעם **הגדלת המערך**, סיבוכיות הזמן המשוערת של פעולות אלו היא $O(\log n)$ בממוצע על הקולט.
- ראיינו בהרצאה ובתרגול שגם משתמש בUnionFind שמנוהש באמצעות עצים הפוכים ונבעץ איחוד לפי גודל וכיום מסלולים, סיבוכיות הזמן המשוערת של סדרת פעולות find, חסום היא $O(\log n)$ כאשר ח' הוא מספר האיברים במבנה.

סיבוכיות מקום: מבנה הנתונים מכיל עצים בגודל A קבוצות לכל היותר וטבלת ערבול בגודל Ch שחקנים ועצים הפוכים כאשר סך כל השחקנים בעצים שווה לח' לכל היותר, בנוסף מספר שדות קבוע לכל שחקן ולכל קבוצה. ומספר שדות סופי בטור cap_w . לנ' $O(k+Ch)$ מקום.

הערות על כל הפקנציות:

במידה ונשלח ערך אשר נחשב לא חוקי לפי הוראות התרגל יוחזר `invalid_input`. במידה ותהיה בעיה בהקצת זיכרון יוחזר `Allocation ERROR`, ובמידה ולא נמצא שחקן או קבוצה שמחפשים יוחזר `Failure`, או אם רוצים להוציא שחקן שכבר קיימים או קבוצה שכבר קיימת יוחזר `Failure` גם כן.

הסבר הפקנציות:

add_team

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

remove_team

מוצאים את הקבוצה, אם הקבוצה לא ריקה, שמיים אצל השחקן שהוא שורש העץ ההפור שלו במצביו אליו `all_teams_by_id` ונקט נוכן לדעת כשקבוצה של שחקן הוסרה. מסירים אותה מ `all_teams_by_id` ומוסיפים 1 ל `num_of_teams` ומוסיפים 1 ל `all_teams_by_abilities`.
סיבוכיות: חיפושים והוצאות מעצם הקבוצות - $O(k \log k)$.

add_player

מוצאים את הקבוצה שאליה עתיד להוסיף השחקן, ב 2 העצים. מוסיפים את `ability` שלו לשדה בקבוצה ומעדכנים את `team_spirit` של הקבוצה. יוצרים לו `PlayerNode` מסוילו, אם זה השחקן הראשון בקבוצה אז גם מעדכנים את המצביע לשורש אצל הקבוצה ואת המצביע לקבוצה עצמו. אחרת, מוחברים אותו לשורש העץ ההפור של הקיימס כבר לקבוצה.

אם זה לא השחקן הראשון בקבוצה אז את `partial_spirit` אצלו נעדכן להיות:
 (הרוח הקבוצתית כולל השחקן) `players_root->partial_spirit.inv * team_spirit` (ההופכי של השורש)
 ואת `games_played` אצלו נעדכן להיות:
`gamesPlayed - players_root->games_played`.

מכניסים את השחקן לטבלת הערבול (אם הגיענו למס' שחקנים בגודל הטבלה, גודיל אותה פי 2) ומגדילים את `players_of_players` שלו ב 1 ואת מס' השוערים אם זה שוער.
סיבוכיות: מציאת הקבוצות - $O(\log k)$ כאשר k מס' הקבוצות במערכת, הכנסה לטבלת ערבול דינמית - $(1/O)$ במנצע על הקלט משוער. לכן סה"כ קיבלנו: $(\log k)^2$ משוער, בממוצע על הקלט.

play_match

מוצאים את הקבוצות ביעדים ובודקים מי המנצח לפי התנאים של התרגיל (אם לאחת מהן אין שוער יוחזר). (FAILURE).

מוסיפים 1 ל-edged של שורשי הקבוצות שהשתתפו ב-match. מסירים את הקבוצות מהעץ abilities ומוחזירים אותן עם התוצאות המעודכנות כדי שהעץ המדודג ישאר מモין 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)

מוצאים את השחקן בטבלת הערבול.

מבצע ($\text{find}(playerId)$) שבמהלכו אנו עולמים מהצומת של השחקן בעץ הפוך עד לשורש (מתקיים גם כיווץ מסלול) וערכו של השדות בהתאם לפיה שיטת הארגזים שנלמדה בתרגול) וטור כדי סוכמים את ערכי played_games של כל צומת במסלול כולל השורש והסכום שקיבلونו הוא played_games של השחקן ואוטו נחזר.

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

add_player_cards(int playerId, int cards)

מוצאים את השחקן בטבלת הערבול ושומרים מצביע אליו.

מבצע ($\text{find}(playerId)$) כדי לקבל את המצביע לקבוצת השחקן ולבדק אם קבוצתו הוסרה או לא. אם הוא **nullptr** אז השחקן הוא שחקן עבר ונחזר FAILURE. אחרת, נוסיף cards למס' הCARDSIMS הקיימים לשחקן.

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

get_player_cards

מוצאים את השחקן בטבלת ערבול ומוחזרים את מספר הCARDSIMS שלו.

סיבוכיות: מציאת השחקן בטבלת הערבול - (1) במנוצע על הקלט, החזרת מספר הCARDSIMS - (1) .

get_team_points

מוצאים את הקבוצה ב-**id_all_teams_by** ומוחזרים את מספר הנקודות שלה.

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

get_ith_pointless_ability(int i)

הע^z **all_teams_by_abilities** מכיל את הקבוצות כאשר הⁿ ממיניות לפי סכום היכולות של השחקנים בה (אם זהה אז לפⁱ מזזה), אך, אם נפעיל פונקציית select על הע^z עם האינדקס ⁱ, נקבל בדיקת הקבוצה שנמצאת במקום הⁱ- בסדר זהה. (מתייחסים לⁱ-ן אליו האינדקס הראשון הוא 0).

סיבוכיות: עץ דרגות מאפשר לבצע פונקציה זו בסיבוכיות $O(\log k)$ כאשר k הוא מספר האיברים בעץ, אך אכן הסיבוכיות היא אקן $O(k)$ כאשר k הוא מס' הקבוצות במערכת.

get_partial_spirit(playerId)

מוצאים את השחקן בטבלת הערבול.

בצע **(playerId)** שבעזרתו אנו מגיימים לקבוצה של השחקן כדי לראות האם היא קיימת, במידה ולא יוחזר Failure. מתקיים גם כיווץ מסלול ועדכו של השדות בהתאם לפⁱ שיטת הארגזים שנלמדה בתרגול. בנוסף במהלך העליה מהצומת של השחקן בעץ ההפוך עד לשורש אנו מחשבים את מכפלת ערכי partial_spirit של כל צומת במסלול כולל השורש והמכפלה שקיבلونו הוא partial_spirit של השחקן אותו נחזר.

סיבוכיות: מציאת השחקן בטבלת הערבול - $O(1)$ בmäßig ערך הקטל, הפעלת find עם כיווץ המסלולים, אך סה"כ $O(\log n)$ משועך, בmäßig ערך הקטל, כאשר n הוא מספר השחקנים בכל המערכת.

buy_team

מוצאים את הקבוצות בעץ **id_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.

בפועל union מביצעים:

החסרת games_played של שורש הקבוצה הנקייה מדינית games_played של שורש הקבוצה הקונה.

החסרת games_played של שורש הקבוצה הקונה מדינית games_played של שורש הקבוצה הנקייה.

אם הקבוצה הקונה גדולה יותר:

מכפילים את **partial_spirit** הופכית של שורש הקבוצה הקונה ב**team_spirit** של הקבוצה שלו וב**partial_spirit** של שורש הקבוצה הנקייה **משמאליימין**, ושים **partial_spirit** של שורש הקבוצה הנקייה.

אם הקבוצה הנקייה גדולה יותר:

מכפילים את **team_spirit** של הקבוצה הקונה ב**partial_spirit** של שורש הקבוצה הנקייה **משמאליימין**, ושים **partial_spirit** של שורש הקבוצה הנקייה.

מכפילים את ההפכית של **partial_spirit** של שורש הקבוצה הנקייה (שבディוק התעדכן) ב**partial_spirit** של שורש הקבוצה הקונה. **משמאליימין**, ושים **partial_spirit** של שורש הקבוצה הקונה.

נסכם את הערכים הרלוונטיים (מוס' שחקנים, שערים, יכולות) ונעדקן אותם אל הקבוצה הרצויה, וכמוון נעדקן גם את הרוח הקבוצתית שלה להיות $\text{team_spirit} * \text{bought} \rightarrow \text{team_spirit}$ בהתאם לדרישת C .

השחקנים של הקבוצה הרצויה נחשבים ותיקים יותר.

שמות ztr במצבייע לקבוצה אצל השורש של הקבוצה שנתקנתה כדי לסמך שהיא הودחה וכעת כל השחקנים בעץ הפוך שהוא השורש הם שחקני עבר.

מוחקים את הקבוצה שנתקנתה משני עצי הקבוצות.

סיבוכיות: $O(n^2 \log k)$ משוערך, בממוצע על היקלט, כאשר n הוא מספר השחקנים בכל המערכת כולל שחקני עבר, k מספר הקבוצות המערכת.

***הערה על הסיבוכיות המשוערת של הפונקציות בלבד `add_player`:**

בכל הפונקציות בהן הסיבוכיות היא משוערת בלבד הפונקציה `add_player` יש קריאות לפונקציות `find`, `union`, `find`, `union` ועל מנת בנה הנתונים `find`, `union` בעל עצים הפוכים, איחוד לפי גודל וכיום מסלולים. לכן, לפי מה שנאמר בהערות בהתחלה על הסיבוכיות, הפונקציות הללו משוערכות יחד ככה שאכן הסיבוכיות המשוערת שלהם תכילה $O(n^2 \log n)$.