רעיון כללי :

האובייקטים שמשתמשים בהם :

1) PLAYER: אובייקט שחקן שנמשור בו את השדות: playerid , שדה עזר לחישוב partial_spirit , שדה עזר לחישוב games_played , מצביע לחוליה NODE , מצביע NEXT שמצביע לשחקן שבה אחריו באותו תא בטבלת הערבול ומצביע FATHER שמצביע לצומת האב בUNION FIND של השחקנים בקבוצה שלו ו שדה בוליאני שמציין אם הוא שוער או לא ושדה לכרטיסים שלו.

2)TEAMBYID :

מחלקה של הקבוצה לפי המזהה שלה כך שבה נשמור את השדות : המזהה של הקבוצה והיכולת שלה וגם כן הניקוד הנוכחי של הקבוצה , מספר השחקנים ומספר השוערים בקבוצה ומצביע לחוליה NODE והרוח העדכני של הקבוצה הזו(ה SPIRIT) .

3)TEAMBYABILITY :

אובייקט של הקבוצה לפי היכולת שבו נשנור שני שדות : המזהה של הקבוצה והיכולת שלה .

4) NODE :

חוליה NODE שמחברת בין הקבוצה בעץ הקבוצות לבין השורש של עץ ההפוך של השחקנים ששיך לקבוצה זו ,גם כן נשמור בחוליה זו מספר המשחקים של הקבוצה .

נחזיק במבנה שלנו :

1-hash_table דינמי לשחקנים בשיטת chain_hashing, כלומר מערך של שחקנים כך שאם מלינו את המערך נקצה מערך בגודל פי שתיים מהמערך שלנו ונעביר את השחקנים כמו שצריך(כך שפקטור העומס יהיה לכל היותר 2).

2-עץ AVL דרגות של הקבוצות כך שהמפתחות הן ability של כל קבוצה .

3- עץ AVL של הקבוצות שממוין לפי ה ID .

הערה :

לכל שחקן ב HASH קיים שדה FATHER שמצביע לשחקן אחר בעץ ההפוך של השחקנים באותה קבוצה , ואם הוא השורש (כלומר השורש של העץ ההפוך) אזי ה FATHER מצביע ל NULL. כלומר לכל קבוצה יש עץ הפוך ששיך לשחקנים בקבוצה זו .

לכל קבוצה קיימת בעץ ה ID יש מצביע לחוליה NODE ולחוליה זאת גם כן יש מצביע לקבוצה(מצביע דו כיווני) ,כמו כן בחוליה זאת יש מצביע לשורש של השחקנים בעץ ההפוך(וגם לשורש בעץ ההפוך יש מצביע לחוליה).

וגם כן הכל חוליה כזו נשמור את מספר המשחקים של הקבוצה .

הערה לקיבוץ מסלולים : אנחנו מחזיקים את שני שדות שמשתמשים בהם בכדי לחשב את ה gamesplayed ו- partial_spirit לכל שחקן כמו שלמדנו בתרגול(בתרגיל הארגזים) ולכן כל פעם שעושים קיבוץ מסלולים ב- Union Find נעכן השדות האלה בהתאם .

* world_cup_t():

מאתחילים שני עצי AVL ריקים ו HASH TABLE לשחקנים להיות ריק .
סיבוכיות זמן : ביצוע מספר סופי של פעולות ולכן סיבוכיות הזמן של הפונקציה הוא O(1) .

*~world_cup_t() :
מוחקים כל השחקנים וכל הקבוצות ואחר כך מוחקים את המבנים שלנו (שתי עצי AVL ו-hash_table).
מחיקה של עץ AVL הוא נעשה בסיור POST ORDER ולכן נעשה בסיבכיות זמן $O(k)$ כך ש K מספר הקבוצות וכמו כן מחיקת n השחקנים בHASH TABLE גם כן נעשה ב $O(n)$ לכן פונקציה זו מתבצעת בסיבוכיות זמן $O(n+k)$.

*add_team(int teamId) :
בודקים אם $teamId \leqslant 0$:
-אם כן אז מחזירים INVALID_INPUT .
-אם לא אז בודקים אם קיימת קבוצה עם מזהה teamId בעץ AVL שממוין לפי id (חיפוש זה לוקח O(log(k))):
--אם כן אז מחזירים FAILURE .
--אם לא ,אז נכניס קבוצה חדשה עם מזהה teamId לשתי עצי ה-AVL שלנו (ability של קבוצה היא 0).
(הכנסה לעץ AVL לוקח O(log(k)) כך ש-k הוא מספר הקבוצות במערכת) , נאתחל את השדות בהתאם(הרוח להיות NEUTRAL מספר השחקנים 0 ..)
סיבוכיות זמן :פונקציה זו AVL לוקחת O(log(k)) כך ש-k הוא מספר הקבוצות במערכת .

:remove_team(int teamId)

קודם כל נבדוק אם teamId מספר שלילי , אם כן מתקיים אזי :
מחזירים INVALID INPUT .
אחרת אזי נעשה חיפוש בעץ של הקבוצות לפי ה ID לקבוצה בעלת ה ID הזה ואם לא מצאנו קבוצה כזו אזי נחזיר FAILURE , אחרת כלומר אם מצאנו את הקבוצה הזו אזי נסיר את הקבוצה הזו משני העצים של הקבוצות (של ה ID ושל ה ABILITY) וגם נשים את המצביעם של החוליה שהייתה שייכת לקבוצה זו ב NULLL ונסיר אותה ואז נחזיר SUCCESS . חיפוש והסרה משני העצים האלו נעשה בסיבוכיות זמן $log(k)$ כאשר k מספר הקבוצות הקיימות במערכת כיוון שהעצים הם מאוזנים .

add_player(int playerId, int teamId, const permutation_t &spirit, int gamesPlayed,int ability, int cards, bool goalKeeper):
נבדוק קודם אם הפרמטרים שקיבלנו תקינים :
כלומר אם מתקיים ש: $playerId \leqslant 0$ או $teamId \leqslant 0$ או $gamesPlayed < 0$ או $cards < 0$ או ש הפרמוטציה לא תקינה אז נחזיר INVALID INPUT .
אחרת , נבדוק נעשה חיפוש בעץ של הקבוצות לפי ה ID לקבוצה בעלת ה ID הזה ואם לא מצאנו קבוצה כזו אזי נחזיר FAILURE , וגם כן נבדוק אם היה שחקן בעל אותו מזהה בקבוצה זו מקודם אם מצאנו נחזיר FAILURE חיפוש זה נעשה ב $O(1)$ בממוצע על הקלט (כי אנחנו שומרים על פקטור עומס קבוע), אחרת נעשה הכנסה ל HASH TABLE של ה שחקנים (הכנסה זו מתבצעת ב $O(1)$) ונחבר אותו לשורש בעץ ההפוך של השחקנים (אם אין שורש אזי נשים אותו שורש ונחבר אותו לחוליה ששיכת לקבוצה שלו) , כמו כן נשים לב

שאתחול השדות של השחקן יהיו בהתאם לשורש של העץ ההפוך (כי GAMESPLAYED ו PARTIAL SPIRIT נחשב אותם לאורך המסלול מהשחקן ל שורש כמו בתרגול של ה UNION FIND שאלת הארגזים) , ואז נעדכן את השדות של הקבוצה בהתאם (נוסיף ל ABILITY של הקבוצה את היכולת של השחקן והרוח של הקבוצה תהיה הרוח שהייתה מקודם מורכבת עם הרוח של השחקן ונוסיף מספר השחקנים של הקבוצה באחד), אחרי העדכון של השדות של הקבוצה נעדכן את המיקום של הקבוצה בעץ הקבוצות לפי היכולת , ואז נחזיר SUCCESS .

///לכל m פעולות של פונקציה ADD PLAYER מתקיים ש : כל הכנסה של שחקן מתבצעת בסיבכיות זמן $O(1)$ ב HASH TABLE כי לכל פעולת הכנסה לטבלת הערבול היא תמיד נעשית בסיבוכיות זמן $O(1)$ חוץ מ כאשר צריך להגדיל את הטבלה ולהעתיק את התוכן ופונקצית הערבול בהתאם , וחיפוש לקבוצה שלו בתוך עץ הקבוצות תמיד נעשה בכל הכנסה ולכן סה"כ פונקציה זו מתבצעת בסיבוכיות זמן $O(log(k))$ בממוצע על הקלט .

*play_match(int teamId1, int teamId2) :

בודקים אם אם teamId1 ≤ 0,teamId2 ≤ 0 או teamId1 = teamId2 :

-אם כן אז מחזירים INVALID INPUT .

--אם לא ,אז מחפשים קבוצות בעץ AVL שממוינת לפי id על קבוצות עם מזהים teamId1,teamId2 ,(חיפוש זה לוקח O(log(k))):

-אם לא מצאנו אותם אז מחזירים FAILURE .

-אם אכן מצאנו אותן ,אז בודקים אם לכל קבוצה מהן יש לפחות שועיר אחד(בודקים ע"י שדה של הקבוצה שמהווה מספר השוערים בקבוצה (num_goal_keeper)) ,אם אחד מהן אין שועיר אז גם מחזירים FAILURE .

אחרת בודקים היכולת הכללית של הקבוצות :

-אם היכולת של הקבוצה הראשונה גדולה מהשנייה אז מוסיפים 3 נקודות של הקבותה הראשונה .

-אם היכולת של הקבוצה השנייה גדולה מראשונה אז מוסיפים 3 נקודות של הקבותה השנייה . .

-אם היכולות שוות אז מוסיפים נקודה לכל קבוצה .

ומוסיפים אחד ל-games_played לשתי הקבוצות בכל מקרה .

בכל קבוצה יש מצביע לחולייה חיצונית שיש בה שדה gamesplayed גם מוסיפים את זה ב-1 בכל לשתי הקבוצות. מחוליות אלה גם יש מצביע להשורש בקבוצה ב-Union Find נוסיף את שדה games_played שלו ב-1 .

סיבוכיות זמן :עשינו מספר סופי של הכנסות והוצאות לעץ AVL עם מספר סופי של פעולות עם סיבוכיות O(1) ,ולכן נקבל סה"כ סיבוכיות זמן של O(log(k)) כך ש-k מספר הקבוצות במערכת .

:num_played_games_for_player(int playerId)

תחילה נבדוק אם playerId שלילי אם כן אזי נחזיר INVALID INPUT אחרת נבדוק אם השחקן היה במערכת מקודם או אם הוא במערכת כרגע אם לא נמצא אזי נחזיר FAILURE (חיפוש זה נעשה בסיבוכיות זמן $O(1)$ בממוצע על הקלט כי אנחנו ב HASH שומרים על פקטור העומס , אם כן נמצא אזי נתחיל לעלות עד

השורש בעץ ההפוך ונסכום את השדה הנוסף(של המשחקים) עד ההגעה לשורש , הסכום הזה הוא יהיה את ערך החזרה
אחרי שמירת ערך החזרה נחזור על המסלול מחדש ונכווץ אותו .ו ובסוף מחזירים ערך ההחזרה שלנו .
חיפוש Find וקיבוץ מסלולים לוקח סיבוכיות זמן משוערך $log^*(n)$ משוערך, בממוצע על הקלט כי משתמשים ב-union find כפי שנלמד בהרצאה .

אם לא קיים שחקן כזה אבל השדה במערכת נחזיר FAILURE , לכן פעולה זאת מתבצעת בסיבוכיות זמן $log^*(n)$ משוערך בממוצע על קלט כיוון שבהינתן m פעולות (שבהם הזמן משוערך חוץ מהוספת שחקן) בגלל ש אנחנו בכל פעם שעושים FIND גם כן מכווצים את המסלול .

* add_player_cards(int playerId, int cards) :
נבדוק אם playerId ≤ 0 : אם כן אז נחזיר INVALID_INPUT .
אם לא אז נחפש בפעולת Find שחקן עם מזהה playerId ב-Union Find אם לא מצאנו אותו אז נחזיר FAILURE .אם כן קיים שחקן כזה אבל השדה active בהחולייה שהוא מצביע אליהה שווה ל-false כלומר הקבוצה שהשחקן שייך אליהה הודחה ולכן נחזיר FAILURE . אחרת אז מוסיפים לשחקן מספר cards להשדה של הכרתיסים שלו cards . בחיפוש השני נעשה קיבוץ מסלולים ונחזיר SECCESS .
עשינו בכל חיפוש קיבוץ מסלולים ב- Union Find בכל מתודה שבה צריך זמן משוערך ולכן סיבוכיות הזמן המשוערך הוא $log^*(n)$ כאשר n מספר השחקנים במערכת (כולל שחקני עבר) .

* get_player_cards(int playerId) :
נבדוק אם playerId ≤ 0 : אם כן אז נחזיר INVALID_INPUT .
אם לא אז נחפש HASH TABLE שחקן עם מזהה playerId אם לא מצאנו אותו אז נחזיר FAILURE .אם כן קיים שחקן כזה אז מחזירים את מספר הכרתיסים שלו cards .
משום שפקטור העומס שלנו ב-HASH TABLE תמיד קטן שווה 2 אז החיפוש השחקן לוקח בממוצע O(1) ולכן סיבוכיות הפונקציה הוא O(1) בממוצע על הקלט .

* get_team_points(int teamId) :
נבדוק אם teamId ≤ 0, אם כן אז מחזירים INVALID_INPUT .
אם לא אז נחפש קבוצה עם מזהה teamId בעץ AVL שממוין לפי id, ואם אין קבוצה כזו אז נחזיר FAILURE .
אם כן יש קבוצה כזו אז נחזיר הנקודות שלה points .
פונקציה זו לוקחת O(log(k)) סיבוכיות זמן כאשר k הוא מספר הקבוצות במערכת, כי עשינו פעולת חיפוש בעץ AVL בגודל k וגם מספר סופי של פעולות .

*get_ith_pointless_ability(int i) :
נבדוק אם i < 0 או i גדול יותר ממספר הקבוצות (ע"י size של עצי AVL) ,אם כן אז מחזירים INVALID_INPUT .
אם לא אז עושים חיפוש בעץ דרגות AVL שממוין לפי ability של הקבוצות ומחזירים points של הקבוצה במקום ה-i על ידי פעולת SELECT כפ שנלמדה בהרצאה , פעולה זו מתבצעת בסיבוכיות O(log(k)) במקרה

הגרוע ביותר כך ש K מספר האיברים בעץ .
פונקציה זו לוקחת O(log(k)) סיבוכיות זמן כאשר k הוא מספר הקבוצות במערכת ,כי עשינו פעולת חיפוש ו SELECT בעץ AVL בגודל k וגם מספר סופי של פעולות .

*get_partial_spirit(int playerId) :
נבדוק אם playerId ≤ 0, אם כן אז מחזירים INVALID_INPUT .
אם לא אז עושים Find ב-HASH TABLE של השחקן עם מזהה playerId, אם אין שחקו כזה או שהקבוצה שלו הודחה (בודקים זה בחולייה זמצביע עליהה השורש כמו קודם)אז מחזירים FAILURE .
אחרת מחזירים את הרכבת ה-spirit בסמלול העץ ההפוך של השחקן שמצאנו (כמו שעשינו בתרגול) ומחזירים את הפרמותציה היוצאת .ונעשה קיבוץ מסלולים ונעדכן כמובן את השדות בהתאם כפי שנלמד בתרגול.
חיפוש ב טבלת ערבול שבה פקטור העומס קבוע וקיבוץ מסלולים לוקח סיבוכיות זמן משוערך $log^*(n)$ משוערך, בממוצע על הקלט כי משתמשים ב- ו UNION FIND, hash_table .

*buy_team(int buyerId, int boughtId) :
בודקים אם אחד ה-id-ים אינו חיובי או שהם שווים, אם כן אז מחזירים INVALID_INPUT .
אם לא אז מחפשים שתי הקבוצות בעץ AVL שממוין לפי id, אם אחד מהן לא קיימת אז מחזירים FAILURE .
אחרת אז ניגשים להחוליות שמצביעות עליהן הקבוצות שהן ניגשים להשחקן בשורש ה-Union של כל קבוצה .
בודקים איזו קבוצה עם מספר שחקנים יותר קטן ואז גורמים להשורש של קבוצה זו להצביע על השורש של הקבוצה הגדולה . מעדכנים את gamesplayed ו-partial_spirit של השורשים בהתאם (כמו בתרגול).
גורמים לשורש החדש שלנו ליצביע על החולייה של הקבוצה הקונה(buyer) ונשנה השדה active ל-false בחולייה של הקבוצה הנקנית(bought) (כי עכשיו שקבוצה לא פעילה).
עשינו חיפוש בעץ AVL בגודל k (מספר הקבוצות), עם פעולות על Union Find .
ולכן נקבל סיבוכיות זמן $O\left(log(k) + log^*(n)\right)$ משוערך , כאשר n מספר השחקנים ,k מספר הקבוצות .