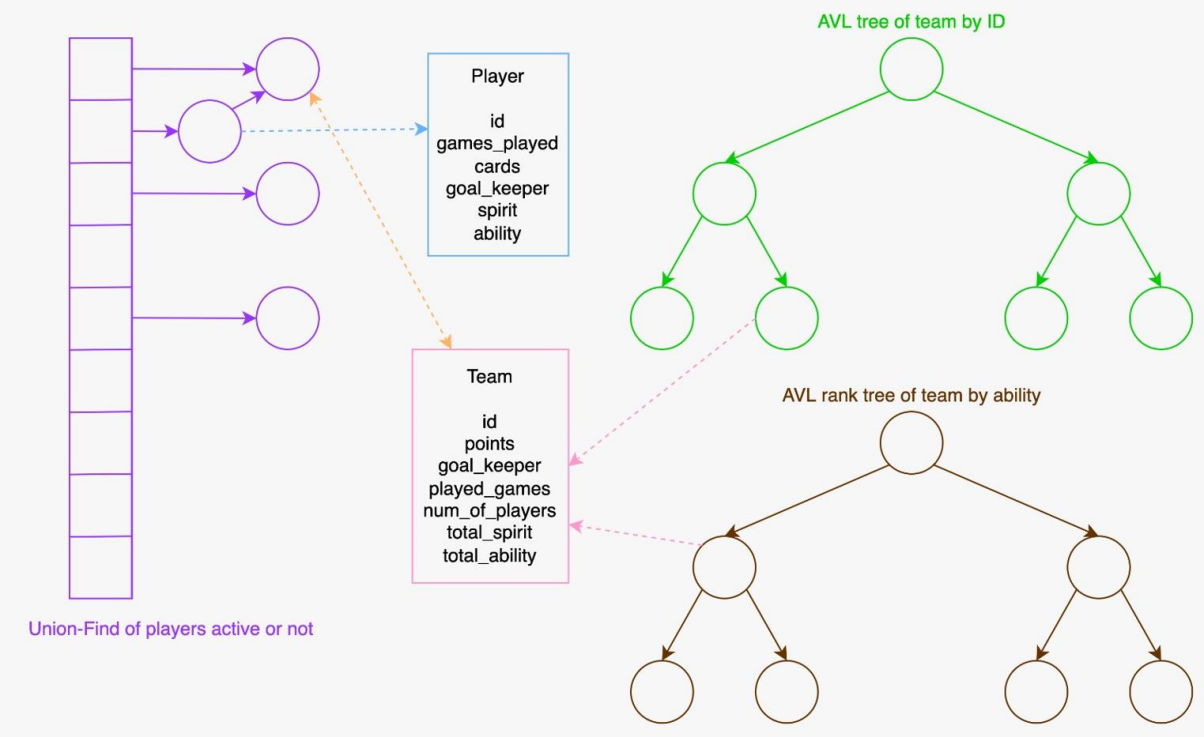
רטוב 2 – מבני נתונים



הסבר כללי על המבנה:

המבנה שלנו מכיל:

- אובייקט של שחקן – יכיל את השדות שהתבקש להכיל בתרגיל. לאורך התרגיל נשתמש במילה "קפטן" על מנת לתאר את השורש של העץ ההפוך, כלומר את השחקן שממנו ניתן לגשת לקבוצה הרלוונטית.
- אובייקט של קבוצה – תכיל את המשתנים הכתובים כאן למעלה. נשים לב שעל מנת לעמוד בסיבוכיות שנדרשה מאיתנו, שמרנו שדות נוספים באובייקט זה כמו למשל total_spirit, total_ability, goal_keeper_status.
- 2 עצי דרגות AVL המכילים מצביעים לכלל הקבוצות הקיימות במערכת בכל שלב - אחד ממוין לפי id והשני לפי יכולת. במבנה של עצים אלו מימשנו כמו ברטוב 1 את פונקציות ההכנסה וההוצאה לעץ, כולל גלגולים ועדכון של גובה וBF. בנוסף לכך, מימשנו גם פונקציית select שנעזרנו בה בתרגיל זה. המימוש התבצע כפי שהוסבר בתרגול. נדגיש גם כי הוספנו שדה של weight לכל צומת על מנת לחשב את הדרגה שלה.
- מערך hash table דינאמי שמכיל את השחקנים הקיימים במערכת, כאשר פקטור העומס לפיו אנו מגדילים את המערך מאפשר מצב שבו יש שחקן אחד בכל תא (בממוצע), כך שהסיבוכיות שנדרשה בתרגיל זה מתאפשרת כאשר אנו נעזרים במערך זה (למשל שחיפוש שחקן יהיה בסיבוכיות O(1) ממוצע, כי מדובר בחיפוש בhash table).
  - במימוש של מבנה זה, מימשנו גם מבנה של רשימה (list), מפני שהשתמשנו בשיטת chain hashing (בהמלצת הסגל בפיאצה 😊).
- Union find – במבנה זה השתמשנו במערך השחקנים הדינאמי. כל תא ברשימה ששייכת לhash table, הכיל union find node של שחקן מסוים. כך יכולנו לשייך כמה שחקנים לאותה קבוצה ולשמר את הקשר ביניהם. כך גם יכולנו "לעלות במעלה העץ ההפוך" של הקבוצה. כפי שנציין בהמשך, בכל מקום שבו השתמשנו במבנה זה, שערכנו ביחד עם כל שאר הפונקציות המשתמשות במבנה את הסיבוכיות של המימוש.

נקודות חשובות על מימוש התרגיל:

- נציין כי עבור כל אחת מהפונקציות, בדקנו תחילה את תקינות הקלטים.
- כפי שנתבקשנו, את שערוך הפונקציות (למעט זו של הוספת שחקן), חישבנו ביחד עם כל שאר הפונקציות – כל הפונקציות שמשתמשות בפעולות של union-find, שערכנו יחד, מפני שכפי שלמדנו, השערוך מתבצע על סדרה של

פעולות. בכל פונקציה שבה קיימת סיבוכיות משוערכת וכתבנו שהסיבוכיות היא למשל לוג-סטאר, הכוונה היא שהיא משוערכת ביחד עם שאר הפונקציות שמשתמשות בפעולות אלו במערכת.

- במימוש פונקציית find בUF, השתמשנו בכיווץ מסלולים ובמימוש של union באיחוד לפי גודל, זאת על מנת לעמוד בסיבוכיות שנדרשה בתרגיל כפי שלמדנו בקורס.
- שימור ועדכון ערכי partial_spirit ו- games_played: ראשית נציין כי את 2 ערכים אלו שמרנו באותה שיטה, רק שעבור מספר המשחקים השתמשנו בחיבור וחיסור, ועבור spirit השתמשנו בכפל וכפל בהופכי. על מנת לשמור על המידע העדכני, ביצענו את העדכון בכמה פונקציות שונות. נפרט בכל אחת מהפונקציות המעורבות בעדכון כיצד עדכנו. נדגיש כי את הפירוט של החישוב של ספיריט, נכתוב לפי הסדר של הכפל (כי יש חשיבות לסדר).
בנוסף נציין, כי המימוש יהיה בעזרת הדרך שהוצגה בתרגול 9 ב"שאלת הארגזים".
- נציין כי במהלך כיווץ מסלולים (בתוך פונקציית find) אנו מעדכנים גם כן את ערכי ערכי partial_spirit ו- games_played: עבור כל node שלא היה מחובר בעבר לשורש (כלומר ששינינו לו את האבא), נעדכן באופן הבא:
  - spirit– ההופכי של האבא החדש (הקפטן), כפול הpartial spirit של האבא הישן שלו (אותו נחשב בעזרת פונקציית עזר שמבצעת פעולה דומה לזו של get_partial_spirit, עליה נפרט בהמשך), כפול הספיריט שלו עצמו.
  - Played games – מספר המשחקים הנוכחי של השחקן, פלוס מספר המשחקים הכולל של האבא הישן שלו (אותו נחשב בעזרת פונקציית עזר שמבצעת פעולה דומה לזו של get_num_played_games),פחות מספר המשחקים של הקפטן (האבא החדש שלו). אם הקבוצה שלו קיימת (כלומר לא נמחקה מהמערכת), נבצע גם: פחות מספר המשחקים הכולל של הקבוצה.

world_cup_t():

מטרת הפונקציה:
אתחול מבנה נתונים ריק.

סיבוכיות נדרשת:
O(1) במקרה הגרוע.

מימוש:
ניצור ונאתחל בעזרת constructors את הבאים:

- ניצור 2 עצי דרגות AVL עבור הקבוצות – עץ קבוצות ממוין לפי id, ועץ קבוצות ממוין לפי ability.
- ניצור בנוסף מבנה של union-find עבור השחקנים.

נכונות הסיבוכיות:
יצירת המבנים ע"י קריאה לconstructor היא בסיבוכיות של O(1) עבור כל אחד מהם. לכן סה"כ **O(1)**.

virtual ~world_cup_t():

מטרת הפונקציה:
שחרור מבנה הנתונים כולו.

סיבוכיות נדרשת:
O(n+k), כאשר n הוא מספר השחקנים הכולל וk הוא מספר הקבוצות מאז תחילת המערכת.

מימוש:

- נבצע שחרור ל2 עצי הדרגות (והשחקנים שבהם אם קיימים כאלה) ונבצע שחרור גם למבנה של union-find, כולל המשתנים הקיימים במבנה זה .(נבחין כי אלו אותם מבנים שהקצנו בבנאי של worldcup).

נכונות הסיבוכיות:
על מנת לשחרר כל אחד מהעצים, נבדוק האם קיימות בכל אחד מהם קבוצות. אם כן, נעבור על כל אחד מהעצים – בעץ הראשון נמחק רק את הnodes שלו, ובעץ השני נמחק גם את האובייקטים של הקבוצות עצמן (בנוסף לnodes בעץ). – מעבר על כל אחד מהעצים ומחיקת האיברים היא בסיבוכיות O(k), כאשר k מספר הקבוצות.
קריאה להורס של מבנה ה-union find היא ב-O(1), שם יקרא ההורס של הhash table שמכילה בתוכה את הnodes של

השחקנים. ההורס של hash הוא בסיבוכיות של מקסימום O(n), כי גודל המערך הוא מקסימום כגודל השחקנים. בכל תא בhash table יהיה שחקן אחד (לפי הגדרת hash), לכן מחיקת שחקן מכל אחד מהתאים תהיה O(1).
לכן סה"כ, סיבוכיות הפונקציה של ההורס היא $\boldsymbol{O(n)+O(k)=O(n+k)}$.

StatusType add_team(int teamId):

מטרת הפונקציה:
הוספת קבוצה חדשה למבנה הנתונים, בעת הכנסתה אין שחקנים בקבוצה.

סיבוכיות נדרשת:
O(logk), כאשר k הוא מספר הקבוצות במערכת.

מימוש:

- יצירת קבוצה חדשה ריקה בעלת המזהה הנתון
- הוספת הקבוצה לעץ הקבוצות הכולל הממוין לפי יכולת ואתחול שדה ability להיות 0.
- הוספת הקבוצה לעץ הקבוצות הכולל הממוין לפי id.

נכונות הסיבוכיות:
יצירת קבוצה חדשה היא ב(O(1 כפי שלמדנו. הוספתה לכל אחד מעצי הקבוצות היא בסיבוכיות O(logk), כפי שראינו גם ברטוב 1. לכן סה"כ, הסיבוכיות של פונקציה זו היא: $O(1)+O(logk)\leq \boldsymbol{O(logk)}$.

StatusType remove_team(int teamId):

מטרת הפונקציה:
הוצאת קבוצה קיימת מהמערכת.

סיבוכיות נדרשת:
O(logk), כאשר k הוא מספר הקבוצות במערכת.

מימוש:

- נחפש את הקבוצה בעץ הקבוצת ע"פ id. אם לא נמצא אז הקבוצה לא קיימת במערכת וסיימנו.
- אם מצאנו את הקבוצה נבדוק תחילה האם יש לה שחקנים.
- אם אין לה שחקנים, נוציא את הצמתים של הקבוצה משני עצי הקבוצות ,נשחרר את המחלקה team וסיימנו.
- אם יש לה שחקנים, נערוך את השדה games_played של השורש של העץ, כך שיהיה שווה לערך של מספר המשחקים שהקבוצה שיחקה עד כה.
- נשחרר את הצומת של קבוצה זו מעצי הקבוצות הכוללים.
- כעת נמחק את האובייקט של הקבוצה וסיימנו.

נכונות הסיבוכיות:

חיפוש בעץ ע"פי מזהה היא פעולה של O(logk) כאשר k הוא מספר הצמתים בעץ. בדיקת קיום שחקנים היא בסיבוכיות O(1). גישה לשדה ספציפי בקבוצה ועדכונו היא בסיבוכיות O(1), שחרור הצומת של הקבוצה מכל אחד מהעצים היא בסיבוכיות O(logk).סה"כ, הסיבוכיות של פונקציה זו היא $O(1)+O(\log k)\leq \boldsymbol{O(\log k)}$.

StatusType add_player(int playerId, int teamId, const permutation_t &spirit, int gamesPlayed, int ability, int cards, bool goalKeeper)

מטרת הפונקציה:
הוספת שחקן חדש למבנה הנתונים עם כל הפרמטרים המתקבלים בפונקציה.

סיבוכיות נדרשת:
O(logk) משוערך בממוצע על הקלט ,k הוא מספר הקבוצות במערכת.

מימוש:

- נוודא שהשחקן לא קיים כבר במערכת ע"י חיפוש במערך השחקנים. אם קיים – סיימנו.
- נחפש בעץ הקבוצות הכולל לפי id את הקבוצה שהתקבלה כפרמטר. אם היא אינה קיימת (כלומר לא הייתה מעולם במבנה או הייתה והפכה ללא פעילה) – סיימנו.
- ניצור אובייקט של שחקן עם הפרמטרים שהתקבלו בפונקציה.
- נעדכן את שדה num_of_nodes , את total_spirit ואת total_ability של הקבוצה המתאימה.
  - לאחר עדכון השדות נבצע הוצאה ואז הכנסה של הקבוצה מעץ הקבוצות לפי ability, על מנת לשמר את האיזון של העץ.
- ניצור צומת שתצביע לאובייקט של השחקן.
- במידה וזהו השחקן הראשון בקבוצה שלו, נגדיר אותו כקפטן של הקבוצה, כלומר הוא יהיה השורש של עץ השחקנים ב-Union find. במקרה זה:
  - עדכון partial spirit – נאתחל את ה total spirit של הקבוצה ואת ה-partial spirit של השחקן, להיות הספיריט שקיבלנו כקלט.
  - עדכון games played – נאתחל לפי הקלט שקיבלנו.
- אם הוא לא קפטן:
  - עדכון partial spirit – (העדכון מתבצע לפי הסדר שכתוב כאן למטה)
    - נעדכן את הספיריט הכולל של הקבוצה להיות הערך הנוכחי שלו, כפול הקלט של הספיריט של השחקן.
    - נאתחל לפי ההופכי של הספיריט של הקפטן (שהוא אבא שלו במקרה זה), כפול הספיריט הכולל של הקבוצה.
  - עדכון games played – נאתחל לפי הקלט שקיבלנו, פחות מספר המשחקים של הקפטן של הקבוצה (שהוא האבא שלו במקרה זה), פחות מספר המשחקים הכולל של הקבוצה.
- אם שחקן זה הוא שוער, נשנה את הסטטוס הרלוונטי עבור הקבוצה אליה הוא מתווסף.
- כעת נוסיף את הצומת שיצרנו לunion-find ע"י הוספת מצביע לצומת במערך השחקנים.

נכונות הסיבוכיות:

בדיקה האם השחקן קיים היא בסיבוכיות O(1) בממוצע. חיפוש הקבוצה של השחקן בעץ הקבוצות היא בסיבוכיות O(logk). יצירת אובייקט וצומת, ועדכון השדות המתאימים בקבוצה הם בסיבוכיות O(1). הגדרת השחקן כקפטן (במידת הצורך), כולל גישה למצביע ושינוי ערכו, הן בO(1). הוספת השחקן למערך השחקנים היא בסיבוכיות O(1) משוערך, בממוצע על הקלט, כפי שנכתב בפיאצה. סה"כ הסיבוכיות של הפונקציה תהיה **O(logk) משוערך, בממוצע על הקלט.**

output_t<int> play_match(int teamId1, int teamId2)

מטרת הפונקציה:

תחרות בין 2 קבוצות ובחירה מי מהן מנצחת.

סיבוכיות נדרשת:

O(logk) במקרה הגרוע, כאשר k מספר הקבוצות במערכת.

מימוש:

- ראשית נחפש את הקבוצות בעץ הקבוצות לפי id, על מנת לראות שאכן קיימות. אם לא – סיימנו.
- אם קיימות, עבור כל אחת מהן נבדוק שיש להן שוער (כלומר שהן קבוצות חוקיות). אם לא – סיימנו.
- אם קיים שוער לשתיהן – ניגש לשדות points, team_ability של כל אחת מהקבוצות ונשווה את הערכים עפ"י הנוסחא שניתנה בתרגיל. (כלומר, אם יש שיוויון בנוסחא, נחשב את ה"כוח הרוחני" ונשווה אותו).
- הקבוצה המנצחת תקבל 3 נקודות תוספת. אם יש תיקו, שתי הקבוצות יקבלו תוספת של נקודה אחת.
- נעדכן גם את מספר המשחקים של כל קבוצה.
- בהתאם להגדרת הפונקציה בתרגיל נחזיר את ערך החזרה הרלוונטי.

נכונות הסיבוכיות:

חיפוש בעץ הקבוצות לפי id הוא בסיבוכיות O(logk). גישה לשדה האם יש שוער, ואם כן אז לשדות של הנקודות והיכולת, כל

אחד מהם בסיבוכיות O(1). הוספת נקודות ועדכון מספר המשחקים הוא גם ב(O(1. חישוב ה"כוח הרוחני" במידת הצורך הוא בסיבוכיות של O(1), עפ"י הפונקציה שניתנה בutils.
לכן סה"כ, הסיבוכיות של פונקציה זו היא $O(1) + O(\log k) \leq \boldsymbol{O(\log k)}$.

output_t<int> num_played_games_for_player(int playerId)

מטרת הפונקציה:
החזרת מספר המשחקים הכולל בהם השתתף השחקן, בין אם פעיל או לא.

סיבוכיות נדרשת:
O(log*n) משוערך בממוצע על הקלט, n מספר השחקנים במערכת (כולל שחקני עבר).

מימוש:

- נחפש את השחקן במערך השחקנים. אם לא קיים – סיימנו.
- אם השחקן קיים, נסכום את מספר המשחקים שקיימים אצל השחקן, ביחד עם מספר המשחקים של כל אחד מהשחקנים שנמצאים מעליו בקבוצה. (לשם כך יש לעלות עד לשורש, כולל השורש). בנוסף נסכום גם את מספר המשחקים הכולל של הקבוצה, שיכול להתעדכן במהלך הפעלת play_match.
- נשים לב כי ערכים אלו משתנים בהתאם לקנייה של קבוצות או ביצוע find, שבמהלכו מתקיים כיווץ מסלולים.
- נחזיר את הסכום המבוקש.

נכונות הסיבוכיות:
חיפוש השחקן בסיבוכיות O(1) בממוצע. גישה לפרטי השחקנים עד השורש בסיבוכיות O(log*n) משוערך, גישה לשדה המתאים אצל כל אחד מהשחקנים בסיבוכיות O(1). גישה לשדה של מספר המשחקים בקבוצה דרך הקפטן הוא ב-O(1).סה"כ סיבוכיות הפונקציה $O(1) + O(\log^* n) \leq \boldsymbol{O(\log^* n)}$ **משוערך, בממוצע על הקלט.**

StatusType add_player_cards(int playerId, int cards)

מטרת הפונקציה:
עדכון שדה cards של השחקן.

סיבוכיות נדרשת:
O(log*n) משוערך, בממוצע על הקלט.

מימוש:

- ראשית נבדוק האם השחקן קיים ע"י גישה למערך השחקנים. אם לא קיים – סיימנו.
- אם קיים, נעלה במעלה העץ , בעזרת פונקציית find על מנת לבדוק האם הקבוצה פעילה – אם לא, סיימנו.
- אם הקבוצה פעילה, ניגש לשחקן במערך השחקנים,ואז ניגש לשדה cards של השחקן ונעדכן אותו.

נכונות הסיבוכיות:
בדיקת קיום השחקן היא ב(O(1 בממוצע. בדיקה האם הקבוצה פעילה היא ב(O(log*n משוערך. עדכון השדה הוא בסיבוכיות O(1). $O(1) + O(\log^* n) \leq \boldsymbol{O(\log^* n)}$ **משוערך בממוצע על הקלט**.

output_t<int> get_player_cards(int playerId)

מטרת הפונקציה:
החזרת הכרטיסים של השחקן, בין אם פעיל או לא.

סיבוכיות נדרשת:
O(1) בממוצע על הקלט.

מימוש:

- נמצא את השחקן במערך השחקנים הכולל. אם לא קיים – סיימנו.
- אם קיים, ניגש לשדה הכרטיסים שלו ונחזיר אותו

נכונות הסיבוכיות:
מציאת השחקן במערך השחקנים הוא בסיבוכיות O(1) בממוצע. גישה לשדה הכרטיסים בסיבוכיות O(1). סה"כ, סיבוכיות הפונקציה היא **O(1) בממוצע על הקלט**.

output_t<int> get_team_points(int teamId)

מטרת הפונקציה:
החזרת מספר הנקודות של הקבוצה.

סיבוכיות נדרשת:
O(logk) במקרה הגרוע, כאשר k מספר הקבוצות במערכת.

מימוש:

- ניגש לעץ הקבוצות הממוין לפי id ונחפש שם את הקבוצה הרלוונטית. אם לא קיימת – סיימנו.
- אם קיימת, ניגש לשדה הנקודות של קבוצה זו ונחזיר את ערכו.

נכונות הסיבוכיות:
חיפוש הקבוצה בעץ הקבוצות הכולל היא בסיבוכיות O(logk). גישה לשדה הנקודות והחזרת ערכו היא ב-O(1).
לכן סה"כ הפונקציה בסיבוכיות $O(1) + O(\log k) \leq \boldsymbol{O(\log k)}$.

output_t<int> get_ith_pointless_ability(int i)

מטרת הפונקציה:
החזרת הקבוצה במקום i אם ממיינים את כל הקבוצות לפי היכולת שלהם.

סיבוכיות נדרשת:
O(logk) במקרה הגרוע, k מספר הקבוצות.

מימוש:

- נוודא שהאינדקס שקיבלנו הוא שייך לטווח של כמות האיברים בעץ.
- נחפש את האיבר שהאינדקס שלו הוא i בעץ הקבוצות הממוין לפי ability. נעשה זאת בעזרת פונקציית select(i) שמימשנו בעץ הדרגות. נציין כי לפונקציה זו נשלח את i+1, מפני שהאינדקס שלנו מתחיל מ0, ואילו דרגות בעץ מתחילות מ-1.
  - מימוש פונקציית select: תהיה על בסיס האלגוריתם שנלמד בתרגול, כך שמשנים את ערך האינדקס שמחפשים בכל איטרציה של הלולאה, ולפי המשקל של הבן השמאלי יודעים לאן להמשיך את האיטרציה בעץ הממוין (לתת העץ השמאלי/הימני).
- נחזיר את הערך שפונקציית select תחזיר.

נכונות הסיבוכיות:
פונקציית select היא כפי שלמדנו בסיבוכיות O(logk), לכן סה"כ הפונקציה בסיבוכיות **O(logk)**.

output_t<permutation_t> get_partial_spirit(int playerId)

מטרת הפונקציה:
חישוב ה"רוח" של הקבוצה בה חבר השחקן שניתן כפרמטר באופן חלקי. השחקנים שנלקחים בחשבון הם אלה שהצטרפו לפני השחקן עם id הניתן כולל אותו.

סיבוכיות נדרשת:
O(log*n) משוערך, בממוצע על הקלט.

מימוש:

- ניגש לשחקן שהתקבל במערך השחקנים. אם הוא לא קיים – סיימנו.
- נוודא שהקבוצה של השחקן קיימת (כלומר לא הוסרה מהתחרות) בעזרת פונקצית find. אם לא קיימת – סיימנו.

- נעבור על כל אחד מהשחקנים, החל מהשחקן שקיבלנו ועד השורש, ונבצע הרכבה בעזרת מחלקת פרמוטציה, של כל הspirit של השחקנים דרכם אנו עוברים. נציין כי כאשר קבוצה נקנית, אנחנו מעדכנים את הערך של השורש של אותה קבוצה, כך שכל אחד מהשחקנים בקבוצה המאוחדת עדיין יהיה כפולה של הערכים הנכונים והמתאימים (כלומר לפי הסדר הכרונולוגי) עבור partial spirit שלו.
- נחזיר את הערך הסופי שהוא התוצאה של ההרכבות.

נכונות הסיבוכיות:

גישה לשחקן במערך השחקנים היא O(1) בממוצע. וידוא שהקבוצה קיימת ולאחר מכן מעבר על כל אחד מהשחקנים במעלה העץ הן בסיבוכיות משוערכת (ביחד עם שאר הפונקציות הרלוונטיות כפי שכתבנו בהתחלה) של O(log*n) משוערך.ביצוע כל אחת מההרכבות היא בO(1). לכן סה"כ, הסיבוכיות של פונקציה זו היא
$O(1) + O(\log^* n) \leq \boldsymbol{O(\log^* n)}$ **משוערך בממוצע על הקלט**.

StatusType buy_team(int buyerId, int boughtId)

מטרת הפונקציה:

קבוצה אחת קונה את הקבוצה השנייה. מזהה את הקבוצה המאוחדת הוא של הקבוצה הקונה.

סיבוכיות נדרשת:

O(logk+log*n) משוערך, k מספר הקבוצות במערכת ו n הוא מספר השחקנים הכולל (כולל שחקני עבר).

מימוש:

- נחפש את 2 הקבוצות בעץ הקבוצות הכולל לפי id. אם אחת מהן לא קיימת – סיימנו.
- אם שתיהן קיימות, נקרא לפונקציית union_teams, שמתבצעת באופן הבא:
  - אם ל-2 הקבוצות אין שחקנים, סיימנו (כי אם אין לה שחקנים, זה אומר שמעולם לא היו לה שחקנים – כלומר מעולם לא שיחקה באף משחק ולכן אין מה לעדכן עבור הקבוצה הקונה).
  - אם הקבוצה הקונה היא הגדולה יותר (כלומר יש לה יותר שחקנים) –
    - אם לקבוצה הנקנית אין שחקנים, סיימנו.
    - אחרת, נחבר את השורש של הקבוצה הנקנית לשורש של הקבוצה הקונה.
    - נעדכן את מספר המשחקים ששוחקו עבור הקבוצה הקונה.
    - עדכון partial spirit – (העדכון מתבצע לפי הסדר שכתוב כאן למטה)
      - נעדכן את הקפטן של הקבוצה הנקנית להיות : ההופכי של האבא החדש שלו (הקפטן של הקונה), כפול הספיריט הכולל של הקבוצה הקונה, כפול הספיריט שלו עצמו.
      - נעדכן את הספיריט הכולל של הקונה להיות : הספיריט הכולל שלו כפול הספיריט של הקבוצה הנקנית.
    - עדכון games played – נעדכן את הקפטן של הקבוצה הנקנית להיות: מספר המשחקים שלו, פלוס מספר המשחקים הכולל של הקבוצה הנקנית, פחות מספר המשחקים הכולל של הקונה, פחות מספר המשחקים של הקפטן של הקונה (הקפטן החדש של הקבוצה המאוחדת).
  - אם הקבוצה הקונה היא הקטנה יותר (יש לה פחות שחקנים) –
    - נגדיר את הקפטן של הקבוצה הנקנית להיות הקפטן החדש.
    - אם בקבוצה הקונה יש שחקנים, נחבר בין השורש של הקבוצה הקונה לשורש של הנקנית.
    - נעדכן את מספר המשחקים ששוחקו עבור השורש של הקבוצה הקונה כך שבסכימה שלו עם המשחקים של הקבוצה שנקנית, נקבל באמת את המספר המקורי ששחקן בקבוצה הקונה שיחק.
    - עדכון partial spirit – (העדכון מתבצע לפי הסדר שכתוב כאן למטה)
      - נעדכן את הספיריט של הקפטן של הקבוצה הנקנית להיות : הספיריט הכולל של הקבוצה הקונה, כפול הספיריט שלו עצמו.
      - נעדכן את הספיריט של קפטן של הקבוצה הקונה להיות: ההופכי של הקפטן של הנקנית (כמובן לאחר ביצוע הנקודה לעיל), כפול הספיריט של עצמו.
      - נעדכן את הספיריט הכולל של הקונה להיות : הספיריט הכולל שלו כפול הספיריט של הקבוצה הנקנית.
    - עדכון games played –

- נעדכן את הערך של הקפטן של הקבוצה הקונה – הערך שלו, פלוס מספר המשחקים של הקבוצה הקונה, פחות מספר המשחקים של הקפטן החדש (שהוא הקפטן של הנקנית), פחות מספר המשחקים הכולל של הנקנית.
- נעדכן את הקפטן החדש (הקפטן של הנקנית) – הערך שלו, פלוס מספר המשחקים הכולל של הקבוצה הנקנית, פחות מספר המשחקים הכולל של הקבוצה הקונה.
  - נגדיר עבור הקפטן החדש (השורש החדש) שהקבוצה שלו היא הקונה (כי לפני כן היה שייך לנקנית), ונגדיר גם עבור הקונה שהוא הקפטן החדש שלה.
  - נעדכן את מספר השחקנים בקבוצה הקונה להיות סכום השחקנים של 2 הקבוצות.
  - נעדכן את סכום הניקוד של הקבוצה החדשה להיות סכום הניקוד של 2 הקבוצות המקוריות.
- נמחק את הnodes של הקבוצה שנקנתה מ2 עצי הדרגות של הקבוצות ואת האובייקט עצמו של הקבוצה מהמערכת.

<u>נכונות הסיבוכיות:</u>

חיפוש הקבוצות בעץ הקבוצות הכולל הוא בסיבוכיות O(logk). השימוש בפונקציית union_teams כפי שלמדנו, תהיה בסיבוכיות משוערכת של O(log*n), בשערוך שמתבצע יחד עם שאר הפונקציות הרלוונטיות. נציין כי עדכוני השדות בתוך פונקציה זו מתבצעים ב(O(1. כך גם ביצוע הפעולות ממחלקת פרמוטציה.עדכון שדה המשחקים בשורש הישן היא בסיבוכיות (O(1 כי היא כוללת גישה לשורש החדש וגישה לשדה בו, שתיהן ב(O(1. עדכון סכום הקבוצה הוא גם כן בסיבוכיות (O(1.
סה"כ סיבוכיות הפונקציה תהיה **O(logk+log*n) משוערך**.

<u>סיבוכיות מקום :</u>

כפי שהסברנו לעיל על מבנה הנתונים הכללי בו בחרנו להשתמש, המבנה יכיל מספר סופי של עצי דרגות AVL כך שכל עץ יכול להכיל מקסימום k צמתים. בנוסף, הunion find יוכל להכיל מקסימום n תאים במערך כשבכל תא יש שחקן אחד בלבד, או פחות תאים במערך כאשר בחלק מהתאים יש יותר משחקן אחד (למרות שאנו מטפלים בזה מבחינת פקטור העומס). כך או כך, הUF יכיל מספר תאים כמספר השחקנים, ולכן המקום שנשמר הוא כמספר השחקנים, כלומר מקסימום n תאים.
מכאן נסיק כי כל פעולה אותה נבצע, ובאופן כללי כל המידע שישמר בזכרון בתרגיל שמימשנו, לא יחרגו מסיבוכיות המקום הנתונה בתרגיל, הלוא היא O(n+k).

(מתנצלות מראש אם מישהו מכם אוהד צרפת 😊)