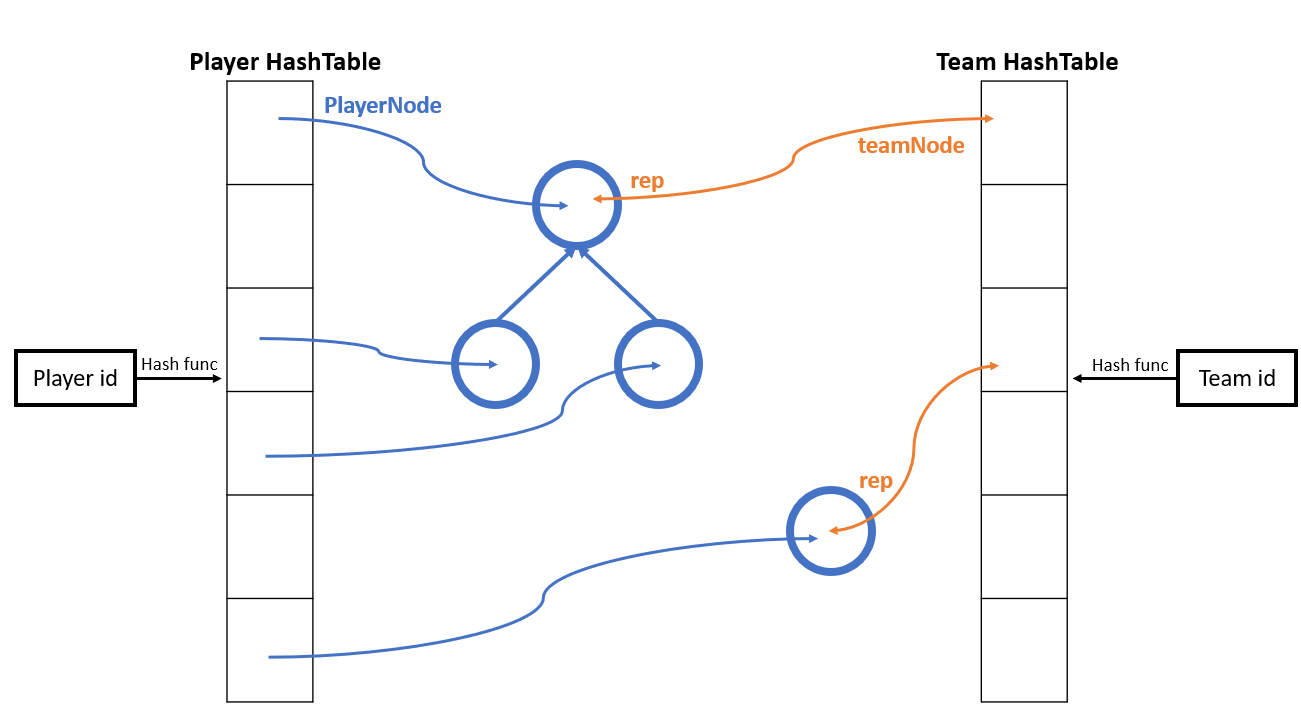
**תיאור המידע במבנה הנתונים:**

* **m\_all\_teams\_id** - עץ AVL ממוין לפי מזהה הקבוצות. בעץ זה נמצאות כל הקבוצות במערכת.
* **m\_all\_teams\_ability** – עץ AVL מסוג Rank Tree ממוין לפי יכולת הקבוצות ללא נקודות (pointless ability). במקרה של שוויון ביכולת – לפי מספר המזהה של הקבוצות.
* **m\_num\_teams** – (int) מספר הקבוצות בכל המערכת.
* **m\_game** – מבנה Union Find המכיל 2 מערכי HashTable:
  + **Player HashTable –**  כל איבר **playerNode** במערך מכיל:
    - key – מזהה השחקן.
    - data - מצביע לשחקן.
    - parent - מצביע לאב שלו בעץ ההפוך המכיל את כל שחקני הקבוצה
    - team - מצביע לאיבר במערך הקבוצות המייצג את הקבוצה בה השחקן משחק (רק השחקן המייצג את הקבוצה מצביע לאיבר זה, אחרת nullptr).
    - rg – (relative games) ערך המאפשר לחשב את כל המשחקים שהשחקן שיחק.
    - rs – (relative spirit) – ערך המאפשר לחשב את ה-partial spirit עד השחקן הנוכחי.
  + **Team HashTable –**  כל איבר **teamNode** במערך מכיל:
    - key – מזהה הקבוצה.
    - data - מצביע לקבוצה.
    - rep - מצביע לשחקן המייצג את הקבוצה – השורש של עץ השחקנים ההפוך.
    - rank – מספר השחקנים בקבוצה.
    - team\_spirit – פרמוטציה של הרכבת הרוח של כל שחקנים הקבוצה.
    - is\_Deleted – מייצג אם קבוצה נמחקה המערכת. true אם כן, אחרת false.



**כל איבר מסוג team מכיל:**

* **m\_has\_goalkeeper** – true אם יש שוער בקבוצה. אחרת false.
* **m\_points** – מספר הנקודות הכולל של הקבוצה.
* **m\_team\_id** – מזהה הקבוצה.
* **m\_team\_spirit –** פרמוטציית רוח הקבוצה**.**
* **m\_total\_ability –** היכולת הכוללת של כל שחקני הקבוצה.

**כל איבר מסוג player מכיל:**

* **m\_player\_Id** - מזהה השחקן.
* **m\_gamesPlayed** - מספר המשחקים שהשחקן שיחק
* **m\_cards** - מספר הכרטיסים שהשחקן קיבל
* **m\_isGoalKeeper** - אם השחקן שוער true – אחרת false
* **m\_ability –** היכולת של השחקן.
* **m\_spirit –** פרומטציית הרוח של השחקן.
* **m\_team\_id –** מזהה הקבוצה אליה משתייך השחקן.

**מימוש הפונקציות:**

* **world\_cup\_t()**

נאתחל את שלושת העצים  **m\_all\_eligible\_teams**, **m\_all\_teams\_id, m\_all\_teams\_ability**כעצים ריקים. אתחול עץ ריק O(1).

בנוסף נאתחל את מבנה ה-UnionFind **m\_game** ב- O(1) כפי שנלמד בכיתה.

סיבוכיות זמן: O(1).

* **virtual ~world\_cup\_t()**

שחרור הזיכרון של שלושת העצים וה-UnionFind. מעבר על n+k איברים (שחקנים וקבוצות) עד כדי כפל בקבוע. O(n+k).

סיבוכיות זמן: O(n+k).

* **StatusType add\_team(int teamId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT. נחפש ב-m\_all\_teams\_idאיבר בעל teamId. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם כבר קיימת קבוצה בעלת המזהה הנ"ל במערכת, נחזיר FAILURE, אחרת:

* נכניס ל-m\_all\_teams\_id איבר חדש מסוג team בעל key=teamId. הכנסה לעץ O(log(k)) כאשר k הוא מספר האיברים.
* נכניס ל-m\_all\_teams\_ability איבר חדש מסוג team בעל key=0, כיוון שבשלב זה אין לקבוצה pointless ability. הכנסה לעץ O(log(k)) כאשר k הוא מספר האיברים.
* נוסיף ל-m\_game קבוצה חדשה למבנה בסיבוכיות O(1) **(\*** נכונות בהמשך**)**.
* נוסיף 1+ ל-m\_num\_teams.
* נחזיר SUCCESS.

סיבוכיות זמן: O(log(k)) כאשר k הוא מספר הקבוצות במערכת.

* **StatusType remove\_team(int teamId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT. נחפש ב-m\_all\_teams\_idאיבר בעל teamId. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם לא קיימת קבוצה בעלת המזהה הנ"ל במערכת, נחזיר FAILURE, אחרת:

* נמחק מ- m\_all\_teams\_idאת הקבוצה בעלת key=teamId. הוצאה מעץ O(log(k)) כאשר k הוא מספר האיברים.
* נחסיר 1- ל-m\_num\_teams.
* נבדוק האם לקבוצה יש שוער. אם כן, נחפש ב-m\_all\_eligible\_teams איבר בעל teamId. חיפוש בעץ עם k איברים O(log(k)). אם האיבר קיים, נמחק אותו מהעץ.
* נבדוק מהו ה-team ability של הקבוצה. נחפש ב-m\_all\_teams\_ability איבר בעלkey=team ability ו- data=teamId. נמחק את האיבר. חיפוש בעץ עם k איברים O(log(k)).
* נחפש את הקבוצה במבנה m\_game ונסמן אותה כמחוקה.
* נחזיר SUCCESS.

סיבוכיות זמן: O(log(k)) במקרה הגרוע. כאשר k הוא מספר הקבוצות במערכת.

* **StatusType add\_player(int playerId, int teamId, const permutation\_t &spirit, int gamesPlayed, int ability, int cards, bool goalKeeper)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT. נחפש ב-m\_all\_teams\_idאיבר בעל teamId. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם לא קיימת קבוצה בעלת המזהה הנ"ל במערכת נחזיר FAILURE. בנוסף נחפש במבנה m\_game את השחקן בעל playerId. אם קיים, נחזיר FAILURE אחרת:

* ניצור שחקן חדש עם המזהה playerId בעל הפרמטרים שהועברו לפונקציה – O(1).
* נעדכן את הערך pointless ability של הקבוצה עם המזהה teamId – ע"י הוספת ה-ability של השחקן החדש לשדה m\_total\_ability של הקבוצה. נמחק את הקבוצה מהעץ הממוין לפי pointless ability ונכניס מחדש לאחר העדכון. מחיקה והכנה לעץ עם k איברים O(log(k)).
* נוסיף את השחקן למבנה m\_game. הכנסה למבנה UnionFind בסיבוכיות O(1) משוערך בממוצע על הקלט כפי שנלמד בכיתה **(\*** נכונות בהמשך**)**.
* נעדכן את הערך total\_spirit ו-rank של הקבוצה במבנה m\_game – O(1).
* אם הקבוצה אליה השחקן הצטרף ריקה – נשמור מצביע לשחקן בתור הנציג של הקבוצה בשדה ה-m\_rep של הקבוצה במבנה m\_game. ובנוסף, המצביע m\_parent של השחקן יהיה הוא עצמו – O(1).

סיבוכיות זמן: אנו משתמשים בפונקציות בסיבוכיות O(1) משוערך בממוצע על הקלט ו-O(log(k)) במקרה הגרוע. בסה"כ הסיבוכיות היא O(log(k))משוערך, בממוצע על הקלט, כאשר 𝑘 הוא מספר הקבוצות במערכת.

* **output\_t<int> play\_match(int teamId1, int teamId2)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT. נחפש ב-m\_all\_teams\_idאיבר בעל teamId1. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם לא קיימת קבוצה בעלת המזהה הנ"ל במערכת, או שקיימת אבל ערך m\_has\_goalkeeper שלה הוא false, כלומר אינה רשאית לשחק, נחזיר FAILURE.נבצע חיפוש זהה עבור teamId2. אחרת:

* + נחשב עבור כל קבוצת את היכולת שלה לנצח ע"י:
  + נשים לב שערכים אלו מעודכנים כי בכל הכנסת שחקן מעדכנים את m\_total\_ability ובפונקציה זו נעדכן את הערך points שמתווספות לאחר משחק.
  + במידה ויש שיוויון, נחשב את הכוח הרוחני של כל קבוצה ע"י הפעלת strength על פרמוטציית הרוח הכוללת של הקבוצה m\_team\_spirit, המתעדכנת בכל הוספת שחקן.
  + נוסיף לקבוצה המנצחת 3+ נקודות לערך m\_points. אם היה תיקו נוסיף לכל קבוצה 1+ לערך m\_points.
  + נוסיף לנציג של שתי הקבוצות במבנה ה-m\_game 1+ לערך m\_rg.

סיבוכיות זמן: O(log(k))במקרה הגרוע. כאשר k הוא מספר הקבוצות במערכת.

* **output\_t<int> num\_played\_games\_for\_player(int playerId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נבצע חיפוש לשחקן עם המזהה playerId במבנה ב-HashTable של השחקנים – סיבוכיות O(1) משוערך (**(\*\*)** נכונות בהמשך). אם קיים – נמשיך, אחרת נחזיר FAILURE.

נחזור למצביע לשחקן בעל המזהה playerId ונעלה חזרה לנציג הקבוצה. כאשר בכל מעבר בצומת בעץ, נסכום את ערכי m\_rg שלהם – זהו מספר המשחקים של השחקן playerId. (**(\*)** נכונות בהמשך). בעצם ביצענו פעולת find במבנה ה-UnionFind. פעולות find ו-union משוערכות ביחד בסיבוכיות O(log\*(n)) משוערך בממוצע על הקלט **(\*\*\*)**.

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

* **StatusType add\_player\_cards(int playerId, int cards)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נבצע חיפוש לשחקן עם המזהה playerId במבנה ב-HashTable של השחקנים – סיבוכיות O(1) משוערך (**(\*\*)** נכונות בהמשך).

אם קיים, נמשיך. אחרת נחזיר FAILURE.

נעלה במעלה העץ ההפוך של שחקני הקבוצה אליה משתייך השחקן עד שנגיע לשורש – נציג הקבוצה. פעולות find ו-union משוערכות ביחד בסיבוכיות O(log\*(n)) משוערך בממוצע על הקלט **(\*\*\*)**.

ניגש למצביע לקבוצה של הנציג ונבדוק האם הערך m\_deleted false. במידה וכן – נמשיך, אחרת נחזיר FAILURE.

נשמור את המצביע ל-Player השמור באיבר המייצג את השחקן בעל המזהה playerId ב-HashTable של השחקנים.

נוסיף לערך m\_cards +cards.

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

* **output\_t<int> get\_player\_cards(int playerId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נבצע חיפוש לשחקן עם המזהה playerId במבנה ב-HashTable של השחקנים – סיבוכיות O(1) משוערך (**(\*\*)** נכונות בהמשך). אם קיים – נמשיך, אחרת נחזיר FAILURE.

נשמור את המצביע ל-Player השמור באיבר המייצג את השחקן בעל המזהה playerId ב-HashTable של השחקנים.

נחזיר את הערך m\_cards.

סיבוכיות זמן: – סיבוכיות O(1) משוערך בממוצע על הקלט.

* **output\_t<int> get\_team\_points(int teamId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נחפש ב-m\_all\_teams\_idאיבר בעל teamId. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם לא קיימת קבוצה בעלת המזהה הנ"ל במערכת נחזיר FAILURE.

נחזיק את הערך m\_points של הקבוצה. נשים לב שערך זה מתעדכן בפונקציה play\_match בה יכולות להתווסף נקודות לקבוצה.

סיבוכיות זמן:O(log(k))במקרה הגרוע. כאשר k הוא מספר הקבוצות במערכת.

* **output\_t<int>** **get\_ith\_pointless\_ability(int i)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נחפש בעץ הדרגות את האיבר העל אינדקס i+1 שכן בעץ זה אנחנו מתחילים את הספירה מאינדקס 1 ובפונקציה זו מבקשים להתחיל את הספירה מאינדקס 0.

נשים לב שבכל הכנסת שחקן חדש ערך ה-pointless ability של הקבוצה מתעדכן, ולכן בפונקציה add\_player אנחנו מוציאים מעץ הדרגות את הקבוצה לה התווסף שחקן ומכניסים אותה מחדש עם הערך המעודכן של pointless ability.

חיפוש איבר בעל אינדקס מסוים בעץ דרגות – ביצוע פעולת select(k) שנלמדה בתרגול בסיבוכיות O(log(k)) כאשר k הוא מספר האיברים.

סיבוכיות זמן:O(log(k))במקרה הגרוע. כאשר k הוא מספר הקבוצות במערכת.

* **output\_t<permutation\_t> get\_partial\_spirit(int playerId)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נבצע חיפוש לשחקן עם המזהה playerId במבנה ב-HashTable של השחקנים – סיבוכיות O(1) משוערך (**(\*\*)** נכונות בהמשך).

אם קיים, נמשיך. אחרת נחזיר FAILURE.

נעלה במעלה העץ ההפוך של שחקני הקבוצה אליה משתייך השחקן עד שנגיע לשורש – נציג הקבוצה. פעולות find ו-union משוערכות ביחד בסיבוכיות O(log\*(n)) משוערך בממוצע על הקלט **(\*\*\*)**.

ניגש למצביע לקבוצה של הנציג ונבדוק האם הערך m\_deleted false. במידה וכן – נמשיך, אחרת נחזיר FAILURE.

נחזור למצביע לשחקן בעל המזהה playerId ונעלה חזרה לנציג הקבוצה. כאשר בכל מעבר בצומת בעץ, נסכום את ערכי m\_rs שלהם – הרכבה משמאל של כל הערכים האלו ייתן את הרוח של הקבוצה עד השחקן בעל המזהה playerId. (**(\*)** נכונות בהמשך). בעצם ביצענו פעולת find במבנה ה-UnionFind. פעולות find ו-union משוערכות ביחד בסיבוכיות O(log\*(n)) משוערך בממוצע על הקלט **(\*\*\*)**.

סיבוכיות זמן:O(log\*(n)) משוערך בממוצע על הקלט.

* **output\_t<int>** **buy\_team(int teamId1, int teamId2)**

נבדוק שהinput- תקין, אם לא נחזיר INVALID\_INPUT.

נחפש ב-m\_all\_teams\_idאיבר בעל teamId1. חיפוש בעץ עם k איברים O(log(k)) כאשר k הוא מספר האיברים. אם לא קיימת קבוצה בעלת המזהה הנ"ל במערכת נחזיר FAILURE.נבצע חיפוש זהה עבור teamId2.

נבצע פעולת union על שתי הקבוצות כאשר נאחד לפי גודל – כלומר נציג הקבוצה בעלת הגודל הקטן יותר יצביע לנציג הקבוצה הגדולה יותר בתור הm\_parent החדש שלו. פעולות find ו-union משוערכות ביחד בסיבוכיות O(log\*(n)) משוערך בממוצע על הקלט **(\*\*\*)**.

נסמן:

נציג קבוצה team1 – ownerParent

נציג קבוצה team2 - addedParent

נחלק למקרים:

1. אם rank team1 >= rank team2:

* נעדכן את הערך m\_rank של הקבוצה החדשה להיות גודל שתי הקבוצות ביחד.
* נעדכן את הspirit של הקבוצה החדשה להיות הרכבת ה-spirit של שתי הקבוצות:
* נעדכן את ה-spirit של addedParent על מנת לשמור על נכונות **(\*\*)**
* נעדכן את m\_rg של addedParent על מנת לשמור על נכונות **(\*\*)**
* נעדכן את ה-m\_parent של addedParent להיות ownerParent.

1. אחרת, rank team1 < rank team2:

* הנציג של team1 יהיה addedParent.
* הקבוצה של ownerParent תהיה team2.
* נעדכן את הערך m\_rank של הקבוצה החדשה להיות גודל שתי הקבוצות ביחד.
* נעדכן את הspirit של הקבוצה החדשה להיות הרכבת ה-spirit של שתי הקבוצות:
* נעדכן את ה-spirit של addedParent ו-ownerParent על מנת לשמור על נכונות **(\*\*)**
* נעדכן את m\_rg של addedParent ו-ownerParent על מנת לשמור על נכונות **(\*\*)**
* נעדכן את ה-m\_parent של addedParent להיות ownerParent.

נקרא לפונקציה remove\_team(teamId2) - O(log(k)) במקרה הגרוע. כאשר k הוא מספר הקבוצות במערכת.

סיבוכיות זמן: O(log(k)+log\*(n)) משוערך בממוצע על הקלט, כאשר k מספר הקבוצות ו- n מספר השחקנים במערכת (כולל שחקני עבר).

**(\*)הצגת נכונות חישוב הערכים partial spirit ו-num games עבור שחקן במערכת:**

נראה שאנו מתחזקים את הערכים (relative spirit) m\_rs ו-m\_rg (relative games) כראוי בכל שלב:

* **הוספת שחקן למערכת:**

נסמן – player - השחקן החדש, rep - נציג הקבוצה אליה השחקן מצטרף.

כאשר השחקן מתווסף לקבוצה ריקה –

כאשר השחקן מתווסף לקבוצה לא ריקה –

* **איחוד קבוצות:**

נסמן ownerParent – נציג הקבוצה הקונה, ו-addedParent - נציג הקבוצה הנקנית.

נחלק למקרים:

1. אם rank owner >= rank added
2. אם rank owner < rank added

* **כיווץ מסלולים:**

אנו מבצעים כיווץ מסלולים בביצוע פעולת find על מנת לשמור על סיבוכיות מבנה UnionFind בפעולות union ו-find.

תיקון הערך m\_rg:

במעבר הראשון ב-find לחיפוש השורש נחשב  *זהו סכום ערכי ה-m\_rg בכל שחקן עד השורש לא כולל!*

*במעבר השני בו גם נשנה את הparent של כל צומת בעץ, נחזיק:*

*זהו סכום ה-m\_rg של הצמתים עליהם עברנו במעבר השני עד עכשיו.*

*נעדכן את הm\_rg של הצומת לה שינינו את parent להיות:*

תיקון הערך m\_rs:

במעבר הראשון ב-find לחיפוש השורש נחשב  *זהו הרכבת ערכי ה-m\_rs בכל שחקן עד השורש לא כולל!*

*במעבר השני בו גם נשנה את הparent של כל צומת בעץ, נחזיק:*

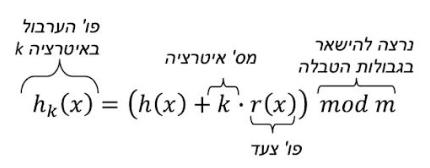
*זהו הרכבת ה-m\_rs של הצמתים עליהם עברנו במעבר השני עד עכשיו.*

*נעדכן את הm\_rs של הצומת לה שינינו את parent להיות:*

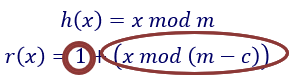
**(\*\*) הצגת נכונות חיפוש והכנסה ב-HashTable ב-:O(1)**

בHashTable שלנו אנו משתמשים ב- Double Hashing.

כפי שנלמד בתרגול, הדרישה מפונקציית הערבול והצעד היא:



כאשר:



כך ש-m ו-r(x) ללא מחלקים משותפים.

פונקציית הערבול שלנו אכן מקיימת:

כאשר:

כיוון שגודל טבלת הערבול שלנו הוא תמיד חזקות של 2, מתקיים ש:

כל התנאים מתקיימים ולכן לפי מה שנלמד בתרגול סיבוכיות חיפוש והכנסה ל-HashTable הינה O(1) בממוצע על הקלט.

**(\*\*\*) הצגת נכונות ביצוע פעולות Union, Find ב-O(log\*(n)) משוערך:**

בפעולת find אנחנו מבצעים כיווץ מסלולים. בפעולת union אנחנו מאחדים קבוצות לפי גודל. לפי משפט: אם נשתמש באיחוד לפי גודל ובכיווץ מסלולים, סיבוכיות הזמן של m פעולות Union, Find הינה O(m log\*(n)) (נראה בהרצאות).

לכן, סיבוכיות הזמן המשוערכת של שתי הפעולות יחד הינה O(log\*(n)).