תיאור מבני הנתונים:

השתמשנו ב 5 עצי AVL:

1- PlayersById, זהו עץ AVL שמכיל מידע על כל השחקנים שאנו מוסיפים למונדיאל, הממוין לפי המזהה של השחקנים כל צומת בעץ היא ממחלקת PlayerById שמכיל:
- PlayerId: המזהה המיוחד של השחקן.
- teamID: shared_ptr למזהה המיוחד של הקבוצה ששייך לה השחקן.
- gamesPlayed: מספר המשחקים שהשחקן שיחק.
- goals: מספר השערים שהבקיע השחקן.
- cards: מספר הכרטיסים שקיבל השחקן.
- goalkeeper: משתנה בוליני, true אם השחקן יכול לשחק שוער, false אחרת.
- gamesPlayedWithTeam: shared_ptr למספר המשחקים ששיחק השחקן עם הקבוצה ולא לבד.

2- PlayersByStats, זהו עץ AVL שמכיל מידע על כל השחקנים שאנו מוסיפים למונדיאל, הממוין לפי הסטטיסטיקות של השחקנים, כל צומת בעץ היא ממחלקת PlayerByStatsa שמכיל אותם שדות של המחלקה PlayerById שמפורטות למעלה, המחלקה מכילה גם אופרטור '>' שמשווה בין שני שחקנים לפי הסטטיסטיקות כמו שמפורט בעבודה.

3- TeamsInSystem, זהו עץ AVL המכיל מידע על כל הקבוצות שנמצאות במערכת, כל צומת בעץ היא ממחלקת Team שמכילה:
- teamId: המזהה הייחודי של הקבוצה.
- Points: מס' הנקודות של הקבוצה עד עכשיו.
- goalKeepers[2]: מערך שבאינדקס 0 יש ערך 1 אם קיים שוער בקבוצה אחרת יש ערך 0, באינדקס 1 מאחסנים את מספר השוערים בקבוצה.
- totalGoals: מספר השערים של הקבוצה עד עכשיו.
- totalCards: מספר הכרטיסים של הקבוצה עד עכשיו.
- totalPlayers: מספר השחקנים שיש בקבוצה עד עכשיו.
- topScorer[2]: מערך שבאינדקס 0 אחסננו את המזהה הייחודי של השחקן שהפקיע הכי הרבה שערים, באינדקס 1 נאחסן מספר השערים שהפקיע אותם.
- totalGamesPlayed: shared_ptr למספר המשחקים ששיחקה הקבוצה עד כה.
- teamTreeByStats: זהו עץ AVL של השחקנים שנמצאים בקבוצה זו הממוין לפי המזהה המיוחד של כל שחקן, כל צומת בעץ היא מסוג PlayerById.
- teamTreeById: זהו עץ AVL של השחקנים שנמצאים בקבוצה זו הממוין לפי הסטטיסטיקות של כל שחקן, כל צומת בעץ היא מסוג PlayerByStats.

4- ActiveTeams, זהו עץ AVL של הקבוצות הקשירות מתוך כל הקבוצות. כלומר כל צומת בעץ זה היא ממחלקת team ויש בה לפחות 11 שחקן ולפחות שוער אחד.

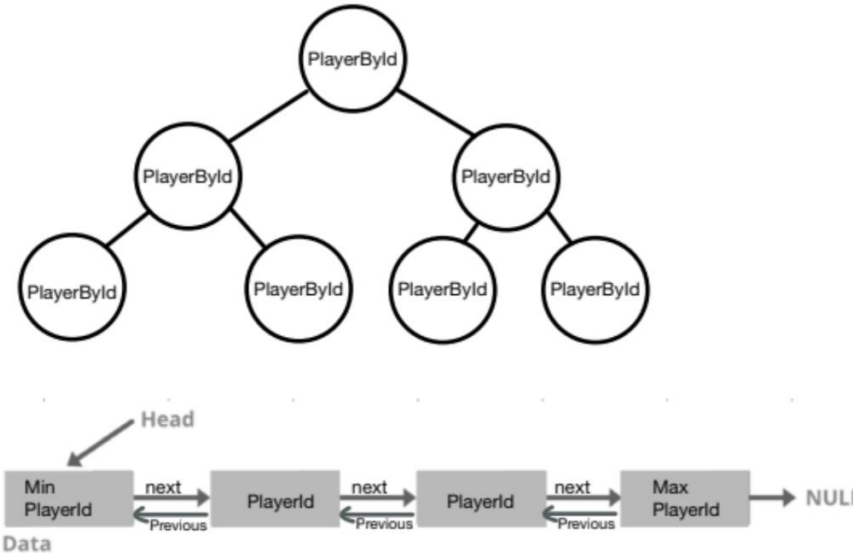
5- NonEmptyTeams, זהו עץ AVL של הקבוצות הלא ריקות מתוך כל הקבוצות, כלומר כל צומת בעץ זה היא ממחלקת team ויש בה לפחות שחקן אחד.

חוץ מעצי ה AVL, עשינו class של רשימה מקושרת, במחלקת עץ ה AVL, שמרנו רשימה מקושרת שהיא תשמש אותנו בכך שכל הצמתים בה ממוינות אחרי כל הוצאה והכנסה, הרשימה ממויינית לפי מיון העץ (לפי סטטיסטיקות או מזהה ייחודי).

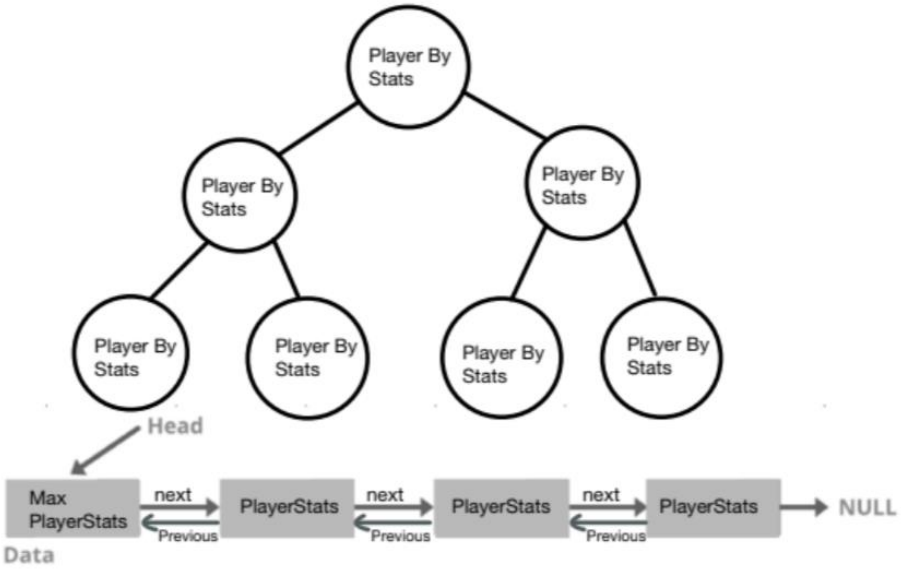
בנוסף שמרנו שני שדות, totalTeams שזה מספר הקבוצות הכולל, topScorer[2], מערך בגודל 2 כך שבאינדקס הראשון שלו נמצא המזהה של השחקן שכבש הכי הרבה שערים, באינדקס השני יש כמה שערים כבש שחקן זה.

שרטוט המערכת:

עץ ה PlayersById ממויין לפי המזהה של השחקן, כך גם הרשימה המקושרת של העץ :

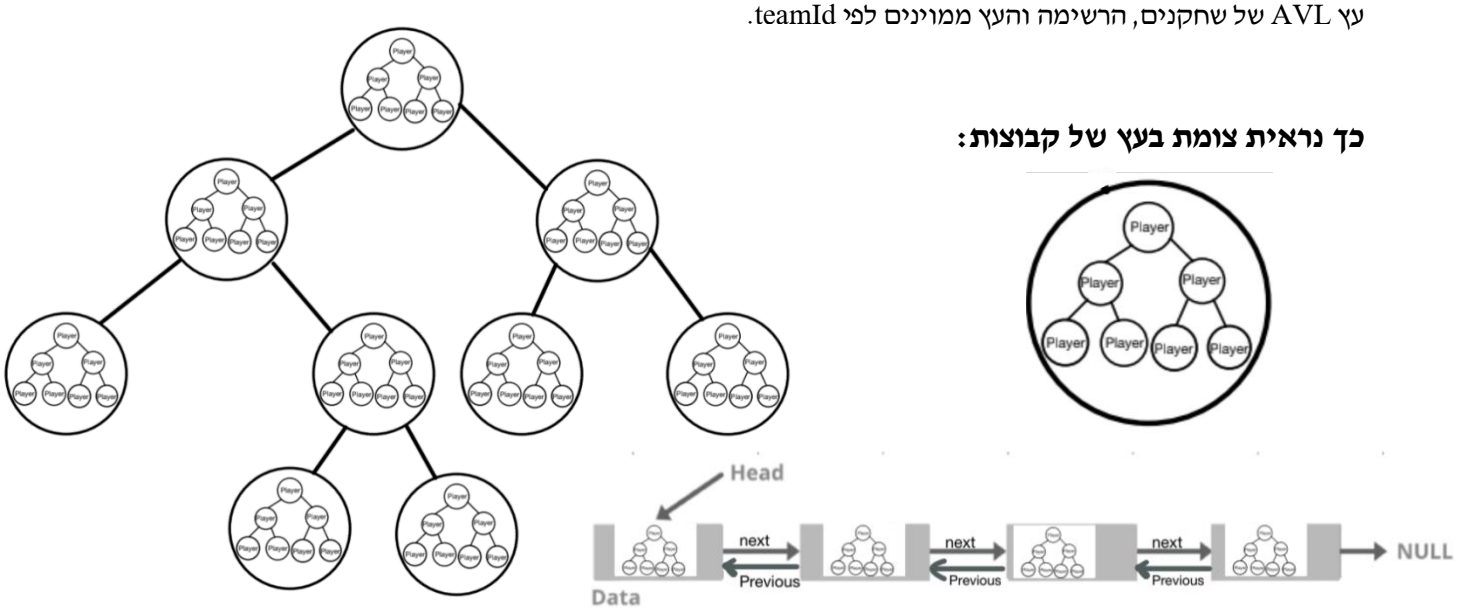


עץ ה PlayersByStats ממוין לפי הסטטיסטקות של כל שחקן, כך גם הרשימה המקושרת :



כל אחד מעצי הקבוצות, כל צומת בו היא קבוצה כלומר היא גם עץ של שחקנים, כל Node ברשימה המקושרת הוא עץ AVL של שחקנים, הרשימה והעץ ממוינים לפי teamId.

כך נראית צומת בעץ של קבוצות:



סיבוכיות המקום:

- שמרנו שני עצים לשחקנים שהם PlayersByID ו PlayersByStats, בכל אחד מהם קיימות n צמתות כך שכל אחת מהן היא שחקן (יש n שחקנים), כל צומת לוקחת o(1) מקום לכן סה"כ o(1)+o(1)+…+o(1) = o(n) ושני העצים ביחד o(n)+o(n)=o(2n)=o(n).
- עבור העץ teamsInSystem, כל צומת בו היא קבוצה אך לא שמרנו בה השחקנים שבקבוצה לכן כל צומת לוקחת o(1) מקום סך כל ומכיוון שיש לנו k קבוצות, סיבוכיות המקום היא o(1)+o(1)+…+o(1)= o(k).
- עבור העץ activeTeams, שיש בו הקבוצות הקשירות, המקרה הכי גרוע מבחינת מקום הוא שכל הגבוצות קשירות, יש k קבוצות וכל אחת לוקחת o(1) מקום לכן סך כל סיבוכיות המקום עבור עץ זה היא o(1)+o(1)+…+o(1)=o(k).
- עבור העץ nonEmptyTeams, המקרה הכי גרוע הוא שכל קבוצה יהיה בה שחקן אחד בלבד, אז יהיה הכי הרבה קבוצות לא ריקות ומספרם יהיה כמספר השחקנים n, לכן סיבוכיות המקום בעץ זה היא o(n).
- עבור הרשימה המקושרת, סך כל שמרנו רק הקבוצות והשחקנים, כלומר אין רשימות מקושרות חוץ מרשימות של קבוצות ושחקנים לכן במקרה הכי גרוע שמרנו O(n+k).
- קיבלנו שסך כל, סיבוכיות המקום של המערכת היא:

O(n)+o(n)+o(k)+o(k)+o(n)+O(n+k) =o(4n+3k)=o(n+k) כך ש n מספר השחקנים הכולל במערכת ו k מספר הקבוצות הכולל במערכת, כנדרש

תיאור הפונקציות והוכחת סיבוכיות הזמן:

נציין שלאורך כל ההסבר k הוא מספר הקבוצות הכולל במערכת ו n הוא מספר השחקנים הכולל במערכת, כפי שהוסבר בניסוח העבודה.

הכנסת והוצאת Node לעץ ה AVL הגנרי:

בהכנסה דאגנו להכניסו גם לרשימה המקושרת של העץ במיון הנכון, כפי שלומד בהרצאה רצנו מצומת לצומת בעץ וכשהגענו ל nullptr הוספנו את הצומת החדשה, שמרנו את הצומת שלפני ה nullptr, הצומת החדש יהיה או ה next שלה ברשימה או ה next->next או ה previous או ה previous->previous, (ה Node של העץ הוא אותו Node של הרשימה המקושרת (מצביעים)), כך הוספנו צומת לרשימה המקושרת הממוינת בסיבוכיות לוגריתמית.

עבור הוצאת צומת current מעץ ה AVL, סך כל קישרנו בין current->previous ו curr->next

ועשינו delete ל current, המיון לא נפגם. (מציאת ה current מתבצעת בעץ ה AVL לכן גם כאן הסיבוכיות לוגריתמית.

חיפוש בעץ הקבוצות הלא ריקות:

לחיפוש קבוצה בעץ זה יש יותר מאפשרות לסיבוכות הזמן, אם מספר השחקנים היה קטן ממספר הקבוצות, אז המקרה הכי גרוע למספר הקבוצות הלא ריקות הוא שכל שחקן נמצא בקבוצה לבד לכן סיבוכיות הזמן במקרה זה היא O(logn).

אם מספר השחקנים היה גדול ממספר הקבוצות, המקרה הכי גרוע הוא שכל הקבוצות לא ריקות לכן אז סיבוכיות הזמן הוא O(logk).

```
StatusType world_cup_t::add_team(int teamId, int points):
```

- תחילה בדקנו אם הקלטים תקינים זה לוקח o(1) זמן.
- אחר כך עשינו insert לעץ ה AVL של teamsInSystem שזה לוקח log(k) כפי שנלמד בהרצאה.
- בתוך פונקציית ה insert יש פונקציית find כדי לבדוק בכלל אם הקבוצה שייכת כבר למערכת, זה גם לוקח log(k).
- אחר כך עשינו find לקבוצה שהכנסנו בעץ כל הקבוצות ועדכננו את הנקודות שלה, זה גם לוקח log(k).

לכן סה"כ סיבוכיות הזמן: O(1)+O(logk)+O(logk)+O(logk) = O(1)+O(3logk)=O(logk) כנדרש.

הערה: כפי שנדרש בעבודה, אי אפשר להוסיף קבוצה לא ריקה, לכן לא צריך לדאוג להוספתה לעץ activeTeams ו nonEmptyTeams כי היא בוודאות לא מקיימת את דרישותיהם.

```
StatusType world_cup_t::remove_team(int teamId):
```

- תחילה בדקנו שגיאות וקלטים, זה לוקח o(1) זמן.
- אחר כך עשינו find לקבוצה בעץ הteamsInSystem כדי לבדוק אם היא בכלל קיימת, זה לוקח o(logk) זמן.
- אם מצאנו את הקבוצה, עשינו לה הסרה מהעץ כפי שנלמד בהרצאה בסיבוכיות של o(logk)

לכן סך כל סיבוכיות הזמן O(logk)+O(logk) = O(2logk)=O(logk), כנדרש.
הערה: כפי שדרשו בעבודה, אי אפשר למחוק קבוצה שיש בה שחקנים, לכן אנחנו לא צריכים לדאוג למחיקת הקבוצה מעץ ה activeTeams ועץ ה nonEmptyTeams כי היא בכלל לא שייכת להם מכיוון שאין בה שחקנים, מספיק למחוק אותה מ teamsInSystem.

```
StatusType world_cup_t::add_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper):
```

- תחילה בדקנו שגיאות ותקינות הקלטים, זה לוקח O(1) זמן.
- יצרנו shared_ptr ל teamId ויצרנו שני טיפוסים לשחקן זה אחד PlayerByStats והשני PlayerById, ה shared_ptr היה אחד הפרמטרים לבנאי השחקן, כל זה לוקח סיבוכיות של O(1).
- חיפשנו את הקבוצה בעץ של כל הקבוצות עם הפונקציה findTeam שמקבלת true אם רוצים לחפש בעץ כל הקבוצות ו false אם רוצים לחפש בעץ הקבוצות הלא ריקות, אם הקבוצה לר קיימת במערכת החזרנו Failure, כל זה לוקח O(logk) זמן.
- בדקנו אם הקבוצה של השחקן נמצאת ב nonEmptyTeams, זה לוקח o(logn) זמן.
- במידה והקבוצה לא נמצאת ב nonEmptyTeam כלומר הקבוצה עדיין ריקה, הכנסנו את השחקן לקבוצה, זה לוקח O(1) זמן כי הקבוצה ריקה, והכנסנו את הקבוצה ל nonEmptyTeams כי היא עכשיו לא ריקה, זה לוקח o(logn) זמן.
- במידה והקבוצה כן נמצאת ב nonEmptyTeams כלומר היא לא ריקה, הכנסנו את השחקן לקבוצה הרלוונטית בעץ ה nonEmptyTeams, זה לוקח o(logn) זמן.
- חיפשנו את השחקן בשני עצי השחקנים, PlayersById ו PlayersByStats אם הוא כבר נמצא, החזרנו Failure אם לא הכנסנו אותו, זה לוקח o(4logn) זמן.
- בדקנו אם הקבוצה אחרי הוספת השחקן היא קבוצה קשירה, o(1).
- במידה וכן, ניסינו למצוא את הקבוצה ב activeTeams, זה לוקח O(logn) זמן, אם כן הוספנו את השחקן אם לא הוספנו את הקבוצה כולה לעץ הקבוצות הקשירה, זה לוקח O(logn).
- בסוף עדכננו את ה topScorer, זה לוקח O(1) זמן.

לכן סך כל הסיבוכיות לפונקציה הזאת: O(1)+O(logk)+3O(logn)+O(4logn)+O(1)+O(logn) = O(logk+logn).

```
StatusType world_cup_t::remove_player(int playerId):
```

- תחילה נחפש את השחקן בעץ PlayersById, אם הוא לא נמצא נחזיר Failure, זה לוקח O(logn).
- נמצא את השחקן בעץ PlayersByStats, הוא בוודאות נמצא כי מצאנו אותו בעץ המזהים, זה לוקח O(logn).
- נמחק את השחקן משני עצים אלו, נבדוק את מספר ה nodes אם הוא קטן ב 1, זה אומר שהמחיקה התבצעה בהצלחה, אם לא נחזיר Failure, זה לוקח 2O(logn).
- נחפש את הקבוצה של השחקן בעץ הקבוצות הלא ריקות ונמחק ממנה את השחקן, זה לוקח O(logn).
- נבדוק אם הקבוצה נהיתה ריקה אחרי הסרת השחקן, אם כן נמחק אותה מעץ הקבוצות הלא ריקות, זה לוקח O(logn).
- נבדוק אם הקבוצה נמצאת בעץ הקבוצות הקשירות ואם הקשירות שלה נפגמה מהסרת השחקן אז נמחק אותה מעץ זה, זה לוקח 2O(logn).
- אם השחקן הזה הוא ה topScorer, נעדכן את ה topScorer על ידי חיפוש המקסימים בעץ PlayerByStats, זה לוקח O(logn).

לכן סך כל סיבוכיות הזמן לפונקציה זו : O(logn)=O(8logn), כנדרש.

```
StatusType world_cup_t::update_player_stats(int playerId, int gamesPlayed,
                                            int scoredGoals, int cardsReceived):
```

- נחפש את השחקן בעץ ה playersById, אם אין שחקן כזה נחזיר Failure,אם יש נשמור עותר למשתנה currentPlayer זה לוקח O(logn) זמן.
- ניצור PlayerByStats עם הנתונים של currentPlayer, נחפש אותו בעץ של PlyersByStats ונשמור אותו במשתנה PlayerToUpdate, זה לוקח o(logn) זמן.
- נמצא את הקבוצה של השחקן לפי המספר הייחודי שלה בעץ ה nonEmptyTeams, זה לוקח o(logn).
- נעדכן את הסטטסטיקות של השחקן בתוך הקבוצה שלו ב nonEmptyTeams, בעזרת הסרת השחקן, עדכון הסטטסטיקות שלו ואז הוספתו מחדש, נעדכן גם PlayerById וגם PlayerBystats, כל זה לוקח O(6logn).
- נחפש את קבוצתו בעץ ה activeTeams אם מצאנו נעדכן את סטטסטיקות שני טיפוסי השחקן בקבוצה, סה"כ O(6logn)+O(logn)
- נעדכן גם ב PlayersByStats וב PlayersById, זה לוקח 2O(6logn)

לכן סך כל סיבוכיות הזמן : O(clogn) = O(logn) , כנדרש.

```
StatusType world_cup_t::play_match(int teamId1, int teamId2):
```

- נבדוק תקינות קלט, O(1).
- נבדוק אם שתי הקבוצות נמצאות ב activeTeams אם אחת מהם לפחות לא נמצאת נחזיר Failure, זה לוקח O(2logn).
- נבדוק מי מנצח לפי הדרישות ונוסיף נקודות לקבוצה המנצחת, O(1).

לכן סך כל הסיבוכיות של פונקציה זו : O(logn).

```
output_t<int> world_cup_t::get_closest_player(int playerId, int teamId):
```

- בודקים את תקינות הקלט, בודקים גם אם יש רק שחקן אחד במערכת, מחזירים Failure , O(1).
- מחפשים את קבוצת השחקן בעץ כל הקבוצות, אם לא מצאנו מחזירים Failure, זה לוקח O(logk).
- אם מצאנו, מחפשים את הקבוצה בעץ הקבוצות הלא ריקות, זה לוקח O(logk) במקרה הגרוע, אחר כך מחפשים את השחקן בקבוצה שלו, זה לוקח ($O(\log n_{teamID})$.

- מחזרים את ה closest של השחקן שמצאנו, closest הוא שדה shared_ptr שכבר עודכן בכל הכנסה או הוצאה מהעץ.

לכן סך כל סיבוכיות הזמן של הפונקציה:
$O(1)+O(\log k)+O(\log k)+O(\log n_{teamID}) = O(\log k)+O(\log n_{teamID})$.

```
StatusType world_cup_t::get_all_players(int teamId, int* const output):
```

- אם teamId < 0, קוראים לפונקציה inOrder שממלאת מערך ממוין של שחקנים בסיבוכיות O(n) כמו שנלמד בהרצאה, רצים שוב על המערך וממלאים מערך ה output בעזרת getPlayerId(), זה לוקח O(n) לכן סך כל: $O(n)+O(n) = O(n)$, כנדרש.

- אם teamId > 0, מוצאים את הקבוצה בעץ הקבוצות הלא ריקות, זה לוקח במקרה הגרוע O(logk), אחר כך קוראים לפונקציית inOrderPlayers שקוראת ל inOrder ורצה שוב על המערך הממוין לפי סטטסטיקות וממלאת ה output במזהים הממוינים, זה לוקח $O(2n_{teamID})$.
  - לכן סך כל הסיבוכיות כש teamId > 0, $O(\log k)+O(n_{teamID})$.

```
output_t<int> world_cup_t::knockout_winner(int minTeamId, int maxTeamId):
```

- בודקים אם ה minTeamId לא נמצאת בעץ הקבוצות הלא ריקות, מוסיפים אותה ואז הקבוצה המינימלית הקיימת היא ה next של הקבוצה שהכנסנו, כי הרשימה המקושרת ממוינת אחר כך מוחקים את הקבוצה שהוספנו, אחרת הקבוצה המינימלית היא זאת שמצאנו עם minTeamId, אותו דבר עם הקבוצה המקסימלית וה previous, לכן זה לוקח $O(3\log\min\{n,k\})$, (מכיוון שעבדנו עם עץ הקבוצות הלא ריקות, אם מספר השחקנים קטן ממספר הקבוצות כפי שהוסבר בהתחלה, החיפוש יהיה O(logn) אחרת O(logk) .
- רצים על הרשימה המקושרת מהקבוצה הוולידית המינימלית ועד המקסימלית וממלאים מערך participatingTeams, זה לוקח O(r).
- קוראים לפונקציה knockout_aux, זאת היא פונקציה רקורסיבית שרצה בקפיצות של 2 על מערך הקבוצות המשתתפות, כל זוג מתחרה, מאחסנים את הקבוצה המנצחת באותו מערך במקום הקבוצה הראשונה בזוג, כלומר אחרי 3 משחקים, הקבוצות המצחות יהיו באנדיקסים 0,1,2, בסוף כל המשחקים של הסיבוב הראשון שולחם שוב את המערך לאותה פונקציה אבל עם size = size/2, לכן בכל האיטרציות זה לוקח: $O(r+r/2+\ldots+r/2^{\wedge}r) = O(r(1+1/2+1/4+1/8+\ldots)) = O(r*1) = O(r)$.

לכן סך כל הסיבוכיות של הפונקציה היא:
$O(\log(\min\{n,k\}))+O(r)+O(r) = O(\log(\min\{n,k\}))+O(r)$.

output_t<int> get_num_played_games(int playerId):

- נבדוק תחילה אם הקלט תקין ונחזיר INVALID_INPUT אם playerId <= 0.
- עתה נבדוק אם השחקן נמצא בכלל במערכת ע"י חיפוש בעץ playersById אשר מחזיק את כל השחקנים ממוינים לפי מזהה. במקרה והשחקן אינו נמצא במערכת נחזיר FAILED והפעולה תסתיים.
- אחרת, (כלומר השחקן קיים במערכת) נקרא לפונקציה getGamesPlayed שנמצאת במחלקה של PlayerById המחושבת לפי השדות –
  - 1) gamesPlayed- שדה המייצג את מספר המשחקים אשר מתעדכן בעת הוספת שחקן למערכת או ע"י update_game_stats.
  - 2) gamesPlayedWithTeam- אשר מהווה מצביע חכם (shared_ptr) לכמות המשחקים אשר השתתף בהם השחקן עם קבוצתו (למשל בפונקציה play_match).
- לסיום (במקרה של הצלחה) נחזיר אובייקט מסוג output שבו SUCCESS ומספר המשחקים בהם השתתף השחקן המבוקש.

- **סיבוכיות הזמן במימוש הפעולה:** O(log(n)) – מכיוון שאנו מחפשים את השחקן בעץ השחקנים הכולל אשר יכיל n שחקנים בסה"כ וכפי שלמדנו חיפוש בעץ AVL לוקח O(log(n)) זמן.

output_t<int> get_team_points(int teamId):

- נחזיר INVALID_INPUT אם teamId <= 0 .
- נבדוק אם הקבוצה נמצאת במערכת ע"י שימוש במתודה find אשר מחפשת node מסוים בעץ לפי מזהה הקבוצה, כאשר נבצע חיפוש בעץ nonEmptyTeams.
- אם הקבוצה אינה נמצאת במערכת (find החזירה nullptr) נחזיר אובייקט מטיפוס output_t עם StatusType::FAILURE.
- אחרת, כלומר הקבוצה אכן נמצאת במערכת, נקרא לפונקציה getTeamPoints() ע"י גישה לשדה data הנמצא בתוך הnode המוחזר מהמתודה find.
- לסיום נחזיר אובייקט מטיפוס output_t עם מספר הנקודות של הקבוצה המבוקשת עם SUCCESS.
- סיבוכיות הזמן במימוש הפעולה: O(log(k))- (חיפוש בעץ nonEmptyTeams אשר מכיל לכל היותר k קבוצות במקרה הכי גרוע ולכן כפי שלמדנו בכיתה חיפוש בעץ AVL מתבצע ב O(log(k)) כנדרש.

StatusType unite_teams(int teamId1, int teamId2, int newTeamId):

- קודם כל נבדוק תקינות קלט כמבוקש ונחזיר INVALID_INPUT עבור המקרים המצויינים בדרישות.
- נבדוק אם אין קבוצות עם המזהים teamId1/teamId2 או שקיימת קבוצה עם מזהה newTeamId אשר אינה teamId1 או teamId2. ואם באמת מתקיימים תנאים אלה נחזיר INVALID_INPUT.
- אחרת, נחפש את team1, team2 בעץ nonEmptyTeams ב O(log(k))לפי המזהים שלהם ונחלק למקרים:

- 1) אם הקבוצה team2 אינה מכילה שחקנים והקבוצה team1 מכילה שחקנים, אז נמחק את הקבוצה team2 מעץ ה teamsInSystem ב O(log(k)) (כמובן קבוצה זו אינה נמצאת בעץ nonEmptyTeams) ונעדכן את הteamId של team1 להיות ה newTeamId ב O(1) בנוסף נבדוק אם team1 נמצאת בעץ activeTeams ב O(log(k)) ואם כן אז נעדכן את השדה של teamId בצומת אשר מכילה את המידע עבור team1 גם כן ב O(1) .

- 2) אם הקבוצה team2 אינה מכילה שחקנים וגם הקבוצה team1 אינה מכילה שחקנים, אז נמחק את הקבוצה team2 מהעץ teamsInSystem ב O(log(k)) ונעדכן את הteamId של team1 להיות ה newTeamId ב O(1) ע"י קריאה לפונקציה setNewTeamId אשר נגישה משדה הdata של הnode team1.

- 3) אם הקבוצה team1 אינה מכילה שחקנים אבל הקבוצה team2 מכילה שחקנים אז באופן דומה למקרה "1" נמחק את הקבוצה team1 מעץ ה teamsInSystem ב O(log(k)), נעדכן את הteamId של team2 להיות ה newTeamId וגם במקרה והקבוצה team2 נמצאת בעץ ה activeTeams נחפש אותה בעץ זה ב O(log(k)) במקרה הכי גרוע, ונעדכן את השדה teamId להיות החדש ב O(1) .

- 4) אם שתי הקבוצות מכילות שחקנים. אז שתיהן בטח נמצאות בעץ nonEmptyTeams. נבצע פעולת mergeTrees אשר מתבצעת בסיבוכיות $O(n_{Team1Id} + n_{Team2Id})$ פעולת הmerge מתבצעת על עצי השחקנים

teamTreeById וגם teamTreeByStats אשר נמצאים כשדות במחלקה Team.
עתה לאחר המיזוג נעדכן את הteamId של הקבוצה team1 להיות ה newTeamId. (המיזוג בפועל מתבצע לתוך הקבוצה הראשונה). עתה נמחק את הקבוצה team2 מהעצים שהייתה בהם, כלומר נבדוק גם אם team2 הייתה בעץ activeTeams ואם כן נמחק אותה גם משם בנוסף לשאר עצי הקבוצות במערכת.
בנוסף נבדוק אם הקבוצה לאחר מיזוג הייתה קשירה כלומר הייתה נמצאת ב acitveTeams, אם כן נמחק אותה משם, ונוסיף אותה חזרה (בגרסתה המעודכנת לאחר מיזוג) כאשר פעולות אלו מתבצעות ב $2Log(k)=O(log(k))$. ואם הקבוצה לאחר המיזוג לא הייתה נמצאת ב activeTeams נוסיף אותה לשם ב $O(log(k))$ וזהו.

- לסיום נחזיר StatusType::SUCCESS .
- **הוכחת סיבוכיות זמן:** נקבל סה"כ במקרה הכי גרוע (שתי הקבוצות מכילות שחקנים):
  - $O(4log(k)) + O(n_{Team1Id} + n_{Team2Id}) = O(log(k)) + n_{Team1Id} + n_{Team2Id})$ כנדרש.

output_t <int> get_top_scorer(int teamId):

- אם teamID = 0 נחזיר INVALID_INPUT אחרת:
- אם teamId < 0:
  - שמרנו בתוך המחלקה world_cup_t שדה פרטי topScorer[2] אשר מהווה מערך באורך 2 ומכיל באינקס אפס את מספר המזהה של השחקן ובאינדקס אחד את מספר השערים העדכני שכבש, שדה זה מתעדכן אם צריך בכל הוספה או מחיקה של שחקן מהמערכת או עדכון סטטיסטיקות של השחקן בפעילות בהתאמה.
  - אם אין שחקנים כלל במערכת נחזיר FAILURE. אחרת נחזיר את הערך topScorer[0] אשר מהווה את מזהה השחקן בעל מספר השערים הגדול ביותר במערכת. ב $O(1)$.
- אם teamId > 0:
  - אז נבדוק אם הקבוצה בעלת המזהה teamId נמצאת במערכת, ב $O(log(k))$ אם לא נחזיר FAILURE, אם אכן נמצאת במערכת נחזיר את המזהה של השחקן בעל מספר השערים הגדול ביותר בקבוצה המבוקשת, כאשר בתוך המחלקה Team קיים שדה דומה לשדה שהסברנו לעיל ב $O(1)$.
- אם אכן קיים מלך שערים בשני המקרים נחזיר SUCCESS עם מספר המזהה של מלך השערים.
- **הוכחת סיבוכיות הזמן:**
  - אם teamId < 0: הפעולה מתבצעת ב $O(1)$ כנדרש.
  - אם teamId > 0: הפעולה מתבצעת ב $O(log(k)) = O(1) + O(log(k))$ כנדרש.
  - לכן במקרה הגרוע הפעולה מתבצעת ב $O(log(k))$.