**הסבר על מבני הנתונים:**

1. עץ הAVL:

מימוש עץ AVL כפי שנלמד בכיתה עם השינויים הבאים:

* שדה המידע של העץ הוא טמפלייטי, כדי לממש עצים מטיפוסים שונים (שחקנים/ קבוצות)
* שדה המפתח של העת הוא טמפלייטי (עבור עצים הממוינים לפי id הוא יהיה מסוג int, ועבור אלו הממוינים לפי שערים וכרטיסים המפתח יהיה מסוג player.
* לכל node בעץ מצביע לשני בניו וגם לאביו.
* לכל node בעץ מצביע למידע ומפתח שלפיו העץ ממוין

מידע חשוב על המחלקות team ,world\_cup\_t ו- player (לצורך הוכחת סיבוכיות בהמשך)

1. World\_cup\_t:

* מחלקה זו מחזיקה כמה עצים:

1. m\_allPlayersIds- עץ של כל השחקנים הממוין לפי הid שלהם.
2. m\_allPlayersGoals- עץ של כל השחקנים הממוין לפי גולים ואז כרטיסים ואז id של השחקנים כנדרש לדירוג השחקנים.
3. m\_teams- עץ של כל הקבוצות במשחק הממוין לפי הid שלהן.
4. m\_qualifiedTeams- עץ של כל הקבוצות הכשירות למשחק ממויין לפי הid שלהן.

* בנוסף המחלקה מחזיקה מצביע לtop scorer של כל השחקנים.

1. Team:

* לכל קבוצה מצביע לtop\_scorer שלה.
* לכל קבוצה שני עצים עבור השחקנים שלה (אחד ממוין לפי id והשני ממוין לפי goals, cards and ids (כפי שנדרש עבור קביעת הדירוג בין השחקנים)
* שדות עבור מספר המשחקים, הכרטיסים ומספר השוערים בקבוצה
* מצביעים לקבוצות משמאל ומימין לקבוצה בסדר in order של עץ הqualified teams(עם הקבוצה לא כשירה למשחק אז הם יהיו nullptr).

1. Player:

* לכל שחקן מצביע לקבוצה אליה הוא שייך.
* לכל שחקן מצביע לשחקן משמאלו ומימינו בסדר in order של העץ m\_allPlayersGoals.
* לכל שחקן שדות שמכילים את הסטטיסטיקות הנדרשות לגביו (שערים, כרטיסים וכו')
* כל שחקן מחזיק את כמות המשחקים ששיחק בלי מספר המשחקים של הקבוצה שלו. למשל, אם שחקן שיחק בקבוצה מסויימת 10 משחקים אז שדה המשחקים הפרטי שלו יהיה 0. כאשר השחקן יעבור קבוצה יעודכן השדה שלו ל-10 וכך יוכל לשמור על מספר המשחקים הכולל שלו ללא קשר לקבוצה בה הוא משחק. כאשר ישאל שחקן כמה משחקים שיחק נחבר את מספר המשחקים הפרטי עם זה של הקבוצה.

**Diagram

Description automatically generatedA picture containing text

Description automatically generated**

**Diagram

Description automatically generatedA picture containing text, necklet

Description automatically generated**

**הסבר על הפעולות:**

מעתה נסמן ב-n את מספר השחקנים ו-k את מספר הקבוצות.

1. **World\_cup\_t()**

יצירת העצים הריקים ואתחול השדות.

סיבוכיות זמן: O(1)- אתחול עצים ללא איברים זו פעולה קבועה.

סיבוכיות מקום: – אתחול עצים ריקים בסיבוכיות מקום של קבוע.

1. **~world\_cup\_t()**

עלינו למחוק את כל השחקנים, קבוצות וnodes שנוצרו. מחיקת העצים נעשית ע"י ההורס של מחלקת AVLTree המבצע את המחיקה בצורה רקורסיבית בעזרת חיפוש post order שהוא כפי שנלמד בהרצאה ב-O(number of nodes). בנוסף נמחק את כל השחקנים ע"י הוצאה שלהם למערך בעזרת סיור in order שלוקח O(n) ולאחר מכן מחיקת האיברים במערך על ידי מעבר עליו, כלומר שוב O(n). יצירת המערך היא בסיבוכיות מקום של O(n) כמספר השחקנים במערכת. נבצע פעולה זו עבור הקבוצות בסה"כ O(k). יצירת המערך היא בסיבוכיות מקום של O(k) כמספר הקבוצות במערכת. בנוסף הריסת כל העצים של הקבוצות לוקחת עוד O(n) מאחר ובעצי כל הקבוצות יחד מוחזקים כל השחקנים במשחק.

סיבוכיות זמן: על ידי סכימת כל הפעולות לעיל נקבל:

סיבוכיות מקום:

1. **Add\_team()**

הוספת הקבוצה לעץ הקבוצות ( לפי ההרצאה הוספה של איבר לעץ AVL לוקחת log(k) במקרה הגרוע כאשר k הוא מספר הקבוצות במערכת.)

סיבוכיות זמן: לפי ההרצאה

1. **Remove\_team():**

הוצאת הקבוצה מעץ הקבוצות לפי האלגוריתם מההרצאה. הקבוצה ריקה בעת ההוצאה לכן העצים שהיא מחזיקה ריקים והריסתם היא פעולה של קבוע.

סיבוכיות זמן: לפי ההרצאה

1. **Add\_player:**

הוספת השחקן לעצים הבאים:

* עצי כל השחקנים במערכת (אחד הממוין לפי id והשני לפי שערים וכרטיסים.)  
   (סיבוכיות של log(n) כאשר n הוא מספר השחקנים במערכת)
* עץ השחקנים של הקבוצה שלו (log(n1) כאשר n1 הוא מספר השחקנים בקבוצה שלו (בוודאות קטן מn)
* עץ השחקנים בקבוצה שלו הממוין לפי השערים והכרטיסים (כפי שהוסבר לעיל, סיבוכיות של log(n1)).
* עץ הקבוצות הכשירות לשחק במידה ובהוספת השחקן השתנתה כשירות קבוצתו (זאת בסיבוכיות של log(k) כך שk הוא מספר הקבוצות הכשירות לשחק, כך שk הוא מספר הקבוצות הכשירות לשחק, ובהכרח k<n מכיוון שקיימות לכל היותר n/11 קבוצות כשירות. כי בכל קבוצה כשירה ישנם 11 שחקנים.)

השוואת השחקן לtop\_scorer בקבוצתו ובמערכת והחלפתו בעת הצורך (סיבוכיות של o(1) מכיוון שהמצביעים לשחקן הטוב ביותר שמורים במערכת)

עדכון שכניו של השחקן (השמורים לכל אחד מהשחקנים במערכת כשדות של המחלקה player), זאת בעזרת פונקציה למציאת שכנים הממומשת בAVLTree.H), פונקציה זו רצה משורש העץ לכל היותר עד אחד מעליו, והיות שגובה העץ הוא log(n) כך שn הוא מספר הצמתים בעץ, סיבוכיות הפונקציה היא גם log(n).

סיבוכיות:

1. **Remove\_player:**

הוצאת השחקן משני העצים בworld\_cup\_t, לפי האלגוריתם מההרצאה בסיבוכיות של O(logn). בנוסף נוציא את השחקן מהעצים בקבוצתו גם כן בסיבוכיות של O(logn) (השחקן מחזיק מצביע לקבוצה שלו לכן אין צורך לחפש אותה).

אם הוא היה top\_scorer אזי נחליף את מצביעי הtop\_scorer הרלוונטים (בקבוצה ובמערכת) לשכן של האיבר משמאל (סיבוכיות של o(1) מכיוון שכל אלו שמורים כשדות של השחקן, הקבוצה והמערכת)

עדכון סטטוס הקבוצה באם הוצאת השחקן פגע בכשירות שלה לשחק (סיבוכיות של o(log(r)) במקרה הגרוע, כאשר r הוא מספר הקבוצות הכשירות, מספר זה קטן מ-n מאחר ומספר הקבוצות הכשירות במערכת הוא n/11).

סיבוכיות:

1. **Update\_player:**

שינוי פרטי השחקן עלול לשנות את דירוגו ביחס לשאר בשחקנים ולכן נוציאו תחילה מהעצים הרלוונטים, נעדכן את פרטיו ואז נוסיפו חזרה.  
-זאת בסיבוכיות של log(n) במקרה הגרוע כאשר n הוא מספר השחקנים במערכת, מסיבות שפורטו לעיל עבור הכנסה והוצאה של שחקן לעצים.

עדכון מצב הכשירות של הקבוצה אליה משתייך השחקן (הכנסה או הוצאה מעץ הקבוצות הכשירות לשחק) וזאת בסיבוכיות של log(r) כאשר r הוא מספר הקבוצות הכשירות לשחק, והראינו לעיל מדוע בהכרח log(r) < log(n), כאשר n הוא מספר השחקנים במערכת.

כמו כן, נעדכן את שכניו של השחקן, בסיבוכיות של log(n) כפי שהוצג בסעיפים קודמים, ואת המצביעים לtop\_scorer במידה וחל שינוי עקב עדכון פרטיו של השחקן. זאת בסיבוכיות של o(1) מכיוון שאלו שמורים כשדות של המחלקות השונות.

סיבוכיות:

1. **Play\_match:**

חיפוש הקבוצות בעץ הקבוצות בסיבוכיות של log(k) (סיבוכיות חיפוש בעץ avl)

לאחר מכן השוואת הscore של הקבוצות בעזרת פונקציית עזר שמבצעת חישוב מתאים על השדות של הקבוצות המתאימות. הניקוד וסך הניקוד והשערים של כל קבוצה שמורים כשדות שלה, ולכן זה נעשה בסיבוכיות של o(1).

קביעת תיקו או קביעת הקבוצה המנצחת ועדכון השדות שלה בהתאם גם נעשה בסיבוכיות o(1) מכיוון שזהו עדכון מספר סופי וידוע של שדות.

סיבוכיות:

1. **Get\_num\_played\_games:**

ראשית נחפש את השחקן בעץ כל השחקנים במערכת בסיבוכיות של log(n) (חיפוש בעץ avl). לאחר מכן נחשב את כמות המשחקים שששיחק בסיבוכיות של o(1) באופן הבא:

שמירת המידע על כמות המשחקים ששיחק כל שחקן נעשית באופן הבא:

כשאר מוסיפים שחקן לקבוצה, שומרים לאותו שחקן כשדה את כמות המשחקים ששיחק עד רגע ההוספה, וממספר זה מורידים את כמות המשחקים ששיחקה הקבוצה בלעדיו עד אותו רגע. החל מרגע זה, בכל משחק של הקבוצה מתעדכן רק השדה של המשחקים שהקבוצה שיחקה.

ואז, כאשר נגיע לחשב את כמות השחקנים של שחקן ספציפי, פשוט נחבר את שדה המשחקים של השחקן ושדה המשחקים של הקבוצה שנמצא בה. סכימת שני השדות האלה נעשית בסיבוכיות o(1).

סיבוכיות:

1. **Get\_team\_points:**

חיפוש הקבוצה בעץ הקבוצות בסיבוכיות של log(k). (מסיבות שתוארו לעיל)

לאחר מכן נחזיר את שדה הנקודות של הקבוצה בסיבוכיות של o(1).

סיבוכיות:

1. **Unite\_teams::**

חיפוש הקבוצות לאיחוד בעץ הקבוצות בסיבוכיות של log(k).

הקצאת שני מערכים (אחד לכל קבוצה), כל אחד בגודל כמות השחקנים בקבוצה. והקצאת מערך שלישי עבור הקבוצה החדשה שתכיל את איחוד הקבוצות ( בגודל סכום הגדלים של מערכי הקבוצות).  
 --סיבוכיות מקום של 2(n1+n2) כאשר n1 + n2 < n כאשר n הוא מספר השחקנים במערכת, ולכן סיבוכיות המקום ליניארית ל n + k.  
מיון שני עצי הקבוצות לאיחור אל תוך מערכי הקבוצות שלהם באמצעות in order בסיבוכיות של o(n1 + n2).   
ביצוע אלגוריתם merge אל תוך המערך של הקבוצה החדשה גם בסיבוכיות של o(n1 + n2)  
בניית העץ החדש בעזרת המערך הממוין, בריקורסיה על מציאת שורש העץ, שבכל שלב יהיה האיבר האמצעי במערך, ושליחת שני חצאי המערך לפעולה דומה עבור בניה (הימני והשמאלי). זאת בסיבוכיות של o(n1 + n2) מכיוון שעבור על איבר במערך מבצעים כמות סופית וידועה מראש של פעולות. (עדכון שדותיו).

הוצאת הקבוצות לאיחוד מהמערכת הוספת הקבוצה החדשה, ועדכון עץ הqualified teams במידת הצורך, בסיבוכיות של o(log(k)) כאשר k הוא מספר הקבוצות, מסיבות שתוארו לעיל עבור הכנסה והוצאה של קבוצות מהמערכת.

סיבוכיות זמן:

1. **Get\_top\_scorer:**

אם מקבלים teamId < 0 נמצא ונחזיר את הtop scorer בO(1) מאחר והוא מוחזק כשדה בworld\_cup\_t.

אחרת נחפש את הקבוצה בסיבוכיות O(logk) כמו שהוסבר לעיל ונחזיר את הtop scorer בO(1) מאחר והוא מוחזק כשדה בקבוצה.

סיבוכיות זמן:

1. **Get\_all\_players\_count:**

עבור teamId > 0 :

חיפוש הקבוצה בעץ הקבוצות בסיבוכיות של o(log(k)) כאשר k הוא מספר הקבוצות המערכת מסיבות שתוארו לעיל.  
החזרת השדה של גודל עץ השחקנים של אותה קבוצה בסיבוכיות של o(1).

סיבוכיות זמן: o(log(k))

עבור teamId < 0 :

החזרת השדה של גודל עץ השחקנים שבמערכת בסיבוכיות של o(1).

סיבוכיות זמן: o(1)

1. **Get\_all\_players:**

נסמן n- מספר השחקנים במערכת. K מספר הקבוצות המערכת.

עבור teamId < 0:  
ביצוע In order על עץ השחקנים של המערכת הממוין לפי שערים, כרטיסים וid's והשמתם במערך, סיבוכיות של o(n).

סיבוכיות זמן: o(n)

עבור teamId > 0:

חיפוש הקבוצה בעץ הקבוצות בסיבוכיות של o(log(k)).  
ביצוע In order על עץ השחקנים של הקבוצה הממוין לפי שערים, כרטיסים וid's והשמתם במערך, סיבוכיות של o(n1) כאשר n1 הוא מספר השחקנים בקבוצה.

סיבוכיות זמן: o(log(k) + n1)

1. **Get\_closest\_player:**

חיפוש הקבוצה בעץ הקבוצות בסיבוכיות של log(k) כאשר k הוא מספק הקבוצות המערכת.  
חיפוש השחקן בקבוצה בסיבוכיות של log(n1) כאשר n1 הוא מספר השחקנים בקבוצה.  
לכל שחקן שמורים שני השחקנים הקרובים לו בדירוג כשדות, ולכן רק נבדוק מבין שניהם מי יותר קרוב אליו בסיבוכיות של o(1).

סיבוכיות זמן: o(log(k) + log(n1))

1. **Knockout\_player:**

נזכיר כי לכל קבוצה בעץ qualified team שמורות הקבוצות הקרובות לך מבחינת id.

נסמן k1 – מספר הקבוצות בעץ הקבוצות הכשירות לשחק.  
נראה כי k1 < n && k1 < k כאשר n הוא מספר הקבוצות במערכת וk הוא מספר הקבוצות במערכת.  
בכל קבוצה כשירה ישנן 11 שחקנים ולכן בהכרח k1 < n. בנוסף הקבוצות הכשירות היא תת קבוצה של כמות הקבוצות המערכת ולכן בהכרח .

ולכן בהכרח: k1 < min{n, k}.

נבצע חיפוש של הקבוצה הקרוב ביותר מלמעלה לmin\_index שקיבלנו כפרמטר, זאת בסיבוכיות של log(k1). (כי החיפוש הוא חיפוש בינארי.  
נקצה מערך בגודל של כמות הקבוצות הכשירות, ונתחיל למלאו כך שבמקום הראשון נמצאת הקבוצה הכשירה הראשונה על אינדקס גדול/שווה לmin\_undex , ובכל פעם נכניס את הקבוצה מימינה (השמורה כשדה שלה) עד שייגמרו הקבוצות או שאינדקס הקבוצה יעבור את max\_index. זאת בסיבוכיות מקסימלית של p כאשר p = max\_index – min\_index.

כעת נרוץ על מערך הקבוצות הכשירות, בלולאה חיצונית שנגמרת כאשר נמצא זוכה. ולולאה פנימית שרצה על איברי המערך, ובכל ריצה צעד הקפיצה שלה גדל פי 2.

באופן זה סיבוכיות הזמן תהיה:

ובכל פעם נשווה בין כל שתי קבוצות בסיבוכיות של o(1) (כי הפרטים שלהם שמורים כשדות שלהם). *ונשמור את הקבוצה הזוכה באינדקס השמאלי מבין שתי הקבוצות ונעדכן את פרטיה בהתאם.*

*לבסוף נחזיר את הקבוצה הנמצאת באינדקס האפס במערך.*

*סיבוכיות זמן: O(log(k1) + p) = O(log{min{n, k} + p)*