**תרגיל רטוב 1 – החלק היבש**

שרה גריפית – ת"ז: 341312304

ליאור בר יוסף – ת"ז: 207022443

**תיאור מבנה הנתונים:**

עבור האפליקציה של פיפ"א, נבנה מבנה נתונים המורכב מארבע עצי AVL מרכזיים:

* **m\_teamsByID** – כל הקבוצות שמשתתפות בגביע העולם, מסודרים לפי מזהה הקבוצה.
* **m\_qualifiedTeams** – כל הקובצות המשתתפות בגביע העולם אשר כשרים לשחק (בעלי לפחות 11 שחקנים ולפחות שוער אחד).
* **m\_playersByID** – כל השחקנים שמשתתפים בגביע העולם, מסודרים לפי מזהה השחקן.
* **m\_playersByScore** – כל השחקנים שמשתתפים בגביע העולם, מסודרים לפי ה-score של השחקן המבוסס על כמות השערים שהבקיעו + הכרטיסים שקיבלו + מזהה השחקן (במקרה של שוויון).

בגביע העולם יש גם משתנה נוסף הסוכם את כמות השחקנים שמשתתפים בכל גביע העולם ומשתנה נוסף הסוכם את כמות הקבוצות, ומצביע לשחקן עם ה-score הכי גבוה.

נוסיף שתי מחלקות: Player ו-Team אשר מכילים את הפרטים של השחקנים ושל הקבוצות כדורגל המשתתפים בגביע העולם.

* **מחלקת Player** תכיל את כל הפרטים הבסיסיים על אותו השחקן: מזהה שחקן, מספר המשחקים שהשחקן שיחק (בנפרד למשחקים ששיחק עם קבוצתו הנוכחית), מספר השערים שהבקיע, מספר הכרטיסים שקיבל והאם יכול לשחק כשוער. בנוסף יכיל מצביע לקבוצה בה משחק, ושני מצביעים לשחקנים שהכי "קרובים" לשחקן מבחינת score (אחד עם score גבוה והשני עם נמוך מזה של השחקן).
* **מחלקת Team** תכיל את הפרטים הבסיסיים של אותה קבוצה: מזהה קבוצה, מספר הנקודות של הקבוצה, מספר השחקנים בקבוצה, מספר השוערים בקבוצה, סה"כ השערים ששחקני הקבוצה הבקיעו, סה"כ כרטיסים שהקבוצה קיבלה ומספר המשחקים שהקבוצה שיחקה. בנוסף הוא מחזיק שני מצביעים לקבוצות עם מזהים סמוכים (אחד המצביע לקבוצה עם המזהה הסמוך הקטן מהמזהה של שהקבוצה הנוכחי והשני לגבוה). המצביעים הללו יצביעו לקבוצה (ולא nullptr) רק במידה והקבוצה כשרה למשחק (בעלת 11 שערים לפחות ולפחות שוער אחד).
  + בנוסף לפרטים הבסיסיים, בקבוצה נחזיק שני עצי AVL – אחד של כל השחקנים באותה קבוצה מסודרים לפי מזהה שחקן, והשני של כל השחקנים באותה קבוצה מסודרים לפי ה-score של שחקני הקבוצה, המבוסס על כמות השערים שהבקיעו + הכרטיסים שקיבלו + מזהה השחקן (במקרה של שוויון).

**סיבוכיות מקום:** בכל פעולה, אין הוספה של סיבוכיות מקום מעבר לנדרש לכלל התרגיל, אשר הוא O(n+k), כאשר n הוא כמות השחקנים ו-k הוא כמות הקבוצות בגביע העולם.

**להלן רשימת הפעולות הנדרשות לממשק והסבר למימושם והסיבוכיות שלהם:**

**world\_cup\_t –**

נאתחל את ארבעת עצי הAVL, וגם אתחול מצביע נוסף לPlayer אשר מצביע על השחקן עם ה-score הכי גבוה. כמו כן, נאתחל את המשתנה שסופר את כל השחקנים המשתתפים בגביע העולם ל-0.

סיבוכיות זמן:

נאתחל את כל המשתנים ל-null או לערכים דיפולטים. עבור העצים – עץ ריק מכיל רק node אחד, וסיבוכיות הזמן ליצירתו הוא O(1). באופן דומה גם המשתנים ה-int-ים והמצביעים הנוספים. ולכן סיבוכיות הזמן הוא O(1).

**world\_cup\_t~ –**

ראשית נשחרר את ה-data של כל השחקנים והקבוצות בגביע העולם. נעשה זאת ע"י שחרור הזכרון של ה-data בעצים m\_teamsByID ו-m\_playersByID שמכילים את כל הקבוצות וכל השחקנים.

לאחר מכן, ה-destructor-ים הדיפולטים של העצים השונים יקראו, וישחררו את הזיכרון של ה-nodes של העצים. בנוסף, יקראו ה-destructor-ים הדיפולטים למצביע של השחקן עם ה-score הכי גבוה ובכן למשתנה int הסופר את כמות סך השחקנים.

סיבוכיות זמן:

כדי לשחרר את ה-data, נעבור על כל הקבוצות והשחקנים שקיימים בגביע העולם, ונשחרר את הזיכרון. זהו נעשה בסיבוכיות O(n+k) כאשר n הוא כמות השחקנים בגביע העולם, ו-k הוא כמות הקבוצות בגיע העולם. לאחר מכן, יקראו ה-destructor-ים הדיפולטים של העצים השונים, אשר עוברים על כל ה-nodes בגביע העולם:

* + יש לנו שני עצים של שחקנים, ולכן עבור n שחקנים, יש לכל עץ מקסימום n nodes. לכן סיבוכיות המקום של מעבר על העצים הללו הוא O(2n) = O(n).
  + יש לנו שני עצים של קבוצות, ולכן עבור k קבוצות, יש לכל עץ מקסימום k nodes. לכן סיבוכיות הזמן של מעבר על העצים הללו הוא O(k) O(2k) =.
    - בתוך כל קבוצה יש שני עצים של השחקנים המשחקים באותה קבוצה. כל השחקנים מפוזרים בין הקבוצות השונות ולכן סך כל השחקנים בעצים הללו בכל הקבוצות הוא 2n (n בכל סוג עץ) – זהו משום שכל שחקן משחק רק בקבוצה אחת.

לכן במעבר על כל הקבוצות ביחד, נעבר על עוד 2n nodes עבור כל עץ של קבוצות – סך הכל עוד 4n nodes.

לכן סיבוכיות הזמן סך הכל הוא O(2n+2k+4n+n+k) = O(7n + 3k) = O(n+k).

**add\_team –**

ראשית נוודא את תקינות הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה הקטנה או שווה ל-0 או מספר הנקודות של השחקן שלילי.

כמו כן, נוודא כי לא קיים כבר קבוצה עם המזהה הזה, אחרת נחזיר FAILURE. נקצה זיכרון עבור הקבוצה החדשה, ונחזיר ALLOCATION\_ERROR במקרה של כישלון בהקצאת הזיכרון.

נוסיף את הקבוצה לעץ AVL שמרכז את כל הקבוצות המסודרות לפי מזהה קבוצה – m\_teamsByID. נשים לב כי בהוספת קבוצה חדשה, מספר השחקנים בה הוא 0 ולכן הקבוצה לא מתווספת לעץ AVL של m\_qualifiedTeams והמצביעים לקבוצות הסמוכות לא מתעדכנים. בסוף נחזיר SUCCESS.

סיבוכיות זמן:

נעשה חיפוש לפי מזהה קבוצה בעץ AVL m\_teamsByID כדי להכניס את הקבוצה החדשה, בסיבוכיות של O(log(k)). לאחר ההכנסה, נבצע גלגולים בעץ כדי לאזן אותו בחזרה, בסיבוכיות של O(1) כפי שנלמד בהרצאה.

לכן סך סיבוכיות הזמן של הפעולה היא O(log(k)).

**remove\_team –**

ראשית נוודא את תקינות הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה שחקן קטן או שווה ל-0.

נחפש בעץ AVL של הקבוצות המסודרות לפי מזהה קבוצה - m\_teamsByID, ונחזיר FAILURE במקרה שקיימים עדיין שחקנים בקבוצה, או שהקבוצה לא קיימת.

אחרת, נסיר את הקבוצה על ידי הסרת הnode בעץ AVL m\_teamsByID (נשים לב כי אם לקבוצה אין שחקנים, אז הוא לא קיים בעץ m\_qualifiedTeams ולכן אין צורך להסיר אותו משם). נשחרר את הזכרון של הקבוצה הזו, ובסוף נחזיר SUCCESS.

סיבוכיות זמן:

בפעולה זו אנו מחפשים את הקבוצה המבוקשת פעמיים נפרדות: פעם אחת לבדוק אם קיימים שחקנים בקבוצה – חיפוש בעץ m\_teamsByID, ופעם שנייה כדי להסיר את הקבוצה מ-m\_teamsByID. בשני המקרים, סיבוכיות הזמן של החיפוש הוא O(log(k)) כי יש מקסימום k קבוצות בעץ הזה. ההסרה מהעץ כוללת גלגולים בשיטה שנלמדה בהרצאה – בסיבוכיות O(1) לכל גלגול. בפוטנציאל יש log(k) גלגולים.

לכן סך סיבוכיות הזמן הוא O(log(k)\*3) = O(log(k)).

**add\_player –**

ראשית נוודא את תקינות הקלט, ואם הוא אינו תקין אז נחזיר INVALID\_INPUT.

כמו כן, נוודא כי לא קיים כבר שחקן עם המזהה הזה וקיימת קבוצה עם המזהה הניתן, אחרת נחזיר FAILURE.

ניצור את השחקן עם הנתונים מהקלט חוץ מכמות המשחקים שהשחקן שיחק – כמות המשחקים ששחקן שיחק מחושב לפי כמות המשחקים שהקבוצה שיחקה + כמות המשחקים שהשחקן שיחק ללא הקבוצה. כדי לא להוסיף אליו משחקים לא רלוונטיים (שהקבוצה שיחקה לפני שהשחקן הצטרף), אז נחסיר מכמות המשחקים של השחקן את כמות המשחקים שהקבוצה שיחקה לפני הצטרפותו.

נכניס אותו לשני עצי AVL הכלליים של השחקנים: אחד המסודר לפי מזהה קבוצה, והשני מסודר לפי ה-score של השחקן. נעדכן את המצביע של השחקן עם ה-score הגבוה ביותר. בנוסף, נעדכן את המצביעים לשחקנים "הקרובים" ביותר מבחינת score, ובכן נעדכן גם את המצביעים של השחקנים ה"קרובים".

נוסיף את השחקן גם לקבוצה מתאימה: נוסיף את השחקן לשני עצי AVL של השחקנים של אותה הקבוצה: עץ אחד המסודר לפי מזהה הקבוצה, ועץ שני המסודר לפי ה-score של שחקני הקבוצה. בנוסף לכך, נעדכן פרטים של הקבוצה בהתאם לנתונים של השחקן החדש: נוסיף את כמות השערים של השחקן לכמות סה"כ שהבקיעה הקבוצה, ונוסיף את כמות הכרטיסים של השחקן לכמות סה"כ כרטיסים שקיבלה הקבוצה. כמו כן, נוסיף 1 לכמות השוערים של הקבוצה אם השחקן החדש יכול לשחק כשוער.

לאחר מכן, נוסיף את הקבוצה לm\_qualifiedTeams במקרה שכעת הקבוצה בעלת לפחות 11 שחקנים ויש לפחות שוער אחד. בהתאם לכך, נעדכן את המצביעים של אותה קבוצה לקבוצות הסמוכות אליו מבחינת מזהי הקבוצה, ונעדכן את המצביעים של הקבוצות הסמוכות.

לאורך כל הפעולה, במקרה של כישלון בהקצאת הזיכרון, נחזיר ALLOCATION\_ERROR.

בסוף, נוסיף 1 לכמות השחקנים שיש סה"כ בגביע העולם, ונחזיר SUCCESS.

סיבוכיות זמן:

בפעולה זו, מחפשים אם השחקן כבר קיים ומכניסים אותו לשני עצי AVL של שחקני המשחק בסיבוכיות של O(log(n)) כאשר n היא כמות השחקנים סה"כ בגביע העולם ובכן כמות השחקנים המקסימלית בכל אחד מהעצים הללו. כמו כן, מחפשים ומעדכנים את שני עצי AVL של הקבוצות במשחק בסיבוכיות של O(log(k)) כאשר k היא כמות הקבוצות סה"כ בגביע העולם, ובכן הכמות המקסימלית של קבוצות בשני העצים.

כמו כן, בקבוצה הספציפית של השחקן, מכניסים את השחקן לשני העצים של שחקני הקבוצה. בכל אחד מהעצים יש מקסימום n שחקנים, ולכן סיבוכיות החיפוש וההכנסה של כל אחד מהעצים הוא O(log(n)). מאזנים את כל אחד מהעצים הללו בעזרת גלגולים בשיטה שנלמדה בהרצאה, בסיבוכיות O(1).

גם מתעדכן המצביע לשחקן שהבקיע הכי הרבה שערים – גם של המשחק כולו וגם של הקבוצה. פעולה זו דורשת חיפוש בעצים של השחקנים עם מקסימום n שחקנים ובכן בסיבוכיות של O(log(n)).

בנוסף, מעדכנים את המצביעים לשחקנים הכי "קרובים" מבחינת score – אשר כל עדכון דורש 2 חיפושים בעץ השחקנים בסיבוכיות O(log(n)). מעדכנים זאת עבור שלוש שחקנים שונים, ולכן מחפשים 6 פעמים בעץ.

מעדכנים באופן דומה את המצביעים של הקבוצה לקבוצות בעלות מזהים סמוכים. גם כאן, כל עדכון דורש שני חיפושים בעץ הקבוצות ולכן בעלות סיבוכיות O(log(k)). מעדכנים 3 קבוצות ולכן מחפשים 6 פעמים.

לכן סך הכל סיבוכיות הזמן הוא: O(log(n)\*11 + log(k)\*8) = O(log(n)+log(k)).

**remove\_player-**

בפעולה זו, נסיר את השחקן מגביע העולם כולו.

נבדוק את תקינות הקלט, ונחזיר INVALID\_ID אם התקבל מזהה שחקן קטן או שווה ל-0. נחפש את השחקן בעץ כל השחקנים המסודר לפי מזהה שחקן, ונחזיר FAILURE אם השחקן לא קיים. אחרת, ניגש לקבוצה בה משחק השחקן דרך המצביע שנמצא באובייקט השחקן, ונסיר אותו מהקבוצה על ידי הסרתו משני העצים של השחקנים של הקבוצה: אחד המסודר לפי מזהה שחקן, והשני לפי ה-score שלו. בנוסף לכך, אם השחקן היה שוער, נפחית ב-1 את כמות השוערים של הקבוצה, ונחסיר 1 מסך השחקנים של הקבוצה. אם כעת הקבוצה כבר אינה כשרה למשחק (מכילה פחות מ-11 שחקנים או שאין בה שוערים) נסיר את הקבוצה מהעץ: m\_qualifiedTeams, ונעדכן את המצביעים של הקבוצה לקבוצות עם מזהים סמוכים, וגם נעדכן את המצביעים של הקבוצות הסמוכות.

לאחר מכן, נסיר את השחקן משני העצים של שחקני גביע העולם: אחד המסודר לפי מזהה שחקן והשני לפי ה-score שלו. נעדכן את המצביעים לשחקן עם ה-score הכי הגבוה, גם של כלל גביע העולם וגם של הקבוצה בה השחקן שיחק. נעדכן את המצביעים של השחקנים שהיו הכי "קרובים" לשחקן מבחינת score, ונשחרר את הזיכרון של השחקן. נחסיר 1 מספירת כלל שחקני גביע העולם, ובסוף נחזיר SUCCESS.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים את השחקן בעץ כל השחקנים בסיבוכיות O(log(n)) כאשר n היא סך כל השחקנים במערכת. בנוסף לכך, אנו מסירים את השחקן מארבע עצים שונים: שני העצים של שחקנים המסודרים לפי score: אחד של כלל גביע העולם, השני של הקבוצה של השחקן ושני העצים של שחקנים המסודרים לפי מזהה שחקן. בכל אחד מהעצים יש לכל היותר n שחקנים, ולכן סיבוכיות ההסרה היא O(log(n)) לכל עץ, שכולל חיפוש בעצים ואז גלגולים נדרשים כדי לאזן את העצים. עדכון המצביעים של השחקנים עם ה-score הגבוה ביותר הוא גם בסיבוכיות O(log(n)) לכל מצביע (אחד של כלל גביע העולם, השני של הקבוצה הספציפית).

אנו גם מסירים את הקבוצה מעץ הקבוצות הכשרותm\_qualifiedTeams , אשר היא מכילה רק את הקבוצות עבורן יש לפחות 11 שחקנים – נסמן מספר זה כ-m. לכן בהכרח כמות הקבוצות (nodes) בעץ הזה הוא קטן ממש מכמות השחקנים במערכת: m < n. הסרת הקבוצה נעשה בסיבוכיות O(log(m)) סך הכל, אשר כולל גם גלגולים שנדרשים כדי לאזן את העץ. לכן פעולות אלו הן בסיבוכיות זמן של O(log(m)) = O(log(n)).

עדכון נתוני הקבוצה וגביע העולם נעשה באופן ישיר למשתנים אשר הם members של Team ושל world\_cup\_t ולכן מוסיפים סיבוכיות זמן של O(1). כמו כן, עדכון השחקנים שהכי קרובים לשחקן מבחינת score הוא בסיבוכיות O(1) משום שהעדכון הוא ישיר דרך המצביעים שהחזיק השחקן. באופן דומה, עדכון המצביעים של הקבוצה לקבוצות הסמוכות מבחינת מזהי הקבוצה הוא גם בסיבוכיות O(1).

לכן סך סיבוכיות הזמן הוא: O(log(n)\*8) = O(log(n)).

**update\_player\_stats –**

בפעולה זו, נעדכן את נתוני השחקן.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה שחקן קטן או שווה ל-0, או שהתקבל נתון אחר שלילי. נחפש את השחקן בעץ כל השחקנים המסודר לפי מזהה שחקן, ונחזיר FAILURE אם השחקן לא קיים. אחרת, ניגש דרך השחקן לקבוצה שבה משחק, באמצעות מצביע לקבוצה אשר קיים באובייקט השחקן. נסיר את השחקן משני העצים של שחקנים המסודרים לפי score של השחקנים: אחד של כלל גביע העולם, והשני של הקבוצה בה השחקן משחק. נעדכן את השחקן לפי הנתונים מהקלט, ואז נחזיר חזרה את השחקן המעודכן לתוך שני העצים של שחקנים המסודרים לפי score של השחקנים: אחד של כלל גביע העולם, והשני של הקבוצה בה השחקן משחק. נעדכן את המצביעים לשחקן עם ה-score הכי גבוה בהתאם לכך. לאחר מכן, נעדכן את נתוני הקבוצה: נוסיף את השערים לסך כל השערים שהבקיעו כל שחקני הקבוצה, ונוסיף את כמות הקלפים לסך כל הקלפים של הקבוצה. בנוסף, נעדכן את המצביעים לשחקנים "הקרובים" ביותר לשחקן מבחינת score, ובכן נעדכן גם את המצביעים הללו בשחקנים שהשחקן הנוכחי הקרוב אליהם.

לאורך כל הפעולה, במקרה של כישלון בהקצאת זיכרון נחזיר ALLOCATION\_ERROR. בסוף נחזיר SUCCESS.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים את השחקן בעץ כל השחקנים בסיבוכיות O(log(n)) כאשר n היא סך כל השחקנים במערכת. בנוסף לכך, אנו מסירים ואז מכניסים את השחקן לשני העצים של שחקנים המסודר לפי score: אחד של כלל גביע העולם, השני של הקבוצה של השחקן. בכל אחד מהעצים יש לכל היותר n שחקנים, ולכן סיבוכיות ההכנסה וההסרה היא O(log(n)) לכל עץ, שכולל חיפוש בעצים ואז גלגולים נדרשים כדי לאזן את העצים. עדכון המצביעים של השחקנים עם ה-score הגבוה ביותר הוא גם בסיבוכיות O(log(n)) לכל מצביע (אחד של כלל גביע העולם, השני של הקבוצה הספציפית. עדכון נתוני השחקן והקבוצה נעשה באופן ישיר למשתנים אשר הם members של Player ו-Team ולכן מוסיפים סיבוכיות זמן של O(1).

בנוסף, עדכון המצביעים לשחקנים הכי "קרובים" מבחינת score – אשר כל עדכון דורש שני חיפושים בעץ השחקנים בסיבוכיות O(log(n)). מעדכנים זאת עבור שלוש שחקנים שונים, ולכן מחפשים 6 פעמים בעץ.

לכן סך סיבוכיות הזמן הוא: O(log(n)\*13) = O(log(n)).

**play\_match–**

בפעולה זו, שני שחקנים משחקים אחד מול השני.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם אחד ממזהי הקבוצות קטנים או שווים ל-0, או שהתקבלו מזהי קבוצות זהות. נחפש את שתי הקבוצות הללו בעץ m\_qualifiedTeams, אשר מכיל מלכתחילה את הקבוצות שכשרות למשחק – עם לפחות 11 שחקנים ושוער אחד. במקרה שלפחות אחת מהקבוצות לא כשרה או לקיימת, נחזיר FAILURE.

הקבוצה המנצחת בעלת הסכום הגבוה ביותר של: כמות הנקודות + כמות השערים של סך שחקניה – כמות הקלפים של סך שחקניה. הקבוצה המנצחת תזכה בעוד 3 נקודות, ובמקרה של תיקו, שתי הקבוצות יזכו בנקודה אחת. בסוף נחזיר SUCCESS.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים את שתי הקבוצות בעץ m\_qualifiedTeams, אשר מכיל מקסימום k קבוצות (מתוך סך k הקבוצות בכלל גביע העולם). לכן החיפוש היא בסיבוכיות O(log(k)\*2) = O(log(k)). הגישה לכל אחד מהנתונים של הקבוצות, כמו סך השערים שכל שחקניה הבקיעו היא ישירה דרך משתנים שהם members של מחלקת Team. ולכן גישה זו היא בסיבוכיות O(1) לכל משתנה. לכן סך סיבוכיות הזמן הוא O(log(k)).

**get\_num\_played\_games –**

בפעולה זו, נחזיר את כמות המשחקים שהשחקן שיחק סה"כ.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה שחקן שלילי. לאחר מכן, נחפש את השחקן בעץ כל השחקנים לפי מזהה שחקן (m\_playersByID). אם לא קיים שחקן כזה, נחזיר FAILURE. אחרת, ניגש דרך השחקן אל הקבוצה שבה הוא משחק (יש באובייקט השחקן מצביע ישירות לקבוצה). נשלוף מהקבוצה את כמות המשחקים שהקבוצה שיחקה סך הכל, ונוסיף את ערך זה לכמות המשחקים שהשחקן שיחק. נחזיר את הסכום שהתקבל.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים בעץ כל השחקנים בשיטה שנלמדה בהרצאה בסיבוכיות O(log(n)) כאשר n הוא כמות השחקנים בגביע העולם. בנוסף לכך, אנו ניגשים ישירות לקבוצה בה השחקן משחק דרך המצביע בשחקן – בסיבוכיות O(1). לכן סך הכל, סיבוכיות הזמן הוא O(log(n)).

**get\_team\_points–**

בפעולה זו, נחזיר את כמות הנקודות השקבוצה צברה סה"כ.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה שלילי. לאחר מכן, נחפש את הקבוצה בעץ כל הקבוצות לפי מזהה קבוצה (m\_teamsByID). אם לא קיימת קבוצה כזו, נחזיר FAILURE. אחרת, נחזיר את כמות הנקודות של אותה קבוצה.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים בעץ כל הקבוצות בשיטה שנלמדה בהרצאה בסיבוכיות O(log(k)) כאשר k הוא כמות הקבוצות בגביע העולם. בקבוצה, אנו ניגשים ישירות למשתנה מטיפוס int אשר הוא member של מחלקת Team, ולכן בסיבוכיות הזמן של O(1). לכן סך הכל, סיבוכיות הזמן הוא O(log(k)).

**unite\_teams –**

בפעולה זו, נאחד בין שתי קבוצות הנתונות לפי מספר ID אל תוך קבוצה חדשה.

ראשית, נבדוק את תקינות הקלט שקיבלנו - נבדוק שהמספרי ID תקינים לפי הגדרת מספרי ID. לאחר מכן, נחפש בעץ של הקבוצות (m\_teamsyID) לבדוק האם קיימת כבר קבוצה עם הID החדש שנתון לנו. אם קיימת קבוצה כזו, והיא לא אחת הקבוצות המתאחדות, פעולה זו לא חוקית ולכן נחזיר FAILURE. לאחר מכן, נבדוק שאכן שתי הקבוצות המתאחדות קיימות במערכת, על ידי חיפוש בעץ AVL m\_teamsByID. אם אחת הקבוצות לא נמצאות, נחזיר FAILURE.

אחרי בדיקת הקלט, נבנה את הקבוצה החדשה על ידי הconstructor של המחקלה Team. נקרא לפונקציית עזר unite\_teams המאחד את שתי הקבוצות על ידי הפעולות הבאות:

* חישוב הנתונים היבשים של הקבוצה החדשה על פי סכום הנתונים של שתי הקבוצות המתאחדות (numCards, numGames, etc.).
* איחוד של עצי השחקנים m\_playersByID וm\_playersByScore על ידי הפעולה insert של העץ AVL.
* עדכון הtopScorer על ידי מעבר על העץ m\_playersByScore החדש (search\_and\_return\_max).

לאחר איחוד שתי הקבוצות, נעדכן את הteam של כל שחקן באופן רקורסיבי על ידי מעבר על כל הnodes בעץ.

נסיר את שתי הקבוצות הישנות מהעץ m\_qualifiedTeams (אם הן לא נמצאות, נתפוס את החריגה שנזרקה ונתקדם), ומהעץ m\_teamsByID. נכניס את הקבוצה החדשה שבנינו אל תוך העץ m\_teamsByID על ידי פעולת insert של העץ AVL. נבדוק האם הקבוצה החדשה חוקית בשביל לשחק, ואם כן, נוסיף אותה לעץ m\_qualifiedTeams. נעדכן את המצביעים לקבוצות הסמוכות, של הקבוצות שהיו סמוכות שתי הקבוצות שאיחדנו, ובכן נעדכן המצביעים לקבוצות הסמוכות של הקבוצה החדשה. לאורך כל הפעולה, במקרה של כישלון בהקצאת זיכרון נחזיר ALLOCATION\_ERROR. בסוף נחזיר SUCCESS.

סיבוכיות הזמן:

בפעולה זו, אנחנו מתחילים מ3 חיפושים בתוך העץ m\_teamsByID, הנעשו בשיטה הנלמדה בהרצאה, בסיבוכיות זמן של O(log(k)), כאשר k הוא מספר הקבוצות במערכת. יצירת הקבוצה החדשה וגישה לpoints של שתי הקבוצות הישנות נעשה בO(1). באיחוד של הקבוצות, גישה לנתונים של שתי הקבוצות הישנות וסכומם נעשה בO(1). פעולת הmergeNodes, הנעשה על כל שחקן בשני העצים playersByID וplayersByScore דורש מעבר על כל שחקן בתוך שתי הקבוצות. פעולת הinsert שנעשה על כל שחקן לוקח זמן של O(log(n)), כאשר n הוא מספר הnodes בעץ. לכן, בסך הכל, פעולת הmergeNodes לוקח זמן של O(log(n1+n2)+n1+n2), כאשר n1 וn2 זה כמות השחקנים בכל אחת משתי הקבוצות המתאחדות. פעולת update\_team\_id דורש מעבר על כל שחקן בעץ באופן רקורסיבי, ולכן בעל סיבוכיות זמן של O(n1+n2). פעולת הremove הנעשה לאחר מכן במקסימום 4 פעמים לוקח זמן של O(logk), כי הוא נעשה לפי השיטה הנלמדה בהרצאה. כמו כן, פעולת הinsert הנעשה במקסימום פעמיים לוקח זמן של O(log(k)). עדכון המצביעים לקבוצות הסמוכות של הקבוצות שהתאחדו הוא ישיר דרך המצביעים ולכן בסיבוכיות של O(1). עדכון המצביעים לקבוצות הסמוכות לקבוצה החדשה הוא בסיבוכיות של O(log(k)\*2) לכל עדכון, ואנו מעדכנים סך הכל שלוש קבוצות. בדיקת תקינות הקבוצה נעשתה בזמן O(1). לכן בסך הכל, נקבל זמן של: n1+n2) O(3\*log(k)+ log(n1+n2)+n1+n2+ n1+n2+4\*log(k)+2\*log(k)+6\*log(k)) = O(log(k)+

**get\_top\_scorer –**

בפעולה זו, נחזיר את מזהה השחקן עם ה-score הכי גבוה.

ה-score של השחקן נקבע לפי הפרמטרים הבאים: השחקן עם כמות השערים הגבוה ביותר ששחקן הבקיע. במקרה של שוויון, אז השחקן עם כמות הקלפים הנמוך ביותר. במקרה של שוויון, אז השחקן עם מזהה הקבוצה הכי גבוה.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה השווה ל-0. כמו כן, אם אין שחקנים כלל המשתתפים בגביע העולם, נחזיר FAILURE.

* אם התקבל מזהה קבוצה הגדולה מ-0, אז נחפש את הקבוצה בעץ כל הקבוצות לפי מזהה קבוצה (m\_teamsByID). אם לא קיימת קבוצה כזו או אם בקבוצה זו אין שחקנים אז נחזיר FAILURE. אחרת, נחזיר את מזהה השחקן שמשחק בקבוצה זו עם ה-score הכי גבוה מתוך כלל השחקנים במשחקים בקבוצה זו.
* אחרת, התקבל מזהה קבוצה הקטנה מ-0, ולכן נחזיר את מזהה השחקן עם ה-score הכי גבוה מתוך כלל השחקנים בגביע העולם.

סיבוכיות הזמן:

בפעולה זו, במקרה שמזהה הקבוצה גדולה מ-0, אז מחפשים בעץ כל הקבוצות בשיטה שנלמדה בהרצאה בסיבוכיות O(log(k)) כאשר k הוא כמות הקבוצות בגביע העולם. בקבוצה, ניגשים ישירות למצביע של השחקן עם ה-score הגבוה ביותר בין השחקנים באותה קבוצה, אשר הוא member של מחלקת Team, ולכן בסיבוכיות הזמן של O(1). לכן סך הכל, סיבוכיות הזמן הוא O(log(k)).

במקרה שמזהה הקבוצה קטנה מ-0, אז ניגש ישירות למצביע לשחקן עם ה-score הגבוה ביותר בגביע העולם, אשר הוא member של מחלקת world\_cup\_t. לכן סיבוכיות הזמן סך הכל הוא O(1).

**get\_all\_players\_count -**

בפעולה זו, נחזיר את מספר השחקנים בקבוצה או בגביע העולם.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה השווה ל-0.

* אם התקבל מזהה קבוצה הגדולה מ-0, אז נחפש את הקבוצה בעץ כל הקבוצות לפי מזהה קבוצה (m\_teamsByID). אם לא קיימת קבוצה כזו אז נחזיר FAILURE. אחרת, נחזיר כמות השחקנים באותה קבוצה.
* אחרת, התקבל מזהה קבוצה הקטנה מ-0, ולכן נחזיר את כמות השחקנים בגביע העולם.

סיבוכיות הזמן:

בפעולה זו, במקרה שמזהה הקבוצה גדולה מ-0, אז מחפשים בעץ כל הקבוצות בשיטה שנלמדה בהרצאה בסיבוכיות O(log(k)) כאשר k הוא כמות הקבוצות בגביע העולם. בקבוצה, ניגשים ישר למשתנה מטיפוס int שמחזיק את כמות השחקנים באותה קבוצה, אשר הוא member של מחלקת Team, ולכן בסיבוכיות הזמן של O(1). לכן סך הכל, סיבוכיות הזמן הוא O(log(k)).

במקרה שמזהה הקבוצה קטן מ-0, אז ניגש ישירות למשתנה מטיפוס int שמחזיק את כמות השחקנים בגביע העולם, אשר הוא member של מחלקת world\_cup\_t. לכן סיבוכיות הזמן סך הכל הוא O(1).

**get\_all\_players –**

בפעולה זו, נחזיר את מזהי כל השחקנים בקבוצה או בגביע העולם.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה השווה ל-0 או שהמערך שהתקבל מצביע ל-NULL. נחזיר FAILURE אם לא קיימים כלל שחקנים בגביע העולם.

* אם התקבל מזהה קבוצה הגדולה מ-0, אז נחפש את הקבוצה בעץ כל הקבוצות לפי מזהה קבוצה (m\_teamsByID). אם לא קיימת קבוצה כזו או שאין בה שחקנים, אז נחזיר FAILURE. אחרת, נעבור על כל השחקנים בקבוצה בסדר עולה לפי ה-score של השחקנים (בעצם נעבור על העץ m\_playersByScore של אותה קבוצה בשיטת in-order), ונכניס את מזהה השחקנים למערך שהתקבל בקלט.
* אחרת, התקבל מזהה קבוצה הקטנה מ-0, ונעבור על כל השחקנים בגביע העולם בסדר עולה לפי ה-score של השחקנים (בעצם נעבור על העץ m\_playersByScore של כל גביע העולם בשיטת in-order), ונכניס את מזהה השחקנים למערך שהתקבל בקלט.

בסוף נחזיר SUCCESS.

סיבוכיות הזמן:

בפעולה זו, במקרה שמזהה הקבוצה גדול מ-0, אז מחפשים בעץ כל הקבוצות בשיטה שנלמדה בהרצאה בסיבוכיות O(log(k)) כאשר k הוא כמות הקבוצות בגביע העולם. בקבוצה, עוברים על כלל השחקנים בקבוצה, אשר נסמן כ-nTEAMID. לכן סך הסיבוכיות זמן במקרה זה הוא O(log(k)+nTEAMID).

במקרה שמזהה הקבוצה קטנה מ-0, אז נעבור על כלל השחקנים בכל גביע העולם, אשר נסמן ב-n. לכן סיבוכיות הזמן עבור מקרה זה הוא O(n).

**get\_closest\_player -**

בפעולה זו, נחזיר את מזהה השחקן שהכי "קרוב" לשחקן עם מזהה מהקלט מבחינת score.

"קירבה" ב-score מוגדר באופן הבא: השחקן הקרוב הוא זה עם כמות ה-goals הכי קרוב אליו. במקרה של שוויון, אז השחקן הקרוב הוא זה עם כמות השערים הכי קרוב אליו. במקרה של שוויון, אז השחקן הקרוב הוא זה עם מזהה השחקן הכי קרוב אליו. במקרה של שוויון, אז השחקן הקרוב הוא זה עם מזהה השחקן הגבוה ביותר.

ראשית נבדוק את הקלט, ונחזיר INVALID\_INPUT אם התקבל מזהה קבוצה או מזהה שחקן הקטן או שווה ל-0. נחזיר FAILURE אם קיימים פחות משני שחקנים בכלל גביע העולם, או אם לא קיימת קבוצה עם המזהה מהקלט. בנוסף, נחזיר FAILURE אם קיימת קבוצה כזו, אך לא קיים לה שחקן עם המזהה מהקלט. נחפש על עץ השחקנים לפי מזהה השחקן, ונשווה בין השחקים שהכי "קרובים" אליו באופן הבא: באותו שחקן יש שני מצביעים לשחקנים שהכי "קרובים" אליו – אחד עם score שגדול ממנו, והשני עם score שקטן ממנו. דרכם ניגש ישירות לשני השחקנים הללו ונשווה למי יש את ההפרש הקטן ביותר בנתונים עם השחקן הדרוש.

בסוף נחזיר את מזהה השחקן שהכי "קרוב" אליו מבחינת score.

סיבוכיות הזמן:

בפעולה זו, אנו מחפשים בעץ כל הקבוצות המסודר לפי מזהה קבוצה את הקבוצה הנדרשת, בשיטה שנלמדה בכיתה בסיבוכיות O(log(k)) כאשר k היא כמות הקבוצות בגביע העולם. לאחר מכן, אנו מחפשים בעץ כל השחקנים של אותה קבוצה את השחקן הדרוש. אם nTEAMID היא כמות השחקנים בקבוצה, אז סיבוכיות הזמן של החיפוש הוא O(log(nTEAMID)). לאחר מכן, אנו ניגשים ישירות דרך השחקן שמצאנו לשני השחקנים שהכי קרובים אליו. לכן הגישה אליהם ובדיקת מי הכי "קרוב" לשחקן הדרוש מוסיף סיבוכיות זמן של O(1).

לכן סך הכל, נקבל סיבוכיות זמן של O(log(k)+log(nTEAMID)).

**knockout\_winner -**

בפעולה זו, אנחנו מבצעים תחרות פנימית בין כמות קבוצות מסויימת, ונחזיר את מזהה הקבוצה של הקבוצה המנצחת. קודם כל, נבדוק את תקינות הקלט, ונחזיר INVALID\_INPUT במקרה שאינו תקין. אחרת, נספור את כמות הקבוצות הכשרות לשחק בטווח המזהים הנתון. אם לא קיימת קבוצה כזו, נחזיר FAILURE.

נספור את הקבוצות בעץ AVL של m\_qualifiedTeams (המכילה רק את הקבוצות בעלי לפחות 11 שחקנים ושוער אחד), ונקצה זיכרון למערך בגודל של כמות הקבוצות הכשרות בטווח הנתון. נמלא את המערך בעותק של הקבוצות המתאימות (העותק אינו כולל את השחקנים כלל, אלא רק את הנתונים של הקבוצה וגם המצביעים לעותקים של הקבוצות הסמוכות אליו מימין ומשמאל).

נעבור על כל הקבוצות הכשרות החל מהקבוצה המינימלית עד המקסימלית, בעזרת המצביעים שיש לכל קבוצה לקבוצות הסמוכות אליו, ונבצע רצף משחקים ביניהם. כל זוג קבוצות ישחקו משחק, כאשר הקבוצה המפסידה מתאחדת עם המנצחת, ומעדכנים את המצביעים של הקבוצות הסמוכות כדי לדלג על הקבוצה שהפסידה. חוזרים על תהליך זה עד שנשארת קבוצה בודדת במערך אשר היא המנצחת בטורניר. חשוב להדגיש כי כל הטורניר מתרחש בין עותקים של הקבוצות, ולכן השינויים שנעשים במהלך פעולה זו לא משפיעים לתווך ארוך על הקבוצות עצמן. לאורך כל הפעולה, במקרה של כישלון בהקצאת זיכרון נחזיר ALLOCATION\_ERROR. בסוף נחזיר את מזהה השחקן שניצח בטורניר.

סיבוכיות זמן:

ראשית, נספור את כמות הקבוצות הכשרות שיש בטווח הנתון. החיפוש נעשה ראשית בעזרת חיפוש על העץ m\_qualifiedTeams עבור השחקן עם מזהה הקבוצה המינימלי בטווח הנתון. הקבוצות בעץ זה בעלי לפחות 11 שחקנים, ולכן אם נסמן את כמות הקבוצות בעץ זה כ-m וכמות כלל השחקנים כ-n, אז m<n. אך הוא גם קטן או שווה לסך כל הקבוצות בגביע העולם, שנסמן כ-k, ולכן m≤k. לכן החיפוש עבור הקבוצה מינימלית נעשית בסיבוכיות זמן של O(log(m)), כאשר m ≤ k, m < n. נעבור על כל הקבוצות הכשרות החל מהקבוצה המינימלית בעזרת המצביעים של הקבוצות הסמוכות. יש מקסימום r קבוצות, ולכן סיבוכיות הזמן של מעבר, ספירת והעתקת הקבוצות הוא O(r). עדכון המצביעים של הקבוצות הסמוכות גם נעשה במעבר על כלל הקבוצות ולכן בסיבוכיות של O(r). חישוב המנצחים בכל משחק נעשה בעזרת גישה ישירה למשתנים של הקבוצות לכן סיבוכיות הזמן עבור כל משחק הוא O(1). עדכון המצביעים לקבוצות הסמוכות בכל משחק נעשה באופן ישיר בעזרת המצביעים ולכן זהו גם בסיבוכיות O(1).

כמות הקבוצות שאנו עוברת עליו בשלב הראשון הוא r (כל הקבוצות בטווח). בשלב השני של הטורניר, אז כבר חצי מהקבוצות הצטמצמו, והמעבר יהיה רק על מחצית הקבוצות (בעזרת המצביעים שעודכנו להצביע על הקבוצות שעדיין משחקות בטורניר ולא הפסידו). בשלב השלישי, יש מחצית מהקבוצות שהיו בשלב השני. וכן הלאה...

לכן סך הכל, נעבור על :

לכן סך סיבוכיות הזמן של כלל משחקי הטורניר הוא O(2r)=O(r).

לכן, בסך הכל, פונקציה זו לוקחת זמן של:

O(log(m)+r\*3+2r) = O(log(m)+r)

נזכר כי m<n וגם m≤k, לכן:

לכן בסך הכל, נקבל כי פונקציה זו לוקחת זמן של: O(log(min{k,n}) + r).