**תרגיל רטוב 1 – החלק היבש**

שרה גריפית – ת"ז: 341312304

ליאור בר יוסף – ת"ז: 207022443

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 יכולים לשחק (בעלי לפחות 11 שחקנים שלפחות אחד יכול לשחק כשוער).
* **m\_playersByID** – כל השחקנים שמשתתפים בגביע העולם, מסודרים לפי מזהה השחקן.
* **m\_playersByScore** – כל השחקנים שמשתתפים בגביע העולם, מסודרים לפי כמות השערים שהבקיעו + הכרטיסים שקיבלו + מזהה השחקן (במקרה של שוויון).

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.
* **מחלקת Team** תכיל את הפרטים הבסיסיים של אותה קבוצה: מזהה קבוצה, מספר הנקודות של הקבוצה, מספר השחקנים בקבוצה, מספר השוערים בקבוצה, סה"כ השערים שהקבוצה הבקיעה, סה"כ כרטיסים שהקבוצה קיבלה ומספר המשחקים שהקבוצה שיחקה.
  + 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 כמות השערים שהבקיעו + הכרטיסים שקיבלו + מזהה השחקן (במקרה של שוויון).

**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 הכמות השערים הכי גדול. כמו כן, נאתחל משתנה שסופר את כל השחקנים המשתתפים בגביע העולם ל-0.

סיבוכיות זמן:

נאתחל את כל המשתנים ל-null או לערכים דיפולטים, ולכן סיבוכיות הזמן הוא O(1).

סיבוכיות מקום:

העצים בשלב זה ריקים, והמצביע לשחקן הוא nullptr. לכן סך סיבוכיות המקום הוא O(1).

**world\_cup\_t~ –**

נשחרר את הזיכרון עבור ארבעת עצי הAVL המרכזיים והמצביע הנוסף לשחקן שהבקיע הכי הרבה שערים.

סיבוכיות זמן:

כדי לשחרר את העצים, נצטרך לעבור על כל הnodes בארבעת העצים:

* + 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. 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) = O(6n + 2k) = O(n+k).

סיבוכיות מקום:

בפעולה זו לא מקצים זיכרון נוסף ולכן אין הוספה לזיכרון של מבנה הנתונים, והוא נשאר בסיבוכיות של O(n+k).

**add\_team –**

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

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

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.

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)).

סיבוכיות מקום:

בפעולה הזו אנו מקצים זכרון לעוד קבוצה אחת אשר מתווספת לעץ AVL קיים – ולכן סיבוכיות מקום הנוסף לקיים הוא O(1). לכן סיבוכיות המקום נשאר O(n+k).

**remove\_team –**

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

נחפש בעץ AVL של הקבוצות המסודרות לפי מזהה קבוצה - m\_teamsByID, ונבדוק אם בקבוצה קיימים שחקנים. במקרה שיש שחקנים בקבוצה, או שהקבוצה בכלל לא קיימת, אז נחזיר FAILURE.

אחרת, נסיר את הקבוצה על ידי הסרת הnode בעץ AVL m\_teamsByID (נשים לב כי אם לקבוצה אין שחקנים, אז הוא לא ימצא בעץ m\_qualifiedTeams ולכן אין צורך להסיר אותו משם).

בסוף 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. השני המקרים, סיבוכיות הזמן של החיפוש הוא O(log(k)) כי יש מקסימום k קבוצות בעץ הזה.

לכן סך סיבוכיות הזמן הוא O(log(k)\*2) = O(log(k)).

סיבוכיות מקום:

בפעולה זו לא מקצים זיכרון נוסף ולכן אין הוספה לזיכרון של מבנה הנתונים, והוא נשאר בסיבוכיות של O(n+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 וכי אכן קיימת 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 ועץ שני המסודר לפי מספר השערים שהשחקן הבקיע, מספר הכרטיסים שקיבל ומזהה הקבוצה במקרה של שוויון. נעדכן את המצביע של השחקן עם הכי הרבה שערים בהתאם.

נוסיף את השחקן גם לקבוצה מתאימה:

* בכל קבוצה יש שני עצי AVL של השחקנים של אותה הקבוצה: עץ אחד המסודר לפי מזהה הקבוצה, ועץ שני המסודר לפי מספר השערים שהשחקן הבקיע, מספר הכרטיסים שקיבל ומזהה הקבוצה במקרה של שוויון. לכן נוסיף את השחקן לשני העצים הללו.
* 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.

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)).

לכן סך הכל סיבוכיות הזמן הוא: O(log(n)\*6 + log(k)\*2) = O(log(n)+log(k)).

סיבוכיות מקום:

בפעולה הזו אנו מקצים זכרון לעוד שחקן אחד אשר מתווסף ל-4 עצי AVL קיימים של שחקנים (שניים כללים, שניים של הקבוצה של השחקן), וגם במקרה הגרוע מוסיפים את הקבוצה של השחקן לעוד עץ AVL של m\_qualifiedTeams– ולכן סיבוכיות מקום הנוסף לקיים הוא O(5) = O(1). לכן סיבוכיות המקום נשאר O(n+k).

**remove\_player-**

* ראשית נבדוק אם הקלט תקין – להחזיר INVALID\_INPUT במקרה שלא
* חיפוש את השחקן הנכון בעץ AVL של Players – בסיבוכיות log(n) לפי שנלמד בהרצאה.
* אם מוציאים את השחקן, ראשית נעדכן את הקבוצה בה משחק.
  + בכל Player, יש מצביע לקבוצה מסוימת. בקבוצה זו נעשה כמה פעולות (סיבוכיות O(1) כי גישה ישירה לפי המצביע, חוץ משחרור הNODE של השחקן מעץ שחקנים של של אותה קבוצה):
    - numPlayers -= 1
    - totalGoalKeepers -= 1 אם השחקן הוא goalkeeper
      * אם numPlayers<11 או totalGoalKeepers<1 אז נסיר את הקבוצה הזו גם מהעץ qualifiedTeams – גישה ישירה לפי המצביע qualifiedTeamPtr – סיבוכיות O(1)
    - totalGoals -= goals – של השחקן שאנו מסירים
    - totalCards -= cards – של השחקן שאנו מסירים
    - allTeamPlayers – חיפוש השחקן הרלוונטי לפי playerId (סיבוכיות log(n)), והסרתו + שחרור הזכרון של הnode הזה
    - teamTopScorers – חיפוש השחקן הרלוונטי לפי איזה שחקן יש לו יותר goals. אם יש שוויון בgoals אז לפי איזה שחקן יש הכי פחות cards, ואז אם יש שוויון אז לפי playerId הכי גדול. (סיבוכיות log(n)), והסרתו + שחרור הזכרון של הnode הזה
* topScorer - חיפוש השחקן הרלוונטי לפי איזה שחקן יש לו יותר goals. אם יש שוויון בgoals אז לפי איזה שחקן יש הכי פחות cards, ואז אם יש שוויון אז לפי playerId הכי גדול. (סיבוכיות log(n))
  + אם allTopScorer מצביע על השחקן הזה, אז נחפש את השחקן הזה ונשנה את המצביע הזה.
  + הסרת הnode של השחקן הזה + שחרור הזכרון.
* החסרה של 1 מהמשתנה numAllPlayers
* בסוף משחררים את הזכרון + מוחקים את הNODE של השחקן מהעץ של Players

**update\_player\_stats –**

* ראשית נבדוק אם הקלט תקין – להחזיר INVALID\_INPUT במקרה שלא
* חיפוש השחקן המתאים שצריך לעדכן בעץ AVL של Players – סיבוכיות של O(logn) כפי שנלמדה בהרצאה
* עדכן המשתנים אצל Player:
  + gamesPlayed += gamesPlayed – נתון
  + goals += scoredGoals – נתון
    - עדכון מיקום השחקן בעץ topScorers – סיבוכיות O(logn)
    - אם goals > allTopPlayer[goals] אז נחליף את המצביע של allTopPlayer להצביע על השחקן הזה
  + cards += cardsReceived – נתון
* עדכון המשתנים אצל Team לפי הteamId של אותו שחקן – גישה ישירה דרך המצביע של השחקן לקבוצה בה משחק ולכן בסיבוכיות של O(1):
  + totalGoals += scoredGoals
  + totalCards += cardsReceived
  + teamTopScorers – עדכון מיקומו של השחקן – סיבוכיות O(logn)

**play\_match–**

* נחפש את שתי הקבוצות בעץ AVL של qualifiedTeams – סיבוכיות O(logk) כל אחד (ולכן סה"כ O(logk)).
  + נשמור את המצביעים לשתי הקבוצות הללו במשתנים מקומיים
* את הנתונים של הסכימה כבר יש לנו מוכנים, ולכן עבור כל קבוצה נסכום:
  + points + totalGoals – totalCards
* אם שני הערכים שווים – יש תיקו:
  + כל קבוצה מקבלת נקודה אחת לpoints: points += 1
* אחרת:
  + הקבוצה המנצחת מקבלת 3 נקודות: points += 3
  + הקבוצה המספידה מקבלת 0 נקודות – לא מתעדכנת
* בסוף נוסיף 1 למשתנה של teamGames

**get\_num\_played\_games –**

* חיפוש בעץ AVL של Players עבור השחקן לפי playerId – בסיבוכיות זמן של O(logn) כפי שנלמד בהרצאה
* אחזור הערך gamesPlayed של אותו שחקן + המשתנה teamGames של הקבוצה שלו – גישה ישירה דרך המצביע – סיבוכיות זמן של O(1).

**get\_team\_points–**

* חיפוש אחר הקבוצה בעץ AVL של Teams – בסיבוכיות זמן של O(logk) כפי שנלמד בהרצאה
* אחזור הערך points של אותה קבוצה

**unite\_teams –**

* חיפוש אחר שתי הקבוצות הנתונות ובדיקה אם newTeamId כבר קיים (ואינו teamId1 או teamId2) – סיבוכיות של O(logk) לשלשת הפעולות, ולכן סך הכל O(logk).
* נוציא את שני הnode-ים teamId1 ו-teamId2 מהעץ ונשמור בצד את המצביעים לשתי הקבוצות שמתבקשות להתאחד במשתנים מקומיים
* נכניס node חדש עבור newTeamId – בסיבוכיות O(logk) ונשמור מצביע עבורו גם בצד (אם newTeamId היא בעלת אחת הID-ים של הקובצות הניתנות, עדיין נבנה עבורו node חדש עם הID כפי שנדרש)
* נעדכן את המשתנים:
  + points החדש - נעדכן לסכום הpoints של שתי הקבוצות הקודמות
  + numPlayers החדש - נעדכן לסכום הnumPlayers של שתי הקבוצות הקודמות
  + totalGoalPlayers החדש - נעדכן לסכום totalGoalPlayers של שתי הקבוצות הקודמות
  + totalGoals החדש - נעדכן לסכום totalGoals של שתי הקבוצות הקודמות
  + totalCards החדש - נעדכן לסכום totalCards של שתי הקבוצות הקודמות
  + allTeamPlayers החדש – נאחד בין שני העצי שחקנים של הקבוצות הקודמות – סיבוכיות n(teamId1)+n(teamId2)
    - בזמן המעבר על השחקנים – נשנה את המצביע של currentTeam של השחקן למצביע לקבוצה החדשה newTeamId
    - בנוסף נעדכן את הgamesPlayed של כל שחקן להיות gamesPlayed+teamGames
  + teamTopScorers החדש – נאחד בין שני העצי שחקנים של הקבוצות הקודמות – סיבוכיות n(teamId1)+n(teamId2)

**get\_top\_scorer –**

* אם teamId<0:
  + יש עץ AVL של top\_scorer שמסודר לפי goals (עולה) ואז סידור משנה של cards (יורד) ואז סידור שלישי לפי playerId (עולה). ניגשים אליו דרך מצביע נפרד שמצביע לאיבר הכי ימני – סיבוכיות זמן O(1)
  + ככה נמצא את השחקן עם מספר הgoals הכי גדול. במקרה של תיקו נעבור למספר הcards הכי קטן. במקרה של תיקו נעבור לplayerId הכי גדול.
* במקרה teamId>0:
  + ניגש לteam הרלוונטי – סיבוכיות זמן של O(logk).
    - בתוך כל Team, יש עץ של allTeamPlayers המסודר לפי goals (עולה) ואז סידור משנה של cards (יורד) ואז סידור שלישי לפי playerId (עולה).
    - ניגשים אליו דרך מצביע נפרד שמצביע לאיבר הכי ימני – סיבוכיות זמן O(1)
    - ככה נמצא את השחקן עם מספר הgoals הכי גדול. במקרה של תיקו נעבור למספר הcards הכי קטן. במקרה של תיקו נעבור לplayerId הכי גדול.

**get\_closest\_player -**

* יש צורך בעץ מטיפוס PLAYER של כל השחקנים במערכת שמסודר לפי הכללים המפורטים: שערים -> כרטיסים -> מזהה שחקן.
* ראשית נחפש את הקבוצה המחפשת בעץ כל הקבוצות
* אז, נחפש בעץ של הPLAYERBYID של אותה קבוצה, את השחקן הספציפי
* השחקן הזה מחזיק מצביע למיקום שלו בעץ של כל השחקנים ALLPLAYERSBYGOAL
  + ניצור תת מחלקה של AVLTREE שמכיל שדה נוסף שהוא מצביע לNODE, והעץ הPLAYERBYID של הקבוצה תהיה מהטיפוס הזה, והמצביע הנוסף הוא זה שיצביע על אותו שחקן בעץ ALLPLAYERSBYGOAL
* צריך לבדוק בעץ של הALLPLAYERSBYGOAL את שני השחקנים שהכי סמוכים אליו – מי הכי קרוב לשחקן המבוקש – אולי בפונקציית עזר של השוואה מול שני שחקנים

**knockout\_winner -**

* יש צורך בעץ נוסף – שמכיל רק את הקבוצות ה"חוקיות" (לפחות 11 שחקנים ולפחות שוער אחד)
* מחפשים את הTEAMID המינימלי – סיבוכיות log(k), ועולים למעלה בעץ לפי סיור INORDER עד הTEAMID המקסימלי
* **רקורסיה בתוך רקורסיה** – ניצור בצד מערך/רשימה בו **עותק** (חלקי) של הקובצות המשחקות, ועליו נפעיל את כל הטורניר
  + בעצם ניצור פונקציית עזר בתוך TEAMS שיחזיר את הנתונים הרלוונטים כעותק של הTEAM, רק ללא שחקנים. לכן ההעתקה תהיה מסיבוכיות קבועה – ככמות המשתנים הרלוונטיים.
  + ואז את העותקים האלו של הקבוצות, נכניס כאיבר במערך/רשימה הגדול באורך r
  + את הTEAMS האלו נשלח לUNITE במקרה הצורך, שבו יש מקרה שמטפל באיחוד קבוצות ריקות
  + את כל העדכונים (כמו UNITE וכו'...) נעשה על המערך/רשימה החדש שיצרנו – כך לא נשנה את הקבוצות המקוריות
  + לפני הUNITE TEAMS, שולחים לPLAYMATCH ואז מוסיפים נקודות איפה שצריך, ורק אז עושים UNITE TEAMS, אשר מחזירה את המזהה של הקבוצה המנצחת