**תרגיל רטוב 1**

**מחלקות הנדרשות:**

מחלקת Team:

* משתנים:
  + int teamId
  + int points
  + Players\* allTeamPlayers (עץ AVL)
  + Players\* teamTopScorers
  + Player\* topTeamPlayer
  + Team\* qualifiedPlayerPtr
  + int numPlayers
  + int totalGoalPlayers
  + int totalGoals
  + int totalCards
  + int teamGames
* פעולות:
  + שני בנאים – אחד שנדרש מadd\_team, והשני שנדרש מadd\_player (עם אופציה של להוסיף שחקן)

מחלקת Player:

* משתנים:
  + int playerId
  + int gamesPlayed
  + int goals
  + int cards
  + bool goalkeeper

**מבנה נתונים הנדרש:**

* עץ AVL של Players (מטיפוס מצביעים לPlayer)
  + מצביע לבן שמאלי
  + מצביע לבן ימני
  + מצביע לאב
  + גובה העץ - int
  + מצביע לקבוצה של השחקן - Team\* currentTeam
  + משתנה מטיפוס Player
* עץ AVL של Teams (מטיפוס מצביעים לTeam)
  + מצביע לבן שמאלי
  + מצביע לבן ימני
  + מצביע לאב
  + גובה העץ
  + ומשתנה מטיפוס TEAM
* עץ AVL של topScorer (מטיפוס מצביעים לPlayer)
  + בנוסף אליו, מצביע לnode הכי ימני בנפרד - allTopPlayer
* משתנה int שסוכם את כל השחקנים – numAllPlayers
* עץ AVL של qualifiedTeams (מטיפוס מצביעים לPlayer)

**פונקציות:**

world\_cup\_t() –

* אתחול ארבעת עצי הAVL: Players, Teams, topScorer, qualifiedTeams – עצים ריקים.
* אתחול מצביע לPlayer – מצביע לnull בהתחלה
* אתחול numAllPlayers ל-0

virtual ~world\_cup\_t() –

* שחרור ארבעת עצי הAVL
* יצריך מעבר על כל הnodes של כל העצים – ולכן סיבוכיות הזמן O(n+k).

StatusType add\_team(int teamId, int points) –

* ראשית נבדוק אם הקלט תקין – להחזיר INVALID\_INPUT במקרה שלא
* הכנסה של node חדש לעץ AVL של Teams – בסיבוכיות O(logk) כפי שנלמד בהרצאה. סדר ההכנסה – לפי teamId. וגלגול איפה שצריך – סיבוכיות זמן של O(1).
* אתחול המשתנים:
  + teamId – נתון
  + points – נתון
  + allTeamPlayers – עץ ריק
  + teamTopPlayers – עץ ריק
  + teamTopScorers – מצביע לnull
  + numPlayers – 0
  + totalGoalKeepers – 0
  + totalGoals – 0
  + totalCards – 0
  + teamGames - 0

StatusType remove\_team(int teamId) –

* ראשית נבדוק אם הקלט תקין – להחזיר INVALID\_INPUT במקרה שלא
* חיפוש על העץ הAVL של Teams את הteamId הנתון – סיבוכיות O(logk) כפי שנלמד בהרצאה.
  + אם numPlayers>0 אז להחזיר Failure
* בסוף, שחרור הnode של הקבוצה מהעץ Teams (נשים לב שהקבוצה לא מכילה שחקנים בכלל ולכן לא תהיה בעץ qualifiedTeams, וגם העצים של allTeamPlayers ו-teamTopScorers יהיו ריקים) – לכן סיבוכיות של O(1).

StatusType add\_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper) –

* ראשית נבדוק אם הקלט תקין – להחזיר INVALID\_INPUT במקרה שלא
* הוספת node חדש בעץ AVL של Players – בסיבוכיות log(n) כפי שנלמד בהרצאה. ההוספה לעץ הזה הוא לפי סדר playerId.
  + אתחול המשתנים:
    - playerId – נתון
    - gamesPlayed – נתון
    - goals – נתון
    - cards – נתון
    - goalKeeper – נתון
    - teamId – מצביע לקבוצה עם הteamId הנתון – חיפוש הקבוצה בעץ AVL של Teams – סיבוכיות של log(k).
* נשמור בצד מצביע לnode החדש של השחקן, ונשתמש בו עבור:
  + - נשלח אותו לעדכון הקבוצה של teamId הנתונה.
      * numPlayers += 1
      * totalGoalKeepers += 1 – נעדכן רק אם קיבלנו true עבור השחקן הנוכחי. אחרת לא נעדכן.
        + אם numPlayers > 10 וגם totalGoalKeepers>0 אז נוסיף node לעץ qualifiedTeams עם מצביע לקבוצה הזו. סדר ההוספה היא לפי teamId.

במקרה הזה, גם נעדכן את המצביע qualifiedTeamPtr להצביע על הnode שיצרו בqualifiedTeams.

* + - * totalGoals += goals – לפי goals הנתון של השחקן החדש
      * totalCards += cards – לפי cards הנתון של השחקן החדש
      * allTeamPlayers – הוספת node לעץ (סיבוכיות log(n)), עם מצביע לשחקן הזה, בסידור לפי teamId
      * teamTopScorers – הוספת node לעץ (סיבוכיות log(n)), עם מצביע לשחקן הזה, 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.
    - לאחר עדכון הקבוצה, נעדכן חזרה את הgamesPlayed של אותו השחקן, ונחסיר ממנו את הteamGames. כלומר, נבצע Player(gamesPlayed) -=teamGames.
* נוסיף 1 לnumAllPlayers
* נוסיף node לעץ topScorer שמצביע לשחקן הזה, 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 זמן של O(logn).
  + אם goals > allTopPlayer[goals] אז נחליף את המצביע של allTopPlayer להצביע על השחקן הזה

StatusType remove\_player(int playerId)

* 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

StatusType update\_player\_stats(int playerId, int gamesPlayed, int scoredGoals, int cardsReceived) –

* 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)

StatusType play\_match(int teamId 1, int teamId2) –

* 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

output\_t<int> get\_num\_played\_games(int playerId) –

* 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).

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

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

StatusType unite\_teams(int teamId1, int teamId2, int newTeamId) –

* 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)

output\_t<int> get\_top\_scorer(int teamId) –

* 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

output\_t <int> knockout\_winner(int minTeamId, int maxTeamId)

* 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