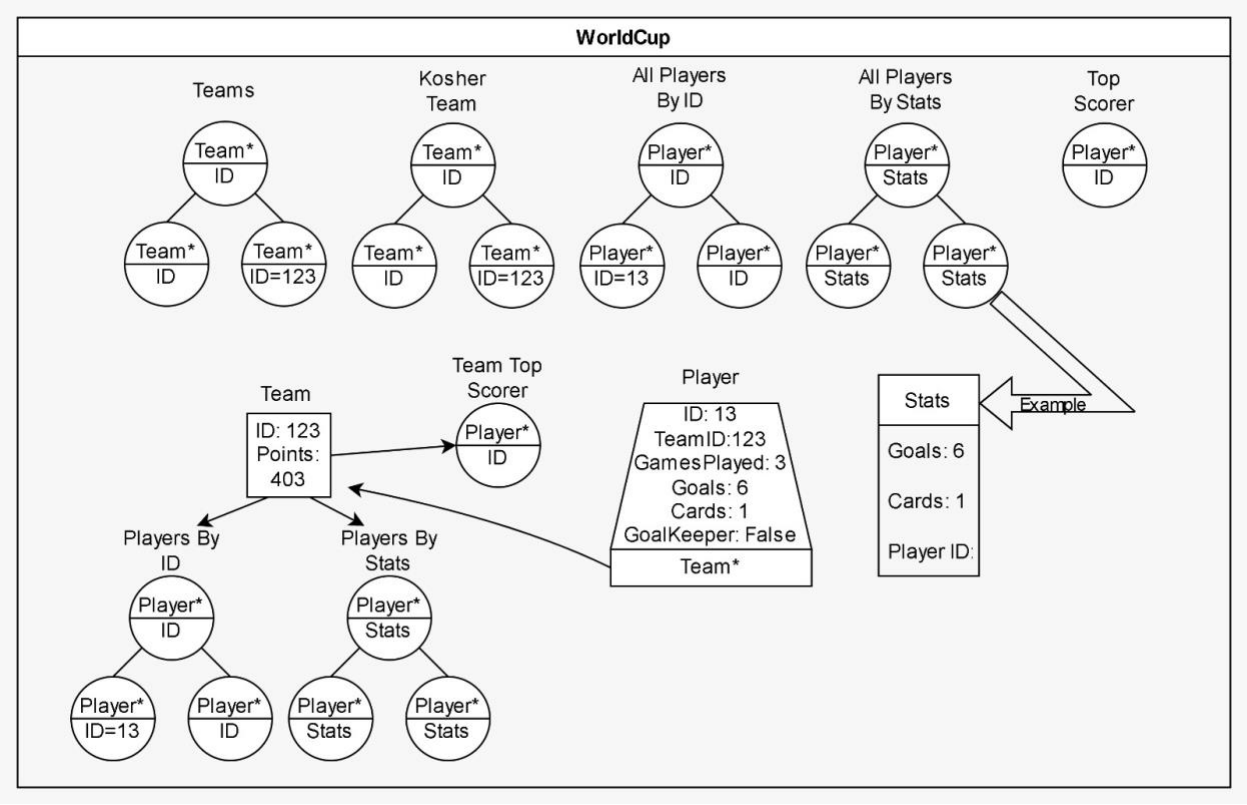
מבני נתונים

רטוב 1 – חלק יבש

מבני הנתונים- מבנה הנתונים הבסיסי של התרגיל הוא WorldCup .
מבנה זה מכיל מצביע לשחקן הטוב ביותר וארבעה עצי AVL כפי שיפורט בציור ובמילים בהמשך. עצי הAVL ממומשים כנלמד בהרצאה, בהם בכל צומת יש מצביע למפתח טמפלייטי, לפיו נמיין את העץ, ומצביע למידע שיהיה או השחקנים שבעץ, או הקבוצות. כפי שנלמד בהרצאה, חיפוש, הכנסה והוצאה של איבר לעץ יהיו חסומים על ידי O(log n). בנוסף, השחקנים גם יהיו מסודרים ברשימה מקושרת דו כיוונית, והקבוצות הכשרות ברשימה מקושרת חד כיוונית, נפרט בהמשך לגבי ההכנסה וההוצאה אליהן.



מימוש הפעולות

world_cup_t()- מאתחלת מבנה הנתונים ריק- נאתחל מצביעים לארבעה עצים ריקים- עץ של כל הקבוצות ממוין לפי הid שלהם - teams, עץ של הקבוצות ה"כשרות"- כל הקבוצות בעלות 11 שחקנים לפחות ושוער 1 לפחות – kosherTeams (ממוין גם לפי id), עץ של כל השחקנים ממוין לפי id- playersById ועץ של כל השחקנים ממוין לפי stats- הסדר כפי שמתואר בפונקציה get_closest_player- playersByStats. אתחול עץ ריק חסומה על ידי O(1) ולכן אתחול ארבעת העצים גם חסומה ע"י O(1).

~world_cup_t()- נמחק את ארבעת העצים שמרכיבים את מבנה הנתונים.
מחיקת כל עץ כוללת מעבר postorder על כל העץ ומחיקת כל צומת (מחיקת צומת היא מחיקת המצביע למפתח שלה, המידע שלה, והמצביעים לימני והשמאלי- 4 פעולות ולכן חסום ע״י O(1)). סך הכל עוברים פעם אחת על כל צומת בעץ.
בשני העצים ששומרים שחקנים יהיו n צמתים כל אחד, בteams יהיו k צמתים, ובkosherTeams, אם ישנן יותר קבוצות מאשר שחקנים- כל קבוצה שנמצאת בעץ מכילה לפחות 11 שחקנים, ובפרט לפחות שחקן אחד ולכן כמות הקבוצות בעץ תהיה קטנה או שווה לn, ואם ישנם יותר שחקנים מאשר קבוצות כמות הקבוצות בעץ תהיה קטנה או שווה לk- סך הכל, נקבל כי הגודל של kosherTeams חסום על ידי min(n, k).
נקבל שהפעולה תעבור על 2n + k + min(n, k) צמתים, כלומר תבצע כמות פעולות שחסומה על ידי 3n+k ולכן תהיה חסומה על ידי O(n + k).

add_team(int teamId, int points)- נאתחל קבוצה חדשה, בה נאתחל את הנקודות, כמות המשחקים שהיא שיחקה = 0, כמות השחקנים שבה = 0, כמות השוערים שבה = 0, כמות הכרטיסים שהיא קיבלה = 0 וכמות הגולים שהיא הבקיעה = 0.
בנוסף, יהיה מצביע ריק לשחקן שהבקיע הכי הרבה גולים – topScorer, מצביע לעץ ריק של השחקנים בקבוצה מסודרים לפי id, ומצביע לעץ ריק של השחקנים בקבוצה מסודרים לפי id.
כעת נכניס את הקבוצה לTeams כפי שנלמד בהרצאה לגבי הכנסת צומת חדש לעץ AVL (הקבוצה ריקה ולכן אין צורך להכניס אותה לkosherTeams).
אתחול קבוצה חדשה – O(1), הכנסת צומת חדש לעץ AVL בגודל k קבוצות – O(log k).
סך הכל נקבל כי הפעולה חסומה ע״י O(log k).

remove_team(int teamId)- ראשית נמצא את הקבוצה בTeams. חיפוש בעץ AVL – O(log k).
אם הקבוצה נמצאת בעץ ויש בה 0 שחקנים נסיר אותה מTeams כפי שנלמד בהרצאה להסיר צומת מעץ AVL (בkosherTeams היא בוודאות לא נמצאת מכיוון שיש בה 0 שחקנים), אחרת- Failure.
הסרה מעץ AVL- O(log k). סך הכל נקבל שהפעולה חסומה ע״י O(log k).

add_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper)- נמצא את הקבוצה בעלת המזהה המתאים – O(log k).
כעת נאתחל שחקן חדש עם כמות המשחקים שהוא שיחק פחות כמות המשחקים שהקבוצה שיחקה (עבור מניעת ספירה כפולה בחישוב עתידי של כמות המשחקים שהשחקן שיחק בו נוסיף את המשחקים שהקבוצה שיחקה למשחקים שהוא שיחק), כמות הגולים שהבקיע, כמות הכרטיסים שקיבל, האם הוא שוער ומצביע לקבוצה אליה הוא שייך.
נוסיף אותו לעצים הממויינים לפי Id ולפי Stats- גם של מבנה הנתונים כולו וגם של הקבוצה אליה הוא שייך.
נגדיל את כמות השחקנים בקבוצה ב1 ונוסיף את כמות הכרטיסים שקיבל וכמות הגולים שהבקיע לערכים המתאימים בסה״כ של הקבוצה.
נבדוק אם הבקיע יותר גולים מהtopScorer של הקבוצה ונשנה את הtopScorer להיות הוא במקרה הצורך.
אם הקבוצה לא הייתה כשרה לפני שהוא התווסף וכשרה לאחר הוספתו נוסיף את הקבוצה לkosherTeams.
אתחול השחקן חסום ע״י O(1).
הוספתו לעצים של כלל השחקנים חסומה ע״י O(log n), ובכל עץ של שחקנים של הקבוצה אליו נוסף יש לכל היותר n שחקנים ולכן גם הכנסתו אליהם חסומה ע״י O(log n).
הכנסת הקבוצה לkosherTeams חסומה ע״י O(log(min(n,k))) ולכן גם חסומה ע״י O(log n).

כעת נטפל ברשימה המקושרת של השחקנים.
לכל שחקן יש מצביע לשחקן שמדורג לפניו בStats ולשחקן שמדורג אחריו. נמצא את צומת השחקן בעץ כל השחקנים לפי Stats- O(log n).
כעת נמצא את השחקן שלפניו בעזרת האלגוריתם הבא; אם לשחקן יש בן שמאלי- נרד במורד העץ עד הבן הימני העמוק ביותר של אותו בן שמאלי והוא השחקן שלפניו. אחרת- נתחיל משורש העץ ונרד במורד העץ באופן הבא- כל עוד המצביע שאנחנו עליו אינו null או שהStats שלו אינם שווים לStats של השחקן עליו אנחנו מחפשים- אם הStats של המצביע הנוכחי קטנים מהStats של השחקן – נגדיר את השחקן שלפני להיות המצביע שאנחנו עליו ונרד ימינה בעץ. אחרת- נרד שמאלה.
במקרה הכי גרוע באלגוריתם זה נרד את כל גובה העץ ולכן הוא חסום ע״י O(log n).
באלגוריתם דומה נמצא גם את השחקן שאחריו. כעת כשמצאנו אותם נגדיר את המצביע ללפני ואחרי של השחקן הנוכחי להיות הם, ובהנחה והם לא nullptr נגדיר את המצביע אחורה/קדימה שלהם (בהתאם לאם אחרי או לפני) להיות השחקן הנוכחי.

נקבל כי סך הכל הפעולה חסומה ע״י O(log n + log k).

<u>-remove_player(int playerId)</u> נמצא את השחקן ברשימת כל השחקנים לפי Id- O(log n).
נגיע אל הקבוצה שלו בעזרת המצביע ממנו אליה.
אם השחקן הוא הtopScorer מבין כל השחקנים- נגדיר את הtopScorer להיות השחקן שלפניו. באופן דומה, אם השחקן הוא הtopScorer של הקבוצה בה הוא נמצא- נמצא את השחקן שלפניו בקבוצה (בקבוצה יש לכל היותר n שחקנים ולכן מציאת השחקן שלפניו חסומה על ידי O(log n)) ונגדיר אותו להיות הtopScorer של הקבוצה.
נסיר אותו משני עצי השחקנים של הקבוצה- O(log n) (יש לכל היותר n שחקנים בכל אחד מהעצים הללו).
אם הקבוצה הייתה כשרה ועכשיו לא, נסיר אותה מעץ הקבוצות הכשרות- O(log(min(n,k))) - חסום ע״י O(log n). נחסיר את הכרטיסים והגולים שלו מהכרטיסים והגולים של הקבוצה, נחסיר 1 ממספר השחקנים בקבוצה וממספר השוערים בקבוצה אם הוא היה שוער. נסיר אותו משני עצי כל השחקנים – O(log n). סה״כ נקבל כי הפעולה חסומה ע״י O(log n).

<u>update_player_stats(int playerId, int gamesPlayed, int scoredGoals, int -cardsReceived)</u> נמצא את השחקן ברשימת כל השחקנים לפי Id- O(log n), נעדכן את הערכים אצלו לפי דרישות הפעולה, ונעדכן בסה״כ של הקבוצה שלו דרך המצביע אליה ממנו. סה״כ O(log n).

<u>-play_match(int teamId1, int teamId2)</u> נמצא את שתי הקבוצות – O(log k) לכל אחת. נחשב את הניקוד של כל אחת לפי מספר נקודות + מספר גולים פחות מספר כרטיסים, נראה מי ניצחה או אם היה תיקו ונעדכן נקודות בהתאם. סה״כ – O(log k).

<u>-get_num_played_games(int playerId)</u> נמצא את השחקן ונחזיר את מספר המשחקים ששיחק (שזה מספר המשחקים ששיחק פחות מספר המשחקים שהקבוצה שיחקה בלעדיו) ועוד מספר המשחקים שקבוצתו שיחקה. סה״כ O(log n).

<u>-get_team_points(int teamId)</u> כל פעם שמספר הנקודות בקבוצה משתנה הוא מתעדכן ולכן נמצא את הקבוצה ונחזיר את מספר הנקודות הנוכחי שלה – O(log k).

-unite_teams(int teamId1, int teamId2, int newTeamId) ראשית, נמצא את שתי הקבוצות – O(log k). ניצור קבוצה חדשה בעלת המזהה החדש ונשים את כל הערכים שלה (מלבד העצים) להיות הסכום של הערכים בשתי הקבוצות המקוריות וtopScorer להיות השחקן בעל מספר הגולים הגבוה יותר מבין שני הtopScorer של הקבוצות.
כעת ניצור את העצים של הקבוצה מתוך העצים של שתי הקבוצות הקודמות.
האלגוריתם של יצירת עץ AVL משני עצי AVL שונים יפעל כך: ראשית נעבור על כל אחד מהעצים במעבר inorder ואחד אחד נשים את כל האיברים של העץ במערך כך שהוא יהיה ממוין (בגלל הinorder) עברנו כאן על כל אחד מאיברי העץ של כל אחת מהקבוצות פעם אחת ולכן זה יהיה חסום על ידי $O(n_{teamId1} + n_{teamId2})$.
כעת יש לנו שני מערכים ממוינים. לפני שנאחד אותם, נעבור על כל השחקנים בהם ונגדיר את המצביע של הקבוצה שלהם להיות מצביע לקבוצה החדשה ונוסיף את כמות המשחקים שהקבוצה שיחקה לכמות המשחקים שהשחקן שיחק (על מנת שנוכל להמשיך לספור את כמות המשחקים של כל שחקן ב(O(1). מעבר על כל השחקנים פעם אחת - $O(n_{teamId1} + n_{teamId2})$.
כעת נאחד אותם באופן הבא- ניצור מערך בגודל של שניהם ונרוץ על כל האיברים מהקטן לגדול כאשר בכל פעם ניקח את האיבר הקטן מבין האיבר שעליו נצביע במערך הראשון והאיבר עליו נצביע במערך השני.
עברנו כאן על כל אחד מאיברי המערכים פעם אחת ולכן גם פעולה זו תהיה חסומה על ידי $O(n_{teamId1} + n_{teamId2})$.
כעת, יש לנו מערך ממוין אחד של כל האיברים אשר היו בשני העצים המקוריים.
נהפוך אותו לעץ AVL באופן הבא- ניקח את האיבר שבאמצע המערך ונשים אותו בשורש העץ. כעת נגדיר את האיבר השמאלי שלו להיות התוצאה של אותה פונקציה כאשר מפעילים אותה על החצי הראשון של המערך לא כולל האמצע, ואת האיבר הימני שלו להיות התוצאה של הפונקציה כאשר מפעילים אותה על החצי השני של המערך לא כולל האמצע.
זהו בעצם מעבר preorder רקורסיבי ולכן נעבור על כל האיברים במערך בדיוק פעם אחת ופעולה זו תהיה חסומה על ידי $O(n_{teamId1} + n_{teamId2})$.
לאחר השימוש באלגוריתם הזה נקבל משתי הקבוצות המקוריות עץ מאוחד של כל השחקנים מסודרים לפי Id ועץ מאוחד של כל השחקנים מסודרים לפי Stats, אשר מצביעים אליהם נשים בקבוצה החדשה.
כעת, נסיר את הקבוצות מTeams ומkosherTeams במקרה הצורך – O(log k). ונוסיף את הקבוצה החדשה לTeams ולkosherTeams במקרה הצורך – O(log k). סה"כ נקבל כי הפעולה חסומה ע"י $O(logk + n_{teamId1} + n_{teamId2})$.

-get_top_scorer(int teamId) אם teamId < 0 - נחזיר את הId של topScorer- O(1). אם teamId > 0, נמצא את הקבוצה ונחזיר את הId של הtopScorer שלה- O(log k).

-get_all_players_count(int teamId) אם teamId < 0 - נחזיר את הגודל של playersById- O(1). אם teamId > 0, נמצא את הקבוצה ונחזיר את הגודל של playersById שלה- O(log k). (כל עץ שומר את הגודל שלו ומתעדכן עם כל הכנסה והוצאה).

-get_all_players(int teamId, int * const output) אם teamId < 0 - נכניס למערך את כל השחקנים מplayersByStats כך שהמערך יהיה ממוין, כמו שעשינו באחד השלבים בunite_teams. זה חסום ע"י O(n). אם teamId > 0, נמצא קודם את הקבוצה ואז נעשה אותו דבר. כך זה יהיה חסום ע"י $O(logk + logn_{teamId})$.

-get_closest_player(int playerId, int teamId) ראשית, נמצא את השחקן בתוך הקבוצה- O(log k + log n_teamId). אם השחקן שלפניו או אחריו הוא null, נחזיר את מזהה השחקן שאחריו או לפניו בהתאמה. אחרת, נשווה בין השחקן שלפניו לבין השחקן שאחריו לפי הקריטריונים שמוגדרים בפעולה ונראה מי השחקן הקרוב יותר ונחזיר את המזהה שלו. סך הכל הפעולה תהיה חסומה ע״י $O(logk + logn_{teamId})$.

-knockout_winner(int minTeamId, int maxTeamId) נתחיל ממציאת הצומת שמצביע על הקבוצה בעלת המזהה המינימלי שבתוך הטווח שבין maxTeamId לבין minTeamId מתוך עץ הקבוצות הכשרות.
נעשה זאת כך – נתחיל בשורש העץ, וניצור מצביע result שיחזיר את התוצאה שיצביע על null. כל עוד הצומת שאנחנו נמצאים בו כרגע אינה null נפעל באופן הבא:
אם המזהה של הקבוצה קטן מminTeamId - נעבור לבן הימני של הצומת שאנחנו עליו. אחרת, אם המזהה גדול מmaxTeamId - נעבור לבן השמאלי של הצומת שאנחנו עליו. אחרת, הקבוצה בטווח ולכן נגדיר את result להיות הצומת הנוכחי ונעבור לבן השמאלי.
במקרה הכי גרוע נעבור על כל גובה העץ ולכן אלגוריתם זה חסום ע״י O(log(min(n, k))).
כעת, ניצור רשימה מקושרת עבור התחרות באופן הבא- נתחיל מהצומת שמצאנו בעלת המזהה המינימלי, ונעבור על הרשימה המקושרת של הקבוצות הכשרות, ועבור כל קבוצה כשרה, ניצור Node חדש אשר יכיל את ID הקבוצה, ואת מספר הנקודות של אותה קבוצה (שמחושב לפי הנוסחה בפונקציה play_match).
לאחר יצירת הרשימה של הקבוצות שמשחקות, נשלח את הרשימה לפונקציית עזר playGames.
הפונק׳ עוברת על הרשימה של הקבוצות ועבור כל שתי קבוצות עוקבות, משווה את הניקוד. הקבוצה המנצחת נשארת ברשימה ומתווספות לנקודות שלה נקודות הקבוצה השנייה ועוד 3, ואילו הקבוצה שהפסידה נמחקת. התהליך חוזר על עצמו עד שנשארת קבוצה אחת ברשימה, והיא הזוכה – ונותר רק להחזיר את ה-ID שלה.
נשים לב שיצירת הרשימה המקושרת של הקבוצות שמשחקות קורה בסיבוכיות זמן של O(r) מאחר ועוברים על כל קבוצה פעם אחת בדיוק, כאשר r הוא מס׳ הקבוצות המשחקות.
פונקציית העזר playGames גם היא רצה בסיבוכיות זמן של O(r) – הרשימה מורכבת מ-r קבוצות. אנו רצים בפעם הראשונה על כל הקבוצות ומבין הזוגות מעיפים קבוצות שהפסידו, כלומר לאחר התהליך הזה, נותרנו עם $\frac{r}{2}$ קבוצות. בפעם השנייה אנו מבצעים את אותו הדבר אך על $\frac{r}{2}$ קבוצות ולאחר הפסד של קבוצה אחת בתוך כל זוג קבוצות, אנו נותרים עם $\frac{r}{4}$ קבוצות. התהליך חוזר על עצמו עד שנותרת קבוצה יחידה. אם נסתכל על כמות הפעולות הכוללת, נראה שסה״כ מתבצעות $r + \frac{r}{2} + \frac{r}{4} + \cdots$ פעולות, אשר חסומות ע״י 2r פעולות, כלומר סה״כ סיבוכיות הזמן היא O(r).
נקבל כי סה״כ סיבוכיות הזמן של הפעולה היא O(log(min(n,k)) + r).

בנוסף לכל הכתוב לעיל, בכל פונקציה תהיה בדיקה של הקלט ובדיקה במהלך הריצה אם התכנית נכשלת ותחזיר הודעה בהתאם לפי דרישות התרגיל.

לגבי סיבוכיות המקום של מבנה הנתונים – יש k קבוצות וn שחקנים, כאשר כל שחקן נמצא בדיוק בקבוצה אחת.
עץ הקבוצות יהיה בגודל k, עץ הקבוצות הכשרות יהיה לכל היותר בגודל k, ועצי השחקנים לפי id והשחקנים לפי stats יהיו בגודל n.
מלבד זאת, בכל קבוצה גם יש שני עצים המכילים שחקנים. מכיוון שכל שחקן נמצא בדיוק בקבוצה אחת, סכום גדלי עצי השחקנים לפי id בכל הקבוצות וסכום גדלי עצי השחקנים לפי stats בכל הקבוצות יהיה n לכל אחד מהסכומים.

סך הכול נקבל כי יש במבנה הנתונים לכל היותר $2k + 4n$ צמתים של עצים ולכן סיבוכים המקום של מבנה הנתונים תהיה $O(n + k)$.