מבני נתונים - רטוב 1: חלק יבש

מגישים:

עידו טאוסי 214008997

אפק נחום 214392706

מבנה הנתונים:

- **מחלקת world_cup_t:** מחלקת האב של התרגיל, מרכזת את כל הפעולות הראשיות בממשק המבנים הנדרשים אליה.

  המחלקה תכיל:

  – **teamsById:** עץ AVL המכיל קבוצות הממויינות לפי ה-id של הקבוצה.

  – **playersById:** עץ AVL המכיל שחקנים הממויינים לפי ה-id של השחקן.

  – **playersByScore:** עץ AVL המכיל שחקנים הממויינים לפי מספר הגולים, יש שוויון בין שחקנים במספר הגולים אז משווים את מספר הכרטיסים, ואם גם מספר הכרטיסים שווה אז משווים את ה-id של השחקנים.

  – **validTeams:** עץ AVL המכיל קבוצות שמכילות לפחות 11 שחקנים ולפחות שחקן אחד שיכול לשחק כשוער,
  שממויינים לפי ה-id של הקבוצה.

  – **playersNum:** מספר השחקנים הכולל בטורניר.

  – **topScorer:** מצביע לצומת בעץ שבו נמצא השחקן שהבקיע הכי הרבה גולים.

- **מחלקת Team:** מחלקה המייצגת קבוצת כדורגל, המחלקה מכילה את התכונות:

  – **teamID:** המספר הייחודי של הקבוצה.

  – **totalPoints:** מספר נקודות.

  – **totalCards:** מספר הכרטיסים.

  – **totalGoals:** מספר גולים (משותף לכל השחקנים בקבוצה).

  – **teamPlayersNum:** מספר השחקנים.

  – **gamesPlayedTeam:** מספר המשחקים ששיחקה
  הקבוצה.

  – **goalkeepersNum:** מספר השוערים.

  – **playersById:** עץ AVL המכיל את שחקני הקבוצה ממוינים לפי id.

  – **playersByScore:** עץ AVL המכיל את שחקני הקבוצה הממויינים לפי מספר הגולים, אם יש שוויון בין שחקנים במספר הגולים אז משווים את מספר הכרטיסים, ואם גם מספר הכרטיסים שווה אז משווים את ה-id של השחקנים.

  – **teamTopScorer:** מצביע לשחקן בקבוצה שהבקיע הכי הרבה
  גולים.

  – **previousValidTeam:** מצביע לקבוצה הקודמת בסדר ה-ID שיש לה לפחות 11 שחקנים ושוער.

  – **nextValidTeam:** מצביע לקבוצה הבאה בסדר ה-ID שיש לה לפחות 11 שחקנים ושוער.

- **closestLeft:** מצביע לשחקן אחר הקרוב ביותר לשחקן הנוכחי לפי קריטריוני העץ playerByScore שמופיע לפניו בעץ בחיפוש inorder.
- **closestRight:** מצביע לשחקן אחר הקרוב ביותר לשחקן הנוכחי לפי קריטריוני העץ playerByScore שמופיע אחריו בעץ בחיפוש inorder.
- **team:** קבוצת הכדורגל של השחקן.
- **gamesPlayedPlayer:** מספר המשחקים ששיחק השחקן.
- **numOfGoals:** מספר הגולים שהבקיע השחקן.
- **gamesOnJoin:** כמות המשחקים שהקבוצה שיחקה בעת הצטרפות השחקן.
- **numOfCards:** כמות הכרטיסים שהשחקן קיבל.
- **isGoalkeeper:** מחזיק True אם השחקן יכול לשחקן כשוער, ו-False אחרת.

- **<u>מחלקות Compare:</u>** כוללות את: ComparePlayerByID, ComparePlayerByScore, CompareTeamByID, CompareValidTeamByID, CompareTeamPlayerByScore המחלקות הנ"ל כולן מכילות פונקציות השוואה (קטן ושווה בין שני עצמים), בנוסף חלקן מכילות מימושים לפונקציות חיבור וניתוק של שחקנים\ קבוצות הנמצאים במצביעים המתאימים לשחקן\ קבוצה כלשהי.

<u>פונקציות המערכת:</u>

(הערה: כל העצים שמוזכרים בפונקציות הנ"ל הם עצי AVL ולכן סיבוכיות זמן של חיפוש\ הוספת\ מחיקת איבר מהם היא O(log(x)) כאשר x הוא גודל העץ המדובר)

- **world_cup_t():** מאתחלת את מבנה הנתונים של world_cup_t כאשר כל העצים שבו יהיו ריקים, ולכן זו סיבוכיות מקום וזמן $O(1)$.
- **virtual ~world_cup_t():** משחררת את כל מבני הנתונים שהוגדרו במערכת, יש לנו 2 עצים של מצביעים לכל השחקנים עם n צמתים בכל אחד, עוד עץ של מצביעים לקבוצות שמכיל k צמתים, בכל אחד מהצמתים של עץ כל הקבוצות, יהיו לנו 2 עצים של שחקני הקבוצה (אחד לפי דירוג ואחד לפי ID), לכן בסך הכול יש עוד 2n צמתים. לבסוף יש לנו עוד עץ של קבוצות שמכילות לפחות 11 שחקנים, במקרה הגרוע ביותר כל הקבוצות הן בעלות 11 שחקנים ולכן עץ זה יהיה בעל k צמתים במקרה הגרוע. לכן בסך הכול יהיו לנו במקרה הגרוע ביותר 2n+k+2n+k איברים לשחרר ובנוסף לכך יהיו עוד n+k מצביעים של שחקנים וקבוצות שישוחררו מהזיכרון כאשר לא כל ה-shared_ptr ייהרסו, לכן 5n+3k פעולות כי ניתן לעבור על עץ בעל n צמתים ב-n פעולות ולשחרר אותו מהזיכרון, ולכן סיבוכיות הזמן היא $O(n+k)$.
- **add_team(int teamId, int points):** המערכת מכניסה קבוצה לתוך עץ כל הקבוצות שלה, זהו עץ AVL ולכן ייקח למערכת $O(log(k))$ פעולות כדי כדי לחפש האם הקבוצה קיימת ולהכניס את הקבוצה למקום המתאים שלה, לכן סיבוכיות הזמן היא $O(log(k))$ וגם סיבוכיות המקום היא $O(log(k))$ מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **remove_team(int teamId):** אם יש שחקנים בקבוצה אז נחזיר Failure, אחרת אין בה שחקנים, ולכן ניגש לעץ ה-AVL שמכיל את כל הקבוצות וממוין לפי teamID ונחפש בעץ את הקבוצה, זה נעשה בסיבוכיות $O(log(k))$ ע"פ הנלמד בתרגול. בנוסף נסיר אותה באותו אופן מעץ הקבוצות validTeams, שמכיל במקרה הגרוע k קבוצות ולכן גם הסרה זו נעשית בסיבוכיות של $O(log(k))$. סיבוכיות המקום היא $O(log(k))$ כיוון שנעשה תהליך ריקורסיבי, והוא תופס מקום בהתאם למספר האיטרציות המקסימלי, שהוא כעומק העץ במקרה הגרוע ולכן היא $O(log(k))$.
- **add_player(int playerId,int teamId,int gamesPlayed,int goals,int cards,bool goalKeeper):** ראשית נחפש את הקבוצה בעץ הקבוצות הממוין לפי teamID, זה נעשה בסיבוכיות $O(log(k))$ ע"פ הנלמד בתרגול. כעת ניצור שחקן חדש עם המידע על הקבוצה ועם הפרמטרים שקיבלנו בסיבוכיות $O(1)$, אם יש פרמטר לא תקין\

חוסר במקום בזכרון נסיים ונזרוק שגיאה בהתאם, אחרת נשים את השחקן בעצי השחקנים במערכת. הם עצים ממוינים (לפי המתואר לעיל) ולכן פעולת ההכנסה תעלה בסיבוכיות ($O(log(n)$), ועל כן סיבוכיות הזמן הכוללת היא ($O(log(n) + log(k)$), כיוון שנעשה תהליך ריקורסיבי, והוא תופס מקום בהתאם למספר האיטרציות המקסימלי, שהוא כעומק עץ הקבוצות\ השחקנים במקרה הגרוע ולכן היא ($O(max\{log(k), log(n)\}$).

- **remove_player(int playerId):** נוציא את השחקן מהמערכת, ראשית נחפש אותו בעץ כל השחקנים לפי ID, כעת התוכנית תיקח את הנתונים שלו, תמחק את הצומת, ותחפש אותו בעץ כל השחקנים לפי דירוג ותמחק את הצומת שהוא נמצא בה ולאחר מכן תעדכן את מצביעי ה- closest של השכנים שלו מימין ומשמאל כך שתחבר ביניהם, עכשיו נעבור בעזרת הפוינטר שיש לשחקן אל הקבוצה שלו, ונחפש אותו בעץ שחקני הקבוצה לפי ID, ועץ שחקני הקבוצה לפי דירוג ונמחק את המופעים, גודל עץ שחקני הקבוצה קטן או שווה במקרה הגרוע לעץ כל השחקנים, ולכן סיבוכיות הזמן שלנו היא ($O(log(n)$) במקרה הגרוע מפני שבמקרה הכי גרוע כל החיפושים שלנו לקחו 4log(n) פעולות.
סיבוכיות המקום היא ($O(log(n)$) מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **update_player_stats(int playerId, int gamesPlayed, int scoredGoals, int cardsReceived):** על מנת לעדכן את נתוני השחקן, נחפש אותו בעץ כל השחקנים לפי ID, כעת משהשגנו את הנתונים שלו, נוכל למצוא אותו בעץ כל השחקנים לפי דירוג, ולעבור לקבוצה שלו ולחפש אותו בעץ שחקני הקבוצה לפי דירוג. כעת ניתן לעדכן את הנתונים, להוציא את השחקן מעץ כל השחקנים ועץ שחקני הקבוצה לפי דירוג, לעדכן אותו, ולהכניס מחדש לשני העצים הללו, במקרה הכי גרוע עשינו 5 חיפושים על העץ ולכן 5log(n) פעולות, לכן סיבוכיות הזמן היא ($O(log(n)$).
סיבוכיות המקום היא ($O(log(n)$) מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **play_match(int teamId1,int teamId2):** ניגש לעץ הקבוצות שיכולות להשתתף במשחקים הממוין לפי teamID, ונמצא את 2 הקבוצות הנ"ל, בסיבוכיות ($O(log(k)$) לפי הנלמד בתרגול. כעת, נשמתמש בנוסחה הנתונה על מנת לחשב איזו קבוצה ניצחה (סיבוכיות $O(1)$) ונעדכן את השדות הרלוונטיים בקבוצות (סיבוכיות $O(1)$), לכן בסה"כ סיבוכיות הזמן היא ($O(log(k)$). כיוון שנעשה תהליך ריקורסיבי, והוא תופס מקום בהתאם למספר האיטרציות המקסימלי, שהוא כעומק העץ במקרה הגרוע והוא log(k), אז סיבוכיות המקום היא ($O(log(k)$).
- **get_num_played_games(int playerId):** נמצא את השחקן בעץ השחקנים המסודר לפי ID (בסיבוכיות ($O(log(n)$, ע"פ הנלמד בתרגול) ונחשב את מספר המשחקים בהם שיחק בהתאם לפרמטרים שלו ושל קבוצתו.
סיבוכיות המקום היא ($O(log(n)$) מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **get_team_points(int teamId):** נמצא את הקבוצה הרצויה בעץ כל הקבוצות, ונחזיר את הנקודות שלה. חיפוש הקבוצה בתוך העץ ייקח במקרה הגרוע log(k) פעולות ולכן סיבוכיות הזמן היא ($O(log(k)$). סיבוכיות המקום היא ($O(log(k)$) מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **unite_teams(int teamId1,int teamId2,int newTeamId):** ניגש לעץ הקבוצות הממוין לפי teamID, ונמצא את 2 הקבוצות הנ"ל, בסיבוכיות ($O(log(k)$) לפי הנלמד בתרגול. כעת ניצור קבוצה חדשה מ-2 הקבוצות הנ"ל, תכונות הקבוצה יעודכנו ע"פ שתי הקבוצות שמאחדים, ועץ השחקנים של הקבוצה יבנה ע"י אלגוריתם איחוד העצים שנלמד בתרגול (בסיבוכיות ($O(n_{TeamID1} + n_{TeamID2}$) וזאת מכיוון שכמות הצמתים בעץ שלם הכי קטן שיכיל לפחות
$n_{TeamID1} + n_{TeamID2}$ צמתים ויהיה בגובה h הוא $2^{h+1} - 1$ ולכן על פי הנוסחה הידועה לחישוב כמות צמתים בעץ בהינתן גובהו, מתקיים ש- $2^{(h-1)+1} - 1 \leqslant n_{TeamID1} + n_{TeamID2} \leqslant 2^{h+1} - 1$ כעת נכפיל ב-2 את אי השיוויון ונקבל $2^{h+1} - 2 \leqslant 2(n_{TeamID1} + n_{TeamID2})$ כעת נוסיף לשני האגפים 1 ומתקיים כי:
$2^{h+1} - 1 \leqslant 2(n_{TeamID1} + n_{TeamID2}) + 1$ ולכן
$n_{TeamID1} + n_{TeamID2} \leqslant 2^{h+1} - 1 \leqslant 2(n_{TeamID1} + n_{TeamID2}) + 1$

זה מספר הצמתים בעץ השלם שעוברים עליהם בפעולה ומבצעים עליהם פעולות בסיבוכיות $O(1)$, לכן סיבוכיות הזמן של בניית עץ שלם בגובה h היא $\Theta(n_{TeamID1} + n_{TeamID2})$, ולכן סיבוכיות הזמן המתקבלת היא אכן ($O(log(k) + n_{TeamID1} + n_{TeamID2}$). סיבוכיות המקום היא ($O(max\{log(k), n_{TeamID1} + n_{TeamID2}\}$ כיוון שתקבע ע"פ המקסימלי מבין: עומק עץ הקבוצות ($log(k)$) (כיוון שקובע את עומק הריקורסיה בחיפוש הקבוצה) וגודל הקבוצה החדשה (size of new team= $n_{Team1ID} + n_{Team2ID}$) כיוון שיוקצע לה מקום בזכרון.
- **get_top_scorer(int teamId):** אם teamId<0 אז ניגש למצביע של topScorer ומשם נשיג את ה-ID של מלך

השערים ב-$O(1)$ פעולות. אם teamId>0, נחפש את הקבוצה בעץ כל הקבוצות שזוהי סיבוכיות זמן $O(log(k))$ ואם הקבוצה אכן קיימת אז ניגש למצביע של מלך השערים בקבוצה, ונשיג משם את ה-ID שלו במספר פעולות $O(1)$. לכן, סיבוכיות הזמן שלנו היא $O(log(k))$ במקרה הגרוע. סיבוכיות המקום היא $O(log(k))$ מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.

- **get_all_players_count(int teamId):** אם teamId<0, נחזיר את מספר השחקנים הכולל ששמור במערכת ב-$O(1)$. אם teamId>0, נחפש את את הקבוצה בעץ כל הקבוצות, זה ייקח במקרה הגרוע $O(log(k))$ ואז נחזיר את מספר השחקנים של הקבוצה ששמור בה ב-$O(1)$. סיבוכיות המקום היא $O(log(k))$ מפני שיש חיפוש רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **get_all_players(int teamId, int * const output):** אם teamId<0, נבצע סיור inorder לעץ כל השחקנים לפי דירוג, וכך נקבל את השחקנים לפי הסדר הנדרש במערך, בסיבוכיות זמן $O(n)$. אם teamId>0, נחפש את את הקבוצה בעץ כל הקבוצות, ואז נבצע סיור inorder לעץ שחקני לפי דירוג, וכך נקבל את השחקנים של הקבוצה לפי הסדר הנדרש במערך בסיבוכיות זמן $O(log(k) + n_{TeamID})$. סיבוכיות המקום היא לכל היותר $O(log(n))$ מפני שיש סיור inorder רקורסיבי שמגיע במקרה הגרוע לתחתית העץ.
- **get_closest_player(int playerId, int teamId):** נחפש את הקבוצה בתוך עץ כל הקבוצות בסיבוכיות $O(log(k))$, לאחר מכן נחפש את השחקן בעץ שחקני הקבוצה לפי ID בסיבוכיות $O(log(n_{TeamID}))$, כעת לאחר שמצאנו את השחקן, נבדוק מי יותר קרוב אליו, השחקן הקרוב ביותר מלמטה closestLeft, או השחקן הקרוב ביותר מלמעלה closestRight. ההשוואה היא $O(1)$ פעולות ולאחריה נחזיר את מזהה השחקן הקרוב ביותר. לכן, סיבוכיות הזמן היא $O(log(k) + log(n_{TeamID}))$. סיבוכיות המקום היא $O(log(k) + log(n_{TeamID}))$ מפני שאנו מבצעים חיפוש רקורסיבי על עצי AVL שמגיע במקרה הגרוע לתחתית העצים שאנו רצים עליהם.
- **knockout_winner(int minTeamId, int maxTeamId):** נחפש קבוצה כלשהי בטווח הנדרש בעץ שמכיל קבוצות בעלות לפחות 11 שחקנים ושוער, נניח כי יש בעץ זה m קבוצות, לכן החיפוש ייקח $O(log(m))$. מתקיים בנוסף כי $m \leqslant k$ מפני ש-k כבר מכיל בתוכו את m הקבוצות. בנוסף, אם יש לנו m קבוצות שמכילות לפחות 11 שחקנים ושוער, מתקיים בהכרח שמספר השחקנים הכולל בטורניר הוא לפחות 11m, ולכן $11m \leqslant n$. לכן סיבוכיות החיפוש בעץ הקבוצות בעלות לפחות 11 שחקנים הוא $O(log(min\{n,k\}))$. כעת לאחר שמצאנו קבוצה כלשהי בטווח נרוץ על המצביעים nextPlayingTeam של כל הקבוצות עד שנגיע לקבוצה עם ה-ID המינימלי הנתון ומשם נרוץ עד למקסימלי, וניצור מערך מהן שזה לכל היותר בסיבוכיות $O(r)$. כעת נרוץ על המערך, ונעשה סימולציות למשחקים, ובסוף הריצה נקצה מערך בגודל חצי מהמערך הקודם לו בתחילת הריצה (אותו נשמור כמערך עזר שגודלו חצי מהקודם לו ואז נשחרר, לכן לא משפיע על סיבוכיות המקום), נמשיך כך עד שיישאר איבר אחד במערך, לכן מספר האיטרציות הוא סכום סדרה הנדסית, שחסום על ידי סכום סדרה הנדסית אינסופית ולכן:

$r + \frac{r}{2} + \frac{r}{4} + \frac{r}{8} + ... + 1 \leqslant \frac{r}{1-\frac{1}{2}} = 2r$, חסום על ידי $O(r)$. מתוך כך נובע כי סיבוכיות הזמן היא $O(log(min\{n,k\} + r)$. סיבוכיות המקום במקרה הגרוע ביותר שחיפשנו בעץ בגודל m, שקטן מ-k וגם מ-n, ואז יצרנו מערך בגודל r שבמקרה הגרוע ביותר שווה ל-k, לכן סיבוכיות המקום היא $O(log(min\{n,k\}) + k)$.