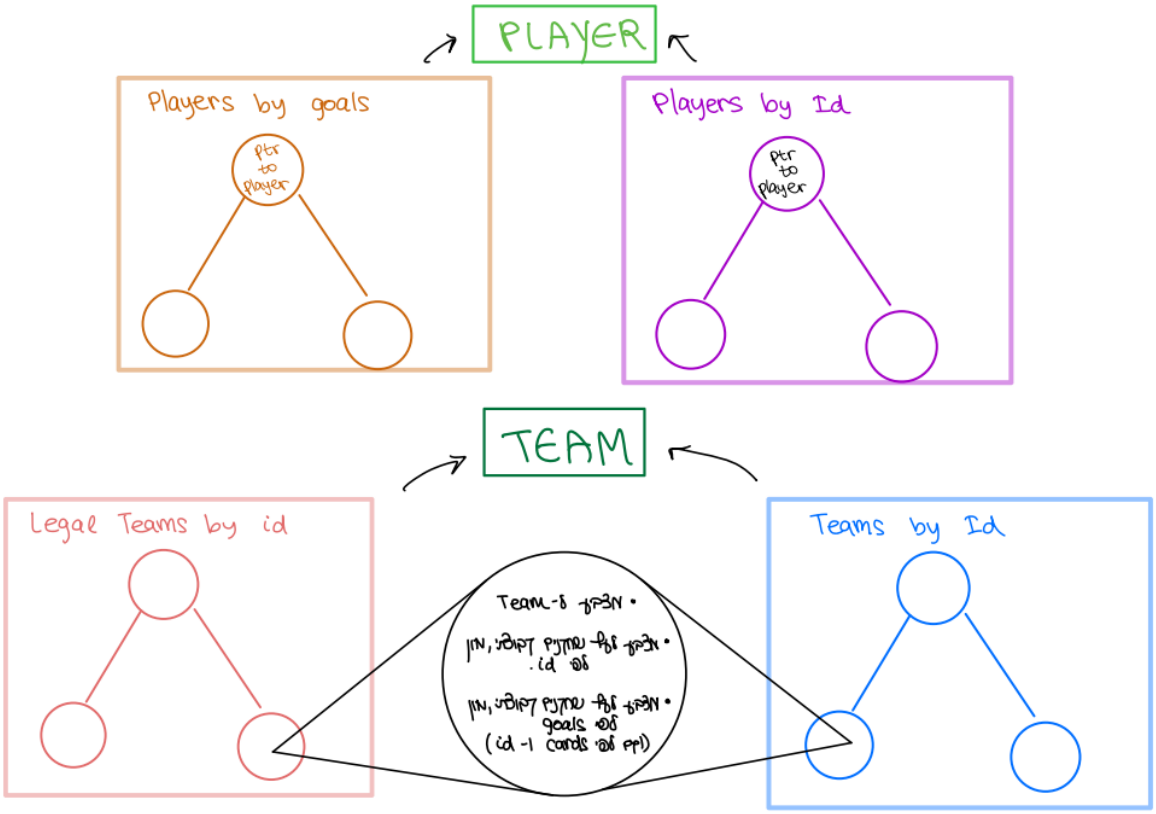
מימוש מבנה הנתונים



הסבר כללי על המבנה:

- כל צומת בעץ הקבוצות יכיל מצביע למשתנה מסוג team שיכיל את נתוני הקבוצה, ו2 מצביעים לעצים: עץ שחקנים קבוצתי הממוין לפי id ועץ שחקנים קבוצתי הממוין לפי goals, cards, id (כמו שנדרשנו בחלק מהפונקציות).
- כל צומת בעצי השחקנים יכיל מצביע למשתנה מסוג player שיכיל את נתוני השחקן.

ראשית מימשנו את העץ באופן הבא:

מחלקת TreeNode: מחלקה זו היא מסוג template, לכן מימשנו את הצמתים עבור עצי השחקנים והקבוצות באמצעותה (כלומר סוג העץ הוגדר בתור הtemplate). כל צומת כזו תכיל גם מפתחות מיון, לפיהם נבצע את המיון של כל אחד מהעצים.

- כל פונקציות set,get הן כמובן בסיבוכיות O(1).
- העמסת אופרטורים שביצענו והשתמשנו בהן בפונקציות במבנה הנתונים – סיבוכיות O(1) כי רק מבצעות השוואות בין מפתחות המיון.
- פונקציות עזר נוספות – גם הן ניגשות לשדות שבתוך הצומת, באמצעות גישה ישירה או באמצעות פונקציות get,set ולכן גם הן בסיבוכיות O(1).

מחלקת Tree: מחלקה זו היא בעצם העץ עצמו, מכילה מצביע לשורש (ואת מספר הצמתים לטובת שימושים עתידיים). נתאר באופן כללי את המימוש:

- כל פונקציות set,get הן כמובן בסיבוכיות O(1).
- findNode – קיימות 2 מתודות כאלה, השוני ביניהן הוא בקלטים: פונקציה אחת תקבל node אותו היא צריכה למצוא, והשניה מקבלת key לפיו היא מוצאת את הצומת הרצויה. הפונקציה מבצעת מעבר על העץ כפי שלמדנו בהרצאה, לכן מתבצעת בסיבוכיות log(k), כאשר k הוא מספר הצמתים בעץ.
- insertNode – המתודה מקבלת node, אותו היא תכניס לעץ. ראשית נבדוק האם קיים שורש לעץ. אם לא, הצומת שקיבלנו תוגדר כשורש, זאת בסיבוכיות O(1). במידה וקיים שורש, נבצע מעבר על העץ בדומה לחיפוש בינארי כפי שלמדנו בהרצאה, לכן מציאת המקום המתאים לצומת היא בסיבוכיות log(k), כאשר kהוא מספר הצמתים בעץ. לאחר מכן נקרא לפונקציה fixTreeBalance (עליה נפרט בנפרד), אך הסיבוכיות שלה היא O(1) במקרה של הוספת צומת. בנוסף נעדכן את מספר הצמתים בעץ, סיבוכיות O(1).לכן סה"כ הסיבוכיות של insert היא O(logk).
- fixTreeBalance – הפונקציה מקבלת node אותו הוספנו (או אחד שנרצה להסיר) ועוברת על האבות שלו, עד שהיא מוצאת את האב הראשון שבו הופר הbalance factor. בהתאם לערכו, כפי שלמדנו בהרצאות, הפונקציה תקרא לאחד

מ4 סוגי הגלגולים האפשריים, עליהם נפרט בהמשך.
במידה ומדובר בצומת שהוספנו, נבצע גלגול אחד בלבד (כפי שלמדנו בהרצאה). במידה ומדובר בהסרה של צומת, נעלה עד השורש בביצוע הגלגולים (כי במקרה זה, אין לדעת כמה גלגולים נצטרך לבצע).
הסיבוכיות של פונקציה זו תהיה O(logk), כאשר k הוא מספר הצמתים בעץ.

- removeNode – הפונקציה מקבלת צומת שאותה יש להסיר מהעץ. ראשית הפונקציה תחפש את המקום של הצומת, בסיבוכיות O(logk) כפי שלמדנו, כדי לוודא שהוא אכן קיים בעץ. הפונקציה תגדיר מחדש את המצביעים של האבא והבנים של הצומת שנרצה להסיר, זאת בסיבוכיות O(1). במקרה שבו מדובר בצומת שהוא לא שורש ולא עלה, הפונקציה תרצה למצוא את האיבר העוקב של הצומת שנרצה להסיר, על מנת לשים אותו במקום האחד שאני מסירים, זאת בסיבוכיות של O(logk) במקרה הגרוע. לאחר מכן הפונקציה תמחק מהעץ את הצומת, סיבוכיות O(1).
כעת נקרא לfixTreeBalance, שכפי שאמרנו במקרה זה בסיבוכיות O(logk). לבסוף תעדכן את מספר הצמתים בעץ (כלומר תוריד אחד), בסיבוכיות O(1). לכן סה"כ הסיבוכיות של פונקציה זו היא O(logk).
- rotateRR, rotateLL– פונקציות אלו מבצעות גלגול יחיד כפי שלמדנו, זאת בסיבוכיות O(1), ואף מעדכנות את הגובה ואת הbf של הצמתים המתאימים, מבצעות זאת בסיבוכיות של O(logk) במקרה הגרוע. כלומר סהכ הסיבוכיות של כל גלגול כזה היא O(logk).
- rotateRL,rotateRR – משתמשים בשני הגלגולים המתוארים לעיל, לכן סה"כ יהיו גם כן בסיבוכיות של O(logk).
- Unite_trees – פונקציה זו מקבלת 2 עצים ומאחדת אותם לעץ יחיד. היא עושה זאת ע"י יצירת 2 מערכי inorder, איחודם בעזרת mergesort (בסיבוכיות o(n), כאשר n הוא סך השחקנים ב2 הקבוצות), ואז הכנסת השחקנים לעץ בצורה ממוינת בעזרת סוג של חיפוש בינארי, אך בגלל שעוברת על כל המערך המאוחד, גם הכנסת הערכים לעץ היא בסיבוכיות O(n), עבור n המאוחד. בסיום האיחוד נמחק את כל הצמתים מהעצים אותם איחדנו, ואז נמחק את העצים עצמם.

פירוט הפונקציות בworldcup23a1

פונקציית world_cup_t() :

מימוש הפונקציה: ניצור ונאתחל בעזרת constructors 4 עצים :

- עץ של כל הקבוצות במערכת (מחלקה מסוג TeamsTree) – ממוין לפי teamId.
- עץ של כל הקבוצות החוקיות במערכת, כלומר כל הקבוצות המכילות לפחות 11 שחקנים כשביניהם קיים שחקן שהוא גם שוער (מחלקה מסוג TeamsTree) – ממוין לפי teamId.
- עץ של כל השחקנים במערכת (מחלקה מסוג PlayersTree) – ממוין לפי playerId.
- עץ של כל השחקנים במערכת (מחלקה מסוג PlayersTree) – ממוין לפי goals.

נכונות הסיבוכיות: קריאה לconstructor של עץ AVL היא בסיבוכיות O(1),אתחול עץ AVL הוא בסיבוכיות O(1). אנו מאתחלים סהכ 4 עצי AVL, לכן סה"כ הסיבוכיות הכוללת של פונקציה זו תהיה **O(1).**

פונקציית virtual ~world_cup_t() :

מימוש הפונקציה: נבצע שחרור לכל אחד מחמשת העצים שהקצאנו בבנאי של worldcup.

נכונות הסיבוכיות: שחרור כל אחד מהעצים דורש מעבר על העץ ווידוא שחרור של כל אחד מהאיברים בו.מעבר על 2 עצי הקבוצות – כל מעבר כזה הוא מעבר על מקסימום k איברים (במקרה של הקבוצות החוקיות, יכול להיות גם פחות). כלומר הסיבוכיות של שחרור כל עץ תהיה O(k). עבור שלושת העצים, לפי תכונות סיבוכיות, תהיה הסיבוכיות סה"כ גם כן O(k).מעבר על 2 עצי השחקנים – כל מעבר כזה הוא מעבר על מקסימום n איברים. כלומר הסיבוכיות של שחרור כל עץ שחקנים תהיה O(n). עבור שני עצי השחקנים, לפי תכונות סיבוכיות, תהיה הסיבוכיות סה"כ O(n).

לכן בסופו של דבר, הסיבוכיות של פונקציה זו תהיה : $O(n) + O(k) = \boldsymbol{O(n+k)}$

StatusType add_team(int teamId, int points)

מימוש הפונקציה:

- נוודא כי הקלטים שקיבלנו חוקיים.
- יצירת קבוצה חדשה בעזרת constructor של מחלקת Team ובדיקה שהקצאת המקום עברה בהצלחה.
- אתחול ערכי הקבוצה – teamId, points, לפי הערכים שניתנו בקלט לפונקציה.
- יצירת node מסוג Team עבור עץ AVL בעזרת קריאה לconstructor של מחלקת Node:TreeNode אשר מוגדר לו מפתח מיון מסוג teamId, אותו נכניס לעץ הקבוצות הממוין לפי teamId

- עבור כל אחד מהnodeים, נוודא כי ההקצאה שלו עברה בהצלחה. אם לא – נשחרר את מה שכבר הקצנו בפונקציה הזו (את הקבוצה החדשה שיצרנו עבור שניהם, ובהקצאה של הnode השני נשחרר גם את ה node הראשון במקרה הצורך).
- כעת נכניס כל אחד מהnode שיצרנו לעץ המתאים לו, כפי שתיארנו לעיל. במידה והקבוצה כבר הייתה קיימת באחד מן העצים, נחזיר FAILURE.

נכונות המימוש: אנו יוצרים קבוצה חדשה לפי הקלטים הניתנים בפונקציה, ומכניסים אותה למבנה הנתונים שלנו .

נכונות הסיבוכיות: יצירת קבוצה חדשה היא בסיבוכיות של O(1). אתחול כל אחד מערכי הקבוצה יהיה בסיבוכיות O(1). יצירת node חדש עבור עץ AVL הוא בסיבוכיות של O(1). אנו יוצרים 2 כאלה ולכן סהכ הסיבוכיות תהיה גם כן O(1). שחרור ההקצאות במידת הצורך (אם הייתה תקלה באחת ההקצאות) תהיה גם היא בסיבוכיות O(1). הכנסת הצומת תהיה בסיבוכיות של מקסימום O(logk), כאשר k הוא מספר הקבוצות המקסימלי שיכול להיות קיים בעץ(ניקח בחשבון שבמהלך ההכנסה יש למצוא את המקום המתאים עבור הצומת, וביצוע גלגולים במידת הצורך). עבור שני העצים הסיבוכיות תהיה גם כן סה"כ : $O(logk) + O(logk) \leq O(logk)$.
לבסוף, הסיבוכיות של הפונקציה כולה תהיה $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

StatusType remove_team(int teamId)

מימוש הפונקציה:

- נוודא שהקלט חוקי.
- נמצא את הnode המכיל את הקבוצה בעזרת פונקציית findNode של העץ שכתבנו, כפי שתארנו היא בסיבוכיות O(logk), כאשר k הוא מספר הקבוצות. במידה והקבוצה לא קיימת בעץ, נחזיר FAILURE.
- בעץ השחקנים השייכים לקבוצה, נבדוק שלא קיימים בה שחקנים (נעשה זאת באמצעות בדיקה האם השורש של העץ לא קיים). אם קיימים – נחזיר שגיאה ולא נמשיך את פעולת ההסרה של הקבוצה.
- לאחר מכן נשחרר את הNode של אותה הקבוצה מעץ הקבוצות הכולל.

נכונות המימוש: במידה ולא קיימים שחקנים בקבוצה, הקבוצה תוסר לחלוטין ממבנה הנתונים.

נכונות הסיבוכיות: החיפוש בעץ הקבוצות הכולל (על מנת למצוא את הקבוצה המבוקשת) הוא בסיבוכיות O(logk), מפני שזוהי הסיבוכיות של פונקציית החיפוש של העץ. בדיקה של השתייכות שחקנים לקבוצה היא בסיבוכיות O(1), מפני שגישה לשורש ובדיקת הערך שלו היא בסיבוכיות O(1). לכן סה"כ הסיבוכיות הכוללת של פונק' זו היא $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

StatusType add_player(int playerId, int teamId, int gamesPlayed, int goals, int cards, bool goalKeeper)

מימוש הפונקציה:

- נוודא שהקלטים שקיבלנו חוקיים.
- נחפש בעזרת פונקציית find את הקבוצה המיועדת של השחקן. אם היא לא קיימת, נעצור את הפונקציה ונחזיר שגיאה.
- ניצור שחקן חדש עם כל הנתונים שקיבלנו בקלט בעזרת הבנאי של מחלקת Player. נוודא כי הקצאת המקום עברה בהצלחה. אם לא – נחזיר שגיאה.
- ניצור 2 nodes עבור השחקן:
  - צומת שמפתח המיון שלה הוא playerId, אותה נוסיף לעץ השחקנים הכולל הממוין לפי Id וגם לעץ השחקנים הקבוצתי הממוין לפי Id.
  - צומת שמפתחות המיון שלה הם goals, לאחר מכן cards ולאחר מכן playerId. אותה נוסיף לעץ השחקנים הכולל הממוין לפי goals וגם לעץ השחקנים הקבוצתי הממוין לפי goals.
- נכניס את הצמתים שיצרנו לעצים המתאימים המתוארים כאן לעיל.
- נעדכן עבור כל אחת מהקבוצות את מספר הגולים הכולל של הקבוצה ומספר הכרטיסים הכולל של הקבוצה (זאת עבור חישובים כמו ב-play_match).
- נבדוק האם השחקן שהוספנו הוא top scorer בקבוצה שלו או בעץ השחקנים הכולל. אם כן, נעדכן זאת.
- נבדוק האם השחקן שהוספנו יכול להיות שוער, אם כן, נעדכן זאת עבור הקבוצה אליה הוא שייך.
- נבדוק האם הכנסת השחקן הפכה את הקבוצה לקבוצה חוקית (11 שחקנים ושוער). אם כן, נוסיף את הקבוצה כעת לעץ הקבוצות החוקיות.

נכונות המימוש: בסיום הפונקצייה השחקן יהיה קיים במבנה הנתונים וישתייך לקבוצה המתאימה לו.

נכונות הסיבוכיות: חיפוש הקבוצה של השחקן היא בסיבוכיות O(logk) כפי שתיארתי בפונקציות קודמות. יצירת שחקן חדש בעזרת הבנאי היא בסיבוכיות O(1). יצירת הnodeים עבור עצי השחקנים היא בסיבוכיות O(1). הכנסת הnodeים לעצי

השחקנים (הכוללים וגם הקבוצתיים) היא בסיבוכיות O(logn) במקרה הגרוע, כי בכל עץ בנפרד יהיו מקסימום n שחקנים.לכן סה"כ נקבל סיבוכיות $O(logn) + O(logk) = \boldsymbol{O(logk + logn)}$.

StatusType remove_player(int playerId)

מימוש הפונקציה:

- נבדוק שהקלט שקיבלנו חוקי.
- נמצא את השחקן שנרצה להסיר (לפי id כמובן) באמצעות פונקציית find. אם השחקן לא קיים במערכת, נחזיר שגיאה.
- ניגש לערך ה-goals שלו, לפיו נמצא את השחקן גם בעץ הכולל הממוין לפי גולים.
- השחקן מחזיק מצביע לקבוצה שלו, ניגש לקבוצה אליה הוא שייך ואף נוודא כי השחקן אכן קיים בעצי השחקנים של קבוצה זו. אם קיים, נסיר אותו מ2 עצי השחקנים הקבוצתיים בעזרת פונקציית remove.
- כעת נסיר את השחקן מכל אחד מהעצים הכוללים (הממוינים לפי ת"ז ולפי גולים).

נכונות המימוש: השחקן יוסר לחלוטין ממבנה הנתונים (בפרט מהקבוצה אליה הוא שייך).

נכונות הסיבוכיות: חיפוש השחקן שנרצה להסיר היא בסיבוכיות O(logn) כפי שתיארנו בפונקציות קודמות. גישה לערך של מספר הגולים שלו היא ב(O(1. גישה לקבוצה אליה השחקן שייך (O(1. שחרור השחקן מכל אחד מהעצים הקבוצתיים של השחקנים, הוא בסיבוכיות של O(logn) במקרה הגרוע, לכן סה"כ עבור 2 עצים אלו הסיבוכיות היא O(logn). שחרור השחקן ב2 עצי השחקנים הכוללים הוא גם בסיבוכיות של O(logn) (שחרור מכל אחד מהעצים הוא O(logn), לכן סה"כ עבור שניהם גם O(logn). סה"כ מימוש הפונקציה יהיה בסיבוכיות $O(logn) + O(1) = \boldsymbol{O(logn)}$.

StatusType update_player_stats(int playerId, int gamesPlayed, int scoredGoals, int cardsReceived)

מימוש הפונקציה:

- נבדוק שהקלטים שקיבלנו חוקיים.
- נמצא את השחקן המתאים לפי ID בעץ השחקנים הכולל, באמצעות פונקציית find. אם הוא לא קיים בעץ, נחזיר שגיאה.
- ניגש לנתוני השחקן שאת הצומת שלו בעץ מצאנו, ונעדכן בעזרת פונקציות set את הנתונים של השחקן.
- אם מספר הגולים שנרצה לעדכן שונה מ-0, ככל הנראה מיקום השחקן בעץ השחקנים הממוין לפי גולים ישתנה, ולכן נרצה לבצע הוצאה של הצומת מעץ זה, ולאחר מכן הכנסה עם הנתונים החדשים (נעשה זאת על מנת לשמר את האיזון בעץ הממוין לפי גולים).
- בנוסף עבור הקבוצה של השחקן, נרצה לעדכן את מספר הגולים הכולל של כל השחקנים שלה, ואת מספר הכרטיסים הכולל של כל שחקני הקבוצה.

נכונות המימוש: הפונקציה מעדכנת את נתוני השחקן בהתאם לקלטים, ומעדכנת את מיקומו בעץ הממוין לפי גולים, זאת בהתאם לשינוי.

נכונות הסיבוכיות: חיפוש השחקן בעץ השחקנים הכולל תהיה במקרה הגרוע O(logn). עדכון הנתונים של השחקן באמצעות setים הוא בסיבוכיות O(1). הוצאה של השחקן (במידת הצורך) תהיה בסיבוכיות O(logn) במקרה הגרוע, כך גם הסיבוכיות של הכנסתו חזרה לעץ. כלומר סה"כ הסיבוכיות של הוצאה ולאחר מכן הכנסה, תהיה O(logn).
לבסוף הסיבוכיות של הפונקציה כולה תהיה $O(logn) + O(1) = \boldsymbol{O(logn)}$.

StatusType play_match(int teamId1, int teamId2)

מימוש הפונקציה:

- נבדוק שכל הקלטים חוקיים.
- נחפש בעץ הקבוצות החוקיות את 2 הקבוצות הללו בעזרת פונקציית find. אם אחת הפונקציות לא קיימת בעץ, נחזיר שגיאה.
- נחשב את הניקוד הכולל של כל קבוצה עפ"י הנוסחא המוצגת בתרגיל. נעשה זאת בעזרת גישה עם פונקציות get לערכים המתאימים השמורים במשתנה של הקבוצה.
- נשווה בין הניקוד של 2 הקבוצות ולפיו נוסיף מספר נקודות מתאים לקבוצה המנצחת, או ל2 הקבוצות במקרה של תיקו.

נכונות המימוש: הפונקציה תחשב מי הקבוצה המנצחת מבין השתיים ותוסיף ניקוד בהתאם לניצחון/תיקו.

נכונות הסיבוכיות: חיפוש 2 הקבוצות בעץ הקבוצות החוקיות יהיה בסיבוכיות של O(logk). כלל החישובים יהיו בסיבוכיות O(1), כי מדובר במשתנים קיימים ואין צורך לעבור על עץ השחקנים לטובת חישובים אלו. גם עדכון הניקוד הקבוצתי של כל אחת מהקבוצות יהיה בסיבוכיות O(1). סה"כ עבור פונקציה זו, הסיבוכיות תהיה $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

output_t < int > get_num_played_games(int playerId)

מימוש הפונקציה:

- נבדוק שהקלט שקיבלנו חוקי.
- נחפש את השחקן בעץ השחקנים הכולל לפי הממוין לפי id בעזרת פונקציית find. אם הוא לא קיים בעץ, נחזיר שגיאה.
- ניגש לקבוצה של השחקן בעזרת מצביע לקבוצה, אותו השחקן מחזיק, וניגש למספר המשחקים שהקבוצה שיחקה עד כה.
- נסכום את מספר המשחקים של הקבוצה ביחד עם הערך של מספר המשחקים האישיים של השחקן (אותו נקבל בעזרת פונקציית get), כך שסה"כ נקבל את המספר הכולל של משחקים שהשחקן שיחק מאז שנכנס למבנה הנתונים.

נכונות המימוש: הפונקציה תחשב את מספר המשחקים בהם השתתף השחקן הרלוונטי.

נכונות הסיבוכיות: חיפוש השחקן באמצעות פונקציית find הוא בסיבוכיות O(logn), גישה למצביע של הקבוצה ושליפת מספר המשחקים הקבוצתיים הוא בסיבוכיות O(1), כך גם הגישה לשדה המשחקים ששוחקו אצל השחקן. לכן הסיבוכיות של סכימת מספר המשחקים הכולל תהיה גם היא O(1). סה"כ מימוש הפונקציה הוא במקרה הגרוע בסיבוכיות $O(1) + O(logn) \leq \boldsymbol{O(logn)}$.

output_t < int > get_team_points(int teamId)

מימוש הפונקציה: בעץ הקבוצות הכולל הממוין לפי teamId נחפש את הקבוצה המתאימה, אם היא לא קיימת בעץ– נחזיר שגיאה. בעזרת פונקציית get שמימשנו בתוך העץ נוציא את הערך הרצוי.

נכונות המימוש: הפונקציה תחזיר את מספר הנקודות של הקבוצה.

נכונות הסיבוכיות: חיפוש הקבוצה בעץ הקבוצות הכולל בעזרת פונקציית find הוא בסיבוכיות של O(logk). פונקציית get היא בסיבוכיות O(1), לכן סהכ הסיבוכיות במקרה הגרוע תהיה $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

StatusType unite_teams(int teamId1, int teamId2, int newTeamId)

מימוש הפונקציה:

- נבדוק שכל הקלטים שקיבלנו חוקיים.
- נמצא את כל אחת מהקבוצות שאותן נרצה לאחד ע"י חיפוש בעץ הקבוצות הכולל הממוין לפי id. אם אחת הקבוצות לא קיימת, נחזיר שגיאה.
- אם המזהה של הקבוצה החדשה שונה מ2 המזהים של הקבוצות שנרצה לאחד (כלומר זה מזהה חדש), נבדוק האם זה מזהה של קבוצה קיימת במבנה הנתונים. אם כבר קיימת קבוצה עם מזהה זה, נחזיר שגיאה.
- ניגש למזהים של הקבוצות עצמן (מסוג Team) עבור הקבוצות המתאחדות, ומשם ניגש עבור כל קבוצה לכל אחד מעצי השחקנים שלה (2 עצי שחקנים לכל קבוצה, אחד ממוין לפי ת"ז ואחד לפי מספר גולים).
- ניגש עבור כל אחד מעצי השחקנים בקבוצות לשדה המכיל את מספר הצמתים בעץ.
- ניגש עבור כל אחת מהקבוצות לשדה בו שמור האם קיים שוער בקבוצה, ונגדיר שדה דומה עבור הקבוצה החדשה, בהתאם ל2 השדות הללו.
- ניגש עבור כל אחת מהקבוצות אל ה-top_scorer_player שלה ונגדיר עבור הקבוצה המאוחדת את הtop_scorer החדש.
- ניצור 2 עצי שחקנים חדשים עבור הקבוצה המאוחדת – אחד הממוין לפי ת"ז ואחד הממוין לפי גולים.
- עבור כל אחד מהשחקנים השייכים ל-2 הקבוצות, נבצע בעזרת פונקציה רקורסיבית מעבר על כל אחד מעצי השחקנים של הקבוצות (הממוינים לפי id), ונעדכן את מספר המשחקים בהם כל שחקן השתתף בהתאם לקבוצה אליה היה שייך.
- נקרא לפונקציית unite_trees שמימשנו בעץ פעמיים – פעם אחת עבור העצים הממוינים לפי ת"ז ופעם אחת עבור העצים הממוינים לפי גולים. פונקציה זו כפי שכתבנו בפירוט הפונקציות של העץ, מאחדת בין 2 עצים באמצעות מערך inorder, ובאמצעות פונקציית insert_node.
- כעת עבור כל אחד מ2 העצים המאוחדים נעדכן את מספר הצמתים הקיימים בהם.
- ניצור קבוצה חדשה, זוהי הקבוצה המאוחדת, בעזרת הבנאי של מחלקת Team, ונכניס את כל הערכים המתאימים שהגדרנו לפני כן.

- ניצור node חדש עבור הקבוצה, על מנת להכניס אותה לעץ הקבוצות הכולל.
- נסיר את הצמתים של 2 הקבוצות הישנות מעץ הקבוצות לפי ת"ז ונמחק אותן ממבנה הנתונים.
- נכניס את הnode שיצרנו עבור הקבוצה החדשה לעץ הקבוצות לפי תז. אם הקבוצה המאוחדת היא חוקית (11 שחקנים ושוער), נכניס אותה גם לעץ של הקבוצות החוקיות.

נכונות המימוש: הפונקציה תאחד בין 2 הקבוצות שניתנו ותיצור בעזרתן קבוצה חדשה המכילה את השחקנים מ2 הקבוצות הישנות. בנוסף תסיר את הקבוצות הישנות ממבנה הנתונים.

נכונות הסיבוכיות: חיפוש כל אחד מהעצים הוא בסיבוכיות O(logk), לכן סהכ החיפוש של שתיהן יהיה גם כן O(logk). בדיקה האם המזהה של הקבוצה החדשה כבר קיים, היא גם באמצעות פונקציית חיפוש ולכן גם תהיה בסיבוכיות O(logk). גישה לכל אחד מעצי השחקנים של הקבוצות, בסיבוכיות O(1). גישה לשדה השוער והגדרת שדה דומה עבור הקבוצה החדשה, סיבוכיות O(1). גישה לtop_scorer של כל אחת מהקבוצות והגדרת שדה זה עבור הקבוצה החדשה O(1). יצירת עצי שחקנים חדשים O(1). ביצוע הפונקצייה הרקורסיבית שעוברת על כל אחד מעצי השחקנים היא בסיבוכיות של $O(n_{teamId1} + n_{teamId2})$, מפני שמעבר על כל אחד מהעצים הוא בסיבוכיות O(n), כאשר n הוא מספר השחקנים בקבוצה המתאימה. ביצוע פונקציית unite_trees היא בסיבוכיות של $O(n_{teamId1} + n_{teamId2})$ גם כן. גישה אל כל אחד מהעצים המאוחדים ועדכון מספר הצמתים בסיבוכיות O(1). יצירת קבוצה חדשה בO(1), כך גם יצירת הnode החדש עבור הקבוצה. הסרת הצמתים של הקבוצות הישנות והכנסת הnode החדש, כל אחד מהם בנפרד בסיבוכיות O(logk) ולכן שלושתם ביחד גם כן בסיבוכיות O(logk).
לבסוף, הסיבוכיות של הפונקציה כולה היא
$O(1) + O(logk) + O(n_{teamId1}) + O(n_{teamId2}) \leq \boldsymbol{O(logk + n_{teamId1} + n_{teamId2})}$.

output_t < int > get_top_scorer(int teamId)

מימוש הפונקציה:

- נבדוק שהקלט שקיבלנו חוקי.
- אם teamID<0, בעץ השחקנים הכולל הממוין לפי ת"ז, ניגש למשתנה שמכיל את המצביע לשחקן שהבקיע את מספר הגולים הגדול ביותר מבין כל השחקנים במערכת. נחזיר את הID שלו.
- אם teamID>0, בעץ הקבוצות הכולל (ממוין לפי ת"ז) נמצא את הקבוצה שזהו המזהה שלה באמצעות פונקציית find, ושם ניגש למשתנה שמכיל את המצביע לשחקן שהבקיע הכי הרבה גולים בקבוצה הזו, ונחזיר את הID של אותו שחקן.

נכונות המימוש: הפונקציה תחזיר את השחקן עם מספר הגולים הגבוה ביותר, מהקבוצה המתאימה או מכל השחקנים במערכת.

נכונות סיבוכיות: אם teamID<0, גישה למצביע בעץ השחקנים הכולל והוצאת המזהה של השחקן הזה, היא בסיבוכיות O(1) (כי אין בעץ הזה שום חיפוש, גישה ישירה למצביע). סהכ במקרה זה, סיבוכיות O(1).
אם teamID>0, חיפוש הקבוצה המתאימה בעץ הקבוצות הכולל שממוין לפי ID היא בסיבוכיות O(logk), גישה למצביע באובייקט של הקבוצה והחזרת המזהה של השחקן שאליו מצביעים, הוא בסיבוכיות O(1). סהכ במקרה זה, סיבוכיות $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

output_t < int > get_all_players_count(int teamId)

מימוש הפונקציה:

- נבדוק שהקלט חוקי.
- אם teamID<0, נלך לעץ השחקנים הכולל הממוין לפי ת"ז ושם ניגש לערך num_of_nodes.
- אם teamID>0, נחפש את הקבוצה המתאימה בעץ הקבוצות הכולל, ניגש לעץ השחקנים הממוין לפי ID של הקבוצה המתאימה וניגש לערך num_of_nodes.

נכונות המימוש: הפונקציה תחזיר את השחקן עם מספר הגולים הגבוה ביותר, מהקבוצה המתאימה או מכל השחקנים במערכת.

נכונות הסיבוכיות: אם teamID<0, גישה לאובייקט של עץ השחקנים הכולל ואז גישה למשתנה שמכיל את מספר הצמתים, שניהם בסיבוכיות O(1) ולכן סה"כ במקרה זה, מדובר בסיבוכיות O(1).
אם teamID>0, חיפוש בעץ הקבוצות עבור הקבוצה המתאימה יהיה בסיבוכיות של O(logk), ואז גישה לערך של מספר הצמתים בעץ הוא ב-O(1). לכן סה"כ במקרה זה, הסיבוכיות תהיה $O(1) + O(logk) \leq \boldsymbol{O(logk)}$.

StatusType get_all_players(int teamId, int * const output)

מימוש הפונקציה:

- נבדוק שהקלט חוקי.
- אם teamID<0, נלך לעץ השחקנים הכולל הממוין לפי מספר הגולים של כל שחקן. נשתמש בפונקציית inorder שמימשנו ונייצא מערך ממוין לפי פרמטרי המיון המתאימים.
- אם teamID>0, נלך לעץ השחקנים הממוין לפי מספר גולים של הקבוצה המתאימה, נשתמש בפונקציית inorder שמימשנו ונייצא מערך ממוין לפי פרמטרי המיון המתאימים.

נכונות המימוש: הפונקציה תמלא את המערך לפי inorder של השחקנים בקבוצה המתאימה או השחקנים בכל המערכת, בהתאם לקלט.

נכונות הסיבוכיות: אם teamID<0, הקריאה לפונקציית inorder והביצוע שלה הוא בסיבוכיות O(n), כי מדובר במעבר על כל השחקנים במערכת, אך פעם אחת בלבד על כל שחקן. סה"כ במקרה זה, הסיבוכיות תהיה O(n), כאשר n הוא מספר השחקנים במערכת.

אם teamID>0, חיפוש בעץ הקבוצות עבור הקבוצה המתאימה יהיה בסיבוכיות של O(logk), ואז בקבוצה עצמה מעבר על כל אחד מהשחקנים על מנת לייצא את המערך הממוין לפי inorder, יהיה בסיבוכיות $O(n_{teamId})$, כאשר n_{teamId} הוא מספר השחקנים בקבוצה זו. סהכ במקרה זה הסיבוכיות תהיה $\boldsymbol{O(logk + n_{teamId})}$.

output_t < int > get_closest_player(int playerId, int teamId)

מימוש הפונקציה:

- נבדוק שהקלטים חוקיים
- נחפש את הקבוצה המתאימה, אם היא אינה קיימת נחזיר שגיאה.
- ניגש לעץ השחקנים הקבוצתי הממוין לפי גולים ובו נחפש את השחקן הרצוי. אם הוא אינו שייך לקבוצה הזו או השחקן היחיד במערכת, נחזיר שגיאה.
- נחשב מיהו השחקן הקרוב אליו, מבין 2 השחקנים הצמודים לו בinorder (כלומר אחד לפניו ואחד אחריו בinorder). את החישוב נבצע כמתואר בשאלה, לפי מפתחות המיון המתאימים.
- נחזיר את המזהה של הקרוב יותר.

נכונות המימוש: הפונקציה מוצאת עבור השחקן הנתון את השחקן הקרוב אליו מבין אלו שבמערכת.

נכונות הסיבוכיות: חיפוש הקבוצה בעץ הקבוצות הכולל הוא בסיבוכיות O(logk). חיפוש השחקן המתאים בעץ השחקנים הקבוצתי הממוין לפי ID, יהיה בסיבוכיות O(logn), כאשר n הוא מספר השחקנים בקבוצה הספציפית הזו. חישוב השחקן הקרוב ביותר יבוצע ע"י השוואות פשוטות ופונקציות get, לכן בסיבוכיות O(1).
סה"כ הסיבוכיות של מימוש הפונקציה יהיה **O(logk+logn).**

output_t < int > knockout_winner(int minTeamId, int maxTeamId)

מימוש הפונקציה:

- נבדוק שהקלטים חוקיים.
- נעבור על עץ הקבוצות הממוין לפי ID בעזרת פונקצייה רקורסיבית, שמגבילה את המעבר על העץ רק עבור צמתים שנמצאים בטווח הניתן בקלט לפונקציה ונחזיר ערך שתכולתו היא מספר הקבוצות ששייכות לטווח. אם הגודל הזה הוא 0, נחזיר שגיאה.
- נקצה 2 מערכים חדשים של int בגודל r, הערך שהוחזר מהפונקציה הרקורסיבית שתיארנו לעיל. מערך אחד ישמור את כל הid של הקבוצות הבטווח, והמערך השני יחזיק את הניקוד שלהם.
- בעזרת פונקציה רקורסיבית נוספת, שגם היא מגבילה את המעבר על העץ רק לטווח המתאים, נכניס בסדר inorder את כל הת"ז של הקבוצות השייכות לטווח למערך הראשון שהקצנו, ואת ערכי הניקוד של כל קבוצה למקום המתאים במערך השני (בהתאם לאינדקס הקבוצה במערך של הid, כי לפי דרישות השאלה המזהה של הקבוצה הוא זה שקובע את הסדר שלהן).
- אם יש רק איבר אחד במערך, הוא יהיה הקבוצה המנצחת ולכן נחזיר את המזהה שלו.
- כעת נבצע את הסבבים של התחרות. בסבב הראשון נעבור על כל ערכי המערך, ונבצע תחרות בין כל 2 איברים לפי הניקוד שלהם. את הid של הקבוצה המנצחת ואת סכום הניקוד של שתי הקבוצות שהתחרו (ועוד 3 על הנצחון) נעביר לאינדקס הנמוך יותר מבין 2 הקבוצות, זהו תמיד יהיה אינדקס זוגי במערך. (במידה ויש מספר אי זוגי של קבוצות במערך, נגיע לאיבר האחרון ככל הנראה בשלב הסופי).
- בסבב השני, נעבור רק על האינדקסים הזוגיים, ונבצע את אותה הפעולה כך שגם הפעם, הנתונים של הקבוצה המנצחת ישמרו באינדקס הנמוך יותר מבין השניים של הקבוצות.
- בסבב השלישי, כבר נעבור על כל אינדקס רביעי, וכך הלאה, עד שנגיע למצב שבו הקפיצה שלנו בין אינדקסים, גדולה יותר מr, כאן נעצור.
- בסיום הסבבים, הערך של הקבוצה המנצחת יהיה במקום ה0 במערך, לכן זהו הערך שנחזיר.

נכונות המימוש: הפונקציה תבצע את תהליך הנוקאאוט כך שהמיון של הקבוצות יהיה בסדר עולה לפי מזהה הקבוצה, ולבסוף תחזיר את המזהה של הקבוצה המנצחת.

נכונות הסיבוכיות: מעבר על עץ הקבוצות על מנת לקבל את גודל המערך בתחום היא בסיבוכיות O(logk), כי עוברים על עץ הקבוצות החוקיות, כלומר מספר הקבוצות שם בוודאות קטן או שווה למספר השחקנים במערכת (אין שם קבוצות ריקות).
הקצאת 2 מערכים בסיבוכיות O(1).
מעבר נוסף על עץ הקבוצות יהיה מאותה הסיבה גם הוא בסיבוכיות O(logk), כי בפועל הוא יעבור בדיוק על אותן קבוצות. הכנסת הערכים למערך וקידום האינדקס שלהם מבוצעים בסיבוכיות O(1).
עבור ביצוע הסבבים של התחרות: מספר הקבוצות שעליהן נעבור בכל סבב, קטן כל פעם בחצי, לכן מדובר בסדרה גיאומטרית סופית עם q=0.5, כלומר זוהי סדרה מתכנסת לערך קבוע (כפול r) ולכן כל ביצוע הסבבים יהיה בסיבוכיות O(r). החזרת הערך במקום ה0 מהמערך היא בO(1). לכן סה"כ הסיבוכיות של הפונקציה, היא: $\boldsymbol{O(\log\min\{n,k\})+r}$.

סיבוכיות מקום:
כפי שהסברנו לעיל על מבנה הנתונים הכללי בו בחרנו להשתמש, המבנה יכיל מספר סופי של עצי AVL, כך שכל עץ יכול להכיל מקסימום n צמתים אם מדובר בעץ שחקנים, או מקסימום k צמתים אם מדובר בעץ קבוצות. מכאן נסיק כי כל פעולה אותה נבצע, ובאופן כללי כל המידע שישמר בזכרון בתרגיל שמימשנו, לא יחרגו מסיבוכיות המקום הנתונה בתרגיל, הלוא היא O(n+k).