חלק יבש – תרגיל רטוב 2 מבני נתונים

מגישים: איתי ברקוביץ 316088970, יובל לזרוס 206703951, איל שטיין 208622142

הסבר כללי על מבנה הנתונים:

המחלקה **Olympic** מחזיקה:

Olympic
TeamTreeHashTable* m_teamHashTable StrengthTeamTree* m_strengthTeamTree

1. טבלת ערבול דינמית של עצי קבוצות $TeamTreeHashTable$.
2. עץ דרגות בינארי מאוזן של כלל הקבוצות במערכת, $StrengthTeamTree$, הממוין תחילה עפ"י כוח הקבוצה ולאחר מכן עפ"י מזהה הקבוצה (ביחס הפוך, כלומר מזהה קטן יותר מימין).

תכן של המחלקות:

PlayerTree
PlayerTreeNode* m_root int m_amountOfPlayers

void insert(Player player); // O(log k)
void remove(Player player); // O(log k)
PlayerTreeNode* getRoot() const; // O(1)
void setRoot(PlayerTreeNode* newRoot); // O(1)
void setSize(int newSize); // O(1)
void InOrderToArray(Player* array); // O(k)
Player findMedian(); // O(log k)
int getAmountOfPlayers() const; // O(1)

StrengthTeamTree
int m_size StrengthTeamNode* m_root

int getMaxRankInTree() const // O(1)
int getNumWins(Team* team) // O(log n)
void insert(Team* team) // O(log n)
void remove(Team* team) // O(log n)
Team* findTeamByIndex(int indexToSearch) // O(log n)
int findTeamIndex(Team* team) // O(log n)
Team* findCorrespondingLowPowerTeam(int lowPower) // O(log n)
Team* findCorrespondingHighPowerTeam(int highPower) // O(log n)
void accumulateWinnings (Team* teamLowBracket, Team* teamHighBracket) // O(log n)

TeamTreeHashTable
DynamicTeamTreeArray* m_array

StatusType insert(Team* teamToInsert) // O(1) amortized
StatusType remove(Team* teamToRemove) // O(1) amortized
Team* find(int teamId) // O(1) amortized
bool member(int teamId) // O(1) amortized
bool isEmpty() const // O(1)

PlayerTreeNode
PlayerTreeNode* m_left PlayerTreeNode* m_right int m_height int m_amountOfPlayersInNode int m_sizeOfSubTree // amount of Players, not nodes Player m_player

StrengthTeamNode (partial diagram)
StrengthTeamNode* m_left StrengthTeamNode* m_right Team* m_team int m_height int m_maxRankInSubTree int m_sizeOfSubtree int m_accumulate;
int getSizeOfSubtree() const // O(1) void pushAccumulateToChildren() // O(1) int getAccumulate() const // O(1)

DynamicTeamTreeArray
AVL_Tree<Team>** m_array int m_arraySize int m_numberOfElements

int getArraySize() const // O(1)
int getNumberOfElements() const // O(1)
void updateNumberOfElements() // O(1)
void incrementNumberOfElements() // O(1)
void decrementNumberOfElements() // O(1)
 	
//operator [] to access elements in the array
AVL_Tree<Team>* operator[](int index) // O(1)

Team
PlayerTree* m_players; arrivalQueue* m_arrivalQueue; int m_teamId; int m_numOfWins; int m_numOfPlayers; int m_strength;

int getID() const // O(1)
int getNumOfWins() const // O(1)
int getNumPlayers() const // O(1)
int getStrength() const // O(1)
void changeNumOfWins(int change) // O(1)
void decrementNumberOfPlayers() // O(1)
void incrementNumberOfPlayers() // O(1)
void insert(int player) // O(log k)
void removeNewestPlayer() //
void addArrivalQueueToStart(Team* otherTeam) // O(k_other)
void updateStrength() // O(log k)
void mergeTeams(Team* otherTeam) // O(k_this + k_other)
bool operator==(const Team& other) const // O(1)
bool operator>=(const Team& other) const // O(1)

Player
typedef int Player

ArrivalQueue
arrivalQueueNode* m_head; int m_size

Player top() const // O(1)
void pop() // O(1) amortized
void push(Player player) // O(1) amortized
void appendToStart(arrivalQueue* otherToAdd) // O(k_otherToAdd)
int getSize() const // O(1)

ArrivalQueueNode
arrivalQueueNode* m_next Player m_data

הערה: במחלקה DynamicTeamTreeArray השתמשנו בעצי AVL מהתרגיל הקודם וזהו עץ AVL סטנדרטי כפי שנלמד בקורס ולכן לא נרחיב עליו.

Team

המחלקה מחזיקה:

1. עץ דרגות מאוזן המחזיק שחקנים $PlayerTree$ (מכיוון שלשחקן אין מזהה ייחודי העץ יחזיק גם את מספר המופעים של שחקן עם אותו כוח)
2. רשימה מקושרת $arrivalQueue$ של שחקנים לפי סדר ההכנסה שלהם לקבוצה.
3. מספר השחקנים בקבוצה.
4. מספר הניצחונות (השדה הזה לא בהכרח מעודכן כי בעת טורניר מתבצע חישוב של הורדה והחסרה של נצחונות בטורניר לפי דירוג הקבוצה)

5. הכוח של הקבוצה
6. המזהה הייחודי של הקבוצה

תתי-פונקציות של המחלקה $Team$:

1. $void\ addArivalQueueToStart(Team * otherTeam)$- לוקחת את הרשימה של הקבוצה השנייה ומוסיפה אותה לתחילת הרשימה שלה, בסיבוכיות $O(k_{players\ in\ team\ 2})$.
2. $void\ updateStrength()$ – מעדכנת את השדה $strength$ של הקבוצה $O(\log k)$.
3. $void\ insert(Player\ player)$ – מכניסה גם לרשימה וגם לעץ $O(\log k)$.
4. $void\ removeNewewstPlayer()$ - $O(\log k)$.

פירוט העץ $StrengthTeamTree$:

זהו עץ דרגות מאוזן המסודר לפי כח של קבוצה (אם שתי קבוצות בעלות אותו הכח אז הקבוצה עם המזהה הנמוך יותר תהיה מימין, כלומר תיחשב כחזקה יותר) בו כל $Node$ מחזיק מידע על כמות הקבוצות בתת העץ שלו, הדירוג המקסימלי בתת העץ שלו וכמות הנצחונות שיש להוסיף לחישוב נצחונות ($accumulate$) כאשר עוברים ב$Node$ הזה במסלול חיפוש. כמו כן, הוא מחזיק את הקבוצה אותה הוא מסמל. נשים לב כי השדה $accumulate$ של כל $Node$ משפיע על מספר הנצחונות של קבוצה שכדי להגיע אליה צריך לעבור בקבוצה שלנו בזמן מסלול החיפוש. לכן אם הגענו לקבוצה בעץ, בהכרח סכמנו את כל הנצחונות שיש להוסיף לה עד להגעה אליה.

מכיוון שזהו עץ דרגות כפי שנלמד בהרצאה, בשדות של שורש העץ נמצא מידע על הדירוג המקסימלי בכל העץ וכמות הקבוצות בכל העץ.

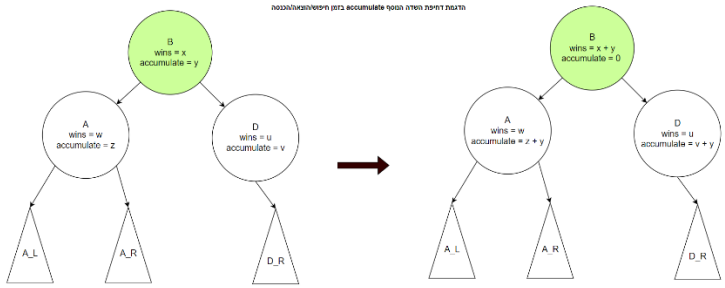
פונקציות עזר ראויות לציון של עץ הקבוצות:

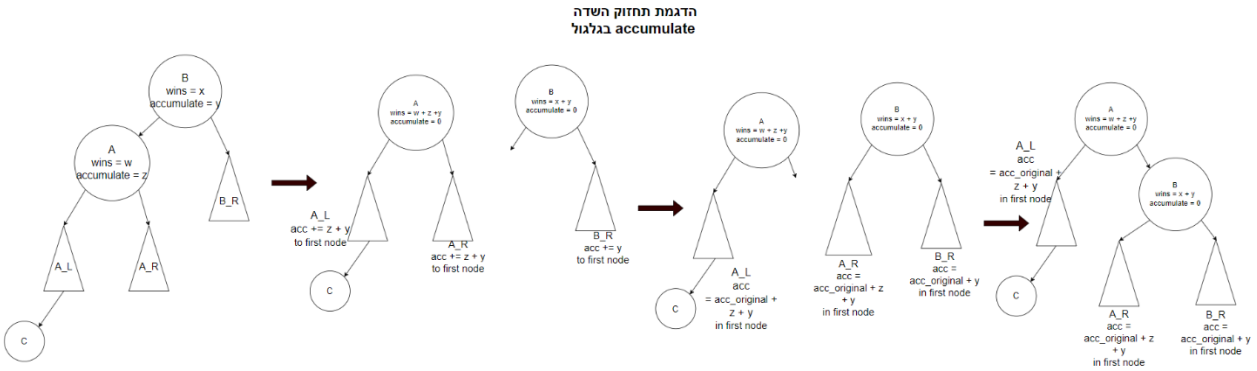
1. $int\ getNumWins(Team * team)$ - $O(\log n)$
   - הפונקציה מבצעת חיפוש בעץ עד להגעה לקבוצה.
     - בכל שלב בחיפוש, היא "דוחפת" את כמות הנצחונות הנוספים (להלן $accumulate$) לתוך שדה הנצחונות של הקבוצה ב$Node$ וגם לתוך השדות $accumulate$ של הבנים שלו (אם קיימים)
   - נשים לב כי בהגעה ל$Node$ של הקבוצה אותה מחפשים כל הנצחונות של הקבוצות על מסלול החיפוש עודכנו, בפרט מספר הנצחונות של הקבוצה אותה מצאנו. לכן נחזיר את מספר הנצחונות של הקבוצה.
2. $void\ accumulateWinnings\ (Team *\ teamLowBracket, Team * teamHighBracket)$
   - הפונקציה פועלת במהלך טורניר ומתבצעת ע"י שתי פונקציות עזר לע מנת להוסיף נצחונות רק לטווח מסוים:
     i. $accumulate(Team * teamHighBracket)$ שתעלה את הניצחונות של כלל הקבוצות עד הקבוצה הנתונה כולל ב- $O(\log n)$.

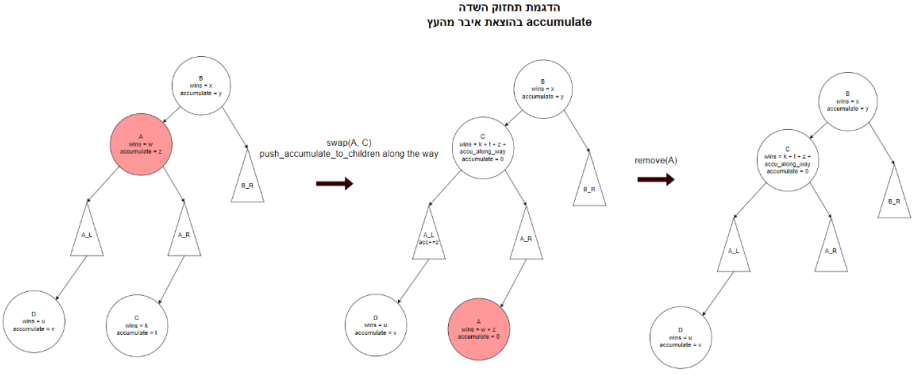
ii. $deccumulate(Team * teamLowBracket)$ שתוריד את הניצחונות של כלל הקבוצות עד הקבוצה הנתונה, לא כולל ב- $O(\log n)$.

3. $void\ insert(Team * teamToInsert)$

- נשים לב כי כאשר אנו עוברים על מסלול החיפוש, עידכנו את מספר הנצחונות של כל קבוצה עד להגעה למקום בו יש להכניס את הקבוצה החדשה. מכיוון שמוסיפים את הקבוצה החדשה כעלה, ניתן להוסיף אותה עם $accumulate = 0$.







פירוט $TeamTreeHashTable$: זוהי טבלת ערבול כפי שנלמד בהרצאה המכילה מערך דינמי שכל תא בו מכיל עץ AVL של מצביעים לקבוצות. הוא אחראי על בדיקת האם קבוצה קיימת במערכת, וכן על הוצאה או הכנסה של קבוצה למערכת.

המחלקה $arrivalQueue$: רשימה מקושרת המכילה שחקנים, כאשר השחקן בתחילת הרשימה הוא השחקן שנוסף אחרון, לפי עיקרון LIFO.

פירוט הפונקציות של Olympics ומימושן:

olympics_t()

הפונקציה יוצרת מופע של Olympic, מאתחלת טבלת ערבול ב-O(1) ומאתחלת עץ קבוצות ריק ב-O(1).

<u>סיבוכיות זמן:</u> $O(1)$ במקרה הגרוע.

<u>סיבוכיות מקום:</u> $O(1)$ - כי אין קבוצות במערכת וטבלת הערבול מתחילה בגודל קבוע כאשר בכל תא יש עץ AVL ריק.

virtual ~ olympics_t()

הפונקציה תקרא להורס של העץ $StrengthTeamTree$ ולהורס של $TeamTreeHashTable$.

נעבור על מבני הנתונים המכילים את כל הקבוצות ובתוך כל קבוצה נעבור על כל השחקנים שלה.

<u>סיבוכיות זמן:</u> $O(n+k)$ במקרה הגרוע.

<u>סיבוכיות מקום:</u> כעומק הרקורסיה של ההורסים של העצים, שהיא $max\{log(k), log(n)\}$.

StatusType add_team(int teamId)

הפונקציה תבדוק את נכונות הקלט, אם לא תקין נחזיר שגיאה.
נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$ שפועלת בסיבוכיות של $O(1)$ משוערך, בדומה למה שנלמד בהרצאה, אך עם עצי AVL בכל תא במקום רשימה מקושרת. נשים לב כי הסיבוכיות עודנה $O(1)$, כפי שראינו בתרגול. אם היא לא נמצאת בטבלת הערבול, נקרא לפונקציה $TeamTreeHashTable \rightarrow insert(teamId)$ שפועלת בסיבוכיות $O(1)$ משוערך, בדומה למה שנלמד בהרצאה, אך עם עצי AVL בכל תא במקום רשימה מקושרת. עדיין לא נכניס לעץ הקבוצות הגדול כי זה לא עומד בסיבוכיות כרגע.

<u>סיבוכיות זמן:</u> $O(1)$ משוערך, בממוצע על הקלט.

<u>סיבוכיות מקום:</u> ייתכן כי טבלת הערבול תצטרך להכפיל את גודלה, בסיבוכיות מקום של $O(n)$.

StatusType remove_team(int teamId)

הפונקציה תבדוק את נכונות הקלט, אם לא תקין נחזיר שגיאה.
נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$. אם הקבוצה בטבלת הערבול, נבצע

הוצאה של הקבוצה מעץ הקבוצות ב-$O(\log n)$ בעזרת הפונקציה $StrengthTeamTree \rightarrow remove()$
ולאחר מכן נבצע הסרה של הקבוצה מטבלת הערבול ב-O(1) משוערך בעזרת הפונקציה
$TeamTreeHashTable \rightarrow remove(teamId)$. הפונקציה הזו גם תמחק את הקבוצה בסיבוכיות
$O(k_{playersInTeam})$.

<u>סיבוכיות זמן</u>: $O\left(\log(n) + k_{playersInTeam}\right)$ משוערך, בממוצע על הקלט.

<u>סיבוכיות מקום</u>: ברקורסיה של ההוצאה מעץ הקבוצות יש סיבוכיות מקום של $O(\log n)$. ייתכן כי טבלת הערבול תצטרך להתכווץ ותבצע זאת בסיבוכיות מקום של $O(n)$.

StatusType add_player(int teamId, int playerStrength)

הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.
נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$ הפועלת בסיבוכיות $O(1)$ משוערך. אם הקבוצה קיימת בטבלת הערבול נוציא את הקבוצה מעץ הקבוצות (אם היא בעץ) ב- $O(\log n)$. נוסיף את השחקן לעץ השחקנים בקבוצה בסיבוכיות $O(\log k)$ ולרשימה המקושרת של השחקנים ב$O(1)$ (הוספת השחקן תעדכן את כח הקבוצה ב-$O(\log k)$). לאחר מכן נוסיף את הקבוצה לעץ הקבוצות ב$O(\log n)$

<u>סיבוכיות זמן</u>: $O\left(\log(n) + \log\left(k_{playersInTeam}\right)\right)$ במקרה הגרוע.

<u>סיבוכיות מקום</u>: המקסימלי מבין עומקי הרקורסיה בהכנת שחקן לעץ השחקנים או בהוצאה/הכנסה של קבוצה מעץ הקבוצות. כלומר $O(\max\{\log n, \log k\})$.

StatusType remove_newest_player(int teamId)

הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.

נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$ הפועלת בסיבוכיות $O(1)$ משוערך.
אם הקבוצה קיימת בטבלת הערבול נוציא את הקבוצה מעץ הקבוצות (אם היא בעץ) ב- $O(\log n)$.

מוצאים מי השחקן שנוסף אחרון ברשימה המקושרת של השחקנים - $O(1)$ במקרה הגרוע. מוציאים את השחקן מהעץ השחקנים בקבוצה בסיבוכיות $O(\log k)$ ולאחר מכן מוציאים את השחקן הזה מהרשימה המקושרת ב-$O(1)$. נעדכן את הכח של הקבוצה ב-$O(\log k)$. לאחר מכן נוסיף את הקבוצה לעץ הקבוצות ב$O(\log n)$.

<u>סיבוכיות זמן</u>: $O\left(\log(n) + \log\left(k_{playersInTeam}\right)\right)$ במקרה הגרוע.

<u>סיבוכיות מקום</u>: המקסימלי מבין עומקי הרקורסיה בהכנת שחקן לעץ השחקנים או בהוצאה/הכנסה של קבוצה מעץ הקבוצות. כלומר $O(\max\{\log n, \log k\})$.

output_t<int> play_match(int teamId1, int teamId2)

הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.

נבדוק ששתי הקבוצות קיימות בסיבוכיות של $O(1)$ משוערבעזרת קריאה לפונקציות:

$TeamTreeHashTable \rightarrow member(int\ teamId1)$

$TeamTreeHashTable \rightarrow member(int\ teamId2)$

נמצא מצביעים לשתי הקבוצות על ידי קריאה לפונקציות ב-$O(1)$ משוערך:

$TeamTreeHashTable \rightarrow find(int\ teamId1)$

$TeamTreeHashTable \rightarrow find(int\ teamId2)$

נעדכן את שדה הניצחונות של הקבוצה המנצחת (נבחר את הקבוצה המנצחת על ידי השוואה בין החוזק של הקבוצות).

נוציא את שתי הקבוצות מעץ הקבוצות ולאחר מכן נחזיר אותן ב$O(\log n)$ כדי לעדכן את שדות הדירוג בעץ.

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע.

<u>סיבוכיות מקום</u>: $O(\log n)$, כעומק הרקורסיה בהוצאה והכנסה של קבוצות מעץ הקבוצות.

output_t<int> num_wins_for_team(int teamId)

הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.

נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$ שפועלת בסיבוכיות $O(1)$ משוערך.
נחזיר את ערך החזרה מהפונקציה:

$StrengthTeamTree \rightarrow getNumWins(teamId)$, כאשר הפונקציה מבצעת חיפוש בעץ הקבוצות ומעדכנת את שדה הנצחונות של כל קבוצה על מסלול החיפוש, בפרט היא מעדכנת את שדה הנצחונות של הקבוצה אותה אנו מחפשים. החיפוש מתבצע ב$O(\log n)$.

<u>סיבוכיות זמן</u>: $O(\log(n))$ במקרה הגרוע.

<u>סיבוכיות מקום</u>: $O(\log n)$, כעומק הרקורסיה בחיפוש בעץ הקבוצות.

output_t<int> get_highest_ranked_team()

כפי שנלמד בהרצאה על עצי דרגות, אם כל $Node$ שומר את הדירוג המקסימלי בתת העץ שלו, אז ה-$Node$ בשורש של עץ הקבוצות מכיל בשדות שלו את ה$rank$ הכי גבוה ולכן ניקח את ה$rank$ שלו (מספר נצחונות + כח). השדות הללו מעודכנים בכל הנכסה והוצאה, וכן בזמן טורניר, בכל הוספה של ניצחון אחד לתחום של קבוצות, אז הדירוג המקסימלי בתת העץ עם הקבוצות הללו יגדל באחד. מכיוון שדירוג מוגדר כסכום (לינארי) של חוזק ומספר נצחונות אז אם הקבוצה עם הדירוג המקסימלי הייתה בתחום שהוספנו לו ניצחון אז מספר

הדירוג הקסימלי יגדל ב-1. אחרת, בזמן החזרה ברקורסיה שאחרי הוספת ניצחון נעדכן בכל $Node$ שהגענו את הדירוג המקסימלי בתת העץ להיות המקסימלי מבין שזה שבשני הבנים שלו ולכן תמיד ניקח את הדירוג המקסימלי המעודכן בתת העץ. בפרט, בשורש העץ ניקח את הדירוג המקסימלי של העץ כולו. העדכון הזה מבתעצ בכל הוספה של ניצחון בטורניר ולכן לאחר הטורניר השדה עם הדירוג המקסימלי בשורש מעודכן.

<u>סיבוכיות זמן:</u> $O(1)$ במקרה הגרוע.

<u>סיבוכיות מקום:</u> $O(1)$.

StatusType unite_teams(int teamId1, int teamId2)

הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.

נקרא לפונקציה $TeamTreeHashTable \rightarrow member(int\ teamId)$ שפועלת בסיבוכיות $O(1)$ משוערך.

נוציא את שתי הקבוצות מעץ הקבוצות בסיבוכיות $O(\log n)$, זו הוצאה מעץ דרגות כפי שנלמד בהרצאה.

נהפוך את שני עצי השחקנים של הקבוצות למערכים ממויינים כפי שנלמד בתרגול $O(k_{playersInTeam1} + k_{playersInTeam2})$. נתחשב גם במספר רב של שחקנים בעלי אותו כח.

נהרוס את עץ השחקנים של קבוצה מספר 1 בסיבוכיות $O(k_{playersInTeam1})$.

נאחד את שני המערכים הללו בסיבוכיות $O(k_{playersInTeam1} + k_{playersInTeam2})$. נשים לב כי הם יישארו ממויינים. נהפוך את המערך המאוחד לעץ $O(k_{playersInTeam1} + k_{playersInTeam2})$ כפי שנלמד בתרגול, כאשר אם שחקן מופיע יותר מפעם אחת אז ב $Node$ המתאים נעדכן את המונה שסופר כמה שחקנים יש עם הכוח הזה. העץ הזה יהיה עץ השחקנים החדש של קבוצה 1.

נחבר את שתי הרשימות המקושרות של הקבוצות כאשר הזנב של קבוצה 2 מתחבר לראש של 1 בעזרת הפונקציה בסיבוכיות $O(k_{playersInTeam2})$.

נמחק את הקבוצה השנייה מהתא בטבלת הערבול ב-$O(1)$ משוערך עבור החיפוש ואז טבלת הערבול מוחקת את הקבוצה לגמרי, שזו פעולה ב $O(k_{playersInTeam2})$.

נחשב את הכוח של קבוצה 1 בסיבוכיות $O(\log k)$.

נכניס את הקבוצה בחזרה לעץ הקבוצות בסיבוכיות $O(\log n)$.

סיבוכיות זמן: $O(\log(n) + k_{playersInTeam1} + k_{playersInTeam2})$ משוערך.

<u>סיבוכיות מקום:</u> המקסימלי מבין $O(\log n)$ ו-$O(\log k)$, כי מתבצעות סיורים רקורסיביים בעץ השחקנים ומתבצעות פעולות הסרה והכנסה מעץ הקבוצות.

output_t<int> play_tournament(int lowPower, int highPower)

1. הפונקציה תבדוק את נכונות הקלט שניתן ע"י המשתמש, אם לא תקין נחזיר שגיאה.
2. נשים לב שקבוצות עם כוח אפס לא נחשבות בטורניר (כי $lowPower \leq 0$ זה INVALID_INPUT) ולכן הן גם לא נמצאות בעץ הקבוצות.
3. העץ מסודר לפי כוח וסידור משני בסדר הפוך של מספר מזהה (כלומר אם הכוח של שתי קבוצות זהה אז הקבוצה עם המספר המזהה הנמוך יותר תהיה מימין)
4. מחפשים את האינדקס של הקבוצה עם כוח $lowPower$ ו-id הכי גדול מבין הקבוצות עם אותו כוח (מבין הקבוצות עם אותו כוח היא תהיה הכי משמאל בעץ) בסיבוכיות $O(\log n)$.
5. מחפשים את האינדקס של הקבוצה עם כוח $highPower$ ו-id הכי קטן מבין הקבוצות עם אותו כוח (מבין הקבוצות עם אותו כוח היא תהיה הכי מימין בעץ) בסיבוכיות $O(\log n)$.
6. נחסיר את האינדקסים שלהם כדי לדעת כמה איברים יהיו בטורניר.
7. נבדוק האם מספר הקבוצות בטורניר, i, הוא חזקה של 2 ב-$O(1)$.
8. בלולאה, נמצא את הקבוצה ה- $\frac{i}{2}$ (מובטח לנו שהיא קיימת מכיוון שבדקנו ש-i הוא חזקה של 2), ונשמור כמשתנה זמני מצביע לאותה קבוצה.

   נקרא לפונקציה $accumulateWinnnings$ עם הקבוצה עם כוח $highPower$ ועם הקבוצה שמצאנו (זאת עם האינדקס $\frac{i}{2}$).

   בסוף כל איטרציה נחלק את i ב-2 עד ש $\frac{i}{2} == 1$.

   סה"כ $\log(i)$ איטרציות של הלולאה.

   הפונקציה $accumulateWinnnings$ מקבלת מצביעים לקבוצות התואמות ל-tempPower ו-highPower ומוסיפה ניצחון אחד לכל הקבוצות בטווח עד highPower כולל ומחסירה ניצחון אחד מכל הקבוצות עד tempPower (לא כולל). היא פועלת בסיבוכיות $O(\log n)$. כפי שהסברנו מקודם, היא מעדכנת את הדירוג המקסימלי בכל תת עץ אליו היא מגיעה בזמן הוספה או החסרה של ניצחון ולכן הדירוג המקסימלי בשורש הוא הדירוג המקסימלי בעץ.
9. הקבוצה המנצחת בטורניר היא הקבוצה עם highPower (וה-id הכי נמוך) שמצאנו בתחילת הפונקציה מכיוון שבכל משחק היא בהכרח קיבלה תוספת ניצחון, כי יש לה את הכח הגדול ביותר מבין הקבוצות בטורניר. לכן נחזיר את המזהה שלה.

סיבוכיות זמן: מבצעים שני חיפושים בעץ הקבוצות ב$O(\log n)$. לאחר מכן מבצעים $O(\log i)$ איטרציות שכל אחת מהן עולה $O(\log(n))$ (חיפוש בעץ הקבוצות לפי אינדקס ואז הוספת ניצחון לכל הקבוצות עד הכח הגבוה והחסרת ניצחון לכל הקבוצות בתחום עד הכח הנמוך)

לכן סיבוכיות הזמן הכוללת היא $O(\log(i) * \log(n))$ במקרה הגרוע.

<u>סיבוכיות מקום:</u> בכל איטרציה בלולאה, עומק הרקורסיה היא $O(\log(n))$ ולכן זו סיבוכיות המקום.