

מבנה הנתונים יכיל:

world_cup_t:

- **Avl<Team> all_teams** - עץ חיפוש בינארי מאוזן AVL שמכיל את כל הקבוצות במערכת
- **Avl<Team> good_teams** - עץ AVL שמכיל רק את הקבוצות הכשירות
- **Avl<Player> players_by_id** - עץ AVL שמכיל את כל השחקנים במערכת כאשר המפתחות בעץ הם המזהה של כל שחקן.
- **Avl<Player> players_by_goals** - עץ AVL שמכיל את כל השחקנים במערכת כאשר האופרטור בעץ זה מוגדר על ידי הסדר שמוגדר בפונקציה `get_all_players`. **כלומר:** שחקן 1 > שחקן 2 אם ורק אם מס' השערים שהבקיע שחקן 1 קטן ממס' השערים שהבקיע שחקן 2. אם מס' השערים זהה, אז אם מספר הכרטיסים שקיבל שחקן 1 גדול ממס' הכרטיסים שקיבל שחקן 2, אם מס' הכרטיסים גם זהה אז אם המזהה של שחקן 1 קטן מהמזהה של שחקן 2.
- **num_of_good_teams** - מספר הקבוצות הכשירות - גם מספר הצמתים בעץ `good_teams`.
- **הבחנה חשובה:** מספר זה תמיד קטן ממספר השחקנים במערכת ובפרט ממספר הקבוצות במערכת.
- **num_of_players** - מספר השחקנים במערכת.
- **Player* top_scorer** - השחקן שמדורג הכי גבוה **במערכת** לפי מספר שערים (אם שווה אז לפי האופרטור שהוסבר למעלה).

:Team

- **teamId** - מזהה הקבוצה
- **Avl<Player> players_by_id** - עץ השחקנים של הקבוצה ממיון לפי מזהה השחקנים.
- **Avl<Player> players_by_goals** - עץ השחקנים של הקבוצה ממיון לפי שערים בהתאם להסבר בעץ של המערכת.
- **num_of_players** - מספר השחקנים בקבוצה.
- **points** - מספר הנקודות של הקבוצה.
- **goals** - סכום השערים של כל השחקנים בקבוצה.
- **cards** - סכום הכרטיסים שקיבלו השחקנים בקבוצה.
- **num_of_matches** - מספר המשחקים ששיחקה הקבוצה מאז שנכנסה למערכת.
- **Player* top_scorer** - השחקן שמדורג הכי גבוה בקבוצה.
- **num_of_goal_keepers** - מספר השוערים בקבוצה. - שניתן יהיה לעקוב בקלות מתי קבוצה הופכת לכשירה ומתי כבר לא.
- **Team* team_in_all_teams** - מצביע של קבוצה בעץ הקבוצות הכשירות לקבוצה המתאימה שלה בעץ הקבוצות של כל המערכת. (המצביע הוא null בקבוצות בעץ כל המערכת)

:Player

- **playerId** - מזהה השחקן
- **teamId** - מזהה הקבוצה של השחקן
- **Team* player_team** - מצביע לטיפוס שמייצג את קבוצת השחקן. - כך נוכל לקבל את הפרטים של הקבוצה בגישה ישירה.
- **goals** - מספר השערים שהבקיע השחקן. - מתעדכן בהתאם.
- **cards** - מספר הכרטיסים שקיבל השחקן. - מתעדכן בהתאם.
- **games_played** - מספר המשחקים ששיחק השחקן כשהצטרף למערכת.
- **games_of_team_without_player** - מספר המשחקים ששיחקה קבוצת השחקן לפני שהצטרף אליה. אם שחקן עבר קבוצה באמצע, הערך מתעדכן בהתאם.
- **bool goalKeeper** - אם השחקן הוא שוער יקבל true, אחרת false.
- **Player* closest_player** - מצביע לשחקן הקרוב ביותר של השחקן. מתעדכן תוך כדי הפונקציות add_player, remove_player, update_player_stats. $O(\log n)$.
וזה מתעדכן אצל השחקן בעץ שממוין לפי id **בקבוצתו** כדי שהסיבוכיות של הפונקציה get_closest לא תיפגע ותהיה $O(\log k + \log n_{teamId})$ כאשר teamId זו קבוצת השחקן שיש להחזיר עבורו את השחקן הקרוב ביותר אליו.
- **bool checking_by_goals** - אם נרצה את השחקן בעץ לפי id, יקבל את הערך false, אחרת יקבל true.
אופרטור של שחקן יהיה מוגדר לפי הדגל הזה. אם הדגל דולק האופרטור יהיה רק לפי id של השחקן, אחרת לפי מה שהוגדר למעלה עם השערים, כרטיסים ואז מזהה.
- **Player* himself_in_players_by_id** - מצביע לשחקן בעץ השחקנים לפי id של כל המערכת.
המצביע קיים מפני שבשינוי קבוצה של שחקן עלינו לשנות את המידע הזה עבורו בצומת שמייצג אותו בעץ כל המערכת כי אנו משתמשים בו לצורך מציאת קבוצת השחקן.

הערה: ראינו בהרצאה שהכנסה, הוצאה, חיפוש בעץ חיפוש מאוזן (AVL) מתקיימים בסיבוכיות של $O(\log n)$ במקרה הגרוע כאשר n זה מספר הצמתים בעץ. לכן כל פעולה של הכנסה, הוצאה או חיפוש צומת בעץ בפונקציות יהיו בסיבוכיות $O(\log n)$.

סיבוכיות מקום: מבנה הנתונים מכיל עצים בגודל n שחקנים לכל היותר ועצים בגודל k קבוצות לכל היותר, בנוסף מספר שדות קבוע לכל שחקן ולכל קבוצה. ומספר שדות סופי בתוך world cup
לכן $O(n+k)$ מקום.

add_player

אם מגיע ערך לא חוקי invalid input יוחזר.

מוצאים את הקבוצה של השחקן בעץ הקבוצות (חיפוש בעץ ב \log). אם לא מצאנו תיזרק חריגה ונחזיר FAILURE.

יוצרים שחקן עם הנתונים המתאימים שהוא by_goals ואחד עם אותם נתונים שיהיה by_id.

מוצאים את השחקן הקרוב ביותר אליו בצורה הבאה:

מוצאים את השחקן העוקב לו בדירוג השחקנים (על ידי חיפוש בעץ players_by_goals בסיבוכיות $O(\log n)$)

מוצאים את השחקן שלפניו בדירוג השחקנים (על ידי חיפוש בעץ players_by_goals בסיבוכיות $O(\log n)$)

משווים מי יותר קרוב ב- $O(1)$ ומעדכנים אותו אצל השחקן שעתיד להיכנס גם במערכת ואז זה יעודכן גם בצומת שלו בקבוצתו.

מכניסים את השחקן שהוא by_goals לplayer_by_goals בסיבוכיות $O(\log n)$ (אם השחקן כבר קיים תיזרק

חריגה ונחזיר FAILURE). ואת השחקן שהוא לא by_id לplayers_by_id בסיבוכיות $O(\log n)$.

טענה: השחקנים היחידים שיכול להשתנות להם closest_player הם שני השחקנים הקרובים ביותר. כי השחקן שנוסף יכול להיות הקרוב ביותר אליהם החדש.

כי אם קיים שחקן נוסף שקרוב יותר זו סתירה לדרך בחירה של אותם שחקנים.

מוצאים את השחקנים הקרובים (אלו שמצאנו מקודם) בסיבוכיות $O(\log n)$

ומעדכנים אצלם את closest_player באותה צורה שעדכנו את השדה בשחקן שיצרנו.

מעדכנים את top_scorer על ידי השמת הערך המקסימלי בעץ players_by_goals בtop_scorer ב $O(\log n)$.

מעדכנים את מספר השחקנים במערכת. $(+1)$

מוצאים את השחקן שהכנסנו בתוך העץ players_by_id בסיבוכיות $O(\log n)$ ומעדכנים לשחקן שיצרנו את

המצביע לשם.

בודקים אם לפני הוספת השחקן הקבוצה כשירה:

אם כן מוסיפים את השחקן לקבוצה, מוסיפים את goals והcards שלו לקבוצה ומעדכנים את top_scorer שלה

כפי שמעדכנים את top_scorer של המערכת. מחזירים success

אם לא, מוסיפים את השחקן לקבוצה, מוסיפים את goals והcards שלו לקבוצה ומעדכנים את top_scorer שלה

שלה כפי שמעדכנים את top_scorer של המערכת.

בודקים האם אחרי הוספת השחקן היא כשירה

אם כן מוסיפים אותה לעץ הקבוצות הכשירות, מעדכנים שם מצביע לקבוצה בעץ כל הקבוצות ומוסיפים 1 למספר

הקבוצות הכשירות.

מחזירים success.

remove_player(int teamId)

אם מגיע ערך לא חוקי יוחזר invalid input.

יוצרים שחקן זמני עם הת.ז של השחקן שרוצים למחוק ומוציאים את השחקן בplayers_by_id. (אם לא מצאנו תיזרק חריגה ויוחזר failure).

יוצרים עותק של השחקן ובעותק משנים את by_goals לאמת וכך מוצאים את השחקן בplayer_by_goals. שומרים פוינטר לקבוצה שלו בעזרת השדה team שבו.

מוחקים את השחקן מהקבוצה (מורידים goals, cards, num_of_goal_keepers בהתאם) ומורידים את מספר השחקנים בקבוצה ב-1.

בודקים אם הקבוצה הייתה כשירה ולאחר ההסרה כבר לא, אז מוחקים אותה מgood_teams ומחסירים 1 ממספר הקבוצות הכשירות. (סיבוכיות של חיפוש בgood_teams היא פחות מlogn כי בכל קבוצה כשירה יש לפחות 11 שחקנים ולכן יש $11 * \text{good_teams} \geq n$ ולכן $O(\log n) < O(\log(\text{num_of_good_teams}))$) מוחקים את השחקן בplayers_by_id ואת השחקן בplayers_by_goals.

מעדכנים את closest_player בשחקן שלפניו ובשחקן שאחריו כמו בadd_player.

מעדכנים את top_scorer על ידי השמת הערך המקסימלי בעץ players_by_goals בtop_scorer מחסירים 1 ממספר השחקנים ויוחזר success.

סיבוכיות: חיפוש והסרה מעצי השחקנים - $O(\log n)$ במקרה הגרוע.

update_player_stats

אם מגיע ערך לא חוקי יוחזר invalid input.

יוצרים שחקן זמני עם הת.ז של השחקן שרוצים למחוק ומוציאים את השחקן בplayers_by_id.

שומרים פוינטר לקבוצה שלו בעזרת השדה team שבו. מעתיקים את השחקן לשחקן copy כדי לשמור את הנתונים הנוכחים.

מסירים את השחקן מהמערכת ומקבוצתו. ויוצרים שחקן חדש עם הערכים המעודכנים בהתאם למקרים הבאים:

אם $\text{games_played} == 0$ מכניסים את השחקן למערכת מחדש עם הנתונים החדשים.

אם $\text{games_played} > 0$ מכניסים את השחקן למערכת עם הוספת הנתונים החדשים אלה הקיימים.

ומחזירים success.

סיבוכיות: חיפוש, הסרה והכנסה לעצי השחקנים - $O(\log n)$ במקרה הגרוע.

add_team(int teamId, int points)

אם אחד הערכים לפחות לא חוקי יוחזר INVALID INPUT בהתאם.

אחרת, יוצרים קבוצה Team עם המזהה שלה ומספר הנקודות ההתחלתי, מאתחלים את עצי השחקנים בה להיות ריקים, ואת שאר הערכים לאפסים. ומכניסים את הקבוצה לעץ הקבוצות של המערכת. אם הקבוצה כבר קיימת

תיזרק שגיאה ונחזיר FAILURE, אחרת, הקבוצה תיכנס בהצלחה ויוחזר SUCCESS.

סיבוכיות: הכנסה לעץ הקבוצות של המערכת, שהוא עץ חיפוש מאוזן, בסיבוכיות $O(\log k)$ במקרה הגרוע כאשר k זה מספר הקבוצות במערכת וזהו סך הסיבוכיות של הפונקציה במקרה הגרוע כנדרש.

remove_team(int teamId)

עבור ערך לא חוקי יוחזר INVALID INPUT בהתאם.

יוצרים קבוצה זמנית עם המזהה הזה כדי למצוא את הקבוצה שנרצה להסיר. אם הקבוצה לא קיימת תיזרק חריגה מתאימה ונחזיר FAILURE. אחרת, נבדוק אם יש בקבוצה שחקן אחד לפחות - נחזיר FAILURE. אחרת, נסיר את הקבוצה מעץ הקבוצות וסיימנו.

סיבוכיות: הסרה מהעץ מתקיימת ב- $O(\log k)$ במקרה הגרוע, וזו אכן הסיבוכיות של הפונקציה.

play_match(int teamId1, int teamId2)

עבור ערך לא חוקי יוחזר INVALID INPUT בהתאם.

מוצאים את הקבוצות בעץ הקבוצות של המערכת ב- $O(\log k)$ (אם אחת הקבוצות לא קיימת תיזרק שגיאה ויוחזר FAILURE).

בודקים ששתי הקבוצות כשירות (11 שחקנים ושוער אחד לפחות), אם אחת מהן לא, נחזיר FAILURE. בודקים מי הקבוצה המנצחת על ידי הנוסחה שמחשבת את התוצאה (עם points, goals, cards). הקבוצה המנצחת תקבל תוספת 3 נק לערך points שלה, ובמקרה של תיקו נוסף 1 לכל קבוצה. בנוסף, מוסיפים 1 לערך num_of_matches של כל קבוצה כדי לעדכן בהתאם את games_played של כל השחקנים בקבוצות.

סיבוכיות: מציאת הקבוצות בעץ הקבוצות ב- $O(\log k)$ במקרה הגרוע, בדיקות ועדכונים ב- $O(1)$, לכן סה"כ סיבוכיות הפונקציה $O(\log k)$ במקרה הגרוע כנדרש.

get_num_played_games(int playerId)

עבור ערך לא חוקי יוחזר INVALID INPUT.

מוצאים את השחקן בעץ השחקנים של המערכת לפי id על ידי שחקן זמני שיוצרים עם המזהה שהתקבל. (אם השחקן לא קיים תיזרק שגיאה ונחזיר FAILURE).

מחשבים ומחזירים את הערך הבא:

$games_played + num_of_matches - games_of_team_without_player$

כאשר num_of_matches זה מספר המשחקים ששיחקה קבוצת השחקן. ככה, יובטח שהערך שיתקבל הוא אכן מספר המשחקים שהשחקן שיחק.

סיבוכיות: מציאת השחקן ב- $O(\log n)$ במקרה הגרוע והחזרת הערך מהשחקן שמצאנו. סה"כ $O(\log n)$.

get_team_points(int teamId)

עבור ערך לא חוקי יוחזר INVALID INPUT.

מוצאים את הקבוצה בעלת המזהה teamId בעץ הקבוצות של המערכת. אם לא מוצאים נחזיר FAILURE. אחרת, מצאנו את הקבוצה. נחזיר את ערך points של הקבוצה.

סיבוכיות: מציאת הקבוצה בעץ הקבוצות של המערכת ב- $O(\log k)$ במקרה הגרוע והחזרה של ערך הנקודות ב- $O(1)$. סה"כ $O(\log k)$ במקרה הגרוע כנדרש.

unite_teams(int teamId1, int teamId2, newTeamId)

עבור ערך לא חוקי כלשהו יוחזר INVALID INPUT.

מוצאים את הקבוצות בעץ הקבוצות של המערכת. (אם לא קיימים נחזיר FAILURE). מתקיים ב- $O(\log k)$ במקרה הגרוע. עבור כל קבוצה:

- מכניסים את השחקנים מהעץ `players_by_id` ב**inorder** לתוך מערך.
 - מכניסים את השחקנים מהעץ `players_by_goals` ב**inorder** לתוך מערך.
- ממזגים את מערכי השחקנים לפי `id` בצורה ממוינת לתוך מערך גדול שמכיל את כל השחקנים של שתי הקבוצות ממוינים לפי `id`.

באותו אופן בדיוק, ממזגים את השחקנים לפי `goals` למערך אחד. מתקיים ב: $O(n_{teamId1} + n_{teamId2})$.

מחלקים למקרים:

1. `newTeamId == teamId1`

- *שומרים דגל שאומר האם קבוצה 1 (הקבוצה בעלת המזהה `teamId1`) כשירה לפני האיחוד או לא.
- *עוברים על השחקנים מקבוצה 2 (הקבוצה בעלת המזהה `teamId2`) ונעדכן לכל שחקן את:
 - המצביע והמזהה של הקבוצה לה שייך - כעת לקבוצה 1.
 - מוסיפים ל**`games_of_team_withput_player`** שלו את `num_of_matches` של קבוצה 1.
- *מחברים את כל הערכים של קבוצה 2 עם קבוצה 1: `points, goals, cards, num_of_players`.
- *`num_of_matches, num_of_goal_keepers` - זה מתקיים ב- $O(n_{teamId2})$.
- *אם קבוצה 2 הייתה קבוצה כשירה, מוחקים אותה מעץ הקבוצות הכשירות.
- *מוחקים את קבוצה 2 מעץ הקבוצות של המערכת. - $O(\log k)$ במקרה הגרוע.
- *יוצרים עץ כמעט שלם ריק שמכיל את מספר הצמתים המתאים שהוא סכום השחקנים בשתי הקבוצות לפי מה שראינו בתרגול.
- *מכניסים לקבוצה 1 את השחקנים מהמערך הממוזג לעץ כמעט שלם שבנינו. (ב**inorder**).
- *מעדכנים `top_scorer` חדש לקבוצה המאוחדת. (מציאת המקסימלי ב- $O(\log n)$ כי הולכים ימינה עד שאי אפשר יותר).
- *אם קבוצה 1 לא הייתה קבוצה כשירה לפני ועכשיו כן כשירה, נכניס אותה לעץ הקבוצות הכשירות עם מצביע לקבוצה בעץ הקבוצות.

2. `newTeamId == teamId2`

זהה למקרה 1 רק שהכל הפוך בין קבוצה 1 וקבוצה 2. (הכל סימטרי לחלוטין).

3. `newTeamId` משני המזהים

- *יוצרים קבוצה עם המזהה החדש ומכניסים אותה עם המזהה `newTeamId` והערכים של קבוצה 1 (לא כולל עצי השחקנים) לעץ הקבוצות של המערכת. (אם קיימת קבוצה עם המזהה הזה תיזרק חריגה ויוחזר FAILURE).
- *מוצאים את הקבוצה בעץ הקבוצות כדי שנוכל להכניס את כל השחקנים ולעדכן את כל המידע בהתאם.
- *עוברים על השחקנים במערכים הממוזגים (לפי `id` ולפי `goals`) ונבצע עבור כל שחקן את הדברים הבאים:

- אם השחקן מקבוצה 1, מוסיפים לgames_of_team_withput_player שלו את num_of_matches של קבוצה 2.
- אם השחקן מקבוצה 2, מוסיפים לgames_of_team_withput_player שלו את num_of_matches של קבוצה 1.
- מעדכנים את המצביע והמזהה לקבוצה אצל השחקן להיות הקבוצה עם המזהה החדש.
- מחברים את כל הערכים של קבוצה 2 לקבוצה 1 כי היא מכילה את המידע של קבוצה 1. (points, goals, num_of_players, num_of_matches, num_of_goal_keepers, cards).
- מכניסים את השחקנים ב-inorder מהמערך לעצים המתאימים להם בקבוצה. (id, goals) בהתאמה מהמערך לעץ). $O(n_{teamId1} + n_{teamId2})$.
- מעדכנים את top_scorer החדש. $O(\log(n_{teamId1} + n_{teamId2}))$.
- *אם הקבוצות היו קבוצות כשירות, מסירים אותן מעץ הקבוצות הכשירות.
- מסירים את הקבוצות מעץ הקבוצות של המערכת. $O(\log k)$.
- *אם הקבוצה החדשה המאוחדת היא קבוצה כשירה, מכניסים אותו לעץ הקבוצות הכשירות, כמובן עם מצביע לקבוצה בעץ הקבוצות.
- סיבוכיות: מציאות הקבוצות והכנסות לעצי הקבוצות מתבצעים ב- $O(\log k)$ במקרה הגרוע, מעברים על המערכים, ועדכונים אצל השחקנים, הכנסה לעצים בגודל של כמות השחקנים בשתי הקבוצות יחד - $O(n_{teamId1} + n_{teamId2})$ אז סה"כ $O(\log k + n_{teamId1} + n_{teamId2})$.
- get_top_scorer(int teamId)**
- אם מגיע ערך לא חוקי יוחזר invalid input.
- אם $teamId < 0$ ומספר השחקנים במערכת הוא 0 יוחזר failure. אחרת, מחזירים את המזהה של top_scorer של המערכת ב- $O(1)$ מהמצביע שמתוחזק.
- אם $teamId > 0$ מוצאים את הקבוצה בעץ הקבוצות ב- $O(\log k)$. אם אין שחקנים בקבוצה מחזירים failure. אחרת, מחזירים success ואת המזהה של top_scorer שנמצא בקבוצה.
- סיבוכיות: אם $teamId < 0$ זה מתקיים ב- $O(1)$, אחרת, $O(\log k)$.
- get_all_players_count(int teamId)**
- אם מגיע ערך לא חוקי יוחזר invalid input.
- אם $teamId < 0$ יוחזר num_of_players של המערכת. אחרת, מוצאים את הקבוצה ב-all_teams ב- $O(\log k)$ ומחזירים את מספר השחקנים בקבוצה. (אם הקבוצה לא קיימת בעץ יוחזר failure).
- סיבוכיות: אם $teamId < 0$ זה מתקיים ב- $O(1)$, אחרת, $O(\log k)$.
- get_all_players(int teamId)**
- אם $teamId < 0$ נכניס למערך בגודל מס' השחקנים במערכת את השחקנים מהעץ players_by_goals ב-inorder. אחרת, מוצאים את הקבוצה ומכניסים למערך בגודל מס' השחקנים בקבוצה את השחקנים מהעץ players_by_goals של הקבוצה ב-inorder.

סיבוכיות: אם $teamId < 0$, $O(n)$, אחרת ב- $O(\log k + \log n_{teamId})$.

get_closest_player(int playerId, int teamId)

אם הקלט לא יהיה תקין יוחזר invalid input. מוצאים את הקבוצה (אם לא נמצאה יוחזר failure), מוצאים את השחקן בקבוצה ובו יהיה closest_player שלו כי דאגנו לעדכן גם שם בפונקציות בהם השתנה.

סיבוכיות: מציאת הקבוצה ואז את השחקן בקבוצה - $O(\log k + \log n_{teamId})$ במקרה הגרוע.

knockout_winner(min teamId, max teamId)

אם הקלט לא יהיה תקין יוחזר invalid input.

ניצור מערך של קבוצות ונוסיף אליו את כל הקבוצות הכשירות שהמזהה שלהם נמצא בטווח הנתון. - נניח שמס' הקבוצות כנ"ל הוא r .

נעשה זאת באופן הבא: בעץ good_teams נמצא את הערך המינימלי שבטווח, המקסימלי והאב המשותף הכי גבוהה שלהם. מכיוון שהערך המינימלי בטווח אז גם כל התת עץ הימני שלו אשר בו ערכים גדולים ממנו, נמצאים בטווח. לכן, נוסיף אותו ואז אותם לפי הסדר במערך, נעלה מעלה בעץ ונעשה אותו דבר לכל ערך בעץ עד שעולים לאב המשותף. האב המשותף בטווח, לכן נוסיף אותו ומשם נרד ימינה בעץ, אם הערך בטווח אז כל התת עץ השמאלי שלו נמצא בטווח גם כן כי הוא גדול מהאב המשותף אך קטן מהערך המקסימלי בטווח. נמשיך לרדת ימינה בכל העץ עד שנסיים אותו. כמובן $k \leq \text{num_of_good_teams}$ וגם $\text{num_of_good_teams} \leq 11n$ ולכן החיפוש בעץ good_teams הוא $O(\log(\min\{n, k\}) + r)$.

נכניס את המזהים של הקבוצות למערך ids בגודל r ואת ערכי הנוסחא של הניקוד של הקבוצות לתוך מערך scores. נתאר את האלגוריתם שמוצא את winner לאחר שכל הערכים הדרושים נמצאים במערכים:

אם r אי זוגי, נשמור את הקבוצה האחרונה שבמערך בצד.

כל עוד r עוד לא 1:

רצים 0 עד $r-1$ ומשווים כל זוג קבוצות: כל קבוצה שמנצחת שומרים את id שלה ואת הניקוד שהוא סכום הניקודים פלוס 3 במערכים המתאימים בתא המתאים.

מחלקים את r בשתיים. אם r אי זוגי כעת ויש קבוצה בצד, נכניס אותה למערך.

אם הוא אי זוגי ואין קבוצה בצד, נשמור את הקבוצה האחרונה בצד.

לבסוף, אם הגענו ל- $r=1$ ואין קבוצה בצד סיימנו ויש לנו את המזהה של הקבוצה המנצחת.

סיבוכיות: תהליך הכנסת הקבוצות המתאימות מתקיים בסיבוכיות $O(\log(\min\{n, k\}) + r)$ במקרה הגרוע,

האלגוריתם למציאת הקבוצה המנצחת רץ: $O(r) = r(1 + 1/2 + \dots) = r + r/2 + r/4 + \dots + 1$.

לכן, סה"כ הסיבוכיות במקרה הגרוע של הפונקציה היא $O(\log(\min\{n, k\}) + r)$.