

**מבני נתונים 1 2340218**

**תרגיל בית רטוב 2**

## תיאור כללי של המערכת:

המערכת מנהלת  $m$  קבוצות רוכבים ו  $n$  רוכבים בצורה יעילה. האיברים מוחזקים ומנוהלים ע"י 3 המבנים הבאים:

- $teams[]$  Chain Hash Table (HT) דינאמי לקבוצות
  - $jockeys[]$  Chain Hash Table (HT) דינאמי לרוכבים
  - $RecordArr[]$  Chain Hash Table (HT) דינאמי לניהול מאזני קבוצות
- ניהול הקבוצות מתבצע בעזרת  $Up Tree (Union find)$  שממומש באופן מותאם ללוגיקה של הקבוצות.

## מבנה המערכת:

### 1. $teams[]$

מבנה נתונים מסוג  $hash - table$  שבו כל תא מכיל רשימה מקושרת ( $chain$ ) של  $Node$  מסוג  $team$  כאשר ה  $key$  הוא  $teamId$ . ההוספה ל-HT מתבצעת באמצעות **פונקציית ערבול** (Hash Function), שממירה את המזהה הייחודי של הקבוצה ( $teamId$ ) לאינדקס. אם מתרחשת התנגשות בכמה  $teamId's$  הם יאוכסנו באותו התא ברשימה המקושרת.

### 2. Node של קבוצה $team$

- כל אובייקט  $team$  במחלקה  $Team$  מכיל:
- $teamId$  : מזהה ייחודי של הקבוצה
  - $record$  : מאזן הנצחונות/הפסדים של הקבוצה
  - $union - find$  : כגון  $root, record$  ,  $size$  המשמשים לפעולות האיחוד וקביעת השורש החדש בעץ  $up tree$  של הקבוצות.
  - כל מצביע של  $root$  של קבוצה מאותחל להיות על עצמו.
- פעולת  $TeamActive$  - שליפה של המפתח עם הערך של הת.ז. של קבוצה מהמערכת, אם לא התקבל נאל נבדוק אם הקבוצה מצביעה על עצמה, או מצביעה על קבוצה עם הת.ז. שחיפשו.
- פעולת  $TeamExists$  - בדיקה אם מתקבל ערך נאל או לא כאשר מבצעים שאילתה על ת.ז. ספציפי אל מול ה  $ht$  שלנו.
- פעולת השליפה מה  $hash table$  מתבצעת בעזרת פונקציית ערבול  $hash function$  שממירה  $teamId$  לאינדקס במערך. לכן, כפי שנלמד בהרצאה, פעולת השליפה וההכנסה היא בסיבוכיות ממוצעת  $O(1)$  (נרחיב על מקרי הקצה בהמשך)

### 3. $jockeys[]$

ה-HT לרוכבים מבוסס על מערך דינאמי, שמכפיל את גודלו כאשר מגיעים לתקרת הקיבולת. ההוספה מתבצעת באמצעות **פונקציית ערבול** (Hash Function) שממפה את המזהה הייחודי של הרוכב ( $jockeyId$ ) לאינדקס במערך. כל תא מחזיק רשימה מקושרת שבנויה מ **Node של רוכבים**.

#### 4. Node של רוכב jockey

כל אובייקט jockey במחלקה Jockey מכיל:

- jockeyId: מזהה ייחודי של הרוכב
- record: מאזן הנצחונות/הפסדים של הרוכב
- teamId: מזהה ייחודי של הקבוצה אליה שייך הרוכב

#### 5. עץ up Tree לניהול הקבוצות Union – Find

כפי שהוזכר, ה- *up – tree* שמנהל את הקבוצות ממומש ברובו במחלקות *Team* ו *TeamArr*:

- בכל אובייקט *Team* קיימים שדות המסייעים לקביעת ושינוי השורש, גודל, ומאזן הקבוצה
- המחלקה *TeamArr* מבצעת את פעולות ה- *union – find* על ה- *Hash – Table* שמכיל בתוכו אובייקטים מסוג *Team*.

#### 6. recordArr[]

מנהל את מאזני הנצחונות/הפסדים של הקבוצות וממפה לפי ערך המאזן. זהו *hash table* שמחזיק אובייקטים מסוג *Record*. בכל אובייקט *Record* נשמר *chain Hash array* דינאמי של קבוצות בעלי אותו מאזן.

למערך פונקציות של הוספת קבוצה לרקורד, הוצאת קבוצה מרקורד, ובדיקה אם אפשר לבצע את פעולת unite by record, המתבצעות כך-  
add\_team\_to\_record - בודק אם קיים הרקורד עם הערך שאליו רוצים להכניס את הקבוצה, אם כן מוסיף אותה, ואם לא יוצר את הרקורד ומכניס את הרקורד למערך רקורדים ואת הקבוצה למערך הקבוצות.  
remove\_team\_from\_record - מוצא את הרקורד במערך רקורדים לפי המפתח של הרקורד המבוקש, מסיר את הקבוצה המבוקשת לפי המפתח שלה, במידה וכעת הרקורד נותר ריק, הוא יימחק.  
can\_unite\_by\_record - פעולה הבודקת אם יש לנו במערך הרקורדס שני רקורדים בעלי מפתח חיובי ומפתח שלילי של הרקורד המבוקש, ובמידה וכן ושני הרקורדים מכילים רק קבוצה אחת, יוחזר אמת.  
כל הפונקציות הנ"ל מבצעות לכל היותר כמות קבועה של חישובים ופעולות, ולכן הסיבוכיות שלהן היא  $O(1)$ .

## מימוש הפונקציות:

### **:plains\_t()**

יצירת המבנים  $teams[], jockeys[], recordArr[]$   
סיבוכיות זמן:  $O(1) = 3O(1)$  - הקצאה ריקה  
סה"כ:  $O(1)$

### **:virtual ~plains\_t()**

מחיקת שלושת מבני הנתונים שיצרנו ואת התוכן שלהם.  
נשים לב כי לכל היותר יש כמות רקורדים ככמות הקבוצות (דאגנו לכך שלא ייתכן רקורדים ריקים).  
 $O(n) + O(r) + O(m) \leq O(n) + 2O(m) = O(n + m)$  (מספר כלל הרוכבים n, הקבוצות m, מאזני  
קבוצות r)  
סה"כ:  $O(n+m)$

### **:StatusType add\_team(int teamId)**

פונקציה זה מוסיפה קבוצה חדשה למערכת שלנו. היא מחפשת ומוסיפה לתוך *HashTable* בשימוש בפונקציית ערבול לפי המזהה הייחודי.

#### **שלבי תהליך ההכנסה:**

- מקבלת *teamId* ומוודא שהקלט תקין ( $teamId > 0$ )
- בודקת אם קיימת כבר קבוצה במערכת עם אותו המזהה (כולל קבוצות שכבר לא פעילות) בעזרת *TeamExists* במידה וקיים מחזיר *FAILURE*
- במידה ולא קיימת יוצרת אובייקט *Team(teamId)* ומכניס ל- $teams[]$  ומוסיפה את האובייקט שיצרנו ל- $records[]$  עם המאזן ההתחלתי 0

#### **ניתוח סיבוכיות זמן:**

##### מקרה גרוע:

- אם כל ה *teamId's* יתנגשו לאותו האינדקס, הרשימה המקושרת באותו התא תכיל את כל n האיברים, כנ"ל לגבי *records*. זמן המעבר וההוספה לרשימה במקרה זה היא  $O(m) = 2O(m)$

##### מקרה ממוצע:

הוספת הקבוצה ל- $teams[]$  שקולה להוספת איבר ל-*union find*, למדנו בהרצאה שהסיבוכיות זמן הממוצעת של הפעולה היא  $O(1)$ . נרחיב-

- חישוב האינדקס באמצעות פונקציית Hash -  $O(1)$
- מעבר על הרשימה המקושרת בתא- בזכות פיזור אחיד של פונקציית Hash, כל רשימה מקושרת בתא

באורך ממוצע של  $\frac{m}{k}$ . לכן, סיבוכיות ההוספה לרשימה המקושרת בממוצע -  $O(1)$

סה"כ:  $O(1)$  בממוצע על הקלט משוערך.

### **:StatusType add\_jockey(int jockeyId, int teamId)**

פונקציה זו מכניסה רוכב חדש ל-*HashTable* דינאמי שמנהל את הרוכבים שלנו במערכת. ההוספה מתבצעת בעזרת המזהה הייחודי של הרכב *jockeyId* ושל הקבוצה *teamId*

### שלבי תהליך ההכנסה:

- מוודא שהקלט תקין  $teamId, jockeyId > 0$
- בודק ב  $jockeys[]$  אם קיים רוכב במערכת עם אותו מזהה (כולל קבוצות לא פעילות), במידה וקיים מחזיר  $FIALURE$
- מחפש את הקבוצה ב  $teams[]$  - בודק שהקבוצה פעילה על ידי שליפת הערך עם המפתח של תעודת הזהות שלה, ובודק אם היא מצביעה על עצמה.
- מוודא שהקבוצה פעילה - בדיקה ב  $Hashfind - teams[]$
- יוצר אובייקט  $jockey(jockeyId, teamId)$  ומכניס לתוך  $jockeys[]$

### ניתוח סיבוכיות זמן:

#### מקרה גרוע:

- אם כל ה  $jockeyId$ 's יתנגשו לאותו האינדקס, הרשימה המקושרת באותו התא תכיל את כל  $n$  האיברים. זמן המעבר וההוספה לרשימה במקרה זה היא  $O(n)$
- באופן דומה עבור  $teamId$ 's -  $O(m)$
- סכ"ה המקרה הגרוע הוא:  $O(n) + O(m)$

#### מקרה ממוצע:

- חישוב האינדקס באמצעות פונקציית Hash -  $O(1)$
- מעבר על הרשימה המקושרת בתא- בזכות פיזור אחיד של פונקציית Hash, כל רשימה מקושרת בתא באורך ממוצע של  $\frac{n}{k}$  (מספר הרוכבים חלקי אורך המערך). זמן החיפוש ברשימה הוא  $O(1)$
- באותו האופן עבור מציאת הקבוצה ובדיקת הפעילות שלה (פעולת  $find$  של  $union find$ )
- הוספה לרשימה המקושרת -  $O(1)$
- סה"כ:  $O(1)$  בממוצע על הקלט משוערך.

### **`statusType update_match(int victoriousJockeyId, int losingJockeyId)`**

#### שלבי הפעולות:

- בדיקת תקינות קלט
- מוצא ב  $jockeys$  את שני הרוכבים  $(Hashfind \times 2)$
- מאתר את השורשים של הקבוצות שלהם  $(union find \times 2)$
- מעדכן לכל רוכב את המאזן האישי  $(\pm 1)$
- מעדכן את המאזן של הקבוצה המנצחת ועדכון מיקום הקבוצה ברשומה  $recordArr$

### ניתוח סיבוכיות זמן:

מקרה גרוע: התנגשות מורבה באותו הבא Hash עשויה להביא ל  $O(n)$

#### מקרה ממוצע:

- חיפוש רוכבים באמצעות  $Hash find$  -  $O(1) = 2O(1)$
- מציאת הקבוצה אליה שייך הרוכב באמצעות  $union find$  - מעבר על  $up tree$  של 2 הקבוצות -  $2O(\log * m) = O(\log * m)$
- עדכון Hash של  $recordArr$  (במידת הצורך) -  $O(1)$  בממוצע
- סה"כ:  $O(\log * m)$  בממוצע על הקלט משוערך.

## **StatusType merge\_teams(int teamId1, int teamId2)**

**שלבי הפעולות:**

- בודק אם שתי הקבוצות פעילות  $Hashfind + Union find - find$
- שולח את המשתנים לפונקציה פנימית שמאחד את שורשי העצים לעץ אחד. האיחוד נקבע לפי גודל העצים והשורש החדש נקבע להיות הקבוצה בעלת מאזן הנצחונות/הפסדים הגדול יותר
- מעדכן שקבוצה אחת תהיה לא פעילה
- מעדכן את הרשמות ב  $recordArr$  (הוספה והסרה למאזן המתאים)

**ניתוח סיבוכיות זמן:**

מקרה גרוע: התנגשות מרובה באותו התא ב Hash עשויה להביא ל  $O(m)$  חיפוש ברשימה מקושרת של קבוצות

מקרה ממוצע:

- חיפוש hash find לקבוצות:  $O(1)$  בממוצע
- union find (find+union): חיפוש ב up tree של קבוצות ו union של 2 קבוצות -  $O(\log^*m)$  בממוצע
- עדכון רשומות ב  $recordArr$ :  $O(1)$

**סה"כ:  $O(\log^*m)$  בממוצע על הקלט יחד עם unite\_by\_record**

## **StatusType unite\_by\_record(int record)**

**שלבי הפעולות:**

- בודק  $recordArr$  האם קיימת קבוצה עם מאזן  $record +$  ומאזן  $record -$
- אם שתיהן סינגלטוניות (כלומר אם כל רשומה מכילה רק איבר אחד), נאחד אותן ע"י קריאה לפונקציה  $merge\_teams(teamId1, teamId2)$

**ניתוח סיבוכיות זמן:**

מקרה גרוע:

מקרה ממוצע: בדיקה ב  $recordsArr[]$  ב  $(Hashfind)$  -  $O(1)$   
union של קבוצות  $(union find)$  -  $O(\log^*m)$  (סיבוכיות זמן פעולת  $merge\_teams$ )

**סה"כ:  $O(\log^*m)$  בממוצע על הקלט יחד עם merge\_teams**

## **output\_t<int> get\_jockey\_record(int jockeyId)**

**שלבי הפעולות:**

- בדיקת קלט
- נמצא את הרכב ב  $jockeys$  HT
- במידה וקיים, נחזיר את ה record שלו

**ניתוח סיבוכיות זמן:**

מקרה גרוע: במידה וכל ה  $jockeyId$ 's יתנגשו לאותו האינדקס, הרשימה המקושרת באותו התא תכיל את כל ה האיברים. זמן המעבר על הרשימה במקרה זה היא לכל היותר  $O(n)$

#### מקרה ממוצע:

- מציאת רוכב בHT באמצעות  $O(1)$ -Hashfind
- החזרת הערך  $record$  של הרוכב -  $O(1)$

סה"כ:  $O(1)$  בממוצע על הקלט

**`output_t<int> get_team_record(int teamId)`**

#### **שלבי הפעולות:**

- בדיקת קלט
  - מציאת הקבוצה ובדיקה שהקבוצה עדיין פועלת במערכת בעזרת פעולת TeamActive
  - במידה והקבוצה פעילה נחזיר את ערך  $record$
- מקרה גרוע: במידה וכל ה- $teamId$ 's יתנגשו לאותו האינדקס, הרשימה המקושרת באותו התא תכיל את כל  $m$  האיברים. זמן המעבר על הרשימה במקרה זה היא לכל היותר  $O(m)$

#### מקרה ממוצע:

- מציאת רוכב בHT באמצעות  $O(1)$ -Hashfind
  - החזרת הערך  $record$  של הקבוצה של  $find$  של  $unionfind$  -  $O(1)$
- סה"כ:  $O(1)$  בממוצע על הקלט