|  |  |  |  |
| --- | --- | --- | --- |
| **שם הפונקציה** | **פרמטרים** | **ערך החזרה:** | **סיבוכיות זמן**  **(מעודכן ל10.2.24)**  **k = countries**  **m = teams**  **n = player** |
| olympics\_t() | אין | אין |  |
| virtual ~olympics\_t() | אין | אין |  |
| StatusType add\_country(int countryId, int medals) | countryId  medals | ALLOCATION\_ERROR/ INVALID\_INPUT/ /FAILURE SUCCESS |  |
| StatusType remove\_country(int countryId) | countryId | " |  |
| StatusType add\_team(int teamId, int countryId, Sport sport) | teamId  countryId  sport | " |  |
| StatusType remove\_team(int teamId) | teamId | " |  |
| StatusType add\_contestant(int contestantId , int countryId, Sport sport, int strength) | contestantId  countryId  sport  strength | " |  |
| StatusType remove\_contestant(int contestantId) | contestantId | " |  |
| StatusType add\_contestant\_to\_team(int teamId, int contestantId ) | teamId  contestantId | " |  |
| StatusType remove\_contestant\_from\_team(int teamId, int contestantId) | teamId  contestantId | " |  |
| StatusType update\_contestant\_strength(int contestantId , int change) | contestantId  change | " |  |
| output\_t < int > get\_strength(int contestantId) | contestantId | " |  |
| output\_t < int > get\_medals(int countryId) | countryId | " |  |
| output\_t < int > get\_team\_strength(int teamId) | teamId | " |  |
| StatusType unite\_teams(int teamId1, int teamId2) | teamId1  teamId2 | " |  |
| StatusType play\_match(int teamId1, int teamId2) | teamId1  teamId2 | " |  |
| output\_t < int > austerity\_measures(int teamId) | teamId | " |  |

**דרישות:**

* בהינתן מתחרה, צריך לדעת ב- אם הוא שייך לנבחרת כלשהי.
* בהינתן מדינה, צריך לדעת ב- האם קיימים לה מתחרים או נבחרות.
* בהינתן נבחרת, צריך לדעת ב- האם יש לה משתתפים או לא.
* כאשר מוסיפים מתחרה לנבחרת, צריך לבדוק אם הוא שייך כבר ל-3 נבחרות.
* ניתן לאחד רק בין קבוצות באותו ספורט שמשחקות עבור אותה מדינה ולכן נבחרת צריכה לדעת לאיזו מדינה היא שייכת.

**רעיון מימוש:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **21** | **20** | **19** | **18** | **17** | **16** | **15** | **14** | **13** | **12** | **11** | **10** | **9** | **8** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **ID** |
| 10 | 20 | 10 | 2 | 3 | 4 | 5 | 1 | 50 | 2 | 2 | 12 | 2 | 2 | 3 | 31 | 30 | 9 | 5 | 2 | 23 | חוזק |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **6** | **5** | **4** | **3** | **2** | **1** |
| 31 | 30 | 9 | 5 | 2 | 23 |

**מקרים של Austerity\_measures:**

נחלק את השחקנים לשלושה עצי ID בגדלים שווים, לפי left, middle, right בסדר עולה, כך שהשליש של השחקנים עם ה-ID הקטן ביותר נמצא בעץ שמאל, השליש האמצעי באמצע והשליש העליון בעץ הימני.

נחלק למקרים זרים ומשלימים לפי מספר השחקנים שהורדנו מכל עץ ID

* **אם הורדנו אפס שחקנים מהעץ השמאלי:**
  1. אם הורדנו אפס מהעץ האמצעי
     + אז בהכרח הורדנו 3 מימין.
       - ואז שני איברי קיצון ימין (ב"מערך" הממויין) של העץ האמצעי עוברים לעץ הימני והאיבר הקיצון ימין של העץ השמאלי עובר לאמצע.
  2. אם הורדנו שחקן אחד מהעץ האמצעי
     + אז בהכרח הורדנו שני שחקנים מעץ הימני.
       - ואז קיצון ימין של העץ האמצעי עובר לעץ הימני וקיצון ימין של העץ השמאלי עובר לעץ האמצעי.
  3. אם הורדנו שני שחקנים מהעץ האמצעי
     + אז בהכרח הורדנו שחקן אח מהעץ הימני.
       - העץ הימני נשאר זהה כי ירד בו רק שחקן אחד.
       - איבר קיצון ימין של העץ השמאלי עובר לעץ האמצעי.
  4. אם הורדנו שלושה שחקנים מהעץ האמצעי
     + אז לא נשאר עוד אנשים להוריד
     + ונעביר את ימין קיצון של העץ השמאלי לעץ האמצעי
     + ונעביר את שמאל קיצון של העץ הימני לעץ האמצעי
* **אם הורדנו שחקן אחד מהעץ השמאלי:**

1. אם הורדנו אפס שחקנים מהעץ האמצעי
   * + בהכרח הורדנו שניים מהעץ הימני
     + ואז איבר קיצון ימין מהעץ האמצעי עובר לעץ הימני.
2. אם הורדנו אחד מהאמצע
   * 1. אז בהכרח הורדנו אחד מימין
     2. ואז כל העצים נשאר ללא שינוי
3. אם הורדנו שניים מהעץ האמצעי
   * 1. אז בהכרח הורדנו אפס מהעץ הימני
     2. ואז איבר קיצון שמאל מהעץ הימני עובר לעץ האמצעי.

אם הורדנו אפס מהעץ האמצעי

* + 1. אז בהכרח הורדנו שניים מהעץ הימני.
       1. ואז איבר קיצון שמאל בעץ הימני עובר לעץ האמצעי.
* **אם הורדנו שניים מהעץ השמאלי:**

1. אם נוריד 1 מהעץ האמצעי

אז בהכרח הורדנו אפס מהעץ הימני

אז קיצון שמאל (שנותר) של העץ האמצעי יעבור לעץ השמאלי

ואיבר קיצון שמאל של העץ הימני יעבור לעץ האמצעי

1. אם נוריד 0 מהעץ האמצעי

אז בהכרח הורדנו 1 מהעץ הימני.

ואז איבר קיצון שמאל של העץ האמצעי עובר לעץ השמאלי

* **אם הורדנו שלושה מהעץ השמאלי**

1. לא נותר עוד שחקנים להעביר ולכן:
   * 1. שני איברי קיצון שמאל של העץ האמצעי יעבור לעץ השמאלי.
     2. איבר קיצון שמאל של העץ הימני יעבור לעץ האמצעי.

**איך לעשות את UNITE:**

* נבנה את העצים כך שכל איבר בעץ ID מחזיק מצביע לאיבר המקביל שלו בעץ הSTRENGH וכן הפוך.
* לוקחים את ששת העצים שמסודרים לפי הID, קוראים אותם בIN-ORDER כדי ליצור שישה מערכים ממויינים. O(n)
* ממזגים אותם למערך ID יחיד וממוין באורך O(n)
* עוברים על המערך הזה כדי למצוא את האורך שלו. O(n)
  + חוצים אותו לפי אינדקסים לשלושה חלקים שווים ונותנים לכל חלק שם: A,B,C
* ממזגים את ששת עצי ה-STRENGTH למערך יחיד ממוין. O(n)
* עוברים על כל מערך ה-ID שוב ועבור כל איבר, הולכים לאיבר המקביל שלו במערך ה-STRENGTH ומסמנים האם הוא בקבוצה A,B,C. O(n)
* עוברים שלוש פעמים על מערך ה-STRENGTH ובכל פעם בוחרים רק את האיברים של קבוצה מסוימת מתוך השלוש. O(3n)= O(n)
  + ככה במעבר הראשון נבחר רק את החוזק של איברים השייכים לקבוצה A, במעבר השני נבחר רק את האיברים השייכים לקבוצה B ובמעבר האחרון נבחר רק את האיברים של קבוצה C.
  + קיבלנו שלושה מערכים ממויינים של חוזקות, כאשר כל מערך כזה מתאים לקבוצת ID ממויינת של קבוצת המיזוג.
* כעת אפשר להפוך כל קבוצה לעץ בפני עצמה ב-O(n), כך ניצור שישה עצים.

**פונקציות נדרשות:**

* עץ AVL טמפלייטי שיכול לקבל כל סוג מידע וגם function object של פונקציית השוואה בין האובייקטים:
  + גלגול לשמאל
  + גלגול לימין
  + LL
  + LR
  + RL
  + RR
  + הכנסה
  + הוצאה
  + חיפוש
  + פונקציה לחישוב הBalance factor לכל Node
  + תתי עצים של הקבוצות צריכים עוד שלוש פונקציות נוספות:
    - Node\* Largest()
    - Node\* Smallest()
  + int getSize()
  + לממש הורס לעץ O(n) –
  + getComparisonFunction – גטר טמפלייטי
* פונקציות שמקבלות עץ, כולן בסיבוכיות O(n):
  + createArrayfromAvlTree(ID/STRENGTH, tree) – קוראinorder ואז מייצר מערך ממוין בגודל העץ. צריך לשמור מה הייתה פונקציית ההשוואה של העץ כדי שיהיה אפשר לשחזר את העץ בחזרה.
  + createAvlTreeFromArray() – מקבל מערך ממוין ופונקציית השוואה ומייצר עץ מכל הNode
  + mergeSortedArrays() – O(n)
  + calculateArraySize – O(n)
* המחלקה ContestantSmarterPointer:
  + הפונקציה UNITE:
    - מאתחלים מערך של ContestantSmarterPointer בגודל של תת העץ שבו אנחנו מתחילים.
    - נתחיל אינורדר על תת העץ כאשר יש לנו אינדקס iבשביל המערך.
      * נקרא לctor של ContestantSmarterPointerArray עם ContestantSmarterPointerAVL בתור ארגומנט.  
        הקונסטרקטור לוקח את player ואת parallel.
      * נקבע את השדה parallel של ContestantSmarterPointerAVL המקביל להיות הNODE שיצרנו עכשיו.
      * ניגש לאינדקס i במערך ונשים שם מצביע לNode החדש שיצרנו.
    - נסיים לעבור על כל העץ ID
    - נעשה MERGE לשלושת המערכים שנוצרו מעצי הID
    - נתחיל לעבור על הSTRENGH באינאורדר עם אינדקס j.
      * ניצור נקרא לctor של ContestantSmarterPointerArray עם ContestantSmarterPointerAVL בתור ארגומנט. הקונסטרקטור לוקח את player ואת parallel.
      * ניגש לparralel (כלומר לID) ונשמור בצד את הNode החדש שיצרנו ונתקן את המצביעים.
      * ניגש לאינדקס j במערך ונשים שם מצביע.
      * עושים MERGE
    - עושים את כל זה גם לקבוצה השנייה
    - מאחדים את המערכים שלהם.
    - עוברים על כל המערך של ID ומעדכנים את הenum
      * נשים לב שאנחנו חוצים את המערך של ID לשלושה חלקים לפי האינדקסים במערך.
    - עוברים על מערך strength ויוצרים שלושה מערכים בגודל n/3.
  + קוראים לפונקציה שבונה את העצים, שזה אותה פונקציה אבל הפוך.
    - פה יש עוד פונקציה שיוצרת עץ ריק.
* המחלקה TEAM:
  + getStrength – O(1)
  + getAusterityStrength – O(1)
  + addContestant – O(log n)
  + איזון מחדש של שלושת העצים – rebalanceSubTrees – O(1). צריך לבצע בכל פעם שגודל העץ מגיע
  + calculateStrength – log(n)
  + calculateAusterityStrength – log(n)
  + removeContestant – log(n)
  + getTeamSize
  + incrementTeamSize
  + decrementTeamSize
  + int getID()
* המחלקה AvlNODE:
  + Bool isLeaf(AvlNode\* currentNode)
  + int getBalanceFactor(AvlNode\* currentNode)
  + setLeftSon(AvlNode\* other)
  + setRightSon(AvlNode\* other)
  + int getHeight(AvlNode\* currentNode)
* המחלקהsubtreeAvlNode - יורשת מNODE ומוסיפה לה שדה parallelNode
  + AvlNode\* getParallel(AvlNode\* currentNode)
* המחלקה COUNTRY:
  + incrementTeamAmount
  + decrementTeamAmount
  + int getTeamAmount ()
  + int getContestantAmount ()
  + bool incrementContestantAmount
  + bool decrementContestantAmount
  + int getMedalAmount)()
  + bool isEmpty() – תבדוק האם יש קבוצות או שחקנים.
  + int getID()
* המחלקה Contestant:
  + int getID()
  + int getStrength()
  + int setStrength()
  + Sport getSport()
  + int getCountryID
  + bool isInTeam(int teamID) – אומר לנו האם השחקן שייך לקבוצה מסוימת.
  + bool registered() – לפני שמוחקים שחקן, צריך לבדוק האם הוא משויך לקבוצה כלשהי.
  + bool registerWithTeam(int teamID) – מחזיר 0 אם לא ניתן להוסיף את השחקן כי המערך מלא ומחזיר 1 אם הצלחנו להוסיף את מספר הקבוצה למערך.
    - לאחר שהצלחנו להוסיף, הפונקציה גם תחפש את הקבוצה אליה הוספנו את השחקן ותבצע עבורה את החישוב calculateStrength ו-calculateAusterityStrength
  + bool isAvailable – בודק האם יש מקום במערך הקבוצות אליהן השחקן שייך.
  + bool sameSport(int teamID) – מחזיר 1 אמ"מ מזהה הספורט של השחקן ושל הקבוצה שווה
  + bool sameCountry(int teamID) – מחזיר 1 אמ"מ מזהה המדינה של השחקן ושל הקבוצה שווה
  + updateStrengths – הפונקציה תבצע לולאה על מערך הקבוצות אליהן השחקן שייך ותבצע עבור כל קבוצה את הפונקציות calculateStrength ו- calculateAusterityStrength