מבני נתונים יבש 2

הסבר על מבנה הנתונים:

מבנה הנתונים יהיה מורכב מ UF של הציים שממומש בעזרת עצים הפוכים, איחוד לפי גדלים וכיווץ מסלולים כפי שנלמד בהרצאה.

ומ – HashTable דינמי עבור הפיראטים.

הערה: המערך של ה UF הוא טבלת ערבול דינמית. מימשנו את ה HashTable בשיטה של chain ובחרנו בפונקציית הערבול להיות פונקציה mod שמקיימת את הנחת הפיזור האחיד כפי שלמדנו בהרצאות.

בנוסף, מימשנו את הטבלת הערבול הדינמית בעזרת מערך דינמי וכאשר פקטור העומס היה גדול מ- 0.75 הגדלנו את טבלת הערבול פי 2 מגודלה הקודם כפי שלמדנו בתרגול, ומעדכן את פונקציית הערבול בהתאם לגודל החדש. מכיוון שאין מחיקות במבנה הנתונים שלו טבלת הערבול רק תגדל במידת הצורך ואינה תקטן.

כאשר נוסיף fleet חדש למבנה הוא יכנס ל UF בתור קבוצה חדשה בפני עצמה.

בכל צומת של fleet נשמור את המידע הבא:

- ID – מספר המזהה של הצי.
- numOfPirates – מספר הפיראטים בצי.
- numOfShips – מספר הספינות בצי.
- updateID – ה ID המעודכן, נפרט בהמשך על השימוש בשדה זה.
- extraToRank – שדה שבעזרתנו נדע מה הדרגה של כל פיראט, נפרט בהמשך.
- nextFleet – מצביע לצי הבא.

בכל צומת של pirate נשמור את המידע הבא:

- ID – מספר המזהה של הפיראט.
- Coin – כמה כסף יש לפיראט.
- Rank – הדרגה המקורית של הפיראט בעת הכנסתו למבנה.
- Fleetptr - מצביע לצי המקורי שאליו נכנס.

<u>השיטה oceans_t() :</u>

מאתחלת UF ריק, וטבלת ערבול ריקה ולכן **סיבוכיות זמן היא O(1) במקרה הגרוע.**

<u>השיטה ~oceans_t:</u>

השיטה עוברת על 2 הטבלאות ערבול, הטבלת ערבול של הפיראטים והטבלת ערבול של הציים(שמימשנו את הUF באמצעות טבלת הערבול של הציים) ומוחקת את האובייקטים שהוקצו דינמית המעבר על טבלאות הערבול מתבצעות כך: עובר תא תא בטבלת הערבול ובכל תא שיש בו רשימה מקושרת הוא עובר על כל הרשימה המקושרת.

לכן בסה"כ מעבר על טבלת הערבול של הפיראטים הוא O(n) ומעבר על טבלת הערבול של הציים הוא O(m) .

לכן סיבוכיות הזמן של פעולה זו היא O(m+n) במקרה הגרוע כנדרש.

<u>השיטה add_fleet(int fleetID):</u>

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נחפש האם קיים fleetID בטבלת הערבול שבתוך ה UF בעזרת השיטה find(i) של טבלת הערבול כפי שלמדנו בהרצאות, במידה ומצאנו ID כזה החזר FAILURE .

אחרת הקצאה דינמית אובייקט חדש של צי עם הערכים הבאים:

numOfPirates = 0, numOfShips = 1, ID = fleetID, updateID = fleetID, nextFleet=nullptr.

ובצע makeSet(ptr,fleetID) כאשר ptr הוא המצביע של הצי החדש שהקצתה דינמית(בכל צומת של טבלת הערבול יהיה את ה ID ובנוסף את המצביע לצי)

במידה ויש בעיות בהקצאת זיכרון החזר ALLOCATION_ERROR וסיים.

אחרת החזר SUCCESS וסיים.

ניתוח סיבוכיות זמן:

בדיקת תקינות קלט – O(1)

ביצוע find(index) בטבלת ערבול - O(1) בממוצע על הקלט.

makeSet שמשתמש ב insert של טבלת ערבול(כאשר הטבלת ערבול דינמית) – O(1) משוערך בממוצע על הקלט.(בתרגול למדנו שהכנסה למערך דינמי הוא O(1) משוערך)

לסיכום, סיבוכיות הזמן של פעולה זו היא O(1) משוערך בממוצע על הקלט.

השיטה add_pirate(int pirateID, int fleetID):

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נבצע פעולת find(pirateID) על טבלת הערבול של הפיראטים, במידה ומצאנו פיראט עם אותו מספר מזהה כמו שקיבלנו בארגומנט, נחזיר FAILURE .

כעת נבצע find(fleetID) על ה UF באופן רגיל כפי שלמדנו בהרצאות(מחזיר את השורש של הקבוצה) במידה וחזר nullptr סימן שלא נכנס למבנה הנתונים צי שהמספר מזהה שלו הוא fleetID ולכן נחזיר FAILURE . אחרת חזר לנו מצביע לשורש של הקבוצה נסמן מצביע זה ב ptr .

(*) **הערה:** המזהה של כל קבוצה(צי פעיל) פעילה יהיה השדה updateID של השורש של הקבוצה. ייתכן שבמהלך איחוד של קבוצות(ציים) מכיוון שהמזהה של הקבוצה החדשה תהיה המזהה של הקבוצה עם מספר הפיראטים הגדול יותר אך מכיוון שאנחנו מאחדים לפי גודל האיברים בקבוצה(מספר ספינות) ייתכן והקבוצה עם מספר הפיראטים הגודל יותר לא יהיה השורש של הקבוצה ולכן נעדכן בשורש של הקבוצה את השדה updateID למספר המזהה של הקבוצה עם מספר הפיראטים הגדול יותר, נפרט יותר בהמשך. הסבר נוסף ב(***) בעמוד 7.

לכן כדי לבדוק אם ה fleetID שקיבלנו הוא מזהה תקין שאפשר להוסיף לו פיראטים נבדוק האם $ptr \rightarrow updateID == fleetID$. במידה ולא החזר FAILURE וסיים.

אחרת הקצה דינמית אובייקט של פיראט עם הערכים הבאים:

ID = pirateID, coin = 0, rank = $ptr \rightarrow numOfPirate - extraToRank$, *fleetptr* = *ptr*

- נסביר בהמשך את המשמעות של השדה *extraToRank*(זה שדה עם אותו תפקיד לשדה הנוסף שהיה בתרגול בשאלת הארגזים שפתרנו בסעיף ב')

נסמן את המצביע לפיראט שהקצנו דינמית ב *piratePtr*

במידה ויש בעיות בהקצאת זיכרון החזר ALLOCATION_ERROR וסיים.

לאחר מכן נבצע *insert(pirateID,piratePtr)* בטבלת הערבול של הפיראטים(כל צומת בטבלת הערבול יחזיק *ID* ומצביע לפיראט). בנוסף נגדיל באחד את מספר הפיראטים במצביע *ptr* ונחזיר *SUCCESS* .

ניתוח סיבוכיות זמן:

בדיקת תקינות קלט – O(1)

ביצוע find(index) בטבלת ערבול - O(1) בממוצע על הקלט.

ביצוע find(index) בUF - O(log*m) משוערך בממוצע על הקלט.

ביצוע insert() בטבלת הערבול - O(1) בממוצע על הקלט.

לסיכום, סיבוכיות הזמן של שיטה זו היא O(log*m) משוערך בממוצע על הקלט.

<u>השיטה pay_pirate(int pirateID, int salary):</u>

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נבצע פעולת find(pirateID) על טבלת הערבול של הפיראטים, במידה ולא מצאנו פיראט עם אותו מספר מזהה כמו שקיבלנו בארגומנט, נחזיר FAILURE .

אחרת, הוסף לשדה coin של הפיראט את הסכום salary והחזר SUCCESS .

<u>ניתוח סיבוכיות זמן:</u>

בדיקת תקינות קלט – O(1)

ביצוע find(index) בטבלת ערבול - O(1) בממוצע על הקלט.

הוספה לשדה coin של הפיראט את הסכום salary – O(1) .

לסיכום, סיבוכיות הזמן של פעולה זו היא O(1) בממוצע על הקלט.

<u>השיטה num_ships_for_fleet(int fleetId) :</u>

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נבצע find(fleetId) על ה UF . במידה וחזר nullptr זה סימן לכך שמתחילת אתחול המבנה לא הכניסו ציי עם מספר מזהה fleetID ולכן החזר FAILURE וסיים.

נסמן את המצביע(של השורש של הקבוצה) שחזר מהפעולה find ב ptr.

כדי לבדוק האם fleetID הוא מזהה תקין לפי אותו הסבר של (*) שהסברנו למעלה נבדוק האם $ptr \rightarrow updateID == fleetID$, במידה ולא סימן שלא קיים ציי פעיל עם מזהה fleetID במבנה הנתונים ולכן החזר FAILURE וסיים.

אחרת, החזר את $ptr \rightarrow numOfShips$ ו SUCCESS וסיים.

<u>ניתוח סיבוכיות זמן:</u>

בדיקת תקינות קלט – O(1)

ביצוע find(index) בUF - O(log*m) משוערך בממוצע על הקלט.

בדיקה האם קיבלנו מזהה של צי פעיל ובמידה וכן החזרת מספר הספינות של הצי – O(1)

לסיכום, סיבוכיות הזמן של שיטה זו היא O(log*m) משוערך בממוצע על הקלט.

השיטה get_pirate_money(int pirateId):

נפעל באותו אופן בדיוק כמו בשיטה pay_pirate אך במקום לעדכן את השדה coin , נחזיר את ערך השדה הזה.

ניתוח סיבוכיות זמן:

בידיוק כמו השיטה pay_pirate לכן, סיבוכיות הזמן של שיטה זו היא

לסיכום, סיבוכיות הזמן של פעולה זו היא O(1) בממוצע על הקלט.

השיטה unite_fleets(int fleetId1,int fleetId2):

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נבצע find(fleetId1) על ה UF . במידה וחזר nullptr זה סימן לכך שמתחילת אתחול המבנה לא הכניסו ציי עם מספר מזהה fleetID1 ולכן החזר FAILURE וסיים.

נבצע find(fleetId2) על ה UF . במידה וחזר nullptr זה סימן לכך שמתחילת אתחול המבנה לא הכניסו ציי עם מספר מזהה fleetID2 ולכן החזר FAILURE וסיים.

(**)כדי לבדוק האם fleetID1 הוא מזהה תקין לפי אותו הסבר של (*) שהסברנו למעלה נבדוק האם $ptr \rightarrow updateID == fleetID$, במידה ולא סימן שלא קיים ציי פעיל עם מזהה fleetID במבנה הנתונים ולכן החזר FAILURE וסיים.

נבדוק עבור fleetID2 באותו האופן כפי שבדקנו עבור fleetID1 (סימנתי את זה ב(**)).

כעת לאחר שגילינו ששני המזהים שקיבלנו תקינים ומציינים ציים פעילים, השתמשנו באלגוריתם שלמדנו בתרגול בשאלת הארגזים, נרשום את ההקבלה.

השדה h(A) - הוא כמו השדה numOfPirates שנמצא בכל fleet.

השדה r(a) - הוא כמו השדה extraToRank שנמצא בכל fleet.

הגודל של קבוצה A כללית כלומר - |A| - הוא מספר הספינות(מספר הציים, הצמתים) שיש בקבוצה A ב UF .

וכמו שבתרגול שמו את הקבוצה B על הקבוצה A זה בדיוק מקביל לכך שבקבוצה A יש מספר פיראטים גדול יותר מאשר הקבוצה מספר הפיראטים בקבוצה B .

נשתמש בדיוק באותן נוסחאות ובכל פעם שנרצה לבצע איחוד נבדוק באיזה צי יש יותר פיראטים, במידה ולצי שהמזהה שלו הוא fleetID1 יש יותר מספר פיראטים נתייחס אליו בתור הקבוצה A (שמתואר בנוסחאות בפתרון התרגיל של הארגזים) ול fleetID2 בתור הקבוצה B .

אחרת במידה ולצי שהמזהה שלו הוא fleetID2 יש יותר מספר פיראטים נתייחס אליו בתור הקבוצה A (שמתואר בנוסחאות בפתרון התרגיל של הארגזים) ול fleetID1 בתור הקבוצה B.

(****)הסבר על עדכון שדה ה updateID:

אנחנו רוצים שהמזהה updateID של השורש החדש לאחר האיחוד יהיה המזהה של הצי עם מספר הפיראטים הגדול ביותר. בה"כ נניח שהקבוצה שיש לה יותר פיראטים היא fleetID1 ונסמנה ב A . והקבוצה שיש לה פחות פיראטים היא fleetID2 ונסמנה ב B .

נסתכל על 2 האפשריות:

אפשרות ראשונה: $|A| \geq |B|$:

במקרה זה לאחר האיחוד של 2 הקבוצות השורש של הקבוצה החדשה הוא השורש של הקבוצה *A* מכיוון שבקבוצה *A* יש יותר ספינות(ציים,איברים) ומכיוון שאנחנו מאחדים לפי גודל מספר איברים (קבוצה עם מספר איברים קטן נאחד לקבוצה עם מספר איברים גדול) נקבל שהשורש של הקבוצה *A* הוא השורש החדש. בנוסף מכיוון שלקבוצה *A* יש יותר פיראטים המזהה החדש לא ישתנה והוא ישאר ה *updateID* שהיה לשורש של *A* .

אפשרות שניה: $|A| < |B|$:

במקרה זה לאחר האיחוד של 2 הקבוצות השורש של הקבוצה החדשה הוא השורש של הקבוצה *B* מכיוון שבקבוצה *B* יש יותר ספינות(ציים,איברים) ומכיוון שאנחנו מאחדים לפי גודל מספר איברים (קבוצה עם מספר איברים קטן נאחד לקבוצה עם מספר איברים גדול) נקבל שהשורש של הקבוצה *B* הוא השורש החדש. בנוסף, מכיוון שלקבוצה *A* יש יותר פיראטים מאשר לקבוצה *B* נצטרך לעדכן את השדה *updateID* של השורש של הקבוצה *B* לערך *updateID* של השורש של הקבוצה *A*.

ובכך נשמור שבכל איחוד מספר המזהה של הקבוצה המאוחדת יהיה מספר המזהה של הקבוצה שבה היה יותר מספר פיראטים בזמן האיחוד.

בנוסף מימשנו את ה *find* של ה *UF* כפי שהוסבר באלגוריתם של הפתרון לשאלת הארגזים על מנת לשמור על הערכים הנכונים של השדות הנוספים של כל צי בעת כיווץ מסלולים.

ניתוח סיבוכיות זמן:

בדיקת תקינות קלט – O(1)

ביצענו 2 פעולות *find* ב *UF* - O(log*m) משוערך בממוצע על הקלט.

ביצענו פעולת *union* אחת - O(log*m) משוערך בממוצע על הקלט.

עדכון של השדות בשורש החדש – *O(1)*.

לסיכום, סיבוכיות זמן של פעולה זו היא O(log*m) משוערך בממוצע על הקלט.

השיטה *pirate_argument(int pirateid1, int pirateid2)*:

תחילה נבצע בדיקת תקינות על הקלט לפי הנדרש בתרגיל.(אם הקלט לא תקין החזר INVALID_INPUT)

נבצע פעולת find(pirateID1) על טבלת הערבול של הפיראטים, במידה ולא מצאנו פיראט עם אותו מספר מזהה כמו שקיבלנו בארגומנט, נחזיר FAILURE ונסיים.

נבצע פעולת find(pirateID2) על טבלת הערבול של הפיראטים, במידה ולא מצאנו פיראט עם אותו מספר מזהה כמו שקיבלנו בארגומנט, נחזיר FAILURE ונסיים .

אחרת, קיבלנו 2 מזהים תקינים של פיראטים. נסמן את המצביע שחזר מהפעולה find(pirateID1) ב ptr1 ואת המצביע שחזר מהפעולה find(pirateID2) ב ptr2 .

נסמן ב fleetID1 את ה-ID של הצי שהשדה fleetPtr של ptr1 מצביע אליו(יש לנו גישה ב O(1) ל ID הזה)

נסמן ב fleetID2 את ה-ID של הצי שהשדה fleetPtr של ptr2 מצביע אליו(יש לנו גישה ב O(1) ל ID הזה)

לאחר מכן נבדוק אם find(fleetID1) == find(fleetID2) (אם 2 הפיראטים באותו צי השורש של הקבוצה שלהם צריך להיות אותו שורש). אם תנאי זה לא מתקיים, סימן ש-2 הפיראטים לא נמצאים באותו צי ולכן נחזיר FAILURE ונסיים.

אחרת, שני הפיראטים נמצאים באותו צי ונמשיך בביצוע הפעולה.

נסמן ב rank1 את ערך השדה rank של ptr1(הפיראט עם הid1)

נסמן ב rank2 את ערך השדה rank של ptr2(הפיראט עם הid2)

לאחר מכן ניגש לfleetPtr של המצביע ptr1 , נאתחל משתנה temp = fleetPtr ונבצע לולאה באופן הבא(נלך מ temp עד לשורש):

- אם temp == nullptr סיים
- הוסף ל rank1 את $temp \rightarrow extraToRank$
- $\text{temp} = temp \rightarrow nextFleet$

לאחר ביצוע לולאה זאת נקבל את הדרגה האמיתית של pirateID1 ב rank1 .

נבצע פעולה זו באופן זהה עבור הפיראט השני, ובסוך נקבל את הדרגה האמיתית של

pirateID2 ב rank2.

(הפעולה של מציאת הדרגה מקבילה לשיטה HeightFromGround מהפתרון של שאלת הארגזים שפתרנו בתרגול).

לאחר שמצאנו את הדרגות של שני הפיראטים נבדוק למי יש דרגה יותר גבוהה.

בה"כ נניח ש rank1>rank2 :

לכן נוסיף לשדה coin של הפיראט שהמזהה שלו הוא pirateID2 את הסכום rank1-rank2.

ולפיראט שהמזהה שלו הוא pirateID1 נחסיר מהשדה coin את הסכום rank1-rank2 .

נחזיר בסיום SUCCESS .

<u>ניתוח סיבוכיות זמן:</u>

בדיקת תקינות קלט – O(1)

ביצוע find(index) בטבלת ערבול - O(1) בממוצע על הקלט.

ביצענו 2 פעולות *find* ב *UF* - O(log*m) משוערך בממוצע על הקלט.

חישוב *rank1/rank2* - O(log*m) משוערך.(****)

(****)הסבר: חישוב *rank1/rank2* נעשה בידיוק כמו שמתבצעת הפעולה *find* ב *UF* ותוך כדי מוסיפה ל *rank1/rank2* את השדה *extraToRank* בכל צומת.

בדיקות נוספות ועדכון השדה *coin* של הפיראטים – *O(1)*

לסיכום סיבוכיות הזמן היא O(log*m) משוערך בממוצע על הקלט.

<u>ניתוח סיבוכיות מקום של מבנה הנתונים:</u>

במבנה הנתונים שלנו יש 2 טבלאות ערבול (*UF* מורכב מטבלת ערבול)

נסמן ב *m* – את מספר הציים.

נסמן ב *n* - את מספר הפיראטים.

בכל רגע נתון גודל טבלת הערבול של הציים(טבלת הערבול שמרכיבה את ה *UF*) היא בסדר גודל של *2m*.

בכל רגע נתון גודל טבלת הערבול של הפיארטים הוא בסדר גודל של *2n*.

בנוסף לכל פיראט יש מספר קבוע של משתנים(שדות) נסמן קבוע זה ב *d*

ולכל ציי יש מספר קבוע של משתנים(שדות) נסמן קבוע זה ב *c*.

לכן סיבוכיות המקום של מבנה הנתונים הוא:

$O\big((2+c)m+(2+d)n\big)=O(m+n)$ כנדרש.