מבני נתונים 1

גליון (רטוב) 2 (החלק היבש)

מגישים:

שם	ת.ז	מייל
אילון הלוי	328137831	eilon.halevy@campus.technion.ac.il
מקסים גורביץ	322207671	maximgurwitz@campus.technion.ac.il

<u>דרישות מבנה הנתונים:</u>

oceans_t( )	אתחול מבנה נתונים ריק. ללא ספינות וללא פיראטים.	$O(1)$ במקרה הגרוע
virtual ~oceans_t( )	שחרור מבנה הנתונים וכל הזיכרון.	$O(n+m)$ במקרה הגרוע
StatusType add_fleet(int fleetId)	מוקם צי חדש בעל מזהה $fleetId$. בהתחלה לצי יש ספינה אחת ואין לו פיראטים.	$O(1)$ משוערך בממוצע על הקלט
StatusType add_pirate(int pirateId, int fleetId)	פיראט בעל מזהה $pirateId$ מגוייס לצי $fleetId$. הפיראט מתחיל בלי כסף ואם יש בצי x פיראטים הוא יקבל את הדרגה $x+1$.	$O(log^* m)$ משוערך בממוצע על הקלט
StatusType pay_pirate(int pirateId, int salary)	תשלום לפיראט עם המזהה $pirateId$ בסך $salary$ מטבעות.	$O(1)$ בממוצע על הקלט
output_t < int > num_ships_for_fleet(int fleetId)	מחזירה את מספר הספינות בצי $fleetId$.	$O(log^* m)$ משוערך בממוצע על הקלט
output_t < int > get_pirate_money(int pirateId)	מחזירה את מספר המטבעות של פיראט בעל מזהה $pirateId$	$O(1)$ בממוצע על הקלט
StatusType unite_fleets(int fleetId1, int fleetId2)	איחוד הציים בעלי המזהים $fleetId1$ ו- $fleetId2$. המזהה החדש שלהם הוא המזהה של הצי הגדול יותר <u>מבחינת מספר הפיראטים</u> (במקרה של שוויון המזהה החדש הוא $fleetId1$). מספר הספינות בצי המאוחד הוא סכום מספר הספינות בשני הציים, והדרגה של כל הפיראטים בצי הקטן יותר תגדל כמספר הפיראטים בצי הגדול יותר. לאחר האיחוד לא קיים יותר צי עם המזהה של הצי הקטן, ולא ניתן להוסיף צי חדש עם מזהה זה.	$O(log^* m)$ משוערך בממוצע על הקלט
StatusType pirate_argument(int pirateId1, int pirateId2)	וויכוח בין שני פיראטים. הפיראט שדרגתו גבוהה יותר ישלם לפיראט שדרגתו נמוכה יותר. סכום הכסף נקבע לפי הפרש הדרגות.	$O(log^* m))$ משוערך בממוצע על הקלט

***כאשר n הוא מספר הפיראטים ו-m הוא מספר הציים שהתווספו למערכת לאורך ריצת התכנית.**

בנוסף, ממבנה הנתונים נדרש לעמוד בהגבלה של סיבוכיות זיכרון $O(n+m)$ במבנה עצמו, ובכל הפעולות.

מימוש:

מימשנו את מבנה הנתונים באמצעות הרכבה של מבני נתונים קיימים וידועים, בעלי סיבוכיות זמן ריצה ומקום ידועים (מההרצאות), טבלאות ערבול ($HashTable$), ועצים הפוכים ($UpTree$).

במבנה הנתונים שלנו, יש סימון מבלבל (בעץ ההפוך) בין דרישות מבנה הנתונים לאופן המימוש שלנו, ולכן נחלק את הסבר הפתרון לשני חלקים:

- הסבר על ההרכבה של מבנה הנתונים הגנרי, של טבלאות ערבול ועץ הפוך
- הסבר על ההתאמה של מבנה הנתונים הגנרי לבעיה שהוצגה בשאלה

אופן ההרכבה של מבני הנתונים הגנרי:

הגדרות:

- לאיברים בעץ ההפוך נקרא קבוצות.
- לאחר איחוד קבוצות, נאמר כי הקבוצה הקטנה יותר אוחדה מלמטה (לאחר האיחוד, קבוצה זו מחזיקה במצביע לקבוצה האחרת, ונאמר שהקבוצה האחרת אוחדה מלמעלה).
- לאחר איחוד קבוצות, נאמר כי הקבוצה שאוחדה מלמטה מחזיקה מצביע לקבוצה מעל (הקבוצה שאוחדה מלמעלה).
- קבוצה שעד כה לא אוחדה מלמטה תקרא קבוצת על.
- לאיברים שמשתייכים לקבוצות, נקרא תאים.
- כאשר מוסיפים תא לקבוצה, נאמר כי התא משוייך לקבוצה זו במבנה הנתונים.

הערות:

השדות של קבוצה הם:

- מזהה (לא בהכרח מספר, אך יש לו מספר כאחד השדות)
- מצביע (ריק אם זו קבוצת על) לקבוצה מעל
- גודל הקבוצה (מספר הקבוצות הכולל תחתיה, כולל היא עצמה)
- דרגה (בדומה לעץ בינארי רגיל)

השדות של תא הם:

- מזהה (לא בהכרח מספר, אבל יש לו מספר כאחד השדות)
- מצביע לקבוצה אליה הוא משוייך

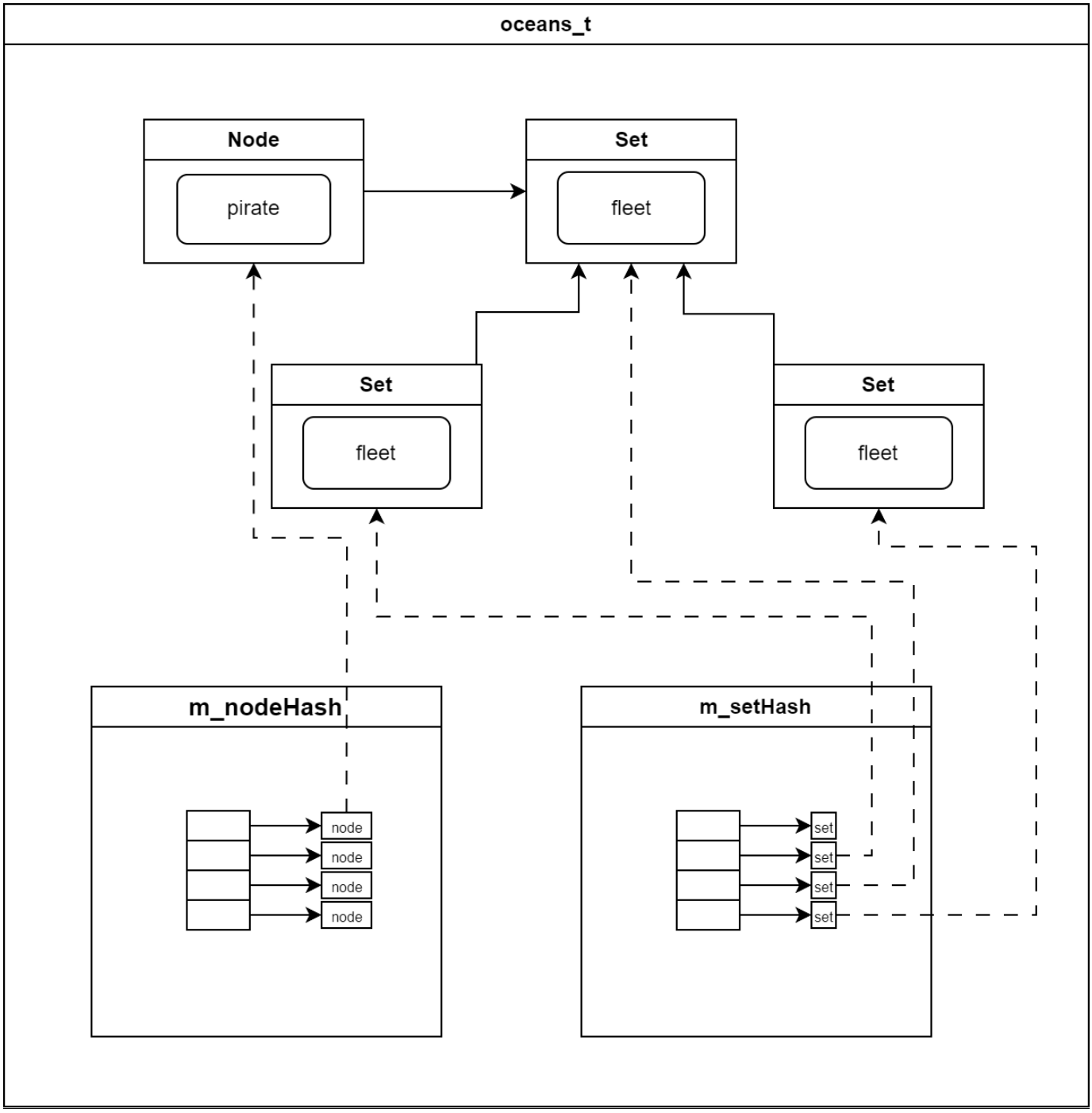
אבחנה: העץ ההפוך הוא בעצם עץ הפוך של קבוצות.

במבנה הנתונים שלנו 2 טבלאות ערבול:

- טבלת הקבוצות- טבלת ערבול, אשר ממפה (מלשון מיפוי) מזהה של קבוצה למצביע לאותה קבוצה בעץ ההפוך.
- טבלת התאים- טבלת ערבול, אשר ממפה (מלשון מיפוי) מזהה של התא למצביע לאותו התא.

רעיונית- נרצה ש"במקום קבוצות יהיו ציים, ובמקום תאים יהיו פיראטים"

שרטוט מבנה הנתונים:



<u>סיבוכיות הזיכרון של מבנה הנתונים:</u>

בהרצאות ראינו כי סיבוכיות הזיכרון הן $HashTable$ והן של עץ הפוך היא $O(n)$ כאשר n הוא גודל הקלט.

נניח כעת כי סיבוכיות הזיכרון של תא וגם של קבוצה היא $O(1)$ (נדרוש זאת בהמשך במעבר לפיראטים וציים) ונחסום את סיבוכיות הזיכרון במבנה הנתונים:

נסמן ב-m את מספר הקבוצות, וב-n את מספר התאים.

- טבלת ערבול של קבוצות- $O(m)$.
- טבלת ערבול של תאים- $O(n)$.
- עץ הפוך של קבוצות- $O(m)$. (העץ אמנם מבוסס על הזיכרון של טבלת ערבול הקבוצות אז אפשר לומר שאין לו תרומה לסיבוכיות הזיכרון)

לכן, סה"כ סיבוכיות הזיכרון של מבנה הנתונים היא $O(m) + O(n) \left(+ O(m)\right) = O(n+m)$.

לכן, כאשר נתאים קבוצות⟵ציים, תאים⟵פיראטים, כך שתתקיים הדרישה לעיל (סיבוכיות זיכרון $O(1)$), נקבל סיבוכיות זיכרון $O(n+m)$ כאשר m מספר הציים אשר נוספו למבנה הנתונים במהלך הריצה, n מספר הפיראטים, כנדרש.

נוכיח את סיבוכיות הזמן של הפעולות הבאות על מבנה הנתונים הגנרי:

init( )	אתחול מבנה נתונים ריק. קבוצות ותאים.	$O(1)$ במקרה הגרוע
~free( )	שחרור מבנה הנתונים וכל הזיכרון.	$O(n+m)$ במקרה הגרוע
StatusType makeSet(const S& setId)	יצירת קבוצה חדשה בעל מזהה setId (שאינו בהכרח מספר)	$O(1)$ משוערך בממוצע על הקלט
output_t < Set *> findSet(int setId)	בהינתן מזהה של קבוצה (מספר), יש למצוא את המצביע <u>לקבוצת העל</u> של הקבוצה בעל המזהה	$O(log^* m)$ משוערך בממוצע על הקלט
StatusType union2Sets(int set1Id, int set2Id)	בהינתן מזהים של שתי קבוצות (מספרים), לאחד את <u>קבוצות העל</u> של הקבוצות בעלי אותם מזהים.	$O(log^* m)$ משוערך בממוצע על הקלט
StatusType insertValue(int setId, const T& value)	שיוך תא ל<u>קבוצת העל</u> של הקבוצה בעל המזהה (המספרי) setId.	$O(log^* m)$ משוערך בממוצע על הקלט
output_t < Set *> fetchSetOf(int valueId)	בהינתן מזהה (מספרי) של תא, הפעולה מחזירה מצביע לקבוצה אליה התא משוייך.	$O(1)$ בממוצע על הקלט
output_t < T *> fetch(int valueId)	בהינתן מזהה (מספרי) של תא, הפעולה מחזירה את המצביע למזהה (לא מספרי) של אותו התא.	$O(1)$ בממוצע על הקלט

<u>הפעולה init( ):</u> אתחול טבלאות ערבול ריקות, סיבוכיות זמן $O(1)$ במקרה הגרוע.

<u>הפעולה ~free( ):</u>

שחרור הזיכרון של טבלת הערבול של התאים- $O(n)$ במקרה הגרוע, ושחרור הזיכרון של טבלת הערבול של הקבוצות, ובכל מחיקת קבוצה, בעצם מוחקים הצומת בעץ ההפוך של אותה הקבוצה, $O(m)$.

לכן, סה"כ סיבוכיות זמן $O(n+m)$ במקרה הגרוע.

הפעולה makeSet(const S& setId):

- יוצרים קבוצה חדשה (קבוצת על) בעלת המזהה setId, סיבוכיות זמן $O(1)$ במקרה הגרוע.
- מכניסים לטבלת הערבול את המזהה (המזהה המספרי של setId) עם המצביע לקבוצה שנוצרה.

כיוון שזו טבלת ערבול, מובטח לנו סיבוכיות זמן ריצה משוערכת בממוצע על הקלט $O(1)$.

הפעולה findSet(int setId):

- שולפים מתוך טבלת הערבול של הקבוצות את הקבוצה בעלת המזהה (המספרי) setId.
- מוצאים את המצביע של קבוצת העל באופן הבא: (נאתחל את סכום הדרגות ל-0)
  - בודקים האם המצביע של הקבוצה מעל הוא ריק-
    - אם כן, מחזירים את המצביע של הקבוצה הנוכחית
  - אחרת,
    - נוסיף לסכום הדרגות את הדרגה של הקבוצה הנוכחית.
    - עוברים לקבוצה מעל ומפעילים עליה את אותו אלגוריתם.
- כיווץ הענף בעץ:
  - ניקח את המצביע לקבוצה ששלפנו מטבלת הערבול, ונפעל כך:
    - נשנה את המצביע לקבוצה מעל כך שיצביע לקבוצת העל.
    - נחסיר מסכום הדרגות את הדרגה של הקבוצה הנוכחית.
    - נעדכן את הדרגה של הקבוצה הנוכחית להיות הדרגה + סכום הדרגות.
    - נפעיל את האלגוריתם על הקבוצה (שהייתה) מעל אלא אם הגענו לשורש.
- מחזירים את המצביע של קבוצת העל.

גישה לאיבר מתוך טבלת ערבול זו פעולה בסיבוכיות זמן ריצה משוערכת בממוצע על הקלט $O(1)$.

שני האלגוריתמים, של מציאת המצביע לקבוצת העל, ושל כיווץ הענף בעץ הם פעולות רקורסיביות, כאשר עומק הרקורסיה חסום ע"י גובה העץ ההפוך, וחישובי הביניים הם $O(1)$ במקרה הגרוע.

לכן (מובטח חסם משוערך בממוצע על הקלט $O(\log^* m)$ לגובה העץ ההפוך כאשר m זהו מספר הקבוצות שהוכנסו למבני הנתונים לאורך הריצה), סיבוכיות הזמן של פעולות אלו היא $O(\log^* m)$ משוערך בממוצע על הקלט.
סה"כ, סיבוכיות זמן הריצה של הפעולה היא $O(\log^* m) + O(1) = O(\log^* m)$ משוערך בממוצע על הקלט.

הפעולה union2Sets(int set1Id, int set2Id):

- מוצאים את קבוצות העל של הקבוצות בעלי המזהים (המספריים) הנתונים, באמצעות findSet.
- מבצעים איחוד על שתי קבוצות העל (הקבוצה הקטנה יותר מביניהן תאוחד מלמטה), $O(1)$.
- מחסירים מדרגת הקבוצה שאוחדה מלמטה את דרגת הקבוצה שאוחדה מלמעלה, סיבוכיות $O(1)$.
- נבצע עדכון דרגות לפי המזהים (כפי שיותאר בפעולה unite_fleets).

הפעולה findSet היא בסיבוכיות זמן משוערכת בממוצע על הקלט $O(\log^* m)$, ולכן גם סיבוכיות הפעולה הזו.

הפעולה insertValue(int setId, const T& value):

- מוצאים את קבוצת העל של הקבוצה בעלת המזהה setId, באמצעות findSet.
- משייכים לקבוצת העל את התא עם המזהה value, סיבוכיות $O(1)$.

סה"כ, סיבוכיות זמן ריצה מושערכת בממוצע על הקלט $O(\log^* m)$.

הפעולה fetchSetOf(int valueId):

- שולפים את המצביע לתא בעל המזהה (המספרי) valueId מתוך טבלת הערבול של התאים.
- מחזירים את המצביע של הקבוצה אליה משוייך התא (אחד מהשדות של התא), סיבוכיות $O(1)$.

בטבלת ערבול מובטחת סיבוכיות זמן ממוצעת על הקלט $O(1)$ על פעולת שליפת ערך לפי מפתח. לכן סיבוכיות הזמן של פעולה זו היא $O(1)$ בממוצע על הקלט.

הפעולה fetch(int valueId):

- שולפים את המצביע לתא בעל המזהה (המספרי) valueId מתוך טבלת הערבול של התאים.
- מחזירים את המצביע של מזהה התא (אחד השדות של התא), סיבוכיות $O(1)$.

בטבלת ערבול מובטחת סיבוכיות זמן ממוצעת על הקלט $O(1)$ על פעולת שליפת ערך לפי מפתח. לכן סיבוכיות הזמן של פעולה זו היא $O(1)$ בממוצע על הקלט.

הערות על הוכחת חישוב סיבוכיות הזמנים המשוערכים בממוצע על הקלט:

- הפעולה findSet מלבד הנגיעה בטבלת הערבול של הקבוצות, ועדכון הדרגות (שאינו משפיע על סיבוכיות זמן הריצה), זהה לפעולה find של עץ הפוך כפי שנלמדה בהרצאה, אשר מובילה לסיבוכיות זמן $O(\log^* m)$ משוערך בממוצע על הקלט (כאשר m הוא מספר ה-$Union$ים שבעץ ההפוך).

- לכן, כל אחת מהפעולות של מבנה הנתונים (מלבד שחרורו) מפעילה עד 2 פעולות על כל אחד ממבני הנתונים, טבלאות הערבול, והעץ ההפוך. לכן, סיבוכיות הזמן המשוערכת נשמרת.

התאמת מבנה הנתונים הגנרי לבעיה שהוצגה בשאלה

מבנה הנתונים הגנרי כמעט מהווה פתרון בעצמו לדרישות מבנה הנתונים. אבל יש בעיה שחייבים לפתור:

הקונפליקט בהתאמת המבנה לציים:

כאשר מאחדים שני ציים (לפי מבנה הנתונים הגנרי כפי שהוצג), הצי בעל מספר הספינות הגבוה יותר מאוחד מלמעלה (נבחין כי גודל הקבוצה הוא מספר הספינות של הציים בקבוצות העל), והוא הופך לקבוצת העל.

מצד שני-

מדרישות מבנה הנתונים, מזהה קבוצת העל "צריך" להיות של הצי עם מספר ה**פיראטים** הגדול יותר.

כיוון שאין קשר בין מספר הספינות תחת צי לבין מספר הפיראטים תחתיו, עלולים להיות מקרים בהם הסדר לפי מבנה הנתונים הגנרי הוא "הפוך" ממה שצריך להיות בדרישה ממבנה הנתונים.

אתגר נוסף (שהופך לקשה יותר בגלל הקונפליקט) הוא מתן דרגה לפיראט ועידכונה במהלך שינויים במבנה הנתונים (בלי פעולות מיותרות שמשפיעות על סיבוכיות הזמן).

הפתרון הוא כולו באופן מימוש המזהה של הקבוצה, כלומר אובייקט הצי (מימוש אובייקט הפיראט טריוויאלי).

תזכורת- פיראט יהיה **מזהה** של תא, וצי יהיה **מזהה** של קבוצה.

שדות הפיראט:

- מזהה (מספרי)
- אוצר (כמות מטבעות)
- דרגתו בצי אליו השתייך (דרגת בסיס)

שדות הצי:

- מזהה (מספרי)
- מספר ספינות תחת הצי
- מספר פיראטים תחת הצי
- מאזן הדרגה של הצי (הדרגה שצריך להוסיף לכל מי שנמצא תחת אותו צי), זו הדרגה של הקבוצה.
- מזהה דורס- מזהה הספינה שבפועל היא בעלת מספר הפיראטים הגבוה יותר (תוצאת האיחוד לפי הדרישות)

נותר רק לתאר את הפעולות ולהשתמש בחסמי סיבוכיות הזמן של מבנה הנתונים הגנרי שהוכחנו.

האובייקטים של הפיראט והצי עומדים בדרישת הסיבוכיות זיכרון $O(1)$. ולכן הפעולות של יצירת מבנה נתונים ריק ו-מחיקת מבנה הנתונים ושחרורו עומדים בסיבוכיות הזמן $O(1)$ ליצירת מבנה הנתונים, oceans_t, וסיבוכיות זמן $O(n+m)$ לפעולה ~oceans_t. (לפי הדרישות)

(מספר הציים = מספר הקבוצות, מספר הפיראטים = מספר התאים)

<u>הפעולה add_fleet(int fleetId):</u>

ניצור אובייקט צי עם המזהה fleetId, ספינה אחת תחתיו, 0 פיראטים תחתיו, מאזן דרגה 0.

נכניס את האובייקט הזה למבנה הנתונים הגנרי באמצעות makeSet.

סיבוכיות זמן הריצה היא $O(1)$ משוערך בממוצע על הקלט, לפי הדרישות.

<u>הפעולה add_pirate(int pirateId, int fleetId):</u>

- ניצור אובייקט פיראט עם המזהה pirateId, עם 0 מטבעות. $O(1)$.
- נשייך את הפיראט שייצרנו לצי של קבוצת העל של הקבוצה עבורה מזהה הצי הוא fleetId, באמצעות הפעולה insertValue.
- נגדיר את דרגת הבסיס של הפיראט להיות מספר הפיראטים בצי של קבוצת העל לפני הוספתו+1, פחות מאזן הדרגה של הצי, (כך כאשר ישאלו על דרגת הפיראט יוסיפו את מאזן הדרגה של הצי ונקבל בדיוק $x+1$, כמו בדרישות מבנה הנתונים.) $O(1)$.

לכן, סה"כ סיבוכיות זמן הריצה הוא $O(\log^* m)$ משוערך הממוצע על הקלט, לפי הדרישות.

<u>הפעולה pay_pirate(int pirateId, int salary):</u>

שולפים את הפיראט בעל המזהה (המספרי) pirateId מתוך מבנה הנתונים, באמצעות הפונקציה fetch.

משנים את כמות המטבעות של הפיראט ב-salary.

סיבוכיות זמן הריצה היא $O(1)$ בממוצע על הקלט, לפי הדרישות.

הפונקציה num_ships_for_fleet(int fleetId):

מוצאים את מצביע הצי של קבוצת העל של הקבוצה עבורה מזהה הצי הוא fleetId, באמצעות findSet.

מחזירים את מספר הספינות תחת צי זה (שזה גם מספר הקבוצות תחת קבוצת העל), $O(1)$.

לכן, סה"כ סיבוכיות זמן הריצה היא $O(\log^* m)$ משוערך בממוצע על הקלט, לפי הדרישות.

הפעולה get_pirate_money(int pirateId):

שולפים את הפיראט בעל המזהה (המספרי) pirateId מתוך מבנה הנתונים, באמצעות הפונקציה fetch.

מחזירים את כמות המטבעות של הפיראט, $O(1)$.

סיבוכיות זמן הריצה היא $O(1)$ בממוצע על הקלט, לפי הדרישות.

הפעולה unite_fleets(int fleetId1, int fleetId2):

באמצעות הפעולה union2Sets, נאחד את קבוצות העל של הקבוצות עבורן הציים בעלי המזהים הנתונים.

כדי לאפשר נכונות מבחינת דרישות מבנה הנתונים, נרצה לזכור את מזהה הצי שאמור להיות הצי בקבוצת העל מבחינת מבנה הנתונים, בקבוצת העל. לשם כך, לאחר איחוד לפי מספר הספינות (גדלי הקבוצות), לקבוצה שאוחדה מעל נשנה את המזהה הדורס של הצי שלה להיות המזהה הדורס של הצי של הקבוצה שאוחדה מלמטה תחתיה, אם בצי הקבוצה שאוחדה מלמטה יש יותר פיראטים, או שיש בדיוק אותו מספר פיראטים, אבל הסדר לפי דרישות מבני הנתונים הוא הפוך.

בנוסף, צריך לעדכן את דרגות כל הפיראטים בצי שבו יש פחות פיראטים לגדול במספר הפיראטים בצי האחר.

כמו בעדכון המזהה הדורס, נחלק למקרים:

אם הסדר נכון (הצי שאוחד מלמעלה אמור להיות מאוחד מלמעלה לפי דרישות מבנה הנתונים):

- נגדיל את מאזן הדרגה של הצי שאוחד מלמטה במספר הפיראטים של הצי שאוחד מלמעלה.

אחרת (הסדר הפוך, כלומר לכל הפיראטים בצי שאוחד מלמעלה ולא בצי שאוחד מלמטה צריך להגדיל דרגה),

- נגדיל את מאזן הדרגה של הצי שאוחד מלמעלה במספר הפיראטים של הצי שאוחד מלטה.
- כדי להחזיר את הדרגה הכוללת של הפיראטים שהיו בצי שאוחד מלמטה או תחתיו, נקטין את מאזן הדרגה של הצי שאוחד מלמטה במספר הפיראטים של הצי שאוחד מלטה.

לבסוף, נעדכן את מספר הספינות בצי שאוחד מלמעלה לסכום הספינות בשני הציים (של קבוצות העל), וכך גם נעדכן את מספר הפיראטים (להיות הסכום).
לכן, סה"כ סיבוכיות זמן הריצה היא $O(\log^* m)$ משוערך בממוצע על הקלט, לפי הדרישות.

הפעולה pirate_argument(int pirateId1, int pirateId2):

מוצאים את הפיראטים וקבוצות העל של הציים שלהם באמצעות הפונקציות findSet, fetch.

לפי תוצאת findSet, ניתן בקלות לבדוק האם הפיראטים תחת אותו צי- האם קבוצת העל שלהן משותפת, כלומר אותה תוצאת חיפוש. בנוסף, פעולה זו קיצצה את עומק העץ החל מהציים המקוריים של הפיראטים עד לקבוצת העל לאורך $O(1)$.

לכן, באמצעות fetchSetOf, ניתן לחשב את דרגת הפיראטים ב-$O(1)$.
(דרגת הבסיס, כתוצאה מ-fetch + סכימת דרגות המאזן של הציים בדרך, כאשר יש עד צי אחד מעל).

לבסוף, מציאת הפרשי דרגות הפיראטים, ושינוי שדות כמות המטבעות של הפיראטים בהתאם, $O(1)$ בממוצע על הקלט. לכן, סה"כ סיבוכיות הזמן היא $O(\log^* m)$ משוערך בממוצע על הקלט, לפי הדרישות.