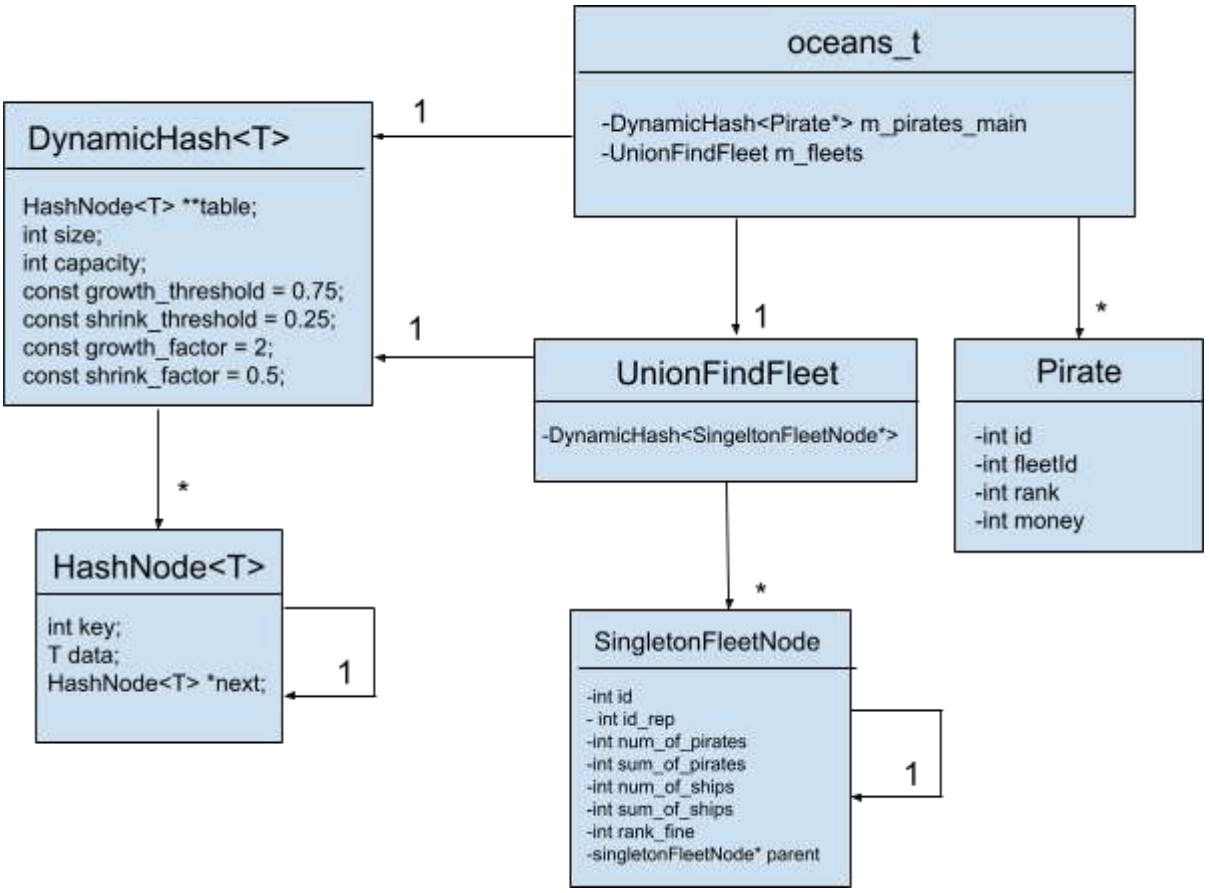
רפאל חדד - 207371709, מיכל סנה: 32381985

מבנה נתונים רטוב 2- חלק יבש

תחילה נתאר את מבנה הנתונים אותו מימשנו כדי לעמוד בדרישות התרגיל. לאחר מכן, נרחיב על מבנהו הפנימי ונסביר מדוע אנו עומדים בדרישות הסיבוכיות.

באופן כללי ביותר, מבנה הנתונים שלנו, oceans_t, מכיל שני מבני נתונים מרכזיים. מבנה נתונים מסוג טבלת האש דינאמית (DynamicHashTable - על מימושה נרחיב בהמשך), המכילה מצביעים לפיראטים, ומבנה מסוג UnionFind בו כל קבוצה זרה מייצגת "צי" (לפי הגדרת צי ברטוב-2).

תרשים UML המתאר את מבנה הנתונים שלנו (נרחיב ונעמיק בכל פרט ופרט בהמשך).

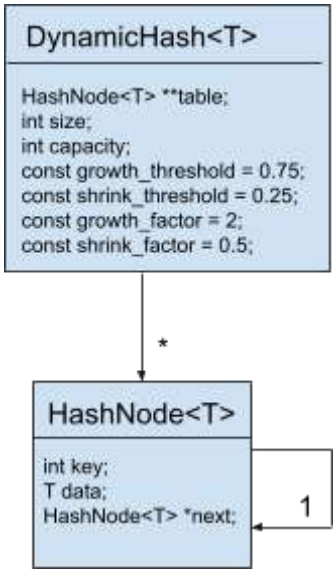


יבש - חלק 2 - מבני הנתונים בהם השתמשנו.

בחלק זה של היבש, נתאר ונסביר על מבני הנתונים בהם השתמשנו למימוש oceans_t, נרחיב על אופן מימושם, ונציג את סיבוכיות המקום והזמן שהם לוקחים. בחלק הבא, נתאר כיצד מימשנו את המתודות הנדרשות מ-oceans_t על ידי שימוש במבני הנתונים הנ"ל ונראה שהמתודות עומדות בדרישות הסיבוכיות הנדרשות מהתרגיל על ידי הסתמכות על תוצאות הסיבוכיות שנראה בחלק הנוכחי.

מבנה הנתונים DynamicHash<T>:

מבנה הנתונים DynamicHash<T>, הוא מבנה נתונים הממש את הפונקציונליות של טבלת האש כאשר המימוש הפנימי הוא דינאמי (כפי שראינו בתרגול על מימוש טבלת האש דינאמית). טבלת ההאש ממומשת בשיטת Chain hashing, כאשר מבנה הנתונים מכיל מערך של רשימות מקושרות (כל תא במערך הוא head של רשימה מקושרת, כלומר 'chain'). כל Node, בכל רשימה מקושרת, מכיל מפתח (int-יחודי), אובייקט גנרי מסוג T, וכן מצביע לאיבר הבא ברשימה. להלן תרשים UML המתאר את המבנה:



במבנה הנתונים, size ייצג את כמות האיברים שנכנסו לתוך טבלת האש (כלומר שקיימים בפועל, כולל כל חוליה בכל רשימה מקושרת). כלומר לפי "עגת" המצגות של ההרצאה והתרגול, size ייצג את n.

כמו כן, capacity ייצג את מספר התאים במערך המכילה את הרשימות המקושרות הדו-כיווניות (כלומר את מספר הרשימות המקושרות הדו-כיווניות). כלומר לפי עגת המצגות של ההרצאה והתרגול, capacity ייצג את m.

פונקציית הערבול שלנו הייתה הפונקציה הבאה: $h(i) = i \bmod (m)$ כאשר m הוא מספר התאים במערך המכיל את הרשימות המקושרות, כלומר **capacity.** ראינו בתרגול שפונקציית $mod\ m$ מפזרת באופן אחיד (את המפתחות) כאשר מניחים שהקלט אקראי, כלומר פונקציית $mod\ m$ מפזרת באופן אחיד **בממוצע על הקלט** (הנחת הפיזור האחיד הפשוט).

בכל שלב ושלב במבנה הנתונים שלנו (כלומר בין כל פעולה ופעולה), שמרנו על התנאי $m = O(n)$ (נראה כיצד בעוד שורות מספר), ולכן, כמו שראינו בהרצאה ובתרגול, בכל שלב ושלב $O(\alpha) = O(\frac{n}{m}) = O(1)$. תנאי זה גורר שעבור פעולת $search$, סיבוכיות הזמן של מתודת $search$ הינה $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט**.

מאוחר יותר נראה שצורת המימוש הדינאמית של טבלת ההאש שלנו גורמת לכך שיתקיים שסיבוכיות הזמן של פעולת ההכנסה לטבלה, $insert$, וכן סיבוכיות הזמן של פעולת החיפוש בטבלה, היא סיבוכיות זמן **משוערכת** של $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט**.

אנו מתייחסים רק לפעולות $insert, search$, שכן במבנה הנתונים שלנו, לא נעשה שימוש בפעולת $delete$, ולכן מתודה זו אינה רלוונטית לניתוח הסיבוכיות בתרגיל.

כפי שראינו בתרגול, המימוש הדינאמי של טבלת האש שלנו היה כלדהלן. בין כל פעולת הכנסה והכנסה, בדקנו האם

האם $\alpha = \frac{n}{m}$, כלומר פקטור העומס, גדול מ3 (כלומר, בממוצע יש בכל תא בטבלה מתוך m התאים, chain של 3 רשומות). במידה והתנאי התקיים, ביצענו פעולת $Rehash$ (בהקשר של chains), כלומר הגדלנו את גודל הטבלה (את m) ב-פי 2. לאחר ההקצאה של המקום החדש, ביצענו הכנסה של כל הרשומות שהיו בטבלה הישנה, לתוך הטבלה החדשה, על ידי פונצקיית ההאש החדשה, שכאמור הוגדרה על ידי $h(i) = i \bmod (m)$ כאשר m הוא גודל הטבלה החדשה. כאמור, ראינו בהרצאה שפעולת ה-$Rehash$ לוקחת (באופן טרוויאלי) $O(m)$ במקרה הגרוע. כעת, כמו שראינו בתרגול, מאחר ופונקציית ההאש שלנו היא $h(i) = i \bmod (m)$, תחת הנחת הפיזור האחיד ובהנחת קלט ממוצע, מתקיים שסיבוכיות ההכנסה $insert$, יחד עם פעולת ה-$Rehash$ כפי שהוגדרה לעיל, פועלת בטבלת ההאש הדינאמית שלנו בסיבוכיות זמן **משוערכת** של $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט**. מאחר ובמתודת $search$ שלנו אין אפשרות שהמאורע $Rehash$ תתרחש, ובמאחר ובכל שלב ושלב במבנה הנתונים שלנו אנו שומרים על התנאי $m = O(n)$ (הרי הטבלה דינאמית), מתקיים, כמו שראינו בהרצאה, שסיבוכיות הזמן של מתודת $search$ הינה $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט.**

לסיכום, בהתבסס על תוצאות מהתרגול וההרצאה ועל פי המימוש שלנו, באובייקט טבלת ההאש שלנו מתקיים:

- מתודת $insert$ פועלת בסיבוכיות זמן **משוערכת** של $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט.**
- מתודת $search$ פועלת בסיבוכיות הזמן של $O(\alpha) = O(\frac{n}{m}) = O(1)$ **בממוצע על הקלט.**
- בכל שלב ושלב, התנאי $m = O(n)$ נשמר.

6.4 סיכום - מימוש מילון כטבלת ערבול

6.4.1 סיבוכיות של הכנסה או הסרה מטבלת ערבול תוך שימוש ב-Chain Hashing

	ערבול אוניברסלי	ערבול "הגיל"
דינמית	$O(\alpha)$ בממוצע הסתברותי באופן משוערך	$O(\alpha)$ בממוצע על הקלט באופן משוערך
סטטית	$O(\alpha)$ בממוצע הסתברותי	$O(\alpha)$ בממוצע על הקלט

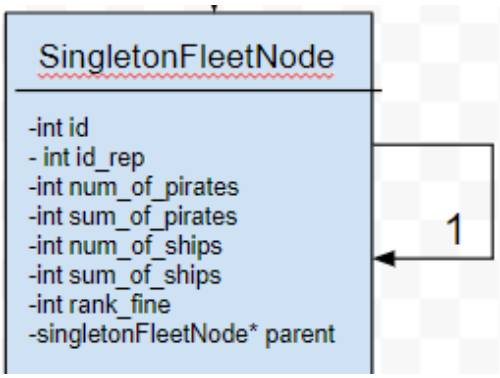
6.4.2 סיבוכיות של חיפוש בטבלת ערבול תוך שימוש ב-Chain Hashing

	ערבול אוניברסלי	ערבול "הגיל"
דינמית	$O(\alpha)$ בממוצע הסתברותי	$O(\alpha)$ בממוצע על הקלט
סטטית	$O(\alpha)$ בממוצע הסתברותי	$O(\alpha)$ בממוצע על הקלט

<u>הערה:</u> למרות ש$Rehash$ מוגדרת ישירות בתרגול עבור $Double\ hashing$, אפשר לדבר גם על $Rehash$ בהקשר של $Chain\ hashing$ כאשר בהקשר זה המתודה מקצה מחדש טבלה חדשה ומכניסה לשם את איברי הטבלה הישנה באמצעות פונקציית ההאש שהוגדרה מחדש עבור הטבלה החדשה כך שגם עבור בטבלה החדשה הפונקציה החדשה מפזרת באופן אחיד (אפשר לקחת פשוט mod m וזה יהיה בסדר, לפי הנלמד בהרצאה).

<u>מבנה הנתונים UnionFindFleet:</u>

על פי דרישות התרגיל, כל צי נוצר עם ספינה אחת ו-0 פיראטים ואי אפשר להוסיף ספינות. לכן, מבנה הנתונים בו נעשה שימוש על מנת לתאר את הציים נקרא UnionFindFleet, ומומש בתצורת UnionFind שהותאם לדרישות התרגיל הנתון. המימוש היה בצורה שהוצגה בהרצאה, עם עצים הפוכים וכיווץ מסלולים. מאחר ואיחוד הציים נעשה, על פי דרישות התרגיל, לפי מספר הפיראטים בצי, כאשר באיחוד הצי בעל מספר הפיראטים הגדול ביותר נעשה לצי המייצג את האיחוד, אך מבחינה מימושית היה צורך שבעת איחוד על הצומת (שורש של תת עץ הפוך) המייצג את הצי הקטן יותר מבחינת מספר **הספינות** להצביע על הצומת (שורש של תת-עץ הפוך) של הצי הגדול יותר מבחינת מספר **הספינות**, היה צורך להוסיף מידע נוסף שאפשר לנו לממש את שתי הדרישות.



כפי שניתן לראות בתרשים המייצג צי (צומת שכשורש יכול לייצג צי סינגלטוני או צי המורכב מאיחוד של ציים), בכל צי שמרנו מידע נוסף שאפשר לנו לממש את הפונקציונליות הנדרשה ממבנה הנתונים שלנו.

בכל צומת כזו, שמרנו את את ה-id והן את הid המייצג, id_rep, מזהה המקבל משמעות רק כאשר הוא נמצא בצומת ללא אב (שורש של תת-עץ). כמו כן, המידע הנוסף בכל צומת היה מספר הפיראטים ומספר הספינות המקומיים שלו, סכום הפיראטים וסכום הספינות שנמצאות תחת פיקודו

(בעל משמעות רק אם הוא שורש של עץ, כלומר אם אבא שלו הוא nullptr כלומר אם parent = nullptr) וכן משתנה שנקרא **קנס_דרגה**, כלומר rank_fine, שעזר לנו לממש את הפונקציונליות של חישוב הדרגה של פיראט בכל צי בו הוא נמצא, גם לאחר איחודים עם ציים אחרים, בסיבוכיות הנדרשת (דומה לתרגיל עם הארגזים בתרגול).
כעת, ביצירת צי חדשה, אנו יוצרים צומת בעל id יחודי עם אבא שהוא nullptr ועם id_rep ששווה לעצמו, כלומר המזהה המייצג של הצי הוא המזהה של הצי בעת אתחולו. כמו כן, ביצירת צי חדש, הצי נוצר עם 0 פיראטים ועם ספינה אחת, כפי שהתבקשנו בהוראות התרגיל. כמו כן, השדה של סכום הפיראטים וסכום ספינות בצי גם מאותחל להיות 0 ו-1 בהתאמה, ושדה קנס הדרגה של הצי מאותחל להיות 0.

כעת, מבנה הUnionFind שלנו מכיל טבלת האש של מצביעים לSingeltonFleetNodes כאלו.

<u>בעת איחוד, Union:</u> אנו מחפשים את הציים **ב(O(1 בממוצע** על הקלט. לאחר מכן אנו עולים עד השורש של כל node המוצבע על ידי המצביע בטבלת האש בעל הid שקיבלנו כקלט. אם בשורש הid המייצג (id_rep) הוא הוא הid שקיבלנו עבור שני הקלטים, האיחוד חוקי ואפשר לבצעו. עלייה עד השורש בהנחת כיווצים, כפי שראינו בהרצאה, לוקחת סיבוכיות **משוערכת** של log*m כאשר m הוא מספר הציים במערכת. לכן סיבוכיות הזמן היא O(log*m). בעת איחוד, בודקים איזה Node מכיל את מספר הספינות המצטבר הגדול ביותר. לאחר מכן, אנו גורמים לשורש המייצג את הצי הקטן יותר מבחינת סכום מספר הספינות שבו להצביע על השורש של הצי הגדול ביותר מבחינת מספר הספינות שבו. אנו עושים זאת תחת תנאי. הצי הגדול ביותר מבחינת מספר הספינות הופך להיות השורש אבל זה תלוי באם יש לו יותר פיראטים מהצי הקטן יותר מבחינת מספר הספינות. **אם יש לו יותר פיראטים מהצי בעל מספר הספינות הקטן יותר,** הid_rep שבו, כלומר הid של הצי אותו השורש מייצג, לא משתנה, שכן הוא היה הכי גדול מבחינת מספר הפיראטים ולכן הוא נשאר המזהה המייצג. אנו מעדכנים את מספר סכום הפיראטים וסכום הספינות בשורש החדש כנדרש ומוסיפים לקנס של הצי שכעת אוחד ומצביע על הצי הגדול יותר מבחינת מספר הספינות את סכום הפיראטים הישן של השורש החדש ומחסרים מהתוצאה את הקנס של השורש. **אם יש לו פחות פיראטים מהצי בעל מספר הספינות הקטן יותר,** השורש יהיה הצומת בעל מספר הספינות הגדול יותר אך מספר הפיראטים הקטן יותר. נעדכן את סכום הפיראטים והספינות שלו בהתאם. נעדכן את הid_rep שלו, כלומר את המזהה המייצג של הצי להיות המזהה של הצי הקטן יותר מבחינת מספר הספינות אך הגדול יותר מבחינת מספר הפיראטים. נשנה את הקנס של ההורה להיות סכום הפיראטים בצי הקטן יותר מבחינת מספר הספינות אך הגדול יותר מבחינת מספר הפיראטים וכדי לאזן את הסכום הזה בעת עלייה לשורש נחסיר את הקנס על הדרגה המעודכן של השורש (הצי בעל מספר הספינות הגדול יותר ומספר פיראטים קטן יותר) מהצומת בעל מספר הספינות הקטן יותר אך מספר הפיראטים הגדול ביותר. משחקי ההצבעות והחישובים האלו לוקחים O(1) סיבוכיות זמן. נשים לב ששמורת הUnionFind לפיה השורש המייצג את הקבוצה בעלת מספר האיברים הקטן יותר (צי בעל פחות ספינות) מצביע בעת איחוד על השורש המייצג קבוצה בעלת מספר איברים גדול יותר, נשמרת, ולכן **סה"כ נקבל סיבוכיות זמן משוערכת (יחד עם find ובהנחת כיווצים) של O(log*m) בממוצע.**

<u>לצורך דוגמא:</u> נניח כי יש לנו צי 1 וצי 2. **מקרה ראשון:** בצי 1 יש 10 פיראטים ו3 ספינות ובצי 2 יש 5 פיראטים ו2 ספינות. מספר הספינות (הצמתים) הקטן יותר הוא בצי 2, לכן נאחד את צי 2 עם צי 1 ונגדיר שהאבא של השורש של צי 2 הוא השורש של צי 1. גם מבחינת כמות הפיראטים צריך לאחד את צי 2 עם צי 1, לכן צריך להוסיף לצי 2 קנס כמספר הפיראטים שהיו בצי 1 לפי האיחוד. יכול להיות שלשורש של צי 1 גם היא קנס, לכן נעדכן את הקנס של צי 2 בצורה הבאה:
קנס2 + (מספר הפיראטים בצי1 − קנס1) = קנס2 .
מקרה שני: בצי 1 יש 10 פיראטים ו2 ספינות ובצי 2 יש 5 פיראטים ו3 ספינות. מספר הספינות (הצמתים) הקטן יותר הוא בצי 1, לכן נאחד את צי 1 עם צי 2 ונגדיר שהאבא של השורש של צי 1 הוא השורש של צי 2. לעומת זאת, מבחינת כמות הפיראטים צריך לאחד את צי 2 עם צי 1, לכן צריך להוסיף לצי 2 קנס כמספר הפיראטים שהיו בצי 1 לפי האיחוד. כדי למצוא קנס של צומת מסויים אנו סוכמים את כל הקנסות של כל הצמתים ממנו ועד השורש. מכיוון שהגדרנו שהשורש של צי 1 הוא צי 2, והוספנו קנס לצי 2, נרצה להחסיר את הקנס הזה מצי 1 (שכן בו היו יותר פיראטים לכן לא צריך לקבל קנס נוסף). לכן כדי שהקנס שהוספנו לצי 2 יתאפס עבור צי 1, נחסיר מהקנס של צי 1 את הקנס שהוספנו לצי 2. כלומר: קנס_2 + מספר_פיראטים_בצי_1 = קנס_2.

בעת פעולת חיפוש\Find, אנו מוצאים את המצביע לnode עם הid בטבלת ההאש שלנו ב(O(1 **בממוצע** על הקלט. לאחר מכן עולים עד השורש ומחזירים אותו כמצביע. הערה: המידע במצביע שהוא השורש נכון עבור כל הצי אבל המזהה של הצי בשורש הוא הid_rep. כלומר המצביע לשורש הוא המצביע לקבוצה בעלת המזהה id_rep. תוך ביצוע פעולת find אנו מבצעים גם כיווצי מסלולים כפי שהוצג בהרצאה (מחברים כל צומת על המסלול לשורש).
המידע בשורש מעודכן למעט המידע על חישובי הדרגות, ולכן רק אותו נצטרך לעדכן. בעת איטרציה על המסלול לשורש, אנו מחברים כל צומת על המסלול לשורש תוך תיקון הקנס דרגה שלו ביחס לקנס המצטבר שהיה בתחילת המסלול וחיסור מהקנס המצטבר את דרגת כל צומת על המסלול לפני התיקון (לפי האלגוריתם שהוצג בתרגול בדוגמא עם הארגזים). תיקון זה מתבצע בכל צומת על המסלול ב-(O(1 סיבוכיות זמן ולכן לא משפיע על סיבוכיות הזמן של פעולת הכיווצים ושומר על המידע לגבי הדרגות מעודכן בכל צומת. לכן, סה"כ, כפי שראינו בהרצאה,
סיבוכיות הזמן המשוערכת (יחד עם uniun) היא O(log*m) בממוצע.
בעת MakeSet, להוספת צי חדש למבנה הנתונים, ראשית נוודא שמזהה הצי שנרצה להוסיף לא היה קיים כלל, גם כצי פעיל וגם בתור צי שהיה פעיל ואוחד עם צי אחר. נבדוק בטבלת ההאש האם הצי קיים. ראינו שזה לוקח $O(1)$ בממוצע. אם הצי קיים נחזיר שגיאה כנדרש. אם לא קיים, ניצור Node חדש המייצג את הצי (שורש של עץ המייצג צי סינגלטוני). הNode החדש מאותחל עם id שקיבלנו בתור מזהה (בתנאי קלט תקין). הid_rep שלו, כלומר הid המייצג, (במידה והNode הנוכחי הוא שורש) מה המייצג של הצי, מאותחל להיות גם הid שקיבלנו כקלט. האבא של הצומת מאותחל להיות nullptr (מה שמסמן את הצומת כשורש ולכן כצי בעל המזהה id_rep. הNode מאותחל להיות עם ספינה אחת ו-0 פיראטים, ובעל סכום מצטבר של 1 ספינות ו-0 פיראטים. בנוסף, נאתחל את שדה קנס הדרגה שלו להיות 0. כל זה זה קורה בסיבוכיות זמן (O(1. נוסיף את הצומת לטבלת האש עם insert, ראינו שזה מתרחש בסיבוכיות זמן **משוערכת** של (O(1 בממוצע. **לכן,** סה"כ, סיבוכיות הזמן של המתודה היא סיבוכיות זמן **משוערכת** של (O(1 **בממוצע**.

לסיכום, בהתבסס על תוצאות מהתרגול וההרצאה, על פי מימוש טבלת האש הדינאמית שלנו ועל פי המימוש לעיל, באובייקט Unionfind שלנו מתקיים:

- מתודות $Find, Union$ פועלות בסיבוכיות זמן **משוערכת** של $log\ *\ (m)$ **בממוצע על הקלט.**
- מתודת $Makeset$ פועלת בסיבוכיות זמן של $O(1)$ **בממוצע על הקלט.**

כעת, נפרט על המתודות אותן התבקשנו לממש:

סיבוכיות המקום: במערכת קיימת טבלת האש דינאמית המכילה את המידע המייצג את n הפיראטים. בכל שלב ושלב, בטבלת האש יש לכל היותר $3n = O(n)$ תאים (כאשר הטבלה נמלאת, היא מקצה עצמה מחדש בגודל משולש). כמו כן, במערכת קיים אובייקט של UnionFind המחזיק רק טבלת האש בעלת מצביעים לm ה-nodes המייצגים את m הקבוצות\איברי הקבוצות (כלומר, הציים), בכל שלב ושלב הטבלה מכילה לכל היותר $3m = O(m)$ תאים. סה"כ בכל שלב ושלב במערכת סיבוכיות הזיכרון היא $O(m + n)$, כנדרש.

oceans_t(): במערכת טבלת האש של פיראטים המאותחלת דיפולטיבית עם $11 = O(1)$ תאים ריקים ולכן ב-$O(1)$ סיבוכיות זמן. במערכת אובייקט של אובייקט של UnionFind המחזיק רק טבלת האש של מצביעים לציים ושמאותחל דיפולטיבית עם $11 = O(1)$ תאים ריקים ולכן ב-$O(1)$ סיבוכיות זמן. לכן סה"כ $O(1)$ סיבוכיות זמן.

virtual ~oceans_t():

טבלת האש של הפיראטים מכילה מצביעים ל-n פיראטים ובכל רגע ורגע בריצת התוכנית, בפרט בסופה, בעלת לכל היותר לכל היותר $3n = O(n)$ תאים. אובייקט הUnionFind נהרס דיפולטיבית על ידי הריסת טבלת ההאש של הציים המכילה מצביעים ל nodes שמייצגים את הקבוצות או איברי הקבוצות (הציים). בכל רגע ורגע בריצת התוכנית, בפרט בסופה, בעלת לכל היותר לכל היותר $3m = O(m)$ תאים. כאשר המערכת נהרסת, נקרא ההורס שהוגדר עבור טבלת

ההאש שהורס את הטבלה בסיבוכיות זמן ליניארית למספר האיברים שהוכנסו אליו (איטרציית שחרור על k תאיו שמכילים סה"כ e איברים וגם מתקיים $k = O(e)$ ולכן סה"כ איטרציה על $O(e) + O(k) = O(e)$ איברים - מספר האיברים בטבלה). בנוסף, מסתמכים על שחרור אוטומטי של זיכרון של המצביעים החכמים (לכל היותר $O(n)$+$O(m)$ כאלו). לכן **סה"כ סיבוכיות זמן ההריסת ליניארית במספר הפיראטים והציים, כלומר** $O(m + n)$**.**

StatusType add_fleet(int fleetId):

לאחר בדיקות קלט, נקרא למתודת makeSet שלנו (לאחר בדיקות תקינות קלט). ראינו לעיל שמתודת $Makeset$ פועלת בסיבוכיות זמן של $O(1)$ **בממוצע על הקלט.**

StatusType add_pirate(int pirateId, int fleetId):

ראשית נוודא שהמזהה של הפיראט שנרצה להוסיף עוד לא קיים בטבלת ההאש של הפיראטים. זה קורה בסיבוכיות זמן של **O(1) בממוצע.** לאחר מכן, נבדוק האם הצי אליו רוצים להוסיף את הפיראט קיים בטבלת ההאש של הציים. זה קורה גם כן בסיבוכיות זמן של O(1) בממוצע. נקבל בעזרת מתודת Find שפועלת בסיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט** את השורש של העץ ההפוך אליו שייך ה-Node שאת ה-id שלו קיבלנו כקלט. אם המזהה המייצג בשורש, כלומר הid_rep שבו, שווה לid שקיבלנו כקלט, זהו צי פעיל ואפשר להוסיף לו פיראט. אחרת, נחזיר שגיאה. כעת, ניצור אובייקט של פיראט חדש ב-O(1) עם הid שקיבלנו כקלט. נגדיר לו שהוא שייך לצי בעל המזהה של הid_rep כלומר המזהה המייצג של הצי בשורש של הצי. ניתן לו את הדרגה של מספר הפיראטים בכל הצי (מצאנו את השורש כבר מקודם, אז יש לנו מבציע אליו ולכן זה קורה ב-O(1)). נוסיף לסכום הפיראטים בשורש פיראט אחד. כעת, מהגדרת המבנה שלנו, מאחר ומהצי המייצג עד השורש סכום הדרגות הוא 0 (כי הוא המייצג ולכן היה הגדול ביותר בסדרת האיחודים) אז הפיראט שהוסף כעת עולה עד השורש וסוכם את קנסות הדרגות (0) ולכן הדרגה שלו היא הדרגה שהוא קיבל בעת אתחול + 0 כלומר דרגתו היא הדרגה שהיינו רוצים, מספר הפיראטים בצי ועוד אחד. כל זה קורה, סה"כ, בסיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט +** O(1) **בממוצע +** O(1) ולכן סה"כ קורה בסיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט.** <u>הערה:</u> נשים לב שהוספת פיראט לצי סינגלטוני עם ספינה אחת מוגדר היטב כמקרה פרטי של מתודת הוספת הפיראט לעיל שכן הid_rep שלו שווה לid שלו והקנס שלו הוא 0 והיינו רוצים לתת לפיראט חדש שמגיע אליו את הדרגה המקומית של סכום הפיראטים בו + 1.

StatusType pay_pirate(int pirateId, int salary):

ראשית נוודא שהמזהה של הפיראט שנרצה לשלם לו קיים בטבלת הhash של הפיראטים (m_pirates_main) בסיבוכיות זמן ממוצעת של $O(1)$. כם כן, נחפשו ($O(1)$ בממוצע) לאחר שמצאנו אותו (מצביע), נעדכן את שדה הכסף שלו בזמן של $O(1)$. סך הכל סיבוכיות הזמן היא $O(1)$ בממוצע על הקלט, כנדרש.

output_t<int> num_ships_for_fleet(int fleetId):

ראשית נוודא שהמזהה של הצי קיים במבנה שלנו ונמצא אותו. כדי לבדוק זאת, נחפש את המזהה בטבלת הhash של הציים בסיבוכיות זמן בממוצע של $O(1)$. לאחר מכן, נשתמש במתודת Find כדי למצוא את השורש שלו במבנה **בסיבוכיות זמן משוערכת של** $O(log * m)$ **בממוצע**. לאחר מכן, נבדוק האם המזהה שקילבנו כקלט שווה למזהה המייצג בשורש, כלומר לid_rep בשורש. אם לא, נחזיר שגיאה. אם כן, ניגש לשדה השומר את כמות הספינות בצי (בצי שמצאנו) בסיבוכיות זמן של $O(1)$ ונחזיר אותו. מספר חסום על ידי קבוע של פעולות שכל אחת מהן בסיבוכיות זמן של $O(1)$ בממוצע ופעולה אחת בסיבוכיות זמן משוערכת של $O(log * m)$ בממוצע על הקלט לכן סיבוכיות הזמן המשוערכת של הפעולה היא $O(log * m)$ בממוצע על הקלט, כנדרש.

output_t<int> get_pirate_money(int pirateId):

ראשית נוודא שהמזהה של הפיראט שנרצה לגשת לסכום הכסף שלו קיים בטבלת הhash של הפיראטים (m_pirates_main) בסיבוכיות זמן $O(1)$ בממוצע. לאחר מכן נחזיר אותו ב$O(1)$ בממוצע. לאחר מכן נחזיר את שדה הכסף שלו בסיבוכיות זמן של $O(1)$. סה"כ סיבוכיות זמן היא $O(1)$ בממוצע על הקלט, כנדרש.

StatusType unite_fleets(int fleetId1, int fleetId2):

נבצע, לאחר בדיקות תקינות קלט ב(O(1 + בדיקות תקינות וקיום האובייקטים ב(O(1 בממוצע, את מתודת $Union$, שהראינו לעיל פועלת בסיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט.** (בדיקות תקינות קלט\קיום\חוקיות מתבצעות כבר וגם ב-$Union$). נחזיר שגיאות כמוגדר. סה"כ סיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט.**

StatusType pirate_argument(int pirateId1, int pirateId2):

ראשית נוודא שמזהי הפיראטים קיימים בטבלת הhash של הפיראטים (m_pirates_main) בסיבוכיות זמן של $O(1)$ בממוצע (כפי שהסברנו) ולאחר מכן נוציא אותם ב- $O(1)$ בממוצע. לאחר מכן, ניגש לשדות המכילים את מזהי הציים אליהם הפיראטים הוספו בעת אתחולם (בסיבוכיות זמן של $O(1)$). נבצע את מתודת find בסיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט,** עבור כל מזהה צי של כל פיראט, וכך נקבל את השורשים של תתי-העץ אליהם שייכים ציי הפיראטים. נבדוק אם השורשים האלו שווים ב-(O(1 (מספיק להשוות id שכן id לעולם לא משתנה, רק id_rep משתנה). אם כן, זה אומר ששתי הפיראטים נמצאים באותו צי (בין אם מלכתחילה או בין אם נמצאים בשתי ציים שאוחדו אל תוך צי גדול יותר). אם לא, נחזיר שגיאה. אם הפיראטים נמצאים באותו צי, נחשב כעת את הדרגה האבסולוטית שלהם בתוך הצי הגדול: לכל פיראט, ניגש לצי אליו הוסף לראשונה בעת אתחולו על ידי גישה לטבלת ההאש ב$O(1)$ בממוצע. לאחר מכן, נעלה עד השורש בסיבוכיות זמן **משוערכת** (תחת הנחת המימוש שלנו + כיווצים (אותם מימשנו) וכמו שראינו בהרצאה האורך **המשוערך** של מסלול מצומת אל השורש הוא $log * (m)$) של $log * (m)$ תוך סכימת ($O(1)$ בכל צומת על המסלול) השדות של קנסות הדרגות של כל צומת במסלול עד השורש). סכום זה תוכנן במימוש שלנו להיות הסכום שיש להוסיף לדרגה המקומית שפיראט קיבל בעת אתחולו והוספתו לצי כדי לקבל את הדרגה האבוסולוטית שלו בצי כולל (במקרה סינגלטוני, פשוט 0, במקרה שהצי של הפיראט אוחד, הסברנו בחלק הקודם כיצד משנים את קנסות הדרגות בצומת המאוחדת והמתאחדת כדי לקבל את הסכום הרצוי). לכן, מחשבים עבור כל פיראט את דרגתו האבסולוטית בסה"כ סיבוכיות זמן **משוערכת** של $log * (m)$ **בממוצע על הקלט.** נשווה בין הדרגות ואז נוסיף ונחסיר כסף מהפיראטים כפי שנדרשנו בהגדרות תרגיל הבית על ידי גישה לשדות הכסף שלהם בסיבוכיות זמן של $O(1)$. סך הכל ביצענו מספר חסום על ידי קבוע של פעולות שכל אחת מהן בסיבוכיות זמן של $O(1)$, מספר חסום על ידי קבוע של פעולות שכל אחת מהן בסיבוכיות זמן ממוצעת של $O(1)$ ומספר חסום של פעולות שכל אחת מהן היא בסיבוכיות זמן משוערכת של $O(log * m)$ בממוצע על הקלט, ולכן סך הכל סיבוכיות הזמן **המשוערכת** היא $O(log * m)$ בממוצע על הקלט, כנדרש.