יבש תרגיל רטוב 2

<u>הקדמה:</u>
השתמשנו בתרגיל הזה במבני נתונים הבאים:
1) טבלת ערבול – Hash Table.
2) Union/Find.

מאפיינים של טבלת ערבול – Hash Table:

1. **זוגות מפתח-ערך**: מאחסנת נתונים בזוגות של מפתח וערך, כאשר כל מפתח הוא ייחודי.
2. **פונקציית ערבול**: משתמשים בפונקציית ערבול כדי לחשב אינדקס למפתח, אשר קובע היכן יאוחסן הערך.
3. **איתור יעיל** :מספקת זמן ריצה ממוצע של O(1) לחיפוש, הוספה ומחיקה.
4. **התמודדות עם התנגשויות**: משתמשת בטכניקות כמו שרשור או כתובת פתוחה(Open Addressing) כדי להתמודד עם התנגשויות כאשר מספר מפתחות מקבלים את אותו אינדקס.
5. **גודל קבוע ושינוי גודל** :בדרך כלל יש גודל קבוע, אך במקרים רבים ניתן לשנות את הגודל באופן דינמי (לדוגמה, להכפיל את הגודל) כאשר היחס בין מספר האלמנטים לגודל הטבלה עולה על סף מסוים. פעולת שינוי הגודל אינה משפיעה על הסיבוכיות הממוצעת של הוספה.

סיבוכיות זמן של פעולות בטבלת ערבול:

- **הוספה**: O(1) משוערך בממוצע על הקלט ,וזה בגלל שתתכן אפשרות שינוי גודל הטבלה, ו-O(n) במקרה הגרוע.
- **חיפוש**: O(1) בממוצע על הקלט ו- O(n) במקרה הגרוע (כאשר יש התנגשויות רבות).
- **מחיקה**: O(1) משוערך בממוצע על הקלט ,וזה בגלל שתתכן אפשרות שינוי גודל הטבלה ו- O(n) במקרה הגרוע (כאשר יש התנגשויות רבות).

<u>הסבר מימוש:</u> בתרגיל הזה השתמשנו ב-chaining בכדי להתמודד עם התנגשויות בטבלה, ובנוסף שמרנו את הרשימות במערך שגודלו L=O(k) כאשר k מסמן את כמות האיברים במערך ו-L הוא גודל המערך הזה בכדי להבטיח שפקטור העומס מקיים $\alpha = O(1)$, פונקציית הערבול שהגדרנו היא ניתנת ע"י:

$$f(Id) = Id(\text{mod } L)$$

ובנוסף שינוי גודל המערך הוא מתקיים כאשר L=k ואז מגדילים L פי 2, המימוש הזה לפי ההרצאה והתרגול מבטיח קבלת סיבוכיות זמן לכל פעולה כפי שמפורט לעיל.

מאפיינים של קבוצות מאוחדות(Union-Find):

1. **ייצוג קבוצות** Union-Find :הוא מבנה נתונים שמייצג קבוצות של אלמנטים המאפשר לבדוק האם שני אלמנטים שייכים לאותה קבוצה.
2. **איחוד(Union):** פונקציה שמחברת בין שתי קבוצות נפרדות לקבוצה אחת.
3. **חיפוש(Find):** פונקציה שמחזירה את המייצג הייחודי (או השורש) של הקבוצה שהאלמנט שייך אליה. משתמשים בטכניקת אופטימיזציה "כיווץ מסלול" כדי לזרז את פעולת החיפוש בפעמים הבאות.
4. **מייצגי קבוצות** :כל קבוצה מיוצגת על ידי אלמנט אחד, והאלמנטים בקבוצה מקושרים דרך מבנה נתונים של עץ הפוך.

סיבוכיות זמן של פעולות Union-Find:

- **איחוד(Union)**: עולה O(log*n) משוערך בממוצע על הקלט כאשר n הוא מספר האיברים במבנה .
- **חיפוש(Find):** עולה O(log*n) משוערך בממוצע על הקלט כאשר n הוא מספר האיברים במבנה כאשר log*n היא פונקציה "איטית" כפי שלמדנו בהרצאה ותרגול, וזה במשוערך בממוצע על הקלט מכיוון שהחסם שמקבלים הוא הממוצע של ביצוע סדרת פעולות של איחוד וחיפוש שמבצעים בו כיווץ מסלולים, ובנוסף הפונקציה עולה O(logn) זמן במקרה הגרוע.
- **יצירת קבוצה חדשה**: עולה O(1) משוערך בממוצע על הקלט עבור הוספת איבר בודד כפי שלמדנו בהרצאה .

<u>הסבר למימוש</u>:
בתרגיל הזה השתמשנו במבנה Union-Find רק עבור הציים, ולכן הגדרנו טבלת ערבול של הציים בתור משתנה פרטי למחלקה בכדי להשתמש בה לשמור את הציים שהם בעצם צמתים בעץ הפוך, כל צומת יש לו מצביע שמצביע לאבא שלו, ועשינו המימוש בצורה דומה לשאלת הארגזים מהתרגול שכל FLEET שמור בו RANK ושסכום הRANK מהFLEET עד האב העליון הוא הRANK שמתחיל ממנו ה RANK של הפיראטים הנמצאים ב FLEET הזה וב **Union מאחדים לפי גודל כדי לשמור על סיבוכיות** log*n ומעדכנים הRANK בהתאם ומזהה של קבוצה הזאת הוא לא בהכרח הראש של העץ (האב העליון) אלא מעכנים אותו בכל UNION להיות ה FLEET שיש לו הכי פיראטים משני הציים שעשינו להם איחוד והסיבוכיות של **Union** היא log*n כי השתמשנו ב find כדי למצוא האב העליון של הקבוצה כי הFLEET שהוא בעל המזהה של הקבוצה הוא הFLEET שיש לו מספר פיראטים הכי גדול אבל הוא לא בהכרח האב העליון לכן הסיבוכיות של **Union** היא log*n , ובכך לפי הרצאה ותרגול נקבל את הסיבוכיות הרצויה לכל פעולה מהנ"ל.

הגדרנו בתרגיל הזה טבלת ערבול לפיראטים ומבנה Union-Find עבור הציים,
ובנוסף שמרנו במשתנה את מספר הדרגות שצריכים להוסיף לפיראטים באותו צי במקרה של איחוד עם צי אחר שהוא גדול מהצי הנוכחי מבחינת מספר פיראטים.

מימוש הפונקציות וניתוח סיבוכיות:

oceans_t():

לפי הרצאה אתחול מבני הנתונים ריקים עולה O(1) במקרה הגרוע.

~oceans_t():

שחרור מבני הנתונים מתקיים ע"י שעוברים על כל צומת בכל רשימה מקושרת בטבלת הערבול של הפיראטים ומשחררים את הזיכרון שנמצא בה וזה עולה O(n) במקרה הגרוע ואז מפעילים ההורס הדיפולטי של מחלקת Union-Find שהוא בעצמו קורה להורס מחלקת טבלת הערבול בכדי לשחרר את הזיכרון שנשמר בטלת הערבול של הציים, וזה כמובן עולה O(m) במקרה הגרוע, ולכן סה"כ נקבל שסיבוכיות הזמן של הפונקציה היא O(n+m) במקרה הגרוע.

add_fleet(fleetId):

נבצע זאת ע"י בדיקת קיום צי עם המזהה המוזן וזה עולה O(1) בממוצע על הקלט לפי תיאור חיפוש בטבלת הערבול של הציים הנ"ל, ואם הצי לא נמצא נצור צי חדש עם המזהה הנתון ומזינים אותו למבנה Union-Find של הציים, וזה לפי התיאור לעיל מתקיים בסיבוכיות זמן O(1) משוערך בממוצע על הקלט, ולכן סה"כ נקבל שהפונקציה הזאת עולה O(1) משוערך בממוצע על הקלט.

add_pirate(pirateId, fleetId):

נבצע זאת ע"י בדיקת קיום פיראט וצי שהוא מזהה קבוצה(בדיקה שהוא נמצא בhash) עם אותם מזהים וזה עולה O(1) בממוצע על הקלט עבור הפיראט לפי תיאור חיפוש בטבלת הערבול הנ"ל, ומציאת הצי שצריך לתת לפיראט מתקיים שמכיוון שצריך לתת לו הצי שהוא שורש של אחד מהעצים ההפוכים ולא בעל מספר הפיראטים הכי גדול וזה עולה O(log*m) משוערך בממוצע על הקלט.
אם הפיראט לא נמצא והצי אכן קיים נצור פיראט חדש עם המזהה הנתון ודרגה מותאמת למספר הפיראטים החדש בצי ולמשתנה של הצי עצמו שמבטא את מספר הדרגות שצריכים להוסיף לכל פיראט (נתן לפיראט הRANK (numOfPirates+1-headFleetRank)), ומזינים אותו לטבלת הערבול של הפיראטים שזה לפי התיאור לעיל מתקיים בסיבוכיות זמן O(1) משוערך בממוצע על הקלט, ולכן סה"כ נקבל שהפונקציה הזאת עולה O(log*m) משוערך בממוצע על הקלט.

pay_pirate(pirateId, salary):

נבצע זאת ע"י בדיקת קיום פיראט עם אותו מזהה וזה עולה O(1) בממוצע על הקלט לפי תיאור חיפוש בטבלת הערבול הנ"ל, ואם כן נמצא נוסיף את הכסף לכסף הפיראט שנשמר

במשתנה לפיראט שזה עולה O(1) במקרה הגרוע, ולכן סה"כ הפונקציה הזאת עולה O(1) בממוצע על הקלט.

:num_ships_for_fleet(fleetId)

ראשית נבדוק את קיום הצי, ונבדוק זאת דרך שנבדה שהצי הזה הוא מזהה של קבוצה ב O(1) בממוצע על הקלט נמצא את האב העליון שנמצא בו הצי הזה ולכן מציאת הצי עולה O(log*m) משוערך בממוצע על הקלט ואז מכיוון שאנו שומרים את מספר הספינות של כל צי ומעדכנים אותו בעת ביצוע איחוד עם ציים אחרים, לכן החזרתו אחרי מציאת הצי עולה O(1) במקרה הגרוע, ומכאן סה"כ נקבל שהפונקציה עולה O(log*m) משוערך בממוצע על הקלט.

:get_pirate_money(pirateId)

נחפש על המזהה כפי שעשינו ב-pay_pirate ולכן זה עולה O(1) בממוצע על הקלט ומחזירים את ערך המשתנה שנשמר בו את כסף הפיראט שזה עולה O(1) במקרה הגרוע, ולכן סה"כ נקבל שהפונקציה עולה O(1) בממוצע על הקלט.

:unite_fleets(fleetId1, fleetId2)

נתחיל בבדיקת שהציים הם מזהים של קבוצה ולאחר מכן נמצא את האב העליון שלהם בעץ ההפוך וזה עולה O(log*m) משוערך בממוצע על הקלט כפי שהזכרנו קודם, אם הם אכן נמצאים נאחד אותם ע"י שינוי מצביע האבא של הצי בעל מספר "האיחודים" הקטן ביותר להצביע לצי בעל מספר "איחודים" הגדול ביותר, וזה מתקיים ע"י שמירת משתנה מונה את מספר האיחודים של כל צי, ואז המזהה של הקבוצה הוא הצי בעל מספר הפיראטים הגדול ביותר , ומעדכנים את הדרגות בצורה המתאימה בדומה לשאלת הארגזים מהתרגול ומעדכנים את מספר הפיראטים והספינות של האב העליון להיות המספר של הפיראטים והספינות של כל הקבוצה .

ולכן סה"כ נקבל שהפונקציה עולה O(log*m) משוערך בממוצע על הקלט.

:pirate_argument(int pirateId1, int pirateId2)

נתחיל בלחפש את הפיראטים שזה עולה כפי שהסברנו קודם O(1) בממוצע על הקלט, ונוודא שהם שונים, ואז נחפש על הציים שלהם שזה עולה O(log*m) משוערך בממוצע על הקלט ובחיפוש נסכום את הRANK מהמזהה הצי הנשמר בתוך הפיראט עד הראש של העץ ההפוך ונוסיף לו הRANK השמור בתוך הפיראט ובכך נקבל ה rankהכללי של הפיראט וזה עולה O(log*m) משוערך בממוצע על הקלט, ואחר כך נוודא ששניהם עם אותו מזהה, ואחר מכן נחסיר את הפרש הדרגות של הפיראט בעל הדירוג הגבוה יותר מסכום הכסף שלו, ובאופן דומה נוסיף את ההפרש לסכום הכסף של הפיראט בעל הדירוג הנמוך יותר.

כל פעולה שעשינו למעט מציאת הציים עולה O(1) במקרה הגרוע, ולכן סה"כ נקבל

שהפונקציה עולה O(log*m) משוערך בממוצע על הקלט.

<u>סיבוכיות המקום של המערכת</u> היא O(n+m) וזה מכיוון שאנו שומרים את הפיראטים והציים בטבלאות ערבול בגודל שווה אסימפטוטית למספרם, ובנוסף העץ ההפוך גודלו שווה למספר הציים שהוא m ולכן סה"כ נקבל שסיבוכיות המקום הנדרשת למערכת בכדי לפעול היא O(n+m).