יבש של רטוב 1

מבני נתונים שבחרנו להשתמש בו **הוא עץ AVL.**

מאפיינים של עציAVL

1. **עץ מאוזן** :עצי AVL הוא עץ חיפוש בינארי שמאזן את עצמו ושומר על גובה מאוזן.
2. **גורם איזון** :לכל צומת יש גורם איזון המחושב כהפרש בין הגבהים של תתי העצים הימני מהשמאלי. גורם האיזון משמש לשמירה על איזון העץ.
3. **גלגול** :כדי לשמור על האיזון, עצי AVL מבצעים גלגולים (גלגול שמאלה, גלגול ימינה, גלגול שמאלה-ימינה, גלגול ימינה-שמאלה). גלגולים אלו מסדרים מחדש את הצמתים כדי להבטיח את קיום תנאי גורם האיזון לאחר הכנסת או מחיקת צומת.
4. **גובה** :הגובה של עץ AVL הוא תמיד O(logn) כאשר n הוא מספר הצמתים בעץ. דבר זה מבטיח כי פעולות חיפוש, הוספה ומחיקה יהיו בעלות סיבוכיות זמן של $O(logn)$.
5. **מאפייני עץ חיפוש בינארי:** כמו כל עצי החיפוש הבינאריים, עץ AVL שומר על המאפיין שלכל צומת, כל האלמנטים בתת העץ השמאלי קטנים מהצומת וכל האלמנטים בתת העץ הימני גדולים מהצומת.

פעולות וסיבוכיות

- **הוספה**: בעת הוספת צומת, עצי AVL מבטיחים איזון על ידי עדכון גורמי האיזון וביצוע גלגולים אם יש צורך. סיבוכיות: O(logn) .
- **מחיקה**: בדומה להוספה, עץ AVL מאזן מחדש את העץ לאחר מחיקת צומת על ידי עדכון גורמי האיזון וביצוע גלגולים. סיבוכיות: O(logn) .
- **חיפוש**: פעולת החיפוש בעץ AVL נשארת בעלת סיבוכיות של O(logn) בשל שמירת האיזון של העץ.
- והוספנו שני משתנים ששומרים מזהה האיבר בעל key מינימלי והמקסימלי ובפונקציות remove, insert מעדכנים אותם דרך findmin,findmax ולפי

הרצאה פונקציות אלו מתבצעות ב O(log n) ולכן remove, insert מתבצעות ב O(3 * log n)=O(log n)

בעיית הפיראטים וספינות:

אתחול של oceans_t() oceans:

בחרנו לאתחל אובייקט oceans שיש לו שני עצי AVL, אחד שהוא אמור להכיל את כל הספינות ממוינות לפי מזהה, ואחד שמכיל את כל הפיראטים שמסודרים לפי מספר הזהות, ובנוסף לכל ספינה מאותחלים לה שני עצי AVL של הפיראטים, אחד אמור להיות מסודר לפי כמות הכספים (המפתח הראשי) של הפיראטים ולפי מספר הזהות שלהם (במקרה שלשני פיראטים יש אותה כמות כספים) , ואחד מסודר לפי סדר הכנסת הפיראטים לספינות שלהם.

לפי תכונות עץ AVL מתקיים שאתחול של עץ AVL ריק הוא O(1) ,ולכן סה"כ נקבל שהזמן הנדרש הוא O(1).

שחרור של oceans_t() oceans~:

שחרור זיכרון של עצי AVL בתוך ה-oceans: הסיבוכיות של שחרור זיכרון של עץ AVL היא O(n) עבור הפיראטים ו- O(m) עבור הספינות, כאשר n הוא מספר הפיראטים ו-m הוא מספר הספינות. זאת מכיוון שיש לבקר בכל צומת בעץ פעם אחת כדי לשחרר אותו מהזיכרון, ובאופן דומה עבור העצים בתוך הספינות מכיוון שסכום כל הפיראטים הוא n ובכל ספינה יש 2 עצים לכן סה"כ משחררים בכל הספינות 2n פיראטים, ואז סה"כ נקבל 3n+m כלומר O(n + m).

הערה: זה מתבצע בצורת postorder הביקור יתחיל מראש העץ (בלי לשחרר בכדי לא לאבד מידע) ויוריד באופן רקורסיבי לתת עץ השמאלי מוחק אותו באותה צורה ולאחר מכן יוריד באופן רקורסיבי לתת עץ הימני מוחק אותו באותה צורה ולאחר מכן מוחק הראש וזה מתבצע והסיבוכיות של שחרור כל צומת הוא O(1), ולכן סה"כ נקבל בשחרור העץ הזמן הנדרש הוא O(number of nods).

הוספת ספינה add_ship(int shipId, int cannons) :

בהתחלה בודקים אם הספינה קיימת בעץ AVL וזה דרך פונקציית find המתבצעת ב O(log m) אם היא לא נמצאת בעץ נוסיף אותה לעץ ה AVL הוספת ספינה תתבצע בעזרת פעולת insert של עץ ה-AVL , ולפי תכונות ה-AVL אכן הפעולה הזו מתבצעת בסיבוכיות של O(log m) , השתמשנו ב-try ו-catch בכדי לטפל במקרה שיש בעיה בהקצאה/שחרור זיכרון בכל אורך הקוד.

הסרת ספינה remove_ship(int shipId):

הסרת ספינה תתבצע בעזרת פעולת remove של עץ ה-AVL, ולפי תכונות ה-AVL אכן הפעולה הזו מתבצעת בסיבוכיות של O(log m) .

בהתחלה בודקים אם הספינה קיימת בעץ AVL וזה דרך פונקציית find המתבצעת ב O(log m) אם היא נמצאת נבדוק אם יש בה פיראטים דרך הפונקציה size של עץ AVL שמחזירה את משתנה ששומר את גודל עץ ה AVL (הוא מתעדכן בפונקציות remove, insert של עץ AVL) אם אין בה פיראטים מוחקים אותה לפי פעולת remove של עץ הAVL

הוספת פיראט add_pirate(int pirateId , int shipId, int treasure):

מציאת הספינה הנדרשת בעץ הספינות לוקחת O(log m) זמן בזכות פונקציית ה-find של עץ ה-AVL שלוקחת O(log number of nodes).
הוספת פיראט תתבצע בעזרת פעולת insert של עץ ה-AVL הוספת כל פיראט תתבצע שלושה פעמים וזה בגלל שאנו מוסיפים אותו לעץ הראשי שמסודר לפי מספר הזהות ולעצים של הספינה שהוא שייך אליה שהם עץ מסודר לפי סכום הכסף ועץ מסודר לפי מספר ההכנסה ונשמור בתוך הפיראט מצביע לעץ שהוא נמצא בו, ולפי תכונות ה-AVL אכן הפעולות האלו מתבצעות בסיבוכיות של O(3*log n)=O(log n), כלומר סה"כ נקבל O(log n + log m) כנדרש.

הסרת פיראט remove_pirate(int pirateId):. מציאת פיראט הנדרשת בעץ הפיראטים הממוין לפי מספר הזהות לוקחת O(log n) זמן בזכות פונקציית ה-find של עץ ה-AVL שלוקחת O(log number of nodes).

בתוך הפיראט יש מצביע לעץ שהוא נמצא בו דרך המצביע נמחק אותו משני עצי AVL הנמצאים בתוך הספינה דרך פונקציית remove של עץ AVL שבמקרה הגרוע זה מתבצע ב O(2* log n)=O(log n)וזה כאשר כל הפיראטים נמצאים בספינה הזאת
ולאחר מכן נסיר את הפיראט מעץ AVL הממוין לפי מזהה וזאת תתבצע בעזרת הפונקציה- remove של עץ ה-AVL, הסרת כל פיראט תתבצע שלושה פעמים וזה בגלל שהוא נמצא בשלושת העצים שהזכרנו קודם, ולפי תכונות ה-AVL אכן הפעולות האלו מתבצעות בסיבוכיות של O(log n), כלומר סה"כ נקבל O(log n) כנדרש.

בגידה treason(int sourceShipId, int destShipId):

השתמשנו בפונקציית find שמימשנו במחלקת ה-AVL בכדי למצוא את מספרי הזהות של כל אחת מהספינות O(log m), ואז השתמשנו בעץ הפיראטים שמסודר לפי סידור הכנסת הפיראטים לספינת המקור הנמצא בתוך הספינה, ולפי עץ AVL שמימשנו ששומר מזהה של פיראט בעל key מינימלי נקבל את המזהה של הפיראט שנכנס ראשון מהעץ AVL הממוין דרך הפונקציה שמחזירה את המזהה של הפיראט בעל key מינימלי, ואחר כך מוציאים את הפיראט הזה מספינת המקור ומכניסים אותו לספינת היעד דרך remove insert, של עץ AVL בהתחלה מוצאים אותו מעץ הAVL הממוין לפי אוצר ומעץ AVL הממוין לפי מספר הכניסה של הפיראט ולאחר מכן מכניסים אותו לשני העצים אלו אבל בספינה השנייה , וכל פעולה מאילה עולה O(log n) כפי שציינו לעיל, ומשנים את המצביע שנמצא בתוך פיראט שמצביע על הספינה שזה O(1) ואחר כך מעדכנים את אוצר הפיראט בכדי להתאים אותו עם אוצר ספינת היעד בשביל לקבל סה"כ האוצר הכולל של הפיראט(אוצר הפיראט הוא נפרד מאוצר הספינה, בעת החזרת האוצר האמתי של הפיראט אנו לוקחים גם בחשבון את אוצר ספינתו, ניתן הסבר יותר מפורט ב- (*)) וזה כמובן עולה O(1) זמן, כלומר סה"כ נקבל O(log n + log m) כפי שנדרש בשאלה.

עדכון אוצר הפיראט update_pirate_treasure(int pirateId , int change) :

בהמשך להסבר ה**בגידה** לגבי תכנון אוצר הפיראט, אנו פה מוציאים את הפיראט בעזרת הפונקציה find ממחלקת AVL שזה עולה O(logn) זמן, יש בפיראט מצביע לספינה שהוא נמצא דרכו נמחק הפיראט מהעץ הממוין לפי האוצר שזה עולה O(logn) ואז מוסיפים את ערך change לאוצרו של הפיראט שזה עולה O(1) ולאחר מכן מוסיפים אותו לעץ AVL הממוין לפי אוצר שזה עולה O(logn) ועושים מחיקה והוספה מעץ AVL הממוין לפי אוצר כדי לשמור על מיון נכון , כלומר סה"כ נקבל סיבוכיות זמן של O(logn) כפי שנדרש.

החזרת אוצר הפיראט get_treasure(int pirateId) (*):

נחזור על ההסבר מקודם בכדי לתאר את הרעיון ולוודא שהכל ברור, אנו מבטאים את אוצר הפיראט ע"י שני משתנים, הראשון הוא המרכזי ששומרים בו את הקלט פחות האוצר שנמצא בספינה שנכנס אליה בעת הכנסת הפיראט ל-ocean ומעדכנים אותו בכל פעם שרוצים לשנות אוצר הפיראט באופן ישיר, והשני הוא אצל הספינה שהפיראט שייך לה, שזה מאותחל מערך-0, והוא אמור להשתנות בהתאם לתוצאות הלחימות של הספינה הזאת, ולכן סה"כ אנו מחזירים את סכום שני המשתנים בכדי לקבל האוצר האמתי של הפיראט, וזה כמובן אחרי שביצענו find בעץ מספרי הזהות של הפיראטים בכדי למצוא את הפיראט הזה ובתוך הפיראט יש מצביע לספינה שהוא נמצא בה דרכו נקבל את האוצר

של הספינה , כלומר סה"כ נקבל סיבוכיות זמן של O(log n) כפי שנדרש.

<u>הערה:</u> בעת שינוי הספינה שהפיראט שייך אליה, אנו מחסרים מאוצר הפיראט "הראשי" את הפרש אוצרי הספינות בכדי לשמור על תקינות הסכום האמתי של אוצר הפיראט.

<u>החזרת מספר התותחים get_cannons(int shipId) :</u>
בעת אתחול כל ספינה שומרים את מספר התותחים במשתנה, כלומר הגישה אליו היא O(1) זמן, וזה ביחד עם מציאת הספינה בעץ AVL של הספינות שמסודר לפי מספרי הזהות שלהן באמצעות find, יוצא סה"כ O(log m) כפי שנדרש.

<u>החזרת הפיראט העשיר ביותר get_richest_pirate(int shipId) :</u>
נמצא את הספינה המתאימה בהזנת מספר הזהות ל-find שמחזירה את הספינה המתאימה בסיבוכיות זמן של O(log m).
מציאת הפיראט העשיר תתבצע בזמן O(1) באמצעות שמירת משתנה לספינה שמעדכנים אותו כל פעם שמכנסים בה פיראט חדש לספינה או מוצאים אחד, ובכך סה"כ נקבל שסיבוכיות הזמן של הפונקציה הזאת היא O(log m).

<u>לחימת ספינות ships_battle():</u>
נתחיל במציאת הספינות בעץ הספינות באמצעות הפונקציה find של עץ ה-AVL וזה סה"כ עולה O(log m) זמן.
אחר מכן שומרים במשתנה את המינימלי בין מספר התותחים ומספר הפיראטים של כל ספינה(כל אחד מהם נשמר בתור משתנה בתוך מחלקת הספינות שאולי יתעדכן בהתאם להוספת פיראטים ולכן הפעולה הזו תתאפשר ב-O(1) , ואז משווים בין שני המשתנים ומשנים את ערך האוצר של כל אחד מהספינות מוסיפים לאוצר של הספינה המנצחת את מספר הפיראטים בספינה המפסידה ומפחיתים מהאוצר של הספינה המפסידה את מספר הפיראטים בספינה המנצחת שזה יעלה O(1) זמן כיוון שכל אוצר נשמר בתור משתנה של מחלקת הספינות כפי שהסברנו קודם ב-(*).
ואז סה"כ נקבל סיבוכיות זמן של O(log m).

לאורך כל התוכנית **<u>סיבוכיות המקום</u>** הנדרשת היא O(n+m) וזה נובע מכך ששמרנו כל ספינה פעם אחת ולכן סה"כ עבור הספינות נדרש מקום O(m) ומכיוון ששמרנו כל פיראט 3 פעמים לאורך כל התוכנית לכן נקבל סה"כ 3n מקום נדרש כלומר O(n), ולכן סה"כ O(n+m).

מקווה שזה היה simple.