תרגיל בית רטוב 1 – יבש

הגשת:

רזי חליחל – 325056240

יארא זועבי – 212264048

תאריך:
15.7.2024

מבני נתונים 1 - 02340218

תוכן עניינים

רעיון כללי

בחרנו לממש את המחלקה הדרושה – Ocean – ע״י שתי מחלקות (מחלקת פיראטים ומחלקת ספינות) ובנוסף לכך מימשנו גם צמתים ועץ AVL גנריים שישמשו אותנו לשלושה עצים שונים בעלי אותה פונקציונליות.

מחלקת הפיראטים: כל אובייקט מהמחלקה ייצג פיראט אחד ייחודי בעל Id מסוים. כל פיראט יהיה מחובר לשני פיראטים אחרים באמצעות מצביעים, הפיראט שנכנס לפניו לאותה ספינה ולפיראט שנכנס אחריו לאותה ספינה (ומצביע רק במקרה שאין כזה). בנוסף, לפיראט יהיה מצביע לספינה שהוא שייך אליה.

מחלקת הספינות: כל אובייקט מהמחלקה ייצג ספינה אחת ייחודית בעלת Id מסוים. כל ספינה תהיה אחראית על תת-עץ פיראטים משלה שממוין לפי כמות המטבעות שלהם. בנוסף כל ספינה תצביע על הפיראטים הראשון והאחרון שנכנסו. כמו כן תשמור על מידע של כמה בסה״כ הספינה זכתה/הפסידה כסף.

מחלקת צומת העץ: כל צומת יכיל

- מצביעים לאבא ובנים שלו.
- מצביע לאובייקט שהוא מייצג
- הגובה שלו ביחס לעץ ומצביע לצומת בעלת התכונה המקסימלית (יותר פירוט על זה בהמשך)

מחלקת עץ הAVL: המחלקה תכיל מצביע לשורש ופונקציונליות הוספת והסרת צמתים וחיפוש.

מחלקת ה Ocean: תכיל שני עצי AVL, עץ פיראטים ועץ ספינות שיכילו את כלל הפיראטים והספינות.

עבור n פיראטים ו- m ספינות, במימוש זה אנו יכולים לגשת לכל פיראט בסיבוכיות של $O(\log n)$ ולכל ספינה בסיבוכיות של $O(\log m)$.

מעבר לזה, בהינתן פיראט כלשהו שמצאנו אותו ניתן למצוא את הספינה שלו ב-$O(1)$ (ע״י המצביע שלו לספינה).

אם נרצה לקבל את הפיראט הראשון שנכנס בהינתן ספינה כלשהיא שמצאנו אותה, נוכל לעשות זאת גם ב-$O(1)$ (ע״י המצביע הייחודי לכך).

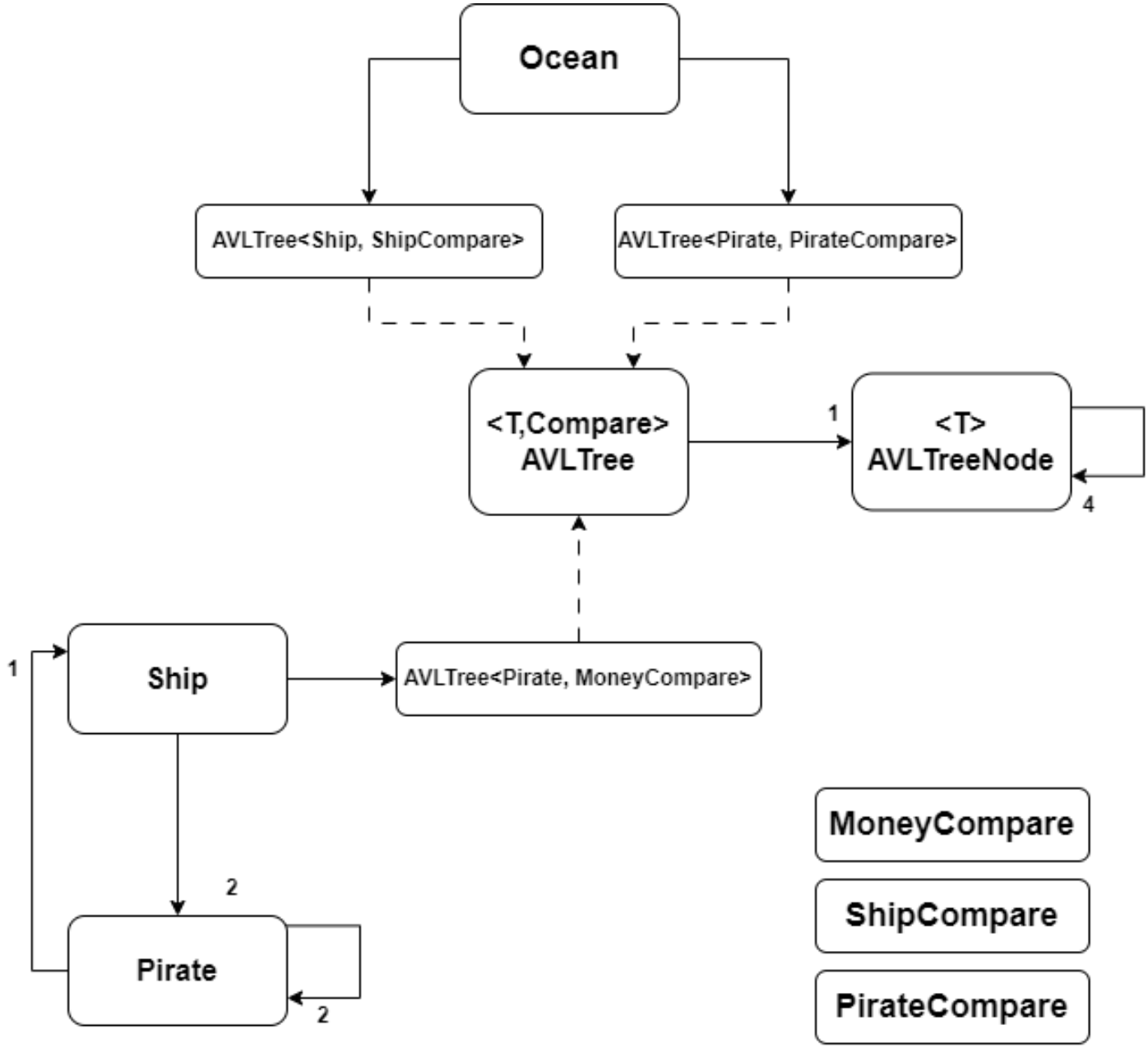
עבור סיבוכיות המקום:

אנו שומרים על כל אובייקט ספינה פעם אחת ועל כל פיראט פעם אחת. לכל ספינה נשמור על צומת ייחודית שתייצג אותה בעץ ולכל פיראט נשמור על שתי צמתים שונות שאחת תייצג אותו בעץ הפיראטים ואחת בעץ הפיראטים ששייכים לספינה.

בסה״כ אנו דורשים לכל ספינה אובייקט אחד וצומת אחת בעלי מספר שדות סופי כך שנשמור על $O(m)$ זיכרון ולכל פיראט שומרים על אובייקט אחד ושתי צמתים בעלי מספר שדות סופי כך שנדרש $O(n)$ זיכרון. ביחד נדרש בסה״כ $O(n+m)$ זכרון.

בנוסף באף פונקציה שמימשנו לא הגדרנו משתנים מקומיים נוספים חוץ ממספר סופי של משתני עזר בעלי מקום סופי, וכמו כן לא ממשנו את הפונקציות באמצעות רקורסיה, ובכך שמרנו על סיבוכיות מקום של $O(1)$ בכל הפונקציות.

UML


Ocean
AVLTree<Ship, ShipCompare>
AVLTree<Pirate, PirateCompare>
<T,Compare>
AVLTree
1
<T>
AVLTreeNode
4
1
Ship
AVLTree<Pirate, MoneyCompare>
2
Pirate
2
MoneyCompare
ShipCompare
PirateCompare

פונקציות ראשיות

בפרק זה נפרט לכל פונקציה את האלגוריתם שעומד מאחוריה ואיך הוא עומד בסיבוכיות הזמן והמקום הדרושים. עבור פרק זה נשתמש בטענות על סיבוכיות פונקציות העזר שמימשנו אותם אשר נוכיח את הסיבוכיות שלהם בפרק הבא.

Ocean() – Constructor

הבנאי של המחלקה מאתחל שני עצים ריקים, כל עץ מכיל בהתחלה רק פונקציה של השוואה שמועברת אליו ע"י הבנאי באופן קבוע לפי סוג העץ, פעולה זו דורשת $O(1)$.

~Ocean() – Destructor

ההורס של המחלקה ממומש ע"י ההורס הדיפולטי של המחלקה משום שכל הזיכרון ממומש ע"י מצביעים חכמים ואין שימוש במערכים. כדי להרוס כל הזיכרון, הוא חייב לעבור על כל צומת בעץ הפיראטים ($O(n)$) ובנוסף יעבור על כל צומת של עץ הספינות ($O(m)$) ולכל צומת (ספינה) יעבור על עץ הפיראטים שלו, שבסה"כ שווה למספר הפיראטים (כלומר $O(n)$) אזי סה"כ סיבוכיות הזמן היא $O(2n+m)$.

אזי פעולה זו דורשת $O(n+m)$.

add_ship

כדי להוסיף ספינה, נחפש אם היא קיימת בעץ הספינות בעזרת פונקציית עזר של העץ וגם הוספת הספינה לעץ הספינות תהיה ע"י פונקציית ההוספה לעץ (insert) הגנרי. שתי הפונקציות דורשות $O(h)$ כלומר $O(\log m)$.

הטיפל בשגיאות הוא שרשרת פעולות קבועות לכן יידרוש $O(1)$. בסה"כ הפונקציה בעלת סיבוכיות של $O(\log m)$.

remove_ship

כדי להסיר ספינה, נדרש לעשות אותו תהליך כמו ב-add_ship אך עם שימוש בפונקציית ה-remove של העץ במקום ה-insert. פעולה זו גם דורשת $O(\log m)$ לכן עם הטיפול בשגיאות זו תהיה סיבוכיות הפונצקיה.

add_pirate

כדי להוסיף פיראט, נחפש אם הוא קיים בעץ הפיראטים בעזרת פונקציית עזר של העץ וגם הוספת הפיראט לעץ הפיראטים תהיה ע"י פונקציית ההוספה לעץ (insert) הגנרי. שתי הפונקציות דורשות $O(h)$ כלומר $O(\log n)$.

בנוסף לכך, נדרש למצוא את הספינה, ע"י פונקציית החיפוש בסיבוכיות $O(\log m)$, ונוסיף לעץ הפיראטים שלה את הפיראט בסדר לפי מספר המטבעות בעזרת פונקציית הinsert עוד פעם שהיא $O(\log n)$

הטיפל בשגיאות הוא שרשרת פעולות קבועות לכן ידרוש $O(1)$. בסה"כ הפונקציה בעלת סיבוכיות של $O(\log n + \log m)$.

remove_pirate

כדי להסיר פיראט, נעשה תהליך דומה להוספת הפיראט (חיפוש על מקום הפיראט והספינה שלו) ואז הסרתו מעץ הפיראטים ועץ הפיראטים של הספינה ע״י remove של עץ ה-AVL. פעולות אלו עם טיפול בשגיאות דורשות $O(\log m + \log n)$.

treason

פונקציה זו קוראת פעמיים לפונקציית החיפוש של הספינות, פעם עם sourceShipId ופעם עם destShipId, כל קריאה לוקחת $O(\log m)$ ולכן סה״כ $O(\log m) = O(\log m) + O(\log m)$.

את הפיראט שיש להוציא נקבל אותו ע״י המצביע לפיראט הראשון שישנו בספינה ($O(1)$) ואז כדי להעביר אותו, נשתמש בremove_pirate ו- add_pirate שמימשנו מה-source ל-dest בהתאם שכל אחת דורשת $O(\log n + \log m)$.

כדי לשמור על תקינות מספר המטבעות – נחבר לפיראט את מספר המטבעות שהספינה המקורית שלו ($sourceShipId$) זכתה לפני שנסיר אותו.

בדיקת השגיאות ופעולת השמירה על המטבעות היא בסיבוכיות $O(1)$ לכן מתקיים שסיבוכיות זמן הפונקציה היא $O(\log n + \log m)$ כנדרש.

update_pirate_treasure

פונקציה זו מוצאת את מיקום הפיראט בעץ הפיראטים ע״י קריאה לפונקציית העזר של החיפוש שסיבוכיות הזמן שלה היא $O(\log n)$.

אם הוא לא בעץ סיימנו. אחרת נעדכן את שדה המטבעות אצלו, אבל עכשיו נצטרך לעדכן את המיקום שלו בעץ הפיראטים הממוין ע״י המטבעות (עץ הפיראטים של הספינה), ולכן נוציא אותו מהעץ הזה ונכניס אותו בחזרה בעזרת שתי הפונקציות שהגדרנו insert, remove של עץ הAVL הגנרי. סיבוכיות הזמן של כל אחת מהן במקרה הגרוע היא $O(\log n)$.

בנוסף בדיקת השגיאות היא בסיבוכיות $O(1)$ ולכן סה״כ מתקיים כי סיבוכיות הזמן של הפונקציה הוא $O(3\log n + 1) = O(\log n)$ כנדרש.

get_treasure

פונקציה זו מוצאת את מיקום הפיראט בעץ הפיראטים ע״י קריאה לפונקציית העזר של החיפוש שסיבוכיות הזמן שלה היא $O(\log n)$. לאחר מכן קוראים לgetter הtreasure של הפיראט. בדיקת השגיאות היא בסיבוכיות $O(1)$ לכן מתקיים שסיבוכיות זמן הפונקציה היא $O(\log n)$ כנדרש.

get_cannons

פונקציה זו מוצאת את מיקום הספינה בעץ הספינות ע״י קריאה לפונקציית העזר של החיפוש שסיבוכיות הזמן שלה היא $O(\log m)$. לאחר מכן קוראים לgetter הcannons של הספינה. בדיקת השגיאות היא בסיבוכיות $O(1)$ לכן מתקיים שסיבוכיות זמן הפונקציה היא $O(\log m)$ כנדרש.

get_richest_pirate

תחילה מחפשים את הספינה ע״י פונקציית העזר - $O(\log m)$. לכל ספינה ישנו שדה לעץ הפיראטים שלה שממנו אפשר לנווט לשורש העץ ($O(1)$). לכל צומת בכל עץ AVL גנרי אצלנו יש שדה שמצביע על הצומת בתת-העץ שלו בעל התכונה המקסימלית של אותו עץ. בעץ הפיראטים של הספינה, צומת זו מוגדרת להיות בעלת כמות המטבעות המקסימלית בעדיפות ראשונה ובעדיפות שנייה במקרה של שיוויון בכמות המטבעות הסדר נקבע לפי הId המקסימלי, לכן שורש זה יכיל מצביע לפיראט בעל כמות מקסימלית של מטבעות בעץ וכך נקבל אותו מיידית. בסה״כ עם הטיפול בשגיאות, נדרש $O(\log m)$.

ship_battle

מחפשים על הספינות בעזרת פונקציית העזר לחיפוש - $O(\log m)$.

בדיקת מספר הפיראטים והתותחים של כל ספינה וההשוואה ביניהם היא $O(1)$.

כדי לעדכן את המטבעות אצל הפיראטים בשתי הספינות הוספנו ל- class Ship שדה treasure_modifier כאשר הוא תכונה לכל הפיראטים הנוכחים בספינה המעיד על כמות המטבעות שיש להוסיף או להחסיר לכל פיראט בספינה המתאימה. שדה זה בעצם משמעותו כמה הספינה הפסידה או הרוויחה בסה״כ מאז יצירתה. מכיוון שזו תכונה משותפת לכל הפירטים שבספינה אז הסדר שלהם בעץ ה AVL הממוין לפי המטבעות לא משתנה כי לכולם המטבעות משתנים באותה כמות, ולכן סיבוכיות הזמן של עדכון ערך שדה זה בשתי הספינות הוא $O(1)$ מכיוון שאנו לא מעדכנים את הפיראטים עצמם אלא רק שדה הספינות שהוזכר.

בעזרת השדה שהוספנו שומרים על סיבוכיות הזמן המבוקשת של הפונקציה $O(\log m)$.

פונקציות עזר

Getters & Setters

כל מתודות ה-get וה-set של המחלקות שהגדרנו פועלות ב-$O(1)$ מפני ושהם ניגשות אך ורק לשדה המחלקה שלהם. בנוסף מוגדר getter עבור מחלקת הצמתים שמחשב את הbalance factor שלו ע"י גובה הילדים שגם דורש $O(1)$.

Compare

מוגדרות אצלנו שלוש מחלקות שמשמשות כפונקציות השוואה בין המחלקות. אחת להשוואה בין ה-Id של הספינות ואחת דומה עבור הפיראטים. השלישית מוגדרת להשוות בין הפיראטים על בסיס מספר המטבעות בעדיפות ראשונה וה-Id בעדיפות שנייה. לפי פונקציות אלו עצי ה-AVL מחליטות בהתאם איך להשוות בין הצמתים ושמירת הסדר של עץ החיפוש.

חיפוש – Find

פונקציות החיפוש שמימשנו פעולות על בסיס עץ חיפוש שהינו AVL לכן כל פעולת חיפוש שנבצע תדרוש סיבוכיות של $O(h)$ (כפי שראינו בהרצאה) כאשר h הוא גובה העץ הנתון.

עדכון צומת

לכל צומת עץ מוגדרות פונקציות עדכון גנריות שבעת הקריאה מעדכנים אל הצומת את הגובה החדש שלה (ע"י גובה הבנים) ואת הצומת בעל התכונה המקסימלית של תת-העץ שלו (גם ע"י השוואת המקסימום של הילדים ועצמו). שתי הפעולות בהיותן השוואות של שלושה אובייקטים בלבד לשתי תכונות דורשות $O(1)$.

מציאת צומת הבא בסדר – nextNode

פונקציה זו אחראית על חיפוש צומת שבאה אחרי הצומת הנתון לה לפי הסדר המוגדר בעץ. הפונקציה משמשת למקרה הסרת צומת מהעץ כאשר לצומת יש ילדים כפי שנלמד בהרצאה. פעולה זו שקולה לחיפוש לכן גם היא במקרה הגרוע דורשת $O(h)$.

פונקציות הגלגול (RL/RR/LR/LL/Rotate)

פונקציות אלו מוגדרות באופן גנרי לפי מה שלמדנו בכיתה בתוספת פונקציית Rotate שאחראית על קריאה לפונקציית הגלגול המתאימה. פעולת הגלגול לשמאל ולימין ממומשות ע"י החלפות של מצביעים ל3 צמתים. פעולות הגלגול של RL/LR קוראות פעמיים לפעולות הגלגול לימין/שמאל לפי הגלגול הדרוש. בכל גלגול גם מעדכנים את הצמתים שמושפעות ישירות מהגלגול. בסה"כ קיבלנו מספר סופי של פעולות לכן $O(1)$.

פונקציית מחיקת צומת שחסר לו ילדים

פונקציה זו מטפלת בהסרת צומת שחסר לו לפחות בן אחד ומחזירה את הצומת שהחליפה אותו. הצומת שהועבר לפונקציה למחיקה מוחלף ע"י הבן היחיד שלו אם קיים כזה אחרת מחליפה אותו במצביע ריק. לכן סה"כ סיבוכיות הזמן של הפונקציה היא $O(1)$.

הוספת צומת – insert

הפונקציה משמשת בפונקציות העזר שהוגדרו כך שהיא מחפשת את המקום התאורטי של הצומת שנרצה להוסיף (אם הוא קיים נסיים) ע״י פונקציית החיפוש ואז אחרי שנחבר אותו, נעבור על המסלול של הצומת בעלייה ממנו אל השורש כך שנעדכן את הצמתים ובמידת הצורך נעשה גלגול (לפי מה שנלמד בכיתה, נדרש אחד לכל היותר ומספיק לעדכן את מסלול הצומת). פעולות אלו כולם $O(1)$ שמתבצעות $O(h)$ (כפי שראינו בכיתה) פעמים לכל היותר, חוץ מפעולת החיפוש שמתבצעת פעם אחת עם $O(h)$ זמן. בסה״כ נקבל $O(h)$ זמן.

הסרת צומת – remove

באופן דומה להוספה, הפונקציה משמשת בפונקציות העזר שהוגדרו כך שהיא מחפשת את המקום של הצומת שנרצה להסיר (אם הוא לא קיים נסיים) ע״י פונקציית החיפוש. כדי להסיר אותו נחליף אותו בצומת הבא אחריו לפי הסדר אם יש לו שני ילדים ואז נמחק את הצומת ע״י מחיקת צומת שחסר לו ילד. אחרת נשתמש ישירות בפונקציית מחיקת צומת שחסר לו ילד.

בכל מקרה, לאחר המחיקה נעבור על המסלול של הצומת (או המוחלף) בעלייה ממנו אל השורש כך שנעדכן את הצמתים ובמידת הצורך נעשה גלגול. פעולות אלו כולם $O(1)$ שמתבצעות $O(h)$ פעמים לכל היותר (כפי שראינו בכיתה), חוץ מפעולות החיפוש ומציאת הצומת הבא שמתבצעת פעם אחת עם $O(h)$ זמן. בסה״כ נקבל $O(h)$ זמן.