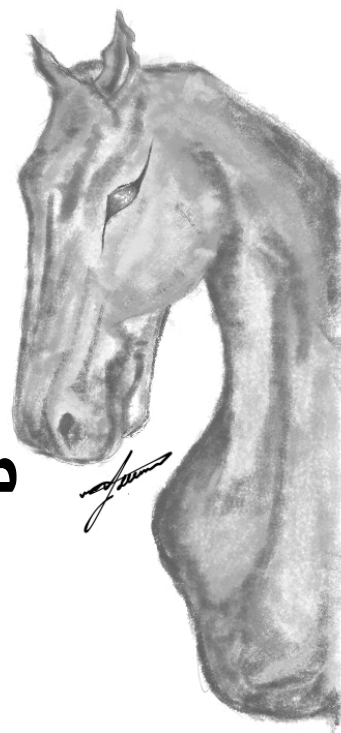


HORSE



מבני נתונים 1 - 234218 - חורף 2025
רטוב 1 - חלק יבש

וסים אנדראוס
ת.ז: 214636615

למאר סוסאן
ת.ז: 325810323

מבנה הנתונים:

מבנה הנתונים שלנו מכיל :

class node - מחלקה גנרית המאפשרת לנו לשמור: מפתחות (key), מידע (data), מצביע לבן השמאלי, מצביע לבן הימני, מצביע לאבא, גובה הצומת.

class AVL - עץ AVL - מחלקה גנרית מהטיפוסים key ו- data ומכילה את המחלקה node.
הערה: ראינו בהרצאה שעצי AVL מאפשרים חיפוש, הכנסה והוצאה בסיבוכיות זמן של $O(\log n)$ במקרה הגרוע ביותר.

class horse - לכל סוס יש:

- speed - המהירות של הסוס.
- herdID - מזהה העדר שהוא שייך לו.
- following - מצביע לסוס שהוא עוקב אחריו.
- version - גרסה המתארת מתי הסוס מצטרף לעדר (משמשת אותנו כדי לעקוב אחרי נכונות ה-follow של הסוס).
- followVersion - משמשת אותנו כדי לעקוב אחרי הסוס אשר אחריו עוקב הסוס שלנו.
- visitedTraversalsID - משמשת אותו כדי לבדוק האם יש מעגלים בפונקציות ...leads, can_run_together

class herd - מכיל עץ AVL של horses אשר מייצג את כל הסוסים שייכים לאותו העדר (כל צומת מייצג סוס). המפתח של כל צומת בעץ מתאר את המזהה הייחודי של הסוס והמידע הוא מסוג horse.

class plains - מכיל 3 עצים:

1. עץ AVL של horses: מכיל את כל הסוסים במערכת, ממויין לפי המפתחות שמייצגים את המזהה של הסוס, המידע הוא מסוג horse.
2. עץ AVL של herds: מכיל את כל העדרים שאינם ריקים במערכת, וכל עדר מכיל עץ של סוסים הנמצאים באותו העדר. (לפי ההסבר הנ"ל לגבי class herd). ממויין לפי המפתחות שמייצגים את המזהה של העדר, המידע הוא מסוג herd.
3. עץ AVL של emptyHerds: מכיל את כל העדרים הריקים במערכת. ממויין לפי המפתחות שמייצגים את המזהה של העדר, המידע הוא מסוג herd.

הסבר לגבי מימוש הפעולות הנדרשות והוכחת דרישות סיבוכיות הזמן:

Plains() :

בנאי - אתחול ריק של מבנה הנתונים plains.
סיבוכיות זמן: באתחול השדות של המחלקה Plains, מתבצעת 3 קריאות לבנאי של המחלקה הגנרית AVL אשר מאתחל 3 עצי AVL ריקים, אתחול עץ AVL עולה $O(1)$ במקרה הגרוע כפי שראינו בהרצאה.

~Plains() :

הורס - שחרור כל הזכרון אותו הקצאנו בסיום ריצת התוכנית.
סיבוכיות זמן: המבנה שלנו מכיל עץ AVL שמכיל n סוסים, עץ AVL שמכיל m עדרים (לא ריקים) ו-עץ AVL שמכיל m_ϕ עדרים ריקים.
נבצע סיור postOrder על צמתי כל עץ מהעצים הנ"ל ונמחק את ה-data שלהם (עצים ומצביעים). סיורי postOrder על צמתי כל העצים עולים $O(n + m + m_\phi)$ כפי שראינו בכיתה. לאחר מכן יקרא ההורס של המחלקה AVL כדי למחוק את העצים הראשיים בסיבוכיות זמן של $O(n + m + m_\phi)$ במקרה הגרוע לכן בסה"כ סיבוכיות הזמן של הפעולה הינה $O(n + m + m_\phi)$.

StatusType add_herd(int herdId) :

הוספת עדר ריק למערכת.
סיבוכיות זמן: נבצע בדיקה של הארגומנט אותו מקבלת הפונקציה, אם הוא אי חיובי, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של herds ובעץ של emptyHerds, אם קיים כבר עדר עם אותו מזהה נחזיר FAILURE. חיפוש בשני העצים הנ"ל עולה $O(\log m + \log m_\phi)$ כפי שראינו בכיתה. אם לא קיים עדר עם המזהה שקיבלה הפונקציה אז נקצה צומת חדש ונוסיף לעץ של emptyHerds. הוספת צומת לעץ AVL עולה $O(\log m_\phi)$. (אם תהייה בעיה בהקצאה מחזירים ALLOCATION_ERROR) לכן בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log m + \log m_\phi) + O(\log m_\phi) = O(\log m + \log m_\phi)$.


StatusType remove_herd(int herdId) :

הסרת עדר בלי סוסים מהמערכת.
סיבוכיות זמן: נבצע בדיקה של הארגומנט אותו מקבלת הפונקציה, אם הוא אי חיובי, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של emptyHerds, אם לא

קיים עדר עם מזהה $herdId$ נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log m_\phi)$ כפי שראינו בכיתה. אם קיים עדר עם המזהה שקיבלה הפונקציה בעץ העדרים הריקים אז נמחק את ה- $data$ שהוא מכיל. הסרת צומת מעץ AVL עולה $O(\log m_\phi)$. (אם תהייה בעיה בשחרור מחזירים ALLOCATION_ERROR) לכן בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log m_\phi) + O(\log m_\phi) = O(\log m_\phi)$.

 **StatusType add_horse(int horseId,int speed) :**

הוספת סוס למערכת.
סיבוכיות זמן: נבצע בדיקה של הארגומנטים אותם מקבלת הפונקציה, אם אחד מהם או שניהם אי חיוביים, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם קיים כבר סוס עם אותו מזהה נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה. אם לא קיים סוס עם המזהה שקיבלה הפונקציה אז נקצה צומת חדש ונוסיף לעץ של horses. הוספת צומת לעץ AVL עולה $O(\log n)$. (אם תהייה בעיה בהקצאה מחזירים ALLOCATION_ERROR) לכן בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log n) + O(\log n) = O(\log n)$.

 **StatusType join_herd (int horseId,int herdId) :**

מוסיפים סוס לעדר כלשהו.
סיבוכיות זמן: נבצע בדיקה של הארגומנטים אותם מקבלת הפונקציה, אם אחד מהם או שניהם אי חיוביים, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם לא קיים סוס עם אותו מזהה או שהסוס כבר שייך לעדר אחר נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה. נבצע חיפוש בעץ של herds ובעץ של emptyHerds, אם לא קיים עדר עם המזהה $herdId$ נחזיר FAILURE. חיפוש בעצים הנ"ל עולה $O(\log m + \log m_\phi)$ כפי שראינו בכיתה.
 אם הסוס קיים והעדר קיים והוא לא ריק (יש בו לכל היותר n סוסים), נקצה צומת חדש ונכניס אותו לעץ horses שנמצא בעדר, הכנסת איבר לעץ עולה $O(\log n)$. (אם תהייה בעיה בהקצאה מחזירים ALLOCATION_ERROR).
 אם הסוס קיים והעדר ריק, נמחק את הצומת של העדר מהעץ של emptyHerds ונוסיף אותו לעץ של herds. פעולת ההכנסה לעץ של herds וההסרה מהעץ של emptyHerds עולות ביחד $O(\log m + \log m_\phi)$. לאחר מכן נקצה צומת חדש ונכניס אותו לעץ horses שנמצא בעדר. (אם תהייה בעיה בהקצאה/שחרור זיכרון מחזירים ALLOCATION_ERROR). בסה"כ סיבוכיות הזמן הנדרשת במקרה הגרוע היא $O(\log n + \log m + \log m_\phi)$.

StatusType follow (int horseId,int horseToFollowId) :

הסוס עם המזהה horseId עוקב אחרי הסוס עם המזהה horseToFollowId במידה והם נמצאים באותו עדר.

סיבוכיות זמן: נבצע בדיקה של הארגומנטים אותם מקבלת הפונקציה, אם אחד מהם או שניהם אי חיוביים או אם הם שווים נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם לא קיים סוס עם המזהה horseId או אם לא קיים סוס עם המזהה horseToFollowId נחזיר FAILURE. אם אין להם אותו herdId נחזיר FAILURE.

חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה. אחרת, שני הסוסים קיימים ונמצאים באותו עדר, אז נשמור במידע של הסוס עם המזהה horseId מצביע לסוס עם המזהה horseToFollowId. (אם תהייה בעיה בהקצאה מחזירים ALLOCATION_ERROR). בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log n)$.

StatusType leave_herd (int horseId) :

הסוס עם המזהה horseId עוזב את העדר שלו. **סיבוכיות זמן:** נבצע בדיקה של הארגומנט אותם מקבלת הפונקציה, אם הוא אי חיובי, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם לא קיים סוס עם המזהה horseId או אם הסוס עם המזהה horseId אינו שייך לאף עדר נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה. אחרי שמצאנו את הסוס, נחפש בעץ של herds את העדר ששייך אליו הסוס, חיפוש בעץ הנ"ל עולה $O(\log m)$ כפי שראינו בכיתה, אחרי שמצאנו את העדר נמחק את הצומת של הסוס עם המזהה horseId. (בעץ horses שנמצא ב-herd שמצאנו יש לכל היותר n סוסים לכן מחיקת צומת בעץ הזה תעלה לנו $O(\log n)$ במקרה הגרוע ביותר).

אחרי שמחקנו את הצומת של הסוס נבדוק אם העדר הפך להיות ריק, אם כן אז נמחק את העדר מהעץ של herds (ראינו בכיתה שמחיקת איבר מעץ AVL עולה $O(\log m)$) ונוסיף את העדר לעץ של emptyHerds (ראינו בכיתה שהכנסת איבר לעץ AVL עולה $O(\log m_\phi)$).

כמובן בכל התהליך הנ"ל אם יש בעיה בהקצאה/שחרור זיכרון מחזירים ALLOCATION_ERROR.

בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log n + \log m + \log m_\phi)$.

output <int> get_speed (int horseId) :

מחזירים את מהירות הסוס בעל המזהה horseId.

סיבוכיות זמן: נבצע בדיקה של הארגומנט אותו מקבלת הפונקציה, אם הוא אי חיובי, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם לא קיים סוס עם המזהה horseId נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה. בסוף מחזירים את מהירות הסוס אותו מצאנו (וּכְמוּבֵן SUCCESS). בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log n)$.

output <bool> leads (int horseId, int otherHorseId) :

בודקים האם קיימת שרשרת של סוסים כך שהסוס במזהה horseId עוקב (בעקיפין) אחרי הסוס בעל המזהה otherHorseId.

סיבוכיות זמן: נבצע בדיקה של הארגומנטים אותם מקבלת הפונקציה, אם אחד מהם או שניהם אי חיוביים או אם הם שווים נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם לא קיים סוס עם המזהה horseId או אם לא קיים סוס עם המזהה otherHorseId נחזיר FAILURE. חיפוש בעץ הנ"ל עולה $O(\log n)$ כפי שראינו בכיתה.

לאחר מכן נבדוק האם קיימת שרשרת של סוסים כך שהסוס במזהה horseId עוקב אחרי הסוס במזהה otherHorseId בלולאה שרצה לכל היותר n_{herdID} פעמים - נכונות: עוברת על כל הסוסים בעדר של הסוסים הנ"ל לכל היותר פעם אחת, בעצם אנחנו בודקים בכל איטרציה אם כבר עברנו על אותו סוס, אם כן אז הלולאה תעצור כי ז"א שיש מעגל ולכן מחזירה false, יש n_{herdID} סוסים בעדר ובמקרה הגרוע נעבור על כל הסוסים בעדר, לכן סיבוכיות הזמן של הלולאה היא $O(n_{herdID})$ במקרה הגרוע.

כמובן בכל התהליך הנ"ל אם יש בעיה בהקצאה/שחרור זיכרון מחזירים ALLOCATION_ERROR.

בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log n + n_{herdID})$.

output <bool> can_run_together (int herdId) :

נבדוק האם העדר יכול לרוץ יחד (אם קיים סוס אחד שמוביל את כל הסוסים בעדר ואף סוס אינו מוביל אותו).

סיבוכיות זמן: נבצע בדיקה של הארגומנט אותו מקבלת הפונקציה, אם הוא אי חיובי, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של herds, אם לא קיים עדר עם מזהה מזהה herdId נחזיר FAILURE. חיפוש בעץ של herds עולה $O(\log m)$ כפי שראינו בכיתה.

אחרי שמצאנו את העדר, נבדוק את הגודל שלו (ששמרנו בשדה size) ב- $O(1)$ אם הוא ריק מחזירים false ואם מכיל סוס יחיד מחזירים true. אחרת, פורשים את העץ של העדר למערך בעזרת הפונקציה fillArrayData (שעוברת על כל צומת בעץ העדר אותו בודקים לכן סיבוכיות הזמן הנדרשת לעשות זאת היא $O(n_{herdID})$) לאחר מכן נבדוק בפונקציית עזר findLeader האם קיים יותר מ- leader אחד בלולאה שרצה לכל היותר n_{herdID} פעמים (עוברת על כל התאים במערך הנ"ל שהוא בגודל n_{herdID} וסופרת כמה leader-ים יש) לכן סיבוכיות הזמן של הלולאה היא $O(n_{herdID})$ במקרה הגרוע. הפונקציה מחזירה מצביע ל-leader אם יש רק אחד כזה ובכל המקרים האחרים תחזיר nullptr.

אחרי שמצאנו leader אחד בודקים אם כל הסוסים מחוברים אחד לשני ע"י המצביע following ובודקים שאין מעגלים, השתמשנו באלגוריתם דומה לפונקציית leads, זה נעשה ע"י שימוש בפונקציית עזר validate connections שרצה על כל הסוסים בעדר (עוברת על כל המערך) ובודקת שכל סוס מגיע ל-leader ושאינ מעגלים בין מצביעי following, בפונקציה זו גם עוברים על כל סוס לכל היותר פעם אחת ולכן סיבוכיות הזמן של הפונקציה היא $O(n_{herdID})$ במקרה הגרוע.

כמובן בכל התהליך הנ"ל אם יש בעיה בהקצאה/שחרור זיכרון מחזירים ALLOCATION_ERROR.

בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log m + n_{herdID})$.

הערה:

בפונקציות leads ו-can_run_together השתמשנו בפונקציות עזר resetVisitedTraversalId ו-resetVisitedTraversalIdInHerd שמאפסות את השדה visitedTraversalId במחלקה horse. זה נעשה ע"י ריצה בלולאה על המערך שבגודל n_{herdID} ללא תלות בלולאות קודמות, הן רצות בסיבוכיות זמן של $O(n_{herdID})$ ולכן לא משפיעות על זמן ריצת הפונקציות הדרוש.

ניתוח סיבוכיות המקום במבנה הנתונים:

- במבנה הנתונים שהצענו השתמשנו בעצי AVL שלמדנו בכיתה.
- ♦ עץ AVL של horses: מכיל את כל הסוסים במערכת, לכן סיבוכיות המקום היא $O(n)$.
 - ♦ עץ AVL של herds: מכיל את כל העדרים שאינם ריקים במערכת, ובתוך כל צומת שמרנו עץ AVL של horses שלכל היותר הגודל של כל העדרים ביחד הוא n (יש n סוסים במערכת) לכן סיבוכיות המקום היא $O(n + m)$.
 - ♦ עץ AVL של emptyHerds: מכיל את כל העדרים הריקים במערכת, לכן סיבוכיות המקום היא $O(m_\phi)$.

במהלך ריצת התוכנית הקצאנו דינמית מערכים, בכל פעם לכל היותר שני מערכים בגודל n_{herdID} ואחרי סיום ריצת הפונקציה הם משוחררים מהזיכרון, לכן הסיבוכיות הכוללת:

$$O(n + 2n_{herdID} + m + m_\phi) \leq O(3n + m + m_\phi) = O(n + m + m_\phi)$$

שוויון זה מוצדק כיוון ש $n_{herdID} \leq n$ ומהגדרת החסם O - או גדול. בסה"כ סיבוכיות מקום במבנה הנתונים הינה $O(n + m + m_\phi)$ כנדרש.