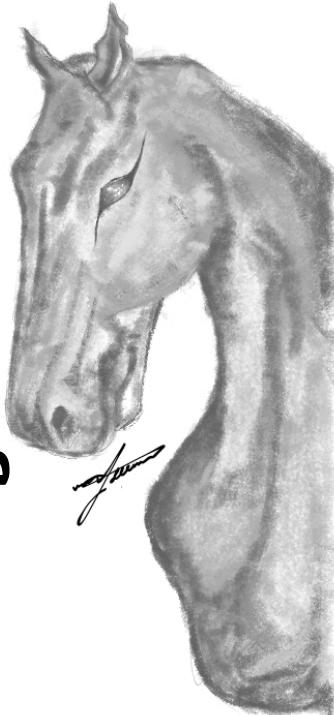


HORSE



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

**ויסים אנדרואס
ת.ז: 214636615**

**למאר סוסאן
ת.ז: 325810323**

מבנה הנתונים:

מבנה הנתונים שלנו מכיל :

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

class AVL - עץ AVL - מחלוקת גנרית מהטיפוסים key ו- data ומכללה את המחלוקת node.

הערה: ראיינו בהרצאה שעצם AVL מאפשר חיפוש, הכנסה והוצאה בסיבוכיות זמן של $O(logn)$ במקורה 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 אותו כדי לעקוב אחריו נכוונות ה-follow של הסוס).
- followVersion - משמשת אותו כדי לעקוב אחריו הסוס אשר אחריו עוקב הסוס שלנו.
- visitedTraversalID - משמשת אותו כדי לבדוק האם יש מעגלים בפונקציות ...leads, can_run_together

class herd - מכיל עצם 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 ומהידע הוא מסוג horse.

class plains - מכיל 3 עצים:

1. עץ AVL של horses: מכיל את כל הסוסים במערכת, ממוחין 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 עצם של סוסים הנמצאים באותו העדר. (לפי ההסבר הנ"ל לגבי class herd). ממוחין לפי המפתחות שמייצגים את המזהה של העדר, המידע הוא מסוג herd.

3. עץ AVL של emptyHerd: מכיל את כל העדרים הריקים במערכת. ממוחין 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**, ממבצעת 3 קריאות לבנאי של המחלקה הגנרטית AVL אשר מאותחל 3 עצי AVL ריקים, אתחול עץ AVL עולה $O(1)$ במקרה הגרוע כפי שראינו בהרצאה.

📌 ~Plains() :

הורס - שחרור כל הזכרן אותו הקצנו בסיום ריצת התוכנית.
סיבוכיות זמן: המבנה שלנו מכיל עץ AVL שמכיל m סוסים, עץ AVL שמכיל m עדרים (לא ריקים) ו-עץ AVL שמכיל m עדרים ריקים.
נבצע סיוור postOrder על צמתי כל עץ מהעצים הנ"ל ונמחק את ה- data שלם (עצים ומצביעים). סיוורי postOrder על צמתי כל העצים עולים (ϕ) העצים הראשיים בסיבוכיות זמן של ($m + m + m + n$) $O(n + m + m + m)$ במקרה הגרוע לכן בסה"כ סיבוכיות הזמן של הפעולה הינה (ϕ). $O(n + m + m + m)$.

📌 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.emptyHerd, אם קיימים כבר עם אותו מזהה נחזיר FAILURE. חיפוש בשני העצים הנ"ל עולה ($\log m + \log m$) $O(\log m + \log m)$ כפי שראינו בכיתה. אם לא קיימים עדר עם המזהה שקיבלה הפונקציה אז נקצת צומת חדש ונוסיף לעץ של emptyHerd. הוספת צומת לעץ AVL עולה ($\log m$) $O(\log m)$. (אם תהיה בעיה בהקצתה מחזירים ALLOCATION_ERROR) לכן בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה $O(\log m + \log m) + O(\log m) = O(\log m + \log m)$

📌 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 herds.emptyHerd, אם לא

קיימים עדר עם מזהה mazeId FAILURE. חיפוש בעץ הנ"ל עליה ($O(\log m_\phi)$ 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 תהיה בעיה בשחרור מחזירם 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 שנייהם אי חיוביים, נחזיר INVALID_INPUT. אחרת, נבצע חיפוש בעץ של horses, אם קיים כבר סוס עם אותו מזהה FAILURE. חיפוש בעץ הנ"ל עליה ($O(\log n)$ כפי שראינו בכיתה. אם לא קיים סוס עם המזהה שקיבלה הפונקציה או נקצת צומת חדש ונוסף לעץ של horses. הוספת צומת לעץ AVL עולה ($O(\log n)$. (אם תהיה 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 שנייהם 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 עליה ($O(\log n)$ כפי שראינו בכיתה. נבצע חיפוש בעץ של herds ובעץ של emptyHerd, אם לא קיים עדר עם המזהה herdId נחזיר FAILURE. חיפוש בעצים הנ"ל עליה ($O(\log m_\phi + \log m_\phi)$ כפי שראינו בכיתה.

אם הסוס קיים והעדר קיים והוא לא ריק (יש בו לפחות אחד נוסף), נקצת צומת חדש ונכנסו אליו לעץ של horses שנמצא בעדר, הכנסת איבר לעץ עולה ($O(\log n)$. (אם תהיה בעיה בהקצאה מחזירים ALLOCATION_ERROR).

אם הסוס קיים והעדר ריק, נמחק את הצומת של העדר מהעץ של emptyHerd, פועלות ההכנסה לעץ של herds וההסרה מהעץ של emptyHerd עולות ביחד ($O(\log m_\phi + \log m_\phi)$. לאחר מכן נקצת צומת חדש ונכנסו אליו לעץ horses שנמצא בעדר. (אם תהיה בעיה בהקצאה/שחרור זיכרון מחזירים ALLOCATION_ERROR). בסה"כ סיבוכיות הזמן הנדרשת במקרה הגרוע היא $O(\log n + \log m_\phi + \log m_\phi)$

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 נשמר בזיכרון של הסוס עם המזהה horseId מצביע לsus עם המזהה horseToFollowId. (אם תהיה בעיה בהקצאה מחזירים ALLOCATION_ERROR).
בזה"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 אותו 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 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 היוטר א סוסים لكن מחיקת צומת בעץ זהה 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 emptyHerd) (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.

בזה"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 ($n \log n$) כפי שראינו בכיתה. בסוף מחזירים את מהירות הסוס אותו מצאנו (וכמובן SUCCESS). בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע הינה ($n \log n$).

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

בודקים האם קיימת שרשרת של סוסים כך שהסוס בmaze[mazeIndex] עוקב (בעקיף) 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. חיפוש בעץ הנ"ל עולה ($n \log n$) כפי שראינו בכיתה.

לאחר מכן נבדוק האם קיימת שרשרת של סוסים כך שהסוס בmaze[mazeIndex] עוקב אחרי הסוס בmaze[otherHorseId] בלבד 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 "א שיש מעגל ולכן מחייב false", יש n_{herdID} סוסים בעדר ובמקרה הגרוע נעבור על כל הסוסים בעדר, לכן סיבוכיות הזמן של הלולאה היא ($n \log n$) במקרה הגרוע.

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

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

📌 output <bool> can_run_together (int herdId) :

נבדוק האם העדר יכול לרוץ יחד (אם קיים סוס אחד שמוביל את כל הסוסים בעדר ו אף 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 זהה herdId נחזיר FAILURE. חיפוש בעץ של herds עולה ($m \log m$) כפי שראינו בכיתה.

אחרי שמצאנו את העדר, נבדוק את הגודל שלו (שמרנו בשדה size) ב- $O(1)$. אם הוא ריק מחזירים false ואם הוא מכיל סוס ייחיד מחזירים true. אחרת, פורשים את העץ של העדר למערך בעזרת הפונקציה fillArrayData (שעוברת על כל צומת בעץ העדר אותו בודקים וכן סיבוכיות הזמן הנדרשת לעשות זאת היא $O(n_{herdID})$). לאחר מכן נבדוק בפונקציית עזר findLeader האם קיים יותר מ- leader单一. בלאה שרצה לכל היותר n_{herdID} פעמים (עוברת על כל התאים במערך הנ"ל שהוא בגודל n_{herdID} וסופרת כמה leader-ים יש) וכך סיבוכיות הזמן של הלולאה היא $O(n_{herdID})$ במקרה הגרוע. הפונקציה מחייבת 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 באлогוריתם דומה לפונקציית validate connections, זה נעשה ע"י שימוש בפונקציית עזר leads. הפעולות שערכו על כל הסוסים בעדר (עוברת על כל המערך) ובודקת שככל סוס מגיע ל-leader ושהאין מעגלים בין מצביעי following, בפונקציה זו גם עוברים על כל סוס לכל היותר פעם אחת ולכון סיבוכיות הזמן של הפונקציה היא $O(n_{herdID})$ במקרה הגרוע. כמובן בכל התהליך הנ"ל אם יש בעיה בהקצתה/שחרור זיכרון מחזירים ALLOCATION_ERROR. בסה"כ סיבוכיות הזמן של הפעולה במקרה הגרוע היא $O(\log m + n_{herdID})$.

הערה:

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

ניתוח סיבוכיות המקום במבנה הנתונים:

- במבנה הנתונים שהצענו השתמשנו בעצי AVL של מדנו בכיתה.
- עץ AVL של horses: מכיל את כל הסוסים במערכת, וכך סיבוכיות המקום היא $O(n)$.
- עץ AVL של herds: מכיל את כל העדרים שאינם ריקים במערכת, ובתווך כל צומת שמרנו עץ AVL של horses שלכל היותר הגודל של כל העדרים ביחד הוא n (יש n סוסים במערכת) וכך סיבוכיות המקום היא $(m + n)O(1)$.
- עץ AVL של emptyHerds: מכיל את כל העדרים הריקים במערכת, וכך סיבוכיות המקום היא $(m_\phi)O(1)$.

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

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

שווינו זה מוצדק כיון ש $n_{herdID} \leq n$ ומהגדלת החסם 0 - או גדול. בסה"כ סיבוכיות מקום במבנה הנתונים הינה $(m_\phi + n + m)$ כנדרש.