מתרגל ממונה על התרגיל: אמיר מן, amir.mann@campus.technion.ac.il

תאריך ושעת הגשה: 26/12/2024 בשעה 23:59

אופן ההגשה: בזוגות. אין להגיש ביחידים. (אלא באישור מתרגל אחראי של הקורס)

הנחיות כלליות:

- שאלות על התרגיל יש לפרסם באתר הפיאצה של הקורס תחת לשונית ״hw-wet-1״:
  - האתר: https://piazza.com/class/m34xgygziyn64l, נא לקרוא את השאלות של סטודנטים אחרים לפני שמפרסמים שאלה חדשה, למקרה שנשאלה כבר.
- נא לקרוא את המסמך ״נהלי הקורס״ באתר הקורס. בנוסף, נא לקרוא בעיון את כל ההנחיות בסוף מסמך זה.
- בפורום הפיאצה ינוהל FAQ ובמידת הצורך יועלו תיקונים כ**הודעות נעוצות** (Pinned Notes). תיקונים אלו **מחייבים**.
- התרגיל מורכב משני חלקים: יבש ורטוב.
  - לאחר קריאת כלל הדרישות, מומלץ לתכנן תחילה את מבני הנתונים על נייר. דבר זה יכול לחסוך לכם זמן רב.
  - לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד בכל דרישות הסיבוכיות בתרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול.
  - את הפתרון שלכם מומלץ לחלק למחלקות שונות שאפשר לממש (ולבדוק!) בהדרגתיות.
  - המלצות לפתרון התרגיל נמצאות באתר הקורס תחת: ״Programming Tips Session״.
    - המלצות לתכנות במסמך זה אינן מחייבות, אך מומלץ להיעזר בהן.
  - חומר התרגיל הינו כל החומר שנלמד בהרצאות ובתרגולים עד אך לא כולל עצי דרגות.
- העתקת תרגילי בית רטובים תיבדק באמצעות תוכנת בדיקות אוטומטית, המזהה דמיון בין כל העבודות הקיימות במערכת, גם כאלו משנים קודמות. לא ניתן לערער על החלטת התוכנה. התוכנה אינה מבדילה בין מקור להעתק! אנא הימנעו מהסתכלות בקוד שאינו שלכם.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת: jonathan.gal@campus.technion.ac.il

הקדמה:

במישורי העשב הגדולים של הנדסת חומרים עדרי סוסי פרא רבים. בשביל לעקוב אחרי הסוסים שנמצאים שם החליטו במשרד לסוסי הבר של הטכניון להקים מערכת למעקב אחרי עדרי הסוסים ויכולתם לרוץ בחופשיות. יחד עם זאת, אין למשרד לסוסי הבר ניסיון כלל במבני נתונים, והם אינם יודעים כיצד לעמוד בדרישות הסיבוכיות של עצמם! עזרו למשרד לכתוב מערכת כזאת לפי התנאים המפורטים מטה.

סימונים לצורכי סיבוכיות:

נסמן ב-n את מספר הסוסים במערכת.

ב-m את מספר העדרים שאינם ריקים במערכת.

ב-$m_\emptyset$ את מספר העדרים הריקים במערכת.

דרוש מבנה נתונים למימוש הפעולות הבאות:

plains_t()

מאתחלת מבנה נתונים ריק. תחילה אין במערכת עדרים או סוסים.

פרמטרים: אין

ערך החזרה: אין

סיבוכיות זמן: $O(1)$ במקרה הגרוע.

---

virtual ~plains_t()

הפעולה משחררת את המבנה (כל הזיכרון אותו הקצאתם חייב להיות משוחרר).

פרמטרים: אין

ערך החזרה: אין

סיבוכיות זמן: $O(n+m+m_\emptyset)$ במקרה הגרוע.

StatusType add_herd(int herdId)

הפעולה מוסיפה למבנה נתונים עדר בלי סוסים.

פרמטרים:

herdId	מזהה העדר שצריך להוסיף.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם herdId <=0.
FAILURE	אם קיים כבר עדר עם מזהה herdId.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log m + \log m_\emptyset)$ במקרה הגרוע.

StatusType remove_herd(int herdId)

העדר בעל המזהה herdId אינו בשימוש יותר, ולכן צריך להוציאו מהמערכת.
אם קיימים סוסים בעדר הוא אינו יכול להימחק ונשאר במערכת.

פרמטרים:

herdId	מזהה העדר.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם herdId<=0.
FAILURE	אם אין עדר עם מזהה herdId או שיש בו סוסים.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log m_{\emptyset})$ במקרה הגרוע.

StatusType add_horse(int $horseId$, int $speed$)

סוס בעל מזהה ייחודי $horseId$ מתוסף למערכת, אין לו עדר או מוביל בתור התחלה, ומהירותו היא speed.

פרמטרים:

$horseId$	מזהה הסוס שצריך להוסיף.
$speed$	מהירות הסוס שצריך להוסיף.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId$<=0 או אם $speed$<=0.
FAILURE	אם קיים כבר סוס במזהה $horseId$ במערכת.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n)$ במקרה הגרוע.

StatusType join_herd(int $horseId$, int $herdId$)

הסוס בעל המזהה $horseId$ מצטרף לעדר בעל המזהה $herdId$, במידה ולפני כן לא היה באף עדר. בעת התוספו לעדר אף סוס אחר אינו עוקב אחריו והוא לא עוקב אחרי אף סוס.

פרמטרים:

$horseId$	מזהה הסוס שמצטרף לעדר.
$herdId$	מזהה העדר אליו הסוס מצטרף.

ערך החזרה:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId$<=0 או אם $herdId$ <= 0.
FAILURE	אם אין סוס עם מזהה $horseId$ הסוס הזה כבר נמצא בעדר, או שאין עדר במזהה $herdId$.
SUCCESS	במקרה של הצלחה.

סיבוכיות זמן: $O(\log n + \log m + \log m_{\emptyset})$ במקרה הגרוע.

StatusType follow(int $horseId$, int $horseToFollowId$)

הסוס בעל המזהה $horseId$ מתחיל לעקוב אחרי הסוס במזהה $horseToFollowId$, במידה והם נמצאים באותו עדר.

אם הסוס העוקב כבר עקב אחרי סוס אחר, הוא מפסיק לעקוב אחריו.

במידה והסוס אחריו עוקבים עוזב את העדר, כלל הסוסים העוקבים ישירות אחריו מפסיקים לעקוב אחריו, ואינם יחזרו לעקוב אחריו, אלה אם תתבצע פעולת follow נוספת שתגרום להם לחזור לעקוב אחריו מפורשות.

<u>פרמטרים</u>:

$horseId$	מזהה הסוס שמתחיל לעקוב.
$horseToFollowId$	מזהה הסוס אחריו מתחילים לעקוב.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId <= 0$ או אם $horseToFollowId <= 0$ או אם $horseId == horseToFollowId$.
FAILURE	אם אין סוס עם מזהה $horseId$, אין סוס במזהה $horseToFollowId$ או שהסוסים הללו אינם נמצאים באותו עדר.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע.

StatusType leave_herd(int $horseId$)

הסוס בעל המזהה $horseId$ עוזב את העדר שלו, הוא מפסיק לעקוב אחרי סוס אחר אם עשה זאת עד הקריאה לפונקציה וכל סוס שעקב אחריו עד הקריאה לפונקציה כבר לא עוקב אחרי אף אחד.

<u>פרמטרים</u>:

$horseId$	מזהה הסוס שעוזב את העדר שלו.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId <= 0$.
FAILURE	אם אין סוס עם מזהה $horseId$ או אם הסוס במזהה $horseId$ אינו באף עדר.
SUCCESS	במקרה של הצלחה.

<u>סיבוכיות זמן</u>: $O(\log n + \log m + \log m_{\emptyset})$ במקרה הגרוע.

---

output_t $< int >$ get_speed(int $horseId$)

מחזיר את מהירות הסוס במזהה $horseId$.

<u>פרמטרים</u>:

$horseId$	מזהה הסוס שאת מהירותו יש להחזיר.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId <= 0$.
FAILURE	אם אין סוס במזהה $horseId$.
SUCCESS	במקרה של הצלחה, במקרה זה תוחזר גם מהירות הסוס.

<u>סיבוכיות זמן</u>: $O(\log n)$ במקרה הגרוע.

output_t < $bool$ > leads(int $horseId$, int $otherHorseId$)

בודק האם קיימת שרשרת של סוסים כך שהסוס במזהה $horseId$ עוקב (בעקיפין) אחרי הסוס במזהה $otherHorseId$.

פורמלית אם קיימים סוסים:

סוס[1], סוס[2], ..., סוס[k]

כך שסוס[1] בעל המזהה $horseId$

סוס[k] בעל המזהה $otherHorseId$

ולכל $2 \leq i \leq k$:

סוס[i-1] עוקב (follows) אחרי סוס[i]

<u>פרמטרים</u>:

$horseId$	מזהה הסוס הראשון בשרשרת
$otherHorseId$	מזהה הסוס המוביל בשרשרת

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $horseId$<=0 או אם $otherHorseId$<=0 או אם המזהים זהים, כלומר $otherHorseId==horseId$.
FAILURE	אם אין סוס במזהה $horseId$ או אין סוס במזהה $otherHorseId$.
SUCCESS	במקרה של הצלחה, במקרה זה תוחזר גם התשובה לשאלה.

<u>סיבוכיות זמן</u>: ($O(\log n + n_{her}$ במקרה הגרוע, כאשר n_{herdId} הינו מספר הסוסים בעדר של הסוס בעל המזהה $horseId$.

---

output_t < $bool$ > can_run_together(int $herdId$)

נאמר שעדר יכול לרוץ יחד אם קיים סוס יחיד בעדר שמוביל (מוביל כמו בפונקציית leads, בשונה מ"עוקב") את כל שאר הסוסים ואף סוס אינו מוביל אותו. הפעולה מחזירה סטאטוס וגם את התשובה על השאלה האם העדר במזהה $herdId$ יכול לרוץ יחד.

פורמלית, אם נסמן את הסוסים בעדר במזהה $herdId$ בתור:

סוס[1], סוס[2], ..., סוס[k]

העדר יכול לרוץ יחד אם קיים j כך שלכל $1 \leq i \leq k$ עבור $i \neq j$ מתקיים:

סוס[j] מוביל (*leads*) את סוס[i].

סוס[i] **אינו** מוביל (*leads*) את סוס[j].

הערה: עדר עם סוס אחד בלבד יכול תמיד לרוץ יחד.

<u>פרמטרים</u>:

$herdId$	מזהה העדר אותו צריך לבדוק אם הוא מסוגל לרוץ יחד.

<u>ערך החזרה</u>:

ALLOCATION_ERROR	במקרה של בעיה בהקצאה/שחרור זיכרון.
INVALID_INPUT	אם $herdId$<=0.
FAILURE	אם אין עדר עם מזהה $herdId$ או שעדר זה קיים אך הוא ריק.
SUCCESS	במקרה של הצלחה, במקרה זה תוחזר גם התשובה לשאלה.

<u>סיבוכיות זמן</u>: $O(\log m + n_{herdId})$ במקרה הגרוע, כאשר n_{herdId} הינו מספר הסוסים בעדר בעל המזהה $herdId$.

דוגמה הרצה:

add_horse(1, 100)
add_horse(2, 200)
add_horse(3, 300)
add_herd(1)
join_herd(1, 1)
join_herd(2, 1)
join_herd(3, 1)
מצב המערכת herd1: [1, 2, 3]
follow(1, 2)
can_run_together(1) מחזיר false, כי 2 ו3 שניהם אינם עוקבים.
follow(2, 3)
can_run_together(1) מחזיר true
מצב המערכת herd1: [1 -> 2 -> 3]
Leave_herd(3)
מצב המערכת herd1: [1 -> 2] no herd: [3]
Join_herd(3, 1)
מצב המערכת herd1: [1 -> 2, 3]
can_run_together(1) מחזיר false, כי 2 ו3 שניהם אינם עוקבים.
follow(2, 1)
can_run_together(1) מחזיר false כי 3 אינו עוקב אך לא מוביל את 1 וגם לא מוביל את 2.
follow(2, 3)
מצב המערכת herd1: [1 -> 2 -> 3]
can_run_together(1) מחזיר true.
leads(1, 3) מחזיר true.
leads(2, 1) מחזיר false.
get_speed(1) מחזיר 100.
get_speed(2) מחזיר 200.
get_speed(3) מחזיר 300.

סיבוכיות מקום:

סיבוכיות המקום הדרושה עבור מבנה הנתונים היא $O(n + m + m_{\emptyset})$ במקרה הגרוע. כלומר בכל רגע בזמן הריצה, צריכת המקום של מבנה הנתונים תהיה לינארית בסכום מספרי הסוסים והעדרים במערכת.

סיבוכיות המקום הנדרשת עבור כל פעולה (כלומר, זיכרון "העזר" שכל פעולה משתמשת בו) אינה מצוינת לכל פעולה לחוד, אך אסור לעבור את סיבוכיות המקום הדרושה שמוגדרת לכל המבנה.

ערכי החזרה של הפונקציות:

כל אחת מהפונקציות מחזירה ערך מטיפוס StatusType שייקבע לפי הכלל הבא:

- תחילה, יוחזר INVALID_INPUT אם הקלט אינו תקין.
- אם לא הוחזר INVALID_INPUT:
  - בכל שלב בפונקציה, אם קרתה שגיאת הקצאה/שחרור יש להחזיר ALLOCATION_ERROR. מצב זה אינו צפוי אלא באחד משני מקרים (לרוב): באמת השתמשתם בקלט גדול מאוד ולכן המבנה ניצל את כל הזיכרון במערכת, או שיש זליגת זיכרון בקוד.
  - אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד FAILURE מבלי לשנות את מבנה הנתונים.
- אחרת, יוחזר SUCCESS.

חלק מהפונקציות צריכות להחזיר בנוסף עוד פרמטר (int או bool), לכן הן מחזירות אובייקט מטיפוס output_t<T>.
אובייקט זה מכיל שני שדות: הסטטוס (__status) ושדה נוסף (__ans) מסוג T.

במקרה של הצלחה (SUCCESS), השדה הנוסף יכיל את ערך החזרה, והסטטוס יכיל את SUCCESS. בכל מקרה אחר, הסטטוס יכיל את סוג השגיאה והשדה הנוסף לא מעניין.

שני הטיפוסים (output_t<T>,StatusType) ממומשים כבר בקובץ "wet1util.h" שניתן לכם כחלק מהתרגיל.

קייסם קונסטרקטור של output_t<T> מ-T ומ-StatusType כך שניתן פשוט לכתוב בפונקציות הרלוונטיות:

```
return 7;
return StatusType::FAILURE;
```

<u>הנחיות ודגשים כלליים:</u>

<u>חלק יבש:</u>

- החלק היבש הווה חלק מהציון על התרגיל כפי שמצוין בנהלי הקורס.
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- החלק היבש חייב להיות מוקלד.
- הגשת החלק הרטוב מהווה תנאי הכרחי לקבלת ציון על החלק היבש, כלומר, הגשה בה יתקבל אך ורק חלק יבש תגרור ציון 0 על התרגיל כולו.
- יש להכין מסמך הכולל תיאור של מבני הנתונים והאלגוריתמים בהם השתמשתם בצירוף הוכחת סיבוכיות הזמן והמקום שלהם. חלק זה עומד בפני עצמו וצריך להיות מובן לקורא גם לפני העיון בקוד. אין צורך לתאר את הקוד ברמת המשתנים, הפונקציות והמחלקות, אלא ברמה העקרונית. חלק יבש זה <u>לא</u> תיעוד קוד.
- ראשית הציגו את מבני הנתונים בהם השתמשתם. רצוי ומומלץ להיעזר בציור.
- לאחר מכן הסבירו כיצד מימשתם כל אחת מהפעולות הנדרשות. הוכיחו את דרישות סיבוכיות הזמן של כל פעולה תוך כדי התייחסות לשינויים שהפעולות גורמות במבני הנתונים.
- הוכיחו שמבנה הנתונים וכל הפעולות עומדים בדרישת סיבוכיות המקום.
- החסמים הנתונים בתרגיל הם לא בהכרח הדוקים ולכן יכול להיות שקיים פתרון בסיבוכיות טובה יותר. מספיק להוכיח את החסמים הדרושים בתרגיל.
- רמת פירוט: יש להסביר את כל הפרטים שאינם טריוויאליים ושחשובים לצורך מימוש הפעולות ועמידה בדרישות הסיבוכיות. אין לדון בפרטים טריוויאליים (הפעילו את שיקול דעתכם בקשר לזה, ושאלו את האחראי על התרגיל אם אינכם בטוחים). אין לצטט קטעים מהקוד כתחליף להסבר. אין צורך לפרט אלגוריתמים שנלמדו בכתה. כמו כן, אין צורך להוכיח תוצאות ידועות שנלמדו בכתה, אלא מספיק לציין בבירור לאיזו תוצאה אתם מתכוונים. אין (וגם אין צורך) להשתמש בתוצאות של עצי דרגות והלאה.
- **על חלק זה לא לחרוג מ-8 עמודים.**
- והכי חשוב **keep it simple!**

<u>חלק רטוב:</u>

- <u>מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.</u>
- אנו ממליצים בחום על מימוש **Object Oriented**, ב**C++**, מימוש כזה יאפשר לכם להגיע לפתרון פשוט וקצר יותר לפונקציות אותן עליכם לממש ויאפשר לכם להכליל בקלות את מבני הנתונים שלכם (זכרו שיש תרגיל רטוב נוסף בהמשך הסמסטר).
- פקודת הקימפול שמורצת בgradescope הינה: g++ -std=c++11 -DNDEBUG -Wall -o main.out *.cpp
- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ plains25a1.h.
- אין לשנות את הקבצים main25a1.cpp ו-wet1util.h אשר סופקו כחלק מהתרגיל, ואין להגיש אותם. ישנה בדיקה אוטומטית שאין בקוד שימוש בSTL, ובדיקה זו נופלת אם מגישים גם את main25a1.cpp.
- את שאר הקבצים ניתן לשנות, ותוכלו להוסיף קבצים נוספים כרצונכם, ולהגיש אותם.
- העיקר הוא שהקוד שאתם מגישים יתקמפל עם הפקודה לעיל, כאשר מוסיפים לו את שני הקבצים main25a1.cpp ו-wet1util.h.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל **אין להשתמש במבנים של STL** ואין להוריד מבני נתונים מהאינטרנט). **כחלק מתהליך הבדיקה אנו נבצע בדיקה ידנית של הקוד ונוודא שאכן מימשתם את מבני הנתונים שבהם השתמשתם.**
  - בפרט, אסור להשתמש ב-std::iterator, std::vector, std::pair, או כל אלגוריתם של STL, רשימה מלאה של הספריות להן אסור לעשות include נמצאת בקובץ dont_include.txt.
  - **ניתן** להשתמש במצביעים חכמים (Smart pointers כמו shared_ptr), בספריית math או בספריית exception.

- חשוב לוודא שאתם מקצים/משחררים זיכרון בצורה נכונה (מומלץ לוודא עם valgrind). לא חייבים לעבוד עם מצביעים חכמים, אך אם אתם מחליטים כן לעשות זאת, לוודא שאתם משתמשים בהם נכון. (תזכרו שהם לא פתרון קסם, למשל, כאשר יוצרים מעגל בהצבעות)
- שגיאות של ALLOCATION_ERROR בד״כ מעידות על זליגה בזיכרון.
- על הקוד להתקמפל ולעבור את כל הבדיקות שמפורסמות לכם ב-gradescope. הטסטים שמורצים באתר מייצגים את הבדיקת אותן נריץ בנתינת הציון, כאשר פרסמנו 5 מתוך 50.
- אותם טסטים שבgradescope גם מפורסמים כקבצי קלט ופלט, יחד עם סקריפט בשם run_tests.py שנכתב בשביל python 3.6 ומעלה, המאפשר לבדוק את הקוד שלכם. מומלץ לבדוק את התרגיל לוקאלית לפני שמגישים.
- **<u>שימו לב</u>:** התוכנית שלכם תיבדק על קלטים רבים ושונים מקבצי הדוגמא הנ"ל. יחד עם זאת הטסטים האלו מייצגים מבחינת אורך ואופן היצירה שלהם את השאר.

<u>אופן ההגשה:</u>

הגשת התרגיל הנה דרך אתר ה-gradescope של הקורס.

<u>חלק הרטוב:</u>

יש להגיש קובץ **ZIP** שמכיל רק את קבצי הקוד שלכם (לרוב קבצי cpp. ,h. בלבד)

<u>חלק היבש:</u>

יש להגיש קובץ PDF אשר מכיל את הפתרון היבש. החלק היבש חייב להיות מוקלד.

- **<u>שימו לב כי אתם מגישים את כל שני החלקים הנ"ל, במטלות השונות.</u>**
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב. ההגשה האחרונה היא הנחשבת.
- הערכת הציון שמופיעה ב-gradescope אינה ציונכם הסופי על המטלה. הציון הסופי יתפרסם רק לאחר ההגשות המאוחרות של משרתי המילואים.
- במידה ואתם חושבים שישנה תקלה מהותית במערכת הבדיקה ב-gradescope נא להעלות זאת בפורום הפיאצה ונתפל בה בהקדם.

<u>דחיות ואיחורים בהגשה:</u>

- דחיות בתרגיל הבית תינתנה אך ורק לפי תקנון הקורס.
- 5 נקודות יורדו על כל יום איחור בהגשה ללא אישור מראש. באפשרותכם להגיש תרגיל באיחור של עד 5 ימים ללא אישור. תרגיל שיוגש באיחור של יותר מ-5 ימים ללא אישור מראש יקבל 0.
- במקרה של איחור בהגשת התרגיל יש עדיין להגיש את התרגיל אלקטרונית דרך אתר הקורס.
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת jonathan.gal@campus.technion.ac.il. לאחר קבלת אישור במייל על הבקשה, מספר הימים שאושרו לכם נשמר אצלנו. לכן, אין צורך לצרף להגשת התרגיל אישורים נוספים או את שער ההגשה באיחור.

בהצלחה!