מבני נתונים 1 - תרגיל רטוב 2

מתרגלת ממונה על התרגיל: אנה בנדרסקי, annaben@cs.technion.ac.il

תאריך ושעת הגשה: 17.6.2009 בשעה 14:00.

אופן ההגשה: בזוגות.

הנחיות:

- שאלות לגבי דרישות התרגיל יש להפנות באימייל לכתובת הנ"ל.
- תשובות לשאלות המרכזיות אשר ישאלו יתפרסמו בחוצץ ה FAQ באתר הקורס לטובת כלל הסטודנטים.

  שימו לב כי **תוכן ה FAQ הוא מחייב וחובה לקרוא אותו**, אם וכאשר הוא יתפרסם.

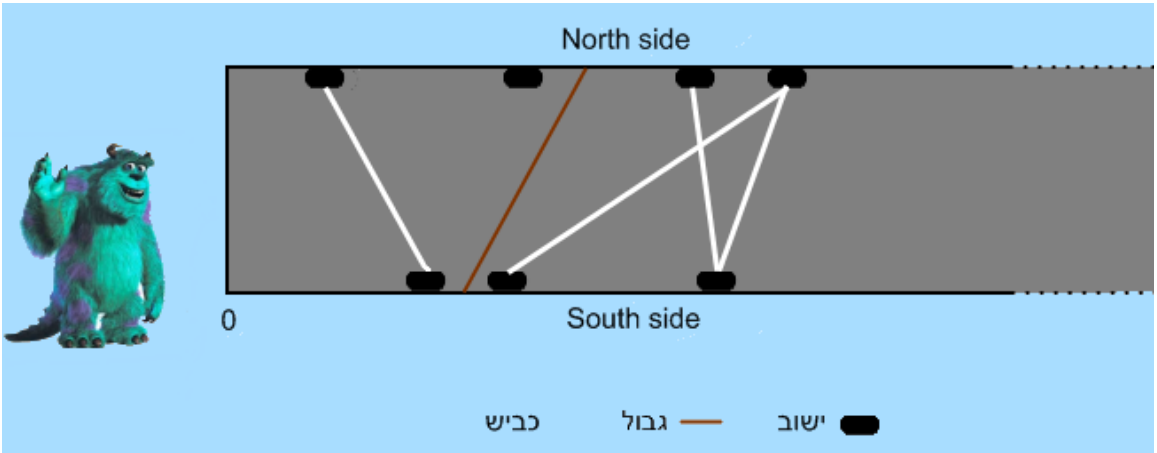
  לא יתקבלו דחיות או ערעורים עקב אי קריאת ה FAQ.

הקדמה:

ארץ אגדות רחוקה בשם RectangleLand היא ארץ בצורת מלבן (אינסופי), מוקפת ים – ראו איור.

בארץ זו, כבכל הארצות: קמים יישובים, אל היישובים מתווספים תושבים חדשים, נבנים ונהרסים כבישים וגבולות, ומידי פעם מגיעות מפלצות נוראיות שאוכלות את התושבים.

עליכם לממש מבנה נתונים המדמה את ארץ האגדות RectangleLand.



חלק א'

דרוש מבנה נתונים למימוש הפעולות הבאות:

```
void*        Init(int ma)
```

מאתחל מבנה נתונים שהוא ארץ אגדות ריקה.

פרמטרים: ma משנה שנועד עבור המפלצת. חייב להיות גדול מ 0.

ערך החזרה: מצביע למבנה נתונים ריק או NULL במקרה של כשלון (אם ma<=0 יש להחזיר NULL).

סיבוכיות: $O(1)$ במקרה הגרוע.

```
StatusType  AddTown(void* DS, Shore side, int location,
                    int maxNeighborhoods)
```

הפעולה מוסיפה יישוב חדש.

פרמטרים:
- DS מצביע למבנה הנתונים.
- side הגדה עליה ממוקם היישוב (צפון/דרום)
- location מציין את מיקום היישוב: מרחק מקו האפס (מספר שלם)
- maxNeighborhoods מספר השכונות המקסימלי ביישוב זה

ערך החזרה:
- INVALID_INPUT אם DS==NULL, location<0 או maxNeighborhoods<0
- FAILURE אם עובר גבול בנקודה זו, אם קיים יישוב בנקודה זו או במקרה של כל שגיאה אחרת.
- SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר היישובים.

```
StatusType  AddRoad(void* DS, int northTown, int southTown)
```

הוסף כביש דו סטרי בין שני היישובים. הכביש הוא קו ישר. מותר שכביש יחצה כביש, אך אסור שכביש יחצה גבול.
הישוב northTown נמצא על הגדה הצפונית, והישוב southTown נמצא על הגדה הדרומית.

פרמטרים:
- DS מצביע למבנה הנתונים.
- northTown מיקום היישוב הנמצא על הגדה הצפונית.
- southTown מיקום היישוב הנמצא על הגדה הדרומית.

ערך החזרה:
- INVALID_INPUT אם DS==NULL, או אחד המיקומים הנ"ל שלילי (קטן ממש מ 0).
- FAILURE אם היישובים לא קיימים, יש ביניהם כביש, הכביש יחצה גבול אם ייבנה, או כל תקלה אחרת.
- SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר היישובים.

```
StatusType  RemoveRoad(void* DS, int northTown, int southTown)
```

הכביש בלוי מכדי שימוש, ולכן מעתה הוא נחשב כלא קיים.

פרמטרים: DS מצביע למבנה הנתונים.

northTown מיקום היישוב הנמצא על הגדה הצפונית.

southTown מיקום היישוב הנמצא על הגדה הדרומית.

ערך החזרה: INVALID_INPUT אם DS==NULL, או אחד המיקומים הנ"ל שלילי (קטן ממש מ 0).

FAILURE אם היישובים לא קיימים, אין ביניהם כביש, או כל תקלה אחרת.

SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר היישובים.

```
StatusType  AddBorder(void* DS, int northPt, int southPt)
```

הוסף גבול ישר העובר בין נקודת northPt לנקודת southPt. ניתן לבנות גבול אם הוא לא חוצה אף גבול קיים ואף כביש קיים. גבולות המשיקים בנקודה אחת לא נחשבים לחוצים זה את זה.

בנוסף, אסור שיהיו יותר גבולות מישובים.

פרמטרים: DS מצביע למבנה הנתונים.

northPt הנקודה על הגדה הצפונית ממנה הגבול יתחיל

southPt הנקודה על הגדה הדרומית בה הגבול יסתיים

ערך החזרה: INVALID_INPUT אם DS==NULL, northPt<0 או southPt<0.

FAILURE אם יש ישוב באחת הנקודות, או הגבול צפוי לחצות גבול/כביש קיים. אם בניית גבול זה תיצור יותר גבולות מישובים, או במקרה של כל בעיה אחרת

SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר היישובים.

```
StatusType  RemoveBorder(void* DS, int northPt, int southPt)
```

הסרת גבול קיים.

פרמטרים: DS מצביע למבנה הנתונים.

northPt הנקודה על הגדה הצפונית ממנה הגבול מתחיל

southPt הנקודה על הגדה הדרומית בה הגבול מסתיים

ערך החזרה: INVALID_INPUT אם DS==NULL, northPt<0 או southPt<0.

FAILURE אם אין גבול כזה, או במקרה של כל בעיה אחרת.

SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log n)$ במקרה הגרוע, כאשר n הוא מספר היישובים.

```
void        Quit(void** DS)
```

משחרר את מבנה הנתונים. בסוף השחרור יש להציב ערך NULL ב DS.

<u>פרמטרים</u>: DS מצביע למבנה הנתונים.

<u>ערך החזרה</u>: אין.

חלק ב'

עליכם להוסיף למבנה הנתונים שכתבתם בסעיף א' את הפעולות הבאות:

```
StatusType  AddNeighborhood(void* DS, Shore side, int location,
                            int population)
```

מוסיף שכונה ליישוב הנמצא בצד side במיקום location עם אוכלוסייה population.

<u>פרמטרים</u>: DS מצביע למבנה הנתונים.

side הגדה עליה ממוקם היישוב (צפון/דרום)

location מציין את מיקום היישוב: מרחק מקו האפס (מספר שלם)

population גודל האוכלוסיה בשכונה

<u>ערך החזרה</u>: INVALID_INPUT אם DS==NULL או location<0 או population<=0.

FAILURE אם היישוב לא קיים, הוספת שכונה זו יגרום לכך שיהיו יותר מידי שכונות בישוב, או במקרה של כל בעיה אחרת.

SUCCESS במקרה של הצלחה.

<u>סיבוכיות</u>: $O(\log(n) + \log(m))$ במקרה הגרוע,

כאשר n הוא מספר היישובים ו m הוא מספר השכונות בישוב.

```
StatusType  AddManyNeighborhoods(void* DS, Shore side, int location,
                                 int size, int* populations)
```

מוסיף שכונות רבות ליישוב הנמצא בגדה side במיקום location.

<u>פרמטרים</u>: DS מצביע למבנה הנתונים.

side הגדה עליה ממוקם היישוב (צפון/דרום)

location מציין את מיקום היישוב: מרחק מקו האפס (מספר שלם)

size מספר השכונות הנוספות כעת

populations גודל האוכלוסיה בשכונות.

<u>ערך החזרה</u>: INVALID_INPUT אם DS==NULL, location<0, populations==NULL, size<=0 או אחד המזהים בוקטור לא חיובי.

	FAILURE	אם אין ישוב במיקום זה, הוספת שכונה זו יגרום לכך שיהיו יותר מידי שכונות בישוב, או במקרה של כל בעיה אחרת.
	SUCCESS	במקרה של הצלחה.

סיבוכיות: $O(\log n + m)$ במקרה הגרוע, כאשר n הוא מספר הישובים ו m הוא מספר השכונות בישוב זה אחרי ההוספה .

```
StatusType  MonsterAttack(void* DS, Shore side, int location,
                          int* population)
```

מפלצת מתקיפה את היישוב townld, הורסת את השכונה ה-ma בגודלה (לפי מספר התושבים) ואוכלת את כל תושביה. הקבוע ma נקבע בפעולת ה Init, או בפעולת ה ChangeMa.

פרמטרים:	DS	מצביע למבנה הנתונים.
	side	הגדה עליה ממוקם היישוב (צפון/דרום)
	location	מציין את מיקום היישוב שנפל קורבן להתקפה
	population	ערך החזרה: מספר התושבים שהמפלצת זללה.
ערך החזרה:	INVALID_INPUT	אם DS==NULL או location<0
	FAILURE	אם היישוב לא קיים, יש בו מעט מידי שכונות, או כל בעיה אחרת.
	SUCCESS	במקרה של הצלחה.

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

כאשר n הוא מספר הישובים ו m הוא מספר השכונות בישוב זה לפני ההתקפה.

```
StatusType  ChangeMa(void* DS, int ma)
```

משנה את הקבוע ma, שתפקידו הוסבר בפונקציה MonsterAttack.

פרמטרים:	DS	מצביע למבנה הנתונים.
	ma	ערך חדש לקבוע
ערך החזרה:	INVALID_INPUT	אם DS==NULL או ma<=0.
	FAILURE	כל בעיה אחרת.
	SUCCESS	במקרה של הצלחה.

סיבוכיות: $O(n + M)$ במקרה הגרוע.

כאשר n הוא מספר הישובים ו M הוא מספר השכונות הכולל בכל היישובים.

סיבוכיות מקום

סיבוכיות המקום של מבנה הנתונים היא $O(n^2 + M)$, כאשר n מספר היישובים, Ml מספר השכונות המקסימלי הכולל האפשרי בכל הישובים.

ערכי החזרה של הפונקציות:

בכל אחת מהפונקציות, ערך ההחזרה שיוחזר יקבע לפי הכלל הבא:

- תחילה, יוחזר `INVALID_INPUT` אם הקלט אינו תקין.
- אם לא הוחזר `INVALID_INPUT`, אם קרתה שגיאה אחרת, כפי שמצוין בכל פונקציה, יש להחזיר מיד `FAILURE` מבלי לשנות את מבנה הנתונים.
- אחרת יוחזר `SUCCESS`.

חלק יבש:

- **הציון על החלק היבש הוא 50% מציון התרגיל.**
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזכרון שלעיל.
- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- 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.
- ראשית הציגו את מבני הנתונים בהם השתמשתם. **רצוי ומומלץ להעזר 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.
- 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 אותי 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.
- על חלק זה לא לחרוג על 4 דפים מודפסים משני צידי הדף.
- יש להגיש העתק מודפס של הקוד שלכם יחד עם החלק היבש. ניתן ואף מומלץ להדפיס בפורמט מצומצם החוסך בדפים.

חלק רטוב:

- **אנו ממליצים בחום על מימוש Object Oriented ב C++**. על מנת לעשות זאת הגדירו מחלקה, נאמר RectangleLand, וממשו בה את דרישות התרגיל. אח"כ, על מנת לייצר התאמה לממשק ה C ב library2.h, ממשו את library2.cpp באופן הבא:

```
#include "library2.h"
#include "RectangleLand.h"

void* Init(int ma) {
     RectangleLand* DS = new RectangleLand(ma);
     return (void*)DS;
}

StatusType  AddRoad(void* DS, int northTown, int southTown){
      return ((RectangleLand *)DS)->AddRoad(northTown,southTown);
}
```

וכולי...

- ב- C או C++. על הקוד להתקמפל על t2 באופן הבא:
  - לתוכנית הכתובה ב- C:
    - **gcc –Wall -c *.c**
    - **g++ –Wall main2.cpp *.o**
  - לתוכנית ב-C++:
    - **g++ –Wall *.cpp**

  עליכם מוטלת האחריות לוודא קומפילציה של התכנית ב g++. אם בחרתם לעבוד בקומפיילר אחר , מומלץ לקמפל ב-g++ מידי פעם במהלך העבודה.

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ library2.h
- בעזרת הקובץ main2.cpp אתם יכולים לבדוק את הפונקציות שמימשתם.
- קראו היטב את שני הקבצים הנ"ל, כולל התיעוד, לפני תחילת העבודה.
- אין לשנות את הקובץ library2.h.
- הפונקציה Init מחזירה void* - כתובת בזכרון שבה נמצאים כל מבני הנתונים בהם אתם משתמשים. ניתן, לדוגמא, לשמור אותם ב-struct או class, שכתובתו תוחזר ע"י Init ותועבר לכל אחת מהפונקציות האחרות.
- עליכם לממש בעצמכם את כל מבני הנתונים (למשל אין להשתמש במבנים של STL ואין להוריד מבני נתונים מהאינטרנט).
- יש לתעד את הקוד בצורה נאותה וסבירה.
- מספר ימים לאחר פרסום התרגיל נפרסם דוגמאות של קבצי קלט וקבצי הפלט המתאימים להם.
- <u>שימו לב</u>: התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט ארוכים, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.

הגשה:

- **חלק יבש**:
  - יוגש לתיבת הקורס בקומה 1, ויכיל את המסמך היבש בצירוף דף שער הכולל את שמות המגישים, מספרי ת.ז., e-mail להבהרות וכן ה e-mail שממנו הוגש החלק הרטוב.
  - מומלץ להגיש את החלק היבש מודפס. במקרה של כתב לא קריא, כל התרגיל לא ייבדק.

- **חלק רטוב**:
  - את קבצי ה-Source Files יש להגיש באופן הבא:
  - יש ליצר ZIP של כל הקבצים ללא תיקיות וללא הקבצים שפורסמו (library2.h , main2.cpp) לשלוח דרך מערכת ה GR, ולעקוב אחר ההוראות.
  - אל קובץ ה ZIP יש לצרף קובץ PDF אשר מכיל את הפתרון לחלק היבש.
    שימו לב: מי שלא יגיש קובץ PDF זה לא יוכל לערער על החלק היבש.
  - אין להשתמש בתוכנת כיווץ אחרת מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים .
  - לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב. ההגשה האחרונה היא הנחשבת.
  - לאחר ההגשה הראשונה כל הגשה נוספת צריכה להתבצע מתוך אותו חשבון ממנו התבצעה ההגשה הראשונה.
  - **תנאי לקבלת ציון בתרגיל הוא התאמה בין החלק הרטוב לחלק היבש**

- **העתקות יטופלו בחומרה!**

דחיות ואיחורים בהגשה:

- למקרים חריגים (כדוגמת מילואים) ייתכן שתאושרנה דחיות במועד ההגשה. במקרים אלו יש לשלוח דוא"ל אליי מוקדם ככל האפשר, ובכל מקרה **לפני מועד הגשת התרגיל. לא תאושרנה דחיות בדיעבד.**
  את צילום המייל המאשר דחייה, ואישור המילואים, יש לצרף להגשת החלק היבש.
- 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 החלק היבש יש למסור את התרגיל לתא של אנה בקומה 5, **ולא** לתא הקורס. חובה לצרף את דף השער לאיחורים המופיע באתר הקורס, ובו לציין את תאריך ושעת ההגשה.
- במקרה של הגשה של החלק היבש והחלק הרטוב במועדים שונים, מועד הגשת התרגיל יהיה המאוחר מבין השניים.

בהצלחה!