מתרגל ממונה על התרגיל: תומר כהן, tomerco20@cs.technion.ac.il

תאריך ושעת הגשה: 15/01/2022 בשעה 23:55

אופן ההגשה: בזוגות. יורד ציון לתרגילים שיוגשו ביחידים בלי אישור מהמתרגל הממונה על התרגיל.

הנחיות כלליות:

- 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.
- לפני שאתם ניגשים לקודד את פתרונכם, ודאו כי יש לכם פתרון העומד בכל דרישות הסיבוכיות התרגיל. תרגיל שאינו עומד בדרישות הסיבוכיות יחשב כפסול.
- 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.
- שאלות על התרגיל יש לפרסם **באתר הפיאצה של הקורס:**
  piazza.com/technion.ac.il/fall2022/234218
- בקשות להגשה מאוחרת יש להפנות למתרגל האחראי בלבד בכתובת barakgahtan@cs.technion.ac.il.

הקדמה:

בתרגיל זה נדון שוב במשחקים מאתגרים בין סטודנטים על מנת לזכות במל"גים נחשקים ומומלצים, מזווית קצת שונה. הפעם אנו יודעים כי יש רק **k** קבוצות של שחקנים. ובתרגיל זה, נרצה לעזור לנשיא הטכניון לעקוב אחר המשחקים המתרחשים, כאשר כאן הנשיא מעוניין שלכל שחקן תהיה תוצאה, אותו הוא (הנשיא) יוכל לשנות כרצונו, תוצאה זו הינה בטווח 1 עד $\boldsymbol{scale}$ (כאשר מתקיים $\mathbf{1} \leq \boldsymbol{scale} \leq \mathbf{200}$, כלומר $\boldsymbol{scale}$ קבוע מבחינת שיקולי חישובים) . ובתור סטודנטים לקורס מבני נתונים, בחר בכם לעזור לו עם מערכת זו.

דרוש מבנה נתונים למימוש הפעולות הבאות:

void* init(int k, int scale)

מאתחל מבנה נתונים עם k קבוצות, כל קבוצה מוגדרת להיות ריקה בהתחלה.

פרמטרים: k — מספר הקבוצות במשחק.

scale — התוצאה המקסימלית במשחק.

ערך החזרה: מצביע למבנה נתונים ריק או $NULL$ במקרה של כישלון ($k \leq 0$, $scale \leq 0, scale > 200$ נחשב ככישלון).

סיבוכיות זמן: $O(k)$ במקרה הגרוע.

StatusType mergeGroups(void *DS, int GroupID1, int GroupID2)

הפעולה מאחדת בין שתי הקבוצות עם המזהים GroupID1, GroupID2, כל השחקנים משתי הקבוצות עוברים להיות בקבוצה החדשה. לאחר איחוד שתי הקבוצות, שני המזהים GroupID1, GroupID2 יתייחסו לקבוצה המאוחדת. לדוגמא, אם מאחדים את קבוצה מספר 3 עם קבוצה מספר 5, אז בעתיד כל התייחסות לקבוצה מספר 3 או 5 תהיה לקבוצה המאוחדת.

פרמטרים: DS — מצביע למבנה הנתונים.

GroupID1 — מזהה הקבוצה הראשונה.

GroupID2 — מזהה הקבוצה השנייה.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.

INVALID_INPUT — אם DS==NULL או $GroupID1 \leq 0, GroupID2 \leq 0$ או $GroupID1 > k, GroupID2 > k$.

SUCCESS — במקרה של הצלחה.

סיבוכיות זמן: $O(\log^*(k) + n)$ משוערך, בממוצע על הקלט, כאשר k הוא מספר הקבוצות ו-n מספר השחקנים בשתי הקבוצות המאוחדות.

StatusType addPlayer(void *DS, int PlayerID, int GroupID, int score)

הוספת שחקן חדש שמתחיל את המשחק משלב Level=0 במשחק, ועם תוצאה התחלתית score, ומשתייך לקבוצה בעלת המזהה GroupID.

פרמטרים: DS מצביע למבנה הנתונים.
PlayerID מזהה השחקן.
GroupID מזהה הקבוצה.
score התוצאה ההתחלתית של השחקן.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.
INVALID_INPUT אם DS==NULL, $GroupID \leq 0, GroupID > k$
או $PlayerID \leq 0$, $score \leq 0, score > scale$
FAILURE אם קיים כבר שחקן עם מזהה PlayerID.
SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log^*(k))$ משוערך, בממוצע על הקלט, כאשר k הוא מספר הקבוצות

StatusType removePlayer(void *DS, int PlayerID)

השחקן בעל המזהה PlayerID "נפסל" מהמשחק, וניתן למחוק אותו מהמערכת

פרמטרים: DS מצביע למבנה הנתונים.
PlayerID מזהה השחקן שיש להסיר מהמערכת.

ערך החזרה: INVALID_INPUT אם DS==NULL או $PlayerID \leq 0$.
FAILURE אם אין שחקן עם מזהה PlayerID.
SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log^*(k) + \log(n))$ משוערך, בממוצע על הקלט, כאשר k זה מספר הקבוצות, n זה מספר השחקנים הכולל במשחק כרגע.

StatusType increasePlayerIDLevel(void *DS, int PlayerID, int LevelIncrease)

הגדלת השלב במשחק של השחקן בעל המזהה PlayerID ב-LevelIncrease .

פרמטרים: DS מצביע למבנה הנתונים.
PlayerID מזהה השחקן שיש לעדכן.
LevelIncrease כמות שלבי המשחק שיש להוסיף לשחקן.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.
INVALID_INPUT אם DS==NULL, $PlayerID \leq 0$ או $\text{LevelIncrease} \leq 0$
FAILURE אם אין שחקן עם מזהה PlayerID.
SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log^*(k) + \log(n))$ משוערך, בממוצע על הקלט, כאשר k זה מספר הקבוצות, n זה מספר השחקנים הכולל במשחק כרגע.

בונוס (5 נק'): תארו בחלק היבש כיצד ניתן היה לממש את הפעולה הנ"ל בסיבוכיות של $O(\log^*(k) + \log(n))$ משוערך (שימו לב שכאן הסיבוכיות אינה בממוצע על הקלט).

StatusType changePlayerIDScore(void *DS, int PlayerID, int NewScore)

שינוי התוצאה של השחקן בעל המזהה PlayerID ל-NewScore.

פרמטרים: DS מצביע למבנה הנתונים.
PlayerID מזהה השחקן שיש לעדכן.
NewScore התוצאה החדשה של השחקן.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.
INVALID_INPUT אם DS==NULL, $PlayerID \leq 0$, $\text{NewScore} \leq 0, \text{NewScore} > scale$
FAILURE אם אין שחקן עם מזהה PlayerID.
SUCCESS במקרה של הצלחה.

סיבוכיות: $O(\log^*(k) + \log(n))$ משוערך, בממוצע על הקלט, כאשר k הוא מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.

StatusType getPercentOfPlayersWithScoreInBounds (void *DS, int GroupID, int score, int lowerLevel, int higherLevel, double * players)

הפעולה מחשבת את אחוז השחקנים בעלי תוצאה השווה בדיוק ל-score מבין השחקנים שנמצאים ברמה שהינה בטווח [lowerLevel, higherLevel], בקבוצה עם המזהה $GroupID$, אם $GroupID == 0$ אז הפעולה מחזירה את את אחוז השחקנים בעלי תוצאה השווה בדיוק ל-score יש מבין השחקנים שנמצאים ברמה שהינה בטווח [lowerLevel, higherLevel], מבין **כל השחקנים במשחק**.

לדוגמא, אם יש לנו את הקבוצה הבאה: 2 שחקנים ברמה 8 ובעלי תוצאות 1, 2. 4 שחקנים ברמה 6 ובעלי תוצאות 1, 3, 3, 4. 6 שחקנים ברמה 3 ובעלי תוצאות 1, 1, 3, 3, 4, 5. וקיבלנו להחזיר את אחוז השחקנים בין הרמות 2 עד 6 ובעלי תוצאה 3 (כלומר $lowerLevel = 2, higherLevel = 6, score = 3$). הערך שנחזיר יהיה 40. כי יש 10 שחקנים בין הרמות 2 עד 6, ומתוכם 4 שחקנים עם התוצאה 3.

פרמטרים: DS מצביע למבנה הנתונים.
GroupID מזהה הקבוצה שעבורה נרצה לקבל את המידע.
score התוצאה אותה אנו בודקים.
lowerLevel הרמה התחתונה שהחל ממנה אנו סופרים שחקנים.
higherLevel הרמה העליונה שעד אליה אנו סופרים שחקנים.
players מצביע למשנה שבו תוחזר התוצאה.

ערך החזרה: ALLOCATION_ERROR במקרה של בעיה בהקצאת זכרון.
INVALID_INPUT אם אחד המצביעים שווה ל-NULL או $GroupID < 0, GroupID > k$
FAILURE אם אין שחקנים בקבוצה עם המזהה $groupID$ שהרמה שלהם היא בתחום [lowerLevel, higherLevel]. אם $groupID = 0$ אז אם אין שחקנים במשחק בתחום [lowerLevel, higherLevel].
SUCCESS במקרה של הצלחה, כלומר כל מצב אחר.

סיבוכיות: $O(\log^*(k) + \log n)$ משוערך, כאשר k הוא מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.

הערה: אין הגבלה על הערך של score, lowerLevel, higherLevel כי מדובר על בעיית ספירה.

StatusType averageHighestPlayerLevelByGroup(void *DS, int GroupID, int m, double * avgLevel)

הפעולה מחשבת את הרמה הממוצעת בה נמצאים m השחקנים ברמות הגבוהות ביותר בקבוצה עם המזהה $GroupID$, אם $GroupID == 0$ אז הפעולה מחזירה את הרמה הממוצעת של m השחקנים ברמות הגבוהות ביותר מבין **כל השחקנים במשחק**.

לדוגמא, אם יש לנו את הקבוצה הבאה: 2 שחקנים ברמה 8. 4 שחקנים ברמה 6. 9 שחקנים ברמה 3 וקיבלנו להחזיר את הרמה הממוצעת של 9 השחקנים ברמות הגבוהות ביותר (כלומר קיבלנו $m = 9$), אזי avgLevel יציב לערך 5.444444, כי סכום הרמות של 9 השחקנים ברמות הגבוהות ביותר הינו 49 (2 שחקנים ברמה 8, 4 שחקנים ברמה 6, ו-3 שחקנים ברמה 3), ולכן נקבל $\frac{49}{9} = 5.44444$.

<u>פרמטרים</u>:	DS	מצביע למבנה הנתונים.
	GroupID	מזהה הקבוצה שעבורה נרצה לקבל את המידע.
	m	מספר השחקנים עבורם אנחנו רוצים לקבל את המידע.
	avgLevel	מצביע למשנה שבו תוחזר התוצאה.
<u>ערך החזרה</u>:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם אחד המצביעים שווה ל-NULL או $GroupID < 0, GroupID > k$ או $m \leq 0$.
	FAILURE	אם m גדול ממספר השחקנים בקבוצה עם המזהה $GroupID$ או ממספר השחקנים במשחק אם $GroupID == 0$.
	SUCCESS	במקרה של הצלחה, כלומר כל מצב אחר.
<u>סיבוכיות</u>:	$O(\log^*(k) + \log n)$ משוערך, כאשר k הוא מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.	

<u>בונוס (5 נקודות)</u>:

StatusType getPlayersBound(void *DS, int GroupID, int score, int m, int * LowerBoundPlayers, int * HigherBoundPlayers)

הפעולה מחשבת את הטווח של מספר השחקנים בעלי מספר נקודות השווה ל-score מבין m השחקנים ברמות הגבוהות ביותר בקבוצה עם המזהה $GroupID$, אם $GroupID == 0$ אז הפעולה מחזירה את הטווח של מספר השחקנים בעלי מספר נקודות השווה ל-score של m השחקנים ברמות הגבוהות ביותר מבין **כל השחקנים במשחק**.

כלומר, בהינתן m, אם ברמה הגבוהה ביותר יש $a_1 \leq m$ אנשים, נבחר את כל האנשים האלה לבדיקה, אם ברמה השנייה הכי גבוהה יש $a_2 \leq m - a_1$ אנשים, נבחר את כל האנשים האלה לבדיקה, וכן הלאה, עד שנגיע לקבוצה שגודלה מקיים $a_k > m - \sum_{i=1}^{k-1} a_i$, ממנה נבחר $m - \sum_{i=1}^{k-1} a_i$ אנשים (כמות האנשים שנשאר לנו לבחור), בחירת אנשים אלה יכולה להשפיע על טווח התוצאות האפשריות.

כלומר, בהינתן m, נגדיר קבוצה חוקית של m אנשים אמ"מ אין אדם <u>שאינו</u> בקבוצה ונמצא ברמה גדולה ממש מאדם שנמצא בקבוצה. ובפונקציה זו נרצה לבדוק מה הטווח של קבוצה עם m אנשים עם תוצאה ששווה ל-$score$ שמקיימת תנאי זה.

ובצורה יותר מתמטית, אם נסמן את האנשים בקבוצה שאנחנו בודקים בתור A, אזי בהניתן קבוצה B של m אנשים, נסמן את הרמה המינימלית של אדם בקבוצה זו בתור $minLevelB$. אזי הקבוצה היא חוקית אמ"מ מתקיים:

$$\forall a \in A \backslash B, level(a) \leq minLevelB$$

לדוגמא, אם יש לנו את הקבוצה הבאה: שחקן אחד ברמה 8 עם תוצאה 2. 4 שחקנים ברמה 6 עם תוצאות 4, 4, 2, 3. וקיבלנו להחזיר את טווח מספר השחקנים ברמה 4 מבין 3 השחקנים ברמות הגבוהות ביותר (כלומר קיבלנו $(m = 3, score = 4$, אזי LowerBoundPlayers יצביע לערך 0, HigherBoundPlayers יצביע לערך 2. כי השחקן ברמה 8 בהכרח יבחר, וצריך לבחור 2 שחקנים מרמה 6. אם נבחר את שני השחקנים עם התוצאות 3,2, נקבל 0 שחקנים עם תוצאה 4. אם נבחר את השחקנים עם התוצאות 4,4, נקבל 2 שחקנים תוצאה 4. עבור אותה דוגמה, רק כאשר $m = 4$ (כלומר מבין ארבעת השחקנים ברמות הגבוהות ביותר). LowerBoundPlayers יצביע לערך 1, HigherBoundPlayers יצביע לערך 2.

<u>פרמטרים</u>:	DS	מצביע למבנה הנתונים.
	GroupID	מזהה הקבוצה שעבורה נרצה לקבל את המידע.
	score	התוצאה אותה בודקים.
	m	מספר השחקנים עבורם אנחנו רוצים לקבל את המידע.
	LowerBoundPlayers	מצביע למשתנה שבו תוחזר תוצאת מספר השחקנים המינימלי.
	HigherBoundPlayers	מצביע למשתנה שבו תוחזר תוצאת מספר השחקנים המינימלי.
<u>ערך החזרה</u>:	ALLOCATION_ERROR	במקרה של בעיה בהקצאת זכרון.
	INVALID_INPUT	אם אחד המצביעים שווה ל-NULL או $m < 0$ או $GroupID < 0, GroupID > k$ $\text{score} \leq 0, \text{score} > scale$
	FAILURE	אם m גדול ממספר השחקנים בקבוצה עם המזהה $GroupID$ או ממספר השחקנים במשחק אם $GroupID == 0$.
	SUCCESS	במקרה של הצלחה, כלומר כל מצב אחר.
<u>סיבוכיות</u>:	$O(\log^*(k) + \log n)$ משוערך, כאשר k הוא מספר הקבוצות ו-n הוא מספר השחקנים הכולל במשחק כרגע.	

void Quit(void **DS)

הפעולה משחררת את המבנה. בסוף השחרור יש להציב ערך NULL ב-DS, אף פעולה לא תקרא לאחר מכן.

<u>פרמטרים</u>:	DS	מצביע למבנה הנתונים.
<u>ערך החזרה</u>:	אין.	
<u>סיבוכיות</u>:	$O(n + k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים ו-k הוא מספר הקבוצות.	

<u>סיבוכיות מקום</u> - $O(n + k)$ במקרה הגרוע, כאשר n הוא מספר השחקנים ו-k הוא מספר הקבוצות.

<u>הנחיות לגבי הסיבוכיות המשוערכת:</u>

- הסיבוכיות המשוערכת הנוגעת לחלק שנובע ממספר הקבוצות (k), היא משוערכת עבור כל הפעולות יחד.
  כלומר, לכל רצף של t פעולות מהקבוצה mergeGroups, averageHighestPlayerLevelByGroup, getPercentOfPlayersWithScoreInBounds, changePlayerIDScore, increasePlayerIDLevel, removePlayer, addPlayer, getPlayersBound
  הסיבוכיות הכוללת תהיה $O(t \cdot \log^*(k) + t \cdot f(n))$ כאשר $f(n)$ היא פונקציה כלשהי שתלויה רק במספר השחקנים (n)

- הסיבוכיות המשוערכת הנוגעת לחלק שנובע ממספר השחקנים (n), היא משוערכת עבור כל הפעולות יחד.
  כלומר, לכל רצף של t_1 פעולות מהקבוצה increasePlayerIDLevel, removePlayer,
  changePlayerIDScore, ו-t_2 פעולות של addPlayer.
  הסיבוכיות הכוללת תהיה $O(f(k) + t_1 \cdot \log(n) + t_2)$ כאשר $f(k)$ היא פונקציה כלשהי שתלויה רק במספר הקבוצות (k).

<u>ערכי החזרה של הפונקציות:</u>

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

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

הנחיות:

חלק יבש:

- **הציון על החלק היבש הוא 50% מהציון של התרגיל.**
- לפני מימוש הפעולות בקוד יש לתכנן היטב את מבני הנתונים והאלגוריתמים ולוודא כי באפשרותכם לממש את הפעולות בדרישות הזמן והזיכרון שלעיל.
- 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.
- 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.
- **על חלק זה לא לחרוג מ-8 עמודים.**
- והכי חשוב **keep it simple!**

חלק רטוב:

- מומלץ לממש תחילה את מבני הנתונים בצורה הכללית ביותר ורק אז לממש את הפונקציות הנדרשות בתרגיל.
- 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. על מנת לעשות זאת הגדירו מחלקה, נאמר PlayersManager, וממשו בה את דרישות התרגיל. אח"כ, על מנת לייצר התאמה לממשק ה C ב library2.h, ממשו את library1.cpp באופן הבא:

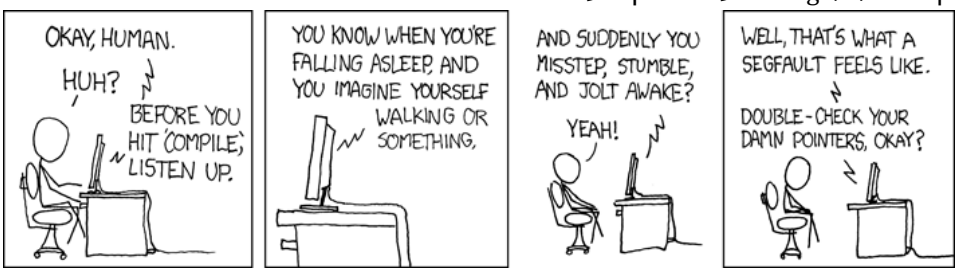
```
#include"library2.h"
#include"PlayersManager.h"

void* Init() {
        PlayersManager *DS = new PlayersManager();
        return (void*)DS;
}
StatusType AddGroup(void *DS, int GroupID){
        return ((PlayersManager*)DS)-> AddGroup (GroupID);
}
```

- על הקוד להתקמפל על csl3 באופן הבא:

g++ -std=c++11 -DNDEBUG -Wall *.cpp

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



הערות נוספות:

- חתימות הפונקציות שעליכם לממש ומספר הגדרות נמצאים בקובץ library2.h
- קראו היטב את הקובץ הנ"ל, לפני תחילת העבודה.
- אין לשנות את הקבצים אשר סופקו כחלק מהתרגיל, **ואין להגיש אותם**.
- 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.**
- יש לתעד את הקוד בצורה נאותה וסבירה.
- מצורפים לתרגיל קבצי קלט ופלט לדוגמא.
- שימו לב: התוכנית שלכם תיבדק על קלטים שונים מקבצי הדוגמא הנ"ל, שיהיו ארוכים ויכללו מקרי קצה שונים. לכן, מומלץ **מאוד** לייצר בעצמכם קבצי קלט, לבדוק את התוכנית עליהם, ולוודא שהיא מטפלת נכון בכל מקרה הקצה.

הגשה:

- **חלק יבש+ חלק רטוב**:

הגשת התרגיל הנה **אך ורק** אלקטרונית דרך אתר הקורס.
יש להגיש קובץ **ZIP** שמכיל את הדברים הבאים:

- בתיקייה הראשית:
  - קבצי ה-Source Files שלכם **(ללא הקבצים שפורסמו).**
  - קובץ PDF אשר מכיל את הפתרון היבש עבור. מומלץ להקליד את החלק הזה אך ניתן להגיש קובץ PDF מבוסס על סריקה של פתרון כתוב בכתב יד. שימו לב כי במקרה של כתב לא קריא, כל החלק השני לא תיבדק.
  - קובץ submissions.txt, המכיל בשורה הראשונה את שם, תעודת הזהות וכתובת הדוא"ל של השותף הראשון ובשורה השנייה את שם, תעודת הזהות וכתובת הדוא"ל של השותף השני. לדוגמה:

Tomer Cohen 012345678 tomerco20@cs.technion.ac.il
Henry Taub 123456789 taub@cs.technion.ac.il

- **שימו לב כי אתם מגישים את כל שלושת החלקים הנ"ל.**
- אין להשתמש בפורמט כיווץ אחר (לדוגמה RAR), מאחר ומערך הבדיקה האוטומטי אינו יודע לזהות פורמטים אחרים.
- **יש לוודא שכאשר נכנסים לקובץ הזיפ הקבצים מופיעים מיד בתוכו ולא בתוך תיקיה שבתוך קובץ הזיפ.** עבור הגשה שבה הקבצים יהיו בתוך תיקייה, הבדיקה האוטומטית לא תמצא את הקבצים ולא תוכל לקמפל ולהריץ את הקוד שלכם ולכן תיתן אוטומטית 0.
- לאחר שהגשתם, יש באפשרותכם לשנות את התוכנית ולהגיש שוב.
- ההגשה האחרונה היא הנחשבת.
- הגשה שלא תעמוד בקריטריונים הנ"ל תפסל ותקנס בנקודות!

דחיות ואיחורים בהגשה:

- 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 barakgahtan@cs.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!