תרגיל בית תכנותי
להגשה עד 26.01.20 בשעה 23:50
בהצלחה!

תרגיל זה מנוסח בלשון זכר מטעמי נוחות בלבד והוא מיועד לכל המגדרים.
מתרגל אחראי על התרגיל: שמעון

<u>הוראות:</u>

1. יש להגיש את **כל** קבצי ה header וה cpp שיצרתם בקובץ **zip יחיד** בעל השם EXP_ID1_ID2 כאשר ID1 ו ID2 הם מספרי תעודות הזהות של שני בני הזוג. אין צורך להגיש קבצים שסופקו ע"י צוות הקורס.
2. ההגשה תתבצע רק ע"י **אחד** מבני הזוג.
3. **עליכם לוודא לפני ההגשה כי הקוד שלכם מתקמפל ורץ בשרת ה t2** (הוראות מצורפות בקובץ נפרד).
4. זוג שהתרגיל שלו לא יתקמפל בשרת ה t2 או יעוף בזמן ריצה **ציונו בתרגיל יהיה 0.**
5. שימו לב כי יש לשחרר כל זיכרון שהקצתם. דליפת זיכרון תגרור הורדה בציון.
6. יש לכתוב קוד קריא ומסודר עם שמות משמעותיים למשתנים, למתודות ולמחלקות.
7. יש להקפיד למלא את כל דרישות התרגיל (שימוש בייצוג הנכון, הוספת include רק עבור קבצי מקור שבמפורש הוגדר שניתן להשתמש בהם, סיבוכיות זמן וכו') אי עמידה בדרישות התרגיל תגרור ציון 0.

בתרגיל בית זה אתם מתבקשים לממש בשפת c++ מבנה נתונים דינמי המאפשר לבצע פעולות על גרף מכוון פשוט בשם <u>Dynamic_Graph</u>.

<u>הוראות התרגיל:</u>

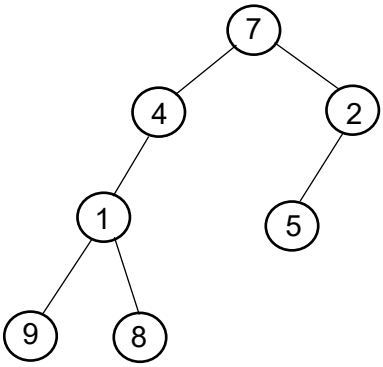
<u>הגדרות:</u>

1. לכל אורך התרגיל נסמן את הגרף המכוון הנוכחי המיוצג באמצעות המחלקה Dynamic_Graph באמצעות הסימון הסטנדרטי לגרף בקורס $G=(V,E)$. את מספר הצמתים ב G נסמן ב n ואת מספר הקשתות ב m .
2. בתרגיל בית זה לכל צומת בגרף (בין אם בגרף G או בעץ מושרש שתדרשו לממש) יש מזהה ייחודי המיוצג באמצעות מספר שלם חיובי.

תחילה עליכם לממש שלוש מחלקות:

1. מחלקה בשם **Graph_Node**.
מחלקה זו מייצגת צומת $v \in V$ בגרף ולה המתודות הפומביות הבאות:
   1. **unsigned Get_out_Degree() const** - מתודה המחזירה את דרגת היציאה של צומת v (כלומר, מספר הקשתות היוצאות מצומת v).
   סיבוכיות נדרשת $O(\deg_{out}(v))$.
   2. **unsigned Get_in_Degree() const** - מתודה המחזירה את דרגת הכניסה של צומת v (כלומר, את מספר הקשתות הנכנסות לצומת v).
   סיבוכיות נדרשת $O(\deg_{in}(v))$.
   3. **unsigned Get_key() const** - המתודה מחזירה את המזהה של הצומת.
   סיבוכיות זמן נדרשת - $O(1)$.
2. מחלקה בשם **Graph_Edge**.
3. מחלקה בשם **Rooted_Tree.**
מחלקה זו מייצגת עץ מושרש ולה המתודות הפומביות הבאות:
   1. **Rooted_Tree()** - בונה ברירת מחדל (default constructor) למחלקה Rooted_Tree היוצר עץ מושרש ריק, ללא צמתים.
   סיבוכיות נדרשת - $O(1)$.
   2. **~Rooted_Tree()** - הורס (destructor). המתודה מוחקת את העץ המושרש וכל המידע השמור בו.
   סיבוכיות נדרשת - $O(k)$, כאשר k מייצג את מספר הצמתים בעץ.
   3. **void Print_By_Layer(std::ostream& stream) const** - המתודה מקבלת רפרנס לאובייקט מסוג ostream ומדפיסה ל stream את המזהים הייחודיים של הצמתים תחת הדרישות הבאות:
      1. מזהים של צמתים השייכים לרמה i בעץ (צמתים בעומק i) יודפסו בשורה ה $i+1$ של stream.
      2. הדפסות של מזהים באותה הרמה יופרדו רק באמצעות פסיקים.
      3. הדפסה של מזהים של צמתים באותה הרמה מתבצעת משמאל לימין. פורמלית, לכל שני צמתים u ו v עם הורה משותף כך ש u הוא אח שמאלי (לאו דווקא ישיר) של v מתקיים:
      המזהה של u יודפס לפני המזהה של v . אם u' ו v' הם צאצאים של u ו v, בהתאמה ונמצאים באותה רמה בעץ, אז המזהה של u' יודפס לפני המזהה של v' .

סיבוכיות זמן נדרשת - $O(k)$, כאשר k מייצג את מספר הצמתים בעץ.
לדוגמא, הפעלת המתודה הנ"ל על הייצוג של העץ המושרש



צריכה להדפיס:

7
4,2
1,5
9,8

שימו לב כי אין פסיק בסוף השורה. ניתן לראות דוגמאות נוספות בפלט החוקי שקיבלתם.

4. **void Preorder_Print(std::ostream& stream) const -** המתודה מדפיסה ל stream את המזהים של צמתי העץ בסדר preorder (כפי שנלמד בתרגול 3) תחת הדרישה שהדפסת תתי העצים של צומת מתבצעת מהילד השמאלי ביותר לימני ביותר. ההדפסה מתבצעת בשורה אחת כאשר בין מזהים מפריד רק פסיק.

כעת אנו יכולים להגדיר את המחלקה Dynamic_Graph. המחלקה Dynamic_Graph משתמשת במחלקות Graph_Node ו Graph_Edge על מנת לממש את המתודות הפומביות הבאות:

- **Dynamic_Graph()** - בונה ברירת מחדל (default constructor) למחלקה Dynamic_Graph היוצר גרף דינמי חדש ריק, ללא קשתות וללא צמתים.
  סיבוכיות זמן נדרשת - $O(1)$.
- **~Dynamic_Graph()** - הורס (destructor). המתודה מוחקת את מבנה הנתונים המייצג את הגרף ואת כל המידע השמור בו.
  סיבוכיות זמן נדרשת - $O(n+m)$.
- **Graph_Node* Insert_Node(unsigned node_key)** – מתודה זו מכניסה צומת חדש לגרף עם מזהה ייחודי node_key ומחזירה מצביע לאובייקט מסוג Graph_Node המייצג את הצומת שהוכנס. הצומת החדש מתווסף לגרף ללא קשתות נכנסות או יוצאות.
  סיבוכיות זמן נדרשת – $O(1)$.

- **void Delete_Node(Graph_Node* node)** – אם לצומת המוצבע ע"י node יש קשתות יוצאות או נכנסות המתודה לא עושה כלום. אחרת, המתודה מוחקת את הצומת המוצבע ע"י node מהגרף.
  סיבוכיות זמן נדרשת - $O(1)$.
- **Graph_Edge* Insert_Edge(Graph_Node* from, Graph_Node* to)** - המתודה מוסיפה לגרף קשת מהצומת המוצבע ע"י from לצומת המוצבע ע"י to.
  סיבוכיות זמן נדרשת - $O(1)$.
- **void Delete_Edge(Graph_Edge* edge)** - המתודה מוחקת את הקשת המוצבעת ע"י edge מהגרף.
  סיבוכיות זמן נדרשת - $O(1)$.
- **Rooted_Tree* SCC() const** - המתודה מחשבת רכיבים קשירים היטב בגרף $G=(V,E)$. המתודה מחזירה מצביע לאובייקט מסוג Rooted_Tree. מבנה ה Rooted_Tree הוא כדלקמן:
  השורש הוא צומת וירטואלי (שלא קיים בגרף) עם מזהה ייחודי 0. תת העץ המושרש ע"י כל ילד של השורש הוא רכיב קשיר היטב בגרף G .
  סיבוכיות זמן נדרשת - $O(n+m)$.
- **Rooted_Tree* BFS(Graph_Node* source) const** - המתודה מחזירה עץ מסלולים קצרים מהצומת המוצבע ע"י source.
  סיבוכיות זמן נדרשת - $O(n+m)$.

עליכם לחשוב איך מחלקה תשתמש במחלקה אחרת, ואולי להגדיר מחלקות, משתנים ומתודות נוספות כרצונכם.

<u>דרישות:</u>

שימו לב, אי עמידה בדרישות אלו תגרור ציון 0.

1. שמות המחלקות והחתימות של המתודות הפומביות צריכים להופיע בקוד שלכם בדיוק כפי שמופיעים בקובץ זה. על מנת למנוע טעויות מצורף לתרגיל קובץ טקסט עם השמות הנדרשים ממנו תוכלו להעתיק.
2. הקוד שלכם יכול להכיל include **רק** עבור מחלקות אותן יצרתם ו
#include < cstddef >.
על מנת לייצג מצביע ל NULL השתמשו במילה השמורה NULL המוגדרת ב cstddef.
(בכל מקום שתצטרכו להשתמש ב NULL הוסיפו #include < cstddef >).
במחלקה Rooted_Tree (ורק במחלקה זו) עליכם להוסיף
#include <ostream>
3. העץ המושרש אותו עליכם לממש באמצעות המחלקה Rooted_Tree משתמש בייצוג left child right sibling כפי שנלמד בתרגול 3.
4. על מנת להקל על תהליך הבדיקה (והוידוא שהפלט תקין) עליכם לממש את המתודות הפומביות

- **Rooted_Tree* SCC() const**
- **Rooted_Tree* BFS(Graph_Node* source) const**

של המחלקה Dynamic_Graph באמצעות האלגוריתמים המתאימים שנלמדו בהרצאה.

כחלק מדרישה זו נוסיף את הדרישות הבאות:

1. אלגוריתמים BFS ו DFS מבצעים סריקה של רשימת השכנויות של צמתים (שורה 4 באלגוריתם BFS ושורה 4 בפרוצדורה DFS_VISIT). בהקשר זה נדרוש שסריקה של קשתות תתבצע מהקשת החדשה ביותר שנכנסה לגרף לקשת הישנה ביותר. בנוסף, אלגוריתם DFS מבצע גם סריקה של צמתים (שורה 5 באלגוריתם DFS). נדרוש שהסריקה תתבצע מהצומת החדש ביותר לצומת הישן ביותר.
דרישה זו מקבעת את סדר הגילוי של הצמתים ונקרא לסדר זה *הסדר המושרה* (כלומר, אם צומת v נמצא לפני צומת u בסדר המושרה אז BFS או DFS גילו את צומת v לפני צומת u).
2. העץ המושרש המוחזר ע"י המתודות הנ"ל צריך לקיים את התכונה הבאה: לכל שני צמתים u ו v בעץ שלהם הורה משותף, אם צומת v מופיע לפני צומת u בסדר המושרה אז צומת v הוא אח שמאלי (לאו דווקא ישיר) של צומת u.

<u>הסבר על הקבצים שקיבלתם:</u>

1. names.txt – קובץ טקסט ובו רשומות שמות המחלקות וחתימות המתודות הפומביות שעליכם לממש במבנה הנתונים כפי שהוגדרו בתחילת התרגיל (מקובץ זה תוכל להעתיק את השמות ואת החתימות על מנת לוודא שיש לכם את החתימה המדויקת הדרושה).
2. main.cpp – דוגמת הרצה.
3. main_output.txt – פלט חוקי לאחר הרצה של main.cpp בשרת ה t2.

<u>הנחות:</u>

- ניתן להניח כי לא תוכנס לגרף קשת שהיא לולאה עצמית.
- ניתן להניח כי לא תוכנס לגרף אותה הקשת יותר מפעם אחת.
- ניתן להניח כי בעת הפעלת המתודה

Insert_Node(unsigned node_key)

של המחלקה Dynamic_Graph הערך node_key הוא ייחודי כלומר, לא קיים צומת ב G עם המזהה node_key.

<u>הדרכה:</u>

אתם נדרשים לממש גרף דינמי המאפשר הכנסה ומחיקה של צמתים בסיבוכיות זמן $O(1)$.
חשבו האם הייצוגים של גרף שנלמדו מאפשרים זאת. במידה ולא, אילו שינויים ניתן לבצע בייצוג על מנת לתמוך בפעולות אלו בסיבוכיות הזמן הדרושה.

<u>הסבר על תהליך הבדיקה האוטומטית:</u>

אנחנו נריץ את הקבצים שלכם עם פונקציית main שונה מזאת שקיבלתם עם פרסום התרגיל. ב main הבדיקה ייתכן ויתווספו אובייקטים, הפעולות וסדר הפעולות ישתנה, גודל הקלט ישתנה, וכו'

במהלך הבדיקה יקומפלו **כל** הקבצים שהגשתם בתוספת main הבדיקה בשרת ה t2 עם הפקודה

```
g++ *.cpp
```

חשוב מאוד שתגישו את כל קבצי ה h וה cpp שיצרתם ותוודאו שהקוד מתקמפל בשרת. בהנחה והקובץ מתקפל, הקוד יורץ והפלט של התוכנית יושווה לפלט חוקי.

<u>המלצות:</u>

1. אל תשאירו את הבדיקה בשרת לרגע האחרון. ייתכן והקוד לא יתקמפל בשרת ותצטרכו לתקנו לפני ההגשה.
2. התחילו מבניית פעולות ההכנסה והמחיקה. רק לאחר שבדקתם שפעולות אלו עובדות נכון המשיכו במימוש שאר הפעולות.

<u>בהצלחה!</u>