מטלה מספר 1 - מימוש גרף על ידי רשימת שכנויות

יושרה אקדמית

במהלך העבודה על המטלות, מותר להתייעץ עם סטודנטים אחרים ולחפש מידע באינטרנט. עם זאת, חל איסור להעתיק קטעי קוד שלמים ממקורות חיצוניים, כולל סטודנטים אחרים, אתרי אינטרנט ומודלי בינה מלאכותית (כגון ChatGPT).

יש לדווח על כל עזרה שקיבלתם, בין אם מדובר בהתייעצות עם סטודנטים אחרים או במידע שנמצא באינטרנט, **יש לצרף את הפרומפטים (AI,** בהתאם לתקנון היושר של המחלקה. **במקרה של שימוש בכלי בינה מלאכותית שהוזנו ואת התשובות שהתקבלו**.

- מחלקות, בנאים, ++C, **מטרת המטלה:** הבנת החומר הנלמד בהרצאות הראשונות, כגון: ניהול זיכרון ב מפרקים, עצמים, מרחבי שמות, העברת פרמטרים לפונקציות, החזרת אובייקטים.
- **שימו לב!** במטלה **אסור** להשתמש בספרייה הסטנדרטית, ניתן להשתמש במערך. **זאת אומרת תצטרכו** *אינם זמינים* stack או vector **למשמ מבנה/י נתונים המתאים לצרכי המטלה**. כל המיכלים הסטנדרטיים כולל לכם במטלה זו.
- **ההגשה ביחידים**.

הוראות הגשה ב Moodle:

במערכת Moodle **קובץ טקסט למשל** (`submission.txt`) יש להגיש המכיל 3 שורות בפורמט הבא:

1. **תעודת זהות** – מספר תעודת הזהות של הסטודנט.
2. **קישור להגשה** – קישור למאגר ה GitHub-שבו נמצא הפרויקט.
3. **פרטי ה-commit האחרון** – המחרוזת המזהה של ה commit-האחרון (`commit hash`)

- דוגמה לקובץ הגשה תקין:

```
123456789
https://github.com/example-user/graph-assignment
e3f1c1a
```

גרפים הם חלק בלתי נפרד ממדעי המחשב. במהלך הלימודים נחשפתם לדרכים שונות לייצוג של גרפים (מטריצת שכנויות, רשימת שכנויות ועוד).

רשימת שכנויות – במטלה זו תממשו גרף באמצעות Adjacency List.

דרישות המטלה:

מימוש המחלקות הבאות:

הוסיפו את המחלקות במרחב שמות (namespace) חדש (קראו לו `graph`).

מחלקה בשם `Graph`:

הגרף יאותחל עם מספר קודקודים מסוים ולא ניתן יהיה לשנות את מספר הקודקודים בגרף. המחלקה תכיל את רשימת השכנויות וכן את הפונקציות הבאות:

1. פונקציה `addEdge` מקבלת שלושה מספרים שלמים: מקור, יעד ומשקל (ברירת המחדל למשקל היא 1). הפונקציה תוסיף צלע (לא מכוונת) לגרף.
2. פונקציה `removeEdge` מקבלת שני מספרים שלמים המייצגים צלע ומוחקת אותה מהגרף. אם הצלע לא קיימת, הפונקציה תזרוק חריגה.
3. פונקציה `print_graph` מדפיסה את הגרף בצורה כלשהי הגיונית לבחירתכם.
4. במידת הצורך, יש להוסיף בנאים\פונקציות עזר רלונטיות למחלקה.

מחלקה בשם `Algorithms`:

מחלקה זו תממש אלגוריתמים שונים על גרף לא מכוון. הכי קרוב שאפשר לאלו שלמדתם בקורס אלגוריתמים 1. המחלקה תכיל את הפונקציות הבאות:

1. פונקציה `bfs` – מקבלת גרף וקודקוד מקור ומחזירה גרף שהוא עץ מושרש (שהשורש הוא קודקוד המקור שהתקבל מסריקת BFS כמובן).
2. פונקציה `dfs` – מקבלת גרף וקודקוד ממנו תתחיל הסריקה ומחזירה גרף (עץ או יער) המתקבל לפי סריקת DFS. (עץ זה יכיל קודקודים מקוריים ורק צלעות מסוג tree edges).
3. פונקציה `dijkstra` – מקבלת גרף וקודקוד התחלה ומחזירה עץ ממושקל של מסלולים קצרים ביותר.
4. פונקציה `prim` – מקבלת גרף, מחשבת את העץ הפורש המינימלי ומחזירה אותו (כלומר מחזירה גרף המייצג את העץ).
5. פונקציה `kruskal` – כמו סעיף הקודם.

- אינה זמינה במטלה זו, לצורך מימוש אלגוריתמים אלו תצטרכו לממש באופן בסיסי מבני נתונים STL-מכיוון ש מימוש בסיסי אומר כי מספיק לממש מבנים אלו רק פונקציאונלית, union find-נוספים: תור ו\או תור עדיפויות ו אין דרישה מיוחדת לסיבוכיות.

דרישות נוספות:

- ציבורי repository-חשוב לוודא שה.
- כתבו בתחילת **כל** קובץ את כתובת המייל שלכם.
- כתבו קוד נקי, מסודר, מחולק לקבצים, מודולרי, מתועד בצורה מספקת וכמובן בדיקות יחידה עבור כל הפונקציות.
- בדקו את תקינות הקלט ולזרוק חריגות מתאימות במידת הצורך.
- בו תוכלו לראות דוגמאות נוספות לשימוש בסיפריה זו doctest לשימושכם הקישור הבא.
- יש לבדוק שאין זליגת זיכרון באמצעות `valgrind`.
- עם הסבר על פרויקט, על חלוקה למחלקות וקבצים וכל מידע אחר רלוונטי `README` יש לצרף גם קובץ.

קובץ `Makefile`:

הוסיפו לפרויקט קובץ `Makefile` הכולל את הפקודות הבאות:

- הפקודה `make Main` – להרצת קובץ ההדגמה.
- הפקודה `make test` – להרצת בדיקות היחידה.
- הפקודה `make valgrind` – בדיקת זליגת זיכרון באמצעות valgrind.

- הפקודה `make clean` - מוחקת את כל הקבצים הלא רלוונטיים לאחר ההרצה.

בהצלחה!