**מטלת גמר - מערכות הפעלה**

שלבים 1-4 (Q\_1\_to\_4)

### **מבנה הקבצים:**

* תיקיית Graph ובתוכה:

Graph.hpp - קובץ הקוד הראשי.

doctest.h - קובץ לשימוש בטסטים.

testingG.cpp - קובץ טסטים שבודק את איכות הפונקציות ומקרי קצה.

* main\_graph.cpp - קובץ ההרצה הנדרש בשלב 3.
* Makefile – קובץ הוראות הידור והרצה (כולל בנייה, ניקוי, הרצה תחת valgrind/coverage וכו').

**דוגמת הרצה לשלב 3:**



**דגש** - הוספנו לקוד הראשי של הגרפים את הפונקציונליות הנדרשת בשלב 7

### **אפשרויות ההרצה:**

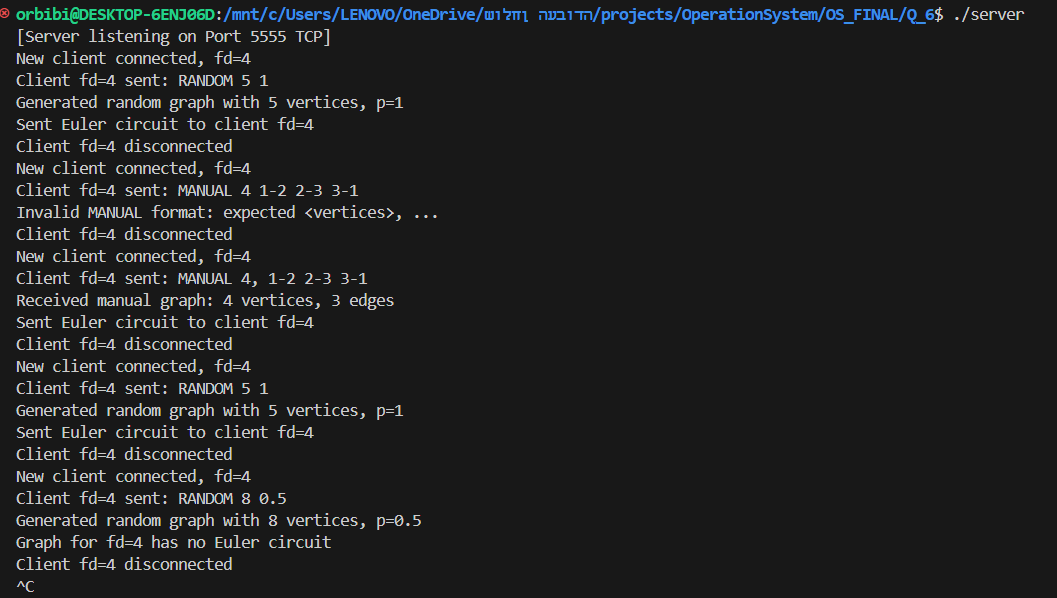
* make → בנייה בלבד של main\_graph.
* make test → בדיקות - לא הכבדות והביצועים (Valgrind + coverage).
* make test-hard → בדיקות כבדות (ביצועים, ללא coverage).
* make coverage → כיסוי קוד על main בלבד.
* make report → דו"ח HTML מסודר של coverage.
* make helgrind → בדיקות race conditions.
* make callgrind → פרופיילינג ביצועים.
* make clean → ניקוי מלא של סביבת העבודה.

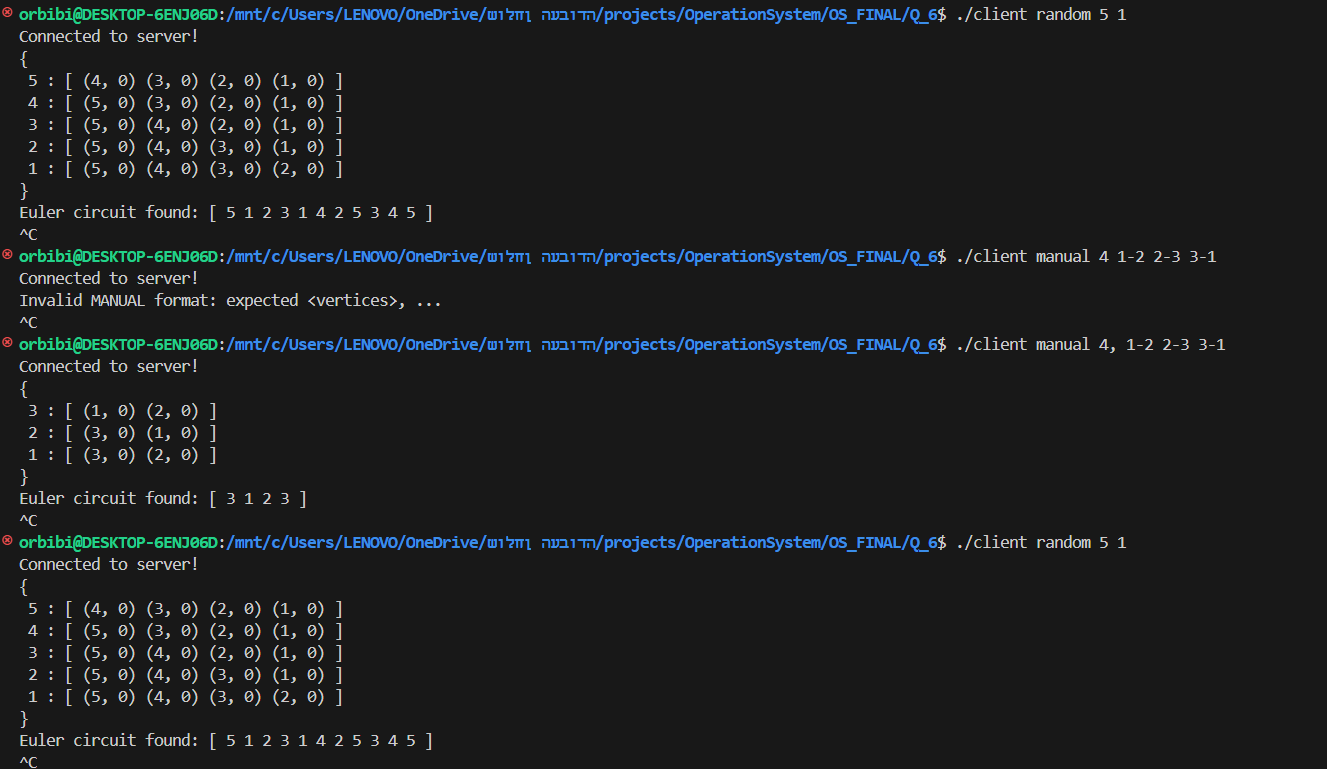
שלב 5 - בוטל

שלב 6  
**מבנה הקבצים:**

* server.cpp - קובץ השרת שמבצע את החישובים ושולח את התוצאה.
* client.cpp - קובץ הלקוח ששולח את הגרף.
* network\_interface.hpp - קובץ ששומר את הinclude הרלוונטיים לתקשורת ואת נתוני התקשורת.
* Makefile – קובץ הוראות הידור והרצה (כולל בנייה, ניקוי, הרצה תחת valgrind/coverage וכו').

**דוגמאות הרצה לשלב 6:**





### **אפשרויות ההרצה:**

* make → מקמפל server+client.
* make valgrind\_server/client - ריצה תחת Valgrind memcheck.
* make helgrind\_server/client - ריצה תחת Helgrind.
* make callgrind\_server/client - ריצה תחת Callgrind (פרופיילינג).
* make cover\_build - יוצר קבצי הרצה מתאימים.
* make cover\_client - הרצת client עם coverage.
* make cover\_server - כנ״ל ל־server.
* make gcov\_report - הפקת דוח gcov.
* make clean → מוחק פלטים והורג תהליכים על הפורט.
* make clean\_coverage - מוחק רק קבצי כיסוי.

**בכל מקום שמריצים את הלקוח לא בצורה ישירה צריך להוסיף ARGS=”” ובסוגרים לשים את הקלט.**

שלב 7  
**מבנה הקבצים:**

* server.cpp – קובץ צד שרת שמבצע את החישובים ומנהל את ההיגיון בצד השרת.
* client.cpp – קובץ צד לקוח ששולח את הגרף/נתונים לשרת.
* server\_functions.hpp – קובץ עזר שמרכז include ומכיל פונקציית שליחת תפריט ורשימת שמות אלגוריתמים.
* network\_interface.hpp – קובץ שמרכז את כל ה־include הרלוונטיים ואת מבני הנתונים המשמשים בתקשורת בין לקוח לשרת.
* Makefile – קובץ הוראות הידור והרצה (כולל בנייה, ניקוי, הרצה תחת valgrind/coverage וכו').

#### ספרייה Socket\_class

* Client\_Socket.cpp / Client\_Socket.hpp – מימוש וממשק למחלקת הסוקט בצד הלקוח, אחראית על יצירת חיבור ושליחת נתונים.
* Server\_Socket.cpp / Server\_Socket.hpp – מימוש וממשק למחלקת הסוקט בצד השרת, אחראית על האזנה, קבלת חיבורים וקליטת נתונים.

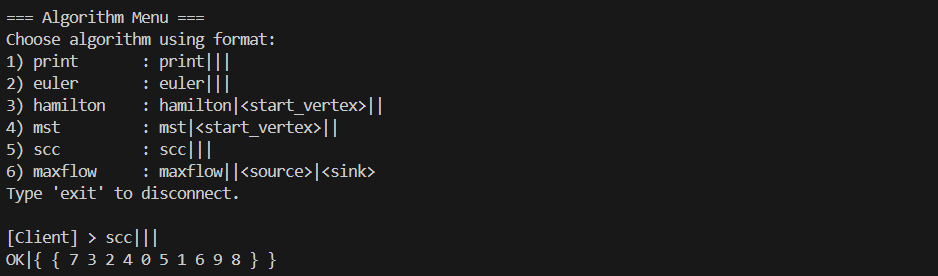
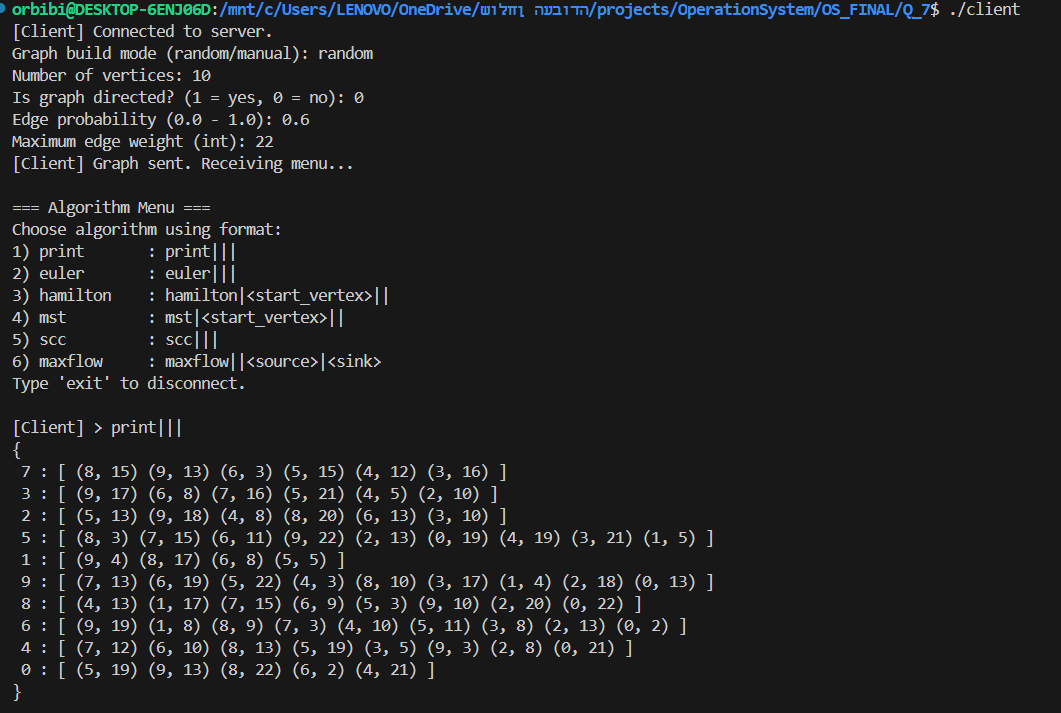
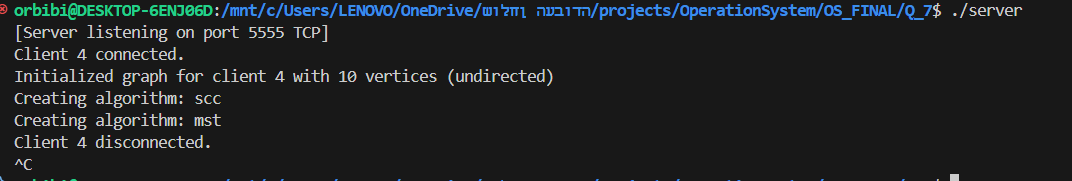
#### ספרייה Strategy

* Strategy\_interface.hpp – ממשק אסטרטגיה מופשט (interface) שמגדיר את שיטת פתרון הבעיה.
* AlgoIO.hpp – קובץ עזר לטיפול בקלט/פלט של גרפים או נתוני אלגוריתם.
* EulerAlgo.hpp – מימוש אלגוריתם לחישוב מסלול אוילרי.
* HamiltonAlgo.hpp – מימוש אלגוריתם לחישוב מסלול המילטוני.
* Max\_Flow.hpp – מימוש אלגוריתם זרימה מקסימלית בגרף.
* MST\_Algo.hpp – מימוש אלגוריתם ליצירת עץ פורש מינימלי.
* SCC\_Algo.hpp – מימוש אלגוריתם לרכיבי קישור חזק בגרף.

#### ספרייה Factory

* Factory\_Algorithms.hpp – מחלקת Factory ליצירת אובייקטים של אלגוריתמים שונים בהתאם לקלט.

**דוגמאות הרצה לשלב 7:**

****

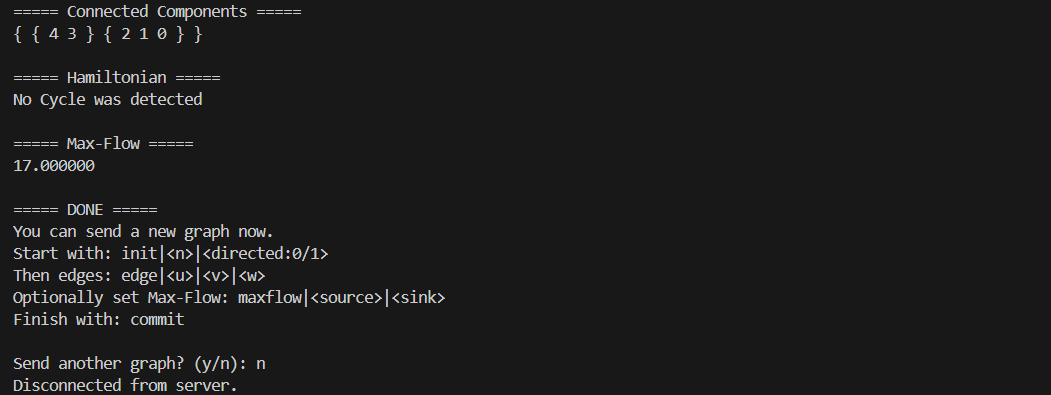
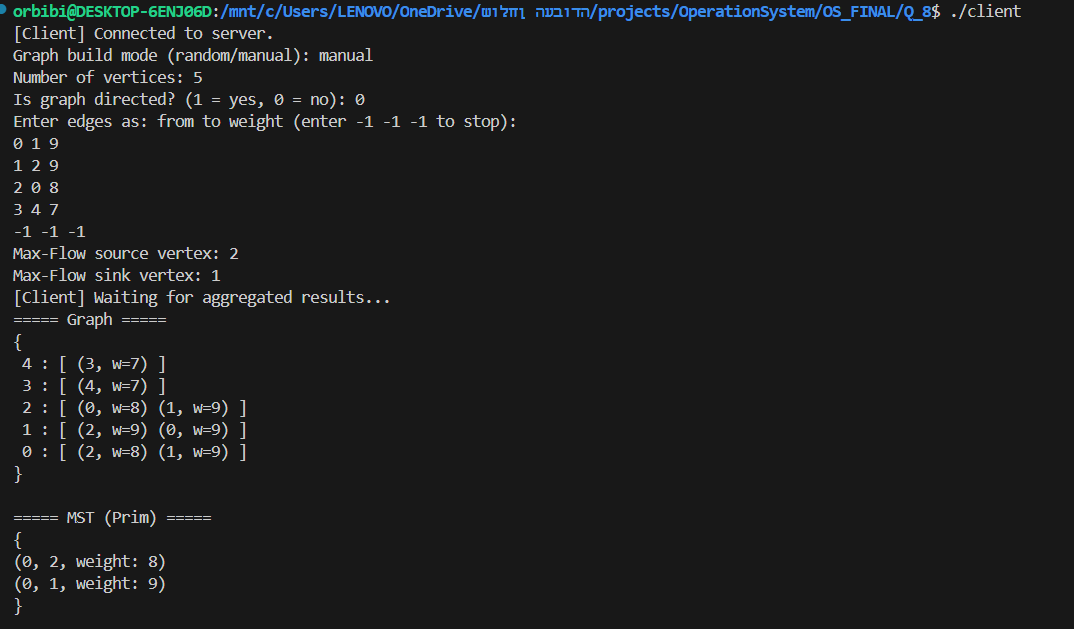
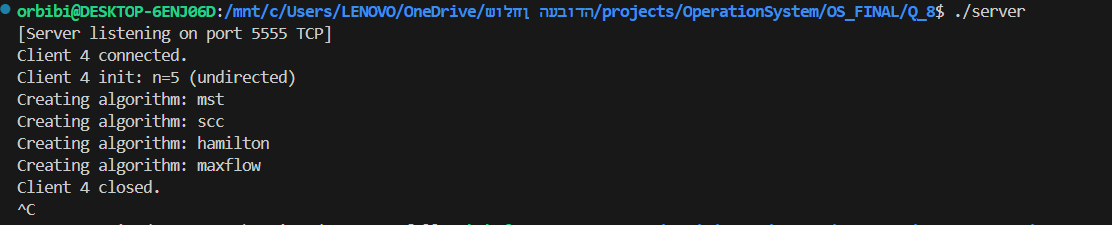
### **אפשרויות ההרצה:**

* make → מקמפל server+client.
* make valgrind\_server/client - ריצה תחת Valgrind memcheck.
* make helgrind\_server/client - ריצה תחת Helgrind.
* make callgrind\_server/client - ריצה תחת Callgrind (פרופיילינג).
* make cover\_build - יוצר קבצי הרצה מתאימים.
* make cover\_client - הרצת client עם coverage.
* make cover\_server - כנ״ל ל־server.
* make gcov\_report - הפקת דוח gcov.
* make clean → מוחק פלטים והורג תהליכים על הפורט.
* make clean\_coverage - מוחק רק קבצי כיסוי.

שלב 8  
**מבנה הקבצים:**

* server.cpp – קובץ צד שרת שמנהל את ההיגיון, מקבל בקשות מהלקוחות ומפעיל את האלגוריתמים דרך ה־Factory.
* client.cpp – קובץ צד לקוח ששולח גרפים או בקשות אל השרת ומקבל את התשובות.
* run\_all.hpp – קובץ שמרכז פונקציות להרצת כל האלגוריתמים על גרף נתון.
* Makefile – קובץ הוראות הידור והרצה (כולל בנייה, ניקוי, הרצה תחת valgrind/coverage וכו').

**דוגמאות הרצה לשלב 8:**

****

### **אפשרויות ההרצה:**

* make → מקמפל server+client.
* make valgrind\_server/client - ריצה תחת Valgrind memcheck.
* make helgrind\_server/client - ריצה תחת Helgrind.
* make callgrind\_server/client - ריצה תחת Callgrind (פרופיילינג).
* make cover\_build - יוצר קבצי הרצה מתאימים.
* make cover\_client - הרצת client עם coverage.
* make cover\_server - כנ״ל ל־server.
* make gcov\_report - הפקת דוח gcov.
* make clean → מוחק פלטים והורג תהליכים על הפורט.
* make clean\_coverage - מוחק רק קבצי כיסוי.

שלב 9

**\*בשלב זה הרחבנו בהסבר כי זה השלב האחרון שמאגד את רוב הפרוייקט**  
**מבנה הקבצים – שלב 9**

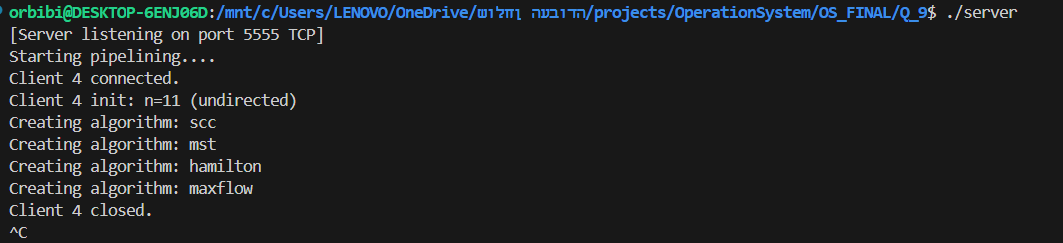
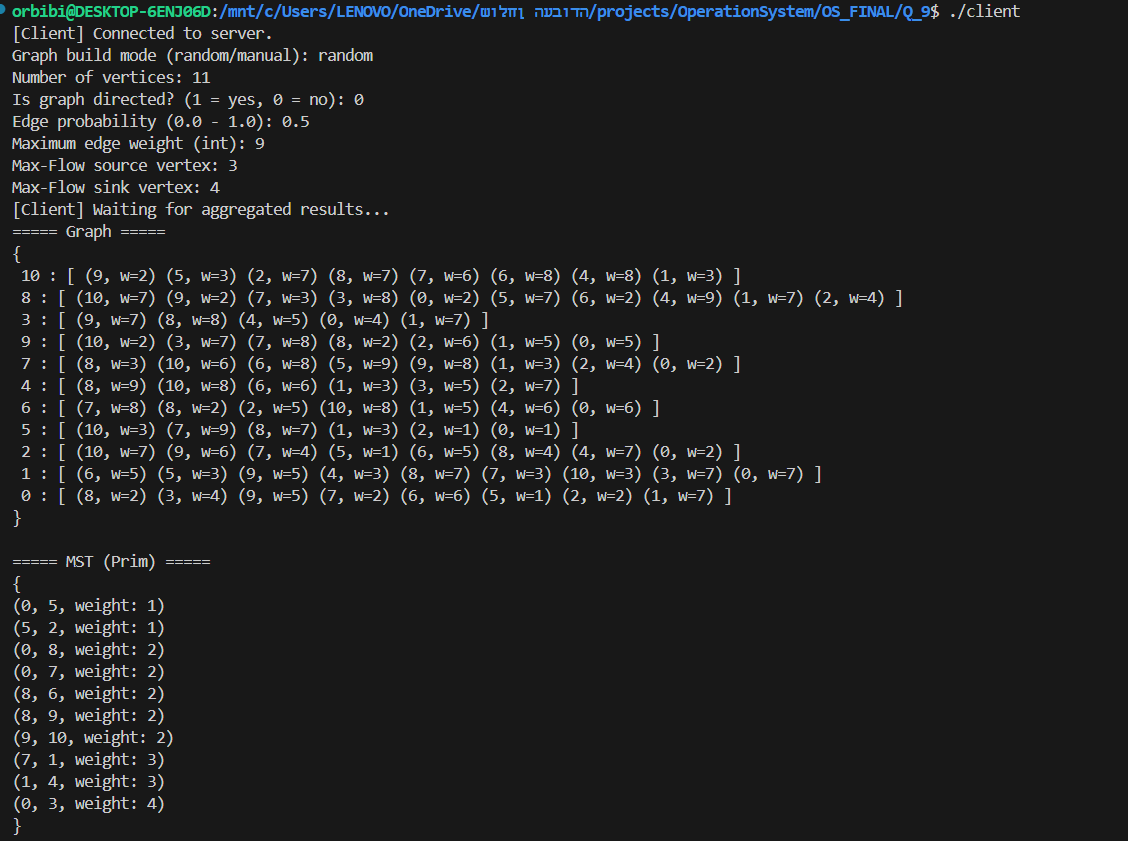
בשלב 9 בפרויקט, התיקייה מחולקת למספר חלקים מרכזיים:

* **server.cpp** – קובץ השרת הראשי.  
  + מאזין לחיבורים נכנסים.
  + קורא פקודות מהלקוח (init, edge, maxflow, commit).
  + בונה גרף לכל לקוח.
  + יוצר **Job** ושולח אותו ל־Pipeline.
* **client.cpp** – קובץ הלקוח.  
  + מתחבר לשרת.
  + שולח פקודות ליצירת גרף.
  + שולח בקשת commit.
  + מציג את תוצאות האלגוריתמים שהשרת מחזיר.
  + מאפשר שליחה של גרף נוסף.
* **pipeline/** – ספריית ניהול העבודות:  
  + **Job.hpp** – מבנה Job עם מזהה, fd של הלקוח, גרף ומידע נוסף.
  + **AlgoBinders.hpp** – קישור האלגוריתמים (MST, SCC, Hamilton, MaxFlow) לפייפליין.
  + **ActiveObject.hpp** – מימוש כללי של Active Object (תור + thread פנימי).
  + **AO\_Aggregator.hpp** – צירוף תוצאות מארבעה אלגוריתמים למבנה פלט אחד.
  + **AO\_Responder.hpp** – שליחת הפלט המלא ללקוח.
  + **Pipeline.hpp / Pipeline.cpp** – מתזמר שמחבר בין ה־Jobs, האלגוריתמים, ה־Aggregator וה־Responder.
* **Graph/** (מ־Q\_1\_to\_4) – מימוש מחלקת הגרף, כולל אחסון צמתים וקשתות, בניה והדפסה.
* **Factory / Strategy** (מ־Q\_7) – מימוש האלגוריתמים עצמם, לפי Design Pattern:  
  + Prim / MST
  + SCC
  + Hamiltonian
  + MaxFlow
* **Socket\_class/** (מ־Q\_7) – עטיפה לתקשורת TCP:  
  + **Server\_Socket.hpp** – ניהול socket של שרת.
  + **Client\_Socket.hpp** – ניהול socket של לקוח.
  + **network\_interface.hpp** – קבועים וממשק כללי.

## **תרשים זרימת העבודה – שלב 9**

1. **לקוח** שולח פקודות לשרת (init, edge, maxflow, commit).
2. **server.cpp** קורא את הפקודות ומרכיב גרף מותאם ללקוח.
3. בעת commit השרת יוצר **Job** ושולח אותו ל־**Pipeline**.
4. **Pipeline**:  
   * רושם את ה־Job ב־Aggregator.
   * מפעיל במקביל את ארבעת האלגוריתמים דרך **AlgoBinders**.
   * כל תוצאה חלקית נשלחת ל־Aggregator.
5. **Aggregator** אוסף את כל התוצאות ומרכיב פלט מסכם אחד (כולל גרף וכל האלגוריתמים).
6. **Responder** שולח את הפלט חזרה ללקוח.
7. **לקוח** מציג את התוצאה, ושואל אם לשלוח גרף נוסף.

## **דוגמת הרצה – שלב 9 :**

****

### **אפשרויות ההרצה:**

* make → מקמפל server+client.
* make valgrind\_server/client - ריצה תחת Valgrind memcheck.
* make helgrind\_server/client - ריצה תחת Helgrind.
* make callgrind\_server/client - ריצה תחת Callgrind (פרופיילינג).
* make cover\_build - יוצר קבצי הרצה מתאימים.
* make cover\_client - הרצת client עם coverage.
* make cover\_server - כנ״ל ל־server.
* make gcov\_report - הפקת דוח gcov.
* make clean → מוחק פלטים והורג תהליכים על הפורט.
* make clean\_coverage - מוחק רק קבצי כיסוי.