

פרויקט גמר – מערכות הפעלה

שאלה 1 :

התבקשנו לממש מבנה עבור גרף .

- גרף לא מכון באמצעות רשימות שכנות (Adjacency List).
- פעולות בסיסיות : יצירה, פירוק , הוספת צלע דו כיוונית (כי הגרף אינו מכון), והדפסת הגרף.
- אין כפילויות בין שני קודקודים שונים, ו-מותרת לכל היותר לולאה אחת לכל קודקוד.

מבני נתונים

- EdgeNode צומת ברשימת שכנות (קשת "מכוונת" אחת ברשימה):
to אינדקס השכן.

next מצביע לצומת הבא ברשימה.

- Vertex ראש רשימת השכנים של קודקוד:

Head מצביע לצומת הראשון ברשימה.

- Graph

n מספר הקודקודים. (0..n-1)

adj מערך בגודל חשל ראשים לרשימות שכנות. (Vertex*)

פונקציות עיקריות

1. Graph* graph_create(int n)
יוצר גרף ריק עם n קודקודים

2. void graph_destroy(Graph* g)
משחרר את כל צומתי הרשימות בכל הקודקודים, ואז את המערך והגרף עצמו.

3. int graph_add_edge(Graph* g, int u, int v)
מוסיף צלע לא-מכוונת בין u ל- v שתי הופעות.

אם $u \neq v$ מונעים כפילות

קודי חזרה:

0 הצלחה

1-חורג טווח

2-כשל הקצאה (אין שינוי במבנה)

3-צלע קיימת כבר / לולאה שנייה

4. void graph_print(const Graph* g)
מדפיס את רשימות השכנות בפורמט: v1 v2 v3 .. i :

- הוספנו Makefile על מנת שנוכל להריץ את ה demo
- הוספנו demo על מנת להמחיש את מבנה הגרף .

שאלה 2 :

יש מעגל אוילר אם ורק אם:

1. הגרף מחובר כשמתעלמים מקודקודים מבודדים (דרגה 0).
 2. כל הדרגות זוגיות לכל קודקוד מספר צלעות זוגי.
- הערה פרקטית למימוש שלנו: דרשנו גם שלגרף תהיה לפחות צלע אחת. מתמטית לפעמים מחשיבים גם גרף ריק/בודד כטריוויאלי, אבל בקוד זה מוחזר "אין מעגל" כשאין צלעות.
- הפונקציה `graph_has_euler_circuit` בודקת:

1. קישוריות של תת-הגרף
 2. כל הדרגות זוגיות.
 3. יש לפחות צלע אחת.
- אם משהו מזה לא מתקיים לא קיים מעגל אוילר ולכן `graph_find_euler_circuit` מחזירה 0.
- ייצוג העזר `EdgeView` :

- `ev.edges[m]` מערך הצלעות הלא-מכוונות בגודל `m`
- `ev.incid[v]` לכל קודקוד `v` רשימה של אינדקסי צלעות שנוגעות בו.

הפונקציה `build_edge_view` :

- אם זו לולאה, $(u=v)$ ברשימות יש שתי הופעות. $u \rightarrow u$ לכן סופרים "חצי-לולאות", וכל שתיים יוצרות צלע לולאה אחת ב- `edges`, ומכניסים את האינדקס פעמיים ל- `incid[u]`.
- אם זו צלע רגילה, $u \neq v$ מוסיפים את הצלע פעם אחת כש- $v < u$ כדי לא להכפיל, ומכניסים את האינדקס גם ל- `incid[u]` וגם ל- `incid[v]`.

`graph_find_euler_circuit`

`used[m]` אם צלע כבר שומשה.

`incid[n]` לילכל קודקוד מורה-מקום איפה הגענו בסריקה של `incid[v]`.

`stack`

`path` יציאת המסלול הסופי (יצטרב בסדר הפוך, אז נהפוך בסוף).

1. בחר נקודת התחלה
 2. בחרים קודקוד שיש לו לפחות צלע אחת ומניחים אותו על מחסנית.
 3. כל עוד יש לאן ללכת – הולכים אחרת – חוזרים אחורה
- מסתכלים על הקודקוד שבראש המחסנית. (u)
 - אם יש מ- u צלע שלא השתמשנו בה עדיין: נסמן את הצלע כמשומשת, עוברים לקצה השני שלו, (v) ומניחים את v על המחסנית.
 - אם אין יותר צלעות פנויות מ- u : אין לאן להתקדם, אז "חוזרים אחורה" מוציאים את u מהמחסנית ומוסיפים אותו לסוף רשימת התוצאה. (`path`)
 - 3. בסיום – הופכים את הרשימה כשהמחסנית מתרוקנת, קיבלנו את המסלול ב- `path` אבל מהסוף להתחלה. הופכים את הסדר – וזה המעגל.

- הכנו makefile ו demo להדמייה .

```
ron@Ron:~/OS_project/part2$ ./demo
===== Graph 1 (square 0-1-2-3-0) =====
0: 3 1
1: 2 0
2: 3 1
3: 0 2
Euler C I R C U I T found. Length (vertices): 5
Cycle: 0 -> 3 -> 2 -> 1 -> 0

===== Graph 2 (path 0-1-2-3) =====
0: 1
1: 2 0
2: 3 1
3: 2
No Euler C I R C U I T

===== Graph 3 (triangle 0-1-2-0) =====
0: 2 1
1: 2 0
2: 0 1
Euler C I R C U I T found. Length (vertices): 4
Cycle: 0 -> 2 -> 1 -> 0
```

שאלה 3:

התבקשנו ליצור גרף אקראי ולהריץ עליו את האלגוריתם.
באמצעות `getopt(3)` אנו קולטים מהמשתמש פרמטרים בדגלים:

- `<vertices> -v` מספר הקודקודים (חובה)
 - `<edges> -e` מספר הקשתות המבוקש (חובה)
 - `<seed> -r` זרע אקראי לשחזור אותו גרף בין הרצות (חובה)
- אם חסר אחד מהפרמטרים (`-v`, `-e`, `-r`) נדפיס `usage` ונצא.

בפונקציה `generate_random_graph`:
אנו בונים גרף ריק באמצעות `graph_create(n)` ואז מוסיפים קשתות

לאחר יצירה:

- מדפיסים את רשימות השכנות. (`graph_print`)
- בודקים קיום **מעגל אוילר** באמצעות `graph_has_euler_circuit`
- אם נמצא — מפעילים `graph_find_euler_circuit`, מקבלים את המסלול (`cycle, cycle_length`) ומדפיסים אותו בפורמט `u -> v -> ... -> u`

```

● ron@Ron:~/OS_project/part3$ ./random -v 4 -e 4 -r 123
=== Random Graph Generation ===
Vertices: 4
Edges to generate: 4
Random seed: 123
Maximum possible edges: 10

Generating random edges...
Added edge: 1 -- 1 (total: 1/4)
Added edge: 1 -- 2 (total: 2/4)
Added edge: 3 -- 3 (total: 3/4)
Added edge: 3 -- 2 (total: 4/4)

=== Generated Graph ===
0:
1: 2 1
2: 3 1
3: 2 3

=== Euler Circuit Analysis ===
Euler circuit exists! Finding it...

=== Euler Circuit Found ===
Circuit length (vertices): 3
The circuit is:
1 -> 2 -> 3

Done!

```

```

● ron@Ron:~/OS_project/part3$ ./random -v 5 -e 7 -r 42
=== Random Graph Generation ===
Vertices: 5
Edges to generate: 7
Random seed: 42
Maximum possible edges: 15

Generating random edges...
Added edge: 1 -- 0 (total: 1/7)
Added edge: 1 -- 1 (total: 2/7)
Added edge: 2 -- 3 (total: 3/7)
Added edge: 0 -- 3 (total: 4/7)
Added edge: 4 -- 3 (total: 5/7)
Added edge: 2 -- 4 (total: 6/7)
Added edge: 1 -- 2 (total: 7/7)

=== Generated Graph ===
0: 3 1
1: 2 1 0
2: 1 4 3
3: 4 0 2
4: 2 3

=== Euler Circuit Analysis ===
No Euler circuit exists in this graph.
(Either the graph is not connected, or some vertices have odd degree)

```

שאלה 6 :

התבקשנו לממש שרת ולקוח, כך שהלקוח שולח לשרת גרף מסוים והשרת מחזיר האם יש מסלול אוילר בגרף שנשלח או לא, אם יש מסלול הלקוח מקבל אותו.

הסבר על המימוש :

שרת :

זה שרת TCP שמאזין לפורט שניתן בשורת הפקודה, מקבל מהלקוח גרף כמטריצת שכנויות, מפעיל עליו את פונקציות הגרף מ- graph.c ומחזיר ללקוח תשובה האם נמצא מעגל אוילר ואם כן הוא מחזיר אותו.

לקוח :

הלקוח פשוט שולח לשרת את מספר הקודקודים ואת הצלעות הנבחרות

דוגמת הרצה :

```
ron@Ron:~/OS_project/part6$ ./euler_server 5052
=== Euler Circuit Server ===
Starting server on port 5052...
Server listening on port 5052 (max 10 clients)
Protocol: [n][nxn matrix] → [status][length][cycle...]
Ready to accept connections...

Client connected from 127.0.0.1:37984
```

```
Enter number of vertices (0 to exit): 4
Now enter edges. Enter '0 0' to finish:
Enter src dest: 0 1
Enter src dest: 0 2
Enter src dest: 2 3
Enter src dest: 3 1
Enter src dest: 0 0
Sending graph to server...
Waiting for server response...
Received 28 bytes from server
Status: 1, Length: 5
✓ Euler circuit found! Length: 5
Circuit: 0->2->3->1->0
```

```

ron@Ron:~/OS_project/part6$ ./euler_client 5052
Enter number of vertices (0 to exit): 4
Now enter edges. Enter '0 0' to finish:
Enter src dest: 0 1
Enter src dest: 0 2
Enter src dest: 0 3
Enter src dest: 1 2
Enter src dest: 1 3
Enter src dest: 2 3
Enter src dest: 0 0
Sending graph to server...
Waiting for server response...
Received 8 bytes from server
Status: 0, Length: 0
X No Euler circuit exists

```

שאלה 7

הוספנו 4 אלגוריתמים :

1. מציאת קליקה מקסימלית .

קליקה מקסימלית : למצוא קבוצת קודקודים שכל זוג בה מחובר בקשת, בגודל המקסימלי.

הקוד עובד עם גרף Graph שמיוצג כרשימות שכנות. כדי לבדוק קליקות ביעילות, הוא ממיר את הגרף זמנית ל-**מטריצת שכנויות** של 0/1.

build_adjacency_matrix : מאפס מטריצה לגודל $n \times n$. עובר על רשימות השכנות ומסמן $adj[u][v] = 1$ לכל קשת $u \rightarrow v$.

בודק אם קודקוד v מחובר לכל הקודקודים שכבר בקליקה הנוכחית.

האלגוריתם:

1. אם הקליקה הנוכחית גדולה מהטובה ביותר שמצאנו – מעדכנים "Best".
2. מנסים להוסיף כל קודקוד v מהטווח $start_vertex..n-1$.
3. מוסיפים רק אם v מחובר לכולם בקליקה הנוכחית.
4. קוראים רקורסיבית משם והלאה $(v+1)$ כדי לשמור על סדר ולמנוע כפילויות.

• מחזיקים שלוש קבוצות:

R הקליקה הנוכחית.

P מועמדים שאפשר עדיין להוסיף. (Potential)

X קודקודים שכבר "ניסו" איתם. (Excluded)

אם P ו- X ריקות מצאנו קליקה מקסימלית (אי אפשר להרחיב, ואי אפשר להגיע אליה בדרך אחרת).

2. מציאת מספר קליקות

ספירת כל הקליקות בכל הגדלים.

ספירת קליקות בגודל מסוים.

ספירת משולשים (3-קליקות) ביעילות.

ספירת קשתות (2-קליקות).

בדיקה אם יש קליקה בגודל מסוים.

הדפסת ניתוח מפורט של התוצאות.

הקוד מממש מנגנון לספירת קליקות בגרף על ידי בניית מטריצת שכנויות וביצוע חיפוש רקורסיבי שמוסיף קודקודים רק אם הם יוצרים קליקה עם הקודקודים שכבר נבחנו. בצורה זו, נבדקות כל תתי-הקבוצות האפשריות של קודקודים אך נמנעת ספירה כפולה, כי החיפוש מתקדם תמיד קדימה ברשימת הקודקודים. המימוש כולל ספירה של כל הקליקות בכל הגדלים, ספירה של קליקות בגודל מסוים, וגם אלגוריתמים יעודיים ומהירים יותר לספירת משולשים וקשתות.

3. זרימה מקסימלית

הקוד הזה מממש חישוב **זרימה מקסימלית** בגרף (רשת זרימה) באמצעות אלגוריתם **Edmonds–Karp**, שהוא וריאציה של **Ford–Fulkerson** המשתמשת ב-**BFS** למציאת מסלולים משפרים.

השלבים המרכזיים במימוש:

1. בניית מטריצת קיבולות (Capacity Matrix)

הפונקציה `build_capacity_matrix` מממשת את רשימות השכנות של הגרף למטריצה `capacity` שבה הערך `capacity[u][v]` הוא הקיבולת של הקשת מ-`u` ל-`v` הקיבולות נקבעות לפי משקל הקשת-`edge_weight`, אם אין קשת.

2. חיפוש מסלול משפר (BFS)

הפונקציה `bfs_find_path` מחפשת מסלול בו כל הקשתות עם קיבולת חיובית. היא מחזירה גם את מערך `parent` שמתאר את המסלול שנמצא.

3. קביעת הזרימה במסלול

הפונקציה `find_path_flow` מוצאת את הקיבולת המינימלית על המסלול שנמצא זהו ערך הזרימה שאפשר להוסיף.

הקוד מממש את אלגוריתם **Edmonds–Karp** לחישוב הזרימה המקסימלית בין שני קודקודים בגרף, על בסיס גרסת **Ford–Fulkerson** שמשתמשת ב-**BFS** למציאת מסלולים משפרים. תחילה הגרף מומר למטריצת קיבולות לפי משקלי הקשתות, ולאחר מכן מתבצע חיפוש חוזר אחר מסלולים מהרצף למקור (`source`) ליעד (`sink`) שבהם נותרת קיבולת חיובית. בכל מסלול כזה מחושבת הקיבולת המינימלית, הזרימה מתווספת לערך הכולל, והגרף הרזידואלי מתעדכן בהתאם. התהליך חוזר עד שאין עוד מסלול משפר, ואז הערך שנצבר מייצג את הזרימה המקסימלית האפשרית.

4. מציאת משקל mst

הקוד הזה מממש חישוב עץ פורש מינימלי (MST) בגרף משוקלל באמצעות האלגוריתם של פריים, תוך שימוש בתור עדיפויות בצורת ערימת מינימום.

מהלך הפעולה:

1. בניית מטריצת משקלים
הפונקציה `build_weight_matrix` ממירה את רשימות השכנות של הגרף למטריצה `matrix` שבה כל תא `[u][v]` מכיל את משקל הקשת מ-`u` ל-`v` (או 0 אם אין קשת).
2. אתחול משתנים

`in_mst` : מערך סימון עבור קודקודים שכבר בעץ ה-MST.

`key` : המינימום של משקל הקשת שמחברת את כל קודקוד לעץ.

`parent` : הקודקוד ב-MST שממנו מגיעה הקשת המינימלית אל קודקוד זה.

3. לולאת Prim

מתחילים מקודקוד 0 (`key[0] = 0`) ודוחפים אותו לתור העדיפויות עם משקל 0.

בכל שלב מוציאים את הקודקוד בעל משקל הקשת הקטן ביותר אל ה-MST.

מעדכנים את המפתחות (`key`) של כל קודקוד סמוך אם נמצאה קשת זולה יותר שמחברת אותו ל-MST.

דוחפים לתור עדיפויות את הקודקודים עם המפתחות החדשים.

4. בדיקת קישוריות

אם לא כל הקודקודים נוספו ל-MST, הגרף אינו קשיר ואין עץ פורש.

5. בניית התוצאה

אם הגרף קשיר, יוצרים מערך של `MST_Edge` עם כל הקשתות שב-MST ומחשבים את המשקל הכולל (`total_weight`).

6. פונקציות נוספות

`graph_print_mst` – מדפיסה את משקל העץ ואת רשימת הקשתות.

`mst_result_free` – משחררת את הזיכרון של מערך הקשתות.

`graph_mst_weight` – מחזירה רק את המשקל הכולל של ה-MST.

הקוד מממש את האלגוריתם של פריים למציאת עץ פורש מינימלי (MST) בגרף משוקלל, תוך שימוש בערימת מינימום (Priority Queue) לניהול הקודקוד הבא להוספה. תחילה הגרף מומר למטריצת משקלים בהתאם למשקלי הקשתות, ואז מתחילים מקודקוד 0 ומרחיבים את העץ בהדרגה על ידי בחירת הקשת הזולה ביותר שמחברת קודקוד חדש ל-MST. מערכי עזר עוקבים אחרי הקודקודים שכבר בעץ, המפתח המינימלי של כל קודקוד, והקודקוד ההורה שממנו מגיעה הקשת. בסיום, אם הגרף קשיר, נבנה מערך הקשתות של ה-MST ונחשב את משקלו הכולל.

Factory Pattern תבנית יצירה שבה יש "מפעל" שמקבל פרמטר ומחזיר אובייקט מתאים מבלי שהקוד הקורא יצטרך לדעת איך האובייקט נבנה בפועל. הרעיון – להפריד בין הלוגיקה של בחירת האובייקט/יצירתו לבין השימוש בו, וכך להקל על שינוי והרחבה של סוגי האובייקטים.

Strategy Pattern תבנית התנהגות שמאפשרת להחליף את האלגוריתם או ההתנהגות של אובייקט בזמן ריצה ע"י העברת אובייקט "אסטרטגיה" שמממש ממשק ידוע. כך הקוד הקורא עובד עם אותו ממשק אבל בפועל מבוצעת לוגיקה שונה לפי האסטרטגיה שנבחרה.

[Factory.c](#)

הקוד משלב בין Factory ל- Strategy כדי להריץ אלגוריתמים שונים על גרף. ה- Factory מקבל מזהה אלגוריתם, ממפה אותו לסוג מתאים, בודק אם הוא נתמך, ויוצר אובייקט Strategy מתאים שמכיל את שם האלגוריתם, תיאורו, ואת פונקציית ההרצה שלו. ה- Strategy מועבר לאובייקט AlgorithmContext שמריץ את האלגוריתם בפועל על הגרף. מבנה זה מאפשר להוסיף, לשנות או להחליף אלגוריתמים בלי לשנות את קוד ההרצה, ולנהל את כל תהליך הבחירה, ההכנה וההרצה בצורה אחידה וגמישה.

[algorithm_strategy.c](#)

הקוד הזה מממש את **תבנית ה Strategy** עבור אלגוריתמים על גרפים, ומספק דרך אחידה לבחור ולהריץ אלגוריתם בזמן ריצה בלי שהקוד הקורא יצטרך לדעת מהי המימוש הפנימי שלו.

איך זה עובד:

1. הגדרת מבנה Strategy

בקובץ algorithm_strategy.h מוגדר AlgorithmStrategy הוא מכיל:

מצביע לפונקציית ההרצה, שמקבלת מצביע לגרף ומחזירה מחרוזת תוצאה.

שם ותיאור של האלגוריתם.

מזהה לזיהוי חד-משמעי.

2.

המבנה AlgorithmContext מחזיק את ה- Strategy הנבחר ואת הגרף שעליו ירוץ האלגוריתם.

algorithm_context_init מאתחל את הקונטקסט.

algorithm_context_set_strategy בוחר אסטרטגיה.

algorithm_context_execute ממריץ את האלגוריתם הנוכחי על הגרף.

3. מימושי אסטרטגיות בפועל

בקובץ algorithm_strategy.c מיושמים ממשקים ממשיים עבור כל אלגוריתם:

- euler_strategy_execute – מוצא מעגל אוילר.
 - maxflow_strategy_execute – מחשב זרימה מקסימלית.
 - mst_strategy_execute – מחשב עץ פורש מינימלי.
 - maxclique_strategy_execute – מוצא גודל קליקה מקסימלית.
 - ocliquecount_strategy_execute – סופר את כל הקליקות.
- כל פונקציה יוצרת מחרוזת עם תוצאה פורמטיבית, אותה מחזירים למשתמש.

לסיכום

הקוד מממש את תבנית **Strategy** להפעלת אלגוריתמים שונים על גרפים, כאשר כל אלגוריתם מוגדר כאובייקט אסטרטגיה הכולל פונקציית הרצה, שם, תיאור ומזהה. האסטרטגיות נשמרות במערך מרכזי, ומודול ה- Context מנהל את בחירת האסטרטגיה והפעלתה בפועל על הגרף. כך ניתן לבחור ולהחליף אלגוריתם

בזמן ריצה לפי מזהה או שם, להריץ אותו דרך ממשק אחיד, ולהרחיב את המערכת בקלות באמצעות הוספת אסטרטגיות חדשות מבלי לשנות את קוד ההרצה הקיים.

Server.c client.c

צד הלקוח

הלקוח נועד להתחבר לשרת אלגוריתמים ב-TCP, לבחור אלגוריתם, להזין נתוני גרף בהתאם לפרוטוקול (לא ממושקל או ממושקל), לשלוח את הבקשה, ולקבל את התוצאה. הוא מטפל בשני סוגי פרוטוקולים:

- **ממושקל** – (Max Flow, MST) שולח מספר קדקודים, מספר קשתות ורשימת קשתות עם משקלים.
 - **לא ממושקל** – (Euler, Max Clique, Clique Count) שולח מטריצת סמיכויות. בסוף מקבל את תשובת השרת עם סטטוס ואורך מחרוזת התוצאה ומדפיס אותה למשתמש.
- מאפשרת** למשתמש לבחור אלגוריתם ולהזין נתוני גרף, מטפלת בהכנת הבקשה לפי פרוטוקול מתאים, ושולחת לשרת לקבלת תוצאה.

צד השרת

השרת מאזין לחיבורים ב-TCP ומקבל בקשות להפעלת אלגוריתמים. הוא מזהה את סוג האלגוריתם לפי ה-ID ושקיבל ומעבד את הבקשה בהתאם לפרוטוקול:

- **ממושקל** – בונה גרף עם קשתות ומשקלים מעודכנים.
 - **לא ממושקל** – בונה גרף ממטריצת סמיכויות.
- לאחר בניית הגרף, השרת מפעיל את האלגוריתם המבוקש באמצעות תבניות Factory + Strategy ומחזיר ללקוח את התוצאה במבנה אחיד ([סטטוס][אורך][מחרוזת]).

יישום שרת גרפים המבוסס על TCP שמקבל בקשות להפעלת אלגוריתמים שונים, בונה את הגרף לפי סוג הנתונים (ממושקל או לא), מריץ את האלגוריתם בעזרת Factory + Strategy ושולח את התוצאה ללקוח במבנה קבוע.

איך הכל עובד ביחד

1. **הלקוח (client.c)**
המשתמש בוחר אלגוריתם ומזין את הגרף.
2. **השרת (server.c)**
השרת קורא את ה-ID והנתונים, בונה מבנה Graph (עם משקלים/מטריצה בהתאם), ואז מפעיל:
Factory ממפה algorithm_id ל-AlgorithmType, בודק תמיכה, ויוצר את אובייקט ה-**Strategy** המתאים.
3. **תוצאה חזרה ללקוח**
Strategy + Context : מציב את האסטרטגיה בקונטקסט ומריץ אותה על הגרף (Max Flow / MST / Max Clique / Clique Count / Euler).
השרת מחזיר מענה אחיד והלקוח מציג את המחרוזת למשתמש.

דוגמאות הרצה :

1. הפעלת השרת והקליינט

```
ron@Ron:~/OS_project/part7$ ./server 4747
=== Enhanced Graph Algorithm Server (Factory + Strategy) ===
Starting server on port 4747...
Algorithm Factory - Available Products:
ID  Type      Strategy Description
--  -
1   EULER      Find Euler Circuit
2   MAX_FLOW   Maximum Flow (Weighted)
3   MST        Min Spanning Tree (Weighted)
4   MAX_CLIQUE Maximum Clique
5   CLIQUE_COUNT Count All Cliques
Server listening on port 4747 (max 10 clients)
Ready to accept algorithm requests...

Client connected from 127.0.0.1:39062
█
```

```
ron@Ron:~/OS_project/part7$ ./client 4747
=== Enhanced Algorithm Server Client ===
Connected to server 127.0.0.1:4747

Available algorithms:
1. Euler Circuit (unweighted)
2. Max Flow
3. MST Weight
4. Max Clique (unweighted)
5. Clique Count (unweighted)

Enter algorithm ID (1-5, 0 to exit): █
```

2. מעגל אוילר :

```
Enter algorithm ID (1-5, 0 to exit): 1
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 1 2
Added edge 1-2
Edge: 2 3
Added edge 2-3
Edge: 3 0
Added edge 3-0
Edge: -1 -1

Sending unweighted request to server...
✓ Result: Euler circuit found (length: 5)

Enter algorithm ID (1-5, 0 to exit): 1
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 1 2
Added edge 1-2
Edge: 2 3
Added edge 2-3
Edge: -1 -1

Sending unweighted request to server...
✓ Result: No Euler circuit exists
```

3. זרימה מקסימלית

```
Enter number of vertices: 4

*** Max Flow/MST Algorithm - Weighted Graph Mode ***
Enter weighted edges (format: src dest weight/capacity, enter -1 -1 -1 to finish):
Edge: 0 1 10
Added edge 0-1 with weight/capacity 10
Edge: 0 2 8
Added edge 0-2 with weight/capacity 8
Edge: 1 3 5
Added edge 1-3 with weight/capacity 5
Edge: 2 3 7
Added edge 2-3 with weight/capacity 7
Edge: 1 2 3
Added edge 1-2 with weight/capacity 3
Edge: -1 -1 -1

Sending weighted request to server (5 edges)...
✓ Detailed Result: Max flow is: 12

Enter algorithm ID (1-5, 0 to exit): 2
Enter number of vertices: 4

*** Max Flow/MST Algorithm - Weighted Graph Mode ***
Enter weighted edges (format: src dest weight/capacity, enter -1 -1 -1 to finish):
Edge: 0 1 5
Added edge 0-1 with weight/capacity 5
Edge: 2 3 7
Added edge 2-3 with weight/capacity 7
Edge: -1 -1 -1

Sending weighted request to server (2 edges)...
✓ Detailed Result: Max flow is: 0
```

mst.4

```
Enter algorithm ID (1-5, 0 to exit): 3
Enter number of vertices: 4

*** Max Flow/MST Algorithm - Weighted Graph Mode ***
Enter weighted edges (format: src dest weight/capacity, enter -1 -1 -1 to finish):
Edge: 0 1 1
Added edge 0-1 with weight/capacity 1
Edge: 0 2 4
Added edge 0-2 with weight/capacity 4
Edge: 1 2 2
Added edge 1-2 with weight/capacity 2
Edge: 1 3 5
Added edge 1-3 with weight/capacity 5
Edge: 2 3 3
Added edge 2-3 with weight/capacity 3
Edge: -1 -1 -1

Sending weighted request to server (5 edges)...
✓ Detailed Result: MST weight: 6, Edges: 0-1(1), 1-2(2), 2-3(3)

Enter algorithm ID (1-5, 0 to exit): 3
Enter number of vertices: 4

*** Max Flow/MST Algorithm - Weighted Graph Mode ***
Enter weighted edges (format: src dest weight/capacity, enter -1 -1 -1 to finish):
Edge: 0 1 2
Added edge 0-1 with weight/capacity 2
Edge: 2 3 5
Added edge 2-3 with weight/capacity 5
Edge: -1 -1 -1

Sending weighted request to server (2 edges)...
✓ Detailed Result: MST calculation failed (graph not connected)
```

5. קליקה מקסימלית

```
Enter algorithm ID (1-5, 0 to exit): 4
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 0 2
Added edge 0-2
Edge: 1 2
Added edge 1-2
Edge: 2 3
Added edge 2-3
Edge: -1 -1

Sending unweighted request to server...
✓ Detailed Result: Max clique size is: 3

Enter algorithm ID (1-5, 0 to exit): 4
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 1 2
Added edge 1-2
Edge: 2 3
Added edge 2-3
Edge: 3 0
Added edge 3-0
Edge: -1 -1

Sending unweighted request to server...
✓ Detailed Result: Max clique size is: 2
```

6. מספר קליקות:

```
Enter algorithm ID (1-5, 0 to exit): 5
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 0 2
Added edge 0-2
Edge: 0 3
Added edge 0-3
Edge: 1 2
Added edge 1-2
Edge: 1 3
Added edge 1-3
Edge: 2 3
Added edge 2-3
Edge: -1 -1

Sending unweighted request to server...
✓ Detailed Result: Total cliques count is: 15

Enter algorithm ID (1-5, 0 to exit): 4
Enter number of vertices: 4

*** Unweighted Graph Mode ***
Enter edges (format: src dest, enter -1 -1 to finish):
Edge: 0 1
Added edge 0-1
Edge: 1 2
Added edge 1-2
Edge: 2 3
Added edge 2-3
Edge: -1 -1

Sending unweighted request to server...
✓ Detailed Result: Max clique size is: 2
```

שאלה 8:

ריבוי הליכים - LF

א. השרת צריך ליישם מספר הליכים

ב. השרת יקבל גרף או בקשה ליצירת גרף אקראי מהלקוחות ויספק תשובות לכל 4 האלגוריתמים המיושמים בסעיף 7 בצורה של LF

האפשרויות 6 ו-7 הן "קיצורי דרך" לייצור גרף אקראי ע"י השרת:

- **בחירה 6** שליחת בקשה לשרת שיהא ייצור גרף לא-ממושקל (Unweighted) באופן אקראי לפי פרמטרים (שתזין) מספר צמתים, מספר קשתות (seed), במקום שתבנה מטריצת שכנויות ידנית בצד הלקוח, הלקוח רק מבקש "Random unweighted graph".
- **בחירה 7** שליחת בקשה לשרת לייצור גרף ממושקל (Weighted) אקראי לפי פרמטרים. כאן השרת יוצר גרף עם קשתות ומשקלים אקראיים (עד משקל מקסימלי מסוים), שוב לפי מספר צמתים/קשתות/seed

```
ron@Ron:~/FinalProject/part8$ ./server 5555
=== Simple Leader-Follower Server ===
Port: 5555, Threads: 4
Server listening...
[LF] Thread 0 started
Thread 0 is Leader - accepting connections
[LF] Thread 1 started
[LF] Thread 0 is initial Leader
Press Ctrl+C to shutdown

[LF] Thread 2 started
[LF] Thread 3 started
[LF] Leader 0 accepted client 127.0.0.1:57992
[LF] Promoted thread 1 to Leader
[LF] Thread 0 processing as Worker
    Processing unweighted algorithm 4: 4 vertices
Factory: Received request for algorithm ID 4
Factory: Converted ID 4 to type 4
Factory: Creating strategy for algorithm type 4
Factory: Creating Max Clique Strategy
Factory: Successfully created strategy 'maxclique'
Factory: Strategy description: 'Maximum Clique'
Strategy: Executing algorithm 'maxclique'
Thread 1 is Leader - accepting connections
Strategy: Execution successful
Factory: Received result from strategy
```

```
ron@Ron:~/FinalProject/part8$  
ron@Ron:~/FinalProject/part8$ ./client 5555  
  
1.Euler 2.MaxFlow 3.MST 4.Clique 5.Count 6.Quick 7.Concurrent 0.Exit  
Choice: 4  
Vertices: 4  
Enter edges (u v), -1 to finish:  
0 1  
1 2  
2 3  
3 2  
1 3  
0 3  
-1 -1  
Max clique size is: 3
```

[שאלה 9 :](#)

```
bash: ./server_pipeline.dSYM/: Is a directory
ron@Ron:~/FinalProject/part9$ ./server
=== Pipeline Pattern Graph Algorithm Server ===
Using 4-stage pipeline: MST → MaxFlow → MaxClique → CliqueCount
Listening on port 3490
[Pipeline] Initialized queue: MST_Queue
[Pipeline] Initialized queue: MaxFlow_Queue
[Pipeline] Initialized queue: MaxClique_Queue
[Pipeline] Initialized queue: CliqueCount_Queue
[Pipeline] All 4 stage workers started
[Stage 4] CliqueCount worker started
[Stage 1] MST worker started
[Stage 3] MaxClique worker started
[Stage 2] MaxFlow worker started
[Main] Server ready - Pipeline pattern active!

[Main] New client connected: 127.0.0.1:36886
[Client] New client connection handler started
[Client] Header received - Seed: 123, MaxWeight: 20, Vertices: 6
[Client] Received 10 edges
[Client] Created Job 1, entering pipeline
[Pipeline] Job 1 added to MST_Queue (queue size: 1)
[Pipeline] Job 1 removed from MST_Queue (queue size: 0)
[Stage 1] Processing Job 1 - MST Algorithm
[Stage 1] Job 1 MST completed: MST: Weight=48, Edges=5
[Pipeline] Job 1 added to MaxFlow_Queue (queue size: 1)
[Pipeline] Job 1 removed from MaxFlow_Queue (queue size: 0)
[Stage 2] Processing Job 1 - MaxFlow Algorithm
[Stage 2] Job 1 MaxFlow completed: MaxFlow: Value=7 (source=0, sink=5)
[Pipeline] Job 1 added to MaxClique_Queue (queue size: 1)
[Pipeline] Job 1 removed from MaxClique_Queue (queue size: 0)
[Stage 3] Processing Job 1 - MaxClique Algorithm
[Stage 3] Job 1 MaxClique completed: MaxClique: Size=3
[Pipeline] Job 1 added to CliqueCount_Queue (queue size: 1)
[Pipeline] Job 1 removed from CliqueCount_Queue (queue size: 0)
[Stage 4] Processing Job 1 - CliqueCount Algorithm
[Stage 4] Job 1 CliqueCount completed: CliqueCount: Total=13
[Stage 4] Sending response to client for Job 1
[Stage 4] Job 1 completed and cleaned up
```



```
● ron@Ron:~/FinalProject/part9$ ./client -r -n 6 -e 10 -w 20 -s 123
client: connected to 127.0.0.1
Result from server:
=== PIPELINE PROCESSING RESULTS ===
Job ID: 1
Graph: 6 vertices
Processing Time: 0.00 seconds

=== ALGORITHM RESULTS ===
MST: Weight=48, Edges=5
MaxFlow: Value=7 (source=0, sink=5)
MaxClique: Size=3
CliqueCount: Total=13
=====
```