

פרויקט גמר – מערכות הפעלה

שאלה 1 :

התקבשנו למשתמש ממבנה עבור גרפּ .

- גרפּ לא מכoon באמצעות רשימות שכנות (Adjacency List).
- פעולות בסיסיות : ייצירה, פירוק , הוספה צלע דו כיוונית (כִי הגרפּ אינו מכoon), והדפסת הגרפּ .
- אין כפליות בין שני קודקודים שונים, ו-モותרת לכל היוטר לולאה אחות לכל קודקוד.

מבנה נתונים

- EdgeNode צומת ברשימה שכנות (קשת "מכונת" אחות ברשימה):
to אינדקס השcn.

next מצביע לצומת הבא ברשימה.

Vertex ראש רשימת השכנים של קודקוד:

Head מצביע לצומת הראשון ברשימה.

Graph •

ח מספר הקודקודים.(0..n-1)

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

פונקציות עיקריות

1. (n) graph_create(int n
יוצר גרפּ ריק עם n קודקודים

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

3. (v u i) int graph_add_edge(Graph* g, int v, int u, int i
מוסיף צלע לא-מכונת בין v ל- u שני הופעות.

אם v != u מונעים כפליות

קודם חזרה:

0 הצלחה

-1-חורג טווח

2-כשל הקצהה (אין שינוי במבנה)

3-צלע קיימת כבר / לולאה שנייה

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

- הוספנו Makefile על מנת שנוכל להריץ את ה so
- הוספנו demo על מנת להמיחיש את מבנה הגרפּ .

שאלה 2:

יש מעגל אוילר אם ורק אם:

1. הגרף מחובר כשתעלמים מקודקודים מבודדים (דרגה 0).
2. כל הדרגות זוגיות לכל קודקוד מספר צלעות זוגי.

הערה פרקטית לימושו שלנו: דרשו גם שלגרף תהיה לפחות צלע אחת. מתמטית לפחותים מחשייבים גם גראף ריק/בודד כטורייאלי, אבל בקוד זה מוחזר "אין מעגל" כאשר צלעות.

הfonקציה `graph_has_euler_circuit` בודקת:

1. קישוריות של תת-הגרף.
 2. כל הדרגות זוגיות.
 3. יש לפחות צלע אחת.
- אם שהוא מזה לא מתקיים לא קיים מעגל אוילר ולכן `graph_find_euler_circuit` מחזיר 0.

: EdgeView :

- `[m] ev.edges[m]` מערך הצלעות הלא-מכוניות בגודל m

- `[v] ev.incid[v]` לכל קודקוד v רשימה של אינדקס צלעות שנוגעות בו.

הfonקציה `build_edge_view` :

- אם זו לולה, (`v==n`) ברשימות יש שתי הופעות. נ>-עלן סופרים "חצ'לוולאות", וכל שתים יוצרות צלע לולה אחת ב-, `edges` מוסיכים את האינדקס פעמיים ל-.`[n][n] incid[n]`.
- אם זו צלע רגילה, `v!=n` מוסיכים את הצלע עם אחת כשי->נדדי לא להכפיל, ומוכנסים את האינדקס גם ל-.`[n][incid[n]]` וגם ל-.`[v][incid[n]]`.

`graph_find_euler_circuit`

`[m] reused[m]` אם צלע כבר שומה.

`[ch] all[incid[n]]` קודקוד מורה-מקוםAiיפה הגענו בסריקה של `[v][incid[n]]`

`stack`

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

1. בחר נקודת התחלת בוחרים קודקוד שיש לו לפחות צלע אחת ומניחים אותו על מחסנית.
2. כל עוד יש لأن לכת – הולכים אחריה – חזירים אחריה מסתכלים על הקודקוד שבראש המחסנית.(n)
- אם יש מ- n צלע שלא השתמשנו בה עדין: נסמן את הצלע כמשומשת, עוברים לקצה השני שלו, (v) ומניחים את v על המחסנית.
- אם אין יותר צלעות פנויות מ- n אין لأن להתקדם, אז "חזירים אחריה" מוצאים את n מהחסנית ומוסיפים אותו לסוף רשימת התוצאה.(path)
3. בסיום – הופכים את הרשימה כשהמחסנית מתפרקנת, קיבלנו את המסלול ב- `path` אבל מהוסף להתחלה. הופכים את הסדר – וזה המעגל.

- הכתוב demo ו makefile להדמיה .

```
● 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 בדרכים:

- <vertices> n-מספר הקודקודים (חובה)
- <edges> e-מספר הקשתות המבוקש (חובה)
- <seed> z-זרע אקראי לשחזר אותו גרף בין הריצות (חובה)

אם חסר אחד מהפרמטרים (z, e, -) נדפס usage ונצא.

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

לאחר יצירה:

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

```
● 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 :

התבקשו למשרת ולקוח , כך שהלקוח שלוח לשרת גרף מסויים והשרת מחזיר האם יש מסלול אוילר בגרף שנשלח או לא , אם יש מסלול הלקוח מקבל אותו.

הסביר על המימוש :

שרות :

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

לקוח :

הלקוח פשוט שלוח לשרת את מספר הקודקודים ואת הצלעות הנבחרות

דוגמת הריצה :

```
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 קלייקה מקסימלית .

קלייקה מקסימלית : למצוא קבוצת קודקודים שכל זוג בה מחובר בקשר, בגודל המקסימלי.

הקוד עובד עם גראף Graph שמייצג כרשיימות שכנות. כדי לבדוק קלייקות ביעילות, הוא ממיר את הגרף זמני ל-**מטריצה שכנות של 1/0**.

build_adjacency_matrix : מאפס מטריצה לגודל חחיל-0. עובר על רשימות השכנות ומסמן $A[u][v] = 1$ אם קודקוד v מחובר לכל הקודקודים שכבר בקיימה הנוכחות.

בודק אם קודקוד v מחובר לכל הקודקודים שכבר בקיימה הנוכחות.

האלגוריתם:

1. אם הקלייקה הנוכחית גדולה מהטובה ביותר שמצאנו – מעדכנים.
2. מנסים להוסיף כל קודקוד מההווח $-1..n-1$ כ- v .
3. מושיפים רק אם מחובר לכלום בקיימה הנוכחית.
4. קוראים רקורסיבית שם והלאה ($+1$) כדי לשמר על סדר ולמנוע כפליות.

- **מחזיקים שלוש קבוצות:**

R הקליקה הנוכחית.

P מועמדים שאפשר עדין להוסיף.

X קודקודים שכבר "ניסינו" איתם.

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

2. מציאת מספר קליקות

ספירת כל הקליקות בכל הגדלים.

ספירת קליקות בגודל מסוים.

ספירת מושלמים (3-קליקות) ביעילות.

ספירת קשתות (2-קליקות).

בדיקה אם יש קליקה בגודל מסוים.

הdfsת ניתוח מפורט של התוצאות.

הקוד ממש מנגנון למספר קליקות בגרף על ידי בניית מטריצת שכנות וביצוע חיפוש רקורסיבי שימושי קודקודים רק אם הם יוצרים קליקה עם הקודקודים שכבר נבחרו. בדומה זו, נבדקת כל תתי-הקבוצות האפשריות של קודקודים אך נמנעת ספירה כפולה, כי החיפוש מתקדם תמיד קדימה בראשית הקודקודים. השימוש כולל ספירה של כל הקליקות בכל הגדלים, ספירה של קליקות בגודל מסוים, וגם אלגוריתמים ייעודיים ומהירים יותר למספר מושלמים וקשתות.

3. זרימה מקסימלית

הקוד זהה ממש חישוב **זרימה מקסימלית** בגרף (רשת זרימה) באמצעות אלגוריתם **Edmonds–Karp**, **Ford–Fulkerson** המשמשת ב-**BFS** למציאת מסלולים משפרים. שהוא וריאציה של פונקציית **find** המשמשת להDFS במקומות המרכזים במימוש:

1. **בנייה מטריצת קיבולות(Capacity Matrix)**

הfonקציה `capacity_matrix` build_capacity את רשימות השכנות של הגרף למטריצה חאצ'ובה שבה הערך `[v][u]` הוא הקיבולת של הקשת מ-`v` ל-`u` הקיבولات נקבעות לפי משקל הקשת-`edge_weight`, אפס אם אין קשת.

2. **חיפוש מסלול משפר(BFS)**

הfonקציה `bfs` מחפשת מסלול בו כל הקשתות עם קיבולת חיובית. היא מחזירה גם את מערך `parent` שמסמן את המסלול שנמצא.

3. **קביעת הזרימה במסלול**

הfonקציה `flow` מוצאת את הקיבולת המינימלית על המסלול שנמצא זהה ערך הזרימה שאפשר להוציא.

הקוד ממש את אלגוריתם **Edmonds–Karp** לחישוב הזרימה המקסימלית בין שני קודקודים בגרף, על בסיס גרסה **Ford–Fulkerson** המשמשת ב-**BFS** למציאת מסלולים משפרים. תחילת הגרף מומר למטריצת קיבולות לפי משקלן הקשותות, ולאחר מכן מבצע חיפוש חוזר אחר מסלולים מהרצף למקור (`source`) ליעד (`sink`) שביהם נותרת קיבולת חיובית. בכל מסלול צזה מחושבת הקיבולת המינימלית, הזרימה מתווספת לערך הכלול, והגרף הרזידואלי מתעדכן בהתאם. התהליך חוזר עד שאין עוד מסלול משפר, ואז הערך שנוצר מייצג את הזרימה המקסימלית האפשרית.

4. מיציאת משקל ב-*mst*

הקוד הזה מימוש חישוב עץ פורש מינימלי (MST) בגרף משוקל באמצעות האלגוריתם של פרים, תוך שימוש בתורה עדיפויות בצורת עירימת מינימום.

מהלך הפעולה:

1. בנית מטריצת משקלים הפונקציה `xix_weight_matrix_build` מקבלת את רשימות השכניםות של הגרף למטריצה חאצ' שבה כל תא $[v][u]$ מכיל את משקל הקשת מ- v ל- u (או 0 אם אין קשת).
 2. אתחול משתנים `mst_in` : 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. לולאתPrim מתחילה מקודקוד 0 ($key[0] = 0$) ווחפים אותו לתור העדיפויות עם משקל 0. בכל שלב מוצאים 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`. ווחפים לתור עדיפויות את הקודקודים עם המפתחות החדשים.
 4. בדיקת קשריות

5. **בנייה הטעאה**
 אם הגרף קשיר, יוצרים מערך של MST_Edge עם כל הקשתות שב- MST ומחשבים את המשקל
 הכלול (total weight).

6. פונקציות נוספות –

- `mst_weight` – משקל הצלול הכלול ב-MST.
- `graph_mst_result_free` – מחררת את הזיכרון של מערך הקשתות.
- `graph_print_mst` – מדפסה את משקל העץ ואת רשימת הקשתות.

הקוד מימוש את האלגוריתם של **פרימ** למציאת עץ פורש מינימלי (MST) בגרף משוקל, תוך שימוש בערימת מינימום (Priority Queue) לניהול הקודקוד הבא להוספה. תחילת הגרף מומר למטריצת משקלים בהתחם למשקל הקשאות, ואז מתחילה מקובוד 0 ומרחיבים את העץ בהדרגה על ידי בחירת הקשת הזולה ביותר שמחברת קודקוד חדש ל-MST. מערכי עזר עוקבים אחריו הקודקודים שכבר בוצע, המפתח המינימלי של כל קודקוד, והקודקוד ההורה שממנו מגיעה הקשta. בסיום, אם הגרף קשור, נבנה מערך הקשאות של ה-MST ונחשב את משקלו הכלול.

Factory Pattern – מבנה ייצור שפה יש "מפעל" שמקבל פרמטר ומוציא אובייקט מתאים מבל' שהקוד הקורא יctrar לדעת איך האובייקט נבנה בפועל. הרעיון – להפריד בין הלוגיקה של בחירת האובייקט/יצירתו לבין השימוש בו, וכך להקל על שינוי והרחבה של סוג האובייקטים.

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

Factory.c

הקוד משלב בין Factory כדי להריץ אלגוריתמים שונים על גרף. ה- Strategy מקבל מזהה אלגוריתם, ממפה אותו לסוג מתאים, בודק אם הוא נתמך, ויציר אובייקט Strategy מתאים שמכיל את שם האלגוריתם, תיאורו, ואת פונקציית ההרצה שלו. ה- AlgorithmContext מועבר לאובייקט Shmaritz 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 בזיכרון אחידה וგמישה.

algorithm_strategy.c

הקוד הזה מימוש את **תבניתה Strategy** עבור אלגוריתמים על גרפים, ומספק דרך אחת לבחור ולהריץ אלגוריתם בזמן ריצה בלי שהקוד הקורא יצרך לדעת מהי המימוש הפנימי שלו.

איך זה עובד:

1. הגדרת מבנה **Strategy**

בקובץ `AlgorithmStrategy.h` הוא מכיל:

מצביע לפונקציית ההרצה, שמקבלת מצביע לגרף ומחזירה מחרוזת תוצאה.

שם ותיאור של האלגוריתם.

מזהה ליזחיי חד-משמעותי.

.2

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

בקובץ `Algorithm_Context.h` מוצג את הקונטקט.

מצביע `Algorithm_Context_Set_Strategy` אסטרטגיה.

מצביע `Algorithm_Context_Execute` את האלגוריתם הנוכחי על הגרף.

3. מימוש אסטרטגיות בפועל

בקובץ `Algorithm_Strategy.c` מימושים ממשיים עבור כל אלגוריתם:

○ `euler_strategy_execute` – מעגל אוילר.

○ `maxflow_strategy_execute` – ממוחשב זרימה מקסימלית.

○ `mst_strategy_execute` – ממוחשב עץ פורש מינימלי.

○ `maxclique_strategy_execute` – ממוחצא גודל קלייקה מקסימלית.

○ `cliquecount_strategy_execute` – סופר את כל הקלייקות.

כל פונקציה יוצרת מחרוזת עם תוצאה פורמטיבית, אותה מוחזרים למשתמש.

לסיכום

הקוד מימוש את **תבנית Strategy** להפעלת אלגוריתמים שונים על גרפים, כאשר כל אלגוריתם מוגדר כאובייקט אסטרטגיה הכלול פונקציית הריצה, שם, תיאור ומזהה. האסטרטגיות נשמרות במערך מרצוי, ומודול `AlgorithmContext` מנהל את בחירת האסטרטגיה והפעלה בפועל על הגרף. כך ניתן לבחור ולהחליף אלגוריתם

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

Server.c client.c

צד הלקוח

הלוקוח נדרש לתחבר לשרת אלגוריתמים ב- TCP לבחור אלגוריתם, להזין נתוני גרפ בהתאם לפרטוקול (לא ממושך או ממושך), לשЛОח את הבקשה, ולקבל את התוצאה. הוא מטפל בשני סוגי פרוטוקולים:

- **ממושך** – (Max Flow, MST) שולח מספר קדיודים, מספר קשתות ורשימת קשותות עם משקלים.
- **לא ממושך** – (Euler, Max Clique, Clique Count) שולח מטריצת סמיcioיות. בסוף מקבל את תשובה השרת עם סטטוס ואורך מחזורת התוצאה ומדפיס אותה למשתמש.

אפשרות למשתמש לבחור אלגוריתם ולהזין נתונים גרפ, מטפלת בהכנות הבקשה לפי פרוטוקול מתאים, ושולחת לשרת לקבלת תוצאה.

צד השרת

השרת מזין לחיבורים ב- TCP ומקבל בקשות להפעלת אלגוריתמים. הוא מזהה את סוג האלגוריתם לפי ה- ID וASICBL ועובד את הבקשה בהתאם לפרטוקול:

- **ממושך** – בונה גרפ עם קשותות ומשקלים מעודכנים.
- **לא ממושך** – בונה גרפ ממטריצת סמיcioיות.

לאחר בניית הגרף, השירות מפעיל את האלגוריתם המבוקש באמצעות תבניות Factory + Strategy ומחזיר ללקוח את התוצאה במבנה אחד ([סטטוס][אורך][מחזורות]).

שימוש השירות גרפים המבוסס על TCP שמקבל בקשות להפעלת אלגוריתמים שונים, בונה את הגרף לפי סוג הנתונים (ממושך או לא), מרץ 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. השירות(server.c)

השירות קורא את ה- ID והנתונים, בונה מבנה Graph (עם משקלים/מטריצה בהתאם), ואז מפעיל:

AlgorithmType algorithm_id – AlgorithmFactory בבודק תמייה, ויוצר את אובייקט ה- Strategy המתאים.

(Max Flow : מציב את האסטרטגייה בקונטקסט ומרץ אותה על הגרף Flow / MST / Max Clique / Clique Count / Euler).

3. תוצאה חוזרת ללקוח

השירות מחזיר מענה אחד וה לוקוח מציג את המחרוזת למשתמש.

דוגמאות הרצה :

.1 הפעלת השירות והקלינט

```
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. מופר 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 לישם מספר הליכים
ב. השרת יקבל גרפ או בקשה ליצור גרפ אקראי מחלוקת ויספק תשובה לכל 4 האלגוריתמים המימושים בסעיף 7 בזורה של LF

האפשרויות 6 ו- 7 הן "קיזורי דרך" לייצור גרפ אקראי ע"י השרת:

- **בחירה 6** שליחת בקשה לשרת שיהיא ייצור גרפ לא-מומשקל (Unweighted) באופן אקראי לפי פרמטרים שתז"ן מספר צמתים, מספר קשתות, seed, במקומ שתבנה מטריצת שכנות ידנית לצד הלקוח, הלקוח רק מבקש "Random unweighted graph".
- **בחירה 7** שליחת בקשה לשרת לייצור גרפ ממושקל (Weighted) אקראי לפי פרמטרים.-can השרת יוצר גרפ עם קשתות ומשקלים אקראיים (עד משקל מקסימלי מסוים), שוב לפ' 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
```

: 9 שאלות

```
bash: ./server_pipeline.dSYM/: ls 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
CliquesCount: Total=13
=====
```