מטלה 2 – חישוב ביולוגי

מגישה: דנה בקשי 322809997

קישור לגיט: https://github.com/danabakshe/Ex2_Biological-Computation.git

הסבר על מימוש הקוד :

במטלה זו התבקשתי לכתוב תוכנית בשפת פיתון, שמבצעת שני שלבים עיקריים: ראשית, יצירה של כל המוטיפים האפשריים (כלומר, תתי־גרפים מכוונים, מחוברים ולא־איזומורפיים) בגודל נתון n. ושנית, בדיקה כמה פעמים כל אחד מהמוטיפים מופיע בתוך גרף קלט גדול יותר.

בשלב הראשון של המימוש, יצרתי את כל הקומבינציות האפשריות של קשתות בין n קודקודים – ללא לולאות עצמיות. מספר הקשתות האפשריות הוא n(n−1) ולכן מספר הגרפים האפשריים הוא בקירוב $2^{n(n-1)} - 1$. עבור כל תת־קבוצה של קשתות נוצר גרף מכוון.

לאחר מכן בדקתי האם הגרף מחובר חלש (באמצעות networkx.is_weakly_connected) והאם הוא איזומורפי לגרף שכבר הופיע קודם (באמצעות networkx.is_isomorphic) רק אם שני התנאים התקיימו – הגרף נשמר כמוטיף חדש. לבסוף, כל המוטיפים נשמרו לקובץ.

בשלב השני, קראתי את גרף הקלט מקובץ טקסט בפורמט של זוגות קודקודים המייצגים קשתות מכוונות. עבור כל קבוצה של n קודקודים בגרף הזה, יצרתי את תת־הגרף המוגבל לצמתים אלו, בדקתי האם הוא מחובר חלש, ואם כן בדקתי גם האם הוא איזומורפי לאחד מהמוטיפים שנשמרו מקודם. במקרה שכן, הגדלתי את מונה ההופעות של אותו מוטיף.

לבסוף, הפלט נכתב לקובץ חדש בפורמט המבוקש, כולל כמה פעמים כל מוטיף הופיע בגרף.

מבחינת ביצועים – בגלל הגידול האקספוננציאלי במספר הגרפים האפשריים, בדקתי את זמני הריצה בפועל. עבור n=4, התוכנית סיימה את פעולתה תוך 2.36 שניות בלבד. לפי חישוב ליניארי ביחס למספר הגרפים, הראיתי שעבור n=5 זמן הריצה צפוי להיות כ־10 דקות, ואילו עבור n=6 כ־171 שעות בערך . לכן הסקתי שה-n המקסימלי שהתוכנית מסוגלת לסיים בתוך שעה הוא 5, ואותו ערך נשמר גם אם מגבילים את זמן הריצה ל־2, 4 או 8 שעות.

*השתמשתי בעזרה של כלי בינה מלאכותית (ChatGPT) על מנת להבין טוב יותר את הדרישות ולתכנן נכון את מבנה הקוד.

שאלה 1 סעיף a:

קוד פיתון עם דוגמת הרצה עבור n=2

```python
import networkx as nx
from itertools import combinations, product
import os

def generate_all_directed_graphs(n):
    nodes = list(range(1, n + 1))
    all_possible_edges = [(i, j) for i in nodes for j in nodes if i != j]

    all_graphs = []
    for k in range(1, len(all_possible_edges) + 1):
        for edges in combinations(all_possible_edges, k):
            G = nx.DiGraph()
            G.add_nodes_from(nodes)
            G.add_edges_from(edges)
            all_graphs.append(G)
    return all_graphs

def is_new_graph(graph, graph_list):
    for existing in graph_list:
        if nx.is_isomorphic(graph, existing):
            return False
    return True

def generate_connected_unique_graphs(n):
    all_graphs = generate_all_directed_graphs(n)
    unique_connected = []
    for g in all_graphs:
        if nx.is_weakly_connected(g) and is_new_graph(g, unique_connected):
            unique_connected.append(g)
    return unique_connected

def save_graphs_to_file(graphs, n, filename="motifs_output.txt"):
    with open(filename, "w") as f:
        f.write(f"n={n}\n")
        f.write(f"count={len(graphs)}\n")
        for i, g in enumerate(graphs, 1):
            f.write(f"# {i}\n")
            for u, v in g.edges():
                f.write(f"{u} {v}\n")

#example n=2
if __name__ == "__main__":
    n = 2
    motifs = generate_connected_unique_graphs(n)
    save_graphs_to_file(motifs, n)
    print(f"Saved {len(motifs)} connected motifs for n={n} to file.")
```

פלט:

```
n=2
count=2
# 1
1 2
# 2
1 2
2 1
```

שאלה 1 סעיף b:

קוד פיתון עם דוגמת הרצה עבור n=1 עד n=4:

```
import networkx as nx
from itertools import combinations

def generate_all_directed_graphs(n):
    """
    Generate all possible directed graphs with n nodes and all edge
combinations.
    """
    nodes = list(range(1, n + 1))
    all_possible_edges = [(i, j) for i in nodes for j in nodes if i
!= j]

    all_graphs = []
    for k in range(1, len(all_possible_edges) + 1):
        for edges in combinations(all_possible_edges, k):
            G = nx.DiGraph()
            G.add_nodes_from(nodes)
            G.add_edges_from(edges)
            all_graphs.append(G)
    return all_graphs

def is_new_graph(graph, graph_list):
    """
    Check if the graph is isomorphic to any graph already in the
list.
    """
    for existing in graph_list:
        if nx.is_isomorphic(graph, existing):
            return False
    return True

def generate_connected_unique_graphs(n):
    """
    Generate all connected and non-isomorphic directed graphs with n
nodes.
    """
    all_graphs = generate_all_directed_graphs(n)
    unique_connected = []
    for g in all_graphs:
        if nx.is_weakly_connected(g) and is_new_graph(g,
unique_connected):
            unique_connected.append(g)
    return unique_connected

def save_graphs_to_file(graphs, n):
    """
    Save the list of graphs in the required textual format to a flat
file.
    """
    filename = f"motifs_n={n}.txt"
    with open(filename, "w") as f:
        f.write(f"n={n}\n")
        f.write(f"count={len(graphs)}\n")
        for i, g in enumerate(graphs, 1):
            f.write(f"# {i}\n")
            for u, v in g.edges():
                f.write(f"{u} {v}\n")
    print(f"Saved {len(graphs)} motifs to {filename}")
    return filename
```

```
def main():
    """
    Main function to generate motifs for n = 1 to 4.
    """
    for n in range(1, 5):
        print(f"Generating motifs for n={n}...")
        motifs = generate_connected_unique_graphs(n)
        save_graphs_to_file(motifs, n)

if __name__ == "__main__":
    main()
```

פלט עבור n=1:

```
n=1
count=0
```

פלט עבור n=2:

```
n=2
count=2
# 1
1 2
# 2
1 2
2 1
```

פלט עבור n=3:

```
n=3
count=13
# 1
1 2
1 3
# 2
1 2
2 3
# 3
1 2
3 2
# 4
1 2
1 3
2 1
# 5
1 2
1 3
2 3
# 6
1 2
2 1
3 1
# 7
1 2
2 3
3 1
# 8
1 2
1 3
2 1
2 3
# 9
```

```
1 2
1 3
2 1
3 1
# 10
1 2
1 3
2 1
3 2
# 11
1 2
1 3
2 3
3 2
# 12
1 2
1 3
2 1
2 3
3 1
# 13
1 2
1 3
2 1
2 3
3 1
3 2
```

פלט עבור n=4: ארוך לקובץ זה ולכן העלתי בגיט את הפלט.

<u>שאלה 1 סעיף c:</u>

עבור סעיף זה שיניתי את הmain להיות:

```
def main():
    for n in range(1, 4):
        print(f"Generating motifs for n={n}...")
        start = time.time()
        motifs = generate_connected_unique_graphs(n)
        end = time.time()
        print(f"n={n}, number of motifs: {len(motifs)}, runtime: {end
- start:.2f} seconds")
        save_graphs_to_file(motifs, n)
```

הפלט יצא:

...Generating motifs for n=1
n=1, number of motifs: 0, runtime: 0.00 seconds
Saved 0 motifs to motifs_n=1.txt
...Generating motifs for n=2
n=2, number of motifs: 2, runtime: 0.00 seconds
Saved 2 motifs to motifs_n=2.txt
...Generating motifs for n=3
n=3, number of motifs: 13, runtime: 0.00 seconds
Saved 13 motifs to motifs_n=3.txt
...Generating motifs for n=4
n=4, number of motifs: 199, runtime: 2.36 seconds
Saved 199 motifs to motifs_n=4.txt

התוכנית מייצרת את כל הגרפים המחוברים ,המכוננים כגרפים מכוונים ,ושאינם איזומורפיים אחד לשני ,בגודל n.
מספר הקשתות האפשריות בגרף מכוון עם n צמתים (ללא לולאות עצמיות) הוא:

$$n \bullet (n-1)$$

לכן מספר הגרפים שנבדקים הוא בקירוב $\sum_{k=1}^{n(n-1)} \binom{n(n-1)}{k} = 2^{n(n-1)} - 1$

עבור כל גרף כזה, מבוצעות שתי בדיקות יקרות חישובית:

1. האם הגרף מחובר חלש בדיקה בעזרת DFS.
2. האם הוא איזומורפי לגרף שכבר הופיע.

המספר העצום של גרפים והצורך לבדוק איזומורפיזם עבור כל אחד מהם גורם לעלייה חדה בזמן הריצה עם כל גידול בערך n.

עבור n=4 מספר הגרפים שנבדקים הוא בקירוב $2^{4(4-1)} - 1 = 4095$ **ולפי הטיימר ששמתי** בתוכנית זה לקח 2.36 שניות.

עבור n=5 מספר הגרפים שנבדקים הוא בקירוב $2^{5(5-1)} - 1 = 1048575$ **ולכן זה כבר יקח** בערך 10 דקות לפי החישוב ביחס לn=4:

$\frac{1048575}{4095}$ * 2.36 sec =604.306 sec = 10 min

עבור n=6 מספר הגרפים שנבדקים הוא בקירוב $2^{6(6-1)} - 1 = 1073741823$ **ולכן זה כבר** יקח $\frac{1073741823}{4095}$ * 2.36 sec = 618810sec = 171hours ולכן התשובה היא **שעבור n=5 התוכנית תושלם לאחר פחות משעה אך עבור n גדול יותר כבר לא.**

<u>שאלה 1 סעיף d:</u>

ככל שn גדל כך גם מספר הגרפים שצריך לבדוק גדל בצורה אקספוננציאלית ונהיה עצום.
לפי החישובים של סעיף קודם נוכל להגיד **שעבור שעתיים, 4 שעות ו8 שעות התוכנית תסתיים עבור n=5** ולא עבור n גדול יותר. (מכיוון שעבור n=6 כבר צריך 171 שעות בערך)

שאלה 2:

קוד פיתון עם דוגמת הרצה שניתנה בתרגיל הבית:

```python
import networkx as nx
from itertools import combinations

def load_graph_from_file(filename):
    """
    Load a directed graph from a file containing lines of 'u v'.
    """
    G = nx.DiGraph()
    with open(filename, 'r') as f:
        for line in f:
            u, v = map(int, line.strip().split())
            G.add_edge(u, v)
    return G

def load_motifs_from_file(motif_file):
    """
    Load all motifs from a motif file (format as in question 1).
    Returns a list of DiGraph objects.
    """
    motifs = []
    current_edges = []
    with open(motif_file, 'r') as f:
        for line in f:
            line = line.strip()
            if line.startswith('n=') or line.startswith('count='):
                continue
            elif line.startswith('#'):
                if current_edges:
                    G = nx.DiGraph()
                    G.add_edges_from(current_edges)
                    motifs.append(G)
                    current_edges = []
            else:
                u, v = map(int, line.split())
                current_edges.append((u, v))
        if current_edges:
            G = nx.DiGraph()
            G.add_edges_from(current_edges)
            motifs.append(G)
    return motifs

def count_motif_instances(graph, motifs, n):
    """
    Count how many times each motif of size n appears in the graph.
    """
    counts = [0] * len(motifs)
    for nodes in combinations(graph.nodes, n):
        subgraph = graph.subgraph(nodes).copy()
        if not nx.is_weakly_connected(subgraph):
            continue
        for i, motif in enumerate(motifs):
            if nx.is_isomorphic(subgraph, motif):
                counts[i] += 1
                break  # assume only one matching motif
    return counts

def output_motif_counts(motifs, counts, motif_file, output_file):
    """
    Write output in required format: motif structure + count=m.
```

```
    """
    with open(output_file, 'w') as f:
        for i, (motif, count) in enumerate(zip(motifs, counts), 1):
            f.write(f"# {i}\n")
            f.write(f"count={count}\n")
            for u, v in motif.edges():
                f.write(f"{u} {v}\n")

#example
if __name__ == "__main__":
    n = 3  # or any desired size
    motif_file = f"motifs_n={n}.txt"
    input_graph_file = "input_graph.txt"
    output_file = "motif_counts_output.txt"

    graph = load_graph_from_file(input_graph_file)
    motifs = load_motifs_from_file(motif_file)
    counts = count_motif_instances(graph, motifs, n)
    output_motif_counts(motifs, counts, motif_file, output_file)
    print(f"Finished counting motif instances. Output written to:
{output_file}")
```