חן כהן גרשון 208955732 אלמוג שרוני 208611764

חישוב ביולוגי- מטלה 2

1) a.

להלן הקוד שכתבנו עבור חלק זה:

```
import timeit
import networkx as nx
import itertools
import time
import os
import matplotlib.pyplot as plt
import numpy as np
import tqdm


def mask_list(n, mask):
    # return a list that gives res[i]==True if the i-th edge is in the graph
    res = [False] * (n ** 2 - n)
    for i in mask:
        res[i] = True
    return res


def k_digraphs(n, k):
    '''
    generate all the directed graphs with exactly k edges
    '''
    possible_edges = [
        (i, j) for i, j in itertools.product(range(n), repeat=2) if i != j
    ]

    # go over all the possibilities of k edges out of all the n*(n-1) edges:
    for edge_mask in itertools.combinations(range(n * n - n), k):
        # The result is already sorted
        yield tuple(edge for include, edge in zip(mask_list(n, edge_mask), possible_edges) if include)


def unique_motifs(n, k):
    '''
    generate all the unique graphs with exactly k edges (up to isomorphism)
    '''
    already_seen = set()
    for graph in k_digraphs(n, k):
        if graph not in already_seen:
            # add all permutation of the current graph to the set of graphs we have already seen
            # (all permutations=all graphs isomorphic to the current one)
            already_seen |= {
                tuple(sorted((perm[i], perm[j]) for i, j in graph))
                for perm in itertools.permutations(range(n))
            }
            yield graph
```

```
def k_motifs(n, k):
    '''
    return all directed graphs with exactly k edges which keep the graph with n nodes connected
    '''
    k_graphs = map(nx.DiGraph, unique_motifs(n, k))
    connected_graphs = filter(nx.is_weakly_connected,
                              filter(lambda g: len(g) == n,
                                     k_graphs)
                              )
    return connected_graphs


def all_motifs(n, format='list'):
    '''
    return all graphs of n nodes which are connected, that are unique up to isomorphism
    list them and the sum of how many are there
    '''
    sum = 0
    str_all_motifs = ''
    for k in range(n - 1, n ** 2 - n + 1):
        # Go over all the graphs of size k \in [n-1,n**2-n] (if k<n-1 the graph cannot be connected)
        if format == 'list':
            cur_str, cur_count = k_motifs_to_str(n, k)
        else:
            motifs = k_motifs(n, k)
            cur_str, cur_count = sum_k_motifs(motifs)
        str_all_motifs += cur_str
        sum += cur_count
    return str_all_motifs, sum


def sum_k_motifs(motifs):
    count = 0
    str_k_motifs = ''
    for motif in motifs:
        str_k_motifs += motif_to_str(motif)
        count += 1
    return str_k_motifs, count


def motif_to_str(motif):
    motif_str = f'#k={motif.number_of_edges()}\n'
    for u, v, d in motif.edges.data():
        motif_str += f'{u} {v}\n'
    return motif_str


def k_motifs_to_str(n, k, verbose=False):
    if n <= 1:
        return '', 0
    motifs = k_motifs(n, k)
    res = f'#{k}\n'
    count = 0
    for motif in motifs:
        edges = list(motif.edges())
```

```
        res += f'{edges}\n'
        count += 1
    return res, count


def main_n(n, format='list'):
    '''
    give the result for the question for graph of size n
    '''
    res_str, count = all_motifs(n, format='list')
    res_str = f'n={n}\ncount={count}\n' + res_str

    return res_str


def save_motifs(n, path='.'):
    '''
    save a file with the details of graph of n nodes
    '''
    format = 'D-%d-%m-T-%H-%M-%S'
    stamp = time.strftime(format, time.localtime())
    file_name = f'M-{n}-{stamp}.txt'
    full_path = os.path.join(path, file_name)
    with open(full_path, 'w') as f:
        f.write(main_n(n))


def save_motifs_range(start, end=None, path='.', verbose=False):
    '''
    save a file with details on all graphs in the range [start,end]
    or [1,start] if end is None
    '''
    if end is None:
        start, end = 1, start
    assert end >= start
    format = 'D-%d-%m-T-%H-%M-%S'  # D-{day}-{month}-T-{hour}-{minute}-{second}
    stamp = time.strftime(format, time.localtime())
    file_name = f'M-{start}to{end}-{stamp}.txt'  # M-{motif range}-D-{day}-T-{time of day}
    full_path = os.path.join(path, file_name)
    res = ''
    for n in range(start, end + 1):
        with open(full_path, 'a') as f:
            res = main_n(n, verbose)
            f.write(res)
            f.write('\n')

#save_motifs_range(5)


# run on different n's and save the running time
running_times = np.zeros(5)
for n in tqdm.tqdm(range(1, 6)):
    start = timeit.default_timer()
    save_motifs(n)
    end = timeit.default_timer()
```

```
    running_times[n - 1] = end - start

# plot the running time
plt.plot(range(1, 6), running_times)
plt.xlabel('n')
plt.ylabel('running time [sec]')
plt.title('running time as a function of n')
plt.show()

# save the running time to a file and note which n the running time is for
with open('running_times.txt', 'w') as f:
    f.write('n\ttime\n')
    for n, t in enumerate(running_times):
        f.write(f'{n + 1}\t{t}\n')
```

התוכנית שכתבנו, מייצרת ושומרת את כלל המוטיבים (תתי הגרפים) של גרף מכון בעל n קודקודים .
על מנת לבצע זאת בקלות יחסית, השתמשנו בספריית networkx של python.

הקוד שלנו עובר על ככל המוטיבים האפשריים ובעצם מזהה את המוטיבים הייחודיים (כלומר מבין כל האפשרויות שחלקן חוזרות על עצמן, הוא מסנן את "השכפולים" על מנת למנות רק את המוטיבים הייחודיים).

להלן הסבר הפונקציות בקוד:

1. mask_list : הפנוקציה הזו מקבלת את n – כמות הקודקודים ורשימה mask בעלת ערכים בוליאניים של true,false
הפנוקציה בודקת אילו קשתות מופיעות בגרף שאנו בוחנים.
היא משיגה זאת ע"י שימוש באיטרציות על כל הקשתות. האפשריות ומעלה ל-true את האינדקס שהקשת שלו מופיעה.

2. K_digraphs: הפנוקציה הזו מייצרת את כל הגרפים המכוונים עם בדיוק k קשתות.
עבור המימוש, השתמשנו ב-itertools.combinations כדי לבצע איטארציות לכל הקובינציות האפשריות של k קשתות עד ל-$n^2 - n$ קשתות אפשריות.
כל קומביצניה מומרת לגרף מכוון אשר קשתותיו מאוכסנות ברשימה possible_edges שמכילה את כל הקשתות האפשריות לגרף עם n קודקודים.
הפונקציה מחזירה כל תת גרף שמיוצר תוך שימוש ב-yield.

3. Unique_motifs : מקבלת כקלט את n- כמות הקודקודים ו-k – כמות הקשתות בגרף.
הפונקציה מייצרת את כל המוטיבים האפשריים עם בדיוק k קשתות (עד כדי איזומורפיזם) .

הפנוקציה הזו, קוראת לפונקציה k_digraphs, ובעזרתה את כל הגרפים האפשריים עם k קשתות ועוקבת אחרי המוטיבים שהם 'unique' בעזרת משתנה מטיפוס Set ששמו הוא already_seen.
לכל גרף שכזה, הפונקציה מייצרת ועוברת כל כל הפרמוטציות של הקודקודים ומוסיפה את כל תתי הגרפים האיזומורפיים (מוטיבים זהים עד כדי שמות הקודקודים) מוסיפה אותו ל-already_seen .
לבסוף הפונקציה עושה yield לכל מוטיב יחודי.

4. K_motifs : מקבלת כקלט את n- כמות הקודקודים ו-k – כמות הקשתות בגרף.
הפונקציה מחזירה את כל הגרפים המכוונים עם בדיוק k קשתות.
הפונקציה משתמשת בפונקציה `unique_motifs` מסעיף 3.
הפונקציה שלנו עושה filtering לגרפים שהם לא "weakly connected"
(אלו גרפים מכוונים שלא ניתן להתחיל מסלול מכל קודקוד שרירותי וממנו להגיע לכל שאר הקודקודים בגרף)
או שיש להם מספר קודקודים שונה מ-n.

5. All_motifs : מקבלת את הפורמט שבו היא מחזירה את התוצאות ואת n
הפונקציה מחזירה את כל הגרפים עם n קודקודים שהם קשירים ויחודיים עד כדי איזמורפיזם.
הפונקציה בנוסף, מייצרת ושומרת את כל המוטיבים לכל מספר קשתות k בגרף.
כאשר $k \in [n-1, n^2-n]$
הפונקציה מחזירה את התוצאה כמחרוזת משורשרת של מוטיבים ומספרם שבהמשך תודפס אל קובץ הפלט.

6. Sum_k_motifs : הפונקציה מקבלת אוסף של מוטיבים ומחזירה את הייצוג של כל מוטיב כמחרוזת .
לכל מוטיב, היא גם סופרת כמה פעמים הוא מופיע .
על מנת להשיג זאת, היא עוברת באיטרציות על כל המוטיבים וממירה כל מוטיב למחרוזת בעזרת הפונקציה motif_to_str ומקדמת את המונה.

7. Motif_to_str : מקבלת כקלט את רשימצ המוטיבים.
כפי שהסברנו בסעיף 6, הפונקציה הזו מבצעת את ההמרה בפועל של המוטיב למחרוזת.
היא עוברת על כל הקשתות של המוטיב שאותו היא בוחנת ובונה עבורו ייצוג ע"י מחרוזת תוך שרשור ה-Source node וה-Target node של כל קשת.

8. K_motifs_to_str : מקבלת כקלט n,k כמו קודם.
הפונקציה ממירה את המוטיב עם בדיוק k קשתות לכדי ייצוגם ע"י מחרוזת
היא מייצרת את כל המוטיבים עם k קשתות לגרף עם n קשתות תוך שימוש בפונקציה k_motifs מסעיף 4 .
לכל מוטיב, נמיר את הקשתות למחרוזות ונקדם את המונה של אותו מוטיב
הפונקציה מחזירה את הייצוג המשורשר של כלל הקשתות ואת המונה.

9. Main_n : מקבלת כקלט n ופורמט השמירה של המידע מאותחל דיפולטיבית לרשימה.
הפונקציה מרכזת את כלל הפונקציות שעליהן הסברנו, היא קוראת ל-all_motifs עם הפורמט המבוקש ומוסיפה את מספר הקודקודים n, את כמות המוטיבים ומחזירה את המחרוזת המלאה של מוטיבים עבור ה- n הנתון.

10. Save_motifs: הפונקציה הנ"ל שומרת את מחרוזת הפלט בתוך קובץ לפי הפורמט שנדרשנו תוך שמירת שם הקובץ עם פרמטים מזהים יחודיים כמו זמן היצירה ותאריך על מנת שנוכל להבדיל ביניהם.

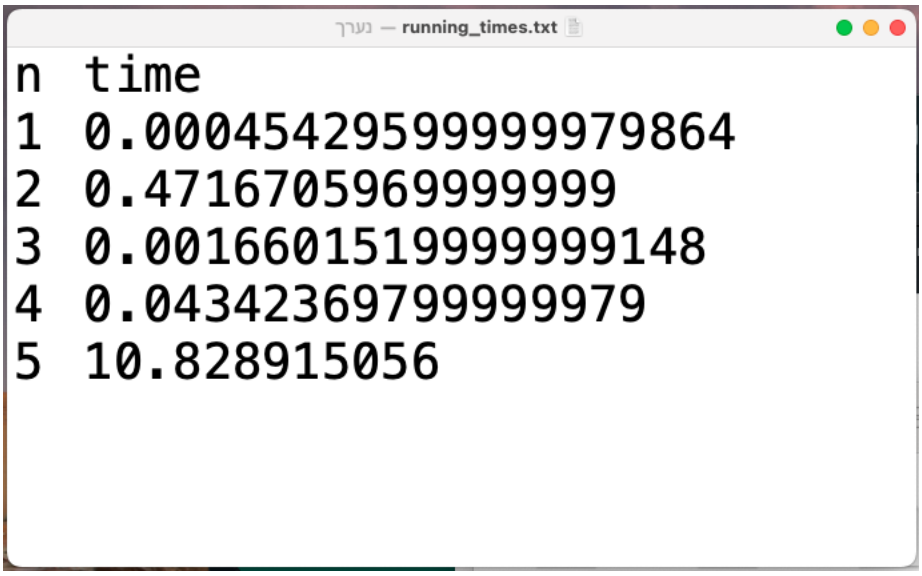
11. Save_motifs_range :

הפונקציה הזו, שומרת לקובץ את הפרטים של כל הגרפים בטווח `[start, end]` או `[1, start]` .
אם end הוא None היא עושה איטרציות על גדלי הגרפים וקוראת ל-save_motifs לכל גודל, ע"י שרשור התוצאה לקובץ הפלט שלנו.

b. את הפלטים, נצרף ע"י 4 קבצי טקסט נפרדים כדי לא להעמיס את הדו"ח.

c. המספר המקסימלי שאליו הצלחנו להגיע בטווח של שעת זמן ריצה היא $n = 5$ שאת הריצה עליו, השלמנו תוך 11 שניות בערך.

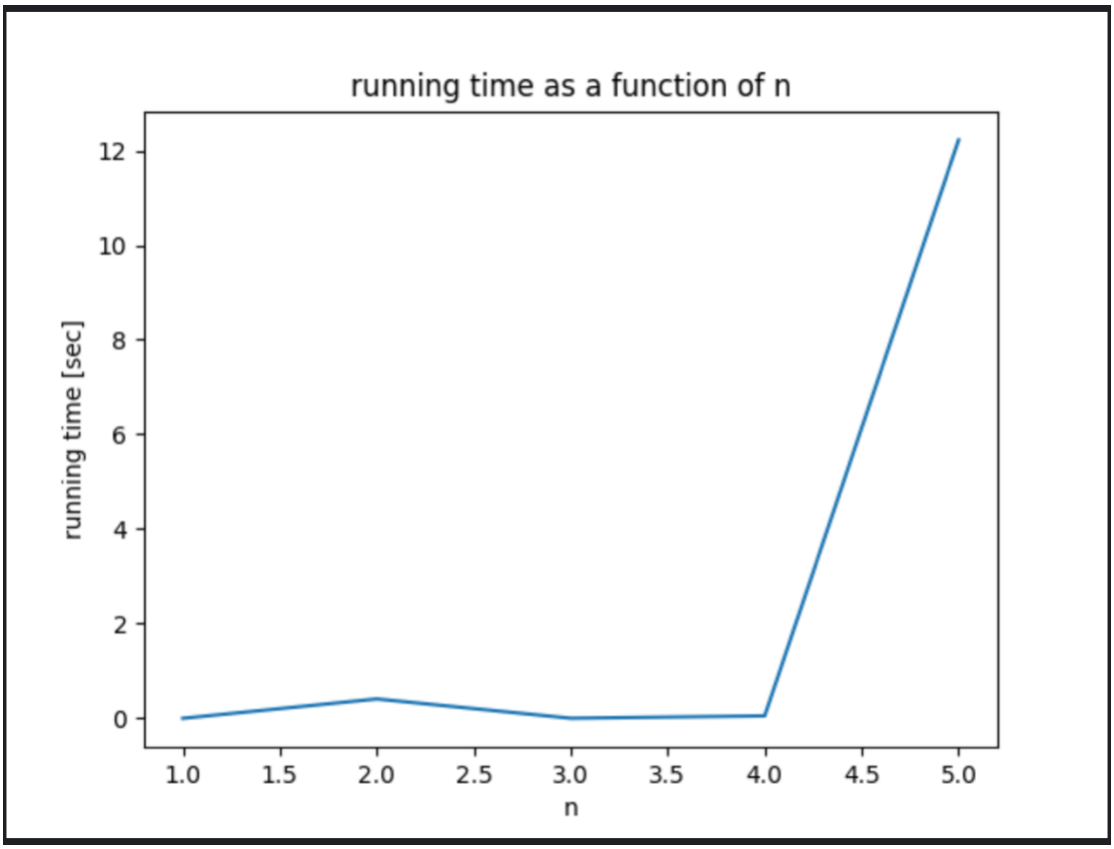
להלן זמן הריצה המדוייק מתוך הקוד שלנו:



ניתן לראות כי זמן הריצה הוא 10.8 שניות.

d. ניסינו להריץ את $n = 6$ עד ל-8 שעות והקוד עדיין לא סיים לרוץ כך שערך ה-n הקסימלי שעבורו הקוד שלנו רץ בזמן סביר הוא $\boldsymbol{n = 5}$.

בנוסף, בנינו גרף של זמן הריצה כפונקציה של n, הגרף מציג את זמני הריצה עד ל-n המקסימלי שאליו הגענו.



ניתן לראות את העלייה האקספוננציאלית ככל ש- n גדל.

2. להלן הקוד שכתבנו עבור חלק 2 :

```
import networkx as nx
import itertools
import time
import os
import matplotlib.pyplot as plt
import numpy as np

#define global variables
graph = nx.DiGraph()

def mask_list(n, mask):
    # return a list that gives res[i]==True if the i-th edge is in the graph
    res = [False] * n
    for i in mask:
        res[i] = True
    return res


def k_digraphs(n, k, ):
    '''
    generate all the directed graphs with exactly k edges
    '''
    global graph
    possible_edges =  graph.edges

    # go over all the possibilities of k edges out of all the n*(n-1) edges:
    for edge_mask in itertools.combinations(range(len(possible_edges)), k):
        # The result is already sorted
        yield tuple(edge for include, edge in zip(mask_list(len(possible_edges), edge_mask), possible_edges) if include)

def ret_and_print(graph,perm):
```

```
  # print(f'{perm=}\n{graph=}')
  ret = tuple(sorted((perm[i-1], perm[j-1]) for i, j in graph))
  return ret

def unique_motifs(n, k):
    '''
   generate all the unique graphs with exactly k edges (up to isomorphism)
   '''
   global graph
    already_seen = set()
    for k_graph in k_digraphs(n, k):
        if k_graph not in already_seen:
            # add all permutation of the current graph to the set of graphs we have already seen
            # (all permutations=all graphs isomorphic to the current one)
            currently_seen = set()
            # {
            #      ret_and_print(k_graph,perm)
            #      for perm in itertools.permutations(graph.nodes)
            # }
            c = 0
            for perm in itertools.permutations(graph.nodes):
              cur_graph = tuple(sorted((perm[i-1], perm[j-1]) for i, j in k_graph))
              cur_subgraph = graph.edge_subgraph(cur_graph)
              if len(cur_subgraph.edges)==len(cur_graph) and cur_graph not in currently_seen:
                  # print(f'{perm=}')
                  # print(f'{cur_graph=}')
                  # print(f'{graph.edge_subgraph(cur_graph).edges=}')
                  c+=1
                  currently_seen.add(cur_graph)

            already_seen |= currently_seen
            yield k_graph,c


def k_motifs(n, k):
    '''
   return all directed graphs with exactly k edges which keep the graph with n nodes connected
   '''
   # print(list(unique_motifs(n, k)))
    k_graphs = map(lambda t: (nx.DiGraph(t[0]),t[1]), unique_motifs(n, k))
    # print(list(k_graphs))
    connected_graphs = filter(lambda t: nx.is_weakly_connected(t[0]),
                              filter(lambda t: len(t[0].nodes) == n,
                                     k_graphs
                              )
                              )
    return connected_graphs


def all_motifs(n, format='list'):
    '''
   return all graphs of n nodes which are connected, that are unique up to isomorphism
   list them and the sum of how many are there
   '''
   sum = 0
```

```
    str_all_motifs = ''
    for k in range(n - 1, n ** 2 - n + 1):
        # Go over all the graphs of size k \in [n-1,n**2-n] (if k<n-1 the graph cannot be connected)
        if format == 'list':
            cur_str, cur_count = k_motifs_to_str(n, k)
        else:
            motifs = k_motifs(n, k)
            cur_str, cur_count = sum_k_motifs(motifs, verbose=verbose)
        str_all_motifs += cur_str
        sum += cur_count
    return str_all_motifs, sum


def sum_k_motifs(motifs, verbose=False):
    count = 0
    str_k_motifs = ''
    for motif in motifs:
        str_k_motifs += motif_to_str(motif, verbose)
        count += 1
    return str_k_motifs, count


def motif_to_str(motif, verbose=False):
    motif_str = f'#k={motif.number_of_edges()}\n'
    for u, v, d in motif.edges.data():
        motif_str += f'{u} {v}\n'
    if verbose:
        print(motif_str)
    return motif_str


def k_motifs_to_str(n, k):
    if n <= 1:
        return '', 0
    motifs = k_motifs(n, k)
    res = f'#{k}\n'
    count = 0
    for motif,c in motifs:
        edges = list(motif.edges())
        res += f'{edges} = {c}\n'
        count += 1
    if count==0:
      return '',0
    return res, count


def main_n(n, format='list'):
     '''
    give the result for the question for graph of size n
    '''
    res_str, count = all_motifs(n, format='list')
     res_str = f'n={n}\ncount={count}\n' + res_str
     return res_str
```

```
def save_motifs(n, path='.'):
    '''
    save a file with the details of graph of n nodes
    '''
    format = 'D-%d-%m-T-%H-%M-%S'
    stamp = time.strftime(format, time.localtime())
    file_name = f'M-{n}-{stamp}.txt'
    full_path = os.path.join(path, file_name)
    with open(full_path, 'w') as f:
        f.write(main_n(n))


def getNfromUser():
    N = input("Enter N: ")
    return N



def main():
    N = getNfromUser()
    # TODO:  read edges from file
    # create an empty directed graph
    global graph



    with open('edges.txt', 'r') as f:
        while True:
            # TODO: the format of the line in file is 'u v',
          edge = f.readline()

            # check if file is empty or end of file is reached
            if not edge:
                break

            # we want to obtain u and v as integers
            u, v = edge.split()
            u = int(u)
            v = int(v)

            # insert the edge to the graph
            graph.add_edge(u, v)
        # close the file
        f.close()
    # print(list(graph.edges))
    # call p1 code
    save_motifs(int(N))


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

הקוד בסעיף זה דומה מאוד לקוד של סעיף 1, הוא טוען רשימת קשתות בשם edges.txt מהתקייה הנוכחית של הפרוייקט ומריץ את אותן הפונקציות מחלק 1 רק עם שינויים קלים.

במקום לייצר את כל המוטיבים, אנחנו בעצם עוברים על כל תתי הקשתות בגרף שקיבלנו וסופרים את כל המוטיבים המופיעים בו עד כדי איזומורפיזם.

את התוצאה אנחנו שומרים בקובץ טקסט בעל אותו פורמט של הקובץ מחלק 1. צירפנו קובץ לדוגמה של קשתות וקובץ פלט.

קישור לגיטאב שלנו המכיל גם את קובץ ה-*ReadMe* וגם את הקוד עצמו :

https://github.com/almog-sharoni/BioComp_Ex2