Ex2

June 16, 2023

1 Biological Computation - Ex2

Names: Tal Rozencweig (21339169) and Ofir Yehezkel (214328940)

1.1 Part 1

1.1.1 a:

Write a program that gets asinput a positive integer n and generates all connected sub-graphs of size n.

[ ]:
```
import itertools
import time
```

[ ]:
```
PRINT_ALL = False
GRAPHS_TO_PRINT = 3
assert GRAPHS_TO_PRINT > 0 or PRINT_ALL, "GRAPHS_TO_PRINT needs to be positive␣
 ↪when PRINT_ALL = False"
```

[ ]:
```
def find_all_graphs_n(n: int) -> dict:
    """
    input: n -> number of vertices
    output:
    """
    total = {}
    for i in range(1, n*(n-1) + 1):
        g = itertools.combinations(itertools.permutations(range(n),2), i) # get␣
 ↪all possible graphs with i edges
        total[i] = list(g)
        # total += list(g)
    return total


def is_connected(graph, n):
    # Define a function that takes a graph as a list of tuples and returns True␣
 ↪if it is connected or False otherwise

    # Check if the size of the graph is zero or one
    if n == 0 or n == 1:
```

```python
        return True

    # Choose an arbitrary vertex as the starting point
    start = graph[0][0]

    # Create a queue to store the vertices to visit and a set to store the visited vertices
    queue = [start]
    visited = set()

    # Loop until the queue is empty
    while queue:
        # Dequeue a vertex from the queue and mark it as visited
        v = queue.pop(0)
        visited.add(v)

        # Loop through the edges of the graph and enqueue the adjacent vertices that are not visited
        for edge in graph:
            if v in edge:
                u = edge[0] if edge[1] == v else edge[1]
                if u not in visited:
                    queue.append(u)

    # Check if all the vertices are visited
    return len(visited) == n


def is_isomorphic(graph1: tuple, graph2: tuple, n: int) -> bool:
    if(len(graph1) != len(graph2)): # check size compatibility
        return False
    return graph1 in get_permutaions(graph2, n)


def get_permutaions(graph: tuple, n: int) -> list:
    total_permutations = []
    for permutaion in itertools.permutations(range(n), n): # iterate through all permutations
        new_graph = []
        for edge in graph: # permute graph per the given permutation
            new_edge = [-1,-1]
            for index, value in enumerate(edge): # permute edge
                new_edge[index] = permutaion[value]
            new_graph.append(tuple(new_edge))
        total_permutations.append(tuple(new_graph))
    return total_permutations
```

```python
def filter_graphs(graphs_dict: dict, n: int) -> list:
    total = []
    for graphs in graphs_dict.values():
        connected_graphs = list(filter(lambda graph: is_connected(graph,n),graphs))
        if(len(connected_graphs) == 1):
            total.append(frozenset(connected_graphs[0]))
        else:
            isomorphic_groups = []
            for graph in connected_graphs:
                isomorphic_group = {frozenset(permut) for permut in get_permutaions(graph,n)}
                if isomorphic_group not in isomorphic_groups:
                    isomorphic_groups.append(isomorphic_group)
            total += [g.pop() for g in isomorphic_groups if len(g) > 0]
    return total


def print_graphs(filtered_graphs: list, n: int):
    print(f'n={n}')
    print(f'count={len(filtered_graphs)}')
    for i,graph in enumerate(filtered_graphs):
        if not PRINT_ALL:
            if i == GRAPHS_TO_PRINT - 1:
                print(".\n.\n.")
                continue
            elif GRAPHS_TO_PRINT - 1 < i < len(filtered_graphs) - 1:
                continue
        print(f'#{i+1}')
        for edge in list(graph):
            print(f'{edge[0] + 1} {edge[1] + 1}')


def find_filter_print(n: int):
    starting_time = time.time()
    graphs = find_all_graphs_n(n)
    filtered_graphs = filter_graphs(graphs, n)
    print_graphs(filtered_graphs, n)
    print("Time to run:", (time.time() - starting_time), "seconds")

# # Get the input n from the user
# n = int(input("Enter a positive integer n: "))
# find_filter_print(n)
```

1.1.2 b:

Output the result of your program for n = 1 to 4.

[ ]:
```
n = 1
find_filter_print(n)
```

```
n=1
count=0
Time to run: 0.0 seconds
```

[ ]:
```
n = 2
find_filter_print(n)
```

```
n=2
count=2
#1
1 2
#2
1 2
2 1
Time to run: 0.0 seconds
```

[ ]:
```
n = 3
find_filter_print(n)
```

```
n=3
count=13
#1
1 2
1 3
#2
2 3
3 1
.
.
.
#13
1 2
2 3
3 2
3 1
1 3
2 1
Time to run: 0.0013840198516845703 seconds
```

[ ]:
```
n = 4
find_filter_print(n)
```

```
n=4
```

```
count=199
#1
4 2
4 3
4 1
#2
4 2
2 3
4 1
.
.
.
#199
1 2
2 3
3 2
4 2
1 4
3 1
4 1
3 4
1 3
2 1
4 3
2 4
Time to run: 0.5342109203338623 seconds
```

1.1.3 c:

`What is the maximal number n for which the program can complete successfully within no more than 1 hour of computing time?`

We can see that the program completed the computation of n = 1 to 4 in less than 1 second.

Let's check the time for the program to compute the output of n = 5:

[ ]:
```
n = 5
find_filter_print(n)
```

```
n=5
count=9364
#1
1 2
1 3
1 4
1 5
#2
2 1
5 2
```

```
5 3
5 4
.
.
.
#9364
5 1
4 5
5 4
4 2
1 3
2 1
2 4
5 3
4 1
1 2
3 5
2 3
1 5
3 2
4 3
5 2
1 4
3 1
2 5
3 4
Time to run: 1325.531239748001 seconds
```

It took about 22 minutes

Now let's check the time for the program to compute the output of n = 6:

[ ]:
```
n = 6
find_filter_print(n)
```

```
---------------------------------------------------------------------------
MemoryError                               Traceback (most recent call last)
c:\Users\   \OneDrive - Bar-Ilan University\Desktop\     \       \    \  \    .
↪HW2\motifs\ .ipynb Cell 19 in <cell line: 2>()
      <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=0'>1</a> n = 6
----> <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=1'>2</a>␣
↪find_filter_print(n)
```

```
c:\Users\   \OneDrive - Bar-Ilan University\Desktop\     \      \    \  \    .
↪HW2\motifs\ .ipynb Cell 19 in find_filter_print(n)
    <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=102'>103</a> def
↪find_filter_print(n: int):
    <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=103'>104</a>
↪starting_time = time.time()
--> <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=104'>105</a>      graphs =
↪find_all_graphs_n(n)
    <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=105'>106</a>
↪filtered_graphs = filter_graphs(graphs, n)
    <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=106'>107</a>
↪print_graphs(filtered_graphs, n)

c:\Users\   \OneDrive - Bar-Ilan University\Desktop\     \      \    \  \    .
↪HW2\motifs\ .ipynb Cell 19 in find_all_graphs_n(n)
      <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=6'>7</a> for i in
↪range(1, n*(n-1) + 1):
      <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=7'>8</a>      g =
↪itertools.combinations(itertools.permutations(range(n),2), i) # get all
↪possible graphs with i edges
```

```
----> <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=8'>9</a>     total[i] =
↪list(g)
     <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=9'>10</a>     # total +=
↪list(g)
     <a href='vscode-notebook-cell:/c%3A/Users/%D7%90%D7%95%D7%A4%D7%99%D7%A8/
↪OneDrive%20-%20Bar-Ilan%20University/Desktop/
↪%D7%97%D7%95%D7%9E%D7%A8%D7%99%20%D7%9C%D7%99%D7%9E%D7%95%D7%93/
↪%D7%A9%D7%A0%D7%94%20%D7%93/%D7%A1%D7%9E%D7%A1%D7%98%D7%A8%20%D7%91/
↪%D7%97%D7%99%D7%A9%D7%95%D7%91%20%D7%91%D7%99%D7%95%D7%9C%D7%95%D7%92%D7%99/
↪%D7%A9.%D7%91/HW2/motifs.ipynb#X24sZmlsZQ%3D%3D?line=10'>11</a> return total

MemoryError:
```

We can see that it takes more than 8 hours.

So, in 1 hour the maximal value of n that the program can compute is 5.

1.1.4 d:

`What is the maximal number n for which the program can complete successfully within 2,4,8 hours of computing time?`

As we saw in section c, he maximal value of n that the program can compute within 2,4,8 hours is 5.

1.2 Part 2

```
Write a program that gets as input positive integer n and a graph of theformat:
1 2
2 3
1 4
The graph in the example contains 4 vertices 1,2,3,4 and directed edges (1,2)
(2,3) (1,4). The program should output all sub-graphs of size n and count how
many instances appear of each motif. The format of the outputof the identified
sub-graphs should be like in question 1, where in the line after #k should appear
the count of number of instances, count=m if the motif appears m times. Output
count=0 if a motif does not appear in the graph.
```

[ ]:
```
def get_graph() -> tuple:
    n = 0
    while n < 1:
        n = int(input("n = "))
```

```
    graph = []
    input_ = [int(x) - 1 for x in input("Enter an edge: ").split()]
    while input_:
        if len(input_) > 1:
            edge = (input_[0], input_[1])
            if edge not in graph:
                graph.append(edge)
        input_ = [int(x) - 1 for x in input("Enter an edge: ").split()]

    return frozenset(graph), n


def count_motifs(input_graph: frozenset, motifs: list, n: int) -> list:
    total = []

    for motif in motifs:
        count = 0
        for permutation in get_permutaions(tuple(motif), n):
            if frozenset(permutation).issubset(input_graph):
                count += 1
            if input_graph == frozenset(permutation):
                count = 1
                break
        total.append((motif,count))

    return total


def print_graphs_2(filtered_graphs: list, input_graph: frozenset, n: int):
    print(f'graph: ')
    for edge in list(input_graph):
            print(f'{edge[0] + 1} {edge[1] + 1}')

    print(f'\n{n=}')
    print(f'count={len(filtered_graphs)}')
    print('_________\n')

    filtered_graphs.sort(key=lambda x: x[1],reverse=True)
    for i,(graph,count) in enumerate(filtered_graphs):
        print(f'#{i+1}')
        print(f'{count=}')
        for edge in list(graph):
            print(f'{edge[0] + 1} {edge[1] + 1}')
        print()
```

[ ]:
```
graph, n = get_graph()
total = count_motifs(graph, filter_graphs(find_all_graphs_n(n), n) ,n)
```

```
print_graphs_2(total, graph, n)
```

```
graph:
1 2
2 1

n=2
count=2
---------

#1
count=2
1 2

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