COT 5405: ANALYSIS OF ALGORITHM

ASSIGNMENT 1

Introduction –
The language that we have used to build our assignment is C++. Since we are given that we can use basic STL data structures, we have used Vector, Lists, Structures, Priority Queue and Arrays. We will discuss in details which data structure is used for what concept.

The project consists of 5 files:

- graph_operations.h
- graph_simulator.h
- simulated_test.cpp
- graph_make.h
- real_test.cpp

Now let us describe the contents of each file in detail.

1. **graph_operations.h –**
   It is a header file that is later included in simulated_test.cpp. It contains a user defined class **'graph'.**
   It contains the following **private members** -
   - int countNodes – the total number of vertices
   - vector <int> adjacencyList[maxNodes] – it represents the graph in the form an adjacency list.
   - bool isNodeVisited[maxNodes] – returns true/false based upon if a node has been visited or not. Is used for the function shortest_path.
   - vector <int> cc – this is an integer vector that stores the list of connected components.
   - int parent[maxNodes] – an array to store parent nodes while performing depth first search operation.
   - int end1 -
   - int end2 -

   and the following **public members** -
   - *Graph(int countVertices)* – a parametrized constructor that initializes int countNodes with the passed parameter.
   - *void add_edge(int u, int v)* – a function that adds edge u-v to the graph.

- *void dfs(int source)* – a function that traverse the graph in a depth-first-search manner from source to the remaining vertices.
- *void isCyclic(int s, int root)* – this is a function that traverses from *root 'r'* and *source 's'* and determines the presence of any cycle.
- *vector < vector<int> > connected_components()* – this is a function that returns a list of all the connected components in the graph. The connected components in itself are a list of nodes. Therefore the return type of the function is *'vector < vector <int> >'.*
- *vector <int> one_cycle()* – this is a function that returns a list of all the nodes that are present in a cycle in the graph. If the graph has no cycles, the function will return an empty list.
- *map<int, vector<int> > shortest_paths(int s)* – this is a function that returns a list of nodes that are present in the shortest path from the vertices to the source node *'s'.* The function returns a map because the key is the node and the value is the list of nodes that lies in the shortest path from the source *'s'* to the node.

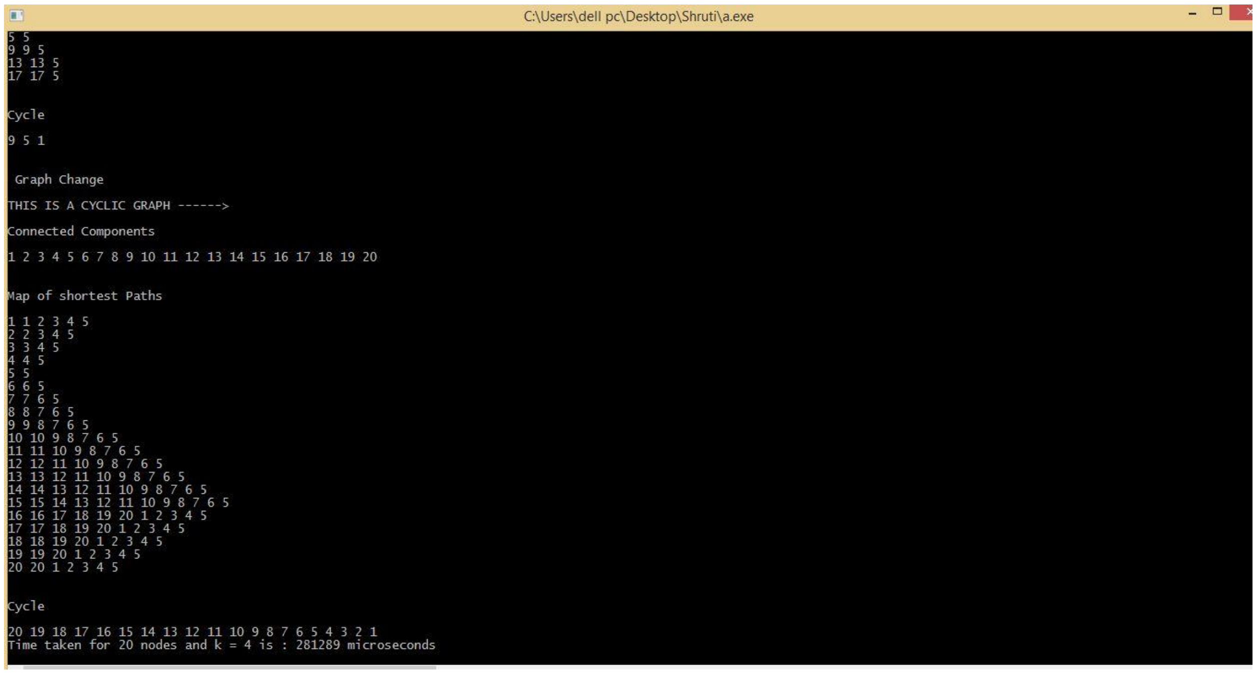
2. **graph_simulator.h –**

   It is a header file that contains functions for generating graphs using the suitable rules of edge creation. We used file handling to create files for each graph and write the vertices and edges into that file. We have created four functions to generate graphs. All of them takes

   - *void generate_complete_graph_on_n_vertices(int n)* - It takes number of nodes to be created as an argument. It opens a file *'clique.txt'* in write mode. It then generates a graph based upon the rule for the complete graph (i.e. all the vertices are connected to each other). As the vertices and edges are created, they are simultaneously stored in the text file.
   - *void generate_cyclic_graph_on_n_vertices(int n)* – Like above, it taken number of nodes as an argument *'n'* and generate a cyclic graph. It opens a file named "*cyclic.txt*" in write mode. In a cyclic graph there exist an edge between two vertices *u* and *v,* if |u-v| = 1 or |u-v| = n-1. Therefore, an edge exists between two consecutive vertices. The vertices and edges are created and stored into the text file.
   - *void generate_empty_graph_on_n_vertices(int n)* – It takes number of nodes as an argument *'n'* and generate an empty graph. It opens a file named "*empty.txt*" in write mode. In an empty graph there exist no edge between any two vertices *u* and *v.* Therefore, no edge is stored into the text file, only the vertices are created.
   - *void generate_equivalence_mod_k_on_n_vertices(int n, int k)* – It takes two arguments : number of node *'n'* and mod value *'k'*. Varying the value of k results in a different graph.

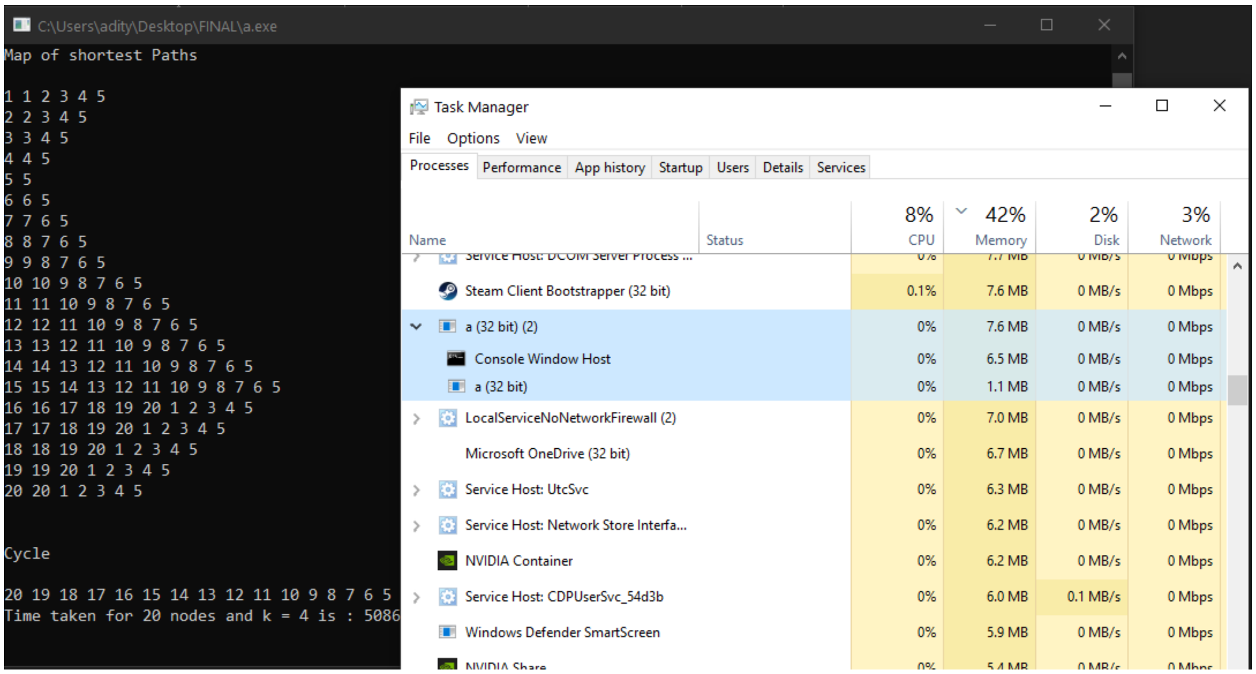
3. **simulated_test.cpp –**
   It contains the main function that will give input to the graph generating functions in *graph_simulator.h*. Once the graphs have been generated and stored into the respective text files, the graph algorithms are implemented on these graphs from the file *graph_operations.h*. The value on '*n'* and *'k'* can be varied up and down so as to compare the execution-time and memory-allocation for each call with different value of *'n'*.
   So, we varied the value of *'n'* and *'k'* as follows and obtained various snapshots of the difference in run-time and allocated memory.

- For n = 20 & k = 4 :



C:\Users\dell pc\Desktop\Shruti\a.exe

```
5 5
9 9 5
13 13 5
17 17 5


Cycle

9 5 1


 Graph Change

THIS IS A CYCLIC GRAPH ------>

Connected Components

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Map of shortest Paths

1 1 2 3 4 5
2 2 3 4 5
3 3 4 5
4 4 5
5 5
6 6 5
7 7 6 5
8 8 7 6 5
9 9 8 7 6 5
10 10 9 8 7 6 5
11 11 10 9 8 7 6 5
12 12 11 10 9 8 7 6 5
13 13 12 11 10 9 8 7 6 5
14 14 13 12 11 10 9 8 7 6 5
15 15 14 13 12 11 10 9 8 7 6 5
16 16 17 18 19 20 1 2 3 4 5
17 17 18 19 20 1 2 3 4 5
18 18 19 20 1 2 3 4 5
19 19 20 1 2 3 4 5
20 20 1 2 3 4 5


Cycle

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Time taken for 20 nodes and k = 4 is : 281289 microseconds
```



C:\Users\adity\Desktop\FINAL\a.exe

```
Map of shortest Paths

1 1 2 3 4 5
2 2 3 4 5
3 3 4 5
4 4 5
5 5
6 6 5
7 7 6 5
8 8 7 6 5
9 9 8 7 6 5
10 10 9 8 7 6 5
11 11 10 9 8 7 6 5
12 12 11 10 9 8 7 6 5
13 13 12 11 10 9 8 7 6 5
14 14 13 12 11 10 9 8 7 6 5
15 15 14 13 12 11 10 9 8 7 6 5
16 16 17 18 19 20 1 2 3 4 5
17 17 18 19 20 1 2 3 4 5
18 18 19 20 1 2 3 4 5
19 19 20 1 2 3 4 5
20 20 1 2 3 4 5


Cycle

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
Time taken for 20 nodes and k = 4 is : 5086
```

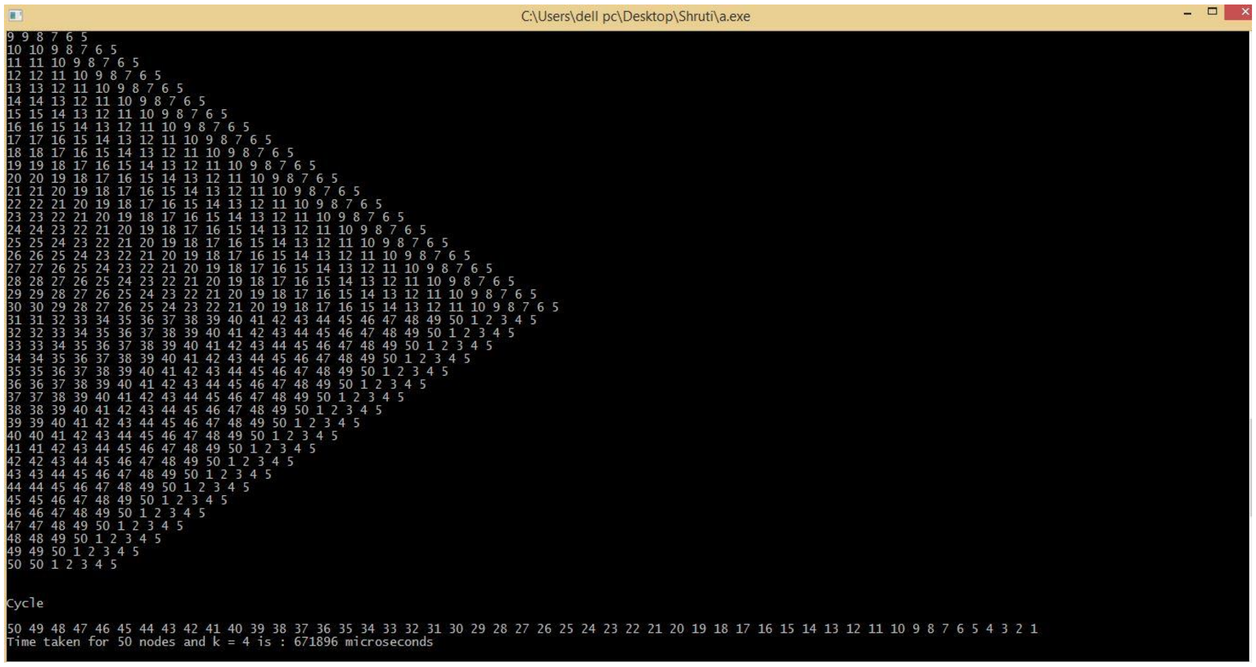
Task Manager

File Options View
Processes Performance App history Startup Users Details Services

Name	Status	8% CPU	42% Memory	2% Disk	3% Network
Service Host: DCOM Server Process ...		0%	7.7 MB	0 MB/s	0 Mbps
Steam Client Bootstrapper (32 bit)		0.1%	7.6 MB	0 MB/s	0 Mbps
a (32 bit) (2)		0%	7.6 MB	0 MB/s	0 Mbps
Console Window Host		0%	6.5 MB	0 MB/s	0 Mbps
a (32 bit)		0%	1.1 MB	0 MB/s	0 Mbps
LocalServiceNoNetworkFirewall (2)		0%	7.0 MB	0 MB/s	0 Mbps
Microsoft OneDrive (32 bit)		0%	6.7 MB	0 MB/s	0 Mbps
Service Host: UtcSvc		0%	6.3 MB	0 MB/s	0 Mbps
Service Host: Network Store Interfa...		0%	6.2 MB	0 MB/s	0 Mbps
NVIDIA Container		0%	6.2 MB	0 MB/s	0 Mbps
Service Host: CDPUserSvc_54d3b		0%	6.0 MB	0.1 MB/s	0 Mbps
Windows Defender SmartScreen		0%	5.9 MB	0 MB/s	0 Mbps
NVIDIA Share		0%	5.4 MB	0 MB/s	0 Mbps

The above snapshot shows the memory allocation for the executable file.

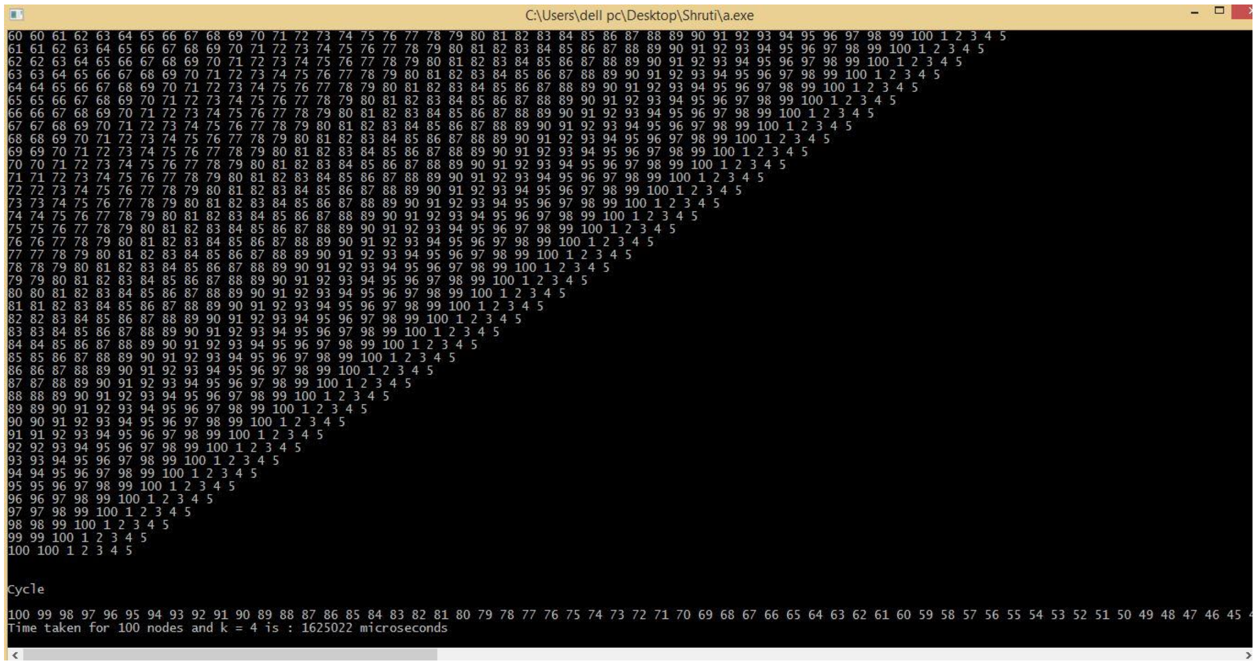
- For n = 50 and k = 4 :



C:\Users\dell pc\Desktop\Shruti\a.exe

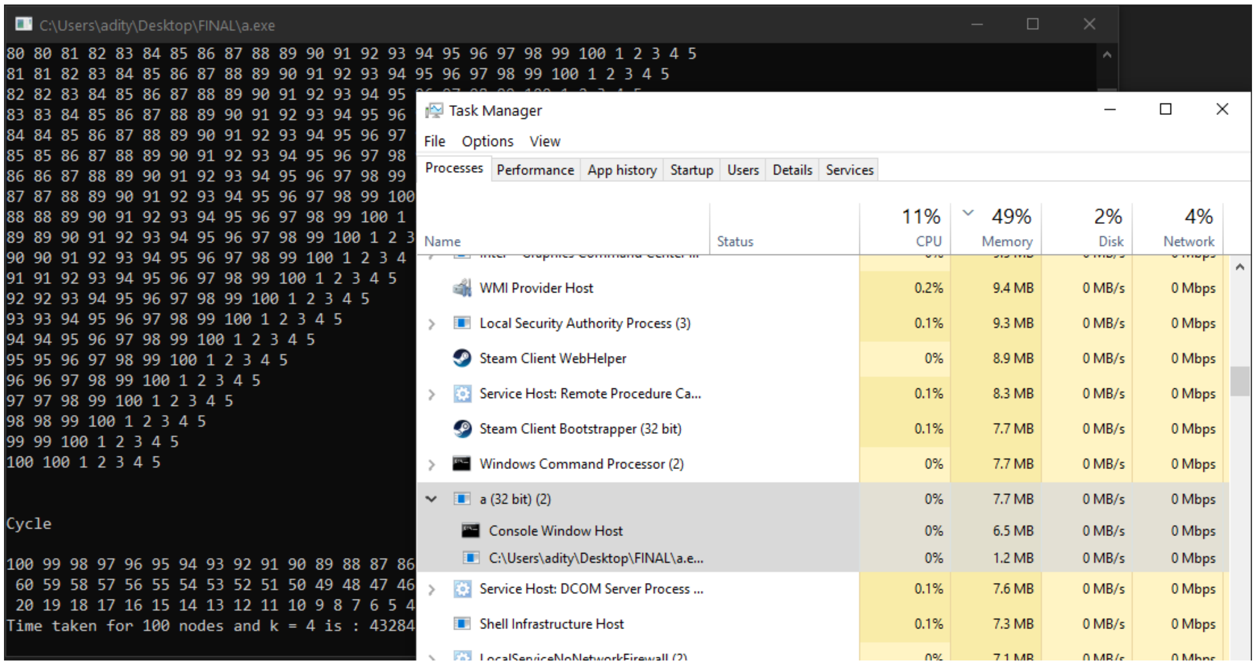
```
9 9 8 7 6 5
10 10 9 8 7 6 5
11 11 10 9 8 7 6 5
12 12 11 10 9 8 7 6 5
13 13 12 11 10 9 8 7 6 5
14 14 13 12 11 10 9 8 7 6 5
15 15 14 13 12 11 10 9 8 7 6 5
16 16 15 14 13 12 11 10 9 8 7 6 5
17 17 16 15 14 13 12 11 10 9 8 7 6 5
18 18 17 16 15 14 13 12 11 10 9 8 7 6 5
19 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
20 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
21 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
22 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
23 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
24 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
25 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
26 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
27 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
28 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
29 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
30 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5
31 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
32 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
33 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
34 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
35 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
36 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
37 37 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
38 38 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
39 39 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
40 40 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
41 41 42 43 44 45 46 47 48 49 50 1 2 3 4 5
42 42 43 44 45 46 47 48 49 50 1 2 3 4 5
43 43 44 45 46 47 48 49 50 1 2 3 4 5
44 44 45 46 47 48 49 50 1 2 3 4 5
45 45 46 47 48 49 50 1 2 3 4 5
46 46 47 48 49 50 1 2 3 4 5
47 47 48 49 50 1 2 3 4 5
48 48 49 50 1 2 3 4 5
49 49 50 1 2 3 4 5
50 50 1 2 3 4 5


Cycle

50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Time taken for 50 nodes and k = 4 is : 671896 microseconds
```

- For n = 100 & k = 4 :



C:\Users\dell pc\Desktop\Shruti\a.exe

```
60 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
61 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
62 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
63 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
64 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
65 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
66 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
67 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
68 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
69 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
70 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
71 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
72 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
73 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
74 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
75 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
76 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
77 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
78 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
79 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
80 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
81 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
82 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
83 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
84 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
85 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
86 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
87 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
88 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
89 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
90 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
91 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
92 92 93 94 95 96 97 98 99 100 1 2 3 4 5
93 93 94 95 96 97 98 99 100 1 2 3 4 5
94 94 95 96 97 98 99 100 1 2 3 4 5
95 95 96 97 98 99 100 1 2 3 4 5
96 96 97 98 99 100 1 2 3 4 5
97 97 98 99 100 1 2 3 4 5
98 98 99 100 1 2 3 4 5
99 99 100 1 2 3 4 5
100 100 1 2 3 4 5


Cycle

100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45
Time taken for 100 nodes and k = 4 is : 1625022 microseconds
```



C:\Users\adity\Desktop\FINAL\a.exe

```
80 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
81 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
82 82 83 84 85 86 87 88 89 90 91 92 93 94 95
83 83 84 85 86 87 88 89 90 91 92 93 94 95 96
84 84 85 86 87 88 89 90 91 92 93 94 95 96 97
85 85 86 87 88 89 90 91 92 93 94 95 96 97 98
86 86 87 88 89 90 91 92 93 94 95 96 97 98 99
87 87 88 89 90 91 92 93 94 95 96 97 98 99 100
88 88 89 90 91 92 93 94 95 96 97 98 99 100 1
89 89 90 91 92 93 94 95 96 97 98 99 100 1 2 3
90 90 91 92 93 94 95 96 97 98 99 100 1 2 3 4
91 91 92 93 94 95 96 97 98 99 100 1 2 3 4 5
92 92 93 94 95 96 97 98 99 100 1 2 3 4 5
93 93 94 95 96 97 98 99 100 1 2 3 4 5
94 94 95 96 97 98 99 100 1 2 3 4 5
95 95 96 97 98 99 100 1 2 3 4 5
96 96 97 98 99 100 1 2 3 4 5
97 97 98 99 100 1 2 3 4 5
98 98 99 100 1 2 3 4 5
99 99 100 1 2 3 4 5
100 100 1 2 3 4 5


Cycle

100 99 98 97 96 95 94 93 92 91 90 89 88 87 86
 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46
 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4
Time taken for 100 nodes and k = 4 is : 43284
```

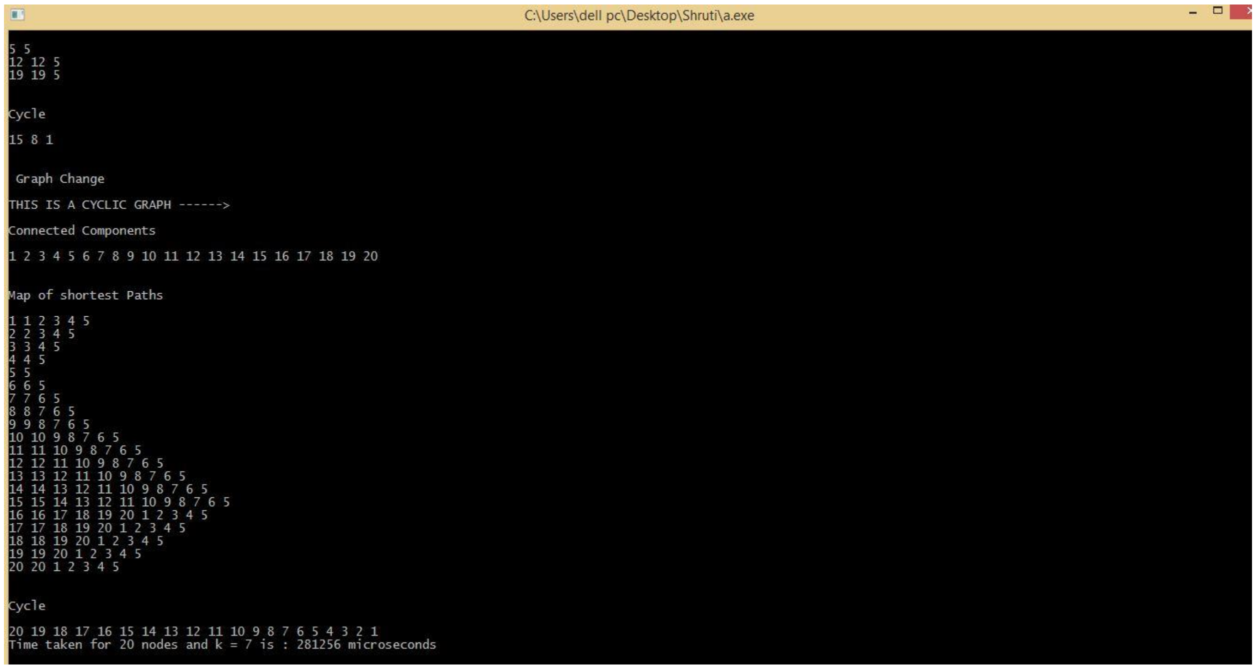
Task Manager

File Options View

Processes | Performance | App history | Startup | Users | Details | Services

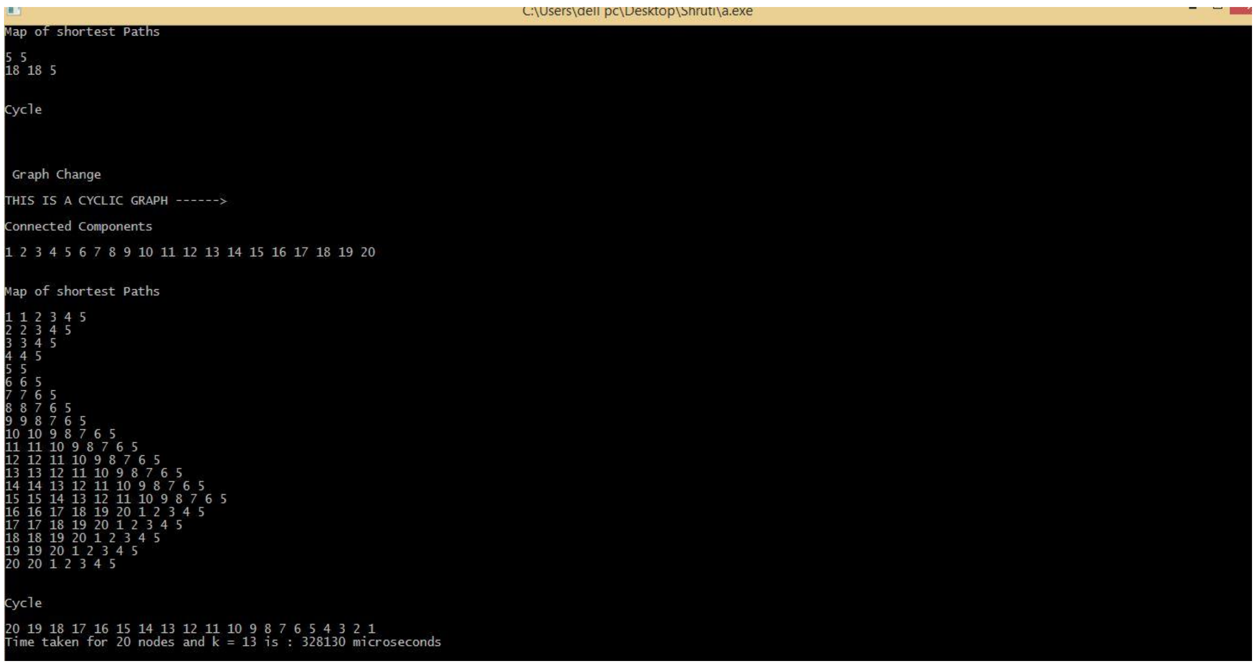
Name	Status	11% CPU	49% Memory	2% Disk	4% Network
WMI Provider Host		0.2%	9.4 MB	0 MB/s	0 Mbps
Local Security Authority Process (3)		0.1%	9.3 MB	0 MB/s	0 Mbps
Steam Client WebHelper		0%	8.9 MB	0 MB/s	0 Mbps
Service Host: Remote Procedure Ca...		0.1%	8.3 MB	0 MB/s	0 Mbps
Steam Client Bootstrapper (32 bit)		0.1%	7.7 MB	0 MB/s	0 Mbps
Windows Command Processor (2)		0%	7.7 MB	0 MB/s	0 Mbps
a (32 bit) (2)		0%	7.7 MB	0 MB/s	0 Mbps
Console Window Host		0%	6.5 MB	0 MB/s	0 Mbps
C:\Users\adity\Desktop\FINAL\a.e...		0%	1.2 MB	0 MB/s	0 Mbps
Service Host: DCOM Server Process ...		0.1%	7.6 MB	0 MB/s	0 Mbps
Shell Infrastructure Host		0.1%	7.3 MB	0 MB/s	0 Mbps

- For n = 20 & k = 7 :


C:\Users\dell pc\Desktop\Shruti\a.exe

```
5 5
12 12 5
19 19 5


Cycle

15 8 1


 Graph Change

THIS IS A CYCLIC GRAPH ------>

Connected Components

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Map of shortest Paths

1 1 2 3 4 5
2 2 3 4 5
3 3 4 5
4 4 5
5 5
6 6 5
7 7 6 5
8 8 7 6 5
9 9 8 7 6 5
10 10 9 8 7 6 5
11 11 10 9 8 7 6 5
12 12 11 10 9 8 7 6 5
13 13 12 11 10 9 8 7 6 5
14 14 13 12 11 10 9 8 7 6 5
15 15 14 13 12 11 10 9 8 7 6 5
16 16 17 18 19 20 1 2 3 4 5
17 17 18 19 20 1 2 3 4 5
18 18 19 20 1 2 3 4 5
19 19 20 1 2 3 4 5
20 20 1 2 3 4 5


Cycle

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Time taken for 20 nodes and k = 7 is : 281256 microseconds
```

- For n = 20 & k = 13 :


C:\Users\dell pc\Desktop\Shruti\a.exe

```
Map of shortest Paths

5 5
18 18 5


Cycle



 Graph Change

THIS IS A CYCLIC GRAPH ------>

Connected Components

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20


Map of shortest Paths

1 1 2 3 4 5
2 2 3 4 5
3 3 4 5
4 4 5
5 5
6 6 5
7 7 6 5
8 8 7 6 5
9 9 8 7 6 5
10 10 9 8 7 6 5
11 11 10 9 8 7 6 5
12 12 11 10 9 8 7 6 5
13 13 12 11 10 9 8 7 6 5
14 14 13 12 11 10 9 8 7 6 5
15 15 14 13 12 11 10 9 8 7 6 5
16 16 17 18 19 20 1 2 3 4 5
17 17 18 19 20 1 2 3 4 5
18 18 19 20 1 2 3 4 5
19 19 20 1 2 3 4 5
20 20 1 2 3 4 5


Cycle

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Time taken for 20 nodes and k = 13 is : 328130 microseconds
```

4. **graph_make.h –**

   It contains a structure called *'movie_rating'* and it stores the information of every rating the movie has received. It contains three attributes -
   1. *int customerID* - this is the ID of the customer that has reviewed the movie.
   2. *char rating* - the rating given by the viewer
   3. *string date* - the date on which the movie was reviewed.

   Now we take an array of size 4 of ifstream objects so that we can read the Netflix rating data from the 4 txt files. We open the 4 txt files into every ifstream object and start reading the data from them.

   It contains '*void read()'* function that reads the data from the four text files that contains the Netflix movie rating data.
   We read line by line and file by file. If at the end of the line we spot a ':' the movie id is updated and otherwise the customerID, rating and date are read from the file and updated into the structure. After every loop, the structure is stored into the above defined vector along with the movie ID, signifying that a viewer has added a rating for the movie.

   This above process is repeated for all the txt files containing Netflix movie rating data.

   This header file is included in the file real_test.cpp and after all the text files are read and stored we will be able to apply some rules and make a graph from the data.

5. **real_test.cpp –**
   This is the file that contains the main() function and also includes the graph_make.h and graph_operations.h header files.

   We call the *read() function* that reads all the data. Because the data was going too long, so for my convenience, I added the limit of number of movies (int CountMovies) to 10 . A loop traverses from 1 to number of movies and stores the IDs of all the customers to a set of integers called S. We then make a graph g1 that has CountMovies number of nodes.

   Now, the criteria that we followed to make a connection between two customers is that – two customers are connected if they have watched at least two movies.

   Therefore in the for loop that traverses through all the customers, if the customer id exists in the *map mp<int, int>,* then it means that the person have watched this movie and this should be true with the second customer too. If both are true then an edge is added in g1 between the two customers else, the loop moves on.

   Once the graph g1 is complete, the functions from header file *graph_operations.h* are executed.