

Aoa Assignment-1 RESULTS

Description of all files:

(a) Graph_operations.java:

This java file contains three functions:

All of the three functions take two inputs as a parameter:

(i) **nodes_count:** It is the total number of nodes present in the graph.

(ii) **adjList:** It is the list of adjacencies(edges) present in the graph. It is represented by an array of sets of all edges. We have used a set-array, so that all adjacencies that are added are unique. When traversing a large data, there are chances when the same edge is added in the adjacency list twice or even more than that. Usage of hashset prevents that problem.

1. connected_components(int nodes_count, HashSet<Integer>[] adjList):

This is a function to count the number of connected components with the size of each component for a given graph. This function is based on a depth-first-search (dfs) algorithm.

2. one_cycle(int nodes_count, HashSet<Integer>[] adjList):

This is a function to detect cycles in a given graph. It returns a list of one cycle if there exists one. This also uses a dfs algorithm.

3. shortest_paths():

This function takes a source node and returns all the shortest paths (if reachable) from the source node to all other present nodes in the graph. This is based on dijkstra's algorithm.

(b) Graph_simulator.java:

It contains 6 different functions to simulate or create a graph based on a given criteria by a user. The user chooses an option and enters the number of nodes and the whole graph with adjacency list is created by these functions.

The user gets 6 different options to simulate the graph from, all of which are mentioned below:

1. N-cycle
2. Complete-Graph
3. Empty-Graph
4. Heap
5. Truncated Heap
6. Equivalence mod-k

(c) Simulated_test.java:

This java file contains the main function to run the above 2 functions: Graph_operations.java and Graph_simulator.java .

This can be run manually by providing number of nodes and all adjacencies as contains the full User-Interface (UI) through which a user can create a graph as well as can call the simulator java file through which the user can create a graph (from a set of 6 options) automatically by only providing the number of nodes.

This file also provides a way to run all 3 functions from graph_operations.java file to test on the graph given by the user.

Data Structure Used For Graph Representation: HashSet<Integer>[] adjList

The graph representation is done by an array of sets of integers. Every index of this array represents a node and the set present at this index represents all the edges between that node and all other nodes of the graph. At every index of this array-adjList, we add a set for storing all the unique nodes which are connected to the node at the specified index.

```
architkhatri@Archits-MacBook-Air AOA_completeFinal % /usr/bin/env /Library/Application Support/Code/User/workspaceStorage/ec7bf
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
1
Enter number of nodes for N-Cycle
6
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
█
```

Simulated Graph Experiments:

All runtime and peak memory analysis is done for all different kinds of graph after running all the 3 operations on it (Connected_components, one_cycle, Shortest_Path). To create a comparison we have done the experiment on all graphs for one small size and another relatively large size.

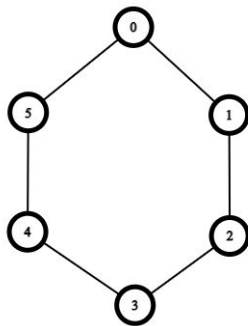
1. n-cycle graph:

Result: One total component.

One Cycle found

All shortest paths have length less than at most $n/2$.

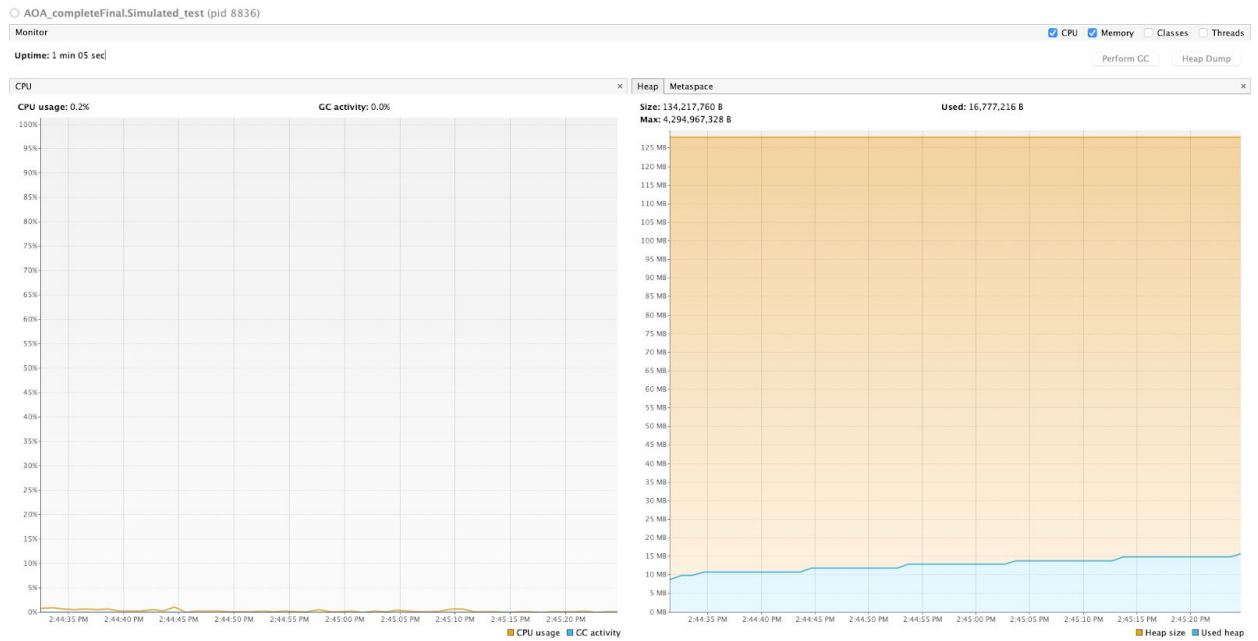
Input 1: No. of nodes (n) = 6



Output 1:

```
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
1
[0, 1, 2, 3, 4, 5]Size of this component set is: 6
No. of components present: 1
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
2
Cycle Detected between following nodes:-
0 1 2 3 4 5
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
3
Enter starting node:
3
Vertex          Distance      Path
3 -> 0           3            3 2 1 0
3 -> 1           2            3 2 1
3 -> 2           1            3 2
3 -> 4           1            3 4
3 -> 5           2            3 4 5
```

Run Time and Memory for Input 1:

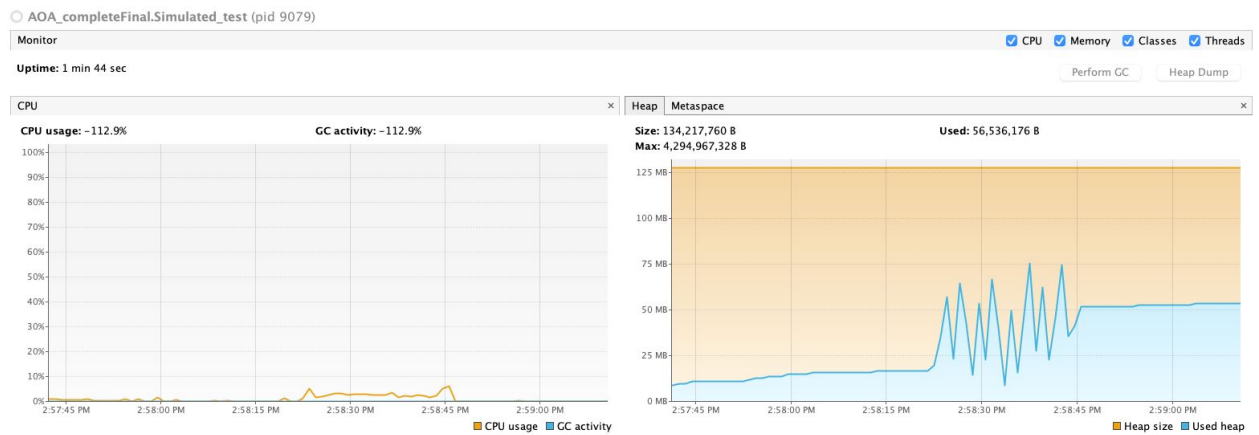


Input 2: No. of nodes (n) =4500

Output 2:

```
architkhatri@Archits-MacBook-Air AOA_comple
rchitkhatri/Library/Application Support/Code
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
1
Enter number of nodes for N-Cycle
4500
```

Run Time and Memory for Input 2:



Analysis:

We can easily observe that both CPU usage and memory have increased as the data size is increased. When the data size is increased we can see spikes of memory heap usage which refers to the memory consumption when all three graph operations were executed. In the case of small input these spikes were not easily seen as the data was very small.

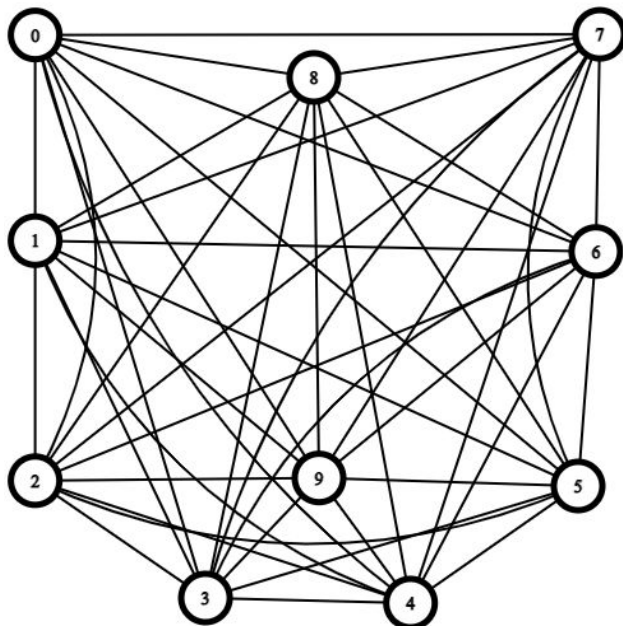
2. Complete Graph:

Result: one connected component.

Many Cycles found.

All shortest paths have unit length (length=1) .

Input 1: No. of nodes (n) = 10



Output 1:

```
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
2
Enter number of nodes for Complete Graph
10
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]Size of this component set is: 10
No. of components present: 1
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
2
Cycle Detected between following nodes:-
0 1 2
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
3
Enter starting node:
3


| Vertex | Distance | Path |
|--------|----------|------|
| 3 -> 0 | 1        | 3 0  |
| 3 -> 1 | 1        | 3 1  |
| 3 -> 2 | 1        | 3 2  |
| 3 -> 4 | 1        | 3 4  |
| 3 -> 5 | 1        | 3 5  |
| 3 -> 6 | 1        | 3 6  |
| 3 -> 7 | 1        | 3 7  |
| 3 -> 8 | 1        | 3 8  |
| 3 -> 9 | 1        | 3 9  |


```

Run Time and Memory for Input 1:

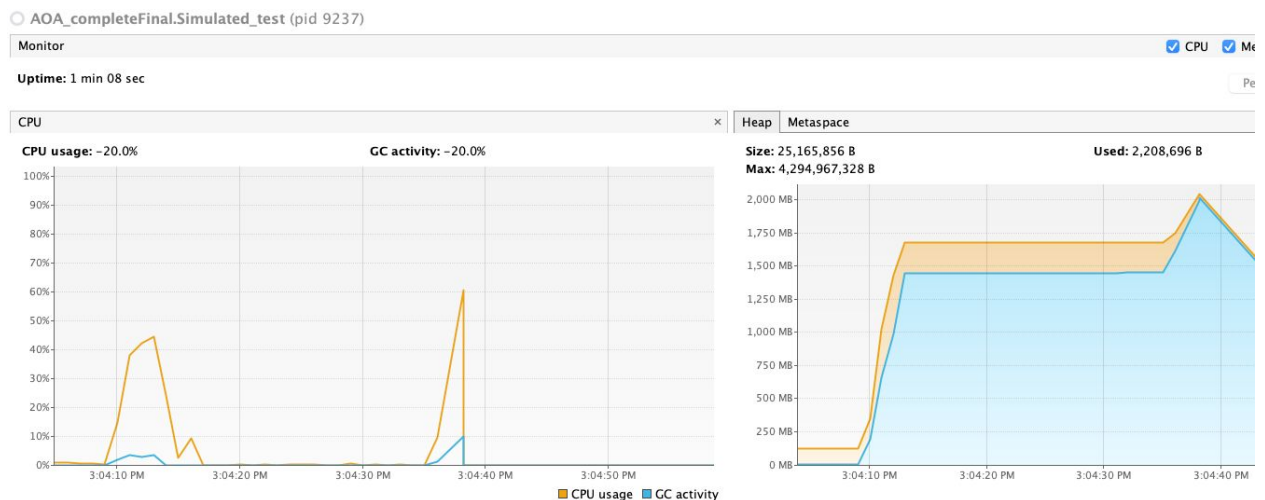


Input 2: No. of nodes (n) = 5000

Output 2:

```
Enter number of nodes for Complete Graph
5000
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
```

Run Time and Memory for Input 2:



Analysis:

We can easily observe that both time and memory have increased as the data size is increased. We can notice a large bump in memory usage when the graph was created for 5000 nodes. Another spike was seen in the second case when graph operations were called.

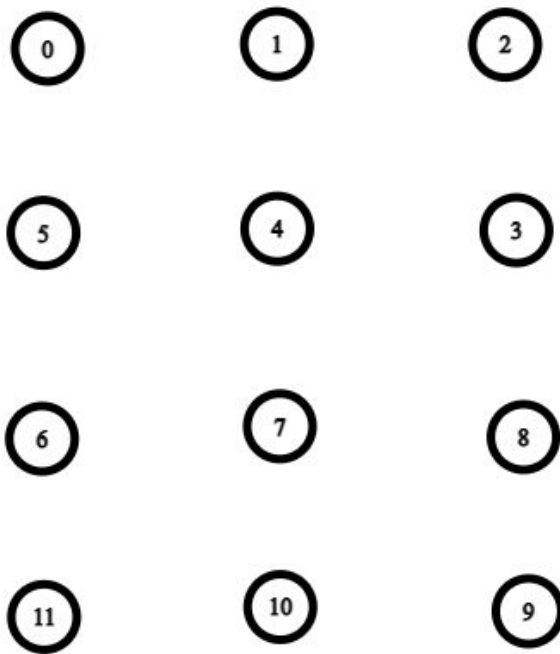
3. Empty Graph:

Result: zero connected component.

No Cycles found.

No path exists.

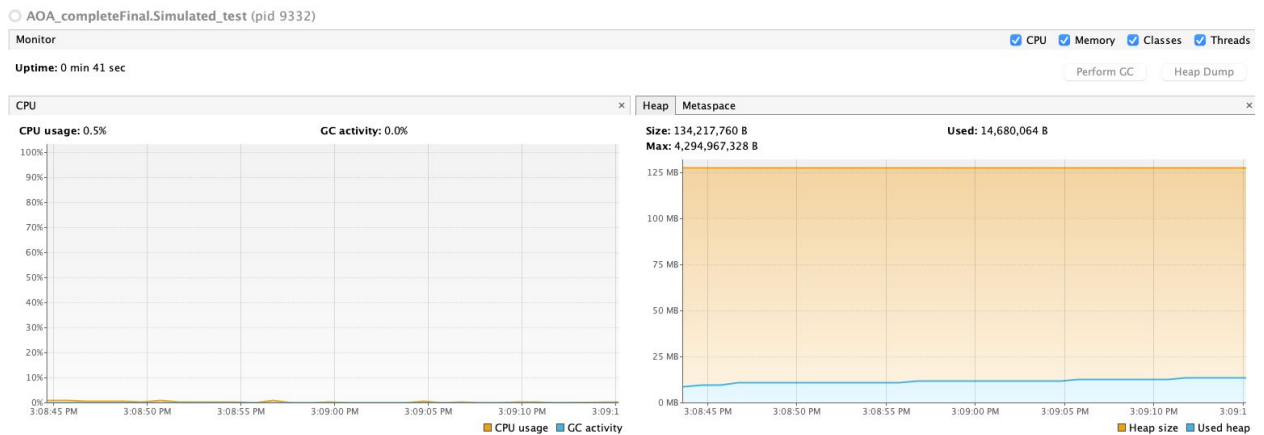
Input 1: No. of nodes (n) = 12



Output 1:

```
architkhatri@Archits-MacBook-Air AOA_con  
rchitkhatri/Library/Application Support/  
test  
Enter number from given choices:  
1: Create Graph using simulator  
2: Create a Graph manually  
Enter any other key to exit  
1  
Enter the operation you want to run:  
1: n-cycle graph  
2: complete graph  
3: empty graph  
4: heap graph  
5: truncated heap graph  
6: equivalence mod k graph  
Enter any key to go to previous menu  
3  
Enter number of nodes for Empty Graph  
12
```


Run Time and Memory for Input 1:

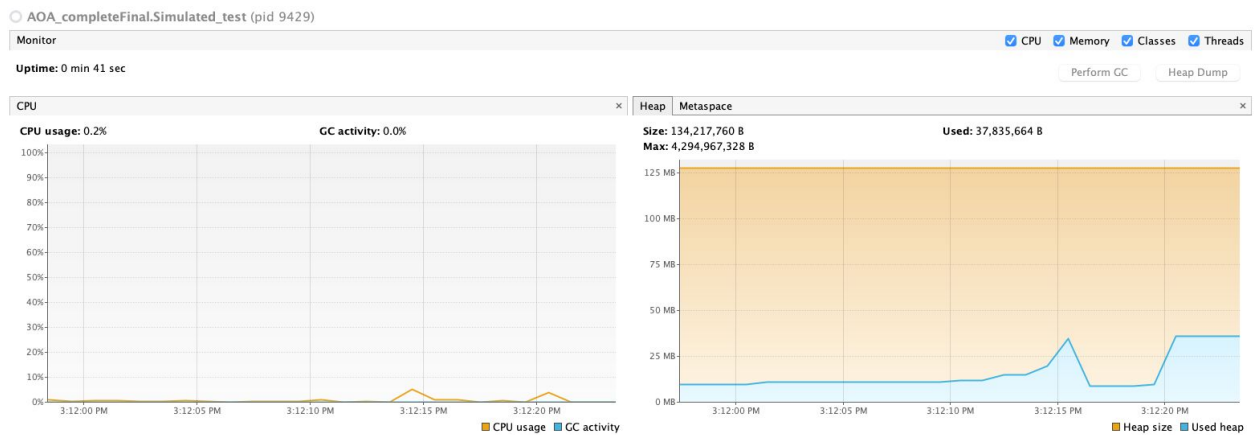


Input 2: No. of nodes (n) = 50000

Output 2:

```
architkhatri@Archits-MacBook-Air AOA_comple
architkhatri/Library/Application Support/Cod
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
3
Enter number of nodes for Empty Graph
50000
```

Run Time and Memory for Input 2:



Analysis:

As this is the case for an empty graph, no adjacencies list were created for both small and large inputs saving much space and we can see the same from the similarity between these two memory graphs. Runtime of large input was slightly higher than that for small input. The only increase in memory use was in the case of large input when graph operations were called.

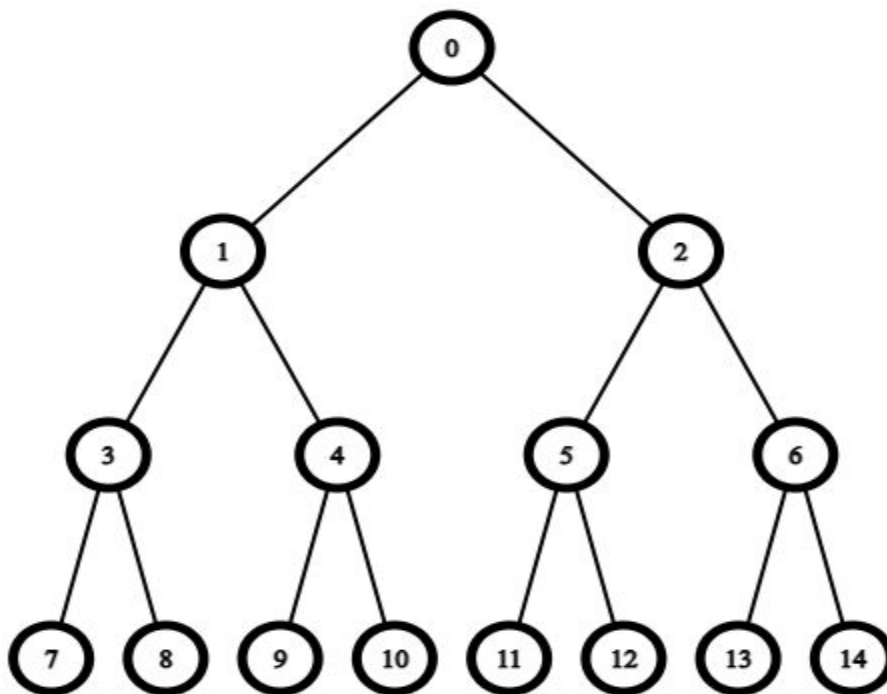
4. Heap:

Result: one connected component.

No Cycles found.

Paths are short.

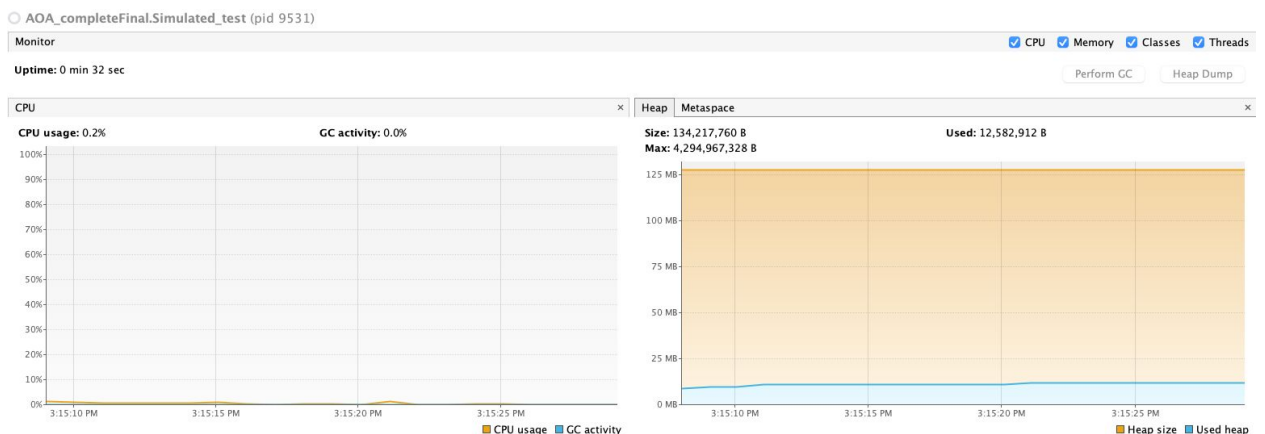
Input 1: No. of nodes (n) = 15



Output 1:

```
Enter number of nodes for Heap Graph
15
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
1
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]Size of this component set is: 15
No. of components present: 1
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
2
There exists no cycle
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
3
Enter starting node:
7
Vertex          Distance      Path
7 -> 0           3             7 3 1 0
7 -> 1           2             7 3 1
7 -> 2           4             7 3 1 0 2
7 -> 3           1             7 3
7 -> 4           3             7 3 1 4
7 -> 5           5             7 3 1 0 2 5
7 -> 6           5             7 3 1 0 2 6
7 -> 8           2             7 3 8
7 -> 9           4             7 3 1 4 9
7 -> 10          4             7 3 1 4 10
7 -> 11          6             7 3 1 0 2 5 11
7 -> 12          6             7 3 1 0 2 5 12
7 -> 13          6             7 3 1 0 2 6 13
7 -> 14          6             7 3 1 0 2 6 14
```

Run Time and Memory for Input 1:

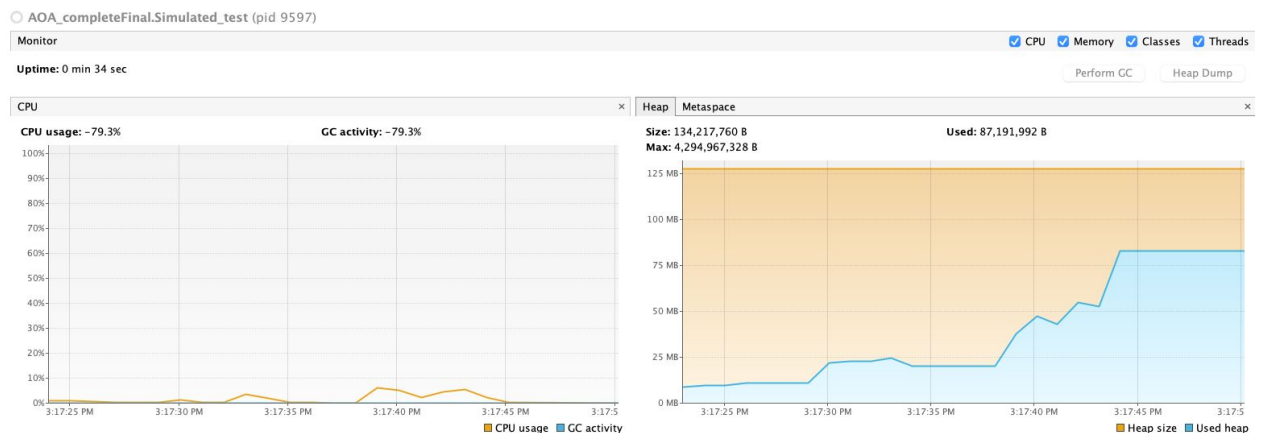


Input 2: No. of nodes (n) =50000

Output 2:

```
architkhatri@Archits-MacBook-Air AOA_completeFinal % /usr/
architkhatri/Library/Application Support/Code/User/workspace
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
4
Enter number of nodes for Heap Graph
50000
```

Run Time and Memory for Input 2:



Analysis:

We can easily observe that both time and memory have increased as the data size is increased. When the data size was increased we can see spikes of memory heap usage which refers to the memory consumption when all three graph operations were run. In the case of small input these spikes were not easily seen as the data was very small.

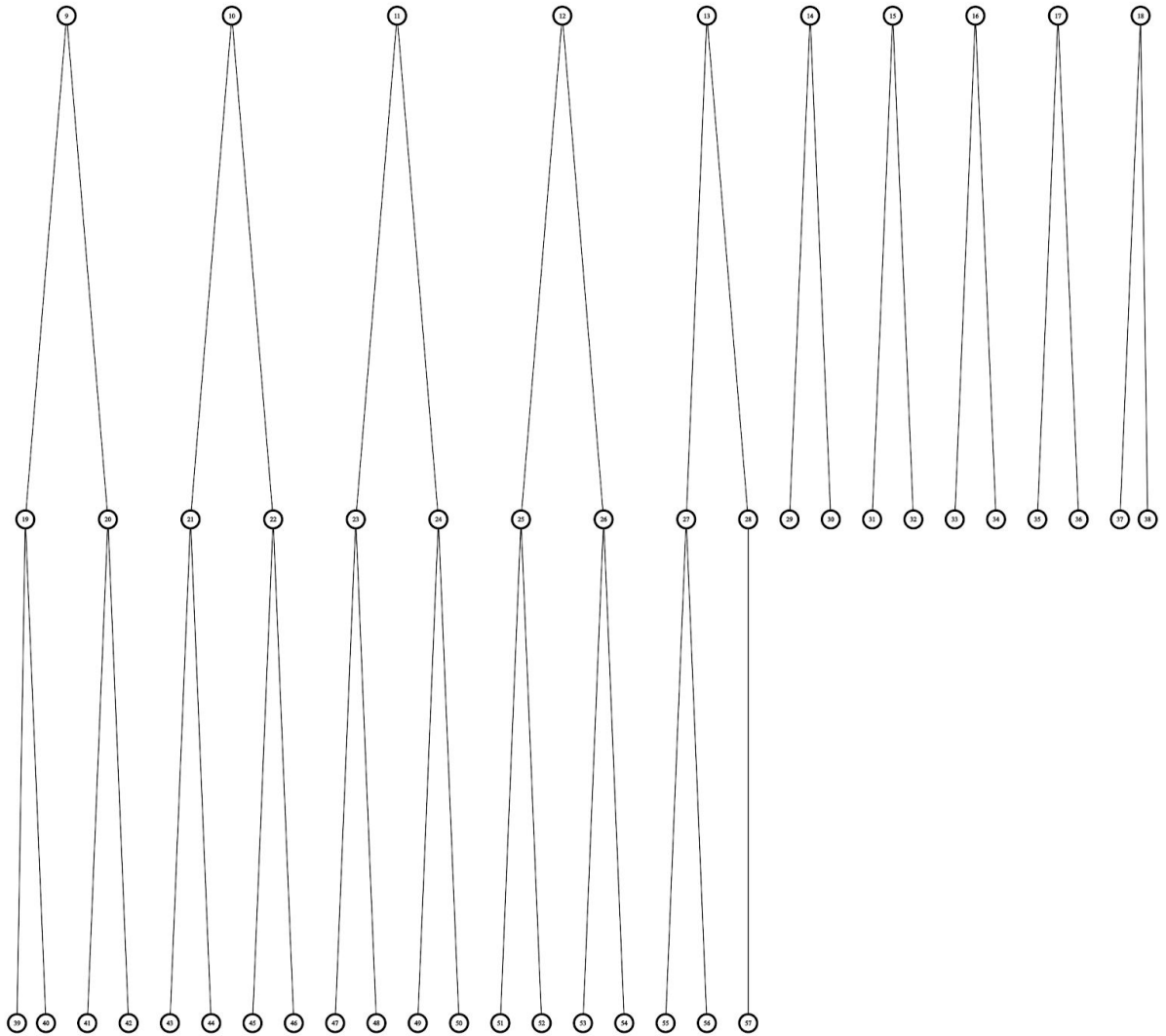
5. Truncated Heap:

Result: $m+1$ connected components

NoCycles found.

Paths are short.

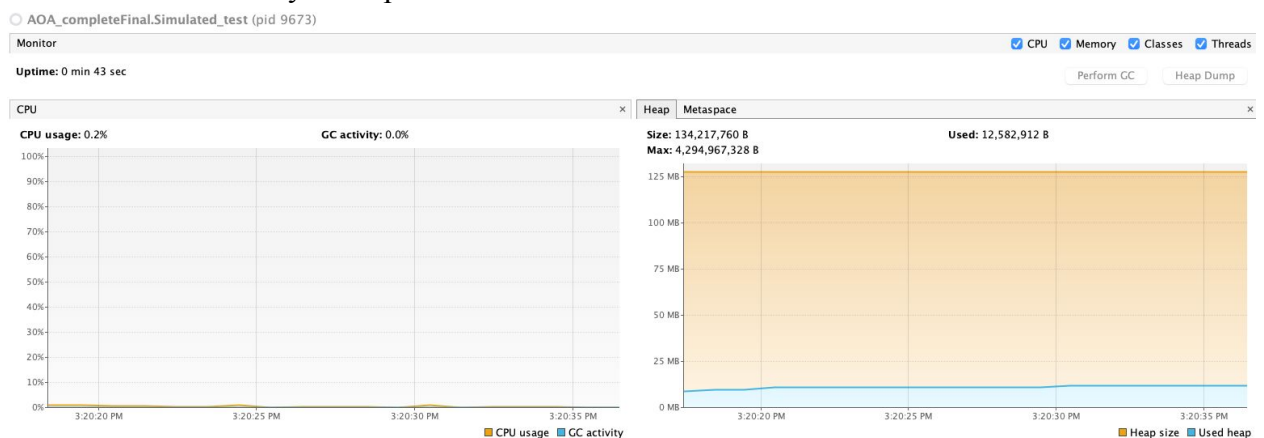
Input 1: No. of nodes (n) = 57 and $m = 9$



Output 1:

```
Enter the last node index 'n' (Starting from 1) for Truncated Heap Graph
57
Enter 'm' for Truncated Heap Graph
9
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
1
[0]Size of this component set is: 1
[1]Size of this component set is: 1
[2]Size of this component set is: 1
[3]Size of this component set is: 1
[4]Size of this component set is: 1
[5]Size of this component set is: 1
[6]Size of this component set is: 1
[7]Size of this component set is: 1
[8]Size of this component set is: 1
[19, 20, 39, 40, 9, 41, 42]Size of this component set is: 7
[21, 22, 10, 43, 44, 45, 46]Size of this component set is: 7
[48, 49, 50, 23, 24, 11, 47]Size of this component set is: 7
[51, 52, 53, 54, 25, 26, 12]Size of this component set is: 7
[55, 56, 27, 28, 13]Size of this component set is: 5
[29, 14, 30]Size of this component set is: 3
[32, 15, 31]Size of this component set is: 3
[16, 33, 34]Size of this component set is: 3
[17, 35, 36]Size of this component set is: 3
[18, 37, 38]Size of this component set is: 3
No. of components present: 19
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
2
There exists no cycle
```

Run Time and Memory for Input 1:

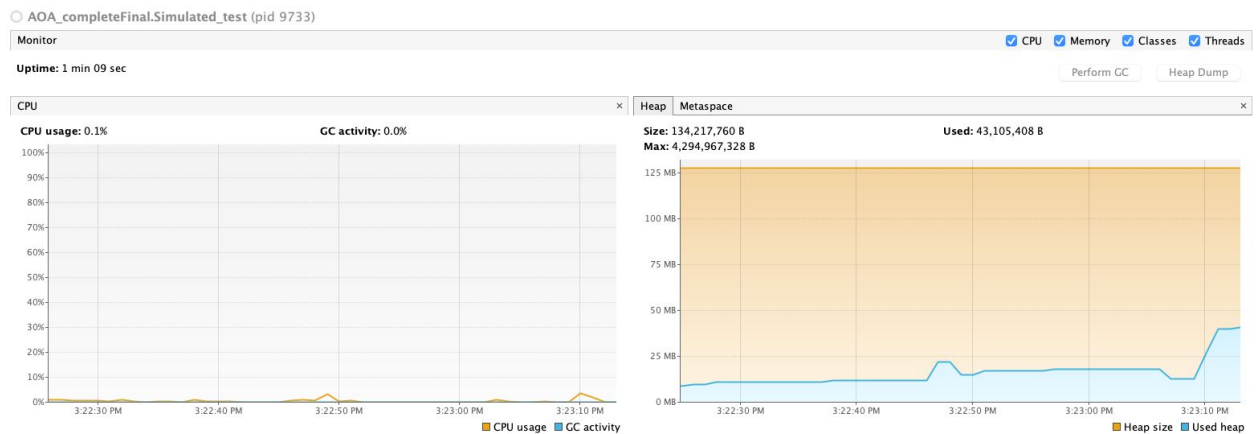


Input 2: No. of nodes (n) =40000 And m=257

Output 2:

```
architkhatri@Archits-MacBook-Air AOA_completeFinal % /usr/bin/env /Library/Architkhatri/Library/Application Support/Code/User/workspaceStorage/ec7bf9
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
5
Enter the last node index 'n' (Starting from 1) for Truncated Heap Graph
40000
Enter 'm' for Truncated Heap Graph
257
```

Run Time and Memory for Input 2:

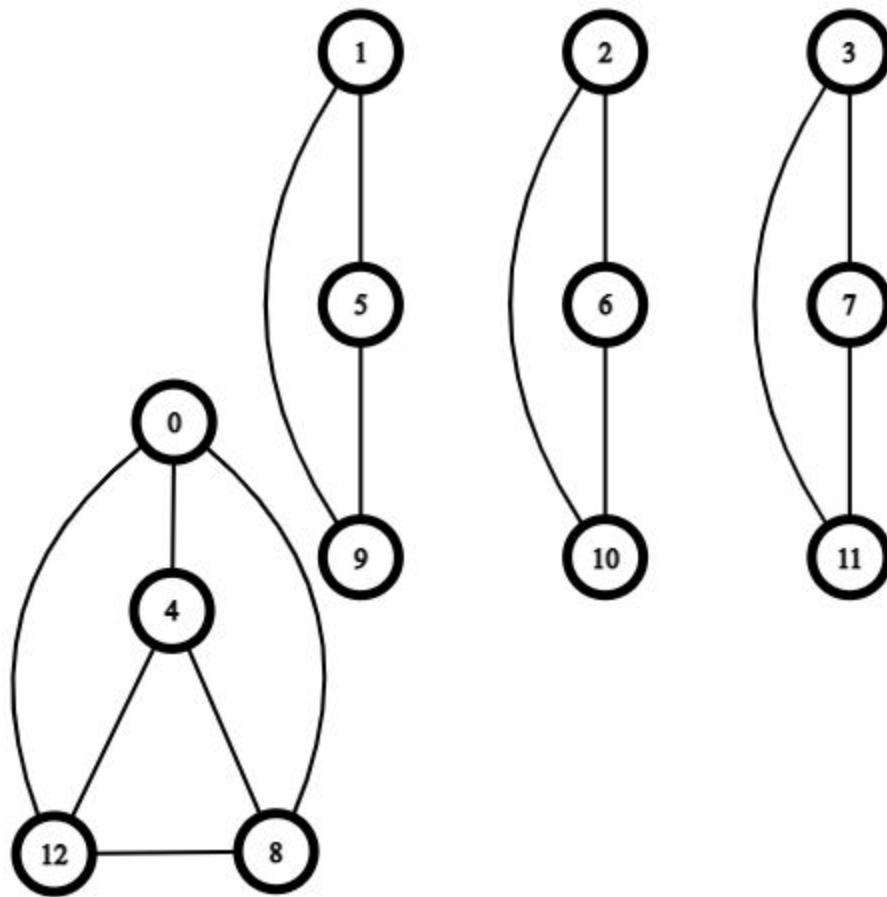


Analysis:

Even after running a large data of 40000 nodes, there was only a moderate change in the memory consumption. CPU usage was about the same for both the cases.

6. Equivalence mod k ($k=4$):
Result: k connected components.

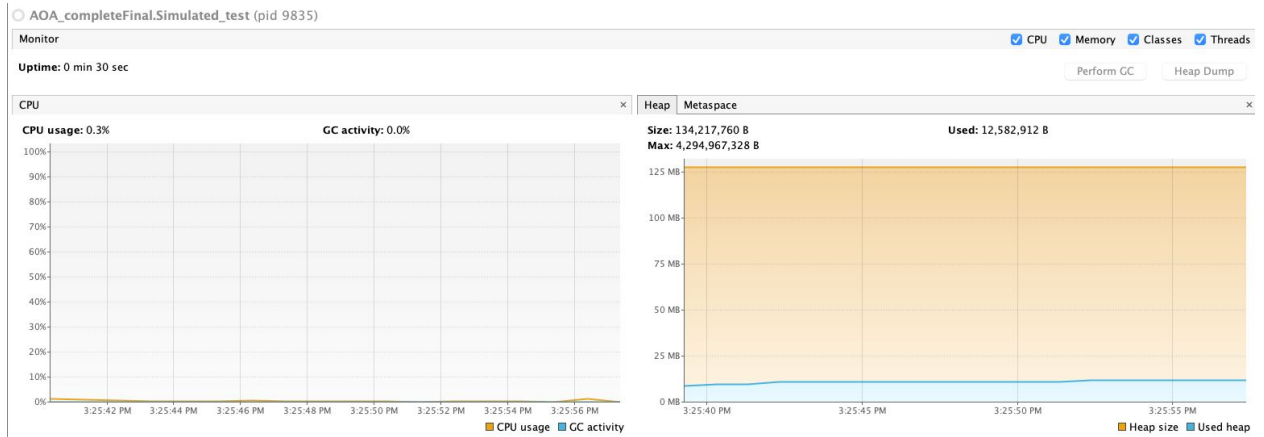
Input 1: No. of nodes (n) = 13



Output 1:

```
Enter number of nodes for Equivalence Mod K Graph
13
Enter the value of mod k
4
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
1
[0, 4, 8, 12]Size of this component set is: 4
[1, 5, 9]Size of this component set is: 3
[2, 6, 10]Size of this component set is: 3
[3, 7, 11]Size of this component set is: 3
No. of components present: 4
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
2
Cycle Detected between following nodes:-
0 4 8
Enter the operation you want to run on your graph:
1: Find Connected Components
2: Find a Cycle in the Graph
3: Shortest Path between a Start node to all other nodes
Press any other key to go back
3
Enter starting node:
3
Vertex          Distance          Path
3 -> 0          Unreachable
3 -> 1          Unreachable
3 -> 2          Unreachable
3 -> 4          Unreachable
3 -> 5          Unreachable
3 -> 6          Unreachable
3 -> 7          1              3 7
3 -> 8          Unreachable
3 -> 9          Unreachable
3 -> 10         Unreachable
3 -> 11         1              3 11
3 -> 12         Unreachable
```

Run Time and Memory for Input 1:



Input 2: No. of nodes (n) = 10000

Output 2:

```
architkhatri@Archits-MacBook-Air AOA_completeFinal
architkhatri/Library/Application Support/Code/User/v
test
Enter number from given choices:
1: Create Graph using simulator
2: Create a Graph manually
Enter any other key to exit
1
Enter the operation you want to run:
1: n-cycle graph
2: complete graph
3: empty graph
4: heap graph
5: truncated heap graph
6: equivalence mod k graph
Enter any key to go to previous menu
6
Enter number of nodes for Equivalence Mod K Graph
10000
Enter the value of mod k
4
```

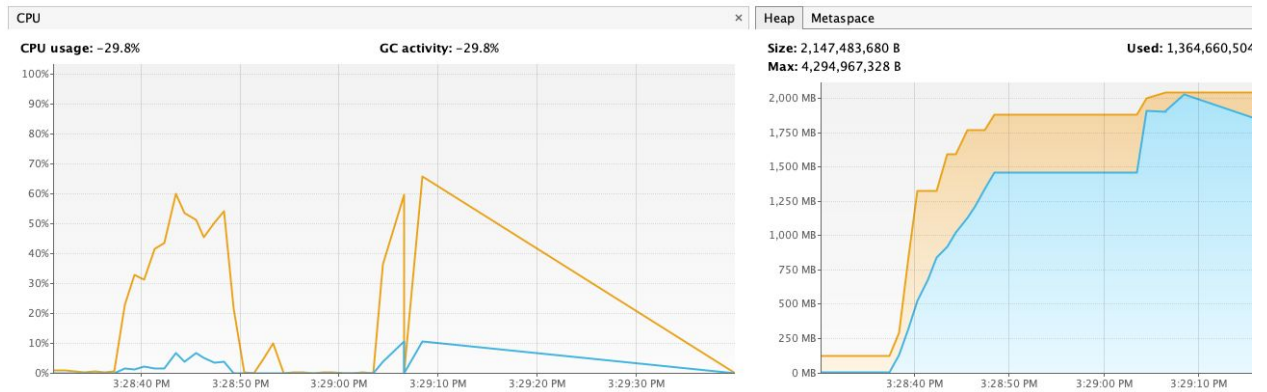
Run Time and Memory for Input 2:

AOA_completeFinal.Simulated_test (pid 9926)

Monitor

CPU

Uptime: 1 min 22 sec



Analysis:

We can easily observe that both time and memory have increased as the data size is increased. In the case of large data (nodes = 10000), we can see that there was huge memory consumption for the creation of the second graph. CPU usage was also high in this case and runtime was significantly higher.

Netflix Data (Real Data Testing):

Hardware Configuration -

Laptop - MacBook Air M1

Storage - 256GB

RAM - 8GB

Data Structures used for creating Adjacency Lists:

1. `HashMap<Integer, HashMap<Pair<Integer, Integer>>> map_og`
This hash map contains movie_id as the key and customer_id, movie_rating as the pair.
2. `HashMap<Integer, Integer> index_map`
This hashmap maps an unique index for each customer id.
3. `HashMap<Integer, HashSet<Integer>> sec_map`
This hashmap contains customer_id as the key and set of movies as values that customer_id has rated.

Data Structure used for Adjacency List:

`HashSet[] adjList` - It is an array of HashSet where each index in the array corresponds to a unique customer_id in the index_map (above) and in the HashSet, we have the edges (other customers who have watched the same movie or similar rating)

Functions for Adjacency List:

We made three separate functions for three adjacency criteria, each function generates `HashSet[] adjList` as the output based on the below criteria.

The Three criterias for creating the Adjacency List for the Netflix data are:

1. Two customers have watched the same movie - If 100 movies are read, there can be a maximum of 100 components where in each component, there is a complete graph of customers as a node connected to each other. If a customer appears in another movie, then the component of that movie will get connected to the other component of the movie in which that customer is present, creating a path between both the components.
2. Two customers have watched the same movie and rated it 4-5 star - In this criteria, we tried to form a component, where each customer in the component has watched the movie and liked it, forming a component of customers having a similar movie taste. We can recommend movies based on how far the other components lie from the parent component. Farthest components shall be recommended the least and closest components shall be recommended with high confidence.
3. Two customers have watched the same movie and rated it 1-3 star - This is like the reverse criteria of the above adjacency scheme. In this criteria, a component of nodes (customers) will represent a movie that the customer didn't like. So, the farthest most

components may represent a movie that could potentially be liked by the customers in the component where everyone has rated the movie between 1 to 3 stars.

While building our adjacency lists, we went through multiple problems but the most significant problem was building the Adjacency list and running out of Heap Space because we used three Hashmaps to store index and customer id such that Adjacency criteria 2 and 3 could be made.

The error looked like this:

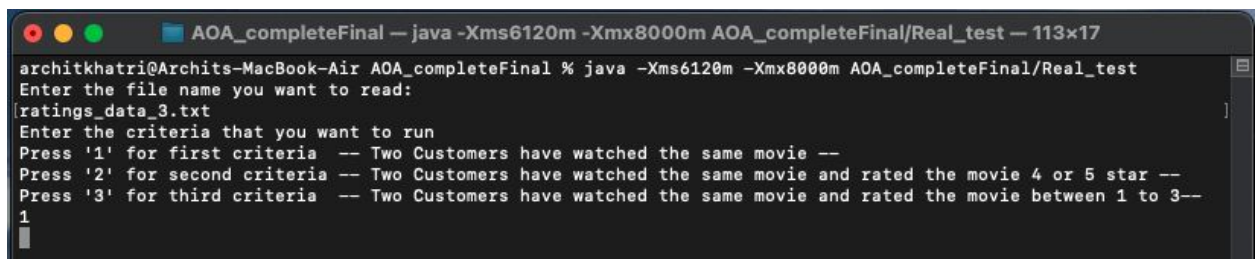
```
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
    at java.base/java.lang.Integer.valueOf(Integer.java:1059)
    at AOA_completeFinal.Graph_make.makeAdjFirst(Graph_make.java:139)
    at AOA_completeFinal.Graph_make.readFileAndCreateAdjList(Graph_make.java
:96)
    at AOA_completeFinal.Real_test.UIReal(Real_test.java:41)
    at AOA_completeFinal.Real_test.main(Real_test.java:74)
```

Some of the reasons for this error could be the amount of RAM on our machine was only 8GB and we need at least thrice that amount to be able to run the data properly.

To solve this problem, we did following solution trials:

1. Use of xms and xmx commands in java to set min and max heap size in command arguments.

This command usage looks like this



```
AOA_completeFinal — java -Xms6120m -Xmx8000m AOA_completeFinal/Real_test — 113x17
architkhatri@Archits-MacBook-Air AOA_completeFinal % java -Xms6120m -Xmx8000m AOA_completeFinal/Real_test
Enter the file name you want to read:
ratings_data_3.txt
Enter the criteria that you want to run
Press '1' for first criteria -- Two Customers have watched the same movie --
Press '2' for second criteria -- Two Customers have watched the same movie and rated the movie 4 or 5 star --
Press '3' for third criteria -- Two Customers have watched the same movie and rated the movie between 1 to 3--
1
```

Our process used complete 8GB of ram by the above command to run as much as it can, but that wasn't enough.

Process Name	Mem... ▾
java	8.19 GB

We ran three adjacency criteria, on the maximum amount of data that we were able to read and following is a sample of the Real_test.java:

```
architkhatri@Archits-MacBook-Air AOA_completeFinal % java AOA_completeFinal.Real_test
Enter the file name you want to read:
ratings_data_3.txt
Enter the criteria that you want to run
Press '1' for first criteria -- Two Customers have watched the same movie --
Press '2' for second criteria -- Two Customers have watched the same movie and rated the movie 4 or 5 star --
Press '3' for third criteria -- Two Customers have watched the same movie and rated the movie between 1 to 3--
2
Enter the operation that you want to run:
Press 1 to find Connected Components
Press 2 to Detect a Cycle in the Graph
Press 3 to find the Shortest Path between a starting node to all other nodes
Press any other key to return to previous menu.
3
Enter the starting node between 0 to 7232
5000
```

Source to Destination	Distance	Path
5000 -> 1249	2	5000 948 1249
5000 -> 1250	Unreachable	
5000 -> 1251	Unreachable	
5000 -> 1252	Unreachable	
5000 -> 1253	2	5000 948 1253
5000 -> 1254	2	5000 948 1254
5000 -> 1255	Unreachable	
5000 -> 1256	2	5000 948 1256
5000 -> 1257	2	5000 948 1257
5000 -> 1258	2	5000 948 1258
5000 -> 1259	Unreachable	
5000 -> 1260	Unreachable	
5000 -> 1261	2	5000 948 1261
5000 -> 1262	2	5000 948 1262
5000 -> 1263	Unreachable	
5000 -> 1264	2	5000 948 1264
5000 -> 1265	2	5000 948 1265
5000 -> 1266	2	5000 948 1266
5000 -> 1267	2	5000 948 1267
5000 -> 1268	2	5000 948 1268
5000 -> 1269	Unreachable	
5000 -> 1270	2	5000 948 1270
5000 -> 1271	2	5000 948 1271

```
Enter the criteria that you want to run
Press '1' for first criteria -- Two Customers have watch
ed the same movie --
Press '2' for second criteria -- Two Customers have watch
ed the same movie and rated the movie 4 or 5 star --
Press '3' for third criteria -- Two Customers have watch
ed the same movie and rated the movie between 1 to 3--
3
Enter the operation that you want to run:
Press 1 to find Connected Components
Press 2 to Detect a Cycle in the Graph
Press 3 to find the Shortest Path between a starting node
to all other nodes
Press any other key to return to previous menu.
1
```

```

6887, 6889, 6891, 6892, 6896, 6898, 6899, 6883, 6889, 68
90, 6891, 6898, 6899, 6900, 6902, 6905, 6906, 6912, 6913,
6914, 6915, 6916, 6917, 6918, 6922, 6923, 6924, 6926, 69
27, 6932, 6934, 6936, 6937, 6938, 6939, 6941, 6943, 6944,
6947, 6948, 6950, 6952, 6955, 6957, 6959, 6960, 6966, 69
67, 6968, 6973, 6975, 6976, 6977, 6978, 6982, 6987, 6988,
6989, 6991, 6995, 6998, 7000, 7004, 7008, 7012, 7015, 70
16, 7018, 7021, 7022, 7023, 7029, 7033, 7034, 7035, 7036,
7039, 7040, 7041, 7042, 7044, 7050, 7053, 7055, 7057, 70
59, 7061, 7069, 7072, 7074, 7076, 7077, 7079, 7083, 7084,
7085, 7086, 7087, 7090, 7091, 7092, 7093, 7095, 7096, 70
97, 7100, 7101, 7102, 7105, 7108, 7112, 7113, 7115, 7119,
7121, 7122, 7123, 7125, 7127, 7130, 7131, 7132, 7136, 71
37, 7139, 7140, 7141, 7142, 7144, 7145, 7146, 7148, 7150,
7156, 7158, 7159, 7163, 7164, 7165, 7166, 7172, 7176, 71
79, 7180, 7181, 7183, 7184, 7185, 7186, 7187, 7188, 7189,
7190, 7191, 7192, 7193, 7196, 7197, 7198, 7199, 7200, 72
03, 7204, 7205, 7206, 7207, 7208, 7210, 7212, 7220, 7223,
7227, 7228]Size of this component set is: 3848
[4]Size of this component set is: 1
[9]Size of this component set is: 1
[18]Size of this component set is: 1
[36]Size of this component set is: 1
[55]Size of this component set is: 1
[56]Size of this component set is: 1
[87]Size of this component set is: 1

```

```

[7225]Size of this component set is: 1
[7226]Size of this component set is: 1
[7229]Size of this component set is: 1
[7230]Size of this component set is: 1
[7231]Size of this component set is: 1
No. of components present: 3385

```

```

architkhatri@Archits-MacBook-Air AOA_completeFinal % java AOA_completeFinal.Real
_test
Enter the file name you want to read:
ratings_data_3.txt
Enter the criteria that you want to run
Press '1' for first criteria -- Two Customers have watched the same movie --
Press '2' for second criteria -- Two Customers have watched the same movie and r
ated the movie 4 or 5 star --
Press '3' for third criteria -- Two Customers have watched the same movie and r
ated the movie between 1 to 3--
1
Enter the operation that you want to run:
Press 1 to find Connected Components
Press 2 to Detect a Cycle in the Graph
Press 3 to find the Shortest Path between a starting node to all other nodes
Press any other key to return to previous menu.
2
Cycle Detected between following nodes:-
0 1 2

```


Meanwhile our memory usage and CPU usage looks like this:

AOA_completeFinal.Real_test (pid 12921)

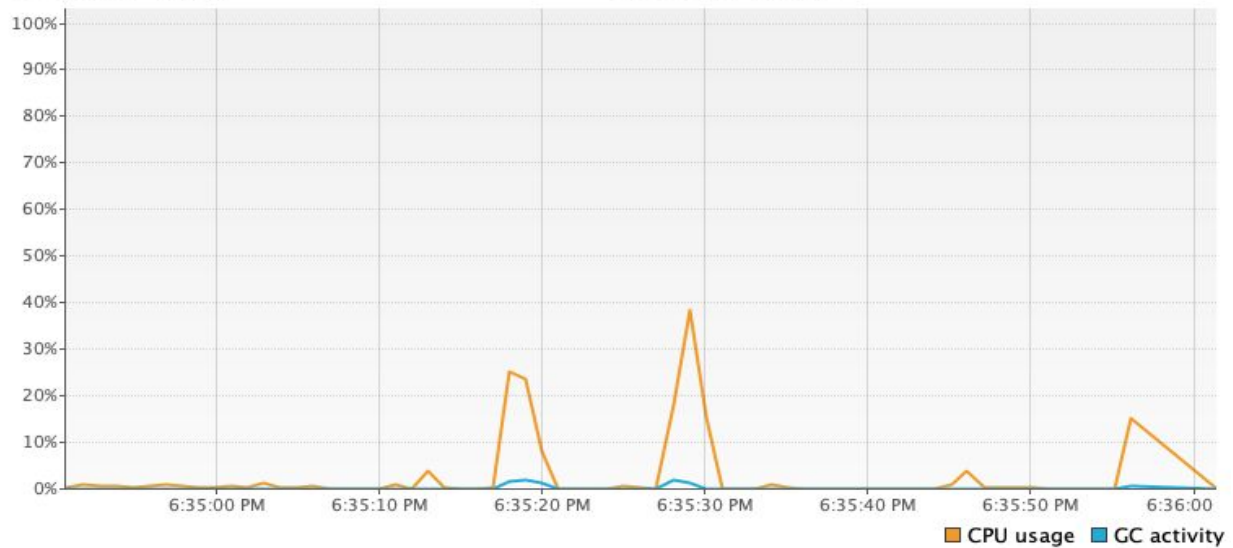
Monitor

Uptime: 2 min 16 sec

CPU

CPU usage: -230.4%

GC activity: -230.4%



Perform GC

Heap Dump

Heap

Metaspace

Size: 1,410,334,752 B

Max: 4,294,967,328 B

Used: 759,693,312 B

