Name: Siddharth Gupta

UFID: 85550520

Section Number: COT 5405, Spring 2021

Date Due February 15, 2020

Assignment 1

THE ASSIGNMENT HAVE THE FOLLOWING JAVA CLASSES UNDER **AOA_PROJECT_FINAL PACKAGE/FILE**

1) graph_operations.java
2) graph_simulator.java
3) simulated_test.java
4) graph_make.java
5) real_test.java

In the assignment I have used **HashMap<<Integer>, Vector<Integer>>** and **Vector < Vector <Integer>>** data structure to store my graph.

Space complexity

The space complexity for the following data structures is of order O(|V|+|E|) in which V represents number of Nodes/Vertices and E represents number of edges.
In the worst case, there can be V*V number of edges in a graph thus consuming O(V^2) space.

Time complexity

Queries like whether there is an edge from vertex source to vertex destination can be done O(V).

Graph_operations.java

The file graph_operations.java contains the following functions.

1) DepthFirstUntil () and connected_Components ()

These functions use depth first search on the **graph HashMap<<Integer>, Vector<Integer>> adjList** to find and print all the connected components.

2) isCyclicUntil () and isCyclic ()

These functions use depth first search on the **graph HashMap<<Integer>, Vector <Integer>> adjList** to detect presence of cycle and returns true if a cycle exists.

3) BFS() and printShortestDistance ()

These functions take source node and destination node from the user and returns the shortest distance and the path between the nodes using Dijkstra's shortest path algorithm in the **graph HashMap<<Integer>, Vector <Integer>> adjList** , breadth first search is used to traverse the nodes of the graph.

The file graph_simulator.java contains the following functions.

4) nCycle ()

This function takes **graph HashMap<<Integer>, Vector <Integer>> adjList** as input and creates edges in the graph if **$u - v = \pm 1$ or $u - v = \pm(n - 1)$** and returns the graph containing edges, it contains **1 connected component and 1 cycle** of length **N,** the shortest distance between 2 nodes is **from 1 to N/2** where **N** is number of nodes

5) complete_graph ()

This function takes **graph HashMap<<Integer>, Vector <Integer>> adjList** as input and creates edges such that each node is connected with every other node and returns a complete graph. There is 1 connected component and many cycles of different lengths. The shortest distance between any 2 nodes in the graph is 1

6) equivalence_mod_k ()

This function takes an integer input k from the user and if the **remainder of (node1-node2)/k is 0** then it creates an edge between node1 and node2 in the **graph HashMap<<Integer>, Vector <Integer>> adjList.** In this graph there are **connected components equal to k** and there are **k unique cycles.** The shortest path various and it is possible that **there is no path between two nodes.**

The file graph_simulator.java contains the following functions.

7) graphmake ()

This function takes an integer value N from the user and returns a **HashMap<<Integer>, Vector <Integer>> adjList of size N**. **HashMap<<Integer>, Vector <Integer>> adjList** is used to store the graph.

8) printGraph ()

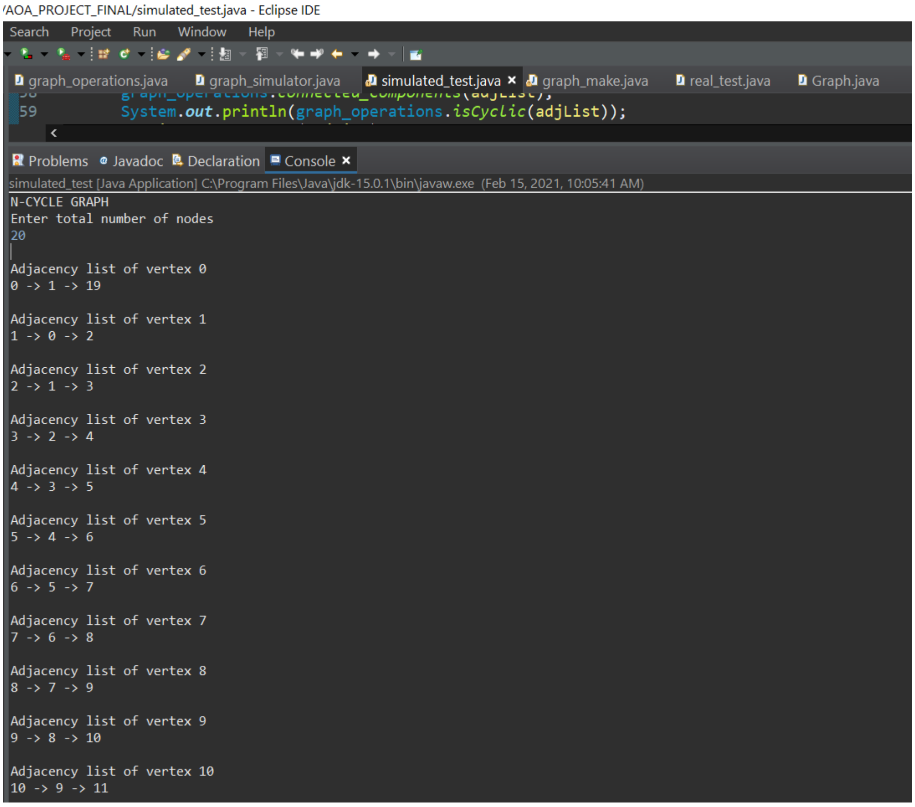
This function prints the current **Graph** and its edges.
HashMap<<Integer>, Vector <Integer>> adjList of size N

9) hashtoarr ()

This function returns a takes **HashMap<<Integer>, ArrayList<Integer>> adjList** as parameter and creates a new **Graph Vector < Vector <Integer>> arrList** and copies all the vertices and edges from **adjList** to **arrList.**

OUTPUTS OF SIMULATED_TEST.JAVA

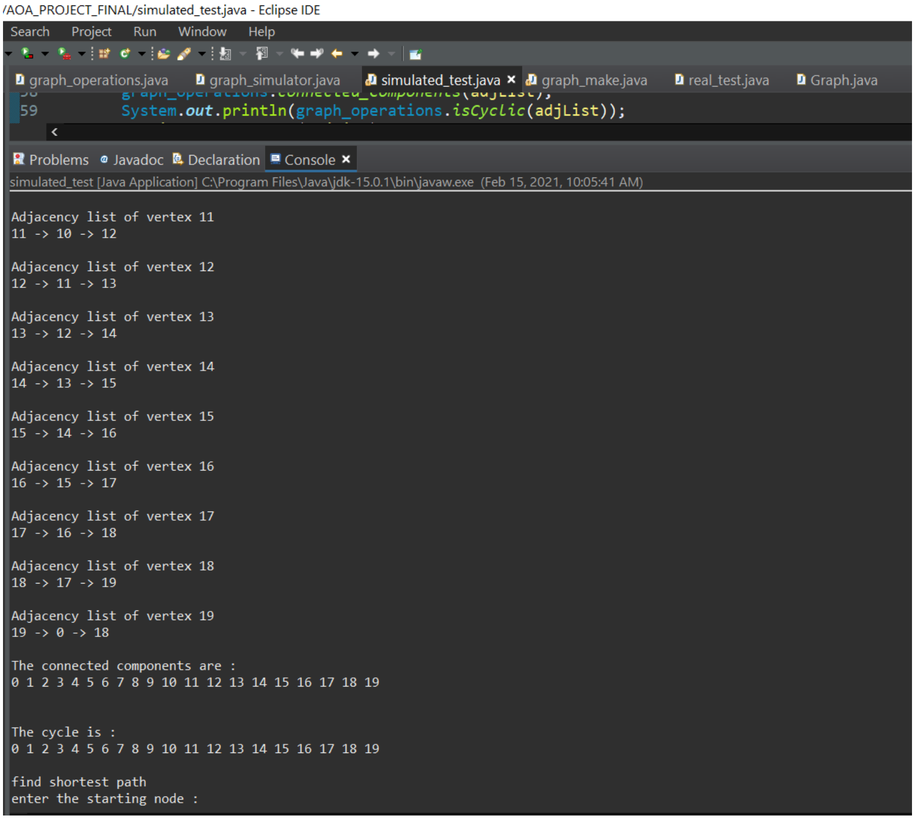
1) N-CYCLE



```
        graph_operations.connected_components(adjList);
        System.out.println(graph_operations.isCyclic(adjList));
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:05:41 AM)

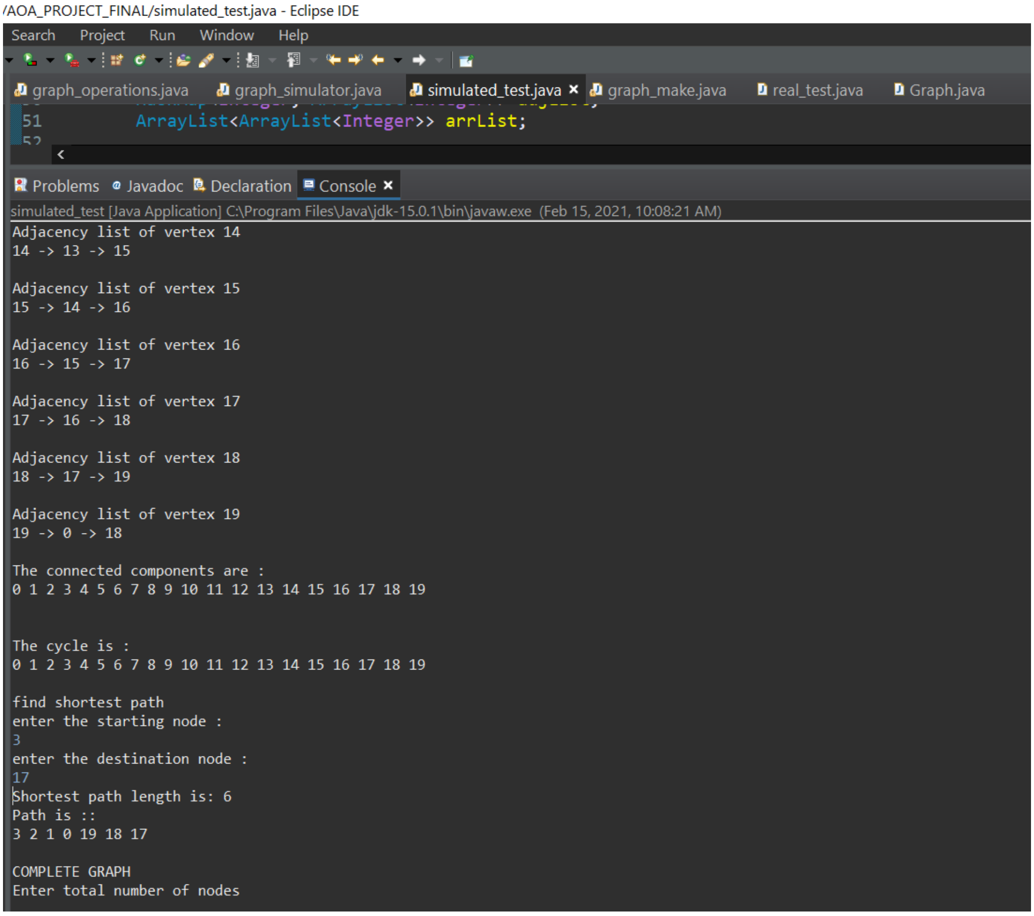
```
N-CYCLE GRAPH
Enter total number of nodes
20

Adjacency list of vertex 0
0 -> 1 -> 19

Adjacency list of vertex 1
1 -> 0 -> 2

Adjacency list of vertex 2
2 -> 1 -> 3

Adjacency list of vertex 3
3 -> 2 -> 4

Adjacency list of vertex 4
4 -> 3 -> 5

Adjacency list of vertex 5
5 -> 4 -> 6

Adjacency list of vertex 6
6 -> 5 -> 7

Adjacency list of vertex 7
7 -> 6 -> 8

Adjacency list of vertex 8
8 -> 7 -> 9

Adjacency list of vertex 9
9 -> 8 -> 10

Adjacency list of vertex 10
10 -> 9 -> 11
```



```
        graph_operations.connected_components(adjList);
        System.out.println(graph_operations.isCyclic(adjList));
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:05:41 AM)

```
Adjacency list of vertex 11
11 -> 10 -> 12

Adjacency list of vertex 12
12 -> 11 -> 13

Adjacency list of vertex 13
13 -> 12 -> 14

Adjacency list of vertex 14
14 -> 13 -> 15

Adjacency list of vertex 15
15 -> 14 -> 16

Adjacency list of vertex 16
16 -> 15 -> 17

Adjacency list of vertex 17
17 -> 16 -> 18

Adjacency list of vertex 18
18 -> 17 -> 19

Adjacency list of vertex 19
19 -> 0 -> 18

The connected components are :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19


The cycle is :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

find shortest path
enter the starting node :
```

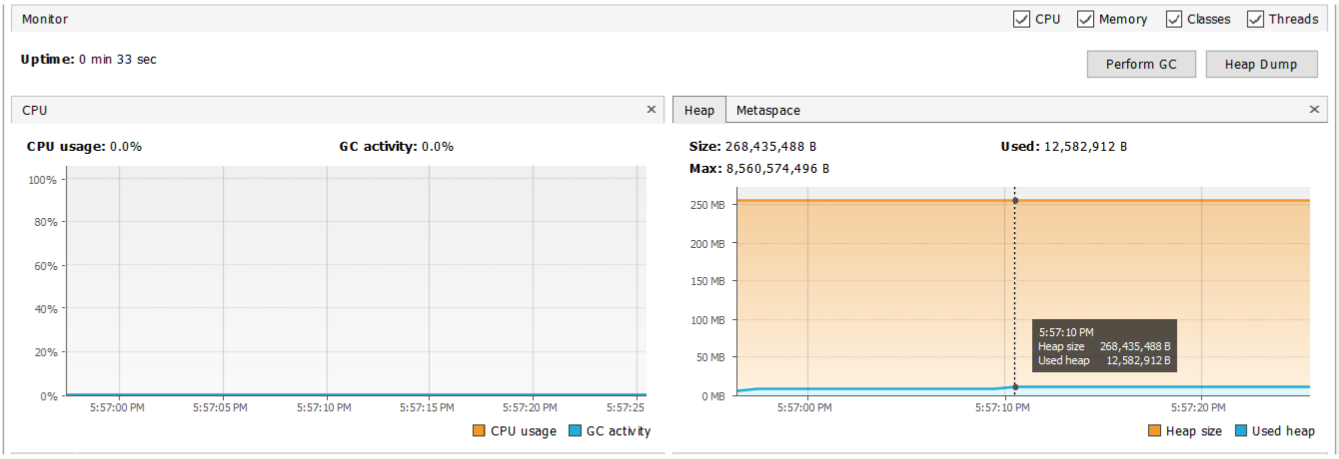


```
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:21 AM)

```
Adjacency list of vertex 14
14 -> 13 -> 15

Adjacency list of vertex 15
15 -> 14 -> 16

Adjacency list of vertex 16
16 -> 15 -> 17

Adjacency list of vertex 17
17 -> 16 -> 18

Adjacency list of vertex 18
18 -> 17 -> 19

Adjacency list of vertex 19
19 -> 0 -> 18

The connected components are :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19


The cycle is :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

find shortest path
enter the starting node :
3
enter the destination node :
17
Shortest path length is: 6
Path is ::
3 2 1 0 19 18 17

COMPLETE GRAPH
Enter total number of nodes
```

MEMORY USAGE AND CPU TIME FOR N-CYCLE

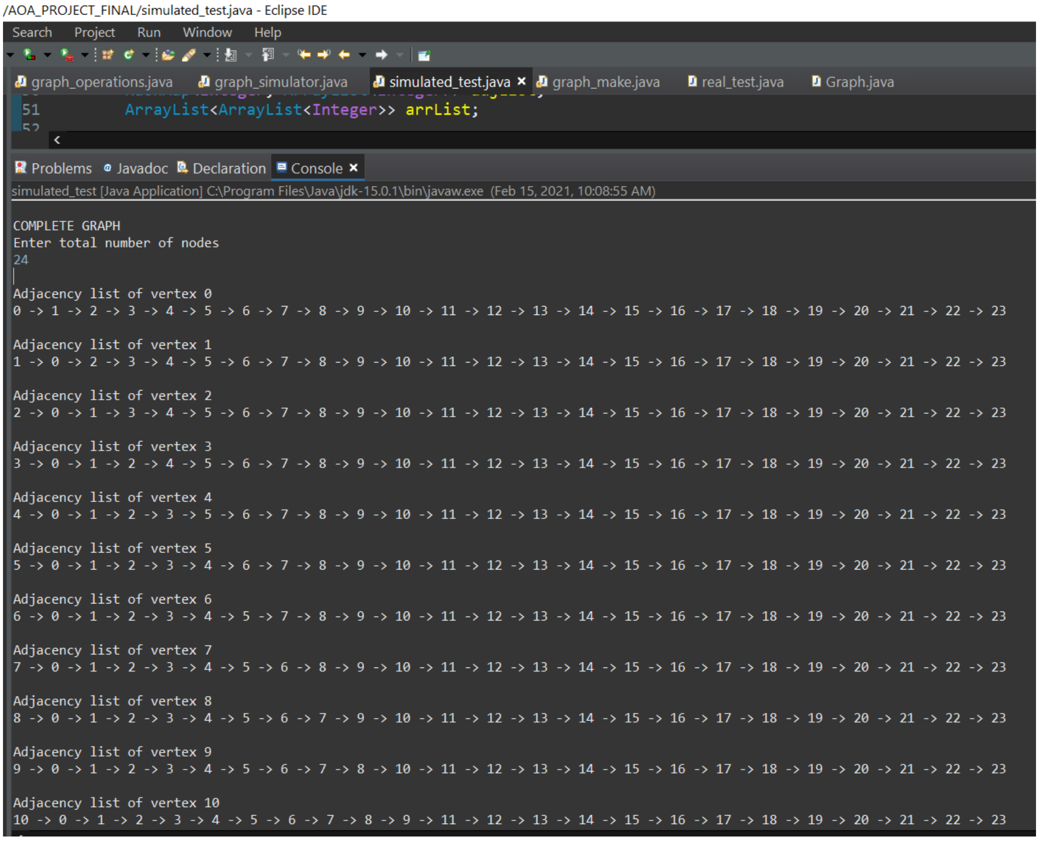


Uptime: 0 min 33 sec

CPU usage: 0.0% **GC activity:** 0.0%

Size: 268,435,488 B **Used:** 12,582,912 B
Max: 8,560,574,496 B

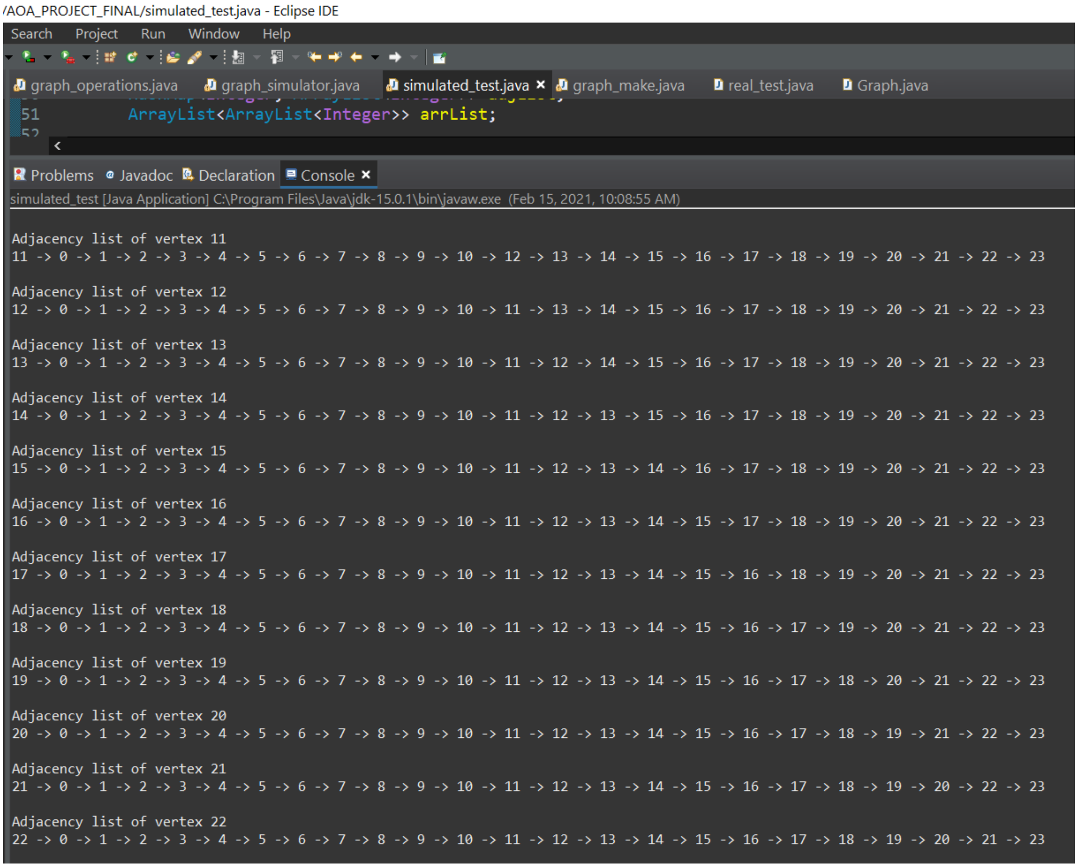
2) COMPLETE GRAPH



```
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM)

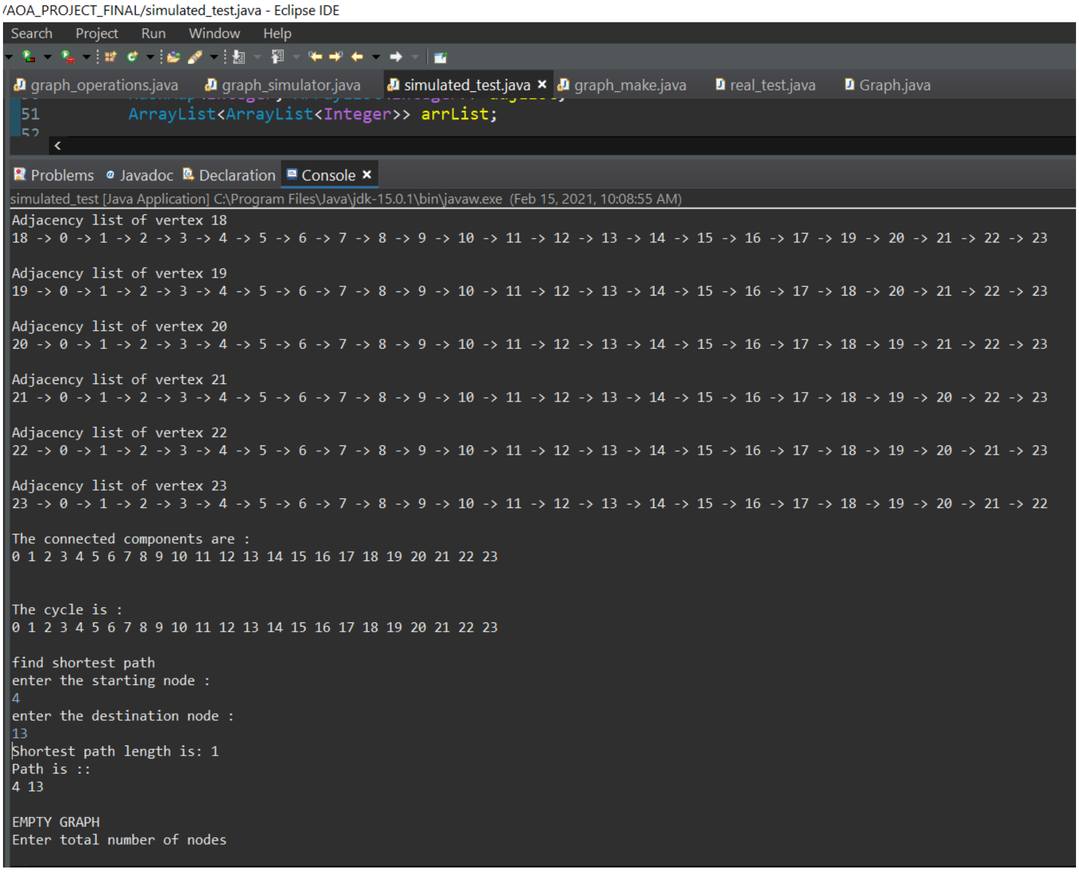
```
COMPLETE GRAPH
Enter total number of nodes
24

Adjacency list of vertex 0
0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 1
1 -> 0 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 2
2 -> 0 -> 1 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 3
3 -> 0 -> 1 -> 2 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 4
4 -> 0 -> 1 -> 2 -> 3 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 5
5 -> 0 -> 1 -> 2 -> 3 -> 4 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 6
6 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 7
7 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 8
8 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 9
9 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 10
10 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23
```



```
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM)

```
Adjacency list of vertex 11
11 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 12
12 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 13
13 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 14
14 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 15
15 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 16
16 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 17
17 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 18 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 18
18 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 19
19 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 20
20 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 21 -> 22 -> 23

Adjacency list of vertex 21
21 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 22 -> 23

Adjacency list of vertex 22
22 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 23
```

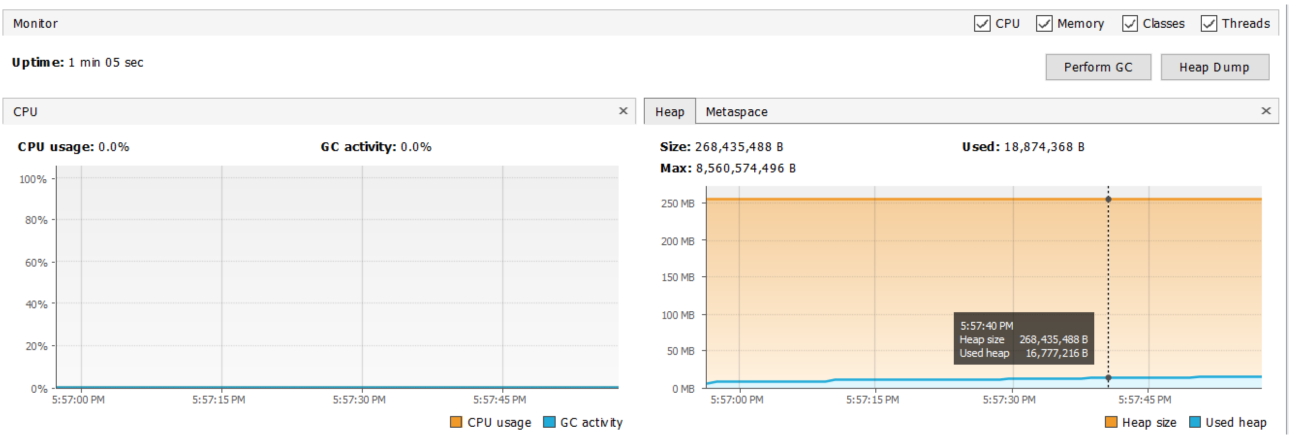


```
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM)

```
Adjacency list of vertex 18
18 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 19 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 19
19 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 20 -> 21 -> 22 -> 23

Adjacency list of vertex 20
20 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 21 -> 22 -> 23

Adjacency list of vertex 21
21 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 22 -> 23

Adjacency list of vertex 22
22 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 23

Adjacency list of vertex 23
23 -> 0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11 -> 12 -> 13 -> 14 -> 15 -> 16 -> 17 -> 18 -> 19 -> 20 -> 21 -> 22

The connected components are :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23


The cycle is :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23

find shortest path
enter the starting node :
4
enter the destination node :
13
Shortest path length is: 1
Path is ::
4 13

EMPTY GRAPH
Enter total number of nodes
```

MEMORY USAGE AND CPU TIME FOR COMPLETE GRAPH



Uptime: 1 min 05 sec

CPU usage: 0.0%

GC activity: 0.0%

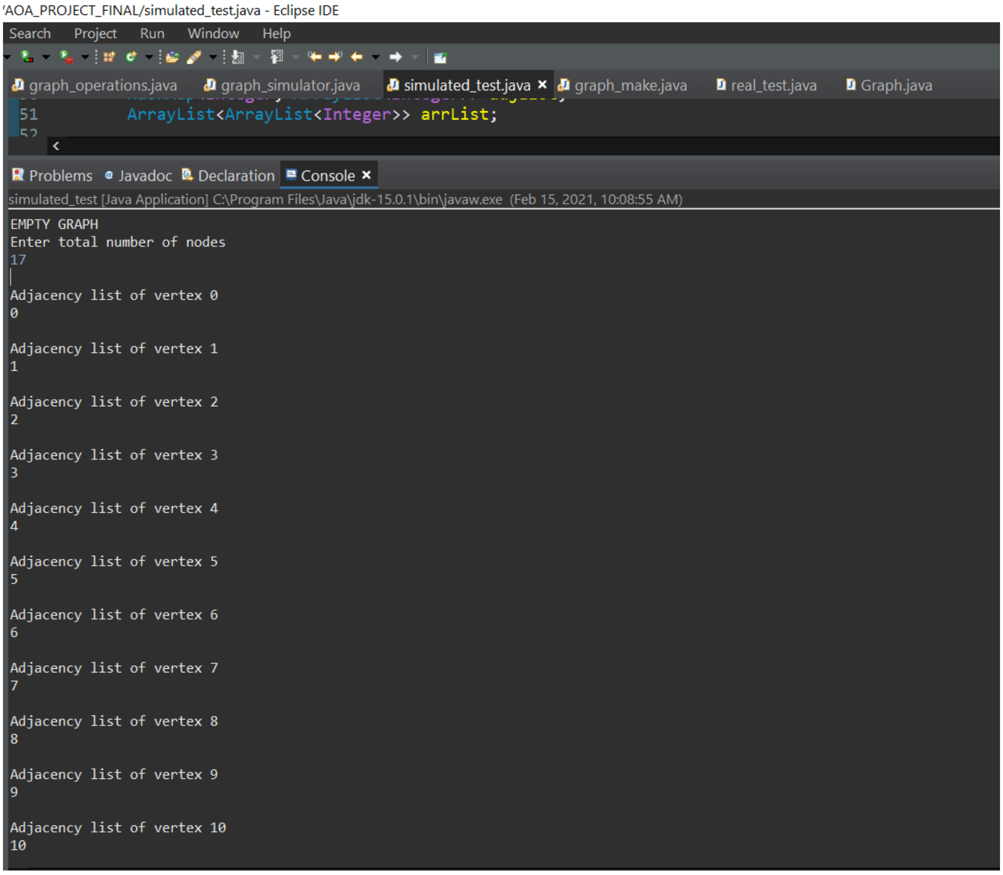
Size: 268,435,488 B

Used: 18,874,368 B

Max: 8,560,574,496 B

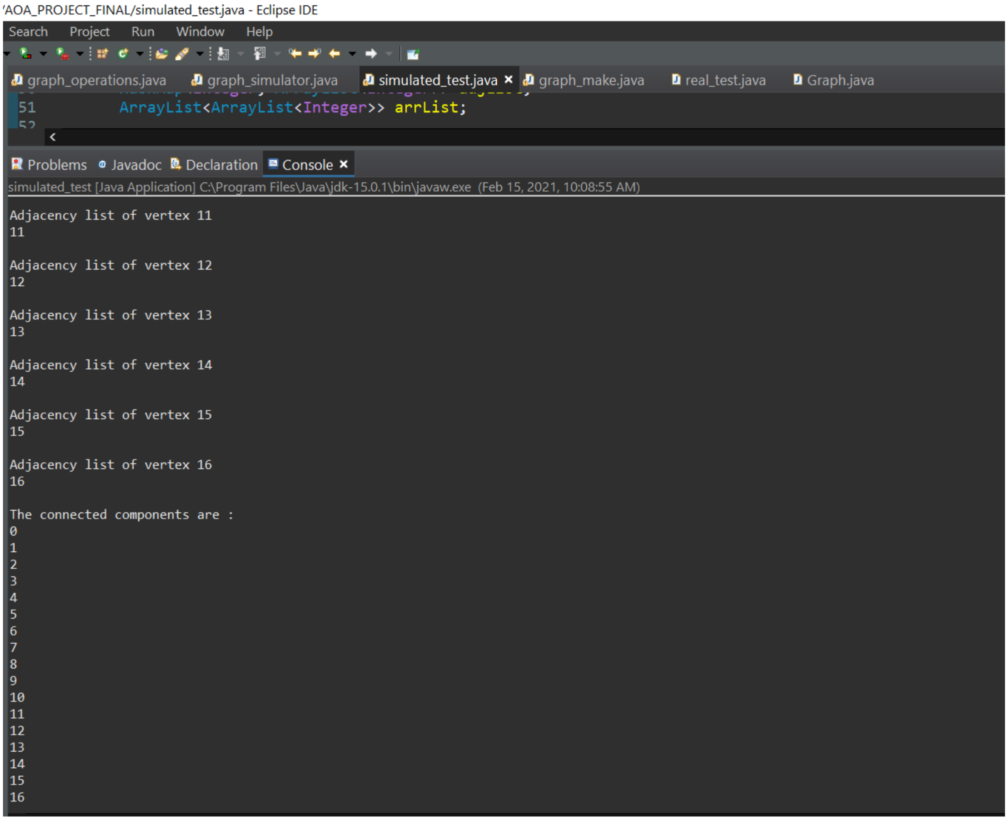
5:57:40 PM
Heap size 268,435,488 B
Used heap 16,777,216 B

3) EMPTY GRAPH



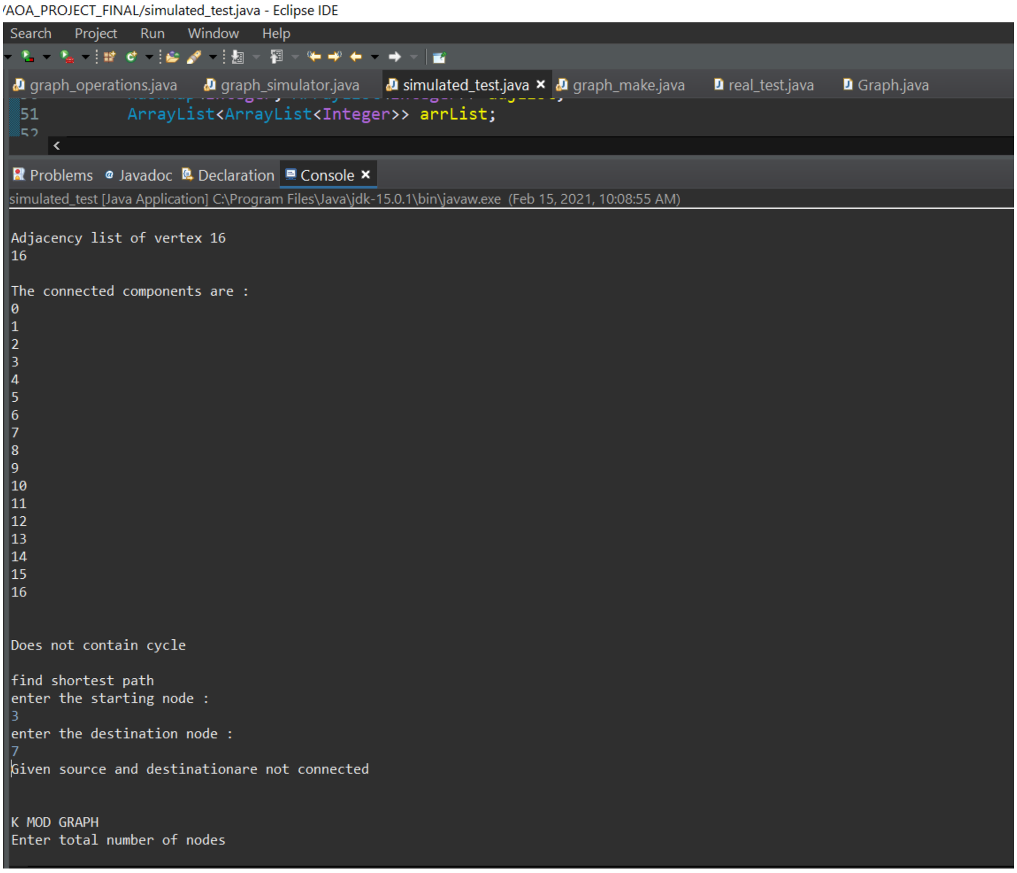
```
ArrayList<ArrayList<Integer>> arrList;
```

```
simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe  (Feb 15, 2021, 10:08:55 AM)
EMPTY GRAPH
Enter total number of nodes
17

Adjacency list of vertex 0
0

Adjacency list of vertex 1
1

Adjacency list of vertex 2
2

Adjacency list of vertex 3
3

Adjacency list of vertex 4
4

Adjacency list of vertex 5
5

Adjacency list of vertex 6
6

Adjacency list of vertex 7
7

Adjacency list of vertex 8
8

Adjacency list of vertex 9
9

Adjacency list of vertex 10
10
```



```
ArrayList<ArrayList<Integer>> arrList;
```

```
simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe  (Feb 15, 2021, 10:08:55 AM)
Adjacency list of vertex 11
11

Adjacency list of vertex 12
12

Adjacency list of vertex 13
13

Adjacency list of vertex 14
14

Adjacency list of vertex 15
15

Adjacency list of vertex 16
16

The connected components are :
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
```

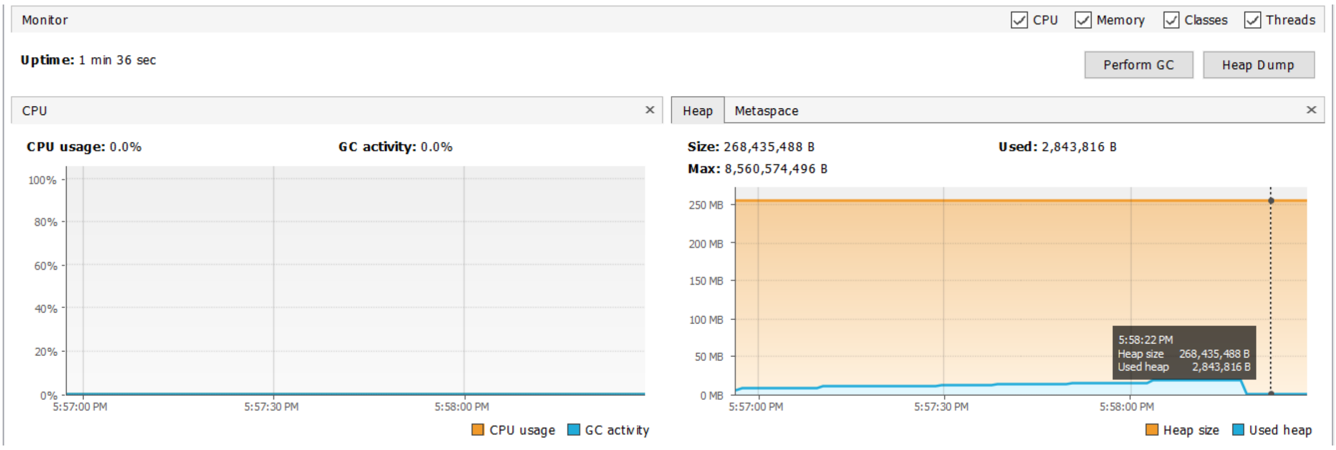


```
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM)

```
Adjacency list of vertex 16
16

The connected components are :
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


Does not contain cycle

find shortest path
enter the starting node :
3
enter the destination node :
7
Given source and destinationare not connected


K MOD GRAPH
Enter total number of nodes
```

MEMORY USAGE AND CPU TIME FOR EMPTY GRAPH

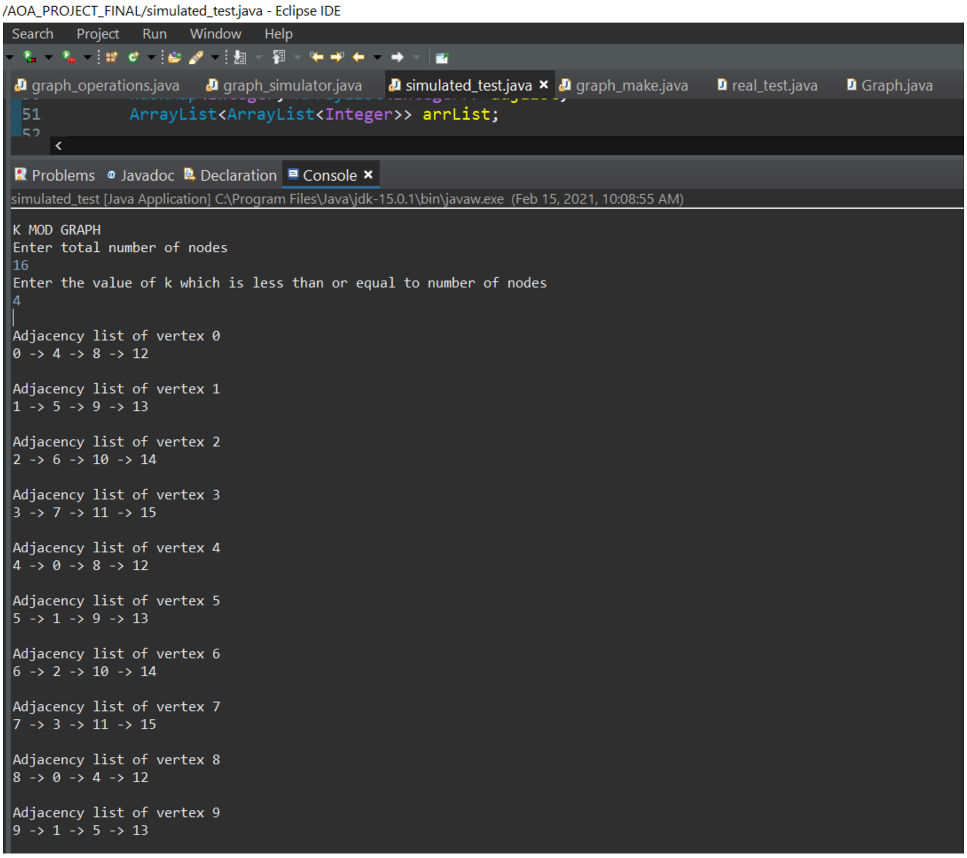


Uptime: 1 min 36 sec

CPU usage: 0.0% **GC activity:** 0.0%

Size: 268,435,488 B **Used:** 2,843,816 B
Max: 8,560,574,496 B

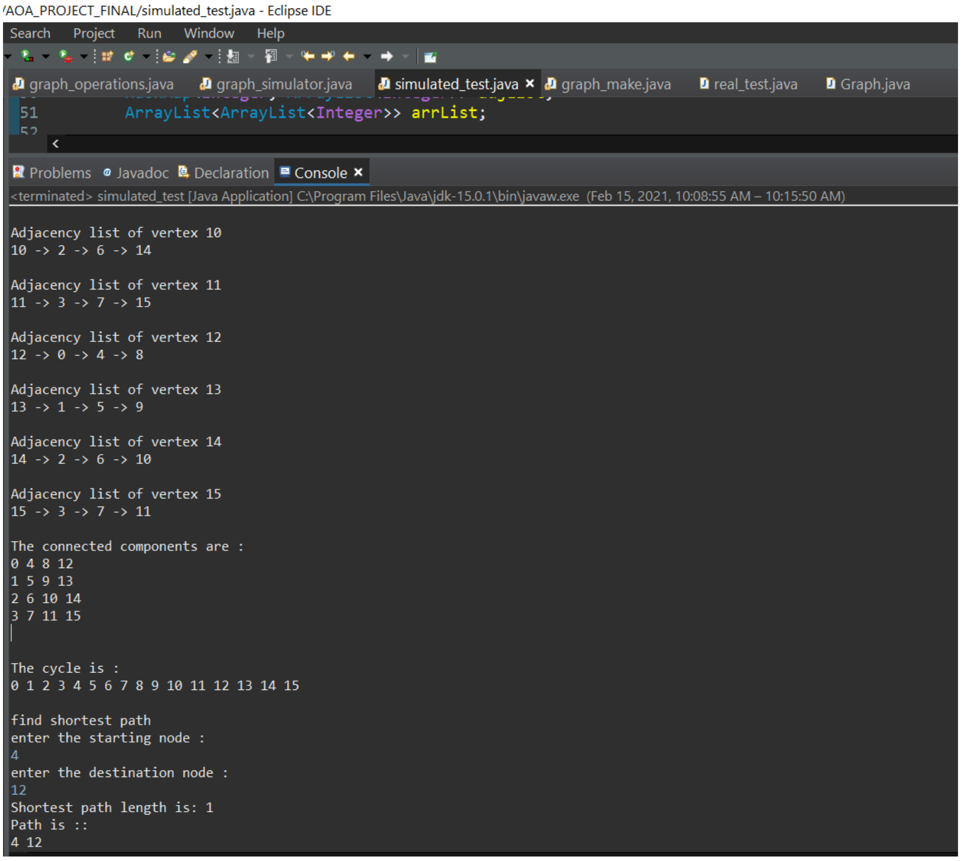
4) K-MOD GRAPH



```java
        ArrayList<ArrayList<Integer>> arrList;
```

simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM)

```
K MOD GRAPH
Enter total number of nodes
16
Enter the value of k which is less than or equal to number of nodes
4

Adjacency list of vertex 0
0 -> 4 -> 8 -> 12

Adjacency list of vertex 1
1 -> 5 -> 9 -> 13

Adjacency list of vertex 2
2 -> 6 -> 10 -> 14

Adjacency list of vertex 3
3 -> 7 -> 11 -> 15

Adjacency list of vertex 4
4 -> 0 -> 8 -> 12

Adjacency list of vertex 5
5 -> 1 -> 9 -> 13

Adjacency list of vertex 6
6 -> 2 -> 10 -> 14

Adjacency list of vertex 7
7 -> 3 -> 11 -> 15

Adjacency list of vertex 8
8 -> 0 -> 4 -> 12

Adjacency list of vertex 9
9 -> 1 -> 5 -> 13
```

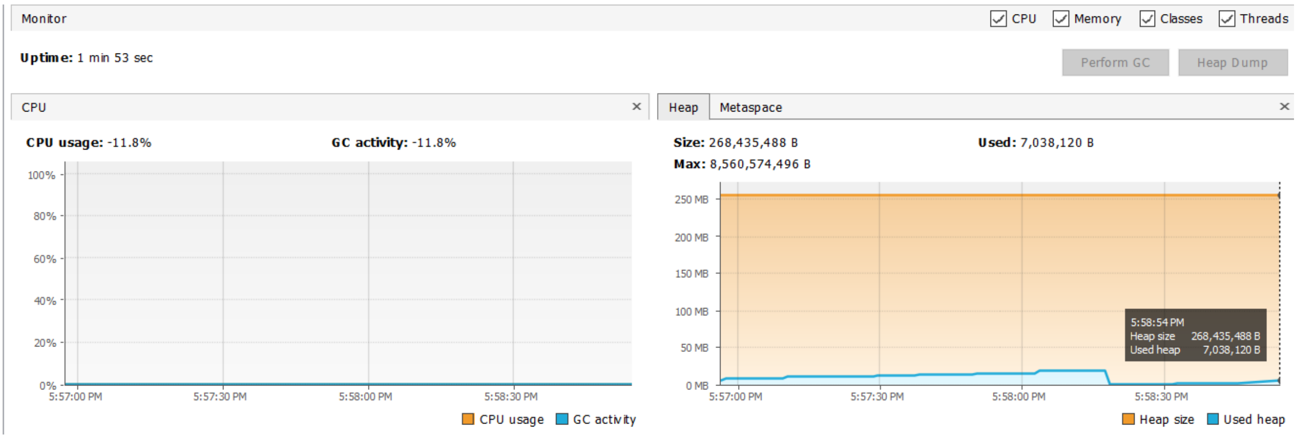


```java
        ArrayList<ArrayList<Integer>> arrList;
```

<terminated> simulated_test [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (Feb 15, 2021, 10:08:55 AM – 10:15:50 AM)

```
Adjacency list of vertex 10
10 -> 2 -> 6 -> 14

Adjacency list of vertex 11
11 -> 3 -> 7 -> 15

Adjacency list of vertex 12
12 -> 0 -> 4 -> 8

Adjacency list of vertex 13
13 -> 1 -> 5 -> 9

Adjacency list of vertex 14
14 -> 2 -> 6 -> 10

Adjacency list of vertex 15
15 -> 3 -> 7 -> 11

The connected components are :
0 4 8 12
1 5 9 13
2 6 10 14
3 7 11 15


The cycle is :
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

find shortest path
enter the starting node :
4
enter the destination node :
12
Shortest path length is: 1
Path is ::
4 12
```

MEMORY USAGE AND CPU TIME FOR K-MOD GRAPH



Uptime: 1 min 53 sec

CPU usage: -11.8%
GC activity: -11.8%

Size: 268,435,488 B
Max: 8,560,574,496 B
Used: 7,038,120 B

5:58:54 PM
Heap size 268,435,488 B
Used heap 7,038,120 B

The file graph_make.java contains the following functions.

10) filereader1 ()

This function first reads a file line by line and stores the result in **string data,** the integer part of the string is stored in a **substring id** which contains only the customer id in form of string **,** the **string id** is converted into **Integer ids ,** different values of ids and is added to **(Vector<integer> movie)** until all ids of **movie 1 is stored.** When **(:)** is read in then all the values of **(Vector <integer> movie)** is added to **HashMap<<Integer>, Vector <Integer>> adjList** and **Vector <integer> movie Is emptied** to store all the customer ids of movie 2 and the same process happens again.

This function makes a graph on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they have rated the same movie.

11) filereader2 ()

This function first reads a file line by line and stores the result in **string data,** the integer part of the string is stored in a **substring id** which contains only the customer id in form of string **,** the **string id** is converted into **Integer ids ,** different values of ids and is added to **(Vector<integer> movie)** until all ids of **movie 1 is stored.** When **(:)** is read in then all the values of **(Vector <integer> movie)** is added to **HashMap<<Integer>, Vector <Integer>> adjList** and **Vector <integer> movie Is emptied** to store all the customer ids of movie 2 and the same process happens again.

This function makes a graph on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they both have rated "4" the same movie.

12) filereader3 ()

This function first reads a file line by line and stores the result in **string data,** the integer part of the string is stored in a **substring id** which contains only the customer id in form of string **,** the **string id** is converted into **Integer ids ,** different values of ids and is added to **(Vector<integer> movie)** until all ids of **movie 1 is stored.** When **(:)** is read in then all the values of **(Vector <integer> movie)** is added to **HashMap<<Integer>, Vector <Integer>> adjList** and **Vector <integer> movie Is emptied** to store all the customer ids of movie 2 and the same process happens again.

This function makes a graph on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they both have rated "5" the same movie.

13) printGraph2 ()

This function has been updated for different criteria.

This function prints the current **Graph** and its edges.
HashMap<<Integer>, Vector <Integer>> adjList of size N

14) DepthFirstUntil () and connected_Components2 ()

This function has been updated for different criteria.

These functions use depth first search on the **graph HashMap<<Integer>, Vector <Integer>> adjList** to find and print all the connected components.

15) isCyclicUntil () and isCyclic2 ()

This function has been updated for different criteria.

These functions use depth first search on the **graph HashMap<<Integer>, Vector <Integer>> adjList** to detect presence of cycle and returns true if a cycle exists.

16) hashtoarr2 ()

This function has been updated for different criteria.

This function returns a takes **HashMap<<Integer>, Vector <Integer>> adjList** as parameter and creates a new **Graph Vector < Vector <Integer>> arrList** and copies all the vertices and edges from **adjList** to **arrList.**

17) BFS() and printShortestDistance2 ()

This function has been updated for different criteria.

These functions take source node and destination node from the user and returns the shortest distance and the path between the nodes using Dijkstra’s shortest path algorithm in the **graph HashMap<<Integer>, Vector <Integer>> adjList** , breadth first search is used to traverse the nodes of the graph.

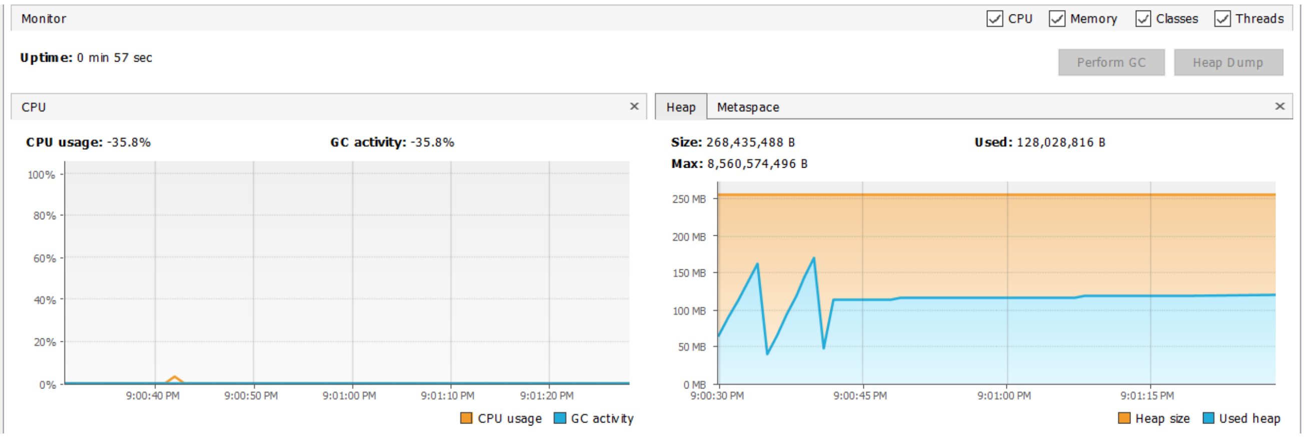
SYSTEM SPECIFICATION

OS : Windows 10 home
PROCESSOR : INTEL I7-10700F
RAM : 16 GB
Hard Drive : 500 GB(SSD) + 1000 GB(HDD)
Graphic Card : RTX 2060

GRAPH CRITERIA 1

The graph nodes are made on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they have rated the same movie.

MEMORY USAGE AND CPU TIME. (READING 5000 LINES)

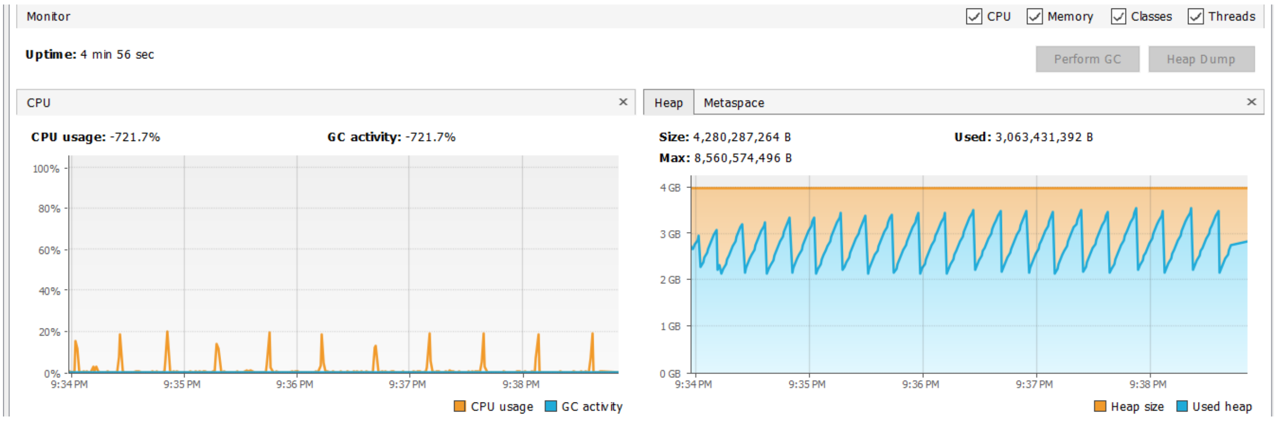


Uptime: 0 min 57 sec

CPU usage: -35.8%   GC activity: -35.8%

Size: 268,435,488 B   Used: 128,028,816 B
Max: 8,560,574,496 B

MEMORY USAGE AND CPU TIME. (READING 80000 LINES)



Uptime: 4 min 56 sec

CPU usage: -721.7%

GC activity: -721.7%

Size: 4,280,287,264 B

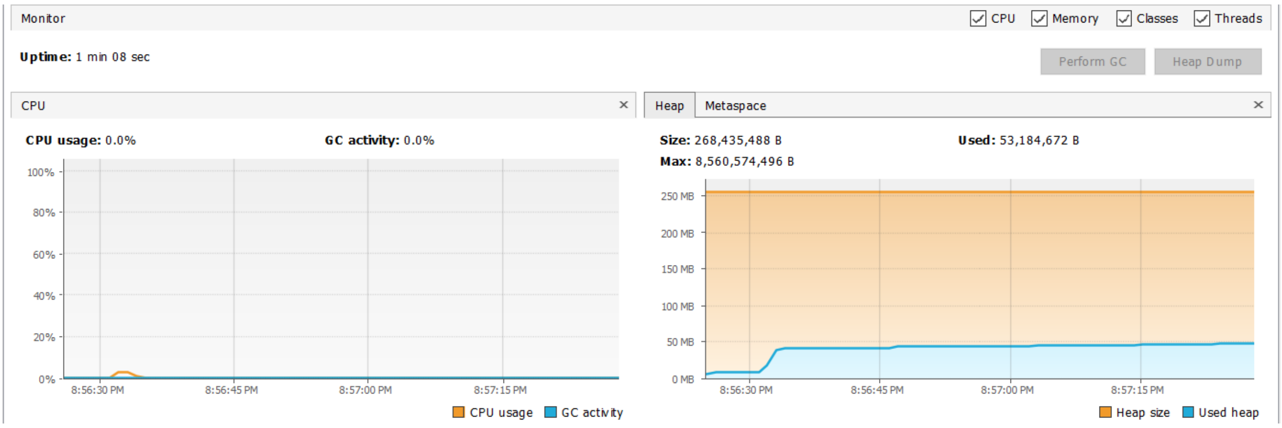
Used: 3,063,431,392 B

Max: 8,560,574,496 B

GRAPH CRITERIA 2

The graph nodes are made on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they have rated “4” to the same movie.

MEMORY USAGE AND CPU TIME. (READING 5000 LINES)



Uptime: 1 min 08 sec

CPU usage: 0.0%

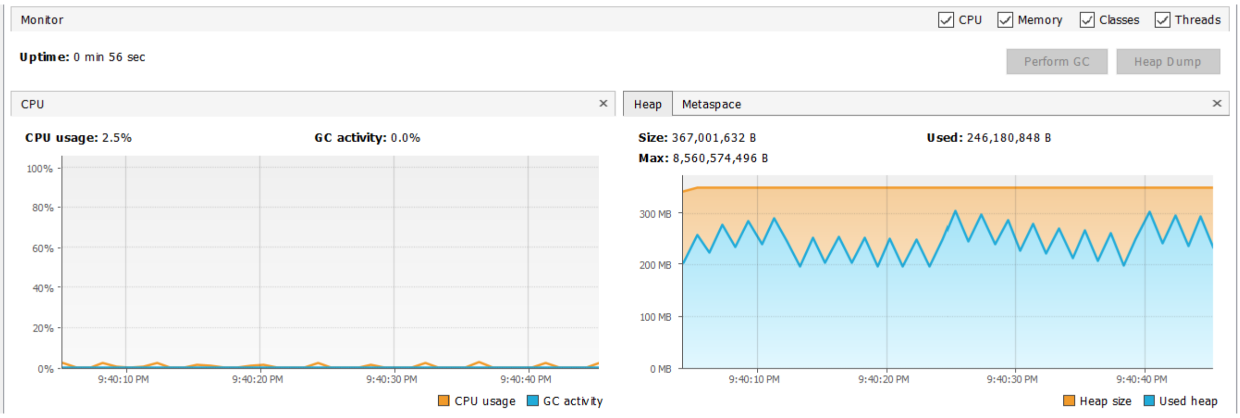
GC activity: 0.0%

Size: 268,435,488 B

Used: 53,184,672 B

Max: 8,560,574,496 B

MEMORY USAGE AND CPU TIME. (READING 80000 LINES)



Uptime: 0 min 56 sec

CPU usage: 2.5%

GC activity: 0.0%

Size: 367,001,632 B

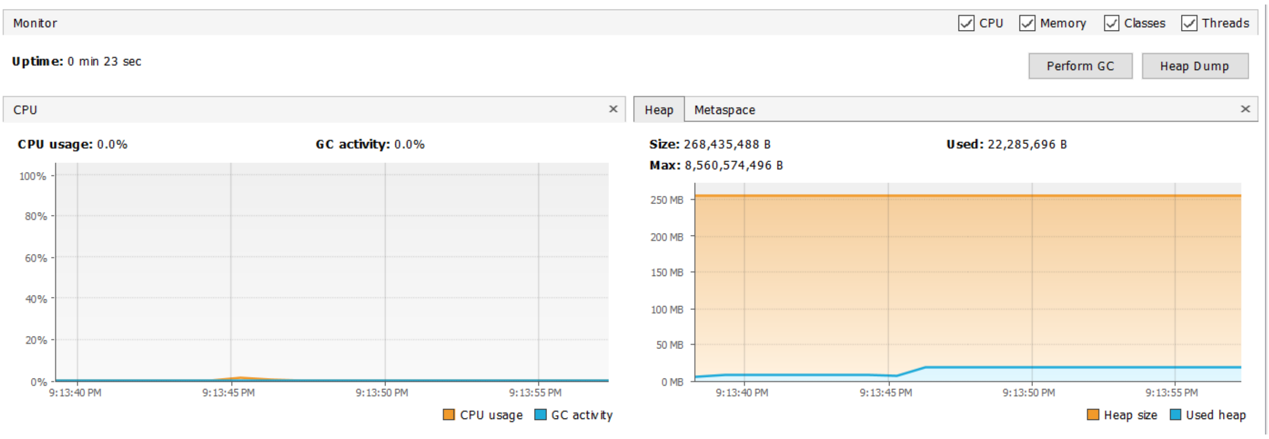
Used: 246,180,848 B

Max: 8,560,574,496 B

GRAPH CRITERIA 3

The graph nodes are made on the criteria that customer ID are taken as node and 2 nodes are connected with each other if they have rated “5” to the same movie.

MEMORY USAGE AND CPU TIME. (READING 5000 LINES)

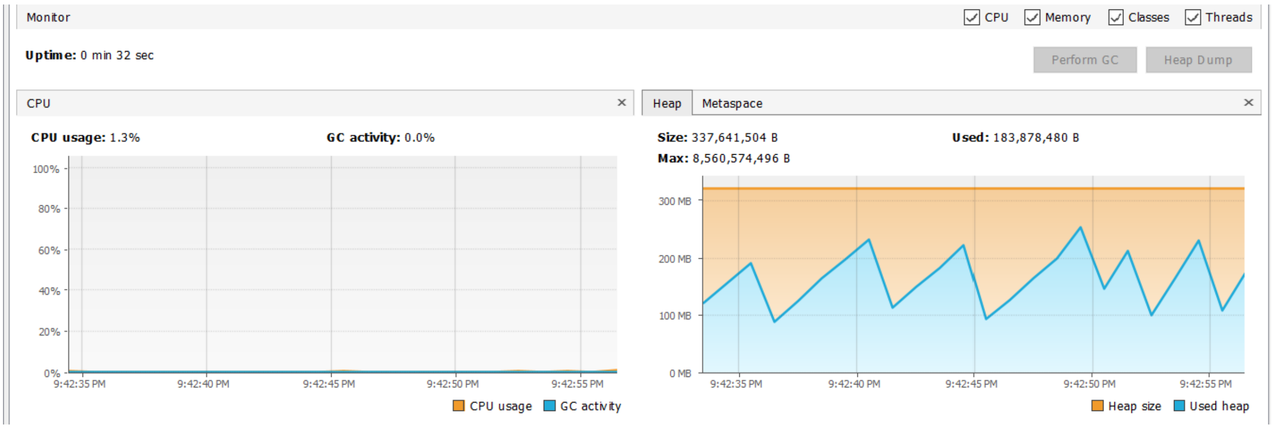


Uptime: 0 min 23 sec

CPU usage: 0.0% **GC activity:** 0.0%

Size: 268,435,488 B **Used:** 22,285,696 B
Max: 8,560,574,496 B

MEMORY USAGE AND CPU TIME. (READING 80000 LINES)



Uptime: 0 min 32 sec

CPU usage: 1.3% **GC activity:** 0.0%

Size: 337,641,504 B **Used:** 183,878,480 B
Max: 8,560,574,496 B