Search for connected components

**Actual Output for udGraphs.txt**

\*\* G1's connected components:

Breadth First Search:

0 2 7 8 3 6 9

1

4 5

Depth First Search:

0 2 8 3 6 9 7

1

4 5

\*\* G2's connected components:

Breadth First Search:

0 7 8

1 5 2

3 6

4

9

Depth First Search:

0 7 8

1 5 2

3 6

4

9

\*\* G3's connected components:

Breadth First Search:

0 4 8

1 2 3 6 9 5 7

Depth First Search:

0 4 8

1 2 3 5 7 6 9

\*\* G4's connected components:

Breadth First Search:

0 5

1 3 4 8 2 6 9 7

Depth First Search:

0 5

1 3 2 6 8 9 7 4

\*\* G5's connected components:

Breadth First Search:

0

1 6 8

2

3 4 7 5 9

Depth First Search:

0

1 6 8

2

3 4 7 5 9

\*\* G6's connected components:

Breadth First Search:

0 1 7 10 13 14 18 4 8 9 17 5 11 15 2 12 16 3 19

6

Depth First Search:

0 1 4 5 2 10 7 9 11 12 3 14 18 15 19 16 13 8 17

6

\*\* G7's connected components:

Breadth First Search:

0 3 13 14 10 18 2 5 11 1 7 17 19 4 15

6

8

9

12 16

Depth First Search:

0 3 13 10 1 2 7 4 15 14 5 19 11 18 17

6

8

9

12 16

\*\* G8's connected components:

Breadth First Search:

0 16 18

1 6 7 10 17 2 11 14

3 12

4 13 15 5 8 9

19

Depth First Search:

0 16 18

1 6 17 2 7 10 11 14

3 12

4 13 5 9 15 8

19

\*\* G9's connected components:

Breadth First Search:

0 5 8 19 15 1 11

2 4 13 6 14 16 10

3 7 9 18

12

17

Depth First Search:

0 5 15 8 19 1 11

2 4 6 10 13 14 16

3 7 9 18

12

17

\*\* G10's connected components:

Breadth First Search:

0 11 15 18

1 2 16 19 3 4 17 8 6 13

5

7 10 12 14 9

Depth First Search:

0 11 15 18

1 2 3 4 6 13 8 19 17 16

5

7 10 9 14 12

\*\* G11's connected components:

Breadth First Search:

0 1 2 6 7 10 11 13 14 15 16 18 19 20 22 25 26 27 28 29 3 4 5 8 12 21 24 17 9 23

Depth First Search:

0 1 2 4 3 8 7 5 9 6 10 11 13 14 17 12 16 20 22 18 15 21 19 25 24 26 28 29 27 23

\*\* G12's connected components:

Breadth First Search:

0 1 7 8 18 23 24 27 29 6 10 21 11 17 14 15 19 20 5 22 13 16

2

3 9 12 26

4 25

28

Depth First Search:

0 1 23 15 6 7 10 19 20 27 21 5 17 8 11 13 16 22 14 18 29 24

2

3 9 12 26

4 25

28

\*\* G13's connected components:

Breadth First Search:

0 1 2 3 4 5 6 7 9 11 13 14 15 17 21 22 24 26 27 28 29 10 16 19 23 12 18 25 20 8

Depth First Search:

0 1 2 5 3 6 7 4 10 9 12 15 11 13 14 16 17 18 22 20 23 19 25 26 24 27 21 29 28 8

\*\* G14's connected components:

Breadth First Search:

0 19 21 27 28 8 15 20 26 3

1 2 22

4 10 11 13 17 29 5 14 9 23 6 12 24 18 7 16

25

Depth First Search:

0 19 8 3 15 21 27 20 26 28

1 2 22

4 10 5 6 9 11 23 7 16 12 14 13 24 29 17 18

25

\*\* G15's connected components:

Breadth First Search:

0 3 5 7 8 9 12 13 14 15 17 18 21 23 24 25 29 1 4 6 19 26 27 10 22 28 2 11 16 20

Depth First Search:

0 3 1 2 4 6 5 7 8 10 9 16 12 14 11 15 13 17 21 19 18 20 24 22 23 25 26 29 28 27

\*\* G16's connected components:

Breadth First Search:

0 1 6 23 34 39 12 21 22 24 26 27 36 5 32 35 11 16 25 17 28 37 8 13 20 29 18 9 10 31

2 7 33 38

3 4 15 30

14 19

Depth First Search:

0 1 12 8 11 10 13 17 16 20 26 22 23 6 5 18 21 27 24 9 25 28 29 32 31 34 37 36 39 35

2 7 33 38

3 4 15 30

14 19

\*\* G17's connected components:

Breadth First Search:

0 9 18 32 34 2 7 22 28 35 39 16 20 36 30 31 38

1 12 14 21 26 5 24

3 11 17 27 29 33 6 13 8 15 37

4 23 25

10 19

Depth First Search:

0 9 2 7 16 22 30 31 28 20 35 18 34 32 39 36 38

1 12 5 21 24 14 26

3 11 6 8 15 13 29 17 27 33 37

4 23 25

10 19

\*\* G18's connected components:

Breadth First Search:

0 6 18 30

1 2 5 22 27 35

3 8 20 23 24 31 32 36 38 4 9 15 17 25 28 19 12 14 29 39 11 21 13

7 10 16 26 33 34 37

Depth First Search:

0 6 18 30

1 2 5 22 27 35

3 8 4 11 14 12 9 17 15 13 23 32 28 19 20 38 24 36 25 21 29 31 39

7 10 16 26 33 34 37

\*\* G19's connected components:

Breadth First Search:

0 2 11 12 26 37 1 19 20 23 27 30 36 9 24 31 33 25 29 39 5 6 35 14 18 15

3 10 16 22 28 38

4 7 13 17 21 32 34

8

Depth First Search:

0 2 1 6 5 9 11 24 23 12 25 26 19 20 37 14 15 31 30 18 36 27 33 29 35 39

3 10 16 22 28 38

4 7 13 17 21 32 34

8

\*\* G20's connected components:

Breadth First Search:

0 12 30 33 18 21 24 25 26 38 4 5 10 13 20 28 2 3 34 32 37 6 29

1 7 11 17 23 27 36 39

8 9 15 16 19 22 31 35 14

Depth First Search:

0 12 18 10 2 24 5 4 25 26 21 3 29 32 28 13 6 37 20 30 34 33 38

1 7 23 11 17 27 36 39

8 9 15 14 16 19 22 35 31

**Summary**

There are 4 helper methods in the program. They are addEdge, DFSRecursive, connectedComponentsDFS and connectedComponentsBFS. Purpose of the addEdge method is to add an edge between given 2 vertices. Since this graph is undirected, the given edge will be added to adjacency lists of both vertices.

DFSRecursive takes in a starting vertex and a boolean array to keep track of visited vertices. It marks the current vertex as visited, prints its value, and recursively calls itself for all adjacent unvisited vertices.

connectedComponentsDFS function initialize an array to keep track of visited vertices and traverses all vertices, calling the DFSRecursive function for unvisited vertices. It prints the values of all vertices in each connected component on a new line and repeats the process for any remaining unvisited vertices until all vertices have been visited.

connectedComponentsBFS initializes an array to keep track of visited vertices and traverses all vertices, follow the BFS algorithm for unvisited vertices. For each unvisited vertex, it creates a queue and performs BFS starting from that vertex. It prints the values of all vertices in each connected component on a new line, and repeats the process for any remaining unvisited vertices until all vertices have been visited.

**Algorithm**

**BFS Algorithm**

* Create a boolean array visited to keep track of visited vertices and a queue q to store vertices to visit.
* Start with a source vertex and mark it as visited and enqueue it to q.
* While q is not empty, dequeue the front vertex and print its value to the console.
* For each unvisited neighbor of the dequeued vertex, mark it as visited and enqueue it to q.
* Repeat steps 3-4 until q is empty.

**DFS Algorithm**

* Initialize a boolean array to keep track of visited vertices
* Traverse all vertices in the graph and for each unvisited vertex, perform a DFS starting from that vertex to find its connected component.
* Call the DFSRecursive function on the unvisited vertex and pass the visited array as an argument.
* The DFSRecursive function marks the current vertex as visited, prints its value, and recursively calls itself for all unvisited neighbors of the current vertex.
* Repeat the above steps for any remaining unvisited vertices until all vertices have been visited.

The graph is represented by an adjacency list. ArrayList of ArrayList is used for that. Integer Array was used to store the already visited vertices. Queue is used to store the unvisited vertices that will be visited in future.

**Time Complexity and Space Complexity**

Both DFS and BFS have the same time complexity for finding connected components in a graph, but their space complexities differ.

DFS is a graph traversal algorithm that explores the graph as deep as possible before backtracking to find more unexplored vertices. The time complexity of DFS for finding connected components is O(V+E), where V is the number of vertices and E is the number of edges in the graph. The space complexity of DFS is also O(V) as it uses a stack to keep track of visited nodes.

BFS, on the other hand, explores the graph layer by layer, starting from the source node. The time complexity of BFS for finding connected components is also O(V+E). However, the space complexity of BFS is higher than DFS, as it uses a queue to keep track of the visited nodes. The space complexity of BFS is O(V+E).