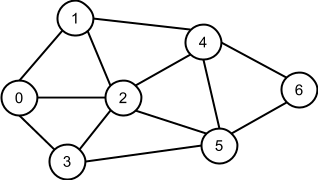
Assignment 3: Graph Search/Traversal

Implement the Breadth First Search (BFS) and Depth First Search (DFS) techniques for traversing the nodes/vertices of a given graph. Use BFS to find *shortest path lengths* in an unweighted graph*.* Use Depth First Traversal to find the *number of connected components* of a graph that may be disconnected.

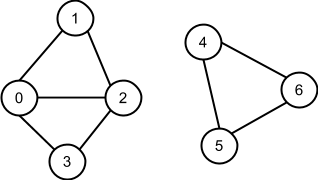
# 

# Connected components of a Graph:

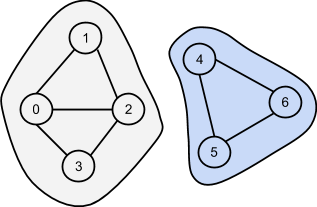
Consider the following graph.



The graph has 7 nodes and 12 edges and is *connected*. Because for any node u, you can reach any other node v of the graph, just following the edges of the graph. (Try it for node labeled as 2). So the graph has only 1 connected component. Now lets delete some edges of the graph to turn in to the following

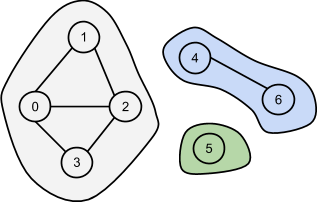


Now the graph has 8 edges (no change in number of nodes). But this time you can’t tell that the graph is connected. See if you can reach 4 from 2 following the edges of the graph. You can’t. However consider the highlighted portions of the graph



Nodes inside each highlighted portion can be reached from each other. So each highlighted portion is a connected component. So there are two now. What? *Connected Components*.

Now lets delete the edges 4 - 5 and 4 - 6. The following graph is found with the connected components highlighted.

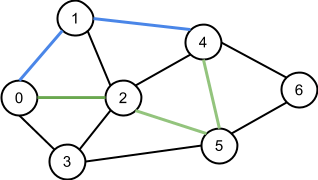


Tell how many connected components are there.

|  |
| --- |
| **Task 1**  Find the *number of connected components* in a given graph using Depth First Traversal. |

# Shortest Path in an unweighted graph:

Consider the following graph again. You want to go from node 0 to node 4. How many edges you have to use?



If you follow the path 0 → 1 → 4, 2 edges are needed (path in blue). On the other hand 4 can also be reached from 0 by following the path of 0 → 2 → 5 → 4(path in green), which includes 3 edges. Which one is shorter in terms of number of edges? The blue one. Try to find another path that uses less than 2 edges to reach from 0 to 4. There is none. So the path in blue is the shortest path from 0 to 4. (Find another path from 0 to 4 that uses only 2 edges.)

This way we can determine the shortest path from 0 to other nodes of the graph. We are starting all of our paths at 0, so 0 is our *source node*. Other nodes are named as *destination node*. Following table shows the shortest paths with 0 as source node.

|  |  |  |
| --- | --- | --- |
| Destination Node | Path[[1]](#footnote-1) | Number of edges |
| 1 | 0 → 1 | 1 |
| 2 | 0 → 2 | 1 |
| 3 | 0 → 3 | 1 |
| 4 | 0 → 1 → 4 | 2 |
| 5 | 0 → 2 → 5 | 2 |
| 6 | 0 → 3 → 5 → 6 | 3 |

Now take 2 as a *source node* and prepare the above table. (Use pencil and paper)

|  |
| --- |
| **Task 2**  Given a source node and a graph, identify the *lengths of shortest paths* from the source node to other nodes of the graph using Breadth First Search. |

# Direction

Create two classes, named as BFS and DFS. The class named BFS will contain a static method named lengths which will take an integer as parameter and an array of ArrayList. This array will contain the *adjacency list* representation of the input graph. The method should return an integer array which will contain the lengths of shortest paths.

In DFS class, create a static method named count that will take a 2-d array of integers as parameter. This is the *adjacency matrix* representation of the input graph. The method should return an integer indicating how many connected components are there.

The nodes will be represented using integers ranging from 0 to n-1 for a graph containing n nodes. If you are planning to implement non-recursive DFS, you can use Stack class from Java SE. For BFS, you can do the following to create a Queue

|  |
| --- |
| Queue<Integer> queue = new LinkedList<Integer>();  queue.add(10); /\* enqueuing 10 \*/  int number = queue.poll(); /\* dequeuing from the queue \*/ |

As earlier, you can add any method you feel needed to those classes. Number of nodes in the graphs will be no more than 1000. Calculate the maximum number of edges possible in a graph with that many nodes. That is the limit for number of edges.

# Way to solutions

Follow the class lectures and resources listed in the Reading Materials section at the end of the document.

# Sample Input/Output

Some set of input and output are given here. Format is explained with the example below.

|  |  |
| --- | --- |
|  |  |

The first line of the input will contain 2 integers. They are number of nodes and number of edges respectively. Following that there will be as many lines as there are edges. Each line will contain the two associated nodes of the corresponding edge.

Sample Inputs are given separately for Task 1 and 2 below.

### Sample Input for Task 1:

**Set 1:**

|  |
| --- |
| 6 7  0 1  0 3  1 2  2 5  1 4  4 5  3 5 |

**Set 2:**

|  |
| --- |
| 13 13 4 3 9 12 6 4 5 4 0 2 11 12 9 10 9 11 5 3  1 11  1 8  4 7  8 10 |

### 

**Set 3:**

|  |
| --- |
| 5 0 |

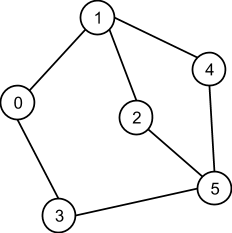
### 

### Sample Output for Task 1:

|  |
| --- |
| **For Set 1:** 1  **For Set 2:** 3  **For Set 3:** 5 |

### 

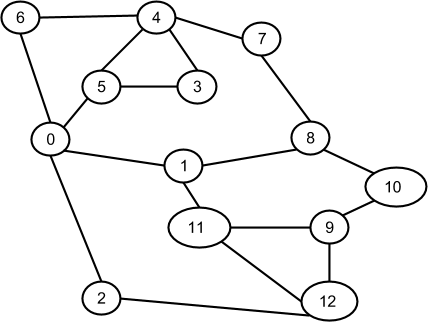
### Sample Input for Task 2:

**Set 1:**

|  |
| --- |
| 6 7  0 1  0 3  1 2  2 5  1 4  4 5  3 5 |

**Set 2:**

|  |
| --- |
| 13 18 0 5 4 3 0 1 9 12 6 4 5 4 0 2 11 12 9 10 0 6 7 8 9 11 5 3  1 11  1 8  2 12  4 7  8 10 |



### Sample output for Task 2:

|  |
| --- |
| (Taking 0 as source node)  **For Set 1:** 0 1 2 1 2 2  **For Set 2:** 0 1 1 2 2 1 1 3 2 3 3 2 2  (Taking 4 as source node)  **For Set 1:** 2 1 2 2 0 1  **For Set 2:** 2 3 3 1 0 1 1 1 2 4 3 4 4 |

# 

# Evaluation

Submission will be evaluated using following criteria (with distribution of marks)

|  |  |
| --- | --- |
| **Criteria** | **Marks** |
| Created necessary java files (according to directions) | 10 |
| Implementation of Depth First Search | 15 |
| Implementation of Breadth First Search | 15 |
| Solving Task 1 | 30 |
| Solving Task 2 | 30 |
| **Total** | **100** |

**For Incorrect Implementations:**

|  |  |
| --- | --- |
| Implementation of Depth First Search \* | 0-10 |
| Implementation of Breadth First Search \* | 0-10 |
| Solving Task 1 \* | 5-25 |
| Solving Task 2 \* | 5-25 |

\* Marking depends on evaluator.

***Notes on copying other’s code: If anybody found copying solution code from other student(s), all of them will be penalized. Penalty includes***

* ***Assigning ZERO as mark for the solution submitted***
* ***Assigning ZERO as mark for the best submission among all other submissions. (assigned at the end of semester)***

***Copying solutions from the Internet will also incur similar penalties.***

## 

# Submission & Contact

|  |  |
| --- | --- |
| **Submission Type** | Individual |
| **Submission Deadline (Full Credit):** | June 8, 2014 |

Students are encouraged to use Google Group for communicating to resolve their issues (instead of personal mails.)

Group Address: <https://groups.google.com/forum/#!forum/algorithmfiesta2014>

# Reading Materials

* *Computer Algorithms*, By Ellis Horowitz , Sartaj Sahni, Sanguthevar Rajasekaran. ISBN: 9780929306414 **Page-320**
* *Introduction to Algorithms*, By Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein. ISBN: 9780262033848 **Page-547**
* <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/GraphAlgor/breadthSearch.htm>

1. There may be different paths from a *source node* to a *destination node,* each having same length. For simplicity, only one of them is listed in the Table. [↑](#footnote-ref-1)