COMP225: Exercises on Week 9 Material

version 1.00

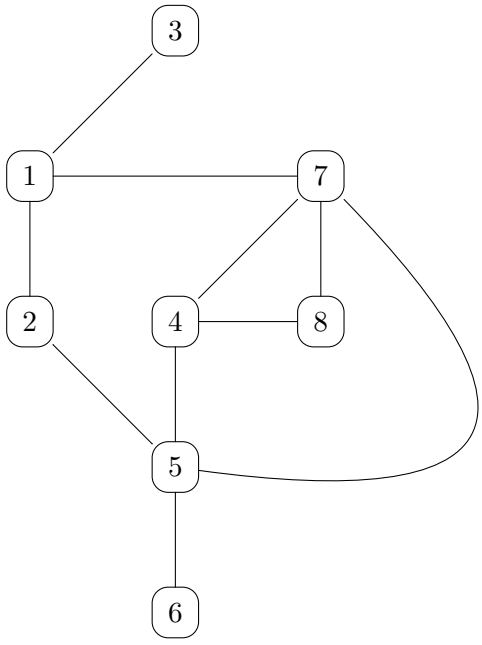
Mark Dras

May 11, 2019

For most of these exercises, you'll be using the graph code I introduced in lectures this week, available in the Week 9 code bundle.

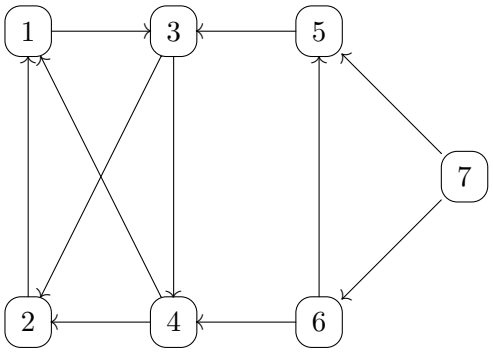
Submission exercises will be marked **Submission** (found at the end of this week's questions) and participation exercises will be marked **Participation**. If you finish all the exercises marked **Participation** and there is still time left in the tutorial please do have a go at a few more exercises.

1. **Participation** Trace the order of traversal of the nodes in the following undirected graph given a **depth**-first traversal starting from node 1. (Assume an ascending numeric ordering on the nodes.)



This graph is the one defined in the `setDefault()` method of class `Graph`, so you can check your answer by running the DFT code in class `GraphApplic`.

2. **Participation** Trace the order of traversal of the nodes in the following directed graph given a **depth**-first traversal starting from node **7**. (Assume an ascending numeric ordering on the nodes.)



3. **Participation** The code in class `GraphApplic` for a depth-first traversal is only for a single connected component (corresponding to `dfs()` in the lecture notes, slide 34). Write a Java method in class `GraphApplic` to apply this to the whole of a (not necessarily connected) graph (corresponding to `depthFirstSearch()` in the lecture notes, slide 34).

   You might find it useful to define in class `Graph` methods for indicating whether all vertices in the graph are marked as having been visited, and for returning the first unmarked vertex in the case where any exist.

4. **Participation** Write the following Java method in class `GraphApplic` as a modification of a depth-first traversal (just the recursive method for a single connected component):

```
public Integer depthFirstTraversalLastVertex(Integer v)
// PRE: Graph is connected, v is the id of a vertex in the graph
// POST: returns the last vertex encountered during a
// depth-first traversal
```

5. **Submission** Write the following Java method in class `GraphApplic` as a modification of a depth-first traversal (just the recursive method for a single connected component):

```
public Integer numVerticesGtrDegreeK(Integer v, Integer k);
// PRE: Graph is connected, v is the id of a vertex in the graph
// POST: returns the number of nodes that are of greater than
// or equal to degree k
```

6. Define a DEPTH-FIRST TRAVERSAL PATH (DFTP) as a path formed by the sequence of vertices placed on the stack (either the explicit stack of the iterative version, or the stack of recursive calls in the recursive version) during a DFT. For the graph in Question 1, 1-2-5-4-7-8, 4-7-8, 1-2-5-6, 2-5-6, and 1-3 are all DFTPs; 1-7-4 is not a DFTP.

   Define a MAXIMAL DFTP (MDFTP) as a DFTP that is not a subsequence of any other DFTP. For the graph in Question 1, 1-2-5-4-7-8, 1-2-5-6, and 1-3 are the only three MDFTPs; all other DFTPs are subsequences of these. In terms of the DFT code, the vertices in a MDFTP are the values on the stack at the point where there are no

unvisited neighbours, and the only option is to pop that vertex off the stack (or that recursive method call off the recursion stack).

Write the following Java method in class `GraphApplic` as a modification of a depth-first traversal (either the recursive version discussed in class, or the iterative version available in the code bundle):

```
public Integer depthFirstTraversalMaxPath(Integer v)
// PRE: Graph is connected, v is the id of a vertex in the graph
// POST: returns the length of the longest MDFTP in the graph
```

For the graph discussed above, the length of the longest MDFTP is 6 (for the MDFTP 1-2-5-4-7-8).

Hint: if you modify the recursive DFT, it will probably be useful to have a second parameter to the function, to keep track of the depth of recursion, i.e.

```
public Integer depthFirstTraversalMaxPath(Integer v, Integer count)
// PRE: Graph is connected, v is the id of a vertex in the graph,
// count a non-negative value representing depth of recursion
// POST: returns the length of the longest MDFTP in the graph
```

7. You'll see that the `Vertex` class contains an attribute `num`, as in the Drozdek code, which can be used for storing within a vertex the order that that vertex is visited during a traversal. In the graph in Question 1, where the DFT starts from vertex 1, vertex 1's num attribute will have the value 1, vertex 5's num attribute will have the value 3, and so on. There are also get and set methods for the attribute.

   Now modify the DFT so that the order of traversal is stored in the vertex. When you print out the graph after a DFT on the graph of Question 1, the output should then look something like:

```
Number of nodes is 8
Number of edges is 10
Node 1 (true,1) : 2 3 7
Node 2 (true,2) : 1 5
Node 3 (true,8) : 1
Node 4 (true,4) : 5 7 8
Node 5 (true,3) : 2 4 6 7
Node 6 (true,7) : 5
Node 7 (true,5) : 1 4 5 8
Node 8 (true,6) : 4 7
```

   The integer in parentheses is the order that that vertex is visited during the DFT.