Week 4 Assessment Questions

Graph Data Structures and Graph Search

1. (Graph properties)

   a. Write pseudocode for computing

   - the minimum and maximum vertex degree
   - all 3-cliques (i.e. cliques of 3 nodes, "triangles")

   of a graph *g* with *n* vertices.

   Your methods should be representation-independent; the only function you should use is to check if two vertices $v,w \in \{0,\ldots,n-1\}$ are adjacent in *g*.

   b. Determine the asymptotic complexity of your two algorithms. Assume that the adjacency check is performed in constant time, *O(1)*.

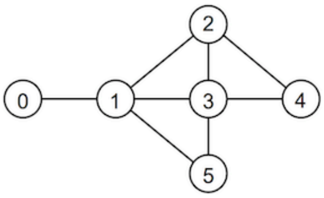
   c. Implement your algorithms in a program `graphAnalyser.c` that

   1. builds a graph from user input:
      - first, the user is prompted for the number of vertices
      - then, the user is repeatedly asked to input an egde by entering a "from" vertex followed by a "to" vertex
      - until any non-numeric character(s) are entered
   2. computes and outputs the minimum and maximum degree of vertices in the graph
   3. prints all vertices of minimum degree in ascending order, followed by all vertices of maximum degree in ascending order
   4. displays all 3-cliques of the graph in ascending order.

   Your program should use the Graph ADT from the lecture (`Graph.h` and `Graph.c`). These files should not be changed.

   Hint: You may assume that the graph has a maximum of 1000 nodes.

   An example of the program executing is shown below for the graph




```
prompt$ ./graphAnalyser
Enter the number of vertices: 6
Enter an edge (from): 0
Enter an edge (to): 1
Enter an edge (from): 1
Enter an edge (to): 2
Enter an edge (from): 4
Enter an edge (to): 2
Enter an edge (from): 1
Enter an edge (to): 3
Enter an edge (from): 3
Enter an edge (to): 4
Enter an edge (from): 1
Enter an edge (to): 5
Enter an edge (from): 5
Enter an edge (to): 3
Enter an edge (from): 2
Enter an edge (to): 3
Enter an edge (from): done
Done.
Minimum degree: 1
Maximum degree: 4
Nodes of minimum degree:
0
Nodes of maximum degree:
1
3
Triangles:
1-2-3
1-3-5
2-3-4
```

   Note that any non-numeric data can be used to 'finish' the interaction.

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `graphAnalyser.c` *in the current directory. You can use dryrun as follows:*

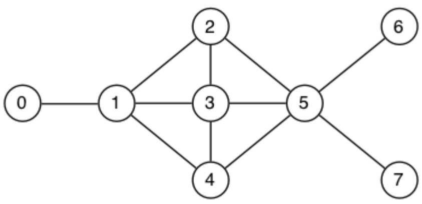
```
prompt$ 9024 dryrun graphAnalyser
```

Please ensure that your program output follows exactly the format shown in the sample interaction above. In particular, the vertices of minimum and maximum degree and the 3-cliques should be printed in ascending order.

2. (Graph traversal: DFS and BFS)

Both DFS and BFS can be used for a *complete* search without a specific destination, which means traversing a graph until all reachable nodes have been visited. Show the order in which the nodes of the graph depicted below are visited by

   a. DFS starting at node 6
   b. DFS starting at node 2
   c. BFS starting at node 2
   d. BFS starting at node 5



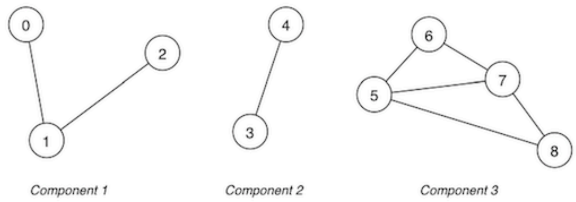
Assume the use of a stack for depth-first search (DFS) and a queue for breadth-first search (BFS), respectively, and use the pseudocode from the lecture: DFS, BFS. Show the state of the stack or queue explicitly in each step. When choosing which neighbour to visit next, always choose the smallest unvisited neighbour.

3. (Cycle check)

  a. Take the "buggy" cycle check from the lecture and design a correct algorithm, in pseudocode, to use depth-first search to determine if a graph has a cycle.

  b. Write a C program `cycleCheck.c` that implements your solution to check whether a graph has a cycle. The graph should be built from user input in the same way as in exercise 2. Your program should use the Graph ADT from the lecture (`Graph.h` and `Graph.c`). These files should not be changed.

  An example of the program executing is shown below for the following graph:




```
prompt$ ./cycleCheck
Enter the number of vertices: 9
Enter an edge (from): 0
Enter an edge (to): 1
Enter an edge (from): 1
Enter an edge (to): 2
Enter an edge (from): 4
Enter an edge (to): 3
Enter an edge (from): 6
Enter an edge (to): 5
Enter an edge (from): 6
Enter an edge (to): 7
Enter an edge (from): 5
Enter an edge (to): 7
Enter an edge (from): 5
Enter an edge (to): 8
Enter an edge (from): 7
Enter an edge (to): 8
Enter an edge (from): done
Done.
The graph has a cycle.
```

If the graph has no cycle, then the output should be:

```
prompt$ ./cycleCheck
Enter the number of vertices: 3
Enter an edge (from): 0
Enter an edge (to): 1
Enter an edge (from): #
Done.
The graph is acyclic.
```

You may assume that a graph has a maximum of 1000 nodes.

To test your program you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a program named* `cycleCheck.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun cycleCheck
```

Note: Please ensure that your output follows exactly the format shown above.

Assessment

After you've solved the exercises, go to COMP9024 20T2 Quiz Week 4 to answer 5 quiz questions on this week's assessment questions and lecture.

The quiz is worth 2 marks.

The deadline for submitting your quiz answers is **Tuesday, 30 June 11:00:00am**.

Please continue to respect the **quiz rules**:

Do …

- use your own best judgement to understand & solve a question
- discuss quizzes on the forum only **after** the deadline on Tuesday

Do not …

- post specific questions about the quiz **before** the Tuesday deadline
- agonise too much about a question that you find too difficult