

1 Instructions

When you are finished submit all your work through the MyClasses page for this class. Create a directory called Project3, put the program in this directory, compress the directory into a single zip file, and then submit this zip file to the Project #3 assignment. Also remember to create make files for each program and make sure the programs compile and run on the lab Linux system.

Projects are to be done strictly on your own and as with all assignments the sharing of files and code is strictly prohibited and constitutes an act of Academic Misconduct. Furthermore the use of any electronic medium, such as code repositories, forums, blogs, message boards, email, etc. is strictly prohibited and constitutes an act of Academic Misconduct.

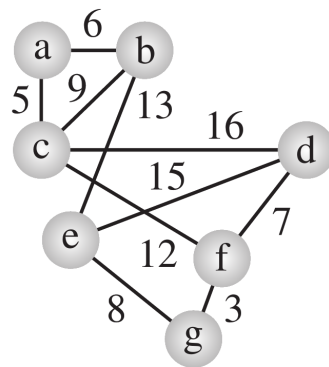
You may use the course textbooks, class notes, and materials I have provided on the MyClasses page.

2 Programming Exercises

This set of exercises is to continue building a library of basic graph algorithms that can be used for graphs that are represented as adjacency matrices. You started a set of these with the last lab by constructing functions that can do the depth first search and the breadth first search on graphs with the adjacency matrix representation. Include these in your library. Add to this library the following six functions, use the list of lists data structure for these implementations as you did for the search algorithms.

1. Dijkstra's Algorithm for finding the shortest path.
2. Ford's Algorithm for finding the shortest path.
3. Cycle Detection
4. Kruskal's Algorithm to find a minimal spanning tree of a connected undirected graph.
5. On page 482 of the main text for the class (attached below) the author gives two algorithms, in pseudo-code form, for finding the minimum spanning tree for weighted connected undirected graphs. One algorithm is due to Otakar Boruvka and the other (separately) by Vojtech Jarnik and Robert Prim. Add each of these to your library.

These functions should give output similar to the ones we constructed in class. Also have the main allow you to open a text file containing a matrix representation of a weighted graph and then apply these algorithms to it. For example, the the following graph,



The adjacency matrix would look like the following. This is also how it would be stored in the text file.

0	6	5	0	0	0	0
6	0	9	0	13	0	0
5	9	0	16	0	12	0
0	0	16	0	15	7	0
0	13	0	15	0	0	8
0	0	12	7	0	0	3
0	0	0	0	8	3	0

Opening a graph file should be done using the tinyfiledialogs dialog boxes. Assume that the file containing the graph will have a txt extension.

25. How can the algorithms for finding the minimum spanning tree be used to find the maximum spanning tree?
26. Apply the following two algorithms to find the minimum spanning tree to the graph in Figure 8.15a.
 - a. Probably the first algorithm for finding the minimum spanning tree was devised in 1926 by Otakar Borůvka (pronounced: boh-roof-ka). In this method, we start with $|V|$ one-vertex trees, and for each vertex v , we look for an $edge(vw)$ of minimum weight among all edges outgoing from v and create small trees by including these edges. Then, we look for edges of minimal weight that can connect the resulting trees to larger trees. The process is finished when one tree is created. Here is a pseudocode for this algorithm:

```

BorůvkaAlgorithm(weighted connected undirected graph)
  make each vertex the root of a one-node tree;
  while there is more than one tree
    for each tree  $t$ 
       $e = \text{minimum weight edge}(vu)$  where  $v$  is included in  $t$  and  $u$  is not;
      create a tree by combining  $t$  and the tree that includes  $u$ 
      if such a tree does not exist yet;

```

- b. Another algorithm was discovered by Vojtech Jarník (pronounced: yar-neek) in 1936 and later rediscovered by Robert Prim. In this method, all of the edges are also initially ordered, but a candidate for inclusion in the spanning tree is an edge that not only does not lead to cycles in the tree, but also is incident to a vertex already in the tree:

```

JarnikPrimAlgorithm(weighted connected undirected graph)
  tree = null;
  edges = sequence of all edges of graph sorted by weight;
  for  $i = 1$  to  $|V| - 1$ 
    for  $j = 1$  to  $|edges|$ 
      if  $e_j$  from edges does not form a cycle with edges in tree and
        is incident to a vertex in tree
        add  $e_j$  to tree;
        break;

```

27. The algorithm `blockSearch()`, when used for undirected graphs, relies on the following observation: in a depth-first search tree created for an undirected graph, each back edge connects a successor to a predecessor (and not, for instance, a sibling to a sibling). Show the validity of this observation.
28. What is the complexity of `blockSearch()`?
29. Blocks in undirected graphs are defined in terms of edges, and the algorithm `blockDFS()` stores edges on the stack to output blocks. On the other hand, SCCs in digraphs are defined in terms of vertices, and the algorithm `strongDFS()` stores vertices on the stack to output SCC. Why?