

EC 504 – Fall 2019 – Homework 4

Due Thursday, Nov 14, 2019 in the beginning of class. Coding problems submitted in the directory /projectnb/alg504/yourname/HW4 on your SCC account by Friday Nov 15, 11:59PM.

Reading Assignment: CLRS Chapters 22, 23, 24 and appendix B.4

1. Consider the directed graph shown below in Fig. 1
 - (a) (10 pts) Perform breadth first search (BFS), starting from node 1 and draw the tree on the figure and label each node by the depth of the node, $d(i)$, and its parent (or predecessor), $p(i)$ for $i = 1, 2, \dots |E|$. Give a list of the order in which the nodes leave the queue. **Assume that arcs out of a node will be examined in order of increasing end node; i.e. break ties in order of arc selection by examining the arc with the smallest value of end node first.**
 - (b) (10 pts) For the same directed graph in Fig. 1, use depth first search (DFS), starting from node 1. **Give two lists for the order in which nodes entered into the stack and the order they leave the stack.** (An extra figure is appended to this Homework below.) Draw the stack in enough detail to understand its function. **Again assume that arcs out of a node will be examined in order of increasing end node; i.e. break ties in order of arc selection by examining the arc with the smallest value of end node first.** Is the reverse order a topological sort? (Explain)

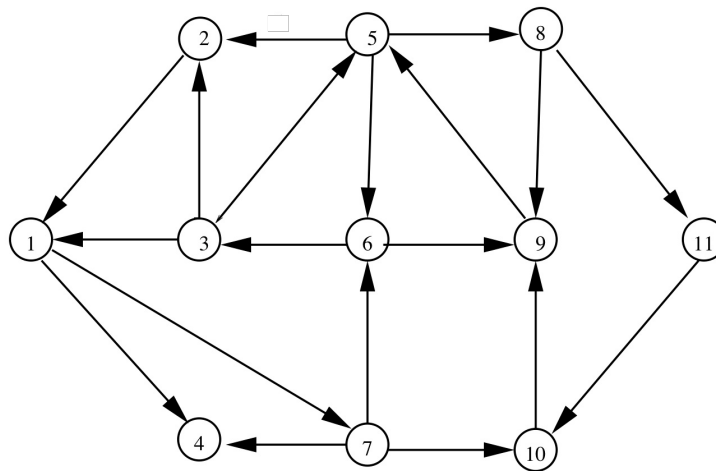


Figure 1

2. (10 pts) Compute a minimum spanning tree for Fig. 2 below, and **evaluate the weight** of that spanning tree using Kruskal's algorithm. Draw the tree and list in order the edges that are added to arrive at the tree. (HINT: You may check this by using Prim's algorithm, which will get the same MST but of course the order of adding edges is very different.)

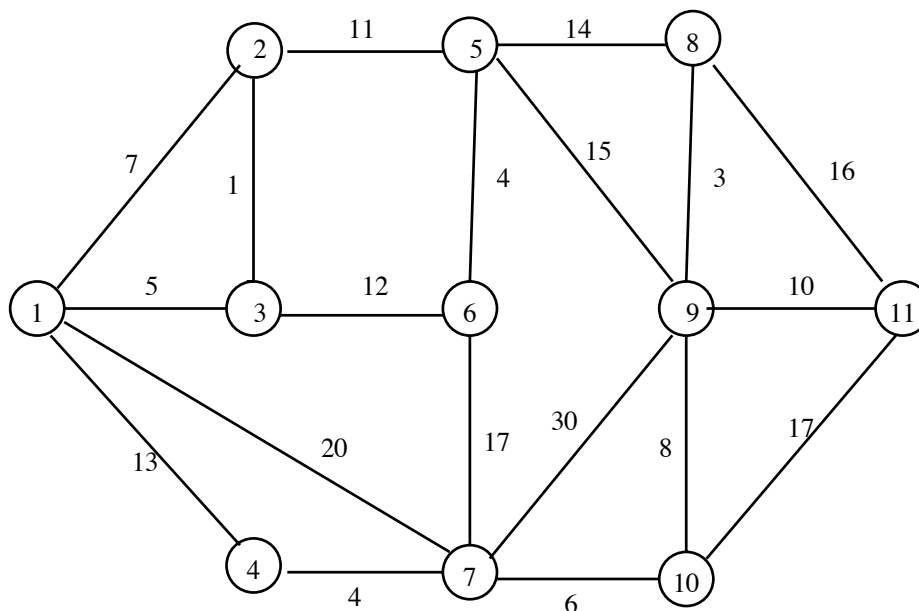


Figure 2:

3. (20 pts) This is a forward star representation for a directed graph with $|V| = 11$ vertices and $|E| = 16$ edges.

Vertex Number: 1 2 3 4 5 6 7 8 9 10 11 12
 Array First: { 1, 3, 4, 5, 7, 8, 12, 12, 14, 14, 15, 17 }

Edge Number: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
 Array Edge: { 2, 6, 6, 7, 3, 7, 8, 5, 8, 9, 10, 9, 11, 9, 9, 10, -1 }

- (a) Draw the graph on the template in Fig. 3. (HINT: You may want to do part b first.)

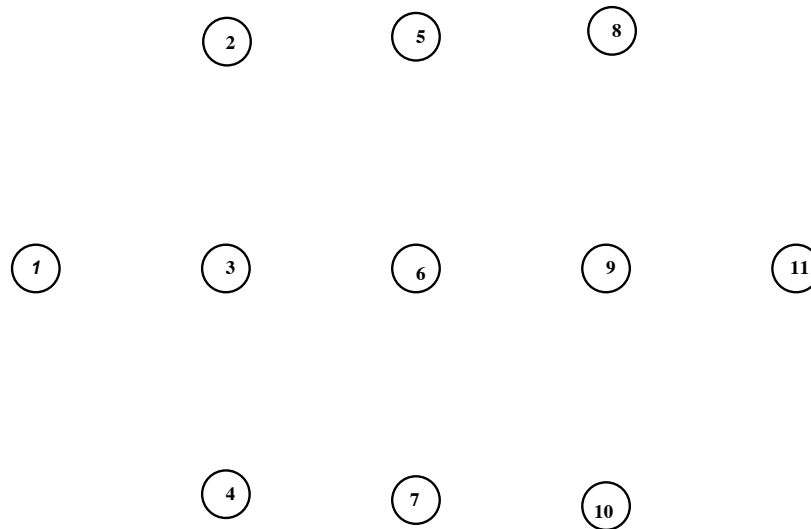


Figure 3:

- (b) Represent this graph as an adjacency list.
 (c) Is this graph a DAG?

4. An undirected bipartite graph $G(V, E)$ is one where its nodes can be partitioned into two disjoint sets $V = V_1 \cup V_2$, such that that every edge $e \in E$ is an arc $\{v_1, v_2\}$, where $v_1 \in V_1$ and $v_2 \in V_2$. Note that, in a bipartite graph, the length of every cycle must be an even number.

Define a pseudo code for an algorithm to determine whether an undirected graph is bipartite with worst case complexity $O(m)$, where m is the number of edges in the graph.

5. While Euler's theorem gives a characterization of planar graphs in terms of numbers of vertices, edges and faces, it is hard to establish whether a graph is planar or not if it is difficult to count faces. There are a couple of other properties of simple, connected planar graphs that derive from Euler's theorem:

- A simple, connected planar graph with $n \geq 3$ vertices and e edges must satisfy $e \leq 3n - 6$
- A simple, connected planar graph with $n \geq 3$ vertices, e edges and no cycles of length 3 must satisfy $e \leq 2n - 4$

A popular architecture for parallel computers is a hypercube. A hypercube of dimension k , denoted by Q_k , has 2^k nodes, and each node is connected to k other nodes. The nodes can be embedded into a k -dimensional boolean vector, and nodes are connected to other nodes that differ along one of its coordinates. Thus, Q_2 has nodes $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$, and has 4 edges. The node $(0,0)$ is connected to $(0,1)$ and $(1,0)$. Q_3 has nodes $(0, 0, 0)$, $(0, 0, 1)$, $(0, 1, 0)$, $(0, 1, 1)$, $(1, 0, 0)$, $(1, 0, 1)$, $(1, 1, 0)$, $(1, 1, 1)$. Node $(1,1,1)$ is connected to nodes $(0,1,1)$, $(1,0,1)$ and $(1,1,0)$. Note that the number of edges in a hypercube of dimension k is $k * 2^{k-1}$, since each node has k edges, and we divide by 2 so as not to count the number of arcs twice. Other important facts about hypercubes is that every hypercube is a bipartite graph.

- (a) Using the above facts, verify that Q_3 is a planar graph.
- (b) Using the above facts, show that Q_4 cannot be a planar graph.

6. Suppose you have computed a minimum spanning tree T for a graph $G(V, E)$. Assume you now add a new vertex n and undirected arcs $E_n = \{\{n, v_i\}, \text{ for some } v_i \in V\}$, with new weights w_{nv_i} . Provide the pseudocode for an algorithm to find the minimum weight spanning tree in the augmented graph $G_a(V \cup \{n\}, E \cup E_n)$. Estimate the complexity of this algorithm.

7. (20 pts) An undirected graph $G(V,E)$ is defined in terms of two arrays: `First[0:Vsize]` and `Edge[0:Esize]` that label the Vertices (nodes) from $0, 1, \dots, Vsize - 1$ and Edges (arcs) from $0, 1, \dots, Esize - 1$ respectively. The last “fake” vertex (`First[Vsize] = Esize`) points to the null “fake” edge with “null” value (`Edge[Esize] = -1`), See Exercise 3 above for an small example.

Implement your algorithm as a C/C++ function that counts the number of connected components. On GitHub there is the main file that reads input and writes output the result. You only write the required functions. A code `makeGraph.cpp` has been posted that generates “random” undirected graphs, which you use for your amusement if you want to test your algorithm small or large graphs of varying sparsity.

Your connected component function needs to call **over and over again** a `Grow(...)` function using either BFS (or DFS) until you have visited **all** the nodes in the graph. It is preferable to use BFS as a more efficient way to grow each connected component from its start node. The functions for BFS have been included. The number of times you call `Grow(...)` is the number of connected components. All you need to return is one extra a global array `Found[0:Vsize]` initialize to `-1` for not found and then set to something else when each node is visited so your next call to search `Grow(...)` goes only to nodes that have been left out.

The task is to first implement connected components with BFS and a Queue and then to add to the code another function doing DFS using a stack.

Write the function:

```
int find_connected_components_BFS()
Stack * createStack()
Push(..)
Pop(..)
int find_connected_components_DFS()
```

Find the number of times and connected components using BFS and then add the find DFS method so you can compare the time of execution and show that the number of connected components are the same of course!

Do not make any changes to the `infile` reading format. so that it is easy for the grader to run your code against a set of input files to test the code correctness. When you implement the DFS version the `outfile` writing format in `main()` You are given a set of input graphs and few solutions.

Analysis For extra credit if you pick the label found as a distinct number inside each connected component (e.g. $1, 2, \dots$, `NumberConnectComponents`), you can learn a lot more – like how many live in the largest most popular cluster etc.

Let’s discuss more extension class and how to graph results. Might lead to a fun project as well.

```
More fun on connected components
Google look at the curious
Regular graphs
Cluster finding variations etc
```