CSCE 221 Cover Page

Homework Assignment #3

First Name Alexander **Last Name** Kaiser **UIN** 924007333

User Name ALKYAYZZZ **E-mail address** alkyayzzz@tamu.edu

Please list all sources in the table below including web pages which you used to solve or implement the current homework. If you fail to cite sources you can get a lower number of points or even zero, read more on Aggie Honor System Office website: `http://aggiehonor.tamu.edu/`

Type of sources				
People				
Web pages (provide URL)				
Printed material	Data Structures and Algorithms in C++ M.T. Goodrich, R. Tamassia and D. Mount			
Other Sources				

I certify that I have listed all the sources that I used to develop the solutions/codes in the submitted work.
On my honor as an Aggie, I have neither given nor received any unauthorized help on this academic work.

Your Name Alexander Kaiser Date 11/18/16

Homework 3

due December 2 at 11:59 pm
Submission to e-Campus

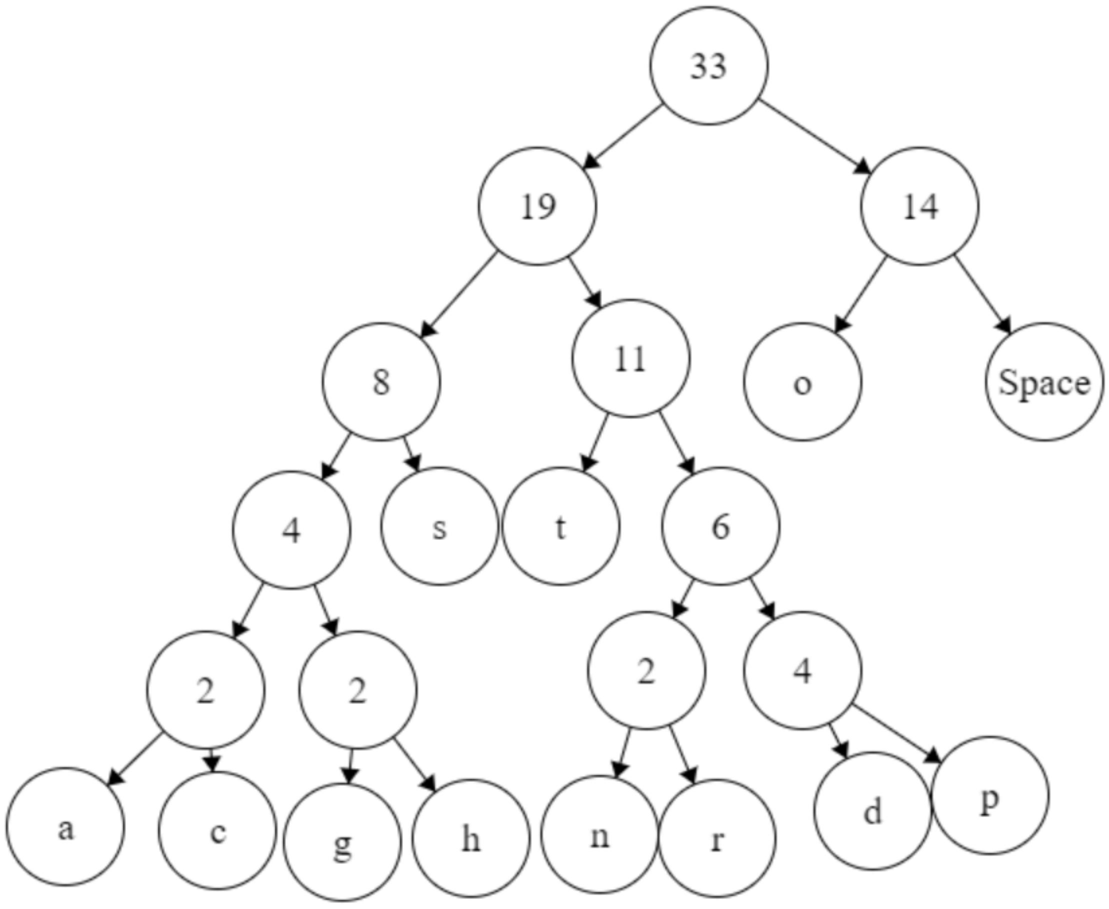
1. (10 points) R-8.7 p. 361
   We would use a heap data structure for the operations since they are each associated with a timestamp when they occur. The insertion would use the heap order property which states that any node besides the root node must be greater than its parent node. Thus the timestamp must be greater in this occation. The extraction uses the same property to ensure that the heap increases in a downwards direction. The big-oh notation for the insert and remove operations are $O(logn)$.

2. (10 points) R-12.14 p. 588

The frequncy table is as follows.

Character	a	c	g	h	n	r	d	p	s	t	o	Space
Count	1	1	1	1	1	1	2	2	4	5	7	7

The huffman tree is shown below. This was created using a priority queue and dropping each character with the lowest priority in the tree at the bottom first.

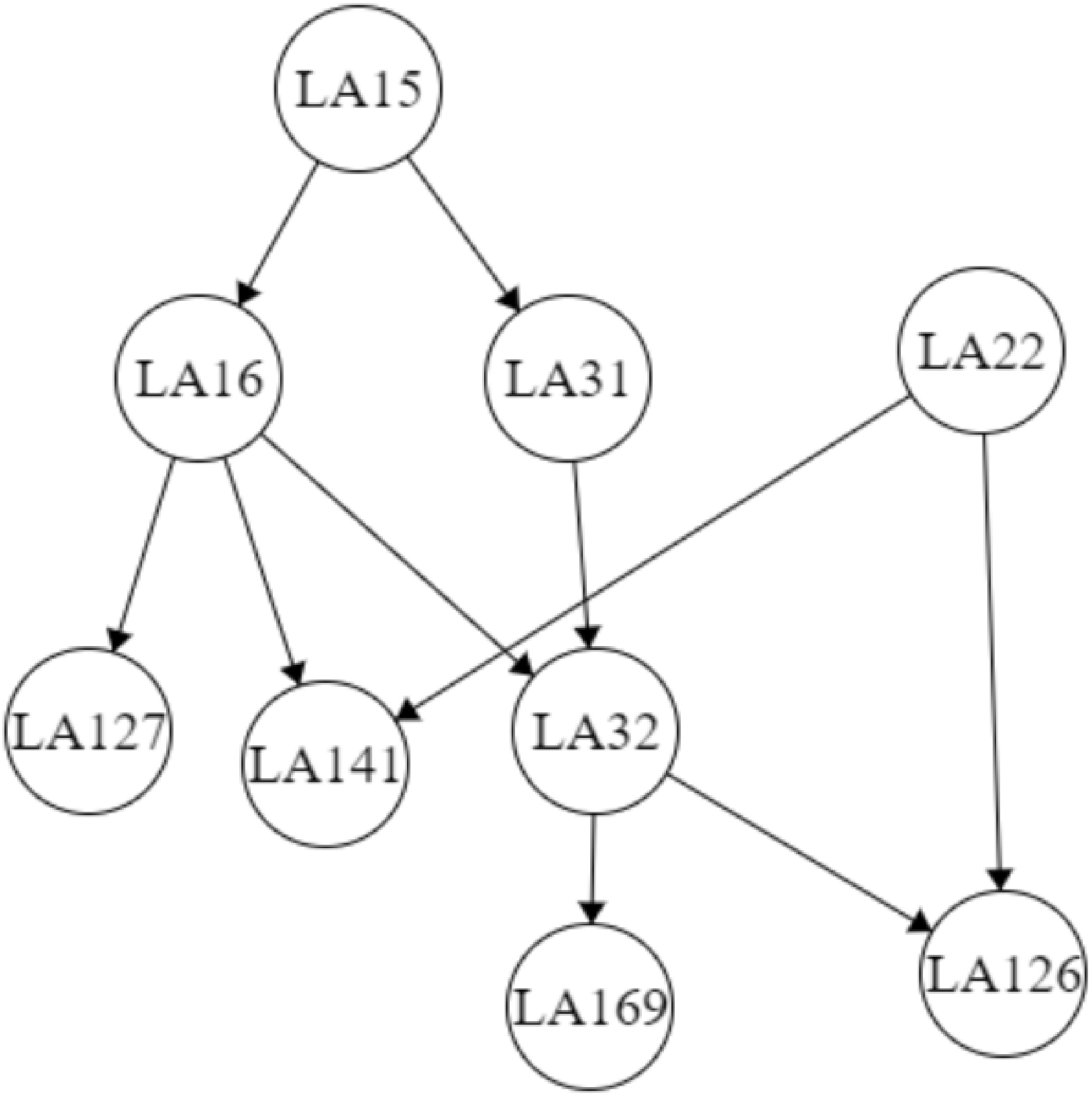


A leaf going to the right of a node has a code of 1 whereas a left leaf has a code of 0. This is how the code column is formulated below. The bits refer to how many digits there are in the code of the character times the frequency.

Character	a	c	g	h	n	r	d	p	s	t	o	Space
Code	00000	00001	00010	00011	01100	01101	01110	01111	001	010	10	11
Bits	5	5	5	5	5	5	10	10	12	15	14	14

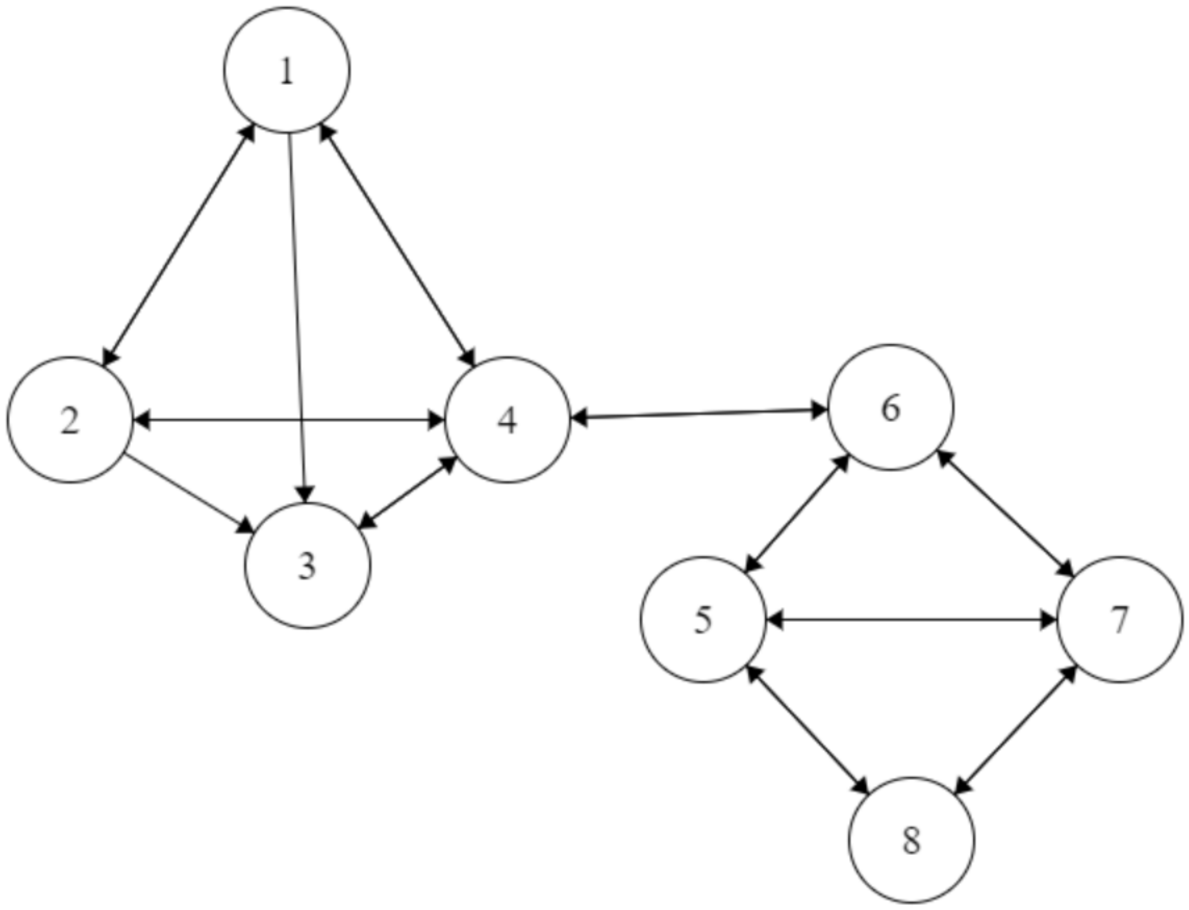
The ascii value of this text would be 264 bits by adding all the bits of the letters together whereas the huffman value would be 105 bits. This results in a 40

3. (10 points) R-13.5, p. 654
The tree for the traversal of courses is shown below.



Given the tree, the following is one possible sequence of classes.
LA15->LA16->LA31->LA22->LA32->LA127->LA141->LA126->LA169.

4. (10 points) R-13.7, p. 655
   a. Graph is drawn below.



b.Since depth first search vists each unexplored vertex and backtracks when passing an explored vertex until all vertex's have been traversed, the depth first search is 1->2->4->6->7->8->5->3.

c.Since bredth first search visits each row before backtracking to the next row, the bredth first search is 1->2->3->4->6->7->5->8.

5. (10 points) R-13.8, p. 655

   a.The choice can be made by finding how much space each would take. With 20000 edges, the number of nodes in an adjacency list would be around 20000. On the other hand the adjacency matrix would contain 10000 x 10000 boolean values and each value takes one byte space. Thus a matrix would require much more space and it would be better to use the adjacency list structure.

   b.Based off the logic of part a, the adjacency list and the adjacency matrix are about the same. Both take up around the same amount of space.

   c.The adjacency matrix is preferrable in this instance. it supports the operation areAdjacent() in $O(1)$ time.

6. (10 points) R-13.16, p. 656

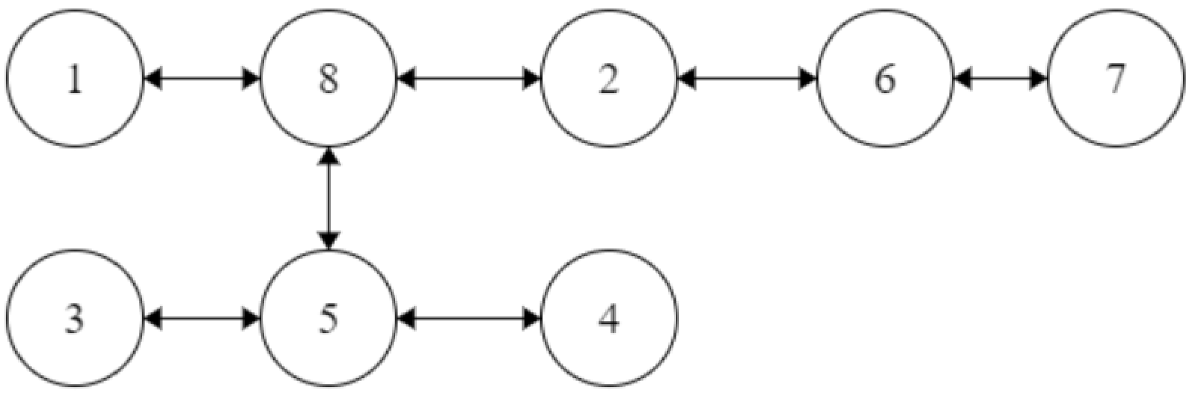
   For each vertex we would introduce a variable closest that stores its adjacent vertex in the cloud. In order to construct the tree, the following changes would be made.
   1. When a vertex that isnt equal to v with a minimum D(u) is removed from the priority queue Q, add the edge (closest[u], u) to the tree T.
   2. if (D[u] + w((u,z)) < D[z]) holds, set closest[z] to u.

```
Algorithm DijkstraDistances(G, v)
Input: A simple undirected weighted graph G with nonnegative edge weights, and a distinguished vertex s.
Output: A label D[u], for each vertex u, such that D[u] is the length of a shortest path from v to u in G.
{ for each u in G do
if ( u = v )
D[u] = 0;
else
D[u] = +inf;
Create a priority queue Q containing all the vertices of G using the D labels as keys;
Create an empty tree T;
while Q is not empty do
{
u = Q.removeMin();
if ( u!=v )
add the edge (closest[u], u) to T;
for each z in Q such that z is adjacent to u do
if ( D[u] + w((u,z)) < D[z] ) // relax edge e
{ D[z] = D[u] + w((u,z));
Change to D[z] the key of vertex z in Q;
closest[z]=u;
}
}
return T and the label D[u] of each vertex u;
}
```

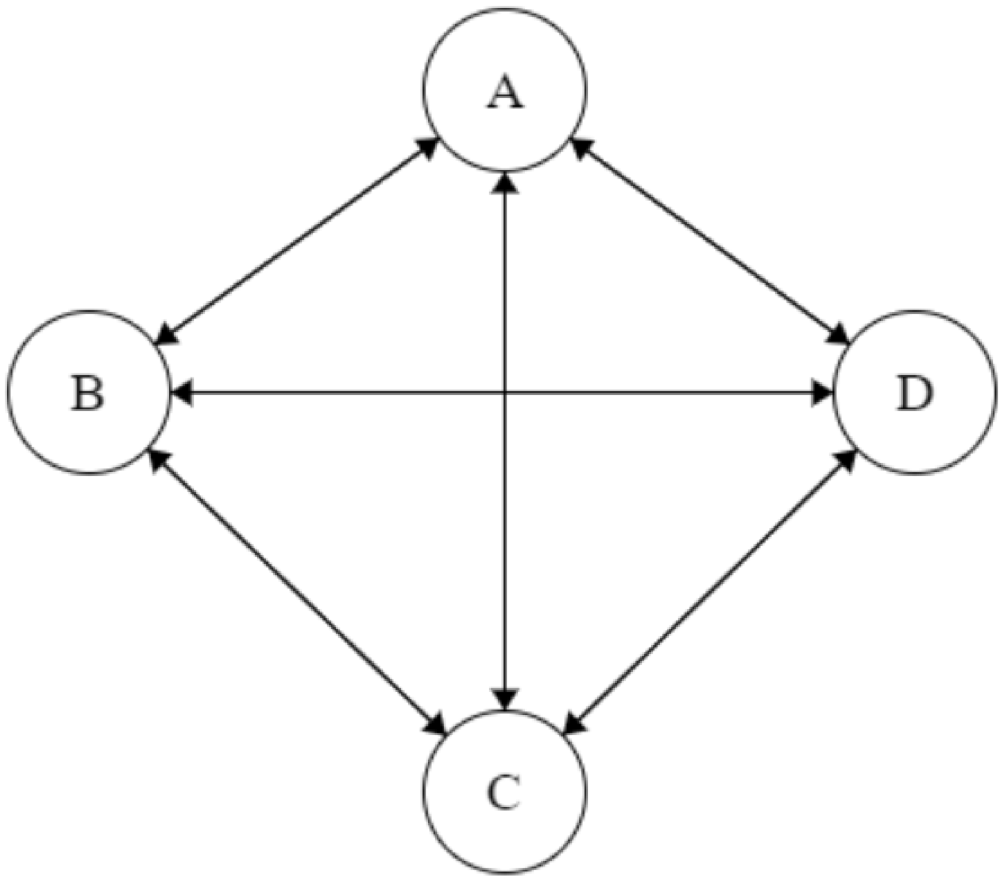
7. (10 points) R-13.17, p. 656
a.Kruskal's algorithm-works by starting with each vertex in a separate cluster and merging 2 clusters in each step. Thus inthis case the clusters are merged in this order:
3 to 5-115
1 to 8-120
2 to 8-155
4 to 5-160
5 to 8-170
6 to 7-175
2 to 6-180

b.Prim-Jarnik algrithm-works by adding a new vector to the tree at each step. The order of adding will depend on the starting vertex. The following is made by choosing 8 as the starting vertex.
8 to 1-120
8 to 2-155
8 to 5-170
5 to 3-115
5 to 4-160
2 to 6-180
6 to 7-175

Both a and b produce this graph.

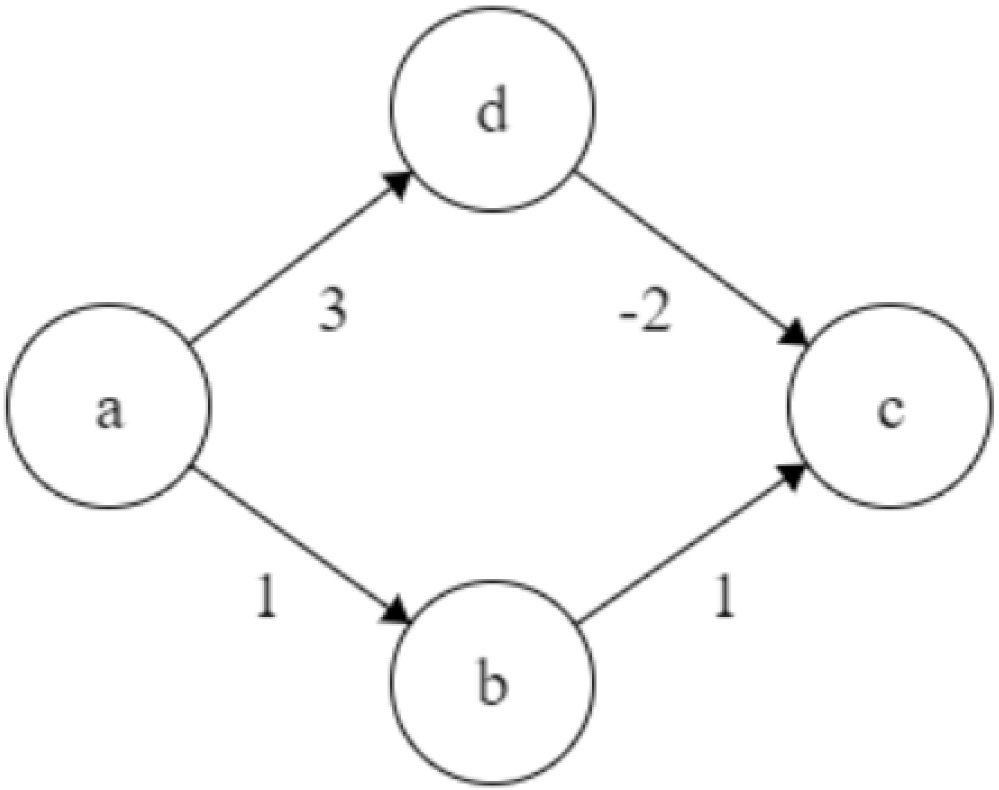


8. (10 points) R-13.31, p. 657
The depth first search of a complete graph is a single path as opposed to the breadth first search of a complete graph which is a star path.
For example the depth first search of the following graph is A->B->C->D.

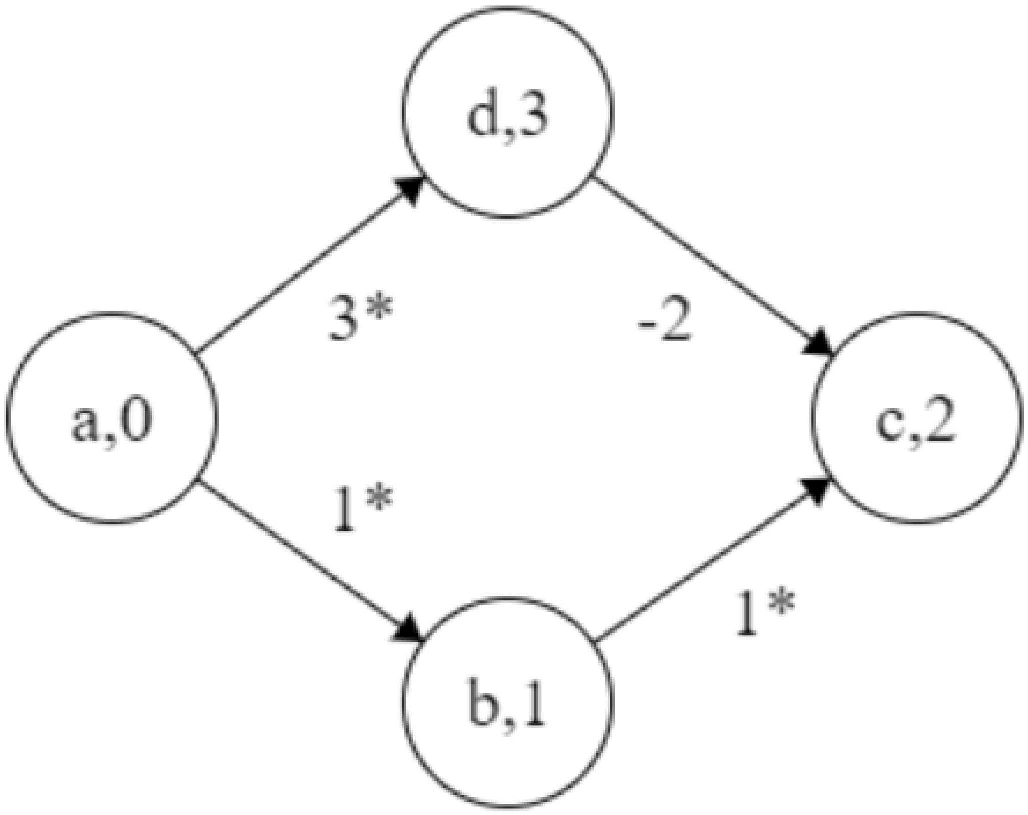

A
B
D
C

9. (10 points) C-13.10, p. 658
We traverse the graph using a depth first search for directed graphs. However, the circuit cannot be closed until all edges have been traversed or we have to add more edges to the circuit. It has a $O(n+m)$ run time because it traverses each edge once in a depth first search manner, then it may spend at most $O(m)$ time reconnecting all the circuits.

10. (10 points) C-13.15, p. 659
If we apply Dijkstra's algorithm to the graph below, with vertex a as the source, it processes the verticies in the order a->b->c->d.



This constructs the shortest path tree shown below, where the path from a to c are not the shortest.(arrows with stars next to their distance represent a bolded arrow)



In this case the shortest path should be a->d->c->b, however the negative path throws this off.