CSCE 221 Assignment 5 Cover Page

First Name Peng Last Name Li UIN 822003660

User Name abc E-mail address billlipeng@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)	http://en.wikipedia.org/wiki/Dijkstra's_algorithm http://stackoverflow.com/questions/6799172/negative-weights-using-d			
Printed material	CSCE221 Slides			
Other Sources				

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

Your Name  Date Nov 25, 2014

Homework 5
due November 25, 2014

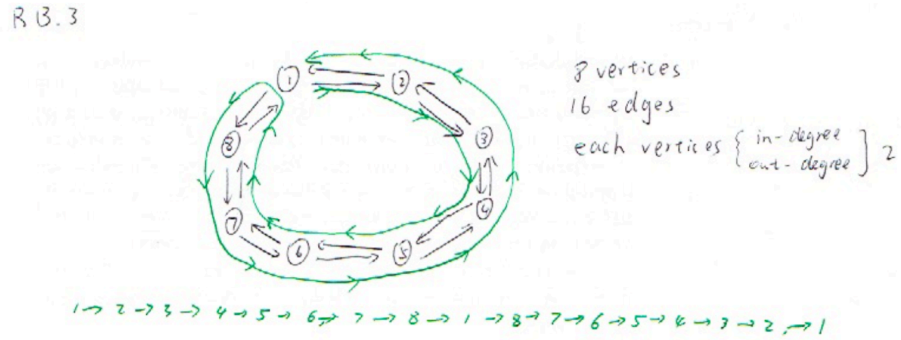
Write clearly and give full explanations to solutions for all the problems. Show all steps of your work.

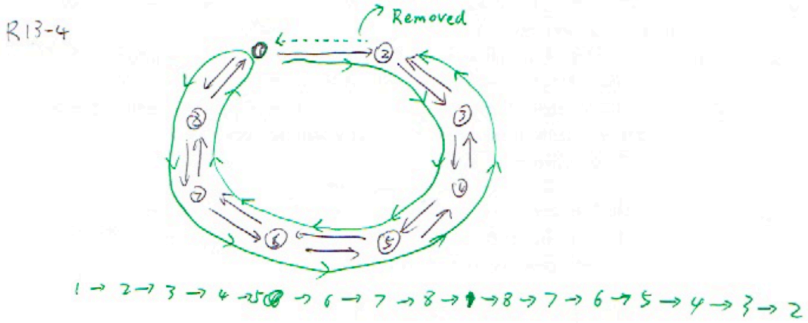
Reading assignment.

- Priority Queue, Chap. 8
- Graphs, Chap. 13

Problems.

1. (10 points) R-13-3 and R-13-4, p. 654





2. (10 points) R-13.8, p. 655
   a. 10000 vertices, 20000 edges, use as little space as possible.
      Adjacency list structure. Using adjacency matrix will cost too much spaces, we might need put 10000 x 10000 item into the matrix. However, using adjacency list will cost about 20000 nodes.
   b. 10000 vertices and 20000000 edges, use as little space as possible.
      Adjacency matrix structure. Because adjacency matrix stores the edges in the a Boolean value that indicates whether an edge is present between the vertices corresponding to the row and column of the cell.
   c. To answer the query isAdjacentTo as fast as possible, no requirement on space.
      Adjacency matrix structure. In adjacency matrix it cost 1 operation to check is one vertex is connected to another vertex.

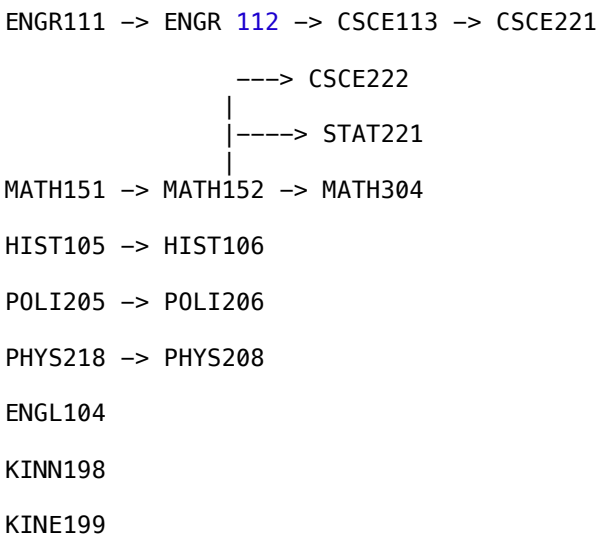
3. (10 points) R-13.16, p. 656

```
Dijkstra(G,w,s):

Initialize-Single-Source(G,s);
S = Ø;
initialize Q to contain all v ∈ V;
while ( Q! = Ø ) {
    u = Extract-Min(Q);
    S = S U {u};
    set v0 as the root of the tree.
    for(each vertex v ∈ Adj[u]){
        Relax(u,v,w);
        update parent of v to u;
        link the childred and parent together;
    }
}
output the distance from v to each vertex;
          output the tree structure;
```

4. (10 points) R-13.17, p. 656

```
Use Prim's algorithm to find the spanning tree.
The tree starts from 1.
1 ----(120)---> 8   {1, 8}
8 ----(155)---> 2   {1, 8, 2}
8 ----(170)---> 5   {1, 8, 2, 5}
5 ----(115)---> 3   {1, 8, 2, 5, 3}
5 ----(160)---> 4   {1, 8, 2, 5, 3, 4}
2 ----(180)---> 6   {1, 8, 2, 5, 3, 4, 6}
6 ----(175)---> 7   {1, 8, 2, 5, 3, 4, 6, 7}

Total weight = 120 + 155 + 170 + 115 + 160 + 180 + 175 = 1075
```

5. (10 points) You want to help CS/CSE freshman students to prepare their course schedules for the first two years in the lower level division. By building a directed graph suggest order in which they should schedule their courses taking into account their corresponding prerequisites. A set of vertices represents courses and a set of edges represents a dependence of a given course on a course prerequisite.



6. (10 points) R-13.31, p. 656

Every internal node in the tree only has one parent and one child except for the root. The root has only one child but no parent. The only one external node has one parent but no child.

7. (10 points) Write what the running time, and provide its justification, of the Dijkstra's algorithm is for a sparse and dense graph and the priority queue implemented based on

(a) a binary heap

Sparse graph.

Function	Cost
CreateBinaryHeap	O(V*log(V))
RemoveMin	O(V*log(V))
DecreaseKey	O(E*log(V))
Total	O((V+E)*log(V))

Dense graph.

Function	Cost
CreateBinaryHeap	O(V*log(V))
RemoveMin	O(V*log(V))
DecreaseKey	$O(V^2*log(V))$
Total	$O(V^2*log(V))$

(b) an unsorted list

Sparse graph.

Function	Cost
CreateBinaryHeap	O(V)
RemoveMin	$O(V^2)$
DecreaseKey	O(E)
Total	$O(V^2+E)$

Dense graph.

Function	Cost
CreateBinaryHeap	O(V)
RemoveMin	$O(V^2)$
DecreaseKey	$O(V^2)$
Total	$O(V^2)$

(c) a sorted list

Sparse graph.

Function	Cost
CreateBinaryHeap	$O(V^2)$
RemoveMin	$O(V)$
DecreaseKey	$O(E)$
Total	$O(V^2+E)$

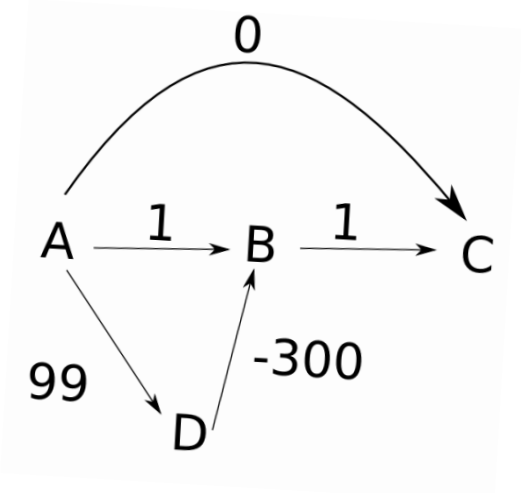
Dense graph.

Function	Cost
CreateBinaryHeap	$O(V^2)$
RemoveMin	$O(V)$
DecreaseKey	$O(V^2)$
Total	$O(V^2)$

8. (10 points) C-13.10, p. 658

A possible implementation is use DFS traversal with some modifications to do an Euler tour. First traverse the graph with DFS. But the algorithm does not travel the vertex that has been travelled before visiting all vertices. Thus, the algorithm checks if current vertex has even number (zero is not count as even number) of visited adjacent edges (both in and out edges). If so, the current vertex is a distinguished vertex that connects two or more loops together. So the algorithm start from this distinguished vertex and find the other closed loops. Once it has visited all the vertices, it should come back to this distinguished vertex. There might be more than one distinguished vertex in a graph, but the algorithm will be the same.
Since there are n vertices and m edges, the algorithm visits each vertex and edge for one time. The running time is O(n+m).

9. (10 points) C-13.15, p. 659



1. Set d(A) to 0 and the other vertices to infinity.
2. Set d(B) to 1, d(C) to 0, d(D) to 99.
3. Expand out C, no change.
4. Expand out B, no change.
5. Expand out D, changes d(B) to −201.

Finally, d(A) = 0, d(B) = −201, d(C) = 0, d(D) = 99. However, d(C) should be −200.

10. (10 points) C-13.18, p. 659

Create a new relax function.

```
Relax(u,v,w):
    if(d[v] < d[u]+w(u,v)){
        d[v] = d[u] + w(u,v);
        p[v]=u;
    }
```

1. Create a maximum priority queue (MPQ).
   Initialize all the vertices in the priority queue to be negative infinity.
   Initialized the first vertex to be zero.
2. Extract-Max(Q) and relax all its adjacent vertices.
3. Repeat this until the queue is empty.

Assume that we are using the binary heap to implement the MPQ.

1. Build MPQ costs O(V∗logV).
2. Remove max costs O(V∗logV).
3. Decrease key costs O(E∗logV).

The complexity of this algorithm is O((V+E)logV).