Avik Bag
Professor Peysakhov
CS 260 – Assignment #8
17th August, 2015

Written Problems

Question 1

According to Dijkstra's algorithm, when the vertex is marked closed and assumes that the shortest path to it is found. If we have a open node such that the cost to the next node is minimal, the minimality will not change. This is because adding positive numbers to a vertex will not change this value. If we have negative numbers, the basic concept of Dijkstra's algorithm is not valid, and thus won't work.

Question 2

```
define delete_edge(graph, node1, node2) // using a dict representation of graphs
begin
        connections1 = graph.get[node1] // list of connections to that node
        connections2 = graph.get[node2] // list of connections to that node
        if node1 is in connections2 && node2 is in connections1 then
                delete node2 from connection1
                delete node1 from connection2
        else
                connection doesnt exist
end

define insert_edge(graph, node1, node2)
begin
        if graph.has_key(node1) == False && graph.has_key(node2) == False
                        return // exit function
        else
                connections1 = graph.get[node1] // list of connections to that node
                connections2 = graph.get[node2] // list of connections to that node
                connections1.append(node2)
                connections2.append(node1)
end
```

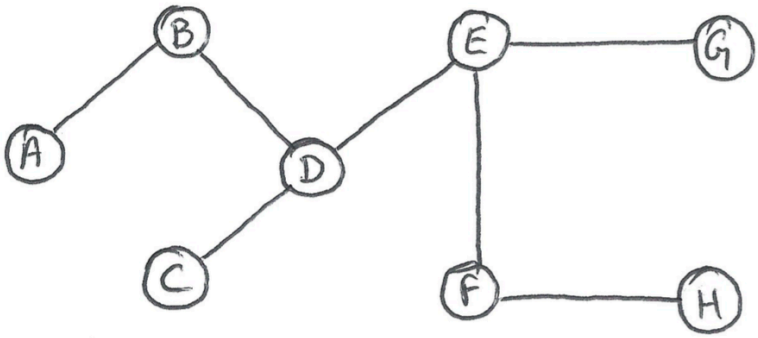
Question 3

To make adjustments to the adjacency list, every connection list should have a corresponding node to which it is connected just once. In other words, if I have node a, and the connection list is [b,d,e] and when I want to represent node b, I do not include node a in the connection list as it would be redundant. Node with the higher order will be given priority can contain the connections to the subsequent nodes. Thus allowing us to delete any node at constant time, and also any edge at constant time.

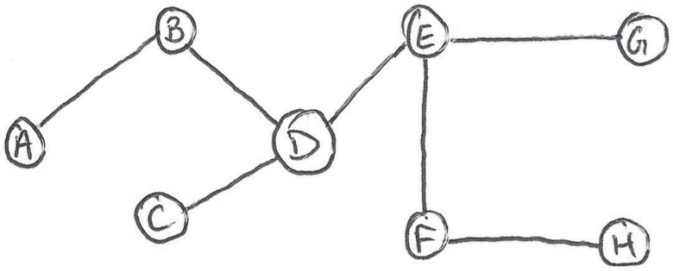
```
define delete_edge_const(graph, node1) // using a dict representation of graphs
begin
        connections = graph.get[node1] // list of connections to that node
        delete element at index 0 of connections
end
```

Question 4

a.) Using Prim's Algorithm.



b.) Using Kruskal's Algorithm.



Queue

~~(c,d) = 1~~
~~(e,f) = 1~~
~~(f,h) = 1~~
~~(a,b) = 2~~
~~(b,d) = 2~~
~~(d,e) = 2~~
~~(e,g) = 2~~
* (f,g) = 2
(a,c) = 3
(g,h) = 3
(d,f) = 4.

Starting at A (both DFS and BFS)

```
Aviks-MacBook-Pro:week_7 avikbag$ python graph_traversal8.py
Adjacency list for the graph used
A  ->  ['B', 'C']
B  ->  ['A', 'D']
C  ->  ['A', 'D']
D  ->  ['B', 'C', 'E', 'F']
E  ->  ['D', 'F', 'G']
F  ->  ['D', 'E', 'G', 'H']
G  ->  ['E', 'F', 'H']
H  ->  ['F', 'G']
DFS on the given graph (root = Node A)
-> A  -> B  -> D  -> C  -> E  -> F  -> G  -> H
BFS on the given graph (root = Node A)
-> A  -> B  -> C  -> D  -> E  -> F  -> G  -> H
```

Starting at D (both DFS and BFS)

```
Aviks-MacBook-Pro:week_7 avikbag$ python graph_traversal8.py
Adjacency list for the graph used
A  ->  ['B', 'C']
B  ->  ['A', 'D']
C  ->  ['A', 'D']
D  ->  ['B', 'C', 'E', 'F']
E  ->  ['D', 'F', 'G']
F  ->  ['D', 'E', 'G', 'H']
G  ->  ['E', 'F', 'H']
H  ->  ['F', 'G']
DFS on the given graph (root = Node D)
-> D  -> B  -> A  -> C  -> E  -> F  -> G  -> H
BFS on the given graph (root = Node D)
-> D  -> B  -> C  -> E  -> F  -> A  -> G  -> H
```