CS2134 Homework 10
Programming Question 1 and Written Questions 1-9
Due* 11:00 p.m. Wed. May 4, 2016
Programming Question 2 Due 11:00 p.m. Mon. May 9, 2016
Written Questions 10 & 11 Do Not Need to Be Turned In

Be sure to include your name at the beginning of each file! Assignment 10 include a programming portion and a written part. The programming portion must compile and consist of a single file (hw10.cpp). The written portion should consist of two file (hw10awritten & hw10bwritten) in a .pdf format. Be sure to include your name at the beginning of each file! You must hand in the file via NYU Classes.

Programming Part:

1. Write a function, call `print_kth_largest` that takes two parameters, a `vector` and an `int`. The function will print out the the `k`th largest items in the vector. Your algorithm must be efficient and must use a `priority_queue`

   ```
   template<class Comparable>
   void print_kth_largest(const vector<Comparable> & a, int k)
   ```

2. Write the code to find how to go from one subway station to another subway station where you have the smallest number of hops (i.e. travel through the smallest number of subway stops or transfers.)

   To do this you need to perform the following steps:

   - Use the code you have previously written to create the adjacency list.[1] This will be the graph you use in the algorithm, `shortestpaths`.
   - Modify the shortest path algorithm, `shortestpaths`, so that it now computes the shortest path to go from one `stop_id` to another `stop_id` and uses the graph *you* created.
     To do this, you will need to change the parameters of the algorithm, `shortestpaths` and part of the code.
   - Modify the code so it prints out the shortest path to go from the one `stop_id` to other `stop_id`. Remember to print out the `stop_id`'s.

3. **1 point extra credit towards final grade.** Create a class that cleanly uses all the MTA code you have written and *adds* some other useful methods that improve the usability of the class[2]. To receive this credit, you must explain your code to me.

*10% extra credit if you turn in the assignment on Tuesday May 3rd by 11:00 p.m.

[1]You may use the solution to the previous homework problem if you provide a comment telling where you got the code from.

[2]Be creative! What additional functionality would be useful to have.

Written Part:

1. What is the *big-Oh* running time of your code in programming problem 1? Use n and k in your answer, where n is the size of the `vector` and `k` is the number of items you print.

2. If we implemented[3] the ADT `priority queue` using a `unsorted vector`, how long would it take to perform the following operations of a priority queue? Write your answer using Big-Oh notation.

   - `insert(x);` // Average case time:____________

   - `deleteMin();` // Average case time:____________

   - `findMin();` // Average case time:____________

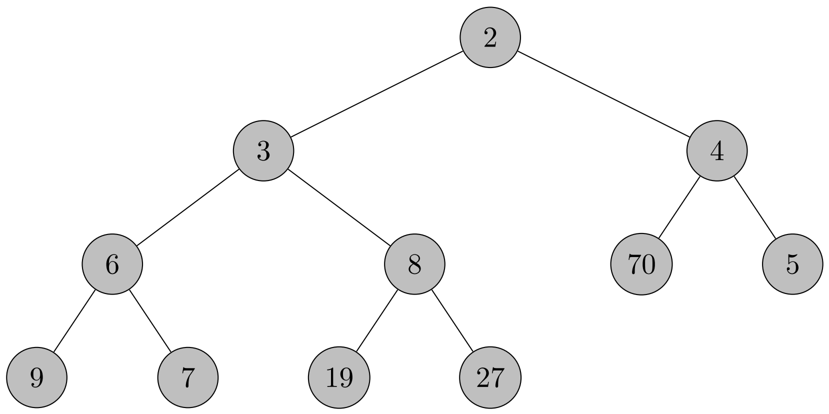
3. Consider the following array representation of a binary heap:

   ```
   [sentinel, 190,  380,  200,  388,  401,  277, 270, 1000, 399,  432]
   ```

   (a) Show the tree representation of the binary heap.

   (b) Insert 197 into the binary heap; show both the tree representation and the array representation after 197 has been inserted.

4. Consider the following tree representation of a binary heap:



   (a) Show the array representation.

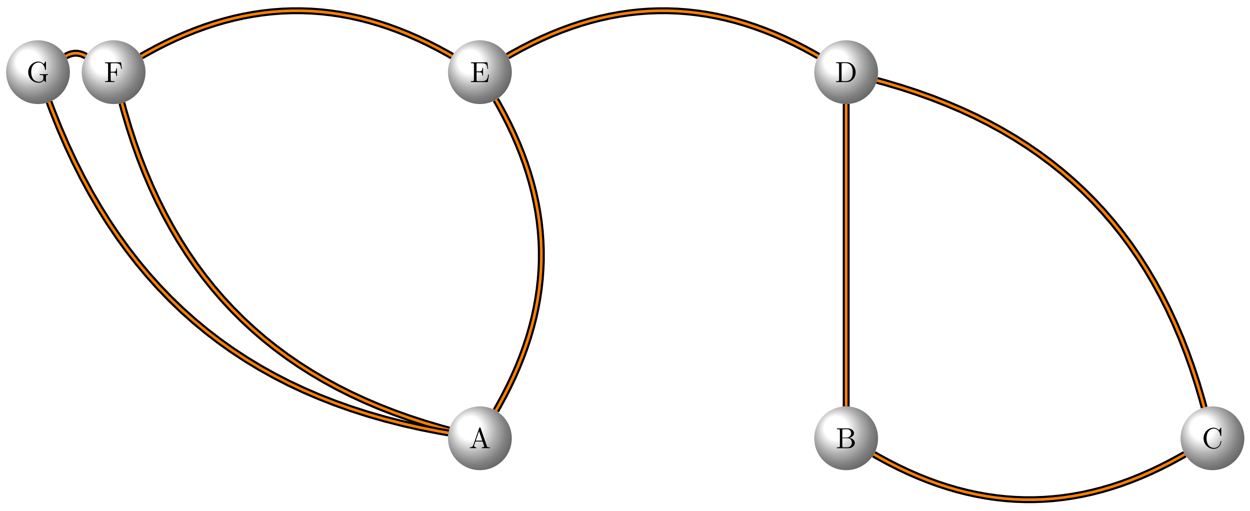
   (b) Show what happens when the root is removed by giving the tree representation of the binary heap.

5. Rewrite the BinaryHeap insert routine where the items are stored in position 0 through `theSize`-1.

6. Show the result of using the linear-time algorithm to build a binary heap on input $10, 12, 1, 14, 6, 5, 8, 15, 3, 7, 4, 11, 10, 0$. In your answer, provide the vector representation of the heap.

[3]Remember for any ADT there are many choices of how it could be implemented. Some choices are more efficient than others.

7. Illustrate the execution of the single source shortest path algorithm for an unweighted graph (breadth first search) on the (undirected) graph below:



At each phase, show the node being visited in that phase, and the relevant data at the end of the phase, including:

- the distance to each vertex that has been discovered (including those already visited)
- the predecessor of each discovered node on a shortest path from the source
- the queue of nodes that have been discovered but not yet visited

The answer at the end of the first phase is shown in the table below. It indicates that at the end of the first phase, we have visited `G` and have discovered `A` and `F`, each with estimated distance 1 and with predecessor `G`. Complete the table. (continue until all the nodes have been removed from the queue.)

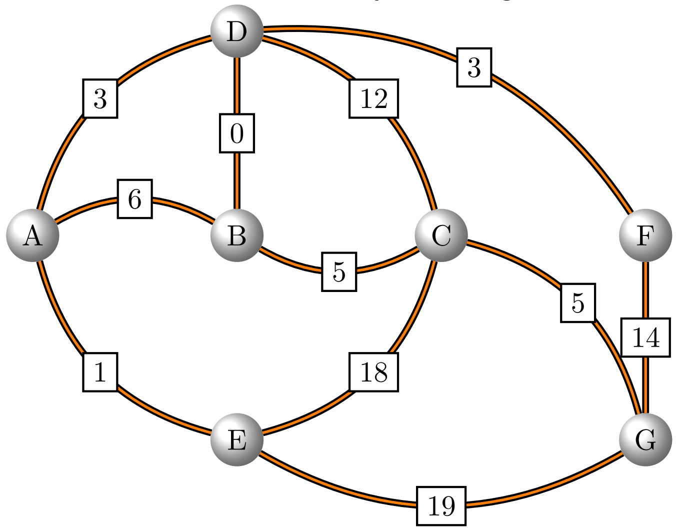
phase	distance/predecessor							visiting	queue
	G	A	B	C	D	E	F		(front at queue on the left)
init	0/-								G
1	0/-	1/G					1/G	G	F A
2									

.
.
.

Use the chart you have created to determine a shortest path from `G` to `C`.

8. For the graph in written question 9 show the adjacency matrix and the adjacency list representation.

9. Illustrate the execution of Dijkstra's algorithm on the weighted (undirected) graph below:



Show all the steps using the following chart:

phase	distance/predecessor							visiting	discovered
	A	B	C	D	E	F	G		
init	0	inf	inf	inf	inf	inf	inf		A
1	0	5/A	inf	3/A	1/A	inf	inf	A	B D E
2									
.									
.									
.									

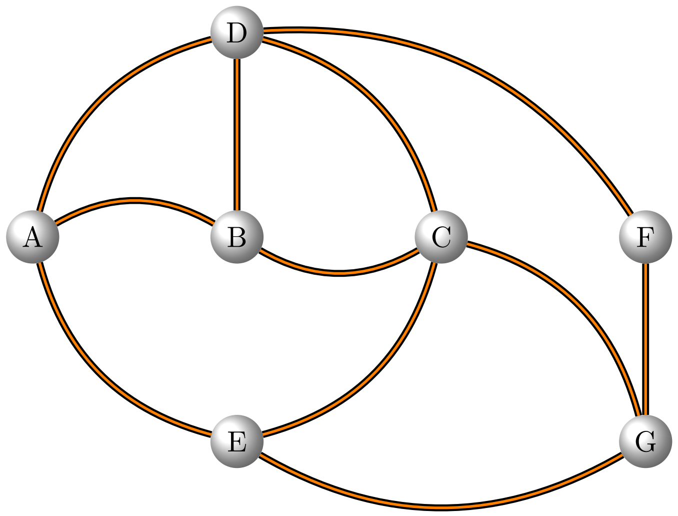
i.e. at each phase, show:

- which phase your are in
- the node being visited in that phase
- the current estimate of distance to each vertex (via nodes already visited)
- the predecessor of the node (on a shortest path via nodes already visited, i.e. the current estimate of the predecessor)
- all the nodes that have been discovered, but not yet visited.

Continue the chart until all the nodes have been visited.

Using the chart you created, **determine** a shortest weighted path from `A` to `G`.

10. Consider the following graph:



Show the order you first *discover* the vertices when starting at node `A`

- if you do a DFS traversal of the graph
- if you do a BFS traversal of the graph

11. Find a topological ordering for the graph given by the following adjacency list:

- $s :\to a \to g$
- $a :\to b \to e$
- $b :\to c$
- $c :\to t$
- $d :\to e$
- $e :\to c \to f \to i$
- $f :\to t$
- $g :\to d \to h$
- $h :\to i$
- $i :\to f \to t$
- $t :$