CS302 Final Exam, December 9, 2015 - James S. Plank

Question 1

In class (and in lecture notes for those who skipped class), I defined the halting problem in terms of the procedure below. Please define the halting problem in terms of this procedure.

```
string halt(string program_file, string input_file);
```

Question 2

For each of the following three arrays, please tell me what the state of the array will be when the array is sorted using merge sort, just before the final merge.

- **Part A:** { 3 1 3 2 4 4 5 2 }
- **Part B:** { 2 4 5 3 1 3 4 2 }
- **Part C:** { 2 4 1 3 3 4 2 5 }

For each of the following three arrays, please tell me what the state of the array will be after the first partitioning of quicksort, assuming that the first element of the array is the pivot. Show me the state *before this element is swapped into its final place at the end of partitioning*.

- **Part D:** { 6 0 3 7 9 4 5 0 0 1 }
- **Part E:** { 7 4 8 1 4 7 7 9 6 4 }
- **Part F:** { 6 7 6 5 9 8 6 4 9 6 }

Question 3

A subsequence of a string *S* is a string that may be obtained by deleting characters of *S*. For example, "pan" and "lak" are subsequences of the string "plank", but "nap" is not. Given the class definition on the right, the method *Maxseq(A, B)* returns the length of the longest string which is a subsequence of both strings *A* and *B*. *Maxseq()* is implemented for you. However, it is implemented in terms of a recursive method called *Maxseq_DP()*. Your job is to implement the recursive method *Maxseq_DP()* so that *Maxseq()* works correctly, and its running time is *O(A.size()*B.size())*.

```
class MS {
  public:
    int Maxseq(string A, string B);
    int Maxseq_DP(int a, int b);
    string SA, SB;
    vector < vector <int> > cache;
};

int MS::Maxseq(string A, string B)
{
  int i;

  SA = A;
  SB = B;
  cache.resize(A.size());
  for (i = 0; i < cache.size(); i++) {
    cache[i].resize(B.size(), -1);
  }
  return Maxseq_DP(0, 0);
}
```

CS302 Final Exam, December 9, 2015 - James S. Plank - Page 2

Question 4

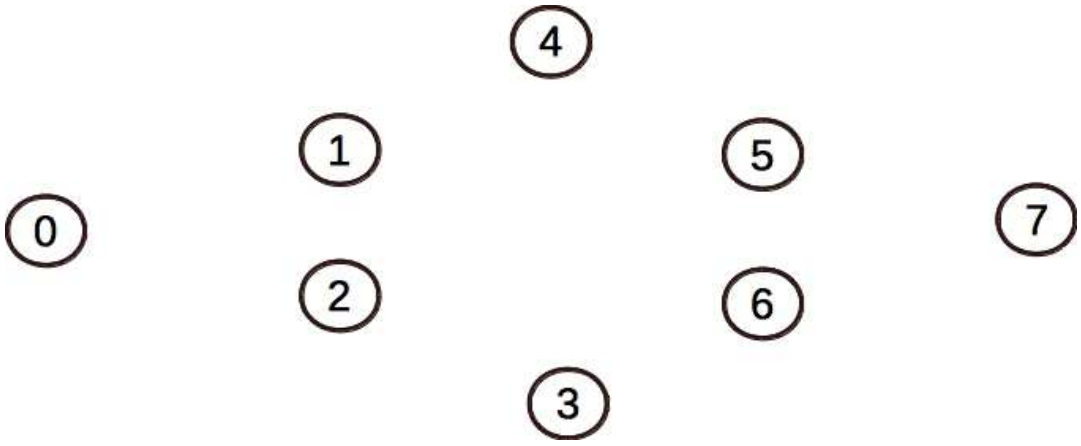
Use the following multiple choice answers for these questions.

- a. Union/Find on Disjoint sets
- b. Depth-First Search
- c. Topological Sort
- d. Push/Pop on Priority Queues
- e. Dijkstra's algorithm modified to find paths with minimum flow
- f. Breadth-First Search
- g. Dijkstra's algorithm modified to find paths with maximum flow
- h. Dijkstra's algorithm for shortest paths
- i. Network flow

Here are the questions:

- **Part A**: Which algorithm is used to find shortest paths in an unweighted, undirected graph?
- **Part B**: Which algorithm can be used to find the maximum matching of a bipartite graph?
- **Part C**: Which algorithm is used when you are processing the sorted edges in Kruskal's algorithm?
- **Part D**: Which algorithm can be used to find the minimum cut of a weigthed graph?
- **Part E**: Which algorithm is used to find the augmenting paths in the Edmonds-Karp algorithm?
- **Part F**: Which algorithm can be used to find shortest paths in a directed acyclic graph in $O(|V|+|E|)$ time?
- **Part G**: Which algorithm can be used to identify the connected components of an unweighted, undirected graph in $O(|V|+|E|)$ time?
- **Part H**: Which algorithm maintains a multimap of nodes, keyed on distance to a starting node?

You may find this scratch drawing useful for question 5:



CS302 Final Exam, December 9, 2015 - James S. Plank - Page 3

Question 5

To the right is an adjacency matrix for a residual graph in a network flow calculation, where the source is node 0 and the sink is node 7. If the entry *x* is in row *r* and column *c* of the matrix, then there is an edge with weight *x* from node *r* to node *c*.

Your job is to perform one pass of the Edmonds-Karp algorithm, which means finding the proper augmenting path, and then processing the residual graph.

	0	1	2	3	4	5	6	7
0		78			11			
1			47					
2	61	6		19				
3	36		71				81	
4	87							
5			74				63	82
6			13			32		
7					37	15	88	

Part A: What is the augmenting path (list the nodes in order)?
Part B: What is the flow through this path?
Part C: Tell me the changes in the residual graph. You do this on the answer sheet, using a table with four columns:

- In the first column, you write "remove", "add" or "change", to specify whether you are removing, adding or changing an edge on the residual graph.
- The second column is the id of the "from" node.
- The third column is the id of the "to" node.
- The fourth column is the new weight of the edge. If you are removing an edge, then leave this column blank.

For example (and this is an example of how you format your output -- it has nothing to do with the graph above), suppose that you remove edge [0,3] from the graph, you add edge [5,6] to the graph with a weight of 20, you change the weight of edge [4,2] from 15 to 27, and you change the weight of edge [8,9] from 88 to 81. Then your answer to part C will look as follows:

Remove/Add/Change	*From*	*To*	*Weight*
Remove	0	3	-
Add	5	6	20
Change	4	2	+12
Change	8	9	-7

CS302 Final Exam, December 9, 2015 - James S. Plank - Page 4

Question 6

You are employing Dijkstra's algorithm to find the shortest path between two nodes. The class definitions of nodes and edges are as follows:

```
class Edge {
  int From;
  int To;
  double *Weight;
};
```

```
class Node {
  int Id;
  vector <Edge *> Adj;       /* Adjacency List */
  double D;                   /* Shortest distance to starting node */
  Edge *BE;                   /* Back edge on the shortest path to the starting node.
  multimap <double, int>::iterator Ptr;
};
```

You'll note, edges are accessed on nodes' adjacency lists using pointers. Nodes, on the other hand, are referenced by integer id's. Although I don't show it, the **Graph** class has a vector of pointers to nodes, and node id's are indices into that vector.

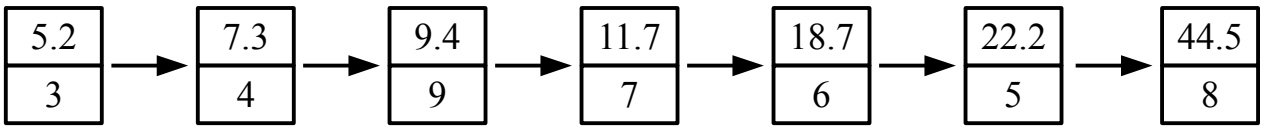
Part A: (This is a short answer that you write on the answer sheet) -- Explain to me in one or two sentences why nodes have the **Ptr** field.

Part B: You are in the middle of Dijkstra's algorithm. The picture on the next page depicts the state of the multimap and the node structs (minus **Ptr**) for nodes 3, 4, 5, 6 and 7. Your job is to process two iterations of Dijkstra's algorithm - in other words, process two entries on the multimap. Then, on the on the answer sheet:

- Show me the new multimap.
- Fill in all of the values of **D** for nodes 3, 4, 5, 6 and 7.
- Draw arrows for all of the values of **BE** for nodes 3, 4, 5, 6 and 7.

You'll note, the graph contains more nodes and edges than are in the picture. However, you have enough information to complete this problem.

Multimap



Nodes

Edges