Name: PID:

Solutions

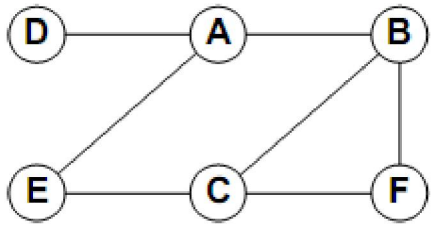
COT5405 Design & Analysis of Algorithms

First Midterm Exam – 10/04/2010

Problem	Points	Points Received
1 a	15	
1 b	10	
2 a	10	
2 b	5	
2 c	10	
3 a	5	
3 b	10	
3 c	10	
4 a	10	
4 b	15	
Total	**100**	

Name: PID:

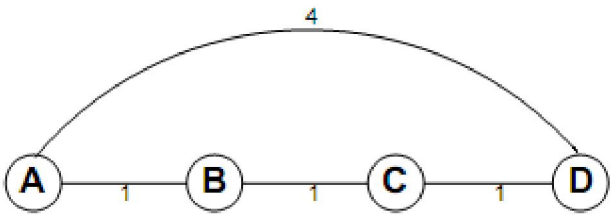
1. a) [15 points] Perform breadth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Fill in the table below showing the order in which the vertices are visited and the queue contents after processing each node. Draw the breadth-first search tree which results from running breadth-first search on the graph.



Order of visitation	Queue contents after processing node
-	[A]
A	**[B D E]**
B	**[D E C F]**
D	**[E C F]**
E	**[C F]**
C	**[F]**
F	**[]**

b) [10 points] Breadth-first search *can* be used for finding shortest paths in a graph under certain conditions. List what conditions are necessary for breadth-first search to find the shortest path in a graph. Draw a *small* graph (no more than 4 vertices), clearly labeling all vertices and edge weights, for which breadth-first search does *not* find a shortest path; say which path that is, what the cost of the path is, and what should be the shortest path and cost.

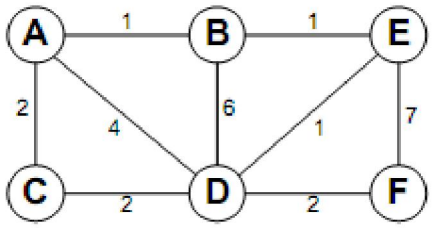
All edges in the graph must have the same weight (i.e. an unweighted graph or a unit-weight graph).



Assuming you start from A, BFS tells you shortest cost from A to D is 4 by the path A->D; however, the actual shortest cost from A to D is 3 by the path A->B->C->D.

Name: PID:

2. Suppose Dijkstra's algorithm is run on the following graph, starting at node *A*.

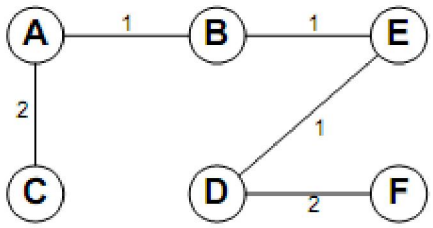


a) [10 points] Draw a table showing the intermediate distance values of all the nodes after each iteration of the algorithm.

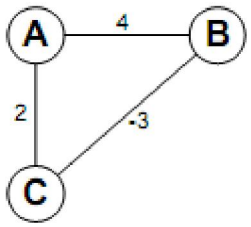
Node	Iteration					
	0	1	2	3	4	5
A	**0**	0	0	0	0	0
B	∞	**1**	1	1	1	1
C	∞	2	**2**	2	2	2
D	∞	4	4	4	**3**	3
E	∞	∞	2	**2**	2	2
F	∞	∞	∞	∞	9	5

(The underlined value is the min unused node pulled out for the next iteration)

b) [5 points] Draw the final shortest-path tree.

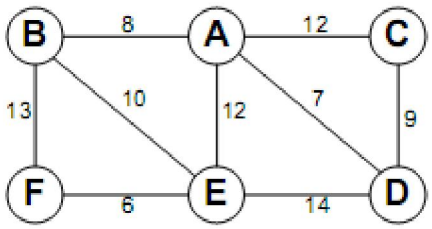


c) [10 points] One of the assumptions with Dijkstra's algorithm is that the input graph has all positive edge lengths. If this is not true, then Dijkstra's algorithm may fail to find shortest paths. Fill in the edge weights on the graph below so that the algorithm will fail, assuming you start from node *A*. All edge weights should be integers, and there should be no negative cycles.



Name: PID:

3. Consider the following graph.



a) [5 points] What is the cost of its minimum spanning tree?

40

b) [10 points] Suppose Prim's algorithm is run on the graph with source node *A*; whenever there's a choice of vertices, pick the one that is alphabetically first. List the order in which the edges are added.

Iteration	Edge Added
1	AD
2	AB
3	CD
4	BE
5	EF

c) [10 points] While the weight of a minimum spanning tree is always unique, the tree itself is not guaranteed to be unique. However, if all the edge weights of the graph are unique, then so will its minimum spanning tree. Give an intuitive argument (i.e. you do *not* need to give a formal proof) as to why this is the case.

Argument 1 – More informal

Consider Kruskal's algorithm. The first thing it does is sort the edges, and this ordering will be unique if all the edge weights of the graph are unique. Since Kruskal's does in fact find a minimum spanning tree of the graph, then this tree must be unique, since the algorithm will never make an arbitrary choice, and thus it can only have one possible outcome.

Argument 2 – More formal

If there were to exist 2 trees X, Y with w(X) = w(Y) but X ≠ Y, then each would have at least one unique edge. However, if this is the case, then swapping these two edges in each tree would result in two different spanning trees, one with a smaller weight than before, contradicting that it was a MST.

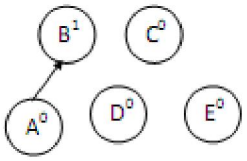
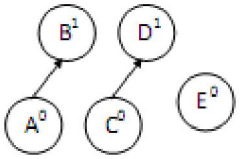
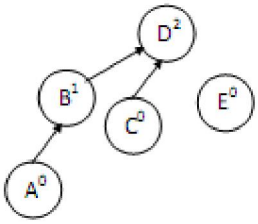
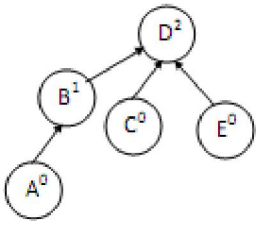
Name: PID:

4. Suppose you are running the disjoint-set algorithm on 5 initially disjoint nodes: *A, B, C, D, E.* Show the state of the sets after each of the following unions:

   1) (A,B)
   2) (C,D)
   3) (B,C)
   4) (A,E)

   when (a) path compression is *not* used and (b) path compression is used. Whenever you have a choice, make the lexicographically smaller letter a child. Show the ranks of each node after each step.

   a) [10 points] Not using path compression

After union(A,B)	After union(C,D)	After union(B,C)	After union(A,E)
			

   b) [15 points] Using path compression

After union(A,B)	After union(C,D)	After union(B,C)	After union(A,E)