Homework 5

Due: Wednesday November 17, 2010

1. For each of the following, either draw a graph satisfying the given criteria or explain why it can't be done. Your graphs should not have any multi-edges (more than one edge between the same pair of vertices) or self-loops (edges with both ends at the same vertex).

(a) Draw a graph with 3 connected components, 12 vertices, and 18 edges.
(b) Draw a graph with 3 connected components, 12 vertices, and 8 edges.
(c) Draw a graph with 3 connected components, 12 vertices, and 50 edges.
(d) Draw a graph with three vertices of degree 3, and four vertices of degree 2.

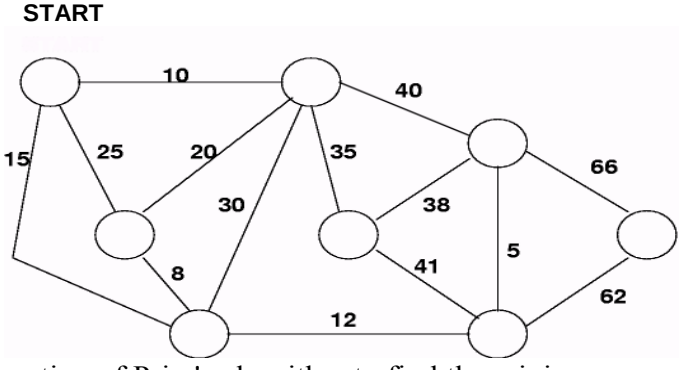
2. Would you use the adjacency list or the adjacency matrix representation in each of the following cases? Justify your choice.

(a) The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.
(b) The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.
(c) You need to answer as fast as possible the query areAdjacent, no matter how much space you use.

3. Would you prefer DFS or BFS (or both equally) for the following tasks? (Assume the graph is undirected and connected.) Justify your answer.

(a) Finding a path to a vertex known to be near the starting vertex
(b) Finding the farthest vertex from a given vertex, where the farthest vertex is one with the longest minimum-length path (in other words, to find the farthest vertex from *u*, take the shortest paths from *u* to every other vertex *v* in the graph and report the *v* for which the shortest path is the longest)
(c) Finding the connected components of the graph

4. In this problem you will show execution of the minimum spanning tree algorithms that you studied in class on the following graph:



(a) Trace the execution of Prim's algorithm to find the minimum spanning tree for this graph. At each step, you should show the vertex and the edge added to the tree and the resulting values of *D* after the relaxation operation. Use START vertex as the first vertex in your traversal.

(b) Trace the execution of Kruskal's algorithm to find the minimum spanning tree for this graph. Give a list of edges in the order in which they are added to the MST.

5. (a) Give an example of a weighted directed graph *G* with negative-weight edges, but no negative-weight cycle, such that Dijkstra's algorithm incorrectly computes the shortest-path distances from some vertex *v*. Trace the execution of Dijkstra's algorithm to show where it goes awry.

(b) Consider the following greedy strategy for finding a shortest path from vertex *tart* to vertex *goal* in a given connected graph.

1. Initialize *path* to *start.*
2. Initialize *VisitedVertices* to *{start}*
3. If *start = goal,* return *path* and *exit.* Otherwise, continue.
4. Find the edge *(start,v)* of minimum weight such that *v* is adjacent to *start* and *v* is not in *VisitedVertices.*
5. Add *v* to *path.*
6. Add *v* to *VisitedVertices.*
7. Set *start* equal to *v* and go to step 3.

Does this greedy strategy always find a shortest path from *start* to *goal*? Either explain intuitively why it works, or give a counter-example.

6. An Euler tour of a directed graph G with *n* vertices and *m* edges is a cycle that traverses each edge of G exactly once according to its direction. Such a tour always exists if the in-degree is equal to the out-degree for each vertex in G. Describe an *O(n + m)* time algorithm for finding a Euler tour of such a graph G. Analyze its running time.

7. Describe how to modify the Floyd-Warshall Dynamic Programming formulation algorithm to solve the following problems, all of which involve some sort of all-pairs path problem. Throughout, G = (V, E) is a directed graph, given as an adjacency matrix, in which each edge $(i, j) \in E$ is associated with a numeric weight $w(i,j)$.

In each case give the Dynamic Programming formulation (not a full algorithm) and justify its correctness. Each formulation should lead to an $O(V^3)$ time algorithm. Be sure to specify how you assign the $w_{i,j}$ values if *(i,j)* is a self-loop or nonexistent edge.

(a) In this problem, the weight *w(i,j)* denotes the probability that the edge *(i,j)* exists. (Nonexistent edges of G are treated having probability 0.) Each weight is in the range from 0 to 1. The probability that a path exists is defined to be the product of the edge probabilities along the path. For each pair of vertices, $i,j \in V$, compute the (probability of) the maximum probability path from *i* to *j*, denoted $p_{i,j}$.

(Note: You may discover that there is a way to reduce this problem directly to the Floyd-Warshall algorithm, but I would like you to solve this problem by dynamic programming.)

(b) Assume in this case that G is a DAG, and *w(i,j)* = 1 if $(i,j) \in E$ and *w(i,j)* = 0 otherwise. Two paths are said to be distinct if they differ in any way. For each pair of vertices $i,j \in V$, compute the total number of distinct paths from *i* to *j*, denoted $C_{i,j}$.

Why did we need to assume that G is a DAG? (Hint: Pay special attention to the basis case here, because it is easy to get it wrong.)

8.(Optional) The objective of this problem is to write a dynamic programming algorithm to play a game. Two players, called Jen and Ben alternate in taking moves, with Jen always going first. Initially the board consists of three piles of diamonds, which we denote (*A, B, C*), meaning that there are *A* diamonds in the first pile, *B* in the second pile, and *C* in the third. The board always consists of three numbers that are nonnegative. During a move a player can do any one of the following:

(1) Remove 1 diamond from pile 1.
(2) Remove either 1 or 2 diamonds from pile 2.
(3) Remove either 2 or 3 diamonds from pile 3.

The first player who cannot make a move loses. (And the winner gets all the diamonds.) That is, if it is a player's turn to move and the board is either (0,0,0) or (0,0,1) then he/she loses.

Write an algorithm that will, given numbers (*A, B C*), determine whether Jen or Ben has a winning strategy, given this starting board and the assumption that Jen plays first. That is, your algorithm should determine which player wins, assuming both players play as well as is possible.

For example, suppose that the initial board is (0, 1, 4). In this case Jen can force a win. She uses rule (3) to remove 2 diamonds from pile 3, resulting in the board (0, 1, 2). Ben's only choices are to remove 1 from pile 2 or 2 from pile 3, resulting in the possible boards (0, 0, 2) and (0, 1, 0). In either case, Jen can remove the final diamonds (using either rules (3) or (2)) leaving Ben without a move.

Derive a dynamic programming formulation to determine the winner, given the initial board (*A, B, C*). Justify the correctness of your formulation. (You do not need to give an entire algorithm, just the DP formulation.) Argue that the running time (if implemented) would be a polynomial function of *A, B,* and *C*.

Hint: Consider a boolean table W[a, b, c], where $0 \le a \le A$, $0 \le b \le B$, and $0 \le c \le C$. The entry W[a, b, c] is true if a player making the next move with this board can force a win and false otherwise. Consider the results of all possible moves starting from this board, assuming that the results of all smaller boards are already computed.