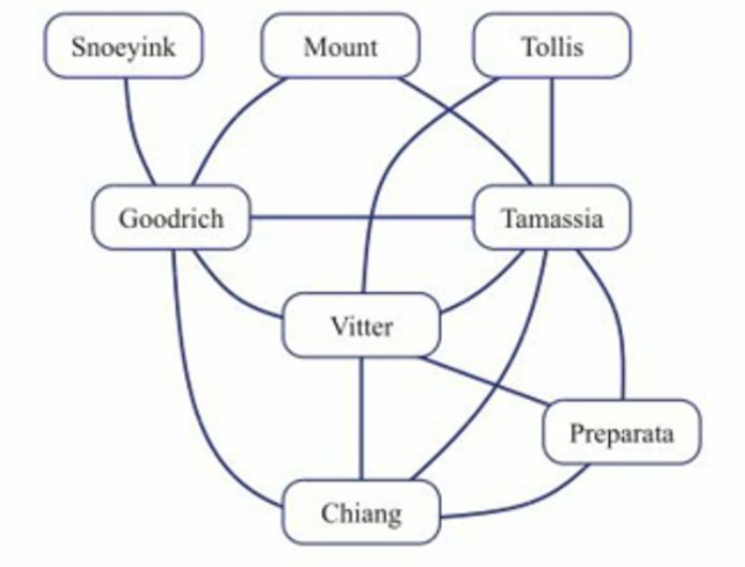
NAME:

Homework #5: Graphs and Graph Search

1. (10 points) In the graph below, each node is the name of a computer science researcher. Two authors are connected by an edge if they have co-authored a paper.



Draw the adjacency list and adjacency matrix representations of the graph.

Solution

Adjacency List

Vertex	**Neighbor List**
Snoeyink	Goodrich
Mount	Goodrich, Tamassia
Tollis	Vitter, Tamassia
Goodrich	Snoeyink, Mount, Tamassia, Vitter, Chiang
Tamassia	Mount, Tollis, Goodrich, Vitter, Chiang, Peparata
Vitter	Goodrich, Tollis, Tamassia, Preparata, Chiang
Preparata	Tamassia, Vitter, Chiang
Chiang	Goodrich, Vitter, Tamassia, Preparata

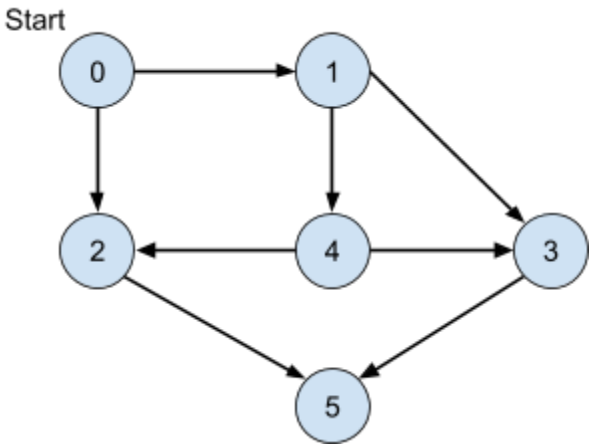
NAME:

Adjacency Matrix

	Sno	Mou	Tol	Goo	Tam	Vit	Pre	Chi
Sno	0	0	0	1	0	0	0	0
Mou	0	0	0	1	1	0	0	0
Tol	0	0	0	0	1	1	0	0
Goo	1	1	0	0	1	1	0	1
Tam	0	1	1	1	0	1	1	1
Vit	0	0	1	1	1	0	1	1
Pre	0	0	0	0	1	1	0	1
Chi	0	0	0	1	1	1	1	0

NAME:

2. (10 points) Consider the following directed graph:



   a. Show all steps in a depth-first search of the graph, starting with vertex zero ("Start"). Show the order in which the vertices are visited and indicate when the algorithm backtracks. If a vertex has more than one neighbor, visit them in numeric order (e.g. starting at 0, visit 1 first since 1 < 2).

   b. Using the same graph, show all steps in a breadth-first search. Show how the queue is initialized. For each iteration, indicate which vertex is dequeued, identify its neighbors, and show the new contents of the queue. Compare the path from 0 to 2 found by DFS with that found by BFS.

Solution

Part (a)

Vertex	**Comments**
0	Start. Will visit vertex 1 next.
1	
3	
5	Backtrack to 3 (no additional neighbors), backtrack to 1
1	
4	3 already visited, so visit 2 next
2	5 already visited. Backtrack 3 steps to 0

0	2 already visited. Done

The DFS path from 0 to 2 is (0, 1, 4, 2).

Part (b)

Vertex		**Queue**
	Initialization	0:Nil (vertex: π)
0	Add 1 & 2 to queue	1:0, 2:0
1	Add 3 & 4 to queue	2:0, 3:1, 4:1
2	Add 5 to queue	3:1, 4:1, 5:2
3	5 already in queue	4:1, 5:2
4	2, 3 already visited	5:2
5	No neighbors	**Empty**

The BFS path from 0 to 2 is reconstructed using the π values. Since $2.\pi = 0$, the path is just (0, 2). The BFS path is optimal in terms of number of steps in the graph; the DFS path is not.

NAME:

3. (10 points) A graph is *simple* if it does not have any parallel edges or self-loops. Show that DFS is $O(n^2)$ for a simple graph with n vertices represented as an adjacency matrix.

   Solution

   The point of this problem is that finding all the neighbors of a vertex v using an adjacency matrix takes $\Theta(n)$ time, whereas it only takes $\Theta(\deg v)$ using an adjacency list representation. Remember how an adjacency matrix is constructed:

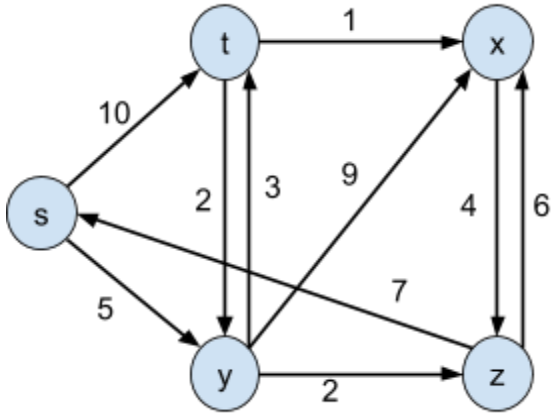
   - Label the vertices $0, 1, \ldots, n-1$ arbitrarily.
   - Create an n-by-n matrix A and initialize the entries to zero.
   - If there is an edge (u, v) in the graph, set $A[u, v] = 1$.

   So, if I want to find the neighbors of some vertex u, I have to iterate over the row $A[u,]$; that is I have to iterate over n entries, which is $\Theta(n)$.

   Now, in DFS, we visit every vertex at most once, so there are $O(n)$ visits, each of which requires $\Theta(n)$ time to check the neighbors, so the entire running time is $O(n^2)$.

NAME:

4. (10 points) Demonstrate the operation of Dijkstra's Algorithm on the following weighted, directed graph. Use vertex "s" as the source (start) vertex.



Describe the initialization phase of the algorithm. For each iteration, indicate which vertex is removed from the min-queue, the updated vertex distances, and the new contents of the min-queue. Track the predecessors for each vertex. Construct the least-weight path from "s" to "x" using the vertex predecessors.

Solution

There are two ways to do this. The examples on the slides only put nodes into the min-queue as they are visited; alternatively, you can start with all the elements in the min-queue. Either way, you would need some data structure, e.g. an array, to keep track of the status of each node (unexamined, enqueued, visited).

<table>
<tr><th>Vertex</th><th>Comment, Distance, Predecessor</th><th>Min-Queue</th></tr>
<tr><td></td><td>Initialization
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>∞</td><td>∞</td><td>∞</td><td>∞</td></tr>
<tr><td>.π</td><td>Nil</td><td>Nil</td><td>Nil</td><td>Nil</td><td>Nil</td></tr>
</table></td><td>s:0 (vertex:d)</td></tr>
</table>

NAME:

<table>
<tr>
<td>0</td>
<td>Relax edges (s,t) and (s,y)
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>10</td><td>∞</td><td>5</td><td>∞</td></tr>
<tr><td>.π</td><td>Nil</td><td>s</td><td>Nil</td><td>s</td><td>Nil</td></tr>
</table></td>
<td>t:10, y:5</td>
</tr>
<tr>
<td>y</td>
<td>Relax edges (y,t), (y,x), and (y,z)
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>8</td><td>14</td><td>5</td><td>7</td></tr>
<tr><td>.π</td><td>Nil</td><td>y</td><td>y</td><td>s</td><td>y</td></tr>
</table></td>
<td>t:8, x:14, z:7</td>
</tr>
<tr>
<td>z</td>
<td>Relax (z,s) (no change) and (z,x)
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>8</td><td>13</td><td>5</td><td>7</td></tr>
<tr><td>.π</td><td>Nil</td><td>y</td><td>z</td><td>s</td><td>y</td></tr>
</table></td>
<td>t:8, x:13</td>
</tr>
<tr>
<td>t</td>
<td>Relax (t,x). y is already visited.
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>8</td><td>9</td><td>5</td><td>7</td></tr>
<tr><td>.π</td><td>Nil</td><td>y</td><td>t</td><td>s</td><td>y</td></tr>
</table></td>
<td>x:9</td>
</tr>
<tr>
<td>x</td>
<td>z has already been visited
<table>
<tr><td></td><td>s</td><td>t</td><td>x</td><td>y</td><td>z</td></tr>
<tr><td>.d</td><td>0</td><td>8</td><td>9</td><td>5</td><td>7</td></tr>
<tr><td>.π</td><td>Nil</td><td>y</td><td>t</td><td>s</td><td>y</td></tr>
</table></td>
<td>Min-queue is empty.</td>
</tr>
</table>

To reconstruct the path from “s” to “x”, start with “x” and trace the “.π” values back to the source vertex “s”: x ← t ← y ← s.

NAME:

5. (10 points) Suppose that rather than using a min-heap to implement the priority queue Q used in Dijkstra's algorithm, we instead used an unsorted sequence implementation of the priority queue. Why in this case is the best-case running time of Dijkstra's algorithm $O(n^2)$ on an n-vertex graph?

   Solution

   Suppose, as in the pseudocode in the textbook, we start with all n vertices in the priority queue. Each vertex will eventually be extracted from the queue. To extract the first vertex, we must search an n-long array to find the vertex with the smallest "d" value. For the second vertex, we must search an $n-1$-long array; for the third, an $n-2$-long array, etc. Summing all the array lengths gives

   $$n + (n-1) + (n-2) + \cdots 3 + 2 + 1 = \tfrac{n(n+1)}{2} = \Theta(n^2)$$

   Not all the min-queue extractions will require searching the entire list, so we can only say that the total running time of all the min-queue extractions is $O(n^2)$.

   The running time of Dijkstra's algorithm was given in terms of the cost of the min-queue extractions plus the cost of all the edge relations, which, when using a min-heap is

   $$n \cdot O(\log n) \; + \; m \cdot O(\log n) = O((m+n)\log n)$$

   What we've shown is that, using the unsorted sequence, the cost of the n min-queue extractions is $O(n^2)$. That still leaves the cost of the m relaxations. However, using an unsorted sequence, relaxation is $O(1)$ since there is no heap property to be maintained. So the total running time is $O(m+n^2)$ which is $O(n^2)$ so long as m is "not too big," meaning m is $O(n^2)$. This will be true for most reasonable graphs, including simple graphs.