Michael Kaiser
Assignment 7
Comp 510
Fall 2012

23.1-1
Let (u, v) be a minimum-weight edge in a graph G. Show that (u, v) belongs to some minimum spanning tree of G.

We can use Theorem 23.1 to prove this: let G = (V, E) be a connected, undirected graph with real-valued weights defined on E. Let A be a subset of E (the empty set in this case) that is included in some minimum-spanning-tree for G, and let (S, V – 2) be any cut of G that respects A, and let (u, v) be a light edge crossing (S, V - S). If S is any set containing u but not v, and the edge (u,v) is safe for A (by this theorem), then we have proven that (u, v) belongs to some minimum sapnning tree of G.

23.1-6
Show that a graph has a unique minimum spanning tree if, for every cut of the graph, there is a unique light edge crossing the cut. Show that the converse is not true by giving a counterexample.

Assume that for every cut of G, there is a unique light edge crossing the cut. Consider two minimum-spanning-trees of G, T and T'. If we remove any edge (u,v) from T, T becomes disconnected, resulting in a cut (S, V – S). We can prove that the edge (u,v) is a light edge crossing the cut. Consider an edge (x,y) in T' that crosses (S, V – S), and it must also be a light edge crossing the cut. Because the light edge crossing the cut is unique, the edges (u,v) and (x,y) must be the same. Thus, (u,v) is in T', and because any edge can be selected as (u,v,), we have proven that T and T' are the same, and is an unique minimum-spanning-tree of G.

The converse is not true, consider this graph:
(x)
  \ <= 1
  (y)
  / <= 1
(z)

If given the cut ({y}, {x,z}), both the edges (x,y) and (y,z) would generate a minimum-spanning-tree.

23.2-1
Kruskal's algorithm can return different spanning trees for the same input graph G, depending on how ties are broken when the edges are sorted into order. Show that for each minimum spanning tree T of G, there is a way to sort the edges of G in Kruskal's algorithm so that the algorithm returns T.

To obtain T from Kruskal's algorithm, we simply sort the edges with the same weights. Edges of the same weight should have those edges which are in T selected over those that are not. This will result in T being generated.

23.2-2
Suppose that the graph G = (V, E) is represented as an adjacency matrix. Give a simple implementation of Prim's algorithm for this case that runs in $O(V^2)$ time.

```
for i = 1 to V
        dist[i] = infinity;
        pred[i] = infinity;
dist[root] = 0;
Create min-priority-queue Q for indexes of verticies based on values of dist
while(Q->notEmpty)
        i = Q->EXTRACT-MIN
        for(j = 1 to V)
                if(m[i][j] = 1)
                        if ((i is in Q) && (weight(i,j) < dist[j]))
                                pred[j] = i;
                                dist[j] = weight(i,j);
```

This is $O(V^2)$, as you can deduce from the for loops.

24.1-1

Run the Bellman-Ford algorithm on the directed graph of Figure 24.4, using vertex z as the source. In each pass, relax edges in the same order as in the figure, and show the d and π values after each pass. Now, change the weight of edge (z, x) to 4 and run the algorithm again, using s as the source.

First run:

	x	s	t	x	y	z
d|		2	5	6	9	0
p|		z	x	y	s	source

Second run:

	x	s	t	x	y	z
d|		0	2	2	7	-2
p|		source	x	z	s	t

However, the second run has a negative weight cycle.

24.1-3

Given a weighted, directed graph G = (V, E) with no negative-weight cycles, let m be the maximum over all pairs of vertices u, v in V of the minimum number of edges in a shortest path from u to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in m + 1 passes, even if m is not known in advance.

To accomplish this, we need to simply monitor changes:

```
BELLMAN-FORD(G, w, s)
INIT-SINGLE-SOURCE(G, s)
change = true;
while(change == true)
        change = false;
        for each edge(u,v) in G->E
                change = RELAX(u, v, w);

RELAX(u,v,w)
if(v.d > u.d + w(u,v))
```

```
v.d = u.d + w(u, v);
v.pi = u;
change = true;
```

24-2

A d-dimensional box with dimensions ($x_1, x_2,..., x_d$) nests within another box with dimensions ($y_1, y_2,..., y_d$) if there exists a permutation π on {1, 2,..., d} such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2,..., x_{\pi(d)} < y_d$.

a. Argue that the nesting relation is transitive.

b. Describe an efficient method to determine whether or not one d-dimensional box nests inside another.

c. Suppose that you are given a set of n d-dimensional boxes {$B_1, B_2,..., B_n$}. Describe an efficient algorithm to determine the longest sequence ($B_{i1}, B_{i2}, ... B_{ik}$) of boxes such that nests within $B_{i\,j+1}$ for j = 1, 2,..., k - 1. Express the running time of your algorithm in terms of n and d.

a)

Given a permutation π, if $x_{\pi(1)} < y_1$, $x_{\pi(2)} < y_2$, ... and so on, and if $y_{\pi(1)} < z_1$, $y_{\pi(2)} < z_2$, ... and so on, we can determine that $x_{\pi(1)} < z_1$, $x_{\pi(2)} < z_2$, ... and so on. Therefore $(x_1,x_2,...,x_d)$ nests within $(y_1,y_2,...,y_d)$, and $(y_1,y_2,...,y_d)$ nests within $(z_1,z_2,...,z_d)$ is true and transitive.

b)

Take the dimensions of the box and sort them. Then iterate through each dimension and ensure that the dimensions of the first box is always larger than the dimensions of other. If this is ever not true, then it is not possible to nest the second box within the first.

c)

Sort the order of the dimensions for all boxes
Sort the boxes in order of size (compare each box as per part B)
For each box, look at this box and the next box, if it does not nest, we exclude the first box, and continue.
We are then left with the longest sequence of boxes which nest within $B_{ij}+1$.

The running time for the sorting is $O(nd\ln(d))$ – quick or heap sort
The running time for the second sort is $O(n\lg(n))$ – same
The runningtime for sequencing through the boxes is $O(dn)$.

Thus we have $O(n(d\lg d + \lg(n) + d)$.

References:

Textbook and solution manual
http://student.csuci.edu/~douglas.holmes253/Assignment7.html