Michael Kaiser
Assignment 5
Comp 510
Fall 2012

19-2
Describe how to implement this algorithm using binomial heaps to manage the vertex and edge sets. Do you need to change the representation of a binomial heap? Do you need to add operations beyond the mergeable-heap operations given in Figure 19.1? Give the running time of your implementation.

I'm not going to explain what a minimum-spanning-tree is (I assume we've all taken discrete math), so this is a possible solution for a graph G = (V,E):
-Find the root x with the minimum key in the root list of E_i, then remove x from E_i.
-Make a binomial heap from E_j.
-Reverse the order of x's children and then set the head of E_j to the head of the resulting list.
-Set heap H equal to the union of the two binomial heaps E_i, E_j,
-Return x.

Each step of this algorithm takes O(lg(n)) time, so the end result is O(lg(n)).

22.1-1
Given an adjacency-list representation of a directed graph, how long does it take to compute the out-degree of every vertex? How long does it take to compute the in-degrees?

The out-degree of each vertex can be determined by analyzing each node in the adjacency list once: O(E).
The in-degree of each vertex can be determined by analyzing each element in the adjacency list so we have O(E + V).

22.1-2
Give an adjacency-list representation for a complete binary tree on 7 vertices. Give an equivalent adjacency-matrix representation. Assume that vertices are numbered from 1 to 7 as in a binary heap?

An adjacency-list representation for a complete binary tree may look like this:
[1]->[2]->[3]->x
[2]->[4]->[5]->x
[3]->[6]->[7]->x
[4]->x
[5]->x
[6]->x

An equivalent adjacency matrix matrix is the following:

X	1	2	3	4	5	6	7
1	X	1	1	0	0	0	0
2	1	X	0	1	1	0	0
3	1	0	X	0	0	1	1
4	0	1	0	X	0	0	0
5	0	1	0	0	X	0	0

6	0	0	1	0	0	X	0
7	0	0	1	0	0	0	X

22.1-3
Describe efficient algorithms for computing G^T from G, for both the adjacency list and adjacency-matrix representations of G. Analyze the running times of your algorithms.

The transpose of the adjacency matrix can simply be done by swapping the indicies of the matrix. This is accomplished by setting Matrix[i][j] to TransposeMatrix[j][i]. Therefore this is $O(V^2)$.

The transpose of the adjacency list is done by redrawing the graph in reverse order. A possible algorithm would be:

```
Transpose(V,E, AList)
{
        for each v in V
                for each e in AList[v]
                        TransposeList[e].add(v);
        return TransposeList
}
```

The running time of this algorithm is O(V + E).

References

Third edition of the textbook and solution manual
http://en.wikipedia.org/wiki/Directed_graph#Indegree_and_outdegree