Quiz 2 Topics

Graphs
- adjacency and edge list representations
- DFS, BFS and variations
- active intervals in DFS leading to edge classifications:
  - tree, back, forward, cross
- DAG's, topological order, critical path
- strongly connected components
- DAG meta-graph of strongly connected components
- Prim, Dijikstra,
- Kruskal's algorithm
- Cut Theorem from notes

Other Greedy Problems
- idea of a greedy algorithm
- Huffman encoding

Dynamic Sets
- B-trees
  - Algorithms for search, insert, delete – apply to specific cases
  - Order of algorithms
  - Relation to red-black trees
- Hash tables - algorithms for chained and open (linear and double);
  - memory usage; average, worst case behavior
- union-find

For all the specific algorithms, you should understand them well enough to follow them by hand in small cases. You can test yourself on these routine problems for all the graph algorithms and check yourself by looking at the results in my demonstration programs.

Two harder problems (solutions on next page):

1: The distance between two vertices in an unweighted graph is the minimum number of edges in a path between them. The diameter of a graph is the maximum of the distance between any two points. In particular this problem makes sense for an undirected graph given in neighbor list form that is also a *tree*. Find an O(n) algorithm for such a tree. Hint: Trees are defined recursively. Imagine combining subtrees into a bigger tree. How is the larger problem related to the smaller one? Draw pictures! There are *two* kinds of cases. As a result you need a *more elaborate* recursive problem than the original problem.

2: If you are looking for the minimum path lengths from vertex s, and if all edges have positive weight *except for possibly some edges with s as starting point*, can you use Dijiksta's algorithm to correctly solve for the minimum distances? Give a proof or a counter-example.

Solutions:

1: Choose any node as the root. Then each other node has one neighbor that is its parent and all the other neighbors are its children. Compare the problem for a tree with the problem for the subtrees from each child. The central idea is that the longest is **either** contained in one subtree or it goes through the root of the combined tree. The longest path through the root goes to the maximal depth in two subtrees. Hence the general recursive subproblem finds **both** the longest path in a subtree and the depth of the subtree.

Consider the longest path through the root:
a: if there are no children, the length is 0 and the total depth is 0.
b: If there one or more children, the total depth is 1 + the maximum depth of any child's subtree
  i: If there is just one child, the max length through the root is the depth
  ii: If there are more than one child, the max length through the root is 2+ the sum of the two greatest depths of child trees.
  iii: In either i. or ii: the overall longest path is the max of the longest through the root and all the longest within individual subtrees.

The pseudocode above is sufficient. Python code with a simple tree implementation and testing is in handouts/maxpath.py

All of these operations are O(number of child edges), and since there are only n-1 edges total, the result after all recursions is O(n).

2: First hint: It is true. There are two approaches: modify the Dijiksta proof to show it works in this situation, or transform this problem to one where the original Dijiksta theorem applies.... Both solutions follow:

a. When a closest fringe element has been chosen, say f, the Dijikstra proof relies on looking at the alternate paths to f, whose last crossing from tree to fringe is at another fringe element, g, and then looking at the rest of the path from g to f. In the original proof, all edges have nonnegative weight, so the extension from g to f cannot make the path shorter. This argument still works in the current case, since the part from g to f does not include any edges connecting to the tree, and in the hypothesis the only edges with negative weight are connected to s, which is immediately in the tree.

b. Let d(s, v) be the minimal distance in the original problem for any vertex v. If -k is the minimum weight edge from s to a neighbor, add k to the weight of all the edges from s to a neighbor. Now the original Dijiksta theorem applies. Solve Dijistra with those weights to get minimal distances D(s, v). Since each minimal path has exactly one edge from s to a neighbor, d(s, v) <= D(s, v) – k. If there were a better path for the original problem, this argument could be reversed, to get D(s, v) <= d(s, v) + k. Putting the two arguments together we get d(s, v) = D(s, v) – k.