Homework 4

Anthony Weems

November 3, 2015

1. **Coin Changing**

   (a) Algorithm description

   A simple greedy algorithm for making change from quarters, dimes, nickels, and pennies is as follows:

   Use as many quarters as possible, followed by as many dimes as possible, followed by as many nickels as possible, and a remainder of pennies.

   Assume an optimal solution of coins $\{c_1, c_2, ...c_k\}$ to a sum n. Removing c_1 from the solution is equivalent to the solution for $n - c_1$. If this new subset of coins is not an optimal solution to the $n - c_1$ problem, there exists a solution with fewer coins, say j. If this imaginary solution did exist, we could add c_1 back in and have a better solution to n. This represents a contradiction, as we began with an optimal solution to n. This shows optimal substructure of the algorithm.

   Next, we know order does not matter in our solution, so we must show that the greedy choice exists somewhere in the optimal solution. Assume the greedy choice is not in the optimal solution. This means a high value coin (less than n) is not in the solution. This also means that multiple lower value coins are used to make change for this high value coin. We could replace these multiple low value coins with our high value coin to yield a better solution. This shows the greedy choice property, therefore, the greedy coin algorithm must produce an optimal solution.

   (b) (1, 7, 15) making change for 21

   - Greedy: 15,1,1,1,1,1,1
   - Optimal: 7,7,7

2. **Another Interval Problem**

   (a) Algorithm description

   Select the interval v with the greatest right endpoint, reachable from your current position. Update your current position to the right endpoint of v. Continue this algorithm until you reach the end of interval X.

   (b) Optimal Substructure

   This algorithm exhibits optimal substructure by updating the current position and ignore all overlapped intervals as a result. This reduces the problem to a smaller problem.

   (c) Greedy Choice Property

   Assume the greedy choice is non-optimal, i.e., the optimal path does not contain a starting interval with the greatest right endpoint. The beginning interval of this solution can be replaced with the interval with the greatest right endpoint and be either as or more efficient.

3. **Greedy Country Road**

   (a) Algorithm description

   As we move along the road, eventually we will reach a point at which there is a house exactly 4 miles west of us. Place a base station at this point and "ignore" all houses covered by the placed station. Continue along the road.

   (b) Optimality

   We assign $G = \{g_1, g_2, ..., g_n\}$ to be the set of base stations placed by the greedy algorithm and $O = \{o_1, o_2, ..., o_k\}$ to be the base stations of an optimal solution. Assume each set is sorted from west to east. We will prove G is equal or better than T by induction. For $i = 1$, g_1 is the greatest point that covers the western-most house and therefore $g_1 \geq o_1$. Assume $g_n \geq o_n$ for some n. This means that all houses less than g_n are covered. The greedy algorithm selects g_{n+1} to be as high as possible, meaning $g_{n+1} \geq o_{n+1}$. By induction, the greedy algorithm is always equal to or better than the optimal solution and the greedy solution cannot perform better than the optimal solution, therefore, the greedy solution is equivalent to the optimal solution.

4. **Single Source Shortest Paths**

   (a) Runtime of Dijkstra's algorithm is $O(V \log V)$ because we have a tree in which $O(E) = O(2V)$.

   (b) We could simply run DFS on the root of the tree. If v is a descendant of u, we simply take the path directly down the tree from u to v. Otherwise, the shortest path must require visiting the root node by using a loop to the root. First we find the leaf with the shortest path to the root. We can do this by adding a the root as a "leaf" of each leaf nodes in the tree and performing BFS and taking the shortest path. Once we are at the root, we take the shortest path to v from our earlier DFS. The cost of DFS and BFS are both $O(V)$, which means the overall cost of $O(V)$.

5. **Shortest Path**

   The path between a pair of verices in a minimum spanning tree is not the shortest path between the two vertices in the full graph.

   Imagine a graph with edges $\{(A, B), (B, C), (C, D), (D, A)\}$ and weights $\{1, 2, 3, 4\}$. The minimum spanning tree has pairs $\{(A, B), (B, C), (C, D)\}$ with a path weight of 6, however, the minimum path between nodes A and D has weight 4.

6. **Minimum Spanning Trees**

   We simply run BFS to create some spanning tree of G and examine the possible 9 edges that not included in the spanning tree. Find the cycle formed by each edge, e, and find the edge in the cycle with the greatest weight, h. Replace the heavy edge, h, with the original edge, e, within our spanning tree. The runtime of this algorithm is $O(V)$ since $E = V + 8$.

7. **Huffman Coding**

   Suppose a binary tree that is not full can correspond to an optimal prefix code. Such a tree contains a node with exactly one child, c. We can create a tree that is represents a more optimal code by replacing the parent of c with c. If the child codes an element that is used, the code for such an element will be shorter (by one) than our original tree. This is a contradiction, therefore, a binary tree that is not full cannot correspond to an optimal prefix code.