Nathan Newbury Hw 1 Writeup

i) Recursive Formula

This recursive formula is below:

```
(1) For pv in parent[v]:
    ancestor = ancestor | LCA[pu][v]
(2) For pu in parent[u]:
    ancestor = ancestor | LCA[u][pv]

(3) LCA[u][v] = ancestor (We will replace this line with more logic).
```

The LCA dictionary records the least common ancestors for 2 nodes. In the code above, ancestor (the ancestor-list) is a set of all ancestors of nodes u and v (we will reduce it to the least common ancestor later). Pu and pv are the parents of node u and node v respectively. The ancestors of the parent nodes are also ancestors of our nodes. Within this set of ancestors, unless the nodes fit the base cases, there is the least common ancestor.

To reduce our ancestor list to only the least common ancestors we use the following logic instead of (3) above:

```
(4) For n1 in ancestor:
            For n2 in ancestor:
                If n1 != n2:
                   ancestor = ancestor – LCA [n1][n2]
    LCA[u][v] = ancestor
```

With (4) above we reduce our ancestor-list only to the least common ancestors before saving it in our dictionary. For any two nodes in our ancestor-list, if they are equal, the LCA will also be equal. We would remove every ancestor from our list. By adding the conditional n1!=n2, we avoid this. If n1 and n2 are not equal, we remove every node from our ancestor-list that is an ancestor of another node in our list. In this way, no node in our ancestor-list is an ancestor of any other node in the list. Only the least common ancestors remain. We then save our ancestor-list into our dictionary for the next recursion.

ii) Base Cases

(5) if u==v:

LCA[u][v] = {u}

Return LCA[u][v]

If both u and v are pointing to the same node, the LCA is also that node. This fills out the “diagonal” or=f our LCA dictionary. Additionally, our recursive algorithm traverses up our DAG one node at a time. Eventually, the two nodes will be equal, and we can exit our recursion.

(6) If parent [u] == empty

LCA[u][v] = {u}

Return LCA[u][v]

(7) If parent [u] == empty

LCA[u][v] = {u}

Return LCA[u][v]

Any node with no parents is the first node of our DAG, and thus their LCA is themselves. Our recursive algorithm traverses up our DAG one node at a time. If we reach the top of our DAG, then the least common ancestor is the first node, and we can exit our recursion.

iii) Asymptotic Running Time

This run time complexity calculation assumes that dictionary lookup and writing is done in constant time. We have |E| edges and |V| vertices where |V|<|E|<|V|^2. |E| represents the lookups in our LCA dictionary and |V| represents the subproblems with a degree of d. Each subproblem is solved in time O(P(d)), where P(d) is a polynomial of d. We traverse every edge. Thus, time complexity is O(|E| + |V|*P(d)). Our subproblem is comprised of loops in (1), (2) and (4), which are on the order of number of parents per node and total number of ancestors. One sub problem has the time complexity of O(d^2(nested loop)). Therefore the total time complexity is O(|E| + |V|d^2). The degree d is usually small, and |V| is much smaller than |E|. We can approximate the time complexity as O(|E|).