HW9

1. ***Give an iterative implementation of the dynamic programming solution for finding the cost of an optimum solution.***

   The recursive formulation is given by

$$MergeList(i,k) = \begin{cases} min_{i\le j\le k}\{MergeList(i,j) + MergeList(j+1,k)\} + \sum_{j=i}^{k} l_j & i < k \\ 0 & i = k \end{cases} \tag{1}$$

   The pseudocode is

> // $SumOfLength[i,j]$ used for storing $\sum_{j=i}^{k} l_j = \sum_{j=1}^{k} l_j - \sum_{j=1}^{i-1} l_j = SumOfLength[1,j] - SumOfLength[1,i-1]$
> // $Cost[i,j]$ used to store the minimal total recursive cost. We initialize $Cost[i,j] = 0$ for all $1 <= i,j <= n$

```
MergeList(n)
for l<-2 to n do //l is to record the minimal cost when merging l items
  for i<-1 to n-l+1 do
    k = i+l-2;
    Cost[i,k] = maxint;
    for j<-i to k do
      Cost[i,k] = min{Cost[i,k],Cost[i,j]+Cost[j+1,k]+SumOfLength[i,k]}
```

2. ***a. Observe that MinEdit(i, j) ≥ |j − i|.***

> The difference between the length of u and v is $|i-j|$. Assume except for the segment $|i-j|$, all of the other parts are the same, i.e., the edit distance is 0. Hence, the minimum number of edits will be obtained by deleting all of the characters in the segment $|i-j|$. Hence, $MinEdit(i,j) \ge |j-i|$.

b. By limiting which recursive calls are made, improve the running time of the efficient recursive algorithm to O((n + m)k). (In fact, a bound of O(min{n, m} · k) can be achieved.) It suffices to give the recursive expression for MinEdit. You also need to explain why your algorithm has this running time.

> $$MinEdit(n,m) = \begin{cases} MinEdit(n-1,m-1) & u_n = v_m, n, m \\ min\{1+MinEdit(n,m-1), 1+MinEdit(n-1,m-1), 1+MinEdit(n-1,m)\} & u_n \ne v_m, n, m \\ m & n = 0 \\ n & m = 0 \end{cases}$$
>
> We initialize a global value k to store the minimum number of edits in the running time. If the number of edits in some of the function call is larger than k, break the function call and make it maxint to reduce the recursive call having this.
>
> For example, assume $n < m$, if all of the characters in range$[1,n]$ such that $u_n = v_m$, then there will be n subproblems, and the minimum edit cost will be $m-n$, which fits $O((n+m)k)$.
>
> If $u_n \ne v_m$, it can be considered into three subproblems and get the minimum, i.e., $min\{1+MinEdit(n,m-1), 1+MinEdit(n-1,m-1), 1+MinEdit(n-1,m)\}$, we start with any of them and continue, we will get a number of edits k that changes u to v. As for the other two subproblems, we only calculate the cost in their runtime, if it is less than k continue, else break the function call. If the final result is less than k, reassign the k with the smaller cost.
>
> To optimize it, we set k with m (Maybe the intersection of the characters in u and v is None). And calculate the cost for the $minEd[i,j]$. If $u_m \ne u_v$, we at most calculate k steps, because each time when $u_n \ne v_m$, we will plus the cost with 1 until we reach k. If $u_m = u_v$, we at most calculate n times because n will be deducted to 0 after n times. Hence, the total cost is $3cnk = O(nk)$. Because for $MinEdit(n,m), n < m$, n can counts from n to 0, m can count from m to $m-k$. There are nk subproblems, and each cost per subproblem is $O(1)$.
>
> For a more general solution, the total cost is $O(min\{n,m\} \cdot k)$

3. ***Let G = (V, E) be a dag, a directed acyclic graph, and let s ∈ V be a designated vertex. Suppose each edge has a postive integer length. Give a linear time algorithm to find the number of shortest paths from s to each vertex v ∈ V .***

   - According to Additional Problems, we can use the following formula to find the length of the shortest paths from every source v to a designated target vertex $t \in V$ in linear time.

$$\text{Shtst}[v] \leftarrow \begin{cases} \min_{(v,w)\in E}\{\text{ length }(v,w) + \text{Shtst}[w]\} & \text{if } v \ne t \\ 0 & \text{if } v = t \end{cases} \tag{3}$$

   - We can use the reversal of G to compute the length of the shortest paths from the source s to each vertex $v \in V$ in linear time.

   Then in this problem, we use $ShtstN[v]$ to store the number of shortest paths from v to s. For vertex v, each edge (v,w), we can find the w, whose $\text{length }(v,w) + \text{Shtst}[w] = \text{Shtst}[v]$, and store the sum of them in $ShtstN[v]$.

The formula is

$$\text{ShtstN}[v] \leftarrow \begin{cases} \text{ShtstN}[v] + \text{ShtstN}[w] & \text{if } v \neq t \text{ and } Shtst[v] = length(v,w) + Shtst[w] \\ \text{ShtstN}[w] & \text{if } v \neq t \text{ and } Shtst[v] > length(v,w) + Shtst[w] \\ 1 & \text{if } v = t \end{cases} \quad (4)$$

To compute this, we initialize $Shtst[v]$ to maxint for all $v \neq s$, and $Shtst[t]$ to 0, $ShtstN[v]$ to 0 for all $v \neq s$, and $ShtstN[t]$ to 1

```
DFS-ShtN(v)
Done[v] <- True;
for each edge (v,w) do
  if not Done(w) then DFS-ShtN(w)
  end if
  tmp_length <- min{Shtst[v],length(v,w)+Shtst[w]}
  if Shtst[v] = tmp_length then
    ShtstN[v] <-ShtstN[v]+ShtstN[w];
  else if Shtst[v] > tmp_length then
    Shtst[v] <- tmp_length;
    ShtstN[v] <-ShtstN[w];
end for
```

4. ***Let T be a rooted tree. By preprocessing the tree and storing suitable results in space O(\|T\|), enable the following type of query to be answered in O(1) time: given two vertices (nodes) in T answer whether one vertex is an ancestor of the other, and if so which one? Hint. You want to use suitable vertex numberings.***

> Convert the tree to the directed graph and store the preorder and postorder for each item in a hash table where the node is the key and (the numbering by preorder, the numbering by postorder).
>
> Compare two nodes v_1, v_2, if $preoder[v_1] < preoder[v_2]$ and $postoder[v_1] > postoder[v_2]$, i.e., $map[v_1][0] < map[v_2][0]$ and $map[v_1][1] > map[v_2][1]$, then v_1 is the ancestor of v_2.
>
> Search in Hash table takes expected $O(1)$ time and each comparison takes $O(1)$ time as well. Hence, the total time is expected $O(1)$.