CSE531 Homework 3 Solutions

Problem 1 Solution

To find the shortest common superstring of $X[0\dots m]$ and $Y[0\dots n]$, construct an array $C[0\dots m][0\dots n]$, where $C[i][j]$ is the length of the longest superstring of $X[0\dots i]$ and $Y[0\dots j]$, $0 \le i \le m$, $0 \le j \le n$. It should be clear that the following holds:

$$C[i][j] = \begin{cases} j & i = 0 \\ i & j = 0 \\ C[i-1][j-1]+1 & ij \neq 0, X[i] = Y[j] \\ \min(C[i][j-1]+1, C[i-1][j]+1) & ij \neq 0, X[i] \neq Y[j] \end{cases}$$

Fill in $C[i][j]$ in appropriate order and do some backtracing to obtain the answer.

To find the shortest common superstring of $X[0\dots m]$,$Y[0\dots n]$ and $Z[0\dots p]$. Construct the array D where $D[i][j][k]$ is the minimum length of a common superstring of X,Y and Z. We also compute, as above, the shortest common superstring of $(X,Y),(X,Z)$ and (Y,Z), and obtain 2d arrays C_{XY}, C_{XZ} and C_{YZ}. We can use the following relation to get the answer.

$$D[i][j][k] = \begin{cases} C_{YZ}[j][k] & i = 0 \\ C_{XZ}[i][k] & j = 0 \\ C_{XY}[i][j] & k = 0 \\ D[i-1][j-1][k-1]+1 & ijk \neq 0, X[i] = Y[j] = Z[k] \\ \min(D[i-1][j-1][k]+1, D[i][j][k-1]+1) & ijk \neq 0, X[i] = Y[j] \neq Z[k] \\ \min(D[i-1][j][k-1]+1, D[i][j-1][k]+1) & ijk \neq 0, X[i] = Z[k] \neq Y[j] \\ \min(D[i][j-1][k-1]+1, D[i-1][j][k]+1) & ijk \neq 0, Y[j] = Z[k] \neq X[i] \\ \min(D[i][j-1][k]+1, D[i-1][j][k]+1, D[i][j][k-1]+1) & \text{Otherwise} \end{cases}$$

Problem 2 Solution

a) Let $G(n)$ denotes the time needed to compute $T(n)$ using recursion. Obviously, $G(i+1) \ge G(i)$ for every non-negative integer i. For $n \ge 2$, to compute $T(n)$, $T(n-1)$ and $T(n-2)$ are called during the computation. Therefore $G(n) \ge G(n-1) + G(n-2)$. Since $G(n-1) \ge G(n-2)$, we have $G(n) \ge 2G(n-2)$. An lower bound of $G(n)$ can be estimated by $G(n) \ge 2G(n-2) \ge 2^2 G(n-4) \ge \dots$. It follows easily that $G(n) \ge 2^{(n-2)/2}\min(G(0), G(1))$, which indicates an exponential running time.

b) We determine and store the value of $T(i)$ in the order of $T(0), T(1)\dots T(n)$. Since, when computing $T(i)$, we already have the value of $T(0), T(1), \dots T(i-1)$, no recursive call is needed. Computing $T(i)$ from known $T(0), T(1), \dots T(i-1)$ values takes $O(i)$ time. Since $i \le n$ and we need to compute no more than n such $T(i)$ values, it is clear that the running time is now $O(n^2)$.

c) Again we determine and store the value of $T(i)$ in the order of $T(0), T(1) \ldots T(n)$. This time, observe by simple comparasion that $T(i) = T(i-1) + T(i-1)T(i-2)$ for $i \geq 3$. Therefore, It is possible to determine the value of $T(i)$ using constant time if we already know the values of $T(j), j < i$.(In fact we only need $T(i-1)$ and $T(i-2)$.) The running time is now $O(n)$ since we need n iterations.

Problem 3 Solution

Construct an array $C[1 \ldots n][0 \ldots k]$, where $C[i][j], 1 \leq i \leq n, 0 \leq j \leq k$ is the minimum cost of a path goes from s to node i with total delay no more than j, or ∞ if such a path does not exist. Initially, we only know $C[s][j] = 0$ for all j, and $C[i][0] = \infty$ for all $i \neq s$. For $i \neq s$ and $j \neq 0$,

$$C[i][j] = \min_e(\infty, \text{cost}(e) + C[i_e][j - \text{delay}(e)])$$

where e is an edge connecting i and some other node i_e, and we ignore e such that $j - \text{delay}(e) < 0$. We fill in entries $C[i][j]$ in the order of $j = 1, 2, \ldots, k$. In each of the k iterations, all nodes and edges in the graph have to be examined. This results in an $O(k(|E| + |V|))$ running time.

Problem 4 Solution

Consider a array $C[1 \ldots n][1 \ldots k]$, where $C[i][j], 1 \leq i \leq n, 1 \leq j \leq k$, is the minimum maximum total weight of the subpartitions, among all partion schemes that devides the subtree rooted at node i into no more than j parts, by removing no more than $j - 1$ edges. $C[i][j] = \text{weight}(i)$ for every j if i is a leaf node. $C[i][1]$ is just the total weight of the subtree rooted at node i. For other i and j values, $C[i][j]$ can be determined if we know for each descendant i' of i, values of $C[i'][j']$ for all j'. Every scheme to partion the subtree rooted at i into j parts can be considered as trying to detach a set of descendant of i, $i'_1, i'_2, \ldots, i'_l$, where no one is a decendant of the other, from the subtree rooted at i, and then partition each subtree rooted at $i'_x, 1 \leq x \leq l$ into j'_x parts, with each $j'_x \geq 1$ and $\sum_x j'_x = j - 1$. For each such attempt, the minimum-maximum weight of the partitions is given by:

$$\max(C[i'_1][j'_1], C[i'_2][j'_2], \ldots, C[i'_l][j'_l], (\text{Total weight of nodes that is not in any of the detached subtrees})$$

The value of $C[i][j]$ is just the minimum value of this minimum-maximum sum among all possible detaching schemes.

To fill in the $C[i][j]$ entries, each time we choose an i such that for every descendant i' of i, $C[i'][j']$ has been determined for all $1 \leq j' \leq k$. Then $C[i][j]$ can be determined for all $1 \leq j \leq k$. It can be easily seen that finaly $C[r][k]$ can be determined, where r is the root of the input tree. The value is the minimum-maximum weight we are looking for, and the partition can be determined by backtracing.

Problem 5 Solution

Consider $C[0\ldots m][0\ldots n]$, where $C[i][j]$ is the minimum cost to convert $A[0\ldots i]$ to $B[0\ldots j]$. We have the relation:

$$C[i][j] = \begin{cases} 4j & i = 0 \\ 3i & j = 0 \\ C[i-1][j-1] & ij \neq 0, A[i] = B[j] \\ \min(C[i-1][j]+3, C[i][j-1]+4, C[i-1][j-1]+5) & \text{Otherwise} \end{cases}$$

From this the algorithm is obvious.