TIN093 A3 Xinyan Liu

Problem 5

Definition:

Let

$$\mathrm{OPT}(n)$$

denote the maximum accumulated payment from day 1 to day n.

$$H_i = \begin{cases} 1, & \text{if work is performed on day } i, \\ 0, & \text{no work performed on day } i \end{cases}$$

Initial Conditions:

$$\mathrm{OPT}(0) = 0, \mathrm{OPT}(1) = P_1.$$

Formula and Prove

For $n \geq 2$, based on the original description, the recurrence is given by considering two cases:

$$\mathrm{OPT}(n) = \begin{cases} \mathrm{OPT}(n-1) + P_n, & \text{if no work is performed on day } n-1 \text{ (i.e., } H_{n-1} = 0\text{)}, \\ \max\left\{\mathrm{OPT}(n-2) + P_n,\ \mathrm{OPT}(n-1)\right\}, & \text{if work is performed on day } n-1 \text{ (i.e., } H_{n-1} = 1\text{)}. \end{cases}$$

The recurrence formula provided above ensures that all scenarios are incorporated. In the case where no work is performed on the previous day, if $P_i > 0$, working on the current day is necessary; this is the only possibility in that scenario. Conversely, if work has been done on previous days, there are two choices:

1. Choose not to work on the current day, thereby retaining the previously computed optimal value.
2. Reevaluate the decisions from the previous days to determine if working on the current day (and possibly altering past decisions) leads to a more favorable outcome.

Since if no work is performed on day $n-1$, then

$$\mathrm{OPT}(n-1) = \mathrm{OPT}(n-2),$$

the recurrence relation for $\mathrm{OPT}(n)$ can be written as:

$$\mathrm{OPT}(n) = \max\Big\{\ \mathrm{OPT}(n-2) + P_n,\ \mathrm{OPT}(n-1)\Big\}.$$

Time Complexity Analysis:

By storing the value of $\mathrm{OPT}(i)$ at each step (in the array or arraylist, etc), the recurrence for $\mathrm{OPT}(n)$ relies solely on the most recent two computed values (i.e., $\mathrm{OPT}(n-1)$ and $\mathrm{OPT}(n-2)$). Therefore, the time complexity for computing $\mathrm{OPT}(n)$ in each step is $O(1)$, and the overall time complexity for calculating all values from 1 to n is

$$O(n).$$

Problem 6

Let two strings A and B be defined as:

$$A = a_1, a_2, a_3, \ldots, a_m, \quad \text{where } a_j \text{ is the } j\text{-th element of string } A,\ j = 1, 2, \ldots, m,$$

$$B = b_1, b_2, b_3, \ldots, b_n, \quad \text{where } b_j \text{ is the } j\text{-th element of string } B,\ i = 1, 2, \ldots, n.$$

We define $OPT(i, j)$ as the minimum cost of aligning the prefix of the first i characters of string A with the prefix of the first j characters of string B. Given the recurrence relation for the dynamic programming solution:

$$OPT(i, j) = \min\{OPT(i-1, j) + 1,\ OPT(i, j-1) + 1\ OPT(i-1, j-1) + \Delta(a_i, b_j)\},$$

where $\Delta(i, j)$ is defined as:

$$\Delta(i, j) = \begin{cases} 0, & \text{if } a_i = b_j, \\ 1, & \text{if } a_i \neq b_j. \end{cases}$$

Optimization Using a Bandwidth of k

We observe that when the edit distance is 0, the path in the dynamic programming table strictly follows the diagonal direction. Once it reaches the length of one side, the path will shift to entirely horizontal or vertical movements. In this case, for each row, only one cell's OPT value needs to be computed.

However, when the edit distance is 1, the path may deviate one step away from the diagonal direction, either upward or downward perpendicular to the diagonal. Similarly, when the edit distance is k, the path can deviate up to k steps vertically above or below the diagonal. Consequently, for each row or diagonal direction, at most $2k + 1$ cells need to be computed.

As k increases, $2k + 1$ eventually exceeds the number of elements on each diagonal. At this point, all the cells in the table $(m \times n)$ will need to be computed, reverting to the original situation.

Since the edit distance k is small compared to the string lengths, most of the optimal alignment paths will lie within a diagonal band of width k. The key idea is to process only the cells $\{(i, j) : |i - j| \leq k\}$ in the dynamic programming table. This reduces the number of states to compute from $O(nm)$ to $O(kn)$.

1. Initialize the DP table such that: - $OPT(i, j) = \infty$ for $|i - j| > k$. - $OPT(0, 0) = 0$ (base case).
2. Fill the DP table for all cells (i, j) satisfying $|i - j| \leq k$ using the recurrence relation.
3. The final solution is stored in $OPT(n, m)$.

This approach ensures that we only compute relevant cells and hence achieve a time complexity of $O(kn)$.

Proof of Correctness

The correctness follows from the observation that any optimal alignment with at most k mismatches will lie within the diagonal band of width k. By restricting our computation to this band, we do not exclude any valid solutions.

Time Complexity Analysis

- There are at most $2k+1$ cells to compute for each of the n rows (with bandwidth 2k+1 in diagonal direction), leading to $O(2kn+n) = O(kn)$ cells in total.
- Each cell computation involves a constant amount of work if we use back tracing or record the OPT(i,j) value in the array, resulting in an overall time complexity of $O(kn)$.