

CS513 Design and Analysis of Algorithms

Fall 2008 - HW3 solution

Instructor: S. Muthukrishnan
TA: André Madeira

Problem 1: We show that the following greedy strategy is optimal. Given a left endpoint l_i find the rightmost endpoint l_{i+1} such that $\sum_{j=l_i+1}^{l_{i+1}} A[j] \leq W$. The algorithm consists of computing each l_i for all $0 \leq i < n$ in increasing order. The running time is clearly linear.

Proof. We prove it by contradiction. Suppose our algorithm finds a partition $L = \{l_1, l_2, \dots, l_{k_L}\}$ but there is an optimal partition $P = \{p_1, p_2, \dots, p_{k_P}\}$ such that $k_P < k_L$. Then, there should be an $1 \leq i < k_P$ s.t. $p_i > l_i$, otherwise it can be easily seen that $k_P \geq k_L$. Let k be the smallest such i . Then,

$$\sum_{j=p_{k-1}+1}^{p_k} A[j] \geq \sum_{j=l_{k-1}+1}^{l_k+1} A[j] > W,$$

a contradiction by the definition of P .

Problem 2: Consider the DP solution. We define the recursive function $P(i, j)$, the minimum cost of partitioning the first i elements into j pieces, as follows, $\forall 1 \leq i \leq n$ and $\forall 1 \leq j \leq k$:

$$P(i, j) = \begin{cases} A[1] & \text{if } i = 1, j > 0 \\ \sum_{p=1}^i A[p] & \text{if } j = 1, i > 1 \\ \min_{\ell} \max \left(P(\ell, j-1), \sum_{p=\ell+1}^i A[p] \right) & \text{o/w} \end{cases}$$

It can be easily seen that the definition is correct and defines an optimal substructure. We argue that its running time is $O(kn^2)$.

Observe that to compute each entry $P(i, j)$, we might sum at most n elements. If we pre-compute all sums $A[1] + \dots + A[b]$, $\forall 1 \leq b \leq n$ then we can answer any query $A[a] + \dots + A[b]$ in $O(1)$ while using $O(n)$ space in total. Furthermore, recall that finding the minimum takes $O(n)$. Therefore, the table $P(i, j)$ with nk entries can be computed in time $O(kn^2)$ time. The space usage is clearly $O(nk)$ as the main table uses $O(nk)$ entries and the pre-computation uses $O(n)$.

Problem 3:

Let strings $S = s_1 \dots s_n$ and $T = t_1 \dots t_n$. To transform $s_1 \dots s_i$ into $t_1 \dots t_j$, we can:

- put t_j at the end: $x \rightarrow s_1 \dots s_n t_j$ and then transform $s_1 \dots s_i$ into $t_1 \dots t_{j-1}$
- delete s_i : $x \rightarrow s_1 \dots s_{i-1}$ and then transform $s_1 \dots s_{i-1}$ into $t_1 \dots t_j$
- change s_i into t_j (if they are different): $x \rightarrow s_1 \dots s_{i-1} t_j$ and then transform $s_1 \dots s_{i-1}$ into $t_1 \dots t_{j-1}$
- do nothing if $s_i = t_j$ and then transform $s_1 \dots s_{i-1}$ into $t_1 \dots t_{j-1}$.

This suggests a recursive scheme where the sub-problems are of the form “how many operations do we need to transform $s_1 \dots s_i$ into $t_1 \dots t_j$ ”. The DP solution is then to define a $(n+1) \times (n+1)$ matrix M and fill it so that for every $0 \leq i, j \leq n$, $M[i, j]$ is the minimum number of operations to transform $s_1 \dots s_i$ into $t_1 \dots t_j$. The content of our matrix M can be formalized recursively as follows:

- $M[0, j] = j$ because the only way to transform the empty string into $t_1 \dots t_j$ is to add the j characters t_1, \dots, t_j .

- $M[i, 0] = i$ for similar reasons.
- For $i, j \geq 1$,

$$M[i, j] = \min \begin{cases} M[i-1, j] + 1, \\ M[i, j-1] + 1, \\ M[i-1, j-1] + \text{change}(s_i, t_j) \end{cases}$$

where $\text{change}(s_i, t_j) = 1$ if $s_i \neq t_j$ and $\text{change}(s_i, t_j) = 0$ otherwise.

Clearly, the number of changes necessary to transform S into T is in $M[n, n]$. The running time is then $O(n^2)$, as each entry takes $O(1)$ to compute, and the space usage is $O(n^2)$. However, note that each entry $M[\cdot, j]$ depends only on rows j and $j-1$. Therefore, we only need to keep two rows (overwriting values in the 2-row matrix as needed) to compute $M[n, n]$.

Problem 4:

The naïve algorithm simply try all permutation, but takes exponential time. A better solution is to just sort the items by increasing order of w_i/f_i . We prove it by contradiction. Suppose that $A = \{(w_1, f_1), \dots, (w_n, f_n)\}$ is a list of minimum cost. Assume there exists a k such that $w_k/f_k > w_{k+1}/f_{k+1}$. Let solution $B = \{(w_1, f_1), \dots, (w_{k-1}, f_{k-1}), (w_{k+1}, f_{k+1}), (w_k, f_{k+1}), (w_{k+2}, f_{k+2}), \dots, (w_n, f_n)\}$ (the one found by our algorithm). Since A is the minimum solution, we have that $\text{cost}(A) \leq \text{cost}(B)$. Observe that the costs of A and B differ only on the k^{th} and $(k+1)^{\text{th}}$ locations. Then, we have:

$$\begin{aligned} \text{cost}(A) &\leq \text{cost}(B) \\ f_k \left(\sum_{j \leq k} w_j \right) + f_{k+1} \left(\sum_{j \leq k+1} w_j \right) &\leq f_{k+1} \left(\sum_{j \leq k-1} (w_j) + w_{k+1} \right) + f_k \left(\sum_{j \leq k-1} (w_j) + w_{k+1} + w_k \right) \\ f_{k+1} w_k &\leq f_k w_{k+1} \\ \frac{w_k}{f_k} &\leq \frac{w_{k+1}}{f_{k+1}}, \end{aligned}$$

a contradiction. The running time and space complexity follows directly from your favorite sorting algorithm.