

CS 3158 Homework 4 Solutions

1. *Neapolitan and Naimipour*, page 132, number 20: Create the optimal binary search tree for the following items, where the probability of occurrence of each word is given in parentheses: CASE (.05), ELSE (.15), END(.05), IF(.35), OF(.05), THEN(.35).

	1	2	3	4	5	6
1	.05	.25	.35	.95	1.05	1.8
2	0	.15	.25	.8	.9	1.65
3	0	0	.05	.45	.55	1.4
4	0	0	0	.35	.45	1.2
5	0	0	0	0	.05	.45
6	0	0	0	0	0	.35

	1	2	3	4	5	6
1	1	2	2	4	4	4
2	0	2	2	4	4	4
3	0	0	3	4	4	4
4	0	0	0	4	4	4/6
5	0	0	0	0	5	6
6	0	0	0	0	0	6

The final tree: IF is the root, with children ELSE and THEN. ELSE has two children CASE and END. THEN's one child is OF.

2. *Neapolitan and Naimipour*, page 133, number 34: Let us consider two sequences of characters S_1 and S_2 . For example, we could have $S_1 = A\$CMA^*MN$ and $S_2 = AXMC4ANB$. Assuming that a subsequence can be constructed by deleting any number of characters from any positions, use the dynamic programming approach to create an algorithm that finds the longest common subsequence of S_1 and S_2 . This algorithm returns the maximum-length common subsequence of each sequence.

Let $X[1..n]$ and $Y[1..m]$ be two (nonempty) strings. A (possibly empty) string $S[1..k]$ is a common subsequence of these strings if and only if there exist indices $1 \leq x_1 < \dots < x_k \leq n$ and $1 \leq y_1 < \dots < y_k \leq m$ such that

$$S[l] = X[x_l] = Y[y_l]$$

for all $1 \leq l \leq k$. In addition, we say that $S[1 \dots k]$ is a *longest common subsequence of $X[1 \dots n]$ and $Y[1 \dots m]$* if its length is maximum among all common subsequences of these strings.

Theorem: If we let $c(i, j)$ be the length of a longest common subsequence of $X[1 \dots i]$ and $Y[1 \dots j]$ where $1 \leq i \leq n$ and $1 \leq j \leq m$, then

$$c(i, j) = \begin{cases} c(i-1, j-1) + 1, & \text{if } X[i] = Y[j] \\ \max\{c(i-1, j), c(i, j-1)\}, & \text{otherwise,} \end{cases}$$

for all $i, j > 0$. (Here $c(i, j)$ is understood to be 0 if either $i < 1$ or $j < 1$.)

Proof: We first prove two useful lemmas.

Lemma: If $1 \leq i \leq i' \leq n$ and $1 \leq j \leq j' \leq m$, and $S[1 \dots k]$ is a common subsequence of $X[1 \dots i]$ and $Y[1 \dots j]$, then $S[1 \dots k]$ is a common subsequence of $X[1 \dots i']$ and $Y[1 \dots j']$.

Proof: Since $S[1 \dots k]$ is a common subsequence of $X[1 \dots i]$ and $Y[1 \dots j]$, it follows that there exist indices $1 \leq x_1 < \dots < x_k \leq i$ and $1 \leq y_1 < \dots < y_k \leq j$ such that $S[l] = X[x_l] = Y[y_l]$ for all $l, j > 0$. Then clearly $1 \leq x_i < \dots < x_k \leq i'$ and $1 \leq y_1 < \dots < y_k \leq j'$, and $S[l] = X[x_l] = Y[y_l]$ for all $l, j > 0$. $S[1 \dots k]$ is a common subsequence of $X[1 \dots i']$ and $Y[1 \dots j']$. Hence we have $c(i, j) \leq c(i', j')$. \square

Lemma: Let $X[1 \dots n]$ and $Y[1 \dots m]$ be two nonempty strings such that $X[n] = Y[m]$. If $S[1 \dots k]$ is a longest common subsequence of these strings then

1. $k > 0$
2. $S[k] = X[n]$
3. There exist indices $1 \leq x_1 < \dots < x_k = n$ and $1 \leq y_1 < \dots < y_k = m$ such that $S[l] = X[x_l]$ for all $1 \leq l \leq k$. (The important point here is that $x_k = n$ and $y_k = m$.)

Proof: Clearly the string $S[1] = X[n]$ is a common subsequence of $X[1 \dots n]$ and $Y[1 \dots m]$, and hence $k > 0$. Now let $1 \leq x_l \leq n$ and $1 \leq y_l \leq m$ be indices such that $S[l] = X[x_l] = Y[y_l]$ for all $1 \leq l \leq k$. If $S[k] \neq X[n]$, then $x_k < n$ and $y_k < m$ since $X[x_k] = Y[y_k] = S[k]$ and $X[n] = Y[m]$. Therefore the sequence $X'[1 \dots (k+1)]$ defined by

- $S'[1 \dots k] = S[1 \dots k]$
- $S'[k+1] = X[n]$

is a common subsequence of $X[1 \dots n]$ and $Y[1 \dots m]$ of length $k + 1$, but this is not possible since S is a longest common subsequence. Hence it must be the case that $S[k] = X[n]$. The above argument can be used to show that either $x_k = n$ or $y_k = m$ since otherwise we could obtain a longer common subsequence. If $x_k = n$ and $y_k = m$, we are done. Now without loss of generality let us assume that $x_k < n$. Since $X[x_k] = Y[y_k] = Y[m] = X[n]$ it follows that the indices x'_j defined by

- $x'_j = x_j$, for $1 \leq j < k$
- $x'_k = n$

and the indices y_l satisfy the condition in 3. \square

Let $S[1 \dots k]$ be a longest common subsequence of $X[1 \dots i]$ and $Y[1 \dots j]$, i.e., $k = c(i, j)$, and let $1 \leq x_1 < \dots < x_l \leq i$ and $1 \leq y_1 < \dots < y_k \leq j$ be such that

$$S[l] = X[x_l] = Y[y_l]$$

for all $1 \leq l \leq k$.

First assume that $X[i] \neq Y[j]$. If $k = 0$, then clearly $c(i, j) = c(i-1, j) = c(i, j-1) = 0$. Thus assume $k > 0$. Then it is clear that either $x_k < i$ or $y_k < j$. Let us consider the case $x_k < i$. It follows that $S[1 \dots k]$ is a common subsequence of $X[1 \dots (i-1)]$ and $Y[1 \dots j]$, since

$$S[l] = X[x_l] = Y[y_l]$$

for all $1 \leq l \leq k$, where $1 \leq x_1 < \dots < x_l \leq i-1$ and $1 \leq y_1 < \dots < y_k \leq j$. Hence $c(i, j) \leq c(i-1, j)$. Similarly we can show that if $y_k < j$ then $c(i, j) \leq c(i, j-1)$. Thus if $X[i] \neq Y[j]$ then

$$c(i, j) \leq \max\{c(i-1, j), c(i, j-1)\}.$$

On the other hand we know that $c(i, j) \geq c(i-1, j)$ and $c(i, j) \geq c(i, j-1)$, and hence

$$c(i, j) \geq \max\{c(i-1, j), c(i, j-1)\}.$$

We then conclude that if $X[i] \neq Y[j]$ then

$$c(i, j) = \max\{c(i-1, j), c(i, j-1)\}.$$

Now assume that $X[i] = Y[j]$. We know we can assume that $x_k = i$ and $y_k = j$. Then clearly $S[1 \dots (k-1)]$ is a longest common subsequence of $X[1 \dots (i-1)]$ and $Y[1 \dots (j-1)]$ since otherwise we could construct a common subsequence of $X[1 \dots i]$ and $Y[1 \dots j]$ of length greater than k . Thus $c(i, j) = c(i-1, j-1) + 1$. \square

Thus if we process the entries of c by increasing order of rows, where rows are processed by increasing order of columns, we can compute all entries in time $O(nm)$. In order to construct a longest common subsequence, we define another $n \times m$ array, d , as follows:

$$d(i, j) = \begin{cases} \swarrow, & \text{if } X[i] = Y[j] \\ \leftarrow, & \text{if } X[i] \neq Y[j] \text{ and } c(i, j-1) < c(i, j) \\ \uparrow, & \text{otherwise} \end{cases}$$

Clearly $d(i, j)$ be computed with constant overhead when $c(i, j)$ is computed, and hence we can compute both arrays in $O(nm)$ time.

Finally, we compute a longest common subsequence by following the arrows in d as follows. We start by looking at $d(n, m)$. If its value is \swarrow , we write down $X[n]$, and we repeat from $d(n-1, m-1)$. If $d(n, m)$ is \leftarrow or \uparrow , then we don't write down anything and repeat from $d(n-1, m)$ or $d(n, m-1)$ respectively. We stop this process as soon as we try to move outside the bounds d . At this point the string we have written down is the reverse of a longest common subsequence.

3. You are given a sequence of positive integers a_1, a_2, \dots, a_n and a positive integer B . Your goal is to determine if some subsequence of a_1, a_2, \dots, a_n sums to exactly B . (In other words, determine if there is a subset S of $\{1, 2, \dots, n\}$ such that $\sum_{i \in S} a_i = B$.)

Give a dynamic programming algorithm for this problem. The algorithm should run in time bounded by cnB , where c is a constant. Hint: Use $f(i, b) = 1$ if some subset of a_1, \dots, a_i sums to exactly b , and 0 otherwise. (For bonus credit, explain why the running time is pathetic).

Theorem: The function f as defined above satisfies the following condition.

$$f(i, b) = \begin{cases} 1, & \text{if } i = 1 \text{ and } a_1 = b \\ 0, & \text{if } i = 1 \text{ and } a_1 \neq b \\ 1, & \text{if } i > 1 \text{ and } f(i-1, b) = 1 \\ f(i-1, b - a_i), & \text{otherwise} \end{cases}$$

Proof: It is clear from the definition of f that $f(1, b)$ equals 1 if $b = a_1$ and 0 otherwise. Hence let us fix $i > 1$ and $1 \leq b \leq B$. There are two cases to consider. Either there exists a subset S of $\{a_1, \dots, a_{i-1}\}$ whose sum is b or not. If such a set S exists, then $f(i, b) = 1$ since S is also a subset of $\{a_1, \dots, a_i\}$. Now suppose that there is no such set S . If there exists a subset T of $\{a_1, \dots, a_i\}$ whose sum is b , then we have that $a_i \in T$ and

$$\begin{aligned} \sum_{t \in T} t &= \left(\sum_{t \in T - \{i\}} t \right) + a_i \\ \sum_{t \in T - \{i\}} t &= \left(\sum_{t \in T} t \right) - a_i \\ \sum_{t \in T - \{i\}} t &= b - a_i. \end{aligned}$$

Hence it must be the case that $f(i, b - a_i) = 1$ since $T - \{i\} \subseteq \{a_1, \dots, a_{i-1}\}$. On the other hand if there exists a subset P of $\{a_1, \dots, a_{i-1}\}$ whose sum is $b - a_i$ then $P \cup \{a_i\}$ is a subset of $\{a_1, \dots, a_i\}$ whose sum is b and hence $f(i, b) = 1$. \square

We can compute all entries of the array f by processing columns in increasing order. Since there are n columns and B rows, and to compute the value of one entry takes constant time, the entire process takes time $O(nB)$. This run time is not good because representing B takes $\log B$ bits, and thus this algorithm is exponential in the representation size of B .

4. Suppose you are going camping in Utah. Before your trip, you list a sequence of items you wish to bring. You creatively call the items $1, 2, 3, \dots, n$. Now each items has some positive integral profit p_i in dollars, and some positive weight w_i , in kg. You can afford to carry B kilograms. What's the most profitable subset of the n items to bring, subject to the weight restriction?

In order to solve this problem, we will maintain two $n \times B$ arrays, c and d . We will let $c(i, b)$ denote the greatest profit achievable by a subset of the first i items, subject to a weight restriction of b .

Theorem: The function c satisfies the following condition.

$$c(i, b) = \begin{cases} p_1, & \text{if } w_1 \leq b \text{ and } i = 1 \\ 0, & \text{if } w_1 > b \text{ and } i = 1 \\ c(i-1, b), & \text{if } w_i > b \text{ and } i > 1 \\ \max\{c(i-1, b - w_i) + p_i, c(i-1, b)\}, & \text{if } w_i \leq b \text{ and } i > 1 \end{cases}$$

Proof: The above statement is clear for the cases in which $i = 1$. Hence let us fix $i > 1$ and $1 \leq b \leq B$. Now there are two cases. Either $w_i \leq b$ or $w_i > b$.

Let us assume that $w_i \leq b$, and let $S \subseteq \{1, \dots, i\}$ achieve the maximum profit. If $i \in S$ then $S - \{i\}$ must achieve the maximum profit with weight constraint $b - w_i$ among all subsets of $\{1, \dots, i - 1\}$, and hence

$$c(i, b) = p_i + c(i - 1, b - w_i).$$

Note that $c(i, b) \geq c(i - 1, b)$ since every subset of $\{1, \dots, i - 1\}$ is a subset of $\{1, \dots, i\}$. Hence

$$c(i, b) = p_i + c(i - 1, b - w_i) = \max\{p_i + c(i - 1, b - w_i), c(i - 1, b)\}.$$

If $i \notin S$, then clearly it must be the case that

$$c(i, b) \geq c(i - 1, b - w_i) + p_i,$$

since we can always obtain a subset of $\{1, \dots, i\}$ observing weight constraint of b whose profit is the quantity on the right (because $w_i \leq b$). Also since $S \subseteq \{1, \dots, i - 1\}$, it must also achieve the maximum profit with weight constraint b among all subsets of $\{1, \dots, i - 1\}$. Hence

$$c(i, b) = c(i - 1, b),$$

and furthermore

$$c(i, b) = c(i - 1, b) = \max\{c(i - 1, b), c(i - 1, b - w_i) + p_i\}.$$

So, whether or not $i \in S$, we conclude that

$$c(i, b) = \max\{c(i - 1, b), c(i - 1, b - w_i) + p_i\}.$$

Finally assume that $w_i > b$, and let $S \subseteq \{1, \dots, i\}$ achieve the maximum profit with weight constraint b . Clearly $i \notin S$ and hence $S \subseteq \{1, \dots, i - 1\}$. Again S must also achieve the maximum profit with weight constraint b among all subsets of $\{1, \dots, i - 1\}$. Hence

$$c(i, b) = c(i - 1, b). \square$$

We make use of an auxiliary array d to compute a maximum profit subset from the information in c . We let $d(1, b) = \mathbf{yes}$ if $w_1 \leq b$ and \mathbf{no} otherwise. For $i > 1$ we let

$$d(i, b) = \begin{cases} \mathbf{no}, & \text{if } w_i > b \\ \mathbf{no}, & \text{if } w_i \leq b \text{ and } c(i - 1, b) \geq c(i - 1, b - w_i) + p_i \\ \mathbf{yes} & \text{otherwise} \end{cases}$$

In other words, $d(i, b)$ tells us if we should include the i^{th} object in order to obtain a maximum profit subset of the first i items given a weight constraint of b .

It is clear that we can compute both arrays in time $O(nB)$ by processing rows in increasing order. We can obtain a maximum profit subset of $\{1, \dots, n\}$ given a weight constraint of B as follows. If $d(n, B) = \mathbf{yes}$ then we add the n^{th} item to the set, and we recurse with $d(n-1, B - w_n)$. If $d(n, B) = \mathbf{no}$ then we don't add the n^{th} item, and we recurse with $d(n-1, B)$. This process will take time $O(n)$ since we decrease the row number by one in every iteration. The correctness proof of this procedure is straightforward. Hence the whole process takes time $O(nB)$. Again this running time is not good because it is exponential in the representation size of B .

5. Your job is to typeset a paragraph of n words whose i th word has exactly l_i characters, on lines with 80 characters. You place exactly one blank between adjacent words. The penalty for b blank characters at the end of a line is b^4 . Every line, including the last line, is penalized.

Show how to compute the minimum cost to typeset the paragraph in time $O(n)$ using dynamic programming. Prove that your answer is correct.

Let $c(i)$ denote the minimum cost to typeset the first i words, and let $b(i)$ be the minimum positive integer such that all the words from $w_{b(i)}$ to w_i can fit in one line, i.e., $\sum_{k:b(i) \leq k \leq i} (l_k + 1) \leq 81$ and $\sum_{k:b(i)-1 \leq k \leq i} (l_k + 1) > 81$.

Theorem:

$$c(i) = \min_{j:b(i) \leq j \leq i} \left\{ c(j-1) + \left(80 - \left(-1 + \sum_{k:j \leq k \leq i} (l_k + 1) \right) \right)^4 \right\}$$

where $c(0) = 0$.

Proof: Assume that we have an optimal formatting of the first i words, and let us assume that the paragraph takes $m \geq 1$ lines. Also let j be the index of the first word in the last line. Clearly the first $m-1$ lines must be an optimal formatting of the first $j-1$ words, as a paragraph by itself. If this were not true then we could obtain a better formatting for the first i words by using a better formatting for the first $j-1$ words and then adding the last line containing the remaining words from j to i . On the other hand,

j has to be such that

$$\sum_{k:j \leq k \leq i} (l_k + 1) \leq 81.$$

in order for it to be the first word on the last line.

Hence it follows that

$$c(i) = \min_{b(i) \leq j \leq i} \left\{ c(j-1) + \left(80 - \left(\sum_{k:j \leq k \leq i} (l_k + 1) - 1 \right) \right)^4 \right\}. \square$$

Now in order to compute c , we proceed in increasing order of i . For a given i , we try all possible choices of the first word j on the last line. There are clearly at most 80 such choices. We do so by starting with $j = i$ and then decreasing j until we reach a word that cannot fit on the same line as the i^{th} word. For every valid value of j we compute

$$c(j-1) + \left(80 - \left(\sum_{k:j \leq k \leq i} (l_k + 1) - 1 \right) \right)^4$$

and we keep the minimum.

In order to actually obtain a formatting of the n words with optimal cost $c(n)$, for each $i = 1, 2, 3, \dots, n$ we define $d(i)$ to be the value of $j \in \{b(i), b(i) + 1, \dots, i\}$ that minimizes

$$c(j-1) + \left(80 - \left(\sum_{k:j \leq k \leq i} (l_k + 1) - 1 \right) \right)^4.$$

In other words $d(i)$ is the first word in the last line of an optimal formatting of the first i words. It is clear that $d(i)$ can be obtained with constant time overhead during the computation of $c(i)$. Hence we can compute both arrays in $O(n)$ time.

We can obtain an optimal formatting of the first n words as follows. From $d(n)$ we obtain the first word in the last line. Then we look at $d(d(n) - 1)$ and we obtain the first word in the penultimate line. We repeat this process until we reach the first word w_1 and hence the first line.