Exercise Sheet 9: Answers
COMS10017 Algorithms 2022/2023

1 Pole Cutting

We consider the Pole Cutting problem as discussed in the lectures.

1. Consider the following price function:

length i	1	2	3	4	5	6	7	8	9
price $p(i)$	1	4	7	9	9	11	12	16	18

Let $r(i)$ be the maximum revenue achievable for a pole of length i. What are the values $r(i)$, for $1 \leq i \leq 9$?

Solution.

length i	1	2	3	4	5	6	7	8	9
price $p(i)$	1	4	7	9	9	11	12	16	18
revenue $r(i)$	1	4	7	9	11	14	16	18	21

✓

For each $1 \leq i \leq 9$, state how a pole of length i needs to be cut in order to achieve the maximum revenue $r(i)$.

Solution.

$$\begin{aligned}
1 &= 1 \\
2 &= 2 \\
3 &= 3 \\
4 &= 4 \\
5 &= 2 + 3 \\
6 &= 3 + 3 \\
7 &= 3 + 4 \\
8 &= 2 + 3 + 3 \\
9 &= 3 + 3 + 3 \ .
\end{aligned}$$

✓

2. Give a price function $p : \mathbb{N} \rightarrow \mathbb{N}$ such that $r(i) = p(i)$ and, for each i, the only way to achieve the maximum revenue is by selling the pole without breaking it.

Solution. Many functions work here, for example, we can use the function $p(i) = 3^i$.

To see that this works, suppose that we cut a pole of length i into smaller pieces $i = i_1 + i_2 + \cdots + i_k$. Then, we have to show that $p(i_1) + p(i_2) + \cdots + p(i_k) < p(i)$ holds.

To this end, first, observe that, for any integers $x, y \geq 3$, $x + y < x \cdot y$ holds. To see this, observe that $x + y < x \cdot y$ is equivalent to $x > \frac{y}{y-1}$, and the inequality $\frac{y}{y-1} \leq \frac{3}{2}$ holds for every $y \geq 3$. The statement is thus true for every $x, y \geq 3$.

Next, from the previous statement, we obtain that $3^x + 3^y < 3^{x+y}$, for every $x, y \geq 1$. Then:

$$\begin{aligned}\sum_{j=1}^{k} p(i_j) &= \sum_{j=1}^{k} 3^{i_j} \\ &< 3^{i_1+i_2} + \sum_{j=3}^{k} 3^{i_j} \\ &< 3^{i_1+i_2+i_3} + \sum_{j=4}^{k} 3^{i_j} \\ &\dots \\ &< 3^{\sum_{j=1}^{k} i_j} = 3^i \ .\end{aligned}$$

Another function, which gives a more straightforward mathematical argument, is the function $p(i) = 2i - 1$. Similar to the argument above, we need to show that

$$p(i_1) + p(i_2) + \cdots + p(i_k) < p(i)$$

holds, for any i and $k \geq 2$. Then,

$$\begin{aligned}p(i_1) + p(i_2) + \cdots + p(i_k) &= (2i_1 - 1) + (2i_2 - 1) + \cdots + 2(i_k - 1) \\ &= \left(2\sum_{j=1}^{k} i_j\right) - k \\ &= 2 \cdot i - k \\ &< 2i - 1 = p(i) \ ,\end{aligned}$$

where the inequality holds since $k \geq 2$.

✓

3. Give a price function $p : \mathbb{N} \to \mathbb{N}$ such that $r(i) = p(i)$ and, for each i, no matter how the pole is cut, it always yields the maximum revenue.

Solution. The price function $p(i) = i \cdot C$, for any constant $C \geq 0$ works here. ✓

4. Is there a price function $p : \{1, 2, \dots, 10\} \to \mathbb{N}$ such that, for every $i \geq 2$, the maximum revenue is achieved by cutting the pole exactly once and no other way of cutting the pole achieves the maximum revenue?

Solution. Such a price function does not exist.

For the sake of a contradiction, suppose that there is such a price function. Then, $p(2) < 2 \cdot p(1)$, since otherwise the maximum for $i = 2$ would be achieved by not cutting the pole at all. Then, consider a pole of length 3. We know that the maximum revenue must be achieved by the cut $3 = 2+1$, and, thus, $r(3) = p(2)+p(1)$. However, since $p(2) < 2 \cdot p(1)$, we have $r(3) = p(2) + p(1) < 3 \cdot p(1)$, which shows that cutting the pole of length 3 into three pieces of length 1 achieves a higher revenue, a contradiction. A price function as in the statement of the question therefore does not exist. ✓

2 Dynamic Programming: Breaking a String

A certain string-processing language allows a programmer to break a string into two pieces. Because this operation copies the string, it costs n time units to break a string of n characters into two pieces. Suppose a programmer wants to break a string into many pieces. The order in which the breaks occur can affect the total amount of time used. For example, suppose that the programmer wants to break a 20-character string after characters $2, 8$, and 10 (numbering the characters in ascending order from the left-hand end, starting from 1). If she programs the breaks to occur in left-to-right order, then the first break costs 20 time units, the second break costs 18 time units, and the third break costs 12 time units, totaling 50 time units. If she programs the breaks to occur in right-to-left order, however, then the first break costs 20 time units, the second break costs 10 time units, and the third break costs 8 time units, totaling 38 time units. In yet another order, she could break first at 8 (costing 20), then break the left piece at 2 (costing 8), and finally the right piece at 10 (costing 12), for a total cost of 40.

Design an algorithm that, given the number of characters after which to break, determines a least-cost way to sequence those breaks. More formally, given a string S with n characters and an array $L[1..m]$ containing the break points, compute the lowest cost for a sequence of breaks, along with a sequence of breaks that achieves this cost. What is the runtime of your algorithm?

Solution. Let us assume that our input is a string S of length n (indexed from 0 to $n-1$) and an array L of length m (indexed from 0 to $m-1$) containing the break points. Furthermore, we assume that the array L of break positions is sorted in increasing order. Further, if we say that we break string S at position p, then we obtain the two substrings $S[0 \dots p]$, and $S[p+1 \dots n-1]$.

Optimal Substructure: Let us first identify the optimal substructure of the problem. Consider an optimal sequence of break position $j_1, j_2, j_3, \dots, j_m$, i.e., we first break S at position $L[j_1]$, then at $L[j_2]$, and so on. Consider the first break point $L[j_1]$: It breaks the string S into two substrings S_1 and S_2. Then, the optimal substructure is as follows: The subsequence of $L[j_2], L[j_3], \dots L[j_m]$ that are smaller than $L[j_1]$ breaks S_1 optimally, and the subsequence of $L[j_2], L[j_3], \dots L[j_m]$ of values larger than $L[j_1]$ breaks S_2 optimally.

For technical reasons let us first increase the array L by two and add the two break points -1 (at the beginning of L) and $n-1$ (at the end of L) to the array. Then, at the end of the algorithm we will have obtained the substrings $S_{0,1}$, $S_{1,2}$, $S_{2,3}$, $\dots$, $S_{m,m+1}$, using the definition (for any $i < j$)

$$S_{i,j} := S[L[i] + 1, \dots, L[j]] \ .$$

From now on, the array L is hence of length $m + 2$.

Our dynamic programming table T is indexed by two variables i, j and will store the optimal cost for breaking $S_{i,j}$, for every $0 \leq i < j \leq m + 1$. Then:

$$\begin{aligned} T[i,i+1] &= 0 \text{ for every } 0 \le i \le m \\ T[i,j] &= \min_{k\in\{i+1,\dots,j-1\}} \{ \underbrace{T[i,k]}_{\text{cost for left substr.}} + \underbrace{T[k,j]}_{\text{cost for right substr.}} + \underbrace{(L[j]-L[i])}_{\text{cost for breaking } S_{i,j}} \} . \end{aligned}$$

The minimum cost for breaking the entire string is then stored in $T[0,m+1]$. Since the table is of size $m+2 \times m+2$, and the minimum in the previous definition is taken over at most m elements, our resulting dynamic programming algorithm has a runtime of $O(m^3)$.

Example: Let's illustrate this approach with the example given in the problem statement. We have $n=20$, and $L=[-1,1,7,9,19]$ (we added the values 0 to the beginning and $n-1$ at the end, we also subtracted 1 from each internal breakpoint since we index arrays starting at 1 in the problem statement). After breaking up the string S, we are left with the substrings

$$\begin{aligned} S_{0,1} &= S[0\dots 1] \\ S_{1,2} &= S[2\dots 7] \\ S_{2,3} &= S[8\dots 9] \\ S_{3,4} &= S[10\dots 19]. \end{aligned}$$

We first enter the values $T[i,i+1]=0$ into our table. Observe that there is no cost since the strings $S_{i,i+1}$ do not need to be broken any further:

	$i=0$	$i=1$	$i=2$	$i=3$
$j=1$	0			
$j=2$		0		
$j=3$			0	
$j=4$				0

Next, we can fill the first off-diagonal. Observe that there is no "choice" for breaking $S_{i,i+2}$, since the only breakpoint within this string is $L[i+1]$. We thus obtain:

	$i=0$	$i=1$	$i=2$	$i=3$
$j=1$	0			
$j=2$	8	0		
$j=3$		8	0	
$j=4$			12	0

Regarding the next off-diagonal, there are two choices to break $S_{i,i+3}$: Either we break at point $L[i+1]$ or $L[i+2]$ first. We obtain:

$$\begin{aligned} T[0,3] &= \min_{k\in\{1,2\}} \{T[0,k]+T[k,3]+10\} = \min\{0+8+10, 8+0+10\} = 18 \\ T[1,4] &= \min_{k\in\{2,3\}} \{T[1,k]+T[k,4]+18\} = \min\{0+12+18, 8+0+18\} = 26 \ . \end{aligned}$$

We obtain the following table:

	$i=0$	$i=1$	$i=2$	$i=3$
$j=1$	0			
$j=2$	8	0		
$j=3$	18	8	0	
$j=4$		26	12	0

Last we compute the minimum cost for breaking the entire string $T[0,4]$:

$$T[0,4] \quad = \quad \min_{k\in\{1,2,3\}}\{T[0,k]+T[k,4]+20\} = \min\{0+26+20, 8+12+20, 18+0+20\} = 38\ .$$

This completes our table: We obtain the following table:

	$i=0$	$i=1$	$i=2$	$i=3$
$j=1$	0			
$j=2$	8	0		
$j=3$	18	8	0	
$j=4$	38	26	12	0

The right-to-left strategy suggested in the problem description is thus optimal.

✓

3 Difficult and Optional: Image Compression by Seam Carving

We are given a color picture of an $m\times n$ array $A[1..m, 1..n]$ of pixels, where each pixel specifies a triple of red, green, and blue (RGB) intensities. Suppose that we wish to compress this picture slightly. Specifically, we wish to remove one pixel from each of the m rows, so that the whole picture becomes one pixel narrower. To avoid disturbing visual effects, however, we require that the pixels removed in two adjacent rows be in the same or adjacent columns; the pixels removed form a "seam" from the top row to the bottom row where successive pixels in the seam are adjacent vertically or diagonally.

1. Show that the number of such possible seams grows at least exponentially in m, assuming that $n \geq 2$.

2. Suppose now that along with each pixel $A[i,j]$, we have calculated a real-valued disruption measure $d[i,j]$, indicating how disruptive it would be to remove pixel $A[i,j]$. Intuitively, the lower a pixel's disruption measure, the more similar the pixel is to its neighbors. Suppose further that we define the disruption of a seam to be the sum of the disruption measures of its pixels.

   Give an algorithm to find a seam with the lowest disruption measure. How efficient is your algorithm?

Solution. See solution to problem 15-8 on page 14 here:

`https://sites.math.rutgers.edu/~ajl213/CLRS/Ch15.pdf`

✓