Problem Set 8 for CS 170

Formatting

Please use the following format for the top of the solution you turn in, with one line per item below (in the order shown below):

<your username on cory.eecs>
<your full name>
`CS170, Spring 2003`
`Homework #8`
`Section` <your section number>
`Partners:` <your list of partners>

(Remember to write your section number, not the name of your TA or the time of your section.) This will make it easier for us to sort and process your homeworks. Thank you!

Note

When asked for an algorithm you must give (1) a brief informal description of the algorithm, (2) a precise description using pseudo-code, (3) an informal argument for termination and correctness of the algorithm, and (4) an analysis of the running time of the algorithm. Be clear about what the input to the algorithm is, how you measure the size of the input, and what constitutes a "step" in your running-time analysis.

Problem 0. [Any questions?] (5 points)

What's the one thing you'd most like to see explained better in lecture or discussion sections? A one-line answer would be appreciated.

(Sometimes we botch the description of some concept, leaving people confused. Sometimes we omit things people would like to hear about. Sometimes the book is very confusing on some point. Here's your chance to tell us what those things were.)

Problem 1. [TeX's Line Breaking Algorithm] (30 points)

A paragraph is composed of n words, $w_1, \ldots, w_n$, where the size of word j is $s(w_j)$. We'd like to find the optimal place to introduce line breaks between the words to produce an aesthetically pleasing paragraph. The page has width $\mathcal{W}$. We'll define the cost of line k to be $c(\ell_k)$ as follows:

$$c(\ell_k) \equiv \left(\mathcal{W} - \sum_{w_i \in \ell_k} s(w_i) \right)^2$$

and the cost of a paragraph to be the sum of the costs of all lines in the paragraph, $\sum_\ell c(\ell)$.

Give a polynomial-time dynamic programming algorithm to find an optimal division of words into lines that minimizes the cost of the paragraph. Analyze the running time of your algorithm in terms of n. You do not need to find the most efficient algorithm, as long as your solution's running time is polynomial in n.

For fun. As an aside, the TEX document preparation system uses a dynamic programming algorithm to break paragraphs into lines, though their cost function is slightly more complicated than the sum of squares cost function presented here. It also uses a similar algorithm to break lines into pages.

Problem 2 [Gasoline Refilling] (35 points)

Suppose you want to drive from San Francisco to New York City on I-80. Your car holds C gallons of gas and gets m miles to the gallon. You are handed a list of the n gas stations that are on I-80 and the price that they sell gas. Let d_i be the distance of the i^{th} gas station from SF, and c_i be the cost of gasoline at the i^{th} gas station. Furthermore, you can assume that for any two stations i and j, the distance $d_i - d_j$ between these two stations is a multiple of m. You start out with an empty tank at station 1. Your final destination is gas station n. You need to end at station n with at least 0 gallons of gas.

Find a polynomial-time dynamic programming algorithm to output the minimum gas bill to cross the country. Analyze the running time of your algorithm in terms of n and C. You do not need to find the most efficient algorithm, as long as your solution's running time is polynomial in n and C.

Remember that your car cannot run if your tank ever holds less than 0 gallons of gas. Also, if you decide to get gasoline at a particular station, you needn't fill up the tank; for example, you might decide to purchase only 7 gallons of gas at one station.

Problem 3 [Tiling] (30 points)

The following puzzle involves assembling tiles to form a given pattern p. The tiles and the pattern are composed of squares that are arranged next to each other; each square has some color. There are four possible colors: $\mathcal{C} = \{\text{Red}, \text{Blue}, \text{Green}, \text{Yellow}\}$. A pattern p of size n is a sequence of colors: $p = (p_1, \ldots, p_n)$, with each $p_i \in \mathcal{C}$. A tile is also a sequence of colors: $t = (t_1, \ldots, t_k)$, with each $t_i \in \mathcal{C}$. Let $\mathcal{T}$ be a set of tiles. A tiling of p is a sequence of tiles $(t^1, \ldots, t^\ell)$ that can be laid down in order to form the pattern p, where each $t^i \in \mathcal{T}$. In other words, we require that $(t^1_1, \ldots, t^1_{k_1}, t^2_1, \ldots, t^2_{k_2}, \cdots, t^\ell_1, \ldots, t^\ell_{k_\ell}) = (p_1, \ldots, p_n)$.

Create a dynamic programming algorithm that, on input p and $\mathcal{T}$, outputs a tiling of p that uses the minimum number of tiles. You have an inexhaustible supply of each kind of tile, so you may use a given tile more than once. In other words, we want to minimize ℓ, the total number of tiles used (with repeated tiles counted once for each time they are used). Analyze the running time of your algorithm in terms of n and $|\mathcal{T}|$. You do not need to find the most efficient algorithm, as long as your solution's running time is polynomial in n and $|\mathcal{T}|$. You may assume that there exists at least one tiling of p.

For instance, we might have the desired pattern

$$p = (R, R, G, Y, B, R, G, Y, B)$$

and the tiles

$$\begin{array}{lll} t_1 = (R) & t_2 = (G) & t_3 = (B) \\ t_4 = (Y) & t_5 = (R, R) & t_6 = (B, R) \\ t_7 = (Y, B, R) & t_8 = (B, R, G) & t_9 = (G, Y, B) \end{array}$$

One particular tiling of p would be

$$(t_1, t_1, t_2, t_4, t_3, t_1, t_2, t_4, t_3).$$

This tiling uses 9 tiles. A better tiling would be

$$(t_5, t_9, t_1, t_9),$$

since this uses only 4 tiles.