

# Data Structures and Algorithms

## Exercise 9

Markus Innerebner, Romans Kasperovics  
dsa(a)inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, April 23th, 2009

# Assignment 09

## Word Wrap + Justify

A paragraph consists of words separated by spaces. A word wrapping procedure breaks the given paragraph into several lines with a length not exceeding the given number  $M$ . Each line contains only whole words separated by spaces, no words can span over several lines. To make a wrapped paragraph justified to the margins we insert extra spaces between the words so that the first word on each line starts at the beginning of the line and the last word ends at the end of the line. The wrapping is optimal if we need minimal number of spaces to make the text justified. For example, the following paragraph is wrapped optimally to  $M = 30$ :

```
Alice was beginning to get
very tired of sitting by
her sister on the bank, and
of having nothing to do.
```

Write a C program that wraps the given paragraph to the given length and makes all lines justified. Your program should use dynamic programming “bottom-up” approach to find the words after which to break a line so that the wrapping is optimal. Insert extra spaces proportionally between all words in every line to justify the text. Use the example from the previous slide to test your program.

Modify the program you wrote for Task 1 in such a way that if the number of extra spaces needed to be inserted into the last line exceeds 10% of  $M$ , the last line is not justified.

# Assignment 09

## Task 3: Matrix Multiplication, Memoization

For the following multiplication of matrices calculate the cost of all possible orders of multiplication. Program a C function `MemoMM(int i, int j)` that returns the minimum cost of multiplication of matrices  $A_i \times \dots \times A_j$ . Use memoization technique to avoid the calculation of same values multiple times. Test this function on the matrices shown below. You can use global variables for auxiliary memory structures or can use an alternative signature for `MemoMM` to avoid the global variables.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} & b_{16} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} & b_{26} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & b_{36} \end{pmatrix} \times \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \\ c_{31} & c_{32} \\ c_{41} & c_{42} \\ c_{51} & c_{52} \\ c_{61} & c_{62} \end{pmatrix}$$

Write a C program that takes as an input the dimensions of  $n$  matrices being multiplied,  $A_1 \times A_2 \times \dots \times A_n$ , and puts the parentheses in a way that leads to the minimum number of multiplications. For example, for matrices  $A_1$  with dimensions  $4 \times 3$ ,  $A_2$  with dimensions  $3 \times 6$ , and  $A_3$  with dimensions  $6 \times 2$ , the program would print “(A1 x ( A2 x A3))”.

Assume  $w(i)$  returns the length of  $i$ -th word in the paragraph. The number of extra spaces to be inserted in a line starting with word  $i$  and ending with word  $j$  is described with a function  $e(i, j)$ :

$$e(i, j) = M - (j - i) - \sum_{k=i}^j w(k)$$

Use the following cost function for a line starting with word  $i$  and ending with word  $j$ :

$$c(i, j) = \begin{cases} (e(i, j))^2 & \text{if } e(i, j) \geq 0 \\ \infty & \text{if } e(i, j) < 0 \end{cases}$$

The cost of the optimal wrapping of a paragraph of  $j$  words can be formulated as the following recurrence:

$$f(j) = \begin{cases} c(1, j) & \text{if } c(1, j) < \infty \\ \min_{1 \leq k < j} \{f(k) + c(k+1, j)\} & \text{if } c(1, j) = \infty \end{cases}$$

In other words, we start “wrapping” from the last line and search for the optimal number of words to be left in the last line. We try all possibilities from 1 to  $j$  words. When we leave  $j - k$  words in the last line, we calculate the cost of wrapping the remaining words recursively and add it to the cost of the last line. We choose the number of words in the last line that gives the minimum cost.

```
#include <stdio.h>
#include <limits.h>
#define N 4 // number of matrices
#define INFTY INT_MAX

int D[N+1] = {4,3,6,2}; // dimensions of 3 matrices
int M[N][N]; // auxiliary memory structure for costs
int P[N][N]; // auxiliary memory structure for decisions
```

Use the pseudocode from the lecture notes!