Henzi Kou

CIS 315: Intermediate Algorithms

Christopher Wilson

21 May 2019

Assignment 5

1. (exercise 15.2-1, p 378 from CLRS) Find an optimal parenthesization of a matrix-chain product instance whose sequence of dimensions is ⟨5,10,3,12,5,50,6⟩. Show the final m and s tables.

*m*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 150 | 330 | 405 | 1655 | 2010 |
| 2 |  | 0 | 360 | 330 | 2430 | 1950 |
| 3 |  |  | 0 | 180 | 930 | 1770 |
| 4 |  |  |  | 0 | 3000 | 1860 |
| 5 |  |  |  |  | 0 | 1500 |
| 6 |  |  |  |  |  | 0 |

*s*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 2 | 3 | 4 | 5 | 6 |
| 1 | 1 | 2 | 2 | 4 | 2 |
| 2 |  | 2 | 2 | 2 | 2 |
| 3 |  |  | 3 | 4 | 4 |
| 4 |  |  |  | 4 | 4 |
| 5 |  |  |  |  | 5 |

Therefore we know that *s*[1, 6] = 2 ((*A*1 *A*2) \* (*A*3 *A*4 *A*5 *A*6)) and we can restructure this with *s*[3, 6] = 4 ((*A*1 *A*2) \* ((*A*3 *A*4) \* (*A*5 *A*6))).

1. … Given the data on robot arrivals X = (x1, x2, ..., xn), and given the recharging function f(−), the problem is to determine the maximum number of robots that can be destroyed by activating the EMP at certain points in time….

**(a)** Let *R*(*i*) be a function to represent the maximum number of robots that are killed from the starting point at 1 second to the end point at *i* seconds.

**(b)** We must begin by initializing a base case in which we can determine to start at specific point; in this case we will initialize it to 0 seconds. Then we must find most recent time *j* that the EMP was used. In order to find the maximum number of robots killed we will sum up the ones killed up to the point in time of *j* plus the waiting time of *i* – *j*. Thus, we have the following,

**(c)** Pseudo-code:

Initialize an array R[0, 1, …, n]

Set the first element within the array to 0

Loop **for** i = 1 **to** n:

Initialize each element within R to 0

Loop **for** j = 0 **to** i - 1:

/\* Similar to the relaxation process, find the inverse (max) value.

\* Using equation from part **(b)**. \*/

**if** ( R[i] < R[j] + min[ x[i], f(i – j) ] ):

R[i] = R[j] + min[ x[i], f(i – j) ]

**return** R[n]

**(d)** With the double nested for loops the overall run time is . The overall space needed for this method is .

1. … Describe an efficient algorithm which takes a sequence s of length n, and two strings x and y, and determines if s is an interleaving of repetitions of x and y.

**(a)** Let *IL*(*s*) be a function to represent the sequence *s* that is an interleaving of the two transmissions *x* and *y*.

**(b)** We begin by initializing an empty string sequence to store the interleaving sequence as the base case. Therefore, it will return true for the base case. Next, for any subsequence *x*, *y* > 0 we will recursively call upon subsequences of either string to find any valid matches. Thus, we have the following,

**(c)** Pseudo-code:

*isInterleaving*(s, x, y):

**if** (seq == “ “):

return true

**if** (s[0] == x[0]):

x = x.substring(1, x.length) + x[0] /\* Shift x by 1 character \*/

**if** (*isInterleaving*(s.substring(1, s.length), x, y): /\* Remove first character \*/

**return** true

**if** (s[0] == y[0]):

y = y.substring(1, y.length) + y[0] /\* Shift y by 1 character \*/

**if** (*isInterleaving*(s.substring(1, s.length), x, y): /\* Remove first character \*/

**return** true

**return** false

**(d)** Both the overall time and space needed for this is .

1. … Given the probabilities r1, r2, ..., rn, the costs c1, c2, ..., cn, and the budget B, find the maximum reliability that can be achieved within budget B.

**(a)** Let *R*(*i*, *b*) represent the probability function of maximum reliability that can be achieved through *i* steps and within budget *b*.

**(b)** First, the budget B must be large enough so that *Mi* can be performed, and we must check that the probability of functioning is not zero; . We can then define a new to represent the remaining budget that we have to add redundancy; (where is the sum of costs). Thus, we have the following,

**(c)** Pseudo-code:

*reliability*(*i*, *b*):

**for** *b* **=** 0 **to** *B*:

*R*(0, *b*) = 1

**for** *i* = 1 **to** *n*:

**for** *b* = 0 **to** *B*:

**for** *k* = 1 **to** (*b* / *ci*):

*R*(*i*, *b*) =

**(d)** The overall runtime with the triple nested for loops is . The overall head space required for this is .