Dynamic programming - Exercices

Advanced Algorithms

Master DSC/MLDM/CPS2

1 Matrix chain multiplication

Give a divide-and-conquer algorithm for the matrix chain multiplication problem (the algorithm computes the $m[i, j]$ in a recursive manner).

Give the complexity of the algorithm and justify that it is not polynomial.

2 Longest Common Subsequence

1. Find below a recursive solution:

```
begin
    if X is empty ou Y is empty then Return empty;
    m ← size(X);
    n ← size(Y);
    if x_m = y_n then Return LCS-R(X_{m-1}, Y_{n-1}).x_m;
    else
        Z_1 ← LCS-R(X_{m-1}, Y);
        Z_2 ← LCS-R(X, Y_{n-1});
        if size(Z_1) > size(Z_2) then Return Z_1;
        else Return Z_2;
    end
end
```

Algorithme 1: LCS-R(X,Y)

Show that this algorithm is not polynomial.

2. Write a version of the LCS using a space of size only $2 \times min(m, n)$. We call this version Space-efficient-LCS-LENGTH. What is the drawback of this approach?

3. We now try to give a solution that uses $O(m+n)$ in space while allowing to output the LCS, this is done by using additional divide-and-conquer procedures.

   (a) First, the algo designed in class is a forward (top-down) procedures. Define a simple way to design a backward (bottom-up) algo that works from the bottom right of the matrix and goes to the top-left where one can find the solution. (denote by g the cost function of this approach)

   (b) Characterize the length of the LCS that passes through an entry $[i, j]$ of the matrix built by the LCS-LENGTH algo.

   (c) Show for any index q that minimizes $m(q, k)+g(q, k)$ for any k, there there is an optimal solution that uses the entry (q, k).

   (d) Design an algorithm based on a divide-and-conquer principle using $O(m + n)$ space.

   (e) Prove that the complexity is still in $O(mn)$

3 Longest Monotonically Increasing Subsequence

We have a sequence of integers $x_1, \ldots, x_n$ and we want to extract a subsequence of numbers monotonically increasing of maximal length. For example in the sequence $< 9, 2, 7, 4, 6, 1, 8, 6 >$ the solution is $2, 4, 6, 8$ (note that $4, 2, 7, 8$ is not a subsequence and that $2, 7, 4, 8$ is not an increasing subsequence, so these two examples are not valid).

1. Can you give the total number of all possible subsequences of a sequence of length n (you can suppose that all the numbers in the sequence are distinct)?
2. Give all the possible monotonically increasing subsequences in the above example.
3. Define an algorithm of quadratic complexity to compute the maximal length of an increasing subsequence extracted from $x_1, \cdots, x_n$. Hint: define a notion of optimal substructure in a recursive way and consider subsequences that end at a particular index of the input sequence.
4. Complete the previous algorithm with an approach for finding the (or one) subsequence solution of the problem.
5. We can also solve this problem, using LCS, there are also some connections with graph problems.

4 Printing neatly

We consider the problem of printing neatly a paragraph on a screen or a printer. The input text is a sequence of n words of length $l_1, l_2, \ldots, l_n$ (the length measuring the number of characters). We want to print this paragraph neatly on a number of lines that holds a maximum of M characters each. To reach this objective, we need to define a notion of *"nice printing"* which is defined as follows. If a given line contains words i through j, $i \leq j$, with exactly one space between two words, the number of extra space characters at then end of the line is

$$M - j + i - \sum_{k=i}^{j} l_k$$

which must be nonnegative so that the words fit on the line. We wish to minimize the sum, over **all lines except the last**, of the cube of the numbers of extra space characters at the ends of the lines. The cube of the numbers of extra space characters on one line containing words i through j is defined by:

$$\left(M - j + i - \sum_{k=i}^{j} l_k \right)^3 .$$

1. Find a notion of optimal substructure.
   Hint: define a cost function $C(k)$ corresponding to the total cost the **first** k words (*ie* words 1 through k). To solve the problem, you need to define $C(k)$ by defining a special case to take into account the cost of one line. $C(k)$ must be defined in function of a value $C(j)$ $j \leq k$.
2. Find a recursive solution to the problem.
3. Find a dynamic programming approach using the notion of optimal substructure.
4. Give an algorithm able to print the text according to the solution found.

(1) Recursive MatrixMul (i, j):

```
If i == j:
    return 0
min ← ∞
for k = 1 to j-1:
    q1 ← Recursive MatrixMul (i, k);
    q2 ← Recursive matrixMul (k+1, j);
    q ← q1 + q2 + P_{i-1} P_k P_j ;
    if min > q then
        min ← q
return min
```

$$\therefore T(n) = c + \sum_{k=1}^{n-1} \{T(k) + T(n-k) + c\} \quad \text{if } n \geq 2$$

$$\Rightarrow T(n) = c$$

$$\therefore T(n) \leq 2\sum_{i=1}^{n-1} T(i) + c(n-1)$$

$$\leq 2\sum_{i=1}^{n-1} (c\, i\, 3^{i-1}) + cn$$

$$\leq 2cn \sum_{i=1}^{n-1} 3^{i-1} + cn \quad \text{since } i < n$$

$$\leq 2cn\left(\frac{3^{n-1} - 1}{2}\right) + c(n)$$

$$\leq cn\, 3^{n-1}$$

(1)

```
begin
If X is empty or Y is empty then return empty
m ← Size(X)
n ← Size(Y)
If x_m = y_n then Return
        LCS-R(X_{m-1}, Y_{n-1}) . x_m
else,
    Z1 ← LCS-R(X_{m-1}, Y)   ← T(n-1)
    Z2 ← LCS-R(X, Y_{n-1})   ← T(n-1)
If size(Z1) > size(Z2) then return Z1
else return Z2
end
end
```

$$\therefore T(n,m) = T(n,m-1) + T(n-1,m) + c$$

$$= \left(T(n,m-2) + T(n-1,m-1)\right) +$$

$$\left(T(n-1,m) + T(n-1,m-1)\right) + c$$

$$\geq 2T(n-1,m-1)$$

$$\text{So, } T(n,m) = 2^{\min(n,m)}$$

---

(2) Algorithm Space-efficient_LCS_Length(X,Y).

```
m ← Size(X)
n ← Size(Y)
C = create array [0..1, 0..n]
for i = 1 to k
    c[0,k] ← 0
for i = 1 to m
    c[1,0] ← 0
    for j = 1 to n do
        if x_i = y_j then
            c[1,j] ← c[0,j-1] + 1
        else if c[1,j-1] < c[0,j] then
                c[1,j] ← c[0,j]
            else
                c[1,j] ← c[1,j-1]
    for j = 0 to n do:
        c[0,j] ← c[1,j]
```

a = [1, 4, 3, 3, 2, 5]

b = [1, 4, 2, 5]

for i = 1 ↔ 4

i = 1

(3)(1) The total no. of all possible subsequence of length (n) is $2^n - 1$

(2) $\langle 2, 7 \rangle$, $\langle 2, 4, 6, 8 \rangle$, $\langle 2, 6, 8 \rangle$

j i
9 2 4 6 1 8 6

(3)

```
def long-inc-sub (A):
    n = len(A)
    temp = create anay of size n
    max = 0
    for i = 2 to n
        for j = 1 to i-1
            if A[j] < A[i]:
                temp[i] = temp[j]+1
```

1	1	2	3	3	4	3

```
    if temp[i] > max:
        max = temp[i]
return temp.
```

Time Complexity $O(n^2)$.