3-1 Dynamic Programming

Jun Ma

majun@nju.edu.cn

September 19, 2020

TC 15.1-1

Show that equation $T(n) = 2^n$ follows from equation $T(n) = 1 + \sum_{j=0}^{n-1} T(j)$ and the initial condition $T(0) = 1$.

TC 15.1-1

Show that equation $T(n) = 2^n$ follows from equation $T(n) = 1 + \sum_{j=0}^{n-1} T(j)$ and the initial condition $T(0) = 1$.

Proof.

$$\begin{aligned} T(n) &= 1 + \sum_{j=0}^{n-1} T(j) = 1 + T(n-1) + \sum_{j=0}^{n-2} T(j) = 2T(n-1) \\ &= 2(2^T(n-2)) = \cdots = 2^n T(0) = 2^n \end{aligned}$$

□

TC 15.1-3

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a DP algorithm to solve this modified problem.

TC 15.1-3

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a DP algorithm to solve this modified problem.

- Original:

  More generally, we can frame the values r_n for $n \geq 1$ in terms of optimal revenues from shorter rods:

  $$r_n = \max\left(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\right) . \tag{15.1}$$

TC 15.1-3

Consider a modification of the rod-cutting problem in which, in addition to a price p_i for each rod, each cut incurs a fixed cost of c. The revenue associated with a solution is now the sum of the prices of the pieces minus the costs of making the cuts. Give a DP algorithm to solve this modified problem.

- Original:

  More generally, we can frame the values r_n for $n \geq 1$ in terms of optimal revenues from shorter rods:

  $$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1) \ . \tag{15.1}$$

- With fixed cost of c:

$$r_n = \max(p_n, r_1 + r_{n-1} - c, r_2 + r_{n-2} - c, ..., r_{n-1} + r_1 - c)$$

TC 15.2-2

Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, \cdots, A_n \rangle$, the s table computed by MATRIX-CHAIN-ORDER, and the indices i and j . (The initial call would be MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$.)

TC 15.2-2

Give a recursive algorithm MATRIX-CHAIN-MULTIPLY(A, s, i, j) that actually performs the optimal matrix-chain multiplication, given the sequence of matrices $\langle A_1, A_2, \cdots, A_n \rangle$, the s table computed by MATRIX-CHAIN-ORDER, and the indices i and j . (The initial call would be MATRIX-CHAIN-MULTIPLY$(A, s, 1, n)$.)

Answer.

```
1: procedure MATRIX-CHAIN-MULTIPLY(A, s, i, j)
2:     if i=j then
3:         return A[i]
4:     b ←MATRIX-CHAIN-MULTIPLY(A, s, i, s[i, j])
5:     c ←MATRIX-CHAIN-MULTIPLY(A, s, s[i, j] + 1, j)
6:     return b ∗ c
```

□

TC 15.2-4

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

Answer.

- How many vertices does it have?

TC 15.2-4

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

Answer.

- How many vertices does it have?
  - The vertices of the subproblem graph are the ordered pairs V_{ij}, where $i \leq j$
  - $\sum_{i=1}^{n} \sum_{j=i}^{n} 1 = \frac{n(n+1)}{2}$
- How many edges does it have?

TC 15.2-4

Describe the subproblem graph for matrix-chain multiplication with an input chain of length n. How many vertices does it have? How many edges does it have, and which edges are they?

Answer.

- How many vertices does it have?
  - The vertices of the subproblem graph are the ordered pairs V_{ij}, where $i \leq j$
  - $\sum_{i=1}^{n} \sum_{j=i}^{n} 1 = \frac{n(n+1)}{2}$
- How many edges does it have?
  - A subproblem V_{ij} has exactly $j - i$ subproblems.
  - $\sum_{i=1}^{n} \sum_{j=i}^{n} (j - i) = \frac{(n-1)n(n+1)}{6}$

□

TC 15.3-3

Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the sequence of matrices so as to maximize, rather than minimize, the number of scalar multiplications. Does this problem exhibit optimal substructure?

TC 15.3-3

Consider a variant of the matrix-chain multiplication problem in which the goal is to parenthesize the sequence of matrices so as to maximize, rather than minimize, the number of scalar multiplications. Does this problem exhibit optimal substructure?

Answer.

Yes! □

TC 15.3-5

Suppose that in the rod-cutting problem of Section 15.1, we also had limit l_i on the number of pieces of length i that we are allowed to produce, for $i = 1, 2, \cdots, n$. Show that the optimal-substructure property described in Section 15.1 no longer holds.

TC 15.3-5

Suppose that in the rod-cutting problem of Section 15.1, we also had limit l_i on the number of pieces of length i that we are allowed to produce, for $i = 1, 2, \cdots, n$. Show that the optimal-substructure property described in Section 15.1 no longer holds.

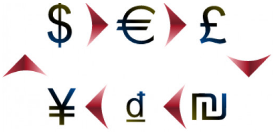
Proof.

- Sub-problems can not be solved **independently**.
  - The number of pieces of length li used on one side of the cut affects the number allowed on the other.

□

TC 15.3-6(Exchange Currency)

- n different currencies
- Given an exchange rate r_{ij} for each pair of currencies i and j
- Let c_k be the commission that you are charged when you make k trades.
- Try to find the **best** sequence of exchanges from currency 1 to currency n trades.



TC 15.3-6(Exchange Currency)

Q1

The problem exhibits **optimal substructure** if $c_k = 0$ for all $k = 1, 2, \cdots, n$.

(Assume no loop has an overall exchange rate greater than 1)

Proof.

- Let k denote a currency which appears in an optimal sequence S of trades to go from currency 1to currency n
- p_k: $1 \to \cdots \to k$ and q_k: $k \to \cdots \to n$
- Then p_k and q_k are both optimal sub-sequences. Take p_k for instance:
  - Suppose that p_k wasn't optimal but that p'_k was.
  - Then, the sequence $p'_k q_k$ would be a sequence better than S

The same argument applies to q_k

□

TC 15.3-6(Exchange Currency)

Q2

The problem does not necessarily exhibit **optimal substructure** if c_k are arbitrary values.

- I do not understand the problem. How is c_k used?

TC 15.4-3

Give a memorized version of LCS-LENGTH that runs in $O(mn)$ time.

TC 15.4-3

Give a memorized version of LCS-LENGTH that runs in $O(mn)$ time.

```
procedure MEM-LCS-LENGTH(X, Y, i, j, c, b)
    if c[i, j] > −1 then
        return c[i, j]
    else if i = 0 or j = 0 then
        c[i, j] ← 0
    else if x[i] = y[j] then
        c[i, j] ← MEM-LCS-LENGTH(X, Y, i − 1, j − 1, c, b)+1
        b[i, j] ← "↖"
    else
        p ← MEM-LCS-LENGTH(X, Y, i − 1, j, c, b)
        q ← MEM-LCS-LENGTH(X, Y, i, j − 1, c, b)
        c[i, j] ← max (p, q)
        if p ≥ q then
            b[i, j] ← "↑"
        else
            b[i, j] ← "←"
    return c[i, j]
```

TC 15.4-5

Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Answer.

TC 15.4-5

Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Answer.

LIS→LCS

TC 15.4-5

Give an $O(n^2)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

Answer.

LIS→LCS

```
procedure LIS(A)
    B ← SORT(A)
    c, b ← LCS-LENGTH(A, B)
    PRINT-LCS(b, A, A.length, B.length)
```

□

$\boxed{\text{LCS} \rightarrow \text{LIS ?}}$

LCS→ LIS ?

An example

$$A = h, b, e, f, g, b$$
$$B = b, e, h, i, g, x, b$$
$$LCS(A, B) = b, e, g, b$$

LCS→ LIS ?

An example

$$A = h, b, e, f, g, b$$
$$B = b, e, h, i, g, x, b$$
$$LCS(A, B) = b, e, g, b$$

Transformation

- Build two maps from A:
  - $M_1 = \{\langle h, 1\rangle, \langle b, \{6, 2\}\rangle, \langle e, 3\rangle, \langle f, 4\rangle, \langle g, 5\rangle\}$
  - $M_2 = \{\langle 1, h\rangle, \langle 2, b\rangle, \langle 3, e\rangle, \langle 4, f\rangle, \langle 5, g\rangle, \langle 6, b\rangle\}$
- Apply M_1 to B and obtain $B' = 6, 2, 3, 1, -, 5, -, 6, 2$
- Find LIS of B', $LIS(B') = 2, 3, 5, 6$
- Apply M_2 to B' and obtain $LCS(A, B) = b, e, g, b$

LCS/LIS in $O(n \lg n)$

TC 15.4-6

Give an $O(n \lg n)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

LCS/LIS in $O(n \lg n)$

TC 15.4-6

Give an $O(n \lg n)$-time algorithm to find the longest monotonically increasing subsequence of a sequence of n numbers.

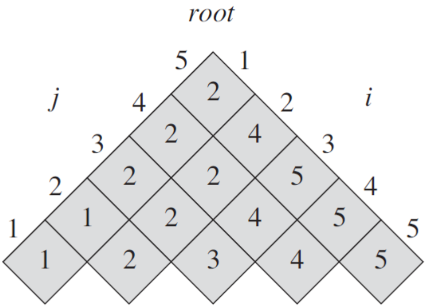
Hint

- $S_i =$ {s| s is a monotonically increasing sub-sequence of$A[1, \cdots, i]$}
- $S_{ij} = \{s \in S_i | s.length = j\}$
- $c_{ij} = \min_{s \in S_{ij}} \textsc{LastCharOf}(s)$

$$c_{ij} = \begin{cases} s[i] & \text{if } j = \min\limits_{1 \le k \le j, s[i] < c_{i-1 k}} (k) \\ c_{(i-1)j} & \text{otherwise} \end{cases}$$

TC 15.5-1

Write pseudocode for the procedure CONSTRUCT-OPTIMAL-BST(*root*) which, given the table *root*, outputs the structure of an optimal binary search tree.



k_2 is the root
k_1 is the left child of k_2
d_0 is the left child of k_1
d_1 is the right child of k_1
k_5 is the right child of k_2
k_4 is the left child of k_5
k_3 is the left child of k_4
d_2 is the left child of k_3
d_3 is the right child of k_3
d_4 is the right child of k_4
d_5 is the right child of k_5

TC 15.5-1

```
procedure CONSTRUCT-OPTIMAL-BST(root,i,j,p)
    if p=0 then
        PRINT("k"+p+" is the root")
    else if i > j then
        if j < p then
            PRINT("d"+j+" is the left child of k"+p)
        else
            PRINT("d"+j+" is the right child of k"+p)
    else
        if j < p then
            PRINT("k"+root[i, j]+" is the left child of k"+p)
        else
            PRINT("k"+root[i, j]+" is the right child of k"+p)
        CONSTRUCT-OPTIMAL-BST(root,i,root[i, j]-1,root[i, j])
        CONSTRUCT-OPTIMAL-BST(root,root[i, j]+1,j,root[i, j])
```

Printing neatly

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer.

- The input text is a sequence of n words of lengths $l_1, l_2, \cdots, l_n$, measured in characters.
- We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each.
- **neatness**: If a given line contains words i through j , where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the end of the line is $M - j + i - \sum_{k=i}^{j} l_k$.
- **minimize** the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of lines.

(Sub-)problems?

15-4 *Printing neatly*

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the 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 cubes of the numbers of extra space characters at the ends of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.

Optimal Substructure

15-4 Printing neatly

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the 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 cubes of the numbers of extra space characters at the ends of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.

first line $\rightarrow$ last line

15-4 Printing neatly

Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the 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 cubes of the numbers of extra space characters at the ends of lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.

last line $\rightarrow$ first line

- Define $extras[i,j] = M - j + i - \sum_{k=i}^{j} l_k$
- $lc[i,j]$: cost of including a line containing words i through j

$$lc[i,j] = \begin{cases} \infty & \text{if } extras[i,j] < 0 \\ 0 & \text{if } j = n \text{ and } extras[i,j] \geq 0 \\ extras[i,j] & \text{otherwise} \end{cases}$$

- $c[j]$: the cost of an optimal arrangement of words $1, ..., j$

$$c[j] = \begin{cases} 0 & \text{if } j = 0, \\ \min_{1 \leq i \leq j} (c[i-1] + lc[i,j]) & \text{if } j > 0. \end{cases}$$

```
PRINT-NEATLY(l, n, M)
▷ Compute extras[i, j] for 1 ≤ i ≤ j ≤ n.
for i ← 1 to n
    do extras[i, i] ← M − l_i
        for j ← i + 1 to n
            do extras[i, j] ← extras[i, j − 1] − l_j −


▷ Compute lc[i, j] for 1 ≤ i ≤ j ≤ n.
for i ← 1 to n
    do for j ← i to n
            do if extras[i, j] < 0
                  then lc[i, j] ← ∞
                elseif j = n and extras[i, j] ≥ 0
                  then lc[i, j] ← 0
                else lc[i, j] ← (extras[i, j])^3


▷ Compute c[j] and p[j] for 1 ≤ j ≤ n.
c[0] ← 0
for j ← 1 to n
    do c[j] ← ∞
        for i ← 1 to j
            do if c[i − 1] + lc[i, j] < c[j]
                  then c[j] ← c[i − 1] + lc[i, j]
                       p[j] ← i
return c and p
```


https://blog.csdn.net/yxc135/article/details/9621643


Thank
You!