Dynamic Programming – III

We have applied DP on two concrete examples:
(i) Matrix chain order (ii) Rod-cutting

Optimization problem S

Will DP yield efficient solution to S?

Characteristics of problems which admit DP technique.

(i) Express optimal solution of S in terms of optimal solution of subproblems of S.

In our examples:

(i) MCM $\quad m_{ij} = \min_{i \le l < j-1} \{ \underbrace{m_{il}}_{\text{optimal solution}} + m_{l+1,j} + p_i p_{l+1} p_{j+1} \}$

$$\underbrace{(A_1 \cdots A_l)}_{\text{optimal sol}} \underbrace{(A_{l+1} \cdots A_n)}_{\text{optimal sol}}$$

Optimal solution of S contains optimal solution of (some) subproblems.

(ii) $\quad R[i] = \max_{1 \le l \le i} \{ P[l] + \underbrace{R[i-l]}_{\text{optimal solution}} \}$

To see where this principle (structure of optimal solution) gets violated,

consider a variation of Rod cutting problem

$l[i]$, $1 \leq i \leq n$ — no. of pieces of length i should be bounded $l[i]$

$i \rightarrow$	1	2	3
$P[i]$	10	15	10
$l[i]$	2	—	—

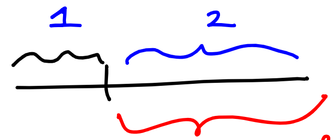
$n = 3$ (In the absence of any constraint the optimal solution is $1+1+1$, 30)

with l_i's given the optimal solution is $1+2$, 25

$$R[3] \neq \max_{1 \leq l \leq 3} \left(P[l] + R[3-l] \right)$$

$R[2] = 1+1 = 20$, with $l = 1$, $P[1] + R[2] = 30$

Alternatively, to get an optimal solution of $n=3$



optimal solution is $1+1$ (20)

not $\underline{\underline{2}}$ (15)

optimal solution for $n=3$, uses a solution to subproblem for $n=2$ — but not optimal solution to this subproblem.

(ii) Second characteristic

Overlapping subproblems

Subproblem graph

(nodes as the subproblems)

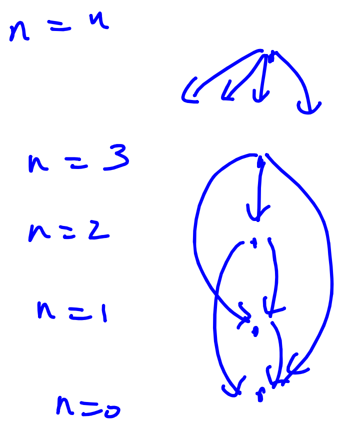
(Edge from S_1 to S_2 if optimal solution of S_1 depends on optimal solution of S_2, as seen from our equation for optimal solution)

Rod-cutting problem

<u>n</u>

$$R[n] = \max_{1 \le l \le n} \{ P[l] + R[n-l] \}$$

$R[n]$ depends on $R[n-1], R[n-2], \dots, R[0]$



$R[2]$ is being used in solving $R[3], R[4]$ ....

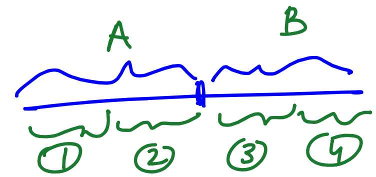
set of subproblems for $n=3$ and $n=4$ is not disjoint.

In the matrix-chain-order example, we saw (m,14) in the recursion graph that many subproblems arise more than once.

In contrast consider Divide and conquer technique.

here also solution to S is constructed from solutions to subproblems but these subproblems are disjoint

For example consider mergesort



Subproblems of 1 are disjoint from subproblems of B or subproblems of 4

(iii) Number of distinct subproblems is small (polynomial in the data of the problem).

Example in Rod cutting $O(n)$ problems

MCO — $O(n^2)$ problems

An Example of (another) problem for which we may use DP.

Problem Largest Common Subsequence (LCS)

A sequence is a finite string over some alphabet.

For example if alphabet is $\{a, b, c, d\}$

acdbaabdcb is a sequence over this alphabet.

A subsequence of a given sequence S is a sequence that arise by dropping some letter from S.

example: aba is a subsequence of the sequence above.

bbb is another " .

More formally, Let $s = x_1 x_2 \dots x_n$ be a sequence

then $x_{i_1} x_{i_2} \dots x_{i_k}$, where $1 \le i_1 < i_2 < i_3 \dots < i_k \le n$

is a subsequence of s.

LCS Given two sequences, u and v, find the longest common subsequence of u, v.

Example $u = \overline{a}bdb\overline{a}d$

$v = \underline{a}c\underline{b}\underline{d}c\underline{a}$

abd is a common subsequence of u, v.

abda is another common subsequence.

longest subsequence of u, v

DNA may be considered as a string over alphabet {A, C, G, D}
?

Given two DNA sequences (could be very long, hundreds of letters), we wish to find similarity (or match) between them.

LCS may be one way of answering this.

$O(n^2)$ solution for LCS problem using DP.

Equation for optimal solution:

What should we take as subproblems?

$x = x_1 \cdots x_m$ $\quad (i,j)$ $\quad$ LCS of $x_1 \cdots x_i$, $y_1 \cdots y_j$ $\quad c[i,j]$

$y = y_1 \cdots y_n$ $\quad 1 \le i \le m$, $1 \le j \le n$

$c[i,j]$ we also admit $i=0$, $j=0$

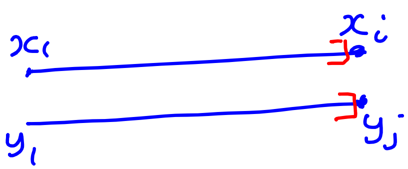
$i=0$ corresponds to empty prefix of x.

$j=0$ corresponds to empty prefix of y.

$c[0,j] = 0$

$c[i,0] = 0$

$$
c[i,j] = \begin{cases} c[i-1, j-1] + 1 & \text{if } x_i = y_j \\ \max\{ c[i, j-1], c[i-1, j] \} & \text{if } x_i \neq y_j \end{cases} \qquad i, j \geq 1
$$



$\underline{x_i = y_j}$

If s is the longest common subsequence of $x_1 \ldots x_{i-1}$, $y_1 \ldots y_{j-1}$

then $s\,x_i$ is lcs of $x_1 \ldots x_i$ and $y_1 \ldots y_j$

$$c[i,j] \geq c[i-1, j-1] + 1$$

To show that $c[i,j] \leq c[i-1, j-1] + 1$.

Consider t be a common subsequence of $x_1 \ldots x_i$ and $y_1 \ldots y_j$ of length $c[i,j]$

$t = x_{l_1} \cdots x_{l_k}$ $\qquad k = c[i,j]$

$= y_{r_1} \cdots y_{r_k}$

Case (i) $\quad l_k < i, \quad r_k < j$

not possible because $t\, x_{l_k}$ would give a longer common subsequence.

Case (ii) $\quad l_k = i, \quad r_k < j$

$t_1 = x_{l_1} \cdots x_{l_{k-1}}$ is a subsequence of $x_1 \cdots x_{i-1}$, $y_1 \cdots y_{j-1}$

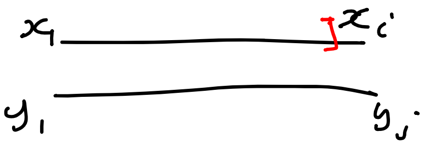
$= y_{r_1} \cdots y_{r_{k-1}}$

$t_1 \leq c[i-1, j-1]$

$c[i,j] = |t| = |t_1| + 1 \leq c[i-1, j-1] + 1$

All the other cases are similar (left as easy exercise) $\square$

$x_i \neq y_j$



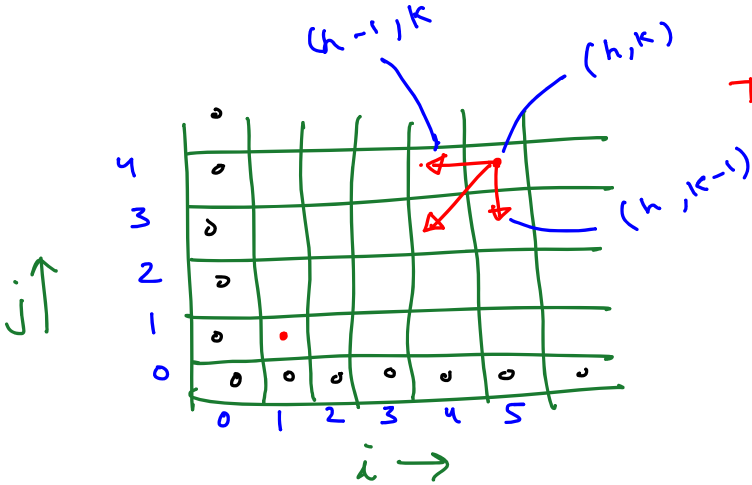
either $i_k < i$

or $j_k < j$

In this case t is a subsequence of $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_j$ $\Big\}$ $c[i-1, j]$

$$\Longrightarrow \quad c[i,j] = \max \left\{ \begin{array}{l} c[i-1, j], \\ c[i, j-1] \end{array} \right\}$$

[To prove the equation, in detail is left as an exercise]



n × m table

To fill a square in this table we need to look at only 3 previously constructed values.

⟹ This table can be constructed in $O(n^2)$ time.

Entry at coordinates (n, m) contains answer to the original problem.

The longest subsequence may be constructed by storing in the table (a constant amount of) additional information.

[just the direction of the arrow].