Math 182 Lecture 12

Chapter 6 Dynamic Programming cont.

Recall: dynamic programming is often useful for optimization problems i.e. where you have many possible configurations, want to find one w/ best "value".

(1) Characterize structure of optimal solution.
(2) Recursively define value of optimal solution.
(3) Compute value of optimal solution either:
   (a) recursively w/ memoization
   (b) iteratively solve subproblems in a bottom-up manner.
(4) Construct an optimal solution from computed information.

§6.2 Matrix-chain multiplication

Sps we want to evaluate

$$A_1 A_2 \cdots A_n$$

there are multiple ways to evaluate:

E.g. $n=4$ $\quad A_1 A_2 A_3 A_4$

5 ways to evaluate:

$A_1 | (A_2 (A_3 A_4))$

$A_1 | ((A_2 A_3) A_4)$

$(A_1 A_2) | (A_3 A_4)$

$(A_1 (A_2 A_3)) | A_4$

$((A_1 A_2) A_3) | A_4$

A $m \times n$

B $p \times q$

$A \times B$ compatible if

$n = p$

columns A = # rows of B.

$A \times B = C$ $\quad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

$(m \times n) \quad (n \times q) \quad (m \times q)$

MATRIX-MULTIPLY(A, B)

```
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7                  for k = 1 to A.columns
8                      c_ij = c_ij + a_ik · b_kj
9      return C
```

total number of scalar multiplications =
A.rows × B.columns × A.columns

$= mnq$

Example: $\langle A_1, A_2, A_3 \rangle$

$10 \times 100 \quad 100 \times 5 \quad 5 \times 50$

better $\rightarrow (A_1 A_2) A_3 \rightsquigarrow 7500$ sc. mults.

worse $\rightarrow A_1 (A_2 A_3) \rightsquigarrow 75000$ sc mults

The **matrix-chain multiplication problem** is as follows: given a chain $\langle A_1, A_2, \ldots, A_n \rangle$ of n matrices, where for $i = 1, 2, \ldots, n$, matrix A_i has dimension $p_{i-1} \times p_i$, fully parenthesize the product $A_1 A_2 \cdots A_n$ in a way that minimizes the number of scalar multiplications.

Really This is a problem about $p = \langle p_0, p_1, \ldots, p_n \rangle$. not $A_1, \ldots, A_n$.

Count # of parenthesizations:

Let $P(n)$ be # of ways to parenthesize $A_1 \cdots A_n$.

$n = 1 \quad P(n) = 1$

Sps $n \geq 2 \quad A_1 \cdots A_n$

to choose parenthesization, choose index to split at

$(A_1 \cdots A_k)(A_{k+1} \cdots A_n) \quad 1 \leq k \leq n-1$

and choose a parenthesization for $A_1 \cdots A_k$ and $A_{k+1} \cdots A_n$ recursively.

$$\rightsquigarrow \quad P(n) = \sum_{k=1}^{n-1} P(k) P(n-k)$$

$$P(n) = \begin{cases} 1 & \text{if } n = 1 \\ \sum_{k=1}^{n-1} P(k)\,P(n-k) & \text{if } n \geq 2. \end{cases}$$

$P(n)$ seq is called the Catalan numbers

$$P(n) = \Omega\left(4^n / n^{3/2}\right)$$

easier: $P(n) = \Omega(2^n)$ Exponential → bad

Let's design dyn. prog. sol.

Step 1: Structure of optimal parenthesization

notation: $A_{i..j} := A_i \cdots A_j$ $\quad i \leq j$

Sps $i < j$ and we have an optimal paren. for $A_{i..j}$.

there must be some index k s.t. $1 \leq k \leq n-1$ and

$$A_{i..j} = (A_{i..k})(A_{k+1..j})$$

some parens here. some parens here.

Must use optimal parenthesizations by swapping-out argument.

Step 2: Recursive solution.

Let $m[i,j] :=$ minimal # of scalar multiplications needed for some paren. of $A_{i..j}$.

$1 \leq i \leq j \leq n$.

Base case: $i = j$.

A_i $p_{i-1} \times p_i$

$m[i,i] = 0$

Sps $i < j$, Then There will be some k as above where

$A_{i..j} = (A_{i..k})(A_{k+1..j})$

$m[i,j] = m[i,k] + m[k+1,j] + p_{i-1} p_k p_j$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \le k \le j-1} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \} \end{cases}$$

→ $m[i,j]$ will be array of optimal sols to subproblems.

For purposes of reconstructing answer, will also store:

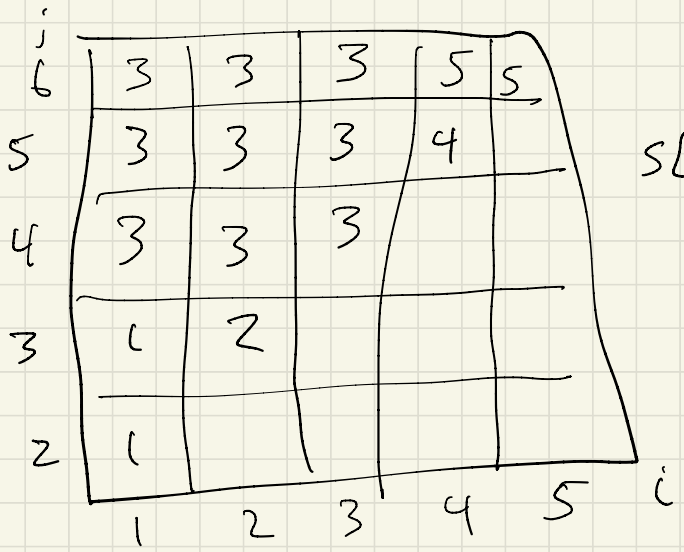
$i < j$ $s[i,j] = k$ where k is s.t. $i \le k \le j-1$ and k achieves minimum in recursive formula.

$$m[i,j] = \min_{i \le k \le j-1} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \}$$

$j - k - 1$, $k - i < j - i$

Subproblem of certain chain length depends only on subproblems of smaller lengths.

$A_1 A_2 \cdots A_6$



$s[i,j]$

$A_{1..6}$ $s[1,6] = 3$

$(A_{1..3}\ A_{4..6})$ $s[1,3] = 1$ $s[4,6] = 5$

$\big((A_1(A_2A_3))((A_4A_5)A_6)\big)$

Step 3 Compute optimal costs:

$A_1, \ldots, A_n$ $\quad A_i : p_{i-1} \times p_i$

$p = \langle p_0, p_1, \ldots, p_n \rangle$

MATRIX-CHAIN-ORDER(p)

```
1   n = p.length − 1
2   let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
3   for i = 1 to n
4       m[i, i] = 0
5   for l = 2 to n                    // l is the chain length
6       for i = 1 to n − l + 1
7           j = i + l − 1
8           m[i, j] = ∞
9           for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

Lines 1-4 create $m[i,j]$, $s[i,j]$ and initialize $m[i,i] = 0$ $\forall i$.

For loop of line 5-13 iterate through all chain lengths from smallest to largest.

For loop line 6-13 iterate through all pairs i,j w/ chain length l.

In lines 8-13 apply reccurence relation to store $m[i,j]$ and $s[i,j]$.

Running time is $O(n^3)$, actually $\Theta(n^3)$

Step 4: Reconstructing an optimal solution.

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2        print "A_i"
3  else print "("
4        PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5        PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6        print ")"
```

If $i = j$ print "A_i" since only one matrix

O/w print

"($A_{[..s[i,j]}$ $A_{s[i,j]+1..j}$)"

recursively these will be optimal parenthesizations for corresponding subproducts.

Chapter 7 Greedy Algorithms

Greedy algorithms also work sometimes for optimization problems.
A greedy algorithm is one where you can make locally optimal) choice in the hopes it leads to globally optimal solution.

§7.1 Activity-selection problem

Sps you have to schedule a common resource (e.g. a room), you recieve large # of requests
goal: want to accomadate most # of distinct requests as possible.

Specifically:

$S = \{a_1, \ldots, a_n\}$ of proposed activities

for each activity a_i have start time s_i finish time f_i w/ $0 \leq s_i < f_i < \infty$

a_i takes place during $[s_i, f_i)$

We say a_i and a_j compatible if $[s_i, f_i) \cap [s_j, f_j) = \emptyset$

equiv $f_i \le s_j$ or $f_j \le s_i$.

The **activity-selection problem**, we want to find a maximum-size subset of $S = \{a_1, \ldots, a_n\}$ of mutually-compatible activities. Note that we want to maximize the *number* of activities scheduled, not necessarily the total *duration* of the activities which are scheduled.

For convenience, will assume activities are listed in increasing order of finish time:

$$f_1 \le f_2 \le f_3 \le \cdots \le f_{n-1} \le f_n$$

E.g.

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	8	10	11	12	14	16

$\{a_3, a_9, a_{11}\}$ mut. comp.

$\{a_1, a_4, a_8, a_{11}\}$ mut. comp. ← optimal sols.

$\{a_2, a_4, a_9, a_{11}\}$ mut. comp. ←