CSE 201 2-18-20

Exercise

Write the 5 Parenthesizations of A·B·C·D.

find # of Paren. of 5 factors.

Exercise

let

$P(n)$ = # of Parenthesizations of $A_1 A_2 \cdots A_n$

Show $P(n)$ satisfies!

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

check: $P(1) = 1$

$$P(2) = P(1)\cdot P(1) = 1$$

$$P(3) = P(1)\cdot P(2) + P(2)\cdot P(1) = 2$$

$$P(4) = P(1)\cdot P(3) + P(2)\cdot P(2) + P(3)\cdot P(1)$$

$$= 1\cdot 2 + 1\cdot 1 + 2\cdot 1 = 5$$

Difficult exercise: show

$$P(n) = \frac{1}{n}\binom{2n-2}{n-1}$$

solves the recurrence

we have

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

Problem

Given $n \geq 1$, $A_1 \cdot A_2 \cdots A_n$ where A_i is of size $p_{i-1} \times p_i$ $(1 \leq i \leq n)$, find Paren. of $A_1 \cdots A_n$ with min # of real multiplications

As usual, 2 Problems:

- min # of basic ops. necessary (Value of opt. soln.)
- optimal Parenthesization of $A_1 \cdots A_n$

observe: any Parenthesization of $A_i \cdots A_j$ $(1 \le i \le j \le n)$ has a top level multiplication:

$$A_i \cdots A_j = (A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$$

↑ "split point"

where $i \le k < j$.

<u>claim</u>

If $A_i \cdots A_j$ is optimally Parenthesized, then so are both factors $(A_i \cdots A_k)$ and $(A_{k+1} \cdots A_j)$

"Split Point" = k

i.e.

$$\underbrace{A_i \cdots A_j}_{p_{i-1} \times p_j} = \underbrace{(A_i \cdots A_k)}_{p_{i-1} \times p_k} \cdot \underbrace{(A_{k+1} \cdots A_j)}_{p_k \times p_j}$$

if optimal — optimal — optimal

then both factors are optimal

(converse is false: exercise)

Proof.

Assume, to get a -X, that $(A_i \cdots A_k)$ is not optimally paren. Then some other parenthesization of $(A_i \cdots A_k)$ yields lower cost than the paren. inherited from the opt. paren. of $A_i \cdots A_j$.

By replacing $(A_i \cdots A_k)$ with the better parenthesization, we obtain a paren. of $A_i \cdots A_j$ of total cost less than the optimal one, which is impossible. ∎

notation

$m[i,j]$ = min # of real multiplications necessary to compute $A_i \cdots A_j$ $(1 \le i \le j \le n)$

initialize: $m[i,i] = 0$ $(1 \le i \le n)$

Val. of opt. soln $= m[1,n]$

By above proof

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

Problem: don't know split point.

Recurrence formula

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right) & i < j \end{cases}$$

i.e.

$$m[i,i] = 0$$

$$m[i,i+1] = \cancel{m[i,i]}^{0} + \cancel{m[i+1,i+1]}^{0} + p_{i-1} p_i p_{i+1}$$

$$m[i,i+2] = \min \begin{cases} \cancel{m[i,i]} + m[i+1,i+2] + p_{i-1} p_i p_{i+2} \\ m[i,i+1] + \cancel{m[i+2,i+2]} + p_{i-1} p_{i+1} p_{i+2} \end{cases}$$

$$\vdots$$

$$m[i,i+4] = \min(\text{-- 4 numbers --})$$

Exercise

Write algorithm taking input $p[0 \ldots n]$, that fills table $m[i,j]$ & returns $m[1,n]$.

let $s[i,j]$ be the k-value (i.e. split-point) that produces min in computation of $m[i,j]$

modify previous algorithm so as to fill in $s[i,j]$ for $1 \le i < j \le n$.

Exercise

write a recursive algorithm that backtraces through $s[i,j]$ to print out opt. paren of $A_1 \cdots A_n$.

more Problems:

- All Pairs shortest Paths (25.2) (3-dim. table)
- longest Com. Subseq. (15.4)
- Opt. Binary search tree (15.5)

Greedy Algorithms

Ex. Continuous Knapsack

- maximize: $\sum_{i=1}^{n} x_i v_i$
- subject to: $\sum_{i=1}^{n} x_i w_i \leq W$

where now: $0 \leq x_i \leq 1$.

Feasible soln. is $x = (x_1, \ldots, x_n)$ such that $\sum_{i=1}^{n} x_i w_i \leq W$

Greedy strategy: make a locally optimal choice, reduce to sub-instance, iterate on that sub-instance

i.e. rank objects from "best" to "worst", steal them in that order.

Exercise

Write pseudo-code, where "best" function $f(i)$ is part of code.

choices for $f(i)$:

- $f(i) = v_i$ (value of i^{th} obj)
- $f(i) = \frac{1}{w_i}$ (inverse weight of i^{th} obj)
- $f(i) = \frac{v_i}{w_i}$ (val. per unit weight of obj i.)

```
Knapsack (v, w, W)
1.) n ← length[v]
2.) for i ← 1 to n
3.)       x[i] ← 0
4.) weight ← 0
5.) while weight < W
*6.)      i ← THE "BEST" REMAINING OBJECT
7.)       if weight + w[i] ≤ W
8.)             x[i] ← 1
9.)             weight ← weight + w[i]
10.)            mark i AS INCLUDED.
11.)      else
12.)            x[i] ← (W - weight) / w[i]
13.)            weight ← W
14.) return x
```

GREEDY LOOP

THE "BEST" OBJECT IN (6) CAN BE INTERPRETED IN SEVERAL WAYS.

IN GENERAL WE DEFINE A <u>SELECTION FUNCTION</u> $f(i)$ WHICH ENCODES THE DESIRABILITY OF OBJECT i. LINE (6) THEN MAXIMIZES f OVER ALL REMAINING (UNMARKED) OBJECTS.

Ex. $n=5$, $W=10$

i	1	2	3	4	5	order
v_i	2	3	6.6	4	6	3, 5, 4, 2, 1
w_i	1	2	3	4	5	1, 2, 3, 4, 5
$\frac{v_i}{w_i}$	2	1.5	2.2	1	1.2	3, 1, 2, 5, 4

$f(i)$	x					Value
v_i	0	0	1	.5	1	14.6
$1/w_i$	1	1	1	1	0	15.6
v_i/w_i	1	1	1	0	.8	16.4

Theorem

If we maximize v_i/w_i on line 6, then Knapsack(v, w, W) returns an optimal solution.