CS 410/584, Algorithm Design & Analysis: Lecture 1

Dynamic Programming

Two elements — Bellman's principle

- Optimal substructure: solution contains optimal solutions to subproblems (Correctness)

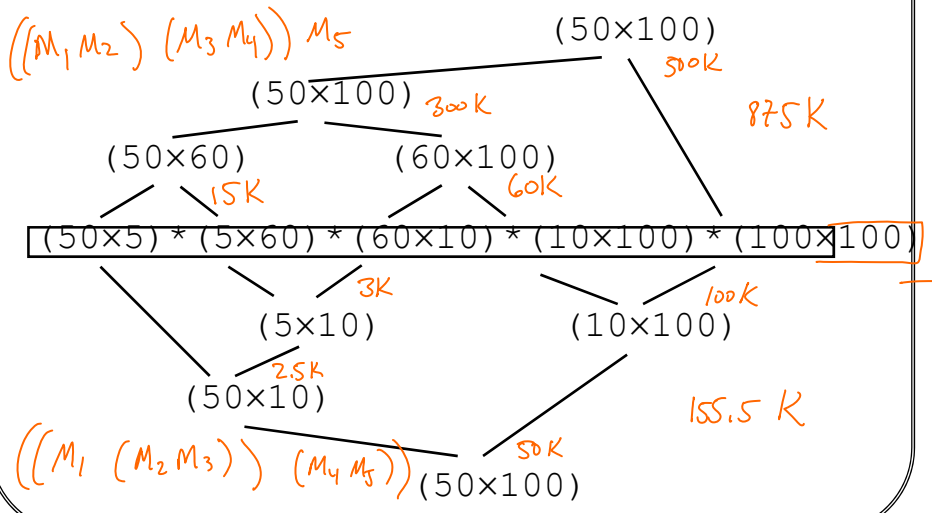
  LCS(7,7) (as sequence) contains LCS(6,6) if $a_7 = b_7$

- Overlapping subproblems: if done recursively (Utility)

  LCS(6,5) used for LCS(7,5), LCS(6,6), LCS(7,6)

Matrix Chain Product

Multiplying an n × p matrix and a p × m matrix takes n·p·m scalar mults.



Recursive Approach

Have matrices $M_1 * M_2 * \ldots * M_n$

Let `Mult[i,j]` be cost to multiply $M_i * \cdots * M_j$

Let $p_0, p_1, \ldots, p_n$ be dimensions such that M_i has dimension

M_i has dims $p_{i-1} \times p_i$

Assume `Mult[i,i] =` 0

Then `Mult[i,j] =`

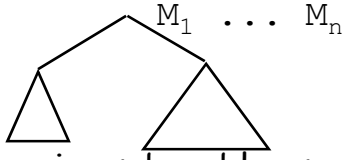
`min (Mult[i,k]+Mult[k+1,j]+`$p_{i-1} p_k p_j$`)`

$i \le k < j$

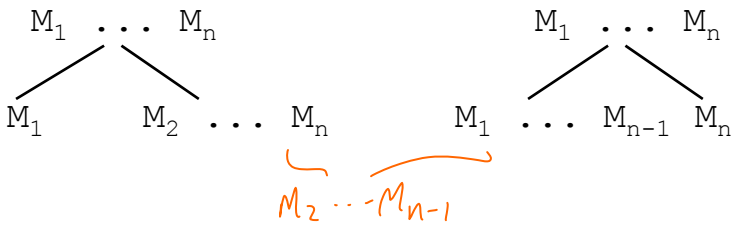
$p_{i-1} \times p_k$ $\quad$ $p_k \times p_j$

Has Necessary Properties

- optimal substructure



- overlapping subproblems



$M_2 \cdots M_{n-1}$

How many subproblems? $\sim \frac{n^2}{2}$

Build a Table

p_0	p_1	p_2	p_3	p_4	p_5
50	5	60	10	100	100

in 1000's

i \ j	5	4	3	2	1
1	83	33	5.5	15	0
2	58	8	3	0	
3	160	60	0		
4	100	0			
5	0				

In 1000's of mults.

Mult[i,j] Values

1,3
1,1 + 2,3 + 50*5*10 = 0 + 3 + 2.5 = 5.5
1,2 + 3,3 + 50*60*10 = 15 + 0 + 30 = 45

2,4
2,2 + 3,4 + 5*60*100 = 0 + 60 + 30 = 90
2,3 + 4,4 + 5*10*100 = 3 + 0 + 5 = 8 ←

3,5
3,3 + 4,5 + 60·10·100 = 0 + 100 + 60 = 160 ←
3,4 + 5,5 + 60·100·100 = 60 + 0 + 600 = 660

Mult[i,j] Values 2

```
1,4    1,1 + 2,4 + 50* 5 *100
       1,2 + 3,4 + 50* 60 *100
       1,3 + 4,4 + 50* 10 *100

2,5    2,2 + 3,5 + 5* 60 *100
       2,3 + 4,5 + 5* 10 *100
       2,4 + 5,5 + 5* 100 *100
```


Mult[i,j] Values 3

```
1,5    1,1 + 2,5 + 50* 5 *100
       1,2 + 3,5 + 50* 60 *100
       1,3 + 4,5 + 50* 10 *100
       1,4 + 5,5 + 50* 100 *100
```

Complexity

$O(n^3)$

Memoizing

"memoized" f

Dynamic Programming in a recursive (top-down) framework

- Whenever you compute the value of a subproblem, store it for later lookup

`f` becomes `mf` plus `MEMO` table

```
mf(x)
    if MEMO[x] defined then {}
    else MEMO[x] ← f(x)
    return MEMO[x]
```

Replace all calls to `f` by calls to `mf`

- includes ones in the body of f
- but not the one in mf

Problem 16-2

TEX

Pretty Paragraphs

words $w_1, w_2, \ldots, w_n$ in Courier

lengths $l_1, l_2, \ldots, l_n$

separated by spaces on lines of length M

If $w_i, \ldots, w_j$ on a line, then extra spaces

$pad(i,j) = M - (j-i) - \sum_{k=i}^{j} l_k$

Minimize $(pad(i,j))^3$ over all lines but the last

$pad(i,n) =$

Example

M = 14

```
this◊is◊an◊◊◊◊
example◊group◊
of◊words
```

pad(1,3) = 4 $(4)^3$ 64

pad(4,5) = 1 $(1)^3$ 1

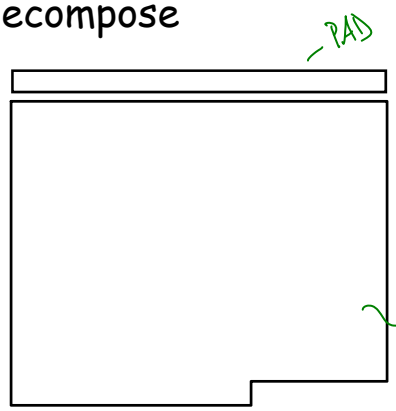
65

padding penalty

M = 14

Recursive Solution

Decompose



Calculating Cost

Let `BEST[i,n]` be the cost of the best layout of words $w_i \cdots w_n$ (on multiple lines)

```
BEST[i,n]
  min   (pad(i,k))   + BEST[k+1 , n ])
```

$1 \le k \le p$

How big is largest `p` we must consider?

$\lceil \frac{M}{2} \rceil$

Strategy

Compute

```
BEST[n,n]
BEST[n-1,n]
BEST[n-2,n]
  ...
BEST[1,n]
```

helps if `BEST[n+1,n]` = 0

How long does each one take?

- `k` ranges over $O(M)$
  - `BEST[k+1,n]` look up
  - `pad(i,k)` might seem like $O(M)$

Computing pad(i,j)

But notice

each takes constant time

```
pad(i,i)   = M - l_i
pad(i,i+1) = pad(i,i) - l_{i+1} - 1
pad(i,i+2) = pad(i,i+1) - l_{i+2} - 1
  ...
pad(i,p)   = pad(i,p-1) - l_p - 1
```

so time complexity is `O(n*M)`