Analysis of Algorithms

V. Adamchik

CSCI 570

Lecture 7

University of Southern California

Fall 2023

Dynamic Programming - 2

Reading: chapter 6

Concept of Dynamic Programming

Optimal substructure:

optimal solution to problem consists of optimal solutions to subproblems

Overlapping subproblems:

few subproblems in total, many recurring instances of each

Solve *bottom-up*, building a table of solved subproblems that are used to solve larger ones.

Static Optimal Binary Search Tree

Build a binary search tree which gives a minimum search cost, assuming we know the frequencies p_i with which data k_i is accessed. The tree cannot be modified after it has been constructed.

Want to build a binary search tree with minimum expected search cost:

$$\text{Expected Cost} = \sum_{i=1}^{n} p_i \,\text{depth}(k_i)$$

depth(root) = 1

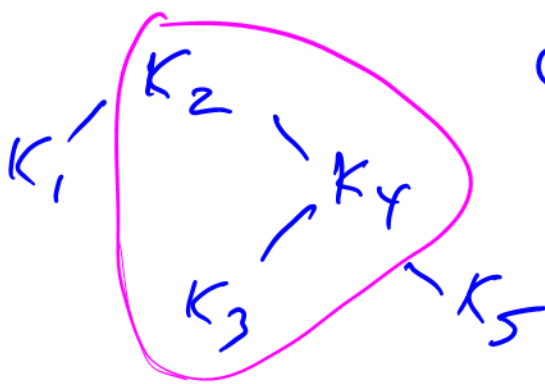
Example

Consider 5 items

$$k_1 < k_2 < k_3 < k_4 < k_5$$

and their search probabilities

$$p_1 = 0.25,\ p_2 = 0.2,\ p_3 = 0.05,\ p_4 = 0.2,\ p_5 = 0.3.$$

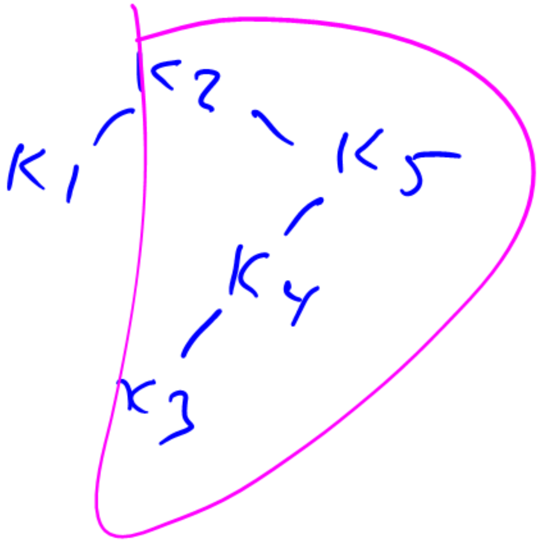


$$\begin{aligned}\text{Cost} = {} & 2 \times 0.25 + \\ & 1 \times 0.2 + \\ & 3 \times 0.05 + \\ & 2 \times 0.2 + \\ & 3 \times 0.3 \\ = {} & 2.15\end{aligned}$$

Another possibility

$k_1 < k_2 < k_3 < k_4 < k_5$

$p_1 = 0.25$, $p_2 = 0.2$, $p_3 = 0.05$, $p_4 = 0.2$, $p_5 = 0.3$



$$\text{Cost} = 2 \times 0.25 + 1 \times 0.2 + 2 \times 0.3 + 3 \times 0.2 + 4 \times 0.05 = 2.1$$

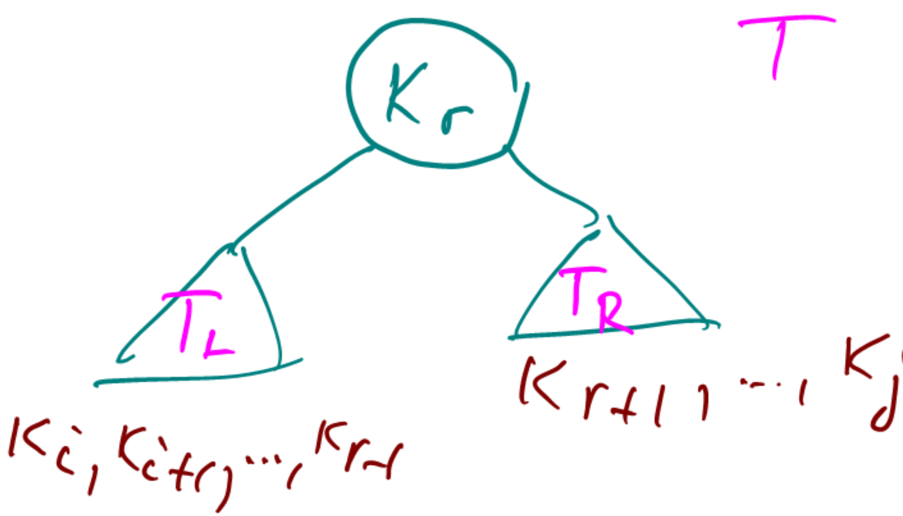
Optimal Substructure

Let OPT[i,j] be the min cost of a tree made of $K_i, K_{i+1}, \dots, K_{j-1}, K_j$.

$$K_i \le K_{i+1} \le \dots \le K_j$$

What is the root of OPT[i,j] subtree?

$$K_i \le K_r \le K_j$$



$$\mathrm{Cost}_T = OPT[i,j] =$$

$$1 \times P_r + \sum_{s=i}^{r-1} P_s \cdot \mathrm{depth}_T(K_s) + \sum_{s=r+1}^{j} P_s \cdot \mathrm{depth}_T(K_s)$$

$$\mathrm{depth}_T(K_s) = \mathrm{depth}_{T_L}(K_s) + 1$$

$$= \mathrm{depth}_{T_R}(K_s) + 1$$

$$\mathrm{Cost}_T = P_r + \sum P_s (1 + \mathrm{depth}_{T_L}(K_s))$$

$$+ \sum P_s (1 + \mathrm{depth}_{T_R}(K_s))$$

$OPT[i, r-1]$

$$= P_r + P_i + P_{i+1} + \dots + P_{r-1} + P_{r+1} + \dots + P_j +$$

$$+ \sum P_s \, \mathrm{depth}_{T_L}(K_s) + \sum P_s \, \mathrm{depth}_{T_R}(K_s)$$

Recurrence Relation

$$OPT[i,j] = p_i + \dots + p_j + \min_{i \le r \le j} \left[OPT[i, r-1] + OPT[r+1, j] \right]$$

$$OPT[i,i] = p_i$$

$$OPT[i,i-1] = 0$$

Filling up the table

array p = [p_1, p_2, …, p_n]
set OPT[i, i-1] = 0, for $1 \le i \le n$
set OPT[i, i] = p_i, for $1 \le i \le n$

for(k = 1; k < n; k++)
  for(i = 1; i <= n-k, i++)
    j = i + k;
    OPT[i, j] =p_i+…+p_j+ $\min_r$(OPT[i,r-1]+OPT[r+1,j]);
    $(i \le r \le j)$

$O(n)$ $O(1)$ $O(1)$

return OPT[1,n];

$\boxed{\text{Runtime Complexity-?}}$

$O(n^3)$

Example

n = 5 p_1 p_2 p_3 p_4 p_5

(prob,value)=(0.1,5), (0.3,6), (0.9,4), (0.3,3), (0.1,8)

		0	1	2	3	4	5
(0.1, 5)	1	0	0.1	? 0.5			?
(0.3, 6)	2		0	0.3			
(0.9, 4)	3			0	0.9		
(0.3, 3)	4				0	0.3	
(0.1, 8)	5					0	0.1

r=1: 0+0.3

r=2: 0.1+0

Exersize: OPT[1, 2]

$$OPT[1,2] = 0.1 + 0.3 + \min_{1 \le r \le 2} \left[OPT[1, r-1] + OPT[r+1, 2] \right]$$

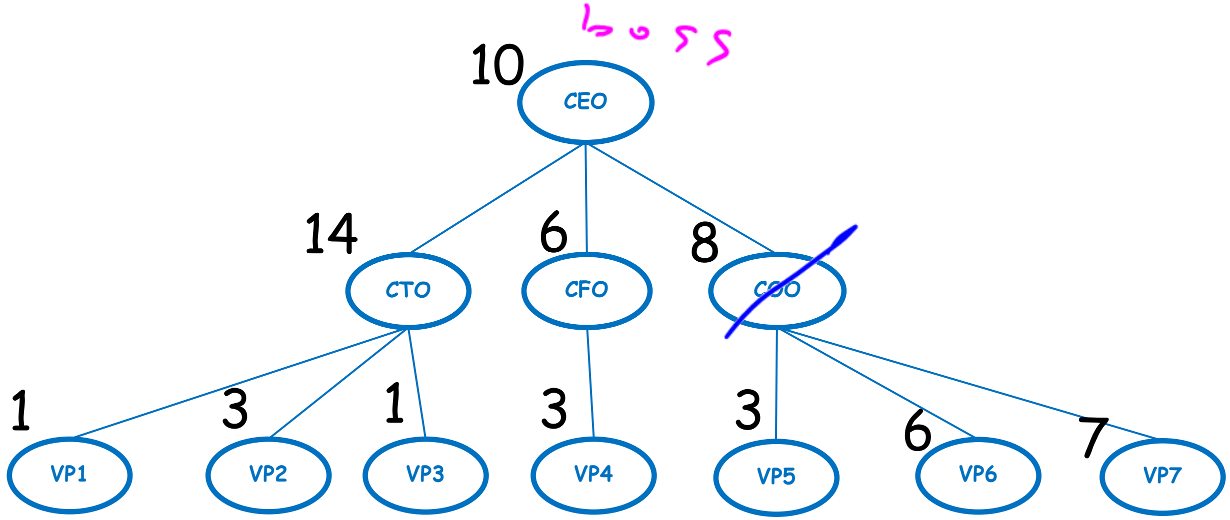
$$= 0.1 + 0.3 + 0.1 = 0.5$$

Discussion Problem 1

Break

Suppose you are organizing a company party. The corporation has a tree-like ranking structure; that is, the CEO is the root node of the hierarchy tree, and the CEO's immediate subordinates are the children of the root node, and so on in this fashion. To keep the party fun for all involved, you will not invite any employee whose immediate superior is invited. Each employee j has a value v_j (a positive integer), representing how enjoyable their presence would be at the party. Our goal is to determine which employees to invite, subject to these constraints, to maximize the total value of invitees.

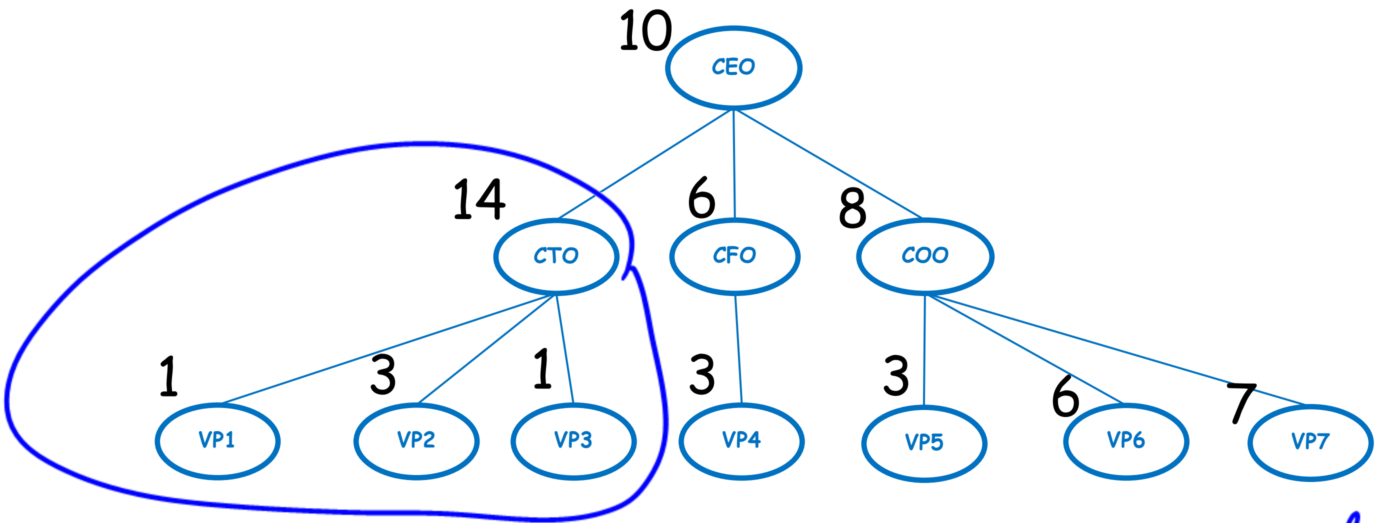
$0 \le j \le 4$



Shall we invite the boss?

① NO, $OPT = 36 = 14 + 6 + (3+6+7)$

② YES, $OPT = 34 = 10 + (1+3+1+\dots+7)$



Let OPT[r] be the max fun value of a subtree rooted at r.

Choices:

① r is selected: $OPT[r] = V_r + \sum_g OPT[g]$

② r is not : $OPT[r] = \sum_c OPT[c]$

g = grandchildren, c = children

Req. Eg.

$$OPT[r] = \underset{O(1)}{MAX}\left[v_r + \sum g, \quad \underset{O(n)}{\sum c} \right]$$

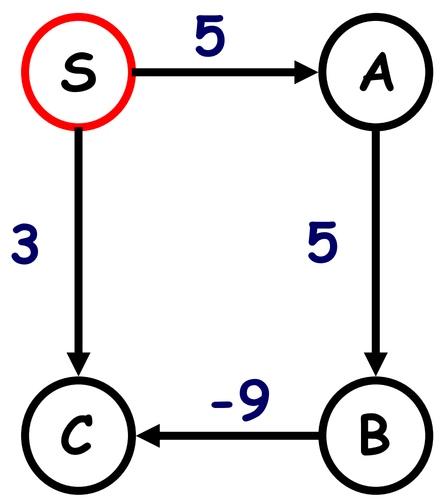
Base case:

$OPT[NULL] = 0$

Runtime: $O(n)$ yes

$O(n^2)$ – !!

The Shortest Path Problem



Dijkstra's greedy algorithm does not work on graphs with negative weights.

How can we use Dynamic Programming to find the shortest path? We need to somehow defined <u>ordered</u> subproblems, otherwise we may get an exponential runtime.

Intuition

Consider the path (with k edges)

$$v = w_0, w_1, \ldots, w_{k-1}, w_k = u.$$

To have an optimal substructure the following path (k-1 edges)

$$v = w_0, w_1, \ldots, w_{k-1}$$

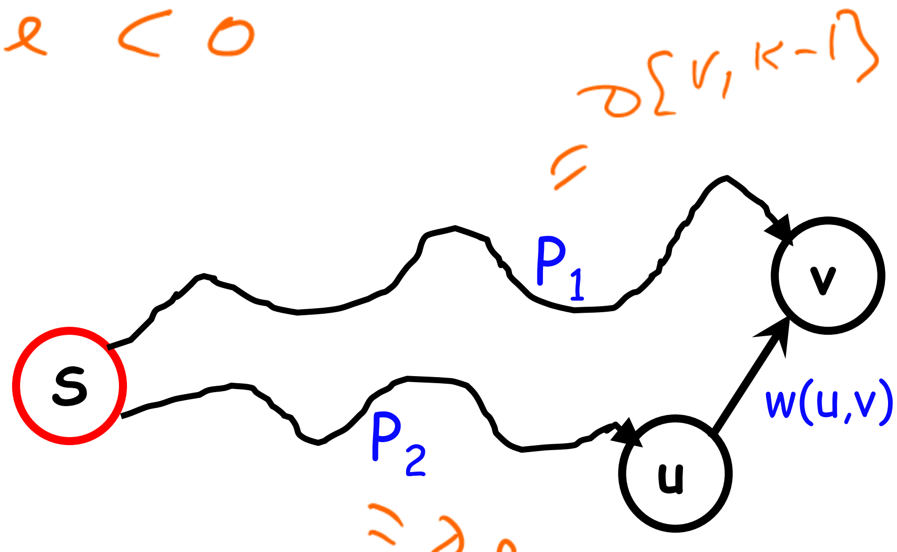
must be a shortest path to w_{k-1}.

Thus, we will be counting *the number of edges* in the shortest path. This is how we order subproblems.

D[v, k] denotes the length of the shortest path from s to v that uses at most k edges.

The Bellman-Ford Algorithm

w could be < 0



$$D[v,k] = \begin{cases} P_1 \\ P_2 + w(u,v) \end{cases}$$

$D[v,k]$ denotes the shortest path from s to v using at most k edges

Case 1. path uses at most $k-1$ edges

$$D[v,k] = D[v,k-1]$$

Case 2. o.w. let u be an adjacent vertex

$$D[v,k] = \min_u \left[D[u,k-1] + w(u,v) \right]$$

$$1 \le k \le V-1$$

Req. Eq.

$$D[v,k] = \underset{O(1)}{\mathrm{MIN}} \left[\begin{array}{l} \overset{O(1)}{D[v,k-1]}, \\ \underset{\substack{u \\ O(V)}}{\min} \left(\underset{O(1)}{D[u,k-1]} + \overset{O(1)}{w(u,v)} \right) \end{array} \right]$$

Base Cases:

$$D[v,0] = \infty$$

$$D[s,k] = 0$$

Implementation

D[v,k] denotes the length of the shortest path from s to v that uses at most k edges.

```
D[v,0] = INFINITY; v != s
D[s,k] = 0; for all k

for k=1 to V-1:
  for each v in V:
    for each edge (u,v) in E:
        D[v, k] = min(D[v,k-1] , w(u,v) + D [u,k-1])
```

V

V

E

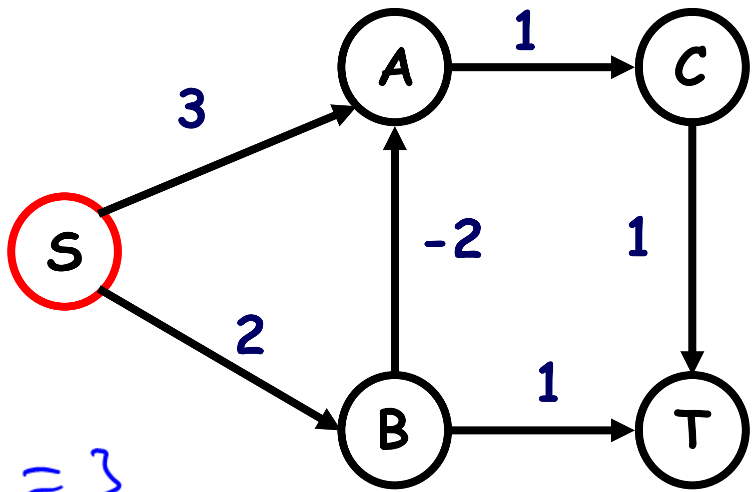
E

Runtime - ?

$O(V \cdot V \cdot E)$

$O(V \cdot E)$

Example



$k=1$: $d[A,1]=3$

$d[B,1]=2$

$k=2$: $d[A,2]=\min(3,\ 2-2)=0$

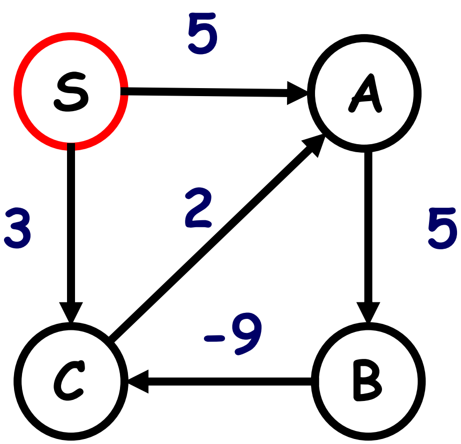
$d[C,2]=4$

$d[T,2]=3$

$k=3$: $d[C,3]=\underline{1}$

$k=4$: $d[T,4]=1+1=2$

How would you apply the Bellman-Ford algorithm to find out if a graph has a negative cycle?



C – A – B – C

$2 + 5 + (-9) = -2$

Break

Discussion Problem 2

There are *n* trading posts along a river numbered *n*, *n*–1..., 1. At any of the posts you can rent a canoe to be returned at any other post downstream. (It is impossible to paddle against the river). For each possible departure point *i* and each possible arrival point $j < i$, the cost of a rental is $C[i, j]$. However, it can happen that the cost of renting from *i* to *j* is higher than the total costs of a series of shorter rentals. In this case you can return the first canoe at some post *k* between *i* and *j* and continue your journey in a second (and, maybe, third, fourth . . .) canoe. There is no extra charge for changing canoes in this way. Give a dynamic programming algorithm to determine the minimum cost of a trip by canoe from each possible departure point *i* to each possible arrival point *j*.

$1 \le i \le n$

$j \le i$

$n, n-1, n-2, \dots, 3, 2, 1$

$i \to K \to j$

Let OPT[i,j] be the min rental cost going from i to j.

$$OPT[i,j] = \min_{j \le k < i} \left[C[i,k] + OPT[k,j] \right]$$

$$OPT[i,i] = 0$$

Runtime: $O(n^2 \times n) = O(n^3)$

Variant B.

$i = n, j = 1$

OPT[n, 1]

Solution

Use an 1-dimensional array

Let OPT[i] be the min rental cost from i to 1

$$OPT[i] = \min_{1 \le k \le i} \left[C[i,k] + OPT[k] \right]$$

$OPT[1] = 0$

Runtime $= O(n^2)$

Chain Matrix Multiplication

Given a sequences of matrices

$$M_1, M_2, \ldots, M_n,$$

determine the optimal order of multiplication that minimize the number of operations.

The matrix multiplication is associative. For example,

$$((AB)C)D = (A(BC))D = A((BC)D) = A(B(CD))$$

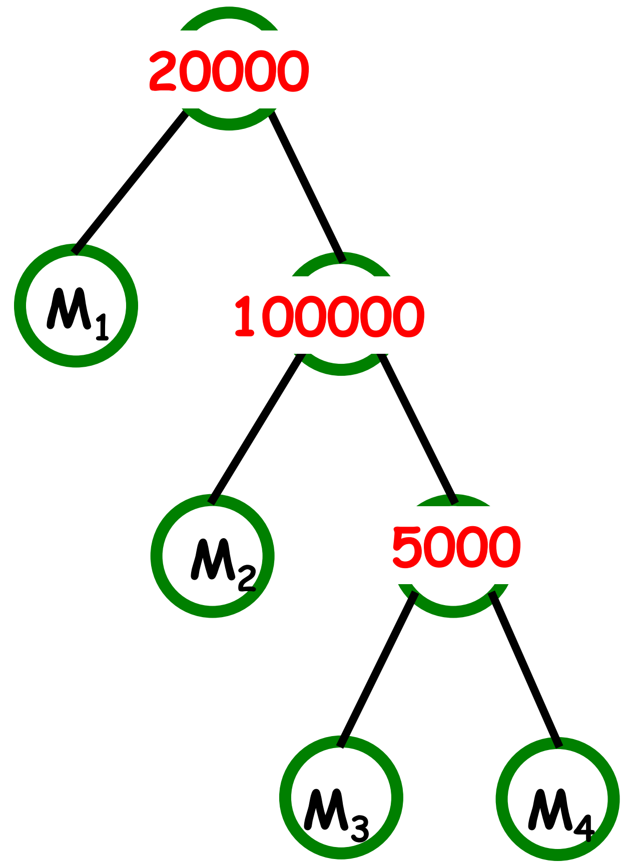
The matrix multiplication is not commutative.

$$A\,B \neq BA$$

Chain Matrix Multiplication

$M_1 = [10x20]$
$M_2 = [20x50]$
$M_3 = [50x1]$
$M_4 = [1x100]$

One order: $M_1*(M_2*(M_3*M_4))$



$M_3 M_4 = [50x100]$

$M_2 (M_3 M_4) = [20x100]$

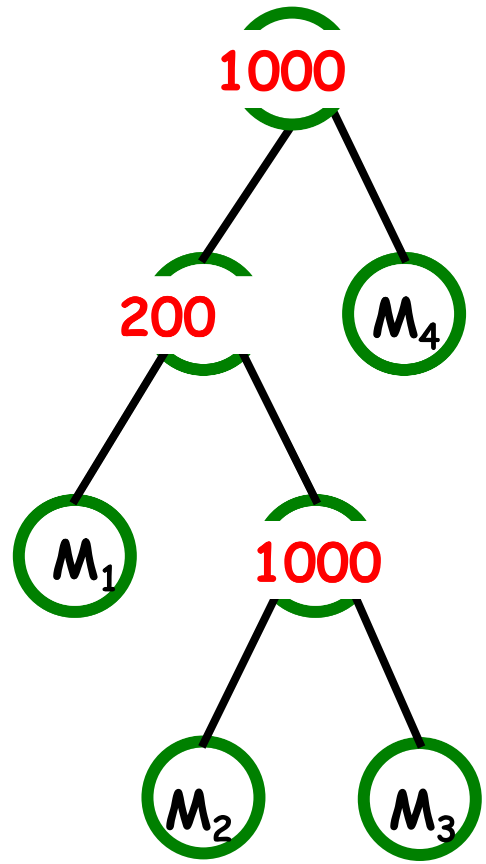
$M_1 (M_2 (M_3 M_4)) = [10x100]$

The total number of multiplications is 125000

Chain Matrix Multiplication

$M_1 = [10 \times 20]$
$M_2 = [20 \times 50]$
$M_3 = [50 \times 1]$
$M_4 = [1 \times 100]$

Another order: $(M_1 * (M_2 * M_3)) * M_4$



$M_2 M_3 = [20 \times 1]$

$M_1 (M_2 M_3) = [10 \times 1]$

$(M_1 (M_2 M_3)) M_4 = [10 \times 100]$

The total number of multiplications is 2200

Structure of an optimal solution

To find an optimal solution for 1..n matrices

$M_1, \ldots, M_n,$

we must be able to find an optimal solution for a lesser number of matrices, $1 \le i < j \le n$:

$M_i, \ldots, M_j$

Let Opt(i,j) be the cost of (i,j) subproblem:

Opt[i, j] = *min cost of* $M_i * M_{i+1} * \ldots * M_j$

Opt[i, i] = 0

Subproblems: $M_i * M_{i+1} * \ldots * M_j$

Question 1: how many subproblems are there? $O(n^2)$

Question 2: how do we compute subproblems ? !!!

We split each product into two parts:

$$\underline{(M_i * M_{i+1} * \ldots * M_k)} * (\underline{\underline{M_{k+1} * \ldots * M_j}}), \qquad i \leq k < j$$

but we don't know the index k a priori…

We will have to consider all possible choices for k.

$$M_i * M_{i+1} * \ldots * M_j =$$

$$(M_i * M_{i+1} * \ldots * M_k) * (M_{k+1} * \ldots * M_j)$$

To compute (i,j) subproblem we split at each k

1) compute (i,k) and (k+1,j) subproblems
2) combine them into one
3) choose the min split over all k

The total cost Opt[i,j] is given by

Opt[i, j] = Opt[i, k] + Opt[k+1, j] + comb_ step

Σp_s
in BST

Combining step

Calls to Opt[i, k] and Opt[k+1, j] will eventually produce two matrices of sizes $r_{i-1} \times r_k$ and $r_k \times r_j$:

$$(M_i * M_{i+1} * \ldots * M_k) * (M_{k+1} * \ldots * M_j)$$

$$\underset{r_{i-1} \times r_k}{} \qquad\qquad\qquad \underset{r_k \times r_j}{}$$

It takes $r_{i-1}\, r_k\, r_j$ multiplications to multiply two matrices. This is the cost of the combining step.

Recurrence Formula

$$Opt[i, j] = \underset{i \le k < j}{\mathrm{MIN}}(Opt[i, k] + Opt[k+1, j] + r_{i-1}\, r_k\, r_j)$$

$$Opt[i, i] = 0$$

where $1 \le i < j \le n$

The solution is $Opt[1,n]$

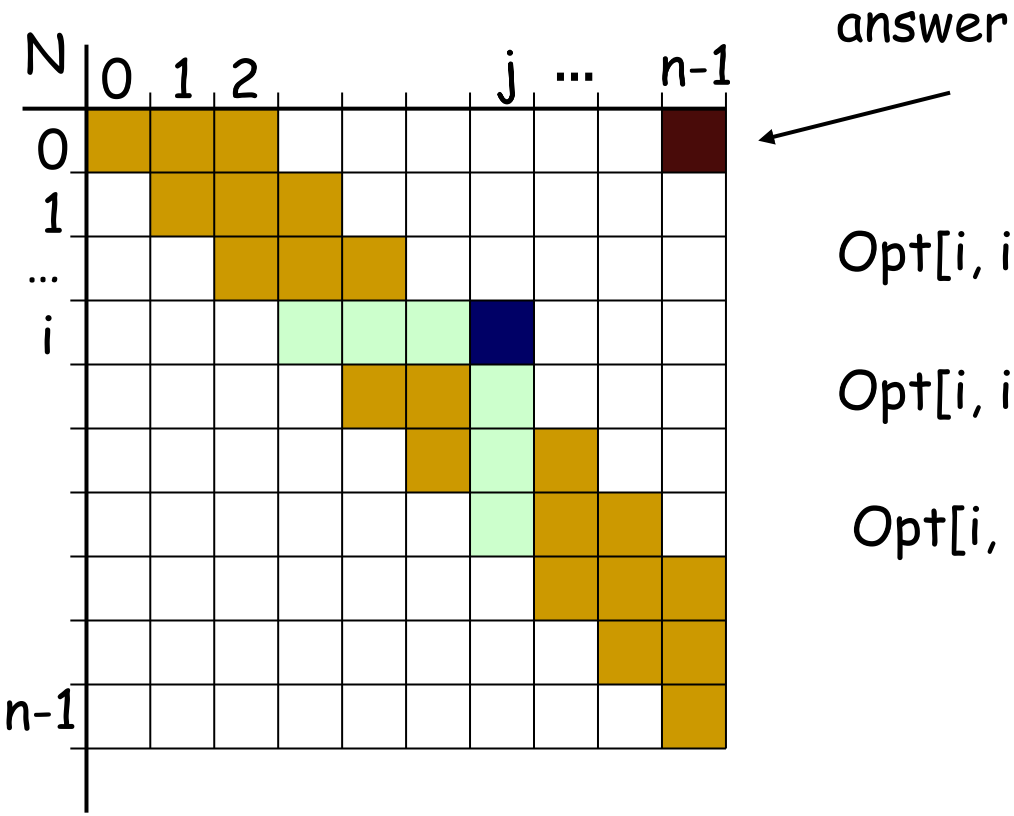
Filling up the table

$Opt[i, i] = 0,$ $i = 1, 2, \ldots, n$

$Opt[i, i+1] = r_{i-1}\, r_i\, r_{i+1},$ $i = 1, 2, \ldots, n-1$

$Opt[i, i+2] = \ldots.$ $i = 1, 2, \ldots, n-2$

Filling up the table



$Opt[i, i] = 0,$

$Opt[i, i+1] = r_{i-1}\, r_i\, r_{i+1}$

$Opt[i, i+2] = ...$

$Opt[i, j]= \min_k(Opt[i, k] + Opt[k+1, j]+ r_{i-1}\, r_k\, r_j\,)$