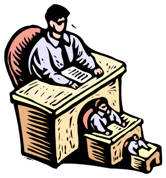
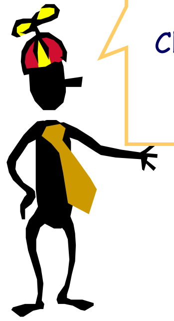
Algorithm Design and Analysis

Victor Adamchik — CS 15-451 Spring 2014

Lecture 5 — Jan 24, 2014 — Carnegie Mellon University

Dynamic programming





Chain matrix multiplication

Given a sequences of matrices, determine the order of multiplication that minimize the number of operations.

Chain matrix multiplication

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

Matrix multiplication is an associative but not a commutative operation. There are several choices:

$M_1*(M_2*(M_3*M_4))$

$(M_1*(M_2*M_3))*M_4$

Chain matrix multiplication

Multiplying an [m x n] matrix by an [n x p] matrix takes m*n*p multiplications.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} e & f & g \\ h & i & j \end{pmatrix} = \begin{pmatrix} ae+bh & af+bi & ag+bj \\ ce+dh & cf+di & cg+dj \end{pmatrix}$$

We are interested in multiplying more than 2 matrices, and we want to know the best order in which to perform multiplications.

Brute Force Approach

1) Do all possible multiplicative orders

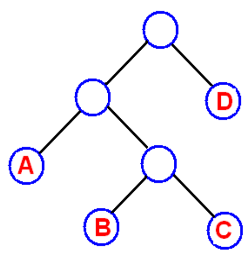
2) Choose the optimal

What is the complexity of this approach?

Chain matrix multiplication

Matrix multiplication is associative and corresponds to a full binary tree

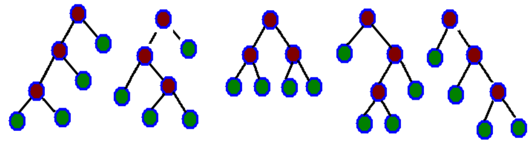
(A * (B * C)) * D



Chain matrix multiplication

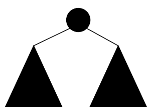
What is the number of full binary trees with n leaves?

…with four leaves



B(n) = # of full binary trees with n leaves

B(n) = B(1) B(n-1) + B(2) B(n-2) + … + B(n-1) B(1)
B(1) = 1



B(n) = C(n-1)

$$C_n = \frac{1}{n+1}\binom{2n}{n}, \; n = 0,1,\ldots$$

Catalan numbers

Brute Force Approach

This approach takes an exponential time…

$$C_n = \frac{1}{n+1}\binom{2n}{n}, \; n = 0,1,\ldots$$

$$n! \approx n^n$$

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} \approx \frac{(2n)^{2n}}{n^{2n}} = 4^n$$

Greedy Approach

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

There are several choices:

$M_1*(M_2*(M_3*M_4))$

$(M_1*(M_2*M_3))*M_4$

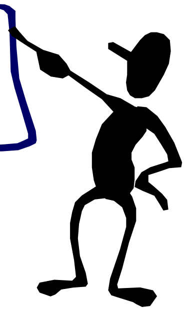
Greedy Approach

Repeatedly select the product that uses the fewest operations.

….not clear why this will lead to an optimal solution…

Dynamic Programming

The main question in DP is, what are the subproblems?



Matrix Multiplication $M_1 * M_2 * \dots * M_n$

How do we define subproblems?

$m(i, j)$ = *min cost of* $M_i * M_{i+1} * \dots * M_j$

$m(i, i) = 0$

$M_i * M_{i+1} * \dots * M_j$

We split that (i-j) product into two pieces

$(M_i * M_{i+1} * \dots * M_k) * (M_{k+1} * \dots * M_j), \quad i \le k < j$

The total cost $m(i,j)$ is given by

$m(i, k) + m(k+1, j)$ + combining step

$m(i,j) = \min_k(m(i,k) + m(k+1,j) + comb\_step)$

What is the complexity of the combining step?

Combining step

These two pieces will eventually produce two matrices

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

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

It takes $r_{i-1}\, r_k\, r_j$ multiplications to multiply two matrices.

Chain matrix multiplication

How would you fill out the table?

Filling up the table

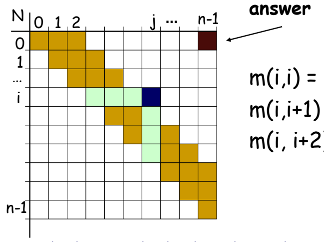
$m(i, j)$ = *min cost of* $M_i * M_{i+1} * \dots * M_j$

$m(i,i) = 0,$ $\quad$ $i = 1, 2, \dots, n$

$m(i,i+1) = r_{i-1}\, r_i\, r_{i+1},$ $\quad$ $i = 1, 2, \dots, n-1$

$m(i, i+2) = \dots.$ $\quad$ $i = 1, 2, \dots, n-2$

Filling up the table



$m(i,i) = 0,$

$m(i,i+1) = r_{i-1}\, r_i\, r_{i+1}$

$m(i, i+2) = \ldots.$

$m(i,j)=\min_k(m(i,k)+m(k+1,j)+comb\_step)$

Filling up the table

Set m(i,i) = 0 for all i.

```
for(s = 1; s < n; s++)
  for(i = 1; i <= n-s, i++)
    j = i + s;
    m(i,j)=min_k(m(i,k)+m(k+1,j)+comb_step);
         (i ≤ k < j)
return m(1,n);
```

Runtime complexity

$m(i,j)=\min_k(m(i,k)+m(k+1,j)+comb\_step)$

What is the complexity of this algorithm?

Table size – $O(n^2)$

Cost per entry – $O(n)$

Total – $O(n^3)$

Chain matrix multiplication

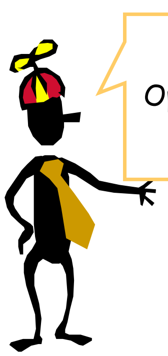
$M_1*M_2*M_3*M_4$

How would you recover the optimal set of parentheses?

We have to memorize the split marker indicating the best split: this is the value k.

Basic Steps of DP

1. Define subproblems.
2. Write the recurrence relation.
3. Prove that an algorithm is correct.
4. Compute its runtime complexity.



Optimal Binary Search Tree

Optimal Binary Search Trees

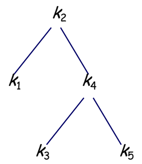
- Given sequence $k_1 < k_2 < \ldots < k_n$ of n sorted keys, with a search probability p_i for each key k_i.
- Want to build a binary search tree (BST) with minimum expected search cost.
- For key k_i, search cost = depth(k_i), where depth of the root is 1.
- Actual cost = # of items examined.

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

Note the difference between this problem and Huffman trees

Example

Consider 5 keys with these search probabilities:
$p_1 = 0.25$, $p_2 = 0.2$, $p_3 = 0.05$, $p_4 = 0.2$, $p_5 = 0.3$.

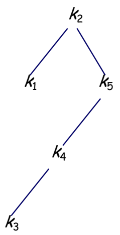


i	depth	depth(k_i)·p_i
1	2	0.5
2	1	0.2
3	3	0.15
4	2	0.4
5	3	0.9

Therefore, E[search cost] = 2.15.

Example

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



i	depth	depth(k_i)·p_i
1	2	0.5
2	1	0.2
3	4	0.2
4	3	0.6
5	2	0.6

Therefore, E[search cost] = 2.1

Example

Observations:
- Optimal BST may not have the smallest height.
- Optimal BST may not have highest-probability key at the root.

Naïve algorithm: build by exhaustive checking
- Construct each n-node BST.
- For each assign keys and compute expected cost.

How many trees? Described by Catalan numbers

tress = $O(4^n)$

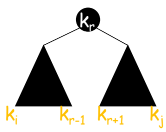
Step 1: Optimal Substructure

To find an optimal solution for

$k_1, \ldots, k_n$,

we must be able to find an optimal solution for

$k_i, \ldots, k_j$



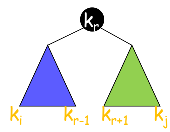
One of the keys in $k_i, \ldots, k_j$, must be the root
Left subtree of k_r contains $k_i, \ldots, k_{r-1}$.
Right subtree of k_r contains $k_r+1, \ldots, k_j$.

Step 2: Recurrence relation

Let $C_{i,j}$ be the optimal cost for $k_i, \ldots, k_j$

$$C_{i,j} = \min_{i \le r \le j} (C_{i,r-1} + C_{r+1,j}) + w_{i,j}$$

$$w_{i,j} = p_i + \ldots + p_j$$



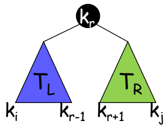
$$C_{i,i} = p_i$$

Step 3: Correctness

Let T be an optimal subtree with k_r be the root.

$$C_{i,j} = \min_{i \le r \le j}(C_{i,r-1} + C_{r+1,j}) + w_{i,j}$$

$$w_{i,j} = p_i + \ldots + p_j$$



To prove the above formula, we compute the tree cost directly

$$Cost(T) = 1 * p_r + \sum_{m=i}^{r-1} p_m depth_T(k_m) + \sum_{m=r+1}^{j} p_m depth_T(k_m)$$

Conclude the proof by changing

$depth_T$ -> $1+depth_{T_L}$ and $depth_T$ -> $1+depth_{T_R}$

Step 3: Correctness

$$Cost(T) = 1 * p_r + \sum_{m=i}^{r-1} p_m depth_T(k_m) + \sum_{m=r+1}^{j} p_m depth_T(k_m)$$

$$= p_r + \sum_{m=i}^{r-1} p_m(1 + depth_{T_L}(k_m)) +$$

$$\sum_{m=r+1}^{j} p_m(1 + depth_{T_R}(k_m))$$

$$= w_{i,j} + \sum_{m=i}^{r-1} p_m depth_{T_L}(k_m) + \sum_{m=r+1}^{j} p_m depth_{T_R}(k_m)$$

$$= w_{i,j} + Cost(T_L) + Cost(T_R)$$

Step 3: Correctness

Finally, we need to prove that

$C_{i,j} = OPT_{i,j}$

Case 1). $OPT_{i,j} \le C_{i,j}$. Trivial, just return a tree with k_r being the root.

Case 2). $C_{i,j} \le OPT_{i,j}$. Proof by induction

We computed in the previous slide that

$$C_{i,j} = w_{i,j} + C_{i,r-1} + C_{r+1,j} \overset{\text{by IH}}{\le} w_{i,j} + OPT_{i,r-1} + OPT_{r+1,j}$$

$$= OPT_{i,j}$$

Filling up the table

Compute w(i,j) = 0 for all $1 \le i \le j \le n$

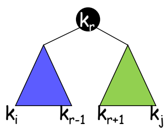
Set m(i,i) = p_i, for $1 \le i \le n$

```
for(k = 1; k < n; k++)
  for(i = 1; i <= n-k, i++)
    j = i + k;
    m(i,j)=w(i,j) + min_r(m(i,r-1)+m(r+1,j));
                    (i ≤ r ≤ j)
return m(1,n);
```

Step 4: Runtime Complexity

$$C_{i,j} = \min_{i \le r \le j}(C_{i,r-1} + C_{r+1,j}) + w_{i,j}$$

$$w_{i,j} = p_i + \ldots + p_j$$



with initial conditions

$C_{i,i} = p_i$ and $C_{i,j} = 0, \text{if } j < i$

Table size – $O(n^2)$

Cost per entry – $O(n)$

Total – $O(n^3)$