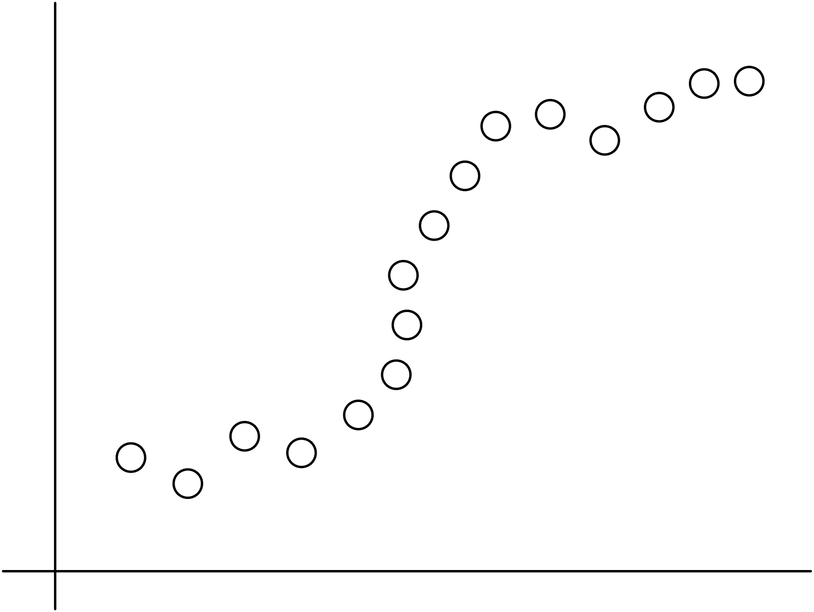
More Dynamic Programming Examples

Slides by Carl Kingsford

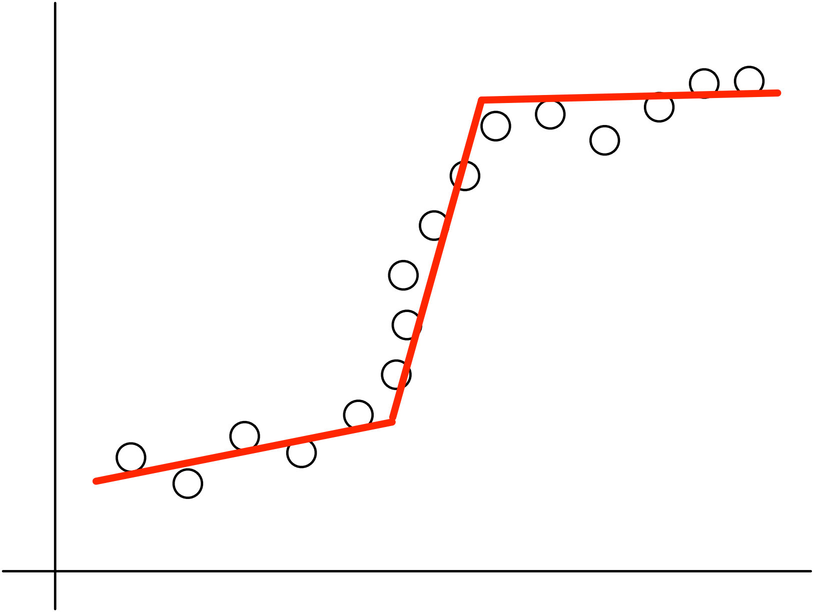
Apr. 1, 2013

AD 6.3, CLR 16.1

Segmented Least Squares



Segmented Least Squares



Segmented Least Squares Problem

Problem. *Given a sequence of points $p_1, \ldots, p_n$ sorted by their x-coordinate, find a partition $S_1, \ldots, S_k$ of the points to minimize:*

$$C \times k + \sum_i fit(S_i),$$

where C is a given constant, and $fit(S)$ is the best least-squares fit of a line ℓ to the set of points S.

- C is the penalty of introducing a new line.
- Note: k is **not** an input!
- $\text{fit}(S, \ell)$ can be computed analytically (next slide).

Least Squares Fit

The least squares fit is:

$$\text{fit}(S) = \sum_{p \in S} (y_p - a x_p - b)^2$$

where the line $y_p = a x_p + b$ is given by:

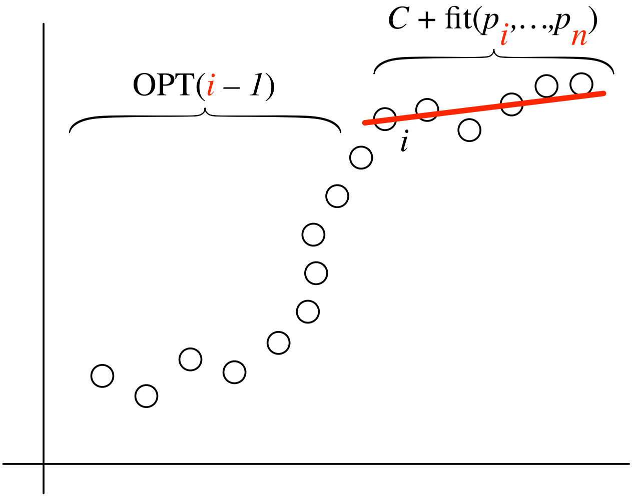
$$a = \frac{|S| \sum_p x_p y_p - \left(\sum_p x_p\right)\left(\sum_p y_p\right)}{|S| \sum_p x_p^2 - \left(\sum_p x_p\right)^2}$$

$$b = \frac{\sum_p y_p - a \sum_p x_p}{|S|}$$

So once you choose $S_1, \ldots, S_k$, the best lines can be computed directly. So, how do we choose this partition?

Subproblems

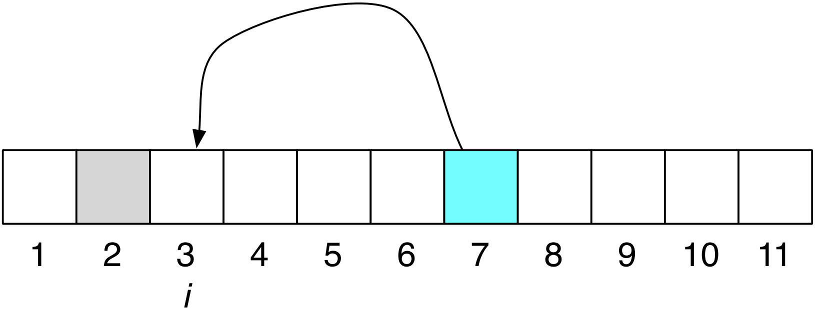
Suppose you knew the last segment was from $p_i, \ldots, p_n$:



Recurrence

$$OPT(j) = \max_{1 \le i \le j} \{C + \text{fit}(p_i, \dots, p_j) + OPT(i-1)\}$$

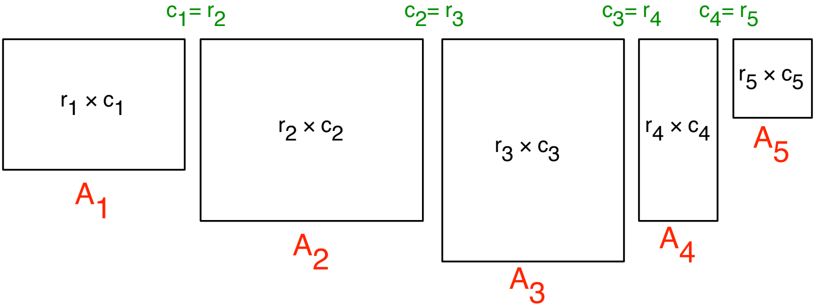
OPT "matrix":



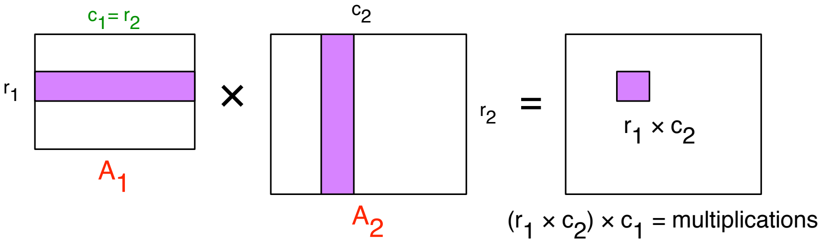
Matrix-Chain Multiplication

Matrix-Chain Multiplication

A series of matrices $A_1, \ldots, A_n$ need to be multiplied:



Recall pairwise multiplication



Let **mul**(A_1, A_2) be the number of matrix multiplications you need to multiply matrices A_1 and A_2.

Associativity of matrix multiplication

Matrix multiplication is associative, so we can add parentheses any way we want:

$$A_1 A_2 A_3 A_4 A_5 A_6$$

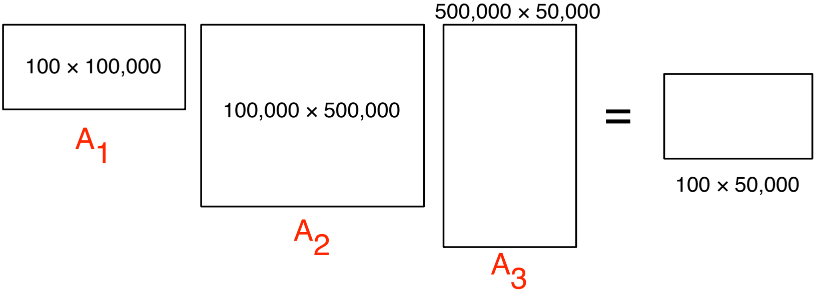
For example, all of these give the same final matrix:

- $(A_1 A_2)((A_3 A_4)(A_5 A_6))$
- $(A_1(A_2 A_3))((A_4 A_5)A_6)$
- $((A_1 A_2)(A_3 A_4))(A_5 A_6)$

The parentheses give the order to do the multiplications.

But different orders can give very different numbers of scalar multiplications.

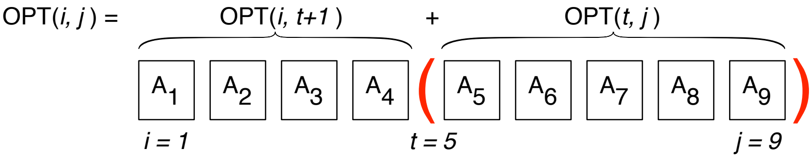
Examples of different costs



Possible solutions:

- $((A_1 A_2)\, A_3) = 100 \times 100,000 \times 500,000 + 100 \times 500,000 \times 50,000 = 7,500,000,000,000$
- $(A_1\, (A_2 A_3)) = 100,000 \times 500,000 \times 50,000 + 100 \times 100,000 \times 50,000 = 2,500,500,000,000,000$

Subproblems



Leads to the recurrence:

$$OPT(i,j) = \max_{1 \leq t \leq j} \{OPT(i, t-1) + OPT(t,j) + r_1 c_{t-1} c_j\}$$

Base cases: $OPT(i, i+1) = \mathbf{mul}(A_i, A_{i+1}) = r_i c_i c_{i+1}$

Optimal Binary Search Trees

Designing an optimal binary search tree

Often, you have a large data set that is fixed at the start of your computation.

You'll make many lookups into this data to find items associated with keys, but the keys will never change.

Some data you know will be accessed frequently, some rarely.

Problem (Optimal Binary Search Trees). *We are given sorted keys $k_1, \ldots, k_n$ and the probabilities $p_1, \ldots, p_n$ that key i will be accessed at any point in time. Construct a binary search tree T that minimizes:*

$$C(T) = \sum_{i=1}^{n} p_i \left(Depth(T, k_i) + 1 \right)$$

Expected Search Cost

The expression:

$$C(T) = \sum_{i=1}^{n} p_i \,(\text{Depth}(T, k_i) + 1)$$

is the expected cost of a "find" operation in the tree.

$\text{Depth}(T, k)$ is the distance from the root of key k. For example, if k is the root, then $\text{Depth}(T, k) = 0$.

Subtrees of the optimal are optimal

Let $\mathrm{D}(T, k) = \mathrm{Depth}(T, k)$ for brevity.

Let T be an optimal tree that has root k_r.

$$\begin{aligned} C(T) &= p_r + \sum_{a=1}^{r-1} p_a \left(\mathrm{D}(T_r, k_a) + 1\right) + \sum_{a=r+1}^{n} p_a \left(\mathrm{D}(T_r, k_a) + 1\right) \\ &= \sum_{a=1}^{r-1} p_a + \sum_{a=1}^{r-1} p_a \mathrm{D}(T_{\mathrm{left}}, k_a) + \sum_{a=r+1}^{n} p_a + \sum_{a=r+1}^{n} p_a \mathrm{D}(T_{\mathrm{right}}, k_a) \\ &= \sum_{a=1}^{n} p_a + C(T_{\mathrm{left}}) + C(T_{\mathrm{right}}) \end{aligned}$$

Recurrence

$C[i,j] :=$ "cost of an optimal binary search tree on keys $k_i, \ldots, k_j$."

$$C[i,j] = \begin{cases} 0 & \text{if } j < i \text{ (tree is empty)} \\ p_i & \text{if } i = j \text{ (tree is single node)} \\ \left(\sum_{a=i}^{j} p_a\right) + \min_r \{C[i, r-1] + C[r+1, j]\} \end{cases}$$

We're looking for $C[1, n]$.

Can fill in $C[i,j]$ matrix in order of increasing $j - i$.

Summary

Three dynamic programming algorithms for three very different problems:

- Segmented least squares
- Matrix-chain multiplication
- Constructing optimal least squares

The dynamic programming algorithms are all different, but share a very similar framework.

Illustrates the power of the dynamic programming technique.