Dynamic Programming

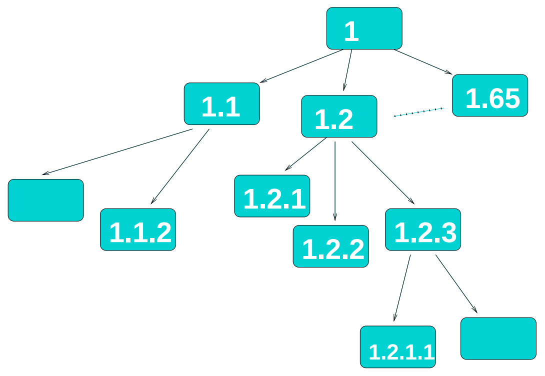
Dynamic programming, like the *divide-and-conquer*[a] method, solves problems by combining the solutions to subproblems.

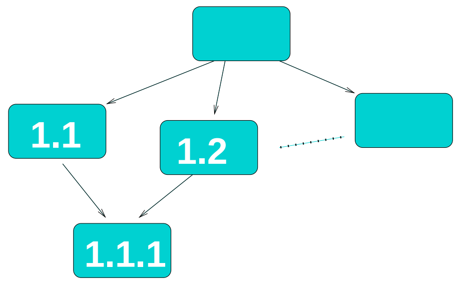
Richard Bellman (1920-1984) gave the name "dynamic programming" to the technique discussed here.

> In the first place I was interested in planning, decision making, in thinking. But planning i s not a good word for various reasons. I decided therefore to use the word "programming". I wanted to get across the idea that this was dynamic, this was multistage, this was time-varying – I thought, let's kill two birds with one stone.

[a] e.g. quicksort

Divide-and-conquer algorithms partition the problem into independent subproblems, solve the subproblems recursively, and then combine their solutions to solve the original problem.





If two subproblems share subsubproblems then a divide-and-conquer algorithm does more work than necessary, repeatedly solving the common subsubproblems.

In contrast, dynamic programming is applicable (and often advisable) when the subproblems are not independent, that is, when subproblems share subsubproblems. A dynamic programming algorithm solves every subproblem just once and then saves its answer in a table, thereby avoiding the work of recomputing the answer every time the subproblem is encountered.

Dynamic programming is typically applied to optimisation problems.

Cookbook recipe for dynamic programming

The development of a dynamic programming algorithm can be broken into a sequence of four steps.

1. Characterize the structure of an optimal solution.
2. Recursively define the value of an optimal solution.
3. Compute the value of an optimal solution in a *bottom-up*[a] fashion.
4. Construct an optimal solution from computed information.

Steps 1–3 form the basis of a dynamic programming solution to a problem. Step 4 can be omitted if only the value of an optimal solution is required. When we do perform step 4, we sometimes maintain additional information during the computation of step 3 to “ease” the construction of an optimal solution.

[a] I.e. solving simplest subproblems first.

Matrix-chain multiplication

We are given a sequence (chain) of n matrices and we wish to compute their product. For example, if $n = 3$

$$\underbrace{\begin{pmatrix} 1 & 2 \\ -1 & 0 \\ -3 & -1 \\ 4 & -2 \end{pmatrix}}_{A_1} \times \underbrace{\begin{pmatrix} 1 & 1 & 1 \\ 0 & -5 & 3 \end{pmatrix}}_{A_2} \times \underbrace{\begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 2 & 3 \end{pmatrix}}_{A_3} = \underbrace{\begin{pmatrix} 24 & 11 \\ -2 & -3 \\ -17 & -13 \\ -14 & 4 \end{pmatrix}}_{B}$$

Exercises

1. How do we multiply two matrices together? ... You tell me!
2. How many multiplications do we need to do $(A_1 \times A_2) \times A_3$?
3. How many multiplications do we need to do $A_1 \times (A_2 \times A_3)$?
4. Given A_i an $n_i \times n_{i+1}$ matrix, for $i \in \{1, \ldots, N\}$, what bounds can you get on the maximum and minimum number of scalar multiplications which may possibly be needed as a function of the smallest and largest of the n_i’s?

To multiply two matrices (assuming for simplicity that the number of columns of the first one matches the number of rows of the second one) we use the well-known algorithm

```
MATRIX-MULTIPLY ($A$, $B$)
    for $i \leftarrow 1$ to rows($A$)
        for $j \leftarrow 1$ to columns($B$)
            $C[i, j] \leftarrow 0$
            for $k \leftarrow 1$ to columns($A$)
                $C[i, j] \leftarrow C[i, j] + A[i, k] \cdot B[k, j]$
```

but it is interesting to note that the running time of our algorithm heavily depends on the order in which we multiply the matrices.

Problem definition

Given a sequence of matrices $A_1, \ldots, A_N$ such that A_i has dimensions $n_i \times n_{i+1}$, find the way to compute $A_1 \times \ldots \times A_n$ that minimises the number of scalar multiplications.

The problem is an optimisation one. The set of instances coincides with sequences of integers $n_1, n_2, \ldots, n_{N+1}$ for some fixed integer N. A solution to the given problem is an ordering of the $N-1$ multiplications. The cost of an ordering is the number of scalar multiplications performed and the problem seeks a minimum cost ordering.

Next we present a dynamic programming heuristic for solving this problem. We follow the four step recipe highlighted at the beginning of the chapter.

Structure of optimal sequence of multiplications

Let us denote $A_i \times \ldots \times A_j$ by $A_{i..j}$.

An optimal ordering of $A_1 \times \ldots \times A_N$ splits the product at A_k:

$$A_1 \times \ldots \times A_N = A_{1..k} \times A_{k+1..N}$$

We have

$$\begin{aligned}&\text{cost}(A_{1..N}, k) = \\ &\qquad \text{cost}(A_{1..k}, j_1) + \text{cost}(A_{k+1..N}, j_2) + \text{mult}(A_{1..k} \times A_{k+1..N})\end{aligned}$$

The cost function depends on the input and on the output[a]

$\text{mult}(A, B)$ denotes[b] the number of multiplications in $A \times B$.

Claim. j_1 and j_2 correspond to two minimum cost orderings of the chain multiplications $A_{1..k}$ and $A_{k+1..N}$.

Else, if the ordering for $A_{1..k}$ was suboptimal we could have chosen an optimal one and reduced $\text{cost}(A_{1..N}, k)$.

Thus, an optimal solution to an instance of the matrix-chain multiplication problem contains within it optimal solutions to subproblem instances!

[a] Actually on a very small part of the output.

[b] This number actually depends only on the dimensions of the two matrices, not on the values of the matrices.

Recursive decomposition

The second step of the dynamic programming paradigm is to define the value of an optimal solution recursively in terms of the optimal solutions to subproblems.

We pick as our subproblems the problems of determining the minimum cost of an ordering of $A_i \times \ldots \times A_j$ for $1 \le i \le j \le N$.

Let $m[i, j]$ be the minimum number of scalar multiplications needed to compute the matrix $A_{i..j}$.

The cost of a cheapest way to compute $A_{1..N}$ should be $m[1, N]$.

Definition of $m[i, j]$

- If $i = j$, the chain consists of a single matrix $A_{i..i} = A_i$, no scalar multiplication is needed and therefore $m[i, i] = 0$.
- If $i < j$ then $m[i, j]$ can be computed by taking advantage of the structure of an optimal solution, as described in the previous section. Therefore, the optimal cost ordering of $A_{i..j}$ is obtained multiplying $A_{i..k}$ and $A_{k+1..j}$ then we can define

  $$m[i, j] = m[i, k] + m[k + 1, j] + \text{mult}(A_{i..k}, A_{k+1..j}).$$

  Notice that $A_{i..k}$ is a $n_i \times n_{k+1}$ matrix and $A_{k+1..j}$ is an $n_{k+1} \times n_{j+1}$ matrix. Therefore multiplying them takes $n_{k+1} \cdot n_i \cdot n_{j+1}$ multiplications. Hence

  $$m[i, j] = m[i, k] + m[k + 1, j] + n_{k+1} \cdot n_i \cdot n_{j+1}.$$

  ... but, alas! We do not know k!!!

No problem, there can only be $j - i$ possible values for k, we can try all of them. Formally we thus have

$$m[i, j] = \begin{cases} 0 & i = j \\ \min_{i \le k < j} \{m[i, k] + m[k + 1, j] + n_{k+1} n_i n_{j+1}\} & i < j \end{cases}$$

Computing the optimal costs

Let's write together a simple recursive program that, given the sequence $n_1, n_2, \ldots, n_{N+1}$, computes $m[1, N]$.

How long does this program take?

Key Observation!

$\boxed{\text{There are relatively few subproblems: one for each choice of } i \text{ and } j \text{ (that's } \binom{N}{2} + N \text{ in total).}}$

Instead of computing the solution to the recurrence recursively (top-down), we perform the third step of the dynamic programming paradigm and compute the optimal costs of the subproblems usign a bottom-up approach.

The following pseudocode assumes the matrix A_i has dimensions $n_i \times n_{i+1}$ for $i \in \{1, \ldots, N\}$. The input sequence is $n_1, n_2, \ldots, n_{N+1}$.

```
MATRIX-CHAIN-ORDER ($n_1, n_2, \ldots, n_{N+1}, N$)
    // First, fill all the elements in the diagonal with zero
    for $i \leftarrow 1$ to $N$      $m[i,i] \leftarrow 0$
    // Next, fill all elements at distance $l$ from the diagonal
    for $l \leftarrow 1$ to $N-1$
        for $i \leftarrow 1$ to $N-l$
            $j \leftarrow i+l$
            define $m[i,j]$ as the minimum of
                $m[i,k] + m[k+1,j] + n_i n_{k+1} n_{j+1}$
            for $i \leq k < j$.
```

Appendix: Linear Algebra

A *matrix* is a rectangular bi-dimensional array. Here is an example

$$\begin{pmatrix} 12 & 1 & -5 & 0 \\ 6 & 11 & 5 & 9 \\ 2 & -8 & -4 & -3 \end{pmatrix}.$$

This could be declared in Java as follows

```
int [][] A = new int[3][4];
```

Input. Applications.

Transpose

$$\begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1m} \\ a_{21} & a_{22} & \ldots & a_{2m} \\ \ldots & & & \\ a_{n1} & a_{n2} & \ldots & a_{nm} \end{pmatrix} \text{ is the matrix } \begin{pmatrix} a_{11} & a_{21} & \ldots & a_{n1} \\ a_{12} & a_{22} & \ldots & a_{n2} \\ \ldots & & & \\ a_{1m} & a_{2m} & \ldots & a_{nm} \end{pmatrix}$$

Scalar Product

$$\begin{pmatrix} \lambda a_{11} & \lambda a_{12} & \ldots & \lambda a_{1m} \\ \lambda a_{21} & \lambda a_{22} & \ldots & \lambda a_{2m} \\ \ldots & & & \ldots \\ \lambda a_{n1} & \lambda a_{n2} & \ldots & \lambda a_{nm} \end{pmatrix}$$

Addition

$$\begin{pmatrix} 31.2 & 2.6 & -13 & 0 \\ 15.6 & 28.6 & 33.8 & 23.4 \\ 5.2 & -20.8 & -10.4 & -7.8 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 2 & 3 & 0 \end{pmatrix}$$

is the matrix

$$\begin{pmatrix} 32.2 & 2.6 & -13 & 0 \\ 16.6 & 30.6 & 33.8 & 23.4 \\ 6.2 & -18.8 & -7.4 & -7.8 \end{pmatrix}$$

Matrix Multiplication

The result of multiplying the matrices:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \\ b_{41} & b_{42} & b_{43} \end{pmatrix}$$

is the matrix

$$\begin{matrix} a_{11} \times b_{11} + a_{12} \times b_{21} + a_{13} \times b_{31} + a_{14} \times b_{41} & a_{11} \times b_{12} + a_{12} \times b_{22} + a_{13} \times b_{32} + a_{14} \times b_{42} & a_{11} \times b_{13} + a_{12} \times b_{2} \\ a_{21} \times b_{11} + a_{22} \times b_{21} + a_{23} \times b_{31} + a_{24} \times b_{41} & a_{21} \times b_{12} + a_{22} \times b_{22} + a_{23} \times b_{32} + a_{24} \times b_{42} & a_{21} \times b_{13} + a_{22} \times b_{2} \\ a_{31} \times b_{11} + a_{32} \times b_{21} + a_{33} \times b_{31} + a_{34} \times b_{41} & a_{31} \times b_{12} + a_{32} \times b_{22} + a_{33} \times b_{32} + a_{34} \times b_{42} & a_{31} \times b_{13} + a_{32} \times b_{2} \end{matrix}$$