# 算法设计与分析

## 动态规划 (二)

# 目录

# 矩阵链乘法

动机：假设我们需要计算多个矩阵的乘积。这将涉及迭代地每次将两个矩阵相乘。

- 矩阵乘法不满足交换律（一般情况下 $A \times B \neq B \times A$），但满足结合律：

$$A \times (B \times C) = (A \times B) \times C$$

- 我们可以用许多不同的方式计算矩阵乘积，取决于如何加括号。

某些方式是否比其他方式更好?

$C_{ik} = A_{ij} \times B_{jk}$ 的复杂度

- $C$ 中的每个元素需要 $j$ 次乘法，共 $ik$ 个元素 $\Rightarrow$ 总复杂度 $\Theta(ijk)$

# 示例

假设我们要计算四个矩阵的乘积 $A \times B \times C \times D$，它们的维度分别为 $50 \times 20$、$20 \times 1$、$1 \times 10$ 和 $10 \times 100$。

| 加括号方式 | 计算 | 代价 |
|---|---|---|
| $A \times ((B \times C) \times D)$ | $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$ | 120,200 |
| $(A \times (B \times C)) \times D$ | $20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$ | 60,200 |
| $(A \times B) \times (C \times D)$ | $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$ | 7,000 |

# 示例

假设我们要计算四个矩阵的乘积 $A \times B \times C \times D$，它们的维度分别为 $50 \times 20$、$20 \times 1$、$1 \times 10$ 和 $10 \times 100$。

| 加括号方式 | 计算 | 代价 |
|---|---|---|
| $A \times ((B \times C) \times D)$ | $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$ | 120,200 |
| $(A \times (B \times C)) \times D$ | $20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$ | 60,200 |
| $(A \times B) \times (C \times D)$ | $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$ | 7,000 |

乘法顺序对最终复杂度有很大影响。

# 示例

假设我们要计算四个矩阵的乘积 $A \times B \times C \times D$，它们的维度分别为 $50 \times 20$、$20 \times 1$、$1 \times 10$ 和 $10 \times 100$。

| 加括号方式 | 计算 | 代价 |
|---|---|---|
| $A \times ((B \times C) \times D)$ | $20 \cdot 1 \cdot 10 + 20 \cdot 10 \cdot 100 + 50 \cdot 20 \cdot 100$ | 120,200 |
| $(A \times (B \times C)) \times D$ | $20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 + 50 \cdot 10 \cdot 100$ | 60,200 |
| $(A \times B) \times (C \times D)$ | $50 \cdot 20 \cdot 1 + 1 \cdot 10 \cdot 100 + 50 \cdot 1 \cdot 100$ | 7,000 |

乘法顺序对最终复杂度有很大影响。自然的贪心策略——
总是执行当前代价最小的矩阵乘法——不一定能得到最优解

- 参见第二种加括号方式作为反例

# 暴力算法

问：对于 $A_1 A_2 \ldots A_n$，有多少种不同的加括号方式?

# 暴力算法

问：对于 $A_1 A_2 \dots A_n$，有多少种不同的加括号方式？观察：一种特定的加括号方式可以自然地用一棵满二叉树表示

- 叶节点：单个矩阵
- 根节点：最终乘积
- 内部节点：中间乘积

[见原文第 8 页图示]

$$((A \times B) \times C) \times D \qquad\qquad A \times ((B \times C) \times D)$$

# 估计可能顺序的数量

可能的顺序数对应于具有 $n$ 个叶子的各种满二叉树的数量。
设 $C(n)$ 为具有 $n+1$ 个叶子（或等价地，共 $n$ 个内部节点）的满二叉树的数量：

[见原文第 9 页图示：$C(0)$ 和 $C(1)$ 的树]

$$C(0) = 1, C(1) = 1, C(2) = C(0)C(1) + C(1)C(0)$$
$$C(3) = C(0)C(2) + C(1)C(1) + C(2)C(0)$$
$$C_n = \sum_{i=0}^{n-1} C_i C_{n-1-i} = \frac{1}{n+1}\binom{2n}{n}$$

上述公式是卷积形式，可以通过生成函数计算。

- 结果是Catalan 数，关于 $n$ 是指数级的

# Catalan 数

Catalan 数（以比利时数学家 Eugène Charles Catalan 命名）。

- 最早由 Euler 在计算将 $n$ 边凸多边形划分为 $(n-2)$ 个三角形的不同方式时发现。

[见原文第 10 页图示：多边形三角剖分]

$$
\begin{aligned}
C(n) &= \Omega\left(\frac{1}{n+1}\frac{(2n)!}{n!n!}\right) \quad \text{// Stirling 公式} \\
&= \Omega\left(\frac{1}{n+1}\frac{\sqrt{2\pi 2n}\left(\frac{2n}{e}\right)^{2n}}{\sqrt{2\pi 2n}\left(\frac{n}{e}\right)^{n}\sqrt{2\pi 2n}\left(\frac{n}{e}\right)^{n}}\right) = \Omega(4^n/(n^{3/2}\sqrt{\pi}))
\end{aligned}
$$

# 暴力算法

Catalan 数出现在各种计数问题中（通常涉及递归定义的对象）

- 加括号方式的数量
- 满二叉树的数量
- 单调格路径的数量

由于 Catalan 数关于 $n$ 是指数级的 $\rightsquigarrow$ 我们当然不能尝试每一棵树，因此排除暴力方法。

我们转向动态规划。

# 动态规划

与二叉树的对应关系给出启示：要使一棵树最优，其子树也必须最优 $\Rightarrow$ 满足最优子结构（具有某种局部性）$\rightsquigarrow$ 不必从头尝试每一棵树

- 子问题对应于子树：形如 $A_i \times A_{i+1} \times \cdots A_j$ 的乘积

优化函数：

$$C(i,j) = \text{计算} A_i \times A_{i+1} \times \cdots A_j \text{ 的最小代价}$$

对应的维度是 $m_{i-1}, m_i, \ldots, m_j$

迭代关系：

$$C(i,j) = \begin{cases} 0 & i = j \\ \min_{i \le k < j}\{C(i,k) + C(k+1,j) + m_{i-1}m_k m_j\} & i < j \end{cases}$$

| $A_i$ | $\ldots$ | $A_k$ | $A_{k+1}$ | $\ldots$ | $A_j$ |
|---|---|---|---|---|---|
| $m_{i-1} \times m_k$ | | | $m_k \times m_j$ | | |

# 一些说明

## 动态规划的关键点

- 定义子问题
- 找到子问题之间的迭代最优子结构
- 按正确的顺序计算子问题

有时子问题之间的关系可能会产生误导。应该用正确的方式来解释和计算，即迭代方式。

# 递归方法（低效）

**Algorithm 1** MatrixChain$(C, i, j)$ // 子问题 $[i, j]$

1: $C(i, i) = 0,\ C(i, j) \leftarrow \infty$
2: $s(i, j) \leftarrow \perp$ // 记录分割位置
3: **for** $k \leftarrow i$ **to** $j - 1$ **do**
4: $\quad t \leftarrow \text{MatrixChain}(C, i, k) + \text{MatrixChain}(C, k + 1, j) + m_{i-1} m_k m_j$
5: $\quad$ **if** $t < C(i, j)$ **then**
6: $\quad\quad C(i, j) \leftarrow t$ // 找到更好的解
7: $\quad\quad s(i, j) \leftarrow k$
8: $\quad$ **end if**
9: **end for**
10: **return** $C(i, j)$

# 复杂度分析

递推关系为：

$$T(n) = \begin{cases} O(1) & n = 1 \\ \sum_{k=1}^{n-1}(T(k) + T(n-k) + O(1)) & n > 1 \end{cases}$$

- $O(1)$：求和与比较

$$T(n) = \sum_{k=1}^{n-1} T(k) + \sum_{k=1}^{n-1} T(n-k) + O(n) = 2\sum_{k=1}^{n-1} T(k) + O(n)$$

断言：$T(n) = \Omega(2^{n-1})$

- 归纳基础：$n = 2$，$T(2) \geq c = c_1 2^{2-1}$，令 $c_1 = c/2$。
- 归纳步骤：$P(k < n) \Rightarrow P(n)$。

$$\begin{aligned} T(n) &= O(n) + c_1 2 \sum_{k=1}^{n-1} 2^{k-1} \quad // \text{ 归纳假设} \\ &\geq O(n) + c_1 2(2^{n-1} - 1) = \Omega(2^{n-1}) \quad // \text{ 等比级数} \end{aligned}$$

本质上与暴力算法相同

# 低效的根源（$n = 5$ 的情况）

[见原文第 17 页图示：递归树]

15 个不同的子问题 vs. 计算 81 个子问题

# 低效的根源（$n = 5$ 的情况）

[见原文第 17 页图示：递归树]

15 个不同的子问题 vs. 计算 81 个子问题

Those who cannot remember the past are condemned to repeat it.

- Dynamic Programming

[见原文第 17 页图片]

# 迭代方法（高效）

size $= 1$：$n$ 个不同的子问题

- $C(i, i) = 0$ 对于 $i \in [n]$（无计算代价）

size $= 2$：$n - 1$ 个不同的子问题

- $C(1, 2), C(2, 3), C(3, 4), \ldots, C(n - 1, n)$

$\ldots$

size $= i$：$n - i + 1$ 个不同的子问题

$\ldots$

size $= n - 1$：2 个不同的子问题

- $C(1, n - 1), C(2, n)$

size $= n$：原问题

- $C(1, n)$

# $n = 8$ 的演示

[见原文第 19 页图示：子问题计算顺序]

$$A_1 \quad A_2 \quad A_3 \quad A_4 \quad A_5 \quad A_6 \quad A_7 \quad A_8$$

size $= 2, 3, 4, 5, 6, 7, 8$

# 算法

# 复杂度分析

## 根据算法

- 第 2 行：子问题规模
- 第 3-4 行：子问题的边界
- 第 5 行：尝试所有分割位置以找到最优断点
- 第 2、3-4、5 行构成三重循环，每层循环的长度为 $O(n)$；内层循环的代价为 $O(1)$ $\rightsquigarrow$ 复杂度 $O(n^3)$

# 复杂度分析

## 根据算法

- 第 2 行：子问题规模
- 第 3-4 行：子问题的边界
- 第 5 行：尝试所有分割位置以找到最优断点
- 第 2、3-4、5 行构成三重循环，每层循环的长度为 $O(n)$；内层循环的代价为 $O(1)$ $\rightsquigarrow$ 复杂度 $O(n^3)$

## 根据备忘录

- 备忘录中共有 $n^2$ 个元素，确定每个元素的值需要 $O(n)$ 的尝试和比较代价 $\rightsquigarrow$ 复杂度 $O(n^3)$

# 复杂度分析

根据算法

- 第 2 行：子问题规模
- 第 3-4 行：子问题的边界
- 第 5 行：尝试所有分割位置以找到最优断点
- 第 2、3-4、5 行构成三重循环，每层循环的长度为 $O(n)$；内层循环的代价为 $O(1)$ $\rightsquigarrow$ 复杂度 $O(n^3)$

根据备忘录

- 备忘录中共有 $n^2$ 个元素，确定每个元素的值需要 $O(n)$ 的尝试和比较代价 $\rightsquigarrow$ 复杂度 $O(n^3)$

回溯复杂度：$n-1$（内部节点的数量）

# 示例

矩阵链：$A_1A_2A_3A_4A_5$，$A_1 : 30 \times 35$，$A_2 : 35 \times 15$，$A_3 : 15 \times 5$，$A_4 : 5 \times 10$，$A_5 : 10 \times 20$

| | | | | |
|---|---|---|---|---|
| $\ell = 2$ | $C(1,2) = 15750$ | $C(2,3) = 2625$ | $C(3,4) = 750$ | $C(4,5) = 1000$ |
| $\ell = 3$ | $C(1,3) = 7875$ | $C(2,4) = 4375$ | $C(3,5) = 2500$ | |
| $\ell = 4$ | $C(1,4) = 9375$ | $C(2,5) = 7125$ | | |
| $\ell = 5$ | $C(1,5) = 11875$ | | | |

| | | | | |
|---|---|---|---|---|
| $\ell = 2$ | $s(1,2) = 1$ | $s(2,3) = 2$ | $s(3,4) = 3$ | $s(4,5) = 4$ |
| $\ell = 3$ | $s(1,3) = 1$ | $s(2,4) = 3$ | $s(3,5) = 3$ | |
| $\ell = 4$ | $s(1,4) = 3$ | $s(2,5) = 3$ | | |
| $\ell = 5$ | $s(1,5) = 3$ | | | |

$$s(1,5) \Rightarrow (A_1A_2A_3)(A_4A_5)$$
$$s(1,3) \Rightarrow A_1(A_2A_3)$$

- 最优计算顺序：$(A_1(A_2A_3))(A_4A_5)$
- 最小乘法次数：$C(1,5) = 11875$

# 目录

# 二叉搜索树

设 $S$ 是一个有序集合，元素为 $x_1 < x_2 < \cdots < x_n$。为了支持高效搜索，我们将它们存储在二叉树的节点上。

搜索：如果 $x \in S$，输出索引。否则，输出所在区间。

[见原文第 26 页图示：二叉搜索树]

$x$ 与根比较

- $x <$ 根，进入左子树；
- $x >$ 根，进入右子树；
- $x =$ 根，停止并输出 $x$；

$x$ 到达叶节点，停止，输出 $\perp$。

# 搜索元素的分布

当 $x \xleftarrow{R} S$ 时 $\Rightarrow$ 平衡二叉树是最优的

如果 $x$ 的分布不是均匀的呢?

设 $S = (x_1, \ldots, x_n)$。考虑区间 $(x_0, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n), (x_n, x_{n+1})$，其中 $x_0 = -\infty$，$x_{n+1} = +\infty$

- $\Pr[x = x_i] = b_i$，$\Pr[x \in (x_i, x_{i+1})] = a_i$

$x$ 在 $S \cup \bar{S}$ 上的分布为

$$P = (a_0, b_1, a_1, b_2, a_2, \ldots, b_n, a_n)$$

**示例**：$S = (1, 2, 3, 4, 5, 6)$。$x$ 的分布 $P$ 为

$$(0.04, 0.1, 0.01, 0.2, 0.05, 0.2, 0.02, 0.1, 0.02, 0.1, 0.07, 0.05, 0.04)$$

- $x = 1, 2, 3, 4, 5, 6$：0.1, 0.2, 0.2, 0.1, 0.1, 0.05
- $x$ 落在区间内：0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04

# 二叉搜索树 1

[见原文第 28 页图示：二叉搜索树 $T_1$]

$S = (1, 2, 3, 4, 5, 6)$
$(0.1, 0.2, 0.2, 0.1, 0.1, 0.05)$
$(0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04)$

平均搜索次数：

$$\begin{aligned} A(T_1) &= [1 \times 0.1 + 2 \times (0.2 + 0.05) + 3 \times (0.1 + 0.2 + 0.1)] \\ &\quad + [3 \times (0.04 + 0.01 + 0.05 + 0.02 + 0.02 + 0.07) + 2 \times 0.04] \\ &= 1.8 + 0.71 = 2.51 \end{aligned}$$

# 二叉搜索树 2

[见原文第 29 页图示：二叉搜索树 $T_2$]

$S = (1, 2, 3, 4, 5, 6)$
$(0.1, 0.2, 0.2, 0.1, 0.1, 0.05)$
$(0.04, 0.01, 0.05, 0.02, 0.02, 0.07, 0.04)$

平均搜索次数：

$$\begin{aligned} A(T_2) = &[1 \times 0.1 + 2 \times 0.2 + 3 \times 0.1 + 4 \times (0.2 + 0.05) + 5 \times 0.1] \\ &+ [1 \times 0.04 + 2 \times 0.01 + 4 \times (0.05 + 0.02 + 0.04) \\ &+ 5 \times (0.02 + 0.07)] = 2.3 + 0.95 = 3.25 \end{aligned}$$

# 平均搜索时间公式

集合 $S = (x_1, x_2, \ldots, x_n)$

分布 $P = (a_0, b_1, a_1, b_2, \ldots, a_i, b_{i+1}, \ldots, b_n, a_n)$

- $x_i$ 在 $T$ 中的深度为 $d(x_i)$，$i = 1, 2, \ldots, n$。
  - 深度从 0 开始计数
  - 第 $k$ 层节点需要 $k+1$ 次比较
- 区间 $I_j$ 的深度为 $d(I_j)$，$j = 0, 1, \ldots, n$。

平均搜索时间

$$A(T) = \sum_{i=1}^{n} b_i(1 + d(x_i)) + \sum_{j=0}^{n} a_j d(I_j)$$

当所有节点的深度增加 1 时，平均搜索时间增加：

$$\sum_{i=1}^{n} b_i + \sum_{j=0}^{n} a_j$$

# 最优搜索树建模

问题：给定集合 $S = (x_1, x_2, \ldots, x_n)$ 和搜索元素的分布 $P = (a_0, b_1, a_1, b_2, a_2, \ldots, b_n, a_n)$
目标：找到一棵最优二叉搜索树（平均搜索次数最小）

[见原文第 31 页图示]

Dynamic
Programming

# 动态规划

子问题：由 $(i, j)$ 定义，$i$ 是左边界，$j$ 是右边界

- 数据集：$S[i, j] = (x_i, x_{i+1}, \ldots, x_j)$
- 分布：$P[i, j] = (a_{i-1}, b_i, a_i, b_{i+1}, \ldots, b_j, a_j)$

输入实例：$S = (A, B, C, D, E)$
$P = (0.04, 0.1, 0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01)$

**子问题**：$(2, 4)$

- $S[2, 4] = (B, C, D)$
- $P[2, 4] = (0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06)$

# 分解为子问题

以 $x_k$ 为根，将一个问题分解为两个子问题：

- $S[i, k-1]$，$P[i, k-1]$
- $S[k+1, j]$，$P[k+1, j]$

**示例**：选择节点 $B$ 为根，将原问题分解为以下两个子问题：

**子问题**：$(1, 1)$

- $S[1, 1] = (A)$，$P[1, 1] = (0.04, 0.1, 0.02)$

**子问题**：$(3, 5)$

- $S[3, 5] = (C, D, E)$，$P[3, 5] = (0.02, 0.1, 0.05, 0.2, 0.06, 0.1, 0.01)$

[见原文第 33 页图示：以 $B$ 为根的分解]

# 子问题的概率和

对于子问题 $S[i,j]$ 和 $P[i,j]$，$P[i,j]$ 中的概率和（包括元素和区间）为：

$$w[i,j] = \sum_{s=i-1}^{j} a_s + \sum_{t=i}^{j} b_t$$

**子问题 $(2,4)$ 的示例**

- $S[2,4] = (B, C, D)$
- $P[2,4] = (0.02, 0.3, 0.02, 0.1, 0.05, 0.2, 0.06)$
- $w[2,4] = (0.3 + 0.1 + 0.2) + (0.02 + 0.02 + 0.05 + 0.06) = 0.75$

# 优化函数

优化函数 $\mathrm{OPT}(i,j)$：对于 $S[i,j]$，$P[i,j]$，子问题 $(i,j)$ 的最优平均比较次数。
参数化优化函数：$\mathrm{OPT}_k(i,j)$：以 $x_k$ 为根时的最优平均比较次数
初始值：$\mathrm{OPT}(i,i-1)=0$ 对于 $i=1,2,\ldots,n,n+1$，对应空子问题。

**示例**：$S=(A,B,C,D,E)$

1. 选择 $A$ 为根（$k=1$），产生子问题 $(1,0)$ 和 $(2,5)$，$(1,0)$ 是空子问题：对应 $S[1,0]$，$\mathrm{OPT}(1,0)=0$
2. 选择 $E$ 为根（$k=5$），产生子问题 $(1,4)$ 和 $(6,5)$，$(6,5)$ 是空子问题：对应 $S[6,5]$，$\mathrm{OPT}(6,5)=0$

# 优化函数的迭代关系

$$\begin{aligned}\mathsf{OPT}(i,j) &= \min_{i\le k\le j}\{\mathsf{OPT}_k(i,j)\}, \quad 1\le i\le j\le n\\ &= \min_{i\le k\le j}\{\underline{\mathsf{OPT}(i,k-1)} + \underline{\mathsf{OPT}(k+1,j)} + w[i,j]\}\end{aligned}$$

[见原文第 36 页图示：以 $x_k$ 为根的子树结构]

- 左子树和右子树中所有节点的深度都增加 1

$$w[i,k-1] + b_k + w[k+1,j] = w[i,j]$$

# $\mathsf{OPT}_k(i,j)$ 的证明

$$
\begin{aligned}
&\mathsf{OPT}_k(i,j)\\
&= (\mathsf{OPT}(i,k-1) + w[i,k-1]) + (\mathsf{OPT}(k+1,j) + w[k+1,j]) + b_k\\
&= (\mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j)) + (w[i,k-1] + b_k + w[k+1,j])\\
&= (\mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j))\\
&\quad + \left(\sum_{s=i-1}^{k-1} a_s + \sum_{t=i}^{k-1} b_t\right) + b_k + \left(\sum_{s=k}^{j} a_s + \sum_{t=k+1}^{j} b_t\right)\\
&= (\mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j)) + \sum_{s=i-1}^{j} a_s + \sum_{t=i}^{j} b_t \quad \text{// 化简}\\
&= \mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j) + w[i,j]
\end{aligned}
$$

# 伪代码

# 演示

$$\mathsf{OPT}(i,j) = \min_{i \le k \le j} \{\mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j) + w[i,j]\}$$

对于 $1 \le i \le j \le n$
$\mathsf{OPT}(i,i-1) = 0$，$i = 1,2,\ldots,n,n+1$

[见原文第 39 页图示：最优二叉搜索树示例]

选择 $B$ 为根，$k = 2$
$\mathsf{OPT}(1,1) = 0.16$，$\mathsf{OPT}(3,5) = 0.88$
$\mathsf{OPT}(3,3) = 0.17$，$\mathsf{OPT}(5,5) = 0.17$
$w[3,5] = 0.54$

$$\begin{aligned}\mathsf{OPT}(1,5) &= 1 + \min_{k \in [5]} \{\mathsf{OPT}(1,k-1) + \mathsf{OPT}(k+1,5)\} \\ &= 1 + (\mathsf{OPT}(1,1) + \mathsf{OPT}(3,5)) = 1 + (0.16 + 0.88) = 2.04\end{aligned}$$

# 复杂度分析

$$\mathsf{OPT}(i,j) = \min_{i \le k \le j} \{\mathsf{OPT}(i,k-1) + \mathsf{OPT}(k+1,j) + w[i,j]\}$$

对于 $1 \le i \le j \le n$
$\mathsf{OPT}(i, i-1) = 0$，$i = 1, 2, \ldots, n, n+1$

$(i,j)$ 组合的数量为 $O(n^2)$
对于每个 $\mathsf{OPT}(i,j)$，计算需要计算 $k$ 项并找最小值。每项计算的代价是常数时间。

- 时间复杂度：$T(n) = O(n^3)$
- 空间复杂度：$S(n) = O(n^2)$