动态规划篇：钢条切割问题

问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

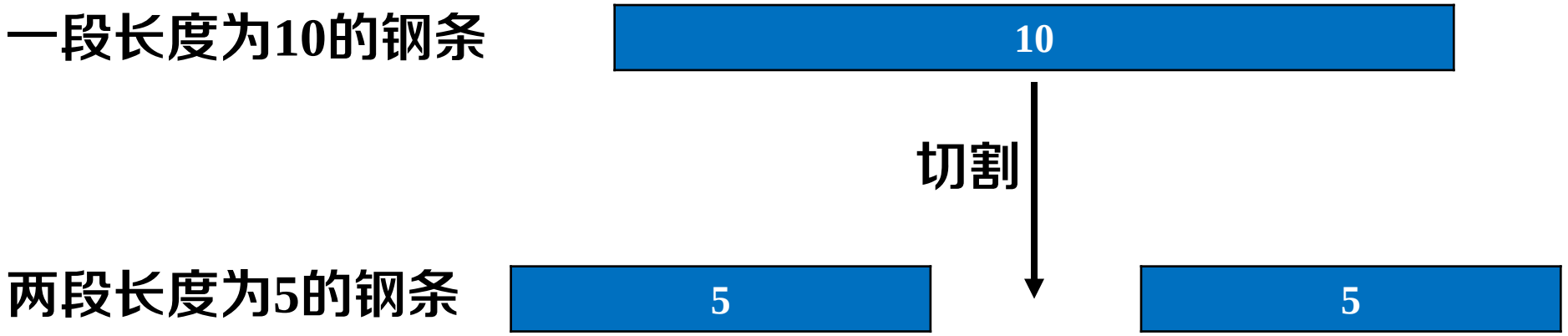
钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

一段长度为10的钢条

问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

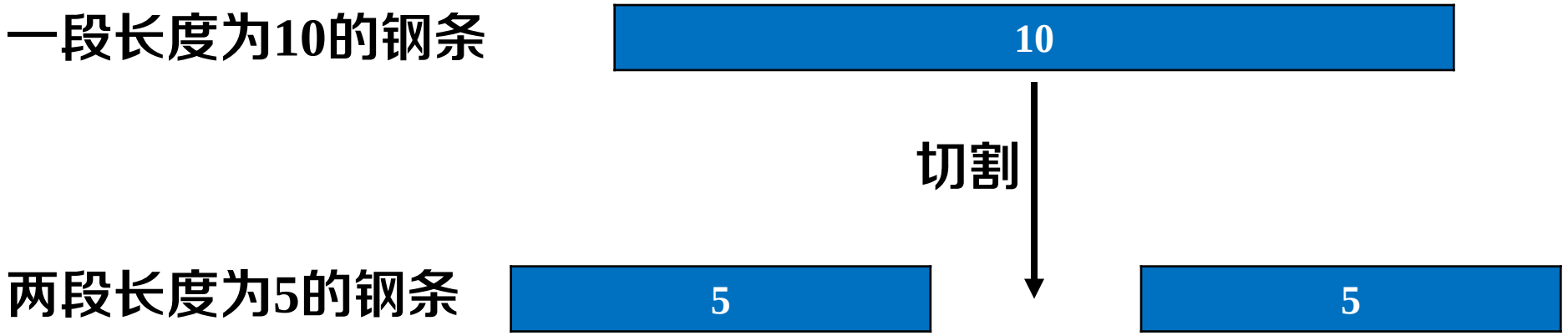
钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24



问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

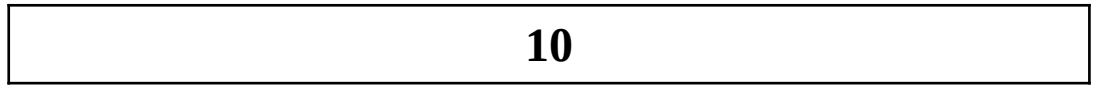


问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

切割方案		总收益
方案1	{10}	24

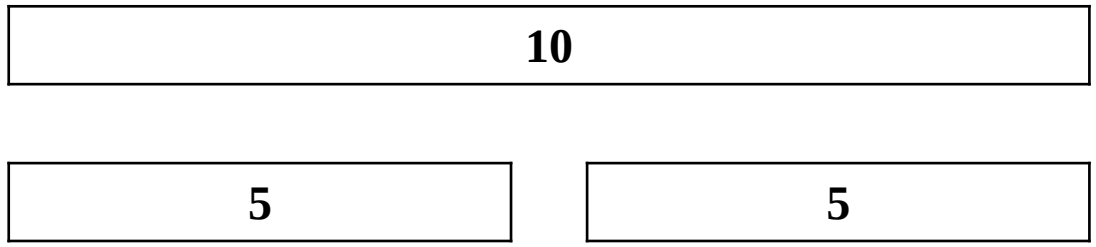


问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

切割方案		总收益
方案1	{10}	24
方案2	{5,5}	10+10=20

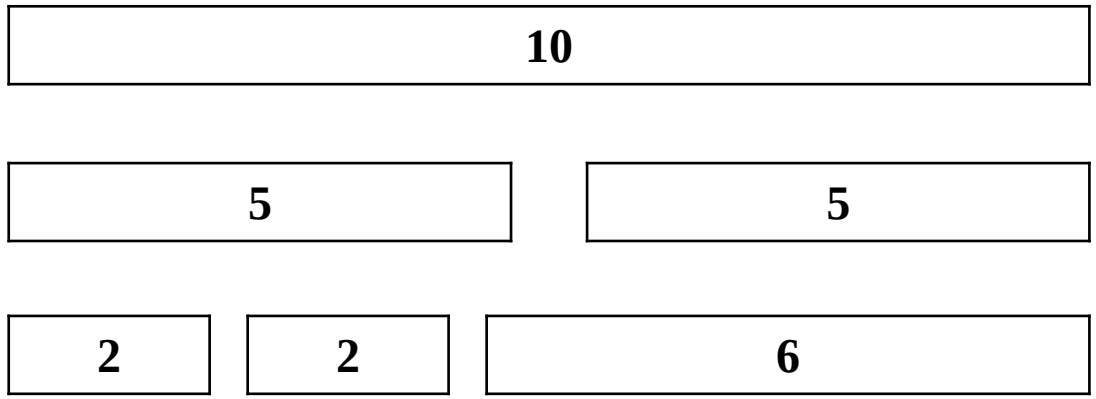


问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

切割方案		总收益
方案1	{10}	24
方案2	{5,5}	10+10=20
方案3	{2,2,6}	5+5+17=27

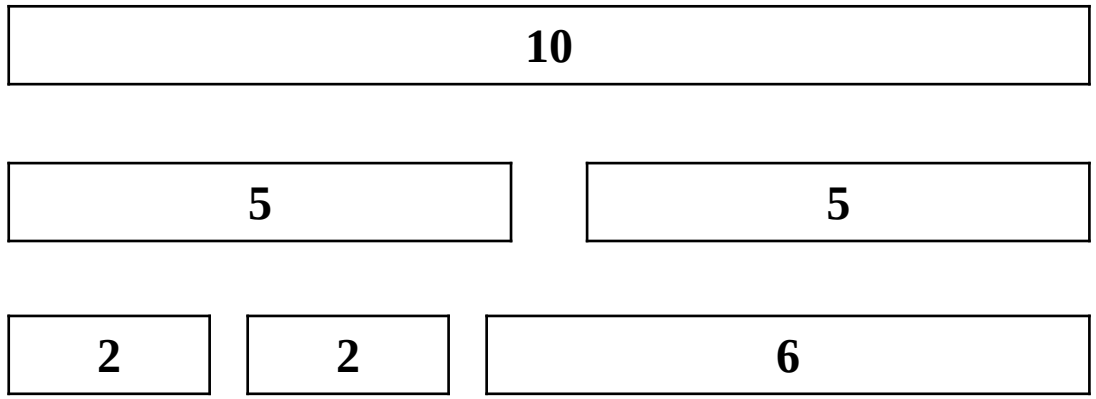


问题背景

- 钢条切割
  - 现有一段长度为10的钢条，可以零成本将其切割为多段长度更小钢条

钢条长度	0	1	2	3	4	5	6	7	8	9	10
价格p	0	1	5	8	9	10	17	17	20	24	24

切割方案		总收益
方案1	{10}	24
方案2	{5,5}	10+10=20
方案3	{2,2,6}	5+5+17=27



问题：怎样合理切割，使总收益最大?

问题定义

- 形式化定义

钢条切割问题

Rod Cutting Problem

输入

- 钢条长度n

问题定义

- 形式化定义

钢条切割问题

Rod Cutting Problem

输入

- 钢条长度n
- 价格表$p_l(1 \leq l \leq n)$：表示长度为l的钢条价格

问题定义

- 形式化定义

钢条切割问题

Rod Cutting Problem

输入

- 钢条长度n
- 价格表$p_l(1 \le l \le n)$：表示长度为l的钢条价格

输出

- 求解一组切割方案$T =< c_1, c_2, \dots, c_m >$，令

$$\max \sum_{l=1}^{m} p_{c_l}$$

$$s.t. \sum_{l=1}^{m} c_l = n$$

问题定义

- 形式化定义

钢条切割问题

Rod Cutting Problem

输入

- 钢条长度n
- 价格表$p_l(1 \le l \le n)$：表示长度为l的钢条价格

输出

- 求解一组切割方案$T = < c_1, c_2, \dots, c_m >$，令

$$\max \sum_{l=1}^{m} p_{c_l}$$

优化目标

$$s.t. \sum_{l=1}^{m} c_l = n$$

问题定义

- 形式化定义

钢条切割问题

Rod Cutting Problem

输入

- 钢条长度n
- 价格表$p_l(1 \le l \le n)$：表示长度为l的钢条价格

输出

- 求解一组切割方案$T = < c_1, c_2, \dots, c_m >$，令

$$\max \sum_{l=1}^{m} p_{c_l}$$

优化目标

$$s.t. \sum_{l=1}^{m} c_l = n$$

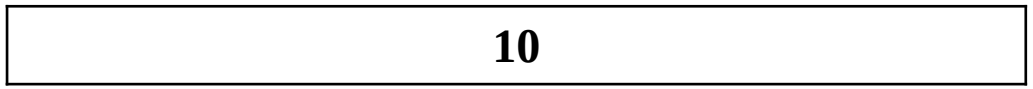
约束条件

问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置

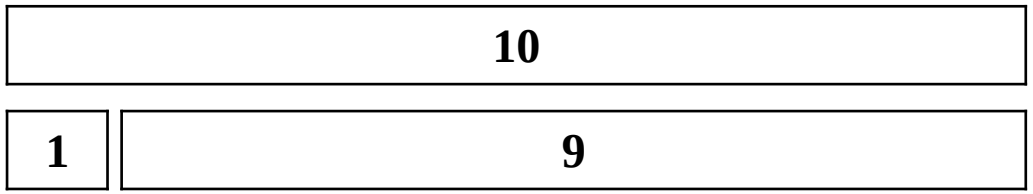
问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置
    - 不切：$p[10]$



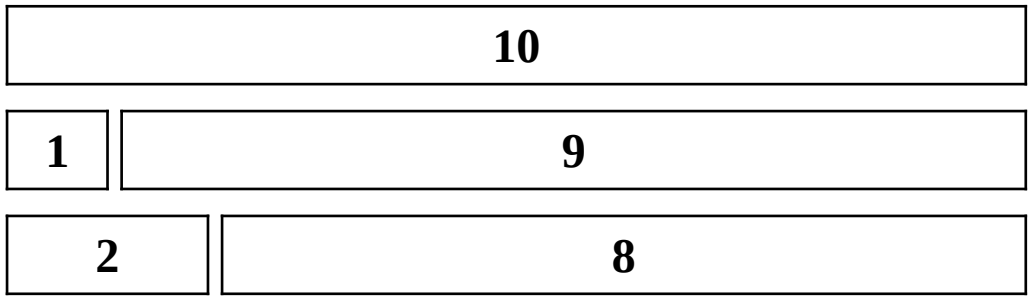
问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置
    - 不切：$p[10]$
    - 切割：$p[i]+p[10-i]$



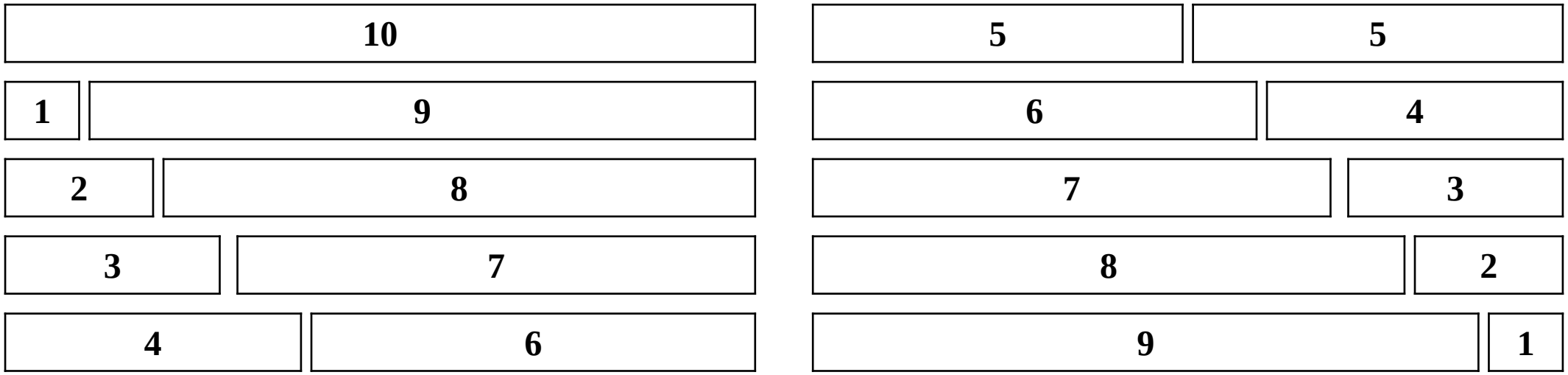
问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置
    - 不切：$p[10]$
    - 切割：$p[i]+p[10-i]$



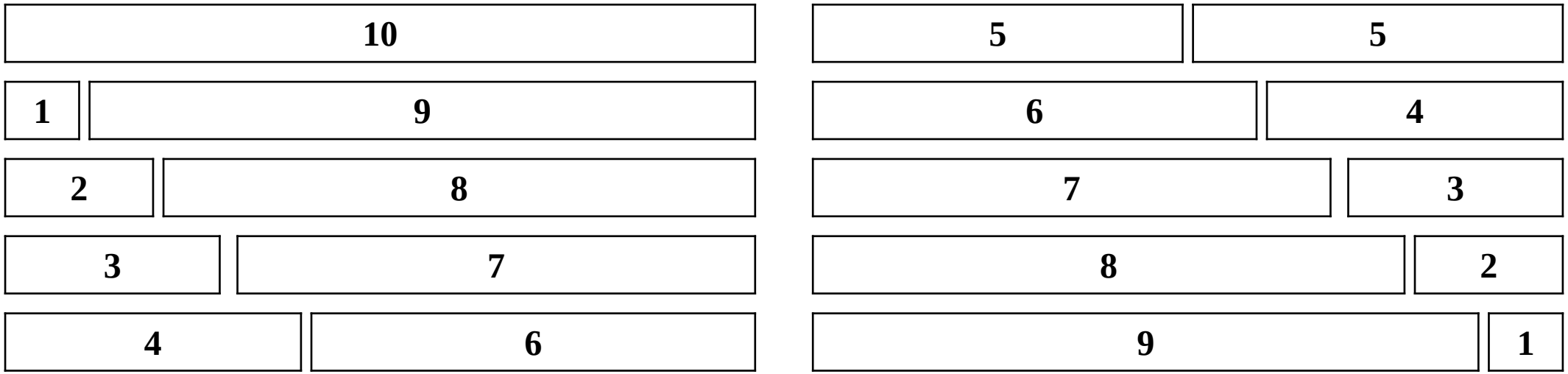
问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置
    - 不切：$p[10]$
    - 切割：$p[i]+p[10-i]$



问题简化

- 假设至多切割1次
  - 枚举所有可能的切割位置
    - 不切：$p[10]$
    - 切割：$p[i]+p[10-i]$
  - 最大收益$\max\limits_{1\le i\le 9}\{p[i]+p[10-i],p[10]\}$

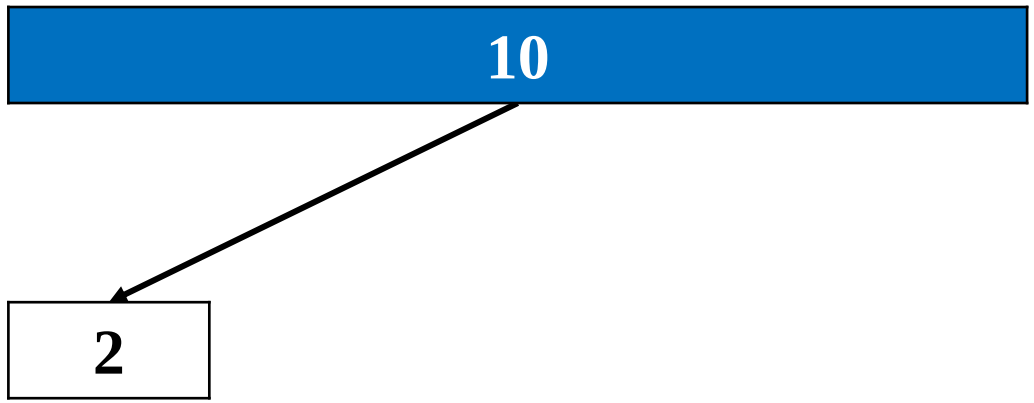


问题简化

- 假设至多切割2次

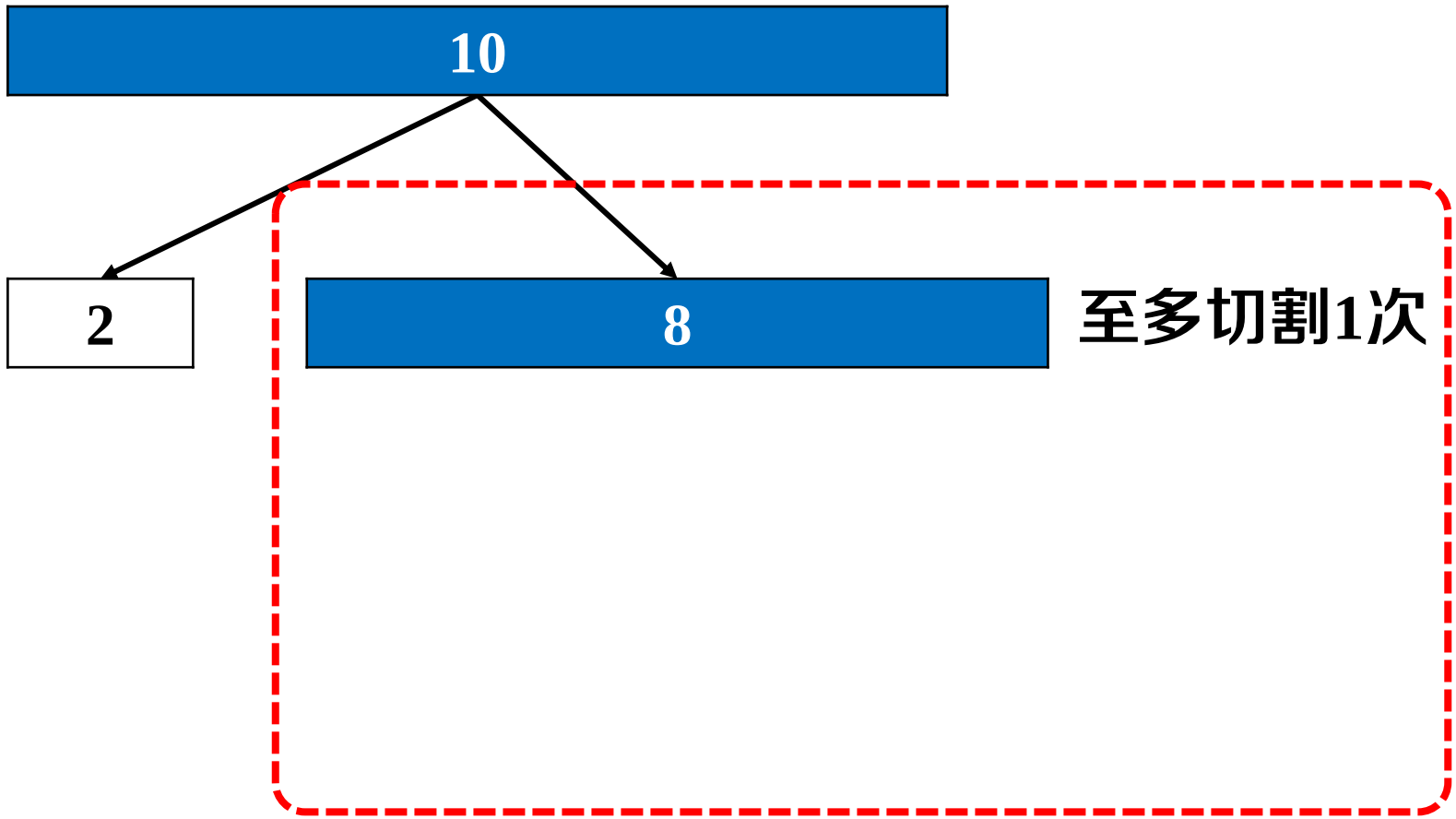
问题简化

- 假设至多切割2次
  - 先将钢条切割出一段



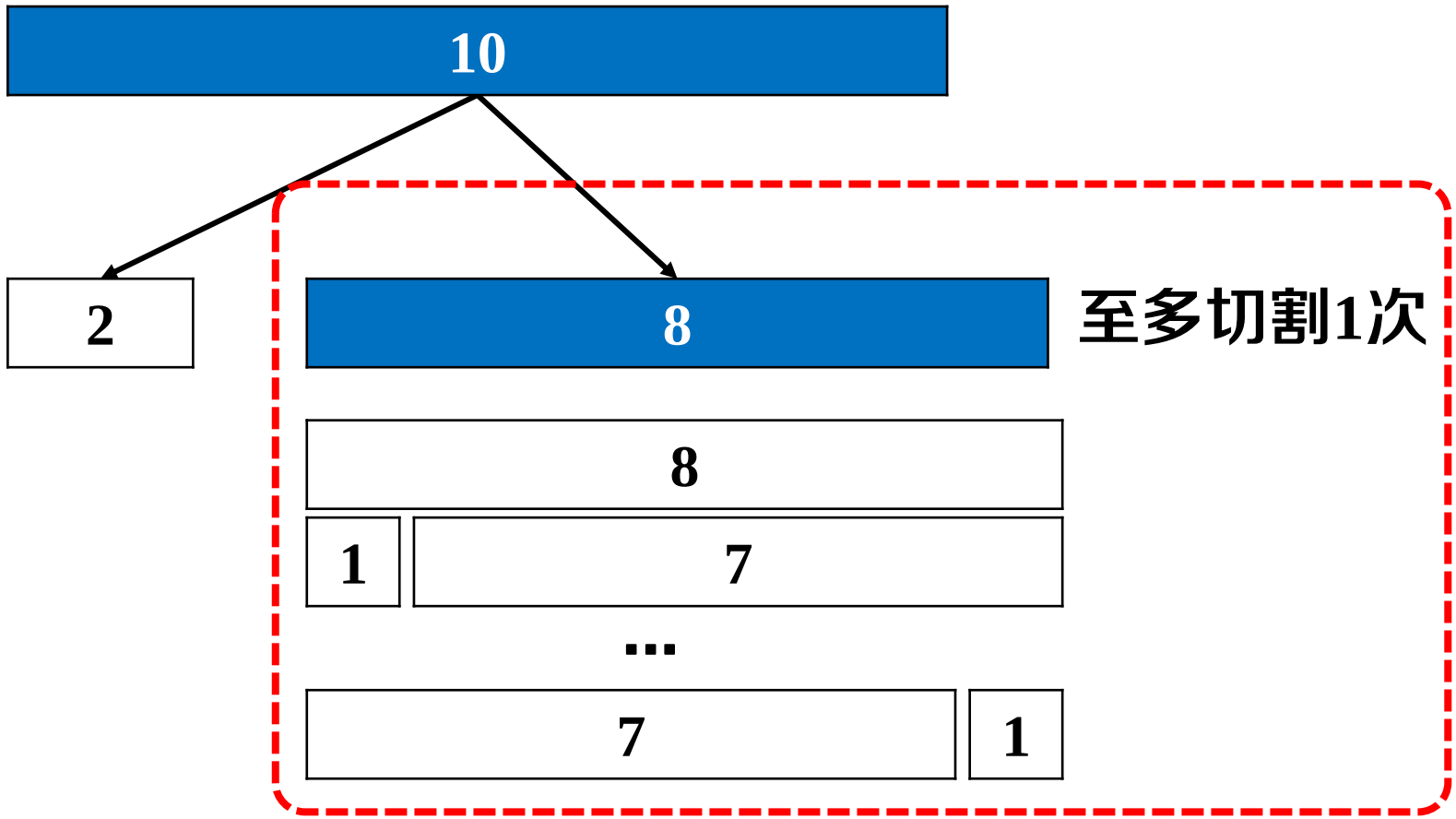
问题简化

- 假设至多切割2次
  - 先将钢条切割出一段
  - 在剩余钢条中继续切割



问题简化

- 假设至多切割2次
  - 先将钢条切割出一段
  - 在剩余钢条中继续切割



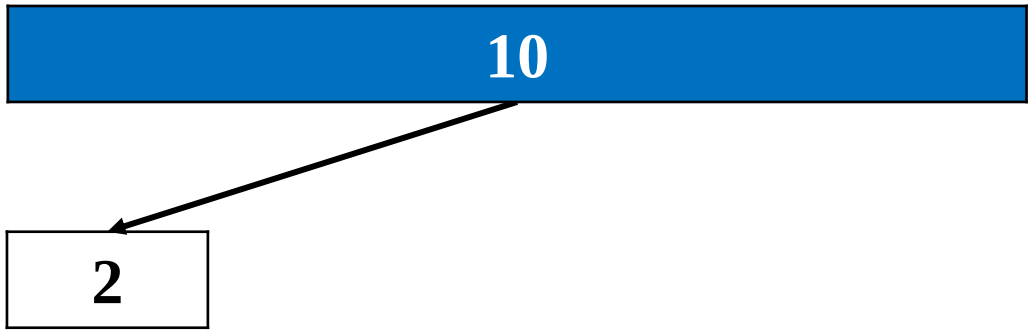
问题观察

- 原始问题不限制切割次数

枚举切割长度，或不切割

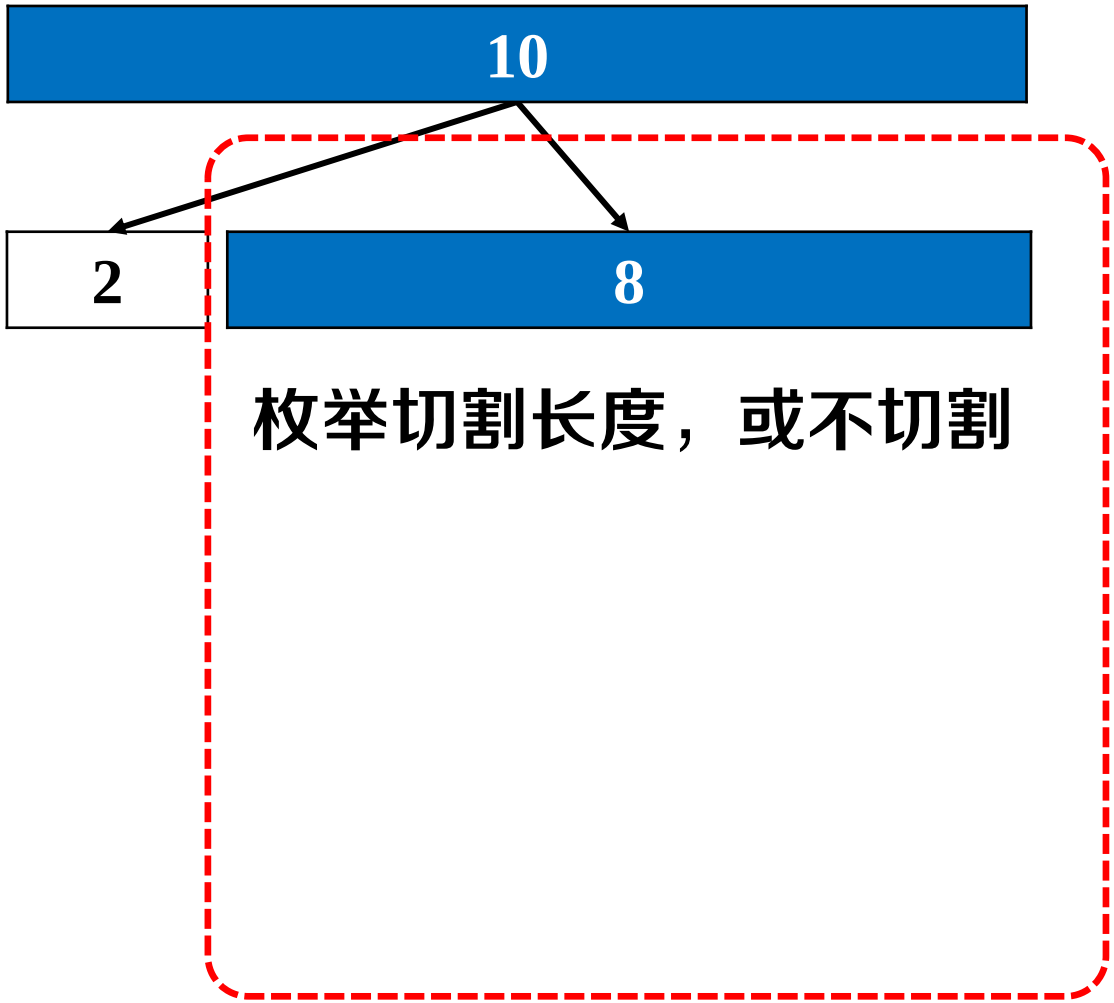
问题观察

- 原始问题不限制切割次数



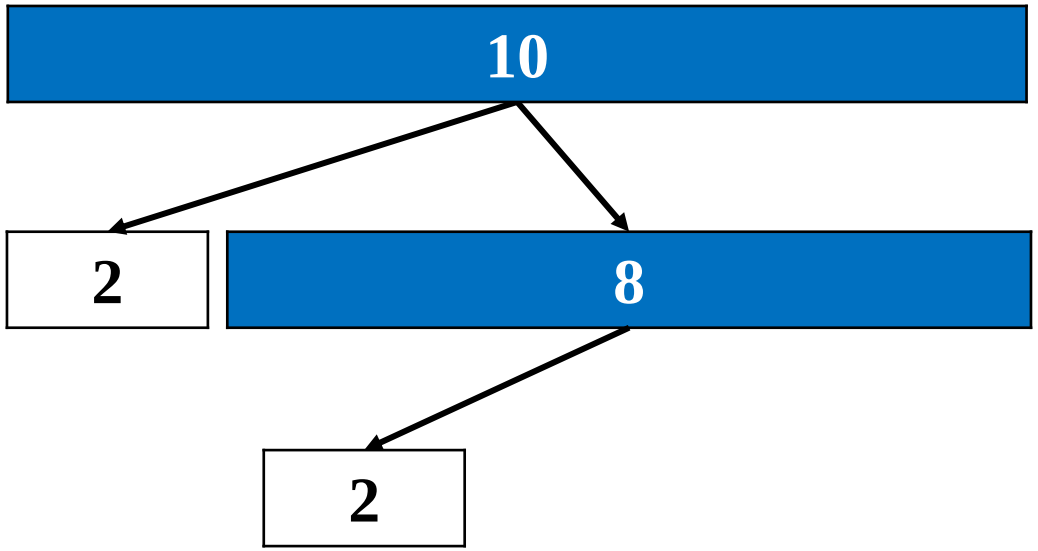
问题观察

- 原始问题不限制切割次数



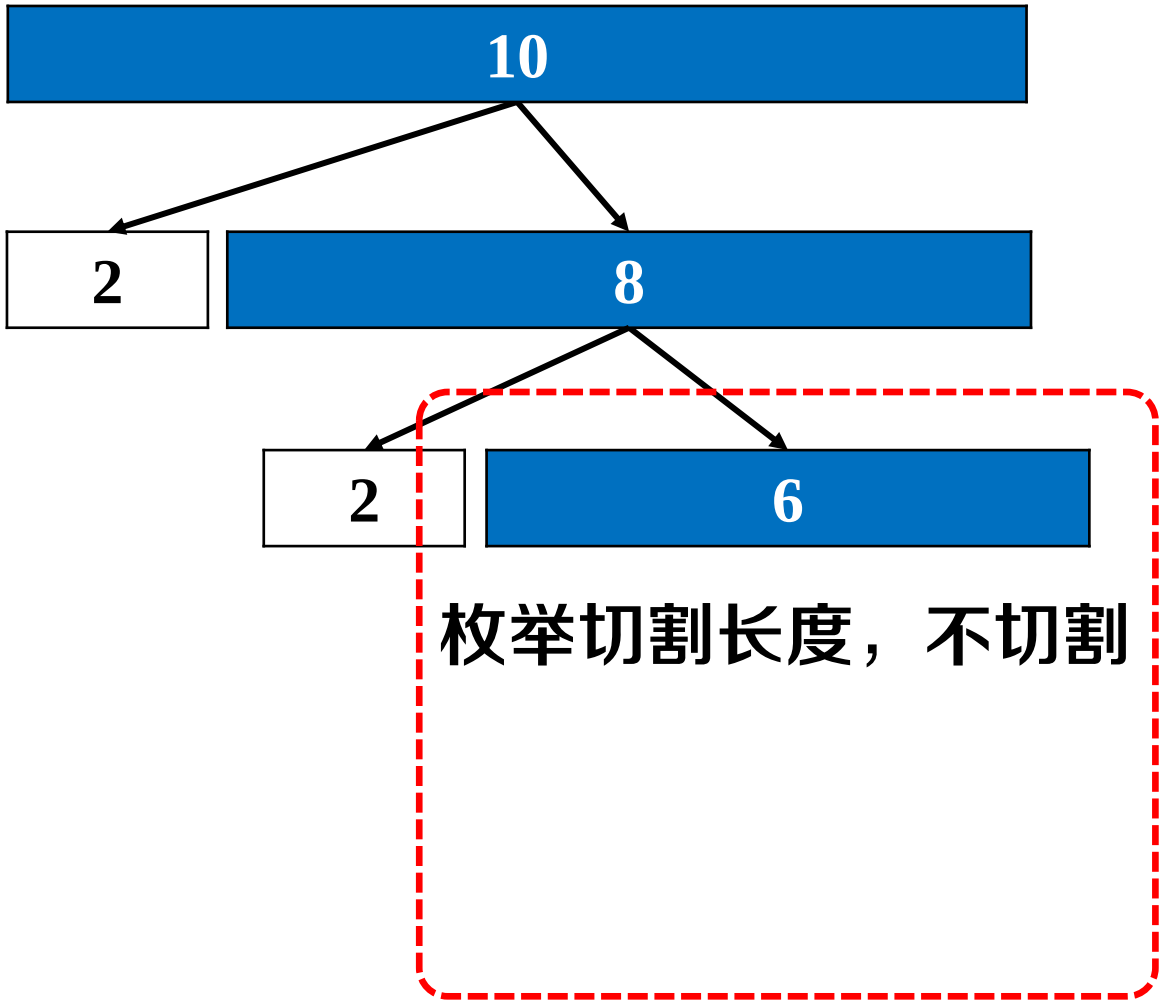
问题观察

- 原始问题不限制切割次数



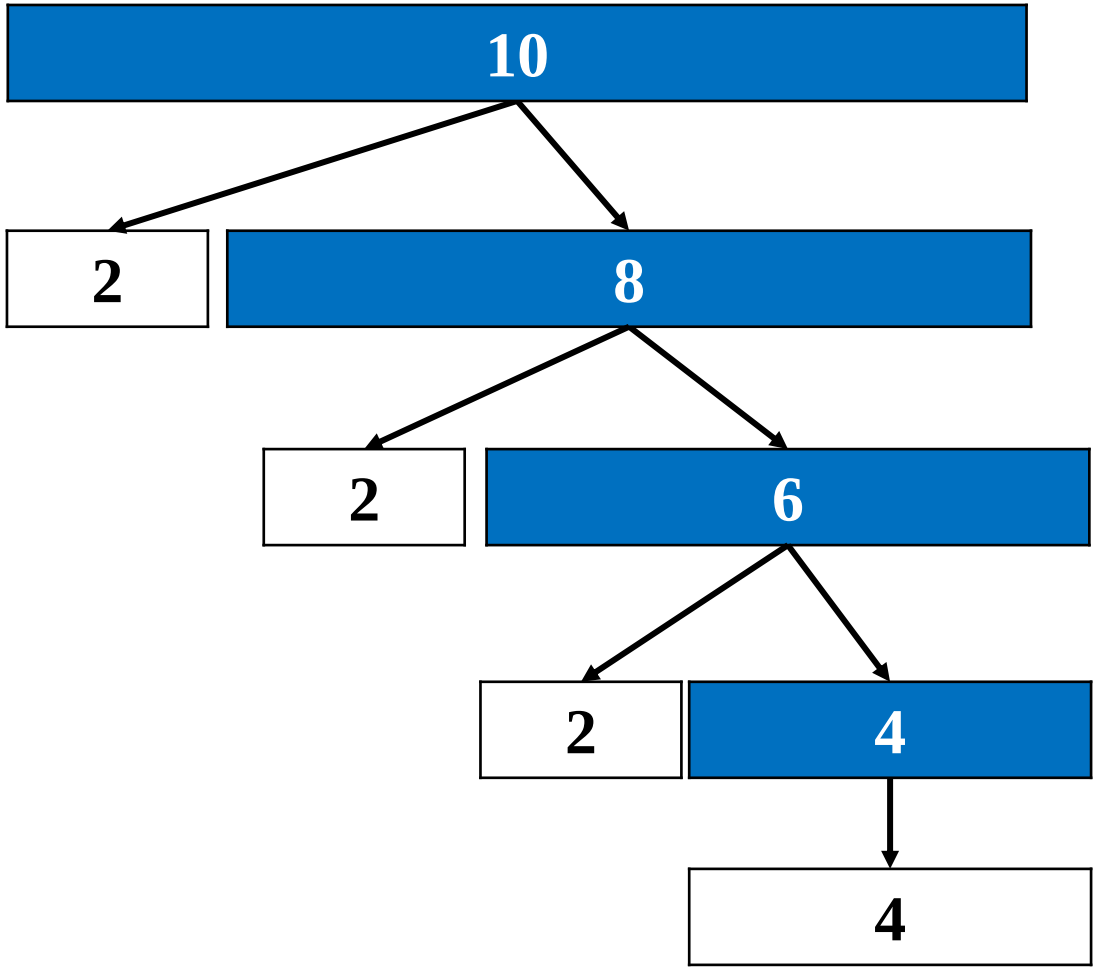
问题观察

- 原始问题不限制切割次数



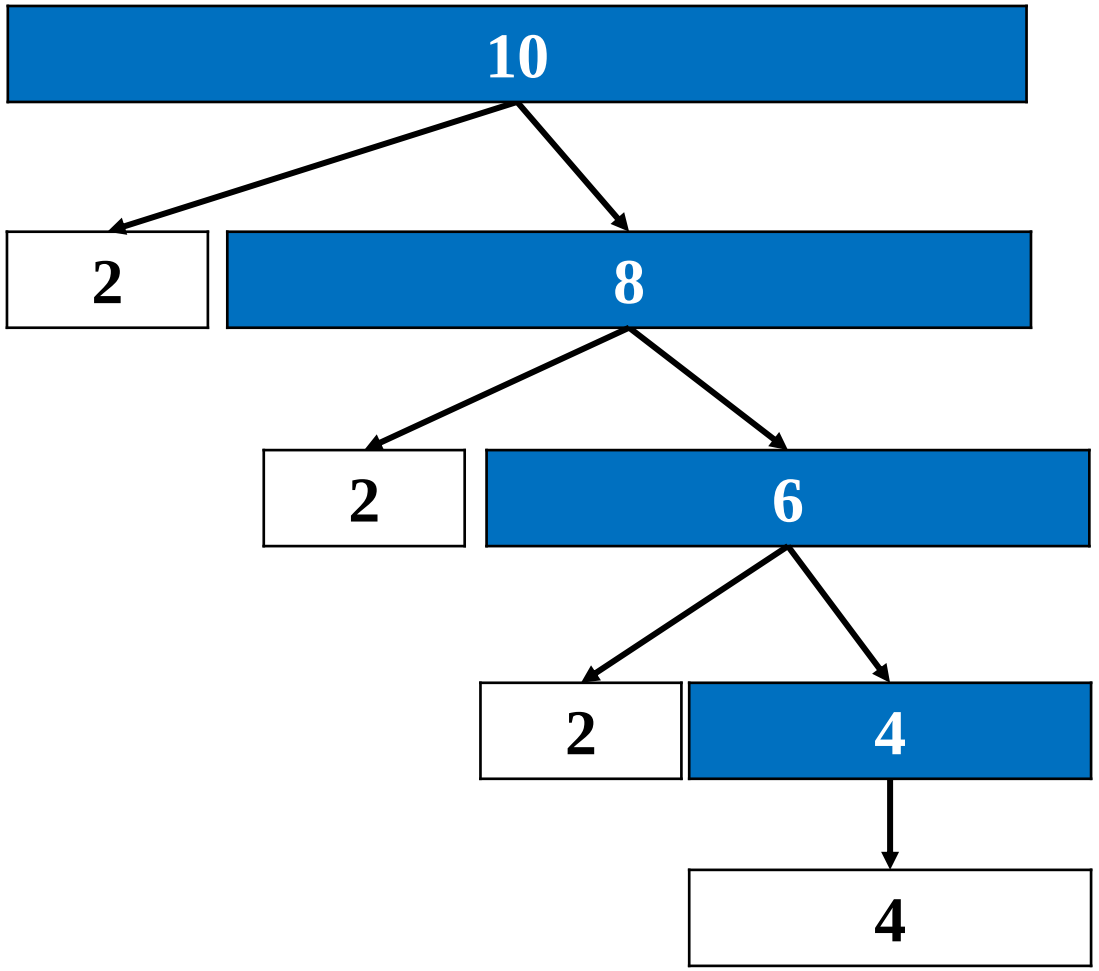
问题观察

- 原始问题不限制切割次数



问题观察

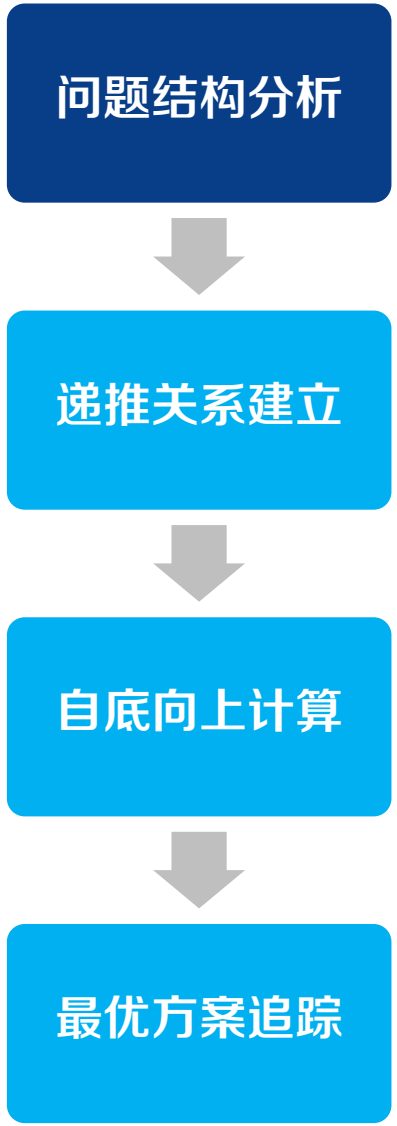
- 原始问题不限制切割次数



- 可能存在最优子结构和重叠子问题

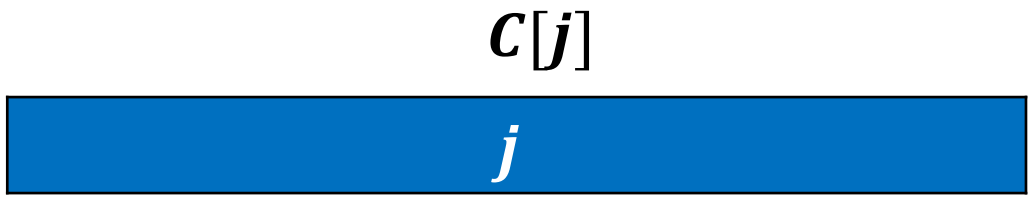
问题结构分析

- 给出问题表示
  - $c[j]$：切割长度为j的钢条可得最大总收益

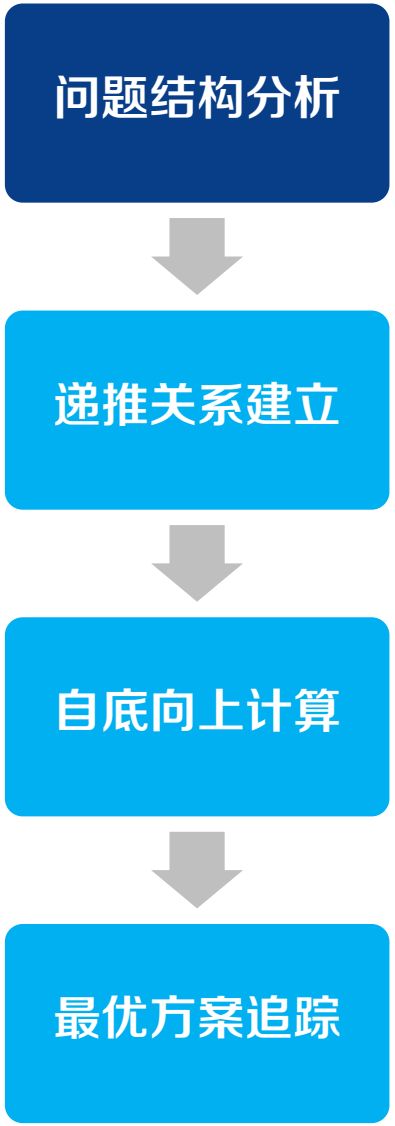


问题结构分析

- 给出问题表示
  - $C[j]$：切割长度为j的钢条可得最大总收益



- 明确原始问题
  - $C[n]$：切割长度为n的钢条可得最大总收益



递推关系建立：分析最优（子）结构

$C[10]$

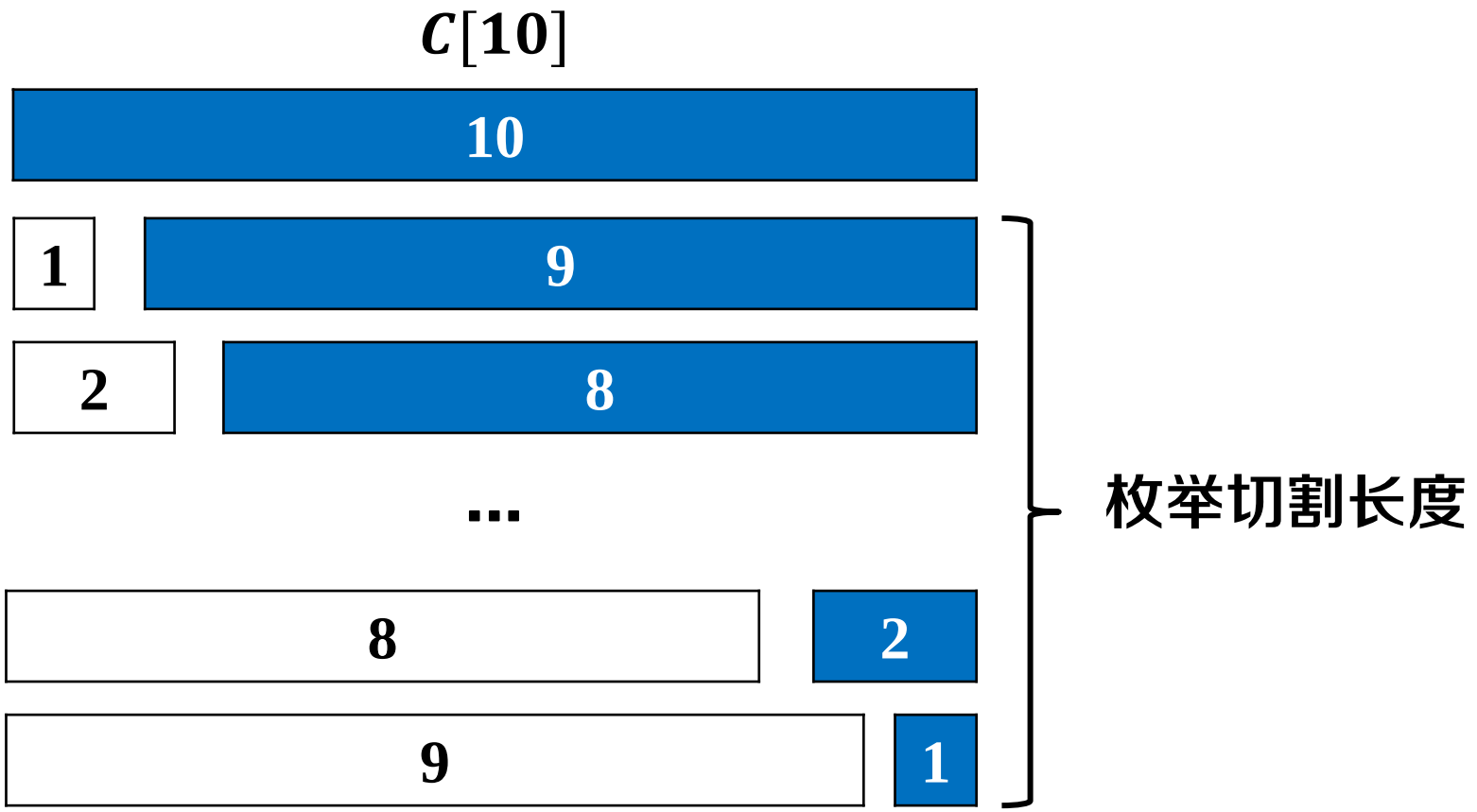
问题结构分析

递推关系建立

自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构



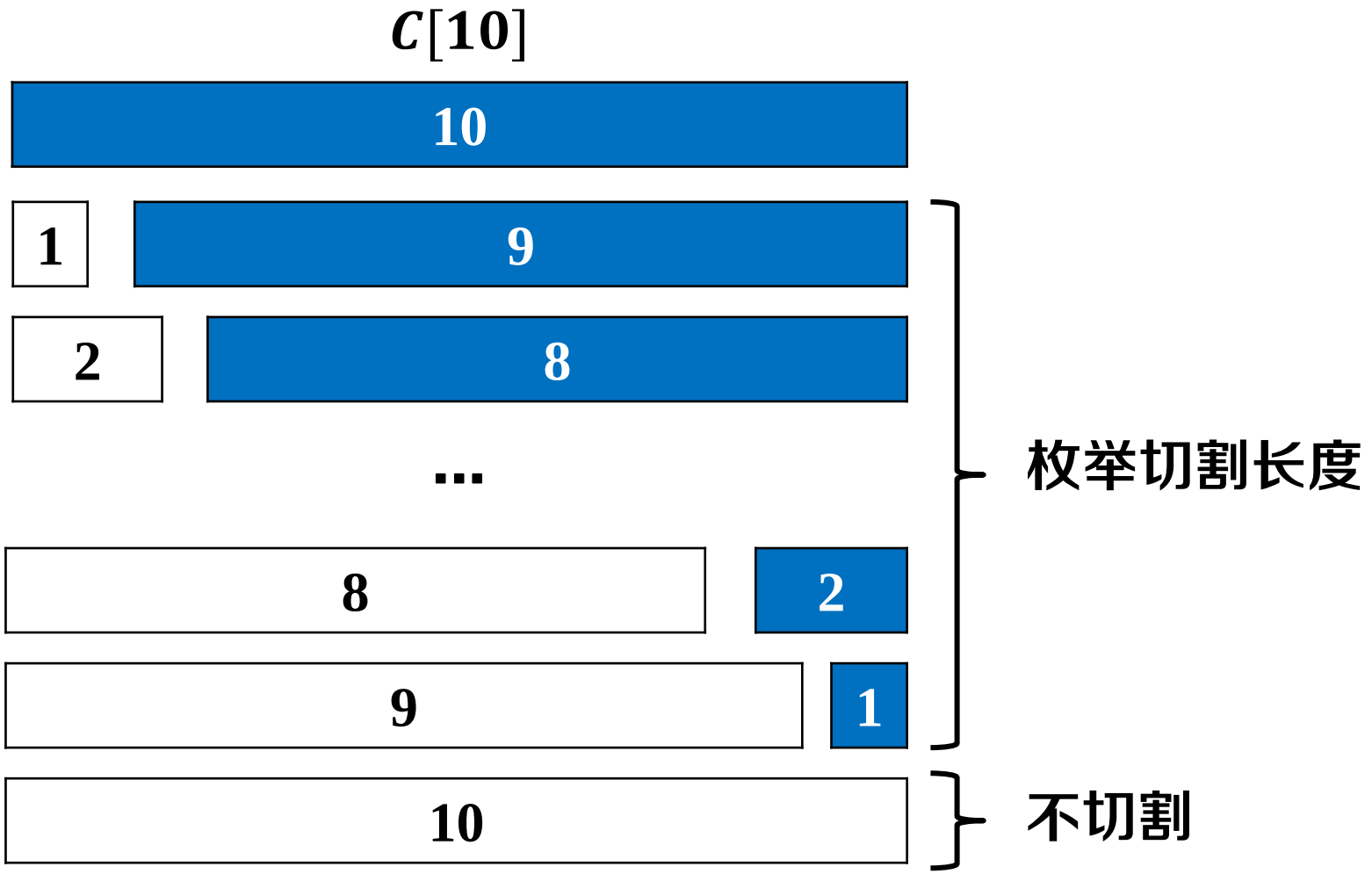
问题结构分析

递推关系建立

自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构



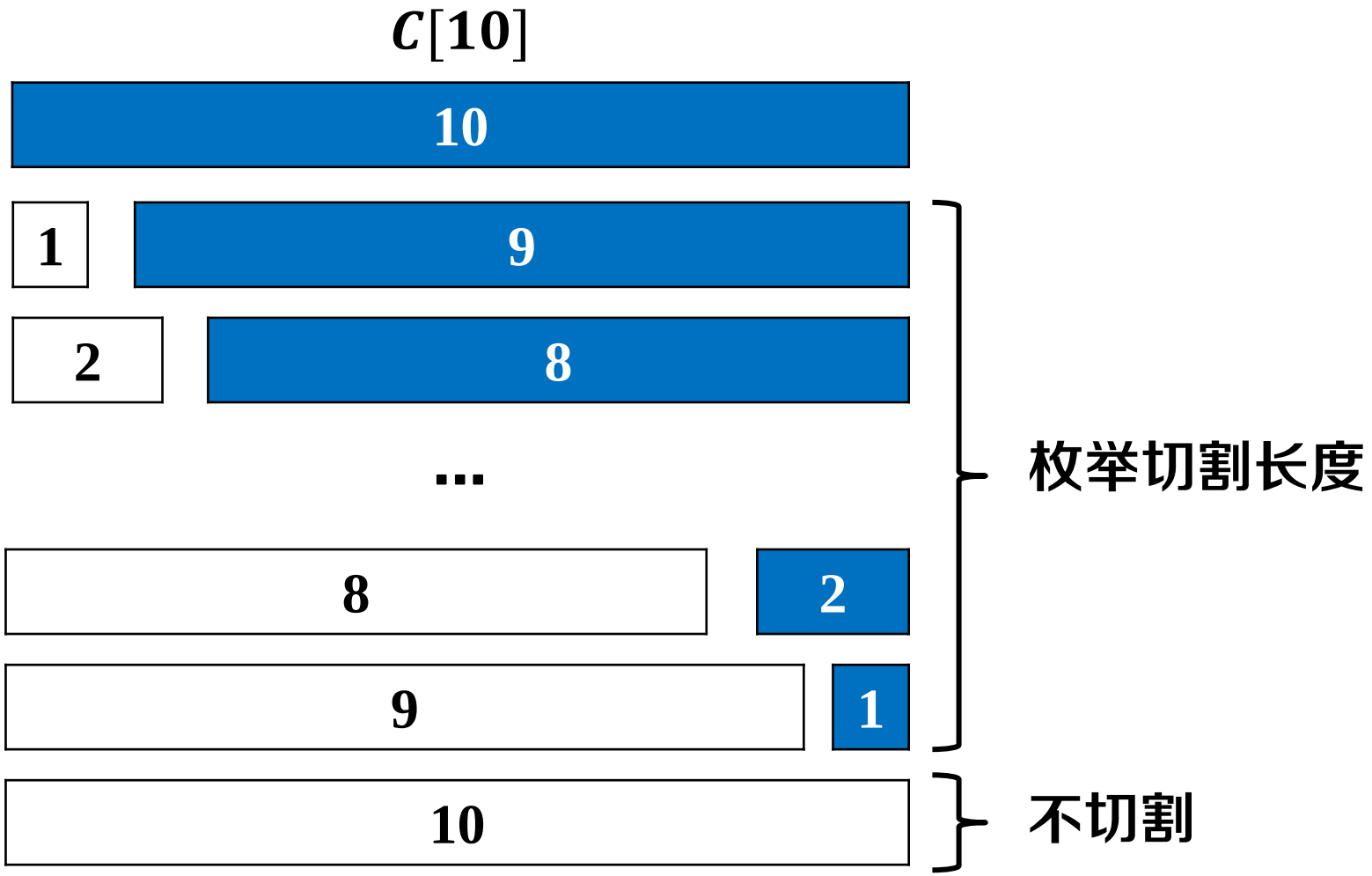
问题结构分析

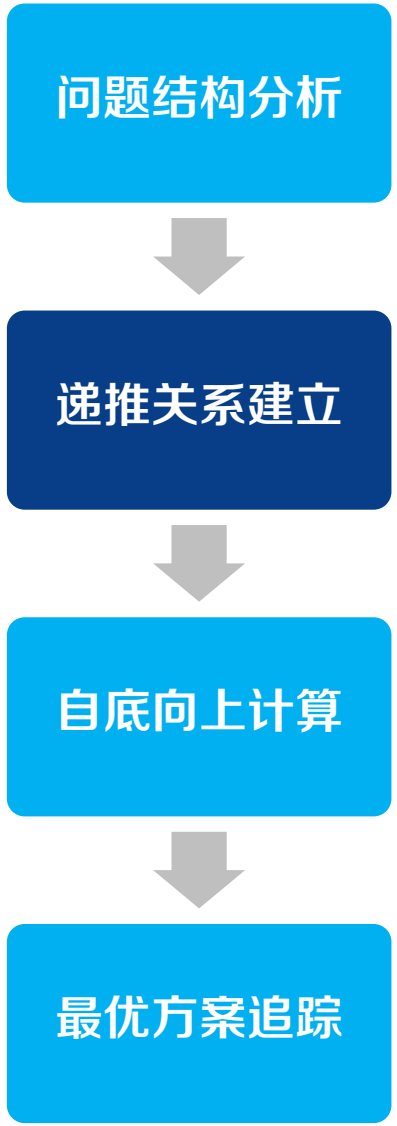
递推关系建立

自底向上计算

最优方案追踪

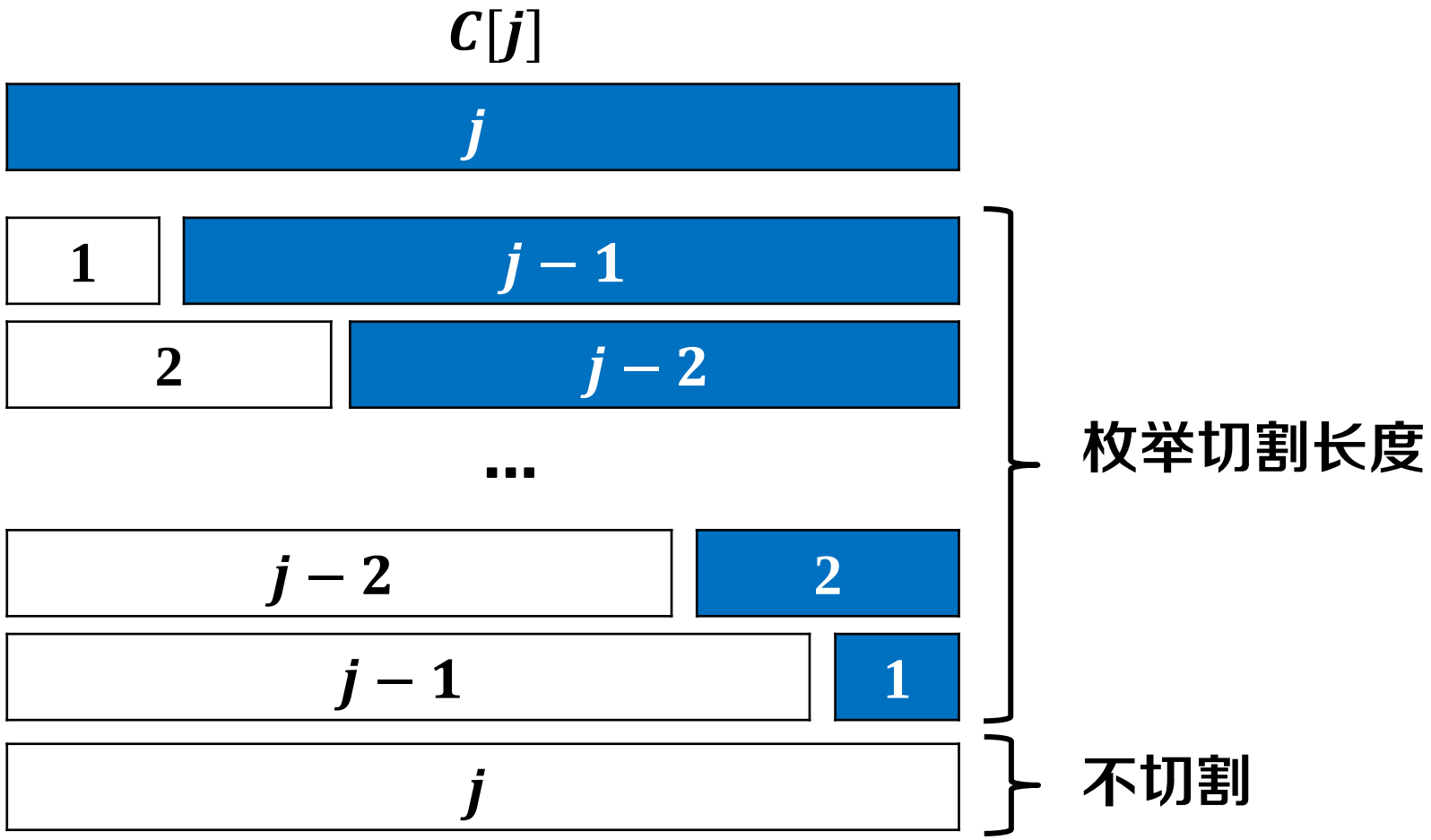
递推关系建立：分析最优（子）结构





- $C[10] = \max_{1 \le i \le 9}\{p[i] + C[10-i], p[10]\}$

递推关系建立：分析最优（子）结构



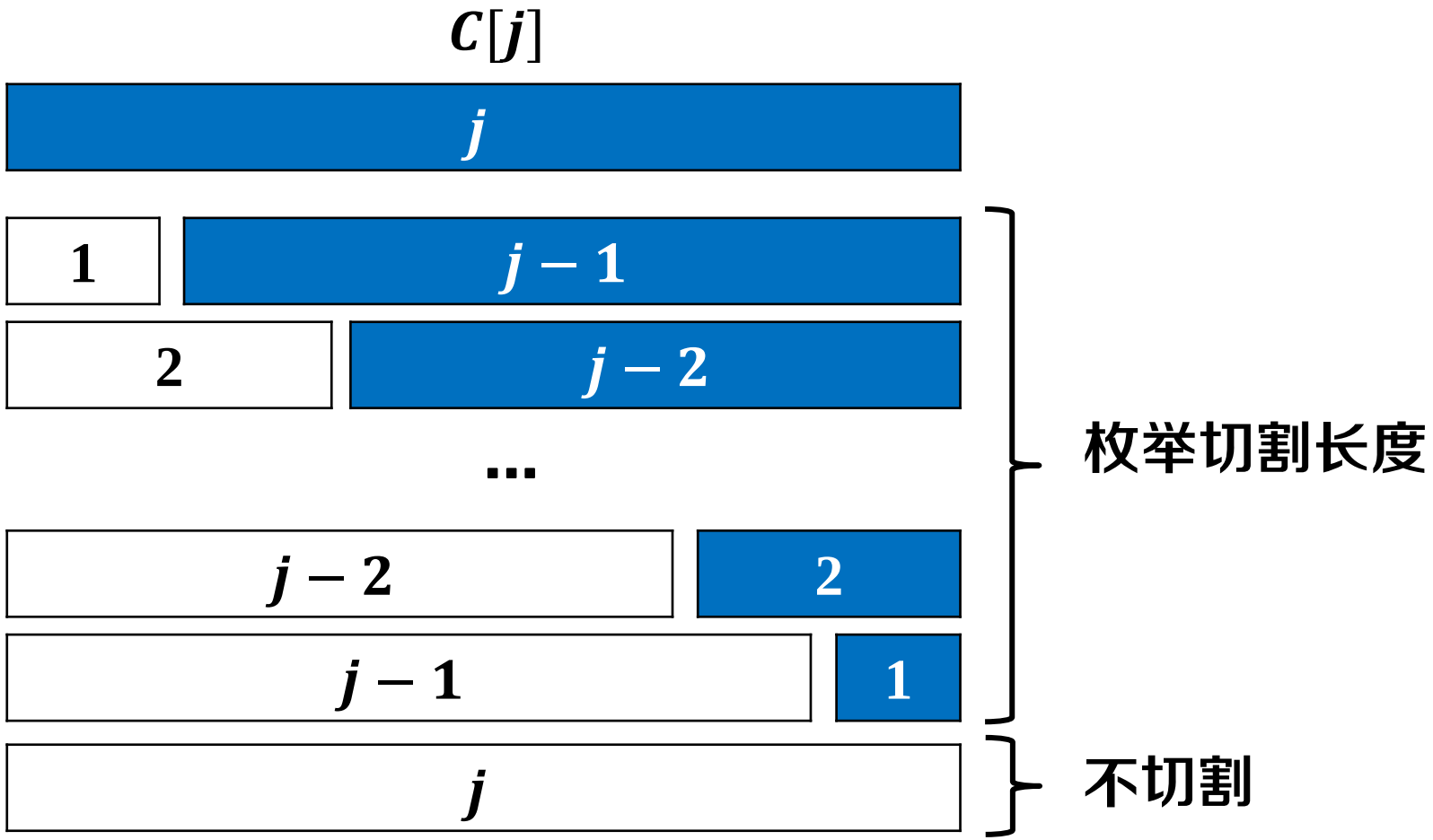
问题结构分析

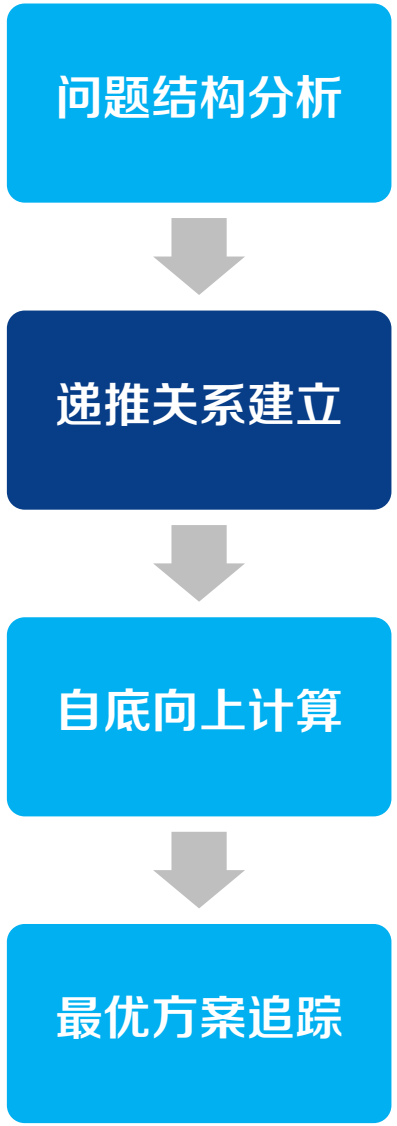
递推关系建立

自底向上计算

最优方案追踪

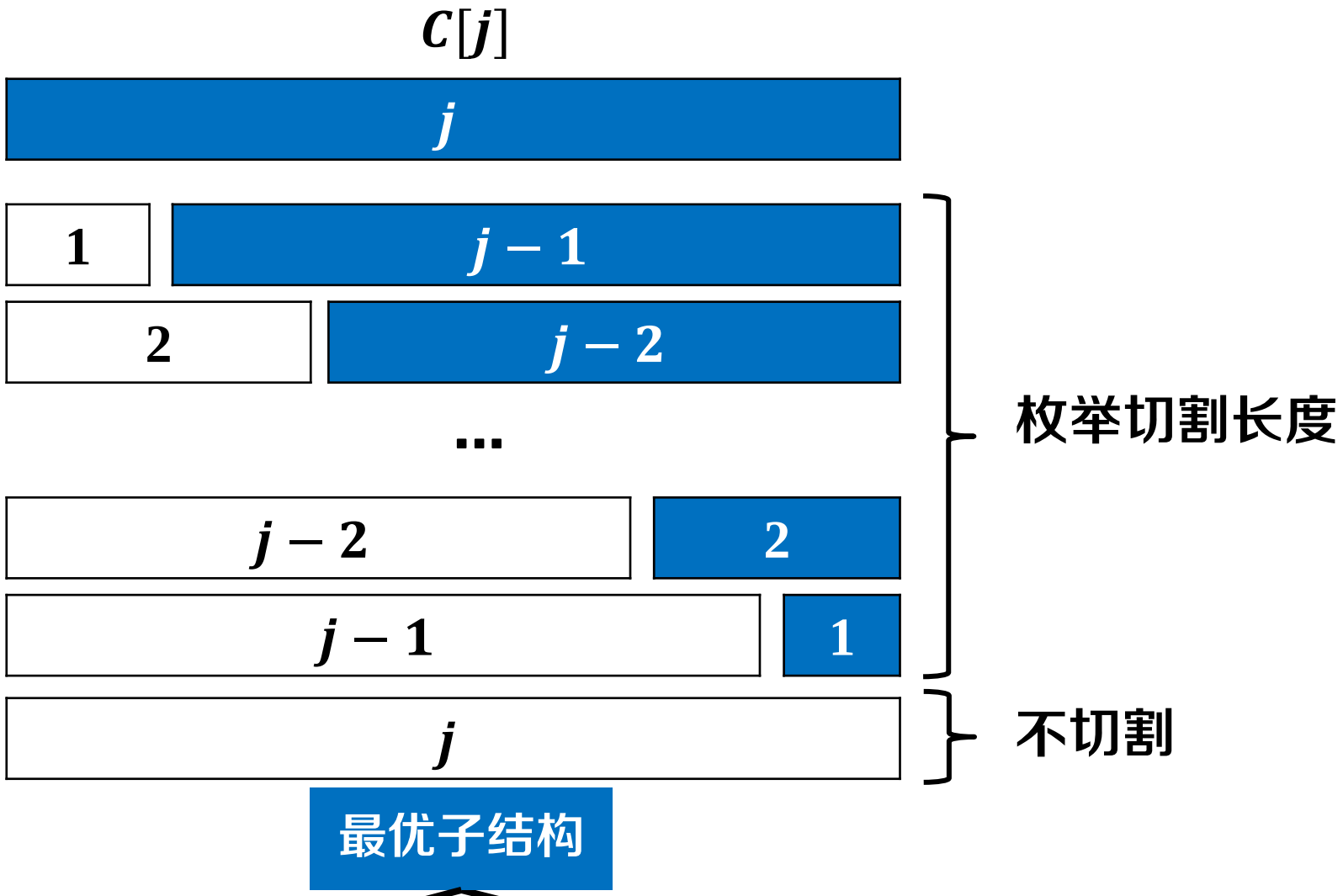
递推关系建立：分析最优（子）结构





- $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$

递推关系建立：分析最优（子）结构



- $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$

问题结构分析

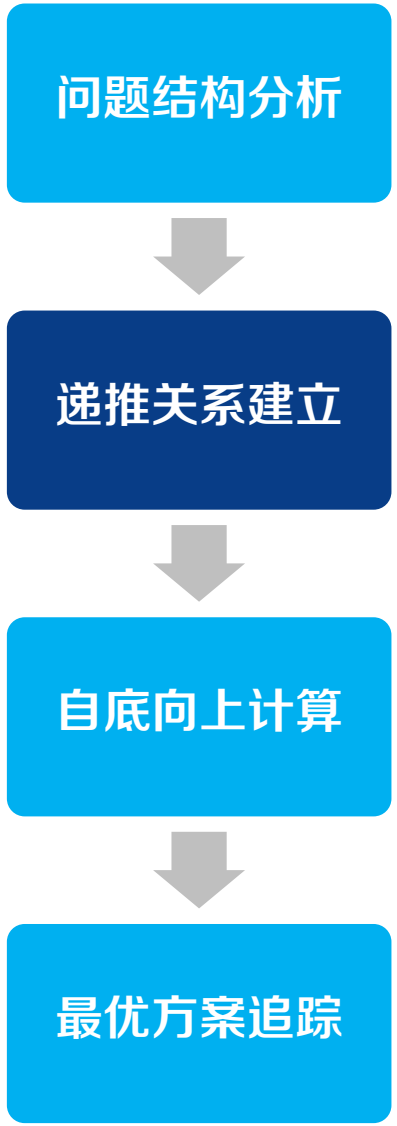
递推关系建立

自底向上计算

最优方案追踪

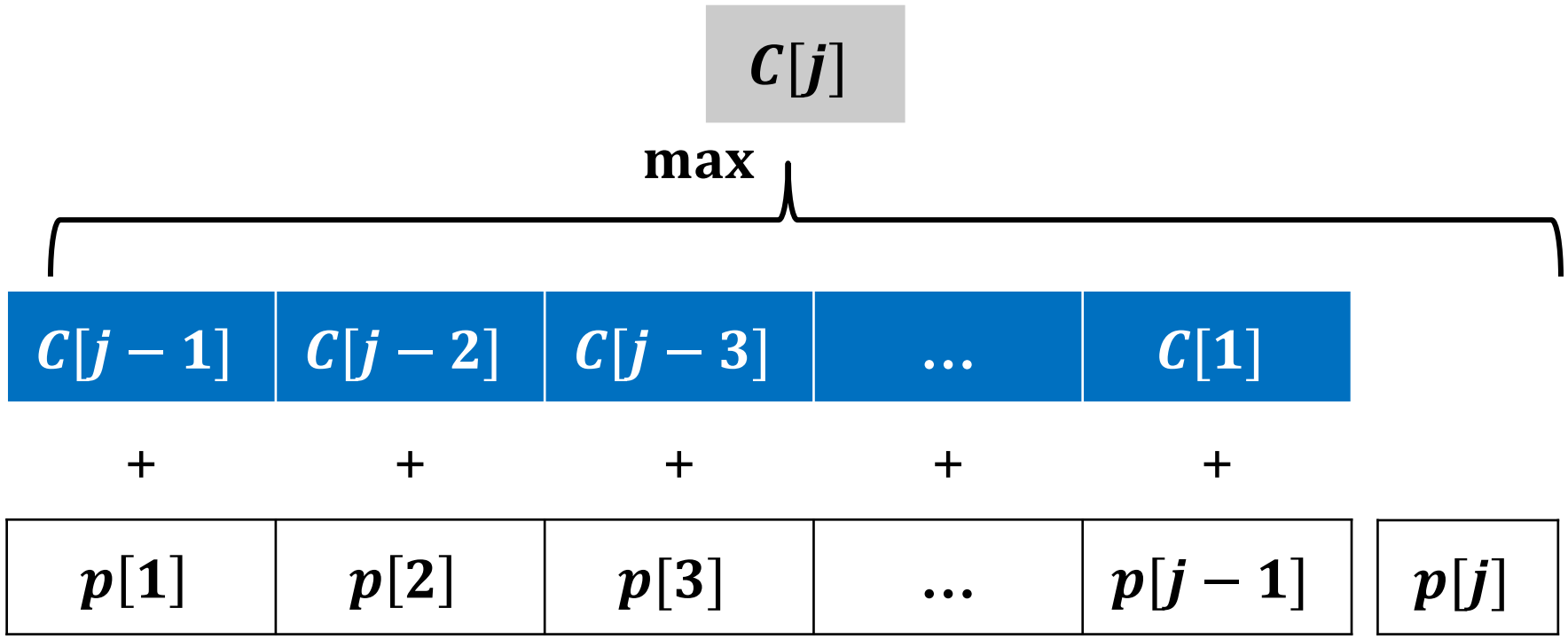
递推关系建立：构造递推公式

- 对于每个钢条长度j
  - $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$



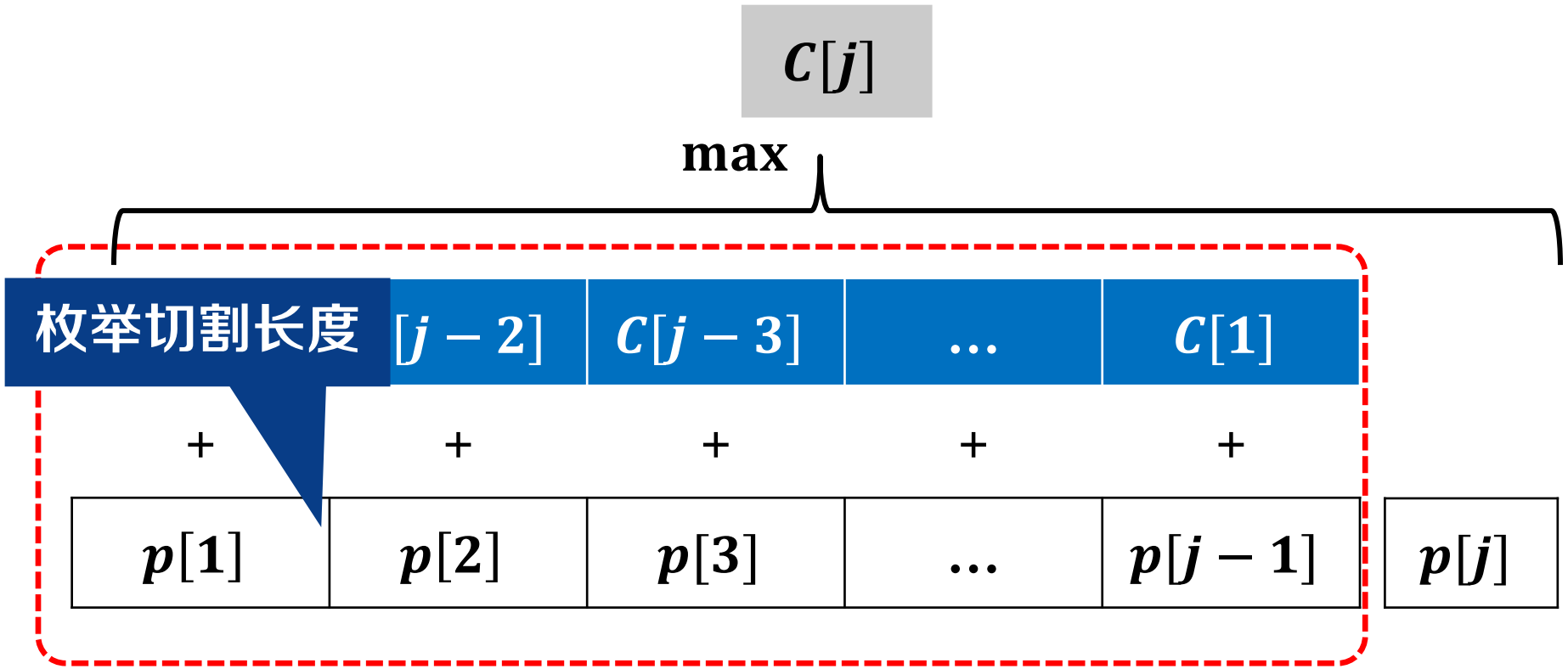
递推关系建立：构造递推公式

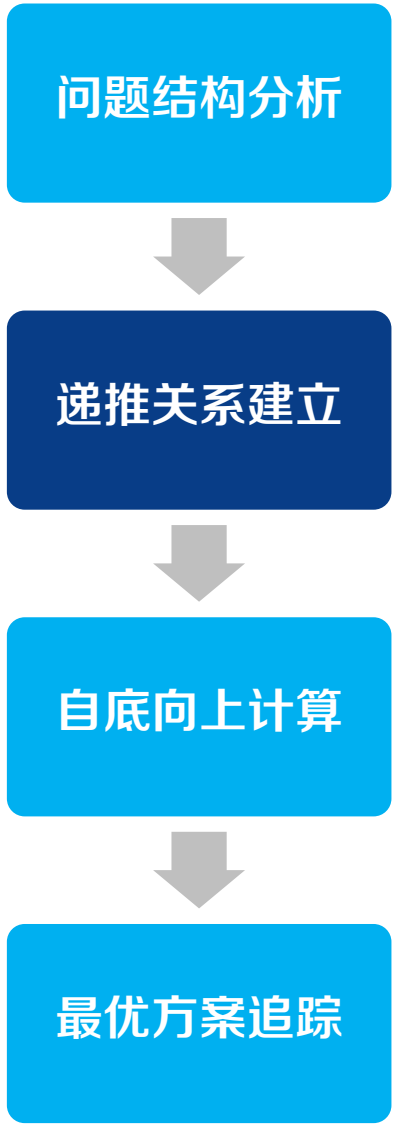
- 对于每个钢条长度j
  - $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$



递推关系建立：构造递推公式

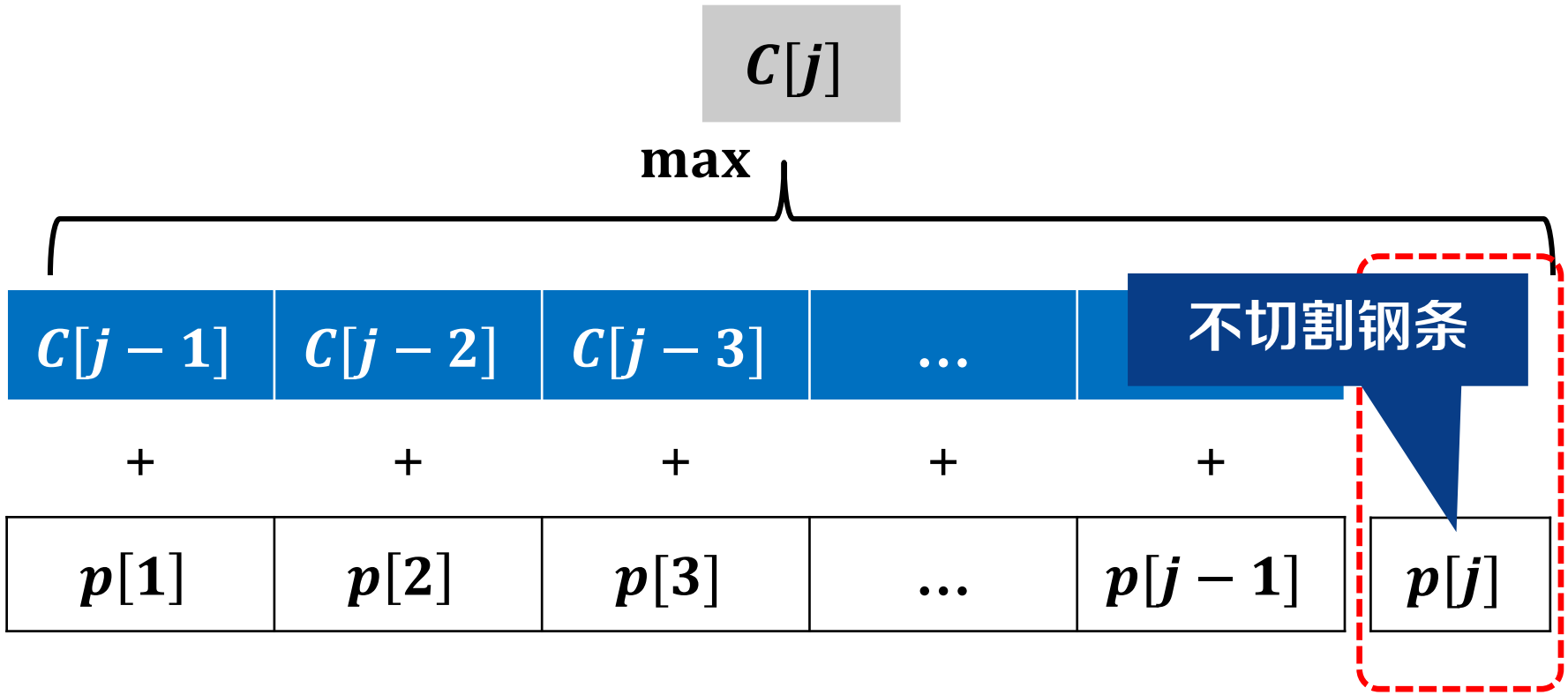
- 对于每个钢条长度j
  - $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$

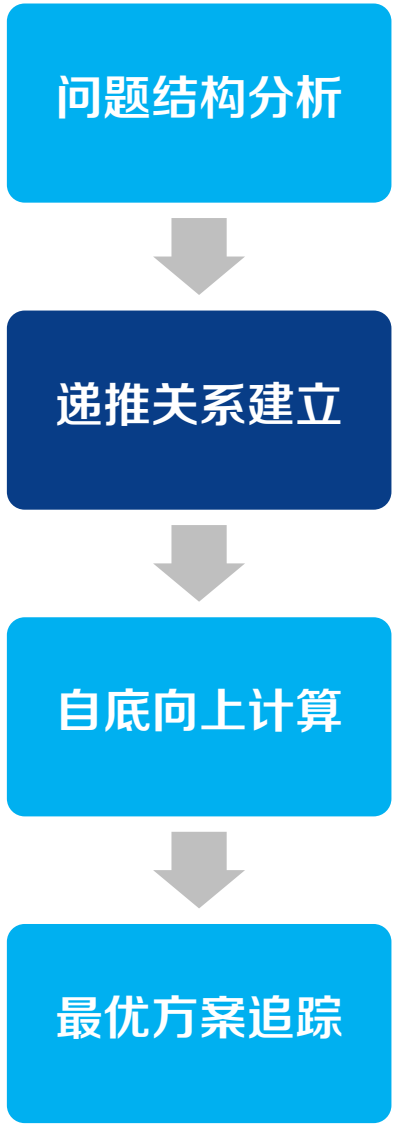




递推关系建立：构造递推公式

- 对于每个钢条长度j
  - $C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$





自底向上计算：确定计算顺序

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

- 初始化
  - $C[0]=0$ 切割长度为0的钢条，总收益为0

初始化

	0	1	2	…	n
$C[\,]$	0				

自底向上计算：确定计算顺序

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

- 初始化
  - $C[0]=0$ 切割长度为0的钢条，总收益为0
- 递推公式
  - $C[j]=\max\limits_{1\le i\le j-1}\{p[i]+C[j-i],p[j]\}$

自底向上计算

	0	1	2	…	n
$C[\,]$	0 →				★

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

初始化

	0	1	2	…	n
$C[\,]$	0 →				

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

$C[1]$

max

$p[1]$

$$C[1] = \max\{p[1]\}$$

	0	1	2	…	n
$C[\,]$	0 →				

问题结构分析

递推关系建立

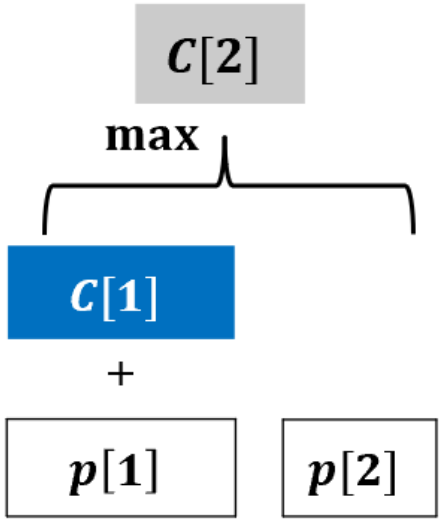
自底向上计算

最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$



$$C[2] = \max\{C[1] + p[1], p[2]\}$$

	0	1	2	…	n
$C[\,]$	0 →				

- 问题结构分析
- 递推关系建立
- 自底向上计算
- 最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

$C[j]$

max

$C[j-1]$	…	$C[1]$
+	+	+
$p[1]$	…	$p[j-1]$

$p[j]$

$$C[j] = \max_{1 \le i \le j-1} \{p[i] + C[j-i], p[j]\}$$

	0	1	2	…	n
$C[\,]$	0 →				

- 问题结构分析
- 递推关系建立
- 自底向上计算
- 最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$

$C[n]$

max

$C[n-1]$	$C[n-2]$	…	$C[1]$
+	+	+	+
$p[1]$	$p[2]$	…	$p[n-1]$	$p[n]$

$$C[n] = \max_{1 \le i \le n-1} \{p[i] + C[n-i], p[n]\}$$

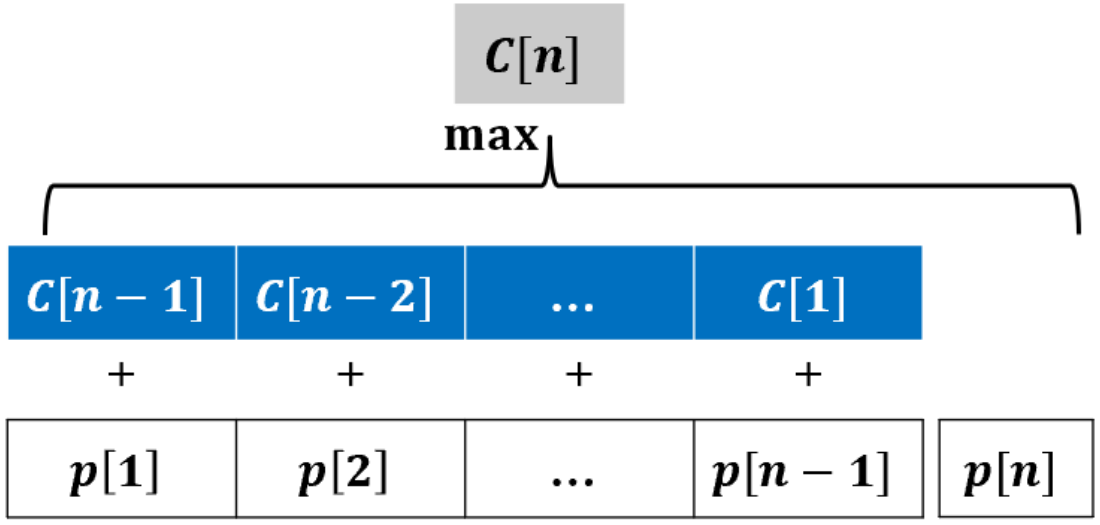
	0	1	2	…	n
$C[\,]$	0 →				

- 问题结构分析
- 递推关系建立
- 自底向上计算
- 最优方案追踪

自底向上计算：依次求解问题

已知钢条价格

$p[\,]$	$p[1]$	$p[2]$	…	$p[n]$



$$C[n] = \max_{1 \le i \le n-1} \{p[i] + C[n-i], p[n]\}$$

	0	1	2	…	n
$C[\,]$	0 →				★

问题结构分析

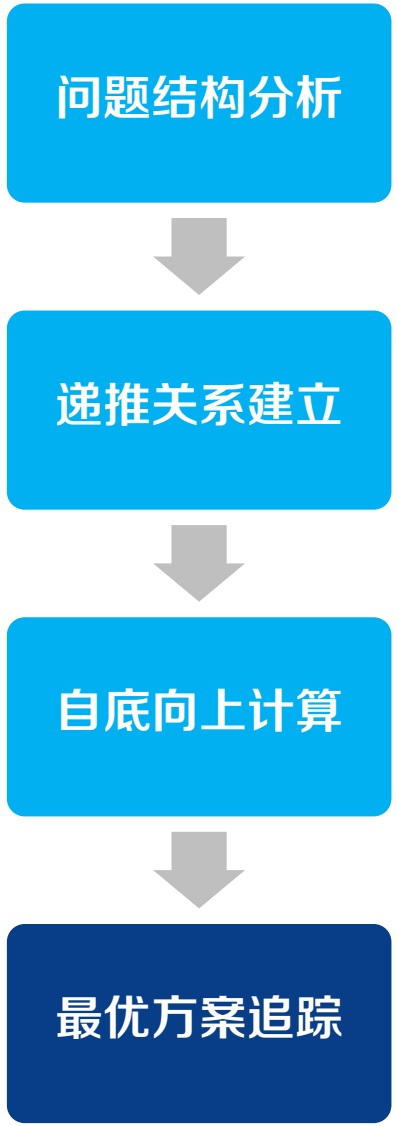
递推关系建立

自底向上计算

最优方案追踪

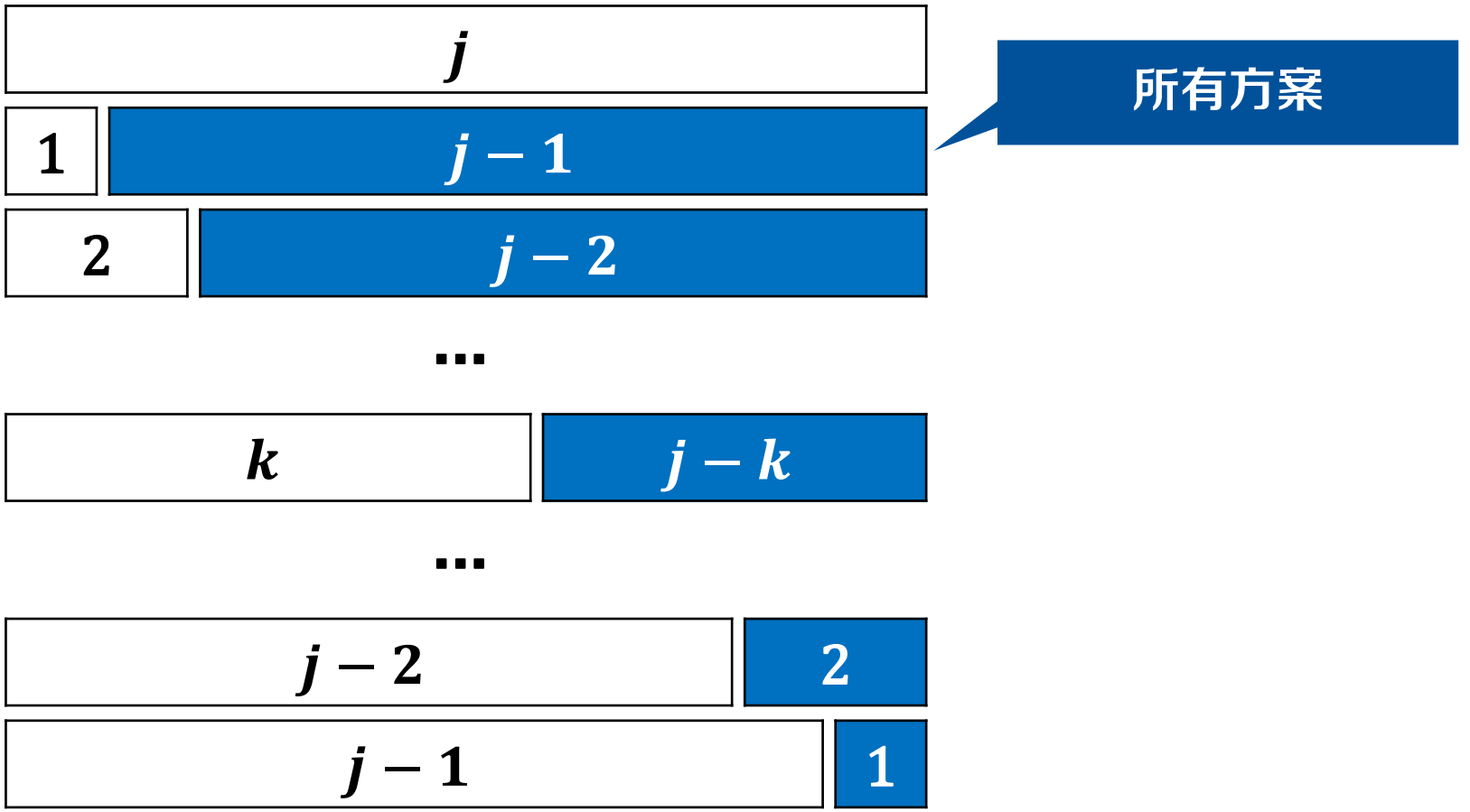
最优方案追踪：记录决策过程

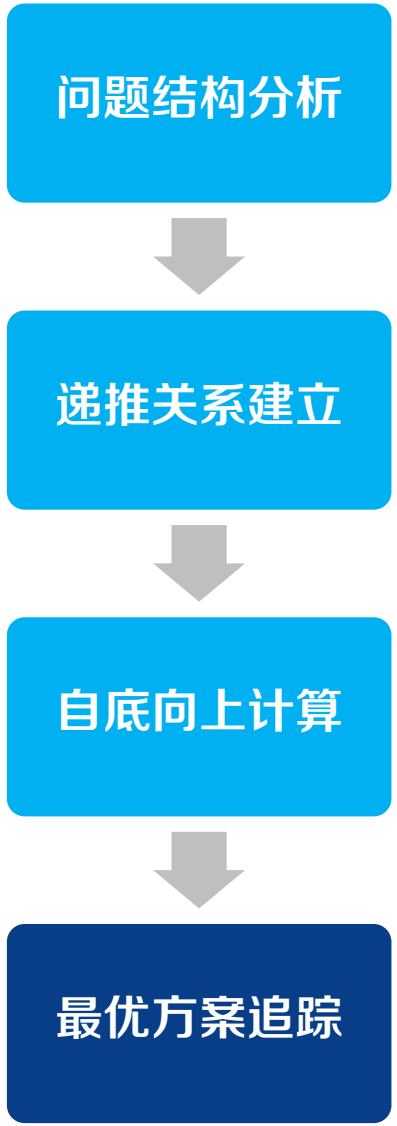
- 构造追踪数组$rec[1..n]$



最优方案追踪：记录决策过程

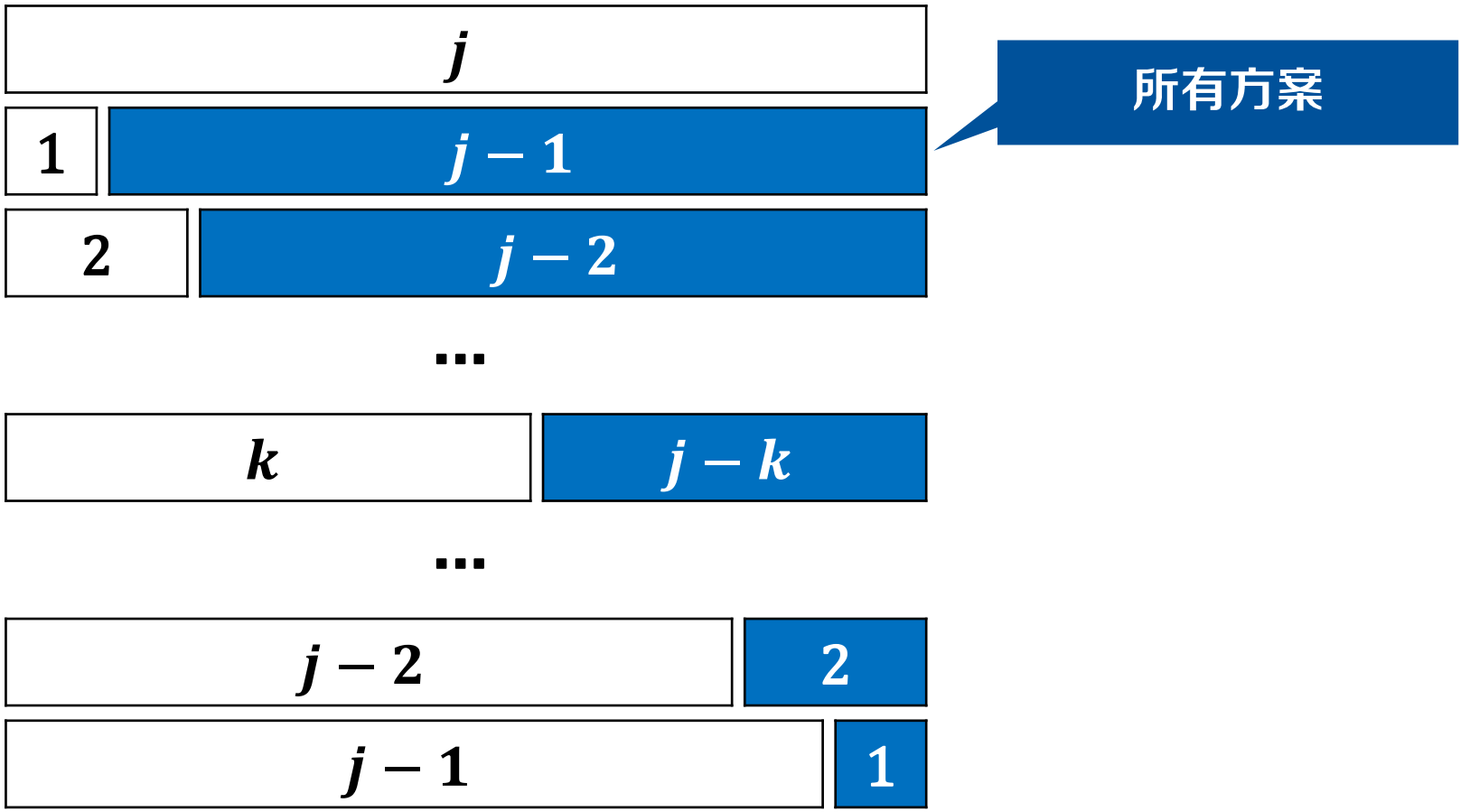
- 构造追踪数组$rec[1..n]$

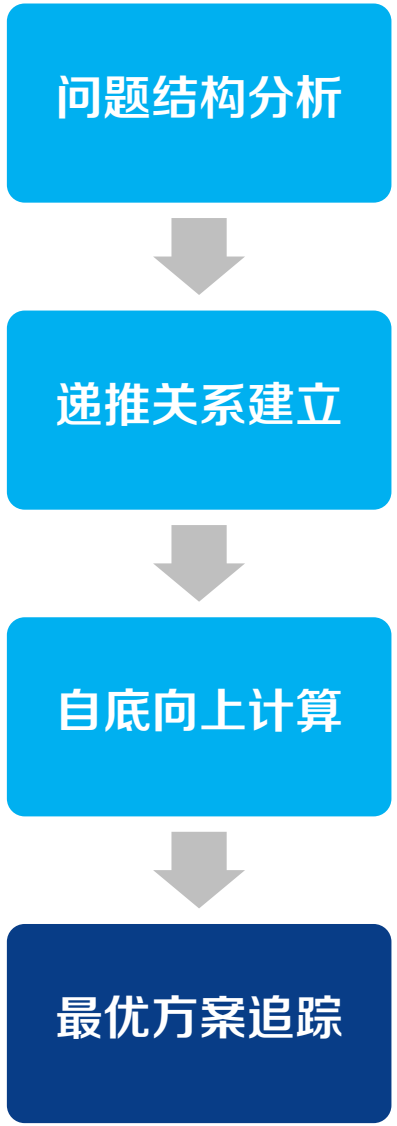




最优方案追踪：记录决策过程

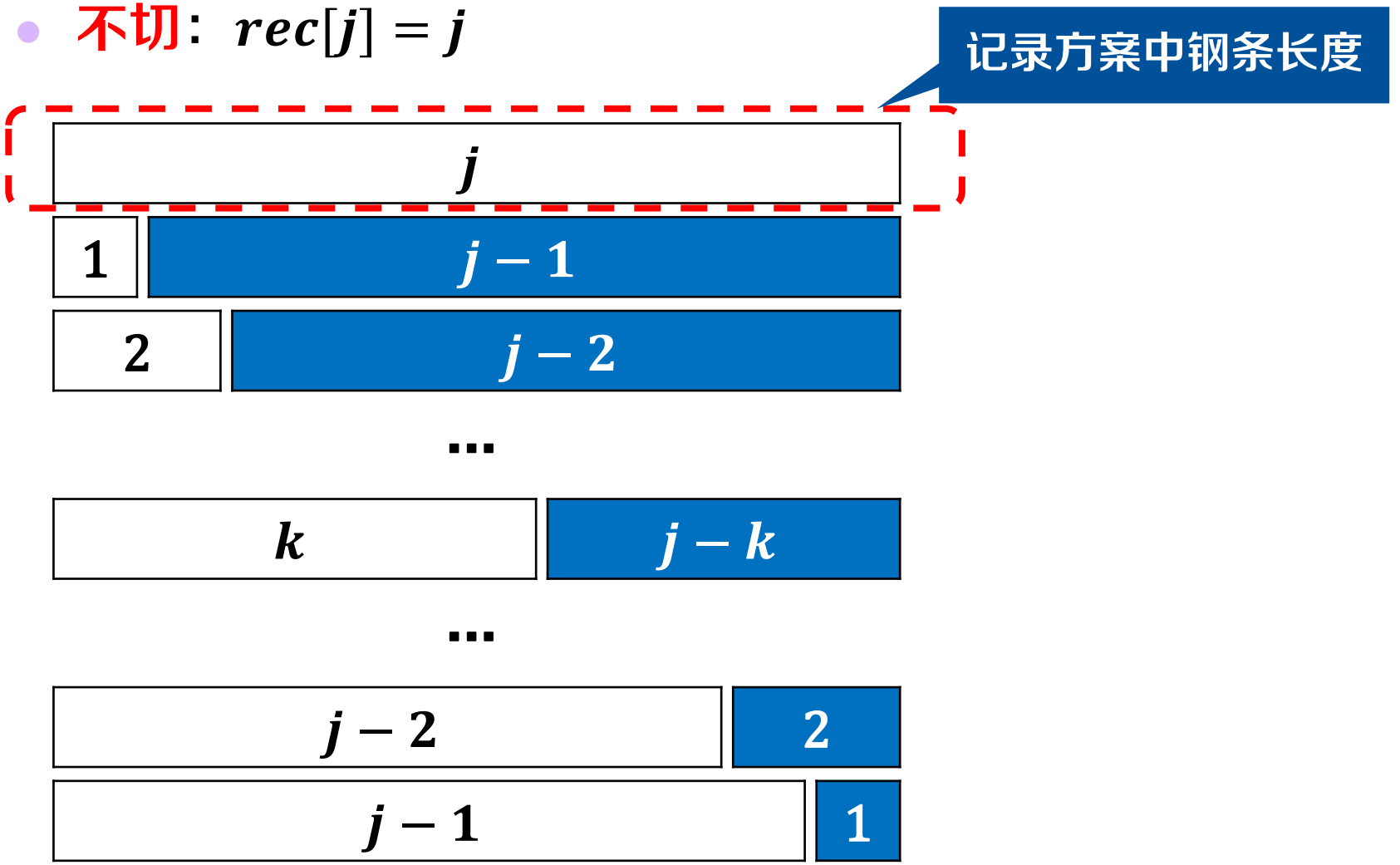
- 构造追踪数组$rec[1..n]$
- $rec[j]$：记录长度为j钢条的最优切割方案

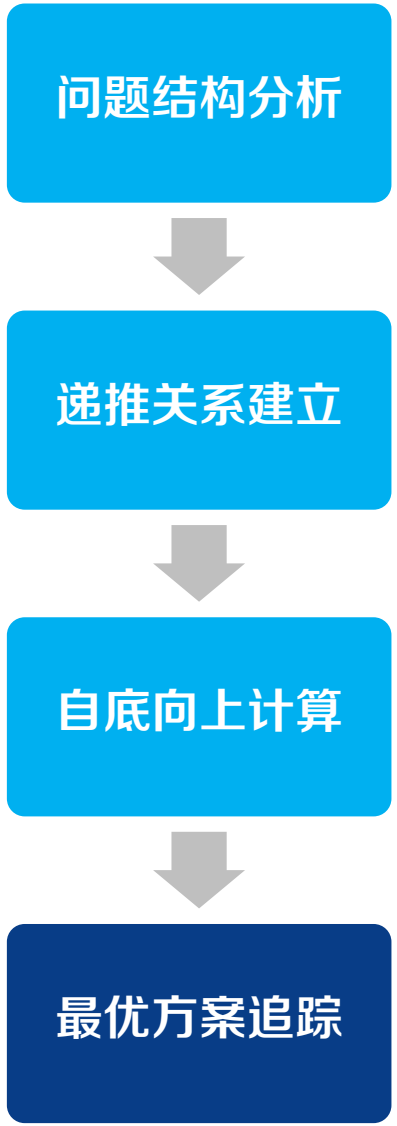




最优方案追踪：记录决策过程

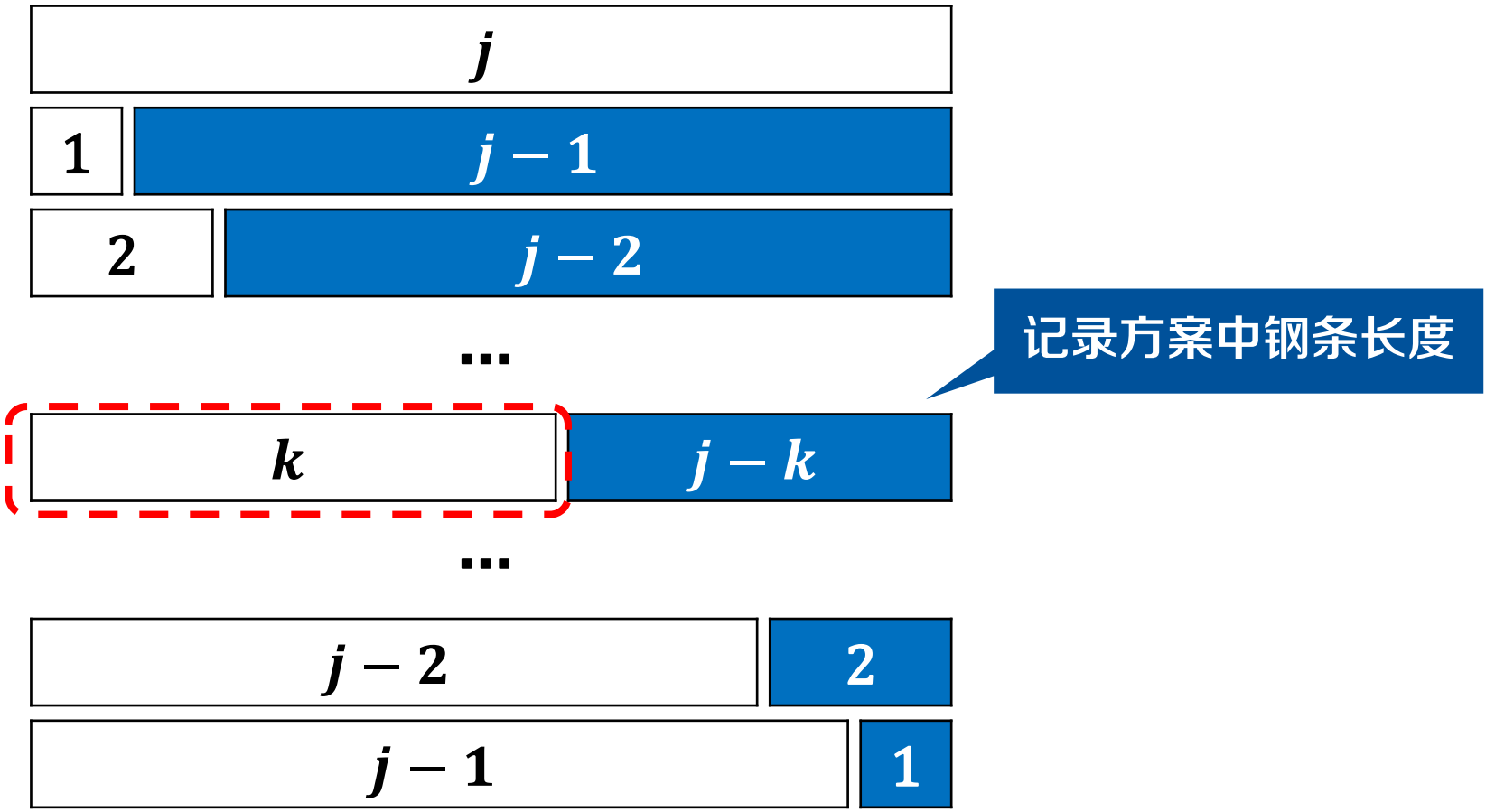
- 构造追踪数组$rec[1..n]$
- $rec[j]$：记录长度为j钢条的最优切割方案
  - 不切：$rec[j]=j$

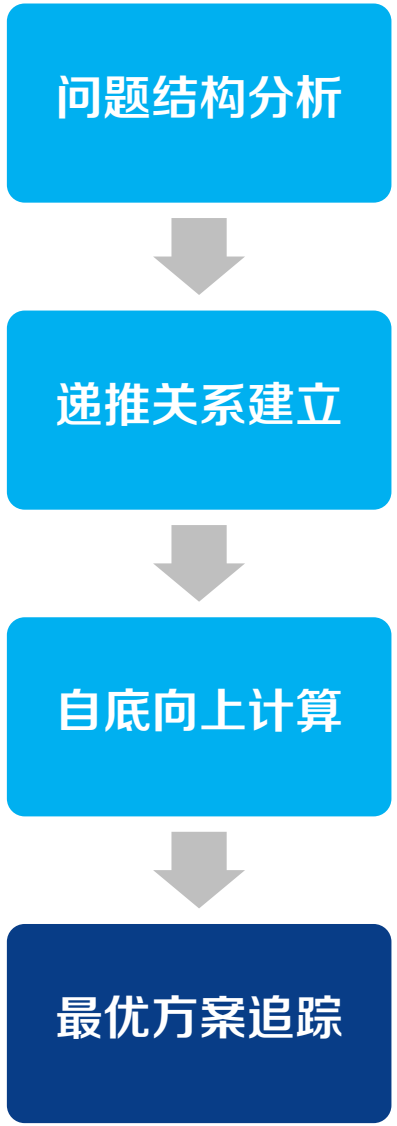




最优方案追踪：记录决策过程

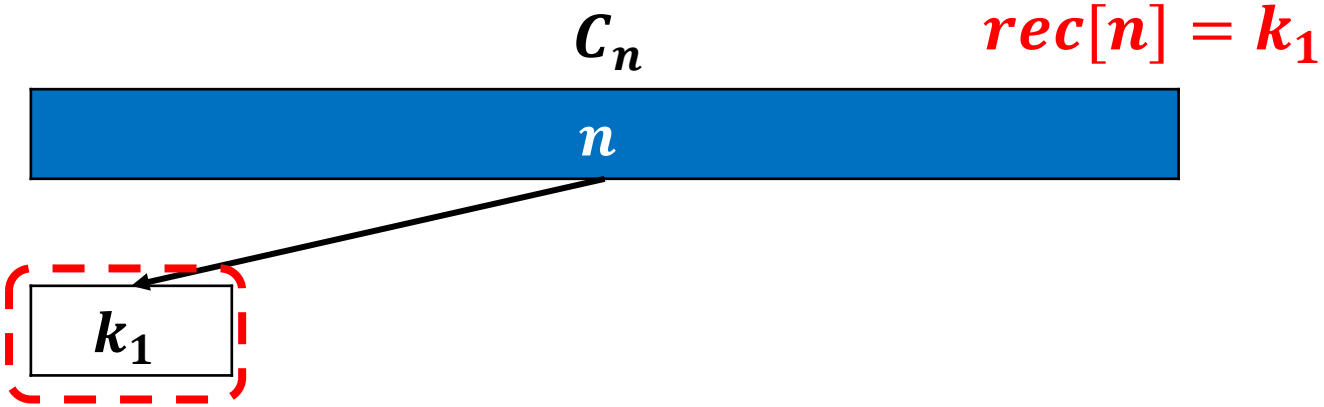
- 构造追踪数组$rec[1..n]$
- $rec[j]$：记录长度为j钢条的最优切割方案
  - 不切：$rec[j]=j$　切割：$rec[j]=k$

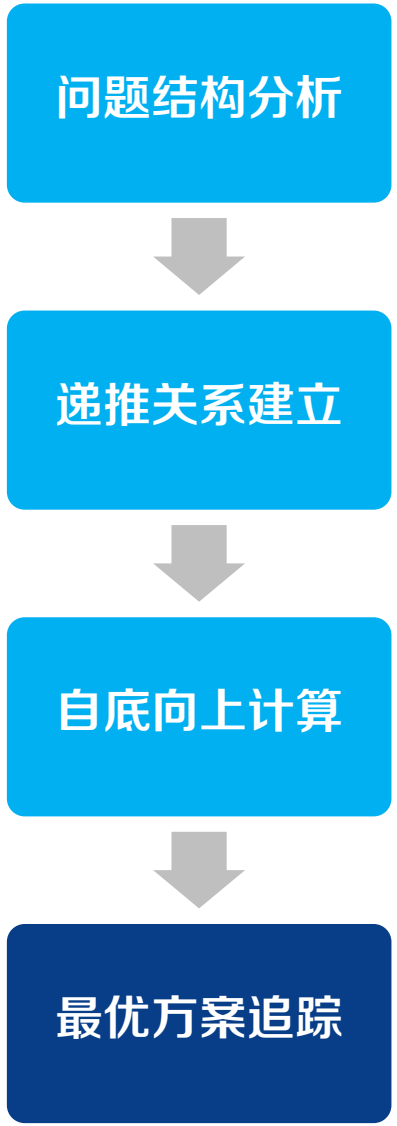




最优方案追踪：输出最优方案

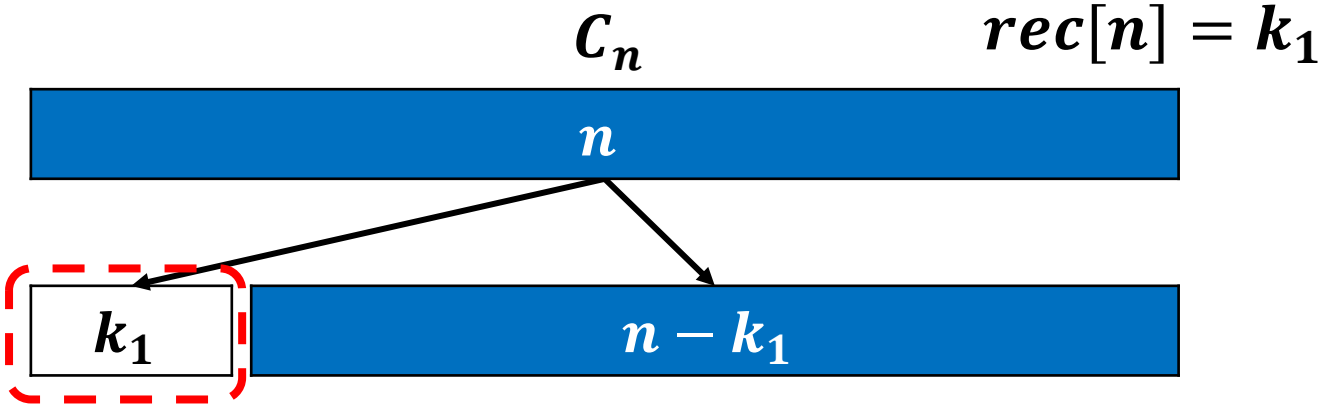
- 根据追踪数组，递归输出方案
  - 输出长度为k_1的钢条

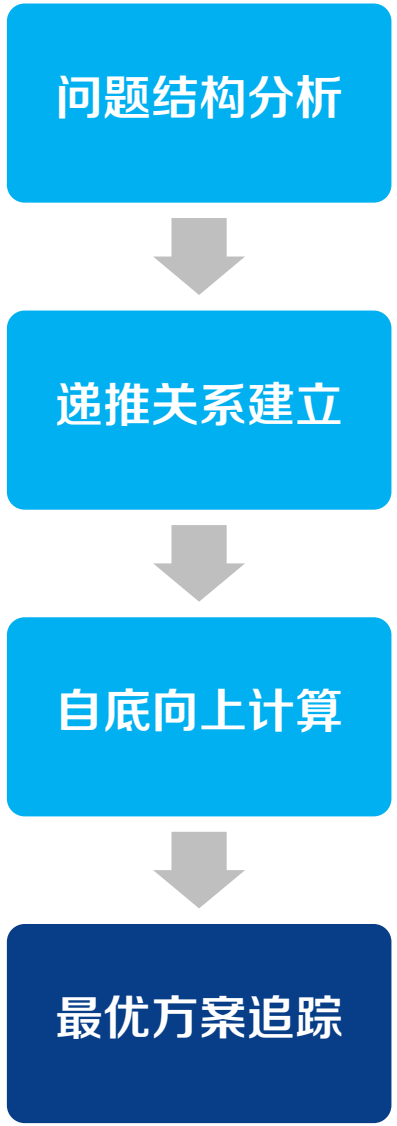




最优方案追踪：输出最优方案

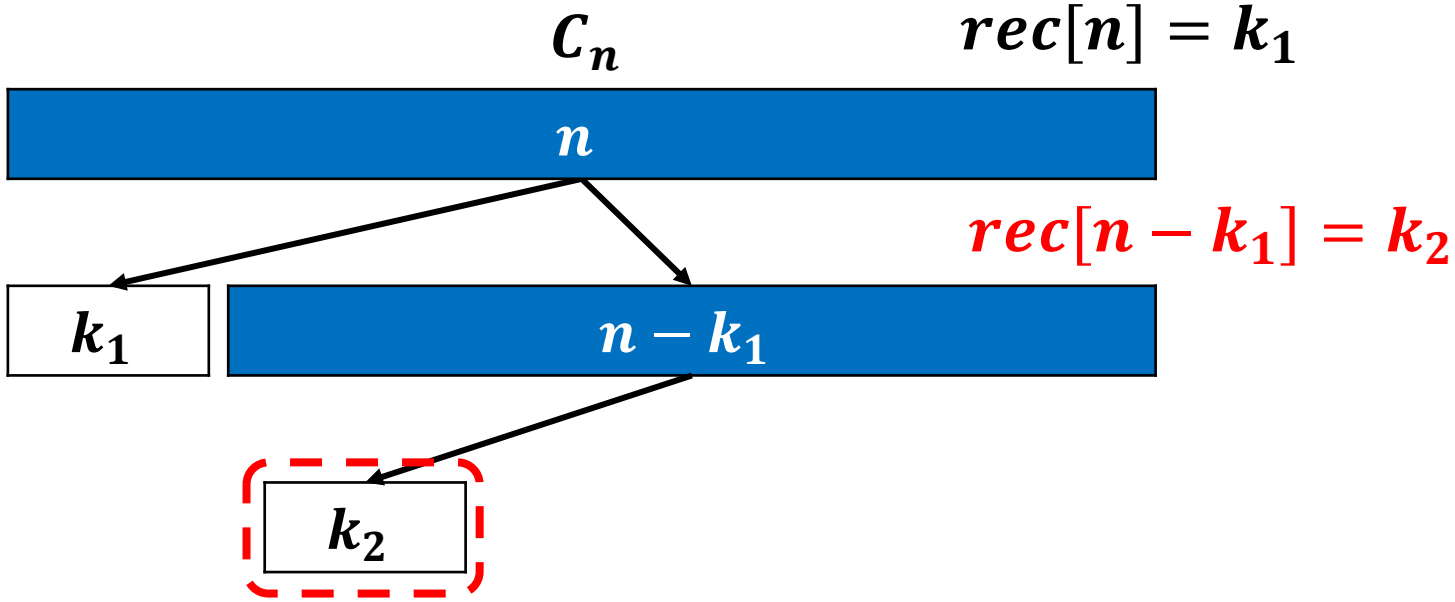
- 根据追踪数组，递归输出方案
  - 输出长度为k_1的钢条

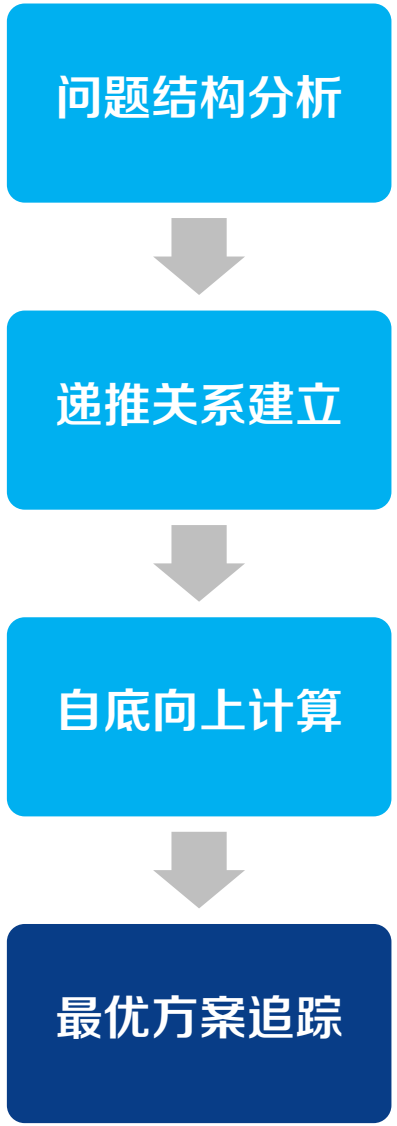




最优方案追踪：输出最优方案

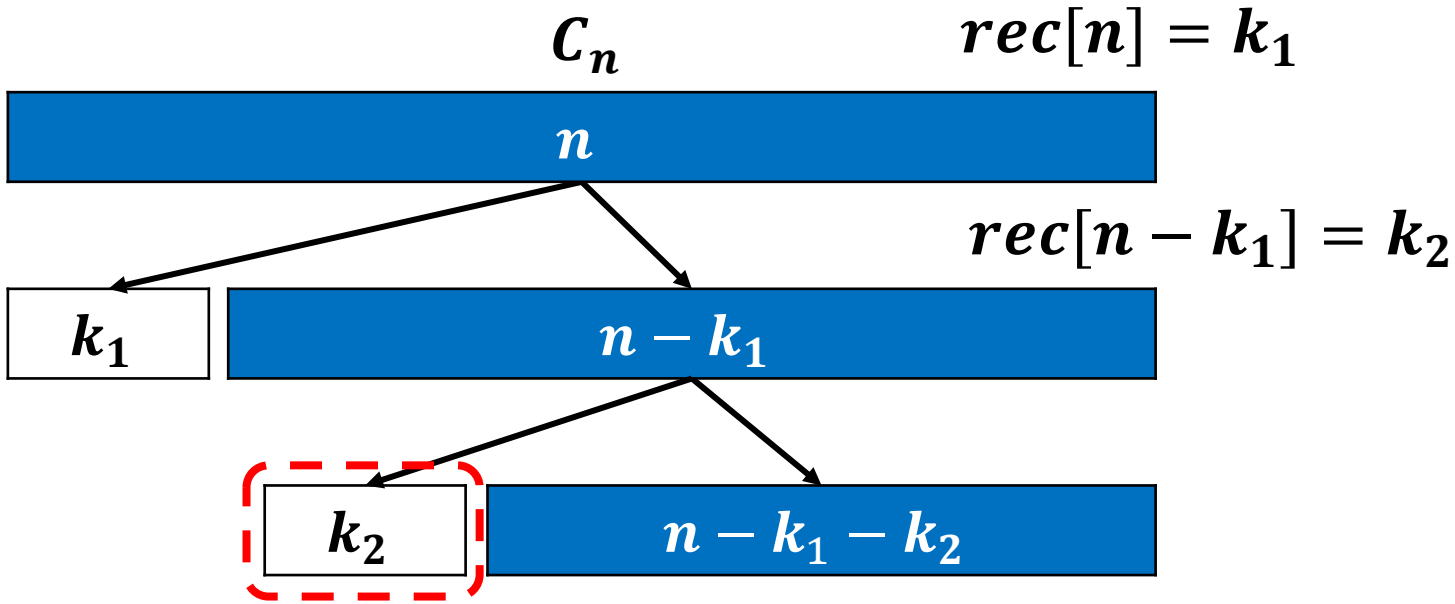
- 根据追踪数组，递归输出方案
  - 输出长度为k_2的钢条

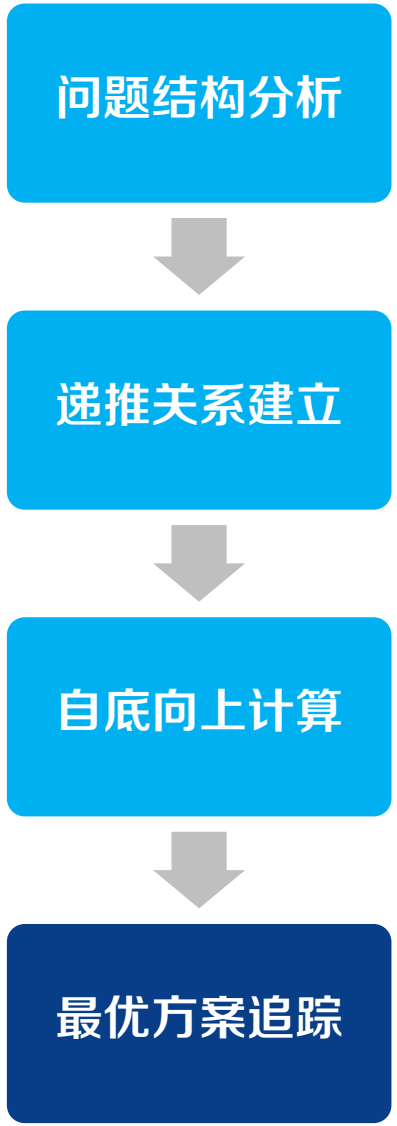




最优方案追踪：输出最优方案

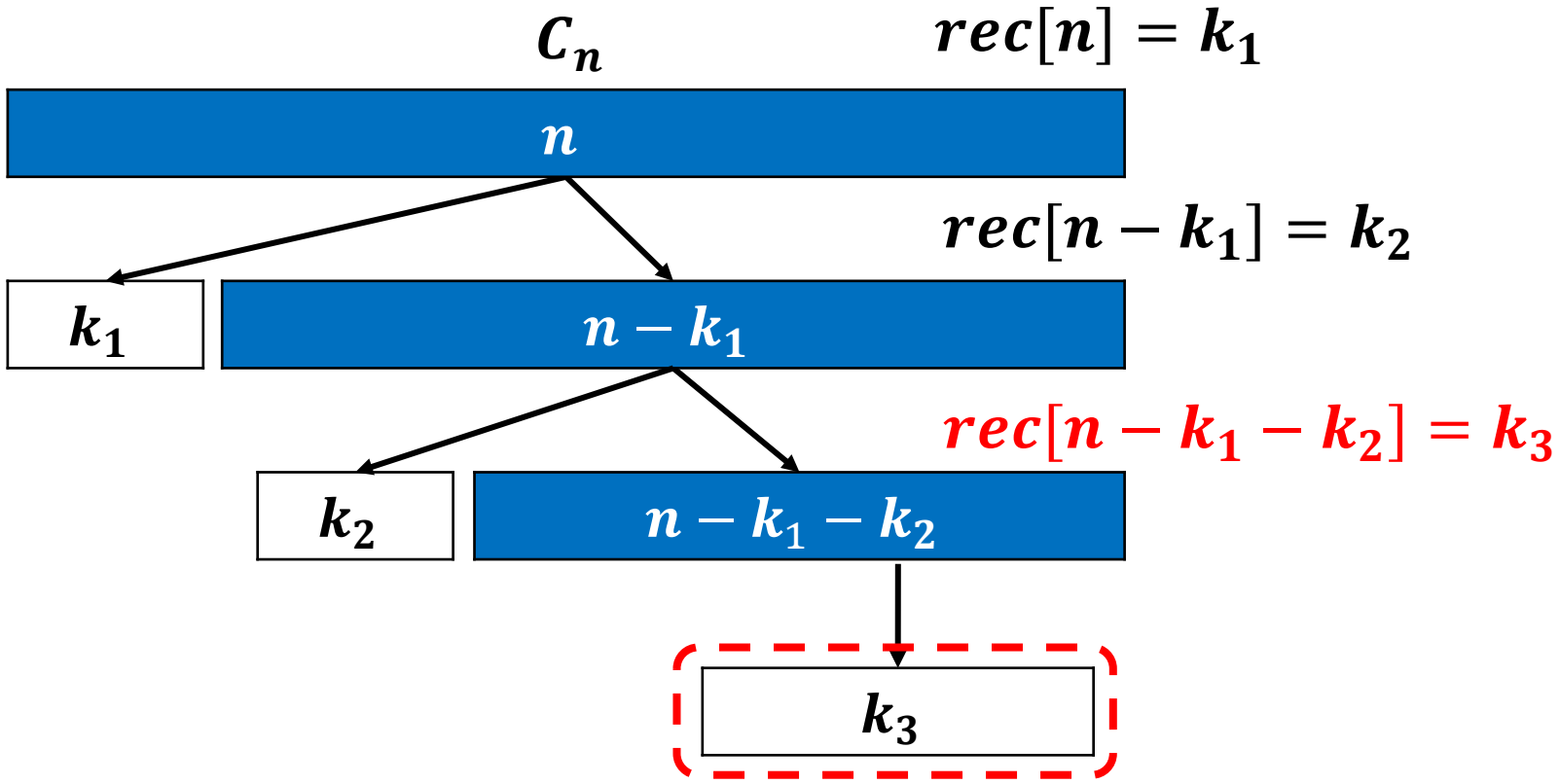
- 根据追踪数组，递归输出方案
  - 输出长度为k_2的钢条





最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案
  - 输出长度为k_3的钢条



问题结构分析

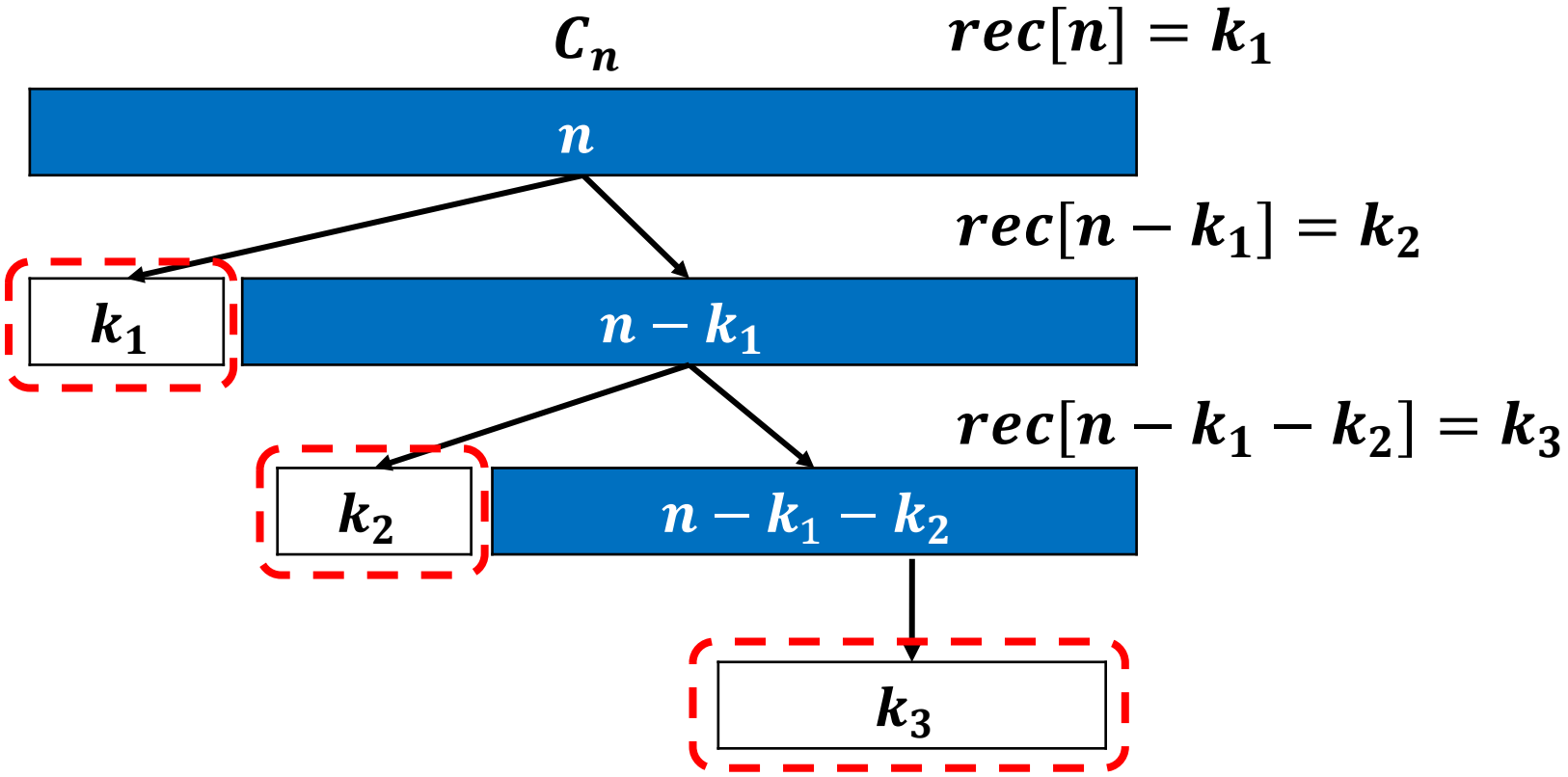
递推关系建立

自底向上计算

最优方案追踪

最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案
  - 递归出口：输出的钢条总长度已达n



$$k_1 + k_2 + k_3 = n$$

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0										
rec	0	1	2	3	4	5	6	7	8	9	10

初始化

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 1$

i	1
	1

$p[j] = p[1]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0										
rec	0	1	2	3	4	5	6	7	8	9	10

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 1$

i	1
	1

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0										
rec	0	1	2	3	4	5	6	7	8	9	10

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 1$

i	1
	1

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1									
rec	0	1	2	3	4	5	6	7	8	9	10
		1									

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 2$

i	1
	2

$p[1] + C[1]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1									
rec	0	1	2	3	4	5	6	7	8	9	10
		1									

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 2$

i	1	2
	2	5

$p[j] = p[2]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1									
rec	0	1	2	3	4	5	6	7	8	9	10
		1									

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 2$

i	1	2
	2	5

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1									
rec	0	1	2	3	4	5	6	7	8	9	10
		1									

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 2$

i	1	2
	2	5

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5								
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2								

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 3$

i	1
	6

$p[1] + C[2]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5								
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2								

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 3$

i	1	2
	6	6

$p[2] + C[1]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5								
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2								

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 3$

i	1	2	3
	6	6	8

$p[j] = p[3]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5								
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2								

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 3$

i	1	2	3
	6	6	8

$$\max\{p[i] + C[j-i], p[j]\}$$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5								
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2								

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 3$

i	1	2	3
	6	6	8

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

i	1
	9

$p[1] + C[3]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

i	1	2
	9	10

$p[2] + C[2]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

i	1	2	3
	9	10	9

$p[3] + C[1]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

i	1	2	3	4
	9	10	9	9

$p[j] = p[4]$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4
	9	10	9	9

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8							
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3							

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 4$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4
	9	10	9	9

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10						
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2						

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 5$

i	1	2	3	4	5
	11	13	13	10	10

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10						
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2						

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 5$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5
	11	13	13	10	10

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10						
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2						

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 5$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5
	11	13	13	10	10

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13					
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2					

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 6$

i	1	2	3	4	5	6
	14	15	16	14	11	17

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13					
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2					

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 6$

i	1	2	3	4	5	6
	14	15	16	14	11	17

$$\max\{p[i] + C[j-i], p[j]\}$$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13					
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2					

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 6$

i	1	2	3	4	5	6
	14	15	16	14	11	17

$\max\{p[i] + C[j-i], p[j]\}$

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17				
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6				

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 7$

i	1	2	3	4	5	6	7
	18	18	18	17	15	18	17

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17				
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6				

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 7$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7
	18	18	18	17	15	18	17

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17				
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6				

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 7$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7
	18	18	18	17	15	18	17

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18			
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1			

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 8$

i	1	2	3	4	5	6	7	8
	19	22	21	19	18	22	18	20

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18			
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1			

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 8$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8
	19	22	21	19	18	22	18	20

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18			
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1			

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 8$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8
	19	22	21	19	18	22	18	20

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22		
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2		

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 9$

i	1	2	3	4	5	6	7	8	9
	23	23	25	22	20	25	22	21	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22		
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2		

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 9$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8	9
	23	23	25	22	20	25	22	21	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22		
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2		

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 9$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8	9
	23	23	25	22	20	25	22	21	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

$\max\{p[i] + C[j-i], p[j]\}$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

最大收益$= C[10] = 27$

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

最大收益$= C[10] = 27$

切割方案$= \{2,$

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

最大收益$= C[10] = 27$

切割方案$= \{2, 2$

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

最大收益$= C[10] = 27$

切割方案$= \{2, 2, 6$

算法实例

$n = 10$

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	24

$j = 10$

i	1	2	3	4	5	6	7	8	9	10
	26	27	26	26	23	27	25	25	25	24

i	0	1	2	3	4	5	6	7	8	9	10
$C[i]$	0	1	5	8	10	13	17	18	22	25	27
rec	0	1	2	3	4	5	6	7	8	9	10
		1	2	3	2	2	6	1	2	3	2

最大收益$= C[10] = 27$

切割方案$= \{2, 2, 6\}$

伪代码

- **RodCutting(p, n)**

```
输入: 钢条价格表p[1..n],钢条长度n
输出: 最大收益C[n],钢条切割方案
//初始化
新建一维数组C[0..n], rec[0..n]
C[0] ← 0
//动态规划
for j ← 1 to n do
    q ← p[j]
    rec[j] ← j
    for i ← 1 to j − 1 do
        if q < p[i] + C[j − i] then
            q ← p[i] + C[j − i]
            rec[j] ← i
        end
    end
    C[j] ← q
end
```

初始化

伪代码

- **RodCutting(p, n)**

```
输入: 钢条价格表p[1..n],钢条长度n
输出: 最大收益C[n],钢条切割方案
//初始化
新建一维数组C[0..n], rec[0..n]
C[0] ← 0
//动态规划
for j ← 1 to n do
    q ← p[j]
    rec[j] ← j
    for i ← 1 to j − 1 do
        if q < p[i] + C[j − i] then
            q ← p[i] + C[j − i]
            rec[j] ← i
        end
    end
    C[j] ← q
end
```

依次计算子问题

伪代码

- **RodCutting(p, n)**

输入: 钢条价格表$p[1..n]$,钢条长度n
输出: 最大收益$C[n]$,钢条切割方案
//初始化
新建一维数组$C[0..n], rec[0..n]$
$C[0] \leftarrow 0$
//动态规划
for $j \leftarrow 1$ *to* n **do**
  $q \leftarrow p[j]$
  $rec[j] \leftarrow j$
  for $i \leftarrow 1$ *to* $j-1$ **do**
    if $q < p[i] + C[j-i]$ **then**
      $q \leftarrow p[i] + C[j-i]$
      $rec[j] \leftarrow i$
    end
  end
  $C[j] \leftarrow q$
end

不切割钢条

伪代码

- **RodCutting(p, n)**

输入: 钢条价格表$p[1..n]$,钢条长度n
输出: 最大收益$C[n]$,钢条切割方案
//初始化
新建一维数组$C[0..n], rec[0..n]$
$C[0] \leftarrow 0$
//动态规划
for $j \leftarrow 1$ *to* n **do**
  $q \leftarrow p[j]$
  $rec[j] \leftarrow j$
  for $i \leftarrow 1$ *to* $j-1$ **do**
    if $q < p[i] + C[j-i]$ **then**
      $q \leftarrow p[i] + C[j-i]$
      $rec[j] \leftarrow i$
    end
  end
  $C[j] \leftarrow q$
end

枚举切割长度

伪代码

- **RodCutting(p, n)**

输入: 钢条价格表$p[1..n]$,钢条长度n
输出: 最大收益$C[n]$,钢条切割方案
//初始化
新建一维数组$C[0..n], rec[0..n]$
$C[0] \leftarrow 0$
//动态规划
for $j \leftarrow 1$ *to* n **do**
  $q \leftarrow p[j]$
  $rec[j] \leftarrow j$
  for $i \leftarrow 1$ *to* $j-1$ **do**
    if $q < p[i] + C[j-i]$ **then**
      $q \leftarrow p[i] + C[j-i]$
      $rec[j] \leftarrow i$
    end
  end
  $C[j] \leftarrow q$
end

记录价格和决策

伪代码

- **RodCutting(p, n)**

输入: 钢条价格表$p[1..n]$,钢条长度n
输出: 最大收益$C[n]$,钢条切割方案
//初始化
新建一维数组$C[0..n], rec[0..n]$
$C[0] \leftarrow 0$
//动态规划
for $j \leftarrow 1$ *to* n **do**
  $q \leftarrow p[j]$
  $rec[j] \leftarrow j$
  for $i \leftarrow 1$ *to* $j-1$ **do**
    if $q < p[i] + C[j-i]$ **then**
      $q \leftarrow p[i] + C[j-i]$
      $rec[j] \leftarrow i$
    end
  end
  $C[j] \leftarrow q$
end

保存子问题的解

伪代码

- **RodCutting(p, n)**

```
//输出最优方案
while n > 0 do
    print rec[n]
    n ← n − rec[n]
end
```

追踪切割方案

时间复杂度分析

- **RodCutting(p, n)**

```
输入: 钢条价格表p[1..n],钢条长度n
输出: 最大收益C[n],钢条切割方案
//初始化
新建一维数组C[0..n], rec[0..n]
C[0] ← 0
//动态规划
for j ← 1 to n do
    q ← p[j]
    rec[j] ← j
    for i ← 1 to j − 1 do
        if q < p[i] + C[j − i] then
            q ← p[i] + C[j − i]
            rec[j] ← i
        end
    end
    C[j] ← q
end
```

时间复杂度：$O(n^2)$

动态规划篇：矩阵链乘法问题

问题背景：基本知识

- 矩阵
  - $p \times q$的矩阵$U_{p,q}$
  - 例
    - 4×3的矩阵$U_{4,3}$

$$U = \begin{bmatrix} 2 & 1 & 3 \\ 9 & 5 & 6 \\ 0 & 8 & 1 \\ 5 & 2 & 7 \end{bmatrix}$$

    - 3×2的矩阵$V_{3,2}$

$$V = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

问题背景：基本知识

- 矩阵
  - $p \times q$的矩阵$U_{p,q}$
  - 例
    - 4×3的矩阵$U_{4,3}$

$$U = \underbrace{\begin{bmatrix} 2 & 1 & 3 \\ 9 & 5 & 6 \\ 0 & 8 & 1 \\ 5 & 2 & 7 \end{bmatrix}}_{q=3} \Big\} p = 4$$

    - 3×2的矩阵$V_{3,2}$

$$V = \underbrace{\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}}_{r=2} \Big\} q = 3$$

问题背景：基本知识

- 矩阵
  - $p \times q$的矩阵$U_{p,q}$
  - 例
    - 4×3的矩阵$U_{4,3}$

$$U = \underbrace{\begin{bmatrix} 2 & 1 & 3 \\ 9 & 5 & 6 \\ 0 & 8 & 1 \\ 5 & 2 & 7 \end{bmatrix}}_{q=3} \Big\} p = 4$$

    - 3×2的矩阵$V_{3,2}$

$$V = \underbrace{\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}}_{r=2} \Big\} q = 3$$

问题：如何计算$U \cdot V$?

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}}_{q=3},\ p=4 \qquad V=\underbrace{\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}}_{r=2},\ q=3 \qquad Z=UV=\begin{bmatrix}\ & \\ \ & \end{bmatrix}$$

问题背景：基本知识

- 2个矩阵相乘

- $U = \begin{bmatrix} 2 & 1 & 3 \\ 9 & 5 & 6 \\ 0 & 8 & 1 \\ 5 & 2 & 7 \end{bmatrix}$，$p = 4$，$q = 3$　$V = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$，$q = 3$，$r = 2$　$Z = UV = \begin{bmatrix} 13 & \\ & \\ & \\ & \end{bmatrix}$

- $2 \times 1 + 1 \times 2 + 3 \times 3 = 13$

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\begin{bmatrix}13 & 31\\ & \end{bmatrix}$$

- $2\times 4+1\times 5+3\times 6=31$

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\begin{bmatrix}13 & 31\\ 37 & \\ & \end{bmatrix}$$

- $9\times1+5\times2+6\times3=37$

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\begin{bmatrix}13 & 31\\ 37 & 97\\ 19 & 46\\ 30 & 72\end{bmatrix}$$

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2&1&3\\9&5&6\\0&8&1\\5&2&7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\underbrace{\begin{bmatrix}13&31\\37&97\\19&46\\30&72\end{bmatrix}}_{r=2}\Big\}p=4$$

问题背景：基本知识

- 2个矩阵相乘
  - $U=\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}$，$p=4$　$V=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$，$q=3$　$Z=UV=\begin{bmatrix}13 & 31\\ 37 & 97\\ 19 & 46\\ 30 & 72\end{bmatrix}$，$p=4$

  $q=3$　　$r=2$　　$r=2$

- 矩阵乘法的时间复杂度
  - 计算1个数字：q次标量乘法

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2&1&3\\9&5&6\\0&8&1\\5&2&7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1&4\\2&5\\3&6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\underbrace{\begin{bmatrix}13&31\\37&97\\19&46\\30&72\end{bmatrix}}_{r=2}\Big\}p=4$$

- 矩阵乘法的时间复杂度
  - 计算1个数字：q次标量乘法
  - 共$p\times r$个数：$\Theta(pqr)$

问题背景：基本知识

- 2个矩阵相乘

$$U=\underbrace{\begin{bmatrix}2 & 1 & 3\\ 9 & 5 & 6\\ 0 & 8 & 1\\ 5 & 2 & 7\end{bmatrix}}_{q=3}\Big\}p=4 \quad V=\underbrace{\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}}_{r=2}\Big\}q=3 \quad Z=UV=\underbrace{\begin{bmatrix}13 & 31\\ 37 & 97\\ 19 & 46\\ 30 & 72\end{bmatrix}}_{r=2}\Big\}p=4$$

- 矩阵乘法的时间复杂度
  - 计算1个数字：q次标量乘法
  - 共$p\times r$个数：$\Theta(pqr)$
  - 上例中，标量乘法次数为：$p\times q\times r=4\times 3\times 2=24$

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(\boldsymbol{UV})\boldsymbol{W} = \boldsymbol{U}(\boldsymbol{VW})$
  - 新问题：矩阵乘法结合的顺序

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(\boldsymbol{UV})\boldsymbol{W} = \boldsymbol{U}(\boldsymbol{VW})$
  - 新问题：矩阵乘法结合的顺序

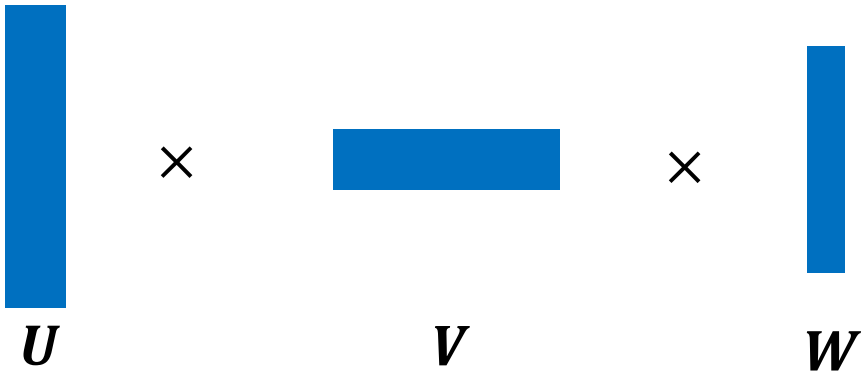
问题：顺序不同，效率是否明显不同?

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$

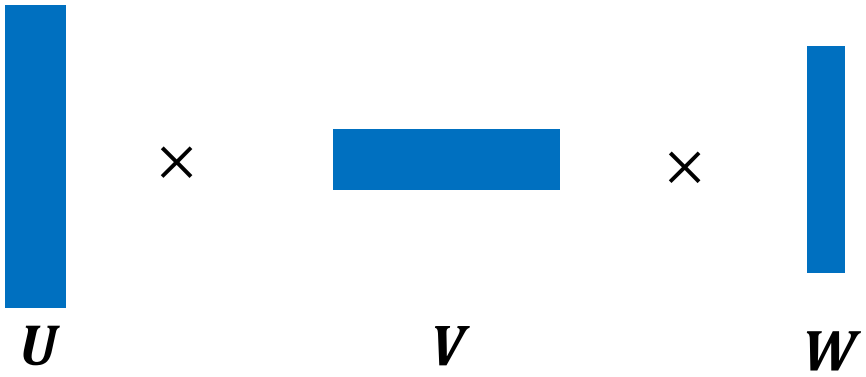


问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



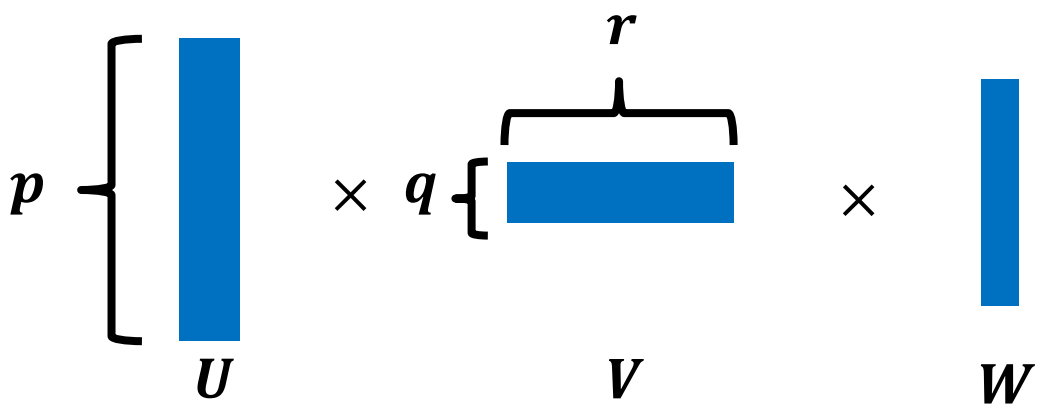
  - 按$(UV)W$计算，标量乘法次数：

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



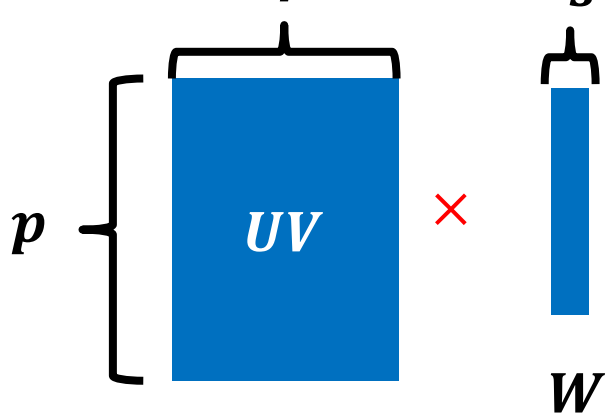
  - 按$(UV)W$计算，标量乘法次数：pqr

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



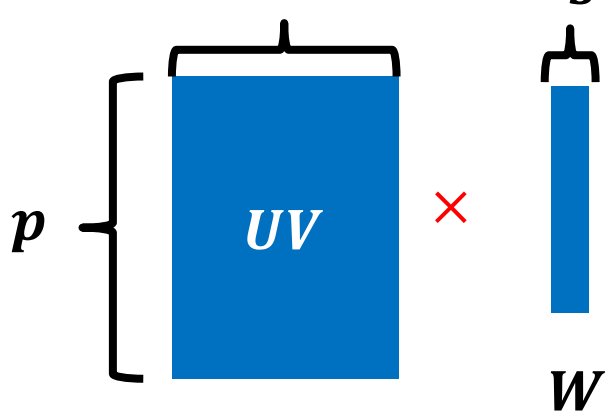
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs$

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



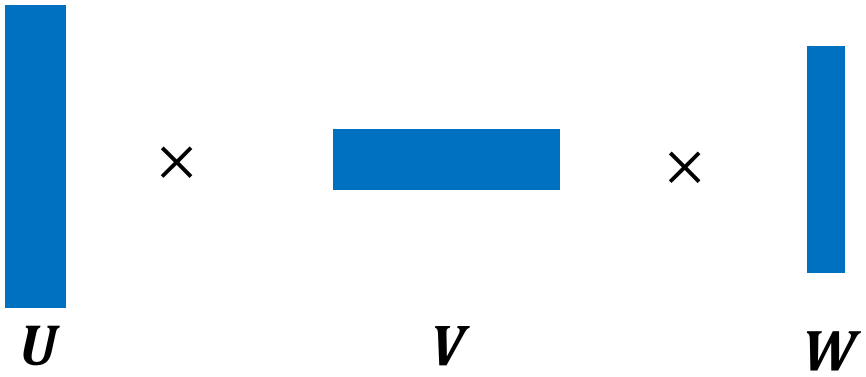
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



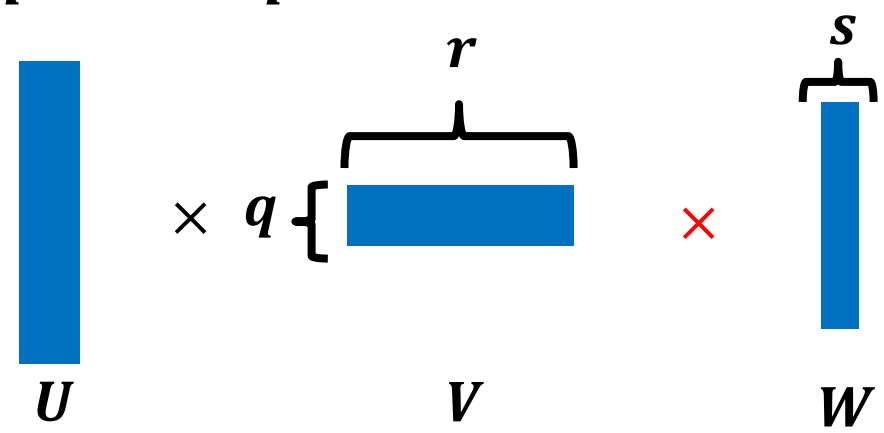
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$
  - 按$U(VW)$计算，标量乘法次数：

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



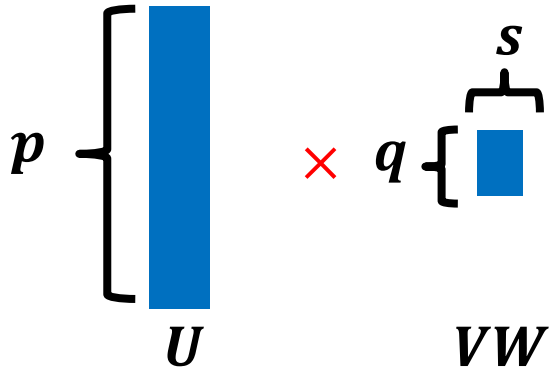
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$
  - 按$U(VW)$计算，标量乘法次数：qrs

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



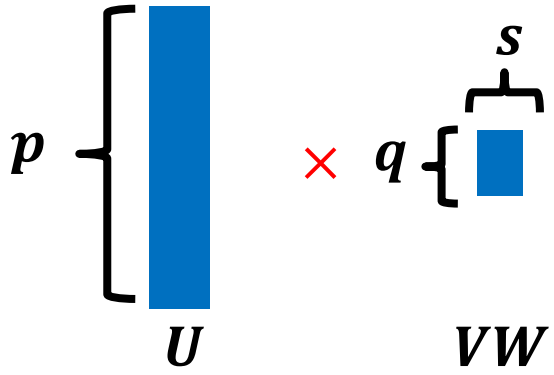
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$
  - 按$U(VW)$计算，标量乘法次数：$qrs + pqs$

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$



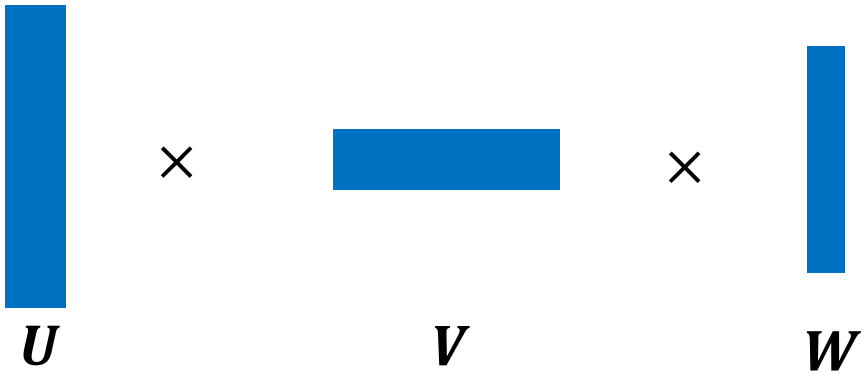
  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$
  - 按$U(VW)$计算，标量乘法次数：$qrs + pqs = 2800$

问题背景：基本知识

- 3个矩阵相乘
  - 矩阵乘法结合率：$(UV)W = U(VW)$
  - 新问题：矩阵乘法结合的顺序

问题：顺序不同，效率是否明显不同?

  - 例如：矩阵维度数为$p = 40,\ q = 8,\ r = 30,\ s = 5$

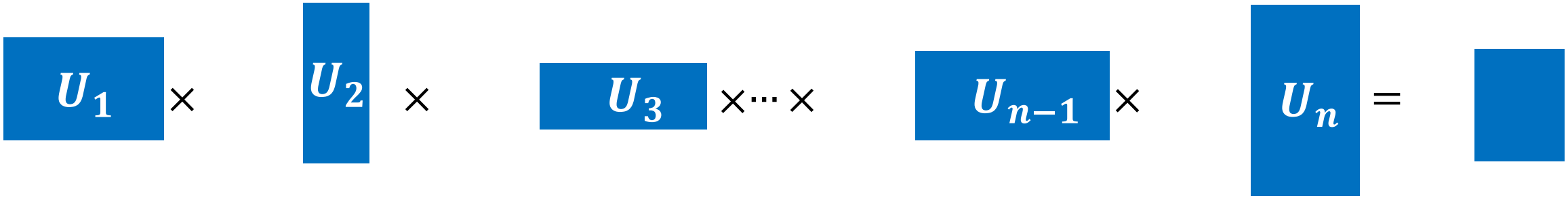


  - 按$(UV)W$计算，标量乘法次数：$pqr + prs = 15600$
  - 按$U(VW)$计算，标量乘法次数：$qrs + pqs = 2800$

差异显著

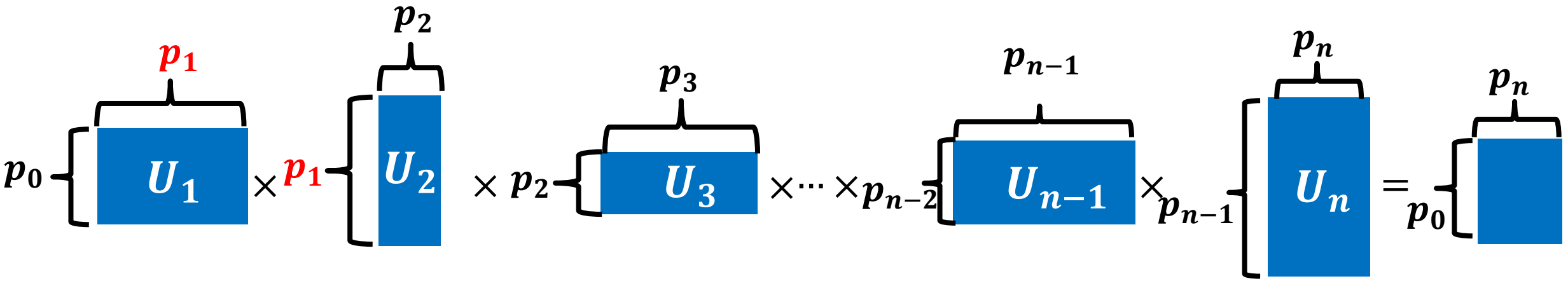
问题背景：基本知识

- n个矩阵相乘
  - 有一系列矩阵按顺序排列



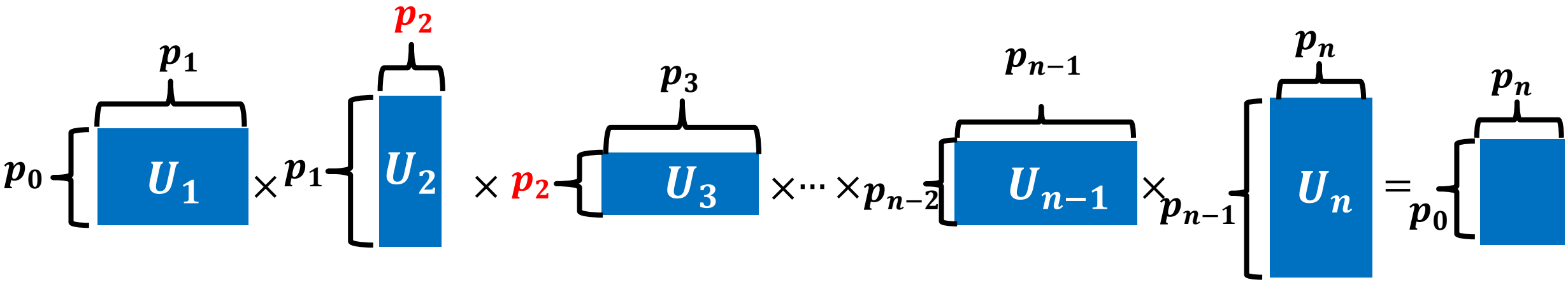
问题背景：基本知识

- n个矩阵相乘
  - 有一系列矩阵按顺序排列
  - 每个矩阵的行数=前一个矩阵的列数



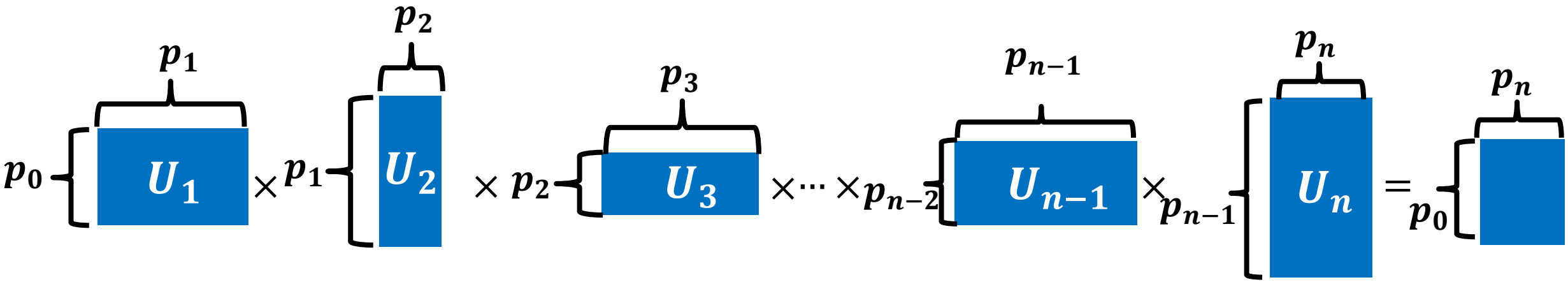
问题背景：基本知识

- n个矩阵相乘
  - 有一系列矩阵按顺序排列
  - 每个矩阵的行数=前一个矩阵的列数



问题背景：基本知识

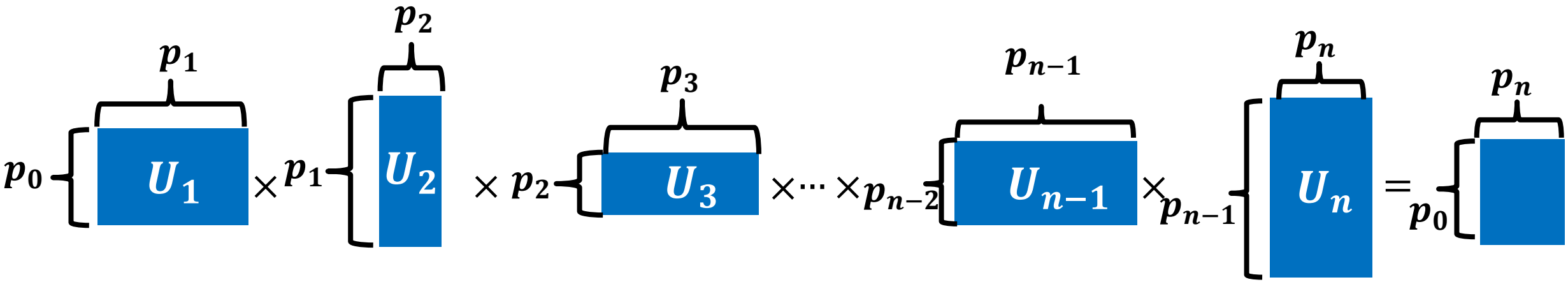
- n个矩阵相乘
  - 有一系列矩阵按顺序排列
  - 每个矩阵的行数=前一个矩阵的列数



- n个矩阵相乘也称为矩阵链乘法

问题背景：基本知识

- n个矩阵相乘
  - 有一系列矩阵按顺序排列
  - 每个矩阵的行数=前一个矩阵的列数



- n个矩阵相乘也称为矩阵链乘法

问题：如何确定相乘顺序（给矩阵链加括号），提高计算效率?

问题定义

矩阵链乘法问题

Matrix-chain Multiplication Problem

输入

- n个矩阵组成的矩阵链$U_{1..n} =< U_1, U_2, \dots, U_n >$
- 矩阵链$U_{1..n}$对应的维度数分别为$p_0, p_1, \dots, p_n$， U_i的维度为$p_{i-1} \times p_i$

输出

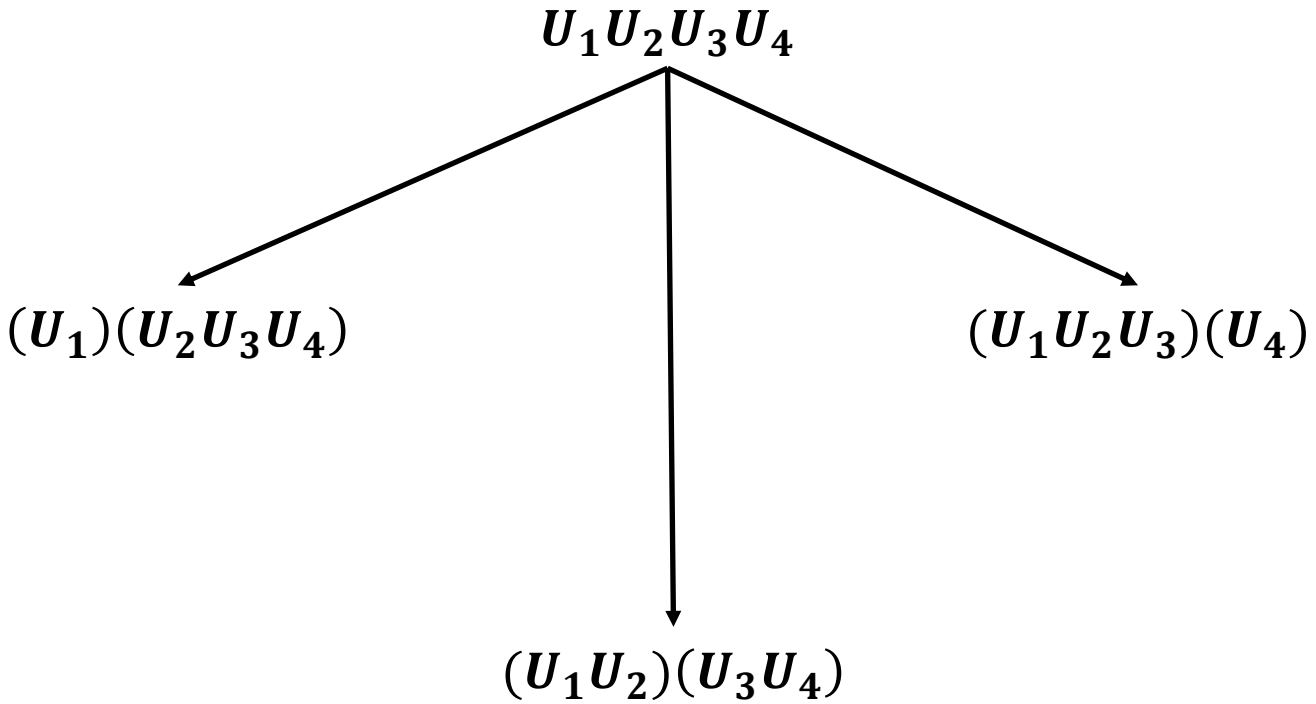
- 找到一种加括号的方式，以确定矩阵链乘法的计算顺序，使得

最小化矩阵链标量乘法的次数

问题示例

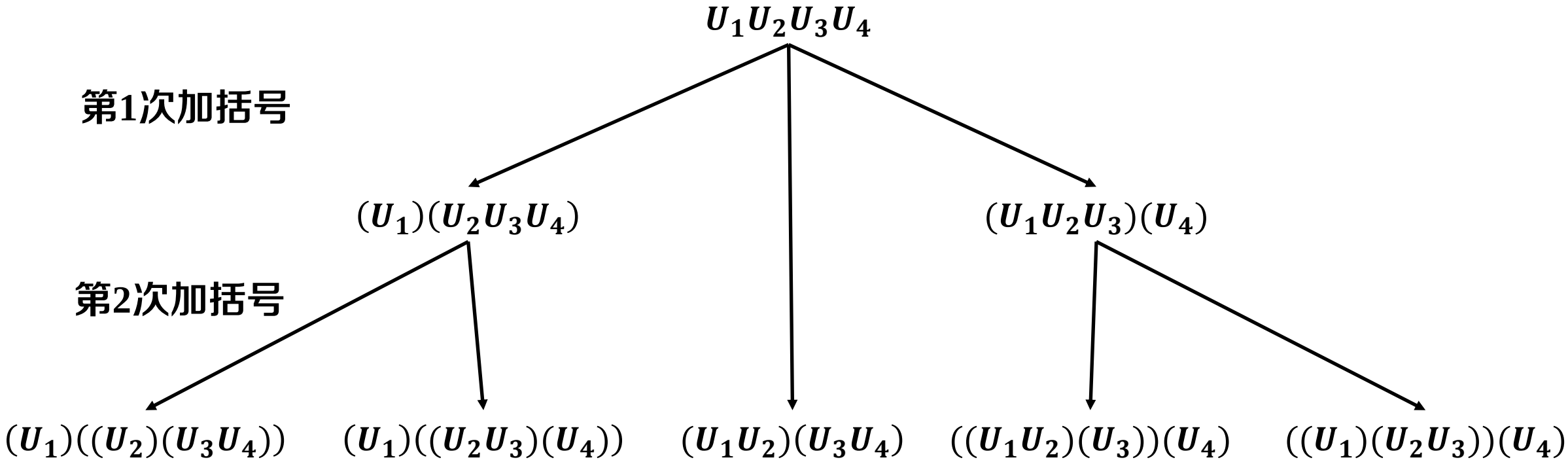
- 给定矩阵链：$U_{1..4} = U_1, U_2, U_3, U_4$
- 有如下加括号方式

第1次加括号



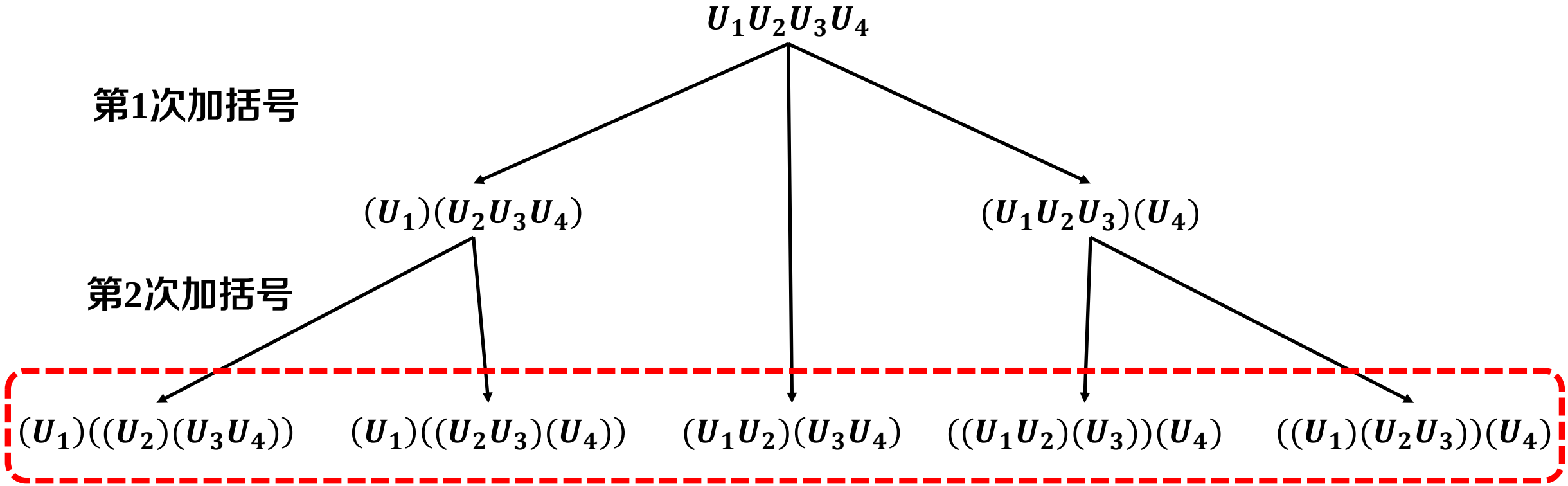
问题示例

- 给定矩阵链：$U_{1..4} = U_1, U_2, U_3, U_4$
- 有如下加括号方式



问题示例

- 给定矩阵链：$U_{1..4} = U_1, U_2, U_3, U_4$
- 有如下加括号方式



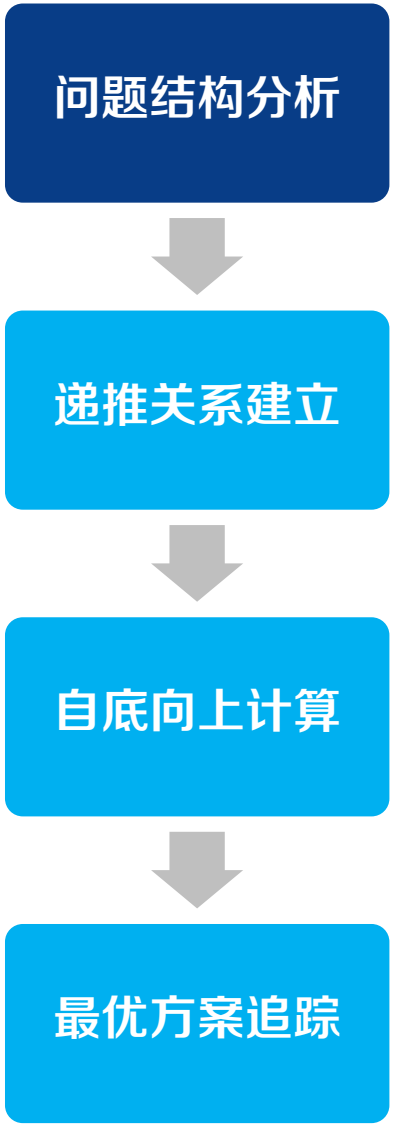
找到使标量乘法次数最小的加括号方式

问题结构分析

- 给出问题表示
  - $D[i,j]$：计算矩阵链$U_{i..j}$所需标量乘法的**最小次数**

$$U_{i..j} = p_{i-1}\underbrace{\overbrace{U_i}^{p_i} \times \cdots \times \overbrace{U_j}^{p_j}}_{D[i,j]}$$

- 明确原始问题
  - $D[1,n]$：计算矩阵链$U_{1..n}$所需标量乘法的**最小次数**



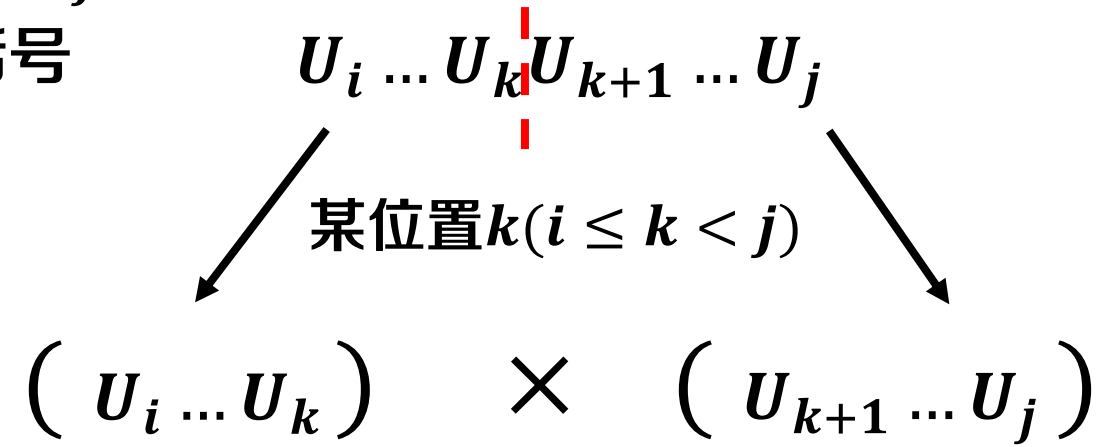
递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$

$$U_i \dots U_k U_{k+1} \dots U_j$$

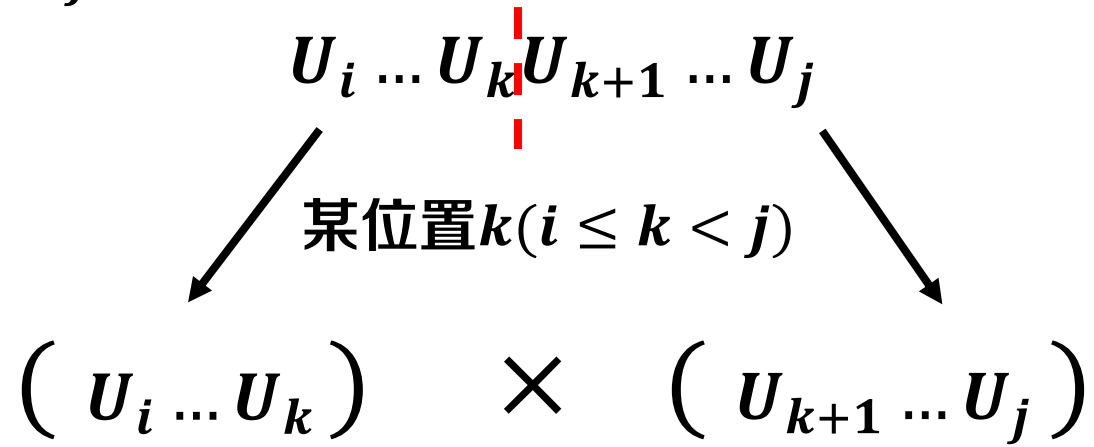
递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$
  - 加一次括号



递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



问题：如何保证不遗漏最优分割位置？

问题结构分析

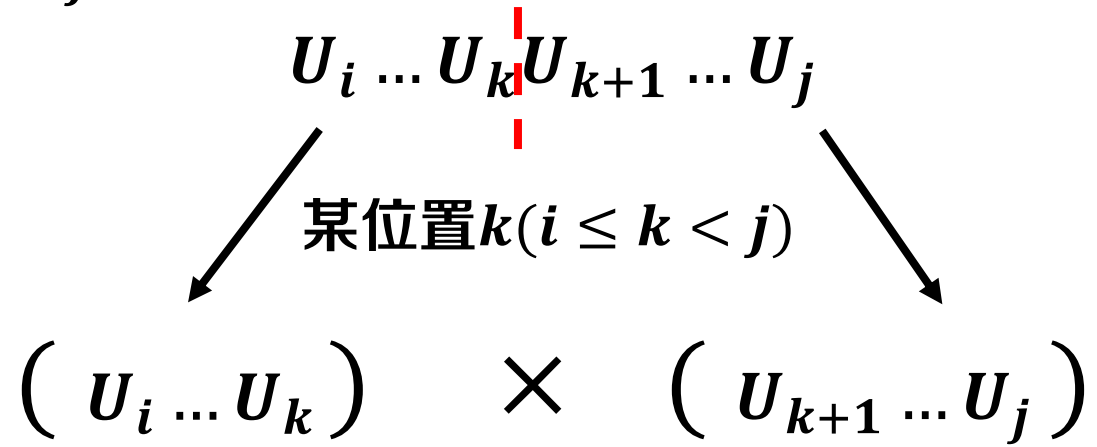
递推关系建立

自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



问题：如何保证不遗漏最优分割位置？

答案：枚举所有可能位置$i..j-1$，共$j-i$种

问题结构分析

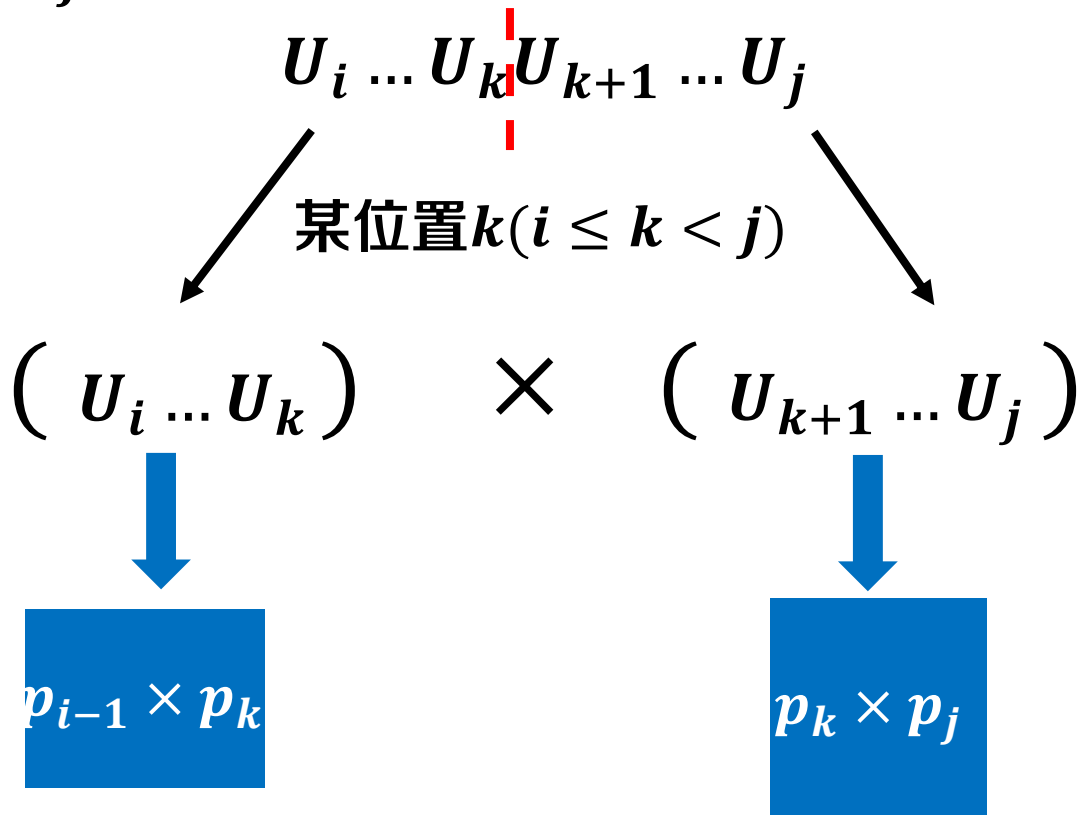
递推关系建立

自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



问题结构分析

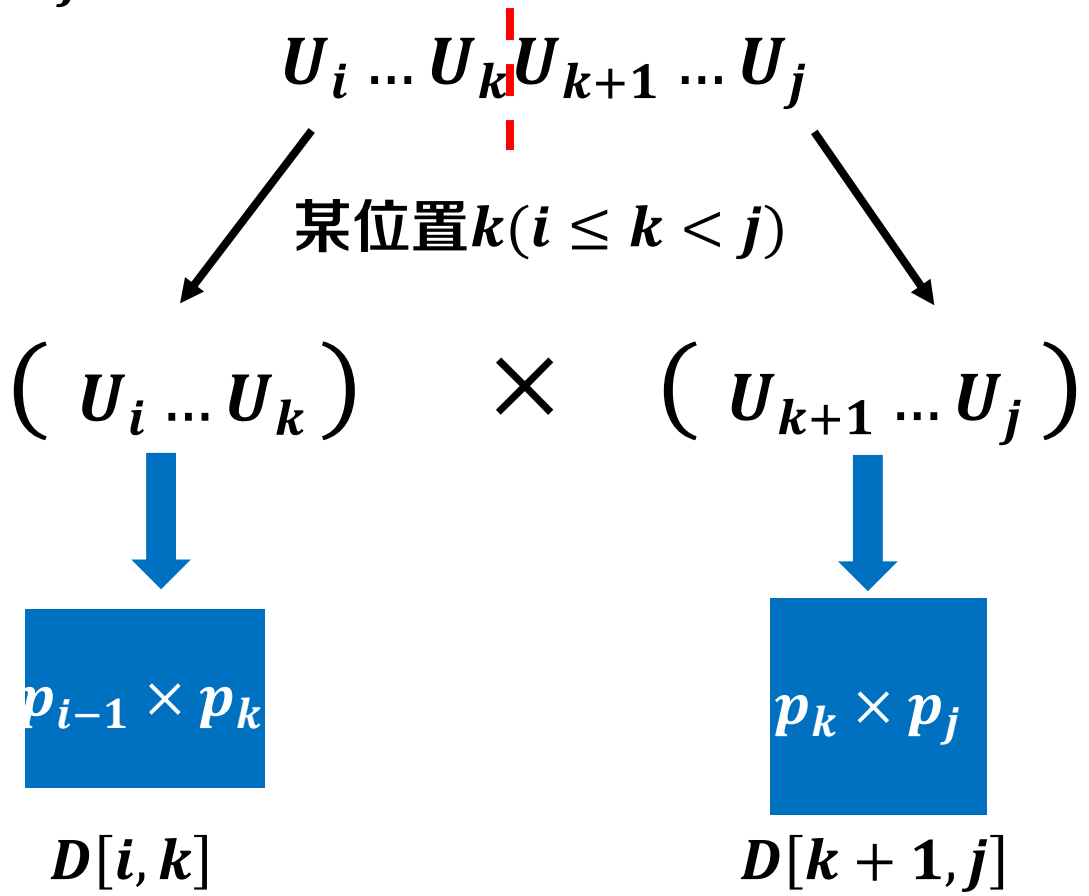
递推关系建立

自底向上计算

最优方案追踪

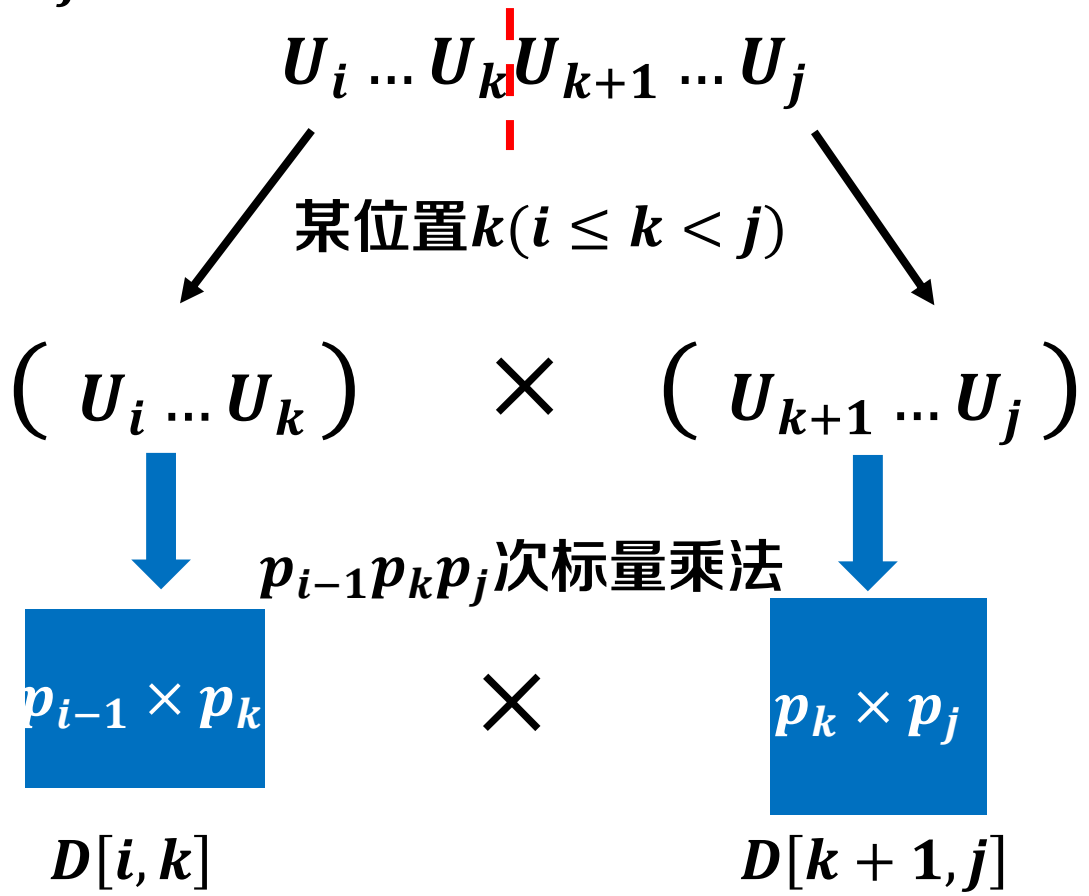
递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



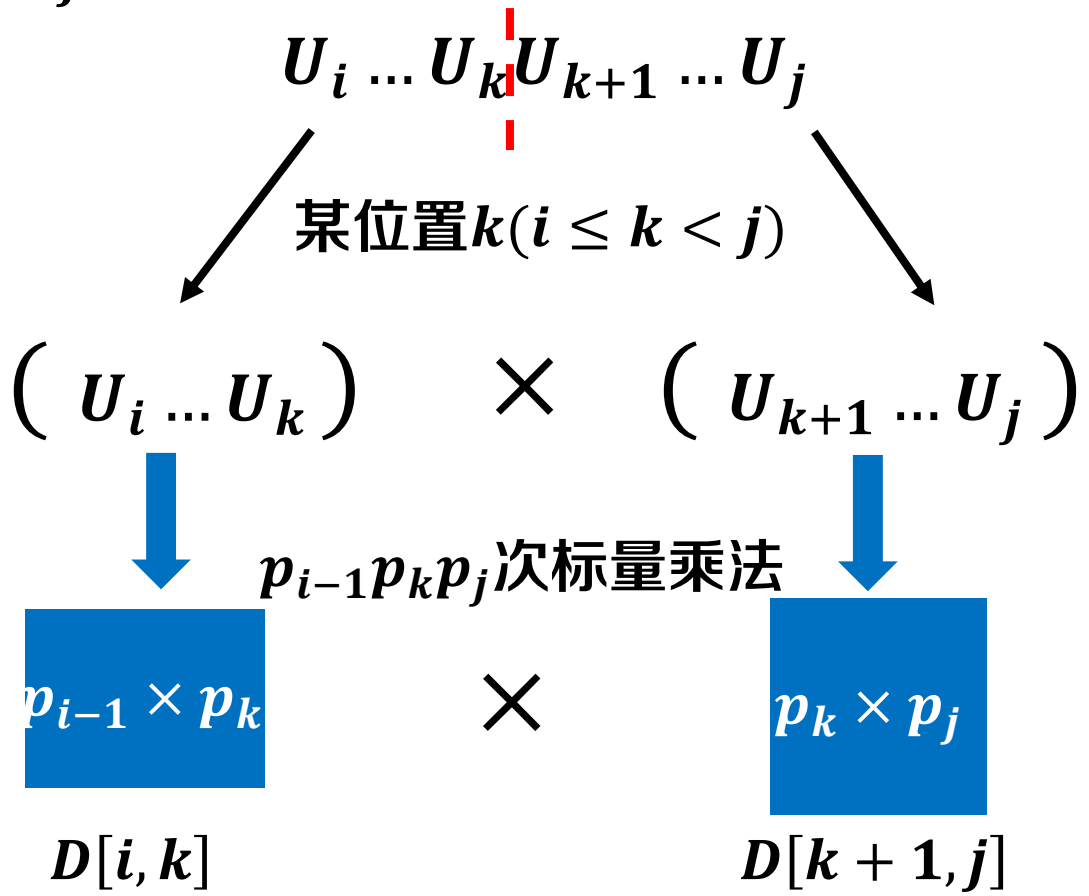
递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



- $D[i,j] = D[i,k] + D[k+1,j] + p_{i-1}p_kp_j$

问题结构分析

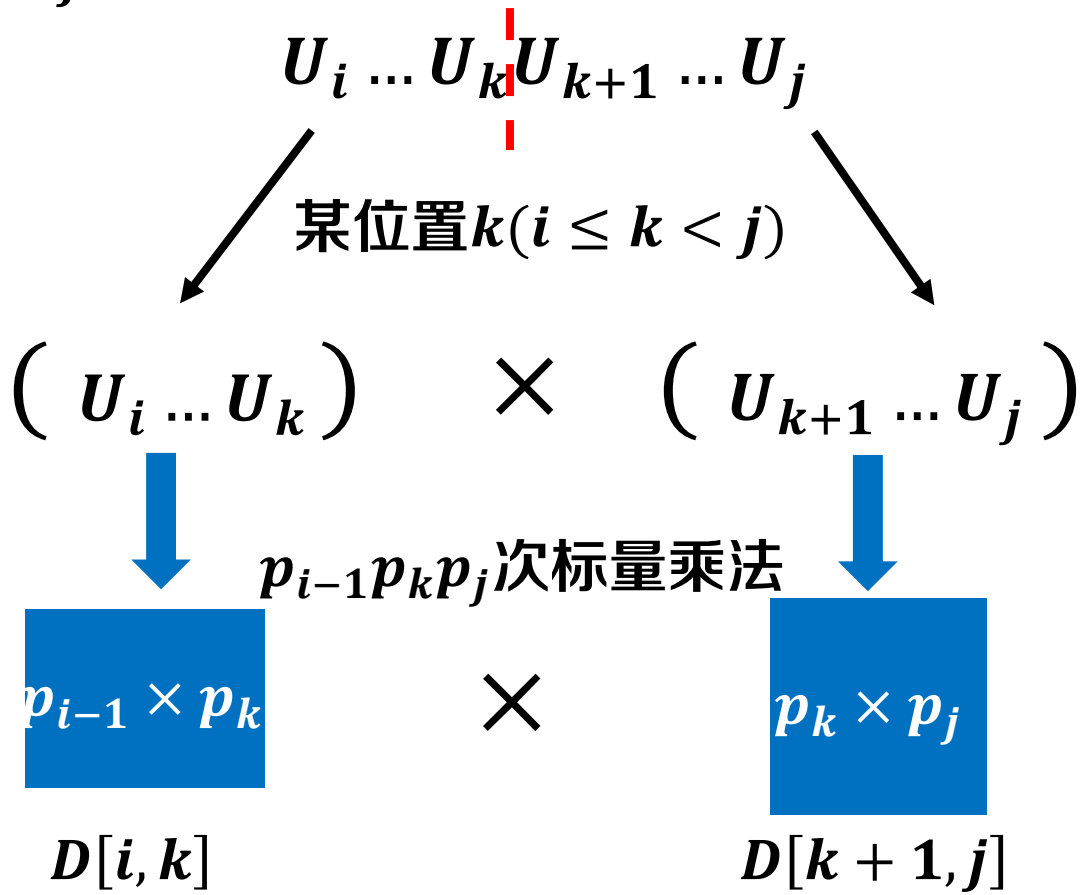
递推关系建立

自底向上计算

最优方案追踪

递推关系建立：分析最优（子）结构

- 对矩阵链$U_{i..j}$，求解$D[i,j]$



- $D[i,j] = D[i,k] + D[k+1,j] + p_{i-1}p_kp_j$

最优子结构

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

递推关系建立：构造递推公式

- 对每个位置$k(i \le k < j)$
  - $D[i,j] = D[i,k] + D[k+1,j] + p_{i-1}p_k p_j$

- 枚举所有k，得到递推式
  - $D[i,j] = \min\limits_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

自底向上计算：确定计算顺序

- 初始化
  - $i = j$时，矩阵链只有一个矩阵，乘法次数为0

$D[i,j]$	$j=1$	2	3	…	…	$n-1$	n
$i=1$	0						
2		0					
3			0				
…				0			
…					0		
$n-1$						0	
n							0

自底向上计算：确定计算顺序

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

$D[i,j]$	$j=1$	2	3	…	…	$n-1$	n
$i=1$	0						
2		0					
3			0				
…				0			
…					0		
$n-1$						0	
n							0

$i < j$ 只用上三角

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

$j=1$	2	3	…	…	$n-1$	n	$D[i,j]$
0							$i=1$
	0						2
		0					3
			0				…
				0			…
					0		$n-1$
						0	n

自底向上计算：确定计算顺序

- 递推公式
  - $D[i,j] = \min_{i \le k < j} \{D[i,k] + D[k+1,j] + p_{i-1}p_k p_j\}$

$j=1$	2	3	…	…	$n-1$	n	$D[i,j]$
0							$i=1$
	0						2
		0	$D[i,k]$		$D[i,j]$		3
			0		$D[k+1,j]$		…
				0			…
					0		$n-1$
						0	n

问题结构分析

递推关系建立

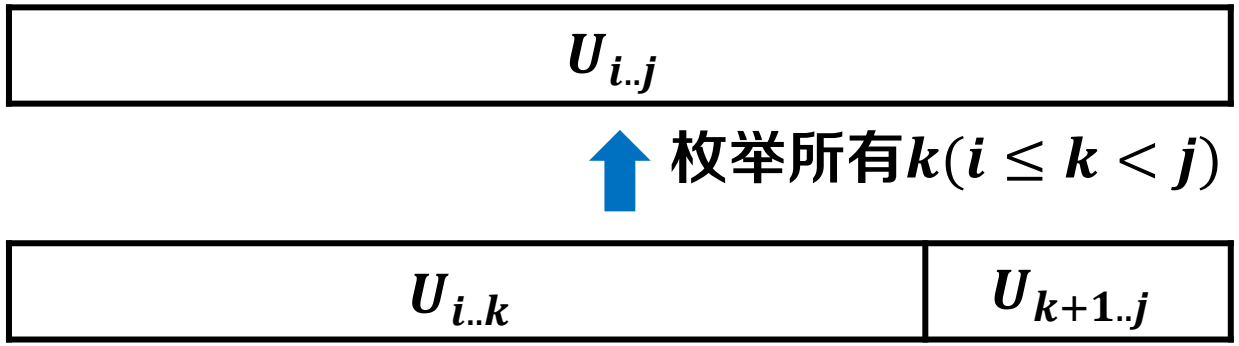
自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

- 观察枚举过程



问题结构分析

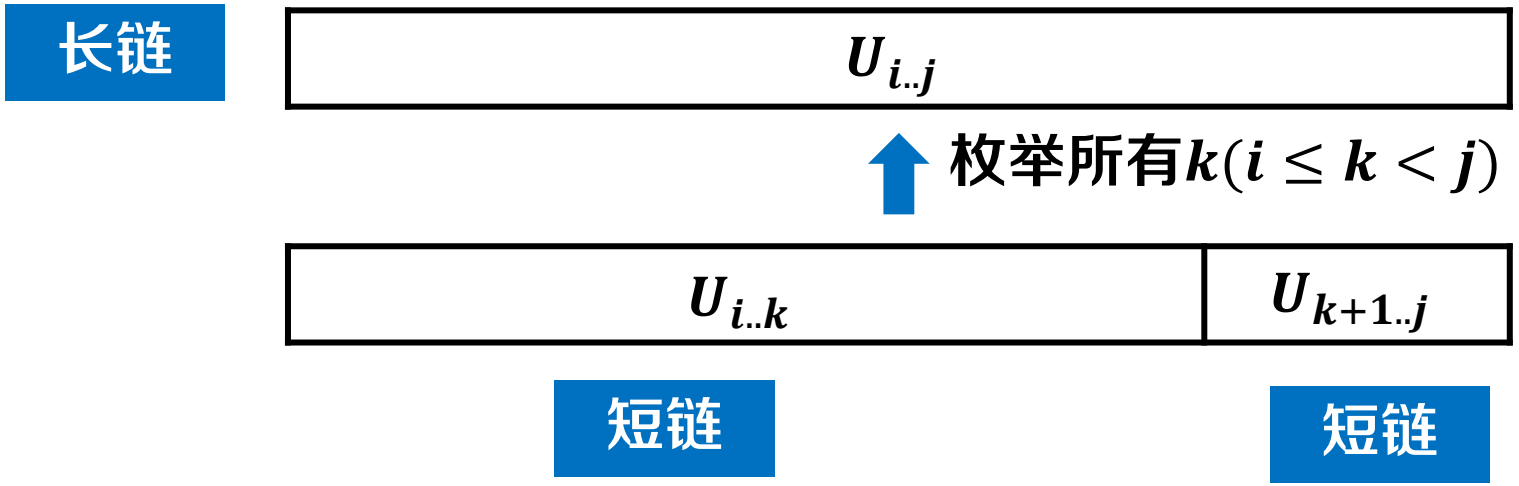
递推关系建立

自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$
- 观察枚举过程



问题结构分析

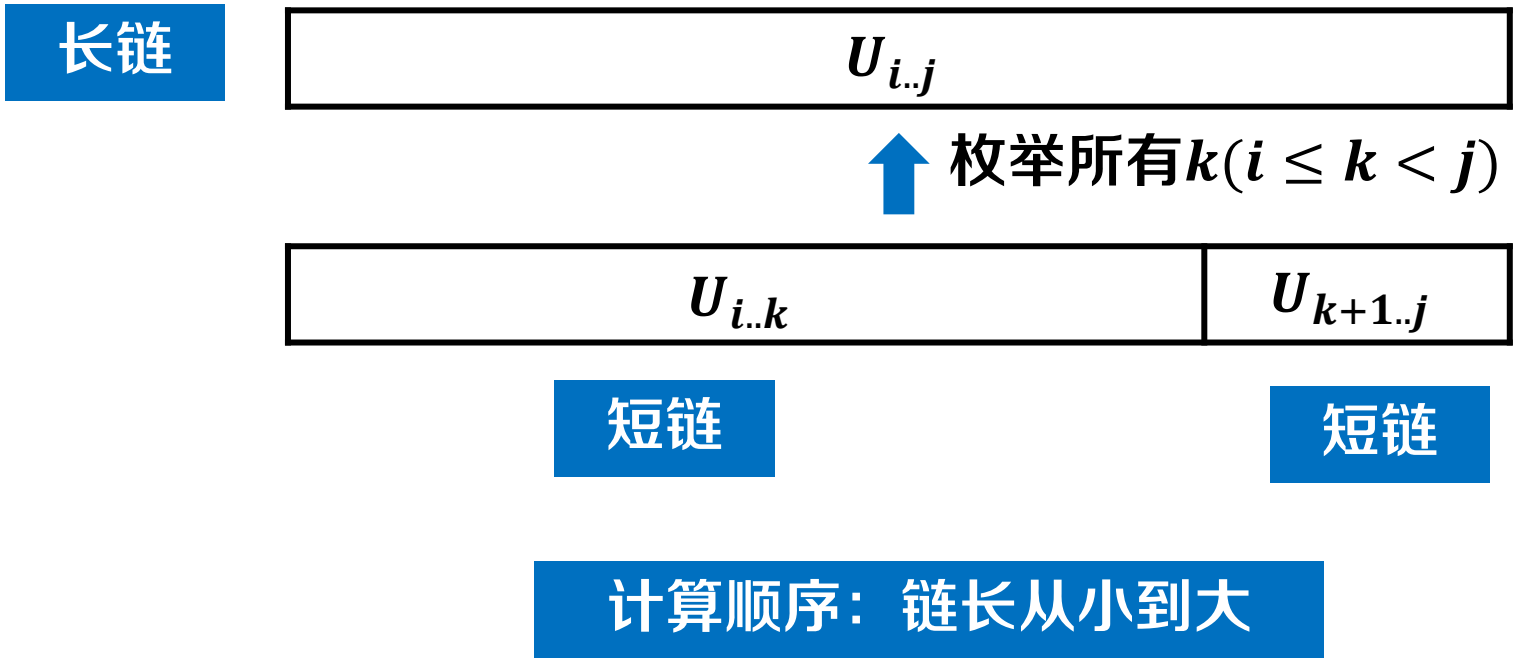
递推关系建立

自底向上计算

最优方案追踪

自底向上计算：确定计算顺序

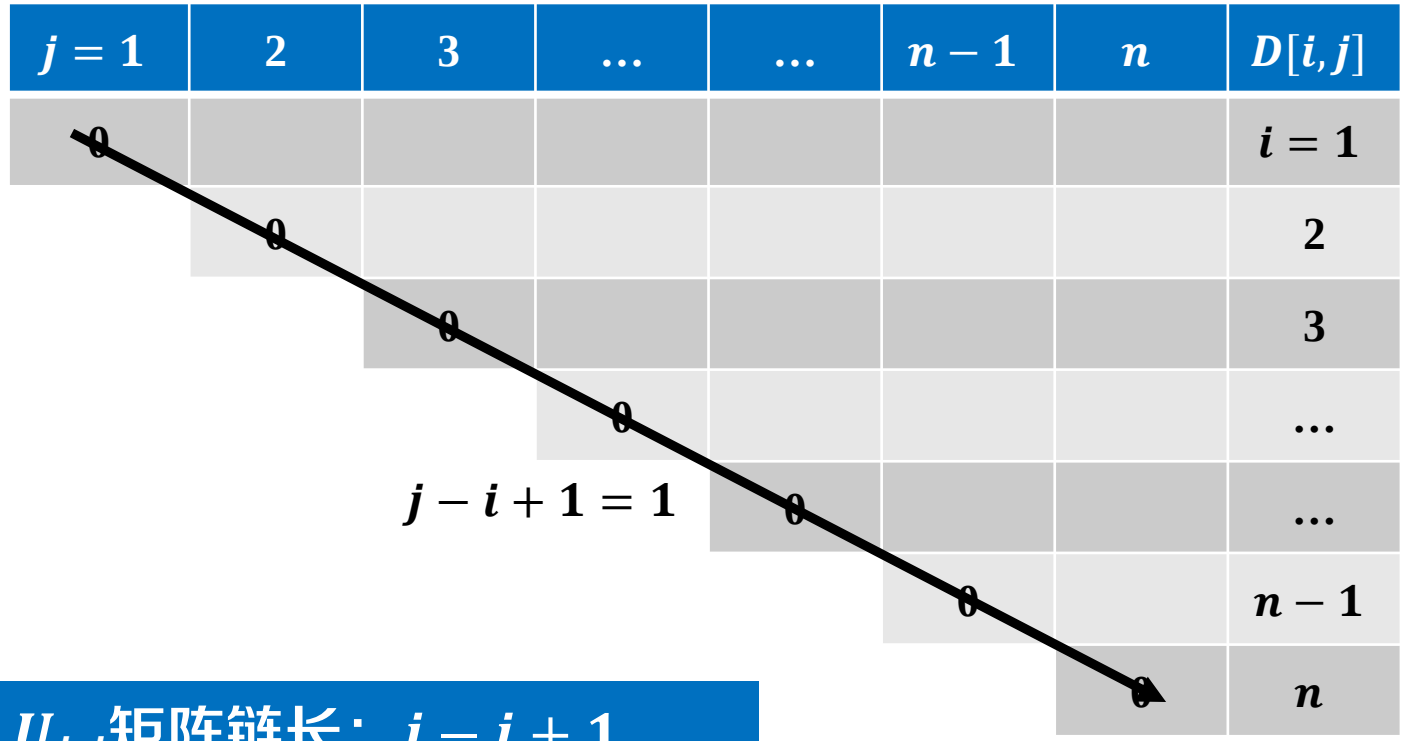
- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$
- 观察枚举过程



计算顺序：链长从小到大

自底向上计算：依次计算问题

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$



$U_{i..j}$矩阵链长：$j - i + 1$

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次计算问题

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

$j=1$	2	3	…	…	$n-1$	n	$D[i,j]$
0							$i=1$
	0						2
		0		$j-i+1=2$			3
			0				…
				0			…
					0		$n-1$
						0	n

矩阵链长：$j-i+1$

问题结构分析

递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次计算问题

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$

$j=1$	2	3	…	…	$n-1$	n	$D[i,j]$
0							$i=1$
	0						2
		0			$j-i+1=3$		3
			0				…
				0			…
					0		$n-1$
						0	n

矩阵链长：$j-i+1$

问题结构分析

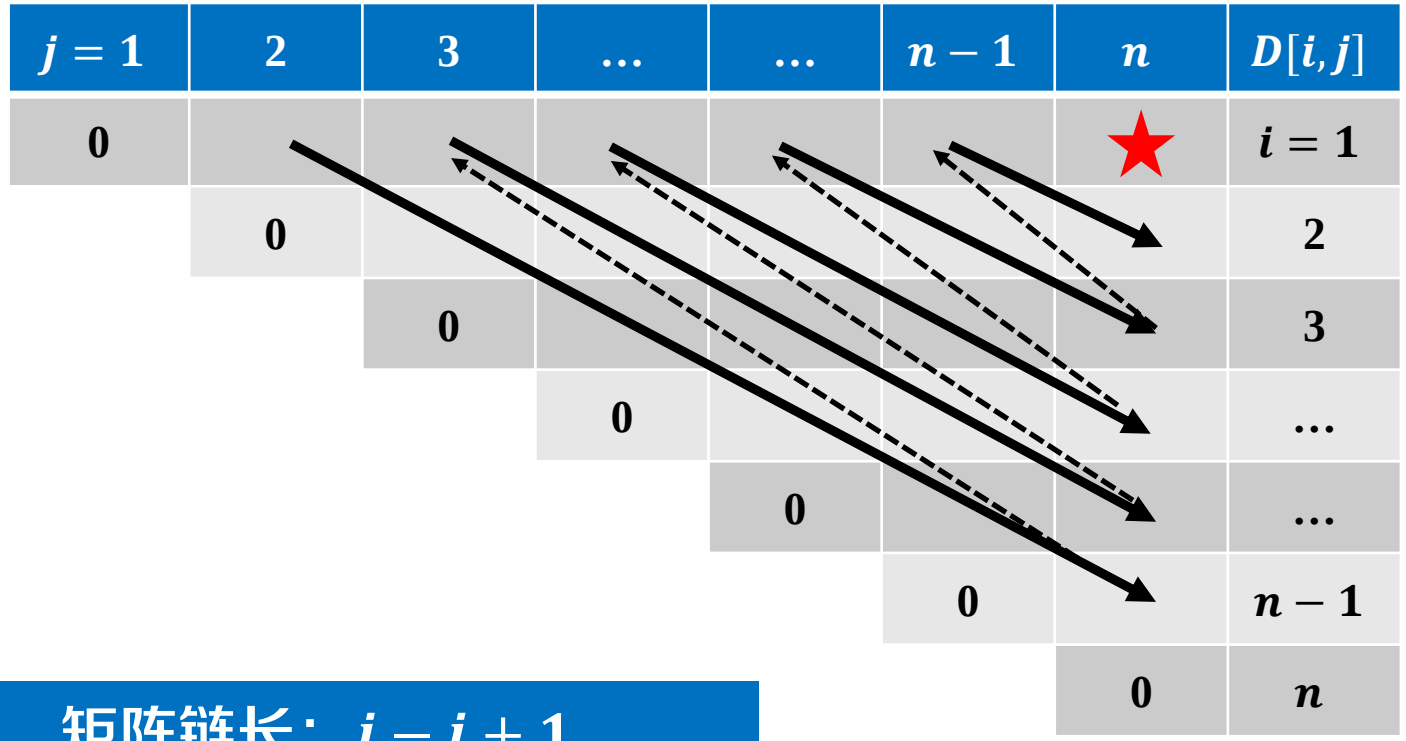
递推关系建立

自底向上计算

最优方案追踪

自底向上计算：依次计算问题

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$



矩阵链长：$j-i+1$

问题结构分析

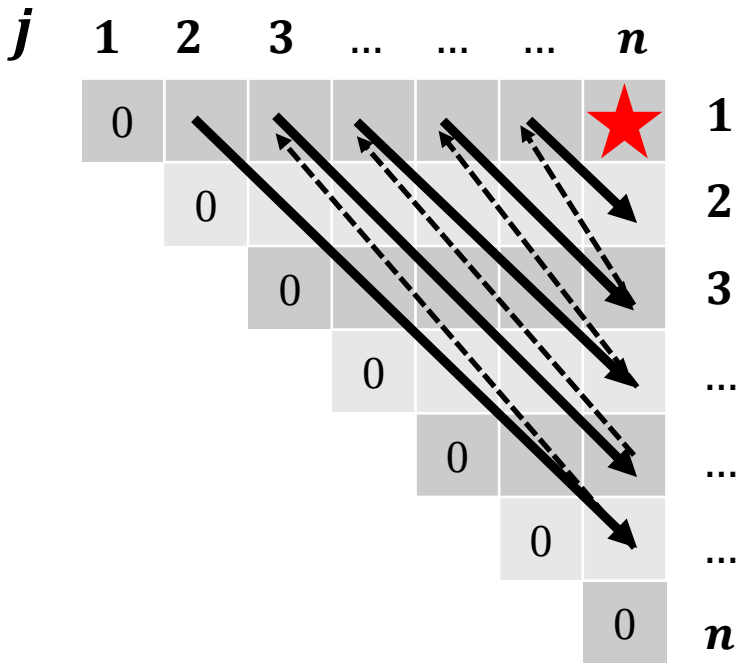
递推关系建立

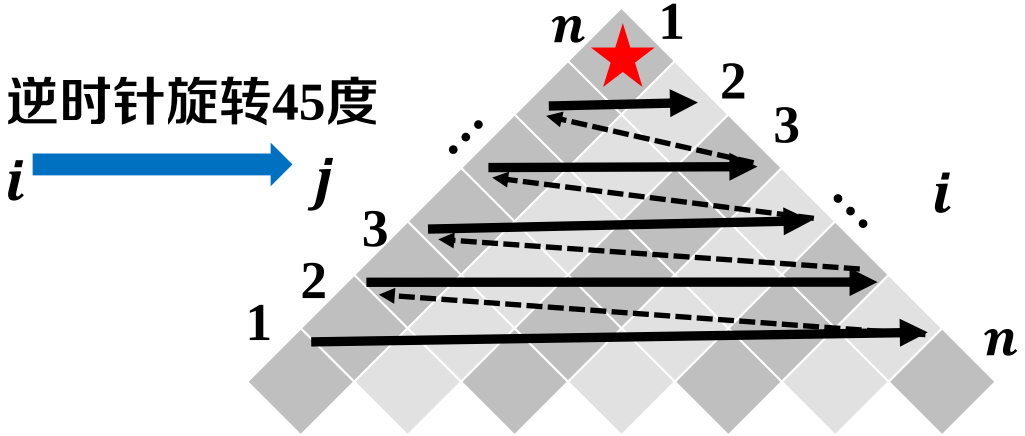
自底向上计算

最优方案追踪

自底向上计算：依次计算问题

- 递推公式
  - $D[i,j] = \min_{i \le k < j}(D[i,k] + D[k+1,j] + p_{i-1}p_k p_j)$





问题结构分析

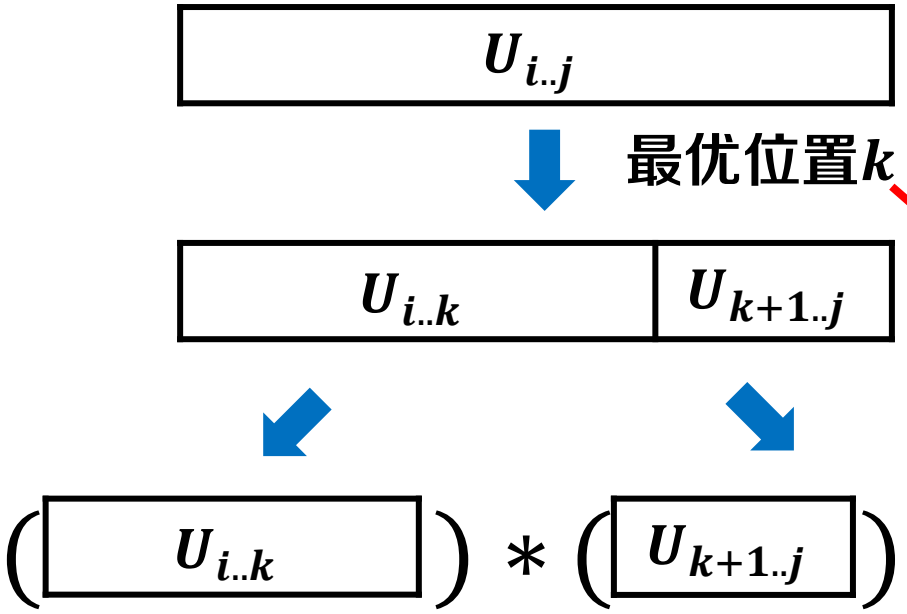
递推关系建立

自底向上计算

最优方案追踪

最优方案追踪：记录决策过程

- 构造追踪数组$Rec[1..n, 1..n]$
- $Rec[i, j]$：矩阵链$U_{i..j}$的最优分割位置



Rec	1	…	j	…	n
1					
…					
i			k		
…					
n					

问题结构分析

递推关系建立

自底向上计算

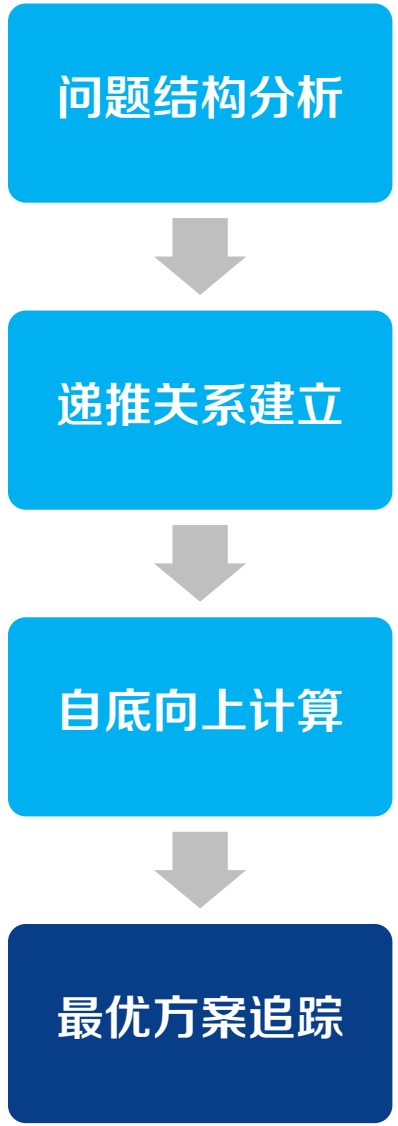
最优方案追踪

最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$U_1 \dots U_s U_{s+1} \dots U_k U_{k+1} \dots U_t U_{t+1} \dots U_{n-1} U_n$$

Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$							k
2							
…							
$k+1$							
…							
$n-1$							
n							

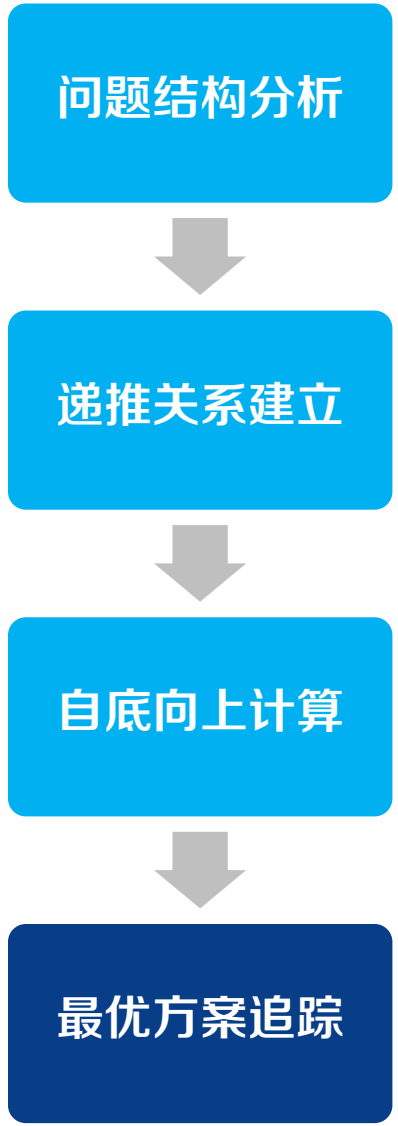


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$(U_1 \cdots U_s U_{s+1} \cdots U_k)(U_{k+1} \cdots U_t U_{t+1} \cdots U_{n-1} U_n)$$

Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$							k
2							
…							
$k+1$							
…							
$n-1$							
n							

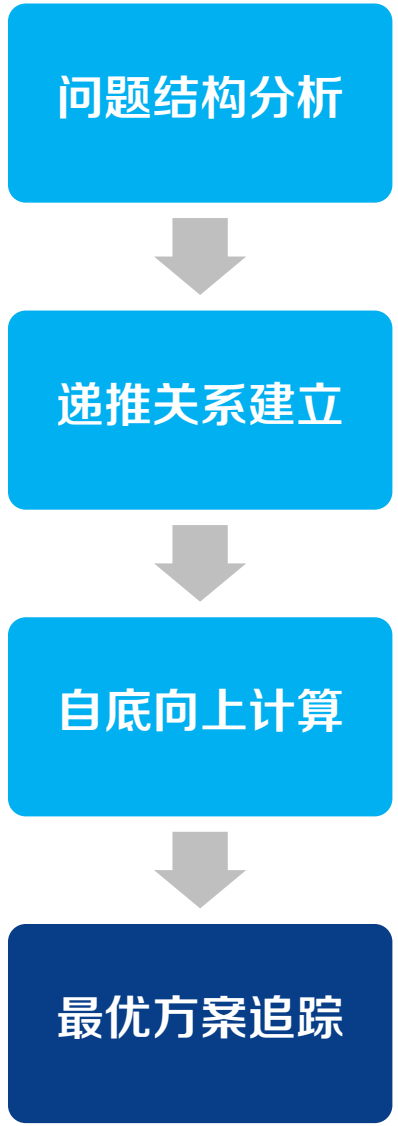


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$(U_1 \dots U_s U_{s+1} \dots U_k)(U_{k+1} \dots U_t U_{t+1} \dots U_{n-1} U_n)$$

Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$				s ←			k
2							
…							
$k+1$							
…							
$n-1$							
n							

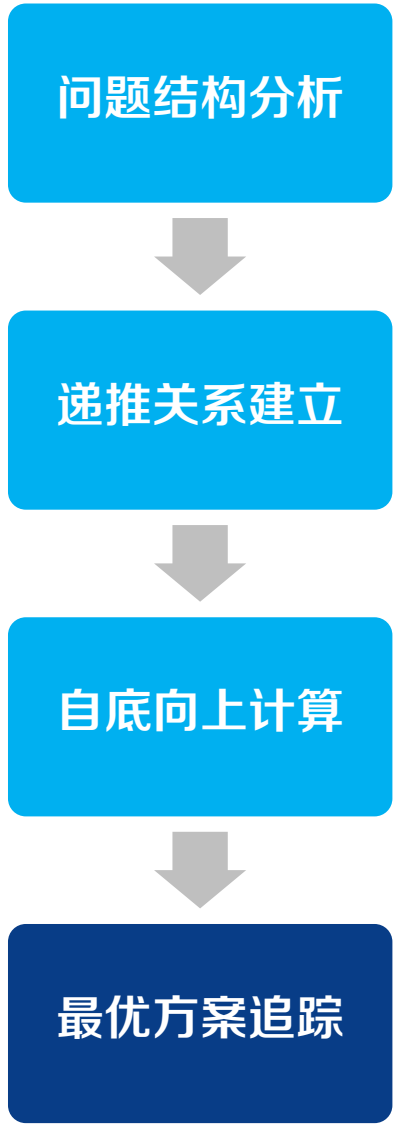


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$(U_1 \cdots U_s U_{s+1} \cdots U_k)(U_{k+1} \cdots U_t U_{t+1} \cdots U_{n-1} U_n)$$

Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$				s			k
2							
…							
$k+1$							t
…							
$n-1$							
n							

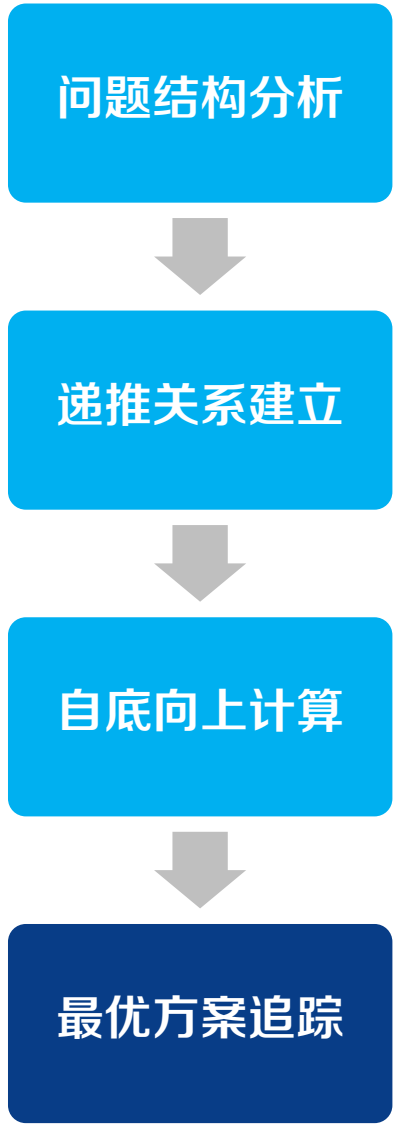


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$(U_1 \cdots U_s)(U_{s+1} \cdots U_k)(U_{k+1} \cdots U_t)(U_{t+1} \cdots U_{n-1} U_n)$$

Rec	$j=1$	2	..	k	…	$n-1$	n
$i=1$				s			k
2							
…							
k							t
…							
$n-1$							
n							

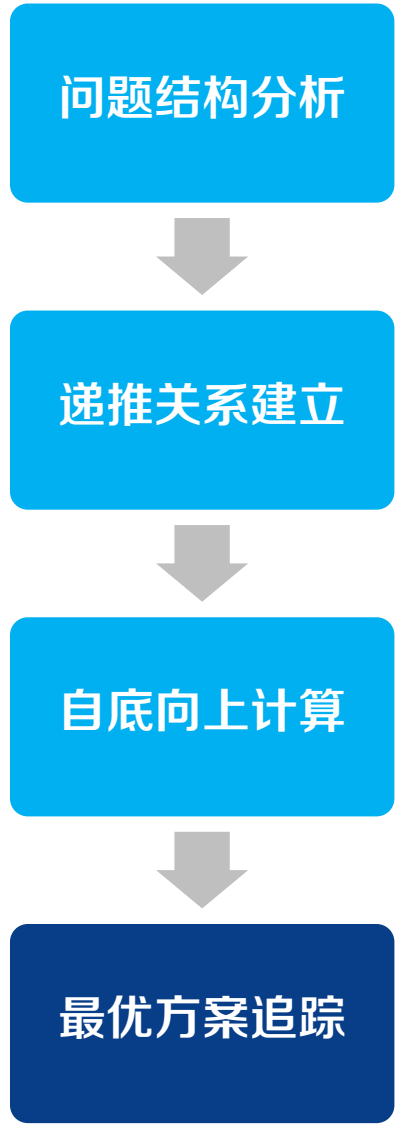


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案

$$((U_1 \cdots U_s)(U_{s+1} \cdots U_k))((U_{k+1} \cdots U_t)(U_{t+1} \cdots U_{n-1}U_n))$$

Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$				s			k
2							
…							
k							t
…							
$n-1$							
n							

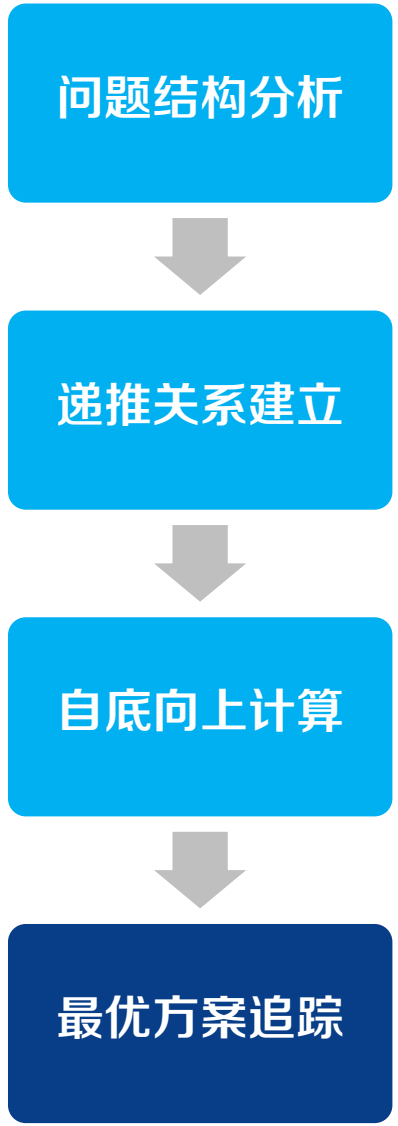


最优方案追踪：输出最优方案

- 根据追踪数组，递归输出方案
  - 递归出口：矩阵链长为1

$$((U_1 \dots U_s)(U_{s+1} \dots U_k))((U_{k+1} \dots U_t)(U_{t+1} \dots U_{n-1}U_n))$$

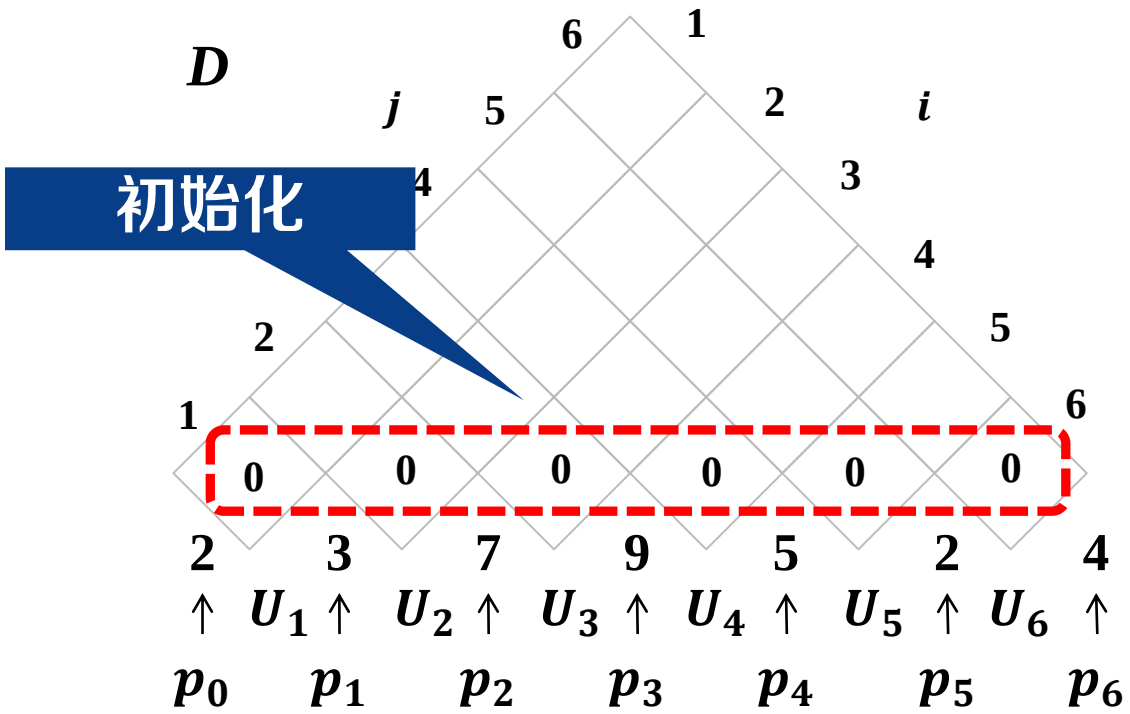
Rec	$j=1$	2	…	k	…	$n-1$	n
$i=1$				s			k
2							
…							
k							t
…							
$n-1$							
n							

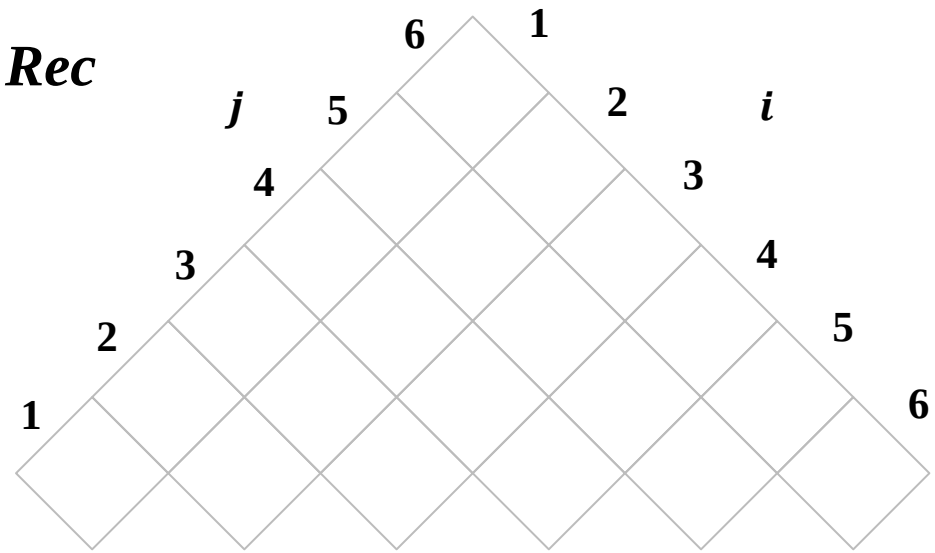


算法实例

- 给定矩阵链：$U_1U_2U_3U_4U_5U_6$
- 对应行列数

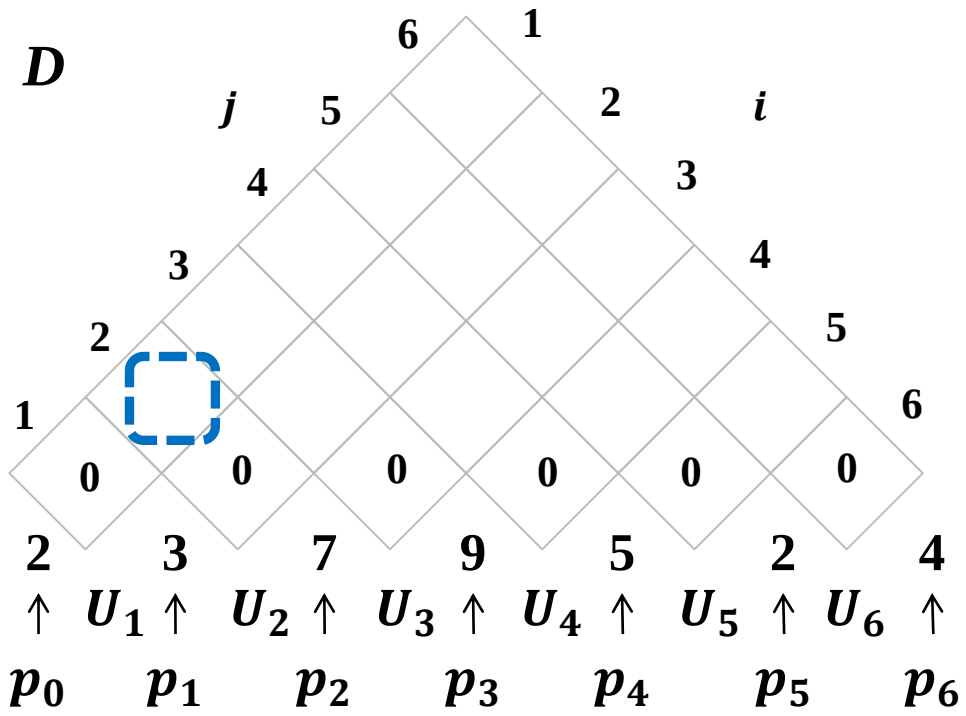
p_0	p_1	p_2	p_3	p_4	p_5	p_6
2	3	7	9	5	2	4

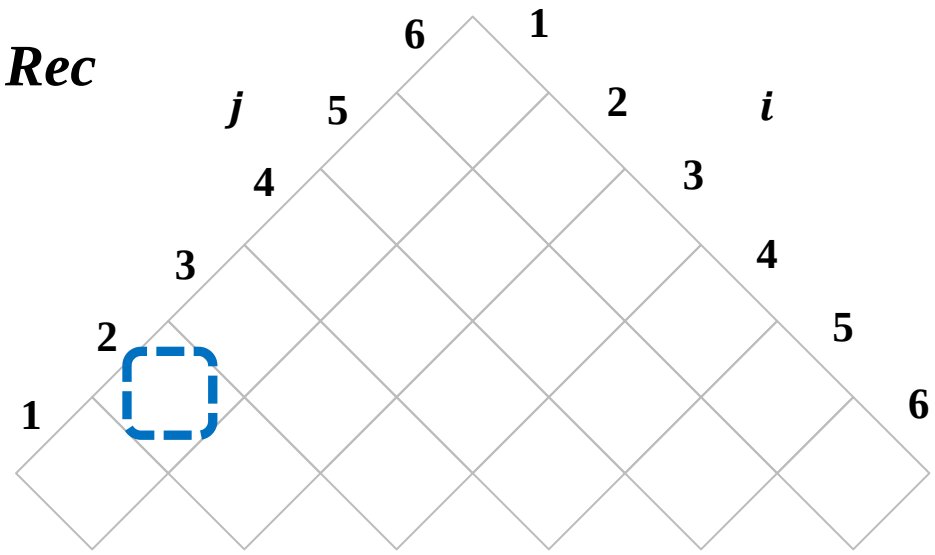




算法实例

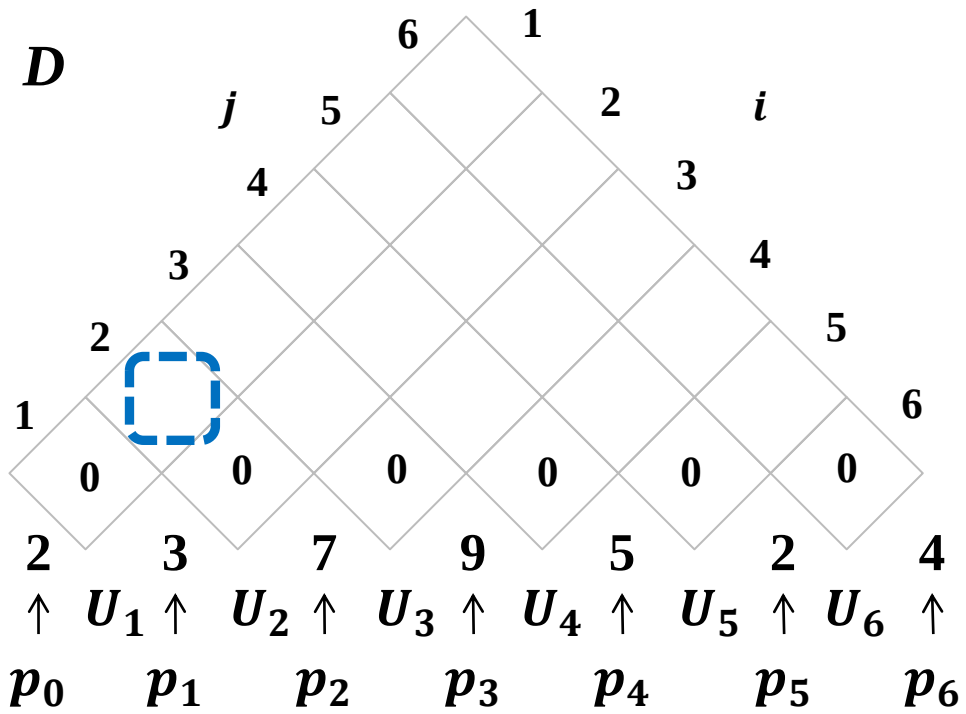
- $D[1,2] = \min_{1 \le k < 2} (D[1,k] + D[k+1,2] + p_0 p_k p_2)$

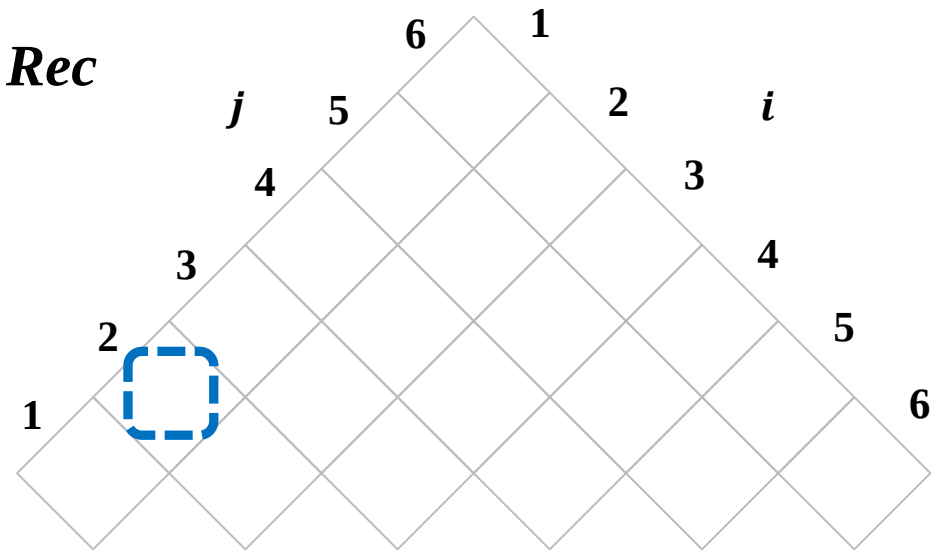




算法实例

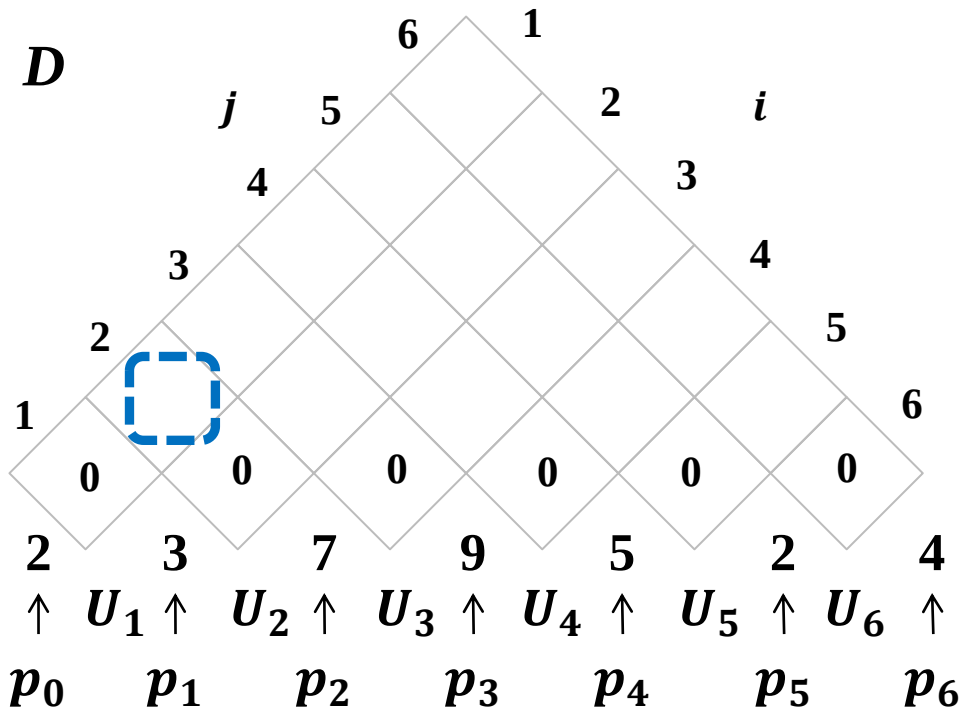
- $D[1,2] = \min_{1 \le k < 2} (D[1,k] + D[k+1,2] + p_0 p_k p_2)$

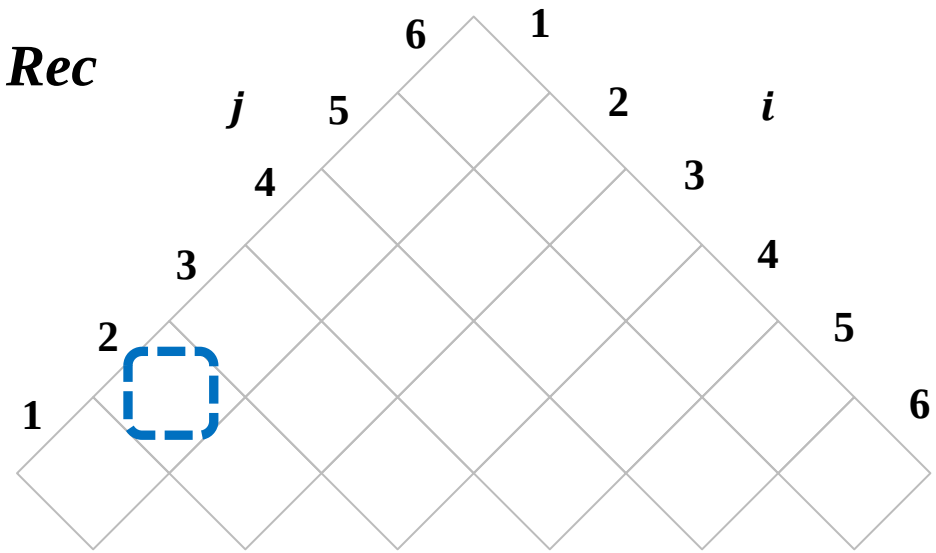




算法实例

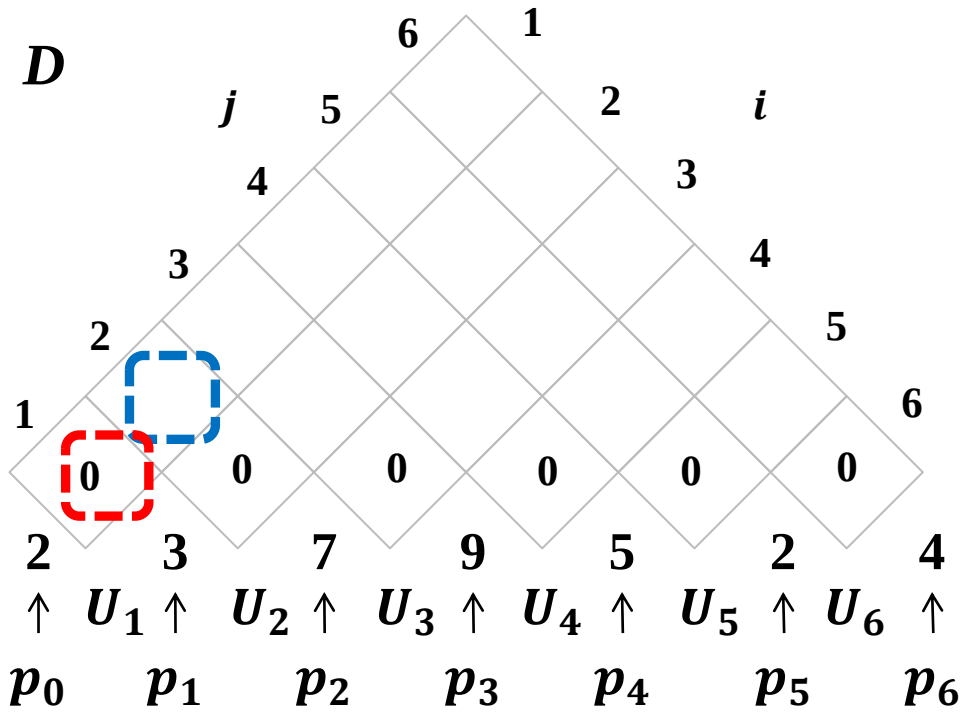
- $D[1,2] = \min_{1 \le k < 2} (D[1,1] + D[1+1,2] + p_0 p_1 p_2)$

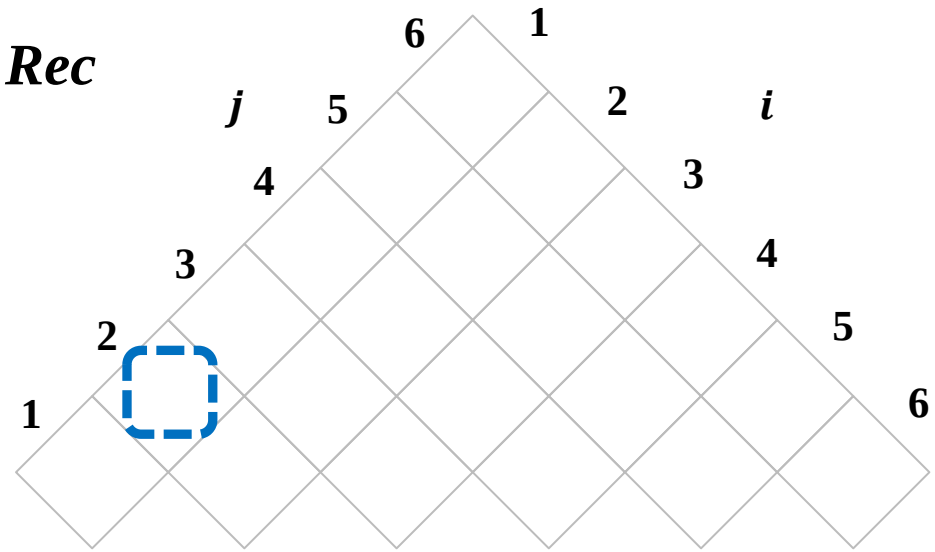




算法实例

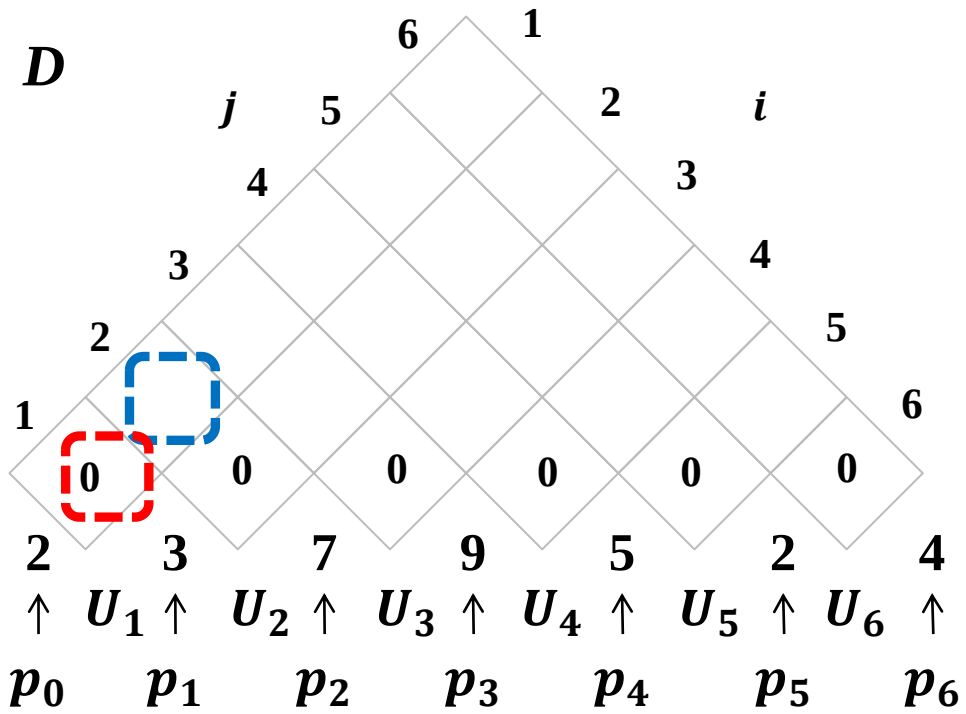
- $D[1,2] = \min_{1 \le k < 2} (D[1,1] + D[1+1,2] + p_0 p_1 p_2)$

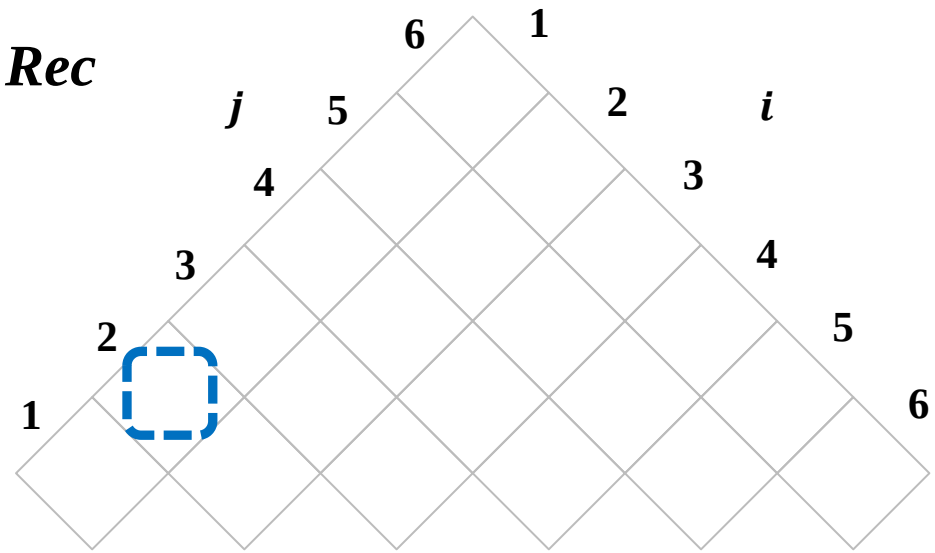




算法实例

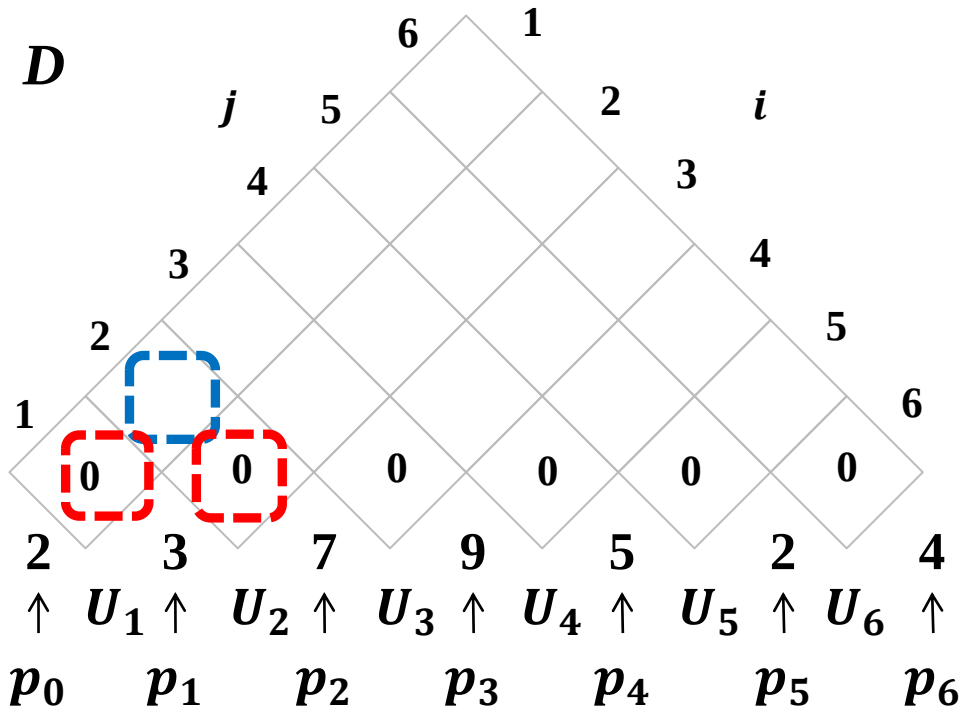
- $D[1,2] = \min_{1 \le k < 2} (0 + D[1+1,2] + p_0 p_1 p_2)$

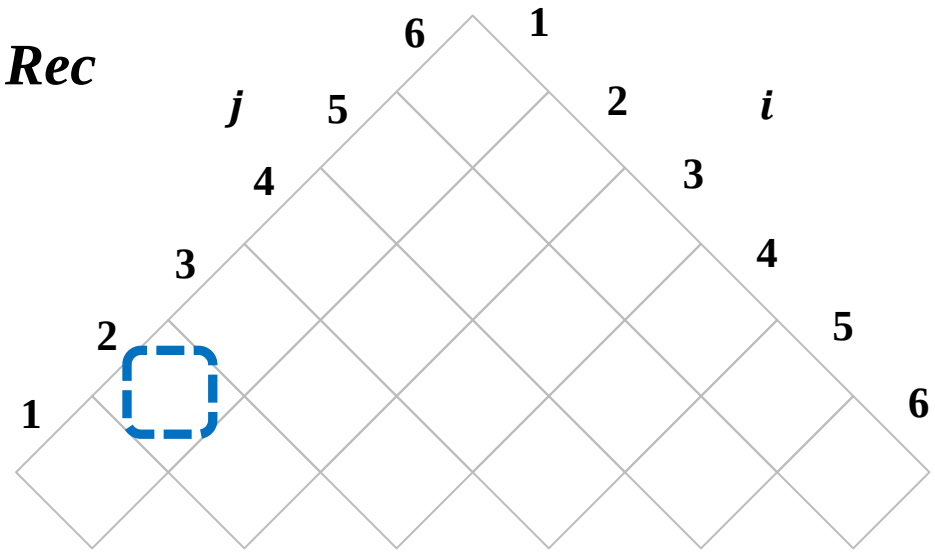




算法实例

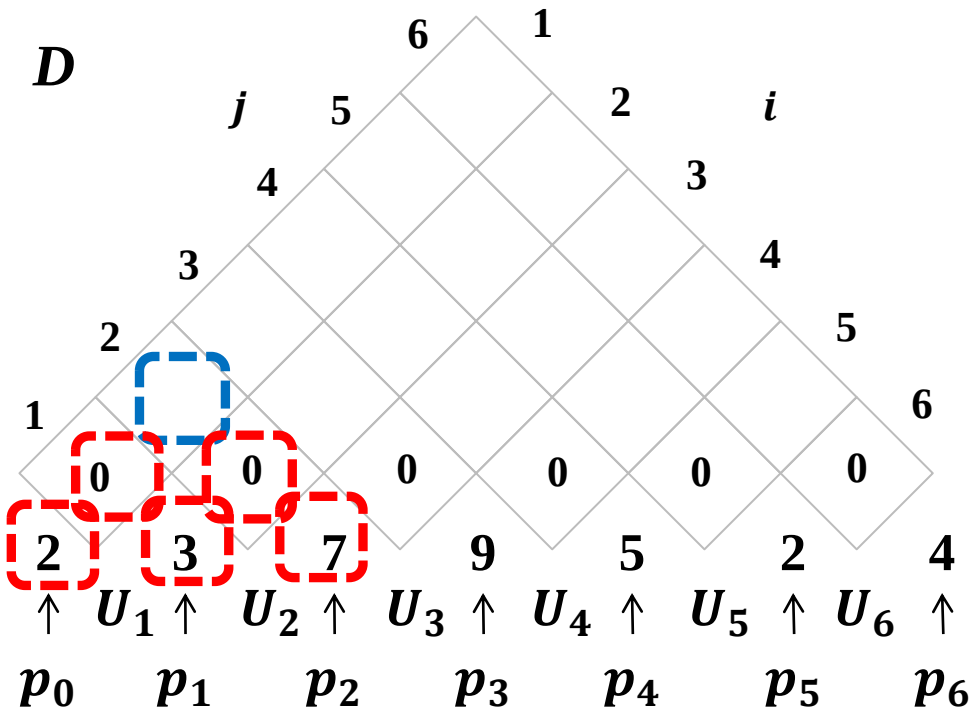
- $D[1,2] = \min_{1 \le k < 2}(0 + D[1+1,2] + p_0 p_1 p_2)$

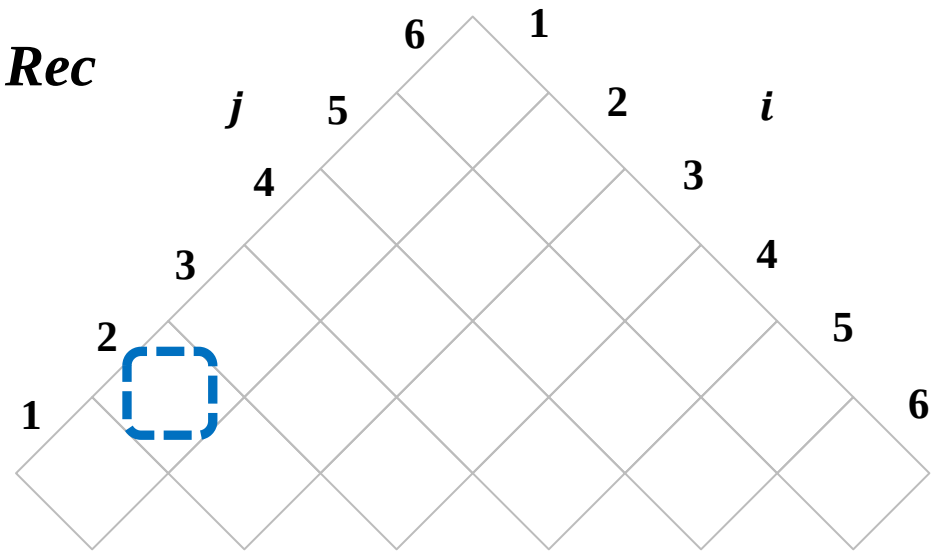




算法实例

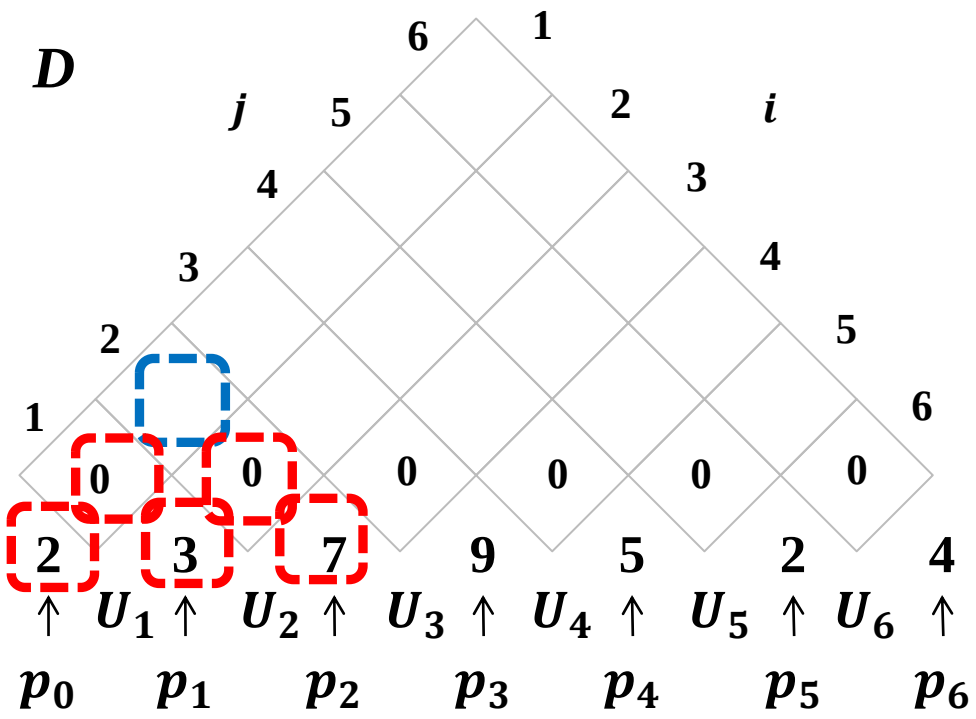
- $D[1,2] = \min_{1 \le k < 2} (0 + 0 + p_0 p_1 p_2)$

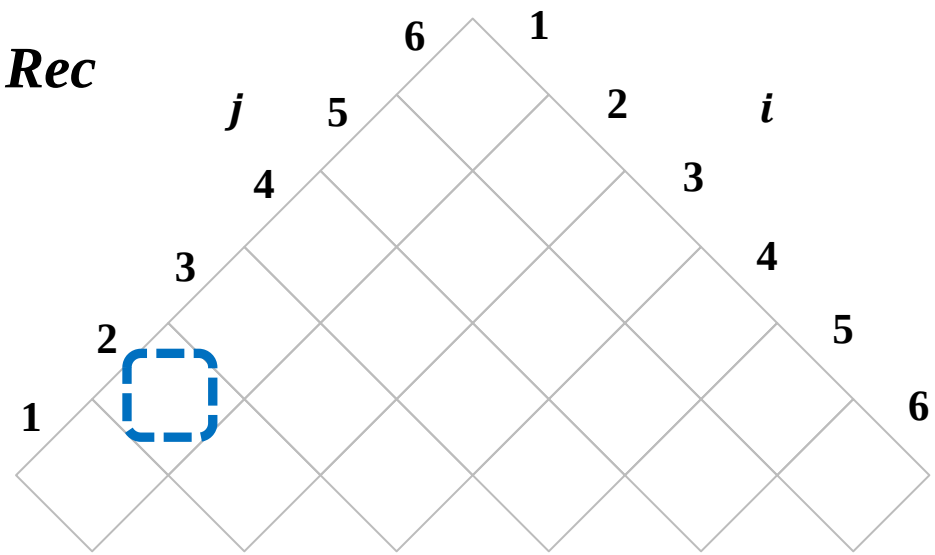




算法实例

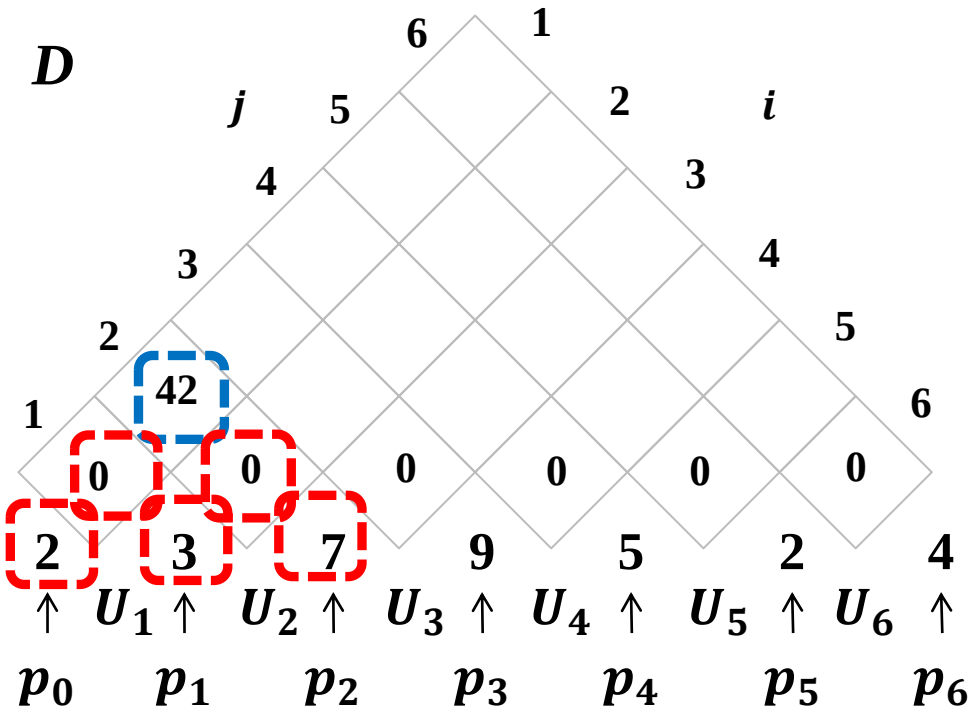
- $D[1,2] = \min_{1 \le k < 2} (0 + 0 + 2 \times 3 \times 7)$

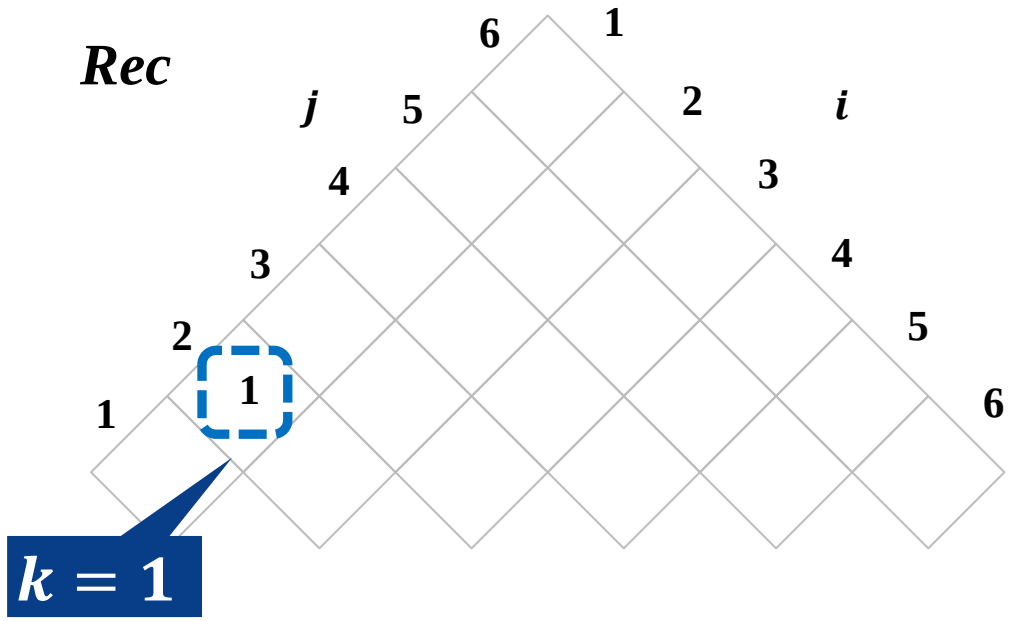




算法实例

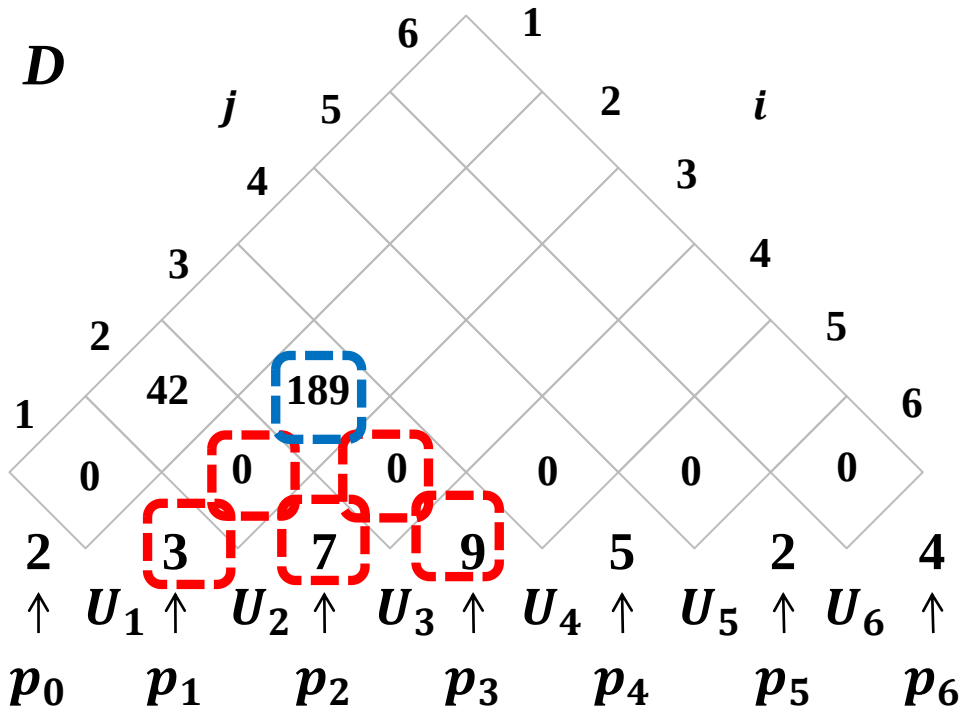
- $D[1,2] = \min_{1 \le k < 2} (0 + 0 + 2 \times 3 \times 7) = 42$

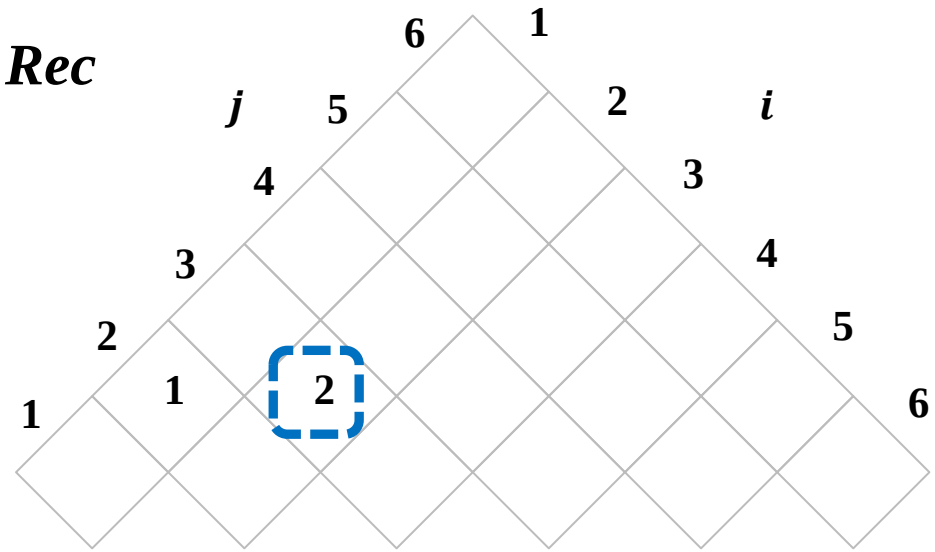




算法实例

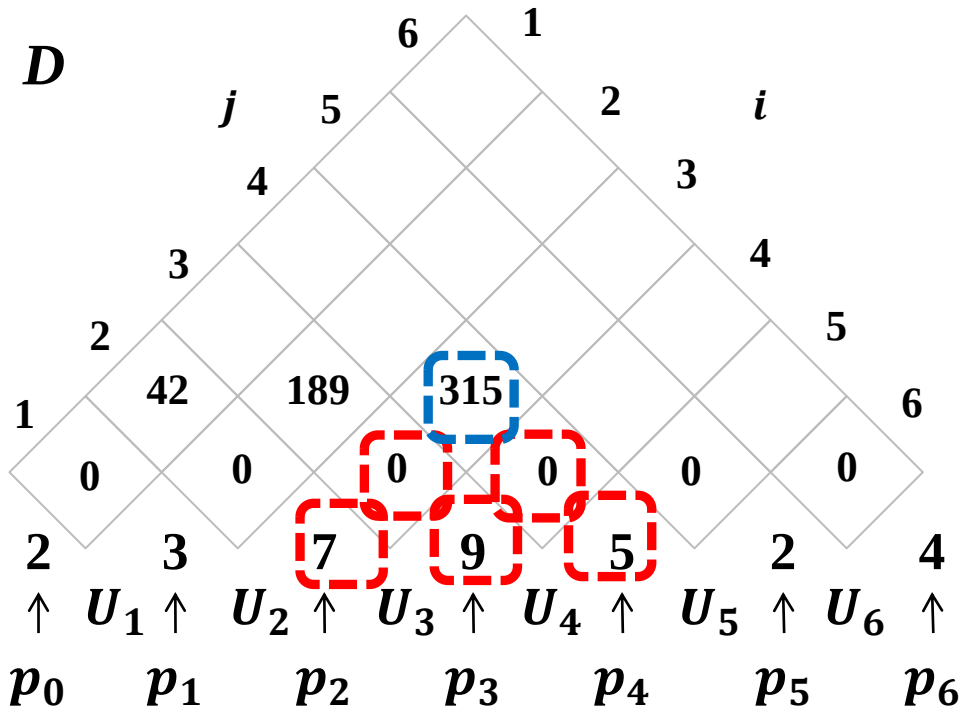
- $D[2,3] = \min_{2 \le k < 3} (D[2,k] + D[k+1,3] + p_1 p_k p_3) = 189$

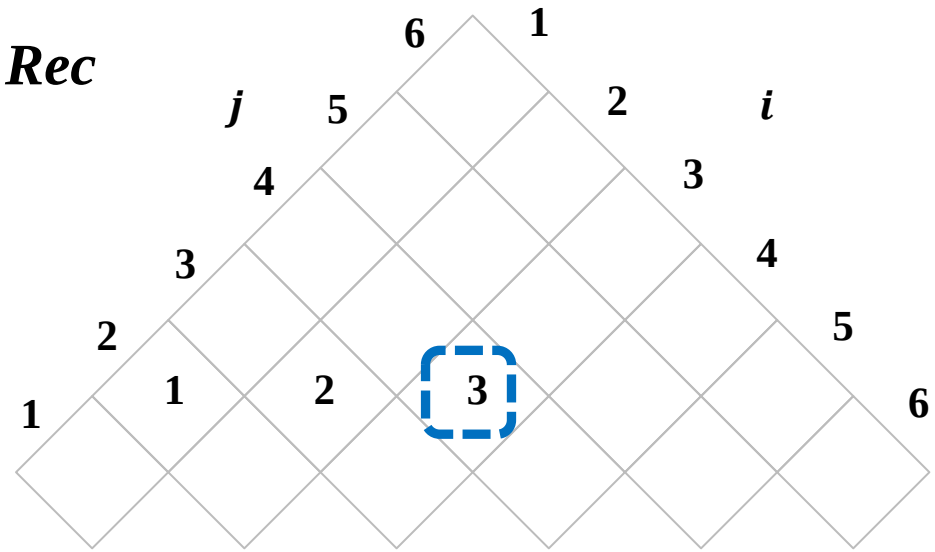




算法实例

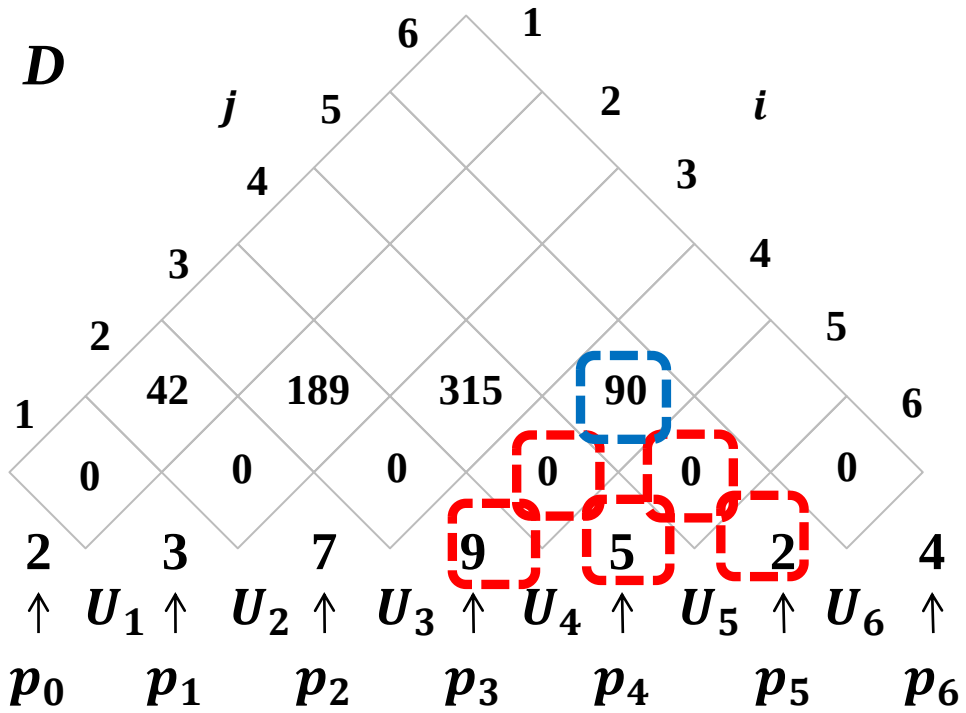
- $D[3,4] = \min_{3 \le k < 4} (D[3,k] + D[k+1,4] + p_2 p_k p_4) = 315$

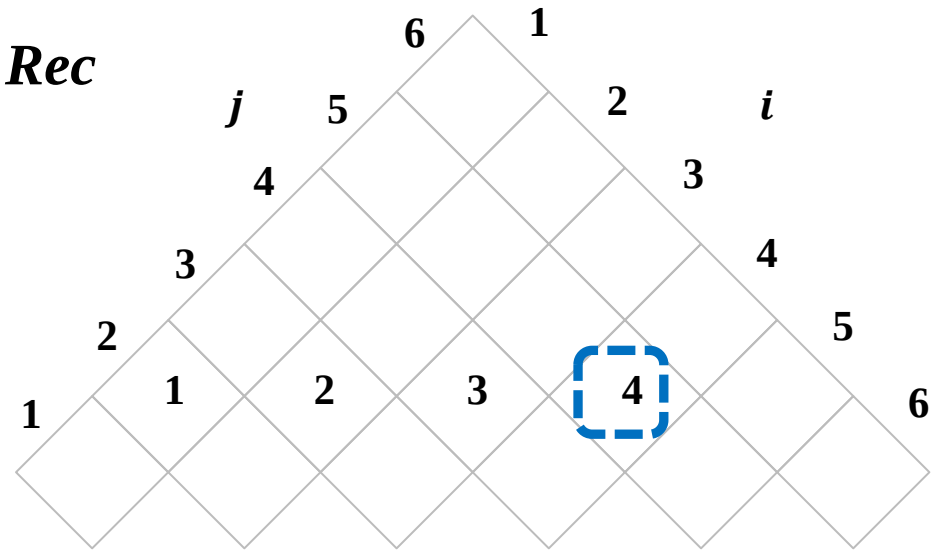




算法实例

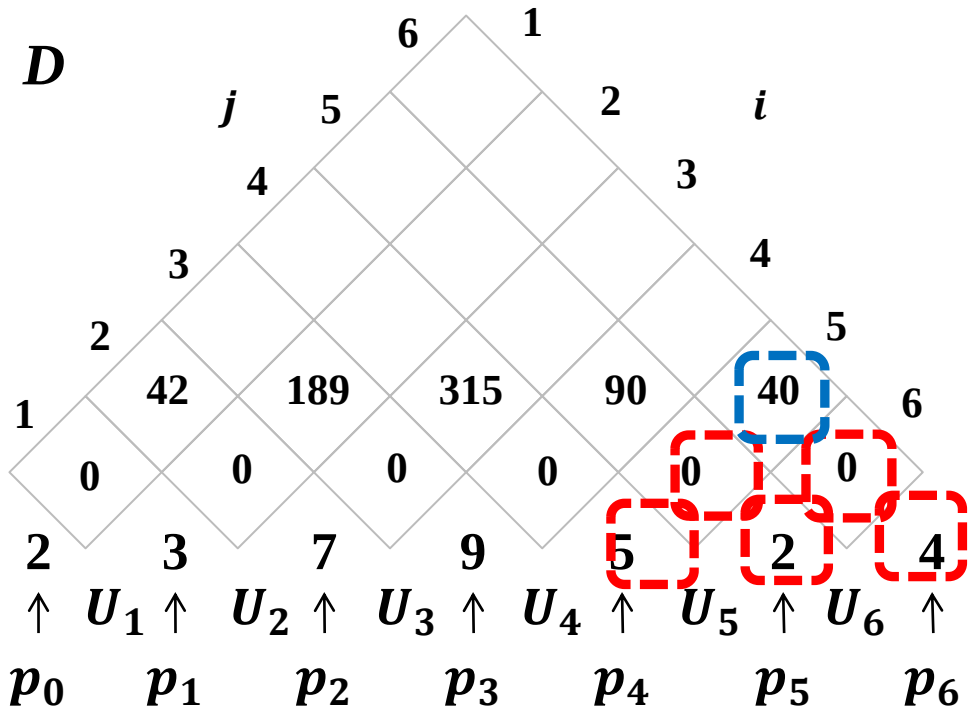
- $D[4,5] = \min_{4 \le k < 5}(D[4,k] + D[k+1,5] + p_3 p_k p_5) = 90$

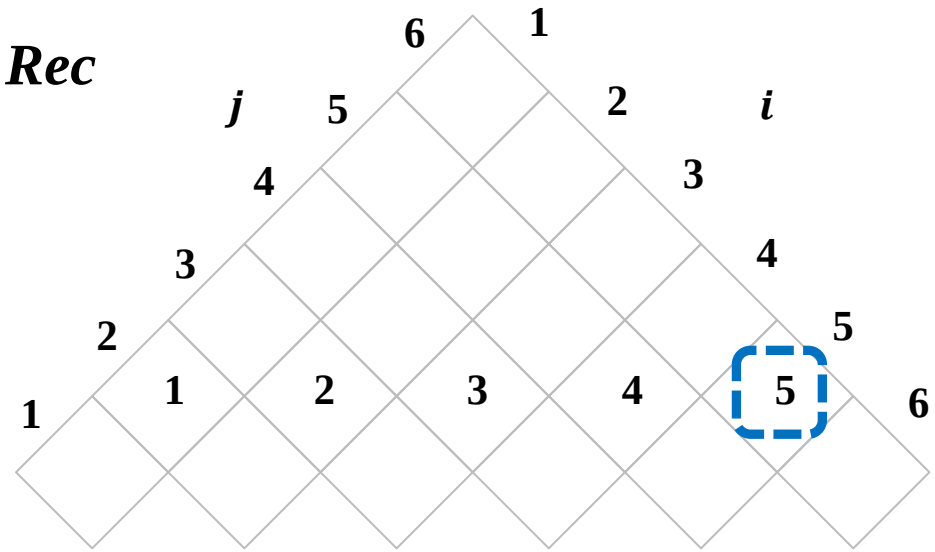




算法实例

- $D[5,6] = \min_{5 \le k < 6}(D[5,k] + D[k+1,6] + p_4 p_k p_6) = 40$

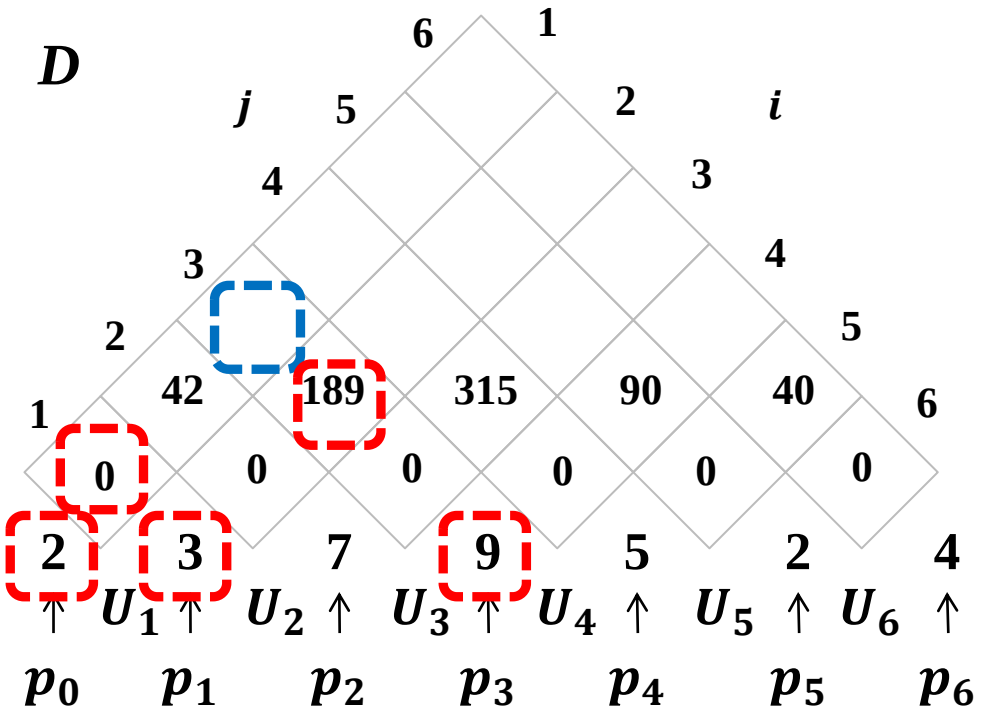


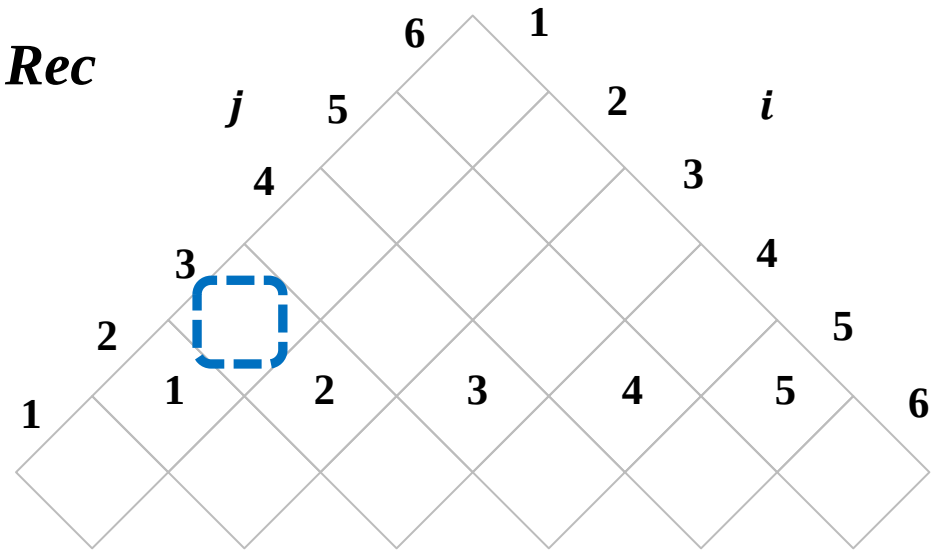


算法实例

- $D[1,3] = \min\limits_{1 \le k < 3} (D[1,k] + D[k+1,3] + p_0 p_k p_3)$

$$= \min \begin{cases} D[1,1] + D[2,3] + p_0 p_1 p_3 = 243 \\ D[1,2] + D[3,3] + p_0 p_2 p_3 = \end{cases}$$

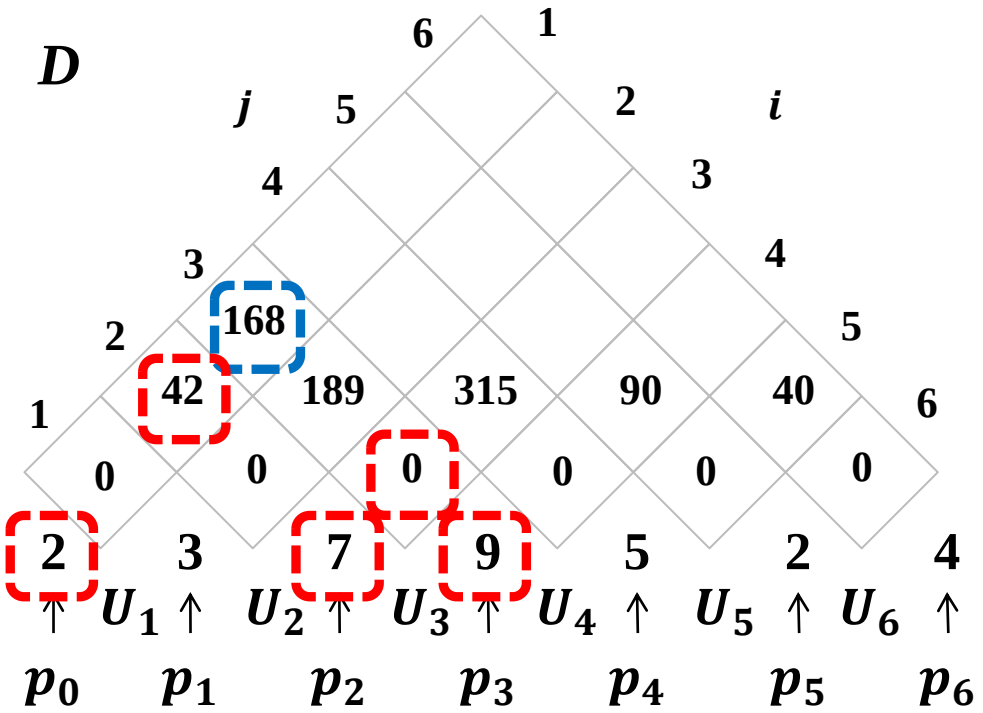


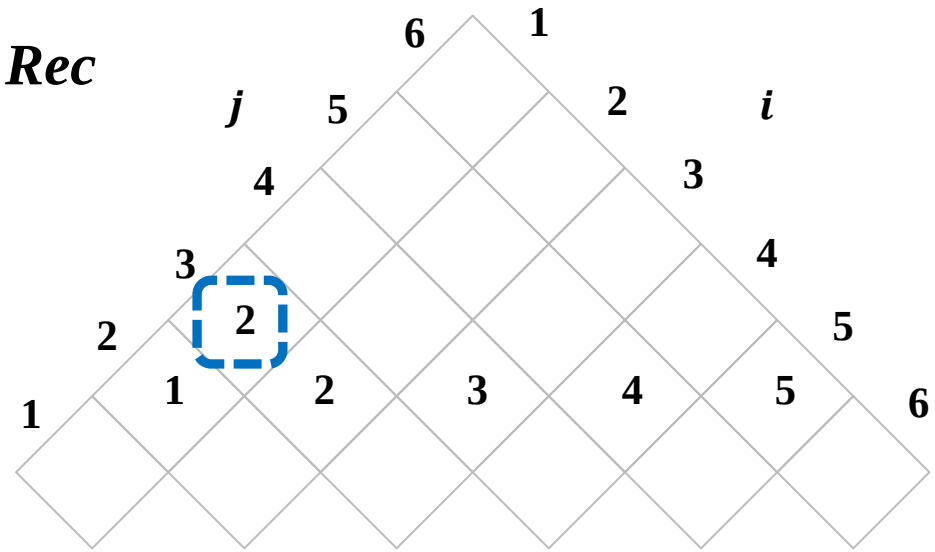


算法实例

- $D[1,3] = \min\limits_{1 \le k < 3} (D[1,k] + D[k+1,3] + p_0 p_k p_3)$

$$= \min \begin{cases} D[1,1] + D[2,3] + p_0 p_1 p_3 = 243 \\ D[1,2] + D[3,3] + p_0 p_2 p_3 = 168 \end{cases}$$

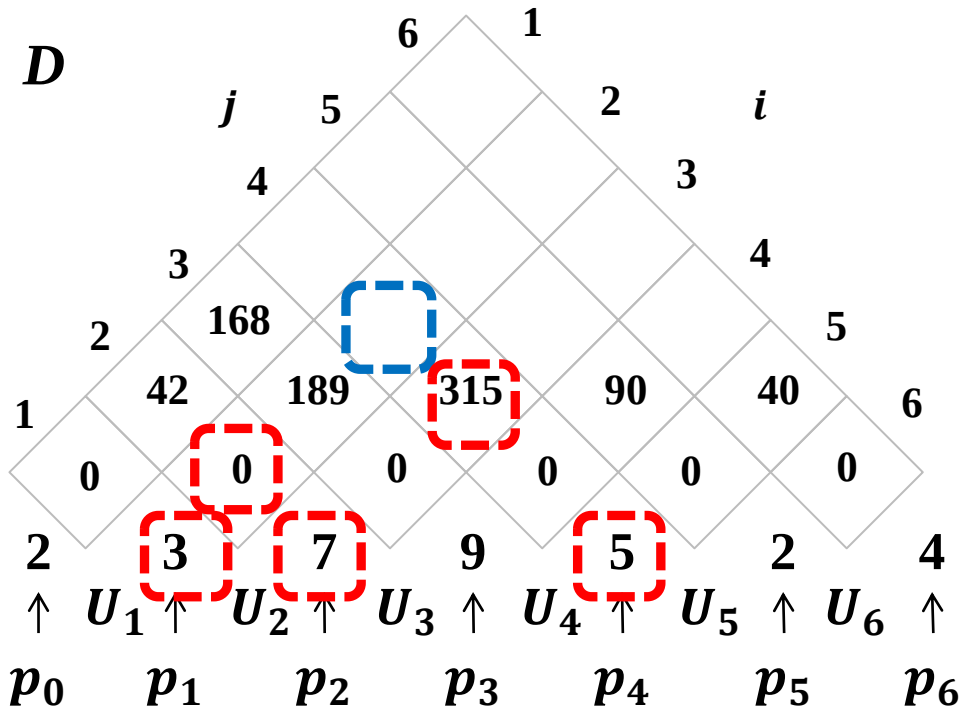


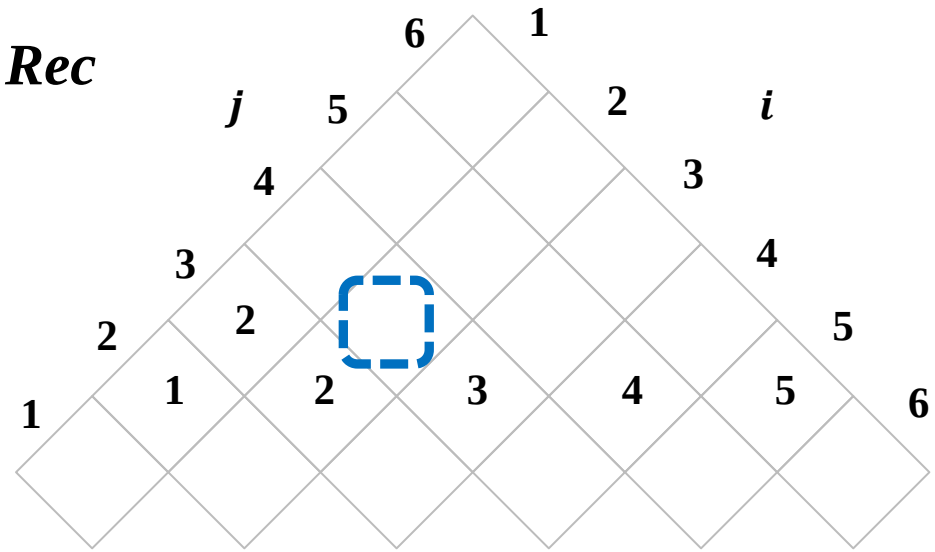


算法实例

- $D[2,4] = \min\limits_{2 \le k < 4}(D[2,k] + D[k+1,4] + p_1 p_k p_4)$

$$= \min \begin{cases} D[2,2] + D[3,4] + p_1 p_2 p_4 = 420 \\ D[2,3] + D[4,4] + p_1 p_3 p_4 = \end{cases}$$

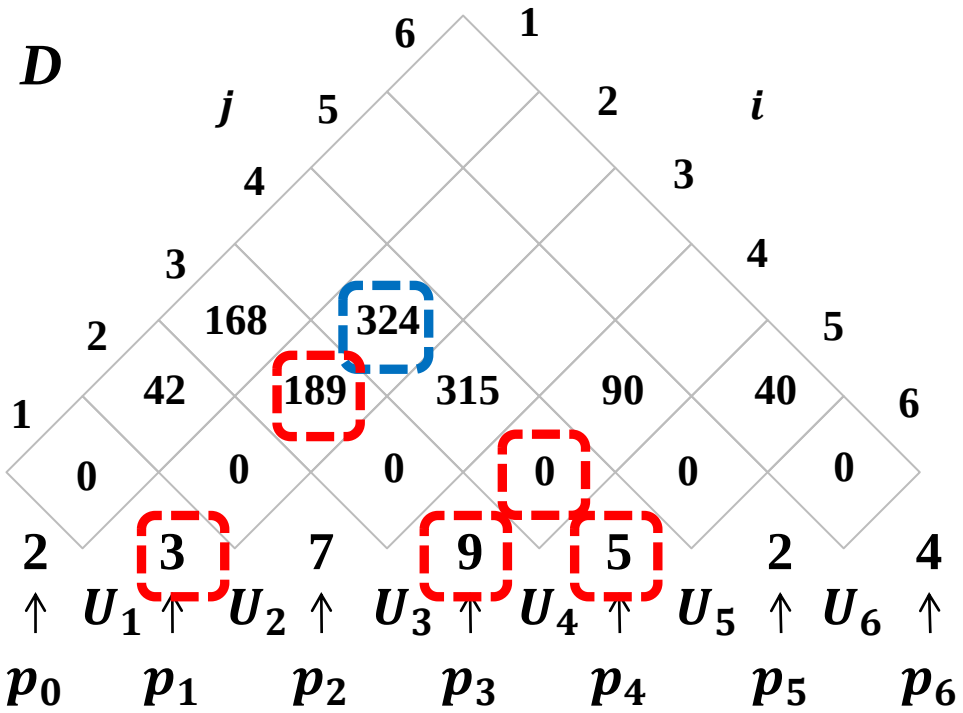


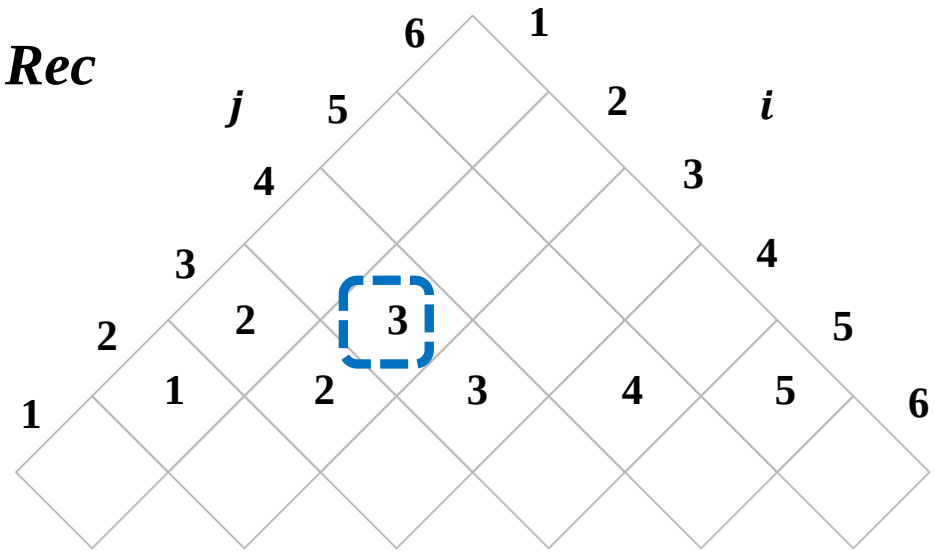


算法实例

- $D[2,4] = \min\limits_{2 \le k < 4} (D[2,k] + D[k+1,4] + p_1 p_k p_4)$

$$= \min \begin{cases} D[2,2] + D[3,4] + p_1 p_2 p_4 = 420 \\ D[2,3] + D[4,4] + p_1 p_3 p_4 = 324 \end{cases}$$

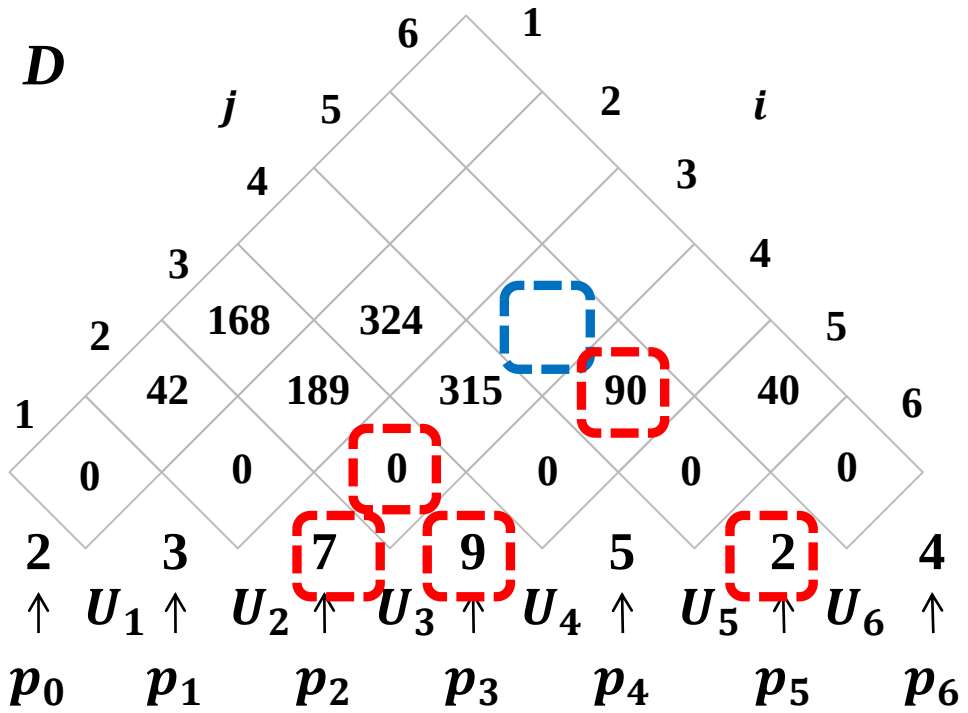


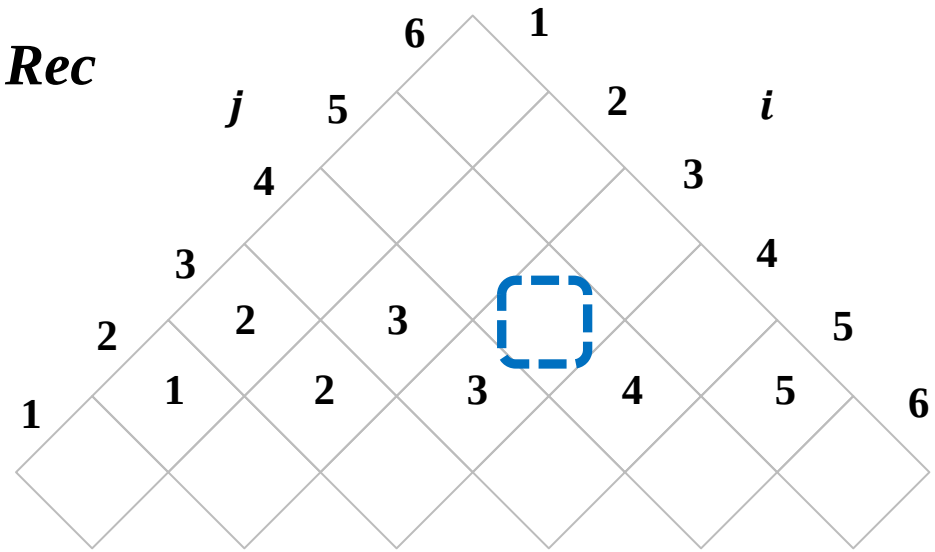


算法实例

- $D[3,5] = \min\limits_{3 \le k < 5}(D[3,k] + D[k+1,5] + p_2 p_k p_5)$

$$= \min \begin{cases} D[3,3] + D[4,5] + p_2 p_3 p_5 = 216 \\ D[3,4] + D[5,5] + p_2 p_4 p_5 = \end{cases}$$

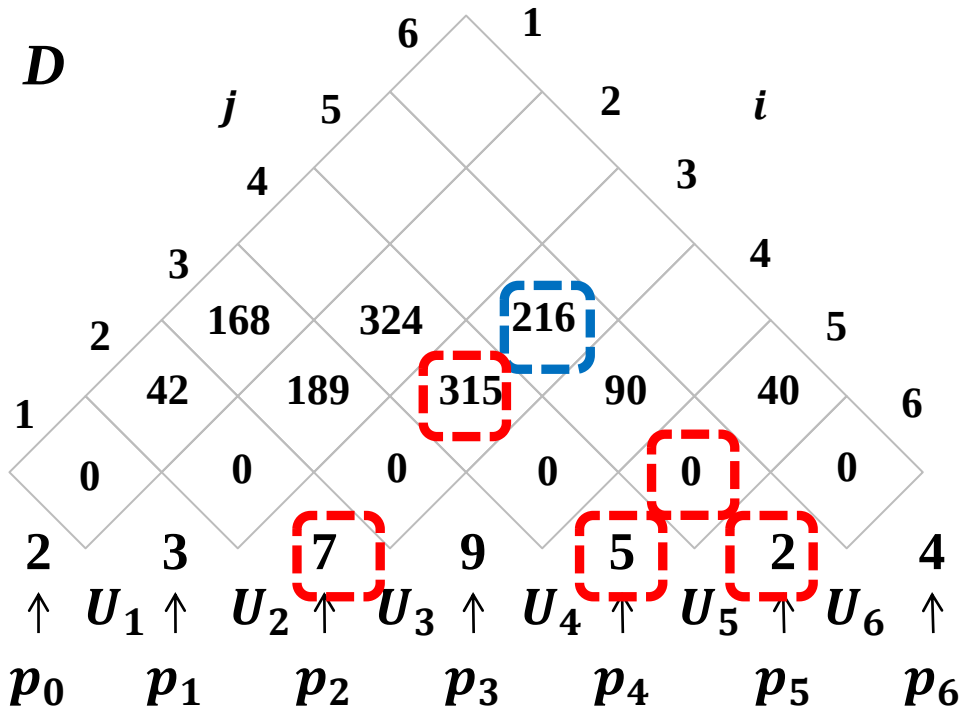


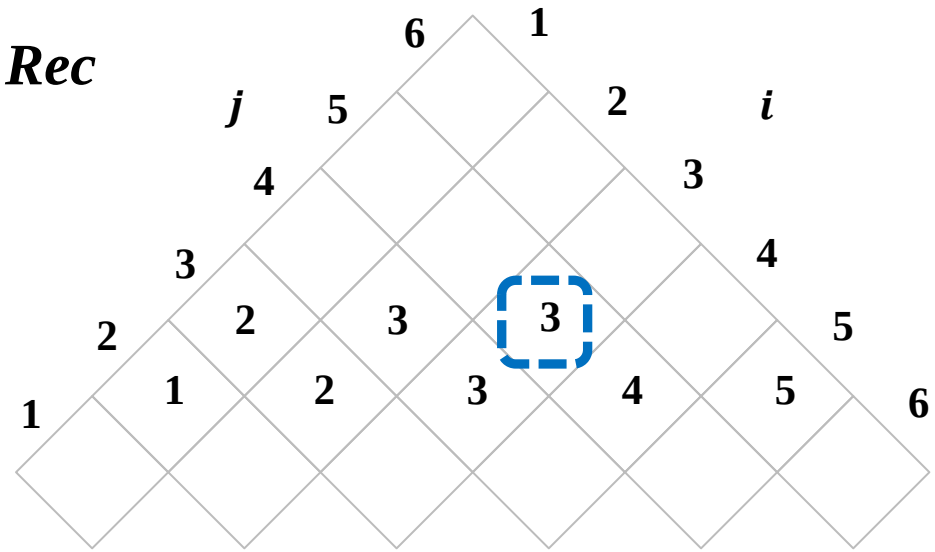


算法实例

- $D[3,5] = \min\limits_{3 \le k < 5}(D[3,k] + D[k+1,5] + p_2 p_k p_5)$

$$= \min \begin{cases} D[3,3] + D[4,5] + p_2 p_3 p_5 = 216 \\ D[3,4] + D[5,5] + p_2 p_4 p_5 = 385 \end{cases}$$

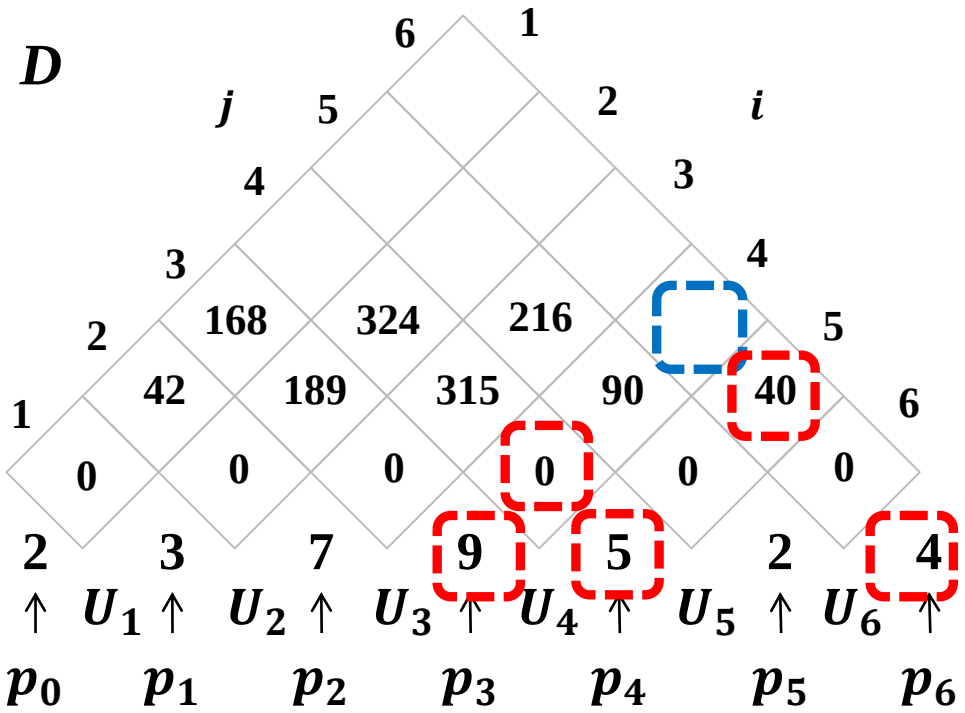


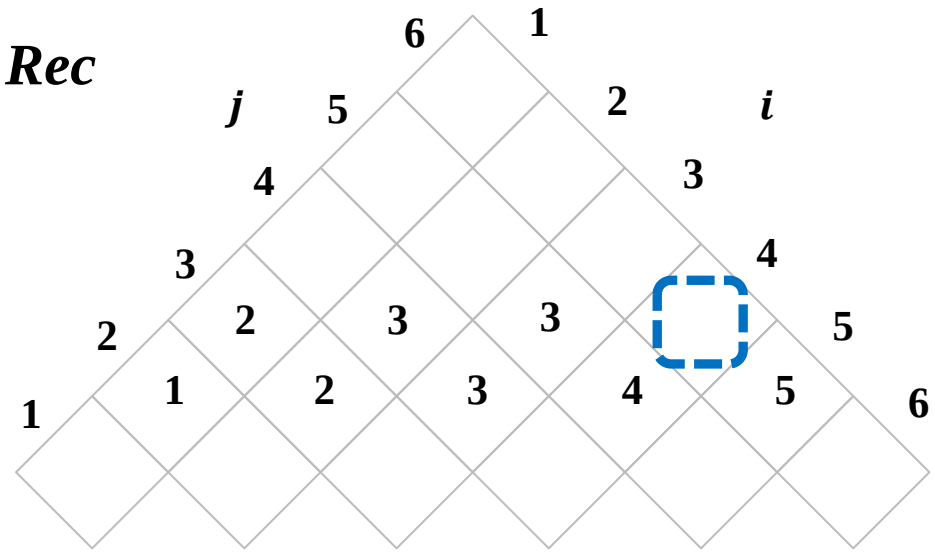


算法实例

- $D[4,6] = \min\limits_{4 \le k < 6}(D[4,k] + D[k+1,6] + p_3 p_k p_6)$

$$= \min \begin{cases} D[4,4] + D[5,6] + p_3 p_4 p_6 = 220 \\ D[4,5] + D[6,6] + p_3 p_5 p_6 = \end{cases}$$

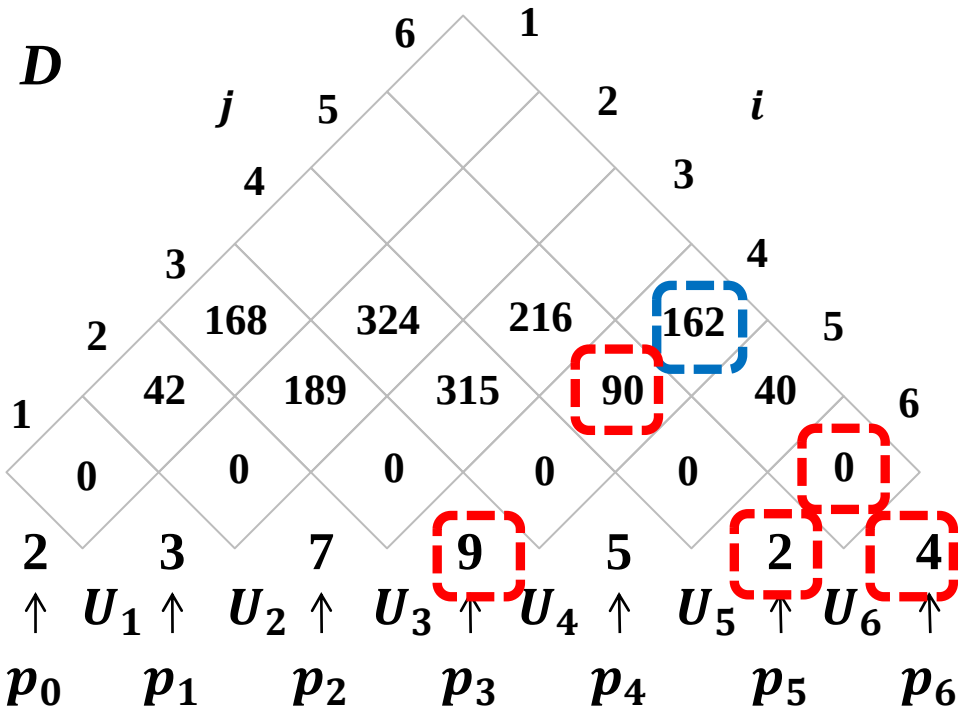


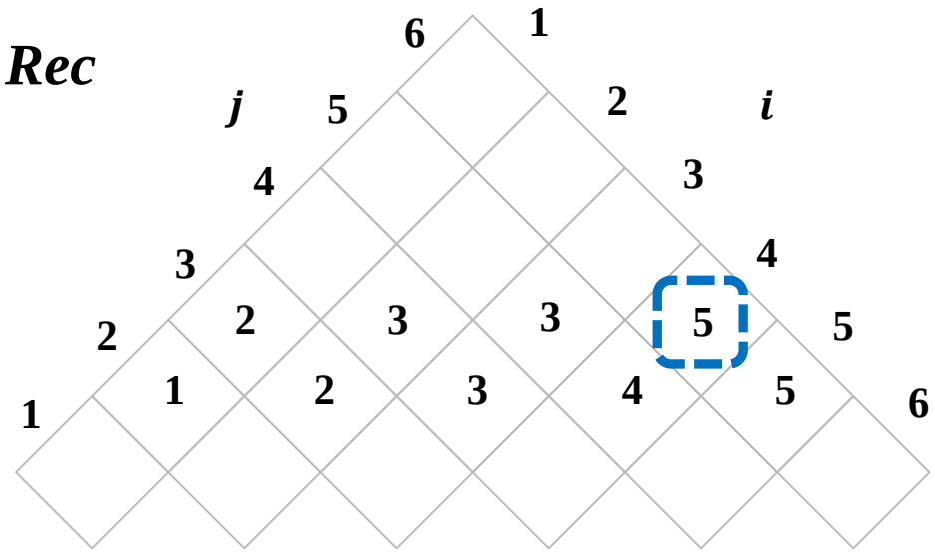


算法实例

- $D[4,6] = \min\limits_{4 \le k < 6}(D[4,k] + D[k+1,6] + p_3 p_k p_6)$

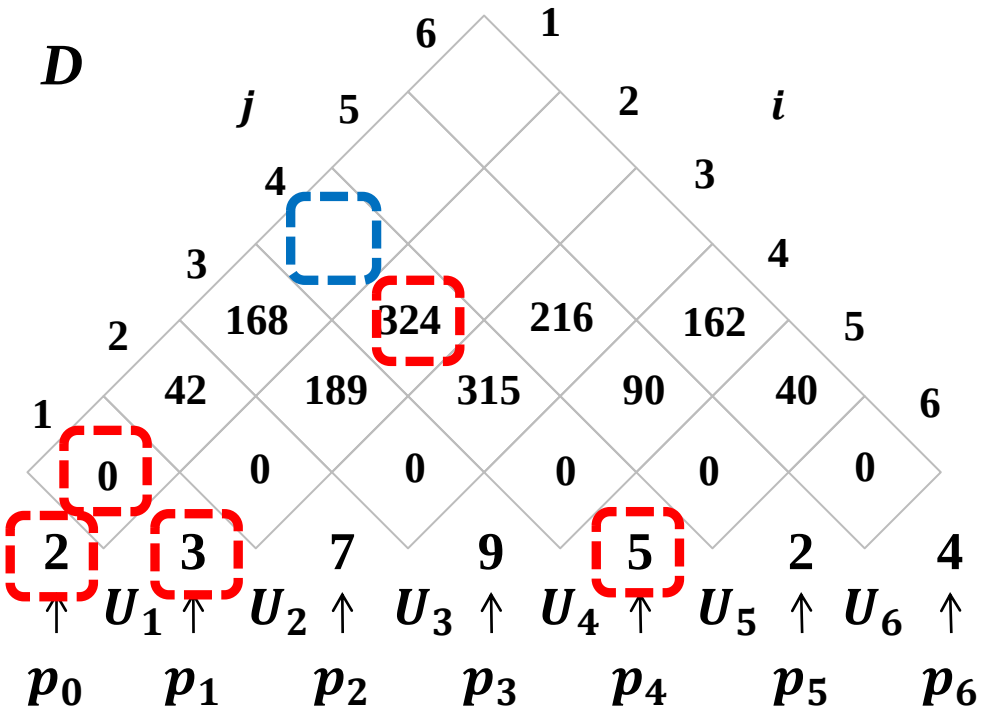
$$= \min \begin{cases} D[4,4] + D[5,6] + p_3 p_4 p_6 = 220 \\ D[4,5] + D[6,6] + p_3 p_5 p_6 = 162 \end{cases}$$

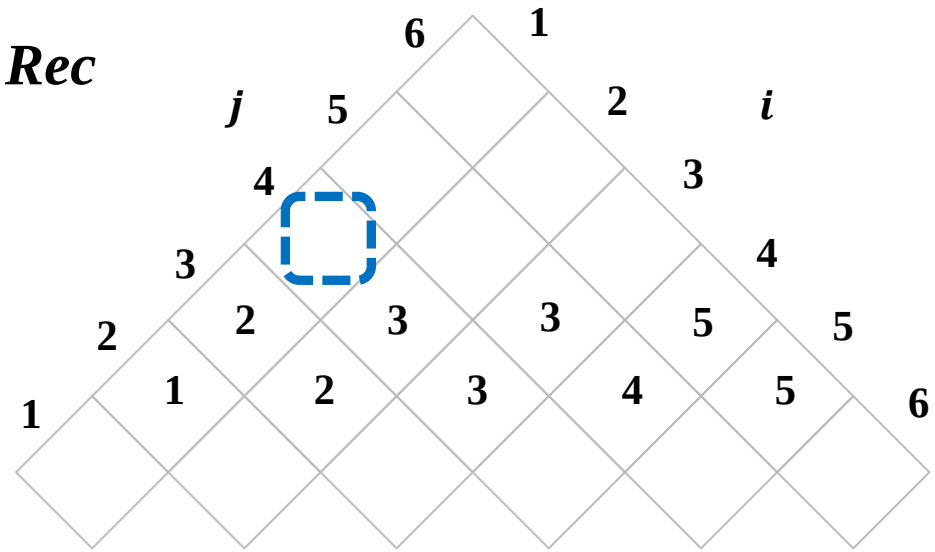




算法实例

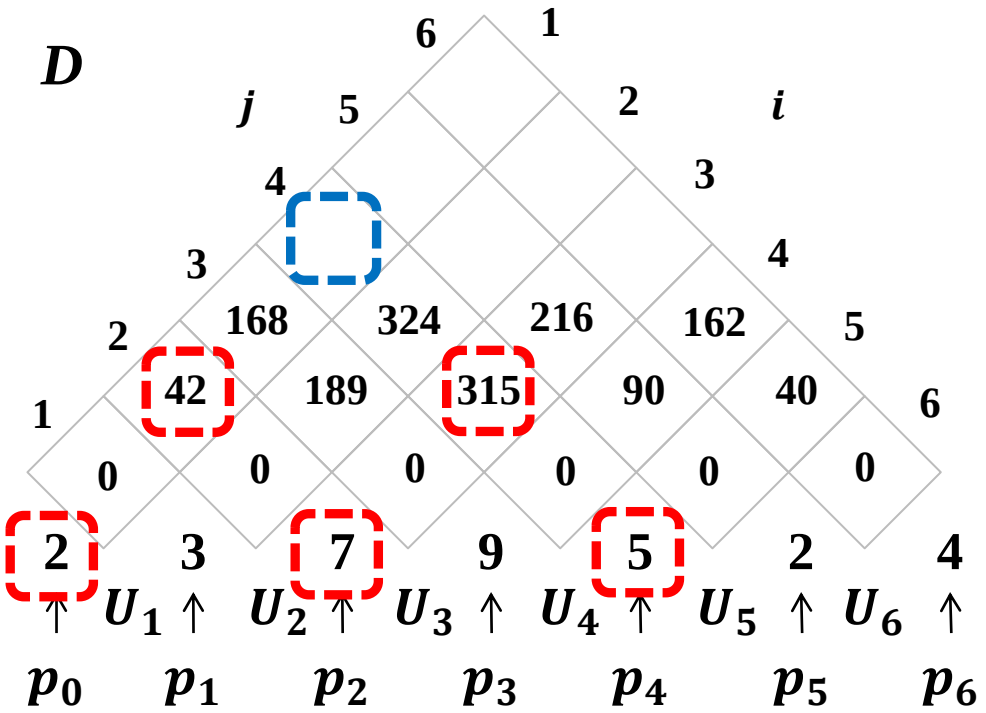
- $D[1,4] = \min \begin{cases} D[1,1] + D[2,4] + p_0 p_1 p_4 = 354 \\ D[1,2] + D[3,4] + p_0 p_2 p_4 = \\ D[1,3] + D[4,4] + p_0 p_3 p_4 = \end{cases}$

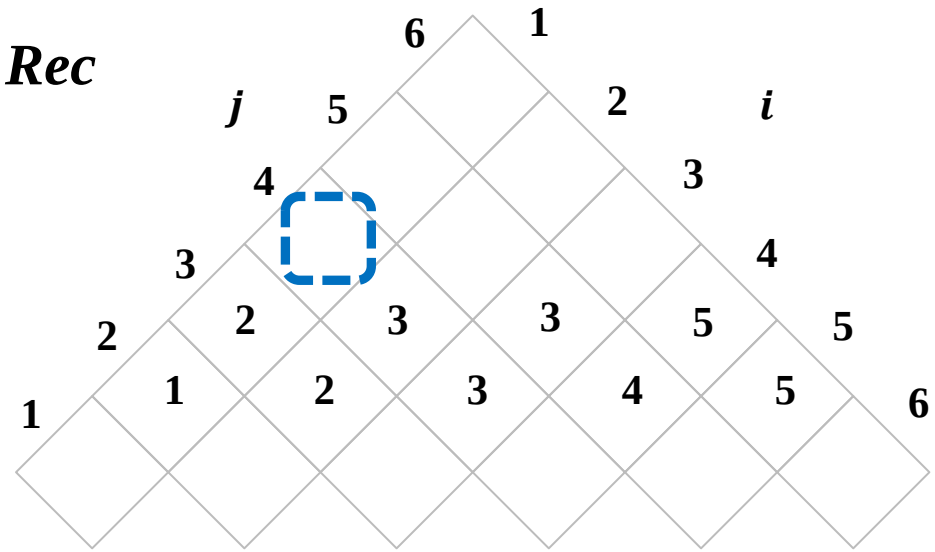




算法实例

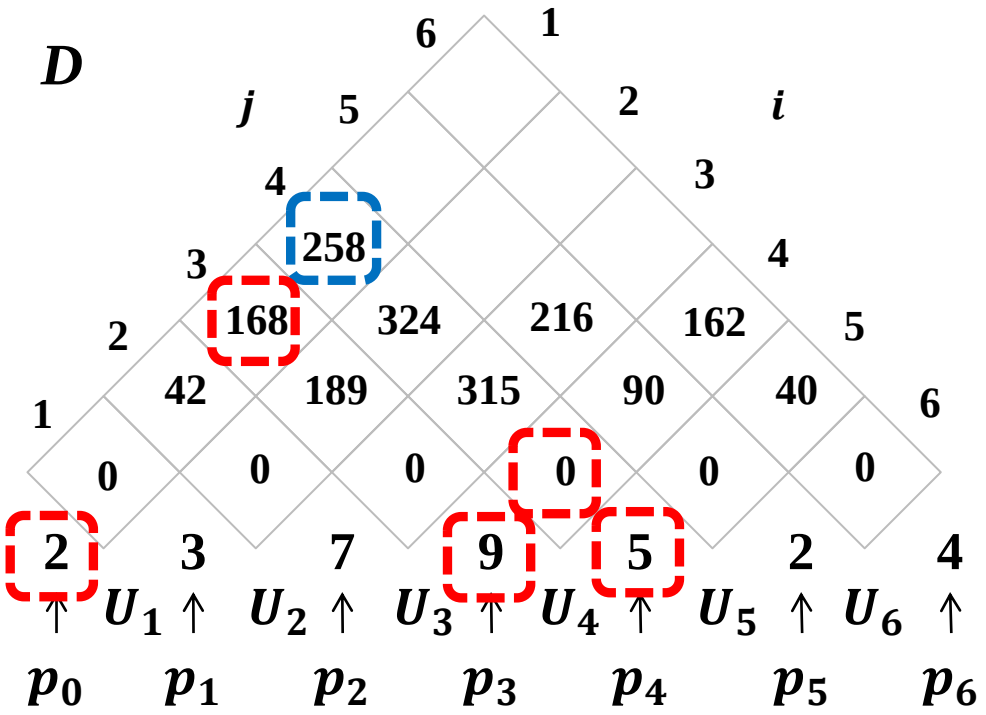
- $D[1,4] = \min \begin{cases} D[1,1] + D[2,4] + p_0 p_1 p_4 = 354 \\ D[1,2] + D[3,4] + p_0 p_2 p_4 = 427 \\ D[1,3] + D[4,4] + p_0 p_3 p_4 = \end{cases}$

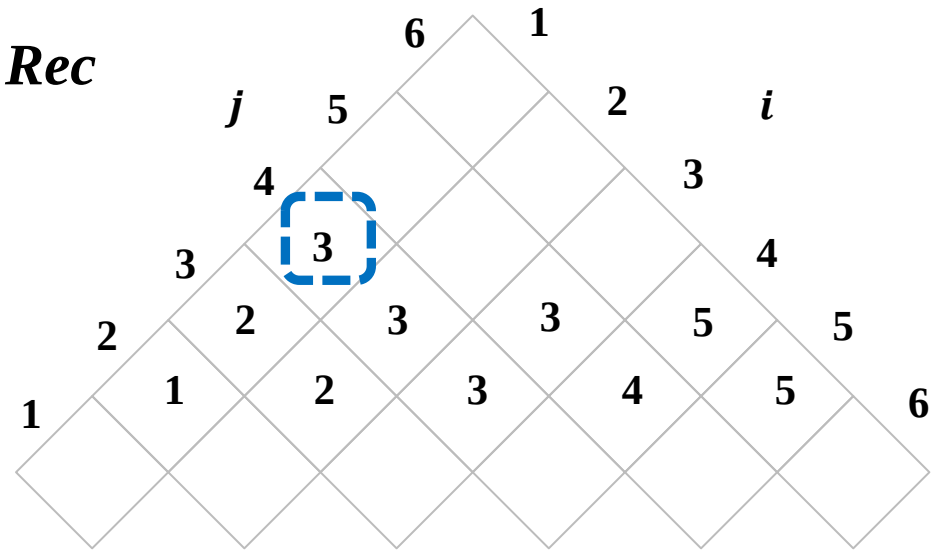




算法实例

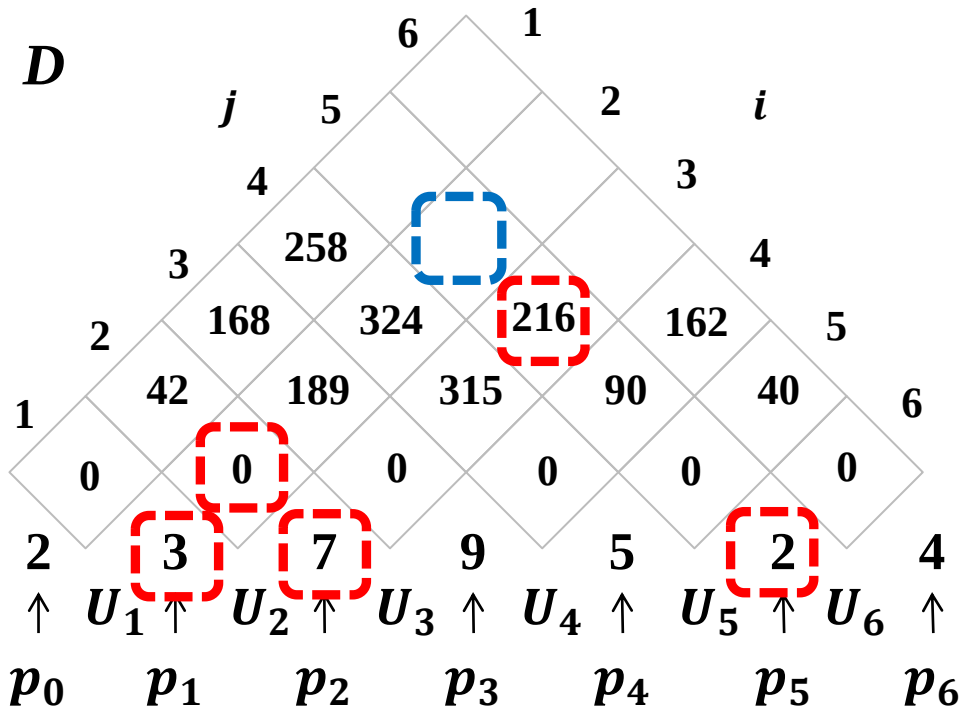
- $$D[1,4]=\min\begin{cases}D[1,1]+D[2,4]+p_0p_1p_4=354\\ D[1,2]+D[3,4]+p_0p_2p_4=427\\ D[1,3]+D[4,4]+p_0p_3p_4=258\end{cases}$$

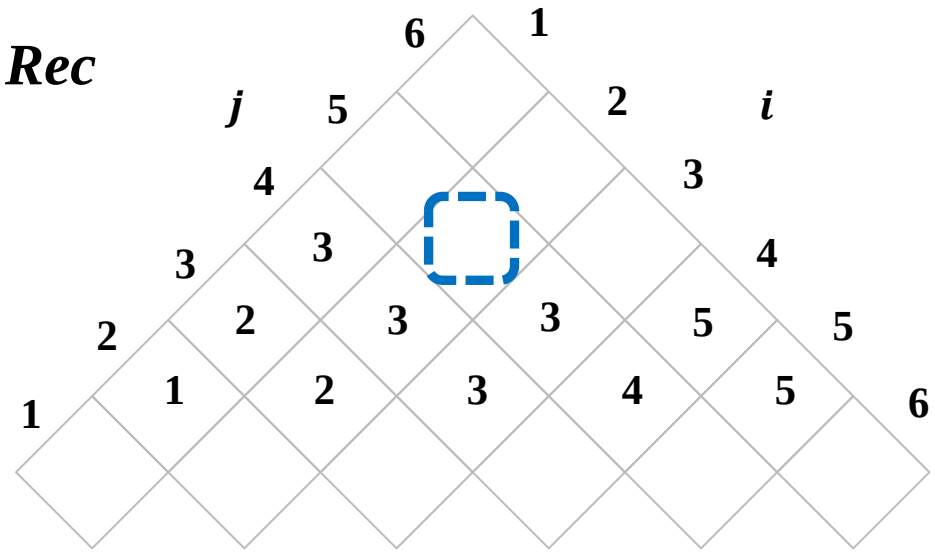




算法实例

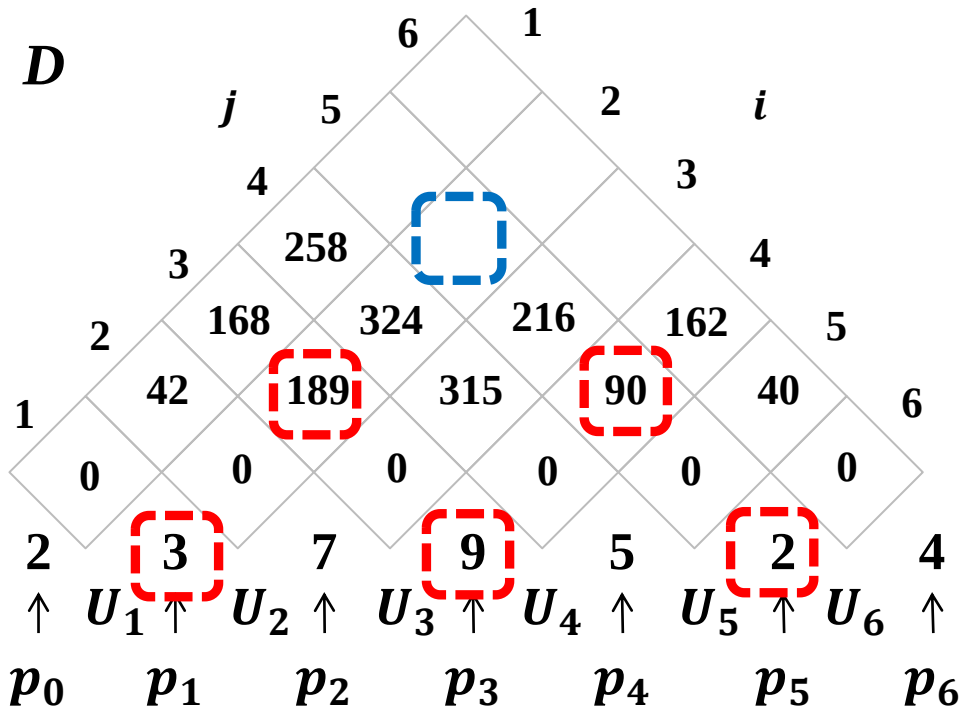
- $$D[2,5]=\min\begin{cases}D[2,2]+D[3,5]+p_1p_2p_5=258\\D[2,3]+D[4,5]+p_1p_3p_5=\\D[2,4]+D[5,5]+p_1p_4p_5=\end{cases}$$

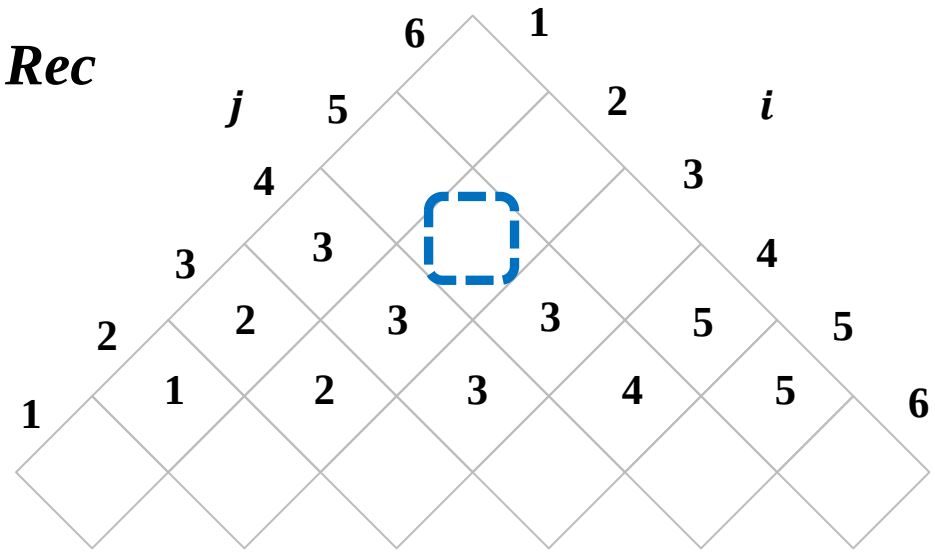




算法实例

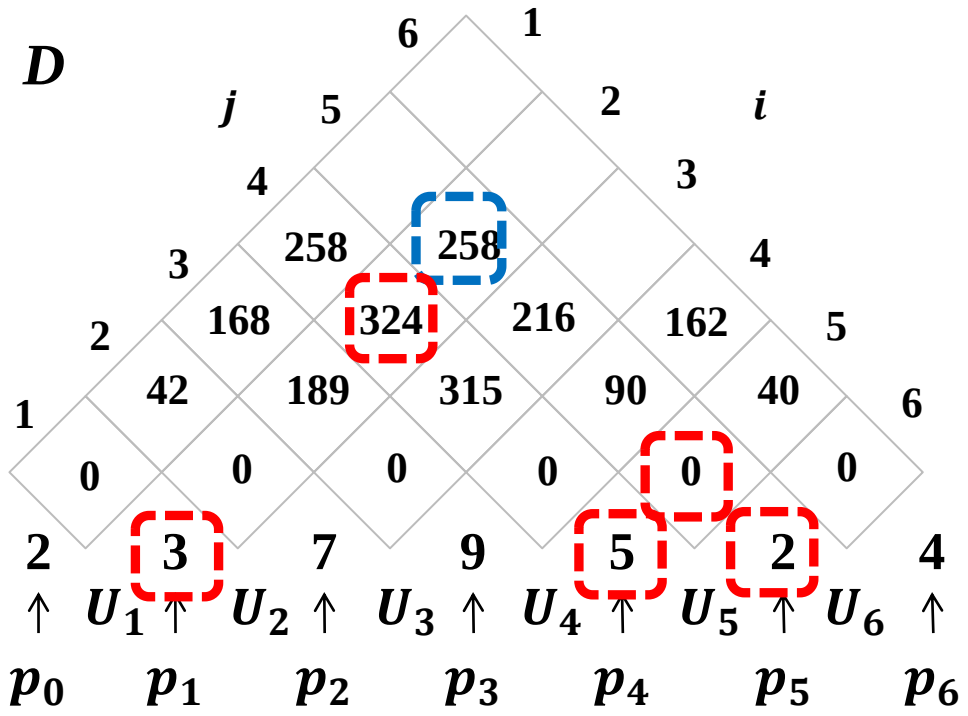
- $D[2,5] = \min \begin{cases} D[2,2] + D[3,5] + p_1 p_2 p_5 = 258 \\ D[2,3] + D[4,5] + p_1 p_3 p_5 = 333 \\ D[2,4] + D[5,5] + p_1 p_4 p_5 = \end{cases}$

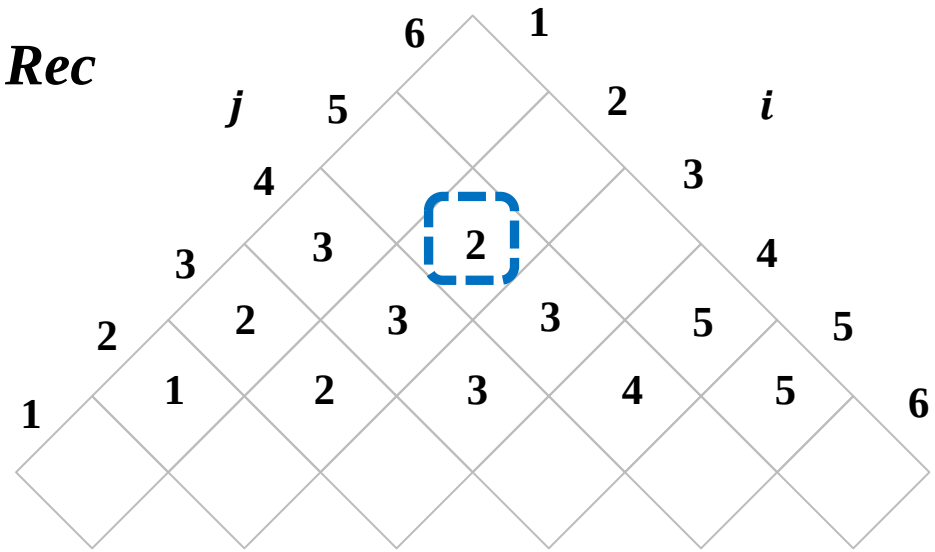




算法实例

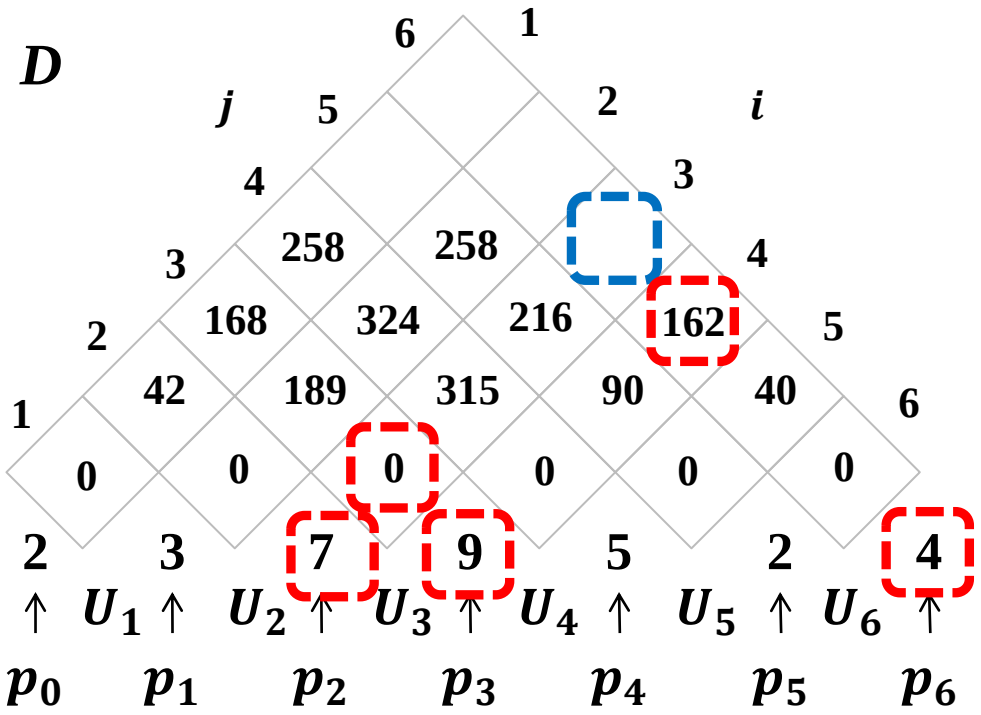
- $$D[2,5]=\min\begin{cases}D[2,2]+D[3,5]+p_1p_2p_5=258\\ D[2,3]+D[4,5]+p_1p_3p_5=333\\ D[2,4]+D[5,5]+p_1p_4p_5=354\end{cases}$$

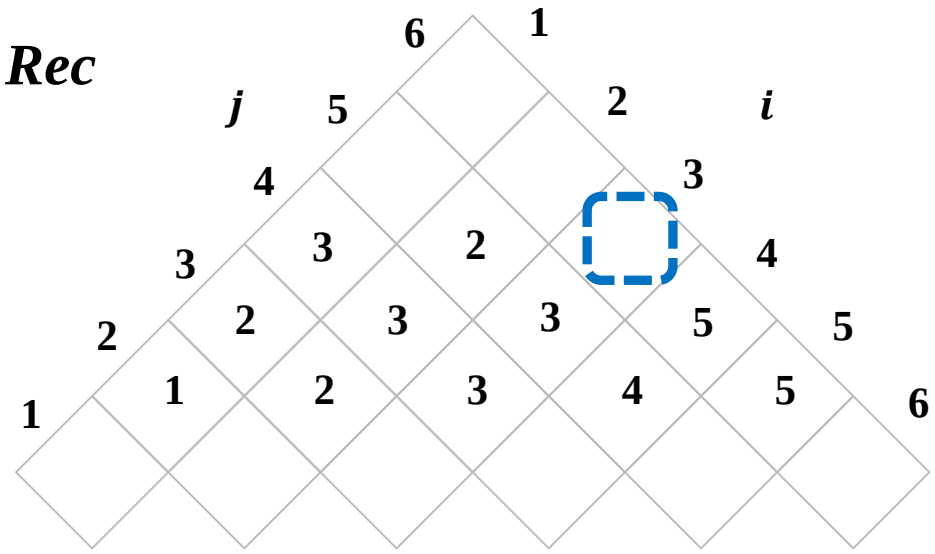




算法实例

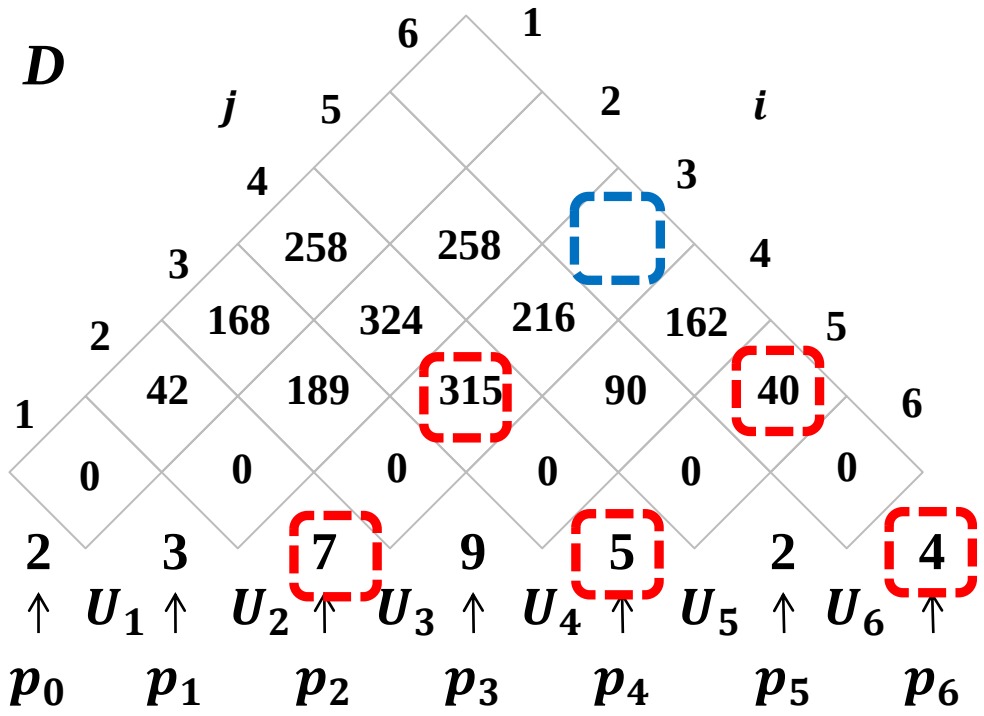
- $$D[3,6]=\min\begin{cases}D[3,3]+D[4,6]+p_2p_3p_6=414\\ D[3,4]+D[5,6]+p_2p_4p_6=\\ D[3,5]+D[6,6]+p_2p_5p_6=\end{cases}$$

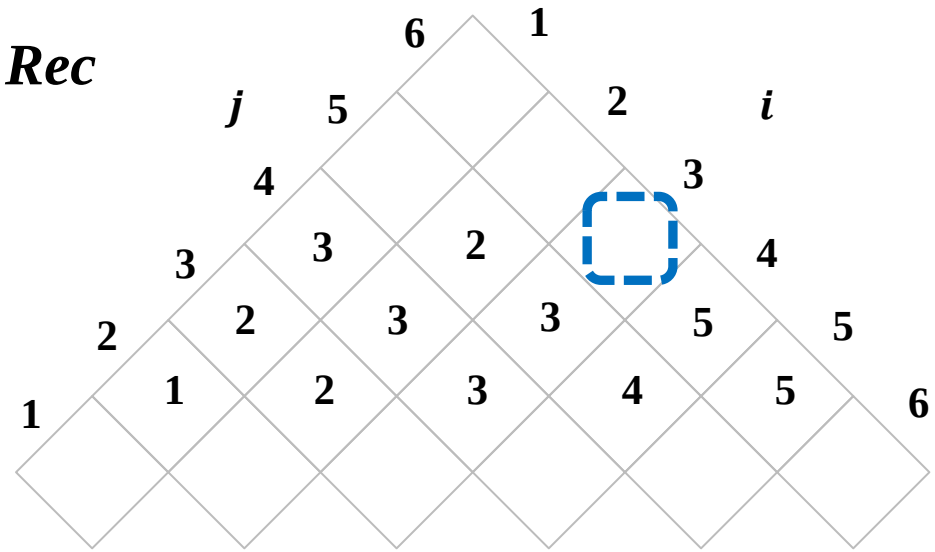




算法实例

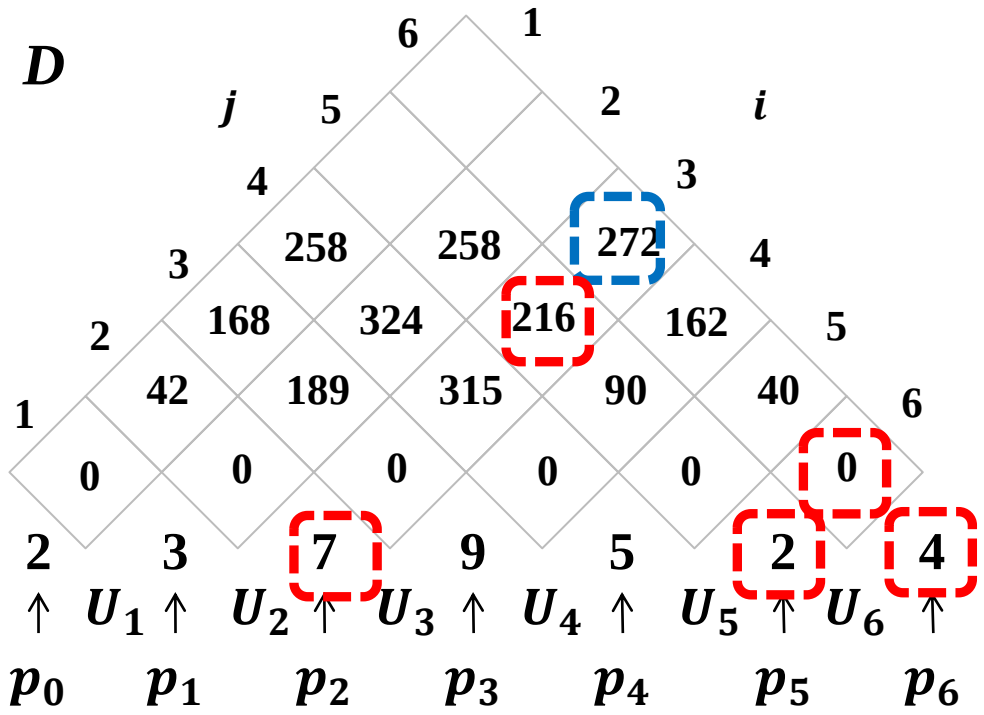
- $D[3,6] = \min \begin{cases} D[3,3] + D[4,6] + p_2 p_3 p_6 = 414 \\ D[3,4] + D[5,6] + p_2 p_4 p_6 = 495 \\ D[3,5] + D[6,6] + p_2 p_5 p_6 = \end{cases}$

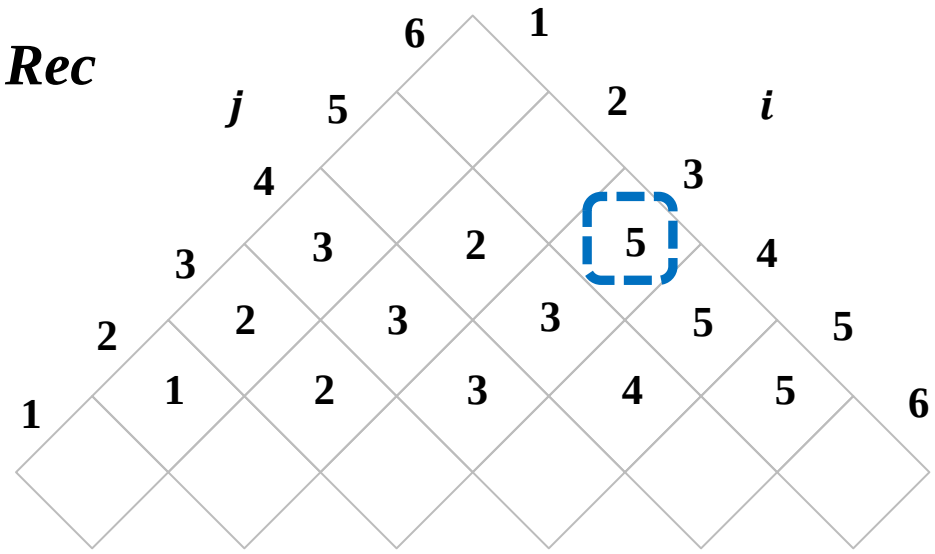




算法实例

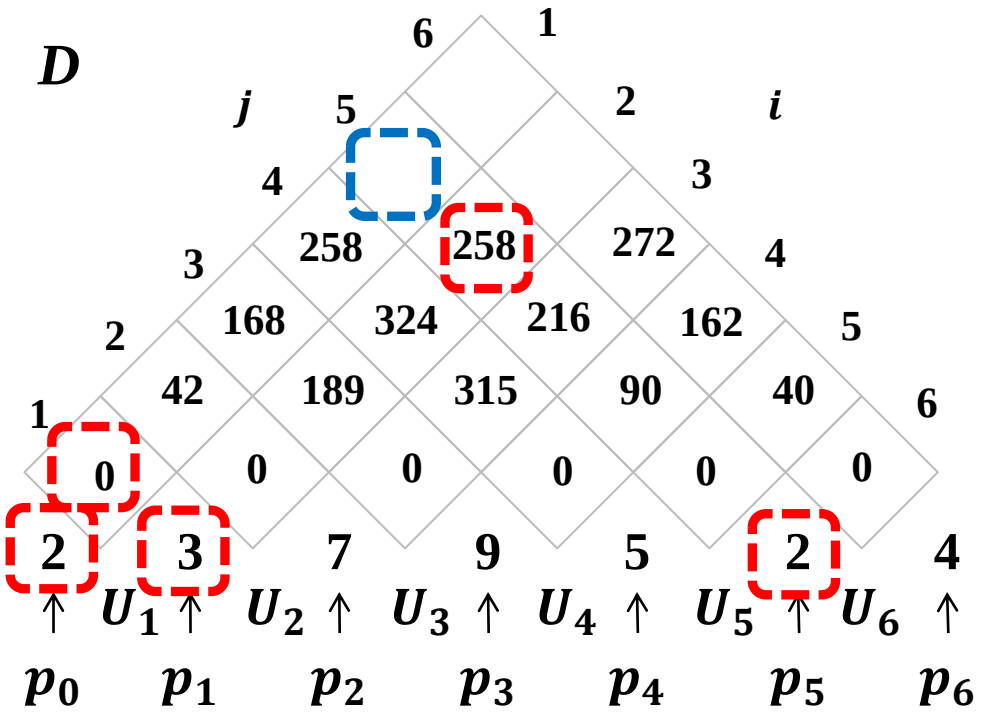
- $$D[3,6]=\min\begin{cases}D[3,3]+D[4,6]+p_2p_3p_6=414\\ D[3,4]+D[5,6]+p_2p_4p_6=495\\ D[3,5]+D[6,6]+p_2p_5p_6=272\end{cases}$$

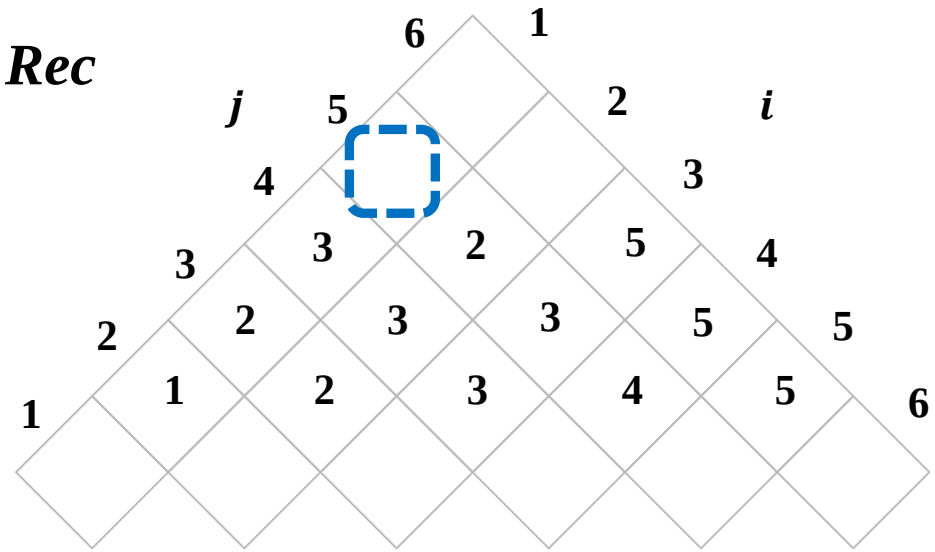




算法实例

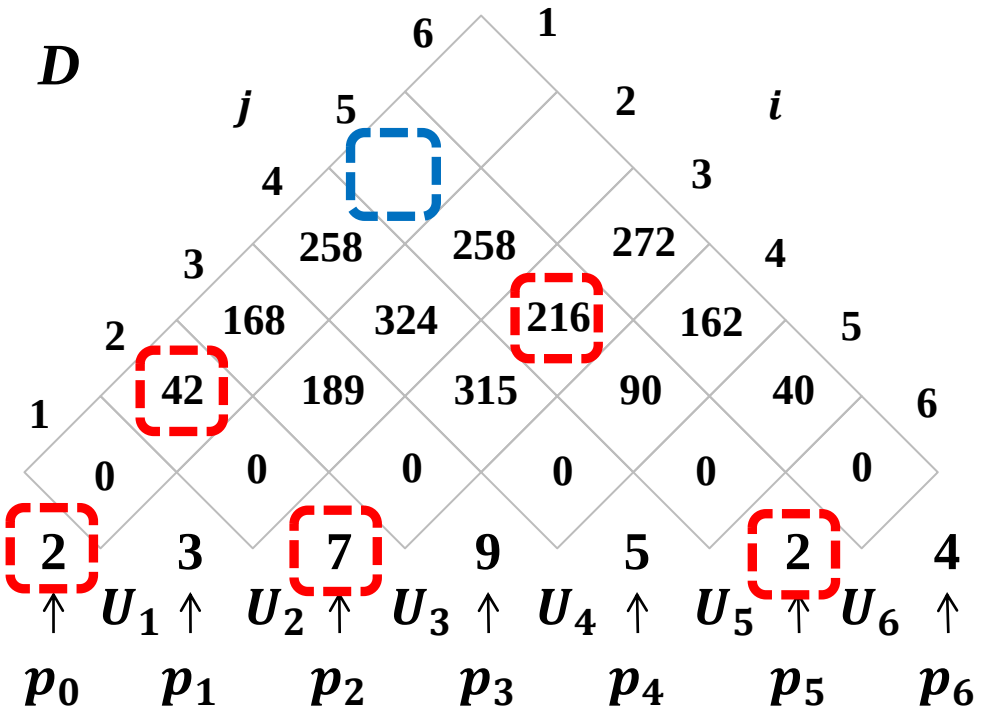
- $D[1,5] = \min \begin{cases} D[1,1] + D[2,5] + p_0 p_1 p_5 = 270 \\ D[1,2] + D[3,5] + p_0 p_2 p_5 = \\ D[1,3] + D[4,5] + p_0 p_3 p_5 = \\ D[1,4] + D[5,5] + p_0 p_4 p_5 = \end{cases}$

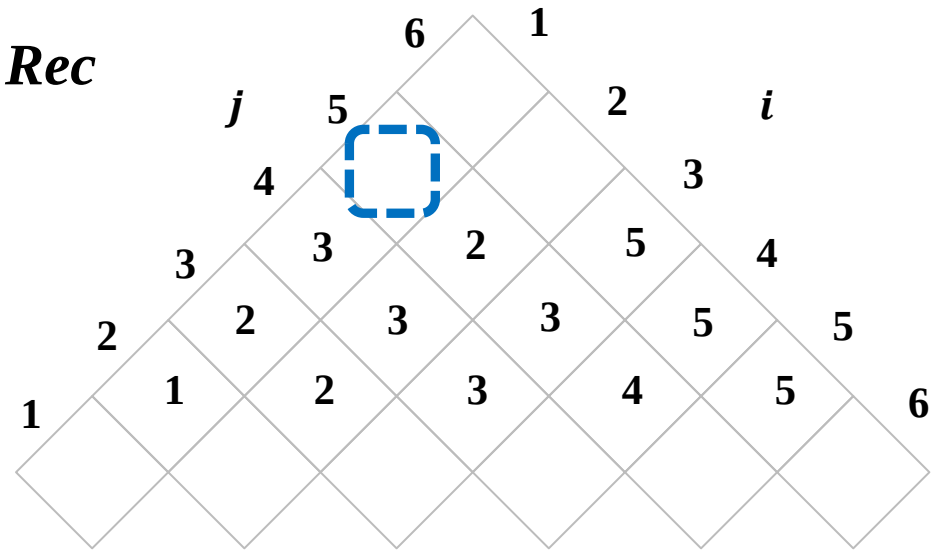




算法实例

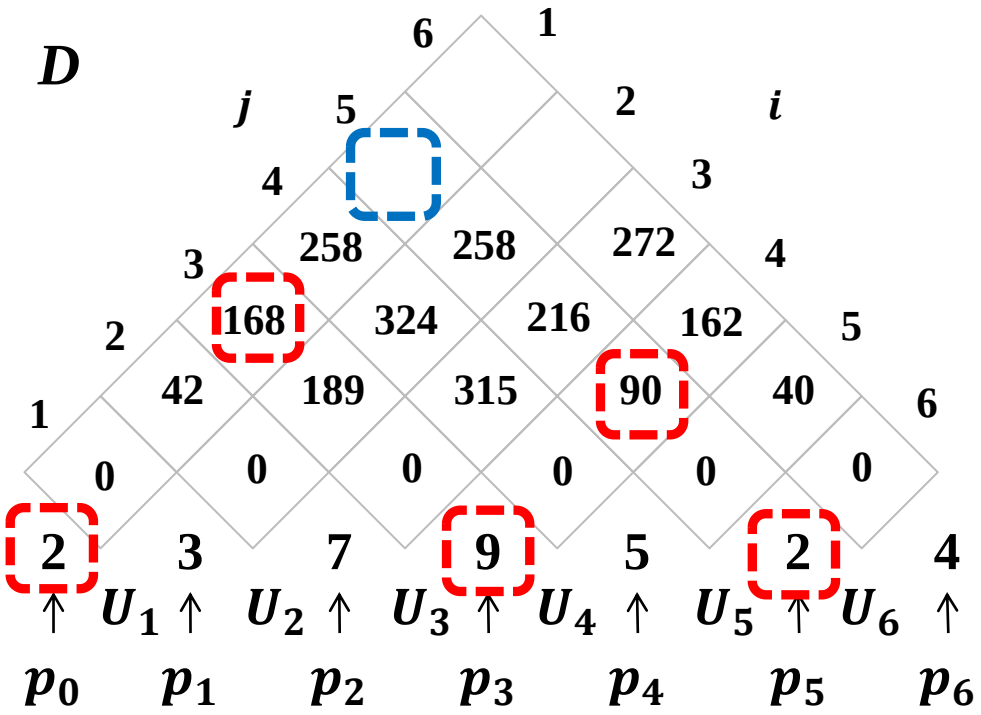
- $D[1,5] = \min \begin{cases} D[1,1] + D[2,5] + p_0 p_1 p_5 = 270 \\ D[1,2] + D[3,5] + p_0 p_2 p_5 = 286 \\ D[1,3] + D[4,5] + p_0 p_3 p_5 = \\ D[1,4] + D[5,5] + p_0 p_4 p_5 = \end{cases}$

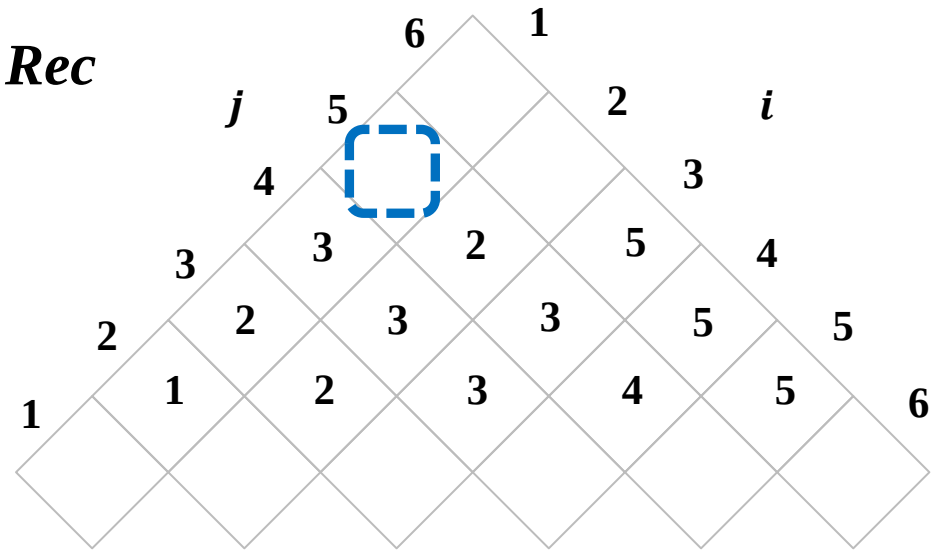




算法实例

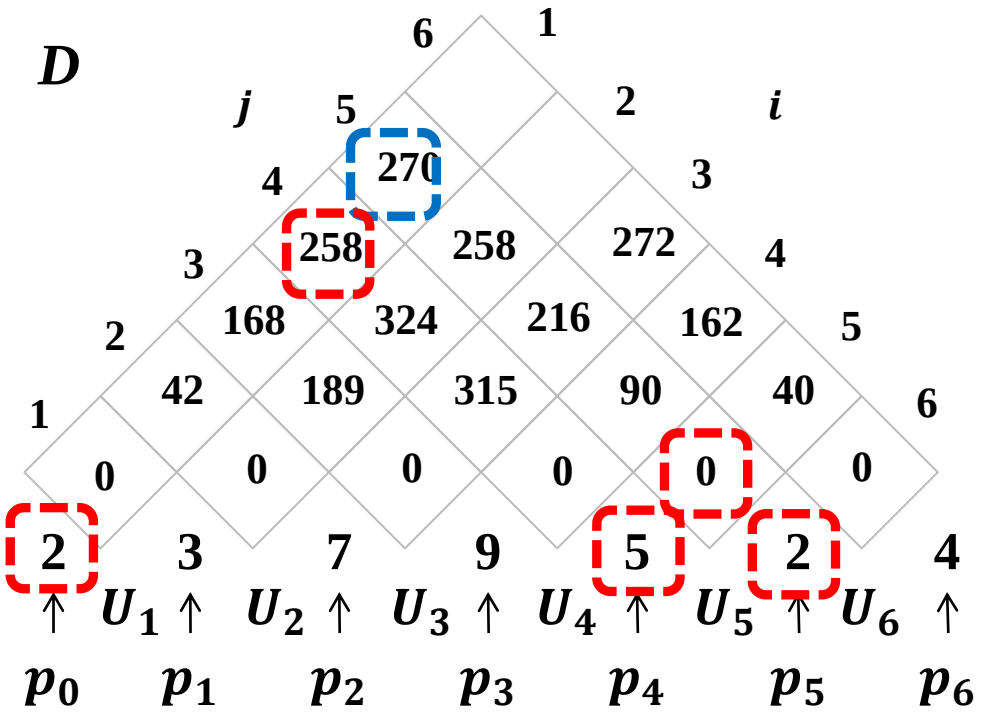
- $D[1,5] = \min \begin{cases} D[1,1] + D[2,5] + p_0 p_1 p_5 = 270 \\ D[1,2] + D[3,5] + p_0 p_2 p_5 = 286 \\ D[1,3] + D[4,5] + p_0 p_3 p_5 = 294 \\ D[1,4] + D[5,5] + p_0 p_4 p_5 = \end{cases}$

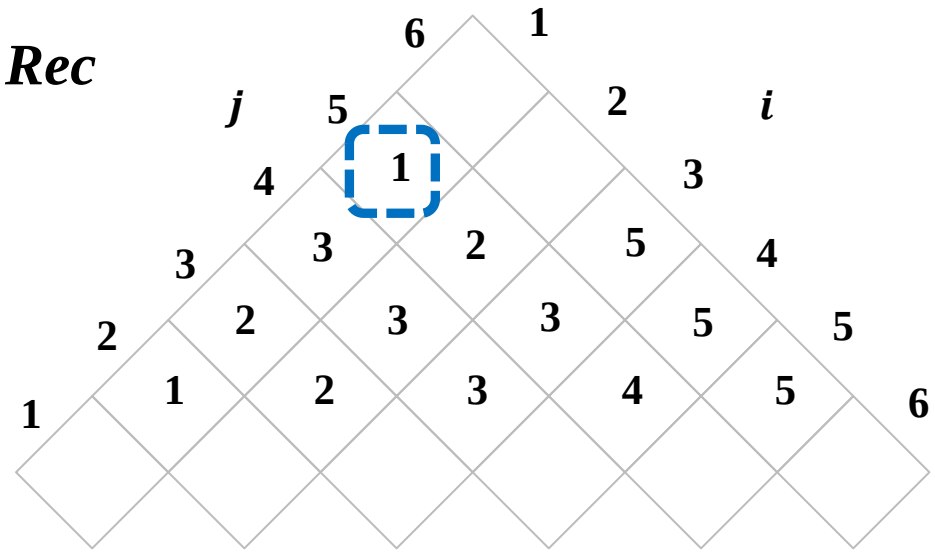




算法实例

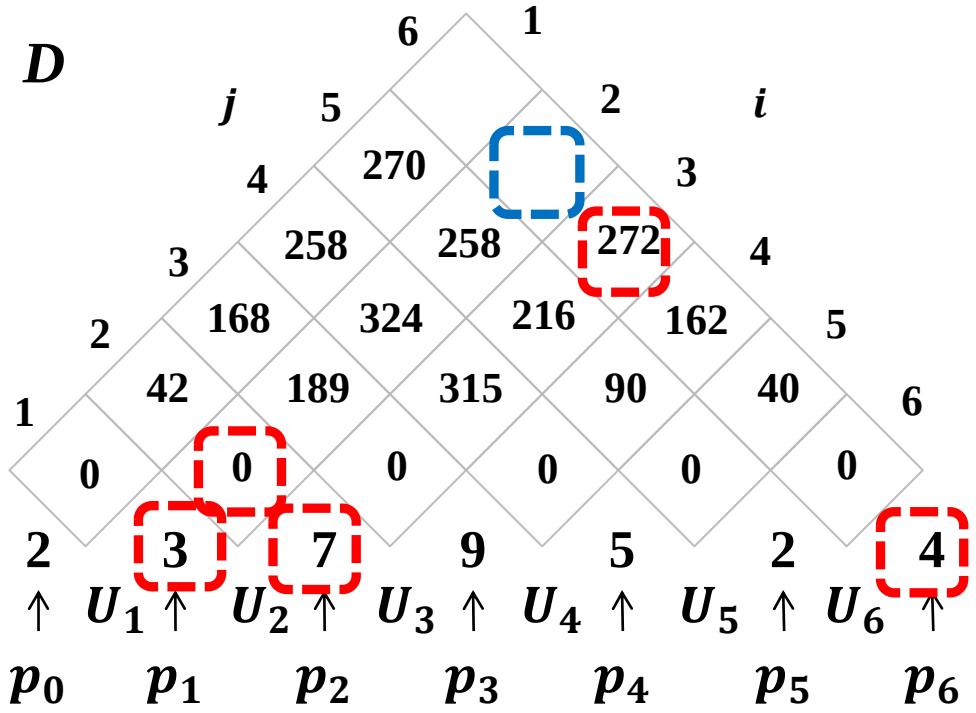
- $$D[1,5]=\min\begin{cases}D[1,1]+D[2,5]+p_0p_1p_5=270\\D[1,2]+D[3,5]+p_0p_2p_5=286\\D[1,3]+D[4,5]+p_0p_3p_5=294\\D[1,4]+D[5,5]+p_0p_4p_5=278\end{cases}$$

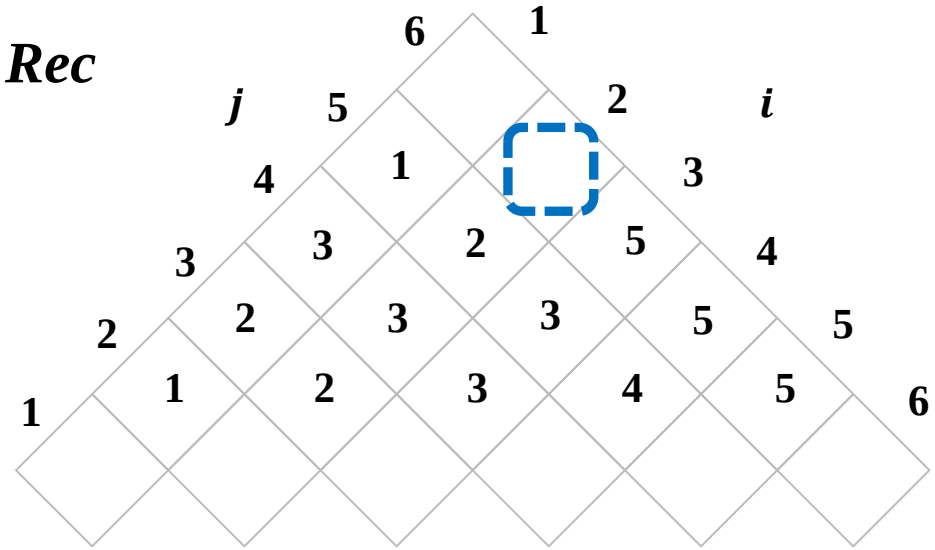




算法实例

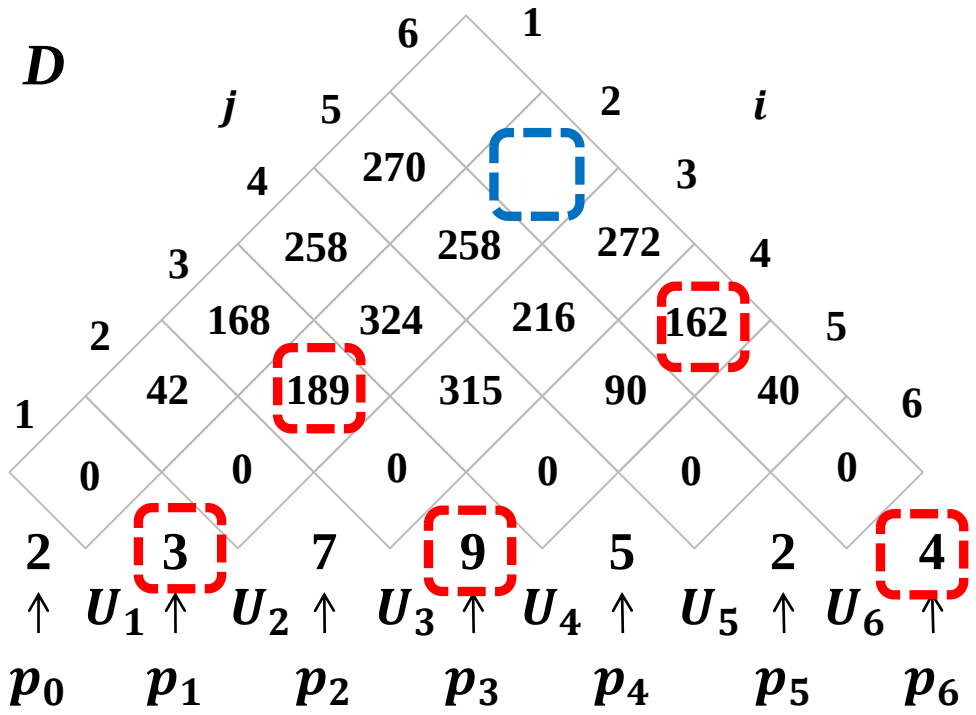
- $D[2,6] = \min \begin{cases} D[2,2] + D[3,6] + p_1 p_2 p_6 = 356 \\ D[2,3] + D[4,6] + p_1 p_3 p_6 = \\ D[2,4] + D[5,6] + p_1 p_4 p_6 = \\ D[2,5] + D[6,6] + p_1 p_5 p_6 = \end{cases}$

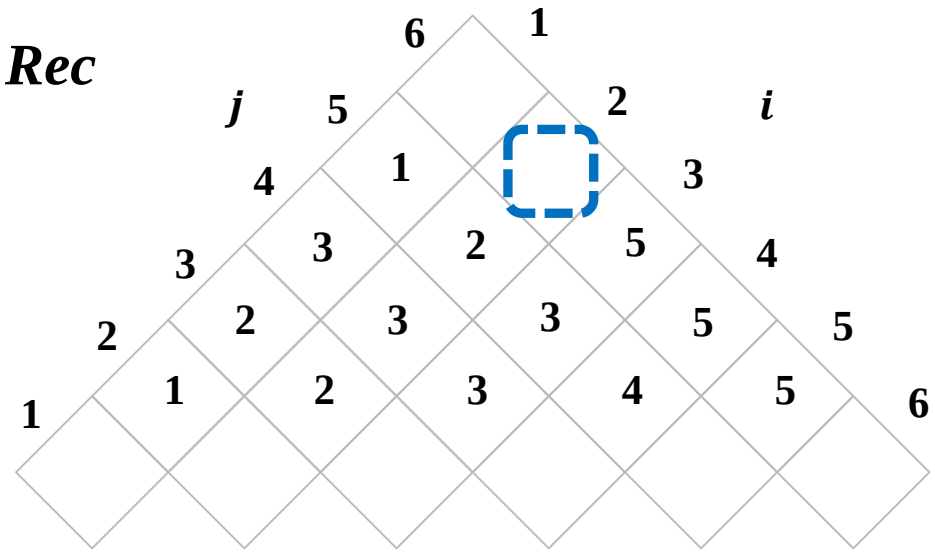




算法实例

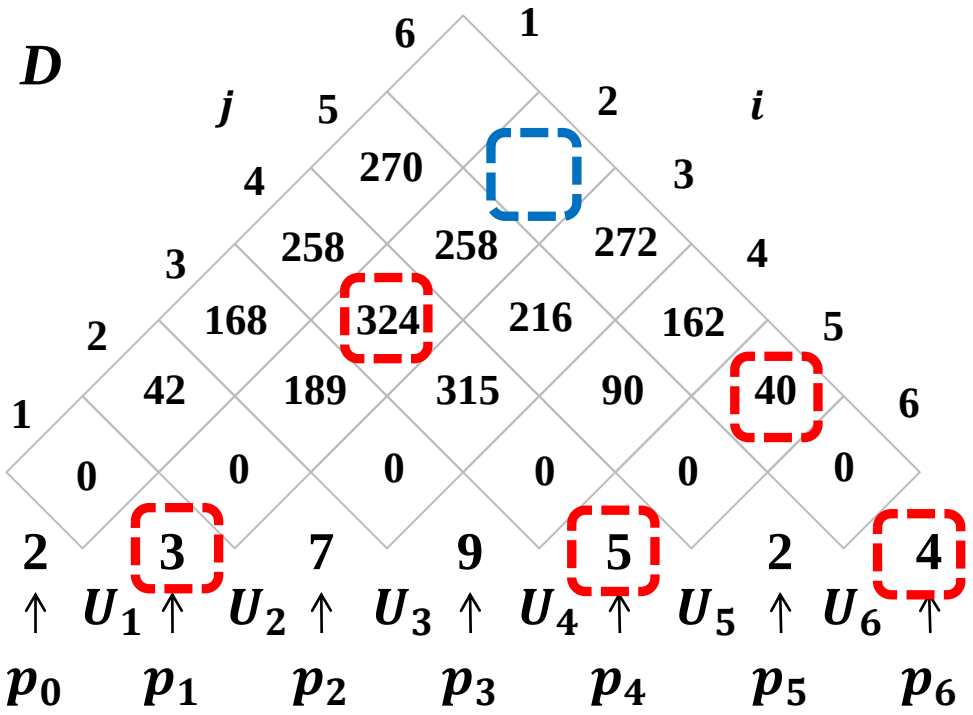
- $D[2,6] = \min \begin{cases} D[2,2] + D[3,6] + p_1 p_2 p_6 = 356 \\ D[2,3] + D[4,6] + p_1 p_3 p_6 = 459 \\ D[2,4] + D[5,6] + p_1 p_4 p_6 = \\ D[2,5] + D[6,6] + p_1 p_5 p_6 = \end{cases}$

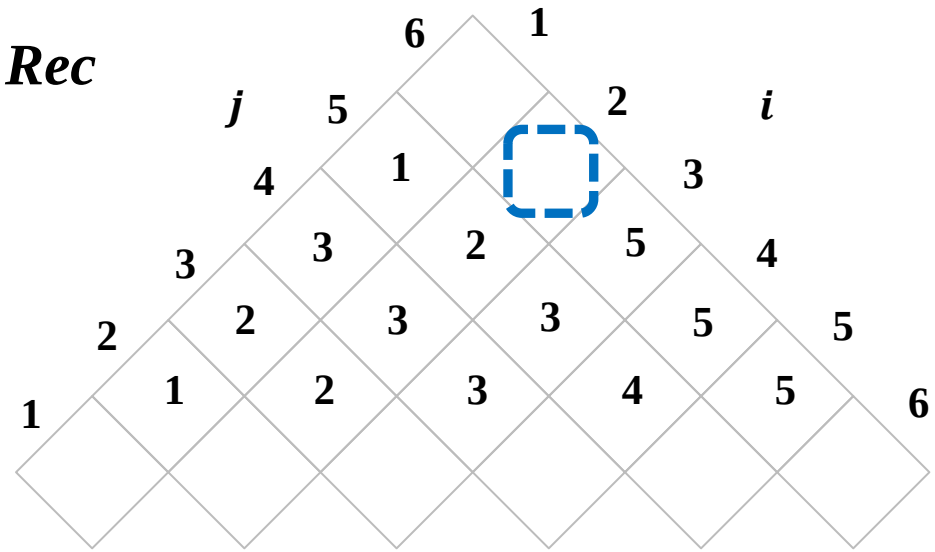




算法实例

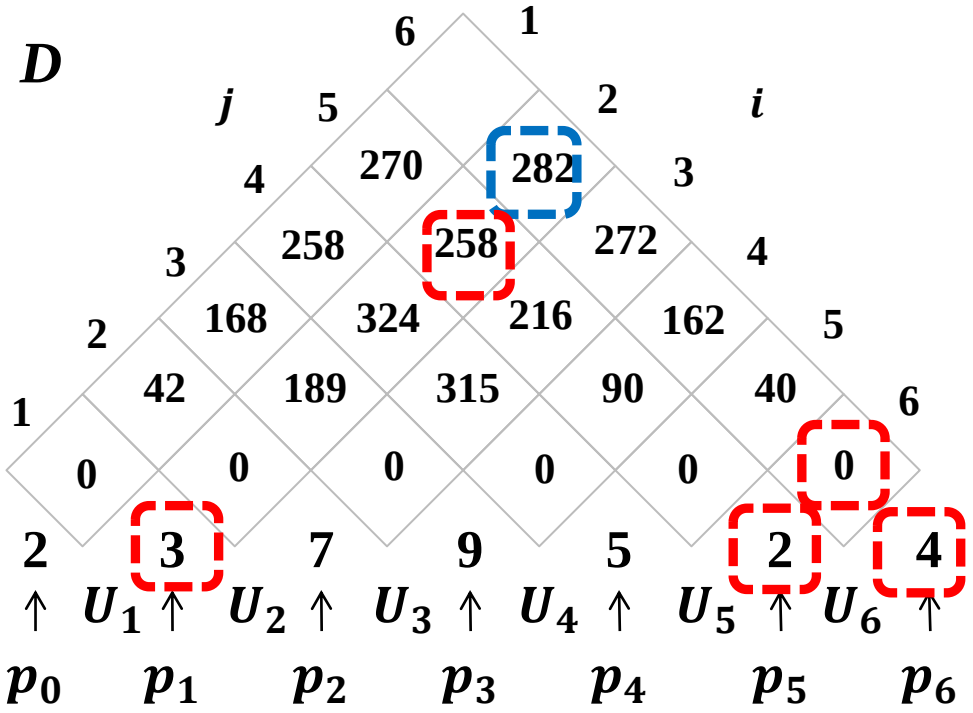
- $D[2,6] = \min \begin{cases} D[2,2] + D[3,6] + p_1 p_2 p_6 = 356 \\ D[2,3] + D[4,6] + p_1 p_3 p_6 = 459 \\ D[2,4] + D[5,6] + p_1 p_4 p_6 = 424 \\ D[2,5] + D[6,6] + p_1 p_5 p_6 = \end{cases}$

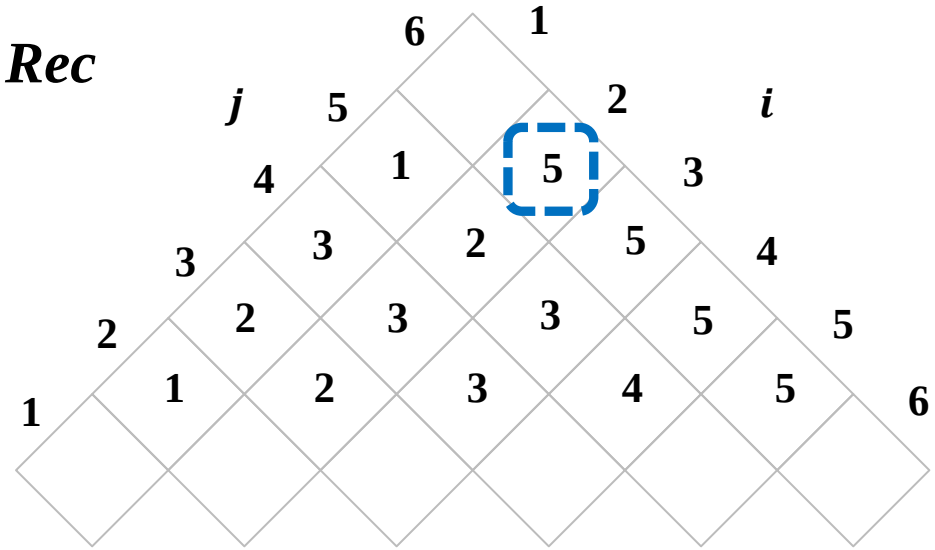




算法实例

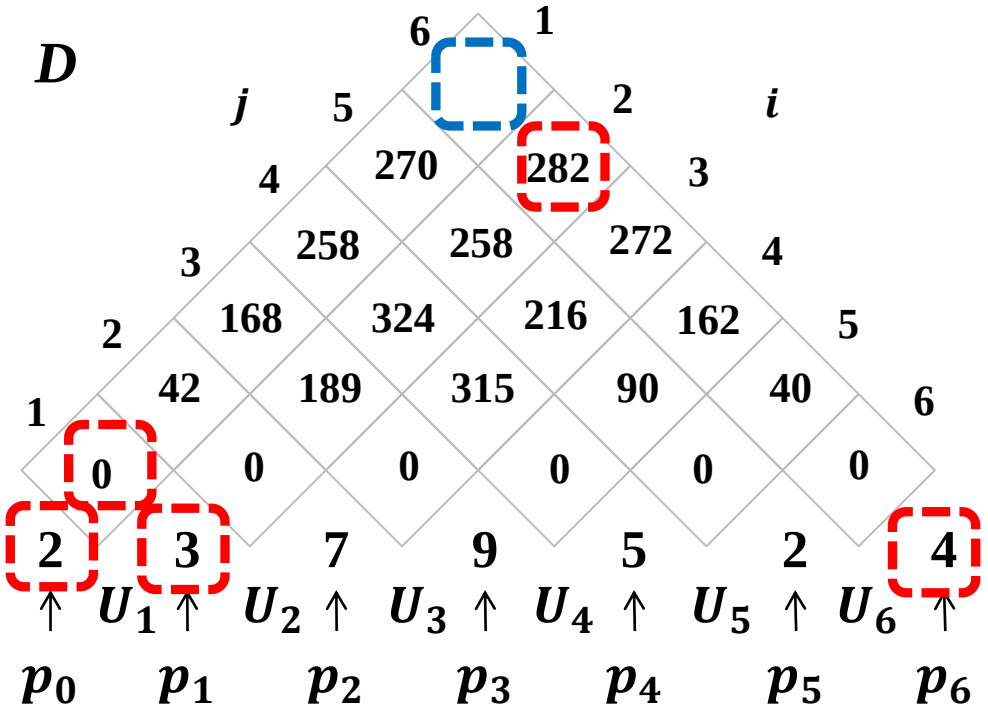
- $$D[2,6]=\min\begin{cases}D[2,2]+D[3,6]+p_1p_2p_6=356\\ D[2,3]+D[4,6]+p_1p_3p_6=459\\ D[2,4]+D[5,6]+p_1p_4p_6=424\\ D[2,5]+D[6,6]+p_1p_5p_6=282\end{cases}$$

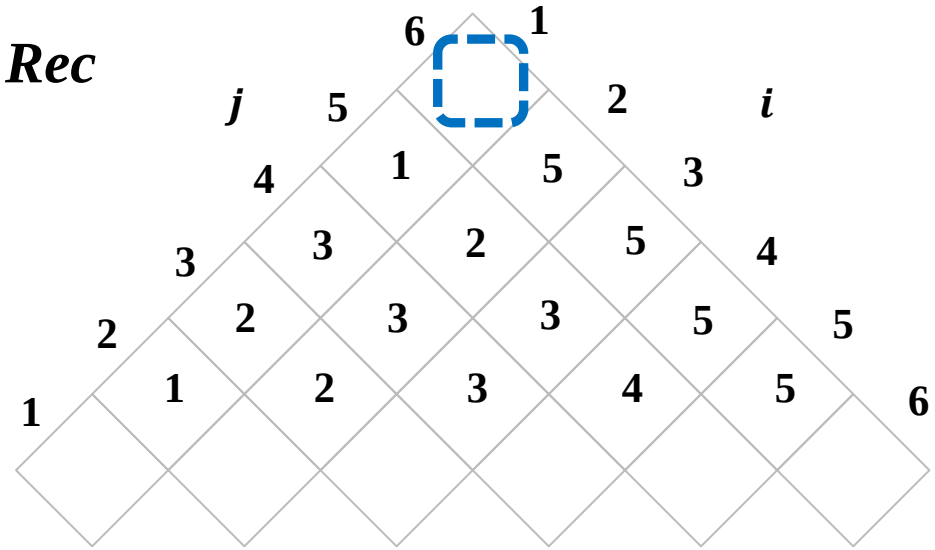




算法实例

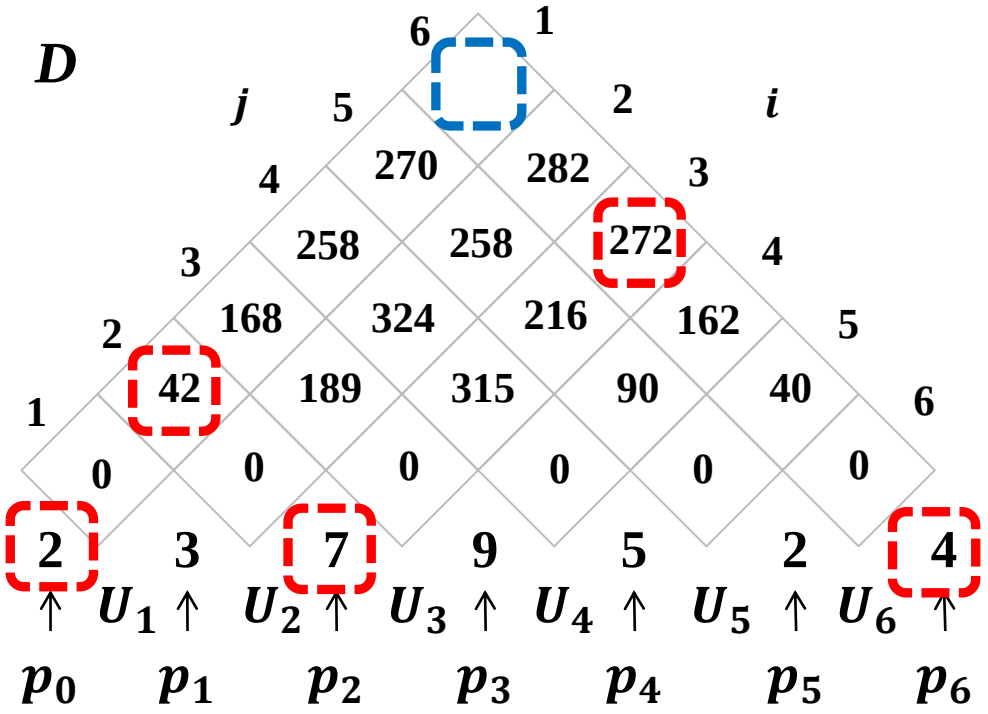
- $$D[1,6]=\min\begin{cases}D[1,1]+D[2,6]+p_0p_1p_6=306\\ D[1,2]+D[3,6]+p_0p_2p_6=\\ D[1,3]+D[4,6]+p_0p_3p_6=\\ D[1,4]+D[5,6]+p_0p_4p_6=\\ D[1,5]+D[6,6]+p_0p_5p_6=\end{cases}$$

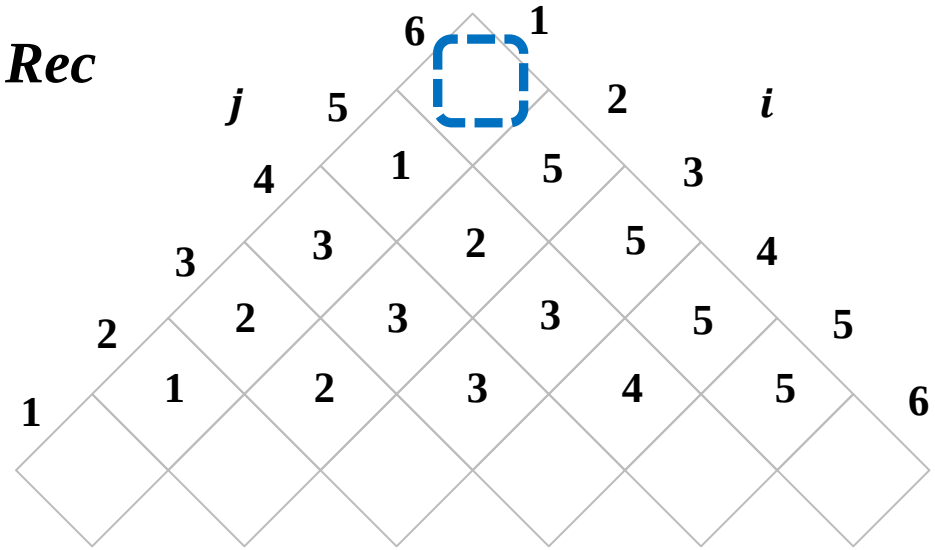




算法实例

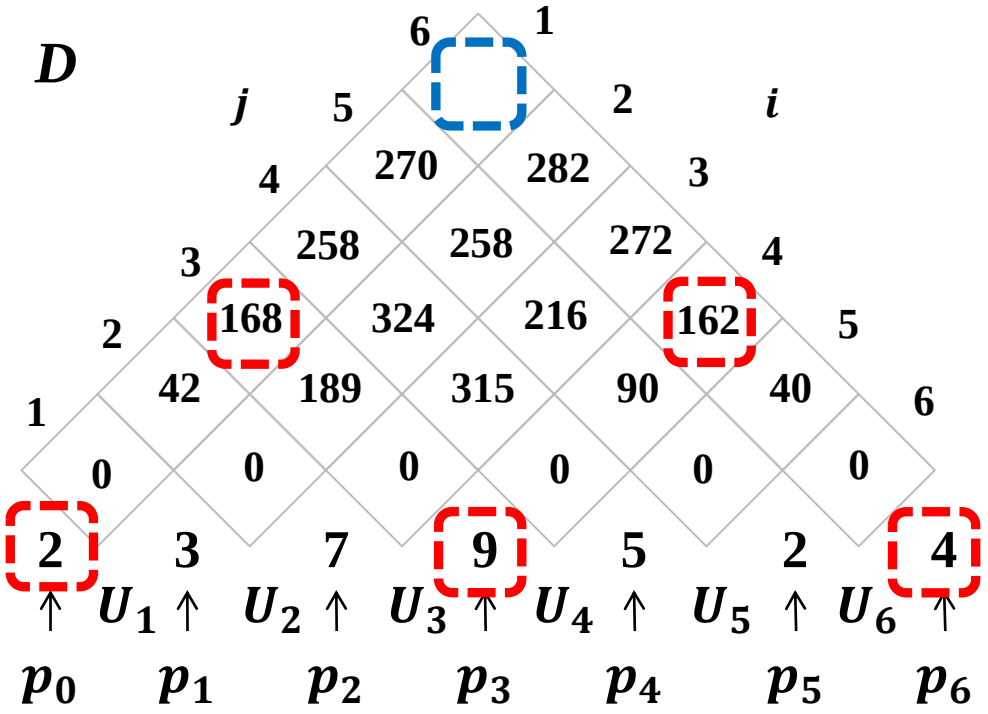
- $$D[1,6]=\min\begin{cases}D[1,1]+D[2,6]+p_0p_1p_6=306\\ D[1,2]+D[3,6]+p_0p_2p_6=370\\ D[1,3]+D[4,6]+p_0p_3p_6=\\ D[1,4]+D[5,6]+p_0p_4p_6=\\ D[1,5]+D[6,6]+p_0p_5p_6=\end{cases}$$

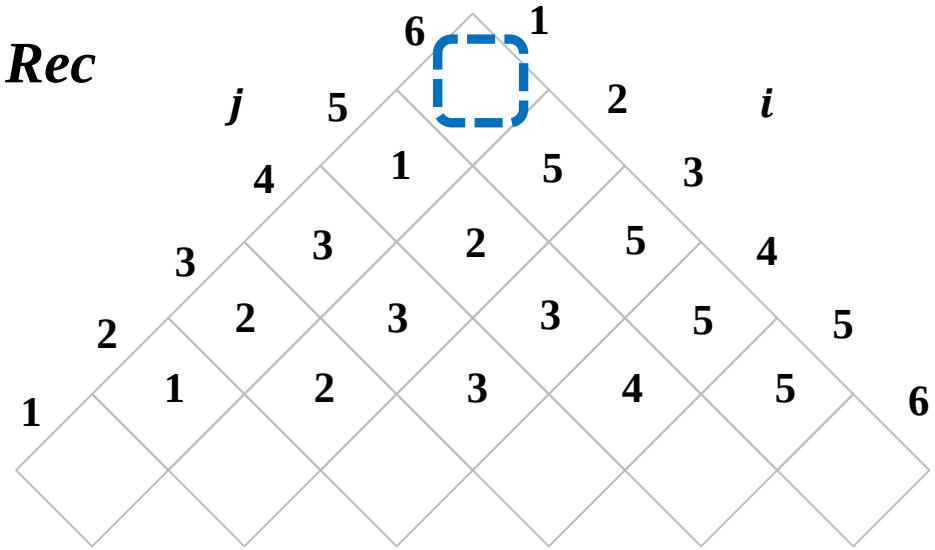




算法实例

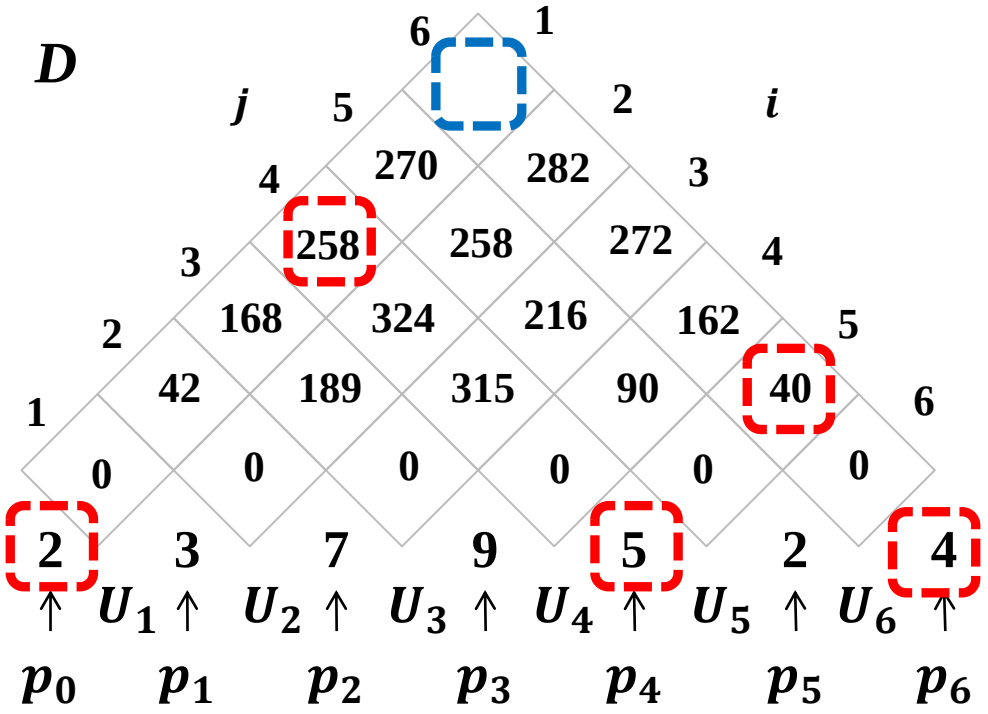
- $$D[1,6]=\min\begin{cases}D[1,1]+D[2,6]+p_0p_1p_6=306\\D[1,2]+D[3,6]+p_0p_2p_6=370\\D[1,3]+D[4,6]+p_0p_3p_6=402\\D[1,4]+D[5,6]+p_0p_4p_6=\\D[1,5]+D[6,6]+p_0p_5p_6=\end{cases}$$

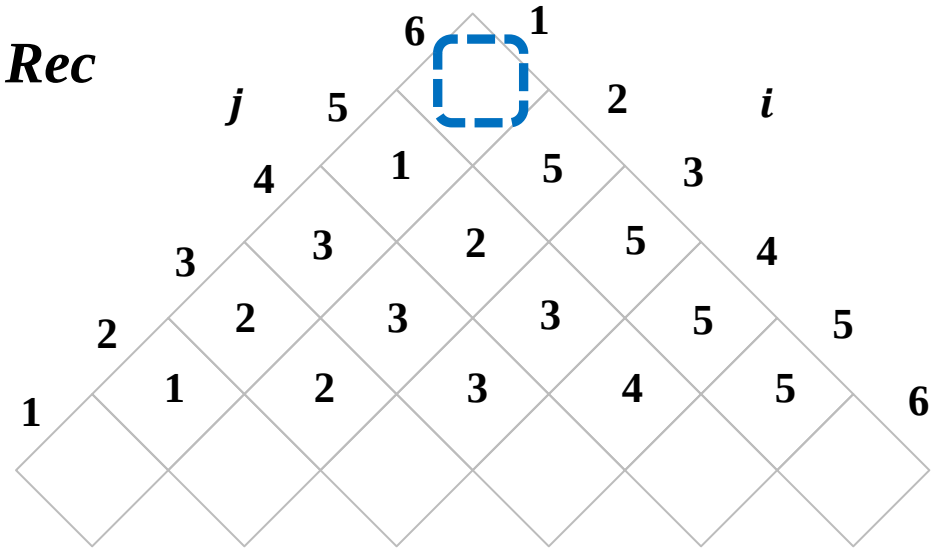




算法实例

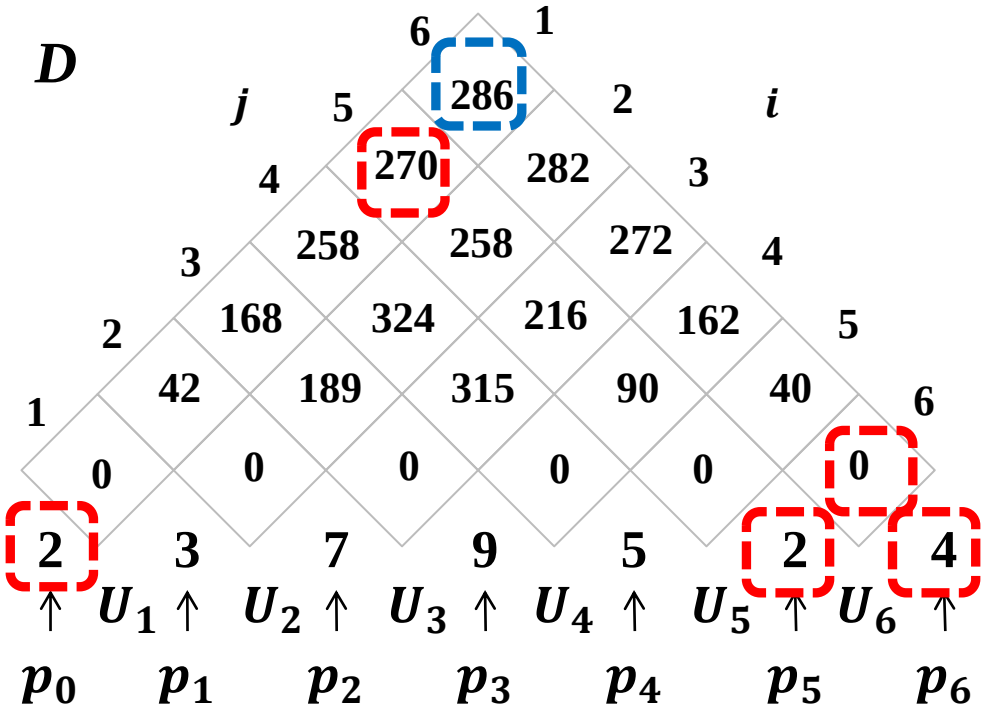
- $$D[1,6]=\min\begin{cases}D[1,1]+D[2,6]+p_0p_1p_6=306\\D[1,2]+D[3,6]+p_0p_2p_6=370\\D[1,3]+D[4,6]+p_0p_3p_6=402\\D[1,4]+D[5,6]+p_0p_4p_6=338\\D[1,5]+D[6,6]+p_0p_5p_6=\end{cases}$$

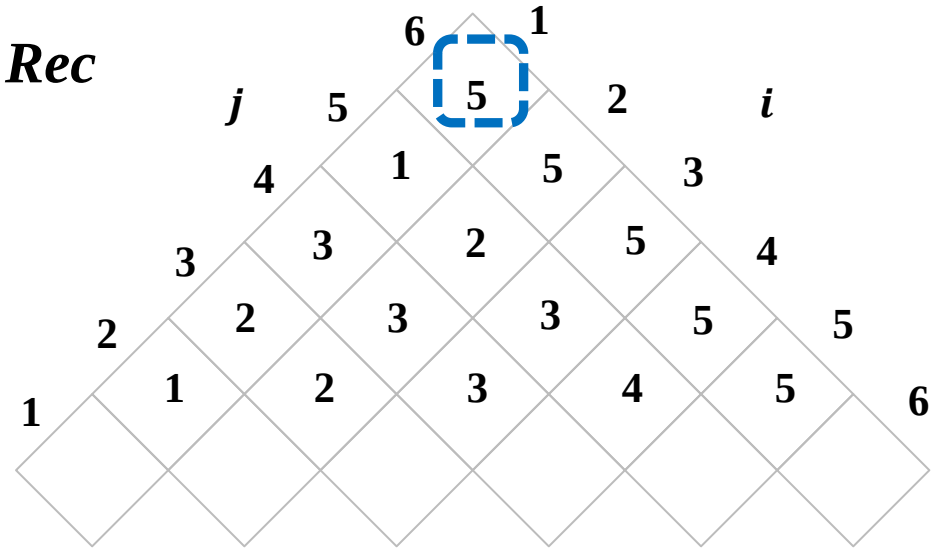




算法实例

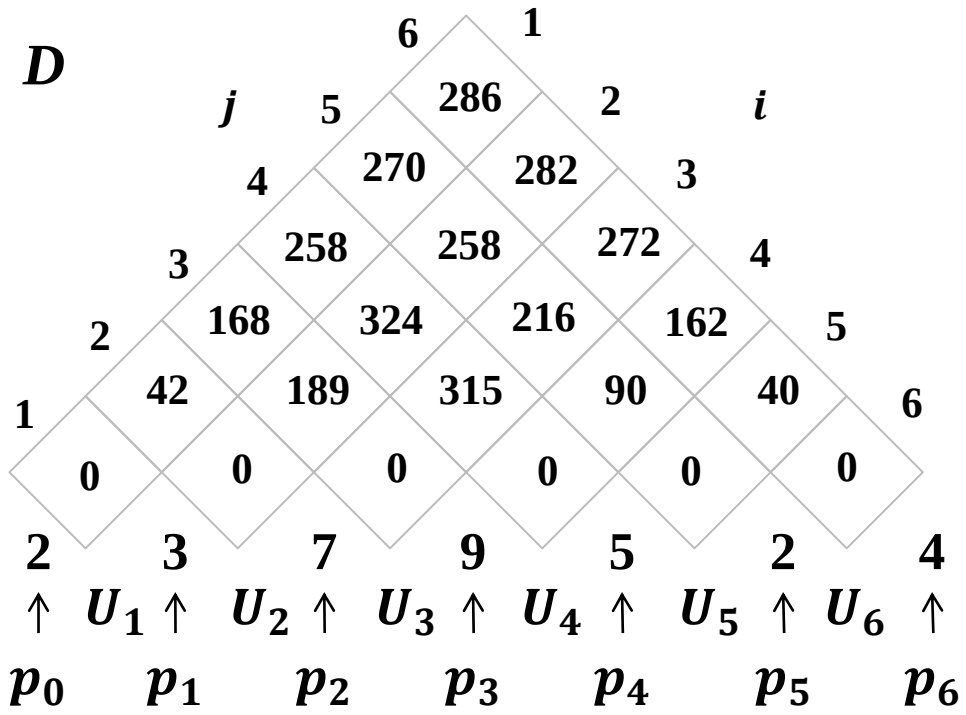
- $$D[1,6]=\min\begin{cases}D[1,1]+D[2,6]+p_0p_1p_6=306\\ D[1,2]+D[3,6]+p_0p_2p_6=370\\ D[1,3]+D[4,6]+p_0p_3p_6=402\\ D[1,4]+D[5,6]+p_0p_4p_6=338\\ D[1,5]+D[6,6]+p_0p_5p_6=286\end{cases}$$

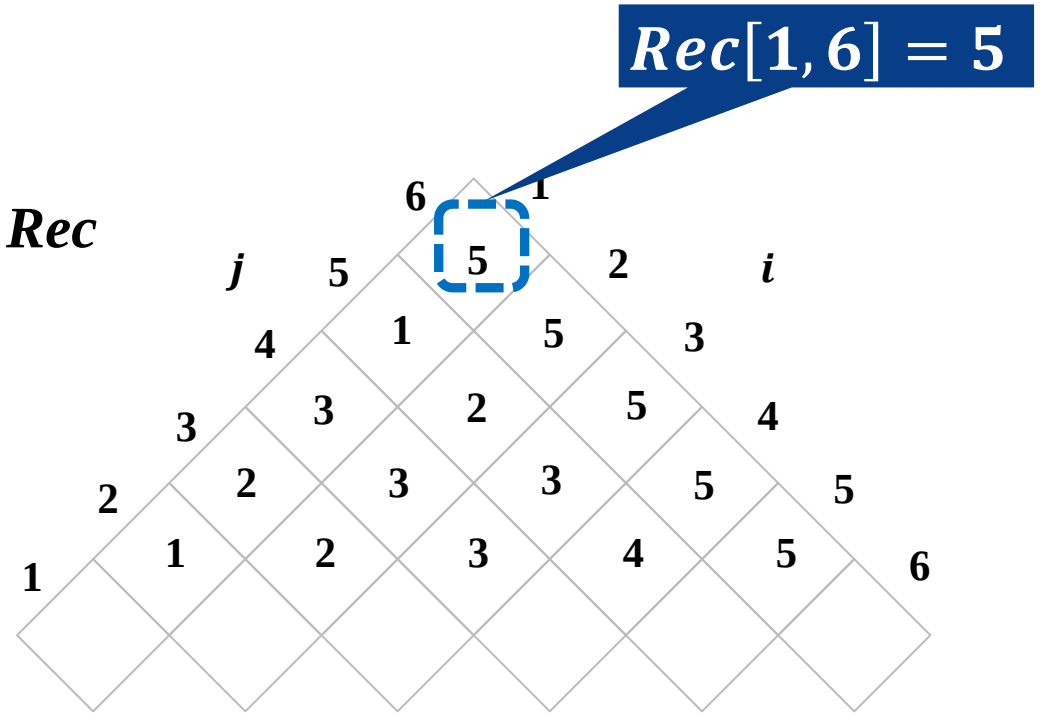




算法实例

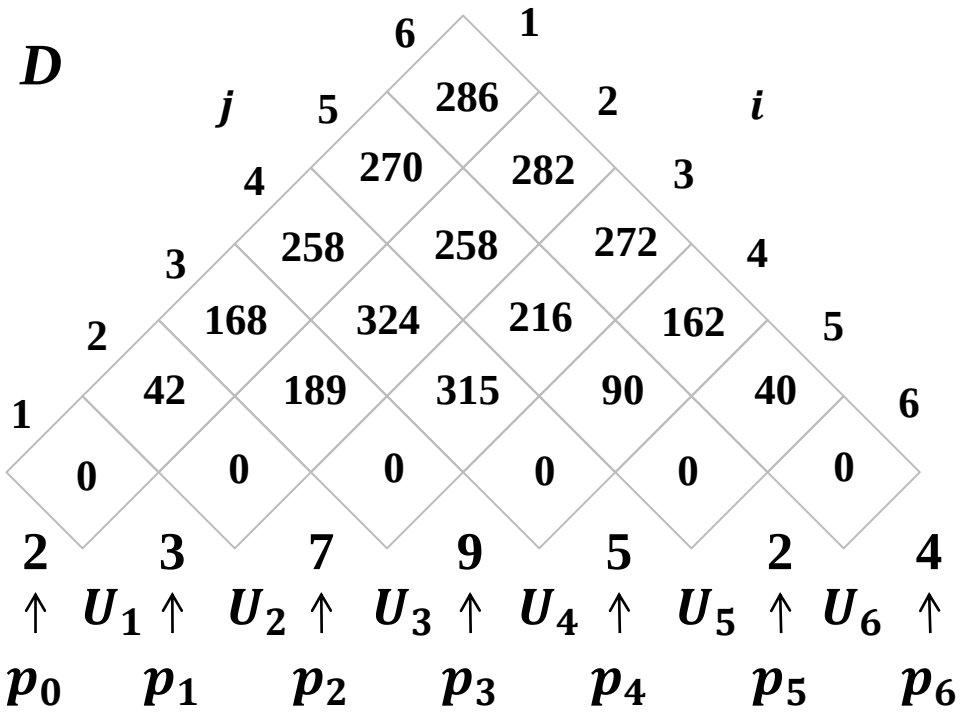
- $U_1U_2U_3U_4U_5U_6$
- $\rightarrow (U_1U_2U_3U_4U_5)(U_6)$

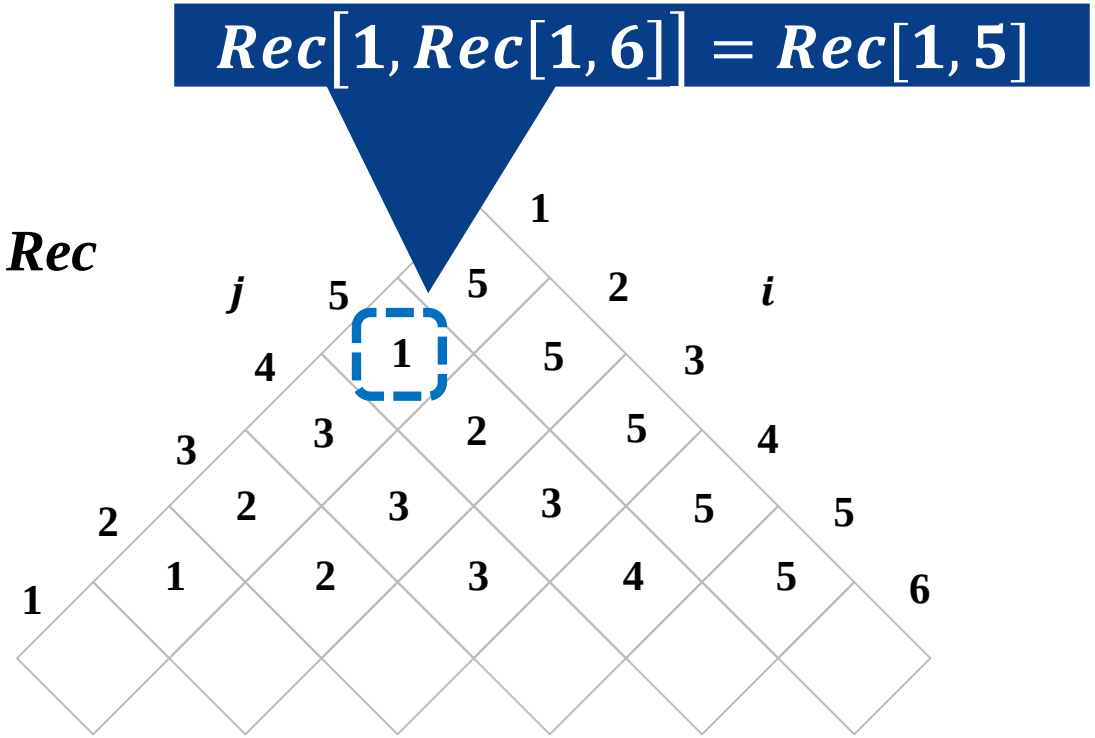




算法实例

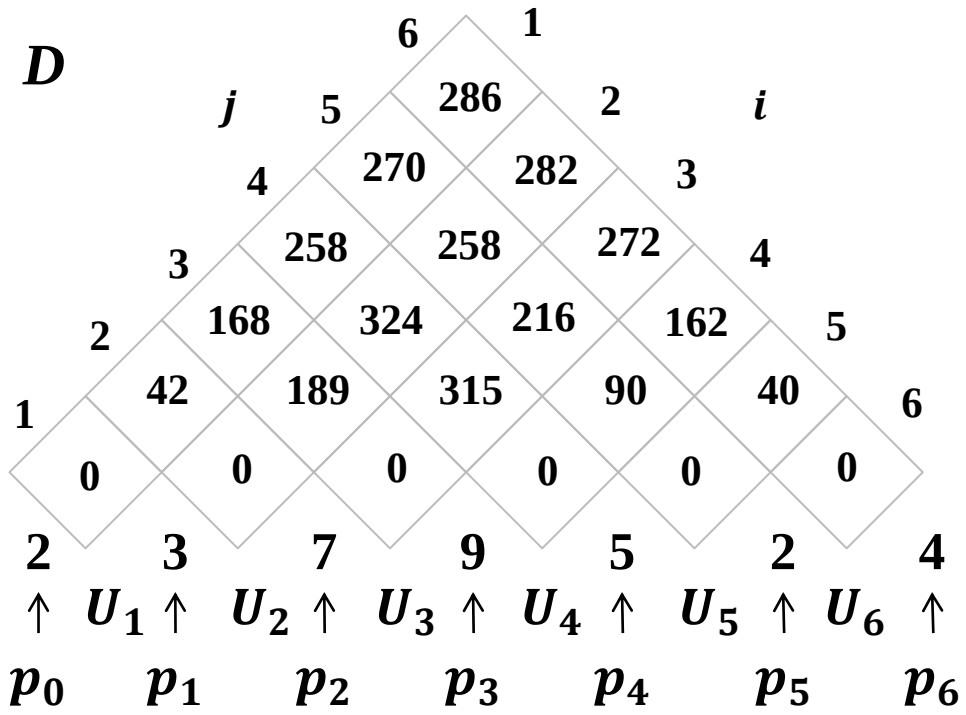
- $U_1U_2U_3U_4U_5U_6$
- $\rightarrow (U_1U_2U_3U_4U_5)(U_6)$

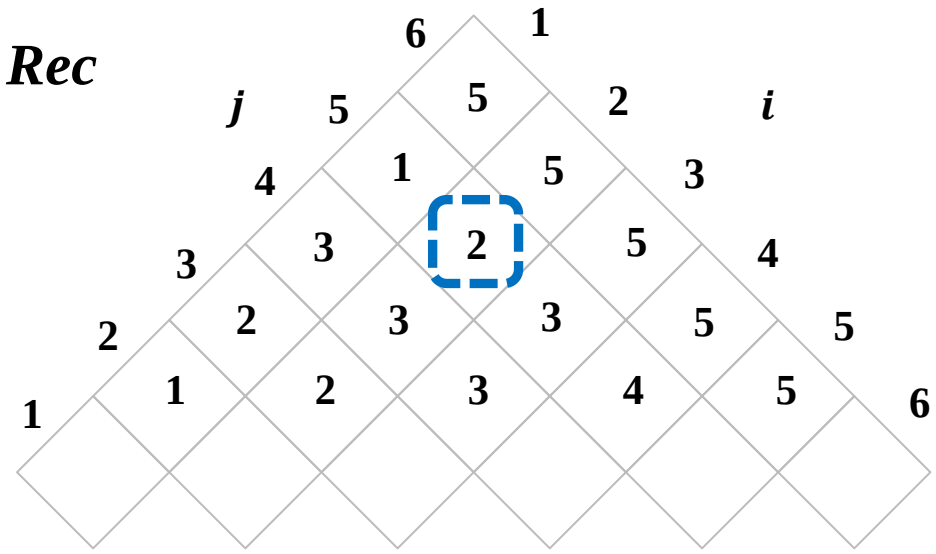




算法实例

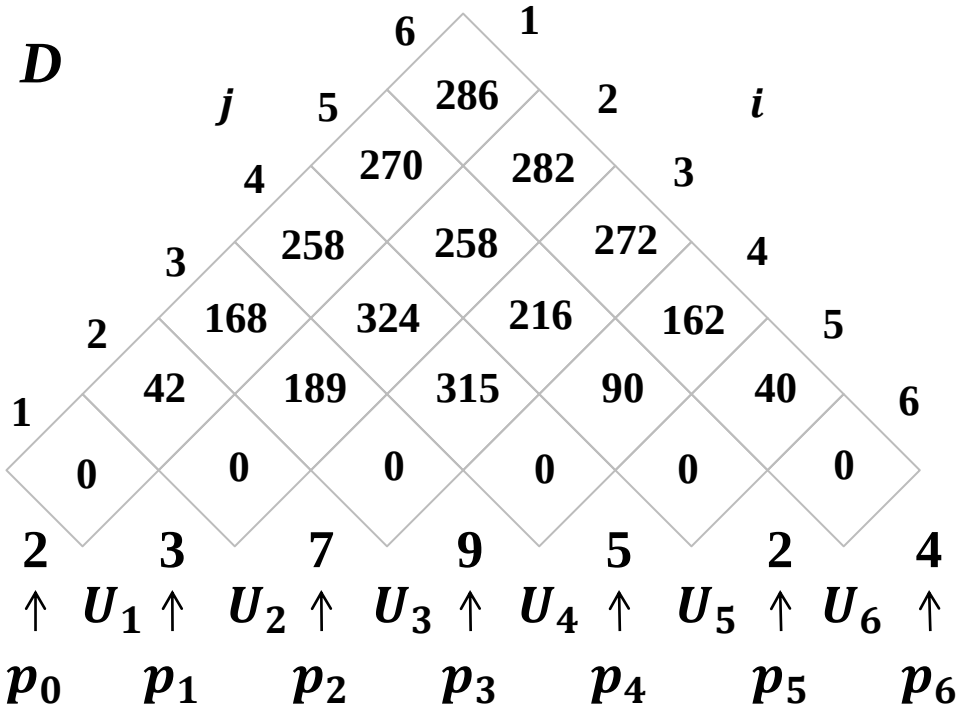
- $U_1U_2U_3U_4U_5U_6$
- $\rightarrow (U_1U_2U_3U_4U_5)(U_6)$
- $\rightarrow (U_1(U_2U_3U_4U_5))(U_6)$

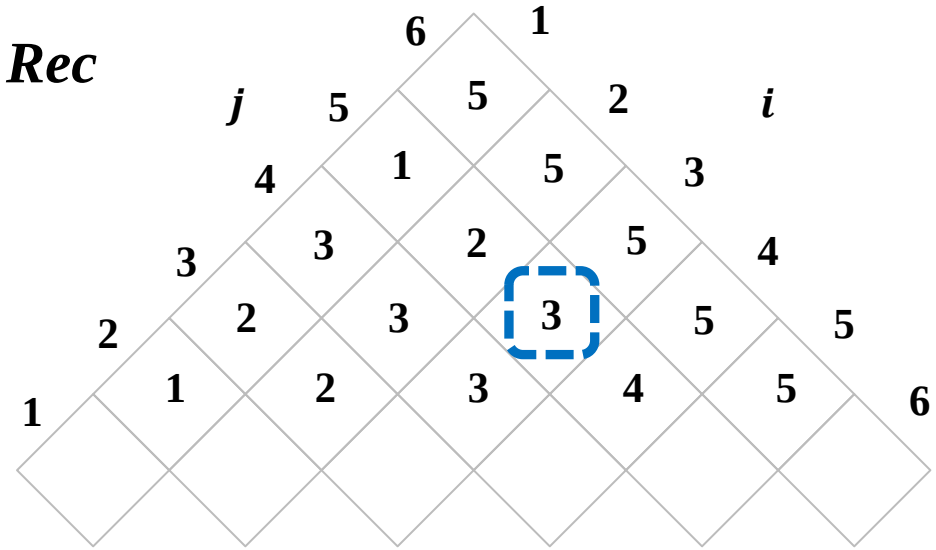




算法实例

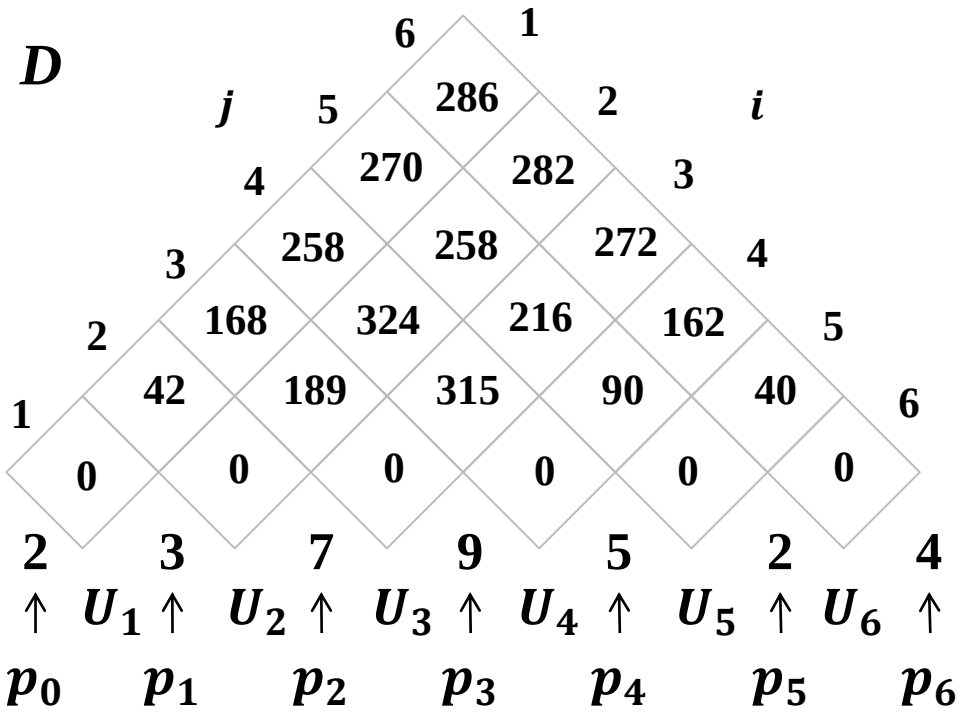
- $U_1U_2U_3U_4U_5U_6$
- $\rightarrow (U_1U_2U_3U_4U_5)(U_6)$
- $\rightarrow (U_1(U_2U_3U_4U_5))(U_6)$
- $\rightarrow (U_1(U_2(U_3U_4U_5)))(U_6)$

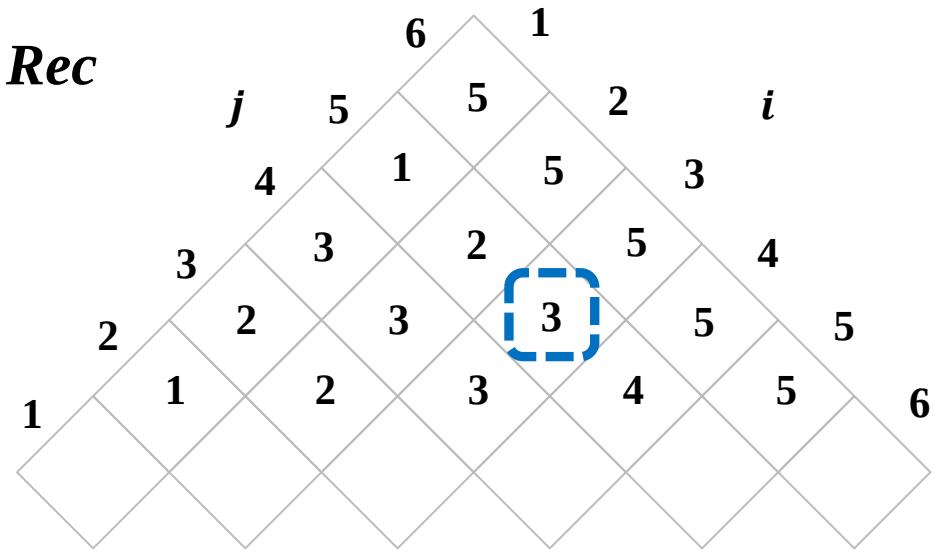




算法实例

- $U_1U_2U_3U_4U_5U_6$
- $\rightarrow (U_1U_2U_3U_4U_5)(U_6)$
- $\rightarrow (U_1(U_2U_3U_4U_5))(U_6)$
- $\rightarrow (U_1(U_2(U_3U_4U_5)))(U_6) \rightarrow (U_1(U_2((U_3)(U_4U_5))))(U_6)$





动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

输入: 矩阵维度数组p, 矩阵的个数n
输出: 最小标量乘法次数, 分割方式追踪数组Rec
新建二维数组$D[1...n, 1...n], Rec[1...n, 1...n]$
//初始化

```
D ← ∞
for i ← 1 to n do
    D[i, i] ← 0
end
```

初始化

动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do                          区间长度从小到大
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```

动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```

依次计算子问题

动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```

枚举所有分割位置

动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```

计算最少乘法次数

动态规划：伪代码

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```

记录最优决策

最优方案追踪：伪代码

- **Print-Matrix-Chain(U, Rec, i, j)**

初始调用：Print-Matrix-Chain($U, Rec, 1, n$)

输入: 矩阵链$U_{1..n}$,追踪数组$Rec[1..n, 1..n]$,位置索引i, j
输出: 矩阵链的加括号方式

```
if i = j then
    print U_i
    return
end
print "("
Print-Matrix-Chain(U, Rec, i, Rec[i, j])
print ")("
Print-Matrix-Chain(U, Rec, Rec[i, j] + 1, j)
print ")"
return
```

链长为1直接输出

最优方案追踪：伪代码

- **Print-Matrix-Chain(U, Rec, i, j)**

初始调用：Print-Matrix-Chain($U, Rec, 1, n$)

输入: 矩阵链$U_{1..n}$,追踪数组$Rec[1..n, 1..n]$,位置索引i, j
输出: 矩阵链的加括号方式

```
if i = j then
    print U_i
    return
end
print "("
Print-Matrix-Chain(U, Rec, i, Rec[i, j])
print ")("
Print-Matrix-Chain(U, Rec, Rec[i, j] + 1, j)
print ")"
return
```

递归输出左侧加括号方式

最优方案追踪：伪代码

- **Print-Matrix-Chain(U, Rec, i, j)**

初始调用：Print-Matrix-Chain($U, Rec, 1, n$)

输入: 矩阵链$U_{1..n}$,追踪数组$Rec[1..n, 1..n]$,位置索引i, j
输出: 矩阵链的加括号方式

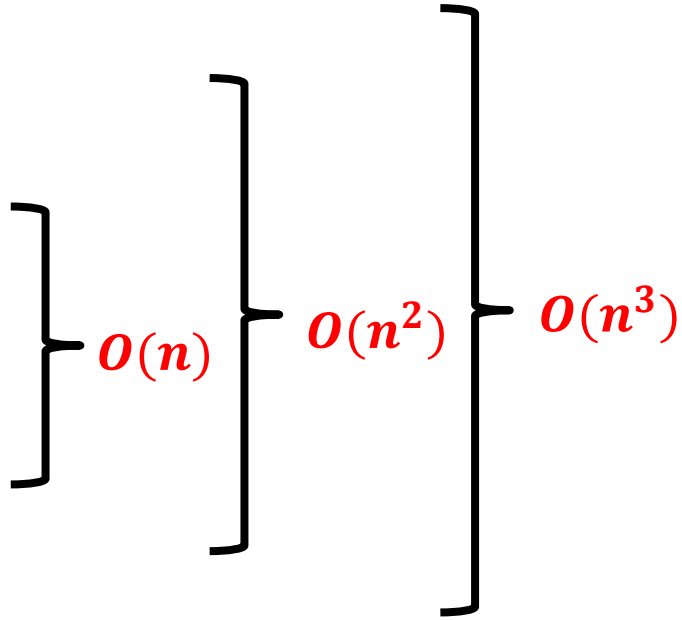
```
if i = j then
    print U_i
    return
end
print "("
Print-Matrix-Chain(U, Rec, i, Rec[i, j])
print ")("
Print-Matrix-Chain(U, Rec, Rec[i, j] + 1, j)
print ")"
return
```

递归输出右侧加括号方式

时间复杂度分析

- **Matrix-Chain-Multiply(p, n)**

```
//动态规划
for l ← 2 to n do
    for i ← 1 to n − l + 1 do
        j ← i + l − 1
        D[i, j] ← ∞
        for k ← i to j − 1 do
            q ← D[i, k] + D[k + 1, j] + p[i − 1] ∗ p[k] ∗ p[j]
            if q < D[i, j] then
                D[i, j] ← q
                Rec[i, j] ← k
            end
        end
    end
end
return D[1, n], Rec
```



时间复杂度：$O(n^3)$

资源分配问题

📖**问题描述：**资源分配问题是将数量一定的一种或若干种资源（原材料、资金、设备或劳动力等），合理地分配给若干使用者，使总收益最大。

例如，某公司有3个商店*A*、*B*、*C*，拟将新招聘的5名员工分配给这3个商店，各商店得到新员工后，每年的赢利情况如下表所示，求分配给各商店各多少员工才能使公司的赢利最大？

商店＼员工数	0人	1人	2人	3人	4人	5人
A	0	3	7	9	12	13
B	0	5	10	11	11	11
C	0	4	6	11	12	12

解

- 设置二维动态规划数组dp，其中dp[i][s]表示前i个商店（商店编号为0～i-1）共分配s（0≤s≤n）名新员工的最优增收。
- 另外设置二维数组pnum，其中pnum[i][s]表示求出dp[i][s]时对应商店i-1的分配人数。
- 显然如果商店个数为0，无论增加多少人员，总增收一定为0，即dp[0][j]=0（0≤j≤n）。

- 现在考虑求dp[i][s]，商店0～商店i-1共分配s名新员工。
- 商店i-1理论上讲可以分配j（0≤j≤s）名新员工，商店i-1分配j名新员工时的增收为v[i-1][j]+dp[i-1][s-j]，则最大增收为：

$$\max_{0\leqslant s\leqslant n,\ 0\leqslant j\leqslant s}\{\ v[i-1][j]+dp[i-1][s-j]\ \}$$

dp[0][j]=0　　边界条件（0≤j≤n）

dp[i][s]=$\max_{0\leqslant s\leqslant n,\ 0\leqslant j\leqslant s}$\{v[i-1][j]+dp[i-1][s-j]\}　　其他

pnum[i][s]=dp[i][s]取最大值的j

当求出dp数组后，dp[m][n]就是最终的最优增收。

```
vector<vector<int>> dp;                    //二维动态规划数组
vector<vector<int>> pnum;                  //分配人数

int plan(int m,int n,vector<vector<int>>&v) {  //求dp和pnum
   dp=vector<vector<int>>(m+1,vector<int>(n+1,0));
   pnum=vector<vector<int>>(m+1,vector<int>(n+1,0));
   for (int j=0;j<=n;j++)                  //置边界条件
      dp[0][j]=0;
```

```
   for (int i=1;i<=m;i++) {
      for (int s=0;s<=n;s++) {
         int maxf=0,maxj=0;
         for (int j=0;j<=s;j++)
          {
            if ((v[i-1][j]+dp[i-1][s-j])>maxf)
            {
               maxf=v[i-1][j]+dp[i-1][s-j];
               maxj=j;
            }
          }
         dp[i][s]=maxf;
         pnum[i][s]=maxj;
      }
   }
   return dp[m][n];
}
```

可以从pnum[*m*][*n*]开始推导出各个商店分配的人数，用vector<int>容器*x*存放各个商店分配的人数。

```
vector<int> getx(int m,int n) {          //求一个最优分配方案
   vector<int> x;                        //存放一个最优分配方案
   int s=pnum[m][n];
   x.push_back(s);
   int r=n-s;                            //r为余下的人数
   for (int k=m;k>1;k--) {
      s=pnum[k-1][r];                    //求下一个阶段分配的人数
      x.push_back(s);
      r=r-s;                             //余下人数递减
   }
   reverse(x.begin(),x.end());           //逆置x
   return x;
}
```