动态规划篇：0-1背包问题

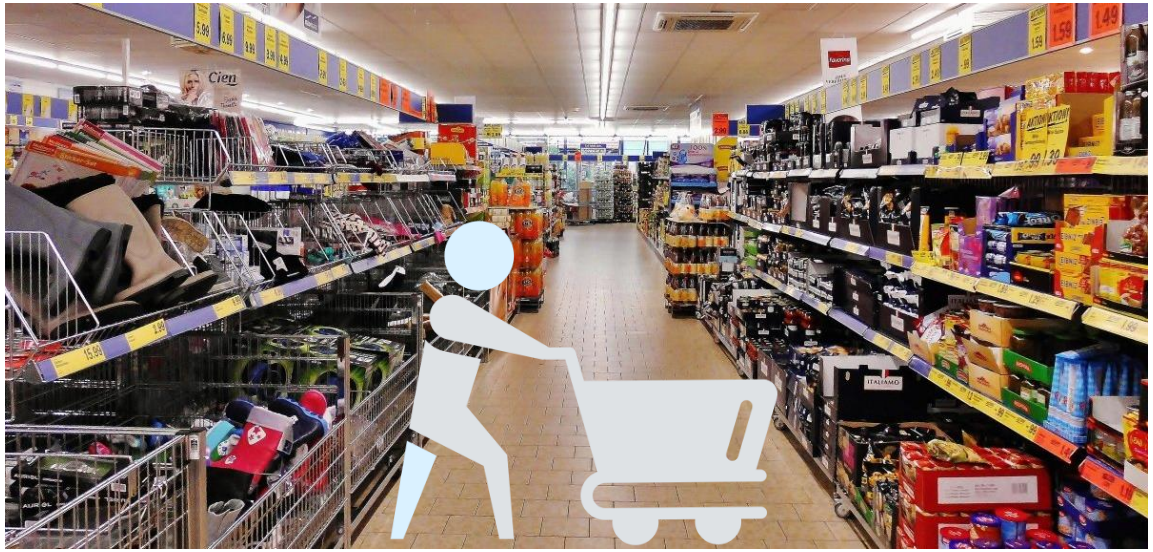
问题背景

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

问题背景

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

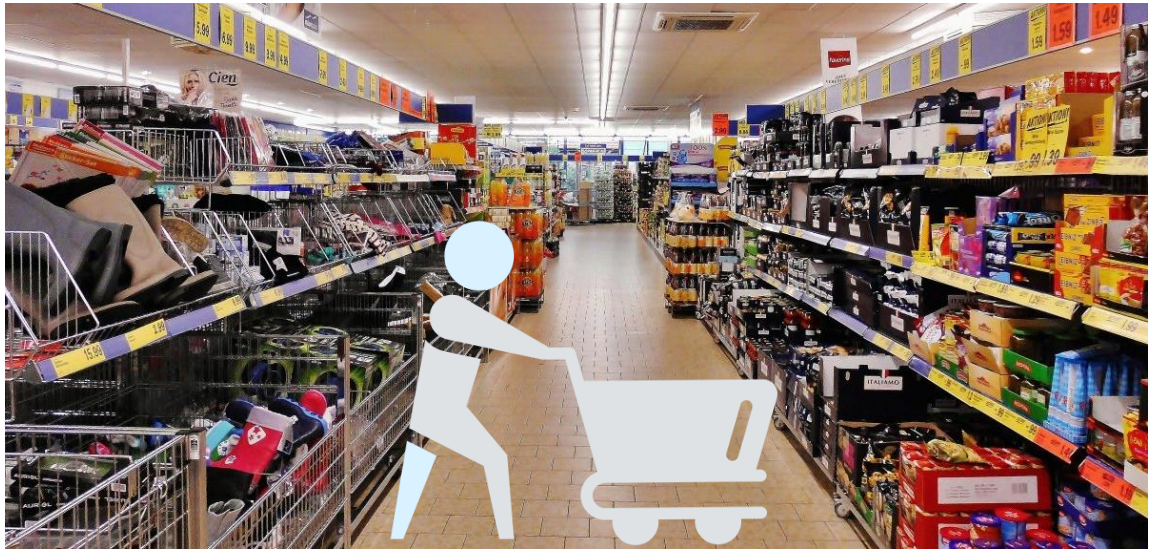
商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4



问题背景

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4



问题：如何带走总价最多的商品?

问题定义

- 形式化定义

0-1背包问题

0-1 Knapsack Problem

输入

- n个商品组成集合O，每个商品有两个属性v_i和p_i，分别表示体积和价格

问题定义

- 形式化定义

0-1背包问题

0-1 Knapsack Problem

输入

- n个商品组成集合O，每个商品有两个属性v_i和p_i，分别表示体积和价格
- 背包容量为C

问题定义

- 形式化定义

0-1背包问题

0-1 Knapsack Problem

输入

- n个商品组成集合O，每个商品有两个属性v_i和p_i，分别表示体积和价格
- 背包容量为C

输出

- 求解一个商品子集$S \subseteq O$，令

$$\max \sum_{i \in S} p_i$$

$$s.t. \sum_{i \in S} v_i \leq C$$

问题定义

- 形式化定义

0-1背包问题

0-1 Knapsack Problem

输入

- n个商品组成集合O，每个商品有两个属性v_i和p_i，分别表示体积和价格
- 背包容量为C

输出

- 求解一个商品子集$S \subseteq O$，令

优化目标

$$\max \sum_{i \in S} p_i$$

$$s.t. \sum_{i \in S} v_i \leq C$$

问题定义

- 形式化定义

0-1背包问题

0-1 Knapsack Problem

输入

- n个商品组成集合O，每个商品有两个属性v_i和p_i，分别表示体积和价格
- 背包容量为C

输出

- 求解一个商品子集$S \subseteq O$，令

优化目标

$$\max \sum_{i \in S} p_i$$

约束条件

$$s.t. \sum_{i \in S} v_i \leq C$$

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案
方案2		26	13	

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
 啤酒	24	10
 汽水	2	3
 饼干	9	4
 面包	10	5
 牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案
方案2		26	13	可行方案

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案
方案2		26	13	可行方案
方案3		28	13	

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案
方案2		26	13	可行方案
方案3		28	13	最优方案

直观策略

- 超市赢家
  - 超市允许顾客使用一个体积大小为13的背包，选择一件或多件商品带走

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

	商品列表	总价格	总体积	说明
方案1		34	15	错误方案
方案2		26	13	可行方案
方案3		28	13	最优方案

问题：如何求解该问题?

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明

剩余体积：13
商品价值：0

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		13	26	

剩余体积：0
商品价值：26

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略1：按商品价格由高到低排序，优先挑选价格高的商品

商品	价格	体积
啤酒	24	10
面包	10	5
饼干	9	4
牛奶	9	4
汽水	2	3

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2				

剩余体积：13
商品价值：0

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		3	2	

剩余体积：10
商品价值：2

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		7	11	

剩余体积：6
商品价值：11

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	

剩余体积：2
商品价值：20

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	

剩余体积：2
商品价值：20

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	

剩余体积：2
商品价值：20

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略2：按商品体积由小到大排序，优先挑选体积小的商品

商品	价格	体积
汽水	2	3
饼干	9	4
牛奶	9	4
面包	10	5
啤酒	24	10

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3				

剩余体积：13
商品价值：0

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		10	24	

剩余体积：3
商品价值：24

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		13	26	

剩余体积：0
商品价值：26

直观策略

- 核心思想：将商品排序，依次挑选
  - 策略3：按商品价值与体积的比由高到低排序，优先挑选比值高的商品

商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	商品列表	总体积	总价格	说明
策略1		13	26	非最优解
策略2		11	20	非最优解
策略3		13	26	非最优解

问题：如何保证获得最优解？

蛮力枚举

- 枚举所有商品组合

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合

C_5^1

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况）

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束

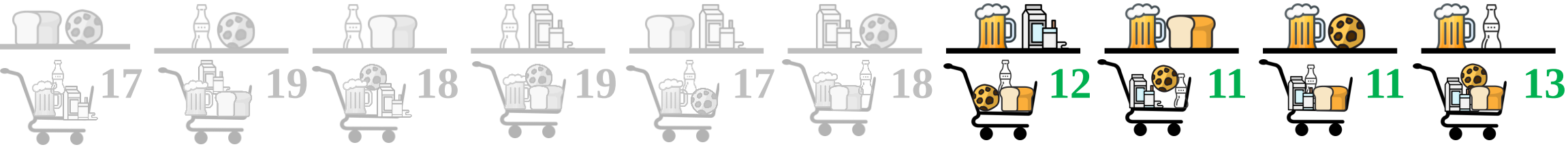
绿色数字：商品组合体积

背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束

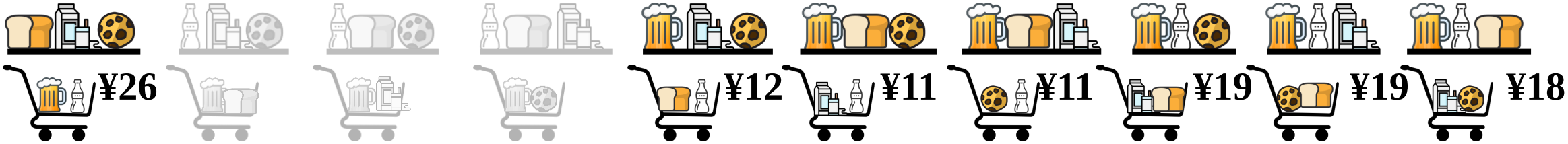


背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束

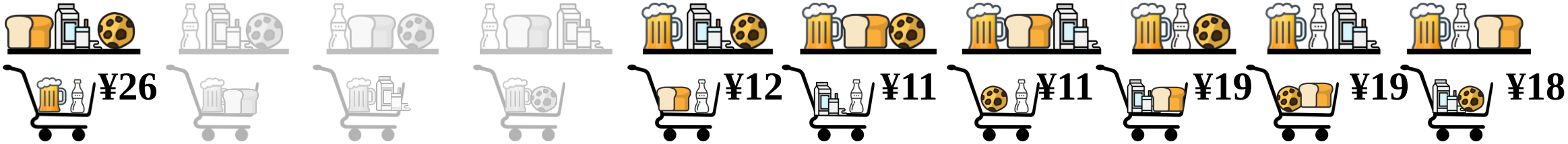


背包体积13

商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束



商品	价格	体积
啤酒	24	10
汽水	2	3
饼干	9	4
面包	10	5
牛奶	9	4

蛮力枚举：递归求解

- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解

KnapsackSR(1, 5, 13)

在第1到5个商品中选择
剩余体积:13

序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5
3	牛奶	9	4
4	汽水	2	3
5	啤酒	24	10

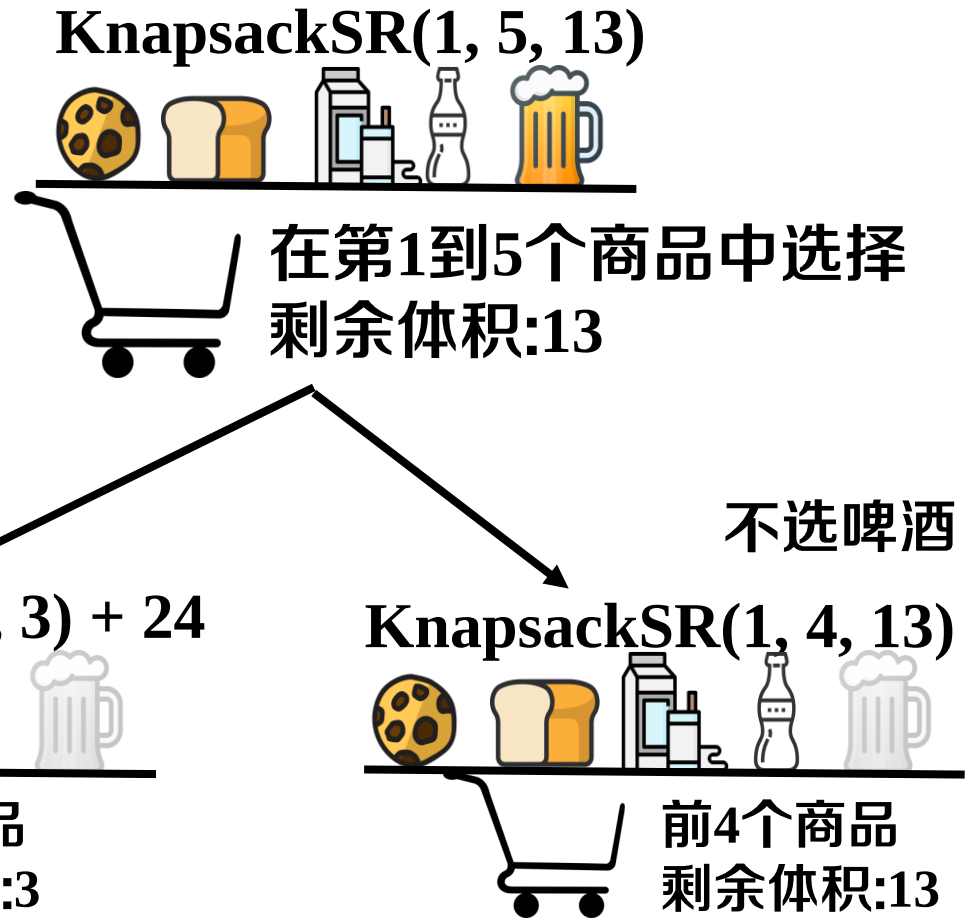
蛮力枚举：递归求解

- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24

序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5
3	牛奶	9	4
4	汽水	2	3
5	啤酒	24	10

蛮力枚举：递归求解

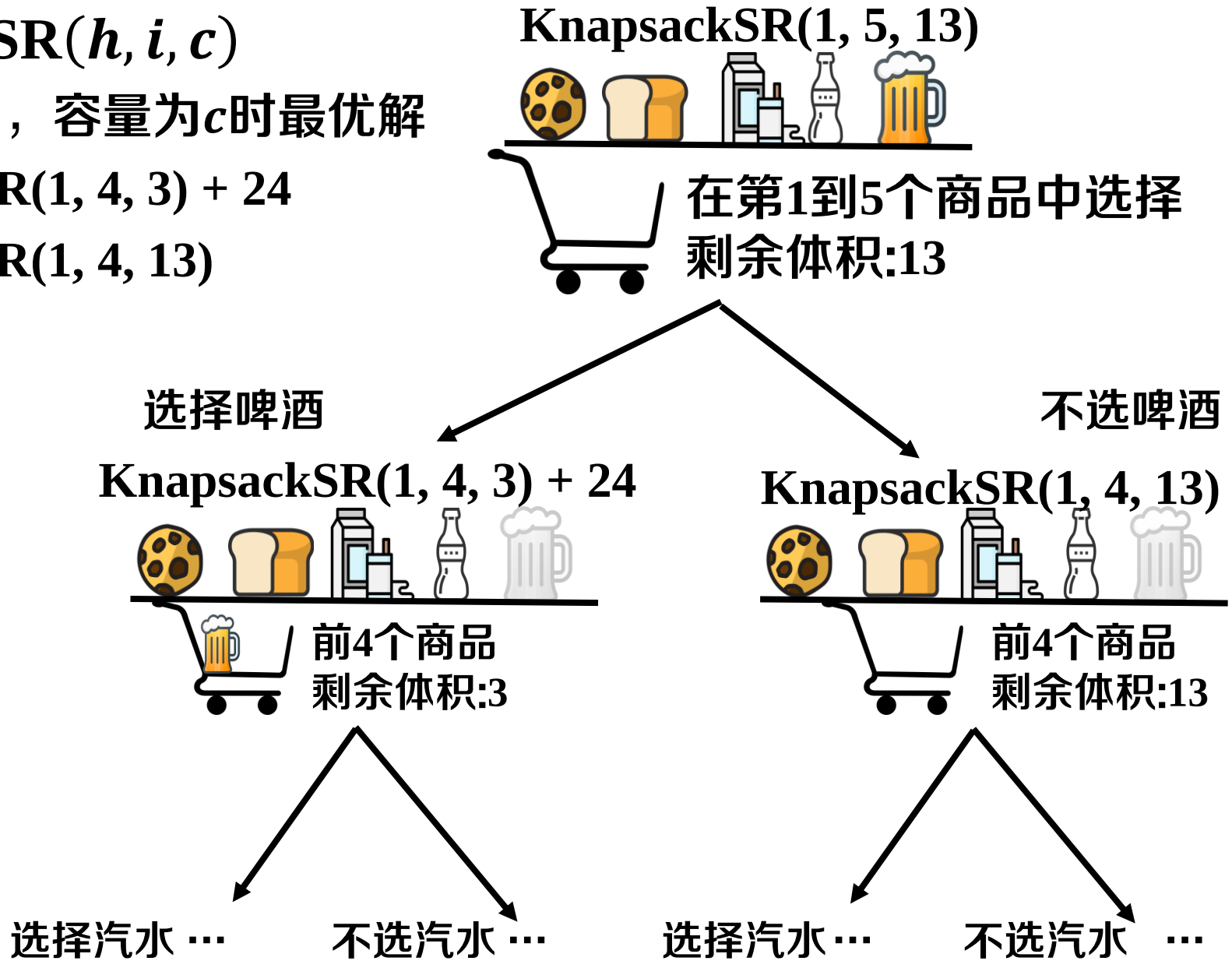
- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24
  - 不选啤酒：KnapsackSR(1, 4, 13)



序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5
3	牛奶	9	4
4	汽水	2	3
5	啤酒	24	10

蛮力枚举：递归求解

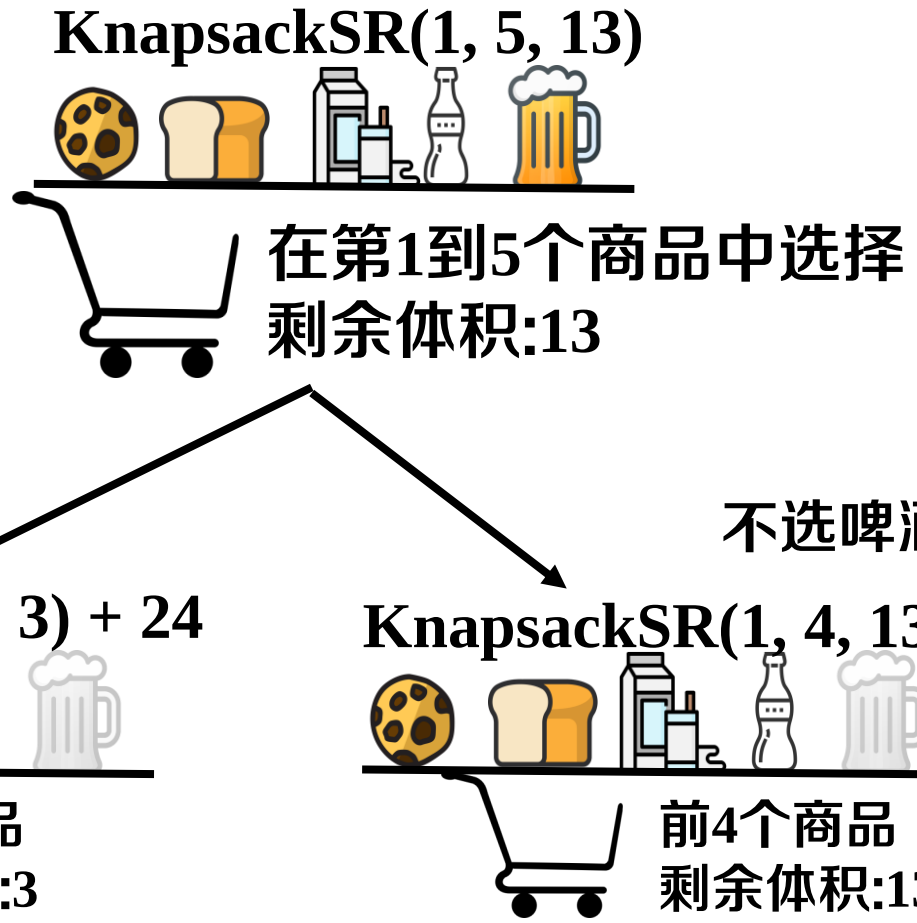
- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24
  - 不选啤酒：KnapsackSR(1, 4, 13)



序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5
3	牛奶	9	4
4	汽水	2	3
5	啤酒	24	10

蛮力枚举：递归求解

- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24
  - 不选啤酒：KnapsackSR(1, 4, 13)



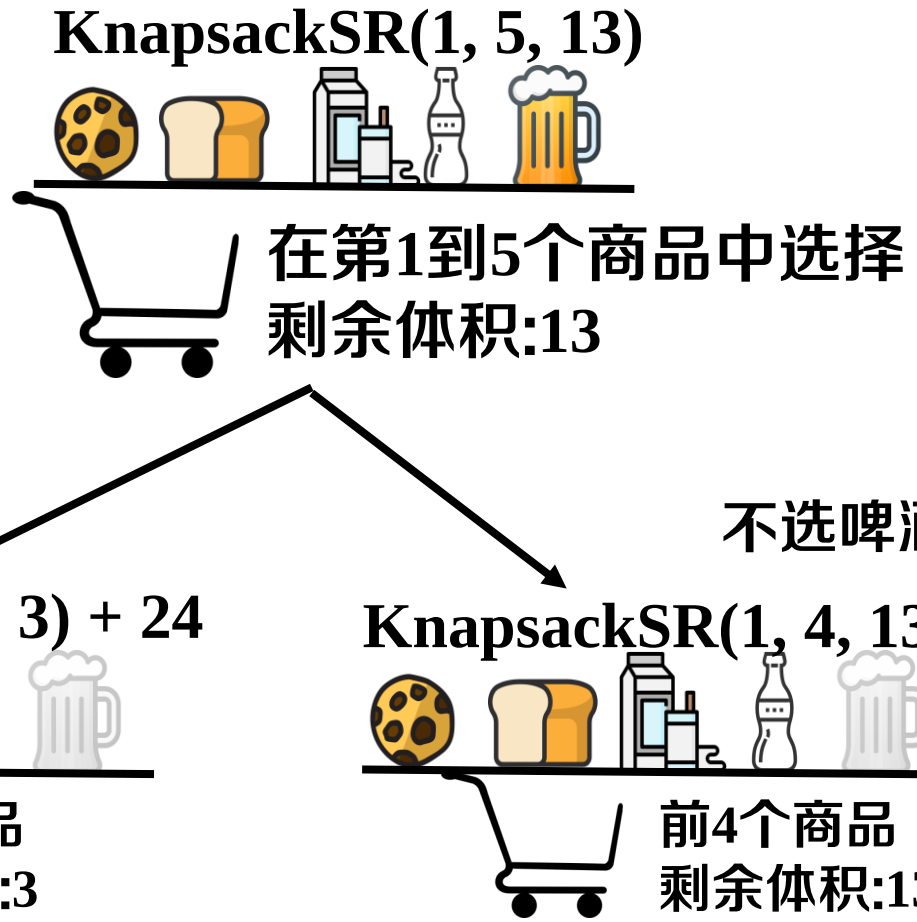
序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5

KnapsackSR(1, 5, 13) =

max{KnapsackSR(1, 4, 3) + 24, KnapsackSR(1, 4, 13)}

蛮力枚举：递归求解

- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24
  - 不选啤酒：KnapsackSR(1, 4, 13)



序号	商品	价格	体积
1	饼干	9	4
2	面包	10	5

$$\text{KnapsackSR}(h, i, c) =$$

$$\max\{\text{KnapsackSR}(h, i-1, c-v_i) + p_i, \text{KnapsackSR}(h, i-1, c)\}$$

蛮力枚举：伪代码

- KnapsackSR(h, i, c)：在第h个到第i个商品中，容量为c时最优解

输入: 商品集合$\{h, ..., i\}$, 背包容量c
输出: 最大总价格P

```
if c < 0 then
    return −∞
end
if i ≤ h − 1 then
    return 0
end
P_1 ←KnapsackSR(h, i − 1, c − v_i)
P_2 ←KnapsackSR(h, i − 1, c)
P ←max{P_1 + p_i, P_2}
return P
```

超出背包容量限制

蛮力枚举：伪代码

- KnapsackSR(h, i, c)：在第h个到第i个商品中，容量为c时最优解

输入: 商品集合$\{h, ..., i\}$, 背包容量c
输出: 最大总价格P
if $c < 0$ then
  return $-\infty$
end
if $i \leq h - 1$ then
  return 0
end
$P_1 \leftarrow \text{KnapsackSR}(h, i-1, c-v_i)$
$P_2 \leftarrow \text{KnapsackSR}(h, i-1, c)$
$P \leftarrow \max\{P_1 + p_i, P_2\}$
return P

所有商品已决策完成

蛮力枚举：伪代码

- KnapsackSR(h, i, c)：在第h个到第i个商品中，容量为c时最优解

输入: 商品集合$\{h, ..., i\}$, 背包容量c
输出: 最大总价格P
if $c < 0$ then
  return $-\infty$
end
if $i \leq h - 1$ then
  return 0
end
$P_1 \leftarrow \text{KnapsackSR}(h, i-1, c-v_i)$
$P_2 \leftarrow \text{KnapsackSR}(h, i-1, c)$
$P \leftarrow \max\{P_1 + p_i, P_2\}$
return P

选则商品i

蛮力枚举：伪代码

- KnapsackSR(h, i, c)：在第h个到第i个商品中，容量为c时最优解

输入: 商品集合$\{h, ..., i\}$, 背包容量c
输出: 最大总价格P

```
if c < 0 then
    return −∞
end
if i ≤ h − 1 then
    return 0
end
P_1 ←KnapsackSR(h, i − 1, c − v_i)
P_2 ←KnapsackSR(h, i − 1, c)
P ←max{P_1 + p_i, P_2}
return P
```

不选商品i

蛮力枚举：伪代码

- KnapsackSR(h, i, c)：在第h个到第i个商品中，容量为c时最优解

输入: 商品集合$\{h, ..., i\}$, 背包容量c
输出: 最大总价格P
if $c < 0$ then
  return $-\infty$
end
if $i \leq h - 1$ then
  return 0
end
$P_1 \leftarrow \text{KnapsackSR}(h, i-1, c-v_i)$
$P_2 \leftarrow \text{KnapsackSR}(h, i-1, c)$
$P \leftarrow \max\{P_1 + p_i, P_2\}$
return P

简化描述

蛮力枚举：伪代码

- **KnapsackSR(i, c)：前i个商品中，容量为c时最优解**

输入: 前i个商品, 背包容量c
输出: 最大总价格P

```
if c < 0 then
    return −∞
end
if i ≤ 0 then
    return 0
end
P_1 ← KnapsackSR(i − 1, c − v_i)
P_2 ← KnapsackSR(i − 1, c)
P ← max{P_1 + p_i, P_2}
return P
```

简化描述

蛮力枚举：伪代码

- **KnapsackSR(i, c)：前i个商品中，容量为c时最优解**

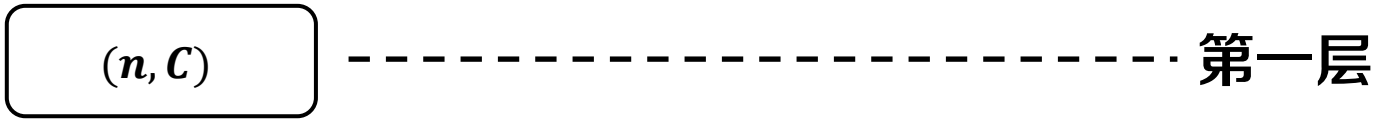
输入: 前i个商品, 背包容量c
输出: 最大总价格P

```
if c < 0 then
    return −∞
end
if i ≤ 0 then
    return 0
end
P_1 ←KnapsackSR(i − 1, c − v_i)
P_2 ←KnapsackSR(i − 1, c)
P ←max{P_1 + p_i, P_2}
return P
```

确定最优子问题

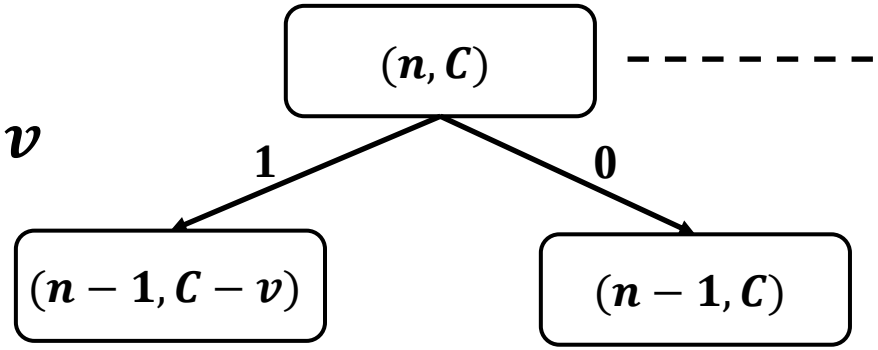
蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v



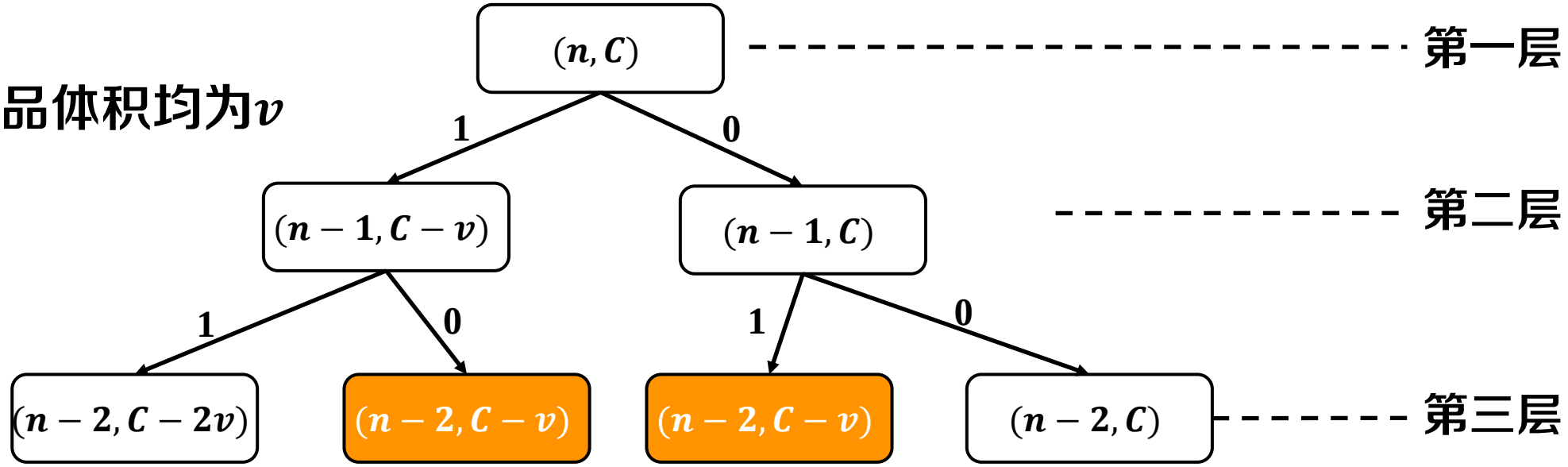
蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v



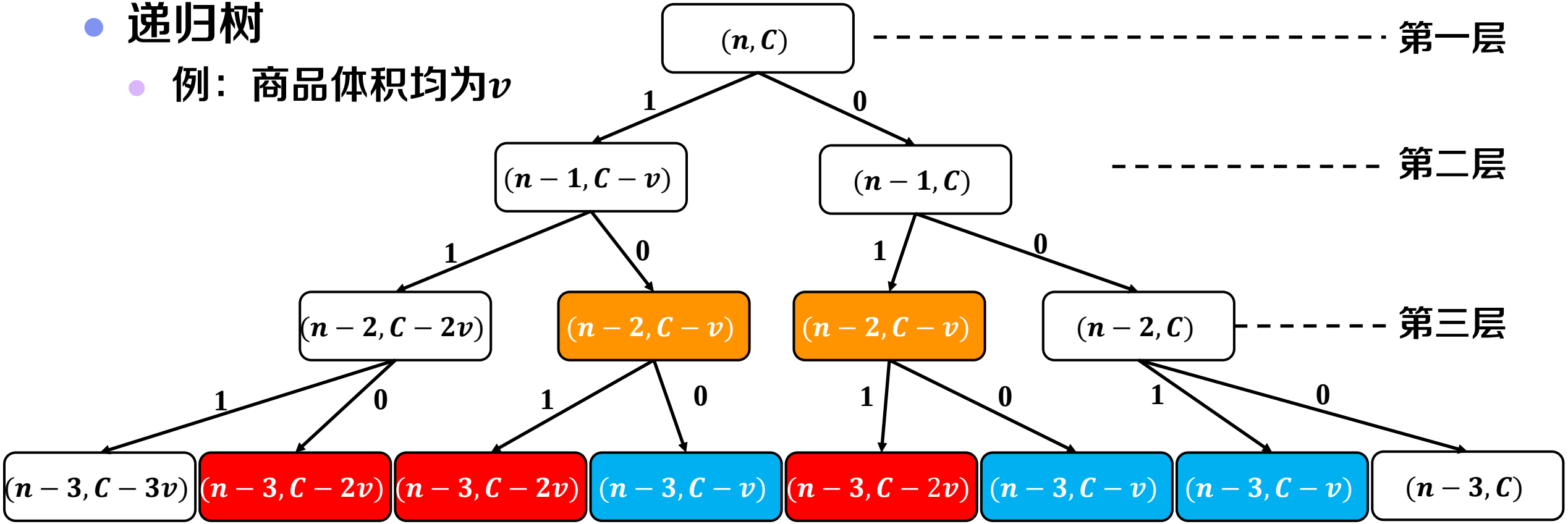
蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v



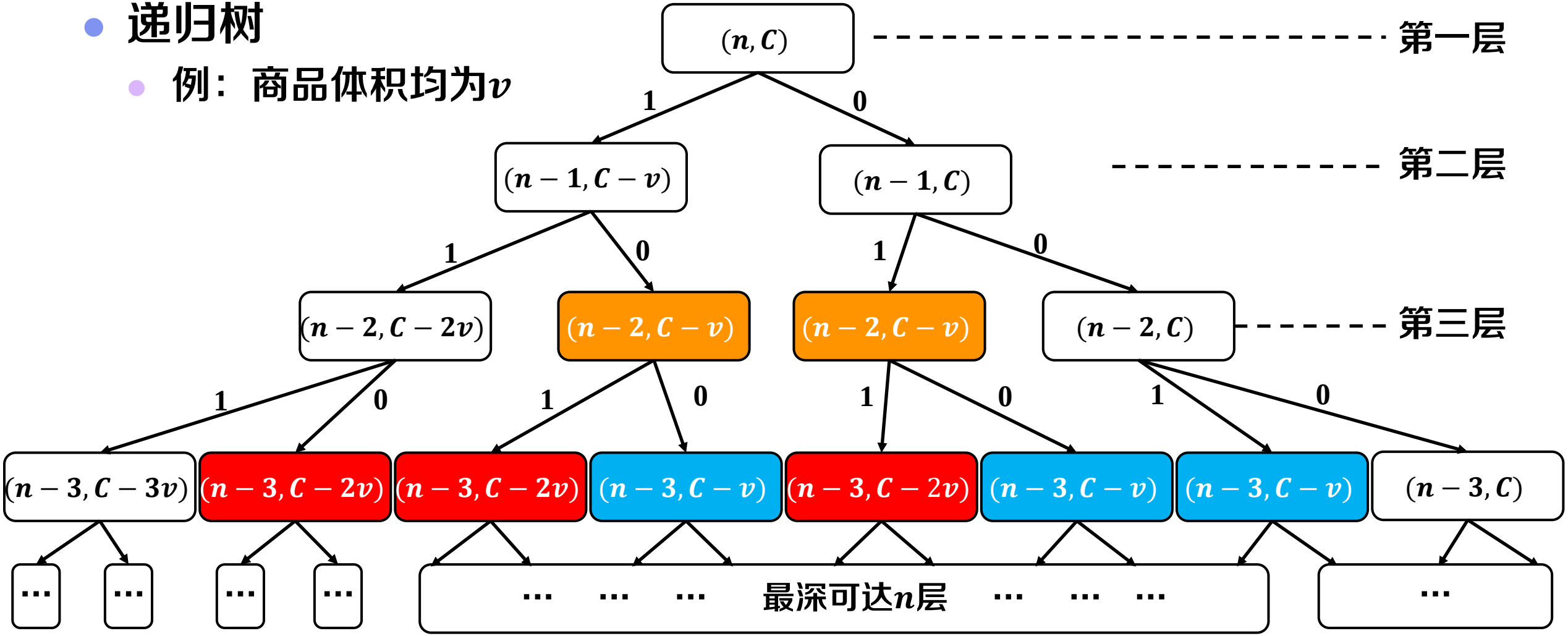
蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v



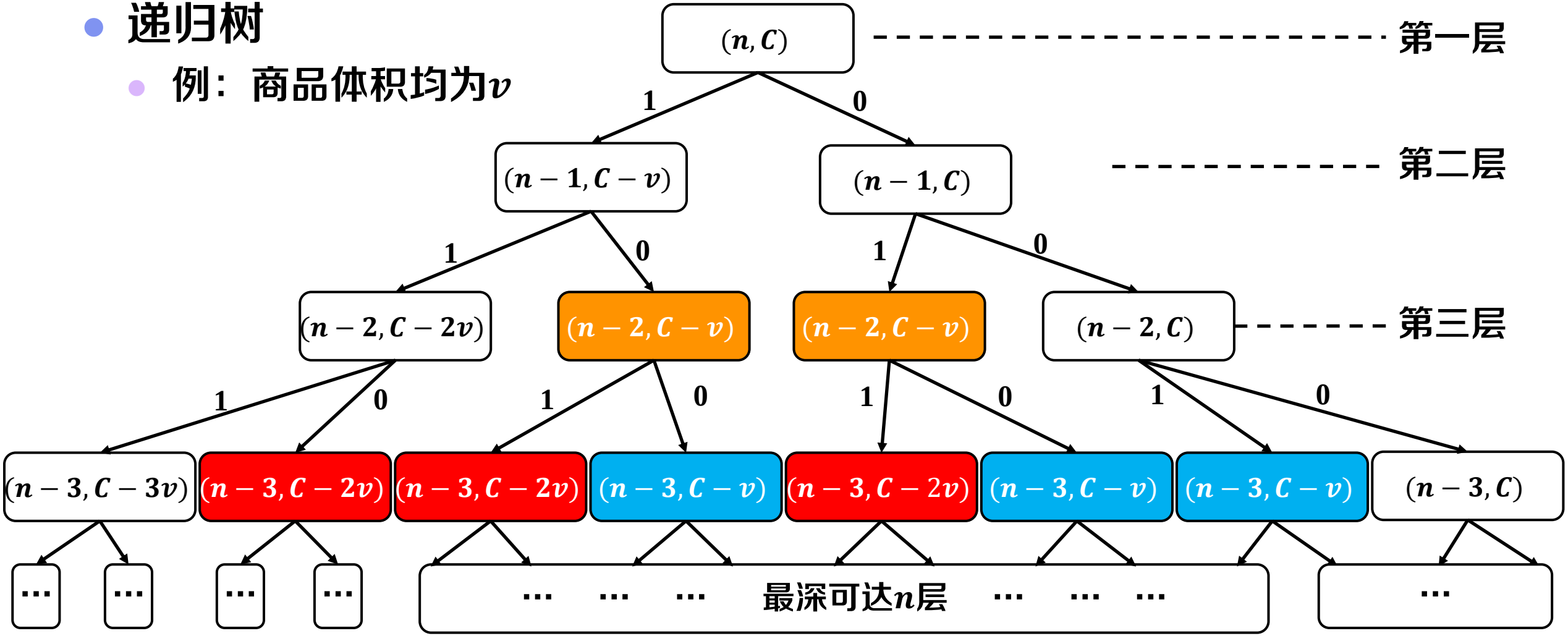
蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v



蛮力枚举：复杂度

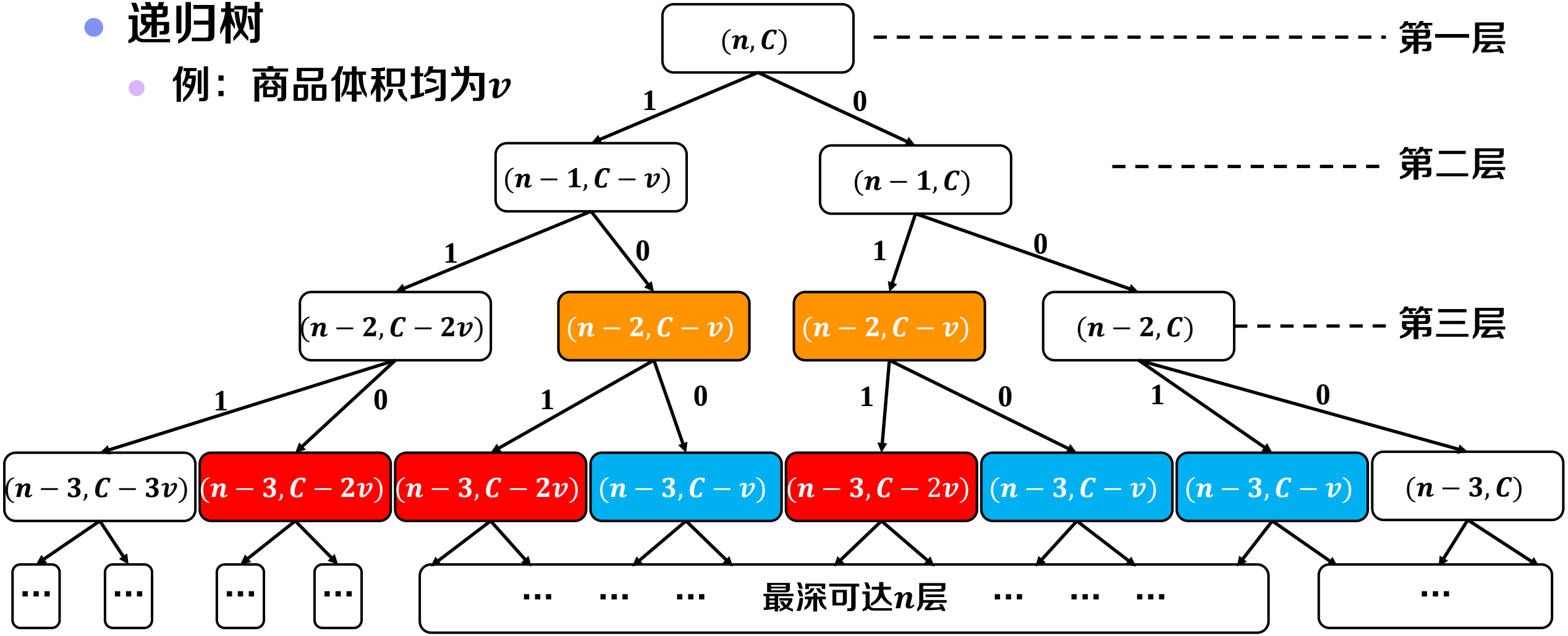
- 递归树
  - 例：商品体积均为v



- 重复求解大量子问题 $O(2^n)$

蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v

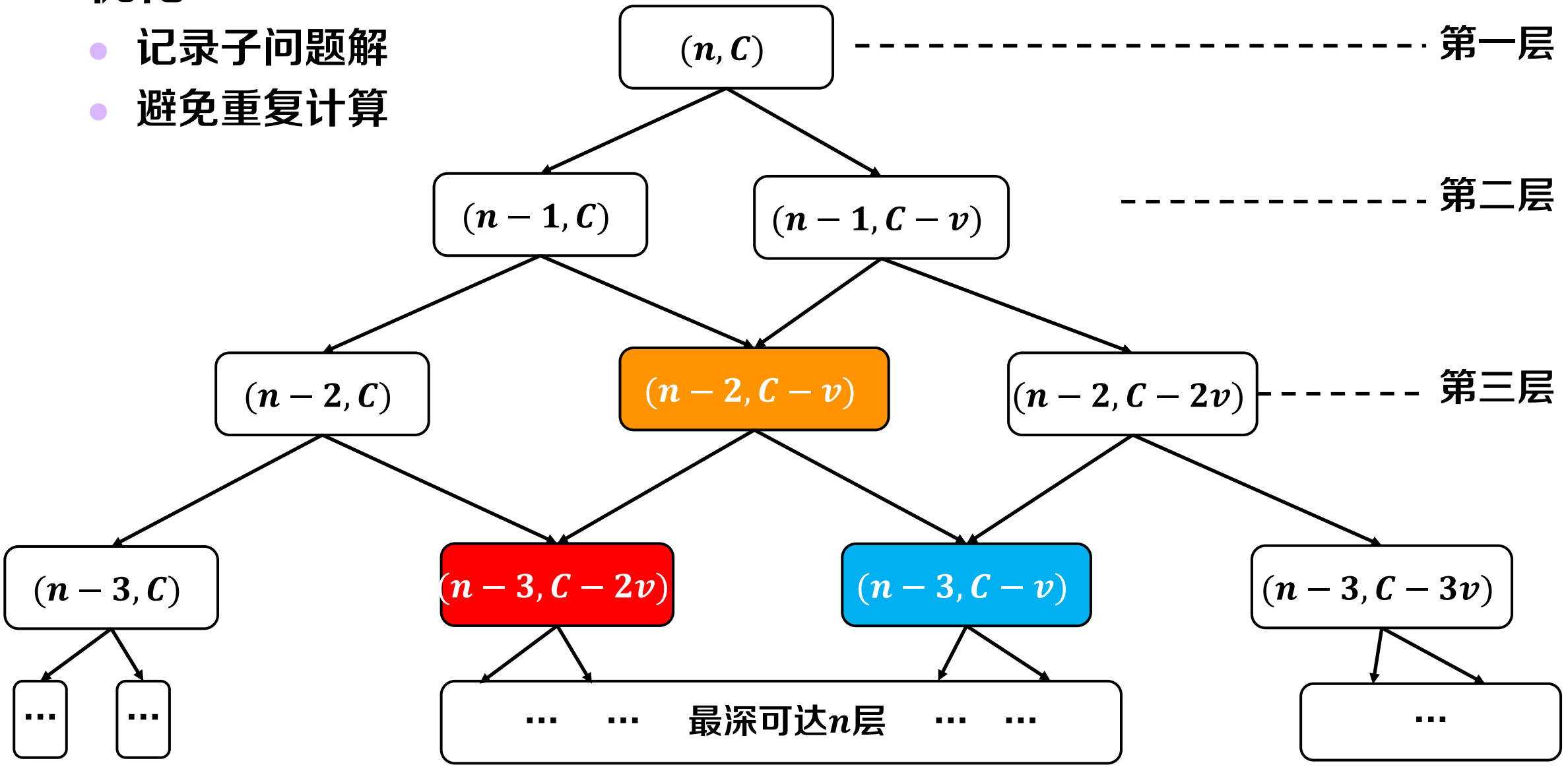


- 重复求解大量子问题 $O(2^n)$

问题：如何优化？

从蛮力枚举到带备忘递归

- 优化
  - 记录子问题解
  - 避免重复计算



带备忘递归：伪代码

- **KnapsackMR(i, c)**

输入: 商品集合$\{1, \ldots, i\}$, 背包容量c
输出: 最大总价格$P[i, c]$
if $c < 0$ **then**
  return $-\infty$
end
if $i \leq 0$ **then**
  return 0
end

$P_1 \leftarrow \text{KnapsackMR}(i-1, c-v_i)$
$P_2 \leftarrow \text{KnapsackMR}(i-1, c)$
$P[i, c] \leftarrow \max\{P_1 + p_i, P_2\}$
return $P[i, c]$

带备忘递归：伪代码

- **KnapsackMR(i, c)**

输入: 商品集合$\{1, ..., i\}$, 背包容量c
输出: 最大总价格$P[i, c]$

```
if c < 0 then
    return −∞
end
if i ≤ 0 then
    return 0
end
P_1 ← KnapsackMR(i − 1, c − v_i)
P_2 ← KnapsackMR(i − 1, c)
P[i, c] ← max{P_1 + p_i, P_2}
return P[i, c]
```

构造备忘录$P[i, c]$
$P[i, c]$表示在前i个商品中选择，背包容量为c时的最优解

带备忘递归：伪代码

- **KnapsackMR(i, c)**

输入: 商品集合$\{1, ..., i\}$, 背包容量c
输出: 最大总价格$P[i, c]$

```
if c < 0 then
    return −∞
end
if i ≤ 0 then
    return 0
end
if P[i, c] ≠ NULL then
    return P[i, c]
end
P_1 ←KnapsackMR(i − 1, c − v_i)
P_2 ←KnapsackMR(i − 1, c)
P[i, c] ← max{P_1 + p_i, P_2}
return P[i, c]
```

构造备忘录$P[i, c]$
$P[i, c]$表示在前i个商品中选择，背包容量为c时的最优解

带备忘递归：伪代码

- **KnapsackMR(i, c)**

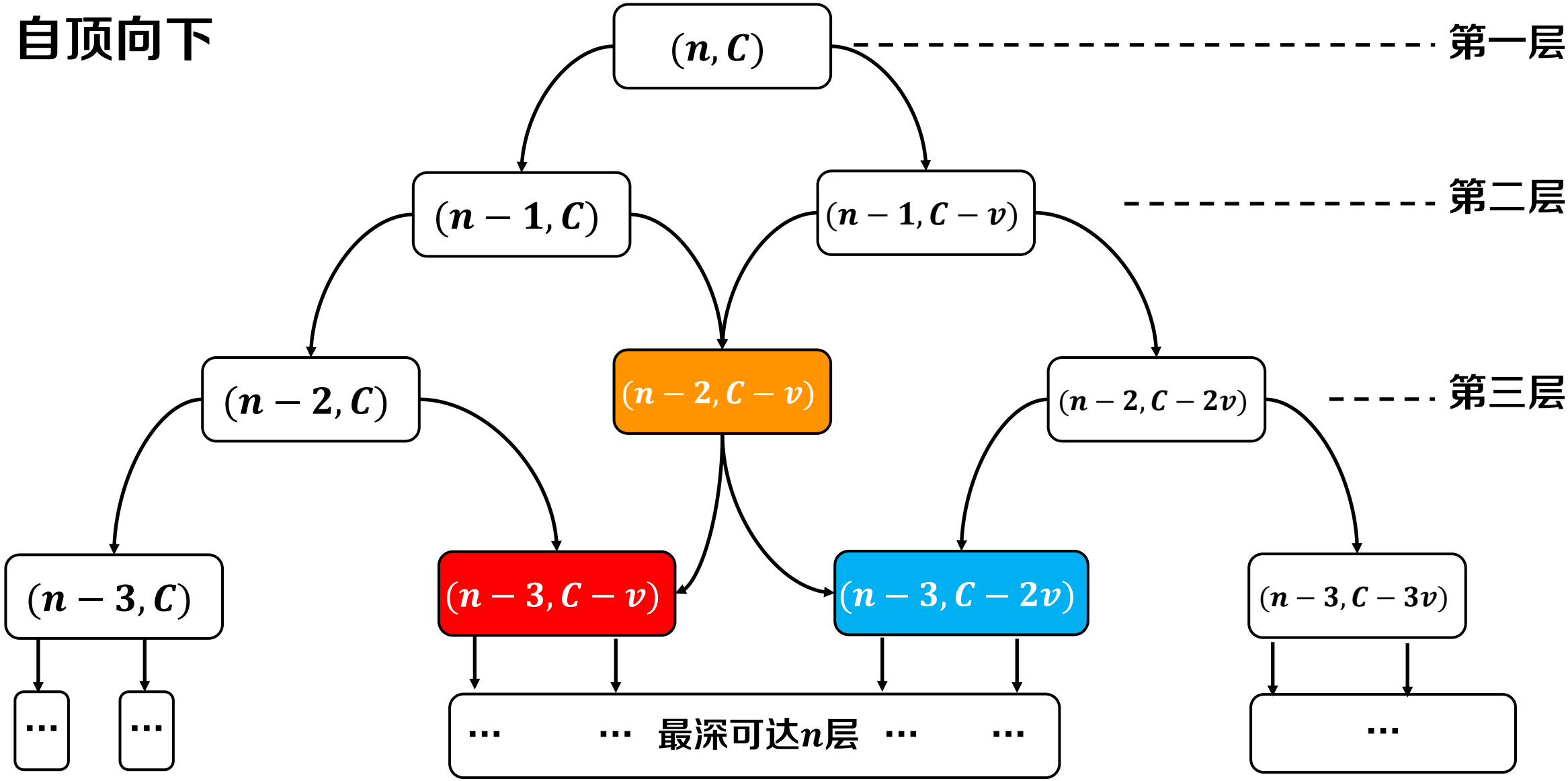
输入: 商品集合$\{1, \ldots, i\}$, 背包容量c
输出: 最大总价格$P[i, c]$
if $c < 0$ then
  return $-\infty$
end
if $i \leq 0$ then
  return 0
end
if $P[i, c] \neq NULL$ then
  return $P[i, c]$
end
$P_1 \leftarrow$KnapsackMR$(i-1, c-v_i)$
$P_2 \leftarrow$KnapsackMR$(i-1, c)$
$P[i, c] \leftarrow \max\{P_1 + p_i, P_2\}$
return $P[i, c]$

重复子问题

构造备忘录$P[i, c]$
$P[i, c]$表示在前i个商品中选择，背包容量为c时的最优解

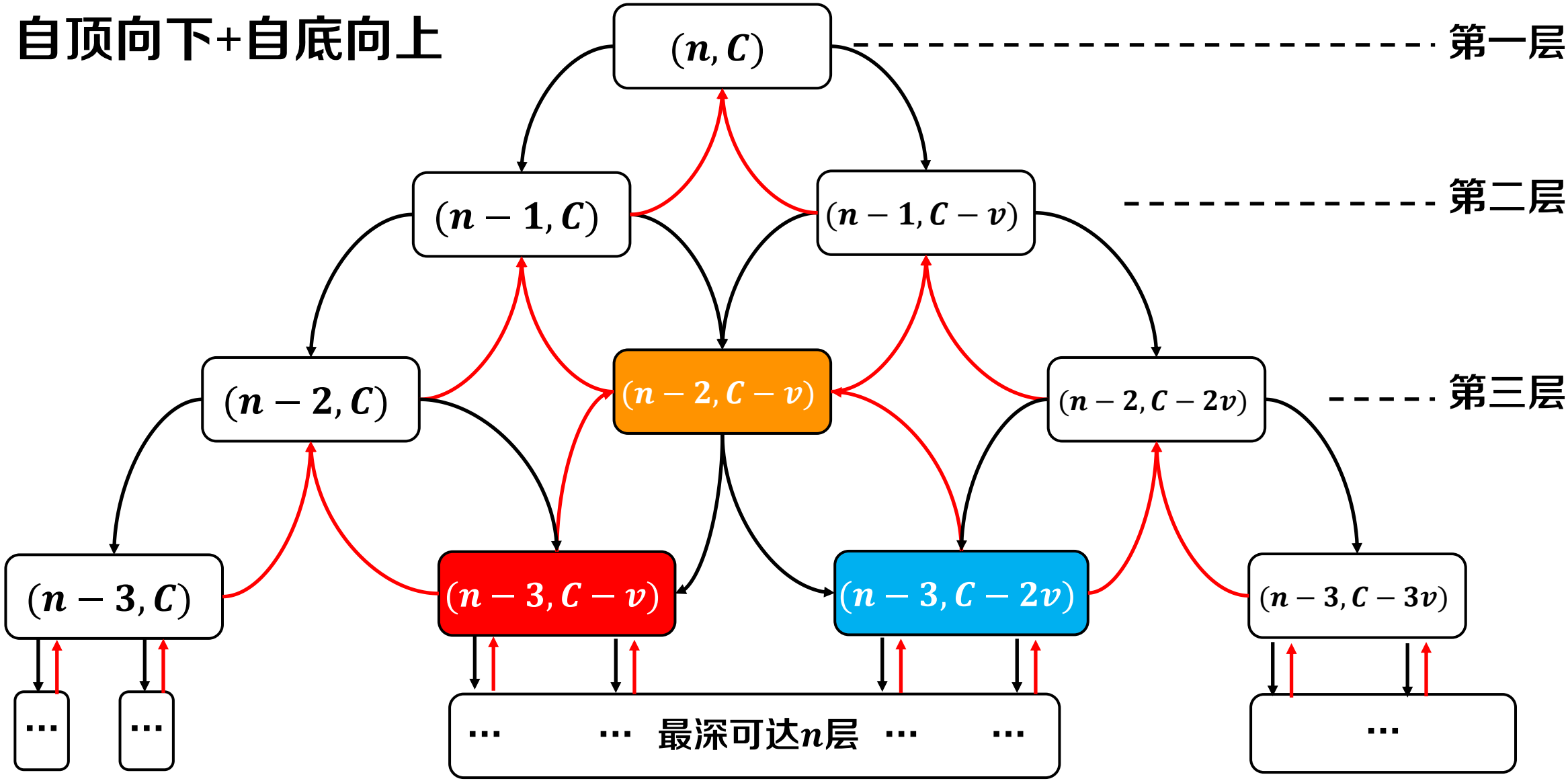
带备忘递归：计算顺序

- 自顶向下

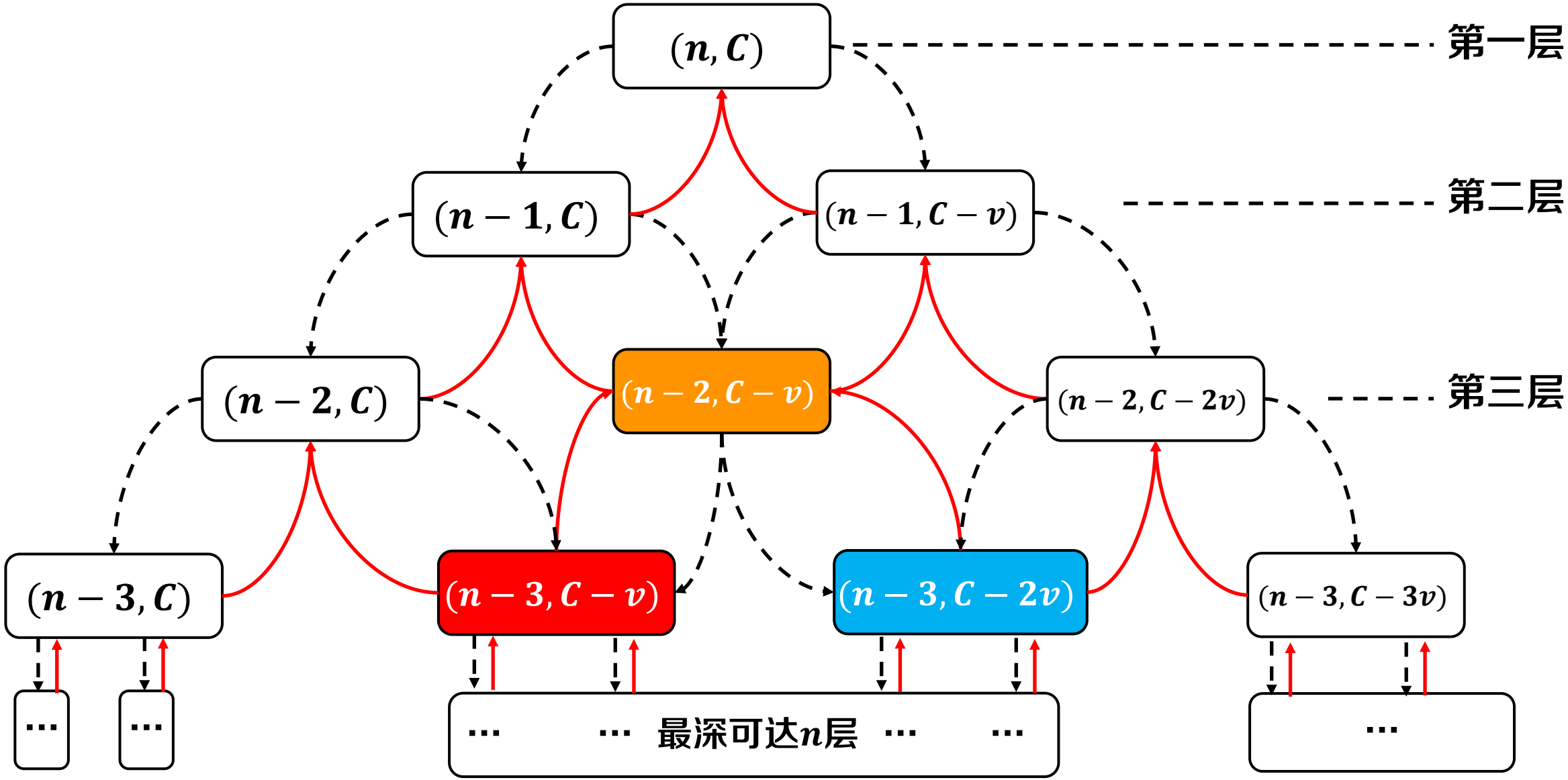


带备忘递归：计算顺序

- 自顶向下+自底向上



带备忘递归：计算顺序



问题：是否可以不递归，直接求解$P[i, c]$?

递推计算

- 初始化
  - 容量为0时：$P[i, 0] = 0$
  - 没有商品时：$P[0, c] = 0$

$P[i, c]$	$c = 0$	1	2	3	…	10	11	12	13
$i = 0$	0	0	0	0	…	0	0	0	0
1	0								
…	0								
n	0								

递推计算

- 确定计算顺序
  - $P[i-1, c-v_i]$和$P[i-1, c]$位于$P[i, c]$的左上方

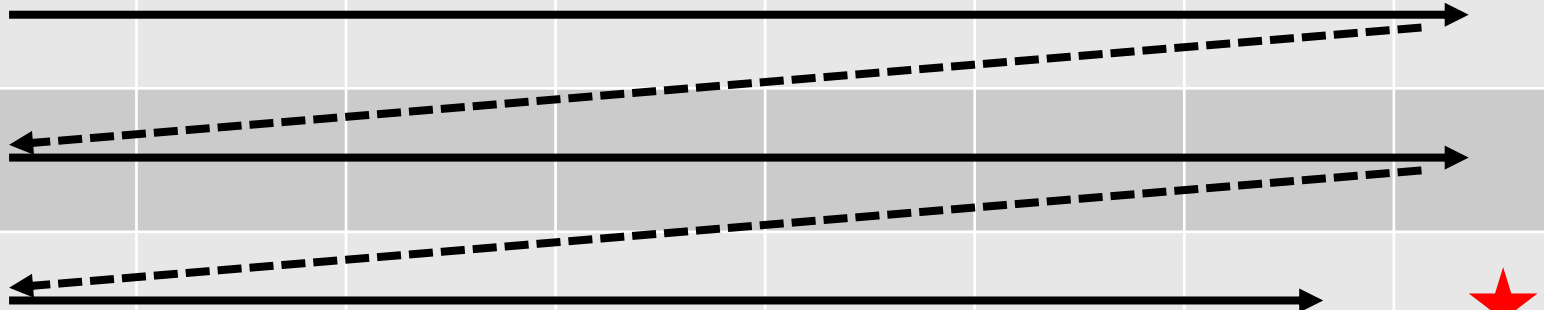
$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0					$P[i-1,c]$			
…	0			$P[i-1,c-v_i]+p_i$		$P[i,c]$			
n	0								

问题：观察子问题依赖关系，如何确定计算顺序?

递推计算

- 确定计算顺序
  - 按从左到右、从上到下的顺序计算

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
…	0								
n	0								★



问题：观察子问题依赖关系，如何确定计算顺序?

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0													
2	0													
3	0													
4	0													
5	0													

初始化

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0													
2	0													
3	0													
4	0													
5	0													

$v[i] > c$

递推公式：$P[i,c] = \max\{P[i-1, c-v[i]] + p[i],\ P[i-1, c]\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0													
2	0													
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{P[i-1, c-v[i]] + p[i],\ P[i-1, c]\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0				
2	0													
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{P[i-1, c-v[i]] + p[i],\ P[i-1, c]\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0				
2	0													
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{P\left[i-1, c-v[i]\right] + p[i],\ P[i-1,c]\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0				
2	0													
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{P[0,0] + p[1],\ P[0,10]\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24			
2	0													
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{0 + 24,\ 0\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24		
2	0													
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	
2	0													
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0													
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0											
3	0													
4	0													
5	0													

$v[i] > c$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2										
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{0+2,\ 0\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2									
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2								
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2							
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2						
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2					
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2				
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2				
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24			
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{0+2,\ 24\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24		
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	
3	0													
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0													
4	0													
5	0													

递推公式：$P[i,c] = \max\{24+2,\ 24\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2										
4	0													
5	0													

$v[i] > c$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9									
4	0													
5	0													

递推公式：$P[i,c] = \max\{0+9,\ 2\}$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9								
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9							
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11						
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11					
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11				
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24			
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24		
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0													
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9									
5	0													

$v[i] > c$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10								
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10							
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11						
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12					
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19				
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24			
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24		
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0													

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2										

$v[i] > c$

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9									

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i, c]$	$c = 0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i = 0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10								

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10							

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11						

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18					

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19				

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24			

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24		

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

最优解

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

最优解

问题：如何确定选取了哪些商品？

最优解追踪

- 递推公式
  - $P[i,c]=\max\{P[i-1,c-v_i\,]+p_i, P[i-1,c]\}$

- 记录决策过程
  - $Rec[i,c]=\begin{cases}1, & \text{选择商品}\\ 0, & \text{不选商品}\end{cases}$

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
…	0					$P[i,c]$			
n	0								

$Rec[i,c]=1$

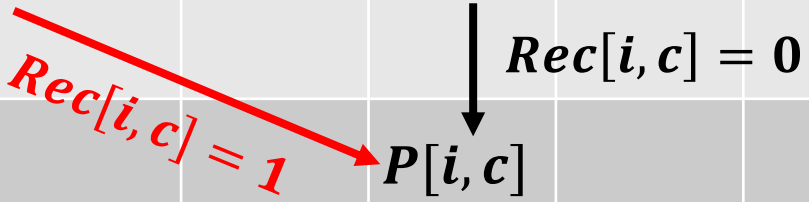
$Rec[i,c]=0$

最优解追踪

- 递推公式
  - $P[i,c]=\max\{P[i-1,c-v_i]+p_i, P[i-1,c]\}$

- 记录决策过程
  - $Rec[i,c]=\begin{cases}1, & \text{选择商品}\\ 0, & \text{不选商品}\end{cases}$

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
…	0					$P[i,c]$			
n	0								

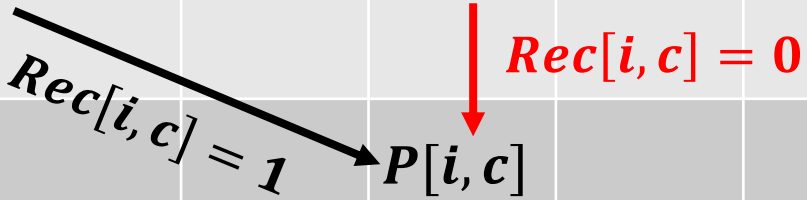


最优解追踪

- 递推公式
  - $P[i,c]=\max\{P[i-1,c-v_i]+p_i, P[i-1,c]\}$

- 记录决策过程
  - $Rec[i,c]=\begin{cases}1, & \text{选择商品}\\ 0, & \text{不选商品}\end{cases}$

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
…	0					$P[i,c]$			
n	0								

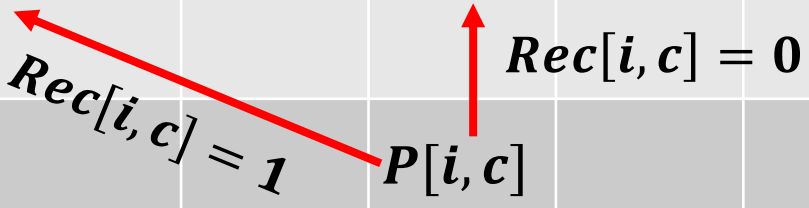


最优解追踪

- 递推公式
  - $P[i,c] = \max\{P[i-1, c-v_i\,] + p_i, P[i-1,c]\}$

- 回溯解决方案
  - 倒序判断是否选择商品
  - 根据选择结果，确定最优子问题

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
…	0					$P[i,c]$			
n	0								



算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0				
2	0													
3	0													
4	0													
5	0													

$v[i] > c$

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0				
2	0													
3	0													
4	0													
5	0													

当前状态最优解不包含商品i

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24			
2	0													
3	0													
4	0													
5	0													

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1			
2	0													
3	0													
4	0													
5	0													

当前状态最优解包含商品i

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24			
3	0													
4	0													
5	0													

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0			
3	0													
4	0													
5	0													

当前状态最优解不包含商品i

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	1

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

选取的商品：5　　$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	1

最优解包含商品5

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

选取的商品：5, 4　　$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	1

最优解包含商品4

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

选取的商品：5, 4, 3　$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	1

当前状态最优解包含商品3

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

选取的商品：5, 4, 3　$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	1

最优解不包含商品2

算法实例

v_i	10	3	4	5	4
p_i	24	2	9	10	9

选取的商品：5, 4, 3　$C = 13$

$P[i,c]$	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=0$	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	24	24	24	24
2	0	0	0	2	2	2	2	2	2	2	24	24	24	26
3	0	0	0	2	9	9	9	11	11	11	24	24	24	26
4	0	0	0	2	9	10	10	11	12	19	24	24	24	26
5	0	0	0	2	9	10	10	11	18	19	24	24	24	28

Rec	$c=0$	1	2	3	4	5	6	7	8	9	10	11	12	13
$i=1$	0	0	0	0	0	0	0	0	0	0	1	1	1	1
2	0	0	0	1	1	1	1	1	1	1	0	0	0	1
3	0	0	0	0	1	1	1	1	1	1	0	0	0	0
4	0	0	0	0	0	1	1	0	1	1	0	0	0	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	1

递推计算：伪代码

- KnapsackDP(n, p, v, C)

输入: 商品数量n,各商品的价值p, 各商品的体积v, 背包容量C
输出: 商品价格的最大值,最优解方案
//初始化
创建二维数组 $P[0..n, 0..C]$ 和 $Rec[0..n, 0..C]$

```
for i ← 0 to C do
  |   P[0, i] ← 0
end
for i ← 0 to n do
  |   P[i, 0] ← 0
end
```

初始化

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end
    end
end
```

依次计算子问题

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end
    end
end
```

选择商品i

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end
    end
end
```

记录价格和决策

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end
    end
end
```

不选商品i

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//输出最优解方案
K ← C
for i ← n to 1 do
    if Rec[i, K] = 1 then
        print 选择商品i
        K ← K − v[i]
    end
    else
        print 不选商品i
    end
end
return P[n, C]
```

倒序判断是否选择该商品

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//输出最优解方案
K ← C
for i ← n to 1 do
    if Rec[i, K] = 1 then
        print 选择商品i
        K ← K − v[i]
    end
    else
        print 不选商品i
    end
end
return P[n, C]
```

选择商品i

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//输出最优解方案
K ← C
for i ← n to 1 do
    if Rec[i, K] = 1 then
        print 选择商品i
        K ← K − v[i]
    end
    else
        print 不选商品i
    end
end
return P[n, C]
```

回溯子问题

递推计算：伪代码

- **KnapsackDP(n, p, v, C)**

```
//输出最优解方案
K ← C
for i ← n to 1 do
    if Rec[i, K] = 1 then
        print 选择商品i
        K ← K − v[i]
    end
    else
        print 不选商品i
    end
end
return P[n, C]
```

不选商品i

时间复杂度分析

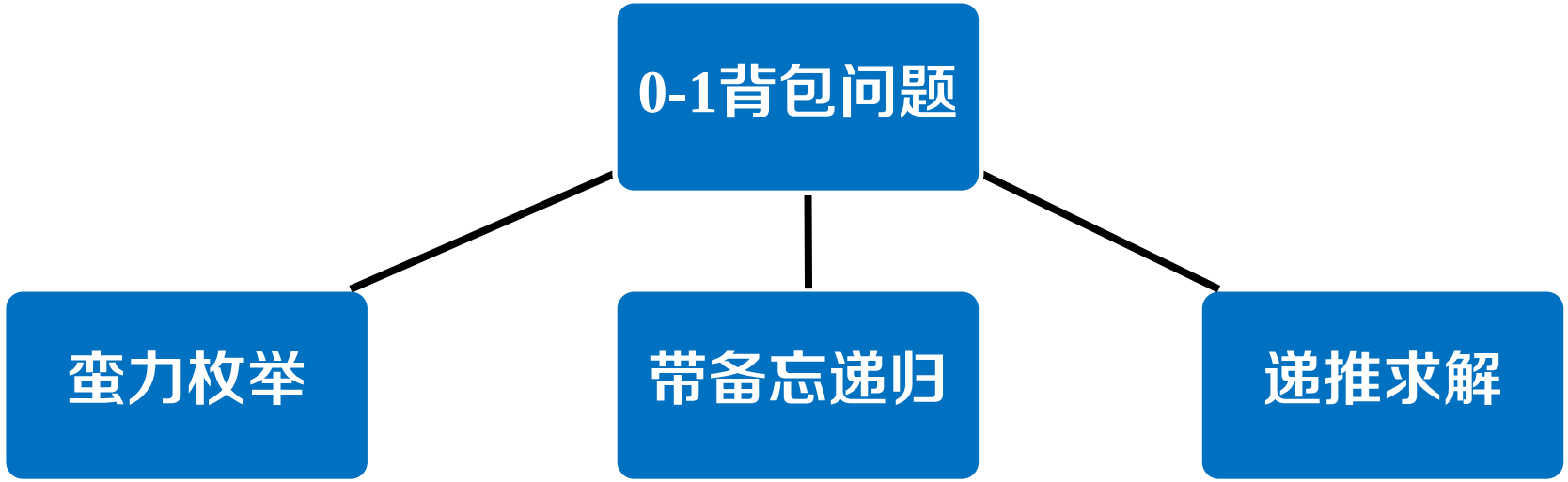
```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end
    end
end
```

$O(C)$

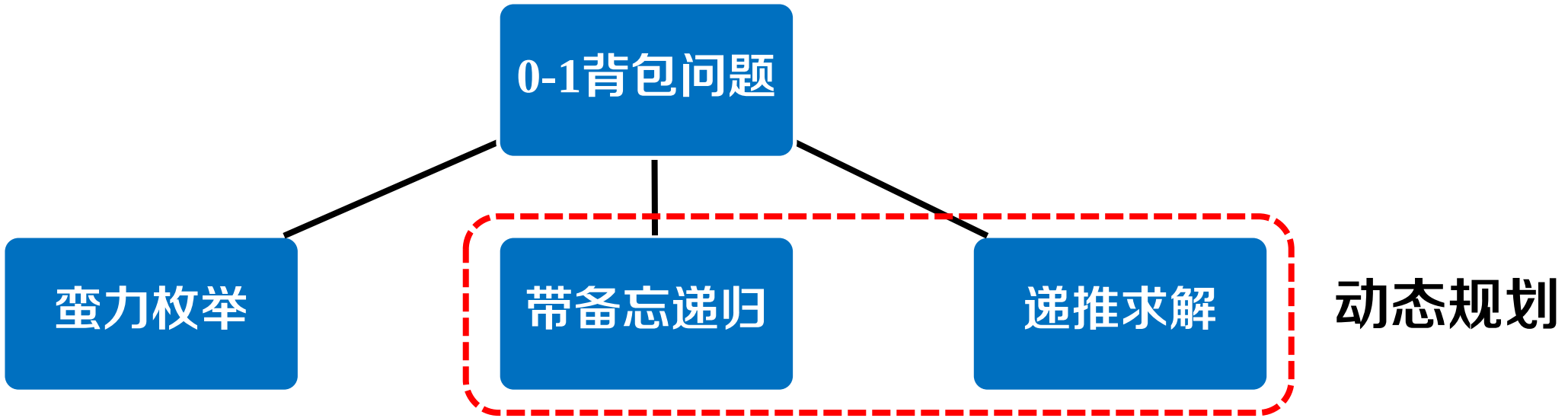
$O(n \cdot C)$

时间复杂度：$O(n \cdot C)$

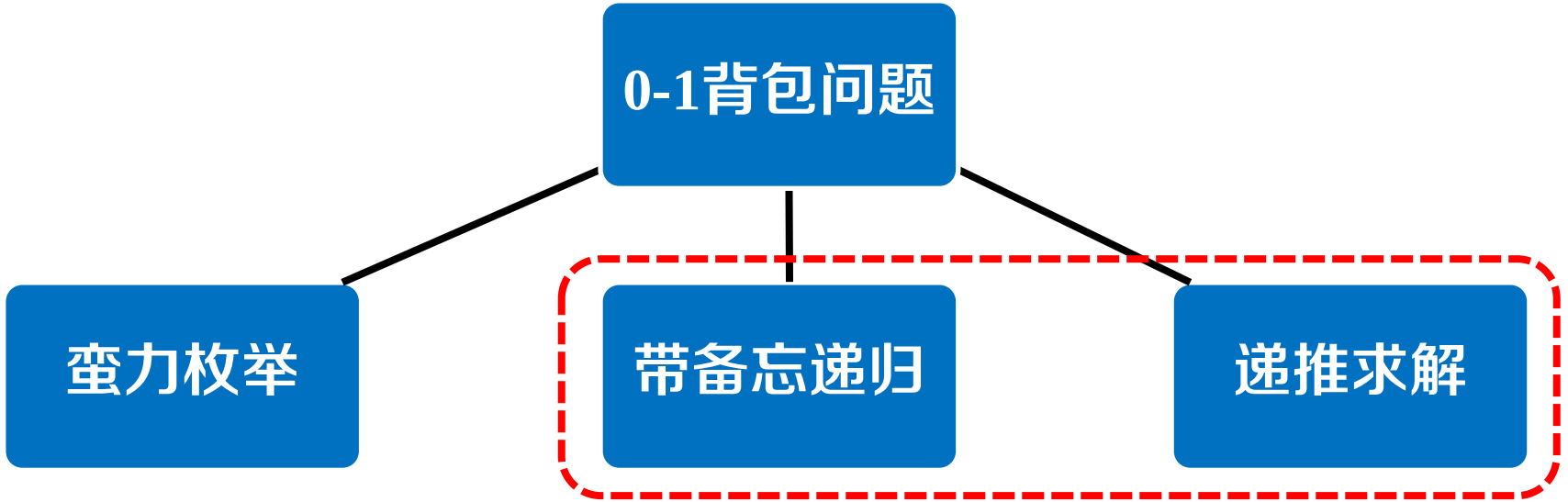
动态规划



动态规划

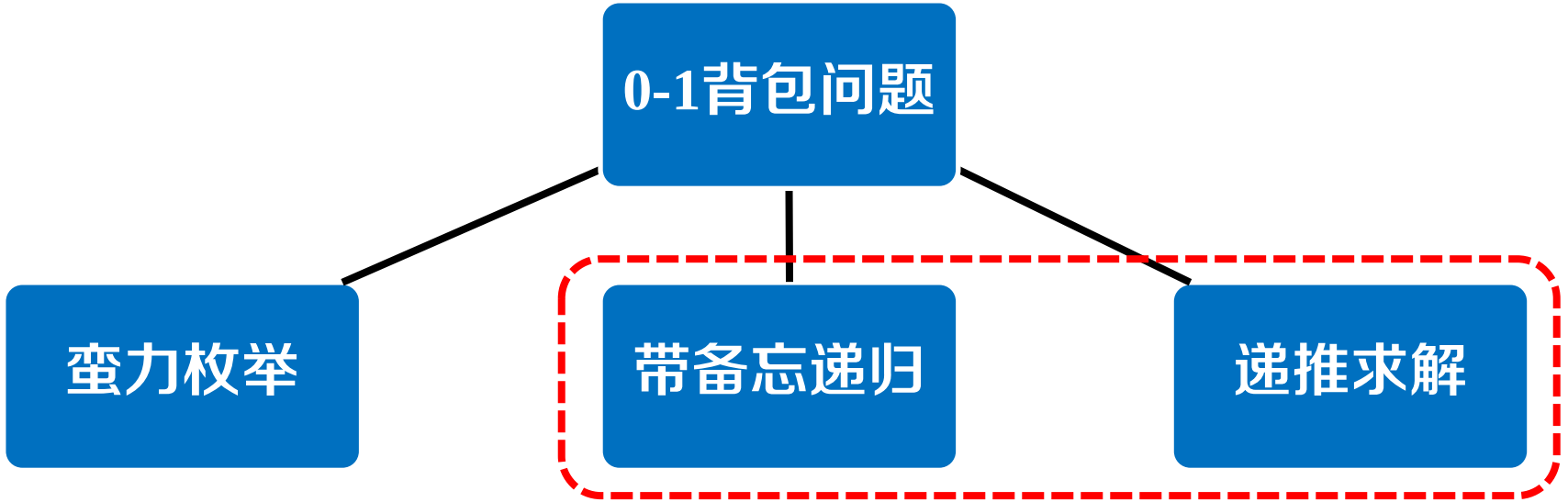


动态规划



	带备忘递归	递推求解
相同	分解问题，寻找关系	
不同	自顶向下	自底向上

动态规划

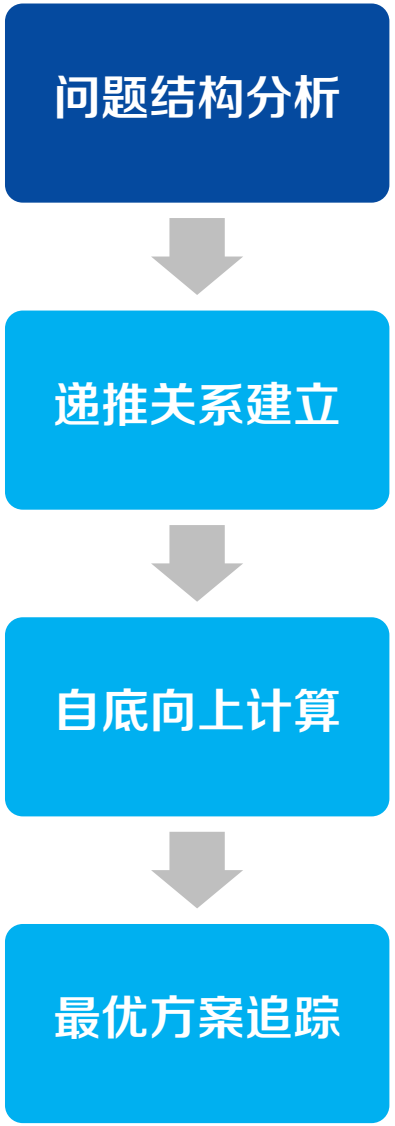


	带备忘递归	递推求解
相同	分解问题，寻找关系	
不同	自顶向下	自底向上

更高效

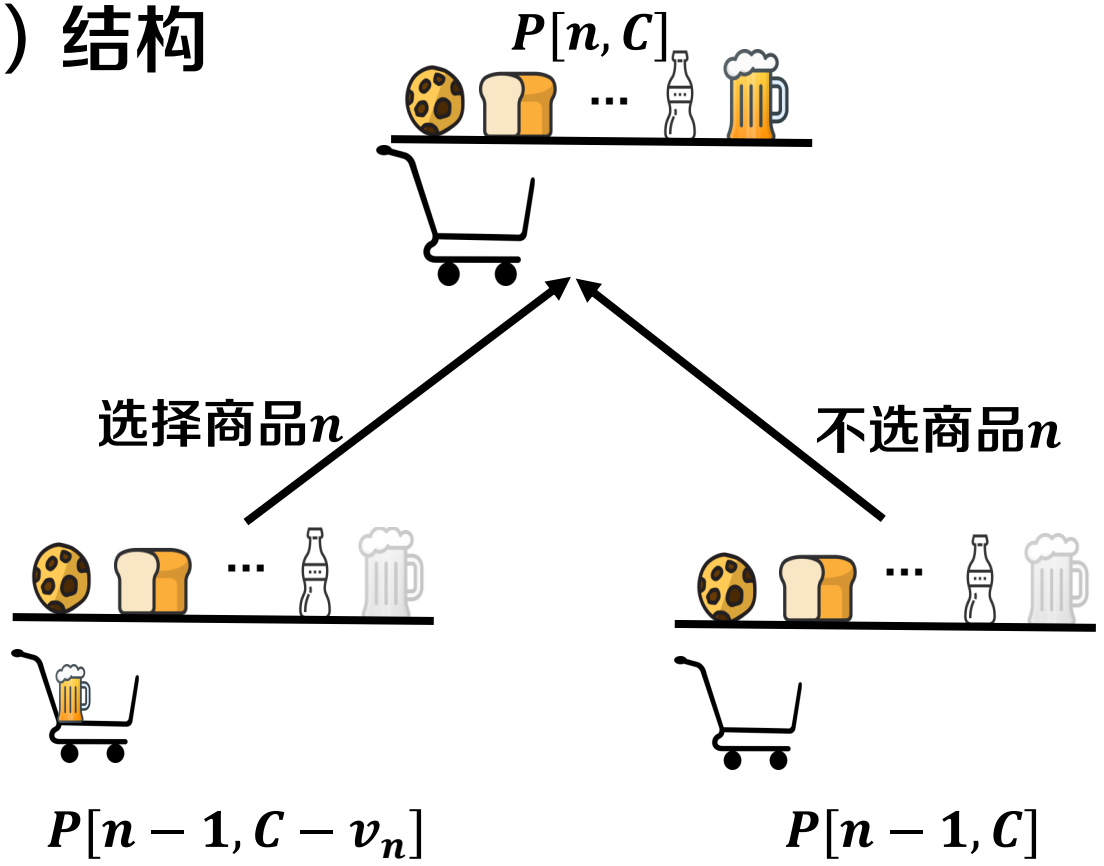
动态规划：一般步骤

- 给出问题表示
  - $P[i, c]$：前i个商品可选、背包容量为c时的最大总价格
- 明确原始问题
  - $P[n, C]$：前n个商品可选、背包容量为C时的最大总价格



动态规划：一般步骤

- 分析最优（子）结构



问题结构分析

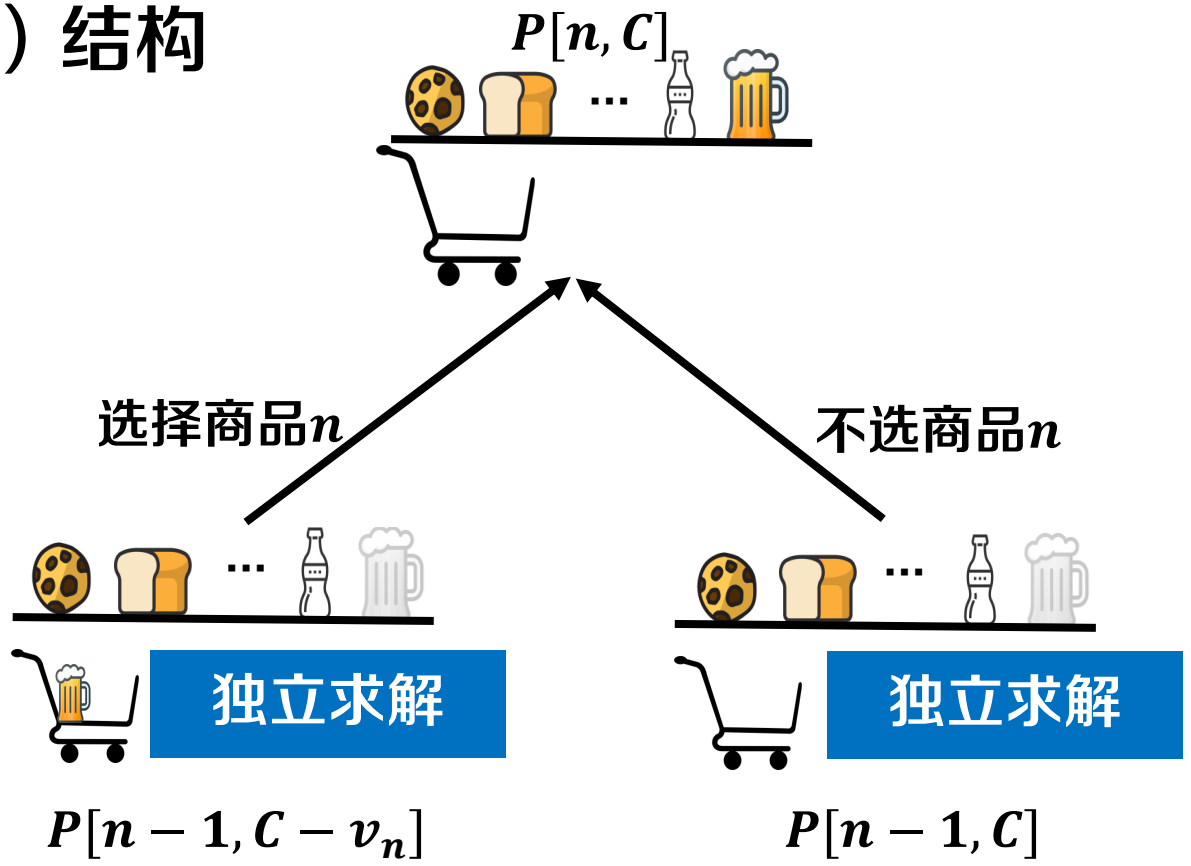
递推关系建立

自底向上计算

最优方案追踪

动态规划：一般步骤

- 分析最优（子）结构



问题结构分析

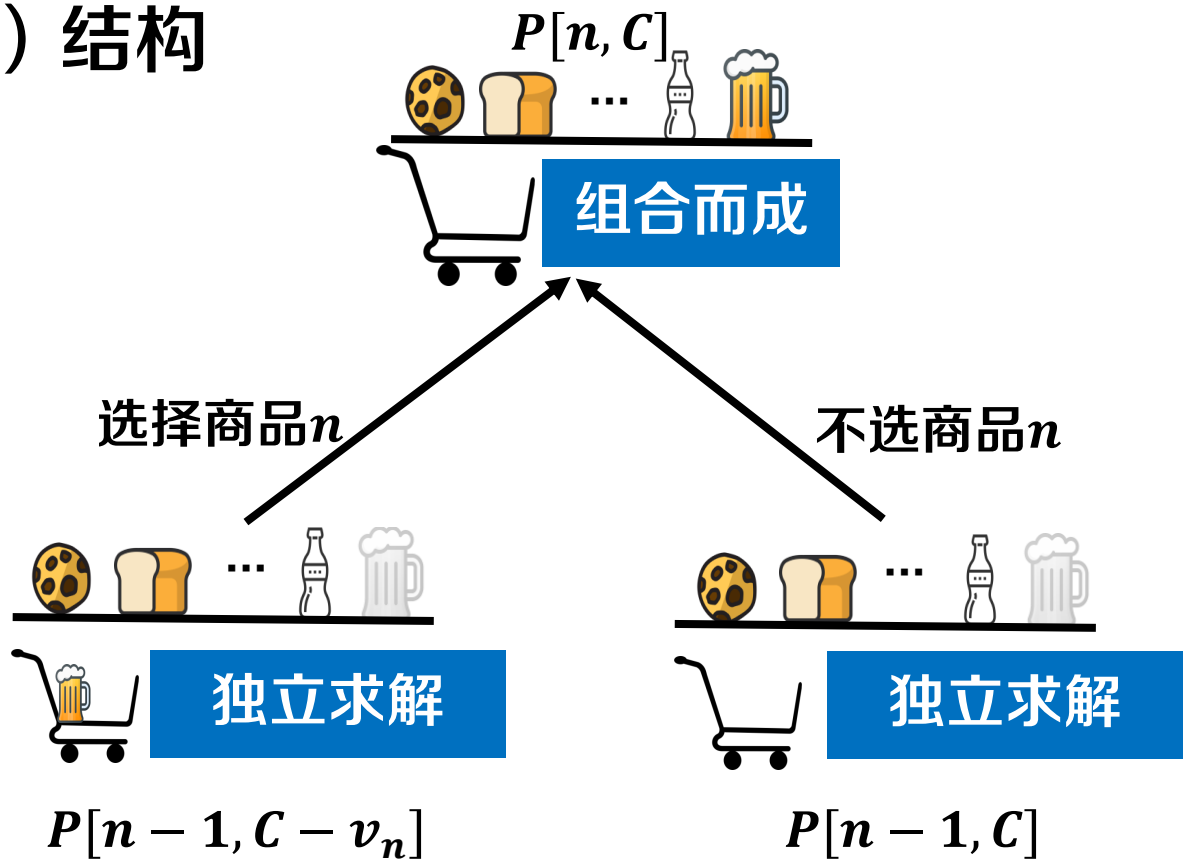
递推关系建立

自底向上计算

最优方案追踪

动态规划：一般步骤

- 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

动态规划：一般步骤

- 分析最优（子）结构

最优子结构性质（Optimal Substructure）

问题的最优解由相关子问题最优解**组合而成**

子问题可以**独立求解**

动态规划：一般步骤

- 分析最优（子）结构

最优子结构性质（Optimal Substructure）

问题的最优解由相关子问题最优解**组合而成**

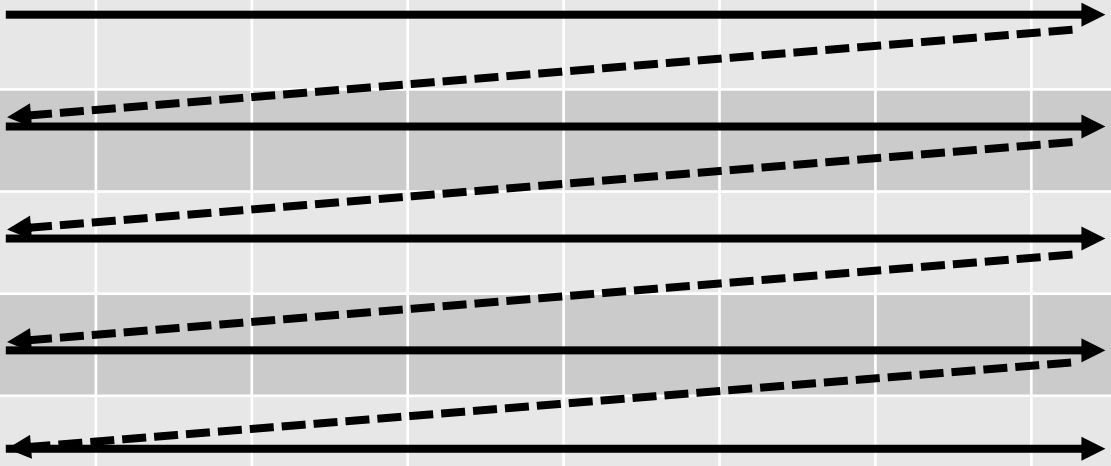
子问题可以**独立求解**

- 构造递推公式
  - $P[i, c] = \max\{P[i-1, c],\ P[i-1, c-v_i] + p_i\}$

动态规划：一般步骤

- 确定计算顺序
  - 初始化
    - 容量为0时：$P[i,0]=0$　　没有商品时：$P[0,c]=0$
  - $P[i,c]$依赖于子问题$P[i-1,c-v_i]$和$P[i-1,c]$
- 依次求解问题

$P[i,c]$	$c=0$	1	2	3	…	10	11	12	13
$i=0$	0	0	0	0	…	0	0	0	0
1	0								
2	0								
3	0								
4	0								
5	0								



动态规划：一般步骤

- 记录决策过程
  - $Rec[i,c]=\begin{cases}1, & \text{选择商品}\\ 0, & \text{不选商品}\end{cases}$

P	$c=0$	1	2	…	9	10	11	12	13
$i=1$									
2									
3						$P[i,c]$			
4									
5									

问题结构分析

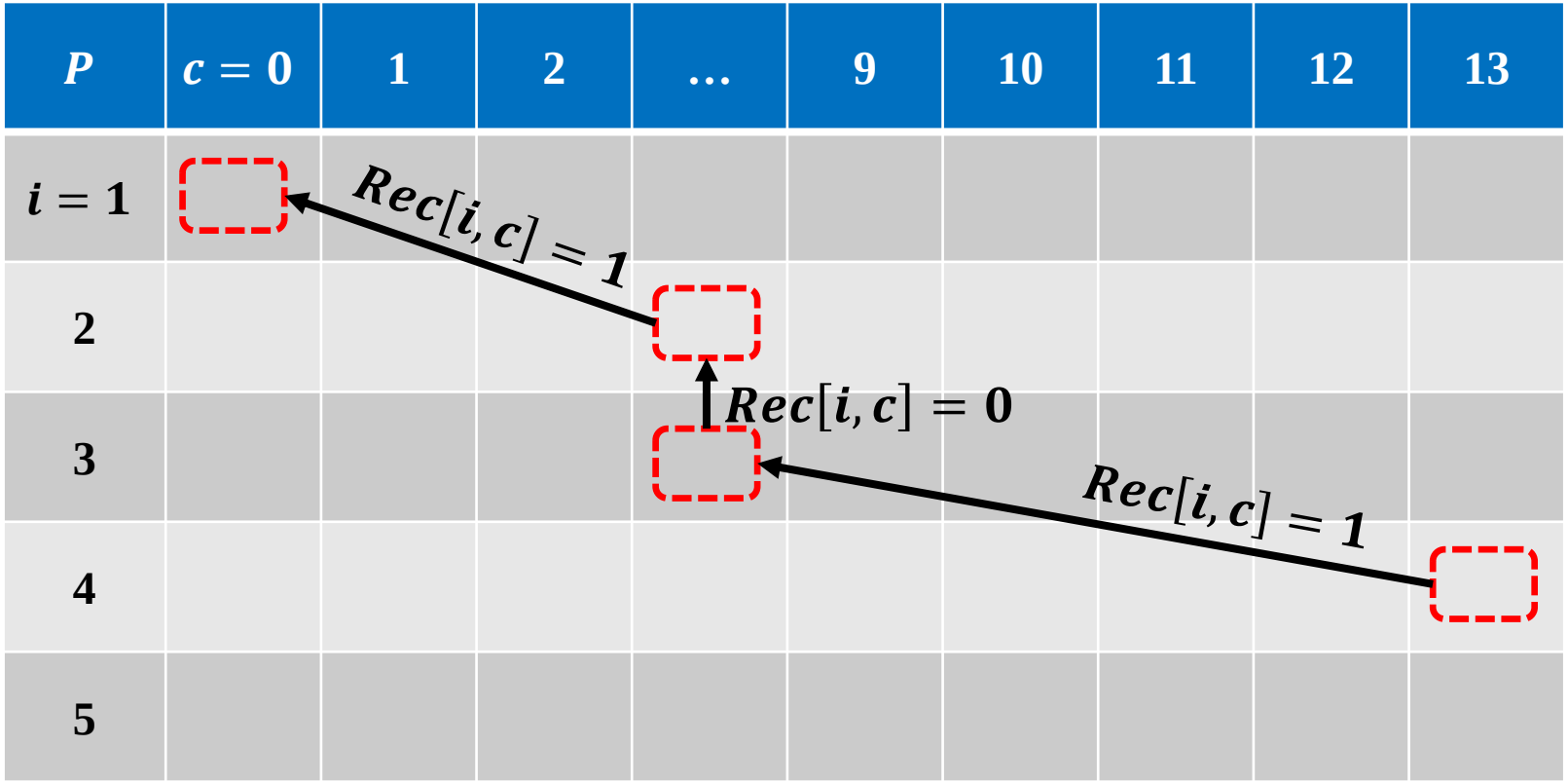
递推关系建立

自底向上计算

最优方案追踪

动态规划：一般步骤

- 输出最优方案
  - 倒序判断是否选择商品
  - $Rec[i,c]=1$时，选择商品i，考察子问题$P[i-1,c-v_i]$
  - $Rec[i,c]=0$时，不选商品i，考察子问题$P[i-1,c]$



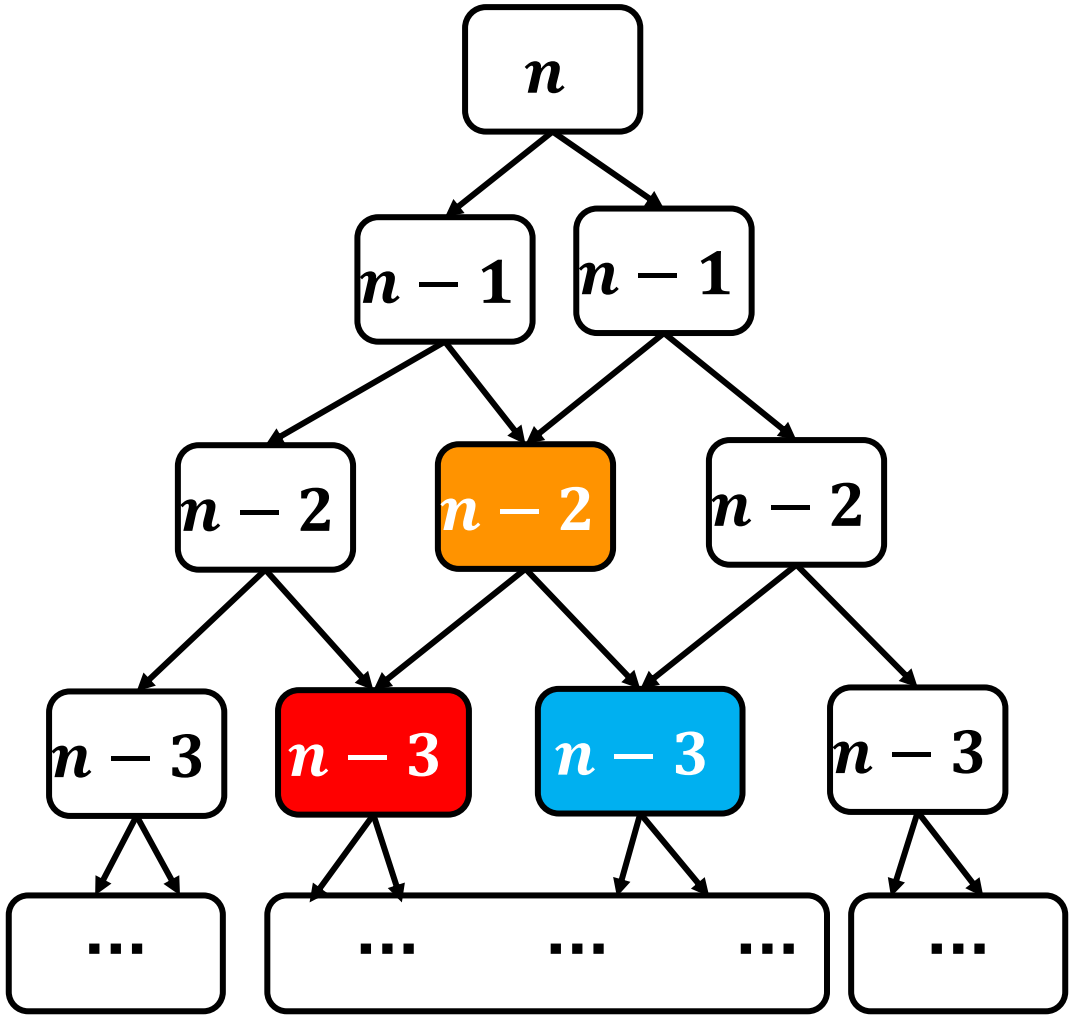
问题结构分析

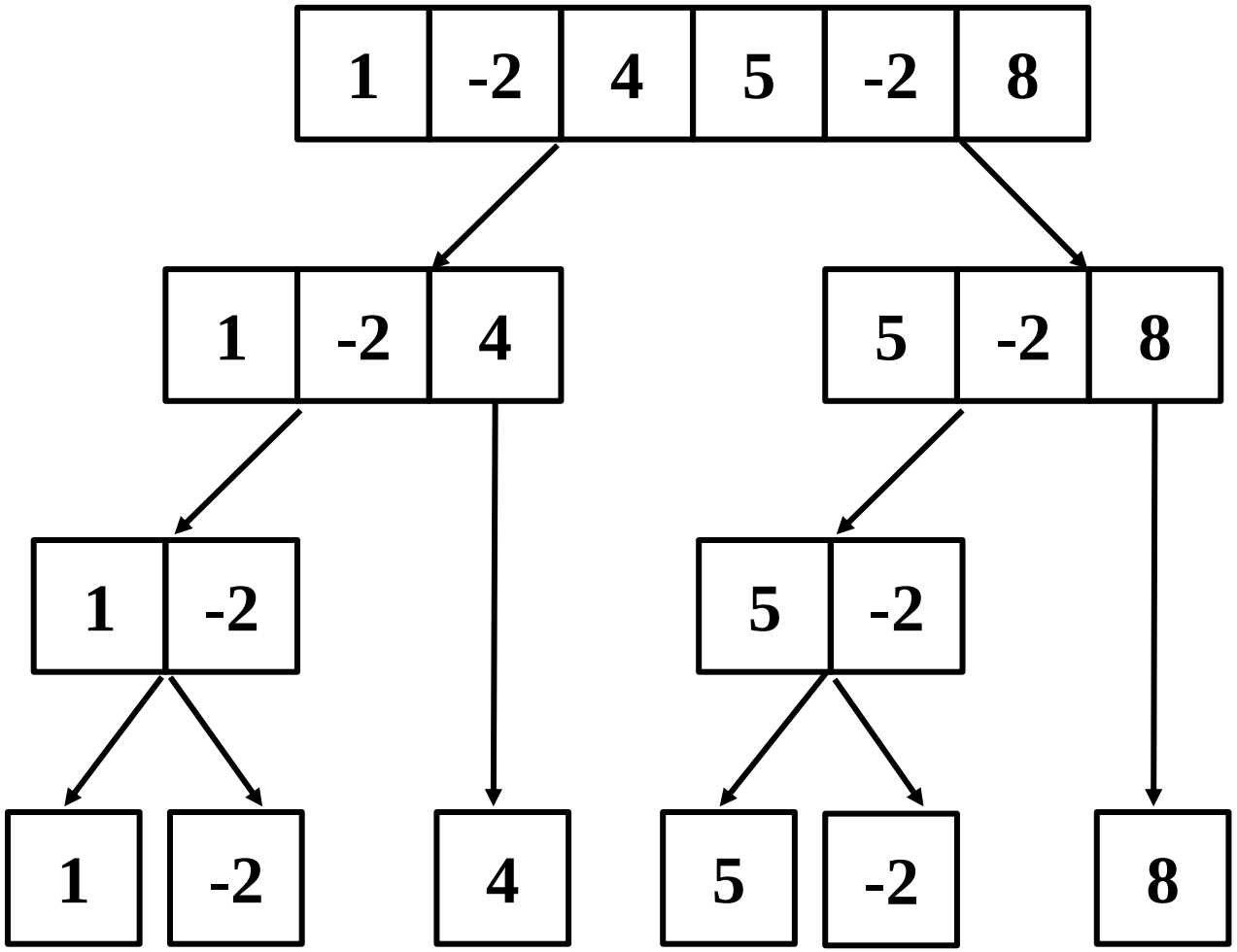
递推关系建立

自底向上计算

最优方案追踪

方法比较



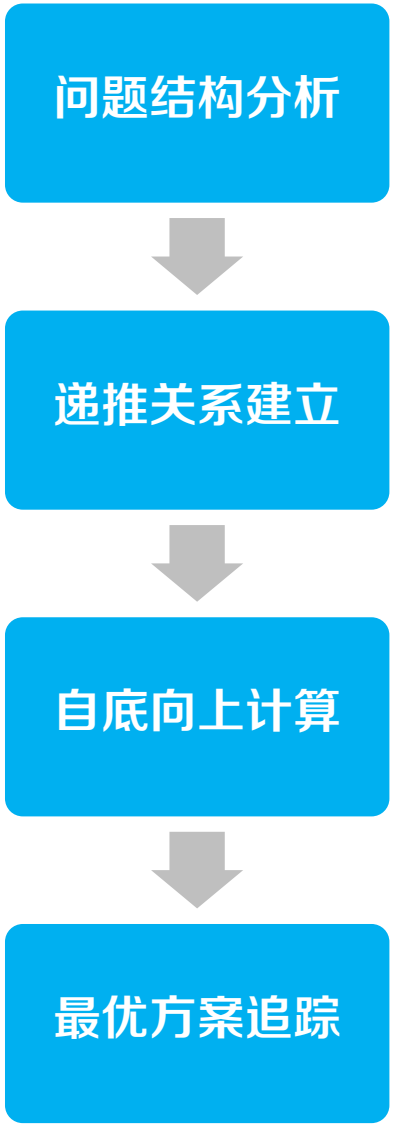


动态规划：重叠子问题

分而治之：独立子问题

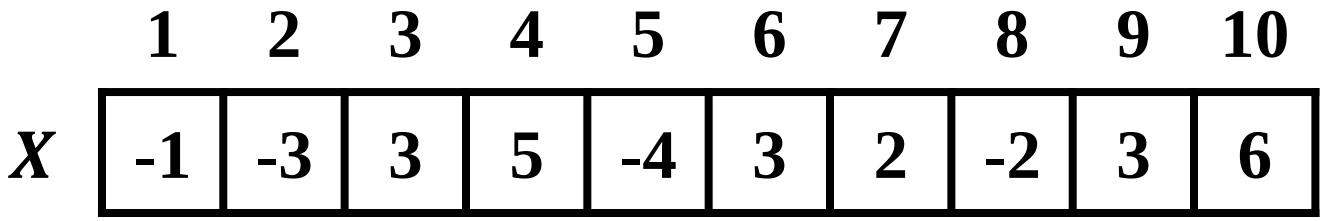
动态规划小结

- 如何设计一个动态规划算法？四个步骤
  - 问题结构分析
    - 给出问题表示，明确原始问题
  - 递推关系建立
    - 分析最优结构，构造递推公式
  - 自底向上计算
    - 确定计算顺序，依次求解问题
  - 最优方案追踪
    - 记录决策过程，输出最优方案



动态规划篇：最大子数组问题II

最大子数组问题

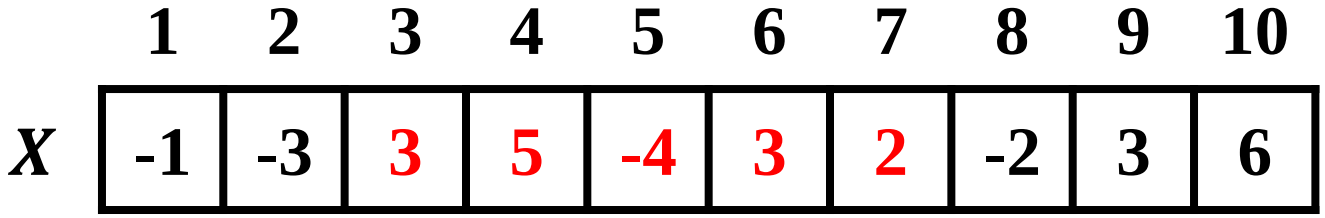


最大子数组问题

	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

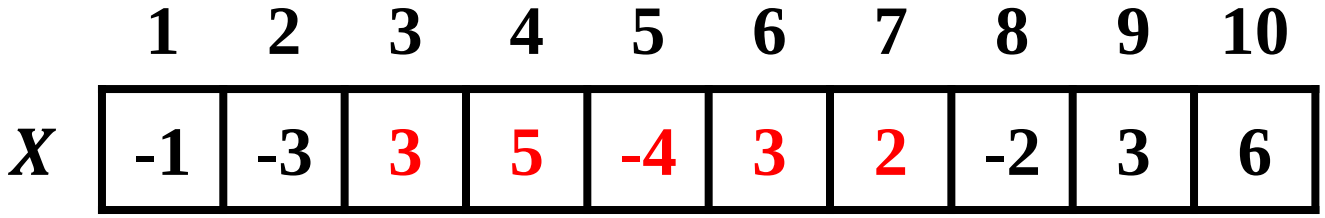
- 子数组为数组X中连续的一段序列

最大子数组问题



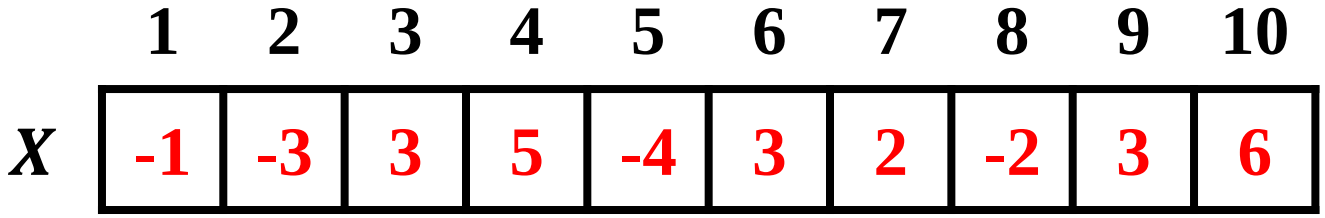
- 子数组$X[3..7]$

最大子数组问题



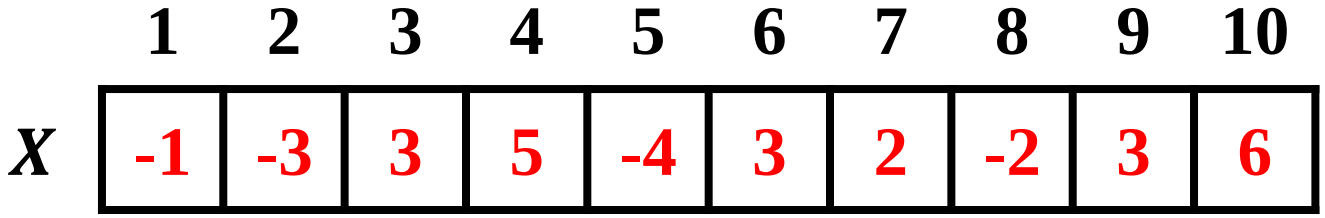
- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$

最大子数组问题



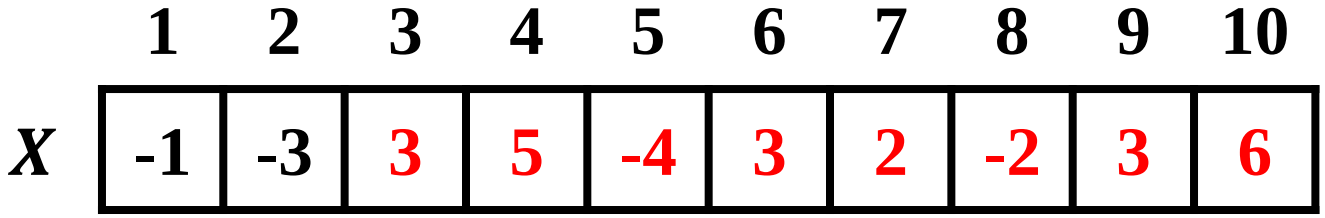
- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$

最大子数组问题



- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$
  - 求和为：$-1-3+3+5-4+3+2-2+3+6=12$

最大子数组问题



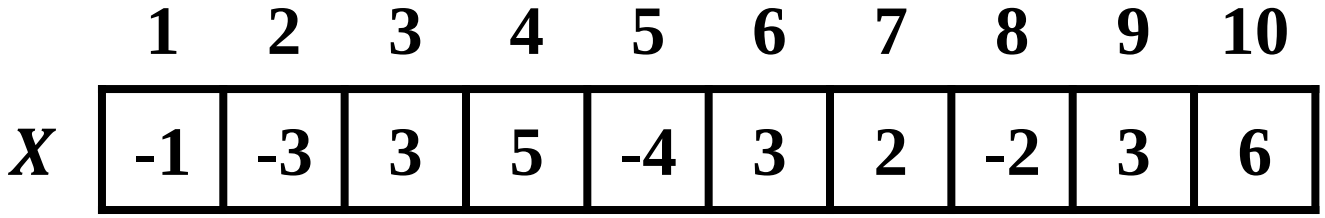
- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$
  - 求和为：$-1-3+3+5-4+3+2-2+3+6=12$
- 子数组$X[3..10]$

最大子数组问题

	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$
  - 求和为：$-1-3+3+5-4+3+2-2+3+6=12$
- 子数组$X[3..10]$
  - 求和为：$3+5-4+3+2-2+3+6=16$

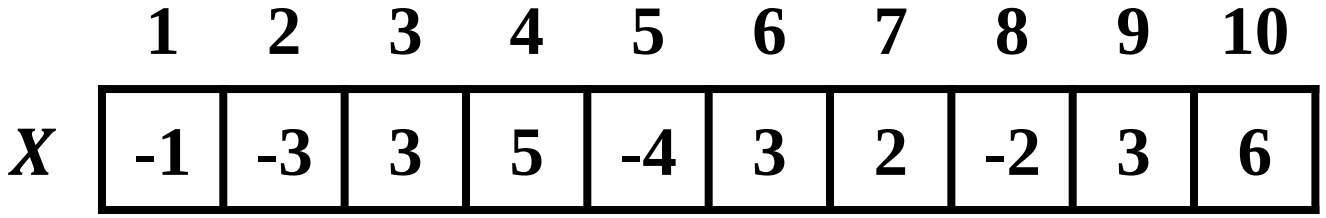
最大子数组问题



- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$
  - 求和为：$-1-3+3+5-4+3+2-2+3+6=12$
- 子数组$X[3..10]$
  - 求和为：$3+5-4+3+2-2+3+6=16$

问题：寻找数组X最大的非空子数组？

最大子数组问题



- 子数组$X[3..7]$
  - 求和为：$3+5-4+3+2=9$
- 子数组$X[1..10]$
  - 求和为：$-1-3+3+5-4+3+2-2+3+6=12$
- 子数组$X[3..10]$
  - 求和为：$3+5-4+3+2-2+3+6=16$

问题：寻找数组X最大的非空子数组？
答案：$X[3..10]=16$

问题定义

- 形式化定义

最大子数组问题

Max Continuous Subarray, MCS

输入

- 给定一个数组$X[1..n]$，对于任意一对数组下标为$l, r\ (l \le r)$的非空子数组，其和记为

$$S(l,r) = \sum_{i=l}^{r} X[i]$$

输出

- 求出$S(l,r)$的最大值，记为S_{max}

蛮力枚举

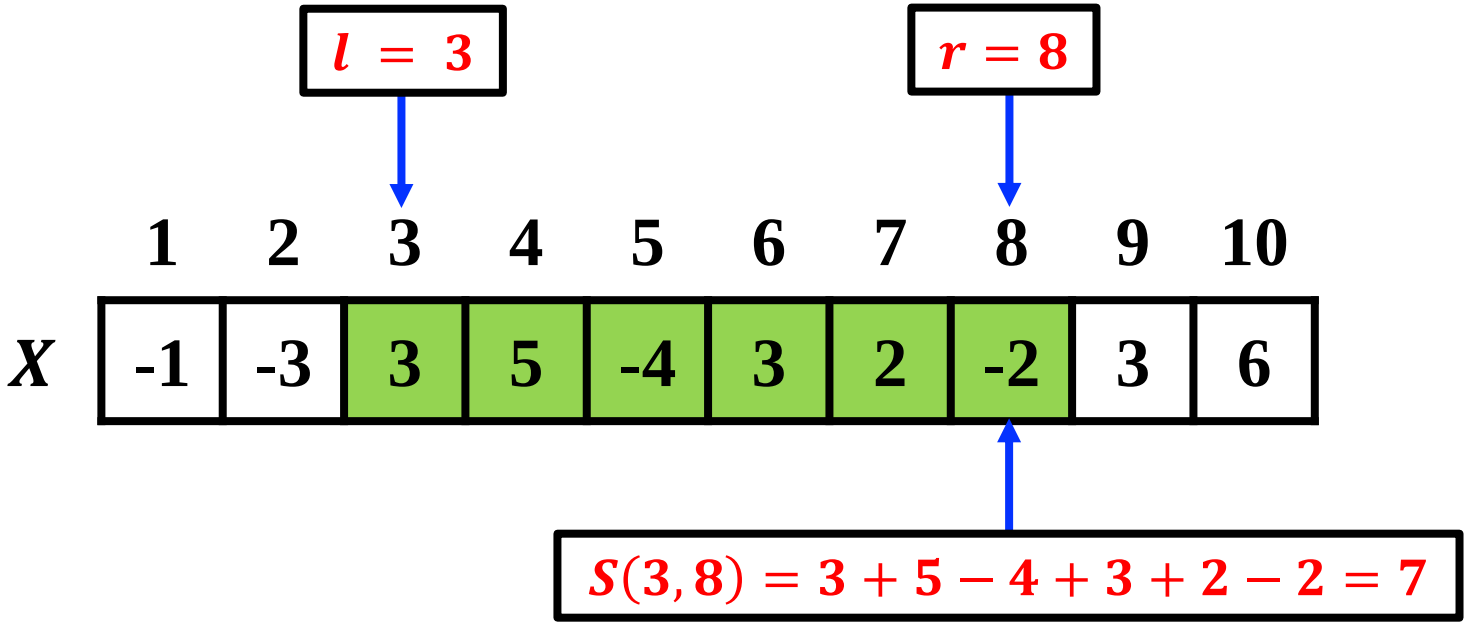
	1	2	3	4	5	6	7	8	9	10
X	-1	-3	3	5	-4	3	2	-2	3	6

- 数组$X[1..n]$，其所有的下标$l, r(l \leq r)$组合分为以下两种情况
  - 当$l = r$时，一共$C_n^1 = n$种组合
  - 当$l < r$时，一共C_n^2种组合

- 枚举$n + C_n^2$种下标l, r组合，求出最大子数组之和

蛮力枚举

- $l = 3,\ r = 8$
- 计算$S(3,8)$：



蛮力枚举

- 枚举$n + C_n^2$种可能的区间$[l, r](l \leq r)$，求解最大子数组之和S_{max}

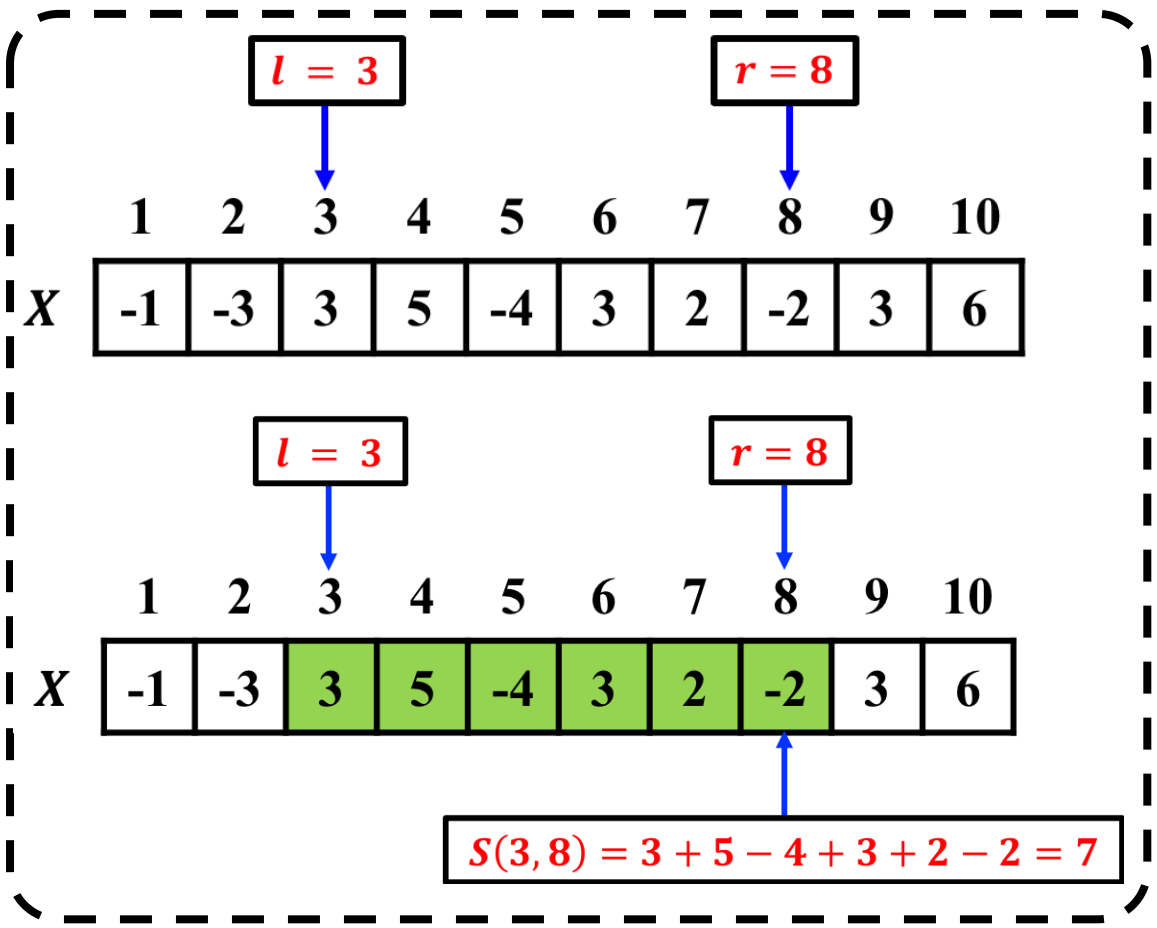
输入: 数组$X[1..n]$
输出: 最大子数组之和S_{max}

```
S_max ← −∞
for l ← 1 to n do
    for r ← l to n do
        S(l, r) ← 0
        for i ← l to r do
            S(l, r) ← S(l, r) + X[i]
        end
        S_max ← max{S_max, S(l, r)}
    end
end
return S_max
```

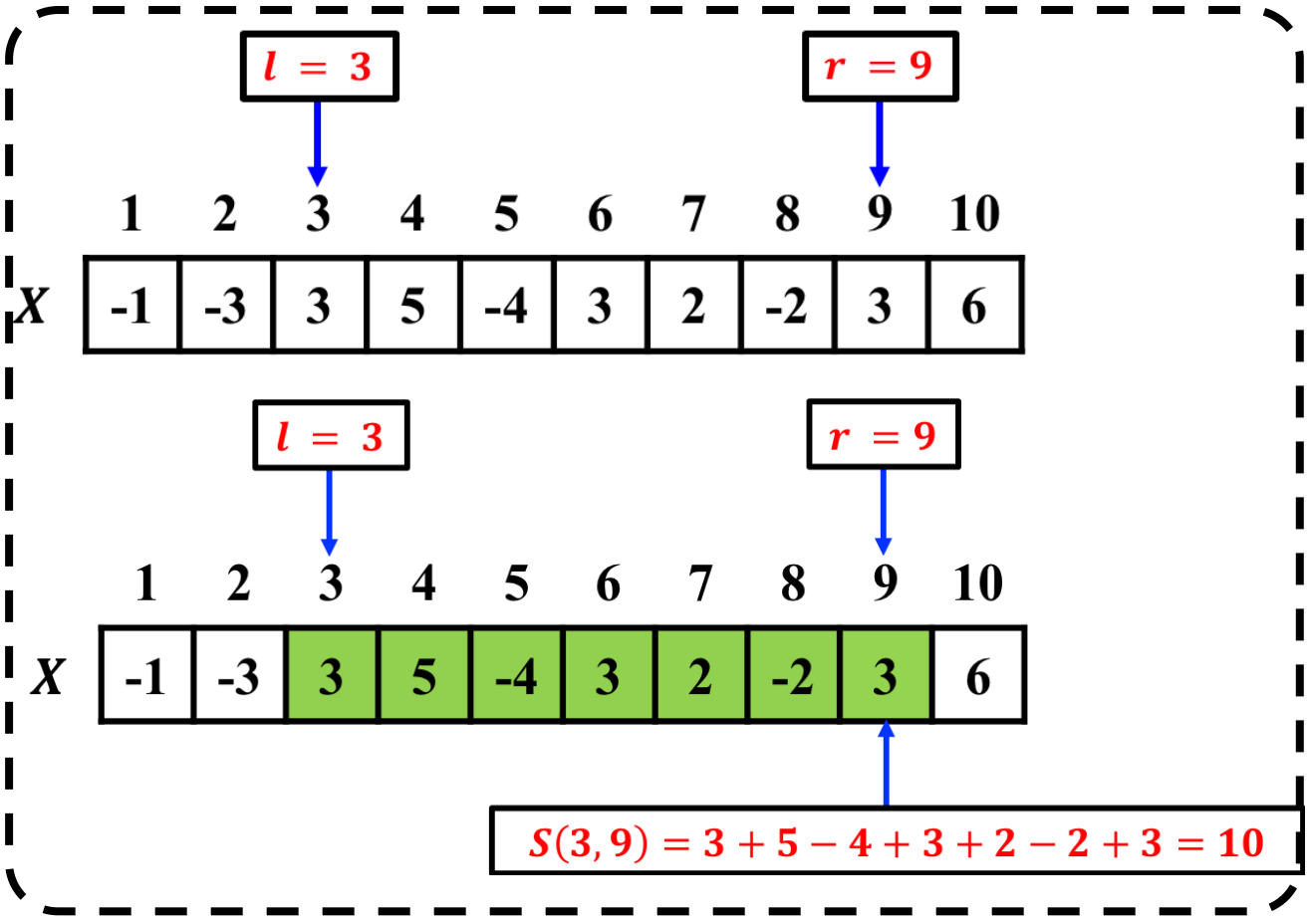
$O(n)$ $O(n^2)$ $O(n^3)$

时间复杂度：$O(n^3)$

从蛮力枚举到优化枚举

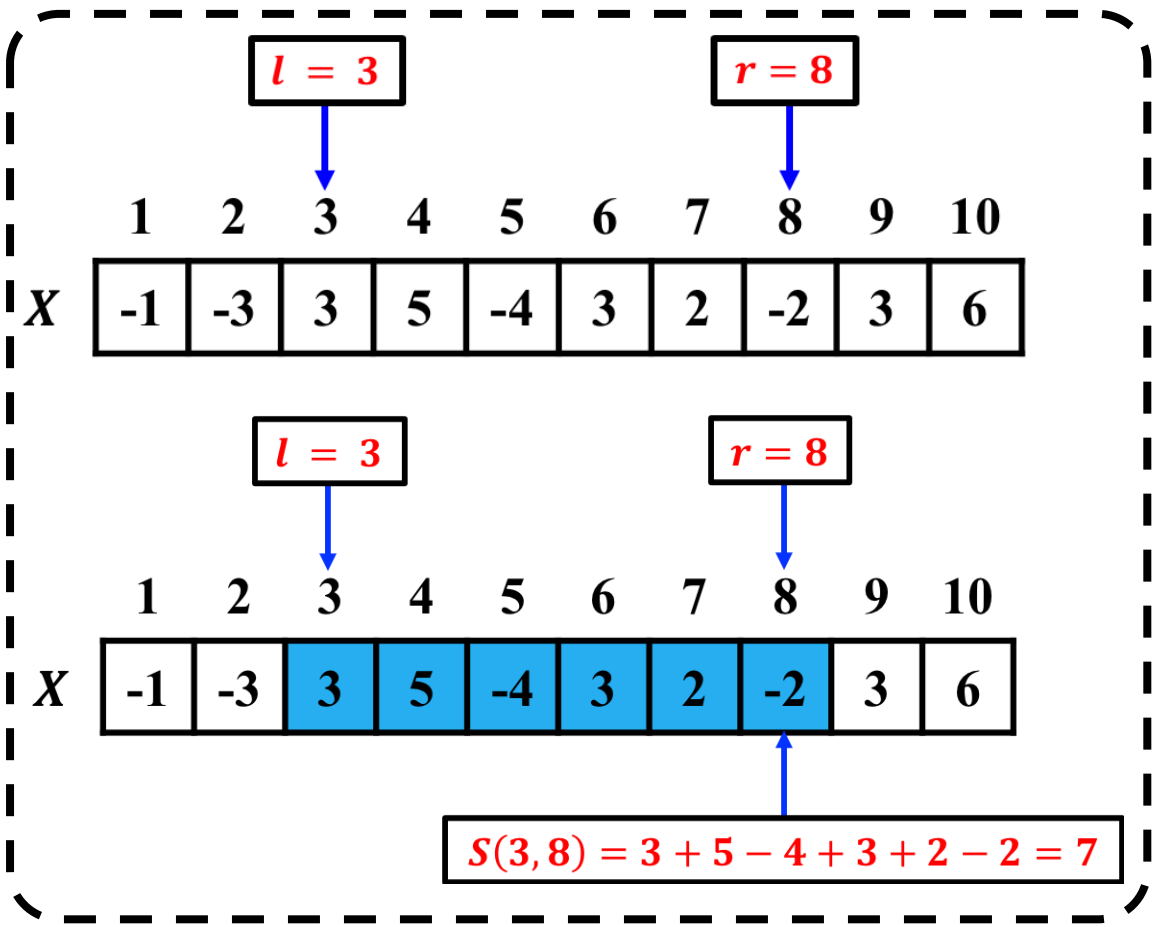


计算$S(3,8)$循环6次

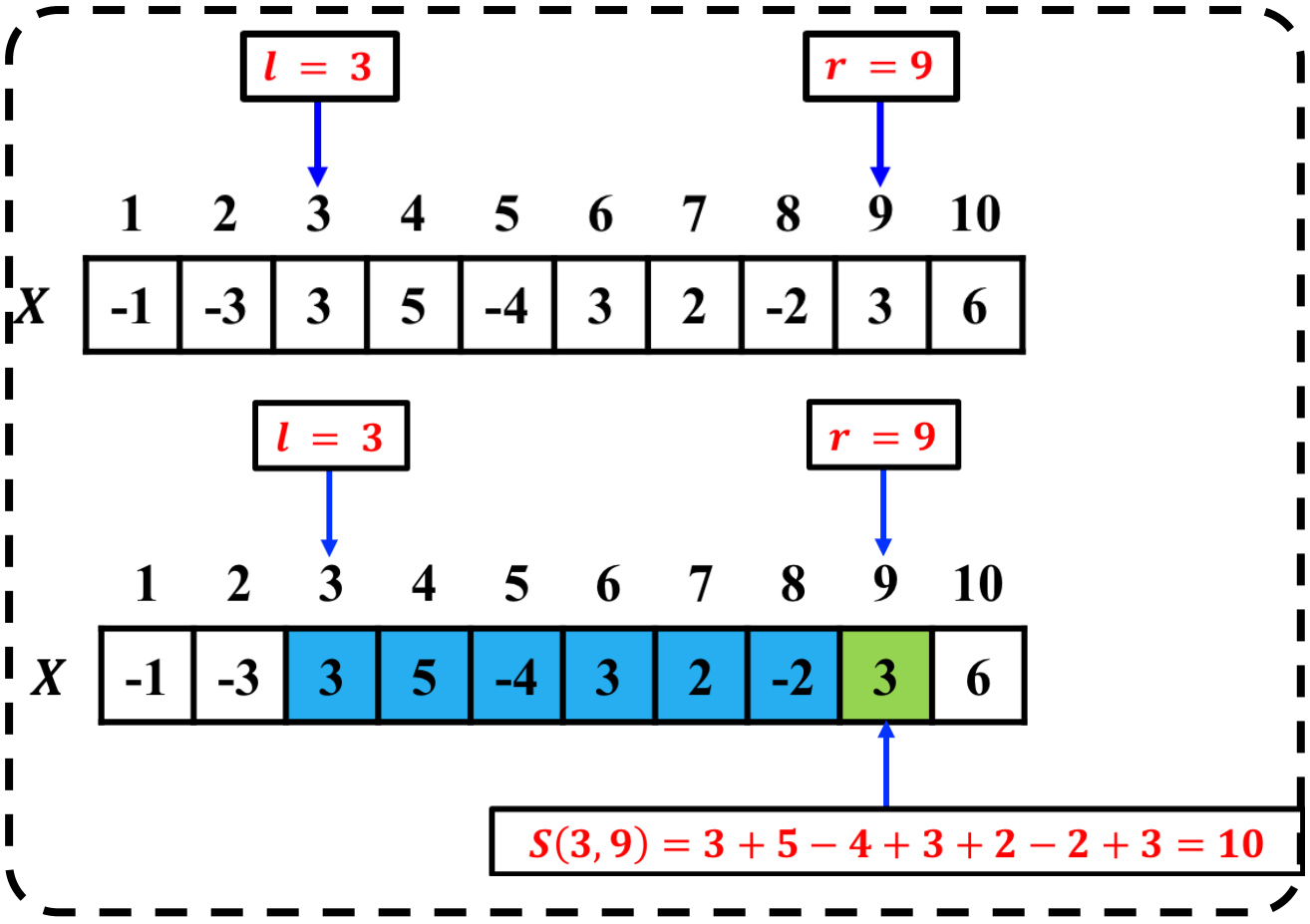


计算$S(3,9)$循环7次

从蛮力枚举到优化枚举



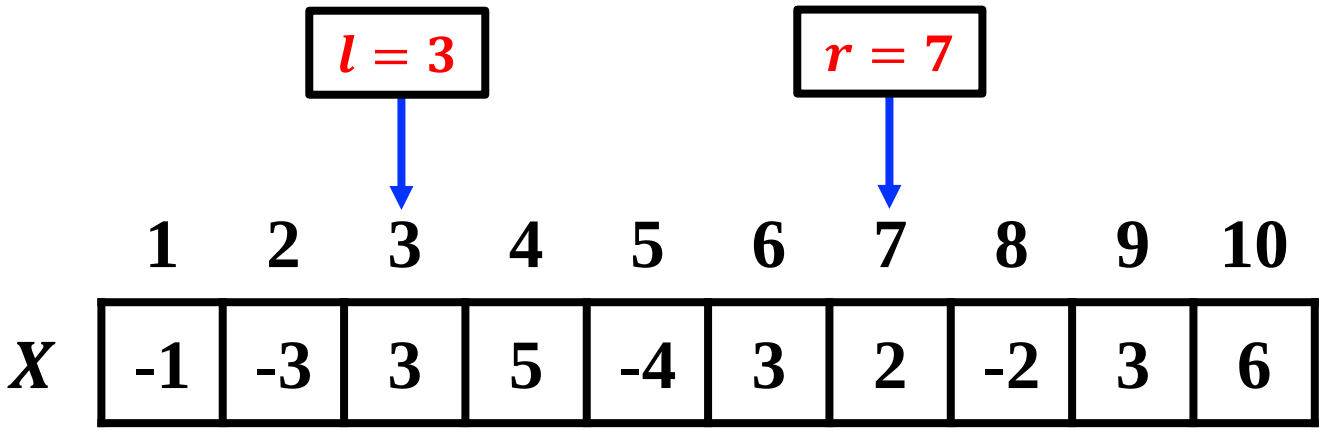
计算$S(3,8)$循环6次



计算$S(3,9)$循环7次

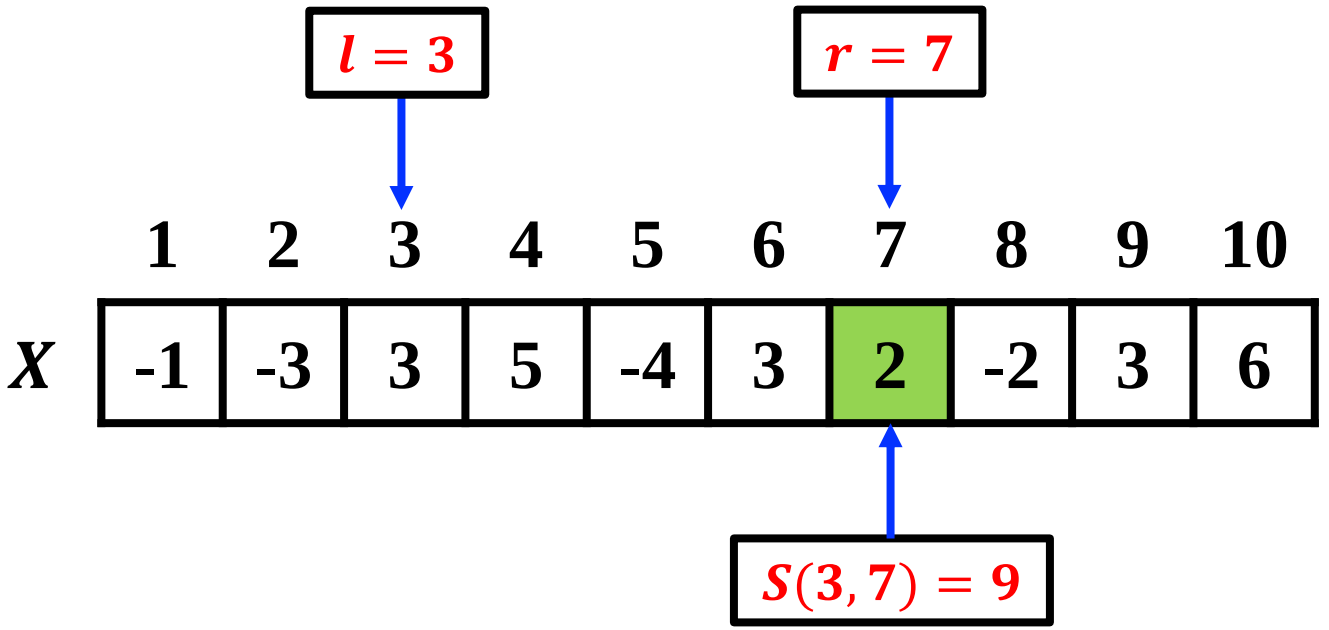
优化枚举

- $l = 3,\ r = 7$



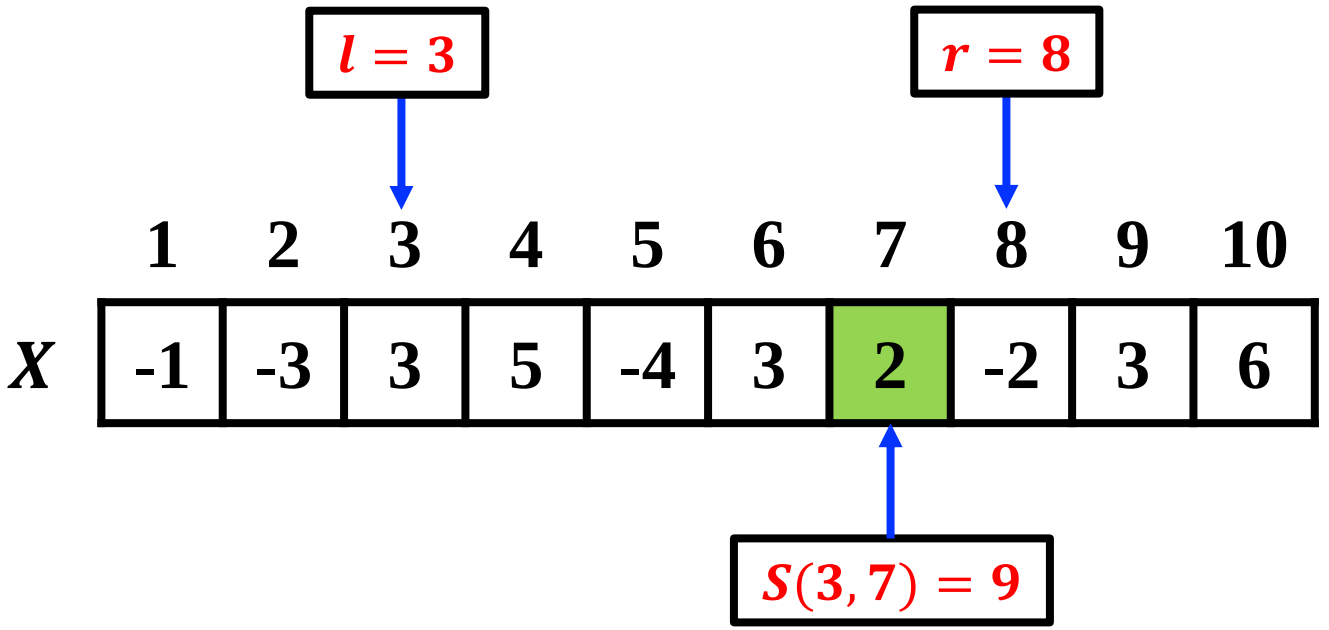
优化枚举

- $l = 3,\ r = 7,\ S(3,7) = 9$



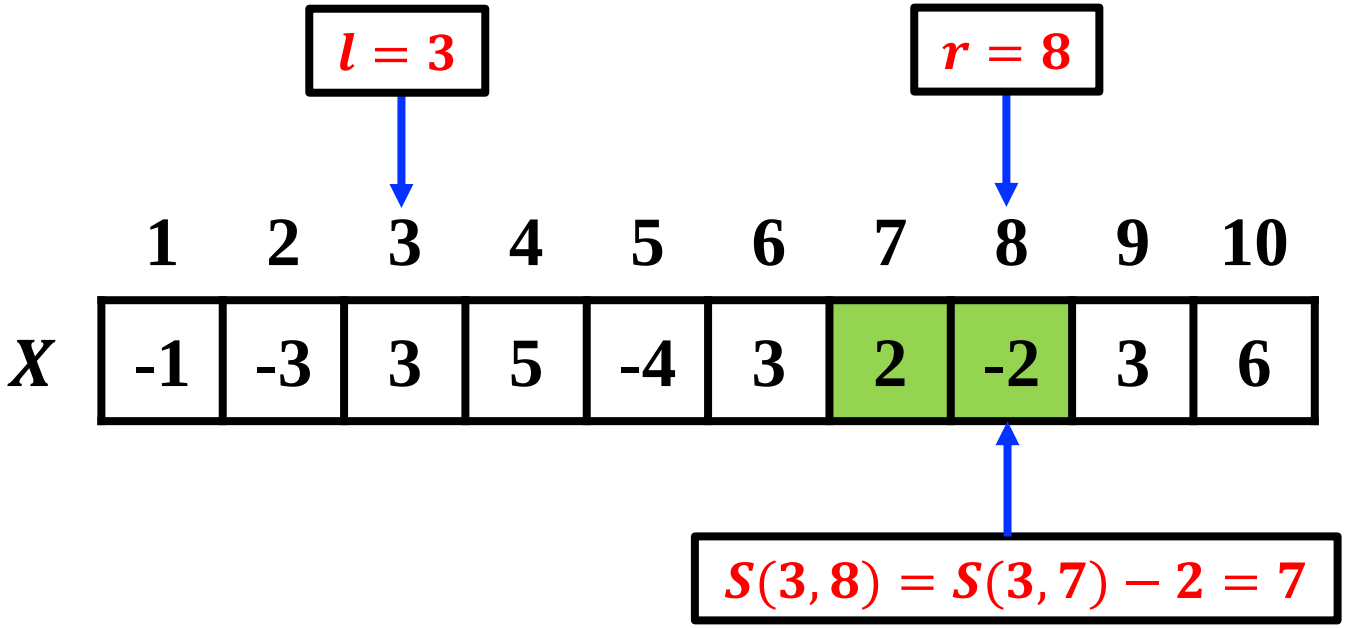
优化枚举

- $l = 3,\ r = 8,\ S(3,8) = ?$



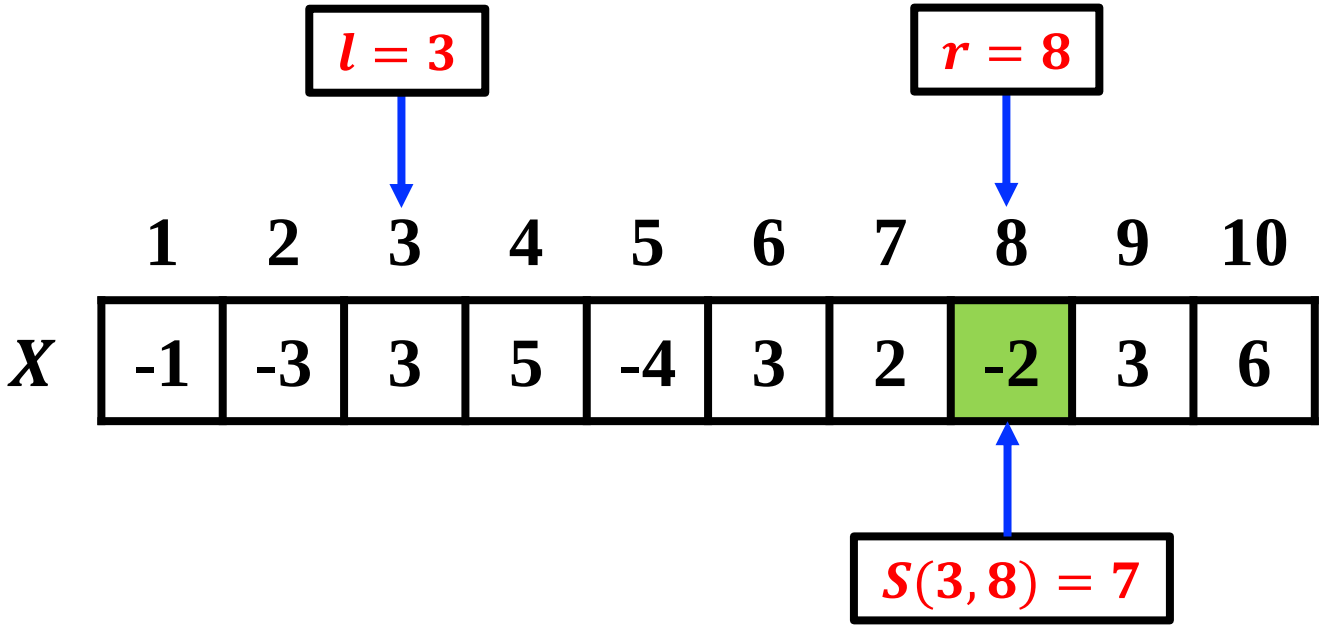
优化枚举

- $l = 3,\ r = 8,\ S(3,8) = 7$



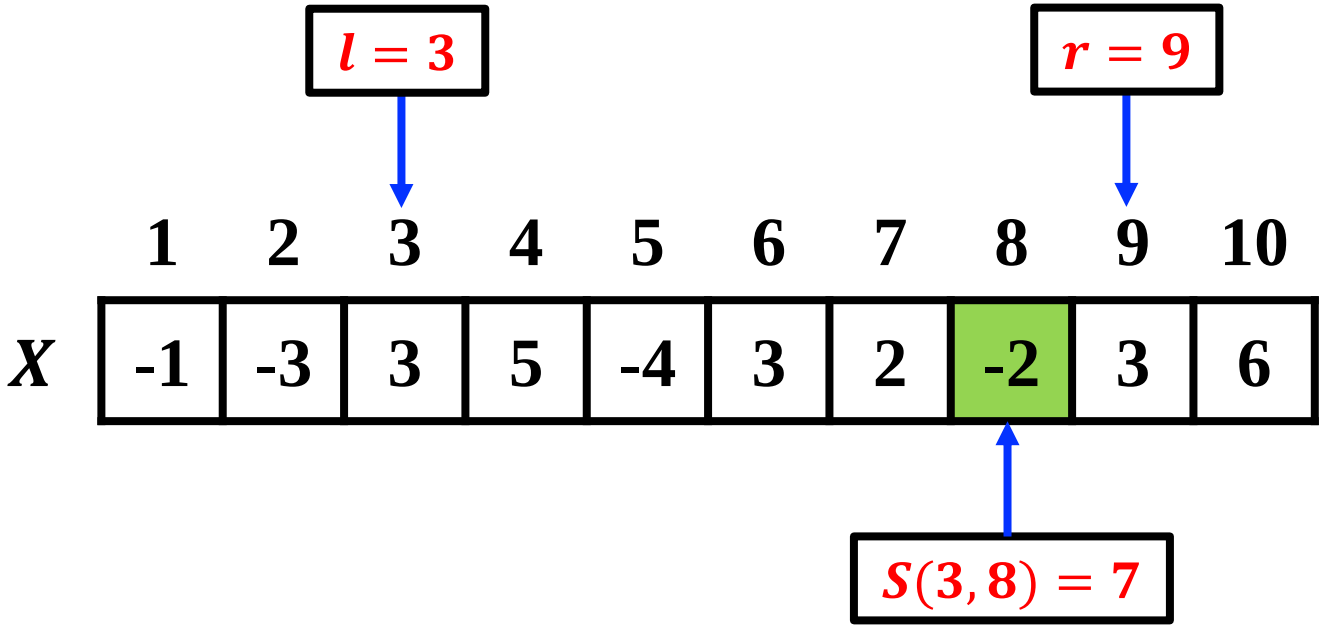
优化枚举

- $l = 3,\ r = 8,\ S(3,8) = 7$



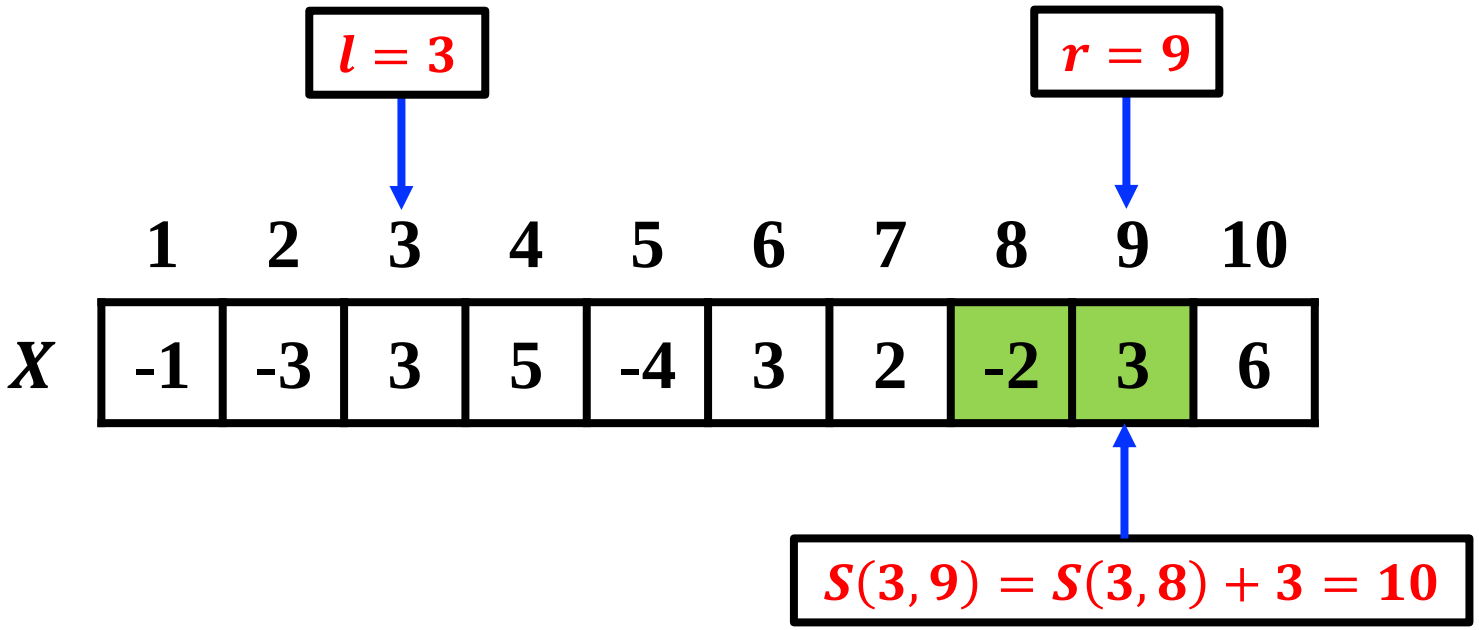
优化枚举

- $l = 3,\ r = 9,\ S(3,9) = ?$



优化枚举

- $l = 3,\ r = 9,\ S(3,9) = ?$



优化枚举

- 核心思想: $S(l,r)=\sum_{i=l}^{r}X[i]=\ S(l,r-1)+X[r]$

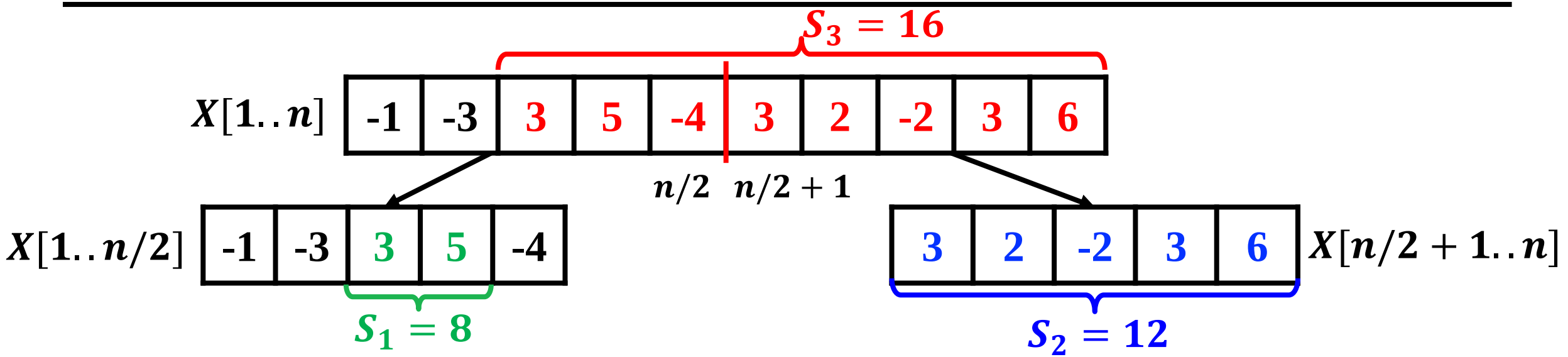
输入: 数组$X[1..n]$
输出: 最大子数组之和S_{max}

```
S_max ← −∞
for l ← 1 to n do
    S ← 0
    for r ← l to n do
        S ← S + X[r]
        S_max ← max{S_max, S}
    end
end
return S_max
```

$O(n)$ $O(n^2)$

时间复杂度: $O(n^2)$

分而治之



- 将数组$X[1..n]$分为$X[1..n/2]$和$X[n/2+1..n]$
- 递归求解子问题
  - S_1：数组$X[1..n/2]$ 的最大子数组
  - S_2：数组$X[n/2+1..n]$ 的最大子数组
- 合并子问题，得到S_{max}
  - S_3：跨中点的最大子数组
  - 数组X的最大子数组之和$S_{max} = max\{S_1, S_2, S_3\}$

分解原问题

解决子问题

合并问题解

分而治之

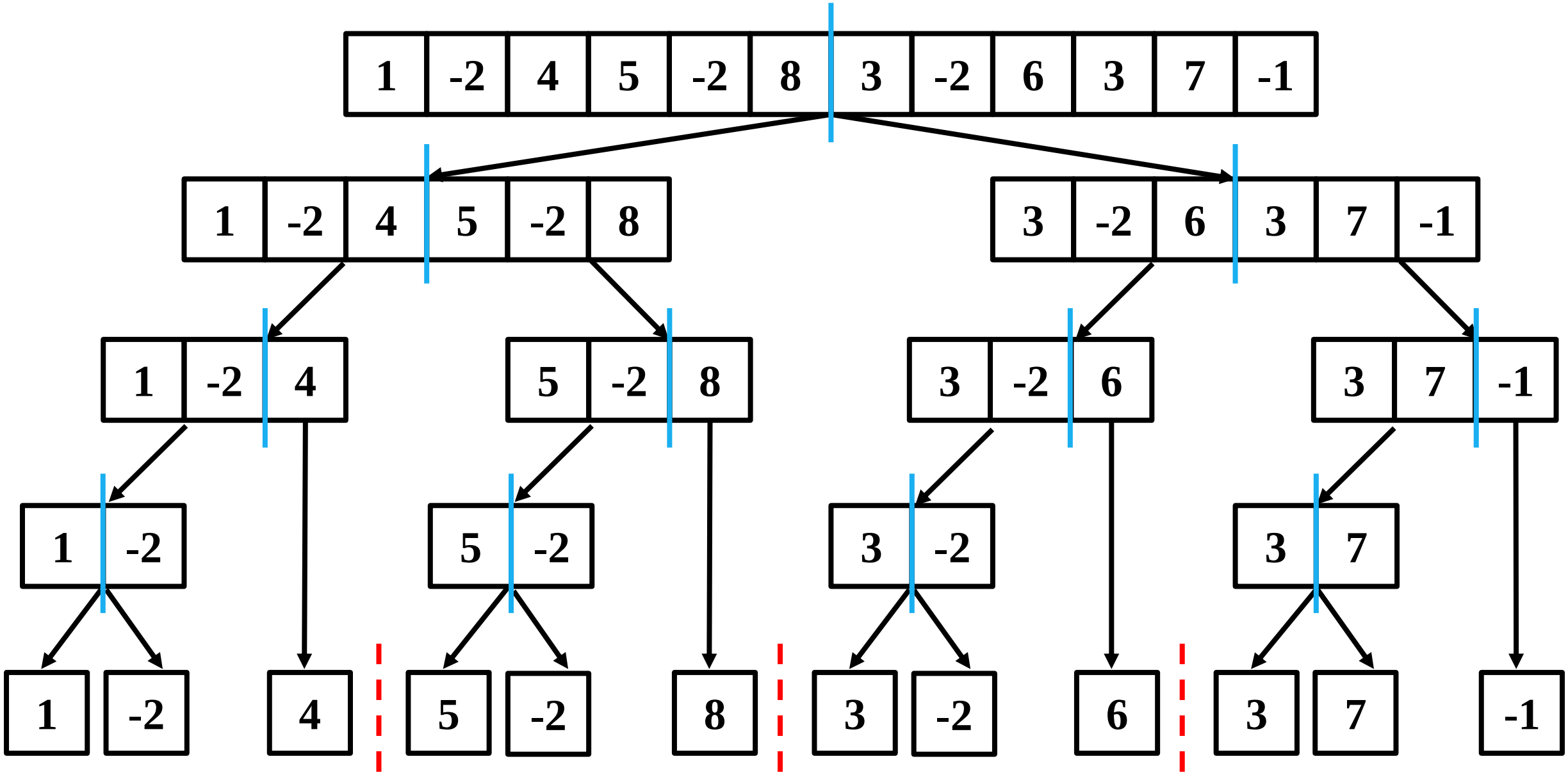
- 时间复杂度分析

输入: 数组X, 数组下标$low, high$
输出: 最大子数组之和S_{max}

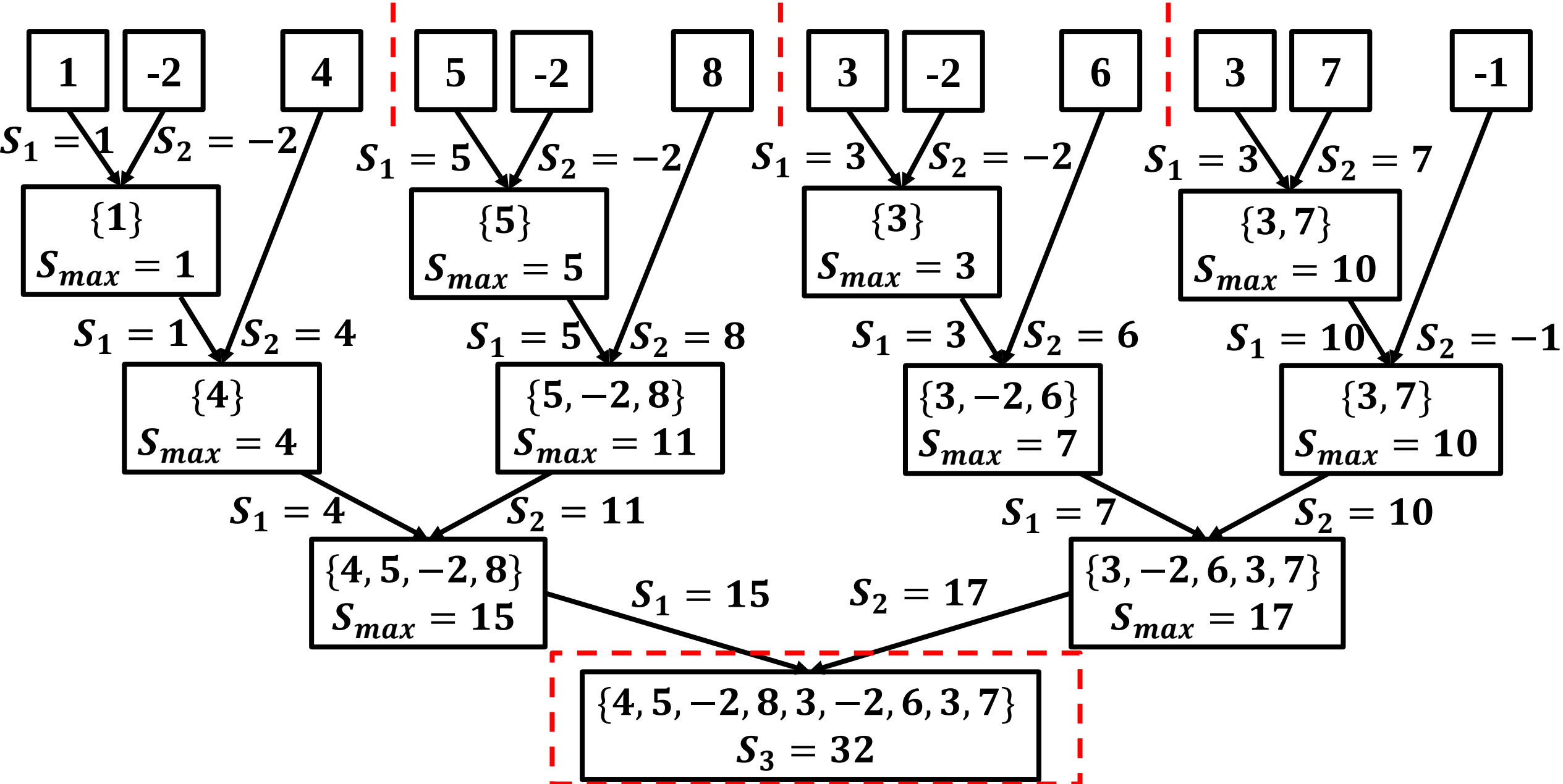
```
if low = high then
    return X[low]
end
else
    mid ← ⌊(low+high)/2⌋
    S1 ← MaxSubArray(X,low,mid)
    S2 ← MaxSubArray(X,mid+1,high)
    S3 ← CrossingSubArray(X,low,mid,high)
    Smax ← max{S1, S2, S3}
    return Smax
end
```

时间复杂度：$O(n \log n)$

分而治之


1 -2 4 5 -2 8 3 -2 6 3 7 -1
1 -2 4 5 -2 8
3 -2 6 3 7 -1
1 -2 4
5 -2 8
3 -2 6
3 7 -1
1 -2
5 -2
3 -2
3 7
1
-2
4
5
-2
8
3
-2
6
3
7
-1

算法实例


1
-2
4
5
-2
8
3
-2
6
3
7
-1
$S_1 = 1$ $S_2 = -2$
$S_1 = 5$ $S_2 = -2$
$S_1 = 3$ $S_2 = -2$
$S_1 = 3$ $S_2 = 7$
{1}
$S_{max} = 1$
{5}
$S_{max} = 5$
{3}
$S_{max} = 3$
{3, 7}
$S_{max} = 10$
$S_1 = 1$ $S_2 = 4$
$S_1 = 5$ $S_2 = 8$
$S_1 = 3$ $S_2 = 6$
$S_1 = 10$ $S_2 = -1$
{4}
$S_{max} = 4$
{5, −2, 8}
$S_{max} = 11$
{3, −2, 6}
$S_{max} = 7$
{3, 7}
$S_{max} = 10$
$S_1 = 4$ $S_2 = 11$
$S_1 = 7$ $S_2 = 10$
{4, 5, −2, 8}
$S_{max} = 15$
{3, −2, 6, 3, 7}
$S_{max} = 17$
$S_1 = 15$ $S_2 = 17$
{4, 5, −2, 8, 3, −2, 6, 3, 7}
$S_3 = 32$

算法比较

算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n\log n)$
?	$O(n)$

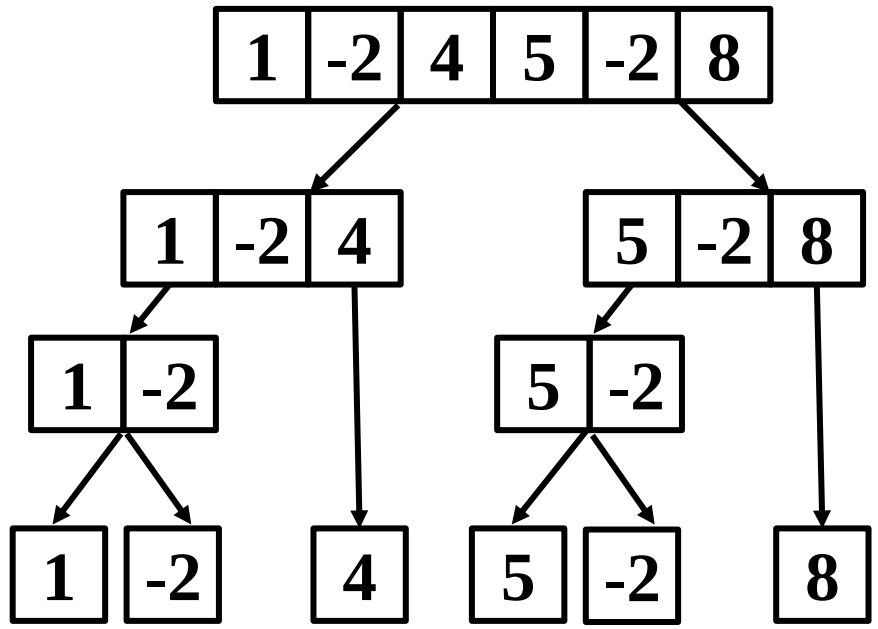
问题：是否可以设计一个时间复杂度为$O(n)$的算法?

算法比较

算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n\log n)$
动态规划	$O(n)$

算法比较

算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n\log n)$
动态规划	$O(n)$

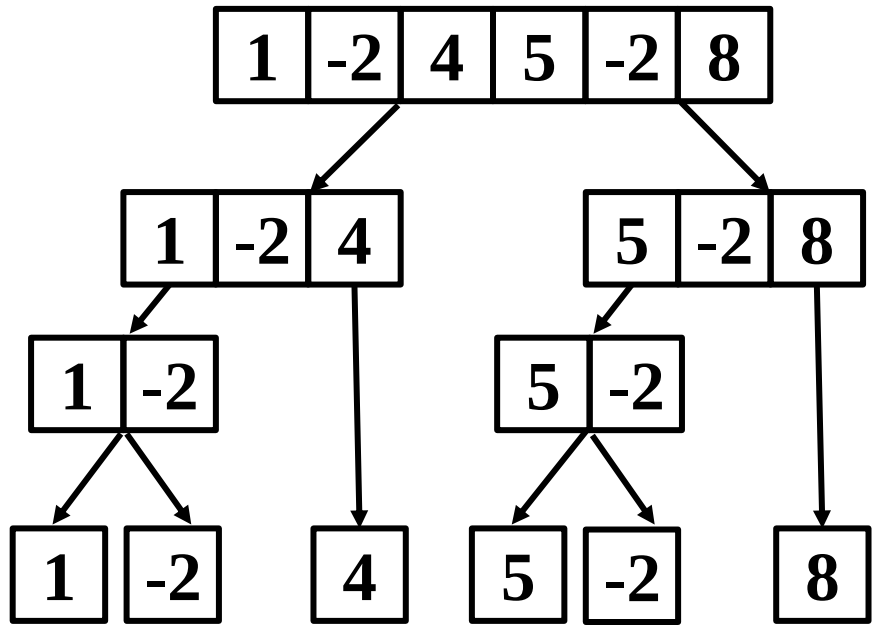


子问题相互独立

算法比较

基于枚举

算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n\log n)$
动态规划	$O(n)$



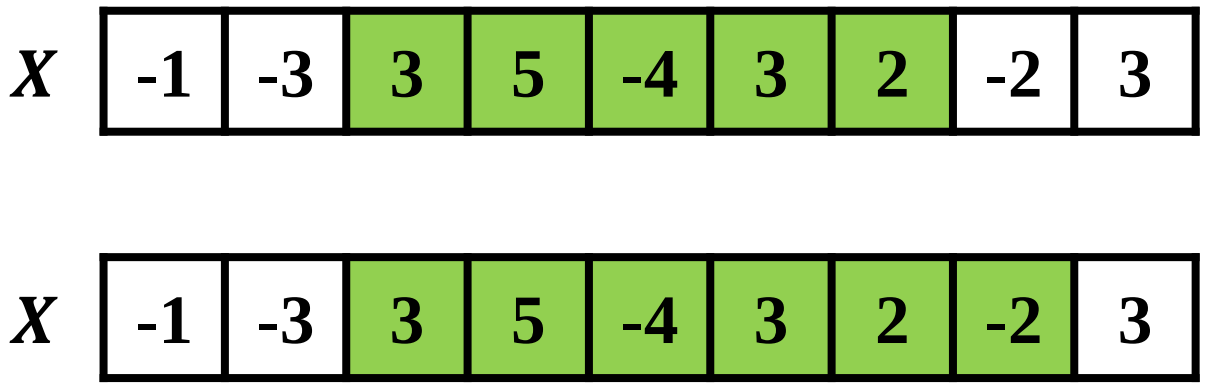
子问题相互独立

算法比较

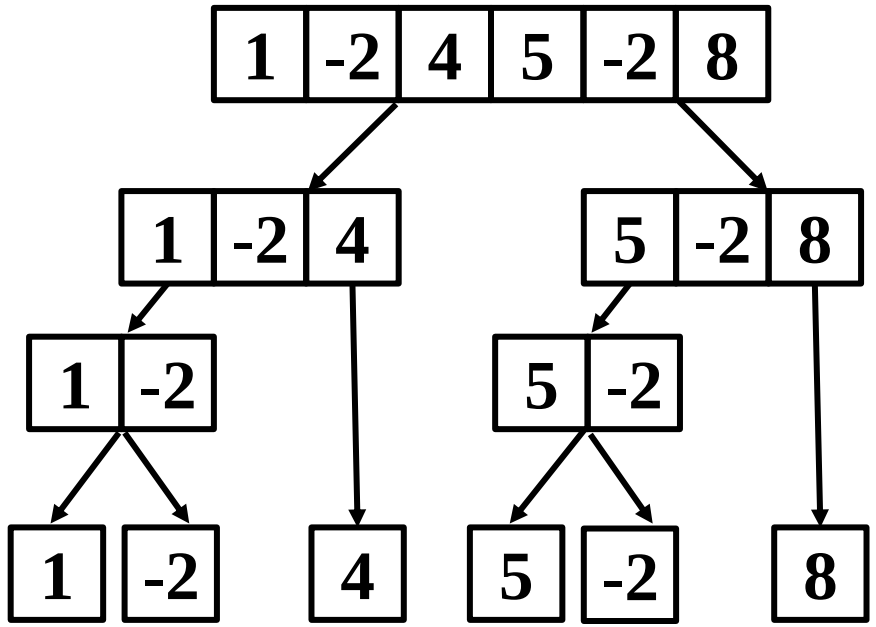
基于枚举

算法名称	时间复杂度
蛮力枚举	$O(n^3)$
优化枚举	$O(n^2)$
分而治之	$O(n\log n)$
动态规划	$O(n)$

- 枚举区间



存在重叠子问题



子问题相互独立

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值$=4$, $S(i,j)=4$

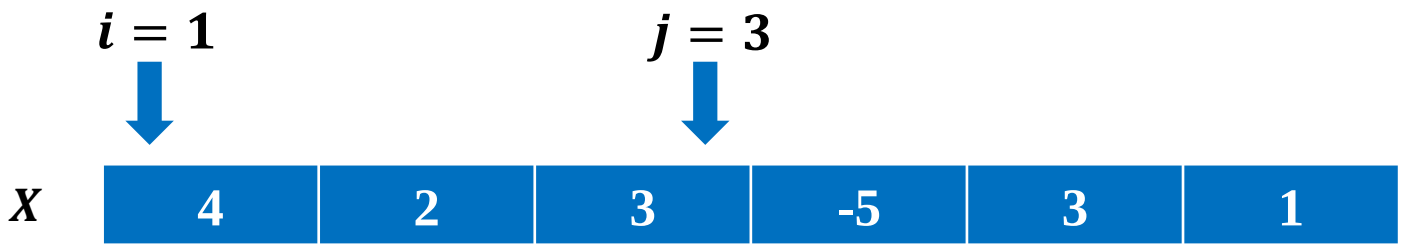
枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值$= 6, S(i, j) = 6$

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾



当前最大值$= 9, S(i, j) = 9$

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值$= 9, S(i, j) = 4$

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值$= 9, S(i, j) = 7$

枚举过程分析

- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值$= 9, S(i, j) = 8$

枚举过程分析

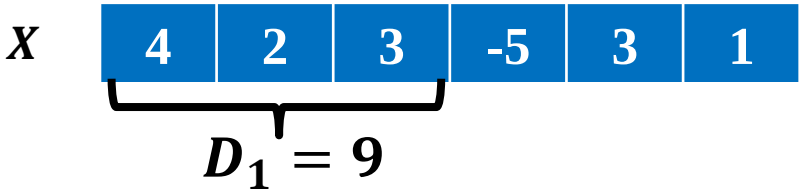
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

当前最大值= 9

D_i：以$X[i]$开头的最大子数组

枚举过程分析

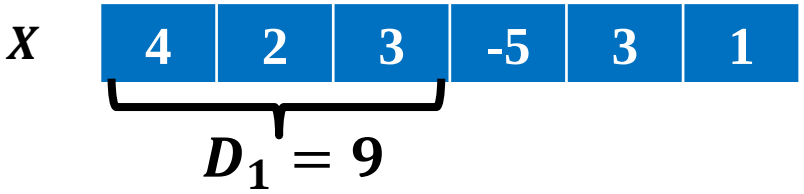
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

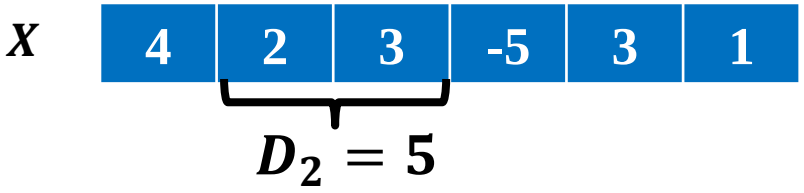


当前最大值= 9

枚举过程分析

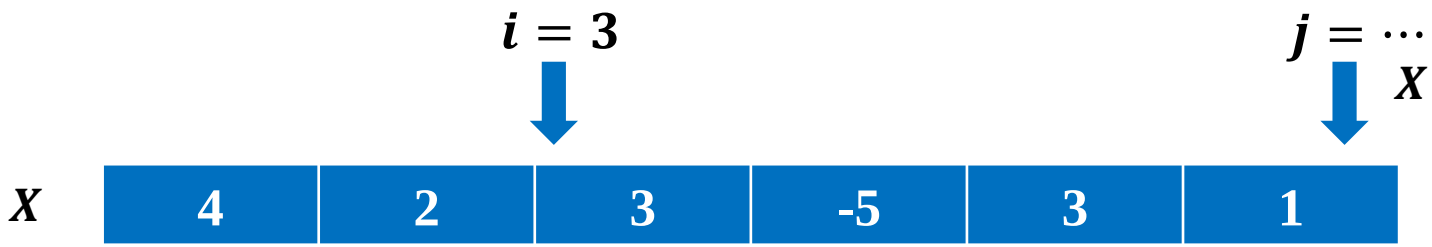
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

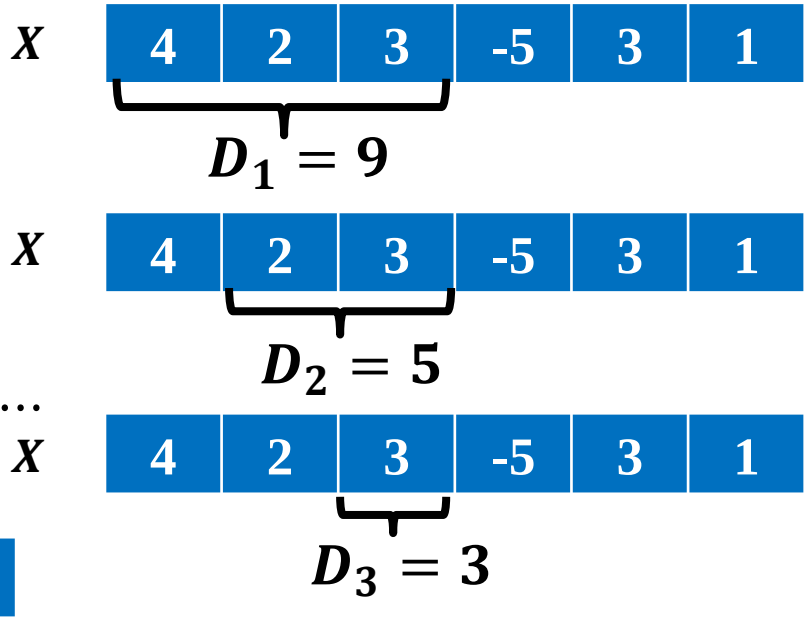




枚举过程分析

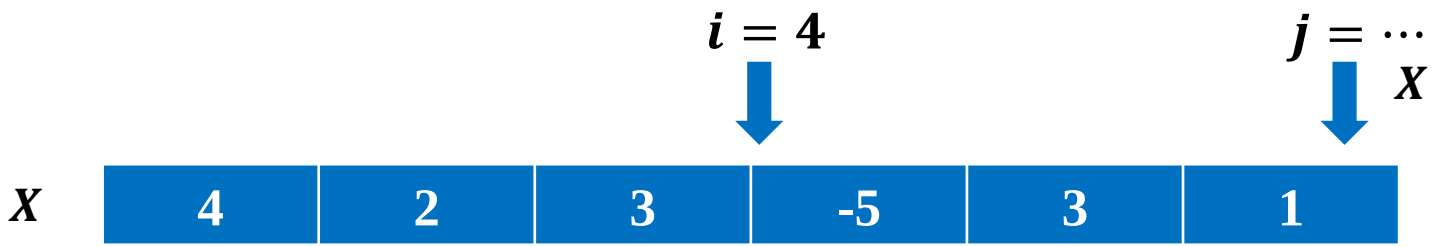
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

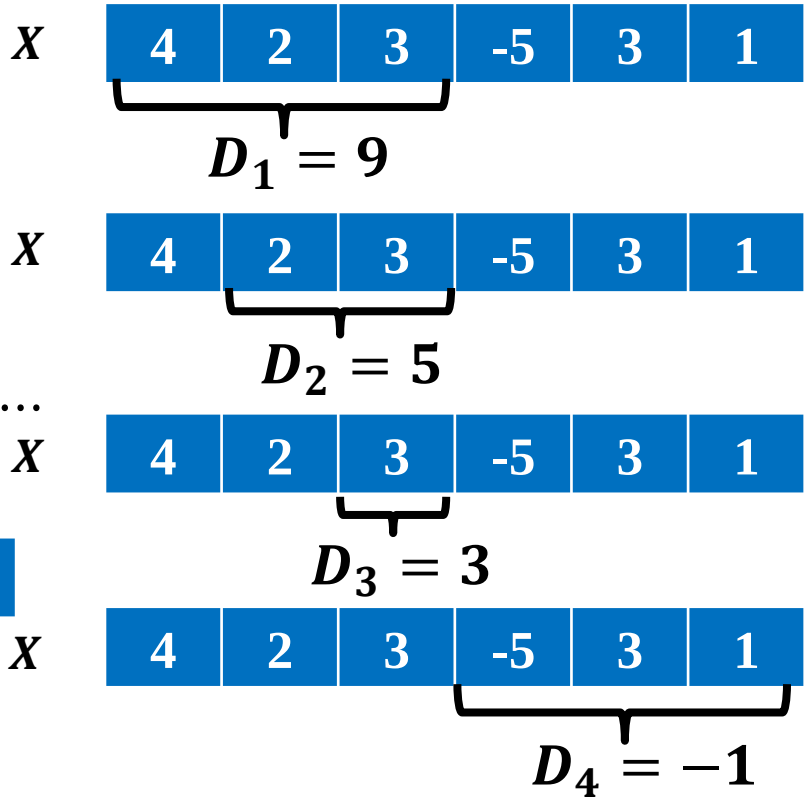




枚举过程分析

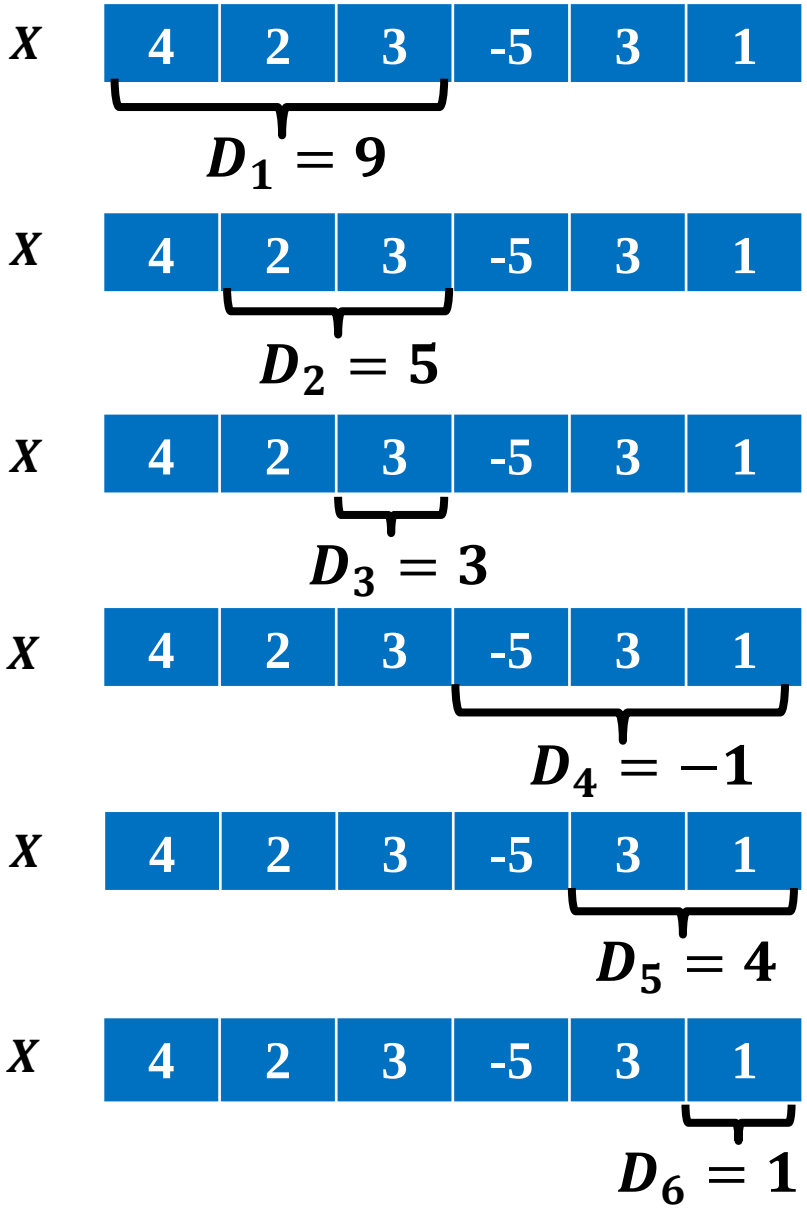
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾





枚举过程分析

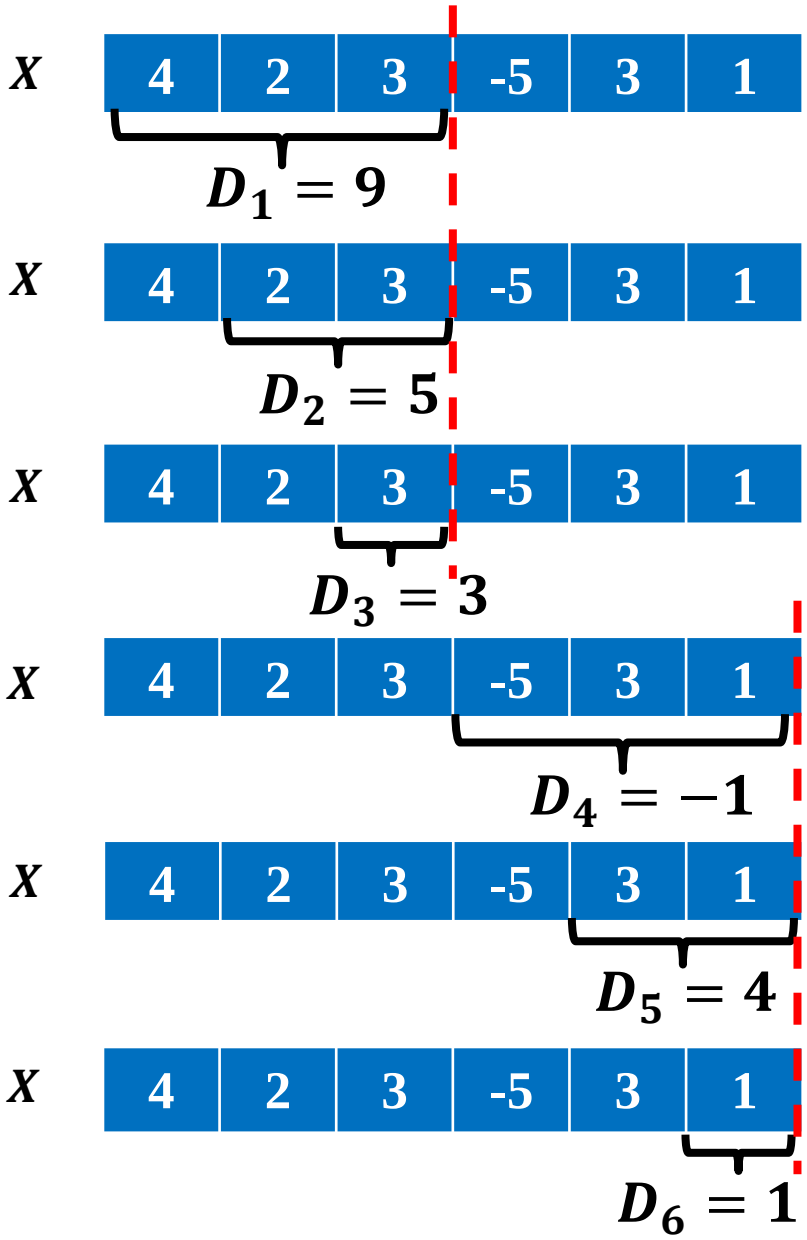
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾



枚举过程分析

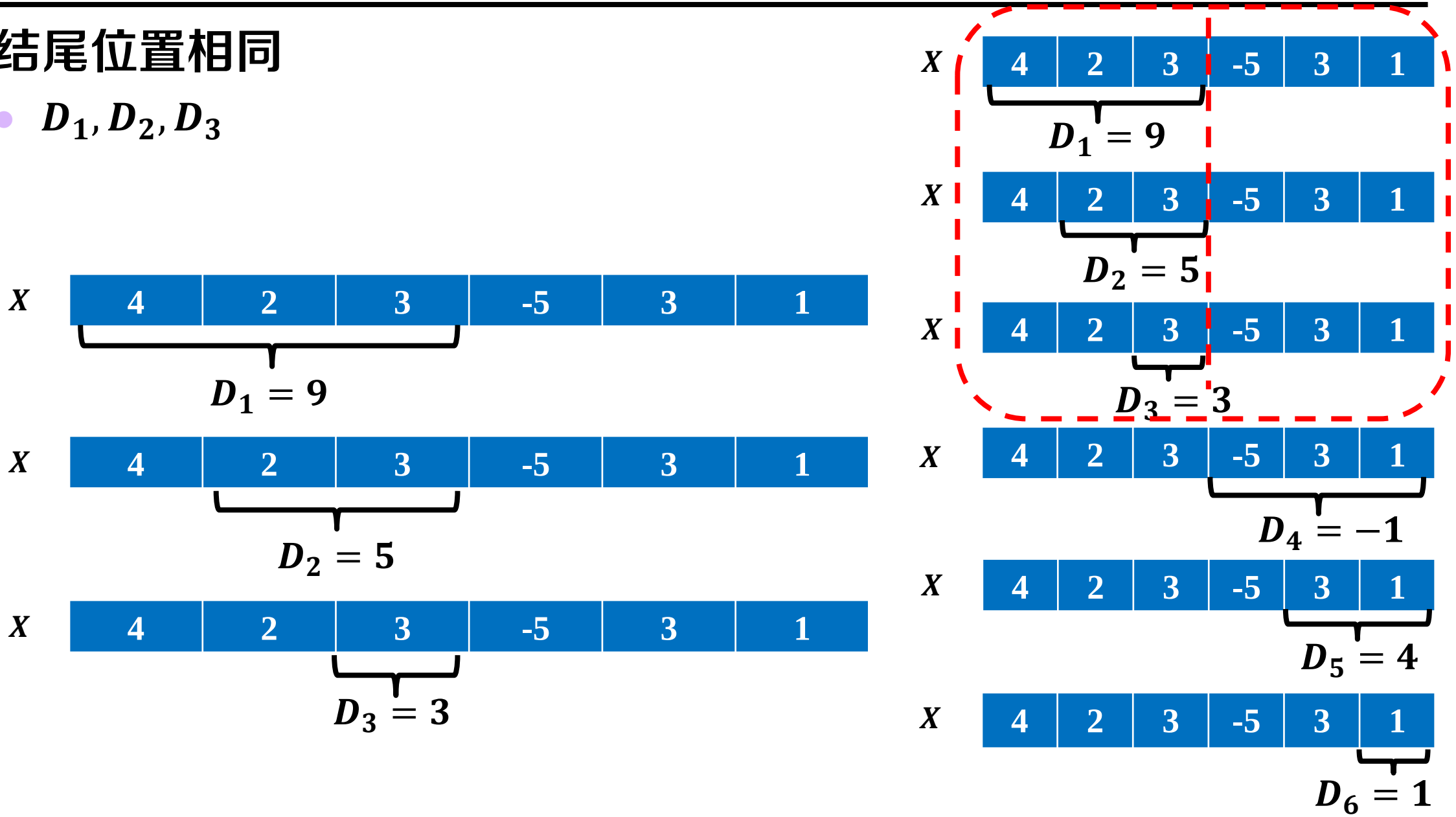
- 两层枚举
  - 第1层：枚举位置i作为区间开头
  - 第2层：枚举位置j作为区间结尾

- 观察D_i
  - 结尾相同：D_1, D_2, D_3
  - 结尾不同：D_3, D_4



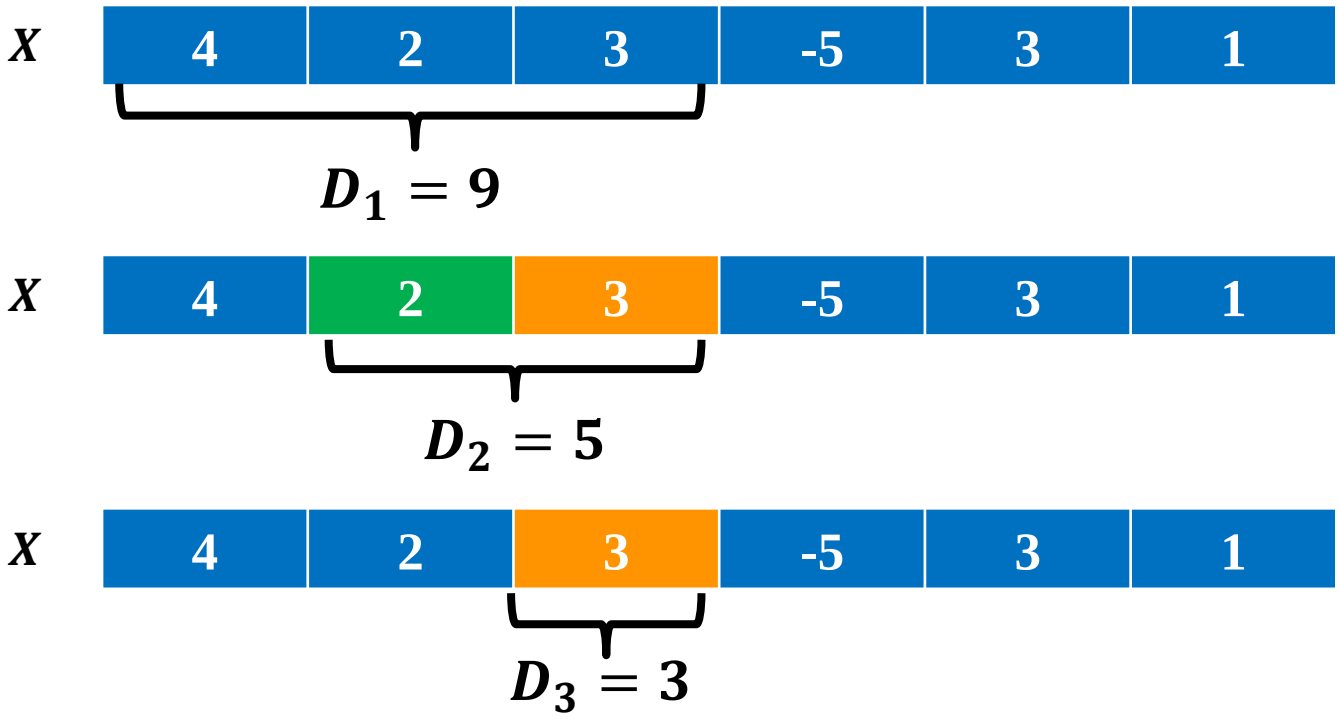
规律观察

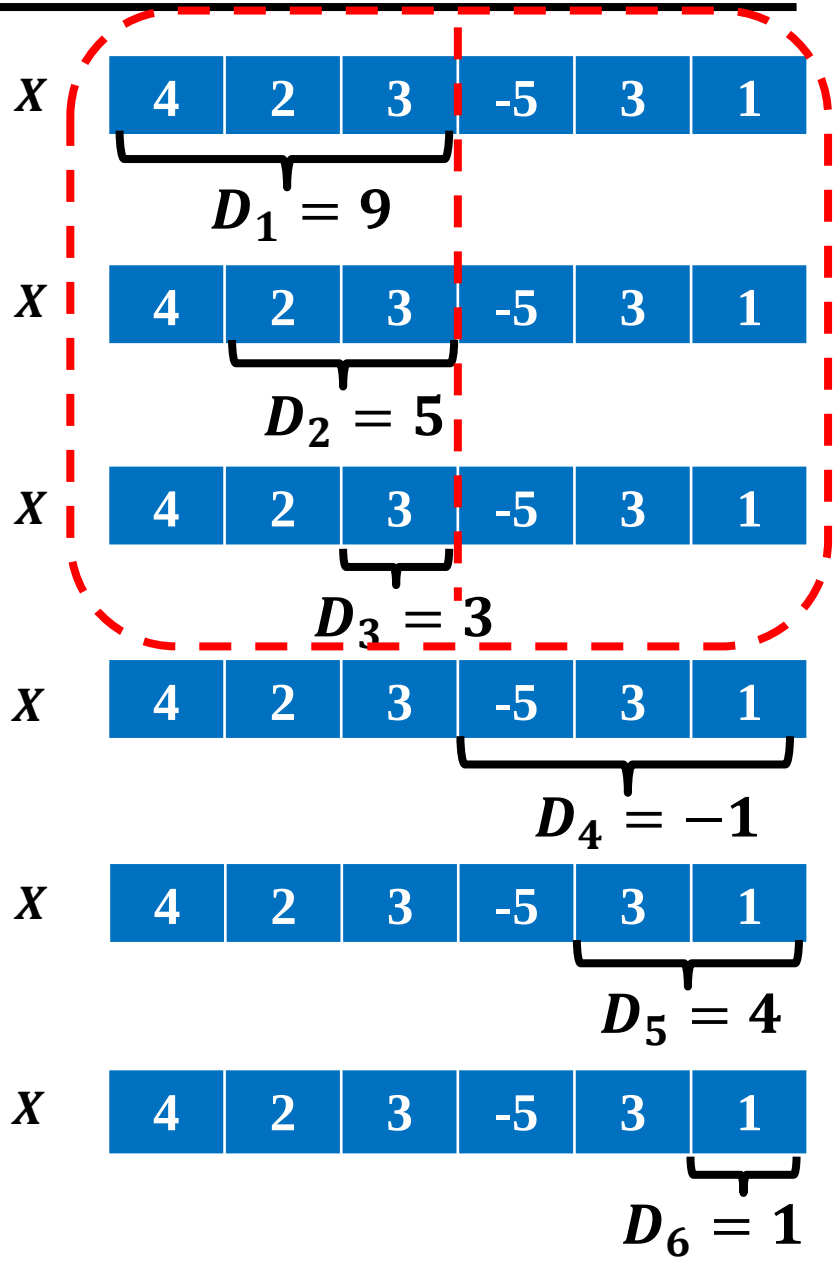
- 结尾位置相同
  - D_1, D_2, D_3



规律观察

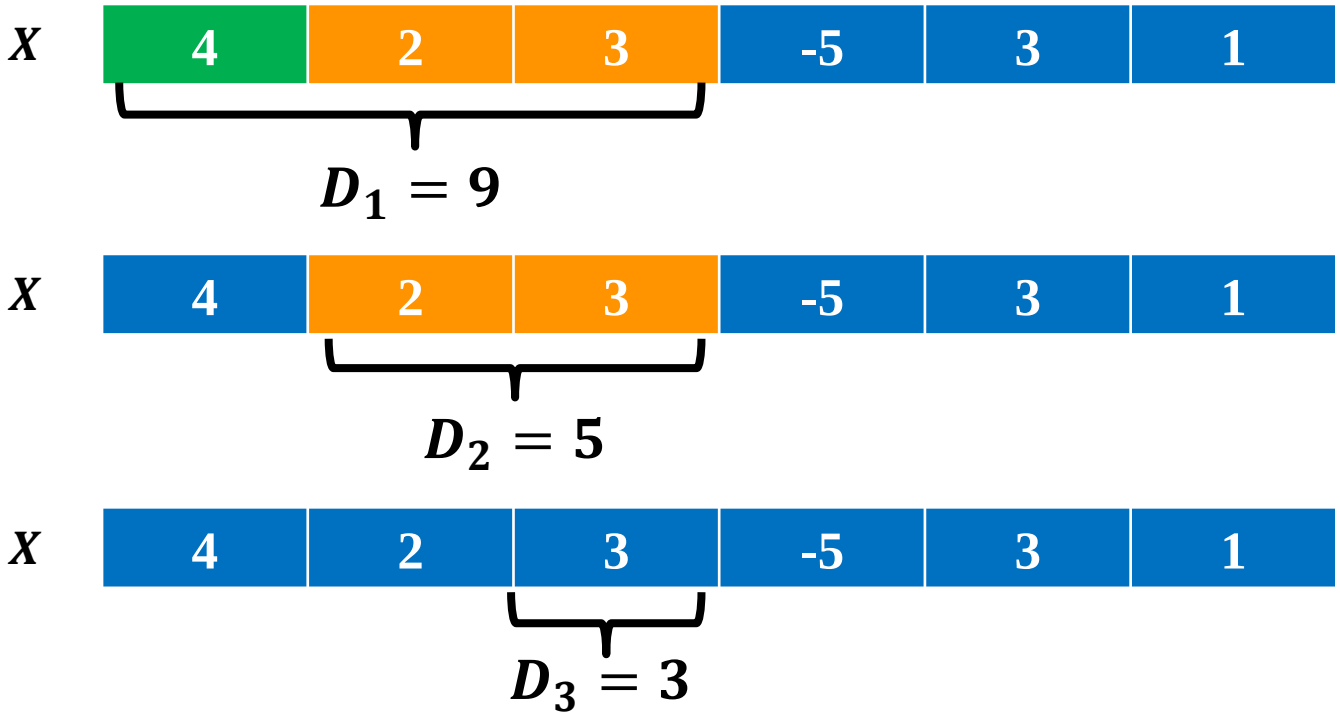
- 结尾位置相同
  - D_1, D_2, D_3
    - $D_2 = X[2] + D_3$

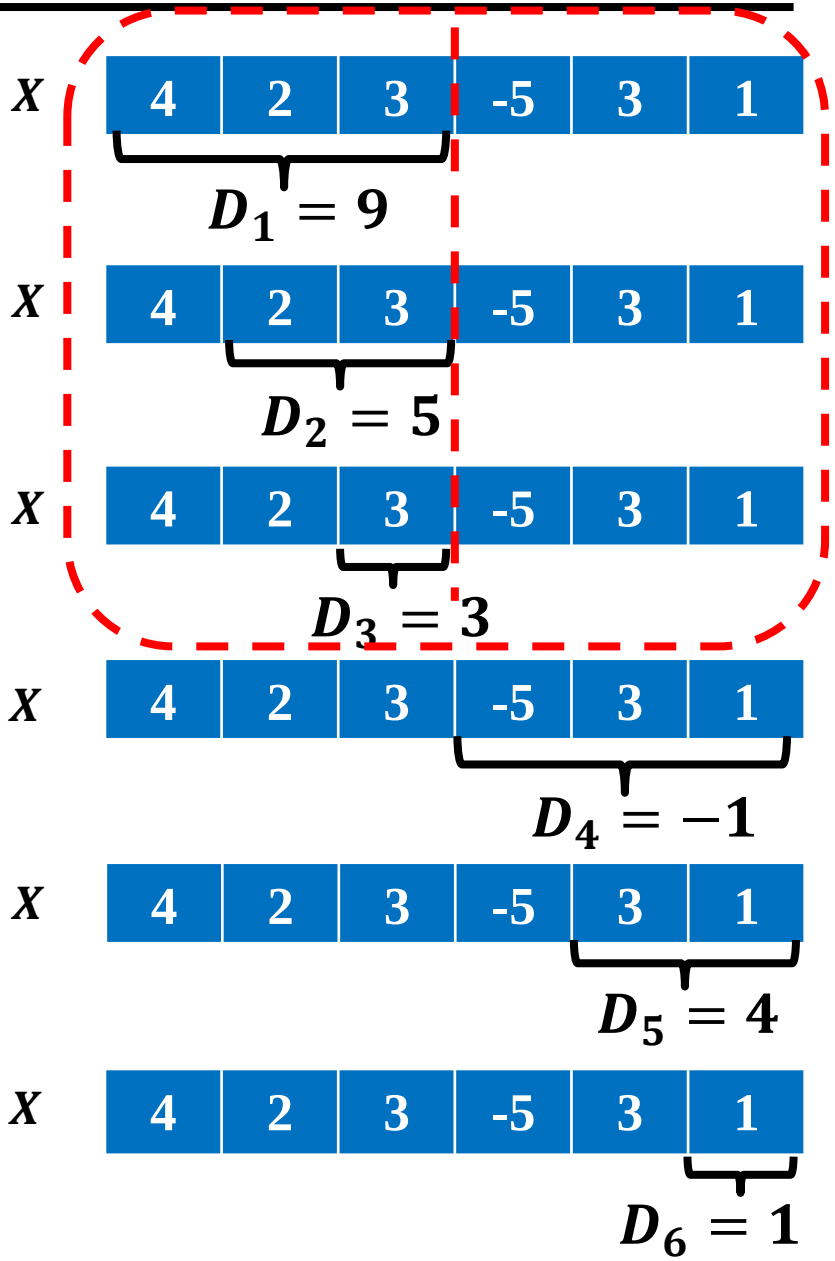




规律观察

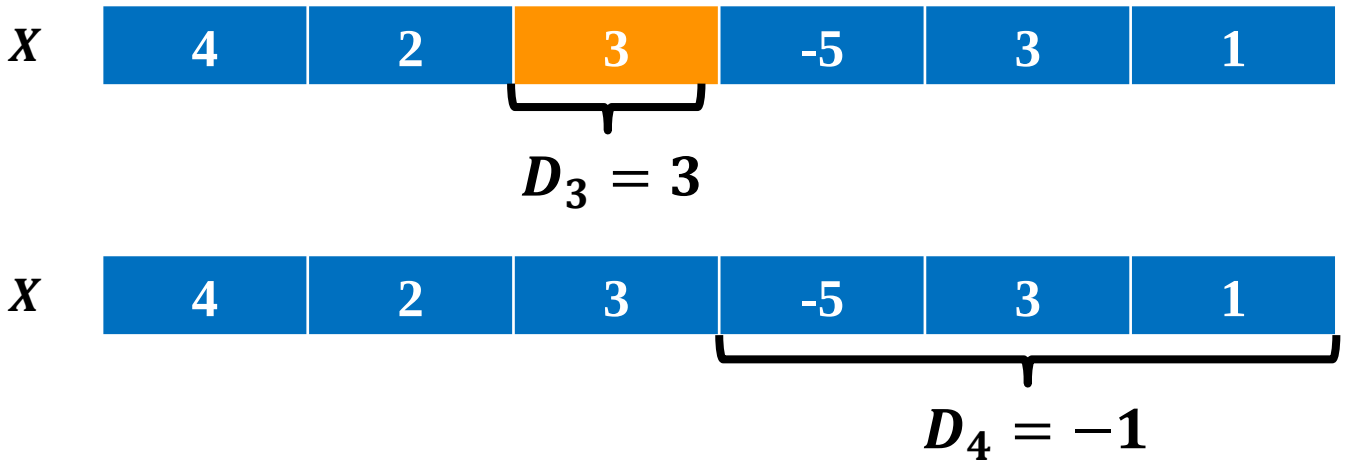
- 结尾位置相同
  - D_1, D_2, D_3
    - $D_2 = X[2] + D_3$
    - $D_1 = X[1] + D_2$

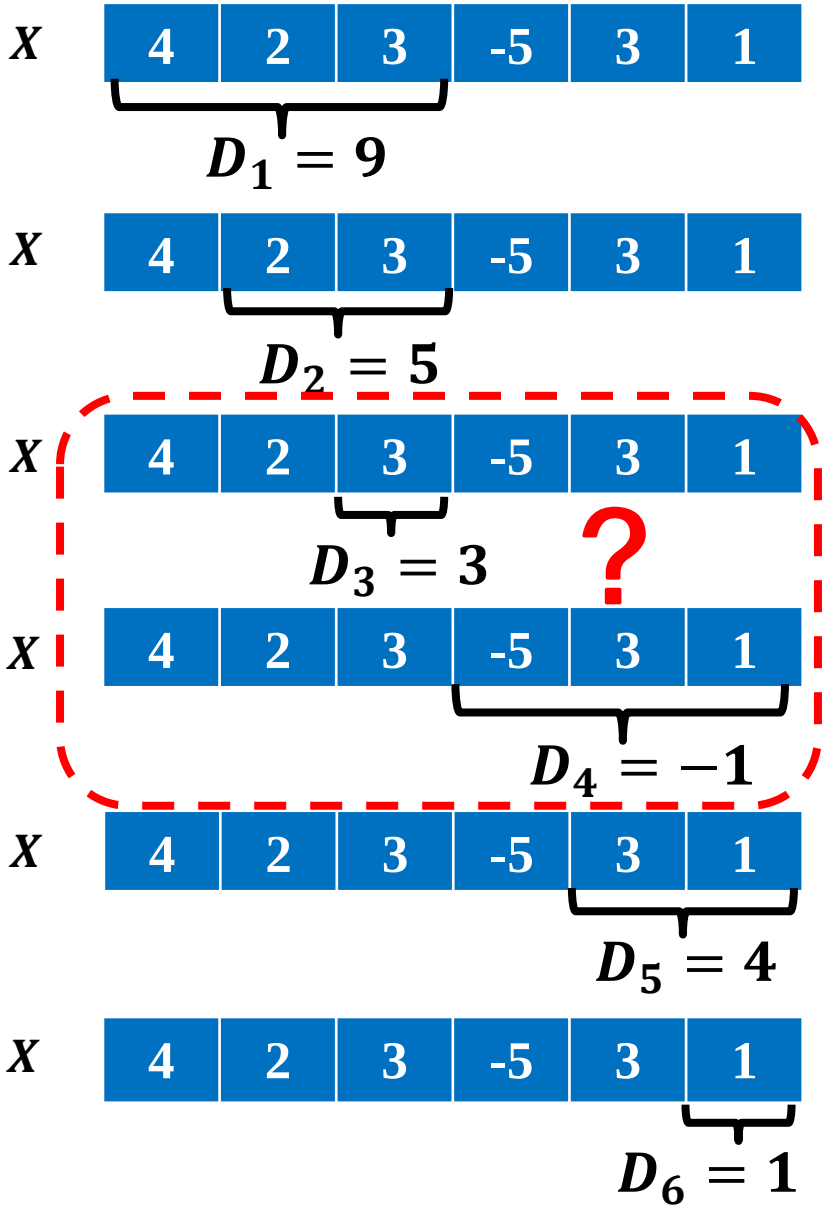




规律观察

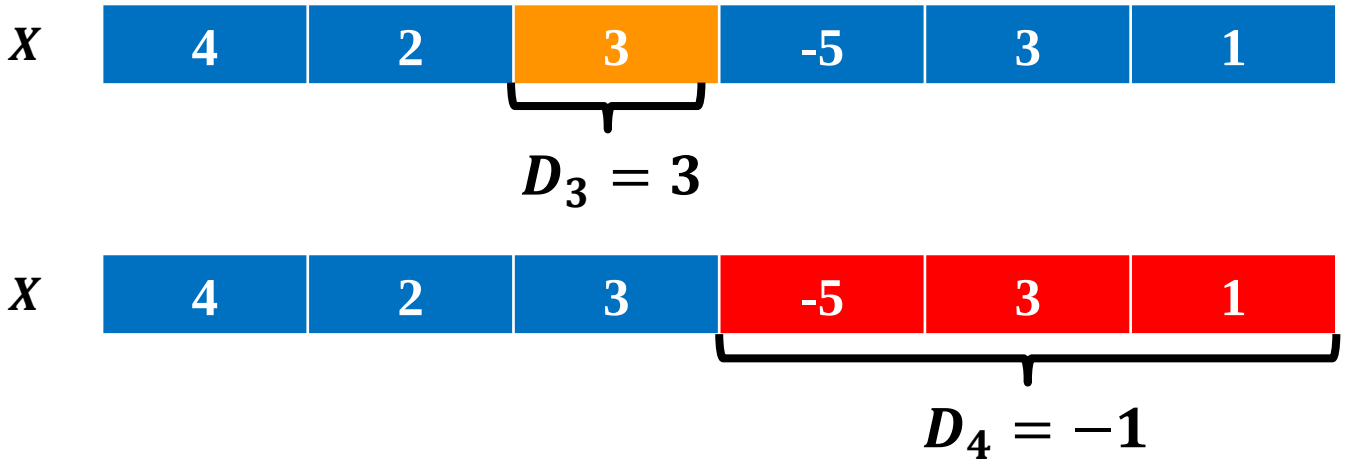
- 结尾位置不同
  - D_3, D_4
    - $D_3 = X[3]$

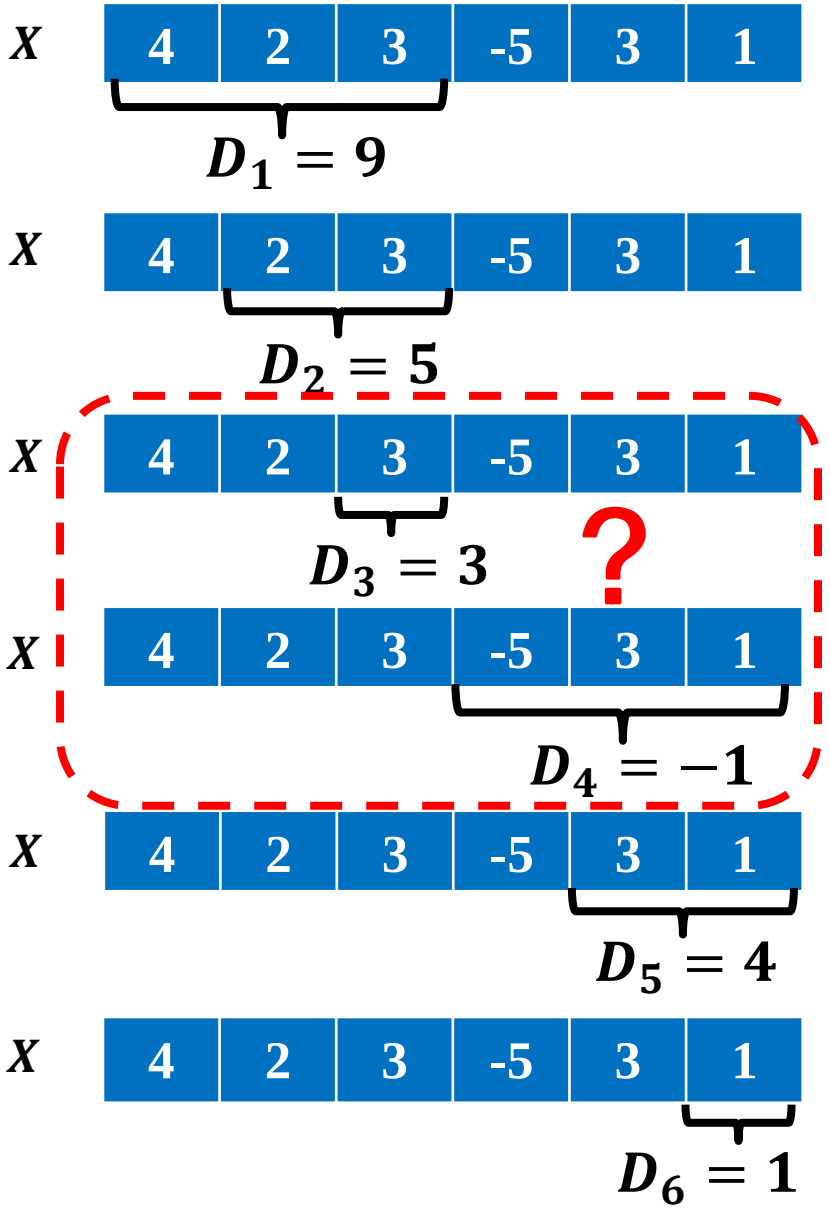




规律观察

- 结尾位置不同
  - D_3, D_4
    - $D_3 = X[3]$
    - $D_4 < 0$

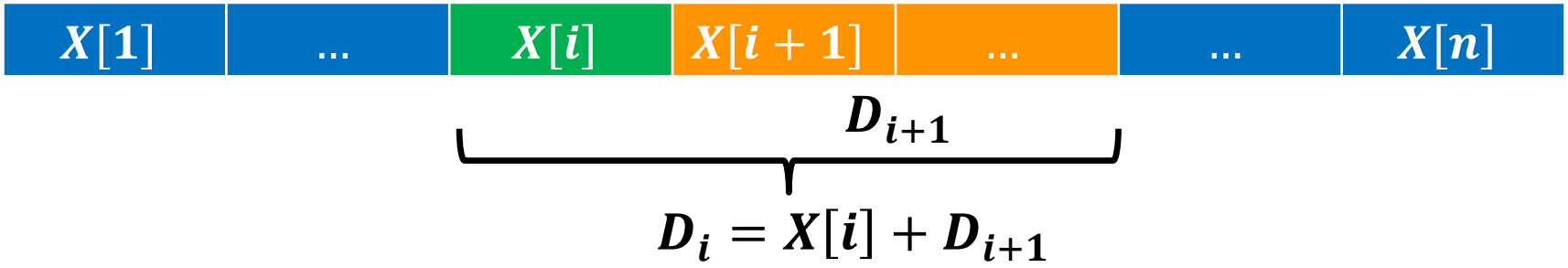




规律描述

- D_i：以$X[i]$开头的最大子数组和

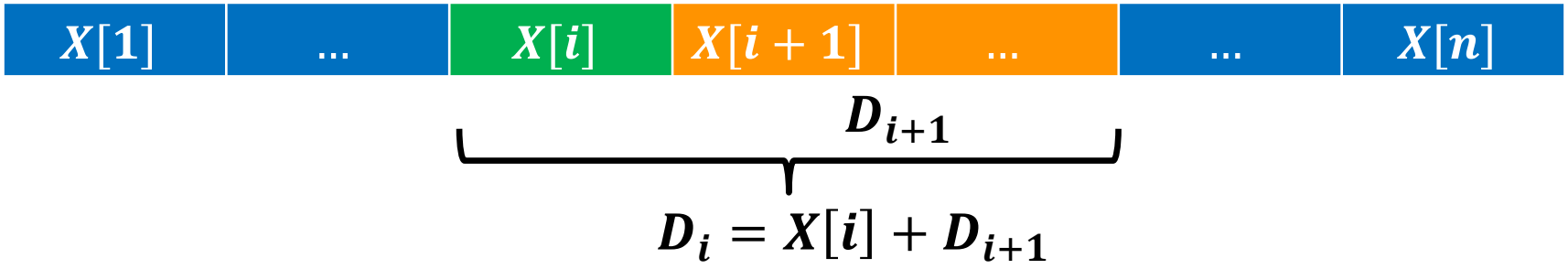
- 情况1：$D_{i+1} > 0$时



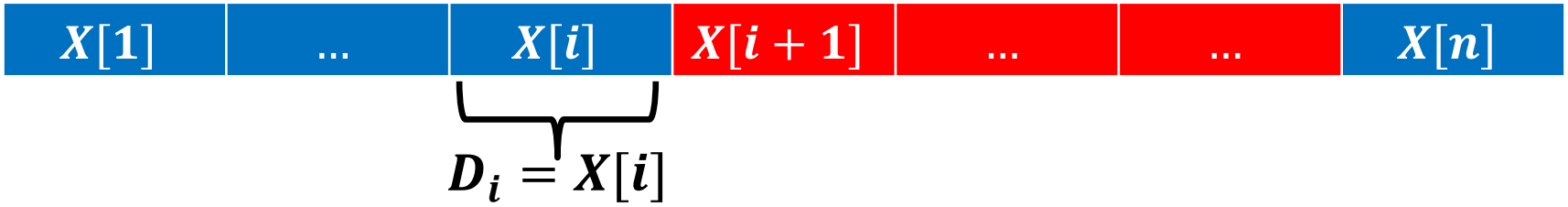
规律描述

- D_i：以$X[i]$开头的最大子数组和

- 情况1：$D_{i+1} > 0$时

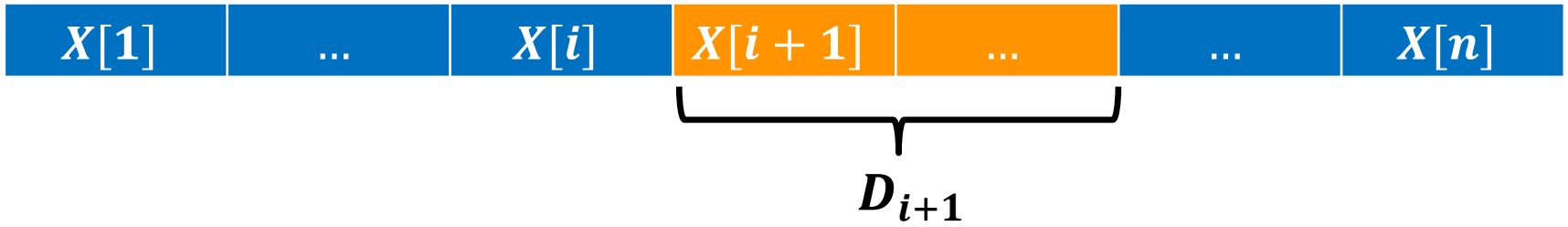


- 情况2：$D_{i+1} \leq 0$时



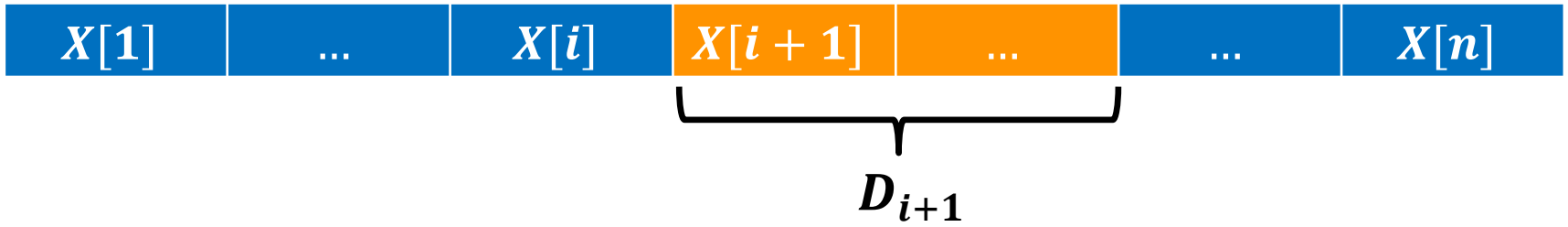
规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} > 0$）

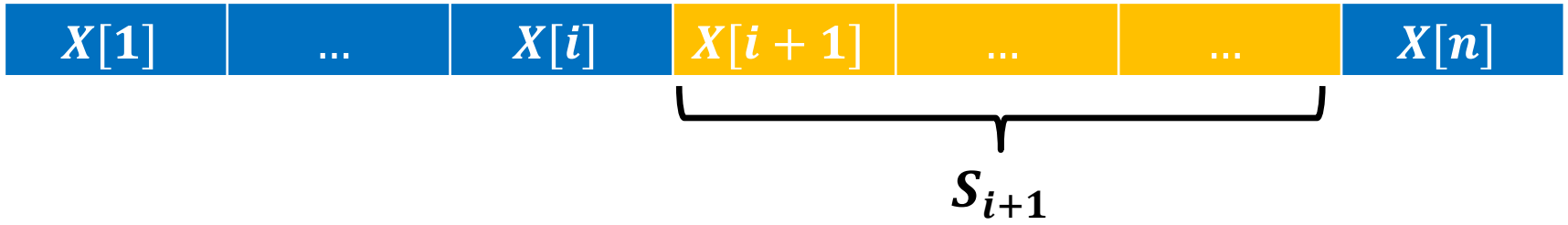


规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} > 0$）

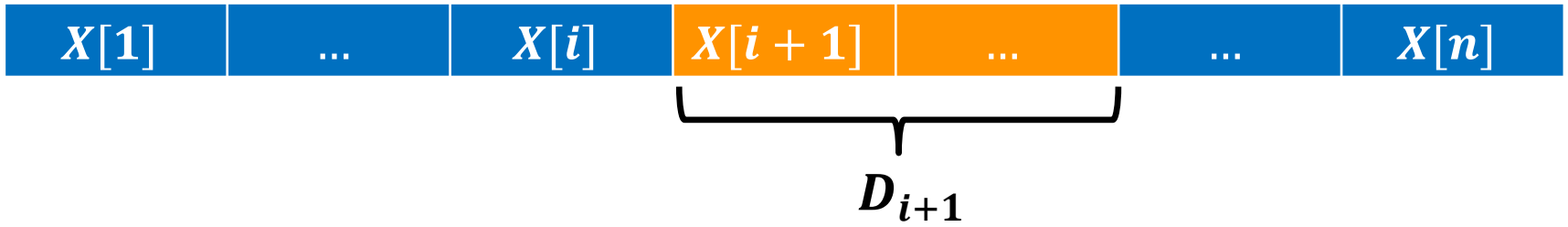


- S_{i+1}：以$X[i+1]$开头的任一子数组和

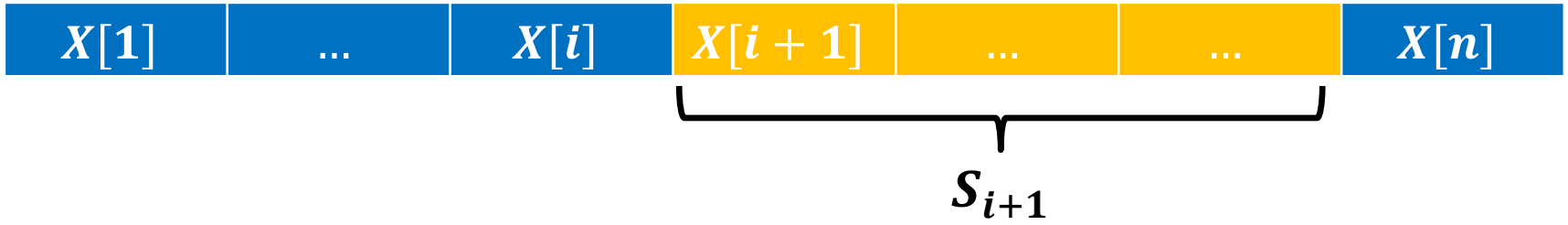


规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的**最大**子数组和（$D_{i+1} > 0$）

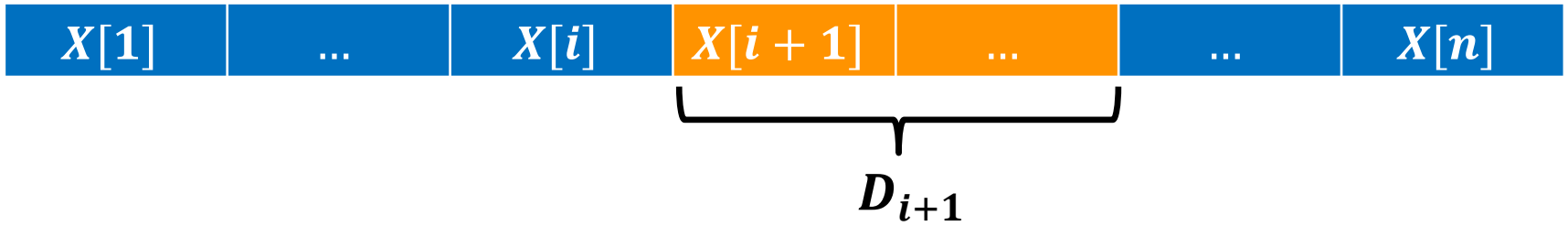


- S_{i+1}：以$X[i+1]$开头的**任一**子数组和

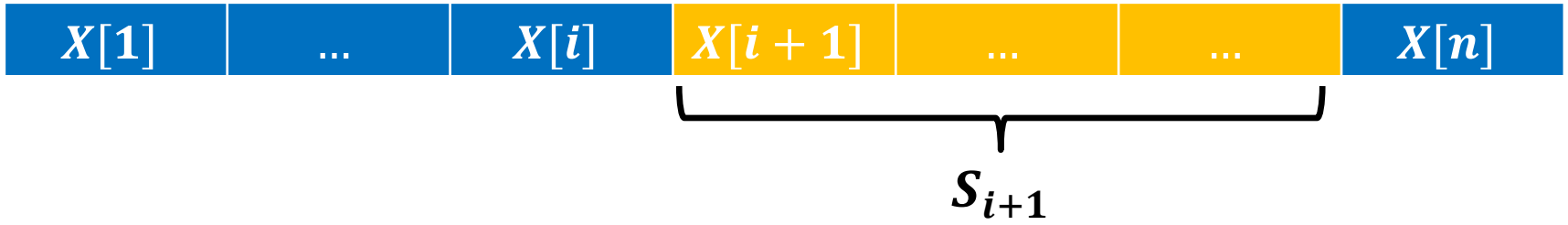


规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的**最大**子数组和（$D_{i+1} > 0$）

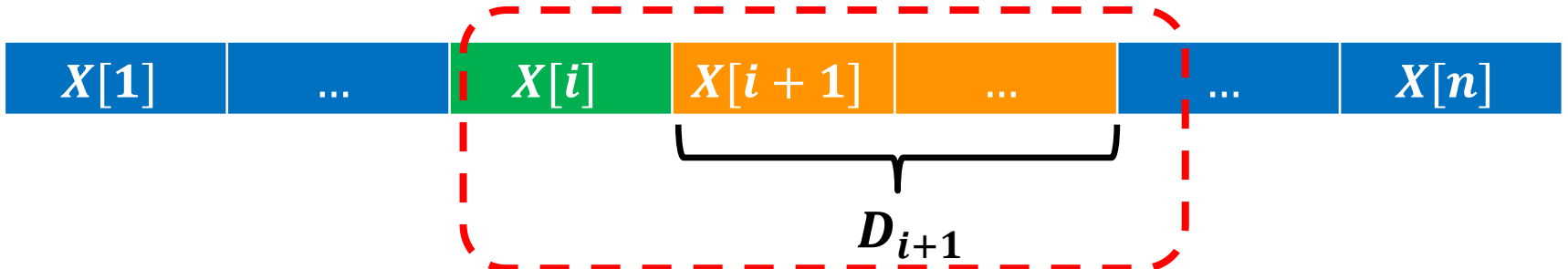


- S_{i+1}：以$X[i+1]$开头的**任一**子数组和（$S_{i+1} \leq D_{i+1}$）

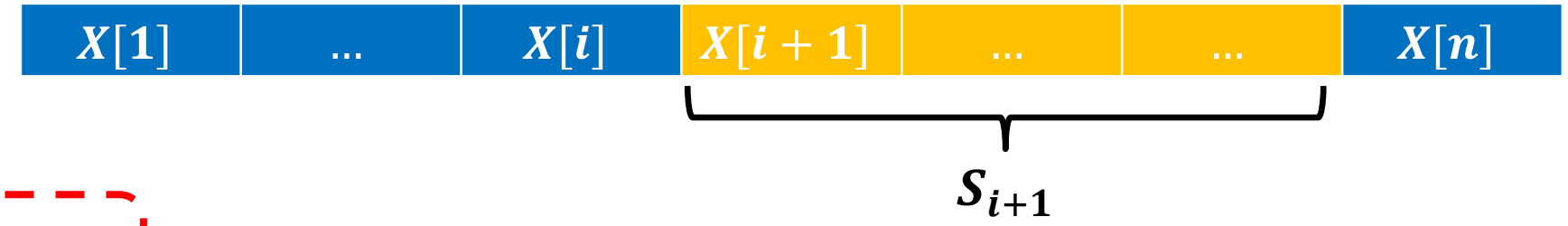


规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和 $(D_{i+1} > 0)$



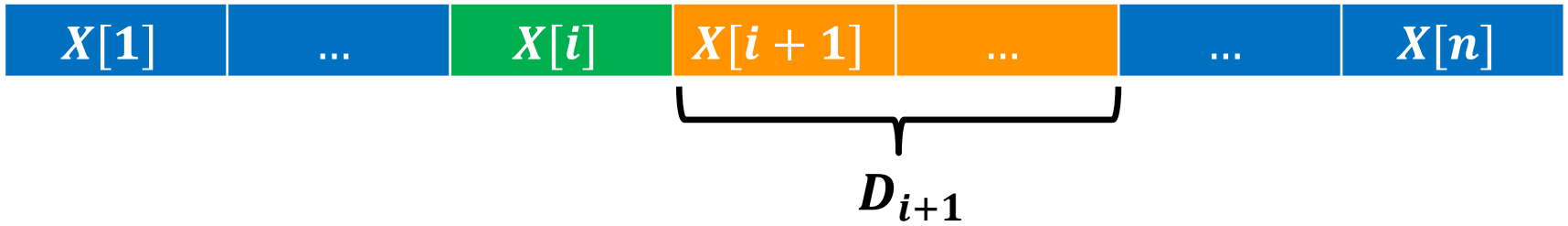
- S_{i+1}：以$X[i+1]$开头的任一子数组和 $(S_{i+1} \leq D_{i+1})$



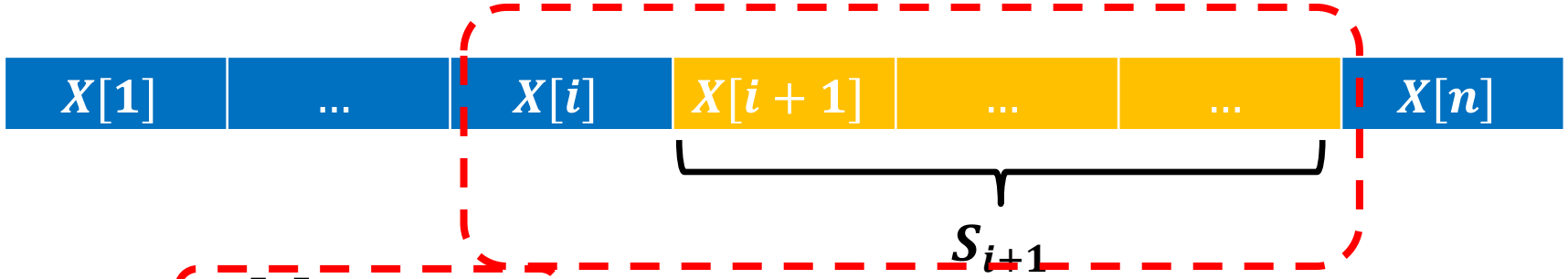
- $X[i] + D_{i+1}$

规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} > 0$）



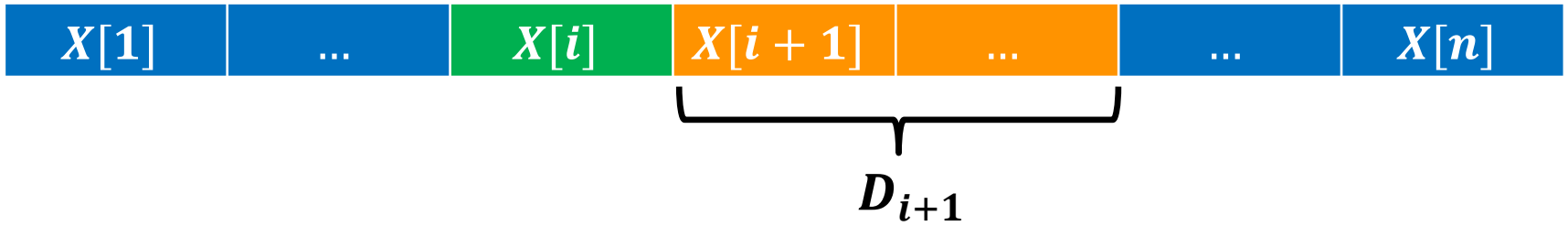
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



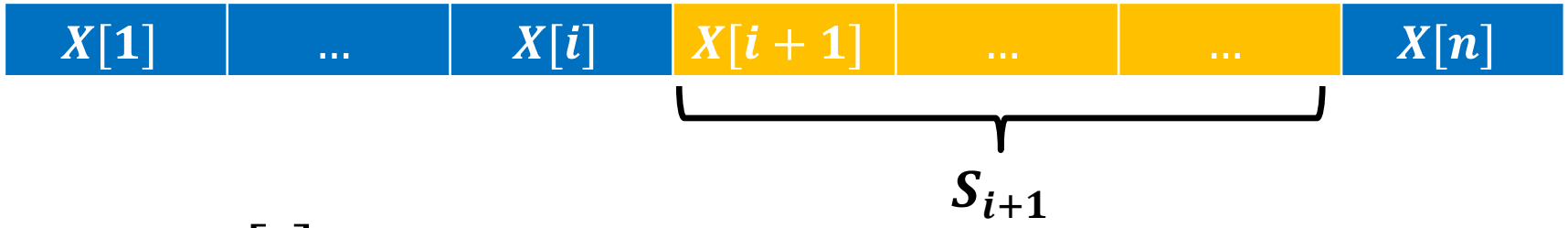
- $X[i] + D_{i+1}$　$X[i] + S_{i+1}$

规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} > 0$）



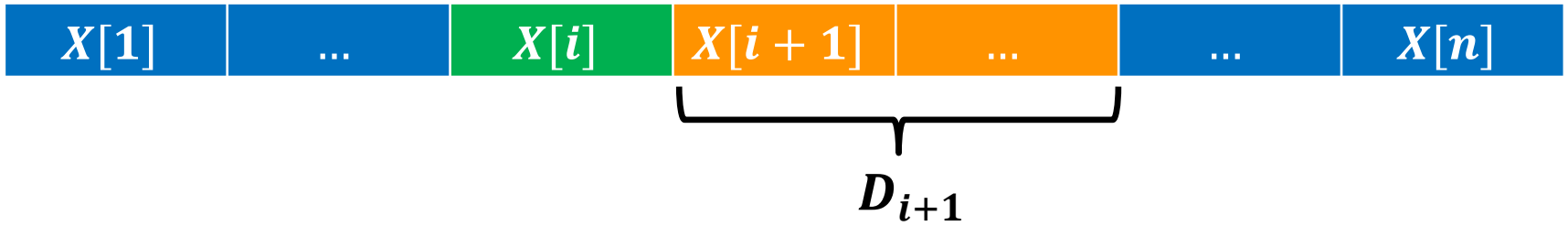
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



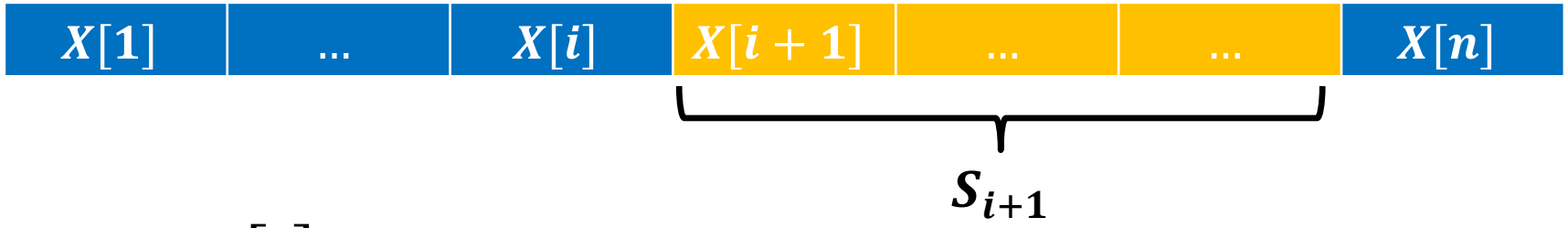
- $X[i] + D_{i+1} \geq X[i] + S_{i+1}$

规律证明：情况1

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} > 0$）



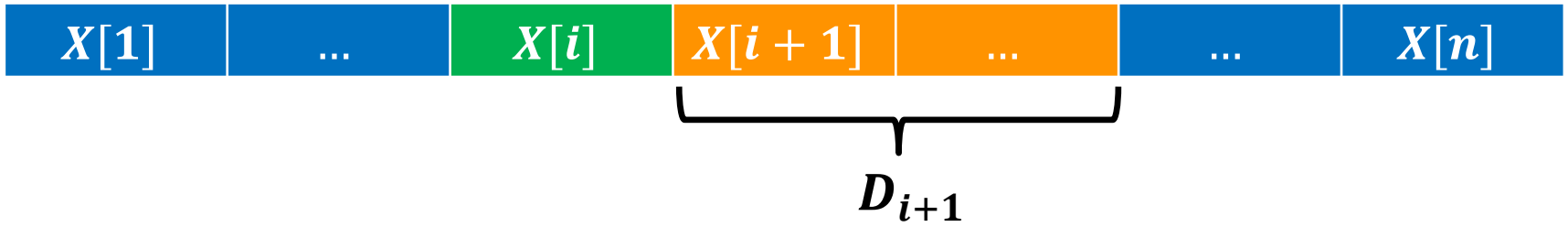
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



- $X[i] + D_{i+1} \geq X[i] + S_{i+1}$
- $D_i = X[i] + D_{i+1}$

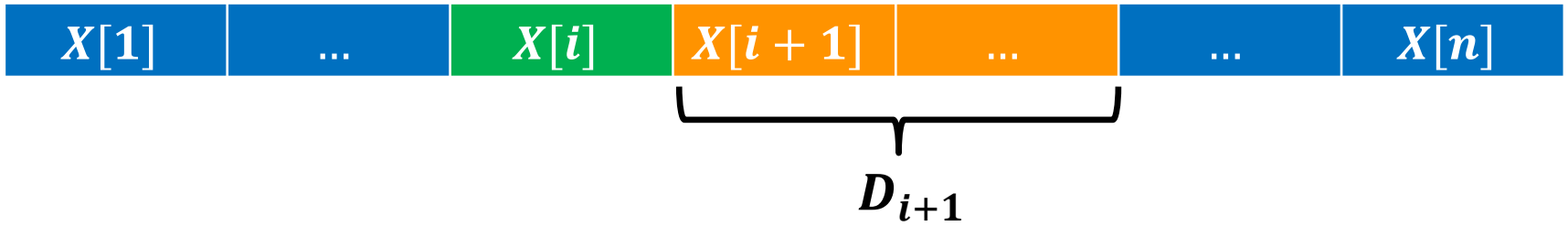
规律证明：情况2

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} \leq 0$）

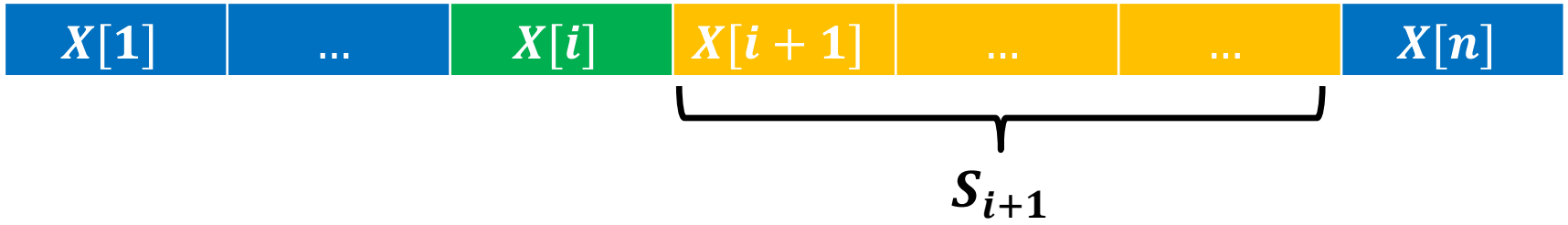


规律证明：情况2

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} \leq 0$）

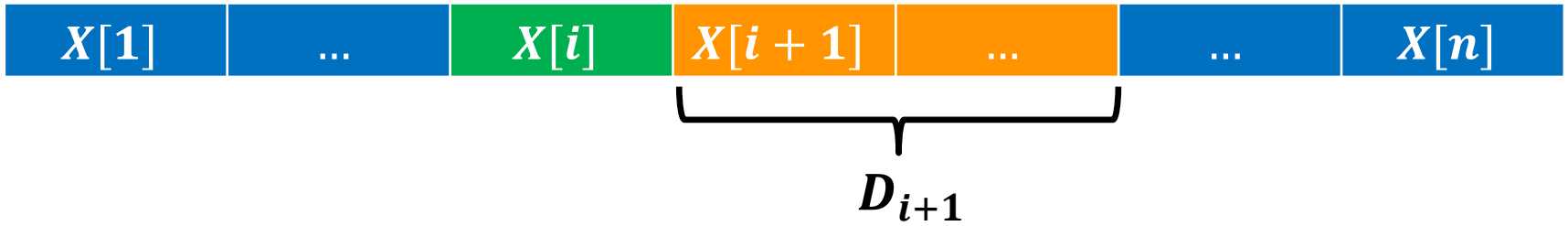


- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）

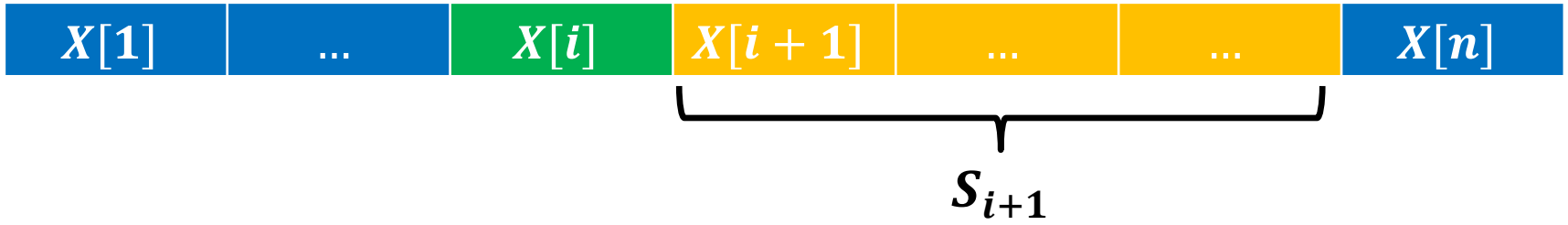


规律证明：情况2

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} \leq 0$）



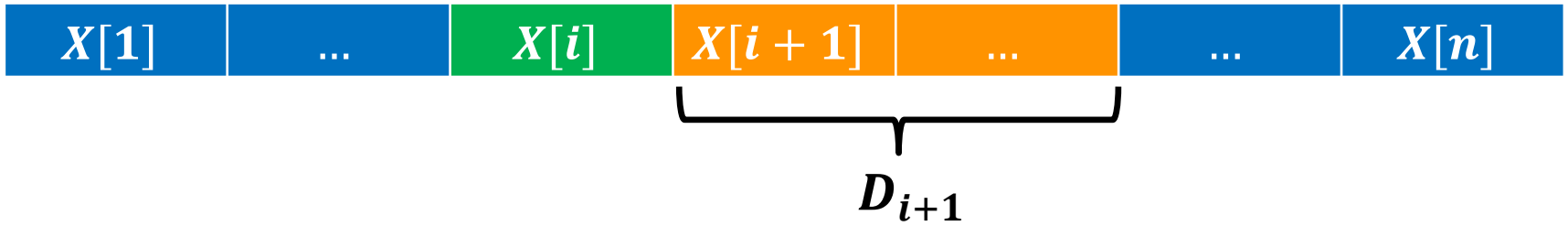
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



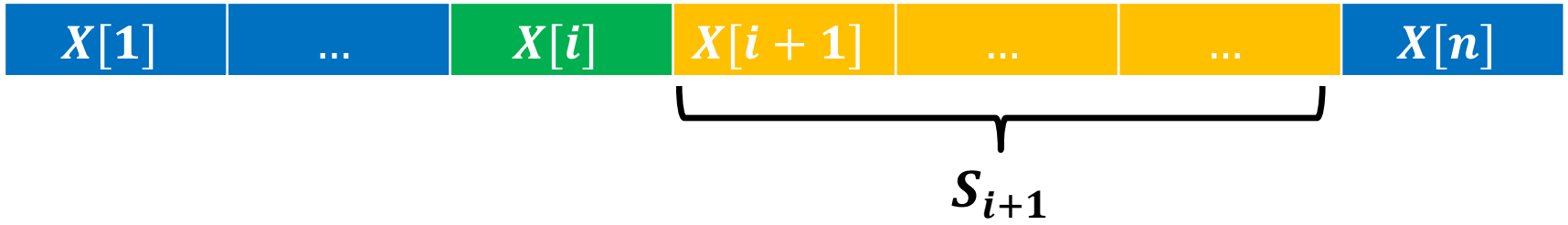
- $X[i] + S_{i+1} \leq X[i] + D_{i+1}$

规律证明：情况2

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} \leq 0$）



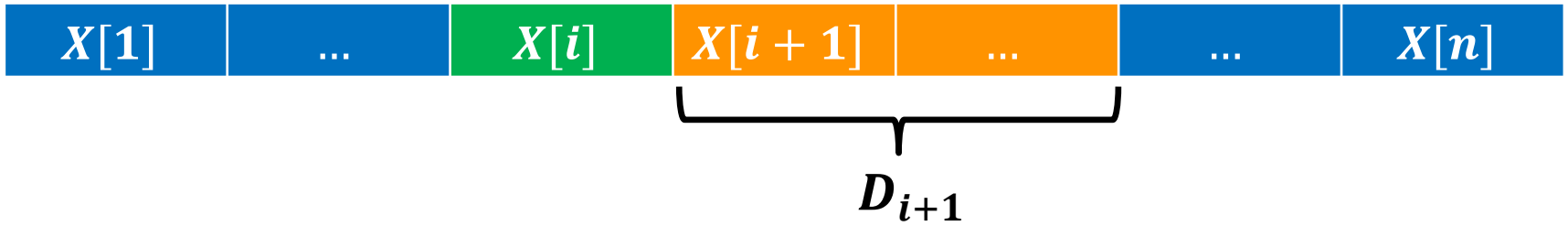
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



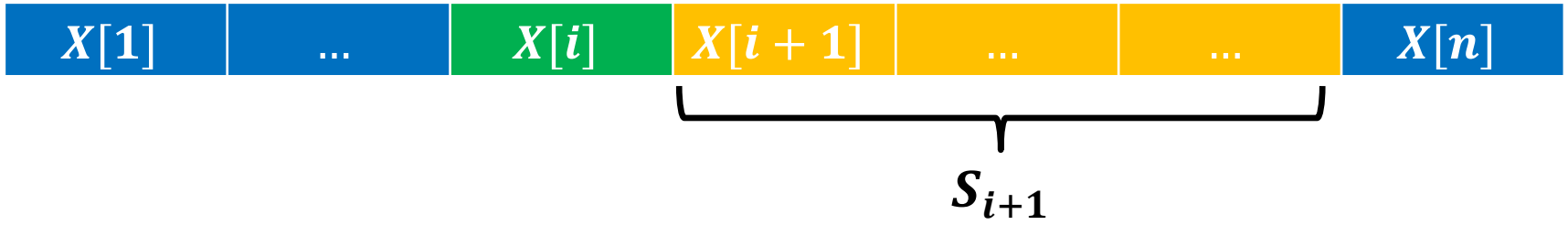
- $X[i] + S_{i+1} \leq X[i] + D_{i+1} \leq X[i]$

规律证明：情况2

- D_{i+1}：以$X[i+1]$开头的最大子数组和（$D_{i+1} \leq 0$）



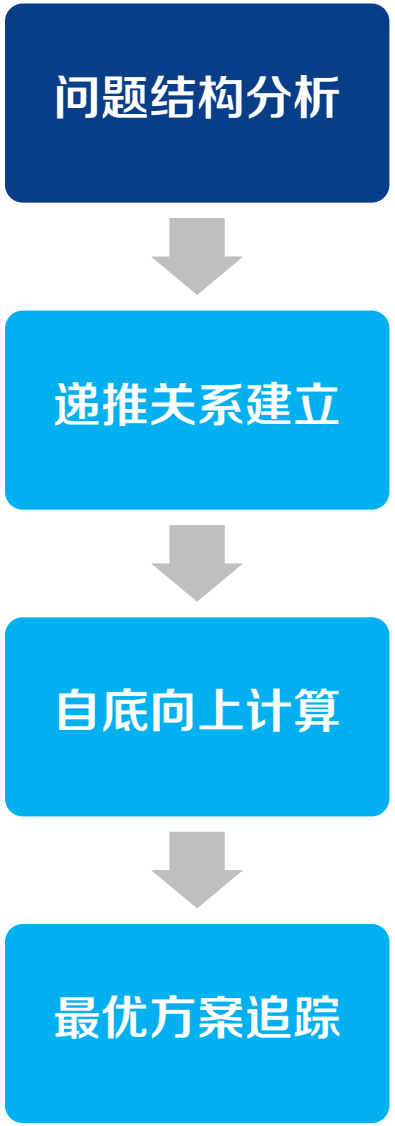
- S_{i+1}：以$X[i+1]$开头的任一子数组和（$S_{i+1} \leq D_{i+1}$）



- $X[i] + S_{i+1} \leq X[i] + D_{i+1} \leq X[i]$
- $D_i = X[i]$

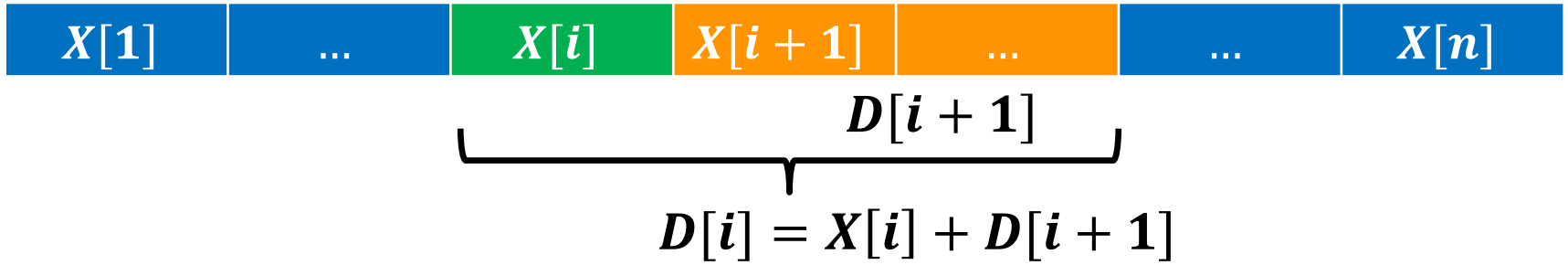
问题结构分析

- 给出问题表示
  - $D[i]$: 以$X[i]$开头的最大子数组和
- 明确原始问题
  - $S_{max} = \max\limits_{1 \le i \le n}\{D[i]\}$

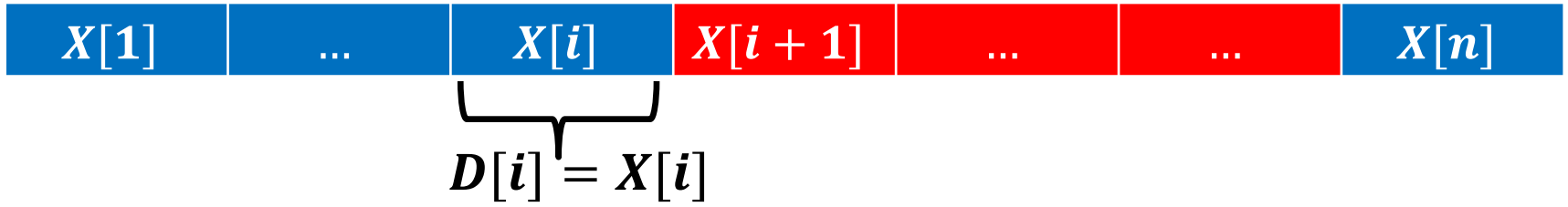


递推关系建立：分析最优（子）结构

- $D[i]$：以$X[i]$开头的**最大**子数组和
- 情况1：$D[i+1] > 0$



- 情况2：$D[i+1] \leq 0$



问题结构分析

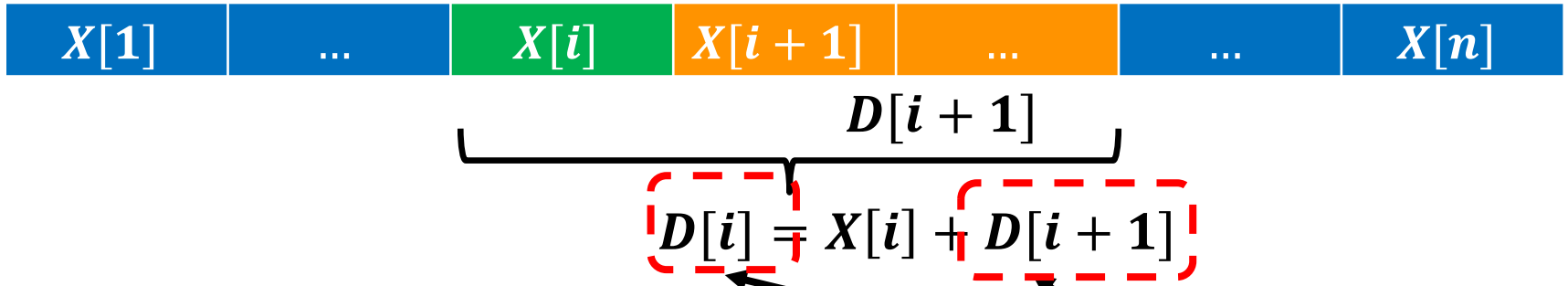
递推关系建立

自底向上计算

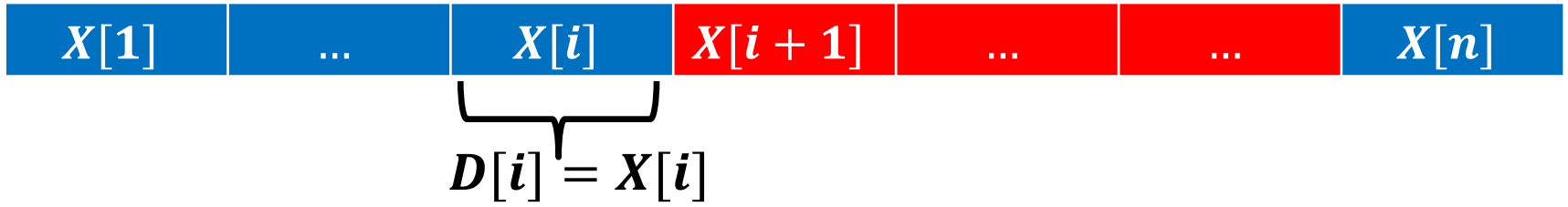
最优方案追踪

递推关系建立：分析最优（子）结构

- $D[i]$：以$X[i]$开头的**最大**子数组和
- 情况1：$D[i+1] > 0$



- 情况2：$D[i+1] \leq 0$



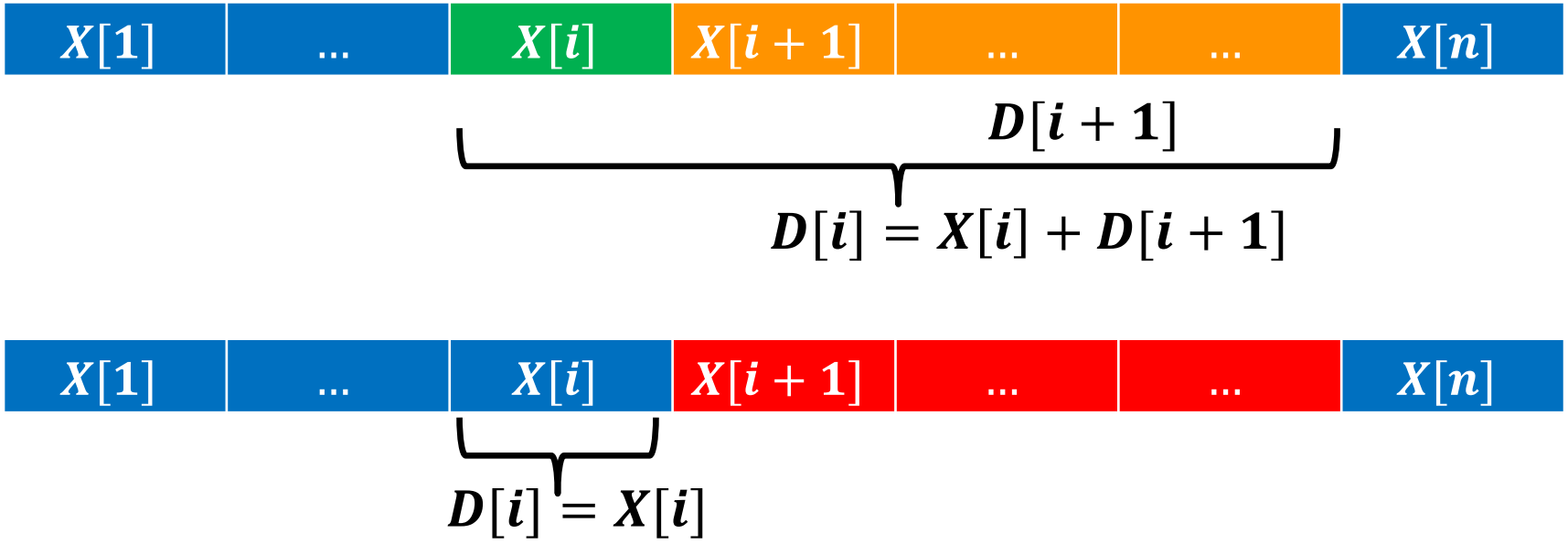
问题结构分析

递推关系建立

自底向上计算

最优方案追踪

递推关系建立：构造递推公式



- $D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$

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

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

自底向上计算：确定计算顺序

- 初始化
  - $D[n] = X[n]$

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	…	i	i+1	…	$n-1$	n
D								$X[n]$

自底向上计算：确定计算顺序

- 初始化
  - $D[n] = X[n]$
- 递推公式
  - $$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \le 0 \end{cases}$$

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	…	i	i+1	…	$n-1$	n
D								$X[n]$

自底向上计算：确定计算顺序

- 初始化
  - $D[n] = X[n]$
- 递推公式
  - $$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \le 0 \end{cases}$$

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	…	i	i+1	…	$n-1$	n
D					$D[i+1]$			$X[n]$

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

- 初始化
  - $D[n] = X[n]$
- 递推公式
  - $$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \le 0 \end{cases}$$

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	…	i	i+1	…	$n-1$	n
D				$D[i]$	$D[i+1]$			$X[n]$

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：依次求解问题

- 初始化
  - $D[n] = X[n]$
- 递推公式
  - $$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \le 0 \end{cases}$$

已知

	1	2	…	i	i+1	…	$n-1$	n
X	$X[1]$	$X[2]$		$X[i]$				$X[n]$

	1	2	…	i	i+1	…	$n-1$	n
D	←							$X[n]$

自底向上计算

问题结构分析

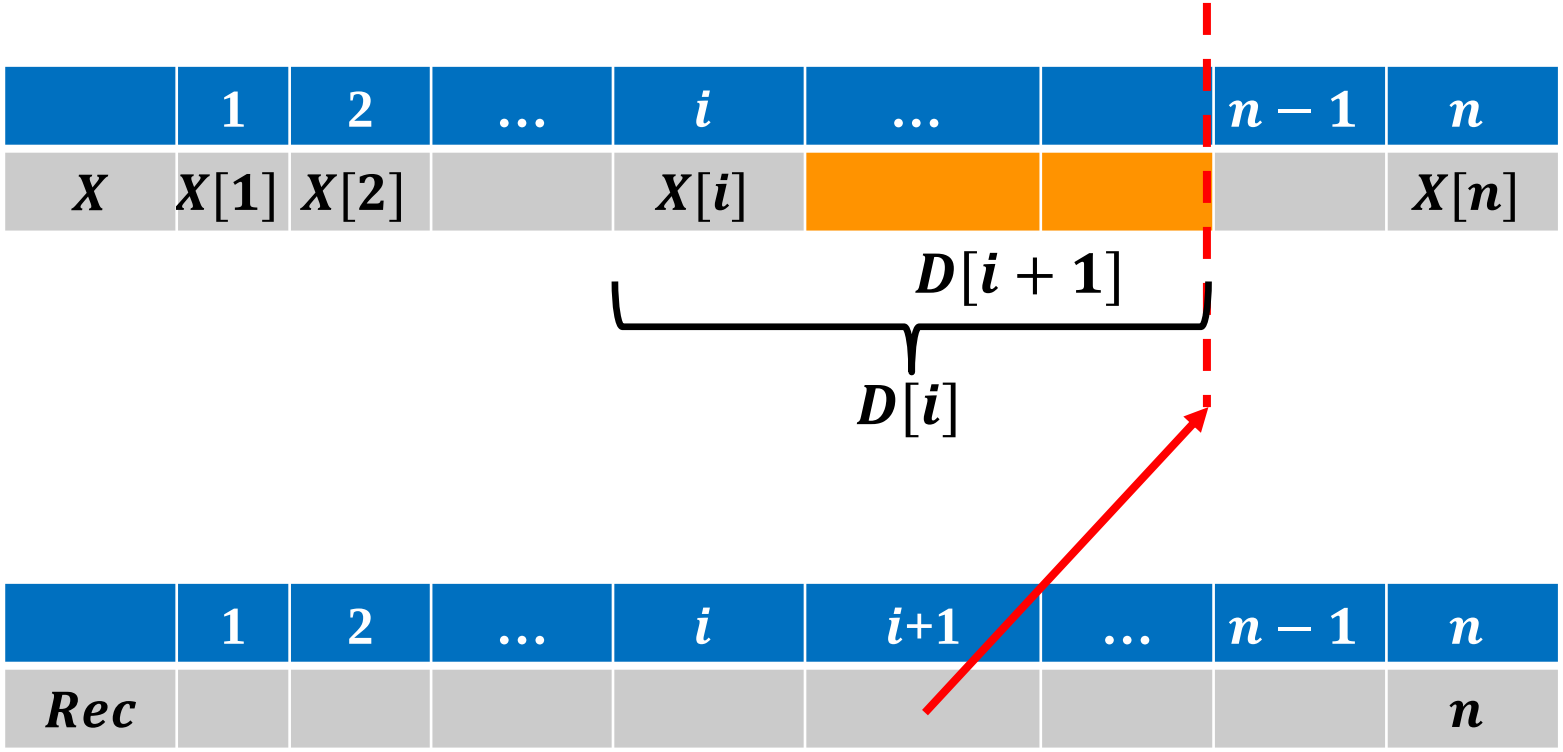
递推关系建立

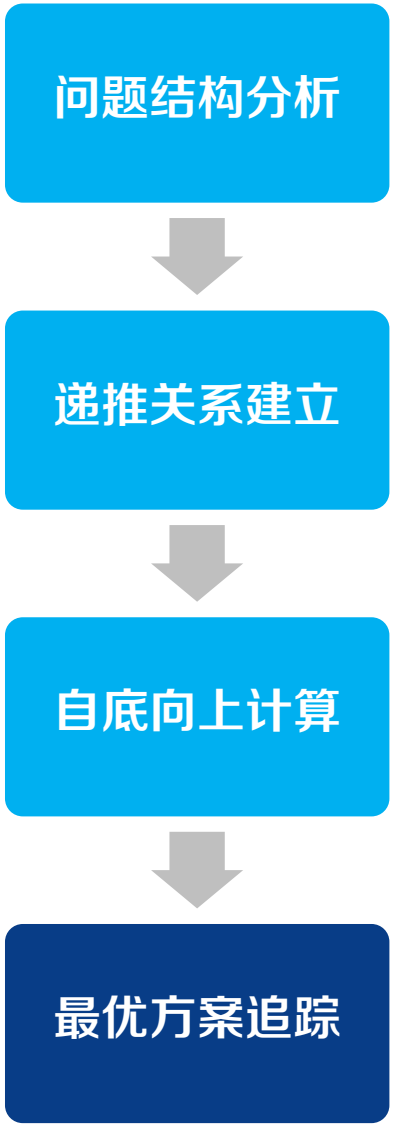
自底向上计算

最优方案追踪

最优方案追踪：记录决策过程

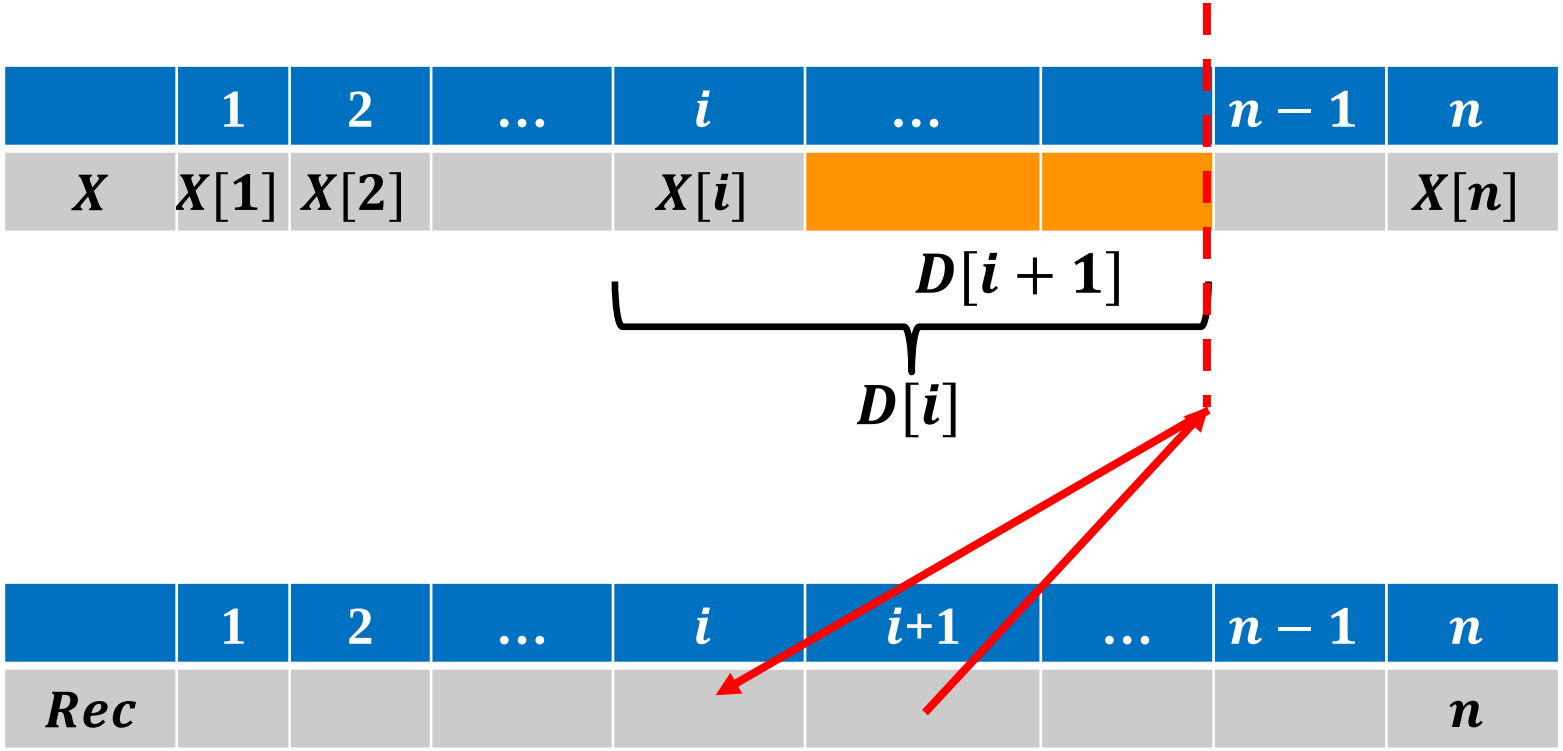
- 构造追踪数组$Rec[1..n]$
  - 情况1：结尾相同
    - $Rec[i] = Rec[i+1]$

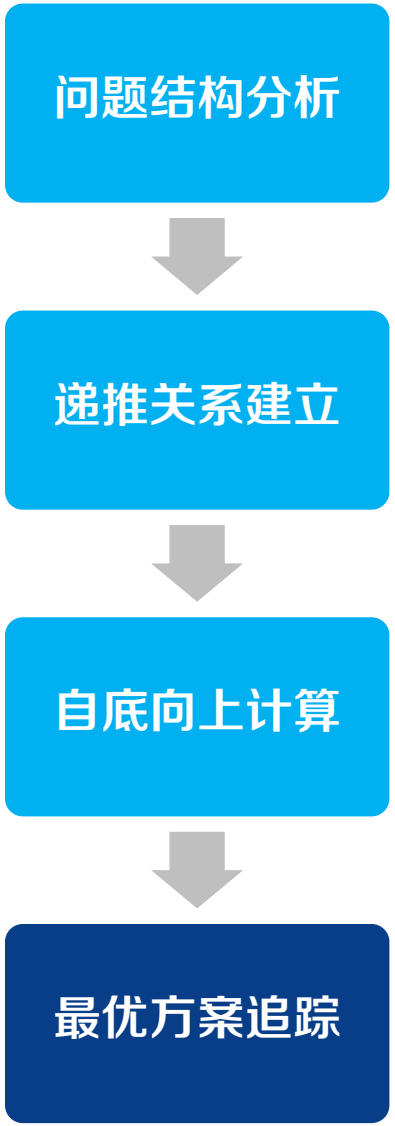




最优方案追踪：记录决策过程

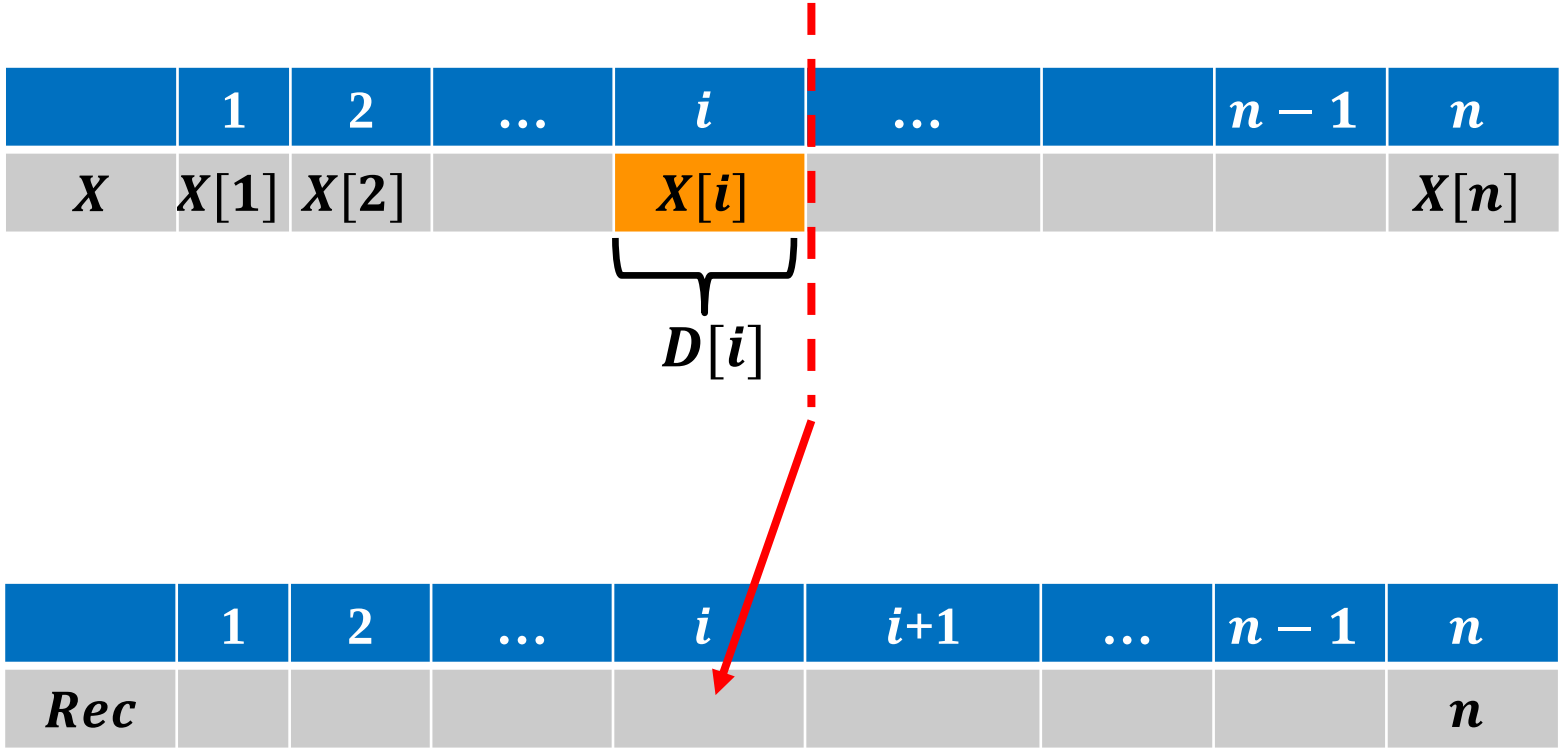
- 构造追踪数组$Rec[1..n]$
  - 情况1：结尾相同
    - $Rec[i] = Rec[i+1]$

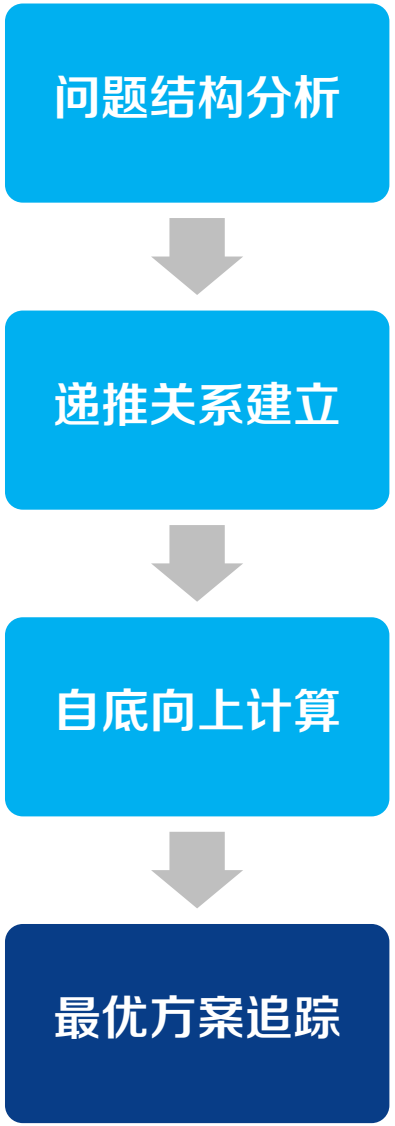




最优方案追踪：记录决策过程

- 构造追踪数组$Rec[1..n]$
  - 情况2：结尾不同
    - $Rec[i]=i$





最优方案追踪：输出最优方案

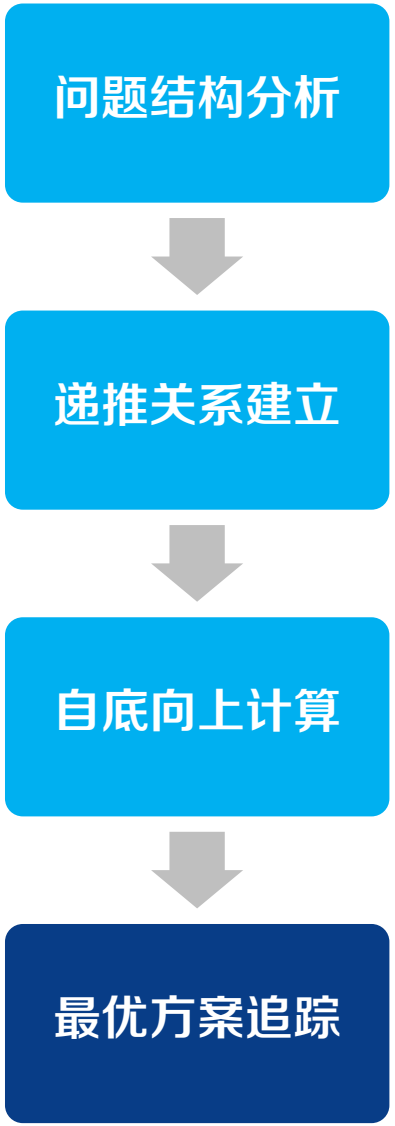
- 从子问题中查找最优解

	1	2	…	…	i		$n-1$	n
X	$X[1]$	$X[2]$			$X[i]$			$X[n]$

已求得

	1	2	…		i		$n-1$	n
D	$D[1]$	$D[2]$			$D[i]$			$D[n]$

	1	2	…		i	…	$n-1$	n
Rec								n



最优方案追踪：输出最优方案

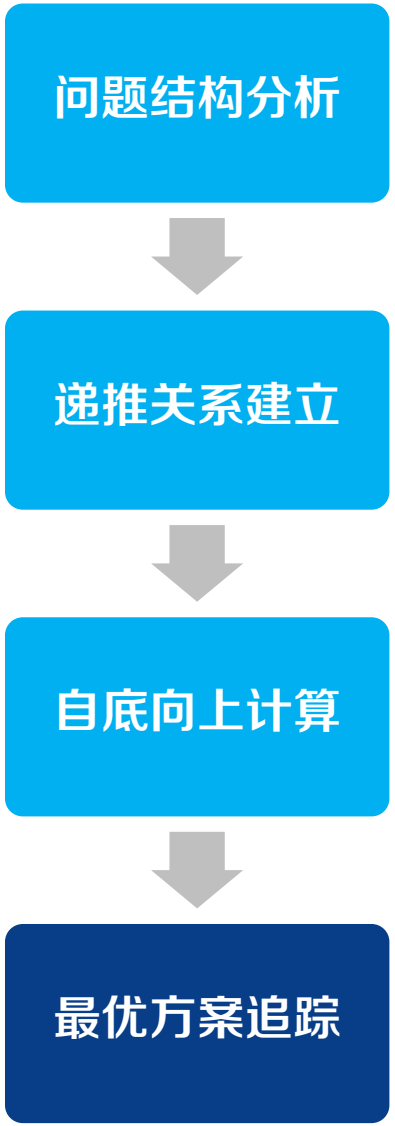
- 从子问题中查找最优解

	1	2	…	…	i		$n-1$	n
X	$X[1]$	$X[2]$			$X[i]$			$X[n]$

	1	2	…		i		$n-1$	n
D	$D[1]$	$D[2]$			$D[i]$			$D[n]$

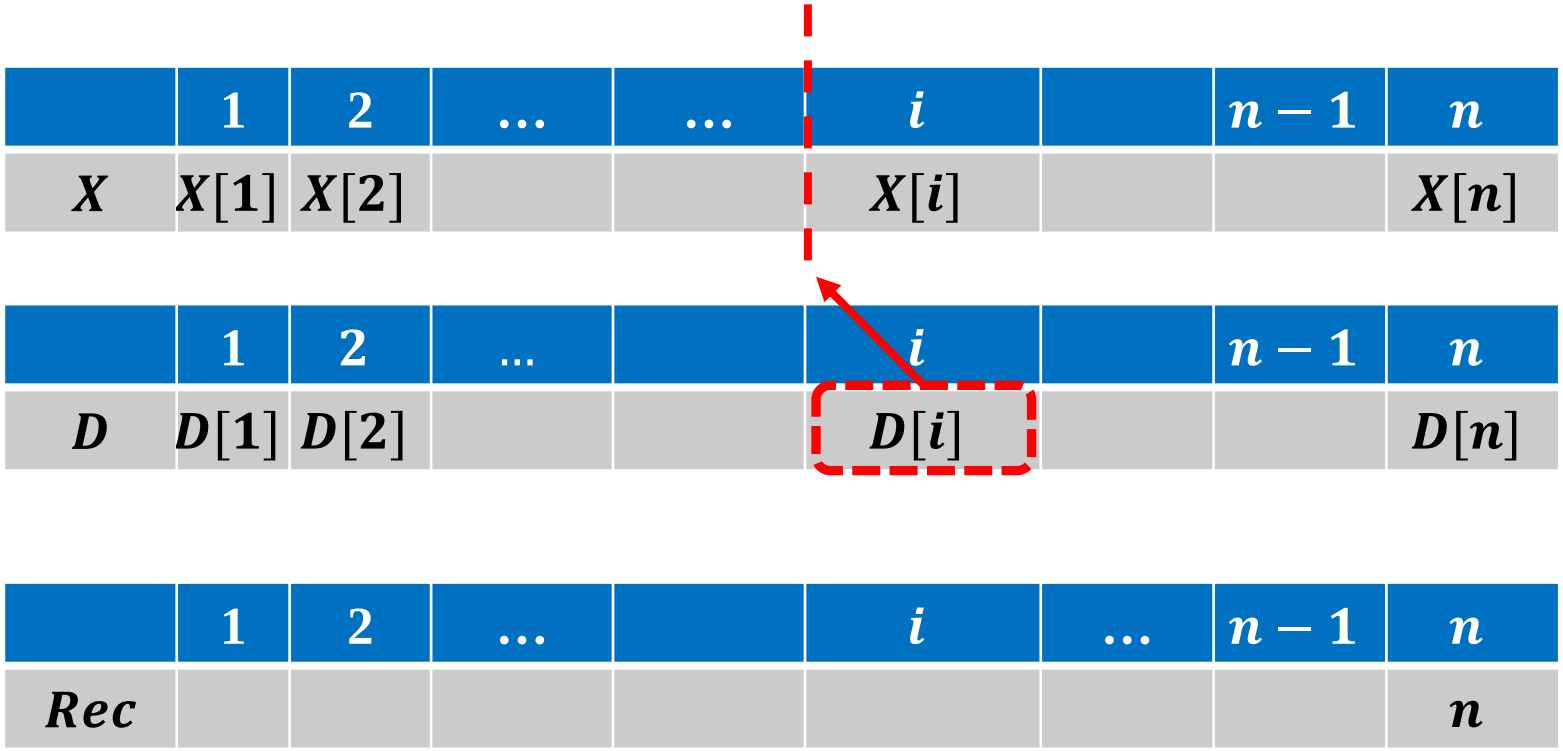
最优解

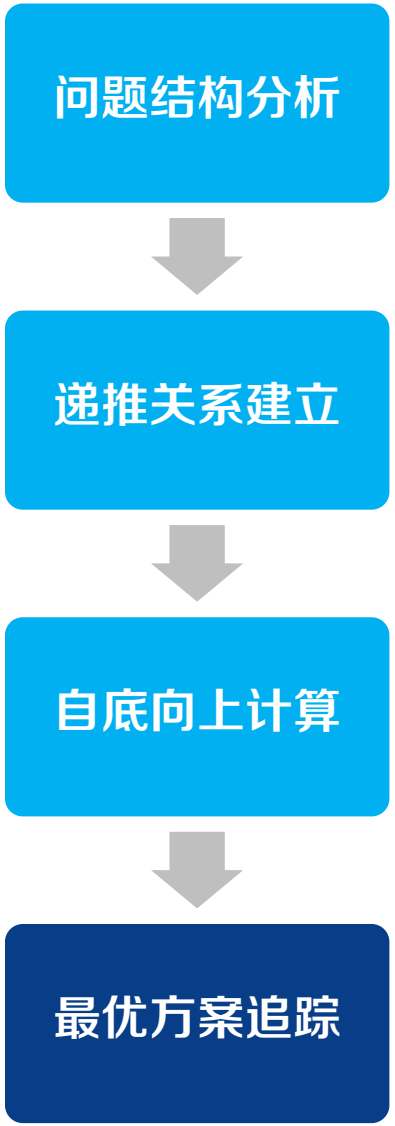
	1	2	…		i	…	$n-1$	n
Rec								n



最优方案追踪：输出最优方案

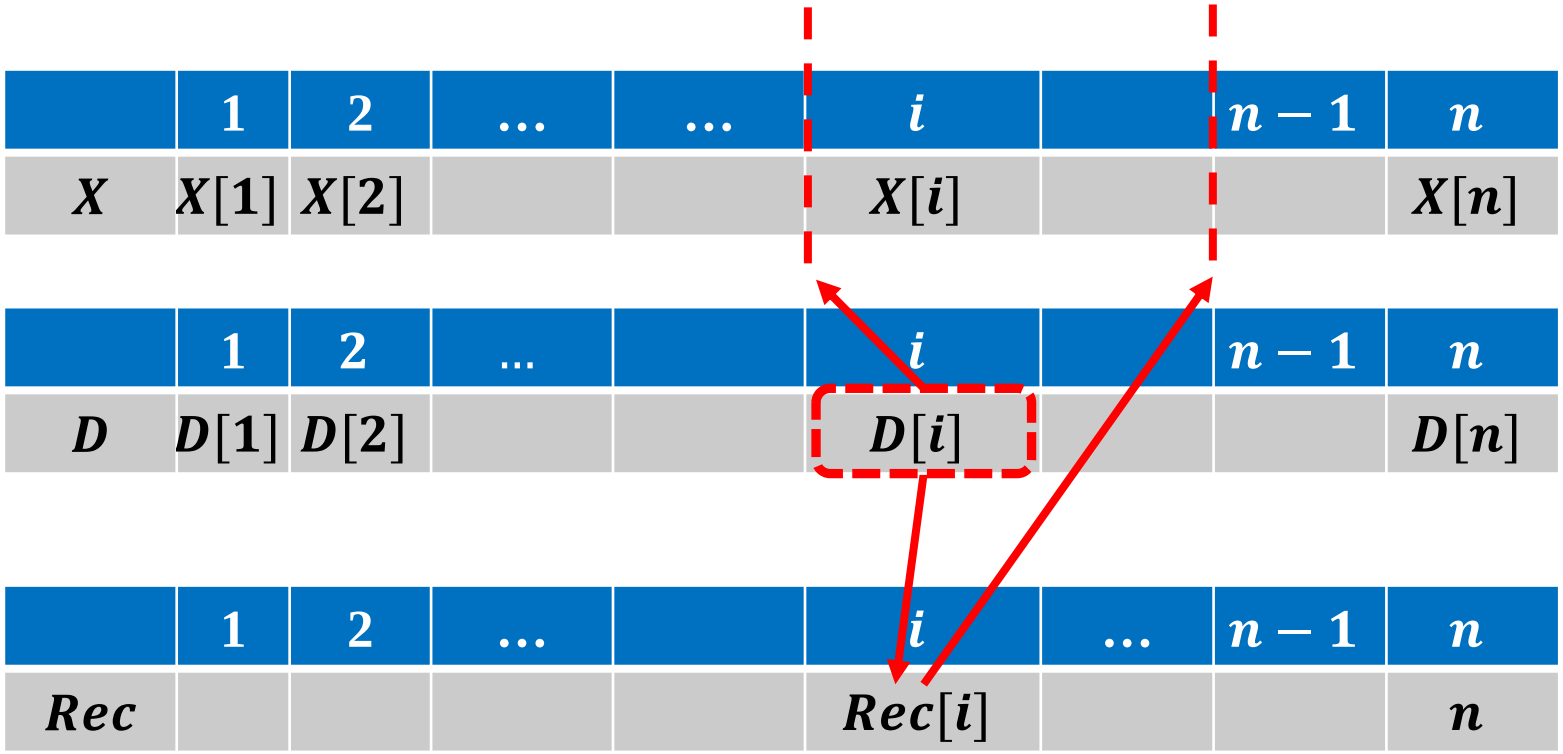
- 从子问题中查找最优解
- 最大子数组开头位置：i

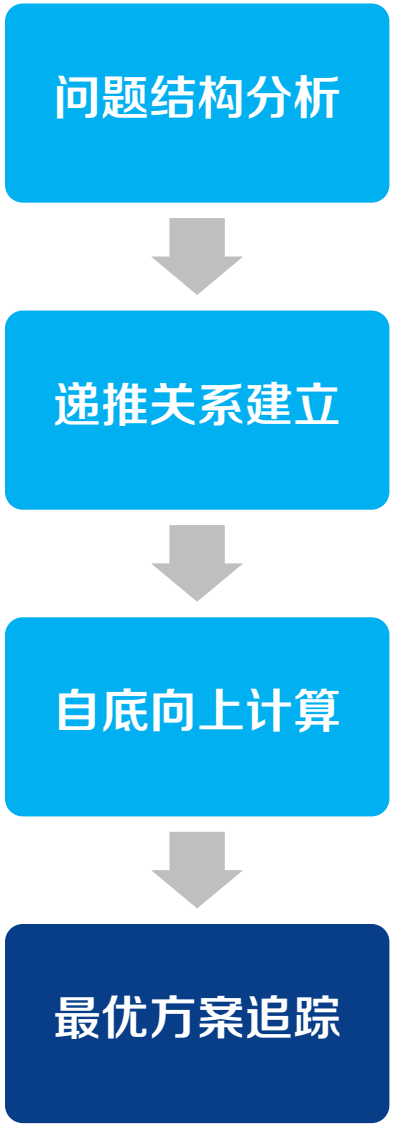




最优方案追踪：输出最优方案

- 从子问题中查找最优解
- 最大子数组开头位置：i
- 最大子数组结尾位置：$Rec[i]$





算法实例

X	i	1	2	3	4	5	6	7	8	9	10	11	12
		1	-2	4	5	-2	8	3	-2	6	3	7	-1

D	i	1	2	3	4	5	6	7	8	9	10	11	12

Rec	i	1	2	3	4	5	6	7	8	9	10	11	12

算法实例

X

i	1	2	3	4	5	6	7	8	9	10	11	12
	1	-2	4	5	-2	8	3	-2	6	3	7	-1

D

i	1	2	3	4	5	6	7	8	9	10	11	12
												-1

初始化

Rec

i	1	2	3	4	5	6	7	8	9	10	11	12
												12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D											7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec												12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D											7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec											11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D										10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec										11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D									16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec									11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D								14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec								11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D							17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \le 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec							11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D						25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec						11	11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D					23	25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec					11	11	11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D				28	23	25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec				11	11	11	11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D			32	28	23	25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec			11	11	11	11	11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D		30	32	28	23	25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec		11	11	11	11	11	11	11	11	11	11	12

算法实例

i	1	2	3	4	5	6	7	8	9	10	11	12
X	1	-2	4	5	-2	8	3	-2	6	3	7	-1

i	1	2	3	4	5	6	7	8	9	10	11	12
D	31	30	32	28	23	25	17	14	16	10	7	-1

$$D[i] = \begin{cases} X[i] + D[i+1], & if\ D[i+1] > 0 \\ X[i], & if\ D[i+1] \leq 0 \end{cases}$$

i	1	2	3	4	5	6	7	8	9	10	11	12
Rec	11	11	11	11	11	11	11	11	11	11	11	12

算法实例

X

i	1	2	3	4	5	6	7	8	9	10	11	12
	1	-2	4	5	-2	8	3	-2	6	3	7	-1

D

i	1	2	3	4	5	6	7	8	9	10	11	12
	31	30	32	28	23	25	17	14	16	10	7	-1

最优解

Rec

i	1	2	3	4	5	6	7	8	9	10	11	12
	11	11	11	11	11	11	11	11	11	11	11	12

算法实例

X	i	1	2	3	4	5	6	7	8	9	10	11	12
		1	-2	4	5	-2	8	3	-2	6	3	7	-1

D	i	1	2	3	4	5	6	7	8	9	10	11	12
		31	30	32	28	23	25	17	14	16	10	7	-1

最优解

Rec	i	1	2	3	4	5	6	7	8	9	10	11	12
		11	11	11	11	11	11	11	11	11	11	11	12

终止位置

算法实例

X

i	1	2	3	4	5	6	7	8	9	10	11	12
	1	-2	4	5	-2	8	3	-2	6	3	7	-1

D

i	1	2	3	4	5	6	7	8	9	10	11	12
	31	30	32	28	23	25	17	14	16	10	7	-1

最优解

Rec

i	1	2	3	4	5	6	7	8	9	10	11	12
	11	11	11	11	11	11	11	11	11	11	11	12

终止位置

$S = \{4, 5, -2, 8, 3, -2, 6, 3, 7\}$

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$

初始化

//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$
//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

自底向上计算

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$
//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

情况1：$D[i+1] > 0$

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$
//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

记录子问题结果与决策

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$
//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

情况2： $D[i+1] \leq 0$

动态规划：伪代码

- **Max-Continuous-Subarray-DP(X, n)**

```
//查找解
S_max ← D[1]
for i ← 2 to n do
    if S_max < D[i] then
        S_max ← D[i]
        l ← i
        r ← Rec[i]
    end
end
return S_max, l, r
```

时间复杂度分析

输入: 数组X, 数组长度n
输出: 最大子数组和S_{max},子数组起止位置l, r
新建一维数组$D[1..n]$和$Rec[1..n]$
//初始化
$D[n] \leftarrow X[n]$
$Rec[n] \leftarrow n$
//动态规划
for $i \leftarrow n-1$ *to* 1 **do**
  if $D[i+1] > 0$ **then**
    $D[i] \leftarrow X[i] + D[i+1]$
    $Rec[i] \leftarrow Rec[i+1]$
  end
  else
    $D[i] \leftarrow X[i]$
    $Rec[i] \leftarrow i$
  end
end

$O(n)$

时间复杂度：$O(n)$