Design and Analysis of Algorithms

Part II: Dynamic Programming

Lecture 10: 0-1 Knapsack

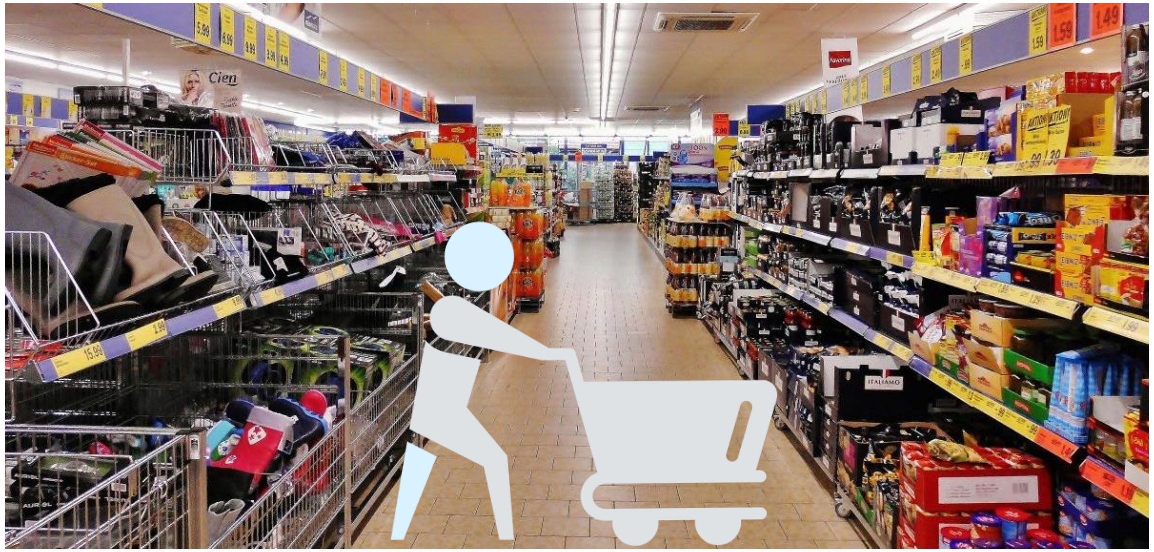
童咏昕

北京航空航天大学
计算机学院

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
- 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

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

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	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
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		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

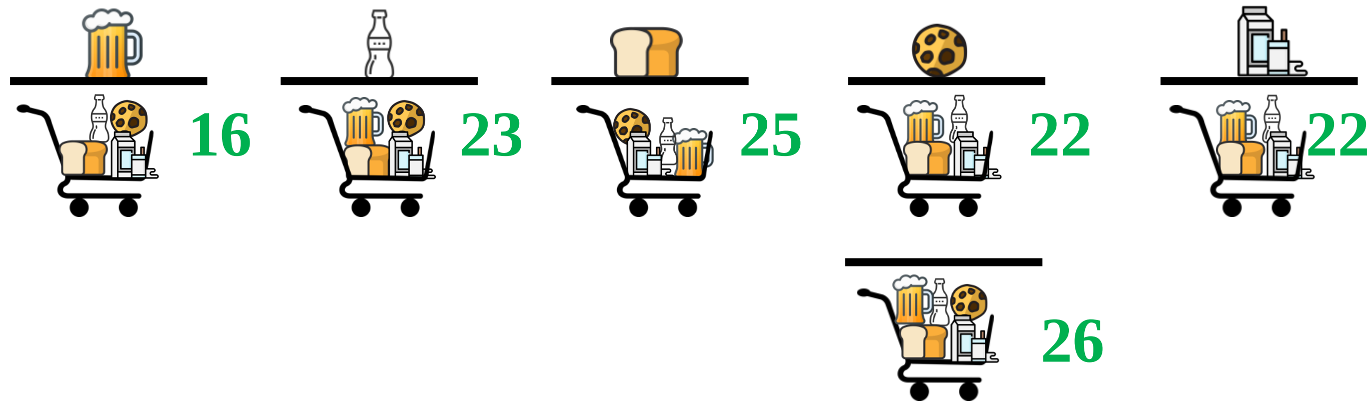
商品	价格	体积	比值
啤酒	24	10	2.4
饼干	9	4	2.25
牛奶	9	4	2.25
面包	10	5	2
汽水	2	3	0.67

	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		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（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

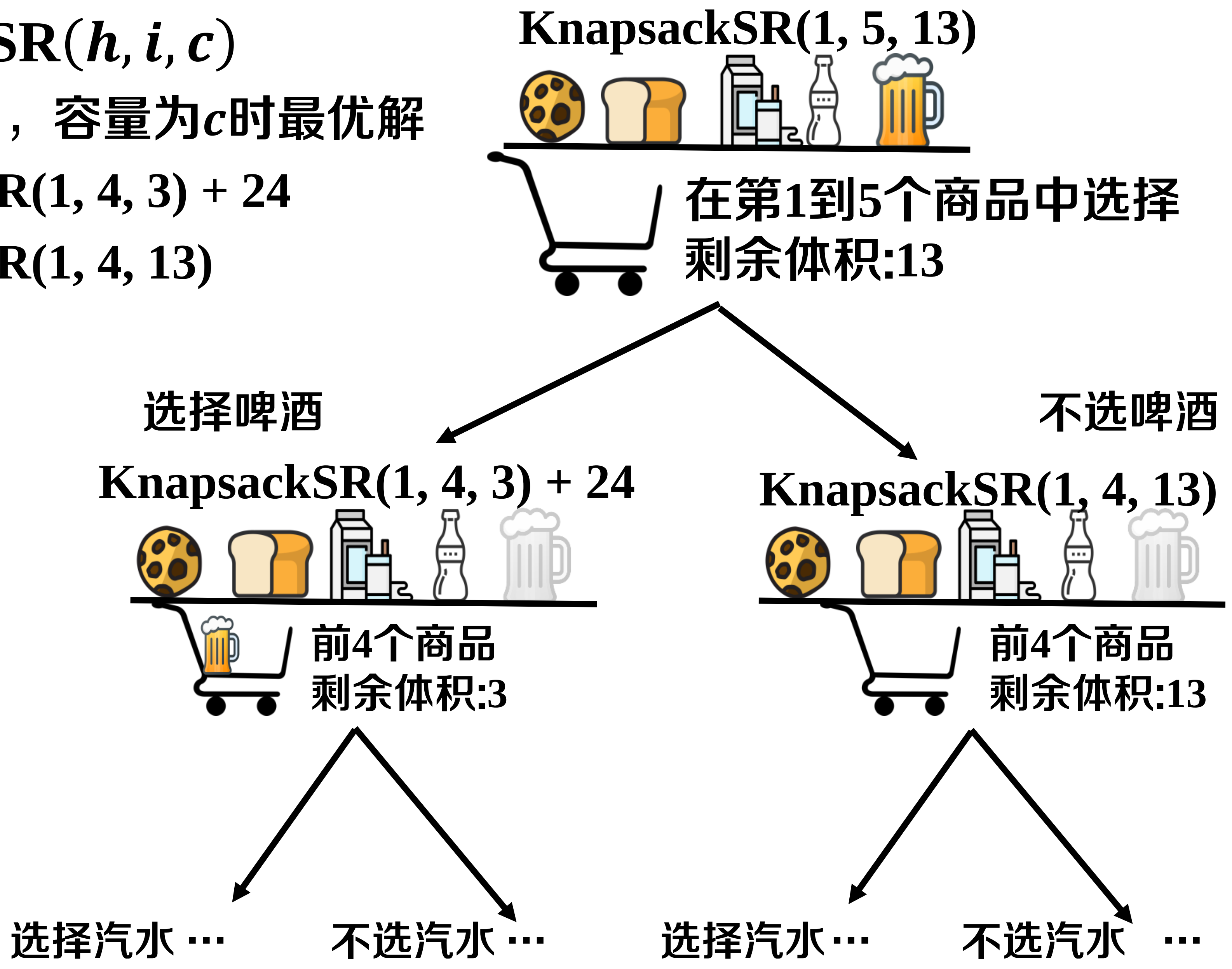
蛮力枚举

- 枚举所有商品组合（共$2^n - 1$种情况），检查体积约束

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

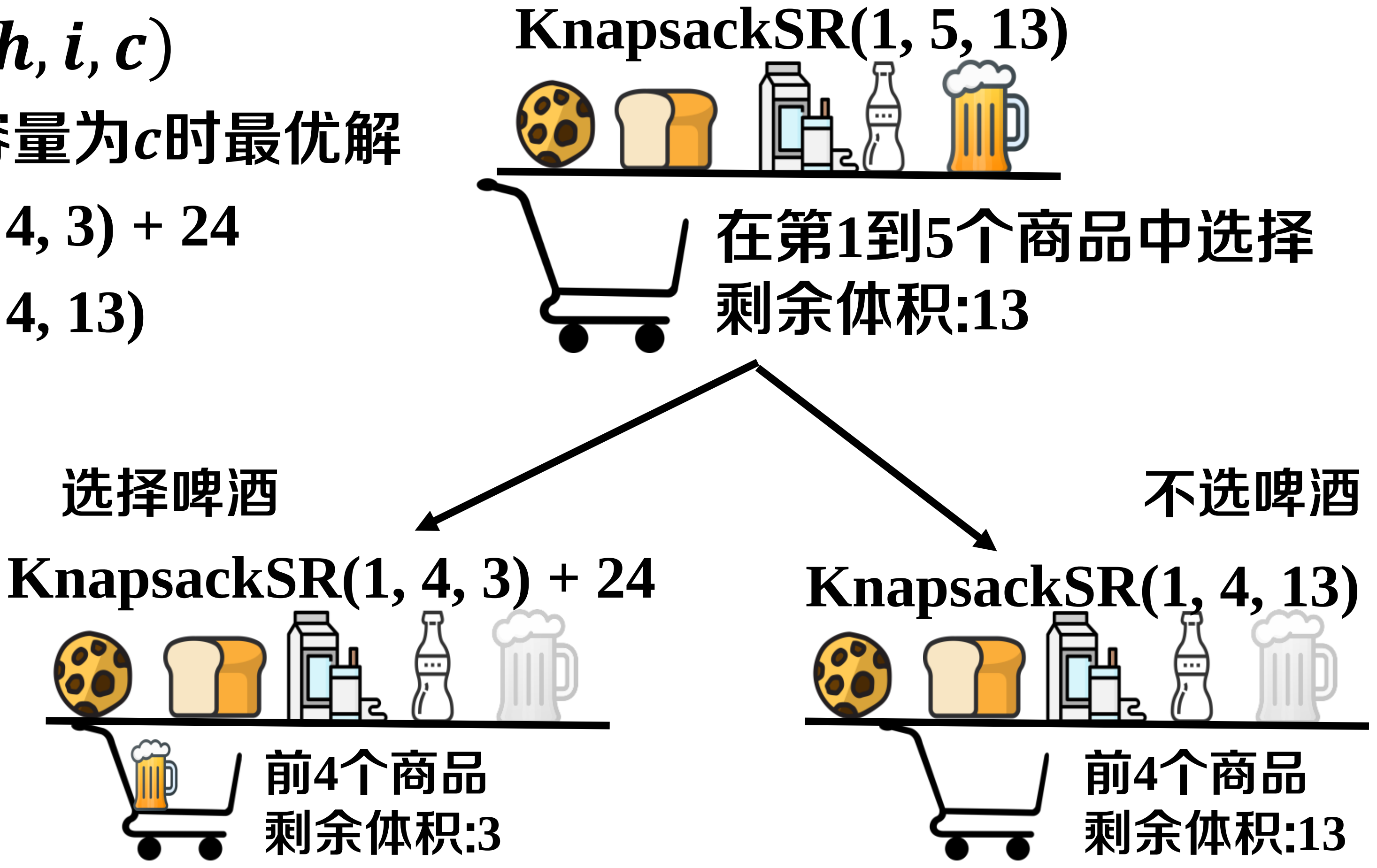
- 递归函数：KnapsackSR(h, i, c)
  - 在第h个到第i个商品中，容量为c时最优解
  - 选择啤酒：KnapsackSR(1, 4, 3) + 24
  - 不选啤酒：KnapsackSR(1, 4, 13)



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

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

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

蛮力枚举：伪代码

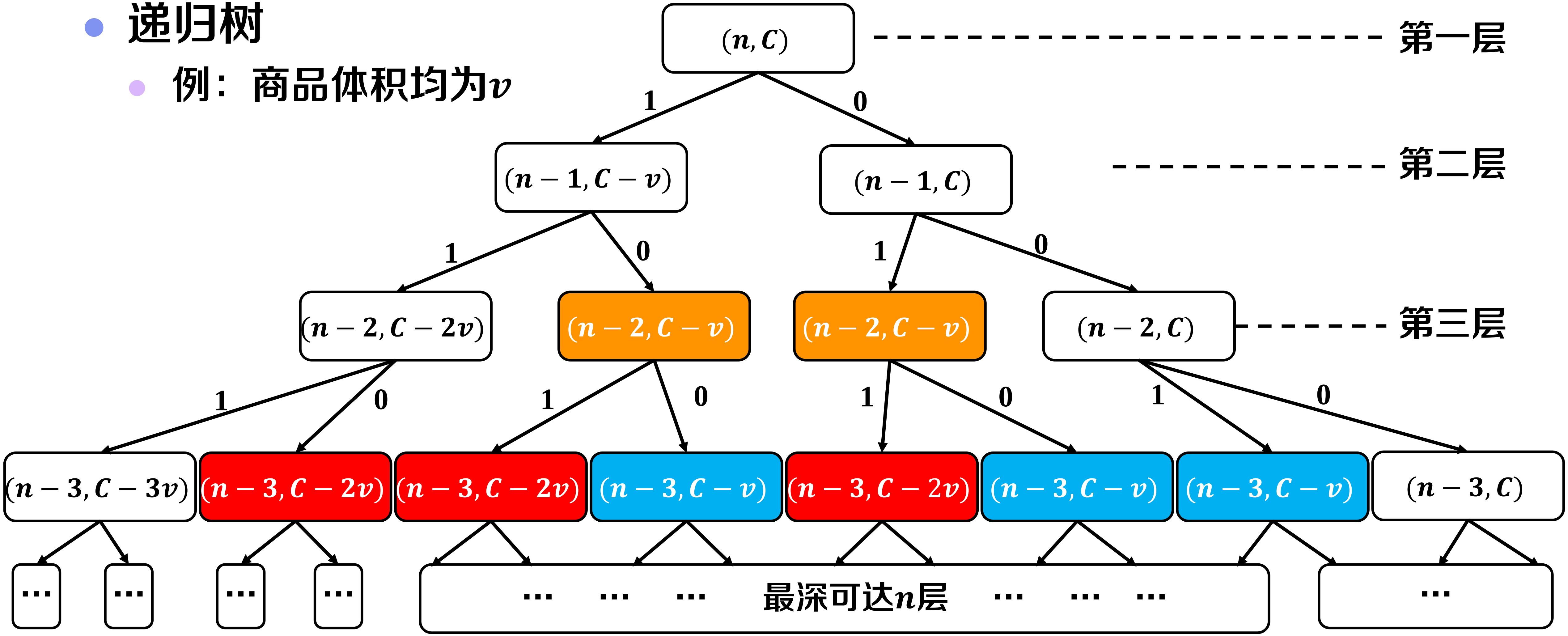
- **KnapsackSR(i, c)：前i个商品中，容量为c时最优解**

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 $-\infty$
end
if $i \leq 0$ **then**
  return 0
end
$P_1 \leftarrow \mathrm{KnapsackSR}(i-1, c-v_i)$
$P_2 \leftarrow \mathrm{KnapsackSR}(i-1, c)$
$P \leftarrow \max\{P_1 + p_i, P_2\}$
return P

蛮力枚举：复杂度

- 递归树
  - 例：商品体积均为v

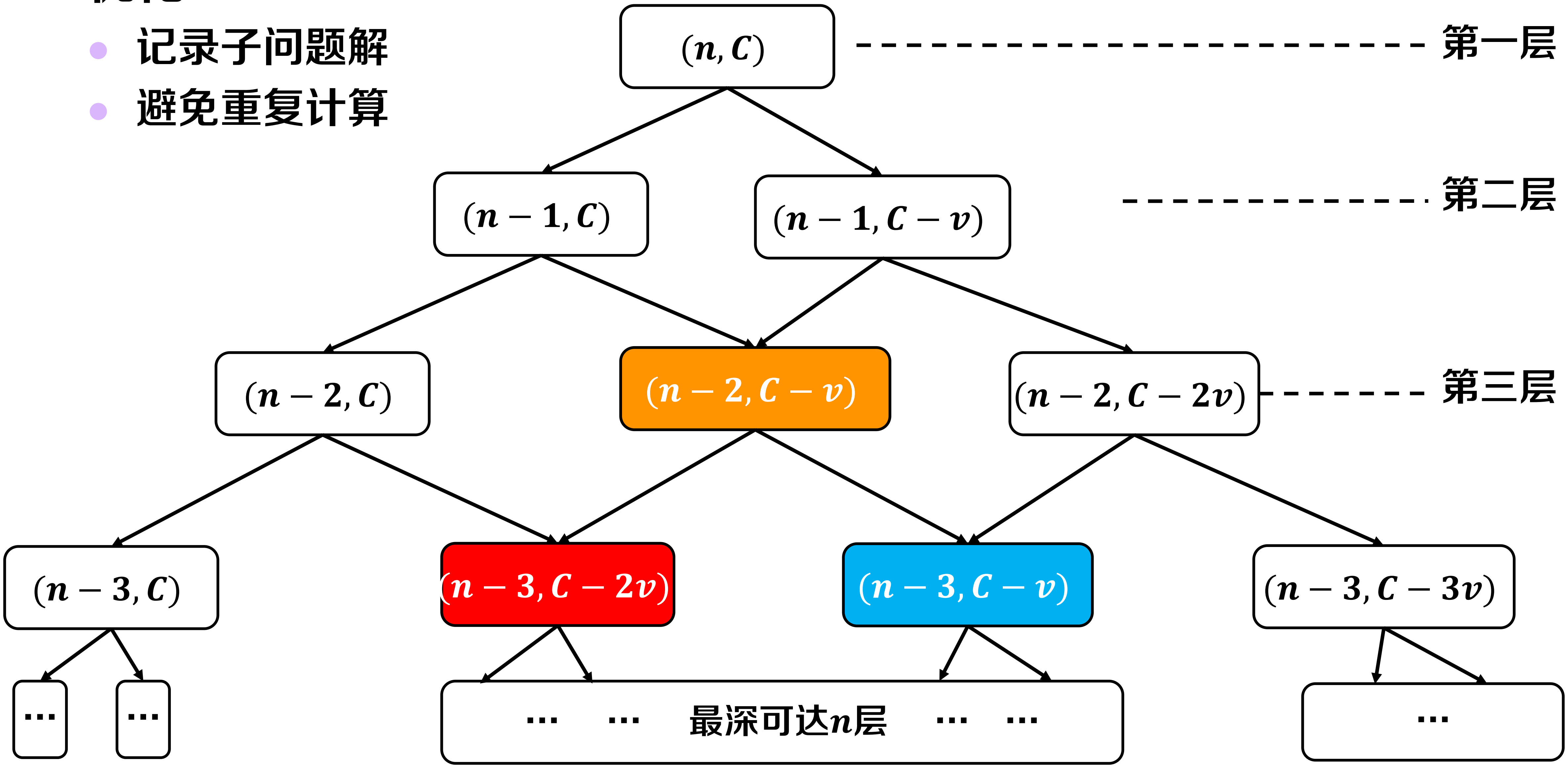


- 重复求解大量子问题 $O(2^n)$

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)**

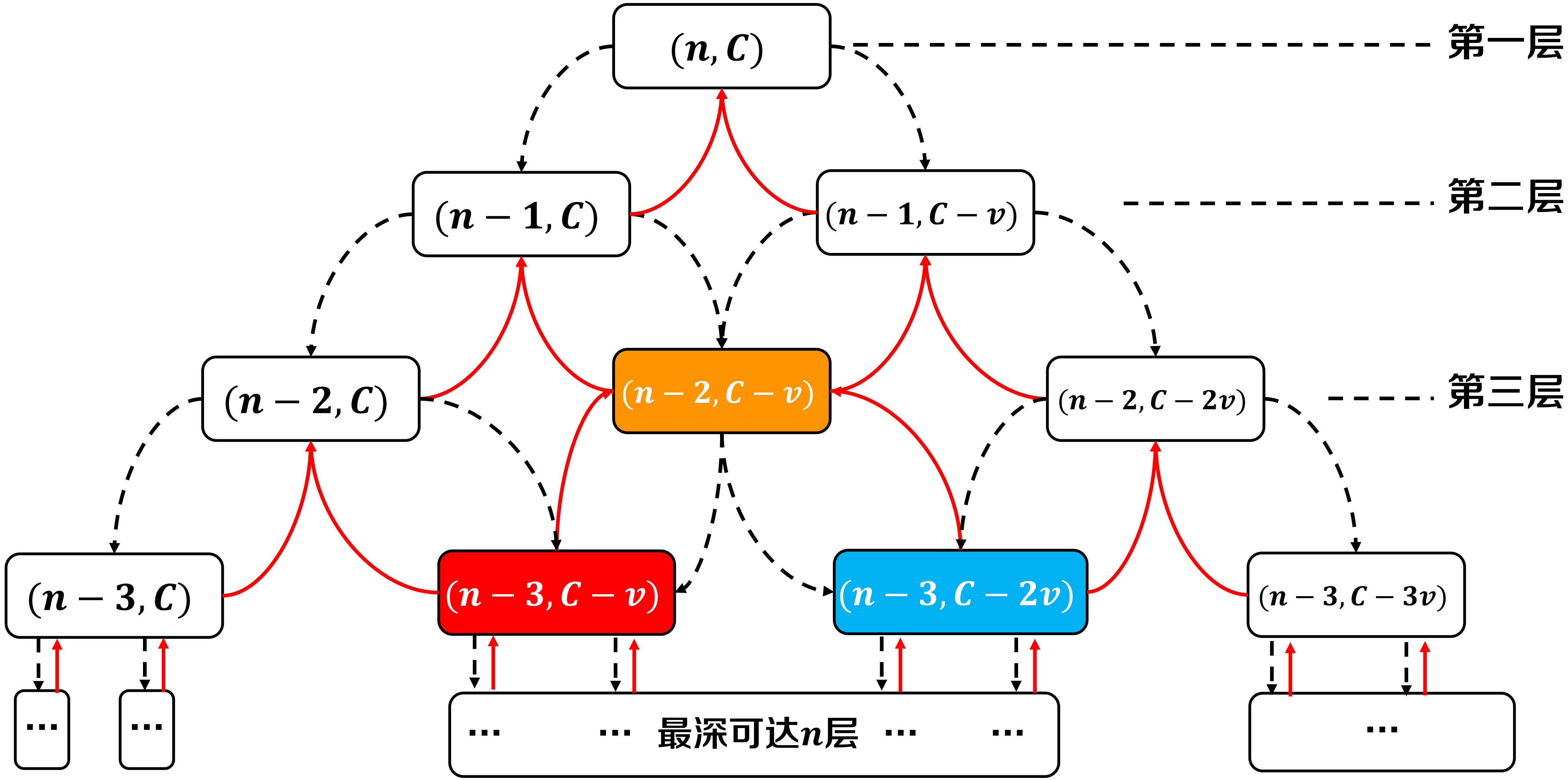
```
输入: 商品集合{1, ..., i}, 背包容量c
输出: 最大总价格P
if c < 0 then
 |  return −∞
end
if i ≤ 0 then
 |  return 0
end
if P[i, c] ≠ NULL then
 |  return P[i, c]
end
P1 ←KnapsackMR(i − 1, c − v_i)
P2 ←KnapsackMR(i − 1, c)
P ← max{P1 + p_i, P2}
P[i, c] ← P
return P
```

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



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	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	[illegible]	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	$\boxed{28}$

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						$Rec[i,c]=0$		
…	0			$Rec[i,c]=1$		$P[i,c]$			
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						$Rec[i,c]=0$		
…	0			$Rec[i,c]=1$		$P[i,c]$			
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

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	[illegible]	[illegible]	[illegible]	[illegible]	0
4	0	0	0	0	0	1	1	0	1	1	0	0	[illegible]	0
5	0	0	0	0	0	0	0	0	1	0	0	0	0	$\boxed{1}$

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	[illegible]	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	[illegible]	1	0	1	1	0	0	0	0
5	0	0	0	0	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	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						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 $i \leftarrow 0$ *to* C **do**
  $P[0, i] \leftarrow 0$
end
for $i \leftarrow 0$ *to* n **do**
  $P[i, 0] \leftarrow 0$
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
```

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]
```

时间复杂度分析

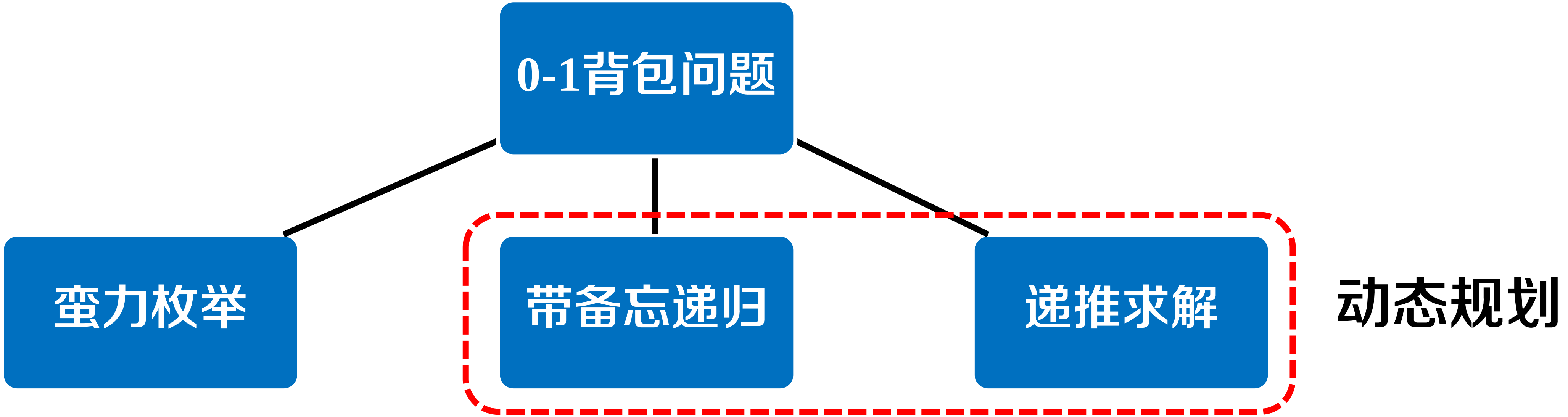
```
//求解表格
for i ← 1 to n do
    for c ← 1 to C do
        if (v[i] ≤ c) and
        (p[i] + P[i − 1, c − v[i]] > P[i − 1, c]) then
            P[i, c] ← p[i] + P[i − 1, c − v[i]]
            Rec[i, c] ← 1
        end
        else
            P[i, c] ← P[i − 1, c]
            Rec[i, c] ← 0
        end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

$O(C)$

$O(n \cdot C)$

时间复杂度：$O(n \cdot C)$

动态规划



	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

问题结构分析

↓

递推关系建立

↓

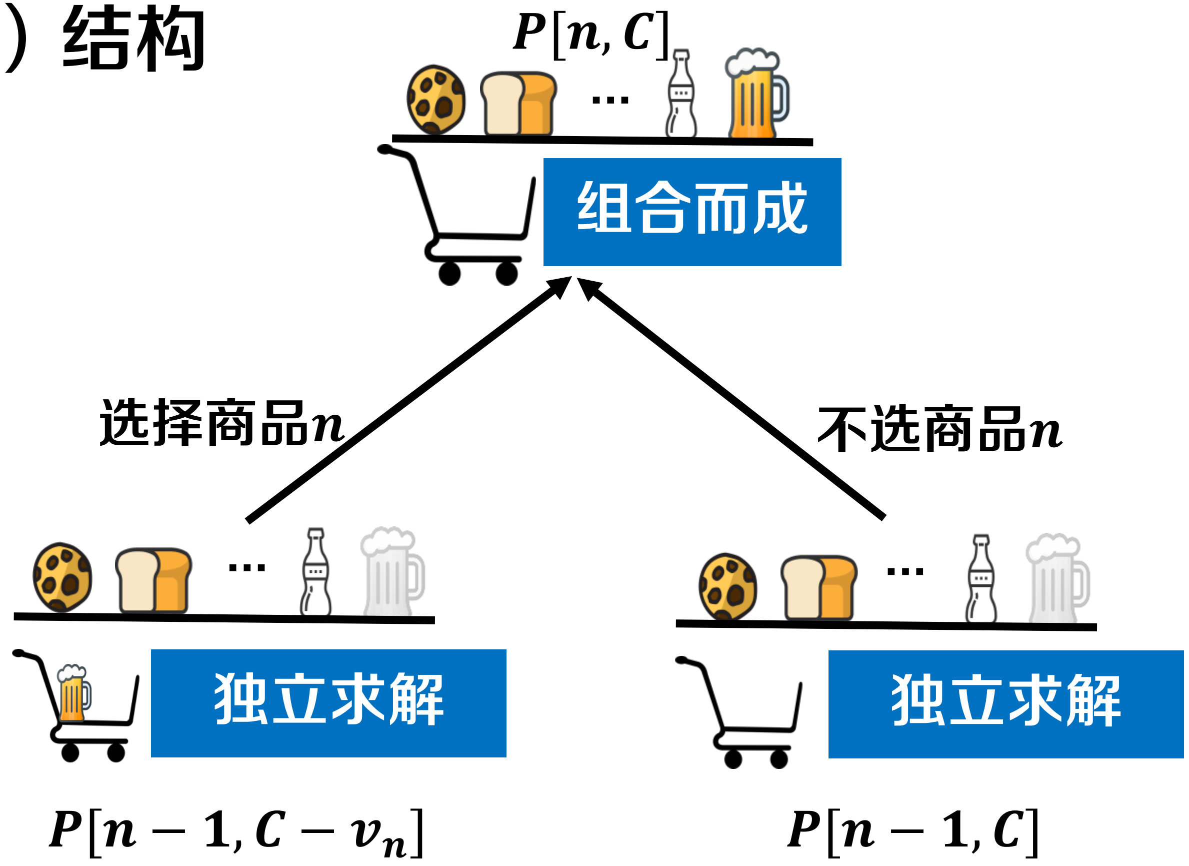
自底向上计算

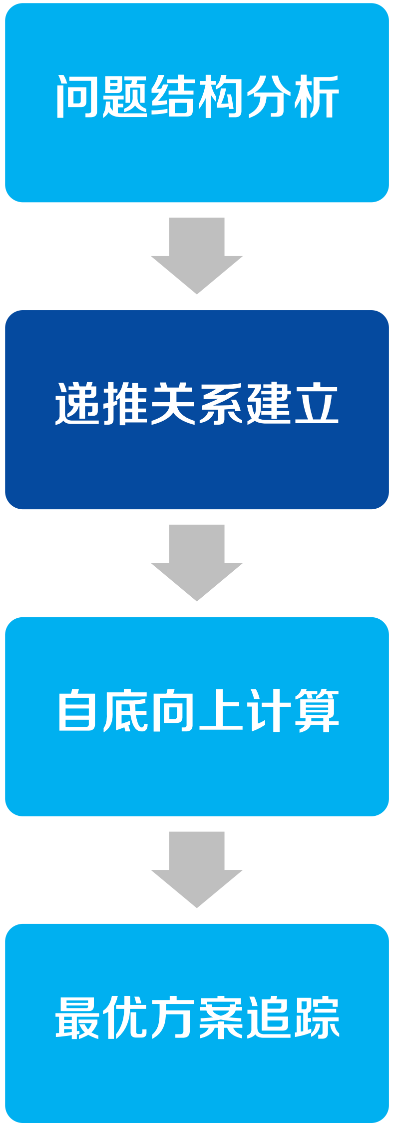
↓

最优方案追踪

动态规划：一般步骤

- 分析最优（子）结构





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

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

- 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								

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$P[i-1,c]$

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

$Rec[i,c]=1$

$Rec[i,c]=0$

$Rec[i,c]=1$

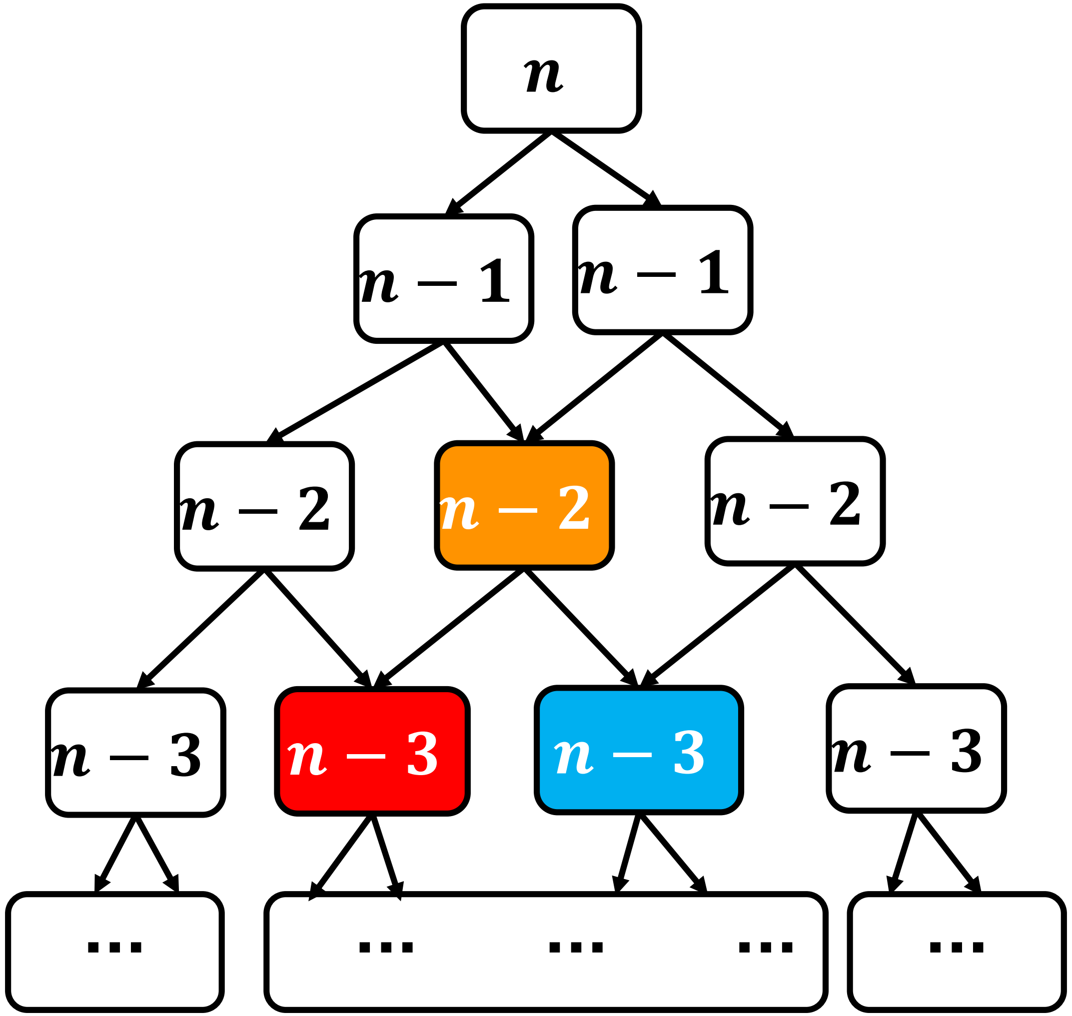
问题结构分析

递推关系建立

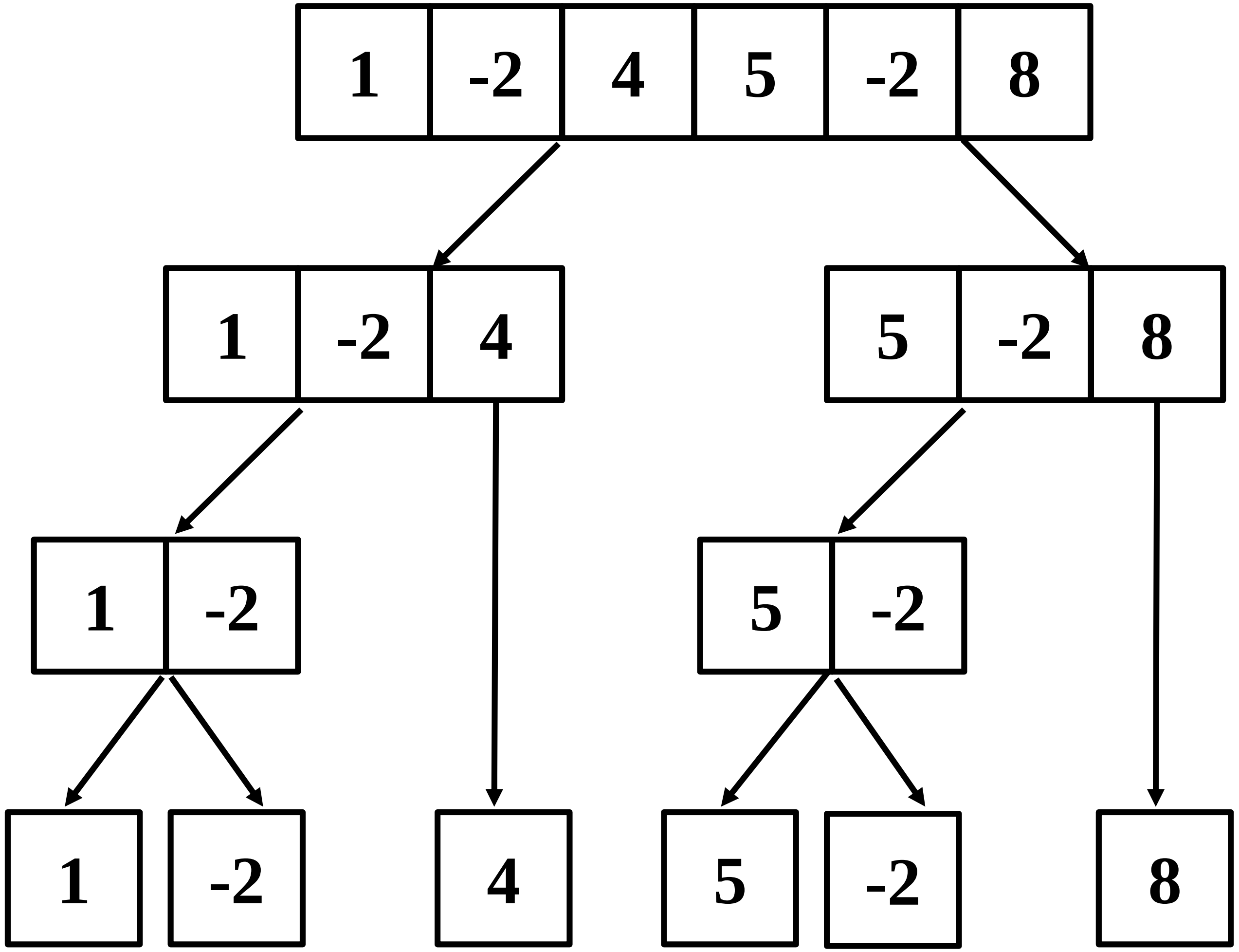
自底向上计算

最优方案追踪

方法比较



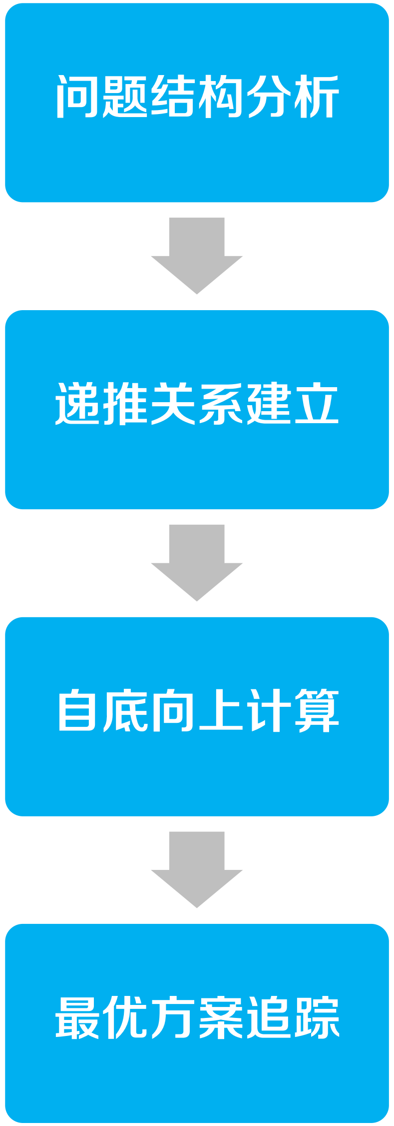
动态规划：重叠子问题



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



谢谢

BDA

Beihang University