算法分析与设计

Analysis and Design of Algorithm

第9次课

要点回顾

- 动态规划算法
  - 动态规划法的基本思想
  - 动态规划法的基本步骤
- 动态规划法的实例
  - 最长公共子序列
  - 最大子段和

图像压缩及其应用

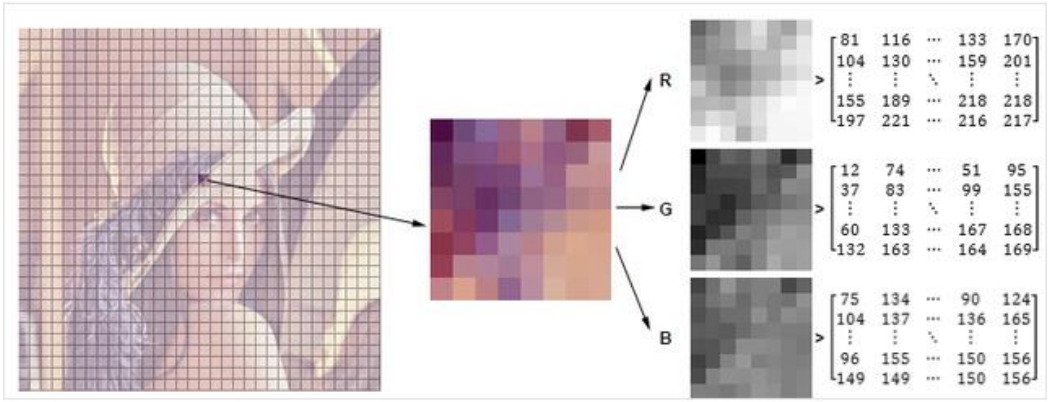
尺寸：1024×768

未压缩大小（bmp格式）
2.25MB

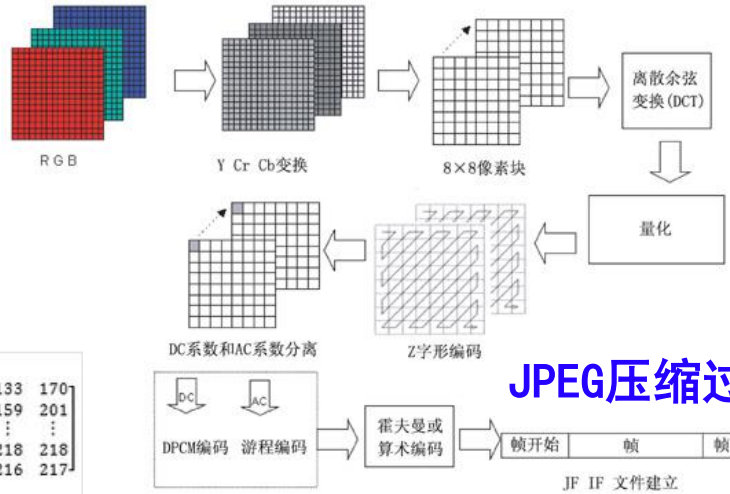
压缩后大小（jpg格式）
202KB

图像压缩步骤

1、将原始图像分解成大量小块图片



2、转化为成Red、Green、Blue三种颜色的矩阵



JPEG压缩过程

3、执行特定的压缩算法

图像的变位压缩存储

- 图像的表示
  - 以像素点灰度值序列$\{p_1, p_2, \cdots, p_n\}$表示
  - $0 \leq p_i \leq 255$
  - 需要8位表示一个像素值

小鸭子.bmp

Address	0	1	2	3	4	5	6
00000000	42	4d	40	d9	04	00	00
00000010	00	00	43	02	00	00	22
00000020	00	00	00	00	00	00	85
00000030	00	00	00	00	00	00	00
00000040	02	00	03	03	03	00	04
00000050	06	00	07	07	07	00	08
00000060	0a	00	0b	0b	0b	00	0c
00000070	0e	00	0f	0f	0f	00	10

图像的变位压缩存储

- 图像的表示
  - 以像素点灰度值序列$\{p_1, p_2, \cdots, p_n\}$表示
  - $0 \leq p_i \leq 255$
  - 需要8位表示一个像素值

- **变位压缩的思想：**灰度值序列分段都用小于8位的数字表示一个灰度值

图像的变位压缩存储

- **变位压缩的思想：** 灰度值序列分段都用小于8位的数字表示一个灰度值

2	3	2	37	40	14	9

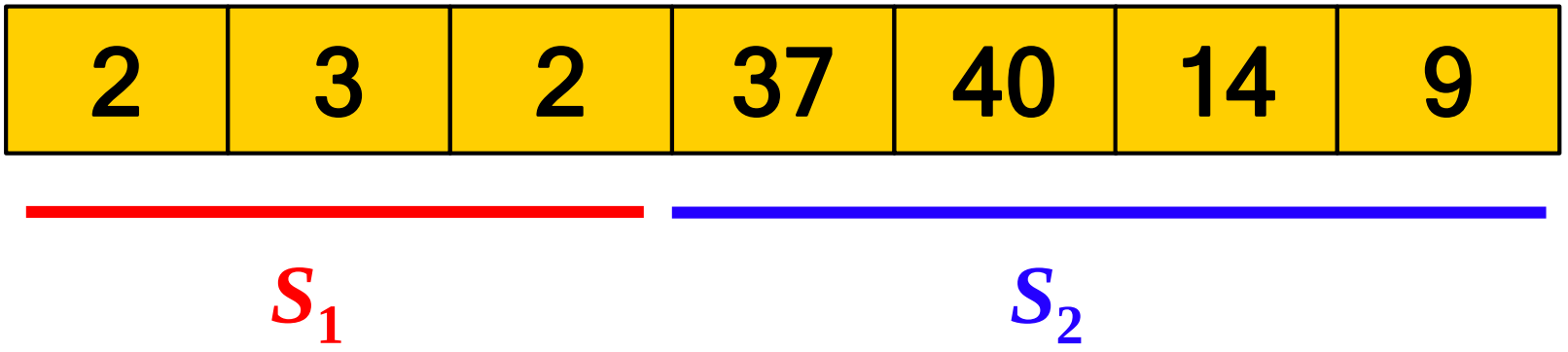
2 bits　2 bits　2 bits　6 bits　6 bits　4 bits　4 bits

压缩前占用比特数：8×7=56

实际需要比特数：2+2+2+6+6+4+4=26

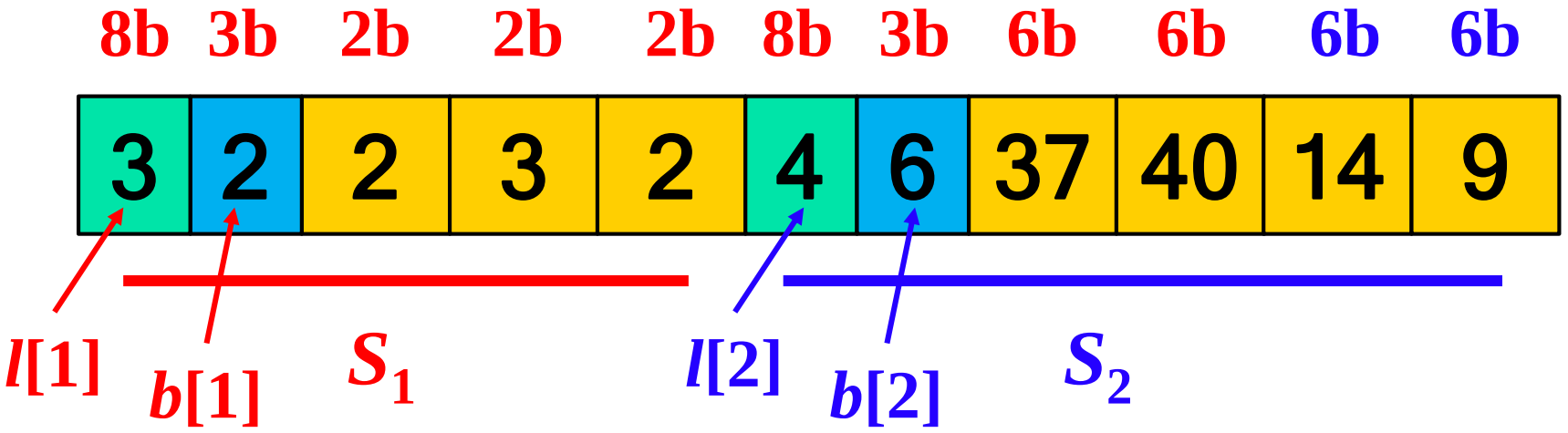
图像变位压缩-解决思路

- 像素点序列{p_1, p_2, … , p_n}分割成m个连续段S_1, S_2, … , S_m



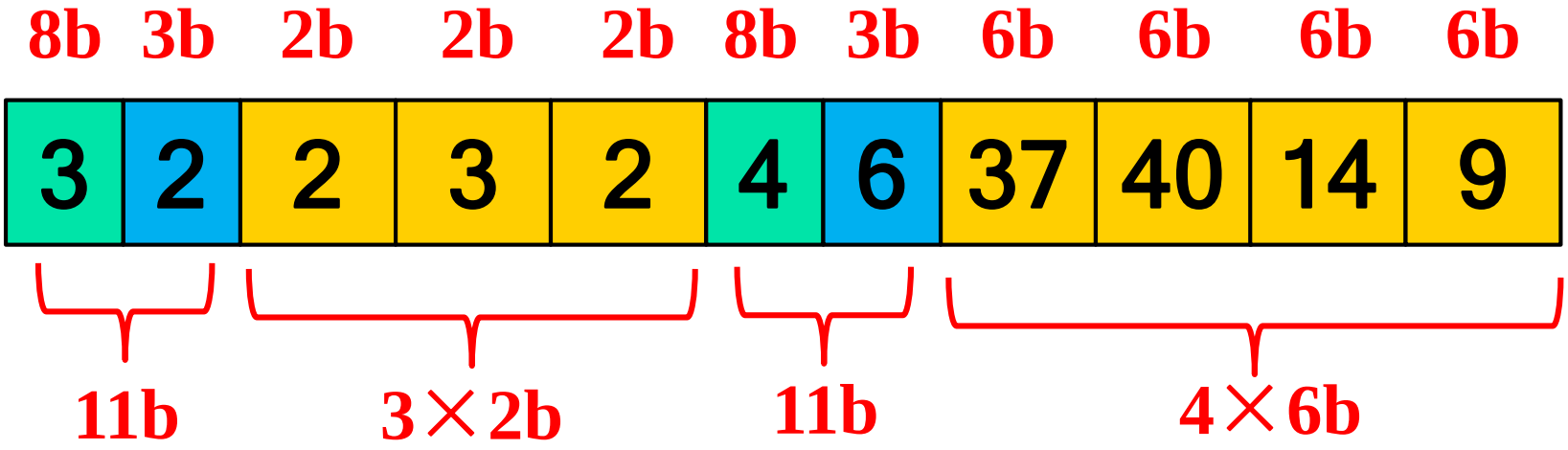
图像变位压缩-解决思路

- 像素点序列{p_1, p_2, … , p_n}分割成m个连续段S_1, S_2, … , S_m
- 第i个像素端S_i中($1 \le i \le m$)由$l[i]$个像素，且该段中每个像素都只用$b[i]$位表示



图像变位压缩-问题定义

- 此时有$h_i \leqslant b[i] \leqslant 8$，所以需要用3位表示$b[i]$；如果限制$1 \leq l[i] \leq 255$，则需要用8位表示$l[i]$。
- 分段S_i需要占用$l[i]*b[i]+11$位
- 存储整张图片$\{p_1,\ldots,p_n\}$共需 $\sum_{i=1}^{m} l[i]*b[i]+11m$位



图像变位压缩-问题定义

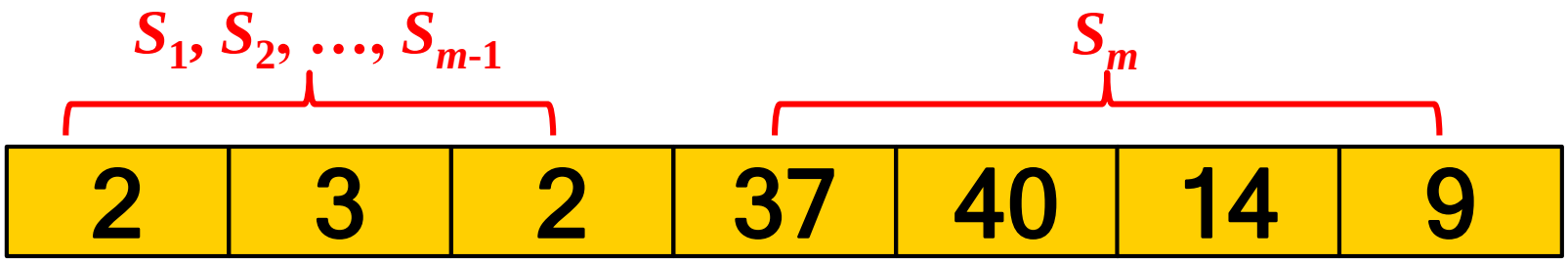
- 此时有$h_i \leqslant b[i] \leqslant 8$，所以需要用3位表示$b[i]$；如果限制$1 \leq l[i] \leq 255$，则需要用8位表示$l[i]$。
- 分段S_i需要占用$l[i]*b[i]+11$位
- 存储 整张图片$\{p_1,\ldots,p_n\}$共需 $\sum_{i=1}^{m} l[i]*b[i]+11m$ 位

- 问题：如何确定像素序列$\{p_1,\ldots,p_n\}$的最优分段，使得依此分段所需的存储空间最小

图像变位压缩-最优子结构

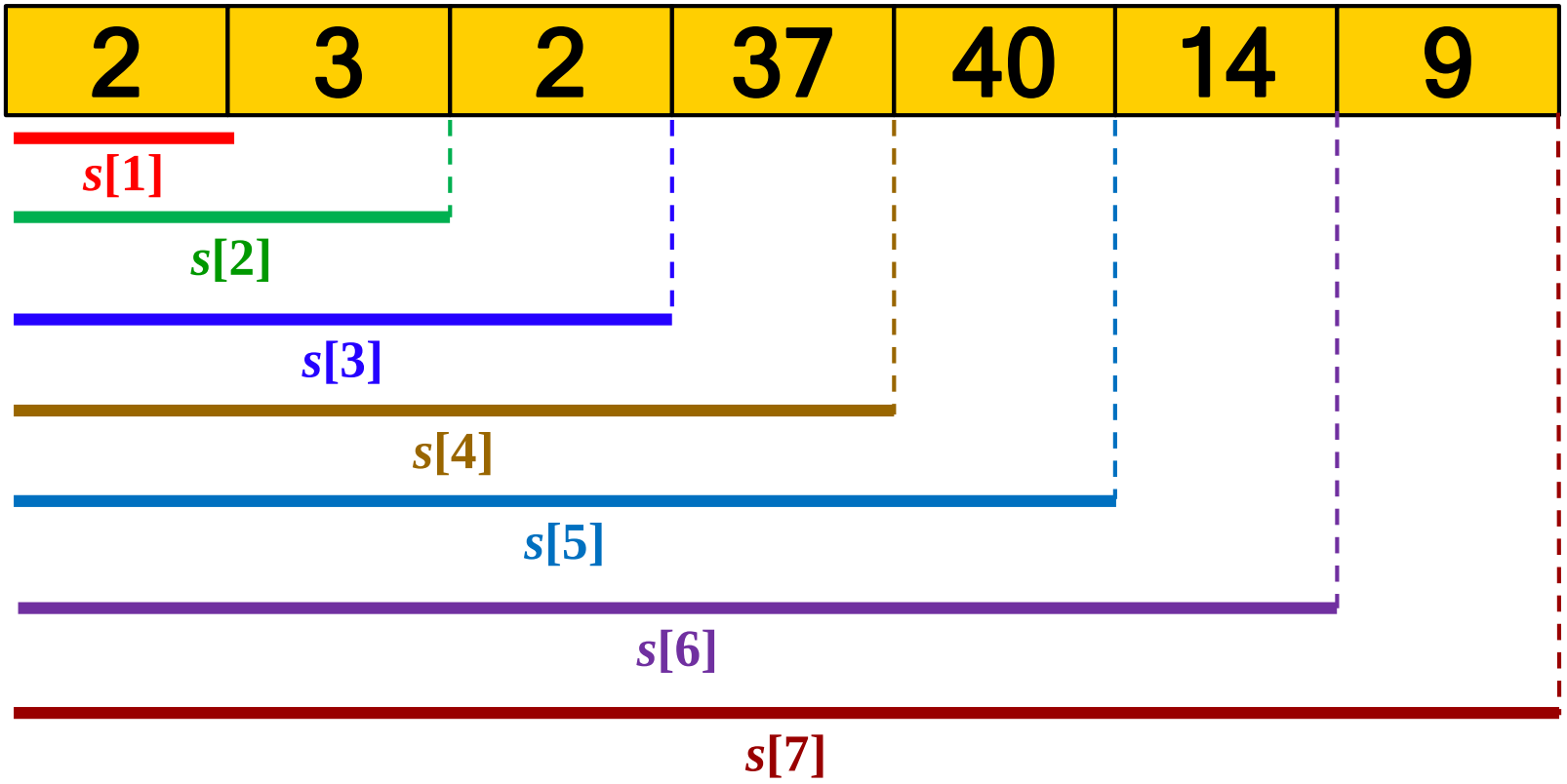
- 设$S_1, S_2, \ldots, S_m$是$\{p_1,p_2,\ldots,p_n\}$的最优分段

- 图象压缩问题满足最优子结构性质
  - S_m是$\{p_{n-l[m]},\ldots,p_n\}$的最优分段
  - $S_1, S_2, \ldots, S_{m-1}$是$\{p_1,\ldots, p_{n-l[m]-1}\}$的最优分段
  - 利用反证法可以证明



图像变位压缩-建立递归关系

- 设$s[i],1\le i\le n$是像素序列$\{p_1,p_2,\ldots,p_n\}$的最优分段所需的存储位数。



图像变位压缩-建立递归关系

- 设$s[i],1\le i\le n$是像素序列$\{p_1,p_2,\ldots,p_n\}$的最优分段所需的存储位数。由最优子结构性质易知：

$$s[i]=\min_{1\le k\le \min\{i,256\}}\{s[i-k]+k*\mathrm{bmax}(i-k+1,i)\}+11$$

$$\mathrm{bmax}(i,j)=\left\lceil \log\left(\max_{i\le k\le j}\{p_k\}+1\right)\right\rceil$$

- **时间复杂度分析：**由于算法compress中对k的循环次数不超256，故对每一个确定的i，可在时间O(1)内完成的计算。因此整个算法所需的计算时间为O(n)。

小结

- 图像压缩的应用
- 图像压缩的动态规划解法
  - 原始问题的分段
  - 最优子结构的性质

背包问题

背包问题及其应用

- **背包问题(Knapsack Problem)**是一种组合优化的NP完全问题。问题可以描述为：给定一组物品，每种物品都有自己的重量和价格，在限定的总重量内，我们如何选择，才能使得物品的总价格最高。

0-1背包问题

- **问题：给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为c。问应如何选择装入背包的物品，使得装入背包中物品的总价值最大?**
- **0-1背包问题是一个特殊的整数规划问题。**

$$\max \sum_{i=1}^{n} v_i x_i$$

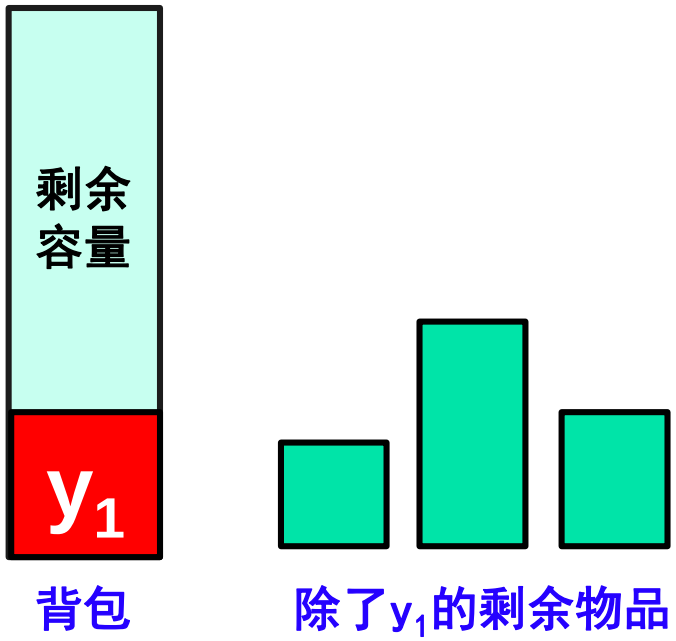
$$\begin{cases} \sum_{i=1}^{n} w_i x_i \le c \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

0-1背包问题—最优子结构

- 设（y_1，y_2，…，y_n）是所给的0-1背包问题的最优解，则（y_2，y_2，…，y_n）是下面子问题的最优解

$$\max \sum_{i=2}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=2}^{n} w_i x_i \le c - w_i y_1 \\ x_i \in \{0,1\}, 2 \le i \le n \end{cases}$$



- **证明：反证法**

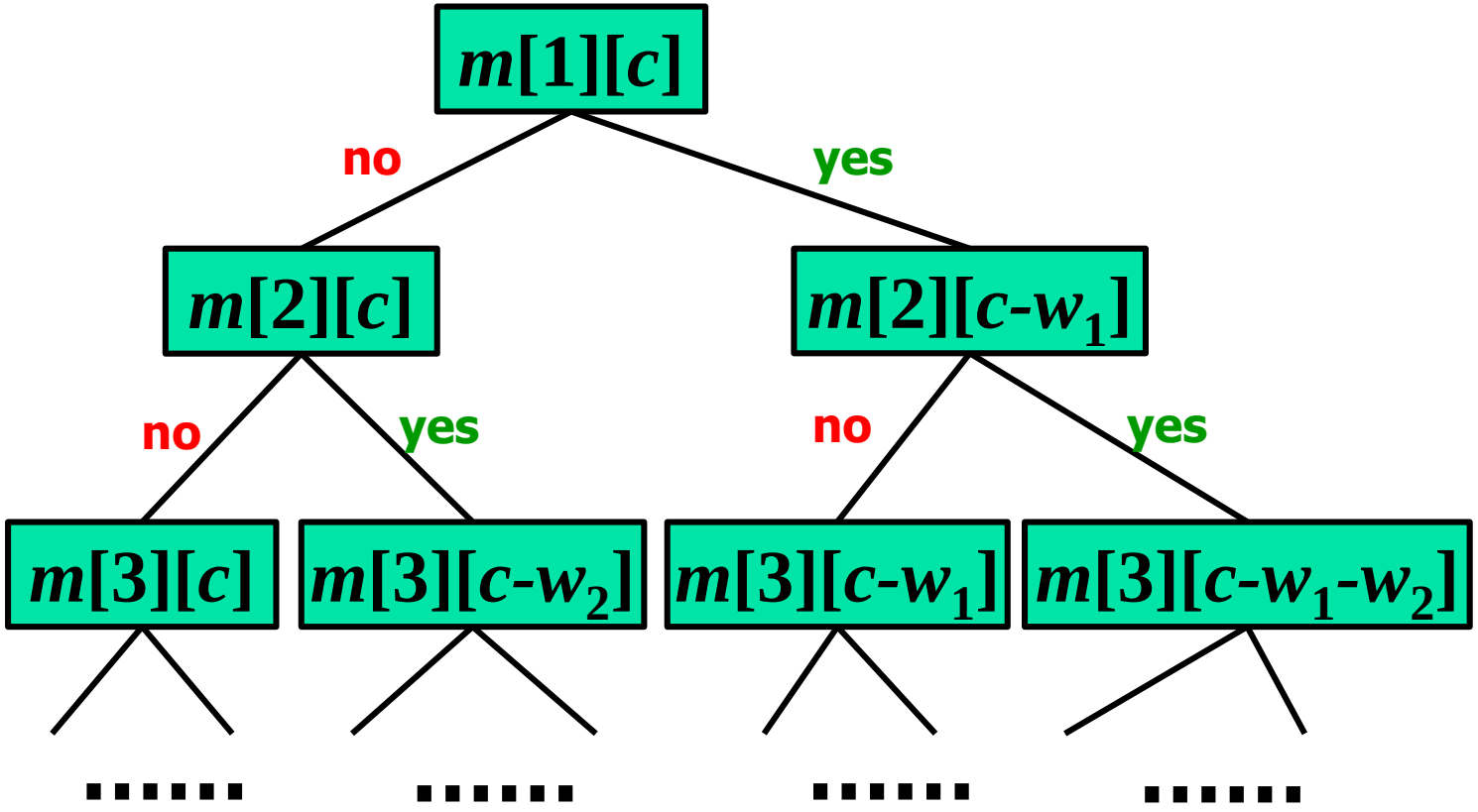
0-1背包问题—建立递归关系

- 设所给0-1背包问题的子问题的最优值为$m(i,j)$，即$m(i,j)$是背包容量为j，可选择物品为i, i+1, …, n时0-1背包问题的最优值。
- 由0-1背包问题的最优子结构性质，可以建立计算$m(i,j)$的递归式如下:

$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

$$m(n,j)=\begin{cases}v_n & j\geq w_n\\ 0 & 0\leq j<w_n\end{cases}$$

0-1背包问题—建立递归关系



0-1背包问题—最直观的算法

```
int Knapsack(int n, int c, int v[], int w[])
{
    for (int i = 0; i <= c; i++){
        if(i < w[n]) m[n][i]=0;
        else m[n][i] = v[n];
    }
    for (int j = n-1; j >=1; j--){
        for (int i = 0; i <= c; i++){
            if(i < w[j]) m[j][i] = m[j+1][i];
            else m[j][i] = max(m[j+1][i],
                               m[j+1][i-w[j]]+v[j]);
        }
    }
    return m[1][c];
}
```

算法复杂度分析：

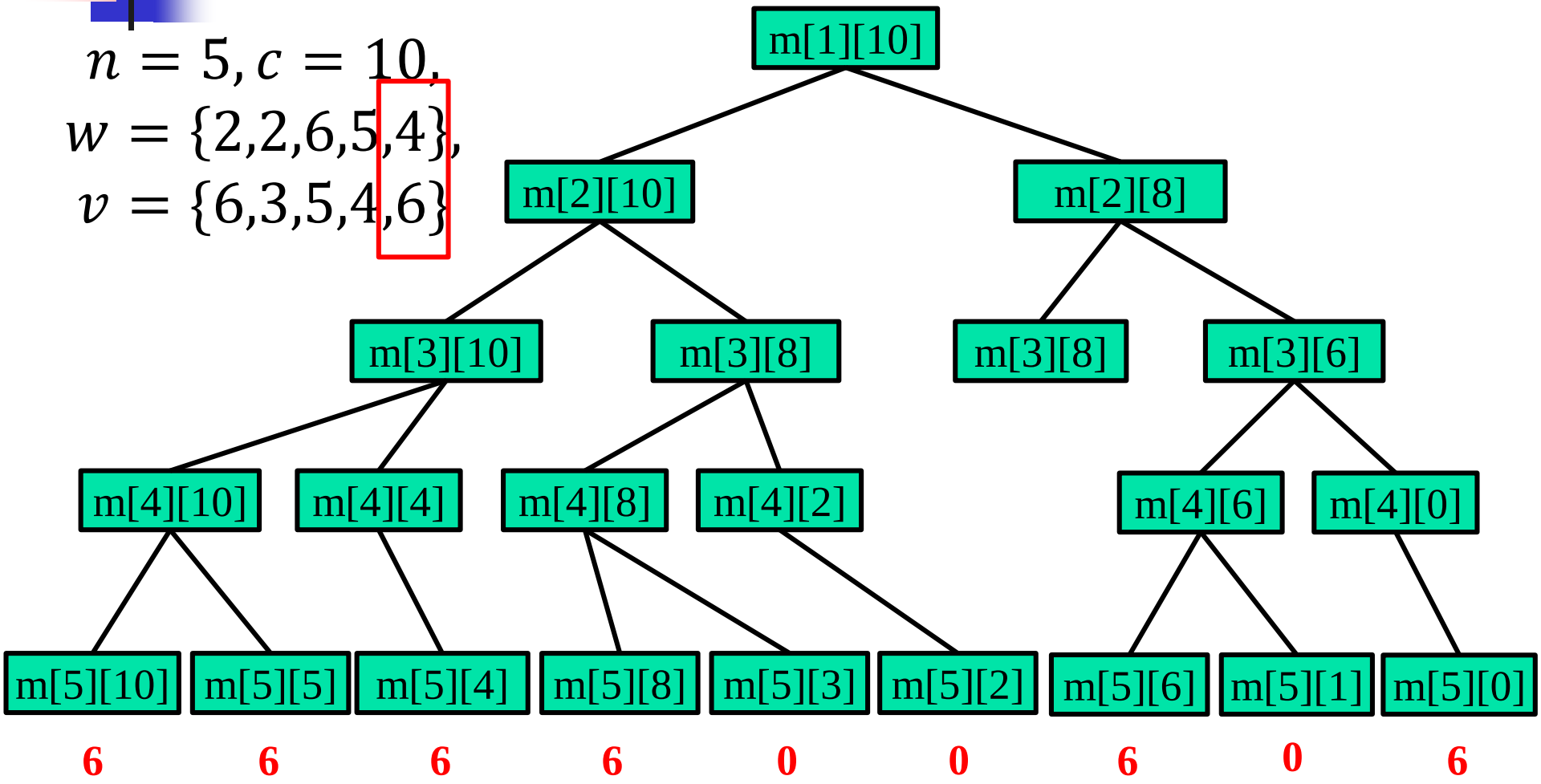
- 当物品重量w_i为整数时
- 递归式共有n层，每层最多c个节点，所以算法需要O(nc)计算时间。

0-1背包问题—存在的问题

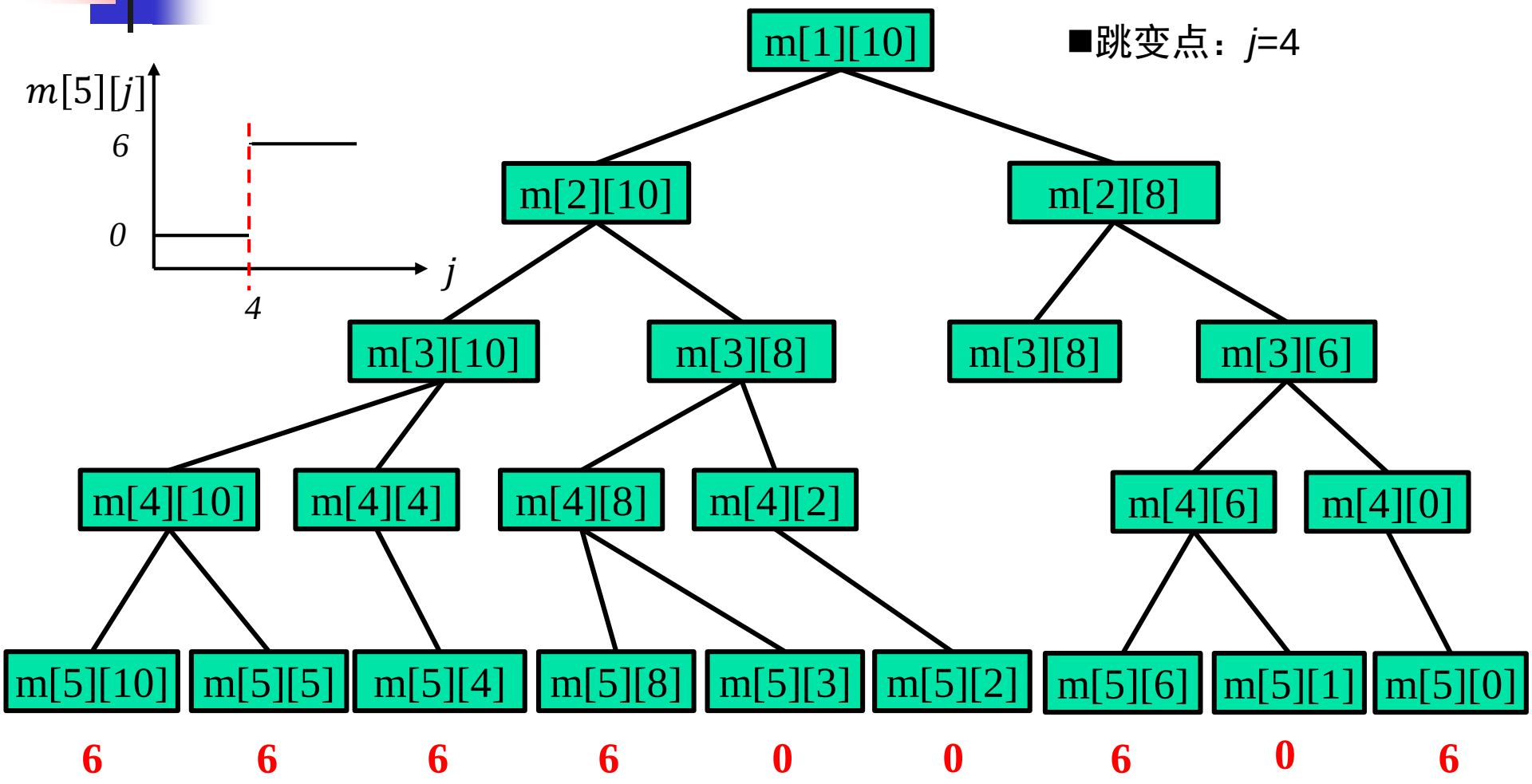
- 物品重量必须是整数
- 当背包容量c很大时（如$c=2^n$），则时间复杂度为$O(n2^n)$
- 能否设计一个算法解决上述问题？

0-1背包问题—算法改进

$n = 5, c = 10,$
$w = \{2,2,6,5,4\},$
$v = \{6,3,5,4,6\}$

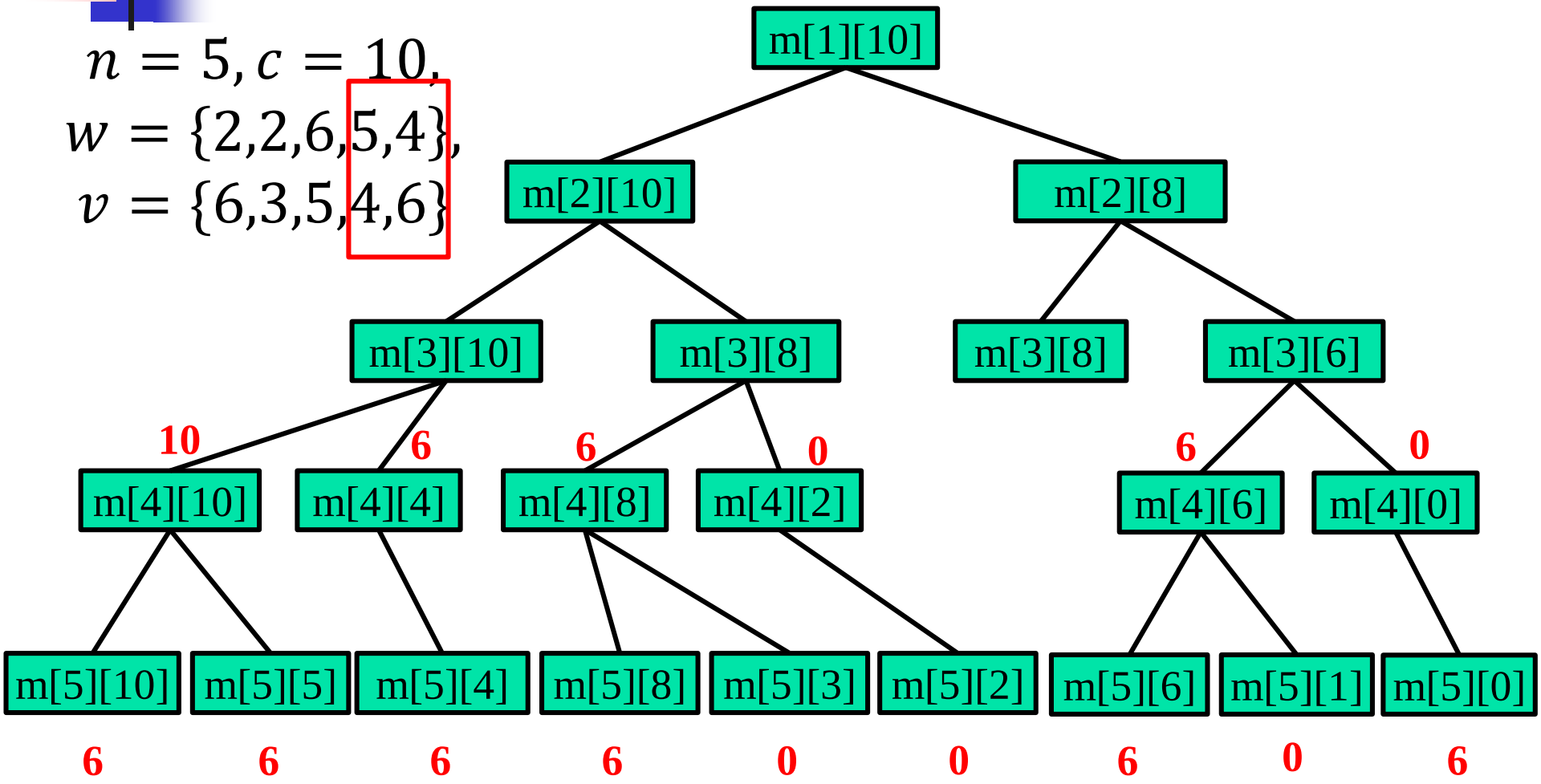


0-1背包问题—算法改进

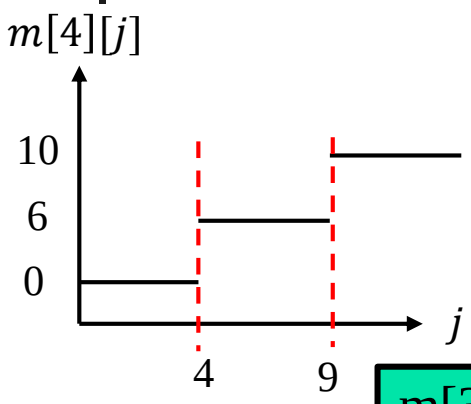

m[1][10]
■跳变点：j=4
m[5][j]
6
0
j
4
m[2][10]
m[2][8]
m[3][10]
m[3][8]
m[3][8]
m[3][6]
m[4][10]
m[4][4]
m[4][8]
m[4][2]
m[4][6]
m[4][0]
m[5][10]
m[5][5]
m[5][4]
m[5][8]
m[5][3]
m[5][2]
m[5][6]
m[5][1]
m[5][0]
6 6 6 6 0 0 6 0 6

0-1背包问题—算法改进

$n = 5, c = 10,$
$w = \{2,2,6,5,4\},$
$v = \{6,3,5,4,6\}$



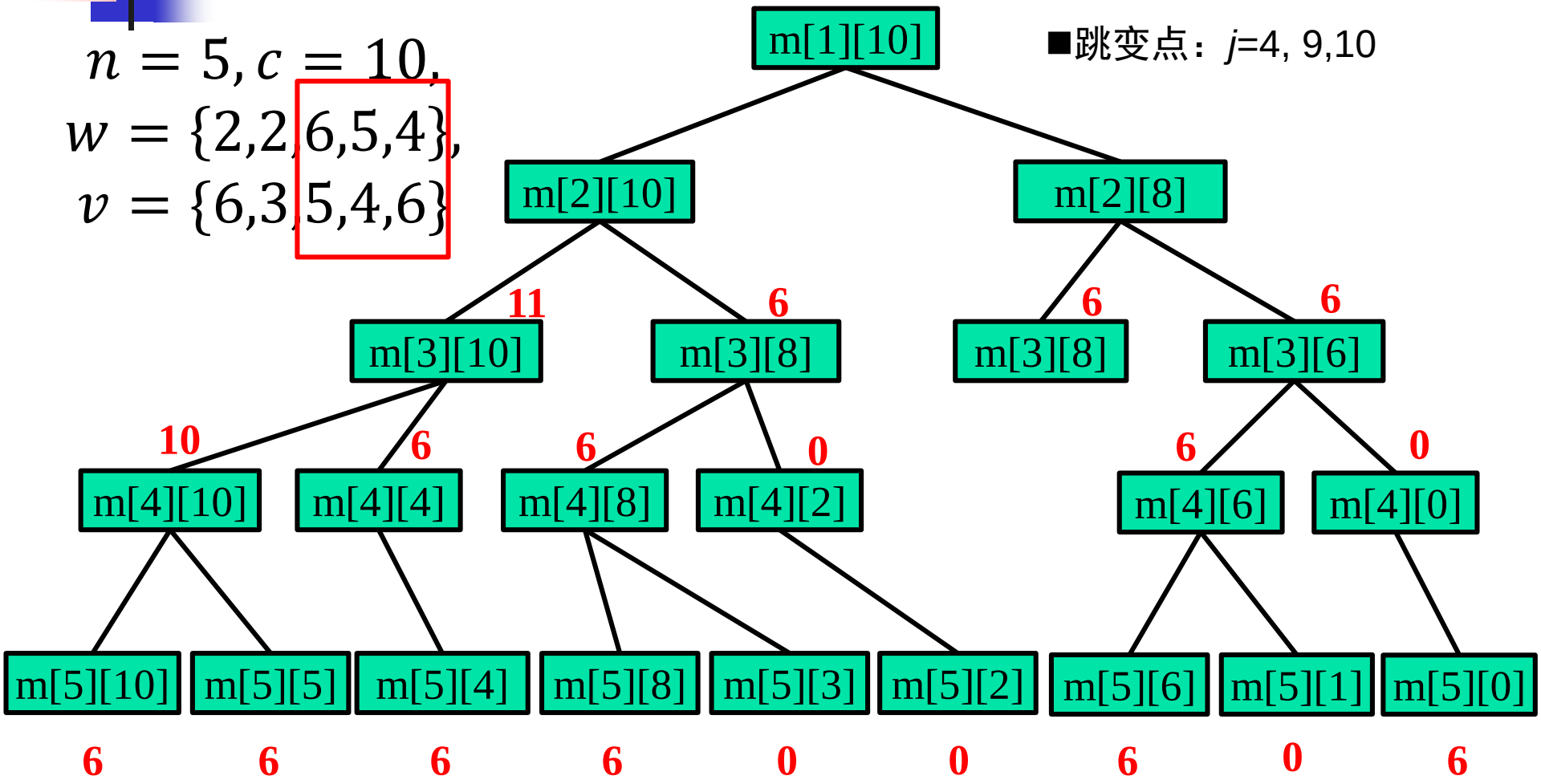
0-1背包问题—算法改进



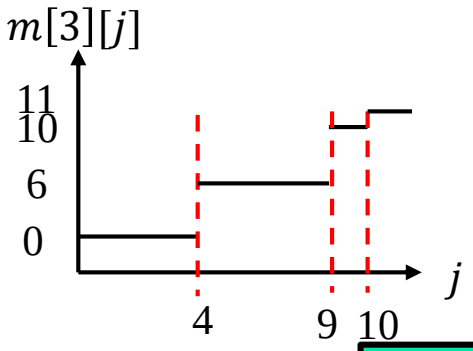
■跳变点：j=4, 9

0-1背包问题—算法改进

$n = 5, c = 10,$
$w = \{2,2,6,5,4\},$
$v = \{6,3,5,4,6\}$



0-1背包问题—算法改进

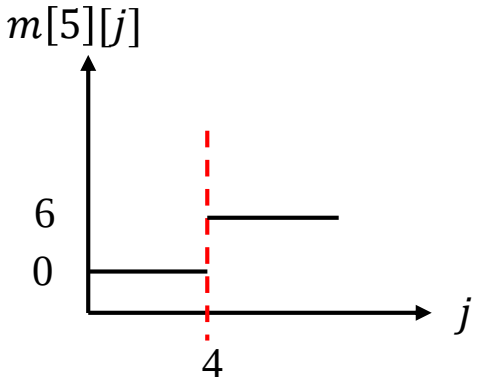


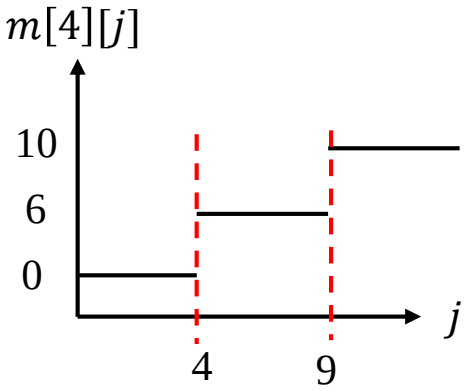
■跳变点：j=4, 9,10

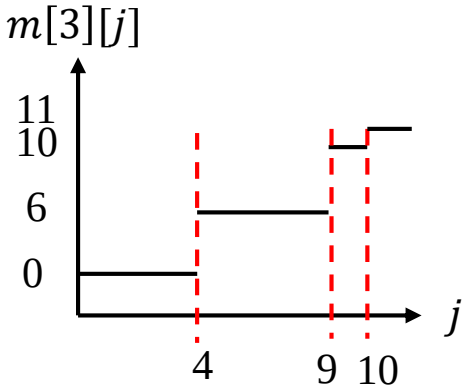
0-1背包问题—算法改进

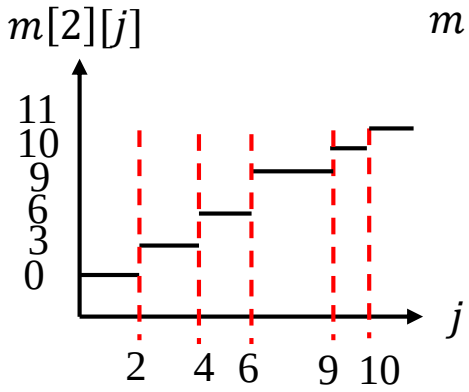
$n = 5, c = 10$

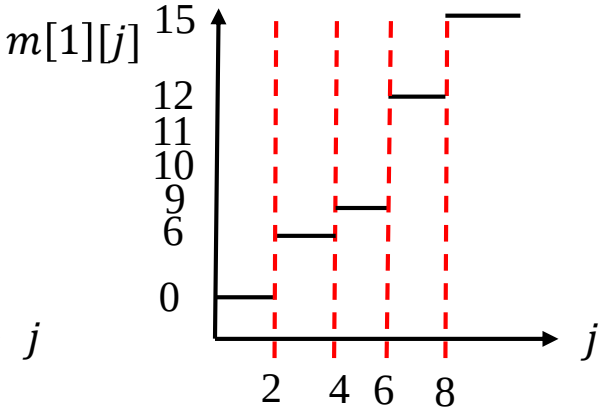
$w[i]$	2	2	6	5	4
$v[i]$	6	3	5	4	6











- 改进后算法的计算时间复杂度为$O(2^n)$。
- 当所给物品的重量$w_i(1 \leqslant i \leqslant n)$是整数时，改进后算法的计算时间复杂度为$O(\min\{nc, 2^n\})$。

本章小结

- 动态规划法的基本概念
  - 将大规模的问题分解为规模较小的子问题
  - 子问题之间相互不独立
  - 通过构建备忘录，以空间换时间
- 动态规划法的基本步骤
  - 分段、分析、求解
- 动态规划法的应用
  - 矩阵连乘
  - 最长公共子序列、最大字段和、图像压缩
  - 0-1背包问题

矩阵连乘

- 最优子结构

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8

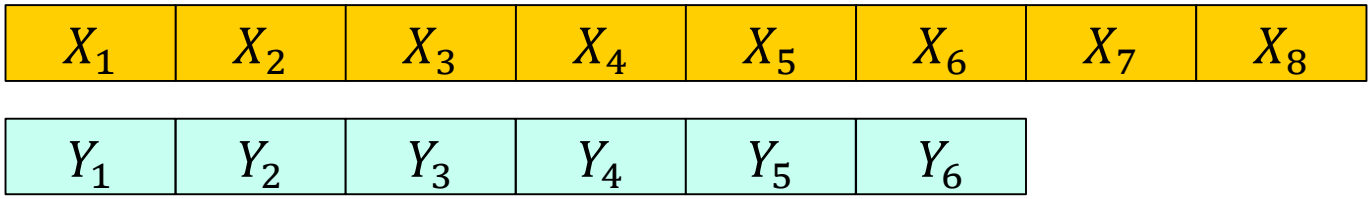
- 递推公式

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases}$$

- 运行实例

最长公共子序列

- 最优子结构



- 递推方程

$$C[i,j]=\begin{cases}0 & \\ C[i-1,j-1]+1 & \\ \max\{C[i,j-1],C[i-1,j]\} & \end{cases}$$

- 运行实例

最大字段和

- 最优子结构

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8

- 递推方程

$$b[j]=\max_{1\le j\le n}\{b[j-1]+a[j],a[j]\}$$

- 运行实例

图像压缩

- 最优子结构

A_1	A_2	A_3	A_4	A_5	A_6	A_7	A_8

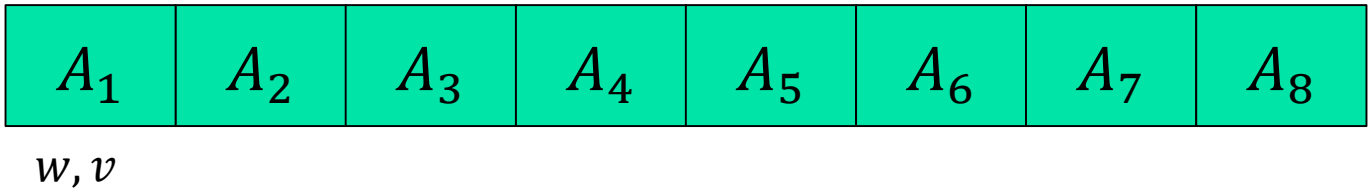
- 递推方程

$$s[i] = \min_{1 \le k \le \min\{i,256\}} \{s[i-k] + k * \mathrm{bmax}(i-k+1, i)\} + 11$$

- 运行实例

0-1背包问题

- 最优子结构



- 递推方程

$$m(i,j)=\begin{cases}\max\{m(i+1,j),m(i+1,j-w_i)+v_i\} & j\geq w_i\\ m(i+1,j) & 0\leq j<w_i\end{cases}$$

- 运行实例

$$m(n,j)=\begin{cases}v_n & j\geq w_n\\ 0 & 0\leq j<w_n\end{cases}$$