

算法设计与分析

第四讲 动态规划

哈尔滨工业大学

李穆

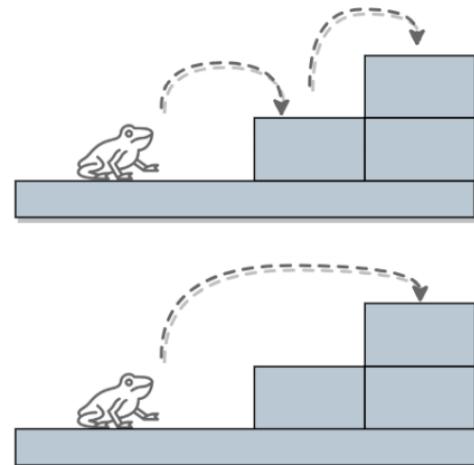


蛙跳问题



蛙跳问题

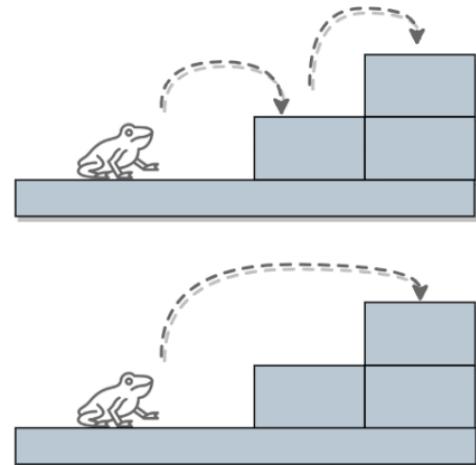
问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个10级的台阶总共有多少种跳法？



蛙跳问题

问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个10级的台阶总共有多少种跳法？

要想跳到第10级台阶，要么是先跳到第9级，然后再跳1级台阶上去;要么是先跳到第8级，然后一次迈2级台阶上去。

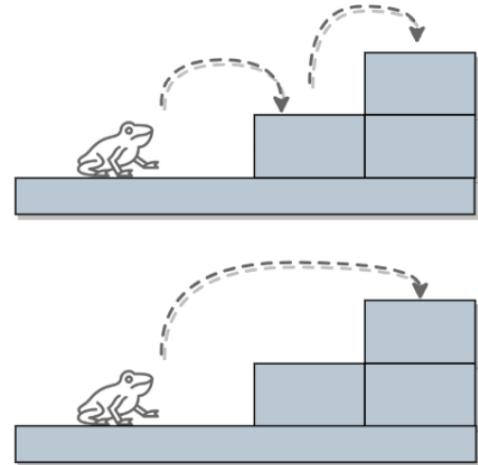


蛙跳问题

问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个10级的台阶总共有多少种跳法？

要想跳到第10级台阶，要么是先跳到第9级，然后再跳1级台阶上去;要么是先跳到第8级，然后一次迈2级台阶上去。

同理，要想跳到第9级台阶，要么是先跳到第8级，然后再跳1级台阶上去;要么是先跳到第7级，然后一次迈2级台阶上去。

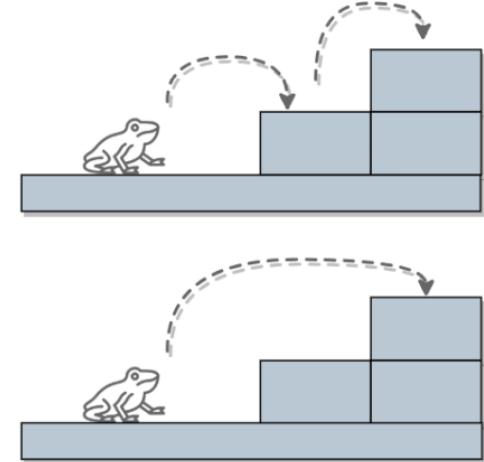


蛙跳问题

问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个10级的台阶总共有多少种跳法？

要想跳到第10级台阶，要么是先跳到第9级，然后再跳1级台阶上去;要么是先跳到第8级，然后一次迈2级台阶上去。

同理，要想跳到第9级台阶，要么是先跳到第8级，然后再跳1级台阶上去;要么是先跳到第7级，然后一次迈2级台阶上去。



$$f(10) = f(9) + f(8)$$

$$f(9) = f(8) + f(7)$$

$$f(8) = f(7) + f(6)$$

...

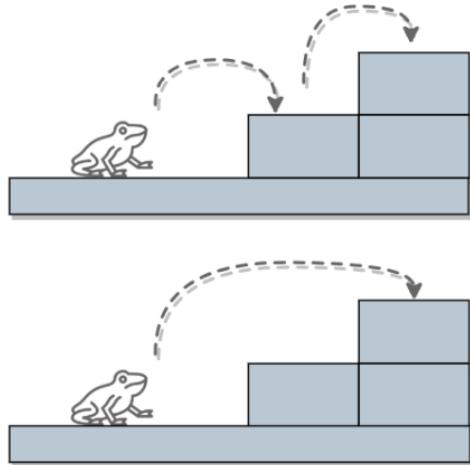
$$f(3) = f(2) + f(1)$$

蛙跳问题

问题：一只青蛙一次可以跳上1级台阶，也可以跳上2级台阶。求该青蛙跳上一个10级的台阶总共有多少种跳法？

要想跳到第10级台阶，要么是先跳到第9级，然后再跳1级台阶上去;要么是先跳到第8级，然后一次迈2级台阶上去。

同理，要想跳到第9级台阶，要么是先跳到第8级，然后再跳1级台阶上去;要么是先跳到第7级，然后一次迈2级台阶上去。



$$f(10) = f(9) + f(8)$$

$$f(9) = f(8) + f(7)$$

$$f(8) = f(7) + f(6)$$

...

$$f(3) = f(2) + f(1)$$

通用公式为: $f(n) = f(n-1) + f(n-2)$

蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

```
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```

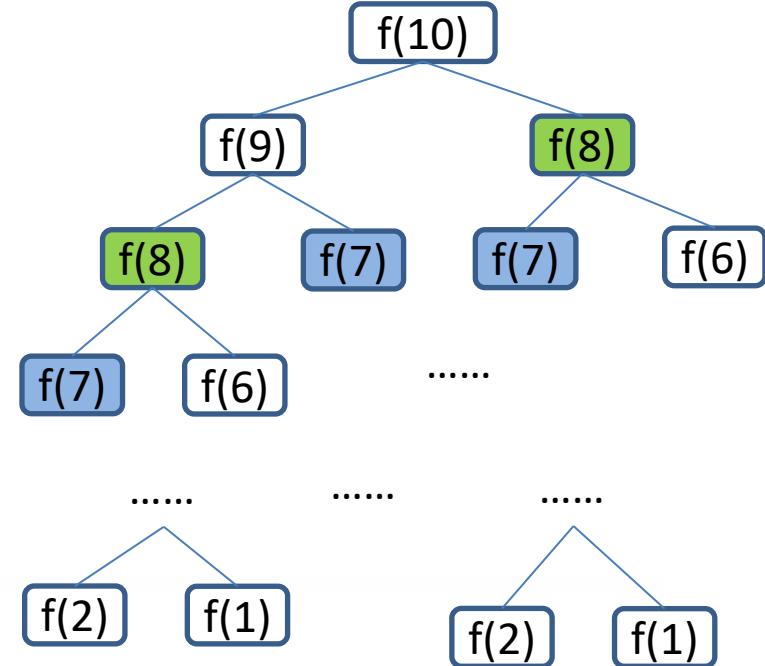
蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

```
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```



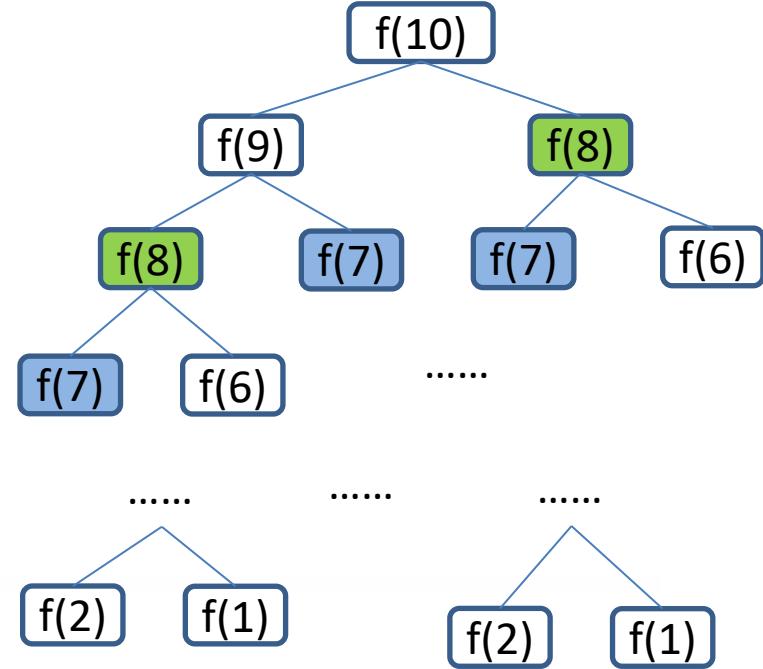
蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

```
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```



时间复杂度： $\mathcal{O}(2^n)$

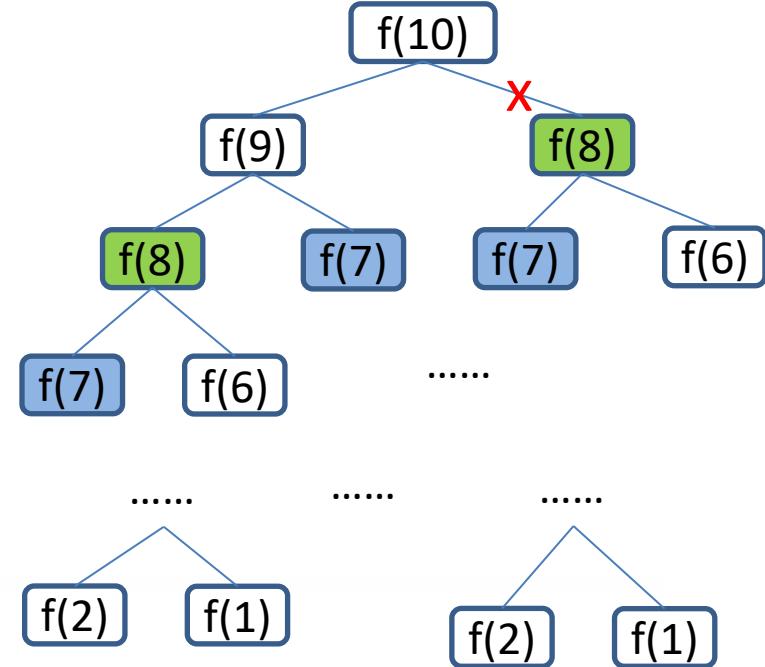
蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

```
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```



时间复杂度： $\mathcal{O}(2^n)$

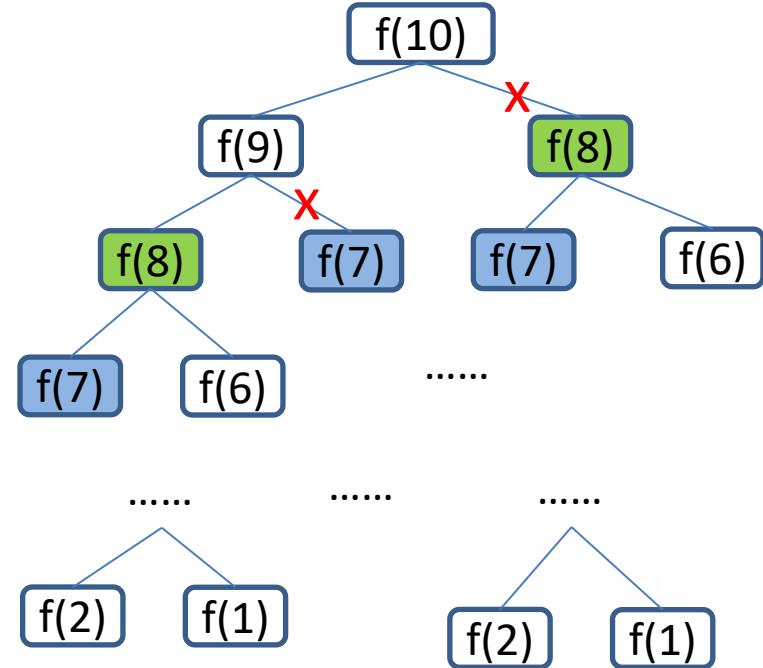
蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}
```

```
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```



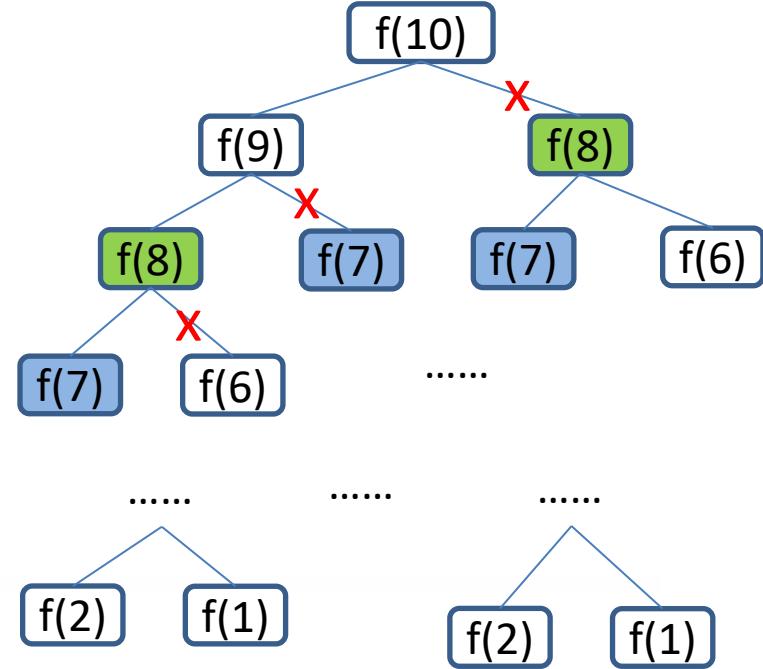
时间复杂度： $\mathcal{O}(2^n)$

蛙跳问题

如何设计算法，编写程序，给定任意的n求f(n)

$$f(n) = f(n-1) + f(n-2)$$

```
public int f(int n) {  
    return f(n - 1) + f(n - 2);  
}  
  
public int f(int n) {  
    if (n == 1) {  
        return 1;  
    }  
    if (n == 2) {  
        return 2;  
    }  
    return f(n - 1) + f(n - 2);  
}
```



时间复杂度： $\mathcal{O}(2^n)$

蛙跳问题

- 首先计算最小的问题
 - 记录 $f[1], f[2]$ 的值
- 然后计算大一点的问题
 - 记录 $f[3]$
- ...
- 然后 计算更大的问题
 - 记录 $f[n-1]$
- 最后计算最终的问题.
 - 得到 $f[n]$

蛙跳问题

- 首先计算最小的问题
 - 记录 $f[1], f[2]$ 的值
- 然后计算大一点的问题
 - 记录 $f[3]$
- ...
- 然后 计算更大的问题
 - 记录 $f[n-1]$
- 最后计算最终的问题.
 - 得到 $f[n]$

时间复杂度： $O(n)$

蛙跳问题

- 首先计算最小的问题
 - 记录 $f[1], f[2]$ 的值
- 然后计算大一点的问题
 - 记录 $f[3]$
- ...
- 然后 计算更大的问题
 - 记录 $f[n-1]$
- 最后计算最终的问题.
 - 得到 $f[n]$

时间复杂度： $O(n)$



本讲内容

- 4. 1 动态规划的原理
- 4. 2 矩阵乘法问题
- 4. 3 最长公共子序列问题
- 4. 4 0-1背包问题
- 4. 5 最优二分搜索树

Why?



Why?

- 分治技术的问题
 - 子问题是相互独立的
 - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低

Why?

- 分治技术的问题
 - 子问题是相互独立的
 - 如果子问题不是相互独立的，分治方法将重复计算公共子问题，效率很低
- 优化问题
 - 给定一组约束条件和一个代价函数，在解空间中搜索具有最小或最大代价的优化解
 - 很多优化问题可分为多个子问题，子问题相互关联，子问题的解被重复使用

What?



What?

- 动态规划的特点

What?

- 动态规划的特点
 - 把原始问题划分成一系列子问题

What?

- 动态规划的特点
 - 把原始问题划分成一系列子问题
 - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间

What?

- 动态规划的特点
 - 把原始问题划分成一系列子问题
 - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
 - 自底向上地计算

What?

- 动态规划的特点
 - 把原始问题划分成一系列子问题
 - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
 - 自底向上地计算
- 适用范围

What?

- 动态规划的特点
 - 把原始问题划分成一系列子问题
 - 求解每个子问题仅一次，并将其结果保存在一个表中，以后用到时直接存取，不重复计算，节省计算时间
 - 自底向上地计算
- 适用范围
 - 一类优化问题：可分为多个相关子问题，子问题的解被重复使用

How?



How?

- 使用动态规划的条件



How?

- 使用动态规划的条件
 - 优化子结构



How?

- 使用动态规划的条件
 - 优化子结构
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。

How?

- 使用动态规划的条件
 - 优化子结构
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
 - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性

How?

- 使用动态规划的条件
 - 优化子结构
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
 - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
 - 优化子结构使得我们能自下而上地完成求解过程

How?

- 使用动态规划的条件
 - 优化子结构
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
 - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
 - 优化子结构使得我们能自下而上地完成求解过程
 - 重叠子问题

How?

- 使用动态规划的条件
 - 优化子结构
 - 当一个问题的优化解包含了子问题的优化解时，我们说这个问题具有优化子结构。
 - 缩小子问题集合，只需那些优化问题中包含的子问题，降低实现复杂性
 - 优化子结构使得我们能自下而上地完成求解过程
 - 重叠子问题
 - 在问题的求解过程中，很多子问题的解将被多次使用



- 动态规划算法的设计步骤
 - 分析优化解的结构
 - 递归地定义最优解的代价
 - 自底向上地计算优化解的代价保存之，并获取构造最优解的信息
 - 根据构造最优解的信息构造优化解



本讲内容

4. 1 动态规划的原理

4. 2 矩阵乘法问题

4. 3 最长公共子序列问题

4. 4 0-1背包问题

4. 5 最优二分搜索树

问题的定义

- 输入: $\langle A_1, A_2, \dots, A_n \rangle$, A_i 是矩阵
- 输出: 计算 $A_1 \times A_2 \times \dots \times A_n$ 的最小代价方法



问题的定义

- 输入: $\langle A_1, A_2, \dots, A_n \rangle$, A_i 是矩阵
- 输出: 计算 $A_1 \times A_2 \times \dots \times A_n$ 的最小代价方法

矩阵乘法的代价/复杂性: 乘法的次数



问题的定义

- 输入: $\langle A_1, A_2, \dots, A_n \rangle$, A_i 是矩阵
- 输出: 计算 $A_1 \times A_2 \times \dots \times A_n$ 的最小代价方法

矩阵乘法的代价/复杂性: 乘法的次数

若 A 是 $p \times q$ 矩阵, B 是 $q \times r$ 矩阵, 则 $A \times B$ 的代价是 $O(pqr)$

动机

- 矩阵链乘法的实现
 - 矩阵乘法满足结合率。
 - 计算一个矩阵链的乘法可有多种方法：

动机

- 矩阵链乘法的实现
 - 矩阵乘法满足结合率。
 - 计算一个矩阵链的乘法可有多种方法：

例如, $(A_1 \times A_2 \times A_3 \times A_4)$

$$= (A_1 \times (A_2 \times (A_3 \times A_4)))$$

$$= ((A_1 \times A_2) \times (A_3 \times A_4))$$

....

$$= ((A_1 \times A_2) \times A_3) \times A_4)$$

- 矩阵链乘法的代价与计算顺序的关系



- 矩阵链乘法的代价与计算顺序的关系
 - 设 $A_1=10\times 100$ 矩阵, $A_2=100\times 5$ 矩阵, $A_3=5\times 50$ 矩阵



- 矩阵链乘法的代价与计算顺序的关系
 - 设 $A_1=10\times 100$ 矩阵, $A_2=100\times 5$ 矩阵, $A_3=5\times 50$ 矩阵

$$T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = \textcolor{red}{7500}$$



- 矩阵链乘法的代价与计算顺序的关系
 - 设 $A_1=10\times 100$ 矩阵, $A_2=100\times 5$ 矩阵, $A_3=5\times 50$ 矩阵

$$T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$$

$$T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$$

- 矩阵链乘法的代价与计算顺序的关系
 - 设 $A_1=10\times 100$ 矩阵, $A_2=100\times 5$ 矩阵, $A_3=5\times 50$ 矩阵

$$T((A_1 \times A_2) \times A_3) = 10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$$

$$T(A_1 \times (A_2 \times A_3)) = 100 \times 5 \times 50 + 10 \times 100 \times 50 = 75000$$

结论: 不同计算顺序有不同的代价

- 矩阵链乘法优化问题的解空间
 - 设 $p(n)$ =计算 n 个矩阵乘积的方法数
 - $p(n)$ 的递归方程



- 矩阵链乘法优化问题的解空间
 - 设 $p(n)$ =计算 n 个矩阵乘积的方法数
 - $p(n)$ 的递归方程

$$(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$$



- 矩阵链乘法优化问题的解空间
 - 设 $p(n)$ =计算 n 个矩阵乘积的方法数
 - $p(n)$ 的递归方程

$$(A_1 \times \dots \times A_k) \times (A_{k+1} \times \dots \times A_n)$$

$$p(n) = 1 \quad \text{if } n=1$$

$$p(n) = \sum_{k=1}^{n-1} p(k)p(n-k) \quad \text{if } n>1$$

$$\begin{aligned} p(n) &= C(n-1) = \text{Catalan数} = \frac{1}{n} \binom{2(n-1)}{n-1} \\ &= \Omega(4^n/n^{3/2}) \end{aligned}$$

- 矩阵链乘法优化问题的解空间
 - 设 $p(n)$ =计算 n 个矩阵乘积的方法数
 - $p(n)$ 的递归方程

如此之大的解空间是无法用枚举方法
求出最优解的！

$$p(n) = \sum_{k=1}^{n-1} p(k)p(n-k) \quad \text{if } n > 1$$

$$\begin{aligned} p(n) &= C(n-1) = \text{Catalan 数} = \frac{1}{n} \binom{2(n-1)}{n-1} \\ &= \Omega(4^n/n^{3/2}) \end{aligned}$$

下边开始设计求解矩阵链乘法问题
的动态规划算法



下边开始设计求解矩阵链乘法问题 的动态规划算法

- 分析优化解的结构



下边开始设计求解矩阵链乘法问题 的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价



下边开始设计求解矩阵链乘法问题 的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价保存之，并获取构造最优解的信息



下边开始设计求解矩阵链乘法问题 的动态规划算法

- 分析优化解的结构
- 递归地定义最优解的代价
- 自底向上地计算优化解的代价保存之, 并获取构造最优解的信息
- 根据构造最优解的信息构造优化解

分析优化解的结构

- 两个记号

- $A_{i:j} = A_i \times A_{i+1} \times \dots \times A_j$

- $\text{cost}(A_{i:j})$ =计算 $A_{i:j}$ 的代价

分析优化解的结构

- 两个记号
 - $A_{i-j} = A_i \times A_{i+1} \times \dots \times A_j$
 - $\text{cost}(A_{i-j})$ = 计算 A_{i-j} 的代价
- 优化解的结构
 - 若计算 A_{1-n} 的优化顺序在 k 处断开矩阵链，即 $A_{1-n} = A_{1-k} \times A_{k+1-n}$ ，则在 A_{1-n} 的优化顺序中，对于子问题 A_{1-k} 的解必须是 A_{1-k} 的优化解，对于子问题 A_{k+1-n} 的解必须是 A_{k+1-n} 的优化解

分析优化解的结构

- 两个记号
 - $A_{i:j} = A_i \times A_{i+1} \times \dots \times A_j$
 - $\text{cost}(A_{i:j})$ =计算 $A_{i:j}$ 的代价
- 优化解的结构
 - 若计算 $A_{1:n}$ 的优化顺序在 k 处断开矩阵链，具有优化子结构：
问题的优化解包括子问题优化解
化解，对应于子问题 $A_{k+1:n}$ 的解必须是 $A_{k+1:n}$ 的优化解

- 子问题重叠性

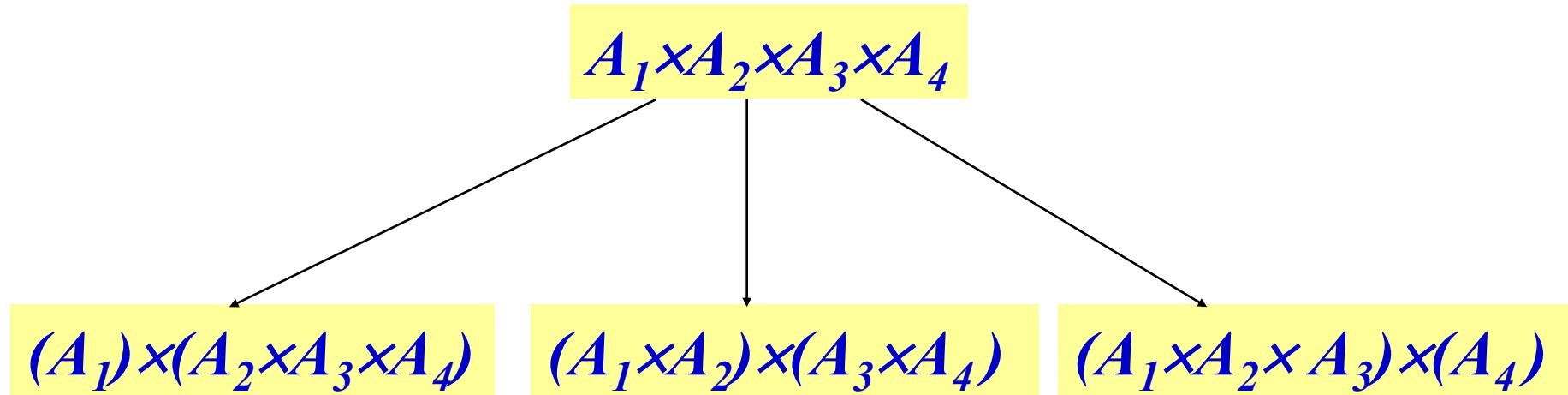


- 子问题重叠性

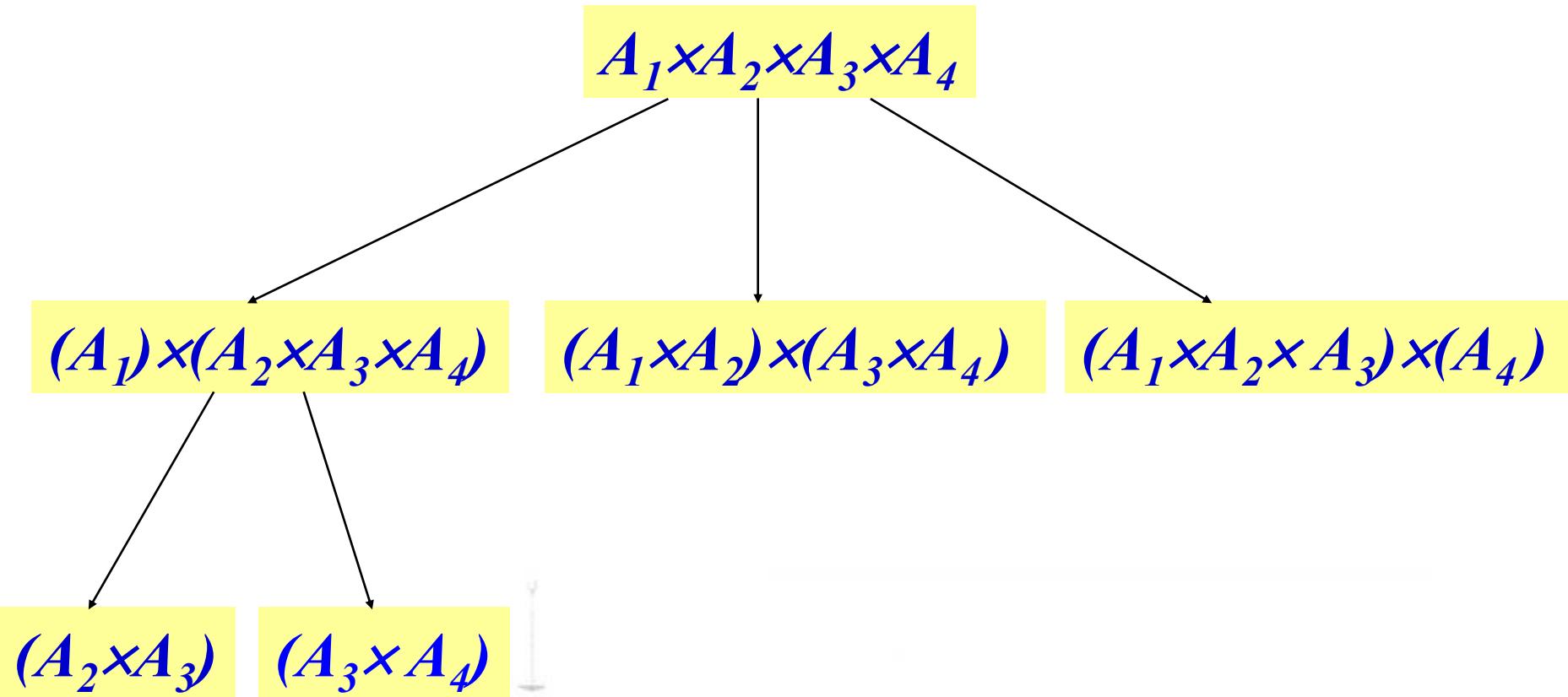
$$A_1 \times A_2 \times A_3 \times A_4$$



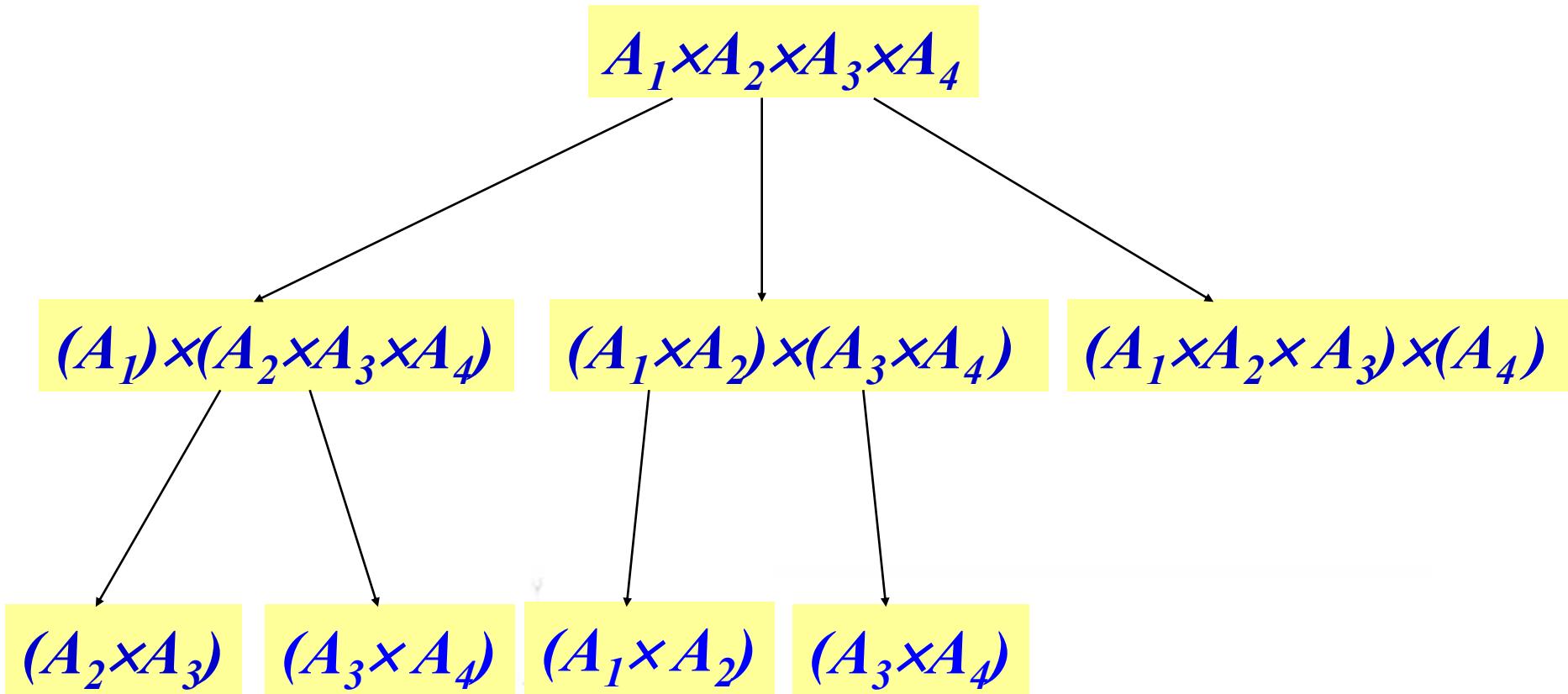
- 子问题重叠性



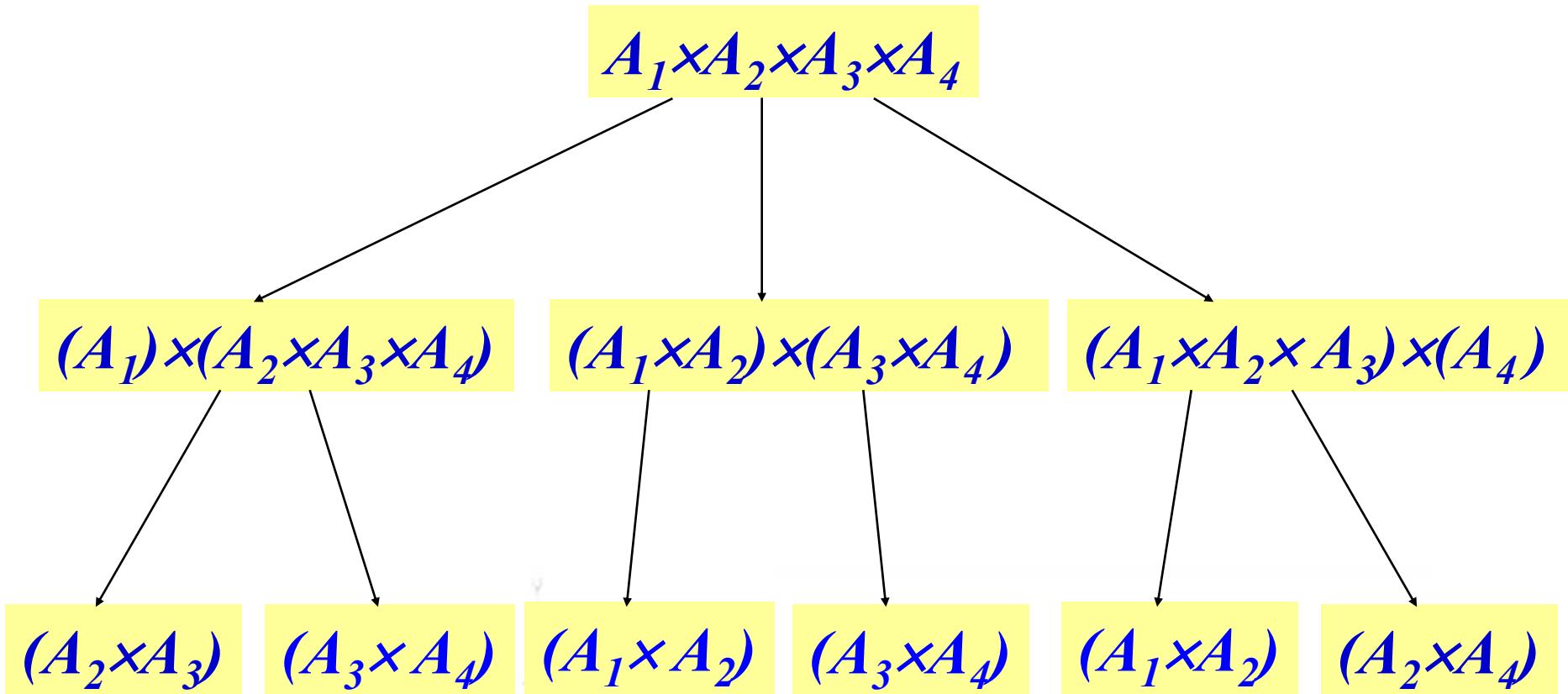
- 子问题重叠性



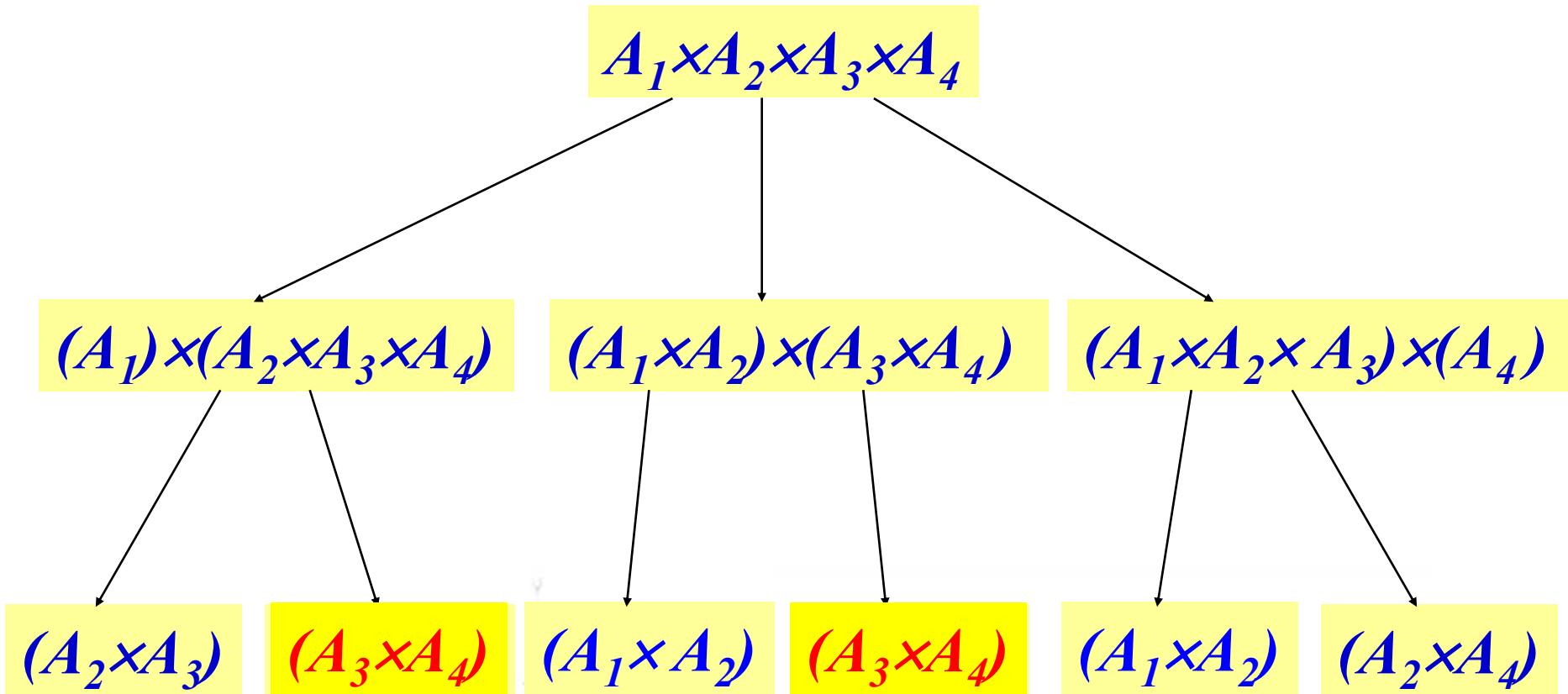
• 子问题重叠性



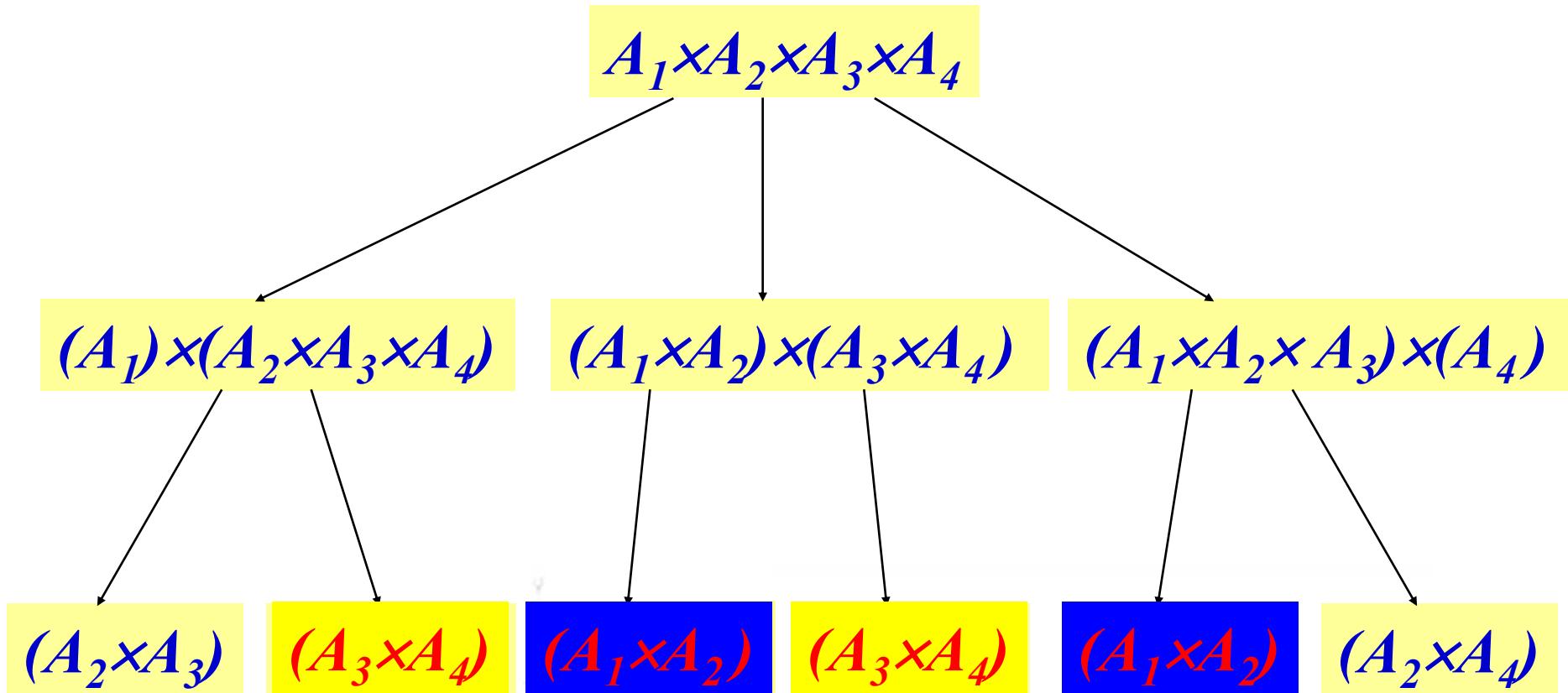
• 子问题重叠性



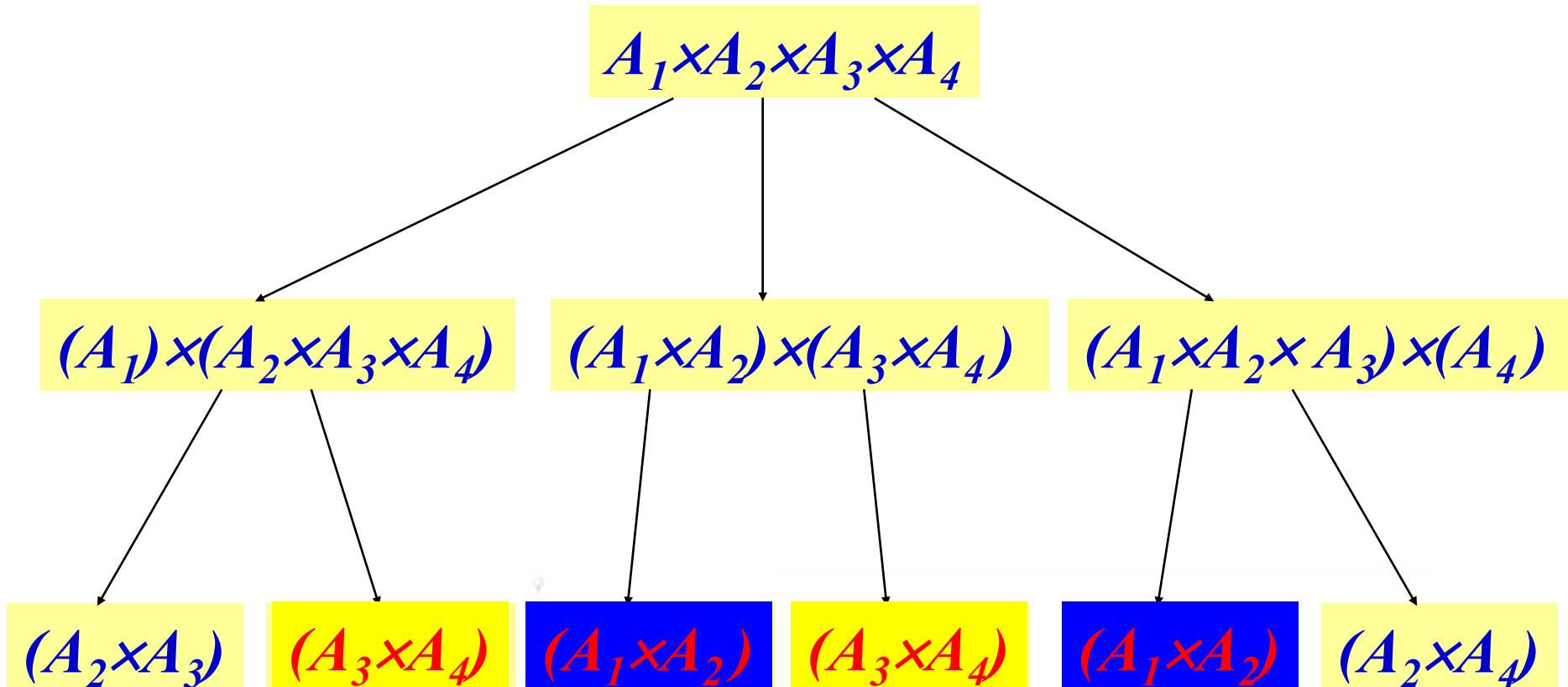
• 子问题重叠性



• 子问题重叠性



- 子问题重叠性



具有子问题重叠性

递归地定义最优解的代价

- 假设
 - $m[i, j]$ = 计算 $A_{i \sim j}$ 的最小乘法数
 - $m[1, n]$ = 计算 $A_{1 \sim n}$ 的最小乘法数
 - $A_1 \dots A_k A_{k+1} \dots A_n$ 是优化解(k 实际上是不可预知)

递归地定义最优解的代价

- 假设

- $m[i, j]$ = 计算 $A_{i \sim j}$ 的最小乘法数

- $m[1, n]$ = 计算 $A_{1 \sim n}$ 的最小乘法数

- $A_1 \dots A_k A_{k+1} \dots A_n$ 是优化解 (k 实际上是不可预知)

- 代价方程

- $m[i, i] = \text{计算 } A_{i \sim i} \text{ 的最小乘法数} = 0$

- $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$

递归地定义最优解的代价

- 假设

- $m[i, j]$ = 计算 $A_{i \sim j}$ 的最小乘法数

- $m[1, n]$ = 计算 $A_{1 \sim n}$ 的最小乘法数

- $A_1 \dots A_k A_{k+1} \dots A_n$ 是优化解 (k 实际上是不可预知)

- 代价方程

- $m[i, i] = \text{计算 } A_{i \sim i} \text{ 的最小乘法数} = 0$

- $m[i, j] = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$

- 其中, $p_{i-1} p_k p_j$ 是计算 $A_{i \sim k} \times A_{k+1 \sim j}$ 所需乘法数,
 $A_{i \sim k}$ 和 $A_{k+1 \sim j}$ 分别是 $p_{i-1} \times p_k$ 和 $p_k \times p_j$ 矩阵.

考慮到所有的 k , 优化解的代价方程为

$$m[i, j] = 0 \quad \text{if } i=j$$

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$$\text{if } i < j$$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$$m[1,5]$$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + pqr \\ m[2,3] + m[4,4] + pqr \end{cases}$$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,5]$

$m[3,5]$

$m[4,5]$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + pqr \\ m[2,3] + m[4,4] + pqr \end{cases}$$

$m[5,5]$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$

$m[3,5]$

$m[4,5]$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + pqr \\ m[2,3] + m[4,4] + pqr \end{cases} \quad m[5,5]$$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$

$m[3,3]$ $m[3,4]$ $m[3,5]$

$m[4,5]$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + pqr \\ m[2,3] + m[4,4] + pqr \end{cases} \quad m[5,5]$$

自底向上计算优化解的代价

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_0 p_k p_5 \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$

$m[3,3]$ $m[3,4]$ $m[3,5]$

$m[4,4]$ $m[4,5]$

$$m[2,4] = \min \begin{cases} m[2,2] + m[3,4] + pqr \\ m[2,3] + m[4,4] + pqr \end{cases} \quad m[5,5]$$

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$



$$m[i,j] = \min_{i \leq k < j} \{ m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \}$$

$$\boxed{m[1,5]}$$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$

$m[1,5]$

$m[2,2]$

$m[3,3]$

$m[4,4]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$

$m[1,5]$

$m[2,2]$

$m[3,3]$

$m[4,4]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$

$m[1,5]$

$m[2,2]$ $m[2,3]$

$m[3,3]$

$m[4,4]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$

$m[1,5]$

$m[2,2]$ $m[2,3]$

$m[3,3]$ $m[3,4]$

$m[4,4]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$

$m[1,5]$

$m[2,2]$ $m[2,3]$

$m[3,3]$ $m[3,4]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$

$m[1,5]$

$m[2,2]$ $m[2,3]$

$m[3,3]$ $m[3,4]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$

$m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$

$m[3,3]$ $m[3,4]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$

$m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$

$m[3,3]$ $m[3,4]$ $m[3,5]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$

$m[3,3]$ $m[3,4]$ $m[3,5]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$

$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$

$m[3,3]$ $m[3,4]$ $m[3,5]$

$m[4,4]$ $m[4,5]$

$m[5,5]$



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

~~$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$~~

~~$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$~~

~~$m[3,3]$ $m[3,4]$ $m[3,5]$~~

~~$m[4,4]$ $m[4,5]$~~

~~$m[5,5]$~~



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

~~$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$~~

~~$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$~~

~~$m[3,3]$ $m[3,4]$ $m[3,5]$~~

~~$m[4,4]$ $m[4,5]$~~

~~$m[5,5]$~~



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

~~$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$~~

~~$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$~~

~~$m[3,3]$ $m[3,4]$ $m[3,5]$~~

~~$m[4,4]$ $m[4,5]$~~

~~$m[5,5]$~~



$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$

~~$m[1,1]$ $m[1,2]$ $m[1,3]$ $m[1,4]$ $m[1,5]$~~

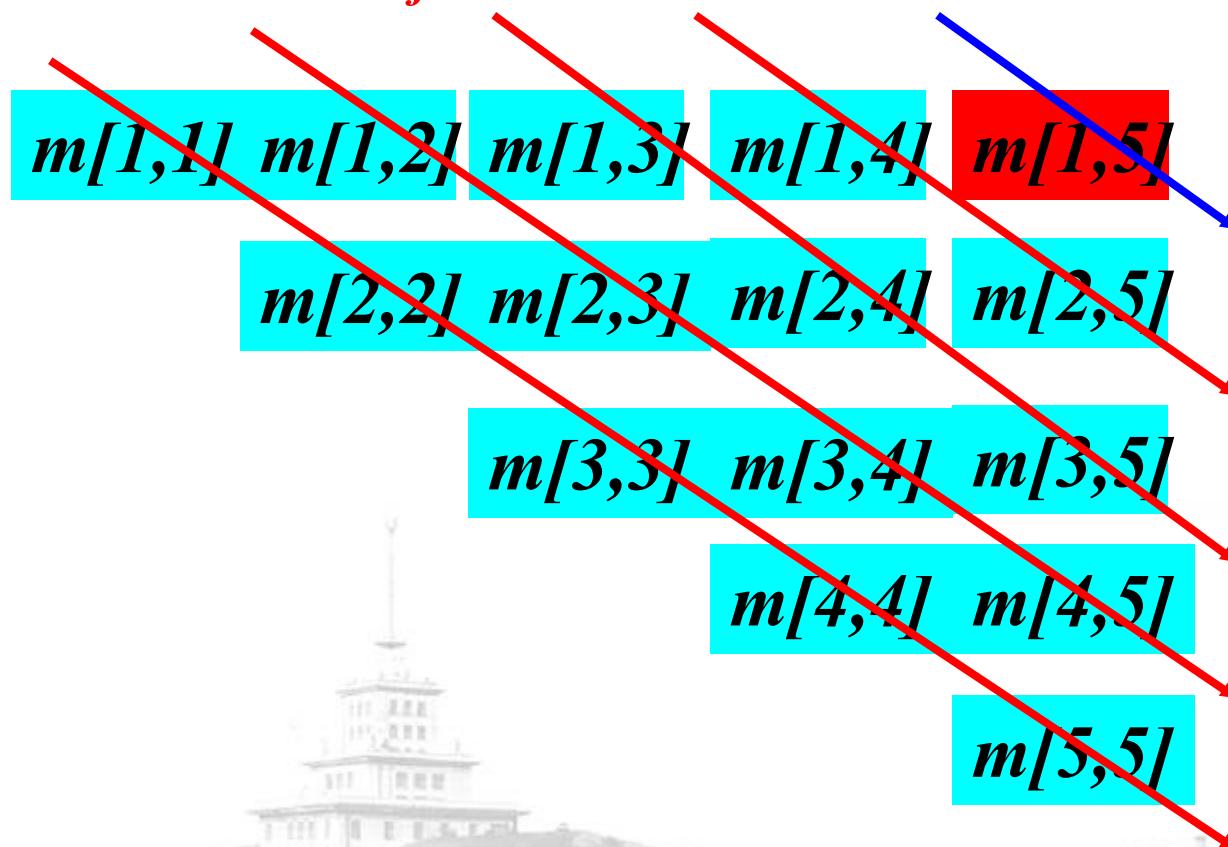
~~$m[2,2]$ $m[2,3]$ $m[2,4]$ $m[2,5]$~~

~~$m[3,3]$ $m[3,4]$ $m[3,5]$~~

~~$m[4,4]$ $m[4,5]$~~

~~$m[5,5]$~~

$$m[i, j] = \min_{i \leq k < j} \{ m[i, k] + m[k+1, j] + p_{i-1} p_k p_j \}$$



Matrix-Chain-Order(p)

$n=\text{length}(p)-1;$

FOR $i=1$ TO n DO

$m[i, i]=0;$

FOR $l=2$ TO n DO /* 计算第 l 对角线 */

FOR $i=1$ TO $n-l+1$ DO

$j=i+l-1;$

$m[i, j]=\infty;$

FOR $k \leftarrow i$ To $j-1$ DO /* 计算 $m[i, j]$ */

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

IF $q < m[i, j]$ **THEN** $m[i, j]=q;$

Return $m.$

获取构造最优解的信息

Matrix-Chain-Order(p)

$n=\text{length}(p)-1;$

FOR $i=1$ TO n DO

$m[i, i]=0;$

FOR $l=2$ TO n DO

FOR $i=1$ TO $n-l+1$ DO

$j=i+l-1;$

$m[i, j]=\infty;$

FOR $k \leftarrow i$ To $j-1$ DO

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

 IF $q < m[i, j]$ THEN $m[i, j]=q,$

$s[i, j]=k;$

Return m and $s.$

获取构造最优解的信息

Matrix-Chain-Order(p)

$n = \text{length}(p) - 1;$

FOR $i=1$ TO n DO

$m[i, i] = 0;$

FOR $l=2$ TO n DO

FOR $i=1$ TO $n-l+1$ DO

$j=i+l-1;$

$m[i, j] = \infty;$

FOR $k \leftarrow i$ To $j-1$ DO

$q = m[i, k] + m[k+1, j] + p_{i-1} p_k p_j$

 IF $q < m[i, j]$ THEN $m[i, j] = q,$

$s[i, j] = k;$

Return m and $s.$

$S[i, j]$ 记录 $A_i A_{i+1} \dots A_j$ 的最优划分处在 A_k 与 A_{k+1} 之间

构造最优解

Print-Optimal-Parens(s, i, j)

IF $j=i$

THEN Print “ A ” i ;

ELSE Print “(”

Print-Optimal-Parens($s, i, s[i, j]$)

Print-Optimal-Parens($s, s[i, j]+1, j$)

Print “)”



构造最优解

Print-Optimal-Parens(s, i ,

IF $j=i$

THEN Print “ A ” i ;

ELSE Print “(”

$S[i, j]$ 记录 $A_i \dots A_j$ 的最优划分处；

$S[i, S[i, j]]$ 记录 $A_i \dots A_{s[i, j]}$ 的最优划分处；

$S[S[i, j]+1, j]$ 记录 $A_{s[i, j]+1} \dots A_j$ 的最优划分处。

Print-Optimal-Parens($s, i, s[i, j]$)

Print-Optimal-Parens($s, s[i, j]+1, j$)

Print “)”



构造最优解

Print-Optimal-Parens(s, i ,

IF $j=i$

THEN Print “ A ” i ;

ELSE Print “(”

 Print-Optimal-Parens($s, i, s[i, j]$)

 Print-Optimal-Parens($s, s[i, j]+1, j$)

 Print “)”

$s[i, j]$ 记录 $A_i \dots A_j$ 的最优划分处；

$s[i, s[i, j]]$ 记录 $A_i \dots A_{s[i, j]}$ 的最优划分处；

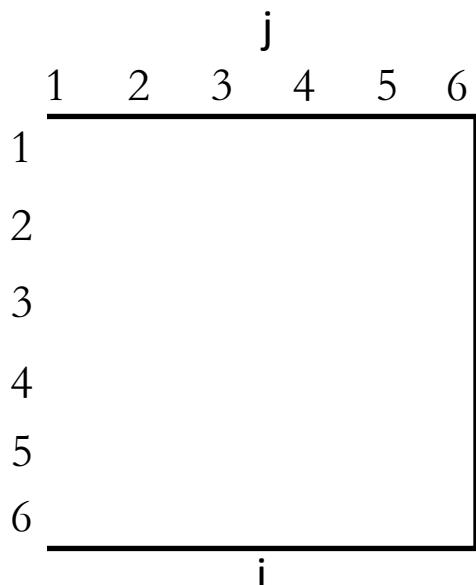
$s[s[i, j]+1, j]$ 记录 $A_{s[i, j]+1} \dots A_j$ 的最优划分处。

调用 Print-Optimal-Parens($s, 1, n$)

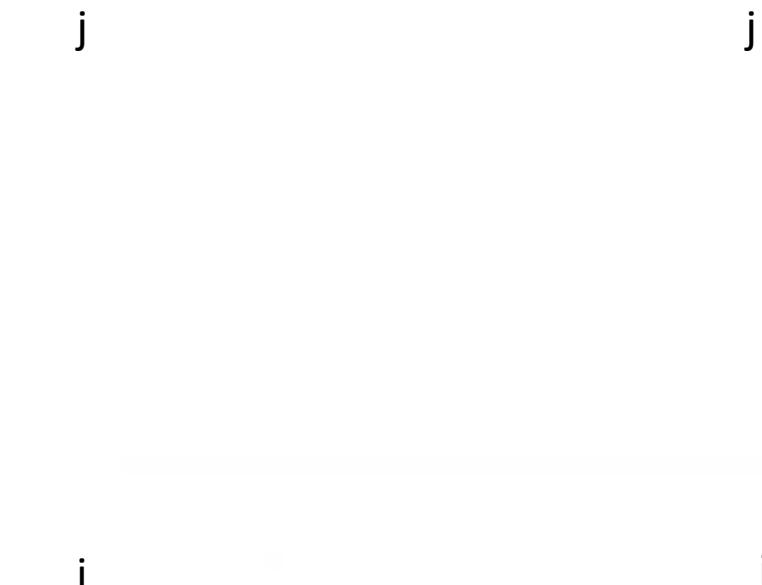
即可输出 $A_{1 \sim n}$ 的优化计算顺序

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序



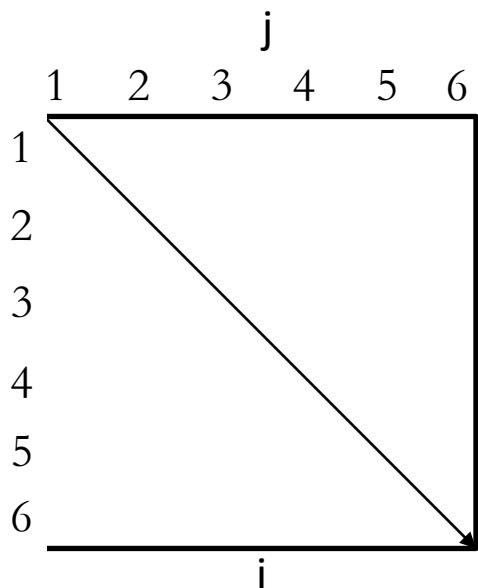
(b) $m[i][j]$



(c) $s[i][j]$

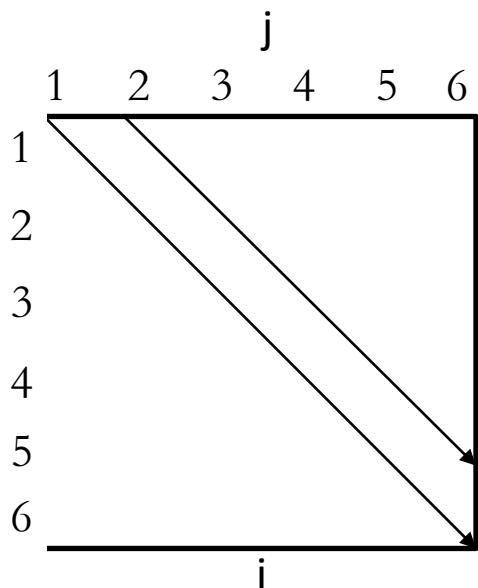
示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序

j

(b) $m[i][j]$

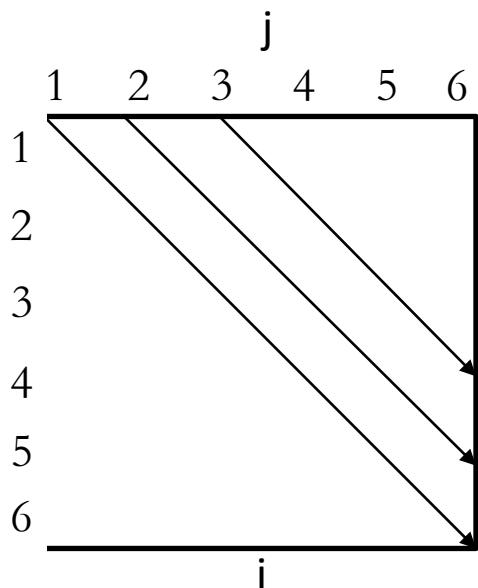
j

(c) $s[i][j]$



示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序



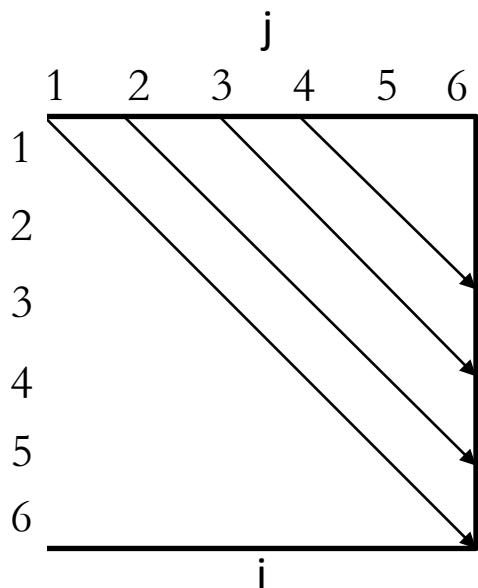
(b) $m[i][j]$



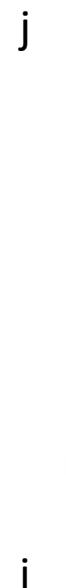
(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序



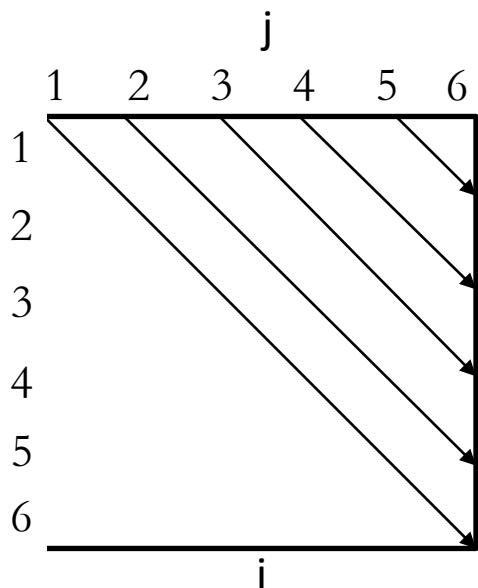
(b) $m[i][j]$



(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序



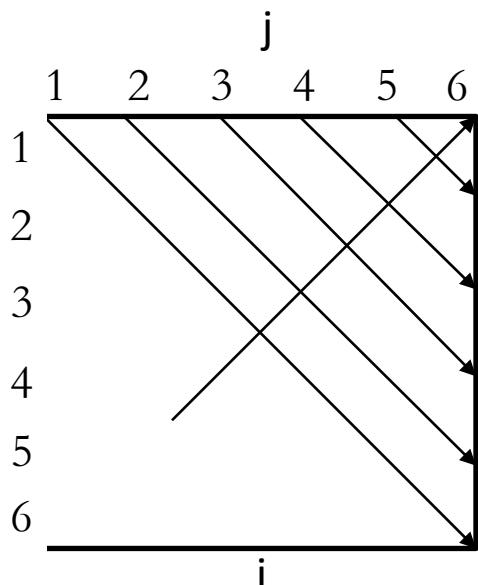
(b) $m[i][j]$



(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序



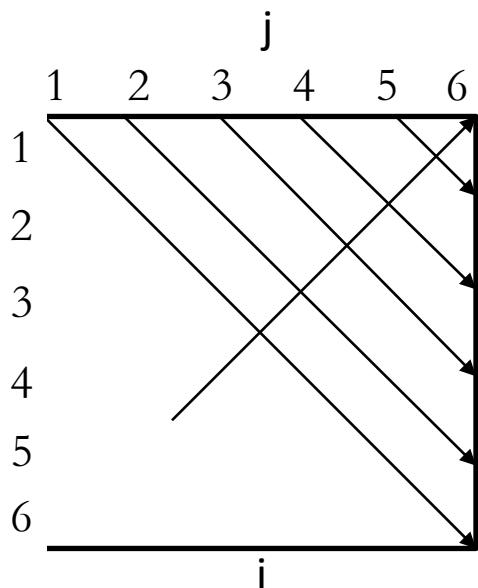
(b) $m[i][j]$



(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序

	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7215	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

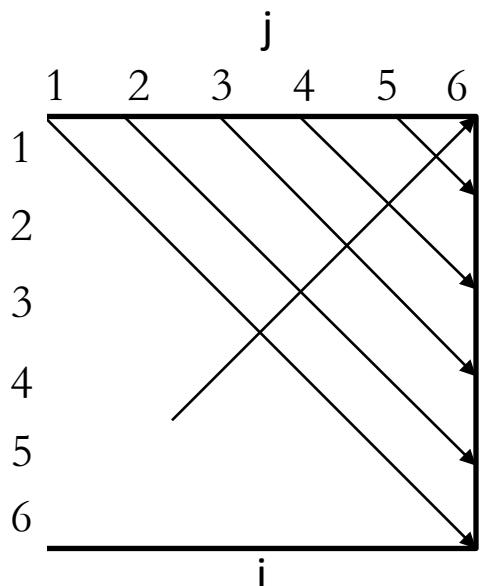
(b) $m[i][j]$

	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



i	1	2	3	4	5	j
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7215	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

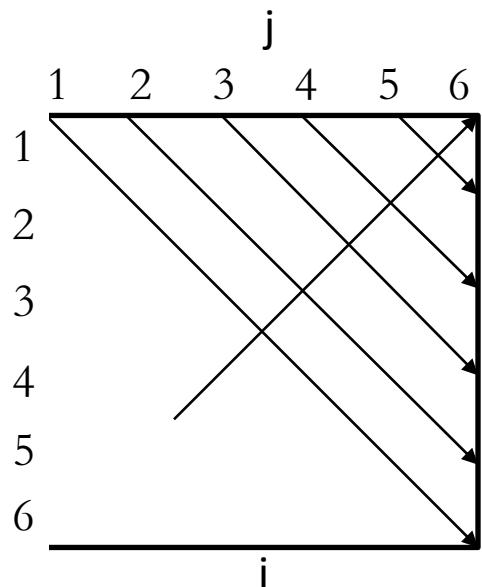
(b) $m[i][j]$

i	1	2	3	4	5	j
1	0	1	1	3	3	3
2		0	4	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



j	1	2	3	4	5	6
i	0	15750	7875	9375	11875	15125
i	2	0	2625	4375	7215	10500
i	3	0	750	2500	5375	
i	4	0	1000	3500		
i	5	0	5000			
i	6	0				

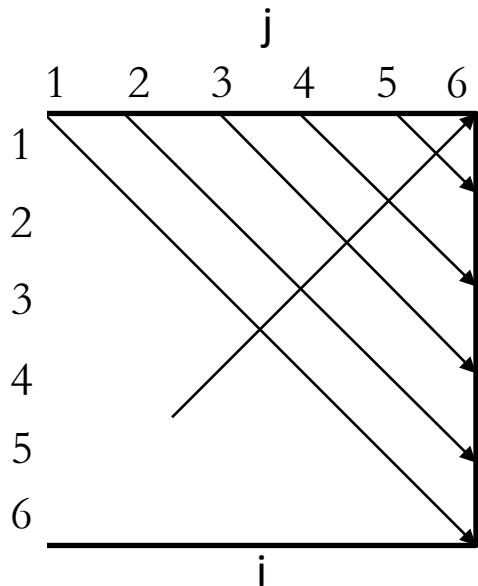
(b) $m[i][j]$

j	1	2	3	4	5	6
i	0	1	1	3	3	3
i	2	0	4	3	3	3
i	3	0	3	3	3	3
i	4	0	4	5		
i	5	0	5			
i	6	0				

(c) $s[i][j]$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



j					
i	1	2	3	4	5
1	0	15750	7875	9375	11875
2		0	2625	4375	7215
3			0	750	2500
4				0	1000
5					0
6					0

(b) $m[i][j]$

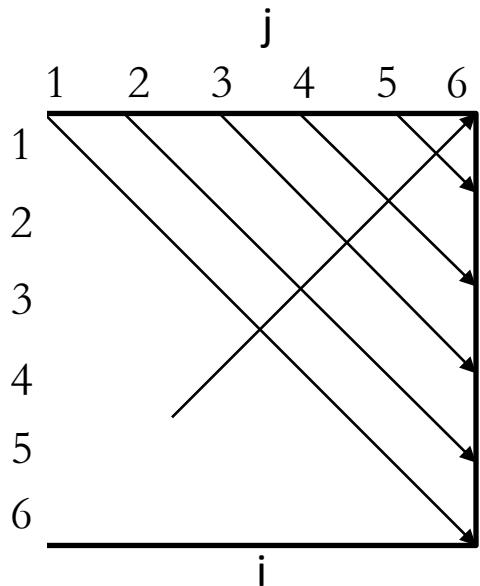
j					
i	1	2	3	4	5
1	0	1	1	3	3
2		0	4	3	3
3			0	3	3
4				0	4
5					0
6					0

(c) $s[i][j]$

$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序

j						
1	2	3	4	5	6	
1	0	15750	7875	9375	11875	15125
2	0	2625	4375	7215	10500	
3	0	750	2500	5375		
4	0	1000	3500			
5	0	5000				
6	0					

(b) $m[i][j]$

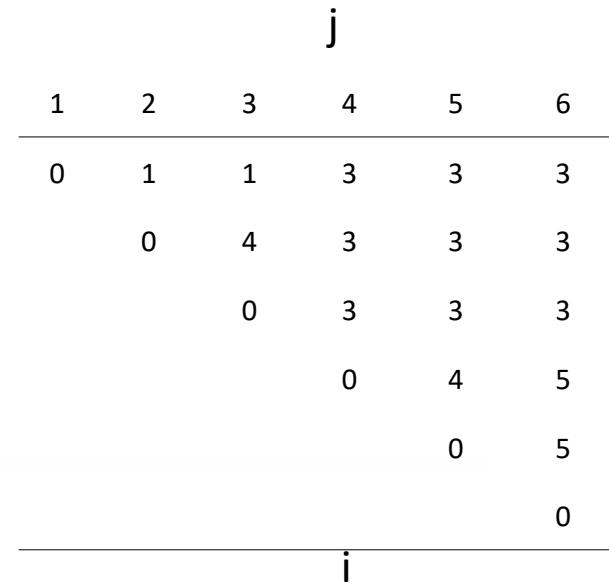
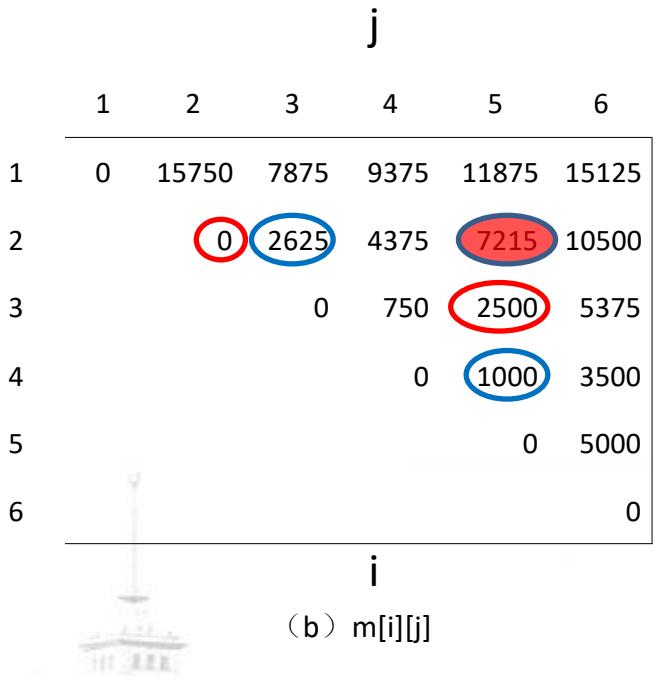
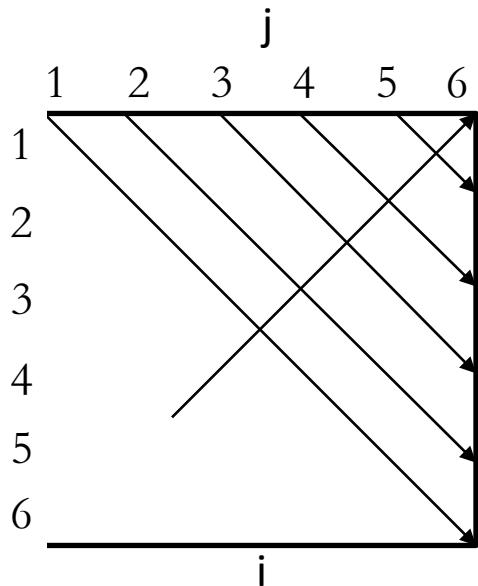
j						
1	2	3	4	5	6	
1	0	1	1	3	3	3
2	0	4	3	3	3	3
3	0	3	3	3	3	3
4	0	4	5			
5	0	5				
6	0					

(c) $s[i][j]$

$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

示例

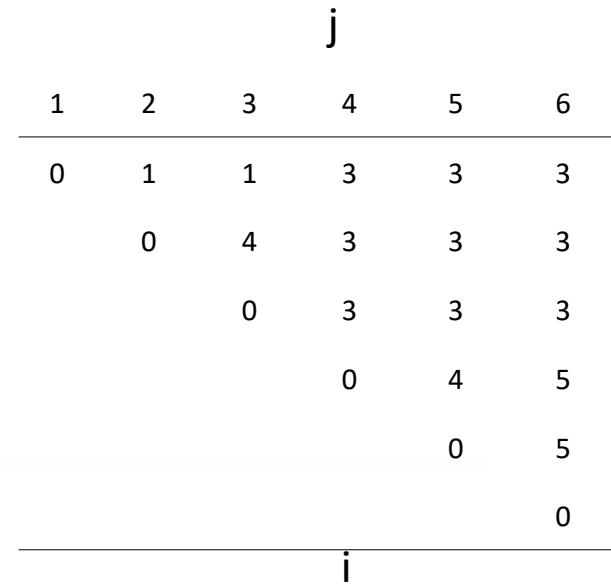
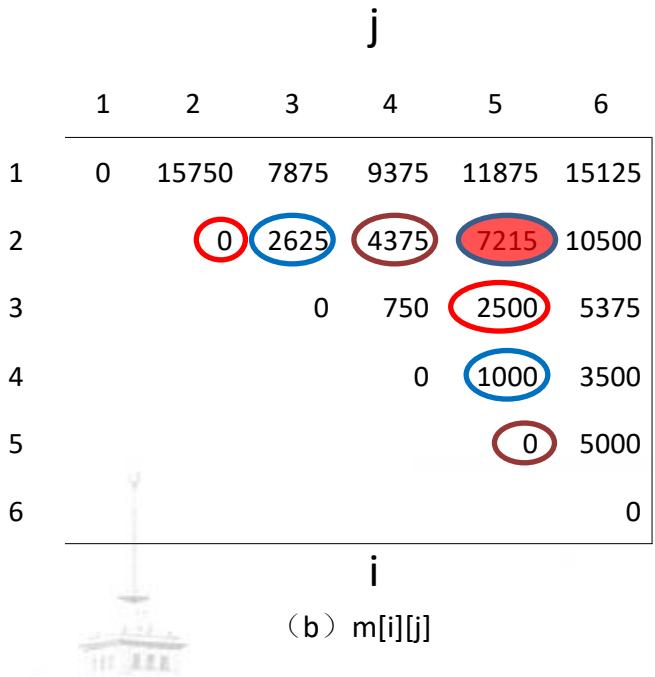
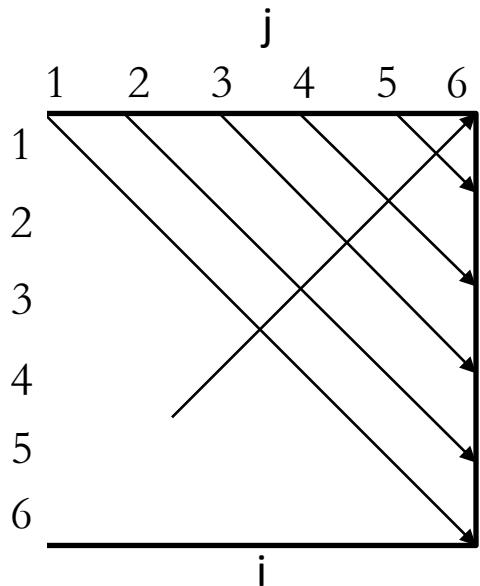
A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

示例

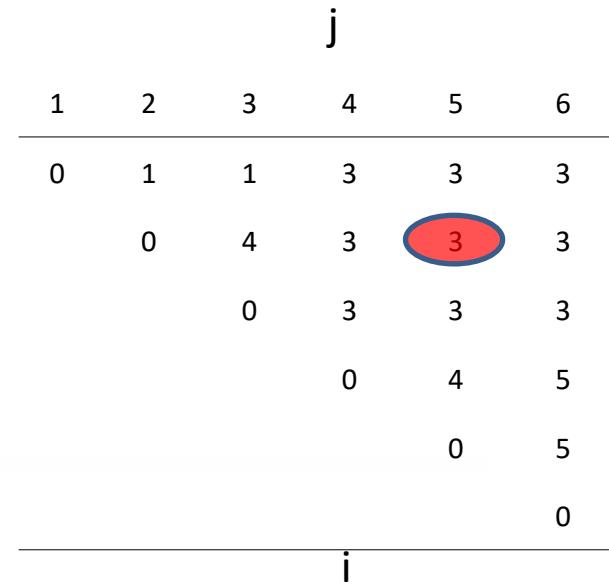
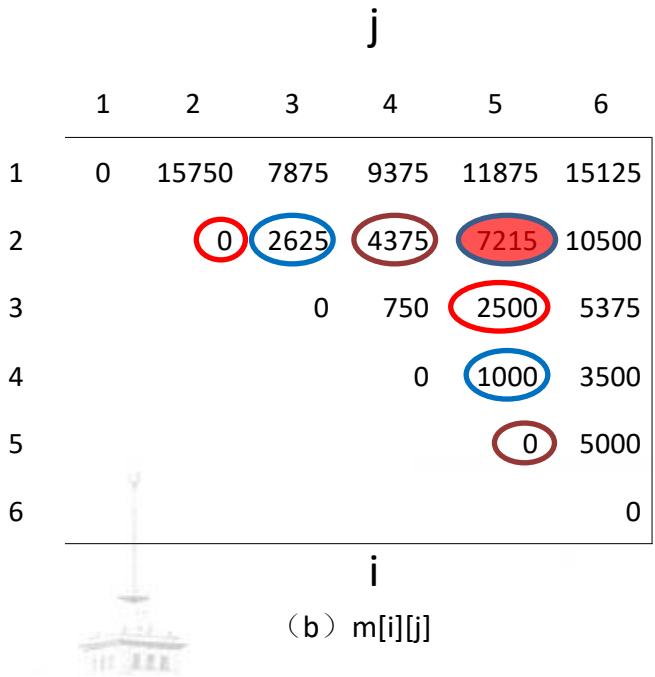
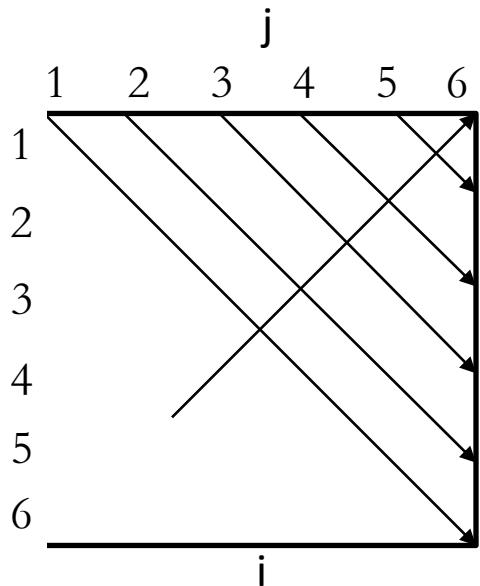
A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

示例

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



$$m[2][5] = \min \begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

动态规划

- 动态规划是一种算法设计的范式 (paradigm) 或思想
不是解决某一具体问题的具体算法
- 理查德 · 贝尔曼 (Richard Bellman) 在1950年代首次提出了这个名字，当时他正为美国兰德公司工作，主要为美国空军和政府项目服务。

“It's impossible to use the word, dynamic, in the pejorative sense...I thought dynamic programming was a good name. It was something not even a Congressman could object to.”——理查德 · 贝尔曼



理查德 · 贝尔曼 (1920年8月26日 – 1984年3月19日)，美国应用数学家，美国国家科学院院士，和动态规划的创始人。

算法复杂性

- 时间复杂性

算法复杂性

- 时间复杂性
 - 计算代价的时间

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$
 - 初始化的时间: $O(n)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$
 - 初始化的时间: $O(n)$
 - 总时间复杂性为: $O(n^3)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$
 - 初始化的时间: $O(n)$
 - 总时间复杂性为: $O(n^3)$
- 空间复杂性

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$
 - 初始化的时间: $O(n)$
 - 总时间复杂性为: $O(n^3)$
- 空间复杂性
 - 使用数组 m 和 S

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (l, i, k) 三层循环, 每层至多 $n-1$ 步
 - $O(n^3)$
 - 初始化的时间: $O(n)$
 - 总时间复杂性为: $O(n^3)$
- 空间复杂性
 - 使用数组 m 和 S
 - 需要空间 $O(n^2)$

思考题

游戏中有一排卡片，每张卡片上都有一个正整数。在每一步中，玩家从排中取出一张卡片，然后该玩家获得等于所取卡片上的数字与它左边和右边卡片上数字的乘积所得得分。不能取出排中的第一张和最后一张卡片。在最后一步之后，排中只会剩下两张卡片。如何按照一定的顺序取卡片，以最小化总得分。

例如，如果排中的卡片数字是10、1、50、20、5。

(1) 玩家可以按照顺序取出1，然后20和50，得分如下：

$$10 * 1 * 50 + 50 * 20 * 5 + 10 * 50 * 5 = 500 + 5000 + 2500 = 8000$$

(2) 如果他按照相反的顺序取卡片，即50、20、1，得分如下：

$$1 * 50 * 20 + 1 * 20 * 5 + 10 * 1 * 5 = 1000 + 100 + 50 = 1150.$$



本讲内容

4. 1 动态规划的原理

4. 2 矩阵乘法问题

4. 3 最长公共子序列问题

4. 4 0-1背包问题

4. 5 最优二分搜索树

这两个物种有多相似



DNA:

AGCCCTAAGGGCTACCTAGCTT



DNA:

GACAGCCTACAAGCGTTAGCTTG

最长公共子序列



DNA:

AGCCCTAA~~GGG~~**GCTACCTAGCTT**



DNA:

GAC~~AGC~~**CTAAC**A~~AGC~~**GTTAGCTT**G

AGCCTAAGCTTAGCTT

问题的定义



问题的定义

- 子序列

- $X=(A, B, C, B, D, B)$

- $Z=(B, C, D, B)$ 是 X 的子序列

- $W=(B, D, A)$ 不是 X 的子序列



问题的定义

- 子序列
 - $X=(A, B, C, B, D, B)$
 - $Z=(B, C, D, B)$ 是 X 的子序列
 - $W=(B, D, A)$ 不是 X 的子序列
- 公共子序列
 - Z 是序列 X 与 Y 的公共子序列如果 Z 是 X 的子序也是 Y 的子序列。



最长公共子序列（LCS）问题

输入： $X = (x_1, x_2, \dots, x_n)$, $Y = (y_1, y_2, \dots, y_m)$

输出： $Z = X$ 与 Y 的最长公共子序列

最长公共子序列结构分析

- 第*i*前缀

- 设 $X=(x_1, x_2, \dots, x_n)$ 是一个序列， X 的第*i*前缀 X_i 是一个序列，定义为 $X_i=(x_1, \dots, x_i)$



最长公共子序列结构分析

- 第*i*前缀

- 设 $X=(x_1, x_2, \dots, x_n)$ 是一个序列， X 的第*i*前缀 X_i 是一个序列，定义为 $X_i=(x_1, \dots, x_i)$

例. $X=(A, B, D, C, A), X_1=(A), X_2=(A, B), X_3=(A, B, D)$



- 优化子结构

定理1（优化子结构） 设 $X=(x_1, \dots, x_m)$ 、
 $Y=(y_1, \dots, y_n)$ 是两个序列， $Z=(z_1, \dots, z_k)$ 是 X 与 Y 的
 LCS ， 我们有：



- 优化子结构

定理1（优化子结构） 设 $X=(x_1, \dots, x_m)$ 、
 $Y=(y_1, \dots, y_n)$ 是两个序列， $Z=(z_1, \dots, z_k)$ 是 X 与 Y 的
 LCS ， 我们有：

(1) 如果 $x_m=y_n$, 则 $z_k=x_m=y_n$, Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的
 LCS , 即, $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.

- 优化子结构

定理1 (优化子结构) 设 $X=(x_1, \dots, x_m)$ 、
 $Y=(y_1, \dots, y_n)$ 是两个序列， $Z=(z_1, \dots, z_k)$ 是 X 与 Y 的
 LCS ， 我们有：

(1) 如果 $x_m=y_n$ ， 则 $z_k=x_m=y_n$ ， Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的
 LCS ， 即， $LCS_{XY} = LCS_{X_{m-1} Y_{n-1}} + <x_m=y_n>$.

(2) 如果 $x_m \neq y_n$ ， 且 $z_k \neq x_m$ ， 则 Z 是 X_{m-1} 和 Y 的
 LCS ， 即 $LCS_{XY} = LCS_{X_{m-1} Y}$

- 优化子结构

定理1 (优化子结构) 设 $X=(x_1, \dots, x_m)$ 、
 $Y=(y_1, \dots, y_n)$ 是两个序列， $Z=(z_1, \dots, z_k)$ 是 X 与 Y 的
 LCS ， 我们有：

- (1) 如果 $x_m=y_n$, 则 $z_k=x_m=y_n$, Z_{k-1} 是 X_{m-1} 和 Y_{n-1} 的
 LCS , 即, $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + <x_m=y_n>$.
- (2) 如果 $x_m \neq y_n$, 且 $z_k \neq x_m$, 则 Z 是 X_{m-1} 和 Y 的
 LCS , 即 $LCS_{XY} = LCS_{X_{m-1}Y}$
- (3) 如果 $x_m \neq y_n$, 且 $z_k \neq y_n$, 则 Z 是 X 与 Y_{n-1} 的 LCS ,
即 $LCS_{XY} = LCS_{XY_{n-1}}$

证明：

(1). $X = \langle x_1, \dots, x_{m-1}, \textcolor{red}{x_m} \rangle$, $Y = \langle y_1, \dots, y_{n-1}, \textcolor{red}{x_m} \rangle$, 则
 $LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle$.

证明：

(1). $X = \langle x_1, \dots, x_{m-1}, \textcolor{red}{x_m} \rangle$, $Y = \langle y_1, \dots, y_{n-1}, \textcolor{red}{x_m} \rangle$, 则

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle.$$

设 $z_k \neq x_m$, 则可加 $x_m = y_n$ 到Z, 得到一个长为 $k+1$ 的
X与Y的公共序列, 与Z是X和Y的LCS矛盾。于是
 $z_k = x_m = y_n$ 。

证明：

(1). $X = \langle x_1, \dots, x_{m-1}, \textcolor{red}{x_m} \rangle$, $Y = \langle y_1, \dots, y_{n-1}, \textcolor{red}{x_m} \rangle$, 则

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle.$$

设 $z_k \neq x_m$, 则可加 $x_m = y_n$ 到Z, 得到一个长为 $k+1$ 的
X与Y的公共序列, 与Z是X和Y的LCS矛盾。于是
 $z_k = x_m = y_n$ 。

现在证明 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的LCS。显然 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的公共序列。我们需要证明 Z_{k-1} 是LCS。

证明：

(1). $X = \langle x_1, \dots, x_{m-1}, \textcolor{red}{x_m} \rangle$, $Y = \langle y_1, \dots, y_{n-1}, \textcolor{red}{x_m} \rangle$, 则

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle.$$

设 $z_k \neq x_m$, 则可加 $x_m = y_n$ 到 Z , 得到一个长为 $k+1$ 的 X 与 Y 的公共序列, 与 Z 是 X 和 Y 的 LCS 矛盾。于是 $z_k = x_m = y_n$ 。

现在证明 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS 。显然 Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的公共序列。我们需要证明 Z_{k-1} 是 LCS 。

设不然, 则存在 X_{m-1} 与 Y_{n-1} 的公共子序列 W , W 的长大于 $k-1$ 。增加 $x_m = y_n$ 到 W , 我们得到一个长大于 k 的 X 与 Y 的公共序列, 与 Z 是 LCS 矛盾。于是, Z_{k-1} 是 X_{m-1} 与 Y_{n-1} 的 LCS .

(2) $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$, $Y = \langle y_1, \dots, y_{n-1}, y_n \rangle$,
 $x_m \neq y_n$, $x_m \neq z_k$, 则 $LCS_{XY} = LCS_{X_{m-1}Y}$



(2) $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$, $Y = \langle y_1, \dots, y_{n-1}, y_n \rangle$,
 $x_m \neq y_n$, $z_k \neq x_m$, 则 $LCS_{XY} = LCS_{X_{m-1}Y}$

由于 $z_k \neq x_m$, Z 是 X_{m-1} 与 Y 的公共子序列。我们来证 Z 是 X_{m-1} 与 Y 的 LCS 。设 X_{m-1} 与 Y 有一个公共子序列 W , W 的长大于 k , 则 W 也是 X 与 Y 的公共子序列, 与 Z 是 LCS 矛盾。

(2) $X = \langle x_1, \dots, x_{m-1}, x_m \rangle$, $Y = \langle y_1, \dots, y_{n-1}, y_n \rangle$,
 $x_m \neq y_n$, $z_k \neq x_m$, 则 $LCS_{XY} = LCS_{X_{m-1}Y}$

由于 $z_k \neq x_m$, Z 是 X_{m-1} 与 Y 的公共子序列。我们来证 Z 是 X_{m-1} 与 Y 的 LCS 。设 X_{m-1} 与 Y 有一个公共子序列 W , W 的长大于 k , 则 W 也是 X 与 Y 的公共子序列, 与 Z 是 LCS 矛盾。

(3) 同(2)可证。

X 和 Y 的LCS的优化解结构为

$$LCS_{XY} = LCS_{X_{m-1}Y_{n-1}} + \langle x_m = y_n \rangle \quad \text{if } x_m = y_n$$

$$LCS_{XY} = LCS_{X_{m-1}Y} \quad \text{if } x_m \neq y_n, z_k \neq x_m$$

$$LCS_{XY} = LCS_{XY_{n-1}} \quad \text{if } x_m \neq y_n, z_k \neq y_n$$

- 子问题重叠性

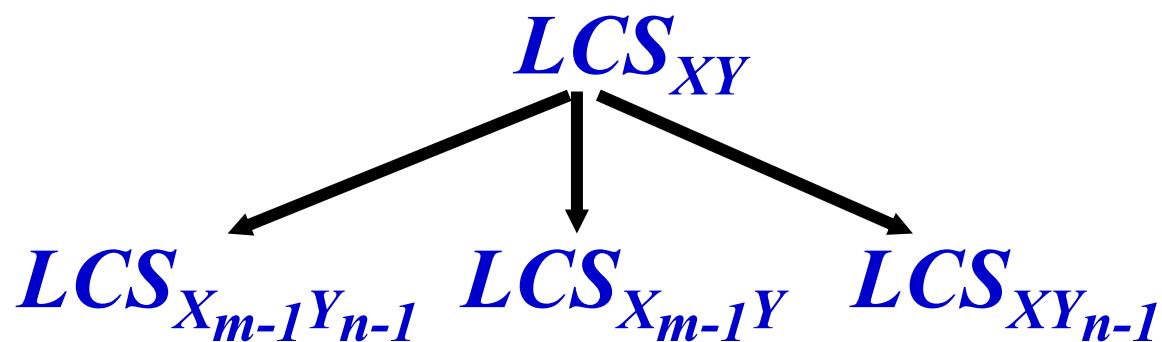


- 子问题重叠性

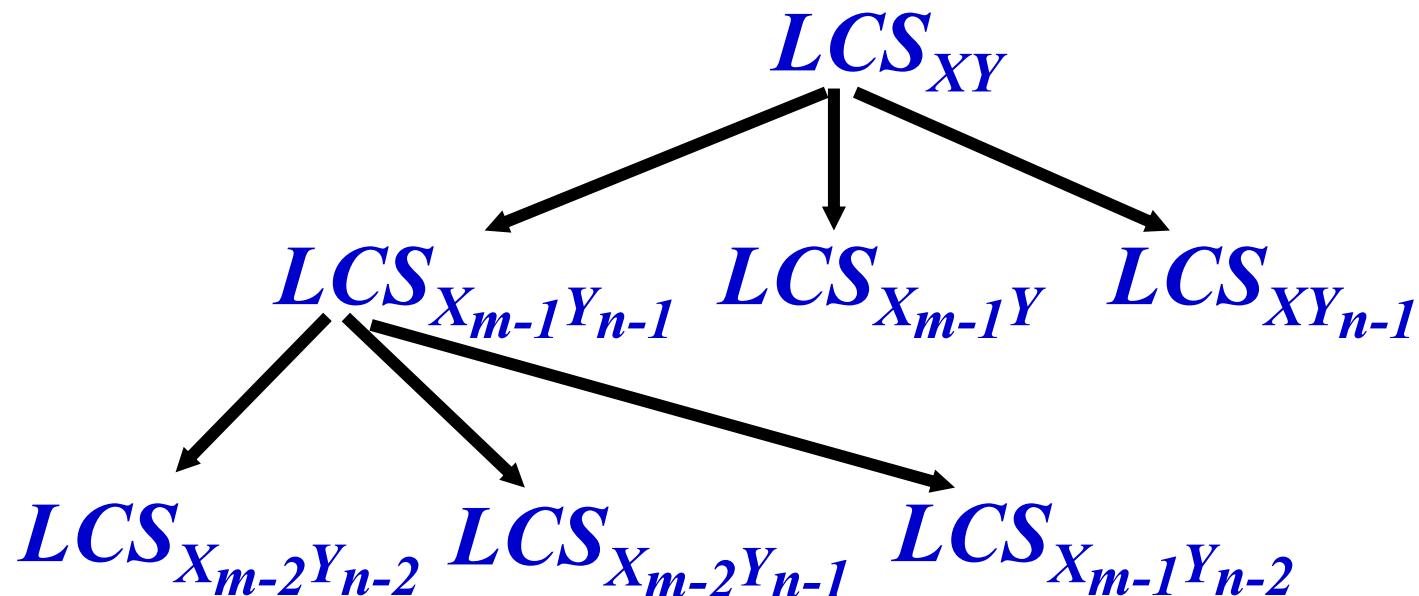
LCS_{XY}



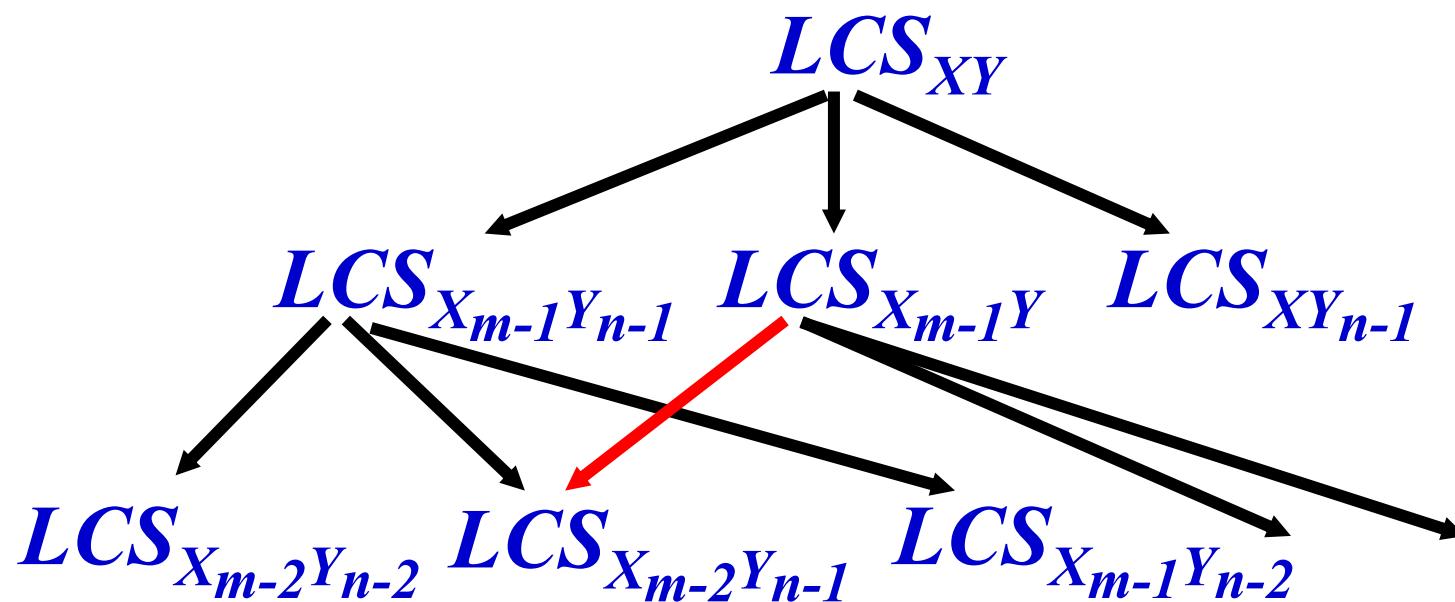
- 子问题重叠性



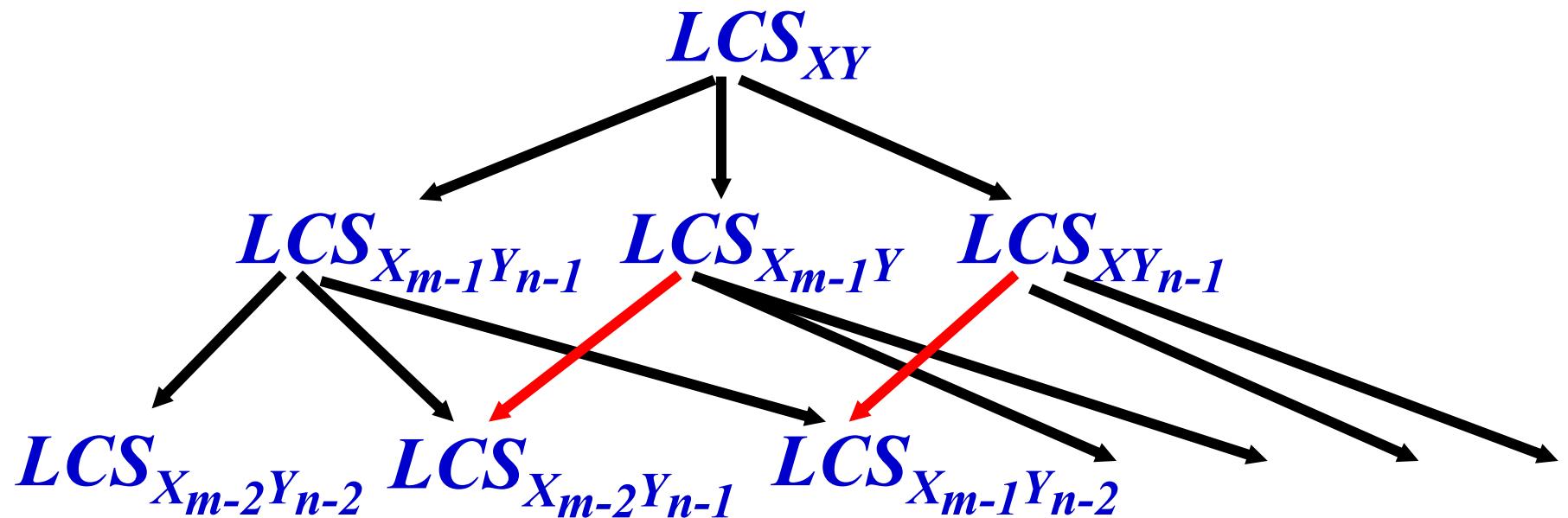
- 子问题重叠性



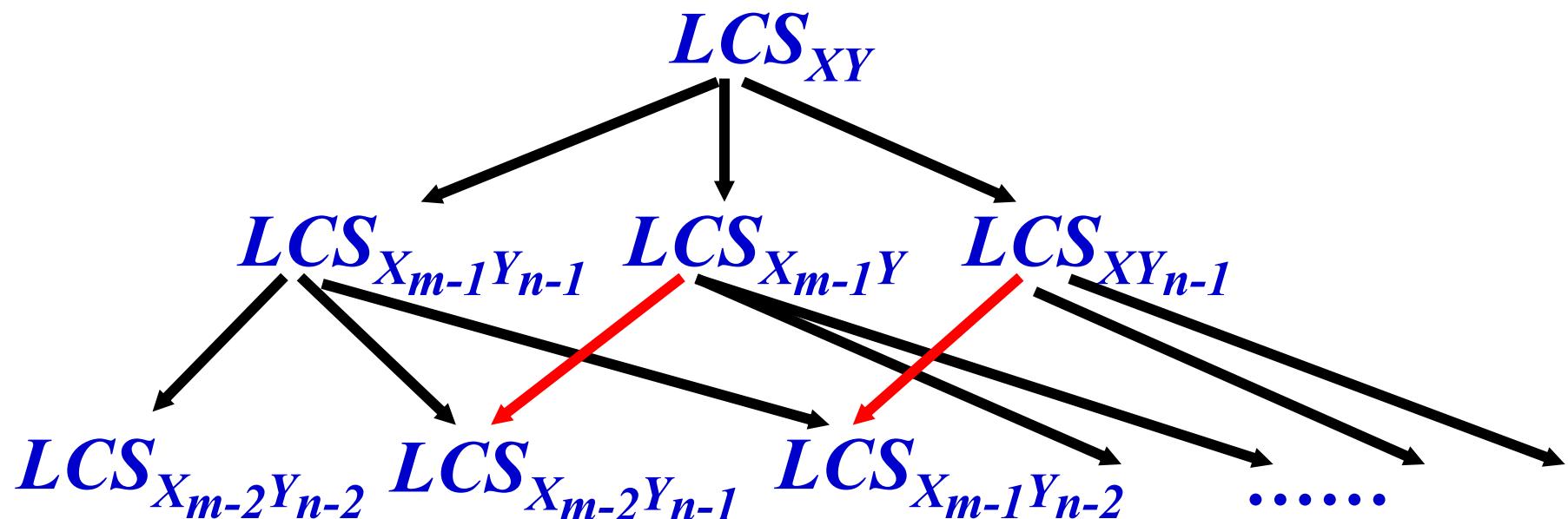
- 子问题重叠性



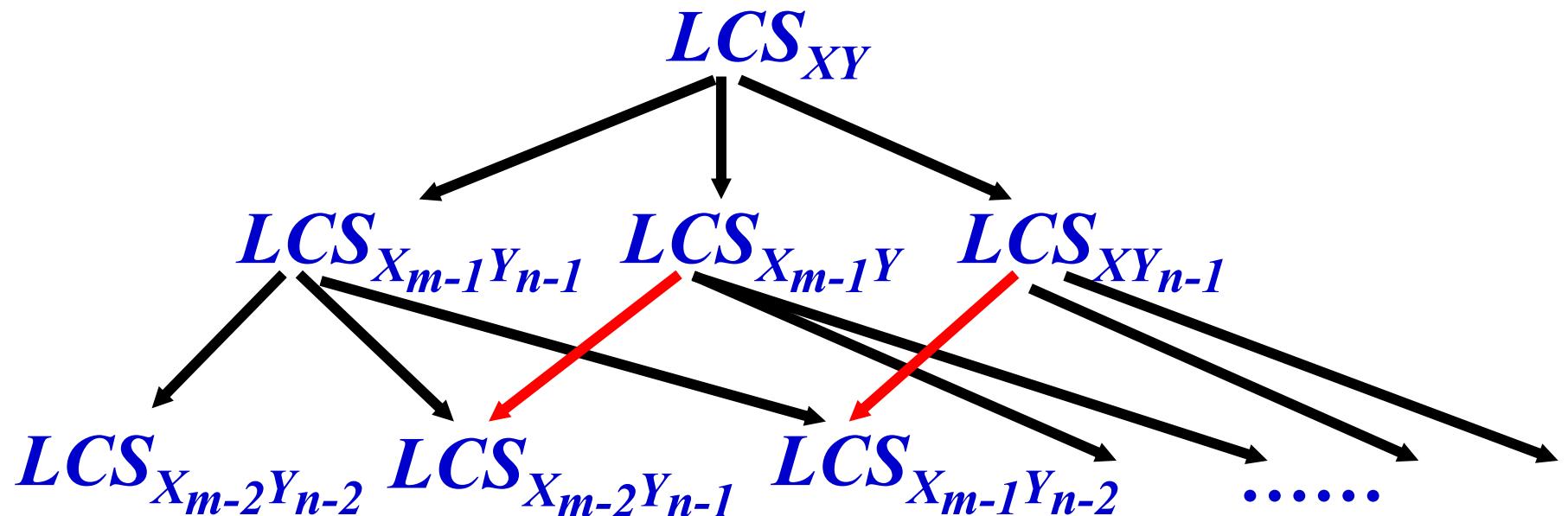
- 子问题重叠性



- 子问题重叠性



- 子问题重叠性



LCS问题具有子问题重叠性

建立LCS长度的递归方程

- $C[i, j]$ = X_i 与 Y_j 的LCS的长度
- LCS长度的递归方程

$$C[i, j] = 0$$

if $i=0$ 或 $j=0$

$$C[i, j] = C[i-1, j-1] + 1$$

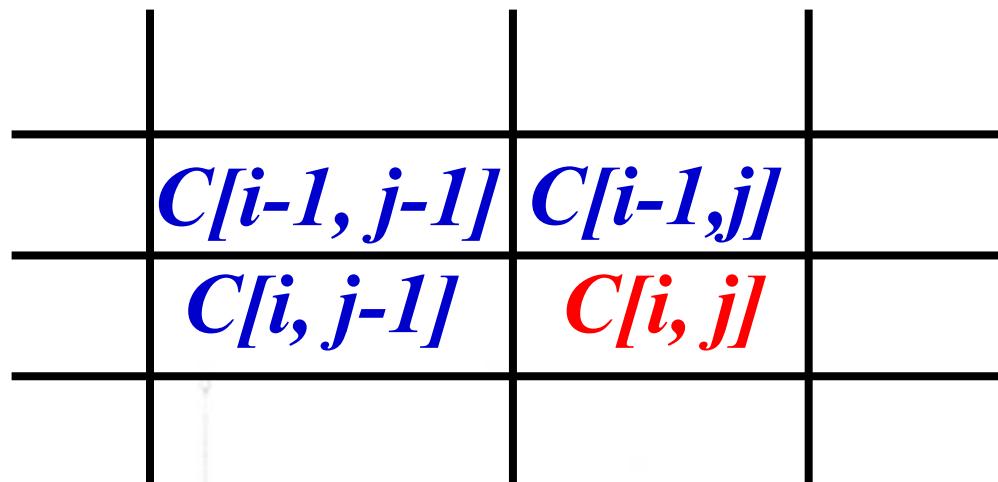
if $i, j > 0$ 且 $x_i = y_j$

$$C[i, j] = \text{Max}(C[i, j-1], C[i-1, j])$$

if $i, j > 0$ 且 $x_i \neq y_j$

自底向上计算LCS的长度

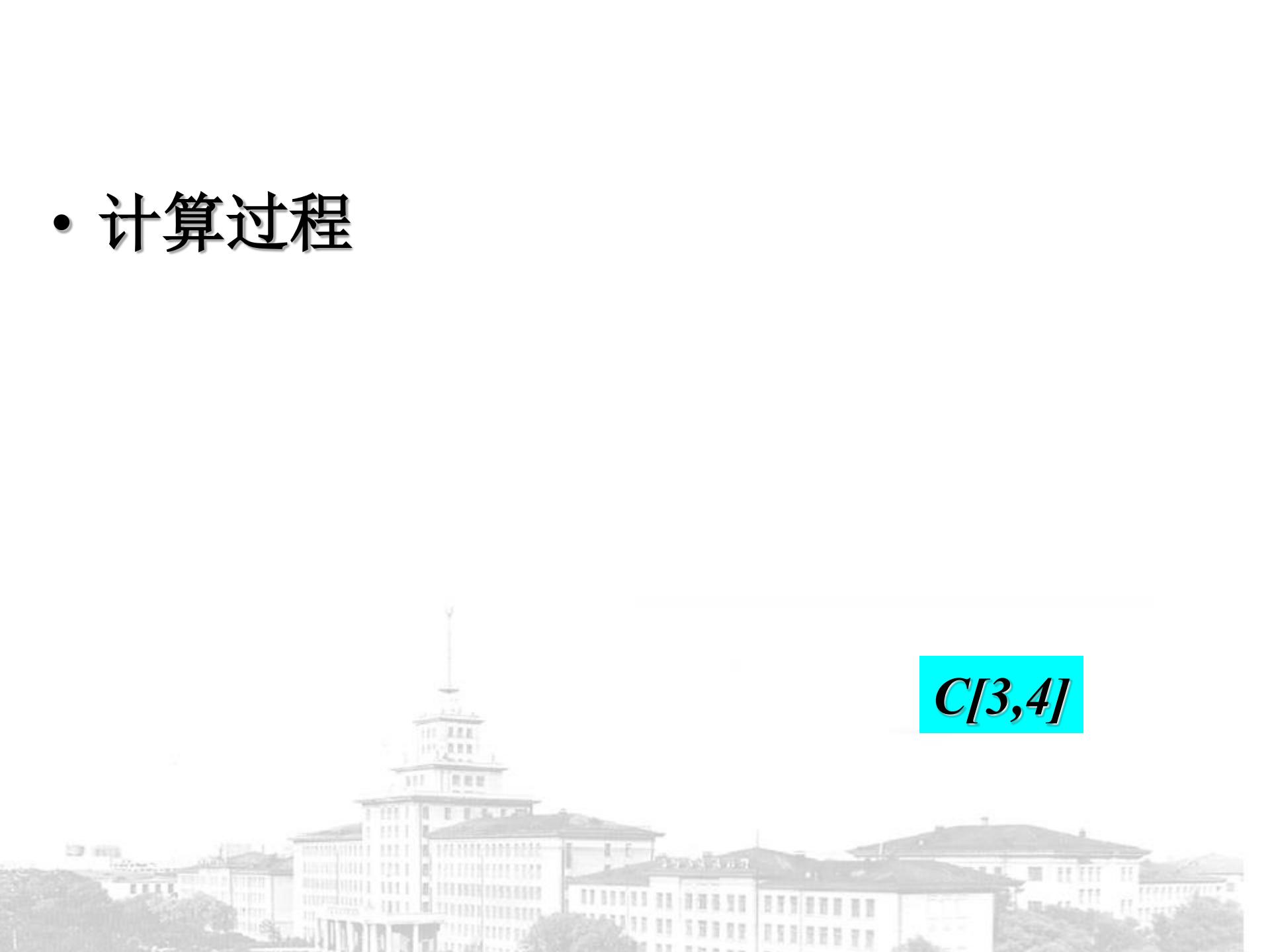
- 基本思想



- 计算过程



- 计算过程



$C[3,4]$

- 计算过程

$C[2,3]$ $C[2,4]$

$C[3,3]$ $C[3,4]$



- 计算过程

 $C[1,3]$ $C[1,4]$ $C[2,3]$ $C[2,4]$ $C[3,3]$ $C[3,4]$ 

- 计算过程

 $C[0,3]$ $C[0,4]$ $C[1,3]$ $C[1,4]$ $C[2,3]$ $C[2,4]$ $C[3,3]$ $C[3,4]$ 

- 计算过程

$C[0,3]$	$C[0,4]$
----------	----------

$C[1,3]$	$C[1,4]$
----------	----------

$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------

$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------



- 计算过程

 $C[0,3]$ $C[0,4]$ $C[1,2]$ $C[1,3]$ $C[1,4]$ $C[2,2]$ $C[2,3]$ $C[2,4]$ $C[3,2]$ $C[3,3]$ $C[3,4]$ 

- 计算过程

$C[0,2]$	$C[0,3]$	$C[0,4]$
----------	----------	----------

$C[1,2]$	$C[1,3]$	$C[1,4]$
----------	----------	----------

$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------

$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------



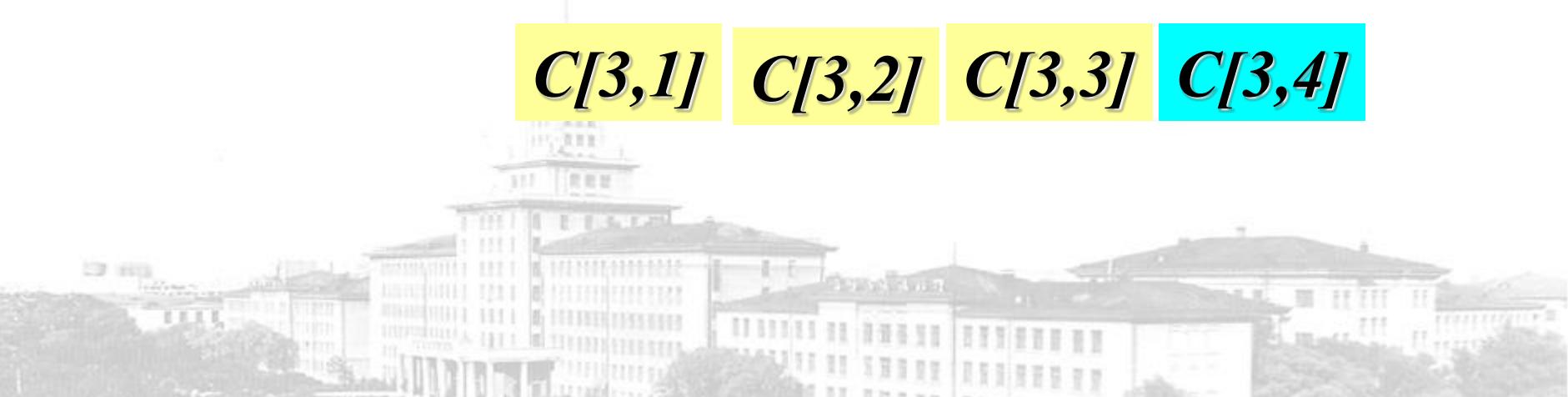
- 计算过程

$C[0,2]$	$C[0,3]$	$C[0,4]$
----------	----------	----------

$C[1,2]$	$C[1,3]$	$C[1,4]$
----------	----------	----------

$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------	----------

$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------	----------



- 计算过程

$C[0,2]$	$C[0,3]$	$C[0,4]$	
$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
----------	----------	----------	----------

$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
----------	----------	----------	----------

$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------	----------

$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------	----------



- 计算过程

$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
----------	----------	----------	----------

$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
----------	----------	----------	----------

$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------	----------	----------

$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------	----------	----------



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
----------	----------	----------	----------	----------

$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
----------	----------	----------	----------	----------

$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
----------	----------	----------	----------	----------

$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$
----------	----------	----------	----------	----------



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算过程

$C[0,0]$	$C[0,1]$	$C[0,2]$	$C[0,3]$	$C[0,4]$
$C[1,0]$	$C[1,1]$	$C[1,2]$	$C[1,3]$	$C[1,4]$
$C[2,0]$	$C[2,1]$	$C[2,2]$	$C[2,3]$	$C[2,4]$
$C[3,0]$	$C[3,1]$	$C[3,2]$	$C[3,3]$	$C[3,4]$



- 计算LCS长度的算法
 - 数据结构



- 计算LCS长度的算法
 - 数据结构

$C[0:m, 0:n]$: $C[i, j]$ 是 X_i 与 Y_j 的LCS的长度

- 计算LCS长度的算法
 - 数据结构

$C[0:m,0:n]$: $C[i,j]$ 是 X_i 与 Y_j 的LCS的长度

$B[1:m,1:n]$: $B[i,j]$ 是指针，指向计算 $C[i,j]$ 时所选择的子问题的优化解所对应的C表的表项

LCS-length(X, Y)

$m \leftarrow \text{length}(X); n \leftarrow \text{length}(Y);$

For $i \leftarrow 1$ **To** m **Do** $C[i,0] \leftarrow 0;$

For $j \leftarrow 1$ **To** n **Do** $C[0,j] \leftarrow 0;$

For $i \leftarrow 1$ **To** m **Do**

For $j \leftarrow 1$ **To** n **Do**

If $x_i = y_j$

Then $C[i,j] \leftarrow C[i-1,j-1] + 1;$ $B[i,j] \leftarrow “\nwarrow”;$

Else If $C[i-1,j] \geq C[i,j-1]$

Then $C[i,j] \leftarrow C[i-1,j];$ $B[i,j] \leftarrow “\uparrow”;$

Else $C[i,j] \leftarrow C[i,j-1];$ $B[i,j] \leftarrow “\leftarrow”;$

Return C and $B.$

构造优化解

- 基本思想
 - 从 $B[m, n]$ 开始按指针搜索
 - 若 $B[i, j]=“↖”$ ， 则 $x_i=y_j$ 是LCS的一个元素
 - 如此找到的 “LCS”是 X 与 Y 的LCS

Print-LCS(B, X, i, j)

IF $i=0$ or $j=0$ THEN Return;

IF $B[i, j] = “\nwarrow”$

THEN Print-LCS($B, X, i-1, j-1$);

Print x_i ;

ELSE If $B[i, j] = “\uparrow”$

THEN Print-LCS($B, X, i-1, j$);

ELSE Print-LCS($B, X, i, j-1$).

Print-LCS(B, X, i, j)

IF $i=0$ or $j=0$ THEN Return;

IF $B[i, j]=“↖”$

THEN Print-LCS($B, X, i-1, j-1$);

 Print x_i ;

ELSE If $B[i, j]=“↑”$

 THEN Print-LCS($B, X, i-1, j$);

 ELSE Print-LCS($B, X, i, j-1$).

Print-LCS(B, X, length(X), length(Y))

可打印出 X 与 Y 的*LCS*。

Demonstration

Demonstration

j	0	1	2	3	4	5	6	7	8	9	10
i	y_j	a	m	p	u	t	a	t	i	o	n
0	x_i	0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	3↖
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	3↑
6	i	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↖	3↑
7	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	4↖
8	g	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	4↑

Demonstration

j	0	1	2	3	4	5	6	7	8	9	10
i	y_j	a	m	p	u	t	a	t	i	o	n
0	x_i	0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↑	1↑	1↑	1↑	1↑	1↑	1↑
3	a	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	3↑
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	3↑
6	i	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	3↑
7	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	4↑
8	g	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	4↑

p a i n

算法复杂性

- 时间复杂性

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (i, j) 两层循环, i 循环 m 步, j 循环 n 步
 - $O(mn)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (i, j) 两层循环, i 循环 m 步, j 循环 n 步
 - $O(mn)$
 - 初始化的时间: $O(m+n)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (i, j) 两层循环, i 循环 m 步, j 循环 n 步
 - $O(mn)$
 - 初始化的时间: $O(m+n)$
 - 总时间复杂性为: $O(mn)$

算法复杂性

- 时间复杂性
 - 计算代价的时间
 - (i, j) 两层循环, i 循环 m 步, j 循环 n 步
 - $O(mn)$
 - 初始化的时间: $O(m+n)$
 - 总时间复杂性为: $O(mn)$
- 空间复杂性
 - 使用数组 C 和 B
 - 需要空间 $O(mn)$

本讲内容

4. 1 动态规划的原理

4. 2 矩阵乘法问题

4. 3 最长公共子序列问题

4. 4 0-1背包问题

4. 5 最优二分搜索树

问题的定义

给定 n 种物品和一个背包，物品 i 的重量是 w_i ，价值 v_i ，背包容量为 C ，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？



问题的定义

给定 n 种物品和一个背包，物品 i 的重量是 w_i ，价值 v_i ，背包容量为 C ，问如何选择装入背包的物品，使装入背包中的物品的总价值最大？

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

- 输入: $C > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$
- 输出: (x_1, x_2, \dots, x_n) , $x_i \in \{0, 1\}$, 满足
 $\sum_{1 \leq i \leq n} w_i x_i \leq C, \quad \sum_{1 \leq i \leq n} v_i x_i$ 最大



- 输入: $C > 0, w_i > 0, v_i > 0, 1 \leq i \leq n$
- 输出: (x_1, x_2, \dots, x_n) , $x_i \in \{0, 1\}$, 满足
 $\sum_{1 \leq i \leq n} w_i x_i \leq C, \quad \sum_{1 \leq i \leq n} v_i x_i$ 最大

等价的整数规划问题

$$\max \sum_{1 \leq i \leq n} v_i x_i$$

$$\sum_{1 \leq i \leq n} w_i x_i \leq C$$

$$x_i \in \{0, 1\}, \quad 1 \leq i \leq n$$

优化解结构的分析

定理 (优化子结构) 如果 (y_1, y_2, \dots, y_n) 是0-1背包问题的优化解，则 (y_2, \dots, y_n) 是如下子问题的优化解：



优化解结构的分析

定理 (优化子结构) 如果 (y_1, y_2, \dots, y_n) 是 0-1 背包问题的优化解，则 (y_2, \dots, y_n) 是如下子问题的优化解：

$$\begin{aligned} & \max \sum_{2 \leq i \leq n} v_i x_i \\ & \sum_{2 \leq i \leq n} w_i x_i \leq C - w_1 y_1 \\ & x_i \in \{0, 1\}, \quad 2 \leq i \leq n \end{aligned}$$

优化解结构的分析

定理 (优化子结构) 如果 (y_1, y_2, \dots, y_n) 是 0-1 背包问题的优化解，则 (y_2, \dots, y_n) 是如下子问题的优化解：

$$\begin{aligned} & \max \sum_{2 \leq i \leq n} v_i x_i \\ & \sum_{2 \leq i \leq n} w_i x_i \leq C - w_1 y_1 \\ & x_i \in \{0, 1\}, \quad 2 \leq i \leq n \end{aligned}$$

证明：如果 (y_2, \dots, y_n) 不是子问题优化解，则存在 (z_2, \dots, z_n) 是子问题更优的解。于是， (y_1, z_2, \dots, z_n) 是原问题比 (y_1, y_2, \dots, y_n) 更优解，矛盾。

建立优化解代价的递归方程

- 设子问题

$$\max \sum_{i \leq k \leq n} v_k x_k$$

$$\sum_{i \leq k \leq n} w_k x_k \leq j$$

$$x_k \in \{0, 1\}, i \leq k \leq n$$

的最优解代价为 $m(i, j)$.



建立优化解代价的递归方程

- 设子问题

$$\begin{aligned} & \max \sum_{i \leq k \leq n} v_k x_k \\ & \sum_{i \leq k \leq n} w_k x_k \leq j \\ & x_k \in \{0, 1\}, \quad i \leq k \leq n \end{aligned}$$

的最优解代价为 $m(i, j)$.

即 $m(i, j)$ 是背包容量为 j , 可选物品为 $i, i+1, \dots, n$ 时问题最优解的代价.

递归方程

$$m(i, j) = m(i+1, j) \quad 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i) + v_i\} \quad j \geq w_i$$

初始值

$$m(n, j) = 0 \quad 0 \leq j < w_n$$

$$m(n, j) = v_n \quad j \geq w_n$$

自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$

$m(2, C-w_1)$

$m(2, C)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$

$m(2, 1)$

...

...

...

$m(2, C-w_1)$

...

$m(2, C)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$

$m(2, 1)$

...

...

...

$m(2, C-w_1)$

...

$m(2, C)$

$m(3, C-w_2)$

$m(3, C)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(2, 1)$...

...

...

$m(2, C-w_1)$...

$m(2, C)$

$m(3, C-w_1-w_2)$

$m(3, C-w_2)$

$m(3, C-w_1)$

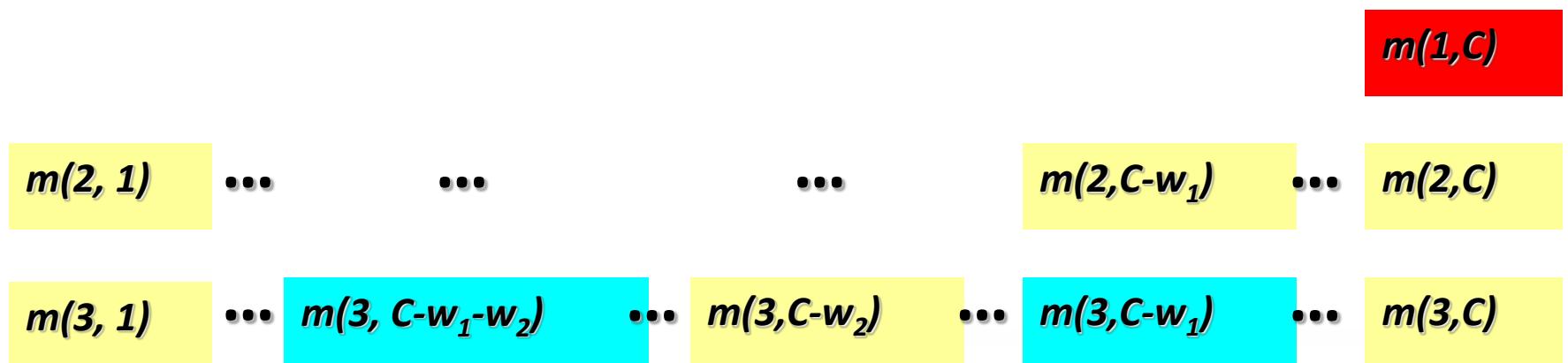
$m(3, C)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$

$m(2, 1)$

...

...

...

$m(2, C-w_1)$

...

$m(2, C)$

$m(3, 1)$

...

$m(3, C-w_1-w_2)$

...

$m(3, C-w_2)$

...

$m(3, C-w_1)$

...

$m(3, C)$

$m(4, 1)$

...

$m(4, C-w_1-w_2)$

..

$m(4, C-w_2)$

...

$m(4, C-w_1)$

...

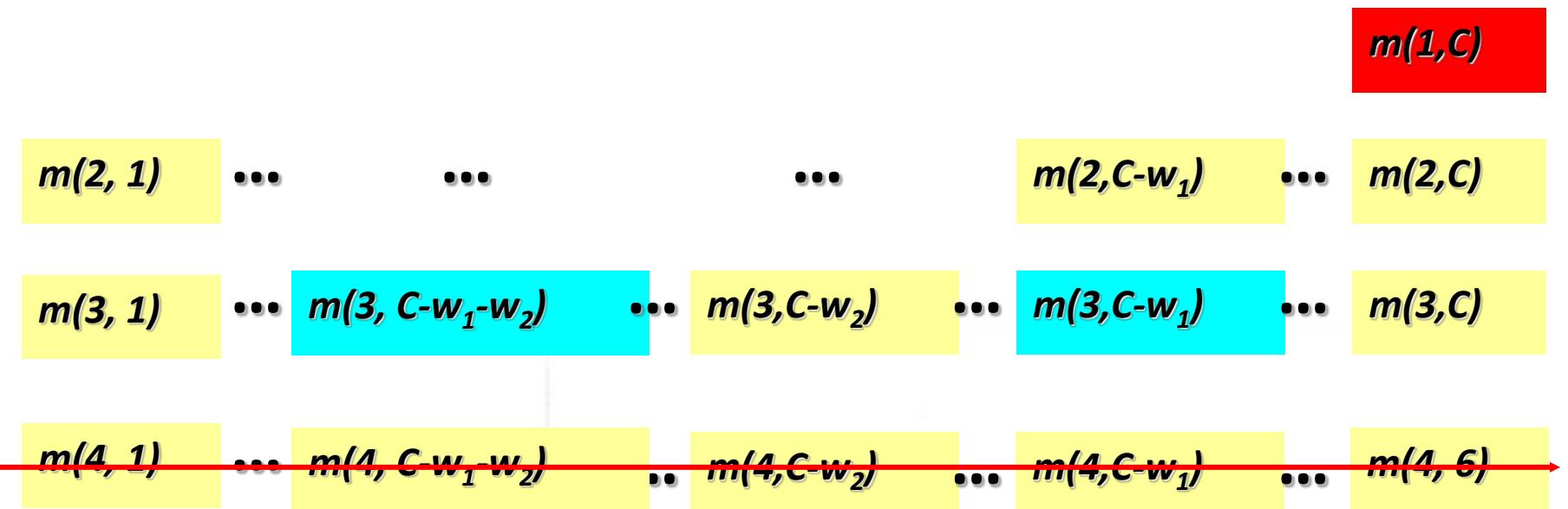
$m(4, 6)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=整数, n=4$

$m(1, C)$

$m(2, 1)$

...

...

...

$m(2, C-w_1)$

...

$m(2, C)$

$m(3, 1)$

...

$m(3, C-w_1-w_2)$

...

$m(3, C-w_2)$

...

$m(3, C-w_1)$

...

$m(3, C)$

$m(4, 1)$

...

$m(4, C-w_1-w_2)$

..

$m(4, C-w_2)$

...

$m(4, C-w_1)$

...

$m(4, 6)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$

$m(1, C)$

$m(2, 1)$

...

...

...

$m(2, C-w_1)$

...

$m(2, C)$

$m(3, 1)$

...

$m(3, C-w_1-w_2)$

...

$m(3, C-w_2)$

...

$m(3, C-w_1)$

...

$m(3, C)$

$m(4, 1)$

...

$m(4, C-w_1-w_2)$

...

$m(4, C-w_2)$

...

$m(4, C-w_1)$

...

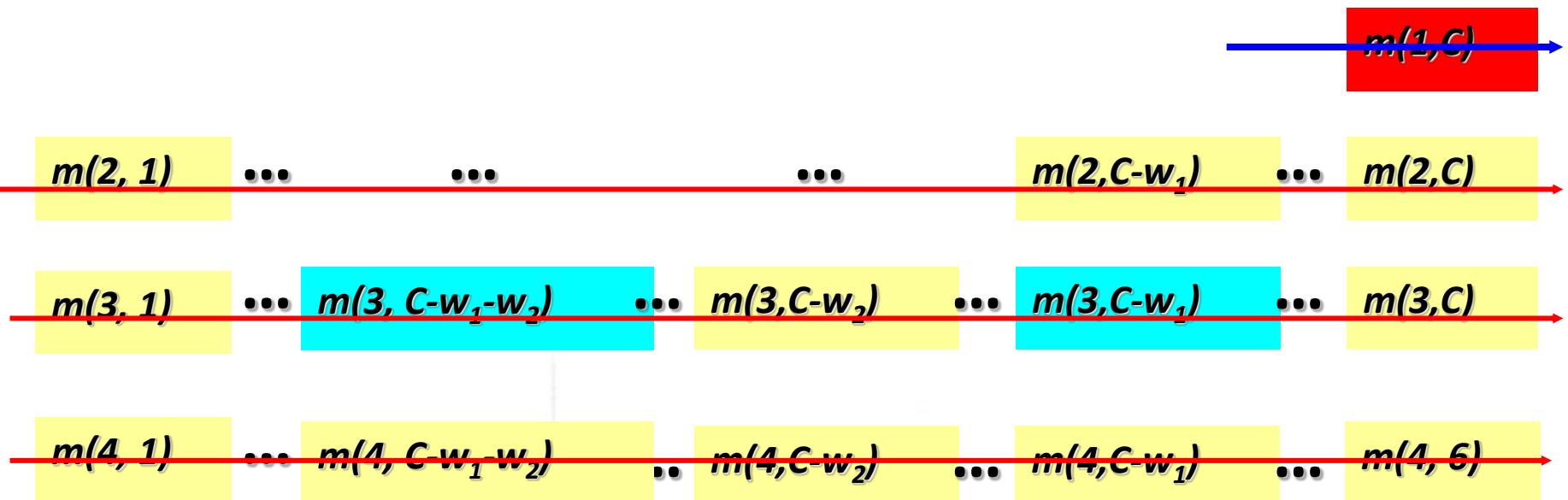
$m(4, 6)$



自底向上计算优化解的代价

计算 $m(i, j)$ 需要
 $m(i+1, j-w_i)$ 和 $m(i+1, j)$

令 $w_i=$ 整数, $n=4$



•算法

For $j=0$ To $\min(w_n-1, C)$ Do

$m[n, j] = 0;$

For $j=w_n$ To C Do

$m[n, j] = v_n;$

For $i=n-1$ To 2 Do

For $j=0$ To $\min(w_i-1, C)$ Do

$m[i, j] = m[i+1, j];$

For $j=w_i$ To C Do

$m[i, j] = \max\{m[i+1, j], m[i+1, j-w_i] + v_i\};$

If $C < w_1$

Then $m[1, C] = m[2, C];$

Else $m[1, C] = \max\{m[2, C], m[2, C-w_1] + v_1\};$

构造优化解

1. $m(1, C)$ 是最优解代价值，相应解计算如下：

If $m(1, C)=m(2, C)$

Then $x_1=0$

Else $x_1=1;$

2. 如果 $x_1=0$, 由 $m(2, C)$ 继续构造最优解；

3. 如果 $x_1=1$, 由 $m(2, C-w_1)$ 继续构造最优解.

例.对于0-1背包问题，给定价值数组
 $v=\{7, 3, 6, 10, 8\}$,重量数组 $w=\{5, 2, 2, 6, 4\}$,
背包容量 $C=10$ 。问：如何选择装入背包的物品，使得装入背包的物品的总价值最大？
并写出递归方程。



$m[i][j]$ 如下表：

	1	2	3	4	5	6	7	8	9	10
1	0	6	6	9	9	14	14	17	17	19
2	0	6	6	9	9	14	14	17	17	19
3	0	6	6	8	8	14	14	16	16	18
4	0	0	0	8	8	10	10	10	10	18
5	0	0	0	8	8	8	8	8	8	8

取第2,3,4个物品时获得最大价值19

$$m(i, j) = m(i+1, j) \quad 0 \leq j < w_i$$

$$m(i, j) = \max\{m(i+1, j), m(i+1, j-w_i)+v_i\} \quad j \geq w_i$$

$$m(n, j) = 0 \quad 0 \leq j < w_n$$

$$m(n, j) = v_n \quad j \geq w_n$$

本讲内容

4. 1 动态规划的原理

4. 2 矩阵乘法问题

4. 3 最长公共子序列问题

4. 4 0-1背包问题

4. 5 最优二分搜索树

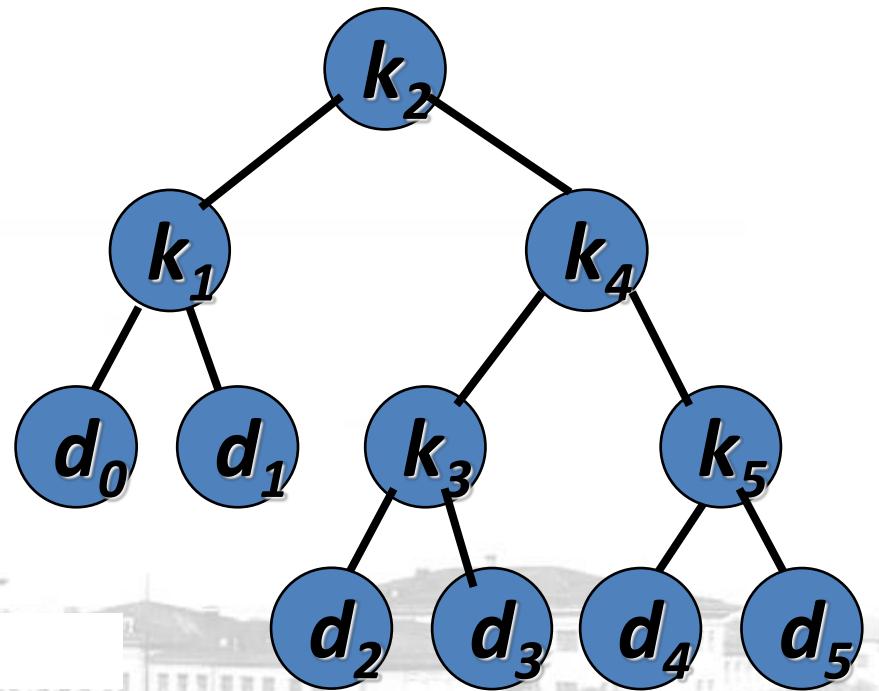
问题的定义

- 二叉搜索树 T



问题的定义

- 二叉搜索树 T

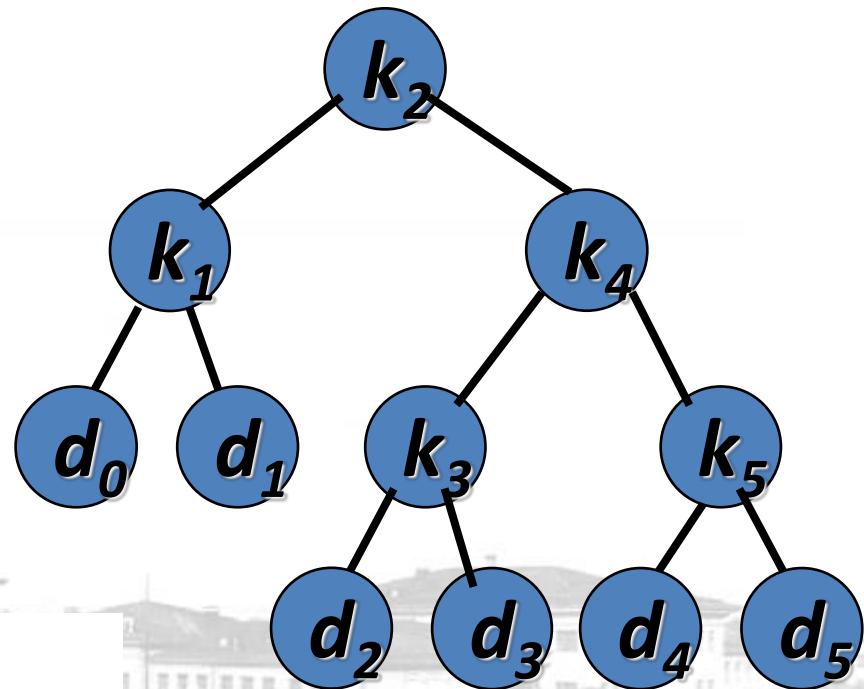


问题的定义

- 二叉搜索树 T

- 结点

- $K = \{k_1, k_2, \dots, k_n\}$
 - $D = \{d_0, d_1, \dots, d_n\}$

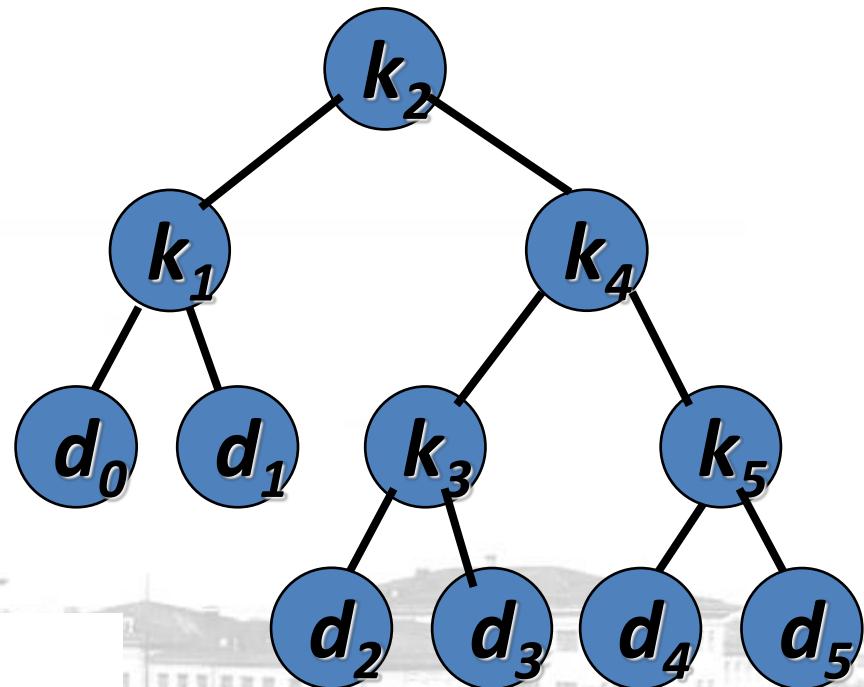


问题的定义

- 二叉搜索树 T

- 结点

- $K = \{k_1, k_2, \dots, k_n\}$
 - $D = \{d_0, d_1, \dots, d_n\}$
 - d_i 对应区间 (k_i, k_{i+1})
 d_0 对应区间 $(-\infty, k_1)$
 d_n 对应区间 $(k_n, +\infty)$



问题的定义

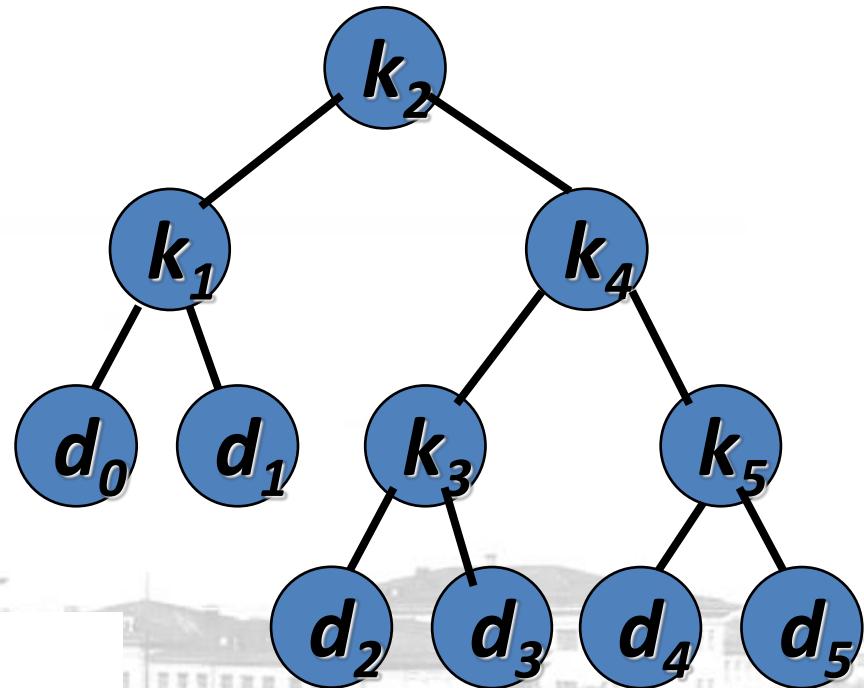
- 二叉搜索树 T

- 结点

- $K = \{k_1, k_2, \dots, k_n\}$
 - $D = \{d_0, d_1, \dots, d_n\}$
 - d_i 对应区间 (k_i, k_{i+1})
 d_0 对应区间 $(-\infty, k_1)$
 d_n 对应区间
 $(k_n, +\infty)$

- 附加信息

- 搜索 k_i 的概率为 p_i
 - 搜索 d_i 的概率为 q_i



问题的定义

- 二叉搜索树 T

- 结点

- $K = \{k_1, k_2, \dots, k_n\}$
 - $D = \{d_0, d_1, \dots, d_n\}$
 - d_i 对应区间 (k_i, k_{i+1})

d_0 对应区间 $(-\infty, k_1)$

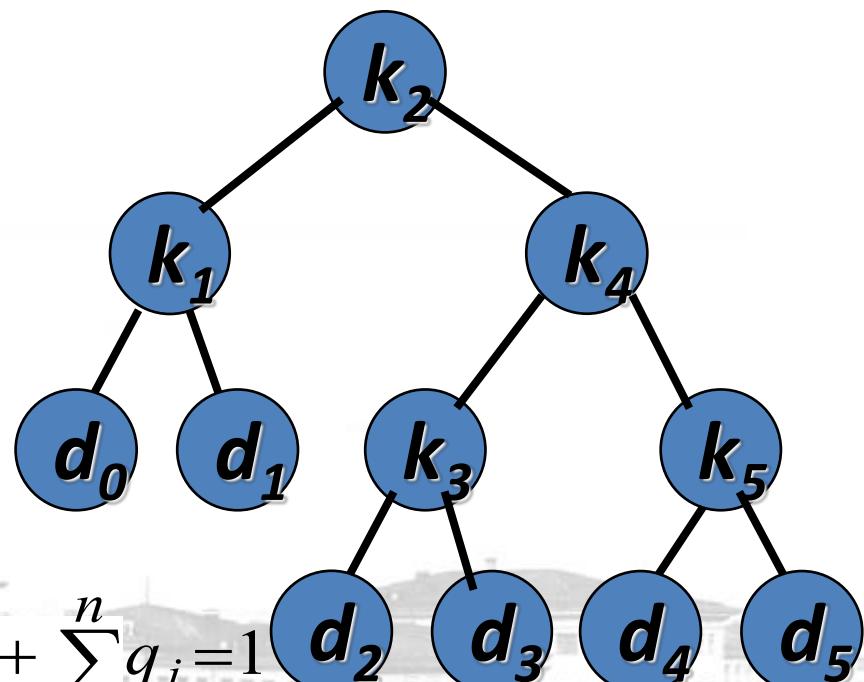
d_n 对应区间
 $(k_n, +\infty)$

- 附加信息

- 搜索 k_i 的概率为 p_i

- 搜索 d_i 的概率为 q_i

$$\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$$



问题的定义

- 二叉搜索树 T

- 结点 搜索树的期望代价

$$E(T) = \sum_{i=1}^n (DEP_T(k_i) + 1)p_i + \sum_{j=0}^n (DEP_T(d_j) + 1)q_j$$

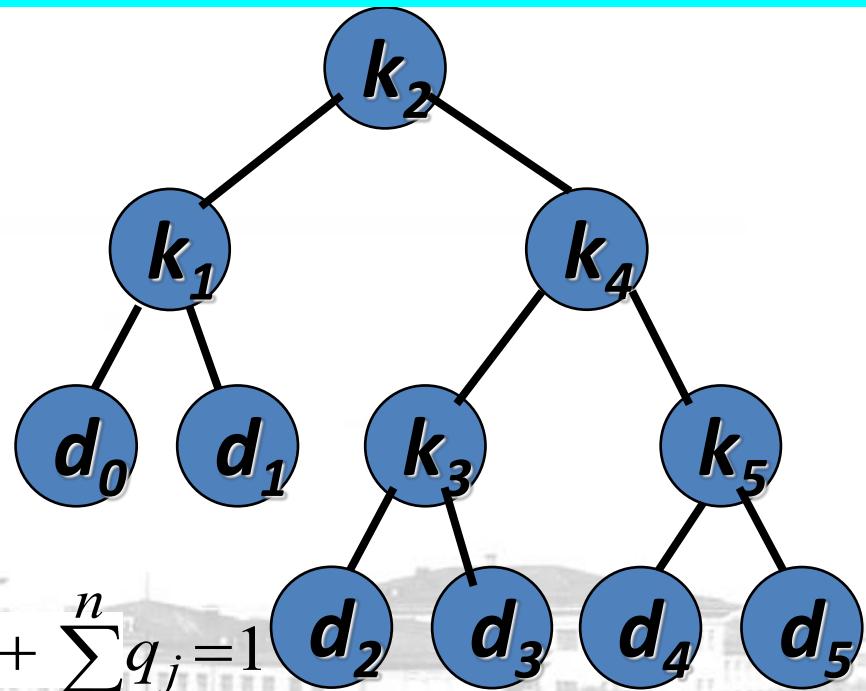
a_n 对应区间
 $(k_n, +\infty)$

- 附加信息

- 搜索 k_i 的概率为 p_i

- 搜索 d_i 的概率为 q_i

$$\sum_{i=1}^n p_i + \sum_{j=0}^n q_j = 1$$



• 问题的定义

输入： $k=\{k_1, k_2, \dots, k_n\}$, $k_1 < k_2 < \dots < k_n$,
 $P=\{p_1, p_2, \dots, p_n\}$, p_i 为搜索 k_i 的概率 $Q=\{q_0, q_1, \dots, q_n\}$, q_i 为搜索值 d_i 的概率

输出： K 的二叉搜索树 T , $E(T)$ 最小

优化二叉搜索树结构的分析

- 优化子结构

定理. 如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T' , 则 T' 必是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子问题的优化解.

优化二叉搜索树结构的分析

- 优化子结构

定理. 如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T' , 则 T' 必是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子问题的优化解.

证明: 若不然, 必有关关键字集 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T'' , T'' 的期望搜索代价低于 T' .

优化二叉搜索树结构的分析

- 优化子结构

定理. 如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T' , 则 T' 必是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子问题的优化解.

证明: 若不然, 必有关关键字集 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T'' , T'' 的期望搜索代价低于 T' .

用 T'' 替换 T 中的 T' , 可以得到一个期望搜索代价比 T 小的原始问题的二叉搜索树,

优化二叉搜索树结构的分析

- 优化子结构

定理. 如果优化二叉搜索树 T 具有包含关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T' , 则 T' 必是关于关键字集合 $\{k_i, k_{i+1}, \dots, k_j\}$ 子问题的优化解.

证明: 若不然, 必有关关键字集 $\{k_i, k_{i+1}, \dots, k_j\}$ 子树 T'' , T'' 的期望搜索代价低于 T' .

用 T'' 替换 T 中的 T' , 可以得到一个期望搜索代价比 T 小的原始问题的二叉搜索树,
与 T 是最优解矛盾.

- 用优化子结构从子问题优化解构造优化解



- 用优化子结构从子问题优化解构造优化解

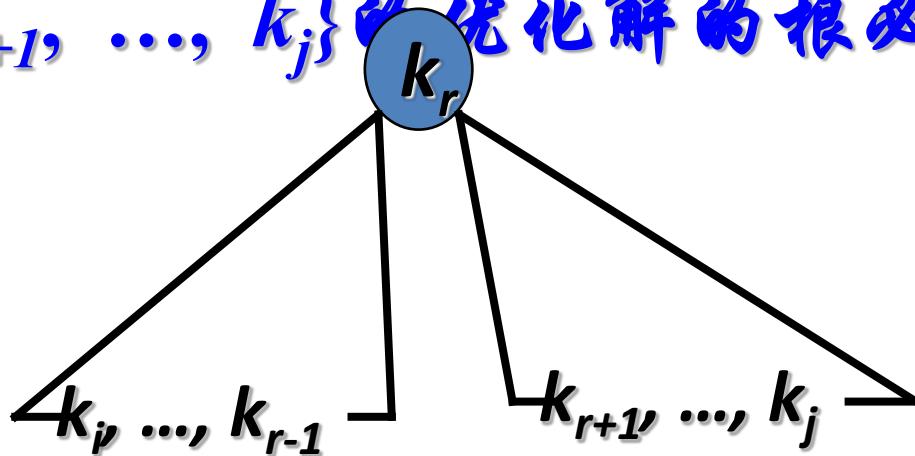
$K=\{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根必为 K 中某个
 k_r



- 用优化子结构从子问题优化解构造优化解

$K = \{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根必为 K 中某个

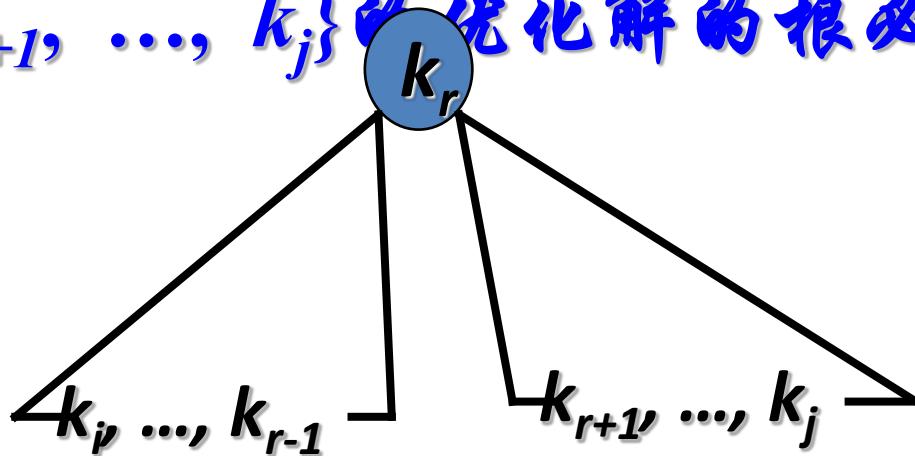
k_r



- 用优化子结构从子问题优化解构造优化解

$K = \{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根必为 K 中某个

k_r

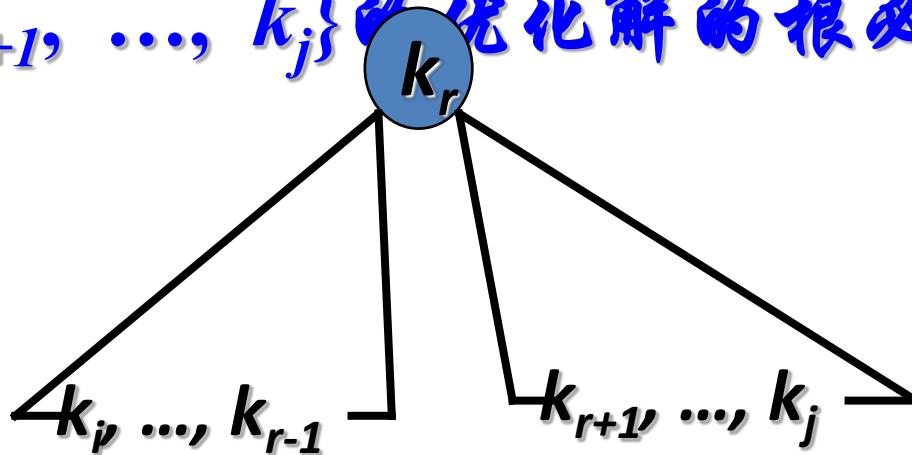


只要对于每个 $k_r \in K$, 确定 $\{k_i, \dots, k_{r-1}\}$ 和 $\{k_{r+1}, \dots, k_j\}$ 的优化解, 我们就可以求出 K 的优化解.

- 用优化子结构从子问题优化解构造优化解

$K = \{k_i, k_{i+1}, \dots, k_j\}$ 的优化解的根必为 K 中某个

k_r



只要对于每个 $k_r \in K$, 确定 $\{k_i, \dots, k_{r-1}\}$ 和 $\{k_{r+1}, \dots, k_j\}$ 的优化解, 我们就可以求出 K 的优化解.

如果 $r=i$, 左子树 $\{k_i, \dots, k_{i-1}\}$ 仅包含 d_{i-1}
 如果 $r=j$, 右子树 $\{k_{r+1}, \dots, k_j\}$ 仅包含 d_j

建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价



建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1)=q_{i-1}$



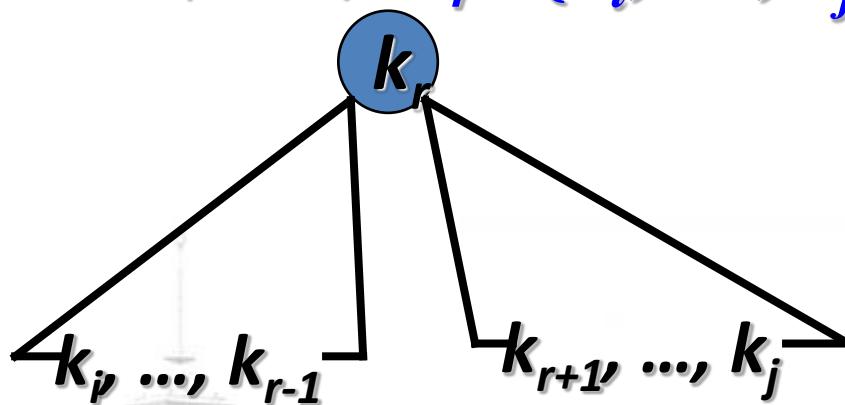
建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1)=q_{i-1}$
 - 当 $j \geq i$ 时, 这样一个 $k_r \in \{k_i, \dots, k_j\}$:



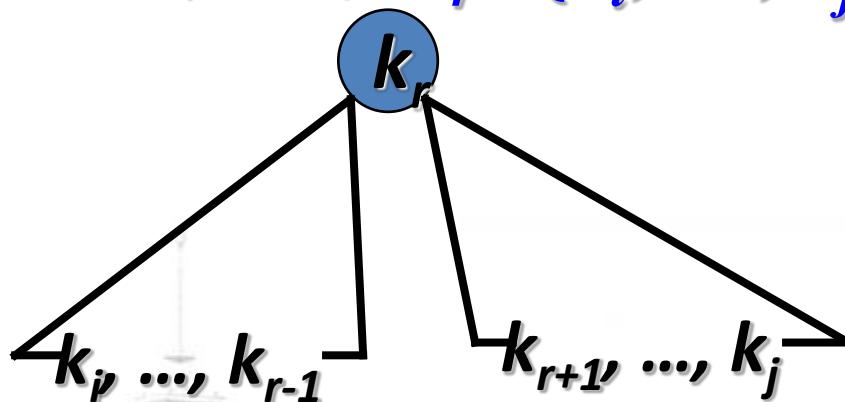
建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1) = q_{i-1}$
 - 当 $j \geq i$ 时, 这样一个 $k_r \in \{k_i, \dots, k_j\}$:



建立优化解的搜索代价递归方程

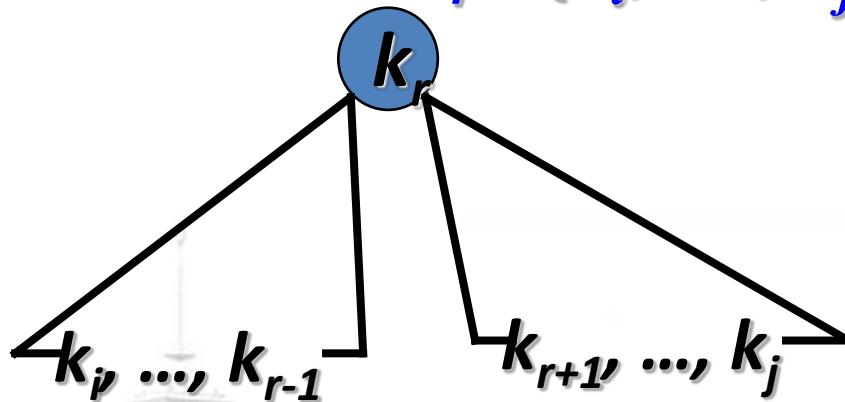
- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1) = q_{i-1}$
 - 当 $j \geq i$ 时, 这样一个 $k_r \in \{k_i, \dots, k_j\}$:



当把左右优化子树放进 T_{ij} 时, 每个结点的深度增加1

建立优化解的搜索代价递归方程

- 令 $E(i, j)$ 为 $\{k_i, \dots, k_j\}$ 的优化解 T_{ij} 的期望搜索代价
 - 当 $j=i-1$ 时, T_{ij} 中只有叶结点 d_{i-1} , $E(i, i-1) = q_{i-1}$
 - 当 $j \geq i$ 时, 这样一个 $k_r \in \{k_i, \dots, k_j\}$:



当把左右优化子树放进 T_{ij} 时, 每个结点的深度增加1

$$E(i, j) = P_r + E(\text{左子树}) + W(i, r-1) + E(\text{右子树}) + W(r+1, j)$$

- 计算 $W(i, r-1)$ 和 $W(r+1, j)$



- 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$



- 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$



• 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$

知 $W(i, r-1) = E(LT+1) - E(LT) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$

• 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$

知 $W(i, r-1) = E(LT+1) - E(LT) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$

同理, $W(r+1, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$

• 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$

知 $W(i, r-1) = E(LT+1) - E(LT) = \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$

同理, $W(r+1, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$

$$W(i, j) = W(i, r-1) + W(r+1, j) + P_r$$

• 计算 $W(i, r-1)$ 和 $W(r+1, j)$

由 $E(LT+1) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 2)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 2)q_l$

$$E(LT) = \sum_{l=i}^{r-1} (DEP_{左}(k_l) + 1)p_l + \sum_{l=i-1}^{r-1} (DEP_{左}(d_l) + 1)q_l$$

$E(i, j) = E(i, r-1) + E(r+1, j) + W(i, j)$

即 $E(i, r-1) = E(LT+1) - E(LT) - \sum_{l=i}^{r-1} p_l + \sum_{l=i-1}^{r-1} q_l$

同理, $W(r+1, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$

$W(i, j) = W(i, r-1) + W(r+1, j) + P_r$

总之

$$E(i, j) = q_{i-1} \quad \text{If } j=i-1$$

$$E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\} \quad \text{if } j \geq i$$

自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$E(1,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$E(1,0)$

$E(1,1)$

$E(1,2)$

$E(1,3)$

$E(1,4)$

$E(2,4)$

$E(3,4)$

$E(4,4)$

$E(5,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$E(1,0)$

$E(1,1)$

$E(1,2)$

$E(1,3)$

$E(1,4)$

$E(2,1)$

$E(2,2)$

$E(2,3)$

$E(2,4)$

$E(3,4)$

$E(4,4)$

$E(5,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$E(1,0)$ $E(1,1)$ $E(1,2)$ $E(1,3)$ $E(1,4)$

$E(2,1)$ $E(2,2)$ $E(2,3)$ $E(2,4)$

$E(3,2)$ $E(3,3)$ $E(3,4)$

$E(4,4)$

$E(5,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$E(1,0)$

$E(1,1)$

$E(1,2)$

$E(1,3)$

$E(1,4)$

$E(2,1)$ $E(2,2)$ $E(2,3)$ $E(2,4)$

$E(3,2)$ $E(3,3)$ $E(3,4)$

$E(4,3)$ $E(4,4)$

$E(5,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$q_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$

$E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$

$E(3,2) \quad E(3,3) \quad E(3,4)$

$E(4,3) \quad E(4,4)$

$E(5,4)$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$\mathbf{q_0} = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$\mathbf{q_1} = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$E(3,2) \quad E(3,3) \quad E(3,4)$$

$$E(4,3) \quad E(4,4)$$

$$E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$\mathbf{q_0} = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$\mathbf{q_1} = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$\mathbf{q_2} = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$E(4,3) \quad E(4,4)$$

$$E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$\textcolor{red}{q_0} = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad \boxed{E(1,4)}$$

$$\textcolor{red}{q_1} = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$\textcolor{red}{q_2} = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$\textcolor{red}{q_3} = E(4,3) \quad E(4,4)$$

$$E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$\mathbf{q}_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$\mathbf{q}_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$\mathbf{q}_2 = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$\mathbf{q}_3 = E(4,3) \quad E(4,4)$$

$$\mathbf{q}_4 = E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$\cancel{\mathbf{q}_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)}$$

$$\cancel{\mathbf{q}_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)}$$

$$\cancel{\mathbf{q}_2 = E(3,2) \quad E(3,3) \quad E(3,4)}$$

$$\cancel{\mathbf{q}_3 = E(4,3) \quad E(4,4)}$$

$$\cancel{\mathbf{q}_4 = E(5,4)}$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$q_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$q_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$q_2 = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$q_3 = E(4,3) \quad E(4,4)$$

$$q_4 = E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$q_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$q_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$q_2 = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$q_3 = E(4,3) \quad E(4,4)$$

$$q_4 = E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$q_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad E(1,4)$$

$$q_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$q_2 = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$q_3 = E(4,3) \quad E(4,4)$$

$$q_4 = E(5,4)$$



自下而上计算优化解的搜索代价

- $E(i, j) = \min_{i \leq r \leq j} \{E(i, r-1) + E(r+1, j) + W(i, j)\}$

$$q_0 = E(1,0) \quad E(1,1) \quad E(1,2) \quad E(1,3) \quad \boxed{E(1,4)}$$

$$q_1 = E(2,1) \quad E(2,2) \quad E(2,3) \quad E(2,4)$$

$$q_2 = E(3,2) \quad E(3,3) \quad E(3,4)$$

$$q_3 = E(4,3) \quad E(4,4)$$

$$q_4 = E(5,4)$$



- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
+ q_j

$$\textcolor{red}{q_0} = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)$$

$$\textcolor{red}{q_1} = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)$$

$$\textcolor{red}{q_2} = W(3,2) \ W(3,3) \ W(3,4)$$

$$\textcolor{red}{q_3} = W(4,3) \ W(4,4)$$

$$\textcolor{red}{q_4} = W(5,4)$$



- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
+ q_j

$$\cancel{q_0 = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)}$$

$$\cancel{q_1 = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)}$$

$$\cancel{q_2 = W(3,2) \ W(3,3) \ W(3,4)}$$

$$\cancel{q_3 = W(4,3) \ W(4,4)}$$

$$\cancel{q_4 = W(5,4)}$$



- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
+ q_j

$$\cancel{q_0 = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)}$$

$$\cancel{q_1 = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)}$$

$$\cancel{q_2 = W(3,2) \ W(3,3) \ W(3,4)}$$

$$\cancel{q_3 = W(4,3) \ W(4,4)}$$

$$\cancel{q_4 = W(5,4)}$$



- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
 $+ q_j$

$$\cancel{q_0 = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)}$$

$$\cancel{q_1 = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)}$$

$$\cancel{q_2 = W(3,2) \ W(3,3) \ W(3,4)}$$

$$\cancel{q_3 = W(4,3) \ W(4,4)}$$

$$\cancel{q_4 = W(5,4)}$$

- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
 $+ q_j$

$$\cancel{q_0 = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)}$$

$$\cancel{q_1 = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)}$$

$$\cancel{q_2 = W(3,2) \ W(3,3) \ W(3,4)}$$

$$\cancel{q_3 = W(4,3) \ W(4,4)}$$

$$\cancel{q_4 = W(5,4)}$$



- $W(i, i-1) = q_{i-1}, \quad W(i, j) = W(i, j-1) + p_j$
 $+ q_j$

$$\cancel{q_0 = W(1,0) \ W(1,1) \ W(1,2) \ W(1,3) \ W(1,4)}$$

$$\cancel{q_1 = W(2,1) \ W(2,2) \ W(2,3) \ W(2,4)}$$

$$\cancel{q_2 = W(3,2) \ W(3,3) \ W(3,4)}$$

$$\cancel{q_3 = W(4,3) \ W(4,4)}$$

$$\cancel{q_4 = W(5,4)}$$

Optimal-BST(p, q, n)

For $i=1$ To $n+1$ Do

$E(i, i-1) = q_{i-1};$

$W(i, i-1) = q_{i-1};$

For $l=1$ To n Do

For $i=1$ To $n-l+1$ Do

$j=i+l-1;$

$E(i, j)=\infty;$

$W(i, j)=W(i, j-1)+p_j+q_j;$

For $r=i$ To j Do

$t=E(i, r-1)+E(r+1, j)+W(i, j);$

If $t < E(i, j)$

Then $E(i, j)=t;$ $Root(i, j)=r;$

Return E and $Root$