



算法分析与设计

Analysis and Design of Algorithm

第6次课



要点回顾

- 改进分治算法的途径
 - 随机子问题划分（快速排序）
 - 减少子问题数目（大整数乘法、矩阵相乘）



要点回顾

- 改进分治算法的途径
 - 随机子问题划分（快速排序）

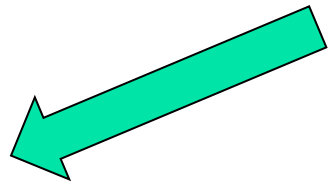
最坏情况下，时间复杂度为 $O(n^2)$ 。如何解决？

```
int RandomizedPartition (int r[ ], int first, int end) {  
    int i = Random(first, end);  
    Swap(r[i], r[first]);  
    return Partition (a, first, end);  
}
```

要点回顾

- 改进分治算法的途径
 - 随机子问题划分（快速排序）
 - 减少子问题数目（大整数乘法）

$$XY = ac 2^n + (ad+bc) 2^{n/2} + bd$$



$$\begin{aligned} ad + bc &= (a-b)(d-c) + ac + bd \\ &= (a+b)(d+c) - ac - bd \end{aligned}$$



要点回顾

- 改进分治算法的途径
 - 随机子问题划分（快速排序）
 - 减少子问题数目（大整数乘法、**矩阵相乘**）

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

由此可得：

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

要点回顾

■ 改进分治算法的途径

- 随机子问题划分（快速排序）
- 减少子问题数目（大整数乘法、**矩阵相乘**）

$$M_1 = A_{11}(B_{12} - B_{22})$$

$$M_2 = (A_{11} + A_{12})B_{22}$$

$$M_3 = (A_{21} + A_{22})B_{11}$$

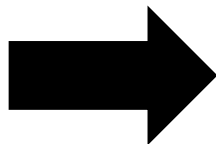
$$M_4 = A_{22}(B_{21} - B_{11})$$

$$M_5 = (A_{11} + A_{22})(B_{11} + B_{22})$$

$$M_6 = (A_{12} - A_{22})(B_{21} + B_{22})$$

$$M_7 = (A_{11} - A_{21})(B_{11} + B_{12})$$

7次乘



$$C_{11} = M_5 + M_4 - M_2 + M_6$$

$$C_{12} = M_1 + M_2$$

$$C_{21} = M_3 + M_4$$

$$C_{22} = M_5 + M_1 - M_3 - M_7$$

改进分治算法的途径3： 增加预处理



最接近点对问题

问题： 给定平面上 n 个点的集合 S ，找其中的一对点，使得在 n 个点组成的所有点对中，该点对间的距离最小。

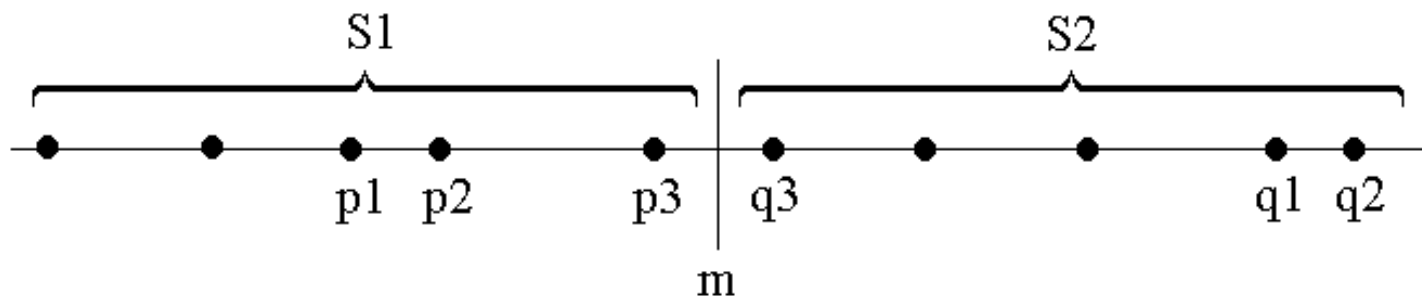
最接近点对问题

问题： 给定平面上 n 个点的集合 S ，找其中的一对点，使得在 n 个点组成的所有点对中，该点对间的距离最小。

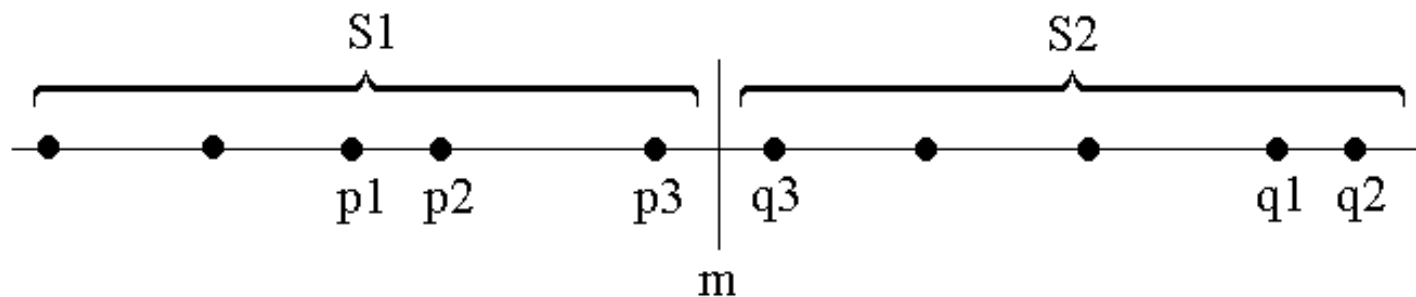


最接近点对问题（一维情况）

为了使问题易于理解和分析，先来考虑**一维**的情形。此时， S 中的 n 个点退化为 x 轴上的 n 个实数 x_1, x_2, \dots, x_n 。最接近点对即为这 n 个实数中相差最小的2个实数。



最接近点对问题（一维情况）



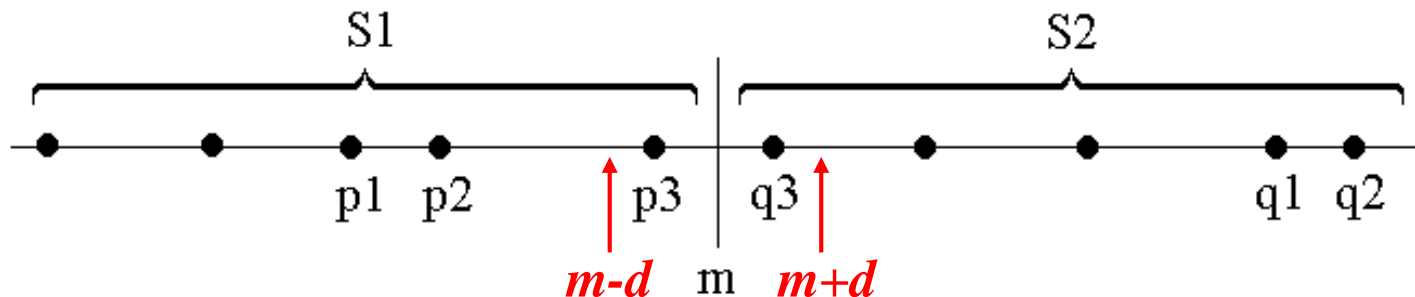
假设： 用 x 轴上某个点 m 将 S 划分为2个子集 $S1$ 和 $S2$ ，基于**平衡子问题**的思想，用 S 中各点坐标的中位数来作分割点。

递归地，在 $S1$ 和 $S2$ 上找出其最接近点对 $\{p1, p2\}$ 和 $\{q1, q2\}$ ，并设 $d = \min\{|p1 - p2|, |q1 - q2|\}$ ， S 中的最接近点对或者是 $\{p1, p2\}$ ，或者是 $\{q1, q2\}$ ，或者是某个 $\{p3, q3\}$ ，其中 $p3 \in S1$ 且 $q3 \in S2$ 。

最接近点对问题（一维情况）

能否在线性时间内找到 p_3, q_3 ?

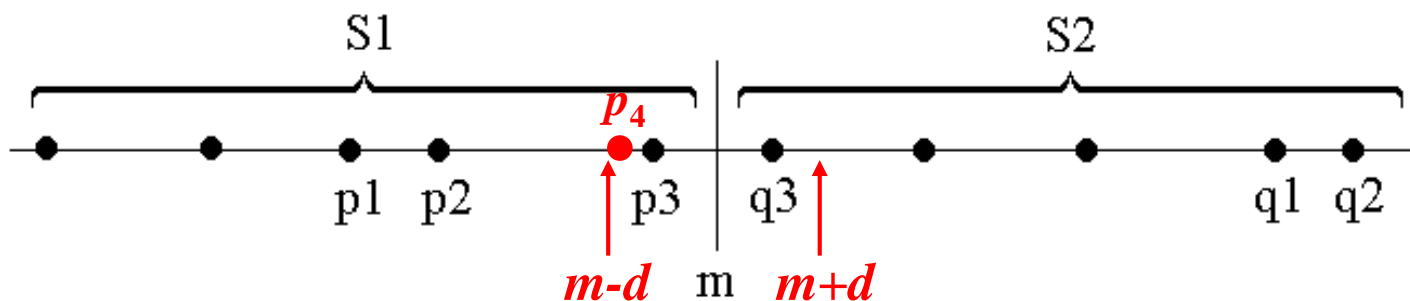
- 如果 S 的最接近点对是 $\{p_3, q_3\}$ ，即 $|p_3 - q_3| < d$ ，则 p_3 和 q_3 两者与 m 的距离不超过 d ，即 $p_3 \in (m-d, m]$ ， $q_3 \in (m, m+d]$ 。



最接近点对问题（一维情况）

能否在线性时间内找到 p_3, q_3 ?

- 由于在 S_1 中，每个长度为 d 的半闭区间至多包含一个点（否则必有两点距离小于 d ），并且 m 是 S_1 和 S_2 的分割点，因此 $(m-d, m]$ 中至多包含 S 中的一个点。由图可以看出，如果 $(m-d, m]$ 中有 S 中的点，则此点就是 S_1 中最大点。





最接近点对问题（一维情况）

能否在线性时间内找到 p_3, q_3 ?

- 因此，用线性时间就能找到区间 $(m-d, m]$ 和 $(m, m+d]$ 中所有点，即 p_3 和 q_3 。从而用线性时间就可以将 S_1 的解和 S_2 的解合并成为 S 的解。


$$d = \min(d_{S_1}, d_{S_2}, q_3 - p_3)$$

最接近点对问题（一维情况）

```
bool Cpair1(S,d)
{
    n = | S |;
    if (n < 2) { d =  $\infty$ ; return false; }
    m = S 中各点坐标的中位数;
    构造 S1 和 S2;
    //S1 = {x  $\in$  S | x  $\leq$  m}, S2 = {x  $\in$  S | x > m}
    Cpair1(S1,d1);
    Cpair1(S2,d2);
    p = max(S1);
    q = min(S2);
    d = min(d1,d2,q - p);
    return true;
}
```



最接近点对问题（一维情况）

复杂度分析：

$$T(n) = \begin{cases} O(1) & n < 4 \\ 2T(n/2) + O(n) & n \geq 4 \end{cases}$$

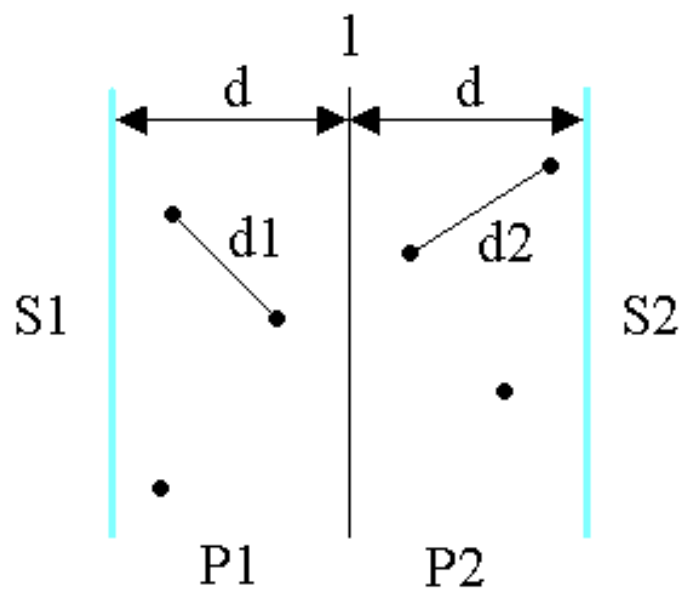
→ $T(n) = O(n \log n)$

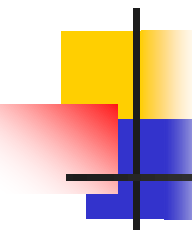
看上去比用排序加扫描的算法复杂，然而它可以推广到二维的情形。

最接近点对问题

下面来考虑二维的情形

- 选取一垂直线 $l: x=m$ 来作为分割直线。其中 m 为 S 中各点 x 坐标的中位数。由此将 S 分割为 S_1 和 S_2 。
- 递归地在 S_1 和 S_2 上找出其最小距离 d_1 和 d_2 ，并设 $d = \min\{d_1, d_2\}$ ， S 中的最接近点对或者是 d ，或者是某个 $\{p, q\}$ ，其中 $p \in P_1$ 且 $q \in P_2$ 。
- 能否在线性时间内找到 p, q ?





```
bool Cpair2(S,d)
```

```
{
```

```
    n = | S |;
```

```
    if (n < 2) { d =  $\infty$ ; return false; }
```

```
1.    m = S 中各点 x 间坐标的中位数;
```

```
    构造 S1 和 S2;
```

```
    //S1 = {p  $\in$  S | x(p)  $\leq$  m}, S2 = {p  $\in$  S | x(p) > m}
```

```
2.    Cpair2(S1,d1);
```

```
    Cpair2(S2,d2);
```

```
}
```

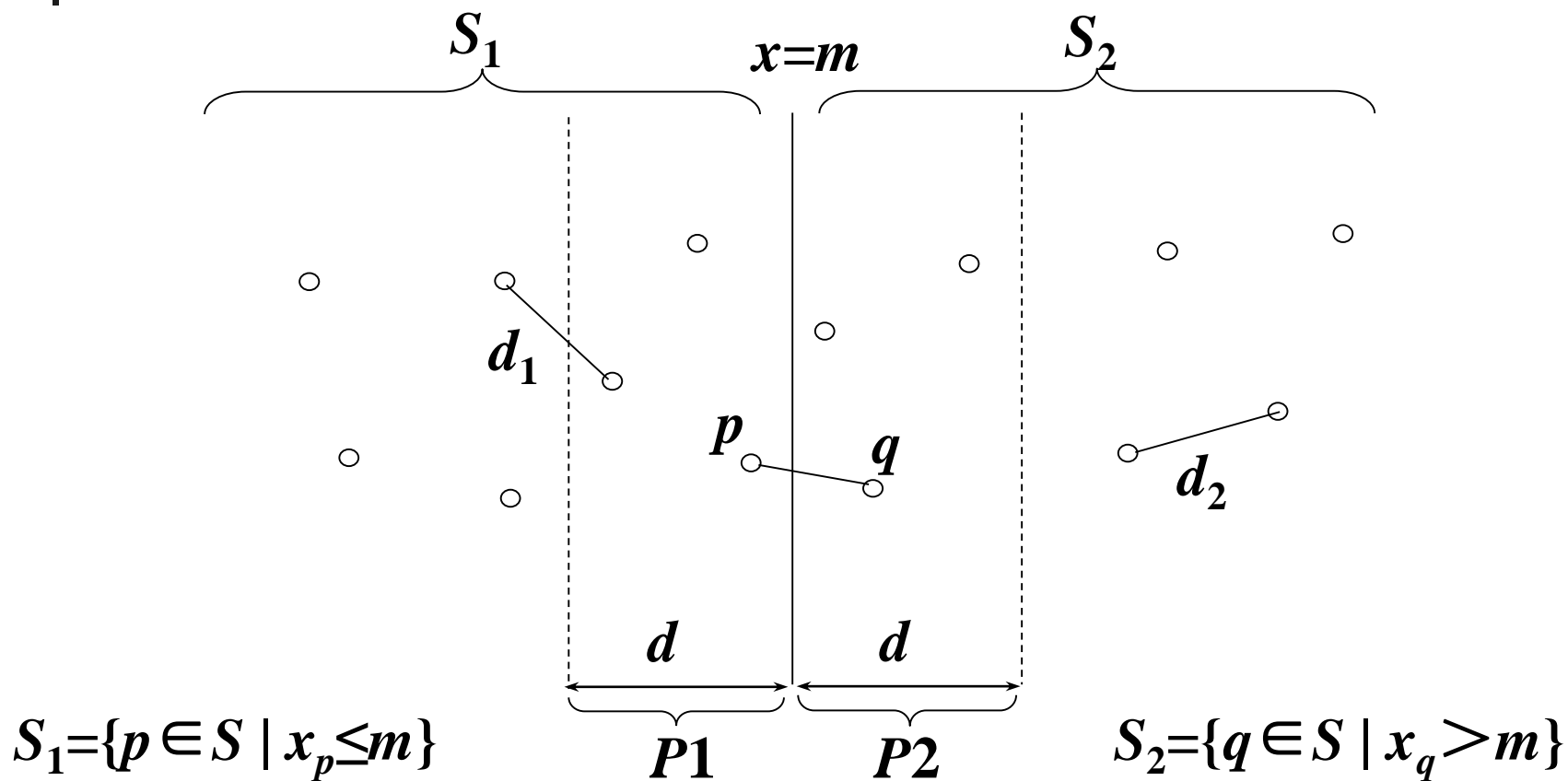


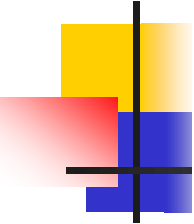
最接近点对问题

能否在线性时间内找到 p_3, q_3 ?

- 考虑 P_1 中任意一点 p ，它若与 P_2 中的点 q 构成最接近点对的候选者，则必有 $\text{distance}(p, q) < d$ 。满足这个条件的 P_2 中的点一定落在一个 $d \times 2d$ 的矩形 R 中
- 由 d 的意义可知， P_2 中任何2个 S 中的点的距离都不小于 d 。由此可以推出**矩形 R 中最多只有6个 S 中的点**。
- 因此，在分治法的合并步骤中**最多只需要检查 $6 \times n/2 = 3n$ 个**候选者

最接近点对问题





```
bool Cpair2(S,d)
```

```
{
```

```
    n = | S |;
```

```
    if (n < 2) { d =  $\infty$ ; return false; }
```

```
1.    m = S 中各点 x 间坐标的中位数;
```

```
    构造 S1 和 S2;
```

```
    //S1 = {p  $\in$  S | x(p)  $\leq$  m}, S2 = {p  $\in$  S | x(p) > m}
```

```
2.    Cpair2(S1,d1);
```

```
    Cpair2(S2,d2);
```

```
3.    dm = min(d1,d2);
```

```
4.    设 P1 是 S1 中距垂直分割线 l 的距离在 dm 之内的所有点组成的集合;
```

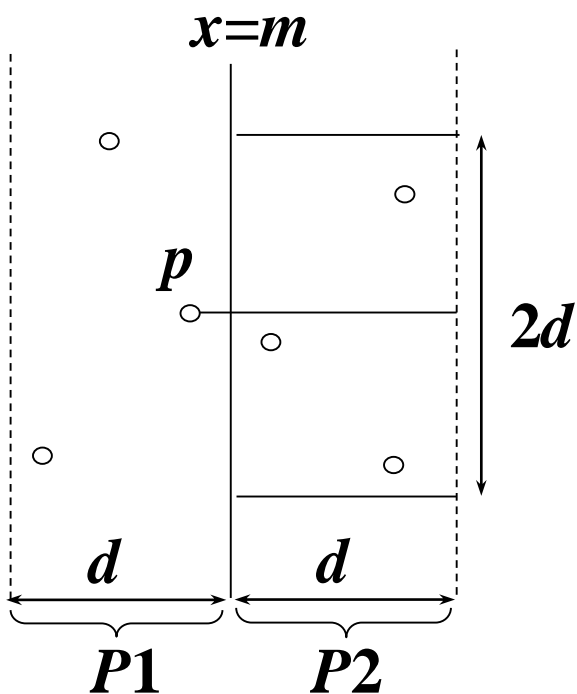
```
    P2 是 S2 中距分割线 l 的距离在 dm 之内所有点组成的集合;
```

```
    将 P1 和 P2 中点依其 y 坐标值排序;
```

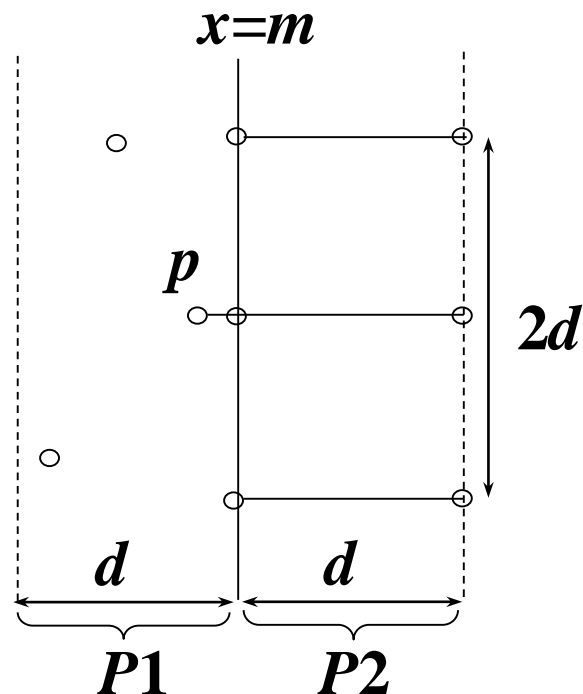
```
    并设 X 和 Y 是相应的已排好序的点列;
```

```
}
```

对于点 $p \in P1$ ，需要考察 $P2$ 中的各个点和点 p 之间的距离是否小于 d ，显然， $P2$ 中这样点的 y 轴坐标一定位于区间 $[y-d, y+d]$ 之间，而且，这样的点不会超过6个。



(a) 包含点 q 的 $d \times 2d$ 的矩形区域



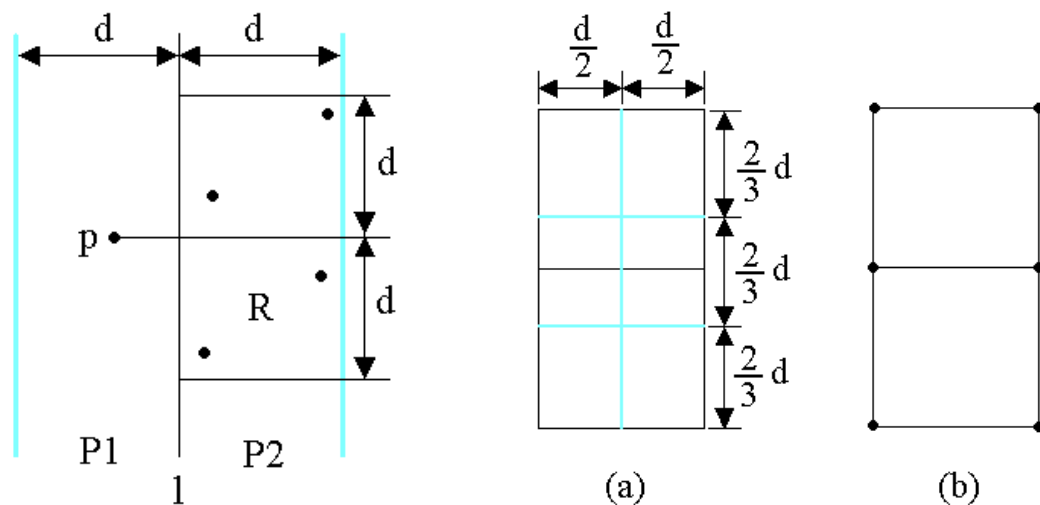
(b) 最坏情况下需要检查的6个点

最接近点对问题

证明： 将矩形R的长为 $2d$ 的边3等分，将它的长为 d 的边2等分，由此导出6个 $(d/2) \times (2d/3)$ 的矩形。若矩形R中有多于6个S中的点，则由鸽舍原理易知至少有一个 $(d/2) \times (2d/3)$ 的小矩形中有2个以上S中的点。设 u, v 是位于同一小矩形中的2个点，则

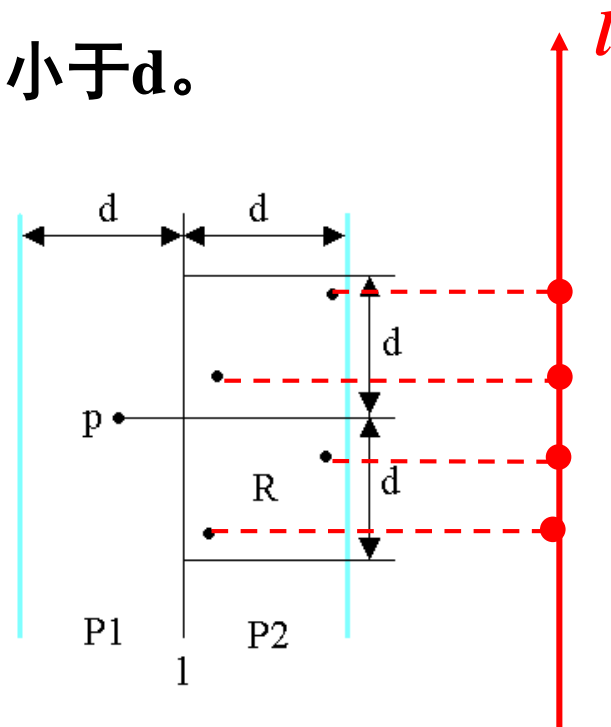
$$(x(u) - x(v))^2 + (y(u) - y(v))^2 \leq (d/2)^2 + (2d/3)^2 = \frac{25}{36}d^2$$

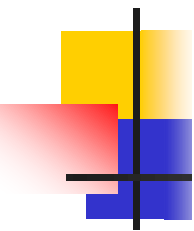
$\text{distance}(u, v) < d$ 。这与 d 的意义相矛盾。



最接近点对问题

- 要检查哪6个点？
 - 将 p 和 $P2$ 中所有 $S2$ 的点投影到垂直线 l 上。
 - 投影点距 p 在 l 上投影点的距离小于 d 。
 - 这种投影点最多只有6个。





```
bool Cpair2(S,d)
```

```
{
```

```
    n = | S |;
```

```
    if (n < 2) { d =  $\infty$ ; return false; }
```

```
1.    m = S 中各点 x 间坐标的中位数;
```

```
    构造 S1 和 S2;
```

```
    //S1 = {p  $\in$  S | x(p)  $\leq$  m}, S2 = {p  $\in$  S | x(p) > m}
```

```
2.    Cpair2(S1,d1);
```

```
    Cpair2(S2,d2);
```

```
3.    dm = min(d1,d2);
```

```
4.    设 P1 是 S1 中距垂直分割线 l 的距离在 dm 之内的所有点组成的集合;
```

```
    P2 是 S2 中距分割线 l 的距离在 dm 之内所有点组成的集合;
```

```
    将 P1 和 P2 中点依其 y 坐标值排序;
```

```
    并设 X 和 Y 是相应的已排好序的点列;
```

```
5.    通过扫描 X 以及对于 X 中每个点检查 Y 中与其距离在 dm 之内的所有点(最多 6  
        个) 可以完成合并;
```

```
    当 X 中的扫描指针逐次向上移动时, Y 中的扫描指针可在宽为 2dm 的一个区间内  
    移动;
```

```
    设 dl 是按这种扫描方式找到的点对间的最小距离;
```

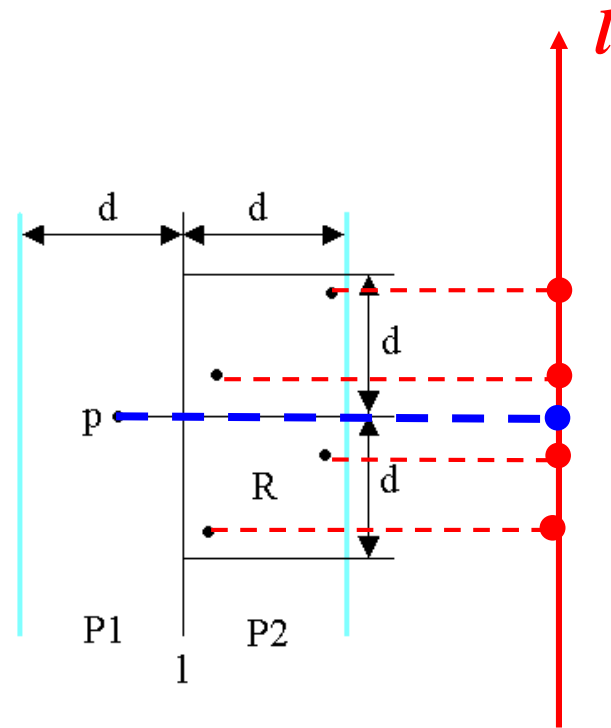
```
6.    d = min(dm,dl);
```

```
    return true;
```

```
}
```

最接近点对问题

- 快速检查6个点的方法
 - 将P1和P2中所有S中点按其y坐标排好序（预处理）
 - 对P1中所有点，对排好序的点列作一次扫描，
 - 对P1中每一点最多只要检查P2中排好序的相继6个点。



最接近点对问题

算法分析

中国大学MOOC

步1 递归边界处理: $O(1)$

步2 排序: $O(n\log n)$

步3 划分: $O(1)$

步4-5子问题: $2T(n/2)$

步6确定 δ : $O(1)$

步7检查跨边界点对: $O(n)$

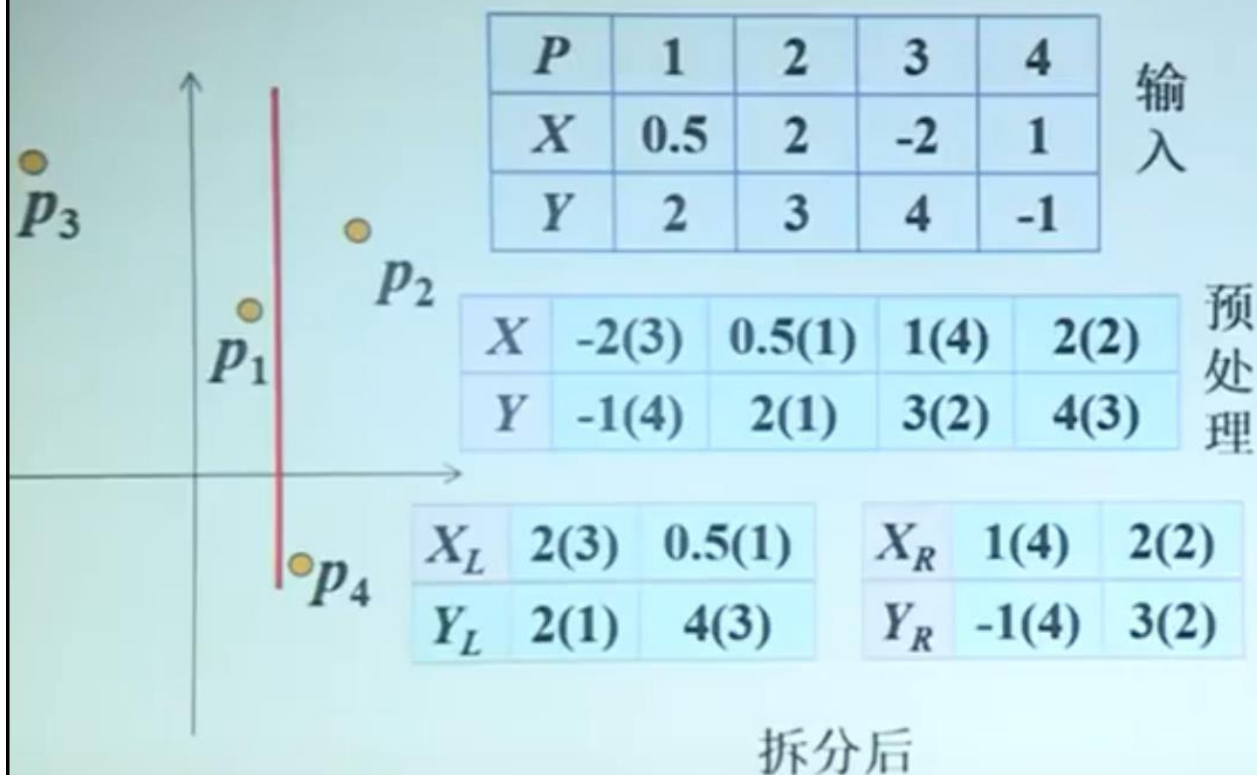
$$T(n) = 2T(n/2) + O(n\log n)$$

$$T(n) = O(1), n \leq 3$$

递归树求解 $T(n) = O(n\log^2 n)$

最接近点对问题

实例：递归中的拆分





最接近点对问题

复杂度分析:

$$T(n) = \begin{cases} O(1) & n < 4 \\ 2T(n/2) + O(n) & n \geq 4 \end{cases}$$

→ $T(n) = O(n \log n)$



第二章要点回顾

■ 递归算法

- 概念（阶乘、Fibonacci数列、双递归）
- 例子（整数划分问题、Hanoi塔问题）
- Hanoi塔算法、运行轨迹、分析时间复杂度
- 递推方程（迭代法求解）
- 递归的优缺点

■ 分治策略

- 基本思想、适用条件、基本步骤
- 分治效率分析
- 范例学习：二分搜索、大整数乘法、Strassen矩阵乘法、二分归并排序、快速排序、最近点对问题

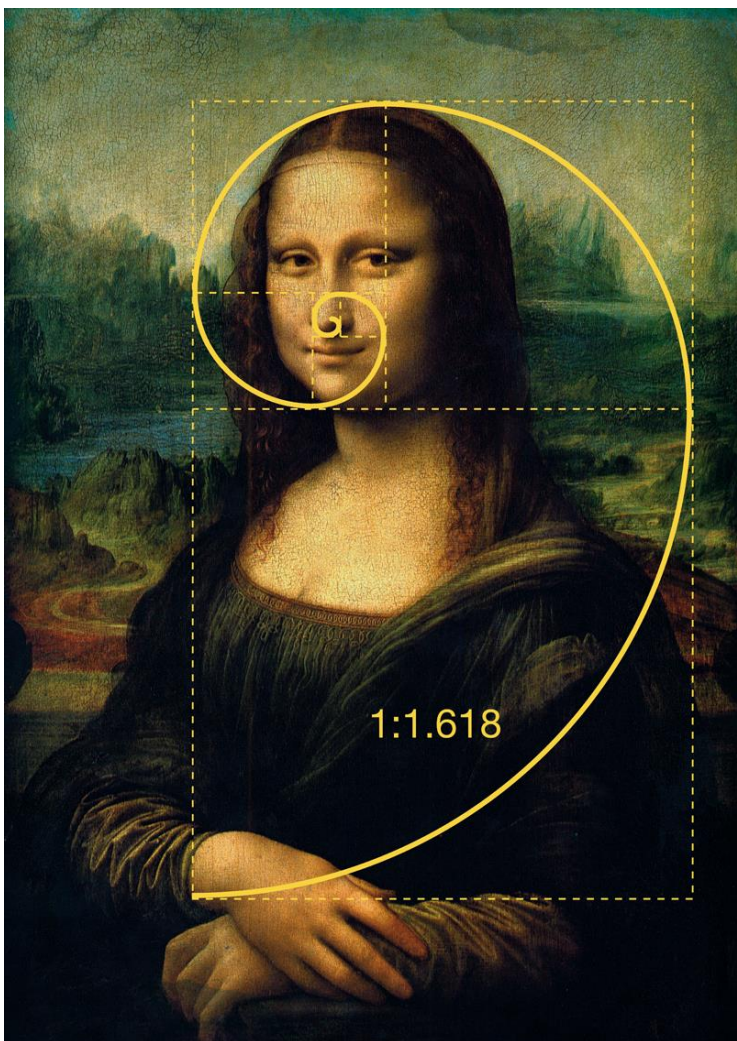
第三章 动态规划



学习要点

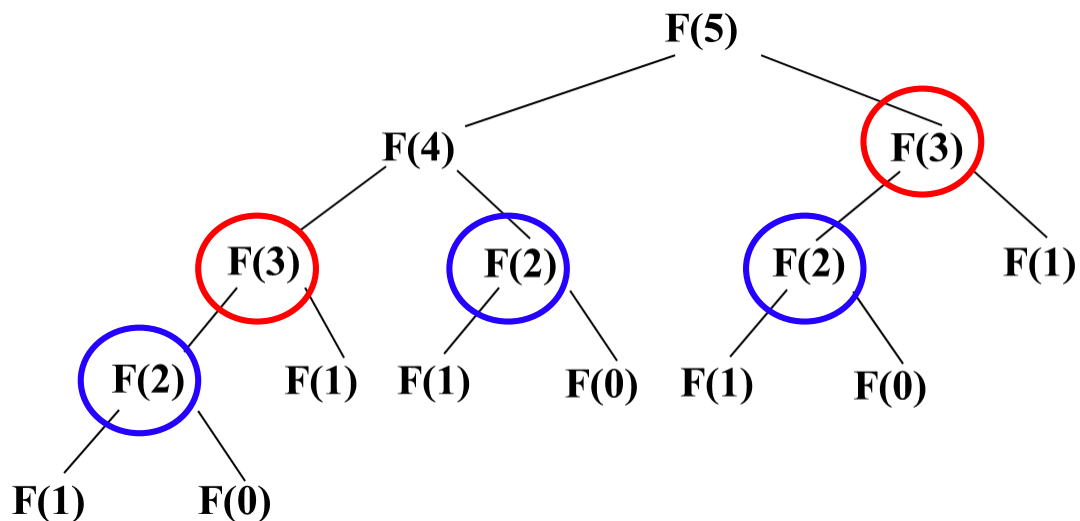
- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
 - 最优子结构性质
 - 重叠子问题性质
- 掌握设计动态规划算法的步骤。
 - 找出最优解的性质，并刻画其结构特征
 - 递归地定义最优值
 - 以自底向上的方式计算出最优值
 - 根据计算最优值时得到的信息，构造最优解

从Fibonacci数列开始



$$F(n) = \begin{cases} 1 & , n = 0 \\ 1 & , n = 1 \\ F(n-1) + F(n-2) & , n > 1 \end{cases}$$

$n=5$ 时分治法计算斐波那契数的过程:





例：计算Fibonacci数列

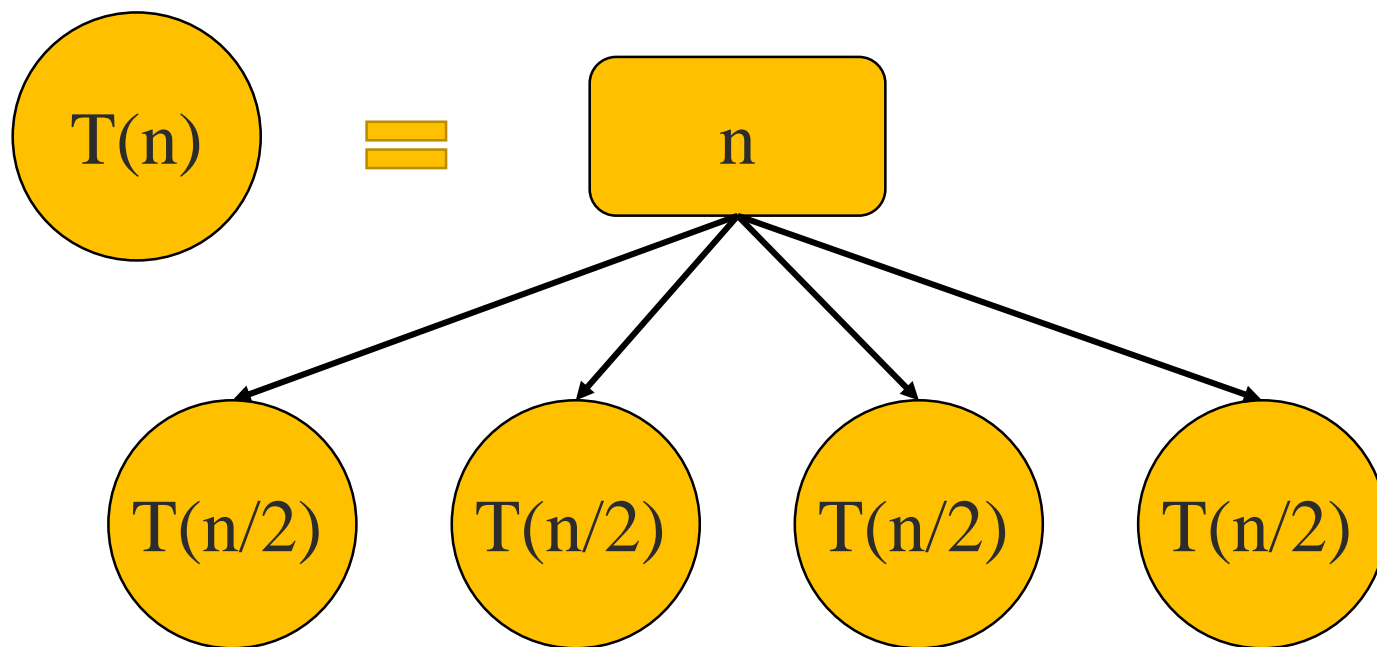
分析：注意到，计算 $F(n)$ 是以计算它的两个重叠子问题 $F(n-1)$ 和 $F(n-2)$ 的形式来表达的，所以，可以设计一张表填入 $n+1$ 个 $F(n)$ 的值。

方案：可以将中间结果缓存到表格中，则斐波那契数 $F(9)$ 的填表过程：

n	0	1	2	3	4	5	6	7	8	9
$F(n)$	0	1	1	2	3	5	8	13	21	34

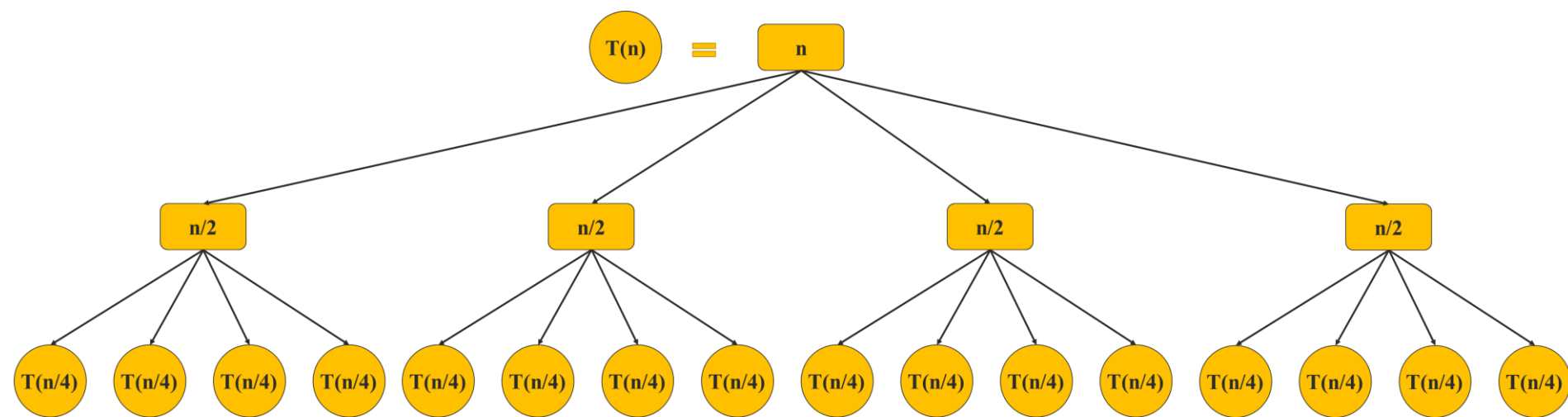
算法总体思想

- 动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题



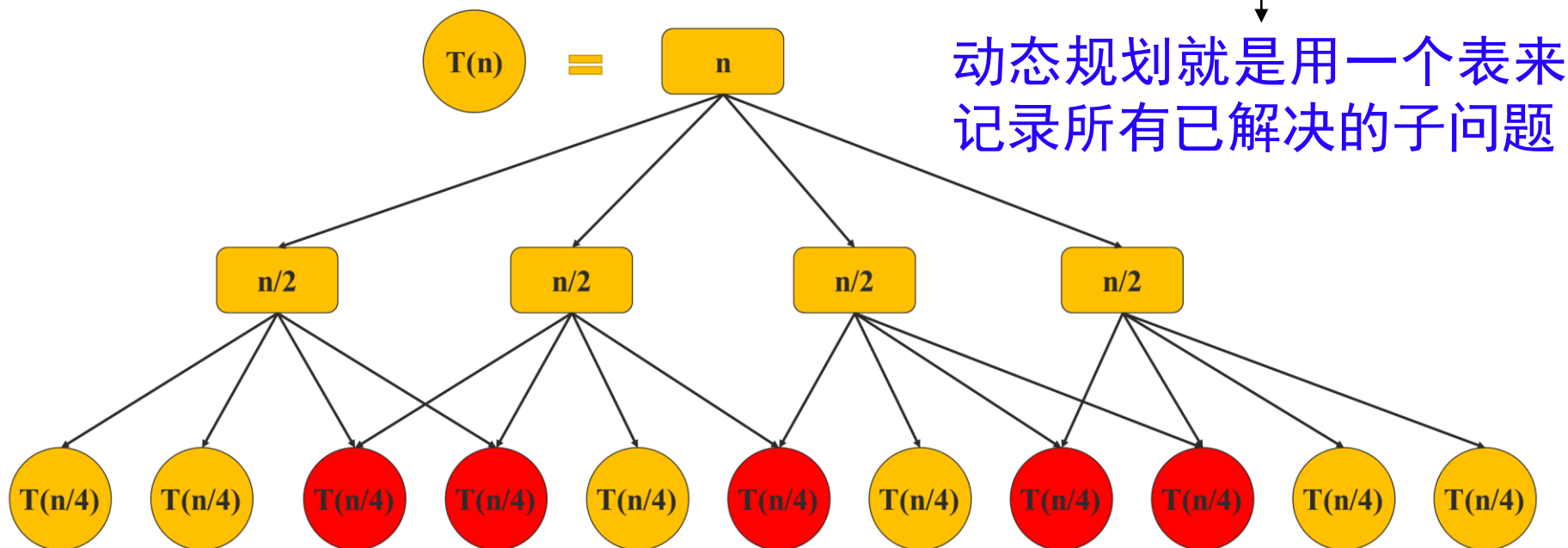
算法总体思想

- **与分治区别：**经分解得到的子问题往往不是互相独立的。不同子问题的数目常常只有多项式量级。在用分治法求解时，有些子问题被重复计算了许多次。



算法总体思想

- 用分治法求解，子问题的数目常常过多，有些子问题被重复计算了多次，如果能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，就可以避免大量重复计算，从而得到多项式时间算法。





动态规划法注意事项

- 用动态规划法求解的问题具有特征：
 - 能够分解为相互重叠的若干子问题；
 - 满足最优性原理（也称**最优子结构性**）：该问题的最优解中也包含着其子问题的最优解。
- 用反证法分析问题是否满足最优性原理：
 - 先假设由问题的最优解导出的子问题的解不是最优的；
 - 然后再证明在这个假设下可构造出比原问题最优解更好的解，从而导致矛盾。



动态规划基本步骤

动态规划法设计算法一般分成三个阶段：

1. **分段**：将原问题分解为若干个相互重叠的子问题；
2. **分析**：分析问题是否满足最优性原理或者优化原则，找出动态规划函数的递推式；
3. **求解**：利用递推式**自底向上**计算，实现动态规划过程。

动态规划法利用问题的优化原则，以**自底向上**的方式从子问题的最优解逐步构造出整个问题的最优解。

最优子结构？
动态规划算法基本步骤？

例：一个最短路径问题

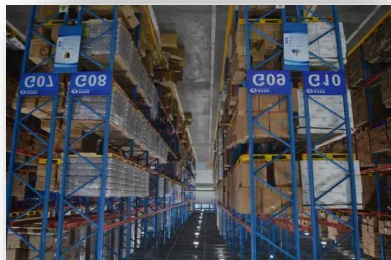
- **问题：** 找任意起点到任意终点的一条最短路径

（例如：加工厂的选址）

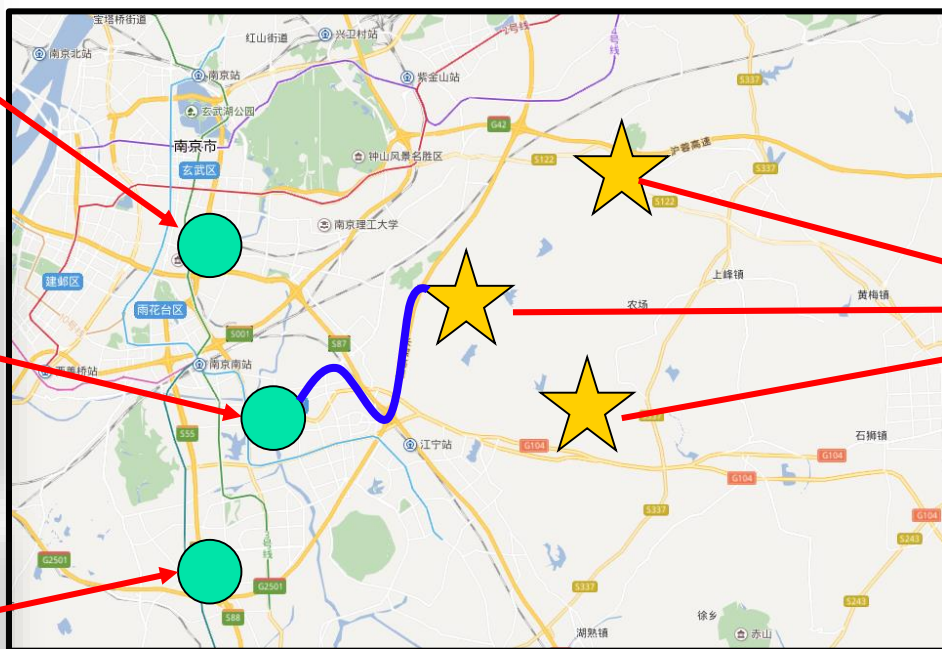
仓库1



仓库2



仓库3



加工厂
候选地址

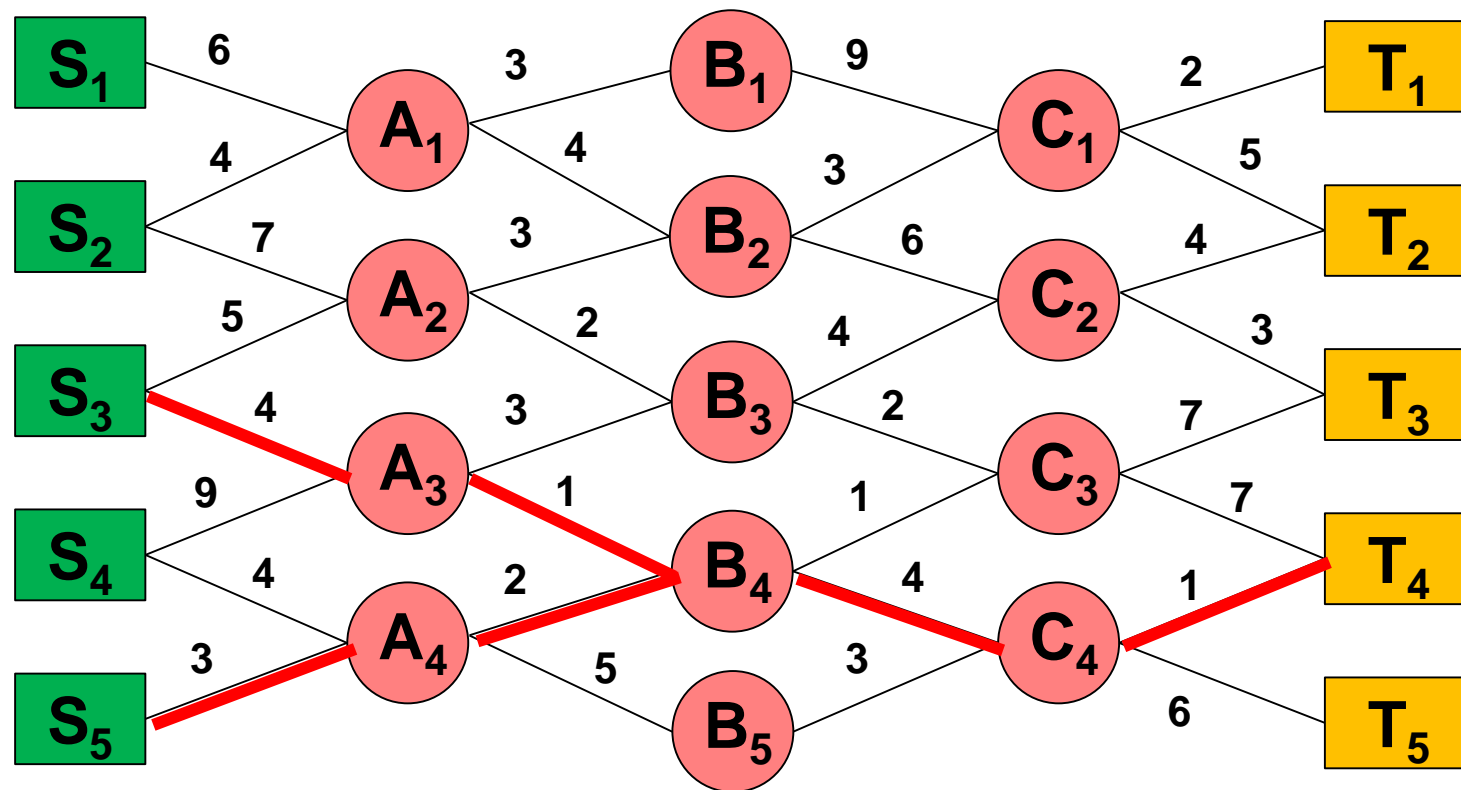
例：一个最短路径问题

- **问题：** 找任意起点到任意终点的一条最短路径
- **输入：**
 - 起点集合 $\{S_1, S_2, \dots, S_n\}$
 - 终点集合 $\{T_1, T_2, \dots, T_m\}$
 - 中间结点集合，边集，对于任意边 e 有长度
- **输出：** 一条从起点到终点的最短路



例：一个最短路径问题

■ 一个实例



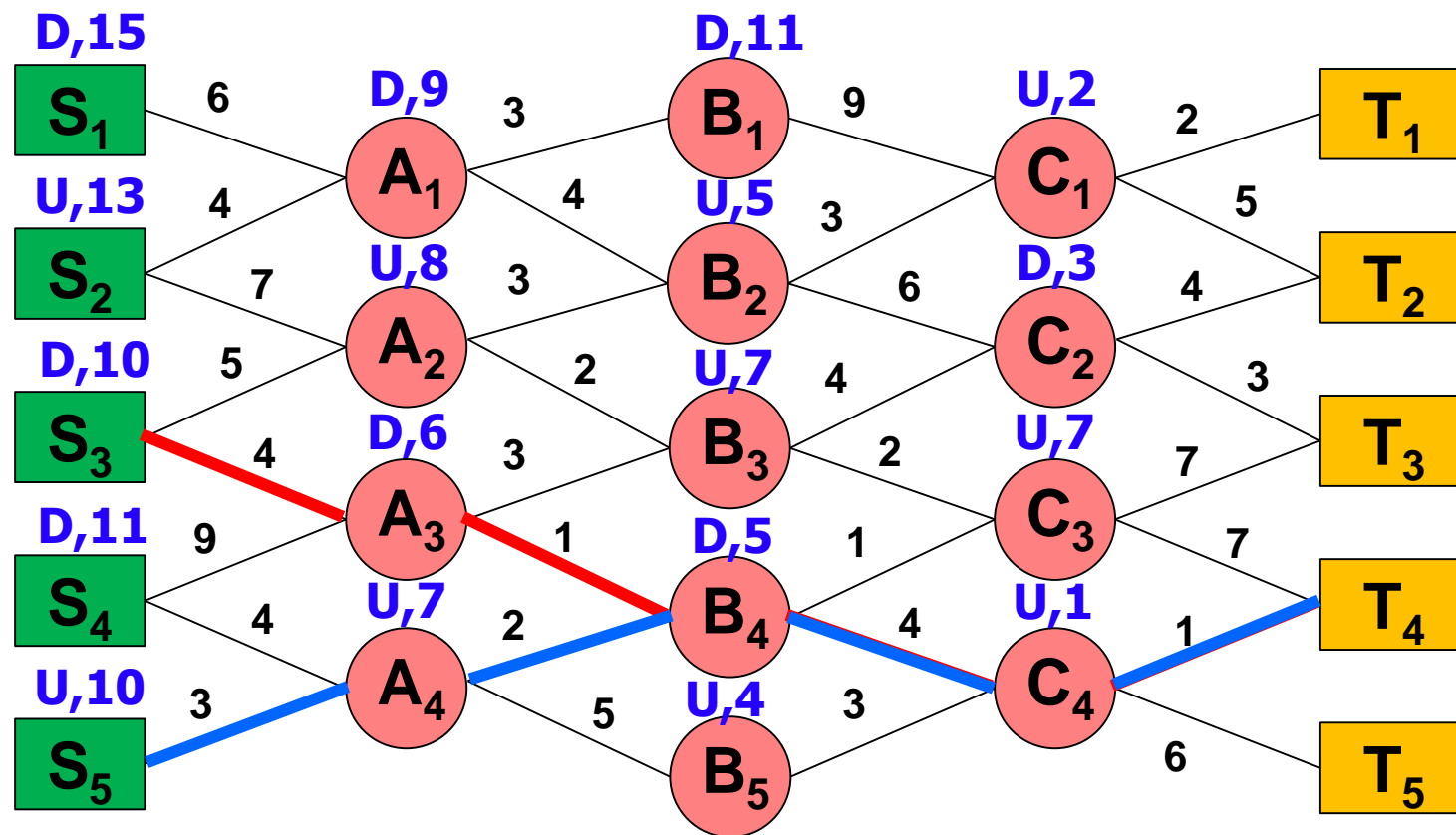


例：一个最短路径问题

- **蛮力算法/穷举法：** 考察每一条从某个起点到某个终点的路径，计算长度，从中找出最短路径。
- 在上述实例中，如果网络的层数为 k ，那么路径条数将接近于 2^k 。
- **动态规划算法：** 多阶段决策过程。每一步求解的问题是后面阶段求解问题的子问题。每步决策将依赖于以前步骤的决策结果。

例：一个最短路径问题

■ 动态规划求解



阶段4

阶段3

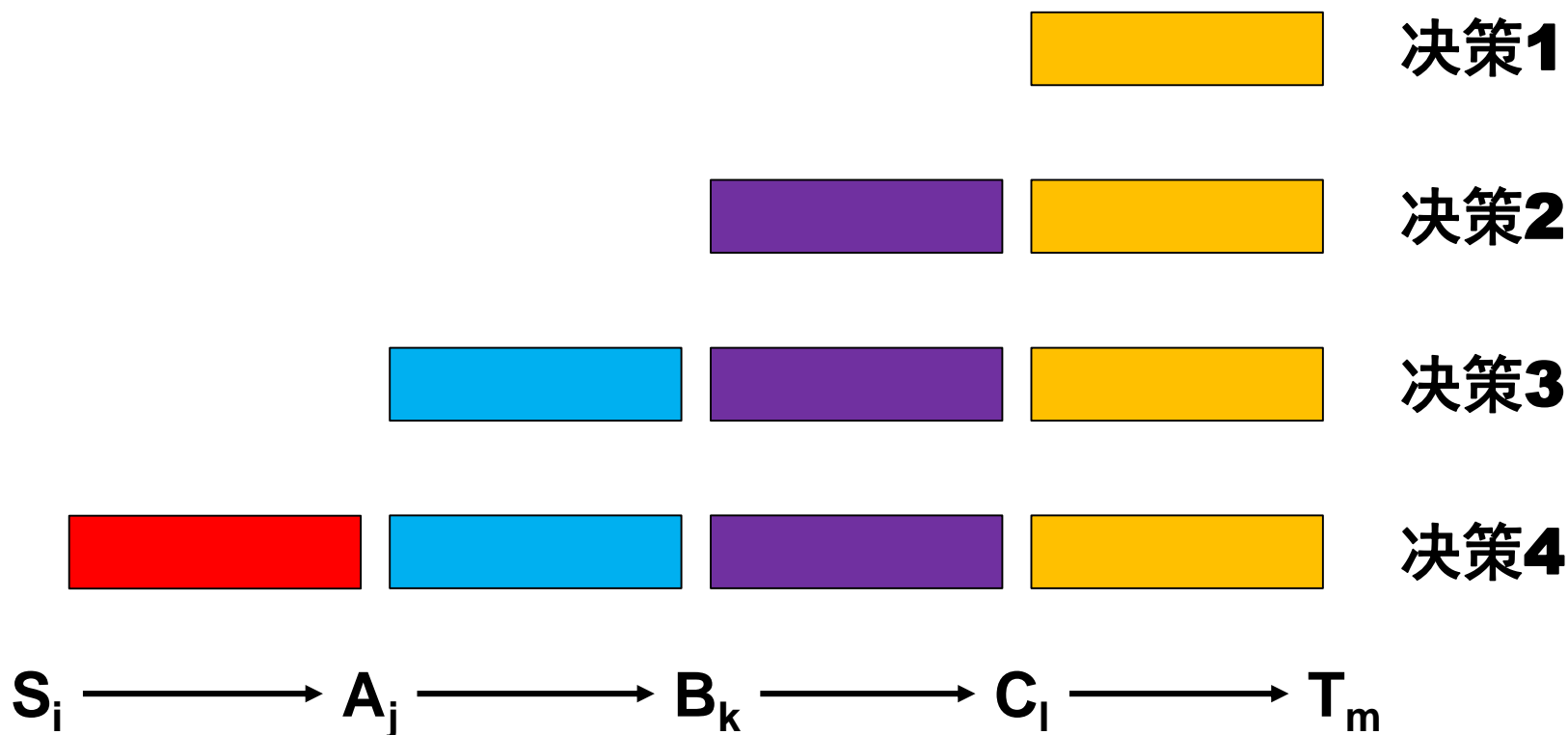
阶段2

阶段1

例：一个最短路径问题

■ 子问题界定

后边界不变，前边界前移



例：一个最短路径问题

■ 最短路径的依赖关系

$$F(C_l) = \min_m \{C_l T_m\}$$

决策1

$$F(B_k) = \min_l \{B_k C_l + F(C_l)\}$$

决策2

$$F(A_j) = \min_k \{A_j B_k + F(B_k)\}$$

决策3

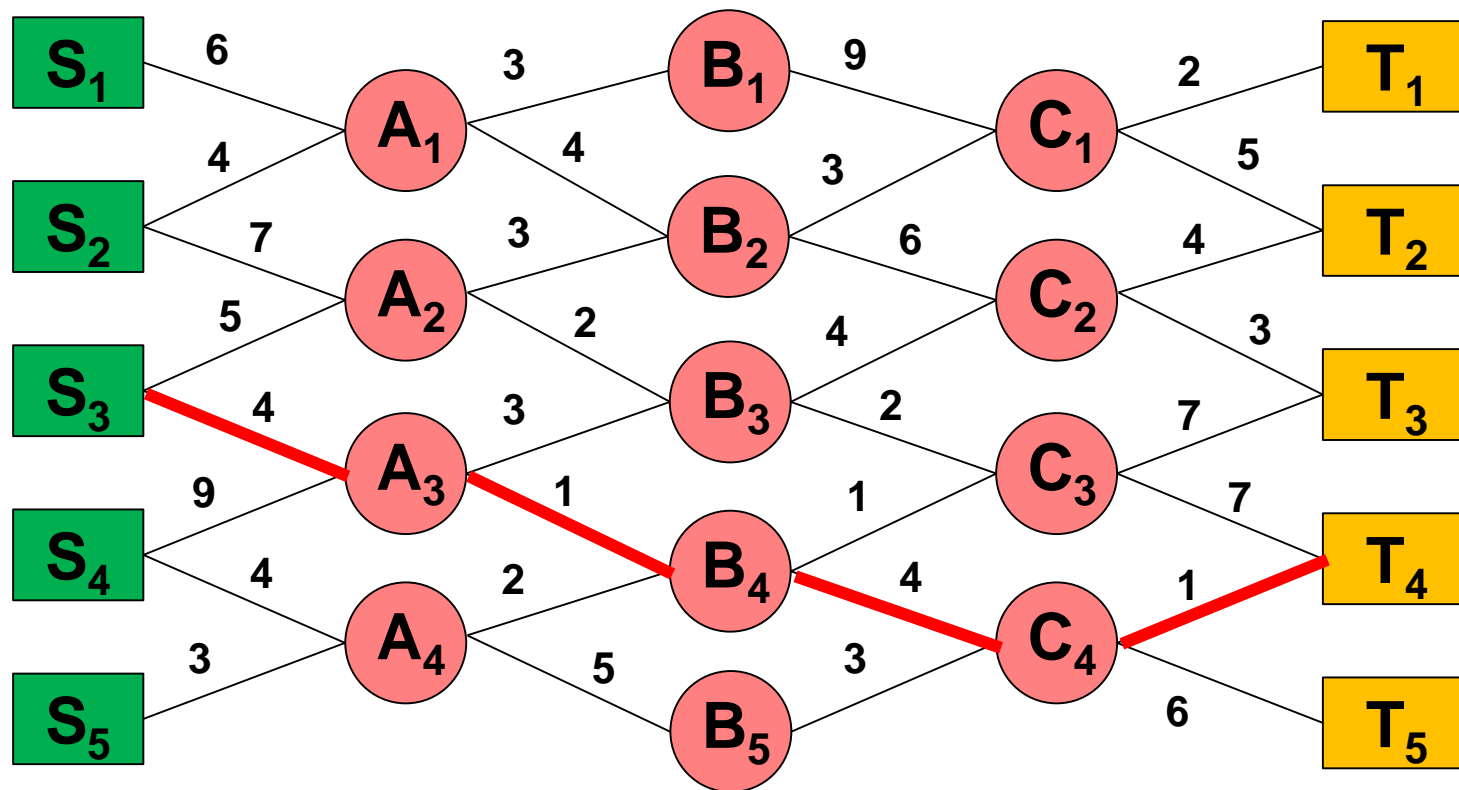
$$F(S_i) = \min_j \{S_i A_j + F(A_j)\}$$

决策4

优化函数值之间存在依赖关系

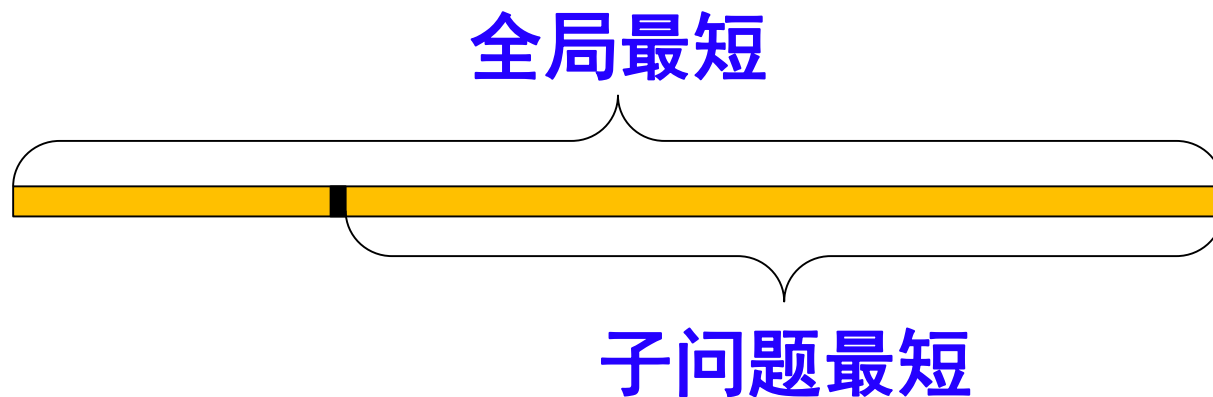
优化原则：最优子结构性质

- 优化函数的特点：**任何最短路的子路径相对于子问题始、终点最短



优化原则：最优子结构性质

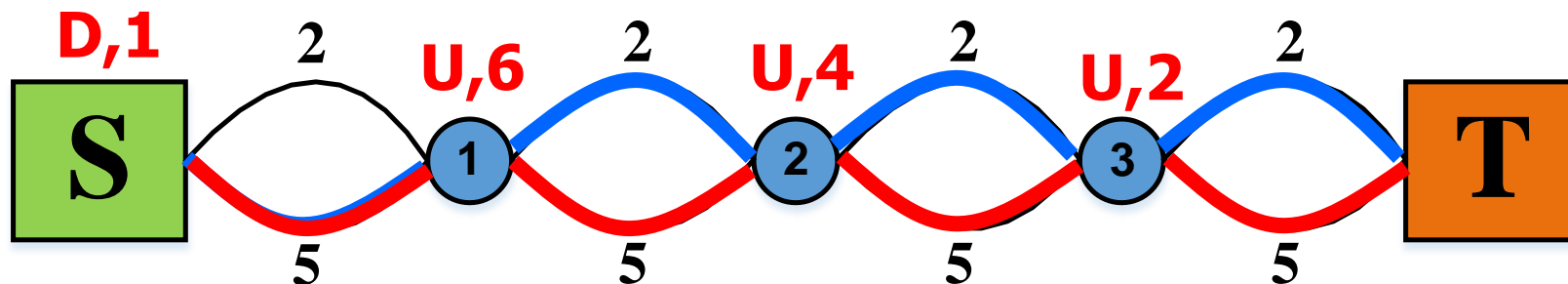
- **优化函数的特点：**任何最短路的子路径相对于子问题始、终点最短



- **优化原则：**一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列

一个反例

- 求总长模10的最小路径



➔ 动态规划算法的解：下、上、上、上

- 最优解：下、下、下、下

不满足优化原则，不能用动态规划！！！！



小结

动态规划(Dynamic Programming)

- 求解过程是多阶段决策过程，每步处理一个子问题，可用于求解组合优化问题
- 适用条件：问题要满足优化原则或者最优子结构性质，即：一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列