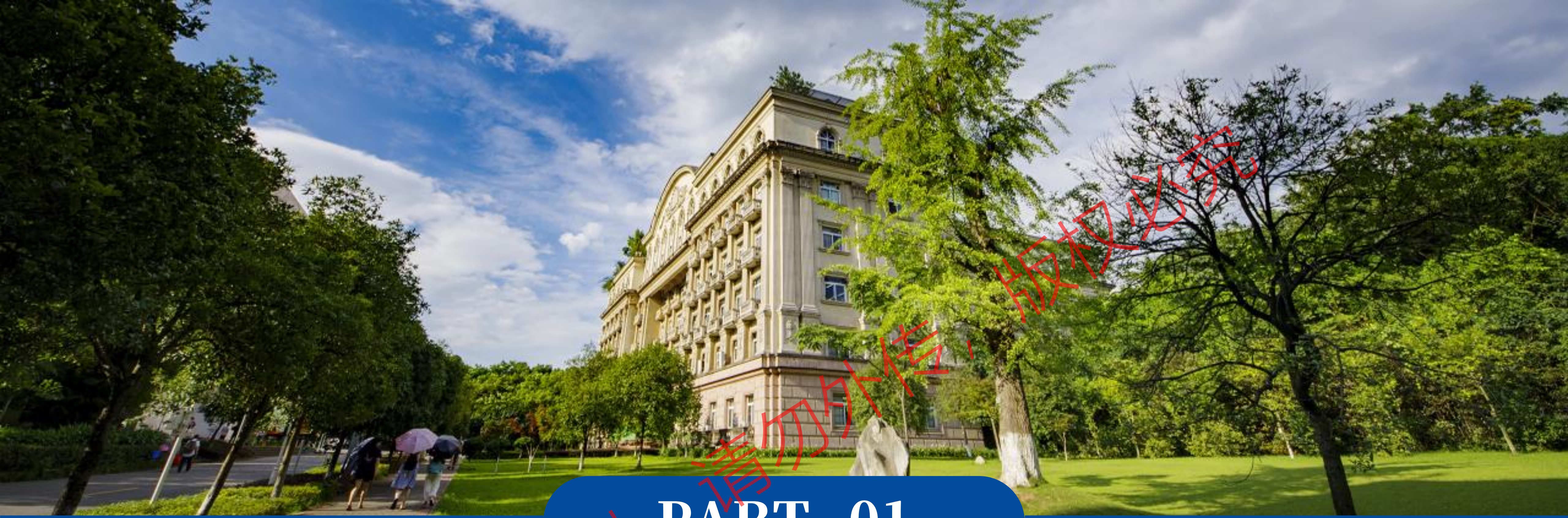
回溯算法

张里博
lbzhang@swu.edu.cn

目录



PART 01

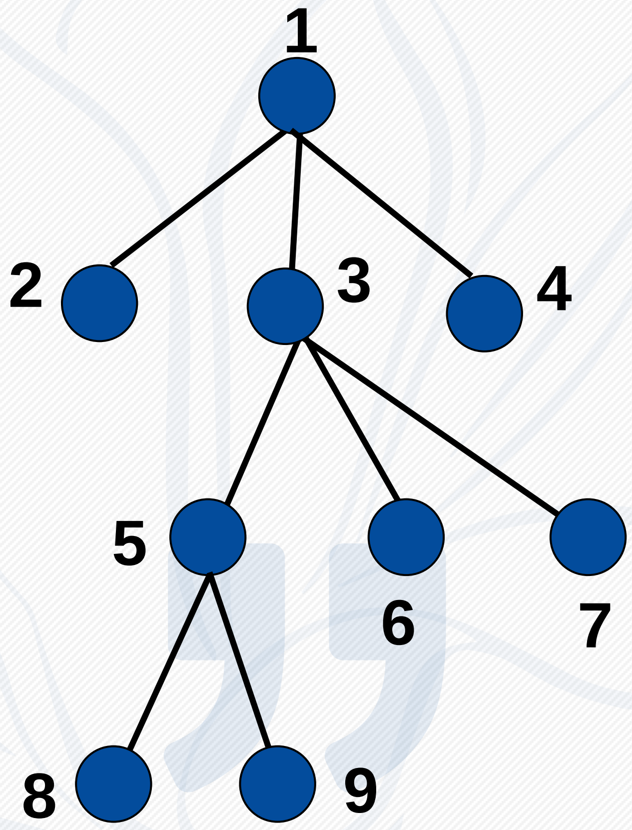
回溯算法的基本思想

Backtracking Algorithm

回溯算法的基本思想

- 组合优化问题中，解是一个向量（每个分量的取值是有限个）。

- 回溯算法将解空间看作树形结构，每一个结点对应于解的一个分量，每一层表示一个分量的取值



- 算法思路：
- 搜索整个解空间，寻找所有可行解向量

深度优先与宽度优先

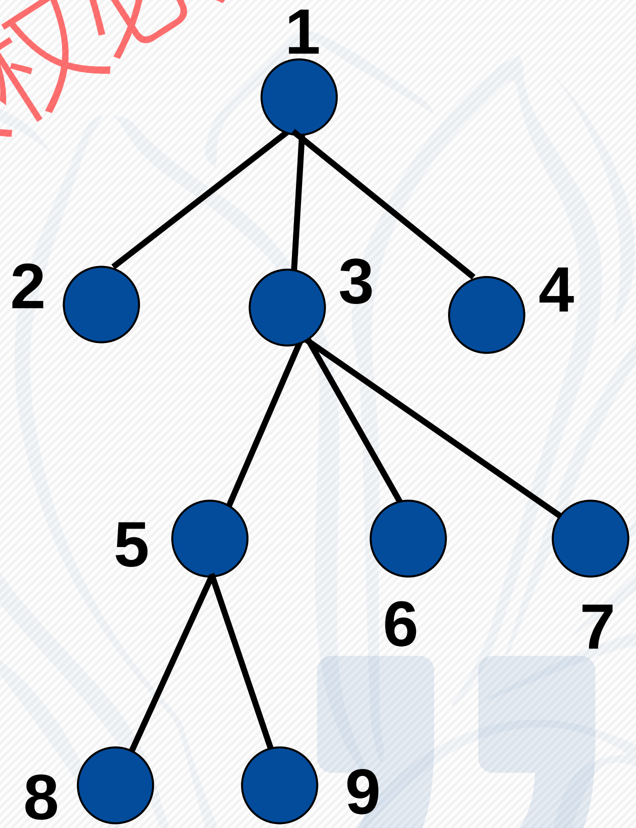
- ■ 深度优先与宽度优先
- ➢ 目标：遍历整棵树
- ➢ 宽度优先：从左往右，从上至下；
- ➢ 深度优先：从上往下，从左至右；

宽度优先访问顺序：

1 →2 →3 →4 →5 →6 →7 →8 →9

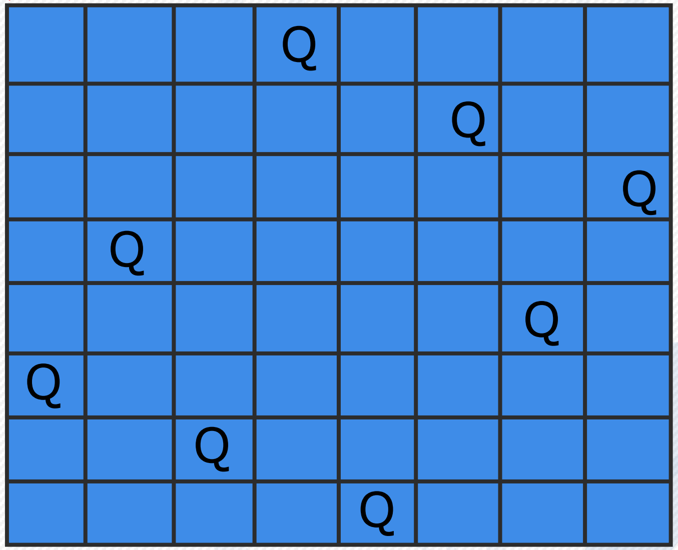
深度优先访问顺序：

1 →2 →3 →5 →8 →9 →6 →7 →4



n后问题

- 在n×n格的棋盘上放置彼此不受攻击的n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。
  - 原因：国际象棋的规则，皇后可以攻击处在同一行或同一列或同一斜线上的棋子；

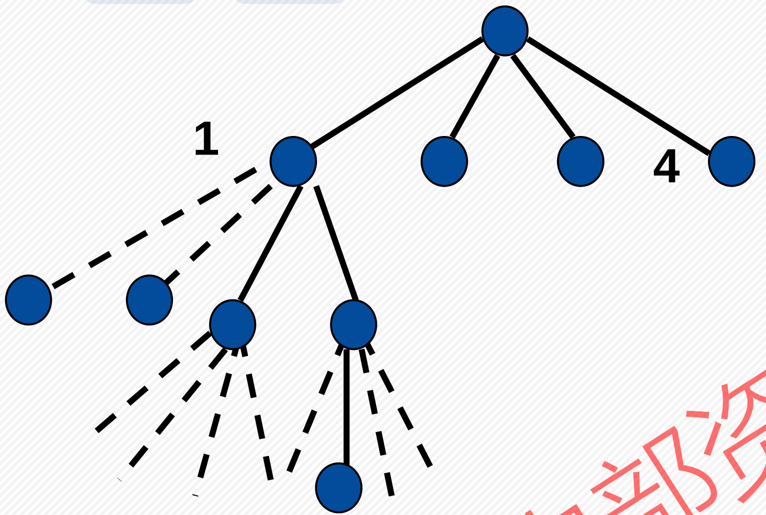


- 求所有的放置方法

- 解是一个n维向量<$x_1, x_2, \ldots, x_n$>，其中x_i表示**第i行放置皇后的列号{1, 2…n}；**

4后问题的搜索树

解是4维向量<x_1, x_2, x_3, x_4>



1. 原始搜索树：完全4叉树；
也是蛮力算法的搜索树

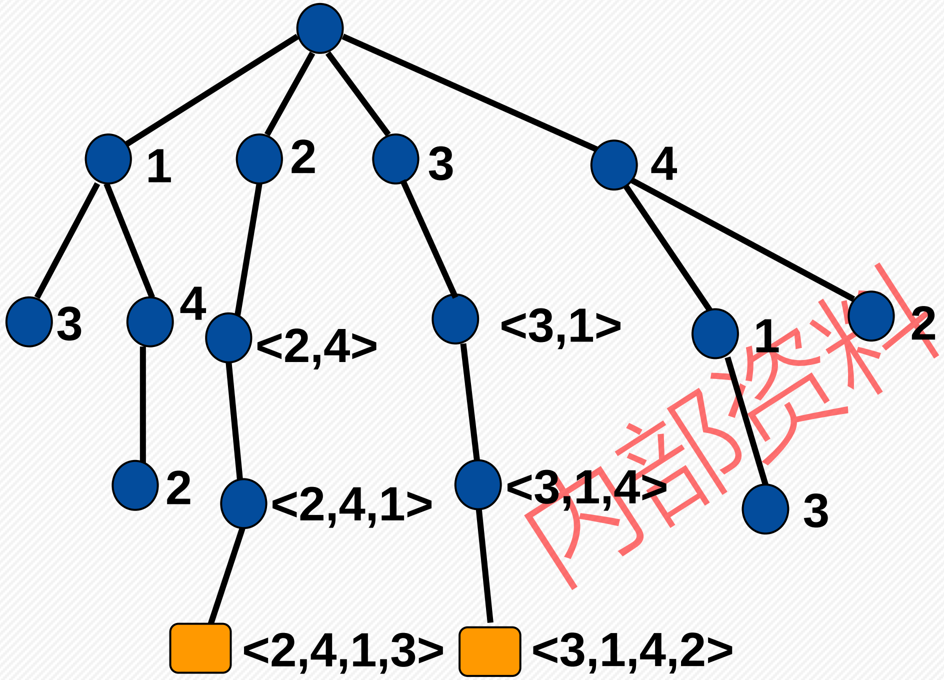
2. 第i层选择解向量中第i个分量的值；

3. 每个结点有4个儿子，从左向右分别代表选择四个列位置1, 2, 3, 4。

按照深度优先的顺序，跳跃式地搜索所有的可行解向量

跳跃式搜索

解是4维向量<x_1, x_2, x_3, x_4>

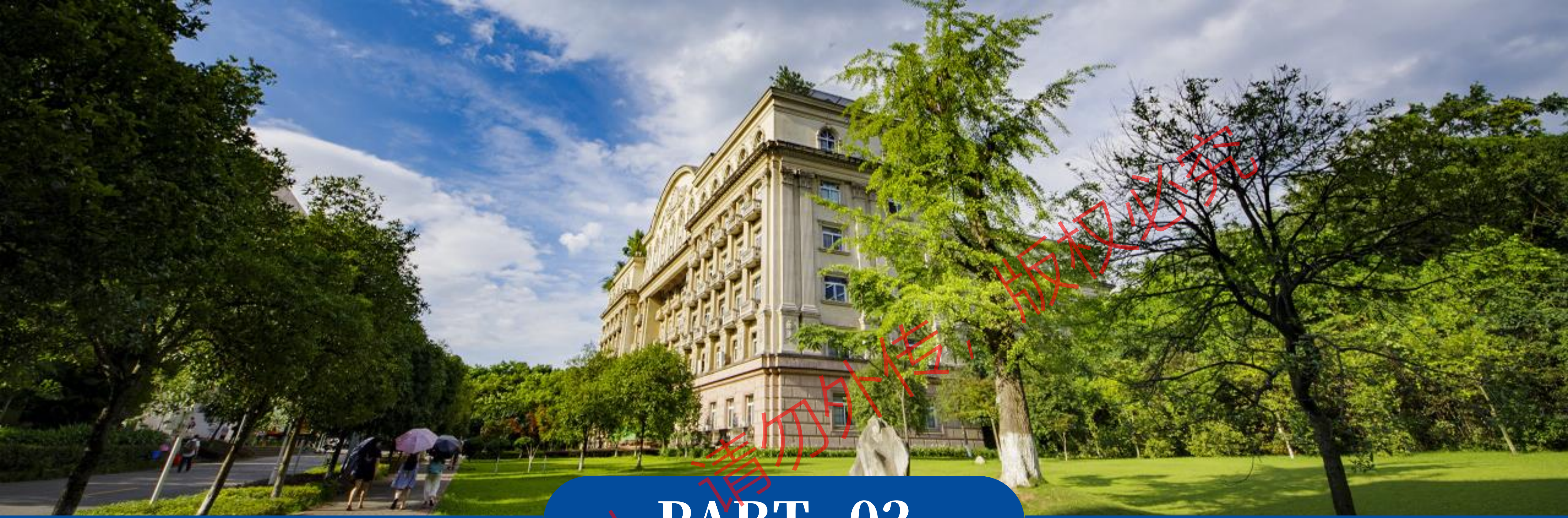


1. 搜索过程中，如可行（不同行，不同列，不同一斜线）就延伸，否则回溯；

2. 存在可行解的前提下，算法运行完成后，搜索树中最深层的树叶是可行解；

跳跃式检索

- 回溯算法会遍历所有的点么？（与蛮力算法的差别）
- 每一个结点对应于解的部分向量，一个可行解对应于搜索树中的（最深层）一个树叶。回溯算法从根节点出发，寻找所有可达的叶节点。
- 满足约束条件（可行）就继续延伸，不满足约束条件（不可能成为可行解）就回溯到父节点，继续探索别的分支。即“可行就延申，不可行就回溯”
- 回溯算法是一种遵照某种规则（避免遗漏）、跳跃式（带裁剪）地搜索解空间的方法。



PART 02

回溯算法的适用条件

Backtracking Algorithm

回溯算法适用的条件

- 设 $P(x_1, x_2, \ldots, x_i)$ 为真，表示向量$<x_1, x_2, \ldots, x_i>$ 满足某个性质（约束条件）。

- **多米诺性质：**

$$P(x_1, x_2, ..., x_{k+1}) \rightarrow P(x_1, x_2, ..., x_k) \qquad 0 < k < n$$

- **逆否命题：**

$$\neg P(x_1, x_2, ..., x_k) \rightarrow \neg P(x_1, x_2, ..., x_{k+1}) \qquad 0 < k < n$$

- k维向量不满足约束条件（不可行），扩张到k+1维后仍然不会满足。因此，回溯（跳跃搜索）而不丢解。

多米诺性质

- 求不等式的整数解：
- $5x_1+4x_2-x_3 \le 10,\ 1 \le x_i \le 3,\ \ i=1,2,3$
- $P(x_1, \ldots, x_k)$：将$x_1, x_2, \ldots, x_k$代入原不等式的相应部分，不等式成立。
- $x_1=1,\ x_2=2,\ \ x_3=3;$
- $5x_1+4x_2-x_3 \le 10 \ \not\Rightarrow\ 5x_1+4x_2 \le 10$恒成立
- 不满足多米诺性质

多米诺性质

- 求不等式的整数解：
- $5x_1+4x_2-x_3 \leq 10$, $1 \leq x_i \leq 3$, $i=1,2,3$

- 通过变换使得该问题满足多米诺性质 ？

- 变换：令 $x_3'=3-x_3$，则 $0 \leq x_3' \leq 2$；
- $5x_1+4x_2+x_3' \leq 13$，$1 \leq x_1, x_2 \leq 3, 0 \leq x_3' \leq 2$

- $5x_1+4x_2+x_3' \leq 13 \Rightarrow 5x_1+4x_2 \leq 13$恒成立

n后问题

- 在n×n格的棋盘上放置彼此不受攻击的n个皇后，任何2个皇后不放在同一行或同一列或同一斜线上。

- 解是一个n维向量<$x_1, x_2, \ldots, x_n$>，其中x_i表示**第i行放置皇后的列号{1, 2…n}**;

- 满足多米诺性质么？
- 若前k个皇后中有位置不可行，则继续延伸也不会得到可行解。（多米诺性质的逆否命题）

回溯算法适用的条件

(1) 适用问题：求解搜索问题（分量取值是离散的且有限个）和组合优化问题

(2) 搜索空间：树，结点对应解分量的取值，（算法完成后的最深层）树叶代表可行解

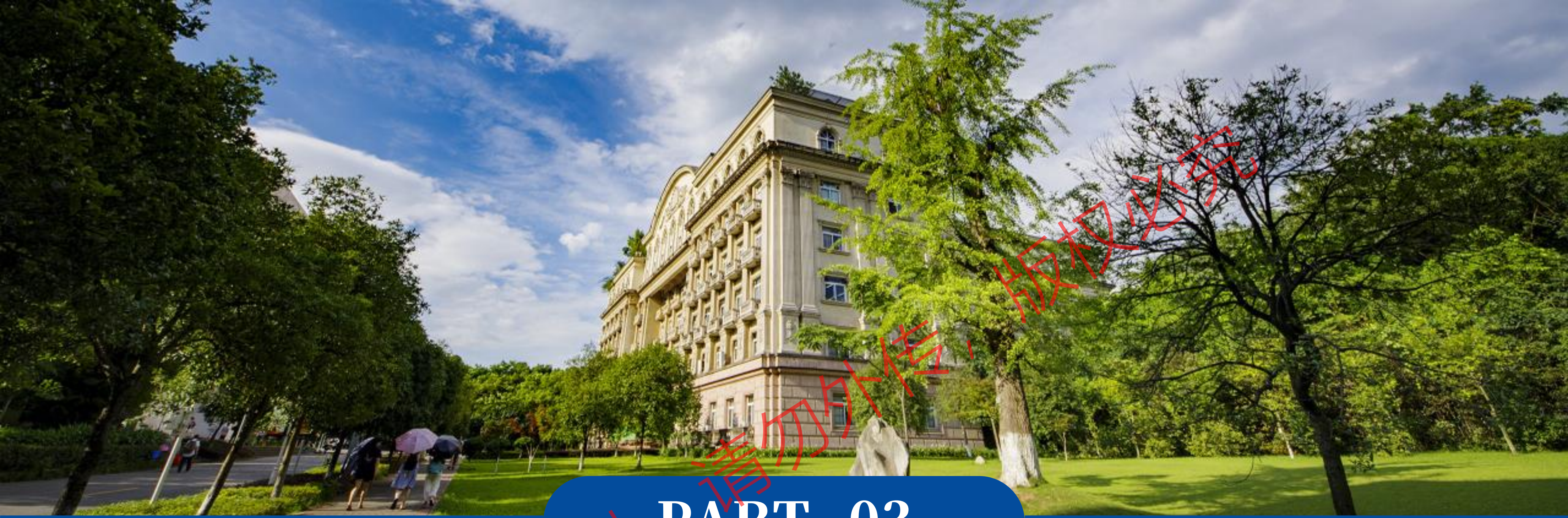
(3) 搜索过程：采用系统的方法隐含遍历搜索树

(4) 搜索策略：深度优先

(5) 结点分支判定条件：

满足约束条件---分支扩张解向量

不满足约束条件，回溯到该结点的父结点



PART 03

回溯算法的设计

Backtracking Algorithm

回溯算法的设计步骤

(1) 定义搜索问题的解向量和每个分量的取值集合

解向量为 $<x_1, x_2, \ldots, x_n>$

确定 x_i 的理论上可能取值的集合为 X_i, $i = 1, 2, \ldots, n$.

(2) 确定结点儿子的排列规则

(3) 判断是否满足多米诺性质

(4) 搜索策略——深度优先等

(5) 确定每个结点分支约束条件

(6) 确定存储搜索路径的数据结构

4后问题的设计步骤

回溯算法的实现

- 当 $x_1, x_2, \ldots, x_{k-1}$ 确定后，计算 x_k 当前（尚未探索的）实际可能取值集合 S_k，$S_k \subseteq X_k$

- 可行就延伸：
- 如果当前实际可能取值集合 S_k 不为空，从 S_k 中选一个值赋给 x_k，并从 S_k 中删除对应值，计算下一个节点的当前实际可能取值集合 S_{k+1}

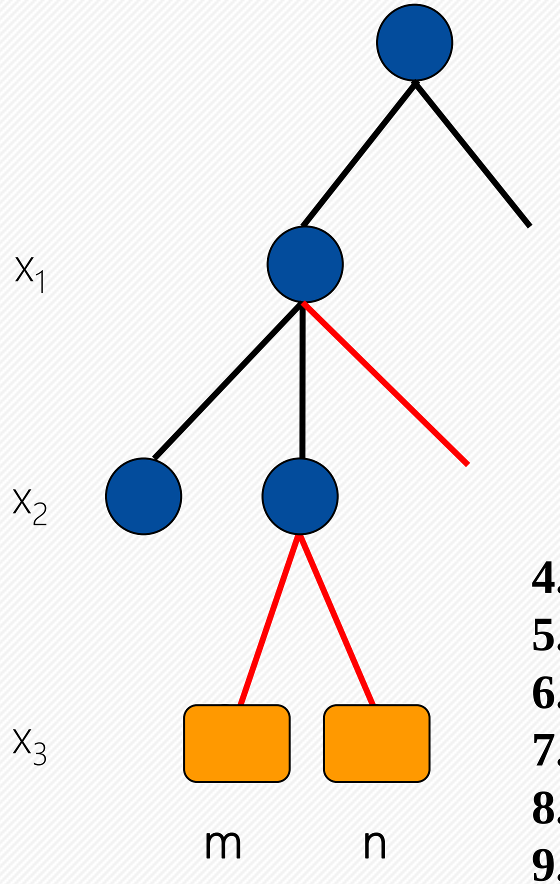
- 不可行就回溯：
- 如果当前实际可能取值集合 S_k 为空，回溯至 S_{k-1}，继续探索其他分支（S_{k-1} 中选一个值赋给 x_{k-1}）

迭代实现

算法5.3 Backtrack(n)

```
1.  对于 i =1, 2, … , n ，确定 X_i
2.  T ← ∅;                        %存储可行解        ⇐ 确定初始取值
3.  k←1；计算 S_k ；
4.  while  S_k ≠ ∅  do                            ⇐ 满足约束分支搜索
5.       x_k←S_k 中最小值；S_k←S_k–{x_k};           ⇐ 延伸
6.       if  k<n  then
7.            k←k+1; 计算 S_k ；
8.       else
9.            T ← T ∪ <x_1,x_2,…,x_n>;
10.      end
11.  end
12.  if  k>1  then
13.           k←k-1；  goto 4 ；                   ⇐ 回溯
14.  end
15.  if  T == ∅  then
16.           return 0;
17.      else
18.           return T;
19.   end
```

实例中的迭代实现



已知解的长度为3，
此时：$\langle x_1, x_2\rangle$，$S_2 \neq \varnothing$；
$S_3 == \{m, n\}$

```
while S_k ≠ ∅ do
    x_k ← S_k中最小值; S_k ← S_k − {x_k};
    if k<n then
        k ← k+1; 计算S_k ;
    else
        T ← T ∪ <x_1,x_2,…,x_n>;
    end
end
```

$S_3 \neq \varnothing$，符合while

$x_3 \leftarrow m$，
$S_3 \leftarrow \{n\}$，
$\boxed{\langle x_1, x_2, m\rangle}$

$S_3 \neq \varnothing$，符合while

$x_3 \leftarrow n$，
$S_3 \leftarrow \varnothing$，
$\boxed{\langle x_1, x_2, n\rangle}$

$S_3 = \varnothing$，不符合while

if语句，$k \leftarrow k-1$
返回对S_2进行判断

S_2不为空
•
•
•

• • • •

递归实现

递归回溯

算法5.2 ReBacktrack(n)

输入：n

输出：所有的解

1. **for k←1 to n *do***
2. 计算 X_k;
3. S_k←X_k;
4. **end**
5. **ReBack(1)**

算法5.1 ReBack(k)

1. **if $k>n$ then**
2. 输出$<x_1, x_2, \ldots, x_n>$; //可行解
3. **else**
4. **while $S_k \neq \varnothing$ do**
5. x_k←S_k中最小值;
6. S_k←S_k−{x_k} ;
7. 计算S_{k+1} ;
8. **ReBack(k+1)** ;
9. **end**
10. **end**

递归实现

while $S_1 \neq \varnothing$ do

　…

　while $S_2 \neq \varnothing$ do

　　…

　　while $S_3 \neq \varnothing$ do

　　　…

　　　while $S_4 \neq \varnothing$ do

　　　　…

　　　end

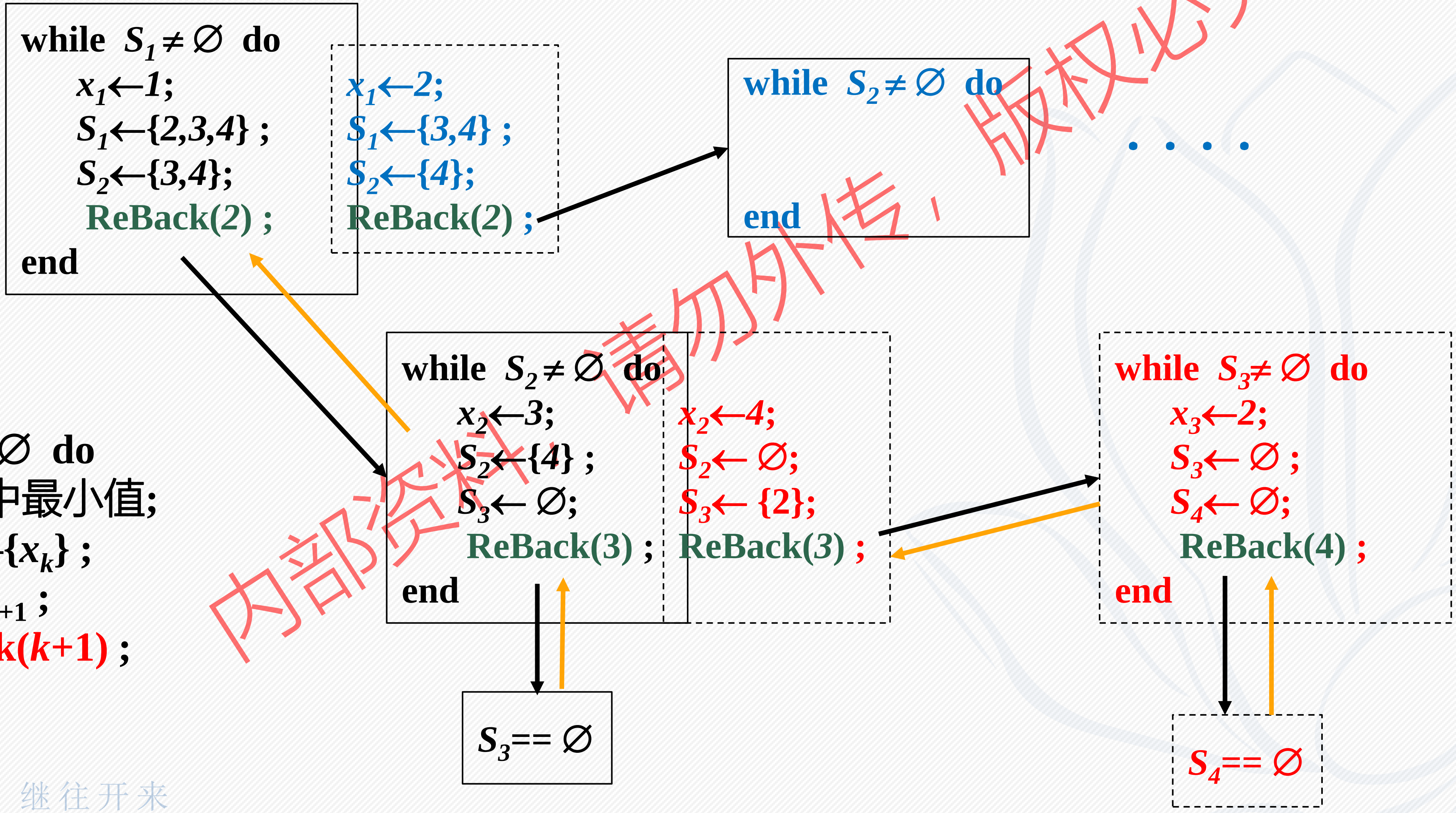
　　end

　end

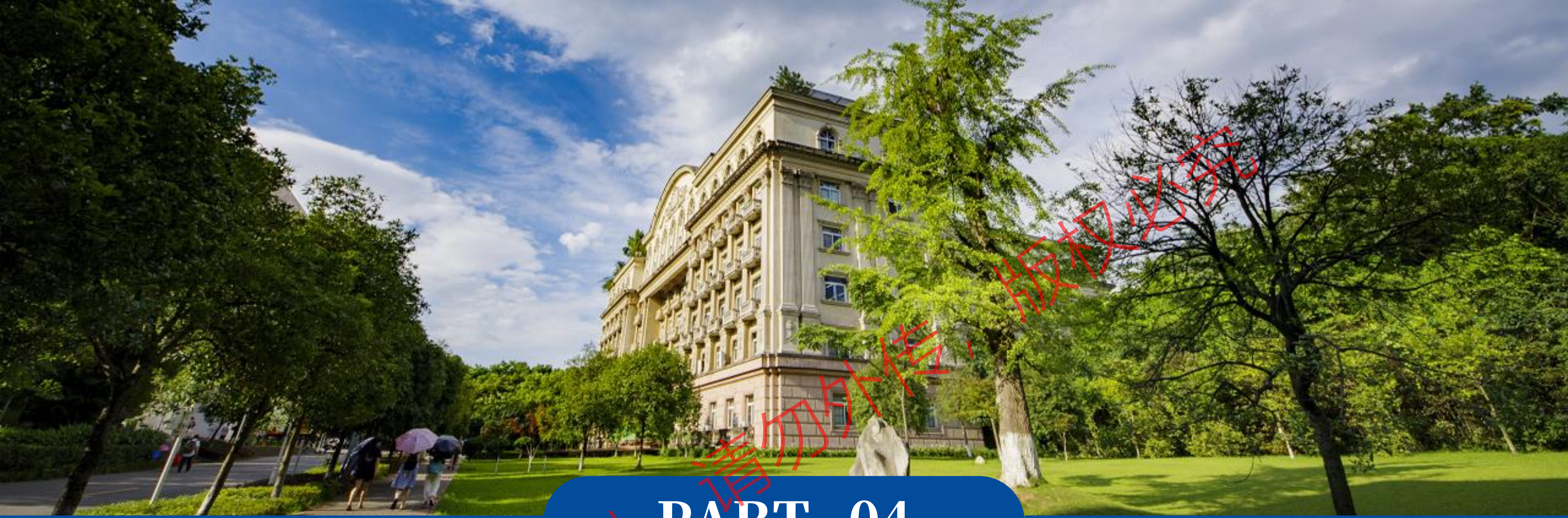
end

4. while $S_k \neq \varnothing$ do
5. 　$x_k \leftarrow S_k$中最小值;
6. 　$S_k \leftarrow S_k - \{x_k\}$;
7. 　计算S_{k+1} ;
8. 　ReBack(k+1) ;
9. end

递归实现



4. while $S_k \neq \varnothing$ do
5. $x_k \leftarrow S_k$中最小值;
6. $S_k \leftarrow S_k - \{x_k\}$;
7. 计算S_{k+1} ;
8. ReBack(k+1) ;
9. end



PART 04

回溯算法的效率估计

搜索树结点数的估计

- 回溯算法跳跃式地遍历整个空间，不会遍历搜索树上的所有节点；
- 回溯算法的时间复杂度取决于算法实际遍历的节点数和每个节点上的工作量
- 采用蒙特卡洛(Monte Carlo)方法估计实际遍历的节点

搜索树结点数的估计

Monte Carlo方法

1. 从根开始，随机选择一条路经，直到不能分支为止；

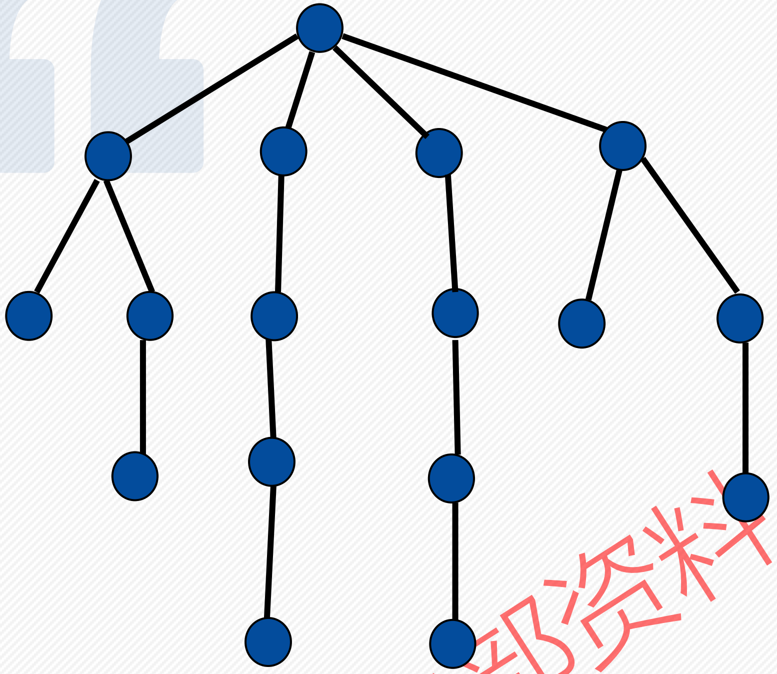
依次对 x_i 赋值，每个 x_i 的取值是从当时的 S_i 中随机选取，直到向量不能扩张为止

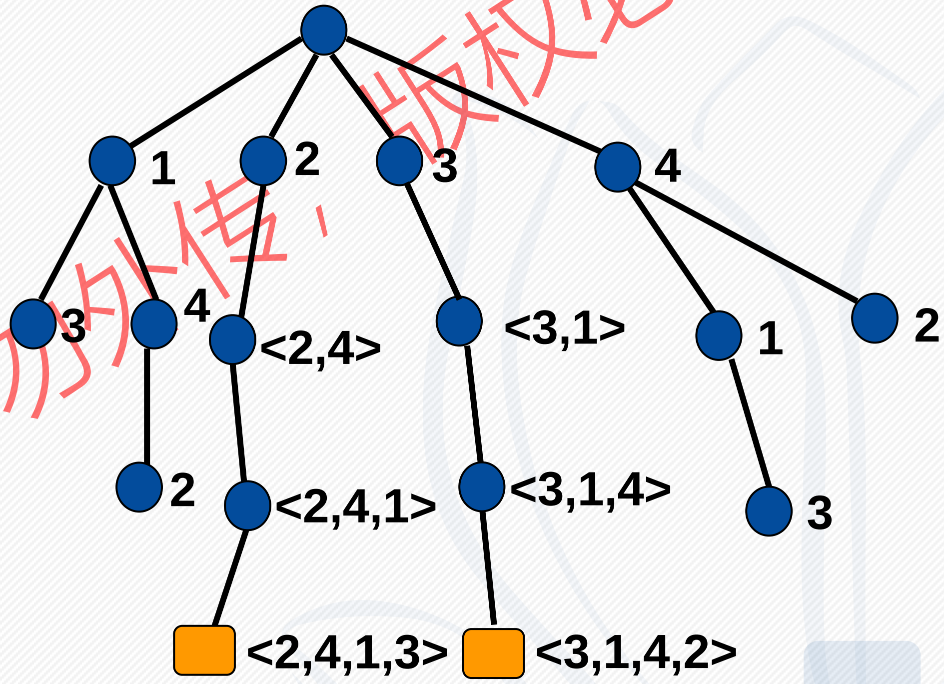
2. 根据已探索的路径及S_i，估计整棵搜索树的节点数；

假设搜索树中每层的其他 $\|S_i\|_0-1$ 个分支与以上随机选出的路径一样，计算整颗搜索树的点数.

3. 重复步骤 1 和 2，将每次结点数进行平均。

4后问题



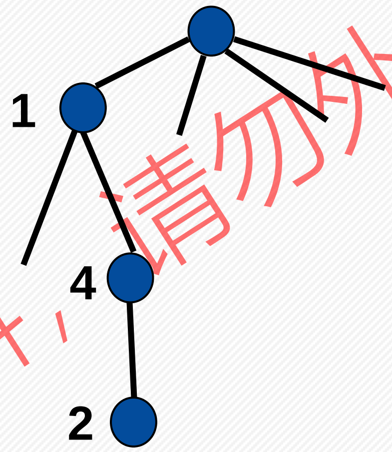


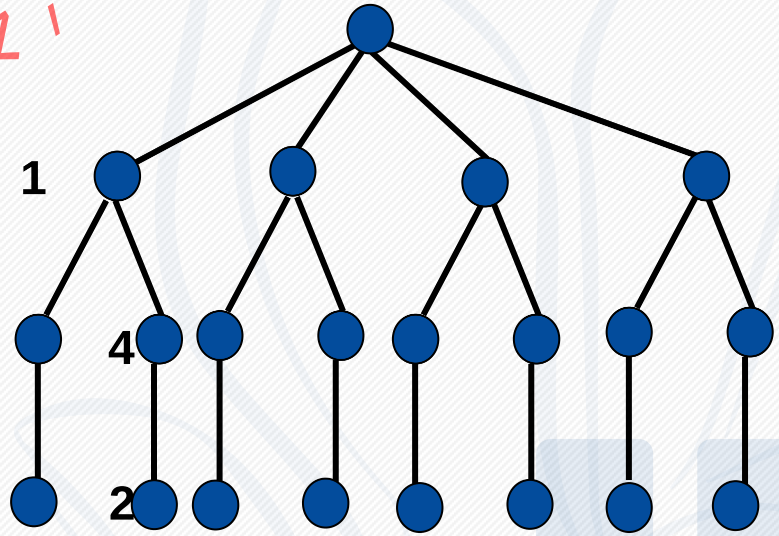
算法实际访问的结点数为 17

4后问题

随机选择的路径1

$S_1=\{1,2,3,4\},\ x_1=1$
$S_2=\{3,4\},\ x_2=4$
$S_3=\{2\},\ x_3=2$



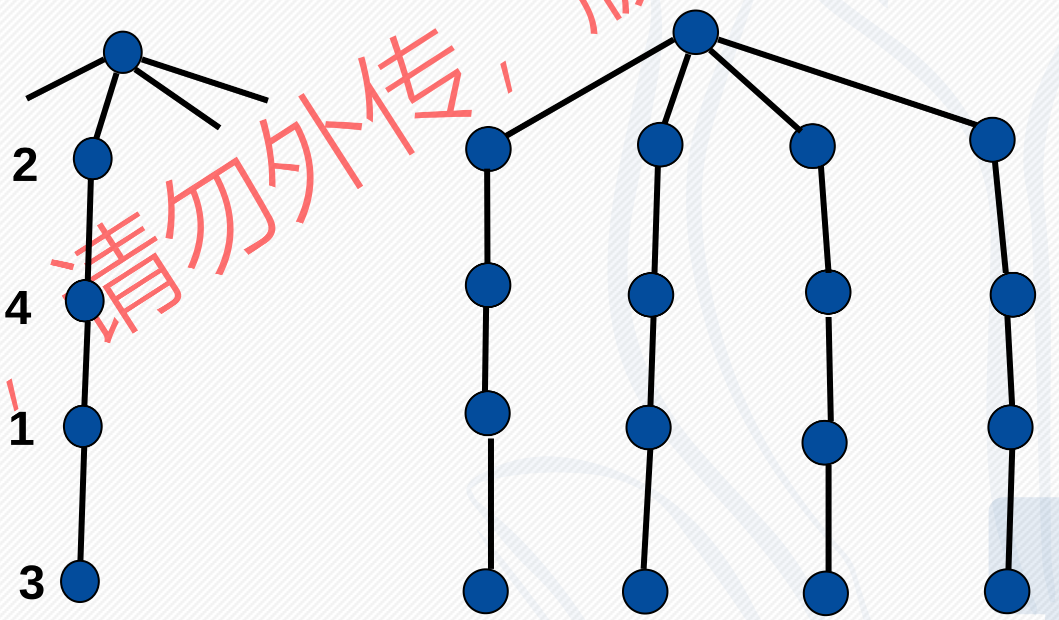


结点数21

4后问题

随机选择的路径2

$S_1=\{1,2,3,4\}, x_1=2$
$S_2=\{4\}, x_2=4$
$S_3=\{1\}, x_3=1$
$S_4=\{3\}, x_4=3$



结点数17

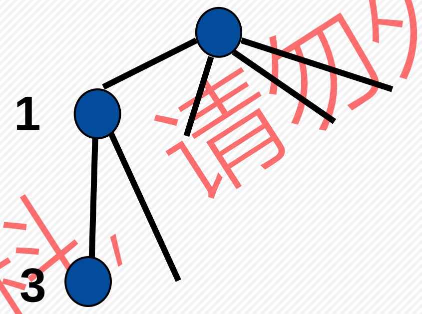
P123，最下面公式(2)，结点数17

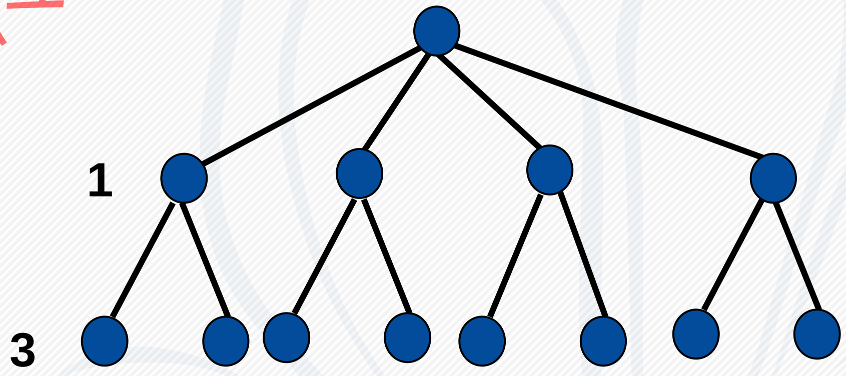
搜索树结点数的估计

随机选择的路径3

$S_1=\{1,2,3,4\}$, $x_1=1$

$S_2=\{3,4\}$, $x_2=3$





结点数13

搜索树结点数的估计

一次抽样

- 从树根向下计算，随机选择，直到不能分支。

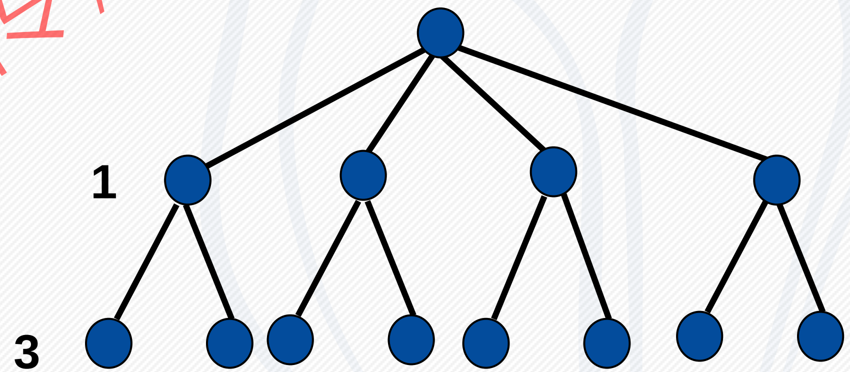
r_2为上层结点数

r_1为本层结点数

$r_1 = r_2$*分支数$= r_2 * \| S_k \|_0$

n为解向量的个数

m为本次取样得到的树结点总数



$r_2 = 4$

$r_1 = r_2 \cdot 2 = 8$

搜索树结点数的估计

子过程的伪码

r_1为扩张后结点总数
r_2为扩张前结点总数

算法Estimate(n)

```
m←1; r2←1; k←1 // m 为结点总数
while k≤n do
    if Sk == Ø then
        return m
    end
    r1←||Sk||0 * r2;      // r1为扩张后结点总数
    m←m+r1 ;
    xk←随机选择Sk的元素;
    k←k+1;
    r2←r1;
    计算Sk ;
end
```

不能分支

随机选择一步

搜索树结点数的估计

子过程的伪码

```
算法Estimate(n)
m←1; r←1; k←1 // m 为结点总数
while k≤n do
    if Sk == Ø then
        return m
    end
    r←|| Sk ||0 * r;     // r为扩张后结点总数
    m←m+r;              // r2为扩张前结点总数
    xk←随机选择Sk的元素 ;
    k←k+1;
end
```

不能分支

随机选择一步

搜索树结点数的估计

- **估计结果（多次抽样取平均值）**

假设 4 次抽样测试：

case1：1次，

case2：1次，

case3：2次，

平均结点=(21×1+17×1+13×2)/4=16

搜索空间实际访问的结点数为 17

搜索树结点数的估计

- **伪码（多次采样，取平均值）**

- Monte Carlo

输入：n 为皇后数，t 为抽样次数
输出：sum, 即t 次抽样路长平均值
1. sum←0
2. for i ←1 to t do // 取样次数 t
3. m←Estimate(n); // m为结点数
4. sum ← sum + m;
5. end
6. return sum / t

一次抽样结果

小结

◆**Monte Carlo 方法**

目的：估计搜索树真正访问结点数

步骤：

随机抽样，选择一条路径

用这条路径代替其他路径

逐层累加树的结点数

多次选择，取结点数的平均值

影响回溯算法的因素

影响回溯算法的因素：

- ■搜索树的结构：分支和树深
- ■解的分布：是否均匀，深度如何
- ■约束条件的复杂性

- ■改进策略：
- ■节点少的分支优先搜索
- ■利用搜索树的对称性裁剪
- ■分解为子问题
- ■加快回溯的进度（增加约束条件）

(1) 定义搜索问题的解向量和每个分量的取值集合

解向量为 $\langle x_1, x_2, \ldots, x_n \rangle$

确定 x_i 的理论上可能取值的集合为 X_i , $i = 1, 2, \ldots, n$.

(2) 确定结点儿子的排列规则

(3) 判断是否满足多米诺性质

(4) 搜索策略——深度优先等

(5) 确定每个结点分支约束条件

(6) 确定存储搜索路径的数据结构

回溯算法小结

(1) 适于求解组合搜索问题及优化问题（离散，不连续）

(2) 求解条件：满足多米诺性质（可行就延伸，不可行就回溯）

(3) 解的表示：解向量，求解是不断扩充解向量的过程

(4) 回溯条件：约束条件（可行）
分支策略：深度优先

(5) 降低时间复杂性的主要途径：节点少的分支优先搜索、利用搜索树的对称性裁剪

回溯算法适用的条件

问题	解性质	解描述向量	搜索空间	搜索方式	约束条件
n后	可行解	$<x_1,x_2,\ldots,x_n>$ x_i：第i 行列号	n叉树	深度优先	彼此不攻击
0-1背包	最优解	$<x_1,x_2,\ldots,x_n>$ x_i=0, 1； x_i=1⇔选i	子集树	深度优先	不超背包重量限制
货郎	最优解	$<i_1=1, i_2,\ldots,i_n>$ 1,2,…,n的排列	排列树	深度优先	选没有经过的城市
特点	搜索解	向量，不断扩张部分向量	树	跳跃式遍历	约束条件 回溯判定