

# 《算法设计与分析》

## 第六章 回溯法

马丙鹏

2024年11月18日



中国科学院大学

University of Chinese Academy of Sciences 1

# 第六章 回溯法

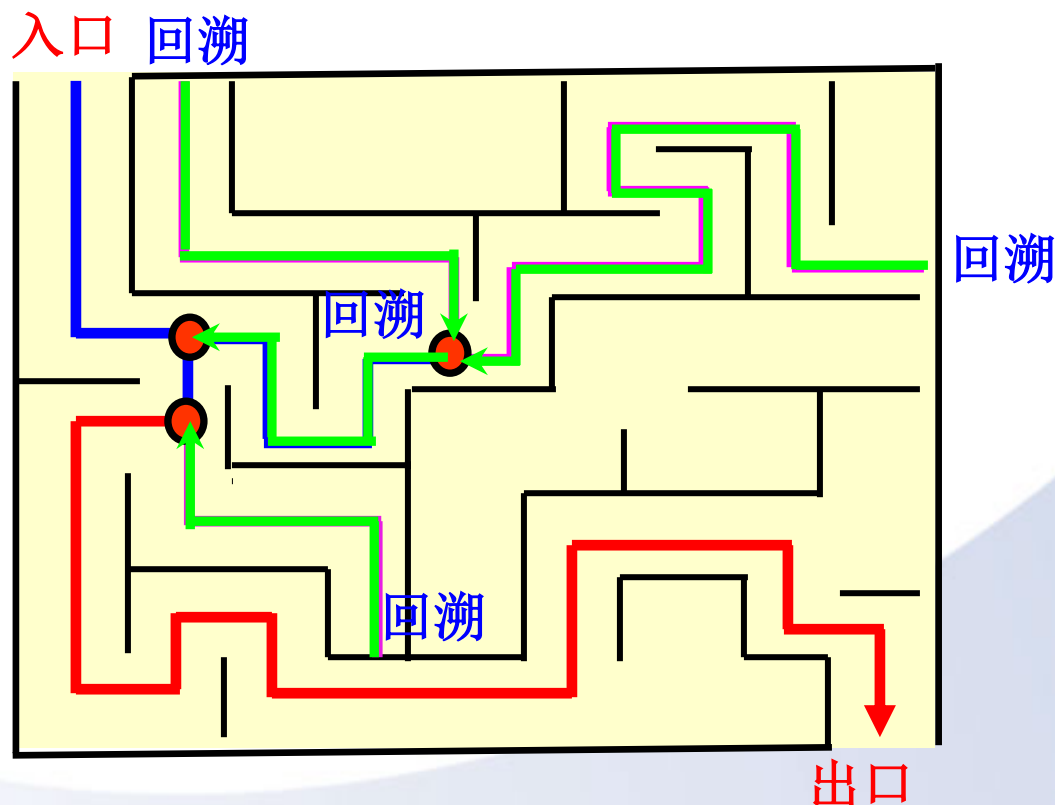
- 6.1 一般方法
- 6.2 8-皇后问题
- 6.3 子集和数问题
- 6.4 图的着色问题
- 6.5 0/1背包问题
- 6.6 哈密顿环
- 6.7 和最小
- 6.8 跳马问题



# 6.1 一般方法

## ■ 回溯法概述

### □ 迷宫游戏



# 6.1 一般方法

## ■ 回溯法概述

- 回溯法是算法设计的基本方法之一。

- 用于求解问题的一组**特定性质的解**或满足某些约束条件的**最优解**。

- 回溯法比**贪心法**和**动态规划法****更一般**的方法。

- 什么样的问题适合用回溯法求解呢？



# 6.1 一般方法

## ■ 可用回溯法求解的问题

- 问题的解可以用一个 $n$ 元组 $(x_1, \dots, x_n)$ 来表示，其中的 $x_i$ 取自于某个有穷集 $S_i$
- 问题的求解目标是求取一个使某一规范函数 $P(x_1, \dots, x_n)$ 取极值或满足该规范函数条件的向量(也可能是满足 $P$ 的所有向量)
- 例子：  $A(1: n)$ 个元素的排序问题
  - 用 $n$ 元组表示解：  $(x_1, x_2, \dots, x_n)$
  - $x_i$ 表示第 $i$ 小元素的下标，取自有穷集 $S_i=[1\dots n]$ ;
  - 规范函数 $P$ :  $A(x_i) \leq A(x_{i+1}), 1 \leq i < n$



# 6.1 一般方法

## ■ 如何求取满足规范函数的元组？

### □ 硬性处理法

- 假定集合 $S_i$ 的大小是 $m_i$ ，于是就有 $m=m_1m_2\dots m_n$ 个 $n$ -元组可能满足函数 $P$ 。
- 所谓硬性处理是构造出这 $m$ 个 $n$ -元组并逐一测试它们是否满足 $P$ ，从而找出该问题的所有最优解。
- 理论上，候选解数量有限，并且通过检查所有或部分候选解能够得到所需解时，上述方法可行。
- 实际中则很少使用，因为候选解的数量通常都非常大(比如指数级，甚至是大数阶乘)，即便采用最快的计算机也只能解决规模较小的问题。



# 6.1 一般方法

## ■ 如何求取满足规范函数的元组？

### □ 回溯或分枝限界法

- 避免盲目求解，对可能的元组进行系统化搜索。
- 在求解的过程中，逐步构造元组分量，并在此过程中，通过不断修正的规范函数(有时称为限界函数)去测试正在构造中的 $n$ 元组的部分向量 $(x_1, \dots, x_i)$ ，看其能否可能导致最优解。
- 如果判定 $(x_1, \dots, x_i)$ 不可能导致最优解，则将可能要测试的 $m_{i+1} \dots m_n$ 个向量一概略去，相对于硬性处理可大大减少计算量。



# 6.1 一般方法

## ■ 约束条件

- 回溯法的解需要满足一组综合的约束条件，通常分为：显式约束和隐式约束。
- 显式约束条件限定每个 $x_i$ 只从一个给定的集合上取值，例如：
  - $x_i \geq 0$       即 $s_i = \{\text{所有非负实数}\}$
  - $x_i = 0$ 或 $x_i = 1$     即 $s_i = \{0, 1\}$
  - $l \leq x_i \leq u$     即 $s_i = \{a: l \leq a \leq u\}$
- 满足显式约束的所有元组确定一个可能的解空间
- 隐式约束描述了 $x_i$ 必须彼此相关的情况，如0/1背包问题中的背包重量 $M$





# 6.1 一般方法

## ■ 8-皇后问题

- 在一个 $8 \times 8$ 棋盘上放置8个皇后，且使得每两个之间都不能互相“攻击”，也就是使得每两个都不能在**同一行、同一列及同一条斜角线**上。
- 给棋盘的行和列都编上1到8的行号。这些皇后也可给以1到8的编号。
- 由于一个皇后应在不同的行上，不失一般性，故可以假定皇后 $i$ 将放在行 $i$ 上。因此，8-皇后问题可以表示成8-元组 $(x_1, x_2, \dots, x_8)$ ，其中 $x_i$ 是放置第 $i$ 行皇后所在的列号。
- 使用这种表示的显示约束条件是 $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}$ ， $1 \leq i \leq 8$ 。



# 6.1 一般方法

## ■ 8-皇后问题

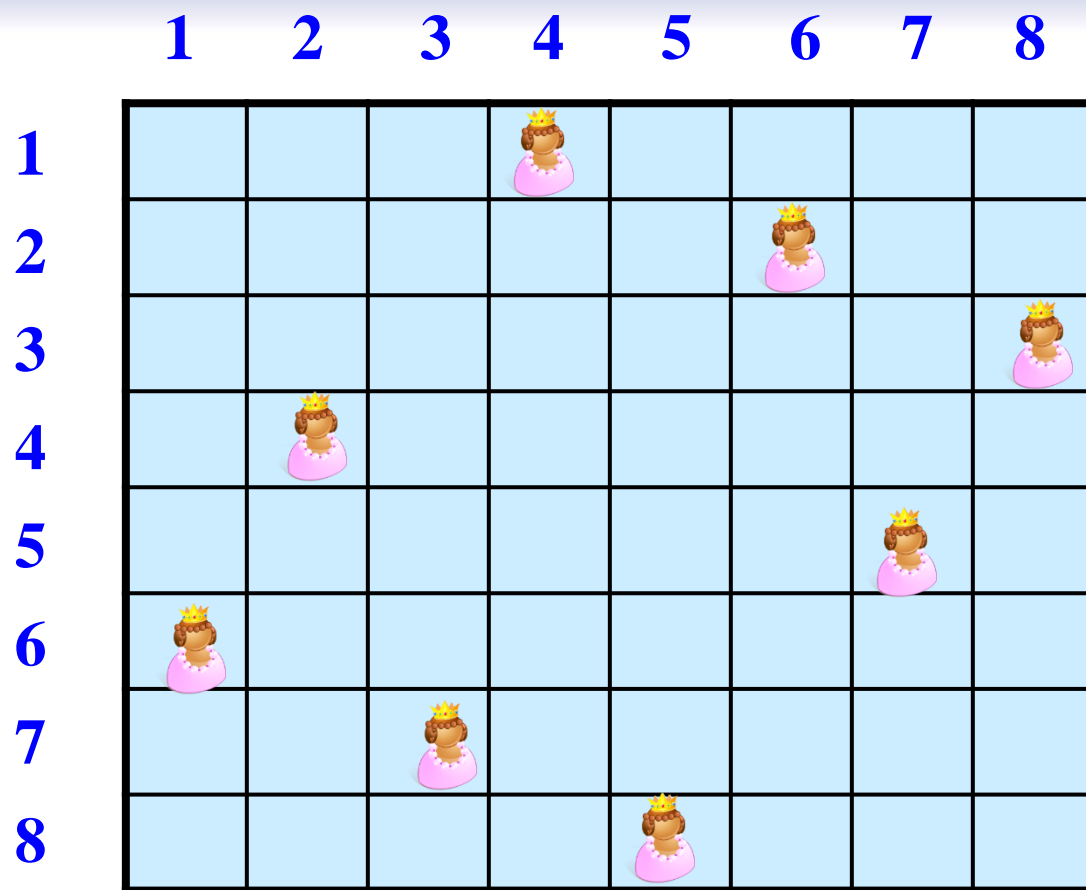
### □ 解空间:

- 所有可能的8元组，有 $8^8$ 个。

### □ 隐式约束条件

- 用来描述 $x_i$ 之间的关系，即没有两个 $x_i$ 可以相同且没有两个皇后可以在同一条斜角线上。
- 由隐式约束条件可知：可能解只能是(1, 2, 3, 4, 5, 6, 7, 8)的置换，最多有 $8!$ 个。





上图中的解表示为一个8-元组。即  
(4, 6, 8, 2, 7, 1, 3, 5)。



# 6.1 一般方法

## ■ 子集和数问题

- 已知 $n+1$ 个正数( $w_1, w_2, \dots, w_n$ )和 $M$ , 均为正数。
- 要求找出 $w_i$ 的和数等于 $M$ 的所有子集。
- 例如: 若 $n=4$ ,  $(w_1, w_2, w_3, w_4)=(11, 13, 24, 7)$ ,  $M=31$ , 则满足要求的子集:
  - 直接用元素表示:  $(11, 13, 7)$  和  $(24, 7)$
  - 用元素下标表示( $k$ -元组):  $(1, 2, 4)$  和  $(3, 4)$
  - 用元素下标表示( $n$ -元组):  $(1, 1, 0, 1)$  和  $(0, 0, 1, 1)$



# 6.1 一般方法

## ■ 子集和数问题

### □ 子集和数问题解的 $k$ -元组表示

- 用 $w_i$ 的下标表示解向量
- 子集和数问题的解可以表示为 $k$ -元组 $(x_1, x_2, \dots, x_k)$ ,  $1 \leq k \leq n$ 并且不同的解可以是大小不同的元组。
- 显式约束条件
  - ✓  $x_i \in \{j \mid j \text{ 为整数且 } 1 \leq j \leq n\}$ 。
- 隐式约束条件
  - ✓ 没有两个 $x_i$ 是相同的;
  - ✓  $w_{x_i}$ 的和为 $M$ ;
  - ✓  $x_i < x_{i+1}$ ,  $1 \leq i < n$  (避免产生同一个子集的重复情况)



# 6.1 一般方法

## ■ 子集和数问题

### □ 子集和数问题解的 $n$ -元组表示

- 解由 $n$ -元组 $(x_1, x_2, \dots, x_n)$ 表示; 其中 $x_i \in \{0, 1\}$ 。如果选择了 $w_i$ , 则 $x_i=1$ , 否则 $x_i=0$ 。
- 显式约束条件
  - ✓  $x_i \in \{0, 1\}$ ,  $1 \leq i \leq n$ , 如果没有选择 $w_i$ , 则 $x_i=0$ ; 如果选择了 $w_i$ , 则 $x_i=1$ 。
  - ✓ 于是上面的解可以表示为 $(1, 1, 0, 1)$ 和 $(0, 0, 1, 1)$ ;
- 隐式约束条件:
  - ✓  $\sum (x_i \times w_i) = M$
- 特点: 所有元组具有统一固定的大小。
- 解空间的大小为 $2^n$ 个元组



# 6.1 一般方法

## ■ 解空间的树结构

- 回溯算法通过系统地检索给定问题的解空间来确定问题的解。
- 这需要有有效的组织问题的解空间
  - 把元组表示成为有结构的组织方式。
- 采用何种形式组织问题的解空间？
  - 可以用**树结构**组织解空间——**状态空间树**。
- **n-皇后问题**
  - 8皇后问题的推广，即在 $n \times n$ 的棋盘上放置 $n$ 个皇后，使得它们不会相互攻击。
  - 解空间由 $n$ -元组 $(1, 2, \dots, n)$ 的 $n!$ 中排列所组成。



# 6.1 一般方法

## ■ 解空间的树结构

□实例：4皇后问题的解空间树结构如下所示：

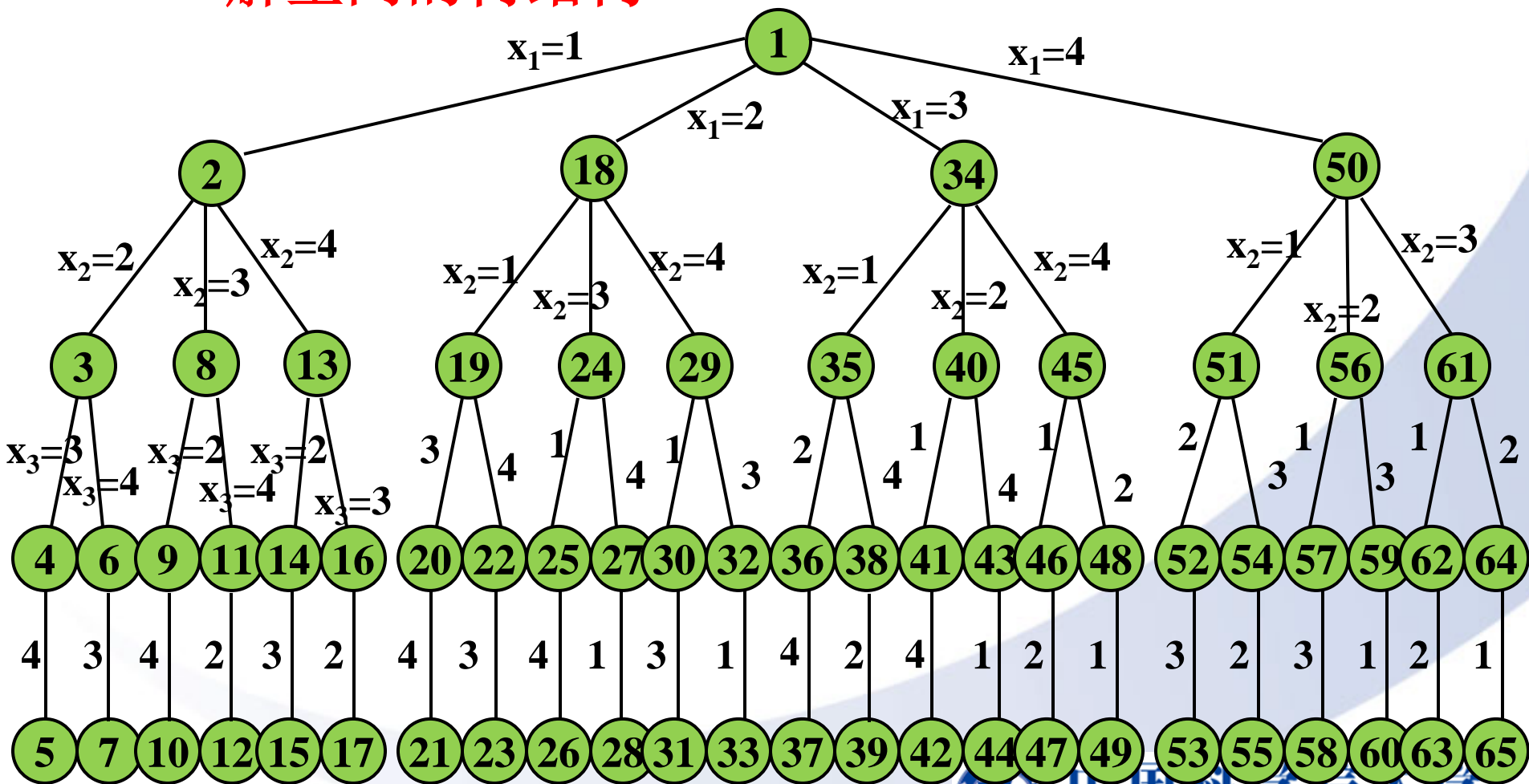
- 从 $i$ 级到 $i+1$ 级的边用 $x_i$ 的值标记，表示将皇后 $i$ 放到第 $i$ 行的第 $x_i$ 列。
- 由1级到2级结点的边给出 $x_1$ 的各种取值：1、2、3、4。
- 最左子树包含 $x_1=1$ 的所有解；这棵子树的最左子树则包含着 $x_1=1$ 且 $x_2=2$ 的所有解，等等。
- 解空间：由从根结点到叶结点的所有路径所定义。
- 注：共有 $4! = 24$ 个叶结点，反映了4元组的所有可能排列。——称为排列树。





# 6.1 一般方法

## ■ 解空间的树结构



# 6.1 一般方法

## ■ 解空间的树结构

□ 子集和数问题的解空间的树结构

□ 两种元组表示形式:

① 元组大小可变( $x_i < x_{i+1}$ )

➤ 树边的标记方法: 由*i*级结点到*i*+1级结点的一条边用 $x_i$ 来表示,

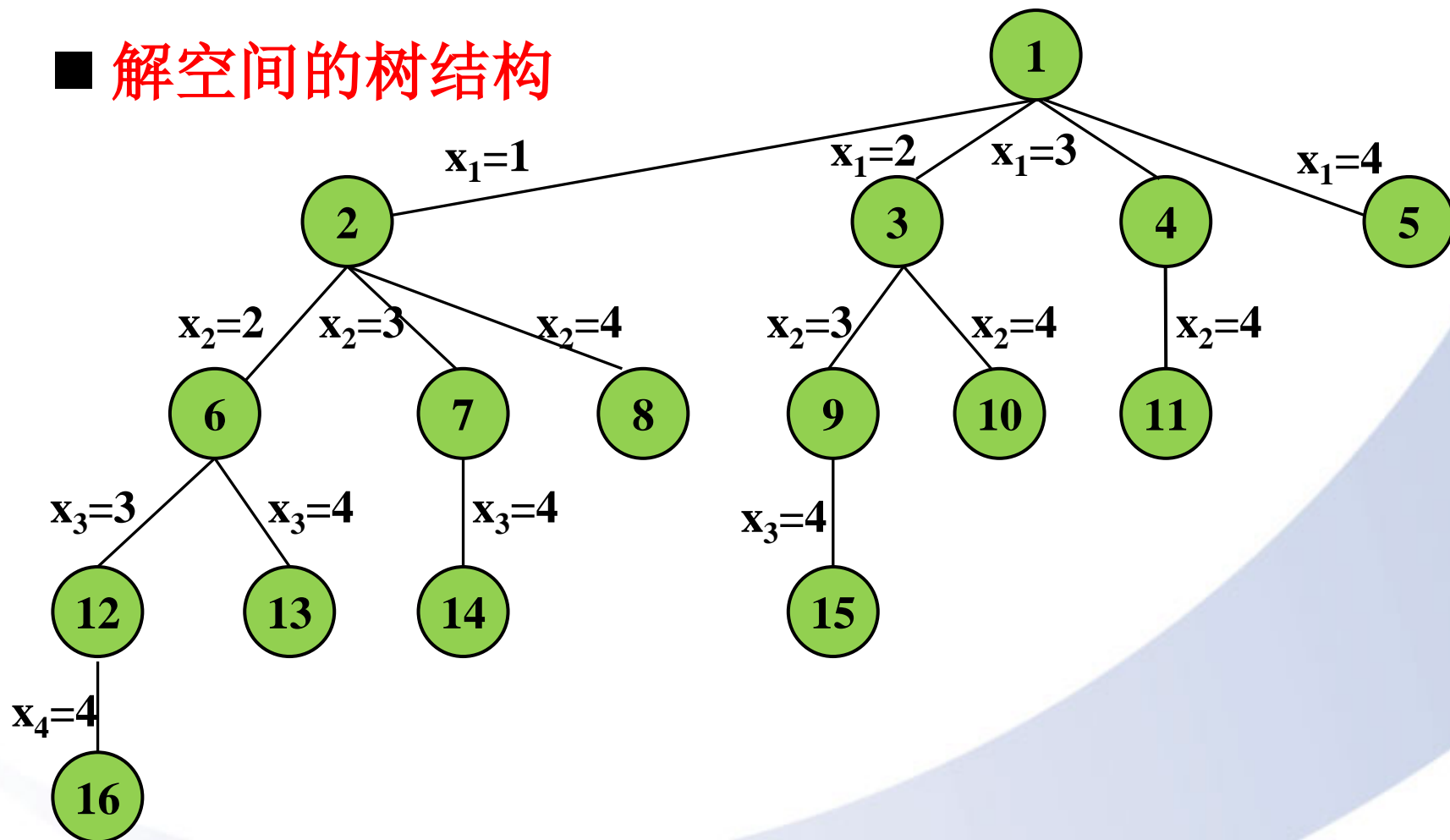
➤ 在每一个结点处, 解空间被分成一些子解空间, 解空间则由树中的根结点到**任何结点**的所有路径所确定, 这些可能的路径是(), (1), (1, 2), (1, 2, 3), (1, 2, 3, 4), (1, 2, 4), (1, 3, 4), (1, 4), (2), (2, 3)等等。

➤ 左子树确定了包含 $w_1$ 的所有子集, 下一棵子树则确定了包含 $w_2$ 但不包含 $w_1$ 的所有子集。



# 6.1 一般方法

## ■ 解空间的树结构



# 6.1 一般方法

## ■ 解空间的树结构

□ 子集和数问题的解空间的树结构

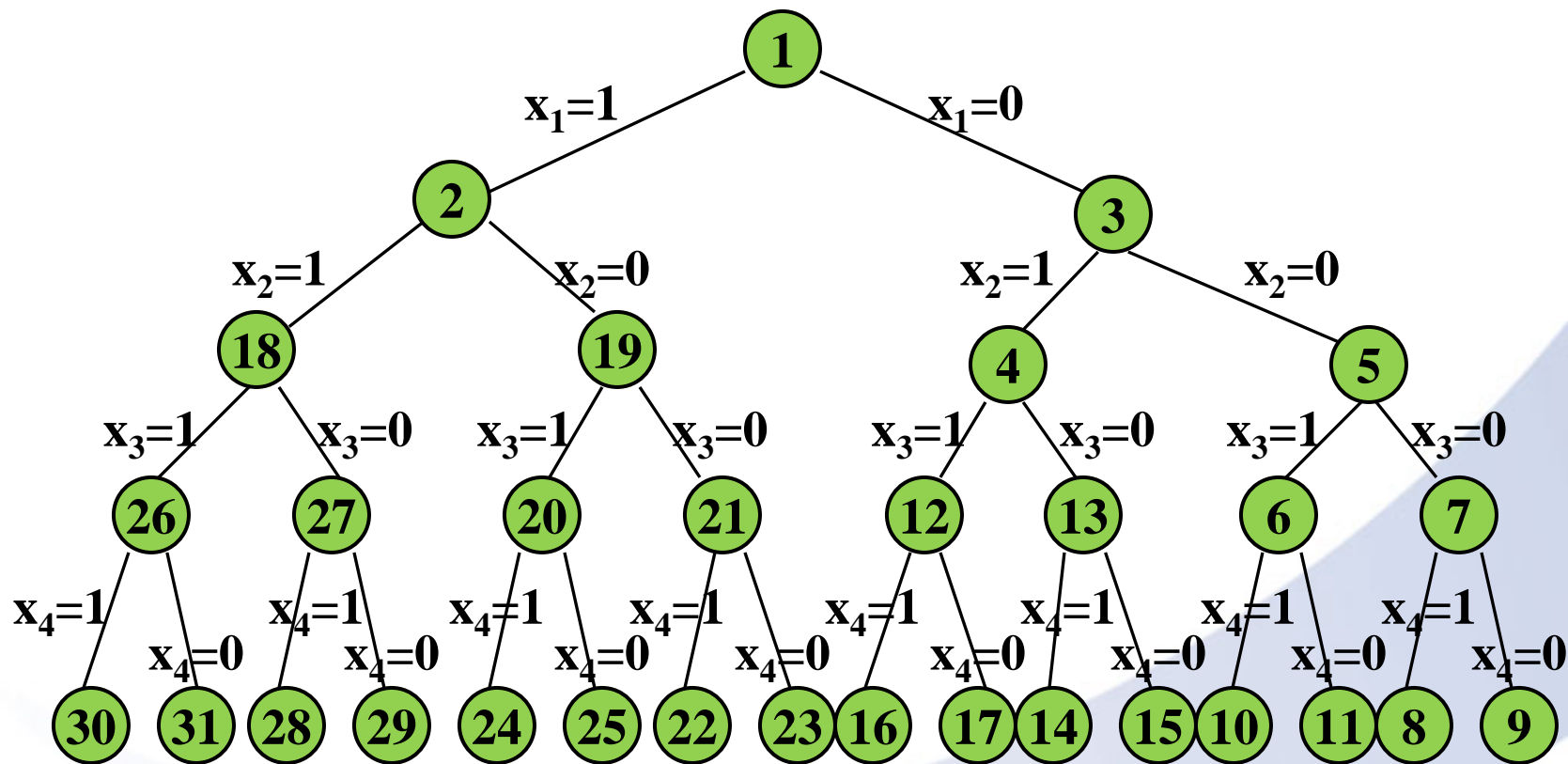
② 元组大小固定，每个都是 $n$ -元组

- 由 $i$ 级结点到 $i+1$ 级结点的那些边用 $x_i$ 的值来标记， $x_i$ 的值或者为1或者为0。
- 由根到叶结点的所有路径确定了解空间，
- 根的左子树确定包含 $w_1$ 的所有子集，而根的右子树则确定不包含 $w_1$ 的所有子集等等。
- 于是有 $2^4$ 个叶结点，表示16个可能的元组。



# 6.1 一般方法

## ■ 解空间的树结构



# 6.1 一般方法

## ■ 解空间的树结构

- 对于任何一个问题，一旦设想出一种状态空间树，
- 那么就可以先系统地生成问题状态，
- 接着确定这些问题状态中的哪些是解状态，
- 最后确定哪些解状态是答案状态从而将问题解出
- 同一个问题的状态空间树可以有不同的形式。(其中某些表示方法更简单，所需表示的状态空间更小，搜索方法更简单！)



# 6.1 一般方法

## ■ 解空间的树结构

□ 为了便于讨论，引进一些关于解空间树结构的术语。

□ 状态空间树(state space tree):

➤ 解空间的树结构称为状态空间树。

□ 状态空间(state space):

➤ 由根结点到其它节点的所有路径则确定了这个问题的状态空间。

□ 问题状态(problem state):

➤ 树中的每一个结点确定所求解问题的一个问题状态。



# 6.1 一般方法

## ■ 解空间的树结构

### □ 解状态(solution states):

- 解状态是这样一些问题状态 $S$ ，对于这些问题状态，由根到 $S$ 的那条路径确定了这解空间中的一个元组。

### □ 答案状态(answer states):

- 答案状态是这样的一些解状态 $S$ ，由根到 $S$ 的这条路径确定了这问题的一个解(即它满足隐式约束条件)。

### □ 状态空间树的分解:

- 在状态空间树的每个结点处，解空间被分解为一些子解空间，表示在一些分量取特定值情况下的解空间元素。

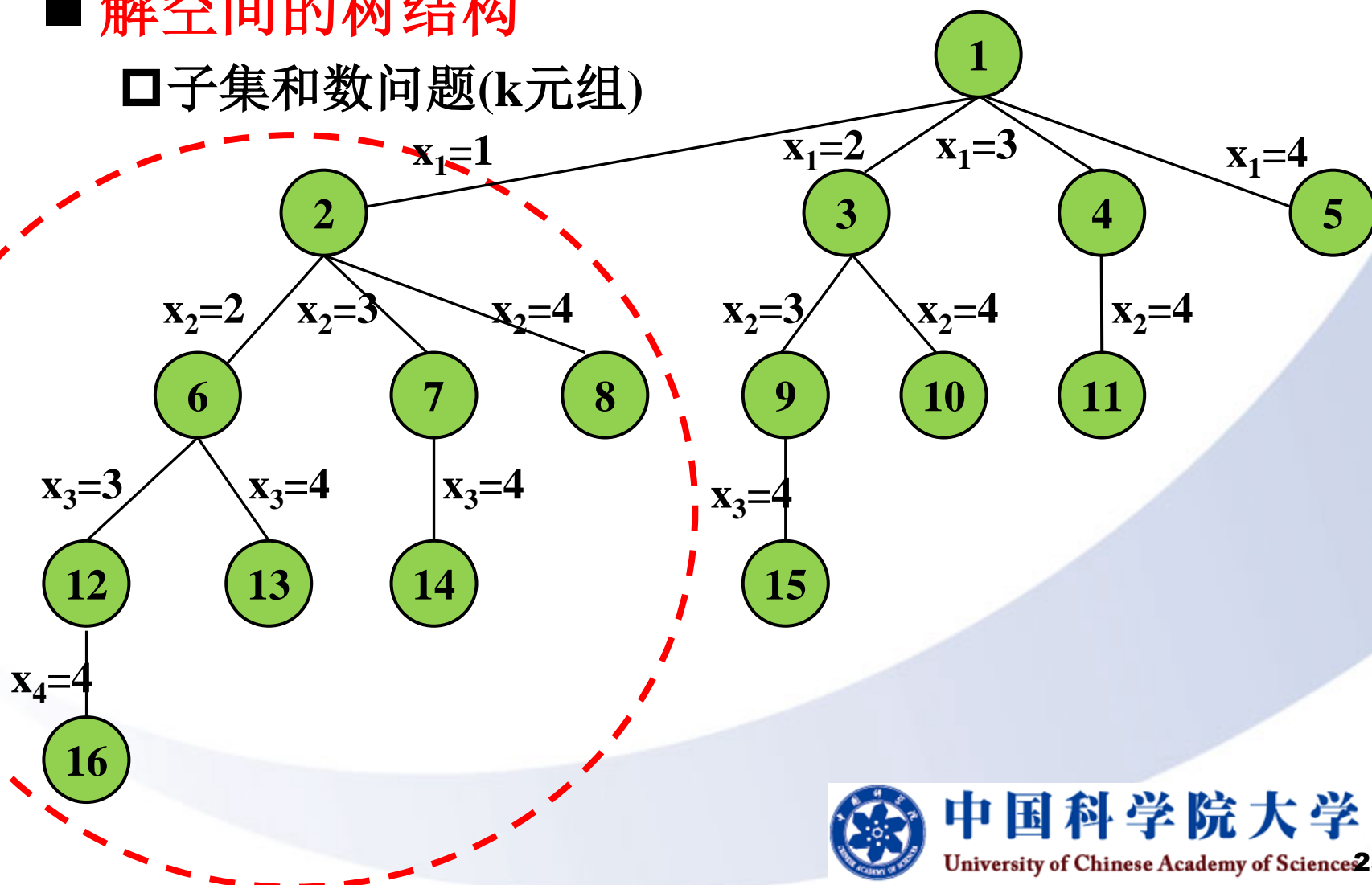




# 6.1 一般方法

## ■ 解空间的树结构

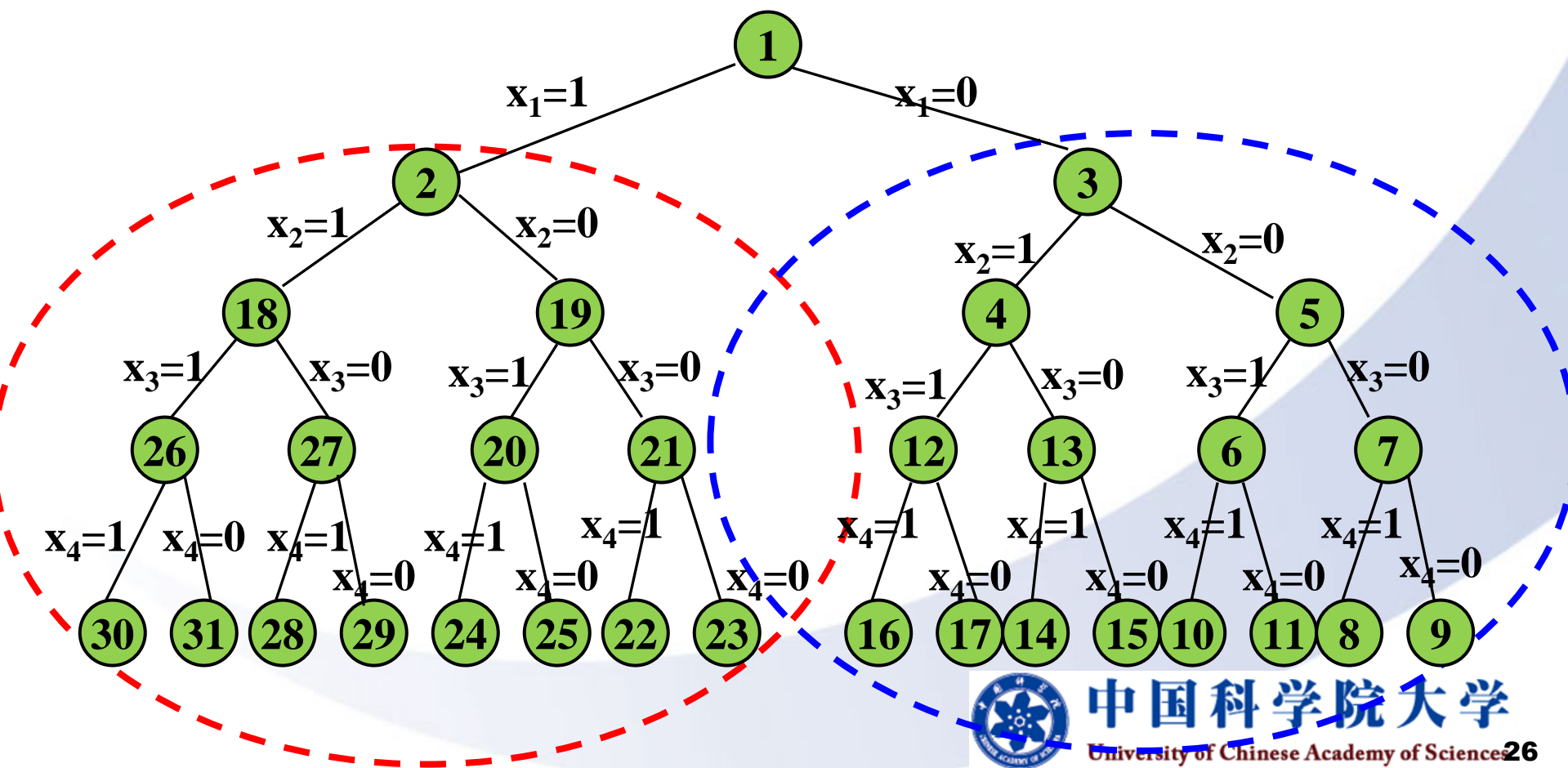
□ 子集和数问题(k元组)



# 6.1 一般方法

## ■ 解空间的树结构

□ 子集和数问题(n元组)



# 6.1 一般方法

## ■ 状态空间树

### □ 求解问题的一般步骤：

- 对任何一个问题，一旦设想出一种状态空间树，
- 第一步、先系统的生成问题状态；
- 第二步、确定这些问题状态中哪些是解状态；
- 第三步、确定哪些解状态是答案状态。

### □ 状态空间树的构造：

- 以问题的初始状态作为根结点，然后系统地生成其它问题状态的结点。



# 6.1 一般方法

## ■ 状态空间树

□ 在状态空间树生成的过程中，结点根据被检测情况分为三类：

- **活结点**：自己已经生成而其所有的**儿子结点还没有全部生成**的结点。
- **E-结点(正在扩展的结点)**：当前**正在生成其儿子结点**的活结点。
- **死结点**：不再进一步扩展或者其**儿子结点已全部生成**的结点。



# 6.1 一般方法

## ■ 状态空间树

### □ 生成问题状态的两种方法

#### ➤ 深度优先策略

- ✓ 当前的E-结点R一旦生成一个新的儿子C，这个儿子结点就变成一个新的E-结点，当完全检测了子树C之后，R结点就再次成为E-结点
- ✓ 相当于问题状态的深度优先生成。

#### ➤ 宽度优先策略

- ✓ 一个E-结点一直保持到变成死结点为止。



# 6.1 一般方法

## ■ 状态空间树

### □ 生成问题状态的两种方法

#### ➤ 限界函数:

- ✓ 在结点的生成过程中，定义一个**限界函数**，用来杀死还没有全部生成儿子结点的一些活结点
- ✓ 这些活结点无法满足限界函数的条件，不可能导致问题的答案。

#### ➤ 回溯法:

- ✓ 使用限界函数的深度优先结点生成方法。

#### ➤ 分枝-限界方法:

- ✓ E-结点一直保持到死为止的状态生成方法。



# 6.1 一般方法

## ■ 状态空间树

□ 算法搜索至任一结点时，先判断以该结点为根的子树是否包含问题的解。如果肯定不包含，则跳过对该结点为根的子树的搜索，逐层向其祖先结点回溯；否则，进入该子树，继续按深度优先策略搜索。

□ 常用的剪枝函数：

- 用约束函数剪去已知不含答案状态(可行解)的子树；
- 用限界函数剪去得不到最优答案结点(最优解)的子树。

□ 回溯法与穷举搜索不同：

- 回溯法使用限界函数，剪去那些可以断定不含答案状态的子树，从而提高算法效率。
- 回溯法适用于解一些组合数相当大的问题。

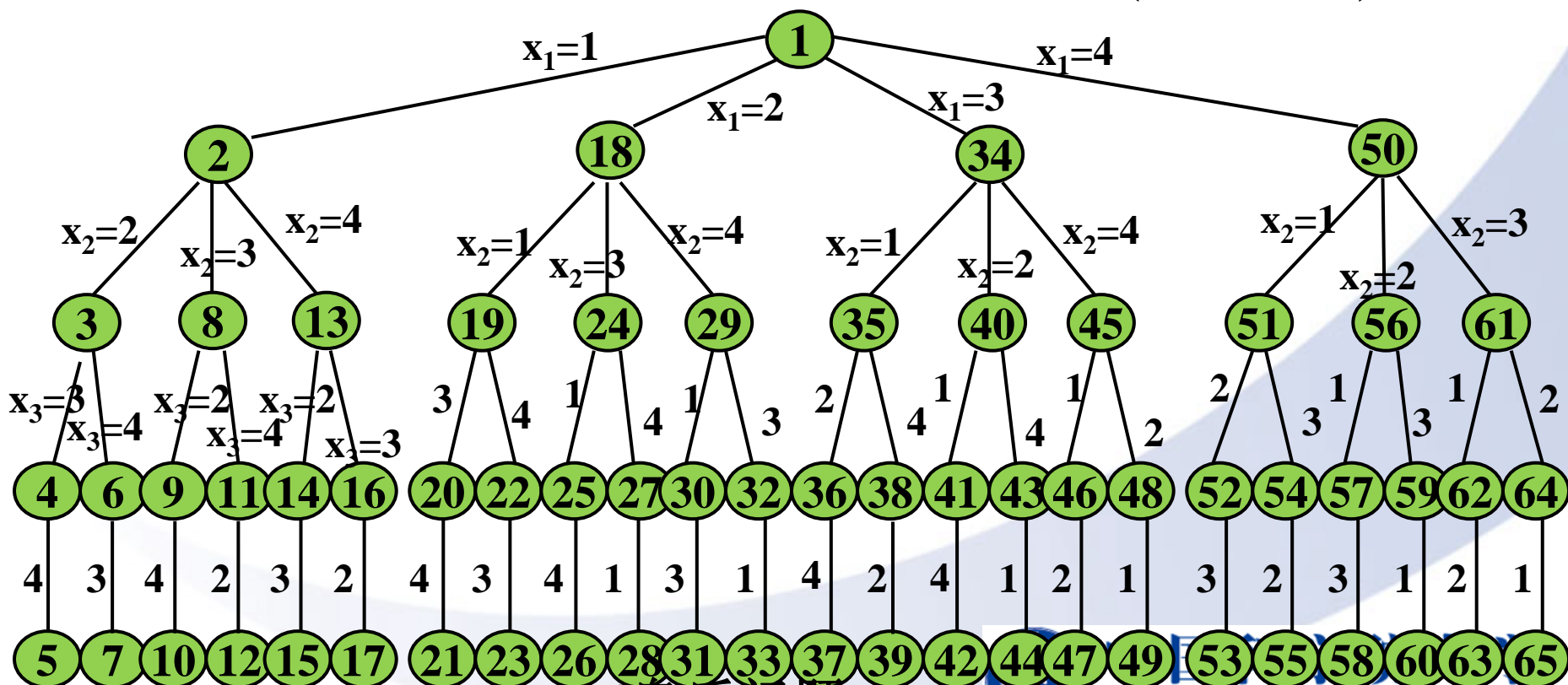


# 6.1 一般方法

## ■ 状态空间树

### □ 生成问题状态的两种方法

➤ 深度优先策略下的结点生成次序(结点编号)



4皇后问题

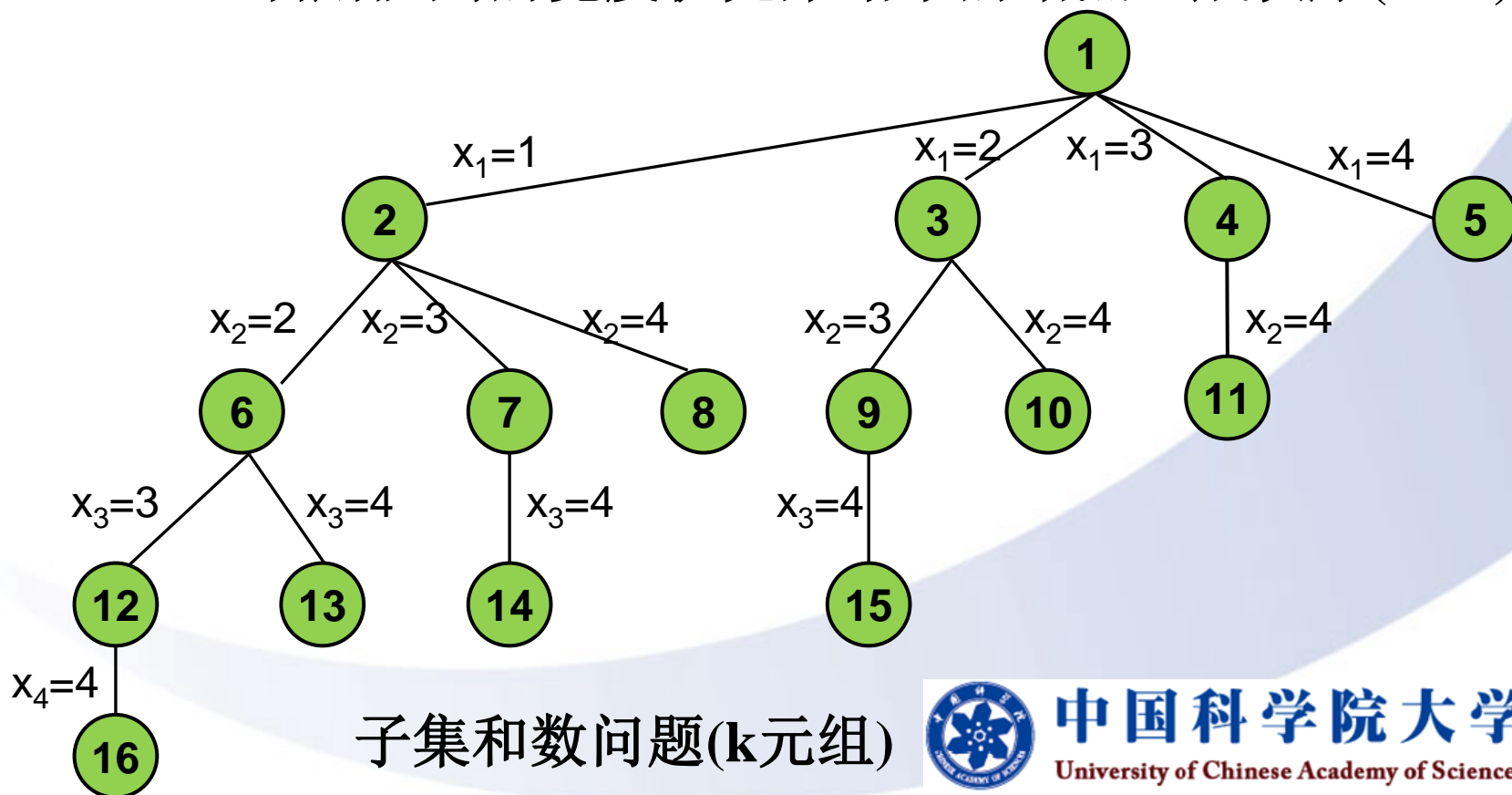


# 6.1 一般方法

## ■ 状态空间树

### □ 生成问题状态的两种方法

➤ 利用队列的宽度优先策略下的结点生成次序(BFS)



子集和数问题(k元组)



中国科学院大学

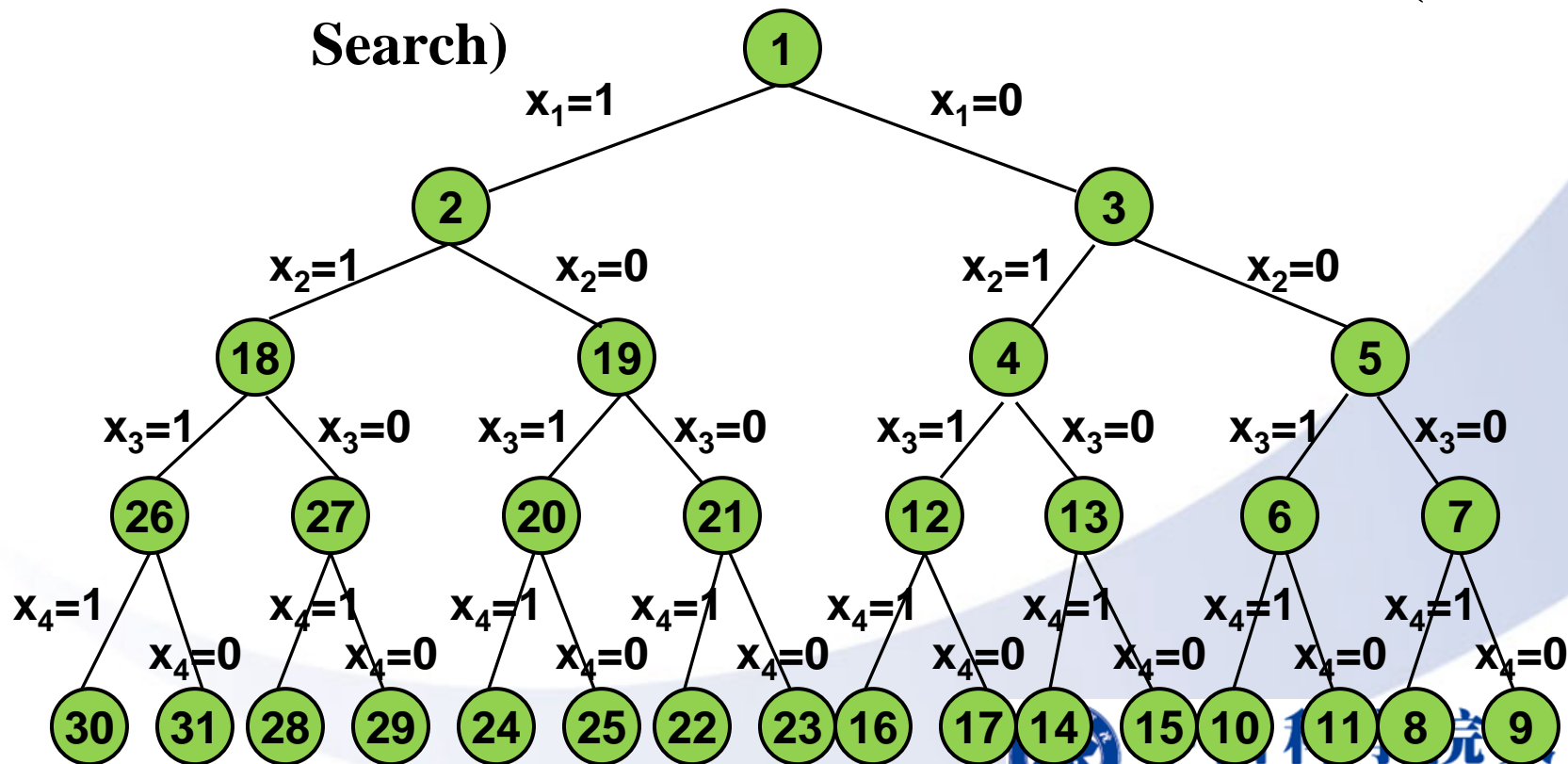
University of Chinese Academy of Sciences 33

# 6.1 一般方法

## ■ 状态空间树

### □ 生成问题状态的两种方法

➤ 利用栈的宽度优先策略下的结点生成次序(D-Search)



子集和数问题(n元组)

# 6.1 一般方法

## ■ 例：4-皇后问题的回溯法求解

### □ 限界函数：

- 如果 $(x_1, x_2, \dots, x_i)$ 是到当前E结点的路径，那么 $x_i$ 的儿子结点 $x_{i+1}$ 是一些这样的结点，它们使得 $(x_1, x_2, \dots, x_i, x_{i+1})$ 表示没有两个皇后正在相互攻击的一种棋盘格局。

### □ 开始状态：

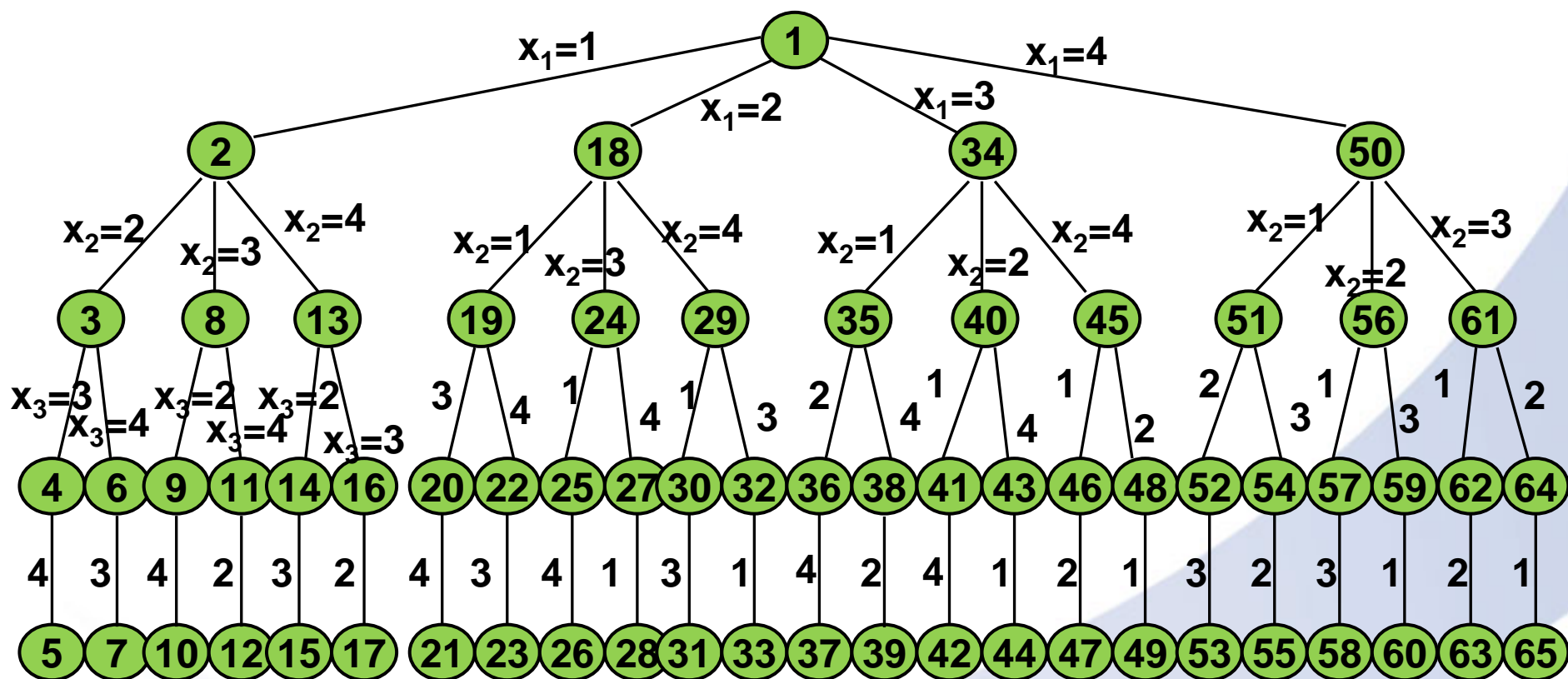
- 根结点1，表示还没有放置任何皇后。

### □ 结点的生成：

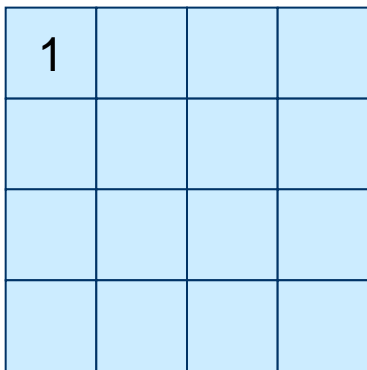
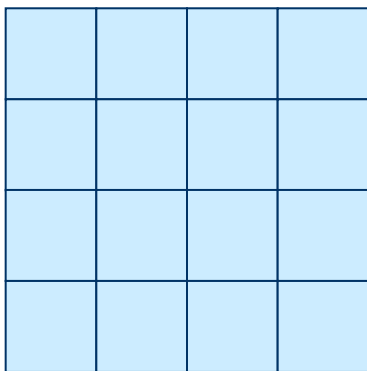
- 依次考察皇后1——皇后n的位置。



# 6.1 一般方法

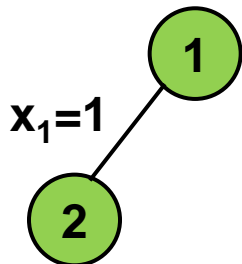


按照自然数递增的次序生成儿子结点。



根结点1，开始状态，唯一的活结点

解向量: ()



生成结点2，表示皇后1被放到第1行的第1列上，该结点是从根结点开始第一个被生成结点。

解向量: (1)

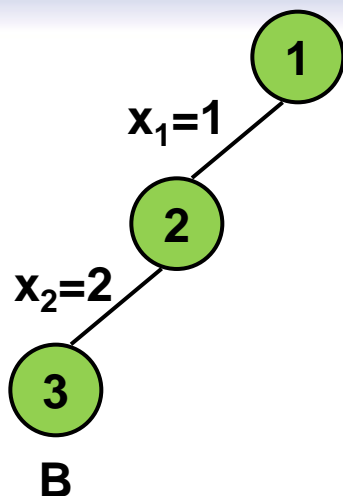
结点2变成新的E结点，下一步扩展结点2



中国科学院大学

University of Chinese Academy of Sciences 37

1			
.	2		



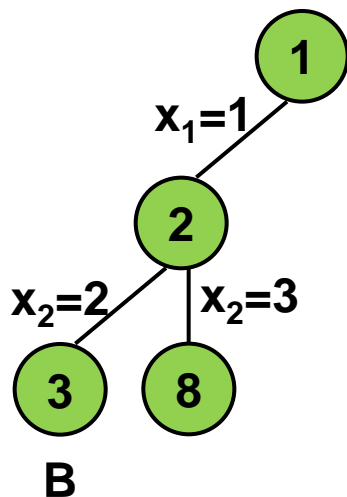
由结点2生成结点3，即皇后2放到第2行第2列。

利用限界函数杀死结点3。

返回结点2继续扩展。

(结点4, 5, 6, 7不会生成)

1			
.	.	2	



由结点2生成结点8，即皇后2放到第2行第3列。

结点8变成新的E结点。

解向量: (1, 3)

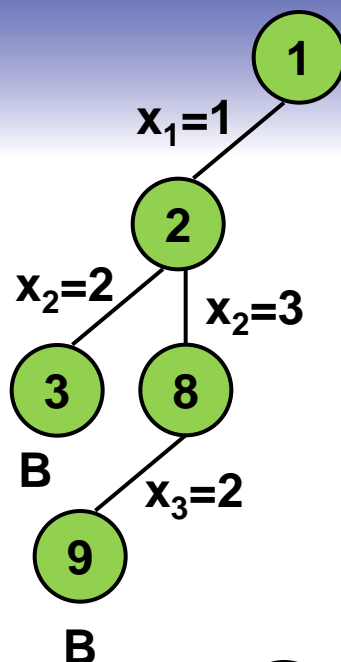
从结点8继续扩展。



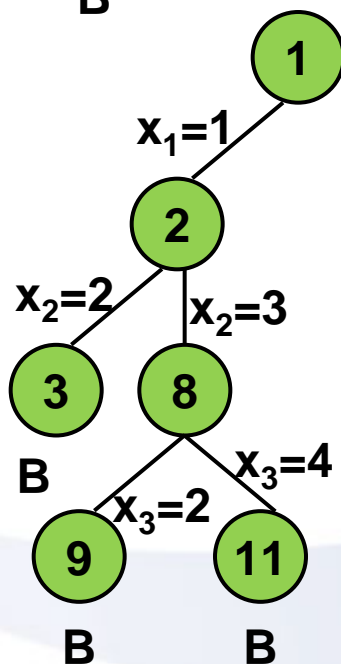
中国科学院大学

University of Chinese Academy of Sciences 38

1			
.	.	2	
	3		



1			
.	.	2	
.	.	.	3



由结点8生成结点9，即皇后3放到第3行第2列。

利用限界函数杀死结点9。

返回结点8继续扩展。

(结点10不会生成)

由结点8生成结点11，即皇后3放到第3行第4列。

利用限界函数杀死结点11。

返回结点8继续。

(结点12不会生成)

结点8的所有儿子已经生成，但没有导出答案结点，变成死结点。结点8被杀死。

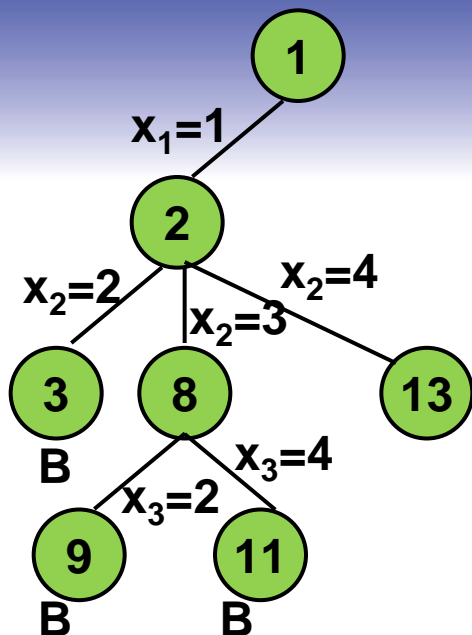
返回结点2继续扩展。



中国科学院大学

University of Chinese Academy of Sciences 39

1			
.	.	.	2



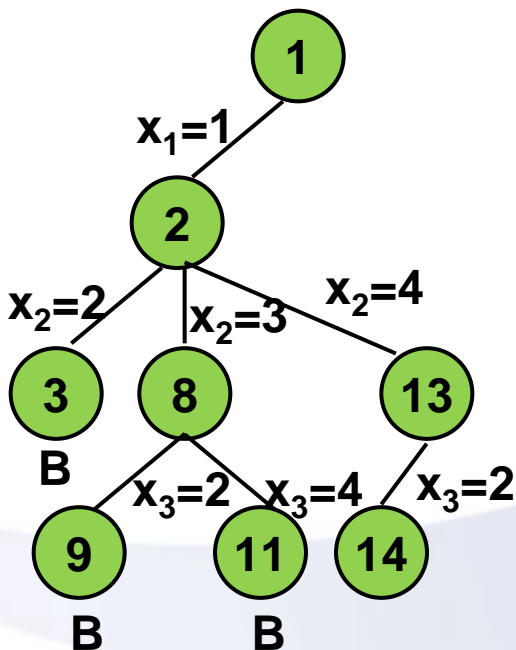
由结点2生成结点13，即皇后2放到第2行第4列。

结点13变成新的E结点。

解向量：(1, 4)

从结点13继续扩展。

1			
.	.	.	2
.	3		



由结点13生成结点14，即皇后3放到第3行第2列。

结点14变成新的E结点。

解向量：(1, 4, 2)

从结点14继续扩展。

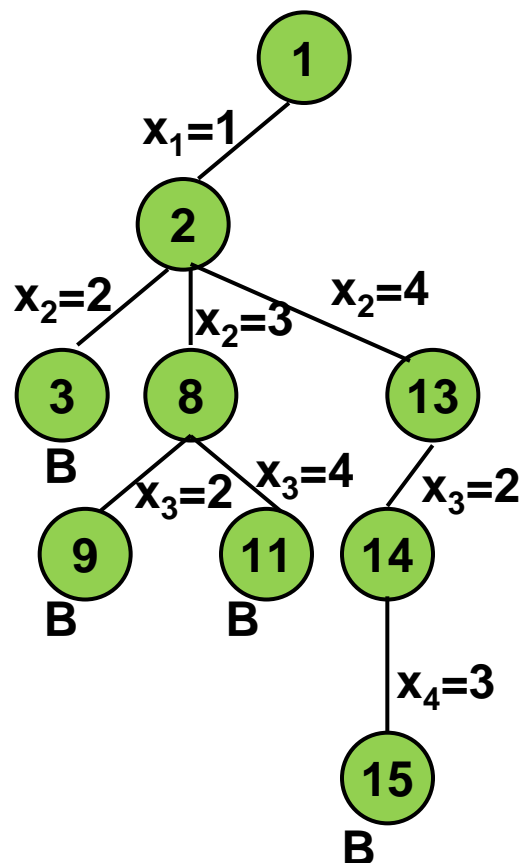


中国科学院大学

University of Chinese Academy of Science 40



1			
.	.	.	2
.	3		
.	.	4	



由结点14生成结点15，即皇后4放到第4行第3列。

利用限界函数杀死结点15。

返回结点14，结点14不能导致答案结点，变成死结点，被杀死。

返回结点13继续扩展。



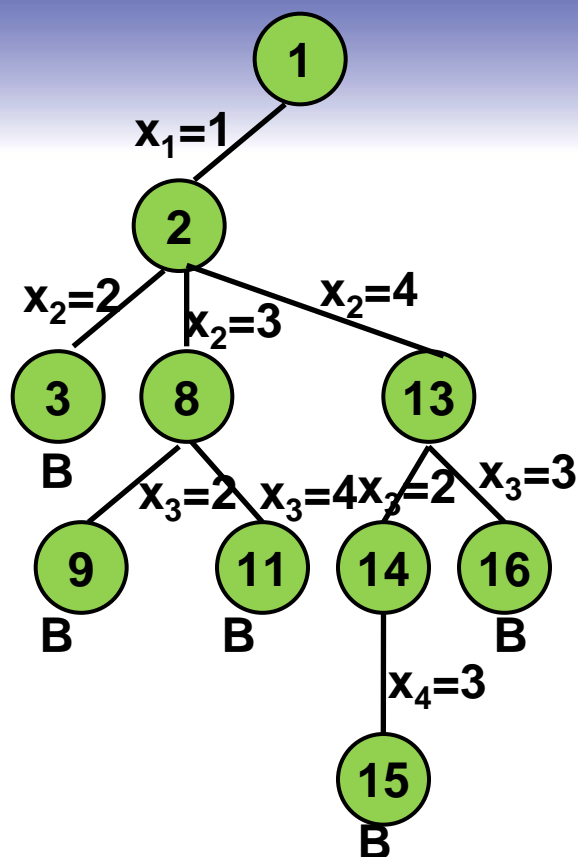
中国科学院大学

University of Chinese Academy of Science 41

1			
.	.	.	2
.	.	3	



.	1		



由结点13生成结点16，即皇后3放到第3行第3列。

利用限界函数杀死结点16。

返回结点13，结点13不能导致答案结点，变成死结点，被杀死。

返回结点2继续扩展。

结点2不能导致答案结点，变成死结点，被杀死。

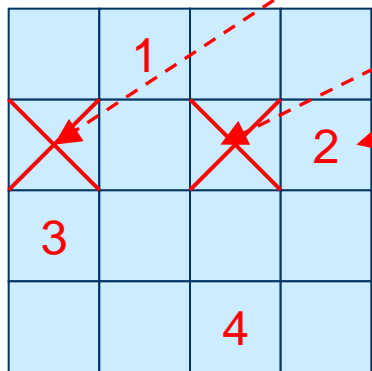
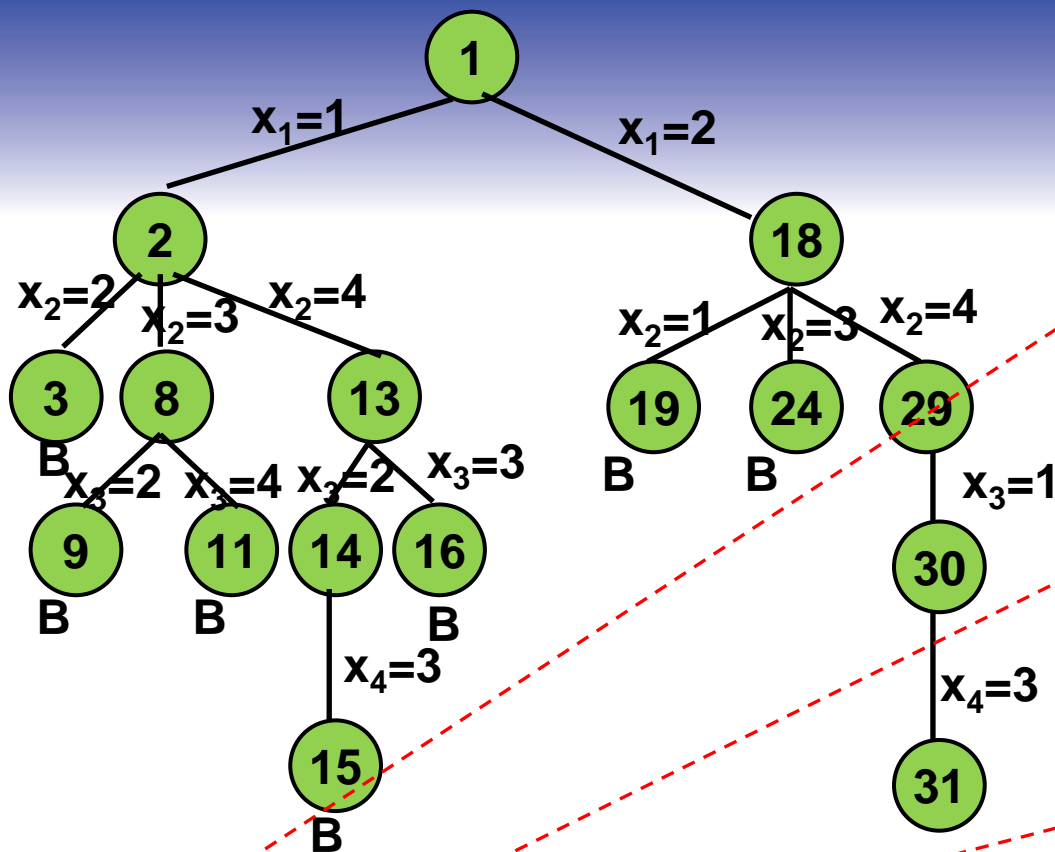
返回结点1继续扩展。

由结点1生成结点18，即皇后1放到第1行第2列。



中国科学院大学

University of Chinese Academy of Science 42



结点31是答案结点。

解向量: (2, 4, 1, 3)

算法终止(找到了一个解)。

答案  
结点

由结点1生成结点18, 即皇后1放到第1行第2列。结点18变成E结点。

扩展结点18生成结点19, 即皇后2放到第2行第1列。

利用限界函数杀死结点19。

返回结点18, 生成结点24, 即皇后2放到第2行第3列。

利用限界函数杀死结点24。

返回结点18, 生成结点29, 即皇后2放到第2行第4列。结点29变成E结点。

扩展结点29生成结点30, 即皇后3放到第3行第1列。结点30变成E结点。

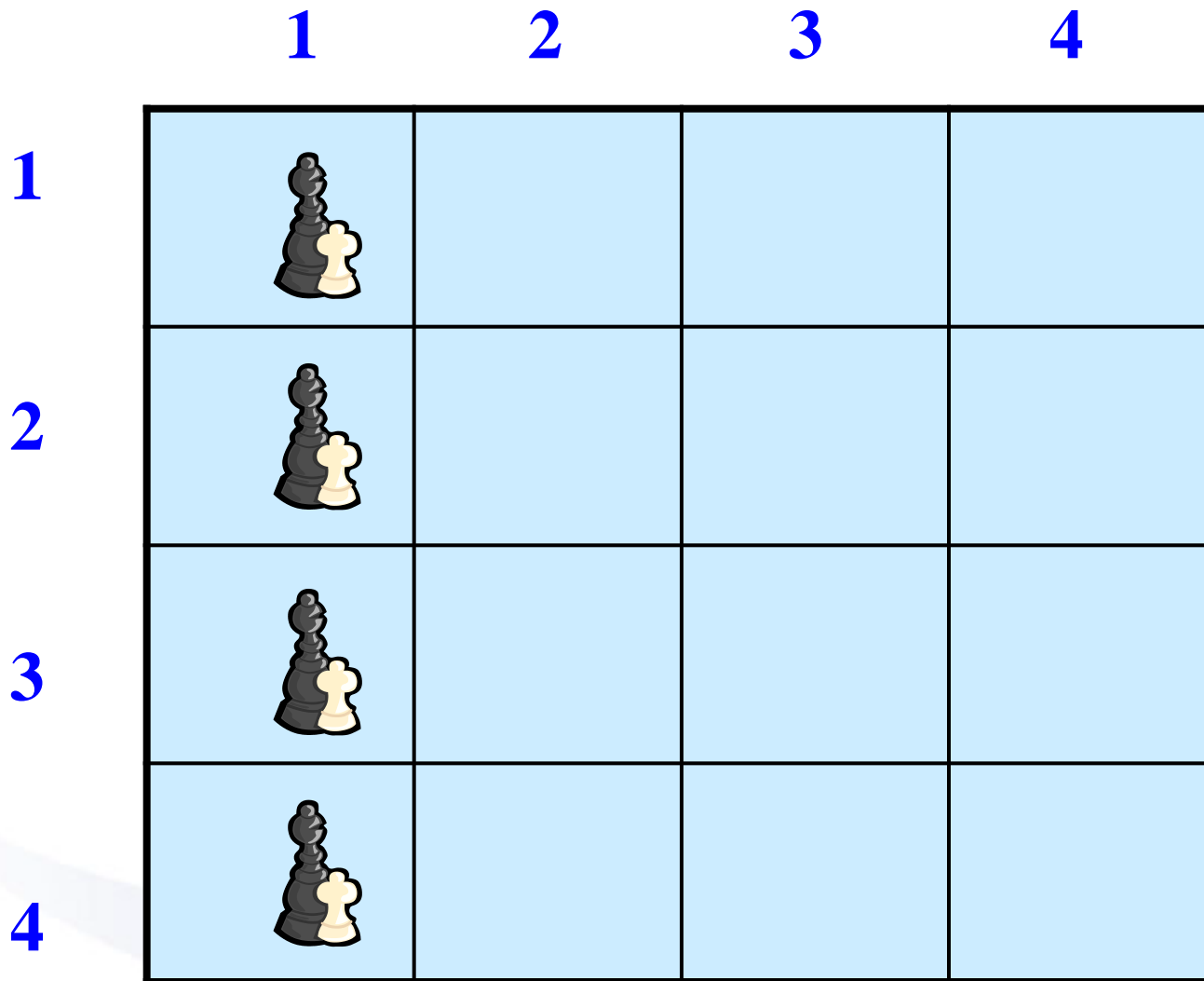
扩展结点30生成结点31, 即皇后4放到第4行第3列。



中国科学院大学

University of Chinese Academy of Science 43

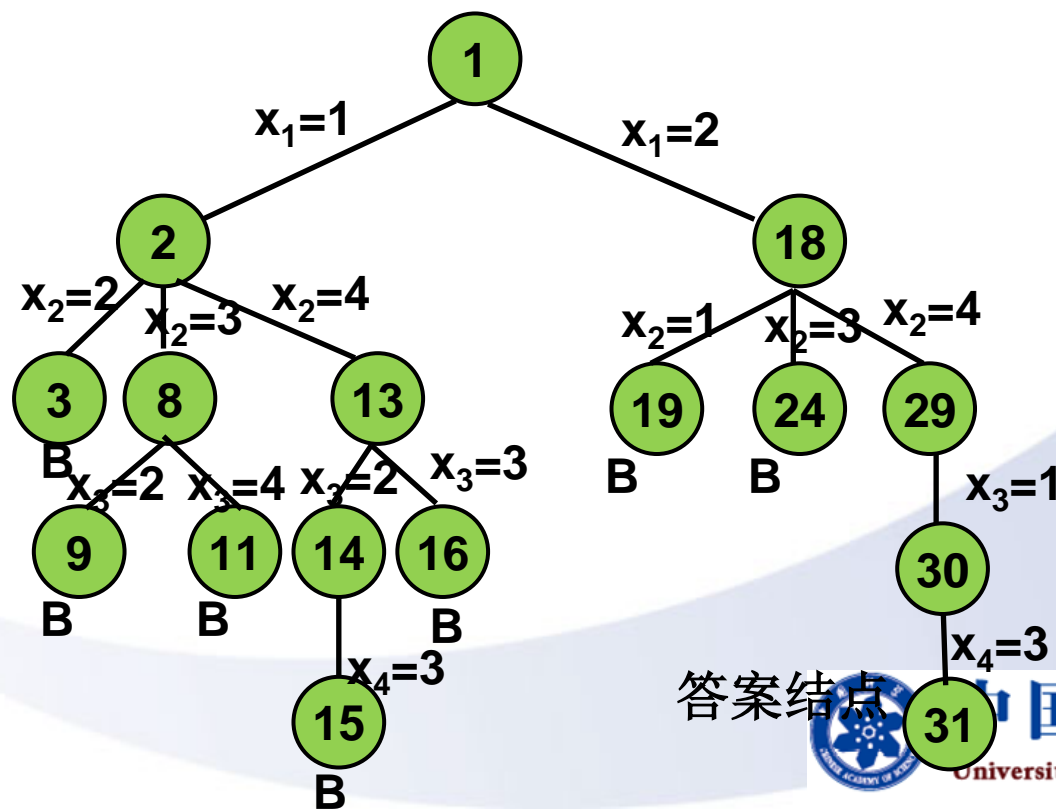
## 4-皇后问题-回溯解



## 6.1 一般方法

### ■ 4-皇后问题回溯期间生成的树

□ 下图显示了在4-皇后问题中求上述一个解的树的实际生成部分。结点按它们生成的次序被编号。由限界函数所杀死的结点则在其下方写上B。



答案结点



中国科学院大学

University of Chinese Academy of Science 45

# 6.1 一般方法

## ■ 回溯法思想

- 第一步：为问题定义一个状态空间(state space)，这个空间必须至少包含问题的一个解
- 第二步：组织状态空间以便它容易被容易地搜索。典型的组织方法是图或树
- 第三步：按深度优先的方法从开始节点进行搜索
  - 开始节点是一个活节点(也是E-节点)
  - 如果能从当前的E-节点移动到一个新节点，那么这个新节点将变成一个活节点和新的E-节点，旧的E-节点仍是一个活节点。



# 6.1 一般方法

## ■ 回溯法思想

□ 第三步：按深度优先的方法从开始节点进行搜索

- 如果不能移到一个新节点，当前的E-节点就“死”了(即不再是一个活节点)，那么便只能返回到最近被考察的活节点(回溯)，这个活节点变成了当前的E-节点。
- 当我们已经找到了答案或者回溯尽了所有的活节点时，搜索过程结束。



# 6.1 一般方法

## ■ 回溯算法的形式描述

假设回溯算法要找出所有的答案结点而不是仅仅只找出一个。

- ① 设 $(x_1, x_2, \dots, x_{i-1})$ 是状态空间树中由根到一个结点(问题状态)的路径。
- ②  $T(x_1, x_2, \dots, x_{i-1})$ 是下述所有结点的 $x_i$ 的集合, 它使得对于每一个 $x_i$ ,  $(x_1, x_2, \dots, x_i)$ 是一条由根到结点 $x_i$ 的路径。
- ③ 存在一些限界函数 $B_i$ (可以表示成一些谓词), 如果路径 $(x_1, x_2, \dots, x_i)$ 不可能延伸到一个答案结点, 则 $B_i(x_1, x_2, \dots, x_i)$ 取假值, 否则取真值。

因此, 解向量 $X(1:n)$ 中的第 $i$ 个分量就是那些选自集合 $T(x_1, x_2, \dots, x_{i-1})$ 且使 $B_i$ 为真的 $x_i$ 。





# 6.1 一般方法

## ■ 回溯算法

**procedure** BACKTRACK(n)

**integer** k, n; **local** X(1:n)

k ← 1

**while** k > 0 **do**

**if** 还剩有没检验过的X(k)使得

$X(k) \in T(X(1), \dots, X(k-1))$  **and**  $B(X(1), \dots, X(k)) = \text{true}$

**then**

$(X(1), \dots, X(k))$  是一条已抵达一答案结点的路径

**then print**(X(1), ..., X(k)) **endif**

k ← k+1

**else**

k ← k-1 **endif**

**repeat**

**end** BACKTRACK

回溯方法的抽象描述。该算法求出所有答案结点。

在 $X(1), \dots, X(k-1)$ 已经被选定的情况下,  $T(X(1), \dots, X(k-1))$ 给出 $X(k)$ 的所有可能的取值。限界函数 $B(X(1), \dots, X(k))$ 判断哪些元素 $X(k)$ 满足隐式约束条件。



中国科学院大学

University of Chinese Academy of Science 49

# 6.1 一般方法

## ■ 回溯算法的递归表示

**procedure** RBACKTRACK(k)

**global** n, X(1:n)

**for** 满足下式的每个X(k)

$X(k) \in T(X(1), \dots, X(k-1))$  **and**  $B(X(1), \dots, X(k)) = \text{true}$  **do**

**if**(X(1), ..., X(k)) 是一条已抵达一答案结点的路径

**then print**(X(1), ..., X(k))

**endif**

**call** RBACKTRACK(k+1)

**repeat**

**end** RBACKTRACK

回溯方法的递归程序描述。

调用: RBACKTRACK(1)。

进入算法时, 解向量的前k-1个分量X(1), ..., X(k-1)已赋值。

说明: 当 $k > n$ 时,  $T(X(1), \dots, X(k-1))$ 返回一个空集, 算法不再进入for循环。

算法印出所有的解, 元组大小可变。

□ **算法说明:** 基本上是一棵树的后根次序周游。这个递归模型最初由call RABCKTRACK(1)调用。



中国科学院大学

University of Chinese Academy of Sciences 50

# 6.1 一般方法

## ■ 效率分析应考虑的因素

- ① 生成下一个 $X(k)$ 的时间
- ② 满足显式约束条件的 $X(k)$ 的数目
- ③ 限界函数 $B_i$ 的计算时间
- ④ 对于所有的 $i$ , 满足 $B_i$ 的 $X(k)$ 的数目

## □ 权衡:

- 限界函数生成结点数和限界函数本身所需的计算时间
- 好的界限函数可以大量减少所生成的结点数, 但是往往计算量较大。



# 6.1 一般方法

## ■ 效率分析

- 效率分析中应考虑的因素

  - (1) - (3) 与实例无关

  - (4) 与实例相关

- 有可能只生成  $O(n)$  个结点，有可能生成几乎全部结点

- 如果解空间的结点数是  $2^n$  或者  $n!$ ，最坏情况时间

  - $O(p(n)2^n)$ ， $p(n)$  为  $n$  的多项式

  - $O(q(n)n!)$ ， $q(n)$  为  $n$  的多项式

- 由于回溯法对同一问题不同实例在计算时间上出现巨大差异，在  $n$  很大时，对某些实例仍然十分有效。

- 用回溯算法处理一棵树所要生成的节点数，可以用蒙特卡罗方法估算出来。



# 6.1 一般方法

## ■ Monte Carlo效率估计

### □ 一般思想

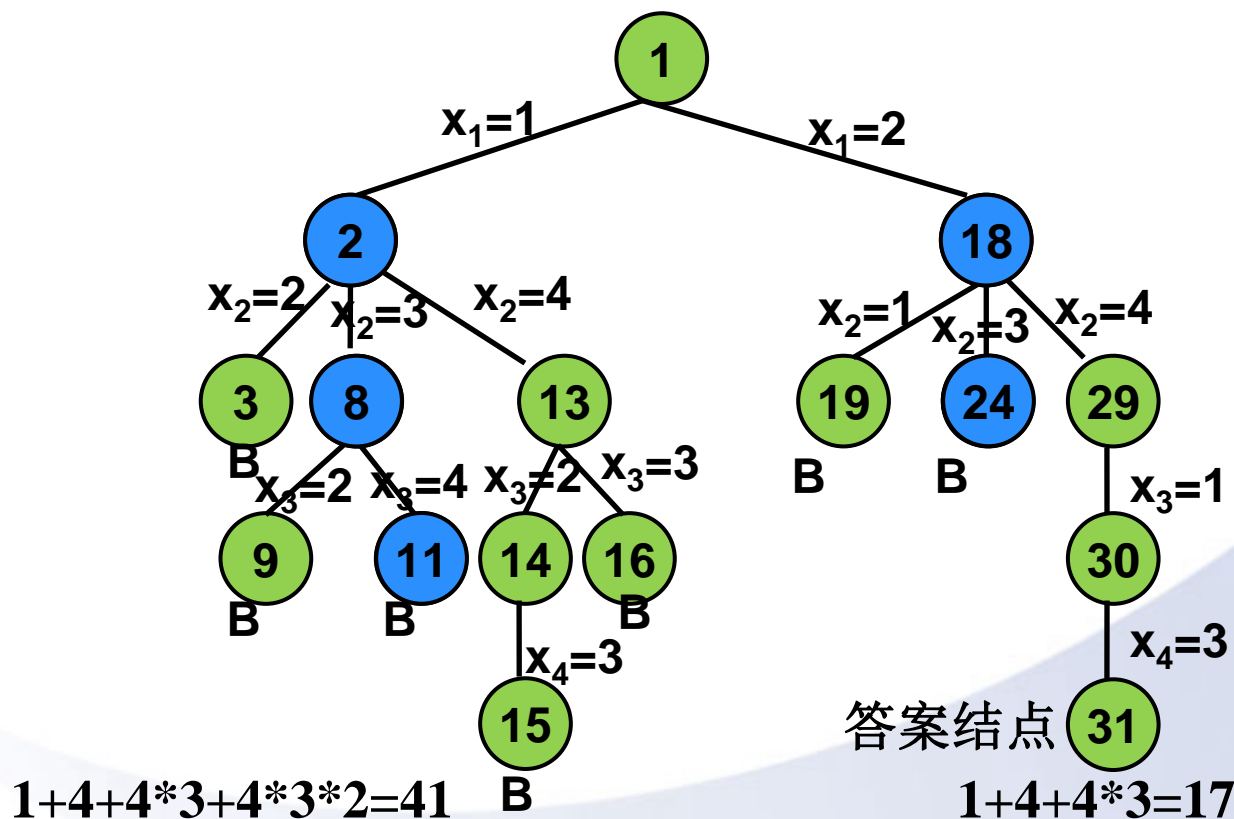
- 在状态空间中生成一条随机路径，
- 设 $X$ 是这条路径上第 $i$ 级的一个结点。
- 在结点 $X$ 处用限界函数确定没受限界的儿子结点数 $m_i$ ，
- 在这 $m_i$ 个儿子结点中随机选择一个结点作为这条路径上的下一个结点
- ...
- 这条路径在以下结点处结束：或者它是一个叶子结点，或者该结点的所有儿子结点都被限界。
- 所有这些 $m_i$ 可估算出状态空间树中不受限界结点的总数 $m$



# 6.1 一般方法

## ■ Monte Carlo效率估计

### □ 使用蒙特卡罗方法估算应用举例



# 6.1 一般方法

## ■ Monte Carlo效率估计

### □ 使用蒙特卡罗方法估算应用举例

- 设第二级没受限结点数为 $m_1$ 。
- 如果同一级上结点有**相同的度**，那么就可以预计到每一个二级结点平均有 $m_2$ 个没界限的儿子，从而得出在第三级上有 $m_1m_2$ 个结点。
- 第四级预计没受限的结点数是 $m_1m_2m_3$ 。
- 一般在 $i+1$ 级上预计的结点数是 $m_1m_2\dots m_i$ 。
- 于是，在求解给定问题的实例中所要生成的不受界限结点的估计数

$$m=1 + m_1 + m_1m_2 + m_1m_2m_3 + \dots$$



# 6.1 一般方法

## ■ Monte Carlo效率估计

□ ESTIMATE是一个确定m值的算法

**procedure** ESTIMATE

$m \leftarrow 1; r \leftarrow 1; k \leftarrow 1$

**loop**

$T_k \leftarrow \{X(k) : X(k) \in T(X(1), \dots, X(k-1)) \text{ and } B(X(1), \dots, X(k))\}$

**if** SIZE( $T_k$ ) = 0 **then exit endif**

$r \leftarrow r * \text{SIZE}(T_k)$       //第k级的结点数

$m \leftarrow m + r$       //前第k级的结点总数

$X(k) \leftarrow \text{CHOOSE}(T_k)$       //从 $T_k$ 中随机地挑选一个元素

$K \leftarrow K + 1$

**repeat**

**return** (m)

**end** ESTIMATE

返回集合 $T_k$ 的大小



中国科学院大学

University of Chinese Academy of Sciences 56



# 第六章 回溯法

- 6.1 一般方法
- 6.2 8-皇后问题
- 6.3 子集和数问题
- 6.4 图的着色
- 6.5 0/1背包问题
- 6.6 哈密顿环
- 6.7 和最小
- 6.8 跳马问题



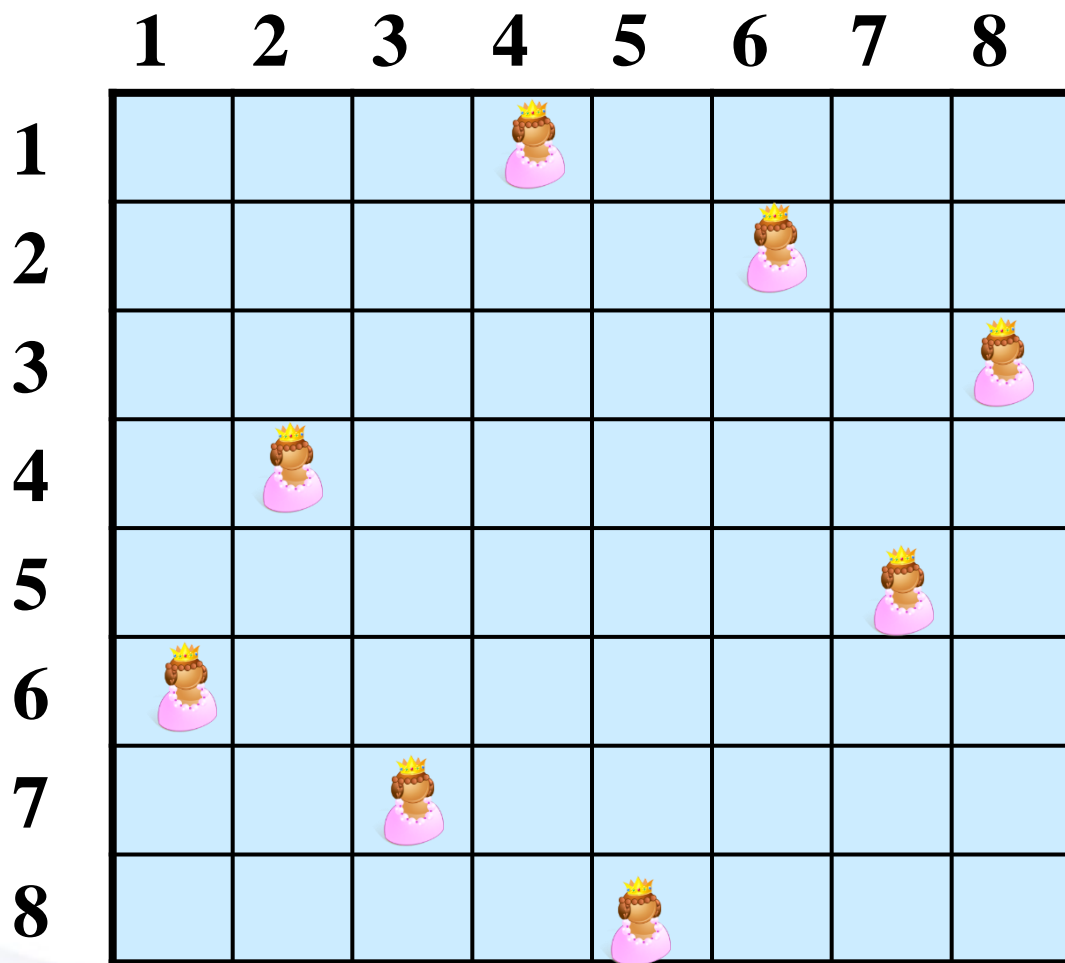
## 6.2 8-皇后问题

### ■ 问题描述

- 将 $n$ 个皇后放置在一个 $n \times n$ 的棋盘上，要求没有两个皇后可以互相攻击。
- 攻击的定义：
  - 两个皇后出现在同一行、或同一列、或者同一条斜线上都视为出现了攻击。



## 6.2 8-皇后问题的一个解



该解的8元组  
表示:

(4, 6, 8, 2, 7, 1,  
3, 5)



中国科学院大学

University of Chinese Academy of Sciences 59

## 6.2 8-皇后问题

- 用 $n$ -元组 $(x_1, x_2, \dots, x_n)$ 表示棋盘上皇后的位置状态
  - 下标表示皇后 $i$  ( $i=1, 2, \dots, n$ )
  - $x_i$ 表示放置皇后 $i$ 所在的列号
- 显式约束条件:
  - 每个 $x_i$ 只从集合 $S_i=\{1, 2, \dots, n\}$ 取值
  - 满足显式约束的所有元组确定一个可能的解空间 → 解空间由 $n^n$ 个 $n$ -元组组成
- 隐式约束条件
  - 没有两个 $x_i$ 可以相同，而且没有两个皇后可以在同一条斜线上 → 由前者得，所有解都是 $n$ -元组 $(1, 2, \dots, n)$ 的置换，因此，解空间缩小为  $n!$  个元组



## 6.2 8-皇后问题

### ■ 测试两个皇后在一条斜角线的方法

$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$	$a_{16}$	$a_{17}$	$a_{18}$
$a_{21}$	$a_{22}$	$a_{23}$	$a_{24}$	$a_{25}$	$a_{26}$	$a_{27}$	$a_{28}$
$a_{31}$	$a_{32}$	$a_{33}$	$a_{34}$	$a_{35}$	$a_{36}$	$a_{37}$	$a_{38}$
$a_{41}$	$a_{42}$	$a_{43}$	$a_{44}$	$a_{45}$	$a_{46}$	$a_{47}$	$a_{48}$
$a_{51}$	$a_{52}$	$a_{53}$	$a_{54}$	$a_{55}$	$a_{56}$	$a_{57}$	$a_{58}$
$a_{61}$	$a_{62}$	$a_{63}$	$a_{64}$	$a_{65}$	$a_{66}$	$a_{67}$	$a_{68}$
$a_{71}$	$a_{72}$	$a_{73}$	$a_{74}$	$a_{75}$	$a_{76}$	$a_{77}$	$a_{78}$
$a_{81}$	$a_{82}$	$a_{83}$	$a_{84}$	$a_{85}$	$a_{86}$	$a_{87}$	$a_{88}$



## 6.2 8-皇后问题

### ■ 测试两个皇后在一条斜角线的方法

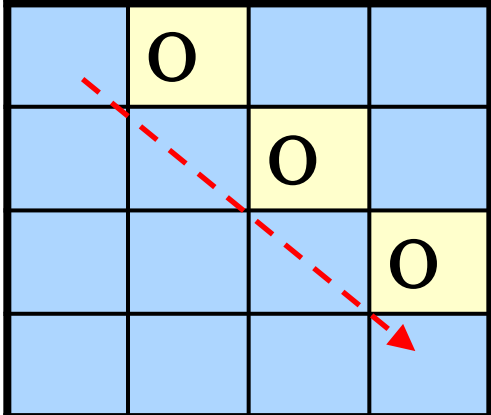
- 令 $(x_1, \dots, x_n)$ 表示一个解，其中 $x_i$ 是把第 $i$ 个皇后放在第 $i$ 行的列数。由于没有两个皇后可以放入同一列，因此这所有的 $x_i$ 将是截然不同。
- 如果设想棋盘的方格像二维数组 $A(1:n, 1:n)$ 的下标那样标记，那么可以看到，
  - 对于在同一条斜角线上的由左上方到右下方的每一个元素有相同的“行-列”值，
  - 在同一条斜角线上的由右上方到左下方的每一个元素则有相同的“行+列”值。



## 6.2 8-皇后问题

### ■ 测试两个皇后在一条斜角线的方法

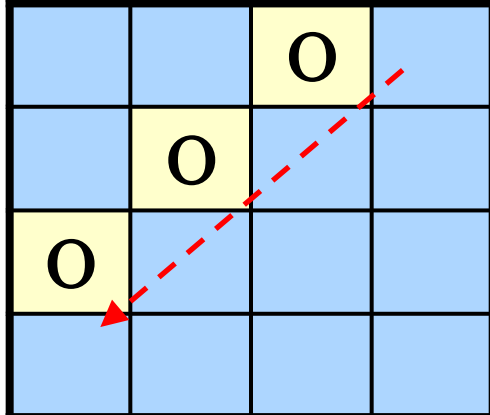
i\j	1	2	3	4
1		O		
2			O	
3				O
4				



左上方——右下方  
相同的“行-列”值

$$1-2 = 2-3 = 3-4$$

i\j	1	2	3	4
1			O	
2		O		
3	O			
4				



右上方——左下方  
相同的“行+列”值

$$1+3 = 2+2 = 3+1$$



## 6.2 8-皇后问题

### ■ 测试两个皇后在一条斜角线的方法

□ 假设有两个皇后被放置在 $(i_1, j_1)$ 和 $(i_2, j_2)$ 位置上，那么根据以上所述，仅当

$i_1 - j_1 = i_2 - j_2$  或  $i_1 + j_1 = i_2 + j_2$   
时，它们才在同一条斜角线上。

□ 将这两个等式分别变换成

$i_1 - i_2 = j_1 - j_2$  和  $i_1 - i_2 = j_2 - j_1$   
因此，当且仅当 $|i_1 - i_2| = |j_1 - j_2|$ 时，两个皇后在同一条斜角线上。





## 6.2 8-皇后问题

### ■ 检测放置新皇后的算法

- 初始化空棋盘（起始状态）
- 从第一行开始，直至第一行出现回溯
  - ◆ 从当前行 $r$ 中查找下一个可以放置皇后的位置
  - ◆ 如果找到了可以放置的位置
    - 放下一个皇后
    - 如果已经是最后一个行
      - ✓ 得到一个解
      - ✓ 撤掉该子，继续寻找下一个解
    - 否则(未到最后一行)
      - ✓ 准备处理下一行
  - ◆ 否则（没有找到可以放置的位置）
    - 回溯到上一行，并撤掉该行的棋子



## 6.2 8-皇后问题

### ■ 检测放置新皇后的算法

**procedure** PLACE(k)

//如果一个皇后可以放在第k行和X(k)列，则返回true； 否则返回false//

**global** X(1:k); **integer** i, k

i ← 1

**while** i < k **do**

**if** X(i) = X(k) //在同一列有两个皇后

**or** ABS(X(i)-X(k)) = ABS(i-k) //在同一条斜角线上

**then return** (false)

i ← i + 1

**repeat**

**return** (true)

**End** PLACE

**PLACE(k):**

- 令X(k)与X(i)逐个比较，i=1...k-1。
- 若存在X(k)=X(i)或者|X(i)-X(k)|=|i-k|
  - ✓ 则返回false;
  - ✓ 否则返回true。



## 过程NQUEENS求n-皇后问题的所有解

**procedure** NQUEENS(n)

**integer** k, n, X(1:n)

X(1)  $\leftarrow$  0; k  $\leftarrow$  1      //K是当前行; X(k)是当前列

**while** k > 0 **do**      //对所有的行执行一下语句

    X(k)  $\leftarrow$  X(k)+1      //移到下一列

**while** X(k)  $\leq$  n **and not** PLACE(k) **do** //此处能放这个皇后吗

        X(k)  $\leftarrow$  X(k)+1

**repeat**

**if** X(k)  $\leq$  n **then**      //找到一个位置

**if** k=n **then**      //是一个完整的解吗

                print (X) //是, 打印这个数组

**else** k  $\leftarrow$  k+1, X(k)  $\leftarrow$  0 //转到下一行

**endif**

**else** k  $\leftarrow$  k-1      // 没有合适的位置, 回溯

**endif**

**repeat**

**End** NQUEENS

当该位置不能放皇后时转到下一列



中国科学院大学

University of Chinese Academy of Science 67

## 6.2 8-皇后问题

### ■ 算法分析

#### □ NQUEENS 优于硬性处理

##### ➤ 硬性处理:

- ✓ 要求 $8 \times 8$ 的棋盘排出8块位置, 有 $(8^8)$ 种可能的方式。
- ✓ 即要检查将近 $4.4 \times 10^9$ 个8-元组。

##### ➤ NQUEENS 回溯法:

- ✓ 只允许把皇后放置在不同的行和列上, 至多需要作 $8!$ 次检查,
- ✓ 即至多只检查40320个8-元组。



## 6.2 8-皇后问题

### ■ 算法分析

□ 估算节点数(蒙特卡罗方法):

- 假设使用固定的限界函数且在检索进行时函数不改变。
- 在状态空间树的**同一级的所有节点都有相同的度**。

静态的状态空间树节点个数为:

$$\begin{aligned} & 1+8+8*7+8*7*6+\dots+8*7*6*5*4*3*2*1 \\ & =1+\sum_{j=0}^7 8*...*(8-j)=1+\sum_{j=0}^7 \prod_{i=0}^j (8-i)=69281 \end{aligned}$$



## 6.2 8-皇后问题

### ■ 效率估计

	1						
			2				
3							
		4					
			5				

8 1+  
 5 8+  
 4 8\*5+  
 3 8\*5\*4+  
 2 8\*5\*4\*3+  
 8\*5\*4\*3\*2  
 =1649

1							
							2
					3		
		4					
						5	
	6						
			7				

8  
 6  
 4  
 2  
 1  
 1  
 1  
 =1401

这5次试验的平均值是1625

不受限节点的估计数大约是8-皇后状态空间树的节点总数的

$1625/69281=2.34\%$



中国科学院大学

University of Chinese Academy of Sciences 70

# End

