

吉林大学



第七章 回溯法



目录

- 7.1 一般方法
- 7.2 效率估计
- 7.3 n -皇后问题
- 7.4 子集和数问题
- 7.5 图着色问题
- 7.6 小结



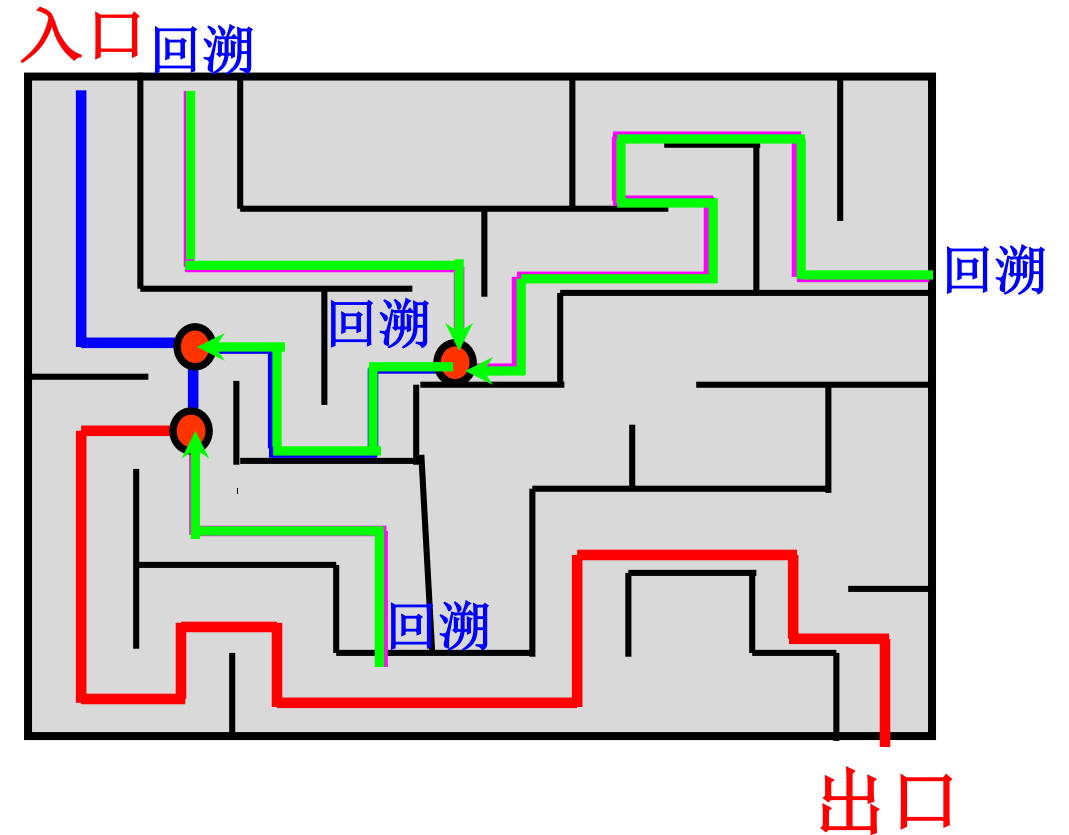
7.1 一般方法

- 适用的问题特点
- 多米诺性质
- 基本概念
- 动态树
- 设计思想
- 算法描述



回溯法的基本思想

- 例：迷宫游戏
- 回溯法是一种搜索算法，是通用的解题法。
- 以深度优先的方式系统地搜索问题的解。



方法适用的问题特点

- 方法适用于解决多阶段决策问题，也称为组合问题
 - 问题的解向量用元组来表示，元素 x_i 通常取自于某个有穷集 S_i ， $1 \leq i \leq n$ ， n 表示问题规模
 - 固定长 n -元组 (x_1, \dots, x_n)
 - 可变长 k -元组 (x_1, \dots, x_k) ， $k < n$
 - 问题的目标：
 - 满足约束条件的一个解或多个解；通常用限界函数表示约束条件；
 - 满足约束条件的最优解；此时需要定义目标函数，使目标函数取极值的解是最优解。
 - 问题满足多米诺性质
 - 适用于求解组合数较大的问题。

组合搜索

组合优化



细分多阶段决策问题

- 多阶段决策问题/组合问题
 - 组合搜索：关注解的存在性
 - 问题描述中**仅存在**约束条件
 - 问题希望找到一个可行解/所有可行解
 - 组合优化：关注解的最优性
 - 问题描述中**还存在**目标函数
 - 问题希望找到一个具有极小值/极大值的可行解，即最优解
 - 也称为极小化/极大化问题



多米诺性质

- 多米诺性质
 - 设 $P(x_1, \dots, x_i)$ 是关于向量 (x_1, \dots, x_i) 的某种性质的判定，当 $P(x_1, \dots, x_{i+1})$ 为真时，一定有 $P(x_1, \dots, x_i)$ 为真， $0 < i < n$
- 根据多米诺性质，如果 $P(x_1, \dots, x_i)$ 不成立，则 $P(x_1, \dots, x_{i+1})$ 亦不成立
- 一个满足多米诺性质的组合问题
 - 是指能够根据约束条件和目标函数不断检验正在构造的部分解向量 (x_1, \dots, x_i) ， $0 < i < n$ ，一旦发现不成立，则无需考虑后续 x_{i+1}, \dots, x_n 的取值



问题的多米诺性质是回溯法提高算法效率的关键

- 设**限界函数B**实现问题的约束条件，对于n-元组 (x_1, \dots, x_n) , $x_i \in S_i$,
- 硬性处理
 - $|S_i|=m_i$, 向量个数 $m=m_1 \times m_2 \times \dots \times m_n$, 对这m个n-元组逐一检测是否满足 $B(x_1, \dots, x_n)$, 从而找出问题的解。
- 回溯法利用多米诺性质
 - **B**无需等待，可以提前检验正在构造中的部分向量 (x_1, \dots, x_i) ，如果发现不能导致问题的解，终止该向量继续构造，即不再构造 x_{i+1}, \dots, x_n 的取值，从而减少了 $m_{i+1} \times \dots \times m_n$ 个向量。

测试次数比硬性处理的m次要少得多

回溯求解的基本概念

- 显式约束：每个 x_i 的取值集合 S_i ，可能与问题实例有关，也可能无关
 - $x_i \geq 0$ ， $S_i = \{\text{所有非负实数}\}$
 - $x_i = 0 \text{ 或 } 1$ ， $S_i = \{0, 1\}$
- 隐式约束：描述了 x_i 彼此相关的情况，与问题实例有关
- 解空间：满足显式约束条件的所有元组
- 可行解：解空间中满足隐式约束条件的元组。
- 解空间树：基于解空间画成的树形状

对应限界函数



解空间树

回溯法解决问题的过程就是在解空间树上搜索答案状态结点的过程

- 解空间树：基于解空间画成的树形状
 - 问题状态：解空间树中的所有结点。
 - 解状态：是这样的一些问题状态 X ，对于这些问题状态，由根到节点 X 的那条路径确定了解空间中的一个元组。

满足显式约束条件
 - 答案状态：是这样的一些解状态 X ，对于这些解状态而言，由根到节点 X 的这条路径确定了问题的一个解，也就是说，它满足隐式约束条件。

答案状态 \subseteq 解状态 \subseteq 问题状态

问题建模

- 确定元组表达形式
 - 固定长元组：用大小固定的 n -元组 (x_1, \dots, x_n) 表示
 - 不定长元组：用大小可变的 k -元组 (x_1, \dots, x_k) 表示, $k \leq n$
- 确定约束条件：显示约束和隐式约束
- 检验问题满足多米诺性质
- 确定解空间树：问题状态、解状态和答案状态



以子集和问题为例

- 问题描述：
 - 已知 $n+1$ 个正数： $w_i (1 \leq i \leq n)$ 和 M , 要求找出 w_i 的和是 M 的所有子集。
- 问题实例：
 - $n=4$, $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$, $M=31$ 。
 - 可行解1: 11, 13, 7
 - 可行解2: 24, 7



固定长元组表达

- n -元组 (x_1, \dots, x_n)
- 显式约束: $x_i \in \{0, 1\}, 1 \leq i \leq n$; 如果选择 w_i , 则 $x_i = 1$; 否则 $x_i = 0$
- 隐式约束: $\sum w_i x_i = M, 1 \leq i \leq n$
- 多米诺性质: 如果部分解向量大于 M , 则包含它的解向量也大于 M
- 解空间共计 2^n 个元组

$n=4, (w_1, w_2, w_3, w_4) = (11, 13, 24, 7), M=31$ 。 解空间: $2^4=16$ 个元组

可行解1: 11, 13, 7

可行解1: (1, 1, 0, 1)

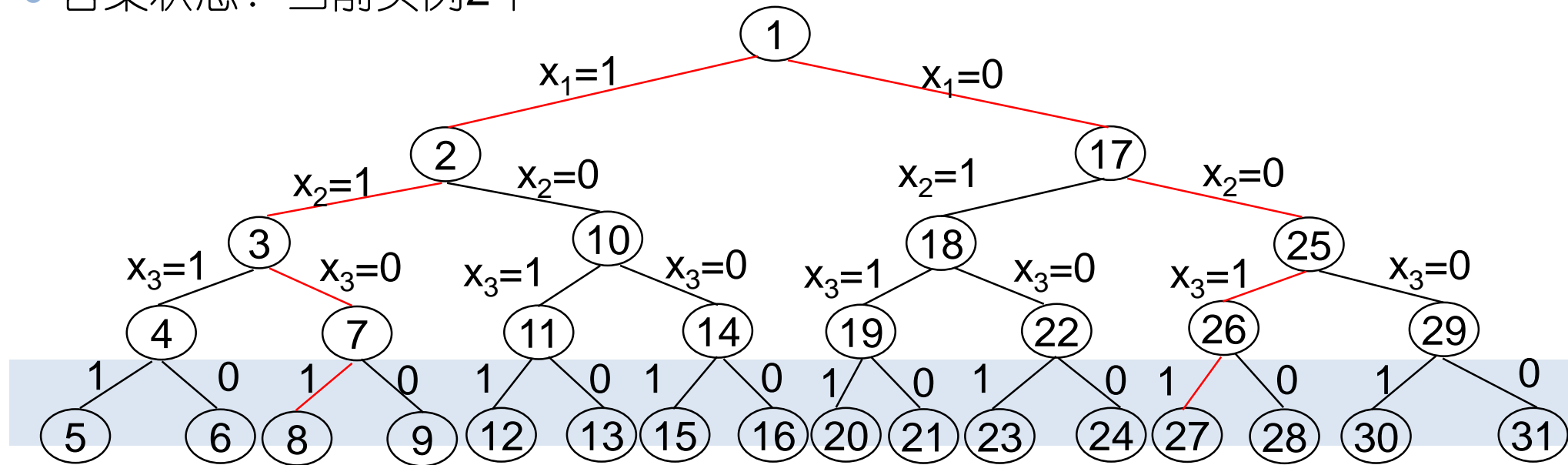
可行解2: 24, 7

可行解2: (0, 0, 1, 1)



子集和数问题的4-元组表达的解空间树

- 问题状态：全部结点31个
- 解状态：叶结点16个
- 答案状态：当前实例2个



从根结点到叶结点的一条路径确定解空间中的一个元组

结点8和27是答案状态

不定长元组表达

避免重复情况,
如(1,2,4)和(1,4,2)

- k -元组 (x_1, \dots, x_k) , $k \leq n$
- 显式约束: $x_i \in \{j \mid j \text{ 是 } w_j \text{ 的下标}, 1 \leq j \leq n\}$, $1 \leq i \leq k$ 。 $x_i < x_{i+1}$, $1 \leq i < k$
- 隐式约束: 相应的 w_i 的和等于 M
- 多米诺性质: 如果部分解向量大于 M , 则包含它的解向量也大于 M
- 解空间共计 2^n 个元组

$n=4$, $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$, $M=31$ 。 解空间: $2^4=16$ 个元组

可行解1: 11, 13, 7

可行解1: (1, 2, 4)

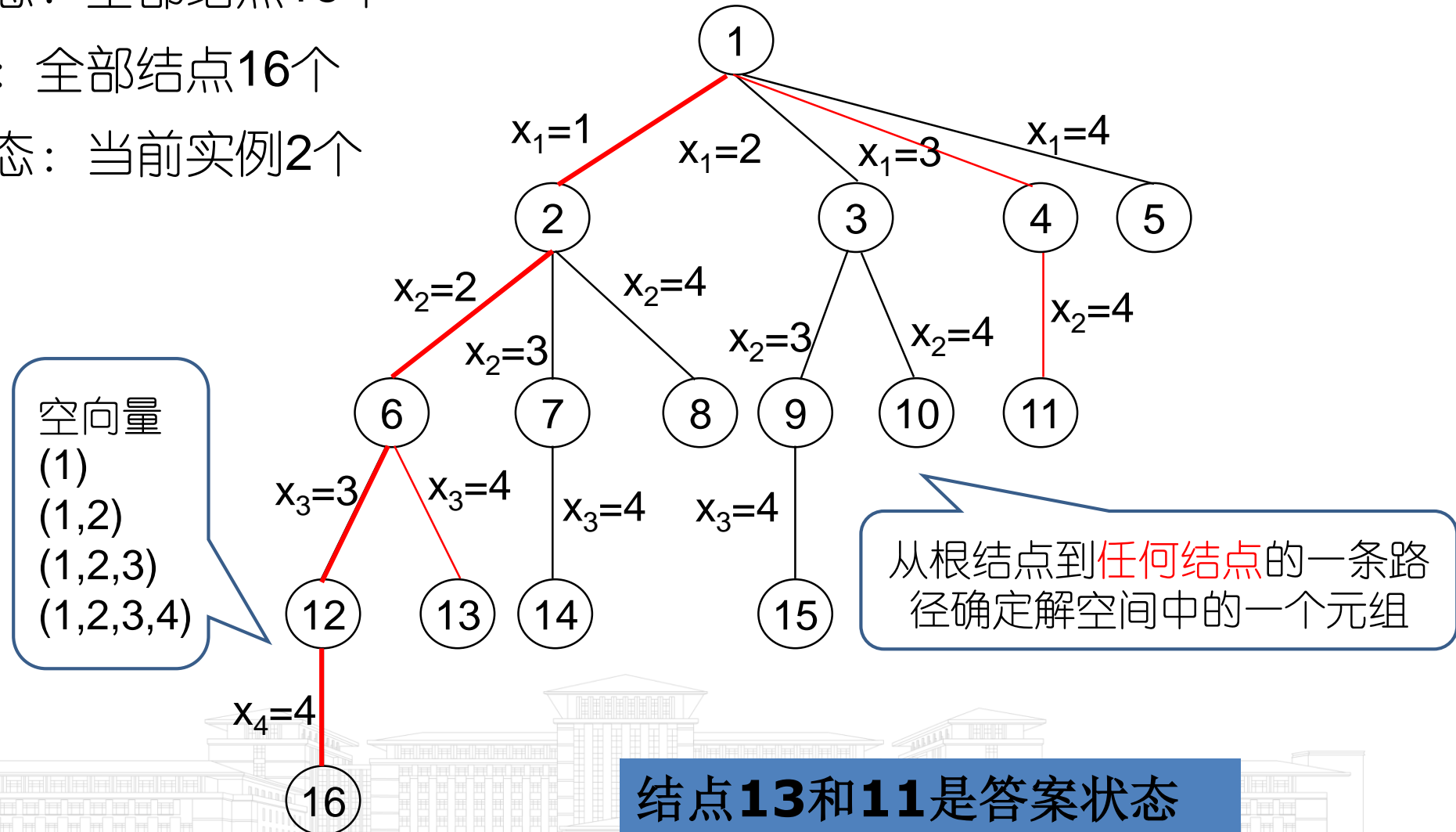
可行解2: 24, 7

可行解2: (3, 4)



- 子集和数问题的 $n=4$ 时， k -元组表达的解空间树

- 问题状态：全部结点16个
- 解状态：全部结点16个
- 答案状态：当前实例2个



动态树

- 静态树: 即解空间树, 树结构与所要解决的问题实例无关。
- 动态树: 树结构与实例相关, 在求解过程中生成结点。
 - **活结点**: 自己已经生成而其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。



动态树结点生成方式

- 第一种方式

- 当前的**E-结点R**一旦生成一个新的儿子结点**C**, 这个**C**结点就变成一个新的**E-结点**, 当检测完了子树**C**后, **R**结点就再次成为**E-结点**, 生成下一个儿子结点。

该方法基于**深度优先**, 称为**回溯法**。

- 第二种方式:

- 当前结点一旦成为**E-结点**, 就一直处理到变成死结点为止。其生成的儿子结点加入到活结点表中, 然后再从活结点表中选择下一个新的**E-结点**。

该方法称为**分支限界法**。



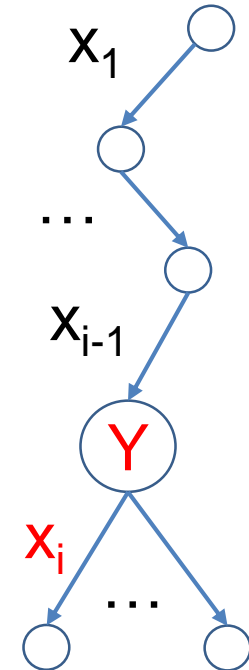
回溯法的设计思想

- 针对问题定义解空间树结构：元组、显式约束条件、隐式约束条件。
- 检验问题满足多米诺性质。
- 以深度优先方式搜索解空间树，在搜索过程中使用限界函数避免无效搜索。
 - 首先根结点成为一个活结点，同时也是当前的扩展结点。沿当前扩展结点向纵深方向移至一个新的活结点，该活节点成为当前新的扩展结点。
 - 如果当前扩展结点不能再向纵深方向移动，则其成为死结点。回溯至最近的一个活结点，并使该活结点成为当前新的扩展结点。
- 在解空间树中搜索，直至找到所要求的解或解空间中已没有活结点时为止。



回溯法的形式化描述

- 假设要找出所有的答案结点
- $(x_1, x_2, \dots, x_{i-1})$ 是状态空间树中由根出发的一条路径，到达结点 Y
- $T(x_1, \dots, x_{i-1})$ 是元素 x_i 的集合，对于每一个 x_i ， $(x_1, x_2, \dots, x_{i-1}, x_i)$ 是一条由根到结点 Y 的一个儿子结点的路径
- 对于限界函数 B_i ，如果路径 $(x_1, x_2, \dots, x_{i-1}, x_i)$ 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



算法7.1 回溯法的非递归算法描述

```
procedure BACKTRACK(n)
```

```
  int k, n
```

```
  local X(1: n)
```

```
  k ← 1
```

```
  while (k>0) do
```

```
    if (还剩有没检验的 $X(k)$ 使得 $X(k) \in T(X(1) \dots X(k-1))$ )
```

```
      and  $B(X(1) \dots X(k)) = \text{TRUE}$ )
```

```
    then if ( $X(1) \dots X(k)$ )是一条抵达答案结点的路径)
```

```
      then print (  $X(1) \dots X(k)$ )
```

```
    endif
```

```
      k ← k+1
```

```
    else k ← k-1 //回溯
```

```
  endif
```

```
  repeat
```

```
end BACKTRACK
```

在 $X(1) \dots X(k-1)$ 已经被确定的情况下, $T(X(1) \dots X(k-1))$ 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 $X(k)$ 满足隐式约束条件

算法7.2 回溯法的递归算法描述

```
procedure RBACKTRACK(k)
global X(1:n);
int k, n;
for ( 满足下式的每个X(k),  $X(k) \in T(X(1)...X(k-1))$ 
    and  $B(X(1),...X(k))=true$ ) do
    if  $(X(1),...,X(k))$ 是一条抵达答案结点的路径 then
        print  $(X(1)...X(k))$  endif
    call RBACKTRACK(k+1)
repeat
end RBACKTRACK
```

进入算法时, 解向量 X 中的前 $k-1$ 个分量 $X(1) \dots X(k-1)$ 已经被赋值



7.2 回溯法的效率分析

- 决定回溯法效率的因素
- 回溯法的效率估计
- 蒙特卡罗方法的一般思想
- 效率估计算法
- 蒙特卡罗方法的特点



决定回溯法效率的因素

- 生成下一个 $X(k)$ 的时间
 - 生成一个结点的时间
- 满足显式约束条件的 $X(k)$ 的数目
 - 子结点的数量
- 限界函数 B_i 的计算时间
 - 检验结点的时间
- 对于所有的 i , 满足 B_i 的 $X(k)$ 的数目
 - 通过检验的结点数量

B_i 能够大大减少生成的结点数, 但在计算时间和减少程度上要进行折中

第四个

思考：哪一个因素会导致不同实例产生的结点数不同？

决定回溯法效率的因素

- 一旦选定了一种状态空间树结构, 前三种因素对于所要解决的实例没有多大的关系, 只有第四种因素, 对于问题的不同实例, 生成的结点数是不相同的。
- 对于某一实例, 回溯算法可能只生成 $O(n)$ 个结点;
- 对于另一实例, 回溯算法可能生成这棵状态空间树的全部结点。



回溯法的效率估计

- 如果解空间的结点数是 2^n 或 $n!$ ，易知，回溯算法最坏情况下的时间复杂度为 $O(p(n)2^n)$ 或 $O(q(n)n!)$ ，其中 $p(n)$ 和 $q(n)$ 为 n 的多项式

B_i 花费的时间等

- 由于回溯法对同一问题不同实例的巨大差异，在 n 很大时，对某些实例是十分有效的。因此，在采用回溯法计算某个实例之前，应估算其工作效能。
- 用回溯算法处理一棵树所要生成的结点数，可以用蒙特卡罗方法估算出来

估计活结点的个数，
即动态树结点个数



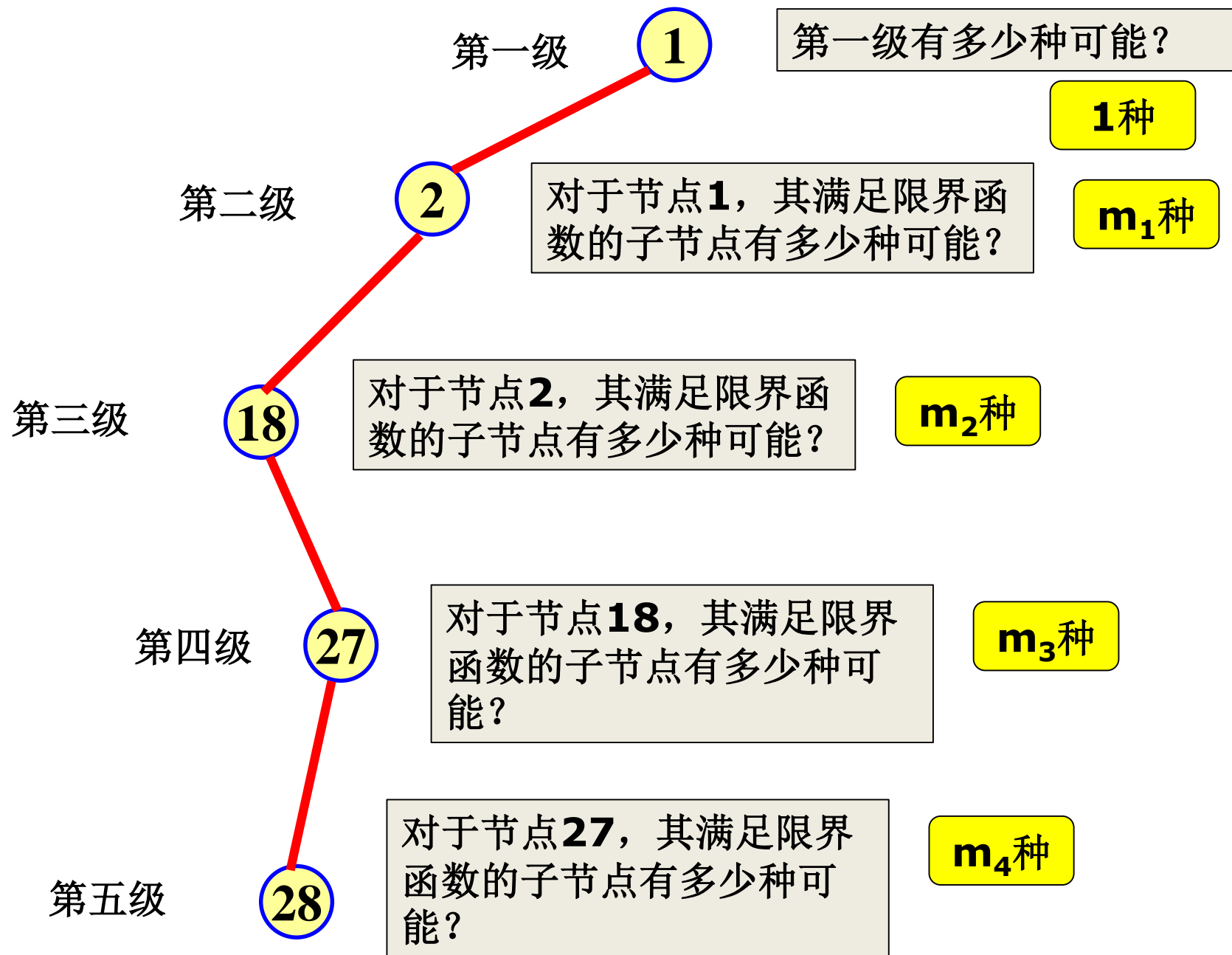
蒙特卡罗方法的一般思想

- 假定限界函数是**固定**的
 - 在状态空间中生成一条随机路径。
 - 设 x 是这条路径上的位于第 i 级的一个结点。
 - 设限界函数确定 x 的可用儿子结点的数目为 m_i 。
 - 从这 m_i 个儿子结点中随机选中一个，重复上述过程，直到当前结点是叶结点或者儿子结点都被限界为止。

不受限界结点的估计数: $m = 1 + m_1 + m_1 * m_2 + m_1 * m_2 * m_3 + \dots$

m_i 表示第 i 级结点没受限界的儿子结点数。





全部不受限结点的估计数目

- 第一级不受限界的结点数为1；
- 第二级不受限界的结点数为 m_1 ；
- **预计**每个二级结点**平均**有 m_2 个没限界的儿子结点，则在第三级上有不受限界的结点 $m_1 * m_2$ 个；
- 第四级预计没受限结点个数为 $m_1 * m_2 * m_3$ 个；
-
- 所有不受限结点的估计数为 $m = 1 + m_1 + m_1 m_2 + m_1 m_2 m_3 + \dots$



算法7.3 效率估计算法

Procedure ESTIMATE() //程序沿着状态空间树中一条随机路径产生这棵树中不受限界结点的估计数//

$m \leftarrow 1; r \leftarrow 1; k \leftarrow 2$

loop

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

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

$X(k) \leftarrow \text{CHOOSE}(T_k)$

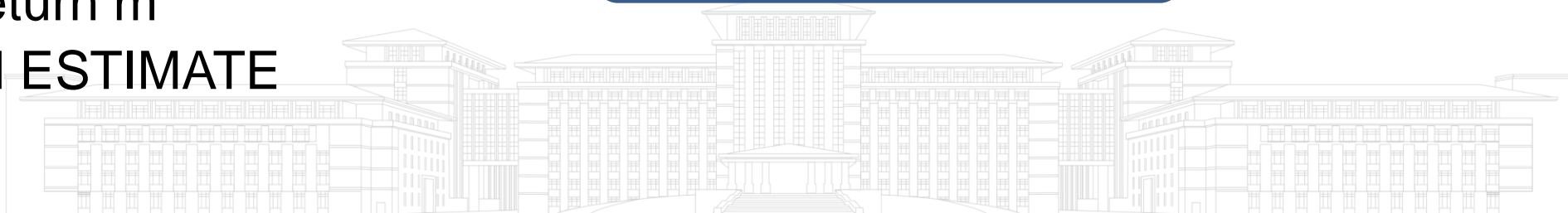
从 T_k 中随机地挑选一个元素

$k \leftarrow k + 1$

repeat

return m

end ESTIMATE



蒙特卡罗方法的特点

- 优点:

找到**所有**答案结点的情况非常有用,

限界函数固定不变, 计算方便, 对状态空间树中同一级结点都适用。

- 缺点:

只求**一个**解时, 生成的结点数远小于 m ,

随着检索的进行, 限界函数应该更强, 使得 m 的值更小。



7.3 n-皇后问题

- 问题描述
- 解空间树
- 问题分析
- 限界函数
- 算法描述
- 效率估计



问题描述

- n-皇后问题:

- 在一个 $n \times n$ 棋盘上放 n 个皇后, 使每两个皇后之间都不能互相“攻击”, 即使得每两个皇后都不能在**同一行、同一列及同一条斜角线上**。

- 基于回溯法求解:

- n-元组 (x_1, \dots, x_n) : 表示皇后 i 放在 i 行 x_i 列上。
- 显式约束条件: $x_i \in \{1, 2, \dots, n\}, 1 \leq i \leq n$
- 隐式约束条件: 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。

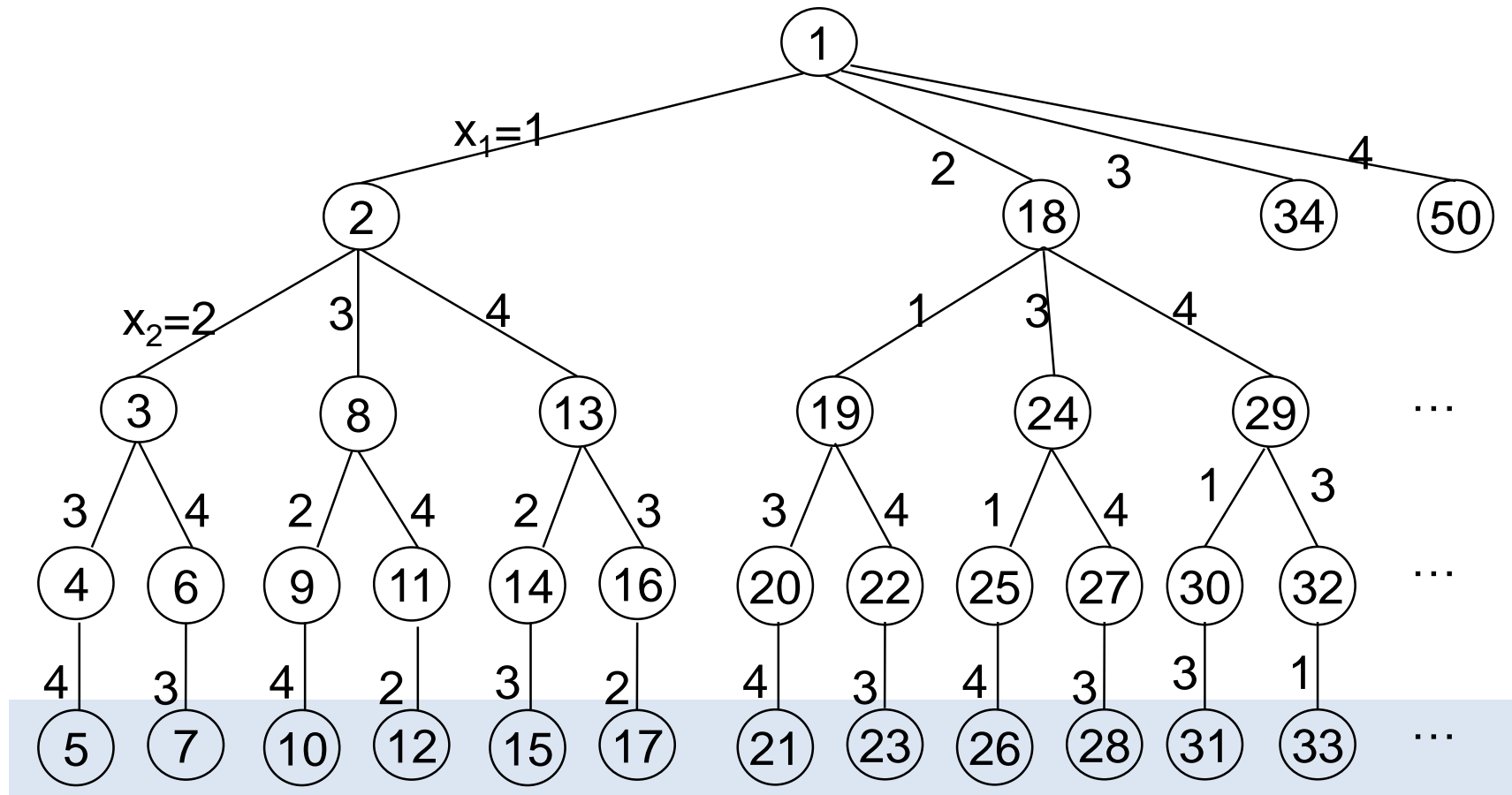
解空间: n^n

与问题实例无关,
解空间: $n!$



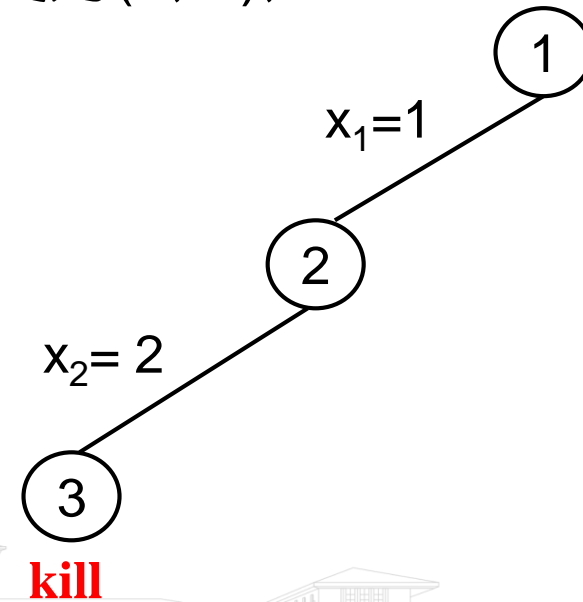
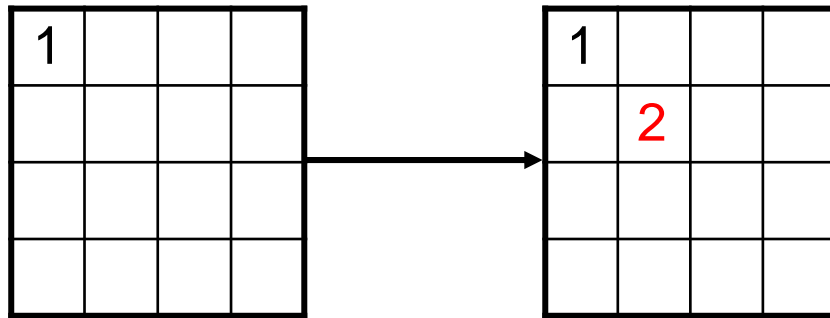
解空间树

- $n=4$ 时，叶结点个数= $4! = 24$ ，解空间是从根结点到叶结点的所有路径。

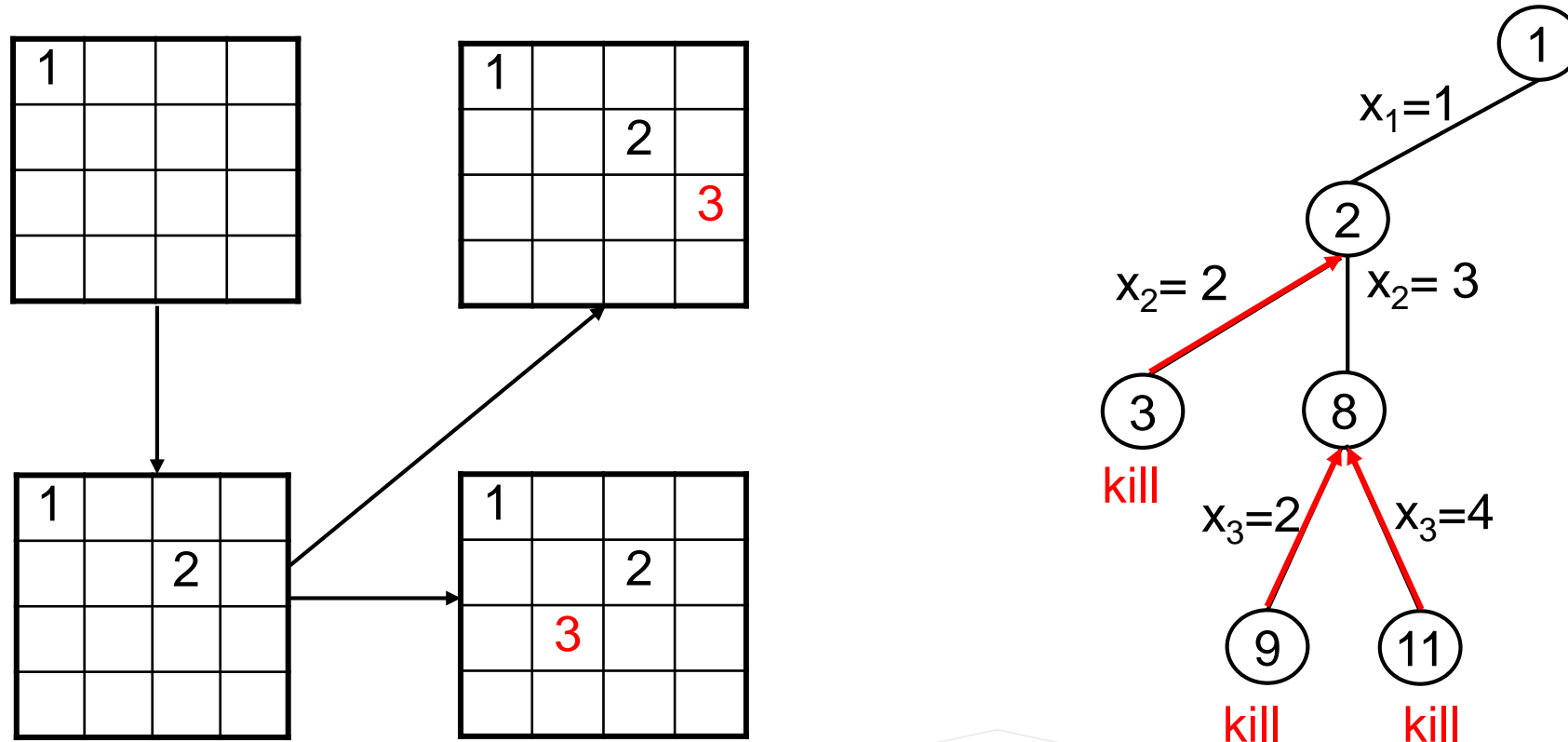


问题分析

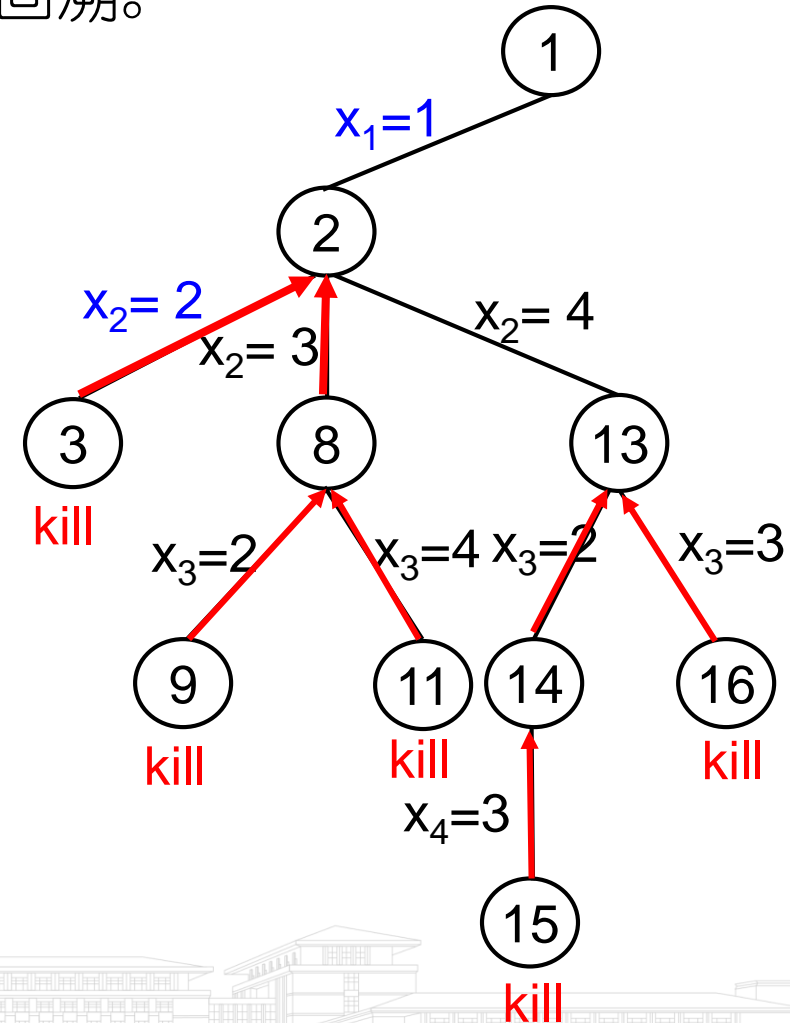
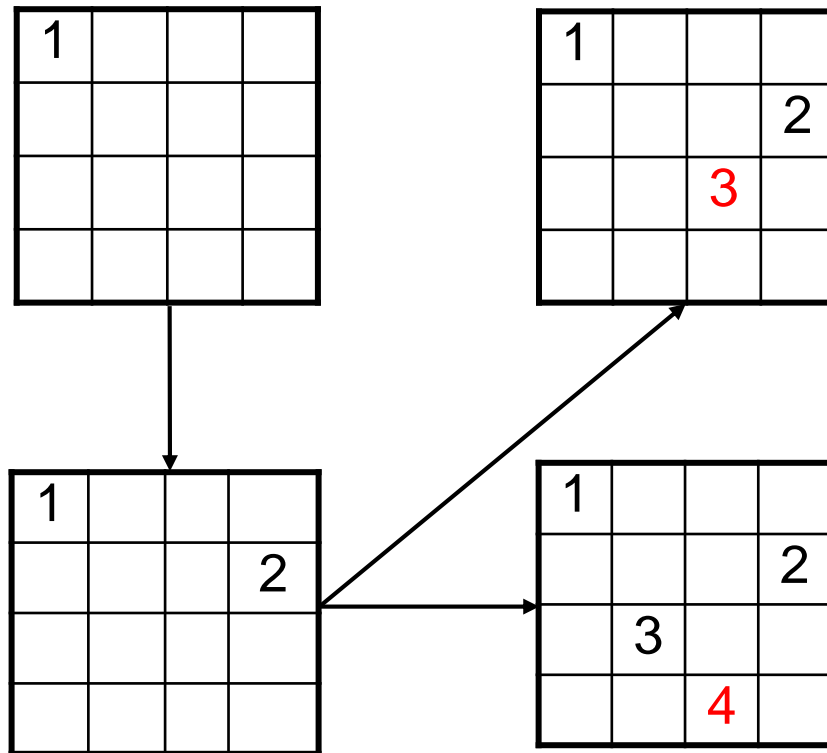
- 开始把根结点作为唯一的活结点, 根结点就成为E-结点而且路径为(); 接着生成儿子结点, 那么结点2被生成, 这条路径为(1), 即把皇后1放在第1列上。
- 结点2变成E-结点, 它再生成结点3, 路径变为(1, 2), 结点3被杀死, 此时回溯。



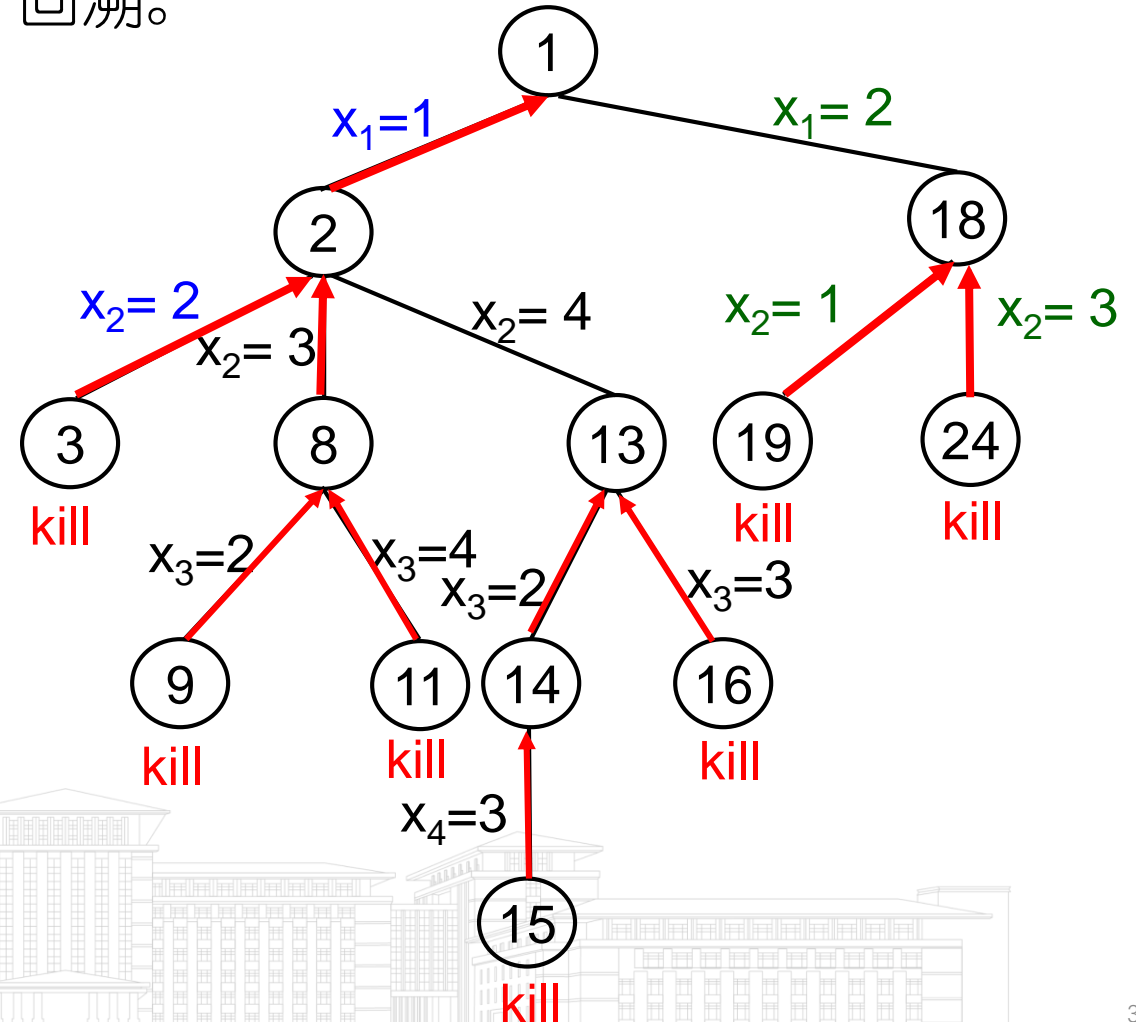
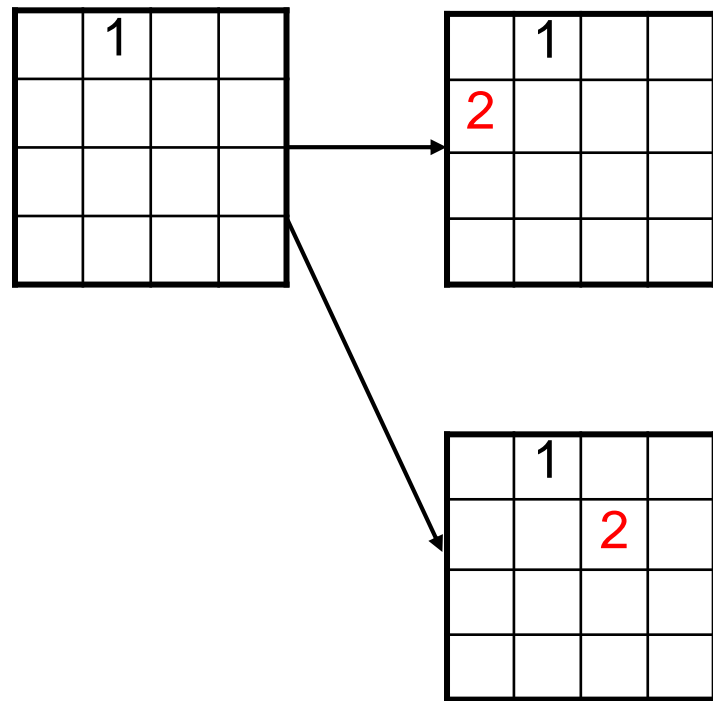
- 回溯到结点2生成结点8, 路径变为(1, 3), 则结点8成为E-结点, 它生成结点9和结点11都会被杀死, 所以结点8也被杀死, 应回溯。



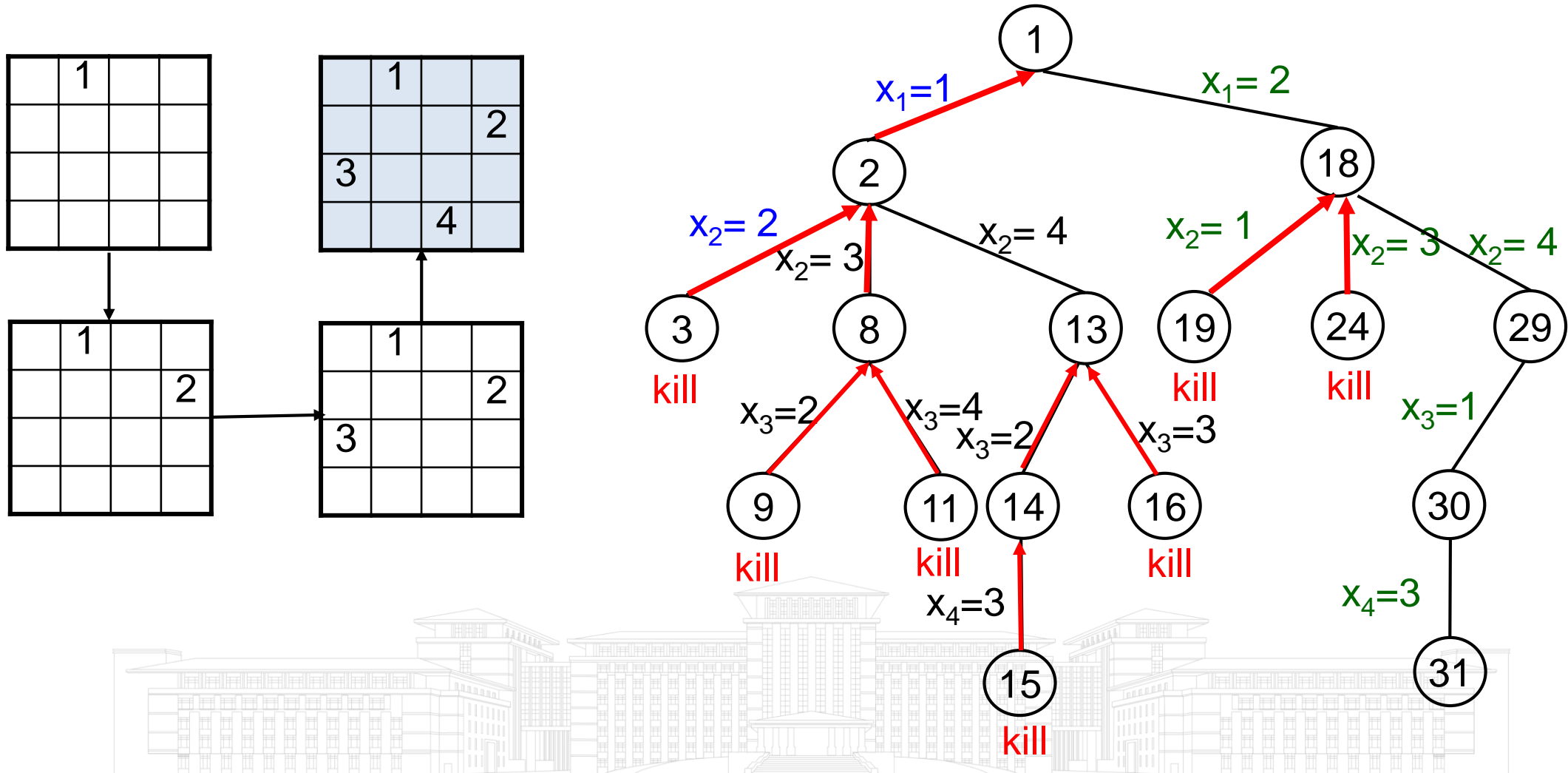
- 回溯到结点2生成结点13, 路径变为(1, 4), 结点13成为E-结点, 它的儿子不可能导致答案结点, 因此结点13也被杀死, 回溯。



- 结点2的所有儿子都不能导致答案棋盘格局, 因此结点2也被杀死; 再回溯到结点1生成结点18, 路径变为(2)。
- 结点18的儿子结点19、结点24被杀死, 回溯。



- 结点18生成结点29, 结点29成为E-结点, 路径变为(2,4)。
- 结点29生成结点30, 路径变为(2,4,1)。
- 结点30生成结点31, 路径变为(2,4,1,3), 找到一个4-皇后问题的可行解。



限界函数

- 在n-皇后问题中， (x_1, x_2, \dots, x_n) 表示一个解， x_i 表示第i个皇后放在第i行的列数。
- 设有两个皇后被放置在 (i, j) 和 (k, l) 位置上，
什么情况下它们在同一条斜角线上？
- ✓ 对于两个皇后所在位置，如果行差距=列差距，
则两个皇后在同一条斜角线上

$$|j - l| = |i - k|$$

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}



限界函数

// 前 $k-1$ 行的皇后已经放置，现在确定第 k 行皇后欲放在 $X(k)$ 列上，是否可以？

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。



算法7.4 能否放置一个新皇后？

procedure PLACE(k)

//若一个皇后能放在第k行和第X(k)列, 则返回true, 否则返回false。

//X是全程数组, 进入此过程时已置入了k个值, ABS是绝对值函数。

int i, k

i ← 1

while (i < k) do

if (X(i)=X(k) or $ABS(X(i)-X(k))=ABS(i-k)$) // 若两个皇后在同一列上, 或在
 then return false
endif

i ← i+1

repeat

return true

end PLACE

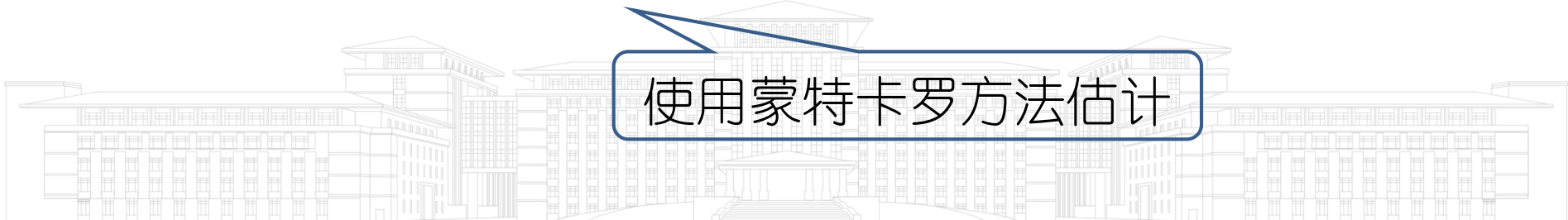


算法7.5 n-皇后问题的回溯算法描述

```
procedure NQUEENS(n)
int k, n, X(1:n)
X(1) ← 0 ; k ← 1 // k是当前行; X(k)是当前列
while (k>0) do // 对所有的行执行循环语句
    X(k) ← X(k)+1 // 移到下一列
    while ( X(k) ≤ n and not PLACE(k) ) do
        X(k) ← X(k)+1; repeat //当前列X(k)不能放皇后k时, 放到下一列
    if(X(k) ≤ n) // 找到一个位置
        then if(k=n) // 若是一个完整的解则打印数组X
            then print (X)
            else k ← k+1; X(k) ← 0 //准备求解下一个皇后
        endif
    else k ← k-1; //没有合适的位置, 回溯
endif
repeat
end NQUEENS
```

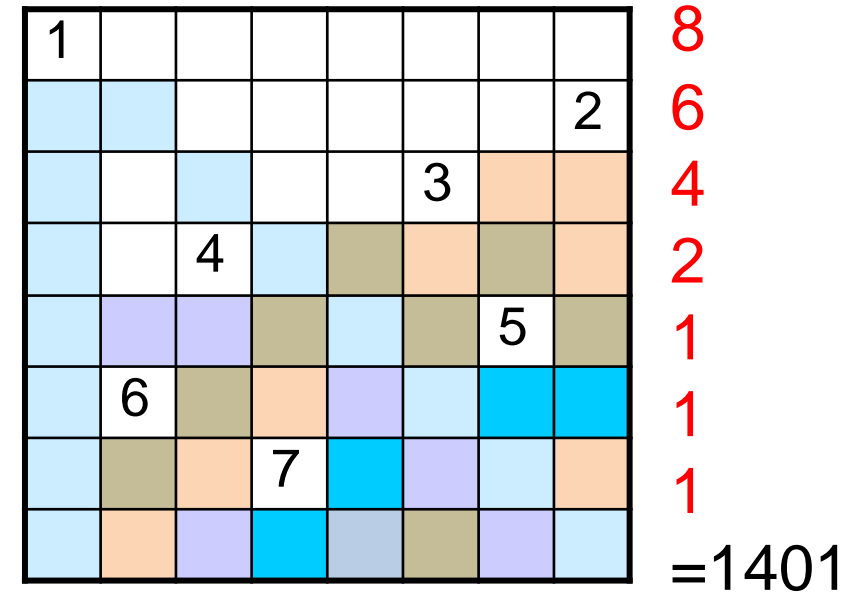
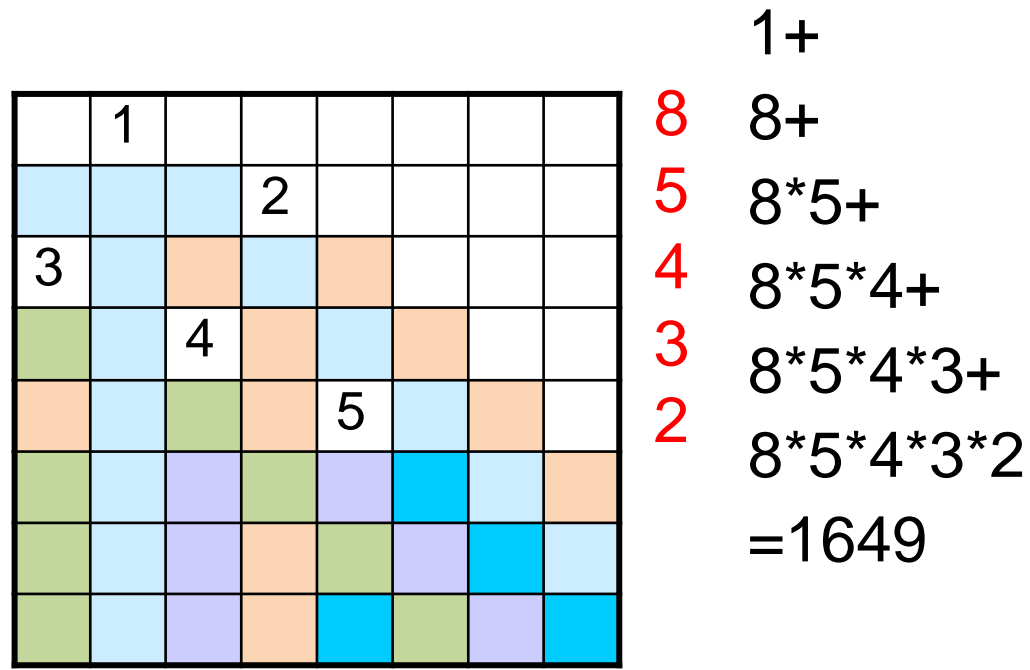
8-皇后问题的效率估计

- 在8-皇后问题中, (x_1, x_2, \dots, x_8) 表示一个解, x_i 表示第*i*个皇后放在第*i*行的列数.
 - 显式: $S_i = \{1, 2, 3, 4, 5, 6, 7, 8\}, 1 \leq i \leq 8$
 - 隐式: 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。
- 硬性处理法(8*8棋盘上随便8个位置): 要检查元组个数为 8^8
 - 状态空间树结点个数: $1+8+8^2+\dots+8^8$
- 没有两个 x_i 可以相同: 要检查元组个数为 $8!$
 - 状态空间树结点个数: $1+8+8*7+8*7*6+\dots+8*7*6*5*4*3*2*1=69281$
- 限界函数实现隐式约束条件 ?



使用蒙特卡罗方法估计

8-皇后问题的不受限结点的估计值



多次实验后取平均值1625

不受限结点的估计数大约是8-皇后状态空间树的结点总数的

$$1625/69281=2.34\%$$



7.4 子集和数问题

- 问题描述
- 限界函数
- 效率估计
- 递归回溯算法
- 实例运行结果



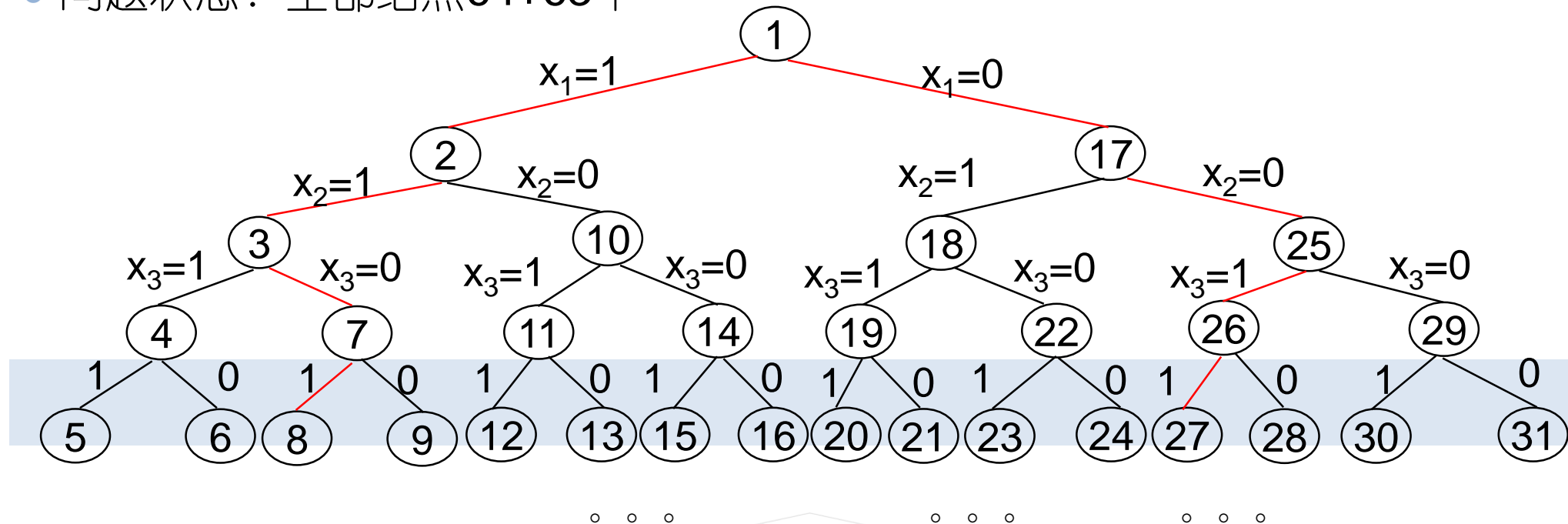
问题描述

- 子集和数问题：
 - 假定有 n 个不同的正数 $W(1:n)$ ，找出这些数中所有使得和为 M 的组合。
元素 $W(i)$ 称为权。
- 回溯法求解：
 - 用固定长的 n -元组 X 来表示，解向量元素 $X(i)$ 取1或0值，表示解中是否包含权数 $W(i)$ 。 $\sum W(i)X(i)=M, 1 \leq i \leq n$



• 6-元组表达的解空间树

- 解状态：叶结点 $2^6=64$ 个
- 部分解结点： $2^0+2^1+2^2+2^3+2^4+2^5=2^6-1=63$ 个
- 问题状态：全部结点 $64+63$ 个



从根结点到叶结点的一条路径确定解空间中的一个元组

限界函数

- 当满足条件：
$$\sum_{i=1}^k W(i) X(i) + \sum_{i=k+1}^n W(i) \geq M$$
 - $X(1), \dots, X(k)$ 能导致一个答案结点,
- 如果一开始 $W(i)$ 按非降次序排列, 那么当满足条件:

$$\sum_{i=1}^k W(i) X(i) + W(k+1) > M$$

- $X(1), \dots, X(k)$ 不能导致一个答案结点。

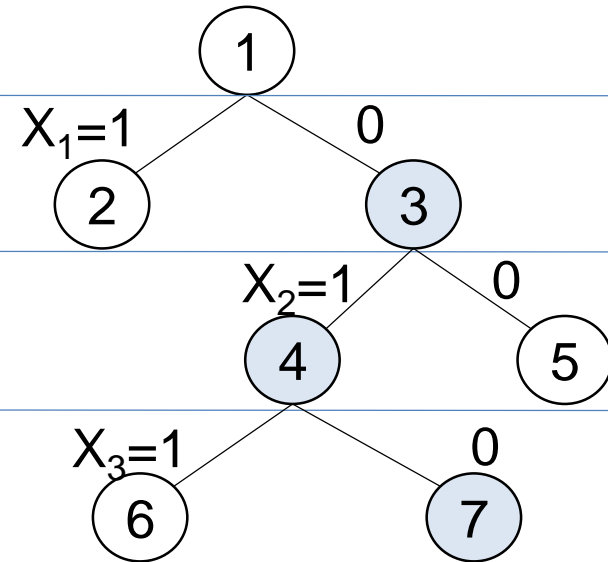
- 综上, 限界函数 $B_k(X(1), \dots, X(k)) = \text{true}$, 当且仅当:

$$\underbrace{\sum_{i=1}^k W(i) X(i)}_s + \underbrace{\sum_{i=k+1}^n W(i)}_r \geq M \quad \text{且} \quad \sum_{i=1}^k W(i) X(i) + W(k+1) \leq M$$

效率估计

- $n=6, M=30, W=(5, 10, 12, 13, 15, 18)$

结点编号	s	r	$W(i+1)$	B值	
2	5	68	10	T	1个
3	0	68	10	T	2个
4	10	58	12	T	
5	0	58	12	T	2个
6	22	46	13	F	
7	10	46	13	T	1个



$$\sum_{i=1}^k W(i) X(i) + \sum_{i=k+1}^n W(i) \geq M$$

s r

$$\sum_{i=1}^k W(i) X(i) + W(k+1) \leq M$$

效率估计

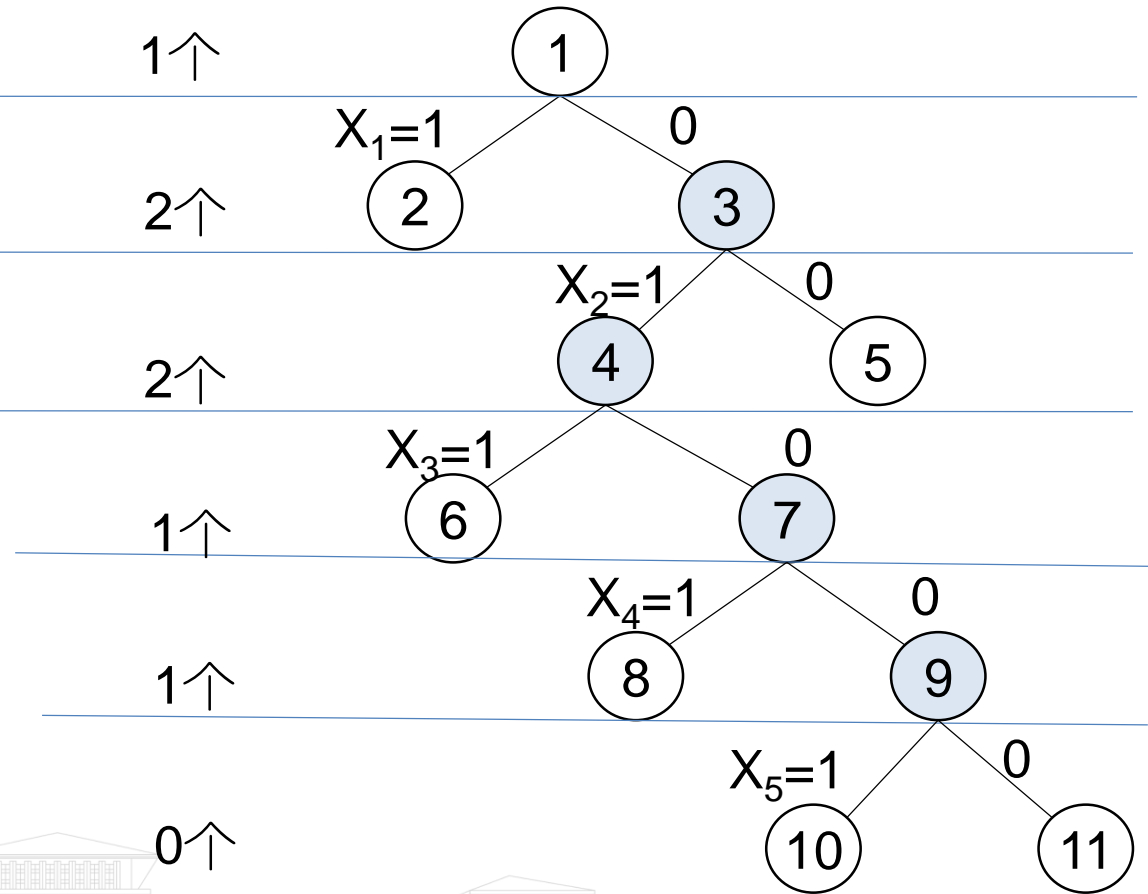
- $n=6, M=30, W=(5, 10, 12, 13, 15, 18)$

结点编号	s	r	$W(i+1)$	B值
8	23	33	15	F
9	10	33	15	T
10	25	18	18	F
11	10	18	18	F

$$\sum_{i=1}^k W(i) X(i) + \sum_{i=k+1}^n W(i) \geq M$$

s r

$$\sum_{i=1}^k W(i) X(i) + W(k+1) \leq M$$



不受限界结点的估计数: $m=1+2+2*2+2*2*1+2*2*1*1+0=15$

算法7.6 子集和数的递归回溯算法

Procedure SUMOFSUB(s,k,r)

//找出 $W(1:n)$ 中和数为 M 的所有子集。 $s = \sum_{j=1}^{k-1} W(i) X(i)$ 且 $r = \sum_{j=k}^n W(j)$
//进入此过程时 $X(1), \dots, X(k-1)$ 的值已确定。

//这些 $W(j)$ 按非降次序排列。 $W(1) \leq M, \sum_{i=1}^n W(i) \geq M$

integer $W(1:n), M, n;$

boolean $X(1:n)$

integer s, k, r



//生成左儿子。由于 $B_{k-1}=\text{true}$ ，因此 $s+W(k)\leq M$ 且 $s+r \geq M$

$X(k) \leftarrow 1$

if $s+W(k) = M$

then print(X)

else if $s+W(k)+W(k+1) \leq M$ // $B_k=\text{true}$

then call SUBOFSUB($s+W(k)$, $k+1$, $r-W(k)$)

endif

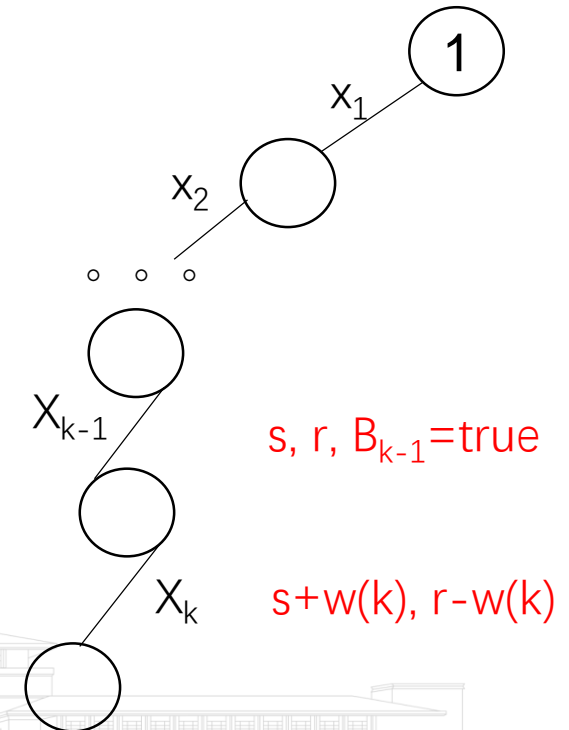
endif

$$\sum_{i=1}^k W(i) X(i) + \sum_{i=k+1}^n W(i) \geq M$$

$$\sum_{i=1}^k W(i) X(i) + W(k+1) \leq M$$

$$s = \sum_{j=1}^{k-1} W(j) X(j) \text{ 且 } r = \sum_{j=k}^n W(j)$$

左子树-递归入口



//生成右儿子和计算 B_k 的值

If $s+r-W(k) \geq M$ and $s+W(k+1) \leq M$

then $X(k) \leftarrow 0$; call SUMOFSUB($s, k+1, r-W(k)$)

endif

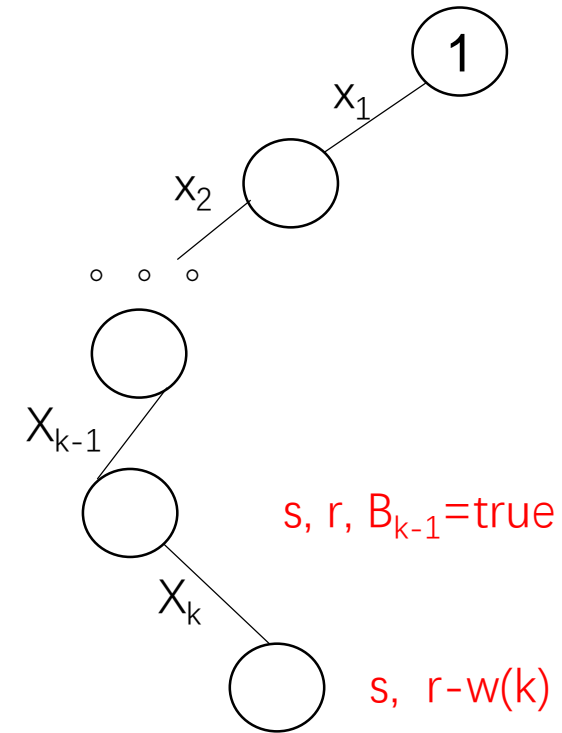
end SUMOFSUB

右子树-递归入口

$$s = \sum_{i=1}^{k-1} W(i) X(i) \text{ 且 } r = \sum_{j=k}^n W(j)$$

$$\sum_{i=1}^k W(i) X(i) + \sum_{i=k+1}^n W(i) \geq M$$

$$\sum_{i=1}^k W(i) X(i) + W(k+1) \leq M$$



思考：如果不将 W 预排序，算法怎样设计？算法效率怎样变化？

实例

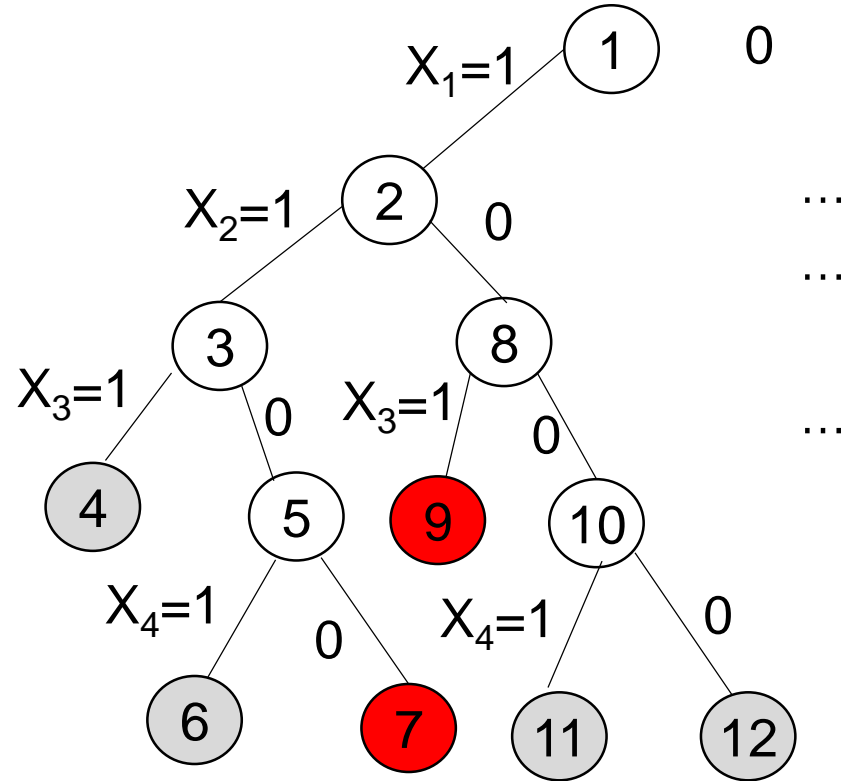
- $n=6, M=30, W=(5, 10, 12, 13, 15, 18)$
- 使用限界函数前，状态空间树中所有结点都会被访问到，叶结点(解状态)个数为 $2^6=64$ 个，部分解结点63个。
- 使用限界函数后，动态树一共生成33个结点。



$M=30, W=(5,10,12,13,15,18)$

$$s+W(i+1) \leq M \text{ 且 } s+r \geq M$$

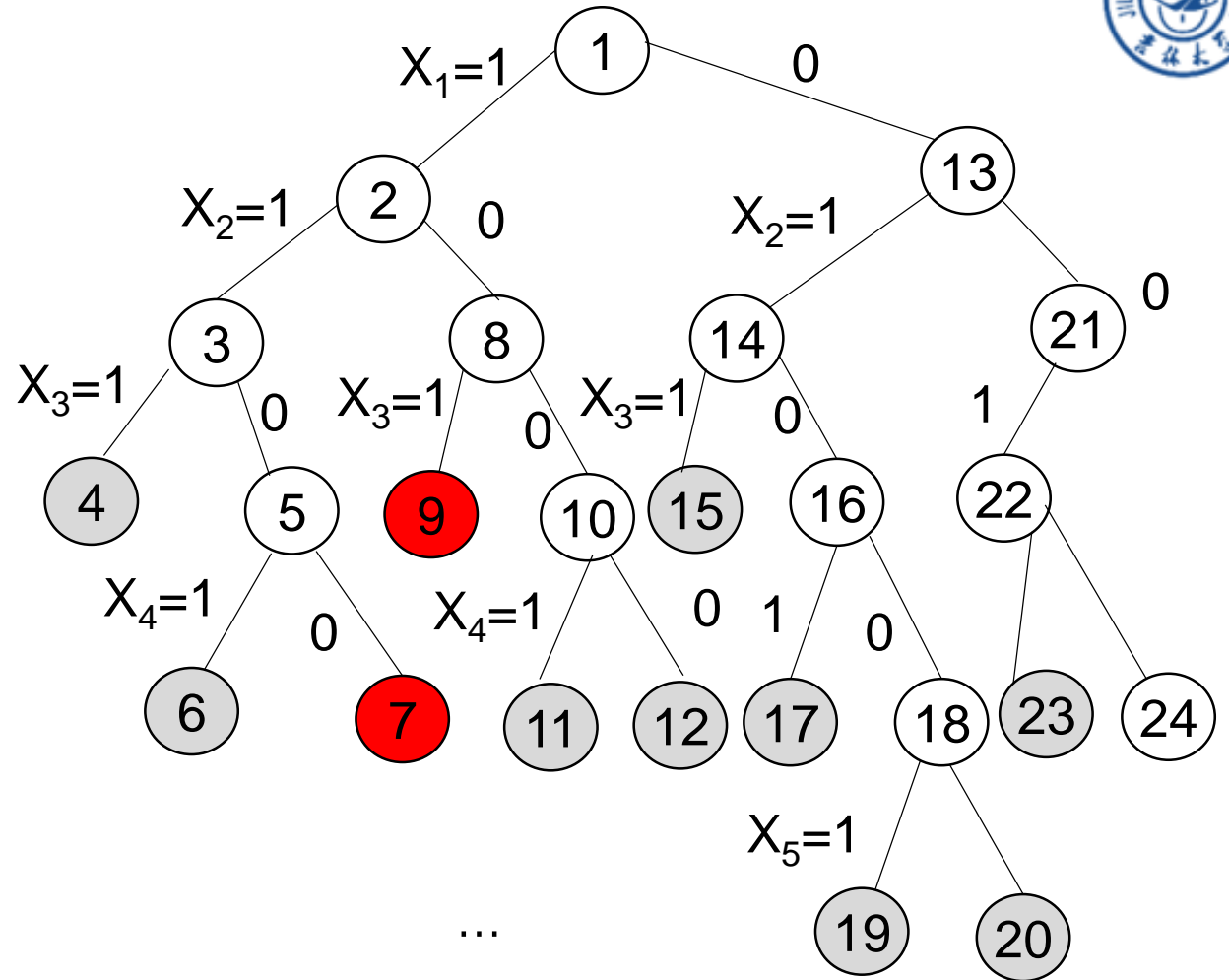
结点编号	s	r	$W(i+1)$	B值
2	5	68	10	T
3	15	58	12	T
4	27	46	13	F
5	15	46	13	T
6	28	33	15	F
7	15	33	15	T
8	5	58	12	T
9	17	46	13	T
10	5	46	13	T
11	18	33	15	F
12	20	33	15	F
13	10	46	13	T



$M=30, W=(5,10,12,13,15,18)$

$$s+W(i+1) \leq M \text{ 且 } s+r \geq M$$

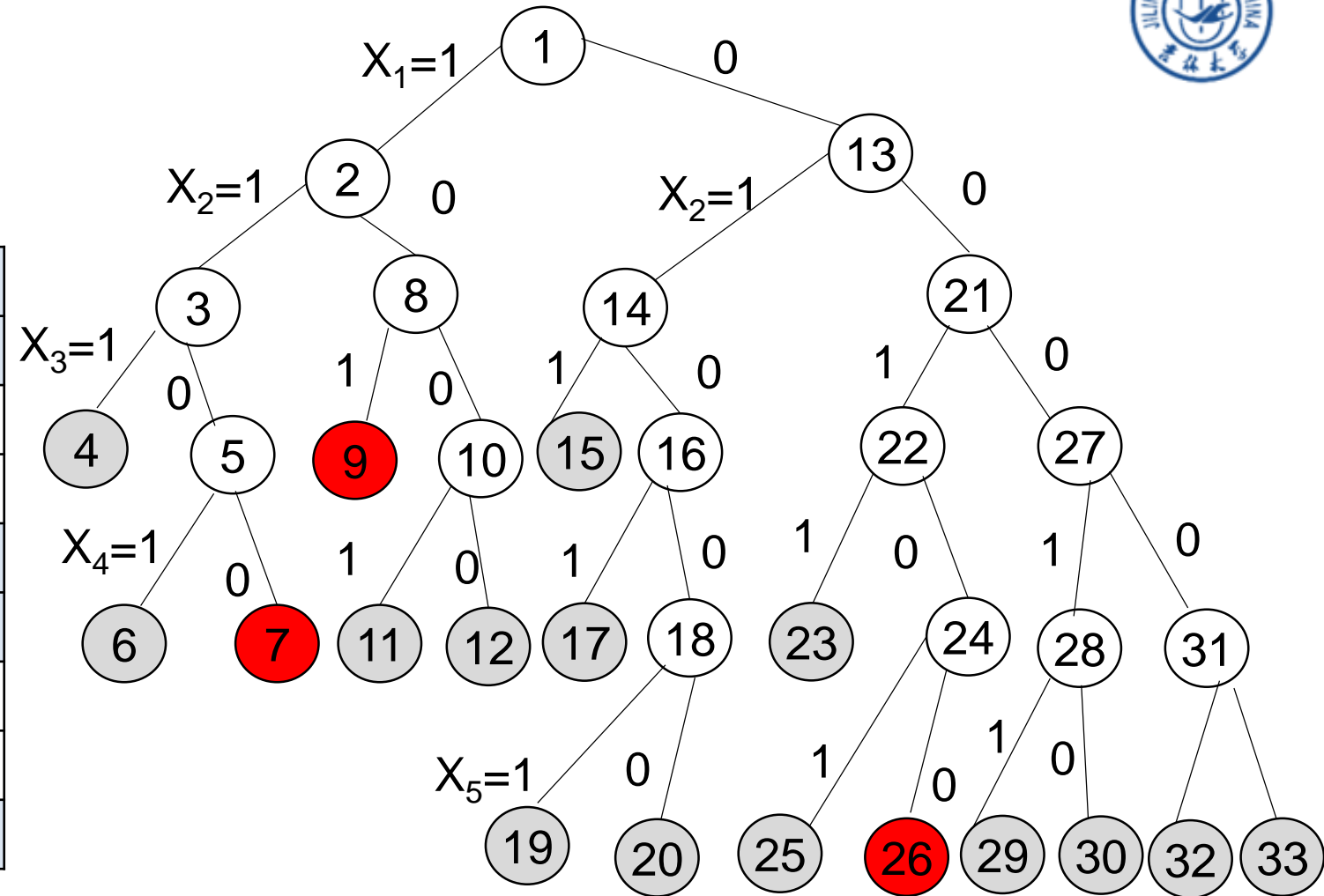
结点编号	s	r	$W(i+1)$	B值
13	0	68	10	T
14	10	58	12	T
15	22	46	13	F
16	10	46	13	T
17	23	33	15	F
18	10	33	15	T
19	25	18	18	F
20	5	18	18	F
21	0	58	12	T
22	12	46	13	T
23	25	33	15	F
24	12	33	15	T



$M=30, W=(5,10,12,13,15,18)$

$$s+W(i+1) \leq M \text{ 且 } s+r \geq M$$

结点编号	s	r	$W(i+1)$	B值
25	27	18	18	F
26	12	18	18	T
27	0	46	13	T
28	13	33	15	T
29	28	18	18	F
30	13	18	18	F
31	0	33	15	T
32	15	18	18	F
33	0	18	18	F



7.5 图着色问题

- 问题描述
- 图的 m -着色判定问题
- 解空间树
- 回溯算法
- 实例分析



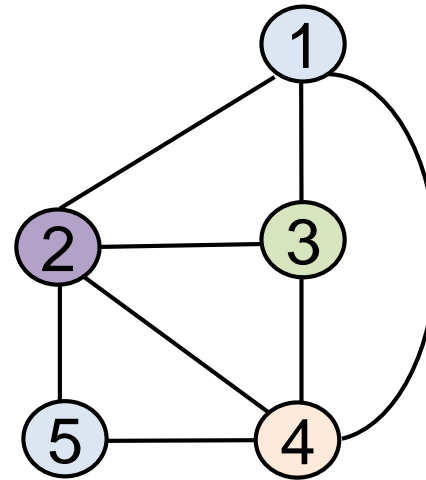
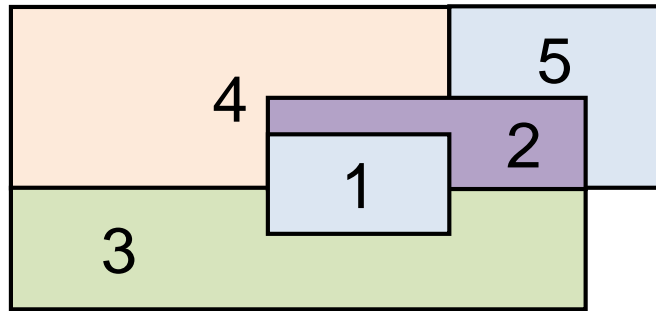
问题描述

- 图着色问题(Graph Coloring Problem, GCP) 又称着色问题, 是最著名的NP-完全问题之一。
- 数学定义: 给定无向连通图 $G=(V,E)$, 其中 V 为顶点集合, E 为边集合, 用不同的颜色给图中顶点着色, 要求任何两个相邻顶点的着色不同。
- 问: 最少需要多少种颜色?



图的m-着色判定问题

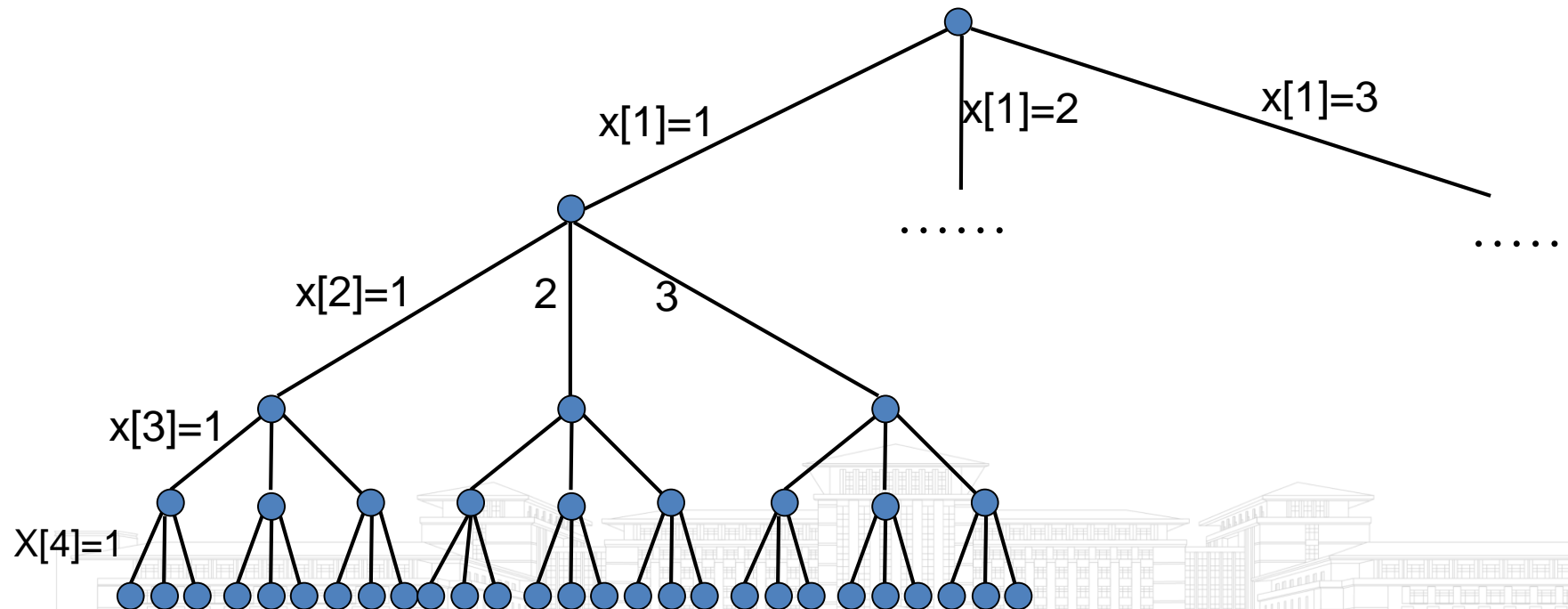
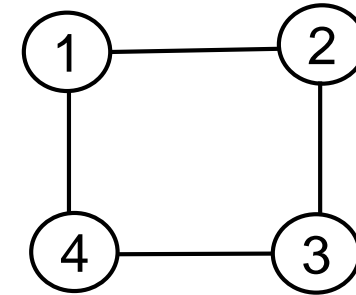
- 给定无向连通图 $G=(V,E)$ ，其中 V 为顶点集合， E 为边集合，用 m 种不同颜色给图中顶点着色，问：是否存在任何两个相邻顶点颜色不同的着色方案？
- 本节用回溯来解决图的 m 着色判定问题，如果判定答案为“是”，要求给出着色方案。



解空间树

考虑 $n=4$ (4个顶点)的连通图, $m=3$ (3种颜色)

- n 元组表示: $X = (x_1, \dots, x_4)$, x_i 表示结点 i 的颜色。
- 显式: $1 \leq x_i \leq 3$ 。
- 隐式: 若结点 i 和 j 之间有边存在, 则 $x_i \neq x_j$ 。



算法7.7 回溯法求解图着色判定问题

Procedure M Coloring(V, E, C, n, m)

// 图 $G=(V, E)$, n 个顶点, m 种颜色

$C(1:n) \leftarrow 0$; $k \leftarrow 1$ // C 记录着色序列, 从第一个顶点开始

while ($k \geq 1$) **$k=0$ 表示整颗树遍历完毕**

$C(k) \leftarrow C(k) + 1$ **$C(k)$ 当前颜色发生冲突**

while (not OK(k) and $C(k) \leq m$) do $C(k) \leftarrow C(k) + 1$ repeat

if $C(k) \leq m$ then

if $k=n$ then print(C); return **true** //全部着色, 打印

else $k \leftarrow k+1$; $C(k) \leftarrow 0$ //准备为下一个顶点着色

endif

else $k \leftarrow k-1$ //顶点 k 无法着色, 回溯

endif

Repeat

return **false**

END M Coloring



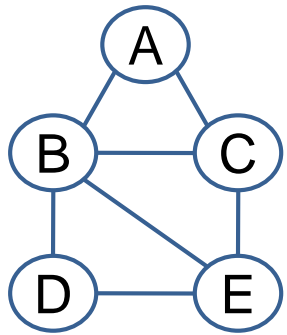
算法7.8 判断顶点k的着色是否合法

```
procedure OK( k )  
  int i , k  
  i  $\leftarrow$  1  
  while (i<k) do  
    if (i和k之间有边存在 and C(i)=C(k))  
      then return false  
    endif  
    i  $\leftarrow$  i+1  
  repeat  
  return true  
end OK
```



实例分析

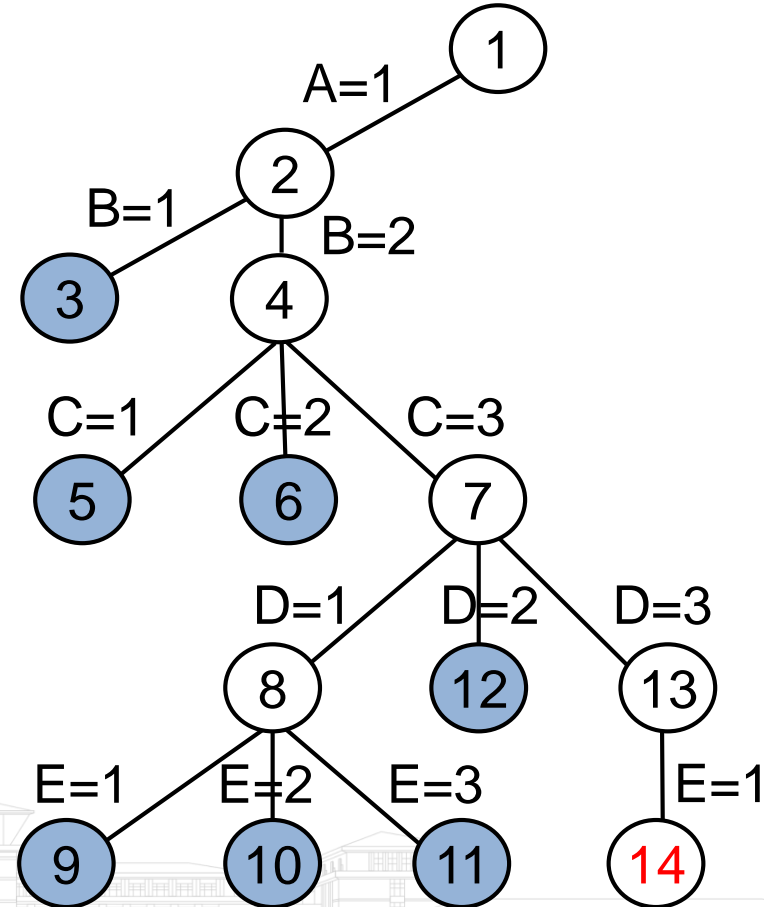
(a) 一个无向图



$n=5$ 个顶点的无向图， $m=3$ ，对应的完全状态空间树是完全 m 叉树，最后一层有多少个叶子结点？

$$m^n$$

(b) 回溯法搜索空间



7.6 小结

- 回溯法适用的问题
 - 多阶段决策问题/组合问题满足多米诺性质
- 回溯法的设计思想概述
 - 确定解向量： n -元组/ k -元组
 - 分解约束条件：显示&隐式
 - 确定解空间树
 - 设计限界函数 B
 - 深度优先方式搜索树



- 解空间树的分类
 - 集合树：问题的解是对已知集合元素的取舍
 - 如子集和数问题，0/1背包问题
 - 排列树：问题的解是对已知集合元素的排列
 - 如n-皇后问题，图着色判定问题
- 回溯法的效率问题
 - 解空间树的大小：决定最坏情况
 - 限界函数**B**的剪枝能力：决定动态树
 - 求问题全部的解时，可以用蒙特卡洛方法估计算法效率



- n-皇后问题
- 子集和数问题
- 图的着色问题

能够识别出适合回溯法的可计算性问题、独立设计算法和分析算法复杂度。





本章结束

