吉林大学

第七章 回溯法

目录

7.1 回溯的一般方法

- 方法适用的问题特点
- 多米诺性质
- 解空间树
- 解空间树的例子
- 动态树中的结点
- 问题状态的生成
- 回溯法的设计思想
- 回溯法的非递归算法描述
- 回溯法的递归算法描述

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 ,$1 \le i \le n$, n表示问题规模
    - 固定长n-元组$(x_1, \ldots, x_n)$
    - 不定长k-元组$(x_1, \ldots, x_k)$, $k \le n$
  - 问题满足多米诺性质
  - 问题寻找满足约束条件的一个解或所有解；有时则要寻找最优解，此时问题中还存在目标函数

组合搜索

组合优化

多米诺性质

- 多米诺性质
  - 设$P(x_1,\ldots,x_i)$是关于向量$(x_1,\ldots,x_i)$的某种性质的判定，当$P(x_1,\ldots,x_{i+1})$为真时，一定有$P(x_1,\ldots,x_i)$为真，$0<i<n$
- 根据多米诺性质，如果$P(x_1,\ldots,x_i)$不成立，则$P(x_1,\ldots,x_{i+1})$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$(x_1,\ldots,x_i)$，$0<i<n$，一旦发现不成立，则无需考虑后续$x_{i+1},\ldots,x_n$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$(x_1,\ldots,x_n)$, $x_i \in S_i$,
- 硬性处理
  - $|S_i|=m_i$,向量个数$m=m_1 \times m_2 \times \ldots \times m_n$,对这m个n-元组逐一检测是否满足$B(x_1,\ldots,x_n)$，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$(x_1,\ldots,x_i)$，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$x_{i+1},\ldots,x_n$的取值，从而减少了$m_{i+1} \times \ldots \times m_n$个向量。

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

- 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
- 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
- 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X：由根到结点X的路径确定了解空间中的一个元组
  - 答案状态X: 由根到解状态结点X的路径确定了问题的一个解。

对应限界函数

满足显式约束条件

还满足隐式约束条件

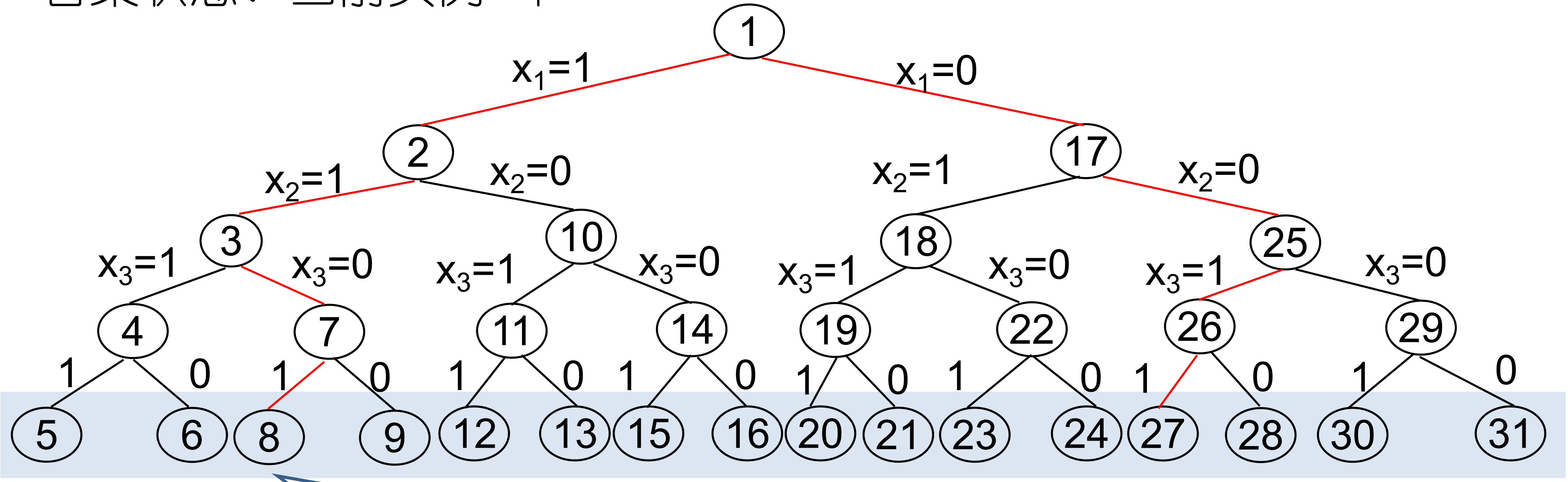
解空间树的例子

- 子集和问题：已知n+1个正数：$w_i(1≤i≤n)$和M, 要求找出w_i的和是M的所有子集。
- 回溯法准备工作
  - 确定元组表达形式
    - 固定长元组：用大小固定的n-元组$(x_1, \ldots, x_n)$表示
    - 不定长元组：用大小可变的k-元组$(x_1, \ldots, x_k)$表示, $k \le n$
  - 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

固定长元组表达

- $n=4$, $(w_1, w_2, w_3, w_4) = (11,13,24,7)$, $M=31$。
  - 可行解1： 11,13,7
  - 可行解2： 24, 7
- 4-元组$(x_1, \dots , x_n)$
  - 可行解1： (1,1,0,1)
  - 可行解2： (0,0,1,1)
- 显式约束： $x_i \in \{0,1\}, 1 \le i \le n$; 如果选择w_i , 则$x_i=1$; 否则$x_i=0$
- 隐式约束： $\sum w_i x_i = M$， $1 \le i \le n$
- 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$2^n=2^4=16$个元组

- 子集和问题的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



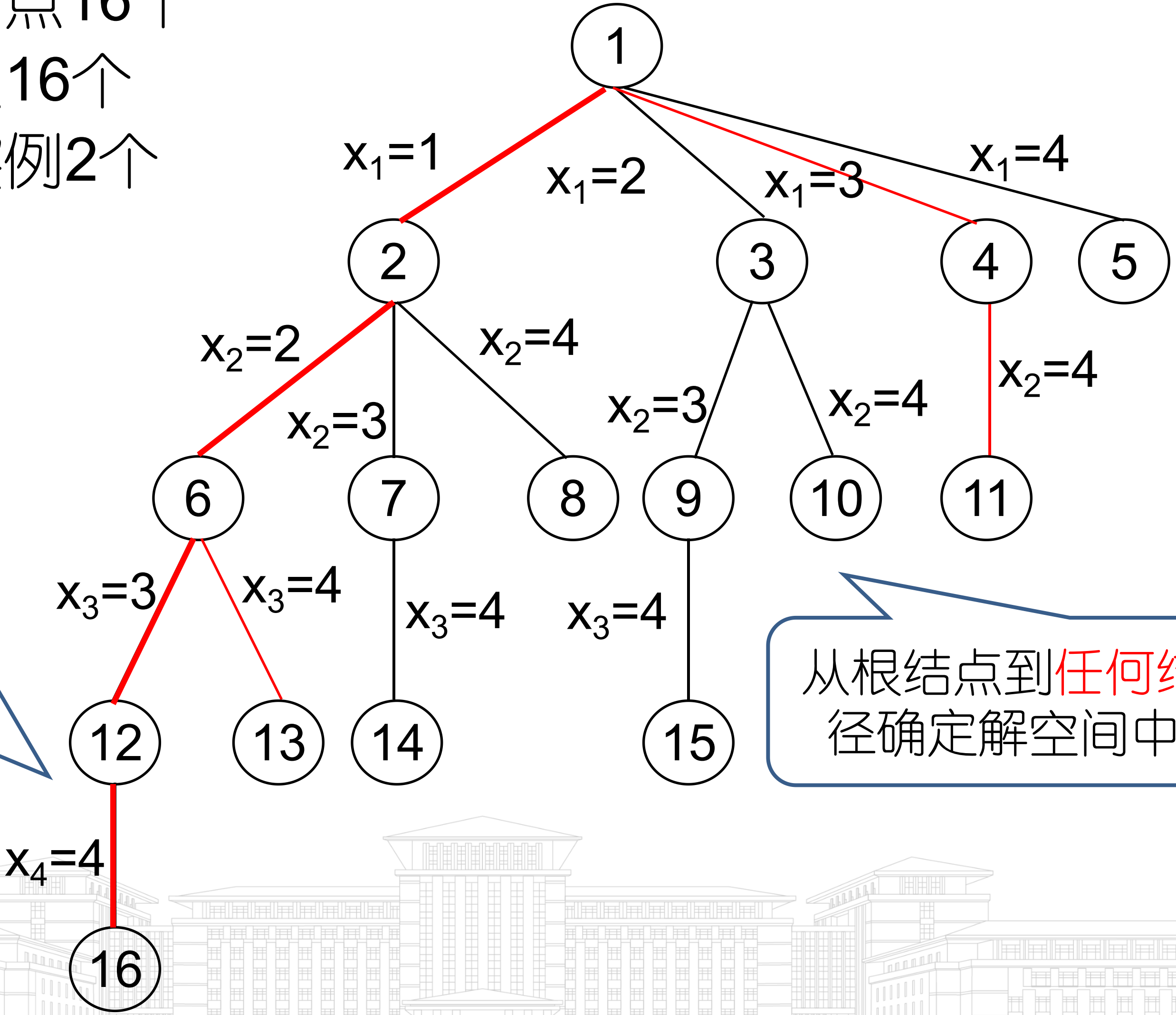
从根结点到叶结点的一条路径确定解空间中的一个元组

可变长元组表达

- $n=4$, $(w_1, w_2, w_3, w_4) = (11,13,24,7)$, $M=31$。
  - 可行解1： 11,13,7
  - 可行解2： 24, 7
- k-元组$(x_1, \ldots, x_k)$
  - 可行解1： (1,2,4)
  - 可行解2： (3,4)
- 显式约束： $x_i \in \{ j \mid j$是w_j的下标值, $1 \le j \le n \}, 1 \le i \le k$
- 隐式约束： 没有两个x_i是相同的, 且相应的w_i的和等于M, $x_i \le x_{i+1}$, $1 \le i < k$
- 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$2^n=2^4=16$个元组

避免重复情况,如(1,2,4)和(1,4,2)

- 子集和问题的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



空向量
(1)
(1,2)
(1,2,3)
(1,2,3,4)

从根结点到任何结点的一条路径确定解空间中的一个元组

动态树中的结点

- 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。

问题状态的生成

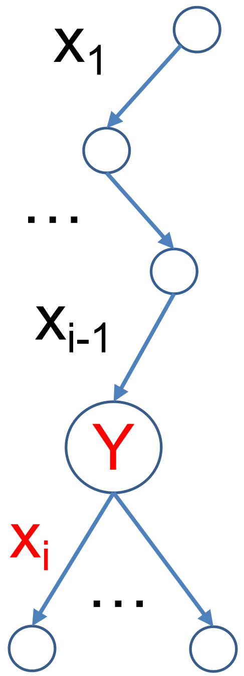
- 第一种状态生成方法
  - 当前的E-结点R一旦生成一个新的儿子结点C，这个C结点就变成一个新的E-结点，当检测完了子树C后，R结点就再次成为E-结点，生成下一个儿子结点。
  - 该方法也称为深度优先生成法，对应回溯法。
- 第二种状态生成方法:
  - 一个E-结点一直保持到变成死结点为止。
  - 当活结点用队列保存时，该方法也称为宽度优先生成法，对应分支限界法。
  - 当活结点用栈保存时，该方法也称为D-检索生成法

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
- $(x_1,x_2,\ldots,x_{i-1})$是状态空间树中由根出发的一条路径，到达结点Y
- $T(x_1,\ldots x_{i-1})$是元素x_i的集合，对于每一个x_i，$(x_1,x_2,\ldots,x_{i-1},x_i)$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$(x_1,x_2,\ldots,x_{i-1},x_i)$不可能延伸到一个答案结点，则$B_i(x_1,x_2,\ldots,x_i)$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)∈T(X(1)…X(k-1))
                and B(X(1)…X(k))=TRUE)
        then if (X(1) …X(k))是一条抵达答案结点的路径)
                then print ( X(1)…X(k))
                endif
                k ← k+1
        else   k ← k-1
        endif
    repeat
end  BACKTRACK
```

算法7.2 回溯法的递归算法描述

进入算法时，解向量X中的前k-1个分量X(1) …X(k-1)已经被赋值

```
procedure  RBACKTRACK(k)
global   X(1:n);
int  k, n;
for  ( 满足下式的每个X(k), X(k) ∈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
```

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？

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

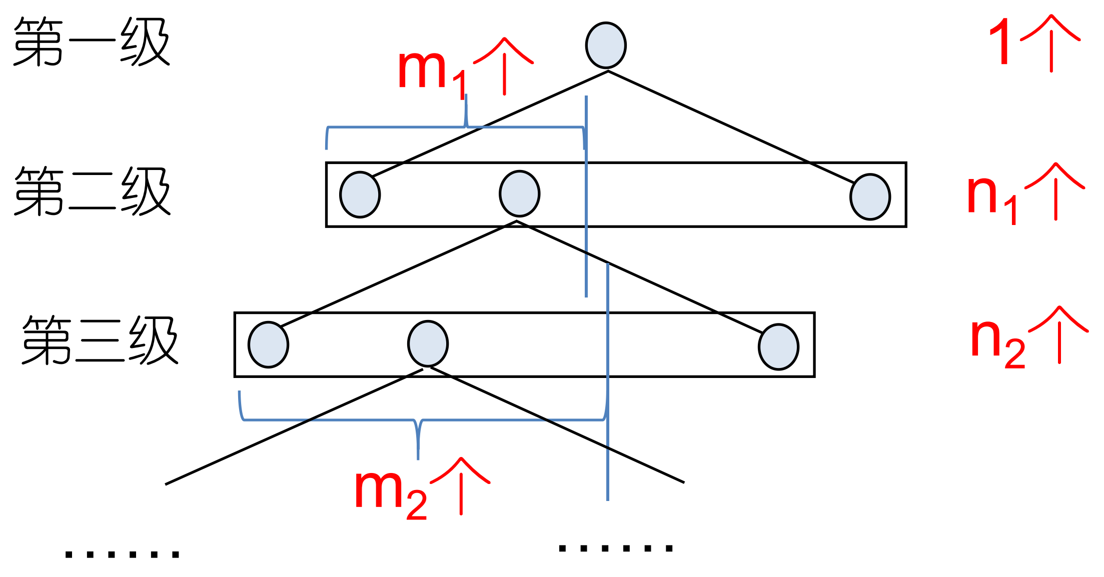
- 由于回溯法对同一问题不同实例的巨大差异，在n很大时，对某些实例是十分有效的。因此，在采用回溯法计算某个实例之前，应估算其工作能效
- 用回溯算法处理一棵树所要生成的结点数，可以用蒙特卡罗方法估算出来

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

- 假定界限函数是固定的
  - 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: $m=1+m_1+m_1*m_2+m_1*m_2*m_3+\ldots$
m_i表示第i级结点平均没受限界的儿子结点数。



第一级通过B函数检验的结点个数有多少个？ 1个

第二级通过B函数检验的结点个数有多少个？ m_1个

从第二级中随机选中一个结点，它的子结点共有m_2个满足B函数检验，推断第三级通过B函数检验的结点个数一共有多少个？ m_1 m_2个

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 ←1;  r ←1;  k ←1
loop
    T_k ←{X(k):X(k)∈T(X(1), …,X(k-1)) and B_k (X(1), …,X(k))}
    if  SIZE(T_k)=0 then exit endif
    r ←r*SIZE(T_k)  //第k级的结点总数
    m ←m+r          //前k级的结点总数
    X(k) ←CHOOSE(T_k) //从T_k中随机地挑选一个元素
   k ←k+1
repeat
return m
end ESTIMATE
```

蒙特卡罗方法的特点

- 优点
  - 适用于找到所有答案结点的情况
  - 限界函数固定不变，计算方便，对树中同一级结点均适用
- 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

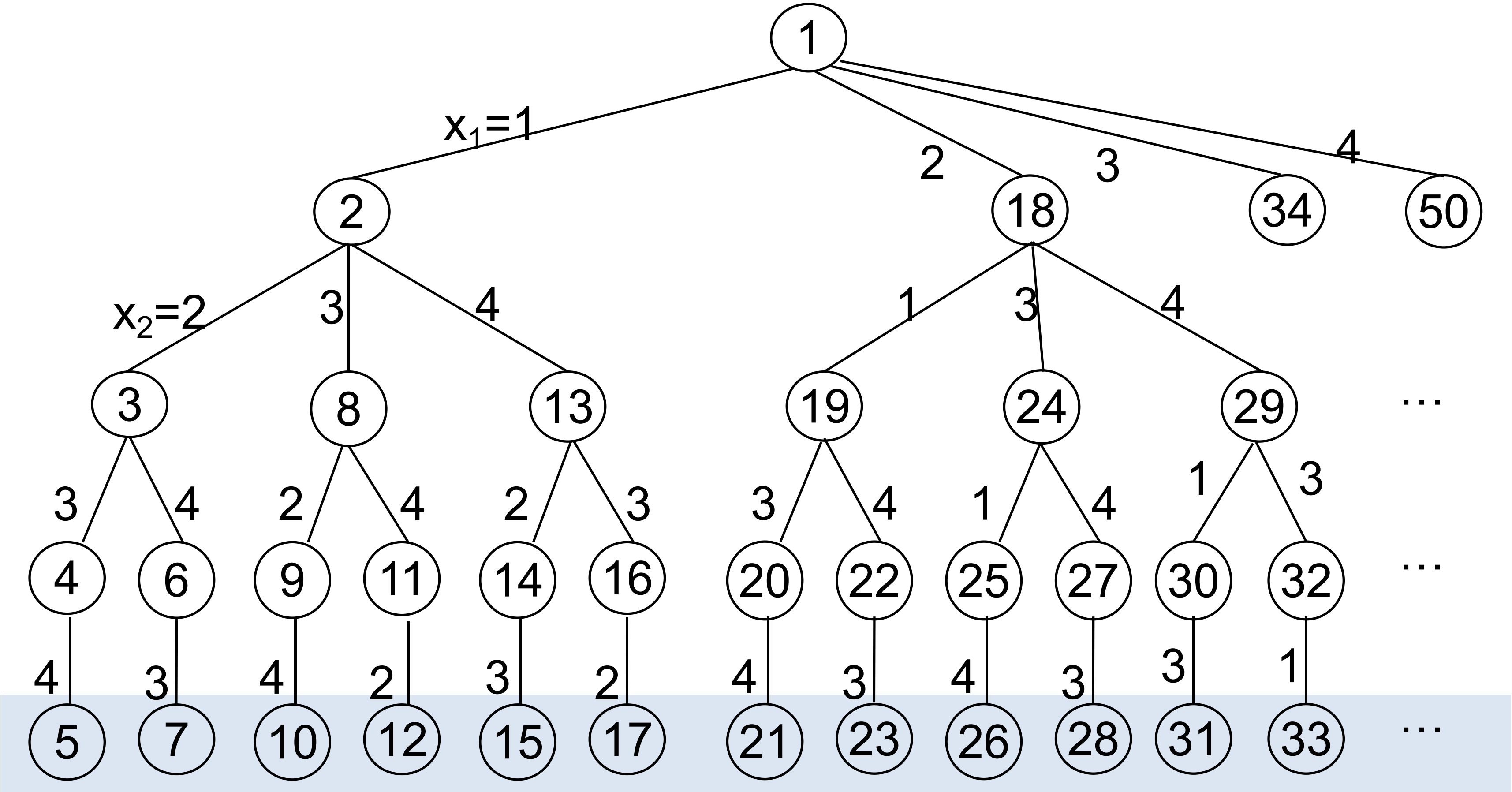
- n-皇后问题：
  - 在一个n*n棋盘上放n个皇后，使每两个皇后之间都不能互相“攻击”，即使得每两个皇后都不能在同一行、同一列及同一条斜角线上。
- 基于回溯法求解：
  - n-元组$(x_1,\ldots x_n)$：表示皇后i放在i行x_i列上。
  - 显式约束条件: $x_i \in \{1, 2, \ldots, n\}, 1 \le i \le n$
  - 隐式约束条件: 没有两个x_i可以相同，且没有两个皇后可以在同一条斜角线上。

解空间：n^n

与问题实例无关，
解空间：n!

解空间树

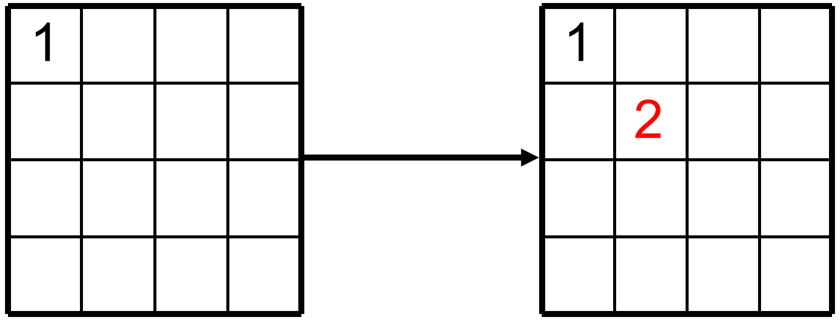
- n=4时，叶结点个数=4！=24，解空间是从根结点到叶结点的所有路径。

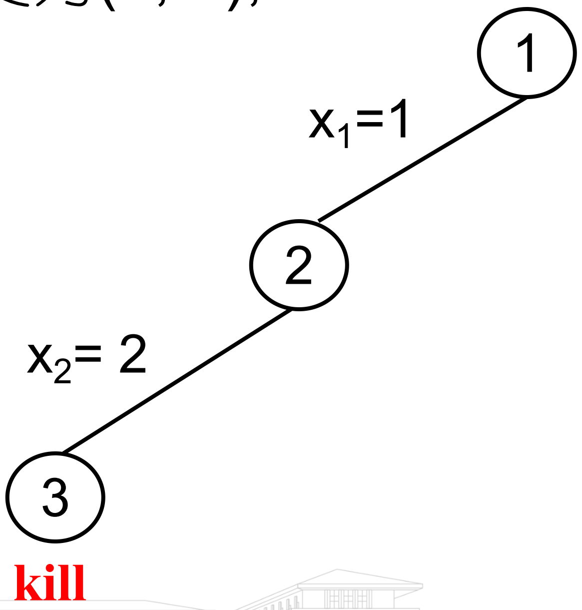


问题分析

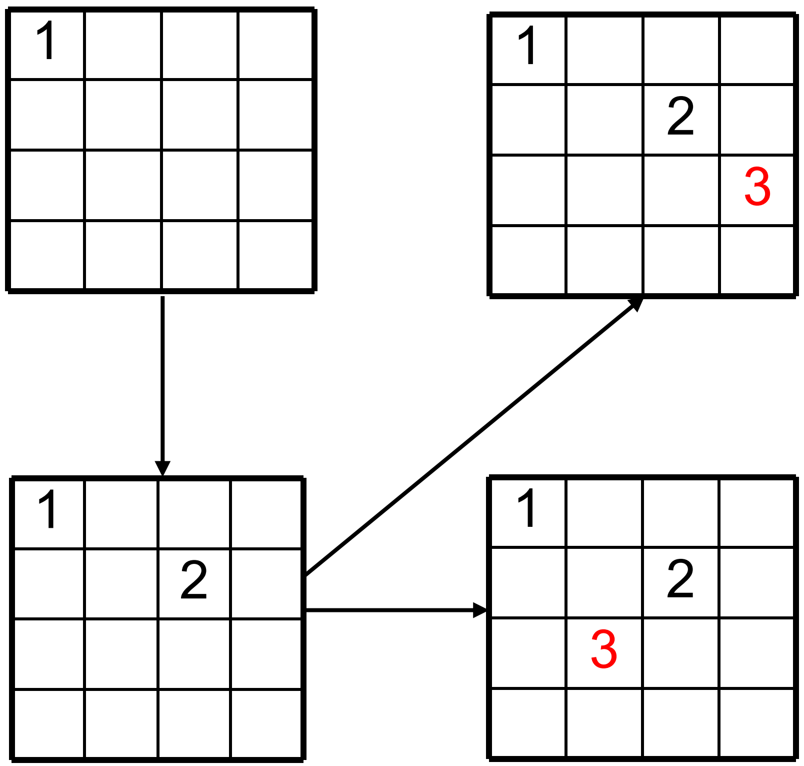
- 开始把根结点作为唯一的活结点，根结点就成为E-结点而且路径为( ); 接着生成儿子结点，那么结点2被生成，这条路径为(1)，即把皇后1放在第1列上。

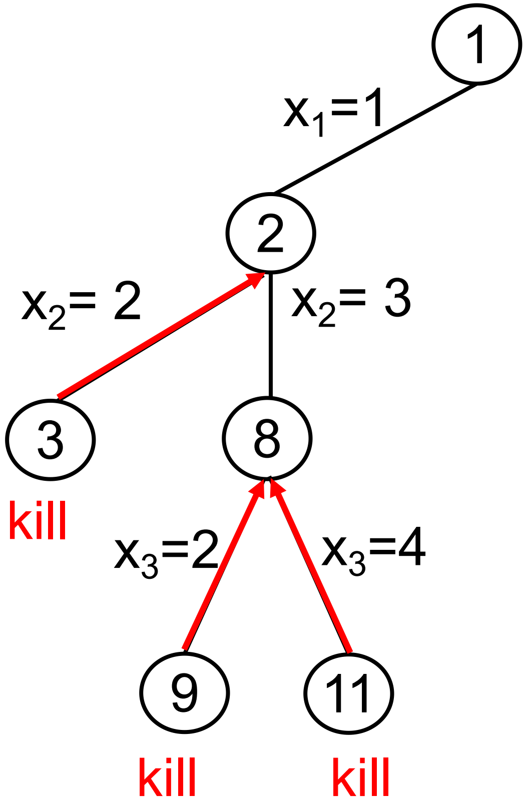
- 结点2变成E-结点，它再生成结点3，路径变为(1, 2)，结点3被杀死，此时回溯。



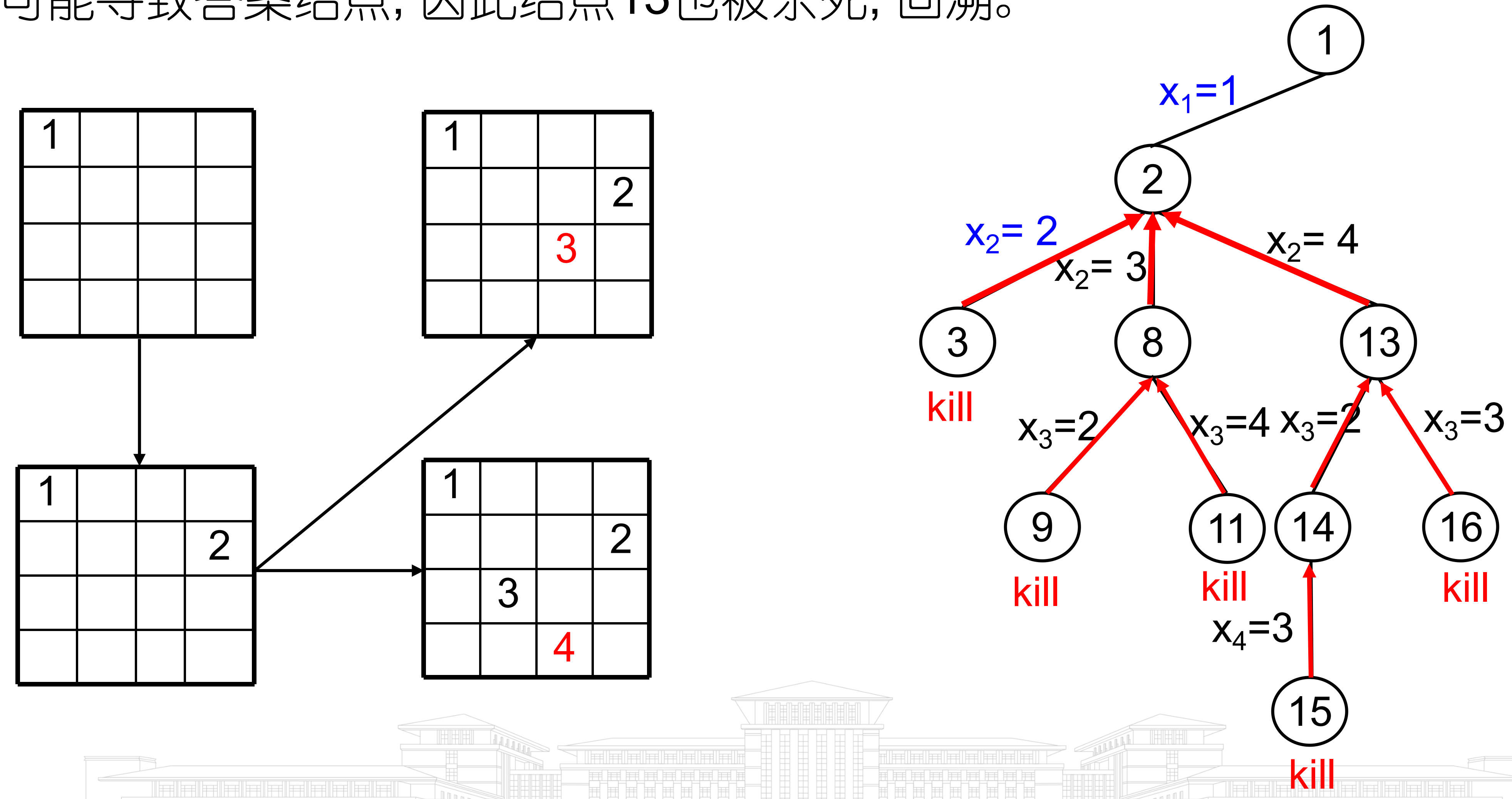


- 回溯到结点2生成结点8, 路径变为(1, 3), 则结点8成为E-结点, 它生成结点9和结点11都会被杀死, 所以结点8也被杀死, 应回溯。



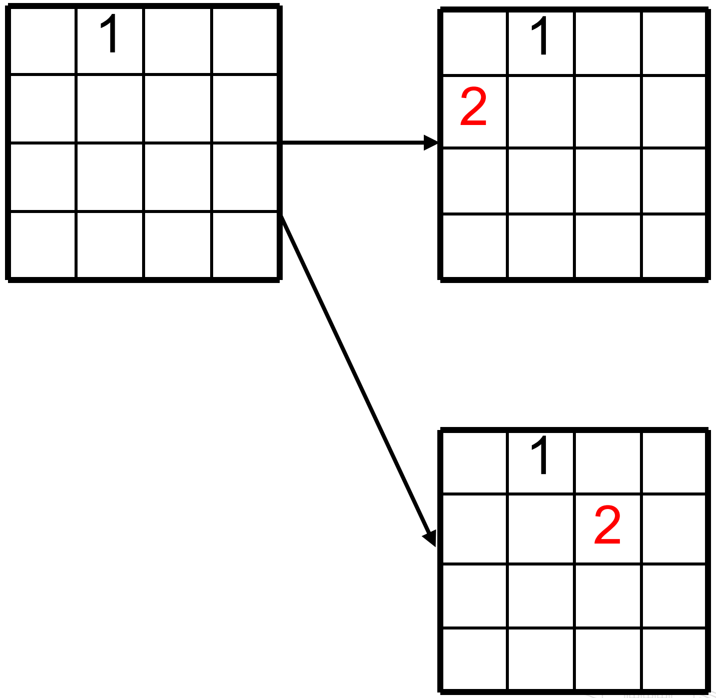


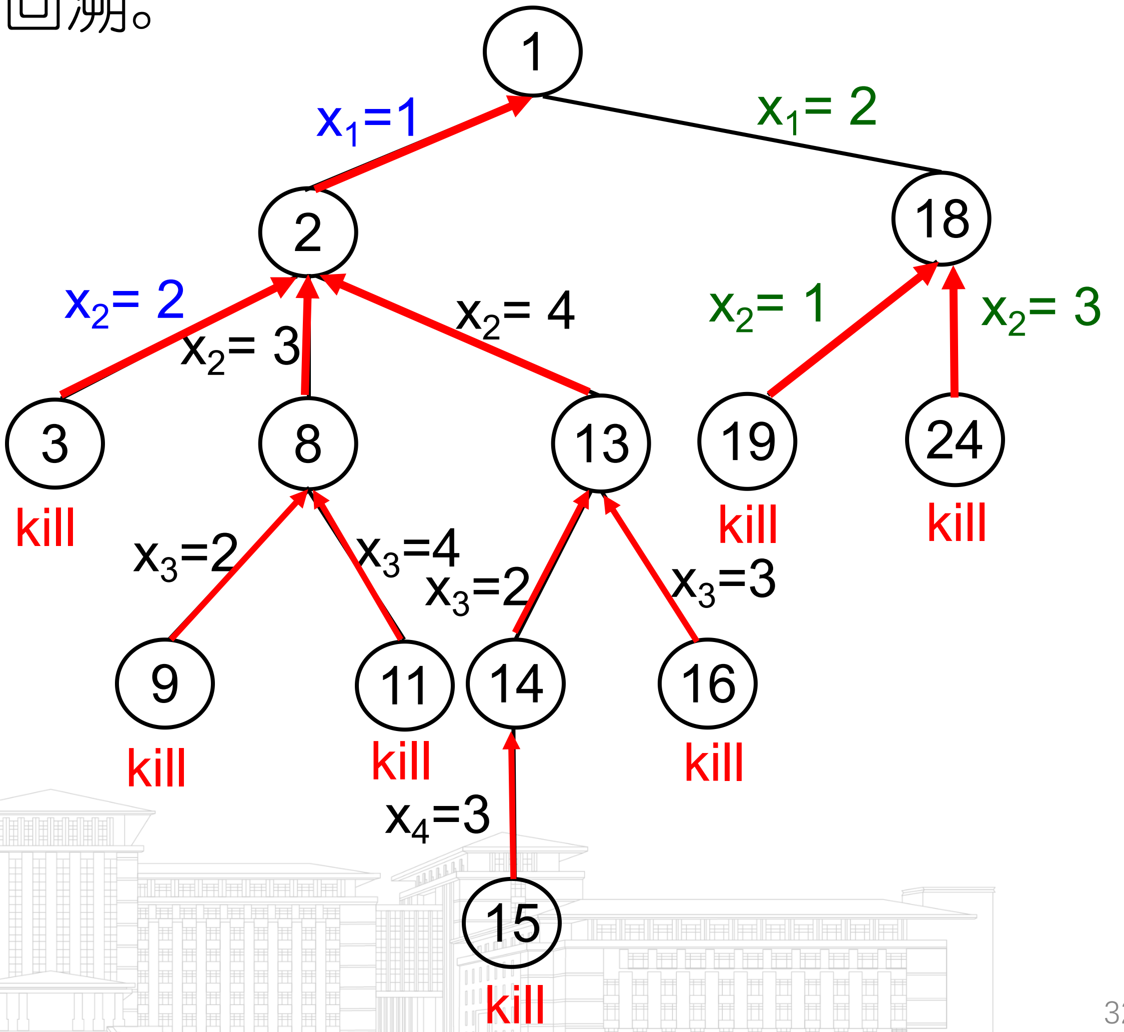
- 回溯到结点2生成结点13，路径变为(1, 4)，结点13成为E-结点，它的儿子不可能导致答案结点，因此结点13也被杀死，回溯。



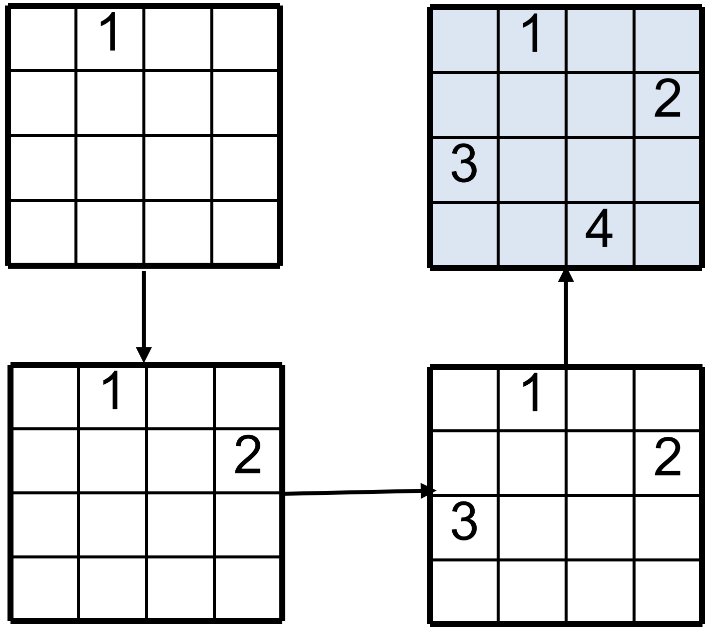
- 结点2的所有儿子都不能导致答案棋盘格局，因此结点2也被杀死；再回溯到结点1生成结点18，路径变为(2)。

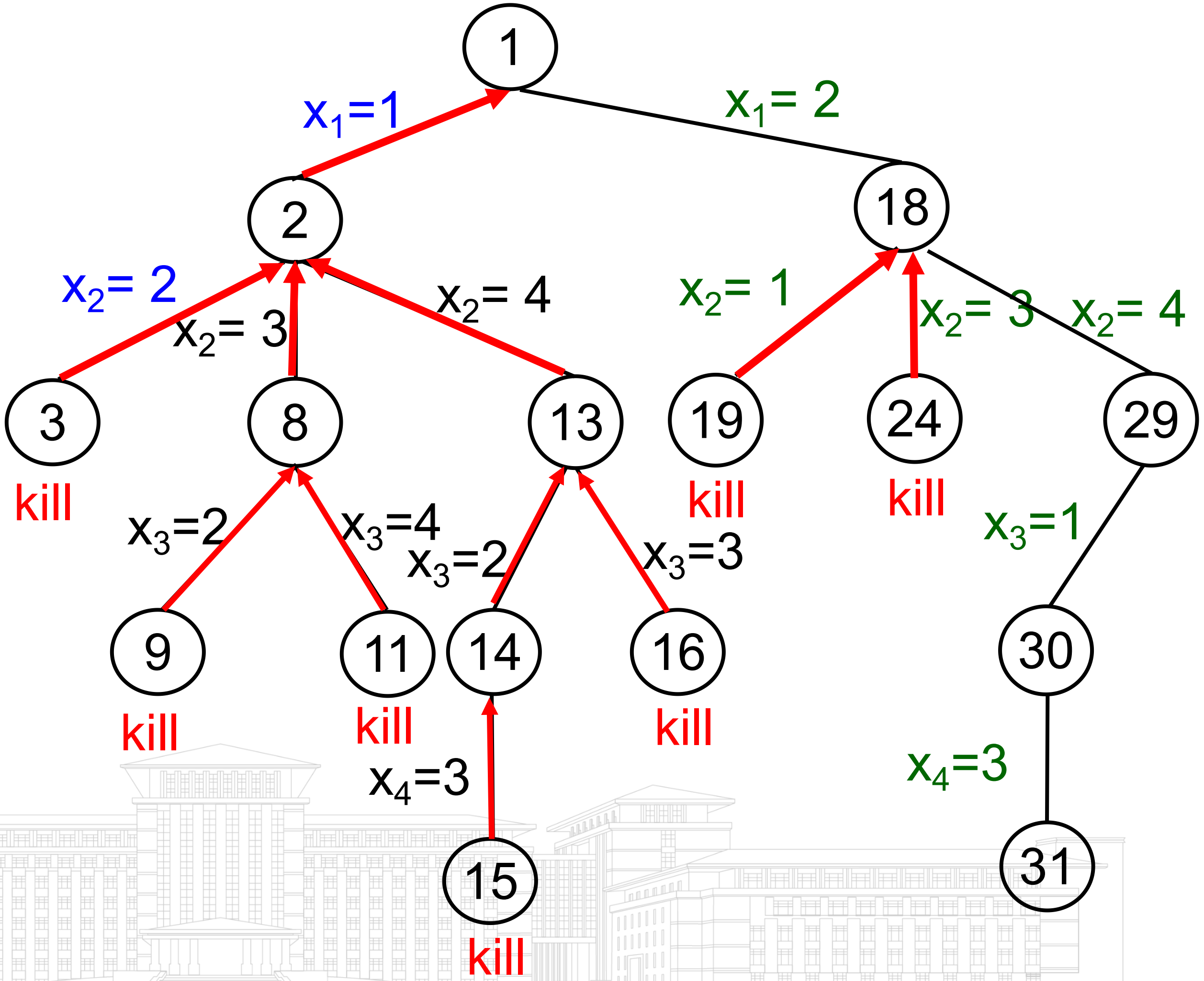
- 结点18的儿子结点19、结点24被杀死，回溯。





- 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，$(x_1,x_2,..x_n)$表示一个解，x_i表示第i个皇后放在第i行的列数。

- 同一条斜角线上的每个元素
  - 由左到右具有相同的“行-列”值;
  - 由右到左具有相同的“行+列”值。

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

- 设有两个皇后位于(i,X(i))和(k,X(k))位置上

$$\left.\begin{array}{l} i-X(i)=k-X(k) \\ i+X(i)=k+X(k) \end{array}\right\} \Longrightarrow |X(i)-X(k)|=|i-k|$$

// 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？

- 限界函数PLACE(k)的设计思想：
  - 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是全局数组, 进入此过程时已置入了前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))
       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
while (k>0) do
    X(k) ← X(k)+1
    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)
        then  if(k=n)
                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
```

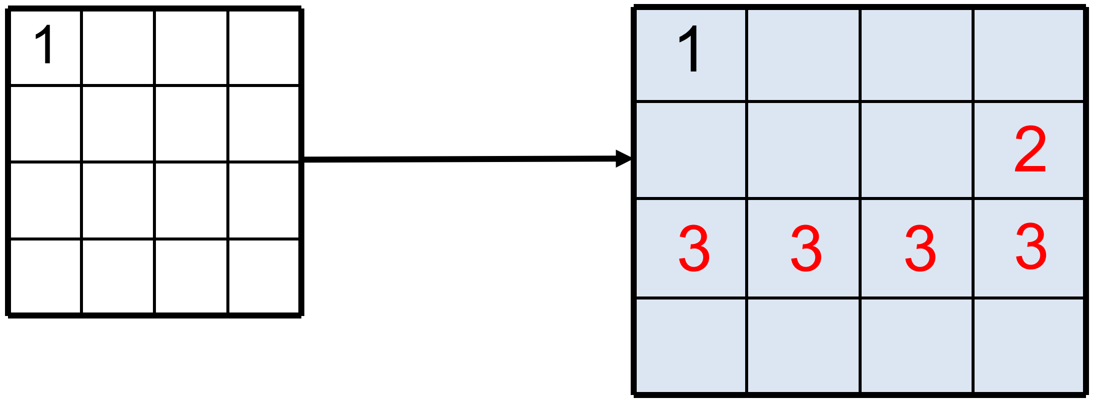
问题实例

- 当前k=3, X(3)=0

i	1	2	3	4
X(i)	1	3	5	

- 回溯到k=2，修改X(2)

i	1	2	3	4
X(i)	1	4	0	



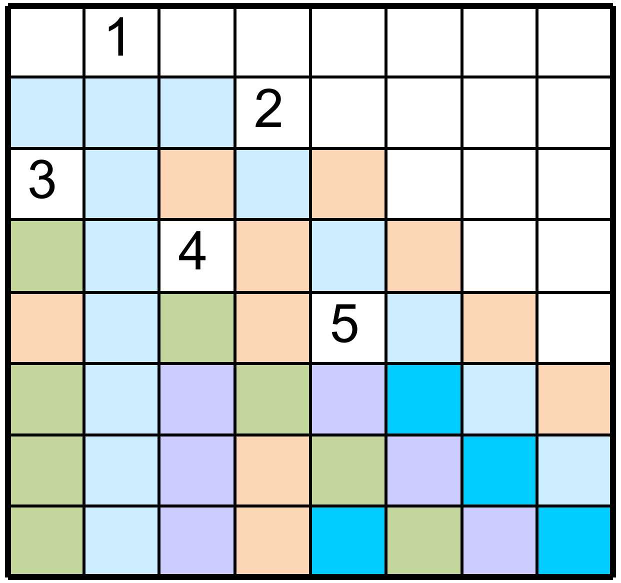
- 令k=3，重新开始构造X(3)

8-皇后问题的效率估计

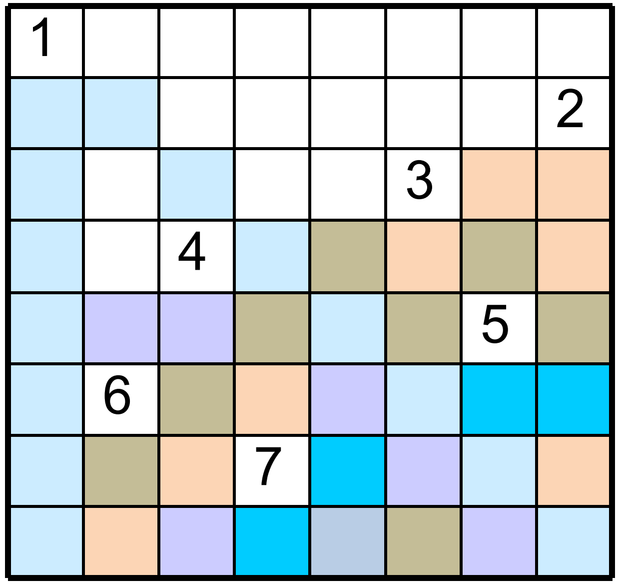
- 在8-皇后问题中，$(x_1,x_2,..x_8)$表示一个解，x_i表示第i个皇后放在第i行的列数。
  - 显式: $S_i=\{1, 2, 3, 4, 5, 6, 7, 8\}, 1\le i\le 8$
  - 隐式: 没有两个x_i可以相同，
    且没有两个皇后可以在同一条斜角线上。
- 硬性处理法：8^8
  - 状态空间树结点个数: $1+8+8^2+\ldots+8^8$
- 没有两个x_i可以相同：8!
  - 状态空间树结点个数：$1+8+8*7+8*7*6+\ldots+8*7*6*5*4*3*2*1=69281$
- 限界函数实现隐式约束条件？

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



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



8
6
4
2
1
1
1
=1401

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 子集和问题

- 问题描述
- 限界函数
- 算法描述
- 效率估计
- 实例运行结果

问题描述

- 子集和问题：
  - 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：
  - n-元组$X=(x_1,\ldots x_n)$
  - 显示约束条件：$x_i\in\{0,1\},1\le i\le n$；如果选择$w_i$，则$x_i=1$；否则$x_i=0$
  - 隐式约束：$\sum w_i x_i=M$，$1\le i\le n$

限界函数

设一维数组X和W，X表示n-元组解向量，W记录n个正数

- 当满足 $\sum_{i=1}^{k} W(i)X(i) + \sum_{i=k+1}^{n} W(i) \geq M$
  - X(1),..,X(k)能导致一个答案结点
- 设W(i)按非降次序排列，那么当 $\sum_{i=1}^{k} W(i)X(i) + W(k+1) > M$
  - X(1),..,X(k)不能导致一个答案结点。
- 综上，限界函数B_k(X(1),…,X(k))=true，当且仅当：

W(i)按非降次序排列

$$\underbrace{\sum_{i=1}^{k} W(i)X(i)}_{s} + \underbrace{\sum_{i=k+1}^{n} W(i)}_{r} \geq M \text{ 且 } \sum_{i=1}^{k} W(i)X(i) + W(k+1) \leq M$$

算法7.6 子集和的递归回溯算法

Procedure SUMOFSUB(s,k,r)

//找出W(1:n)中和为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),..X(k-1)的值已确定。

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

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

boolean X(1:n)

integer s, k, r

//生成左儿子。由于B_{k-1}=true，因此s+W(k)≤M 且s+r ≥ M

```
X(k)←1
if s+W(k) = M
      then print(X)
   else if s+W(k)+W(k+1) ≤ M then call SUMOFSUB(s+W(k), k+1, r-W(k))
         endif
endif
```

左子树-递归入口

//生成右儿子和计算B_k的值

```
If s+r-W(k) ≥ M and s+W(k+1) ≤ M
   then X(k)←0; call SUMOFSUB( s,k+1,r-W(k))
endif
end SUMOFSUB
```

右子树-递归入口

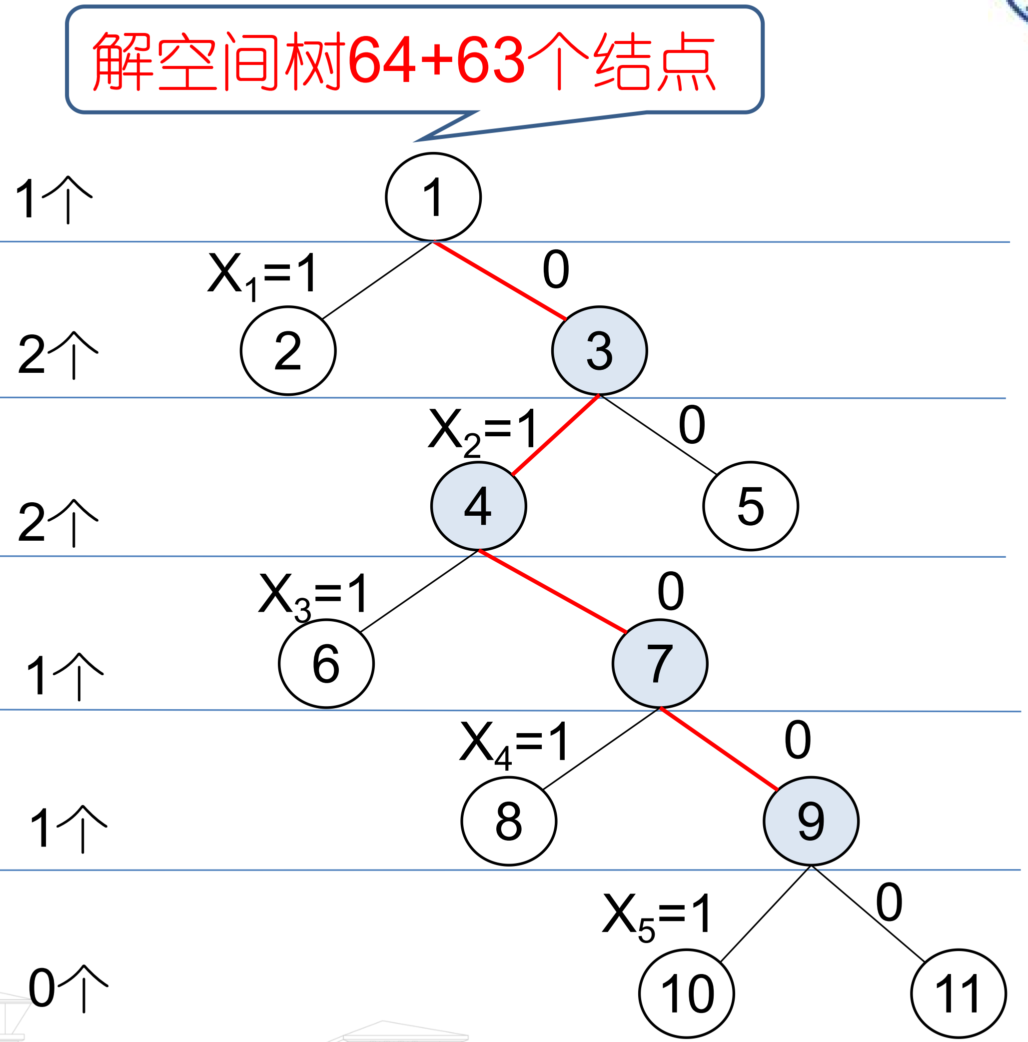
初始调用： SUMOFSUB(0,1, $\sum_{1\leqslant i\leqslant n} w(i)$)

思考：如果不将W预排序，算法怎样设计？算法效率怎样变化？

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
3	0	68	10	T
4	10	58	12	T
5	0	58	12	T
6	22	46	13	F
7	10	46	13	T
8	23	33	15	F
9	10	33	15	T
10	25	18	18	F
11	10	18	18	F



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

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23个结点，寻找/提前寻找到3个答案结点

和估计结果接近

7.5 图着色问题

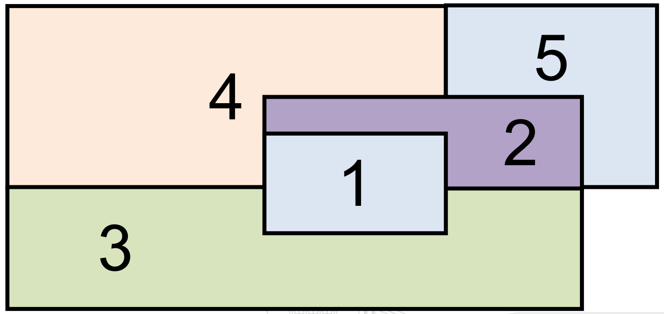
- 问题描述
- 图的m-着色判定问题
- 解空间树
- 限界函数
- 回溯算法
- 实例分析

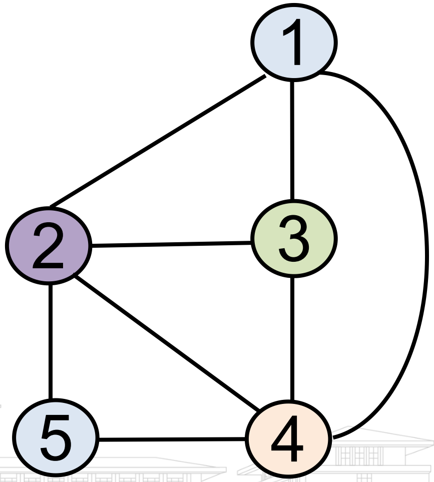
问题描述

- 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

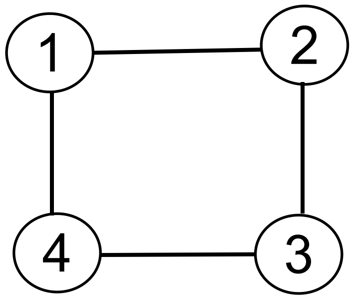
- 给定无向连通图G=(V,E)，其中V为顶点集合，E为边集合，用m种不同颜色给图中顶点着色，问：是否存在任何两个相邻顶点颜色不同的着色方案？
- 本节用回溯来解决图的m着色判定问题，如果判定答案为“是”，要求给出着色方案。

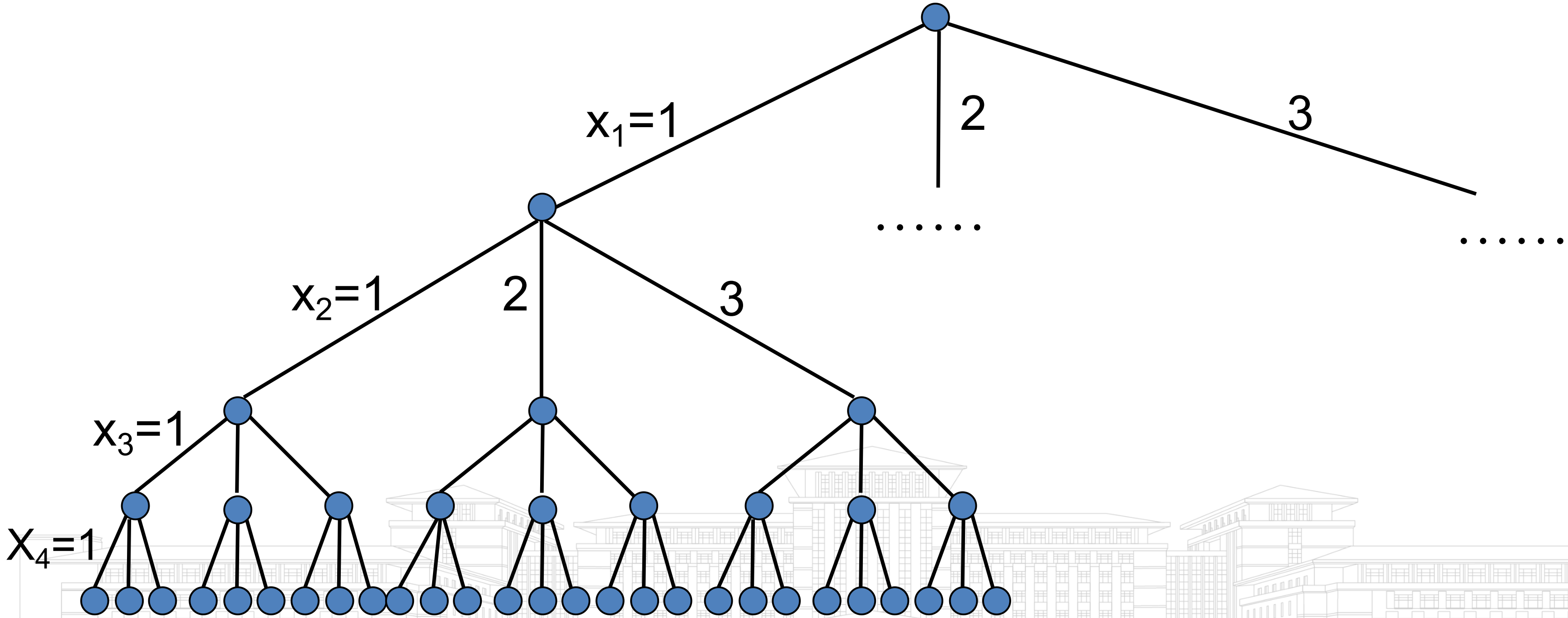




解空间树

- 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,..x_4)， x_i表示结点i的颜色。
  - 显式： 1≤ x_i ≤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 判断顶点k的着色是否合法

限界函数

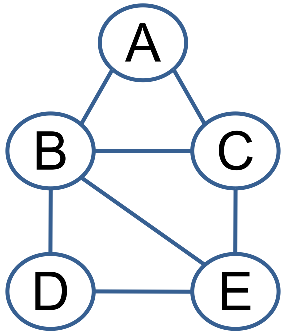
```
procedure  OK(k)
// C是全局数组, 进入此过程时已置入了前k个值
   int  i , k
   i ← 1
  while (i<k)  do
      if (i和k之间有边存在 and C(i)=C(k))
          then  return false
      endif
      i ← i+1
   repeat
   return true
end  OK
```

算法7.8 回溯法求解图着色判定问题

```
Procedure MCOLORING(V,E,C,n,m)
// 图G=(V,E),n个顶点，m种颜色
C(1:n) ← 0; k ← 1 //C记录决策序列，从第一个顶点开始
while (k≥1)
    C(k) ← C(k) +1
    while (not OK(k) and C(k)≤m) do C(k) ← C(k) +1 repeat
    if C(k) ≤ m then
            if k=n then print(C); return true //全部着色,打印
                    else k ← k+1; C(k) ← 0 //准备为下一个顶点着色
            endif
        else k ← k-1 //顶点k无法着色，回溯
    endif
repeat //k=0表示整颗树遍历完毕
return false
END MCOLORING
```

实例分析

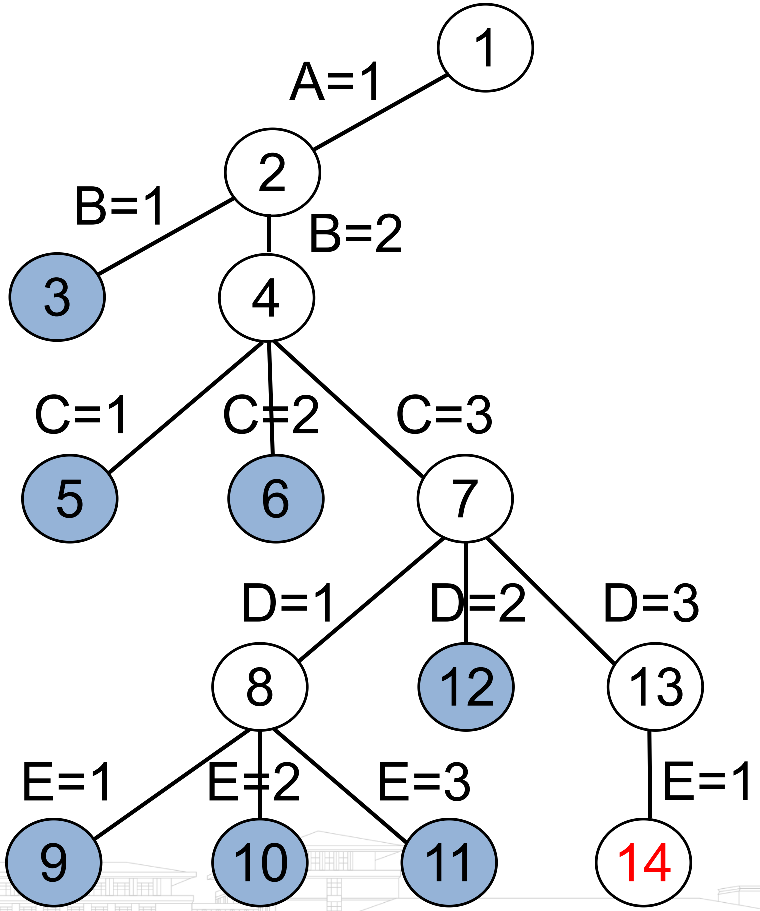
(a) 一个无向图



n=5个顶点的无向图，m=3，对应的空间树是完全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

- 回溯法的改进
  - 根据树的分支情况设计优先策略，如优先搜索结点少/解多的分支
  - 利用搜索树的对称性对子树进行剪裁
  - 分解成子问题，求解完子问题的解之后合并出原问题的解

- 7.1 回溯一般方法
    - 掌握回溯法适用的问题特点、解空间树、状态结点生成办法等基本知识。能掌握回溯法求解问题的设计思想和一般方法。
- 7.2 回溯法的效率估计
    - 掌握影响回溯法效率的要素和蒙特卡洛方法的设计思想
- 7.3 n-皇后问题
- 7.4 子集和问题
- 7.5 图着色问题
    - 掌握解空间树、限界函数和回溯算法设计，运用蒙特卡洛方法对算法效率进行分析
    - 掌握经典问题的回溯法求解方法，掌握限界函数优化思想，提高回溯算法剪枝能力，提高算法效率

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