算法设计与分析

华中科技大学计算机学院

赵峰

八 回溯法

8.1 一般方法

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。

1. 什么样的问题适合用回溯法求解呢？

基本要求：

1）问题的解可用一个n元组$(x_1,\ldots,x_n)$来表示，其中的x_i取自于某个有穷集S_i。

2）问题的求解目标是求取一个使某一规范函数$P(x_1,\ldots,x_n)$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：分类问题

对A(1:n)的元素分类问题

- 用n元组表示解：（$x_1,x_2,...,x_n$)
- x_i：表示第i小元素在原始数组里的下标，

  取自有穷集S_i=[1..n]。
- 规范函数P：$A(x_i) \leqslant A(x_{i+1})$，$1 \leqslant i < n$

如何求取满足规范函数的元组？

1. 硬性处理法(brute force)

- 枚举，列出所有候选解，逐个检查是否为所需要的解

  假定集合S_i的大小是m_i，则候选元组个数为

$$m = m_1 m_2 \ldots m_n$$

- 缺点：盲目求解，计算量大

2.寻找其它有效的策略

回溯或分枝限界法

回溯（分枝限界）法带来什么样的改进?

- 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（限界函数）去测试正在构造中的n元组的部分向量（$x_1, \cdots, x_i$），看其能否导致问题的解。**
- 如果判定（$x_1,...,x_i$）不可能导致问题的解，则将可能要测试的$m_{i+1}...m_n$个向量一概略去——剪枝，这使得相对于硬性处理大大减少了计算量。

概念

1.约束条件：问题的解需要满足的条件。

可以分为显式约束条件和隐式约束条件。

显式约束条件：一般用来规定每个x_i的取值范围。

如： $x_i \geq 0$ 即S_i={所有非负实数}

$x_i=0$或$x_i=1$ 即 $S_i=\{0,1\}$

$l_i \leq x_i \leq u_i$ 即$S_i=\{l_i \leq a \leq u_i\}$

解空间：实例I的满足显式约束条件的所有元组，即所有x_i合法取值的元组，构成I的解空间。

隐式约束条件：用来规定I的解空间中那些满足规范函数的元组，隐式约束将描述x_i必须彼此相关的情况。

例8.1：8-皇后问题

在一个8×8棋盘上放置8个皇后，且使得每两个皇后之间都不互相“攻击”：即使得每两个皇后不在同一行、同一列及同一条斜角线上。

	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

行、列号： 1…8

皇后编号： 1…8, 约定皇后i放到第i行的某一列上。

解的表示： 可以用8-元组($x_1,\ldots,x_8$)表示， 其中x_i是皇后i所在的列号。

显式约束条件： $S_i=\{1,2,3,4,5,6,7,8\}$, $1 \leqslant i \leqslant 8$

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（排列），最多有8！个。

	1	2	3	4	5	6	7	8
1				Q				
2						Q		
3								Q
4		Q						
5							Q	
6	Q							
7			Q					
8					Q			

图中的解表示为一个8-元组为（4，6，8，2，7，1，3，5）

例8.2 子集和数问题

已知n+1个正数$w_1, w_2, \ldots, w_n$和M。要求找出w_i的和数等于M的所有子集。

例：n＝4，（w_1, w_2, w_3, w_4）＝（11，13，24，7），M=31。则满足要求的子集有：

- 直接用元素表示：（11，13，7）和（24，7）
- k-元组（用元素下标表示）：（1，2，4）和（3，4）
- n-元组（用n元向量表示）：（1，1，0，1）和（0，0，1，1）

解的表示：

形式一：

问题的解为**k-元组**$(x_1, x_2, \ldots, x_k)$, $1\le k\le n$。不同的解可以是大小不同的元组，如(1,2,4)和(3,4)。

显式约束条件：$x_i \in \{ j \mid j$为整数且$1\le j\le n \}$。

隐式约束条件：1）没有两个x_i是相同的；

2）w_{xi}的和为M；

3）$x_i < x_{i+1}, 1\le i<n$（避免重复元组）

形式二：

解由n-元组$(x_1, x_2, \ldots, x_n)$表示，其中$x_i \in \{0,1\}$。如果选择了w_i，则$x_i = 1$，否则$x_i = 0$。

例：（1，1，0，1）和（0，0，1，1）

特点：所有元组具有统一固定的大小。

显式约束条件：$x_i \in \{0,1\}$，$1 \le i \le n$;

隐式约束条件：$\Sigma(x_i \times w_i) = M$

解空间：所有可能的不同元组，总共有2^n个元组

解空间的组织形式

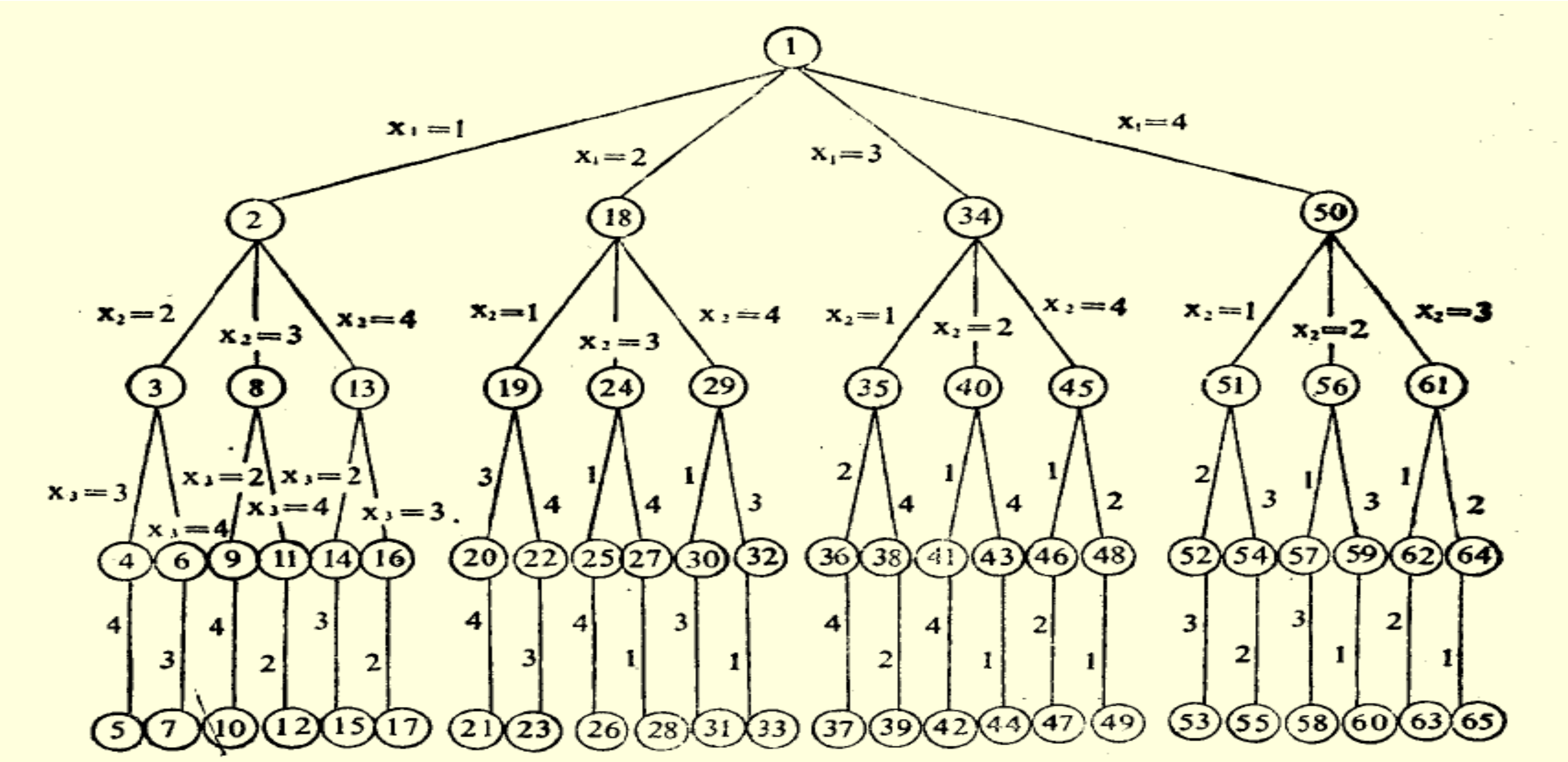
回溯法将通过系统地检索给定问题的解空间来求解，这需要有效的组织问题的解空间——把元组表示成为有结构的组织方式。**采用何种形式组织问题的解空间?**

可以用树结构组织解空间——状态空间树。

例8.3 n-皇后问题。8皇后问题的推广，即在n×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！个n-元组组成.

实例：4皇后问题的解空间树结构如下所示：



边：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。

如由1级到2级结点的边给出x_1的各种取值：1、2、3、4。

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

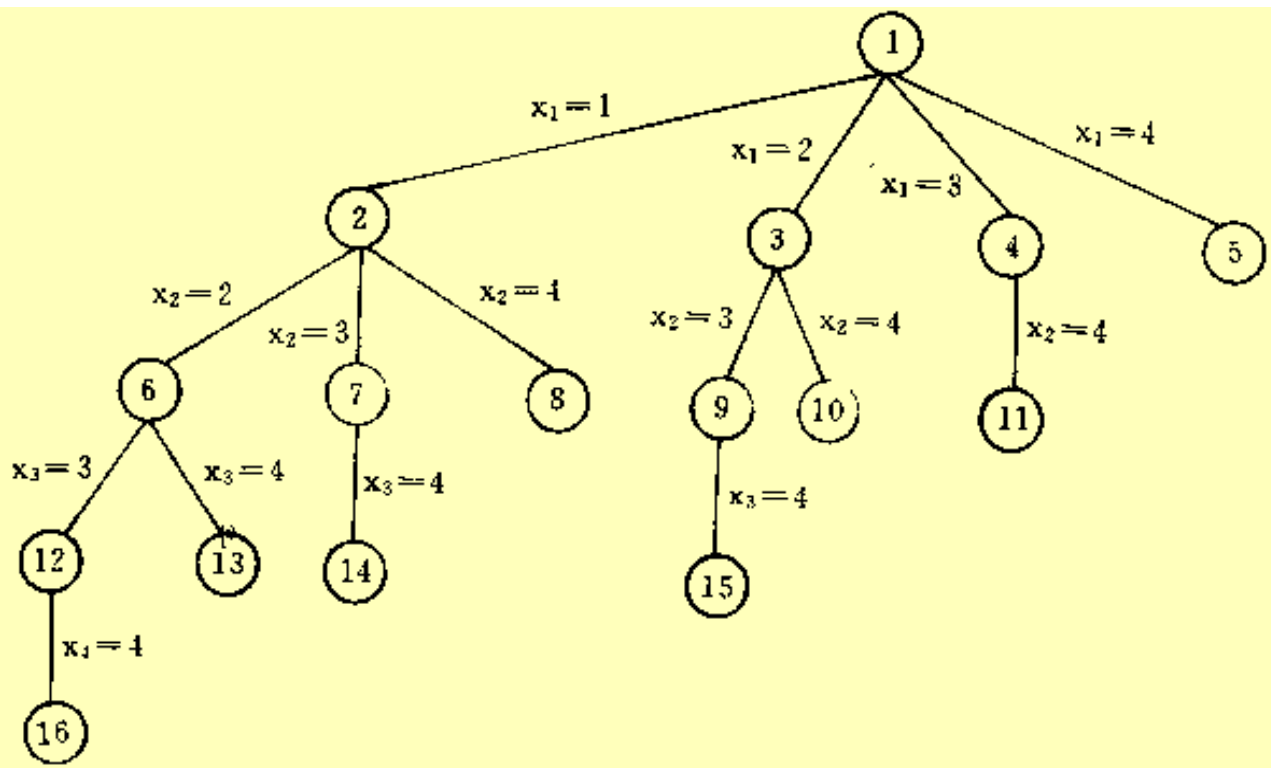
——称为排列树。

例8.4 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**（$x_i < x_{i+1}$）

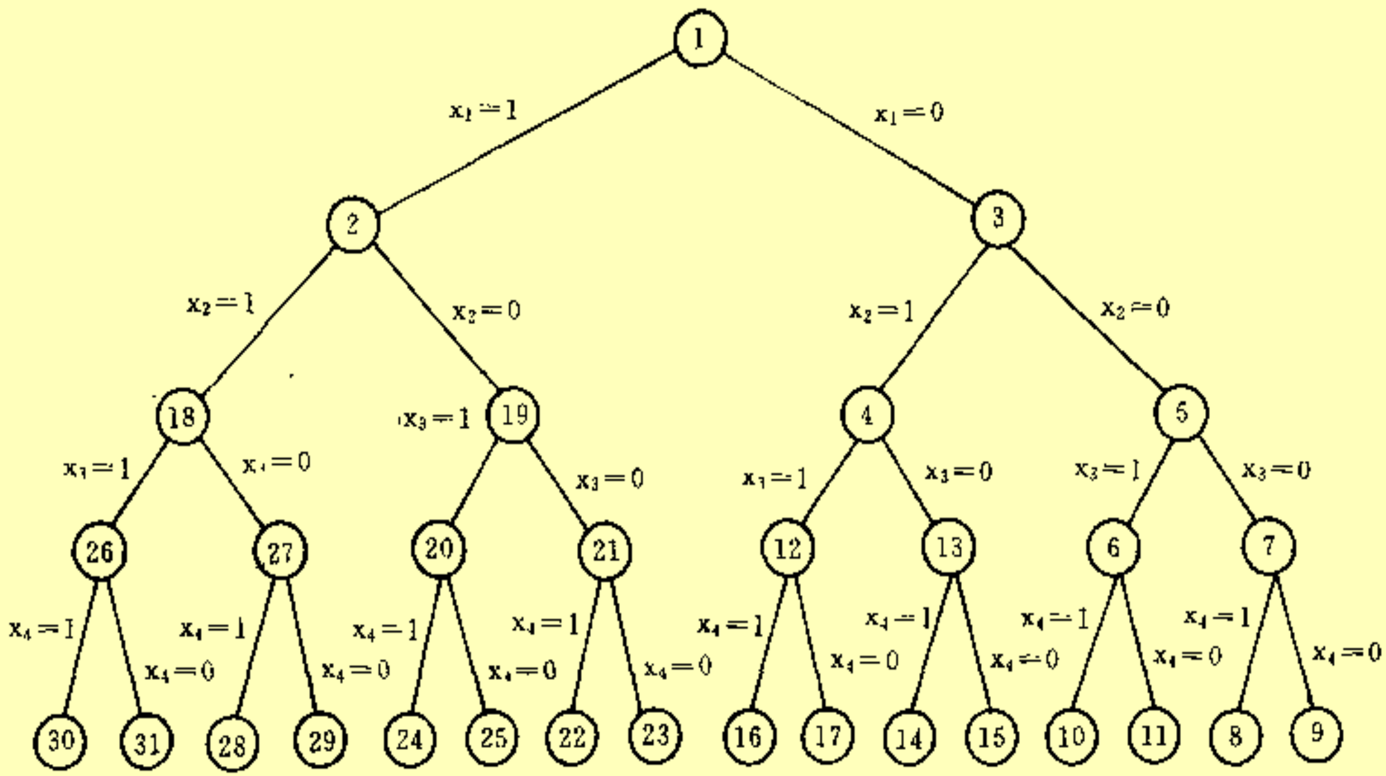
树边标记：由i级结点到i＋1级结点的一条边用x_i来表示，表示k-元组里的第i个元素是已知集合中下标为x_i的元素。



解空间由树中的根结点到任何结点的所有路径所确定：( ), (1),(1,2),(1,2,3),(1,2,3,4),(1,2,4),(1,3,4),(1,4),(2),(2,3)等。

2）元组大小固定，每个都是n-元组

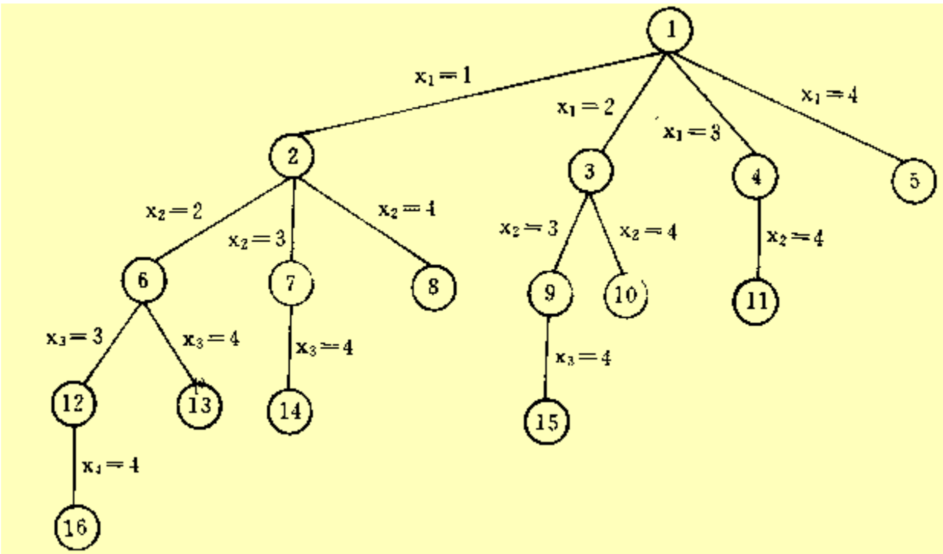
树边标记：由i级结点到i＋1级结点的那些边用x_i的值来标记，x_i＝1或0。

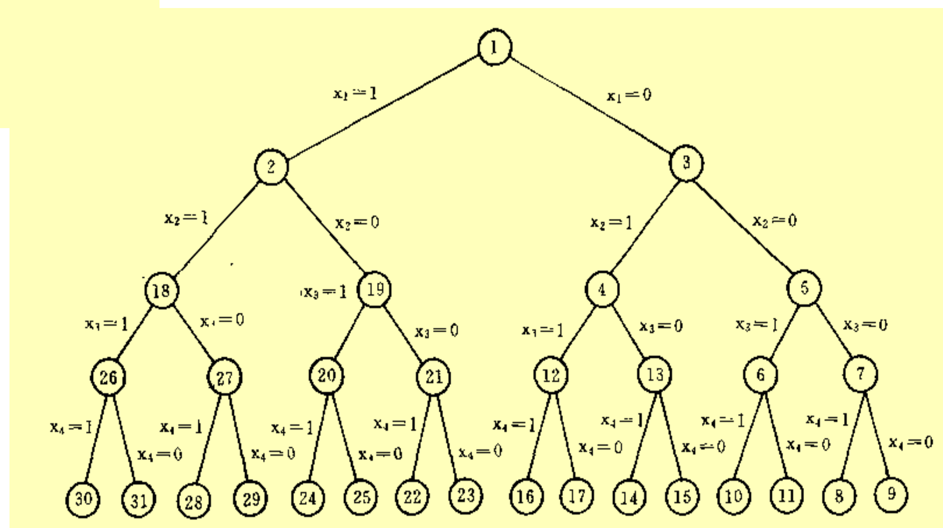


解空间由根到叶结点的所有路径所确定。共有16个可能的元组。

共有2^4＝16个叶子结点，代表所有可能的4元组。

同一个问题可以有不同形式的状态空间树。

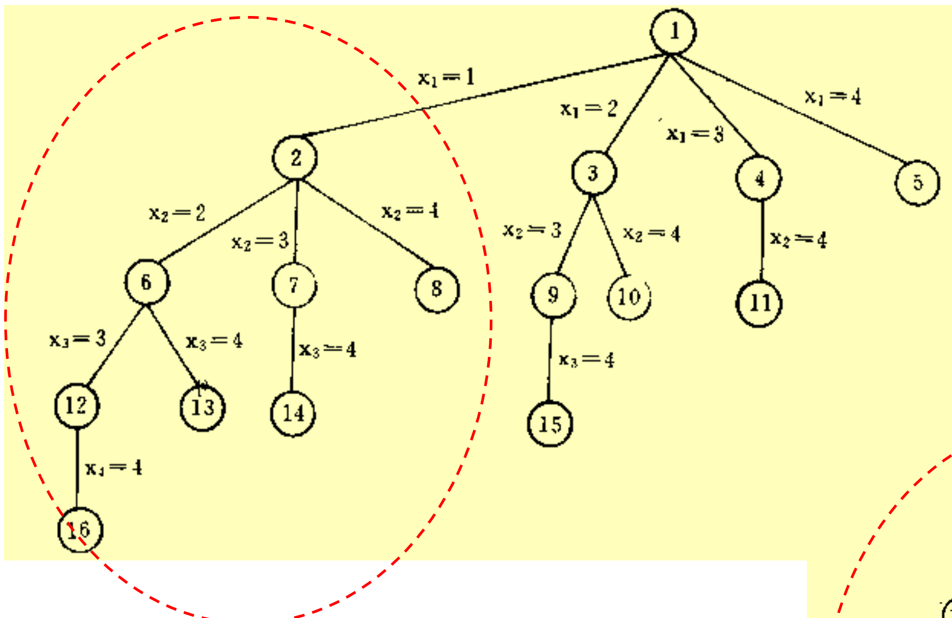




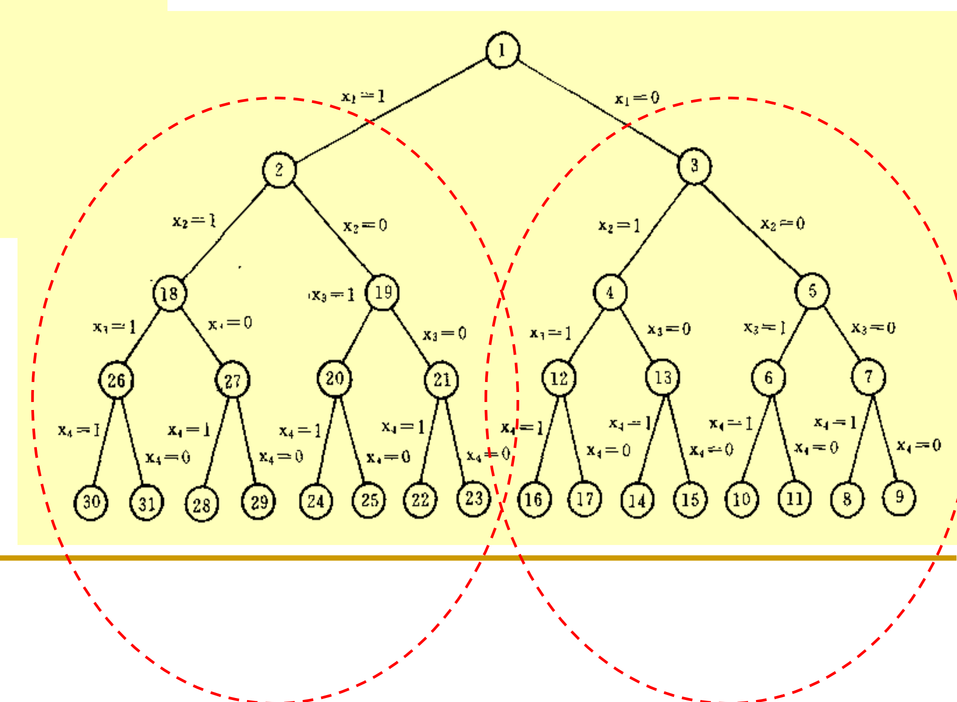
关于状态空间树的概念

- 状态空间树：解空间的树结构称为状态空间树(state space tree)
- 问题状态：树中的每一个结点确定问题的一个状态，称为问题状态(problem state)。
- 状态空间：由根结点到其他结点的所有路径则确定了这个问题的状态空间(state space)。
- 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个解空间中的一个元组(solution states)。
- 答案状态：是这样的一些解状态S，对于这些解状态而言，由根到S的这条路径确定了这问题的一个解（满足隐式约束条件的解）(answer states)。

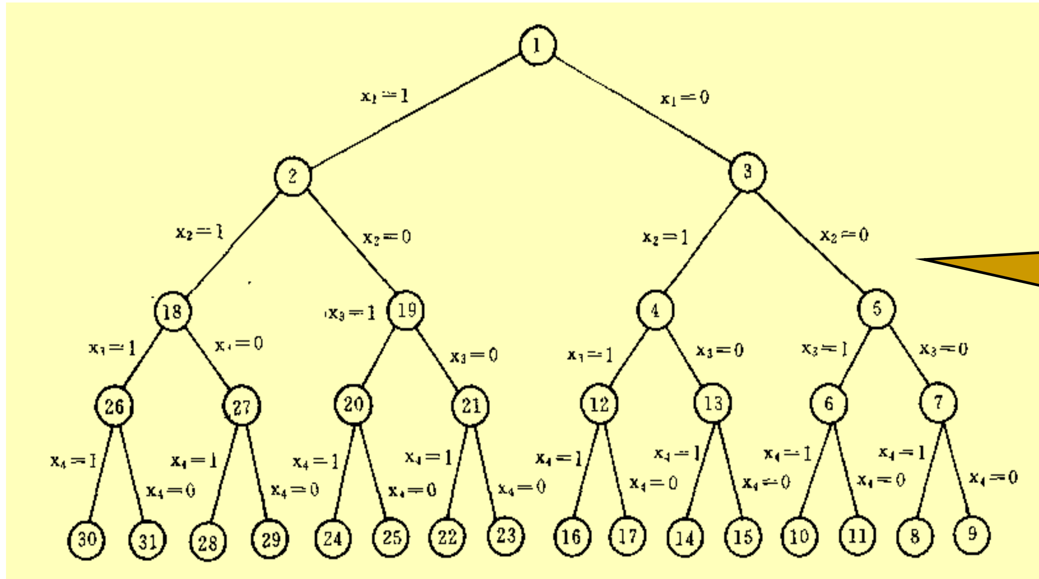
- **状态空间树的分解**：在状态空间树的每个结点处，解空间被分解为一些子解空间，表示在一些分量取特定值情况下的解空间元素。



特点：被分解的子解空间互不相交。但不是必须条件。



- 静态树：树结构与所要解决的问题的实例无关 (static trees)。



只要n=4，
状态空间树
都是这样

- 动态树：与实例有关的树称为动态树。 (dynamic trees)

  ——对有些问题，根据不同的实例使用不同的树结构可能更好，如：是不是先考虑x_2的取值更好呢？这就需要根据实例来动态构造状态空间树。

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。

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,但其儿子结点还没有全部生成并且有待生成的结点。
- **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**：不需要再进一步扩展或者其儿子结点已全部生成的结点。

构造状态空间树的两种策略

1. 深度优先策略

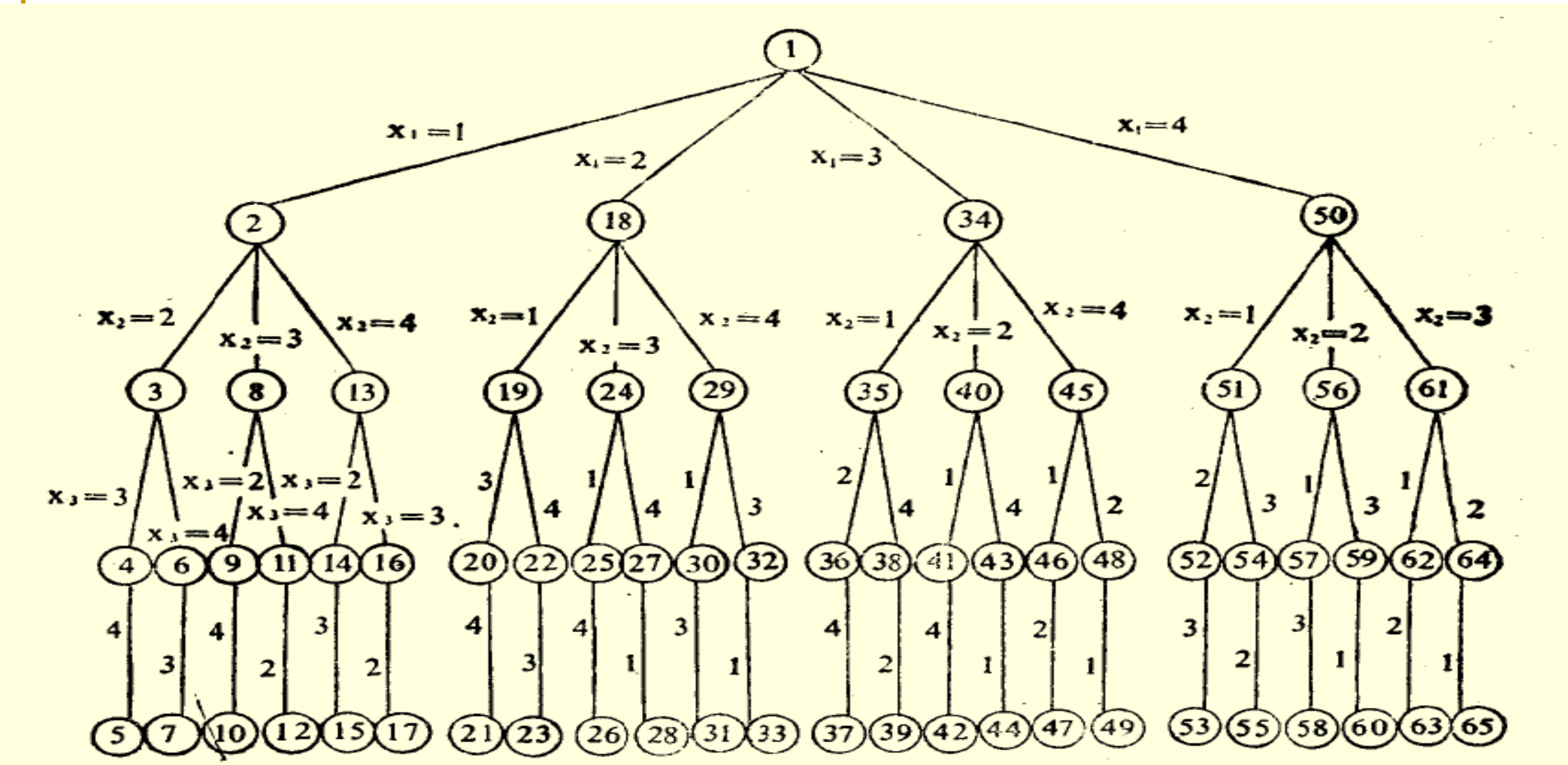
当E-结点R一旦生成一个新的儿子C时，C就变成一个新的E-结点，当完全检测了子树C之后，R结点再次成为E-结点。

2.宽度优先策略

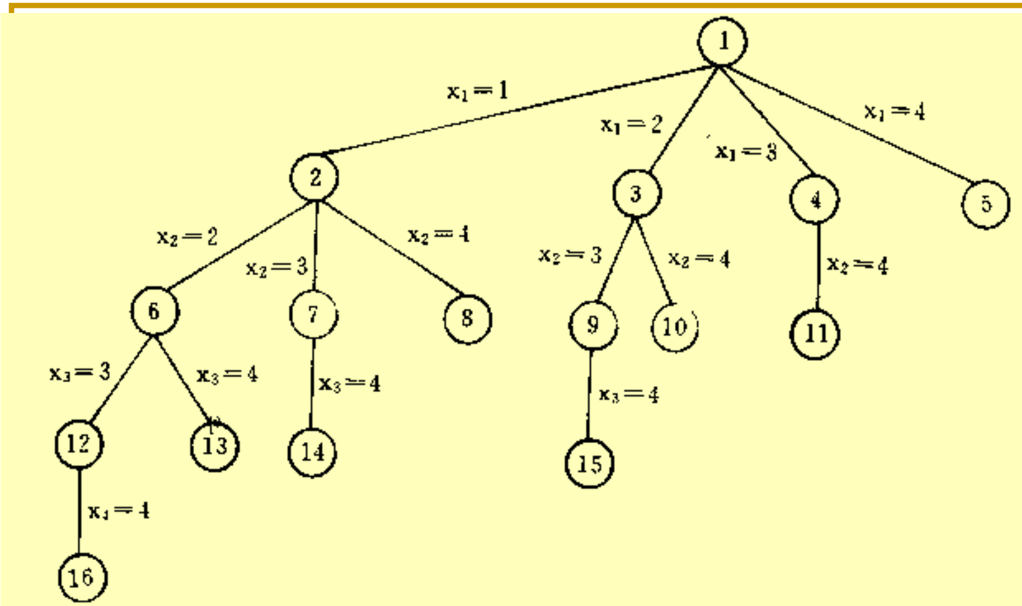
一个E-结点一直保持到变成死结点为止。

限界函数：在结点生成的过程中，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——这些活结点已是无法满足限界函数的条件，不可能导致问题的答案。

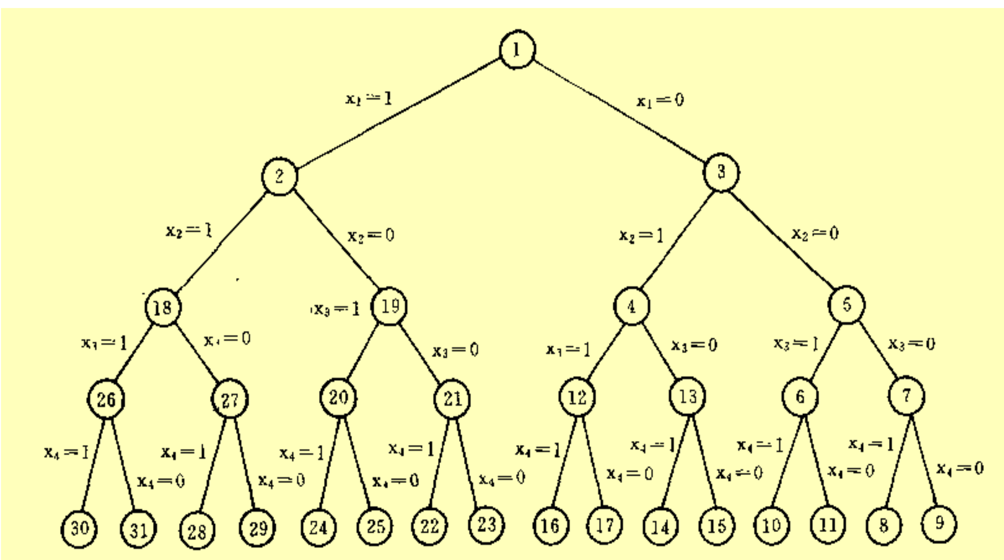
- 回溯法：使用限界函数的深度优先结点生成方法称为回溯法（backtracking）
- 分枝-限界方法：E结点一直保持到死为止的状态生成方法称为分枝-限界方法（branch-and-bound）



- 深度优先策略下的结点生成次序（结点编号）



- 利用队列的宽度优先策略下的结点生成次序(BFS)

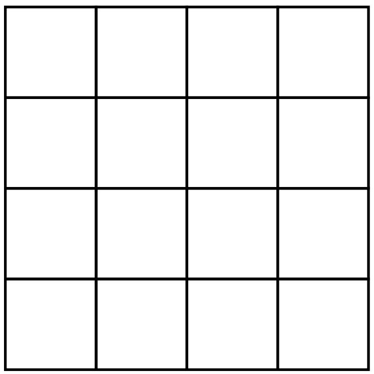


- 利用栈的宽度优先策略下的结点生成次序（D-Search）

例：4-皇后问题的回溯法求解

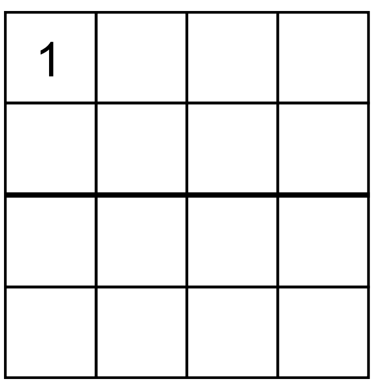
- 限界函数：如果（$x_1,x_2,...,x_i$）是到当前E结点的路径，那么x_i的儿子结点x_{i+1}是一些这样的结点，它们使得（$x_1,x_2,...,x_i,x_{i+1}$）表示没有两个皇后正在相互攻击的一种棋盘格局。
- 开始状态：根结点1，表示还没有放置任何皇后。
- 结点的生成：依次考察皇后1——皇后n的位置。

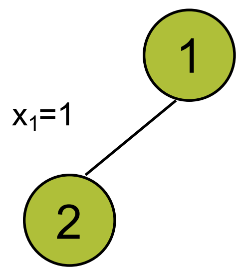
按照自然数递增的次序生成儿子结点。



根结点1，开始状态，唯一的活结点

解向量：()

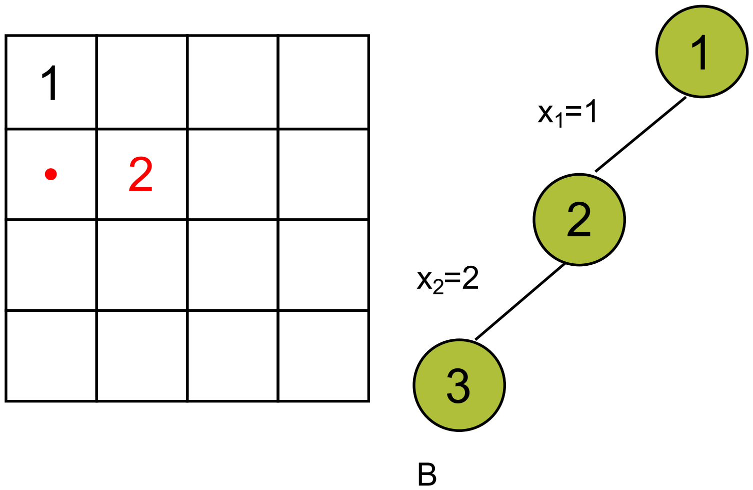




生成结点2，表示皇后1被放到第1行的第1列上，该结点是从根结点开始第一个被生成结点。

解向量：(1)

结点2变成新的E结点，下一步扩展结点2

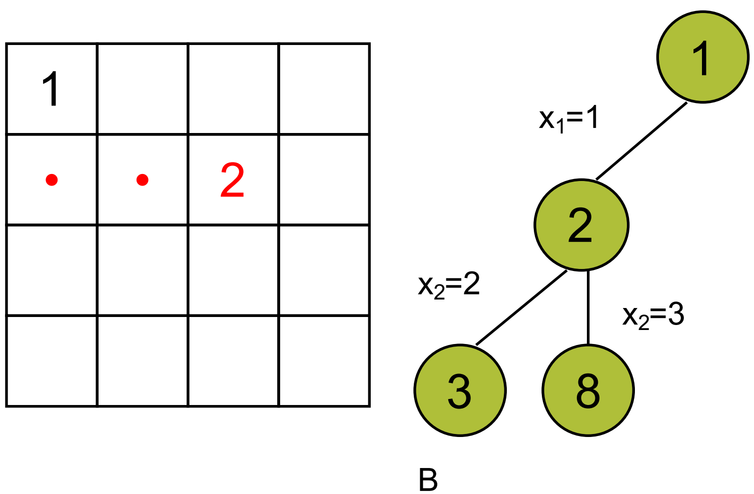


由结点2生成结点3，即皇后2放到第2行第2列。

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）

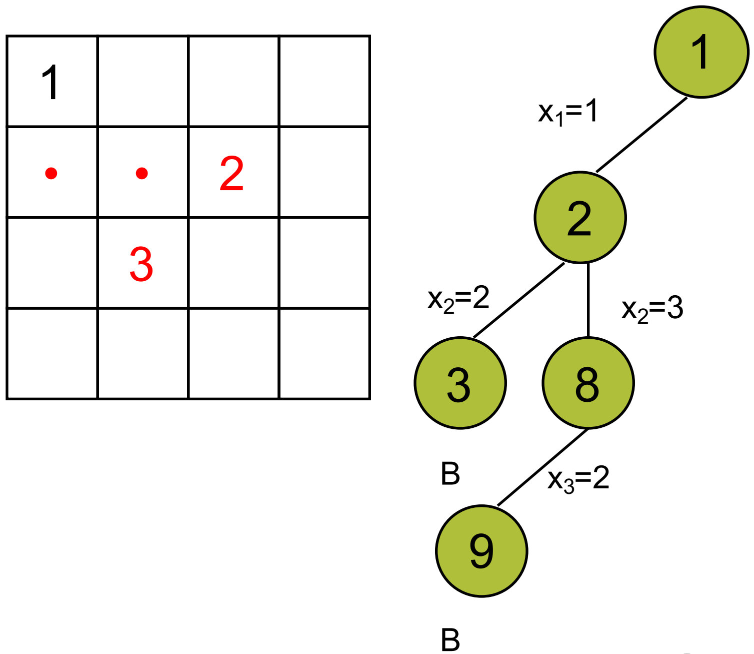


由结点2生成结点8，即皇后2放到第2行第3列。

结点8变成新的E结点。

解向量：（1，3）

从结点8继续扩展。

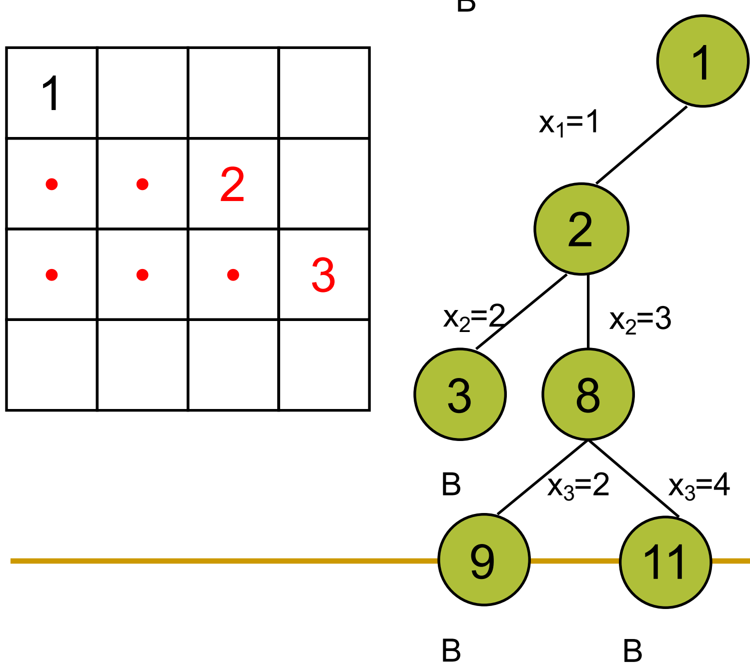


由结点8生成结点9，即皇后3放到第3行第2列。

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。

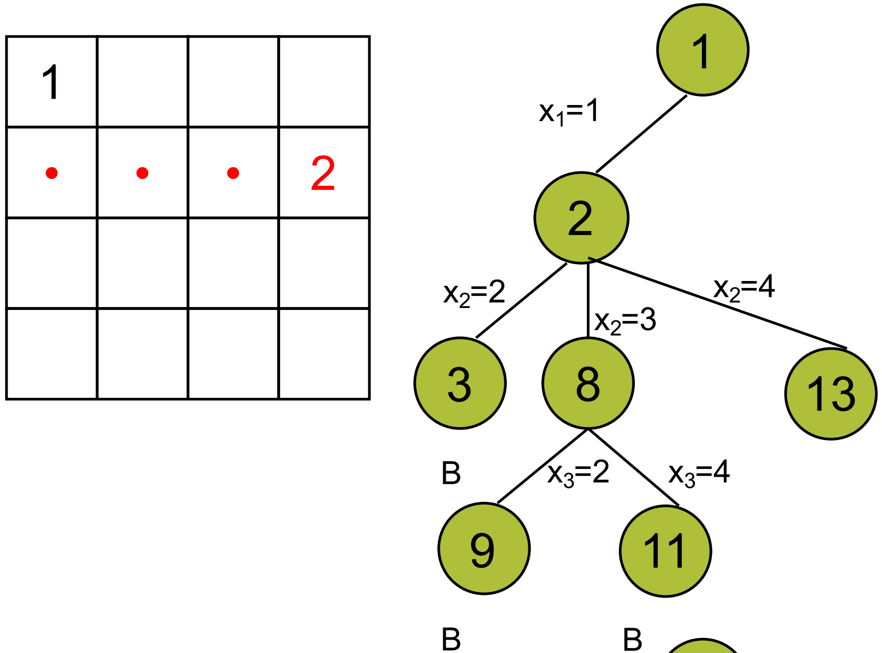
返回结点8继续。

（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。

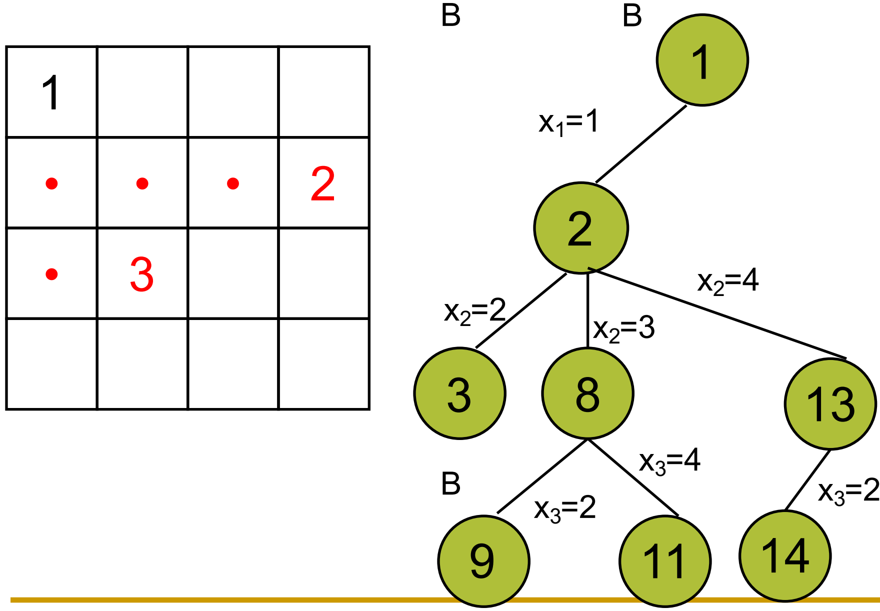


由结点2生成结点13，即皇后2放到第2行第4列。

结点13变成新的E结点。

解向量：（1，4）

从结点13继续扩展。



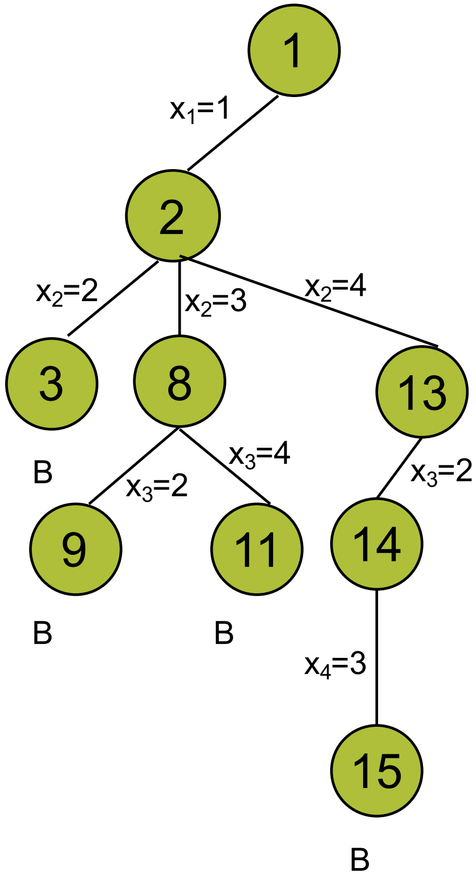
由结点13生成结点14，即皇后3放到第3行第2列。

结点14变成新的E结点。

解向量：（1，4，2）

从结点14继续扩展。

1			
•	•	•	2
•	3		
•	•	4	

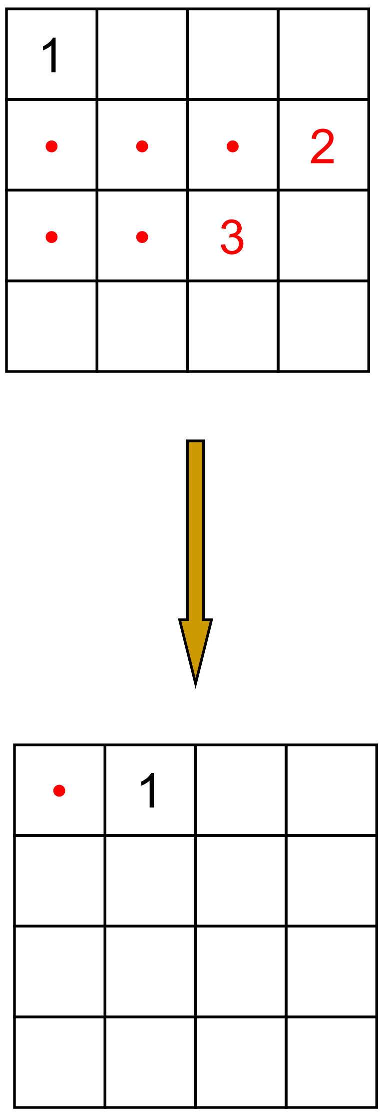


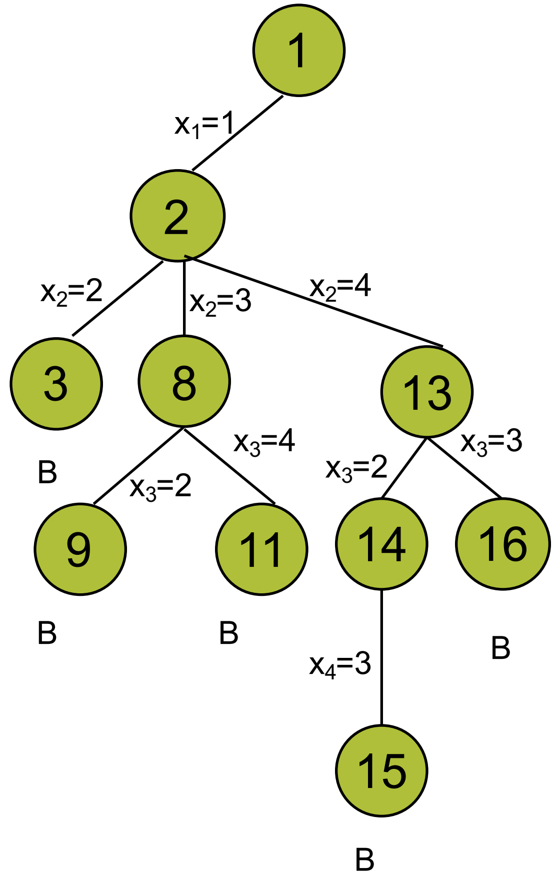
由结点14生成结点15，即皇后4放到第4行第3列。

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。





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。

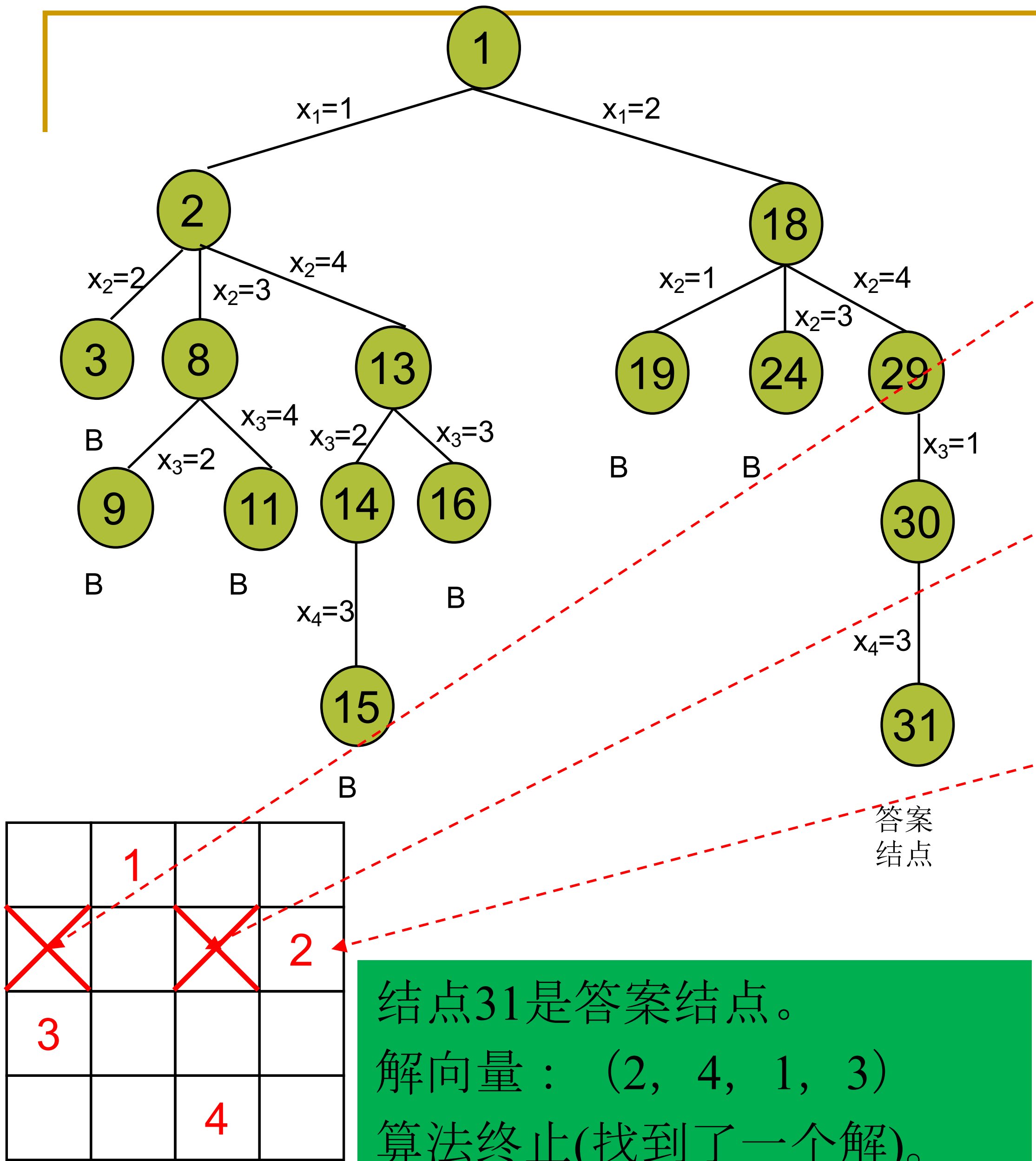
返回结点13，结点13不能导致答案结点，变成死结点，被杀死。

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。



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。

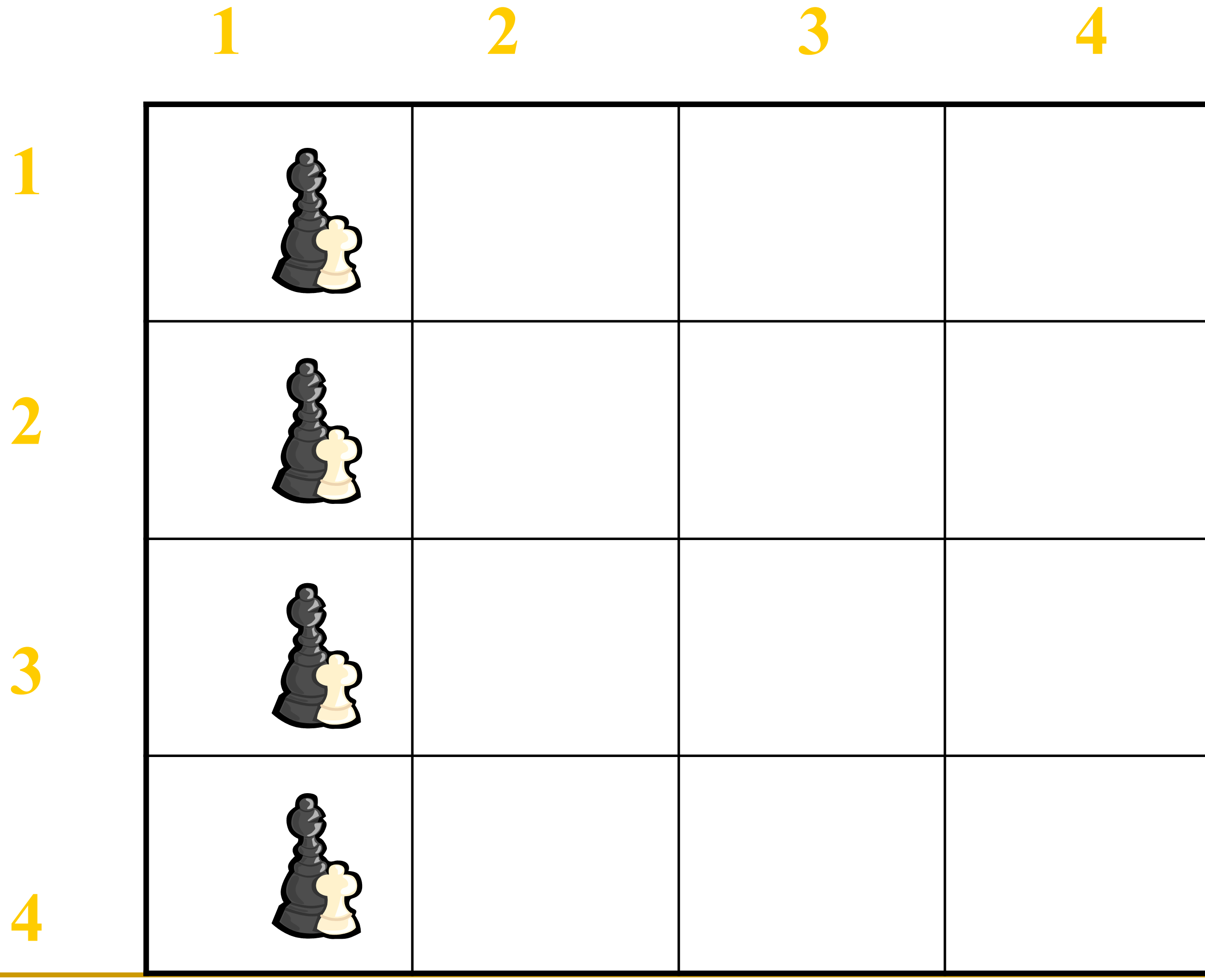
扩展结点30生成结点31，即皇后4放到第4行第3列。

结点31是答案结点。

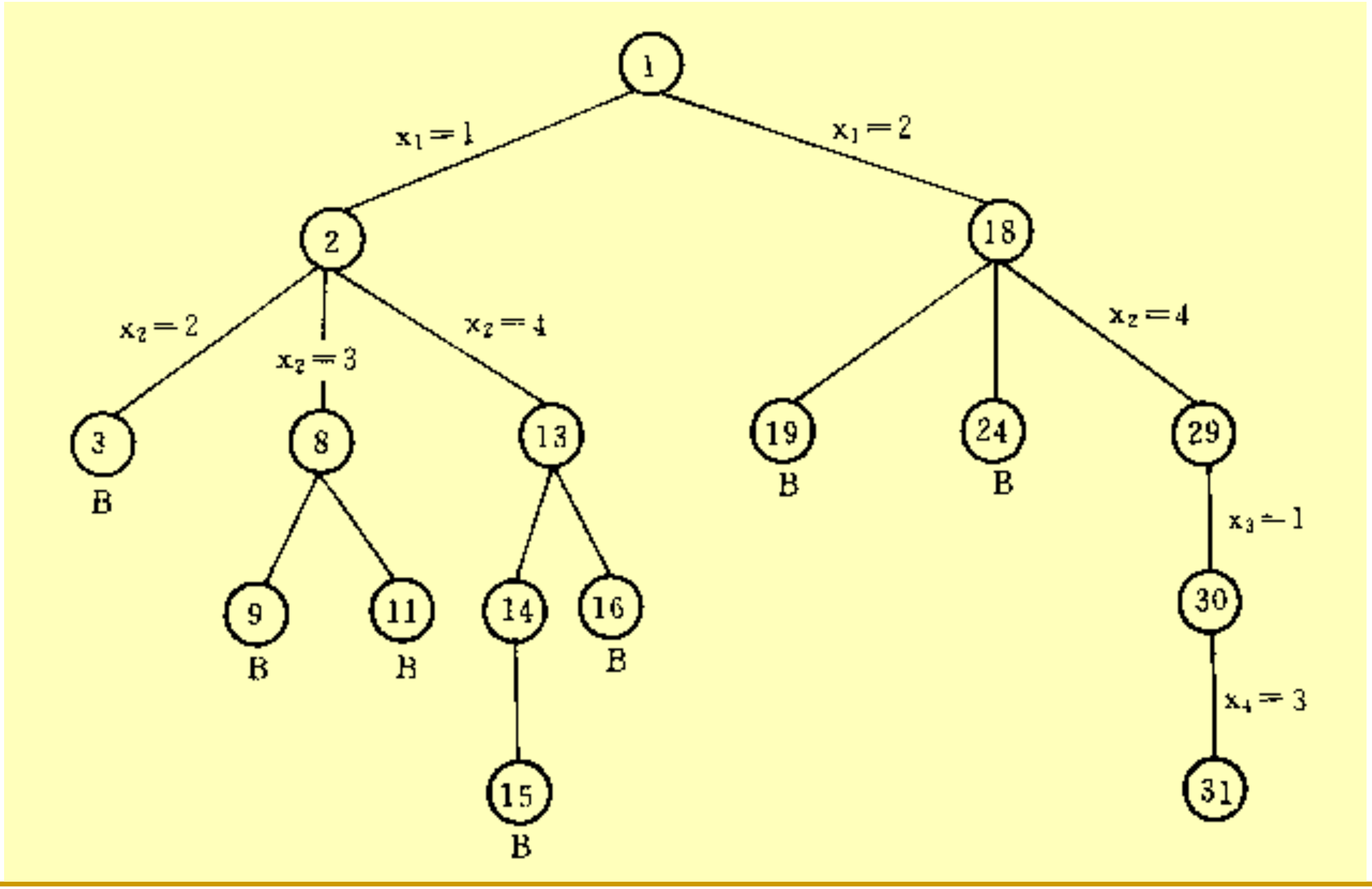
解向量：（2，4，1，3）

算法终止(找到了一个解)。

4-皇后问题的回溯法求解动画



4-皇后问题回溯期间生成的树



回溯算法的描述

- 设$(x_1, x_2, \ldots x_{i-1})$是由根到结点$x_{i-1}$的路径。
- $T(x_1, x_2, \ldots x_{i-1})$是下述所有结点$x_i$的集合，它使得对于每一个$x_i$， $(x_1, x_2, \ldots\ x_{i-1}\ ,x_i)$是由根到结点$x_i$的路径。
- **限界函数B_i**：如果路径$(x_1, x_2, \ldots x_i)$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。
- 解向量$X(1:n)$中的每个x_i即是选自集合 $T(x_1, x_2, \ldots x_{i-1})$且使$B_i$为真的$x_i$。

回溯法思想

- 第一步：为问题定义一个状态空间，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能被容易地搜索。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
  - 开始节点是第一个活节点（也是 E-节点：expansion node）
  - 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。
  - 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。

回溯的一般方法

回溯方法的抽象描述。该算法求出所有答案结点。

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

```
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) ∈T(X(1),…X(k-1)) and  B(X(1),…X(k))=true
      then
           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
           k ←k+1  //考虑下一个集合//
      else
           k ←k-1   //回溯到先前的集合//
      endif
   repeat
end BACKTRACK
```

回溯算法的递归表示

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。

```
procedure RBACKTRACK(k)
    global n, X(1: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
```

说明：当k>n时， T(X(1),…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。

效率分析应考虑的因素

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

重新排列方法

- 用于检索效率的提高
- 基本思想：在其它因素相同的情况下，从具有最少元素的集合中作下一次选择。
- 该策略已证明对n-皇后问题及其它一些问题无效

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
- 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

Monte Carlo效率估计

- 一般思想
  - 在状态空间中生成一条随机路径
  - X为该路径上的一个结点，且X在第i级
  - m_i为X没受限界的儿子结点数目
  - 从m_i随机选择一个结点作为下一个结点
  - …
  - 路径生成的结束条件：1）叶子结点；或者2）所有儿子结点都已被限界
  - 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

效率估计算法

```
procedure ESTIMATE
  m←1; r ← 1; k ← 1

  loop
     T_k←{X(k):X(k)∈ T(X(1),…X(k-1)) and  B(X(1),…X(k))}
      if SIZE(T_k)=0 then exit endif
      r←r*SIZE(T_k)
      m←m+r
      X(k)←CHOOSE(T_k)
      K←K+1
  repeat
  return(m)
end ESTIMATE
```

估算的条件限制：使用固定的限界函数

8.2 n-皇后问题

- n元组：（$x_1, x_2, \ldots, x_n$）
- 怎么判断是否形成了互相攻击的格局？
  - 不在同一行上：约定不同的皇后在不同的行
  - 不在同一列上：$x_i \neq x_j$，（$i, j \in [1:n]$）
  - 不在同一条斜角线上：如何判定？

1）在同一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

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

2）在同一斜角线上的由右上方到左下方的每一个元素有相同的“行＋列”值

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

判别条件：假设两个皇后被放置在(i,j)和(k,l)位置上，
则仅当：

i-j=k-l 或 i+j=k+l

时，它们在同一条斜角线上。

即： j-l = i-k 或 j-l = k-i

亦即： 当且仅当 | j-l | = | i-k |时，两个皇后在同一斜角线上。

过程PLACE(k)根据以上判别条件，判定皇后k是否可以放置在当前位置X(k)处——满足下述条件即可：

- 不等于前面的X(1)，…， X(k-1)的值，且
- 不能与前面的k-1个皇后在同一斜角线上。

Place算法

```
procedure PLACE(k)
  //如果皇后k可以放在第k行第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)  //在同一列上//
         or ABS(X(i)-X(k))=ABS(i-k)   //在同一斜角线上//
         then return(false)
      endif
      i← i＋1
   repeat
   return(true)
end PLACE
```

NQUEENS算法

对算法8.1用PLACE过程改进后得到求解n-皇后问题的算法：NQUEENS

```
procedure NQUEENS(n)
//在n×n棋盘上放置n个皇后，使其不能相互攻击。算法求出所有可能的位置//
  integer 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                                          //对所有的行执行以下语句//
          X(k)←X(k)+1                                     //移到下一列//
          while X(k) ≤ n and not PLACE(k) do              //检查是否能放置皇后//
                X(k)←X(k)+1                               //当前X(k)列不能放置，后推一列//
           repeat
           if X(k)≤n                                      //找到一个位置//
               then   if  k=n                             //是一个完整的解吗？//
                          then print(X)                   //是，打印解向量//
                          else k←k+1; X(k)←0              //否，转下一皇后//
                      endif
               else k←k-1
          endif
    repeat
end NQUEENS
```

算法分析

- PLACE算法：O(k-1)
- NQUEENS算法： C_{64}^{8} $\longrightarrow$ 8!

8.3 子集和数问题

- 元组大小固定：n元组（$x_1,x_2,\ldots,x_n$），x_i＝1或0
- 结点：对于i级上的一个结点，其左儿子对应于x_i=1，右儿子对应于x_i＝0。
- 限界函数的选择

约　定：W(i)按非降次序排列

条件一： $$\sum_{i=1}^{k} W(i)X(i)+\sum_{i=k+1}^{n} W(i) \geq M$$

条件二： $$\sum_{i=1}^{k} W(i)X(i)+W(k+1) \leq M$$

仅当满足上述两个条件时，限界函数B(X(1),…X(k))=true

注：如果不满足上述条件，则X(1),…X(k)根本不可能导致一个答案结点。

子集和数的递归回溯算法

//W(i)按非降次序排列,

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

W(1)≤M, $\sum_{i=1}^{n}W(i)\geq M$ //

```
procedure SUMOFSUB(s,k,r)
    global integer M,n; global real W(1:n);
    global boolean X(1:n) , real r,s; integer k,j
    X(k)←1                                  //生成左儿子，Bk-1=true,s+W(k)≤M//
    if s+W(k)=M then                        //找到答案//
            print(X(j),j←1 to k)            //输出答案//
    else  if s+W(k)+W(k+1)<=M then          //确保Bk＝true//
              call SUMOFSUB(s+W(k),k+1,r-W(k))
          endif
    endif
    //生成右儿子，计算Bk的值//
    if  s+r-W(k)>=M and s+W(k+1)<=M         //确保Bk＝true//
        then  X(k)←0
              call SUMOFSUB(s,k+1,r-W(k))
    endif
end SUMOFSUB
```

向前看两步,可以的话才进行下一步处理

首次调用SUMOFSUB(0,1,$\sum_{i=1}^{n}W(i)$)

SUMOFSUB的一个实例

- n=6, M=30, W(1:6)=(5,10,12,13,15,18)
- 方形结点：s，k，r，圆形结点：输出答案的结点，共生成20个结点