

计算机算法基础

何琨 brooklet60@gmail.com

华中科技大学计算机学院

八回溯法

第八章 回溯法

- 8.1 一般方法

 - 8.1.1 回溯的一般方法

- 8.2 8皇后问题

- 8.3 子集和数问题

8.1 回溯的一般方法

回溯法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. 适合范围:

1) 问题的解可用一个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

2) 求解目标: 满足约束条件的最优解 或合法解

即: 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: 背包问题, 一个装包方案可用**向量** $X = (x_1, x_2, \dots, x_n)$ 表示, 每个 x_i 表示物品 i 被放入背包的比例, $x_i \in \{0, 1\}$,

约束: $\sum_{1 \leq i \leq n} w_i x_i \leq M$

目标: $\max P(x_1, \dots, x_n) = \sum_{1 \leq i \leq n} p_i x_i$

如何求取满足规范函数的元组？

1. 硬性处理法

- 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 $m = m_1 m_2 \dots m_n$
- 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

- 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。

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：规定每个 x_i 的取值范围。

如： $x_i=0$ 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 L 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 L 的解空间。如： $x_i \in \{0,1\}$

隐式约束条件：用来规定 L 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。

如： $\sum_{1 \leq i \leq n} w_i x_i \leq M$

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。

	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, 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, \dots, x_8) 表示，其中 x_i 是皇后*i*所在的列号。

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

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，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, \dots, 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：

- 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 $(0, 0, 1, 1)$

解的表示：元素下标

形式一：

问题的解为**k-元组** (x_1, x_2, \dots, x_k) , $1 \leq k \leq n$ 。

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： $x_i \in \{ j \mid j \text{为整数且} 1 \leq j \leq n \}$ 。

隐式约束条件：1) 没有两个 **x_i** 是相同的；

2) **w_{x_i}** 的和为 **M** ；

3) **$x_i < x_{i+1}$** , $1 \leq i < n$

（避免产生同一个子集的重复情况，如

$\{1, 2, 4\}$, $\{2, 4, 1\}$ ）

形式二：

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 $x_i = 0$ 。

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： $x_i \in \{0, 1\}$ ， $1 \leq i \leq n$;

隐式约束条件： $\sum (x_i \times w_i) = M$

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

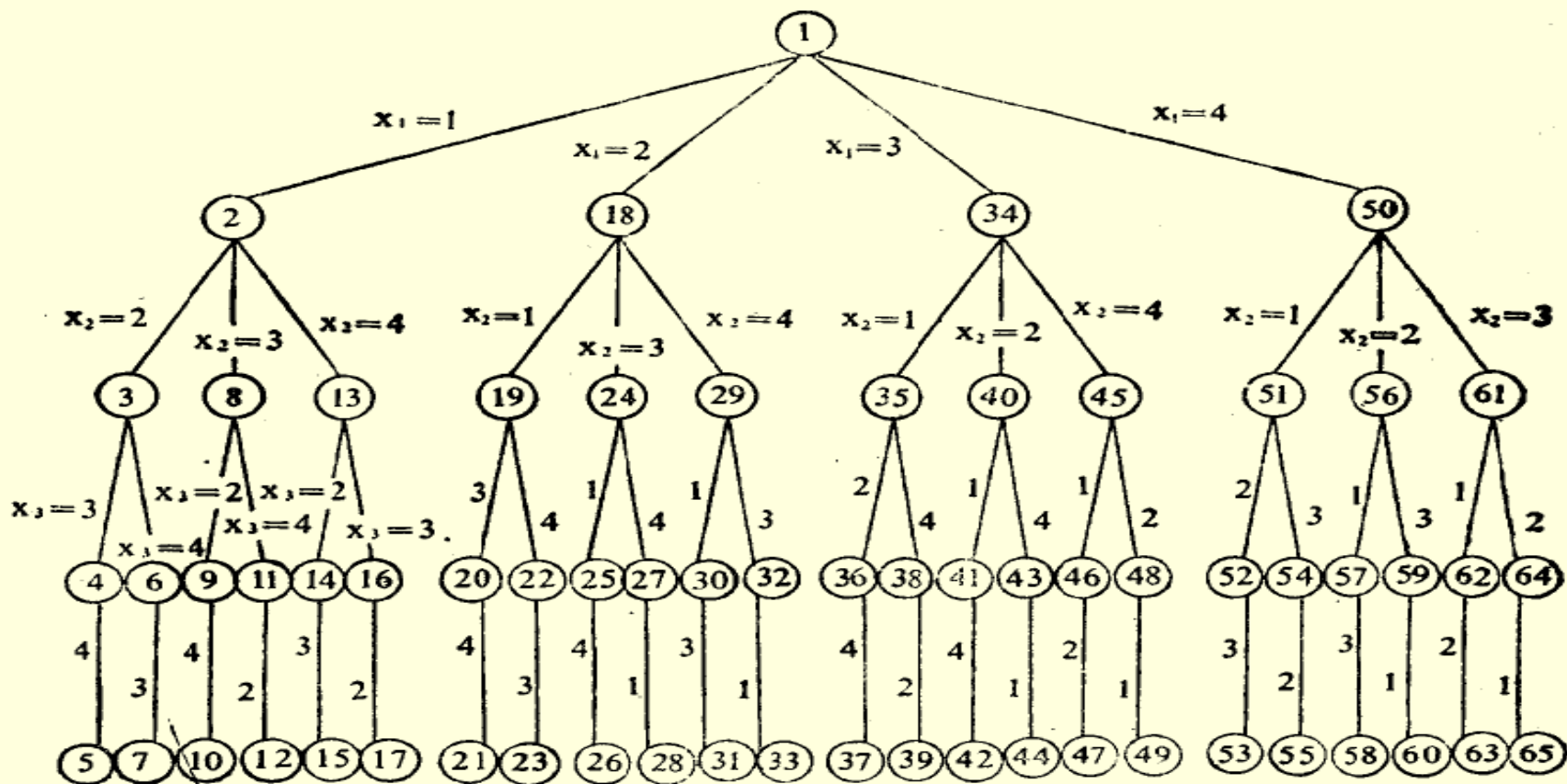
回溯法将通过系统地检索给定问题的解空间来求解，这需要有效的组织问题的解空间——把元组表示成为有结构的组织方式。采用何种形式组织问题的解空间？

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 $n \times n$ 的棋盘上放置 n 个皇后，使得它们不会相互攻击。

解空间：由 $n!$ 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。

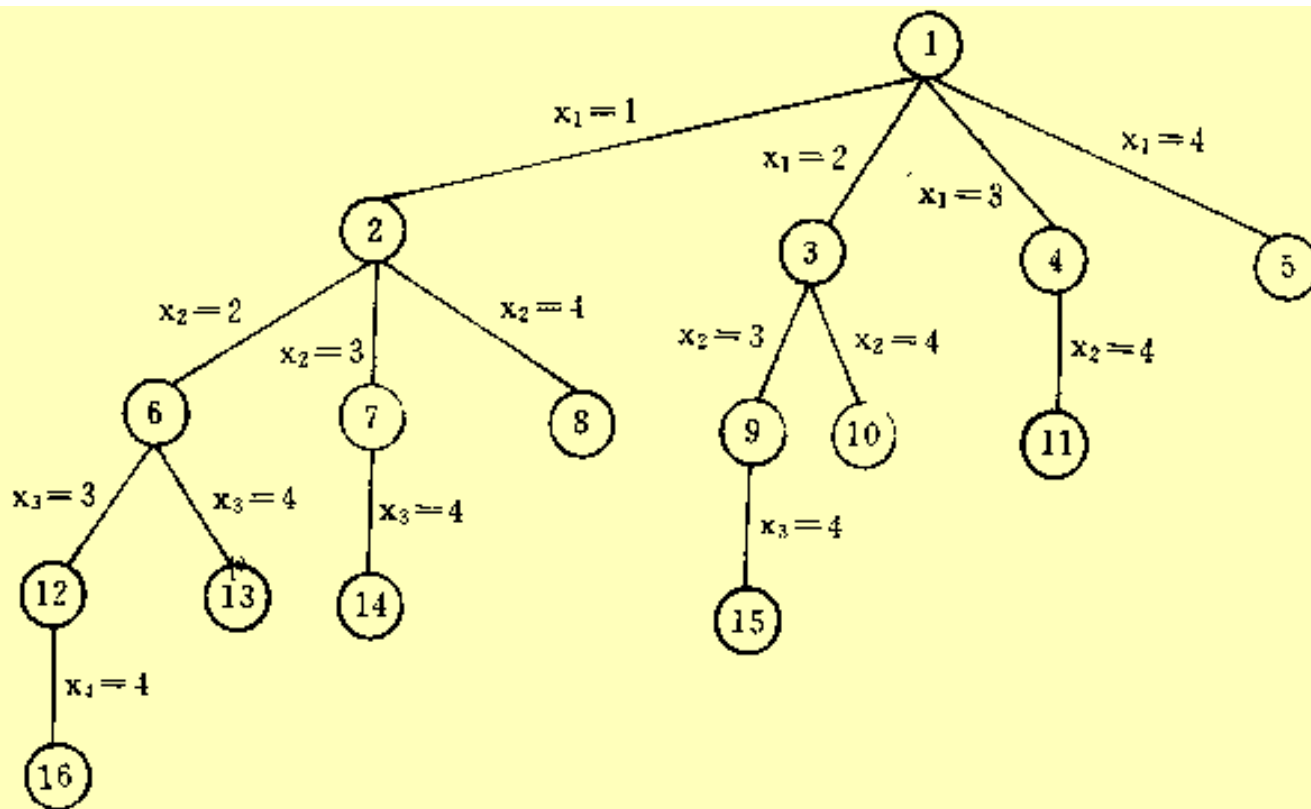
注：共有 $4! = 24$ 个叶结点，反映了4元组的所有可能排列
——称为排列树。

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 ($x_i < x_{i+1}$)

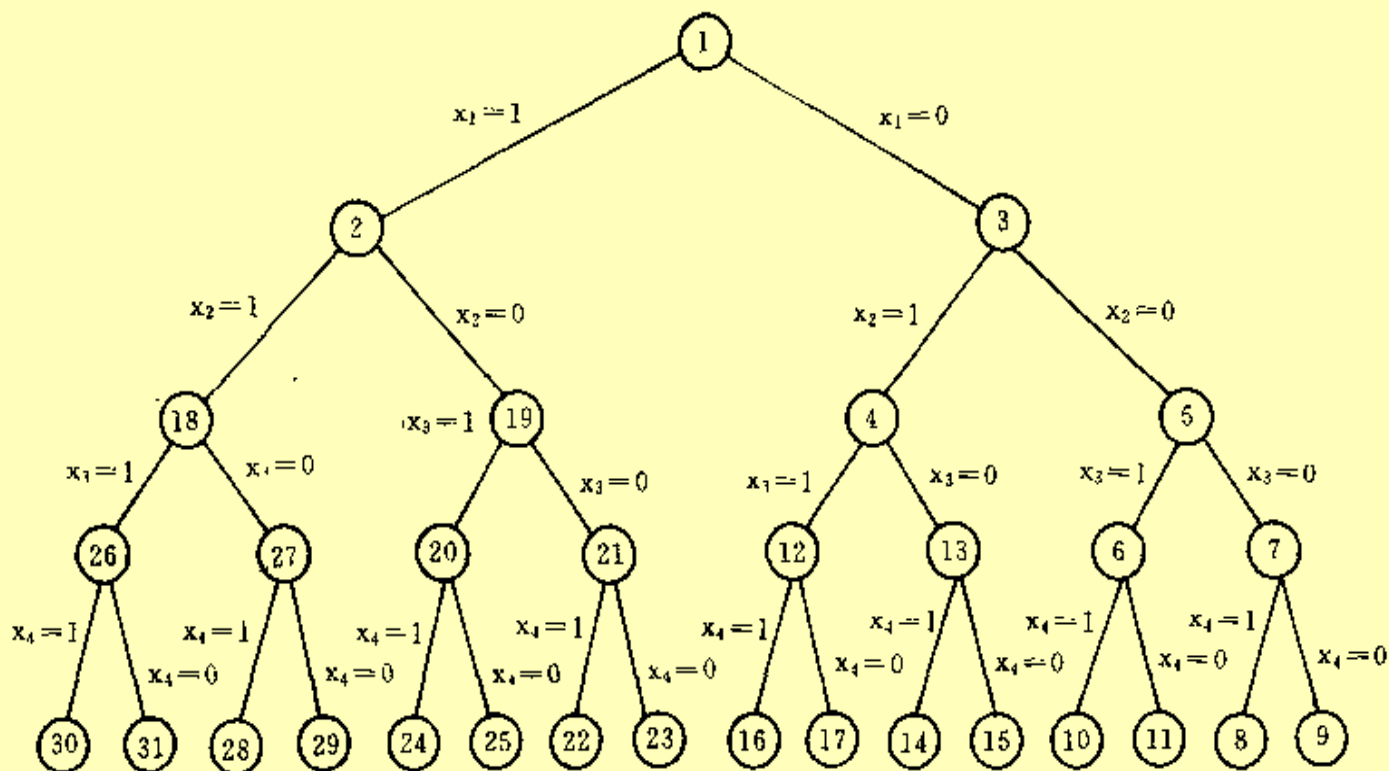
树边标记: 由*i*级结点到*i*+1级结点的一条边用 x_i 来表示。



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

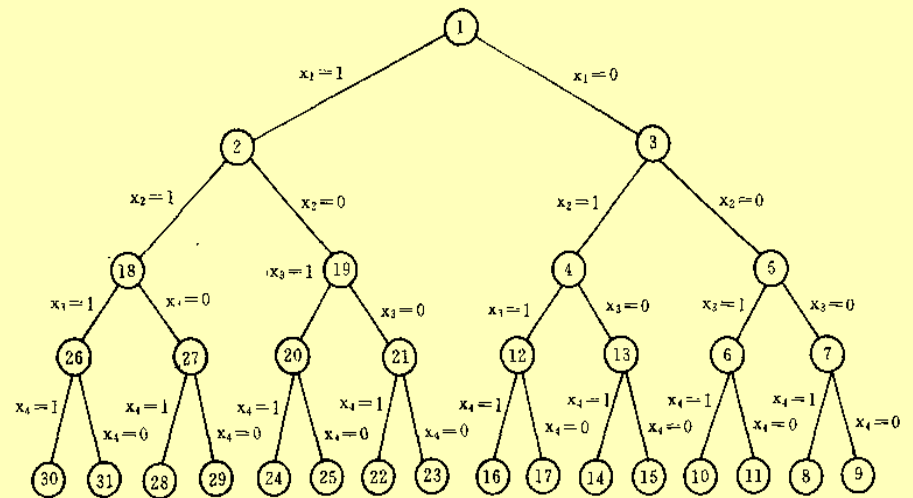
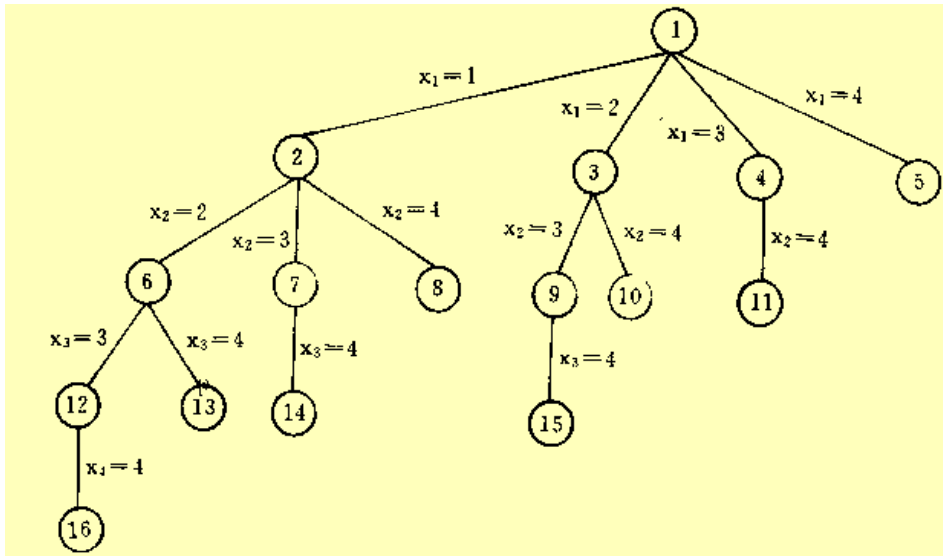
树边标记：由i级结点到i+1级结点的那些边用 x_i 的值来标记， $x_i=1$ 或0。



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。

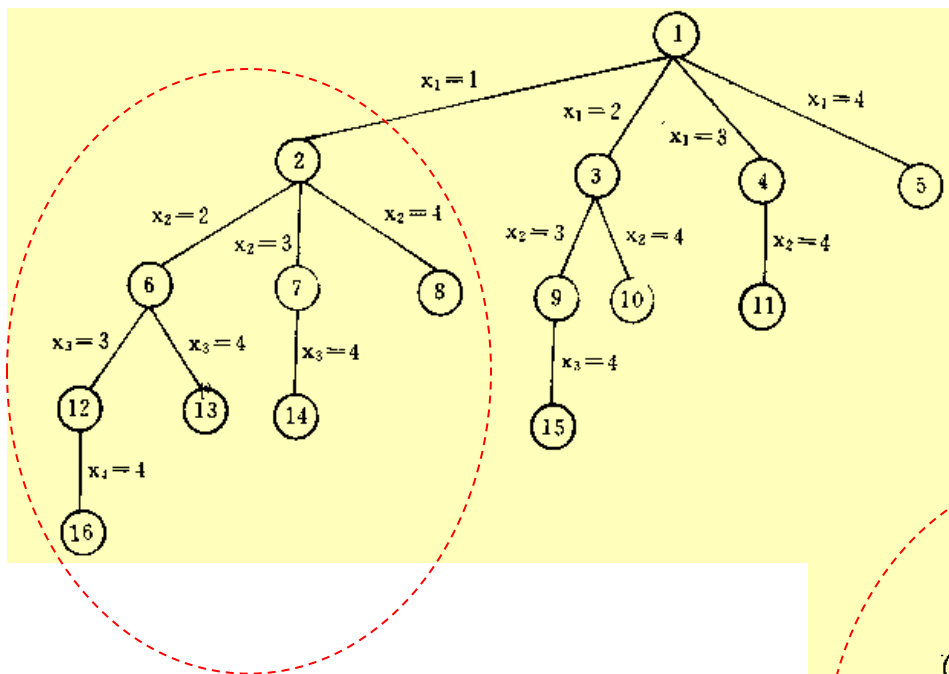
同一个问题的状态空间树可以有不同的形式。



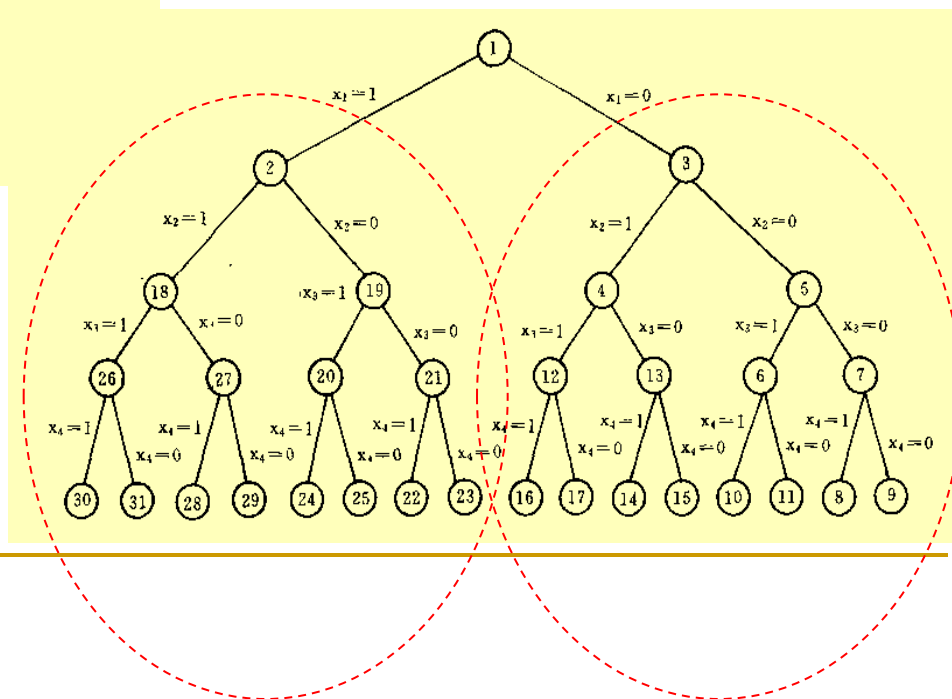
关于状态空间树的概念

- **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**)。

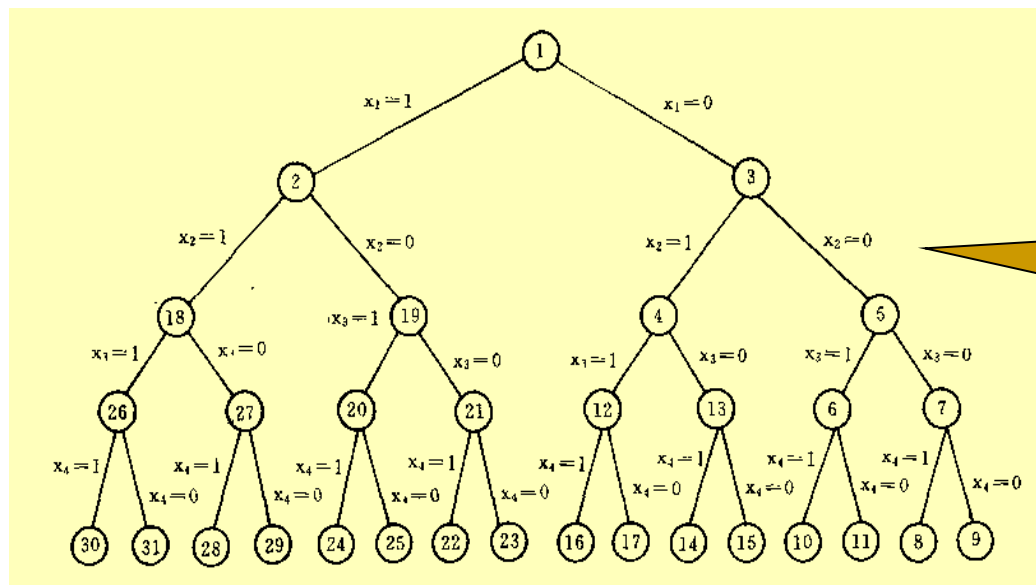
- **状态空间树的分解**: 在状态空间树的每个结点处, 解空间被分解为一些子解空间, 表示在一些分量取特定值情况下的解空间元素。



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

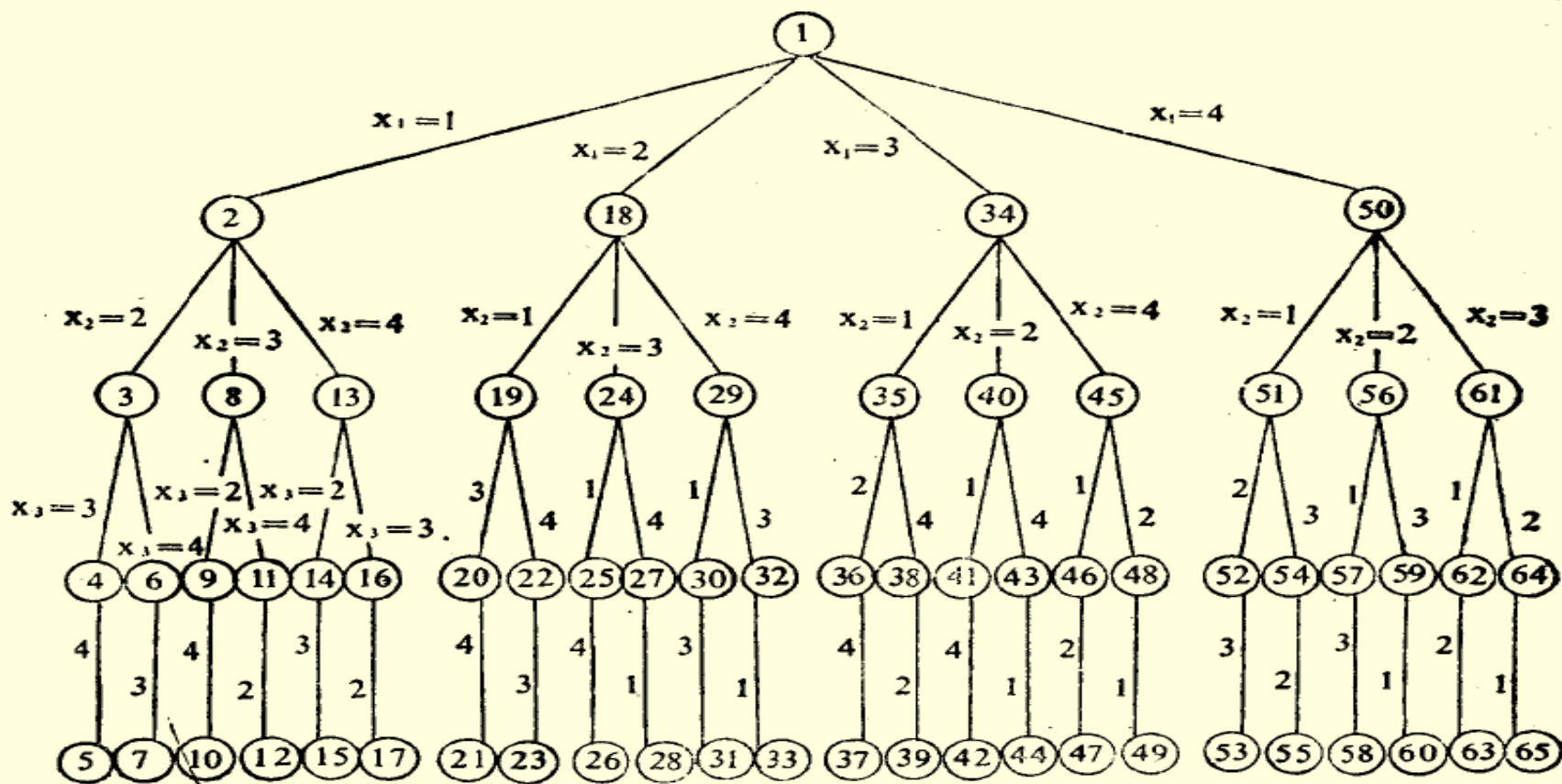
当前**E-结点R**一旦生成一个新的儿子**C**，**C**就变成一个新的**E-结点**，当完全检测了子树**C**之后，**R**结点再次成为**E-结点**。

2. 宽度优先策略

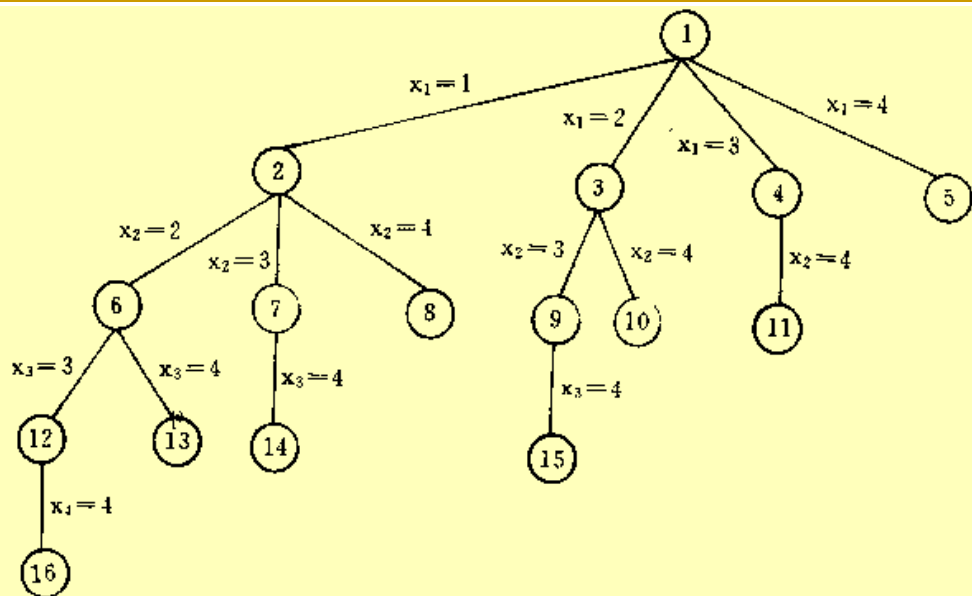
一个**E-结点**一直保持到变成死结点为止。

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。

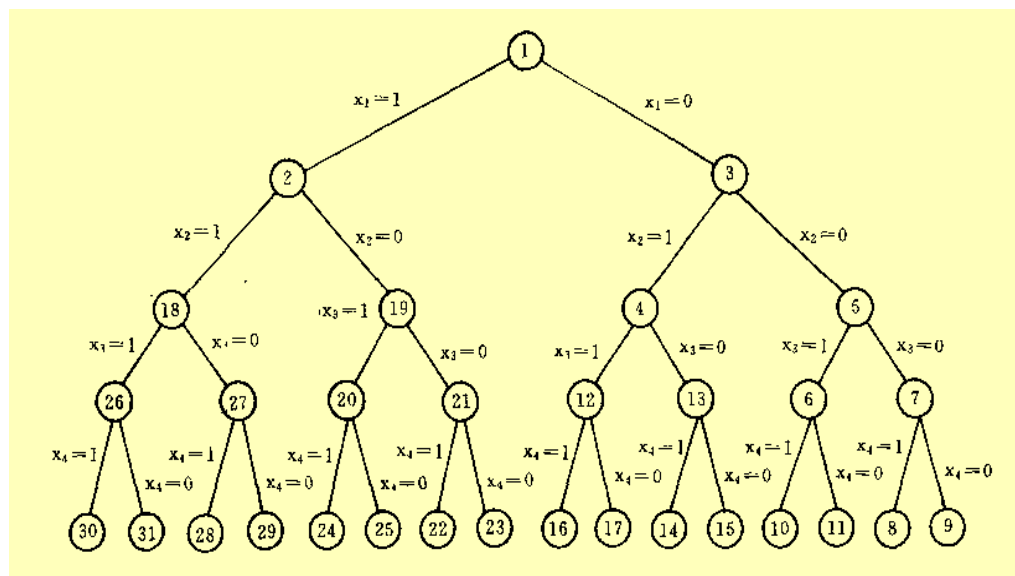
- **回溯法**：使用限界函数的深度优先结点生成方法称为**回溯法**（backtracking）
- **分枝-限界方法**：**E**结点一直保持到死为止的状态生成方法称为**分枝-限界方法**（branch-and-bound）



- 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

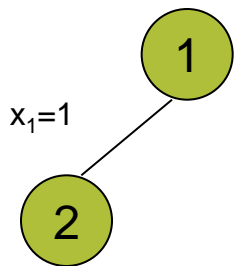
- 限界函数：如果 (x_1, x_2, \dots, x_i) 是到当前E结点的路径，那么 x_i 的儿子结点 x_{i+1} 是一些这样的结点，它们使得 $(x_1, x_2, \dots, x_i, x_{i+1})$ 表示没有两个皇后正在相互攻击的一种棋盘格局。
- 开始状态：根结点1，表示还没有放置任何皇后。
- 结点的生成：依次考察皇后1——皇后n的位置。

按照自然数递增的次序生成儿子结点。

1

根结点1，开始状态，唯一的活结点
解向量：（）

1			

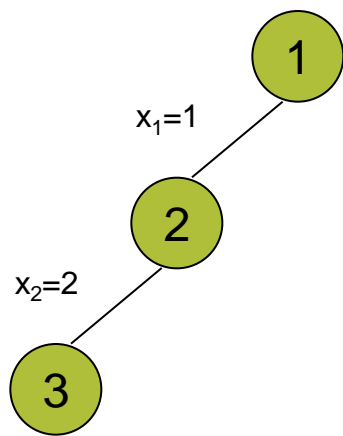


生成结点2，表示皇后1被放到第1行的第1列上，该结点是从根结点开始第一个被生成结点。

解向量：（1）

结点2变成新的E结点，下一步扩展结点2

1			
•	2		



B

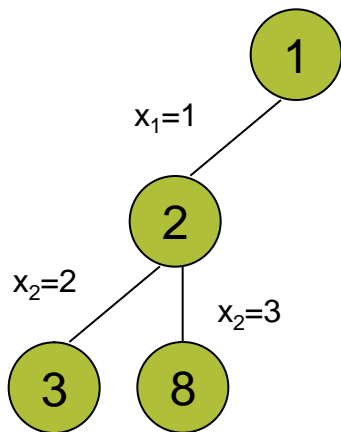
由结点2生成结点3，即皇后2放到第2行第2列。

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	



B

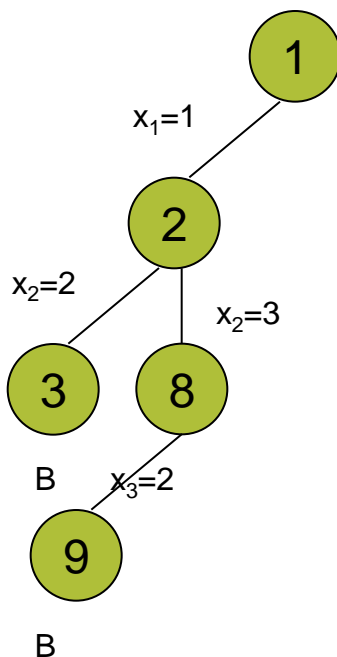
由结点2生成结点8，即皇后2放到第2行第3列。

结点8变成新的E结点。

解向量：（1，3）

从结点8继续扩展。

1			
.	.	2	
	3		



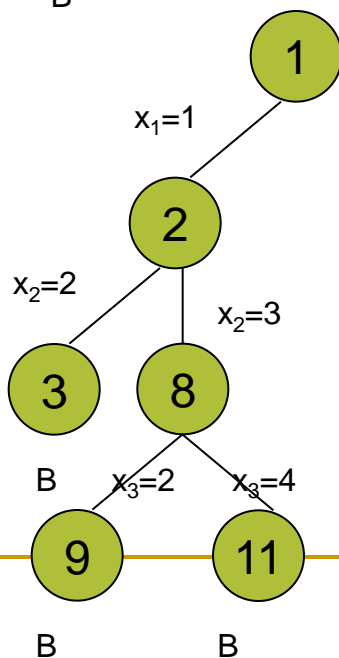
由结点8生成结点9，即皇后3放到第3行第2列。

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)

1			
.	.	2	
.	.	.	3



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。

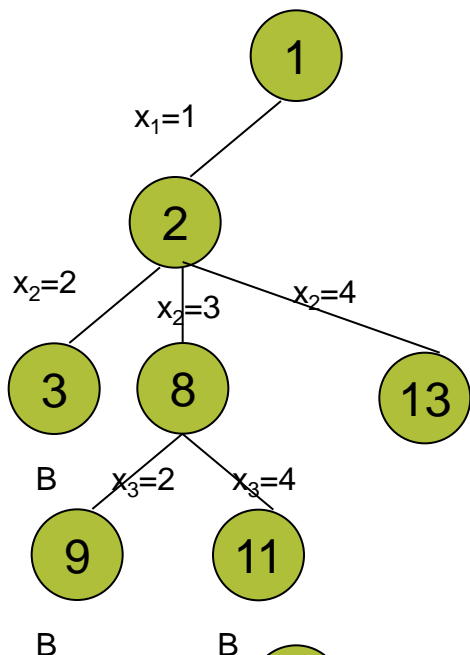
(结点12不会生成)

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。

1			
.	.	.	2



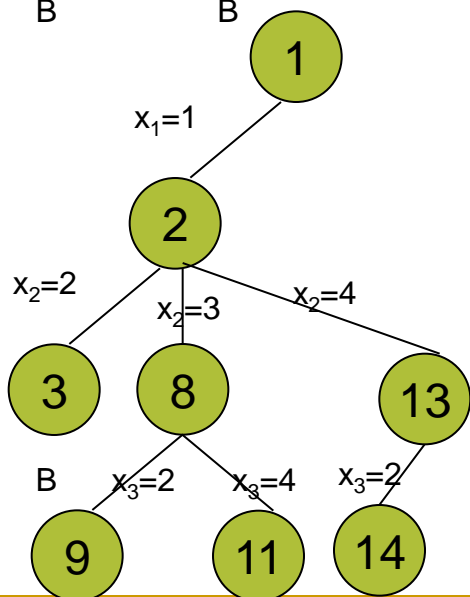
由结点2生成结点13，即皇后2放到第2行第4列。

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		



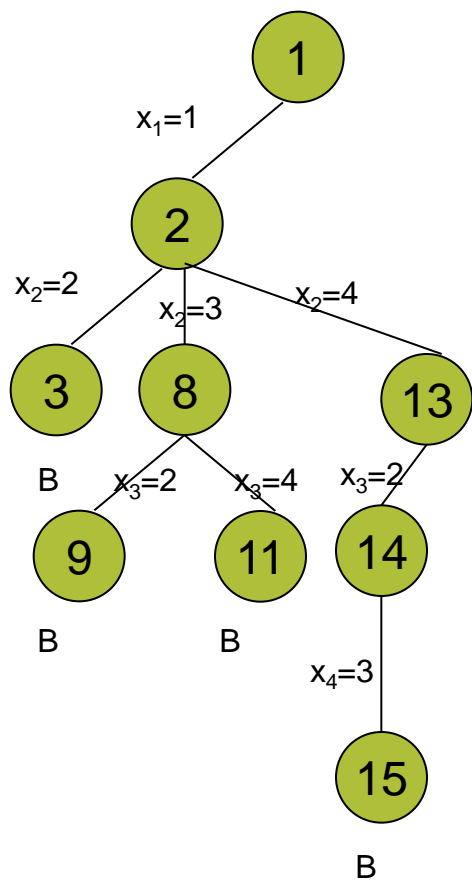
由结点13生成结点14，即皇后3放到第3行第2列。

结点14变成新的E结点。

解向量：(1, 4, 2)

从结点14继续扩展。

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。

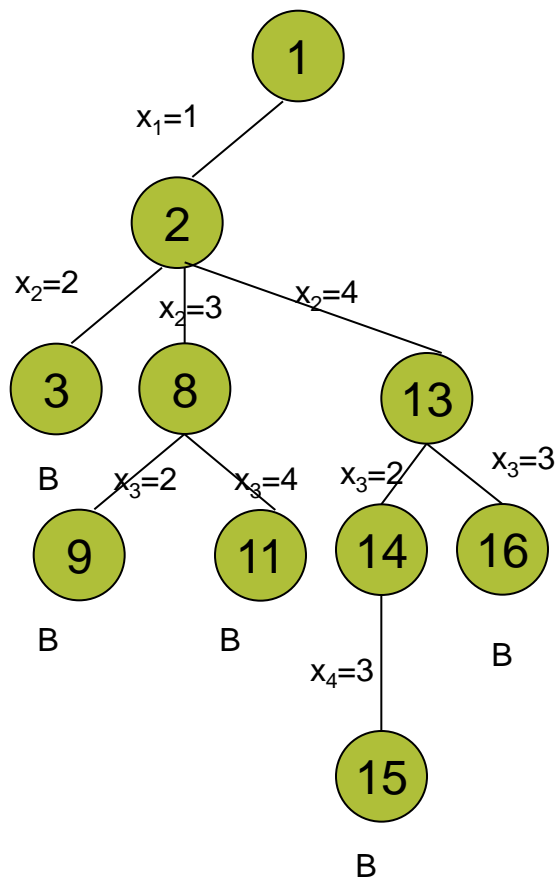
返回结点14，结点14不能导致答案结点，变成死结点，被杀死。

返回结点13继续扩展。

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。

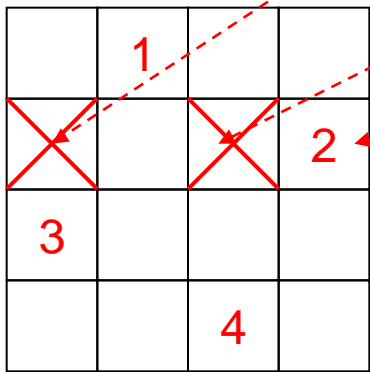
返回结点13，结点13不能导致答案结点，变成死结点，被杀死。

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。

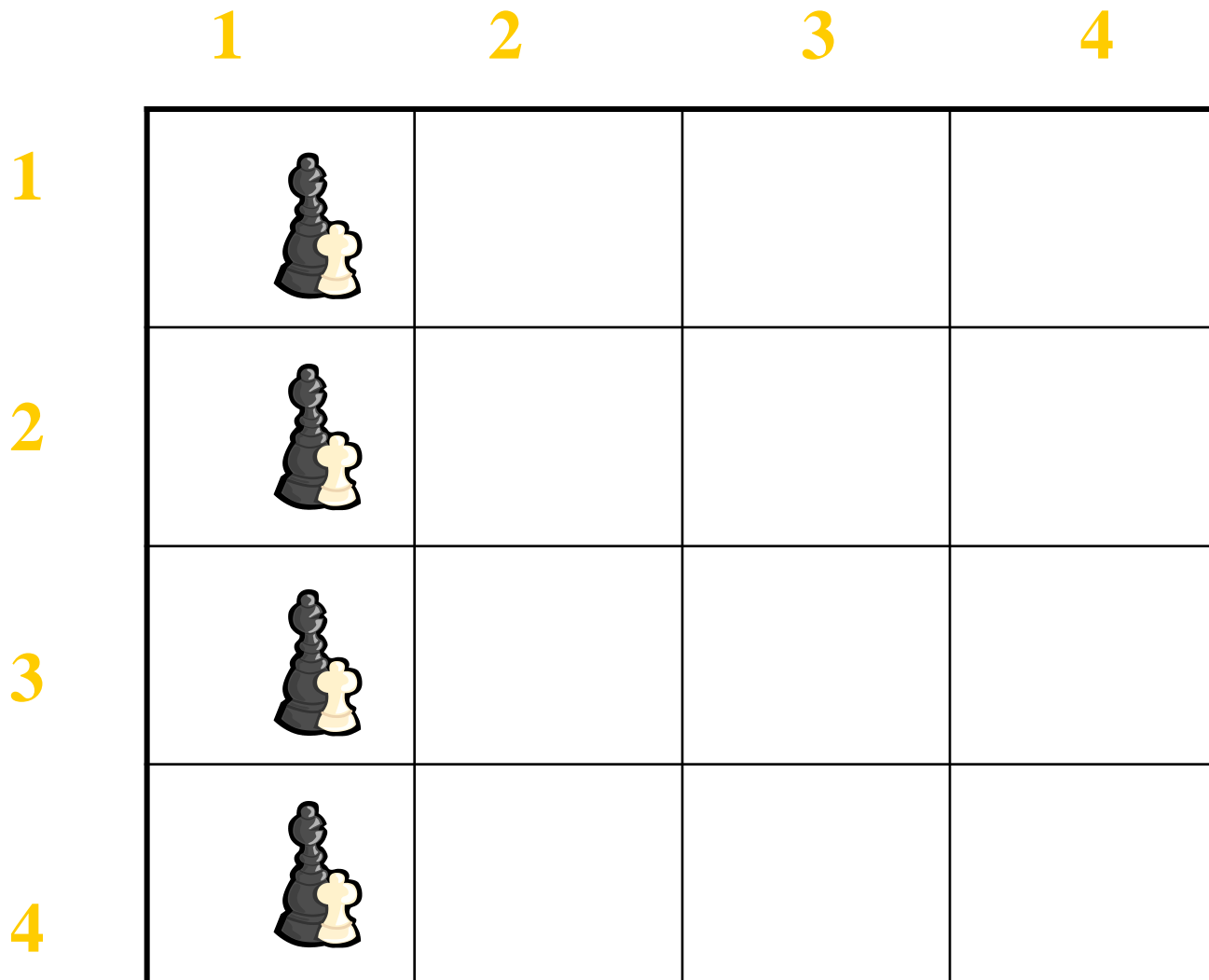
由结点1生成结点18，即皇后1放到第1行第2列。



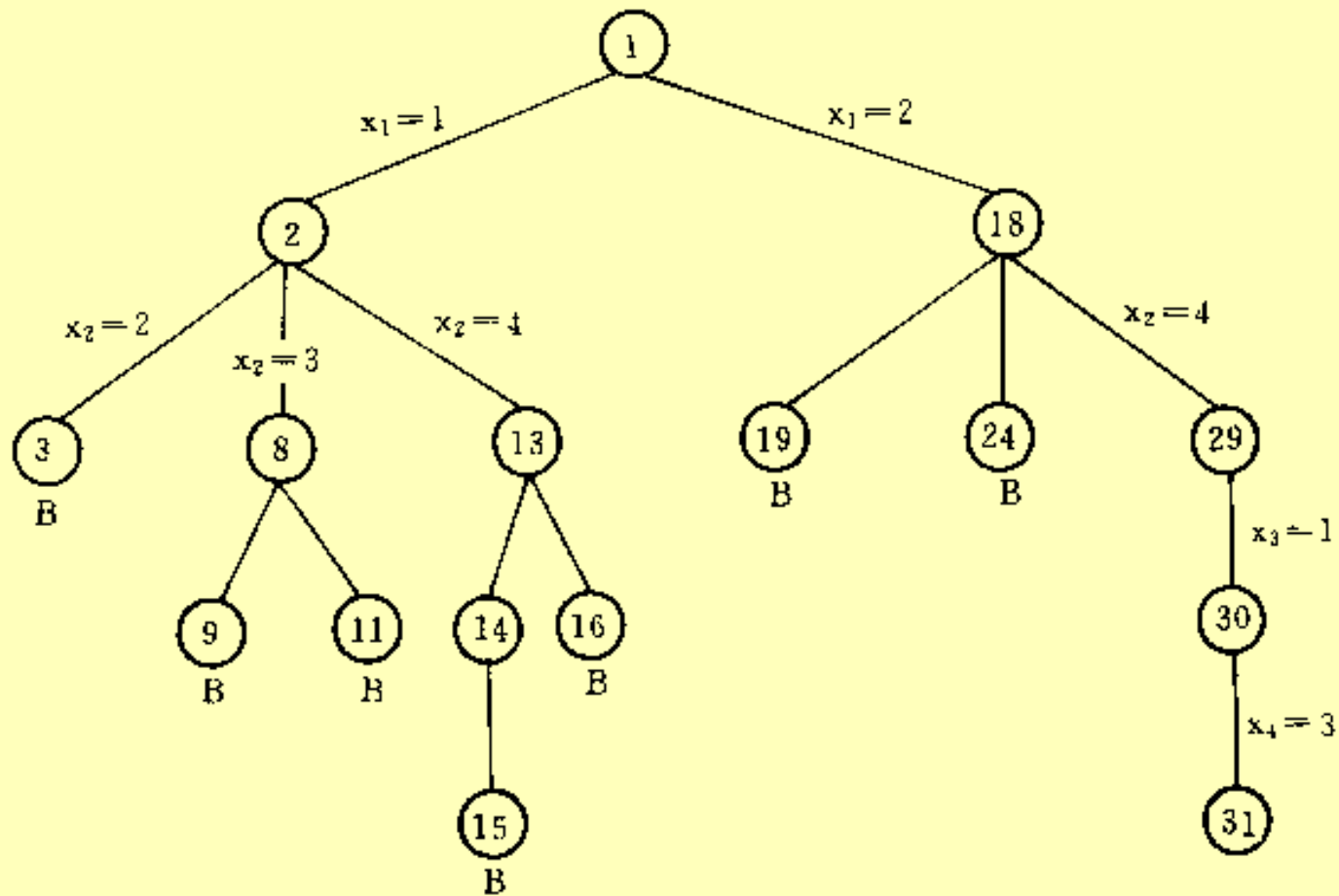
答案
结点

扩展结点30生成结点31，即皇后4放到第4行第3列。

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 $(x_1, x_2, \dots, x_{i-1})$ 是由根到一个结点的路径。
- $T(x_1, x_2, \dots, x_{i-1})$ 是下述所有结点 x_i 的集合，它使得对于每一个 x_i ， $(x_1, x_2, \dots, x_{i-1}, x_i)$ 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, \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。
- 解向量 $X(1:n)$ 中的每个 x_i 即是选自集合 $T(x_1, x_2, \dots, x_{i-1})$ 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**(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

```
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
      if  $(X(1), \dots, X(k))$  是一条已抵达一答案结点的路径
      then print( $X(1), \dots, X(k)$ ) 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。

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  $(X(1), \dots, X(k))$  是一条已抵达一答案结点的路径
```

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

```
      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 $X(1), \dots, X(k-1)$ 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。

第八章 回溯法

➤ 8.1 一般方法

8.1.1 回溯的一般方法

➤ 8.2 8皇后问题

➤ 8.3 子集和数问题

8.2 8-皇后问题

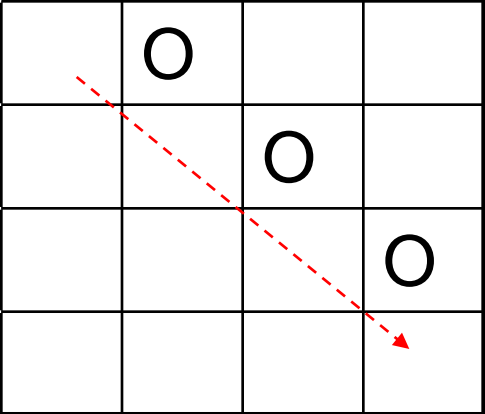
在一个 8×8 棋盘上放置8个皇后，且使得每两个之间都不能互相“攻击”，也就是使得每两个都不能在同一行、同一列及同一条斜角线上。

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

8.2 n-皇后问题

- n元组: (x_1, x_2, \dots, 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) 棋盘行、列用 $1 \dots n$ 编号
 - 2) 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
 - 3) 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

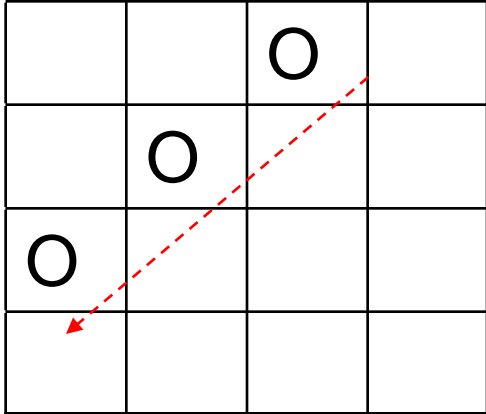
i \ j	1	2	3	4
1		O		
2			O	
3				O
4				



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

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

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：

$$i-j=k-l \text{ 或 } i+j=k+l$$

时，它们在同一条斜角线上。

即：

$$j-l = i-k \text{ 或 } j-l = k-i$$

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X(k)的当前值处：

- ① 不等于前面的 $X(1), \dots, X(k-1)$ 的值，且
- ② 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 //与前k-1个皇后做比较
```

```
    if X(i)=X(k) //在同一列上//
```

```
        or ABS(X(i)-X(k))=ABS(i-k) //在同一正或反斜角线上//
```

```
    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 $n \times n$ 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) \leftarrow 0; k \leftarrow 1 //k是当前行，X(k)是当前列//

while k>0 do //对所有的行执行以下语句//

X(k) \leftarrow X(k)+1 //移到下一列//

while X(k) \leq n and not PLACE(k) do //检查是否能放置皇后//

X(k) \leftarrow X(k)+1 //当前X(k)列不能放置，后推一列//

repeat

if X(k) \leq 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 \leftarrow k+1; X(k) \leftarrow 0 //否，转下一皇后//

endif

else k \leftarrow k-1

endif

repeat

end NQUEENS

第八章 回溯法

- 8.1 一般方法

 - 8.1.1 回溯的一般方法

- 8.2 8皇后问题

- 8.3 子集和数问题

8.3 子集和数问题

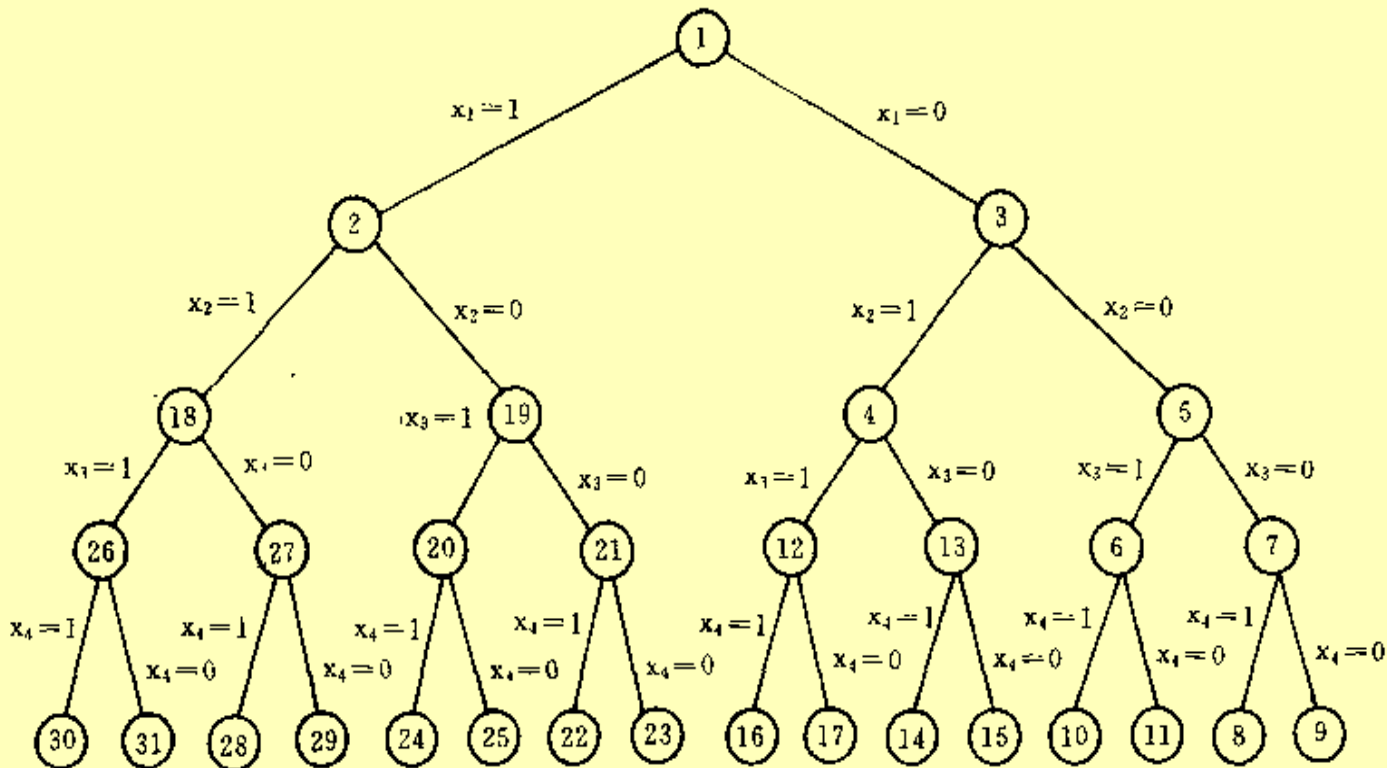
已知 $n+1$ 个正数 w_1, w_2, \dots, w_n 和 M 。要求找出 w_i 的和数等于 M 的所有子集。

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：

- 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 $(0, 0, 1, 1)$

状态空间树

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。

子集和数问题的表示

- 元组大小固定：n元组 (x_1, x_2, \dots, 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 $B(X(1), \dots, X(k)) = \text{true}$

注：如果不满足上述条件，则 $X(1), \dots, X(k)$ 根本不可能导致一个答案结点。

子集和问题的递归回溯算法

procedure SUMOFSUB(s,k,r)

//s:1至k-1已放入数之和, r: k+1至n的数之和

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

global boolean X(1:n) , real r,s; integer k,j

X(k)←1 //放入k, 生成左儿子, $B_{k-1}=\text{true}, s+W(k) \leq M$ //

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 $B_k=\text{true}$ //

call SUMOFSUB(s+W(k),k+1,r-W(k))

endif

endif

//生成右儿子, 计算 B_k 的值//

if s+r-W(k)≥M and s+W(k+1)≤M //确保 $B_k=\text{true}$ //

then X(k)←0

call SUMOFSUB(s,k+1,r-W(k))

endif

end SUMOFSUB

//W(i)按非降次序排列,

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

$$W(1) \leq M, \quad \sum_{i=1}^n W(i) \geq M //$$

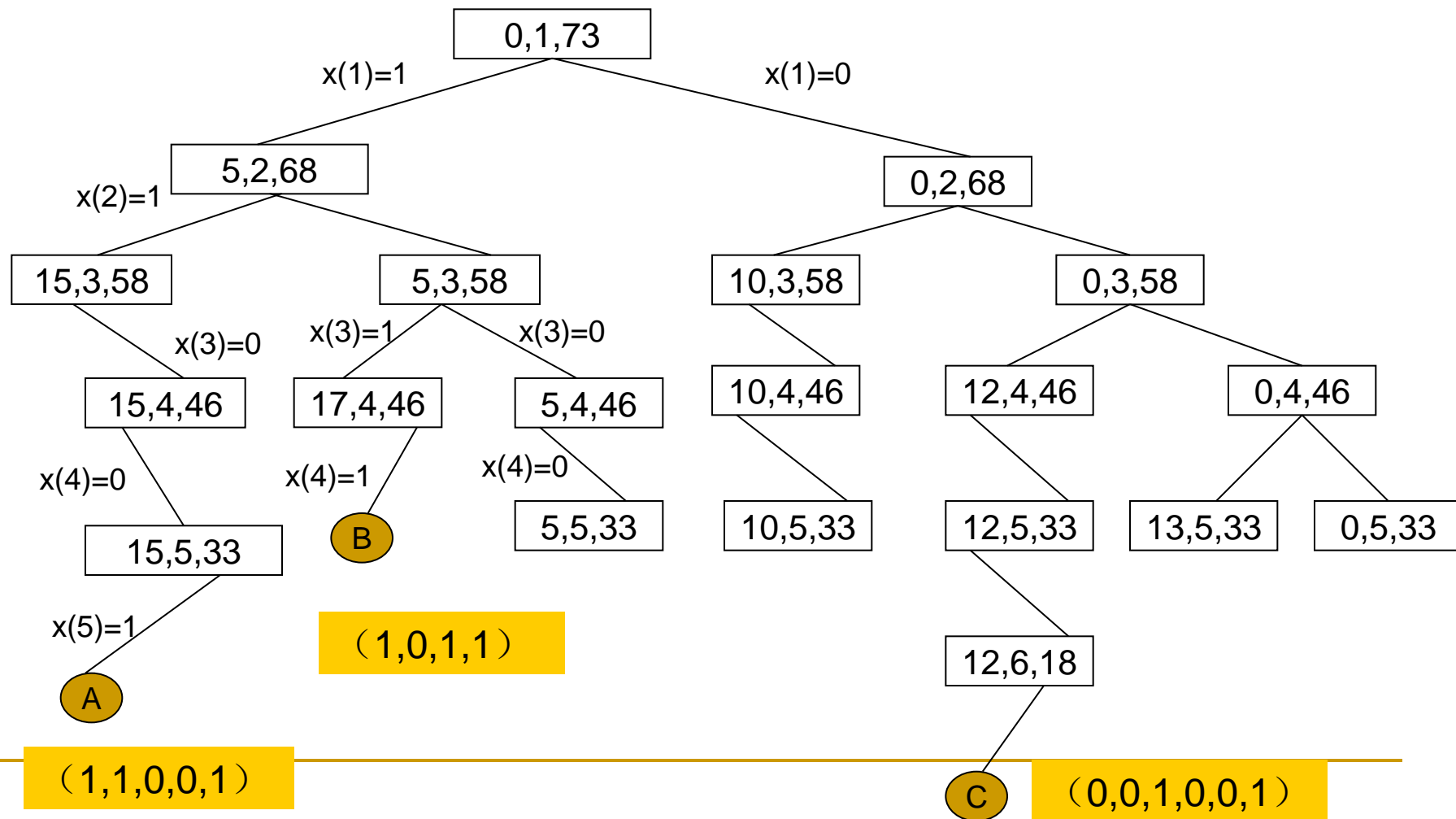
左子树, 放入k, 看k能否放满

左子树, k已放入且未放满, 看k+1是否满足条件

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



作业

- 8.7 8.8 8.22