



第16章 回溯

例16.1(8-皇问题)

- 在 8×8 的棋盘上放置8个皇后, 使得这8个皇后不在同一行、同一列及同一斜角线上. 如下图16.1:

	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-皇后问题的形式化

- 8皇后问题的解可以表示为一个8-元组 (x_1, \dots, x_8) , 其中 x_i 是放在第 i 行的皇后所在的列号, 则8皇后问题可形式化为在 8^8 个8-元组中找满足以下约束条件的元组:

$$x_i \neq x_j \text{ for all } i, j$$

$$|x_i - x_j| \neq |j - i|$$

- 这 8^8 个8-元组构成的集合称为8皇后问题的解空间-搜索范围.
- 如果将约束条件之一,任意两个皇后不在同一列上, 加入到元组的定义中, 这时每个8-元组为 $(1, 2, 3, 4, 5, 6, 7, 8)$ 的一个排列, 解空间的大小由 8^8 个元组减少到 $8!$ 个元组.
- 解空间不是唯一的, 取决于算法的设计.
- 设计解空间时还要考虑生成解空间的算法的复杂度; 在8皇后问题中, 如果加入第二个条件, 解空间很难生成.

16.1 搜索问题的形式化-解空间

- 假定问题的解能用 n -元组 (X_1, \dots, X_n) 表示,其中 X_i 取自某个有穷集 S_i .
- 这些 n -元组构成的集合称为问题的解空间. 假设 $|S_i|=m_i$, 则解空间的大小为 $m=m_1m_2\dots m_n$.
- 我们考虑两类问题:
 - 存在性问题是: 求满足某些条件的一个或全部元组, 如果存在这样的元组算法应返回Yes, 否则返回No. 这些条件称为约束条件.
 - 优化问题: 给定一组约束条件, 在满足约束条件的元组中求使某目标函数达到最大(小)值的元组. 满足约束条件的元组称为问题的可行(feasible)解.



16.1 搜索问题的形式化

- 解决这类问题的最一般方法是使用搜索技术,即系统化的搜索解空间的技术. 本章介绍回溯法,下章介绍分支限界法.

例16.2：子集和数问题

- 已知 $n+1$ 个正数： $w_i, 1 \leq i \leq n$, 和 M . 要求找出 $\{w_i \mid 1 \leq i \leq n\}$ 的所有子集, 使得子集内元素之和等于 M .
- 例如: $n=4, (w_1, w_2, w_3, w_4)=(11, 13, 24, 7), M=31$. 则满足要求的子集是 $(11, 13, 7)$ 和 $(24, 7)$.
- 我们可以用 w_i 的下标 i 构成的元组表示一个解, 则这两个解可表示为 $(1, 2, 4)$ 和 $(3, 4)$.
- 元组 $(1, 2, 4)$ 和 $(2, 1, 4)$ 代表同一子集, 为此限制元组分量按升序排列, 即不考虑元组 $(2, 1, 4)$.
- 还可以用其他方式表示一个解, 如下:



子集和数问题的另一种表示

- 每个子集由n-元组 (x_1, \dots, x_n) 表示, 其中 $x_i \in \{0, 1\}, 1 \leq i \leq n$, 表示:

- $x_i=0$, 表示没有选择 w_i ;

- $x_i=1$, 表示选择 w_i ;

则例中的解可以表示为 $(1, 1, 0, 1)$ 和 $(0, 0, 1, 1)$



解空间的状态空间树

- 任何搜索算法都可以用建立在解空间上的状态空间树加以描述.
- 状态空间树是我们尝试选择元组的各个分量时产生的树结构.
- 搜索算法并非事先将状态空间树存在计算机内再进行遍历,而是通过展开状态空间树来找所求的解.
- 展开过程中通过使用启发式的限界方法(剪去状态空间树上的某些分枝)使搜索算法只展开状态空间树的一部分,从而降低搜索算法的时间和空间复杂度.

例16.3: n -皇后问题

- n -皇后问题是8-皇后问题的推广. n 个皇后将被放置在 $n \times n$ 的棋盘上且使得没有两个皇后可以互相攻击.这是8-皇后问题的推广,其解空间由 n -元组 $(1, 2, \dots, n)$ 的 $n!$ 个排列所组成.
- 其状态空间树如图6.2所示($n=4$).树的边由 x_i 的可能的取值标记.由 i 级到 $i+1$ 级节点的边给出 x_i 的值.这种树称为排列树.
- 从根节点到叶节点的一条路径对应解空间的一个元组.

图16.2 4-皇后问题的状态空间树

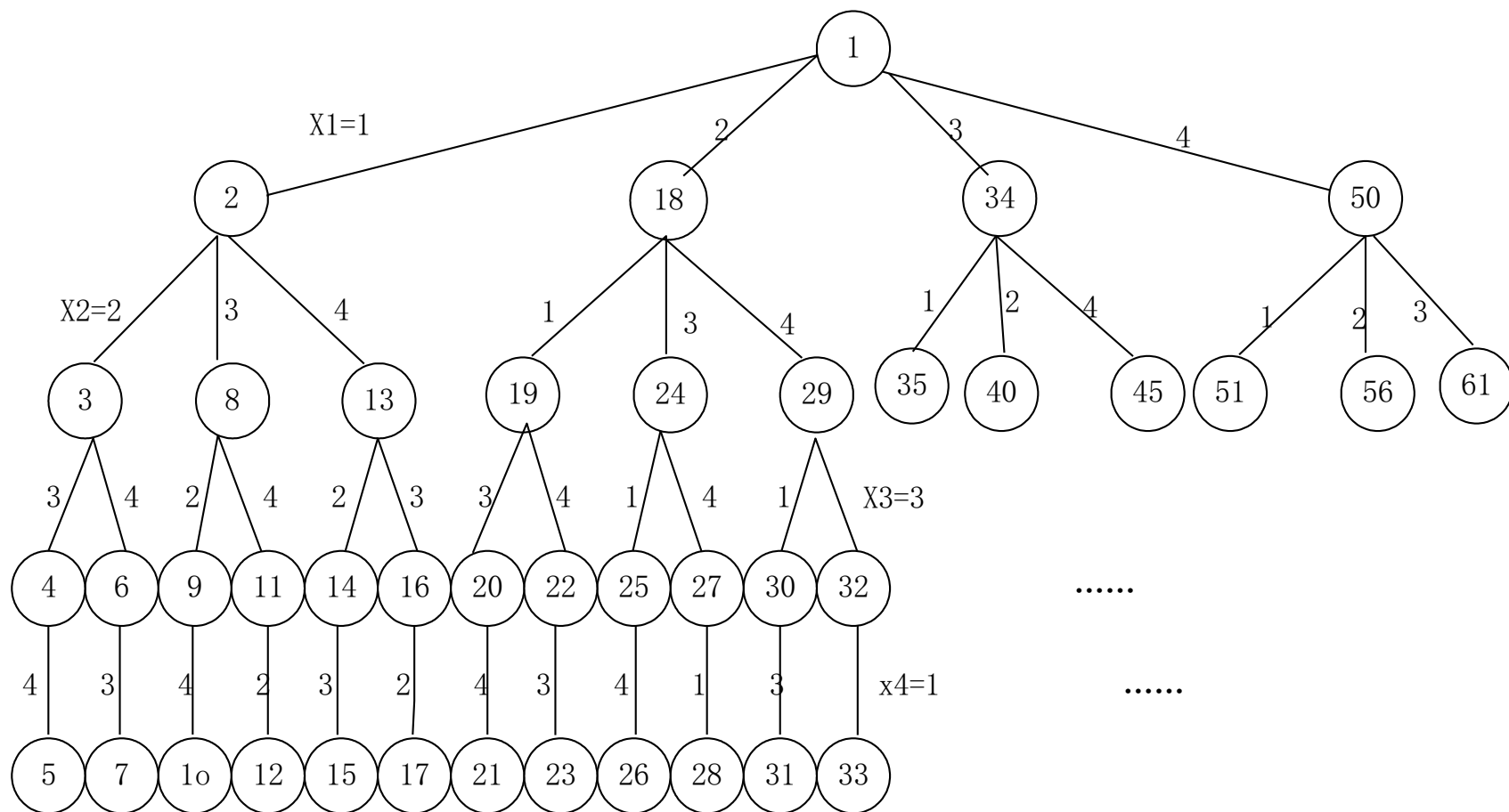


图16.3子集和数问题的状态空间树

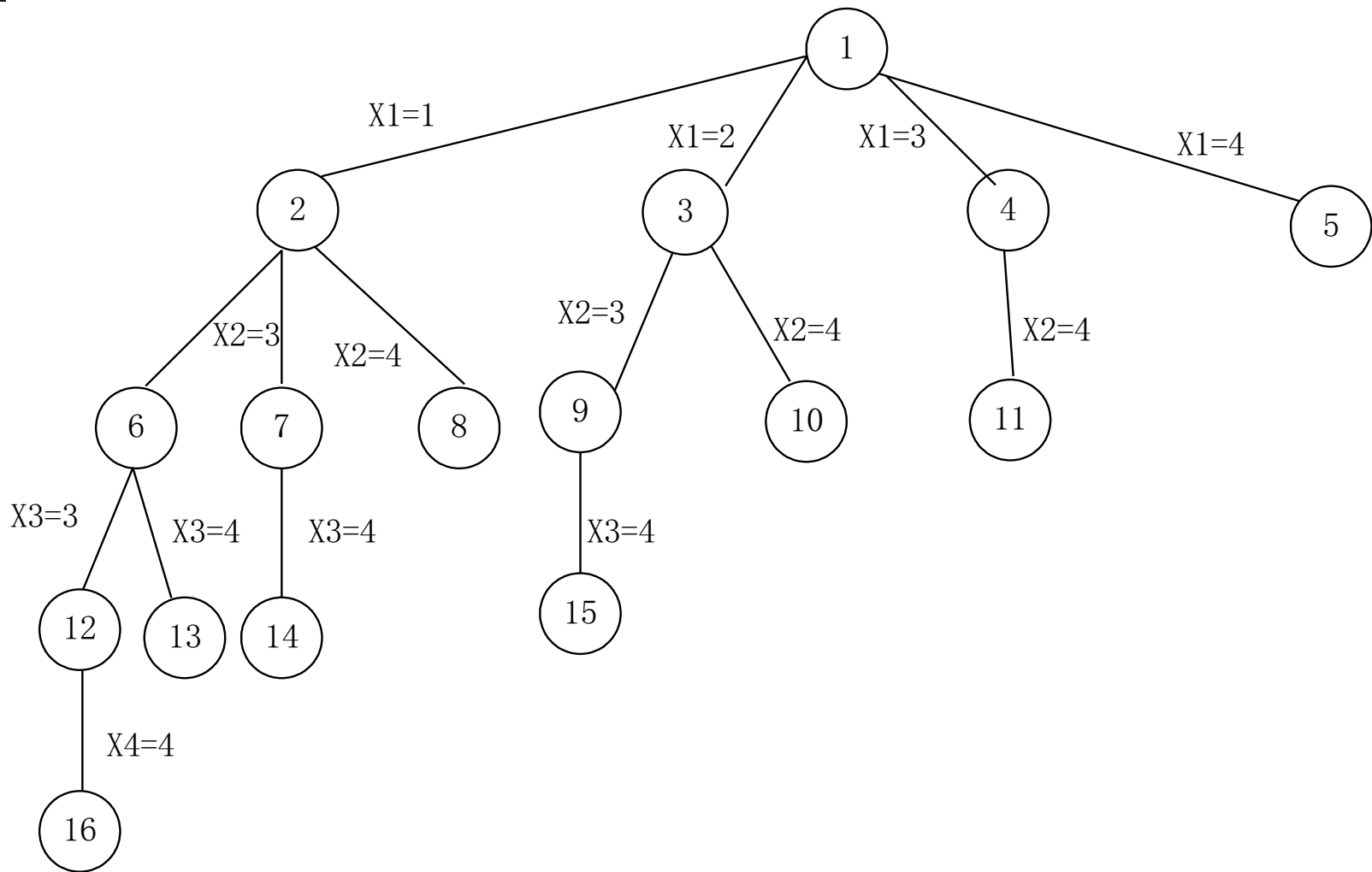
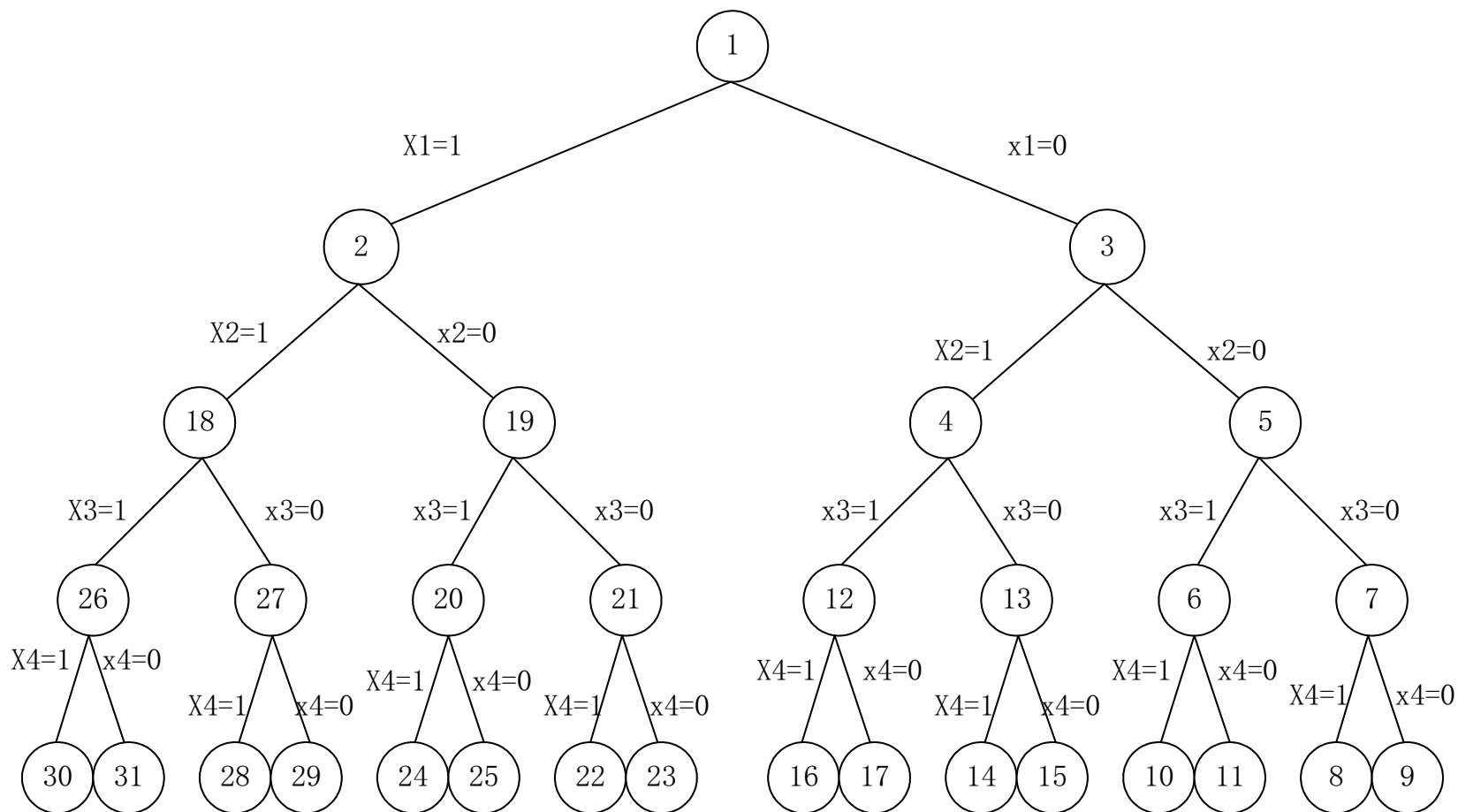


图16.4 子集和数问题的状态空间树



有关状态空间树的术语

- 状态空间树的每个结点代表问题求解过程中达到的一个状态,根节点到它的路径代表对一些分量已作出的选择. 状态空间树的所有节点构成的集合称为求解该问题的状态空间.
- 根节点到状态空间树的一个节点X的路径可以表示为 $(x(1), \dots, x(k))$, 其中 $x(i)$, $1 \leq i \leq k$, 为搜索过程中已经选择的分量. 今后我们也用这个元组标识该节点:

$$X = (x(1), \dots, x(k)).$$

- $(x(1), \dots, x(k))$ 也对应一个子问题, 即在后 $n-k$ 个元组分量所对应的子空间上找满足要求的解. 该子空间是状态空间树中以X为根的子树. 所以也称节点X为问题节点.
- 如果从根节点到节点S的那条路径确定了解空间的一个元组, 则称S为状态空间树的一个解节点.
- 如果一个解节点S(代表的元组)满足约束条件称其为答案节点.



状态空间树的展开方法

- 每个搜索算法都是一种系统地展开状态空间树的算法.
- 我们用以下术语描述展开状态空间树的各种方法.
- 活结点: 已展开了部分子节点, 但所有子结点尚未全部展开的结点.
- 死结点: 被限界或已展开了所有子结点的结点.
- E-结点: 当前正在展开子结点的活结点.



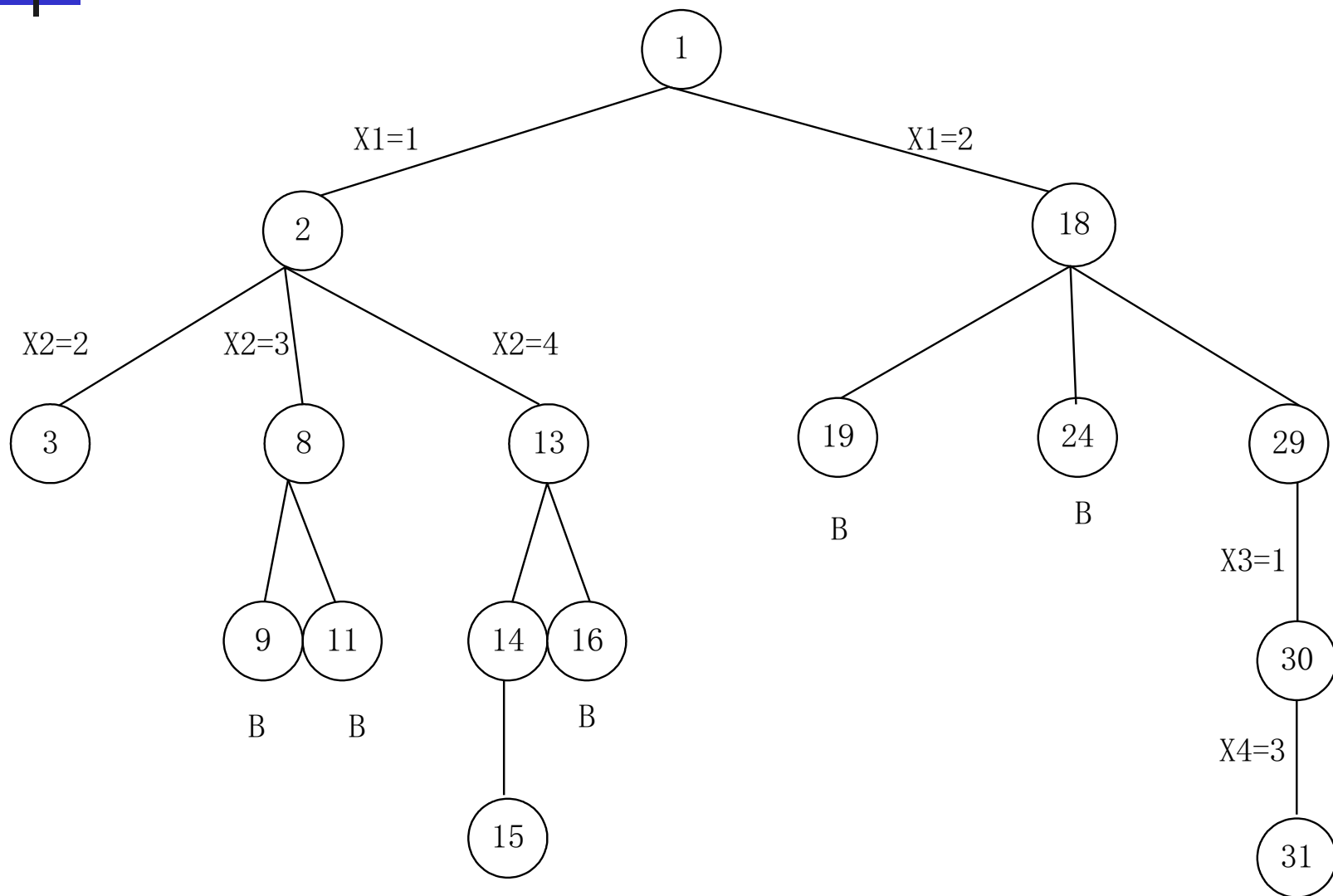
状态空间树的展开方法

- 深度优先展开方法: 一个E-结点展开自己的一个子结点后, 就让该子结点成为E-结点的展开方法 (相当于对状态空间树做深度优先搜索), 称为深度优先展开方法.
- 回溯法: 加限界的深度优先展开状态空间树的方法称为回溯法.
- 分枝—限界法: 一个结点一旦成为E-结点则它将展开其全部子节点, 之后自己变成死结点. 这样的展开状态空间树的方法称为分枝—限界法.
- 在分枝—限界法中要维持一个活结点表的结构, 存放已展开但还未成为E结点的那些结点.

限界

- 设 $A=(x(1),x(2),\dots,x(i-1))$ 是状态空间树中一个节点. 以后我们用 $T(x(1),x(2),\dots,x(i-1))$ 表示 $x(i)$ 所有可能取值的集合.
- 设 $x_i \in T(x(1),x(2),\dots,x(i-1))$ 则 $(x(1),\dots,x(i))$ 是状态空间树上 A 的一个子节点.
- 限界函数 $B_i(x(1),\dots,x(i))$ 表示一个条件:
 $B_i(x(1),\dots,x(i))$ 为true当且仅当从 $(x(1),\dots,x(i))$ 展开不能得到一个答案结点.
- 这时搜索算法停止展开结点 $(x(1),\dots,x(i))$ 及其子树, 称为限界或 “kill”这个节点.
- 限界函数从分析约束条件或优化的要求得到.
限界函数必须计算简单.

图6.6 用回溯法解4-皇后问题





回溯的一般方法

算法说明:

- 每个解用数组 $X(1:n)$ 来表示.
- 假定 $X(1), \dots, X(k-1)$ 的值已确定,
 $T(X(1), \dots, X(k-1))$ 代表 $x(k)$ 的所有可能的取值.
- 限界函数 $B(X(1), \dots, X(k))$ 判断那些 $X(k)$ 的取值不能导致问题的解, 从而停止展开该子节点.

回溯方法描述（续）

BACKTRACK:

$k \leftarrow 1$

while $k > 0$ do

{

if $(X(1), \dots, X(k-1))$ 还有没展开的子节点

{取 $X(k) \in T(X(1), \dots, X(k-1))$;

if $B(X(1), \dots, X(k)) = \text{false}$

{if $(X(1), \dots, X(k))$ 是一答案节点

则输出 $(X(1), \dots, X(k))$;

$k \leftarrow k+1$ } // 继续展开子节点 //

else $k \leftarrow k-1$ // 回溯到父节点 //

}

递归形式的回溯算法

■ RBACKTRACK(k)

//进入该子程序时解 $X(1:n)$ 的前 $k-1$ 个分量
 $X(1), \dots, X(k-1)$ 已取值//

for $X(k) \in T(X(1), \dots, X(k-1))$ and

$B(X(1), \dots, X(k)) = \text{false}$

{

if $(X(1), \dots, X(k))$ 是答案节点 {

 输出 $(X(1), \dots, X(k))$;

 RBACKTRACK($k+1$)}

}

16.2子集和数的问题

- 给定 n 个正数 $W(i)$ 和另一个正数 M . 找出 $\{W(i), i=1, \dots, n\}$ 中所有使得和数等于 M 的子集. 该问题称为子集和数的判定问题.
- 最大子集和数问题: 找不超过 M 的和数最大的子集.
- 当 $W(i)$ 和 M 为正整数时前者有解当且仅当后者有解.
- 最大子集和数问题是背包问题的特例: $W(i)=p(i)$.
- 用长度固定的元组来设计一种回溯算法. 这时, 解向量的元素 $X(i)$ 取1或0值, 表示子集是否包含 $W(i)$.
- 对于图16.4中第 i 级上的结点, 左子节点对应于 $X(i)=1$, 右子节点对应于 $X(i)=0$.
- 限界函数的一种简单选择是: 当且仅当

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

时, $B(X(1), \dots, X(k)) = \text{true}$, 并停止展开节点 $(X(1), \dots, X(k))$.

强化限界函数

- 如果已将 $W(i)$ 按非降次序排列, 则可以进一步限界如下: 如果

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

则继续展开 $X(1), \dots, X(k)$ 不可能得到答案结点.

- 限界函数 $B(X(1), \dots, X(k)) = \text{ture}$ 当且仅当

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

- 这时停止产生子节点 $(X(1), \dots, X(k))$ 及其子树.

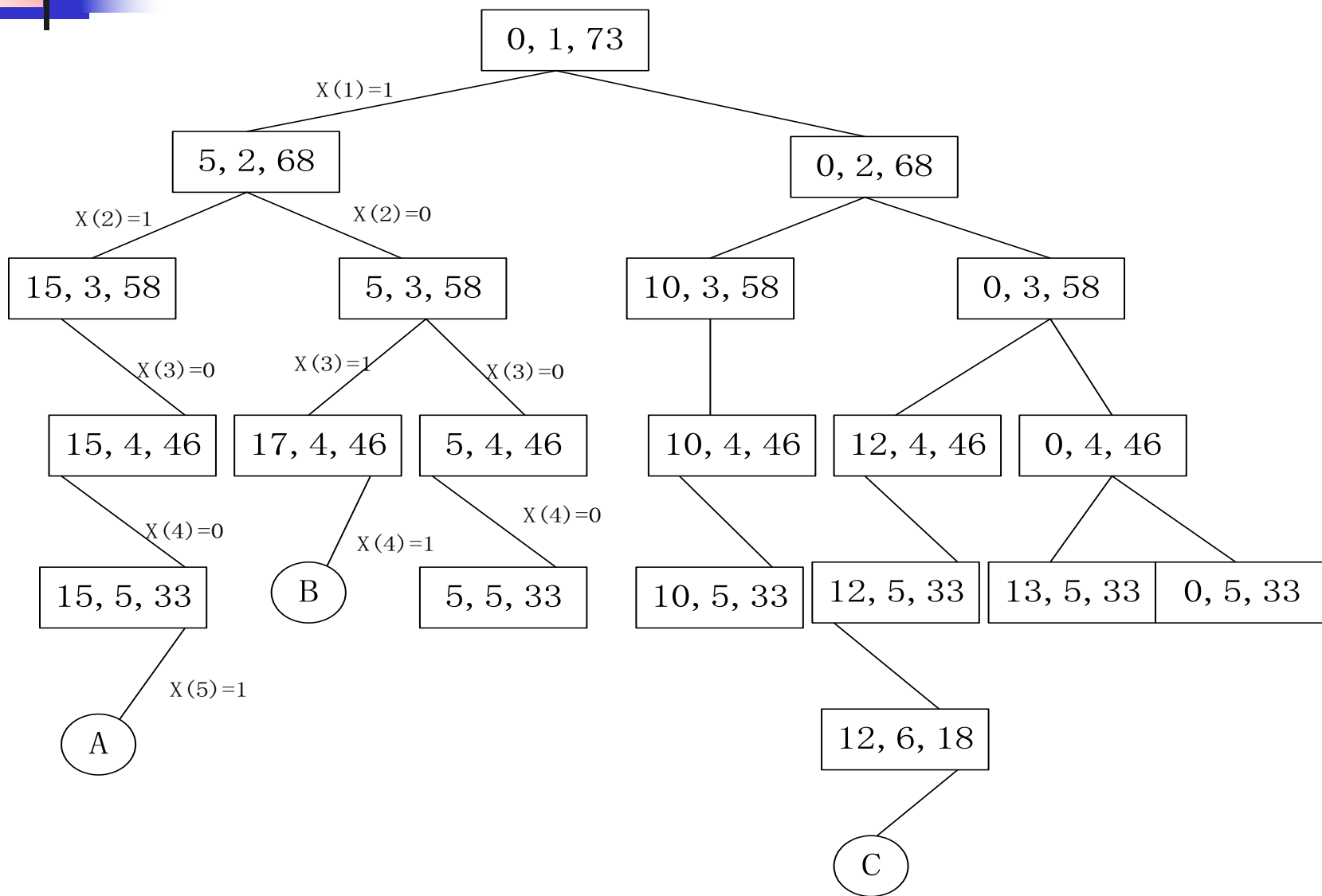
子集和数问题的回溯法

- 令 $S = w(1)X(1) + \dots + w(k-1)X(k-1)$
 $r = w(k) + \dots + w(n)$, 假定 $S + r \geq M$ (不满足该条件的节点已被限界掉)
- 如果 $S = M$, 则找到了一个和数为 M 的子集. 回溯找其它解.
- 展开左子节点:
 - 如果 $S + W(k) + W(k+1) > M$ 则停止展开左子节点,
 $r \leftarrow r - w(k)$, 并展开右子节点;
 - 否则, $X(k) \leftarrow 1$,
 $S \leftarrow S + w(k)$,
 $r \leftarrow r - w(k)$, $k \leftarrow k + 1$ (令 $(x(1), \dots, x(k))$ 为 E 节点);
- 展开右子节点:
 - 如果 $S + r < M$ 或 $S + w(k+1) > M$ 则停止展开右子节点并回溯;
 - 否则, $X(k) \leftarrow 0$,
 $r \leftarrow r - w(k)$, $k \leftarrow k + 1$ (令 $(x(1), \dots, x(k))$ 为 E 节点);
- 回溯: $k \leftarrow k - 1$ (回到父节点).

例16.6

- 图16.9给出了 $n=6, M=30$ 和 $W(1:6)=(5,10,12,13,15,18)$ 时回溯法展开的部分状态空间树.
- 图中矩形结点内的数字是 s, k, r 的值.
- 圆形结点表示一个答案节点,其对应的子集和数= M . 这些节点是:
 $A=(1,1,0,0,1,0), B=(1,0,1,1,0,0)$ 和
 $C=(0,0,1,0,0,1)$.
- 图6.9中的树只有20个矩形结点,而 $n=6$ 的满状态空间树有 $2^6-1=63$ 个节点.

图16.9 , $W(1:6)=(5,10,12,13,15,18), M=30$

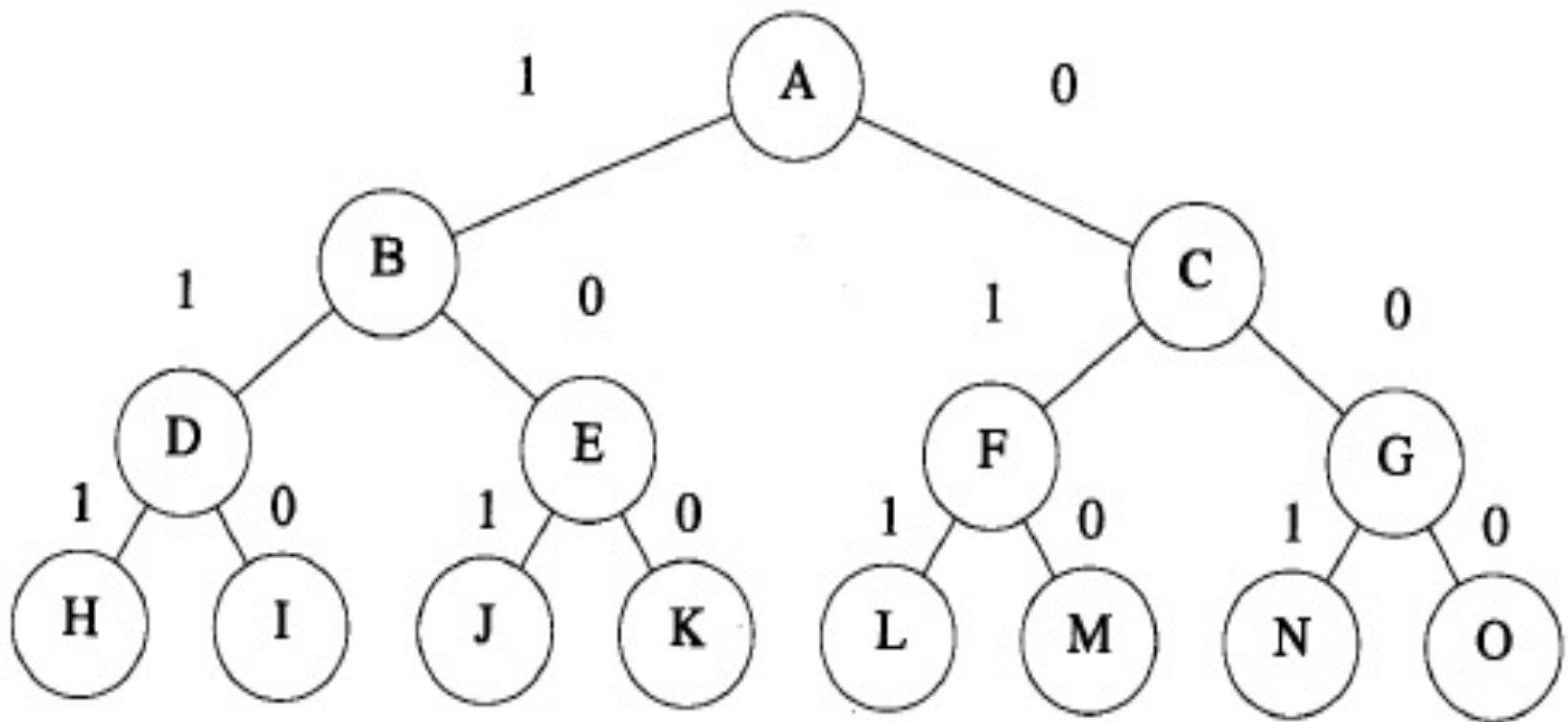




例16-2 [0/1背包问题]

- 考察如下背包问题: $n=3, w=[20, 15, 15], p=[40, 25, 25]$ 且 $c=30$.

图16-2 背包问题的状态空间树





例16-3 [旅行商问题]

- 给定一个 n 节点的网络, 称一条包含网络中 n 个节点的环路为一条周游路线.
- 旅行商问题要求找出一条最小成本的周游路线.
- 可以说是算法中最难的问题!!

图16-4 一个四节点网络

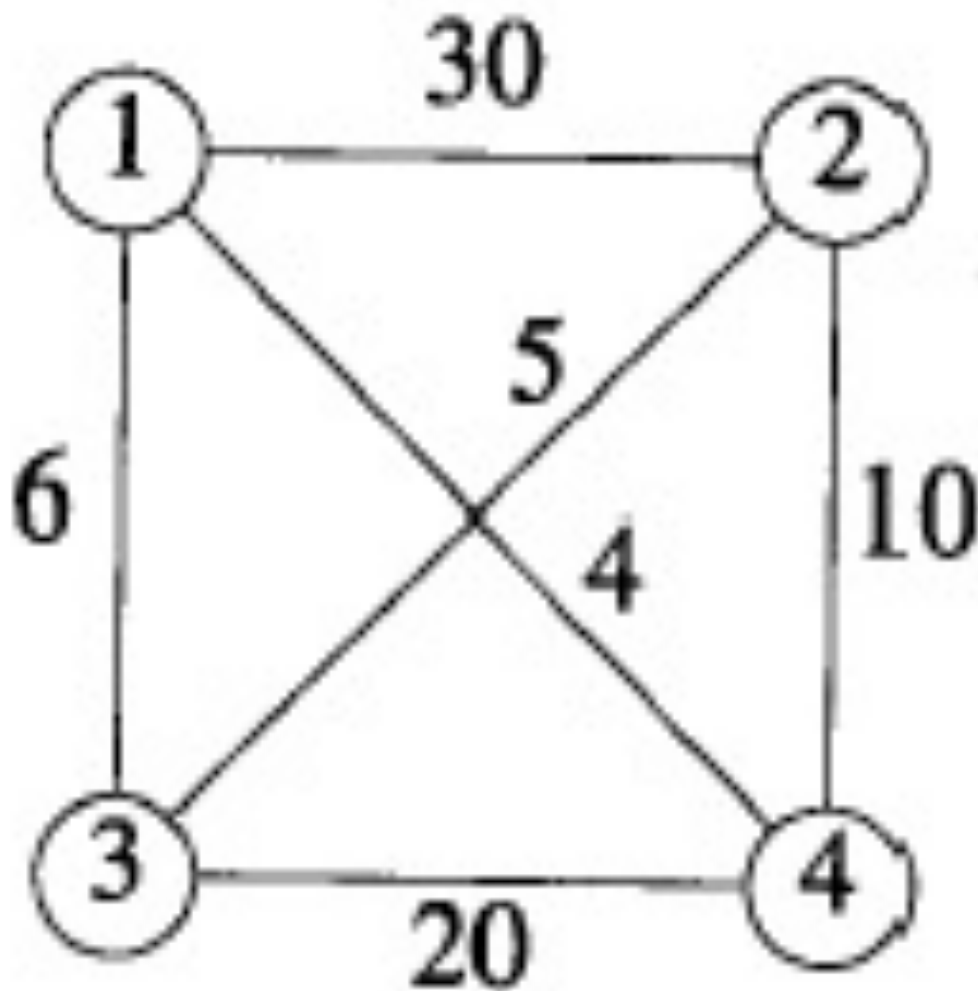
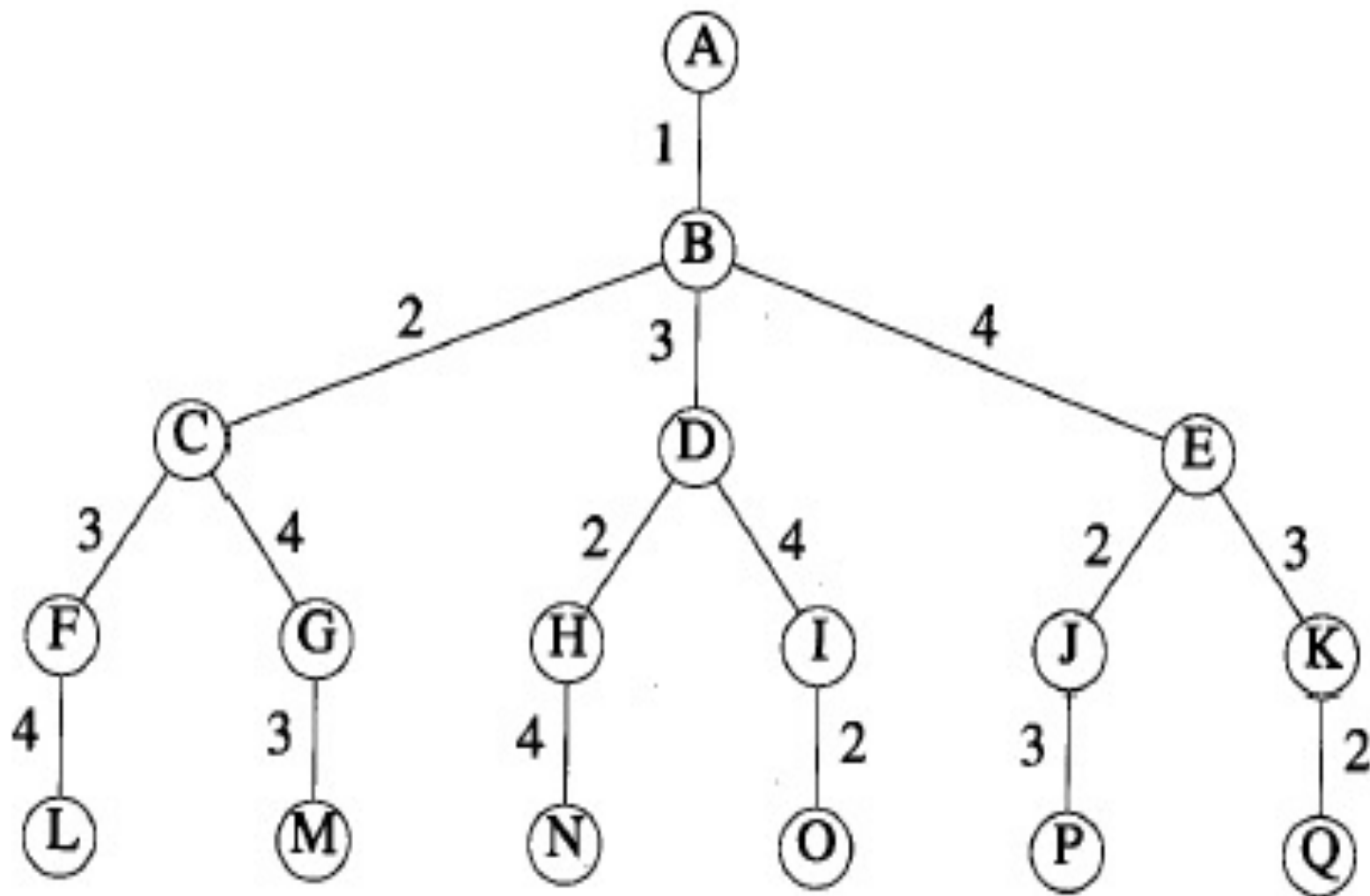


图16-5 状态空间树





续

- 边上的编号代表一个城市,
- 从城市1出发,经过城市2,...n,最终回到城市1,
- 叶节点代表最后经过的城市.
- 状态空间树中每条路径上相邻的两条边的编号对应网络的一条边.
- 从根到叶节点的一条路径上的编号为(2,...n)的一个置换(permutation).
- 置换树



16.2 应用

- 货箱装船
- 0/1背包问题
- 最大完备子图
- 旅行商问题

16.2.1 货箱装船

- 装船问题：给定载重量 c 的货船, 找一种装船的方法, 使得装载的货箱数目最多.
- 现对该问题做一些改动: 有两艘船和 n 个货箱; 第一艘船的载重量是 c_1 , 第二艘船的载重量是 c_2 , $w(i)$ 是货箱 i 的重量且

$$\sum w(i) \leq c_1 + c_2$$

- 是否有一种可将所有 n 个货箱全部装走的方法(可行解)? 若有的话找出该方法.

例16-4

- 当 $n=3, c_1=c_2=50, w=[10,40,40]$ 时,可将货箱1, 2装到第一艘船上,货箱3装到第二艘船上.如果 $w=[20,40,40]$,则无法将货箱全部装走.
- 当 $c_1=c_2$ 且货箱总重量为 $2c_1$ 时该问题等价于分划问题:
- 能否将一组非负整数分成两组使得每组内的数之和相等.这是NP难度问题.
- 如果两船装船问题存在可行解则按下述方法一定可找到(习题4): 最大化第一艘船的装船量,剩下的货箱如能装入第二艘船,则找到一个可行解.
- 这是因为: 设可行解在第一艘船装入 W_1 ($\leq c_1$) 重量,在第二艘船装入 W_2 ($\leq c_2$) 重量; 设最大化第一艘船的装船量为 M_1 ,则 $W_1 \leq M_1$; 而剩下的货箱重量之和 = 总重量 - $M_1 \leq$ 总重量 - $W_1 = W_2$,剩下的箱子一定能装入第二艘船.



回溯解

- 装船问题的解：
 - 求解优化问题:极大化第一只船的装箱重量.
 - 如剩余货箱能装入第二只船则找到了一个可行解,否则无解.

- 等价的优化问题:

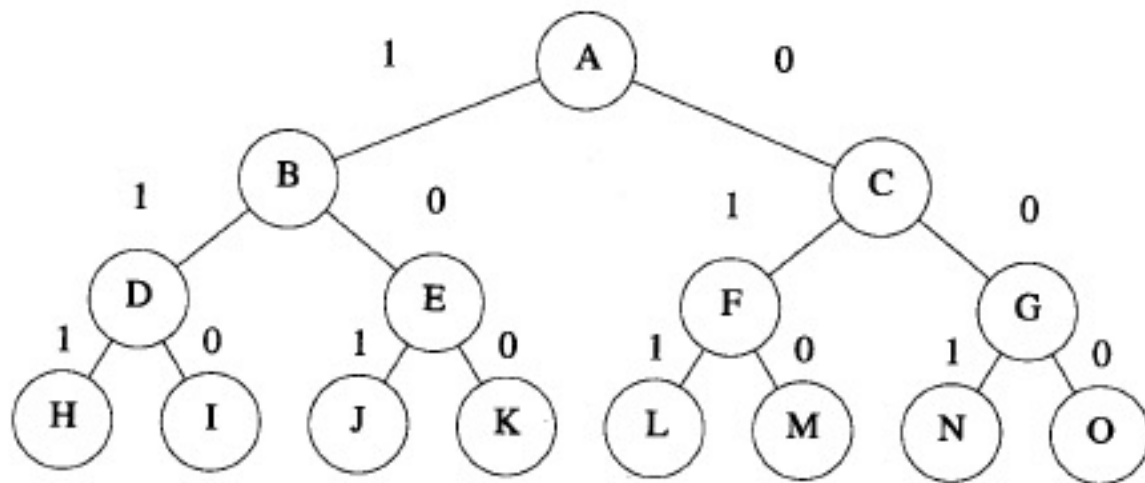
$$\text{Maximize } \sum w(i)x(i)$$

$$\text{Subject to } \sum w(i)x(i) \leq c_1 \text{ and } x(i) \in \{0,1\}$$

- 一类背包问题: $p(i)=w(i)$
- 也是子集和数问题的优化形式
- 类似背包问题可用动态规划求解($p_i=w_i$).
- 下面用回溯法求解:

例题16-5

- 假定 $n=4$, $w=[8, 6, 2, 3]$, $c_1=12$. 状态空间树



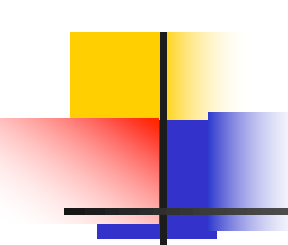
装船问题的状态空间树, 未显示叶节点


- 
- 限界方法1：设 cw 为当前已装的重量，

$$cw = \sum_{0 < j < i} w(j) \times (j)$$

如 $cw + w(i) > c_1$ 则杀死该(左)子节点.

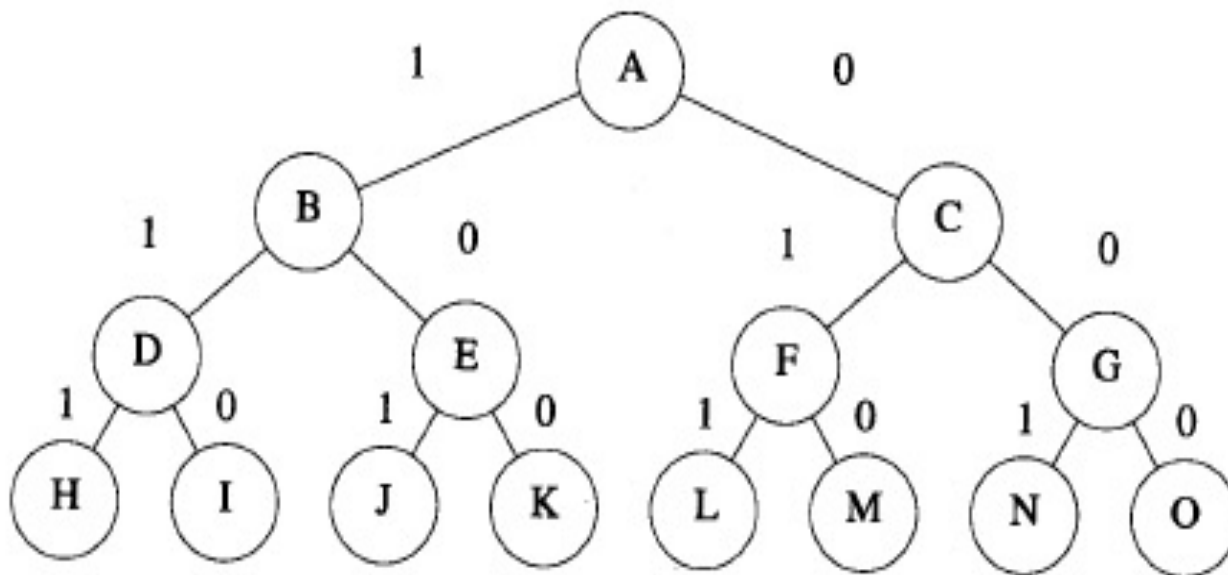
- 限界方法2：设 $bestw$ 为当前最优装箱重量, r 为未装货箱的总重量, 如 $cw + r \leq bestw$, 则停止展开该节点.
- $cw + r$ 为该节点对应的子问题的优化值的上界. 如果能找到更好的上界, 限界效果会更好.
- 两种限界同时使用.
- $Bestw$ 初始值为 $-\infty$.

- 
- 设 $x=(x(1), \cdots, x(k-1))$, 为当前E节点($cw+r > bestw$ 成立);
 - 展开左子节点:
 - 如果 $cw+w(k) > c1$, 则停止展开该左子节点, $r \leftarrow r-w(k)$, 并展开右子节点;
 - 否则, $x(k) \leftarrow 1$, $cw \leftarrow cw+w(k)$, $r \leftarrow r-w(k)$ 并继续展开 $(x(1), \cdots, x(k))$ (令其为E节点);
 - 展开右子节点:
 - 如果 $cw+r \leq bestw$ 则停止展开该右子节点, 并回溯到最近的一个活节点;
 - 否则, 令 $x(k)=0$, $(x(1), \cdots, x(k))$ 为E节点.
 - 回溯: 从 $i=k-1$ 开始, 找 $x(i)=1$ 的第一个 i ; 修改 cw 和 r 的值: $cw \leftarrow cw-w(i)$, $r \leftarrow r-w(i)$.

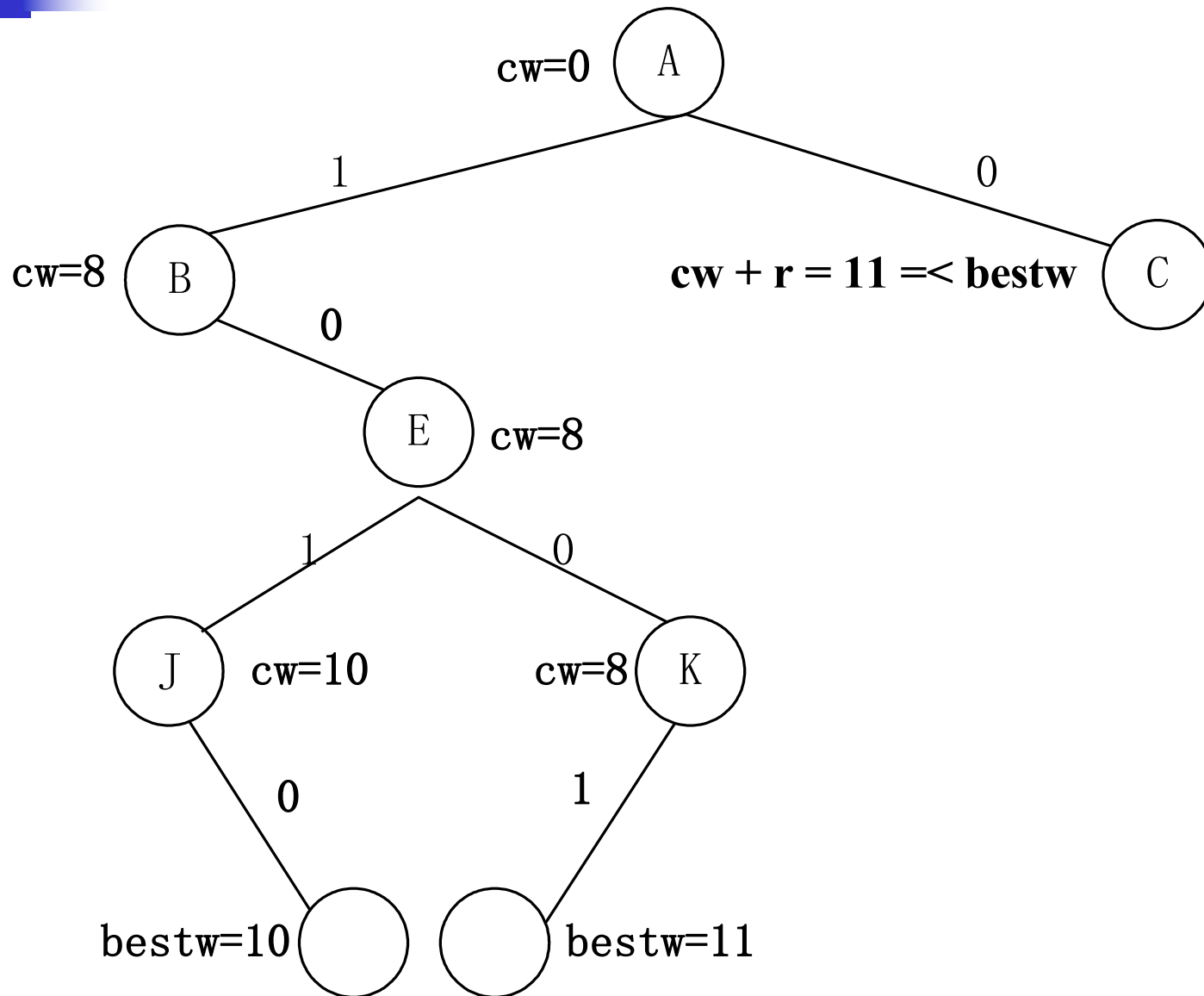
- 
- 如果 $k=n$ 且 $cw+w(n)>c1$,则:
 - 如果 $cw>bestw$, $bestw \leftarrow cw, x(n) \leftarrow 0$;
 - 否则, 回溯.
 - 否则($cw+w(n) \leq c1$),
 - $cw \leftarrow cw+w(n)$, 如果 $cw>bestw$, $bestw \leftarrow cw$, $x(n) \leftarrow 1$;
 - 否则, 回溯.
 - 当回溯过程到达根节点时算法结束.
 - 因为左子节点和父节点有相同的 $cw+r$ 值,所以算法在展开左子节点时不检查限界条件2.

例16-5

- $n=4, w=[8, 6, 2, 3], c_1=12$.



例题16.5展开的部分状态空间树





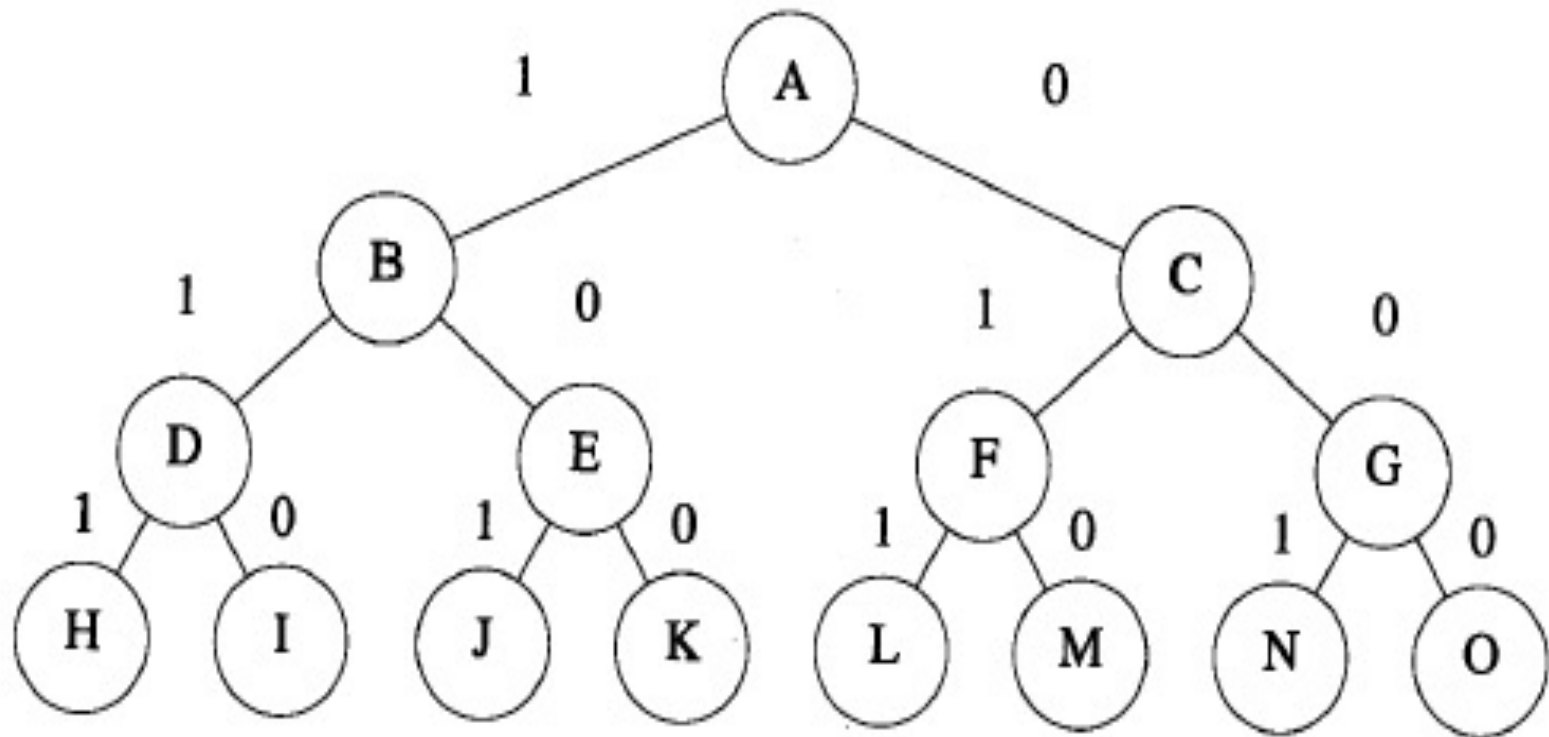
给出最优解

- 前面程序只给出了最优解的值; 程序16.3使用了一个数组bestx[]记录最优解.
- bestx[]存放达到最优解的路径
- 数组x[]存放回溯法当前使用的路径n
- 回溯时如果 $X[i]=1$ 说明还可以展开右子节点; 如果 $X[i]=0$ 说明所有子节点都展开了, 该节点成为死节点, 应向上一级节点回溯.

16.2.2 0/1背包问题

- 0/1背包问题是一个NP-难度问题, 本节用回溯法求解该问题.
- 使用定长元组表示, 其状态空间树同于装船问题.
- 设 $x=(x(1), \dots, x(k))$, 为状态空间树的节点, $bestp$ 是算法当前获得最优效益值,
 $cp=x(1)p(1)+\dots+x(k)p(k)$.
- 令 $bound(x)=cp$ +其余物品的连续背包问题的优化效益值(贪心法得到). $bound(x)$ 为节点 x 对应的子问题的优化效益值的上界:
- 如 $bound \leq bestp$ 则停止展开 x .

状态空间树



背包问题回溯法

- 按密度降序对物品排序
- $bestp \leftarrow -\infty$
- 设 $x=(x(1), \dots, x(k-1))$, 为当前E节点 ($bound > bestp$ 成立);
- 展开左子节点:
 - 如果 $cw + w(k) \leq c$, 则装入物品 k , 且
 $cw \leftarrow cw + w(k)$, $cp \leftarrow cp + p(k)$,
 $x(k) \leftarrow 1$
 - 否则, 展开右子节点.



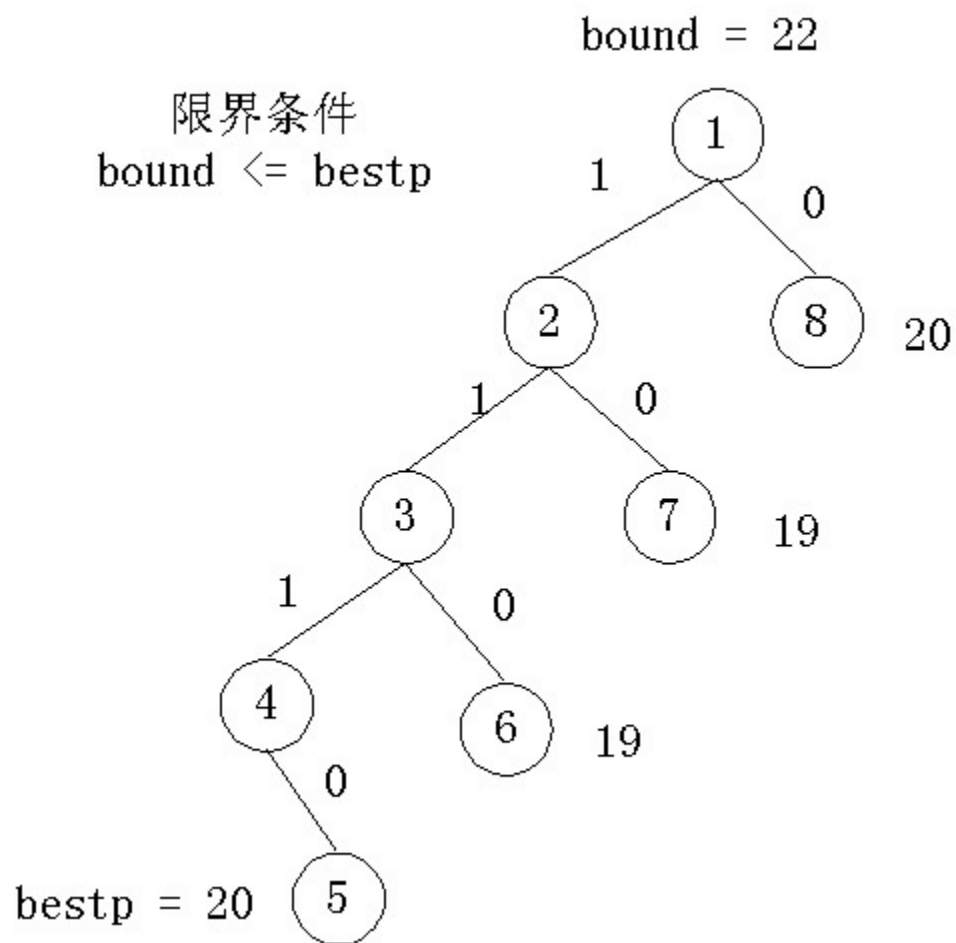
背包问题的回溯法

- 展开右子节点：
 - 如果 $\text{bound} \leq \text{bestp}$ 则停止产生右子树,
 - 否则, $x(k) \leftarrow 0$, 并令 $(x(1), \dots, x(k))$ 为E节点;
- 回溯的细节类似装船问题

例题16.7

- 考察一个背包例子: $n=4, c=7, p=[9, 10, 7, 4], w=[3, 5, 2, 1]$. 这些对象的收益密度为 $[3, 2, 3.5, 4]$.
- 按密度排列为: $(4, 3, 1, 2)$
- $w'=[1, 2, 3, 5], p'=[4, 7, 9, 10]$

展开的部分状态空间树





续

- 在结点5处得到 $best\ p=20$; 结点6处 $bound=19$, 故限界掉; 类似, 结点7, 8也被限界掉
- 其解为 $x=[1, 1, 1, 0]$, 回到排序前为 $x=[1, 0, 1, 1]$



16.2.3 最大集团(clique)

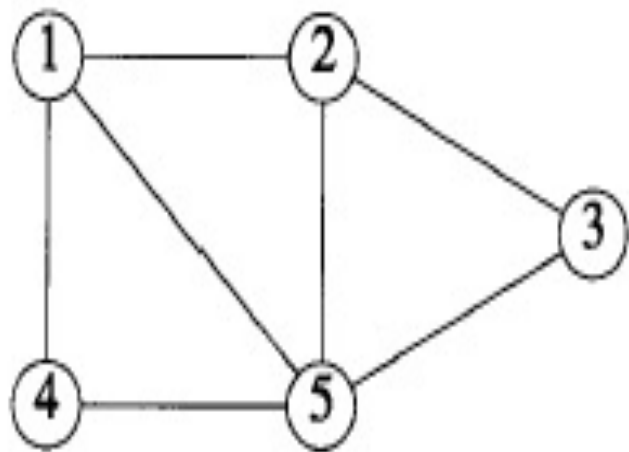
- 令 U 为无向图 G 的顶点的子集, 当且仅当对于 U 中的任意点 u 和 v , (u, v) 是图 G 的一条边时, U 定义了一个完全子图 (complete subgraph). 子图的尺寸为图中顶点的数量. 当且仅当一个完全子图不被包含在 G 的一个更大的完全子图中时, 它是图 G 的一个集团. 最大集团是具有最大尺寸的集团.



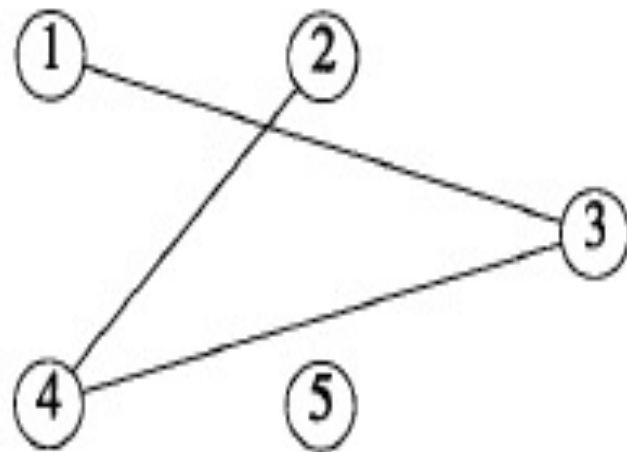
例 16-8

- 在图16-7a 中,子集 $\{1, 2\}$ 是尺寸为2的完全子图,但不是一个集团,它被包含在完全子图 $\{1, 2, 5\}$ 中.
- $\{1, 2, 5\}$ 为一最大集团.点集 $\{1, 4, 5\}$ 和 $\{2, 3, 5\}$ 也是最大集团.
- 独立(顶点)集:无边相连的顶点集合且包含意义下最大.
- 最大独立集: 顶点数最多的独立集.

图16-7 图及其(边)补图



a)



b)

a) 图G b) 补图 \bar{G}



例16-9

- $\{2, 4\}$ 为16-7a 的一个独立集. 同样还有 $\{1, 3\}$ 和 $\{3, 4\}$.
- 独立集和集团问题有对应关系:
- 图16-7b 是图16-7a 的(边)补图
- $\{2, 3\}$ 和 $\{1, 2, 5\}$ 是图16-7b 中图的独立集.
 $\{1, 2, 5\}$ 是最大独立集.
- G 的集团对应其补图的独立集, 反之亦然.
- 两个问题都是NP-难度问题

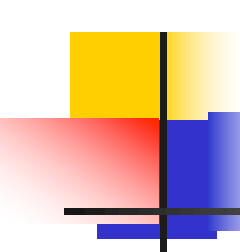
集团问题的状态空间树

- 最大集团问题的状态空间树：元组长度等于图的顶点数,分量取值 $\{0,1\}$ ： $x_i=1$ 表示顶点 i 在所考虑的集团中. 状态空间树类似0/1背包问题的状态空间树.
- 检查根节点到状态空间树的某一状态节点的路径上对应的图的顶点子集是否构成一个完全子图？如果不是则不再展开.
- 和装船问题相似,使用当前集团的顶点数 cn +剩余顶点数 $\leq bestn$ 进行限界.



程序16-10 最大完备子图


```
void AdjacencyGraph::maxClique(int i)
{
    // 计算最大完备子图的回溯代码
    if (i > n) { // 在叶子上
        // 找到一个更大的完备子图，更新
        for (int j = 1; j <= n; j++)
            bestx[j] = x[j];
        bestn = cn;
        return;
    }
}
```



```
// 在当前完备子图中检查顶点 i 是否与其它顶点相连
int OK = 1;
for (int j = 1; j < i; j++)
    if (x[j] && a[i][j] == NoEdge) {
        // i 不与 j 相连
        OK = 0;
        break;}

if (OK) { // 尝试 x[i] = 1
    x[i] = 1; // 把 i 加入完备子图
    cn++;
    maxClique(i+1);
    x[i] = 0;
    cn--;}

if (cn + n - i > bestn) { // 尝试 x[i] = 0
    x[i] = 0;
    maxClique(i+1);}
}
```

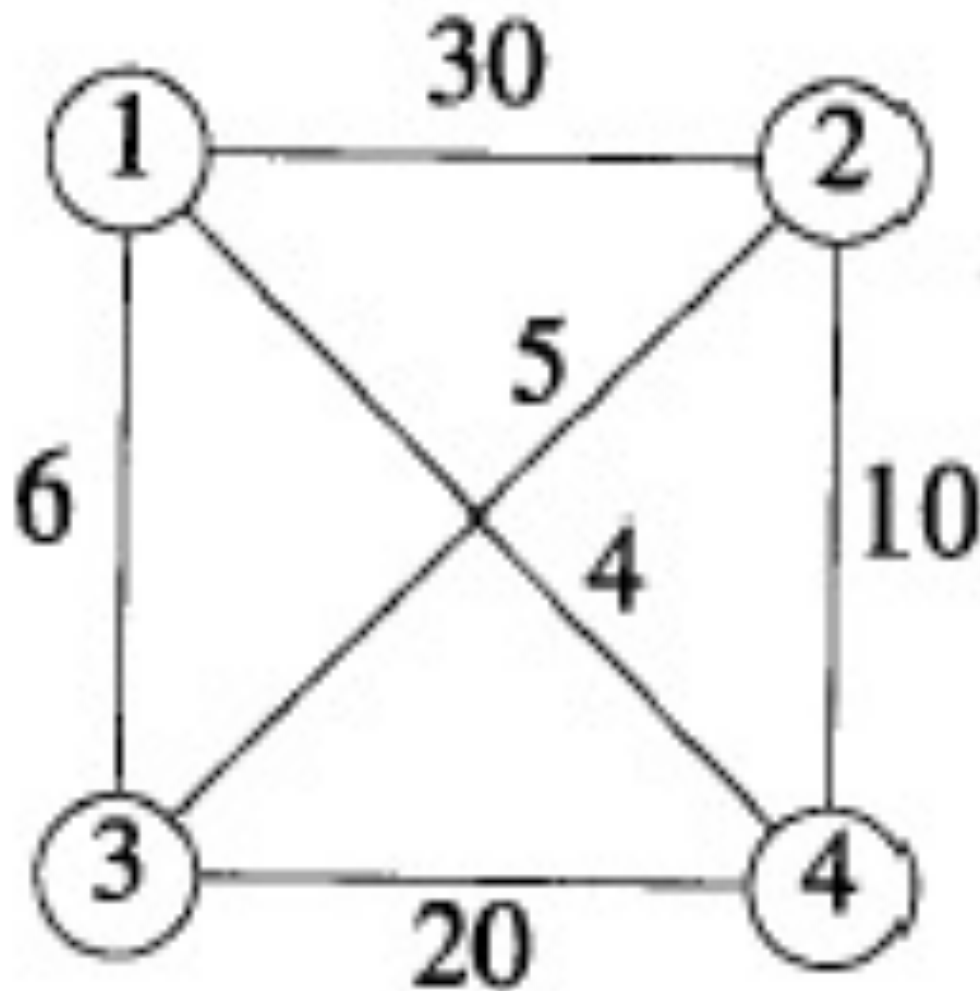



```
int AdjacencyGraph::MaxClique(int v[])
{
    // 返回最大完备子图的大小
    // 完备子图的顶点放入 v[1:n]
    // 初始化
    x = new int [n+1];
    cn = 0;
    bestn = 0;
    bestx = v;

    // 寻找最大完备子图
    maxClique(1);

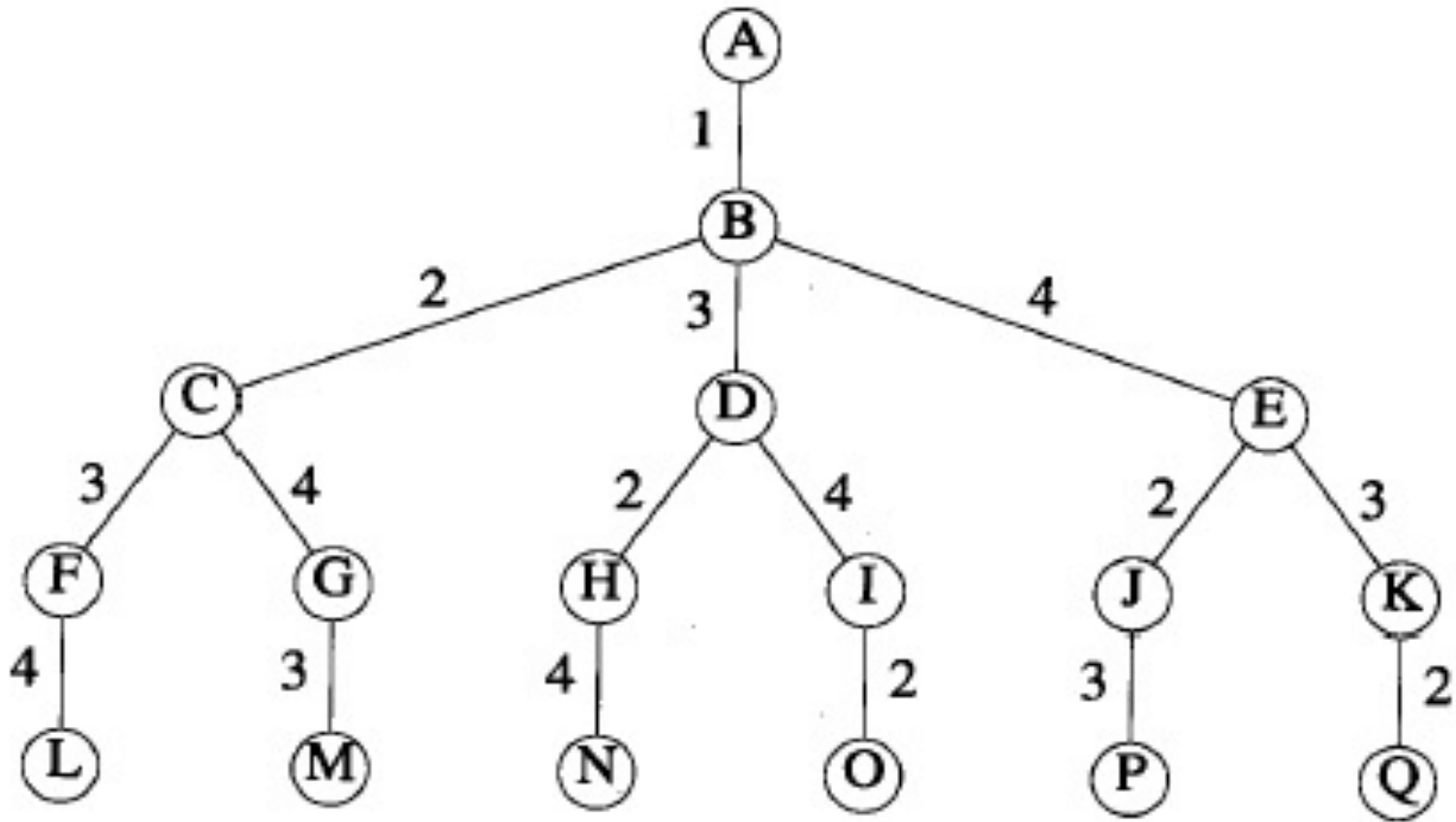
    delete [] x;
    return bestn;
}
```

图16-4 一个四顶点网络



16.2.4 旅行商问题

- 旅行商问题的状态空间树是一棵置换树。





限界条件

- 如果

- (1) $i-1$ 级节点 $x[i-1]$ 和它的子节点 $x[i]$ 之间有边相连($\neq \text{NoEdge}$)

- (2) 设 bestc 为当前得到的最优解的成本值；
路径 $x[1:i]$ 的长度 $< \text{bestc}$

则搜索继续向下进行；否则施行限界

- 程序16.12使用了上述限界条件.
- 例题16.3,该图的邻接矩阵为:

算法执行过程

- $\infty, 30, 6, 4$
30, $\infty, 5, 10$
6, 5, $\infty, 20$
4, 10, 20, ∞

x[1]	1	1	1	1	1
x[2]	2	2	3	3	4
x[3]	3	4	2	4	2
x[4]	4	3	4	2	
bestc	59	56	25		



续

- 在结点L处 $bestc=59$,结点M处 $bestc=56$
结点N处 $bestc=25$
- 结点I处对应的路径长度为 $26>25$,不再 产生I以下的子树
- 如果在产生子节点时使用启发式: 优先产生边长最小的子节点, 有望获得更好的限界效果.

程序16-11 旅行商回溯算法的预处理程序

```
template<class T>
T AdjacencyWDigraph<T>::TSP(int v[])
{
    // 用回溯算法解决旅行商问题
    // 返回最优旅游路径的耗费，最优路径存入 v[1:n]
    // 初始化
    x = new int [n+1];
    // x 是排列
    for (int i = 1; i <= n; i++)
        x[i] = i;
    bestc = NoEdge;
    bestx = v; // 使用数组 v来存储最优路径
    cc = 0;

    // 搜索x[2:n]的各种排列
    tSP(2);

    delete [] x;
    return bestc;
}
```

程序16-12 旅行商问题的迭代回溯算法

```
void AdjacencyWDigraph<T>::tSP(int i)
{
    // 旅行商问题的回溯算法
    if (i == n) { // 位于一个叶子的父节点
        // 通过增加两条边来完成旅行
        if (a[x[n-1]][x[n]] != NoEdge &&
            a[x[n]][1] != NoEdge &&
            (cc + a[x[n-1]][x[n]] + a[x[n]][1] < bestc ||
             bestc == NoEdge)) { // 找到更优的旅行路径
            for (int j = 1; j <= n; j++)
                bestx[j] = x[j];
            bestc = cc + a[x[n-1]][x[n]] + a[x[n]][1];
        }
    }
}
```


程序16-12 旅行商问题的迭代回溯算法(续)

```
else { // 尝试子树
    for (int j = i; j <= n; j++)
        // 能移动到子树 x[j] 吗?
        if (a[x[i-1]][x[j]] != NoEdge &&
            (cc + a[x[i-1]][x[i]] < bestc ||
             bestc == NoEdge)) { // 能
            // 搜索该子树
            Swap(x[i], x[j]);
            cc += a[x[i-1]][x[i]];
            tSP(i+1);
            cc -= a[x[i-1]][x[i]];
            Swap(x[i], x[j]);
        }
    }
}
```