有限离散问题总可以用穷举法求得问题的全部.

例 题 0-1背包问题(0-1Knapsack Problem)

设有n个物体和一个背包,物体i的重量为w_i价值为p_i背包的载荷为M,若将物体i($1 \le i \le n$,)装入背包,则有价值为p_i.

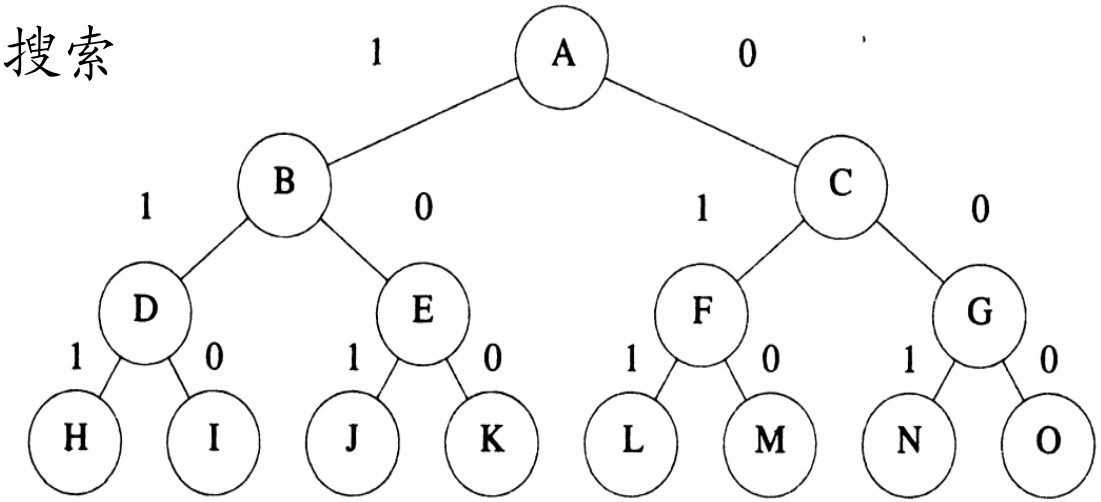
目标是找到一个方案,使得能放入背包的物体总价值最高

例如 取N=3 , 问题所有可能的解为(解空间):

(0, 0, 0), (0, 0, 1), (0, 1, 0), (0, 1, 1), (1, 0, 0), (1, 0, 1), (1, 1, 0), (1, 1, 1)

可表示为一棵3层的完全正则二叉树

求解过程相当于在树中搜索满足条件的叶结点.



若取W= (20,15, 15),

P= (40,25, 25),

C=30

时间复杂性: $O(2^n)$

第五章. 回溯法 (Back traiking)

5.1 基本思想

设问题的解可表示为n元组$(x_1, x_2,\dots x_n)$, $x_i \in s_i$, s_i为有限集, n元组的子组$(x_1, x_2,\dots x_i)$ $i<n$ 应满足一定的约束条件D. 设已有满足约束条件的部分解$(x_1, x_2,\dots x_i)$, 添加$x_{i+1} \in s_{i+1}$, 若$(x_1, x_2,\dots x_i, x_{i+1})$满足约束条件, 则继续添加$x_{i+2}$; 若所有可能的$x_{i+1} \in s_{i+1}$均不满足约束条件,则去掉$x_i$, 回溯到$(x_1, x_2,\dots x_{i-1})$, 添加尚未考虑过的$x_i$, 如此反复进行,直到$(x_1, x_2,\dots x_k)$ $k\le n$满足所有的约束条件或证明无解.

E={ $(x_1, x_2,\dots x_n), x_i \in s_i, s_i$为有限集 **}**称为问题的**解空间.**

约束条件
- 显约束:每个xi 的范围都给定的约束.满足显约束的全体向量构成解空间.
- 隐约束:元组的分量间满足函数关系f(x1,...xn)

求解过程可表示为在一棵解空间树 作 深度优先搜索.

解空间树构造:

1.子集树:当解向量为不定长n元组时 ， 树中从根至每一结点的路径集合构成解空间.树的每个结点称为一个**解状态**,有儿子的结点称为**可扩展结点**,叶结点称为**终止结点**, 若结点v对应解状态$(x_1, x_2,\dots x_i)$,则其儿子对应扩展的解状态$(x_1, x_2,\dots x_i, x_{i+1})$.满足所有约束条件的解状态结点称为**回答结点**.

2.排序树:当解向量为定长n元组时, 树中从根至叶结点的路径的集合构成解空间.树的每个叶结点称为一个解状态.

搜索过程:

例 题

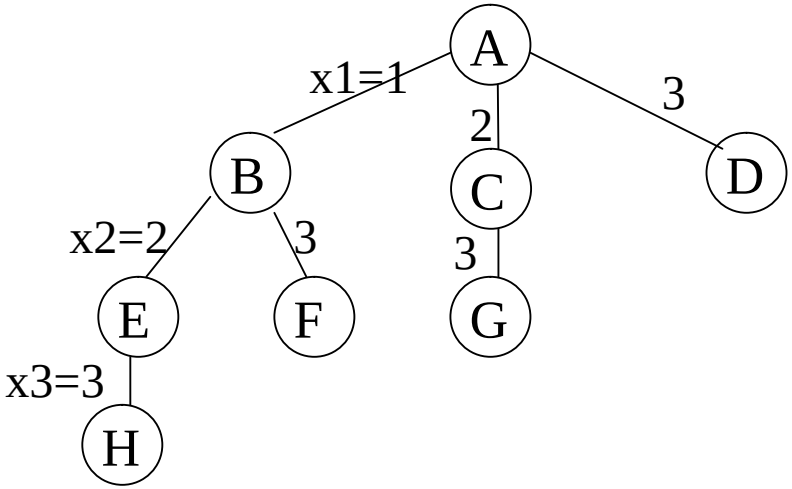
搜索按深度优先策略从根开始, 当搜索到任一结点时,判断该点是否满足约束条件D(剪枝函数),满足则继续向下深度优先搜索,否则跳过该结点以下的子树(剪枝),向上逐级回溯.

例 题 0-1背包问题

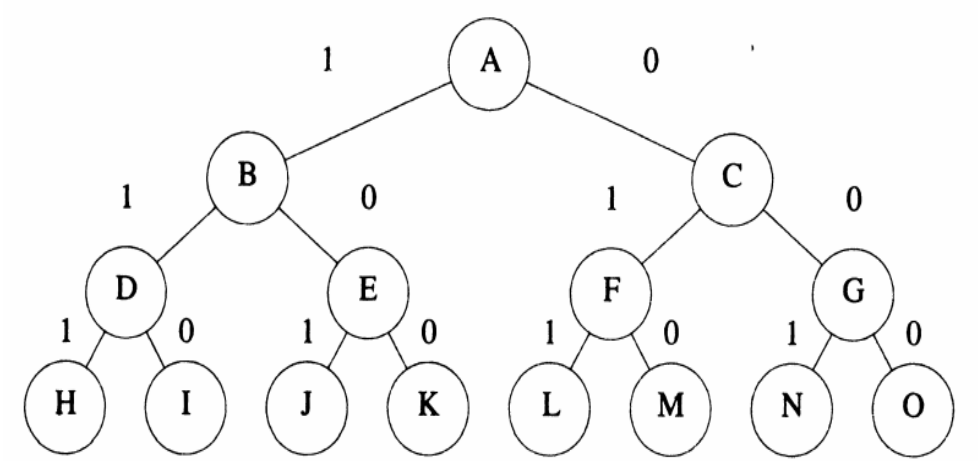
设有n个物体和一个背包,物体i的重量为w_i价值为p_i背包的载荷为M,若将物体i(1≤ i ≤n,)装入背包,则有价值为p_i .

目标是找到一个方案,使得能放入背包的物体总价值最高

设N=3, W=(20,15,15), P=(40, 25, 25), C=30



子集树



排序树

回溯法解题步骤:

1).针对所给问题,定义问题的解空间

2).确定解空间结构.

3).以深度优先方式搜索解空间.

算法模式

```
Procedure BACKTRACK(n);
{k:=l;
  repeat
    if  T_K (x1,x2,...x_K-1 )中的值未取遍 then
      { x_K:=T_K (x1, x2, ..., x_K-1 )中未取过的一个值;
        if B_K (x1, x2, ..., x_K) then //状态结点(x1,...xk)被激活
          if k=n then output(x1, x2, ..., xk) //输出一个回答结点
        e1se k:=k + l; } //深度优先
    e1se k:=k-l;  //回溯
  until k=0;
end; {BACKTRACK}
```

递归回溯

```
void Backtrack(int t)
{ if (t > n) Output(x);
  else
    for (int i = f(n,t);i <=g(n,t) ;i ++)
       x[t] = h(i);
       if (Constraint(t)&&Bound(t) ) Backtrack(t + 1)
    }}
```

迭代回溯

```
void IterativeBacktrack(void)
{int t = 1;
  while (t > 0) {
    if (f(n,t) < = g(n,t))
      for (int i = f(n,t);i < = g(n,t);i ++)
        x[t] = h(i);
        if (Constraint(t) &&Bound(t) ) {
          if (Solution(t)) Output(x);
          else t ++;}
      else t --;
```

5.2 子集和问题

问题陈述:给定n个不同正实数的集合W={w(i)|1≤ i ≤n }和一个正整数M,要求找到子集S, 使得$\sum_{i\in s} w(i)$=M

例如　设n=4, W=(11,13, 24, 7), M=31

满足约束条件的子集为(11,13,7), 和 (24, 7),

S=(1, 2, 4), 和 S=(3, 4),

算法思路1: 问题的解可表示为k元向量{x1, x2, ... xk }, 1≤ k≤n, xi∈W

解空间树为子集树.

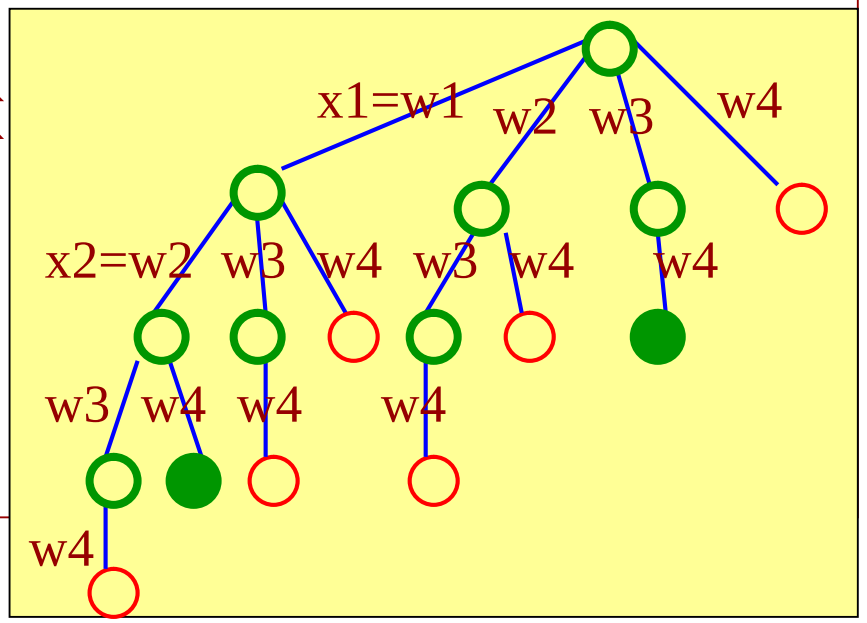
约束条件

1)当i ≠ j , xi ≠ xj (元素不能重

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

3) $\sum_{i=1}^{j} x_i + x_{j+1} \le M$

4) $\sum_{i=1}^{j} x_i + \sum_{i=j+1}^{n} w(i) \ge M$

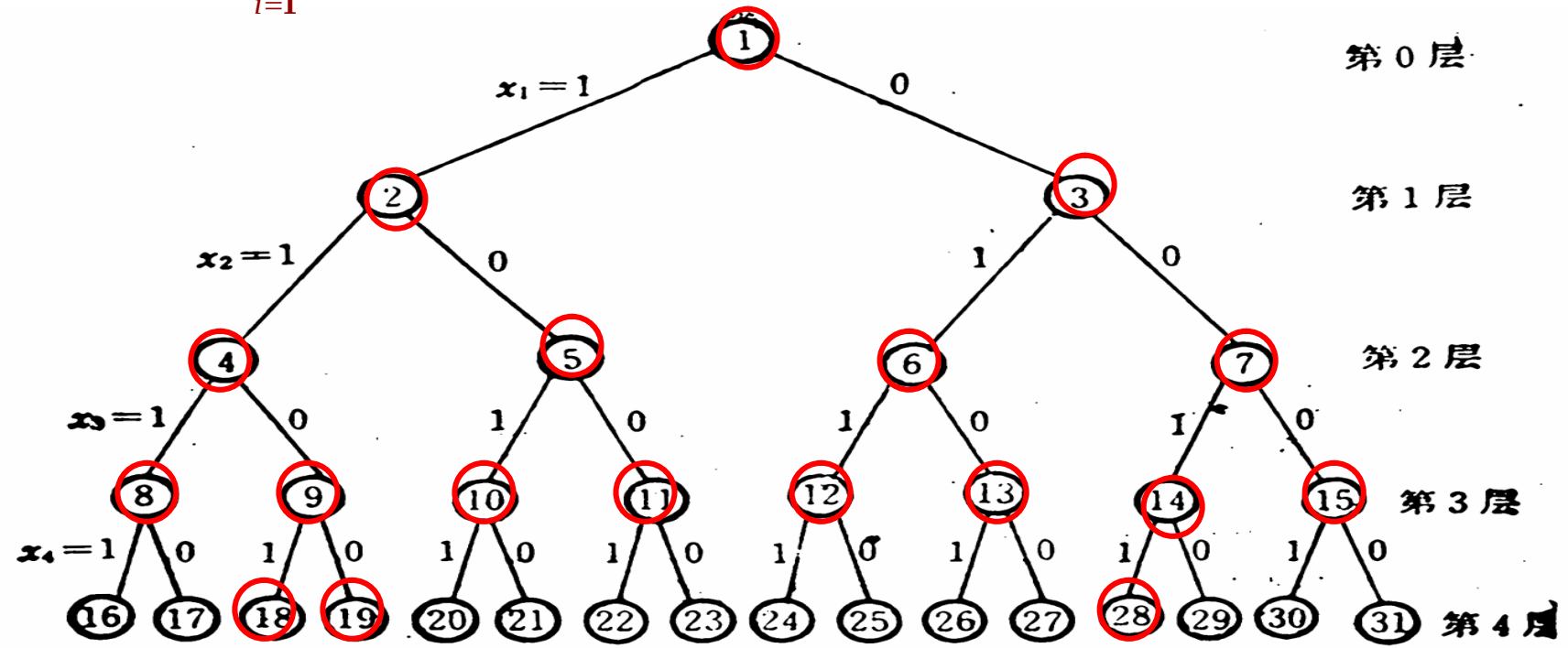


算法思路2: 问题的解可表示为n元向量{x1, x2, ... xn }, xi=1或0

则问题的解空间树为排序树

约束条件: $\sum_{i=1}^{K} w(i)x_i + W(K+1) \le M$

$\sum_{i=1}^{j} w(i)x_i + \sum_{i=j+1}^{n} w(i) \ge M$

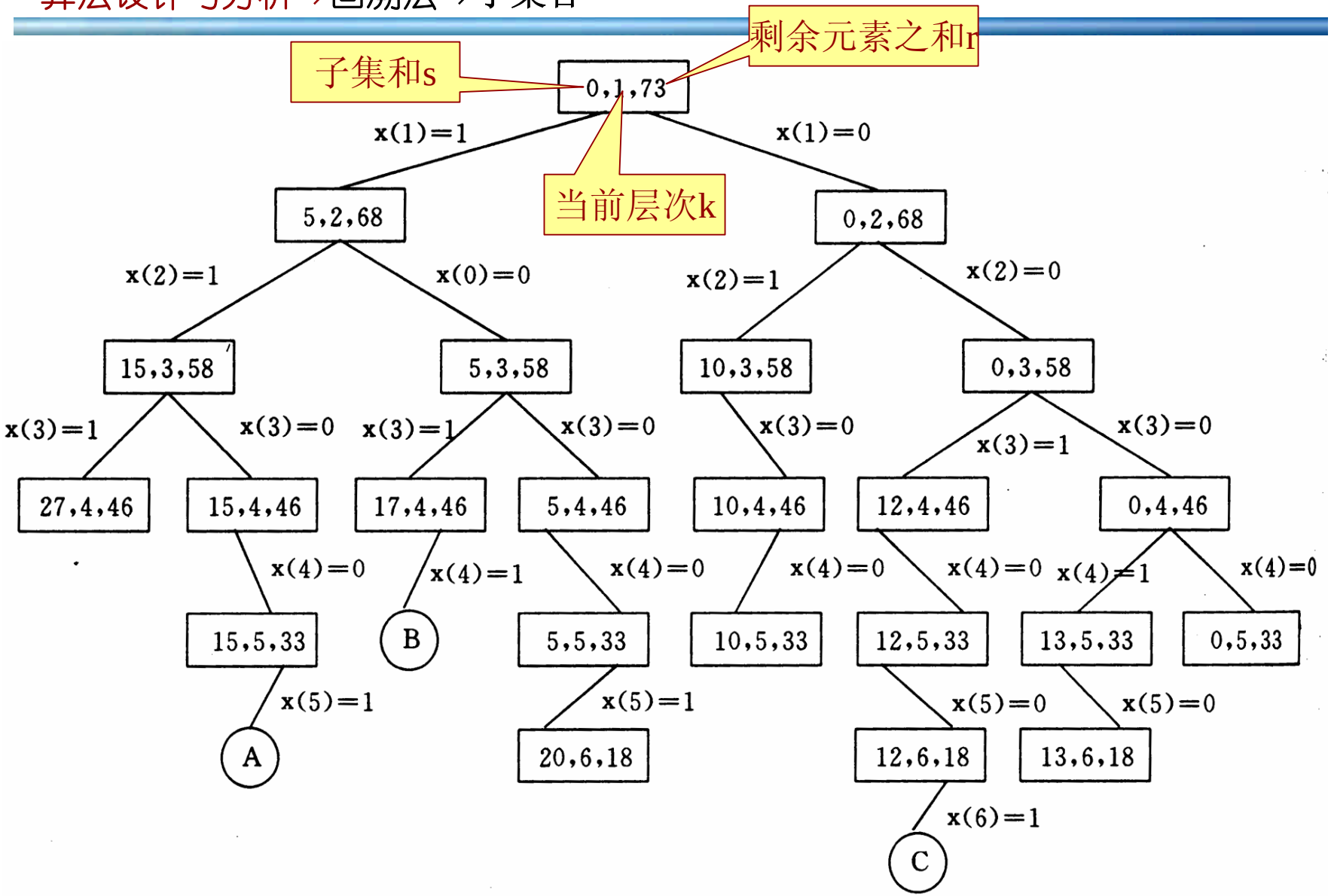


n=4, W=(11,13, 24, 7), M=31

子集合问题回溯算法(排列树)

```
procedure sumofsub(s，k，r);
 { 正实数组W[l..n]，布尔数组X[l..n).M，n：均为全局量。
   W[l..n]按非降次序排列，s= Σ(j=1..k-1) W[j]X[j]，r= Σ(j=k..n) w[j]
   假定W[l]≤M及 Σ(k=1..n) W[i]≥M }
 { X[k]:=1;
   lf  s+W[k]=M then
       Print the result X[1..k]
   else if s+W[k]+W[k+1]<＝M  then
       sumofsub(s+W[k]，k+1，r-W[k])
   if s+r-W[k]>＝M and S+W[k+1]≤＝M then
      X[k]:=0; sumofsub(s, k+1，r-W[k] )；  }
```

算法复杂性:约为穷举法的1/3



n=6, m=30, w={5, 10, 12, 13, 15, 18}时由算法生成的部分解空间树

5.3 装载问题

问题描述:n个集装箱装到2艘载重量分别为c1, c2的货轮, 其中集装箱i的重量为wi且 $\sum_{i=1}^{n} w_i \le c_1 + c_2$
问题要求找到一个合理的装载方案可将这n个货箱装上这2艘轮船。

例如 当n=3, c1=c2=50, w=[10, 40, 40], 可将货箱1和2装到第一艘船上; 货箱3装到第二艘船上; 若w=[20, 40, 40], 则无法将全部货箱装船.

- 当 $\sum_{i=1}^{n} w_i = c_1 + c_2$ 时问题等价于子集和问题; 当c1=c2且 $\sum_{i=1}^{n} w_i = 2c_1$ 时问题等价于划分问题.

若装载问题有解, 采用如下策略可得一个最优装载方案:
(1)将第一艘轮船尽可能装满; (2)将剩余的货箱装到第二艘轮船上。
将第一艘船尽可能装满等价于如下0-l背包问题:

$$\max \sum_{i=1}^{n} w_i x_i \quad \sum_{i=2}^{n} w_i x_i \le c_1 \quad x_i \in \{0,1\},\ 1 \le i \le n$$

可采用动态规划求解, 其时间复杂性为:O(C1,2^n)

算法思路:用排序树表示解空间,则解为n元向量{x1, ... ,xn }, $x_i \in \{0, 1\}$

约束条件: $\sum_{i=1}^{j} w_i x_i + w_{j+1} \leq c1$

由于是最优化问题, 可利用最优解性质进一步剪去不含最优解的子树:

设 bestw: 当前最优载重量,

$cw = \sum_{i=1}^{j} w_i x_i$; 当前扩展结点的载重量 ;

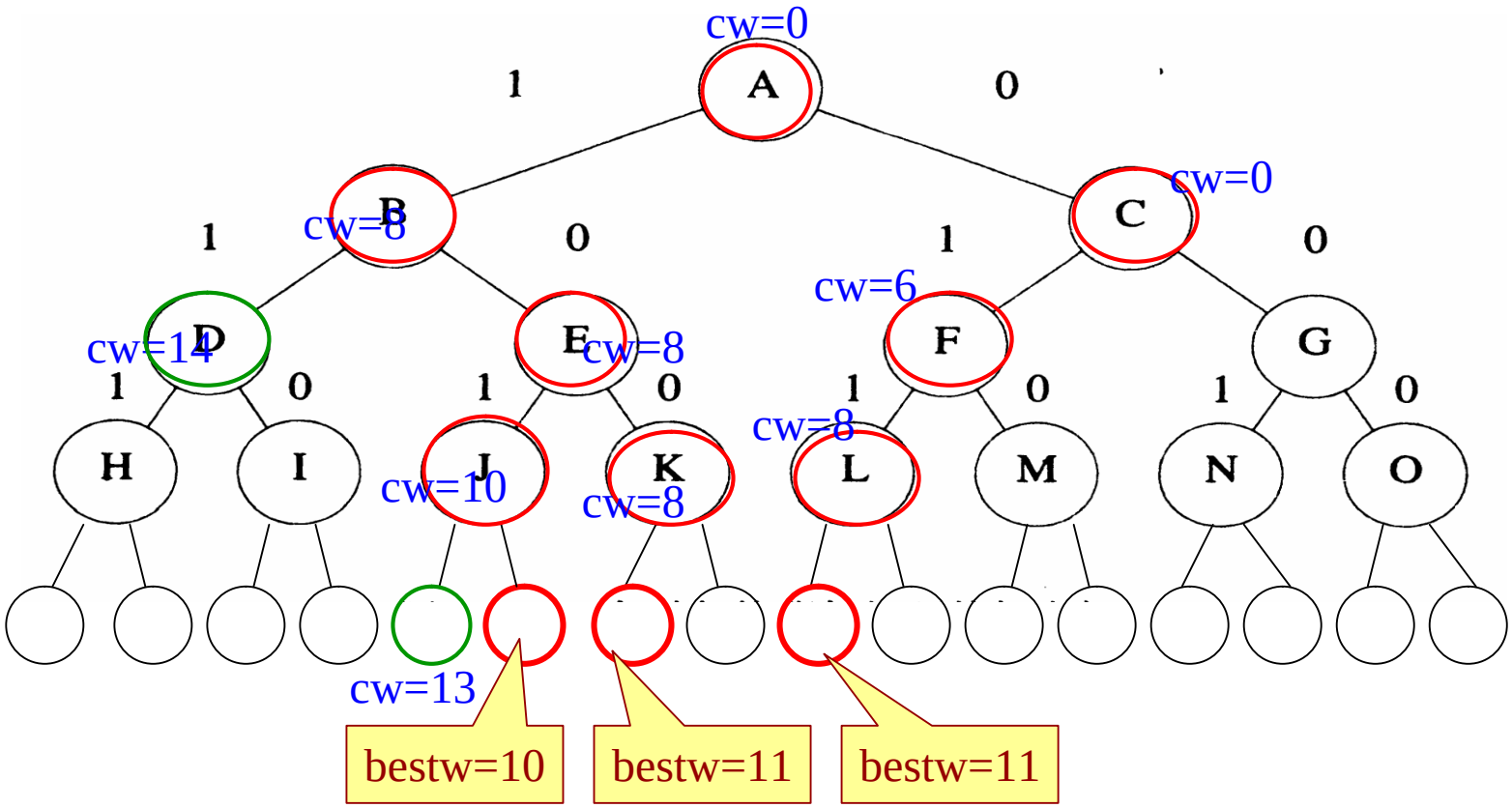
$r = \sum_{i=j+1}^{n} w_j$; 剩余集装箱的重量;

当 cw+r (限界函数)≤bestw 时, 将cw对应右子树剪去。

$$\sum_{i=1}^{j} w_i x_i + w_{j+1} > c1$$

$$cw+r \leqslant bestw$$

例如 n=4, c1=12, w=[8, 6, 2, 3].



装载问题的回溯算法

```
template < class Type >
Type Maxloading(type w[], type c,
                int n,)
    loading <Type> X;
    //初始化X
    X. w=w;  //集装箱重量数组
    X. c=c;   //第一艘船载重量
    X. n=n;   //集装箱数
    X. bestw=0; //当前最优载重
    X. cw=0; //当前载重量
    X. r=0;   //剩余集装箱重量
    for (int i=1; i<=n; i++)
        X. r +=w[i]
    //计算最优载重量
    X.Backtrack(1);
    return X.bestw;    }
```

```
template<classType>
void Loading<Type>::Backtrack(int i)
{ //搜索第i层结点
    if (i>n) {//到达叶结点
        bestw=cw;
        return;  }
    //搜索子树
    r - = w[i];
    if (cw+w[i]]<=c){ //x[i]=1
        cw += w[i];
        Backtrack (i+1);
        cw - = w[i]; }
     if (cw+r > bestw){ //x[i]=0
        Backtrack(i+1);
    r+=w[i]
    }
```

算法复杂性： $O(2^n)$

5．3 批处理作业调度

例 题

问题描述:给定n个作业的集合J=(J1, J2, ... , Jn)。每一作业Ji都有两项任务要分别在2台机器上完成. 每一作业须先由机器l处理, 再由机器2处理. 设tji是作业Ji在机器j上的处理时间, i=1,...,n, j=1, 2.Fji是作业Ji在机器j上完成处理的时间. 所有作业在机器2上完成时间和: $f=\sum F_{2i}$ 称为该作业调度的**完成时间和**.

对于给定的J, 要求制定一个最佳作业调度方案, 使完成时间和最小.

算法思路:设解为n元向量{x1,... ,xn }, $x_i \in \{1,..n\}$, 用排序树表示解空间

约束条件: 当i ≠ j , xi ≠ xj (元素不能重复选取)

限界函数: bestx:为当前最小时间和

x : 当前扩展结点的时间和;

r : 剩余作业的时间和;

当 x+r <= bestx 时, 将x对应右子树剪去

例 题

作业调度回溯算法

```
int Flow(int ** M, int n, int bestx[ ] )
{ int ub = 32767;
   Flowshop X;
   X.x = new int [n+ 1]; //当前调度
   X.f2 = new int[n+ l];
   X.M = M; //各作业所需处理时间
   X.n= n; //作业数
   X.bestx = bestx; //当前最优调度
   X. bestf = ub; //当前最优调度时间
   X.fl = 0; //机器2完成处理时间
   X.f = 0; //机器1完成处理时间
   for(int i = 0;i<= n; i++)
      X.f2[i] = 0,X.x[i]=i;
   X. Backtrack( 1 );
   delete [ ] X. x;
   delete [ ] X. f2;
   return X. bestf ;}
```

```
void Flowshop: :Backtrack(int i)
{ if (i>n) {
      for(int j = 1;j <= n; j++)
          bestx[j] = x[j];
      bestf = f;}
   else
      for (int j = i; j < = n; j ++) {
          fl += M[ x[j] ] [1];
          f2[i]= ((f2[i-1]>fl)?f2[i-1]:fl)
                    +M[x[j]][2];
          f+= f2[i];
          if (f <bestf) {
              Swap(x[i], x[j]);
              Backtrack(i + 1 );
              Swap(xl i], x[j] );}
          fl - = Mix[j]][1];
          f- = f2[i] ;}}
```

算法复杂性： O(n!)

例 题

tji	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3

这三个作业的6种可能调度方案是:

1, 2, 3; 1, 3, 2; 2, 1, 3;

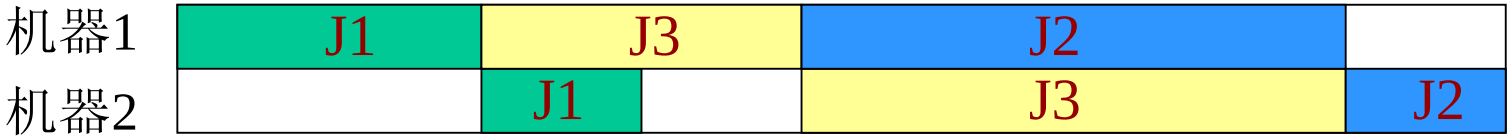
2, 3, 1; 3, l, 2; 3, 2, l;

相应的完成时间和分别是:

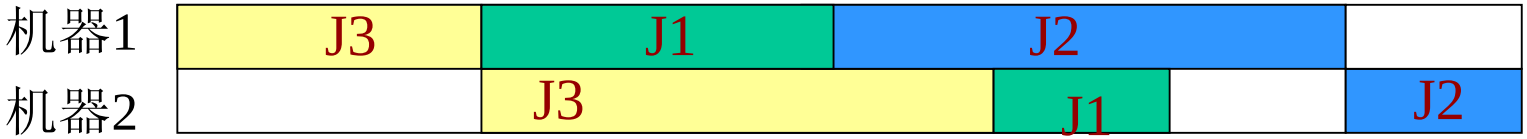
10, 8, 10，9，7，8。

最佳调度: 3, 1, 2; 完成时间为7。

调度1,3,2



调度3,1,2



例 题

tji	机器1	机器2
作业1	2	1
作业2	3	1
作业3	2	3

这三个作业的6种可能调度方案是:

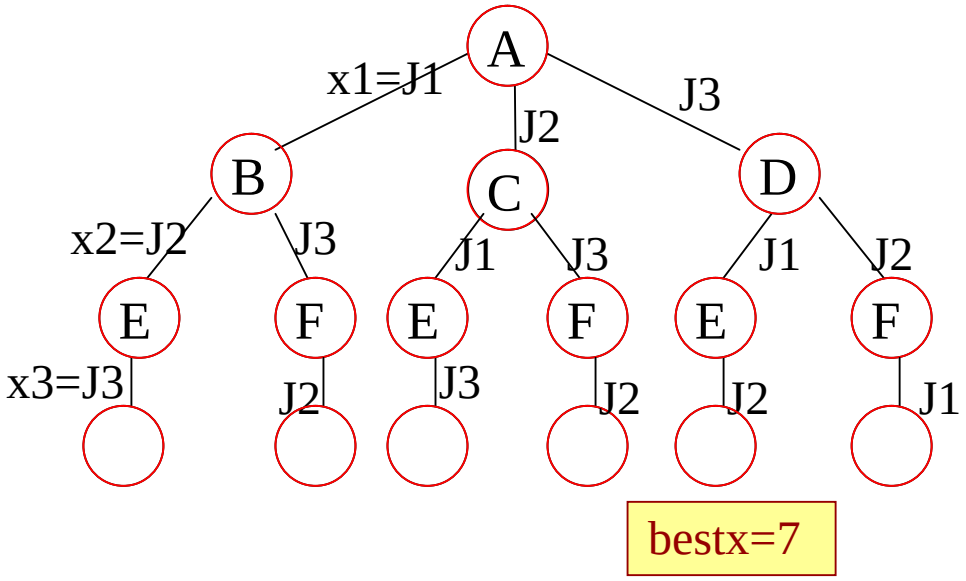
1, 2, 3; 1, 3, 2; 2, 1, 3;

2, 3, 1; 3, l, 2; 3, 2, l;

相应的完成时间和分别是:

10, 8, 10，9，7，8。

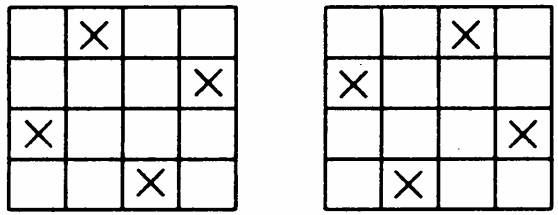
最佳调度: 3, 1, 2; 完成时间为7。



5.3 n后问题

问题描述:nXn棋盘上放置n个皇后使得每个皇后互不受攻击. 即任二皇后不能位于同行同列和同一斜线上.

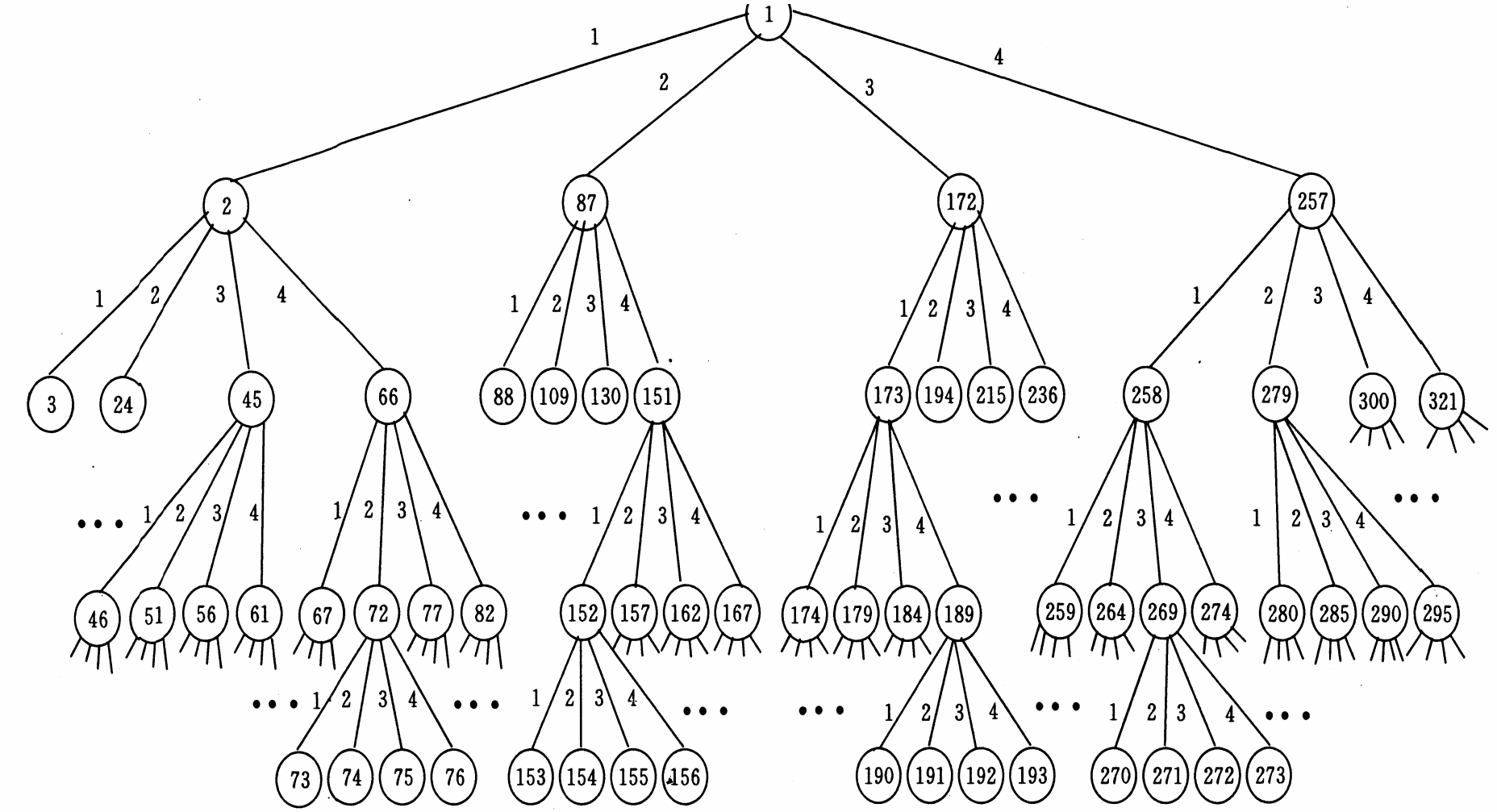
如四后问题的解



算法思路:将棋盘从左至右,从上到下编号为1,...,n,皇后编号为1,...,n.
设解为$(x1, ..., xn)$, xi为皇后i的列号,且xi位于第i行.
解空间:$E=\{ (x1,..., xn) \mid xi \in Si, i=1,...,4\}$, $Si=\{1, ..., 4\}, 1 \le i \le n$
解空间为排列树.
其约束集合D为

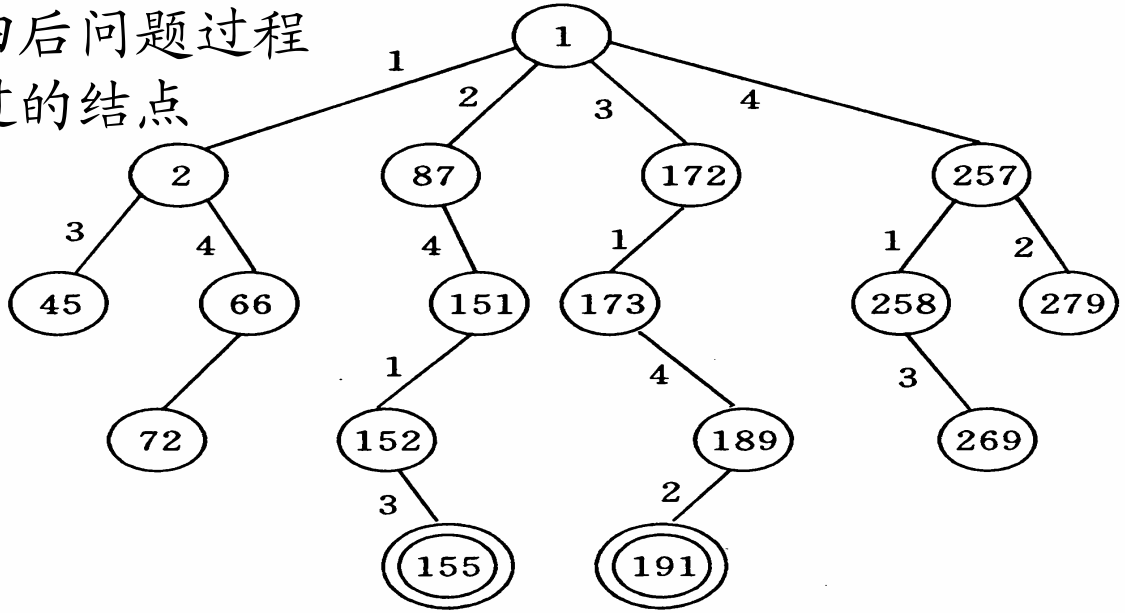
1) $x_i \neq x_j$ 皇后i, j不在同一列上
2) $x_i - i \neq x_j - j$
3) $x_i + i \neq x_j + j$

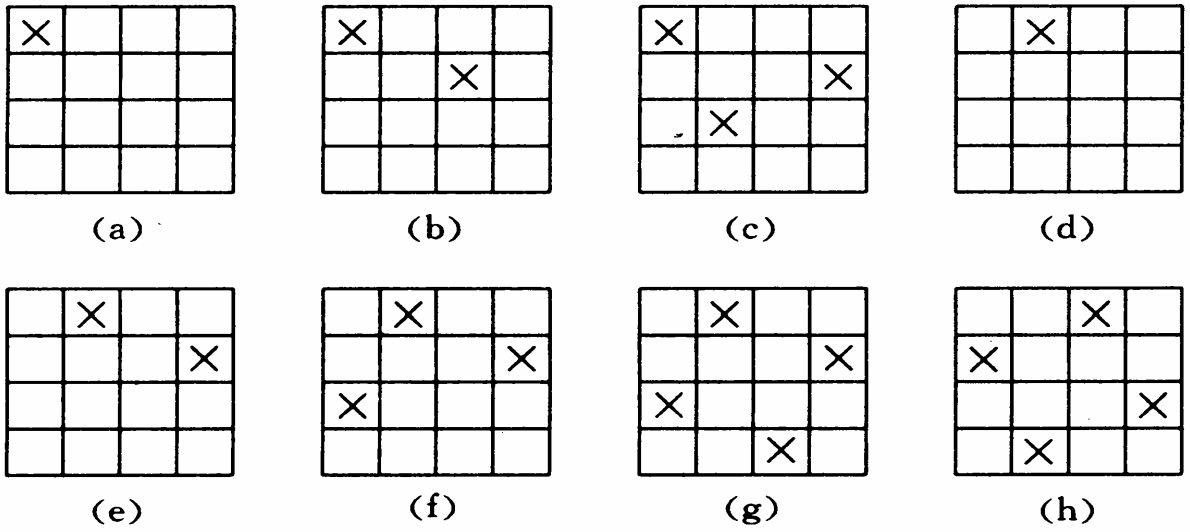
皇后i, j不在同一斜线上


1
2
87
172
257
3
24
45
66
88
109
130
151
173
194
215
236
258
279
300
321
46
51
56
61
67
72
77
82
152
157
162
167
174
179
184
189
259
264
269
274
280
285
290
295
73
74
75
76
153
154
155
156
190
191
192
193
270
271
272
273

四后问题的状态空间树

回溯求解四后问题过程中被激活过的结点





n后问题的回溯算法

```
bool Queen:: Place(int k)
  { for (int j = 1; j < k; j++)
      if ((abs(k-j) = = abs(x[j] - x[k]))
        ||(x[j] = = x[ k] ) ) return false;
   return true; }

void Queen:: Backtrack(int t)
{ if (t > n) sum++;
      else
        for (int i=1;i < = n;i++) {
           x[t] = i;
           if (Place(t)) Backtrack(t + 1) }
}
```

```
intn Queen(int n)
   {
   QueenX;
   //初始化X
   X. n=n;  //皇后个数
   X. sum=0;
   int*p=new int [n+1];
   for(int i=0;  i<=n;  i++)
   p[i]= 0;
   X.x=p;
   X.Backtrack(1);
   ddete [] p;
   returnX. sum;  }
```

算法复杂性:

5.7 最大团问题

基本概念

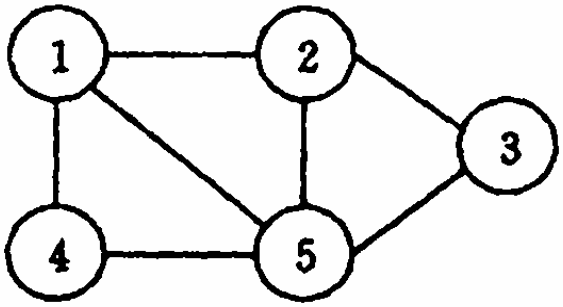
设无向图G=(V, E), U⊂V, 若对任意u, v∈U, 有(u,v) ∈ E, 则称U是G的一个完全子图。G的完全子图U是G的一个团(完备子图)当且仅当U不包含在G的更大的完全子图中。G的最大团是G中所含顶点数最多的团。

如果U⊂V,且对任意u,v∈U,(u,v)∉E, 则称U是G的一个空子图。G的空子图U是G的一个独立集当且仅当U不包含在G的更大的空子图中。G的最大独立集是G中所含顶点数最多的独立集。

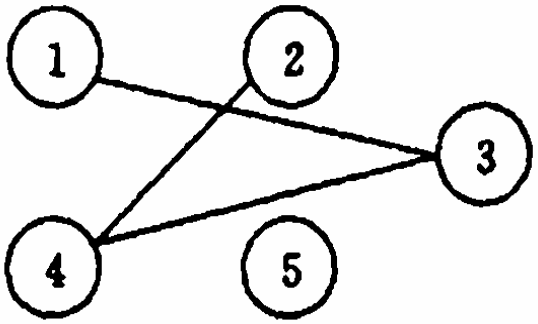
*若U是G的一个完全子图,则U是G的补图$\overline{G}$ 的一个独立集.

问题描述:在G 中找一个最大团.

例如



最大团:{1, 2, 5}, {1, 4, 5}, :{2, 3, 5}



G的补图

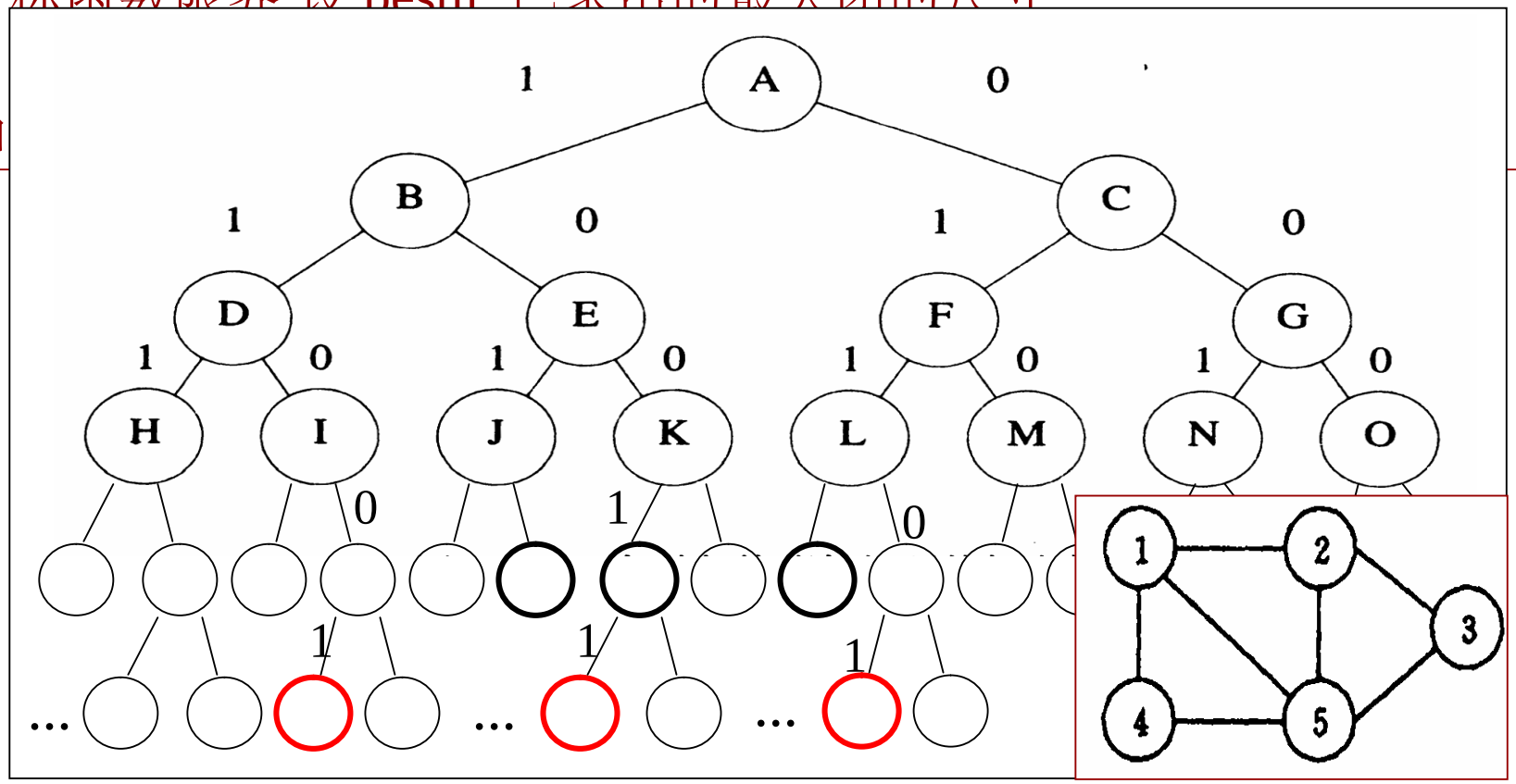
算法思路

设无向图G=(V, E), |V|=n,用邻接矩阵a表示图G, 问题的解可表示为n元向量{x1, ... xn }, xi∈{0,1}. 问题的解空间用排序树表示.

约束条件:{x1,x2,...xi}∪{xi+1}是团.

目标函数限界:设 bestn: 已求出的最大团的尺寸.

当



最大团问题的回溯算法

```
void clique::Backtrack(int i)
 { if (i>n){ //找到更大团,更新
        for (int j=1; j<=n;  j++)
          bestx[j] = x[j];
          bestn = cn;
          return; }
//检查顶点i是否与当前团相连
 int OK = 1;
 for(int j =1; j < =i ; j++)
     if (x[j]&&a[i][j]==0){//i不与j相连
         OK = 0;
         break; }
 if (OK) {
        x[i] = 1;//把i加入团
        cn ++;
        Backtrack(i+1); }
       { x[i]=0;
        cn -- ; }
 if (cn+n- i>bestn) {
     x[i] = 0;
      Backtrack(i + 1);}
       }
```

```
int MaxClique(int ** a, int v[i],
                        int n)
   Clique Y;
   Y.x = new int [n+l];
   Y.a= a; //图G的邻接矩阵
   Y.n= n; //图G顶点数
   Y.cn = 0; //当前团顶点数
   Y.bestn=0; //当前最大团顶点数
   Y.bestx = v; //当前最优解
   Y. Backtrack( 1 );
   delete [ ] Y. x;
   return Y. best
```

5.8 图的m着色问题

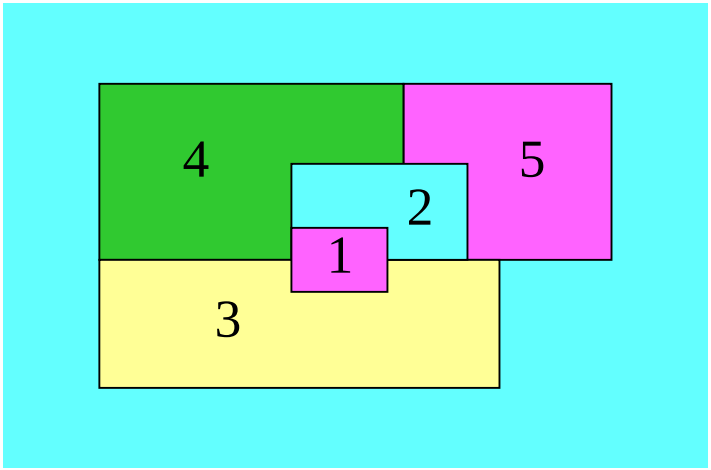
问题描述

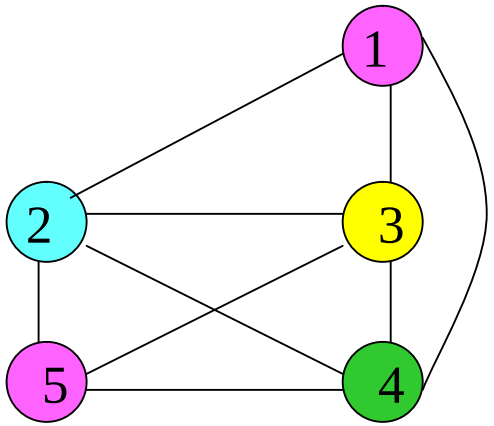
图的m色判定问题: 给定无向连通图G和m种颜色。用这些颜色为图G的各顶点
着色. 问是否存在着色方法, 使得G中任2邻接点有不同颜色。

图的m色优化问题:给定无向连通图G,为图G的各顶点着色, 使图中任2邻接点着不同颜色,问最少需要几种颜色。所需的最少颜色的数目m称为该图的色数

若图G是可平面图,则它的色数不超过4色(4色定理).

4色定理的应用:在一个平面或球面上的任何地图能够只用4种颜色来着色使得相邻的国家在地图上着有不同颜色

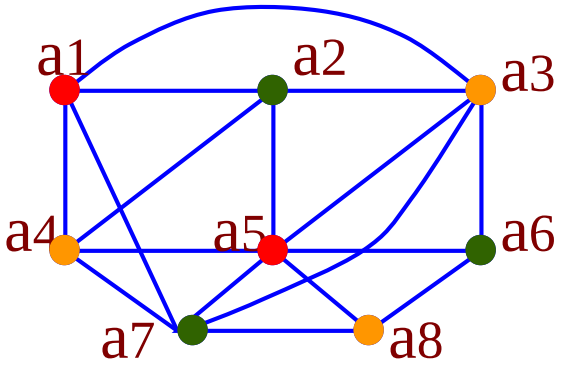




任意图的着色

Welch Powell法

a). 将G的结点按照度数递减的次序排列.
b). 用第一种颜色对第一个结点着色, 并按照结点排列的次序对与前面着色点不邻接的每一点着以相同颜色.
c). 用第二种颜色对尚未着色的点重复步骤b). 用第三种颜色继续这种作法, 直到所有点着色完为止.



- 排序: **a5, a3, a7, a1, a2, a4, a6, a8**
- 着第一色: **a5, a1,**
- 着第二色:**a3,a4, a8**
- 着第三色:**a7, a2, a6**

算法思路

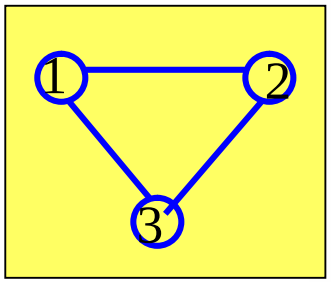
设图G=(V, E), |V|=n, 颜色数= m, 用邻接矩阵a表示G, 用整数1, 2…m来表示m种不同的颜色。顶点i所着的颜色用x[i]表示。

问题的解向量可以表示为n元组x={ x[1],...,x[n] }. x[i]∈{1,2,...,m},

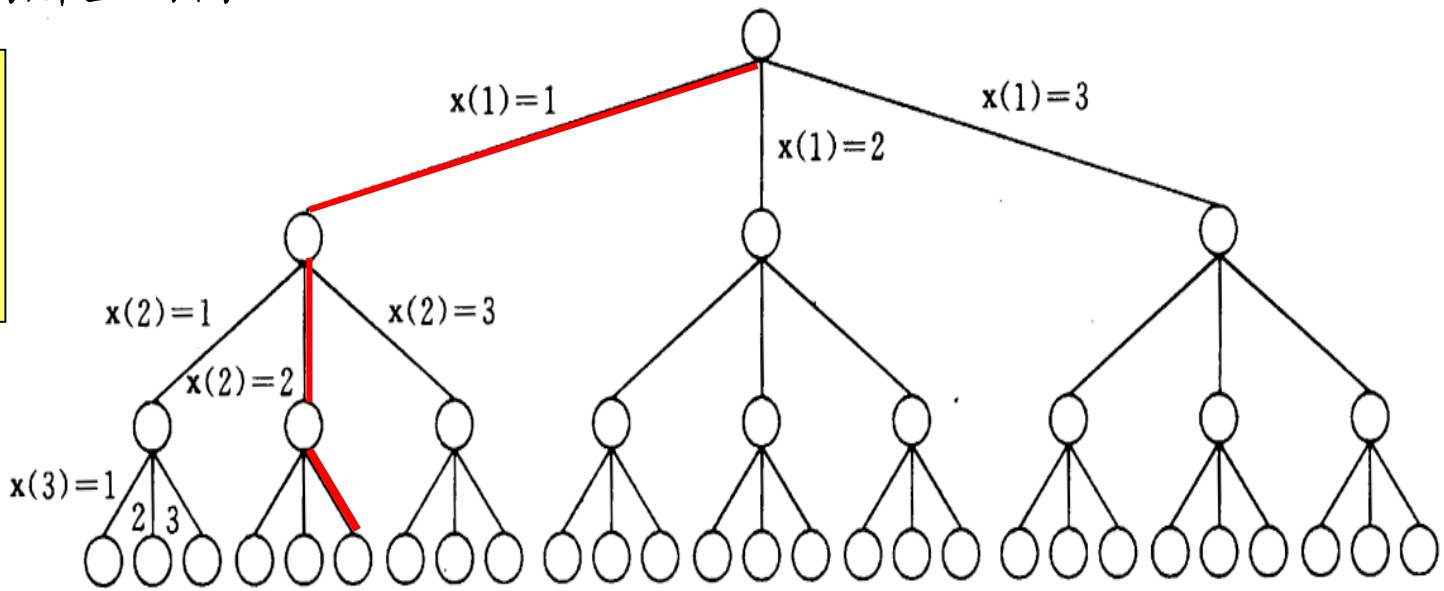
解空间树为排序树,是一棵n+1层的完全m叉树.

在解空间树中做深度优先搜索, 约束条件: x[i] ≠ x[j], 如果a[j][i]=1.

n=3, m=3时的解空间树



$$a=\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}$$



着色问题回溯算法

```
int mColoring(int n, int m,  int **a  )
 { Color X;
  //初始化X
   X．n=n;  //图的顶点数
   X．m=m  //可用颜色数
   X．a=a;  //图的邻接矩阵
   X．Sum=0; //已找到的着色方案数
   int*p=new int [n+1];
   for (int i=0;  i<=n;  i++)
    p[i]=0;
   X. x=p    //当前解;
   X. Backtrack(1);
   delete [ ]p;
   returnX．sum;  }
```

```
voidColor backtrack(int t)
 {if  (t>n){
   sum++;
   for(int i=1; i<=n; i++)
   cout << x[i] <<‘’;
   cout << endl ;   }
   else
      for ( int i=1; i<=m; i++){
         x[t]=i;
         if (Ok(t)) Backtrack( t+1);  }}
```

```
bool Color::Ok(int k)
 {//检查颜色可用性
   for(int j=1;  j<=n;  j++)
     if((a[k][j]==1)&&(x[j]==x[k]) ) return false;
   retrun true;
```

算法复杂性:

$$\sum_{i=0}^{n-1} m^i(mn) = nm(m^n - 1)/(m-1) = O(nm^n)$$