第5章　回溯法

回溯法的步骤

- 针对所给问题，定义问题的解空间；
- 确定易于搜索的解空间结构；
- 以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索。

旅行售货员问题

- 旅行售货员问题又称货郎担问题，是指某售货员要到n个城市去推销商品，已知各城市之间的路程（或旅费）。
- 售货员要选定一条从驻地出发经过每个城市一次，最后回到驻地的路线，使总的路程（总旅费）最短（最小）。

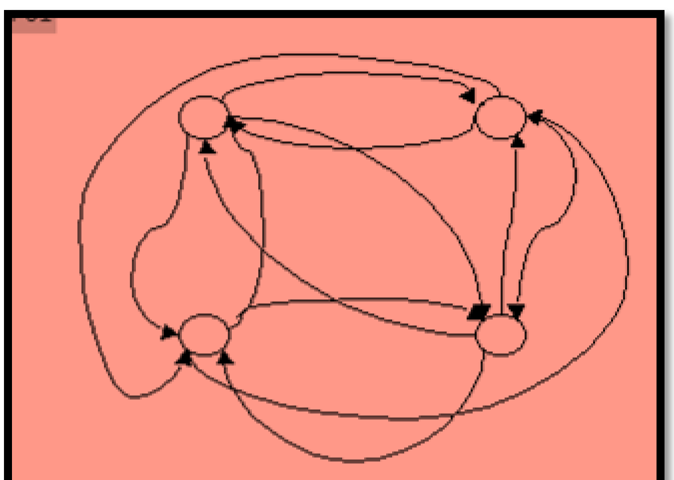
旅行售货员问题

(Traveling Saleman Problem)

- 使用图论语言描述：
  - 设图G=（V，E）是一个加权连通图
  - 要求：
    - |V|=n,
    - 边（i, j）(i不等于j)的费用c[]i[j]为正数。
- 图G的一条周游路线是经过V中的每个顶点恰好一次的一条回路。
- 周游路线的费用是这条回路上的所有边的费用之和。
- TSP问题就是要在图G中找出费用最小的周游路线。

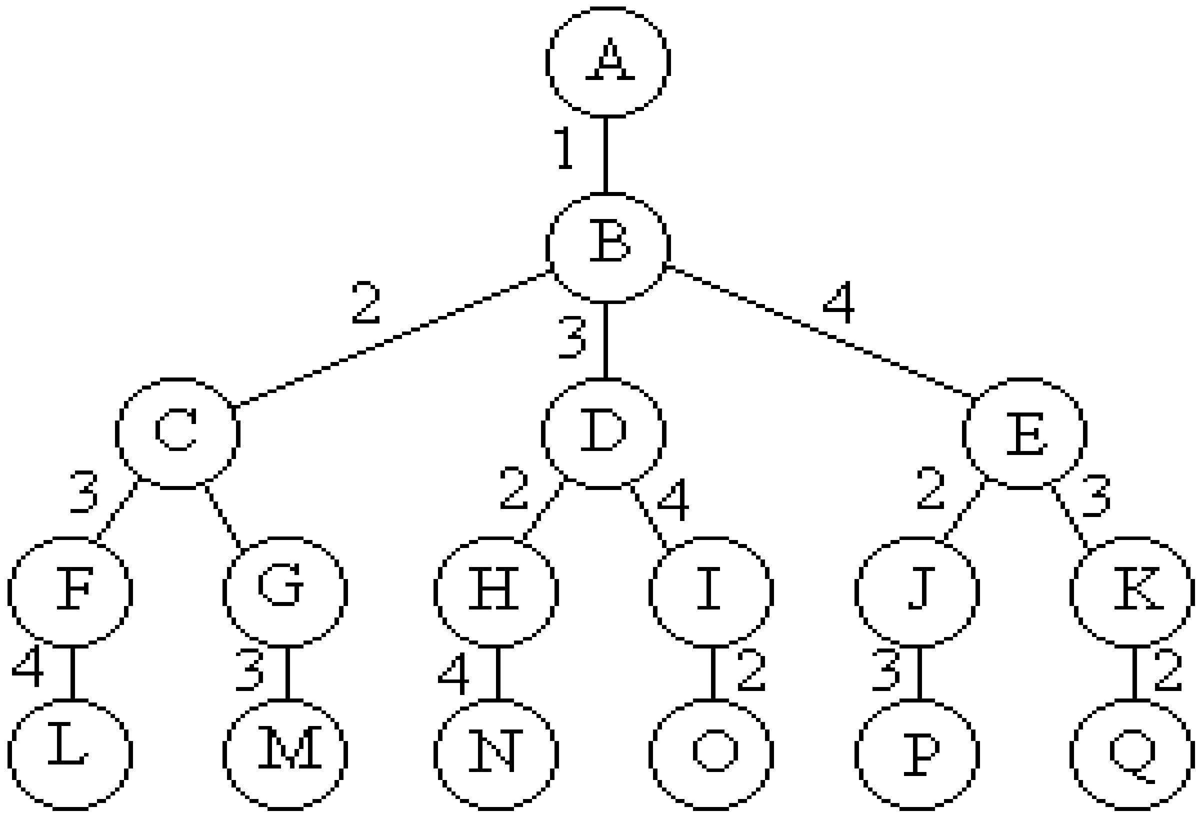
旅行售货员问题
(Traveling Saleman Problem)

- 方法一：穷举法
  - 由起始点出发的周游路线一共有(n-1)!条，即等于除始结点外的n-1个结点的排列数。
  - 旅行售货员问题是一个排列问题。
  - 穷举法的时间复杂度：O(n!)



$$C = \begin{bmatrix} \infty & 2 & 1 & 3 \\ 1 & \infty & 6 & 1 \\ 3 & 4 & \infty & 4 \\ 7 & 1 & 5 & \infty \end{bmatrix}$$

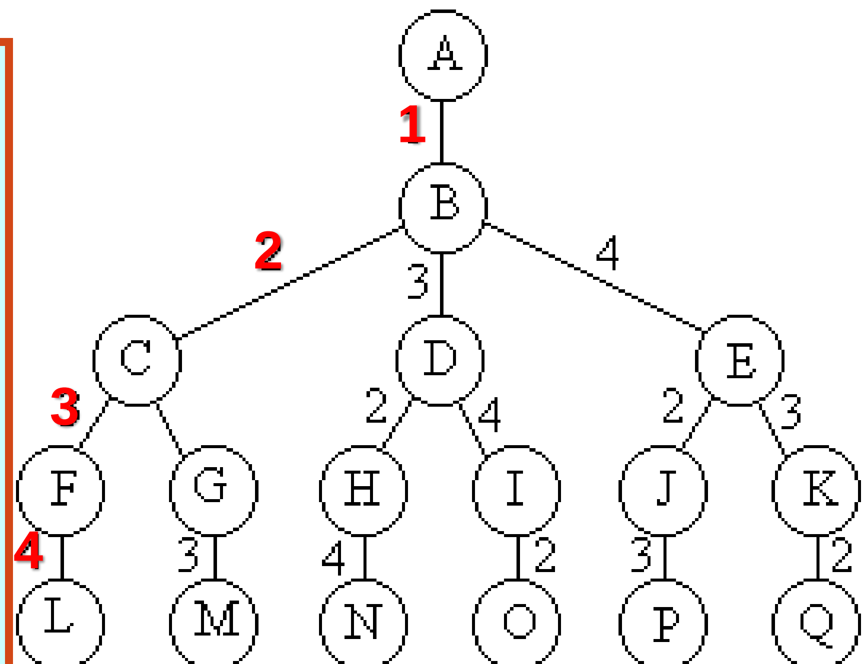
|V|=4旅行售货员问题的解空间



(6) 子集树与排列树

- 排列树：当所给问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树。如：旅行售货员问题的解空间树。
- 遍历排列树需要O(n!)计算时间。

```
void backtrack (int t)
{
  if (t>n) output(x);
    else
      for (int i=t;i<=n;i++) {
        swap(x[t], x[i]);
        if (legal(t)) backtrack(t+1);
        swap(x[t], x[i]);
      }
}
```



```
template < class Type >
class Traveling {
   friend Type TSP(int * * , int [], int, Type);
   private:
      void Backtrack(int i);
      int n,         // 图 G 的顶点数
          * x,       // 当前解
          * bestx;   // 当前最优解
   Type * * a,       // 图 G 的邻接矩阵
         cc,         // 当前费用
         bestc,      // 当前最优值
         NoEdge;     // 无边标记
   };

template < class Type >
void Traveling < Type >::Backtrack(int i)
{
  if (i == n) {
    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)) {
      for (int j = 1; j <= n; j++) bestx[j] = x[j];
      bestc = cc + a[x[n-1]][x[n]] + a[x[n]][1];}
   }
  else {
   for (int j = i; j <= n; j++)
     // 是否可进入 x[j] 子树?
```

```
if (a[x[i-1]][x[j]] != NoEdge &&
  (cc + a[x[i-1]][x[j]] < bestc || bestc == NoEdge)
  // 搜索子树
  Swap(x[i], x[j]);
  cc += a[x[i-1]][x[i]];
  Backtrack(i+1);
  cc -= a[x[i-1]][x[i]];
  Swap(x[i], x[j]);}
}
}
```

5.4.1 迷宫问题

- 1、形式化描述问题：
  - 迷宫——用A[ ][ ]描述迷宫，
  - 当A[i,j]=0 表示该房间为空，当A[i,j]=1 表示该房间封闭
- 2、解的形式：
  - $(x_1, x_2, \dots, x_n)$
  - n取值不确定　　$X_i=\{1, 2, 3, 4\}$---显约束
- 3、隐约束条件
  - 当A[i,j]=2 表示该房间已经走过，不能再走； A[i,j]=1 表示该房间封闭，也不能走；
  - 每步走的方向：上下左右

Sum=1; a[i_l][j_l]=2; a[i_z][j_z]=3;   //sum为所能走过的房间数

for(i=1;i<=m;i++)

  for(j=1;j<=n;j++)

    if(a[i][j]==0) sum++;

i= i_l ; j= j_l ; start=1; s=1;//s为树层次， start表示四个方向

While(1)

{ for(k=start;k<=4; k++)

  { i_l =i+v[k]; j_l=j+u[k];

    if(i_l <1 or i_l >m or j_l<1 or j_l>n) continue;

    if (a[i_l][j_l]==0) {}

        else {}   }

  if(k>4) {} //回溯 else {}}

走到了墙边界

空房间

房间不空

未找到合适的方向

空房间执行的操作：

if ($a[i_l][j_l]$==0)

{ i =i_l; j = j_l; //记录当前位置

x[s]=k; //记录决策

s=s+1; //下一步（纵深搜索）

$a[i_l][j_l]$=2; //房间已经走过

start=1; break; //又从1开始

}

if(k>4) //回溯

{a[i][j]=0;s--; L=x[s];

i=i-v[L]; j=j-u[L];

start=L+1;

决策回退

}

else { if(s<sum) 输出解;

break;}

房间不空执行的操作：

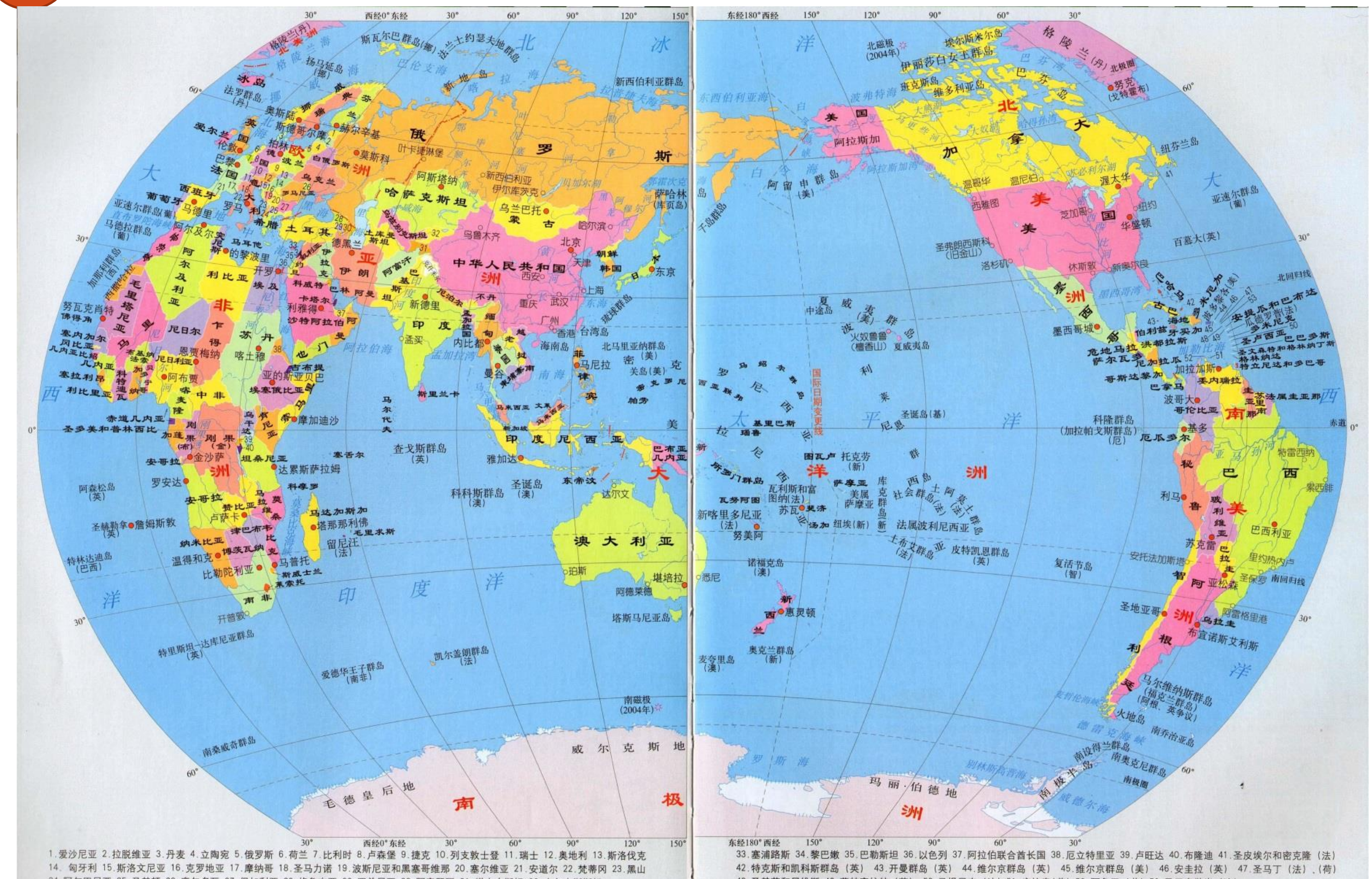
else { if ($a[i_l][j_l]$==3 or s==sum)

{x[s]=k; break;}

}

```
Sum=1; a[i_l][j_l]=2; a[i_z][j_z]=3;
for( i=1;i<=m;i++)
   for( j=1;j<=n;j++)
       if(a[i][j]==0)  sum++;
i= i_l  ; j= j_l ;  start=1;  s=1;
While(1)
{  for(k=start;k<=4; k++)
   { i_l =i+v[k]; j_l=j+u[k];
       if(i_l <1 or i_l >m  or j_l<1 or j_l>n)
     continue;
     if ( a[i_l][j_l]==0 )
     { i =i_l; j = j_l; //记录当前位置
         x[s]=k; //记录决策
         s=s+1; //下一步（纵深搜索）
        a[i_l][j_l]=2; //房间已经走过
         start=1; break; //又从1开始
        }
else { if (a[i_l][j_l]==3 or s==sum)
                {x[s]=k; break;}
          }
 }
if(k>4)
 {   a[i][j]=0;s--; L=x[s];
     i=i-v[L]; j=j-u[L];
     start=L+1;
   }
 else  if(s<sum)
                {输出解 break;}
}
```

5.4.2 图的m着色问题

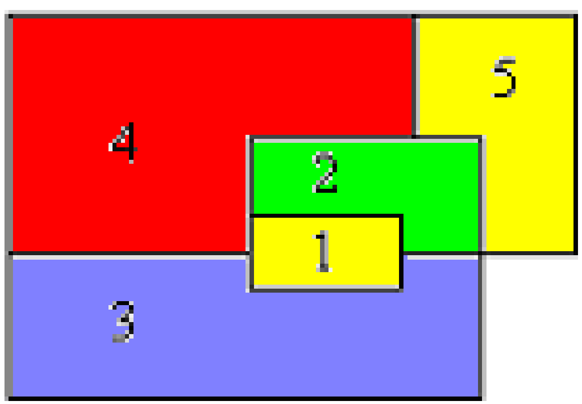


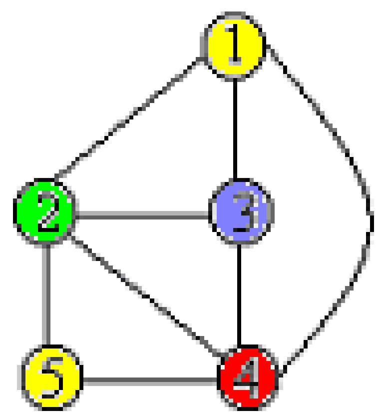
亚洲
欧洲
非洲
北美洲
南美洲
大洋洲
南极洲
太平洋
大西洋
印度洋
北冰洋
俄罗斯
中华人民共和国
印度
澳大利亚
加拿大
美国
巴西
毛德皇后地
玛丽·伯德地
1.爱沙尼亚 2.拉脱维亚 3.丹麦 4.立陶宛 5.俄罗斯 6.荷兰 7.比利时 8.卢森堡 9.捷克 10.列支敦士登 11.瑞士 12.奥地利 13.斯洛伐克
14. 匈牙利 15.斯洛文尼亚 16.克罗地亚 17.摩纳哥 18.圣马力诺 19.波斯尼亚和黑塞哥维那 20.塞尔维亚 21.安道尔 22.梵蒂冈 23.黑山
24.阿尔巴尼亚 25.马其顿 26.摩尔多瓦 27.保加利亚 28.格鲁吉亚 29.亚美尼亚 30.阿塞拜疆 31.塔吉克斯坦 32.吉尔吉斯斯坦
33.塞浦路斯 34.黎巴嫩 35.巴勒斯坦 36.以色列 37.阿拉伯联合酋长国 38.厄立特里亚 39.卢旺达 40.布隆迪 41.圣皮埃尔和密克隆（法）
42.特克斯和凯科斯群岛（英） 43.开曼群岛（英） 44.维尔京群岛（英） 45.维尔京群岛（美） 46.安圭拉（英） 47.圣马丁（法）、(荷)
48.圣基茨和尼维斯 49.蒙特塞拉特（英） 50.马提尼克（法） 51.库拉索(荷) 52.阿鲁巴（荷） 53.圣巴泰勒米（法）

5.4.2 图的m着色问题

■给定无向连通图G和m种不同的颜色。用这些颜色为图G的各顶点着色，每个顶点着一种颜色。是否有一种着色法使G中每条边的2个顶点着不同颜色。这个问题是图的m可着色判定问题。

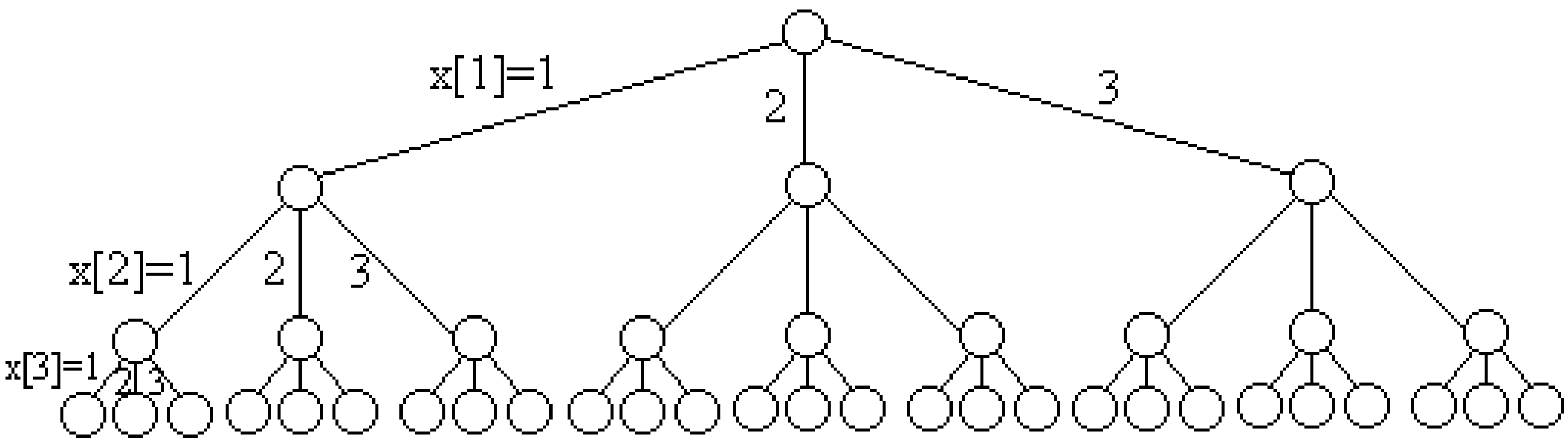
■若一个图最少需要m种颜色才能使图中每条边连接的2个顶点着不同颜色，则称这个数m为该图的色数。求一个图的色数m的问题称为图的m可着色优化问题。



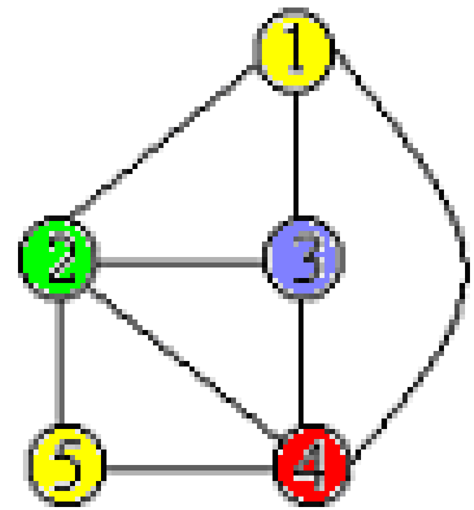


5.4.2 图的m着色问题

- 解向量：(x_1， x_2， … ， x_n)表示顶点i所着颜色x[i]
- 可行性约束函数：顶点i与已着色的相邻顶点颜色不重复。
- 解空间树：n=3，m=3



5.4.2 图的m着色问题




```
private static void backtrack(int   t)
  {
    if (t>n)  {sum++; 输出解； }
    else
p
{
    if (a[k][j] && (x[j]==x[k])) return false;
    return true;
}}
```

复杂度分析

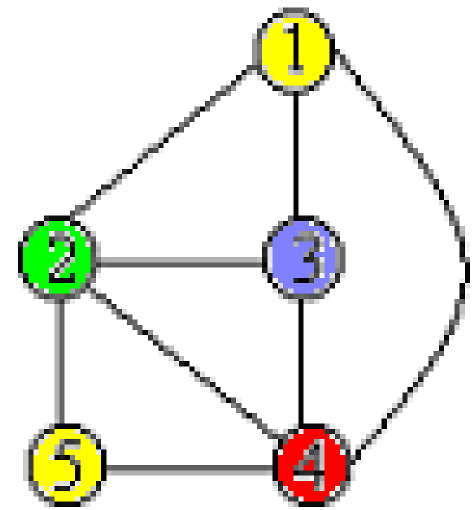
图m可着色问题的解空间树中内结点个数是 $\sum_{i=0}^{n-1} m^i$

对于每一个内结点，在最坏情况下，用ok检查当前扩展结点的每一个儿子所相应的颜色可用性需耗时O(mn)。

因此，回溯法总的时间耗费是

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

M_coloring(int n, int m, int g[ ][ ]) //非递归算




```
{ for(i=1;i<=n;i++)  x[i]=1;
  k=1;
  do{ if(x[k]<=m)
      {  for (i=1;i<k;i++)
           {if(g[i][k]==1 and x[i]==x[k])  break;}
         if(i<k)  x[k]++;
         else  k++;}
     else{ x[k]=1;k=k-1;
           if(k>=1) x[k]++;  }
  }while(k<=n and k>=1)
  if(k>n)  输出解；  if(k<1)  无解；  }
```

树的深度

非正常退出，未找到合适的值，换颜色

找到合适的值，向下纵深搜索，试探

回溯

M_coloring(int n, int m, int g[ ][ ]) //非递归算法2

```
{ for(i=1;i<=n;i++)  x[i]=0;
  k=1;
  while(k>0){ x[k]=x[k]+1;
      while(x[k]<=m&&!color(k))  x[k]=x[k]+1;
      if(x[k]<=m)
      {  if(k==n){
            输出x[ ];
            break;}
         else  k++;}
    else{ x[k]=0;k=k-1; }
  } }
```

未找到合适的值，换颜色

试探

回溯

Bool Color(k)

```
{ for(i=1;i<=n;i++)
     f(g[i][k]==1 and x[i]==x[k]) return(false);
  return(true);
}
```