## 无向连通图如何存储

联想在离散数学中所学习的图的知识，如果使用邻接矩阵来存储图，那么无向连通图的邻接矩阵有以下两点性质：

1. adjacent[v][w]=adjacent[w][v]；
2. 离散数学中有判定一个邻接矩阵所描述的图是否是连通图的算法

## 为什么退出递归的条件是 if (t > count)

第一层函数：

mColoring(int m)//这是第一层函数，用来调用第二层的递归函数

{

int x[graph\_size];

colors = m; //颜色数

int sum = 0; //统计所有的着色方案

BackTrack(1, sum,x);//1说明图中的结点是从1开始计数的，当然图中的结点也可以从0开始计数，如果从0开始计数，那么下面的条件就要改成t〉=count

return sum;

}

第二层函数：

template<int graph\_size>

void Connected\_Graph<graph\_size>::BackTrack(int t, int sum)

{

if (t > count){

sum++;

for (int i = 0; i < count; i++)

cout << x[i] << ' ';

cout << endl;

}

else{

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

x[t] = i;

if (Ok(t))

BackTrack(t + 1);

x[t] = 0;//回溯回来时表示当前路径不是合法解，那么就要撤销之前的赋值

}

}

}

正如昨天在“[回溯算法框架解析](回溯算法框架解析.docx)”一文中叙述的，从根结点到叶结点的路径表示问题的解，该图中共有count个结点，对这count个结点着色，那么就需要count条边来组成路径。根据几何学的知识，连接这count个结点共需要count+1条点。

在上述回溯函数中，参数t代表的是递归的层次，对应于解空间树中的结点。因此递归的层次应该一共有count+1层，即从1到count+1.

再从另外一个角度来解释这个问题，我们现在来考虑上述回溯函数的倒数第二次调用BackTrack（count），显然函数此时会执行else分支中的代码，此时会对x[count]进行赋值，if (Ok(count))成立，那么此时我们已经得到了问题的完整解了，接着函数会调用BackTrack(count + 1)，显然函数此时会执行if (count+1 > count)分支的代码，也就是将问题的解打印出来。

所以在此我们可以总结：回溯函数的最后一次调用执行的代码是打印问题的解