递归，回溯，DFS，BFS的理解和模板

<http://blog.sina.com.cn/s/blog_eb52001d0102v26v.html>

http://blog.sina.com.cn/s/blog\_eb52001d0102v1t0.html

<https://github.com/yuzhangcmu/LeetCode/blob/master/recursion/Factorial.java>

https://github.com/yuzhangcmu/LeetCode/blob/master/dp/Fibonacci.java

LeetCode 里面很大一部分题目都是属于这个范围，例如Path Sum用的就是递归+DFS，Path Sum2用的是递归+DFS+回溯  
  
这里参考了一些网上写得很不错的文章，总结一下理解与模板  
递归：就是出现这种情况的代码： （或者说是用到了栈）  
解答树角度：在dfs遍历一棵解答树        
优点：结构简洁  
缺点：效率低，可能栈溢出  
  
递归的一般结构：  
void f()    
{    
     if(符合边界条件)    
    {    
       ///////    
        return;    
    }    
        
     //某种形式的调用    
     f();    
}    
  
回溯：递归的一种，或者说是通过递归这种代码结构来实现回溯这个目的。回溯法可以被认为是一个有过剪枝的DFS过程。  
解答树角度：带回溯的dfs遍历一棵解答树  
回溯的一般结构：  
[java] view plaincopy  
void dfs(int 当前状态)    
    {    
          if(当前状态为边界状态)    
          {    
            记录或输出    
            return;    
          }    
          for(i=0;i  
         {    
               //扩展出一个子状态。    
               修改了全局变量    
               if(子状态满足约束条件)    
                {    
                  dfs(子状态)    
               }    
                恢复全局变量//回溯部分    
            }    
    }   
  
BFS和DFS是相似。  
BFS（显式用队列）  
DFS（隐式用栈）（即递归）  
当然，对于DFS，用递归可能会造成栈溢出，所以也可以更改为显示栈。  
BFS：典型例题：P101 对于二叉树的层次遍历，P108对于图的走迷宫最短路径  
  
将（起始）首节点加入队列： q.push(head);    
                     标记首节点已经被访问：         isvisited[head]=true;    
                    以下自动反应：                       while(!q.empty())    
                                                                    {    
                                                                          int temp=q.front();    
                                                                           q.pop();    
                                                                          访问temp，并标记temp已被访问过，将temp的子相关节点加入队列    
                                                                          q.push(temp相关节点);    
                                                                    }    
  
**DFS:典型例题：P107黑白图像**  
格式：将所有节点遍历一遍，在遍历每个节点是，DFS的遍历该节点相关的所有节点  
[cpp] view plaincopy  
void dfs(int x, int y)    
{    
if(!mat[x][y] || vis[x][y]) return;     // 曾经访问过这个格子，或者当前格子是白色    
vis[x][y] = 1;                          // 标记(x,y)已访问过    
dfs(x-1,y-1); dfs(x-1,y); dfs(x-1,y+1);    
dfs(x-1,y);               dfs(x,y+1);    
dfs(x+1,y-1); dfs(x+1,y); dfs(x+1,y+1); // 递归访问周围的八个格子    
}    
主循环：    
for(int i = 1; i <= n; i++)    
for(int j = 1; j <= n; j++)    
if(!vis[i][j] && mat[i][j])    
{    
count++;    
dfs(i,j);    
} // 找到没有访问过的黑格