递归

前置

递归的想法

将一个大问题化成若干小问题去解决或化成若干步骤去解决（简化问题思想）

如何写递归

1. 想想什么是最简单/特殊的情况（这往往是递归的出口）
2. 思考如何将当前问题关联到若干小问题上去解决
3. 按上面的想法用代码实现（不要纠结函数具体会如何运作，先把你的逻辑直观地用递归代码表现出来）

何时用递归

A、问题的解可以用递推式表达（如n的阶乘，斐数列）
B、问题的解法是递归（回溯）
C、数据结构是递归定义的（链表，树）

回溯法

将问题的解决步骤画成一棵树，树的最大深度位置就是返回并获得结果的位置，每个结点都会分出很多分支，分支的宽度就是for循环探索的宽度，在for循环里用递归往深处探索，回溯就是在探索结束后回来，并取消刚刚在这一层for循环里添加的记录。

```
void function() {
        if (判断是否结束) {
                收获结果
                return;
        }
        for () {
                if (是否满足要求) {
                        标记
                        function(向更深一层探索);
                        回溯：取消标记
                }
        }
}
```

N皇后问题

> *如何能够在 N×N 的国际象棋棋盘上放置N个皇后，使得任何一个皇后都无法直接吃掉其他的皇后？为了达到此目的，任两个皇后都不能处于同一条横行、纵行或斜线上。请输出所有的摆放方法*

```cpp
#include<iostream>
using namespace std;
int a[20][20];//棋盘
int colFlag[20];//用colFlag[col]记录这一列上有没有皇后
int leftBias[40];//坐标=col-row的位置在一个左斜线上，用leftBias[col-row]表示该斜线上是否有皇后
int rightBias[40];//坐标=col+row的位置在一个右斜线上，用rightBias[col+row]表示该斜线上是否有皇后
int total=0;
//1、isOk()函数一开始总写不对，两个斜线的关系数组没想出来
//2、何时撤销标记有点不清楚
bool isOk(int i, int j) {//判断a[i][j]能不能放皇后
        if(leftBias[j-i]==1||rightBias[i+j]==1)return  false;
        if (colFlag[j] == 1)return false;
        else return true;
}
void Place(int N, int row) {
        //row>N说明第N行已经放了皇后，可以展示结果了
        if (row > N) {
                for (int i = 1; i <= N; i++)
                        for (int j = 1; j <= N; j++) {
                                if (a[i][j] == 1)cout << '(' << i << "," << j << ')' << " ";
                        }
                cout << endl;
                total++;//结果总数+1
                return;
        }
        //每一行都试探所有位置（N列），看能不能放
        for (int col = 1; col <= N; col++) {
                if (isOk(row, col)) {//如果可以放
                        //放置皇后的同时做好记录
                        a[row][col] = 1;
                        colFlag[col] = 1;
                        leftBias[col - row] = 1;
                        rightBias[col + row] = 1;
                        //下一行放皇后
                        Place(N, row + 1);
                        //回溯：又回到这一行，试探其他位置，在这之前，要先把刚刚的标记和放上的皇后取消
                        a[row][col] = 0;
                        colFlag[col] = 0;
                        leftBias[col - row] = 0;
                        rightBias[col + row] = 0;
                }
        }
}
int main() {
        int row = 1;
        int N = 8;
        Place(N, row);
        cout << "共有" << total << "种解法";
        return 0;
}
```

数据结构是递归定义的

链表

树

前序遍历递归复制树

```
template<class T>
TreeNode<T>* copy(TreeNode<T>* &a, TreeNode<T>* b) {
        if (b == nullptr)return nullptr;
        else {
                a = new TreeNode<T>;
                a->data = b->data;
                a->leftChild = copy(a->leftChild, b->leftChild);
                a->rightChild = copy(a->rightChild, b->rightChild);
                return a;
        }
}
```

后序遍历递归计算树的高度

```
int getHeight(TreeNode<T>* root) {//规定空树高度为0，一个结点时高度为1
                if (root == nullptr)return 0;
                else {
                        //结点高度=max（左子树高度，右子树结点高度）+1
                        int heightLeft=getHeight(root->leftChild);
                        int heightRight=getHeight(root->rightChild);
                        return (heightLeft > heightRight) ? (heightLeft + 1) : (heightRight + 1)
                }
        }
```

后序遍历递归计算树的结点个数

```
int getNodeCount(TreeNode<T>* root) {
                //后序遍历递归计算树的结点个数
                //总节点数=左树结点数+右树结点数+1
                if (root == nullptr)return 0;
                else {
                        int leftCount = getNodeCount(root->leftChild);
                        int rightCount = getNodeCount(root->rightChild);
                        return leftCount + rightCount + 1;
                }
        }
```