阅读：23,610　　作者：解学武

回溯法(八皇后问题)及C语言实现

上一节　　下一节

回溯法，又被称为“试探法”。解决问题时，每进行一步，都是抱着试试看的态度，如果发现当前选择并不是最好的，或者这么走下去肯定达不到目标，立刻做回退操作重新选择。这种走不通就回退再走的方法就是回溯法。

例如，在解决列举集合 {1,2,3} 中所有子集的问题中，就可以使用回溯法。从集合的开头元素开始，对每个元素都有两种选择：取还是舍。当确定了一个元素的取舍之后，再进行下一个元素，直到集合最后一个元素。其中的每个操作都可以看作是一次尝试，每次尝试都可以得出一个结果。将得到的结果综合起来，就是集合的所有子集。

实现代码为：

```
01.  #include <stdio.h>
02.  //设置一个数组，数组的下标表示集合中的元素，所以数组只用下标为1，2，3的空间
03.  int set[5];
04.  //i代表数组下标，n表示集合中最大的元素值
05.  void PowerSet(int i,int n){
06.      //当i>n时，说明集合中所有的元素都做了选择，开始判断
07.      if (i>n) {
08.          for (int j=1; j<=n; j++) {
09.              //如果树组中存放的是 1，说明在当初尝试时，选择取该元素，即对应的数组下标，所以，可以输出
10.              if (set[j]==1) {
11.                  printf("%d ",j);
12.              }
13.          }
14.          printf("\n");
15.      }else{
16.          //如果选择要该元素，对应的数组单元中赋值为1；反之，赋值为0。然后继续向下探索
17.          set[i]=1;PowerSet(i+1, n);
18.          set[i]=0;PowerSet(i+1, n);
19.      }
20.  }
21.  int main() {
22.      int n=3;
23.      for (int i=0; i<5; i++) {
24.          set[i]=0;
25.      }
26.      PowerSet(1, n);
27.      return 0;
```

```
28.  }
```

运行结果：

```
1 2 3
1 2
1 3
1
2 3
2
3
```

回溯VS递归

很多人认为回溯和递归是一样的，其实不然。在回溯法中可以看到有递归的身影，但是两者是有区别的。

回溯法从问题本身出发，寻找可能实现的所有情况。和穷举法的思想相近，不同在于穷举法是将所有的情况都列举出来以后再一一筛选，而回溯法在列举过程如果发现当前情况根本不可能存在，就停止后续的所有工作，返回上一步进行新的尝试。

递归是从问题的结果出发，例如求 n！，要想知道 n！的结果，就需要知道 n*(n-1)! 的结果，而要想知道 (n-1)! 结果，就需要提前知道 (n-1)*(n-2)!。这样不断地向自己提问，不断地调用自己的思想就是递归。

回溯和递归唯一的联系就是，回溯法可以用递归思想实现。

回溯法与树的遍历

使用回溯法解决问题的过程，实际上是建立一棵“状态树”的过程。例如，在解决列举集合{1,2,3}所有子集的问题中，对于每个元素，都有两种状态，取还是舍，所以构建的状态树为：

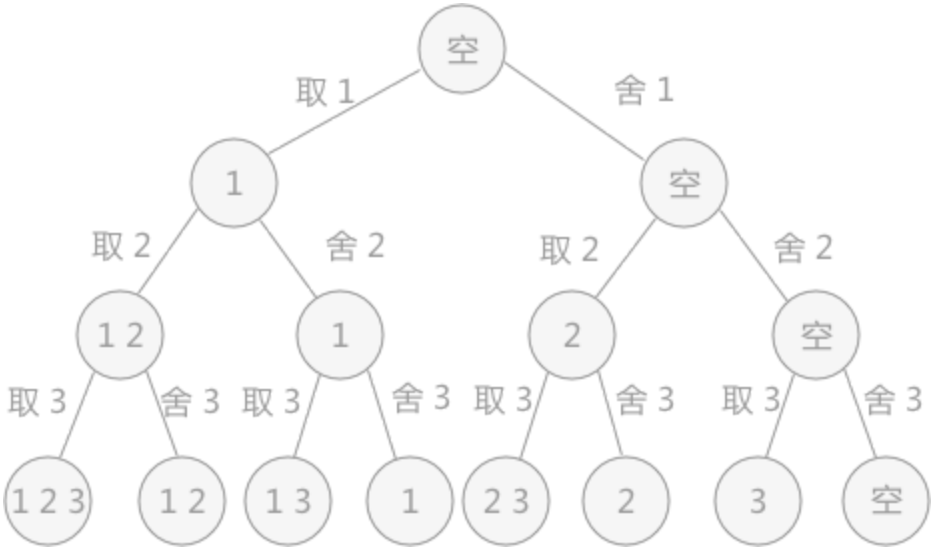

图1 状态树

回溯法的求解过程实质上是先序遍历“状态树”的过程。树中每一个叶子结点，都有可能是问题的答案。图 1 中的状态树是满二叉树，得到的叶子结点全部都是问题的解。

在某些情况下，回溯法解决问题的过程中创建的状态树并不都是满二叉树，因为在试探的过程中，有时会发现此种情况下，再往下进行没有意义，所以会放弃这条死路，回溯到上一步。在树中的体现，就是在树的最后一层不是满的，即不是满二叉树，需要自己判断哪些叶子结点代表的是正确的结果。

回溯法解决八皇后问题

八皇后问题是以国际象棋为背景的问题：有八个皇后（可以当成八个棋子），如何在 8*8 的棋盘中放置八个皇后，使得任意两个皇后都不在同一条横线、纵线或者斜线上。

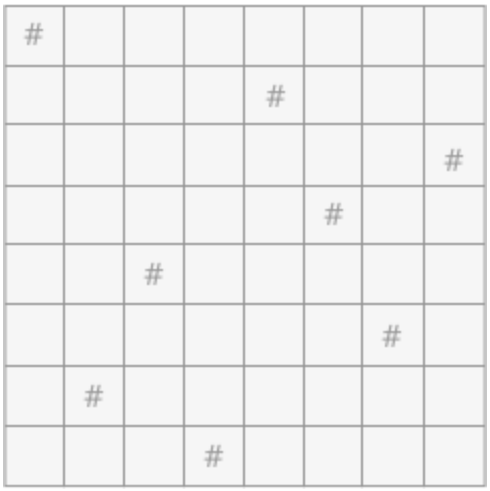

图 2 八皇后问题示例（#代表皇后）

八皇后问题是使用回溯法解决的典型案例。算法的解决思路是：

1. 从棋盘的第一行开始，从第一个位置开始，依次判断当前位置是否能够放置皇后，判断的依据为：同该行之前的所有行中皇后的所在位置进行比较，如果在同一列，或者在同一条斜线上（斜线有两条，为正方形的两个对角线），都不符合要求，继续检验后序的位置。
2. 如果该行所有位置都不符合要求，则回溯到前一行，改变皇后的位置，继续试探。
3. 如果试探到最后一行，所有皇后摆放完毕，则直接打印出 8*8 的棋盘。最后一定要记得将棋盘恢复原样，避免影响下一次摆放。

实现代码：

```
01.  #include <stdio.h>
02.  int Queenes[8]={0},Counts=0;
03.  int Check(int line,int list){
04.      //遍历该行之前的所有行
05.      for (int index=0; index<line; index++) {
06.          //挨个取出前面行中皇后所在位置的列坐标
07.          int data=Queenes[index];
08.          //如果在同一列，该位置不能放
09.          if (list==data) {
```

```
            return 0;
        }
        //如果当前位置的斜上方有皇后，在一条斜线上，也不行
        if ((index+data)==(line+list)) {
            return 0;
        }
        //如果当前位置的斜下方有皇后，在一条斜线上，也不行
        if ((index-data)==(line-list)) {
            return 0;
        }
    }
    //如果以上情况都不是，当前位置就可以放皇后
    return 1;
}
//输出语句
void print()
{
    for (int line = 0; line < 8; line++)
    {
        int list;
        for (list = 0; list < Queenes[line]; list++)
            printf("0");
        printf("#");
        for (list = Queenes[line] + 1; list < 8; list++){
            printf("0");
        }
        printf("\n");
    }
    printf("================\n");
}

void eight_queen(int line){
    //在数组中为0-7列
    for (int list=0; list<8; list++) {
        //对于固定的行列，检查是否和之前的皇后位置冲突
        if (Check(line, list)) {
            //不冲突，以行为下标的数组位置记录列数
            Queenes[line]=list;
            //如果最后一样也不冲突，证明为一个正确的摆法
            if (line==7) {
                //统计摆法的Counts加1
                Counts++;
                //输出这个摆法
                print();
                //每次成功，都要将数组重归为0
                Queenes[line]=0;
                return;
```

```
57.                 }
58.                 //继续判断下一样皇后的摆法，递归
59.                 eight_queen(line+1);
60.                 //不管成功失败，该位置都要重新归0，以便重复使用。
61.                 Queenes[line]=0;
62.             }
63.         }
64.     }
65.     int main() {
66.         //调用回溯函数，参数0表示从棋盘的第一行开始判断
67.         eight_queen(0);
68.         printf("摆放的方式有%d种",Counts);
69.         return 0;
70.     }
```

大家可以自己运行一下程序，查看运行结果，由于八皇后问题有92种摆法，这里不一一列举。

上一节 下一节