**1．遍历问题**

**【算法分析】**

根据二叉树先序遍历和后序遍历的特点，可以知道，先序遍历的第一个结点是后序遍历的最后一个结点，对于中序遍历来说却是中间的一个结点，这里所说的中间也只是相对而言的中间。如果一棵二叉树的根结点没有左子树，那么先序遍历的第一个结点也是中序遍历的第一个结点，如果一棵二叉树的根结点没有右子树，那么先序遍历的第一个结点是中序遍历的最后一个结点。我们这里还认为就是中序遍历的中间结点，上面两种情况只是特殊的情况。

设二叉树的结点总数为n(对于输入的字符串来说是它的长度)，对于先序遍历的结果，第一个结点为根结点，从第二个结点到最后一个结点分为n种情况：

根结点的左子树结点个数为n-1，右子树结点的个数为0；

根结点的左子树结点个数为n-2，右子树结点的个数为1；

……

根结点的左子树结点个数为n-i，右子树结点的个数为i-1；{0<＝i<=n-1)；

……

根结点的左子树结点个数为0，右子树结点的个数为n-1。

根据这n种情况，分别将二叉树拆分为左子树和右子树，左子树结点个数为n-i，右子树结点的个数为i-l(0<＝i<＝n-1)，先序遍历的结果是从第二个结点(字符)开始取，而后序遍历的结果里是从第1个结点字符开始取。也就是说对于每一种情况，分两步处理：第一步在先序遍历和后序遍历的结果里取左子树，看是否符合规则，统计这部分可能有的中序遍历的结果数目；第二步在先序遍历和后序遍历的结果里取右子树，看是否符合规则，统计这部分可能有的中序遍历的结果数目。这两步都递归调用了统计过程，不再递归调用的条件就是当统计的是空树或只有一个结点的树，这时返回的值是可能有的中序遍历结果数目。

结合“分类相加原理”和“分步相乘原理”，可以得到下面的递归函数：

|  |
| --- |
| Function count (先序结果first，后序结果last : string) : longint;  begin  Len:=遍历结果的长度；  如果len为0或1，则返回结果即count:=l  否则 begin  t为当前统计后符合条件的数目，初值为0；  分类统计for i:=len-1 downto 0 do  begin  在first中取出长度为i的左子树结果LF；  在last中取出长度为i的左子树结果LL；  在first中取出长度为len-1-i的左子树结果RF；  在last中取出长度为len-1-i的右子树结果RL；  如果LF、LL符合基本规则(LF的首字符跟LL的尾字符相同、LF中，所有的  字符在LL中也都有)  并且RF、RL也符合基本规则，那么  t:=t+count(LF，LL)\*count(RF，RL);  {分步相乘、分步相加}  {这里count函数中递归调用了count}  end;  返回值为t即count:=t;  end;  end;  其中，检查先序结果和后序结果两个字符串是否符合基本规则，可以再通过一个函数来实现：  function check(先序字符串F，后序字符串L)：boolean；  begin  Check:=true;  如果F的首字符不等于L的尾字符则check:=false;  从F的第二个字符取到最后一个字符，如果该字符不在L中，则check:=false;  end; |

**【思考与提高】**

上面的算法通过递归，结合统计的基本原理“分步相乘，分类相加”，从而统计出所有可能解的个数，如果输入的两个字符串没有解，上述算法同样能得到结果。

在肯定有解的情况下，上述算法最终可以递归调用到0、1个结点，如果有多组解，那么调用到两个结点时，如先序为ab、后序为ba，此时有可能有如下两种结构：

a a

/ \

b b

这两种结构的中序遍历结果分别为：ba、ab，有两种。

根据分步相乘的原理，对比两个字符串，每出现一次如上的情况，可能有的结构数目(结构不同，中序遍历结果也不同，因此可能有的二叉树结构的数目就是可能有的中序遍历结果数目)就乘以2一次，最终得到总的数目。这也可以理解为一种递推的方法。

从这里可以看到，在肯定有解的情况下，给定先序遍历的结果和后序遍历的结果，可能有2n种可能的结构，也就是中序遍历可能得到2n种不同的结果，其中n>＝0。那么这里的n最大可能是多少呢？可以证明n的最大值为字符串的长度加1整除2。

递推的程序如下：

|  |
| --- |
| Program travel(intput,output);  Var  Total,I,m:longint;  S1,s2:string;  Begin  Assign(input,’travel.in’);  Assign(output,’travel.out’);  Reset(input); rewrite(output);  Readln(s1); readln(s2); total:=1;  For i:=1 to length(s1)-1 do  Begin  M:=pos(s1[i],s2);  If m>1 then if s[i+1]=s[m-1] then total:=total\*2;  End;  Writeln(total); close(iinput); close(output);  End. |

**2. 售货员的难题**

**【算法分析】**

题目给定的村庄数不多(≤40)，所以可以用回溯的方法，从起点(第一个村庄)出发找出所有经过其他所有村庄的回路，计算其中的最短路程。当村庄数n比较大时这种方法就不太适用了。

用一个过程road(step，line：byte)来描述走的状况，其中step是当前已到村庄数、line是当前所在的村庄。如果step＝n，下面只能回起点了，直接看第line个村庄到第一个村庄的路程加上已走的总路程，如果比最小值还小则替换最小值(要保存路径的话也可保存，这是回溯算法的优点，考虑到达最小值的路径可能不止一条，不便于测试，题目没要求输出路径)。如果step还小于n，那么将还没有到过的村庄一个一个地试过去，再调用下一步road(step+1，新到的村庄号)。

**3. 驾车旅游**

**【算法分析】**

驾车者从出发地出发后对于每个加油站都可能有两种操作，一是进去加油买食品，二是不进去继续前行(如果当前汽车的余油可以的话)，这样有n个加油站最多可能有2n种选择。由于加油站数目不太多，可以采用回溯的算法来解决问题。从第一个加油站开始，依次选择所要停下的下一个加油站，从而找出总费用最少的方案，加油站数目最多为50，这样回溯不会进行得很深。在选择下一个要停下的加油站时比较麻烦，不能完全一个一个地试过去，这样时间太长。可以用这样的方法：先找出第一个要停下的加油站，判断其后面的加油站是否可以到达，如果不可到达就必须在这里停下来加油；否则就找出可以到达但如果只用一半汽油则无法到达的所有加油站，依次进行停靠。

**4. 关路灯**

**【算法分析】**

设老张开始所在位置为c，以起始点c为分界点，算出左右两部分总的功率p\_left和p\_right，再来分别看向左与向右的情况。

向左走时，相应地可以减小左边应费的功，而增加右边应费的功，如果到一个点(一盏路灯处)所要时间为t，减少的功为(p\_left+w[i])\*t，增加的功为p\_right\*2t。

向右走时，相应地可以减小右边应费的功，而增加左边应费的功，如果到一个点(一盏路灯处)所要时间为t，减少的功为(p\_righ+w[i])\*t，增加的功为p\_left\*2t。

比较向左与向右的情况，找出比较好的一种确定方法。大部分情况能够解出最小值，但不能求出所有情况下最佳的解。

对于每一个所处位置，都可以选择向左或向右，不管是向左还是向右，相应耗电的变化都跟上面所述一样。所以可以选择回溯的算法来实现有限的搜索，对每一个点试探向左与向右的情况，在所有可能的情况中找出最优解。

**【思考与提高】**

上面的程序运算的范围很有限，当n比较大时就会栈溢出，如n>30时速度就比较慢了。实际情况调头的次数并不会多的，到底在什么时候掉头根据情况而定。我们可以从最后一步来思考：

最后一次关的可能是第一个灯也可能是最后一个灯，哪种情况所费的功小就选哪种；

最后一次关的是第一个灯的话，说明最后的方向是从最后到最前(右边到左边)，最后倒数第二次的方向为从左到右，起点可能是原始起点(此时是第一趟)，也可能是原始起点左边的点(此时至少是第二趟)，一个个地试过去，先设拐一次弯，有可能拐的点都试过去，再试有两次拐弯换方向的情况，当再多的拐弯超过已有的解时就不要再向下试了。采用这种回溯方法，效率更高。

如果n再大一些，如到300以上，上述方法也有它的局限性，此时最好从动态规划法的角度去思考。