第 8 次上机解题报告

15081070 张雨任

一、A 题

1. 题目代号与评测记录序号：{A284744}
2. 解题思路：

   此题考查通过广义表的输入建立二叉树以及二叉树的递归。

   首先说一下广义表的输入，先把输入的内容放在一个 string 里，方便分离不是字符而是字符串的元素。接下来遍历字符串。如果是左括号，则说明要读取左子树了，那么把 flag 置为 1，把先前的根节点放到栈中。如果是逗号，说明接下来读取的将会是右子树，那么要把 flag 置为 2。如果是字母，则要一直及继续遍历字符串，直到读取到别的符号。并把这些字母依次放在当前新建的结点的 data 中，通过 string 类的+=操作即可实现，当然结点的 data 域的类型也是 string，这样就可以建立 data 域为 string 字符串类型的的树了。

   接下来递归比较树的两个子树就比较简单了。

   规则 1：如果左右子树都存在，若节点内容都相同，则继续递归比较左子树的左孩子和右子树的右孩子以及左子树的右孩子和右子树的左孩子，不相同返回 false。如果左右子树有一不存在，则为错误。

   规则 2：和规则 1 类似，不需要镜面比较了，直接比较左右子树的两个左结点，再比较左右子树的两个右节点即可，然后并不需要比较结点的内容，只要结构相似即可。

   此题麻烦在输入 string 那里，需要处理字符串并放在 data 中，其次卡时间了，所以用的 cstdio 来输入输出。其他不难。

二、B 题

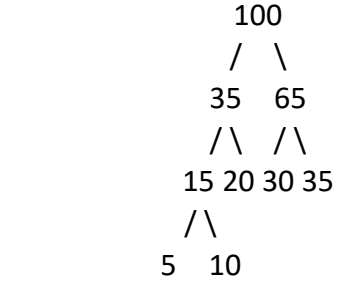
1. 题目代号与评测记录序号：{B285843}
2. 解题思路：

   这道题要求一段字符经过霍夫曼编码之后的码长。

   首先进行预处理，由于这里要读入空格和换行符，所以使用 getchar 来吸收字符比较好一些。然后还要统计每种字符出现的次数，因此设置两个数组，一个 char 类型的 chrArr 数组，来存放各种字符，一个 int 类型的 cnt 数组，来统计每种字符对应的出现频数是多少。这里字符 chrArr[i]的频数即为 cnt[i]。

   每读入一个字符，就要遍历 chrArr 数组：如果出现过就在对应的 cnt 数组的元素自增；没出现过就把字符添加到 chrArr 数组的末尾，并让对应的 cnt 数组的元素自增。这样就得到了一个存放着出现过的字符的 char 类型数组和一个存放各种字符出现频数的 int 类型数组。

   接下来对 cnt 数组进行霍夫曼编码得到结果。这里其实并不需要建立二叉树模拟整个编码过程，而是使用到了第四次上机 B 题“合并果子”的思想。计算总编码长度就是对于霍夫曼树的每个叶子节点所带的权重（也就是代表实际编码的符号的频数）乘以各自的深度，再求和。这个值等于这棵树的所有内结点的权重之和。用书上的例子来说，权重集合为{5,10,20,30,35}，依照霍夫曼算法可生成形如下图的树：



按照霍夫曼编码定义来计算，码长=5*3+10*3+20*2+30*2+35*2=215.

根据前文推论，码长=15+35+65+100=215.

“合并果子”的过程就是：取最小的两个权重作为左右子树相加，置新的根节点的权重为其左右子树上根节点的权重之和，累计这个和……以此往复，直到只剩下一个根节点（也就是一个果子），也就是说，合并果子和建立霍夫曼树的过程是相同的，在这个“合并果子”的过程中，5 累加了 3 次，10 累加了 3 次，20 累加了 2 次，30 累加了 2 次，35 累加了 2 次，通过这样的方法累积的和即为 15+35+65+100=215。因此可以使用这种方法。

三、C 题

1. 题目代号与评测记录序号：{C285410}
2. 解题思路：

   此题可以画出一个树一样的图，但是并不需要用树来解决，这里要求分组的组数，有上下级关系的人不能在同一组，同级的人才能在一组，那么求这棵树的深度就可以了。设立一个数组来存放某下标对应的人的上司是谁，比如：第 i 个人的上司是 j，可以表示为 people[i-1]=j，由于数组是从 0 开始的，所以是 i-1。这样，遍历数组的每一个元素，追溯它的上司，level++，再追溯上司的上司，level++，再追溯上司的上司的上司……直到某人的值为-1，说明他是 boss，没有上司，这样就可以求出某人的 level，所在的层次，那么所有层次的最大值就是所求的组数。坑点在于上司可能比下属出现得晚，所以不能一边输入一边遍历。

四、D 题

1. 题目代号与测评记录序号：{D284898}
2. 解题思路：

   此题通过公式求解。对于具有 n 个结点的二叉树，设具有互不相似的二叉搜索树的个数为 ans(n)。一棵具有 n（n>1）个结点的二叉树可以看成是有一个根节点、一个拥有 i 个结点的左子树、一个拥有 n-i-1 个结点的右子树组成，期中 0<=i<n。所以可以有递推公式：

$$ans(0) = 1 \text{ 空树}$$

$$ans(1) = 1 \text{ 仅有一个根节点的时候}$$

$$ans(n) = \sum_{i=0}^{n-1} ans(i)ans(n-i-1), \text{ n>1}$$

   定义生成函数：

$$A(x)=ans(0)+ans(1)x+ans(2)x^2+\ldots+ans(n)x^n+\ldots=\sum_{j=0}^{\text{正无穷}} ans(j)x^j$$

又

$$A^2(x)=ans(0)*ans(0)+(ans(0)*ans(1)+ans(1)*ans(0))*x+\ldots$$

$$=\sum_{j=0}^{正无穷}(\sum_{i=0}^{p} ans(i)ans(p-i))x\text{^}p$$

根据递推公式，有：

$$(A^2)(x)=\sum_{j=0}^{正无穷} ans(j+1)x\text{^}j$$

所以，有：

$xA^2(x)=A(x)-1$，解得$A(x)=\frac{1-\sqrt{1-4x}}{2x}$

展开可得

$$A(x)=1+x+2x^2+5x^3+14z^4+42z^5+\ldots$$

由生成函数和上述展开式可得：

$$ans(n)=[0.5*(0.5-1)*(0.5-2)*\ldots*(0.5-n)]/(n+1)!*[(-1)\text{^}n]*[2\text{^}(2*n+1)]$$

所以

$$ans(n)=(2*n)!/(n!*n!)/(n+1)=C(2*n,n)/(n+1)$$

然后套用公式即可。公式详解在严蔚敏教授的书上第 152 页。这个重要结论需要记住。

五、F 题

1. 题目代号与测评记录序号：{F286116}
2. 解题思路：

这道题的坑点在于输入的两个城市不清楚哪个是父结点哪个是子结点。所以要各自记录自己都和谁相连，以此来弱化父结点和子结点的概念。再者，从叶子节点从下往上遍历时间复杂度达到 n^2 级别，会超时，所以这里运用了栈的方法，用栈来记忆之前遍历过的结点，就不需要从头遍历了。

首先说一说结点的结构，一个 int 类型的 goodOrBad 来记录这个城市是好是坏，一个 vector 类型、元素为 int 的 next 向量来记录与之相连的结点的编号都是谁，还需要一个 int 类型的 parent 来记录该结点的父结点的编号是多少，因为 next 向量中既有父结点又有子结点，通过这个变量就可以辨认哪些是子结点哪些是父结点，还有一个 int 类型的 lengthOfPre 来记录遍历到其父节点为止累积的连续的不好城市的个数是多少。

接下来进行初始化，声明结构体数组和栈，输入各个城市的好坏值，构造每个节点的 next 向量，这里可以体现 vector 的方便，可以自由地向向量后面添加新的成员，因此使用 vector 较为方便。初始化栈，把 1（根节点，家所在的城市编号）放到栈中。在栈不空的情况下，弹出栈顶元素，是一个城市的编号，如果好坏值为 1，则当前累计连续坏城市数量（变量名为 now）为此城市的父结点城市的累计连续坏城市数量加一，如果好坏值为 0，则清零 now 的值，代表连续坏城市的情况被打破，重新累计连续坏城市的数量。接下来

考察 now 值，如果 now 大于 m（也就是连续坏城市的忍耐限度），说明该结点已经没有继续遍历的意义了，它的子结点也不需要再遍历了，继续弹出下一个结点考察它就可以了。此时考虑如果便利到了叶子结点累积的 now 值还没有超出 m（连续坏城市的忍耐限度），说明这个景点（叶子结点）符合题意，累计 ans。如果还没有到达叶子结点，就要考察该结点的子女结点。这里设立 int 类型变量 cntNext 用来遍历 next 向量的每一项。遍历 next 向量，也就是遍历该结点的每一个子女结点（要保证父结点的 next 向量的元素不等于 parent，因为 next 向量会包含一个父结点）。对于遍历到的子女结点，令其 lengthOfPre 等于 now（当前长度也就是该子女结点的父结点的长度），再令其 parent（父结点的编号）等于这个循环内弹出的结点的编号，方便下次分辨父结点和子女结点，最后把该结点放进栈中，总体来讲就是把这次循环弹出的父结点的所有子女结点都放进栈中，并记录相应的数据。最后，当栈为空的时候，说明所有可能产生可选择的景点的结点都已遍历完成（某些结点其实并没有遍历到，因为他们的某个祖先结点就已经超过了 m 值，所以更不必去遍历他们了），输出 ans 答案即可。

3. 易错点：
   首先不可暴力遍历，会超时，要用栈来记忆每个结点。
   其次，输入两个节点的时候并不知道哪个是父结点哪个是子结点，所以只能记录某结点都和哪些结点相连，遍历的时候再进行分辨。