我保证没有抄袭别人作业

1. 已知某电文中共出现了 10 种不同的字母（A, B, C, D, E, F, G, H, I, J），每种字母出现的次数分别为 6、8、5、3、7、22、10、9、1、40，现在对这段电文用三进制进行非定长前缀编码（码字由 0、1、2 组成），请问 Huffman 电文编码总长度至少有多少位？相应于等长编码，只需要多少空间？请画出相应编码方案的图示（不需要画中间过程，只需要最终的结果）。

解: 最少的 Huffman 编码需要 201 位

由于等长编码需要 333 位,所以节省了 132 位的空间

[111]										
40	[24]			47						
	7	8	9	22	10	[15]				
						6	5	[4]		
								1	3	

2. 证明：判断以下三种情况是否成立，并给出证明，对于不成立的，需要给出反例：

1）已知先序遍历序列和中序遍历序列可以确定唯一一棵二叉树。

2）已知中序遍历序列和后序遍历序列可以确定唯一一棵二叉树。

3） 已知先序遍历序列和后序遍历序列可以确定唯一一棵二叉树。

证明:

1) 成立

(归纳法)对序列的长度进行归纳,当长度为 1 时,可以唯一确定一棵二叉树

先假设对任何的 k < N,都可以唯一确定,则当长度为 N 时,由先序序列的第一个元素可以确定二叉树的根节点,在中序序列中找到该根节点,则其在中序序列中它左面的序列对应它的左子树,且左面的序列长度 l < N,且在前序序列中前 1 到 l+1 个字符(从零开始计算下标)确定的是左子树序列的前序序列,右面的序列同理,都可以唯一确定该根节点的子树,所以是可以唯一确定二叉树的.

2) 成立

类似 1)的证明,利用归纳法,假设对任何的 k < N,都可以唯一确定,则当长度为 N 时,此时后序序列的最后一个元素是根节点,在中序序列中找到它,就可以确定它的左子树和右子树,并且可以找到左子树和右子树的中序遍历序列和后序遍历序列,由归纳法,这时也是可以唯一确定的.

3) 不成立

反例: A 的左子节点为 B,B 的左子节点为 C 和右子节点为 C 时两种情况的前序遍历和后序遍历的序列相同

3. 在一棵表示有序集 S 的二叉搜索树中，任意一条从根到叶子结点的路径将 S 分为 3 个部分：在该路径左边结点中的元素组成的集合 S1；在该路径上的结点中的元素组成的集合 S2；在该路径右边结点中的元素组成的集合 S3。$S = S1 \cup S2 \cup S3$。若对于任意的 $a \in S1$，$b \in S2$，$c \in S3$ 是否总有 $a \le b \le c$？为什么？

解：不是

<table>
<tr><td colspan="8">8</td></tr>
<tr><td colspan="4">4</td><td colspan="4">12</td></tr>
<tr><td colspan="2">2</td><td colspan="2">6</td><td colspan="2">10</td><td colspan="2">14</td></tr>
<tr><td>1</td><td>3</td><td>5</td><td>7</td><td>9</td><td>11</td><td>13</td><td>15</td></tr>
</table>

如图的搜索树中 5 在路径的左面,但路径上的 4 比它小

4. 下面的算法是使用非递归的方法中序遍历二叉树。请填充算法中的空格，使其成为完整的算法。

```
enum Tags{Left, Right};
template <class T>
class StackElement                    //栈元素定义
{
public:
    BinaryTreeNode<T> *pointer;
    Tags tag;
};

template <class T>
void BinaryTree<T>::InOrderWithoutRecursion(BinaryTreeNode<T> *root)
{
    using std::stack;
    StackElement<T> element;
    stack<StackElement<T> > aStack;
    BinaryTreeNode<T> *pointer = root;
    do
    {
        while(pointer != NULL)
        {
            element.pointer = pointer;
            element.tag = Left;
            aStack.push(element);
            pointer = pointer->leftchild();
        }
        while ( aStack.top().tag == Right )        aStack.pop();
        if (!aStack.empty() && aStack.top().tag == Left)
        {
```

```
            Visit(aStack.top().pointer);
            pointer = aStack.top().pointer->rightchild();
            aStack.top().tag = Right;
        }
    }
    while(!aStack.empty());
}
```

5. 下面的算法是从根节点开始广度优先遍历二叉树。若二叉树有 n 个结点，设遍历过程中的队列的最大长度是 f(t)，t 可以是任意一棵二叉树。当 t 为完全二叉树时，f(t)取得最大值，这种说法对吗？如果正确，请给出证明；如果错误，请举出反例。（完全二叉树的定义：只有最后一层的节点是可能为空的，且最后一层中左边是满的）

```
void BinaryTree<T>::LevelOrder(BinaryTreeNode<T>* root){
     using std::queue;                                   // 使用 STL 的队列
     queue<BinaryTreeNode<T>*> aQueue;
     BinaryTreeNode<T>* pointer = root;                  // 保存输入参数
     if (pointer) aQueue.push(pointer);                  // 根结点入队列
     while (!aQueue.empty())    {                        // 队列非空
    pointer = aQueue.pop();                              // 当前结点出队列
            Visit(pointer->value());                     // 访问当前结点
    if(pointer->leftchild())
            aQueue.push(pointer->leftchild());           // 左子树进队列
    if(pointer->rightchild())
            aQueue.push(pointer->rightchild());          // 右子树进队列
     }
}
```

解:

正确

完全二叉树的广度优先遍历,由于完全二叉树的性质,可知,一定先遍历到度为 2 的结点,之后遍历到 1 或 0 个度为一的结点,最后遍历度为零的结点,所以队列的长度 l(i)看做关于结点序号的函数的话,图像会是一个不完整的底角 45°的等腰梯形(因为初值为 1,最后取 0),且最大值为[(n + 1) / 2](向下取整),因为每遍历到一个度为 2 的结点,l(i) += 1,类似的,度为一 l(i)不变,度为零 l(i) -= 1;

又对任何 l(i),其中 i 定义在 0~n-1 之间的函数,且满足|l(i + 1) - l(i)| = 1,最开始取 1(根节点入列)和最后函数值都为零的函数,最多有[(n - 1) / 2]个结点会增加函数值,所以最大值会小于[(n - 1) / 2] + 1 = [(n + 1) / 2](向下取整)

所以完全二叉树确实是可以取到最大值的一种情况,

6. 初始关键码序列为{49, 38, 27, 49, 76, 13, 65, 97}，试给出用筛选法所建立的最大堆，并写出其相应的序列。在建堆过程中，移位次数是多少？

解:

```
            97
          /    \
        76      65
       /  \    /  \
     49   49 13    27
    /
  36
```

97 76 65 49 49 13 27 38

6 次