山东大学 计算机科学与技术 学院

数据结构与算法 课程实验报告

| 学　　号　　：202200400053 | 姓名： 王宇涵 | 班级： 22 级 2 班 |
|---|---|---|
| 实验题目：堆及其应用 | | |
| 实验学时：2 | 实验日期： 2023-11-15 | |

实验目的：

掌握堆排序及其应用

软件开发环境：

Vscode

1. 实验内容

题目一:堆的操作

创建**最小堆类**。最小堆的存储结构使用**数组**。提供操作:插入、删除、初始化。题目第一个操作是建堆操作，接下来是对堆的插入和删除操作，插入和删除都在建好的堆上操作。

题目二:霍夫曼编码

**输入**一串小写字母组成的字符串（不超过 1000000), 输出这个字符串通过 Huffman 编码后的长度。

2. 数据结构与算法描述 （整体思路描述，所需要的数据结构与算法）

题目一:

**概述**

MaxPriorityQueue 类：是一个抽象类，定义了优先队列的基本操作接口，包括 empty()（判断队列是否为空）、size()（获取队列大小）、top()（获取队列顶部元素）、pop()（弹出队列顶部元素）、push()（插入元素）等纯虚函数。

MinHeap 类：实现了最小堆，继承自 MaxPriorityQueue 类。最小堆满足根节点小于其子树中每个节点的值。类中包括了 push()（插入元素）、pop()（删除根节点）、initialize()（初始化堆）、output()

（输出堆中元素）等方法。

MaxHeap 类：实现了最大堆，同样继承自 MaxPriorityQueue 类。最大堆满足根节点大于其子树中每个节点的值。类似于 MinHeap 类，包含了 push()、pop()、initialize()、output() 等方法。

heapSort 函数：利用最大堆进行堆排序，首先将数组初始化为最大堆，然后逐步将堆顶元素（最大值）取出，并放置到数组的末尾，最终实现升序排序。

main 函数：从用户输入中构建一个最小堆，随后根据用户输入的操作进行相应的最小堆操作，包括插入元素、删除堆顶元素、对部分数组进行堆排序等。

**动态数组**

由于需要通过 delete 释放 heap,所以初始化的时候传入的 arr 数组一定要是动态数组,而不能是 int a[]创建的静态数组,须通过 new 来创建.

**HeapSort**

由于初始状态下 arr 已经赋值给了 heap,因此无法通过小根堆进行排序,因为每次改变 a[i]的值同时也会改变 heap 中的值!必须通过大根堆进行排序,按从后往前的顺序进行赋值.

题目二:

**概述**

LinkedBinaryTree 类：实现了二叉树的各种操作。其中 createTreeByLevel() 方法用于根据输入的数据创建一个二叉树，根据权重构建了一个哈夫曼树。preOrder()、inOrder()、postOrder() 和 leverOrder() 方法分别对二叉树进行前序、中序、后序和层次遍历。makeTree() 方法用于构建一棵新的树。还有一些其他的辅助方法，比如 empty()、size()、erase()、height() 等。

MinHeap 类：实现了最小堆的功能，用于构建哈夫曼树。包括了堆的基本操作，如插入元素 push()、删除顶部元素 pop()、获取顶部元素 top()、判断堆是否为空 empty()、获取堆大小 size() 等。

createHuffmanTree 函数：用于根据给定的权重数组构建哈夫曼树。利用最小堆来实现哈夫曼树的构建，首先将权重数组中的元素构建成哈夫曼节点，并加入最小堆中，然后根据权重构建哈夫曼树。

main 函数：从输入中读取字符串，统计字符出现的频率，根据字符频率构建哈夫曼树,并输出编码后的长度.

**求长度思路**

思路 1:在构造的过程中直接求出

每两次取出 heap.top()后将 weight 进行相加,并更新 ans=sum(weight)+ans.

思路 2:对最后的求出的树进行递归遍历

函数 addLength(结点,高度)进行递归遍历,每找到一个叶子结点,就更新 ans=leat.weight*height+ans.

3. 测试结果（测试输入，测试输出）

题目一:

**输入**

10

-225580 113195 -257251 384948 -83524 331745 179545 293165 125998 376875

10

1 -232502

1 -359833

1 95123

2

2

2

1 223971

1 -118735

1 -278843

3 10

-96567 37188 -142422 166589 -169599 245575 -369710 423015 -243107 -108789

**输出**

-257251

-257251

-359833

-359833

-257251

-232502

-225580

-225580

-225580

-278843

-369710 -243107 -169599 -142422 -108789 -96567 37188 166589 245575 423015

题目二

**输入**

abcdabcaba

**输出**

19

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

过程中出现一些问题,最后通过查询资料克服了挑战

**问题一:主函数没有正常退出**

解决:经过调试发现最后卡在 minheap 的析构函数上,通过查询资料发现不能 delete 静态数组的指针,因此需要将 initialize 函数传入的数组改为动态数组,成功通过实验.

**问题二:利用小根堆进行堆排序结果出问题**

解决:经过调试发现从小到大赋值的时候,会改变 heap.top()的 weight,因此需要从大到小赋值,因此需要大根堆进行排序.

**问题三:实验二的代码总是最后一个检验点超时**

解决:要求 20ms 以内,最高运行时间是 23ms,因此差别很小,改变了算法也没有见到提升,因此我们改变思路,将输入的 cin 改成 scanf,最后成功将时间缩小到 7ms,成功 AC

## 5. 附录：实现源代码（本实验的全部源程序代码，程序风格清晰易理解，有充分的注释）

### 题目一

```
#include<iostream>

using namespace std;


template<class T>

class MaxPriorityQueue

{

    public:

        virtual ~MaxPriorityQueue(){}

        virtual bool empty()const =0;

        virtual int size()const=0;

        virtual const T& top()=0;

        virtual void pop()=0;

        virtual void push(const T& theElement)=0;

};



template<class T>

class MinHeap:public MaxPriorityQueue<T>

{

public:

    MinHeap(int arraySize)

    {
```

```
        arrayLength=arraySize+1;
        heap=new T[arrayLength];
        heapSize=0;
    }
    ~MinHeap()
    {
        if(heap!=NULL)
            delete[]heap;
    }

    void push(const T& theElement);
    void pop();
    const T& top()
    {
        return heap[1];
    }
    int size()const
    {
        return heapSize;
    }
    bool empty()const
    {
        return heapSize==0;
    }
    void initialize(T* theHeap,int heapSize);
    void output(ostream& out)const
    {
        for(int i=1;i<=heapSize;i++)
            out<<heap[i]<<" ";
    }
    void deactiveArray()
    {
```

```
        heap=NULL;arrayLength=heapSize=0;
    }

private:
    T *heap;
    int arrayLength;
    int heapSize;
};

template<class T>
void doubleArraySize(T*& heap,int arrayLength)
{
    T* newArray=new T[arrayLength*2];
    copy(heap,heap+arrayLength,newArray);
    delete []heap;
    heap=newArray;
}

//时间复杂度,高度 log2n
template <class T>
inline void MinHeap<T>::push(const T &theElement)
{
    //数组满,扩大数组
    if(heapSize==arrayLength-1)
    {
        doubleArraySize(heap,arrayLength);
        arrayLength*=2;
    }

    //父节点下沉
    int currentNode=++heapSize;
    while(currentNode!=1&&heap[currentNode/2]>theElement)
```

```
        {
            heap[currentNode]=heap[currentNode/2];
            currentNode/=2;
        }

        heap[currentNode]=theElement;
}

//时间复杂度 log2n,为高度 h
template <class T>
inline void MinHeap<T>::pop()
{
        if(heapSize==0)
        {
            cout<<"heap empty"<<endl;
            return;
        }

        heap[1].~T();
        T lastElement=heap[heapSize--];

        int currentNode=1,child=2;

        while(child<=heapSize)
        {
            if(child<heapSize&&heap[child+1]<heap[child])
                child++;

            if(lastElement<=heap[child])
                break;
            heap[currentNode]=heap[child];
            currentNode=child;
```

```
            child*=2;
        }
        heap[currentNode]=lastElement;
}

//大根堆的初始化,从有孩子的结点开始依此操作
template<class T>
inline void MinHeap<T>::initialize(T* theHeap,int theHeapSize)
{
        delete[]heap;
        heap=theHeap;
        heapSize=theHeapSize;
        for(int root=heapSize/2;root>=1;root--)
        {
            T element=heap[root];
            int cur=root;
            int child=cur*2;

            while(child<=heapSize)
            {
                if(child<heapSize&&heap[child+1]<heap[child])
                    child++;

                if(element<=heap[child])
                    break;

                //孩子上移
                heap[cur]=heap[child];
                cur=child;
                child*=2;
            }
```

```
            heap[cur]=element;
        }
    }

    template<class T>
    ostream & operator<<(ostream& out,const MinHeap<T> &x)
    {
        x.output(out);return out;
    }

    template<class T>
    class MaxHeap:public MaxPriorityQueue<T>
    {
    public:
        MaxHeap(int arraySize)
        {
            arrayLength=arraySize+1;
            heap=new T[arrayLength];
            heapSize=0;
        }

        ~MaxHeap()
        {
            if(heap!=NULL)
                delete[]heap;
        }
        void push(const T& theElement);
        void pop();
        const T& top()
        {
            return heap[1];
        }
```

```
        int size()const
        {
            return heapSize;
        }
        bool empty()const
        {
            return heapSize==0;
        }
        void initialize(T* theHeap,int heapSize);
        void output(ostream& out)const
        {
            for(int i=1;i<=heapSize;i++)
                out<<heap[i]<<" ";
        }
        void deactiveArray()
        {
            heap=NULL;arrayLength=heapSize=0;
        }

private:
        T *heap;
        int arrayLength;
        int heapSize;
};



//时间复杂度,高度 log2n
template <class T>
inline void MaxHeap<T>::push(const T &theElement)
{
        //数组满,扩大数组
        if(heapSize==arrayLength-1)
```

```
        {
            doubleArraySize(heap,arrayLength);
            arrayLength*=2;
        }

        //父节点下沉
        int currentNode=++heapSize;
        while(currentNode!=1&&heap[currentNode/2]<theElement)
        {
            heap[currentNode]=heap[currentNode/2];
            currentNode/=2;
        }

        heap[currentNode]=theElement;
}

//时间复杂度 log2n,为高度 h
template <class T>
inline void MaxHeap<T>::pop()
{
        if(heapSize==0)
        {
            cout<<"heap empty"<<endl;
            return;
        }

        heap[1].~T();
        T lastElement=heap[heapSize--];

        int currentNode=1,child=2;

        while(child<=heapSize)
```

```
        {
            if(child<heapSize&&heap[child+1]>heap[child])
                child++;

            if(lastElement>=heap[child])
                break;

            heap[currentNode]=heap[child];
            currentNode=child;
            child*=2;
        }
        heap[currentNode]=lastElement;
}

//大根堆的初始化,从有孩子的结点开始依此操作
template<class T>
inline void MaxHeap<T>::initialize(T* theHeap,int theHeapSize)
{
        delete[]heap;
        heap=theHeap;
        heapSize=theHeapSize;
        for(int root=heapSize/2;root>=1;root--)
        {
            T element=heap[root];
            int cur=root;
            int child=cur*2;

            while(child<=heapSize)
            {
                if(child<heapSize&&heap[child+1]>heap[child])
                    child++;
```

```
            if(element>=heap[child])
                break;

            //孩子上移
            heap[cur]=heap[child];
            cur=child;
            child*=2;
        }

        heap[cur]=element;
    }
}

template<class T>
ostream & operator<<(ostream& out,const MaxHeap<T> &x)
{
    x.output(out);return out;
}



template<class T>
void heapSort(T* a,int n)
{
    MaxHeap<T>heap(1);
    heap.initialize(a,n);
    for(int i=n;i>=1;i--)
    {
        T x=heap.top();
        heap.pop();
        a[i]=x;
```

```
        }


        heap.deactiveArray();
}


int main()
{
        int n,m;
        cin>>n;
        MinHeap<int>heap(1);
        int *arr=new int [n+1];
        for(int i=1;i<=n;i++)
                cin>>arr[i];
        heap.initialize(arr,n);
        // for(int i=0;i<n;i++)
        // {
        //          int num;cin>>num;
        //          heap.push(num);
        // }
        cout<<heap.top()<<endl;
        cin>>m;
        while(m--)
        {
                int op;
                cin>>op;
                switch (op)
                {
                        case 1:
                        {
                                int num;
                                cin>>num;
                                heap.push(num);
```

```
                cout<<heap.top()<<endl;
                break;
            }
            case 2:
            {
                heap.pop();
                cout<<heap.top()<<endl;
                break;
            }
            case 3:
            {
                int count;
                cin>>count;
                int arr[count+1];
                for(int i=1;i<=count;i++)
                    cin>>arr[i];
                heapSort(arr,count);
                for(int i=1;i<=count;i++)
                    cout<<arr[i]<<" ";
                cout<<endl;
                break;
            }
            default:
                break;
        }
    }
    return 0;
}
```

## 题目二

```
#pragma once
#include<iostream>
using namespace std;
```

```
#pragma once
#include<iostream>
#include<map>
using namespace std;
template<class T>
struct BinaryTreeNode
{
	T element;
	BinaryTreeNode<T>* leftChild;
	BinaryTreeNode<T>* rightChild;
	BinaryTreeNode()
	{
		leftChild=rightChild=NULL;
	}
	BinaryTreeNode(const T& theElement)
	{
		element=theElement;
		leftChild=rightChild=NULL;
	}
	BinaryTreeNode(const T& theElement,BinaryTreeNode * theLeftChild,BinaryTreeNode* theRightChild)
	{
		element=theElement;
		leftChild=theLeftChild;
		rightChild=theRightChild;
	}
};
#pragma once
#include<iostream>
using namespace std;

template<class T>
class BinaryTree
```

```
{
    public:
        virtual ~BinaryTree(){}
        virtual bool empty()const =0;
        virtual int size()const =0;
        virtual void preOrder(void (*) (T*))=0;
        virtual void inOrder(void(*)(T*))=0;
        virtual void postOrder(void(*)(T*))=0;
        virtual void leverOrder(void(*)(T*))=0;
};
#include"queue"

template<class E>
class LinkedBinaryTree:public BinaryTree<BinaryTreeNode<E>>
{
    public:
        LinkedBinaryTree()
        {
            root=NULL;
            treeSize=0;
            visit=output;
        }
        void makeTree(const E& theElement,LinkedBinaryTree<E>&a,LinkedBinaryTree<E>&b);
        ~LinkedBinaryTree()
        {
            erase();
        }
        void createTreeByLevel();
        bool empty() const
        {
            return treeSize==0;
        }
```

```
int size()const
{
    return treeSize;
}
//遍历函数,传入 visit 函数指针
void preOrder(void (*theVisit) (BinaryTreeNode<E>*))
{
    visit=theVisit;preOrder(root);
}
void inOrder(void (*theVisit) (BinaryTreeNode<E>*))
{
    visit=theVisit;inOrder(root);
}
void postOrder(void (*theVisit) (BinaryTreeNode<E>*))
{
    visit=theVisit;postOrder(root);
}
void leverOrder(void (*theVisit) (BinaryTreeNode<E>*))
{
    visit=theVisit;leverOrder(root);
}
void preOrderOutput()
{
    preOrder(output);
    cout<<endl;
}
void inOrderOutput()
{
    inOrder(output);
    cout<<endl;
}
void postOrderOutput()
```

```
        {
            postOrder(output);
            cout<<endl;
        }
        void leverOrderOutput()
        {
            leverOrder(output);
            cout<<endl;
        }
        void erase()
        {
            postOrder(dispose);
            root=NULL;
            treeSize=0;
        }
        int height() const
        {
            return height(root);
        }
        int height(BinaryTreeNode<E>*t)
        {
            if(t==NULL)
                return 0;
            int hl=height(t->leftChild);
            int hr=height(t->rightChild);
            if(hl>hr)
                return ++hl;
            else
                return ++hr;
        }

    public:
```

```
        BinaryTreeNode<E>* root;
        int treeSize;
        int index=1;
        int ans=0;
        //定义函数指针,规定如何访问元素
        static void (*visit)(BinaryTreeNode<E>*);
        static void preOrder(BinaryTreeNode<E>*t);
        static void inOrder(BinaryTreeNode<E>*t);
        static void postOrder(BinaryTreeNode<E>*t);
        static void leverOrder(BinaryTreeNode<E>*t);
        static void output(BinaryTreeNode<E>*t)
        {
            cout<<t->element<<" ";
        }
        static void dispose(BinaryTreeNode<E>*t)
        {
            delete t;
        }


};


template<class E>
void (*LinkedBinaryTree<E>::visit)(BinaryTreeNode<E>*);


template <class E>
inline void LinkedBinaryTree<E>::createTreeByLevel()
{
    int n;cin>>n;
    int a[n];
    for(int i=0;i<n;i++)
```

```
            cin>>a[i];


        queue<BinaryTreeNode<E>*>q;


        int index=0;
        root=new BinaryTreeNode<E> (a[index++]);
        q.push(root);
        BinaryTreeNode<E>* p=NULL;
        while(!q.empty()&&index<n)
        {
            p=q.front();
            q.pop();
            //如果不空就创建一个节点
            BinaryTreeNode<E>* leftNode=new BinaryTreeNode<E>(a[index++]);
            p->leftChild=leftNode;
            q.push(leftNode);
            treeSize++;


            if(index<n)
            {
                BinaryTreeNode<E>* rightNode=new BinaryTreeNode<E>(a[index++]);
                p->rightChild=rightNode;
                q.push(rightNode);
                treeSize++;
            }
        }
}


template <class E>
inline void LinkedBinaryTree<E>::preOrder(BinaryTreeNode<E> *t)
{
        if(t!=NULL)
```

```
        {
            LinkedBinaryTree::visit(t);
            preOrder(t->leftChild);
            preOrder(t->rightChild);
        }
}

template <class E>
inline void LinkedBinaryTree<E>::inOrder(BinaryTreeNode<E>*t)
{
        if(t!=NULL)
        {
            inOrder(t->leftChild);
            LinkedBinaryTree<E>::visit(t);
            inOrder(t->rightChild);
        }
}

template <class E>
inline void LinkedBinaryTree<E>::postOrder(BinaryTreeNode<E>*t)
{
        if(t!=NULL)
        {
            postOrder(t->leftChild);
            postOrder(t->rightChild);
            LinkedBinaryTree<E>::visit(t);
        }
}

template <class E>
inline void LinkedBinaryTree<E>::leverOrder(BinaryTreeNode<E>*t)
{
```

```
        queue<BinaryTreeNode<E>*> q;
        while(t!=NULL)
        {
                visit(t);

                if(t->leftChild!=NULL)
                        q.push(t->leftChild);
                if(t->rightChild!=NULL)
                        q.push(t->rightChild);

                if(!q.empty())
                {
                        t=q.front();
                        q.pop();
                }
                else
                        return;
        }
}

template <class E>
inline void LinkedBinaryTree<E>::makeTree(const E &theElement, LinkedBinaryTree<E> &a, LinkedBinaryTree<E> &b)
{
        root=new BinaryTreeNode<E> (theElement,a.root,b.root);
        treeSize=a.treeSize+b.treeSize+1;

        a.root=b.root=NULL;
        a.treeSize=b.treeSize=0;
}



#pragma once
```

```
#pragma once
#include<iostream>
using namespace std;

template<class T>
class MaxPriorityQueue
{
    public:
        virtual ~MaxPriorityQueue(){}
        virtual bool empty()const =0;
        virtual int size()const=0;
        virtual const T& top()=0;
        virtual void pop()=0;
        virtual void push(const T& theElement)=0;
};

template<class T>
class MinHeap:public MaxPriorityQueue<T>
{
public:
    MinHeap(int arraySize)
    {
        arrayLength=arraySize;
        heap=new T[arrayLength];
        heapSize=0;
    }
    ~MinHeap()
    {
        if(heap!=NULL)
        delete[] heap;
    }
```

```
        void push(const T& theElement);
        void pop();
        const T& top()
        {
            return heap[1];
        }
        int size()const
        {
            return heapSize;
        }
        bool empty()const
        {
            return heapSize==0;
        }
        void initialize(T* theHeap,int heapSize);
        void output(ostream& out)const
        {
            for(int i=1;i<=heapSize;i++)
                out<<heap[i]<<" ";
        }
        void deactiveArray()
        {
            if(heap!=NULL)
                delete []heap;
            heap=NULL;
            arrayLength=heapSize=0;
        }

private:
        T *heap;
        int arrayLength;
        int heapSize;
```

```
};


template<class T>
void changeLength1D(T*& a, int oldLength, int newLength) {
	T* tmp = new T[newLength];
	int number = min(oldLength, newLength);
	copy(a, a + number, tmp);
	delete[] a;
	a = tmp;
}


//时间复杂度,高度 log2n
template <class T>
inline void MinHeap<T>::push(const T &theElement)
{
	//数组满,扩大数组
	if(heapSize==arrayLength-1)
	{
		changeLength1D(heap,arrayLength,arrayLength*2);
		arrayLength*=2;
	}

	//父节点下沉
	int currentNode=++heapSize;
	while(currentNode!=1&&heap[currentNode/2]>theElement)
	{
		heap[currentNode]=heap[currentNode/2];
		currentNode/=2;
	}

	heap[currentNode]=theElement;
}
```

```
//时间复杂度 log2n,为高度 h
template <class T>
inline void MinHeap<T>::pop()
{
	if(heapSize==0)
	{
		cout<<"heap empty"<<endl;
		return;
	}

	heap[1].~T();
	T lastElement=heap[heapSize--];

	int currentNode=1,child=2;

	while(child<=heapSize)
	{
		if(child<heapSize&&heap[child+1]<heap[child])
			child++;

		if(lastElement<=heap[child])
			break;


		heap[currentNode]=heap[child];
		currentNode=child;
		child*=2;
	}
	heap[currentNode]=lastElement;
}
```

```
//大根堆的初始化,从有孩子的结点开始依此操作
template<class T>
inline void MinHeap<T>::initialize(T* theHeap,int theHeapSize)
{

    delete[]heap;
    heap=theHeap;
    heapSize=theHeapSize;//5
    for(int root=heapSize/2;root>=1;root--)
    {
        T element=heap[root];
        int cur=root;
        int child=cur*2;

        while(child<=heapSize)
        {
            if(child<heapSize&&heap[child+1]<heap[child])
                child++;

            if(element<=heap[child])
                break;

            //孩子上移
            heap[cur]=heap[child];
            cur=child;
            child*=2;
        }

        heap[cur]=element;
    }
}
```

```
template<class T>
ostream & operator<<(ostream& out,const MinHeap<T> &x)
{
	x.output(out);return out;
}



template<class T>
class HuffmanNode
{
	public:
	LinkedBinaryTree<int>* tree;
	T weight;

	operator T()const {return weight;}
};



//构造 huffmantree

template <class T>
LinkedBinaryTree<int>* createHuffmanTree(T weight[],int n)
{
	MinHeap<HuffmanNode<int>>heap(1);
	HuffmanNode<T>* hnode=new HuffmanNode<T>[n+1];
	LinkedBinaryTree<int> emptyTree;
	for(int i=1;i<=n;i++)
	{
		hnode[i].weight=weight[i];
		hnode[i].tree=new LinkedBinaryTree<int>;
		hnode[i].tree->makeTree(i,emptyTree,emptyTree);
	}
```

```
        heap.initialize(hnode,n);


        HuffmanNode<T> newNode,x,y;
        LinkedBinaryTree<T>* z;
        int ans=0;
        for(int i=1;i<n;i++)
        {
                x=heap.top();heap.pop();
                y=heap.top();heap.pop();


                int w=x.weight+y.weight;
                z=new LinkedBinaryTree<int>;
                z->makeTree(0,*x.tree,*y.tree);
                newNode.weight=w;
                newNode.tree=z;
                ans+=w;
                heap.push(newNode);


                delete x.tree;
                delete y.tree;
        }


        cout<<ans<<endl;
        return heap.top().tree;
}


#include<cstring>
const int N=1e6+10;
bool st[N]={false};
int sl[N]={0};
int ans=0;
```

```
// template<class E>
// static void addLength(BinaryTreeNode<E>*t,int h=0)
// {
//         if(t!=NULL)
//         {
//                 if(t->leftChild==NULL&&t->rightChild==NULL)
//                 {
//                         ans+=weight[t->element]*(h++);
//                 }
//                 addLength(t->leftChild,h+1);
//                 addLength(t->rightChild,h+1);
//         }
// }
#include<map>
int main()
{
        char s[N];
        scanf("%s",s);

        int count=0;
        int len=strlen(s);
        for(int i=0;i<len;i++)
        {
                if(st[s[i]]==false)
                        count++;
                st[s[i]]=true;
                sl[s[i]]++;
        }
        int *weight=new int [count+1];
        memset(st,false,sizeof(st));
        int index=1;
        for(int i=0;i<len;i++)
```

```
        {
            if(st[s[i]]==false)
                weight[index++]=sl[s[i]];
            st[s[i]]=true;
        }
        // int len=s.length();
        // for(int i=0;i<len;i++)
        // {
        //     if(mp.find(s[i])==mp.end())
        //     {
        //         mp[s[i]]=1;
        //     }
        //     else
        //     {
        //         mp[s[i]]++;
        //     }
        // }

        // int index=1;
        // for(map<char,int>::iterator it=mp.begin();it!=mp.end();it++)
        // {
        //     weight[index++]=it->second;
        // }
        createHuffmanTree(weight,count);
    }
```