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：202200130048	姓名： 陈静雯	班级： 6
实验题目：堆及其应用		
实验学时：2	实验日期： 11.30	
实验目的： 堆及其应用		
软件开发工具： Vscode		

1. 实验内容

（1）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。

（2）霍夫曼编码

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）

（1）最小堆

建堆：无顺序放入堆，从最后一个子树开始，检查父节点是否小于子节点，若不小于，把最小的节点上放，父节点下放，以此类推

Push：在堆的最后插入元素，检查元素的节点与父节点谁大，元素大则不变，父节点大，父节点下放，元素往上放，一直检查到根节点

Pop：删除根节点，从根节点开始，把子节点中较小的一个放入父节点，以此类推

排序：输出根节点，pop 根节点，直到堆为空

（2）霍夫曼

先对字符出现的频率创建节点数组建堆，再不断 pop 堆中的两个节点，两个节点组成一个新的节点即他们的父节点，元素值为子节点相加，把父节点 push 进堆，如此循环，直到堆中只有一个节点，即霍夫曼树的根节点。

遍历霍夫曼树，叶节点的元素乘它们各自的层高，相加即为最终结果

3. 测试结果（测试输入，测试输出）

（1）

```
10
-225580 113195 -257251 384948 -83524 331745 179545 293165 125998 376875
-257251
10
1 -232502
-257251
1 -359833
-359833
1 95123
-359833
2
-257251
2
-232502
2
-225580
1 223971
-225580
1 -118735
-225580
1 -278843
-278843
3 10
-96567 37188 -142422 166589 -169599 245575 -369710 423015 -243107 -108789
-369710 -243107 -169599 -142422 -108789 -96567 37188 166589 245575 423015
```

（2）

```
abcdabcaba
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）

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）

（1）

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

template<class T>
class minheap{
public:
    minheap(){ element=NULL; heapsize=0; length=10;}
    void init(int n);
    void push(const T& thelement);
    void pop();
    bool empty(){ return heapsize==0;}
    void peak(){ cout<<element[1];}
private:
    T* element;
    int heapsize;
    int length;
};

template <class T>
```

```
void minheap<T>::init(int n){
    heapsize=n;
    length=2*n;
    element = new T [length];
    for(int i=1;i<=heapsize;i++){
        cin>>element[i];
    }
    for(int i=heapsize/2;i>=1;i--){              //检查每个根节点与子树的关系
        T temp = element[i];
        int child = 2*i;
        while(child<=heapsize){
            if(child<heapsize && element[child] > element[child+1]) child++;
            if(temp<element[child])                                          break;
//找到最小的元素
            element[child/2]=element[child];
            child*=2;
        }
        element[child/2]=temp;
    }
}

template <class T>
void minheap<T>::push(const T& thelement){
    if(heapsize==length-1){                    //空间扩大
        length*=2;
        T* temp = new T [length];
        for(int i=1;i<=heapsize;i++){
            temp[i]=element[i];
        }
        element=temp;
    }
    int cur = ++heapsize;
    while(cur!=1 && element[cur/2]>thelement){//根节点比插入的元素大，根节点下放
        element[cur]=element[cur/2];
        cur/=2;
    }
    element[cur]=thelement;
    cout<<element[1]<<endl;
}

template<class T>
void minheap<T>::pop(){
    if(heapsize==0) return ;
    T thelement = element[heapsize--];    //删除根节点，把最后一个叶节点往上放
    int cur = 1;
    int child = 2;
```

```
    while(child<=heapsize){
        if(child<heapsize && element[child]>element[child+1]) child+=1;
        if(thelement<=element[child]) break;
        element[cur]=element[child];
        cur=child;
        child*=2;
    }
    element[cur]=thelement;
}

template<class T>
void heapsort(minheap<T>& h){
    while(!h.empty()){
        h.peak();
        h.pop();
        cout<<' ';
    }
}

int main(){
    int n;
    cin>>n;
    minheap<int> H;
    H.init(n);
    H.peak();
    cout<<endl;
    int m;
    cin>>m;
    for(int i=0;i<m;i++){
        int p;
        cin>>p;
        if(p==1){
            int num;
            cin>>num;
            H.push(num);
        }
        if(p==2){
            H.pop();
            H.peak();
            cout<<endl;
        }
        if(p==3){
            int t;
            cin>>t;
            minheap<int> H2;
            H2.init(t);
```

```
            heapsort(H2);
        }
    }
}

 (2)
#include<iostream>
using namespace std;

template<class T>
struct huffmannode{
    T weight;
    huffmannode<T>* leftchild;
    huffmannode<T>* rightchild;
    bool operator>(const huffmannode b) const{    //运算符重载
        return weight>b.weight;
    }
    bool operator>=(const huffmannode b) const{
        return weight>=b.weight;
    }
    bool operator<(const huffmannode b) const{
        return weight<b.weight;
    }
    bool operator<=(const huffmannode b) const{
        return weight<=b.weight;
    }
    bool operator==(const huffmannode b) const{
        return weight==b.weight;
    }
    huffmannode(){
        weight=0;
        leftchild=rightchild=NULL;
    }
    huffmannode(const huffmannode<T>& a){
        weight = a.weight;
        leftchild = a.leftchild;
        rightchild = a.rightchild;
    }
    void maketree(huffmannode<T>* a,huffmannode<T>* b){
        weight = a->weight+b->weight;
        leftchild = a;
        rightchild = b;
    }
};
```

```
template<class T>
class minheap{
public:
    minheap(){ element=NULL; heapsize=0; length=10;}
    void init(T a[]);
    void push(const T& thelement);
    void pop();
    bool empty(){ return heapsize==0;}
    T peak(){ return element[1];}
    int size(){ return heapsize;}
private:
    T* element;
    int heapsize;
    int length;
};

template <class T>
void minheap<T>::init(T a[]){
    element = new T [27];
    length=27;
    for(int i=1;i<=26;i++){
        if(a[i].weight!=0){
            element[++heapsize]=a[i];
        }
    }
    for(int i=heapsize/2;i>=1;i--){
        T temp = element[i];
        int child = 2*i;
        while(child<=heapsize){
            if(child<heapsize && element[child] > element[child+1]) child++;
            if(temp<element[child]) break;
            element[child/2]=element[child];
            child*=2;
        }
        element[child/2]=temp;
    }
}

template <class T>
void minheap<T>::push(const T& thelement){
    if(heapsize==length-1){
        length*=2;
        T* temp = new T [length];
        for(int i=1;i<=heapsize;i++){
            temp[i]=element[i];
```

```
        }
        element=temp;
    }
    int cur = ++heapsize;
    while(cur!=1 && element[cur/2]>thelement){
        element[cur]=element[cur/2];
        cur/=2;
    }
    element[cur]=thelement;
}

template<class T>
void minheap<T>::pop(){
    if(heapsize==0) return ;
    T thelement = element[heapsize--];
    int cur = 1;
    int child = 2;
    while(child<=heapsize){
        if(child<heapsize && element[child]>element[child+1]) child+=1;
        if(thelement<=element[child]) break;
        element[cur]=element[child];
        cur=child;
        child*=2;
    }
    element[cur]=thelement;
}

minheap<huffmannode<int>> tree;
int ans=0;

void preorder(huffmannode<int>* x,int num){
    if(x->leftchild==NULL && x->rightchild==NULL){
        ans+=num * (x->weight);
        return ;
    }
    if(x->leftchild!=NULL) preorder(x->leftchild,++num);
    num--;
    if(x->rightchild!=NULL) preorder(x->rightchild,++num);
}

void huffmantree(){     //把堆的前两个节点 pop 之后组成一个新的节点，值为两个节点
                          值相加 push 进堆
    huffmannode<int>* x = new huffmannode<int>;
    while(tree.size()>1){
        huffmannode<int> * l =new huffmannode<int> (tree.peak());
        tree.pop();
```

```
        huffmannode<int> * r = new huffmannode<int> (tree.peak());
        tree.pop();
        x->maketree(l,r);
        tree.push(*x);
    }
    preorder(x,0);
    cout<<ans;
}

int main(){
    huffmannode<int> node[27];
    char c;
    c = getchar();
    while(c!= '\n'){
        node[c-'a'+1].weight++;
        c = getchar();
    }
    tree.init(node);
    huffmantree();
}
```