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：斐波那契堆（难度 5）		
实验学时：4	实验日期： 5.14	
实验目的： 实现斐波那契堆		
软件开发工具： Vscode		

1. 实验内容

（1）设计并实现斐波那契堆（Fibonacci heap）的 ADT，该 ADT 包括 Heap 的组织存储以及其上的基本操作：包括初始化，合并两个堆，查找最小结点，插入结点，删除最小结点等。并分析基本操作的时间复杂性。

（2）实现 Fibonacci heap ADT 的基本操作演示（鼓励应用图形界面）。

（3）采用斐波那契堆对堆优化 Dijkstra 进行优化，对比普通的堆优化 Dijkstra，分析斐波那契堆的实用价值。

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）结点的属性值包括：left 、right 、parent 、child、degree：孩子个数、mark：是否有孩子被删除过

＊每个结点的 child 指针只会有一个，指向它其中任意一个孩子

＊结点 x 成为其他节点的孩子结点后，设值为 false，若之后 x 的孩子减少了，则 mark 变为 true

（2）堆的成员：H.min 堆中 key 值最小的结点，查找最小节点直接找 min 指针、H.n 结点总数

（3）插入结点 insert(x)：插入一个结点 x，直接将它插入到根链表中，判断它是否比 H.min 小，如果比 H.min 小，则 H.min 指向 x，否则不变。

＊分析其势的变化，插入前后 m(H)不变， t(H)+1，则势的增加量为 1。实际代价 O(1)。
摊还代价为 O(1)+1=O(1)

（4）合并两个堆 union（H1，H2）：两个斐波那契堆的合并，将 H1 和 H2 的根链表合并，再确定新链表的最小结点

＊势能变化为 0， Φ(H)-[Φ(H1)+Φ(H2)]=0，实际代价 O(1)
摊还代价=O(1)+0=O(1)

（5）删除最小结点 extract_min（）：将最小结点的每个孩子变为根节点插入到根链表中，再删除最小结点，H.min 顺序右移，最后合并根链表并确定新的 H.min

＊摊还代价=O(lg n)

（6）consolidate（）：在根链表中找到两个度数相同的根 x、y，设 x 为 key 值小的结点；再将 y 链接到 x：从根链表中移除 y 结点，将 y 变成 x 的孩子；重复以上两步直到根链表中没有度数相同的结点为止

＊摊还代价=O(D(n)+t(H))+[(D(n)+1)+2m(H)]-(t(H)+2m(H))=O(D(n))=O(lg n)

（7）对 x 赋新值 decrease（x，k）：把新值 k 赋给 x，保证 k<x；若 k≥y(x.parent)，则不变；若 k<y，切断 x 与其父结点 y 的链接，x 变为根结点；不断判断父结点，x.parent（y），y.parent（z），是否是被标记结点（mark 值是否为 true）是，则对父结点进行级联切断，直到父结点未被标记过，停止切断，更新标记值。

＊摊还代价=O(c)+[(t(H)+c)+2*(m(H)-c+2)]-(t(H)+2m(H))=O(c)+4-c=O(1)

（8）删除结点 delete（node *x）：把要删除的结点减值到-∞，或者任意比 H.min 小的值；删除最小结点

＊摊还代价=O(lg n)

（9）与堆优化的 dikjstra 算法对比：斐波那契堆优化后的 dijkstra 算法就是原本使用普通二叉堆改成斐波那契堆。

	普通堆优化	斐波那契堆优化
插入	O(lg n)	O(1)
删除最小结点	O(lg n)	O(lg n)
合并	最坏O(n)	O(1)

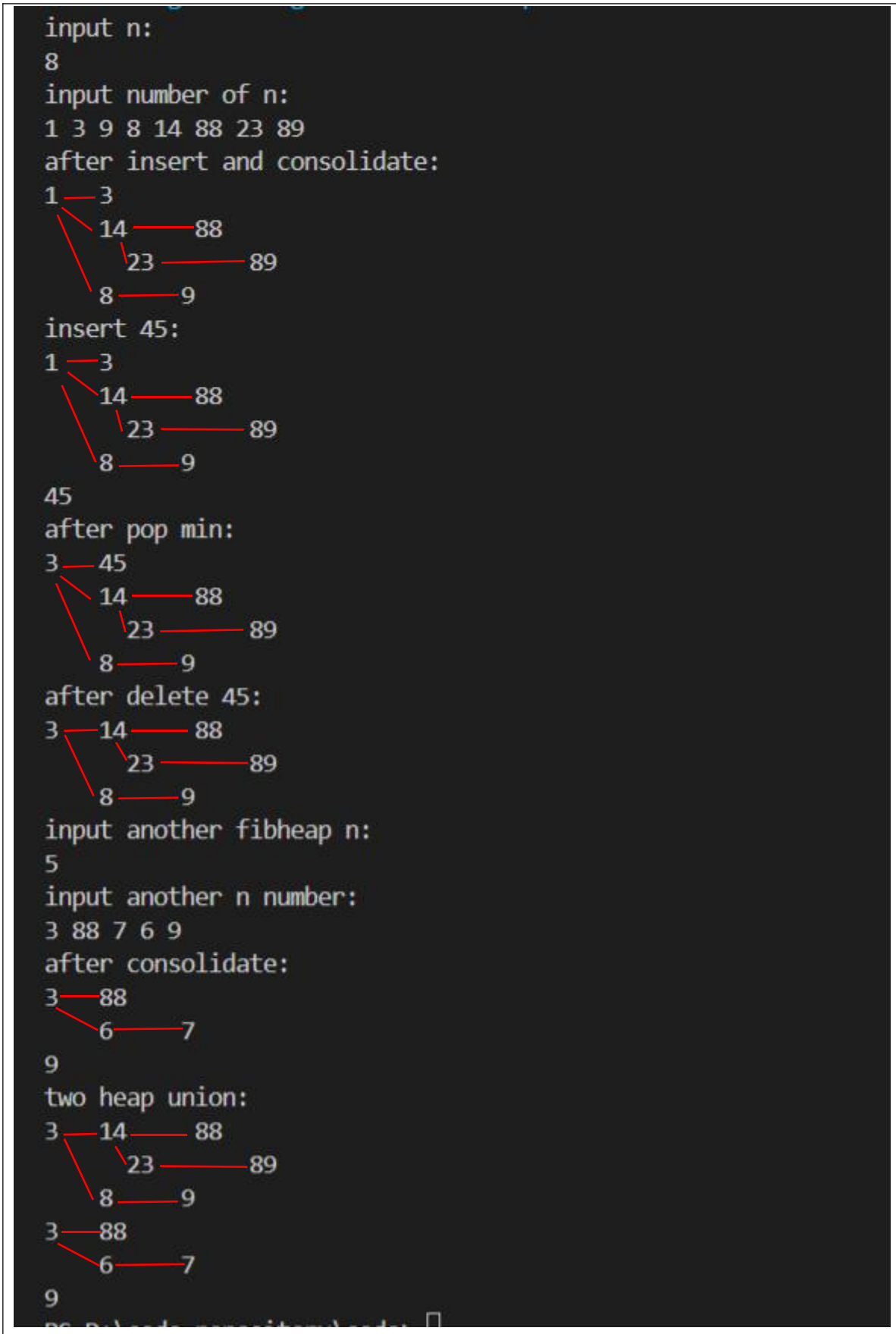
＊优点：

斐波那契堆总体的平均性能优于二叉堆，尤其适合于那些涉及到大量插入、删除和合并操作的场合，比如大规模图的最短路径问题。

＊缺点：

实际应用中要考虑的因素还包括缓存局部性、数据结构的内存占用和实际程序执行时的分支预测等因素。斐波那契堆通常构造更为松散，可能导致更多的内存访问，这在一定程度上可能会抵消其理论上的时间复杂度优势。

3. 测试结果（测试输入，测试输出）


input n:
8
input number of n:
1 3 9 8 14 88 23 89
after insert and consolidate:
1 3
14 88
23 89
8 9
insert 45:
1 3
14 88
23 89
8 9
45
after pop min:
3 45
14 88
23 89
8 9
after delete 45:
3 14 88
23 89
8 9
input another fibheap n:
5
input another n number:
3 88 7 6 9
after consolidate:
3 88
6 7
9
two heap union:
3 14 88
23 89
8 9
3 88
6 7
9

```
input n:
10
input number of n:
8 3 4 59 2 99 45 90 5 455
after insert and consolidate:
2   99
    3     8
      4       59
    45     90
5   455
insert 45:
2   99
    3     8
      4       59
    45     90
5   455
45
after pop min:
3   8
    5     455
      45       99
    4     59
45   90
after delete 45:
3   8
    5     455
    4     59
45   90
99
input another fibheap n:
5
input another n number:
22 33 11 44 77
after consolidate:
11   22
    33     44
77
two heap union:
3   8
    5     455
```

4. 分析与探讨（结果分析，若存在问题，探讨解决问题的途径）

还可以更进一步的用图形化界面展示操作过程，本题仅用空格，不同层级不同父节点通过空格与换行进行简单的展示

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）

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

template<class T>
struct fibnode{
    fibnode<T>* left;
    fibnode<T>* right;
    fibnode<T>* child;
    fibnode<T>* parent;
    int degree;
    bool mark;
    T key;
    fibnode(T k){
        key=k;
        degree=0;
        mark=false;
        child=NULL;
        parent=NULL;
    }
};

template<class T>
class fibheap{
public:
    fibheap(){
        n=0;
        themin=NULL;
    }
    fibheap<T> make_heap(){
        fibheap<T> fib_heap;
        return fib_heap;
    }
    void insert(fibnode<T>* x);
    fibnode<T>* minimun(){
        return themin;
    }
    fibheap<T>* union_fibheap(fibheap<T>& H1,fibheap<T>& H2);
    void extract_min();
    void consolidate();
    void nodelink(fibnode<T>* y, fibnode<T>* x);
```

```
    void decrease(fibnode<T>* x, T k);
    void cut(fibnode<T>* x, fibnode<T>* y);
    void cascading_cut(fibnode<T>* y);
    void deletenode(fibnode<T>* x){
        int tt=themin->key-10;
        decrease(x,tt);
        extract_min();
    }
    void show(){                        //按结构输出所有结点
        if(themin==NULL) return;
        fibnode<T>* z=themin->right;
        cout<<themin->key;
        show_node(themin->child,2);
        while(z!=themin){
            cout<<z->key;
            show_node(z->child,2);
            z=z->right;
        }
    }
    void show_node(fibnode<T>* thenode, int n){
        if(thenode==NULL){
            cout<<'\n';
            return ;
        }
        else{
            fibnode<T>* z=thenode->right;
            for(int i=0;i<n*2-1;i++) cout<<' ';
            cout<<thenode->key;
            show_node(thenode->child,++n);
            n--;
            while(z!=thenode){
                for(int i=0;i<n*2;i++) cout<<' ';
                cout<<z->key;
                show_node(z->child,++n);
                n--;
                z=z->right;
            }
        }
    }
private:
    int n;
    fibnode<T>* themin;
};

template<class T>
void fibheap<T>::insert(fibnode<T>* x){ //直接在 min 结点旁插入
```

```
    if(themin==NULL){
        x->left=x;
        x->right=x;
        themin=x;
    }
    else{
        x->left=themin->left;
        themin->left->right=x;
        themin->left=x;
        x->right=themin;
        if(x->key<themin->key) themin=x;
    }
    n++;
}

template<class T>
fibheap<T>* fibheap<T>::union_fibheap(fibheap<T>& H1,fibheap<T>& H2){ //合并两
个堆
    themin=H1.themin;
    H2.themin->left->right=themin->right;
    themin->right->left=H2.themin->left;
    themin->right=H2.themin;
    H2.themin->left=themin;
    if(H1.themin==NULL && H2.themin!=NULL){
        themin=H2.themin;
    }
    else if(H2.themin!=NULL && H1.themin!=NULL&&H1.themin->key>H2.themin->key){
        themin=H2.themin;
    }
    n=H1.n+H2.n;
    return this;
}

template<class T>
void fibheap<T>::extract_min(){      //删除最小结点
    fibnode<T>* z=themin;
    if(z!=NULL){
        fibnode<T>* w=z->child;
        for(int i=0;i<z->degree;i++){  //把 min 结点的所有孩子插入根链表
            fibnode<T>* x=w;
            w=w->right;
            x->left->right=x->right;
            x->right->left=x->left;
            x->parent=NULL;
            x->mark=false;
            insert(x);
```

```
        }
        z->child=NULL;
        if(z==z->right) themin=NULL;  //从根链表中把 min 结点删除
        else{
            z->left->right=z->right;
            z->right->left=z->left;
            themin=z->right;
            consolidate();
        }
        n--;
    }
}

template<class T>
void fibheap<T>::consolidate(){      //合并根链表
    int d=(int)floor(log(n)/log(2));
    fibnode<T>* A[d+1];
    for(int i=0;i<=d;i++) A[i]=NULL;
    fibnode<T>* w=themin;
    int rootn=1;
    while(w->right!=themin){
        rootn++;
        w=w->right;
    }
    w=themin;
    for(int i=0;i<rootn;i++){  //找所有度数相同的结点
        fibnode<int>* x=w;
        int de=x->degree;
        while(A[de]!=NULL){
            fibnode<T>* y=A[de];
            if(x->key>y->key){
                w=y;
                fibnode<T>* temp=x;
                x=y;
                y=temp;
            }
            nodelink(y,x);      //将 y 链接到 x
            A[de]=NULL;
            de++;
        }
        A[de]=x;
        w=w->right;
    }
    themin=NULL;
    for(int i=0;i<=d;i++){
        if(A[i]!=NULL) insert(A[i]);//将度数数组的节点连成根链表
```

```
    }
}

template<class T>
void fibheap<T>::nodelink(fibnode<T>* y, fibnode<T>* x){
    y->parent=x;
    y->left->right=y->right;
    y->right->left=y->left;
    x->degree++;
    if(x->child==NULL){
        x->child=y;
        y->left=y;
        y->right=y;
    }
    else{
        y->right=x->child->right;
        x->child->right->left=y;
        x->child->right=y;
        y->left=x->child;
    }
    y->mark=false;
}

template<class T>
void fibheap<T>::decrease(fibnode<T>* x, T k){//x 减值到 k
    if(k>x->key) return ;
    x->key=k;
    fibnode<T>* y=x->parent;
    if(y!=NULL && x->key<y->key){
        cut(x,y);
        cascading_cut(y);
    }
    if(x->key<themin->key) themin=x;
}

template<class T>
void fibheap<T>::cut(fibnode<T>* x, fibnode<T>* y){//x 从 y 处切断
    if(y->child==x){
        if(y->degree<=1) y->child=NULL;
        else y->child=x->right;
    }
    x->left->right=x->right;
    x->right->left=x->left;
    n--;
    y->degree--;
    n--;
```

```
    insert(x);            //插入到根链表
    x->parent=NULL;
    x->mark=false;
}

template<class T>
void fibheap<T>::cascading_cut(fibnode<T>* y){//如果 y 是 true, 将 y 也切断
    fibnode<T>* z=y->parent;
    if(z!=NULL){
        if(y->mark==false){
            y->mark=true;
        }
        else{
            cut(y,z);
            cascading_cut(z);
        }
    }
}

int main(){
    fibheap<int> fib_heap;
    int n;
    cout<<"input n:"<<endl;
    cin>>n;
    cout<<"input number of n:"<<endl;
    while(n){
        int t;
        cin>>t;
        fibnode<int>* temp=new fibnode<int> (t);
        fib_heap.insert(temp);
        fib_heap.consolidate();
        n--;
    }
    cout<<"after insert and consolidate:"<<endl;

    fib_heap.show();
    fibnode<int>* temp=new fibnode<int> (45);
    fib_heap.insert(temp);
    cout<<"insert 45:"<<endl;
    fib_heap.consolidate();
    fib_heap.show();
    cout<<"after pop min:"<<endl;
    fib_heap.extract_min();
    fib_heap.show();
    cout<<"after delete 45:\n";
    fib_heap.deletenode(temp);
```

```
    fib_heap.show();
    cout<<"input another fibheap n:"<<endl;
    int m;
    cin>>m;
    fibheap<int> fib_heap2;
    cout<<"input another n number:"<<endl;
    while(m){
        int t;
        cin>>t;
        fibnode<int>* tp=new fibnode<int> (t);
        fib_heap2.insert(tp);
        m--;
    }
    fib_heap2.consolidate();
    cout<<"after consolidate:"<<endl;
    fib_heap2.show();
    fibheap<int> fib_heap3;
    cout<<"two heap union:"<<endl;
    fib_heap3.union_fibheap(fib_heap,fib_heap2);
    fib_heap3.show();
}
```