实验二

PB19030888张舒恒

实验设备和环境

PC一台，Win11企业版操作系统，gcc 9.1.0编译器，Clion 2021.2.2代码编辑器，Excel绘图工具

实验内容

1.斐波那契堆：建立斐波那契堆数据结构，完成INSERT，MINIMUM，DELETE，DECREASE-KEY，EXTRACT-MIN，UNION等基本操作的API编写

2.家族数：建立不相交集合的并查集数据结构，给定一个 N * N 的矩阵 M，表示不同人之间的亲戚关系，求出所有人中的家族数。

实验要求

代码限制C/C++，建立根文件夹80-张舒恒-PB19030888-project2，在根文件夹下建立本实验报告，ex1和ex2实验文件夹，每个实验文件夹中建立3个子文件夹：input文件夹：存放输入数据，src文件夹：源程序，output文件夹：输出数据。

实验步骤及方法

斐波那契堆

1.文件输入输出

采用绝对路径的输入输出流

```
    string from = "C:/Users/ASUS/Desktop/AL/AL2/ex1/input/2_1_input.txt";
    string dest = "C:/Users/ASUS/Desktop/AL/AL2/ex1/output/result.txt";
    string time = "C:/Users/ASUS/Desktop/AL/AL2/ex1/output/time.txt";
    ifstream file_in;
    ofstream file_out, time_out;
    file_in.open(from);
    file_out.open(dest);
    time_out.open(time);
```

2.INSERT

给新节点分配空间，将其链入根链表，判断新节点的key是否更小，如果更小则更新head，节点总数+1。

```
node* insert(int k, int v){//插入节点
        node *p = new node(k, v);
        linkNode(p, head);
        if (p->key < head->key)
            head = p;
        ++n;
        return p;
    }
```

3.MINIMUM

返回head节点指针即是最小key的节点指针

```
    node* minimum()const{
        return head;
    }
```

4.DELETE

将要删除的节点p的key改为当前最小key-1，再删除最小key节点也即需要DELETE的节点

```
    void erase(node *p){//删除节点
        node *p_min = minimum();
        decreaseKey(p, p_min->key - 1);
        extractMin();
    }
```

5.DECREASE-KEY

如果新key更大则报错，否则更改节点的key，如果新的key比父节点key还小，则剪掉以p为根的树，级联剪枝父节点，最后判断是否需要更新head指针

```
    void decreaseKey(node *p, int k){//减小节点key
        if (k>p->key){//新key更大则报错
            cerr << "error" << endl;
            return;
        }
        p->key = k;
        node *par = p->parent;
        if (par != nullptr && (p->key<par->key)){
            prune(p, par);//剪掉以p为根的树
            cascadingPrune(par);//级联剪枝父节点
        }
        if (p->key<head->key)
            head = p;
    }
```

6.EXTRACT-MIN

将要删除的节点所有孩子转入根链表中，在根链表中去除该节点，如果堆已空则head赋为空指针，否则合并度数相同的根节点，最后节点数减1

```
    pair<int, int> extractMin(){//删除最小key节点
        node *p = head;
        removeChildsToRoot(head);
        head->child = nullptr;
        removeNode(head);
        if (head->next == nullptr)//说明堆已空
            head = nullptr;
        else{
            head = head->next;
            consolidate();//合并度数相同的根节点
        }
        --n;
        pair<int, int> tmp = pair<int, int>(p->key, p->value);
```

```
        delete p;
        return tmp;
    }
```

7.UNION

如果被合并堆为空则不用处理，如果当前堆为空则交换两个堆，否则链接两个双向根链表，修改当前堆和被合并堆的head和n

```
    void FibHeapUnion(fibonacci_heap &rhs){//合并两个斐波那契堆
        if (rhs.empty())return;//若被合并堆为空
        if (empty()){//若当前堆为空
            swap(head, rhs.head);
            swap(n, rhs.n);
            return;
        }
        node *head_prev = head->prev;
        head_prev->next = rhs.head->prev;
        rhs.head->prev->prev->next = head;
        head->prev = rhs.head->prev->prev;
        rhs.head->prev->prev = head_prev;
        if (rhs.head->key < head->key)
            head = rhs.head;
        n += rhs.n;
        rhs.head = nullptr;
        rhs.n = 0;
    }
```

8.打印运行时间和结果，画出时间曲线并分析

家族数

1.文件输入输出

采用绝对路径的输入输出流

```
    string from = "C:/Users/ASUS/Desktop/AL/AL2/ex2/input/2_2_input.txt";
    string dest = "C:/Users/ASUS/Desktop/AL/AL2/ex2/output/result.txt";
    string time = "C:/Users/ASUS/Desktop/AL/AL2/ex2/output/time.txt";
    ifstream file_in;
    ofstream file_out, time_out;
    file_in.open(from);
    file_out.open(dest);
    time_out.open(time);
```

2.搜索x所属集合

采用路径压缩启发策略，如果x的所属集合的代表元素就是x则返回x，否则返回x的父节点所属集合并将其作为x的新父节点

```
int findset(int x){    //路径压缩启发策略
    if(parent[x]==x)
        return x;
    else
        return parent[x]=findset(parent[x]);
}
```

3.合并两个集合

采用按秩合并启发策略，将秩小的树链入秩大的树，如果两个树的秩相同则合并后的树的秩还需加1

```
void link(int x,int y){  //按秩合并启发策略
    if(rank_[x]>rank_[y]){
        parent[y]=x;
    }
    else{
        parent[x]=y;
        if(rank_[y]==rank_[x])
            rank_[y]=rank_[y]+1;
    }
}
```

4.合并两个元素所在集合

调用findset求出元素所在集合，调用link合并两个集合，并将总集合树减1

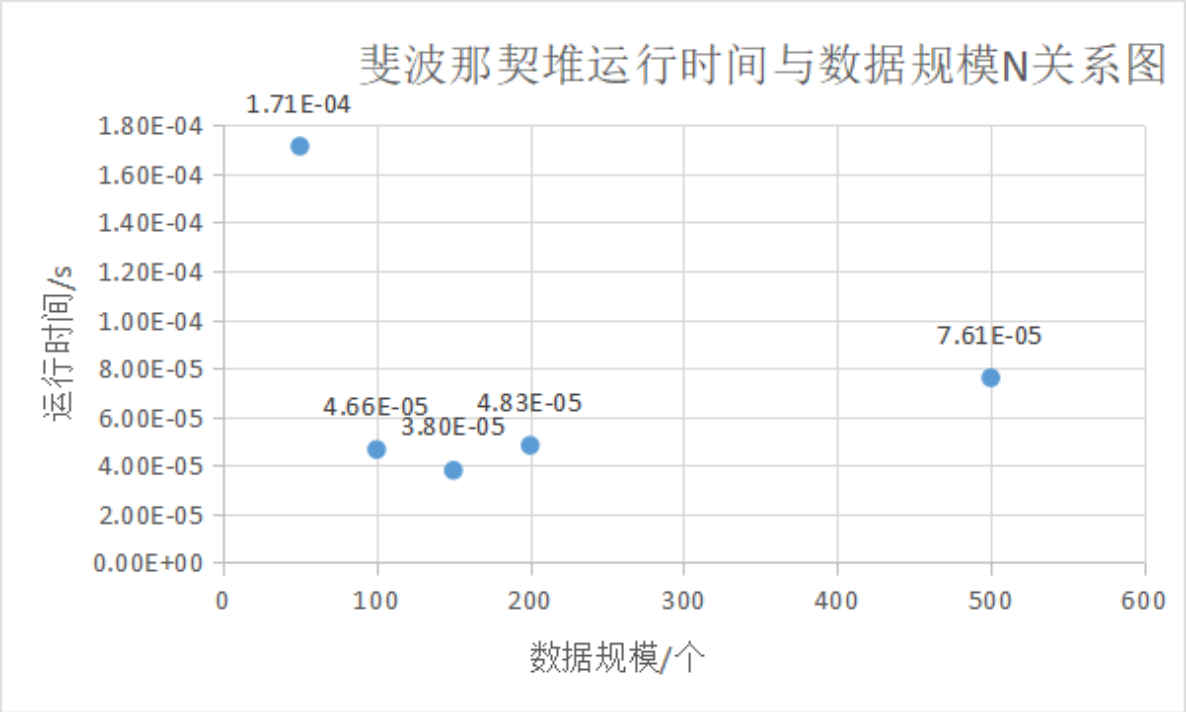
```
void union_(int x, int y, int i){
    int xx = findset(x);
    int yy = findset(y);
    if(xx != yy){
        link(xx, yy);
        NUM[i]--;
    }
}
```

5.打印运行时间和结果，画出时间曲线并分析

实验结果分析

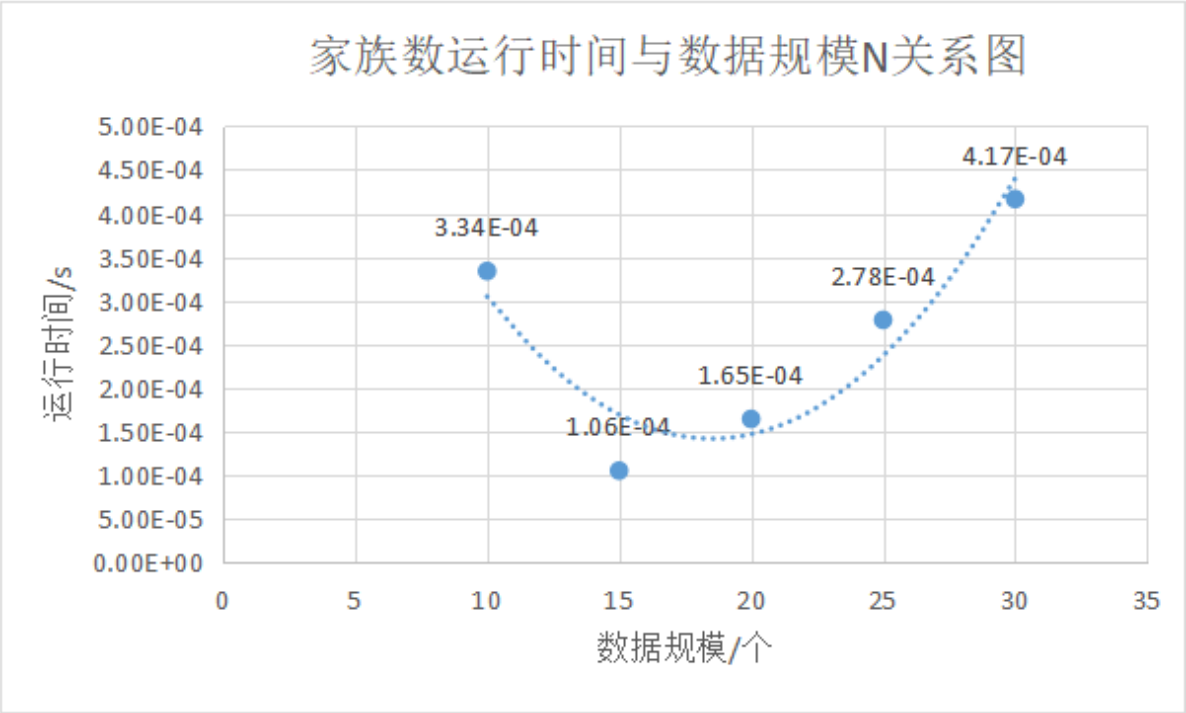
斐波那契堆

斐波那契堆的时间复杂度受不同操作的影响较大，并且在数据规模较小时也受数据分布的影响，下图中第1个堆时间较大可能是根链表较长，第2，3，4，5堆近似随着数据增多线性增长。



家族数

家族数运行时间复杂度为$O(n^2)$，第一组数据不符合主要原因可能是程序中部分变量需要初始化且受数据密度的影响，第2，3，4，5组基本符合。



实验总结

通过本次实验我基本掌握了斐波那契堆和不相交集合的并查集。