Lab3 二叉堆和斐波那契堆

- **实验内容**

  > 通过二叉堆和斐波那契堆的操作，实现对数据的处理，并输出最小的20个数。统计每个操作所需要的时间，归纳每个操作的时间复杂度

- **实验要求**

  > 一共有5组数据，对每组数据依次进行建堆，插入，减值，删除，抽取最小节点的5个操作，并统计操作花费的时间。抽取最小节点的结果输出到result.txt中，时间结果输出到time.txt中，每组数据对应1组结果，并做出不同规模下操作时间的变化示意图

- **实验设备和环境**

  > - 实验平台为：MacOS下的Xcode集成IDLE
  > - 代码语言为：C语言

- **实验方法**

  > - 在二叉堆中采用了最小堆序的建堆方法，插入，减值，删除和抽取最小节点均使用课本上的算法实现
  > - 斐波那契堆的建立先构建所需要的斐波那契堆节点的数据结构，如下所示：
  >
  > ```
  > typedef struct FibNode{
  >     struct FibNode *parent;
  >     struct FibNode *child;
  >     struct FibNode *right;
  >     struct FibNode *left;
  >     int key;
  >     int mark;
  >     int degree; //该节点下孩子的数目
  > }FibNode;
  > ```
  >
  > - 斐波那契堆的5个操作均和课本上的操作类似，初始建堆和插入均调用了一个插入函数完成，如下所示：

```
void Build_Fib(){
    int n;
    int i;
    int x;
    fscanf(fp1,"%d",&n);
    H_min=(FibNode*)malloc(sizeof(FibNode));
    H_min=NULL;
    account=0;
    for(i=0;i<n;i++){
        fscanf(fp1,"%d",&x);
        Insert(x);
    }
}

void Insert_Fib(){
    int n;
    int i;
    int x;
    fscanf(fp2,"%d",&n);
    for(i=0;i<n;i++){
        fscanf(fp2,"%d",&x);
        Insert(x);
    }
}
```

- 在斐波那契堆的减值和删除中，使用了一个新的函数，命名为Find_Key，作用是寻找到堆中和Key值相同的节点x，返回x以完成之后的减值和删除操作，函数如下：

```
FibNode *Find_Key(FibNode *x,int key){
    FibNode *p,*w;
    w=x;
    p=NULL;
    do{
        if(x->key==key){
            p=x;
            break;
        }
        else{
            if(x->child) p=Find_Key(x->child,key);
            if(p!=NULL) break;
        }
        x=x->right;
    }while(x!=w);
    return p;
}
```

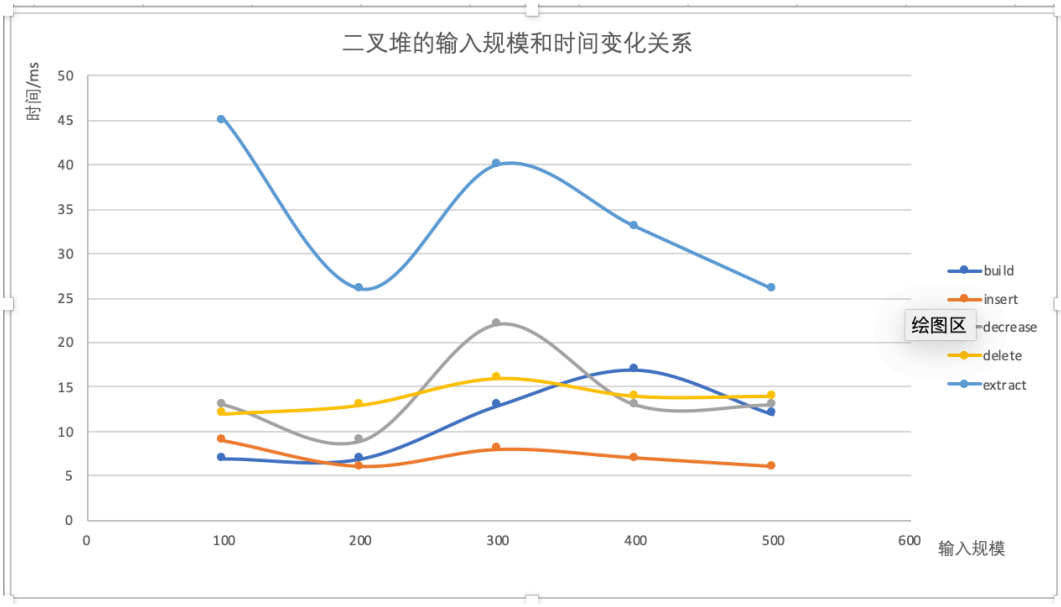
- 在时间的处理上，调用time.h库中的函数clock（），记录每个函数的开始时间begin和结束时间end，函数执行的时间为end-begin。

- **实验步骤**

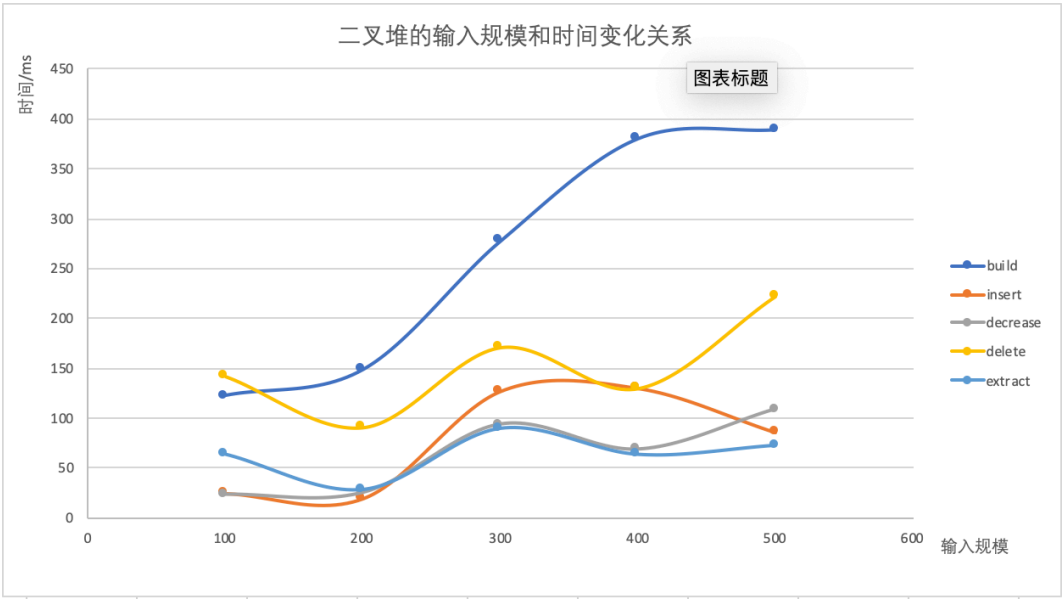
1. 建立文件夹ex1和ex2，其中有input，source和output三个文件夹，分别存放输入数据，源文件和输出数据
2. 完成主函数main
3. 分别实现每个操作所需要的函数功能
4. 综合并完成操作对应时间的分析

- **实验结果**

- 二叉堆的时间结果如下：



- 斐波那契堆的时间结果如下：



- **结果分析**

> 实验结果中时间的部分有比较大的波动，原因应该是在执行的时候，访存的变化导致时间上的变化和规模变化不同步。因为在time.txt中记录的是一次执行的时间，所以随机性很大，在斐波那契堆的时间结果中还能大致上线性变化，二叉堆的变化就显得很凌乱。在处理时间的时候，应该要采用对每组数据执行多次，记录每一次的时间情况，然后对这些数据取平均值作为每一个操作的时间，然后对这些时间进行分析