# 开发思路

参考文献：[1]Chazelle, Bernard. The soft heap: an approximate priority queue with optimal error rate[J]. Journal of the ACM, 2000, 47(6):1012-1027.

本代码参考原算法作者论文中提供的C源码进行实现。

## 基本数据结构

// 每个node的list中的元素的数据类型，item list node

**class itemListNode**

**{**

**public:**

**//这个key是真实值，不会变化的真实值**

**int key;**

**itemListNode \*next;**

**};**

// 软堆的节点node的数据类型

**class** node

{

**public:**

    //ckey的意思是common key，近似值，是这个node的list中所有元素的key的一个上界，可以变

**int** ckey;

**int** rank;

    // 这个技巧用于将高度数的节点表示为二叉节点

    // node是一个rank-1的软队列的根节点

    // 软队列是指从root往next走，不管child

    node \*next;

    // 子节点的rank比node的rank小1

    node \*child;

    // next的ckey要保证比child的ckey小

    // node的list的头一个节点

    itemListNode \*il;

    // node的list的尾节点

    itemListNode \*il\_tail;

    node()

    {

        next = NULL;

        child = NULL;

        il = NULL;

        il\_tail = NULL;

        rank = 0;

        ckey = 0;

    }

**bool** isLeaf()

    {

**if** (next == NULL && child == NULL)

        {

**return** true;

        }

**else**

        {

**return** false;

        }

    }

};

// 优先队列中最顶层的数据结构，各自指向根节点，head之间是双向list。headlist是根据rank值进行从小到大排序的

**class** head

{

**public:**

    // 这个rank同根节点的rank

**int** rank;

    head \*next;

    head \*prev;

    // 指向根节点

    node \*queue;

    // 指向在head list中，排在当前head后面（和自身）的head中，其根节点ckey最小的head

    head \*suffix\_min;

    head()

    {

        queue = NULL;

        next = NULL;

        prev = NULL;

        suffix\_min = NULL;

        rank = 0;

    }

};

// 软堆的数据结构，指由许多head带领的一系列软序列

**class** softHeap

{

**public:**

    // headlist因为是双向链表，所以header是链表的头

    head \*header;

    // headlist的链表尾

    head \*tail;

    // 和错误率相关的一个参数

**int** r;

    softHeap(**int** temp\_r)

    {

        header = new head();

        tail = new head();

        tail->rank = INFTY;

        header->next = tail;

        tail->prev = header;

        r = temp\_r;

    }

## 函数功能

// 从当前的head h向前一个个head更新suffix\_min指针的指向，前提是h后面的head的suffix\_min指针都是对的

**void** fix\_minlist(head **\***h)

// q是一个queue的根节点，我们要把这个queue合并进软堆heap中。如果以q为根节点的queue的rank和原本的headlist中的某一个head具有相同rank，那么进行合并。如果rank相同，则合并，如果rank不同，则插入。

**void** meld(node **\***q)

// insert的实现方法为：新建一个只有一个node的树，然后挂在一个新的head下，然后合并进软堆

**void** insert(**int** newkey)

// 筛一遍node v和它的下面所有next的list和ckey的合法性，进行重新设置，并且删掉部分node节点，压缩整个软序列。

node **\***sift(softheap **\***heap, node **\***v)

// 返回ckey最小的item，并且将其从相应的item list中删掉。然后重新检查一遍当前软队列和整个软堆的合法性。

**int** popMin()

// 删除优先队列中的某个节点，然后返回成功或者失败（bool）

**bool** deleteOne(**int** new\_key)

# 实现功能

使用soft heap数据结构实现了一个容许一定错误的优先队列，可以进行插入insert和弹出最小值popMin。

支持三种输入模式：

1. 手动输入：用户一个个向优先队列中插入数，然后可以在任意时刻进行弹出操作，如果队列已经空了则无效操作。
2. 自动生成：随机生成一定数量的数，分别插入soft heap和C++中<heap>库包含的优先队列，然后全部弹出，进行耗时的对比和错误率的统计。
3. 读取文件：同手动输入，只不过输入指令从input.txt中进行读取，并且同时将弹出的结果显示在cmd界面中以及保存到output.txt中。

# 时间复杂度

作为允许损坏的代价，每个软堆的操作（除insert外）都能够在平均时间内执行的。

# 测试样例

## 手动输入：

选择模式0

设置参数r为3

输入：

i 0

i 1

i 2

i 10

i 1000

i 999

i 998

i 1000000

d 0

d 3

d 2

p

p

p

p

p

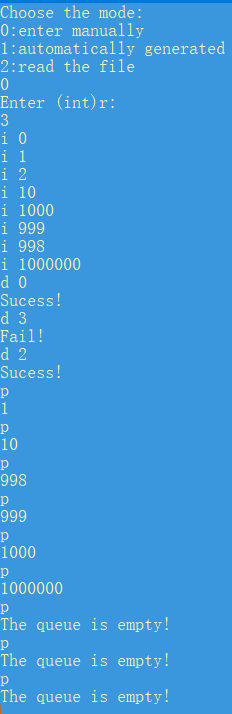
p

p

p

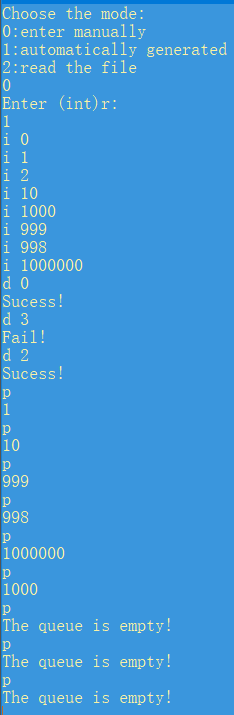
p

结果：



设置参数r为1

结果：



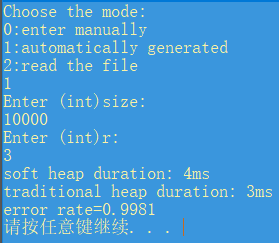
## 自动生成

选择模式1

设置插入的随机数的个数为10000

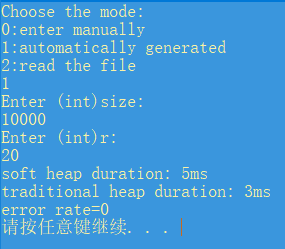
设置参数r为3

结果：



设置参数r为20

结果：



## 读取文件

input.txt：

i 0

i 1

i 2

i 10

i 1000

i 999

i 998

i 1000000

d 0

d 3

d 2

p

p

p

p

p

p

p

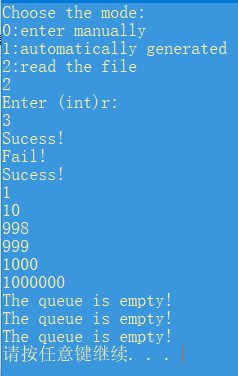
p

p

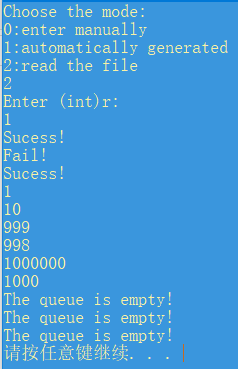
选择模式2

设置参数r=3。

结果：



设置参数r=1



# 使用方法

## 手动输入

1. 双击启动softHeap\_LC\_fix.exe
2. 选择模式0
3. 输入参数r的值（整数）
4. 如果想要向优先队列中插入一个数：输入：i+想要插入的数
5. 如果想要从优先队列中弹出最小的数：输入：d
6. 可以在cmd看到结果

## 自动生成

1. 双击启动softHeap\_LC\_fix.exe
2. 选择模式1
3. 输入需要插入多少个随机数
4. 输入参数r的值（整数）
5. 可以在cmd看到结果

## 读取文件

1. 向input.txt中写入操作，格式同手动输入
2. 双击启动softHeap\_LC\_fix.exe
3. 选择模式2
4. 输入参数r的值（整数）
5. 可以在cmd中和output.txt中看到结果