**堆的创建与实现**

#include <iostream>

#include<malloc.h>

#include <assert.h>

using namespace std;

#define Heap\_Elem\_Type int

#define Heap\_Default\_Size 10

typedef struct Heap {

Heap\_Elem\_Type \*hp;

size\_t capacity;

size\_t size;

};

void HeapInit(Heap&php) {

php.hp = (Heap\_Elem\_Type\*)malloc(sizeof(Heap\_Elem\_Type)\*Heap\_Default\_Size);

assert(php.hp != NULL);

php.capacity = Heap\_Default\_Size;

php.size = 0;

}

//堆的创建方法一，一个一个数据插入

bool HeapFull(Heap&php) {

return php.size == php.capacity;

}

bool HeapEmpty(Heap&php) {

return php.size == 0;

}

//调整函数

//求父结点的公式：如果下标从1开始 则为i/2

//如果下标从0开始 则为（i-1）/2

void \_AdjustUp(Heap&php,int start) {

int j = start;

int i = (j - 1) / 2;

while (j > 0) {

if (php.hp[j] > php.hp[i]) {

int temp = php.hp[j];

php.hp[j] = php.hp[i];

php.hp[i] = temp;

j = i;

i = (j - 1) / 2;

}

else {

break;

}

}

}

void HeapInsett(Heap&php, Heap\_Elem\_Type v) {

if (!HeapFull(php)) {

//先在顺序表的尾部进行插入

php.hp[php.size] = v;

//根据堆的规则进行向上调整

\_AdjustUp(php, php.size);

php.size++;

}

}

void \_AdjustDown(Heap&php, int start) {

int i = start;

int j = 2 \* i + 1;//左子树

while (j < php.size) {

if (j + 1 < php.size&&php.hp[j+1]>php.hp[j]) { //寻找左右子树最大的数

j++;

}

if (php.hp[i] < php.hp[j]) {

int temp = php.hp[j];

php.hp[j] = php.hp[i];

php.hp[i] = temp;

i = j;

j = 2 \* i + 1;

}

else {

break;

}

}

}

void HeapErase(Heap&php) {

if (!HeapEmpty(php)) {

//最后一个数替换堆顶元素

php.hp[0] = php.hp[php.size - 1];

php.size--;

//向下调整

\_AdjustDown(php, 0);

}

}

//堆的创建方法二

void HeapCreat(Heap&php, Heap\_Elem\_Type ar[], int n) {

php.hp = (Heap\_Elem\_Type\*)malloc(sizeof(Heap\_Elem\_Type)\*n);

for (int i = 0; i < n; i++) {

php.hp[i] = ar[i];

}

php.capacity = n;

php.size = n;

//找最后一个分支 n/2-1;

int end = n / 2 - 1;

while (end >= 0) {

\_AdjustDown(php, end);

end--;

}

}

//堆排序

//堆排序的过程是每次将堆顶元素跟堆的最后一个元素交换（不是存储空间的最后一个元素，而是堆结构中的有效节点）

//再对堆进行依次向下调整，反复循环，直到完成所有元素的排序

//升序排序：建立大堆

//降序排序：建立小堆

void HeapSort(Heap&php, Heap\_Elem\_Type ar[], int n) {

while (!HeapEmpty(php)) {

Heap\_Elem\_Type tmp = php.hp[0];

php.hp[0]= php.hp[php.size-1];

php.hp[php.size] = tmp;

php.size--; //交换之后相当于把最后一个元素封存，从接下来要操作的堆中剔除

\_AdjustDown(php, 0);

}

memcpy(ar, php.hp, sizeof(Heap\_Elem\_Type)\*n);

}