<一>Sift覆盖式最小堆实现

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

template <class T>
class MinHeap{
    private:
        T* heap;
        int maxsize;        //堆最大元素容量的个数
        int scale;          //堆的当前元素个数
    public:
        MinHeap(int defaultsize = 10);
        MinHeap(T array[], int n);
        ~MinHeap();
        void MakeEmpty();
        void Show();
        bool IsEmpty();
        bool IsFull();
        bool Insert(const T number);            //向堆尾插入number
        bool Remove(T& removal);                //removal记录堆顶被删除的元素
    protected:
        void SiftDown(int start, int end);      //从start开始，向下调整到到end结束
        void SiftUp(int terminal);              //从terminal开始，向上调整
};

template <class T>
MinHeap<T>::MinHeap(int size){
    this->maxsize = (10 < size)?size:10;
    this->heap = new T[maxsize];
    if(heap == NULL){
        cerr << "Allocation Error!" << endl;
        exit(1);
    }
    this->scale = 0;
}
template <class T>
MinHeap<T>::MinHeap(T array[], int n){
```

```
    this->maxsize = (10 < n)?n:10;
    this->heap = new T[maxsize];
    if(heap == NULL){
        cerr << "Allocation Error!" << endl;
        exit(1);
    }
    for(int i=0; i<maxsize; i++){
        heap[i] = array[i];
    }
    this->scale = n;          //当前堆的元素个数等于数组array的元素个数
    int start = (scale-1)/2;     //start指向堆树的最后子女的父结点所在堆数组的索引
    while(start >= 0){
        SiftDown(start, scale-1);   //scale-1表示堆数组的最后元素的索引
        start--;
    }
}
template <class T>
MinHeap<T>::~MinHeap(){
    delete []heap;       //释放heap指针和其所指的一片空间
}
template <class T>
void MinHeap<T>::MakeEmpty(){
    scale = 0;           //让堆树或堆数组的当前元素个数为0，置空
}
template <class T>
void MinHeap<T>::Show(){
    for(int i=0; i<scale; i++){
        cout << heap[i] << " ";
    }
    cout << endl;
}
template <class T>
bool MinHeap<T>::IsEmpty(){
    return scale == 0;
}
template <class T>
bool MinHeap<T>::IsFull(){
    return scale == maxsize;
}
template <class T>
bool MinHeap<T>::Insert(const T number){
    if(IsFull()){
```

```
        cerr << "Heap is Full!" << endl;
        return false;
    }
    else{
        heap[scale] = number;        //堆尾后一处（索引恰为scale）插入
number
        scale++;                     //堆数组的规模增大一个单元
        SiftUp(scale-1);             //从terminal为索引scale-1（堆数组
最后元素）向上调整
        return true;
    }
}
template <class T>
bool MinHeap<T>::Remove(T& removal){
    if(IsEmpty()){
        cerr << "Heap is Empty!" << endl;
        return false;
    }
    else{
        removal = heap[0];           //把堆顶元素记录给removal
        heap[0] = heap[scale-1];     //把堆尾元素赋给堆顶
        scale--;                     //堆数组的规模减小一个单元
        SiftDown(0, scale-1);        //从堆顶索引0开始，向下调整到索引为
scale-1（堆数组最后元素）
        return true;
    }
}
template <class T>
void MinHeap<T>::SiftDown(int start, int end){
    int i = start;
    int j = 2*start+1;
    T Temporary_Variable = heap[start];
    while(j <= end){
        if(j == end){
            //什么都不需要操作
        }
        if(j < end && heap[j] > heap[j+1]){
            j++;
        }
        if(Temporary_Variable <= heap[j]){
            break;
        }
        else{
```

```
            heap[i] = heap[j];
            i = j;
            j = 2*j+1;              //这里一直进行覆盖操作，而非交换操作
        }
    }
    heap[i] = Temporary_Variable;
}
template <class T>
void MinHeap<T>::SiftUp(int terminal){
    int i = terminal;
    int j = (terminal-1)/2;
    T Temporary_Variable = heap[terminal];
    while(i > 0){
        if(heap[j] <= Temporary_Variable){
            break;
        }
        else{
            heap[i] = heap[j];
            i = j;
            j = (j-1)/2;
        }
    }
    heap[i] = Temporary_Variable;
}

int main(){
    MinHeap<int> heap_1;        //如果写heap_1(10)也可以，不写取默认值
    int array_1[8] = {4,2,5,8,3,6,10,14};
    for(int i=0; i<8; i++){
        heap_1.Insert(array_1[i]);
        cout << "插入 " << array_1[i] << " :\t";
        heap_1.Show();
    }
    cout << endl;
    int removal;
    heap_1.Remove(removal);
    cout << "第 1 次删除的元素为: " << removal << "  堆为: ";
    heap_1.Show();
    heap_1.Remove(removal);
    cout << "第 2 次删除的元素为: " << removal << "  堆为: ";
    heap_1.Show();
    cout << endl;
    cout << endl;
```

```
    cout << endl;
    int array_2[10] = {100,86,48,73,35,39,42,57,66,21};
    cout << "原数组的顺序: ";
    for(int i=0; i<10; i++){
        cout << array_2[i] << " ";
    }
    cout << endl;
    MinHeap<int> heap_2(array_2, 10);
    cout << "其中一最小堆: ";
    heap_2.Show();
    cout << endl;
    cout << endl;
    cout << endl;
    system("pause");
    return 0;
}
```

<二>Sift交换式最小堆实现

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

template <class T>
class MinHeap{
    private:
        T* heap;
        int maxsize;        //堆最大元素容量的个数
        int scale;          //堆的当前元素个数
    public:
        MinHeap(int defaultsize = 10);
        MinHeap(T array[], int n);
        ~MinHeap();
        void MakeEmpty();
        void Show();
        bool IsEmpty();
        bool IsFull();
        bool Insert(const T number);            //向堆尾插入number
        bool Remove(T& removal);                //removal记录堆顶被删除的元素
    protected:
        void SiftDown(int start, int end);      //从start开始，向下调整到到end结束
```

```
        void SiftUp(int terminal);              //从terminal开始，向
上调整
        void Swap(int& i, int& j);              //交换堆元素函数
};

template <class T>
MinHeap<T>::MinHeap(int size){
    this->maxsize = (10 < size)?size:10;
    this->heap = new T[maxsize];
    if(heap == NULL){
        cerr << "Allocation Error!" << endl;
        exit(1);
    }
    this->scale = 0;
}
template <class T>
MinHeap<T>::MinHeap(T array[], int n){
    this->maxsize = (10 < n)?n:10;
    this->heap = new T[maxsize];
    if(heap == NULL){
        cerr << "Allocation Error!" << endl;
        exit(1);
    }
    for(int i=0; i<maxsize; i++){
        heap[i] = array[i];
    }
    this->scale = n;        //当前堆的元素个数等于数组array的元素个数
    int start = (scale-1)/2;    //start指向堆树的最后子女的父结点所在堆数
组的索引
    while(start >= 0){
        SiftDown(start, scale-1);   //scale-1表示堆数组的最后元素的索引
        start--;
    }
}
template <class T>
MinHeap<T>::~MinHeap(){
    delete []heap;      //释放heap指针和其所指的一片空间
}
template <class T>
void MinHeap<T>::MakeEmpty(){
    scale = 0;          //让堆树或堆数组的当前元素个数为0，置空
}
template <class T>
```

```
void MinHeap<T>::Show(){
    for(int i=0; i<scale; i++){
        cout << heap[i] << " ";
    }
    cout << endl;
}
template <class T>
bool MinHeap<T>::IsEmpty(){
    return scale == 0;
}
template <class T>
bool MinHeap<T>::IsFull(){
    return scale == maxsize;
}
template <class T>
bool MinHeap<T>::Insert(const T number){
    if(IsFull()){
        cerr << "Heap is Full!" << endl;
        return false;
    }
    else{
        heap[scale] = number;        //堆尾后一处（索引恰为scale）插入
number
        scale++;                     //堆数组的规模增大一个单元
        SiftUp(scale-1);             //从terminal为索引scale-1（堆数组
最后元素）向上调整
        return true;
    }
}
template <class T>
bool MinHeap<T>::Remove(T& removal){
    if(IsEmpty()){
        cerr << "Heap is Empty!" << endl;
        return false;
    }
    else{
        removal = heap[0];           //把堆顶元素记录给removal
        heap[0] = heap[scale-1];     //把堆尾元素赋给堆顶
        scale--;                     //堆数组的规模减小一个单元
        SiftDown(0, scale-1);        //从堆顶索引0开始，向下调整到索引为
scale-1（堆数组最后元素）
        return true;
    }
```

```
}
template <class T>
void MinHeap<T>::SiftDown(int start, int end){
    int i = start;
    int j = 2*start+1;
    while(j <= end){
        if(j == end){
            //什么都不需要操作
        }
        if(j < end && heap[j] > heap[j+1]){
            j++;
        }
        if(heap[i] <= heap[j]){
            break;
        }
        else{
            Swap(i,j);
            i = j;
            j = 2*j+1;
        }
    }
}
template <class T>
void MinHeap<T>::SiftUp(int terminal){
    int i = terminal;
    int j = (terminal-1)/2;
    while(i > 0){
        if(heap[j] <= heap[terminal]){
            break;
        }
        else{
            Swap(i,j);
            i = j;
            j = (j-1)/2;
        }
    }
}
template <class T>
void MinHeap<T>::Swap(int& i, int& j){
    T t;
    t = heap[i];
    heap[i] = heap[j];
    heap[j] = t;
```

```
}

int main(){
    MinHeap<int> heap_1;          //如果写heap_1(10)也可以，不写取默认值
    int array_1[8] = {4,2,5,8,3,6,10,14};
    for(int i=0; i<8; i++){
        heap_1.Insert(array_1[i]);
        cout << "插入 " << array_1[i] << " :\t";
        heap_1.Show();
    }
    cout << endl;
    int removal;
    heap_1.Remove(removal);
    cout << "第 1 次删除的元素为: " << removal << "  堆为: ";
    heap_1.Show();
    heap_1.Remove(removal);
    cout << "第 2 次删除的元素为: " << removal << "  堆为: ";
    heap_1.Show();
    cout << endl;
    cout << endl;
    cout << endl;
    int array_2[10] = {100,86,48,73,35,39,42,57,66,21};
    cout << "原数组的顺序: ";
    for(int i=0; i<10; i++){
        cout << array_2[i] << " ";
    }
    cout << endl;
    MinHeap<int> heap_2(array_2, 10);
    cout << "其中一最小堆: ";
    heap_2.Show();
    cout << endl;
    cout << endl;
    cout << endl;
    system("pause");
    return 0;
}
```