数据结构-优先队列

优先队列介绍：

11.5 优先队列

1 优先队列（priority queue）可以在 $O(1)$ 时间内获得最大值，并且可以在 $O(\log n)$ 时间内取出最大值或插入任意值。

2 优先队列常常用堆（heap）来实现。堆是一个完全二叉树，其每个节点的值总是大于等于子节点的值。实际实现堆时，我们通常用一个数组而不是用指针建立一个树。这是因为堆是完全二叉树，所以用数组表示时，位置 i 的节点的父节点位置一定为 i/2，而它的两个子节点的位置又一定分别为 2i 和 2i+1。

3 数组实现堆

以下是堆的实现方法，其中最核心的两个操作是上浮和下沉：4 如果一个节点比父节点大，那么需要交换这个两个节点；交换后还可能比它新的父节点大，因此需要不断地进行比较和交换操作，我们称之为上浮；5 类似地，如果一个节点比父节小，也需要不断地向下进行比较和交换操作，我们称之为下沉。如果一个节点有两个子节点，我们总是交换最大的子节点。

题目描述：

给你一个链表数组，每个链表都已经按升序排列。

请你将所有链表合并到一个升序链表中，返回合并后的链表。

示例 1：

```
输入：lists = [[1,4,5],[1,3,4],[2,6]]
输出：[1,1,2,3,4,4,5,6]
解释：链表数组如下：
[
  1->4->5,
  1->3->4,
  2->6
]
将它们合并到一个有序链表中得到。
1->1->2->3->4->4->5->6
```

手写优先队列数据结构

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode() : val(0), next(nullptr) {}
 *     ListNode(int x) : val(x), next(nullptr) {}
 *     ListNode(int x, ListNode *next) : val(x), next(next) {}
 * };
 */
class Solution {
private:
    vector<ListNode*>heap;
    //取最大值
    ListNode*top(){
        return heap[0];
    }
    //插入数据
    void push(ListNode*data){
        heap.push_back(data);
        swim(heap.size()-1);
    }
```

```
    //上浮
    void swim(int pos){
        while(pos>0&&heap[(pos-1)/2]->val > heap[pos]->val){
            swap(heap[(pos-1)/2],heap[pos]);
            pos=(pos-1)/2;
        }
    }
    //删除数据
    void pop(){
        if(heap.empty())return;
        heap[0]=heap.back();
        heap.pop_back();
        sink(0);
    }
    //下沉
    void sink(int pos){
        int n=heap.size();
        //小顶推
        //把子节点中较小的换上来
        while(2*pos+1<n){
            int temp=2*pos+1;
            if(2*pos+2<n&&heap[2*pos+2]->val < heap[2*pos+1]->val){
                temp=2*pos+2;
            }
            if(heap[temp]->val < heap[pos]->val)
            swap(heap[temp],heap[pos]);
            else break;
            pos=temp;

        }
    }


public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        //通过堆来合并K个升序链表
        //然后排序的进行一个输出
        for(auto list:lists){
            while(list){
                push(list);
                list=list->next;
            }
        }
        if(heap.empty())return nullptr;
        ListNode*res=top();
        pop();
        ListNode*temp=res;
        while(!heap.empty()){
            temp->next=top();
```

```
            temp=temp->next;
            pop();
        }
        temp->next=nullptr;
        return res;


    }
};
```

优先队列的作用：

1. 对数据进行排序。（堆排序-可以排出一列数组中最小的前**k**个数）

题目描述：

面试题 17.14. 最小K个数

难度 中等 96

设计一个算法，找出数组中最小的k个数。以任意顺序返回这k个数均可。

示例：

```
输入：  arr = [1,3,5,7,2,4,6,8], k = 4
输出：  [1,2,3,4]
```

```
class Solution {
public:
    vector<int> smallestK(vector<int>& arr, int k) {
        //优先队列和堆试一下
        if(k==0)return {};
        priority_queue<int>que;
        for(int i=0;i<k;i++){
            que.push(arr[i]);
        }
        int n=arr.size();
        for(int i=k;i<n;i++){
            if(arr[i]<que.top()){
                que.pop();
                que.push(arr[i]);
            }
        }
        vector<int>res;
```

```
        while(!que.empty()){
            res.push_back(que.top());
            que.pop();
        }
        return res;
    }
};
```

优先队列的自定义排序（C++）：

```
struct cmp
{
    bool operator()(const pair<int, int>& a, const pair<int, int>& b)
    {
        //优先队列的排序与普通排序相反
        //此为从小到大排序
        return a.first > b.first;
    }
};
 /*将capital和profits绑定，按照capital从小到大排序，用堆排序*/
        priority_queue<pair<int, int>, vector<pair<int, int> >, cmp>
projects;
```