哈尔滨工业大学(深圳)

《数据结构》实验报告

实验五

排序、查找及其应用

学院	计算机科学与技术
姓名	梁鑫嵘
学号	200110619
专业	计算机科学与技术
日期	2021-05-25

1 题目1

一位农场主在对自己的牛群产奶量作统计，想找出“中位数”母牛的产奶量，即一半的母牛产奶量等于或高于该母牛产奶量，一半的母牛产奶量等于或低于该母牛的产奶量。

要求：算法的平均时间复杂度不得大于$O(n\log_2 n)$。

1.1 分析

寻找中位数，我们可以考虑直接对数据进行排序，然后取出中间位置的数作为中位数。但是注意到题目要求“算法的平均时间复杂度不得大于$O(n\log_2 n)$”，所以可以选择快速排序作为排序算法进行求解。

1.2 核心代码

核心代码即快速排序部分，如下：

```
void quick_sort(int *data, int start, int end) {
  if (start >= end) return;
  int key = data[start];
  int i = start, j = end - 1;
  while (i < j) {
    // 找到右边第一个比key小的数
    while (i < j && data[j] > key) j--;
    if (i < j) {
      data[i] = data[j];
      i++;
    }
    // 找到左边第一个比key大的数
    while (i < j && data[i] < key) i++;
    if (i < j) {
      data[j] = data[i];
      j--;
    }
  }
  data[i] = key;
  quick_sort(data, start, i - 1);
  quick_sort(data, j + 1, end);
}
```

2 题目2

设计一个算法，找出未排序数组中最大的k个元素，输出元素请按从小到大顺序排序。

要求：输出要按照从小到大的顺序排序，算法的平均时间复杂度不得大于$O(n\log_2 n)$。

2.1 分析

这题也可以直接排序之后得出结果，结合题目限制，可以选择堆排序来完成。

2.2 核心代码

在此题目中，堆排序过程是由本人自己写的`chilib::priority_queue`（以同样是自己写的`chilib::vector`作容器）来完成的。相关代码如下：
（`chilib/queue.hpp`）

```
/*!
 * 上浮操作
 * @param index 操作节点
 */
void shift_up(size_t index) {
  if (cmp(data[father(index)], data[index])) {
    std::swap(data[index], data[father(index)]);
    shift_up(father(index));
  }
}

/*!
 * 下沉操作
 * @param index 操作节点
 */
void shift_down(size_t index) {
  if (left(index) >= data.length()) return;
  if (right(index) >= data.length()) {
    if (cmp(data[index], data[left(index)])) {
      std::swap(data[left(index)], data[index]);
      shift_down(left(index));
    }
    return;
  }
  if (cmp(data[right(index)], data[left(index)])) {
```

```
26      if (cmp(data[index], data[left(index)])) {
27        std::swap(data[left(index)], data[index]);
28        shift_down(left(index));
29      }
30    } else {
31      if (cmp(data[index], data[right(index)])) {
32        std::swap(data[right(index)], data[index]);
33        shift_down(right(index));
34      }
35    }
36  }
37  /*...*/
38  /*!
39   * 向队列尾部添加元素
40   * @param d 元素
41   */
42  void push(T d) {
43    data.emplace_back(d);
44    shift_up(data.length() - 1);
45  }
46
47  /*!
48   * 取队列头元素
49   * @return 元素引用
50   */
51  T &top() {
52    empty_check();
53    return data[0];
54  }
55
56  /*!
57   * 弹出队列头元素
58   */
59  void pop() {
60    empty_check();
61    T back = data[data.length() - 1];;
62    data.pop_back();
63    if (data.empty()) return;
```

```
    data[0] = back;
    shift_down(0);
  }
```

3 题目3

学校组织同学们接种新冠疫苗，需要统计M个同学们空闲的时间段，故需要统计人数最多的时间段，从而调配疫苗的供应量。空闲时间被分为了N个时间段，其中N可能会非常大，可以假设N为1亿，也就是100000000（这要求程序不能声明长度为N的数组或定义N个变量），若未考虑此情况则不能得分。

要求：排序算法的平均时间复杂度不得大于$O(M\log_2 M)$。

3.1 分析

当在纸上尝试直接作出给出的输入样例的时候，能够推演出程序运行规律，从而模拟这种规律写出该题目。

基本思想为：首先分辨出这些学生所占据的阶段的划分，解决空闲时间被划分得很多的问题，接着从头到尾遍历可用划分，每遇到一个同学有可用时间就把当前计数加一，遇到一个同学可用时间段过去就当前计数减一。最终计算哪些划分的可用学生数量最多。

至于排序算法部分，基于适用性，仍然可使用基于堆排序的优先队列，平均时间复杂度为$O(M\log_2 M)$。

3.2 核心代码

完整代码请看文件。

```
int main() {
  int n, m;
  // 读取数据
  freopen("5_3_input.in", "r", stdin);
  int case_count = 1;
  while (scanf("%d%d", &n, &m) > 0) {
    printf("==== Case %d ====\n", case_count++);
    // 用优先队列
    chilib::priority_queue<StuData, chilib::greater<StuData>> q;
    // 用来记录有多少阶段
    chilib::priority_queue<int, chilib::greater<int>> periods_queue;
    chilib::map<int, bool> periods_map;
    chilib::vector<int> periods;
```

```
    chilib::vector<Period> periods_data;
    chilib::vector<int> points_count;
    // 从start转换到index
    chilib::map<int, int> start2index;
    for (int i = 0; i < m; i++) {
      StuData d;
      scanf("%d%d", &d.start, &d.end);
      q.push(d);
      if (!periods_map.has(d.start)) {
        periods_queue.push(d.start);
        periods_map[d.start] = true;
      }
      if (!periods_map.has(d.end)) {
        periods_queue.push(d.end);
        periods_map[d.end] = true;
      }
    }
    int started = -1;
    while (!periods_queue.empty()) {
      periods.emplace_back(periods_queue.top());
      if (started > 0) {
        periods_data.emplace_back(Period{started,
periods_queue.top()});
        start2index.insert(chilib::pair<int, int>{started, (int)
periods_data.size() - 1});
      }
      points_count.emplace_back(0);
      started = periods_queue.top();
      periods_queue.pop();
    }
    start2index.insert(chilib::pair<int, int>{started, (int)
periods_data.size()});
    chilib::vector<bool> points_same(points_count.size());
    chilib::priority_queue<int, chilib::greater<int>> index_ends;
    int count = 0;
    for (int index = 0; index < points_count.size(); index++) {
      bool changed = false;
      // 遇到新的开始阶段就计数加一
```

```
      while (!q.empty() && start2index[q.top().start] == index) {
        auto &top = q.top();
        index_ends.push(start2index[top.end]);
        count++;
        changed = true;
        q.pop();
      }
      points_count[index] = count;
      // 遇到阶段结束就减一
      while (!index_ends.empty() && index_ends.top() == index) {
        int top = index_ends.top();
        index_ends.pop();
        count--;
        changed = true;
      }
      if (!changed) points_same[index] = true;
    }
    chilib::vector<std::pair<int, int>> result;
    int period_max_count = 0;
    for (int i : points_count)
      if (i > period_max_count) period_max_count = i;
    for (int i = 0; i < points_count.size(); i++) {
      if (period_max_count == points_count[i]) {
        int period_index_max_start = i;
        int origin_i = i;
        while (i != points_count.size() - 1 && points_count[i + 1] ==
period_max_count && points_same[i + 1]) i++;
        if (i != origin_i) {
          int period_index_max_end = ++i;
          result.emplace_back({periods[period_index_max_start - 1],
periods[period_index_max_end - 1]});
        } else {
          result.emplace_back({i + 1, i + 1});
        }
      }
    }
    for (int i = 0; i < result.size(); i++)
      printf("%d %d%s", result[i].first, result[i].second,
```

```
            i == result.size() - 1 ? "\n" : ", ");
    }
    return 0;
}
```

> 注：本实验代码已经保证输出和样例输出一致，故不放出成功截图。