程序设计（A类）第二次大作业(ICPC)——复杂度分析报告

ACM_CLASS_2022 李兴阳 2022.10.31

综述

在这一文档中，我会逐个分析、推导一个操作的**最坏**时间复杂度，并证明它们是符合要求的。

添加队伍

我是采用**unordered_map**来存放每支队伍的。

这是一个使用**hash_table**来映射、组织数据的数据结构（在表格位置重复时再特殊地进行其他操作）。访问元素的时间复杂度几乎为$O(1)$。

AddTeam函数的具体实现代码如下：

```
std::unordered_map<std::string, Team> team_map;// store teams' inforamtion via unordered_map
void AddTeam() {
  // add a team
  if (start_check) {// if we have started, teams can't be added
    std::cout << "[Error]Add failed: competition has started." << std::endl;
    return;
  }
  std::string name;
  std::cin >> name;
  auto iter = team_map.find(name);
  if (iter != team_map.end()) {
    // whether such team has existed
    std::cout << "[Error]Add failed: duplicated team name." << std::endl;
    return;
  }
  std::cout << "[Info]Add successfully." << std::endl;
  team_map[name].team_name = name;
}
```

可以看到，里面的主要操作只有**查找键值为"name"的元素和在该元素之前不存在时插入键值为"name"的元素，这两个操作（平均来说，除非是退化到线性时）都是**$O(1)$级别的。

所以我们AddTeam过程的时间复杂度为$O(1)$,为常数级别，符合上限为$O(\log N)$的要求

提交题目

这一过程我的思路分为这么几个部分：

1. 当场面不为冻结状态，我们直接在**unordered_map（team_map）**中查找并更新这道题目的状态和对应的罚时、通过状态等数值（这些操作是$O(1)$级别的），并在**team_set**中更新排名（当然，我们查找的还是Flush()操作得出的排名）。
2. 当场面为冻结状态，我们在**另一个unordered_map(frozen_map)**中进行相似的操作，但是不会在**team_set**中更新排名。

   具体实现代码（主体）如下：

```
  Team &team_edit = team_map[name];
  Team &team_actual = unfrozen_map[name];
  if (!freeze_check) {
    // no freeze, save the information into the set
    Team old_team = team_map.find(name)->second;
    team_set.erase(old_team);
    if (status == "Accepted") {
      // update the necessary information
    } else {
        // not accepted, update some other information
    }
    Team new_team = team_map.find(name)->second;
    team_set.insert(new_team);
  } else {
    // we need to use another unordered_map to store the updated information
    // the process is somehow similar to the aforementioned one
  }
```

可以看到，如上文分析的那样，在**unordered_map**中查找、编辑的过程是$O(1)$级别的。（我们的题目数量、编辑Team中元素的次数都有上限）

另一方面，**set**的底层结构是红黑树（一个近似平衡的**二叉搜索树**），这意味着在其中删除、插入元素的复杂度可以看作是$O(logN)$的。

所以我们可以得到Submit()过程的复杂度为$O(\log N)$的，这也是符合要求的。

刷新榜单

根据题意，刷新榜单后我们查询到的排名应当是刷新榜单这一时刻的排名。由于我们的**team_set**已经是在实时更新排名的，那么只要在这个过程中遍历**team_set**并将每个元素的排名传回**unordered_map**即可。

具体代码如下：

```
void Flush() {
  // updating the rank
  std::cout << "[Info]Flush scoreboard." << std::endl;
  int temp_rank = 1;
  for (auto iter = team_set.begin(); iter != team_set.end(); iter++) {
    // updating the rank one by one
    team_map[iter->team_name].rank = temp_rank;
    temp_rank++;
  }
}
```

可以看到，我们只需要遍历**team_set**，这一过程等价于在红黑树中查找了N次（可以看作一种二分查找），所需要的时间复杂度为$O(N)\cdot O(\log(N))=O(N\cdot\log(N))$,符合最坏复杂度的要求。

滚榜

于我而言，滚榜是做起来最繁琐的一步（算法一开始也没有完全想清楚）

根据题意，这一过程主要分为三个部分：

1. 打印滚榜前的最新榜单（这等价于一次Flush,时间复杂度为$O(N \cdot \log(N))$
2. 进行滚榜操作，输出排名上升的队伍、它超过的队伍的第一名、以及更新的通过题数、总罚时

   这一过程中我们首先定位榜单最后的队伍，即

   ```
   auto last_iter = team_set.end();
     --last_iter;// find the last team
   ```

   然后寻找他们有冻结状态的第一道题目，以**team_map**中找到的这道题的信息为抓手在**team_set**中删除，并在**unfrozen_map**中找到对应的真实状态，完成更新。最后在**team_set**中插入更新的元素。我们通过比较原始状态和更新后状态中该队伍后面的队伍是否相同来判断队伍排名是否提升。

   最后，如果该队伍没有冻结的题目或者是仅仅有一道冻结的题目，那我们可以完成**--last_iter;**来把目光放到它的上一道题。

   这一过程的最坏时间复杂度为
   $O(\log N)(在set中查找) \cdot O(N) \cdot m(m为题目数量) = O(N \cdot \log(N))$。
3. 打印全部解冻后的榜单（等价于我们**unfrozen_map**上更新的榜单），只要清空**team_map**把其中的元素全部插入**team_set**，同时更新最新排名即可，时间复杂度为
   $N \cdot O(\log(N)) + O(N) + O(N) = O(N \cdot \log(N))$。

故而滚榜过程的总时间复杂度为$O(N \cdot \log(N))$，符合要求。

查询队伍排名

查询排名的步骤相对比较简单，因为在每次Flush的过程中我们已经将更新过的排名传回原来的**unordered_map**，所以只要在**unordered_map**中查询以name为键值的元素并输出它的排名就可以了。根据上文对**unordered_map**的阐述，该过程时间复杂度为$O(1)$，显然是符合最坏复杂度要求的。

查询队伍提交情况

尽管从代码实现来看这一部分十分繁琐，但是我们其实可以发现这一部分代码的本质也就是在**unordered_map**中查询到键值为name的元素的一些信息，所以该过程的时间复杂度为$O(1)$，显然是符合最坏复杂度要求的。

附1：开始比赛

这一过程中其实是蕴含了Flush()的，因为我们要把队伍按照字典序排序，所以时间复杂度为$O(N \cdot \log(N))$。

附2：封榜

我的封榜代码如下：

```
void Freeze() {
  if (freeze_check && !scroll_check) {
    std::cout << "[Error]Freeze failed: scoreboard has been frozen." <<
std::endl;
    return;
  }
  freeze_check = true;
```

```
  std::cout << "[Info]Freeze scoreboard." << std::endl;
  // the state of submission_after_freeze should be updated to 0!
  for (auto iter = team_map.begin(); iter != team_map.end(); iter++) {
    memset(iter->second.after_freeze, 0, sizeof(iter->second.after_freeze));
    unfrozen_map[iter->first] = iter->second;
  }
  scroll_check = false;
}
```

可以看到，在这个过程中，我遍历了**team_map**，对其中after_freeze[]（规模为26）进行了清空，并且将其中的元素对应地传入了**unfrozen_map**，这个过程的时间复杂度为$O(N \cdot 26 \cdot 1) = O(N)$。这是符合林田川助教的要求的。

Acknowledgements

感谢林田川助教和周秉霖助教不厌其烦地回答我的各种问题，并且向我科普了STL库中的诸多数据结构、它们的各类操作的复杂度以及教会了我许多DEBUG技巧。

感谢陈圣尧同学对我的许多帮助和陪伴，以及对我许多愚钝问题的耐心解答。