第二周直播课（上午）作业讲解

洛谷网校
基础衔接提高计划
2025-02
Maxmilite

洛谷
www.luogu.com.cn

图书馆自助存取柜

题目链接 https://www.luogu.com.cn/problem/T568459

图书馆有 n 个自助存取柜，每个柜子有 a_i 个柜格（未知）。进行 q 次操作：

1. 在第 i 个自助存取柜的第 j 个柜格存入物品 k。
2. 查询第 i 个自助存取柜的第 j 个柜格的物品。

$n, q \leq 10^5$，总柜格数量不超过 10^7，保证查询的柜格曾存过物品。

图书馆自助存取柜

解析

注意到不知道每个柜子的具体大小，只知道总和。且二维数组会内存超限。暗示可能需要动态数组。

使用 `vector` 来解决即可。

对 `vector v`，当访问 `v[i]` 时，如果 `i >= v.size()`，那么此时的行为未定义。即，C++ 委员会没有规定此时的行为标准，发生什么与运行环境有关。

为了保险，对本题而言，操作 1 需要适时使用 `vector` 的 `resize()` 函数来调整数组大小。这个函数在上节课中曾提及过。

图书馆自助存取柜

代码实现

```
vector<int> a[100010];
int n, q;
cin >> n >> q;
while(q--) {
    int opr, x, y, z;
    cin >> opr;
    if(opr == 1) {
        cin >> x >> y >> z;
        if(a[x].size() < y + 1)
            a[x].resize(y + 1);
        a[x][y] = z;
    } else {
        cin >> x >> y;
        cout << a[x][y] << endl;
    }
}
```

关于 STL 动态数组（vector）的警示

解析

对 vector v，当访问 v[i] 时，如果 i >= v.size()，那么此时的行为未定义。即，C++ 委员会没有规定此时的行为标准，发生什么与运行环境有关。比赛时大概率会报 Runtime Error。

复习：常用的 vector 调整大小的方式：

1. push_back 元素；
2. resize 函数；
3. 创建时定义好（例如 cin >> n; vector<int> v(n); ）

书堆管理

题目链接 https://www.luogu.com.cn/problem/T568460

实现一个书堆，支持以下操作：

- add(x)：加入编号为 x 的书。
- remove()：取出书堆顶上的书，若书堆为空则输出 Empty。
- peek()：查看书堆顶上的书的编号，若书堆为空则输出 NoBook!。
- count()：输出书堆内书的数量。

$1 \le T, n \le 10^6$，总操作次数不超过 10^6，$0 \le x < 2^{64}$。

书堆管理

解析

栈例题。

add，remove，peek，count 分别对应 push，pop，top，size。

直接使用 stack 套用操作即可。注意 remove，peek 时需要首先判断栈非空。

$x < 2^{64}$ ，long long 是不够的，需要使用 unsigned long long。

单个测试点内有多组测试。执行每组测试前均需将栈清空。

书堆管理

代码实现

```
#define ull unsigned long long
stack<ull> s;
while (!s.empty()) { s.pop(); }
cin >> q;
for (ull j = 1; j <= q; j++) {
    cin >> x;
    if (x == "add") {
        ull tmp; cin >> tmp; s.push(tmp);
    }
    if (x == "remove") {
        if (s.empty()) cout << "Empty" << endl;
        else s.pop();
    }
    if (x == "peek") {
        if (s.empty()) cout << "NoBook!" << endl;
        else cout << s.top() << endl;
    }
    if (x == "count") {
        cout << s.size() << endl;
    }
}
```

消息管理系统

题目链接 https://www.luogu.com.cn/problem/T568461

设计一个消息管理系统，支持以下操作：

- arrive(x)：系统中到来一条最新消息 x。
- serve()：移出最早到来的、且仍存在于系统中的一条消息。如果系统为空，则输出 ERR_CANNOT_POP。
- front()：输出最早到来的、且仍存在于系统中的一条消息。如果系统为空，则输出 ERR_CANNOT_QUERY。
- total()：输出系统内消息个数。

$$n \leq 10000$$

消息管理系统

解析

队列例题。

`arrive`, `serve`, `front`, `total` 分别对应 `push`, `pop`, `front`, `size`。

直接使用 `queue` 套用操作即可。注意 `serve`, `front` 时需要首先判断队列非空。

好像没有更多的坑了。

消息管理系统

代码实现

```
queue<int> q;
cin >> op;
if (op == 1) {
    cin >> x; q.push(x);
}
if (op == 2) {
    if (q.empty()) cout << "ERR_CANNOT_POP" << endl;
    else q.pop();
}
if (op == 3) {
    if (q.empty()) cout << "ERR_CANNOT_QUERY" <<
endl;
    else cout << q.front() << endl;
}
if (op == 4) {
    cout << q.size() << endl;
}
```

关于 STL 栈 & 队列的警示

解析

在使用 STL 栈的 pop/top 方法，队列的 pop/front 方法时，一定要确认栈/队列内非空。

在栈/队列为空的情况下使用上述方法，会直接导致程序报告运行时错误（Runtime Error），正式赛时该测试点直接得零分。

对于 STL 容器变量 s，可以使用

```
if (s.size() && s.top()/pop()/front()) {...}
```

来保证容器内非空。涉及到运算符短路，仅做简单介绍。

STL 容器的 size/empty 函数的时间复杂度都是 $O(1)$ 的。

火车车厢管理系统

题目链接 https://www.luogu.com.cn/problem/T568462

实现一个火车车厢管理系统，初始状态下只有一个编号为 1 的车厢。系统需要支持以下操作，保证任何时候车厢中的编号均不相同。

- 将编号为 y 的车厢挂到编号为 x 的车厢后面；
- 查询编号为 x 的车厢后面是哪个车厢。如果编号为 x 的车厢是最后一个车厢，则输出 0；
- 将编号为 x 的车厢后面的车厢从列车中删除，不改变其他车厢的顺序。

$$1 \leq x, y \leq 10^6, \qquad 1 \leq q \leq 10\text{^}5$$

火车车厢管理系统

解析

只关心某节车厢的前面和后面是谁，而不关心这节车厢是从头数第几个车厢。考虑链表。

对这道题而言，我们需要随时查询某节车厢的下一节是什么（不是第几个，而是车厢这个数字本身的下一个）。因此考虑使用手写链表，使用数组。

使用 `nxt[i]` 表示数字 `i` 代表的车厢的下一节是什么即可，剩余的与普通的链表操作一致。

火车车厢管理系统

代码实现

```
int op;
cin >> op;
if (op == 1) {
    int x, y;
    cin >> x >> y;
    nxt[y] = nxt[x];
    nxt[x] = y;
} else if (op == 2) {
    int x;
    cin >> x;
    cout << nxt[x] << endl;
} else if (op == 3) {
    int x;
    cin >> x;
    nxt[x] = nxt[nxt[x]];
}
```

小镇邮递员的派送路线

题目链接 https://www.luogu.com.cn/problem/T568463

现有 m 个投递箱，每个投递箱的门牌号为 a_i。有 n 封邮件，每封邮件的投递地址为 b_i。

要求为每封邮件推荐一个投递箱，使得所有邮件的不满意度和最小。

不满意度是投递箱的门牌号和邮件投递地址之差的绝对值。

$$1 \le n, m \le 10^5, 0 \le a_i, b_i \le 10^6$$

小镇邮递员的派送路线

解析

对于每一个 b_i，求出大于等于 b_i 的最小值和小于等于 b_i 的最大值。

将二者进行比较，取差值更小者计入答案。

特别地，可以先尝试寻找大于等于 b_i 的最小值所在的下标 p。如果该值等于 b_i，则直接将不满意度记为 0，否则取 p 和 $p-1$ 对应的值进行判断。

以上方法都需要判断边界（找不到最小值/最大值，或 $p=1$ 的情况），以防止出现找不到邮箱的情况。

使用 `lower_bound` 来做也是可以的，写起来也会更简单。

小镇邮递员的派送路线

代码实现

```
lint find(lint x) {
    lint l = 1, r = m, ans = 1e+9;
    lint now = m + 1;
    while (l <= r) {
        lint mid = (l + r) / 2;
        if (a[mid] == x) return 0;
        if (a[mid] > x) now = mid, r = mid - 1;
        else l = mid + 1;
    }
    if (now == 1) return abs(a[now] - x);
    else return min(abs(a[now] - x), abs(a[now - 1] - x));
}
```

```
cin >> m >> n;
for (int i = 1; i <= m; i++) cin >> a[i];
sort(a + 1, a + m + 1);
for (int i = 1; i <= n; i++) {
    cin >> b[i]; sum += find(b[i]);
}
cout << sum << endl;
```

扫地机器人

参见：扫地机器人题解.PDF

考虑二分答案。

二分清扫时间 t，考虑时间 t 内每个机器人的路线分配。

从最左边的机器人开始，注意到它左边的区域由它清理是最优的。如果有剩余时间，它应该在保证能回来的前提下尽可能向右移动。

对于第二个机器人也是同样的，如果它左边（即与第一个机器人间隔的区域）有未被清理的方格，应该首先清理这些方格。如果有剩余时间，也应该在保证能回来的前提下尽可能向右移动。以此类推。

【模板】双端队列 1

题目链接 https://www.luogu.com.cn/problem/B3656

实现 m 个双端队列，支持 q 次操作：

- push_back(a,x)：在第 a 个双端队列尾部插入元素 x；
- pop_back(a)：在第 a 个双端队列尾部弹出元素；
- push_front(a,x)：在第 a 个双端队列头部插入元素 x；
- pop_front(a)：在第 a 个双端队列头部弹出元素；
- size(a)：查询第 a 个双端队列的元素个数；
- front(a)：查询第 a 个双端队列的队首元素；
- back(a)：查询第 a 个双端队列的队尾元素。

$1 \le m,q \le 10^6, 1 \le x \le 10\text{^}9$

【模板】双端队列 1

解析

deque 是一个非常强大的 STL 容器，几乎实现了常见 STL 容器能做到所有功能。

代价是高额的时间和空间常数。对于本题，如果直接使用 10^6 个 deque，则会直接 MLE。

参见 https://zh.cppreference.com/w/cpp/container/deque

如果只想使用 push/pop front/back，不需要用 [] 来随机访问下标，那可以使用 list 快捷地代替。本题使用 list 完成。

【模板】双端队列 1

代码实现

```
map<int, list<int>> d;
while (q--) {
    string op; cin >> op;
    if (op == "push_back") {
        int a, x; cin >> a >> x; d[a].push_back(x);
    } else if (op == "pop_back") {
        int a; cin >> a; if (!d[a].empty()) { d[a].pop_back(); }
    } else if (op == "push_front") {
        int a, x; cin >> a >> x; d[a].push_front(x);
    } else if (op == "pop_front") {
        int a; cin >> a; if (!d[a].empty()) { d[a].pop_front(); }
    } else if (op == "size") {
        int a; cin >> a; cout << d[a].size() << endl;
    } else if (op == "front") {
        int a; cin >> a; if (!d[a].empty()) { cout << d[a].front() << endl; }
    } else if (op == "back") {
        int a; cin >> a; if (!d[a].empty()) { cout << d[a].back() << endl; }
    }
}
```

珠心算测验

题目链接 https://www.luogu.com.cn/problem/P2141

给定 n 个正整数，求其中有多少个数，恰好等于另外两个不同的数之和。

$$n \leq 100$$

珠心算测验

解析

本意是想练习 set，但是发现由于数据规模太小，随便用数组都能解决。也希望大家尝试用 set 完成这道题目。

准备一个 set，做二维循环计算所有的两数之和塞入 set。

之后遍历一次数组，对数组里每个元素，用 set count() 查询 set 里是否有这个元素。有则令答案计数 + 1。

可能思路上比数组直接做简单一点。时间复杂度是 $O(n^2)$ 的，可能也会比直接暴力快一点。

珠心算测验

代码实现

```
set<int> s;
for (int i = 1; i <= n; ++i) {
    for (int j = i + 1; j <= n; ++j) {
        s.insert(a[i] + a[j]);
    }
}
int ans = 0;
for (int i = 1; i <= n; ++i) {
    if (s.count(a[i])) {
        ans++;
    }
}
cout << ans << '\n';
```

于是他错误的点名开始了

题目链接 https://www.luogu.com.cn/problem/P2580

给定学生的人数和名单，你需要判断教练点名的名字是否正确。

输入包括班上人数 n 和学生名单，以及教练点名的名字个数 m 和名字列表。

对于每个名字，如果名字正确且第一次出现，输出 OK；如果名字错误，输出 WRONG；如果名字正确但不是第一次出现，输出 REPEAT。

$$n \leq 10^4, m \leq 10^5$$

于是他错误的点名开始了

解析

想要查询一个字符串出现了多少次。直接使用 map 即可。

map<string, int> 记录某个字符串出现的次数。

初始时读入一个字符串后可以将对应的 map 赋值为 1。后面再判断时，如果 map 对应 0，则报告 WRONG；如果对应 1，则报告 OK；如果对应 2，则报告 REPEAT。

应该没有其他的坑点。

用两个 set 应该也可以。第一个放未点名的，第二个放已点名的。

于是他错误的点名开始了

代码实现

```cpp
for (int i = 1; i <= n; ++i) {
    string s; cin >> s; v[s] = 1;
}
for (int i = 1; i <= m; ++i) {
    string s;
    cin >> s;
    if (!v[s]) {
        cout << "WRONG" << endl;
    } else if (v[s] == 1) {
        ++v[s];
        cout << "OK" << endl;
    } else {
        cout << "REPEAT" << endl;
    }
}
```

合并果子

题目链接 https://www.luogu.com.cn/problem/P1090

给定若干堆果子 $a_1, a_2, \cdots, a_n$。设计一种合并果子的次序方案，使得总耗费的体力最少。

每次合并两堆果子，耗费的体力等于两堆果子的重量之和。目标是通过 $n-1$ 次合并使所有果子合成一堆，并输出最小的体力耗费值。

$$1 \le n \le 10000$$

合并果子

解析

注意到一定是先合并当前重量最小的两堆果子最好。

因为一堆果子对体力耗费值的贡献，只与这堆果子本身的重量和被合并的次数有关。重量大的果子堆合并次数一定越少越好。

在算法竞赛中，贡献指的是某个元素或操作对总结果的独立影响或变化量。

问题转换成：给定一个序列，每次从其中找到两个最小的元素，合并后再丢回去。

找到最小的元素可以使用优先队列完成。

详细介绍代码细节。

合并果子

代码实现

```
vector<int> v;
for (int i = 1; i <= n; ++i) {
    int x; cin >> x;
    v.push_back(x);
}
priority_queue<int> q; // 默认是最大
// range-based for
for (int i : v) { q.push(-i); }

long long ans = 0;
while (q.size() > 1) {
    int x = -q.top(); q.pop();
    int y = -q.top(); q.pop();
    ans += (x + y); q.push(-(x + y));
}
cout << ans << endl;
```

关于 STL set, map, priority_queue 的警示

解析

这三个 STL 在使用时，需要保证内部的数据类型（尖括号内填写的）一定有定义过小于号（bool operator<）。

struct 定义方法：

```
struct node {
    int x;

    // 第一种选择：写这个固定格式的函数，return 你想要比较的结构体内变量
    bool operator<(const node &other) const {
        // 不等号的朝向与写 sort 自定义函数时基本相同
        return x > other.x;
    }
};

// 第二种选择：在结构体外写一个比较函数
bool operator<(node a, node b) {
    return a.x > b.x;
}
```

关于 STL set, map, priority_queue 的警示

解析

对于基本数据类型（int, double 等），无法修改其小于号的定义。

对于 priority_queue<基本数据类型>，如果想修改其为「找到最小元素」，则需要做一定修改。

方式 1：用一个结构体包住基本数据类型；

方法 2：使用

```
priority_queue<类型, vector<类型>, greater<类型>>
```

括号画家

题目链接 https://www.luogu.com.cn/problem/P10472

给定一排长为 n 的括号序列，包括小括号 ()、中括号 [] 和大括号 {}。

需要在这个序列中找出最长的美观子序列。

美观的括号序列需要满足以下条件：

- 空序列是美观的；
- 如果 A 是美观的，则 (A)、[A]、{A} 也是美观的；
- 如果 A 和 B 都是美观的，则 AB 也是美观的。

$$n \leq 10000$$

括号画家

解析

了解一下括号序列的定义。这种定义方式后面会常见。

对于寻找「区间内最长 XXX 序列」的问题，一种常见思路是定死左边界，对每个左边界寻找右边界。

考虑一个二重循环，第一维 `i` 枚举左边界，第二维 `j` 不断向右试探右边界。

过程中使用一个栈来存储当前遇到的所有左括号。`j` 向右试探时，遇到左括号则塞入栈里，遇到右括号则判断当前右括号和栈顶左括号是否相同，相同则弹掉栈顶。

过程中一旦栈变空，则可更新答案。一旦遇到右括号和栈顶不匹配，则直接 `break`。

括号画家

代码实现

```
string s; stack<char> q;
int n = s.length(); int ans = 0;
for (int i = 0; i < n; i++) {
    while (!q.empty())
        q.pop();
    if (s[i] == ')' || s[i] == ']' || s[i] == '}') continue;
    for (int j = i; j < n; j++) {
        if (s[j] == '(' || s[j] == '[' || s[j] == '{') q.push(s[j]);
        else {
            if (q.empty()) break;
            if (s[j] == ')' && q.top() == '(') { q.pop(); }
            else if (s[j] == ']' && q.top() == '[') { q.pop(); }
            else if (s[j] == '}' && q.top() == '{') { q.pop(); }
            else break;
            if (q.empty()) ans = max(ans, j - i + 1);
        }
    }
}
cout << ans << endl;
```

约瑟夫问题

题目链接 https://www.luogu.com.cn/problem/P1996

在一个有 n 个人的圆圈中，从第一个人开始报数，每数到 m 的人就出列，然后从下一个人开始重新报数，直到所有人都出列。

输出每个人出列的顺序。

$$1 \leq n, m \leq 100$$

约瑟夫问题

解析

可以使用队列处理这种转圈问题。

将所有人塞入队列中，当队列非空时做循环：

1. 做 $m-1$ 次操作：将队列中的第一个人弹出，并塞到最后。
2. 将队列中当前的第一个人弹出，并输出，不放到最后。

这种转圈且时刻有可能有元素出局的套路可以考虑用队列完成。队列的特性天然适合这种套路。

约瑟夫问题

代码实现

```
queue<int> q;
int n, m;
for (int i = 1; i <= n; i++)
    q.push(i);
while (!q.empty()) {
    for (int i = 1; i <= m - 1; ++i) {
        q.push(q.front()); q.pop();
    }
    cout << q.front() << ' ';
    q.pop();
}
```

学籍管理

题目链接 https://www.luogu.com.cn/problem/P5266

设计学籍管理系统，支持以下几个操作：插入、修改、查询、删除、汇总。数据条目总数用 n 表示。

$$n \leq 10^5$$

学籍管理

解析

纯正的 map 练习题。

所有操作直接按照 map 来即可。

一个不常用的操作是「删除」，可以熟悉 erase 函数。

学籍管理

代码实现

```
map<string, int> a;
for (int i = 1; i <= q; i++) {
    int op;
    string name;
    cin >> op;
    if (op != 4) cin >> name;
    if (op == 1) {
        int score; cin >> score;
        a[name] = score; cout << "OK\n";
    } else if (op == 2) {
        if (a.count(name)) cout << a[name] << endl;
        else cout << "Not found\n";
    } else if (op == 3) {
        if (a.count(name)) {
            a.erase(name); cout << "Deleted successfully\n";
        } else cout << "Not found\n";
    } else if (op == 4) {
        cout << a.size() << endl;
    }
}
```

海港

题目链接 https://www.luogu.com.cn/problem/P2058

有 n 艘船到达海港，每艘船有到达时间 t_i、乘客数 k_i 以及每名乘客的国籍 $x_{i,j}$。

对于每艘船，需要统计其到达时间前的 24 小时内（86400 秒内），所有船只上乘客的国籍种类数。

$$1 \le n \le 10^5, \sum k_i \le 3 \times 10^5, 1 \le x_{i,j} \le 10^5, 1 \le t_i \le 10^9$$

海港

题目链接 https://www.luogu.com.cn/problem/P2058

给若干个结构体 struct node {int t; vector<int> x;} a[XX]。

按照 t 由小到大排序，对每个 a[i].t，计算：所有 t 在 a[i].t - 86400 ~ a[i].t 范围内的结构体的 x[i]，有多少种。

海港

解析

考虑能否使用一个 STL 容器，去记录有多少结构体在这个范围内。

这个容器需要支持：

- 放入 t 大一些的结构体；
- 把 t 小一些的结构体挪出去；

很显然队列天然支持这些特性。

至于 x 的统计，可以考虑下用桶 cnt 计数。$cnt[i]$ 记录某个数在队列里的所有结构体里出现了多少次。

使用 STL 容器时也不必拘泥于只在 STL 里存全部信息。有可能可以与其他信息联动，STL 里存一部分，其他的存一部分。

海港

代码实现

```
struct node { int t; vector<int> x; } a[100005];
map<int, int> mp; int cnt; queue<int> q;
for (int i = 1; i <= n; ++i) {
    int k;
    cin >> a[i].t >> k;
    for (int j = 1; j <= k; ++j) {
        int x; cin >> x; a[i].x.push_back(x);
    }
}
for (int i = 1; i <= n; ++i) {
    q.push(i);
    for (int i : a[i].x) { if (mp[i] == 0) { cnt++; } mp[i]++; }
    while (!q.empty()) {
        int x = q.front();
        if (a[x].t <= a[i].t - 86400) {
            q.pop();
            for (int i : a[x].x) { if (mp[i] == 1) { cnt--; } mp[i]--; }
        } else break;
    }
    cout << cnt << '\n';
}
```