算法竞赛 STL 速成讲义

string · vector · map · set · unordered_map · lambda 排序

ICPC 省赛备考资料

g++ -std=c++17 -O2 -Wall

2026-11

目录

0. 讲义说明
- 0.1 适用对象
- 0.2 内容范围
- 0.3 约定
- 0.4 阅读顺序

1. 基础
- 1.1 万用头文件与命名空间
- 1.2 迭代器与范围 for
- 1.3 复杂度与数据规模
- 1.4 通用约束

2. string
- 2.1 定义与性质
- 2.2 接口
- 2.3 模板程序
- 2.4 例题
- 2.5 注意

3. vector
- 3.1 定义与性质
- 3.2 接口
- 3.3 模板程序
- 3.4 排序、去重与二分
- 3.5 例题
- 3.6 注意

4. map：键值映射容器
- 4.1 定义与性质
- 4.2 接口
- 4.3 模板程序
- 4.4 例题
- 4.5 注意

5. set：有序唯一集合
- 5.1 定义与性质
- 5.2 接口
- 5.3 模板程序
- 5.4 例题
- 5.5 注意

6. unordered_map / unordered_set：哈希表关联容器
- 6.1 定义与性质
- 6.2 接口
- 6.3 模板程序
- 6.4 卡哈希与 reserve
- 6.5 例题
- 6.6 注意

7. lambda 与自定义排序
- 7.1 lambda 语法
- 7.2 三种比较器模板
- 7.3 捕获列表
- 7.4 set 与 priority_queue 的比较器
- 7.5 例题
- 7.6 注意

8. 综合例题
- 8.1 vector 模拟
- 8.2 string 与 set
- 8.3 map 计数与 lambda 排序

9. 速查总表
- 9.1 六容器常用操作
- 9.2 算法函数
- 9.3 选型

10. 常见错误

11. 练习
- 11.1 必做（25 题）
- 11.2 提高（10 题）
- 11.3 拓展（5 题）

12. 配套例题（ICPC 省赛）
- 12.1 字符串（第 2 章）
- 12.2 vector 与二分（第 3 章）
- 12.3 map（第 4 章）
- 12.4 set（第 5 章）
- 12.5 unordered_map（第 6 章）
- 12.6 lambda 自定义排序（第 7 章）
- 12.7 平台题单
- 12.8 训练安排

算法竞赛 STL 速成讲义

> 适用对象：已掌握 C++ 基本语法（变量、分支、循环、数组、函数）的参赛者。全文代码使用 g++ 与 `<bits/stdc++.h>`，样例均可直接编译运行。

0. 讲义说明

0.1 适用对象

已掌握 C++ 基本语法（变量、分支、循环、数组、函数），需要在短期内掌握算法竞赛常用 STL 容器的读者。

0.2 内容范围

本讲义覆盖六个组件：`string`、`vector`、`map`、`set`、`unordered_map` / `unordered_set`、lambda 自定义排序。每个组件给出完整接口表、可编译模板与例题；第 9 章为速查总表，第 11 章为分级练习，第 12 章为 ICPC 省赛配套例题。

0.3 约定

- 编译命令：`g++ -std=c++17 -O2 -Wall`；评测机为 Linux + g++。
- 统一使用万用头文件 `#include <bits/stdc++.h>` 与 `using namespace std;`。
- 复杂度记号：`O(1)`、`O(log n)`、`O(n)`、`O(n log n)`；`n` 表示容器元素个数。
- 接口表中 ⭐ 表示竞赛常用；未标记者为了解即可。
- 代码风格：左大括号独占一行，4 空格缩进，一行一条语句；`vector<int> v = {1, 2, 3};` 这类初始化列表保持行内。
- 所有程序均为完整可编译程序，可直接提交。

0.4 阅读顺序

1. 第 1 章：STL 代码的三项基础（万用头文件、迭代器、复杂度）。
2. 第 2~7 章：按容器顺序阅读，接口表优先，例题次之。
3. 第 8 章：跨容器综合题。
4. 第 9~10 章：速查与错误清单，做题时回查。
5. 第 11 章：练习。
6. 第 12 章：ICPC 省赛配套例题与训练安排。

---

1. 基础

1.1 万用头文件与命名空间

竞赛代码通常以 `#include <bits/stdc++.h>` 与 `using namespace std;` 两行开始。`<bits/stdc++.h>` 为 GCC 扩展头文件，一次性包含全部标准库；`using namespace std;` 使 `std::` 前缀可省略。二者缺一即出现 `'vector' was not declared in this scope` 一类编译错误。该头文件非 C++ 标准，仅适用于 g++/libstdc++ 环境。

1.2 迭代器与范围 for

容器通过迭代器给出元素位置：`begin()` 指向首元素，`end()` 指向末元素之后的位置，区间为左闭右开 `[begin, end)`；`end()` 不可解引用。对迭代器解引用 `*it` 得到元素，`it++` 后移一位。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    vector<int> v = {10, 20, 30, 40};

    for (auto it = v.begin(); it != v.end(); it++)
    {
        cout << *it << ' ';
    }
    cout << endl;

    for (int i = 0; i < (int)v.size(); i++)
    {
        v[i] *= 2;
    }
    for (int i = 0; i < (int)v.size(); i++)
    {
        cout << v[i] << ' ';
    }
    cout << endl;
    return 0;
}
```

输入：（无）

输出：

```
10 20 30 40
20 40 60 80
```

上例第二个循环采用下标形式；其等价简写为范围 for `for (T &x : c)`，`&` 表示引用，可修改元素，写 `for (T x : c)` 仅读取副本。只读遍历时可用 `const T &` 避免拷贝。本讲义后续统一使用下标循环；第 4~6 章的关联容器无下标，使用迭代器或范围 for 遍历。

1.3 复杂度与数据规模

经验值：1 秒约可执行 10^8 次基本运算。

数据规模 n	O(n)	O(n log n)	O(n^2)
10^3	可行	可行	可行
10^4	可行	可行	边界
10^5	可行	可行	不可行
10^6	可行	边界	不可行

容器单次操作复杂度：

操作	`vector` / `string`	`map` / `set`	`unordered_map` / `unordered_set`
按下标/键访问	`O(1)`（下标）	不支持	不支持
查找	`O(n)`	`O(log n)`	平均 `O(1)`
插入/删除	末尾 `O(1)` 均摊，中间 `O(n)`	`O(log n)`	平均 `O(1)`
遍历	`O(n)`	`O(n)`，有序	`O(n)`，无序

1.4 通用约束

1. `operator[]` 不检查越界，合法下标为 `[0, size - 1]`。
2. `size()` 返回无符号类型；下标运算先写 `int n = (int)c.size();`。
3. `sort`、`lower_bound`、`unique` 等算法函数以迭代器区间为参数；`unique` 仅去除相邻重复，去重前须排序。
4. 遍历过程中删除元素会使迭代器/下标失效，需按各容器的接口约定处理。

---

2. string

2.1 定义与性质

`std::string` 是字符序列容器，元素类型为 `char`，元素连续存放，合法下标区间为 `[0, size())`。长度由容器维护，与声明时的容量无关；与 `char[]` 相比支持赋值、拼接、字典序比较与子串查找，代价是每次操作附带容量维护开销。

项目	性质
存储	连续内存，`&s[0]` 可作 `char *` 使用
长度	`size()` 为 `O(1)`，类型为无符号 `size_t`
尾部操作	`push_back`、`pop_back`、`+=` 均摊 `O(1)`
中间操作	`insert`、`erase` 为 `O(n)`
未定义行为	越界 `s[i]`；空串的 `front()`、`back()`、`pop_back()`

2.2 接口

（1）构造、赋值与容量

函数 / 写法	作用	复杂度	备注
`string s;`	空串	`O(1)`	长度 0 ⭐
`string s = "abc";`	字面量初始化	`O(n)`	亦写作 `string s("abc")` ⭐
`string s(n, c)`	`n` 个字符 `c`	`O(n)`	⭐
`string s(t)`	复制构造	`O(n)`	深拷贝 ⭐
`string s(t, pos, len)`	取 `t` 的子串构造	`O(len)`	`pos` 越界抛 `out_of_range`
`string s(first, last)`	迭代器区间构造	`O(n)`	半开区间
`s = t;`	整体赋值	`O(n)`	⭐
`s.assign(t)`	整体赋值	`O(n)`	另有 `s.assign(n, c)`，竞赛少用
`s += t;`	尾部追加	均摊 `O(size(t))`	⭐
`s.append(t)`	尾部追加	均摊 `O(size(t))`	另有 `s.append(n, c)`；`+=` 的函数版
`s.size()` / `s.length()`	字符数	`O(1)`	返回 `size_t` ⭐
`s.empty()`	是否为空	`O(1)`	⭐
`s.resize(n)`	改长度	`O(n)`	变长部分补 `'\0'` ⭐
`s.resize(n, c)`	改长度	`O(n)`	变长部分补 `c`
`s.reserve(n)`	预留容量	`O(n)`	不改变长度
`s.capacity()`	当前容量	`O(1)`	不小于 `size()`
`s.shrink_to_fit()`	释放多余容量	`O(n)`	C++11 起
`s.max_size()`	理论最大长度	`O(1)`	竞赛不用

（2）元素访问与迭代器

函数 / 写法	作用	复杂度	备注
`s[i]`	第 `i` 个字符	`O(1)`	不检查越界 ⭐
`s.at(i)`	第 `i` 个字符	`O(1)`	越界抛 `out_of_range`
`s.front()`	首字符	`O(1)`	空串为未定义行为
`s.back()`	末字符	`O(1)`	等价 `s[s.size() - 1]` ⭐
`s.data()`	首字符指针	`O(1)`	等同 `&s[0]`

函数 / 写法	作用	复杂度	备注
`s.c_str()`	C 风格字符串	`O(1)`	传给 C 函数时使用 ⭐
`s.begin()` / `s.end()`	首、尾后迭代器	`O(1)`	半开区间 `[begin, end)` ⭐
`s.rbegin()` / `s.rend()`	反向迭代器	`O(1)`	⭐
`s.cbegin()` / `s.cend()`	只读迭代器	`O(1)`	C++11 起，竞赛少用
`s.crbegin()` / `s.crend()`	只读反向迭代器	`O(1)`	竞赛少用

(3) 修改

函数 / 写法	作用	复杂度	备注
`s.push_back(c)`	尾部加一个字符	均摊 `O(1)`	⭐
`s.pop_back()`	删除末字符	`O(1)`	空串为未定义行为 ⭐
`s.insert(pos, t)`	下标 `pos` 处插入	`O(n)`	⭐
`s.insert(pos, n, c)`	插入 `n` 个 `c`	`O(n)`	竞赛少用
`s.erase(pos, len)`	从 `pos` 删 `len` 个	`O(n)`	省略 `len` 则删至末尾 ⭐
`s.replace(pos, len, t)`	替换 `[pos, pos + len)`	`O(n)`	第二个参数是长度 ⭐
`s.swap(t)`	交换内容	`O(1)`	亦可写 `swap(s, t)` ⭐
`s.clear()`	清空	`O(n)`	⭐
`s.insert(it, c)`、`s.erase(it)`、`s.erase(first, last)`、`s.replace(first, last, t)`	迭代器定位的插入、删除、替换	`O(n)`	参数与返回值均为迭代器，竞赛少用

(4) 查找、子串与比较（查找失败均返回 `string::npos`）

函数 / 写法	作用	复杂度	备注
`s.find(t)`	首次出现位置	`O(n * m)`	⭐
`s.find(t, pos)`	从 `pos` 起查找	`O(n * m)`	枚举全部出现时使用 ⭐
`s.rfind(t)`	最后一次出现位置	`O(n * m)`	⭐
`s.find_first_of(t)` / `s.find_last_of(t)`	首次、最后一次出现 `t` 中任一字符	`O(n * m)`	参数为字符集合
`s.find_first_not_of(t)` / `s.find_last_not_of(t)`	首个、最后不属于 `t` 的字符	`O(n * m)`	跳过前导字符、去行尾空格常用
`s.substr(pos, len)` / `s.substr(pos)`	从 `pos` 截 `len` 个，省略 `len` 则截到末尾	`O(len)`	⭐
`s.compare(t)`	字典序比较	`O(n)`	返回正数、零或负数 ⭐

(5) 数值转换与非成员函数

函数 / 写法	作用	复杂度	备注
`stoi(s)`	转 `int`	`O(n)`	非法或溢出抛异常 ⭐
`stoi(s, &pos)`	转 `int`，`pos` 返回解析终止位置	`O(n)`	配合 `substr` 切分文本 ⭐
`stol(s)`	转 `long`	`O(n)`	64 位下与 `long long` 同宽
`stoll(s)`	转 `long long`	`O(n)`	大整数使用 ⭐
`stoul(s)` / `stoull(s)`	转 `unsigned long`、`unsigned long long`	`O(n)`	竞赛少用
`stof(s)`	转 `float`	`O(n)`	精度低，少用

函数 / 写法	作用	复杂度	备注
`stod(s)`	转 `double`	`O(n)`	⭐
`to_string(x)`	数值转字符串	`O(n)`	浮点保留 6 位小数 ⭐
`getline(cin, s)`	读入整行	`O(n)`	保留空格 ⭐
`operator+`	拼接为新串	`O(n + m)`	每次构造新对象 ⭐
`==`、`!=`、`<`、`>`、`<=`、`>=`	字典序比较	`O(min(n, m))`	前缀较短者更小 ⭐
`swap(a, b)`	交换	`O(1)`	⭐
`cin >> s` / `cout << s`	读入 / 输出	`O(n)`	读入遇空白停止 ⭐

2.3 模板程序

以下程序读入一行文本，统计其中小写元音字母的个数。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    getline(cin, s);
    int cnt = 0;
    for (int i = 0; i < (int)s.size(); i++)
    {
        char c = s[i];
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u')
        {
            cnt++;
        }
    }
    cout << cnt << endl;
    return 0;
}
```

输入：

```
hello world
```

输出：

```
3
```

2.4 例题

例题 2-1：可重叠子串计数

题述：给定主串 `s` 与模式串 `p`，统计 `p` 在 `s` 中出现的次数，允许重叠。

输入输出：第 1 行为 `s`，第 2 行为 `p`，均只含小写字母；输出一个整数表示出现次数。

思路：`find` 从 `pos` 起查找，命中后答案加一，下一轮从 `pos + 1` 继续，使重叠情况被计入；返回 `npos` 时结束。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string s, p;
    cin >> s >> p;
    int ans = 0;
    size_t pos = s.find(p);
    while (pos != string::npos)
    {
        ans++;
        pos = s.find(p, pos + 1);
    }
    cout << ans << endl;
    return 0;
}
```

输入：

```
aaaa
aa
```

输出：

```
3
```

复杂度：`O(n * m)`，`n`、`m` 分别为主串与模式串长度。

例题 2-2：字符串压缩

题述：将连续相同字符合并为“字符 + 个数”，输出压缩结果。

输入输出：一行只含小写字母的字符串；输出压缩后的字符串。

思路：双下标圈定每一段：`i` 为段首，`j` 右移至首个不同字符，该段长度为 `j - i`；拼接字符与 `to_string(j - i)` 后令 `i = j`。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    cin >> s;
    string ans;
    int i = 0;
    while (i < (int)s.size())
    {
        int j = i;
        while (j < (int)s.size() && s[j] == s[i])
        {
            j++;
        }
        ans += s[i];
        ans += to_string(j - i);
        i = j;
    }
    cout << ans << endl;
    return 0;
}
```

输入：

```
aaabbc
```

输出：

```
a3b2c1
```

复杂度：`O(n)`，`n` 为字符串长度。

2.5 注意

1. `s.size()` 为无符号类型，与 `int` 比较时统一转 `(int)`；空串上计算 `s.size() - 1` 会得到极大值。
2. `find` 系列失败时返回 `string::npos`，判断写 `pos == string::npos`，不得与 `-1` 比较。
3. `substr(pos, len)` 与 `replace(pos, len, t)` 的第二个参数是长度，不是结束下标。
4. `s[i]`、`front()`、`back()` 不做越界检查；`at(i)` 检查并抛异常，竞赛中不依赖异常处理。
5. `unique` 仅去除相邻重复，去重前须先 `sort`；`insert`、`erase` 单次 `O(n)`，避免在循环中反复调用。

---

3. vector

3.1 定义与性质

`vector<T>` 为动态数组容器：元素连续存放，长度在运行时确定并可增长，合法下标范围恒为 `[0, size())`，随机访问 `v[i]` 为 `O(1)`。

- **长度与容量分离**：`size()` 为元素个数，`capacity()` 为已分配空间可容纳的元素个数，恒有 `size() <= capacity()`。
- **尾部增删均摊** `O(1)`：`size() == capacity()` 时 `push_back` 触发重新分配（容量通常成倍增长）并搬迁元素。
- **中部插删** `O(n)`：`insert`、`erase` 之后元素整体移动。
- **预分配**：`reserve(n)` 仅扩大 `capacity()`，不改变 `size()`，用于避免重复扩容。

3.2 接口

构造与赋值

函数 / 写法	作用	复杂度	备注
`vector<int> v;`	空容器	`O(1)`	⭐
`vector<int> v(n);`	`n` 个值初始化元素（`int` 为 0）	`O(n)`	⭐
`vector<int> v(n, x);`	`n` 个值为 `x` 的元素	`O(n)`	⭐
`vector<int> v = {1, 2, 3};`	初始化列表构造	`O(n)`	⭐
`vector<int> v(other);`	拷贝构造	`O(n)`	少用
`vector<int> v(first, last);`	区间 `[first, last)` 构造	`O(n)`	少用
`vector<vector<int>> g(n, vector<int>(m, 0));`	`n` 行 `m` 列二维表	`O(nm)`	⭐
`v = other;`	拷贝赋值，长度随之改变	`O(n)`	⭐
`v = {1, 2, 3};`	列表赋值	`O(n)`	⭐
`v.assign(n, x);`	置为 `n` 个 `x`	`O(n)`	⭐
`v.assign(first, last);`	置为区间内容	`O(n)`	少用

容量、访问与迭代器

函数 / 写法	作用	复杂度	备注
`v.empty()`	是否为空，返回 `bool`	`O(1)`	⭐
`v.size()`	元素个数，返回无符号整数	`O(1)`	⭐
`v.max_size()`	可容纳元素的理论上限	`O(1)`	少用
`v.capacity()`	已分配容量	`O(1)`	少用
`v.reserve(n)`	预留容量，不改变 `size()`	`O(n)`	⭐
`v.shrink_to_fit()`	请求将 `capacity()` 收缩至 `size()`	`O(n)`	少用
`v.resize(n)`	长度改为 `n`，新增元素值初始化	`O(n)`	⭐
`v.resize(n, x)`	长度改为 `n`，新增元素值为 `x`	`O(n)`	⭐
`v[i]`	第 `i` 个元素，不检查越界	`O(1)`	⭐
`v.at(i)`	第 `i` 个元素，越界抛出异常	`O(1)`	少用
`v.front()` / `v.back()`	首元素 / 末元素	`O(1)`	⭐
`v.data()`	连续内存首地址	`O(1)`	少用
`v.begin()` / `v.end()`	首元素位置 / 末元素后一位置	`O(1)`	⭐
`v.rbegin()` / `v.rend()`	反向首位置 / 反向末位置	`O(1)`	少用
`v.cbegin()` / `v.cend()`	只读首位置 / 只读末位置	`O(1)`	少用

函数 / 写法	作用	复杂度	备注
`v.crbegin()` / `v.crend()`	只读反向首位置 / 只读反向末位置	`O(1)`	少用

修改

函数 / 写法	作用	复杂度	备注
`v.push_back(x)`	尾部插入 `x`	均摊 `O(1)`	⭐
`v.emplace_back(x)`	尾部原地构造	均摊 `O(1)`	⭐
`v.pop_back()`	删除末元素	`O(1)`	⭐
`v.insert(pos, x)` / `v.insert(pos, n, x)` / `v.insert(pos, first, last)`	迭代器 `pos` 前插入元素、`n` 个元素或区间	`O(n)`	⭐
`v.emplace(pos, x)`	`pos` 前原地构造	`O(n)`	少用
`v.erase(pos)`	删除 `pos` 处元素	`O(n)`	⭐
`v.erase(first, last)`	删除区间 `[first, last)`	`O(n)`	⭐
`v.clear()`	清空元素，`capacity()` 不变	`O(n)`	⭐
`v.swap(other)`	交换两容器内容	`O(1)`	⭐
`v.get_allocator()`	返回分配器	`O(1)`	少用

非成员

函数 / 写法	作用	复杂度	备注
`swap(a, b)`	交换两容器内容	`O(1)`	⭐
`a == b` / `a != b` / `a < b`	逐元素判等 / 字典序比较	`O(n)`	少用

3.3 模板程序

读入 `n` 个整数存入 `vector`，输出元素个数、总和、首元素与末元素。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> a;
    int sum = 0;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        a.push_back(x);
        sum += x;
    }
    cout << (int)a.size() << ' ' << sum << ' ' << a.front() << ' ' << a.back() << endl;
    return 0;
}
```

输入：

```
4
1 2 3 4
```

输出：

```
4 10 1 4
```

3.4 排序、去重与二分

函数 / 写法	作用	复杂度	备注
`sort(first, last)`	升序排序	`O(n log n)`	⭐
`sort(first, last, greater<int>())`	降序排序	`O(n log n)`	⭐
`reverse(first, last)`	反转区间	`O(n)`	⭐
`unique(first, last)`	去除相邻重复，返回去重区间末尾	`O(n)`	⭐
`lower_bound(first, last, x)`	首个大于等于 `x` 的位置	`O(log n)`	⭐
`upper_bound(first, last, x)`	首个大于 `x` 的位置	`O(log n)`	⭐
`binary_search(first, last, x)`	`x` 是否存在，返回 `bool`	`O(log n)`	⭐

去重写法为 `sort` 后执行 `v.erase(unique(v.begin(), v.end()), v.end())`；`unique` 只消除相邻重复且不改变 `size()`。`lower_bound`、`upper_bound` 返回迭代器，转下标需减去 `v.begin()`，二者之差为 `x` 的出现次数。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    sort(a.begin(), a.end(), greater<int>());
    a.erase(unique(a.begin(), a.end()), a.end());
    reverse(a.begin(), a.end());
    int x;
    cin >> x;
    int lo = (int)(lower_bound(a.begin(), a.end(), x) - a.begin());
    int hi = (int)(upper_bound(a.begin(), a.end(), x) - a.begin());
    bool ok = binary_search(a.begin(), a.end(), x);
    cout << (int)a.size();
    for (int i = 0; i < (int)a.size(); i++)
    {
        cout << ' ' << a[i];
    }
    cout << ' ' << lo << ' ' << hi << ' ' << ok << endl;
    return 0;
}
```

输入：

```
7
3 1 3 2 1 8 2
2
```

输出：

```
4 1 2 3 8 1 2 1
```

3.5 例题

例题 3-1：去重后第 k 小元素

给定 `n` 个整数与整数 `k`，求去重并按升序排列后的第 `k` 个元素；不同元素不足 `k` 个时输出 `-1`。 输入格式：第一行 `n`、`k`（`1 <= k <= n <= 2 * 10^5`）；第二行 `n` 个整数。输出格式：一行，第 `k` 小元素或 `-1`。

思路：排序后相同元素相邻，`unique` 与 `erase` 完成去重并保持升序；去重结果第 `k` 个元素的下标为 `k - 1`，长度不足时为无解。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, k;
    cin >> n >> k;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());
    cout << (k > (int)a.size() ? -1 : a[k - 1]) << endl;
    return 0;
}
```

输入：

```
6 3
5 1 5 2 1 4
```

输出：

```
4
```

复杂度：`O(n log n)`，由排序决定。

3.6 注意

1. `v.size()` 返回无符号整数，与有符号整数比较须显式转换；`v.size() - 1` 在空容器上回绕为极大值。
2. `v[i]` 不检查越界；空容器上调用 `v[0]`、`v.front()`、`v.back()`、`v.pop_back()` 属未定义行为。
3. `push_back`、`insert`、`erase` 之后既有迭代器可能失效；容量变化时全部迭代器与引用失效。
4. `unique` 仅去除相邻重复且不改变 `size()`，使用前必须排序，并与 `erase` 配合。
5. `lower_bound`、`upper_bound`、`binary_search` 要求区间有序；`vector<bool>` 为特化实现，元素不具常规引用语义，位序列改用 `vector<char>`。

---

4. map：键值映射容器

4.1 定义与性质

`map` 是按键（key）有序存储键值对（key → value）的关联容器，每个 key 至多对应一个 value。

- **key 唯一**：重复插入同一 key 不产生新元素。
- **按 key 升序**：遍历顺序由 key 的比较结果决定，与插入顺序无关；字符串 key 按字典序。
- **红黑树**：插入、删除、查找、前驱与后继查询均为 `O(log n)`。
- **元素类型**：`pair<const K, V>`，`first` 为 key（只读），`second` 为 value（可写）。

约定：键值对以 `make_pair(k, v)` 构造；代码统一使用 `bits/stdc++.h` 与 `using namespace std;`。

4.2 接口

`K` 为 key 类型，`V` 为 value 类型，`n` 为元素个数；⭐ 表示常用。注意 `m[key]` 在 key 不存在时先插入值初始化的元素再返回引用，具有插入副作用，仅判断存在性应使用 `count` 或 `find`。

函数 / 写法	作用	复杂度	备注
构造与赋值			
`map<K, V> m;`	空容器	`O(1)`	⭐
`map<K, V> m = other;`	拷贝构造	`O(n)`	
`map<K, V> m(first, last);`	迭代器区间构造	`O(n log n)`	少用
`map<K, V> m = {{k, v}, {k2, v2}};`	初始化列表	`O(n log n)`	C++11
`m = other;`	赋值	`O(n)`	先清空
`m.get_allocator()`	取分配器	`O(1)`	少用
元素访问			
`m[key]`	返回 value 引用；key 不存在时插入值初始化元素	`O(log n)`	⭐ 有插入副作用
`m.at(key)`	返回 value 引用；不存在时抛出 `out_of_range`	`O(log n)`	少用
迭代器			
`m.begin()` / `m.end()`	指向最小 key / 尾后位置	`O(1)`	⭐
`m.rbegin()` / `m.rend()`	反向迭代器，自最大 key 起	`O(1)`	⭐
`m.cbegin()` / `m.cend()`	只读迭代器	`O(1)`	
`m.crbegin()` / `m.crend()`	只读反向迭代器	`O(1)`	
容量			
`m.empty()`	是否为空	`O(1)`	⭐
`m.size()`	元素个数	`O(1)`	⭐
`m.max_size()`	容量上限	`O(1)`	少用
修改			
`m.clear()`	清空	`O(n)`	⭐
`m.insert(make_pair(k, v))`	插入；key 已存在时不覆盖	`O(log n)`	返回 `pair<it, bool>` ⭐
`m.insert(first, last)`	插入区间内全部键值对	`O(k log n)`	少用
`m.insert(hint, value)`	带位置提示的插入	平均 `O(1)`	少用
`m.emplace(k, v)`	原地构造并插入；已存在则不插	`O(log n)`	C++11 ⭐
`m.emplace_hint(hint, k, v)`	带提示的 `emplace`	平均 `O(1)`	少用
`m.try_emplace(k, v)`	key 不存在时才构造 value	`O(log n)`	C++17

函数 / 写法	作用	复杂度	备注
`m.insert_or_assign(k, v)`	有则覆盖，无则插入	`O(log n)`	C++17 ⭐
`m.erase(key)`	按 key 删除，返回删除个数	`O(log n)`	⭐
`m.erase(it)`	按迭代器删除，返回后继迭代器	均摊 `O(1)`	`it` 失效 ⭐
`m.erase(first, last)`	删除区间 `[first, last)`	`O(log n + k)`	少用
`m.swap(other)`	交换两个容器	`O(1)`	
`m.extract(key)`	摘出节点（不销毁）	`O(log n)`	C++17，少用
`m.merge(other)`	迁移另一容器的节点	`O(k log n)`	C++17，已存在者不覆盖
查找			
`m.count(key)`	key 的个数（0 或 1）	`O(log n)`	⭐
`m.find(key)`	指向该键值对的迭代器；未命中返回 `end()`	`O(log n)`	⭐
`m.contains(key)`	key 是否存在	`O(log n)`	C++20，本讲义不可用
`m.lower_bound(key)`	第一个 key `>= key` 的迭代器	`O(log n)`	⭐
`m.upper_bound(key)`	第一个 key `> key` 的迭代器	`O(log n)`	⭐
`m.equal_range(key)`	返回 `[lower_bound, upper_bound)` 迭代器对	`O(log n)`	
观察器			
`m.key_comp()`	取 key 比较器	`O(1)`	少用
`m.value_comp()`	取键值对比较器	`O(1)`	少用
非成员函数			
`swap(a, b)`	交换两个容器	`O(1)`	⭐
`a == b` / `a != b` / `a < b`	按字典序比较整个容器	`O(n)`	
`erase_if(m, pred)`	按条件批量删除	`O(n)`	C++20，本讲义不可用

4.3 模板程序

读入 `n` 组字符串与整数，按键累加同一字符串对应的数值，按键升序输出，并查询指定键。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    map<string, int> cnt;
    for (int i = 0; i < n; i++)
    {
        string k;
        int x;
        cin >> k >> x;
        cnt[k] += x;              // k 不存在时先插入 0
    }
    cout << cnt.size() << endl;
    for (auto &p : cnt)                 // 按键升序
    {
        cout << p.first << ' ' << p.second << endl;
    }
    auto it = cnt.find("banana");
    if (it != cnt.end())
    {
        cout << it->first << ' ' << it->second << endl;
    }
    return 0;
}
```

输入：

```
4
apple 3
banana 5
apple 2
cherry 1
```

输出：

```
3
apple 5
banana 5
cherry 1
banana 5
```

4.4 例题

例题 4-1：离散化

给定 `n` 个整数（绝对值不超过 `10^9`），将每个数替换为它在这些数中的升序名次（数值相同则名次相同，从 1 开始），按原顺序输出。

输入格式：第一行一个整数 `n`（`1 <= n <= 100000`），第二行 `n` 个整数。

输出格式：一行 `n` 个整数，表示各数对应的名次。

思路：以数值为 key 建立 `map`；`map` 的升序遍历即去重升序序列，对其依次编号，再回查原数组。

```
#include <bits/stdc++.h>
using namespace std;

const int N = 100005;

int a[N];

int main()
{
    int n;
    cin >> n;
    map<int, int> rk;
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
        rk[a[i]] = 0;                   // 只要求 key 出现
    }
    int k = 0;
    for (auto &p : rk)                  // 升序编号
    {
        p.second = ++k;
    }
    for (int i = 0; i < n; i++)
    {
        cout << rk[a[i]];
        if (i + 1 == n)
        {
            cout << endl;
        }
        else
        {
            cout << ' ';
        }
    }
    return 0;
}
```

输入：

```
5
10 50 10 -3 50
```

输出：

```
2 3 2 1 3
```

复杂度：`O(n log n)`，空间 `O(n)`。

4.5 注意

1. `m[key]` 具有插入副作用：key 不存在时先插入值初始化元素；判断存在性使用 `count` 或 `find`。
2. `insert` 在 key 已存在时不覆盖，返回 `pair<iterator, bool>`；需要覆盖时使用 `operator[]` 或 `insert_or_assign`（C++17）。
3. `find` 未命中返回 `end()`，`end()` 不可解引用；取值前须判断 `it != m.end()`。
4. `erase(it)` 使 `it` 失效，其返回值为后继迭代器；边遍历边删除写作 `it = m.erase(it);`。
5. 单次操作 `O(log n)` 且常数较大；值域较小时以数组代替可降低常数。

---

5. set：有序唯一集合

5.1 定义与性质

`set` 是存储唯一元素并按升序排列的关联容器，与仅保留 key 的 `map` 等价；`multiset` 是其允许重复元素的同构容器。

- **元素唯一**：重复插入被忽略，`multiset` 中则插入。
- **升序排列**：遍历顺序由元素比较结果决定，与插入顺序无关。
- **红黑树**：插入、删除、查找、前驱与后继查询均为 `O(log n)`。
- **迭代器只读**：`*it` 不可赋值；无下标访问，最小与最大元素为 `*begin()` 与 `*rbegin()`。

5.2 接口

`T` 为元素类型，`n` 为元素个数；⭐ 表示常用。

函数 / 写法	作用	复杂度	备注
构造与赋值			
`set<T> s;`	空容器	`O(1)`	⭐
`set<T> s = other;`	拷贝构造	`O(n)`	
`set<T> s(first, last);`	区间构造，自动去重排序	`O(n log n)`	⭐
`set<T> s = {3, 1, 5};`	初始化列表	`O(n log n)`	C++11
`s = other;`	赋值	`O(n)`	先清空
`s.get_allocator()`	取分配器	`O(1)`	少用
迭代器			
`s.begin()` / `s.end()`	指向最小元素 / 尾后位置	`O(1)`	⭐
`s.rbegin()` / `s.rend()`	反向迭代器，自最大元素起	`O(1)`	⭐
`s.cbegin()` / `s.cend()`	只读迭代器	`O(1)`	
`s.crbegin()` / `s.crend()`	只读反向迭代器	`O(1)`	
容量			
`s.empty()`	是否为空	`O(1)`	⭐
`s.size()`	元素个数	`O(1)`	⭐
`s.max_size()`	容量上限	`O(1)`	少用
修改			
`s.clear()`	清空	`O(n)`	⭐
`s.insert(x)`	插入；已存在则不插	`O(log n)`	返回 `pair<it, bool>` ⭐
`s.insert(first, last)`	插入区间内全部元素	`O(k log n)`	少用
`s.insert(hint, x)`	带位置提示的插入	平均 `O(1)`	少用
`s.emplace(x)`	原地构造并插入	`O(log n)`	C++11 ⭐
`s.emplace_hint(hint, x)`	带提示的 `emplace`	平均 `O(1)`	少用
`s.erase(x)`	按值删除全部等于 x 的元素	`O(log n + k)`	`multiset` 需注意 ⭐
`s.erase(it)`	按迭代器删除，仅删该元素	均摊 `O(1)`	删一个用 `erase(find(x))` ⭐
`s.erase(first, last)`	删除区间 `[first, last)`	`O(log n + k)`	少用
`s.swap(other)`	交换两个容器	`O(1)`	
`s.extract(x)`	摘出节点（不销毁）	`O(log n)`	C++17，少用
`s.merge(other)`	迁移另一容器的节点	`O(k log n)`	C++17，少用

函数 / 写法	作用	复杂度	备注
查找			
`s.count(x)`	x 的个数（`set` 中 0 或 1）	`O(log n)`	⭐
`s.find(x)`	指向 x 的迭代器；未命中返回 `end()`	`O(log n)`	⭐
`s.contains(x)`	x 是否存在	`O(log n)`	C++20，本讲义不可用
`s.lower_bound(x)`	第一个 `>= x` 的迭代器	`O(log n)`	⭐
`s.upper_bound(x)`	第一个 `> x` 的迭代器	`O(log n)`	⭐
`s.equal_range(x)`	返回 `[lower_bound, upper_bound)` 迭代器对	`O(log n)`	
观察器			
`s.key_comp()`	取比较器	`O(1)`	少用
`s.value_comp()`	同 `key_comp`	`O(1)`	少用
非成员函数			
`swap(a, b)`	交换两个容器	`O(1)`	⭐
`a == b` / `a != b` / `a < b`	按字典序比较整个容器	`O(n)`	
`erase_if(s, pred)`	按条件批量删除	`O(n)`	C++20，本讲义不可用

`multiset` 的接口与 `set` 相同，差异仅在重复元素：`erase(x)` 删除全部等于 `x` 的元素，仅删除一个写作 `erase(find(x))`。

5.3 模板程序

读入 `n` 个整数插入集合，输出去重后的元素个数与升序序列，并查询指定元素的出现次数。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    set<int> s;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        s.insert(x);                    // 重复元素不插入
    }
    cout << s.size() << endl;
    for (int x : s)                     // 升序遍历
    {
        cout << x << endl;
    }
    cout << s.count(5) << endl;
    return 0;
}
```

输入：

```
5
5 1 5 3 1
```

输出：

```
3
1
3
5
1
```

5.4 例题

例题 5-1：前驱与后继

给定 `n` 个互不相同的整数与 `q` 次询问，每次询问整数 `x`，输出集合中小于 `x` 的最大值（前驱）与大于 `x` 的最小值（后继），不存在时输出 `-1`。

输入格式：第一行两个整数 `n`、`q`（`1 <= n, q <= 100000`）；第二行 `n` 个整数；接下来 `q` 个整数 `x`。

输出格式：每次询问输出一行两个整数，依次为前驱与后继。

思路：`lower_bound(x)` 给出第一个不小于 `x` 的位置，其前一位置即前驱；`upper_bound(x)` 给出第一个大于 `x` 的位置，即后继；两者返回迭代器，越界处输出 `-1`。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, q;
    cin >> n >> q;
    set<int> s;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        s.insert(x);
    }
    for (int i = 0; i < q; i++)
    {
        int x;
        cin >> x;
        int lo = -1;
        set<int>::iterator it = s.lower_bound(x);
        if (it != s.begin())            // 存在前驱
        {
            --it;
            lo = *it;
        }
        int hi = -1;
        it = s.upper_bound(x);
        if (it != s.end())              // 存在后继
        {
            hi = *it;
        }
        cout << lo << ' ' << hi << endl;
    }
    return 0;
}
```

输入：

```
6 4
3 9 1 7 5 9
5 1 10 9
```

输出：

```
3 7
-1 3
9 -1
7 -1
```

复杂度：每次询问 `O(log n)`，总计 `O(q log n)`。

5.5 注意

1. `multiset` 的 `erase(x)` 删除全部等于 `x` 的元素；仅删除一个须写 `erase(find(x))`，两者在 `set` 中效果相同。

2. 迭代器只读：`*it = v` 无法通过编译；修改元素须先 `erase` 再 `insert`。
3. `--begin()` 为未定义行为；求前驱前须判断 `it != s.begin()`，且 `end()` 不可解引用。
4. `multiset` 的 `count(x)` 复杂度为 `O(log n + k)`（`k` 为相同元素个数），不宜用于仅判断存在性。
5. 仅需判断存在性且值域较小时，`bool` 数组比 `set` 更快。

---

6. `unordered_map` / `unordered_set`：哈希表关联容器

6.1 定义与性质

`unordered_map` 是以哈希表实现的关联容器，存储键值对；`unordered_set` 为其只存键的版本。二者的接口与 `map`、`set` 同源，差异见下表。平均情况下单次查找、插入、删除为 `O(1)`，最坏为 `O(n)`：大量键落入同一桶时，桶内退化为线性查找。遍历顺序由哈希函数与桶数量决定，既不反映插入顺序，也不反映键的大小关系。

性质	`map` / `set`	`unordered_map` / `unordered_set`
底层结构	平衡树	哈希表（桶数组 + 链表）
有序性	按 key 升序	无序
单次操作平均复杂度	`O(log n)`	`O(1)`
单次操作最坏复杂度	`O(log n)`	`O(n)`
常数因子	较大	较小
内存占用	较小	较大
`lower_bound` / `upper_bound`、反向迭代	提供	不提供
迭代器失效条件	删除元素时	插入触发 `rehash` 时全部失效
构造数据退化的风险	无	有

6.2 接口

`unordered_set` 无 `[]` 与 `at`，其余成员与 `unordered_map` 相同，下列接口对两者通用。

（1）构造、赋值、容量

函数 / 写法	作用	复杂度	备注
`unordered_map<K, V> m;`	构造空表	`O(1)`	⭐
`unordered_map<K, V> m = {{k, v}, {k, v}};`	列表构造	`O(n)`	⭐
`unordered_map<K, V> m(m2);`	拷贝构造	`O(n)`	少用
`m = m2;` / `m = {{k, v}, {k, v}};`	赋值	`O(n)`	少用
`m1.swap(m2)` / `swap(m1, m2)`	交换两个容器	`O(1)`	多组数据重置
`empty()` / `size()` / `max_size()`	空判定 / 元素个数 / 个数上限	`O(1)`	⭐ `size()` 返回无符号数

（2）迭代器、元素访问、修改、查找

函数 / 写法	作用	复杂度	备注
`begin()` / `end()` / `cbegin()` / `cend()` / `begin(n)` / `end(n)`	首、尾后、只读迭代器；第 n 个桶的迭代器	`O(1)`	⭐ 顺序不确定
`rbegin()` / `rend()`	不存在	—	无序容器无反向迭代
`m[k]`	访问，键不存在则插入默认值	平均 `O(1)`	⭐ 会插入
`m.at(k)`	访问，键不存在抛 `out_of_range`	平均 `O(1)`	少用
`insert({k, v})`	插入，键已存在则不覆盖	平均 `O(1)`	⭐ 返回 pair
`emplace(k, v)` / `emplace_hint(h, k, v)`	构造并插入	平均 `O(1)`	⭐ 提示位置无效
`insert_or_assign(k, v)` / `try_emplace(k, v)`	存在则覆盖 / 不存在才插入	平均 `O(1)`	C++17

函数 / 写法	作用	复杂度	备注
`erase(k)`	按键删除，返回 `0` 或 `1`	平均 `O(1)`	⭐
`erase(it)`	按迭代器删除，返回后继	平均 `O(1)`	⭐ 遍历中删除
`erase(first, last)` / `clear()`	删除区间 / 清空	`O(n)`	⭐
`merge(m2)`	并入另一容器	平均 `O(n)`	C++17
`find(k)` / `equal_range(k)`	键存在返回迭代器，否则 `end()`；或返回一对迭代器	平均 `O(1)`	⭐
`count(k)`	键出现次数：`0` 或 `1`	平均 `O(1)`	⭐
`contains(k)`	键是否存在	—	C++20，C++17 不可用

（3）桶接口、哈希策略、非成员函数

函数 / 写法	作用	复杂度	备注
`bucket_count()`	当前桶数量	`O(1)`	⭐
`max_bucket_count()`	桶数量上限	`O(1)`	少用
`bucket_size(n)`	第 n 个桶内元素数	`O(1)`	冲突诊断
`bucket(k)`	键 k 所在桶号	平均 `O(1)`	冲突诊断
`load_factor()`	平均装载率 `size() / bucket_count()`	`O(1)`	⭐
`max_load_factor()` / `max_load_factor(z)`	读 / 设最大装载率（默认 `1.0`）	`O(1)`	⭐ 插入前设置
`rehash(n)`	桶数调整为不小于 n	`O(n)`	⭐ 使迭代器失效
`reserve(n)`	预留可容纳 n 个元素的桶	`O(n)`	⭐ 避免反复扩容
`hash_function()` / `key_eq()`	返回哈希函数 / 键相等判定对象	`O(1)`	少用
`m1 == m2` / `m1 != m2`	内容是否相同（与顺序无关）	`O(n)`	—
`erase_if(m, pred)`	按条件批量删除	`O(n)`	C++20，C++17 不可用

6.3 模板程序

读入 n 个整数并计数，随后查询目标值的出现次数与不同值个数。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    unordered_map<int, int> cnt;
    for (int i = 0; i < n; i++)
    {
        int x;
        cin >> x;
        cnt[x]++;
    }
    int q;
    cin >> q;
    if (cnt.find(q) != cnt.end())
    {
        cout << cnt[q] << endl;
    }
    else
    {
        cout << 0 << endl;
    }
    cout << cnt.size() << endl;
    return 0;
}
```

输入：

```
8
3 1 4 1 3 5 1 4
1
```

输出：

```
3
4
```

6.4 卡哈希与 `reserve`

若测试数据由若干落于同一桶的键构成，单次操作退化为 `O(n)`，总体达到 `O(n^2)`。只需消除扩容开销时，可在插入前设置 `max_load_factor(0.7)` 并调用 `reserve`；需要保证最坏复杂度时，应改用 `map`，其最坏复杂度仍为 `O(log n)`。

6.5 例题

例题 6-1：两数之和

题述：给定 n 个整数与目标值 t，求一对下标 `i < j` 使 `a[i] + a[j] = t`，保证存在解。

输入格式：第一行 n 与 t；第二行 n 个整数。输出格式：一行两个整数，为下标（从 1 开始）。

思路：顺序读入，对每个 `a[i]` 查询 `t - a[i]` 是否已出现，平均 `O(1)`。先查询后插入，避免元素与自身配对。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    long long t;
    cin >> n >> t;
    unordered_map<long long, int> p;
    for (int i = 1; i <= n; i++)
    {
        long long x;
        cin >> x;
        long long y = t - x;
        if (p.find(y) != p.end())
        {
            cout << p[y] << ' ' << i << endl;
            return 0;
        }
        p[x] = i;
    }
    return 0;
}
```

输入：

```
5 10
3 9 4 7 1
```

输出：

```
1 4
```

复杂度：`O(n)` 时间，`O(n)` 空间。

6.6 注意

1. `m[k]` 在键不存在时插入默认值，判断存在性须用 `find` 或 `count`。
2. 遍历顺序不确定，需要有序输出时应复制到 `vector` 后排序。
3. `rehash` 使全部迭代器失效，遍历期间不得插入；删除元素用 `it = m.erase(it)`。

---

7. lambda 与自定义排序

7.1 lambda 语法

`sort` 的第三个参数为比较器：接受两个元素并返回 `bool`，返回 `true` 表示第一个参数应排在第二个之前。lambda 形式为 `[捕获列表](参数列表) { 函数体 }`，返回类型由 `return` 推出。降序即 `[](int x, int y) { return x > y; }`。比较器与普通函数等价，但可直接写在调用处，并可通过捕获列表访问外部变量。

7.2 三种比较器模板

模板一：降序。 元素类型为 `long long` 时，参数类型须同步修改。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    vector<long long> a(n);
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
    }
    sort(a.begin(), a.end(), [](long long x, long long y)
    {
        return x > y;
    });
    for (int i = 0; i < n; i++)
    {
        cout << a[i];
        if (i + 1 == n)
        {
            cout << endl;
        }
        else
        {
            cout << ' ';
        }
    }
    return 0;
}
```

输入：

```
4
10000000000 5 9999999999 5
```

输出：

```
10000000000 9999999999 5 5
```

模板二：多关键字。 第一关键字不同时直接返回，相同时再比较第二关键字。

```
#include <bits/stdc++.h>
using namespace std;

struct Stu
{
    string nm;
    int sc;
};

int main()
{
    int n;
    cin >> n;
    vector<Stu> s(n);
    for (int i = 0; i < n; i++)
    {
        cin >> s[i].nm >> s[i].sc;
    }
    sort(s.begin(), s.end(), [](const Stu &p, const Stu &q)
    {
        if (p.sc != q.sc)
        {
            return p.sc > q.sc;
        }
        return p.nm < q.nm;
    });
    for (int i = 0; i < n; i++)
    {
        cout << s[i].nm << ' ' << s[i].sc << endl;
    }
    return 0;
}
```

输入：

```
4
bob 90
alice 90
cindy 85
dave 95
```

输出：

```
dave 95
alice 90
bob 90
cindy 85
```

模板三：`pair`。 `pair` 自带比较规则：先比 `first`，相同再比 `second`，故 `sort(a.begin(), a.end());` 即按第一、第二关键字升序排列，无需比较器。需要调换优先级时，按模板一、二的形式写出 `[](pair<int, int> p, pair<int, int> q) { return p.second > q.second; }` 一类的 lambda 即可。

7.3 捕获列表

`[]` 为空，比较只依赖参数；`[&]` 按引用捕获全部外部变量，`[=]` 按值捕获全部；`[x]`、`[&x]` 只捕获指定变量。按另一数组的值排序时，比较下标：`sort(idx.begin(), idx.end(), [&](int x, int y) { return a[x] < a[y]; });`。引用捕获不产生副本，但被捕获变量须在使用期间存活。

7.4 `set` 与 `priority_queue` 的比较器

比较器在 `set` 中作为第二个模板参数、在 `priority_queue` 中作为第三个模板参数给出：

写法	含义
`set<int, greater<int>> s;`	降序集合，元素自动去重
`priority_queue<int, vector<int>, greater<int>> q;`	小根堆，`top()` 为最小元素

方向约定相反：`sort`、`set` 的比较器返回 `true` 表示第一参数在前；`priority_queue` 返回 `true` 表示第一参数优先级更低，默认 `less<int>` 为大根堆。

7.5 例题

例题 7-1：成绩排名

题述：给定 n 个学生的姓名、语文成绩、数学成绩，按总分降序排名；总分相同按语文降序，语文相同按姓名字典序升序。

输入格式：第一行 n；随后 n 行，每行一个姓名与两个整数成绩。输出格式：n 行，每行姓名与总分。

思路：总分在输入时计算并存储，避免比较器重复求值。比较器按总分、语文、姓名逐层判断，各层不等即返回。

```
#include <bits/stdc++.h>
using namespace std;

struct Stu
{
    string nm;
    int ch;
    int ma;
    int sm;
};

int main()
{
    int n;
    cin >> n;
    vector<Stu> s(n);
    for (int i = 0; i < n; i++)
    {
        cin >> s[i].nm >> s[i].ch >> s[i].ma;
        s[i].sm = s[i].ch + s[i].ma;
    }
    sort(s.begin(), s.end(), [](const Stu &p, const Stu &q)
    {
        if (p.sm != q.sm)
        {
            return p.sm > q.sm;
        }
        if (p.ch != q.ch)
        {
            return p.ch > q.ch;
        }
        return p.nm < q.nm;
    });
    for (int i = 0; i < n; i++)
    {
        cout << s[i].nm << ' ' << s[i].sm << endl;
    }
    return 0;
}
```

输入：

```
5
bob 80 90
alice 95 85
cindy 70 70
dave 90 80
eve 95 85
```

输出：

```
alice 180
eve 180
dave 170
bob 170
cindy 140
```

复杂度：`O(n log n)` 时间，`O(n)` 空间。

7.6 注意

1. 比较器须满足严格弱序：`cmp(x, x)` 为 `false`；`cmp(x, y)` 为 `true` 时 `cmp(y, x)` 为 `false`；且满足传递性。写成 `x <= y` 违反第一条，`sort` 行为未定义，可能越界或死循环。
2. 相等情形必须返回 `false`；多关键字比较须显式写出全部判等分支。
3. 并列情形须给出确定的后备比较规则，否则元素相对顺序由实现决定。
4. 比较依据不得在排序过程中改变；需要额外信息时应在排序前存入结构体。

---

8. 综合例题

本章三道例题分别组合 `vector` 与下标模拟、`string` 与 `set`、`map` 与 lambda 排序，只使用第 2~7 章介绍的类型与算法。程序统一以 `#include <bits/stdc++.h>` 与 `using namespace std;` 起始，编译命令为 `g++ -std=c++17 -O2 -Wall`。

8.1 `vector` 模拟

例题 8-1：报数出列

题述：`n` 个人围成一圈，编号 `1` 至 `n`；从 1 号起依次报数，报到 `m` 者出列，其后由下一位重新从 1 报数，直至全员出列。

输入格式：一行两个整数 `n`、`m`（`1 ≤ n ≤ 2000`，`1 ≤ m ≤ n`）。输出格式：一行 `n` 个整数，即出列顺序，相邻两数以单个空格分隔。

思路：`vector` 仅保存尚未出列者，出列即按下标 `erase`（中间删除为 `O(n)`）；报到 `m` 者对应当前圈内下标 `(p + m - 1) % size`。删除后 `p` 可能等于 `size`，下一轮取模即回到圈首，无需额外修正。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n, m;
    cin >> n >> m;
    vector<int> a(n);
    for (int i = 0; i < n; i++)
    {
        a[i] = i + 1;
    }
    int p = 0;
    for (int k = 0; k < n; k++)
    {
        p = (p + m - 1) % (int)a.size();
        cout << a[p];
        if (k + 1 < n)
        {
            cout << ' ';
        }
        else
        {
            cout << endl;
        }
        a.erase(a.begin() + p);
    }
    return 0;
}
```

输入：

```
7 3
```

输出：

```
3 6 2 7 5 1 4
```

复杂度：时间 `O(n^2)`，空间 `O(n)`。

8.2 `string` 与 `set`

例题 8-2：单词去重与字典序输出

题述：给定一行小写字母文本，单词以单个空格分隔；输出其中出现的不同单词，按字典序升序，每行一个，末行输出不同单词的个数。

输入格式：一行字符串，长度不超过 10^5，行首行尾无空格。输出格式：若干行，前若干行为单词，末行为单词个数。

思路：`getline` 读入整行，`find` 与 `substr` 按空格切分（见第 2 章）；`find` 返回 `npos` 表示其后已无空格，此时以串长作为末词终点。切分结果插入 `set<string>`，重复者自动舍弃，且遍历序即字典序，无需再排序。

```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string s;
    getline(cin, s);
    set<string> S;
    size_t i = 0;
    while (i < s.size())
    {
        size_t j = s.find(' ', i);
        if (j == string::npos)
        {
            j = s.size();
        }
        S.insert(s.substr(i, j - i));
        i = j + 1;
    }
    for (const string &w : S)
    {
        cout << w << endl;
    }
    cout << S.size() << endl;
    return 0;
}
```

输入：

```
cherry apple cherry
```

输出：

```
apple
cherry
2
```

复杂度：时间 `O(L + k log k)`，其中 `L` 为文本长度、`k` 为单词数；空间 `O(k)`。

8.3 `map` 计数与 lambda 排序

例题 8-3：词频排序

题述：给定 `n` 个小写字母单词，按其出现次数降序输出；次数相同者按字典序升序，每个单词只输出一次。

输入格式：第一行整数 `n`（`1 ≤ n ≤ 10^5`），第二行 `n` 个单词，长度不超过 10。输出格式：每行一个单词及其次数，以单个空格分隔。

思路：`map<string, int>` 以单词为键累计次数；其迭代器不支持随机访问，不可直接 `sort`，故用区间构造（见 3.2 节）将键值对复制到 `vector<pair<string, int>>`。比较器用 lambda 依次比较次数（降序）与单词（升序），两关键字均须写出以满足严格弱序。

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;
    map<string, int> cnt;
    for (int i = 0; i < n; i++)
    {
        string w;
        cin >> w;
        cnt[w]++;
    }
    vector<pair<string, int>> v(cnt.begin(), cnt.end());
    sort(v.begin(), v.end(),
        [](const pair<string, int> &a, const pair<string, int> &b)
        {
            if (a.second != b.second)
            {
                return a.second > b.second;
            }
            return a.first < b.first;
        });
    for (int i = 0; i < (int)v.size(); i++)
    {
        cout << v[i].first << ' ' << v[i].second << endl;
    }
    return 0;
}
```

输入：

```
5
pear apple apple banana banana
```

输出：

```
apple 2
banana 2
pear 1
```

复杂度：时间 `O(n log n)`，空间 `O(k)`，`k` 为不同单词数。

---

9. 速查总表

9.1 六容器常用操作

操作	`string s`	`vector<int> v`	`map<K, V> m`	`set<K> s`	`unordered_map<K, V> m`	`unordered_set<K> s`
声明	`string s;`	`vector<int> v;`	`map<string, int> m;`	`set<int> s;`	`unordered_map<string, int> m;`	`unordered_set<int> s;`
添加	`s += "ab";`	`v.push_back(x);`	`m[k] = v;`	`s.insert(x);`	`m[k] = v;`	`s.insert(x);`
删除	`s.erase(pos, len);`	`v.erase(v.begin() + i);`	`m.erase(k);`	`s.erase(x);`	`m.erase(k);`	`s.erase(x);`
查找	`s.find(t);`	线性扫描	`m.find(k);`	`s.find(x);`	`m.find(k);`	`s.find(x);`
存在性	`s.find(t) != string::npos`	线性扫描	`m.count(k)`	`s.count(x)`	`m.count(k)`	`s.count(x)`
长度	`s.size()`	`v.size()`	`m.size()`	`s.size()`	`m.size()`	`s.size()`
遍历	下标 / 范围 for	下标 / 范围 for	范围 for（升序）	范围 for（升序）	范围 for（无序）	范围 for（无序）
清空	`s.clear()`	`v.clear()`	`m.clear()`	`s.clear()`	`m.clear()`	`s.clear()`
排序	`sort(s.begin(), s.end())`	`sort(v.begin(), v.end())`	键有序	元素有序	不支持	不支持
二分	无	`lower_bound`（须有序）	`m.lower_bound(k)`	`s.lower_bound(x)`	不支持	不支持

9.2 算法函数

函数	作用	用法	复杂度
`sort`	升序排序	`sort(v.begin(), v.end());`	`O(n log n)`
`sort` + `greater`	降序排序	`sort(v.begin(), v.end(), greater<int>());`	`O(n log n)`
`sort` + lambda	自定义序	`sort(v.begin(), v.end(), cmp);`	`O(n log n)`
`stable_sort`	稳定排序	`stable_sort(v.begin(), v.end());`	`O(n log n)`
`reverse`	反转	`reverse(v.begin(), v.end());`	`O(n)`
`unique`	去除相邻重复	`v.erase(unique(v.begin(), v.end()), v.end());`	`O(n)`
`lower_bound`	首个 `>= x` 的位置	`lower_bound(v.begin(), v.end(), x) - v.begin();`	`O(log n)`
`upper_bound`	首个 `> x` 的位置	`upper_bound(v.begin(), v.end(), x) - v.begin();`	`O(log n)`
`binary_search`	判断存在	`binary_search(v.begin(), v.end(), x);`	`O(log n)`
`count` / `count_if`	计数	`count(v.begin(), v.end(), x);`	`O(n)`
`find`	查找位置	`find(v.begin(), v.end(), x);`	`O(n)`
`max_element` / `min_element`	最值迭代器	`*max_element(v.begin(), v.end());`	`O(n)`
`fill`	区间赋值	`fill(v.begin(), v.end(), 0);`	`O(n)`
`next_permutation`	下一个排列	`next_permutation(a, a + n);`	每次 `O(n)`
`accumulate`	求和	`accumulate(v.begin(), v.end(), 0LL);`	`O(n)`
`gcd` / `lcm`	最大公约数/最小公倍数	`gcd(a, b);`	`O(log min(a, b))`

9.3 选型

需求	选择
变长数组、随机访问、排序	`vector`
文本处理	`string`
键值映射、按键有序输出	`map`
去重并保持有序、前驱后继查询	`set`
仅计数/判重、不要求顺序	`unordered_map` / `unordered_set`
允许重复的有序集合	`multiset`
自定义排序规则	`sort` + lambda
图/树邻接表	`vector<int> g[MAXN];`

---

10. 常见错误

1. 开头缺 `#include <bits/stdc++.h>` 或 `using namespace std;`，编译报未声明标识符。
2. `operator[]` 越界不报错，结果未定义。
3. `size()` 为无符号类型，空容器上 `size() - 1` 得到极大值。
4. 对 `end()` 解引用，行为未定义。
5. 遍历中 `erase` 导致下标/迭代器失效；`vector` 需按返回值或逆序遍历处理。
6. 用 `m[x]` 判断键存在，会插入默认值并改变 `size()`。
7. `find` 返回 `end()` 后仍解引用。
8. `multiset::erase(x)` 删除全部等于 `x` 的元素；删除单个用 `erase(find(x))`。
9. `string::find` 返回 `string::npos` 时当作合法下标使用。
10. 依赖 `unordered_*` 的遍历顺序。
11. 哈希表被构造数据退化；必要时改用 `map` 或预先 `reserve`。
12. 比较函数不满足严格弱序（相等时返回 `true`），导致 `sort` 运行错误。
13. 未排序即调用 `lower_bound`、`upper_bound`、`binary_search`。
14. 多组数据未清空容器。
15. 函数参数按值传递大容器，造成复制开销。

---

11. 练习

11.1 必做（25 题）

序号	任务	章节
1	统计一行中的单词数	2.3
2	判断回文串	2.3
3	大小写转换	2.3
4	统计子串出现次数（可重叠）	2.4
5	求数字字符之和	2.3
6	输出字符串的字典序最小排列	2.3
7	读入 n 个数，正序与逆序输出	3.2
8	求和、最大值、最小值	3.2
9	排序后去重输出	3.4
10	多次询问 x 是否存在	3.4
11	多次询问 x 首次出现位置	3.4
12	删除指定元素后输出	3.2
13	矩阵转置	3.2
14	统计单词频次	4.3
15	输出频次最高的单词	4.3
16	按姓名排序学生	4.2
17	统计各分数段人数	4.3
18	删除指定键	4.2
19	输出去重后的数列	5.2
20	判断元素是否在集合中	5.2
21	查询大于 x 的最小数	5.2
22	查询小于 x 的最大数	5.2
23	统计出现次数最多的数	6.2
24	按绝对值排序	7.3
25	多关键字排序	7.3

11.2 提高（10 题）

序号	任务	章节
26	判断两个字符串是否为字母异位词	2.3
27	求最长连续相同字符段	2.2
28	词频排序：次数降序，同次数按字典序	4.3 + 7.3
29	二维矩阵行列和	3.2
30	邻接表读图并输出邻居	3.2
31	区间合并	4.4
32	离散化	4.4
33	前缀出现次数统计	4.3
34	输出所有重复元素	4.3

序号	任务	章节
35	两个有序数组合并去重	9.2

11.3 拓展（5 题）

序号	任务	章节
36	用 `next_permutation` 枚举全排列	9.2
37	求不同元素个数（n 至 10^6）	3.4
38	动态中位数（两个 `multiset`）	5.3
39	第 k 小查询	5.3
40	最长无重复字符子串	5.2

12. 配套例题（ICPC 省赛）

下列题目按章节对应，难度标注为洛谷难度或 Codeforces rating（约 800~~1500~~），~~适合省赛前的短期训练。建议按表格顺序完成；每题先独立实现，再对照第 28 章的接口与模板。~~

12.1 字符串（第 2 章）

#	题目	平台 / 编号	难度	考点
1	统计单词数	洛谷 P1308	普及-	`getline`、`find`、大小写转换
2	拼数	洛谷 P1012	普及/提高-	字符串比较、自定义排序
3	Way Too Long Words	CF 71A	800	字符串长度、拼接
4	String Task	CF 118A	800	字符筛选、大小写

12.2 vector 与二分（第 3 章）

#	题目	平台 / 编号	难度	考点
5	【模板】排序	洛谷 P1177	普及-	`sort`
6	明明的随机数	洛谷 P1059	普及-	`sort` + `unique`
7	【深基13.例1】查找	洛谷 P2249	普及-	`lower_bound`
8	烦恼的高考志愿	洛谷 P1678	普及/提高-	排序 + 二分
9	T-primes	CF 230B	1300	平方数判定 + 筛法/二分

12.3 map（第 4 章）

#	题目	平台 / 编号	难度	考点
10	保龄球	洛谷 P1918	普及-	`map` 映射、查找
11	Registration system	CF 4C	1300	`map` 计数与重名处理
12	Equal Sums	CF 988C	1500	`map` 记录状态

12.4 set（第 5 章）

#	题目	平台 / 编号	难度	考点
13	Tom Riddle's Diary	CF 855A	800	`set` 判重
14	眼红的 Medusa	洛谷 P1571	普及-	`set` / `map` 判存在

12.5 unordered_map（第 6 章）

#	题目	平台 / 编号	难度	考点
15	Same Differences	CF 1520D	1200	哈希计数
16	A-B 数对	洛谷 P1102	普及/提高-	哈希计数 / 排序双指针

12.6 lambda 自定义排序（第 7 章）

#	题目	平台 / 编号	难度	考点
17	奖学金	洛谷 P1093	普及-	多关键字排序
18	生日	洛谷 P1104	普及-	多关键字排序、稳定性
19	攀爬者	洛谷 P5143	普及-	按自定义键排序

12.7 平台题单

- 牛客 STL 专项题单：https://ac.nowcoder.com/acm/problem/collection/1686
- 牛客字符串题单：https://ac.nowcoder.com/acm/problem/collection/6760

12.8 训练安排

阶段	内容	时间
1	阅读第 1~7 章，完成各章模板程序	1 天
2	完成第 8 章例题与 12.1~12.6 的 19 道配套题	1 天
3	完成第 11 章 25 道必做题	1 天
4	限时模拟：从 12.7 题单中各选 2 题，60 分钟内完成	1 天
