作业讲解（一）

数据结构概述、算法复杂度、栈与队列、STL 应用

陈宇琪

2020 年 3 月 28 日

Content

STL

More About Vector

fx Member functions

(constructor)	Construct vector (public member function)
(destructor)	Vector destructor (public member function)
operator=	Assign content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

size	Return size (public member function)
max_size	Return maximum size (public member function)
resize	Change size (public member function)
capacity	Return size of allocated storage capacity (public member function)
empty	Test whether vector is empty (public member function)
reserve	Request a change in capacity (public member function)
shrink_to_fit C++11	Shrink to fit (public member function)

Element access:

operator[]	Access element (public member function)
at	Access element (public member function)
front	Access first element (public member function)
back	Access last element (public member function)
data C++11	Access data (public member function)

Modifiers:

assign	Assign vector content (public member function)
push_back	Add element at the end (public member function)
pop_back	Delete last element (public member function)
insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_back C++11	Construct and insert element at the end (public member function)

Allocator:

get_allocator	Get allocator (public member function)

图: Vector Reference

STL

More About Vector

- v.erase() 函数的算法复杂度是 $O(N)$，也就是说在执行 v.erase() 函数的时候，会将其他所有元素的迭代器向前移动。

Listing 1: **vector.cpp**

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

int main()
{
        vector<int> v;
        v.push_back(1);
        v.push_back(2);
        v.push_back(3);
        for (int i=0;i<v.size();i++)
                cout<<(&v[i])<<' '<<v[i]<<endl;
        v.erase(v.begin());
        cout<<"-------"<<endl;
        for (int i=0;i<v.size();i++)
                cout<<(&v[i])<<' '<<v[i]<<endl;
        return 0;
}
```

上面程序在我的电脑上运行结果为：
0x21440 1
0x21444 2
0x21448 3
———-
0x21440 2
0x21444 3

- 由此证明了 erase 函数在执行时，修改了所有的迭代器！

STL

More About Vector

- 1、如何定义一个 int 类型的 vector。

Listing 2: **ans1.1**

```
vector<int> vec;
```

STL

More About Queue

fx Member functions

(constructor)	Construct queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
front	Access next element (public member function)
back	Access last element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove next element (public member function)
swap C++11	Swap contents (public member function)

图: Queue Reference

Listing 3: **queue.cpp**

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

int main()
{
        queue<int> q;
        ...
        // Bad Habit for beginner
        cout<<q.top()<<endl;

        // Good Habit for beginner
        assert(q.size());
        cout<<q.top()<<endl;

        // Good Habit for beginner
        if (!q.empty())
                cout<<q.top()<<endl;
        return 0;
}
```

STL

More About Stack

fx Member functions

(constructor)	Construct stack (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access next element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

图: Stack Reference

STL

More About Set

fx Member functions

(constructor)	Construct set (public member function)
(destructor)	Set destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Modifiers:

insert	Insert element (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return comparison object (public member function)
value_comp	Return comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific value (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

图: Set Reference

STL

More About Set

- set 的迭代器是不连续的，所以迭代器不能进行大小比较!

Listing 4: **set.cpp**

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

int main()
{
        set<int> s;
        s.insert(1);
        s.insert(2);
        s.insert(3);
        for (set<int>::iterator it=s.begin();it!=s.end();it++)
                cout<<(&(*it))<<' '<<(*it)<<endl;
        s.erase(1);
        cout<<"-------"<<endl;
        for (set<int>::iterator it=s.begin();it!=s.end();it++)
                cout<<(&(*it))<<' '<<(*it)<<endl;
        return 0;
}
```

上面程序在我的电脑上运行结果为：

0x171460 1
0x1714b0 2
0x171950 3
———-
0x1714b0 2
0x171950 3

- 由于 set 内部使用红黑树的数据结构实现，所以迭代器的内存空间是不连续的。
- 我们还可以发现，在 erase 之后，其余迭代器的值不变。

- 2、如何遍历一个 set 中的所有元素并输出。

Listing 5: **ans1.2**

```
// a is a set.
for (auto &i : a) {
    cout << i << '\n';
}

// Ok, if you don't like range-for...
for (auto it = a,begin(); it != a.end(); ++it) {
    cout << *it << '\n';
}
```

STL

More About Set

- s.lower_bound(x) 函数返回 s 中第一个大于等于 x 的元素，如果 x 大于 s 中所有元素，返回 s.end() 迭代器。
- s.upper_bound(x) 函数返回 s 中第一个大于 x 的元素，如过不存在这样的元素，返回 s.end() 迭代器。
- s.lower_bound 和 s.upper_bound 的复杂度均为 $O(logN)$
- s.find(x) 函数查找 s 中是否存在 x，如果存在返回 x 对应的迭代器，否则返回 s.end()。

Listing 6: **set2.cpp**

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

int main()
{
        set<int> s;
        s.insert(1);
        set<int>::iterator it1=s.lower_bound(1);
        assert(it1!=s.end());
        set<int>::iterator it2=s.upper_bound(1);
        assert(it2==s.end());
        set<int>::iterator it3=s.find(1);
        assert(it3==it1);
        set<int>::iterator it4=s.find(0);
        assert(it4==s.end());
        return 0;
}
```

- 3、查找一个 int 类型的 set 中大于 1 的第一个元素（如果不存在，如何做处理）。

Listing 7: **ans1.3**

```
// a is a set.
if (auto it = a.upper_bound(1); it == a.end()) {
    cout << "No element in a is greater than 1.";
} else {
    cout << *it;
}
```

STL

More About Set

- s.erase(x) 函数 s 中值为 x 的元素，注意如果 x 不存在，不会造成异常！
- s.erase(s.lower_bound(x)) 这时候由于 lower_bound 可能返回 s.end()，如果执行 s.erase(s.end()) 是会造成异常的！

- 4、定义一个 int 类型的 set，并在定义的 set 中删除一个元素 1。

Listing 8: **ans1.4**

```
set<int> a;

// I just defined a, and there is no element in it at all.
a.erase(1);
```

STL

More About Set

Listing 9: **set3**

```
#include <bits/stdc++.h>
using namespace std;
struct node
{
        int a,b;
        bool operator < (const node &rhs) const {
                return a < rhs.a;
        }
};
int main() {
        set<node> s;
        s.insert(node{1,2});
        s.insert(node{1,3});
        cout<<s.size()<<endl;    // 1
        // s.begin()->a = 2;  // compile-error: read-only object
}
```

- 在重载运算符是否需要小心各种情况!
- 修改 set 元素的值，只能删除后重新插入。

STL

More About Set

- 为什么 set 不需要重载 $==$?
- $x == y$ 等价于 $!(x < y)$ && $!(y < x)$。

Listing 10: **set4**

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int a;
    bool operator < (const node &rhs) const {
        return a<=rhs.a;
    }
};
int main() {
    set<node> s{node{1},node{2},node{1},node{2}};
    for (set<node>::iterator it=s.begin();it!=s.end();it++)
        cout<<it->a<<(it==--s.end()?'\n':' ');   // 1 1 2 2
    s.erase(s.lower_bound(node{1}));
    for (set<node>::iterator it=s.begin();it!=s.end();it++)
        cout<<it->a<<(it==--s.end()?'\n':' ');  // 1 1 2
}
```

Listing 11: **set5**

```
#include <bits/stdc++.h>
using namespace std;
struct node {
    int x,y;
    bool operator < (const node &rhs) const {
        return x < rhs.x;
    }
    bool operator == (const node &rhs) const {
        return x==rhs.x && y==rhs.y;
    }
};
int main() {
    set<node> s{node{1,3},node{1,2},node{2,2},node{3,3},node{3,4}};
    cout<<s.size()<<endl; // 3
}
```

Listing 12: **multiset1.cpp**

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

void output(const multiset<int> & ms) {
    for (multiset<int>::iterator it=ms.begin();it!=ms.end();it++)
        cout<<(&(*it))<<' '<<(*it)<<endl;
}
int main() {
    multiset<int> ms{1,1,2,2,2,3};
    output(ms);
    ms.erase(1);
    cout<<"-------"<<endl;
    output(ms);
    if (ms.count(2))
        ms.erase(ms.lower_bound(2));
    cout<<"-------"<<endl;
    output(ms);
    return 0;
}
```

上面程序在我的电脑上运行结果为：

0x10518f0 1
0x1051910 1
0x1051930 2
0x1051950 2
0x10573d0 2
0x10573f0 3
———-
0x1051930 2
0x1051950 2
0x10573d0 2
0x10573f0 3
———-
0x1051950 2
0x10573d0 2
0x10573f0 3

STL

More About Multiset

- multiset 在删除元素的时候如果只要求删除一个的话，应该使用迭代器删除，在使用迭代器的时候，一定要确保传进去的参数不是 ms.end()。
- 我们可以使用 ms.count(x) 和 ms.find(x) != ms.end() 两种方法来判断 ms 中是否存在元素 x。

STL

More About Multiset

- 5、定义一个 int 类型的 multiset，并在定义的 multiset 中删除一个元素 1（如果有多个 1 只删除 1 个，保留剩下的 1）。

Listing 13: **ans1.5**

```
multiset<int> a;
if (auto it = a.find(1); it != a.end()) {
    // a.erase(1) would cause all key == 1 nodes to be erased.
    // So we use iterator instead.
    // The iterator passed to multiset<>::erase must be
        dereferencable,
    // thus it cannot be a.end();
    a.erase(a.find(1));
}

// if you are using C++ 11/C++ 14 you can write like this
multiset<int> a;
multiset<int>::iterator it = a.lower_bound(1);
if (it != a.end() && *it==1) {
        a.erase(it);
}
```

STL

More About Multiset

- 6、定义一个 pair<int,int> 类型的 multiset 在定义的 multiset 中删除一个满足 a.first==1 的最小的元组 a（假设 multiset 中所有数的绝对值小于 10^5，如果有多个元组满足要求，只删除其中最小的 1 个，保留剩下的元组)。

Listing 14: **ans1.6**

```
multiset<pair<int, int>> a;
if (auto it = a.lower_bound({1, numeric_limits<int>::min()}); it !=
    a.end()) {
    a.erase(it);
}

// if you are using C++ 11/C++ 14 you can write like this
multiset<pair<int,int>> a;
multiset<pair<int,int>>::iterator it = a.lower_bound(make_pair
    (1,-1000000));
if (it != a.end() && it->first==1) {
        a.erase(it);
}
```

STL

More About Multiset

- 第五题的两个典型错误是：s.erase(1); 和 s.erase(s.find(1));。
- 两道题目都可以使用遍历的方法完成，但是这样做的复杂度过高！

STL

More About Map

Member functions

(constructor)	Construct map (public member function)
(destructor)	Map destructor (public member function)
operator=	Copy container content (public member function)

Iterators:

begin	Return iterator to beginning (public member function)
end	Return iterator to end (public member function)
rbegin	Return reverse iterator to reverse beginning (public member function)
rend	Return reverse iterator to reverse end (public member function)
cbegin C++11	Return const_iterator to beginning (public member function)
cend C++11	Return const_iterator to end (public member function)
crbegin C++11	Return const_reverse_iterator to reverse beginning (public member function)
crend C++11	Return const_reverse_iterator to reverse end (public member function)

Capacity:

empty	Test whether container is empty (public member function)
size	Return container size (public member function)
max_size	Return maximum size (public member function)

Element access:

operator[]	Access element (public member function)
at C++11	Access element (public member function)

Modifiers:

insert	Insert elements (public member function)
erase	Erase elements (public member function)
swap	Swap content (public member function)
clear	Clear content (public member function)
emplace C++11	Construct and insert element (public member function)
emplace_hint C++11	Construct and insert element with hint (public member function)

Observers:

key_comp	Return key comparison object (public member function)
value_comp	Return value comparison object (public member function)

Operations:

find	Get iterator to element (public member function)
count	Count elements with a specific key (public member function)
lower_bound	Return iterator to lower bound (public member function)
upper_bound	Return iterator to upper bound (public member function)
equal_range	Get range of equal elements (public member function)

图: Map Reference

STL

More About Map

- c 语言对所有基本类型都有初始值，map 里的元素初始值也是基本类型的初始值。
- map 删除一个元素用 mp.erase(x)。

Listing 15: **map**

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

int main()
{
        map<int,int> mp;
        mp[0]++;
        cout << mp[0] << endl;          // 1
        cout << mp.count(0) << endl;    // 1
        mp[0]--;
        cout << mp[0] << endl;          // 0
        cout << mp.count(0) << endl;    // 1
        mp.erase(0);
        cout << mp.count(0) << endl;    // 0
}
```

STL

More About Map

- map 和 unordered_map 的区别在于 map 是基于红黑树实现的，而 unordered_map 基于哈希实现。
- map 按照 key 排序，所以需要重载小于号，unordered_map 不会按照 key 排序。
- unordered_map 如果需要支持自定义结构体，需要较为复杂的运算符重载。
- map 插入和查询复杂度均为 $O(log(N))$，unordered_map 插入和查询复杂度均为 $O(1)$。

Listing 16: **map2**

```
#include <bits/stdc++.h>
using namespace std;
int main() {
        map<int,int> mp;
        unordered_map<int,int> ump;
        for (int i=0;i<10;i++)
                mp[rand()]=rand(),ump[rand()]=rand();
        for (map<int,int>::iterator it=mp.begin();it!=mp.end();it++)
                cout<<it->first<<' ';
        cout<<endl;
        // 41 292 2995 14771 19169 19718 23281 25667 26962 32391
        for (unordered_map<int,int>::iterator it=ump.begin();it!=ump
             .end();it++)
                cout<<it->first<<' ';
        cout<<endl;
        // 17035 5447 17421 11478 6334 5705 1869 9961 4827 3902
        return 0;
}
```

STL

More About Priority Queue

fx Member functions

(constructor)	Construct priority queue (public member function)
empty	Test whether container is empty (public member function)
size	Return size (public member function)
top	Access top element (public member function)
push	Insert element (public member function)
emplace C++11	Construct and insert element (public member function)
pop	Remove top element (public member function)
swap C++11	Swap contents (public member function)

图: Priority Queue Reference

STL

More About Priority Queue

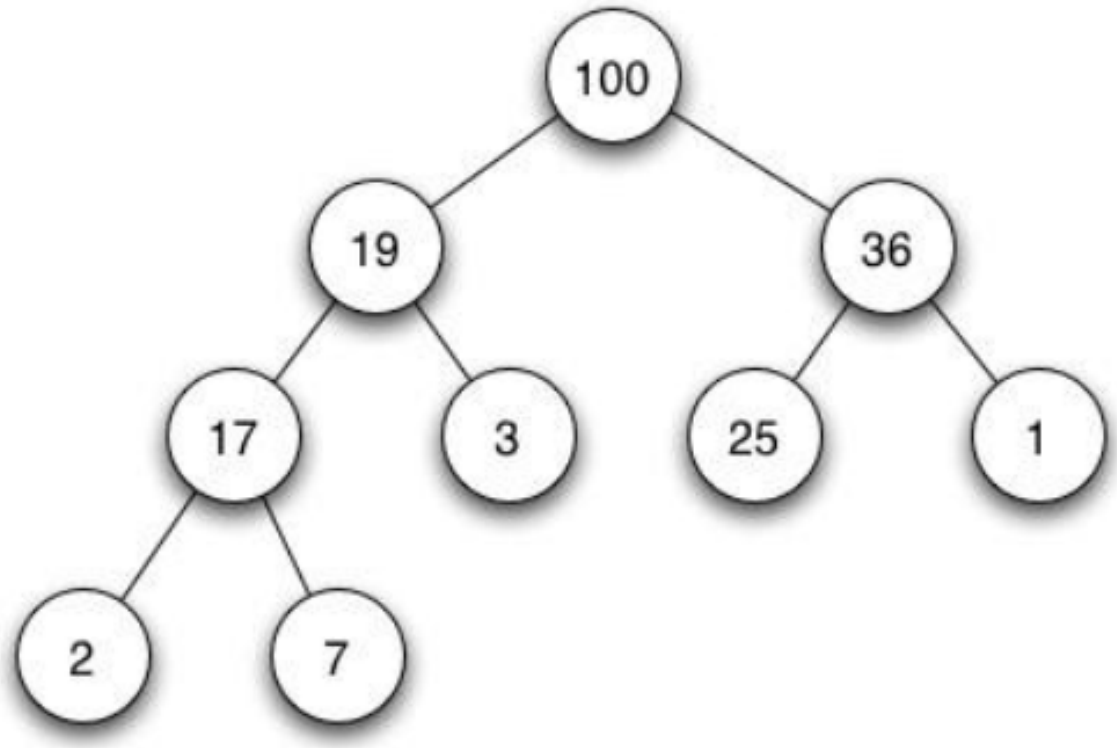

图: Heap

STL

More About Priority Queue

- 大根堆满足：根节点的值大于子节点的值。
- 最大的值在根的地方。
- pq.top() 的复杂度是 $O(1)$，因为返回根的元素就可以。
- 堆是一棵完全二叉树，所以树的深度是 $O(log(N))$。
- 插入和删除的时候需要维护堆的性质。
- pq.pop()/pq.push(x) 的复杂度是 $O(log(N))$。

- 7、priority_queue 默认是大根堆还是小根堆？如果默认是小根堆，如何定义一个 int 类型的大根堆？相反，如果默认是大根堆，如何定义一个 int 类型的小根堆？

priority_queue 默认是大根堆，定义小根堆方式如下：

Listing 17: **ans1.7**

```
priority_queue<int, vector<int>, greater<int>> pq;
```

STL

More About Priority Queue

- 8、假设如下结构体：struct www{int,a,b;}; 我们希望在 priority_queue 中按照 a 的值从小到大 pop 出来，那么应该对这个结构体做些什么呢?

Listing 18: **ans1.8**

```
struct www {
    int a, b;
    // remember the const, for elements in priority queue is const.
    bool operator<(www const& rhs) const {
        // note that we use > instead of <
        return a > rhs.a;
    }
};

// Another apporach is to write another struct to define the
    comparision
// But I think that this apporach is too complex
// So I only show you this answer
```

- 9、离散化问题。

Listing 19: **ans1.9.1**

```
// Appoach 1: vector + sort + unique + lower_bound
int n,a[maxn];
vector <int> key;
int main() {
        cin >> n;
        for (int i = 0; i < n; i++) {
                cin >> a[i];
                key.push_back(a[i]);
        }
        sort(key.begin(),key.end());
        int m = unique(key.begin(), key.end()) - key.begin();
        for (int i = 0; i < n; i++) {
                // Typically we do not want the index to start with
                    0
                a[i] = lower_bound(key.begin(), key.end(), a[i]) -
                    key.begin() + 1;
        }
}
```

- 9、离散化问题。

Listing 20: **ans1.9.2**

```
// Appoach 2: map
int n,a[maxn];
map<int,int> mp;
int main() {
        cin >> n;
        for (int i = 0; i < n; i++) {
            cin >> a[i];
            mp[a[i]] = 1;
        }
        int idx = 1;
        for (map<int,int>::iterator it = mp.begin();it != mp.end();
            it++, idx++) {
            mp[it->first] = idx;
        }
        for (int i = 0; i < n; i++) {
            a[i] = mp[a[i]];
        }
}
```

- 9、离散化问题。

Listing 21: **ans1.9.3**

```
// Appoach 3: set
int n,a[maxn];
set<int> s1;
set<pair<int,int> > s2;
int main() {
        cin >> n;
        for (int i = 0; i < n; i++) {
                cin >> a[i];
                s1.insert(a[i]);
        }
        int idx = 1;
        for (set<int>::iterator it = s1.begin();it != s1.end(); it
            ++, idx++) {
                s2.insert(make_pair(*it,idx));
        }
        for (int i = 0; i < n; i++) {
                a[i] = s2.lower_bound(make_pair(a[i],0))->second;
        }
```

- 在 ACM 比赛中通常使用第一种，因为常数比较小，但是复杂度都是 $O(nlog(n))$。

- 10、前 K 大问题：给定一个长度为 N 数组，求每个连续 K 个数的最大值，复杂度要求 $O(N \times log(K))$。
- 解法一：使用 multiset 维护一个大小为 K 的窗口中的元素，使用 ms.erase() 删除，使用 ms.insert() 插入，使用 *(–ms.end()) 输出最大值。

常见错误：

- 使用 set 维护，如果使用 set 的话，则需要将 (a[i],i) 作为元组插入才可以保证元组的唯一性。
- ms.erase() 需要使用迭代器删除。

- 10、前 K 大问题：给定一个长度为 N 数组，求每个连续 K 个数的最大值，复杂度要求 $O(N \times log(K))$。
- 解法二：使用 priority_queue 维护一个 (a[i],i) 的大根堆，每次将不在区间范围内的所有元组 pop 掉，再插入当前元组，使用 top() 函数获取当前区间的最大值。
- 复杂度：$O(N \times log(N))$。

常见错误：

- 需要 pop 掉所有不在区间中的元组。

- 10、前 K 大问题：给定一个长度为 N 数组，求每个连续 K 个数的最大值，复杂度要求 $O(N \times log(K))$。
- 解法三：使用 deque 维护一个单调队列。
- 具体描述：设置一个双端队列来维护窗口内数据最多纪录窗口大小 K 个元素，由于采用单调队列所以队列中的元素是严格按照元素大小递减的方式，为了维护窗口的可是范围，即窗口以外的元素要及时丢掉，**队列中保存的并不是元素本身而是元素在原数组当中的序号或则索引。**
- 复杂度 $O(N)$。

STL

More About STL

- 10、前 K 大问题：给定一个长度为 N 数组，求每个连续 K 个数的最大值，复杂度要求 $O(N \times log(K))$。
- 异常处理（2 分）：判断是否如果 $K <= 0$ 或者 $K > N$ 成立，则做异常处理（如输出错误信息后结束程序）。

More About STL

代码查重

- 题意：给定若干同义关系 x 和 y，表示 x 和 y 互相同义，特别的，x 和 x 是同义的，要求判断 n 对数 a_i 和 m 个数 b_i 是否有 a_i 和 b_i 同义。
- 解法 1：使用 set 记录所有的用以关系即可，注意当 $n \neq m$ 的情况。

More About STL

代码查重

- 题意：给定若干同义关系 x 和 y，表示 x 和 y 互相同义，特别的，x 和 x 是同义的，要求判断 n 对数 a_i 和 m 个数 b_i 是否有 a_i 和 b_i 同义。
- 解法 2：注意到 x 和 y 的绝对大小没有影响，所以我们可以将所有的输入数据进行离散化，同时我们还注意到这些数的相对大小也没有影响，所以可以使用 unordered_map 进行离散化。离散好之后为每个数开一个 vector 数组，记录和这个数有同义关系的数的列表（这里直接使用离散后的值），对每一个 vector 进行排序以支持在查询过程中使用 lower_bound 或者二分查找。
- 注意：之所以需要离散化是因为数的范围有 10^9，我们不能直接开一个长度为 10^9 的数组。

Stack and Queue

简答题分析

- 1、用数组实现队列有什么不足的地方？是否可以用 vector 实现一个 $O(1)$ 插入，$O(1)$ 删除的队列？
- 数组实现的缺点：需要维护两个指针，分别指向队列的头和尾部，删除的时候会造成空间浪费，数组大小可以通过扩容操作实现动态变化。
- 循环队列的缺点：队列长度有限。
- vector 实现的缺点：vector 的 erase 函数的复杂度是 $O(N)$，所以并不能完全解决数组实现过程中的问题。

Stack And Queue

1.2 10 以内的后缀表达式求解

- 题意：计算后缀表达式的值。
- 分析：主要的问题出在对于后缀表达式的定义的不清楚，例如 35+3* 这种也是后缀表达式。

Stack And Queue

1.3 铁路调度

- 题意：判断一个出栈顺序是否合法。
- 分析：模拟进栈出栈的过程。
- 维护一个数 x 表示下一个将要进栈的数（初始为 1），如果下一个读入的数 y 大于 x，则将 x 到 y 的数插入栈中，更新 x=y，同时将 ypop 出栈；否则只要栈的 top 元素不是 y，则说明非法。

Stack And Queue

2.3 车厢调度

- 题意：判断是否可以将初始队列调度成 1 到 n 的排列。
- 解法 1：模拟题目的意思，维护已经调度好的最后一个数 x，如果下一个数不是 x+1，则放到一个等待的队列中，否则更新 x=x+1，每次读取下一个数之前，在等待队列中将可以调度的编号 pop 掉，同时更新 x 的值。

Stack And Queue

2.3 车厢调度

- 题意：判断是否可以将初始队列调度成 1 到 n 的排列。
- 解法 2：问题可以等价于将序列拆成两个单调递增的序列，例如 1 3 4 2 5 可以拆成 1 2 和 3 4 5。我们可以用贪心的思想实现：维护两个数表示两个序列中最大的数，如果下一个读取的数比两个序列的最大的数都小，则无法加入哪个序列都无法满足条件；否则如果恰好只有一个数小于读取的数，则更新这个最大值；否则根据贪心的思想（尽可能给最后的数尽可能大的空间），我们应该将将大的一个数更新为读入的数。

Stack And Queue

2.3 车厢调度

- 题意：判断是否可以将初始队列调度成 1 到 n 的排列。
- 解法 3：问题可能还可以等价于是否存在三个数 i, j, k，满足 $i < j < k$ 同时 $a_i > a_j > a_k$，其中 a 为输入的序列。直接这样实现的复杂度是 $O(N^3)$，但是我们可以枚举 j，同时维护 $pre_i = max\{a_i | 1 \leq i \leq i\}$，$post_i = min\{a_i | i \leq i \leq n\}$，其中 $pre_0 = 0, post_{n+1} = INF$，则等价于判断时候存在 j，满足 $pre_{j-1} > a_j > post_{j+1}$，这样复杂度是 $O(N)$。

Stack And Queue

栈的实现评分

基础分：

- 100 分：对函数进行封装：class/struct/namespace。
- 95 分：写若干函数实现。
- 90 分：只有 main 函数。
- 85 分：使用 STL 完成。
- 代码中有 bug，会相应扣分。

加分：

- 提前交
- 面向对象 OO 设计
- try-catch 异常处理
- linked-stack
- ...

Stack And Queue

栈的实现

Listing 22: **bug**

```
template <typename T>
class stack {
private:
        T dat[maxsize];
        T p = 0;
public:
        int push(T x);
        int pop();
        int size() {
                return p;
        }
        bool empty();
        T top();
        void clear();
};
```

Stack And Queue

栈的实现

- T p = 0; 这句话应该是 int p = 0;
- 在 STL 实现中 push() 和 pop() 函数应该返回 void，top() 函数应该返回数据。

Complexity Analysis

Definition

- $f(x) \in O(g(x)) : \exists c > 0, \exists x_0 >> 2, \forall x \geq x_0, 0 \leq f(x) \leq c * g(x)$
- $f(x) \in o(g(x)) : \forall c > 0, \exists x_0 >> 2, \forall x \geq x_0, 0 \leq f(x) < c * g(x)$
- $f(x) \in \Omega(g(x)) : \exists c > 0, \exists x_0 >> 2, \forall x \geq x_0, 0 \leq c * g(x) \leq f(x)$
- $f(x) \in \omega(g(x)) : \forall c > 0, \exists x_0 >> 2, \forall x \geq x_0, 0 \leq c * g(x) < f(x)$
- $f(x) \in \Theta(g(x)) : \exists c_1, c_2 > 0, \exists x_0 >> 2, \forall x \geq x_0, 0 \leq c_1 * g(x) \leq f(x) \leq c_2 * g(x)$

Complexity Analysis

Definition

- 定义的书写要严谨：“O 是小于等于的意思”这样的回答只是描述而不是定义。
- 不要使用极限定义复杂度，因为极限可以不存在。

Exercise

- 给定一个长度为 N **排好序的**数组和两个数 L 和 R，要求在 $log(N)$ 的求出有多少个数介于 $[L, R]$ 之间。
- 提示：巧妙使用 lower_bound 和 upper_bound。
- 思考：是否需要做特殊情况的判断?
- **请大家在下课之前提交在超星上，将作为这节课的签到！**