数据结构与算法 STL 应用

陈宇琪

2020 年 3 月 21 日

摘要

主要内容包括 STL 的应用。

提交方式：在超星上提交第二部分第 1,3 题，在 EOJ 上完成三道练习题。

ddl：2020-03-25

目录

1 STL 介绍

1.1 vector

Vectors Vectors are sequence containers representing arrays that can change in size.

注意：vector 的 erase() 的复杂度不是 $O(1)$。

1.2 queue

Queues Queue are a type of container adaptor, specifically designed to operate in a FIFO context (first-in first-out), where elements are inserted into one end of the container and extracted from the other.

1.3 stack

Stacks Stacks are a type of container adaptor, specifically designed to operate in a LIFO context (last-in first-out), where elements are inserted and extracted only from one end of the container.

1.4 set/multiset

Sets Sets are containers that store unique elements following a specific order.In a set, the value of an element also identifies it (the value is itself the key, of type T), and each value must be unique. The value of the elements in a set cannot be modified once in the container (the elements are always const), but they can be inserted or removed from the container.Internally, the elements in a set are always **sorted** following **a specific strict weak ordering criterio**n indicated by its internal comparison object (of type Compare).Sets are typically implemented as **binary search trees**.

注意：lower_bound() 和 upper_bound() 函数复杂度是 $O(log(N))$，区别是 stl 的 lower_bound() 函数：

set 的 lower_bound() 函数使用方式：s.lower_bound(1)

stl 的 lower_bound() 函数使用方式：lower_bound(v.begin(),v.end(),1)/lower_bound(a,a+n,1)

其实 v 表示一个 vector，s 表示一个 set，a 表示一个数组。

Multisets Multisets are containers that store elements following a specific order, and where multiple elements can have equivalent values.

1.5 priority queue

Priority queues Priority queues are a type of container adaptors, specifically designed such that its first element is always the greatest of the elements it contains, according to some strict weak ordering criterion. This context is similar to a heap, where elements can be inserted at any moment, and only the max heap element can be retrieved (the one at the top in the priority queue).

1.6 map/unordered_map

Maps Maps are associative containers that store elements formed by a combination of a key value and a mapped value, following a specific order.

Unordered Maps Unordered maps are associative containers that store elements formed by the combination of a key value and a mapped value, and which allows for fast retrieval of individual elements based on their keys. In an unordered_map, the key value is generally used to uniquely identify the element, while

the mapped value is an object with the content associated to this key. Types of key and mapped value may differ.

注意：区别于 map，unordered_map 是基于哈希实现的数据结构，在查询效率上可以达到近似 $O(1)$，但是 unordered_map 中的数据是无序的。

2 编程练习

1、熟练各种 STL 容器的使用（参考附录中框出的函数），并回答以下问题：

- 如何定义一个 int 类型的 vector。
- 如何遍历一个 set 中的所有元素并输出。
- 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）。
- 定义一个 int 类型的 set，并在定义的 set 中删除一个元素 1。
- 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）。
- 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 1e5，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）。
- 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？
- 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？
- 利用 stl 函数实现离散化算法（参考下面的定义和解释）。

  加分项：（1）使用多种方式实现最后一小问；（2）分析这几种方式的利弊（代码长度/效率...）

注意：以上题目不需要写完整代码！

2、完成 EOJ 上的题目：3.1-3.2 和“代码查重”，**3.3 为加分题**。

3、（提交完整代码）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 $O(n \times log(K))$。

解释：例如数组为 $[1,3,2,4,5]$，$K=2$，则连续 K 个数的区间有 $[1,3],[3,2],[2,4],[4,5]$，每个区间的最大值分别为 $3,3,4,5$，所以输出 $3,3,4,5$。

离散化

离散化，**把无限空间中有限的个体映射到有限的空间中去，以此提高算法的时空效率。**

为什么要离散化？当以权值为下标的时候，有时候值太大，存不下。所以把要离散化的每一个数组里面的数映射到另一个值小一点的数组里面去。

打个比方，某个题目告诉你有 10^4 个数，每个数大小不超过 10^{10}，要你对这些数进行操作，那么肯定不能直接开 10^{10} 大小的数组，但是 10^4 的范围就完全没问题。

通俗的说，离散化是在不改变数据相对大小的条件下，对数据进行相应的缩小。例如：

原数据：$1, 999, 100000, 15$；处理后：$1, 3, 4, 2$。

但是离散化仅适用于只关注元素之间的大小关系而不关注元素本身的值!

离散化的数学定义

构造一个函数 $f(x)$，假设原数组为 $a_1, a_2, ..., a_n$，离散化后的数组为 $b_1, b_2, ..., b_n$，则满足以下条件：

- $\forall 1 \leq i \leq n$，$b_i = f(a_i)$。
- 如果 $a_i < a_j$ 成立，则 $b_i < b_j$ 也成立。
- 如果 $a_i = a_j$ 成立，则 $b_i = b_j$ 也成立。
- 如果 $a_i > a_j$ 成立，则 $b_i > b_j$ 也成立。
- $max\{b_i | 1 \leq i \leq n\}$ 尽可能小。
- 根据个人的喜好，$min\{b_i | 1 \leq i \leq n\}$ 可以是 0，也可以是 1。

3 参考答案

3.1 熟练 STL 的应用

1、如何定义一个 int 类型的 vector。

Listing 1: **ans1.1**

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

2、如何遍历一个 set 中的所有元素并输出。

Listing 2: **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';
}
```

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 3: **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;
}
```

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 4: **ans1.4**

```
set<int> a;

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

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 5: **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);
}

// or you can simply write like this
multiset<int> a;
multiset<int>::iterator it = a.find(1);
if (it != a.end()) {
        a.erase(it);
}
```

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 1e5，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 6: **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);
}
```

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 7: **ans1.7**

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

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 8: **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、利用 stl 函数实现离散化算法。

提供两种做法，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 $O(nlog(n))$。

Listing 9: **ans1.9**

```
// Appoach 1: vector + sort + unique + lower_bound
#include <bits/stdc++.h>
using namespace std;
const int maxn = 1e6+5;
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;
        }
}

// Appoach 2: map
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e6+5;
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]];
        }
}
```

3.2 前 K 大问题

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 $O(n \times log(K))$。

解释：例如数组为 $[1,3,2,4,5]$，$K=2$，则连续 K 个数的区间有 $[1,3],[3,2],[2,4],[4,5]$，每个区间的最大值分别为 $3,3,4,5$，所以输出 $3,3,4,5$。

分析 1：维护一个大小为 K 的 multiset，注意 set 是错的！

分析 2：使用 priority_queue，在 pq 中记录每个数的位置，每次不在区间中的数 pop 掉，但是这样可能 pq 中的数会大于 K 个，所以复杂度可以认为是 $O(n \times log(n))$ 的。

Listing 10: **ans2**

```
// Appoach 1: multiset
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;

int n,k,a[maxn];
multiset<int> ms;

int main() {
        cin >> n >> k;
        if (k > n || k == 0) {
                cerr << "Input Error" << endl;
        }
        for (int i = 0; i < n; i++) {
                cin >> a[i];
        }
        for (int i = 0; i < k; i++) {
                ms.insert(a[i]);
        }
        cout << *(--ms.end()) << ' ';
        for (int i = k; i < n; i++) {
                ms.erase(ms.lower_bound(a[i - k]));
                ms.insert(a[i]);
                cout << *(--ms.end()) << ' ';
        }
```

```
        cout << endl;
}

// Appoach 2: priority_queue
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;

int n,k,a[maxn];
priority_queue<pair<int,int>,vector<pair<int,int> > > pq;

int main() {
        cin >> n >> k;
        if (k > n || k == 0) {
                cerr << "Input Error" << endl;
        }
        for (int i = 0; i < n; i++) {
                cin >> a[i];
        }
        for (int i = 0; i < k; i++) {
                pq.push(make_pair(a[i], i));
        }
        cout << pq.top().first << ' ';
        for (int i = k; i < n; i++) {
                while (!pq.empty() && pq.top().second <= i - k) {
                        pq.pop();
                }
                pq.push(make_pair(a[i], i));
                cout << pq.top().first << ' ';
        }
        cout << endl;
}
```

3.3 前 K 大问题 2

前 K 大问题 2：给定一个长度为 n 数组，求每个长度至少为 K 的前缀的前 K 大的数之和，复杂度要求 $O(n \times log(K))$

解释：例如数组为 $[1,3,2,4,5]$，$K=2$，则长度大于等于 K 的前缀有 $[1,3],[1,3,2],[1,3,2,4],[1,3,2,4,5]$，每个区间的前 K 大值的和分别为 $4,5,7,9$，所以输出 $4,5,7,9$。

分析：维护一个大小为 K 的小根堆，如果插入的数比堆中最小的数小，则更新堆。

Listing 11: **ans2.2**

```
// priority_queue
// Sorry for mistake, pq can only calculate the sum of top K element in such pre-subarray
#include <bits/stdc++.h>
using namespace std;
const int maxn=1e5+5;

int n,k,a[maxn];
```

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

int main() {
	cin >> n >> k;
	if (k > n || k == 0) {
		cerr << "Input Error" << endl;
	}
	for (int i = 0; i < n; i++) {
		cin >> a[i];
	}
	long long sum = 0;
	for (int i = 0; i < k; i++) {
		pq.push(a[i]);
		sum += a[i];
	}
	cout << sum << ' ';
	for (int i = k; i < n; i++) {
		// note that pq can not be empty!
		if (pq.top() < a[i]) {
			sum -= pq.top();
			pq.pop();
			pq.push(a[i]);
			sum += a[i];
		}
		cout << sum << ' ';
	}
	cout << endl;
}
```

4 附录

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 1: Vector Reference

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 2: Queue Reference

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 3: Stack Reference

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 4: Set Reference

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 5: Priority Queue Reference

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 6: Map Reference