C++ 高级数据结构

ZeitHaum

2023 年 6 月 28 日

目录

1 greater

用于比较的函数对象,定义于 <functional>。类似的还有 less,greater_equal,less_equal。

使用时，需要先确保对应对象已经实现了相应的比较方法。注意: 重载方法的参数一定是 const 函数, 参数变量也要被 const 修饰。

例子:

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

struct student{
    int math_score;
    int PE_score;
    bool operator>(const student& s1)const{
        if(this->math_score!=s1.math_score) return this->
            math_score > s1.math_score;
        else return this->PE_score>s1.PE_score;
    }
};


int main(){
    vector<student> student_scores
        {{80,100},{90,70},{80,99},{70,100}};//初始化列表的嵌套
        使用
    greater<student> g;
    auto print = [](student i){
        cout<<i.math_score<<" "<<i.PE_score<<endl;
    };
    sort(student_scores.begin(),student_scores.end(),g);
    for_each(student_scores.begin(),student_scores.end(),print)
        ;
}
```

2 set

2.1 概述

set 中的元素都是唯一的，且以某种特定的顺序排列。set 中的元素不可以更改，但是是可以插入或者移除，并且具有对数复杂度。set 常常使用红黑树实现。set 默认按升序排列 (比较函数对象为 less<T>)，使用 greater<T> 可以更改为降序排列。

set 和 vector 可以通过传入迭代器的方式互转，相关例子见**??**.

2.2 multiset

与 set 类似，区别在于允许元素重复。

2.3 algorithm

Algorithm 库中提供了求取 set 求交集、并集、差集和对称差集等函数。分别对应 set_intersection,set_union,set_difference,set_symmetric_difference.

例子:

```
1  class Solution {
2  public:
3      vector<int> intersection(vector<int>& nums1, vector<int>&
           nums2) {
4          set<int> s1{nums1.begin(),nums1.end()};
5          set<int> s2{nums2.begin(),nums2.end()};
6          set<int>intersec;
7          set_intersection(s1.begin(),s1.end(),s2.begin(),s2.end
               (),inserter(intersec,intersec.begin()));
8          vector<int>ans{intersec.begin(),intersec.end()};
9          return ans;
10     }
11 };
```

该算法也对 multiset 具备一样的效果，不同的是返回的集合中元素也会出现多次，和其运算的定义相同。如交集集合元素重复出现的次数等于在两个集合中出现的次数的较小值。

例子:

```cpp
class Solution {
public:
    vector<int> intersect(vector<int>& nums1, vector<int>&
        nums2) {
        multiset<int>s1{nums1.begin(),nums1.end()};
        multiset<int>s2{nums2.begin(),nums2.end()};
        multiset<int>intersec;
        set_intersection(s1.begin(),s1.end(),s2.begin(),s2.end
            (),inserter(intersec,intersec.begin()));
        vector<int> ans{intersec.begin(),intersec.end()};
        return ans;
    }
};
```

2.4 例题

2.4.1 力扣-丑数 II

链接:力扣-265

记丑数序列为 $U_n = \{u_1, u_2, ..., u_n\}$, 且对于任意 $i < j$,$u_i < u_j$。显然, 对于任意丑数 u_i, 其可以写为 $u_i = 2^x 3^y 5^z (x, y, z \in \mathbb{N})$。于是对于任意丑数 u_i, 必然存在丑数 u_j, 使得 $u_j < u_i$ 且 $u_i = 2u_j$ 或者 $u_i = 3u_j$ 或者 $u_i = 5u_j$。

这样我们便可以维护一个待选集 S_n, 其元素为前 $n-1$ 个丑数的 2 倍数、3 倍数和 5 倍数。此时第 n 个丑数便是 S_n 的第 n 小数, 也即是 $S_n - U_{n-1}$ 的最小值。随着 n 变化, 我们可以通过动态删除 S_n 最小值和插入第 n 个丑数的 2 倍数、3 倍数、5 倍数来维护 $S_n - U_{n-1}$. 此操作可以非常简单地通过 set 实现。

代码:

```cpp
class Solution {
public:
    int nthUglyNumber(int n) {
        int cnt = 1;
        set<long long> s;
        s.insert(1ll);
        while(true){
            if(cnt==n) break;
```

```
9               long long minS = *s.begin();
10              s.insert(minS*2);
11              s.insert(minS*3);
12              s.insert(minS*5);
13              s.erase(s.begin());
14              cnt++;
15          }
16          return (int)(*s.begin());
17      }
18  };
```

3 优先队列

优先队列是类似于栈或者队列的抽象数据结构。每个优先队列中的元素均有一个优先级。与普通队列不同，高优先级的元素将比低优先级的元素先出队列。优先队列常常使用堆实现，但是也可以使用其他数据结构实现。

3.1 C++ 实现

C++ queue 库提供了插入和取出复杂度为 $\Theta(\log(n))$ 的优先队列，其中默认设置优先级为元素的大小，即元素大的将会被优先出队列。使用 greater<T> 可以得到另外一种对称的优先队列。C++ 中 priority_queue 使用堆实现，与自己写堆相比，封装的 priority_queue 可以避免因为操作失误导致堆失效。除了需要提供类模板外，priority_queue 还需要提供实现的数据类型，其必须具备 front(),pop_back() 和 push_back() 函数。C++ 标准库中 vector 和 dequeue 满足要求。

3.2 例题

3.2.1 力扣-数组中的第 k 大元素

链接:力扣-215. 此题只需维护一个大小为 k 的优先队列即可，并设置将小元素的优先设置为最高即可。最终的答案便是优先队列的最顶端。

代码:

```
1  class Solution {
2  public:
```

```
    int findKthLargest(vector<int>& nums, int k) {
        int n = nums.size();
        priority_queue<int,vector<int>,greater<int>> pq;
        int i = 0;
        for(;i<k;i++) pq.push(nums[i]);
        for(;i<n;i++){
            pq.push(nums[i]);
            pq.pop();
        }
        return pq.top();
    }
};
```

3.2.2 力扣-前 K 个高频元素

链接:力扣-347

上一题的进阶版，将优先级改为对应元素的频数即可。注意考虑此过程中和其他数据结构的配合使用。

```
class Solution {
public:
    struct element{
        int val;
        int frequence;
        bool operator>(const element& e) const{
            if(this->frequence==e.frequence) return this->val>e
                .val;
            return this->frequence>e.frequence;
        }
        bool operator==(const element& e) const{
            return this->frequence == e.frequence && this->val
                == e.val;
        }
        bool operator<(const element& e) const{
            if(this->frequence==e.frequence) return this->val<e
                .val;
            return this->frequence<e.frequence;
        }
    };
```

```
18
19      vector<int> topKFrequent(vector<int>& nums, int k) {
20          int n = nums.size();
21          map<int,int> freq;
22          for(int i = 0;i<n;i++){
23              if(freq.count(nums[i])==0) freq[nums[i]] = 1;
24              else freq[nums[i]]++;
25          }
26          set<element>s;
27          for(int i = 0;i<n;i++){
28              element e{nums[i],freq[nums[i]]};
29              s.insert(e);
30          }
31          priority_queue<element,vector<element>,greater<element
                >> pq;
32          auto p = s.begin();
33          for(int i = 0;i<k;i++,p++){
34              if(p==s.end()) break;
35              pq.push(*p);
36          }
37          while(p!=s.end()){
38              pq.push(*p);
39              pq.pop();
40              p++;
41          }
42          vector<int>ans(0);
43          while(!pq.empty()){
44              ans.push_back(pq.top().val);
45              pq.pop();
46          }
47          return ans;
48      }
49  };
```

4 双端链表

4.1 基本概述

包含在头文件 <list> 中，其迭代器是双向迭代器。只能 ++, 不能 +n。

4.2 迭代器不变特性

对于双向链表创建的元素，其对应的迭代器一经赋值直至销毁都不会改变。

见以下例子:

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

int main(){
    list<int> a;
    a.emplace_front(1);
    auto a1 = a.begin();// a1 -> 1
    a.emplace_front(2);
    a.emplace_back(3); //链表结构: 2->1 -> 3
    auto a2 = a.begin(); // a2 -> 2
    auto a3 = a1++; // a3 -> 3
    a.erase(a1); //链表结构: 2 -> 3

    //Check iterator Unchanged.
    cout<<(a.begin()==a2);
    auto new_a3 = ++a.begin();
    cout<<(new_a3 == a3);

}
```

5 智能指针

5.1 概述

可以自动调用析构函数，避免内存泄露。

C++ 98 引入了 auto$_ptr$, C++ $unique_ptr, weak_ptr, shared_ptr, auto_ptr$

5.2 用法

见以下示例:

```
#include <memory>//智能指针头文件
#include <bits/stdc++.h>
using namespace std;

struct test{
    int id;
};

int main(){
    unique_ptr<int> int_ptr = unique_ptr<int>(new int()); //构造函数
    cout<<(*int_ptr); //获取左值对象
    unique_ptr<test> test_ptr = unique_ptr<test>(new test());
    cout<<test_ptr->id;
}
```

、

注意给左值赋值智能指针、左值构造智能指针、new 智能指针都是未定义行为。

5.3 区别

unique_ptr 只允许单个指针对象管理这片内存 (互斥)。shared_ptr 支持多个指针对象管理，并可以使用 use_count 方法查看对象数。week_ptr 和 shared_ptr 共同作用，解决内存锁死的情况，一般不需要考虑。

6 tuple

6.1 基本概述

多元组。

6.2 基本用法

模板: tuple<Type t1, ...>;

已经重定义赋值类方法和偏序方法,但是偏序方法 (除了 ==) 在 C++20 被移除。

获取其第 i 个元素，get<i>(实例名)

6.3 示例代码

(力扣 15).

```
class Solution {
public:
    unordered_map<int,int> hash;
     //double find
    unordered_map<int,int> cnt;
    vector<int> nums_;
    vector<vector<int>> threeSum(vector<int>& nums) {
        //a number have more than 2 is useless(zero for 3).
        for(int i = 0;i<nums.size();i++){
            if(cnt.count(nums[i])==0){
                cnt[nums[i]] = 1;
                nums_.push_back(nums[i]);
            }
            else if(cnt[nums[i]]>=1 &&cnt[nums[i]]<=2){
                cnt[nums[i]]++;
                nums_.push_back(nums[i]);
            }
            else if(cnt[nums[i]]==3 && nums[i]==0){
                cnt[nums[i]]++;
                nums_.push_back(nums[i]);
            }
        }
        //build hash
        for(int i = 0;i<nums_.size();i++){
            hash[nums_[i]] = i;
        }
        set<tuple<int,int,int>> ret_set;
        for(int i = 0;i<nums_.size();i++){
            for(int j = i+1;j<nums_.size();j++){
                if(hash.count(-nums_[i] - nums_[j])!=0){
                    int k = hash[-nums_[i] - nums_[j]];
```

```
32                      if(k>j){
33                          vector<int> vec = {nums_[i], nums_[j],
                                nums_[k]};
34                          sort(vec.begin(), vec.end());
35                          ret_set.insert({vec[0],vec[1],vec[2]});
36                      }
37                  }
38              }
39          }
40
41          vector<vector<int>> ret;
42          for(tuple<int,int,int> x : ret_set){
43              ret.push_back({get<0>(x), get<1>(x), get<2>(x)});
44          }
45
46          return ret;
47      }
48  };
```

参考资料

[1].Cplusplus-set

[2].CppReference-set

[3].Wiki-PriorityQueue

[4].CppReference-priority_queue