第八章 C++ STL

STL是提高C++编写效率的一个利器。

——闫学灿

1. #include <vector>

vector是变长数组，支持随机访问，不支持在任意位置O(1)插入。为了保证效率，元素的增删一般应该在末尾进行。

声明

#include <vector> 头文件

vector<int> a; 相当于一个长度动态变化的int数组

vector<int> b[233]; 相当于第一维长233，第二位长度动态变化的int数组

struct rec{…};

vector<rec> c; 自定义的结构体类型也可以保存在vector中

size/empty

size函数返回vector的实际长度（包含的元素个数），empty函数返回一个bool类型，表明vector是否为空。二者的时间复杂度都是O(1)。

所有的STL容器都支持这两个方法，含义也相同，之后我们就不再重复给出。

clear

clear函数把vector清空。

迭代器

迭代器就像STL容器的“指针”，可以用星号“\*”操作符解除引用。

一个保存int的vector的迭代器声明方法为：

vector<int>::iterator it;

vector的迭代器是“随机访问迭代器”，可以把vector的迭代器与一个整数相加减，其行为和指针的移动类似。可以把vector的两个迭代器相减，其结果也和指针相减类似，得到两个迭代器对应下标之间的距离。

begin/end

begin函数返回指向vector中第一个元素的迭代器。例如a是一个非空的vector，则\*a.begin()与a[0]的作用相同。

所有的容器都可以视作一个“前闭后开”的 结构，end函数返回vector的尾部，即第n个元素再往后的“边界”。\*a.end()与a[n]都是越界访问，其中n=a.size()。

下面两份代码都遍历了vector<int>a，并输出它的所有元素。

for (int I = 0; I < a.size(); I ++) cout << a[i] << endl;

for (vector<int>::iterator it = a.begin(); it != a.end(); it ++) cout << \*it << endl;

front/back

front函数返回vector的第一个元素，等价于\*a.begin() 和 a[0]。

back函数返回vector的最后一个元素，等价于\*==a.end() 和 a[a.size() – 1]。

push\_back() 和 pop\_back()

a.push\_back(x) 把元素x插入到vector a的尾部。

b.pop\_back() 删除vector a的最后一个元素。

1. #include <queue>

头文件queue主要包括循环队列queue和优先队列priority\_queue两个容器。

声明

queue<int> q;

struct rec{…}; queue<rec> q; //结构体rec中必须定义小于号

priority\_queue<int> q; // 大根堆

priority\_queue<int, vector<int>, greater<int> q; // 小根堆

priority\_queue<pair<int, int>>q;

循环队列 queue

push 从队尾插入

pop 从队头弹出

front 返回队头元素

back 返回队尾元素

优先队列 priority\_queue

push 把元素插入堆

pop 删除堆顶元素

top 查询堆顶元素（最大值）

1. #include <stack>

头文件stack包含栈。声明和前面的容器类似。

push 向栈顶插入

pop 弹出栈顶元素

1. #include <deque>

双端队列deque是一个支持在两端高效插入或删除元素的连续线性存储空间。它就像是vector和queue的结合。与vector相比，deque在头部增删元素仅需要O(1)的时间；与queue相比，deque像数组一样支持随机访问。

[] 随机访问

begin/end，返回deque的头/尾迭代器

front/back 队头/队尾元素

push\_back 从队尾入队

push\_front 从队头入队

pop\_back 从队尾出队

pop\_front 从队头出队

clear 清空队列

1. #include <set>

头文件set主要包括set和multiset两个容器，分别是“有序集合”和“有序多重集合”，即前者的元素不能重复，而后者可以包含若干个相等的元素。set和multiset的内部实现是一棵红黑树，它们支持的函数基本相同。

声明

set<int> s;

struct rec{…}; set<rec> s; // 结构体rec中必须定义小于号

multiset<double> s;

size/empty/clear

与vector类似

迭代器

set和multiset的迭代器称为“双向访问迭代器”，不支持“随机访问”，支持星号(\*)解除引用，仅支持”++”和--“两个与算术相关的操作。

设it是一个迭代器，例如set<int>::iterator it;

若把it++，则it会指向“下一个”元素。这里的“下一个”元素是指在元素从小到大排序的结果中，排在it下一名的元素。同理，若把it--，则it将会指向排在“上一个”的元素。

begin/end

返回集合的首、尾迭代器，时间复杂度均为O(1)。

s.begin() 是指向集合中最小元素的迭代器。

s.end() 是指向集合中最大元素的下一个位置的迭代器。换言之，就像vector一样，是一个“前闭后开”的形式。因此--s.end()是指向集合中最大元素的迭代器。

insert

s.insert(x)把一个元素x插入到集合s中，时间复杂度为O(logn)。

在set中，若元素已存在，则不会重复插入该元素，对集合的状态无影响。

find

s.find(x) 在集合s中查找等于x的元素，并返回指向该元素的迭代器。若不存在，则返回s.end()。时间复杂度为O(logn)。

lower\_bound/upper\_bound

这两个函数的用法与find类似，但查找的条件略有不同，时间复杂度为 O(logn)。

s.lower\_bound(x) 查找大于等于x的元素中最小的一个，并返回指向该元素的迭代器。

s.upper\_bound(x) 查找大于x的元素中最小的一个，并返回指向该元素的迭代器。

erase

设it是一个迭代器，s.erase(it) 从s中删除迭代器it指向的元素，时间复杂度为O(logn)

设x是一个元素，s.erase(x) 从s中删除所有等于x的元素，时间复杂度为O(k+logn)，其中k是被删除的元素个数。

count

s.count(x) 返回集合s中等于x的元素个数，时间复杂度为 O(k +logn)，其中k为元素x的个数。

1. #include <map>

map容器是一个键值对key-value的映射，其内部实现是一棵以key为关键码的红黑树。Map的key和value可以是任意类型，其中key必须定义小于号运算符。

声明

map<key\_type, value\_type> name;

例如：

map<long, long, bool> vis;

map<string, int> hash;

map<pair<int, int>, vector<int>> test;

size/empty/clear/begin/end均与set类似。

Insert/erase

与set类似，但其参数均是pair<key\_type, value\_type>。

find

h.find(x) 在变量名为h的map中查找key为x的二元组。

[]操作符

h[key] 返回key映射的value的引用，时间复杂度为O(logn)。

[]操作符是map最吸引人的地方。我们可以很方便地通过h[key]来得到key对应的value，还可以对h[key]进行赋值操作，改变key对应的value。