



STL容器成员函数速查表

一、序列容器

1.1 vector

成员函数	作用	时间复杂度
<code>vector()</code>	默认构造函数，创建空vector	$O(1)$
<code>vector(size_type n)</code>	构造包含n个默认初始化元素的vector	$O(n)$
<code>vector(size_type n, const T& value)</code>	构造包含n个value的vector	$O(n)$
<code>vector(const vector& other)</code>	拷贝构造函数	$O(n)$
<code>vector(vector&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~vector()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$ (如果大小不同)
<code>assign(size_type n, const T& value)</code>	替换为n个value	$O(n)$
<code>assign(InputIt first, InputIt last)</code>	替换为[first, last)的元素	$O(n)$
<code>at(size_type pos)</code>	访问指定位置，带边界检查	$O(1)$
<code>operator[](size_type pos)</code>	访问指定位置，无边界检查	$O(1)$
<code>front()</code>	访问第一个元素	$O(1)$
<code>back()</code>	访问最后一个元素	$O(1)$
<code>data()</code>	返回指向底层数组的指针	$O(1)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>reserve(size_type new_cap)</code>	预留存储空间	$O(n)$ (如果需要重新分配)
<code>capacity()</code>	返回当前分配的存储容量	$O(1)$

成员函数	作用	时间复杂度
<code>shrink_to_fit()</code>	请求移除未使用的容量	$O(n)$ (可能重新分配)
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const_iterator pos, const T& value)</code>	在指定位置前插入元素	$O(n)$
<code>insert(const_iterator pos, size_type count, const T& value)</code>	插入count个value	$O(n + count)$
<code>insert(const_iterator pos, InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(n + \text{distance}(first, last))$
<code>emplace(const_iterator pos, Args&&... args)</code>	在pos前就地构造元素 (C++11)	$O(n)$
<code>erase(const_iterator pos)</code>	删除指定位置元素	$O(n)$
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(n)$
<code>push_back(const T& value)</code>	尾部添加元素	$O(1)$ (均摊)
<code>push_back(T&& value)</code>	尾部移动添加元素 (C++11)	$O(1)$ (均摊)
<code>emplace_back(Args&&... args)</code>	尾部就地构造元素 (C++11)	$O(1)$ (均摊)
<code>pop_back()</code>	删除尾部元素	$O(1)$
<code>resize(size_type count)</code>	改变大小, 新元素默认初始化	$O(count - \text{size})$
<code>resize(size_type count, const T& value)</code>	改变大小, 新元素初始化为value	$O(count - \text{size})$
<code>swap(vector& other)</code>	交换内容	$O(1)$

1.2 deque

成员函数	作用	时间复杂度
<code>deque()</code>	默认构造函数	$O(1)$
<code>deque(size_type n)</code>	构造包含n个默认初始化元素的deque	$O(n)$
<code>deque(size_type n, const T& value)</code>	构造包含n个value的deque	$O(n)$
<code>deque(const deque& other)</code>	拷贝构造函数	$O(n)$
<code>deque(deque&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~deque()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>assign(size_type n, const T& value)</code>	替换为n个value	$O(n)$

成员函数	作用	时间复杂度
<code>assign(InputIt first, InputIt last)</code>	替换为[first, last)的元素	$O(n)$
<code>at(size_type pos)</code>	访问指定位置, 带边界检查	$O(1)$
<code>operator[](size_type pos)</code>	访问指定位置, 无边界检查	$O(1)$
<code>front()</code>	访问第一个元素	$O(1)$
<code>back()</code>	访问最后一个元素	$O(1)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>shrink_to_fit()</code>	请求移除未使用的容量	至少 $O(n)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const_iterator pos, const T& value)</code>	在指定位置前插入元素	$O(n)$ (通常比vector快)
<code>insert(const_iterator pos, size_type count, const T& value)</code>	插入count个value	$O(n + count)$
<code>insert(const_iterator pos, InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(n + distance(first, last))$
<code>emplace(const_iterator pos, Args&&... args)</code>	在pos前就地构造元素 (C++11)	$O(n)$
<code>erase(const_iterator pos)</code>	删除指定位置元素	$O(n)$
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(n)$
<code>push_front(const T& value)</code>	头部添加元素	$O(1)$
<code>push_front(T&& value)</code>	头部移动添加元素 (C++11)	$O(1)$
<code>emplace_front(Args&&... args)</code>	头部就地构造元素 (C++11)	$O(1)$
<code>push_back(const T& value)</code>	尾部添加元素	$O(1)$
<code>push_back(T&& value)</code>	尾部移动添加元素 (C++11)	$O(1)$
<code>emplace_back(Args&&... args)</code>	尾部就地构造元素 (C++11)	$O(1)$

成员函数	作用	时间复杂度
<code>pop_front()</code>	删除头部元素	$O(1)$
<code>pop_back()</code>	删除尾部元素	$O(1)$
<code>resize(size_type count)</code>	改变大小	$O(count - size)$
<code>resize(size_type count, const T& value)</code>	改变大小并用value填充	$O(count - size)$
<code>swap(deque& other)</code>	交换内容	$O(1)$

1.3 list

成员函数	作用	时间复杂度
<code>list()</code>	默认构造函数	$O(1)$
<code>list(size_type n)</code>	构造包含n个默认初始化元素的list	$O(n)$
<code>list(size_type n, const T& value)</code>	构造包含n个value的list	$O(n)$
<code>list(const list& other)</code>	拷贝构造函数	$O(n)$
<code>list(list&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~list()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>assign(size_type n, const T& value)</code>	替换为n个value	$O(n)$
<code>assign(InputIt first, InputIt last)</code>	替换为[first, last)的元素	$O(n)$
<code>front()</code>	访问第一个元素	$O(1)$
<code>back()</code>	访问最后一个元素	$O(1)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const_iterator pos, const T& value)</code>	在指定位置前插入元素	$O(1)$
<code>insert(const_iterator pos, size_type count, const T&</code>	插入count个value	$O(count)$

成员函数	作用	时间复杂度
<code>value)</code>		
<code>insert(const_iterator pos, InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(\text{distance(first, last)})$
<code>emplace(const_iterator pos, Args&&... args)</code>	在pos前就地构造元素 (C++11)	$O(1)$
<code>erase(const_iterator pos)</code>	删除指定位置元素	$O(1)$
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(\text{distance(first, last)})$
<code>push_front(const T& value)</code>	头部添加元素	$O(1)$
<code>push_front(T&& value)</code>	头部移动添加元素 (C++11)	$O(1)$
<code>emplace_front(Args&&... args)</code>	头部就地构造元素 (C++11)	$O(1)$
<code>push_back(const T& value)</code>	尾部添加元素	$O(1)$
<code>push_back(T&& value)</code>	尾部移动添加元素 (C++11)	$O(1)$
<code>emplace_back(Args&&... args)</code>	尾部就地构造元素 (C++11)	$O(1)$
<code>pop_front()</code>	删除头部元素	$O(1)$
<code>pop_back()</code>	删除尾部元素	$O(1)$
<code>resize(size_type count)</code>	改变大小	$O(\text{count} - \text{size})$
<code>resize(size_type count, const T& value)</code>	改变大小并用value填充	$O(\text{count} - \text{size})$
<code>swap(list& other)</code>	交换内容	$O(1)$
<code>merge(list& other)</code>	合并两个已排序列表	$O(\text{size} + \text{other.size})$
<code>merge(list& other, Compare comp)</code>	按comp比较合并	$O(\text{size} + \text{other.size})$
<code>splice(const_iterator pos, list& other)</code>	将other的所有元素移动到pos前	$O(1)$
<code>splice(const_iterator pos, list& other, const_iterator it)</code>	将other的it元素移动到pos前	$O(1)$
<code>splice(const_iterator pos, list& other, const_iterator first, const_iterator last)</code>	将other的[first, last)元素移动到pos前	$O(1)$
<code>remove(const T& value)</code>	删除所有等于value的元素	$O(n)$
<code>remove_if(UnaryPredicate p)</code>	删除所有满足p的元素	$O(n)$
<code>reverse()</code>	反转列表	$O(n)$
<code>unique()</code>	删除连续重复元素	$O(n)$

成员函数	作用	时间复杂度
<code>unique(BinaryPredicate p)</code>	按p删除连续满足条件的元素	$O(n)$
<code>sort()</code>	排序	$O(n \log n)$
<code>sort(Compare comp)</code>	按comp排序	$O(n \log n)$

1.4 forward_list (C++11)

成员函数	作用	时间复杂度
<code>forward_list()</code>	默认构造函数	$O(1)$
<code>forward_list(size_type n)</code>	构造包含n个默认初始化元素的forward_list	$O(n)$
<code>forward_list(size_type n, const T& value)</code>	构造包含n个value的forward_list	$O(n)$
<code>forward_list(const forward_list& other)</code>	拷贝构造函数	$O(n)$
<code>forward_list(forward_list&& other)</code>	移动构造函数	$O(1)$
<code>~forward_list()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>assign(size_type n, const T& value)</code>	替换为n个value	$O(n)$
<code>assign(InputIt first, InputIt last)</code>	替换为[first, last)的元素	$O(n)$
<code>front()</code>	访问第一个元素	$O(1)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>cbefore_begin()</code>	返回第一个元素前的位置	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert_after(const_iterator pos, const T& value)</code>	在pos后插入元素	$O(1)$
<code>insert_after(const_iterator pos, size_type count, const T& value)</code>	在pos后插入count个value	$O(count)$
<code>insert_after(const_iterator pos, InputIt first, InputIt last)</code>	在pos后插入[first, last)的元素	$O(distance(first, last))$
<code>emplace_after(const_iterator pos, Args&&... args)</code>	在pos后就地构造元素	$O(1)$
<code>erase_after(const_iterator pos)</code>	删除pos后的元素	$O(1)$

成员函数	作用	时间复杂度
<code>erase_after(const_iterator first, const_iterator last)</code>	删除(first, last)的元素	$O(\text{distance}(first, last))$
<code>push_front(const T& value)</code>	头部添加元素	$O(1)$
<code>push_front(T&& value)</code>	头部移动添加元素	$O(1)$
<code>emplace_front(Args&&... args)</code>	头部就地构造元素	$O(1)$
<code>pop_front()</code>	删除头部元素	$O(1)$
<code>resize(size_type count)</code>	改变大小	$O(count - \text{size})$
<code>resize(size_type count, const T& value)</code>	改变大小并用value填充	$O(count - \text{size})$
<code>swap(forward_list& other)</code>	交换内容	$O(1)$
<code>merge(forward_list& other)</code>	合并两个已排序列表	$O(\text{size} + \text{other.size})$
<code>merge(forward_list& other, Compare comp)</code>	按comp比较合并	$O(\text{size} + \text{other.size})$
<code>splice_after(const_iterator pos, forward_list& other)</code>	将other的所有元素移动到pos后	$O(1)$
<code>splice_after(const_iterator pos, forward_list& other, const_iterator it)</code>	将other的it元素移动到pos后	$O(1)$
<code>splice_after(const_iterator pos, forward_list& other, const_iterator first, const_iterator last)</code>	将other的(first, last)元素移动到pos后	$O(1)$
<code>remove(const T& value)</code>	删除所有等于value的元素	$O(n)$
<code>remove_if(UnaryPredicate p)</code>	删除所有满足p的元素	$O(n)$
<code>reverse()</code>	反转列表	$O(n)$
<code>unique()</code>	删除连续重复元素	$O(n)$
<code>unique(BinaryPredicate p)</code>	按p删除连续满足条件的元素	$O(n)$
<code>sort()</code>	排序	$O(n \log n)$
<code>sort(Compare comp)</code>	按comp排序	$O(n \log n)$

1.5 array (C++11)

成员函数	作用	时间复杂度
<code>array()</code>	默认构造函数 (值初始化)	$O(n)$
<code>array(const array& other)</code>	拷贝构造函数	$O(n)$

成员函数	作用	时间复杂度
<code>array(array&& other)</code>	移动构造函数	$O(n)$
<code>~array()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>at(size_type pos)</code>	访问指定位置, 带边界检查	$O(1)$
<code>operator[](size_type pos)</code>	访问指定位置, 无边界检查	$O(1)$
<code>front()</code>	访问第一个元素	$O(1)$
<code>back()</code>	访问最后一个元素	$O(1)$
<code>data()</code>	返回指向底层数组的指针	$O(1)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>fill(const T& value)</code>	用 <code>value</code> 填充所有元素	$O(n)$
<code>swap(array& other)</code>	交换内容	$O(n)$

1.6 string

成员函数	作用	时间复杂度
<code>string()</code>	默认构造函数	$O(1)$
<code>string(const char* s)</code>	从C字符串构造	$O(n)$
<code>string(const string& other)</code>	拷贝构造函数	$O(n)$
<code>string(string&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~string()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>assign(size_type count, char ch)</code>	替换为 <code>count</code> 个 <code>ch</code>	$O(count)$
<code>assign(const char* s)</code>	用C字符串替换	$O(strlen(s))$

成员函数	作用	时间复杂度
<code>at(size_type pos)</code>	访问指定位置, 带边界检查	$O(1)$
<code>operator[](size_type pos)</code>	访问指定位置, 无边界检查	$O(1)$
<code>front()</code>	访问第一个字符	$O(1)$
<code>back()</code>	访问最后一个字符	$O(1)$
<code>data()</code>	返回指向底层数组的指针 (C++17前非const)	$O(1)$
<code>c_str()</code>	返回C风格字符串	$O(1)$
<code>begin()</code>	返回指向第一个字符的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个字符的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查是否为空	$O(1)$
<code>size()</code>	返回字符数	$O(1)$
<code>length()</code>	返回字符数	$O(1)$
<code>max_size()</code>	返回可容纳的最大字符数	$O(1)$
<code>reserve(size_type new_cap)</code>	预留存储空间	$O(n)$ (如果需要重新分配)
<code>capacity()</code>	返回当前容量	$O(1)$
<code>shrink_to_fit()</code>	请求移除未使用的容量	$O(n)$ (可能重新分配)
<code>clear()</code>	清除所有字符	$O(1)$
<code>insert(size_type pos, const string& str)</code>	在pos插入字符串	$O(n)$
<code>insert(const_iterator pos, char ch)</code>	在迭代器位置插入字符	$O(n)$
<code>erase(size_type pos = 0, size_type count = npos)</code>	删除字符	$O(n)$
<code>erase(const_iterator pos)</code>	删除迭代器位置的字符	$O(n)$
<code>push_back(char ch)</code>	尾部添加字符	$O(1)$ (均摊)
<code>pop_back()</code>	删除尾部字符	$O(1)$
<code>append(const string& str)</code>	尾部追加字符串	$O(str.size())$
<code>operator+=</code>	追加字符串或字符	$O(\text{追加内容大小})$
<code>compare(const string& str)</code>	比较字符串	$O(\min(\text{size}, \text{str.size}()))$

成员函数	作用	时间复杂度
<code>replace(size_type pos, size_type count, const string& str)</code>	替换子串	$O(n)$
<code>substr(size_type pos = 0, size_type count = npos)</code>	返回子串	$O(count)$
<code>copy(char* dest, size_type count, size_type pos = 0)</code>	复制到C字符串	$O(count)$
<code>resize(size_type count)</code>	改变大小	$O(count - size)$
<code>resize(size_type count, char ch)</code>	改变大小并用ch填充	$O(count - size)$
<code>swap(string& other)</code>	交换内容	$O(1)$
<code>find(const string& str, size_type pos = 0)</code>	查找子串	$O(n*m)$ (最坏情况)
<code>rfind(const string& str, size_type pos = npos)</code>	反向查找	$O(n*m)$ (最坏情况)
<code>find_first_of(const string& str, size_type pos = 0)</code>	查找str中任意字符首次出现	$O(n*m)$ (最坏情况)
<code>find_last_of(const string& str, size_type pos = npos)</code>	查找str中任意字符最后出现	$O(n*m)$ (最坏情况)
<code>find_first_not_of(const string& str, size_type pos = 0)</code>	查找不在str中的字符首次出现	$O(n*m)$ (最坏情况)
<code>find_last_not_of(const string& str, size_type pos = npos)</code>	查找不在str中的字符最后出现	$O(n*m)$ (最坏情况)

二、关联容器

2.1 set/multiset

成员函数	作用	时间复杂度
<code>set()</code>	默认构造函数	$O(1)$
<code>set(InputIt first, InputIt last)</code>	用[first, last)元素构造	$O(n \log n)$
<code>set(const set& other)</code>	拷贝构造函数	$O(n)$
<code>set(set&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~set()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$

成员函数	作用	时间复杂度
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const value_type& value)</code>	插入元素	$O(\log n)$
<code>insert(value_type&& value)</code>	移动插入元素 (C++11)	$O(\log n)$
<code>insert(const_iterator hint, const value_type& value)</code>	在hint附近插入	$O(1)$ (均摊, 如果hint正确)
<code>insert(InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(m \log (n+m))$
<code>emplace(Args&&... args)</code>	就地构造元素 (C++11)	$O(\log n)$
<code>emplace_hint(const_iterator hint, Args&&... args)</code>	在hint附近就地构造 (C++11)	$O(1)$ (均摊, 如果hint正确)
<code>erase(const_iterator pos)</code>	删除迭代器位置的元素	$O(1)$ (均摊)
<code>erase(const key_type& key)</code>	删除键为key的元素	$O(\log n)$
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(\log n + \text{distance}(first, last))$
<code>swap(set& other)</code>	交换内容	$O(1)$
<code>count(const key_type& key)</code>	返回键为key的元素数	$O(\log n)$
<code>find(const key_type& key)</code>	查找键为key的元素	$O(\log n)$
<code>equal_range(const key_type& key)</code>	返回匹配键的范围	$O(\log n)$
<code>lower_bound(const key_type& key)</code>	返回第一个不小于key的元素	$O(\log n)$
<code>upper_bound(const key_type& key)</code>	返回第一个大于key的元素	$O(\log n)$
<code>key_comp()</code>	返回键比较函数	$O(1)$
<code>value_comp()</code>	返回值比较函数	$O(1)$

注: `multiset`与`set`接口相同, 但允许重复元素

2.2 map/multimap

成员函数	作用	时间复杂度
<code>map()</code>	默认构造函数	$O(1)$
<code>map(InputIt first, InputIt last)</code>	用[first, last)元素构造	$O(n \log n)$
<code>map(const map& other)</code>	拷贝构造函数	$O(n)$
<code>map(map&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~map()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>at(const key_type& key)</code>	访问指定键, 带边界检查	$O(\log n)$
<code>operator[](const key_type& key)</code>	访问指定键, 不存在则插入 (仅map)	$O(\log n)$
<code>operator[](key_type&& key)</code>	移动键访问 (C++11, 仅map)	$O(\log n)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>rbegin()</code>	返回指向第一个元素的逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const value_type& value)</code>	插入元素	$O(\log n)$
<code>insert(value_type&& value)</code>	移动插入元素 (C++11)	$O(\log n)$
<code>insert(const_iterator hint, const value_type& value)</code>	在hint附近插入	$O(1)$ (均摊, 如果hint正确)
<code>insert(InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(m \log (n+m))$
<code>emplace(Args&&... args)</code>	就地构造元素 (C++11)	$O(\log n)$
<code>emplace_hint(const_iterator hint, Args&&... args)</code>	在hint附近就地构造 (C++11)	$O(1)$ (均摊, 如果hint正确)
<code>erase(const_iterator pos)</code>	删除迭代器位置的元素	$O(1)$ (均摊)
<code>erase(const key_type& key)</code>	删除键为key的元素	$O(\log n)$
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(\log n + distance(first, last))$

成员函数	作用	时间复杂度
<code>swap(map& other)</code>	交换内容	$O(1)$
<code>count(const key_type& key)</code>	返回键为key的元素数	$O(\log n)$
<code>find(const key_type& key)</code>	查找键为key的元素	$O(\log n)$
<code>equal_range(const key_type& key)</code>	返回匹配键的范围	$O(\log n)$
<code>lower_bound(const key_type& key)</code>	返回第一个不小于key的元素	$O(\log n)$
<code>upper_bound(const key_type& key)</code>	返回第一个大于key的元素	$O(\log n)$
<code>key_comp()</code>	返回键比较函数	$O(1)$
<code>value_comp()</code>	返回值比较函数	$O(1)$

注: `multimap`与`map`接口相似, 但没有`operator[]`, 且允许重复键

三、无序容器 (C++11)

3.1 `unordered_set/unordered_multiset`

成员函数	作用	时间复杂度
<code>unordered_set()</code>	默认构造函数	$O(1)$
<code>unordered_set(size_type bucket_count)</code>	指定桶数构造	$O(bucket_count)$
<code>unordered_set(InputIt first, InputIt last, size_type bucket_count = n)</code>	用[first, last)元素构造	$O(n)$ 平均, $O(n^2)$ 最坏
<code>unordered_set(const unordered_set& other)</code>	拷贝构造函数	$O(n)$
<code>unordered_set(unordered_set&& other)</code>	移动构造函数	$O(1)$
<code>~unordered_set()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const value_type& value)</code>	插入元素	$O(1)$ 平均, $O(n)$ 最坏

成员函数	作用	时间复杂度
<code>insert(value_type&& value)</code>	移动插入元素	$O(1)$ 平均, $O(n)$ 最坏
<code>insert(const_iterator hint, const value_type& value)</code>	在hint附近插入	$O(1)$ 平均, $O(n)$ 最坏
<code>insert(InputIt first, InputIt last)</code>	插入[first, last)的元素	$O(m)$ 平均, $O(m*n)$ 最坏
<code>emplace(Args&&... args)</code>	就地构造元素	$O(1)$ 平均, $O(n)$ 最坏
<code>emplace_hint(const_iterator hint, Args&&... args)</code>	在hint附近就地构造	$O(1)$ 平均, $O(n)$ 最坏
<code>erase(const_iterator pos)</code>	删除迭代器位置的元素	$O(1)$ 平均, $O(n)$ 最坏
<code>erase(const key_type& key)</code>	删除键为key的元素	$O(1)$ 平均, $O(n)$ 最坏
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(\text{distance}(first, last))$ 平均
<code>swap(unordered_set& other)</code>	交换内容	$O(1)$
<code>count(const key_type& key)</code>	返回键为key的元素数	$O(1)$ 平均, $O(n)$ 最坏
<code>find(const key_type& key)</code>	查找键为key的元素	$O(1)$ 平均, $O(n)$ 最坏
<code>equal_range(const key_type& key)</code>	返回匹配键的范围	$O(1)$ 平均, $O(n)$ 最坏
<code>bucket_count()</code>	返回桶数	$O(1)$
<code>max_bucket_count()</code>	返回最大桶数	$O(1)$
<code>bucket_size(size_type n)</code>	返回第n个桶的元素数	$O(1)$ 平均, $O(\text{bucket_size}(n))$ 最坏
<code>bucket(const key_type& key)</code>	返回键的桶索引	$O(1)$ 平均, $O(n)$ 最坏
<code>load_factor()</code>	返回负载因子	$O(1)$
<code>max_load_factor()</code>	返回最大负载因子	$O(1)$
<code>max_load_factor(float m1)</code>	设置最大负载因子	$O(1)$
<code>rehash(size_type count)</code>	设置桶数至少为count	$O(n)$ 平均, $O(n^2)$ 最坏
<code>reserve(size_type count)</code>	预留空间, 使可容纳count个元素	$O(n)$ 平均, $O(n^2)$ 最坏
<code>hash_function()</code>	返回哈希函数	$O(1)$
<code>key_eq()</code>	返回键相等比较函数	$O(1)$

3.2 unordered_map/unordered_multimap

成员函数	作用	时间复杂度
<code>unordered_map()</code>	默认构造函数	$O(1)$

成员函数	作用	时间复杂度
<code>unordered_map(size_type bucket_count)</code>	指定桶数构造	$O(bucket_count)$
<code>unordered_map(InputIt first, InputIt last, size_type bucket_count = n)</code>	用 [first, last) 元素构造	$O(n)$ 平均, $O(n^2)$ 最坏
<code>unordered_map(const unordered_map& other)</code>	拷贝构造函数	$O(n)$
<code>unordered_map(unordered_map&& other)</code>	移动构造函数	$O(1)$
<code>~unordered_map()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>at(const key_type& key)</code>	访问指定键, 带边界检查	$O(1)$ 平均, $O(n)$ 最坏
<code>operator[](const key_type& key)</code>	访问指定键, 不存在则插入 (仅 <code>unordered_map</code>)	$O(1)$ 平均, $O(n)$ 最坏
<code>operator[](key_type&& key)</code>	移动键访问 (仅 <code>unordered_map</code>)	$O(1)$ 平均, $O(n)$ 最坏
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>empty()</code>	检查容器是否为空	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$
<code>max_size()</code>	返回可容纳的最大元素数	$O(1)$
<code>clear()</code>	清除所有元素	$O(n)$
<code>insert(const value_type& value)</code>	插入元素	$O(1)$ 平均, $O(n)$ 最坏
<code>insert(value_type&& value)</code>	移动插入元素	$O(1)$ 平均, $O(n)$ 最坏
<code>insert(const_iterator hint, const value_type& value)</code>	在 hint 附近插入	$O(1)$ 平均, $O(n)$ 最坏
<code>insert(InputIt first, InputIt last)</code>	插入 [first, last) 的元素	$O(m)$ 平均, $O(m * n)$ 最坏
<code>emplace(Args&&... args)</code>	就地构造元素	$O(1)$ 平均, $O(n)$ 最坏
<code>emplace_hint(const_iterator hint, Args&&... args)</code>	在 hint 附近就地构造	$O(1)$ 平均, $O(n)$ 最坏
<code>erase(const_iterator pos)</code>	删除迭代器位置的元素	$O(1)$ 平均, $O(n)$ 最坏
<code>erase(const key_type& key)</code>	删除键为 key 的元素	$O(1)$ 平均, $O(n)$ 最坏

成员函数	作用	时间复杂度
<code>erase(const_iterator first, const_iterator last)</code>	删除[first, last)的元素	$O(\text{distance(first, last)})$ 平均
<code>swap(unordered_map& other)</code>	交换内容	$O(1)$
<code>count(const key_type& key)</code>	返回键为key的元素数	$O(1)$ 平均, $O(n)$ 最坏
<code>find(const key_type& key)</code>	查找键为key的元素	$O(1)$ 平均, $O(n)$ 最坏
<code>equal_range(const key_type& key)</code>	返回匹配键的范围	$O(1)$ 平均, $O(n)$ 最坏
<code>bucket_count()</code>	返回桶数	$O(1)$
<code>max_bucket_count()</code>	返回最大桶数	$O(1)$
<code>bucket_size(size_type n)</code>	返回第n个桶的元素数	$O(1)$ 平均, $O(\text{bucket_size}(n))$ 最坏
<code>bucket(const key_type& key)</code>	返回键的桶索引	$O(1)$ 平均, $O(n)$ 最坏
<code>load_factor()</code>	返回负载因子	$O(1)$
<code>max_load_factor()</code>	返回最大负载因子	$O(1)$
<code>max_load_factor(float m1)</code>	设置最大负载因子	$O(1)$
<code>rehash(size_type count)</code>	设置桶数至少为count	$O(n)$ 平均, $O(n^2)$ 最坏
<code>reserve(size_type count)</code>	预留空间, 使可容纳count个元素	$O(n)$ 平均, $O(n^2)$ 最坏
<code>hash_function()</code>	返回哈希函数	$O(1)$
<code>key_eq()</code>	返回键相等比较函数	$O(1)$

四、容器适配器

4.1 stack

成员函数	作用	时间复杂度
<code>stack()</code>	默认构造函数	$O(1)$
<code>explicit stack(const Container& cont)</code>	用容器cont构造	$O(n)$
<code>stack(const stack& other)</code>	拷贝构造函数	$O(n)$
<code>stack(stack&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~stack()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>top()</code>	访问栈顶元素	$O(1)$

成员函数	作用	时间复杂度
<code>empty()</code>	检查栈是否为空	O(1)
<code>size()</code>	返回元素数量	O(1)
<code>push(const T& value)</code>	入栈	O(1)
<code>push(T&& value)</code>	移动入栈 (C++11)	O(1)
<code>emplace(Args&&... args)</code>	就地构造入栈 (C++11)	O(1)
<code>pop()</code>	出栈	O(1)
<code>swap(stack& other)</code>	交换内容	O(1)

4.2 queue

成员函数	作用	时间复杂度
<code>queue()</code>	默认构造函数	O(1)
<code>explicit queue(const Container& cont)</code>	用容器cont构造	O(n)
<code>queue(const queue& other)</code>	拷贝构造函数	O(n)
<code>queue(queue&& other)</code>	移动构造函数 (C++11)	O(1)
<code>~queue()</code>	析构函数	O(n)
<code>operator=</code>	赋值运算符	O(n)
<code>front()</code>	访问队首元素	O(1)
<code>back()</code>	访问队尾元素	O(1)
<code>empty()</code>	检查队列是否为空	O(1)
<code>size()</code>	返回元素数量	O(1)
<code>push(const T& value)</code>	入队	O(1)
<code>push(T&& value)</code>	移动入队 (C++11)	O(1)
<code>emplace(Args&&... args)</code>	就地构造入队 (C++11)	O(1)
<code>pop()</code>	出队	O(1)
<code>swap(queue& other)</code>	交换内容	O(1)

4.3 priority_queue

成员函数	作用	时间复杂度
<code>priority_queue()</code>	默认构造函数	O(1)

成员函数	作用	时间复杂度
<code>explicit priority_queue(const Compare& comp)</code>	指定比较器构造	O(1)
<code>priority_queue(InputIt first, InputIt last)</code>	用[first, last)元素构造	O(n)
<code>priority_queue(const priority_queue& other)</code>	拷贝构造函数	O(n)
<code>priority_queue(priority_queue&& other)</code>	移动构造函数 (C++11)	O(1)
<code>~priority_queue()</code>	析构函数	O(n)
<code>operator=</code>	赋值运算符	O(n)
<code>top()</code>	访问堆顶元素	O(1)
<code>empty()</code>	检查是否为空	O(1)
<code>size()</code>	返回元素数量	O(1)
<code>push(const T& value)</code>	插入元素	O(log n)
<code>push(T&& value)</code>	移动插入元素 (C++11)	O(log n)
<code>emplace(Args&&... args)</code>	就地构造元素 (C++11)	O(log n)
<code>pop()</code>	删除堆顶元素	O(log n)
<code>swap(priority_queue& other)</code>	交换内容	O(1)

五、特殊容器

5.1 bitset

成员函数	作用	时间复杂度
<code>bitset()</code>	默认构造函数, 所有位为0	O(N)
<code>bitset(unsigned long long val)</code>	用val初始化	O(N)
<code>bitset(const string& str, size_t pos = 0, size_t n = string::npos)</code>	用字符串初始化	O(N)
<code>~bitset()</code>	析构函数	O(1)
<code>operator=</code>	赋值运算符	O(N)
<code>operator[](size_t pos)</code>	访问特定位	O(1)
<code>test(size_t pos)</code>	测试特定位	O(1)
<code>all()</code>	检查是否所有位都为1 (C++11)	O(N)
<code>any()</code>	检查是否有位为1	O(N)
<code>none()</code>	检查是否没有位为1	O(N)

成员函数	作用	时间复杂度
<code>count()</code>	返回1的个数	$O(N)$
<code>size()</code>	返回位数	$O(1)$
<code>operator== , operator!=</code>	比较两个bitset	$O(N)$
<code>set()</code>	设置所有位为1	$O(N)$
<code>set(size_t pos, bool value = true)</code>	设置特定位	$O(1)$
<code>reset()</code>	重置所有位为0	$O(N)$
<code>reset(size_t pos)</code>	重置特定位	$O(1)$
<code>flip()</code>	翻转所有位	$O(N)$
<code>flip(size_t pos)</code>	翻转特定位	$O(1)$
<code>to_string()</code>	转换为字符串	$O(N)$
<code>to_ulong()</code>	转换为unsigned long	$O(N)$
<code>to_ullong()</code>	转换为unsigned long long (C++11)	$O(N)$
<code>operator& , operator , operator^</code>	位运算	$O(N)$
<code>operator<< , operator>></code>	移位运算	$O(N)$

注: N表示bitset的大小 (位数)

5.2 valarray

成员函数	作用	时间复杂度
<code>valarray()</code>	默认构造函数	$O(1)$
<code>valarray(size_t n)</code>	构造n个元素的valarray	$O(n)$
<code>valarray(const T& val, size_t n)</code>	构造n个val的valarray	$O(n)$
<code>valarray(const T* ptr, size_t n)</code>	从数组构造	$O(n)$
<code>valarray(const valarray& other)</code>	拷贝构造函数	$O(n)$
<code>valarray(valarray&& other)</code>	移动构造函数 (C++11)	$O(1)$
<code>~valarray()</code>	析构函数	$O(n)$
<code>operator=</code>	赋值运算符	$O(n)$
<code>operator[](size_t i)</code>	访问元素	$O(1)$
<code>size()</code>	返回元素数量	$O(1)$

成员函数	作用	时间复杂度
<code>sum()</code>	返回所有元素的和	$O(n)$
<code>min()</code>	返回最小值	$O(n)$
<code>max()</code>	返回最大值	$O(n)$
<code>shift(int n)</code>	移位操作	$O(n)$
<code>cshift(int n)</code>	循环移位	$O(n)$
<code>apply(T func(T))</code>	应用函数到每个元素	$O(n)$
<code>resize(size_t n, T val = T())</code>	改变大小	$O(n)$

六、迭代器相关函数

6.1 所有容器通用迭代器操作

函数	作用	时间复杂度
<code>begin()</code>	返回指向第一个元素的迭代器	$O(1)$
<code>end()</code>	返回尾后迭代器	$O(1)$
<code>cbegin()</code>	返回const迭代器 (C++11)	$O(1)$
<code>cend()</code>	返回const尾后迭代器 (C++11)	$O(1)$
<code>rbegin()</code>	返回逆向迭代器	$O(1)$
<code>rend()</code>	返回逆向尾后迭代器	$O(1)$
<code>crbegin()</code>	返回const逆向迭代器 (C++11)	$O(1)$
<code>crend()</code>	返回const逆向尾后迭代器 (C++11)	$O(1)$

七、C++17新增功能

7.1 提取节点 (node handle)

成员函数	作用	容器支持
<code>extract(const_iterator position)</code>	提取迭代器位置的节点	<code>set, map, unordered_set, unordered_map</code>
<code>extract(const key_type& k)</code>	提取键为k的节点	<code>set, map, unordered_set, unordered_map</code>
<code>insert(node_type&& nh)</code>	插入节点	所有关联容器
<code>merge(container& source)</code>	合并另一个容器	所有关联容器

7.2 try_emplace和insert_or_assign (map/unordered_map)

成员函数	作用	时间复杂度
<code>try_emplace(const key_type& k, Args&&... args)</code>	如果键不存在则插入	$O(\log n)$ 或 $O(1)$ 平均
<code>try_emplace(key_type&& k, Args&&... args)</code>	移动键版本	$O(\log n)$ 或 $O(1)$ 平均
<code>insert_or_assign(const key_type& k, M&& obj)</code>	插入或赋值	$O(\log n)$ 或 $O(1)$ 平均
<code>insert_or_assign(key_type&& k, M&& obj)</code>	移动键版本	$O(\log n)$ 或 $O(1)$ 平均

总结

本表格详细列出了STL主要容器的成员函数及其时间复杂度。需要注意：

1. 时间复杂度表示法：

- $O(1)$: 常数时间
- $O(\log n)$: 对数时间
- $O(n)$: 线性时间
- $O(n \log n)$: 线性对数时间
- "平均": 平均情况复杂度
- "最坏": 最坏情况复杂度
- "均摊": 均摊分析复杂度

2. 影响因素：

- 容器类型（连续vs链表vs树）
- 操作类型（访问vs插入vs删除）
- 数据分布和大小
- 内存分配策略

3. 实际性能：

- 理论复杂度与实际性能可能因实现而异
- 缓存友好性影响实际性能 (`vector > deque > list`)
- 小对象优化 (SSO) 影响string性能

根据具体需求选择合适的容器和操作，是编写高效C++程序的关键。