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

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

1.2 deque

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

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

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

1.3 list

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

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

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

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

1.5 array (C++11)

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

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

1.6 string

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

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

成员函数	作用	时间复杂度
`replace(size_type pos, size_type count, const string& str)`	替换子串	O(n)
`substr(size_type pos = 0, size_type count = npos)`	返回子串	O(count)
`copy(char* dest, size_type count, size_type pos = 0)`	复制到C字符串	O(count)
`resize(size_type count)`	改变大小	O(count - size)
`resize(size_type count, char ch)`	改变大小并用ch填充	O(count - size)
`swap(string& other)`	交换内容	O(1)
`find(const string& str, size_type pos = 0)`	查找子串	O(n*m)（最坏情况）
`rfind(const string& str, size_type pos = npos)`	反向查找	O(n*m)（最坏情况）
`find_first_of(const string& str, size_type pos = 0)`	查找str中任意字符首次出现	O(n*m)（最坏情况）
`find_last_of(const string& str, size_type pos = npos)`	查找str中任意字符最后出现	O(n*m)（最坏情况）
`find_first_not_of(const string& str, size_type pos = 0)`	查找不在str中的字符首次出现	O(n*m)（最坏情况）
`find_last_not_of(const string& str, size_type pos = npos)`	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
`set()`	默认构造函数	O(1)
`set(InputIt first, InputIt last)`	用[first, last)元素构造	O(n log n)
`set(const set& other)`	拷贝构造函数	O(n)
`set(set&& other)`	移动构造函数（C++11）	O(1)
`~set()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)

成员函数	作用	时间复杂度
`begin()`	返回指向第一个元素的迭代器	O(1)
`end()`	返回尾后迭代器	O(1)
`rbegin()`	返回指向第一个元素的逆向迭代器	O(1)
`rend()`	返回逆向尾后迭代器	O(1)
`empty()`	检查容器是否为空	O(1)
`size()`	返回元素数量	O(1)
`max_size()`	返回可容纳的最大元素数	O(1)
`clear()`	清除所有元素	O(n)
`insert(const value_type& value)`	插入元素	O(log n)
`insert(value_type&& value)`	移动插入元素（C++11）	O(log n)
`insert(const_iterator hint, const value_type& value)`	在hint附近插入	O(1)（均摊，如果hint正确）
`insert(InputIt first, InputIt last)`	插入[first, last)的元素	O(m log (n+m))
`emplace(Args&&... args)`	就地构造元素（C++11）	O(log n)
`emplace_hint(const_iterator hint, Args&&... args)`	在hint附近就地构造（C++11）	O(1)（均摊，如果hint正确）
`erase(const_iterator pos)`	删除迭代器位置的元素	O(1)（均摊）
`erase(const key_type& key)`	删除键为key的元素	O(log n)
`erase(const_iterator first, const_iterator last)`	删除[first, last)的元素	O(log n + distance(first, last))
`swap(set& other)`	交换内容	O(1)
`count(const key_type& key)`	返回键为key的元素数	O(log n)
`find(const key_type& key)`	查找键为key的元素	O(log n)
`equal_range(const key_type& key)`	返回匹配键的范围	O(log n)
`lower_bound(const key_type& key)`	返回第一个不小于key的元素	O(log n)
`upper_bound(const key_type& key)`	返回第一个大于key的元素	O(log n)
`key_comp()`	返回键比较函数	O(1)
`value_comp()`	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
`map()`	默认构造函数	O(1)
`map(InputIt first, InputIt last)`	用[first, last)元素构造	O(n log n)
`map(const map& other)`	拷贝构造函数	O(n)
`map(map&& other)`	移动构造函数（C++11）	O(1)
`~map()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`at(const key_type& key)`	访问指定键，带边界检查	O(log n)
`operator[](const key_type& key)`	访问指定键，不存在则插入（仅map）	O(log n)
`operator[](key_type&& key)`	移动键访问（C++11，仅map）	O(log n)
`begin()`	返回指向第一个元素的迭代器	O(1)
`end()`	返回尾后迭代器	O(1)
`rbegin()`	返回指向第一个元素的逆向迭代器	O(1)
`rend()`	返回逆向尾后迭代器	O(1)
`empty()`	检查容器是否为空	O(1)
`size()`	返回元素数量	O(1)
`max_size()`	返回可容纳的最大元素数	O(1)
`clear()`	清除所有元素	O(n)
`insert(const value_type& value)`	插入元素	O(log n)
`insert(value_type&& value)`	移动插入元素（C++11）	O(log n)
`insert(const_iterator hint, const value_type& value)`	在hint附近插入	O(1)（均摊，如果hint正确）
`insert(InputIt first, InputIt last)`	插入[first, last)的元素	O(m log (n+m))
`emplace(Args&&... args)`	就地构造元素（C++11）	O(log n)
`emplace_hint(const_iterator hint, Args&&... args)`	在hint附近就地构造（C++11）	O(1)（均摊，如果hint正确）
`erase(const_iterator pos)`	删除迭代器位置的元素	O(1)（均摊）
`erase(const key_type& key)`	删除键为key的元素	O(log n)
`erase(const_iterator first, const_iterator last)`	删除[first, last)的元素	O(log n + distance(first, last))

成员函数	作用	时间复杂度
`swap(map& other)`	交换内容	O(1)
`count(const key_type& key)`	返回键为key的元素数	O(log n)
`find(const key_type& key)`	查找键为key的元素	O(log n)
`equal_range(const key_type& key)`	返回匹配键的范围	O(log n)
`lower_bound(const key_type& key)`	返回第一个不小于key的元素	O(log n)
`upper_bound(const key_type& key)`	返回第一个大于key的元素	O(log n)
`key_comp()`	返回键比较函数	O(1)
`value_comp()`	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
`unordered_set()`	默认构造函数	O(1)
`unordered_set(size_type bucket_count)`	指定桶数构造	O(bucket_count)
`unordered_set(InputIt first, InputIt last, size_type bucket_count = n)`	用[first, last)元素构造	O(n)平均，O(n²)最坏
`unordered_set(const unordered_set& other)`	拷贝构造函数	O(n)
`unordered_set(unordered_set&& other)`	移动构造函数	O(1)
`~unordered_set()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`begin()`	返回指向第一个元素的迭代器	O(1)
`end()`	返回尾后迭代器	O(1)
`empty()`	检查容器是否为空	O(1)
`size()`	返回元素数量	O(1)
`max_size()`	返回可容纳的最大元素数	O(1)
`clear()`	清除所有元素	O(n)
`insert(const value_type& value)`	插入元素	O(1)平均，O(n)最坏

成员函数	作用	时间复杂度
`insert(value_type&& value)`	移动插入元素	O(1)平均，O(n)最坏
`insert(const_iterator hint, const value_type& value)`	在hint附近插入	O(1)平均，O(n)最坏
`insert(InputIt first, InputIt last)`	插入[first, last)的元素	O(m)平均，O(m*n)最坏
`emplace(Args&&... args)`	就地构造元素	O(1)平均，O(n)最坏
`emplace_hint(const_iterator hint, Args&&... args)`	在hint附近就地构造	O(1)平均，O(n)最坏
`erase(const_iterator pos)`	删除迭代器位置的元素	O(1)平均，O(n)最坏
`erase(const key_type& key)`	删除键为key的元素	O(1)平均，O(n)最坏
`erase(const_iterator first, const_iterator last)`	删除[first, last)的元素	O(distance(first, last))平均
`swap(unordered_set& other)`	交换内容	O(1)
`count(const key_type& key)`	返回键为key的元素数	O(1)平均，O(n)最坏
`find(const key_type& key)`	查找键为key的元素	O(1)平均，O(n)最坏
`equal_range(const key_type& key)`	返回匹配键的范围	O(1)平均，O(n)最坏
`bucket_count()`	返回桶数	O(1)
`max_bucket_count()`	返回最大桶数	O(1)
`bucket_size(size_type n)`	返回第n个桶的元素数	O(1)平均，O(bucket_size(n))最坏
`bucket(const key_type& key)`	返回键的桶索引	O(1)平均，O(n)最坏
`load_factor()`	返回负载因子	O(1)
`max_load_factor()`	返回最大负载因子	O(1)
`max_load_factor(float ml)`	设置最大负载因子	O(1)
`rehash(size_type count)`	设置桶数至少为count	O(n)平均，O(n²)最坏
`reserve(size_type count)`	预留空间，使可容纳count个元素	O(n)平均，O(n²)最坏
`hash_function()`	返回哈希函数	O(1)
`key_eq()`	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
`unordered_map()`	默认构造函数	O(1)

成员函数	作用	时间复杂度
`unordered_map(size_type bucket_count)`	指定桶数构造	O(bucket_count)
`unordered_map(InputIt first, InputIt last, size_type bucket_count = n)`	用[first, last)元素构造	O(n)平均，O(n²)最坏
`unordered_map(const unordered_map& other)`	拷贝构造函数	O(n)
`unordered_map(unordered_map&& other)`	移动构造函数	O(1)
`~unordered_map()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`at(const key_type& key)`	访问指定键，带边界检查	O(1)平均，O(n)最坏
`operator[](const key_type& key)`	访问指定键，不存在则插入（仅unordered_map）	O(1)平均，O(n)最坏
`operator[](key_type&& key)`	移动键访问（仅unordered_map）	O(1)平均，O(n)最坏
`begin()`	返回指向第一个元素的迭代器	O(1)
`end()`	返回尾后迭代器	O(1)
`empty()`	检查容器是否为空	O(1)
`size()`	返回元素数量	O(1)
`max_size()`	返回可容纳的最大元素数	O(1)
`clear()`	清除所有元素	O(n)
`insert(const value_type& value)`	插入元素	O(1)平均，O(n)最坏
`insert(value_type&& value)`	移动插入元素	O(1)平均，O(n)最坏
`insert(const_iterator hint, const value_type& value)`	在hint附近插入	O(1)平均，O(n)最坏
`insert(InputIt first, InputIt last)`	插入[first, last)的元素	O(m)平均，O(m*n)最坏
`emplace(Args&&... args)`	就地构造元素	O(1)平均，O(n)最坏
`emplace_hint(const_iterator hint, Args&&... args)`	在hint附近就地构造	O(1)平均，O(n)最坏
`erase(const_iterator pos)`	删除迭代器位置的元素	O(1)平均，O(n)最坏
`erase(const key_type& key)`	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
`erase(const_iterator first, const_iterator last)`	删除[first, last)的元素	O(distance(first, last))平均
`swap(unordered_map& other)`	交换内容	O(1)
`count(const key_type& key)`	返回键为key的元素数	O(1)平均，O(n)最坏
`find(const key_type& key)`	查找键为key的元素	O(1)平均，O(n)最坏
`equal_range(const key_type& key)`	返回匹配键的范围	O(1)平均，O(n)最坏
`bucket_count()`	返回桶数	O(1)
`max_bucket_count()`	返回最大桶数	O(1)
`bucket_size(size_type n)`	返回第n个桶的元素数	O(1)平均，O(bucket_size(n))最坏
`bucket(const key_type& key)`	返回键的桶索引	O(1)平均，O(n)最坏
`load_factor()`	返回负载因子	O(1)
`max_load_factor()`	返回最大负载因子	O(1)
`max_load_factor(float ml)`	设置最大负载因子	O(1)
`rehash(size_type count)`	设置桶数至少为count	O(n)平均，O(n²)最坏
`reserve(size_type count)`	预留空间，使可容纳count个元素	O(n)平均，O(n²)最坏
`hash_function()`	返回哈希函数	O(1)
`key_eq()`	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
`stack()`	默认构造函数	O(1)
`explicit stack(const Container& cont)`	用容器cont构造	O(n)
`stack(const stack& other)`	拷贝构造函数	O(n)
`stack(stack&& other)`	移动构造函数（C++11）	O(1)
`~stack()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`top()`	访问栈顶元素	O(1)

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

4.2 queue

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

4.3 priority_queue

成员函数	作用	时间复杂度
`priority_queue()`	默认构造函数	O(1)

成员函数	作用	时间复杂度
`explicit priority_queue(const Compare& comp)`	指定比较器构造	O(1)
`priority_queue(InputIt first, InputIt last)`	用[first, last)元素构造	O(n)
`priority_queue(const priority_queue& other)`	拷贝构造函数	O(n)
`priority_queue(priority_queue&& other)`	移动构造函数（C++11）	O(1)
`~priority_queue()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`top()`	访问堆顶元素	O(1)
`empty()`	检查是否为空	O(1)
`size()`	返回元素数量	O(1)
`push(const T& value)`	插入元素	O(log n)
`push(T&& value)`	移动插入元素（C++11）	O(log n)
`emplace(Args&&... args)`	就地构造元素（C++11）	O(log n)
`pop()`	删除堆顶元素	O(log n)
`swap(priority_queue& other)`	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
`bitset()`	默认构造函数，所有位为0	O(N)
`bitset(unsigned long long val)`	用val初始化	O(N)
`bitset(const string& str, size_t pos = 0, size_t n = string::npos)`	用字符串初始化	O(N)
`~bitset()`	析构函数	O(1)
`operator=`	赋值运算符	O(N)
`operator[](size_t pos)`	访问特定位	O(1)
`test(size_t pos)`	测试特定位	O(1)
`all()`	检查是否所有位都为1（C++11）	O(N)
`any()`	检查是否有位为1	O(N)
`none()`	检查是否没有位为1	O(N)

成员函数	作用	时间复杂度
`count()`	返回1的个数	O(N)
`size()`	返回位数	O(1)
`operator==`，`operator!=`	比较两个bitset	O(N)
`set()`	设置所有位为1	O(N)
`set(size_t pos, bool value = true)`	设置特定位	O(1)
`reset()`	重置所有位为0	O(N)
`reset(size_t pos)`	重置特定位	O(1)
`flip()`	翻转所有位	O(N)
`flip(size_t pos)`	翻转特定位	O(1)
`to_string()`	转换为字符串	O(N)
`to_ulong()`	转换为unsigned long	O(N)
`to_ullong()`	转换为unsigned long long（C++11）	O(N)
`operator&`，`operator |`，`operator^`	位运算	O(N)
`operator<<`，`operator>>`	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
`valarray()`	默认构造函数	O(1)
`valarray(size_t n)`	构造n个元素的valarray	O(n)
`valarray(const T& val, size_t n)`	构造n个val的valarray	O(n)
`valarray(const T* ptr, size_t n)`	从数组构造	O(n)
`valarray(const valarray& other)`	拷贝构造函数	O(n)
`valarray(valarray&& other)`	移动构造函数（C++11）	O(1)
`~valarray()`	析构函数	O(n)
`operator=`	赋值运算符	O(n)
`operator[](size_t i)`	访问元素	O(1)
`size()`	返回元素数量	O(1)

成员函数	作用	时间复杂度
`sum()`	返回所有元素的和	O(n)
`min()`	返回最小值	O(n)
`max()`	返回最大值	O(n)
`shift(int n)`	移位操作	O(n)
`cshift(int n)`	循环移位	O(n)
`apply(T func(T))`	应用函数到每个元素	O(n)
`resize(size_t n, T val = T())`	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
`begin()`	返回指向第一个元素的迭代器	O(1)
`end()`	返回尾后迭代器	O(1)
`cbegin()`	返回const迭代器（C++11）	O(1)
`cend()`	返回const尾后迭代器（C++11）	O(1)
`rbegin()`	返回逆向迭代器	O(1)
`rend()`	返回逆向尾后迭代器	O(1)
`crbegin()`	返回const逆向迭代器（C++11）	O(1)
`crend()`	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
`extract(const_iterator position)`	提取迭代器位置的节点	set, map, unordered_set, unordered_map
`extract(const key_type& k)`	提取键为k的节点	set, map, unordered_set, unordered_map
`insert(node_type&& nh)`	插入节点	所有关联容器
`merge(container& source)`	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
`try_emplace(const key_type& k, Args&&... args)`	如果键不存在则插入	O(log n) 或 O(1)平均
`try_emplace(key_type&& k, Args&&... args)`	移动键版本	O(log n) 或 O(1)平均
`insert_or_assign(const key_type& k, M&& obj)`	插入或赋值	O(log n) 或 O(1)平均
`insert_or_assign(key_type&& k, M&& obj)`	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。