Bài 8.10: Tìm hiểu lớp unordered_set

- ✓ Tổng quan
- ✓ Các phương thức và mô tả
- ✓ Ví dụ minh họa

Tổng quan

- Lớp unordered_set là một container lưu trữ các phần tử duy nhất và không sắp đặt chúng theo bất kì trật tự nào. Do đó cho phép nhanh chóng truy cập vào các phần tử cụ thể dựa vào giá trị của chúng.
- Trong unordered_set, giá trị của mỗi phần tử cũng đồng thời là key để phân biệt sự duy nhất của phần tử đó với các phần tử khác.
- Các key trong unordered_set là bất biến, do đó ta không thể sửa đổi giá trị của phần tử trong container này. Ta chỉ có thể thêm mới hoặc tìm kiếm, xóa bỏ phần tử khỏi container.
- Các phần tử trong container này không được sắp đặt theo bất kì trật tự nào. Tuy nhiên chúng được lưu tại các vị trí ô nhớ theo giá trị băm của phần tử đó. Mục đích để truy cập nhanh tới các phần tử theo giá trị.

Tổng quan

- unordered_set nhanh hơn set về mặt tốc độ truy cập phần tử đơn.
- Tính chất của container: tính liên kết, tính không trật tự, tính tập hợp, tính duy nhất và tính tự động quản lý bộ nhớ.
- Để sử dụng ta include thư viện <unordered_set> vào đầu file chương trình.

Các hàm thông dụng

unorderd_set(); ***explicit unorderd_set(size_type bucket_count, const Hash& hash = Hash(), const key_equals& equal = key_equal(), const Allocator& alloc = Allocator());*** ***explicit unorderd_set(size_type bucket_count, const Allocator& alloc); (Từ C++14)*** ***explicit unorderd_set(size_type bucket_count, const Hash& hash, const Allocator& alloc); (Từ C++14)*** ***explicit unorderd_set(const Allocator& alloc); (Từ C++11)*** Hàm tạo mặc định. Dùng để tạo một container rỗng. Gán giá trị max_load_factor() bằng 1.0.
unordered_set(InputIt first, InputIt last, size_type bucket_count = default, const Hash& hash = Hash(), const key_equal& equal = key_equal(), const Allocator& alloc = Allocator()); (Từ C++11) ***unordered_set(InputIt first, InputIt last, size_type bucket_count, const Allocator& alloc); (Từ C++14)*** ***unordered_set(InputIt first, InputIt last, size_type bucket_count, const Hash& hash, const Allocator& alloc); (Từ C++14)*** Hàm tạo container với nội dung cho trong một khoảng [first, last). Gán giá trị max_load_factor() bằng 1.0. Nếu trong khoảng đã cho có nhiều phần tử có giá trị tương đương, nó sẽ không xác định cụ thể phần tử nào được chọn để chèn vào set.
unordered_set(const unordered_set& other); ***unordered_set(const unordered_set& other, const Allocator& alloc);*** Hàm tạo copy, tạo container với các bản sao của nội dung cho trong other. Nhân bản cả load factor, predicate và hàm băm.
unordered_set(initializer_list<value_type> init, size_type bucket_count = default, const Hash& hash = Hash(), const key_equal& equal = key_equal(), const Allocator& alloc = Allocator()); (Từ C++11) ***unordered_set(initializer_list<value_type> init, size_type bucket_count, const Allocator& alloc); (Từ C++11)***

Các hàm thông dụng

unordered_set(initializer_list<value_type> init, size_type bucket_count, const Hash& hash, const key_equal& equal, const Allocator& alloc); (Từ C++11) Tạo một container mới với nội dung cho trước trong danh sách khởi tạo init.
iterator begin() noexcept; (Từ C++11) ***const_iterator begin() const noexcept; (Từ C++11)*** ***const_iterator cbegin() const noexcept; (Từ C++11)*** Trả về iterator trỏ tới phần tử đầu tiên trong unordered_set. Nếu unordered_set rỗng thì kết quả trả về tương đương end().
iterator end() noexcept; (Từ C++11) ***const_iterator end() const noexcept; (Từ C++11)*** ***const_iterator cend() const noexcept; (Từ C++11)*** Trả về một iterator trỏ tới vị trí sau phần tử cuối của unordered_set. Phần tử này chỉ sử dụng để làm giá trị đối sánh. Việc cố truy cập tới giá trị của nó sẽ gây ra hành vi không xác định.
bool empty() const noexcept; (Từ C++11-C++20) ***[[nodiscard]] bool empty() const noexcept; (Từ C++20).*** Hàm kiểm tra xem container có rỗng không. Nếu container rỗng thì begin() == end().
size_type size() const noexcept; (Từ C++11) Trả về số phần tử hiện có trong container.
size_type max_size() const noexcept; (Từ C++11) Trả về số lượng phần tử tối đa mà container có thể chứa tùy theo hệ điều hành và thư viện đang sử dụng. Đây là giá trị theo lý thuyết có thể đạt được tuy nhiên nó phụ thuộc vào trạng thái RAM khả dụng của thiết bị.
void clear() noexcept; (Tới C++11) Xóa tất cả các phần tử có trong container. Sau khi gọi hàm này, size của container sẽ về 0.

Các hàm thông dụng

std::pair<iterator, bool> insert(const value_type& value); (Từ C++11) ***std::pair<iterator, bool> insert(value_type& value); (Từ C++11)*** ***iterator insert(const_iterator hint, const value_type& value); (Từ C++11)*** ***iterator insert(const_iterator hint, value_type&& value); (Từ C++11)*** ***void insert(InputIt first, InputIt last);*** ***void insert(initializer_list<value_type> iList); (Từ C++11)*** ***iterator insert(const_iterator hint, node_type&& nh); (Từ C++17)*** Hàm này dùng để chèn thêm 1 hoặc một số phần tử vào container nếu set chưa tồn tại phần tử nào tương đương với key truyền vào. Hint là một iterator dùng để gợi ý vị trí bắt đầu việc tìm kiếm chỗ để chèn (các) phần tử mới vào.
std::pair<iterator, bool> emplace(Args&&... args); (Từ C++11) Chèn thêm phần tử mới vào set nếu phần tử đó chưa tồn tại trong set. Việc sử dụng hàm này sẽ trực tiếp tạo đối tượng mới, tránh được việc sao chép hoặc di chuyển không cần thiết.
iterator erase(const_iterator pos); (Từ C++11) ***iterator erase(iterator pos); (Từ C++17)*** ***iterator erase(const_iterator first, const_iterator last); (Từ C++11)*** ***size_type erase(const key_type& key);*** Xóa một phần tử cụ thể khỏi container. Có thể xóa phần tử tại vị trí pos, trong khoảng [first, last) hoặc xóa phần tử có giá trị tương đương với key truyền vào.
void swap(unordered_set& other); (Từ C++11 tới C++17) ***void swap(unordered_set& other) noexcept; (Từ C++17)*** Tráo đổi nội dung của container hiện thời với container other.
node_type extract(const_iterator position); (Từ C++17).(1)

Các hàm thông dụng

node_type extract(const key_type& x); (Từ C++17).(2) (1) Hủy liên kết node chứa phần tử trỏ tới bởi position và trả về node chứa phần tử đó. (2) Nếu container chứa một phần tử trùng với x, hủy liên kết node chứa phần tử đó khỏi container và trả về một node chứa nó. Ngược lại, trả về một node rỗng.
void merge(unordered_set<Key, H2, P2, Allocator>& source); (Từ C++17) ***void merge(unordered_set<Key, H2, P2, Allocator>&& source); (Từ C++17)*** ***void merge(unordered_multiset<Key, H2, P2, Allocator>& source); (Từ C++17)*** ***void merge(unordered_multiset<Key, H2, P2, Allocator>&& source); (Từ C++17)*** Thử trích xuất từng phần tử trong source và chèn nó vào set hiện thời sử dụng đối tượng so sánh của *this. Nếu có một phần tử trong *this có key tương đương với key của phần tử trong source, phần tử đó sẽ không được trích xuất từ source. Chỉ xảy ra việc thay đổi con trỏ bên trong các set tham gia vào hành động này.
size_type count(const Key& key) const; (Từ C++11) ***template<class K> size_type count(const K& x) const; (Từ C++14)*** Trả về số lượng phần tử có key tương đương với tham số đầu vào. Tức là có thể là 1 hoặc 0 do set không cho phép các phần tử trùng lặp.
iterator find(const Key& key); ***const_iterator find(const Key& key) const;*** **iterator find(const K& x); (Từ C++20)** **const_iterator find(const K& x) const; (Từ C++20)** Tìm phần tử có key tương đương với key trong tham số hoặc tương đương với giá trị x.
bool contains(const Key& key) const; (Từ C++20) ***template<class K> bool contains(const K& x) const; (Từ C++20)*** Kiểm tra xem có tồn tại phần tử có key tương đương với tham số trong container không.

Các hàm thông dụng

local_iterator begin(size_type n); (Từ C++11) ***const_local_iterator begin(size_type n) const; (Từ C++11)*** ***const_local_iterator cbegin(size_type n) const; (Từ C++11)*** Trả về một iterator trỏ tới phần tử đầu tiên trong ô nhớ có chỉ số n.
local_iterator end(size_type n); (Từ C++11) ***const_local_iterator end(size_type n) const; (Từ C++11)*** ***const_local_iterator cend(size_type n) const; (Từ C++11)*** Trả về một iterator trỏ tới phần tử sau phần tử cuối của nô nhớ có chỉ số n. Phần tử này hoạt động như một giá trị đối sánh không có giá trị khi truy cập vào nó.
size_type bucket_count() const; (Từ C++11) Trả về số lượng các ô nhớ(tính theo đơn vị phần tử) trong container.
size_type bucket_count() const; (Từ C++11) Trả về số ô nhớ tối đa trong container có thể có. Giá trị này tùy thuộc hệ điều hành và thư viện đang sử dụng.
size_type bucket_size(size_type n) const; (Từ C++11) Trả về số lượng phần tử trong ô nhớ với chỉ số n.
size_type bucket(const Key& key) const; (Từ C++11) Trả về chỉ số của ô nhớ có chứa giá trị key.
float load_factor() const; (Từ C++11) Trả về số phần tử trung bình trên mỗi ô nhớ. Tức giá trị size()/bucket_count().
float max_load_factor() const; (Từ C++11) ***void max_load_factor(float ml); (Từ C++11)***

Các hàm thông dụng

Quản lý giá trị load factor tối đa. Container tự động tăng số lượng các ô nhớ nếu load factor đạt ngưỡng của giá trị này.
***void* rehash(size_type *count*); (Từ C++11)** Gán số lượng ô nhớ thành count và băm lại container.
***void* reserve(size_type count); (Từ C++11)** Thiết lập giá trị số ô nhớ trong container về số ô nhớ cần thiết để lưu tối thiểu count phần tử mà không vượt ngưỡng load factor tối đa và băm lại bảng băm.
***hasher hash_function() const*; (Từ C++11)** Trả về hàm băm đang sử dụng trong container.

Nội dung tiếp theo

Tìm hiểu lớp unordered_map