STL: ассоциативные контейнеры

Александр Смаль

CS центр
20 марта 2018
Санкт-Петербург

Общие сведения

Ассоциативные контейнеры делятся на две группы:

- *упорядоченные* (требуют отношение порядка),
- *неупорядоченные* (требуют хеш-функцию).

Общие методы

1. `find` по ключу,
2. `count` по ключу,
3. `erase` по ключу.

Шаблоны `set` и `multiset`

`set` хранит упорядоченное множество (дерево поиска). Операции добавления, удаления и поиска работают за $O(\log n)$. Значения, которые хранятся в `set`, неизменяемые.

- `lower_bound, upper_bound, equal_range`.

```
std::set<int> primes = {2, 3, 5, 7, 11};
// дальнейшее заполнение
if (primes.find(173) != primes.end())
    std::cout << 173 << " is prime\n";

// std::pair<iterator, bool>
auto res = primes.insert(3);
```

В `multiset` хранится упорядоченное мультимножество.

```
std::multiset<int> fib = {0, 1, 1, 2, 3, 5, 8};
// iterator
auto res = fib.insert(13);
// pair<iterator, iterator>
auto eq = fib.equal_range(1);
```

Шаблоны map и multimap

Упорядоченное отображение (дерево поиска по ключу).

Операции добавления, удаления и поиска работают за $O(\log n)$.

```
    typedef std::pair<const Key, T> value_type;
```

- lower_bound, upper_bound, equal_range,
- operator[], at.

```
std::map<std::string, int> phonebook;
phonebook.emplace("Marge", 2128506);
phonebook.emplace("Lisa",  2128507);
phonebook.emplace("Bart",  2128507);
// std::map<string,int>::iterator
auto it = phonebook.find("Maggie");
if ( it != phonebook.end())
    std::cout << "Maggie: " << it->second << "\n";
```

```
std::multimap<std::string, int> phonebook;
phonebook.emplace("Homer", 2128506);
phonebook.emplace("Homer", 5552368);
```

Особые методы `map`: `operator[]` и `at`

```
auto it = phonebook.find("Marge");
if (it != phonebook.end())
     it->second = 5550123;
else
    phonebook.emplace("Marge", 5550123);
// или
phonebook["Marge"] = 5550123;
```

Метод `operator[]`:

1. работает только с неконстантным `map`,
2. требует наличие у `T` конструктора по умолчанию,
3. работает за $O(\log n)$ (не стоит использовать `map` как массив).

Метод `at`:

1. генерирует ошибку времени выполнения, если такой ключ отсутствует,
2. работает за $O(\log n)$.

Использование собственного компаратора

Отношение строгого порядка: $\neg(x \prec y) \wedge \neg(y \prec x) \Rightarrow x = y$

```
struct Person { string name; string surname; };

bool operator<(Person const& a, Person const& b) {
    return a.name < b.name ||
           (a.name == b.name && a.surname < b.surname);
}
// уникальны по сочетанию имя + фамилия
std::set<Person> s1;

struct PersonComp {
    bool operator()(Person const& a, Person const& b) const {
        return a.surname < b.surname;
    }
};
// уникальны по фамилии
std::set<Person, PersonComp> s2;
```

Шаблоны `unordered_set` и `unordered_multiset`

`unordered_set` хранит множество как хеш-таблицу.

Операции добавления, удаления и поиска за $O(1)$ в среднем.

Значения в `unordered_set` — неизменяемые.

- `equal_range`, `reserve`,
- методы для работы с хеш-таблицей.

```
unordered_set<int> primes = {2, 3, 5, 7, 11};
// дальнейшее заполнение
if (primes.find(173) != primes.end())
    std::cout << 173 << " is prime\n";

// std::pair<iterator, bool>
auto res = primes.insert(3);
```

В `unordered_multiset` хранится мультимножество.

```
unordered_multiset<int> fib = {0, 1, 1, 2, 3, 5, 8};
// iterator
auto res = fib.insert(13);
```

Шаблоны unordered_map и unordered_multimap

Хранит отображение как хеш-таблицу.

Операции добавления, удаления и поиска за $O(1)$ в среднем.

- equal_range, reserve, operator[], at,
- методы для работы с хеш-таблицей.

```
#include <unordered_map>
```

```
unordered_map<std::string, int> phonebook;
phonebook.emplace("Marge", 2128506);
phonebook.emplace("Lisa",  2128507);
phonebook.emplace("Bart",  2128507);
// unordered_map<string,int>::iterator
auto it = phonebook.find("Maggie");
if ( it != phonebook.end())
    std::cout << "Maggie: " << it->second << "\n";
```

```
unordered_multimap<std::string, int> phonebook;
phonebook.emplace("Homer", 2128506);
phonebook.emplace("Homer", 5552368);
```

Использование собственной хеш-функции

```
struct Person { string name; string surname; };

bool operator==(Person const& a, Person const& b) {
    return a.name    == b.name
        && a.surname == b.surname;
}

namespace std {
    template <> struct hash<Person> {
        size_t operator()(Person const& p) const {
            hash<string> h;
            return h(p.name) ^ h(p.surname);
        }
    };
}

// уникальны по сочетанию имя + фамилия
unordered_set<Person> s;
```