OTUS

Онлайн образование

otus.ru

Проверить, идет ли запись

Меня хорошо видно && слышно?

Тема вебинара

Алгоритмы, которые лучше знать лучше

Андрей Рыжиков

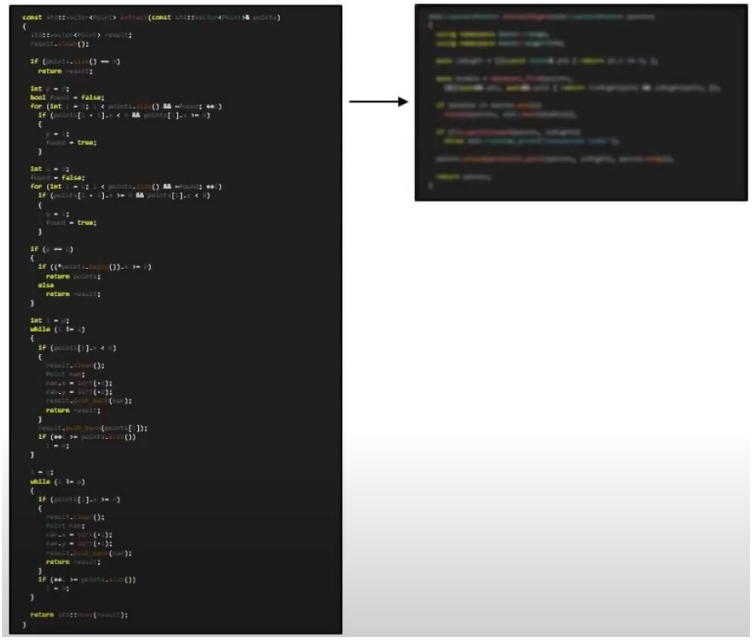
@ryzhikovas

Используйте готовые алгоритмы STL

```
#include <algorithm>
```

Можно написать самому. Но

- Возможны ошибки
- Падает читаемость кода
- У вас есть тесты?
- У вас есть документация?
  - И, разумеется, описаны гарантии по быстродействию
- Параллельная реализация?



[Михаил Матросов. Алгоритмы и итераторы]

Отступление. Лямбда-функции

Lambda - безымянный функциональный объект.

```
[captures](params)specs requires  {
    //body
    return …;
}
```

Опциональны. Можем явно указать тип возвращаемого значения: `-> int mutable, noexcept…`

// корректная программа на языке C++:

```
int main() {[](){}();}
```

// Если не хватило скобочек: C++20:

```
int main() {[]<typename T=int>(){}();}
```

Минимум/максимум

(1) const T&	**min/max**	(1) (const T& a, const T& b, Cmp cmp)
(2) T		(2) (std::initializer_list<T> l,Cmp cmp)
(1) const T&	**min/max**	
(2) T		
(1) pair<const T&,const T&>		
(2) pair<T,T>	**minmax**	

FwdIt	**min/max_element**	(FwdIt begin, FwdIt end, Cmp cmp)
pair<FwdIt, FwdIt>	**minmax_element**	

“Обрезает” значение, приводя в диапазон [`lo`, `hi`]:

const T&	**clamp**	(const T& v, const T& lo, const T& hi, Cmp cmp)

Константные алгоритмы

```
// true, если p вернул true для всех элементов
bool all_of(It first, It last, UnaryPredicate p)

// true, если p вернул true для кого-то
bool any_of(It first, It last, UnaryPredicate p)

// true, если p вернул false для всех элементов
bool none_of(It first, It last, UnaryPredicate p)
```

Предикаты не должны менять свое состояние (!)

Константные алгоритмы

```
// Цикл по диапазону
void for_each(It first, It last, UnaryFn f)
void for_each_n(It first, Size n, UnaryFn f)

// Подсчет числа вхождений
diff count(It first, It last, const T& value)
diff count_if(It first, It last, UnaryPredicate p)

// Поиск одного элемента
It find(It first, It last, const T& value)
It find_if(It first, It last, UnaryPredicate p)
It find_if_not(It first, It last, UnaryPredicate p)
```

Константные алгоритмы

// Поиск подряд идущих `count` элементов `value`.

```
It search_n(It first, It last, Size count,
            const T& value, BinaryPred p)
```

// Поиск поддиапазона. **O(m·n)**

```
It1 search(It1 first, It1 last, It2 a, It2 b, BinaryPred p)
```

// Поиск поддиапазона. **O(?)**

```
It search(It first, It last, const Searcher& searcher)
```

Константные алгоритмы

```
// Диапазоны эквивалентны?
bool equal(It1 first1, It1 last1, It2 first2, BinaryPred p)

// Первый различающийся элемент
pair<It1,It2> mismatch(It1 first, It1 last, It2 first2,
                       BinaryPred p)
```

Модифицирующие алгоритмы

// Задаем диапазон значений:

- `fill`
- `fill_n`
- `generate`
- `generate_n`

// Копируем/перемещаем диапазон:

- `copy`
- `copy_n`
- `copy_if`
- `move`
- `copy_backward`
- `move_backward`

Модифицирующие алгоритмы

// Трансформируем:

```
OutIt transform(InIt first, InIt last, OutIt out, UnaryOp op)

OutIt transform(InIt1 first, InIt1 last, InIt2 other,
                OutIt dest, BinaryOp op)
```

// Удаляем:

```
It remove(It first, It last, const T& value)
It remove_if(It first, It last, UnaryPredicate p)
```

// А также делаем более сложные вещи::

- **remove_copy**
- **remove_copy_if**
- **sample**
- **shuffle**
- **reverse**
- **rotate**
- **unique**

Упорядочивающие алгоритмы

// Сортируем:

```
void sort(RandomIt first, RandomIt last, Cmp cmp)
```

Q: Какая сложность у `std::sort`?

A: O(N·logN).

```
void stable_sort(RandomIt first, RandomIt last, Cmp cmp)
```

O(N·logN) - если хватило памяти. Иначе O(N·log²N)

// Разделяем:

```
It partition(It first, It last, UnaryPredicate p)
```

Q: Сложность?

A: O(N).

// ~~Властвуем~~ Кто в отсортированной последовательности будет на месте n?

```
void nth_element(RandomIt first, RandomIt nth,
                 RandomIt last, Cmp cmp)
```

5 3 7 2 1

O(N)+

Упорядочивающие алгоритмы

// Сортируем, но не все:

```
void partial_sort(RandomIt first, RandomIt middle, RandomIt last,
                  Cmp cmp)
```

```
//std::vector<int> v = {5, 3, 7, 2, 1};
auto getBestN(std::vector<int> v, size_t n) {
    std::partial_sort(v.begin(), v.begin() + n +1, v.end());
     v.resize(n);
      return v;
}
```

O(N·log**K**).

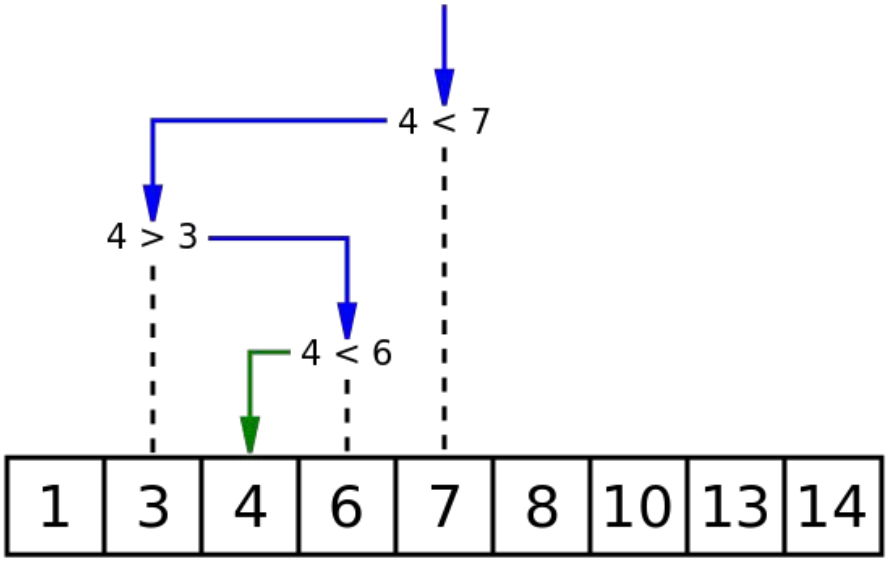
5 3 7 2 1

Бинарный поиск

Работает только на отсортированной последовательности

`It`	**`lower_bound`**	
`It`	**`upper_bound`**	`(It first, It last,`
`bool`	**`binary_search`**	`const T& value, Cmp cmp)`
`pair<It, It>`	**`equal_range`**	

O(logN).



Численные алгоритмы

// Заполняем последовательными элементами:

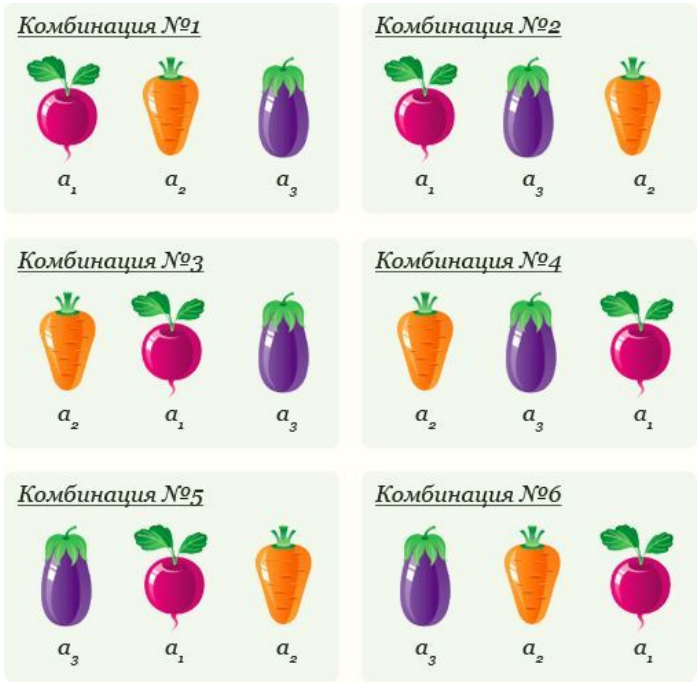
```
void iota(It first, It last, T init)
```

// Сумма:

```
T accumulate(It first, It last, T init, BinaryOp op)
```

Перестановки

```
bool next_permutation(RandomIt first, RandomIt last)   // O(1)+
bool prev_permutation(RandomIt first, RandomIt last)   // O(1)+
```




```
#include <algorithm>
#include <string>
#include <iostream>

int main()
{
    std::string s = "aba";
    std::sort(s.begin(), s.end());
    do {
        std::cout << s << '\n';
    } while(std::next_permutation(s.begin(), s.end()));
}
```

Бонус

```
#include <vector>
#include <algorithm>

enum class Kind { Beautiful, Intelligent};

struct Person {
    Kind kind;
};


// Переупорядочивает элементы так, что сначала идут умные,
потом - красивые
void reorderIntelligentFirst(std::vector<Person>& persons);
```

Бонус

```
bool cmp(Person lhs, Person rhs) {
  return lhs.kind > rhs.kind;
};
void reorderIntelligentFirst(std::vector<Person>& persons) {
  std::ranges::sort(persons, cmp); //N·logN
}

void reorderIntelligentFirst(std::vector<Person>& persons) {
  size_t count = std::ranges::count_if(persons, [] (auto& p) {
    return p.kind == Kind::Intelligent;
  });
  std::ranges::partial_sort(persons,
      persons.begin() + count, cmp); //N·logN
 }
```

Бонус

```
void reorderIntelligentFirst(std::vector<Person>& persons) {
  size_t count = std::ranges::count_if(persons, [] (auto& p) {
    return p.kind == Kind::Intelligent;
  });
  std::ranges::nth_element(persons, persons.begin() + count,
      cmp); // ~N
}

void reorderIntelligentFirst(std::vector<Person>& persons) {
  std::ranges::partition(persons, [] (auto p) {   // N
    return p.kind == Kind::Intelligent;
  });
}
```

Заключение

Заполните, пожалуйста,
опрос о занятии
по ссылке в чате

Вопросы?