Алгоритмы STL

- немодифицирующие
- модифицирующие
- алгоритмы удаления
- перестановочные
- сортировки
- алгоритмы для упорядоченных диапазонов
- численные алгоритмы

Немодифицирующие алгоритмы
Подсчет элементов

int **count**(InputIterator beg, InputIterator end, const &T value)
Подсчет элементов в диапазоне [begin, end), равных value
int **count_if**(InputIterator beg, InputIterator end, UnaryPredicate op)
Подсчет элементов в диапазоне [begin, end), для которых унарный предикат op возвращает true
Сложность: линейная.

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

using namespace std;

bool isEven (int elem)
{
    return elem % 2 == 0;
}

int main()
{
    vector<int> coll = {1, 2, 3, 4, 5, 6, 7, 8, 9};
    int num;
    // подсчитать и вывести на экран количество элементов, равных 4
    num = count(coll.begin(), coll.end(), 4);
    cout << num << endl;
    // подсчитать количество четных элементов
    num = count_if(coll.begin(), coll.end(), isEven);
}
```

Немодифицирующие алгоритмы
Минимум и максимум

ForwardIterator min_element(ForwardIterator beg, ForwardIterator end)
Нахождение минимального элемента, сравнение с помощью операции <

ForwardIterator min_element(ForwardIterator beg, ForwardIterator end, CompFunc op)
Нахождение минимального элемента, сравнение с помощью бинарного предиката op

ForwardIterator max_element(ForwardIterator beg, ForwardIterator end)
Нахождение максимального элемента, сравнение с помощью операции <

ForwardIterator max_element(ForwardIterator beg, ForwardIterator end, CompFunc op)
Нахождение максимального элемента, сравнение с помощью бинарного предиката op

pair<ForwardIterator, ForwardIterator> minmax_element(ForwardIterator beg, ForwardIterator end)
pair из максимального и минимального элементов

pair<ForwardIterator, ForwardIterator> minmax_element(ForwardIterator beg, ForwardIterator end, CompFunc op)

Сложность: линейная.

Немодифицирующие алгоритмы
Минимум и максимум

```cpp
#include <iostream>
#include <deque>
#include <algorithm>

using namespace std;

bool absLess (int elem1, int elem2)
{
    return abs(elem1) < abs(elem2);
}

int main()
{
    deque<int> coll = {6, 1, 2, -9, 0, -5, 7};
    cout << "minimum: " << *min_element(coll.begin(),coll.end()) << endl;
    cout << "maximum: " << *max_element(coll.begin(),coll.end()) << endl;
    cout << "minimum of absolute values: "
         << *min_element(coll.begin(),coll.end(), absLess) << endl;
    cout << "maximum of absolute values: "
         << *max_element(coll.begin(),coll.end(), absLess) << endl;
}
```

Немодифицирующие алгоритмы
Поиск элементов
Поиск первого совпадения

InputIterator find(InputIterator beg, InputIterator end, const &T value)
Возвращает позицию первого элемента в диапазоне [beg, end), значение которого равно value

InputIterator find_if(InputIterator beg, InputIterator end, UnaryPredicate op)
Возвращает позицию первого элемента в диапапзоне [beg, end), для которого унарный предикат op возвращает true

InputIterator find_if_not(InputIterator beg, InputIterator end, UnaryPredicate op)
Возвращает позицию первого элемента в диапапзоне [beg, end), для которого унарный предикат op возвращает false

Если искомых элементов нет, все алгоритмы возвращают позицию end.
Сложность: линейная.

Немодифицирующие алгоритмы
Поиск элементов
Поиск первых n последовательных совпадений

ForwardIterator search_n(ForwardIterator beg, ForwardIterator end, Size count, const &T value)
Возвращает позицию первого элемента из count элементов в диапазоне [begin, end), значение которых равно value.

ForwardIterator search_n(ForwardIterator beg, ForwardIterator end, Size count, const &T value, BinaryPredicate op)
Возвращает позицию первого элемента из count элементов в диапазоне [begin, end), для которого бинарный предикат op(elem, value) возвращает true.

Немодифицирующие алгоритмы
Поиск элементов
Поиск первых n последовательных совпадений

```cpp
#include <iostream>
#include <deque>
#include <algorithm>

using namespace std;

int main()
{
   deque<int> coll = {1,2,7,7,6,3,9,5,7,7,7,3,6};
   // поиск трех последовательных элементов со значением 7
   deque<int>::iterator pos;
   pos = search_n(coll.begin(), coll.end(), 3, 7);
   if (pos != coll.end()) {
       cout << "four consecutive elements with value 3 "
            << "start with " << distance(coll.begin(),pos) +1
            << ". element" << endl;
    }
}
```

Немодифицирующие алгоритмы
Поиск элементов
Поиск первого подынтервала

```
ForwardIterator1 search(ForwardIterator1 beg, ForwardIterator1 end,
                        ForwardIterator2 searchBeg, ForwardIterator2 searchEnd)

ForwardIterator1 search(ForwardIterator1 beg, ForwardIterator1 end,
                        ForwardIterator2 searchBeg, ForwardIterator2 searchEnd,
                        BinaryPredicate op)
```

Возвращают позицию первого элемента в превом подынтервале диапазона [beg, end), соответствующего диапазону [searchBeg, searchEnd).

В первом случае элементы подынтервала должны быть равны элементам диапазона.
Во втором случае при каждом сравнении элементов вызывается бинарный предикат (elem, searchElem), который должен возвращать true.

Немодифицирующие алгоритмы
Поиск элементов
Поиск первого подынтервала

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

using namespace std;

// проверяет четность элемента
bool checkEven (int elem, bool even)
{
    if (even) { return elem % 2 == 0; }
    else { return elem % 2 == 1; }
}

int main()
{
    vector<int> coll = {1,2,3,4,5,6,7,8,9};
    /* ищем подынтервал, в котором сначала идет четный элемент
    затем нечетный, затем снова четный */
    bool checkEvenArgs[3] = {true,false,true};
    vector<int>::iterator pos;
    pos = search (coll.begin(), coll.end(),
                  checkEvenArgs, checkEvenArgs+3, checkEven);

    cout << distance(coll.begin(),pos);
}
```

Немодифицирующие алгоритмы
Поиск элементов
Поиск последнего подынтервала

```
ForwardIterator1 ind_end(ForwardIterator1 beg, ForwardIterator1 end,
                         ForwardIterator2 searchBeg, ForwardIterator2 searchEnd)

ForwardIterator1 find_end(ForwardIterator1 beg, ForwardIterator1 end,
                          ForwardIterator2 searchBeg, ForwardIterator2 searchEnd,
                          BinaryPredicate op)
```

Немодифицирующие алгоритмы
Поиск элементов
Поиск первого из нескольких возможных элементов

ForwardIterator1 find_first_of(ForwardIterator1 beg, ForwardIterator1 end,
ForwardIterator2 searchBeg, ForwardIterator2 searchEnd)
Возвращает позицию первого элемента в диапазоне [beg, end), который также есть в диапазоне [searchBeg, searchEnd)

ForwardIterator1 find_first_of(ForwardIterator1 beg, ForwardIterator1 end,
ForwardIterator2 searchBeg, ForwardIterator2 searchEnd,
BinaryPredicate op)
Возвращает позицию первого элемента в диапазоне [beg, end), для которого любой вызов op(elem, searchElem) для всех элементов диапазона [searchBeg, searchEnd) выдает true.

Немодифицирующие алгоритмы
Поиск элементов
Поиск двух смежных элементов с одинаковыми значениями

ForwardIterator adjacent_find(ForwardIterator beg, ForwardIterator end)
Возвращает первый элемент в диапазоне [beg, end), значение которого равно значению следующего элемента.

ForwardIterator adjacent_find(ForwardIterator beg, ForwardIterator end, BinaryPredicate op)
Возвращает первый элемент в диапазоне [beg, end), для которого бинарный предикат op(elem, nextElem) возвращает значение true.

Немодифицирующие алгоритмы
Поиск элементов
Поиск двух смежных элементов с одинаковыми значениями

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// проверяем, является ли второй объект вдвое больше первого
bool doubled (int elem1, int elem2)
{
   return elem1 * 2 == elem2;
}

int main()
{
   vector<int> coll = {1,3,2,4,5,5,0};
   // ищем первые два элемента с одинаковыми значениями
   vector<int>::iterator pos;
   pos = adjacent_find(coll.begin(), coll.end());
   // ищем первые два элемента, таких, что второй элемент в два раза больше первого
   pos = adjacent_find(coll.begin(), coll.end(), doubled);
}
```

Немодифицирующие алгоритмы
Сравнение диапазонов
Проверка равенства

bool equal(InputIterator1 beg, InputIterator1 end, InputIterator2 cmpBeg)
Проверяет, равны ли элементы диапазона [beg, end) элементам диапазона, начинающегося с позиции cmpBeg

bool equal(InputIterator1 beg, InputIterator1 end, InputIterator2 cmpBeg, BinaryPredicate op)
Проверяет, возвращает ли значение true каждый вызов бинарного предиката op(elem, cmpElem) для соответствующих элементов в диапазоне [beg, end) в диапазоне, начинающемся с позиции cmpBeg

Немодифицирующие алгоритмы
Сравнение диапазонов
Проверка равенства

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>

using namespace std;

bool bothEvenOrOdd (int elem1, int elem2)
{
    return elem1 % 2 == elem2 % 2;
}

int main()
{
    cout << boolalpha;
    vector<int> coll1 = {1,2,3,4,5,6,7};
    list<int> coll2 = {3,4,5,6,7,8,9};
    // проверка, одинаковы ли коллекции
    bool eq = equal(coll1.begin(), coll1.end(), coll2.begin());
    cout << eq << endl; // выведет false, потому что коллекции не равны
    // проверяет четность и нечетность соответствующих элементов
    eq = equal(coll1.begin(), coll1.end(), coll2.begin(), bothEvenOrOdd);
    cout << eq; /* выведет true, потому что четности и нечетности соответствующих
    элементов в двух интервалах совпадают */
}
```

Немодифицирующие алгоритмы
Сравнение диапазонов
Проверка неупорядоченного равенства

bool is_permutation(ForwardIterator1 beg1, ForwardIterator1 end1, ForwardIterator2 beg2)
Проверяет, является ли диапазон [beg1, end1) перестановкой элементов диапазона, начинающегося с позиции beg2, сравнение с помощью операции ==

bool is_permutation(ForwardIterator1 beg1, ForwardIterator1 end1, ForwardIterator2 beg2, CompFunc op)
Аналогично, но сравнение элементов с помощью бинарного предиката op(elem1, elem2).

Немодифицирующие алгоритмы
Сравнение диапазонов
Поиск первого различия

pair<InputIterator1, InputIterator2> mismatch(InputIterator1 beg, InputIterator1 end,
InputIterator2 cmpBeg)

Возвращает позиции первых двух различающихся элементов в диапазоне [beg, end) и диапазоне, начинающемся с позиции cmpBeg.

pair<InputIterator1, InputIterator2> mismatch(InputIterator1 beg, InputIterator1 end,
BinaryPredicate op)

Возвращает позиции первых двух различающихся элементов в диапазоне [beg, end) и диапазоне, начинающемся с позиции cmpBeg, для которых предикат op(elem, cmpElem) выдает False.

Немодифицирующие алгоритмы
Сравнение диапазонов
Поиск первого различия

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll1 = {1,2,3,4,5,6};
    list<int> coll2 = {1,2,4,8,16,3};
    // ищем первое несовпадение
    pair<vector<int>::iterator,list<int>::iterator> values;
    values = mismatch(coll1.begin(), coll1.end(), coll2.begin());
    cout << "first mismatch: " << *values.first  << " and " <<
       *values.second << endl; // выведет 3 и 4

    /* ищем первую позицию, в которой элемент коллекции coll1 больше
       соответствующего элемента коллекции coll2 */
    values = mismatch(coll1.begin(), coll1.end(), coll2.begin(), less_equal<int>());
    cout << "not less-or-equal: " << *values.first << " and "
             << *values.second << endl; // выведет 6 и 3
}
```

Немодифицирующие алгоритмы
Сравнение диапазонов
Проверка «меньше чем»

bool lexicographical_compare(InputIterator1 beg1, InputIterator1 end1,
InputIterator2 beg2, InputIterator2 end2)
Проверка, что элементы диапазона [beg1, end1) лексикографически меньше элементов диапазона [beg2, end2), сравнение с помощью оператора <

bool lexicographical_compare(InputIterator1 beg1, InputIterator1 end1,
InputIterator2 beg2, InputIterator2 end2,
CompFunc op)
Проверка, что элементы диапазона [beg1, end1) лексикографически меньше элементов диапазона [beg2, end2), сравнение с помощью предиката op

Если одна из последовательностей не содержит элементов, то исчерпанная последовательность считается меньше другой.

Немодифицирующие алгоритмы
Предикаты для диапазонов
Проверка (частичного) упорядочения

bool is_sorted(ForwardIterator beg, ForwardIterator end)
Проверяет, упорядочены ли элементы в диапазоне [beg, end), сравнение с помощью <

bool is_sorted(ForwardIterator beg, ForwardIterator end, BinaryPredicate op)
Проверяет, упорядочены ли элементы в диапазоне [beg, end), сравнение с помощью op

ForwardIterator is_sorted_until(ForwardIterator beg, ForwardIterator end)
Возвращает позицию первого элемента в диапазоне [beg, end), нарушающего порядок в диапазоне, или end, если порядок не нарушается, сравнение с помощью <

ForwardIterator is_sorted_until(ForwardIterator beg, ForwardIterator end,
BinaryPredicate op)
Возвращает позицию первого элемента в диапазоне [beg, end), нарушающего порядок в диапазоне, или end, если порядок не нарушается, сравнение с помощью op

Немодифицирующие алгоритмы
Предикаты для диапазонов
Проверка разделения

bool is_partitioned(InputIterator beg, InputIterator end, UnaryPredicate op)
Проверяет, разделены ли элементы в диапазоне [beg, end) так, что все элементы, удовлетворяющие предикату op(), расположены перед элементами, которые этому предикату не удовлетворяют.

ForwardIterator partition_point(ForwardIterator beg, ForwardIterator end, UnaryPredicate op)
Возвращает позицию первого элемента в разделенном диапазоне.

Немодифицирующие алгоритмы
Предикаты для диапазонов
Проверка разделения

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

int main()
{
    cout << boolalpha;
    vector<int> coll = {5,3,9,1,3,4,8,2,6};
    // Проверка нечетности элемента
    auto isOdd = [](int elem){return elem%2==1;};
    cout << is_partitioned(coll.begin(),coll.end(),isOdd) << endl;
    // Ищем первый четный элемент
    auto pos = partition_point(coll.begin(), coll.end(), isOdd);
    cout << *pos << endl; // выведет 4
    return 0;
}
```

Немодифицирующие алгоритмы
Предикаты для диапазонов
Все, хотя бы один или ни одного

bool all_of(InputIterator beg, InputIterator end, UnaryPredicate op)

bool any_of(InputIterator beg, InputIterator end, UnaryPredicate op)

bool none_of(InputIterator beg, InputIterator end, UnaryPredicate op)

Эти алгоритмы возвращают результат проверки того, что унаррный предикат op(elem) возвращает значение true для всех, хотя бы одного или ни для одного из элементов диапазона [beg, end).

Если диапазон пуст, то алгоритмы all_of() и none_of() возвращает значение true, а алгоритм any_of — значение false.

Модифицирующие алгоритмы
Копирование элементов

OutputIterator copy(InputIterator sourceBeg, InputIterator sourceEnd, outputIterator destBeg)

OutputIterator copy_if(InputIterator sourceBeg, InputIterator sourceEnd,
outputIterator destBeg, UnaryPredicate op)

OutputIterator copy_n(InputIterator sourceBeg, Size num, OutputIterator destBeg)

BidirectionalIterator2 copy_backward(BidirectionalIterator1 sourceBeg,
BidirectionalIterator1 sourceEnd,
BidirectionalIterator2 destEnd)

Эти алгоритмы копируют все элементы диапазона-источника [sourceBeg, sourceEnd), или num элементов, начиная с позиции sourceBeg, в диапазон-получатель, начиная с позиции destBeg или заканчивая позицией destEnd. Алгоритмы возвращают позицию, находящуюся после последнего скопированного элемента в диапазоне-получателе. Алгоритм copy выполняет прямой обход последовательности, а алгоритм copy_backward — обратный обход.

Модифицирующие алгоритмы
Копирование элементов

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <string>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<string> coll1 = {"Hello", "this", "is", "an", "example"};
    vector<string> coll2;
    // копируем элементы coll1 в coll2
    // используем итератор вставки, чтобы выполнить вставку, а не замену
    copy(coll1.begin(), coll1.end(), back_inserter(coll2));
    // вывод на экран элементов coll2
    copy(coll2.begin(), coll2.end(), ostream_iterator<string>(cout," "));
    cout << endl;
    // копируем элементы coll1 в coll2 в обратном порядке
    // теперь заменяем исходные элементы копируемыми
    copy(coll1.rbegin(), coll1.rend(), coll2.begin());
    copy (coll2.begin(), coll2.end(), ostream_iterator<string>(cout," "));
}
```

Модифицирующие алгоритмы
Перемещение элементов

OutputIterator move(InputIterator sourceBeg, InputIterator sourceEnd, outputIterator destBeg)

BidirectionalIterator2 move_backward(BidirectionalIterator1 sourceBeg,
BidirectionalIterator1 sourceEnd,
BidirectionalIterator2 destEnd)

Модифицирующие алгоритмы
Преобразование элементов

OutputIterator transform(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg, UnaryFunc op)

Вызывает предикат op(elem) для каждого элемента в диапазоне-источнике [sourceBeg, sourceEnd) и записывает результат каждого вызова предиката op в диапазон-получатель, начиная с позиции destBeg.

```
#include <iostream>
#include <vector>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll1 = {1,2,3,4,5};
    list<int> coll2 = {10,20,40};
    // изменяем знак всех элементов в coll1
    transform (coll1.begin(), coll1.end(), coll1.begin(), negate<int>());
    // выводим элементы контейнера coll2 с противоположным знаком
    transform (coll2.rbegin(), coll2.rend(),
               ostream_iterator<int>(cout," "),
               negate<int>());
}
```

Модифицирующие алгоритмы
Объединение элементов двух последовательностей

OutputIterator transform(InputIterator1 source1Beg, InputIterator1 source1End,
InputIterator2 source2Beg, OutputIterator destBeg, UnaryFunc op)

Вызывает предикат op(source1Elem, source2Elem) ко всем соответствующим элементам из первого диапазона-источника [source1Beg, source1End) и второго диапазона-источника, начиная с позиции source2Beg, и записывает результат каждого вызова в диапазон-получатель, начиная с позиции destBeg.

Модифицирующие алгоритмы
Объединение элементов двух последовательностей

```cpp
#include <iostream>
#include <vector>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll1 = {1,2,3,4,5,6,7,8,9};
    list<int> coll2;
    // возведение каждого элемента  в квадрат
    transform (coll1.begin(), coll1.end(),
               coll1.begin(),
               coll1.begin(),
               multiplies<int>());
    // получим 1,4,9,16,25,36,49,64,81
    /* суммируем каждый элемент в порядке прямого обхода с каждым элементом
       в порядке обратного обхода и вставляем результат в контейнер coll2 */
    transform (coll1.begin(), coll1.end(),       // first source range
               coll1.rbegin(),                   // second source range
               back_inserter(coll2),             // destination range
               plus<int>());                     // operation
    // получим 82, 68, 58, 52, 50, 52, 58, 68, 82
}
```

Модифицирующие алгоритмы
Обмен элементов

ForwardIterator2 swap_ranges(ForwardIterator1 beg1, ForwardIterator1 end1,
ForwardIterator2 beg2)

Обменивает элементы диапазона [beg1, end1) с соответствующими элементами диапазона, начинающегося с позиции beg2. Оба диапазона не должны перекрываться.

Модифицирующие алгоритмы

Присвоение новых значений

Заполнение одним и тем же значением

OutputIterator **fill**(ForwardIterator beg, ForwardIterator end, const T& newValue)
Присваивает значение newValue каждому элементу в диапазоне [beg, end)
OutputIterator **fill_n**(OutputIterator beg, Size num, const T& newValue)
Присваивает значение newValue первым num элементам в диапазоне, начинающемся с позиции beg

Присвоение сгенерированных значений

OutputIterator **generate**(ForwardIterator beg, ForwardIterator end, Func op)
Присваивает значения, сгенерированные вызовом op() для каждого элемента в диапазоне [beg, end)
OutputIterator **generate_n**(OutputIterator beg, Size num, Func op)
Присваивает значения, сгенерированные вызовом op первым num элементам в диапазоне, начинающемся с позиции beg

Присвоение последовательности возрастающих чисел

void iota(ForwardIterator beg, ForwardIterator end, T startValue)
Присваивает значения startValue, startValue+1, startValue+2, и т.д.

Модифицирующие алгоритмы
Присвоение сгенерированных значений

```cpp
#include <cstdlib>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    list<int> coll;
    // вставка в список пяти случайных значений
    generate_n (back_inserter(coll), 5, rand);
    // заменяем элементы пятью случайными числами
    generate (coll.begin(), coll.end(), rand);
}
```

Модифицирующие алгоритмы

Замена элементов

Замена значений в последовательности

void **replace**(ForwardIterator beg, ForwardIterator end, const T& oldValue, const T& newValue)
Заменяет значением newValue каждый элемент диапазона [beg, end), равный oldValue

void **replace_if**(ForwardIterator beg, ForwardIterator end, UnaryPredicate op, const T& newValue)
Заменяет значением newValue каждый элемент диапазона [beg, end), для которого унарный предикат op(elem) возвращает значение true

Копирование и замена элементов

OutputIterator **replace_copy**(InputIterator sourceBeg, InputIterator sourceEnd,
OutputIterator destBeg, const T& oldValue, const T& newValue)
Заменяет значением newValue каждый элемент диапазона-источника [sourceBeg, sourceEnd), равный значению oldValue, и копирует эти элементы в диапазон-получатель, начинающийся с позиции destBeg

OutputIterator **replace_copy_if**(InputIterator sourceBeg, InputIterator sourceEnd,
OutputIterator destBeg, UnaryPredicate op, const T& newValue)
Заменяет значением newValue каждый элемент диапазона-источника [sourceBeg, sourceEnd), для которого унарный предикат op(elem) возвращает значение true, и копирует эти элементы в диапазон-получатель, начинающийся с позиции destBeg

Алгоритмы удаления

Удаление определенных значений

Удаление элементов из последовательности

ForwardIterator **remove**(ForwardIterator beg, ForwardIterator end, const T& value)
Удаляет все элементы диапазона [beg, end), имеющие значение value.

ForwardIterator **remove_if**(ForwardIterator beg, ForwardIterator end, UnaryPredicate op)
Удаляет все элементы диапазона [beg, end), для которых унарный предикат op(elem) возвращает true.

Удаление элементов при копировании

OutputIterator **remove_copy**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg,
const T& value)
Копирует каждый элемент диапазона-источника [sourceBeg, sourceEnd), не равный значению value, в диапазон-получатель, начинающийся с позиции destBeg.

OutputIterator **remove_copy_if**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg,
UnaryPredicate op)
Копирует каждый элемент диапазона-источника [sourceBeg, sourceEnd), для которого унарный предикат op(elem) возвращает false, в диапазон-получатель, начинающийся с позиции destBeg.

Алгоритмы удаления
Удаление дубликатов
Удаление соседних дубликатов

ForwardIterator unique(ForwardIterator beg, ForwardIterator end)
Удаляет из диапазона [beg, end) элементы, равные предыдущим элементам (если есть дубликаты, которые не соседние, они удаляться не будут)

ForwardIterator unique(ForwardIterator beg, ForwardIterator end, BinaryPredicate op)
Удаляет все элементы, следующие за элементом e, для которых бинарный предикат op(e, elem) возвращает true

Оба алгоритма возвращают новый конец модифицированной последовательности

Алгоритмы удаления
Удаление дубликатов
Удаление соседних дубликатов

```cpp
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    int source[] = {1,4,4,6,1,2,2,3,1,6,6,6,5,7,5,4,4};
    int sourceNum = sizeof(source)/sizeof(source[0]);
    list<int> coll;
    // инициализируем coll элементами из source
    copy(source, source+sourceNum, back_inserter(coll));
    // удаляем последовательные дубликаты
    list<int>::iterator pos;
    pos = unique (coll.begin(), coll.end());
    copy(coll.begin(), pos, ostream_iterator<int>(cout," "));
    // выведет 1 4 6 1 2 3 1 6 5 7 5 4
    cout << endl;
    // повторно инициализируем coll элементами из source
    copy(source, source+sourceNum, coll.begin());
    // удаляем элемент, если ему предшествует элемент с большим значением
    pos = unique(coll.begin(), coll.end(), greater<int>());
    copy(coll.begin(), pos, ostream_iterator<int>(cout," "));
    // выведет 1 4 4 6 6 6 6 7
}
```

Алгоритмы удаления
Удаление дубликатов
Удаление дубликатов при копировании

OutputIterator **unique_copy**(InputIterator sourceBeg, InputIterator sourceEnd,
OutputIterator destBeg)

OutputIterator **unique_copy**(InputIterator sourceBeg, InputIterator sourceEnd,
OutputIIterator destBeg, BinaryPredicate op)

Копирование элементов [sourceBeg, sourceEnd), у которых нет дубликатов

Перестановочные алгоритмы
Перестановка элементов в обратном порядке

void **reverse**(BidirectionalIterator beg, BidirectionalIterator end)
Перестановка элементов в обратном порядке

void **reverse_copy**(BidirectionalIterator sourceBeg, Bidirectionalterator sourceEnd,
OutputIterator destBeg)
Перестановка элементов в обратном порядке, при этом копирование их из диапазона-источника [beg, end) в диапазон-получатель, начинающийся с позиции destBeg

Перестановочные алгоритмы
Перестановка элементов в обратном порядке

```cpp
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll = {1,2,3,4,5,6,7,8,9};
    // переставляем элементы в обратном порядке от второго до предпоследнего
    reverse (coll.begin()+1, coll.end()-1);
    // получим {1,8,7,6,5,4,3,2,9}
    // вывести элементы в обратном прядке
    reverse_copy (coll.begin(), coll.end(), ostream_iterator<int>(cout," "));
    // выведет 9 2 3 4 5 6 7 8 1
}
```

Перестановочные алгоритмы

Циклическая перестановка элементов в последовательности

ForwardIterator **rotate**(ForwardIterator beg, ForwardIterator newBeg, ForwardIterator end)

Выполняет циклическую перестановку элементов в диапазоне [beg, end) так, что newBeg — новый первый элемент после вызова

Циклический сдвиг элементов при копировании

OutputIterator **rotate_copy**(ForwardIterator sourceBeg, ForwardIterator newBeg,
ForwardIterator sourceEnd, OutputIterator destBeg)

Копирует элементы диапазона-источника [sourceBeg, sourceEnd) в диапазон-получатель, начинающийся с позиции destBeg, с циклическим сдвигом так, что *newBeg становится новым первым элементом

Перестановочные алгоритмы

Циклическая перестановка элементов в последовательности

```cpp
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll = {1,2,3,4,5,6,7,8,9};
    // выполняем циклический сдвиг на один элемент влево
    rotate (coll.begin(),       // начало диапазона
            coll.begin() + 1,   // новый первый элемент
            coll.end());        // конец диапазона
    for (auto elem: coll)
        cout << elem << ' '; // выведет 2,3,4,5,6,7,8,9,1
    cout << endl;
    // циклический сдвиг на два элемента вправо
    rotate (coll.begin(),       // начало диапазона
            coll.end() - 2,     // новый первый элемент
            coll.end());        // конец диапазона
    // получим {9,1,2,3,4,5,6,7,8}
    // циклический сдвиг таким образом, что элемент 4 стал первым
    rotate (coll.begin(),                       // начало диапазона
            find(coll.begin(),coll.end(),4),    // новый первый элемент
            coll.end());                        // конец диапазона
    // получим {4,5,6,7,8,9,1,2,3}
}
```

Перестановки, лексикографический порядок

Пусть есть первая перестановка (например, 1234)
Для нахождения каждой следующей:

1. Сканируем текущую перестановку справо налево в поисках первой пары соседних элементов таких, что a[i] < a[i+1]. Для перестановки 1234 это число 3 (3<4).

2. Находим наименьший элемент из «хвоста», больший [i], и перемещаем его на позицию i. В данном случае на место 3 ставим 4.

3. Позиции с i+1 по n заполняем элементами a[i], a[i+1], …, a[n], из которых изъят помещенный в позицию i элемент, в возрастающем порядке.

1234
1243
1324
1342
1423
1432
2134
2143
2314
2341
2413
2431
…

(всего 4!=24)

Перестановочные алгоритмы
Перестановка элементов

Изменение порядка следования элементов в диапазоне [beg, end) в соответствии со следующей или предыдущей перестановкой.

bool **next_permutation**(BidirectionalIterator beg, BidirectionalIterator end)

bool **next_permutation**(BidirectionalIterator beg, BidirectionalIterator end, BinaryPredicate op)

bool **prev_permutation**(BidirectionalIterator beg, BidirectionalIterator end)

bool **prev_permutation**(BidirectionalIterator beg, BidirectionalIterator end, BinaryPredicate op)

Алгоритмы возвращают значение false, если элементы образуют возрастающий порядок для next_permutation и убывающий для prev_permutation, в остальных случаях — true.

Перестановочные алгоритмы
Перетасовка элементов

void **shuffle**(RandomAccessIterator beg, RandomAccessIterator end,
UniformRandomNumberGenerator gen)
Случайным образом изменяет порядок следования элементов диапазона, используя генератор случайных чисел из библиотеки

void **random_shuffle**(RandomAccessIterator beg, RandomAccessIterator end)
Случайным образом изменяет порядок следования элементов диапазона, используя зависящий от реализации генератор случайных чисел

void **random_shuffle**(RandomAccessIterator beg, RandomAccessIterator end, RandomFunc&& op)
Случайным образом изменяет порядок следования элементов диапазона, используя функцию op для генерации случайных чисел

Перестановочные алгоритмы

Перетасовка элементов

```cpp
#include <cstdlib>
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>

using namespace std;

class MyRandom {
  public:
    ptrdiff_t operator() (ptrdiff_t max) {
        double tmp;
        tmp = static_cast<double>(rand())
                / static_cast<double>(RAND_MAX);
        return static_cast<ptrdiff_t>(tmp * max);
    }
};

int main()
{
    vector<int> coll = {1,2,3,4,5,6,7,8,9};
    random_shuffle(coll.begin(), coll.end()); // случайная перетасовка элементов
    for (auto elem: coll)
        cout << elem << ' ';
    cout << endl;
    sort(coll.begin(), coll.end()); // сортировка по возрастанию
    // перетасовываем элементы с помощью собственного генератора случайных чисел
    MyRandom rd;
    random_shuffle (coll.begin(), coll.end(),   // диапазон
                    rd);                        // генератор
    for (auto elem: coll)
        cout << elem << ' ';
    cout << endl;
}
```

Перестановочные алгоритмы

Перемещение элементов в начало

ForwardIterator **partition**(ForwardIterator beg, ForwardIterator end, UnaryPredicate op)
Перемещает в начало все элементы диапазона [beg, end), для которых унарный предикат op(elem) возвращает значение true

ForwardIterator **stable_partition**(ForwardIterator beg, ForwardIterator end, UnaryPredicate op)
Перемещает в начало все элементы диапазона [beg, end), для которых унарный предикат op(elem) возвращает значение true, при этом сохраняет относительный порядок следования элементов, соответствующих и не соответствующих критерию

Разделение на два подынтервала

pair<OutputIterator1, OutputIterator2> **partition_copy**(InputIterator sourceBeg, InputIterator sourceEng,
OutputIterator1 destTrueBeg, OutputIterator2 destFalseBeg, UnaryPredicate op)

Разделяет все элементы диапазона [beg, end) на два подынтервала в соответствии с предикатом op. Все элементы, для которых унарный предикат op(elem) возвращает true, копируется в диапазон, начинающийся с позиции destTrueBeg. Все элементы, для которых предикат возвращает значение false, копируются в диапазон, начинающийся с destFalseBeg

Перестановочные алгоритмы
Перемещение элементов в начало

```cpp
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    vector<int> coll1 = {1,2,3,4,5,6,7,8,9};
    vector<int> coll2 = {1,2,3,4,5,6,7,8,9};
    // перемещаем четные элементы в  начало
    vector<int>::iterator pos1, pos2;
    pos1 = partition(coll1.begin(), coll1.end(),          // диапазон
                     [](int elem){return elem%2==0;});   // критерий
    pos2 = stable_partition(coll2.begin(), coll2.end(),       // диапазон
                     [](int elem){return elem%2==0;}); // критерий
    for (auto elem: coll1)
        cout << elem << ' '; // выведет 8 2 6 4 5 3 7 1 9
    cout << endl;
    for (auto elem: coll2)
        cout << elem << ' '; // выведет 2 4 6 8 1 3 5 7 9
}
```

Алгоритмы сортировки
Сортировка всех элементов

void **sort**(RandomAccessIterator beg, RandomAccessIterator end)

void **sort**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

void **stable_sort**(RandomAccessIterator beg, RandomAccessIterator end)

void **stable_sort**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

stable_sort — гарантирует, что порядок одинаковых элементов остается неизменным

Алгоритмы сортировки
Сортировка всех элементов

```cpp
#include <iostream>
#include <deque>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    deque<int> coll = {1,2,3,4,5,1,2,3,4,5};
    // сортировка
    sort(coll.begin(), coll.end());
    // 1 1 2 2 3 3 4 4 5 5
    // сортировка по убыванию
    sort (coll.begin(), coll.end(),     // диапазон
          greater<int>());              // критерий сортировки
    for (auto elem: coll)
        cout << elem << ' ';
    // 5 5 4 4 3 3 2 2 1 1
}
```

Алгоритмы сортировки
Сортировка всех элементов

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <iterator>
#include <algorithm>

using namespace std;

bool lessLength (const string& s1, const string& s2)
{
    return s1.length() < s2.length();
}

int main()
{
    vector<string> coll1 = {"1xxx", "2x", "3x", "4x", "5xx", "6xxxx", "7xx", "8xxx",
    "9xx", "10xxx", "11", "12", "13", "14xx", "15", "16", "17"};
    vector<string> coll2 = coll1;
    // сортировка по длине строк
    sort (coll1.begin(), coll1.end(),           // диапазон
          lessLength);                          // критерий
    for (auto elem: coll1)
        cout << elem << ' ';
    // выведет 2x 17 16 15 13 12 11 4x 3x 9xx 7xx 5xx 8xxx 14xx 1xxx 10xxx 6xxxx
    cout << endl;
    stable_sort (coll2.begin(), coll2.end(),    // диапазон
                 lessLength);                   // критерий
    for (auto elem: coll2)
        cout << elem << ' ';
    // выведет 2x 3x 4x 11 12 13 15 16 17 5xx 7xx 9xx 1xxx 8xxx 14xx 6xxxx 10xxx
}
```

Алгоритмы сортировки
Частичная сортировка

Сортировка элементов диапазона [beg, end) так, что диапазон [beg, sortEnd) содержит упорядоченные элементы

void **partial_sort**(RandomAccessIterator beg, RandomAccessIterator sortEnd, RandomAccessIterator end)

void **partial_sort**(RandomAccessIterator beg, RandomAccessIterator sortEnd, RandomAccessIterator end,
BinaryPredicate op)

Копирование упорядоченных элементов из диапазона-источника [sourceBeg, sourceEnd) в диапазон-получатель [destBeg, destEnd)

RandomAccessIterator **partial_sort_copy**(InputIterator sourceBeg, InputIterator sourceEnd,
RandomAccessIterator destBeg, RandomAccessIterator destEnd)

RandomAccessIterator **partial_sort_copy**(InputIterator sourceBeg, InputIterator sourceEnd,
RandomAccessIterator destBeg, RandomAccessIterator destEnd,
BinaryPredicate op)

Алгоритмы сортировки
Сортировка по n-му элементу

Сортировка элементов диапазона [beg, end) так, чтобы правильный элемент находился в n-й позиции, а все элементы, стоящие перед ним, не превосходили его по величине или были равны ему, а все элементы, стоящие после него, превосходили его по величине или были равны ему. В качестве критерия сортировки можно использовать бинарный предикат.

void **nth_element**(RandomAccessIterator beg, RandomAccessIterator nth, RandomAccessIterator end)

void **nth_element**(RandomAccessIterator beg, RandomAccessIterator nth, RandomAccessIterator end,
BinaryPredicate op)

Алгоритмы сортировки

Сортировка по n-му элементу

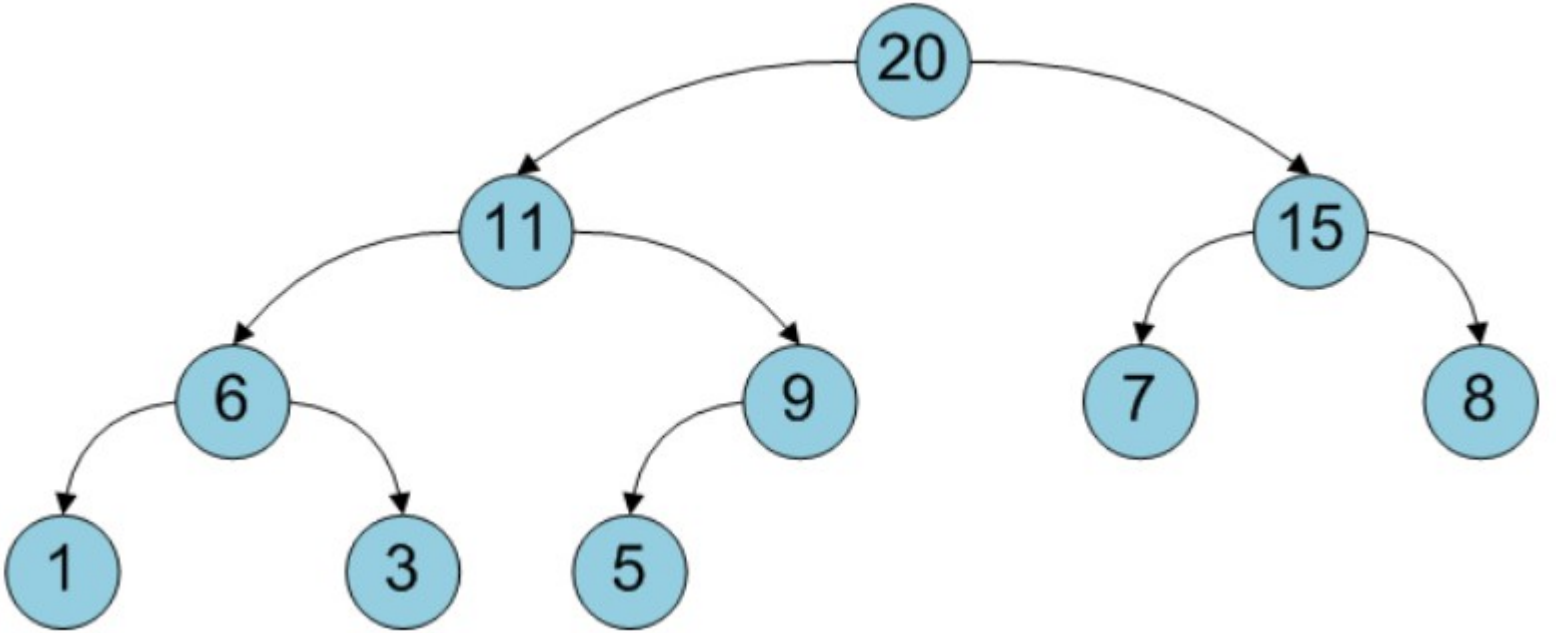
```cpp
#include <iostream>
#include <deque>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    deque<int> coll = {3,4,5,6,7,2,3,4,5,6,1,2,3,4,5};
    // находим четыре наименьших элемента
    nth_element(coll.begin(),     // начало диапазона
                coll.begin()+3,   // граничный элемент
                coll.end());      // конец диапазона
    // вывод четырех наименьших элементов
    copy(coll.begin(), coll.begin()+4, ostream_iterator<int>(cout," "));
    cout << endl;
    // выведет 2 1 2 3
    // находим четыре наибольших элемента
    nth_element(coll.begin(),     // начало диапазона
                coll.end()-4,     // граничный элемент
                coll.end());      // конец диапазона
    // вывод четырех наибольших элементов
    copy(coll.end()-4, coll.end(), ostream_iterator<int>(cout," "));
    cout << endl;
    // находим четыре наибольших элемента (теперь сравнение с помощью >)
    nth_element(coll.begin(),     // начало диапазона
                coll.begin()+3,   // граничный элемент
                coll.end(),       // конец диапазона
                greater<int>());  // критерий сортировки
    copy(coll.begin(), coll.begin()+4, ostream_iterator<int>(cout," "));
}
```

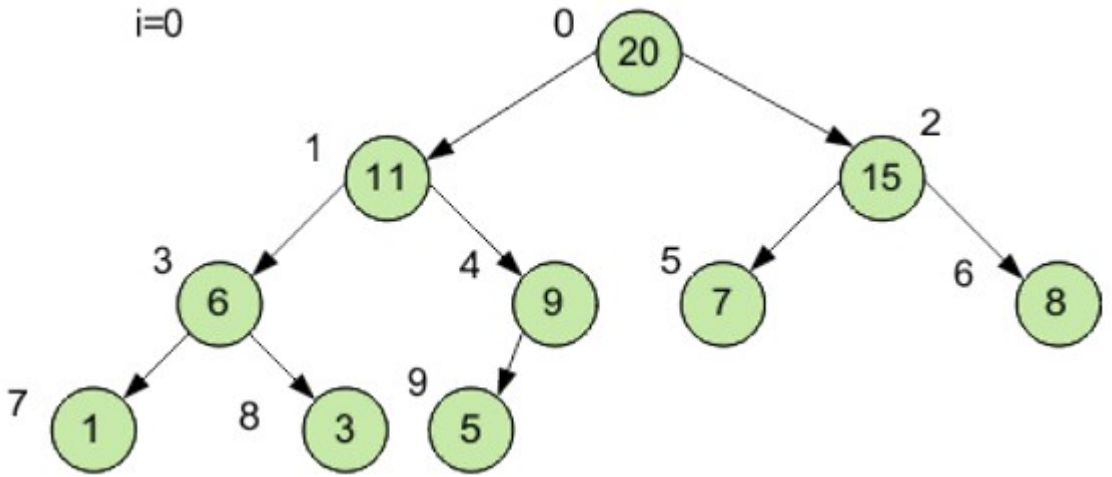
Куча

Куча — это структура бинарное дерево, для которого выполняется основное свойство кучи: приоритет каждой вершины больше приоритетов ее потомков.



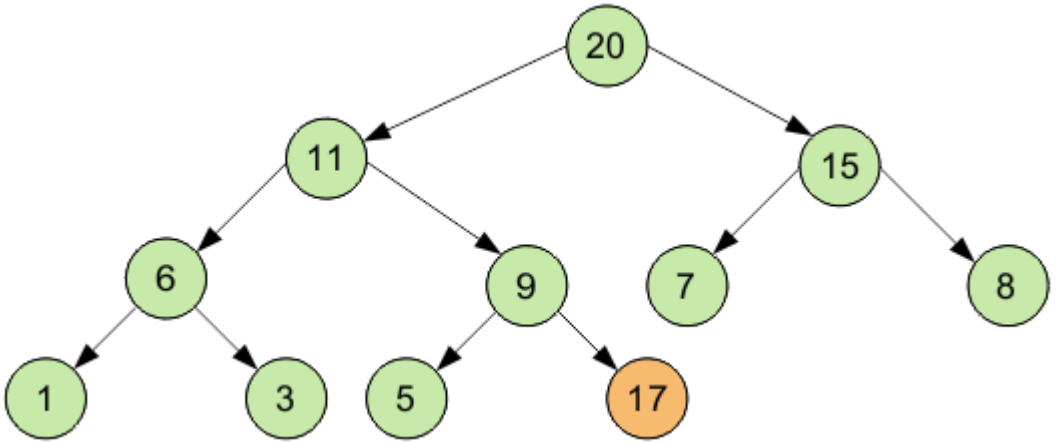
Куча

Двоичную кучу удобно хранить в виде одномерного массива
левый потомок вершины с индексом i имеет индекс 2*i+1,
правый потомок вершины с индексом i имеет индекс 2*i+2.
Корень дерева (кучи) – элемент с индексом 0.

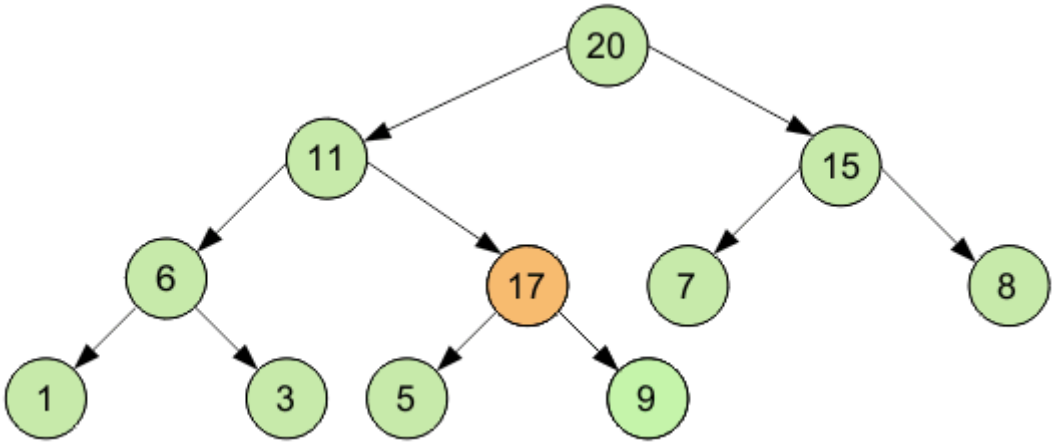


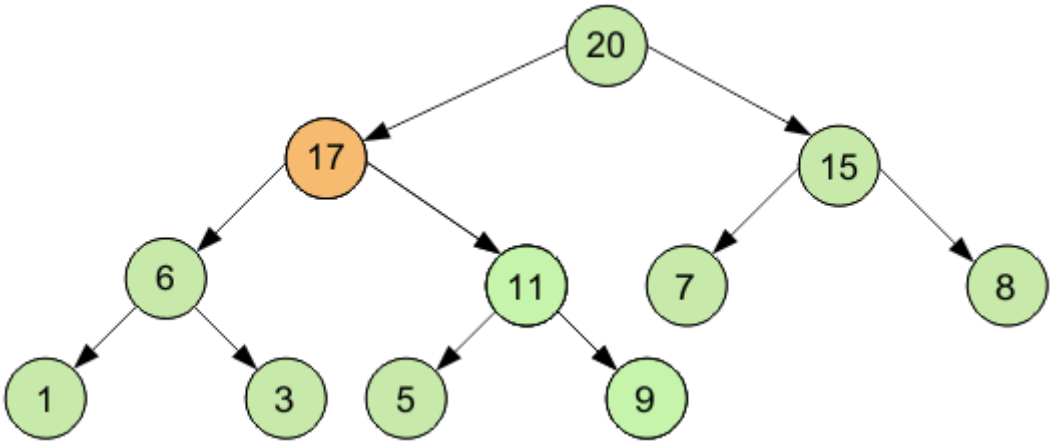
Куча. Добавление элемента

Новый элемент добавляется на последнее место в массиве, то есть позицию с максимальным индексом.



Куча. Добавление элемента





Возможно, что при этом будет нарушено основное свойство кучи, так как новый элемент может быть больше родителя. В таком случае новый элемент «поднимается» на один уровень (менять с вершиной-родителем) до тех пор, пока не будет соблюдено основное свойство кучи.

Алгоритмы сортировки
Алгоритмы для работы с кучей

Преобразование элементов диапазона [beg, end) в кучу
void **make_heap**(RandomAccessIterator beg, RandomAccessIterator end)
void **make_heap**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

Добавление последнего элемента, стоящего перед позицией end, в существующую кучу в диапазоне [beg, end-1), так что весь диапазон [beg, end) становится кучей.
void **push_heap**(RandomAccessIterator beg, RandomAccessIterator end)
void **push_heap**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

Перемещение наибольшего элемента кучи [beg, end), который является ее первым элементом, в последнюю позицию, и создание новой кучи из элементов [beg, end-1)
void **pop_heap**(RandomAccessIterator beg, RandomAccessIterator end)
void **pop_heap**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

Преобразование кучи в упорядоченную последовательность
void **sort_heap**(RandomAccessIterator beg, RandomAccessIterator end)
void **sort_heap**(RandomAccessIterator beg, RandomAccessIterator end, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Поиск элементов

Проверка наличия элементов

Проверяют, содержит ли упорядоченный диапазон элемент, равный value

bool binary_search(ForwardIterator beg, ForwardIterator end, const &T value)

bool binary_search(ForwardIterator beg, ForwardIterator end, const &T value, BinaryPredicate op)

Проверка наличия нескольких элементов

Проверяют, содержит ли упорядоченный диапазон [beg, end) все элементы упорядоченного диапазона [searchBeg, searchEnd), т. е. существует ли для каждого элемента в диапазоне [searchBeg, searchEnd) равный ему элемент в диапазоне [beg, end)

bool includes(InputIterator1 beg, InputIterator1 end,
InputIterator2 searchBeg, InputIterator2 searchEnd)

bool includes(InputIterator1 beg, InputIterator1 end,
InputIterator2 searchBeg, InputIterator2 searchEnd, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Поиск первой или последней возможной позиции

Возвращают позицию первого элемента, который не меньше значения value.

ForwardIterator **lower_bound**(ForwardIterator beg, ForwardIterator end, const T& value)

ForwardIterator **lower_bound**(ForwardIterator beg, ForwardIterator end, const T& value,
BinaryPredicate op)

Возвращают позицию последнего элемента, превышающего значение value

ForwardIterator **upper_bound**(ForwardIterator beg, ForwardIterator end, const T& value)

ForwardIterator **upper_bound**(ForwardIterator beg, ForwardIterator end, const T& value,
BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Поиск первой и последней возможной позиции

Возвращают позицию первого элемента, который не меньше значения value и последнего элемента, превышающего значение value.

pair<ForwardIterator, ForwardIterator> **equal_range**(ForwardIterator beg, ForwardIterator end, const T& value)

pair<ForwardIterator, ForwardIterator> **equal_range**(ForwardIterator beg, ForwardIterator end, const T& value, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Слияние двух упорядоченных множеств

Слияние элементов двух упорядоченных диапазонов так, что диапазон-получатель, начинающийся с позиции destBeg, содержит все элементы, содержащиеся в первом источнике, и все элементы, содержащиеся во втором источнике.

Например, вызов алгоритма merge для последовательностей 1 2 2 4 6 7 7 9 и 2 2 2 3 6 6 6 8 9 вычисляет следующий результат: 1 2 2 2 2 2 3 4 6 6 6 7 7 8 9 9

OutputIterator **merge**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg)

OutputIterator **merge**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Слияние двух упорядоченных множеств

```cpp
#include <iostream>
#include <list>
#include <set>
#include <iterator>
#include <algorithm>

using namespace std;

int main()
{
    list<int> coll1 = {1,2,3,4,5,6};
    set<int> coll2 = {3,4,5,6,7,8,9};
    // вывести на экран совмещенную последовательность
    merge (coll1.begin(), coll1.end(),
           coll2.begin(), coll2.end(),
           ostream_iterator<int>(cout," "));
    // выведет 1 2 3 3 4 4 5 5 6 6 7 8 9
}
```

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Объединение двух упорядоченных последовательностей

Слияние элементов двух упорядоченных диапазонов так, что диапазон-получатель, начинающийся с позиции destBeg, содержит все элементы, содержащиеся в первом источнике, все элементы, содержащиеся во втором источнике или все элементы, содержащиеся в обоих диапазонах.

Например, вызов алгоритма set_union для последовательностей 1 2 2 4 6 7 7 9 и 2 2 2 3 6 6 6 8 9 вычисляет следующий результат: 1 2 2 2 3 4 6 6 7 7 8 9

OutputIterator **set_union**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg)

OutputIterator **set_union**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Пересечение двух упорядоченных множеств

Слияние элементов двух упорядоченных диапазонов так, что диапазон-получатель, начинающийся с позиции destBeg, содержит все элементы, содержащиеся в обоих диапазонах.

Например, вызов алгоритма set_intersection для последовательностей 1 2 2 4 6 7 7 9
и 2 2 2 3 6 6 6 8 9 вычисляет следующий результат: 2 2 6 9

OutputIterator **set_intersection**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg)

OutputIterator **set_intersection**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Разность между двумя упорядоченными множествами

Слияние элементов двух упорядоченных диапазонов так, что диапазон-получатель, начинающийся с позиции destBeg, содержит все элементы, содержащиеся в первом, но не во втором диапазоне—источнике.

Например, вызов алгоритма set_intersection для последовательностей 1 2 2 4 6 7 7 9
и 2 2 2 3 6 6 6 8 9 вычисляет следующий результат: 1 4 7 7

OutputIterator **set_difference**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg)

OutputIterator **set_difference**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg, BinaryPredicate op)

Алгоритмы для упорядоченных диапазонов

Слияние диапазонов

Разность между двумя упорядоченными множествами

Слияние элементов двух упорядоченных диапазонов так, что диапазон-получатель, начинающийся с позиции destBeg, содержит все элементы, содержащиеся либо в первом, либо во втором источнике, но не в обоих одновременно.

Например, вызов алгоритма set_intersection для последовательностей 1 2 2 4 6 7 7 9
и 2 2 2 3 6 6 6 8 9 вычисляет следующий результат: 1 2 3 4 6 7 7 8

OutputIterator **set_symmetric_difference**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg)

OutputIterator **set_symmetric_difference**(InputIterator sourceBeg1, InputIterator sourceEnd1,
InputIterator sourceBeg2, InputIterator sourceEnd2, OutputIterator destBeg, BinaryPredicate op)

Численные алгоритмы

Вычисление

Вычисление результата по одной последовательности

T **accumulate**(InputIterator beg, InputIterator end, T initValue)
Вычисляет и возвращает сумму значения initValue и всех элементов диапазона [beg, end). В частности, она применяет к каждому элементу следующую операцию:
initValue = initValue + elem.

T **accumulate**(InputIterator beg, InputIterator end, T initValue, BinaryFunc op)
Вычисляет и возвращает результат применения операции op к значению initValue и всем элементам диапазона [beg, end). В частности, она применяет к каждому элементу следующую операцию: initValue = op(initValue, elem).

Численные алгоритмы

Вычисление

Вычисление результата по одной последовательности

```cpp
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <numeric>

using namespace std;

int main()
{
    vector<int> coll = {1,2,3,4,5,6,7,8,9};
    // вычисление суммы элементов
    cout << "sum: "
         << accumulate(coll.begin(), coll.end(),     // диапазон
                       0)                            // начальное значение
         << endl;
    // вычисление произведения элементов
    cout << "product: "
         << accumulate (coll.begin(), coll.end(),    // диапазон
                        1,                           // начальное значение
                        multiplies<int>());          // операция
}
```

Численные алгоритмы

Вычисление

Вычисление скалярного произведение двух последовательностей

T **inner_product**(InputIterator1 beg1, InputIterator end1, InputIterator beg2, T initValue)
Вычисляет и возвращает сумму значения initValue и скалярного произведения элементов диапазона [beg, end) и диапазона, начинающегося с позиции beg2. В частности, она применяет к каждому элементу следующую операцию:

initValue = initValue + elem1*elem2

T **inner_product**(InputIterator1 beg1, InputIterator end1, InputIterator beg2, T initValue,
BinaryFunc op1, BinaryFunc op2)
Вычисляет и возвращает результат применения операции op к значению initValue и всем элементам диапазона [beg, end) и диапазона, начинающегося с позиции beg2. В частности, она применяет ко всем парам соответствующих элементов операции op1 и op2:

initValue = op1(initValue, op2(elem1, elem2))

Численные алгоритмы

Вычисление

Вычисление скалярного произведение двух последовательностей

```cpp
#include <iostream>
#include <list>
#include <iterator>
#include <algorithm>
#include <numeric>

using namespace std;

int main()
{
    list<int> coll = {1,2,3,4,5,6};
    // сумма поэлементных произведений
    // (0 + 1*1 + 2*2 + 3*3 + 4*4 + 5*5 + 6*6)
    cout << "inner product: "
         << inner_product (coll.begin(), coll.end(),  // первый диапазон
                          coll.begin(),               // второй диапазон
                          0)                          // начальное значение
         << endl;
    // вычисляем сумму (0 + 1*6 + 2*5 + 3*4 + 4*3 + 5*2 + 6*1)
    cout << "inner reverse product: "
         << inner_product (coll.begin(), coll.end(),  // первый диапазон
                          coll.rbegin(),              // второй диапазон
                          0)                          // начальное значение
         << endl;
    // вычисление произведения сумм (1 * 1+1 * 2+2 * 3+3 * 4+4 * 5+5 * 6+6)
    cout << "product of sums: "
         << inner_product (coll.begin(), coll.end(),  // первый диапазон
                          coll.begin(),               // второй диапазон
                          1,                          // начальное значение
                          multiplies<int>(),          // внешняя операция
                          plus<int>());               // внутренняя операция
}
```

Численные алгоритмы

Преобразование относительных значений в абсолютные

OutputIterator **partial_sum**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg)
Для значений a1 a2 a3 … алгоритм вычисляет a1, a1+a2, a1+a2+a3, …

OutputIterator **partial_sum**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg,
BinaryFunc op)
Для значений a1 a2 a3 … алгоритм вычисляет a1, a1 op a2, a1 op a2 op a3, …

Преобразование абсолютных значений в относительные

OutputIterator **adjacent_difference**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg)
Для значений a1 a2 a3 … алгоритм вычисляет a1, a2-a1, a3-a2, …

OutputIterator **adjacent_difference**(InputIterator sourceBeg, InputIterator sourceEnd, OutputIterator destBeg,
BinaryFunc op)
Для значений a1 a2 a3 … алгоритм вычисляет a1, a2 op a1, a3 op a2, …

Численные алгоритмы

Преобразование относительных значений в абсолютные и наоборот

```cpp
#include <iostream>
#include <vector>
#include <iterator>
#include <algorithm>
#include <numeric>

using namespace std;

int main()
{
    vector<int> coll = {17,-3,22,13,13,-9};
    // преобразование в относительные значения
    adjacent_difference(coll.begin(), coll.end(),    // источник
                        coll.begin());               // получатель
    for (auto elem: coll)
        cout << elem << ' '; // 17 -20 25 -9 0 -22
    cout << endl;
    // преобразование в абсолютные значения
    partial_sum(coll.begin(), coll.end(),            // источник
                coll.begin());                       // получатель
    for (auto elem: coll)
        cout << elem << ' '; // 17 -3 22 13 13 -9
}
```