已排序区间算法：针对已排序区间执行的算法。

1. 查找元素binary\_search，如果是random-access iterator，为对数logn复杂度，否则为线性复杂度。

bool

binary\_search(Iterator beg, Iterator end, const T& value)

bool

binary\_search(Iterator beg, Iterator end,const T& value, BinaryPredicate op)

判断“已排序区间[beg, end)”是否包含“和value等值”的元素。

第二形式op(elem1, elem2)被视为排序准则。

例：程序stl\_test77

// binary\_search

*list*<int> coll;

InsertElements(coll, 1, 9);

PrintElements(coll, "coll: ");

// check existence of element with value 5

if (*binary\_search*(coll.*cbegin*(),coll.*cend*(), 5))

{

*cout* << "5 is present" << *endl*;

}

else {

*cout* << "5 is not present" << *endl*;

}

输出为：

coll: 1 2 3 4 5 6 7 8 9

5 is present

1. 检查数个元素是否存在，includes，线性复杂度

bool

includes(Iterator1 beg, Iterator1 end,

Iterator2 searchBeg, Iterator2 searchEnd)

bool

includes(Iterator1 beg, Iterator1 end,

Iterator2 searchBeg, Iterator2 searchEnd,

BinaryPredicate op)

判断已排序区间[beg, end)是否包含另一个已排序区间[searchBeg, searchEnd)的全部元素。注意：只要求[searchBeg, searchEnd)的每一个元素都在[beg, end)区间。

例：程序stl\_test77

// includes

*list*<int> coll1;

*vector*<int> search;

InsertElements(coll1, 1, 9);

PrintElements(coll1, "coll1: ");

search.*push\_back*(3);

search.*push\_back*(4);

search.*push\_back*(7);

PrintElements(search, "search: ");

if (*includes*(coll1.*cbegin*(), coll1.*cend*(),

search.*cbegin*(), search.*cend*()))

{

*cout* << "all elements of search are also in coll1" << *endl*;

}

else {

*cout* << "not all elements of search are also in coll1" << *endl*;

}

输出为：

coll1: 1 2 3 4 5 6 7 8 9

search: 3 4 7

all elements of search are also in coll1

1. 查找第一个或最后一个可能位置， 如果搭配random-access iterator是对数复杂度，否则是线性复杂度。

Iterator

lower\_bound(Iterator beg, Iterator end, const T& value)

Iterator

lower\_bound(Iterator beg, Iterator end, const T& value,

BinaryPredicate op)

返回第一个“大于等于value”的元素位置。这是可插入“元素值为value”且“不破坏区间[beg, end)之已排序性”的第一个位置。

如果不存在“其值为value”的元素，返回end。

Iterator

upper\_bound(Iterator beg, Iterator end, const T& value)

Iterator

upper\_bound(Iterator beg, Iterator end, const T& value,

BinaryPredicate op)

返回第一个“大于value”的元素位置，这是可插入“元素值为value”且“不破坏区间[beg, end)之已排序性”的最后一个位置。

如果不存在“其值为value”的元素，返回end

例：程序stl\_test77

// lower\_bound, upper\_bound

*list*<int> coll2;

InsertElements(coll2, 1, 9);

InsertElements(coll2, 1, 9);

coll2.*sort*();

PrintElements(coll2, "coll2: ");

// print first and last position 5 could get inserted

auto first\_pos = *lower\_bound*(coll2.*cbegin*(), coll2.*cend*(), 5);

auto last\_pos = *upper\_bound*(coll2.*cbegin*(), coll2.*cend*(), 5);

*cout* << "5 could get position "

<< *distance*(coll2.*cbegin*(), first\_pos) + 1

<< " up to "

<< *distance*(coll2.*cbegin*(), last\_pos) + 1

<< "without breaking the sorting" << *endl*;

// insert 3 at the first possible position

// without breaking the sorting

coll2.*insert*(*lower\_bound*(coll2.*begin*(), coll2.*end*(), 3), 3);

// insert 7 at the last possible position

// without breaking the sorting

coll2.*insert*(*upper\_bound*(coll2.*begin*(), coll2.*end*(), 7), 7);

PrintElements(coll2, "coll2: ");

输出为：

coll2: 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9

5 could get position 9 up to 11without breaking the sorting

coll2: 1 1 2 2 3 3 3 4 4 5 5 6 6 7 7 7 8 8 9 9

1. 两个已排序集合的总和，merge，线性复杂度

Iterator3

merge(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg)

Iterator3

merge(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg, BinaryPredicate op)

两者都将源区间[source1Beg, source1End)和[source2Beg, source2End)内的元素合并，使得“以destBeg起始之目标区间”内含两个源区间的所有元素。

目标区间的所有元素都将处于排序状态下。

源区间不会有任何变化。

返回目标区间内“最后一个被复制元素”的下一个位置

源区间和目标区间不能重叠。

例：程序stl\_test78

// merge

*list*<int> coll1;

*set*<int> coll2;

InsertElements(coll1, 1, 6);

InsertElements(coll2, 3, 8);

PrintElements(coll1, "coll1: ");

PrintElements(coll2, "coll2: ");

*cout* << "merged: ";

*merge*(coll1.*cbegin*(), coll1.*cend*(),

coll2.*cbegin*(), coll2.*cend*(),

*ostream\_iterator*<int>(*cout*, " "));

输出为：

coll1: 1 2 3 4 5 6

coll2: 3 4 5 6 7 8

merged: 1 2 3 3 4 4 5 5 6 6 7 8

1. 两个已排序集合的并集，set\_union，线性复杂度

Iterator3

set\_union(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg)

Iterator3

set\_union(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg, BinaryPredicate op)

两者都将源区间[source1Beg, source1End)和[source2Beg, source2End)内的元素合并，使得“以destBeg起始之目标区间”，后者内含要么来自第一源区间，要么来自第二源区间，抑或同时来自两者。

目标区间内的所有元素都处于排序状态。

同时出现于两个源区间内的元素，在并集区间只会出现一次。注意：如果原来的某个源区间内原本就存在重复元素，则目标区间内也会存在重复元素--重复的次数是两个源区间内的重复次数的较大值。

返回目标区间内“最后一个被复制元素”的下一位置。

源区间和目标区间不能重叠。

例：程序stl\_test78

// set\_union

*vector*<int> c1 = {1, 2, 2, 4, 6, 7, 7, 9};

*deque*<int> c2 = {2, 2, 2, 3, 6, 6, 8, 9};

PrintElements(c1, "c1: ");

PrintElements(c2, "c2: ");

*cout* << "set\_union(): ";

*set\_union*(c1.*cbegin*(), c1.*cend*(),

c2.*cbegin*(), c2.*cend*(),

*ostream\_iterator*<int>(*cout*, " "));

输出为：

c1: 1 2 2 4 6 7 7 9

c2: 2 2 2 3 6 6 8 9

set\_union(): 1 2 2 2 3 4 6 6 7 7 8 9

1. 两个已排序集合的交集，set\_intersection，线性复杂度

Iterator3

set\_intersection(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg)

Iterator3

set\_intersection(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg, BinaryPredicate op)

两者都将源区间[source1Beg, source1End)和[source2Beg, source2End)内的元素合并，使得“以destBeg起始之目标区间”，后者内含的元素不但存在于第一源区间，也存在于第二源区间。

目标区间内的所有元素都处于排序状态。

注意：如果某个源区间内存在重复元素，则目标区间也会有重复元素--重复次数是两个源区间内的重复次数的较小值。

源区间和目标区间不能重叠。

例：程序stl\_test78

*vector*<int> c1 = {1, 2, 2, 4, 6, 7, 7, 9};

*deque*<int> c2 = {2, 2, 2, 3, 6, 6, 8, 9};

// set\_intersection

PrintElements(c1, "c1: ");

PrintElements(c2, "c2: ");

*cout* << "set\_intersection(): ";

*set\_intersection*(c1.*cbegin*(), c1.*cend*(),

c2.*cbegin*(), c2.*cend*(),

*ostream\_iterator*<int>(*cout*, " "));

输出为：

c1: 1 2 2 4 6 7 7 9

c2: 2 2 2 3 6 6 8 9

set\_intersection(): 2 2 6 9

1. 两个已排序集合的差集，set\_difference，线性复杂度

Iterator3

set\_difference(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg)

Iterator3

set\_difference(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg, BinaryPredicate op)

两者都将源区间[source1Beg, source1End)和[source2Beg, source2End)内的元素合并，使得“以destBeg起始之目标区间”，后者内含的元素只存在于第一源区间，不存在于第二源区间。

目标区间内的所有元素都处于排序状态。

注意：如果某个源区间内存在重复元素，则目标区间也会有重复元素--重复次数是第一源区间内的重复次数减去第二源区间内相应的重复次数，如果第二源区间的重复次数大于第一源区间的重复次数，则目标区间对应的重复次数为0.。

源区间和目标区间不能重叠。

例：程序stl\_test78

// set\_difference

PrintElements(c1, "c1: ");

PrintElements(c2, "c2: ");

*cout* << "set\_difference(): ";

*set\_difference*(c1.*cbegin*(), c1.*cend*(),

c2.*cbegin*(), c2.*cend*(),

*ostream\_iterator*<int>(*cout*, " "));

输出为：

c1: 1 2 2 4 6 7 7 9

c2: 2 2 2 3 6 6 8 9

set\_difference(): 1 4 7 7

1. set\_symmetric\_difference

Iterator3

set\_symmetric\_difference(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg)

Iterator3

set\_symmetric\_difference(Iterator1 source1Beg, Iterator1 source1End,

Iterator2 source2Beg, Iterator2 source2End,

Iterator3 destBeg, BinaryPredicate op)

两者都将源区间[source1Beg, source1End)和[source2Beg, source2End)内的元素合并，使得“以destBeg起始之目标区间”，后者内含的元素或存在于第一源区间，或存在于第二源区间，但不同时存在于两个源区间。

目标区间内的所有元素都处于排序状态。

注意：如果某个源区间内存在重复元素，则目标区间也会有重复元素--重复次数是两个源区间内的对应重复元素的个数差值。

例：程序stl\_test78

// set\_symmetric\_difference

PrintElements(c1, "c1: ");

PrintElements(c2, "c2: ");

*cout* << "set\_symmetric\_difference(): ";

*set\_symmetric\_difference*(c1.*cbegin*(), c1.*cend*(),

c2.*cbegin*(), c2.*cend*(),

*ostream\_iterator*<int>(*cout*, " "));

输出为：

c1: 1 2 2 4 6 7 7 9

c2: 2 2 2 3 6 6 8 9

set\_symmetric\_difference(): 1 2 3 4 6 7 7 8