



STL

-CHAPTER4-

SOULSEEK



목차

1. 원소를 수정하지 않는 알고리즘



The background is a dark blue gradient. In the corners, there are decorative white line art elements resembling circuit boards or neural network connections. These elements consist of thin lines that branch out and terminate in small circles.

원소를 수정하지 않는 알고리즘

1. 원소를 수정하지 않는 알고리즘

- 원소의 순서나 원소의 값을 변경하지 않고 원소를 읽기만 하는 알고리즘.

알고리즘	설명(설명에 사용되는 p 는 구간 $[b, e)$ 의 반복자)
p=adjacent_find(b,e)	p 는 구간 $[b,e)$ 의 원소 중 $*p==*(p+1)$ 인 첫 원소를 가리키는 반복자.
p=adjacent_find(b,e,f)	p 는 구간 $[b,e)$ 의 원소 중 $f(*p, *(p+1))$ 이 참인 첫 원소를 가리키는 반복자.
n=count(b,e,x)	n 은 구간 $[b,e)$ 의 원소 중 x 원소의 개수
n=count(b,e,f)	n 은 구간 $[b,e)$ 의 원소 중 $f(*p)$ 가 참인 원소의 개수
equal(b,e,b2)	$[b,e)$ 와 $[b2,b2+(e-b))$ 의 모든 원소가 같은($==$)가?
equal(b,e,b2,f)	$[b,e)$ 와 $[b2,b2+(e-b))$ 의 모든 원소가 $f(*p,*q)$ 가 참인가?
p=find(b,e,x)	p 는 구간 $[b,e)$ 에서 x 와 같은 첫 원소의 반복자
p=find_end(b,e,b2,e2)	p 는 구간 $[b,e)$ 의 순차열 중 구간 $[b2,e2)$ 의 순차열과 일치하는 순차열 첫 원소의 반복자. 단, $[b2,e2)$ 와 일치하는 순차열이 여러 개라면 마지막 순차열 첫 원소의 반복자.
p=find_end(b,e,b2,e2,f)	p 는 구간 $[b,e)$ 의 순차열 중 구간 $[b2,e2)$ 의 순차열과 일치하는 순차열 첫 원소의 반복자. 단, $[b2,e2)$ 와 일치하는 순차열이 여러 개라면 마지막 순차열 첫 원소의 반복자. 이때 비교는 f 를 사용
find_first_of(b,e,b2,e2)	p 는 구간 $[b,e)$ 에서 구간 $[b2,e2)$ 의 원소 중 같은 원소가 발견된 첫 원소의 반복자
find_first_of(b,e,b2,e2,f)	p 는 구간 $[b,e)$ 에서 구간 $[b2,e2)$ 의 원소 중 같은 원소가 발견된 첫 원소의 반복자. 이때 비교는 f 를 사용

1. 원소를 수정하지 않는 알고리즘

알고리즘	설명(설명에 사용되는 p 는 구간 $[b, e)$ 의 반복자)
P=find_if(b,e,f)	P 는 구간 $[b,e)$ 에 f(*p) 가 참인 첫 원소를 가리키는 반복자
F=for_each(b,e,f)	구간 $[b,e)$ 의 모든 원소에 f(*p) 동작을 적용한다. F 를 다시 되찾는다.
Lexicographical_compare(b,e,b2,e2)	구간 $[b,e)$ 의 순차열이 구간 $[b2,e2)$ 의 순차열 보다 작다면 true , 아니면 false 를 반환한다. 이때 작음은 사전순이다.
Lexicographical_compare(b,e,b2,e2,f)	구간 $[b,e)$ 의 순차열이 구간 $[b2,e2)$ 의 순차열 보다 작다면 true , 아니면 false 를 반환한다. 이때 작음은 $[b,e)$ 의 반복자 p 와 $[b2,e2)$ 의 반복자 q 에 대해 f(*p, *q) 가 참이다
K=max(a,b)	K 는 a 와 b 중 더 큰 것
K=max(a,b,f)	K 는 a 와 b 중 더 큰 것. 이때 큰 것은 f(a, b) 를 사용
P=max_element(b,e)	P 는 구간 $[b,e)$ 에서 가장 큰 원소의 반복자.
P=max_element(b,e,f)	P 는 구간 $[b,e)$ 에서 가장 큰 원소의 반복자. 이때 비교는 f 를 사용
K=min(a,b)	K 는 a 와 b 중 더 작은 것
K=min(a,b,f)	K 는 a 와 b 중 더 작은 것. 이때 더 작은 것은 f(a, b) 를 사용
P=min_element(b,e)	P 는 구간 $[b, e)$ 에서 가장 작은 원소의 반복자
P=min_element(b,e,f)	P 는 구간 $[b, e)$ 에서 가장 작은 원소의 반복자. 이때 비교는 f 를 사용
Pair(p, q)=mismatch(b, e, b2)	(p, q) 는 구간 $[b,e)$ 와 $[b2+(e-b))$ 에서 !(*p == *q) 인 첫 원소를 가리키는 반복자의 쌍

1. 원소를 수정하지 않는 알고리즘

알고리즘	설명(설명에 사용되는 p 는 구간 $[b, e)$ 의 반복자)
Pair(p, q) = mismatch(b, e, b2, f)	(p, q) 는 구간 $[b, e)$ 와 $[b2, b2 + (e - b))$ 에서 if(*p, *q) 가 참인 첫 원소를 가리키는 반복자의 쌍
P = search(b, e, b2, e2)	P 는 구간 $[b, e)$ 의 순차열 중 구간 $[b2, e2)$ 의 순차열과 일치하는 순차열 첫 원소의 반복자(find_end() 와 비슷하나 find_end() 는 일치하는 순차열의 마지막 순차열의 반복자)
P = search(b, e, b2, e2, f)	P 는 구간 $[b, e)$ 의 순차열 중 구간 $[b2, e2)$ 의 순차열과 일치하는 순차열 첫 원소의 반복자. 이때 비교는 f 를 사용
P = search_n(b, e, n, x)	P 는 구간 $[b, e)$ 의 원소 중 x 값이 n 개 연속한 첫 원소의 반복자
P = search_n(b, e, n, x, f)	P 는 구간 $[b, e)$ 의 원소 중 f(*p, x) 가 참인 값이 n 개 연속한 첫 원소의 반복자

1. 원소를 수정하지 않는 알고리즘

adjacent_find() 알고리즘 예제..

```
void main()
{
```

```
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(30);
    v.push_back(40);
    v.push_back(40);
    v.push_back(50);
```

```
    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;
```

```
    vector<int>::iterator iter;
    //구간 [v.begin(), v.end())에서 현재 원소와 다음 원소가 같아지는 첫 번째 반복자를 반환
    iter = adjacent_find(v.begin(), v.end());
```

```
    if (iter != v.end()) //같은 원소가 없다면 구간의 끝 반복자 반환
        cout << *iter << endl;
```

```
    for (; iter != v.end(); ++iter)
        cout << *iter << " ";
    cout << endl;
```

```
}
```

v.begin()



10

20

30

30

40

40

50

N

v.end()



iter

1. 원소를 수정하지 않는 알고리즘

adjacent_find() 알고리즘이 원소를 찾지 못했을 때..

```
void main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(40);
    v.push_back(50);

    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;

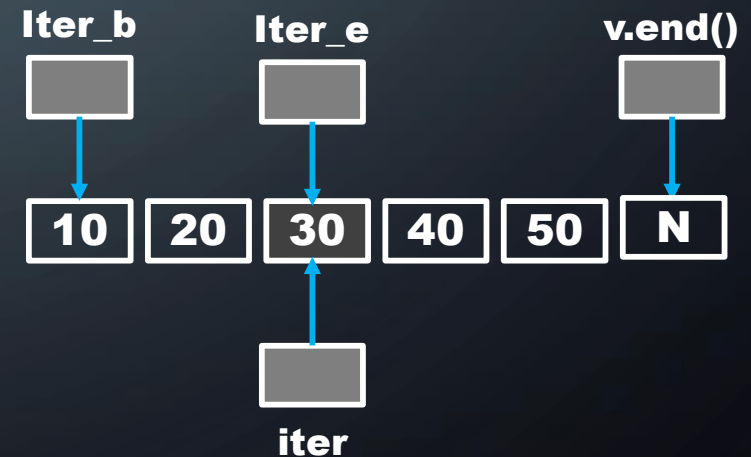
    vector<int>::iterator iter;
    vector<int>::iterator iter_b = v.begin();
    vector<int>::iterator iter_e = v.begin() + 2;
    //구간 [iter_b, iter_e)의 순차열은 10, 20이므로
    //찾는 원소가 없어 iter는 구간의 끝인 iter_e입니다.
    iter = adjacent_find(iter_b, iter_e);

    if (iter != iter_e) //찾은 원소가 없는지 확인 <<맞는 표현>>
        cout << *iter << endl;

    cout << "======" << endl;
    if (iter != v.end()) //찾은 원소가 없는지 확인 <<틀린 표현>>
        cout << *iter << endl;
}
```

```
iter_b = v.begin();
iter_e = v.end();
iter = adjacent_find(iter_b, iter_e);
```

```
//모두 맞는 확인
if (iter != iter_e)
    cout << *iter << endl;
if (iter != v.end())
    cout << *iter << endl;
```



1. 원소를 수정하지 않는 알고리즘

adjacent_find() 조건자 사용 예제..

```
void main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(50);
    v.push_back(90);

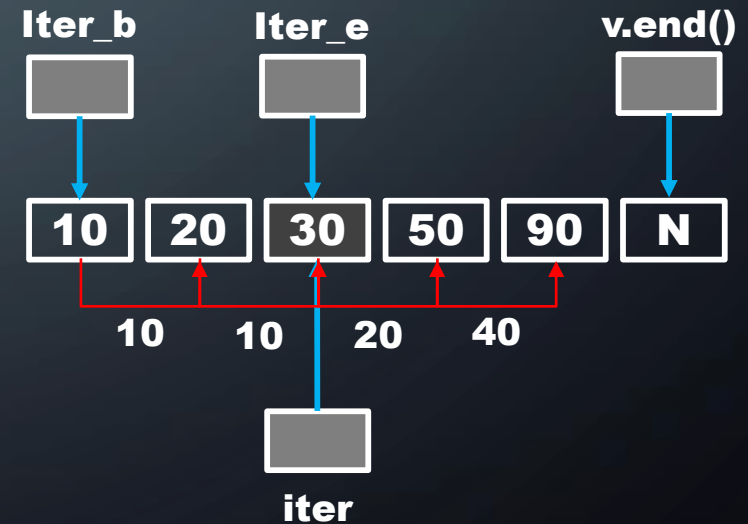
    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;

    vector<int>::iterator iter;
    iter = adjacent_find(v.begin(), v.end(), Pred);

    if (iter != v.end())
        cout << *iter << endl;

    for ( ; iter != v.end(); ++iter)
        cout << *iter << " ";
    cout << endl;
}
```

```
//이항 조건자(b = a+1)
bool Pred(int a, int b)
{
    return abs(b - a) > 10;
}
```



1. 원소를 수정하지 않는 알고리즘

count() 알고리즘 예제..

```
void main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(40);
    v.push_back(30);
    v.push_back(50);

    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";

    cout << endl;

    // 구간 [v.begin(), v.end())에서 30 원소의 개수를 반환
    int n = count(v.begin(), v.end(), 30);

    cout << "30의 개수: " << n << endl;
}
```

1. 원소를 수정하지 않는 알고리즘

count_if() 조건자 사용 예제..

```
bool Pred(int n)
{
    return 25 < n;
}
```

```
void main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(40);
    v.push_back(50);

    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;

    // 구간 [v.begin(), v.end())에서 25 보다 큰 원소의 개수를 반환
    int n = count_if(v.begin(), v.end(), Pred);
    cout << "25보다 큰 원소의 개수 :" << n << endl;
}
```

1. 원소를 수정하지 않는 알고리즘

equal() 알고리즘 조건자 사용 예제..

```
void main()
```

```
{
```

```
    vector<int> v1;  
    v1.push_back(10);  
    v1.push_back(20);  
    v1.push_back(30);
```

```
}
```

```
    vector<int> v2;  
    v2.push_back(10);  
    v2.push_back(20);  
    v2.push_back(30);  
    v2.push_back(40);  
    v2.push_back(50);
```

```
    cout << "v1 : ";  
    for (vector<int>::size_type i = 0; i < v1.size(); ++i)  
        cout << v1[i] << " ";  
    cout << endl;
```

```
    cout << "v2 : ";
```

```
    for (vector<int>::size_type i = 0; i < v2.size(); ++i)  
        cout << v2[i] << " ";
```

```
    cout << endl;
```

```
//구간 [v1.begin(), v1.end)와 구간 [v2.begin(),v2.begin()+3)을 비교  
if (equal(v1.begin(), v1.end(), v2.begin()))  
    cout << "두 순차열은 같습니다." << endl;
```

v1.begin, v1.end의 구간에 원소가 **3개**이므로
v2.begin에서 **v2.begin + 2**만큼의 원소와 비교를 한다.

1. 원소를 수정하지 않는 알고리즘

equal() 알고리즘 조건자 사용 예제..

```
bool Pred(int left, int right)
{
    return abs(right - left) < 5;
}
```

```
void main()
{
```

```
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(21);
    v1.push_back(30);
```

```
    vector<int> v2;
    v2.push_back(10);
    v2.push_back(20);
    v2.push_back(33);
```

```
    cout << "v1 : ";
    for (vector<int>::size_type i = 0; i < v1.size(); ++i)
        cout << v1[i] << " ";
    cout << endl;
```

```
    cout << "v2 : ";
    for (vector<int>::size_type i = 0; i < v2.size(); ++i)
        cout << v2[i] << " ";
    cout << endl;
```

```
    //구간 [v1.begin(), v1.end)와 구간 [v2.begin(), v2.begin()+3)을
    //비교 합니다.
```

```
    if (equal(v1.begin(), v1.end(), v2.begin(), Pred))
        cout << "두 순차열은 같습니다." << endl;
```

```
}
```

v1.begin, v1.end의 구간과 **v2.begin, v2.begin+3**을 비교해서
조건자에 맞는지 검사하고 모두 맞으면 참으로 반환한다.

1. 원소를 수정하지 않는 알고리즘

find() 알고리즘

```
bool Pred(int n)
{
    return 35 < n;
}
```

```
void main()
{
    vector<int> v;
    v.push_back(10);
    v.push_back(20);
    v.push_back(30);
    v.push_back(40);
    v.push_back(50);
```

```
    for (vector<int>::size_type i = 0; i < v.size(); ++i)
        cout << v[i] << " ";
    cout << endl;
```

```
vector<int>::iterator iter;
iter = find(v.begin(), v.end(), 20);
```

```
if (iter != v.end())
    cout << *iter << "을 찾다!" << endl;
```

```
// 구간 [v.begin(), v.end())에서 25 보다 큰 원소의 개수를 반환
iter = find_if(v.begin(), v.end(), Pred);
```

```
if (iter != v.end())
    cout << "순차열에서 35보다 큰 첫 번째 원소: " << *iter << endl;
```

1. 원소를 수정하지 않는 알고리즘

find_end() 알고리즘을 이용해 컨테이너를 판단하는 예제..

```
void main()
```

```
{
```

```
    vector<int> v1;
```

```
    v1.push_back(10);
```

```
    v1.push_back(20);
```

```
    v1.push_back(30);
```

```
    v1.push_back(40);
```

```
    v1.push_back(50);
```

```
    v1.push_back(60);
```

```
    v1.push_back(70);
```

```
    v1.push_back(30);
```

```
    v1.push_back(40);
```

```
    v1.push_back(50);
```

```
    vector<int> v2;
```

```
    v2.push_back(30);
```

```
    v2.push_back(40);
```

```
    v2.push_back(50);
```

```
    vector<int>::iterator iter;
```

```
    iter = find_end(v1.begin(), v1.end(), v2.begin(), v2.end());
```

```
    if (iter != v1.end())
```

```
    {
```

```
        // 일치하는 마지막 순차열의 첫 원소의 반복자 iter
```

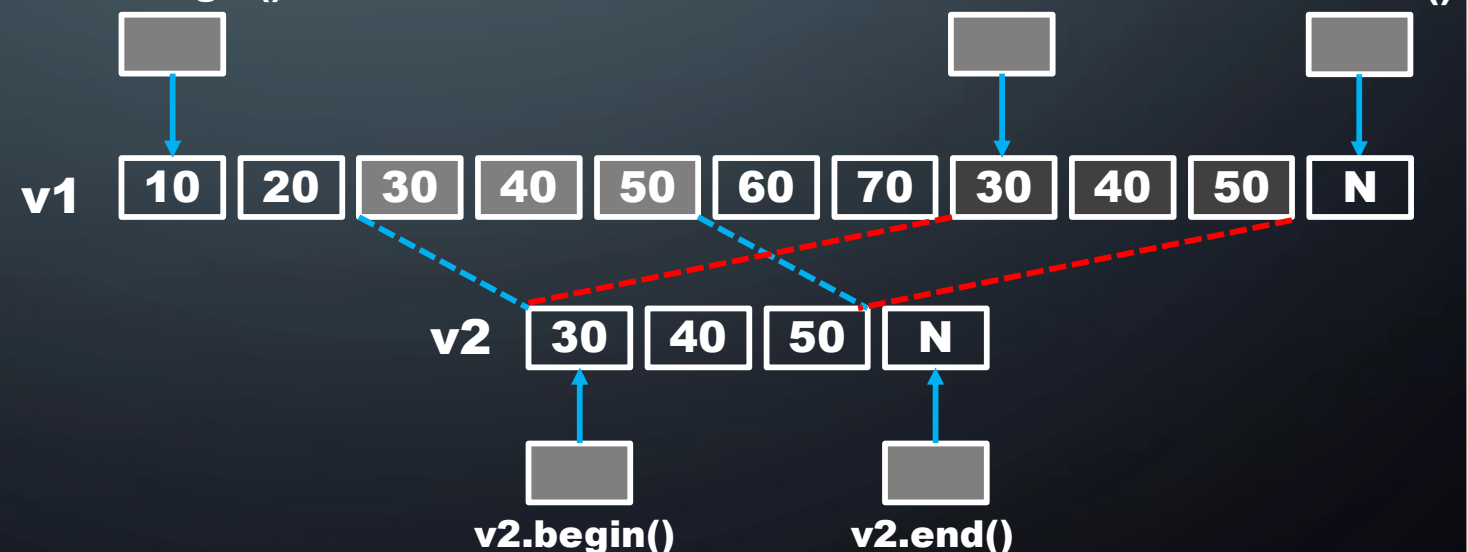
```
        cout << "iter : " << *iter << endl;
```

```
        cout << "iter-1 : " << *(iter - 1) << endl;
```

```
        cout << "iter+1 : " << *(iter + 1) << endl;;
```

```
    }
```

```
    }
```



1. 원소를 수정하지 않는 알고리즘

find_end 조건자 사용 예제..

```
bool Pred(int left, int right)
{
    return left <= right;
}
```

```
void main()
{
```

```
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(15);
    v1.push_back(20);
    v1.push_back(40);
    v1.push_back(50);
    v1.push_back(60);
    v1.push_back(10);
    v1.push_back(11);
    v1.push_back(12);
    v1.push_back(80);
```

```
    vector<int> v2;
    v2.push_back(10);
    v2.push_back(15);
    v2.push_back(25);
```

```
    vector<int>::iterator iter;
```

```
    iter = find_end(v1.begin(), v1.end(), v2.begin(), v2.end(), Pred);
```

```
    if (iter != v1.end())
```

```
    {
```

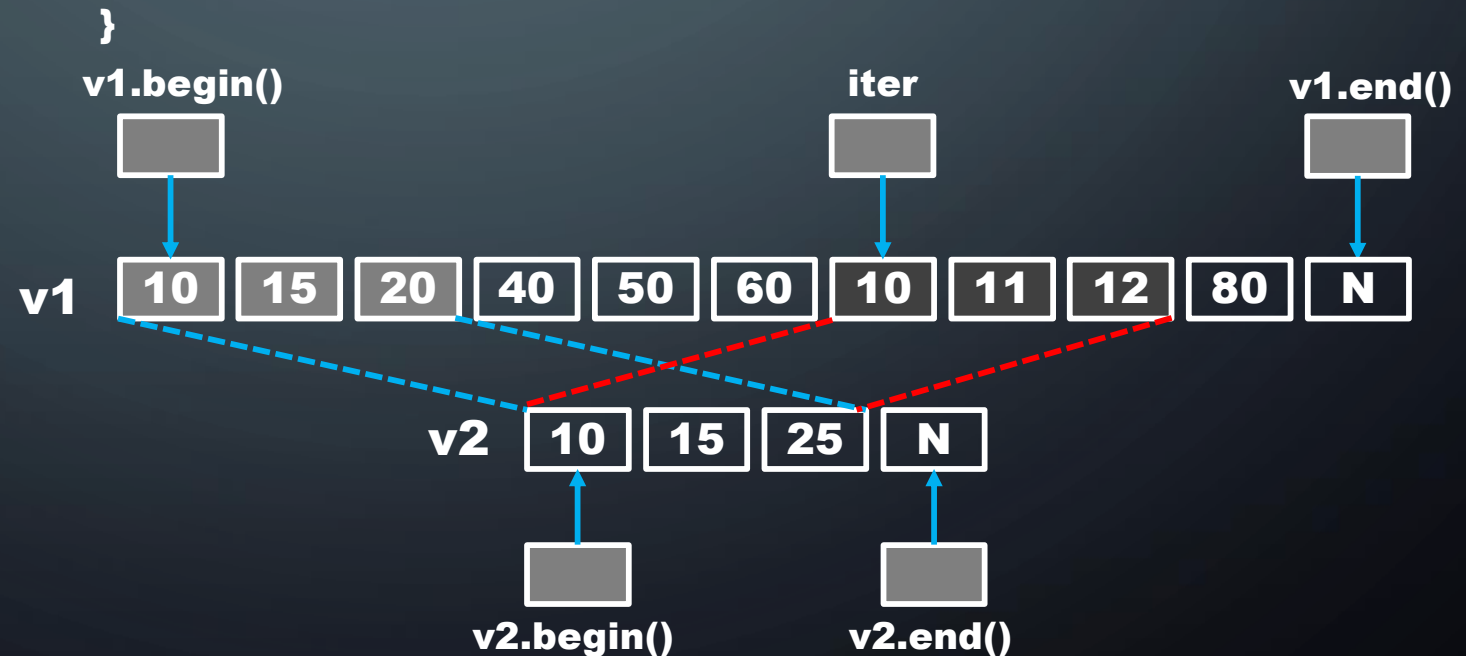
```
        // 일치하는 마지막 순차열의 첫 원소의 반복자 iter
```

```
        cout << "iter : " << *iter << endl;
```

```
        cout << "iter-1 : " << *(iter - 1) << endl;
```

```
        cout << "iter+1 : " << *(iter + 1) << endl;;
```

```
    }
```



1. 원소를 수정하지 않는 알고리즘

find_first_of() 알고리즘 사용 예제..

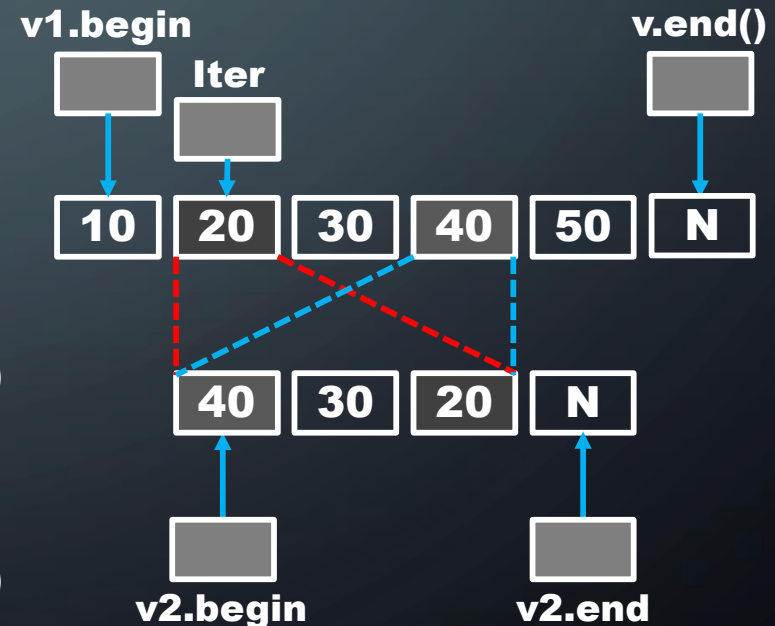
```
void main()
{
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    vector<int> v2;
    v2.push_back(40);
    v2.push_back(80);
    v2.push_back(20);

    cout << "v1 : ";
    for (vector<int>::size_type i = 0; i < v1.size(); ++i)
        cout << v1[i] << " ";
    cout << endl;
    cout << "v2 : ";
    for (vector<int>::size_type i = 0; i < v2.size(); ++i)
        cout << v2[i] << " ";
    cout << endl;

    vector<int>::iterator iter;
    iter = find_first_of(v1.begin(), v1.end(), v2.begin(), v2.end());

    if (iter != v1.end())
        cout << "iter : " << *iter << endl;
}
```



1. 원소를 수정하지 않는 알고리즘

find_first_of() 알고리즘 사용 예제..

```
bool Pred(int left, int right)
{
    return left > right;
}

void main()
{
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    vector<int> v2;
    v2.push_back(40);
    v2.push_back(80);
    v2.push_back(20);

    cout << "v1 : ";
    for (vector<int>::size_type i = 0; i < v1.size(); ++i)
        cout << v1[i] << " ";
    cout << endl;

    cout << "v2 : ";
    for (vector<int>::size_type i = 0; i < v2.size(); ++i)
        cout << v2[i] << " ";
    cout << endl;

    vector<int>::iterator iter;
    iter = find_first_of(v1.begin(), v1.end(), v2.begin(), v2.end(), Pred);

    if (iter != v1.end())
        cout << "iter : " << *iter << endl;
}
```

1. 원소를 수정하지 않는 알고리즘

for_each() 알고리즘

```
void Print(int n)
```

```
{
```

```
    cout << n << " ";
```

```
}
```

```
void main()
```

```
{
```

```
    vector<int> v;
```

```
    v.push_back(10);
```

```
    v.push_back(20);
```

```
    v.push_back(30);
```

```
    v.push_back(40);
```

```
    v.push_back(50);
```

```
    for_each(v.begin(), v.begin() + 2, Print);
```

```
    cout << endl;
```

```
    for_each(v.begin(), v.begin() + 4, Print);
```

```
    cout << endl;
```

```
    //[v.begin(), v.end()) 구간의 원소 출력
```

```
    for_each(v.begin(), v.end(), Print);
```

```
    cout << endl;
```

```
}
```

1. 원소를 수정하지 않는 알고리즘

for_each() 알고리즘 함수자 사용 예제..

```
class PrintFuncor
```

```
{
```

```
    char fmt;
```

```
public:
```

```
    explicit PrintFuncor(char c = ' ') : fmt(c) { }
```

```
    void operator()(int n) const
```

```
    {
```

```
        cout << n << fmt;
```

```
    }
```

```
};
```

```
void main()
```

```
{
```

```
    vector<int> v;
```

```
    v.push_back(10);
```

```
    v.push_back(20);
```

```
    v.push_back(30);
```

```
    v.push_back(40);
```

```
    v.push_back(50);
```

```
    for_each(v.begin(), v.end(), PrintFuncor()); // 원소의 구분을 ' '로
```

```
    cout << endl;
```

```
    for_each(v.begin(), v.end(), PrintFuncor(',')); // 원소의 구분을 ','로
```

```
    cout << endl;
```

```
    for_each(v.begin(), v.end(), PrintFuncor('\n')); // 원소의 구분을 '\n'로
```

```
    cout << endl;
```

```
}
```

1. 원소를 수정하지 않는 알고리즘

lexicographical_compare() 알고리즘 사용 예제..

```
void main()
```

```
{
```

```
    vector<int> v1;  
    v1.push_back(10);  
    v1.push_back(20);  
    v1.push_back(30);
```

```
    vector<int> v2;  
    v2.push_back(10);  
    v2.push_back(20);  
    v2.push_back(50);
```

```
    vector<int> v3;  
    v3.push_back(10);  
    v3.push_back(20);  
    v3.push_back(30);
```

```
    if (lexicographical_compare(v1.begin(), v1.end(), v2.begin(), v2.end()))  
        cout << "v1 < v2" << endl;
```

```
    else  
        cout << "v1 >= v2" << endl;
```

```
    if (lexicographical_compare(v1.begin(), v1.end(), v3.begin(), v3.end()))  
        cout << "v1 < v3" << endl;
```

```
    else  
        cout << "v1 >= v3" << endl;
```

```
}
```

v3		v1		v2
10	=	10	=	10
20	=	20	=	20
30	=	30	<	50
N		N		N
v1 < v3		v1 < v2		
false		true		

1. 원소를 수정하지 않는 알고리즘

lexicographical_compare() 알고리즘 조건자 사용 예제..

```
void main()
{
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);

    vector<int> v2;
    v2.push_back(10);
    v2.push_back(25);
    v2.push_back(30);

    if (lexicographical_compare(v1.begin(), v1.end(), v2.begin(), v2.end(), less<int>()))
        cout << "기준 less v1과 v2의 비교: true" << endl;
    else
        cout << "기준 less v1과 v2의 비교: false" << endl;

    if (lexicographical_compare(v1.begin(), v1.end(), v2.begin(), v2.end(), greater<int>()))
        cout << "기준 greater v1과 v2의 비교: true" << endl;
    else
        cout << "기준 greater v1과 v2의 비교: false" << endl;

    if (lexicographical_compare(v1.begin(), v1.end(), v2.begin(), v2.end(), Less<int>()))
        cout << "사용자 기준 Less v1과 v2의 비교: true" << endl;
    else
        cout << "사용자 기준 Less v1과 v2의 비교: false" << endl;
}
```

```
template< typename T>
struct Less
{
    bool operator()(const T& left, const T& right) const
    {
        return left < right;
    }
};
```


1. 원소를 수정하지 않는 알고리즘

max(), min() 알고리즘 사용 예제..

```
void main()
{
    int a = 10, b = 20;
    int r;

    r = max(a, b);
    cout << "max: " << r << endl;

    r = min(a, b);
    cout << "min: " << r << endl;

    Point pt1(5, 8), pt2(3, 9);
    Point pt3;

    pt3 = max(pt1, pt2, XCompare);
    cout << "x max: "; pt3.Print();

    pt3 = max(pt1, pt2, YCompare);
    cout << "y max: "; pt3.Print();
}
```

```
class Point
{
    int x, y;
public:
    explicit Point(int _x = 0, int _y = 0) :x(_x), y(_y) { }
    int GetX() const { return x; }
    int GetY() const { return y; }
    void Print() const { cout << '(' << x << ',' << y << ')'
        << endl; }
};

bool XCompare(const Point& left, const Point& right)
{
    return left.GetX() < right.GetX();
}

bool YCompare(const Point& left, const Point& right)
{
    return left.GetY() < right.GetY();
}
```

Point 클래스가 비교연산자를 지원하지 않기 때문에
전역 함수로 비교를 해서 **point**를 반환한다.

1. 원소를 수정하지 않는 알고리즘

max_element(), min_element() 알고리즘 사용 예제..

```
void main()
{
    vector<int> v;

    v.push_back(30);
    v.push_back(10);
    v.push_back(20);
    v.push_back(50);
    v.push_back(40);

    vector<int>::iterator iter;
    iter = max_element(v.begin(), v.end());
    cout << *iter << endl;

    iter = min_element(v.begin(), v.end());
    cout << *iter << endl;
}
```


1. 원소를 수정하지 않는 알고리즘

max_element() 조건자 사용 예제..

```
bool Compare(const Point& left, const Point& right)
```

```
{
    if (left.GetX() < right.GetX())
        return true;
    else if (left.GetX() > right.GetX())
        return false;
    else
        return left.GetY() < right.GetY();
}
```

```
void main()
```

```
{
    vector<Point> v;
    v.push_back(Point(3, 2));
    v.push_back(Point(2, 5));
    v.push_back(Point(1, 5));
    v.push_back(Point(3, 3));
    v.push_back(Point(3, 2));
}
```

```
vector<Point>::iterator iter;
```

```
iter = max_element(v.begin(), v.end(), Compare);
```

```
cout << "max: "; iter->Print(); //반복자가 가리키는 객체의 멤버는 -> 연산자로 접근
```

```
cout << "max: "; (*iter).Print(); //위와 같음
```

```
}
```

```
class Point
```

```
{
```

```
    int x, y;
```

```
public:
```

```
    explicit Point(int _x = 0, int _y = 0) :x(_x), y(_y) { }
```

```
    int GetX() const { return x; }
```

```
    int GetY() const { return y; }
```

```
    void Print() const
```

```
{
```

```
        cout << '(' << x << ',' << y << ')' << endl;
```

```
}
```

```
};
```

1. 원소를 수정하지 않는 알고리즘

mismatch() 알고리즘 사용 예제..

```
void main()
```

```
{
```

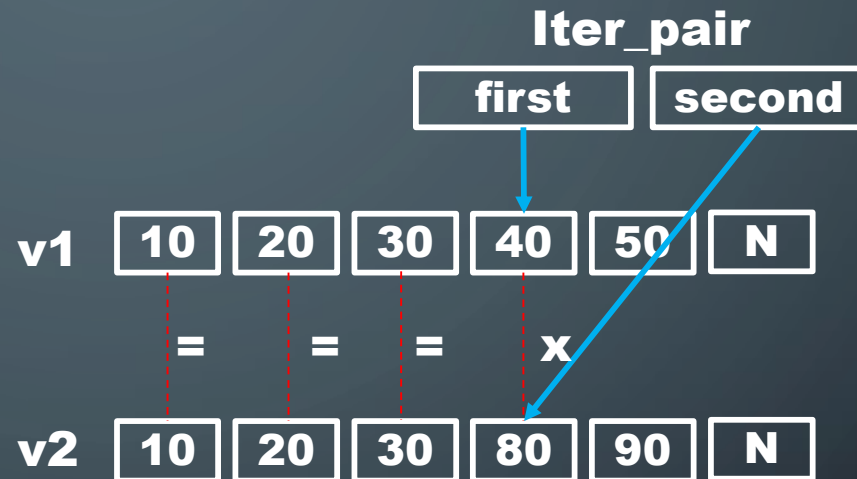
```
    vector<int> v1;  
    v1.push_back(10);  
    v1.push_back(20);  
    v1.push_back(30);  
    v1.push_back(40);  
    v1.push_back(50);
```

```
    vector<int> v2;  
    v2.push_back(10);  
    v2.push_back(20);  
    v2.push_back(30);  
    v2.push_back(80);  
    v2.push_back(90);
```

```
    pair<vector<int>::iterator, vector<int>::iterator> iter_pair;  
    iter_pair = mismatch(v1.begin(), v1.end(), v2.begin());
```

```
    cout << "v1: " << *iter_pair.first << ", " << "v2: " << *iter_pair.second << endl;
```

```
}
```



1. 원소를 수정하지 않는 알고리즘

mismatch() 조건자 사용 예제..

```
void main()
{
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);

    vector<int> v2;
    v2.push_back(11);
    v2.push_back(25);
    v2.push_back(33);
    v2.push_back(46);
    v2.push_back(50);

    pair<vector<int>::iterator, vector<int>::iterator> iter_pair;
    iter_pair = mismatch(v1.begin(), v1.end(), v2.begin(), Pred);

    cout << "v1: " << *iter_pair.first << ", " << "v2: " << *iter_pair.second << endl;
}
```

```
bool Pred(int left, int right)
{
    return abs(right - left) <= 5;
}
```

1. 원소를 수정하지 않는 알고리즘

search() 알고리즘 사용 예제..

```
void main()
{
```

```
    vector<int> v1;
    v1.push_back(10);
    v1.push_back(20);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);
    v1.push_back(60);
    v1.push_back(70);
    v1.push_back(30);
    v1.push_back(40);
    v1.push_back(50);
```

```
    vector<int> v2;
    v2.push_back(30);
    v2.push_back(40);
    v2.push_back(50);
```

```
    vector<int>::iterator iter;
    iter = search(v1.begin(), v1.end(), v2.begin(), v2.end());
```

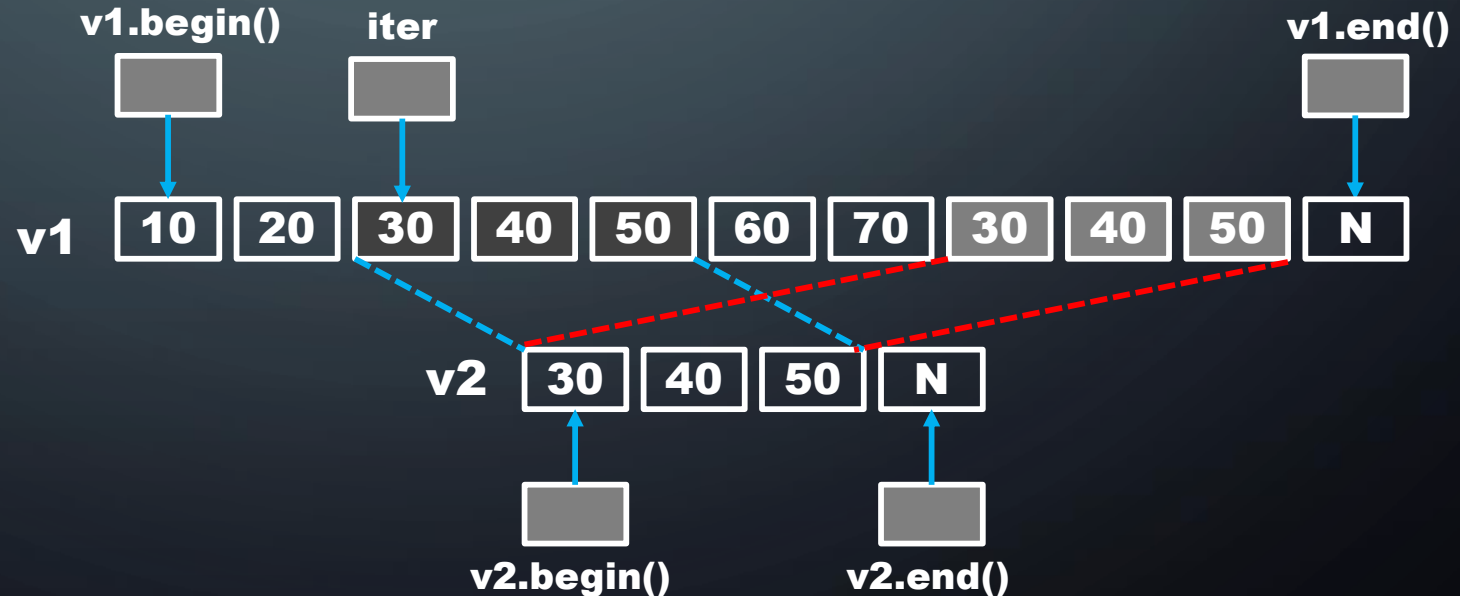
```
    if (iter != v1.end())
```

```
    {
```

```
        // 일치하는 첫 번째 순차열의 첫 원소의 반복자 iter
        cout << "iter : " << *iter << endl;
        cout << "iter-1 : " << *(iter - 1) << endl;
        cout << "iter+1 : " << *(iter + 1) << endl;
```

```
    }
```

```
}
```



1. 원소를 수정하지 않는 알고리즘

search_n() 알고리즘 사용 예제..

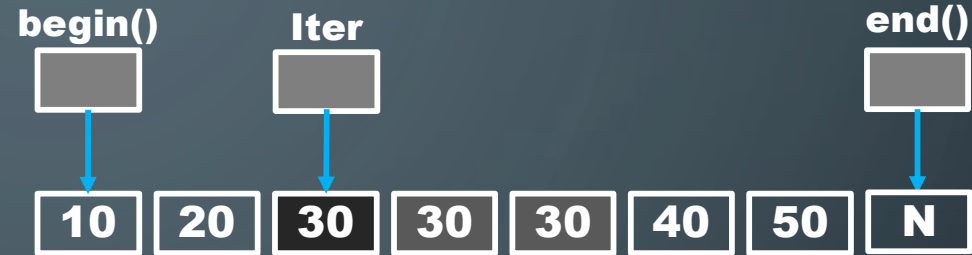
```
void main()
```

```
{
```

```
    vector<int> v;  
    v.push_back(10);  
    v.push_back(20);  
    v.push_back(30);  
    v.push_back(30);  
    v.push_back(30);  
    v.push_back(40);  
    v.push_back(50);
```

```
    vector<int>::iterator iter;  
    iter = search_n(v.begin(), v.end(), 3, 30);
```

```
    if (iter != v.end())  
    {  
        cout << "iter : " << *iter << endl;  
        cout << "iter-1 : " << *(iter - 1) << endl;  
        cout << "iter+1 : " << *(iter + 1) << endl;;  
    }  
}
```



1. 원소를 수정하지 않는 알고리즘

search_n() 조건자 사용 예제..

```
void main()
```

```
{
```

```
    vector<int> v;  
    v.push_back(10);  
    v.push_back(20);  
    v.push_back(32);  
    v.push_back(28);  
    v.push_back(33);  
    v.push_back(40);  
    v.push_back(50);
```

```
    vector<int>::iterator iter;
```

```
    iter = search_n(v.begin(), v.end(), 3, 30, Pred);
```

```
    if (iter != v.end())
```

```
    {
```

```
        cout << "iter : " << *iter << endl;
```

```
        cout << "iter-1 : " << *(iter - 1) << endl;
```

```
        cout << "iter+1 : " << *(iter + 1) << endl;;
```

```
    }
```

```
}
```

```
bool Pred(int left, int right)
```

```
{
```

```
    return abs(right - left) <= 5;
```

```
}
```

begin()



Iter



end()



30

30과의 차이가 5보다 작은 원소가 3번 이상
연속한 첫 원소의 반복자를 반환한다.