STL <ALGORITHM>

컴퓨터학부 20132303 권성광

목차

lower_bound, upper_bound

- 이미 정렬된 데이터에서 정렬 상태를 유지하고 원소를 삽입 할 수 있는 위치를 반환하는 함수
- 이분 탐색을 통해 위치 탐색
- 중복된 원소가 있어도 사용 가능
- 여러번 불러도 항상 같은 값을 반환

Defined in header `<algorithm>`

```
template< class ForwardIt, class T >
ForwardIt lower_bound( ForwardIt first, ForwardIt last, const T& value );
```

```
template< class ForwardIt, class T >
ForwardIt upper_bound( ForwardIt first, ForwardIt last, const T& value );
```

Parameters

first, last	검사를 위한 범위의 시작과 끝을 지정
value	지정한 범위 안 원소들과 비교할 값

Return Value

lower_bound	value보다 첫번째로 작지 않은 원소가 있는 위치의 Iterator 반환 조건을 만족하는 원소가 없다면 last 반환
upper_bound	value보다 첫번째로 큰 원소가 있는 위치의 Iterator 반환 조건을 만족하는 원소가 없다면 last 반환

출처: cppreference.com

lower_bound, upper_bound

```cpp
#include <array>
#include <algorithm>
using namespace std;

int main() {
    //    0-based index: 0  1  2  3  4  5  6  7
    array<int, 8> arr = {1, 2, 3, 3, 3, 4, 5, 6};

    // lower_bound 3: 1 2 3 3 3 4 5 6 _
    auto it = lower_bound(arr.begin(), arr.end(), 3);
    distance(arr.begin(), it); // -> index: 2

    // upper_bound 3: 1 2 3 3 3 4 5 6 _
    it = upper_bound(arr.begin(), arr.end(), 3);
    distance(arr.begin(), it); // -> index: 5

    // lower_bound 4: 1 2 3 3 3 4 5 6 _
    it = lower_bound(arr.begin(), arr.end(), 4);
    distance(arr.begin(), it); // -> index: 5

    // upper_bound 4: 1 2 3 3 3 4 5 6 _
    it = upper_bound(arr.begin(), arr.end(), 4);
    distance(arr.begin(), it); // -> index: 6

    // lower_bound 8: 1 2 3 3 3 4 5 6 _
    it = lower_bound(arr.begin(), arr.end(), 8);
    distance(arr.begin(), it); // -> index: 8
}
```

lower_bound, upper_bound

11053번 - 가장 긴 증가하는 부분 수열

문제

수열 A가 주어졌을 때, 가장 긴 증가하는 부분 수열을 구하는 프로그램을 작성하시오.

예를 들어, 수열 A = {10, 20, 10, 30, 20, 50} 인 경우에 가장 긴 증가하는 부분 수열은 A = {**10**, **20**, 10, **30**, 20, **50**} 이고, 길이는 4이다.

입력

첫째 줄에 수열 A의 크기 N (1 ≤ N ≤ 1,000)이 주어진다.

둘째 줄에는 수열 A를 이루고 있는 A_i가 주어진다. (1 ≤ A_i ≤ 1,000)

출력

첫째 줄에 수열 A의 가장 긴 증가하는 부분 수열의 길이를 출력한다.

예제 입력 1 복사

```
6
10 20 10 30 20 50
```

예제 출력 1 복사

```
4
```

출처: www.acmicpc.net

lower_bound, upper_bound

Input 10 20 10 30 20 50

value –

LIS

```
int N, value;
vector<int> LIS;
scanf("%d%d", &N, &value);
LIS.push_back(value);
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    if (LIS.back() < value) {
        LIS.push_back(value);
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        LIS[distance(LIS.cbegin(), it)] = value;
    }
}
```

lower_bound, upper_bound

Input 20 10 30 20 50

value 10

LIS


```
vector<int> LIS;
scanf("%d", &value);
// value: 10
// before: []
LIS.push_back(value);
// after:  [10]
```

lower_bound, upper_bound

Input 10 30 20 50

value 20

LIS

10	v				

```
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    // value:      20
    // LIS.back(): 10
    if (LIS.back() < value) {
        // before: [10]
        LIS.push_back(value);
        // after:  [10, 20]
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        LIS[distance(LIS.cbegin(), it)] = value;
    }
}
```

lower_bound, upper_bound

Input 30 20 50

value 10

LIS	**10**	**20**				

```
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    // value:      10
    // LIS.back(): 20
    if (LIS.back() < value) {
        LIS.push_back(value);
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        // before: [10, 20], index: 0
        LIS[distance(LIS.cbegin(), it)] = value;
        // after:  [10, 20]
    }
}
```

lower_bound, upper_bound

Input 20 50

value 30

LIS	**10**	**20**	**v**			

```
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    // value:      30
    // LIS.back(): 20
    if (LIS.back() < value) {
        // before: [10, 20]
        LIS.push_back(value);
        // after:  [10, 20, 30]
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        LIS[distance(LIS.cbegin(), it)] = value;
    }
}
```

lower_bound, upper_bound

Input 50

value 20

LIS	**10**	**20**	**30**			

```
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    // value:      20
    // LIS.back(): 30
    if (LIS.back() < value) {
        LIS.push_back(value);
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        // before: [10, 20, 30], index: 1
        LIS[distance(LIS.cbegin(), it)] = value;
        // after:  [10, 20, 30]
    }
}
```

lower_bound, upper_bound

Input –

value 50

LIS	10	20	30	v		

```
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    // value:      50
    // LIS.back(): 30
    if (LIS.back() < value) {
        // before: [10, 20, 30]
        LIS.push_back(value);
        // after:  [10, 20, 30, 50]
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        LIS[distance(LIS.cbegin(), it)] = value;
    }
}
```

lower_bound, upper_bound

Input -

value -

LIS	10	20	30	50		

```
vector<int> LIS;
scanf("%d", &value);
LIS.push_back(value);
for (int i = 1; i < N; ++i) {
    scanf("%d", &value);
    if (LIS.back() < value) {
        LIS.push_back(value);
    } else {
        auto it = lower_bound(LIS.cbegin(), LIS.cend(), value);
        LIS[distance(LIS.cbegin(), it)] = value;
    }
}
printf("%lu", LIS.size()); // -> 4
```

distance

- 두 iterator 간의 거리를 구하는 함수

Defined in header <iterator>

```
template< class InputIt >
typename std::iterator_traits<InputIt>::difference_type
    distance( InputIt first, InputIt last );
```

Parameters

first	첫번째 원소를 가리키는 iterator
last	마지막 원소를 가리키는 iterator

Return Value

	first부터 시작하여 last까지 도달하기까지의 iterator의 증가 횟수 도달할 수 없는 경우에 대한 행동은 정의되지 않음
c++11 이후	(위의 내용에 추가) 임의 접근이 가능한 iterator 타입이고 first, last가 접근 가능한 경우 음수가 반환될 수 있음

출처: cppreference.com

distance

```cpp
#include <iterator>
#include <array>

using namespace std;

int main()
{
    array<int, 3> arr = { 3, 1, 4 };
    distance(v.begin(), v.end()); // -> 3
    distance(v.end(), v.begin()); // -> -3
}
```

```cpp
#include <iterator>
#include <forward_list>

using namespace std;

int main()
{
    forward_list<int> li = { 3, 1, 4 };
    auto size = distance(li.begin(), li.end()); // -> 3
}
```

unique

- 범위 안의 연속되는 값의 첫번째 원소만 남기는 함수
- 범위 안 중복된 원소들을 지우는 함수가 아님
- 컨테이너의 크기를 조절하는 함수가 아님

Defined in header `<algorithm>`

```
template< class ForwardIt >
ForwardIt unique( ForwardIt first, ForwardIt last );
```

Parameters

first	시작 지점을 가리키는 iterator
last	범위의 마지막을 가리키는 iterator

Return Value

줄어든 범위의 end를 가리키는 iterator가 반환됨

출처: cppreference.com

unique

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

using namespace std;

int main()
{
    vector<int> arr{1, 1, 2, 2, 3, 3};
    // before: 1 1 2 2 3 3
    auto new_end = unique(arr.begin(), arr.end());
    // after:  1 2 3 x x x
    arr.erase(new_end, arr.end());
    // result: 1 2 3
}
```

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

using namespace std;

int main()
{
    vector<int> arr{1, 1, 2, 2, 3, 3, 2, 2};
    // before: 1 1 2 2 3 3 2 2
    auto new_end = unique(arr.begin(), arr.end());
    // after:  1 2 3 2 x x x x
    arr.erase(new_end, arr.end());
    // result: 1 2 3 2
}
```

* 어떤 값이 들어있을지 모르는 경우 x로 표시했다.

next_permutation

- 현재 순열의 다음 순서의 순열을 찾아주는 함수

Defined in header <algorithm>

```
template< class BidirIt >
bool next_permutation( BidirIt first, BidirIt last );
```

Parameters

first	첫번째 원소를 가리키는 iterator
last	마지막 원소를 가리키는 iterator

Return Value

만들어진 순열이 이전 순열보다 크다면 true / 아니라면 false
즉, 주어진 순열이 마지막 순열이라면 false를 반환한다.

출처: cppreference.com

next_permutation

```cpp
#include <cstdio>
#include <algorithm>
#include <array>

using namespace std;

int main() {
    array<int, 3> arr{1, 2, 3};
    do {
        for (auto i : arr) {
            printf("%d ", i);
        }
        puts("");
    } while (next_permutation(arr.begin(), arr.end()));
    /** output
     *  1 2 3
     *  1 3 2
     *  2 1 3
     *  2 3 1
     *  3 1 2
     *  3 2 1
     */
}
```

next_permutation

15649번 - N과 M (1)

문제

자연수 N과 M이 주어졌을 때, 아래 조건을 만족하는 길이가 M인 수열을 모두 구하는 프로그램을 작성하시오.

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열

입력

첫째 줄에 자연수 N과 M이 주어진다. (1 ≤ M ≤ N ≤ 8)

출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

출처: www.acmicpc.net

next_permutation

15649번 - N과 M (1)

$_nP_r$ 을 구하는 문제

N = 4, R = 2 일 때

1 2 3 4
1 3 2 4
1 4 2 3
2 1 3 4
2 3 1 4
2 4 1 3
3 1 2 4
3 2 1 4
3 4 1 2
4 1 2 3
4 2 1 3
4 3 1 2

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

using namespace std;

int N, R;
vector<int> arr;

void print_array() {
    for (int i = 0; i < R; ++i) {
        printf("%d ", arr[i]);
    }
    puts("");
}

int main() {
  scanf("%d%d", &N, &R);
  for (int i = 1; i <= N; ++i) {
      arr.push_back(i);
  }
  do {
    print_array();
    reverse(arr.begin() + R, arr.end());
  } while (next_permutation(arr.begin(), arr.end()));
}
```

next_permutation

15650번 - N과 M (2)

문제

자연수 N과 M이 주어졌을 때, 아래 조건을 만족하는 길이가 M인 수열을 모두 구하는 프로그램을 작성하시오.

- 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열
- 고른 수열은 오름차순이어야 한다.

입력

첫째 줄에 자연수 N과 M이 주어진다. (1 ≤ M ≤ N ≤ 8)

출력

한 줄에 하나씩 문제의 조건을 만족하는 수열을 출력한다. 중복되는 수열을 여러번 출력하면 안되며, 각 수열은 공백으로 구분해서 출력해야 한다.

수열은 사전 순으로 증가하는 순서로 출력해야 한다.

출처: www.acmicpc.net

next_permutation

15650번 - N과 M (2)

$_nC_r$을 구하는 문제

N = 4, R = 2 일 때

```
1 2    C: 0 0 1 1
1 3       1 2
1 4
2 3    C: 0 1 0 1
2 4       1   3
3 4
       C: 0 1 1 0
          1     4

       C: 1 0 0 1
            2 3

       C: 1 0 1 0
            2   4

       C: 1 1 0 0
              3 4
```

```cpp
#include <cstdio>
#include <algorithm>

using namespace std;

int N, R, C[8];

void print_array() {
    for (int i = 0; i < N; ++i) {
        if (C[i] == 0) {
            printf("%d ", i + 1);
        }
    }
    puts("");
}

int main() {
  scanf("%d%d", &N, &R);
  for (int i = R; i < N; ++i) {
      // if N = 5 and R = 3 then
      //    C = 0 0 0 1 1
      C[i] = 1;
  }
  do {
      print_array();
  } while (next_permutation(C, C + N));
}
```

nth_element

- 주어진 원소들의 n번째 위치에 정렬 하였을때 n번째에 위치하게 되는 원소를 가져다 놓음
- 정렬을 보장하지 않음
- 내부적으로 quickselect를 사용; 최근 c++ STL 구현체에서는 Introselect를 사용
  quickselect (worst) : $O(n^2)$ Introselect (worst) : $O(n \log n)$

Defined in header <algorithm>

```
template< class RandomIt >
void nth_element( RandomIt first, RandomIt nth, RandomIt last );
```

Parameters

first	첫번째 원소의 위치를 가리키는 iterator
nth	n번째 위치를 가리키는 iterator
last	마지막 원소의 위치를 가리키는 iterator

Return Value

없음

출처: cppreference.com

nth_element

11004번 - K번째 수

문제

수 N개 A_1, A_2, ..., A_N이 주어진다. A를 오름차순 정렬했을 때, 앞에서부터 K번째 있는 수를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 N(1 ≤ N ≤ 5,000,000)과 K (1 ≤ K ≤ N)이 주어진다.

둘째에는 A_1, A_2, ..., A_N이 주어진다. ($-10^9 \le A_i \le 10^9$)

출력

A를 정렬했을 때, 앞에서부터 K번째 있는 수를 출력한다.

nth_element

11004번 - K번째 수

N = 10, K = 5
arr = 7 8 9 10 1 3 2 4 5 6 일 때

5
arr = 1 2 3 4 **5** 6 10 9 8 7

기준이 된 5번째 원소 왼쪽에는 그보다 작은 원소들이 오른쪽에는 그보다 큰 원소가 있다.

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

using namespace std;

int N, K;
vector<int> arr;

int main() {
  scanf("%d%d", &N, &K);
  arr.resize(N);
  for (int i = 0; i < N; ++i) {
      scanf("%d", &arr[i]);
  }
  K -= 1;
  nth_element(arr.begin(), arr.begin() + K, arr.end());
  printf("%d", arr[K]);
}
```

partial_sort

- 주어진 범위의 지정된 위치까지만 정렬을 하는 함수
- 내부적으로 heap을 이용해 구현

Defined in header <algorithm>

```
template< class RandomIt >
void partial_sort( RandomIt first, RandomIt middle, RandomIt last );
```

Parameters

first	첫번째 원소의 위치를 가리키는 iterator
middle	정렬할 범위를 지정하는 iterator
last	마지막 원소의 위치를 가리키는 iterator

Return Value

없음

출처: cppreference.com

partial_sort

```cpp
#include <algorithm>
#include <array>

using namespace std;

int main()
{
    array<int, 10> arr{5, 7, 4, 2, 8, 6, 1, 9, 0, 3};
    // before: 5 7 4 2 8 6 1 9 0 3
    partial_sort(arr.begin(), arr.begin() + 3, arr.end());
    // after:  0 1 2 7 8 6 5 9 4 3
}
```

sort, stable_sort

- 정렬 함수
- 정렬을 할 원소의 개수에 따라 내부적으로 정렬 알고리즘을 선택
- 같은 값을 가진 원소들을 정렬할 때 stable_sort는 그 값들이 등장하는 순서가 바뀌지 않음

Defined in header <algorithm>

```
template< class RandomIt >
void sort( RandomIt first, RandomIt last );
template< class RandomIt >
void stable_sort( RandomIt first, RandomIt last );
```

Parameters

first	첫번째 원소의 위치를 가리키는 iterator
last	마지막 원소의 위치를 가리키는 iterator

Return Value

없음

출처: cppreference.com

sort, stable_sort

sort() 실행 결과

```
0, 0
0, 5
1, 6
1, 1
2, 7
2, 2
3, 3
3, 8
4, 4
4, 9
```

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

using namespace std;

struct Pair {
    int value;
    int nth;
};

bool operator<(const Pair& lhs, const Pair& rhs) {
    return lhs.value < rhs.value;
}

int main()
{
    vector<Pair> v;

    for (int i = 0; i < 10; ++i) {
        v.push_back(Pair{(i % 5), i});
    }

    random_shuffle(v.begin(), v.end());
    sort(v.begin(), v.end());

    for (const Pair& e : v)
        cout << e.value << ", " << e.nth << endl;
}
```

sort, stable_sort

stable_sort() 실행 결과

```
0, 0
0, 5
1, 1
1, 6
2, 2
2, 7
3, 3
3, 8
4, 4
4, 9
```

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

using namespace std;

struct Pair {
    int value;
    int nth;
};

bool operator<(const Pair& lhs, const Pair& rhs) {
    return lhs.value < rhs.value;
}

int main()
{
    vector<Pair> v;

    for (int i = 0; i < 10; ++i) {
        v.push_back(Pair{(i % 5), i});
    }

    random_shuffle(v.begin(), v.end());
    stable_sort(v.begin(), v.end());

    for (const Pair& e : v)
        cout << e.value << ", " << e.nth << endl;
}
```

comparisons

Defined in header `<functional>`

- less

```
template< class T >
struct less;

auto comp = std::less<int>();
comp(1, 2); // true
comp(2, 2); // false
```

- greater

```
template< class T >
struct greater;

auto comp = std::greater<int>();
comp(1, 2); // false
comp(2, 2); // true
```

- less_equal

```
template< class T >
struct less_equal;

auto comp = std::less_equal<int>();
comp(1, 2); // true
comp(2, 2); // true
```

- greater_equal

```
template< class T >
struct greater_equal;

auto comp = std::greater_equal<int>();
comp(1, 2); // false
comp(2, 2); // true
```

- equal_to

```
template< class T >
struct equal_to;

auto comp = std::equal_to<int>();
comp(1, 2) // false
comp(2, 2) // true
```

- not_equal_to

```
template< class T >
struct not_equal_to;

auto comp = std::not_equal_to<int>();
comp(1, 2) // true
comp(2, 2) // false
```

출처: cppreference.com

comparisons

sort(_1, _2, less<int>()) 실행 결과

```
0
1
2
3
4
5
6
7
8
9
```

sort(_1, _2, greater<int>()) 실행 결과

```
9
8
7
6
5
4
3
2
1
0
```

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

using namespace std;

int main()
{
    vector<int> v;
    for (int i = 0; i < 10; ++i) {
        v.push_back(i);
    }

    random_shuffle(v.begin(), v.end());
    sort(v.begin(), v.end(), less<int>());

    for (const auto& e : v) {
        cout << e << endl;
    }

    random_shuffle(v.begin(), v.end());
    sort(v.begin(), v.end(), greater<int>());
    for (const auto& e : v) {
        cout << e << endl;
    }
}
```

misc

- max - a, b 중 큰 값을 반환

```
template< class T >
const T& max( const T& a, const T& b );
```

```
int a = 2, b = 3;
double c = 1.0, d = 2.0;
max(a, b); // -> 3
max(c, d); // -> 2.0
```

- minmax - min, max를 동시에 반환

```
template< class T >
std::pair<const T&,const T&> minmax( const T& a, const T& b );
```

```
int a = 2, b = 3;
auto p = minmax(a, b);
p.first; // -> 2
p.second; // -> 3
```

- max_element - 원소들 중 가장 큰 값 반환

```
template< class ForwardIt >
ForwardIt min_element( ForwardIt first, ForwardIt last );
```

```
array<int, 4> arr{1, 2, 3, 4};
max_element(arr.begin(), arr.end()); // -> 4
```

- shuffle - 지정된 범위의 원소들을 임의로 섞음

```
template< class RandomIt, class URBG >
void shuffle( RandomIt first, RandomIt last, URBG&& g );
```

```
#include <array>
#include <algorithm>
#include <random>

using namespace std;

int main() {
    array<int, 4> arr{1, 2, 3, 4};
    int seed = 0;
    mt19937 g(seed);
    // before: 1 2 3 4
    shuffle(arr.begin(), arr.end(), g);
    // after:  1 3 2 4
}
```

- reverse - 지정된 범위의 원소들의 순서를 뒤집음

```
template< class BidirIt >
void reverse( BidirIt first, BidirIt last );
```

```
array<int, 4> arr{1, 2, 3, 4};
// before: 1 2 3 4
reverse(arr.begin(), arr.end());
// after:  4 3 2 1
```

출처: cppreference.com

Q & A

감사합니다