22차시

Queue
문제 풀이

문제

- 큐 BOJ 10845
- 덱 BOJ 10866
- 카드2 BOJ 2164
- 식당 메뉴 BOJ 26043
- 회전하는 큐 BOJ 1021

큐 BOJ 10845

- 구현해야 할 함수: push, pop, size, empty, front, back
- 모두 STL Queue와 함수명이 같고, 기능도 동일하다

- Queue를 구현 또는 STL Queue를 연습하는 문제
- 큐를 직접 구현해도 되며, STL을 활용해보자

BOJ 10845

```cpp
#include <queue>

int n, x;
string op;
queue<int> q;

cin >> n;
while (n--) {
    cin >> op;
}
```

BOJ 10845

```
if (op == "push") {
    cin >> x;
    q.push(x);
}
if (op == "pop") {
    cout << (q.empty() ? -1 : q.front()) << '\n';
    q.pop();
}
if (op == "size")
    cout << q.size() << '\n';
if (op == "empty")
    cout << (q.empty() ? 1 : 0) << '\n';
if (op == "front")
    cout << (q.empty() ? -1 : q.front()) << '\n';
if (op == "back")
    cout << (q.empty() ? -1 : q.back()) << '\n';
```

덱 BOJ 10866

- 구현해야 할 함수: push_front, push_back, pop_front, pop_back, size, empty, front, back
- 모두 STL Deque와 함수명이 같고, 기능도 동일하다

- Deque를 구현 또는 STL Deque를 연습하는 문제
- 덱를 직접 구현해도 되며, STL을 활용해보자

덱 BOJ 10866

```cpp
#include <queue>
#include <deque>

int n, x;
string op;
deque<int> dq;

cin >> n;
while (n--) {
    cin >> op;
}
```

덱 BOJ 10866

```
if (op == "push_front") {
    cin >> x;
    dq.push_front(x);
}
if (op == "push_back") {
    cin >> x;
    dq.push_back(x);
}
if (op == "pop_front") {
    cout << (dq.empty() ? -1 : dq.front()) << '\n';
    dq.pop_front();
}
if (op == "pop_back") {
    cout << (dq.empty() ? -1 : dq.back()) << '\n';
    dq.pop_back();
}
```

덱 BOJ 10866

```
if (op == "size")
    cout << dq.size() << '\n';
if (op == "empty")
    cout << (dq.empty() ? 1 : 0) << '\n';
if (op == "front")
    cout << (dq.empty() ? -1 : dq.front()) << '\n';
if (op == "back")
    cout << (dq.empty() ? -1 : dq.back()) << '\n';
```

카드2 BOJ 2164

- 1번 카드가 제일 위에, N번 카드가 제일 아래에 가도록 N장의 카드가 존재한다
- 제일 위에 있는 카드를 버리고, 그 다음 카드를 제일 아래로 옮기는 작업을 반복
- 마지막에 남은 카드는 무엇인가

0	0	0	1	1	0	0	1

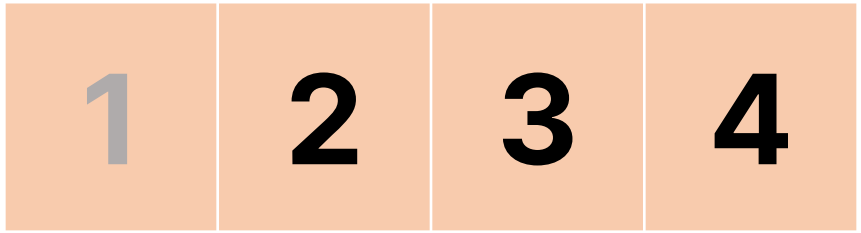
카드2 BOJ 2164

- 4장의 카드를 살펴보자
- 제일 처음에는 다음과 같다

1	2	3	4

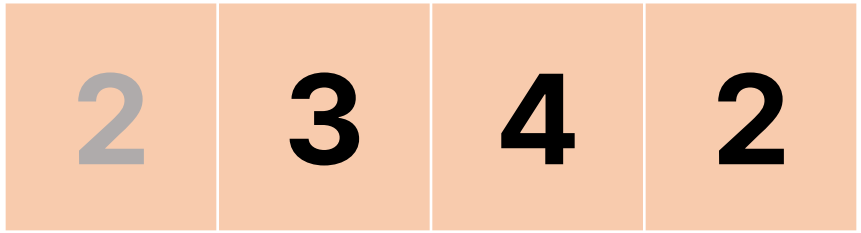
카드2 BOJ 2164

- 제일 위의 1을 버린다



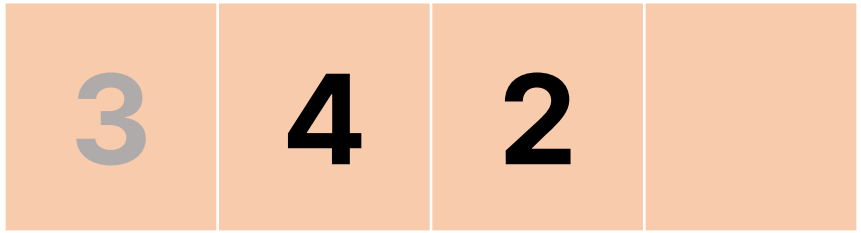
카드2 BOJ 2164

- 제일 위의 2를 제일 아래로 옮긴다



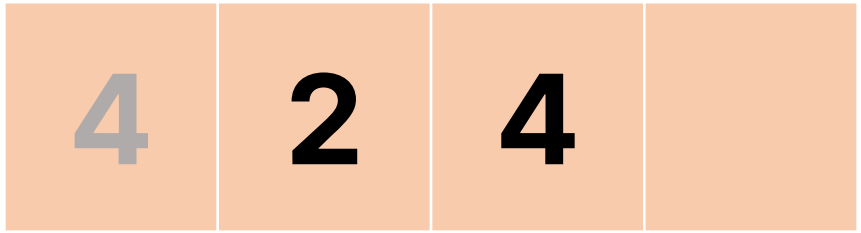
카드2 BOJ 2164

- 제일 위의 3을 버린다



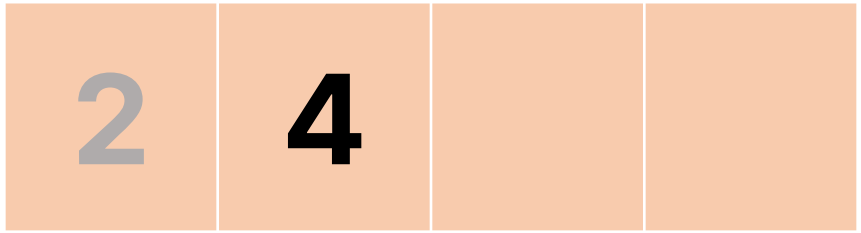
카드2 BOJ 2164

- 제일 위의 4를 제일 아래로 옮긴다



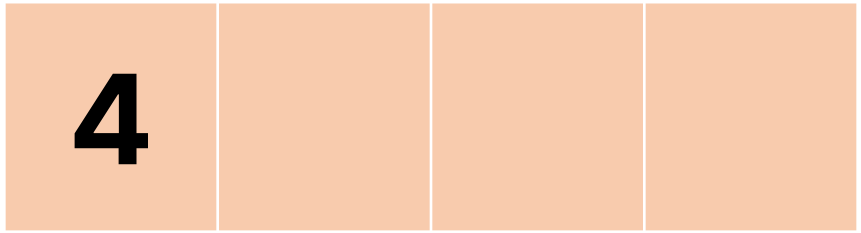
카드2 BOJ 2164

- 제일 위의 2를 버린다



카드2 BOJ 2164

- 카드가 한 장 남을 때 까지 반복하므로, 4가 정답이다



카드2 BOJ 2164

- 지금까지의 과정을 보면 항상 저장된 어떠한 자료구조의 위(왼)쪽에서 데이터가 제거되며, 아래(오른)쪽에서 데이터가 삽입됨
- 큐의 형태와 유사함

- 카드가 1부터 삽입된 데이터라고 가정할 때, 카드는 FIFO의 구조를 보임
- 즉, 큐를 사용하여 처리 작업을 나타낼 수 있다.

카드2 BOJ 2164

- 카드를 제거하고, 옮기는 작업의 반복이므로 홀수차시에는 카드를 제거, 짝수차시에는 카드를 옮기면 된다

카드2 BOJ 2164

```
int n;
queue<int> q;

cin >> n;
for (int i = 1; i <= n; i++)
    q.push(i);

for (int i = 1; q.size() != 1; i++) {
    if (i % 2 == 1) { // if (i & 1 == 1)
        q.pop();
    } else {
        q.push(q.front());
        q.pop();
    }
}
cout << q.front();
```

카드2 BOJ 2164

- 두 연산을 나누어야 할까?
- 카드가 감소하는 경우는 홀수차시에만 감소
- 따라서 카드를 옮기는 연산 이후에는 반드시 카드를 빼는 연산을 해야한다

카드2 BOJ 2164

```
if (q.size() != 1)
    q.pop();

while (q.size() != 1) {
    // 카드 옮기기
    q.push(q.front());
    q.pop();

    // 카드 버리기
    q.pop();
}

cout << q.front();
```

식당 메뉴 BOJ 26043

- 식사가 2종류가 있으며, 학생들은 각각 선호하는 식사가 있다
- 선호하는 식사를 한 학생과 선호하지 않은 식사를 한 학생, 식사를 하지 못한 학생을 구분하여 출력하여라

식당 메뉴 BOJ 26043

- 줄을 설 때는 맨 뒤에 줄을 서며, 제일 앞에 있는 학생부터 식사를 하므로 FIFO구조이다
- 즉, 대기열을 큐로 관리하면 된다

- 결과를 출력할 때는 오름차순을 이용해 출력해야 하므로 정렬한 뒤에 출력해야한다
- 벡터나 배열에 결과를 저장, 정렬 후 출력한다

식당 메뉴 BOJ 26043

- 원하는 식사를 했는지 확인하기 위해, 각 학생이 선호하는 식사를 기억해야 한다
- 배열에 각 학생의 선호 식사를 저장하거나 큐에 넣을 때 학생번호와 선호하는 유형을 같이 넣는다

식당 메뉴 BOJ 26043

- 대기열이 필요 – 큐 1개
- 원하는 식사를 한 학생과 원하는 식사를 하지 못한 학생을 구분하여 저장 – 벡터 2개
- 식사를 하지 못한 학생들을 한 곳에 모아 정렬할 벡터가 필요 – 벡터 1개 또는 기존 벡터 재사용

식당 메뉴 BOJ 26043

- 식사 1인분이 준비됐을 때는 식당 입구에서 대기 중인 학생이 반드시 존재하므로, 대기열에서 학생을 빼기 전에 빈 대기열인지 확인할 필요가 없다

식당 메뉴 BOJ 26043

```
int n, a, b, op;
queue<int> q;
vector<int> v1, v2;
int preference[500001];

cin >> n;
while (n--) {
    cin >> op;
    if (op == 1) {
        cin >> a >> b;
    } else { // op == 2
        cin >> b;
    }
}
```

식당 메뉴 BOJ 26043

```
if (op == 1) {
    cin >> a >> b;
    cin >> a >> b;
    q.push(a);
    preference[a] = b;
} else { // op == 2
    cin >> b;
    if (preference[q.front()] == b)
        v1.push_back(q.front());
    else
        v2.push_back(q.front());
    q.pop();
}
```

식당 메뉴 BOJ 26043

```
if (op == 1) {
    cin >> a >> b;
    cin >> a >> b;
    q.push(a);
    preference[a] = b;
} else { // op == 2
    cin >> b;
    (preference[q.front()] == b ? v1 : v2).push_back(q.front());
    q.pop();
}
```

식당 메뉴 BOJ 26043

```
if (!v1.empty()) {
    sort(v1.begin(), v1.end());
    for (auto a : v1)
        cout << a << ' ‘;
    cout << '\n';
} else
    cout << "None\n";

if (!v2.empty()) {
    sort(v2.begin(), v2.end());
    for (auto a : v2)
        cout << a << ' ‘;
    cout << '\n’;
} else
    cout << "None\n";
```

식당 메뉴 BOJ 26043

```
while (!q.empty()) {
     v.push_back(q.front());
     q.pop();
}

if (!v.empty()) {
    sort(v.begin(), v.end());
    for (auto a : v)
        cout << a << ' ‘;
    cout << '\n';
} else
    cout << "None\n";
```

회전하는 큐 BOJ 1021

- 첫 번째 원소를 뽑아낸다
- 왼쪽으로 한 칸 이동한다
- 오른쪽으로 한 칸 이동한다
- 2, 3번 연산을 최소한으로 사용하여 원하는 숫자를 뽑아내라

회전하는 큐 BOJ 1021

- 첫번째(제일 왼쪽)인 경우 2, 3번(회전) 연산을 사용할 필요가 없다
- 왼쪽에 가까운 경우, 왼쪽으로만 이동하는 것이 최소
- 오른쪽에 가까운 경우, 오른쪽으로만 이동하는 것이 최소

- 즉, 한 쪽 방향으로만 이동한다

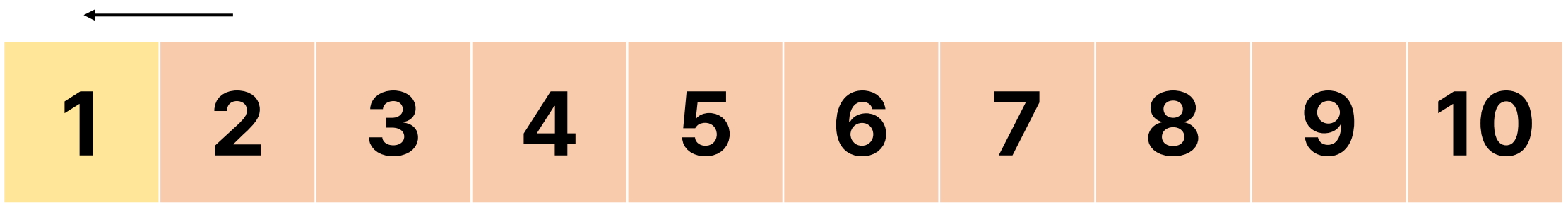
회전하는 큐 BOJ 1021

- 10까지 넣은 큐에서 2를 뽑는다고 해보자

1	2	3	4	5	6	7	8	9	10

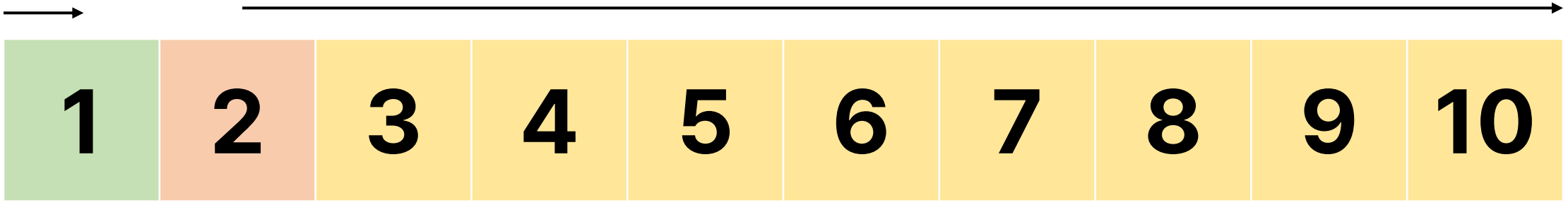
회전하는 큐 BOJ 1021

- 2를 왼쪽으로 이동하는 경우 한 칸만 이동하면 된다



회전하는 큐 BOJ 1021

- 2를 오른쪽으로 이동하는 경우 끝까지 이동하고 한 번 더 이동해야 한다



회전하는 큐 BOJ 1021

- 위치를 알아내면 된다
- find함수를 이용해 deque에서 특정 원소의 위치를 찾을 수 있다
- 위치를 알아낸 다음 더 가까운 방향으로 이동시키면 된다

- 2, 3번 연산은 deque를 이용해 구현이 가능하다

회전하는 큐 BOJ 1021

```
deque<int> dq;
int n, m;

cin >> n >> m;
for (int i = 1; i <= n; i++)
    dq.push_back(i);

int ans = 0, inp;
while (m--) {
    cin >> inp;
    int ind = find(dq.begin(), dq.end(), inp) - dq.begin();
}
```

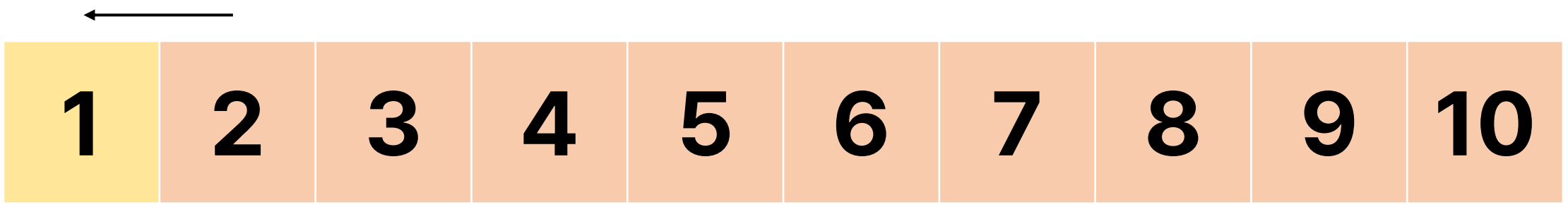
회전하는 큐 BOJ 1021

- 10까지 넣은 큐에서 2를 뽑는다고 해보자

1	2	3	4	5	6	7	8	9	10

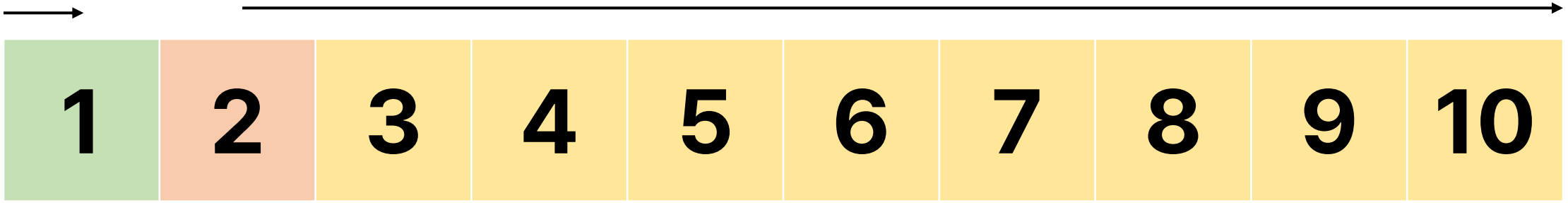
회전하는 큐 BOJ 1021

- 2를 왼쪽으로 이동하는 경우 한 칸만 이동하면 된다



회전하는 큐 BOJ 1021

- 2를 오른쪽으로 이동하는 경우 끝까지 이동하고 한 번 더 이동해야 한다



회전하는 큐 BOJ 1021

- 내가 왼쪽에서 K번째라면
- 왼쪽으로 갈 때는 K-1번 움직이면 된다
- 오른쪽으로 갈 때는 N-K +1번 움직이면 된다

- 둘을 합치면 N임을 알 수 있다

회전하는 큐 BOJ 1021

- 뽑으려는 수가 첫번째에 올 때까지 이동시킨다
- 그렇다면 반대쪽으로 이동할 때 사용할 연산의 횟수는 N-(한쪽 이동에 사용한 연산수) 이다

- 둘 중 최소값을 더하면 된다

회전하는 큐 BOJ 1021

- 문제 분류는 덱이다
- 왼쪽으로만 이동시키면서 왼쪽에서는 데이터의 제거가 일어나며 오른쪽에서는 데이터가 추가된다
- 큐를 사용해도 문제를 풀 수 있다

회전하는 큐 BOJ 1021

```
queue<int> q;
int n, m;

cin >> n >> m;
for (int i = 1; i <= n; i++)
    q.push(i);
```

회전하는 큐 BOJ 1021

```
int ans = 0, cur, inp;

while (m--) {
    cur = 0;
    cin >> inp;
    while (q.front() != inp) {
        q.push(q.front());
        q.pop();
        cur++;
    }
    ans += min(cur, (int)q.size() - cur);
    q.pop();
}

cout << ans;
```

Binary Search

이분 탐색

- 이분 탐색을 하기 위해서는 어느 한 지점에 대해 판단했을 때 절반에 공통된 내용을 적용할 수 있어야 한다
- ex) 50보다 작은 수라면, 50보다 큰 수들은 모두 정답이 아니다
- ex) 주사위에서 나온 수를 찾으려고 할 때, 3이 아니다라고 할 때, 1, 2, 4, 5, 6 중 답이 아닌 것을 확정 지을 수 없다

이분 탐색

- 찾으려는 범위는 정렬이 되어있어야 한다
- 찾으려는 범위를 정확하게 다루어야 한다

이분 탐색

- 1 부터 10까지의 수 중에서 찾는다면 찾으려는 범위는 [1, 10], [1, 11), (0, 10], (0, 11) 로 표현할 수 있다
- 이분 탐색을 사용할 때는 범위의 시작 값을 포함하는지, 끝 값을 포함하는지를 정확히 해야 한다

이분 탐색

- 찾으려는 범위의 시작과 끝 그리고 찾으려는 값을 함수의 인자로 전달한다
- 찾으려는 범위에 따라 탐색의 조건이 달라진다

이분 탐색

- 시작과 끝 값을 포함할 때, 시작 값 L = 1, 끝 값 R =10으로 표현할 수 있다
- 해당 범위에서는 L<=R인 경우 해당 범위 안에 값이 하나 이상 존재하므로 L<=R인 범위에서 탐색을 진행하면 된다

```
bool binary_search(int l, int r, int x) {
    while (l <= r) {
        // do binary search
    }
}
```

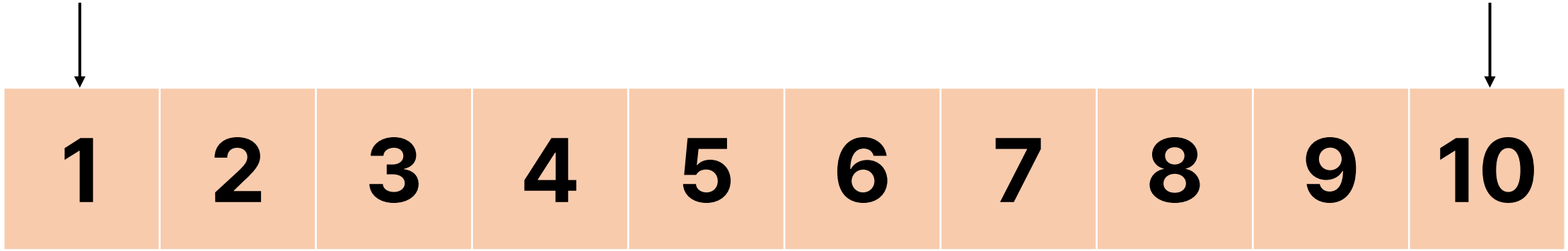
이분 탐색

- 시작 값을 포함하며 끝 값을 포함하지 않을 때, 시작 값 L = 1, 끝 값 R =11로 표현할 수 있다
- 해당 범위에서는 L<R인 경우 해당 범위 안에 값이 하나 이상 존재하므로 L<R인 범위에서 탐색을 진행하면 된다

```
bool binary_search(int l, int r, int x) {
    while (l < r) {
        // do binary search
    }
}
```

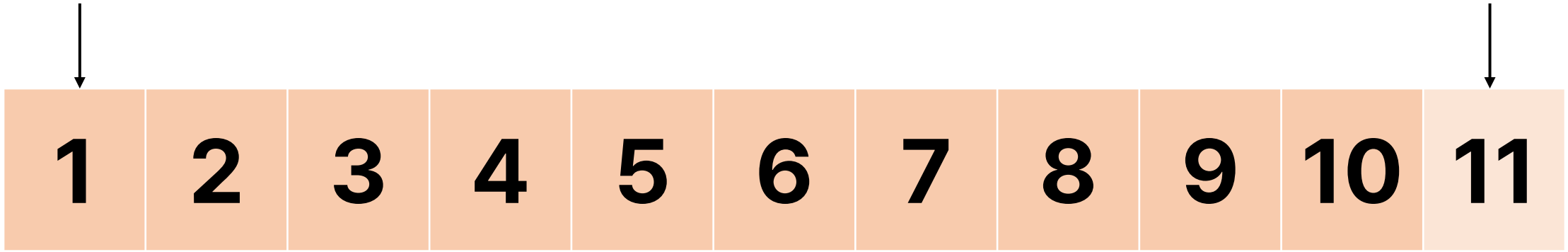
이분 탐색

- 범위 [1, 10]



이분 탐색

- 범위 [1, 11)



이분 탐색

- 중간을 잡고 탐색을 진행해야 한다
- M = (L + R) / 2
- M이 찾으려는 값인지, 찾으려는 값이 아니라면 찾으려는 값보다 크거나 작은 지 확인해야한다

이분 탐색

- 현재 M보다 값이 작은 경우 찾으려는 값은 [L, M) 범위에 존재한다

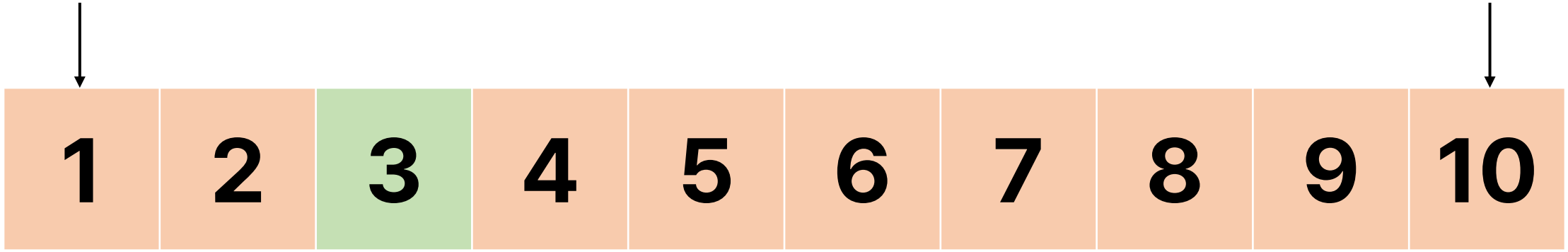
  R값에 M-1을 대입한다

- 반대로 M보다 값이 큰 경우 찾으려는 값은 (M, R] 범위에 존재한다

  L값에 M+1을 대입한다

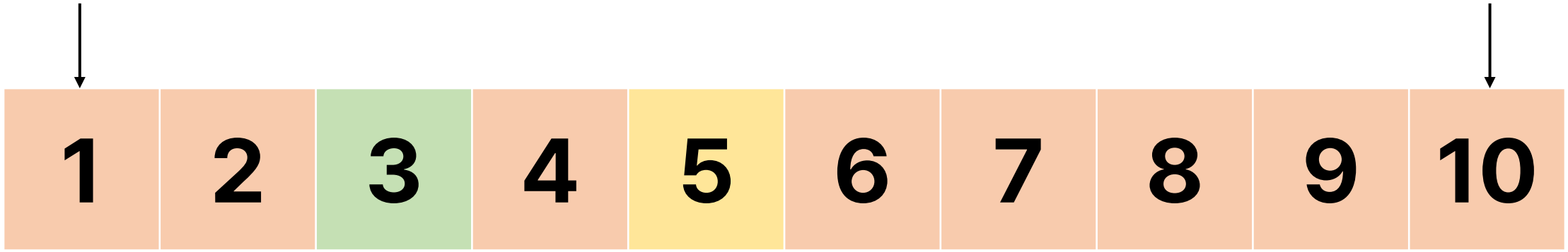
이분 탐색

- 범위 [1, 10], 찾으려는 값 3



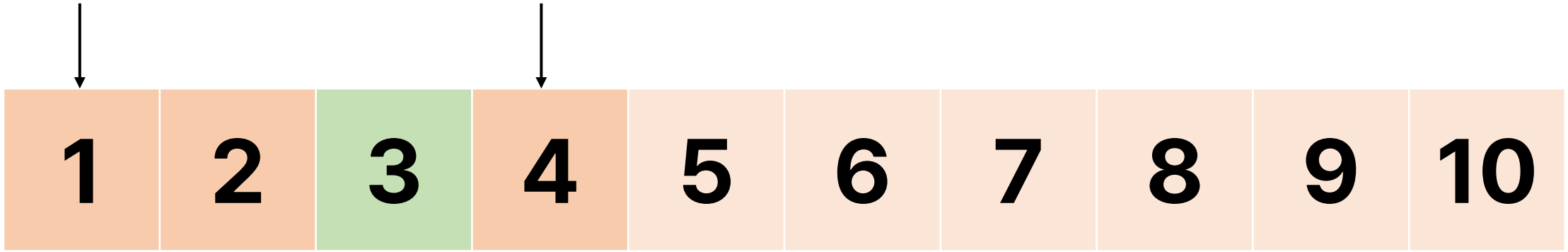
이분 탐색

- M = (1 + 10) / 2 = 5(소수점 아래 버림), M값에 대해 확인



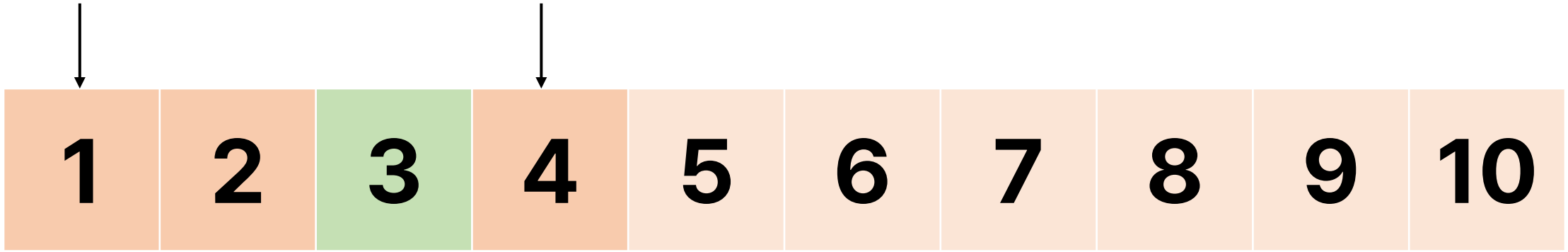
이분 탐색

- M값보다 찾는 값이 작으므로, 찾는 범위를 수정



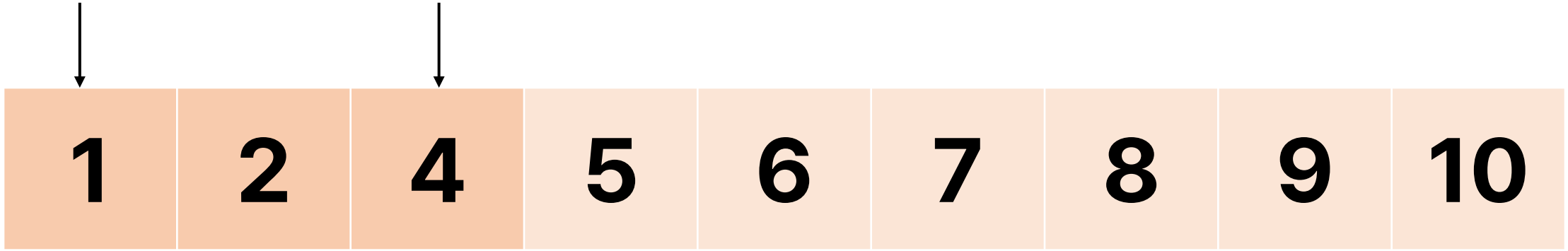
이분 탐색

- 값을 찾을 때 까지 반복



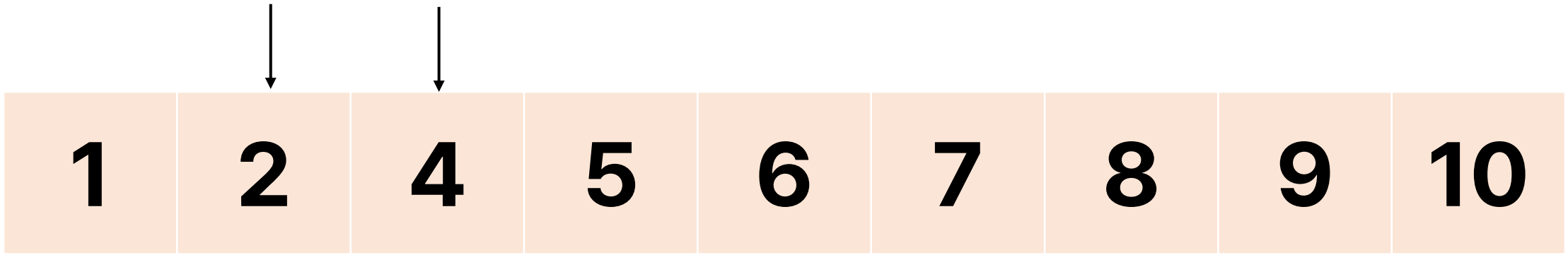
이분 탐색

- 만일 찾으려는 값이 존재하지 않는 경우



이분 탐색

- 탐색을 진행하다 보면 찾는 범위에 더 이상 값이 존재하지 않는다, 찾는 값이 없다



이분 탐색

```
bool binary_search(int l, int r, int x) {
    while (l <= r) {
        int m = (l + r) / 2;
        if (x > arr[m])
            l = m + 1;
        else
            r = m - 1;
    }
    return false;
}
```

이분 탐색

- 이분탐색은 STL로 구현되어 있음

- #include <algorithm>
- binary_search: 값이 존재하는 지 이분탐색으로 검색하고 true/false를 리턴함

이분 탐색

- 이를 이용하여 함수가 lower_bound와 upper_bound

- lower_bound: 찾는 수보다 같거나 큰 수 중에서 제일 왼쪽의 index
- upper_bound: 찾는 수보다 큰 수 중에서 제일 왼쪽의 index

- 즉, 값이 있는 것이 보장이 된다면 lower_bound를 이용해 값을 찾을 수 있다

Coordinate Compression

문제

- 좌표 압축 BOJ 18870
- N차원 여행 BOJ 12867
- 개수 세기 BOJ 10807
- 통계학 BOJ 2108

좌표 압축 BOJ 18870

- 기본적인 좌표 압축을 해보자
- 중간에 비는 값 없이 연속된 값으로 좌표를 배정해야 한다
- 좌표들의 중복된 값을 제거하고 값이 증가하는 순서대로 좌표를 배정해주어야 한다

좌표 압축 BOJ 18870

- 먼저 들어온 값들을 배열에 저장해야한다

```
int n;
vector<int> v;
cin >> n;

v.resize(n);
for (auto &a : v)
    cin >> a;

int num;
for (int i = 0; i < n; i++) {
    cin >> num;
    v.push_back(num);
}
```

좌표 압축 BOJ 18870

- 배열에서 중복된 값을 제거해야한다
- unique를 사용하기 위해서는 배열이 정렬되어 있어야 한다

```
vector<int> unique_v(v);
sort(unique_v.begin(), unique_v.end());
unique_v.erase(unique(unique_v.begin(), unique_v.end()), unique_v.end());
```

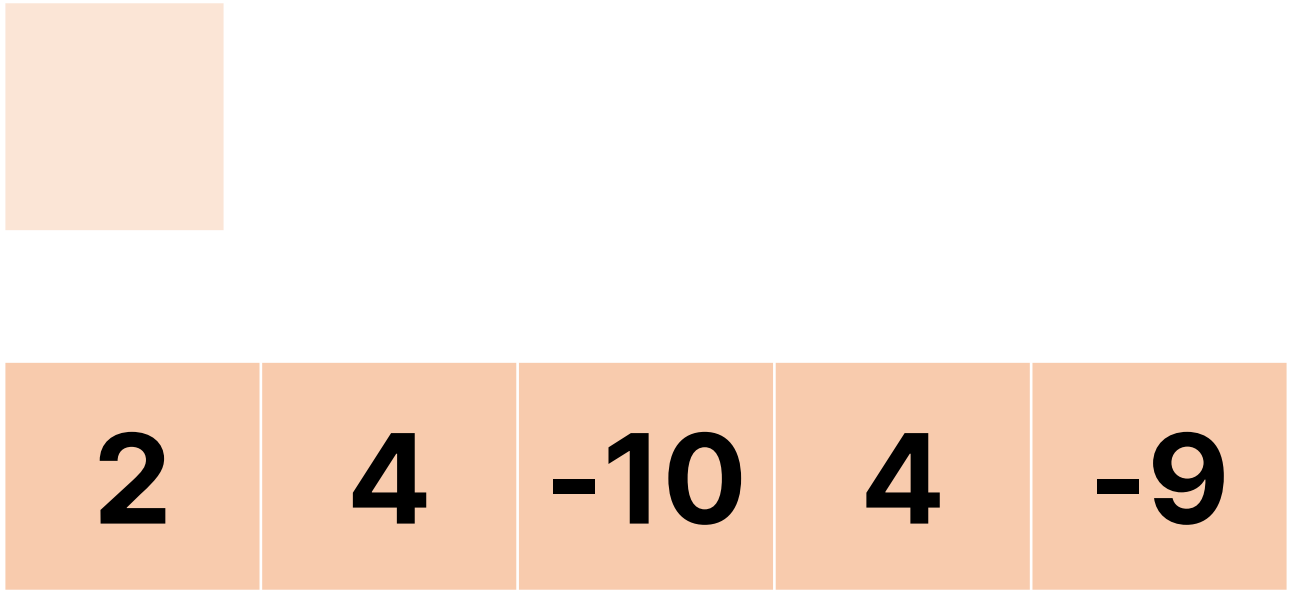
좌표 압축 BOJ 18870

- 배열에서 중복된 값을 제거해야한다
- unique를 사용하기 위해서는 배열이 정렬되어 있어야 한다

```
vector<int> unique_v, tmp(v);
sort(tmp.begin(), tmp.end());
unique_v.push_back(tmp[0]);
for (int i = 1; i < n; i++)
    if (unique_v.back() != tmp[i])
        unique_v.push_back(tmp[i]);
```

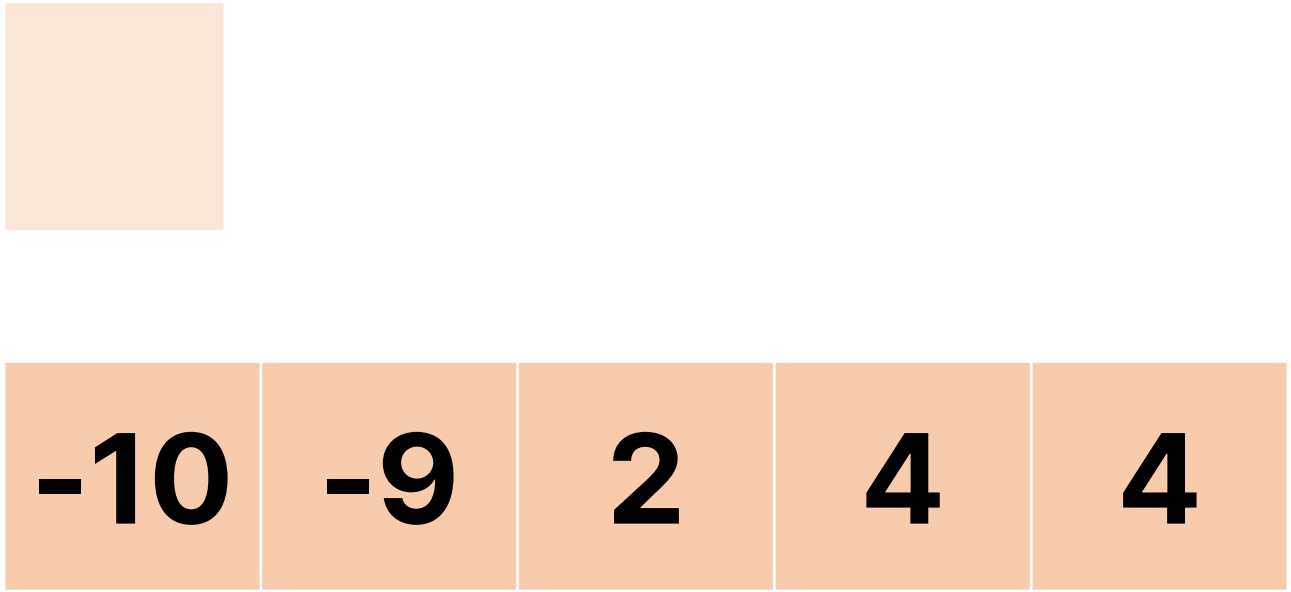
좌표 압축 BOJ 18870

- unique_v는 빈 벡터, tmp 벡터는 입력의 복사



좌표 압축 BOJ 18870

- tmp 벡터를 정렬



좌표 압축 BOJ 18870

- 제일 첫 값을 복사해 넣음

-10

-10	-9	2	4	4

좌표 압축 BOJ 18870

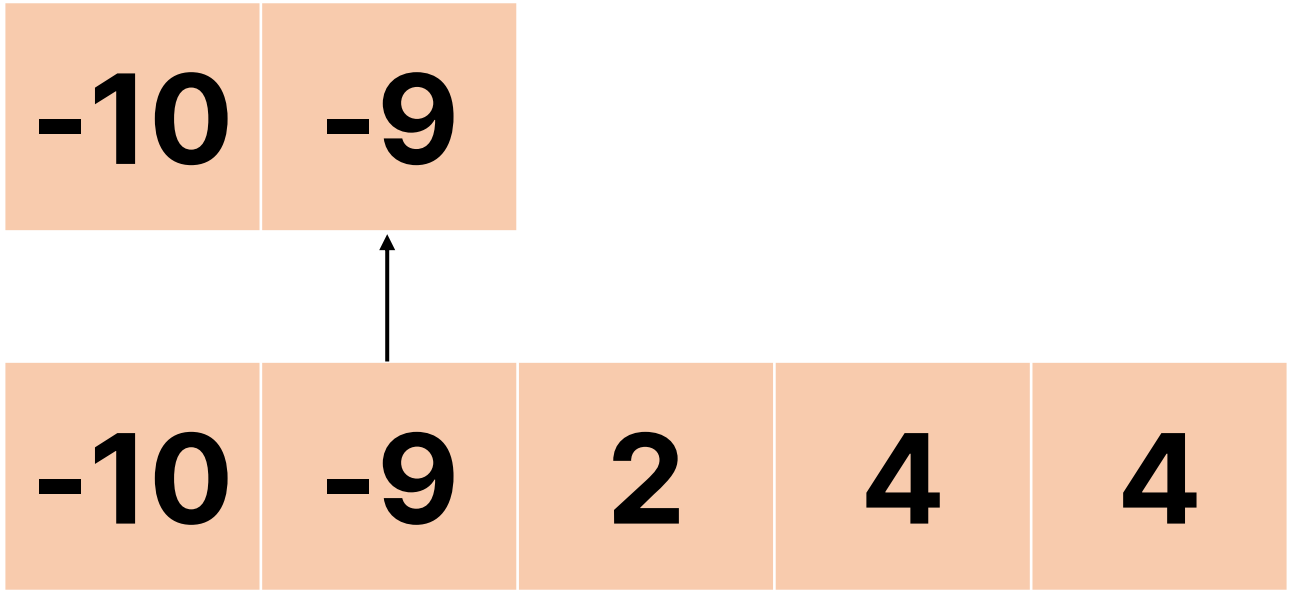
- 두번째 값부터 확인하며, unique_v에 값이 없다면 집어 넣는다

-10

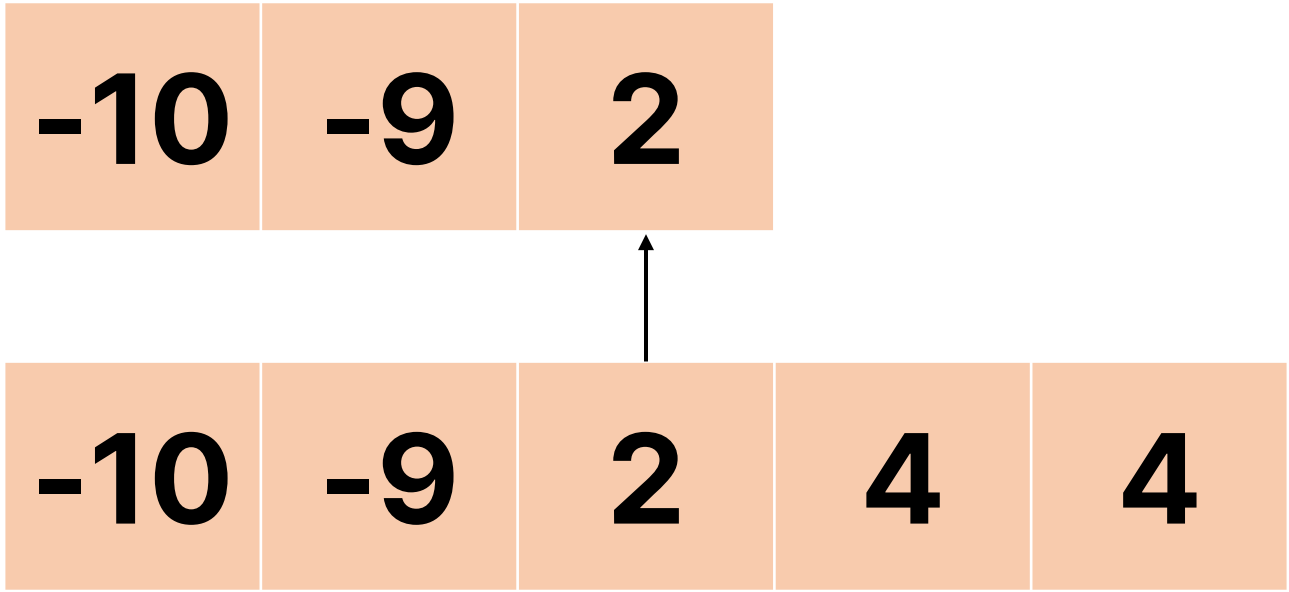
-10	-9	2	4	4

좌표 압축 BOJ 18870

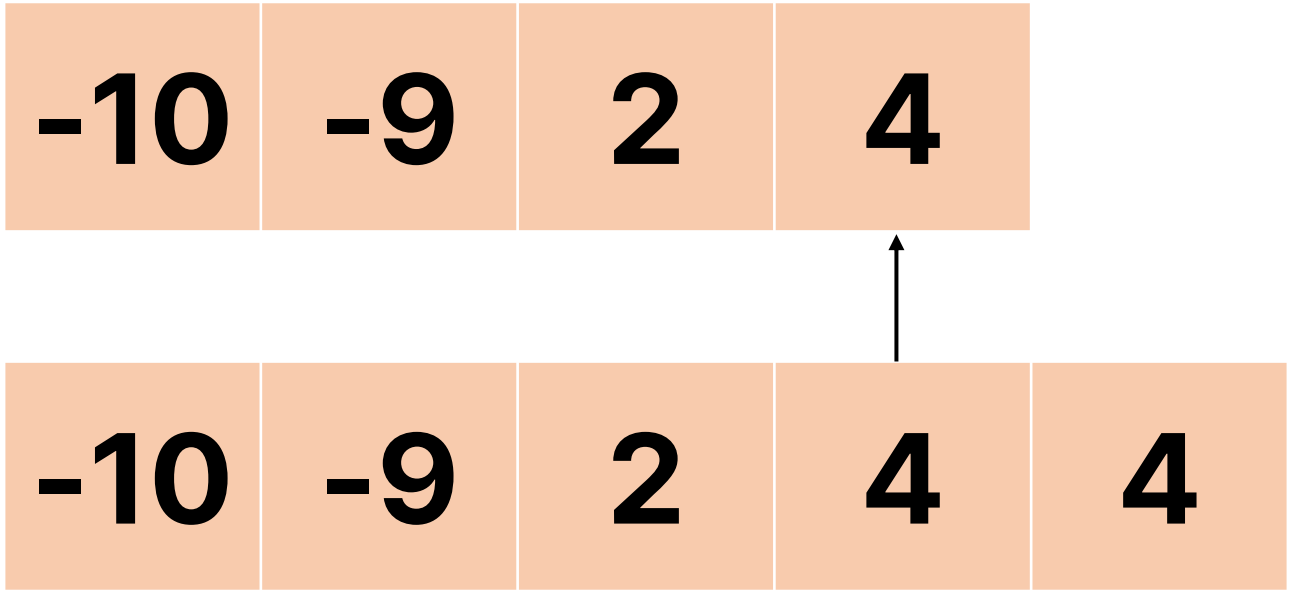
- 정렬되어 있으므로 unique_v의 제일 마지막 값만 확인하면 된다(Monotone하다)



좌표 압축 BOJ 18870



좌표 압축 BOJ 18870



좌표 압축 BOJ 18870

- 4는 이미 들어와 있는 값이므로 unique_v에 삽입하지 않는다

-10	-9	2	4

-10	-9	2	4	4

좌표 압축 BOJ 18870

- 중복을 제거한 벡터를 이용해 좌표를 부여한다

```
for (auto a : v)
    for (int i = 0; i < unique_v.size(); i++)
        if (a == unique_v[i])
            cout << i << ' ';

for (auto a : v)
    cout << (lower_bound(unique_v.begin(), unique_v.end(), a) - unique_v.begin()) << ' ';
```

N차원 여행 BOJ 12867

- N차원 좌표를 다루어야 한다
- 중복된 좌표를 방문하는 경우 0, 그렇지 않은 경우 1을 출력
- $1 \leq N \leq 1\,000\,000\,000$

N차원 여행 BOJ 12867

- 1,000,000,000 차원의 좌표를 다루는 것은 비효율적
- 최대 여행 계획은 50까지이므로 최대 50개의 인덱스만 움직인다
- 그 외의 인덱스들은 값이 변하지 않으므로 고려할 필요가 없다

N차원 여행 BOJ 12867

- 3번째 줄에는 움직일 인덱스의 위치가 나온다
- 이를 좌표 압축하여 사용
- Map을 이용하면 방문한 좌표들을 손쉽게 관리할 수 있다

N차원 여행 BOJ 12867

- 3번째 줄에는 움직일 인덱스의 위치가 나온다
- 이를 좌표 압축하여 사용
- Map 또는 Set을 이용하면 방문한 좌표들을 손쉽게 관리할 수 있다

N차원 여행 BOJ 12867

```
struct Coordinates {
    int arr[50];
}
```

N차원 여행 BOJ 12867

```
int n, m;
vector<int> v;
cin >> n >> m;

v.resize(m);
for (auto &a : v)
    cin >> a;
```

N차원 여행 BOJ 12867

```
vector<int> compressed(v);
sort(v.begin(), v.end());
for (auto &a : compressed)
    a = lower_bound(v.begin(), v.end(), a) - v.begin();
```

N차원 여행 BOJ 12867

```
char direction;
Coordinates cur;
set<Coordinates> s;
s.insert(cur);
```

N차원 여행 BOJ 12867

```
for (auto a : compressed) {
    cin >> direction;
    if (direction == '+')
        cur.arr[a]++;
    else
        cur.arr[a]--;
    if (s.find(cur) != s.end()) {
        cout << 0;
        return 0;
    }
    s.insert(cur);
}
cout << 1;
```

N차원 여행 BOJ 12867

- 새롭게 만든 구조체를 이용해 STL을 사용해야한다
- 새로 만든 구조체는 연산자에 대한 정의가 없다

- 연산자를 새롭게 만들어 주어야 한다

N차원 여행 BOJ 12867

- 연산자도 함수이다
- ex) 벡터에서 삽입: vector.push_back(k)
- 연산자는 나누어 쓰지만 동일한 방식이다

ex) a + b -> a.+(b)

N차원 여행 BOJ 12867

- 같은 이름의 함수를 사용하는 법 2가지: 오버로딩, 오버라이딩
- 오버로딩: 매개변수의 종류, 개수 등을 다르게 해 서로 다른 함수로 인식하게 하는 것
- 오버라이딩: 부모 자식 관계에서 새로운 함수를 만들어 기존의 함수를 덮어 씌우는 것

- 구조체에 새롭게 연산자를 만드는 것은 동일한 이름의 연산자를 다른 매개변수로 만드는 것이므로 오버로딩이다

N차원 여행 BOJ 12867

- 연산자 오버로딩
- a < b라고 한다면 a는 함수를 실행하는 구조체, b는 매개변수 t로 들어가게 된다
- 연산자 조건을 만족하면 true, 아니면 false를 반환

```
struct T {
    bool operator<(const T &t) const {
        // code
    }
};
```

N차원 여행 BOJ 12867

- ex) 등급이 낮을수록, 등급이 같다면 점수가 높을수록 크다

```
struct T {
    int grade;
    int score;

    bool operator<(const T &t) const {
        if (grade == t.grade)
            return score < t.score;
        return grade < t.grade;
    }
};
```

N차원 여행 BOJ 12867

- Set은 <연산자와 == 연산자가 필요하다

```
struct Coordinates {
    int arr[50];

    bool operator<(const Coordinates &t) const {
        for (int i = 0; i < 50; i++) {
            if (arr[i] == t.arr[i]) {
                continue;
            return arr[i] > t.arr[i];
        }
        return false;
    }
};
```

N차원 여행 BOJ 12867

- Set은 <연산자와 == 연산자가 필요하다

```
struct Coordinates {
    int arr[50];

    bool operator==(const Coordinates &t) const {
        for (int i = 0; i < 50; i++)
            if (arr[i] != t.arr[i])
                return false;
        return true;
    }
};
```

문제

- 멀티버스 II BOJ 18869