이분 탐색

업다운 게임을 잘하는 법

- 50을 말했는데 상대는 UP이라고 했다
- 그럼 49,48, ...1을 질의할 필요가 있는가?
- 50의 대답은 Up이고 75의 대답은 down이다
- 우린 51~74중 하나를 질의하면 된다
- 이런 상황에서 몇을 질의하면 게임이 잘 풀릴까?

이분 탐색

1. 무한한 범위에선 탐색할 일이 없다. 하한start과 상한end을 정한다
2. 현재 탐색 범위는 (start, end)이다. 이제 탐색범위를 절반 줄여 나간다. 절반씩 필요 없는 부분을 날리는 것이다
3. 계속 절반씩 줄이다가 탐색 범위의 크기가 1이 된다면 찾은 것이다!

시간 복잡도

- $\log(N)$
- 어떤 값(초기 탐색범위)을 1이 될 때까지 2로 나눈다는 것은?

이분 탐색(기본)

```
int N;          // 배열의 길이
int A[N];       // 정렬된 배열
int left = 0;  // 탐색 범위 시작
int right = N; // 탐색 범위 끝
int target;    // 찾고 싶은 값
while (left <= right)
{
    int mid = (left + right) / 2;
    if (A[mid] == target)
    {
        // 탐색이 완료됨
        return mid;
    }
    else if (A[mid] < target)
    {
        // mid보다 작은 값들은 볼 필요가 없음
        left = mid + 1;
    }
    else
    { // A[mid] > target
        // mid보다 큰 값은 볼 필요 없음
        right = mid - 1;
    }
}
// 찾지 못함
return -1;
```

이런 것도 있다.

- 이상(lower bound)/초과(upper bound)를 찾을 수도 있다
- 이상/초과를 찾을 수 있으면 이하/미만도 찾기 가능!
- 코드를 보면서 알아갑시다

이상(lower bound)

- target은 우리의 목적
- 정렬된 배열에서 target이상이 되는 수가 있는 배열의 최초의 인덱스를 찾는다.

이상(lower bound)

```
int N;          // 배열의 길이
int A[N];       // 정렬된 배열
int left = 0;  // 탐색 범위 시작
int right = N; // 탐색 범위 끝
int target;     // 찾고 싶은 값
while (left < right) // 왜 등호가 없을까?
{
    int mid = (left + right) / 2;
    if (A[mid] >= target)
    {
        right = mid; // 포함시킨다는 의미
    }
    else
    { // A[mid] < target
        left = mid + 1; // 포함 안시킨다는 의미
    }
}
return left;
```

초과(upper bound)

- 정렬된 배열에서 target초과가 되는 수가 있는 배열의 최초의 인덱스를 찾는다.

초과(upper bound)

```
int N;          // 배열의 길이
int A[N];       // 정렬된 배열
int left = 0;  // 탐색 범위 시작
int right = N; // 탐색 범위 끝
int target;     // 찾고 싶은 값
while (left < right)
{
    int mid = (left + right) / 2;
    if (A[mid] <= target)
    {
        left = mid + 1;
    }
    else
    { // A[mid] > target
        right = mid;
    }
}
return left;
```

Desc하다면..?

- 배열을 뒤집을 수도 있지만 귀찮다. 게다가 가끔은 불가능하다
- 마이너스와 대소관계
- [5, 3, 1]은 감소한다. [-5, -3, -1]은 증가한다

두 용액 BOJ 2470

문제

두 용액 BOJ 2470

- 같은 종류로만 생각하면 정렬하고 0에 가까운 것 2개만 선택
- 다른 종류끼리 선택한다면?
- -77을 포함한다 했을 때 77을 찾는다면 행복한 일이다

인문예술탐사주간 BOJ 23829

문제

인문예술탐사주간 BOJ 23829

- P가 오름차순 정렬 돼있다고 생각해보자
- $P_i \leq X_k < Pj$ 이라고 하자.
- 점수는 $X_k - P_1 + X_k - P_2 + \ldots + X_k - P_i + P_j - X_k + P_{j+1} - X_k + \ldots + P_n - X_k$
- 누적합을 최적화
- lower bound을 이용해 X_k 가 P의 어디 사이에 위치해 있는지 찾을 수 있다.

합이 0 BOJ 3151

문제

합이 0 BOJ 3151

- 1명일 땐 그냥 0인걸 찾기
- 2명일 땐 $O(N^2)$ 하지만 더 효율적인 것이 있다
- 3명일 때 이 효율적인 방식을 응용한다면?

합이 0인 네 정수 BOJ 7453

문제

합이 0인 네 정수 BOJ 7453

- 전 문제(BOJ 3151)의 아이디어를 활용할 수 없을까?

이분 탐색은 도구이므로...

- 코드를 외우자! 정말 많이 쓰입니다
- 외우고 문제에 맞게 커스텀 하는게 맞는 것 같다

질문

- NEXT: 매개변수 탐색

매개 변수 탐색

- 절반씩 필요 없는 부분을 날린다는 철학
- 문제에 최대 최소라는 키워드가 있으면 의심해보기
- 매개 변수와 결정 함수를 찾기 f(x)
- 무엇을 매개 변수로 하고 결정 함수를 어떻게 작성할 것인가?
- 결정 함수는 그리디, dp, 투포인터등을 이용해 구현

매개 변수

- 그래서 우리가 무엇을 기준으로 이분탐색 알고리즘을 적용할 것인가?
- 결정함수에 매개 변수로 넣을 값
- 매개 변수에 따라 문제의 조건에 해당하는 값이 바뀌어야 한다
- 그래야 매개 변수에 따라 절반을 날릴 수 있다

결정 함수

- 문제 속의 작은 문제라고 생각하자
- 추가로 input값이 주어지면(조건이 추가되면) 문제를 쉽게 풀 수 있다는 관점
- 결정 함수가 통과한다면 더 혹독한 조건, 거짓이라면 더 널널한 조건
- f(매개변수) = 문제의 조건 통과 여부

템플릿 예시(최대 찾기)

```cpp
bool check(int param)
{
    // param으로 문제의 조건을 통과하는지 작성
    return true;
}

int paramSearch()
{
    int left = 0;       // 탐색 범위
    int right = 999999; // 탐색 범위
    int ans = left;     // 정답
    while (left <= right)
    {
        int mid = (left + right) / 2;
        if (check(mid))
        {
            // 통과했으면 정답을 갱신시키고, 조건을 더 가혹하게
            ans = max(ans, mid);
            left = mid + 1;
        }
        else
        {
            // 더 널널한 조건
            right = mid - 1;
        }
    }
    return ans;
}
```

과일 탕후루 BOJ 30804

문제

과일 탕후루 BOJ 30804

- 정해는 매개 변수 탐색이 아닌 $O(N)$ 투포인터
- 하지만, 많은 문제가 복잡도가 $\log(N)$ 차이가 난다고 시간 초과가 나진 않는다
- 실전에서 생각이 잘 나지 않거나, 구현이 복잡할 것 같으면 매개 변수 탐색이 대안이 될 수도 있다

나무 자르기 BOJ 2805

문제

나무 자르기 BOJ 2805

- 톱날이 낮을 수록 얻어가는 나무가 많다
- 톱날을 0부터 1씩 올린다면 이 문제를 푸는데 한세월이다
- 높이X로 설정했을 때 M만큼 얻었다면, X보다 낮다면 M이상으로 얻을 것
- 높이X로 설정했을 때 M보다 적게 얻었다면, X보다 높다면 M보다 적게 얻을 것

성심당 밀키트 BOJ 24041

문제

성싶당 밀키트 BOJ 24041

- 세균수가 중요한데 일을 모른다면 골치 아프다
- 일을 정한다면 무엇을 빼야 할까? 어떻게 빼야 할까?

카드 팩 구매하기 BOJ 15823

문제

카드 팩 구매하기 BOJ 15823

- 카드 팩당 카드 수량이 많아질수록 제약이 생겨서 카드 팩 개수가 줄어들 것이다
- 투 포인터와의 융합 문제: 어떻게 카드가 이미 포함됐는지 알 수 있는가?

숫자 구슬 BOJ 2613

문제

숫자 구슬 BOJ 2613

- 세그먼트가 많아진다면 최댓값은 줄어들까 늘어날까?
- 케이스 처리를 고민해보기. 조건에 맞기만 한다고 그게 답일까?

질문

- NEXT: LIS

최장 증가 부분 수열 LIS

- 배열에서 수 몇개를 제거하자
- 이때 남은 수들이 차례로 증가하면 이를 증가 부분 수열이라고 한다
- 가능한 경우 중 가장 긴 수열이 LIS이다

비효율적인 알고리즘

- $O(N^2)$ DP
- 배열의 인덱스가 i일때 j를 $0 \sim i-1$까지 순회한다
- 배열의 j번째 값이 배열의 i번째 값보다 작다면 $\mathrm{dp[i]} = \max(dp[i],\ 1 + dp[j])$

비효율적인 알고리즘(코드)

```
int N;
int A[N];
int dp[N];
for (int i = 0; i < N; i++)
{
    for (int j = 0; j < i; j++)
    {
        if (A[i] > A[j])
        {
            dp[i] = max(dp[i], dp[j] + 1);
        }
    }
}
```

하지만 이분탐색, 스택과 함께 라면

- $O(N^2)$ 이 $O(NlogN)$으로 줄어드는 마술

무엇이 좋을까?

Index	0	1	2	3	4	...
Array	1	4	3	10	5	?

- 현재 가능한 LIS의 길이는 3
- 인덱스 4 다음엔 무엇이 올지 모른다. 그럼 어떤 것을 선택할까요?

1. [1, 4, 10]
2. [1, 3, 5]

- 그것이 스택이 담고 있는 정보가 될 것입니다

Point: 최대한 작게 만들기

- 스택이 담고 있는 정보를 이해하자
- 현재 스택의 i $(0 \leq i < stack.size())$번째 원소는
- 길이가 $i + 1$인 증가 부분 수열의 가능한 마지막 수 중 가장 작은 값이여야 한다

가장 작은 값을 만들기 위한 이분 탐색

- 스택의 원소는 결국 증가한다. → 이분 탐색이 사용 가능하다
- 현재 배열의 원소가 스택의 원소를 더 작게 바꿀 수 있는가?
- 바꿀 수 있다면 스택에서 어떤 원소를 바꿀 수 있는가?
- 이를 이분 탐색 - 이상(lower bound)로 찾을 수 있다

푸는 법

1. 만약에 스택이 비었거나, 배열의 원소가 스택의 마지막 원소보다 크다면 스택에 원소를 push_back()
2. 그렇지 않다면, 현재 배열의 원소를 target으로 설정하고 stack에서 이분 탐색 – lower bound알고리즘으로 바꿀 수 있는 인덱스를 찾고 바꿔준다
3. 스택의 길이가 정답

- 해당 알고리즘을 돌린 뒤 스택이 LIS 그 자체가 아님을 주의하자

푸는 법(부연 설명)

Index	0	1	2	3	4	...
Array	1	4	3	10	5	?

Index	0
Stack	1

스택이 비었으므로 1을 넣어준다.

Index	0	1
Stack	1	4

4는 스택의 마지막(1)보다 크므로 넣어준다.

Index	0	1
Stack	1	3

스택에서 3이상인 것을 찾고 바꿔준다.

가장 긴 증가하는 부분 수열 2 BOJ 12015

문제

연습 문제

2417 : 정수 제곱근 (많이 쓰이는 테크닉. 오차를 피할 수 있습니다.)

27977 : 킥보드로 등교하기(KUPC)

1114 : 통나무 자르기

14003 : 가장 긴 증가하는 부분 수열 5

30459 : 현수막 걸기(KUPC)

2977 : 폭탄 제조

1981 : 배열에서 이동

9460 : 메탈