이분 탐색

업다운 게임

- 상대는 $A \leq N \leq B (A \leq B,\ A$와 B는 정수$)$를 만족하는 정수 N을 선택한다
- 나는 상대에게 다음을 질의한다
- "x는 너가 생각한 수보다 커(업)? 작아(다운)? 똑같아(게임 끝)?"
- 상대는 N과 x를 비교해 그에 맞는 대답을 한다

업다운 게임

- $A = 0, B = 100$이면 처음엔 다 50을 질의한다
- 왜?

업다운 게임

- 게임을 제일 효율적으로 하는 법을 고민해보자
- 만약 상대가 20을 생각했고, 나는 40을 말했을 때 상대는 다운! 이라고 대답한다
- 그럼 나는 $41, 42, 43, 44, \ldots, B$를 질의할 필요가 있을까?

시간 복잡도

- 게임을 진행하면 할수록, 더 질의할 필요가 없는 수들의 범위가 점점 더 늘어난다
- 우리가 질의 해야 할, 즉 아직 결과를 확정 지을 수 없는 범위가 점점 더 줄어든다는 것
- 이 범위를 탐색 범위이라고 한다
- 예를 들어 $A = 0, B = 100$일 때 50을 질의했고 업! 이라는 대답을 들었다면
- 탐색 범위는 $[51,100]$이 된다
- 그럼 탐색 범위를 왜 절반 씩 줄이는게 효율적인 것일까?

시간 복잡도

- 우리의 목적은 탐색 범위의 크기를 1로 만드는 것이다
- 처음 탐색 범위의 크기를 $S_0 = B - A + 1$이라고 하자
- 게임을 효율적으로 진행해 K번 만에 끝냈다면
- $S_1 = \frac{S_0}{2}, S_2 = \frac{S_1}{2}, S_3 = \frac{S_2}{2}, \dots, S_k = 1$ 이다
- 계속 2로 나눠서 1이 되는 횟수는 log복잡도이다
- $k = \log(S_0)$

시간 복잡도

- 우리가 탐색할 구간의 크기가 N이라고 하자.
- 아무 생각 없이 탐색한다면, 처음부터 끝까지 봐야 하므로 $O(N)$
- 이전과 같이 효율적으로 탐색 범위를 절반 씩 줄여 탐색한다면 $O(\log(N))$

N	$log(N)$
100000	16
10000000	23
100000000000000	46

이분 탐색

- 탐색 범위를 절반씩 줄이면서 탐색해 나가는 것을 “이분 탐색”이라고 한다
- 아쉽게도 모든 상황에 대해 이분 탐색을 적용할 수 없다
- 탐색 범위를 줄이는게 가능해야 한다 = 더 탐색할 필요 없는 구간이 생긴다
- 대표적인 상황은 정렬된 배열에서의 탐색, 어떤 매개변수에 따른 결과값이 계속 커지는 상황에서의 특정 상황에 따른 매개변수를 찾는 탐색

코드로 구현해보기

- 우린 길이가 N인 정렬된 배열(A)에서 우리가 찾고 싶어하는 값($target$)을 찾아야한다
- 배열에서 찾는 것이므로 탐색범위는 $[left = 0,\ right = N\ - 1]$

```
int left = 0;  // 탐색 범위 시작
int right = N - 1; // 탐색 범위 끝
```

코드로 구현해보기

- 탐색 범위가 1이 될 때 까지 찾아야한다

```
while (left <= right){
  // 다음 슬라이드에 코드 공개
}
```

코드로 구현해보기

- 탐색 범위를 절반 씩 줄이기 위해 기준(mid)을 설정해야 한다.
- 현재 탐색 범위 시작과 끝을 절반 씩 나눈 값이 기준이다

```
while (left <= right){
  int mid = (left + right) / 2;
}
```

코드로 구현해보기

- 이제 탐색이 끝날 때 까지 $A[mid]$와 $target$을 비교한 결과에 따라 탐색 범위$(left, right)$를 변경한다

```
int left = 0;  // 탐색 범위 시작
int right = N - 1; // 탐색 범위 끝
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 못했을 때 케이스 처리
return -1;
```

질문1

- 배열에 중복되는 값들이 있으면 어떻게 되죠?
- 만약, $target$이 배열안에 2개 이상 있다면, 이분 탐색으로 "있다는 사실"은 찾을 수 있지만, 어떤 인덱스에 있는지는 정확히 알 수 없다

질문2

- 꼭 오름차순으로 정렬돼야 하나요?
- 약간의 트릭을 써주면 된다
- $[8, 6, 4, 2]$라는 내림차순으로 정렬된 배열을
- $[-8, -6, -4, -2]$와 같이 바꾼다
- 굳이 배열 원소에 -1을 곱하지 않고, 비교할 때 -1을 곱해서 비교하는 것을 추천

추가 질문

- NEXT: 이분 탐색을 약간 응용해 더 다양한 것들을 해보자

더 많은 것을 할 수 있다.

- $target$이상인 원소와 초과인 원소도 약간의 응용으로 찾을 수 있다
- 배열에서 $target$이상(초과)인 원소의 제일 빠른 인덱스를 찾는 것
- 이상을 찾는 것은 Lower bound
- 초과를 찾는 것은 Upper bound

목적 추가 설명용

- $target = 20$

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

이분 탐색 리마인드

- $target == mid$라면 $return\ mid$
- $target < mid$라면 $right = mid\ - 1$
- $target > mid$라면 $left = mid + 1$
- Lower Bound와 Upper Bound에서 중요한 것은 $target == mid$일 때 처리이다.

Lower bound

- $target$보다 크거나 같은 원소를 찾는다.
- mid가 $target$과 같다면 어떻게 해야 할까?
- 같아도 되므로 탐색 범위에 포함 시켜야 한다
- $target \leq mid$라면 $right = mid$
- $target > \ mid$라면 $left = mid + 1$

Lower bound

- $target$은 20이다.

0	1	2	3	4	5	6	7	8
10	20	20	20	20	20	20	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

코드로 구현해보기

- 우린 길이가 N인 정렬된 배열(A)에서 $target$ 이상인 원소를 찾아야한다

```
    int left = 0;  // 탐색 범위 시작
    int right = N; // 탐색 범위 끝
```

코드로 구현해보기

```
while (left < right) // 왜 등호가 없을까?
{
    int mid = (left + right) / 2;
    // 다음 슬라이드에서 mid와 A[target]을 비교해봅시다.
}
// 여기선 left == right
return left;
```

코드로 구현해보기

```
int left = 0;  // 탐색 범위 시작
int right = N; // 탐색 범위 끝
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;
```

Upper bound

- $target$보다 큰 원소를 찾는다.
- mid가 $target$과 같다면 어떻게 해야 할까?
- 같으면 안되므로 탐색 범위에 포함 시키면 안된다.
- $target < mid$라면 $right = mid$
- $target \geq mid$라면 $left = mid + 1$

Upper bound

- $target$은 20이다.

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

0	1	2	3	4	5	6	7	8
10	15	19	20	20	24	24	24	29

코드로 구현해보기

- $target == mid$ 일 때 케이스 처리만 달라진다

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

문제 풀이 전 질문 + 비고

- 코드를 이해하는 것을 넘어 외워야 한다. 왜냐하면 정말 많은 곳에서 쓰이기 때문이다.
- 이상과 초과를 찾았다면, 이하와 미만도 찾을 수 있다.

숫자 카드 BOJ 10815

- 문제

숫자 카드 BOJ 10815

- 있는지 없는지가 궁금하다. 단순히 탐색하면 된다
- 하지만 무작정 탐색한다면, 시간 초과가 발생한다
- 따라서 효율적으로 탐색해야 한다
- 숫자 카드 배열을 정렬한다. 그럼 이분 탐색 알고리즘을 사용할 수 있다

숫자 카드 BOJ 10815

```
    int babo = 10000001;
    while (left <= right)
    {
        int mid = (left + right) / 2;
        if (A[mid] == target)
        {
            // 탐색이 성공
            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 babo;
```

숫자 카드 BOJ 10815

- 해당 이분 탐색의 리턴 값에 따라 $0, 1$을 결정해주면 된다
- 이렇게 찾지 못했을 때 케이스처리는 나올 수 없는 수를 return해주면 된다

숫자 카드 2 BOJ 10816

- 문제

숫자 카드 2 BOJ 10816

- 단순히 있는지 없는지를 찾는 것은 부족하다
- 효율적인 탐색을 위해 숫자 카드 배열을 정렬한다
- Lower bound로 찾은 i, Upper bound로 찾은 j가 있다
- i, j를 통해 몇개 인지 알 수 있다
- $j - i$
- 굳이 찾지 못한 것에 대해 케이스처리를 할 필요는 없다

숫자 카드 2 BOJ 10816

- 왜냐하면 숫자 카드 배열이 정렬됐다면
- 그 배열의 어떤 수 x가 존재하는 인덱스 구간은
- $[x$이상인 수가 처음으로 나온 인덱스 $= i$, x초과인 수가 처음으로 나온 인덱스 $= j)$

```
for (int m = 0; m < M; m ++){
  int i = lowerBound(target[m]);
  int j = upperBound(target[m]);
  ans[m] = j - i;
}
```

숫자 카드 2 BOJ 10816

- 사실 map계열 자료구조를 사용해도 된다
- 카운터를 만드는 상황은 자주 마주할 것이다
- 편한대로 쓰면 된다
- 하지만 Lower bound, Upper bound를 이용한 테크닉도 알고 있는 것을 추천

표절 BOJ 2428

- 문제

표절 BOJ 2428

- 문제의 의미를 생각해보면, $size(F_i) \leq size(F_j)$, $size(F_i) \geq 0.9 * size(F_j)$
- 파일 크기 배열을 정렬한다. 자연스럽게 $i < j$ 라면 $size(F_i) \leq size(F_j)$를 만족

표절 BOJ 2428

- j를 $[1, N-1]$에 대해 순회
- 왜냐하면 $j = 0$은 의미가 없기 때문
- 0을 포함해도 큰 상관은 없다

표절 BOJ 2428

- 순회할 때 마다 탐색 범위는 $[0, j]$, $target = 0.9 * size(F_j)$ 의 Lower bound 알고리즘을 수행한다. 조건을 만족하는 i를 찾는 것이라고 생각하자
- $i \neq j$ 조건 때문에 $[0, j-1]$이라고 생각이 들 것 같지만 케이스 처리 방식에 따라 갈릴 뿐이다
- 만족하는 i가 0개라면 $[0, j]$로 두는 것이 더 편리하다.

표절 BOJ 2428

```
int cnt = 0;
for (int j = 0; j < N; j++){
  cnt += j - lowerBound(0, j, A[j] * 0.9)
}
```

질문

- NEXT: 매개 변수 탐색

배열에서만 이분 탐색이 가능할까?

- log복잡도는 억 단위의 input에서도 효율적
- 하지만, 그 정도 크기의 배열을 초기화하는 것에만 현실적으로 힘든 시간과 메모리가 필요
- 일부 문제는 배열로 모델링 하는 것 자체에 어려움이 존재
- 업다운 게임도 굳이 배열로 생각 안하고 풀고 있다

이분 탐색 다시 생각하기

```
while (left <= right)
{
    int mid = (left + right) / 2;
    if (myFunc(mid) == 0)
    {
        return mid;
    }
    else if (myFunc(mid) == 1)
    {
        left = mid + 1;
    }
    else if (myFunc(mid) == 2)
    {

        right = mid - 1;
    }
}
```

이분 탐색 다시 생각하기

- 강의 처음에 보았던 이분 탐색도 결론적으로 mid에 따른 어떤 결과(상태)에 따라 탐색 범위를 조정한다
- 업다운 게임도 $speak(number)$의 결과인 다운! 업! 에 따라 탐색 범위를 조정한다
- 어떤 문제와 그에 맞는 함수는 함수의 결과에 따라 이분 탐색 방식으로 탐색 범위를 조정할 수 있다

매개 변수 탐색

- 절반 씩 필요 없는 부분을 날린다
- 문제에 최대, 최소라는 키워드가 있으면 의심해보기
- 무엇을 매개 변수로 할 것인가?
- 결정 함수를 어떻게 작성할 것인가?
- 결정 함수는 문제 속의 작은 문제, 매개 변수라는 추가적인 input

매개 변수 탐색

- 절반 씩 필요 없는 부분을 날린다
- 문제에 최대, 최소라는 키워드가 있으면 의심해보기
- 무엇을 매개 변수로 할 것인가?
- 결정 함수를 어떻게 작성할 것인가?
- 결정 함수는 문제 속의 작은 문제, 매개 변수라는 추가적인 input

결정함수와 매개변수 코드 예시

```
int left = 0;       // 탐색 범위
int right = 999999; // 탐색 범위
int ans = left;     // 정답 초기화
while (left <= right)
{
    int mid = (left + right) / 2;
    // check가 결정 함수, mid가 매개 변수
    if (check(mid))
    {
        // 정답 갱신
        ans = max(ans, mid);
        // 결정 함수 리턴에 따라 탐색 범위 조정
        left = mid + 1;
    }
    else
    {
        // 결정 함수 리턴에 따라 탐색 범위 조정
        right = mid - 1;
    }
}
return ans;
```

이상한 술집 BOJ 13702

- 문제

이상한 술집 BOJ 13702

- K명에게 최대 ans만큼 막걸리를 분배할 수 있다
- 만약 $ans+1, ans+2, \ldots$만큼의 막걸리를 분배한다면 K명보다 적은 인원에게 분배
- 만약 $ans-1, ans-2, \ldots$만큼의 막걸리를 분배한다면 K명 이상의 인원에게 분배
- 어떤 양을 분배했을 때 K명 이상의 인원에게 분배했다면, 그 이하의 양으로도 K명 이상의 인원에게 분배됨
- 어떤 양을 분배했을 때 K명 미만의 인원에게 분배했다면, 그 이상의 양으로도 K명 미만의 인원에게 분배됨

이상한 술집 BOJ 13702

- 따라서 매개 변수 탐색 적용이 가능하다
- 분배하는 양(매개 변수)에 따라 막걸리를 받는 인원수(결과)가 달라짐
- 분배하는 양에 따라 막걸리를 받는 인원수를 계산하는 함수(결정 함수)를 만들어야 함
- 결정 함수는 매개 변수라는 분배하는 막걸리 양이라는 추가적인 input이 있을 때, 몇 명의 인원에게 막걸리를 분배할 수 있는가? 라는 문제를 풀면 됨

이상한 술집 BOJ 13702

- 더 가혹하게: 결정 함수의 리턴이 실패할 가능성이 높은 방향으로 탐색 범위 조정
- 막걸리 분배하는 양을 늘림
- 더 널널하게: 결정 함수의 리턴이 성공할 가능성이 높은 탐색 범위 조정
- 막걸리 분배하는 양을 줄임
- 어떤 매개 변수에 대한 결정 함수가 조건을 만족한다면 매개 변수로 정답 갱신 후 더 가혹하게
- 만족하지 못했다면 더 널널하게

이상한 술집 BOJ 13702

```cpp
using ll = long long;
ll left = 0;       // 이 문제는 int로 사용해도 됩니다
ll right = (1 << 31) - 1;
ll ans = left;     // 정답 초기화
while (left <= right)
{
    ll mid = (left + right) / 2;
    // check가 결정 함수, mid가 매개 변수
    if (check(mid))
    {
        // mid로 K명에게 분배했으므로 정답 갱신
        ans = max(ans, mid);
        // 가혹한 조건
        left = mid + 1;
    }
    else
    {
        // 실패했으므로 널널한 조건
        right = mid - 1;
    }
}
return ans;
```

이상한 술집 BOJ 13702

```
bool check(ll param){
  ll cnt = 0;
  for (ll a : A){
    cnt += a / param;
    if (cnt >= K) return true;
  }
  return false;
}
```

나무 자르기 BOJ 2805

- 문제

나무 자르기 BOJ 2805

- 절단기의 높이가 높아질수록 집에 가져가는 나무의 양은 어떻게 될까?
- M보다 적은 나무를 집에 가져갔다면 절단기의 높이를 더 높이는 것이 의미 있는가?
- 매개 변수: 절단기의 높이
- 결정 함수: 절단기의 높이에 따라 집으로 가져갈 수 있는 나무의 양
- 결정 함수 리턴: 집으로 가져갈 수 있는 나무의 양과 M을 비교
- 가혹하게: 절단기의 높이를 높여야 한다
- 널널하게: 절단기의 높이를 낮춰야 한다

질문

- NEXT: LIS

LIS

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

예시

15	3	11	7	10	1
X	3	X	7	10	X

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. 둘 중 하나만 기억할 수 있다. 어떤 것이 좋을까?

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

- 2번이 좋아 보인다

무엇이 좋을까?

- 왜냐하면, 최대한 작은 것을 기억해야 다음에 오는 수를 추가할 가능성이 높아지기 때문
- 기억할 배열의 길이가 최대한 길어야 한다
- 기억할 배열의 원소들이 최대한 작아야 한다
- 기억할 배열을 $stack$으로 사용할 것이다

스택의 목표

- 수열 A의 $0 \sim i-1$번째 원소까지 스택으로 어찌어찌 잘 처리 한 상황
- $A[i] > stack.back()$ 이라면 $stack.push\_back(A[i])$
- 최대한 길어져야 하기 때문
- 만약 $A[i] \leq stack.back()$ 이라면?

스택의 목표

- 이때 $stack$의 원소를 작게 만들 수 있다
- $A[i]$이상인 $stack$의 원소 하나를 $A[i]$로 바꾸면 된다. 근데 아무거나?
- 최종적인 LIS가 아닌 $A[i]$를 $stack$에 넣을 시점에 $A[i]$로 만들 수 있는 가장 긴 증가 부분수열을 어떻게 만들까?
- $stack$에서 $A[i]$이상인 원소를 찾는다

스택 부연 설명

- 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

스택의 목표

- $stack$의 원소는 증가할 것이다
- 따라서 Lower bound로 이상을 찾을 수 있다
- 그런 방식으로 스택을 처리해오면 $A[i]$를 $stack$에 넣을 시점에 $stack$은 최대한 길고, 원소들이 최대한 작다
- 따라서 스택의 길이는 LIS의 길이가 된다
- 스택이 LIS 그 자체가 아님을 유의하자

푸는 법

1. 만약에 스택이 비었거나, 현재 배열의 원소가 스택의 마지막 원소보다 크다면 스택에 원소를 $push\_back()$
2. 그렇지 않다면, 현재 배열의 원소를 $target$으로 설정하고 $stack$에서 Lower bound로 바꿀 수 있는 인덱스를 찾고 현재 배열의 원소로 바꿔준다
3. 스택의 길이가 LIS의 길이

예시

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

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

예시

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

Index	0	1	2
Stack	1	3	10

10은 3보다 크다

Index	0	1	2
Stack	1	4	5

10은 5이상이므로 바꿔준다

Index	0	1	2
Stack	1	2	5

4은 2이상이므로 바꿔준다

코드

```
    vector<int> stack;
    for (int a : A) {
        if (stack.empty() || a > stack.back()) stack.push_back(a);
        else {
          int idx = lowerBound(0, stack.size(), a); // 탐색시작, 탐색끝, target
          stack[idx] = a;
        }
    }
    return stack.size(); // 최장 증가 부분 수열의 길이 반환
```

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)

14003 : 가장 긴 증가하는 부분 수열 5

30459 : 현수막 걸기(KUPC)

2121 : 넷이 놀기

14233 : 악덕 사장

1477 : 휴게소 세우기

References

- 이동훈님 블로그
- 2023 AIkon 세미나 자료