#04-4. 이분 탐색

나정휘

https://justiceHui.github.io/

목차

이분 탐색

업/다운 게임

- 철수는 1 이상 N 이하인 자연수 X를 하나 선택한다.
- 영희는 철수가 생각한 X를 맞춰야 한다.
- 영희가 어떤 수 Y를 말하면, 철수는 아래와 같은 대답을 한다.
  - X > Y라면: Up
  - X < Y라면: Down
  - X = Y라면: Accept
- 영희의 질문 횟수를 최소화 시키자.

이분 탐색

영희의 전략

- 정답은 [1, N] 사이에 있음
  - 중간 지점 M_1 = floor((1+N)/2) 선택
  - 정답이 M_1보다 작다면 구간을 [1, M_1-1]로 조정
  - 정답이 M_1보다 크다면 구간을 [M_1+1, N]으로 조정
  - 정답이 존재하는 구간을 [S_2, E_2]라고 하자
- 정답은 [S_2, E_2] 사이에 있음
  - 중간 지점 M_2 = floor((S_2+E_2)/2)를 선택
  - 정답이 M_2보다 작다면 구간을 [S_2, M_2-1]로 조정
  - 정답이 M_2보다 크다면 구간을 [M_2+1, E_2]로 조정
- 반복

이분 탐색

영희의 전략

- 정답이 존재할 수 있는 구간이 매번 절반으로 감소함
  - 최악의 경우에도 $\log_2 N + 1$번의 질문으로 답을 구할 수 있음
- 이게 최적 전략일까?
  - YES
  - 상대자 논증을 이용해 증명할 수 있음

이분 탐색

정렬된 배열 A가 주어진다. K가 A의 몇 번째 원소인지 구해라.

- A = {1, 2, 3, 4, 5, 6, 7}, K = 3
- A = {1, 2, 3, 4, 5, 6, 7}, K = 3
- A = {1, 2, 3, 4, 5, 6, 7}, K = 3

시간 복잡도

- 기본 연산: 배열의 원소와 어떤 수를 비교하는 것
- 기본 연산의 수행 횟수: 최악의 경우 $\log_2 N + 1$번
  - 매번 정답이 존재할 수 있는 구간의 크기가 절반씩 감소
- 따라서 최악의 경우 비교 횟수는 $O(\log N)$

이분 탐색

BOJ 1920. 수 찾기

- N개의 수로 구성된 배열이 주어짐
- X가 주어지면 배열에 값이 X 원소가 존재하는지 판별하는 작업을 M번해야 함

- 배열을 정렬한 다음 매번 이분 탐색

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, M, A[101010];

bool Find(int v){
    int l = 1, r = N;
    while(l <= r){
        int m = (l + r) / 2;
        if(v == A[m]) return true;
        else if(v < A[m]) r = m - 1;
        else l = m + 1;
    }
    return false;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    sort(A+1, A+N+1);
    cin >> M;
    for(int i=1; i<=M; i++){
        int t; cin >> t;
        cout << Find(t) << "\n";
    }
}
```

질문?

결정 문제와 최적화 문제

결정 문제와 최적화 문제

- 결정 문제: YES/NO로 답할 수 있는 문제
  - 가중치가 K 이하인 해가 존재하는가?
- 최적화 문제: 최댓값/최솟값을 구하는 문제
  - 가능한 최소 가중치는 얼마인가?

- 어떤 문제가 더 어려울까?
  - 최적화 문제가 결정 문제가 쉬울 수 없음
  - 최적화 문제를 해결할 수 있으면 무조건 결정 문제를 해결할 수 있음

결정 문제와 최적화 문제

결정 문제와 최적화 문제

- 결정 문제의 예시: 배열 A의 최댓값이 3 이하인지 판별하라.
- 최적화 문제의 예시: 배열 A의 최댓값을 구해라.

- 최적화 문제를 푸는 함수 optimization(A)가 있으면
- 결정 문제를 푸는 함수 decision(A) := optimization(A) ≤ 3

결정 문제와 최적화 문제

최적화 문제를 결정 문제로 바꿔서 해결

- 배열 A의 최댓값을 구하는 최적화 문제 → 배열 A의 최댓값이 K 이하인지 판별하는 결정 문제
- K = 0, 1, 2, ... 에 대해 결정 문제를 해결하면 됨

- 이렇게 보면 더 비효율적이지만...

결정 문제와 최적화 문제

최적화 문제를 결정 문제로 바꿔서 해결

- 배열 A의 최댓값을 구하는 최적화 문제 → 배열 A의 최댓값이 K 이하인지 판별하는 결정 문제
- K = 0, 1, 2, ... 에 대해 결정 문제를 해결하면 됨

- 이렇게 보면 더 비효율적이지만...

- 배열의 최댓값을 10이라고 하자
  - K = 0, 1, 2, ... , 9 일 때는 NO
  - K = 10, 11, 12, ... 일 때는 YES
  - 결정 문제의 답이 NNN...NNYYY...YY 꼴이므로 가장 먼저 등장하는 Y의 위치를 찾으면 됨
  - 이분 탐색

결정 문제와 최적화 문제

최적화 문제를 결정 문제로 바꿔서 해결

- 배열의 최댓값을 10이라고 하자
  - K = 0, 1, 2, … , 9 일 때는 NO
  - K = 10, 11, 12, … 일 때는 YES
  - 결정 문제의 답이 NNN…NNYYY…YY 꼴이므로 가장 먼저 등장하는 Y의 위치를 찾으면 됨

- 이분 탐색
  - 정답이 존재하는 구간 [S, E]와 중간 지점 M = floor((S+E)/2)에 대해
  - decision(M)이 참이면(최댓값이 M 이하이면) 정답은 [S, M]에 존재
  - decision(M)이 거짓이면(최댓값이 M 초과이면) 정답은 [M+1, E]에 존재

질문?

결정 문제와 최적화 문제

매개 변수 탐색 (Parametric Search)

- 조건을 만족하는 최댓값을 구하는 문제
  - 결정 문제의 답이 YYY...YYNNN...NN 꼴일 때 Y가 등장하는 마지막 위치를 구하는 문제
- 조건을 만족하는 최솟값을 구하는 문제
  - 결정 문제의 답이 NNN...NNYYY...YY 꼴일 때 Y가 등장하는 첫 번째 위치를 구하는 문제

- 이분 탐색을 이용해 해결할 수 있음
  - 최댓값을 구하는 문제
    - 정답이 존재하는 구간 [S, E]와 중간 지점 M에 대해
    - decision(M)이 참이면 정답은 [M, E]에 존재
    - decision(M)이 거짓이면 정답은 [S, M-1]에 존재
  - 최솟값을 구하는 문제
    - 정답이 존재하는 구간 [S, E]와 중간 지점 M에 대해
    - decision(M)이 참이면 정답은 [S, M]에 존재
    - decision(M)이 거짓이면 정답은 [M+1, E]에 존재

결정 문제와 최적화 문제

```
int Minimization(){
    int s = 1, e = N;
    while(s < e){
        int m = (s + e) / 2; // floor((s + e) / 2)
        if(Decision(m)) e = m;
        else s = m + 1;
    }
    return e;
}

int Maximization(){
    int s = 1, e = N;
    while(s < e){
        int m = (s + e + 1) / 2; // ceil((s + e) / 2)
        if(Decision(m)) s = m;
        else e = m - 1;
    }
    return s;
}
```

결정 문제와 최적화 문제

BOJ 2805. 나무 자르기

- N개의 나무가 있고, i번째 나무의 높이는 A[i]임
- 높이가 H인 지점에서 절단기를 사용하면 높이가 H 이상인 나무에서 A[i] – H 만큼 가져갈 수 있음
- 나무를 M 이상 가져가기 위해 필요한 높이의 최댓값

- f(h) := h에서 절단기를 사용했을 때 가져가게 되는 나무의 양
- f(h) ≥ M을 만족하는 가장 큰 h를 구하는 문제

결정 문제와 최적화 문제

BOJ 2805. 나무 자르기

- N개의 나무가 있고, i번째 나무의 높이는 A[i]임
- 높이가 H인 지점에서 절단기를 사용하면 높이가 H 이상인 나무에서 A[i] – H 만큼 가져갈 수 있음
- 나무를 M 이상 가져가기 위해 필요한 높이의 최댓값

- f(h) := h에서 절단기를 사용했을 때 가져가게 되는 나무의 양
- f(h) ≥ M을 만족하는 가장 큰 h를 구하는 문제

- h가 증가하면 f(h)는 단조 감소함
- decision(h) := f(h) ≥ M으로 정의하면 결정 문제의 답은 YYY…YYNNN…NN 꼴
- Y가 등장하는 마지막 지점을 구하는 문제

결정 문제와 최적화 문제

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, M, A[1010101];

ll Get(int h){
    ll res = 0;
    for(int i=1; i<=N; i++) res += max(0, A[i] - h);
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> A[i];

    int l = 0, r = 1e9;
    while(l < r){
        int m = (l + r + 1) / 2;
        if(Get(m) >= M) l = m; // M만큼 가져갈 수 있으면 정답은 l 이상
        else r = m - 1; // M만큼 가져갈 수 없으면 정답은 r 미만
    }
    cout << l;
}
```

질문?

결정 문제와 최적화 문제

BOJ 15810. 풍선 공장

- N명의 스태프가 M개의 풍선을 만들어야 함
- i번째 스태프는 A[i]분마다 풍선을 만들 수 있음
- M개의 풍선이 모두 만들어지는 데 걸리는 최소 시간을 구하는 문제

- f(m) := m분 동안 만들 수 있는 풍선의 최대 개수 = sum(floor(m / A[i]))
- f(m) ≥ M을 만족하는 가장 작은 m을 구하는 문제

- m이 증가하면 f(m)은 단조 증가함
- decision(m) := f(m) ≥ M으로 정의하면 결정 문제의 답은 NNN...NNYYY..YY 꼴
- Y가 등장하는 첫 번째 지점을 구하는 문제

결정 문제와 최적화 문제

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, M, A[1010101];

ll Get(ll m){
    ll res = 0;
    for(int i=1; i<=N; i++) res += m / A[i];
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> A[i];

    ll l = 0, r = 1e12;
    while(l < r){
        ll m = (l + r) / 2;
        if(Get(m) >= M) r = m;
        else l = m + 1;
    }
    cout << r;
}
```

결정 문제와 최적화 문제

BOJ 2417. 정수 제곱근

- N이 주어지면 ceil(√N)을 구하는 문제

```cpp
#include <bits/stdc++.h>
using namespace std;
using ull = unsigned long long;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    ull N; cin >> N;
    ull l = 0, r = (1LL << 32) - 1;
    while(l < r){
        ull m = (l + r) / 2;
        if(m * m >= N) r = m;
        else l = m + 1;
    }
    cout << l;
}
```

질문?