10815. 숫자 카드

김인태

1. 문제 접근

시간 제한	메모리 제한	제출	정답	맞힌 사람	정답 비율
2 초	256 MB	38042	18902	13178	48.891%

문제

숫자 카드는 정수 하나가 적혀져 있는 카드이다. 상근이는 숫자 카드 N개를 가지고 있다. 정수 M개가 주어졌을 때, 이 수가 적혀있는 숫자 카드를 상근이가 가지고 있는지 아닌지를 구하는 프로그램을 작성하시오.

입력

첫째 줄에 상근이가 가지고 있는 숫자 카드의 개수 N(1 ≤ N ≤ 500,000)이 주어진다. 둘째 줄에는 숫자 카드에 적혀있는 정수가 주어진다. 숫자 카드에 적혀있는 수는 -10,000,000보다 크거나 같고, 10,000,000보다 작거나 같다. 두 숫자 카드에 같은 수가 적혀있는 경우는 없다.

셋째 줄에는 M(1 ≤ M ≤ 500,000)이 주어진다. 넷째 줄에는 상근이가 가지고 있는 숫자 카드인지 아닌지를 구해야 할 M개의 정수가 주어지며, 이 수는 공백으로 구분되어져 있다. 이 수도 -10,000,000보다 크거나 같고, 10,000,000보다 작거나 같다

출력

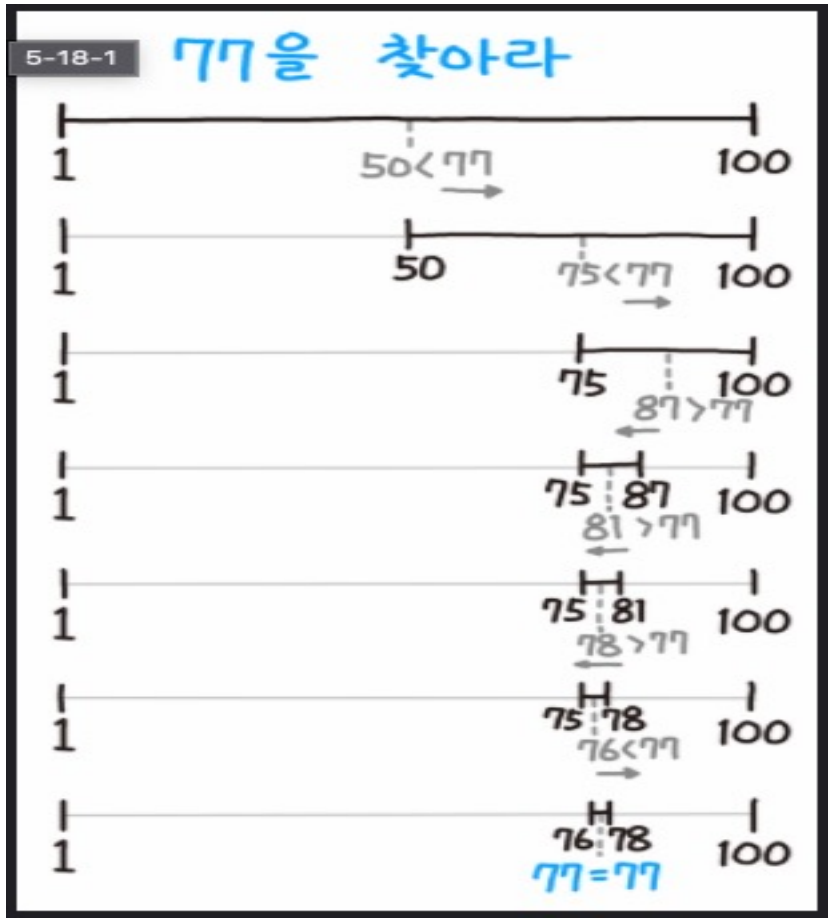
첫째 줄에 입력으로 주어진 M개의 수에 대해서, 각 수가 적힌 숫자 카드를 상근이가 가지고 있으면 1을, 아니면 0을 공백으로 구분해 출력한다.

1. 문제 접근

- 1<= n <= 500,000 (오십만), 1<=m<=500,000
- 완전탐색 이용 시, 시간 복잡도 = O(m*n)
- m = n = 500,000 경우 총 비교 횟수는 250,000,000,000(2천5백억)
- 10억회 연산에 1초라 해도 250초…
- 다른 방법 생각

2. 이분탐색 (이진탐색)

- 정렬된 정수의 리스트를 같은 크기의 두 부분 리스트로 나누고
- 필요한 부분에서만 탐색하도록 제한하여 원하는 원소를 찾는 알고리즘.
- 리스트의 중간 부분에 찾는 원소가 있는지 확인하고,
- 없으면 위쪽에 있는지 아래쪽에 있는지 판단하여 맨 앞부터 검색하거나 중간부터 검색한다.
- 매 탐색마다 탐색의 범위가 절반으로 줄어들기에 시간복잡도 O(logN) 이다.



2-1. 구현

```
function 이진탐색(데이터, 찾는 값)

데이터가 혹시 비어 있는가?
(네) return 찾는 값 없음.

데이터의 가운데 지점을 찾는다.
찾은 지점의 값을 뽑는다.
뽑은 값이 찾는 값인가?

(네) return 뽑은 값.
(아니요)
  뽑은 값과 찾는 값을 비교한다.
  (뽑은 값이 찾는 값보다 큰 값인가?)
    return 이진탐색(데이터 앞쪽 절반, 찾는 값)
  (작은 값인가?)
    return 이진탐색(데이터 뒤쪽 절반, 찾는 값)
```

```java
43    // 이분 탐색
44    public static int bis(int[] arr, int M) {
45        int l = 0;
46        int r = arr.length - 1;
47
48        while(l <= r) {     // 왼쪽과 오른쪽이 겹치면 더이상 탐색안함.
49            int mid = (l+r)/2;
50            if(arr[mid]==M) {   //  값을 자료에서 찾음
51                return 1;
52            }else if(arr[mid] < M) {    //  입력 값이 현재 범위보다 큰 경우, 왼쪽 범위를 축소 시킴.
53                l = mid + 1;
54            }else { // M < arr[mid]     //  입력 값이 현재 범위보다 작은 경우, 오른쪽 범위를 축소 시킴.
55                r = mid - 1;
56            }
57        }
58        return 0;   //  여기까지 도달하면 만족하는 값을 찾지 못한 것.
59    }
```

3. Hash

- 문제 요구 사항을 보면 값이 존재하는지 안 하는지의 여부에만 관심이 있고 그 외에 다른 것은 관심이 없다.
- 입력에 대해 최대한 빠르게 탐색해 결과를 찾는 방법은 Hashing, 시간복잡도 : O(1)

3-1. 구현(HashSet 사용)

```java
public static void main(String[] args) throws NumberFormatException, IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int N = Integer.parseInt(br.readLine());

    HashSet<Integer> set = new HashSet<>();

    StringTokenizer st = new StringTokenizer(br.readLine());
    for(int i = 0; i < N; i++) {
        set.add(Integer.parseInt(st.nextToken()));      // HashSet에 값을 넣어줌.
    }

    int M = Integer.parseInt(br.readLine());
    st = new StringTokenizer(br.readLine());
    StringBuilder sb = new StringBuilder();
    for(int i = 0; i < M; i++) {
        int m = Integer.parseInt(st.nextToken());
        if(set.contains(m))     //  존재하는지 확인
            sb.append(1).append(" ");
        else
            sb.append(0).append(" ");
    }

    System.out.println(sb);
}
```

3-2. 구현(배열 이용)

```java
public static void main(String[] args) throws NumberFormatException, IOException {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

    int N = Integer.parseInt(br.readLine());
    int offset = 10000001;
    boolean[] arr = new boolean[20000001];

    StringTokenizer st = new StringTokenizer(br.readLine());
    for(int i = 0; i < N; i++) {
        arr[Integer.parseInt(st.nextToken()) + offset] = true;
    }

    StringBuilder sb = new StringBuilder();
    int M = Integer.parseInt(br.readLine());
    st = new StringTokenizer(br.readLine());
    for(int i = 0; i < M; i++) {
        if(arr[Integer.parseInt(st.nextToken()) + offset])
            sb.append(1).append(" ");
        else
            sb.append(0).append(" ");
    }

    System.out.println(sb);
}
```

4. 속도 및 교훈

	메모리	시간
이분탐색	108864KB	1320ms
Hashing(HashSet 자료형)	136540KB	1044ms
Hashing(배열 이용)	117728KB	792ms

1. 이분 탐색도 충분히 빠르다! (O(logN))
2. 문제가 무엇을 요구하는지를 좀 더 직관적이고 깊이있게 볼 필요성이 있다. (Hash 기법의 사용)

https://eine.tistory.com/entry/%EC%9D%B4%EC%A7%84-%ED%83%90%EC%83%89-%EC%9D%B4%EB%B6%84-%ED%83%90%EC%83%89binary-search-%EA%B5%AC%ED%98%84%EC%8B%9C-%EA%B3%A0%EB%A0%A4%ED%95%A0-%EA%B2%83%EB%93%A4