2023 성균관대학교 프로그래밍 경진대회

공식 해설

출제

- ✓ 김주원 `faang12594` 수학과 19학번
- ✓ 박세훈 `prarie` 소프트웨어학과 22학번
- ✓ 신정환 `shjohw12` 컴퓨터공학과 16학번
- ✓ 안우솔 `ansol4328` 소프트웨어학과 18학번
- ✓ 오해성 `deuslovelt` 반도체시스템공학과 18학번

검수

- ✓ hibye1217
- ✓ junseo
- ✓ gunwookim
- ✓ jthis
- ✓ mjhmjh1104
- ✓ max804

후원

주최/주관

성균관대학교
SW중심대학사업단

성균관대학교
소프트웨어융합대학

진행

문제		의도한 난이도	출제자
A	증가 배열 만들기	**Very Easy**	박세훈
B	토크나이저	**Easy**	김주원
C	마법 박스	**Easy**	오해성
D	벌레컷	**Medium**	오해성
E	AB	**Medium**	신정환
F	최소 트리 분할	**Medium**	박세훈, 오해성
G	시험	**Medium**	안우솔
H	점프	**Hard**	안우솔
I	양궁	**Hard**	신정환
J	정렬	**Hard**	김주원
K	GCD와 K번째 쿼리	**Challenging**	박세훈

A. 증가 배열 만들기

`ad_hoc, constructive`
출제진 의도 – **Very Easy**

- ✓ 출제자: 박세훈

A. 증가 배열 만들기

- ✓ $N \times M$ 의 2차원 배열을 증가 상태로 만들려면 배열을 어떻게 구성해야 할까요?
- ✓ 먼저 $(1, 1)$ 에 1을 둔 뒤, 아래쪽 또는 오른쪽으로 이동할 때마다 1씩 증가시켜 칸에 채우면 됩니다.
- ✓ 이렇게 하면, 출발점에서 도착점으로 갈 때 항상 칸에 적힌 수가 증가하게 됩니다.
- ✓ 즉, $A_{i,j} = i + j - 1$ 이라고 하면, 2차원 배열이 증가 상태가 됩니다.
- ✓ 이로 인해 $N \times M$ 의 2차원 배열이 증가 상태가 되기 위해 필요한 가장 작은 수는 $N + M - 1$ 이 됩니다.
- ✓ 따라서 $N + M - 1 \leq K$ 를 만족하면 답이 존재함을 알 수 있습니다.

B. 토크나이저

`parsing`, `implementation`
출제진 의도 – **Easy**

- ✓ 출제자: 김주원

B. 토크나이저

- ✓ 명령문을 앞에서부터 순차적으로 읽습니다.
- ✓ 구분자를 만나면, 그 구분자를 기준으로 명령문을 자릅니다.
- ✓ 따라서 $O(N)$에 문제를 해결할 수 있습니다.

C. 마법 박스

`math`, `binary search`
출제진 의도 – **Easy**

- ✓ 출제자: 오해성

C. 마법 박스

- ✓ 특정 수 이하의 수가 박스에 있는지 확인하는 쿼리를 통해 N 범위 내에서 정답을 찾는 문제입니다.
- ✓ 우선 어떤 수 K에 대해 쿼리를 날려서 그 아래 수들이 모두 박스에 있는 것이 확인되면 1부터 K까지 수에는 정답이 존재하지 않기 때문에 20번의 쿼리를 적절히 이용해 정답이 존재하는 구간을 줄여 나갈 수 있습니다.
- ✓ 어렵지 않게 이분 탐색을 사용하면 범위를 매번 절반 씩 줄일 수 있음을 알 수 있습니다.

C. 마법 박스

- ✓ 하지만 단순히 이분 탐색만을 이용하게 되면 20번의 질문 횟수를 초과하기 때문에 문제의 조건하나를 더 활용해야합니다.
- ✓ 문제의 조건 상 어떤 수가 존재하면 그 수에 모든 배수 또한 존재하기 때문에 정답은 무조건 소수임을 알 수 있습니다. 정답이 소수가 아니라면 그 수의 약수 또한 박스 내에 존재하게 되는데 이는 모순이기 때문입니다.
- ✓ 따라서 N 이하의 모든 소수에 대해서만 이분탐색을 활용하면 문제를 해결할 수 있습니다.

D. 벌레컷

`prefix sum`, `two pointer`, `binary search`
출제진 의도 – **Medium**

- ✓ 출제자: 오해성

D. 벌레컷

- ✓ 양의 정수 배열에 대해 문제의 조건을 만족하는 구간 분할의 개수를 구하는 문제입니다.
- ✓ 단순하게는 이중 포문을 통해 문제를 해결할 수 있지만 $O(N^2)$ 이므로 시간초과를 받게됩니다.
- ✓ prefix sum을 활용하면 특정 구간의 합을 $O(1)$ 에 알아낼 수 있습니다.

D. 벌레컷

- ✓ 머리,가슴,배에 해당하는 구간을 prefix sum을 활용한 수식으로 나타내 봅시다.
- ✓ 머리는 Psum[X],가슴은 Psum[Y]-Psum[X],배는 Psum[N] - Psum[Y]가 되고 부등식에 대입하면
- ✓ Psum[X] < Psum[N]-Psum[Y] < Psum[Y] - Psum[x] 이고 수식을 Psum[Y]에 대해 정리하면 (Psum[N] + Psum[X])/ 2< Psum[Y] < Psum[N] - Psum[X] 가 됩니다.
- ✓ X값을 증가시키면서 X가 고정된 상태라면 부등식 또한 좌측과 우측이 상수로 고정되기 때문에 특정 값 이상 특정 값 이하에 해당하는 Y의 개수를 이분탐색 혹은 투포인터를 활용하여 찾아내면 됩니다.

E. AB

`string`, `trie`, `hash`
출제진 의도 – **Medium**

✓ 출제자: 신정환

E. AB

- ✓ trie 자료구조를 이용하여 해결할 수 있습니다. trie 자료구조에 관한 설명은 여기에서 확인하실 수 있습니다.
- ✓ A의 문자열을 관리하는 trie와 B의 문자열을 관리하는 trie를 만듭니다.
- ✓ trie에 저장할 때 add 쿼리의 경우는 값을 1씩 더하고, delete 쿼리의 경우는 값을 1씩 뺍니다.
- ✓ 이때 B의 문자열은 뒤집어서 처리합니다. 그렇게 하면 A의 trie의 각 노드는 문자열이 prefix로 몇 번 등장하였는지를 나타내고, B의 trie의 각 노드는 문자열이 suffix로 몇 번 등장하였는지를 나타냅니다.
- ✓ find 쿼리가 나올 때는 두 trie를 모두 탐색하여(B의 trie는 뒤집어서 탐색) 해당 문자열의 prefix가 A에 등장한 횟수와 suffix가 B에 등장한 횟수를 곱하여 모두 더한 것이 답이 됩니다.
- ✓ 시간 복잡도는 $O(Q \times length\ of\ string)$ 입니다.
- ✓ hashing 등을 이용하여 해결할 수도 있습니다.

F. 최소 트리 분할

`tree`, `graph`, `dfs`, `union find`
출제진 의도 – **Medium**

✓ 출제자: 박세훈, 오해성

F. 최소 트리 분할

- ✓ 가장 큰 A_i 부터 차례대로 봅시다.
- ✓ 현재 가장 큰 A_i 를 x, 그 다음 큰 A_i 를 y 라고 봅시다.
- ✓ 처음에는 아무 노드도 없다고 생각해봅시다.
- ✓ 이후 x 와 같은 가중치를 가지고 있는 노드들을 "활성화"한다고 해봅시다.
- ✓ 이럴 경우 활성화된 노드들로 이루어진 컴포넌트가 생기게 됩니다.
- ✓ 각각의 컴포넌트에 $d = x - y$ 만큼의 연산을 적용합시다.
- ✓ 즉 (컴포넌트의 개수)$\times d$ 만큼의 연산이 필요합니다.
- ✓ 이를 계속해서 모든 노드가 목표 가중치 A_i 가 되도록 하면 됩니다.
- ✓ 컴포넌트의 개수는 disjoint set 자료구조를 통해서 쉽게 관리할 수 있습니다.

F. 최소 트리 분할

- ✓ DFS를 통한 다른 풀이도 존재합니다.
- ✓ 먼저 1 번 노드를 루트로 생각하고 트리를 관찰해봅시다.
- ✓ 어떤 노드 i 의 부모를 노드 par 라고 한다면, 마치 i 번 노드는 par 번 노드에서 사용한 연산의 가중치를 가져올 수 있다고 생각할 수 있습니다.
- ✓ 따라서 맨 처음 루트 노드를 제외하고 $\max(A_i - A_{par}, 0)$ 값을 정답에 더해주면 됩니다.
- ✓ 최종적인 정답은 $A_1 + \sum_{i=2}^{N} \max(A_i - A_{par}, 0)$ 가 됩니다.

G. 시험

`parametric search`, `fraction`

출제진 의도 – **Medium**

✓ 출제자: 안우솔

G. 시험

- ✓ 가장 높은 점수 비율을 갖는 시험을 K 개 고르는 방식은 WA를 받습니다.
- ✓ 어떤 실수 D에 대하여, 평균 성적이 D 이상이 되도록 하는 시험의 조합이 존재하는지 판단해주는 함수가 있다고 가정하면, parametric search를 통해 해결이 가능합니다.
- ✓ 이를 판단할 수 있는 식을 쓰면 아래와 같습니다.

$$\frac{\sum P_i}{\sum Q_i} \geq D \quad (i \in X)$$

- ✓ 이고, 식을 정리하면 다음과 같습니다.

$$\sum(-Q_i D + P_i) \geq 0 \quad (i \in X)$$

- ✓ 즉, $y = -Q_i \times x + P_i$ 라는 직선의 x 좌표에 D 를 대입한 값이 가장 큰 상위 K 개의 직선을 고르는 것이 최적임을 알 수 있습니다.
- ✓ 이 과정은 정렬이나 priority queue 등을 이용해서 $O(N \log N)$ 에 구할 수 있습니다.
- ✓ 따라서 최종 시간 복잡도는 $O(TN \log N)$ 이 됩니다. ($T =$ 충분한 정밀도를 얻기 위한 parametric search의 반복 횟수)

H. 점프

`dynamic_programming`, `segment tree`
출제진 의도 – **Hard**

- ✓ 출제자: 안우솔

H. 점프

- ✓ 가장 먼저 좌표의 값이 너무 크니 좌표 압축을 합니다. 이 과정은 당연히 $O(N \log N)$ 입니다.
- ✓ 좌표를 압축했다면 아래와 같은 점화식을 세울 수 있습니다.
- ✓ $D(i, j) =$ 캐릭터의 x좌표가 j 이고 i 층 이하의 발판 중 가장 높은 발판에 위치하기 위한 최소 점프 횟수
- ✓ $D(1, j) = 0$ (1층에 존재하는 발판 중 j 를 포함하는 발판이 있는 경우)
- ✓ $D(1, j) = \infty$ (otherwise)

H. 점프

- ✓ 이후 $i+1$ 층에 위치한 어떤 발판이 $[L, R]$ 로 표현되는 경우, $L \le x \le R$ 인 x 에 대하여 아래 식이 성립합니다.
- ✓ $D(i+1, x) = 1 + \min\left(D(i, L), D(i, L+1), \cdots, D(i, R)\right)$
- ✓ 이 과정을 naive 하게 구현한 경우 $O(N^2)$ 이 되고 TLE를 받습니다.
- ✓ 구간에 대한 최솟값 쿼리와 구간 업데이트를 효율적으로 하는 자료구조를 도입하면 문제는 해결됩니다.
- ✓ 대표적으로 최솟값 segment tree와 lazy propagation을 사용하는 방법이 존재하고, 이 경우 시간복잡도는 $O(N \log N)$ 이 됩니다.

I. 양궁

`geometry`, `convex_hull`, `binary search`
출제진 의도 – **Hard**

✓ 출제자: 신정환

I. 양궁

- ✓ 주어진 점으로 convex hull을 만들고, convex hull 위의 점을 없애고 다시 convex hull을 만들고, ... 하는 과정을 반복합니다. $O(N^2 \log N)$ 이 소요됩니다.
- ✓ 점이 볼록 다각형 내부에 있는지 판단하는 것은 로그 시간에 가능합니다. (여기 참조)
- ✓ 모든 볼록 다각형에 대해 그러한 과정을 반복한다면 최악의 경우 총 $O(Q \times NlogN)$ 이 소요되어 시간 초과를 받게 됩니다.
- ✓ 점이 P_i 내부에 있다면 P_{i-1} 내부에도 있다는 점을 착안하면, 이분 탐색을 적용할 수 있다는 것을 알 수 있습니다.
- ✓ 시간 복잡도는 $O(N^2 \log N + Q(\log N)^2)$ 입니다.
- ✓ 사실 이 문제는 N 이 큰 경우에도 해결 가능합니다. convex layers를 구성하는 것을 빠르게 하면 됩니다. (여기, 여기 참조)

J. 정렬

`sorting`, `ad_hoc`, `constructive`, `divide_and_conquer`
출제진 의도 – **Hard**

- ✓ 출제자: 김주원

J. 정렬

- ✓ $O(N^2)$의 성능을 가진 버블정렬, 삽입정렬, 선택정렬부터 그보다 빠른 퀵정렬, 병합정렬, 힙정렬, 셸정렬 등 다양한 정렬 알고리즘이 존재합니다.
- ✓ 그리고 우리는 c++ sort(vector.begin(), vector.end()), python list.sort()를 이용하여 매우 편리하게 정렬할 수 있습니다.
- ✓ 하지만 그 정렬 과정을 우리는 얼마나 잘 이해하고 있을까요?
- ✓ 널리 알려진 훌륭한 정렬 알고리즘을 들여다보며 우리는 이 문제의 해결을 위한 실마리를 얻을 수 있습니다.
- ✓ 이 문제에서는 퀵정렬과 병합정렬로부터 분할 정복의 아이디어를 차용합니다.

J. 정렬

- ✓ N 개의 원소를 퀵정렬과 병합정렬처럼 두 개의 데이터 블록으로 나누어 정렬하는 해결책은 $N \log_2 N$ 에 비례한 수의 연산을 이용합니다. 이러한 해결책은 10 만개의 수를 200 만개보다 많은 연산을 이용하므로, 유효한 해결책이 아닙니다.
- ✓ 병합정렬을 조금 더 확장해서 N 개의 수를 3 개의 데이터 블록으로 나누고, 각각의 데이터 블록을 정렬한 후에 합치는 해결책이 있습니다.
- ✓ 이러한 해결책은 $N \log_3 N$ 에 비례한 수의 연산을 이용하고, 10 만개의 수를 최대 175 만 개의 연산을 이용하여 해결합니다.

K. GCD와 K번째 쿼리

`mo's algorithm`, `sqrt decomposition`, `preprocessing`, `binary search`, `math`, `sparse_table`, `segment_tree`
출제진 의도 – **Challenging**

- ✓ 출제자: 박세훈

- ✓ L이 고정되어 있는 상태에서 $L \leq R \leq N$인 모든 R에 대해 서로 다른 $\gcd(A_L, A_{L+1}, \cdots, A_R)$의 값은 총 몇개가 나올까요?
- ✓ R이 증가할 $\gcd$값이 변한다면, $\gcd$의 값이 최소 2배 이상 감소하므로 서로 다른 수는 $O(\log A_{MAX})$개 나올 수 있습니다.
- ✓ 이를 통해 L이 고정되어 있을 때 R값이 증가하면 나올 수 있는 ($\gcd$, 개수)를 전처리할 수 있습니다.
- ✓ 반대로, R이 고정되어 있을 때 L이 감소할 때 나올 수 있는 값도 전처리할 수 있습니다.

- ✓ 어떻게 전처리할 수 있을까요?
- ✓ 먼저 $\gcd(A_i, A_{i+1}, \cdots, A_j)$ 를 빠르게 구할 수 있는 방법을 생각해봅시다.
- ✓ 구간의 $\gcd$ 를 구하는 방법에는 여러 가지가 있습니다.
- ✓ 1. GCD Segment tree를 사용하면 됩니다.
- ✓ 2. Sparse Table을 사용하면 됩니다.
- ✓ 무엇을 사용해도 괜찮습니다. 하지만, Sparse Table을 사용할 경우 구간의 $\gcd$ 를 $O(\log A_{MAX})$ 에 구할 수 있습니다. 세그먼트 트리의 경우 $O(\log^2 A_{MAX})$ 이 걸려 더 느릴 수 있습니다.

- ✓ 그럼 이제 고정된 L에 대해서 나올 수 있는 (gcd, 개수)를 전처리합시다.
- ✓ 먼저 나올 수 있는 서로 다른 gcd의 개수는 총 $O(\log A_{MAX})$ 개 있었으므로, L에 대해서 어떤 R에서 구간 gcd가 바뀌는지 찾으면 됩니다.
- ✓ 이는 이분 탐색으로 쉽게 찾을 수 있습니다.
- ✓ 이를 서로 다른 gcd의 개수만큼 반복한 뒤, (gcd, 개수)를 전처리하면 됩니다.
- ✓ 모든 L에 대해 전처리하기 위해 걸리는 시간은 $O(N \log N \log^2 A_{MAX})$ 만큼 걸립니다.
- ✓ 이를 반대로 R을 고정한 뒤, L이 감소할 때 나올 수 있는 값들도 전처리하면 됩니다.

- ✓ 이제 쿼리는 어떻게 처리하면 될까요?
- ✓ 현재 (L, R)의 쿼리에서 $(L, R+1)$의 쿼리로 움직일 때 관리해야할 값을 관찰해봅시다.
- ✓ K번째 값을 찾아야 하므로 구간을 움직일 때마다 $\gcd$의 값 i의 개수 C_i를 관리한다고 합시다.
- ✓ 이미 우리는 고정된 L에 대해 나올 수 있는 $\gcd$값을 전처리했으므로, R이 한 칸 증가할 때 C 값을 $O(\log A_{MAX})$에 관리할 수 있습니다.
- ✓ 이렇게 되면, 구간 쿼리를 mo's algorithm을 통해 빠르게 처리할 수 있습니다.
- ✓ mo's algorithm을 사용하면, C 값을 각 쿼리마다 빠르게 관리할 수 있습니다.
- ✓ C 값만 관리한다고 하면, 전체 시간 복잡도는 $O(N\sqrt{N}\log A_{MAX})$이 걸리게 됩니다.

- ✓ 그런데 A_{MAX} 값이 $500\,000$ 이므로 K 번째 값을 C 를 통해서 빠르게 찾기는 힘들어 보입니다.
- ✓ 이 부분을 개선하기 위해 C 값을 $\sqrt{A_{MAX}}$ 개로 쪼개서 CS_i 는 $[i\sqrt{A_{MAX}}, (i+1)\sqrt{A_MAX})$ 에서 C 값의 합을 관리한다고 합시다.
- ✓ mo's algorithm으로 쿼리를 처리할 때 CS 값을 C 값을 관리하는 거랑 비슷하게 $O(\log A_{MAX})$ 에 처리할 수 있습니다.
- ✓ 이렇게 CS 를 관리하면, K 번째 값을 $O(\sqrt{A_{MAX}})$ 값에 찾을 수 있습니다.
- ✓ K 번째 값을 Q 번 찾아야 하므로, $O(Q\sqrt{A_{MAX}})$ 의 시간이 걸립니다.

- ✓ 결과적으로, $O(N \log N \log^2 A_{MAX} + N\sqrt{N} \log A_{MAX} + Q\sqrt{A_{MAX}})$에 전체 문제를 해결할 수 있습니다.
- ✓ 실제로는 $\gcd$ 값이 엄청 많이 변할 순 없기 때문에 매우 빠르게 도는 것을 확인할 수 있습니다.

✓ 감사합니다.