코드포스 9주차 문제 풀이

문제 난이도

	문제 번호	문제 이름	난이도
A	1802	종이 접기	S1
B	11582	치킨 TOP N	S4
C	18222	투에-모스 문자열	S2
D	21870	시철이가 사랑한 GCD	G5
E	2339	석판 자르기	G1
F	22958	안산 탐지기	G1
G	16885	벡터의 합	P2
H	4645	힐베르트 곡선	P3

A. 종이 접기

애드 혹, 분할 정복

A. 종이 접기

- ✓ **주어진 문자열을 동호의 규칙대로 접을 수 있는지 확인**:
  - ✓ while 루프를 통해 문자열이 길이가 1이 될 때까지 반복합니다.
    - ✓ 만약 어떤 i번째 위치에서 s[i]와 s[length-1-i]가 같다면, 이는 동호의 규칙대로 접힌 것이 아니므로 ans를 false로 설정하고 루프를 종료합니다.
    - ✓ 문자열의 길이가 1이 되면 루프를 종료합니다.
  - ✓ 매번 문자열을 절반으로 줄여서 업데이트 합니다.

- ✓ **결과 출력**:
  - ✓ 각 테스트 케이스마다 결과를 "YES" 또는 "NO"로 출력합니다.

- ✓ **시간 복잡도:** O(NlogN)

B. 치킨 TOP N

정렬, 분할 정복

B. 치킨 TOP N

- ✓ **정렬 및 출력**:
  - ✓ 각 회원이 담당할 구간의 크기 gap을 계산합니다. 이는 n / k입니다.
  - ✓ gap 크기만큼의 구간을 순차적으로 정렬합니다.
    - ✓ 첫 번째 구간은 arr[0]부터 arr[gap-1]까지입니다.
    - ✓ 두 번째 구간은 arr[gap]부터 arr[2*gap-1]까지입니다.
    - ✓ 이러한 방식으로 마지막 구간까지 순차적으로 정렬합니다.
  - ✓ 각 구간이 정렬될 때마다 해당 구간의 요소를 출력합니다.

- ✓ **시간 복잡도:** $O\left(N \log\left(\frac{N}{k}\right)\right)$

C. 투에-모스 문자열

분할 정복, 재귀, 비트마스킹

C. 투에-모스 문자열

- ✓ **이진수 변환 및 비트 카운트**:
  - ✓ k를 이진수로 변환합니다.
  - ✓ k의 이진수 표현에서 1의 개수를 셉니다.
- ✓ **출력 결정**:
  - ✓ k의 이진수 표현에서 1의 개수가 홀수이면 1을 출력하고, 짝수이면 0을 출력합니다.
- ✓ **시간 복잡도:** O(logK)

D. 시철이가 사랑한 GCD

수학, 브루트포스 알고리즘, 정수론, 분할 정복, 재귀, 유클리드 호제법

D. 시철이가 사랑한 GCD

- **분할 정복 함수 구현(두 개의 값 반환):**
  - 함수는 배열 arr의 구간 [left, right)에서의 최대 아름다움을 계산합니다.
  - 기본 종료 조건: 구간이 한 원소일 경우 해당 원소를 반환합니다.
  - 구간을 절반으로 나누어 재귀적으로 왼쪽과 오른쪽 구간의 결과를 계산합니다.
  - 왼쪽 구간의 결과, 오른쪽 구간의 결과를 따로 저장합니다.
  - 두 구간의 GCD를 계산합니다(첫 번째 값).
  - 두 가지 경우를 고려하여 최댓값을 계산합니다(두 번째 값):
    - 왼쪽 구간의 첫 번째 값(GCD) + 오른쪽 구간의 두 번째 값 (GCD선택 X)
    - 왼쪽 구간의 두 번째 값(GCD 선택 X) + 오른쪽 구간의 첫 번째 값(GCD)
  - 이 최대값과(두 번째 값) 왼쪽과 오른쪽 원소들의 GCD 값(첫번째 값)을 둘 다 반환 합니다.

- **최종 결과 계산**:
  - 전체 배열에 대해 dfs(0, n)을 호출하여 결과를 계산합니다.
  - 결과 중 최댓값을 출력합니다.

- **시간 복잡도:** O(NlogN)

E. 석판 자르기

분할 정복

E. 석판 자르기

- ✓ **2차원 구간 합 배열 생성**:
  - ✓ jewelrySUM(보석 누적 합)과 impuritySUM(불순물 누적 합) 배열을 사용하여 각 위치까지의 보석 결정체와 불순물의 개수를 저장합니다.
  - ✓ 이 배열들은 구간 합을 빠르게 계산하기 위해 사용됩니다.

- ✓ **분할 정복 함수 구현:**
  - ✓ dfs(x1, y1, x2, y2, cut) 함수는 (x1, y1)에서 (x2, y2)까지의 영역을 분할합니다.
  - ✓ 현재 구간 내의 보석 결정체와 불순물의 개수를 계산합니다.
  - ✓ 불순물이 없고 보석이 하나만 있으면 가능한 조합으로 카운트합니다.
  - ✓ 그렇지 않다면, 가로와 세로 방향으로 재귀적으로 분할을 시도합니다.
  - ✓ 이 때 불순물이 있고 보석이 없는 줄을 선택해 분할해야 합니다.
  - ✓ Cut 변수는 현재 분할 방향(가로 또는 세로)을 나타내며, 연속된 동일 방향의 분할을 방지합니다.

- ✓ **최종 결과 계산 및 출력**:
  - ✓ 전체 석판에 대해 DFS를 두 번 호출합니다(초기 분할 방향이 가로와 세로).
  - ✓ 결과를 출력합니다. 가능한 조합이 없으면 -1을 출력합니다.

F. 안산 탐지기

애드 혹, 분할 정복

F. 안산 탐지기

- ✓ **초기 설정**:
  - ✓ 입력으로 봉우리의 개수 N을 읽어옵니다.
  - ✓ 첫 번째 질문을 통해 전체 봉우리 중 가장 높은 봉우리의 높이를 알아냅니다.

- ✓ **분할 정복 함수 구현**:
  - ✓ dfs(start, gap, tri) 함수는 탐지할 시작 봉우리, 간격, 종료 조건을 인자로 받습니다.
  - ✓ battery는 탐지할 봉우리의 개수를 정합니다.
  - ✓ 탐지기를 사용하여 지정된 범위 내에서 최대 높이를 구하고, 그 결과를 ans에 저장합니다.
  - ✓ 만약 탐지된 최대 높이와 같다면, 해당 범위 내에 가장 높은 봉우리가 있는 것입니다.
  - ✓ 탐지할 봉우리의 범위를 점차 좁히는 방식으로 가장 높은 봉우리를 찾습니다.
  - ✓ 재귀적으로 탐색 범위를 줄여 나가며, 탐지기의 사용 횟수와 배터리의 사용 개수를 관리합니다.
  - ✓ 다음 dfs함수에서 탐지할 봉우리 수를 계산하여 2개 이하이면 tri=true입니다.
  - ✓ tri = true이거나 battery = 1인데 최대 봉우리가 탐지되었다면 그 때가 위치를 알아낸 시점입니다.

G. 벡터의 합

정렬, 기하학, 스위핑, 분할 정복

G. 벡터의 합

- ✓ **벡터 입력 및 초기 설정**:
  - ✓ 먼저, N개의 벡터를 입력 받습니다.
  - ✓ 각 벡터를 1사분면으로 넣기 위해 연산을 합니다. 이 때의 연산 번호를 cmd로 저장합니다.
  - ✓ 절대값으로 변환한 좌표와 원래 인덱스, cmd 값을 구조체 Point로 저장합니다.

- ✓ **벡터 정렬**:
  - ✓ 절대값으로 변환한 좌표를 기준으로 정렬합니다. x 좌표를 기준으로 정렬하고, x 좌표가 같으면 y 좌표를 기준으로 정렬합니다.

G. 벡터의 합

- ✓ **최솟값 탐색(스위핑)**:
  - ✓ 정렬된 벡터들 중에서 인접한 벡터들을 비교하여 최솟값을 탐색합니다.
  - ✓ dist 변수는 두 벡터의 거리 제곱의 최솟값을 저장합니다.
  - ✓ 이중 루프를 사용하여 인접한 벡터 쌍을 비교합니다. 외부 루프는 현재 벡터를 고르고, 내부 루프는 그 이전의 벡터들을 비교합니다.
  - ✓ 두 벡터 arr[i]와 arr[j]의 거리 제곱을 계산하여 dist보다 작으면 갱신합니다.
  - ✓ 이 때 두 벡터의 x좌표의 거리 제곱이 dist보다 크면 내부 루프에서 빠져나옵니다.

- ✓ **결과 출력**:
  - ✓ 최종적으로 선택된 두 벡터의 인덱스와 적용된 변환 형태를 출력합니다. 변환 형태는 cmd 값을 5에서 뺀 값(반대로 뒤집기)을 사용하여 출력합니다.

H. 힐베르트 곡선

구현, 다이나믹 프로그래밍, 분할 정복, 재귀

H. 힐베르트 곡선

- ✓ **분할 정복 함수 구현:**
  - ✓ 이 함수는 주어진 깊이와 길이, 좌표 범위에 대해 힐베르트 곡선과 교차하는 점의 수를 계산합니다.
  - ✓ 수평선의 경우와 수직선의 경우를 따로 처리합니다.
  - ✓ 수직선 또는 수평선이 그림의 반절 부분에 걸쳐 있다면 반절 부분을 기준으로 두 부분으로 나누어 처리합니다.
  - ✓ 수직선 또는 수평선이 그림의 반절 부분을 지나고 있는 경우 예외 처리를 해줘야 합니다. (4번 부분)
  - ✓ 이미 계산된 결과가 있으면 저장된 값을 반환합니다. 이는 중복 계산을 피하고 성능을 향상시킵니다.(DP 활용)