나는 코더다 입부시험 문제 풀이

ICQT 2023

by

나는 코더다 39기 및 40기

문제		의도한 난이도	출제자
A1	옷걸이	**Easy**	ehlee0815
A2	다이어트	**Easy**	ehlee0815
A3	공룡 게임	**Medium**	roynine
A4	낱말 퍼즐	**Medium**	gs22059
A5	외계 분자	**Medium**	denniskim
A6	정화조	**Hard**	ehlee0815
A7	성벽 쌓기	**Challenging**	gs22123

A1. 옷걸이

`constructive`
출제진 의도 - **Easy**

- 제출 243회, 정답 67명
- 처음 푼 사람: bnb2011, 3분
- 출제자: **ehlee0815**

부분 문제 1 (23점)

- 옷걸이의 종류가 1 또는 2 밖에 없는 경우입니다.
- 이 경우 1의 개수가 U 개고 2의 개수가 D 개라면, 모든 옷을 걸 수 있습니다. 이때는 1에 상의를, 2에 하의를 걸면 됩니다.
- 만약 그렇지 않다면, 어떻게 하더라도 옷을 걸 수 없습니다.

부분 문제 2 (77점)

- 만약 1의 개수와 3의 개수의 합이 U 개보다 많다면, 하의를 모두 걸 수 없습니다.
- 만약 2의 개수와 3의 개수의 합이 D 개보다 많다면, 상의를 모두 걸 수 없습니다.
- 위 두 경우가 모두 아닌 경우에는 항상 옷걸이에 모든 옷을 걸 수 있습니다.
- 먼저, 1에는 상의를 2에는 하의를 겁니다. 이후, 3에 나머지 상의와 하의를 모두 걸어주면 됩니다.
- 시간복잡도는 $\mathcal{O}(N)$ 입니다.

A2. 보디빌딩

`greedy`
출제진 의도 - **Easy**

- 제출 135회, 정답 28명
- 처음 푼 사람: bnb2011, 7분
- 출제자: **ehlee0815**

- 이 문제는 두 가지의 greedy 알고리즘 풀이 방법이 존재합니다. 먼저 출제자의 의도를 설명하겠습니다.
- C_i 라는 새로운 배열을 정의하겠습니다. 이는 건표가 마지막 날까지 쓰러지지 않고 살기 위해서 i 번째 날에 가져야 하는 최소 몸무게를 의미합니다.
- 먼저 $C_N = A_N$ 이라는 사실을 알 수 있습니다. A_N 이상의 몸무게를 가진다면 건표는 살겠고, 그렇지 않는다면 쓰러질 것이기 때문입니다.
- 또한 $C_i = max(A_i, C_{i+1} - B_{i+1})$ 이라는 점화식을 얻을 수 있습니다. $i+1$ 번째 날에는 B_{i+1} 만큼 살이 찐 뒤, 그 몸무게가 C_{i+1} 이상이어야 하기 때문입니다.

- 그다음, i 번째 날의 몸무게를 나타내는 수열 D_i 를 정의해봅시다.
- 먼저, $D_i = D_{i-1} + B_i$ 일 것입니다.
- 이후 $C_i \leq D_i - X$ 일 때까지 D_i 에서 X 를 빼주고, D_{i+1} 을 업데이트하면 됩니다.
- 이 빼주는 과정을 나눗셈으로 잘 처리해주면 만점을 받을 수 있습니다.
- 만약 $D_i < C_i$ 이면 -1을 출력해주면 됩니다.

- 또다른 풀이가 있습니다.
- 어떤 날에 루틴을 진행할 수 있다고 가정해봅시다. 그럼 그 날에 루틴을 굳이 진행하지 않고, 마지막 날에 몰아서 한꺼번에 루틴을 진행할 수도 있습니다.
- 따라서, $\left\lfloor \frac{B_1 + B_2 + \ldots + B_N - A_N}{X} \right\rfloor$ 가 답이 됩니다.
- 단, $B_1 + B_2 + \ldots + B_i < A_i$ 가 되는 i 가 존재한다면 -1을 출력해주면 됩니다. 이는 누적합으로 계산해도 되고, 스위핑하면서 업데이트해도 됩니다.
- 어떤 방법을 이용하든 $\mathcal{O}(N)$ 만에 답을 구할 수 있습니다.

A3. 공룡 게임

dp
출제진 의도 - **Medium**

- 제출 41회, 정답 8명
- 처음 푼 사람: bnb2011, 22분
- 출제자: **roynine**

부분 문제 1 (10점)

- 옷걸이 문제랑 같은 논리입니다. 1에서는 점프만 하면 되고, 2에서는 슬라이딩만 하면 됩니다. 다르게 액션을 취하여 주어진 장애물을 모두 넘는 방법은 없습니다.
- 따라서 주어진 장애물을 모두 넘을 수 있다는 것은 임의의 두 장애물 1 사이의 시간 차가 모두 A 이상이고, 임의의 두 장애물 2 사이의 시간 차가 모두 B 이상이라는 것과 필요충분조건 관계에 있습니다.
- 만약 장애물을 모두 넘을 수 있다면, 장애물 1의 개수 cnt_1 과 장애물 2의 개수 cnt_2 에 대해, $Xcnt_1 + Ycnt_2$ 가 최소 비용이 됩니다.

부분 문제 2 (20점)

- $D_{i,j,k}$ 를 i 번째 장애물까지 넘었을 때 가장 마지막으로 점프한 시각이 j 초이고 가장 마지막으로 슬라이딩한 시각이 k 초인 경우의 최소비용을 나타낸다고 해봅시다.
- 그럼 $D_{i,j,k}$ 다음으로 전이할 수 있는 상태를 생각해봅시다. 먼저, $i+1$ 번째 장애물에서 점프하는 경우, $D_{i+1,T_{i+1},k}$ 인 상태로 전이합니다. 만약, $i+1$ 번째 장애물에서 슬라이딩하는 경우, $D_{i+1,j,T_{i+1}}$ 인 상태로 전이합니다.
- $i+1$ 에서 점프하려면 j 와 T_{i+1} 의 차이가 A 이상이고, $i+1$ 번째 장애물이 1 또는 3 이어야 함에 유의합시다. 슬라이딩할 때도 쿨타임과 장애물의 종류를 잘 확인하여 주어진 시각에 슬라이딩이 가능한지 판별해 줍시다.

- 만약 $i+1$ 번째 장애물에서 점프했다면 $D_{i+1,T_{i+1},k} = \min(D_{i+1,T_{i+1},k}, D_{i,j,k} + X)$ 로, 슬라이딩했다면 $D_{i+1,j,T_{i+1}} = \min(D_{i+1,j,T_{i+1}}, D_{i,j,k} + Y)$ 로 dp를 업데이트해주면 됩니다.
- 그렇다면, 답은 $\min_{1 \le j \le T_i}(D_{N,T_N,j}, D_{N,j,T_N})$ 가 됩니다.
- 초기에 모든 i, j, k 에 대해 $D_{i,j,k}$ 를 매우 큰 하나의 정수 INF로 초기화시켜준 뒤, $\min_{1 \le j \le T_i}(D_{N,T_i,j}, D_{N,j,T_i})$ 값이 마지막까지 INF을 유지하고 있다면 -1를 출력해 줍시다.
- 아, 참! $D_{0,0,0}$ 을 0으로 초기화하는 것을 잊지 마세요. 검수자의 구현 팁으로 말씀드리자면, 0번째 장애물을 -INF로 설정하면 초기 상태를 쉽게 표현할 수 있습니다.
- 시간복잡도는 $\mathcal{O}(NT^2)$ 입니다.

부분 문제 3 (30점)

- 부분 문제 2에서 j, k로 가능한 값은 $T_1, T_2, \ldots, T_N$ 밖에 없습니다. 이를 이용해서 dp를 살짝 수정해줍시다.
- $D_{i,j,k}$를 가장 마지막으로 점프로 넘은 장애물이 j번째 장애물이고, 가장 마지막으로 슬라이딩으로 넘은 장애물이 k번째 장애물인 경우의 최소 비용이라고 정의합시다.
- 이렇게만 바꿔주고 부분 문제 2처럼 업데이트해주면 $\mathcal{O}(N^3)$이 됩니다.

I'm

만점 풀이

- 부분 문제 3의 $D_{i,j,k}$에서 j, k 중 하나는 무조건 i라는 관찰을 할 수 있습니다. 결국에는 마지막 장애물인 i번째 장애물에서 점프하면 j가 i, 슬라이딩하면 k가 i가 될테니까요.
- 따라서 $D_{i,j,0}$을 부분 문제 3의 $D_{i,i,j}$로, $D_{i,j,1}$을 부분 문제 3의 $D_{i,j,i}$로 대응해주면, $\mathcal{O}(N^2)$짜리 dp가 만들어집니다.
- 추가로 이 문제는 백준의 2618번 경찰차 문제랑 비슷한 문제입니다. 꼭 한 번 풀어보시기 바랍니다.

A4. 낱말 퍼즐

ad-hoc
출제진 의도 – **Medium**

- 제출 14회, 정답 4명
- 처음 푼 사람: aeren, 40분
- 출제자: **gs20059**

부분 문제 1 (20점)

- ✓ Q가 충분히 작기 때문에, 1번 쿼리가 주어질 때마다 각 조각을 돌리면 됩니다.
- ✓ 배열의 원소를 그 배열의 중심에 대칭되는 원소와 바꿔주시면 됩니다.
- ✓ 참고로, 이때 1번 쿼리의 시간복잡도는 쿼리당 $\mathcal{O}(NM)$ 입니다.

I'm

부분 문제 2 (80점)

- ✓ Q가 좀 많이 크기 때문에, 1번 쿼리를 $\mathcal{O}(1)$이나 $\mathcal{O}(logNM)$에 해결해야 합니다.
- ✓ 풀이에 들어가기 전에, 실제로 배열을 몇 번만 돌려 봅시다.
- ✓ 쿼리 1 2 2, 쿼리 1 3 3 이 순서대로 들어간 모습입니다.

```
abcde.........gfjih.........yuvwx
fghij.........baedc.........eabcd
klmno.........vuyxw.........jfghi
pqrst.........qptsr.........oklmn
uvwxy.........lkonm.........tpqrs
```

- 가장 오른쪽에 있는 배열을 보면, 원래 배열을 오른쪽 아래로 한 칸 민 것처럼 생겼습니다.
- 이로써, 쿼리 1이 짝수 번 실행된 후의 배열은 초기 상태를 민 상태임을 추측할 수 있습니다.
- 홀수 번 실행된 후는 최초 배열의 점대칭 상태를 민 모습을 띠고 있습니다.
- 이렇게 해서 최초의 맨 왼쪽 위 칸이 어딨는지만 알면 나머지 배열을 채울 수 있습니다.

```
abcde.....gfjih.....yuvwx
fghij.....baedc.....eabcd
klmno.....vuyxw.....jfghi
pqrst.....qptsr.....oklmn
uvwxy.....lkonm.....tpqrs
```

- 이게 진짜 되는지 증명해 봅시다.
- 우선 배열을 미는 게 정확히 무엇인지 확실히 하고 갑시다.
- '배열을 (a,b) 만큼 밀었다'라는 것은 기존에 `arr[i][j]`에 있던 원소가 `arr[(i+a)%n][(j+b)%m]`로 가는 것을 이야기합니다. 0-index를 전제로 합니다.
- 이는 `arr[n-a][m-b]`를 기준으로 배열을 쿼리 1번에서와 같이 네 조각으로 자른 뒤, 왼쪽 위 조각은 오른쪽 아래로, 오른쪽 위 조각은 왼쪽 아래로 옮기는 것과 같은 작업입니다.

- 배열 돌리기를 한 번 하면 무슨 일이 일어날까요?
- (a,b)에서 배열 돌리기를 한 번 시행한다고 합시다.
- 그 상태에서, 시행에 의해 잘린 네 조각을 기준으로 해 배열을 밀어봅시다. 왼쪽 위의, (1,1)부터 (a,b)까지 조각은 오른쪽 아래, 오른쪽 아래 조각은 왼쪽 위...와 같이 이동하게 됩니다.
- 이때 배열의 상태는, 최초의 전체 배열을 180도 회전한 결과와 같게 됩니다!

- 이 과정을 그려보면 다음과 같습니다. (2,3)을 기준으로 돌려 봅시다.
- abcde.....hgfji.....yxwvu
  fghij.....cbaed.....tdrqp
  klmno.....wvxyx.....onmlk
  pqrst.....rqpts.....jihgf
  uvwxy.....mlkon.....edcba
- 첫번째 배열을 180도 회전하면 3번째 배열이 됩니다.
- 따라서, 우리는 각 배열 회전을 전체 배열을 회전한 뒤 이를 미는 것으로 처리할 수 있습니다!

- 왜 배열 회전을 배열을 미는 것으로 처리했을까요?
- 배열을 밀어놓은 상태는 배열을 실제로 밀지 않고도 빠르게 출력할 수 있기 때문입니다!
- 최초 배열에서 `(a,b)`에 있던 칸이 현재 `(1,1)`으로 밀려 왔다면, $0 \leq i \leq n-1$ 및 $0 \leq j \leq m-1$의 모든 `i,j`에 대해 `arr[(a+i)%n][(b+j)%m]`를 출력하면 됩니다.
- 따라서, 우리는 1번 쿼리가 주어질 때마다 최초 위치가 `(1,1)`이었던 것이 어디로 이동하는지만 저장해 두면 됩니다.
- 1번 쿼리가 짝수번 주어진 뒤 2번 쿼리가 나온다면 처음 배열을 밀어서, 홀수번 주어졌다면 처음 배열을 돌린 배열을 밀어서 배열을 출력할 수 있습니다.

- 마지막으로 시간 복잡도를 구해 봅시다.
- 배열을 회전할 때마다 임의의 위치에 있는 칸이 어디로 이동하는지는, 자명하게도 쿼리당 $\mathcal{O}(1)$에 구할 수 있습니다.
- 그와 별개로, 2번 쿼리는 최대 75번 주어지게 되고 각 쿼리에서 출력되는 문자의 수는 최대 nm개입니다.
- 전체 시간복잡도는 $\mathcal{O}(75nm)$입니다. 시간복잡도 표기상 상수를 표기하는 것은 옳지 않으나, 이 문제에서는 상수가 크므로 따로 표기해두었습니다.

A5. 외계 분자

`segtree`, `lazyprop`, `hashing`
출제진 의도 – **Medium**

- 제출 18회, 정답 3명
- 처음 푼 사람: aeren, 60분
- 출제자: **denniskim**

I'm

부분 문제 1 (10점)

- ✓ $1 \leq Q \leq 20$ 입니다.
- ✓ 그냥 문제에서 시키는대로 구현하면 됩니다.
- ✓ 시간복잡도는 $\mathcal{O}(Q(|S| + \sum_{i=1}^{n} |T_i|))$ 입니다.

부분 문제 2 (30점)

- 1번 질의가 주어지지 않습니다.

- hash[i]를 1부터 i까지의 문자를 해싱한 값이라고 정의합시다.
- hash[i] = $\sum_{k=1}^{i} S[k] * p^k \bmod M$ (M는 소수)
- 위 배열과 그들의 누적합을 구하면, l부터 r까지의 문자를 해싱한 값을 모듈러 역원을 사용하여 빠르게 구할 수 있습니다.
- 모든 문자열 T_i에 대해 해싱한 값을 먼저 구해놓고, l부터 r까지의 문자를 해싱한 값과 같은 값이 있는지를 구하면 됩니다. 이 과정은 set 등을 사용하여 빠르게 처리할 수 있습니다.
- 시간복잡도는 $\mathcal{O}(|S| + \sum_{i=1}^{n} |T_i| + Q(logN + logM))$ 입니다.

부분 문제 3 (30점)

- 1번 질의에서 $l = r$ 입니다.

- 세그먼트 트리를 사용해서 구간의 해시값을 빠르게 구합시다.
- $H[i] = S[i] * p^i \bmod M$ 라고 정의합시다.
- $s \sim e$ 의 구간을 담당하는 노드의 값을 $\sum_{k=s}^{e} H[k]$ 라고 정의합시다.
- 세그먼트 트리를 만들었으면, 마찬가지로 l 부터 r 까지의 문자를 해싱한 값을 구간 합 쿼리와 모듈러 역원을 사용하여 빠르게 구할 수 있습니다.
- 비교는 마찬가지로 set 등으로 처리합니다.
- 시간복잡도는 $\mathcal{O}(|S|log|S| + \sum_{i=1}^{n} |T_i| + Q(log|S| + logN + logM))$ 입니다.

부분 문제 4 (30점)

- 추가 제한 조건이 없습니다.
- 부분 문제 3에서 사용한 세그먼트 트리를 lazy로 바꿔주면 됩니다.
- 길이 len의 같은 문자만 있는 문자열의 해싱값을 미리 전처리해두면 lazy처리를 하기 편해집니다.
- 시간복잡도는 $\mathcal{O}(|S|log|S| + \sum_{i=1}^{n} |T_i| + Q(log|S| + logN + logM))$ 입니다.

A6. 정화조

`tree`, `dp`, `convex_hull_trick`, `smaller_to_larger`, `segment_tree`, `offline_queries`
출제진 의도 - **Challenging**

- 제출 1회, 정답 0명
- 처음 푼 사람: -, -분
- 출제자: **ehlee0815**

- 부분 문제를 떠나서 이 문제에서 요구하는 핵심적인 관찰 몇 가지만 해봅시다.
- D_i 를 i 번째 정화조에서 정화하여 0등급수를 얻기까지 걸리는 최소 비용으로 정의합시다. 만약 그러한 방법이 존재하지 않는다면 매우 큰 수 INF 를 넣어줍니다.
- 그렇다면 $D_i = \min_j(D_j + C_i(dep_j - dep_i)) = \min_j(D_j + C_i dep_j) - C_i dep_i$ 가 됩니다. 여기서 dep_i 는 트리의 루트로부터의 거리라고 두겠습니다. 즉, 나가는 간선이 없는 정화조와 i 번째 정화조 사이의 경로 중 간선의 개수라고 정의하겠습니다.
- D_i 점화식이 cht 꼴이네요. 좋습니다. 이젠 이 문제에서 가장 처리하기 까다로운 조건을 고려해보겠습니다.

I'm

부분 문제 1 (10점)

- 앞 장의 점화식에서 j로 가능한 후보들을 생각해 봅시다. 이는 i번째 정화조의 subtree에 존재하는 정화조 중 $l_i \leq dep_j - dep_i \leq r_i$을 만족하는 정화조라고 생각해볼 수 있습니다.
- 이 j를 어떻게 구하느냐가 부분 문제의 난이도를 결정해준다고 보면 되겠습니다.
- 부분 문제 1의 경우, D_i를 업데이트할 때마다 모든 j를 다 탐색하면서 j가 i의 subtree 내에 존재하는지와 $l_i \leq dep_j - dep_i \leq r_i$를 만족하는지를 확인하면 됩니다.
- Euler Tour Technique을 이용합시다. 즉, dfs ordering을 이용하여 자신의 subtree 정점들을 하나의 구간으로 표현해봅시다. 이를 통하여 j가 i의 subtree 내에 존재하는지를 $\mathcal{O}(1)$에 탐색할 수 있고, dp 업데이트하는데 걸리는 시간은 총 $\mathcal{O}(N^2)$입니다.

- 그럼 쿼리는 어떻게 처리할까요?
- X 번 정점에 도달했을 때, K 이하의 수질을 얻으려면 결국 X 의 subtree에 존재하는 정화조 j 중 $dep_j - dep_X \leq K$ 를 만족하는 j 에서 정화가 한 번 이상 일어나야 합니다.
- 또한 만약 j 에서 정화했다면 더이상 다른 정화조에서 정화하여 비용을 낭비할 필요가 없습니다.
- 따라서 위 조건을 만족하는 j 들 중에 D_j 의 최솟값을 구하여 출력하면 답이 됩니다. 참고로, 그 최솟값이 INF인 경우에는 -1을 출력해야 합니다.
- 이 역시 똑같이 모든 j 에 대하여 탐색하면서 j 가 X 의 subtree에 있는지, 그리고 $dep_j - dep_X \leq K$ 가 만족하는지를 검사해주면 됩니다.
- 시간복잡도는 $\mathcal{O}(N(N+Q))$ 입니다.

부분 문제 2 (30점)

- ✓ dep_i 는 N 을 넘을 수 없습니다. 따라서, 부분 문제 2는 i 의 subtree에 존재하는 모든 정화조가 j 의 후보가 됩니다.
- ✓ 아까 언급했던 dp 점화식은 cht를 사용하여 하나의 D_i 값을 $\mathcal{O}(\log N)$ 에 계산할 수 있습니다.
- ✓ cht에 대한 자세한 내용은 이곳에서 언급하지 않겠습니다.

- segment tree 하나를 형성합시다. segment tree의 한 정점이 $s \sim e$ 까지를 관할한다면, 이 정점은 dfs ordering의 번호가 $s \sim e$ 인 정화조의 직선들의 lower envelope을 가지고 있습니다. 즉, dfs ordering 번호가 s 이상 e 이하인 모든 정화조 i 에 대하여 $D_i + x \times dep_i$ 라는 일차함수들의 lower envelope를 가지고 있습니다.
- 한 직선은 최대 $\mathcal{O}(\log N)$ 개의 노드에만 관여하므로 공간복잡도는 $\mathcal{O}(N \log N)$ 이 됩니다.
- dfs ordering 번호가 l_i 이상 r_i 이하인 직선들의 $x = C_i$ 에서의 함숫값의 최솟값은 segment tree의 query 연산을 통해 $\mathcal{O}(\log^2 N)$ 에 구현할 수 있습니다. 이는 어떤 노드가 $s \sim e$ 를 관할하고 있고, $l \leq s$ 이며, $e \leq r$ 이라면 그 노드에 저장하고 있는 직선의 함숫값 중 최솟값을 이분 탐색으로 계산할 수 있기 때문입니다.

- ✓ 따라서, 자신의 subtree를 Euler Tour Technique을 이용하여 하나의 구간으로 대응시키고, 이 구간에 존재하는 convex hull에서 함숫값의 최솟값을 구해주면 $\mathcal{O}(N \log N + Q \log^2 N)$ 만에 dp 업데이트와 쿼리 처리를 실행할 수 있습니다.
- ✓ lower envelope을 업데이트하는 과정에서 li chao tree를 쓰면 $\mathcal{O}((N+Q) \log N \log C)$ 가 되지만, 그렇게 추천드리는 방법은 아닙니다.

부분 문제 3 (60점)

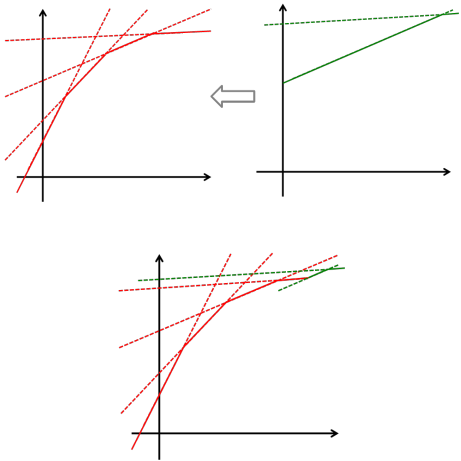
- 전체적인 풀이의 틀은 아래와 같습니다.
- 정화조마다 segment tree를 만듭시다. 다만, 이때 정화조 i에 부여된 segment tree에서는 한 정점이 $s \sim e$까지를 관할할 때, dep가 $s \sim e$이면서 i의 subtree에 해당하는 정화조의 직선들의 lower envelope를 가지고 있습니다.
- dep 별로 lower envelope를 만들어 이를 segment tree로 관리해준다면, cht를 이용해 $l_i \leq dep_j \leq r_i$이면서 i의 subtree에 속하는 모든 j들에 대해 $D_j + C_i dep_j$의 최솟값을 $\mathcal{O}(\log^2 N)$ 계산할 수 있게 됩니다.

- 이젠 우리는 두 가지의 문제에 직면하게 됩니다.
- 먼저 각 정점마다 자신의 subtree에 포함된 정화조들의 직선들을 모두 담게 된다면, 정점마다 $\mathcal{O}(N)$ 개의 직선들을 담고 있는 segment tree를 만들게 되고, 따라서 총 공간복잡도는 $\mathcal{O}(N^2 \log N)$ 이 됩니다.
- 이뿐만 아니라 각 직선은 최대 $\mathcal{O}(N)$ 개의 노드에 삽입하게 되므로, 시간 복잡도상으로도 $\mathcal{O}(N^2 \log N)$ 이 되어 시간 초과를 받게 됩니다.
- 하지만, 놀랍게도 이 과정을 dynamic segment tree와 smaller to larger technique을 이용해 $\mathcal{O}((N+Q) \log^2 N)$ 으로 만들 수 있습니다.

- 어떤 정화조가 두 개의 자식 정화조가 있다고 합시다. 그럼 그 정화조의 segment tree는 자신의 두 자식 정화조에 저장되어 있는 직선들과 자신의 직선을 모두 저장하게 될 것입니다.
- 여기서 두 자식의 정화조 중 저장되어 있는 직선의 개수가 적은 segment tree를 직선의 개수가 많은 segment tree로 넣어 주면, smaller to larger technique을 이용해서 시간을 줄이는 것이 가능하다고 생각해 볼 수 있습니다.
- 각 직선마다 최대 $\mathcal{O}(\log N)$ 개의 segment tree에 삽입될 수 있고, 하나의 segment tree에 하나의 직선을 삽입할 때마다 $\mathcal{O}(\log N)$ 개의 노드(convex hull)를 건드리게 됩니다.

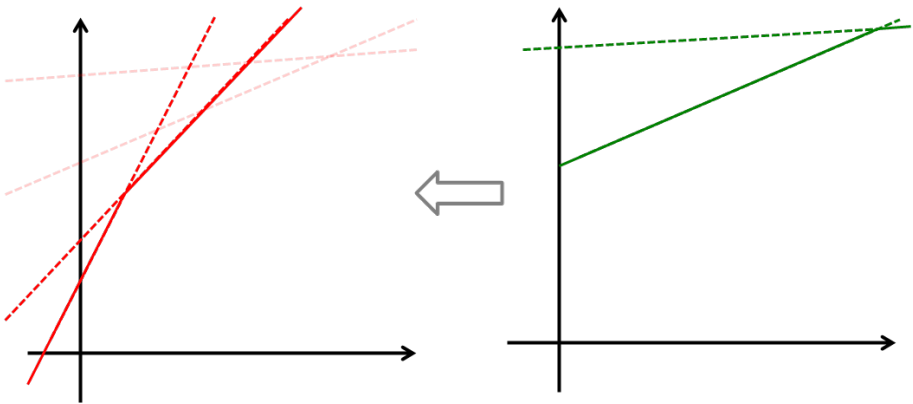
- 이젠 하나의 노드(convex hull)에 직선을 $\mathcal{O}(1)$ 만에 삽입할 수 있다면, 각 직선마다 segment tree 삽입에 최대 $\mathcal{O}(\log^2 N)$ 의 시간복잡도가 걸리고, 모든 직선을 삽입하는데는 최대 $\mathcal{O}(N \log^2 N)$ 의 시간 복잡도가 됩니다.
- 하지만, cht에 저장되는 직선의 형태는 $D_i + x \times dep_i$ 으로 두 segment tree를 merge할 때 dep_i 가 단조성을 가지지 않아 일반적인 방법으로는 삽입하기 어려워 보입니다.
- 만약 기존의 convex hull에 직선을 그대로 삽입하게 된다면, 다음 슬라이드와 같이 잘못된 cht가 만들어집니다.
- 이를 해결하기 위해 li chao tree를 노드에 박게 된다면, 한 노드에 직선을 삽입하는 과정이 $\mathcal{O}(\log C)$ 가 되어 segment tree 구성에 총 $\mathcal{O}(N \log^2 N \log C)$ 가 소요됩니다.

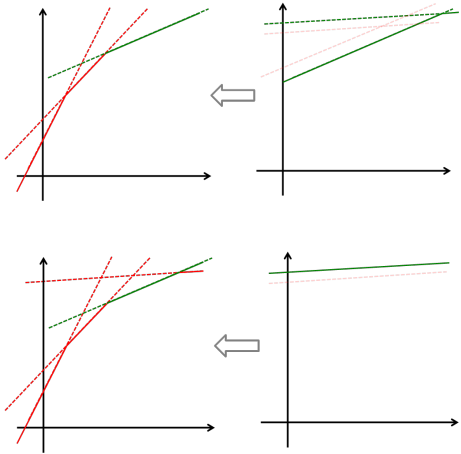
A6. 정화조



- 그럼 smaller to larger를 사용할 수 없게 되는 건가요?
- 그렇지만은 않습니다. 삽입 과정을 잘 처리해준다면 amortized $\mathcal{O}(1)$에 직선을 삽입할 수 있습니다. 그러기 위해서는 두 가지 포인트를 짚고 넘어가야 합니다.
- 첫 번째는 smaller to larger 과정에서 작은 집합의 크기만큼 연산한다면 $\mathcal{O}(\log N)$이 보장된다는 당연한 사실입니다.
- 두 번째는 직선의 기울기가 dep_i라는 점입니다! 이 두 번째 사실이 smaller to larger를 가능하게 만듭니다.

- 먼저 큰 집합에 삽입된 직선들 중 작은 집합에 삽입된 직선들의 기울기보다 작거나 같은 기울기를 가진 직선들을 모두 제거해줍니다.
- 이때 기울기는 dep이고 이는 1씩 늘어나고 줄어드는 연속성이 있기 때문에, **큰 집합에 제거된 직선들의 크기는 작은 집합의 크기보다 작거나 같습니다.**
- 이후 제거된 직선과 작은 집합에 있는 직선 중에서 기울기가 같은 두 직선끼리 묶은 다음, 그중 y 절편이 작은 직선들을 선택합니다.
- 이렇게 선택된 직선을 큰 집합에 기울기가 큰 순서대로 삽입해주면 $\mathcal{O}(1)$에 한 직선을 한 노드에 삽입할 수 있습니다.





- 따라서, 시작할 때 리프마다 먼저 segment tree를 구현해준 뒤, 위상정렬 순서대로 D_i를 업데이트하면서 segment tree를 merge해주면 됩니다.
- 위 방법으로 하면 모든 직선을 최대 $\mathcal{O}(N \log^2 N)$에 삽입할 수 있습니다.
- 한 정화조는 smaller to larger technique을 이용하여 합쳐준 뒤, 쿼리를 부분 문제 2와 같이 처리해주면 됩니다.
- 이때, 쿼리를 오프라인 쿼리로 처리하여 X가 같은 쿼리끼리 모아 한꺼번에 처리해줍니다.
- 따라서 시간복잡도는 최대 $\mathcal{O}((N+Q) \log^2 N)$입니다.

I'm

- 마지막으로 리프마다 segment tree를 만들면 공간복잡도가 $\mathcal{O}(N^2 \log N)$ 가 됩니다.
- 그렇지만, 다시 이야기해서 한 직선은 최대 $\mathcal{O}(\log N)$ 개의 segment tree에 삽입될 수 있습니다.
- 그렇기에 필요한 정점만 직접 생성해주는 dynamic segment tree를 통하여 $\mathcal{O}(N \log N)$ 의 공간복잡도로 해결할 수 있습니다.
- dynamic segment tree 안에 lower envelope를 넣어야 하는 셈이라 구현이 굉장히 복잡합니다. 화이팅하시길 바랍니다.

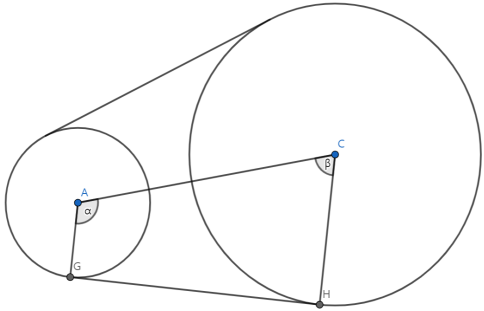
A7. 성벽 쌓기

`geometry`, `convex_hull`, `divide_and_conquer`, `implementation`
출제진 의도 – **Challenging**

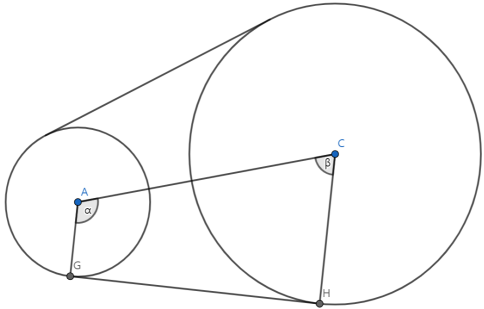
- 제출 1회, 정답 0명
- 처음 푼 사람: -, -분
- 출제자: **gs22123**

부분 문제 1 (1점)

- 원이 2개 밖에 안 주어집니다.
- 그렇기 때문에 두 원의 중심의 좌표와 반지름을 가지고 볼록 껍질의 둘레를 계산할 수 있습니다.
- 또한, 한 원이 다른 원에 포함 되는지를 미리 확인하고 예외 처리해야 합니다.
- 두 원의 좌표를 $(x_1, y_1), (x_2, y_2)$ 라 하고, 반지름의 길이를 $r_1, r_2 (r_1 \leq r_2)$ 라 합시다.



- $d = \overline{AC} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- $\alpha = \arccos \dfrac{r_1 - r_2}{d}$
- $\beta = \arccos \dfrac{r_2 - r_1}{d}$



- ✓ $l = \overline{GH} = d \sin \beta$
- ✓ $\therefore ans = 2l + 2r_1\beta + 2r_2\alpha$

부분 문제 2 (1점)

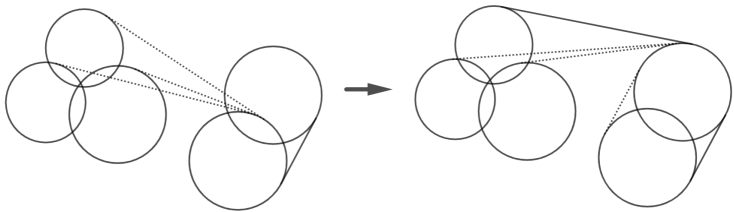
- 주어지는 모든 원의 반지름이 같습니다.
- 이 부분 문제는 백준 7420번 맹독 방벽과 같습니다
- 원의 반지름이 전부 같기 때문에 볼록 껍질에 있는 접선들은 원의 중심의 볼록 껍질과 평행하며 그 거리는 반지름 r 만큼 떨어져 있습니다.
- 그래서 원의 중심의 볼록 껍질의 둘레의 길이를 구한 뒤, 원의 호는 따로 계산해주면 됩니다.
- 원의 호 부분을 다 모으면 원 한바퀴가 나온다는 것을 알 수가 있습니다.
- 그래서 답은 원의 중심들의 볼록 껍질의 둘레와 반지름 r 짜리 원의 둘레를 더하면 구할 수 있습니다.

부분 문제 3 (총 9점)

- 주어지는 원의 개수가 100개 이하입니다.
- 그렇기 때문에 웬만한 시간복잡도는 통과할 수 있을 것입니다.
- 여러가지 풀이가 있겠지만, 여기서는 시간복잡도 $O(Nh)$ 풀이를 설명하겠습니다.
- 여기서 h는 볼록 껍질에 포함된 호의 수입니다.

- 점들의 볼록 껍질을 구하는 알고리즘 중에 Gift Wrapping Algorithm, 혹은 Jarvis March라 불리는 알고리즘이 있습니다.
- 이 알고리즘을 간단하게 설명하면 현재 볼록 껍질 위에 있는 점이랑 모든 다른 점들을 각각 이었을 때, 나머지 점들이 그 선분의 왼쪽(혹은 오른쪽)에만 놓이는 점을 찾습니다.
- 그렇게 찾은 점은 볼록 껍질에 포함이 되는 점이며, 현재 점을 이렇게 찾은 점으로 옮기고, 볼록 껍질이 완성될 때까지 반복합니다.
- 이제 이 알고리즘을 원에다가 똑같이 적용하면 됩니다.

- 우선 볼록 껍질에 포함되는 원 하나를 찾을 때는 가장 낮은 점을 포함하는 원을 찾으면 됩니다 (가장 높은 점, 가장 왼쪽에 있는 점 등도 상관없습니다).
- 이후 점일 때랑 똑같이 현재 원에서 다른 모든 원에 공통외접선을 그었을때, 다른 모든 원이 왼쪽에 오는 원을 찾아서 진행하면 됩니다.



- 이 알고리즘은 다음 원을 찾는데 $O(N)$의 시간이 걸리고, 이를 볼록 껍질에 포함된 호마다 반복하므로, 시간복잡도는 $O(Nh)$입니다.
- 여담으로, 일반적으로 원들이 랜덤하게 놓여있을 때 볼록 껍질에 포함되는 호의 수가 그렇게 많지 않아서, 이 풀이만으로 거의 모든 테스트 케이스의 답을 시간 초과 없이 구할 수 있습니다.
- 사실, 이 방법 말고도 원주에 수천 개의 점을 찍은 다음, 이들의 convex hull의 둘레를 구해도 맞긴 합니다.

만점 풀이 (100점)

- 추가적인 제한이 없습니다.
- 볼록 껍질을 $O(NlogN)$에 구할 수 있는 알고리즘이 필요합니다.
- 본 문제는 David Rappaport의 A convex hull algorithm for discs and applications을 구현한 문제임을 밝힙니다.
- 위 논문 또는 https://equinox134.github.io/posts/convex-hull-of-circles/에 자세한 풀이가 적혀 있으니 참고해주세요

- 우선 몇가지 기호들을 정의합시다.
- $Hull(S)$: S의 볼록 껍질
- $\partial(R)$: R의 경계, 즉, $\partial(Hull(S))$는 S의 볼록 껍질의 경계
- $H(L)$: L이 방향이 있는 선일 때, L의 오른쪽 반평면
- $S \in H(L)$이고 $H(L)$의 어떠한 부분 집합도 S를 포함하지 않으면 L은 S의 $support\ line$이라 한다.
- $L(A, B)$: A에서 B로 향하는 A와 B의 공통 $support\ line$(존재한다면)
- $t(A, B)$: $L(A, B)$의 부분 집합인 선분으로, 한쪽 끝은 $\partial(A)$, 다른 쪽은 $\partial(B)$에 있다.

- ✓ $a, b \in S$일 때, $L(a, b)$가 $Hull(S)$의 $support\ line$이면 $t(a, b)$를 $Hull(S)$의 $edge$라 한다.
- ✓ $\partial(Hull(S))$는 $s \in S$인 s들의 배열로 나타낸다. 즉, $CH(S) = (s_0, s_1, \cdots, s_h)$이다. 이때 하나의 원이 $\partial(S)$에 여러번 나타날 수 있으므로 $i \neq j$이고 $s_i = s_j$일 수 있다. 또한, $s_0 = s_h$를 유지한다(처음 원을 마지막에도 추가함).
- ✓ $p \in P, q \in Q$에 대해서, $L(p, q)$가 $CH(P \cup Q)$를 $support$하면 $t(p, q)$를 $CH(P)$와 $CH(Q)$의 $bridge$라 한다.

- 알고리즘이 시간 내에 돌아가기 위해서는 $CH(S)$의 크기는 $O(N)$이어야 합니다. 이를 증명합시다.
- $u, v \in S$에 대해서 $CH(S)$를 $S_0, u, S_1, v, S_2, u, S_3, v, S_4$로 표현합시다.
- 여기서 S_i는 $CH(S)$의 부분집합입니다.
- 만약 S_1과 S_3의 크기가 0이라면 $t(u, v)$가 $CH(S)$의 $edge$로 2번 나타난다는 것이데, 이는 말이 안됩니다.
- 만약 S_1의 크기가 0이 아니라면, $L(u, v)$는 $Hull(S)$과의 어떠한 점 ψ에서 만나야 합니다.
- $CH(S)$에서 나타는 두 번째 $u, \cdots, v$는 $L(u, v)$가 ψ가 아닌 다른 점에서 만나거나, 안 만나야 한다는 것을 의미하는데, 이는 모순입니다.

- 즉, $CH(S)$ 에서는 $u, \cdots, v, \cdots, u, \cdots, v$ 는 나타날 수 없습니다.
- 이런식으로 $u, \cdots, v, \cdots, u, \cdots, v$ 꼴이 나타날 수 없는 수열을 Davenport-Schinzel Sequence of Order 2라고 합니다.
- 이러한 수열의 길이는 $2n-1$ 을 넘을 수 없습니다.
- 결론적으로 $CH(S)$ 의 크기는 $2n-1$ 이하입니다.

- 다음으로는 몇가지 함수들을 정의합니다.
- 두 방향 있는 선 L_1, L_2에 대해, $\alpha(L_1, L_2)$는 L_1에서 L_2까지의 시계방향 각도를 반환한다.
- $s \in CH(P)$에 대해 $succ(s)$는 $CH(P)$에서 s 다음으로 오는 원소를 반환한다. s가 마지막 원소이면 첫 원소를 반환한다.
- P와 Q의 평행한 $support\ line$을 각각 L_p, L_q라 할 때, $H(L_q) \in H(L_p)$이면 $dom(L_p, L_q)$는 참, 아니면 거짓을 반환한다.
- 어떤 배열 ϑ와 원소 ψ가 주어졌을때, $add(\vartheta, \psi)$는 ϑ의 마지막 원소가 ψ이면 ϑ를, 안 그러면 ψ를 ϑ에 마지막에 추가하고 ϑ를 반환한다.

- 볼록 껍질을 구하는 알고리즘은 분할 정복을 사용하여 구합니다.
- 즉, 우선 S를 크기가 최대 1 차이나는 두 집합 P와 Q로 나눕니다.
- 그 다음으로는 재귀적으로 $CH(P)$와 $CH(Q)$를 구합니다.
- 마지막으로 두 볼록 껍질을 합쳐 $CH(S) = CH(P \cup Q)$를 구합니다.
- 이제 두 볼록 껍질을 합치는 법에 대해 설명하겠습니다.

- 우선 $CH(S)$ 는 길이 0으로 초기화합니다.
- 또, 어떠한 선 L^* 에 평행하면서 $p \in P, q \in Q$ 에 접하는 P 와 Q 의 $support\ line$ L_p 와 L_q 를 정합니다.
- $Advance$ 는 P 와 Q 중 어떤 집합의 다음 원으로 넘어가게 하는 함수로, 나중에 설명합니다.
- 이제 아래의 과정을 $CH(S)$ 가 완성될 때까지 반복합니다:
- 만약 $dom(L_p, L_q)$ 라면 $add(CH(S), p)$ 를 한 후, $Advance(L^*, p, q)$ 를 합니다.
- 그렇지 않다면 $add(CH(S), q)$ 를 한 후, $Advance(L^*, q, p)$ 를 합니다.
- 이후 L_p 를 L^* 와 평행하며 P 와 p 에서 접하는 $support\ line$, L_q 를 L^* 와 평행하며 Q 와 q 에서 접하는 $support\ line$ 으로 바꿉니다.

- 이제 $Advance(L^*, x, y)$ 함수에 대해서 설명하겠습니다 (x 와 y 는 원).
- $Advance$ 함수는 간단하게 $t(x, y)$ 와 $t(y, x)$ 가 두 볼록 껍질을 연결하는지 (즉, $bridge$ 인지) 를 확인하고, x, 혹은 y 를 다음 원으로 옮깁니다.
- 우선 4개의 변수 a_1, a_2, a_3, a_4 를 구합니다:
- $a_1 = \alpha(L^*, L(x, y))$, $a_2 = \alpha(L^*, L(x, succ(x)))$, $a_3 = \alpha(L^*, L(y, succ(y)))$, $a_4 = \alpha(L^*, L(y, x))$
- 이때 $L(x, y)$ 가 존재하지 않으면 $\alpha(L^*, L(x, y))$ 는 정의되지 않습니다.
- 만약 $a_1 = min(a_1, a_2, a_3)$ 라면 $t(x, y)$ 가 $bridge$ 이므로, $add(CH(S), y)$ 를 합니다.
- 이 상태에서 $a_4 = min(a_2, a_3, a_4)$ 라면 $t(y, x)$ 도 $bridge$ 이므로, $add(CH(S), x)$ 를 합니다.

- 이제 x 나 y 를 다음 원으로 옮길 차례입니다.
- 만약 $a_2 < a_3$ 라면 L^* 를 $L(x, succ(x))$ 로, x 를 $succ(x)$ 로 바꿉니다.
- 그렇지 않다면 L^* 를 $L(y, succ(y))$ 로, y 를 $succ(y)$ 로 바꿉니다.
- 이상으로 알고리즘에 대한 설명을 마치겠습니다.

- 이제 알고리즘의 정당성을 증명합시다.
- 우선 두 볼록 껍질을 합치는 과정에서 L_p와 L_q가 평행하며, 각각 P와 Q를 $support$함을 보입시다.
- 처음 L_p와 L_q는 위 조건을 만족하도록 설정했습니다.
- 임의의 iteration에서 L_p와 L_q가 위 조건을 만족할 때, 다음 iteration에서도 위 조건을 만족한다는 것을 보이면 됩니다.
- L_p와 L_q가 매번 평행한 것은 자명합니다.

- $CH(P)$(혹은 $CH(Q)$)에서 p와 연속한 원들을 생각해보면, 어떤 선이 $L(pred(p), p)$와 $L(p, succ(p))$ 사이에 있으면 p에 접하는 P의 $support\ line$이 될 수 있습니다.
- 이때 $Advance$에서 L_p와 L_q 중 하나는 연속한 원의 공통접선으로 바뀝니다. 편의상 바뀌는 선을 L_q라 합시다. 그러면 L_q는 $L(q, succ(q))$로 바뀝니다.
- 바뀌는 선은 각도가 더 작은 걸로 바뀌기 때문에 $L(q, succ(q))$는 $L(p, succ(p))$보다 작고, 그러므로 L_p의 각도도 $L(p, succ(p))$보다 작습니다. 즉, L_p는 P를 $support$합니다.
- 결론적으로, 어떤 iteration에서 L_p, L_q가 위 조건을 만족하면, 다음 iteration에서도 만족하므로, L_p와 L_q는 두 볼록 껍질을 합치는 과정에서 위 조건을 항상 만족합니다.

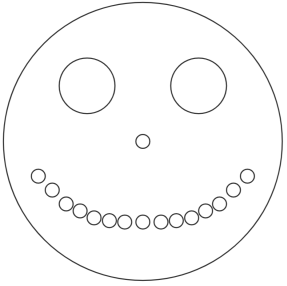
- 다음으로는 $dom(L_p, L_q)$ 가 참일 때, $a_1 = min(a_1, a_2, a_3)$ 일 때만 $L(p, q)$ 가 $bridge$ 가 되고, $L(q, p)$ 는 $t(p, q)$ 가 $bridge$ 이면서 $a_4 = min(a_2, a_3, a_4)$ 일 때만 $bridge$ 가 됨을 보입시다.
- 먼저 저희는 P 는 L_p, $L(p, succ(p))$, Q 는 L_q, $L(q, succ(q))$ 에 의해 $support$ 됨을 압니다.
- a_1 이 최소가 아니라고 가정합시다.
- a_1 이 최소가 아니라는 것은 $L(p, succ(p))$ 와 $L(q, succ(q))$ 중에서 $L(p, q)$ 보다 L^* 와의 각도가 작은 선이 있다는 것입니다.
- 이는 다른 말로 $L(p, q)$ 의 왼쪽 영역에 $CH(P)$ 나 $CH(Q)$ 에 해당하는 점이 있다는 뜻이고, 그렇기 때문에 $L(p, q)$ 는 $bridge$ 가 될 수 없습니다.
- 결론적으로 a_1 이 최소일 때만 $L(p, q)$ 가 $bridge$ 가 됩니다.
- 같은 방식을 사용하면 $L(q, p)$ 에 대해서도 보일 수 있습니다.

- 마지막으로 두 볼록 껍질을 합치는 과정이 $CH(S) = CH(P \cup Q)$ 를 제대로 구함을 보입시다.
- $CH(S)$ 의 $edge$ 들은 $CH(P), CH(Q)$ 의 $edge$ 이거나 둘의 $bridge$ 일 것입니다.
- 합치는 과정에서 $CH(P)$ 와 $CH(Q)$ 의 모든 $edge$ 를 확인하므로, $CH(S)$ 의 $edge$ 가 될 수 있는 $edge$ 들을 모두 확인함을 알 수 있습니다.
- $bridge$ 의 경우에는 합치는 과정에서 평행한 $support\ line$ 이 있는 모든 $p \in P, q \in Q$ 쌍을 확인합니다.
- 각 쌍마다 $t(p, q)$, 그리고 $t(q, p)$ 의 2개의 $bridge$ 가 생길 수 있는데, 이는 바로 전에 $Advance$ 에서 제대로 구함을 보였습니다.
- 결론적으로 앞에서 설명한 두 볼록 껍질을 합치는 알고리즘은 $CH(S) = CH(P \cup Q)$ 를 제대로 구합니다.

- 진짜 마지막으로 시간복잡도를 분석해 봅시다.
- 이 알고리즘은 다음 점화식을 만족합니다: $T(N) = 2T(N/2) + Cost(Merge)$
- 여기서 $Merge$는 두 볼록 껍질을 합치는 알고리즘으로, $O(N)$에 작동함을 알 수 있습니다.
- 고로, 이 알고리즘의 시간복잡도는 $O(NlogN)$입니다.

I'm

✓ 수고하셨습니다.



이 문제 테스트 케이스 中