제 2회 웰노운컵 Day 1 풀이

N과 M

- Euler's theorem: a, n이 서로소면 a^phi(n) 을 n으로 나눈 나머지는 1이다.
  - 여기서 phi(n) = n * \prod_{p: n의 소인수} (1-1/p) 이다
- 문제의 답을 f(n, m)이라 하자. 만약 n과 m이 서로소라면 f(n, m) = n^(f(n, phi(m)) mod m이다.
- 재귀적으로 계산. 두 가지 방법이 있다.
- #1: n과 m에 대해 m의 소인수분해에서 n의 소인수들로 이루어진 부분 m1과 그렇지 않은 부분 m2로 나눠 따로 계산
  - m1, m2가 서로소이므로 f(n, m1)과 f(n, m2)에서 Chinese Remainder Theorem으로 f(n, m) 계산 가능
- #2: a, n이 서로소가 아니더라도 N1, N2가 충분히 크고 phi(n)으로 나눈 나머지가 같다면 a^N1 == a^N2 mod n
  - CRT를 안 쓰고 구현 가능

Build a Wall!

중요한 것은 삼각분할 후 준원이의 집을 둘러싸게 되는 삼각형이다.

삼각형의 각 변이 되는 대각선이 변을 각각 a≤b≤c개 가로지른다고 하자. (a+b+c=n)

답은 a에만 관련이 있으며, 실제로는 log_2(a)+1를 버림한 값이다.

이제 a에 대해 **매개변수 탐색(parametric search)**을 할 수 있다: 1≤a≤n/3.

어떤 k에 대해 k≤a인 삼각형을 만들 수 있을 조건은: (P(i)는 i mod n번째 꼭짓점을 말한다)

"각 i=1..n에 대해 P(i-k)P(i), P(i-1)P(i+k-1), P(i-1)P(i)가 만드는 삼각형에 준원이의 집이 들어가지 않는다"

이다.

Build a Wall!

① 조건 → 가능함: 모든 i에 대한 삼각형 P(i-k)P(i)P(i+k)의 합집합의 바깥쪽 경계는 조건의 삼각형의 경계들과 같다. (삼각형들의 합집합 중간에 빈 공간이 있을 수 있는데, 이는 k를 더 증가시켰을 때 덮인다.)

② 가능함 → 조건: 아 또 까먹었다. 곧 채우도록 하겠다.

이는 “선분 P(i)P(i+k)의 바깥쪽(P(i+1), …, P(i+k)가 있는 쪽)에 준원이의 집이 있게 되는 i가 k개 이하이다.” 라는 조건과 동치이다.

따라서 위 조건을 CCW를 이용해 열심히 활용하여 최대의 a값을 구한 다음, log_2(a)+1의 버림값을 출력하면 답이 된다.

우산

N≤5,000이라면, 간단한 트리DP 방법을 생각할 수 있다.

- dp1(v, k) := 정점 v에서 시작해서 찾고 싶은 정점 k개를 방문하고 돌아오는 최소 시간
- dp2(v, k) := 정점 v에서 시작해서 찾고 싶은 정점 k개를 방문하는 (돌아올 필요 없음!) 최소 시간

이 풀이의 시간 복잡도는 N^2가 되는데, 트리의 크기가 크더라도 같은 dp배열을 가지는 정점 v가 여럿 존재한다는 점에서, 실제로 여러 개의 정점에 대한 dp값을 합쳐야 하는 경우가 아니라면(자식의 서브트리들 중 찾고 싶은 정점이 있는 것이 하나일 때) 배열의 레퍼런스만을 전달하는 방식으로 시간 복잡도 O(N+K^2)로 풀 수 있다.

옥토끼나라

- 트리에서 해결하는것은 서브트리의 크기 및 서브트리 상 감염된 정점들의 개수를 DP로 세는 것으로 충분하며, 어렵지 않다.
- 일반 그래프에서 dfs tree를 생각해보면 무방향 그래프이니 차이는 back edge밖에 없다.
- 서브트리 내에서 back edge로 자기보다 위로 올라갈 수 있다면 자기 위의 컴포넌트와 합쳐지고, 아니면 서브트리가 독립적으로 나누어진다.
- dfs tree 내에서 똑같이 DP를 돌리면서 가장 높이 올라갈 수 있는 위치를 고려해 케이스를 나누어주면 해결 가능하다.

힘 겨루기

- 주어진 대로 구현하면 된다..

정렬하기

- 답은 -1, 0, 1 중 하나로 결정된다.
- 0일때는 원래 정렬되어 있으며, 1은 에르맥의 swap 후 swap 한번만에 정렬이 되는경우, 아니면 에르맥은 그 후 내 선택을 똑같이 따라하면 영원히 정렬되지 않게 할 수 있다.
- 답이 0인지는 현재 배열이 정렬되어있는지를 알아야 한다.
- A[i]=i 인 개수를 저장하면, 각 swap에 대해서 업데이트가 가능하다.
- A[i]=i 인 개수가 N이면 답은 0이다.

정렬하기

- 답이 -1일 때와 1일 때는 어떻게 구분할까?
- 에르맥의 전략을 생각해보자!
- A[i]=i인 개수를 K라고 하자.
- K가 2~N-1일때, A[i]=i인 두 위치를 swap하면 K가 2 감소해 K+2<N이 된다.
- K가 0~N-3일때, 같은 두 위치를 swap하면 위치가 바뀌지 않아 K+2<N이 된다.
- K가 1~N-2일때, A[i]=i인 위치와 아닌 위치를 swap 하면 K가 1 감소해 K+2<N이 된다.
- N>=3 이면, 모든 K에 대해서 정렬되지 않게 하는 에르맥의 전략이 존재한다.
- N=2이며 K=0일때만 답은 1이다.

회의

- Interval Graph에서의 최대 매칭을 구하라는 문제로 귀결된다. : 구간을 정점이라고 하자.
- 시작점을 기준으로 정렬해보자.
- 이제 가장 오른쪽부터 순서대로 보자. 자기보다 오른쪽에 있는 정점들을 모두 해결한 상태이고, 자신이 매칭에 포함되지 않은 상태일 때, 자신과 매칭이 가능한 사용하지 않은 정점이 존재하면, 그런 정점들 중 가장 오른쪽 정점과 자신을 매칭시키는 것이 항상 이득이다.

회의

- 매칭을 항상 해야하는 것은 (왼쪽 정점들만 사용한 매칭) <= (가장 오른쪽 정점을 사용한 매칭) 이기 때문이다. 이는 매칭한 정점을 왼쪽 정점들의 매칭 상에서 사용하면 이를 가장 오른쪽 정점과의 매칭과 바꾸어도 되며, 안 사용하면 이를 가장 오른쪽 정점과 매칭하면 1 증가하는 답을 얻기 떄문이다.
- 매칭 가능한 정점들 중 가장 오른쪽 정점과 매칭해야 함은, 이미 더 오른쪽 정점들은 이미 모두 보았기에 R의 차이는 의미가 없기 때문에, 더 큰 L을 쓰는 것이 항상 이득이기 때문이다.
- 여기까지 하면 O(N^2) 알고리즘을 어렵지 않게 얻을 수 있다.
- 각 오른쪽 정점들에 대해, 바로 매칭 대상을 찾지 않고 set으로 관리해주면서 순회하며 set 상에서 매칭 가능한 정점이 있는지 찾는 방식으로 하면 O(NlgN)에 해결할 수 있다.

모독

세그먼트 트리를 쓴다.

세그먼트 트리의 각 노드는, **이 노드가 관리하는 구간이 꽉 차 있는가?**(관리하는 구간 [l, r]에 대해 생명력이 l,...,r인 하수인이 하나씩 존재하는가?)에 대한 정보를 가진다.

업데이트 시, 다음과 같이 처리한다.

i) 내부 노드의 경우, 두 자식 노드가 꽉 차 있다면 이 노드도 꽉 차 있다.

ii) 리프 노드의 경우(관리하는 구간의 크기가 1), **이 생명력을 가진 하수인이 몇 개인지**를 같이 관리해준다.

이제, 루트로부터 차례로 내려가는 방법으로(왼쪽 자식이 꽉 차 있으면 오른쪽 자식으로, 아니면 왼쪽 자식으로 이동한다) 쿼리의 답을 찾을 수 있다.

모독

jh05013의 별해

- 크기 100만의 펜윅 트리
- 1부터 100만까지 하나씩 들어있는 set을 준비
  - 이 set은 생명력 x의 하수인이 **<u>없음</u>**을 나타냄
- 생명력 x의 하수인의 개수를 체크하는 배열 C
- 정수 k 추가: 펜윅트리 add(k, 1), C[k]++
- 정수 k 제거: 펜윅트리 add(k, -1), C[k]-- 후, C[k]가 0이 되었으면 set에 삽입
- 모독: set의 최소 원소를 보고 펜윅 트리 연산

모독

전 파이썬 유저라 set에서 최솟값을 못 뽑는데요?!?!?!

- **풀지 마세요!**
- 가 아니라…
- set 대신 힙을 사용, 각 수가 힙에 들어있는지 체크하는 배열 H를 준비
- 정수 k 제거: C[k]가 0이고 H[k]가 false이면, true로 바꾸고 힙에 삽입
- 모독: 힙의 최소 원소 x를 보고, C[x]가 양수이면 H[x]를 false로 바꾸고 힙에서 제거. C[x]가 양수일 때까지 반복
- 사실 이 풀이는 C++에도 도움이 된다. set이 굉장히 무거운 탓에 이 풀이가 set을 사용한 풀이보다 훨씬 빠르기 때문이다!!

Celtic Knot

- 상태 DP
- 왼쪽 위부터 글 읽는 순서대로 빈칸을 하나씩 채우는 상황을 가정
- 이미 채워진 부분들이 여러 선들의 집합으로 이루어질텐데, 그 중 어떤 끝점이 어디와 연결되어 있는지에 나머지 경우의 수가 결정됨
- 끝점들이 연결된 방법으로 상태 DP - 총 1 * 3 * … * (2N - 1)가지의 끝점 연결 방식이 있음
- 미리 모든 끝점 연결 방식들이 빈칸 하나를 채우면 어떻게 바뀌는지를 테이블로 저장. memoization으로 구현.

클러스터

- 거리 조건이 없다면 간단한? Convex Hull Trick 문제. 기울기에 별다른 특성이 없어 Dynamic CHT 자료구조가 필요하다
- 세그먼트 트리와 같은 방법으로 구간에 선분을 추가/쿼리하자. how?
- 세그먼트 트리의 각 노드를 CHT 자료구조로 하면 된다
- Lazy propagation?
- Lazy를 stack 등으로 해결 -> 메모리, 시간 전부 O(N^2) 이상

클러스터

- 주목해야 하는 부분은 구간 업데이트-구간 쿼리가 아니라 구간 업데이트-점 쿼리와 점 업데이트-구간 쿼리라는 점
- 구간 쿼리를 하듯 lgN개의 정점만 업데이트 후 점 쿼리는 x가 포함되는 노드를 모두 파고들듯이 참조 : O(lg^2N)
- 점 쿼리는 반대로 파고들며 방문하는 모든 정점에 업데이트, 일반적인 구간 쿼리로 해결 : 역시 O(lg^2N)
- 그러므로 총 O(Nlg^2N)에 해결가능
- 구간 업데이트 점 쿼리는 Lichao Tree 하나로도 해결 가능