1. 가희와 무궁화호

출제 의도

기본적인 파싱을 할 수 있는가?
시각 처리를 할 수 있는가?

의도한 난이도

실버 2 ~ 실버 1

풀이

먼저, [표 1]에 있는 내용을 복사해서 가공하기 편하게 파일로 떨어트립니다. 그 다음에는 tab이나, 스페이스를 기준으로 분리할 수 있게 됩니다. 역명과 서울역으로부터 거리를 얻어옵니다. 그 다음에 무엇이 필요할까요? 가희가 탄 열차가 station1을 출발한 시각과, station2를 도착한 시각이 필요합니다. 이동 거리를 계산한 두 시각의 차이로 나누면 됩니다.

여기서 조심해야 할 코너 케이스가 있습니다. [예제 2]를 보면 알 수 있듯, 중간에 날짜가 바뀌는 경우가 있습니다. 즉, station1을 출발한 시각보다, station2에 도착한 시각이 앞에 있을 수 있는데, 이 경우를 처리해야 합니다.

필요한 스킬

시각 처리, 파싱

2. 가희와 탑

출제 의도

case work를 할 수 있는가

의도한 난이도

골드 5

풀이

한 가지 관찰해야 할 사실 중 하나는 높이는 최대한 작을수록 좋다는 것입니다. 가능한 경우가 주어졌을 때, Case를 4가지로 나누어 보겠습니다.

(1) a, b가 1보다 크고 a < b일 때
높이가 b 이상인 건물이 최소 하나 이상 있어야 함은 자명합니다. 높이는 작을수록 좋으므로, 높이 b짜리 건물을 기준으로 배치를 어떻게 생성할지 생각해 봅시다. ? ... ? b ? ... ? 꼴에서, 왼쪽에서 보는 건물의 개수를 a개, 오른쪽에서 볼 수 있는 건물의 개수를 b라고 하면 b 오른쪽에 있는 것들은 b-1 ... 1로 채워지게 됩니다. 그러면 ? ... ? b b-1 ... 1 이리 채워지게 됩니다.

b 왼쪽에 있는 ? ... ?는 어떨까요? 1 ... a-1로 채워넣으면 됩니다. a<b 이므로, a-1<b입니다. 즉 1 ... a-1 b b-1 ... 1로 채워넣으면, 왼쪽에서 볼 수 있는 건물은 a개, 오른쪽에서 볼

수 있는 건물은 b개가 됩니다. 이제 이것을 최대한 오른쪽으로 밀어버리면 됩니다.

(2) a, b가 1보다 크고 a > b일 때
높이가 a 이상인 건물이 최소 하나 이상 있어야 함은 자명합니다. 높이는 작을수록 좋으므로, 높이 a짜리 건물을 기준으로 어떻게 생성할지를 생각해 보면 됩니다. (1)과 과정이 유사하므로 이 부분은 생략하겠습니다.

(3) b = 1이고 a > b일 때
최소한 높이가 a 이상인 빌딩이 하나 있어야 합니다. 이 건물은 어디에 들어가야 할까요? 만약에 가장 오른쪽에 있는 건물의 높이 r이 a가 아니라고 해 보겠습니다. 그러면 건물의 높이는 ? ? ... ? r이 될 겁니다. 그런데 r이 a가 아니므로, ? ... ? 부분에 높이가 a인 건물이 최소 하나 이상 오게 됩니다. 그런데, 그렇게 되면 단비가 볼 수 있는 건물은 1개가 아니게 됩니다.

따라서, 가장 오른쪽에 있는 건물의 높이는 a가 됩니다. 이제 ? ... ? a 꼴에서 가희가 볼 수 있는 건물이 a개가 되게 해 보겠습니다. 건물 배치 1 2 ... a는 가희가 a개의 건물을 볼 수 있게 합니다. 이것을 최대한 오른쪽으로 땡기면 ? ... ? 1 2 ... a가 됩니다. 이제 ? ... ?에 1을 채워넣으면 됩니다.

(4) a가 1이고 a < b일 때
최소한 높이가 b 이상인 빌딩이 하나 있어야 합니다. 만약에 1을 맨 앞에 두면 어떨까요? 높이 배치가 1 ? ? ... ? b ...가 됩니다. 이 때 가희는 최소 둘 이상의 건물을 볼 수 있게 됩니다. 이를 일반화 시켜 봅시다. 가장 왼쪽에 높이가 b보다 작은 건물이 오면 어떨까요? 높이 배치가 k ? ? ... ? b ... 가 되는데, k < b 이므로 k와 b 사이에 있는 건물들이 모두 b보다 작다면 높이가 k와 b인 건물을 볼 수 있게 됩니다. 그 사이에 있는 건물들이 b보다 크다면 높이가 k인 건물과 높이가 b인 건물 사이에 있는 것을 볼 수 있게 됩니다. 따라서, 가장 왼쪽에는 높이가 b인 건물이 있어야 합니다.

이제 b ? ... ? 꼴에서 가희가 볼 수 있는 건물의 개수를 1개로 만들어 봅시다. 그렇게 하려면, 먼저 1번째 건물 오른쪽에 있는 건물들은 높이가 b보다 무조건 작거나 같아야 합니다. 이제, 단비가 볼 수 있는 건물이 b개가 되게 만들어 봅시다. 그렇게 하려면 b 오른쪽에 있는 것들을 2번부터 순서대로, **1을 연속으로 최대한 많이 붙여보아야** 합니다. 즉 b 1 ... 1 ? ... ? 꼴을 만들었을 때, 오른쪽에서 볼 수 있는 건물의 수가 b개가 되게 만들면 됩니다. 이 때, 1이 언제까지 나오면 될까요?

오른쪽에서 볼 수 있는 건물의 수가 b개가 되게 하는 최소 길이의 배치는 b ... 1이 됩니다. 이 배치를 최대한 우측으로 보내면 됩니다. 그러면 답이 b 1 ... 1 [b ... 1]이 됩니다. 그런데, 이 경우 단비는 1번 건물을 보지 못합니다. 만약에 [] 안에 있는 높이가 b인 건물을 1로 바꾸면 어떻게 될까요? b 1 ... 1 1 [b-1 ... 1]이 되는데, 이 경우에도 단비는 b개의 건물을 볼 수 있게 됩니다.

이제 불가능한 경우를 생각해 봅시다. a+b의 최대값을 생각해 보겠습니다. 먼저 가희와 단비가 동시에 볼 수 있는 건물은 많아야 1개 있음을 보이겠습니다. 만약, 가희와 단비가 동시에 볼 수 있는 건물이 둘 이상이라 하겠습니다. 건물이 ? ? ... ? v(1) ? ... ? v(2) ? ... ?와 같이 배치되어 있고 가희와 단비가 볼 수 있는 건물은 v(1)과 v(2)로 표시해 두었습니다. 가희가 v(1)과 v(2)를 볼 수 있다면, v(1)의 높이 < v(2)가 됩니다. 이 때 단비는 v(1)을 보지 못합니다. 반대로 단비가 v(2)와 v(1)을 볼 수 있다고 하면, v(2) < v(1)이 됩니다. 그런데 이 때에는 가희가 v(2)를 보지 못합니다. **이는 가정에 모순이므로, 가희와 단비가 동시에 볼 수 있는 건물은 많아야 1개입니다.**

즉, a+b의 값은 많아야 n+1이 됩니다. 그렇게 배치를 만들 수 있을까요? 1 ... n 과 같이 건물을 배치하면 가희는 n개의 건물을, 단비는 1개의 건물을 볼 수 있게 됩니다. 따라서 a+b의 값이 n+1을 넘어가면 -1을 출력하면 됩니다.

필요한 스킬
greedy, case work

3. 가희와 gc
출제 의도
요구 사항을 잘 읽고 적절한 자료구조를 선택할 수 있다.
mark & sweep이 어떻게 동작하는지 알고 이를 응용할 수 있다. (원래 의도)

의도한 난이도
골드 2 ~ 골드 1

풀이
object의 id와 reference의 id가 1 이상 10억 이하의 정수이므로, 배열로 관리할 수 없습니다. key값을 빠르게 찾을 수 있는 hash 계열의 자료 구조나 균형 이진 트리로 관리해야 합니다.

이제 어떤 것을 관리해야 하는지를 생각해 봅시다. M과 m 연산이 주어질 때 마다 루트로부터 도달 불가능한 것을 찾아서 제거해야 합니다. 그러므로, bfs와 dfs를 써야 하는 것은 맞아 보입니다. 그런데, 문제는 어떤 것을 어떻게 관리할 것인가입니다. 즉, con[obj_id]에 어떤 정보를 넣어야 하느냐는 것입니다.

하나씩 생각해 봅시다. 일단, obj_1에서 obj_2로 가는 연결 관계가 있습니다. 이 연결 관계에 obj_1로부터 obj_2가 어떤 연결 관계 (강하거나, 혹은 약하거나)를 통해서 가는지가 저장될 겁니다. bfs를 돌릴 때, 보통 con[노드 번호]에는 다음으로 가는 노드 번호를 저장하곤 했습

니다. 그런데 여기에서는 con[obj_id]에 ref_id를 저장합니다. 이 오브젝트로부터 나가는 연결 관계의 id만 저장하면, obj_id에 도달했을 때, 탐색한 ref_id들을 통해 다음으로 갈 오브젝트 번호를 알 수 있기 때문입니다.

그런데 ref_id 또한 매우 커질 수 있으므로, 적절한 자료구조[ref_id]에 reference 정보를 넣어야 합니다. 키 값을 빠르게 찾기 위해서 hash나 tree 계열의 구조를 쓰므로, 레퍼런스 정보들도 거기에 넣으면 좋을 겁니다.

즉, con[obj_id]에 obj_id로부터 나가는 ref_id들이 저장되고, ref에 ref_id에 대응되는 참조 정보가 저장됩니다. 이제 con[obj_id]에 ref_id들을 어떻게 저장할지 생각해 보겠습니다. 중간에 참조 관계들이 제거되는 경우가 있으므로, ref_id를 빠르게 찾는 구조가 필요합니다. 이는 역시 hash나 tree 계열로 할 수 있습니다.

여기서 이를 좀 더 빠르게 하는 방법은 없을까요? 레퍼런스를 삭제하기 위해, ref_id를 굳이 빠르게 찾아야 하고, 삽입/삭제를 해야 할까요? 가비지 콜렉터가 가비지가 만들어 졌다고 바로 객체들을 삭제하지 않습니다. 그런 것처럼, **제거 되었다는 정보만 마킹을 해 놓으면 됩니다. 이것을 어떻게 할까요?**

일단, 실제 ref_id들이 삭제가 되지는 않으므로, ref_id들을 벡터로 관리하겠습니다. 문제는 M 이나 m 연산이 일어나기 전에 id가 ref_id인 참조 관계가 제거되고 추가되고 또 제거되고 추가되는 경우에는 어떻게 해야 할까요? 여기서 한 가지 관찰할 수 있는 사실은, 객체가 추가되고 제거되고 추가되고 제거되고 하는 시점은 모두 다르다는 것입니다. 즉, id가 같은 참조관계가 같은 시간에 동시에 추가되지 않는다는 것을 이용합니다. 여기서 문제. 레퍼런스 id의 상태를 어떻게 최신화 시킬까요?

ref_id에 이벤트를 적용하면 어떨까요? event[a]를 연결 관계 id가 a인 친구가 언제 삽입되거나, 삭제되었는지를 나타내 봅시다. 예를 들자면, ref_id가 30인 친구가 시각 30인 시점에 추가되었다면, event[30] = 30이 됩니다. 만약에 40인 시점에 제거되었다면 event[30] = 40이 됩니다. 이 경우, 30일 때 생성된 30에 대한 연결 정보는 유효하지 않다는 것을 의미합니다. 또 50인 시점에 추가되었다면 event[30]의 값은 50이 됩니다. 즉, obj_id가 연결하는 레퍼런스를 관리할 때, a, b, weakness 뿐만이 아니라, **어느 시점에 추가되었는지까지 관리합니다.** 만약에 o가 추가된 시점이 event[o]보다 작다면, 제거된 것이므로, 무시하고 다시 구축하면 됩니다.

그런데, 연결 관계가 제거되는 시점은 **직접 연결 관계를 제거하는 것 뿐만이 아니라, M이나 m 연산이 수행될 때도 수행됩니다.** 이 때에는 어떻게 해야 좋을까요? 간단합니다. con에 연결 관계 정보 벡터를 담았다고 하였습니다. 그러면 이 정보는 어느 시점에 추가되었는지가 저장될 겁니다. M 이나 m 연산을 수행하게 되면, 방문하지 않는 노드들이 생길 겁니다. 만약에 a에서 b를 잇는 연결 관계가 있는데 a나 b 둘 중 하나를 방문하지 않았다면, 해당 연결 관계는 M이나 m 연산을 수행한 시점에 삭제되었다고 표시하면 됩니다. 즉, 삭제된 시점 이후에 추가되지 않은 연결 관계들은 무효라고 생각하면 됩니다.

필요한 스킬
bfs, dfs, gc에 대한 이해

4. 가희와 베개
출제 의도
bfs와 dfs의 component의 개념을 알고 응용할 수 있다.

의도 난이도
골드 2

풀이
bfs와 dfs에서 component를 잘 생각해 봅시다. 같은 component에 속한 친구들 끼리는 이동이 가능합니다. 이 개념을 확장하면 가희가 위치 (i, j)에 있을 때 어느 지점들을 갈 수 있는지를 묶을 수 있습니다. 이는 bfs나 dfs로 할 수 있고, union find를 이용해도 됩니다.

경사로의 방향에 따라, 어느 component들을 연결하는지를 저장합니다. 그 다음에, 모든 경우에 대해서 component끼리 연결을 해 보면서 가희가 있는 component에서 베개나 가방이 있는 component로 이동할 수 있는지 보면 됩니다. 그런데, 이렇게 해도 될까요? 당연하게도 됩니다. 경사로의 개수가 최대 18개이므로, 경사로와 직접적으로 연결되는 component들은 많아봐야 40개 이하입니다. 이 말은 bfs를 적절한 방법으로 하면 40개 정도의 component들을 탐색할 수 있다는 것입니다. 이를 구현하면 됩니다.

5. 가희와 btd5 2
출제 의도
문제를 잘 읽고, 제약 사항에 맞는 자료 구조를 사용할 수 있는가?

의도한 난이도
플레 4 ~ 플레 3

풀이
문제에서 요구하는 것은 FIRST, LAST, STRONG 타겟을 빠르게 찾고 제거하는 것입니다. 우선 순위가 제일 높은 것을 찾아서 제거까지 해야 하므로, priority queue나 균형 tree를 이용하면 됩니다.

이제, FIRST부터 봅시다. 출구로부터 가장 가깝다는 의미를 역으로 생각해 보면, 풍선이 공격을 받지 않고 계속 이동할 때 사라지는 시간이 제일 작은 것을 찾으라는 의미입니다. 입구로

부터 가장 가까운 것은 어떨까요? 등장한 시간이 제일 큰 풍선을 찾으면 된다는 것을 알 수 있습니다. 즉, FIRST, LAST, STRONG은 아래와 같이 우선 순위를 정리할 수 있습니다.

* FIRST는 사라지는 시간인 eTime이 작고, id가 작은 것
* LAST는 나타난 시간인 sTime이 크고, id가 작은 것
* STRONG은 풍선의 레벨인 lv이 제일 크고, id가 작은 것

즉, 풍선의 정보에는 sTime, eTime, lv이 반드시 들어가야 합니다. 그리고 이러한 정보를 FIRST 맵, LAST 맵, STRONG 맵에 관리하면 됩니다. 그런데, camo 풍선이 문제입니다. 예를 들자면, 아무런 스킬도 배우지 않은 원숭이는 camo 풍선을 볼 수 없는데, camo와 그렇지 않은 풍선을 맵에다가 다 넣어버리면 구분을 하지 못하게 됩니다.

어떻게 하면 될까요? 은신 풍선인지 아닌지 구분해 주면 됩니다. FIRST를 예로 들어 보겠습니다. 풍선이 나올 때 FIRST[0]에는 은신이 아닌 것을 저장하고, FIRST[1]에는 은신인 것을 저장합니다. 그렇게 하면 은신 풍선을 볼 수 없는 원숭이는 FIRST[0]만 보면 되고, 그렇지 않은 경우 FIRST[0]과 [1]을 보면 됩니다.

이제 원숭이 이벤트와 풍선 이벤트를 관리해야 합니다. 어떻게 할까요? 시간별로 이벤트 벡터를 만들면 쉽고 빠르게 순회할 수 있습니다. 제가 2회에서 출제한 가희와 은행의 원래 의도 또한 시간 별로 이벤트 벡터를 만든다는 것이였습니다.

필요한 스킬
자료구조, 파싱