컴퓨터공학과 20211507 김동건

문제1) Finding Celebrities

문제에 대해 설명하기 위해 N명의 사람에게 1 ~ N까지의 수를 배정하였다. 또, i번 사람이 j번 사람을 아는지 물어보는 것을 본 글에서는 query(i, j)라고 표현하였다. 마지막으로, N명의 사람들의 관계를 그래프로 나타낼 때, 각 i번 사람을 노드 i로 표현하였고, i번 사람이 j번 사람을 알면 노드i에서 노드j로 가는 단방향 간선이 있는 것으로 생각하였다. I번 사람이 j번 사람을 모르면 i에서 j로 향하는 간선이 없는 것으로 가정하였다.

우선, 문제를 끝까지 읽고 질문 a)에 있는 “some”을 보고 “과연 celebrity가 있다면 여러명이 존재할 수 있을까?”라는 의문이 들었다. 이 의문에 대해서는 생각보다 쉽게 답을 알 수 있었다. 만약, i번 사람과 j번 사람이 둘 다 celebrity라고 가정해보자. 그렇다면 i번 사람은 j번 사람을 모른다는 뜻이다. 그러나, 이는 j번 사람이 celebrity라는 사실에 모순이다. 즉, celebrity가 존재한다면 오직 한 명 존재한다.

그렇다면 celebrity가 존재하지 않을 수도 있을까? 답은 “존재하지 않을 수도 있다”이다. 만약 모든 i != j인 query(i, j)의 답이 YES라면, 각 사람들 모두가 아는 사람이 존재하기 때문에 celebrity가 없다.

이제, 효율적으로 celebrity를 찾는 방법을 생각해보자. 우선 모든 i != j인 모든 (i, j)에 대해 query를 물어보면 답을 쉽게 알 수 있다. 각 사람을 보면서 그 사람이 다른 사람을 아무도 모르고, 모든 다른 사람이 그 사람을 아는지 확인하여 확인할 수 있다. 이 풀이는 N^2 – N 번의 질문이 필요하다. (전체 i, j 쌍에서 i = j인 쌍의 수를 뺀 값이다.)

조금 더 적은 질문으로 해결할 수 있을지 고민해 보았다. 맨 처음 celebrity의 후보가N명이 있다. 이때 후보에 있는 i, j에 대해 query(i j)를 한 번 사용하면 후보에서 i와 j중 하나를 무조건 없앨 수 있다. 만약 query(i, j)가 YES라면 i는 j를 알기 때문에 celebrity가 아니다. 반대로 NO라면 i는 j를 모르기 때문에 j는 celebrity가 아니다. 즉, N개의 후보를 추가해두고, 후보 중 제일 작은 i와 j를 꺼내 query(i, j)를 물어보자. 이때 후보가 아닌 것이 확실히 된 것을 후보에서 제외시켜주고 나머지는 다시 후보에 넣어준다. 이를 N-1번 반복하면 1명의 후보만 남는다. 그렇지만 위에서 말했듯이 celebrity는 존재하지 않을 수도 있기 때문에 그 사람이 celebrity가 아닐 수도 있다. 아래와 같은 예시를 생각해보자.

원, 그림, 라인, 화이트이(가) 표시된 사진

자동 생성된 설명

위에서의 풀이대로 후보에 [1, 2, 3, 4]가 있고, 순서대로 진행을 하게 되면 [1, 2, 3, 4] -> [2, 3, 4] -> [2, 4] -> [4]가 된다. 그렇지만 1번 사람이 4번 사람을 모르기 때문에 4번 사람이 celebrity가 아님을 알 수 있다. 즉, 아직 물어보지 않은 i != 4에 대해 query(i, 4)를 모두 물어봐서 4번사람이 celebrity임을 확인해 주어야 한다. 위에서 든 예시에서는 query(1,4), query(3, 4)를 물어보고( query(2, 4)는 이미 물어보았기 때문에 생략), 모두 YES인 경우에만 4번 사람이 celebrity이다.

정리하면 처음 N개의 후보를 한 명으로 줄이는 과정에서 N-1번 질문을 하게 되고, 남은 후보가 celebrity인지 확인하는 데에 최악의 경우 N-2번 질문을 하게 된다. 따라서 최악의 경우에도 2N-3번의 질문으로 문제를 해결할 수 있다. 즉, O(N)의 시간이 걸린다. 공간 복잡도는 어떨지 생각해보자. 마지막에 후보가 하나 남았을 때, 이미 물어본 쿼리들에 대해 저장해야 한다. 얼핏 생각해보면 N^2개를 저장해야 할 것 같지만, 후보에서 지워져 나갈 때마다 그 후보에 대한 쿼리 기록들은 지워주어도 되기 때문에 최악의 경우에도 O(N)보다 큰 공간을 사용하는 경우는 없다.

문제2) Minimum Calculation

우선 n이 2의 제곱수이면 어렵지 않게 답을 구할 수 있다. 같은 수를 계속하여 제곱하여 곱하면 되기 때문이다. 즉, n = 8(=2^3)라면

x1=a;

x2=x1\*x1;

x3=x2\*x2;

b=x3\*x3와 같은 과정을 거치면 된다. 즉, n이 2^k 꼴이라면 k+1번의 연산을 통해 a^n을 만들 수 있다. k가 0인 경우에도 b=a라는 연산 한 번에 가능하므로 성립한다.

이제, n이 2의 제곱수가 아닐 때를 생각해보자. 예를 들어, n = 11인 경우를 생각해보자.

먼저, 위에서 2의 제곱수인 수일 때 계산하는 과정에서 우리는 2의 제곱수 꼴은 쉽게 만들 수 있음을 알게 되었다. 즉, 그것들을 적절히 곱하여 답을 구하는 방법을 생각해볼 수 있다. n = 11이라면 a^8, a^2, a^1를 구하여 모두 값을 곱하면 a^11가 되는 것을 알 수 있다. 이때, 8, 2, 1은 11을 이진수 꼴로 나타냈을 때 (1011) 1이 위치한 자리라는 것을 알 수 있다. 따라서 우리는 n을 이진수로 나타낸 후, 비트가 1인 위치의 값들을 모두 구하여 곱하는 것을 생각해볼 수 있다.

x1=a; (a)

x2=x1\*x1; (a^2)

x3=x2\*x2; (a^4)

x4=x3\*x3 (a^8)

y1 = x2\*x4 (a^10)

b = y1\*x1 (a^11)

와 같이 해결할 수 있다. 즉, n을 이진수로 나타냈을 때 가장 큰 1의 위치가 오른쪽에서 p번째라 하고, n을 이진수로 나타냈을 때 1의 개수를 q라 했을 때, 필요한 값들을 만드는데 p + 1번의 연산이 들고, 1인 값들을 모두 곱하여 합쳐주는 데에 q – 1번의 연산이 든다. 즉, 총 p + q번의 연산이 들고, 두 값 모두 최악의 경우에 ceil(log2\_n)의 값을 가진다. (ceil은 정수 올림 연산을 의미한다.) 즉, O(log2\_n)의 시간복잡도를 가진다. 또, 최악의 경우에도 연산을 하는 만큼의 변수가 필요하므로 공간복잡도도 마찬가지로 O(log2\_n)이다.