­­­­1주차 보고서

전공: 컴퓨터공학과 학년: 2학년 학번:20211547 이름:신지원

**1.**

총 n명의 사람 중 유명인을 찾아야 할 때, 사람들에게 다음과 같이 물어볼 수 있다. “너가 유명인이니? 아니라면 유명인을 아니? 안다면 너와 인접하고 있니?” 만약 “인접하고 있다.” 라고 대답한다면 그 사람의 상하좌우에 유명인이 위치하고 있을 경우의 수를 고려하여 5개의 경우의 수 중 하나에 유명인이 위치하고 있을 것이다. 따라서 너비탐색으로 유명인을 찾을 수 있다.

폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

위의 그림과 같이 (2,D)에 유명인이 위치하고 있다고 가정하자. 이때, (1,A) 부터 (1,H) 까지 돌면서 1줄에 있는 사람들에게 유명인과 인접해 있냐고 물어볼 것이다. (1,D) 에 위치한 사람이 인접해 있다고 말할 것이고, (1,D)가 의 상하좌우에 위치한 (1,C), (1,E), (2,D) 에게 다시 물어볼 수 있다. 하지만 (1,C) 에겐 이미 물어보았기 때문에 이미 물어본 사람에겐 묻지않도록 처리해주어야 한다. 따라서 (1,D) – (2,D) 의 순으로 유명인을 찾을 수 있을 것이다.

폰트, 디자인이(가) 표시된 사진

자동 생성된 설명

그렇다면 유명인이 2명일 때는 어떻게 고려해야 할까, (1,D) – (2,D) 순으로 유명인을 한 명 찾았다면 (2,D) 를 제외하고 2번째에 있는 모든 사람에게 물어볼 것이다. 그렇다면 (2,F) 에 위치한 사람이 유명인과 인접해있다고 답할 것이며, (3,F) 에 위치한 사람을 찾을 수 있을 것이다. 따라서 큐에는 ‘(1,D) 에 의하여 (1,E), (2,D) 추가 - (1,E) 삭제 – (2,D) 발견 후 삭제 – (2,F)에 의하여 (2,G), (3,F) 추가 - (2,G) 삭제 - (3,F) 발견 후 삭제’ 의 순으로 진행될 것이다.

이때, 가로의 열과 세로의 열을 모두 돌면서 물어보지 않은 사람에게만 유명인에 대하여 질문한다. 그 사람이 유명인이 맞는지 확인한 뒤, (유명인을 아직 다 찾지 않았을 경우에) 맞는지에 상관없이 유명인과 인접하여 있는지 물어본다. 만약 인접해 있다면 상하좌우 중 방문하지 않은 사람들에게 다시 물어보아 유명인을 찾는다. 시간복잡도를 구한다면 (행의 수) \* (열의 수) \* ( 유명인인지 물어보는 if문 \* 맞다면 상하좌우를 체크해보는 for 문) 을 계산하여 “row \* column ( 1 \* 4) = 4 \* row \* column” 으로 표기할 수 있다. 따라서 시간복잡도는 T(row\*column) 으로 행과 열의 곱으로 나타낼 수 있으며 n 이 총 사람의 수일 때, O(n) 으로 나타낼 수 있다. 최선의 경우의 수는 유명인이 없을 때로 for 문이 실행되지 않을 것이다. 공간복잡도는 큐라는 빈리스트에 행의 수와 열의 수만큼 들어갈 수 있기 때문에 O(n) 이라고 말할 수 있다.

해법에 대한 반론은 충분히 제기될 수 있다. 유명인의 수를 정해주지 않은 경우엔 남은 유명인이 없다고 하더라도 끝까지 돌아 방문하지 않은 모두에게 질문을 해야 한다.

질문을 최소로 하기 위해선 사실 한 사람을 붙잡고 “유명인이 모두 어디에 있는지 알려줘!” 라고 물어보면 되지만 컴퓨터상에서 라이브러리를 사용하지 않고는 어렵기 때문에 너의 옆에 유명인이 있는지 라는 질문을 통해 찾도록 하였다.

**2.**

대입 연산과 곱셉 연산만으로 이루어진 산술 연산을 수행하는 계산기기 때문에 a와 n이 주어졌을 때 결과값은 항상 a^n 이 도출될 것이다. 가장 일반적인 방법으로는 a를 계속 곱해주는 반복문을 돌거나 재귀함수를 사용하여 곱셈 연산을 수행하는 방법이 있다. 이 방식으로 진행하였을 때 a^n을 구하기 위해서는 n번 반복문이나 재귀함수를 돌면서 연산해야 한다. 이를 시간복잡도로 나타냈을 때 O(n)으로 나타낼 수 있다.

하지만 더 효율적인 코드를 작성하기 위하여 분할하는 방법을 생각해보았다. 곱셈을 쪼갤때 흔히 지수법칙을 사용하기 때문에 이 문제에도 지수법칙을 반영하여 코드를 구현하는 것이 효율적이라고 생각하였다. 지수법칙은 a^(n+m) = a^n \* a^m 으로 표기할 수 있다. 따라서 이 문제에서 a의 n+1 승을 구한다면 a^n \* a^n = a ^ (2n) 으로 나타낼 수 있다. 또한 지수 법칙은 지수가 짝수일 때와 홀수일 때로 나눌 수 있는ㄷ-, 짝수일 때는 a^n = a^(n/2) \* a^(n/2) 으로 나타낼 수 있으며 홀수일 때는 a^n = a((n-1)/2) \* a((n-1)/2) \* a 로 나타낼 수 있다. 따라서 이 문제에서도 n이 짝수와 홀수일 때로 나누어서 코드를 구현해야 한다.

지수법칙을 사용하여 n의 값이 0또는 1일 때까지 반복할 수 있도록 호출하는 함수를 하나 만든다. 이 함수 내에서는 지수가 0또는 1이 될 때까지 지수를 반으로 쪼개주는 중복 호출문을 호출한다. 자연수를 반으로만 쪼개기 때문에 지수는 1이 될 수 밖에 없다. 그 뒤에 다시 쪼갰던만큼 (짝수일 떄는 n/2, 홀수는 (n-1)/2) 다시 반복하여 a를 n 번 곱하였을 때 어떤 수가 나오는지 찾아줄 수 있다. 만약 지수가 홀수라면 k \* k \* a 를, 지수가 짝수라면 k \* k 를 return 하여 곱해줄 수 있다. (여기서 k는 지수/2) 이해를 돕기 위하여 아래 중복 호출문의 코드를 첨부하였다.

텍스트, 폰트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

따라서 이렇게 작성한 코드의 시간 복잡도는 O(long n) 을 나타낼 수 있다. 그 이유는 log N 만큼의 레벨이 있고, 각 레벨에서 걸리는 시간이 O(1)이기 때문에 O(log N) \* O(1) 은 O(log N) 이 된다. 아래의 그림을 참고할 수 있다. 공간복잡도 또한 log N의 레벨이 있기에 O(log N) 으로 구성될 것이다.

도표, 라인이(가) 표시된 사진

자동 생성된 설명

이보다 더 시간복잡도를 낮추기 위해선 내장되어있는 함수를 사용하는 방법이 있을 것이다. 또한 int 형이 아니라 long long 형이라면 더 큰 수도 다룰 수 있을 것이다.