2019년 상반기 LINE 인턴 채용 코딩테스트 문제 해설

권정현 | 2019.09.19
LINE에서 VoIP를 활용한 iOS 앱을 개발하고 있습니다.

LINE에서 개발 직군을 뽑을 때 신입이든 경력이든 가장 먼저 보는 것이 코딩 테스트입니다. LINE의 코딩 테스트는 일반적인 알고리즘 경진대회와는 경향이 조금 다른데요. 알고리즘 경진대회는 1등을 가려내기 위한 복잡하고 어려운 문제를 출제하는 경향이 있다면, LINE은 면접으로 가는 과정에서 개발자로서의 개발 능력을 확인하는 데 목적이 있습니다.

이를 위해서 어려운 알고리즘을 이해하고 활용하는 데 익숙한 기술을 가진 분들을 찾기보다는, 문제의 요구사항을 이해하고 컴퓨터공학 이론을 바탕으로 그에 맞는 적절한 해결책을 찾아 구현할 수 있는 기술을 가진 분들을 찾고자 합니다. 어떤 문제가 나오는지 이해할 수 있도록 2019년 상반기 코딩 테스트에서 실제 출제된 문제와 간단한 해설을 공유합니다.

문제 설명

문제

연인 코니와 브라운은 광활한 들판에서 '나 잡아 봐라' 게임을 한다. 이 게임은 브라운이 코니를 잡거나, 코니가 너무 멀리 달아나면 끝난다. 게임이 끝나는데 걸리는 최소 시간을 구하시오.

조건

1. 코니는 처음 위치 C에서 1초 후 1만큼 움직이고, 이후에는 가속이 붙어 매 초마다 이전 이동 거리 + 1만큼 움직인다. 즉 시간에 따른 코니의 위치는 C, C + 1, C + 3, C + 6, …이다.
2. 브라운은 현재 위치 B에서 다음 순간 B − 1, B + 1, 2 * B 중 하나로 움직일 수 있다.
3. 코니와 브라운의 위치 p는 조건 0 <= x <= 200,000을 만족한다.
4. 브라운은 범위를 벗어나는 위치로는 이동할 수 없고, 코니가 범위를 벗어나면 게임이 끝난다.

입력 형식

표준 입력의 첫 줄에 코니의 위치 C와 브라운의 위치 B를 공백으로 구분하여 순서대로 읽는다.

출력 형식

브라운이 코니를 잡을 수 있는 최소시간 N초를 표준 출력한다. 단 브라운이 코니를 잡지 못한 경우에는 -1을 출력한다.

예제

입력: 11 2

출력: 5

코니의 위치: 11 → 12 → 14 → 17 → 21 → 26

브라운의 위치: 2 → 3 → 6 → 12 → 13 → 26

브라운은 코니를 5초 만에 잡을 수 있다.

문제 풀이

잘못 접근한 방법

다음 설명할 두 가지 방법이 대표적으로 잘못 접근한 방법이라고 할 수 있습니다.

먼저 아래 `solve1` 함수는 시간이 t1일 때 코니와 브라운의 위치가 p1으로 같으면 잡았다고 판단하는 알고리즘입니다. 반례로는 C = 11, B = 1인 경우인데요. 코드를 실행해 보면 1 → 2 → 4 → 5 → 10 → 20 → 40 → 39로 이동하여 7초 만에 잡습니다. 하지만 실제로는 1 → 2 → 3 → 4 → 8 → 16 → 32로 이동하여 6초 만에도 잡을 수 있습니다. 이 코드에선 32 위치에 최초 5초에 도착하지만 6초에 도착하는 경우는 고려하지 않았기 때문에 최소 시간을 찾을 수 없게 됩니다.

```
int solve1(int conyPosition, int brownPosition) {
    bool visit[200001];
    queue<pair<int, int> > queue;

    memset(visit, 0, sizeof(visit));
    visit[brownPosition] = true;
    queue.push(make_pair(brownPosition, 0));

    while(!queue.empty()) {
        int currentPosition = queue.front().first;
        int currentTime = queue.front().second;
        int newPosition;

        queue.pop();

        if (currentPosition == conyPosition + currentTime * (currentTime + 1) / 2)
            return currentTime;
```

```
18.
19.         // if not visist than push queue
20.       }
21.
22.       return -1;
23.     }
```

다음으로 아래 `solve2` 함수는 시간이 t1일 때 브라운이 코니가 방문한 곳을 방문했다면 잡았다고 판단하는 알고리즘입니다. 반례로는 C = 6, B = 3인 경우가 있습니다. 코드를 실행해 보면 3초 만에 잡는다고 출력되는데요. 절대로 3초 만에 잡을 수 없고 3 → 6 → 7 → 8 → 16로 이동하여 4초에 최초로 잡을 수 있습니다. 이 코드는 `sovle1` 에서 고려하지 못했던 '방문했던 것'을 고려했습니다. 브라운의 경우 2초가 지났을 때 3 → 6 → 12 위치에 방문했다고 표시됩니다. 코니의 경우 3초가 지나면 6 → 7 → 9 → 12 위치에 방문하게 됩니다. 이때 `solve2` 함수에선 코니가 방문한 위치 12가 브라운이 2초에 방문했던 위치이기 때문에 3초에도 방문할 수 있다고 판단하여 3초를 출력하게 됩니다. 하지만 문제의 조건에 의해 2초 때 12 위치에서 3초 때 12 위치로 이동할 방법이 없으므로 아래 알고리즘에는 구멍이 존재합니다.

```
01.   int solve2(int conyPosition, int brownPosition) {
02.     int time = 0;
03.     bool visit[200001];    queue<int> queue;
04.
05.     memset(visit, 0, sizeof(visit));
06.     visit[brownPosition] = true;
07.     queue.push(brownPosition);
08.
09.     while (1) {
10.       conyPosition += time;
11.
12.       if (conyPosition > 200000)
13.         return -1;
14.       if (visit[conyPosition])
15.         return time;
16.
17.       for (int i = 0, size = queue.size(); i < size; i++) {
18.         int currentPosition = queue.front();
19.         int newPosition;
20.
21.         queue.pop();
22.
23.
24.         // if not visist than push queue
25.       }
26.       time++;
27.     }
28.   }
```

해답

위 `solve2` 함수를 유심히 관찰하면 문제를 해결할 수 있는 포인트를 발견할 수 있습니다. t 초에서 위치가 p라고 가정할 때, t + 1초에서 위치는 p일 수 없습니다. 하지만 t + 2초에서는 위치가 p일 수 있습니다(t → t − 1 → t 혹은 t → t + 1 → t). 위 사실을 토대로 방문 시간을 홀수, 짝수로 나눠서 고려해야 한다는 것을 알 수 있습니다. 아래 `solve` 함수는 t 값을 증가시키면서 '코니가 t 초 후에 p 위치에 도착했을 때, 브라운이 p 위치에 t − 2k(단, k >= 0인 정수) 시간에 도착했는지 여부'를 판단하여 해당 조건을 만족하는 t를 찾는 알고리즘입니다.

```
int solve(int conyPosition, int brownPosition) {
    int time = 0;
    bool visit[200001][2];
    queue<pair<int, int> > queue;

    memset(visit, 0, sizeof(visit));
    queue.push(make_pair(brownPosition, 0));

    while (1) {
     conyPosition += time;

      if (conyPosition > 200000)
        return -1;
      if (visit[conyPosition][time % 2])
        return time;

      for (int i = 0, size = queue.size(); i < size; i++) {
        int currentPosition = queue.front().first;
        int newTime = (queue.front().second + 1) % 2;
        int newPosition;

        queue.pop();

        newPosition = currentPosition - 1;
        if (newPosition >= 0 && !visit[newPosition][newTime]) {
          visit[newPosition][newTime] = true;
          queue.push(make_pair(newPosition, newTime));
        }

        newPosition = currentPosition + 1;
        if (newPosition < 200001 && !visit[newPosition][newTime]) {
          visit[newPosition][newTime] = true;
          queue.push(make_pair(newPosition, newTime));
        }

        newPosition = currentPosition * 2;
```

```
            if (newPosition < 200001 && !visit[newPosition][newTime]) {
                visit[newPosition][newTime] = true;
                queue.push(make_pair(newPosition, newTime));
            }
        }
        time++;
    }
}
```

마무리

2019년 상반기 코딩 테스트에서 출제된 문제 중 하나를 풀어 보았습니다. 문제를 풀어 본 사람들은 기억이 새록새록 날 것이라고 생각합니다. 문제가 어렵지 않아서 풀이를 단번에 생각해 낸 사람도 있을 것입니다. 만약 그렇지 않다면 기본적인 BFS(Breadth First Search)로 접근하여 반례를 찾고, 반례를 분석하여 올바른 접근 방법을 찾아내는 것이 핵심이라고 할 수 있습니다.

이번 글이 LINE 코딩 테스트를 준비하시는 분들께 조금이나마 도움이 되었으면 합니다.

Security (19)

미분류 (8)

태그

Advent Calendar Android Architecture Armeria Blockchain Bug Bounty BugBounty C/C++ chatbot CircuitBreaker design sprint DevOps devweek Docker frontend GitHub Infra Intertrust Interview JavaScript Kotlin LINE API Expert LINE BOT linedevday LINE Events LINE LIVE LINE Notify LINK LLVM Management/Operation Messaging API Obfuscation OpenSource Openstack Optimization Prometheus Promgen Redis Security Server Server-dev sharding Spark summer homework workshop