[1.4] 대표문제: Ladder

대표문제: Ladder

사다리 게임에서 결과 문제에서 Ladder 문제를 풀어보자. 2차원 그리드 형태의 사다리 게임에서 시작점 (0,0)에서 출발하여 아래로 이동하며 사다리를 만나면 좌우로 이동하여 다른 열로 이동하고, 이동하는 방향의 반대 방향으로 계속 이동하여 최종 도착점 (0,N-1)에 도달하는 경로를 찾는 문제입니다. 사다리는 0(빈 칸), 1(오른쪽 이동), 2(왼쪽 이동)의 세 가지 값으로 표현되며, 유효한 사다리는 정확히 하나의 시작점과 하나의 도착점을 가지게 됩니다. 예제에서는 작은 크기의 맵 문제를 제공할 수 있다.

Step 1 / 3

사다리를 표시하는 2차원 정수 맵는 전형 프로그래밍과 알고리즘에 자주 등장하는 입력 형태이다.

```
4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

해결 능력 (L)

사다리 문제를 푸는 방법은 다음과 같다 (Level 1 수준):

1. 인덱스 시작 위치 설정하기: 맨 아래 줄(N-1)에서 시작점 찾기.
2. 스캔하기: 맨 아래 줄부터 위로 올라가면서 각 칸의 값을 확인하기.
3. 이동하기: 값이 1이면 오른쪽, 2이면 왼쪽 방향으로 이동하여 다른 열로 이동하고, 0이면 계속 위로 올라가기.
4. 도착하기: 맨 위 줄에 도착하면 현재 열 번호가 정답이다.

사다리는 배열로 표현할 수 있다. 아이디 인덱스 기준 배열의 0~N-1까지 사다리 배열 요약해 배열이 필요하다. 그렇다 하지만 2차원 배열을 다룰 것이다. 그러므로 프로그램 작성 시 사다리는 배열로 간주해 배열의 인덱스 형태로 접근한다.

2차원 배열 arr[][]이라고 선언할 수 있다. 우리는 arr[배열의 인덱스][열 인덱스] 형태로 접근하게 된다. 이 때 arr[0][0]은 맨 왼쪽 위 칸, arr[N-1][0]은 맨 왼쪽 아래 칸을 가리키게 된다. 사다리 게임에서 시작점은 맨 아래 줄(N-1)에서, 도착점은 가장 위 줄(0)에서 찾아야 하므로 인덱스 사용하는 것이 적합한 방법이다.

2차원 배열의 길이를 알아보기 위해 arr.length[0]을 가져와 배열 arr의 첫 번째 인덱스 값을 가져오게 되는데 이는 arr.length[1]을 가져와 배열 arr의 두 번째 인덱스 값을 가져오게 되는 것과 다르다. 2차원 배열의 경우 arr.length[1]을 가져와 배열 arr의 두 번째 인덱스 값을 가져오게 되는 것과 다르다. 2차원 배열의 경우 arr.length[0]을 가져와 배열 arr의 첫 번째 인덱스 값을 가져오게 되는 것과 다르다. 2차원 배열의 경우 arr.length[1]을 가져와 배열 arr의 두 번째 인덱스 값을 가져오게 되는 것과 다르다. 2차원 배열의 경우 arr.length[0]을 가져와 배열 arr의 첫 번째 인덱스 값을 가져오게 되는 것과 다르다.

다 배열의 순회

배열의 순회는 for 루프 등을 통한 반복이 필요하다. 일반 1차원 배열의 경우 for (int i = 0; i < arr.length; i++) 형태로 순회하는 것이 일반적이다. 2차원 배열의 경우 이중 for 루프를 사용하여 각 행(i)과 각 열(j)을 순회하는 방식을 사용한다. 이 때 arr.length[0]이 아닌 arr.length[1]을 사용하는 것에 주의해야 한다. 이 때 arr.length[0]이 아닌 arr.length[1]을 사용하는 것에 주의해야 한다. 이 때 arr.length[0]이 아닌 arr.length[1]을 사용하는 것에 주의해야 한다.

2차원 배열의 요소를 접근하기 위해 arr[배열의 인덱스][열 인덱스] 형태로 접근하는 것이다.

```
int[][] arr = {{1, 2, 3}, {4, 5, 6}};
```

2차원 배열의 선언 및 사용 예시입니다. 2차원 배열의 arr을 생성하고 초기값을 지정하는 방법이다. int arr[][]=new int[2][3]; 2차원 배열의 경우 행의 개수와 열의 개수를 지정하는 것이다. 2차원 배열의 경우 행의 개수와 열의 개수를 지정하는 것이다. 2차원 배열의 경우 행의 개수와 열의 개수를 지정하는 것이다.

사다리 게임에 2차원 구조를 적용하기 위해 2차원 배열 arr을 사용하여 한다고 생각하자. 이 때 arr[0][0]은 맨 왼쪽 위 칸을 가리키고, 마지막 인덱스 값은 맨 아래 줄에 해당하는 것이다. 이 때 arr[0][0]은 맨 왼쪽 위 칸을 가리키고, 마지막 인덱스 값은 맨 아래 줄에 해당하는 것이다. 이 때 arr[0][0]은 맨 왼쪽 위 칸을 가리키고, 마지막 인덱스 값은 맨 아래 줄에 해당하는 것이다.

```
int[][] arr = {{1, 2, 3}, {4, 5, 6}};
int[][] arr2 = new int[2][3];
int[][] arr3 = new int[2][3];
int[][] arr4 = new int[2][3];
int[][] arr5 = new int[2][3];
int[][] arr6 = new int[2][3];
int[][] arr7 = new int[2][3];
int[][] arr8 = new int[2][3];
int[][] arr9 = new int[2][3];
int[][]
```

```
48      nxt = x + dx[d_prev];
49      nyt = y + dy[d_prev];
50
51      if(nxt>=0 && nxt<n && nyt>=0 && nyt<n && x != nxt && y != nyt) {
52          d = (d+1+dir_num)%dir_num;
53          nxt = x + dx[d_prev];
54          nyt = y + dy[d_prev];
55      }
56
57      for(int i=0; i<n; i++)
58          for(int j=0; j<n; j++)
59              ans[i][j] = ans[i][j] + ans[i][j];
60  }
```

P32

```
38      pr_cnt[*tp] = arr[*tp].size();
39      pr_cnt[*tp] = 1;
40  }
41
```

1 ~ 17 배열에서 트스값을 찾는다.

7 ~ 13 트스값을 찾기 위해서 굳이 일방향이게 순회할 할 필요는 없다.

15 트스값을 중복하여 찾지 않기 위해서 중중하기 큰 값으로 바꾸어 준다.

34 ~ 35 배열을 순회 할 때 다음으로 이동할 방향을 지정한다.

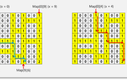
47 ~ 51 경계 문제시 방향을 바꾼다.

[1.4] 대표문제. Ladder 해결하기

대표문제. Ladder

사다리 게임에서 점에 도착하기 되는 출발 점을 찾아라. 정은 2로 표시한다.

x = 4인 출발점에서 점심표로 표시한 바와 같이, 아래 방향으로 진행하면서 좌우 방향으로 이동 가능한 통로가 나타나면 방향전환을 하게 된다. 방향 전환 이후는 다시 아래 방향으로만 이동하게 되며, 바닥에 도착하면 종주게 된다.



입력

첫 번째 줄에 test case의 수 T(1 ≤ T ≤ 100) 가 주어진다. 각 테스트 케이스의 첫 번째 줄에는 테스트케이스의 번호가 주어지며, 바로 다음 줄에 테스트 케이스가 주어진다. 각 테스트 케이스 별로 100개의 사다리는 연속적 / 독립적인 형태로 출현한다. 도착 지점은 2로 표현한다.

P33

입력 예시

```
2
1
1001100000001100000101000000...
1001100000001100000101000000...
1001111111111100000101000000...
...
2
1001100000001100000101000000...
1001100000001100000101000000...
1001111111111100000101000000...
...
```

출력

X에 도착하기 되는 출발점의 x좌표를 출력한다.

출력 예시

```
67
3
```

해설

사다리는 출발 점이 다르면 도착점도 다르기 때문에 X에 도착하는 출발점을 알기 위해서 모든 출발 점을 조사할 필요는 없다. 도착점에서 거꾸로 올라가면 되기 때문이다. 사다리의 정보를 저장하기 위해선 100 × 100크기의 2차원 배열이면 충분 하지만 양쪽면에서 처리를 쉽게 하기 위해 아래 그림과 같이 아래 공간을 만든다. 왼쪽으로 될 때전 부분이 추가됨 부분이다.

Map[0][5] (x = 4)

0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0	1	0	0	1	0
0	1	0	0	0	0	1	0	1	1	1	1	0
0	1	0	0	0	0	1	0	1	0	0	1	0
0	1	0	0	0	0	1	1	1	0	0	1	0
0	1	0	0	0	0	1	0	0	0	0	1	0
0	1	1	1	1	1	1	0	1	1	1	1	0
0	1	0	0	0	0	1	0	1	0	0	1	0
0	1	1	1	1	1	1	0	1	0	0	1	0
0	1	0	0	0	0	1	0	1	0	0	1	0
0	1	0	0	0	0	1	0	1	0	0	2	0
0	0	0	0	0	0	0	0	0	0	0	0	0

만약 이런 여유를 두지 않는다면 배열의 크기를 벗어난 부분을 조사 하지 않도록 처리를 해주어야 한다. 해당조를 조금 이러는 것보다는 처리를 실용하게 하는 것이 중요 하다. 이렇게 주어진 입력보다 조금 더 크게 배열을 잡는 것부는 처리 찾기, 지표 찾기 등이 쉽다.

가장 먼저 해야 될 일은 도착점을 찾는 것이다. 도착점을 찾은 이후에는 점우에 있이 있는지 살펴 보고 없다면 바로 건너, 만약 점우에 있이 있다면 끝까지 이동한 뒤 바로 건너. 대려분은 이 작업만 반복해서 진행 하면 된다.

P35

코드 제공

```
1   // ladder[][] : 0=길, 1=벽, 2=시작
2   int[][] ans = new int[100][100];
3
4   int y = 100, x = start;
5
6   while (y >= 1)
7   {
8       if (ladder[y - 1][x - 1] == 1 || x == 100)
9       {
10          while (ladder[y - 1][x - 1] == 1)
11          {
12              y--;
13          }
14          y--;
15      }
16      else if (ladder[y - 1][x + 1] == 1)
17      {
18          while (ladder[y - 1][x + 1] == 1 && x != 100)
19          {
20              x++;
21          }
22          y--;
23      }
24      else {
25          y--;
26      }
27  }
28
29  return x;
30
31  int[][] ans(int, int);
32  int[] ans(int);
33
34  int n=0;
35  int[] tmp;
36  int[][] ans2[100][100];
37  int[] ans_temp(int, int);
38  int[] ans_temp(int);
39
40  scanf("%d", &test_case);
41
42  for (int t = 1; t <= test_case; t++) {
43      scanf("%d", &tc_num);
44
45      // ACCD 초기화
46      for (int i = 0; i < 100; i++)
47          for (int j = 0; j < 100; j++)
48              ans2[i][j] = 0;
49
50      for (int i = 0; i < 100; i++)
51          for (int j = 0; j < 100; j++)
52              scanf("%d", &map[i][j]);
53
54      // 모든 경우 찾기
55      for (int i = 0; i < 100; i++) {
56          if (map[0][i] == 2) {
57              tmp = ans(i);
58              n = tmp[0];
59          }
60      }
61
62      printf("%d\n", ans(n)[0], tmp[1] + 1);
63
64      return 0;
65  }
66
```

1 위에서 설명한 것처럼 2차원 배열의 크기를 100 × 100이 아닌 102 × 102로 여유를 두고 할당한다.

6 ~ 11 오른쪽 사다리가 있으면 끝까지 이동한 뒤 바로 한 칸 이동한다.

12 ~ 17 우측에 사다리가 있으면 끝까지 이동한 뒤 바로 한 칸 이동한다.

18 ~ 20 정우측에 사다리가 없는 경우로 바로 한 칸 이동한다.

P36


```
67  for (i = 0; i < 100; i++)
68      for (j = 0; j < 100; j++)
69          scanf("%d", &map[i][j]);
70
71  // 모든 경우 찾기
72  for (i = 0; i < 100; i++) {
73      if (map[0][i] == 2) {
74          tmp = ans(i);
75          n = tmp[0];
76      }
77  }
78
79  printf("%d\n", ans(n)[0], tmp[1] + 1);
80
81  return 0;
82  }
83
```

6 ~ 11 오른쪽 사다리가 있으면 끝까지 이동한 뒤 바로 한 칸 이동한다.

12 ~ 17 우측에 사다리가 있으면 끝까지 이동한 뒤 바로 한 칸 이동한다.

18 ~ 20 정우측에 사다리가 없는 경우로 바로 한 칸 이동한다.

P36