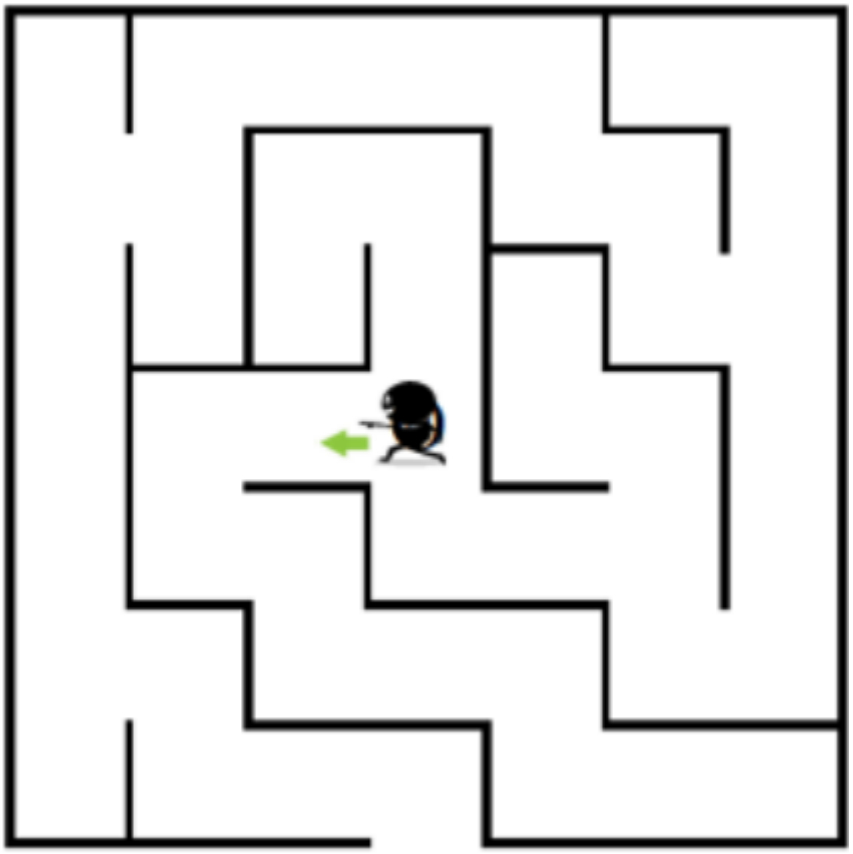
알고리즘 설계 구현 결과 **1: <**가장 큰 수 찾기**>**

<문제 정의**>**

문제이름**:** 미로 찾기

문제설명**:** 주어진 미로 **(**배열**)**을 받고 결승점을 찾는다

문제예시 : 다음과 같이 미로가 설정 되어있을 때 결승점을 찾는다.



<해결 아이디어**>**

(해결아이디어**1) :** 임민규

재귀함수를 이용하여 길을 찾는다, 개미가 있는 현 좌표에서 상하좌우 좌표에서 갈 수 있는 길이 있는지 탐색을 반복한후 미로에 존재하는 결승점에 도착한다

(해결아이디어**2) :** 이주형

스택을 이용하여 **push**로 좌표를 저장하고 앞으로 가다가 막히게 된다면 **pop**을 이용해 다시 뒤로 간다**.**

아이디어 채택: 의논한 결과 아이디어 1을 채택.

<알고리즘>

명칭: 미로찾기

입력: 배열로 이루어진 미로

출력: 결승점 탐색 성공여부

처리순서:

미로 배열 초기 설정을 한다.

개미의 좌표를 (1,1)로 설정 한 후, (x-1, x+1, y-1, y+1) 에 대한 코드를 설계한다 (재귀함수

point 변수를 통해 처리된 데이터를 다시 계산하지 않도록 한다.

<코드 설계>

1. 자료 구조 정의

maze (미로 배열 초기 설정)

point (개미의 현 위치)

2. 함수 정의

- find_way(int x, int y)

재귀함수를 통해 goal(6,6) 지점을 찾을 때 까지 반복해서 명령을 실행한다.

point 변수를 통해 처리된 미로의 좌표는 재계산하지 않도록 미로 배열의 벽과 같은 '1'로 변수를 변환한다.

3. 테스트 케이스 설계

<C코드 구현 결과>

```
#include <stdio.h>

// 미로의 크기
#define SIZE 8

// 미로 배열 초기 설정
int maze[SIZE][SIZE] = {
{1, 1, 1, 1, 1, 1, 1, 1},
{1, 0, 0, 0, 0, 0, 0, 1},
```

```
{1, 0, 1, 1, 0, 1, 0, 1},
{1, 0, 1, 0, 0, 0, 0, 1},
{1, 0, 1, 0, 1, 1, 0, 1},
{1, 0, 1, 0, 1, 0, 0, 1},
{1, 0, 0, 0, 1, 0, 0, 1},
{1, 1, 1, 1, 1, 1, 1, 1}
};
//현재 위치
int point[SIZE][SIZE];

void find_way(int x, int y)
{
// 도착점에 도달한 경우
if (x == SIZE-2 && y == SIZE-2) {
printf("goal!\n");
maze[SIZE - 2][SIZE - 2] == 3;
return;
}
else {
// 현재 위치를 방문한 것으로 표시
point[x][y] = 1;

if (maze[x - 1][y] == 0 && point[x - 1][y] == 0) // 상
find_way(x - 1, y);
if (maze[x + 1][y] == 0 && point[x + 1][y] == 0) // 하
find_way(x + 1, y);
if (maze[x][y - 1] == 0 && point[x][y - 1] == 0) // 좌
find_way(x, y - 1);
if (maze[x][y + 1] == 0 && point[x][y + 1] == 0) // 우
find_way(x, y + 1);

}
}
```

```
int main()
{
printf("start\n");
//시작 포인트는 1.1
find_way(1, 1);
return 0;
```

-TC1

}

goal 지점이 있는 문제없는 경우

{1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 1, 1, 0, 1, 0, 1},

{1, 0, 1, 0, 0, 0, 0, 1},

{1, 0, 1, 0, 1, 1, 0, 1},

{1, 0, 1, 0, 1, 1, 0, 1},

{1, 0, 0, 0, 0, 1, 0, 1},

{1, 1, 1, 1, 1, 1, 1, 1}

-> goal! 출력 성공

-TC2

2. goal 지점에 도착할 수 없는 경우

{1, 1, 1, 1, 1, 1, 1, 1},

{1, 0, 0, 0, 0, 0, 0, 1},

{1, 0, 1, 1, 0, 1, 0, 1},

{1, 0, 1, 0, 0, 0, 0, 1},

{1, 0, 1, 0, 1, 1, 0, 1},

{1, 0, 1, 0, 1, 1, 1, 1},

{1, 0, 0, 0, 0, 1, 0, 1},

{1, 1, 1, 1, 1, 1, 1, 1}

-> goal 출력 실패

알고리즘 설계 구현 결과 **1: <**가짜 동전 찾기**>**

<문제 정의**>**

문제이름**:** 가짜 동전 찾기

문제설명**:** 주어진 동전**(**정수**)**중 가짜 동전을 찾는다**.**

문제예시 : 아주 많은 동전 더미 속에 있는 1개의 가짜 동전을 찾는다.

가짜 동전의 무게는 정상적인 동전보다 약간 가볍다.

<해결 아이디어**>**

(해결아이디어**1) :** 이주형

동전을 하나씩 비교하여 순차탐색으로 찾는다.

(해결아이디어**1) :** 임민규

배열 이진탐색을 활용하여 동전의 좌 우의 값을 비교하며 무게가 다른 동전을 찾는다.

아이디어 채택: 의논한 결과 아이디어 **1**을 채택.

<알고리즘**>**

명칭**:** 가짜 동전 찾기

입력**:** 동전의 개수

출력**:** 몇 번째에 가짜 동전이 있는지

처리순서**:**

가짜 동전이 어디에 있는지 fake에 넣는다.

a부터 b까지 동전과 c에서 d까지의 동전 무게를 비교한다.

가짜 동전이 a, b사이에 있으면 –1 c, d사이에 있으면 1 가짜 동전이 없으면 0을 반환 한다.

이 과정을 반복 한다.

<코드 설계**>**

1. 자료 구조 정의

동전의 개수: n

가짜 동전의 위치: fake

2. 함수 정의

기능:weight 함수를 이용하여 저울을 만들어준다.

프로토타입: int weight(int a, int b, int c, int d)

기능: find_fakecoin(int left, int right) 함수를 만들어 가짜 동전을 찾는다.

3. 테스트 케이스 설계

TC1. 무게를 잴 저울 함수 weight를 선언하고 find_fakecoin 함수를 만들어 가짜 동전의 위치를 찾는다.

<C코드 구현 결과**>**

//가짜 동전 찾는 프로그램

```
#include <stdio.h>

int weight(int a, int b, int c, int d){

int fake = 55;

if(a <=fake && fake <= b)

return -1;

if(c <= fake && fake <= d)

return 1;

return 0;

}

int find_fakecoin(int left, int right){

int i = left + 1;

for (i; i<= right; i++){

int result = weight(left, left, i, i);

if (result == -1){

return left;

}

else if(result ==1){

return i;

}

}

return -1;
```

```
}


int main(void)

{

int n = 100;

printf("%가짜 동전의 위치는: d\n", find_fakecoin(0, n-1));

return 0;
```

<코드 저장소 링크**>**

조원 1(임민규): https://github.com/m1ngu01/-Algorithm.git

조원 2(이주형): https://github.com/obongbong/algorithm-2.git