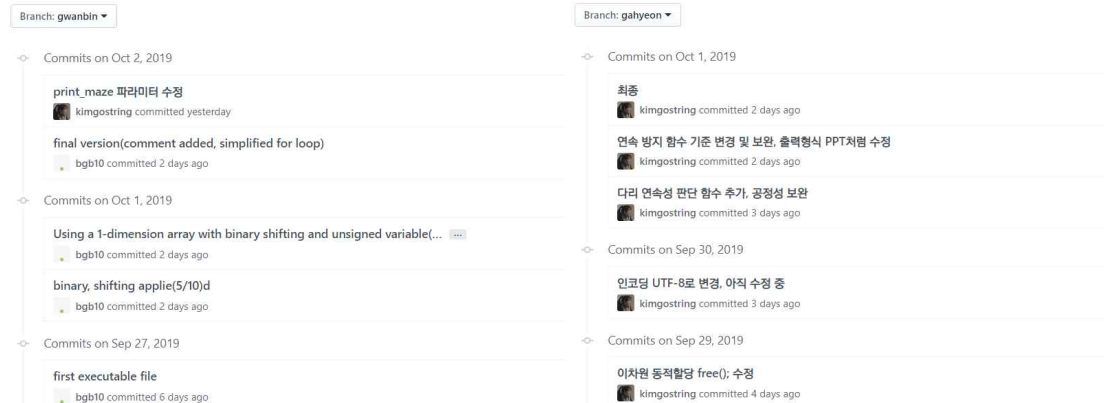


I. 회의 자료

대부분의 회의는 오프라인에서 진행되었으며, 코드는 각자 맡은 부분을 따로 작업하였다. 코딩 협업은 github를 이용하여 진행하였다.



[그림 1, 2] : 이번 과제 협업에 사용된 github의 commit 기록 일부.

2019. 09. 27 (금)

- 사다리 및 미로를 만들 방법을 간단히 얘기해보았다.
- 사다리와 미로를 만드는 세부 조건들은 일단 고려하지 않고, 프로그램을 완성한 뒤 덧붙여 가기로 했다.
- 사다리의 경우, 가로 크기가 [알파벳 수 - 1]인 배열을 만들고 가로선을 그을 유무를 0과 1을 이용해 저장하기로 했다. 사다리 결과는 for문을 이용하여 양옆의 0, 1의 유무를 확인해가며 위에서부터 한 칸씩 내려가며 계산하기로 했다. 정적 지역 변수를 활용하여 결과 계산 함수가 호출될 때마다 지역 변수가 초기화되지 않도록 하기로 했다.
- 미로의 경우, 밖의 틀을 저장하지 않고 0과 1을 통해 선을 그을 유무를 저장하기로 했다.

2019. 10. 01 (화)

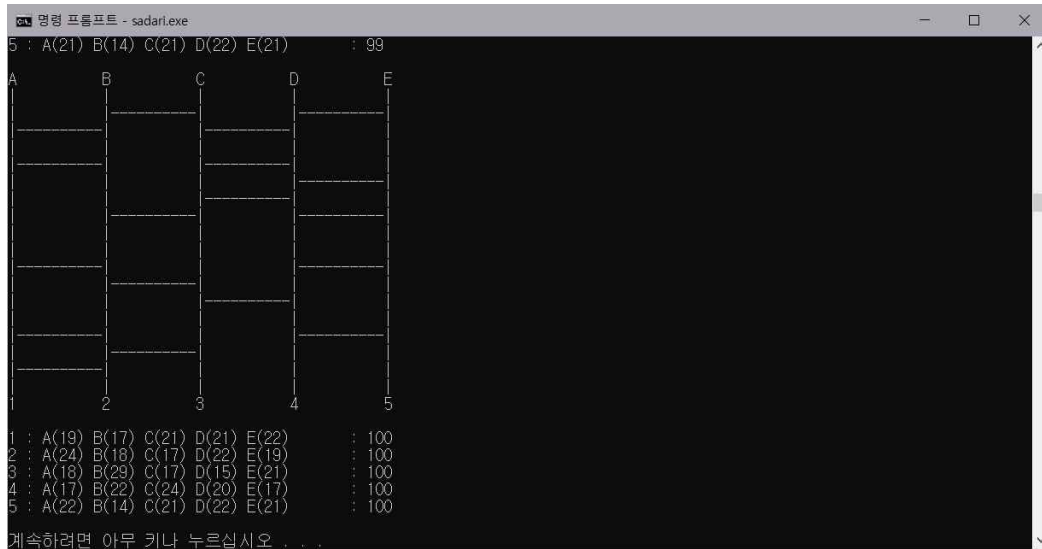


[사진 1] : 오프라인 회의를 진행하는 모습.

- 완성한 프로그램에 아이디어를 추가하여 세부 조건들을 충족시키기로 했다. 공정한 사다리와 적은 배열을 이용한 미로를 위한 아이디어들은 II, III에서 보다 자세히 서술하겠다.

II. 사다리 과제

1. 실행화면

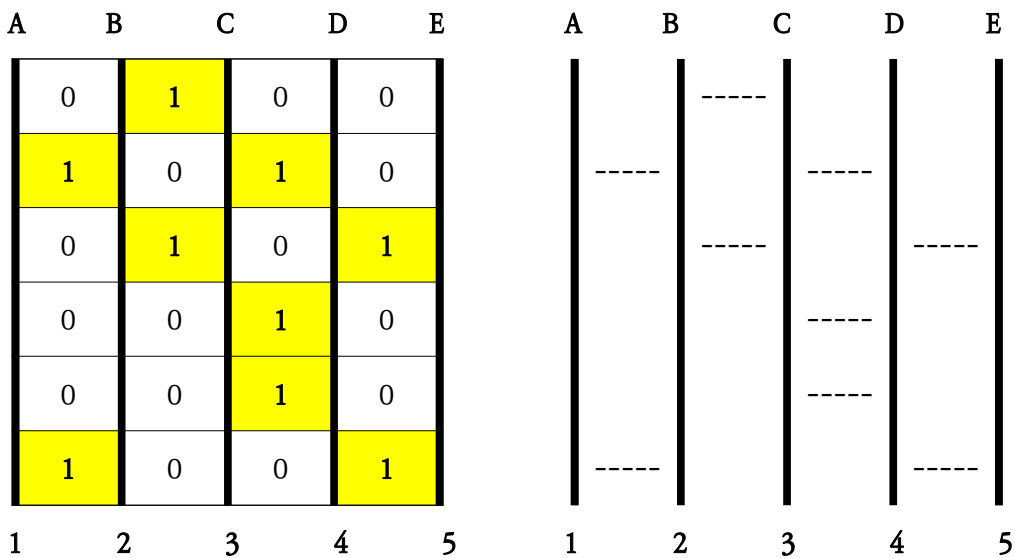


[그림 3] : cmd 창에서 사다리 프로그램을 실행시킨 화면.

2. 문제와 그 해결 방안

가. 랜덤하고, 특정 경향성을 갖지 않는 사다리를 그려야 한다.

- 1) 가로선의 크기가 [알파벳 수 - 1]인 이차원 배열을 만든다. 배열의 각 칸에는 0과 1을 이용하여 두 세로선을 잇는 가로선의 유무를 저장한다.
- 2) 난수 값을 time(NULL)로 설정하고, rand 함수를 통해 프로그램 실행 때마다 매번 다른 결과가 나오도록 가로선의 위치를 추천한다.
- 3) 만일 선택된 좌표에 1이 저장되어 있다면, 다시 한번 위치를 추천한다.
- 4) 하나의 사다리에 위와 같은 과정을 15회 이상 반복한다.



[그림 4] : 가로선을 저장하는 배열의 모습 (좌), 실제 가로선이 그려진 모습 (우).

```

for (i = 0; i < DARI_NUM; i++) {
    do {
        do {
            select_x = (rand() % (PEOPLE - 1));
            select_y = (rand() % Y_NUM);
        } while (arr[select_y][select_x] != 0); // 중복 방지 (다리 개수 늘 18개 유지)
    } while (1);
}

```

[그림 5] : 코드로 구현한 모습.

나. 애매한 선을 굵지 않아야 한다.

- 1) 선택된 가로선의 좌표가 (x, y)이고 $x \neq 0$ 이라면(가장 오른쪽 세로선을 잇는 가로선이 아니라면), (x - 1, y)의 위치에 0이 저장되어 있지 않다면 다시 한번 위치를 추천한다.
- 2) 선택된 가로선의 x 좌표가 $x \neq [\text{알파벳 수} - 3]$ 이라면(가장 왼쪽 세로선을 잇는 가로선이 아니라면), (x + 1, y)의 위치에 0이 저장되어 있지 않다면 다시 한번 위치를 추천한다.
- 3) 만일 x 좌표가 0도 아니고 $[\text{알파벳 수} - 3]$ 도 아닌 경우 1), 2)를 모두 고려한다.

A	B	C	D	E	A	B	C	D	E
					1	0			
1	2	3	4	5	1	2	3	4	5

[그림 6] : 다시 한번 위치를 추천하는 경우 (좌), 위치를 다시 추천하지 않아도 되는 경우 (우).

```

if (select_x == 0) { // x좌표 0일 때, 오른쪽 다리 유무만 고려하면 됨
    if (arr[select_y][select_x + 1] != 1) {
        arr[select_y][select_x] = 1;
    }
}
else if (select_x == PEOPLE - 2) { // x좌표 PEOPLE - 2일 때, 왼쪽 다리 유무만 고려하면 됨
    if (arr[select_y][select_x - 1] != 1) {
        arr[select_y][select_x] = 1;
    }
}
else { // x좌표가 위의 경우가 아닐 때, 둘 다 고려해야 함
    if ((arr[select_y][select_x + 1] != 1) && (arr[select_y][select_x - 1] != 1)) {
        arr[select_y][select_x] = 1;
    }
}
}

```

[그림 7] : 코드로 구현한 모습.

다. 모든 인접한 두 개의 세로선 사이에 최소 3개의 가로선이 있어야 한다.

- 최소 개수를 따지는 일은 다리를 다 완성한 이후에만 판단 가능하다. 그러나 그렇게 이렇게 코딩한다면 불필요한 계산이 많아지고 프로그램 실행 속도가 느려질 것이다.

- 따라서, 두 세로선을 잇는 가로선이 (int)([추첨할 가로선 수 - 3] % [알파벳 수 - 2]) 개 이하가 되도록 문제를 바꾸어 푸는 것이 더 효율적이다.

- 1) for문을 통해 선택된 가로선과 x 좌표가 동일한 모든 곳의(선택된 가로선 자신도 포함) 1 유무를 확인하여, 만일 해당 위치에 1이 있다면 0으로 초기화된 특정 변수에 1씩 더한다.
- 2) for문을 다 돈 이후, 특정 변수의 값이 기준값을 넘으면 다시 한번 위치를 추첨한다.

A	B	C	D	E	A	B	C	D	E
	0					0			
	1					1			
	1					0			
	0					0			
	1					1			
	1					0			
1	2	3	4	5	1	2	3	4	5

※ 그림에서, 세로 칸에 가능한 최대 가로선 개수는 3이라고 가정

[그림 8] : 다시 한번 위치를 추첨하는 경우 (좌), 위치를 다시 추첨하지 않아도 되는 경우 (우).

```

comparison = 0;
for (j = 0; j < Y_NUM; j++) { // 한 다리에 올 수 있는 최대 개수 판단
    if (arr[j][select_x] == 1) {
        comparison++;
    }
}
if (comparison > NUM_MAX) {
    arr[select_y][select_x] = 0;
}

```

[그림 9] : 코드로 구현한 모습.

do~while문의 조건이 `arr[select_y][select_x] == 0`이므로 다시 좌표를 추첨한다.

라. 가로선은 특정한 곳에 집중되어서는 안 된다.

- ‘가로선이 특정 곳에 집중된다’는 문제를 a와 b로 나누어 고려해보았다.

a. x 좌표가 같고 y 좌표가 연속된 지점의 수가 특정 기준값을 초과하지 않아야 한다.

- 1) 추첨된 좌표가 (x, y)일 때, for문을 통해 0부터 [기준값] - 1까지 k를 1씩 변화시키며 (x, y - ([기준값] - 1) + k)의 좌표에 1이 들어있는지 확인한다. 1이 있는 경우 초깃값이 0인 특정 변수에 1씩을 더한다. 이때 y 좌표가 배열의 범위를 넘어가는지 잘 확인한다.
- 2) 1)를 감싸고 있는 바깥쪽 for문을 통해, k의 초깃값을 1씩 증가시키며 k가 기준값과 같아질 때까지 1), 2)를 반복한다.
- 3) 특정 변수의 값이 기준값과 같아진다면, 좌표를 다시 추첨한다.

A	B	C	D	E
	0	첫 번째 for		
	1	두 번째 for		
	1	세 번째 for		
	1			
	0			
	1			
1	2	3	4	5

A	B	C	D	E
	0			
	0			
	1			
	0			
	1			
	1			
1	2	3	4	5

※ 그림에서, 연속 가능한 최대 가로선 개수는 2라고 가정

[그림 10] : 다시 한번 위치를 추천하는 경우 (좌), 위치를 다시 추천하지 않아도 되는 경우 (우).

```

for (j = 0; j < SERIES_MAX; j++) { // 연속성 판단 (다닥다닥 다리 올 수 없음)
// 이 함수를 통해 다닥다닥 오는 개수를 변경 가능
    comparison = 0;
    if ((select_y - (SERIES_MAX - 1) + j >= 0) && (select_y + j < Y_NUM)) {
        for (k = j; k < j + SERIES_MAX; k++) {
            if (arr[select_y - (SERIES_MAX - 1) + k][select_x] == 1) {
                comparison++;
            }
        }
        if (comparison == SERIES_MAX) {
            arr[select_y][select_x] = 0;
            break;
        }
    }
}

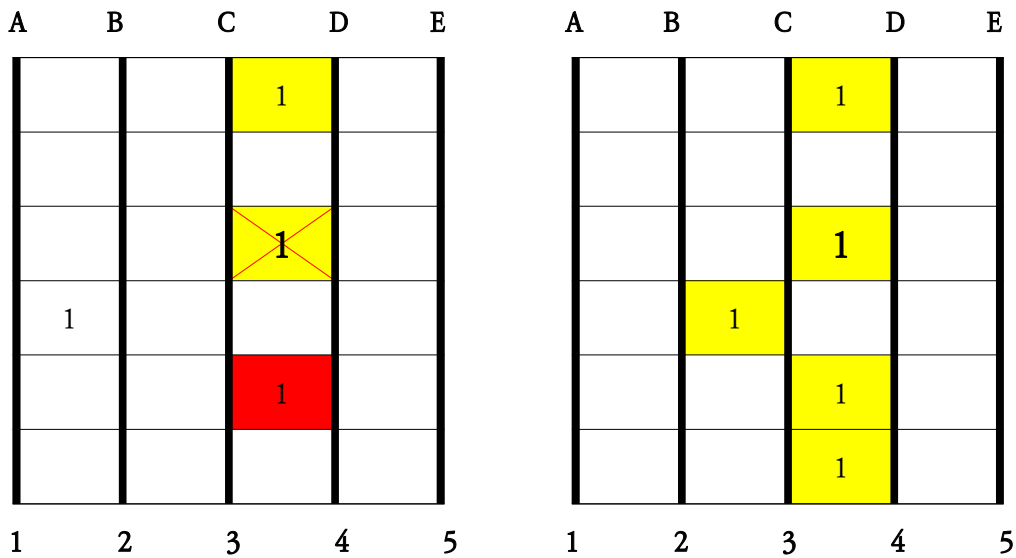
```

[그림 11] : 코드로 구현한 모습.

do~while문의 조건이 arr[select_y][select_x] == 0이므로 다시 좌표를 추천한다.

b. x 좌표가 같은 가로선 사이사이 인접한 x 좌표를 가진 가로선들이 위치해야 한다.
 사이사이 가로선이 없다면, 그 줄의 가로선은 기준 개수를 초과할 수 없다.

- 1) 추천된 좌표가 (x, y)라면, for문을 통해 y 좌표의 값을 0부터 1씩 키워가며 x 좌표의 값이 같은 지점에서 가장 먼저 1이 나오는 곳의 y 좌표를 알아낸다. 이 for문은 2)가 실행되기 전까지만 돌아간다.
- 2) 알아낸 y 좌표에서부터 또 다른 for문을 돌려, 만일 (x, k) 지점에 1이 들어있으면 0으로 초기화된 특정 변수에 1씩 더하고, (x - 1, k)나 (x + 1, k) 지점에 1이 들어있으면 특정 변수에 0을 넣는다.
- 3) 만일 2)의 for문이 돌아가던 중 특정 변수의 값이 기준 개수를 넘는다면, 좌표를 다시 추천한다.



[그림 12] : 다시 한번 위치를 추천하는 경우 (좌), 위치를 다시 추천하지 않아도 되는 경우 (우).

```

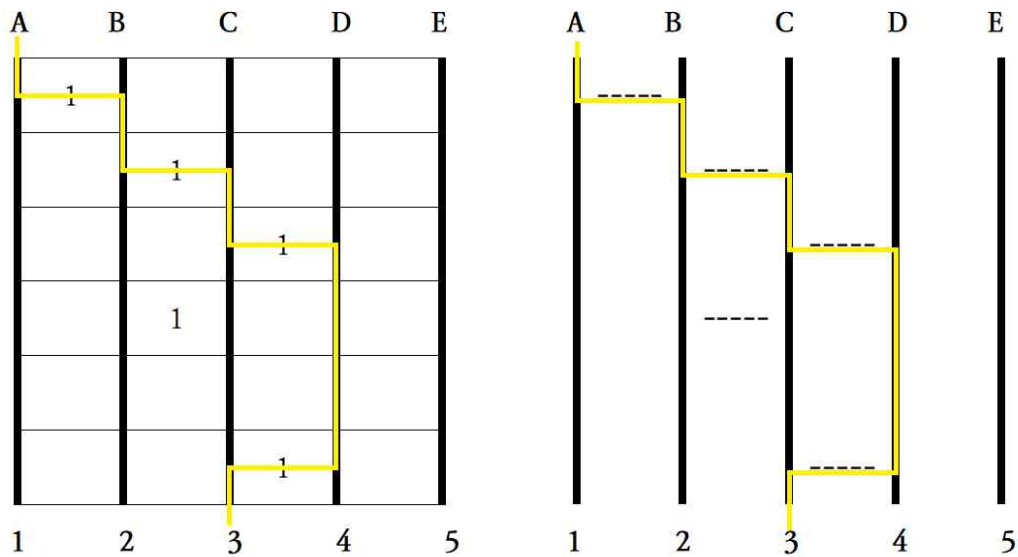
comparison = 0;
for (j = 0; j < Y_NUM; j++) { // 연속성 판단 2 (최대 3개까지만 근처에 다른 다리들 없)
    if (arr[j][select_x] == 1) {
        for (k = j; k < Y_NUM; k++){
            if (arr[k][select_x] == 1) {
                comparison++;
            }
            else if (((select_x != 0) && (arr[k][select_x - 1] == 1))
                || ((select_x != PEOPLE - 2) && (arr[k][select_x + 1] == 1))) {
                comparison = 0;
            }
            if (comparison > SERIES_MAX_2) {
                arr[select_y][select_x] = 0;
                break;
            }
        }
        break;
    }
}
} while (arr[select_y][select_x] == 0);

```

[그림 13] : 코드로 구현한 모습.

마. 사다리 결과를 계산하고, 누계를 반영한 도착 위치의 통계를 출력해야 한다.

- 1) 내려갈 알파벳을 선택하고, 알파벳에 해당하는 세로줄을 타고 한 칸씩 내려온다.
- 2) 만일 내려가는 세로줄이 첫 번째 세로줄이 아니고 오른쪽에 가로선이 그어져 있으면, 세로줄을 오른쪽으로 하나 옮기고 한 칸 내려온다. 또한
- 3) 만일 내려가는 세로줄이 마지막 세로줄이 아니고 왼쪽에 가로선이 그어져 있으면, 가로줄을 왼쪽으로 하나 옮기고 한 칸 내려온다.
- 4) 맨 아래에 도착할 때까지 for문을 통해 이를 반복한다.
- 5) for문을 통해 위의 과정을 알파벳 총 개수만큼 반복한다. 통계를 저장할 이차원 배열을 static으로 선언하여, 함수를 호출할 때마다 지역 변수가 초기화되지 않도록 한다.



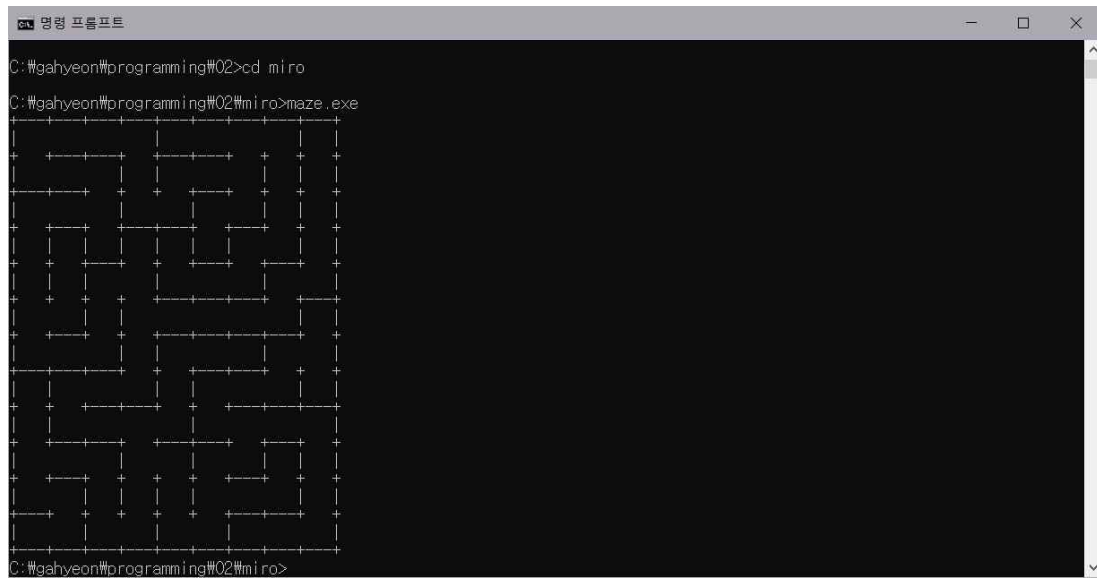
[그림 14] : 배열 안 1의 유무에 따라 결정되는 사다리 결과.

```
void result(int **arr) {
    static int count[PEOPLE][PEOPLE + 1] = { 0 }; // static으로 저장, 여러 번 불러도 값 초기화 안 되도록 (100번 통계 위함)
    // count[시작(알파벳)][결과(숫자)], 결과 쪽 마지막 배열에는 각 결과 더한 값 저장
    int i, j, temp; // temp는 사다리 내려가며 위치 저장해두는 변수, A와 같은 줄이 0, E와 같은 줄이 4
    for (i = 0; i < PEOPLE; i++) {
        temp = i; // A부터 사다리 타기 시작
        for (j = 0; j < Y_NUM; j++) { // 알파벳에서부터 점점 내려오며(j), 양쪽(혹은 한쪽)의 다리 유무 확인
            if ((arr[j][temp] == 1) && (temp != PEOPLE - 1)) {
                temp++; // temp의 오른쪽에 사다리 있음 한 칸 오른쪽 이동
            }
            else if ((arr[j][temp - 1] == 1) && (temp != 0)) {
                temp--;
            }
        }
        count[i][temp] = count[i][temp] + 1; // i는 abcde, temp는 12345 결정
    }
    for (i = 0; i < PEOPLE; i++) {
        printf("%d : ", i + 1);
        for(j = 0; j < PEOPLE; j++){
            printf("%c(%d) ", 'A' + j, count[j][i]);
        }
        count[i][PEOPLE]++;
        printf("\t: %d\n", count[i][PEOPLE]);
    }
    printf("\n");
}
```

[그림 15] : 코드로 구현한 모습.

II. 미로 과제

1. 실행화면



```
C:\gahyeon\programming\02>cd miro
C:\gahyeon\programming\02\miro>maze.exe
```

The screenshot shows a Windows command prompt window titled "명령 프롬프트". The user has navigated to the directory "C:\gahyeon\programming\02\miro" and executed the command "maze.exe". The output of the program is a maze represented by a grid of '+' characters. The maze is approximately 15 columns wide and 15 rows high. The path through the maze is indicated by a series of '+' characters that form a continuous line from the top-left corner to the bottom-right corner. The command prompt shows the prompt "C:\gahyeon\programming\02\miro>" at the bottom.

[그림 16] : cmd 창에서 미로 프로그램을 실행시킨 화면.