1. 실험시간에 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술하시오. 완성한 알고리즘의 시간 및 공간복잡도를 보이고, 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술하시오.

```c
#include <stdio.h>
#include [redacted]

int main(void) {

    /* Get WIDTH, HEIGHT */
    get_size();

    /* Allocate memory */
    allocate_memory();

    /* Make .maz file */
    open_maze_file();

    /* Draw the top horizontal wall.
       It should be blocked. */
    draw_border();

    /* Exception: If HEIGHT==1, then maze should not be blocked. */
    if (HEIGHT == 1) {
        draw_vertical();
        draw_border();
        finish_program();
        return 0;
    }

    /* Randomization */
    srand((unsigned)time(NULL));

    /* 첫 번째 라인의 세로 벽을 세우며 group을 정한다. */
    set_first_line();

    /* 첫 번째 라인의 세로 벽을 그린다. */
    draw_vertical();

    /* 각 라인마다 아래로 뚫고, 가로 선 그리고, 옆으로 뚫고, 세로 선 그리기를 반복 */
    for (int i = 1; i < HEIGHT; i++) {
        /* 아래로 뚫을 곳을 결정해 뚫는다. */
        break_down();
        /* 가로 선을 그린다. */
        draw_horizontal();
        /* 옆으로 뚫을 곳을 결정해 뚫는다. */
        if (i != HEIGHT-1) {
            /* 중간 라인들의 경우, 어떤 곳을 뚫을지 랜덤으로 결정해 뚫는다. */
            break_side();
        }
        else {
            /* 마지막 라인의 경우, 다른 집합끼리 인접하면 무조건 뚫는다. */
            break_side_last();
        }
        /* 세로 선을 그린다. */
        draw_vertical();
    }

    /* Draw the bottom horizontal wall.
       It should be blocked. */
    draw_border();
```

```
    /* Close .maz file and deallocate memory. */
    finish_program();

    return 0;
}
```

① get_size(): 사용자에게 stdin으로 미로의 가로 길이(WIDTH), 세로 길이(HEIGHT)를 입력받아 저장한다. 그리고 WIDTH, HEIGHT가 0보다 작다면 프로그램을 종료한다.

② allocate_memory(): 가로 벽, 세로 벽, 행의 각 칸이 속하는 집합 정보를 저장할 메모리를 동적 할당하고, 할당에 실패했다면 프로그램을 종료한다.

③ open_maze_file(): 메이즈를 출력할 .maz 확장자의 파일을 오픈한다. 만약 파일 오픈에 실패하면 프로그램을 종료한다.

④ draw_border(): 미로의 가장 위쪽 가로 벽을 그린다. 모두 막혀 있어야 한다.

⑤ 미로의 세로 길이가 1인 경우, 내부가 모두 뚫린 미로를 만들어야 한다. 따라서 모두 뚫린 세로 벽을 그리고, 가장 아래 가로 벽을 그려서 마무리한다.

⑥ Eller's algorithm에서 뚫을 벽을 랜덤하게 선택해야 하고, 프로그램을 실행할 때마다 다르게 랜덤한 선택을 해야 하므로 srand()를 통해 seed 값을 초기화한다.

⑦ set_first_line(): 첫 번째 라인은 랜덤하게 세로 벽을 세우며 그룹을 정한다.

⑧ draw_vertical(): set_first_line() 함수에서 정한대로 세로 벽을 한 줄 그려준다.

⑨ 두번째 행부터 마지막 행까지 for문을 돌며 아래 방향으로 벽을 뚫고, 가로 벽을 그리고, 옆 방향으로 벽을 뚫고, 세로 벽을 그리는 작업을 반복한다. 단, 가장 마지막 행은 다른 집합끼리 인접할 경우 모두 뚫어야 미로 전체의 모든 칸이 한 집합에 속하게 된다.

i. break_down(): 미로의 가로 길이(WIDTH)만큼 각 칸에 대해 for문을 돌며 수행한다. 랜덤하게 뚫거나 벽을 세우고, 같은 그룹 내에서 최소 한 번은 아래로 벽을 뚫어야 하기 때문에, broken이라는 변수를 사용해 그룹 내에서 벽을 뚫은 과거가 있는지 기억하도록 했다. 이 함수의 시간복잡도는 O(WIDTH * WIDTH)이다.

```
void break_down(void) {
    /* +-+-+-+-+-+ 중에서 어느 -를 뚫을 것인가 */
    int prev_group = maze_set[0];
    int broken = 0;
    int i;

    for (i = 0; i < WIDTH; i++) {
        horizontal[i] = rand() % 2; //0(뚫는다) or 1(막는다)
        if (!horizontal[i]) { //뚫었다면 뚫었다고 표시한다.
            broken = 1;
        }
        if (horizontal[i]) { //막힌 곳은 다른 group으로 지정한다.
```

```
            maze_set[i] = group;
            group++;
        }
        if (i < WIDTH-1) {
            if (prev_group != maze_set[i+1]) {
                if (!broken) { //한 번도 뚫지 않았다면 최소 한 번은 뚫어야 한다.
                    horizontal[i] = 0;
                }
                else {
                    broken = 0;
                }
                prev_group = maze_set[i+1];
            }
        }
    }
    if (!broken) { //끝까지 한 번도 뚫지 않았다면 뚫어줘야 한다.
        horizontal[i-1] = 0;
    }
}
```

ii. draw_horizontal(): break_down() 함수에서 정한 대로 가로 벽을 한 줄 그린다.

iii. break_side() 또는 break_side_last(): 가로 길이가 WIDTH라면, 가장 왼쪽과 가장 오른쪽의 세로 벽은 반드시 세우기 때문에 WIDTH-1만큼 각 칸에 대해 for문을 돌며 수행한다. 같은 그룹끼리 인접할 경우에는 벽을 세워두고, 다른 그룹끼리 인접할 경우에는 랜덤하게 뚫거나 벽을 세운다. 만약 뚫게 되었을 경우 같은 group으로 만들어준다. 이 함수의 시간복잡도는 O(WIDTH * WIDTH)이다.

```
void break_side(void) {
    /* | | | | | | |에서 어떤 |를 뚫어줄 것인가 */
    int break_flag = 0;
    int prev_group;
    int i, j;

    for (i = 0; i < WIDTH-1; i++) {
        //같은 그룹끼리 인접할 때는 벽을 제거하지 않는다.
        if (maze_set[i] == maze_set[i+1]) {
            vertical[i] = 1;
        }
        //다른 그룹끼리 인접할 때는 벽을 제거할지, 막아둘지 랜덤으로 결정한다.
        else {
            break_flag = rand() % 2; //0(뚫는다) or 1(막는다)
            //뚫는 경우
            if (break_flag) {
                vertical[i] = 0; //벽을 뚫는다.
                prev_group = maze_set[i+1]; //합치기 전 그 그룹의 번호를
저장해둔다.
                maze_set[i+1] = maze_set[i]; //같은 group으로 합친다.
                for (j = 0; j < WIDTH; j++) { //prev_group의 번호를 가진 group을
모두 합쳐준다.
                    if (maze_set[j] == prev_group)
                        maze_set[j] = maze_set[i];
                }
            }
            //막는 경우
            else {
                vertical[i] = 1; //벽을 세워둔다.
            }
        }
```

```
        }
    }
```

한편, 마지막 가로 줄의 세로 벽은 별도의 처리를 해줘야 한다. 다른 집합이 인접했을 경우에 랜덤하게 벽을 뚫는 것이 아니라 항상 벽을 제거해서, 미로의 모든 칸이 같은 그룹에 속하도록 해야 한다. 이 함수의 시간복잡도는 O(WIDTH * WIDTH)이다.

```
void break_side_last(void) {
    /* | | | | | | |에서 어떤 |를 뚫어줄 것인가 */
    /* 가장 마지막 라인이므로, 다른 집합이 인접했다면 무조건 뚫어준다. */
    int prev_group;
    int i, j;

    for (i = 0; i < WIDTH-1; i++) {
        //같은 그룹끼리 인접할 때는 벽을 제거하지 않는다.
        if (maze_set[i] == maze_set[i+1]) {
            vertical[i] = 1; //벽을 세워둔다.
        }
        //다른 그룹끼리 인접할 때는 벽을 제거한다.
        else {
            vertical[i] = 0; //벽을 뚫는다.
            prev_group = maze_set[i+1]; //합치기 전 그 그룹의 번호를 저장해둔다.
            maze_set[i+1] = maze_set[i]; //같은 group으로 합친다.
            for (j = 0; j < WIDTH; j++) { //prev_group의 번호를 가진 group을 모두
합쳐준다.
                if (maze_set[j] == prev_group)
                    maze_set[j] = maze_set[i];
            }
        }
    }
}
```

iv. draw_vertical(): break_side() 또는 break_side_last()에서 정한 대로 세로 벽을 그린다.

⑩ draw_border(): 미로의 가장 아래쪽 가로 벽을 그린다. 모두 막혀 있어야 한다.

⑪ finish_program(): .maz 파일을 닫고, 동적 할당된 메모리를 해제한다.

예비보고서 작성 시, Eller's algorithm으로 미로를 생성할 시 세로 벽, 가로 벽, 각 칸이 속하는 집합을 저장할 O(WIDTH)의 배열 3개가 필요하다고 생각했다. 또한 각 행의 칸에 대해 살펴보는 작업을 모든 행에 대해 살펴보기 때문에 시간복잡도는 O(WIDTH * HEIGHT), 공간복잡도는 O(WIDTH)이라고 생각했다. 공간복잡도는 예상한 바와 동일하다. 그런데 시간복잡도는 조금 다르게 계산된다. 위에서 나타낸 대로 O(WIDTH * WIDTH)의 시간복잡도를 가지는 함수를 O(HEIGHT)만큼 호출하기 때문에 전체 알고리즘의 시간복잡도는 $O(WIDTH^2 * HEIGHT)$가 된다. 예비보고서 작성 시에는 이전 칸들의 group 정보를 변경해주는 시간을 고려하지 않았기 때문에 계산의 차이가 발생한 것으로 보인다.