컴퓨터공학실험1　11주차 결과보고서

전공: 컴퓨터공학과　　학년: 2　　학번: 20191559　　이름: 강상원

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

예비보고서의 내용과 상당 부분 서술이 겹친다.

Eller's algorithm을 사용하였다.

자료구조 기술:

```
int WIDTH, HEIGHT;

vector<int> horizontal_wall; //가로 벽 그리는 배열->포인터
vector<int> vertical_wall; //세로 벽 그리는 배열->포인터
vector<int> maze; //전체 배열

int Id = 0; //방에 적힌 숫자 의미
```

아래에 서술하는 세 배열들은 C++의 vector를 이용해 선언하였다.

int maze[WIDTH]: 미로의 전체 한 줄(열) 정보를 저장하는 배열. 각 칸 안의 Id (숫자)를 저장하는 역할을 한다.

int horizontal_wall[WIDTH]: 현재 열 (가로 줄)과 다음 열 사이에 있는 벽 정보 저장하는 동적 배열. 벽이 있을시 1, 없을 시 0 저장.

int vertical_wall[WIDTH-1]: 현재 열 안에서 서로 인접한 두 칸끼리의 벽을 의미한다. 사이의 벽이므로 WIDTH-1개 존재한다. horizontal_wall과 마찬가지로 벽이 존재하면 1, 존재하지 않으면 0을 저장한다.

Id: 각 방에 적힌 숫자를 의미한다. 두 방(칸) 사이에 경로가 존재하면 같은 칸으로 간주하고, 내부의 번호가 같다.

열마다 각 행에 대해 비교 연산을 진행해야 하므로 **시간 복잡도는** O(WIDTH*HEIGHT)이다.

공간 복잡도는 O(WIDTH)가 될 것이다. (한 열씩만 저장하므로)

```
void first_rand() {
    srand(time(NULL));
    for (int i = 0; i < WIDTH - 1; i++) {
        vertical_wall[i] = rand() % 2; //0 아니면 1
        if (vertical_wall[i] == 1) { //1 일 경우, 막혀 있다는 의미이므로 그 이전
칸까지 모두 뚫는 처리.
            for (int j = (before_wall + 1); j <= i; j++)
                maze[j] = Id; //방에 적힌 숫자 통일

            before_wall = i; // 이전 벽 위치
```

```
            Id++; // 이제 다음 집합에 적힌 숫자는, 1 증가.
        }
    }
    for (int i = before_wall + 1; i < WIDTH; i++)
        maze[i] = Id; //마무리 처리->나머지 곳도 숫자 맞추기.
    Id++;
}
```

<main 함수>

```
int main() {
    fp = fopen("mazeRes.maz", "wt");

    //미로의 너비, 높이 입력받기
    cin >> WIDTH >> HEIGHT;

    horizontal_wall.resize(WIDTH);
    vertical_wall.resize(WIDTH-1);
    maze.resize(WIDTH, 0);

    //첫 line
    line_feed();
    first_rand();
    ver_write();

    for (int k = 1; k < HEIGHT - 1; k++) { //반복
        vertical_meet();
        hor_write();
        meet_rand();
        ver_write();
    }

    //마지막 line
    vertical_meet();
    hor_write();

    meet_rand_last();
    ver_write();
    line_feed();

    return 0;
}
```

미로의 너비와 높이를 입력받는다. 이후 vector.resize(n) method를 이용해 horizontal_wall과 vertical_wall, maze의 길이를 알맞게 조정한다. (maze는 원소를 모두 0으로 초기화)
이후 첫 번째 열을 파일에 출력하는 기능을 수행한다. line_feed는 가로부분의 테두리 (+-+-+-+-+ 모양)을 그리는 역할을 한다. first_rand()는 앞 쪽에 서술되어 있다.

for문을 통해 (높이-2)번만큼 vertical_meet(), hor_write(), meet_rand(), ver_write()함수를 호출한다. (높이-2)번인 이유는 맨 윗줄과 맨 아랫줄은 따로 처리해주기 때문이다.
vertical_meet()는 rand 함수를 통해 한 라인에서의 벽 유무를 결정해준다. 이 때, 전 줄과 현재 줄이 적어도 한 번은 만나야 하므로 다음과 같이 서술한다.

```
void vertical_meet() {
    int before_num;
    int ver_meet = 0;

    before_num = maze[0];
    for (int i = 0; i < WIDTH; i++) {
        horizontal_wall[i] = rand() % 2;
        if (horizontal_wall[i] == 0)
            ver_meet = 1;

        if (i < WIDTH - 1) {
            if (before_num != maze[i + 1]) { //적어도 한 번 만나야...
                if (ver_meet == 0) horizontal_wall[i] = 0;
                else ver_meet = 0;
                before_num = maze[i + 1];
            }
        }
        if ((i == WIDTH - 1) && (ver_meet == 0))
            horizontal_wall[i] = 0;

        if (horizontal_wall[i]) {
            maze[i] = Id;
            Id++;
        }
    }
}
```

meet_rand()는 rand()를 통해 두 칸이 만날지 여부를 결정하고 적용한다. 만나는 것으로 결정되면, 사이의 벽을 허물고 경로가 생긴 칸들의 Id를 통일해 준다.

```
void meet_rand() {
    int shall_we_meet = 0;
    int before_num;

    for (int i = 0; i < WIDTH - 1; i++) {
        if (maze[i] != maze[i + 1]) {
            shall_we_meet = rand() % 2; //만날지 여부 결정
            if (shall_we_meet) {
                before_num = maze[i + 1];
                maze[i + 1] = maze[i];
                for (int j = 0; j < WIDTH; j++)
                    if (maze[j] == before_num) maze[j] = maze[i];
                vertical_wall[i] = 0;
            } else vertical_wall[i] = 1;
        } else vertical_wall[i] = 1;
    }
}
```

ver_write()와 hor_write()는 앞에서 주어진 함수들이 결정한 미로의 데이터 (벽의 유무)에 따라 세로, 가로 벽을 파일에 입력하는 역할을 한다.

아래는 ver_write()이다. (ver_write()와 hor_write()는 유사하다.)

```
void ver_write() {
    fprintf(fp, "|");
    for (int i = 0; i < WIDTH - 1; i++) {
        fprintf(fp, " ");
        if (vertical_wall[i] == 1) fprintf(fp, "|");
        else fprintf(fp, " ");
    }
    fprintf(fp, " |\n");
}
```

마지막 라인을 meet_rand_last(), ver_write(), line_feed()로 마감해주면서 미로 작성이 끝난다.

작성된 미로는 “mazeRes.maz” 파일로 저장되었다.

실습 안내를 처음 접했을 때는 2차원 배열을 하나 만들어서 그곳에 모든 데이터를 집어넣는 방식, struct를 만들어서 처리하는 방식 등을 생각했으나 공간, 시간 복잡도가 비효율적으로 나올 것 같았다. 그리하여 예비보고서를 쓰기 전, 어느 정도 자료구조에 대한 구상을 마쳐 미로의 열 정보를 저장하는 배열, 현재 열과 다음 열 사이의 벽 정보를 저장하는 배열, 현재 열 안에서 서로 인접한 두 칸끼리의 벽에 대한 배열을 따로 만들어 구현하는 방식으로 작성하게 되었다.

따라서, 앞서 기술하였듯이, 시간 복잡도 O(WIDTH*HEIGHT), 공간 복잡도 O(WIDTH)의 알고리즘을 짜게 되었다.