11주차 결과보고서

전공: 철학과 학년: 3학년 학번: 20180032 이름: 남기동

1. 실습에 작성한 프로그램의 자료구조와 알고리즘, 완성한 알고리즘의 시간 및 공간 복잡도

우선 미로의 높이와 너비의 값을 받아서 2차원 배열로 선언한 maze에 필요한 크기만큼을 동적 할당해야 한다. 이때, 미로의 높이와 너비는 각각 2를 곱하고 1을 더한 값으로 설정하는데, 그 이유는 미로의 높이와 너비가 미로의 방만을 센 값인 반면 미로의 정보를 저장하는 maze에는 미로의 방뿐만 아니라 벽에 대한 정보도 포함할 것이기 때문이다. 또한 이후에 벽을 삭제할지 유지시킬지를 랜덤하게 정하기 위해 난수를 생성하기 위해 srand함수를 main함수 초반에 사용한다.

```
srand((unsigned int)time(NULL));

printf("Put width, height: ");
scanf("%d %d", &width, &height);
int WIDTH = width * 2 + 1;
int HEIGHT = height * 2 + 1;

maze = (int**)malloc(sizeof(int*) * HEIGHT);
for (i = 0; i < HEIGHT; i++) {
        maze[i] = (int*)malloc(sizeof(int) * WIDTH);
}

Init_maze(HEIGHT, WIDTH);
```

Init_maze함수는 정해진 미로의 높이와 너비를 이용하여 모서리, 가로벽, 세로벽, 빈방을 설정하여 미로를 초기화하는 작업이다. 이 때 maze에서 1은 '+' 모서리를 의미하고 2는 '-' 가로벽을 의미하며 3은 '|' 세로벽을 의미하고 방을 의미하는 곳은 0으로 간주한다. 따라서 for loop을 높이 만큼, 그리고 너비만큼 2번 돌면서 행이 짝수, 열도 짝수인 경우는 모서리로 간주하여 1을 대입하고, 행은 짝수, 열이 홀수인 경우에는 가로벽으로 간주하여 2를, 행이 홀수, 열이 짝수인 경우에는 세로벽이 들어갈 자리이므로 3을 대입하고 마지막으로 행이 홀수 열도 홀수인 경우에는 미로의 방이

므로 0을 대입한다.

```
void Init_maze(int HEIGHT, int WIDTH) {
	int i, j;
	for (i = 0; i < HEIGHT; i++) { //height * 2 + 1
		for (j = 0; j < WIDTH; j++) { //width * 2 + 1
			if ((i % 2) == 0 && (j % 2) == 0) maze[i][j] = 1
			if ((i % 2) == 0 && (j % 2) == 1) maze[i][j] = 2;
			if ((i % 2) == 1 && (j % 2) == 0) maze[i][j] = 3;
			if ((i % 2) == 1 && (j % 2) == 1) maze[i][j] = 0;
		}
	}
}
```

그 다음으로는 첫 번째 줄을 설정해준다. 우선 첫 번째 줄을 서로 다른 집합에 속하도록 초기화한 후, 첫 번째 줄의 가로벽들을 랜덤으로 제거 혹은 유지하는데, 만약 벽이 없어진다면 벽 기준 왼쪽과 같은 집합에 속하도록 해당 위치의 maze의 값을 변경한다.

```
//첫번째 줄, 서로 다른 집합에 속하게 초기화
for (i = 0; i < WIDTH; i++) {
	if ((i % 2) == 1) {
		maze[1][i] = number++;
	}
}
//첫번째 줄, 벽 랜덤으로 제거 or 유지
for (j = 1; j < WIDTH - 1; j++) { //왼쪽 벽이랑 오른쪽 벽은 제거 대상이 아니기 때문
	if ((j % 2) == 0) {
		select = rand() % 2; //0 또는 1이 저장 (0이면 벽 지우기, 1이면 벽 유지)
		if (select == 0) {
			maze[1][j] = 4; //세로벽 지우기(지워진 벽은 4로 간주)
			maze[1][j + 1] = maze[1][j - 1]; //벽이 없어지면 벽 기준 왼쪽과 같은
집합에 속하도록 설정
		}
	}
}
```

그 다음에는 마지막 줄 전까지 수직 경로를 만들고 행을 정리하는 작업을 반복해야 한다.

우선 수직 경로 만드는 것을 설명하면, 무조건 집합 당 1개의 수직 경로는 반드시 있어야 하며, 만약 이미 같은 집한 안에서 수직 경로가 있다면 그 다음부터는 수직 경로를 임의로 정할 수 있다. 단, 바로 이전 칸이 수직 경로가 뚫려 있는 경우에는 한 칸은 수직 경로를 만들 수 없는데, 그 이유는 만약 두 칸 연속으로 같은 집합에서 수직 경로를 만든다면 미로 내에서 순환하는 부분이 발생하여 완전 미로에 부합하지 않기 때문이다.

```
for (j = 1; j < WIDTH - 3; j += 2) {
	if (maze[i][j] == maze[i][j + 2] && possible == 0) {
		select = rand() % 2;
```

```
                if (select == 0) {
                        maze[i+1][j] = 4; //가로벽 제거
                        maze[i+2][j] = maze[i][j]; //아래칸을 위칸과 같은 집합으로
                                flag = 1; possible = 1;
                }

                if (j + 2 == WIDTH - 2) { //마지막칸 처리
                        if (flag == 1 && possible == 0) {
                                select = rand() % 2;
                                if (select == 0) {
                                        maze[i+1][j + 2] = 4;
                                        maze[i+2][j + 2] = maze[i][j + 2];
                                }
                        }
                        else if(flag == 0) { //집합 내 수직 경로가 아직 없음
                                maze[i+1][j + 2] = 4; //수직 경로가 없으니 무조건 수직 경로가
나야함
                                maze[i+2][j + 2] = maze[i][j + 2];
                        }
                        else if (flag == 1 && possible == 1) {
                                possible = 0;
                        }
                }
        }

        else if (maze[i][j] == maze[i][j + 2] && possible == 1) {
        //바로 이전꺼가 수직경로를 내린 경우
                possible = 0; //다음 시행에서는 그릴 가능성 있음, flag는 이미 1
        }

        else if (maze[i][j] != maze[i][j + 2] && flag == 0) {
        //다음부터 다른 집합인데 기존 집합에서 수직 경로가 없었다면,
                maze[i+1][j] = 4; //가로벽 제거
                maze[i+2][j] = maze[i][j]; //아래칸을 위칸과 같은 집합으로 설정

                flag = 0;
                possible = 0;

                if (j + 2 == WIDTH - 2) { //마지막칸 처리
                        maze[i+1][j + 2] = 4; //가로벽 제거(1개밖에 없는 집합이니깐)
                        maze[i+2][j + 2] = maze[i][j + 2];
                }
        }

        else if (maze[i][j] != maze[i][j + 2] && flag == 1) {
                if (possible == 0) {
                        select = rand() % 2;
                        if (select == 0) {
                                maze[i + 1][j] = 4;
                                maze[i + 2][j] = maze[i][j]; //아래칸을 위칸과 같게
                        }
                }
                flag = 0; //다음 시행에서 집합이 flag 0으로 시작할 수 있도록
                possible = 0;
```

```
                if (j + 2 == WIDTH - 2) { //마지막 칸 처리
                        maze[i+1][j + 2] = 4; //가로벽 제거(1개밖에 없는 집합이니깐)
                        maze[i+2][j + 2] = maze[i][j + 2];
                }
        }
}
flag = 0;
possible = 0;
```

위의 코드가 수직 경로를 만드는 것에 해당하는 부분이다. 수직 경로를 만드는 것은 해당 열을 조사하면서 크게 4가지의 케이스로 구분된다.

첫 번째는 해당 미로의 방과 그 오른쪽(다음) 방이 같은 집합에 속하며 수직 경로를 내릴 수 있는 경우이다. 이 경우에는 난수로 0 혹은 1을 설정하고 만약 0이면 가로벽을 해당 열의 한 칸 아래에 있는 가로벽을 제거하고 해당 미로 방 아래의 방도 같은 집합이 되게 설정한 후, flag와 possible을 1로 설정한다. Flag는 이미 같은 집합에서 수직 경로가 있을 때 1이며, possible은 이번 미로의 방이 수직의 경로를 내릴 수 없을 때 1을 가리킨다.

두 번째 경우는 해당 미로의 방과 그 다음 방이 같은 집합에 속하는데 possible이 1인, 즉 수직 경로를 내릴 수 없는 경우이다. 이런 경우는 첫 번째 경우처럼 이전 방이 이미 수직경로를 내려서 이번 방에서 수직 경로를 내린다면 순환 미로가 되기 때문에 고려하는 케이스이다. 이 때는 possible만 0으로 설정한 후 다음 방을 조사하게 넘어간다.

세 번째 경우는 해당 방이 다음 방과 다른 집합에 속하며 flag가 0인, 즉 해당 집합에서는 아직 수직 경로가 없는 경우이다. 이 경우에는 집합 안에서 적어도 하나의 방은 수직 경로를 내려야 한다는 원칙을 지키기 위해 무조건 수직 경로를 만들어야 한다. 따라서 가로벽을 제거하고 아래 방을 해당 방과 같은 집합이 되게 설정한다. 이때 flag와 possible을 0으로 초기화하는데 그 이유는 해당 방과 다음 방이 다르기 때문에 다음 방부터는 새로운 집합에 해당하기 때문이다.

마지막 네 번째 경우는 해당 방이 다음 방과 다른 집합에 속하면서 flag는 1인 경우이다. 이것은 해당 방에서 기존의 집합이 끝나는데 이미 해당 집합에 수직 경로가 존재하다는 것이다. 이 경우에는 임의로 수직 경로를 내리도록 설정하면 되지만 그 전에 possible이 0인지 확인해야 한다. 즉 이전 방에서 수직 경로를 내렸다면 연달아서 수직 경로를 내릴 수 없기 때문이다. Possible이 0인 경우에만 수직 경로를 만들지 말지 임의로 결정한다.

이렇게 네 가지 케이스를 모두 고려한 이후에는 다음 줄에 내려 갔을 때 flag와 possible이 초기화되어 있어야 하므로 for loop 바깥에 flag와 possible을 0으로 초기화해 주어야 한다.

그 다음은 해당 줄을 다 읽어서 수직 경로를 내릴지 말지 결정한 이후에 아래 줄에서 수직 경로가 만들어지지 않은 미로의 방들의 경우 새로운 집합으로 만드는 과정이 필요하다.

```
for (j = 1; j < WIDTH - 1; j += 2) {
        if (maze[i + 2][j] == 0) maze[i + 2][j] = number++;
}
```

그 다음에는 아래 줄의 미로 방들 중에서 서로 다른 집합에 속한 방이 접해 있는 경우, 세로벽을 허물지 말지 임의로 결정하고 만약 세로벽을 허문다면 두 방의 집합을 하나의 집합으로 통합시킨다. 필드에서 우측 방의 집합에 속했던 모든 방들을 좌측 방의 집합을 나타내는 값을 대입하여 같은 집합에 속한 것으로 나타낸다.

```
if (i + 2 == HEIGHT - 2) break; //마지막 줄의 경우에는 벽 제거의 원칙이 다르기 때문에 종료
//행에서 임의로 벽 제거
for (j = 1; j < WIDTH - 3; j += 2) {
if (maze[i + 2][j] != maze[i + 2][j + 2]) {
        select = rand() % 2;
                if (select == 0) {
                        maze[i + 2][j + 1] = 4; //세로벽 허물기(우측)
                        temp = maze[i + 2][j + 2];
                        maze[i + 2][j + 2] = maze[i + 2][j];
                        for (p = 1; p < HEIGHT - 1; p += 2) {
                                for (t = 1; t < WIDTH - 1; t += 2) {
                                        if (maze[p][t] == temp) maze[p][t] = maze[i + 2][j];
                                }
                        }
                }
        }
}
```

이와 같은 방식으로 모든 열을 조사하는데, 마지막 열의 경우에는 처리 방법이 다르기 때문에 마지막 열은 따로 처리해야 한다. 마지막 줄의 경우에는 인접해 있으며 서로 다른 집합에 속한 방들 사이의 모든 벽을 제거한다.

```
//마지막줄 처리: 인접해 있으며, 서로 다른 집합에 속한 방들 사이의 모든 벽을 제거
for (j = 1; j < WIDTH - 3; j += 2) {
        if (maze[HEIGHT - 2][j] != maze[HEIGHT - 2][j + 2]) {
                maze[HEIGHT - 2][j + 1] = 4; //가로벽 제거

                temp = maze[HEIGHT - 2][j + 2];
                maze[HEIGHT - 2][j + 2] = maze[HEIGHT - 2][j];
                for (p = 0; p < HEIGHT ; p++) {
```

```
                        for (t = 0; t < WIDTH; t++) {
                                if (maze[p][t] == temp) maze[p][t] = maze[HEIGHT - 2][j];
                        }
                } //왼쪽 집합으로 오른쪽 집합을 변경
        }
}
```

벽이 제거되어 서로 다른 집합이었다가 접하게 된 두 개의 미로의 방은 서로 같은 집합에 속하도록 설정해야 한다. 이는 미로의 모든 방을 탐색하여 우측 방의 집합을 나타내는 숫자를 갖고 있는 방의 경우에 모두 좌측 방의 집합을 나타내는 숫자로 변경시킨다.

이렇게 하여 maze에 랜덤하게 만들어진 미로의 정보는 모두 설정되고 저장되었다. 마지막으로는 미로를 그리고 미로를 maz 파일에 저장해야 하며 동적으로 할당했던 메모리를 해제해야 한다.

```
print_maze(HEIGHT, WIDTH);
printf("₩n");

fwrite_maze(HEIGHT, WIDTH);

for (i = 0; i < HEIGHT; i++) free(maze[i]);
free(maze);



void print_maze(int HEIGHT, int WIDTH) {
        int i, j, command;
        for (i = 0; i < HEIGHT; i++) {
                for (j = 0; j < WIDTH; j++) {
                        command = maze[i][j];
                        switch (command) {
                        case 1:
                                printf("+"); break;
                        case 2:
                                printf("-"); break;
                        case 3:
                                printf("|"); break;
                        case 4:
                                printf(" "); break;
                        default:
                                printf(" ");
                                //printf("%d", maze[i][j]); //for debugging
                        }
                }
                printf("₩n");
        }
}
```

이 함수는 미로를 화면에 출력하는 함수로 꼭 필요한 함수는 아니지만 maz 파일에 저장되기 이

전에 프로그램 실행의 결과로 미로가 제대로 만들어졌는지 파악하기 위한 용도로 유용하기에 추가했다. 미로의 높이와 너비를 두 개의 for loop으로 탐색하여 maze의 모든 term을 탐색하면서 1은 모서리, 2는 가로벽, 3은 세로벽, 4는 가로벽이나 세로벽 중 없어진 벽을 의미하는 값이기에 빈칸으로 출력하고 그 외의 값은 미로의 방을 나타내기 때문에 이 역시 빈칸으로 출력한다.

```
void fwrite_maze(int HEIGHT, int WIDTH) {
        FILE* fp;
        fp = fopen("maze.maz", "w");
        int i, j, command;
        for (i = 0; i < HEIGHT; i++) {
                for (j = 0; j < WIDTH; j++) {
                        command = maze[i][j];
                        switch (command) {
                        case 1:
                                fprintf(fp, "+"); break;
                        case 2:
                                fprintf(fp, "-"); break;
                        case 3:
                                fprintf(fp, "|"); break;
                        case 4:
                                fprintf(fp, " "); break;
                        default:
                                fprintf(fp, " ");
                        }
                }
                fprintf(fp, "₩n");
        }
        fclose(fp);
}
```

이 함수는 실습에서 요구하는 maz 파일에 미로를 출력 및 저장하기 위한 함수이다. 기존의 print_maze함수와 유사한 방식으로 구현되며 유일한 차이는 파일 포인터를 사용하여 "maze.maz"라는, maz를 확장자로 갖는 파일을 'w', 즉 쓰기 목적으로 열고 print 대신 fprintf로 해당 파일에 저장한 후 fclose로 파일을 닫고 마무리하는 것이다.

실습에서 작성한 프로그램의 시간 복잡도는 O(h^2 x w^2)이다. h는 미로의 높이에 해당하며 w는 미로의 너비에 해당하는 값이다. 서로 다른 집합 사이의 벽이 무너지고 서로 다른 두 집합에 속한 모든 방들을 하나의 집합에 속한 방으로 바꾸는 부분을 잘 처리하면 보다 낮은 시간 복잡도로 구현할 수 있을 것 같은데 다른 부분을 구현하는 것에 신경을 쓰느라 이 부분을 고민하여 업그레

이드하지 못하고 가장 단순하게 모든 미로의 값을 조사하는 방식을 사용했기 때문에 시간 복잡도가 예비 보고서에서 생각했던 것보다 커졌다. 공간 복잡도는 미로의 정보를 저장한 maze 이외에는 동적으로 할당한 것이 없기 때문에 O(h x w)가 된다.