실험 PRJ-2 1주차 미로 (Maze) 결과보고서

전공: 컴퓨터공학과 학년: 2 학번: 20211599 이름: 주현수

1.

실험시간에 작성한 프로그램에서 메이즈를 만드는 함수에서 수평부분과 수직부분에 대해서 벽을 뚫는 부분에서 eller’s algorithm이 사용된다.

우선 메이즈를 만들 때, 2차원 배열을 사용해서 만들며, ‘+’,’-‘,’|’이런 기호를 사용해서 만들 때, 통로부분에 각자 영역을 나타낼 번호를 부여한다. 이후 메이즈를 만드는 함수에서 수평부분을 먼저 보자면, 영역이 같지 않으면 랜덤함수를 사용해 50%의 확률로 벽을 뚫고 영역을 숫자가 더 작은 것으로 병합하는 과정을 만든다. 그리고 수직부분에 대해서는 flag라는 변수를 사용해서 윗줄의 한 영역 안에서 밑줄의 벽이 뚫렸는지 확인하는 작업을 해준다. 수평과 마찬가지로 50%의 확률로 벽을 뚫고 영역을 합쳐준 후, 영역의 끝까지 갔을 때, flag변수를 확인해서 그동안 벽이 뚫린 적이 한 번이라도 있는지 확인해준다. 만약 그러지 않았을 경우, 벽을 한 번 뚫어준다. 벽이 한 번도 뚫리지 않았을 경우, 그 영역에 대해서 아예 막힌 경로가 나오기 때문이다.

이 과정이 들어간 maze()함수의 시간복잡도를 생각해보면 O((n-1)(m-1))이 된다. 그리고 최대 (2\*n+1)\*(2\*m+1)만큼의 2차원 배열을 선언했기 때문에 이와 같은 공간복잡도를 갖는다.

예비보고서에서 작성한 backtracking을 사용하면 간 방향에 대한 변수를 만들고 변수의 상태를 확인해야한다. 이는 수직으로 병합할 경우와 비슷하다. 그러나 각 상황에서 이미 간 경우에 대해서는 다시 원래 자리로 돌아가야 하는 수고가 있기 때문에 실습시간에 활용한 알고리즘의 경우가 더 효율적이다.

void maze(void)

{

int flag;

for (int i = 0; i < n - 1; i++)

{

//수평으로 병합하는 과정

for (int j = 0; j < m - 1; j++)

{

//서로 영역이 같지 않으면

if (c[i][j] != c[i][j + 1])

{

//임의로 벽 삭제

if (rand() % 2 == 1)

{

mapp[i \* 2 + 1][(j + 1) \* 2] = ' ';

equalize(c[i][j], c[i][j + 1]);

}

}

}

//수직으로 병합하는 과정

flag = 0;

for (int j = 0; j < m; j++)

{

//임의로 벽 합치기

if (rand() % 2 == 1)

{

mapp[(i + 1) \* 2][j \* 2 + 1] = ' ';

c[i + 1][j] = c[i][j];

flag = 1;

}

//끝부분까지 갔을 때

if (c[i][j + 1] != c[i][j])

{

// 그 전에 합쳐진 경우가 없을 때

if (flag == 0)

{

mapp[(i + 1) \* 2][j \* 2 + 1] = ' ';

c[i + 1][j] = c[i][j];

}

//다시 초기화

flag = 0;

}

}

}

for (int j = 0; j < m - 1; j++)

{

if (c[n - 1][j] != c[n - 1][j + 1])

{

mapp[2 \* n - 1][(j + 1) \* 2] = ' ';

equalize(c[n - 1][j], c[n - 1][j + 1]);

}

}

}