11주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181294 이름: 임승섭

**1.**

1 - 1.

프로그램을 구현하기 위해 사용한 자료구조는 배열이다. 예비보고서와 유사하게 배열 3개를 만들었다. 현재 행의 정보를 담고 있는 배열 set[WIDTH], 현재 행의 벽 정보를 담고 있는 배열 row\_wall[WIDTH - 1], 다음 행의 정보를 담고 있는 배열 column\_wall[WIDTH]를 만들었다. 앞에 두 배열은 예비보고서와 동일하나, 마지막 column\_wall에 약간 차이가 생겼다. 실험 전에는 아래 벽의 유무만 판단하면 될 것이라고 생각하여 0 또는 1 값을 가지는 배열을 구상하였는데, 실험 진행 과정에서 다음 행의 정보(방 이름)를 담는 배열을 만들고, 같은 열에 있는 두 방의 이름이 같으면 벽이 없고, 다르면 벽이 있다고 판단하는 것이 더 효율적이라고 판단하였다. column\_wall이 다음 행의 정보를 담도록 하였고, 다음 행으로 넘어가는 loop마다 set의 정보를 column\_wall의 정보로 업데이트 해주었다.

이 세 가지 배열은 코드 초반에 malloc을 이용하여 메모리를 동적 할당 해주었다. 미로의 첫 번째 행의 방 이름은 1부터 WIDTH로 설정해주고, 위에 언급했듯이 loop을 지나면서 column\_wall의 정보로 바꿔주었다. 다음 행으로 넘어가는 loop이 시작될 때마다 column\_wall의 값은 -1, row\_wall의 값은 1로 설정하여 초기에는 모든 방이 존재한다고 가정하였다.

1 - 2.

먼저 세로 방 개수(HEIGHT, M)만큼 for loop을 시작하였다. loop 시작 직후 column\_wall과 row\_wall의 정보를 초기화하고, 가장 왼쪽의 벽(테두리)를 그렸다.

다음으로 row\_wall을 다루기 위해 가로 방 개수(WIDTH, N)만큼 for loop을 시작하였다. 두 방이 다를 때, 즉 방 사이에 벽이 존재할 때만 수정해주면 되므로, if(set[i] != set[i+1]) 조건문을 사용하였다. 방 사이의 벽이 없어지는 것은 랜덤이지만, 맨 마지막 줄에서는 방의 이름이 다르면 무조건 벽이 없어져야 한다. 이를 if(rand()%2==1 || j == M-1) 조건문을 이용하여 다루었다. 조건문 안에 들어가서는 먼저 row\_wall[i]의 값을 0으로 바꿔주고, set[i+1]과 같은 이름을 갖는 모든 방들을 탐색하여 set[i]로 바꿔주었다. 여기서 예상치 못한 time complexity가 더 생겼다. 이는 이후에 설명하겠다. row\_wall의 값을 업데이트 하였다면 그에 따라 벽을 출력하였다. 조건문을 이용하여 0이면 “ “, 1이면 “ |’을 출력하여 화면에서 볼 수 있도록 하였다.

row\_wall을 위한 loop이 끝났으면 가장 오른쪽 벽(테두리)를 그려주고 줄넘김을 해준다. 이제 column\_wall을 위한 작업을 해준다. 먼저 정수형 타입의 flag 변수를 0으로 설정한다. flag는 현재 수정하고 있는 방 기준에서 왼쪽 방 아래 벽을 뚫었는지 판단하기 위해 사용한다. 변수 선언 후 하단 벽을 출력하는 공간으로 가장 왼쪽 벽(테두리)를 그려준다.

이제 본격적으로 column\_wall의 정보를 수정한다. 편의를 위해 먼저 첫 번째 열(방)일 때는 loop을 돌지 않고 수정한다. 만약 첫 번째 방이 두 번째 방과 다르다면(여기서 다르다는 건 방 이름이 다르다는 의미로 사용하겠다) 첫 번째 방은 무조건 수직 경로를 생성해야 한다. 따라서 column\_wall[0]를 set[0]로 업데이트하고 수직 경로를 생성했다는 의미로 flag를 1로 바꿔준다. 만약 첫 번째 방이 두 번째 방과 같다면 랜덤으로 뚫을지 결정한다. 뚫게 되면 위와 마찬가지로 수정하고, 뚫지 않는다면 column\_wall은 room\_number에 1 추가해준 값이 들어간다. 여기서 room\_number란 새로 생긴 방의 이름을 뜻하는 변수이다. 초기에 N + 1로 설정해주고 새로 방이 생길 때마다 1씩 추가한 값으로 바뀐다. 수직 경로를 생성하지 않았으므로 flag는 0으로 해준다. column\_wall의 정보를 수정하였으므로 이 정보에 따라 실제 화면에 벽을 출력한다. 맨 마지막 줄일 때는 수직 경로가 필요 없으므로 if (j < M-1)로 조건문을 걸어주고, flag가 0이면 “-+”, 1이면 “ +”를 출력한다.

두 번째 열(방)부터 마지막 열까지는 for loop을 이용한다. 첫 번째 열을 따로 구분한 이유는 loop 내에서는 현재 방과 왼쪽 방을 비교하기 때문이다. 즉 set[i]와 set[i-1]을 비교한다. 먼저 두 방이 같을 때와 다를 때로 나눈다. 이유는 두 방이 같게 되면 연속으로 수직 경로를 생성할 수 없기 때문이다. 하지만 두 방이 다르면 각 방의 수직 경로를 신경 쓸 필요가 없어진다. 따라서 두 방이 같을 때, 또 다시 flag가 1일 때와 0일 때로 나눈다. flag가 1일 때는 현재 방의 수직 경로를 생성할 수 없으므로 위와 같이 column\_wall을 ++room\_number로 해주고, flag는 0으로 바꿔준다. flag가 0일 때는 랜덤으로 뚫을지 말지 결정한다. 수직 경로를 생성하는 코드를 굳이 다시 설명하진 않겠다. flag가 0일 때의 코드가 그대로 두 방(set[i], set[i-1])이 다를 때의 코드와 같아진다. 위에 언급했듯이 두 방이 다를 때는 각 방의 수직 경로 유무가 필요 없기 때문에 flag를 신경 쓸 필요 없이 랜덤으로 뚫을지 말지 결정하면 되기 때문이다.

이 과정에서 문제가 하나 발생한다. 연속으로 뚫릴 것은 조건문을 이용해서 막을 수 있었지만, 하나도 뚫리지 않은 경우를 고려하지 않았기 때문에 수직 경로가 하나도 없는 방이 생길 수 있다. 이를 위해 우선 if(set[i] != set[i+1])로 현재 방이 같은 방들의 가장 오른쪽인지 확인하고, 그 같은 방들의 수직 경로가 혹시 하나도 없는지 확인하기 위한 변수로 flag2를 선언한다. 0번 인덱스부터 i-1번 인덱스까지 set[i]와 같은 방인데 수직 경로가 있으면 (column\_wall[k] == set[k]) flag2를 1로 바꿔준다. loop이 끝났는데도 flag2가 0이라면 그 같은 방들에 수직 경로가 여태까지 하나도 없었다는 뜻이므로 맨 마지막(오른쪽) 방에 수직 경로를 생성한다. 마지막으로 column\_wall의 정보를 업데이트하였으므로 이를 토대로 각 방의 아래 벽을 출력한다. 이 역시 i=0일 때 설명하였으므로 생략하겠다.

두 번째 방부터 마지막 방까지의 loop이 끝났다면 우선 “\n”을 출력하여 줄넘김을 해준다. 이 역시 마지막 행에서는 필요 없으므로 if (j < M-1) 조건문을 걸어준다. 각 방의 아래 방들의 정보가 현재 column\_wall에 담겨 있으므로 이들을 set에 업데이트 시켜준다. 맨 마지막에 업데이트 시켜주어야 다음 loop을 돌 때 확실하게 이를 이용할 수 있다.

마지막 행까지의 loop이 드디어 끝났으면 맨 아래 테두리를 그려준다.

1 - 3.

현재 코드 내에서

for (j = 0; j < M; j++) : 다음 행으로 넘어가는 loop

for (i = 0; i < N; i++) : 같은 행에서 다음 방으로 넘어가는 loop

for (k = 0; k < N; k++) : 같은 이름을 가진 방의 이름을 모두 바꿔주기 위해 그 방들을 찾는 loop

이 연속으로 있는 가장 큰 시간복잡도를 가지는 loop이라고 할 수 있다. 따라서 시간복잡도는 O(M\*N\*N)이라고 할 수 있다.

메모리는 맨 처음 생각했던 배열 3개 이외에는 따로 추가된 것이 없기 때문에 O(N)이라고 할 수 있다.

1 - 4.

실험 전에 생각했을 때와 다른 부분은 먼저 column\_wall의 정보이다. 한 번 언급했듯이 실험을 계획할 때에는 수직 경로의 유무만 판단하기 위해 0 또는 1 값만 저장하려고 하였지만, 다음 행의 정보를 저장하고 있는 것이 더 효율적이라고 판단되어 계획과는 약간 다르게 값을 저장하였다.

사실 전체적으로 어려움이 없을 것이라고 생각한 실험이었는데 가장 시간을 많이 사용한 부분은 column\_wall을 바꿔주는 과정이었다. 현재 방과 이전 방을 비교해야 했기 때문에 첫 번째 방은 loop에서 제외시키고 따로 빼서 고려하였고, 같은 방에서 수직 경로가 하나도 생기지 않은 경우를 어떻게 다뤄야 할 지도 고민이 많았다. 결과적으로 조건문과 반복문을 사용하여 잘 처리했다고 판단된다.