11주차 예비보고서

전공: 수학과,컴퓨터공학과 학년: 3학년 학번: 20181256 이름: 김도현

1.

미로 생성에 또 다른 알고리즘은 조사 결과 Recursive Backtracking 이라는 깊이우선탐색(DFS) 을 이용하는 알고리즘이 있다. 우선 어느 곳에서 노드를 시작하는지는 결정하지 않고 랜덤하게 첫 current 노드를 정한다. 그리고 이 current 노드를 기준으로 되돌아 가지 않고 움직일 수 있는 곳을 rand를 이용하여 랜덤한 곳을 찾아 next 노드로 간다. 랜덤하게 선택한 곳으로 간다는 의미는 current 노드와 next 노드 사이의 벽이 허물어져야 된다는 것이다. 이런 과정을 반복적으로 시행하여 더 이상 움직일 곳이 없을 때 까지 진행한다. 만약 current 노드에서 더이상 움직일 곳이 없다면 이제는 previous 노드로 돌아가서 방문하지 않은 노드 중 다시 움직일 수 있는 노드가 존재하는지 찾고 가능하다면 그 노드로 이동, 불가능하다면 다시 previous 노드로 이동하여 모든 노드를 다 탐색하면 이 알고리즘은 끝난다.

이 알고리즘은 current 노드에서 next 노드로 이동하고 다시 거기서 움직일 수 있는 노드를 찾기 때문에 계속해서 깊이 내려가는 특성이 있기 때문에 깊이 우선이라는 말에 부응하고 리스트로 표현 시 선형적인 알고리즘으로 볼 수 있고 이는 재귀함수를 호출하여 계속해서 next 노드를 찾는게 가능하다. 여기서 주의 해야 할 점은 모든 노드 들의 정보를 가지고 있어 노드들을 모두 탐색했는지 안 했는지 판단을 하는 코드가 필수적으로 들어있어야 한다. 만약 없다면 무한 loop로 돌아갈 수 있다.

이때 일반적인 깊이우선탐색 알고리즘은 재귀 호출 있는데 찾아간 노드를 true로 하고 다음 이동가능한 노드 중 false인 것들 중 random 하게 찾는다. 혹은 LIFO(last in first out) 인 스택 형태를 구현하여 current 노드를 순차적으로 스택에 저장하고 더 이상 방문 할 곳이 없다면 스택에서 노드를 하나씩 꺼내 다시 탐색 할 수 있을 것이다. 인접리스트로 dfs를 만들면 current 노드의 연결된 간선의 수 만큼 탐색한다. 즉 전체적인 시간 복잡도는 일반적으로 O(모든 노드 수+모든 간선 수)로 생각하면 된다. 만약 인접행렬이라면 반복문을 모든 노드 수 만큼 탐색하고 current node 방문 시 이에 따라 다시 함수가 호출된다. 모든 노드를 다 탐색하기 때문에 시간 복잡도는 O(모든 노드 수)\*모든 노드 수=O(모든 노드 수^2) 이다. 전체적인 공간 복잡도는 배열로 표현 시 [노드 수][간선 수], 일반적으로 노드와 그에 따른 간선을 저장 할 O(노드의 수\*최대간선의 수) 이다.

2.

Ppt에서 설명한 Eller’s 알고리즘을 이용 할 것이다. 이 알고리즘은 1행 1열을 시작으로 1행부터 순차적으로 열들을 탐색하며 랜덤하게 같은 집합에 속할지 속하지 않을지 결정한다. 같은 집합에 속한다면 길이 연결되어 있다는 뜻으로 즉 벽을 뚫을지 뚫지 않을 지와 동일한 의미이다. 주의해야 할 점은 같은 집합에 속해 있는 칸에서는 적어도 한개의 수직 경로가 있어야 하고 옆 칸을 뚫는 경우에도 무조건 랜덤하게 벽을 뚫는게 아닌 집합 내 이동 할 수 있는 경로가 하나만 나오도록 벽을 뚫어야 한다. (이는 side 벽 사이에 같은 집합이 있으면 벽을 뚫으면 안된다는 것이다.) 이때 길이 뚫린다면 서로 다른 집합이 이제 같아져야 하므로 두개의 집합을 비교하여 더 작은 숫자의 집합으로 바뀌도록 설정해주어야 한다.(큰 것으로 바꿔도 상관 없지만 작게 바꾸도록 실험할 것이다.) 이 과정을 마지막까지 반복하는 것이다. 결국 모두 같은 집합에 속해 있는 임의의 칸에서 어느 칸을 가도 경로가 1개 뿐인 미로가 생긴다.

우선 이번 실험에서는 고정된 가로 세로가 아니라 scanf 등을 통해 미로의 가로, 세로 길이가 임의로 정해지고 이에 따른 배열 메모리를 동적 할당하여 받아야한다. 그래서 일단 한 행이 어떤 집합 정보인지 담고 있는 maze[WIDTH] 배열이 필요하다. 그리고 열 사이에 있는 벽들의 정보를 저장할 side[WIDTH-1] 배열 도 필요할 것이다. C언어는 bool이 없기 때문에 비록 int 형이지만 우리는 Bool 형처럼 벽이 있으면 1 없으면 0으로 표현하면 된다. 여기서 한 칸에 따른 아래의 벽을 저장하는 under[WIDTH] 배열도 있어야 하는데 이 역시 side 배열처럼 1,0 으로 표현할 것이다. 코드를 아직 짜보지는 않았지만 이렇게 0,1 로만 표현했을 시 어느 집합 번호에 속해 있는지를 몰라 다음 maze 배열에 값을 부여하기 힘들 수 도 있다. 그래서 under의 경우는 벽이 비었다면 해당 열의 집합 번호를 부여하는 방법도 생각해볼 수 있다. 이렇게 순차적으로 한 칸 씩 같은 집합으로 만들지 안 만들지 결정해야 한다. 즉 한 칸 씩 모든 칸을 탐색해야 하기 때문에 시간 복잡도는 O(WIDTH\*HEIGHT) 이고 공간 복잡도는 각 각 한 행 씩만 저장하는 배열 O(WIDTH), 열 사이의 벽 정보 저장 배열 O(WIDTH-1), 아래의 벽을 저장하는 배열 O(WIDTH), 등 이 필요하기 때문에 O(WIDTH) 이다. 그 이후 행은 under에 저장된 정보를 통해 다시 maze 배열에 집합의 번호를 부여할 수 있을 것으로 예상한다. 이들을 출력할 때는 반목문이 돌며 각 행 씩 maze[WIDTH], under[WIDTH] 이 계산될 때마다 print 하면 될 것이다. 이 때 가장 윗 줄과 아랫 줄은 그대로 ‘+-‘ 이 전부 출력되고 마지막에 ‘+’ 도 출력되어야 한다.