2020년도 2학기 컴퓨터공학설계및실험I

12주차 예비 보고서

20170175 김태안

1. 실습 목적

미로는 복잡한 길을 찾아 출발점부터 시작해 도착점까지 도달하는 퍼즐로 직사각형으로 배열된 방과 이러한 방들 사이를 막고 있는 벽으로 구성된다. 벽은 각 방의 상하좌우 사면에 존재할 수 있는데 인접한 두 방 사이에 벽이 없을 경우 두 방 사이를 자유로이 지날 수 있다. 본 프로젝트에서는 다음에 설명하는 완전미로를 만들 수 있는 여러 가지 방법을 알아보고 이를 직접 구현하기 위한 자료구조 설계 및 프로그래밍을 통해 미로를 생성해보도록 한다.

1. 관련 이론

**Eller’s Algorithm**

해답이 있는 미로를 생성하기 위해서 특정 알고리즘을 사용해 미로를 생성한다. Eller’s Algorithm은 미로 생성 알고리즘의 하나로, 다른 알고리즘에 비해 빠르지는 않지만 공간을 효율적으로 활용한다. 미로를 생성할 때 모든 미로의 정보가 필요한 것이 아니라, 한 줄 씩 필요하기 때문이다. 다음은 Eller’s Algorithm의 절차는 다음과 같다.[1]

1. 첫번째 줄을 그린다. 각 칸은 어떤 집합에도 포함되지 않는다.
2. 집합에 포함되지 않은 칸에 집합을 할당한다.
3. 오른쪽 경계선을 그린다. 인접한 두 칸이 같은 집합에 포함된다면, 오른쪽 경계선을 그린다. 그렇지 않다면, 랜덤하게 오른쪽 경계선을 그리거나 그리지 않는데, 그리지 않는다면 두 칸을 같은 집합에 포함시킨다.
4. 아래쪽 경계선을 그린다. 랜덤하게 경계선을 그릴지 말지 결정한다. 이때, 각 집합은 최소한 한 개의 아래쪽 경계선이 없는 칸이 있어야한다. 집합에 칸이 하나밖에 없다면 경계선을 그리지 않는다.
5. 다음 줄로 이동한다.
6. 모든 오른쪽 경계선을 제거한다.
7. 위쪽 경계선이 있는 칸은 집합에서 제거한다.
8. 아래쪽 경계선을 제거한다.
9. 2-8의 단계를 마지막 줄까지 반복한다.
10. 아래쪽 경계선을 추가한다.
11. 이웃한 칸이 다른 집합에 포함되어 있는지 확인하고, 다르면 오른쪽 경계선을 없앤 뒤, 집합을 병합한다.

**Recursive Backtracking Algorithm**

Recursive Backtracking Algorithm은 재귀적 방법을 사용해 미로를 생성하는 알고리즘이다.[1]

1. 먼저, 모든 칸의 사방에 경계선을 그린다.
2. 임의의 한 칸을 골라 이동하고, 인접한 칸을 탐색한다.
3. 탐색한 칸의 경계선이 모두 존재한다면 칸을 뚫고, 해당 칸으로 이동한다.
4. 인접한 칸에 이동할 곳이 없다면, 이전 칸으로 돌아간다.
5. 3-4를 반복해 모든 칸을 탐색한다.
6. 실습 방법

**Eller’s Algorithm 구현**

Eller’s Algorithm을 구현하기 위해, 경계선의 유무와 어떤 집합에 포함되는 지를 알아야한다. 이때, 경계선은 오른쪽과 아래쪽 경계선의 유무만 저장하여도, 인접한 칸의 경계선 유무를 통해 현재 칸의 나머지 경계선을 알 수 있다. 따라서 각 칸은 다음과 같은 구조체로 구현한다.

|  |
| --- |
| typedef struct cell {  int setNum;  bool right;  bool bottom;  }; |

모든 칸의 집합으로 이루어진 미로 필드는 탐색이 용이해야 한다. 모든 칸을 탐색하며 미로를 생성할 뿐만 아니라, 인접한 칸의 정보가 현재 칸의 경계선을 그릴 때 필요하기 때문이다. 따라서 위에서 정의한 cell로 구성된 2차원 배열을 사용하는 것이 적절하다.

미로의 줄 당 칸 수를 WIDTH, 미로의 줄 수를 HEIGHT라고 하자. Eller’s Algorithm은 각 줄마다 6, 7, 8, 2, 3, 4의 단계를 실행하는 데, 이때 각 칸을 모두 탐색한다. 즉, 한 줄을 완성하는 데 O(6\*WIDTH)의 시간이 필요하다. 이를 HEIGHT만큼 반복하므로, Eller’s Algorithm의 시간 복잡도는 O(HEIGHT\*WIDTH)이다. 미로의 정보를 저장하기 위한 공간은 미로의 필드로 충분하므로 공간 복잡도는 O(HEIGHT\*WIDTH)이다.

**Recursive Backtracking Algorithm 구현**

Recursive Backtracking Algorithm을 구현하기 위해 각 칸의 경계선 여부를 알아야한다. 그리고, Backtracking을 진행하며 해당 칸을 방문했는 지 확인해야한다. 따라서 다음과 같은 구조체를 사용한다.

|  |
| --- |
| typedef struct cell {  bool top;  bool right;  bool left;  bool bottom;  bool flag;  }; |

Eller’s Algorithm과 마찬가지로 모든 칸의 집합으로 이루어진 미로 필드는 탐색이 용이해야 한다. 따라서 cell로 구성된 2차원 배열을 사용해 미로 필드를 정의한다.

Recursive Backtracking Algorithm은 랜덤한 칸에서 시작해 이동할 수 있는 인접한 칸으로 이동하며 미로의 모든 칸을 방문한다. 따라서, 시간 복잡도는 O(HEIGHT\*WIDTH)라고 할 수 있다. 미로의 정보를 저장하기 위한 공간은 미로의 필드 이외의 공간이 필요하지 않다. 따라서 공간 복잡도는 O(HEIGHT\*WIDTH)이다.

1. 기타

**완전 미로와 불완전 미로**

완전 미로 (perfect maze, ideal maze)는 미로의 서로 다른 두 칸을 선택했을 때, 두 지점을 잇는 경로가 오직 하나뿐인 미로를 의미한다. 그에 반해 불완전 미로(imperfect maze, non-ideal maze)는 모든 칸에 접근은 가능하나, 임의로 선택한 두 점을 잇는 경로가 2개 이상일 수도 있는 미로로, 순환하는 구간이 존재하기도 한다. 두 미로를 구현하기 위해서는 다른 알고리즘을 사용한다. 완전 미로를 구현하는 알고리즘으로는 위에서 설명한 Eller’s Algorithm, Recursive Backtracking Algorithm 등이 있고, 불완전 미로를 구현하는 알고리즘으로는 Rooms Algorithm, Vertical Blocks Algorithm 등이 존재한다. 다음은 Rooms Algorithm을 사용해 불완전 미로를 생성하는 방법이다.[1]

1. 모든 칸의 경계선을 그린다.
2. 한 줄 씩, 두번째 줄의 맨 왼쪽 칸부터 시작한다.
3. 현재 칸과 다음, 그리고 그 다음 칸을 비운다.
4. 다음 칸의 아래 칸을 비운다.
5. 현재 칸에서 4칸 앞으로 이동한다.
6. 맨 오른쪽에 도달할 때까지 3-5를 반복한다.
7. 두줄 아래의 맨 왼쪽으로 이동한다.
8. 맨 아래에 도달할 때까지 6-7을 반복한다.
9. 미로 가운데의 세로 통로를 자른다.
10. 모든 칸을 탐색하며 두 이웃한 칸이 비어 있는 칸을 자른다.
11. 참고 문헌

[1] Dagaev, A. V., Sorokin, A. A., Kovalenko, R. A., & Yakovleva, E. A. (2020). Mazes creation for further study of swarm intelligence. *IOP Conference Series: Materials Science and Engineering,* *919*, 052058. doi:10.1088/1757-899x/919/5/052058