실험 PRJ-2 미로(Maze) 1주차 결과보고서

전공: 컴퓨터공학 학년: 2학년 학번: 20201635 이름: 전찬

**1. 목적**

실험 시간에 작성한 프로그램의 알고리즘과 자료구조를 요약해서 설명한다. 또한 완성된 알고리즘의 시간 및 공간 복잡도를 보인다.

**2. 구현한 알고리즘과 자료구조**

이번 실습에서는 Eller’s maze algorithm을 직접 구현하는 형식으로 실습을 진행했다. 총 두 가지의 함수(makemaze, savemaze)를 통해서 미로 제작 알고리즘을 구현했다.

우선 makemaze 함수를 실행하기 전에 main에서 n\*m의 각 원소가 strcuture로 구성된 matrix를 정의하고, 초기화하며 시작했다. 각 structure는 현재 x, y좌표와 함께, 오른쪽 벽면과 아래 벽면, 현재 그 위치의 블록이 속한 집합에 대한 정보를 갖고 있도록 설정했다.(사실 x, y좌표의 존재가 미로를 제작할 때는 필요하지 않아서, structure를 약간 바꾼다면 개선할 수 있을 것 같다.) 초기화할 때 오른쪽 벽면과 아래 벽면 = 1 이라고 설정해 모든 면이 막힌 미로를 구현했고, 모든 집합의 값을 0으로 초기화해주었다.

이후에는 makemaze(Mazeblock\*\* array, int width, int height); 함수를 제작했다. 이 함수는 array에 Eller의 알고리즘을 기반으로 완벽한 미로를 제작해서 저장해주는 함수이다. 이 함수에서는 우선 maze의 첫 번째 줄(array[0])을 집합 1~width로 나누며 초기화해주었다. 초기화줄 때에는 set\_value라는 전역변수를 통해서 집합을 표시해주었다. 이후에는 ppt에 나와있는 2~4의 과정을 첫 번째 줄~마지막-1 줄까지 반복하는 형태로 제작했는데, 각 과정은 다음과 같다.

2. 이웃한 block이 서로 다른 집합에 속하는 경우 랜덤으로 합쳐주며 벽을 삭제해준다.

3. 랜덤으로 현재 줄에서 아래로 한 칸 내리며 벽을 삭제해주며, 각 집합에서 최소한 1번은 아래로 내린다.

4. 다음 줄에서 value = 0(내려가지 않은 부분)을 새로운 집합으로 만들어준다.

이때 2번 과정에서 중요한 것은, 1 6 6 6 형태로 집합이 존재할 때, 1 6 을 합치는 경우, 나머지 6도 1로 만들어주어야 하는 것이었다. 따라서 합쳐질 때마다, 한 줄을 전부 읽어서 오른쪽에 존재하는 집합과 동일한 집합에 대해서 왼쪽의 집합을 대입해주는 형태로 진행했다. (python의 dictionary 형태를 사용한다면 훨씬 효율적으로 제작할 수 있었을 것 같다.)

3번 과정은 각 집합에서 최소 1개는 내리는 형태가 중요한 부분이다. 이는 현재 줄에서 아래 벽을 부수지 않은 어떤 block이 존재할 때,그 block과 같은 집합 중 아래 벽을 부순 block이 있는지를 판단하며 진행해주었다. 만약 그 집합의 모든 block이 아래로 벽을 부수지 않았을 경우, 현재 줄의 downwall을 다시 부수거나/부수지 않거나 초기화해주는 형태이다.(사실 그 집합의 마지막 원소에 대해서 무조건 아래로 내리는 형태를 고려했었다. 하지만 이렇게 되는 경우에, ‘랜덤’이라는 요소에 영향을 미칠 수 있을 것 같아 위와 같이 구현했다.)

4번 과정은 비교적 쉽게 구현할 수 있다. 현재 줄의 아래 줄에 대해서 value = 0인 block을 다른 set\_value 변수를 통해서 초기화해주는 형태로 코드를 작성했다.

makemaze의 마지막은 마지막 줄을 연결시키는 작업이었다. 이는 2번 작업과 유사하지만, 이웃한 block의 집합이 다른 경우에 무조건 벽을 부수고, 같은 집합으로 포함시키는 형태로 코드를 작성했다.

이후 savemaze(Mazeblock\*\* array, int width, int height) 함수는 maze.maz 파일에 형식에 맞게 loop를 통해서 maze를 저장하는 방식으로 제작해주었다.

**3. 구현한 알고리즘과 자료구조의 시간 및 공간 복잡도.**

이번 실습에서 작성한 코드는 크게 3가지 부분으로 나눌 수 있다. 1. 미로의 동적 할당 및 초기화, 2. makemaze 함수의 작동, 3. savemaze 함수의 작동 이때 미로의 높이 = h, 미로의 가로 길이 = w라 하자.

1. 미로의 동적 할당 및 초기화

시간 복잡도: O(wh), 각 block 마다 초기화해주는 형태이다.

공간 복잡도: O(wh), 각 block 마다 일정 공간을 차지하는 형태이다.

2. makemaze 함수

시간 복잡도: O(wh2), 2, 3, 4 과정에서 각 줄에서 한 block마다 한 줄을 읽는 형태로 수행되서 앞과 같다. 처음에 예비보고서에는 O(wh) 형태가 나올 것이라 생각했었지만, dictionary 형태를 사용하지 못한 점, ‘랜덤’을 더 살리기 위해서 코드를 작성해서 앞과 같은 시간 복잡도가 나왔다.

공간 복잡도: O(1), 추가적으로 공간을 할당하는 것은 없으며, main의 array를 변형한다.

3. savemaze 함수

시간 복잡도: O(wh), 각 block에 대해서 적절한 형태로 파일에 저장해준다.

공간 복잡도: O(1), 별다른 공간을 사용하지는 않는다.