11주차 결과보고서

전공: 경제학과 학년: 4학년 학번: 20180501 이름: 김연수

1.

Eller’s algorithm을 이용해 완전미로를 설계했다. main함수는 아래와 같다. 사용자에게서 미로의 넓이와 높이를 입력 받고 전역변수 WIDTH와 WEIGHT를 업데이트한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

generate\_wall()은 벽이 전부 막혀있는 미로를 생성해준다. generate\_wall을 실행하고 나면 다음과 같이 생긴 미로가 생성된다.

블랙이(가) 표시된 사진

자동 생성된 설명

generate\_newline과 generate\_space를 번갈아 호출해서 구현했다. 자세한 코드 내용은 아래 첨부를 참고하라.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

initialize\_num\_set함수는 미로 내 집합을 초기화하는 기능을 한다. 미로는 각 칸마다 집합을 가지는 데 처음 상태에서는 집합의 모든 수가 서로 달라야 한다. 1,2,3,4,5 순으로 각 칸마다 집합을 배정해준다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

eliminate\_wall 함수는 전체가 다 벽으로 막혀있는 미로를 이용해서 완전미로를 만들어 주는 함수다.

HEIGHT-2만큼 loop를 돌아 아래로 천천히 내려가면서 벽을 제거한다. 마지막 줄은 eliminate\_lastline함수를 호출해서 따로 처리해야 하므로 HEIGHT-2로 설정했다. loop안에서 i%2 == 1과 그렇지 않은 부분으로 나눈다. 이유는 가로벽이 존재하는 행과 칸이 존재하는 행을 구분하기 위함이다. 홀수 행과 짝수 행에 대해 각각 다른 알고리즘 이용해 처리한다.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

홀수 행에 대해서는 eliminate\_virtical함수를, 짝수 행에 대해서는 eliminate\_minus함수를 호출한다. 자세한 코드는 아래와 같다.

텍스트, 스크린샷, 폰트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

eliminate\_virtical은 세로막대를 제거하는 역할을 한다. 각각의 세로막대는 짝수 인덱스에 위치하므로 j%2 == 0인지 확인한다. 그리고 세로막대를 기준으로 좌측의 집합과 우측의 집합이 다른 경우 임의로 세로막대를 제거하거나, 제거하지 않는다. 여기서 k를 rand함수를 이용해 0과 1중 임의로 설정한 후, k가 1인 경우 막대를 제거하도록 설계했다.

두 집합의 숫자 중 더 작은 숫자로 update하도록 한다. update 작업은 update\_num\_set함수를 이용해 처리한다. 이 함수의 자세한 코드 설명은 아래에서 후술하도록 하겠다.

텍스트, 스크린샷, 디스플레이, 폰트이(가) 표시된 사진

자동 생성된 설명

eliminate\_minus는 위 보다 조금 더 복잡하다. 임의로 수직개방을 하되 최소한 하나의 집합에서 수직 개방을 해줘야 한다. 첫 번째 for loop의 설명은 다음과 같다. 현재 위치한 행에서 오른쪽으로 훑으며, 임의로 수직개방을 한다. 오픈할 지 말지를 결정하는 것은 open을 통해서 결정한다. open == 0인 경우 wall\_set[i][j] = ‘ ‘을 해준다. 그러면 ‘-‘였던 미로의 벽이 ‘ ‘로 바뀌면서 수직 개방이 이뤄진다. k는 위치해있던 벽 위에 있는 집합을 의미한다. check배열을 둬서 집합에 수직개방이 최소한 한 번 이루어졌음을 표시한다. 이후 미로의 집합을 update해준다.

두번 째 forloop은 check표시가 되어있지 않은 것들에 대해서 처음 마주치는 원소의 아랫 벽을 수직 개방 시켜주는 작업을 한다. 두 번 째 forloop을 마치고 나면 각 집합 당 최소한 하나는 수직 개방이 이루어진 상태가 된다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

eliminate\_lastline은 마지막 행을 처리해주는 함수다. 마지막 행에서는 vertical 막대를 기준으로 좌측과 우측의 집합이 같은 지를 검사한다. 만약 다른 경우 벽을 제거 해줘야 한다.

eliminate\_wall에서 벽을 제거할 때마다 update\_num\_set을 호출해줘야 한다. 벽을 제거한 후, 인접한 방과 집합이 같아지면, 열려 있는 방들 모두 같은 집합을 가져야 하기 때문이다. 따라서 방 하나에서 집합이 바뀌면 연결되어 있는 방들까지 전부 연쇄작용을 불러 일으킨다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

이것은 내가 생각했던 봐와 달랐다. 예비보고서에서는 시간 복잡도를 O(WIDTH \* HEIGHT)가 될 것으로 예상했다. 하지만 이는 집합을 update하는 것의 시간 복잡도를 간과했기 때문이었다. 예상했던 집합 update의 시간 복잡도는 어림잡아 O(1)이었지만, 실제로 구현해보면서 집합을 update하는 부분만 고려했을 때(즉, update\_num\_set의 시간복잡도는) O(WIDTH \* HEIGHT)가 도출되는 것을 확인했다. 그렇다면 update\_num\_set이 몇 번이나 호출되는 지를 확인해야 전체 프로그램의 시간복잡도를 알아낼 수 있다.

전체 프로그램의 시공간 복잡도를 구하기 위해서, main에서 호출되는 함수마다 시공간 복잡도는 계산해보자. generate\_wall()에서 시간 복잡도는 O(HEIGHT \* WIDTH)이다. initialize\_num\_set()에서 시간 복잡도도 마찬가지로 O(HEIGHT \* WIDTH)이다. 다음으로 가장 복잡한 eliminate\_wall()를 살펴보자. HEIGHT만큼 loop를 돌면서 eliminate\_virtical, eliminate\_minus함수를 호출하고 있다. 그리고 각 함수는 WIDTH만큼 loop를 돌며 update\_num\_set함수를 호출하고 있다. 따라서 eliminate\_wall()함수는 최대 O(HEIGHT^2 \* WIDTH^2)의 시간 복잡도를 가진다. print\_maze()또한 O(HEIGHT \* WIDTH)의 시간복잡도를 가진다.

main의 모든 함수들을 종합적으로 고려해본 결과, 전체 프로그램의 시간 복잡도는 O(HEIFHT^2 \* WIDTH^2)임을 알았다. 이는 예비보고서에서 예상했던 결과와 다르다. 예비보고서에서는 O(HEIGHT \* WIDTH)의 시간 복잡도를 가질 것으로 예상했지만, 이는 집합을 update할 때의 시간 복잡도를 잘못 예측한 탓이었다.

전체 프로그램의 공간 복잡도는 미로를 생성하는 데 필요한 공간을 의미한다. 따라서 O(HEIGHT \* WIDTH)의 공간 복잡도를 가진다.