11주차 결과보고서

전공: 수학/컴퓨터공학 학년: 3학년 학번: 20181288 이름: 윤성호

이번 실습에서는 미로의 너비와 높이를 입력하면, 입력한 사이즈의 완전 미로를 생성하는 것을 요구한다. 생성한 완전 미로는 .maz 파일로 저장하였다. 완전 미로는 eller’s algorithm을 이용하여 생성하는데, 수평연결과 수직연결을 처리해주는 과정이 각각 존재한다. 실습시간에 구현한 과정을 코드를 살펴보며 설명해보고자 한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 1]

[사진 1]은 프로그램에 필요한 헤더파일과 정의한 것을 보여준다. 가장 먼저, 프로그램에서 필요한 헤더파일을 #include를 이용하여 불러왔다. 기본적인 입출력과정을 위해 <stdio.h>를 불러왔고, 동적할당을 위한 malloc, calloc 함수가 있는 <stdlib.h> 또한 사용해주었다. 마지막으로 <time.h> 를 사용하였는데, 이는 rand() 함수를 이용하는데, 매번 실행할 때 마다 rand() 함수가 반환해주는 수를 다르게 하기 위해 사용하였다. Rand()함수가 반환해주는 수를 다르게 하기 위해서는 현재 프로그램의 시드 값이 달라야 하고, 매번 달라지는 값을 시드로 설정해주면 rand()함수가 정말 무작위의 값을 반환할 수 있기 때문에, 매번 달라지는 값으로 시간을 선택한 것이다. <time.h>로 현재 시간을 불러와, 그를 시드값으로 설정해주었다. 그리고 #define으로 min과 max 함수를 정의해주었다. 이는, 두 집합을 합해주는 과정에서 더 작은 수로 통일하는 방향으로 코드를 작성하였는데, 이때 자주 쓰이기에 사전에 미리 정의해두었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 2]

[사진 2]에서는 프로그램에서 필요한 전역변수들과 함수들이 선언되어 있는 부분이다. 먼저, 미로의 가로 사이즈를 N, 미로의 세로 사이즈를 M으로 선언해두었다. 그리고, 포인터변수로 prevMave와 nowMaze를 정의해두었는데, 이는 엘러 알고리즘을 동작시킬때 필요한 정보를 저장한다. 엘러 알고리즘은 현재행과 바로 아랫행만을 가지고 미로의 형태를 결정한다. 따라서, 전체 미로를 저장하게 되면 공간을 O(N\*M)을 사용해야 하지만, 인접한 두개의 행만을 저장하고 이를 이용한다면 공간을 O(2\*N)만 사용해도 같은 결과를 낼 수 있다. 다음으로는 TYPES, COUNT, INDEX, CONNECTED 라는 이름으로 포인터 변수를 정의하였다. 이들은 모두 엘러 알고리즘의 수평과정, 수직 과정 중 수직과정에서 사용되는 배열들이다. TYPES배열은 현재 탐색주인 행의 집합의 종류를 저장하는 배열이다. COUNT배열은 TYPES배열에 저장되어있는 집합의 종류가 각각 몇개 있는지 저장하는 배열이다. INDEX배열은 TYPES배열과 COUNT배열의 정보를 조금 더 직관적으로 사용하고자 만든 배열이다. 예를 들어 현재 TYPES = {5, 3, 7} 이고, COUNT = {2, 6, 4} 라고 하자. 이렇게 저장되어 있는 것의 의미는 현재 행에 5번집합은 2칸, 3번집합은 6칸, 7번집합은 4개가 있다는 의미이다. 엘러 알고리즘의 수직연결처리를 진행할 때는, 현재 탐색중인 칸의 번호는 쉽게 받아올 수 있다. 그러나, 현재 탐색중인 칸의 번호가 TYPES 배열상의 몇번째 인덱스에 저장되어있는지는 확인하려면 매번 TYPES 배열을 순회하여 찾아주어야 한다. 엘러 알고리즘 특성상 집합번호의 인덱스를 확인해야 하는 경우가 매우 빈번하여 시간복잡도가 굉장히 커진다. 이를 방지하기 위해 각 집합번호가 TYPES 배열에서 몇번 인덱스에 저장되어 있는지를 단숨에 알 수 있는 배열을 만들고자 하였고, 그게 INDEX 배열이다. 앞선 TYPES 배열과 COUNT 배열은 사이즈가 최대 N이지만, INDEX 배열은 사이즈가 N\*M+1로 설정되어야 한다. INDEX[집합번호] = TYPES 배열상에서 집합번호의 인덱스값으로 가져올 수 있어야 하는데, 칸에 배정될 수 있는 집합번호의 최댓값이 N\*M이기 때문이다. 그리고 다음으로 CONNECTED배열인데, TYPES 배열에 저장된 집합번호들이 아랫행과 연결되어있는지 여부를 저장하는 배열이다.

그리고 변수들 또한 선언해두었는데, connected는 rand()함수의 결과값을 저장하는 변수로, 현재 상황에서 연결을 할 지 말지를 저장하는 변수이다. Types 변수는 현재 탐색중인 행의 집합의 종류를 저장하는 변수이고, 1로 초기화한다. roomNum은 집합번호가 부여되지 않은 방들에게 어느 숫자를 배정해줄까를 저장하는 변수이고, 1로 초기화한다. Fp는 파일포인터로, .maz 파일을 생성하기 위해 사용된다. 아래 정의된 두개의 함수는 이름에서 보는 것과 같이 각각 수평연결과정, 수직연결과정을 처리하는 함수이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 3]

[사진 3]의 코드를 살펴보며 본격적으로 main함수의 동작을 살펴보고자 한다. 가장 먼저, srand함수를 이용해 시드값의 변화를 준다. 그리고 scanf를 이용하여 미로의 가로(N)과 미로의 세로(M)을 입력받는다. 그리고, fopen함수를 이용하여 20181288.maz를 write mode로 연다. Write mode는 파일이 없으면 새로 생성하고, 파일이 존재한다면, 파일 내부의 내용을 모두 지우고 새롭게 써내려가는 모드이다. 만약, 파일 오픈 에러가 등장한다면 “File open Error” 라는 에러 메세지와 함께 프로그램을 종료한다. 만약, 파일이 성공적으로 생성되었다면 “파일이 생성되었습니다.” 라는 문구를 출력해준다. 그 이후, prevMaze와 nowMaze를 동적할당해준다. 두 배열의 사이즈는 미로의 가로(N)으로 설정해주고 모든 원소를 0으로 초기화해준다. 이 두 과정을 한번에 진행해주기 위해 calloc함수를 사용하였다. 그 이후, 텍스트 파일에 미로를 출력해주기 시작한다. 텍스트 파일에 내용을 출력하는것은 fprintf를 이용한다. 가장 먼저, 가장 꼭대기 지점을 출력해야 한다. 가장 꼭대기는 벽을 가로막혀있다. 이를 표현해주기 위해서는 제일 윗 행의 짝수 지점은 “+”모양, 홀수지점은 “-“을 출력한다. 그 이후, 다음 줄에 미로를 출력해주어야 하므로, newline도 출력해준다. 이부분의 시간복잡도는 가장 꼭대기 지점을 출력하는 부분에 의해 결정되며, O(2\*N) = O(N)이라고 이야기할 수 있다. 또한, 공간복잡도는 prevMaze와 nowMaze 배열의 크기에 의해 결정되며 각각 O(N)이기 때문에 공간복잡도는 O(2N) = O(N)이다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 4]

[사진 4]에서는 수평연결처리 과정을 코드로 구현한 것이다. 모든 행에 대해서 같은 과정을 반복해주어야 한다. 가장 먼저, horizontalCheck(line)를 통해 line번째 행에 대해 수평연결처리를 해준다. 현재 행에 대해 수평연결처리가 끝났다면, TYPES, COUNT, INDEX 배열을 채워넣어 수직연결처리과정을 위한 준비를 한다. 이 세 배열은 동적할당으로 정의한다. TYPES배열은 집합의 종류를 저장할 배열로, 크기는 types이다. COUNT배열은 집합의 종류의 개수를 저장할 배열이고, 크리는 types이다. 0으로 초기화를 해주는 과정이 추가적으로 필요하기 때문에 calloc함수를 이용하여 동적할당하였다. INDEX 배열은 집합의 종류가 저장된 인덱스를 저장하기 위한 배열이고, 사이즈는 N\*M+1이다. INDEX 배열은 집합이 저장되어있는 인덱스를 저장하기때문에, -1로 초기화해준다. 그리고 tIndex라는 변수를 0으로 두는데, 이는 TYPES에 값을 저장할 때 필요한 변수이다. 그리고, TYPES, COUNT, INDEX를 채우러 간다. 모든 행에 대해서, 만약 현재 탐색중인 칸의 집합번호가 TYPES에 등록되어있지 않았다면, INDEX배열에 현재 집합번호의 값을 tIndex로 설정해주고, TYPES배열에 현재 집합번호를 넣어준다. 그리고 tIndex는 1 증가시켜 그 다음 집합번호를 저장할 수 있도록 한다. 그리고, 현재 집합번호에 해당하는 COUNT배열의 값을 1 증가시킨다. 이 과정이 끝났다면, nowMaze의 모든 집합번호를 prevMaze로 옮겨주고, nowMaze의 모든 집합번호는 0으로 초기화해준다. 이렇게 하면 수직연결처리를 위한 준비과정이 끝났다. 이 후, 다음칸에 미로를 출력해주어야하기 때문에 fprint를 이용하여 newline을 출력해준다.

이 부분의 시간복잡도는 horizontalCheck함수의 시간복잡도인 O(N^2), INDEX 배열을 초기화해주는 과정에서 O(N\*M), TYPES, COUNT배열을 채워주기 위해 O(N), 그리고 현재 행을 이전 행으로 바꿔주는 과정에서 O(N)이 걸린다. 이들을 모두 더해주면 O(N^2 + N\*M + N + N) = O(N^2)이다. 공간복잡도는 사용한 배열의 사이즈를 확인하면 O(N\*M)임을 알 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 5]

[사진 5]에는 수직연결처리를 해주고, 파일을 닫아주고 main함수를 종료하는 과정이 코드로 구현되어있다. 먼저, 수직연결처리는 verticalCheck(line)을 이용행 모든 행에 대해 수직연결처리를 진행해준다. 이 과정이 끝났다면 다음줄에 미로를 출력해주어야 하기 때문에 fprintf를 이용하여 파일에 newline을 출력해준다. 모든 행에 대해 수평연결처리와 수직연결처리를 진행해주었다면, fclose를 이용하여 파일을 닫아주다. 그리고 return 0을 통해 main함수를 종료한다.

이 부분의 시간복잡도는 verticalCheck의 시간복잡도인 O(N)이고, 공간복잡도는 verticalCheck의 공간복잡도인 O(N\*M)이다.

Main 함수의 시간복잡도는 [사진 3], [사진 4], [사진 5]의 과정을 M만큼 반복한 것이다. 즉, main함수의 O(M\*(N + N^2 + N)) = O(M\*N^2)이다.

Main 함수의 공간복잡도는 [사진 3], [사진 4], [사진 5]에서의 공간복잡도를 합하면 된다. 즉, O(N + N\*M + N\*M) = O(N\*M)이다.

Main 함수의 시간복잡도 및 공간복잡도는 곧 프로그램 전체의 시간복잡도 및 공간복잡도와 같다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 6]

[사진 6]은 수평연결 처리를 위한 horizontalCheck함수가 정의되어있는 부분이다. 가장 먼저, 모든 칸에 집합번호를 배정해주어야 하는데, 집합번호가 배정되어 있는 칸은 무시하고, 집합번호가 배정되어 있지 않은 칸은 전역변수로 저장된 roomNum을 이용하여 매번 다른 수로 방번호를 매겨준다. 이 과정에서의 시간복잡도는 O(N)이다. 그리고, 현재 행에서 탐색하는 집합의 개수를 types변수에 저장하기 위해 정의해놓고, 1로 초기화를 시켜준다. 그 이후, 모든 열에 대해서 체크를 해야하기 때문에 반복문을 이용해 검사를 진행한다. 먼저, 열의 가장 처음 부분이나 가장 끝부분은 무조건 벽이므로 fprintf를 이용하여 “|”를 출력해준다. 만약 첫번째 열이었다면, 다음 열을 탐색해야 하므로 continue를 이용해 반복문의 최상단으로 되돌아가지만, 만약 마지막 열이었다면, 더 이상 탐색할 열이 남아있지 않기 때문에 break를 이용하여 반복문을 빠져나온다.

그리고, 수평연결처리과정은 현재 탐색하는 행이 마지막 행인지 아닌지의 여부에 따라 다르게 처리된다. 먼저, 현재 탐색하는 행이 마지막 행인 경우에는 인접한 칸의 집합번호가 다르다면, 수직경로를 통해 연결 될 가능성이 없으므로, 필수적으로 연결을 시켜주어야 한다. 집합을 연결시켜줄때는, 인접한 두 방의 번호 중 큰 번호를 nowMax, 작은 번호를 nowMin이라고 하면 nowMin으로 통일시켜주는 방식으로 연결시켜주었다. 큰 번호, 작은 번호를 얻어 오기 위해서는 프로그램의 가장 윗부분에 #define 부분에 정의해둔 min, max 매크로를 이용하였다. nowMax와 nowMin을 얻어 왔다면, 현재 행의 모든 열을 순회하며 현재 탐색중인 방의 번호가 nowMax인 경우 nowMin으로 방번호를 변경해준다. 이 순회과정에서 또한 시간복잡도는 O(N)이다. 연결되었다는 것은 fprintf로 “ “을 파일에 출력해준다. 한편, 인접한 칸의 집합번호가 같을 때는 연결을 시켜주게 되면 불완전 미로가 생성되므로 벽을 설치해주어야 한다. 즉, fprintf를 이용하여 “|”를 파일에 출력해준다.

현재 탐색하는 행이 마지막 행이 아닌경우, 과정은 조금 더 복잡하다. 이 때는 인접한 칸의 집합번호가 다르더라도 수직경로로 연결될 수 있는 가능성이 있기에 무조건 연결시켜주지 않아도 된다. 따라서, 연결여부를 무작위로 결정해주어도 상관없다는 것을 의미한다. Connected 변수에 연결여부를 저장하는데, 이 때 rand()함수를 이용한다. Connected는 1이면 연결하라는 의미히고, 0이면 연결하지 말라는 것을 의미하도록 프로그램을 작성하였다. 따라서, connected가 1이면서 인접한 칸의 방번호가 다른 경우에는 두 방을 연결시켜준다. 두 방을 연결시켜줄 때는 위에서 언급했듯이 nowMax, nowMin을 이용하여 통일시킨다. 연결되었다는 것은 fprintf로 “ “를 파일에 출력하여 표시해준다. 만약, 방이 연결되어있거나, connected 가 0이면, 벽을 설치해야 한다. 벽은 “|” 모양으로 설치한다. 그리고 방이 연결되어 있지 않다는 것은 새로운 집합번호가 등장함을 의미하므로 types변수를 1 증가시킨다.

horizontalCheck의 시간복잡도는 가장 먼저 등장하는 nowMaze에 방번호를 배정하는 부분에서 O(N), 그리고, 연결여부를 판별할 때 모든 행을 돌아보며, 집합번호를 통일하는 과정은 O(N^2)이다. 다른 과정들은 모두 O(1)의 시간복잡도이므로 무시할 수 있다. 결과적으로 시간복잡도는 O(N+N^2) = O(N^2)이다.

horizontalCheck의 공간복잡도는 이용하는 전역변수의 사이즈를 통해 측정하고자 한다. TYPES, COUNT의 공간복잡도는 각각 O(types)이고, INDEX의 공간복잡도는 O(N\*M+1)이다. 또한, nowMaze배열은 O(N)의 공간복잡도를 가진다. 이들을 모두 합한다면, O(2\*types + N\*M+1 + N) = O(N\*M)으로 생각할 수 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

[사진 7]

[사진 6]은 수직연결 처리를 위한 verticalCheck함수가 정의되어있는 부분이다. 가장 먼저, 현재 탐색하는 행이 마지막인지 아닌지 여부에 따라 해야할 행동이 다르다. 만약 마지막 행이라면, 바닥부분에 벽을 설치해주어야 한다. 이는 탐색중인 칸이 짝수번째면 “+”를 출력하고, 홀수번째면 “-“를 출력한다. 이 과정이 종료되면 마지막행까지 모두 탐색이 완료된 것이므로 함수를 바로 종료한다. 마지막 행이 아니라면 수직연결 처리를 진행해주어야 한다. 먼저, CONNECTED배열을 calloc을 이용하여 동적할당 해주는데, 사이즈는 types로 해준다. 이렇게 한 후 모든 열을 탐색하며 수직 연결처리를 진행해준다. 인덱스 상 짝수지점은 모서리이므로“+”를 출력해준다. 인덱스 상 홀수지점은 벽일 수도 아닐 수도 있다. 만약, 현재 탐색중인 칸의 집합번호가 이전 행에 하나밖에 없는데 아직 연결이 되어 있지 않다면 무조건 연결을 시켜주어야 한다. 이는 이전 행의 집합번호를 현재 행의 집합번호가 되게 하고, 이전 행과 현재 칸은 연결되었으므로 현재 집합의 CONNECTED값을 1로 설정해준다. 그리고, 현재 탐색 중인 집합번호의 개수를 1 줄여준다. 그리고, 연결이 되었으므로 “ “를 fprintf를 이용하여 파일에 출력해준다. 만약, 무조건 연결시켜주어야 하는 상황이 아니면서, 현재 탐색중인 방의 집합번호가 이전 행에 하나 이상 남아있다면 연결을 해줄 수도 해주지 않을 수도 있다. 이 때는 연결여부를 무작위로 결정해야 하므로 connected 변수에 rand()함수를 이용하여 값을 배정해준다. Connected 값이 1이면 연결하라는 의미이고, 0이면 연결하지 말라는 의미이다. 연결을 해준다고 결정을 했다면, 이전 행의 집합번호가 현재 행이 되게 하고, 이전 행과 현재 칸은 연결되었으므로 현재 집합의 CONNECTED 값을 1로 설정해준다. 그리고, 연결이 되었으므로 “ “를 fprintf를 이용하여 파일에 출력해준다. 만약, 연결을 하지 않는다고 결정하면, 아무런 처리도 해주지 않고 fprintf를 이용하여 “-“를 출력해주면 된다. 이러한 과정이 끝나면 현재 탐색중인 집합번호의 개수를 저장한 COUNT배열의 값을 1 줄여준다.

VerticalCheck의 시간복잡도는 마지막행을 탐색하는 부분은 O(2N), 마지막이 아닌 행을 탐색하는 부분 또한 O(2N)이다. INDEX 배열이 없었다면 매번 TYPES, COUNT, CONNECTED배열을 이용하기 위해 모든 열을 다 탐색해주었어야 하지만, 덕분에 위 3개의 배열을 O(1)의 시간복잡도로 이용할 수 있게 되었다. 따라서, 시간복잡도는 O(2N) = O(N)이다.

VerticalCheck의 공간복잡도는 이용하는 전역변수의 사이즈를 통해 측정하고자 한다. TYPES, COUNT, CONNECTED의 공간복잡도는 각각 O(types)이고, INDEX의 공간복잡도는 O(N\*M+1)이다. 또한, nowMaze, prevMaze배열은 O(N)의 공간복잡도를 가진다. 이들을 모두 합한다면, O(2\*types + N\*M+1 + 2\*N) = O(N\*M)으로 생각할 수 있다.

1. 실험 전에는 Eller’s algorithm를 적용하기 위해서 전체 미로의 정보를 저장하는 크기가 N\*M인 이차원 배열이 필요하다고 생각하여, 이를 프로그램의 가장 초반에 정의한 후 이용하려고 계획을 세웠었다. 그러나, 실습을 진행하며 eller’s algorithm을 더 자세히 분석해본 결과, 전체 미로의 정보를 저장하고 있을 필요가 없다는 사실을 깨달았다. 그 이유는 알고리즘이 인접한 두개의 행의 정보만을 가지고 완전 미로를 생성하기 때문이다. 따라서, 크기가 N\*M 인 이차원 배열을 정의하여 미로의 정보를 저장하지 않고, 이전 행을 저장하는 크기 N인 배열과, 현재 행을 저장하는 크기 N인 배열 총 2개만을 정의해도 eller’s alrorithm을 이용하여 완전 미로를 생성할 수 있게 되었다. M값이 굉장히 커지면 메모리가 아주 많이 사용되게 되는데, 새롭게 자료구조를 구성하면 M값이 아무리 커진다고 해도 크기가 N인 두개의 배열만을 이용하면 완전 미로를 구성할 수 있게 된다. 컴퓨터공학설계및실험1 과목을 수강하며 가장 인상깊게 배운 점은 프로그램을 작성하면서 언제나 시간복잡도와 공간복잡도를 생각하야 한다는 점이다. 빠른시간내에 작동해야 하는 프로그램이나, 한정된 메모리 공간을 이용하여야 하는 프로그램을 작성해야 하는 상황을 앞으로 많이 마주하게 될 것이다. 그러한 상황에 미리 대비하여 어떤 프로그램을 짜든 시간복잡도와 공간복잡도를 신경쓰며 공부를 해야된다는 것을 느꼈다. 미로정보를 저장하는 방식 이외에는 실험 전에 생각한 방법을 모두 코드로 구현하였다.