12주차 MAZE3 결과 보고서

전공 : 수학 학년 : 4 학번 : 20151378 이름 : 진상우

1. 실험시간에 작성한 프로그램에서 자료구조와 구성한 자료구조를 화면에 그리는 방법들을 설명한다. 완성한 자료구조를 이용한 그래픽 전환 작업의 시간 및 공간복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술한다.

검은색, 사진, 어두운, 시계이(가) 표시된 사진

자동 생성된 설명

DFS의 경우 구현을 위해 경로 모든 것들을 저장하는 스택을 위한 구조체로 좌표의 정보를 가지며 미로의 크기에 따라 동적할당을 하기 위한 포인터를 선언해 주었다. Stack과 stackall을 선언해 주었는데, stack은 최소경로를 저장하고, stackAll은 탐색한 모든 경로를 저장한다.

Stack과 stackall를 동적할당을 진행한 후, stack은 row\*col만큼 할당, stackAll은 3\*row\*col만큼 할당 하였다.

DFS함수는 maze와 row, col 값을 사용하였고, 모든 경로를 저장하기 위한 좌표 저장 포인터를 배열처럼 사용하였다. 그리고 index를 저장하는 변수를 설정하였고, DFS를 실행하였는지 여부를 파악하기 위한 변수 sflag도 지정해 주었다. Dflag는 DFS와 BFS중 가장 최근에 실행한 함수가 무엇인지 파악하여 어떤 것을 그리는지 확인 하여 1이면 DFS를 0이면 BFS를 그렸다.

- 알고리즘

1) dflag를 1로 할당하고나서, sflag가 0이 아니라면 함수를 바로 끝낸다.

2) visit을 0으로 모두 초기화한다

3) V = 시작점, visit를 체크 한 뒤에, stack, stackAll에 push한다.

4) V의 좌표(r, c)가 도착점이면 10)으로 이동하고, 5) ~ 6) 에서는 visit이 false일 때와 visit이right\_flag이거나 down\_flag가 1일 때, 갈 수 있는 것으로 확인한다.

5) V의 right\_flag나, 우측점의 visit으로 갈 수 있는지 확인한다.

갈 수 있다면 V = 우측점을, visit을 체크하고, stack, stackAll에 push한 뒤, 4)로 이동하고,

갈 수 없다면 6)로 이동한다.

6) V의 down\_flag와 아래점의 visit으로 갈 수 있는지 확인한다.

갈 수 있다면 V = 아래점, visit을 체크하고, stack, stackAll에 push한뒤, 4)로 이동한다.

갈 수 없다면 7)로 이동한다.

7) V 좌측점의 right\_flag와 좌측점의 visit으로 갈 수 있는지 확인한다.

갈 수 있다면 V = 좌측점, visit 을 체크하고,, stack, stackAll에 push, 4)로 이동한다.

갈 수 없다면 8)로 이동한다.

8) V 위점의 down\_flag와 위의점의 visit으로 갈 수 있는지 체크

갈 수 있다면 V = 위의점, visit 을 체크하고,, stack, stackAll에 push, 4)로 이동한다.

갈 수 없다면 9)로 이동한다.

9) 갈 수 있는 점이 없다면 경로를 다시 되돌아와야 하기에

stack에서 pop을 하고, stackAll은 모든 경로를 저장하기에 돌아온 점을 다시 push한다. pop을 하는 과정에서 V의 좌표인 r, c를 이전의 점의 좌표로 수정한다. 4)로 이동

10) sflag를 1로 변경한 뒤에 함수를 종료한다.

- 시간복잡도 : O(row \* col)

visit의 초기화 row \* col

최악의 경우 모든 좌표를 다 방문해야하기 때문에 row \* col

- 공간복잡도 : O(row \* col)

visit : row \* col

stack : row \* col. stackAll : row \* col \* 3

<BFS>

사진, 검은색, 녹색, 도시이(가) 표시된 사진

자동 생성된 설명

BFS를 구현하기 위해 탐색 순서를 저장하는 구조체를 만들어 좌표와 어느 방향에서 왔는지 정보를 가지며 미로크기에 따라 할당을 위해 queueAll을 선언하였다. 이는 탐색한 모든 경로를 저장하낟.

파일을 읽는 과정에서 queueAll은 동적할당을 해주고, 이는 탐색한 모든 경로를 저장하며, 모든 길을 1번씩 지나가기에 미로의 크기인 row\*col만큼 할당하였다. 이후에 DrawBuffered함수에서 sflag가 1이고 dflag가 -1일때 방향에 맞춰 상하좌우로 길을 그리고, 0번째는 생략한다.

탐색한 모든 경로를 저장하기 위한 좌표와 방향을 저장하는 포인터를 할당을 이용하여 배열처럼 사용한다. Index를 저장하기 위해 변수도 만들고, BFS가 실행되었는지 파악하는 qflag도 지정해 준다. 위와 마찬가지로 dflag에서 DFS와 BFS 중 최근에 실행한 함수를 체크해서 무엇을 그려야 하는지 체크한다.

알고리즘

1) dflag를 -1로 할당한 뒤에 qflag가 0이 아니라면 함수를 바로 끝낸다.

2) visit을 0으로 모두 초기화 한다.

3) V = 시작점, visit를 체크 한 뒤에 queueAll에 enqueue, 최초 방향은 아무렇게나 잡아준다.

4) V = queueAll[k], r, c 는 V의 좌표, visit를 체크 한다.

5) V의 좌표 (r, c)가 도착점이면 queueA\_count를 해당 학목의 인덱스인 k로 맞춰주고 10)로 이동, 6) ~ 9) 에서는 visit이 false일 때와 right\_flag 혹은 down\_flag가 1일 때, 갈수 있는 것으로 판별한다.

6) V의 right\_flag와 우측의 점의 visit으로 갈 수 있는지 체크하고 7)로 이동한다.

이동할 수 있다면 우측의 점의 좌표와 ‘l’을 enqueue한다.

7) V의 down\_flag와 아래의 점의 visit으로 갈 수 있는지 체크하고 8)로 이동한다.

이동할 수 있다면 아래의 점의 좌표와 ‘u’을 enqueue한다.

8) V의 좌측의 점의 right\_flag와 좌측의 점의 visit으로 갈 수 있는지 체크하고 9)로 이동한다.

이동할 수 있다면 좌측의 점의 좌표와 ‘r’을 enqueue한다.

9) V의 위의 점의 down\_flag와 위의 점의 visit으로 갈 수 있는지 체크하고 k를 1만큼 늘려주고 4)로 이동한다.

갈 수 있다면 위의 점의 좌표와 ‘d’을 enqueue한다.

10) qflag를 1로 변경하고 함수를 종료한다.

- 시간복잡도 : O(row \* col)

visit의 초기화 row \* col

최악의 경우 모든 좌표를 다 방문해야하기 때문에 row \* col

- 공간복잡도 : O(row \* col)

visit : row \* col

queueAll : row \* col

1. 자신이 설계한 프로그램을 실행하여 보고 DFS와 BFS알고리즘을 서로 비교한다. 각각 알고리즘은 어떤 장단점을 가지고 있는지, 자신의 자료구조에는 어떤 알고리즘이 적합한지 관찰하고 설명한다.

시간이 부족하여 정확히 실행하여 보지는 못하였으나, DFS는 BFS보다 시간적으로 효율적이게 경로를 찾을 수 있고, BFS는 DFS와 비교하여 불완전미로에서 최단경로를 찾을 수 있기에 각각의 장단점이 존재할 것이다.