실험 13주차(Maze 2주차) 결과보고서

전공 : 국제한국학 학년 : 4 학번 : 20181202 이름 : 김수미

1. 실습 및 숙제로 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술한다. 완성한 알고리즘의 시간 및 공간 복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 기술한다.

1) 실습 프로그램

DFS 알고리즘 : Depth First Search

미로의 각 칸에서 4가지 방향으로 탐색을 한다. 이 때 이동가능한 방향이 발견 되는 경우 다른 방향은 이후에 고려하는걸로 하고 이동 가능한 방향이 발견되는 즉시 그곳으로 이동한다. 계속 이동하다가 더 이상 이동할 수 없는 막다른 길이 나오면 가장 마지막으로 만났던 갈림길로 돌아가 아까 고려하지 못했던 다른 방향으로 이동한다. 이것을 미로의 출구에 도착할 때 까지 반복한다.

자료구조 : 구조체 배열 STACK / 2차원 배열

막다른 길에 도착한 경우 지나온 길을 '되돌아가야' 하기 때문에, 출구에 도착하기 전까지는 계속 지나온 경로를 스택에 저장해야 한다. 배열에 경로를 차례대로 저장하면서 top 변수는 언제나 스택의 top, 즉 배열에 가장 마지막으로 추가된 원소의 index를 저장한다. 스택에서 원소가 pop 되면 top 의 값을 -1 해주면 된다(배열 index -1).

또한 한번 지나간 길을 다시 가게 되면 한번 들어갔던 막다른 길에 계속 들어갈 수 있으므로 지나간 미로의 셀을 표시해두기 위한 2차원 배열이 필요하다(지나간 셀 : 1, 가지 않은 셀 : 0).

시간 및 공간복잡도(N=미로의 가로, M=미로의 세로) : O(N*M) / O(N*M)

최악의 경우는 미로의 모든 셀을 방문한 다음에 출구에 도착하므로 이 때의 시간복잡도는 O(N*M)이다. 또한 미로의 각 셀에 대한 방문 여부를 저장해두어야 하기 때문에 언제나 O(N*M)만큼의 공간이 확보되어야 한다.

실험 전에 생각한 방법과의 차이

미로의 현재 셀에서 다음 셀로 이동이 가능한지 여부를 체크하는 공식을 짜는 것이 생각보다 어려웠다. 미로의 전체적인 정보를 2차원 구조체 배열로 저장하고 시작하면 쉬웠겠지만 그렇게 하는 경우 공간복잡도가 너무 커질 것 같아 입력 .maz 파일에 읽어들인 문자열 정보만으로 이를 계산하는 공식을 작성해 문제를 해결했다.

2) 과제 프로그램

BFS 알고리즘 : Breadth First Search

DFS와 마찬가지로 미로의 각 칸에서 4가지 방향으로 탐색을 한다. 그러나 이동 가능한 칸으로 동시에 진출하며 경로를 완성한다. 따라서 미로의 모든 셀을 방문할 기세로 경로를 확장해 나가다가 미로의 출구를 발견하면 그 확장이 중단된다. 여러개의 경로로 동시에 진출하며 경로를 저장하기 때문에 구조체 포인터 연결리스트를 통해 구현한 트리 구조와 QUEUE 자료구조를 사용했다.

자료구조 : 구조체 포인터 연결리스트 트리 / 구조체 배열 QUEUE / 2차원 배열

지나온 경로를 저장하기 위해 구조체 포인터 연결리스트 트리 자료구조를 사용했다. 각 노드는 현재 미로 셀의 위치 정보를 저장하는 필드와 parent cell(미로의 어느 셀로부터 왔는지)를 가리키는 포인터 필드를 가진다. 미로의 출구가 발견되면 출구 셀로부터 parent 포인터를 따라가 미로의 탈출 경로를 확인할 수 있다.

다음으로 탐색할 경로를 정하기 위해 QUEUE 자료구조를 사용했다. QUEUE의 구현은 구조체 배열을 사용했는데 배열에 새로운 값을 삽입하면 QUEUE 삽입이고, 배열의 가장 첫번째 값을 반환한 다음 배열의 가장 첫번째 원소를 그 다음 index의 요소로 지정해주면 QUEUE 삭제를 구현할 수 있다.

BFS에서도 DFS에서와 마찬가지로 미로의 각 셀에 대한 방문여부를 저장해두어야 하기 때문에(한번 갔던 길을 다시 가지 않도록) 이를 위한 2차원 배열이 필요하다.

시간 및 공간복잡도(N=미로의 가로, M=미로의 세로) : O(N*M) / O(N*M)

최악의 경우는 미로의 모든 셀을 방문한 다음에 출구에 도착하므로 이 때의 시간복잡도는 O(N*M)이다. 또한 미로의 각 셀에 대한 방문 여부를 저장해두어야 하기 때문에 언제나 O(N*M)만큼의 공간이 확보되어야 한다.

실험 전에 생각한 방법과의 차이

DFS처럼 출구를 찾으면 그 경로가 완성되는 줄 알았는데, 출구를 찾은 다음 다시 되돌아가며 경로를 완성해야 한다는 것을 코드를 작성하며 깨달았다. 또한 다시 되돌아가는 과정에서 스택이 필요할 것 같다고 생각했는데, 연결리스트를 사용하니 스택 없이도 경로를 완성할 수 있었다.

2. 자신이 설계한 프로그램을 실행하여 보고 DFS, BFS 알고리즘을 서로 비교한다. 각각의 알고리즘은 어떤 장단점을 가지고 있는지, 자신의 자료구조에는 어떤 알고리즘이 더 적합한지 등에 대해 관찰하고 설명한다.

1) DFS 알고리즘

장점 : 출구로 향하는 경로의 후보가 될 만한 길을 빠르게 탐색한다.

단점 : 미로의 어느 경로가 한참이 지나서야 막다른 길에 도착하는 경우 많이 헤멜 수 있다.

2) BFS 알고리즘

장점 : 여러 경로를 동시에 탐색하면서 다양한 경로의 정보를 빠르게 수집한다.

단점 : 여러 경로를 동시에 탐색해야 하기 때문에 동일한 시간 동안 DFS보다 더 많은 저장공간을 필요로 하게 된다.

3) 구현한 자료구조에 더 적합한 알고리즘

구현한 자료구조라기 보다는 '미로찾기'에 더 적합한 알고리즘은 DFS 라고 생각한다. 효율성을 떠나서, 사람이 미로찾기 퍼즐을 풀 때와 유사한 논리구조로 미로의 탈출경로를 탐색하기 때문에 프로그래머나 읽는 사람 입장에서 코드를 이해하고 활용하기 더 쉽기 때문이다.