13주차 예비보고서

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

1. 대표적인 그래프 탐색 알고리즘으로는 dfs 알고리즘과 bfs알고리즘이 있다. 이 두 알고리즘을 이용하면 어떠한 그래프의 모든 노드들이 연결되어 있다면, 중복없고 빠짐없이 순회할 수 있다. 그래프의 모든 원소를 순회할 수 있다는 것은 매우 큰 의미를 가지는데, 이를 통해 그래프의 연결 여부 파악이 가능하고, minimum spanning tree를 만들 때도 그래프 순회는 필수적이다. 그래프 알고리즘의 가장 기본이 되면서, 가장 중요한 dfs, bfs 알고리즘을 설명해보고자 한다.

<DFS>

DFS는 depth first search의 줄임말로, 깊이 우선 탐색이다. 이름에서 볼 수 있듯이 그래프를 계속 깊이 들어가는 방식으로 탐색한다. DFS는 recursive하게 구현하는 방법과 스택 자료구조를 이용하여 iterative하게 구현하는 방법이 모두 있는데, 시간복잡도 측면에서는 두 과정이 같기 때문에 아래에서는 recursive하게 구현된 dfs라고 가정하고 시간복잡도를 분석해보고자 한다. Dfs는 그래프 탐색 알고리즘이기 때문에 그래프가 먼저 구현이 되어 있어야 한다. 그래프는 일반적으로 두가지 방법으로 구현할 수 있다. 인접행렬 (adjacent matrix)와 인접리스트(adjacent list)이다. 그래프 구현 방식에 따라서 dfs의 시간복잡도가 달라진다. 구현에 앞서 방문여부를 저장할 배열을 하나 정의한다. 방문한 노드를 다시 방문하게 된다면 무한루프에 빠질수도 있고, 불필요하게 같은 노드를 여러 번 방문하게 되므로 이를 방지하기 위함이다. 먼저 인접행렬로 구현한 DFS를 살펴보자. 이 경우, 우리는 모든 노드를 시작점으로 하여 연결되어 있는 지의 여부를 먼저 판단해야 한다. 이는 반복문을 이용해서 모든 노드를 순회한다. 여기서 노드를 선택하여 그와 연결되어 있는 노드를 찾아야 하는데, 이는 인접 행렬의 특성상 연결되어 있는 노드를 바로 접근할 수 없고 모든 노드를 탐색하며 노드가 연결되어 있는지 아닌지를 판단해야 한다. 따라서, 이 경우에도 반복문으로 모든 노드를 탐색해야 한다. 만약 노드의 개수를 N개라 한다면 가장 먼저, 시작점이 되는 노드를 정하기 위해서 dfs함수의 호출이 N번 일어나고, 각 노드가 다른 노드와 연결되어 있는 지의 여부를 판단하기 위해 반복문은 N번 더돌아간다. 따라서, 인접행렬로 구현한 dfs 알고리즘의 시간복잡도는 O(N^2)이다. 다음으로는 인접 리스트로 구현한 dfs 알고리즘이다. 인접 행렬과 다른점은, 인접 리스트는 연결되어 있는 그 다음 노드를 저장하여서 시간복잡도 측면에서 인접 행렬보다 유리하다는 것을 알 수 있다. 인접 행렬은 연결된 다음 노드를 찾기위해 모든 노드를 순회해야 했지만, 인접 행렬은 연결된 노드를 탐색하기 위해서 전체적으로 노드의 개수만큼의 순회면 충분하기 때문이다. 인접 리스트로 구현한 dfs의 작동과정은 다음과 같다. 먼저, 노드를 선택하고, 그와 연결되어 있는 노드의 개수만큼 순회하며 방문했으면 무시하고, 방문하지 않으면 다시 dfs함수를 호출한다. 전체적으로 봤을 때는 dfs함수는 노드의 개수만큼 호출된다. 방문되지 않았으면 호출하고, 방문했으면 호출하지 않기 때문이다. 또한, 호출된 노드에 대해서, 연결된 간선만큼의 반복문을 수행한다. 현재 그래프는 undirected graph임을 가정하고 생각하고 있으므로 하나의 간선당 두번의 확인과정이 진행된다. 즉, 반복문은 간선의 2배만큼 진행된다. 최종적으로, 모든 반복문에서의 반복횟수는 않지만, 최종적으로는 간선의 2배만큼의 반복문이 진행된다. 노드의 개수를 N, 간선의 개수를 E라고 한다면, 인접 리스트로 구현된 DFS의 시간복잡도는 O(2\*E+N)이다. 따라서 O(N+E)라고 쓸 수 있다.

<BFS>

BFS는 breadth first search의 줄임말로, 너비 우선 탐색이다. 이름에서 볼 수 있듯이 그래프를 넓게 펴지는 방식으로 탐색한다. bfs알고리즘은 기본적으로 큐 자료구조를 이용하여 구현한다. 큐 자료구조는 삽입과 삭제 방법이 일정한 자료구조 중 하나인데, 삽입 및 삭제 방법은 fifo이다. 즉, 가장 먼저 자료구조에 삽입된 원소가 가장 먼저 자료구조에서 삭제된다는 것을 의미한다. BFS 알고리즘 또한 그래프 알고리즘이므로 그래프가 구현된 방법에 따라서 구현 방식이 달라진다. 구현에 앞서 방문여부를 저장할 배열을 하나 정의한다. 방문한 노드를 다시 방문하게 된다면 무한루프에 빠질수도 있고, 불필요하게 같은 노드를 여러 번 방문하게 되므로 이를 방지하기 위함이다. 먼저, 인접 행렬(adjacent matrix)로 구현된 그래프가 주어졌을 때의 bfs 알고리즘의 시간복잡도를 생각해보자. 인접 행렬은 다음 노드와의 연결여부를 배열의 값이 1이면 연결이 되어 있고, 0이면 연결이 되어 있지않다는 것으로 해석한다. 따라서, 어느 노드와 연결되어 있는지 파악하기 위해서는 모든 노드를 탐색하여 배열의 값을 얻어내야 한다. 즉, 하나의 노드 당 노드의 개수만큼의 반복문이 필요하다. 만약, 아직 방문하지 않았으며 연결되어 있는 노드를 발견하면 큐 자료구조에 삽입한다. 즉, 이러한 것을 만족하는 노드들은 알고리즘 전체적으로 봤을 때는 노드의 개수만큼이다. 이는 모든 노드에 대해서 노드의 개수만큼의 반복문이 실행된다는 것을 의미한다. 노드의 개수를 N이라고 하면, 인접행렬로 구현한 bfs 알고리즘의 시간복잡도는 O(N^2)이다. 다음으로는 인접 리스트(adjacent list)로 구현한 bfs 알고리즘에 대해 생각해보자. 이 경우는 연결된 노드 자체를 리스트에 저장하므로 각 노드에 대해 반복문은 그 노드와 연결되어 있는 노드의 개수만큼만 돌면 된다. 즉, 각 노드에 연결되어 있는 노드의 개수는 다르므로 매번 반복문이 돌아가는 횟수는 다르다. 하지만, 반복문은 연결되어 있는 노드의 개수만큼 돌아가는데, 하나의 간선당 두개의 노드가 연결이 되어 있으므로 프로그램 전체적으로 보면 간선의 개수의 2배만큼 반복문이 돌아간다. 이러한 반복문은 각 노드를 방문하였을 때 돌아간다. 노드의 개수를 N, 간선의 개수를 E라고 한다면, 인접 리스트로 구현한 bfs 알고리즘의 시간복잡도는 O(2\*E+N)이고, 이는 O(N+E)라고 말할 수 있다.

1. 3주차 실습에서는 dfs, bfs 알고리즘을 이용하여 실제 미로의 탈출경로를 찾는 것을 요구한다. 미로는 이차원 vector로 저장해놓았고, 이를 순회하며 실제 경로를 찾아 갈 수 있다. 먼저, dfs 알고리즘을 이용하여 실제 경로를 찾아보자. Dfs 알고리즘은 recursive한 구현과 iterative한 구현이 있는데, 실습에서는 iterative한 구현을 요구하기에 그렇게 코드를 작성할 계획이다. 먼저, 방문여부를 저장하고 있는 배열을 초기화한다. 방문여부를 저장하지 않으면 불필요한 중복이 많이 생길 수 있기 때문에 시간복잡도나 공간복잡도 측면에서 매우 불리하다. 또한, 무한루프에 빠질 수 있으므로 알고리즘이 제대로 동작하지 않을 수 있다. 그리고 실제 탈출경로를 저장할 pathDfs를 스택 자료구조로 정의한다. 미로는 (1,1)좌표에서 시작하므로 pathDfs 에 (1,1)을 추가해주고 (1,1)을 방문처리 해준다. 그리고 while loop를 pathDfs가 비어있지 않으면 동작하게 설정한다. 그리고 만약 끝 지점이 pathDfs에 삽입이 되었다면 탈출경로를 찾은 것이므로 종료를 한다. 그렇게 하고 현재 스택의 최상단 원소의 x, y좌표를 기준으로 아래쪽, 오른쪽, 위쪽, 왼쪽 순으로 탐색을 진행하고 벽이 있으면 아무 행동도 하지 않고, 벽이 없다면 이동가능하므로 새로운 위치를 pathDfs에 삽입해주고 그 좌표를 방문처리 해준다. 만약, 4방향 모두 탐색하였는데 이동가능한 위치가 나타나지 않았다면 그 노드는 탈출경로가 될 수 없으므로 pathDfs에서 삭제하고 다시 같은 과정을 반복한다. 이렇게 되면 최종적으로 pathDfs에 탈출경로의 좌표들이 저장되게 된다. 다음으로는 bfs 알고리즘을 이용하여 실제 경로를 찾아보고자 한다. bfs또한 dfs와 마찬가지로 방문여부를 저장할 visited 배열을 0으로 초기화한다. 방문여부를 저장하지 않으면 불필요한 중복이 많이 생길 수 있기 때문에 시간복잡도나 공간복잡도 측면에서 매우 불리하다. 또한, 무한루프에 빠질 수 있으므로 알고리즘이 제대로 동작하지 않을 수 있다. Bfs 알고리즘을 동작시키기 위한 큐 자료구조가 필요한데 이는 pathBfs라는 이름으로 저장한다. 최종적으로 이 배열에는 탈출경로가 들어있지는 않지만 bfs과정을 진행하는 데 꼭 필요한 자료구조이다. 그리고 실제 탈출경로를 저장하기 위해 Parent 배열을 하나 만든다. 그 배열에는 현재 좌표로 오기 전에 있던 좌표를 저장하고 있는 배열이다. Parent 배열을 모두 구성하게 된다면, 미로의 끝 부분에서부터 부모 좌표들을 타고 올라가며 미로의 시작 부분까지 갈 수 있게 되고, 그렇게 가는 경로가 곧 미로의 탈출경로가 된다. Bfs 또한 dfs와 마찬가지로 미로는 (1,1)좌표에서 시작하므로 pathDfs 에 (1,1)을 추가해주고 (1,1)을 방문처리 해준다. 그리고 while loop를 pathBfs가 비어있지 않으면 동작하게 설정한다. 그리고 만약 끝 지점이 pathBfs에 삽입이 되었다면 탈출경로를 찾은 것이므로 종료를 한다. 그렇게 하고 현재 큐의 최상단 원소의 x, y좌표를 기준으로 아래쪽, 오른쪽, 위쪽, 왼쪽 순으로 탐색을 진행하고 벽이 있으면 아무 행동도 하지 않고, 벽이 없다면 이동가능하므로 새로운 위치를 pathBfs에 삽입해주고 그 좌표를 방문처리 해준다. 또한, parent 배열에 새로운 위치의 부모 위치를 저장해준다. 이 과정을 반복하다가 미로의 끝 지점을 만나게 되면 함수는 종료되고, parent배열을 미로의 끝부분부터 따라가게 되면 최종적인 미로의 탈출경로를 알 수 있게 된다.