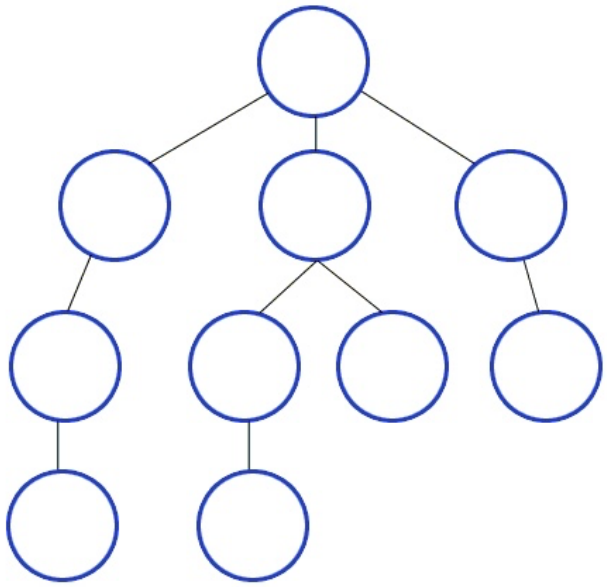
DFS & BFS

깊이 우선 탐색 (DFS, Depth-First Search)



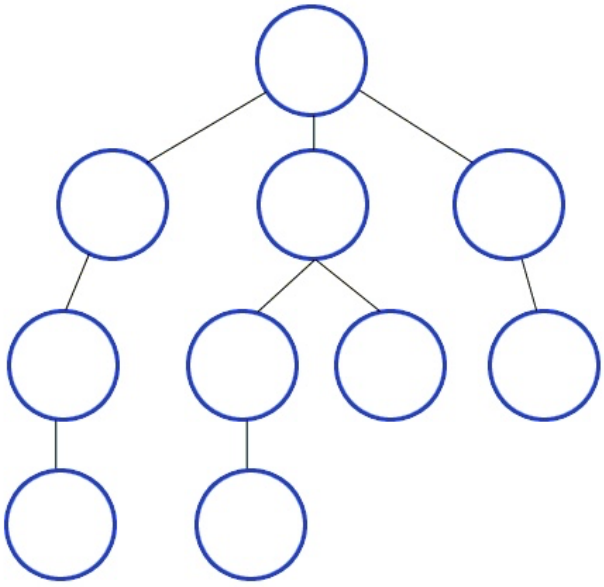
깊이 우선 탐색(Depth First Search, DFS)은 가장 기본적인 완전 탐색 알고리즘으로, 재귀를 이용하면 쉽게 구현할 수 있습니다. DFS는 **탐색 공간이 제한**되어 있고, **탐색 공간 내 탐색 목표가 있는지 검사**할 때 유용하게 사용됩니다.

DFS (with Java)

```java
import java.util.Stack;

class Graph {
    private int V;
    private LinkedList<Integer> adj[];

    Graph(int v) {
        V = v;
        adj = new LinkedList[v];
        for (int i=0; i<v; ++i)
            adj[i] = new LinkedList();
    }

    void addEdge(int v, int w) { adj[v].add(w); }

    void DFS(int s) {
        boolean visited[] = new boolean[V];
        Stack<Integer> stack = new Stack<>();
        stack.push(s);

        while (!stack.isEmpty()) {
            s = stack.pop();

            if (!visited[s]) {
                visited[s] = true;
                System.out.print(s + " ");
            }

            Iterator<Integer> itr = adj[s].iterator();
            while (itr.hasNext()) {
                int v = itr.next();
                if (!visited[v])
                    stack.push(v);
            }
        }
    }
}
```

위 코드에서는 DFS를 구현하기 위해 스택을 사용합니다. 방문한 노드를 스택에 저장하고, 스택이 빌 때까지 반복적으로 노드를 꺼내어 방문하며, 해당 노드와 연결된 노드를 스택에 추가합니다. 이를 반복하여 그래프의 모든 노드를 방문합니다.

너비 우선 탐색 (BFS, Breadth-First Search)



BFS (Breadth-First Search)는 그래프를 탐색하는 방법 중 하나로, 시작 정점에서 거리가 가까운 정점들을 먼저 탐색하는 방법입니다. 즉, 현재 정점과 인접한 정점을 모두 큐에 넣고, 그 다음 큐에서 꺼낸 정점과 인접한 정점들을 모두 큐에 넣는 방식으로 탐색을 진행합니다.

BFS (with Java)

```java
import java.util.*;

public class BFSExample {

    static int[][] map; // 인접 행렬
    static int[] visited; // 방문 여부를 저장할 배열

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int N = scanner.nextInt(); // 정점의 개수
        int M = scanner.nextInt(); // 간선의 개수
        int V = scanner.nextInt(); // 시작 정점

        // 인접 행렬과 방문 여부 배열 초기화
        map = new int[N + 1][N + 1];
        visited = new int[N + 1];
        for (int i = 1; i <= N; i++) {
            visited[i] = 0;
            for (int j = 1; j <= N; j++) {
                map[i][j] = 0;
            }
        }

        // 간선 정보 입력 받아 인접 행렬 생성
        for (int i = 0; i < M; i++) {
            int start = scanner.nextInt();
            int end = scanner.nextInt();
            map[start][end] = 1;
            map[end][start] = 1;
        }

        bfs(V, N);

    }

    public static void bfs(int start, int N) {

        Queue<Integer> queue = new LinkedList<>();
        visited[start] = 1;
        queue.offer(start);

        while (!queue.isEmpty()) {
            int current = queue.poll();
            System.out.print(current + " ");

            for (int i = 1; i <= N; i++) {
                if (map[current][i] == 1 && visited[i] == 0) {
                    visited[i] = 1;
                    queue.offer(i);
                }
            }
        }

    }

}
```

위 코드에서는 BFS 함수에서 Queue를 사용하여 BFS를 구현하고 있습니다. 시작 정점을 먼저 Queue에 넣고, Queue가 빌 때까지 반복문을 실행합니다. 반복문에서는 Queue에서 정점을 하나 꺼내어 출력하고, 해당 정점과 인접한 정점을 모두 Queue에 넣습니다. 이렇게 Queue에 넣은 정점은 이미 방문한 정점이므로 visited 배열을 통해 방문 여부를 체크해줍니다.