Depth First Search (DFS)

Definition

Depth first search is a recursive algorithm for searching all the vertices of a graph or tree. One interesting part of DFS is that it uses the idea of **backtracking**.

- It involves **exhaustive searchs of all nodes** by going ahead if possible, else by backtracking
  The vertices of the graph are categorized between:

1. Visited
2. Unvisited

The point of DFS like BFS is to mark each vertex as visited while avoiding cycles.

Pseudocode

Below you see the recursive DFS implementation:

```
function dfs(v):
        if v is unvisited do:
                visited[v] = true
        for each node in neighbors(v) do:
                dfs(node)
```

Below is a Iterative DFS implementation using a stack:

```
function dfs(v):
        let S be a stack
        S.push(v)
        visited[v] = true
        while S is not empty do:
                u = S.top()
                S.pop()
                for each node in neighbors(v):
                        if node is unvisited do:
                                S.push(node)
                                visited[node]
```

Source

Algorithm Description

The **backtracking** in DFS follows the following pattern:
1. You move forward down a path until there are no more nodes.
2. Move backwards on that same path until you find neighboring nodes to traverse.
3. All the nodes will be visited on the current path till all the unvisited nodes have been traversed.
4. After which the next path is selected.

Recursive Implementation

While DFS can be **implemented using a stack**, I personally prefer using just plain recursion for the sake of brevity.
Recall that recursive functions use the **call stack**. When a program calls the function, the function goes to the top of the call stack.
In the recursive implementation above, we **check if a vertex is unvisited**. If so we **mark** that vertex. Then we make a **recursive call on each of its neighbors**.

Iterative Implementation

The recursive nature of DFS can be implemented using a stack. First you pick a node to start from and push all of its adjacent nodes into the stack. Pop from the top of the stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. It's important to ensure that all of the visited nodes are marked

Time Complexity

The worst case time complexity is O(V + E).