Depth first search

The Depth First Search(DFS) is the most fundamental search algorithm used to explore the nodes and edges of a graph.

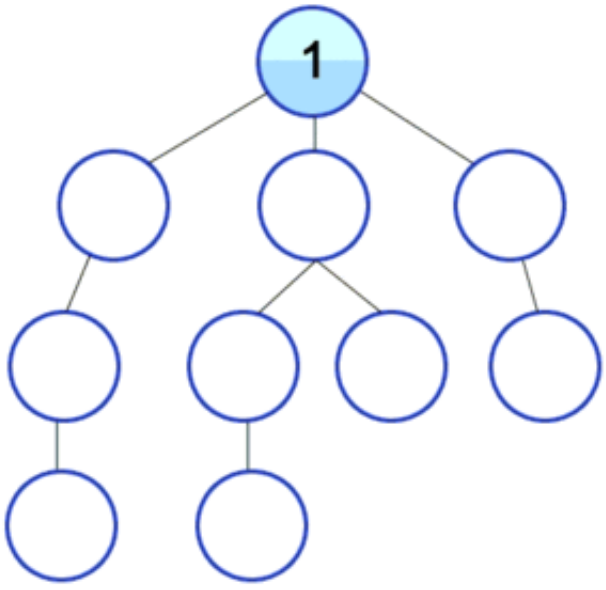
DFS is an algorithm for traversing or searching tree or graph data structures.It starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. It makes use of the Stack data structure to traverse graphs and trees.

Time cmplexity: O(V+E) , V = n. f ndes, E = no. of edges

Use:

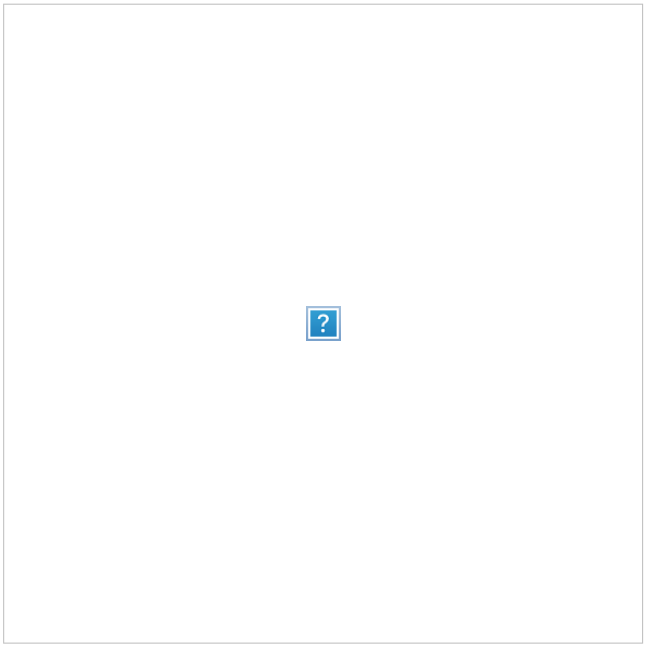
- Finding the bridges of a graph
- Detect cycles in a graph
- Finding connected components

Example:



[Reference: https://en.wikipedia.org/wiki/Depth-first_search]

2. DFS for graphs



Steps:

Consider an empty “Stack” that contains the visited nodes for each iteration. Our task here is as follows:

- Start at the root node and push it onto the stack.
- Check for any adjacent nodes of the tree and select one node.
- Traverse the entire branch of the selected node and push all the nodes into the stack.
- Upon reaching the end of a branch (no more adjacent nodes) ie nth leaf node, move back by a single step and look for adjacent nodes of the n-1th node.
- If there are adjacent nodes for the n-1th node, traverse those branches and push nodes onto the stack.

Method1: Recursion based DFS

Recursion method require two things: stop condition and repeat calls. stop condition will help finishing function call.

Method 2: Iteration based DFS

Here, Stack Data Structure is necessary to provide the stack functionality inherently present in the recursive function.

TASK1:

Complete the below code. Use inline comments as a hint. 5 lines are incomplete, write a code there. Do not touch any other code in GraphDFS class.

In [3]:

```
class GraphDFS:
    def __init__(self, graph, source):
        self.graph = graph
        self.source = source

    def dfs_non_recursive(self):
        if self.source is None or self.source not in self.graph:
            return "invalid input"      #Return invalid input message
        path = []                       #create empty list
        stack = [self.source]
        while(len(stack) != 0):
            s = stack.pop()#3                 #pop element from stack
            if s not in path:
                path.append(s)                 #Append s to path
            if s not in self.graph: #leaf node
                continue
            for neighbor in self.graph[s]:
                stack.append(neighbor)                 #Append neighbour to s
        return ",".join(path)

#Do not touch code below this line
graph = {"A":["B","C", "D"],
           "B":["E"],
           "C":["F","G"],
           "D":["H"],
           "E":["I"],
           "F":["J"]}

graphDFS = GraphDFS(graph, "A")
print("DFS on graph using iterative method: ")
print(graphDFS.dfs_non_recursive())

#similarly for graphs, DFS can be implemeted using recursion method too.
```

```
DFS on graph using iterative method:
A,D,H,C,G,F,J,B,E,I
```

1. DFS for trees

Depth first traversal can be done in three ways in binary tree:

- Inorder (Left, Root, Right)
- Preorder (Root, Left, Right)
- Postorder (Left, Right, Root)

1. Inorder traversal

- 1. Traverse the left subtree, i.e., call Inorder(left-subtree)
- 2. Visit the root
- 3. Traverse the right subtree, i.e., call Inorder(right-subtree)

2. Preorder traversal

- 1. Visit the root
- 2. Traverse the left subtree, i.e., call Preorder(left-subtree)
- 3. Traverse the right subtree, i.e., call Preorder(right-subtree)

3. Postorder traversal

- 1. Traverse the left subtree, i.e., call Postorder(left-subtree)
- 2. Traverse the right subtree, i.e., call Postorder(right-subtree)
- 3. Visit the root

TASK 2:

Complete the below code. Use inline comments as a hint. 3 lines are incomplete, write a code there. Do not touch any other code in DFS class.

In [24]:

```
# A class that represents an individual node in a Binary Tree
class Node:
    def __init__(self, key):
        self.left = None
        self.right = None
        self.val = key

#Que 1 -
#create class Node with left, right and val as a instance variables.
#create constructor in class and take key as argument in constructor.
# Assign None, None and key values to left, right and val instance variable r
```

```
3

class DFS:
    def __init__(self, root):
        self.root = root

    # Inorder tree traversal
    def printInorder(self):
        if root:
            printInorder(self.root.left) # First recur on left child
            print(self.root.val, end=' ') # then print the data of node
            printInorder(self.root.right) # now recur on right child

    # Postorder tree traversal
    def printPostorder(self):
        if root:
            printPostorder(self.root.left) # First recur on left child
            printPostorder(self.root.right)                          # the rec
            print(self.root.val, end=' ') # now print the data of node

    # Preorder tree traversal
    def printPreorder(self):
        if root:
            print(self.root.val, end=' ') # First print the data of node
            printPreorder(self.root.left)         # Then recur on left child
            printPreorder(self.root.right) # Finally recur on right child

# code executiona starts from here
root = Node(1)
root.left = Node(2)
root.right = Node(3)
root.left.left = Node(4)
root.left.right = Node(5)

dfs = DFS(root)
print("Preorder traversal of binary tree is")
dfs.printPreorder()

print("\nInorder traversal of binary tree is")
dfs.printInorder()

print("\nPostorder traversal of binary tree is")
dfs.printPostorder()
```

```
Preorder traversal of binary tree is
1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-05ae79240045> in <module>
     49 dfs = DFS(root)
     50 print("Preorder traversal of binary tree is")
---> 51 dfs.printPreorder()
     52
     53 print("\nInorder traversal of binary tree is")

<ipython-input-24-05ae79240045> in printPreorder(self)
     37         if root:
     38             print(self.root.val, end=' ') # First print the data of no
de
---> 39             printPreorder(self.root.left)         # Then recur on left
child
     40             printPreorder(self.root.right) # Finally recur on right ch
ild
     41

NameError: name 'printPreorder' is not defined
```

In [ ]: