1. Depth First Search

25/07/2019

INTRODUCTION

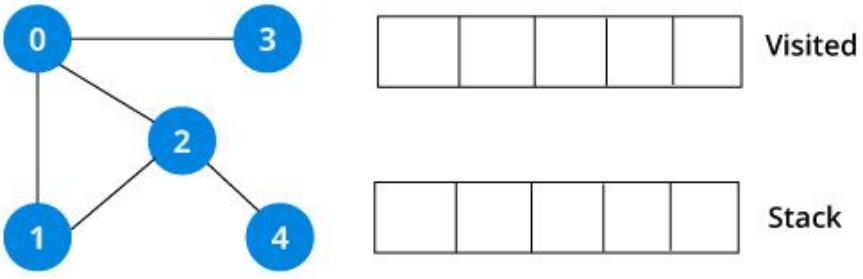
Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. The algorithm does this until the entire graph has been explored.

ALGORITHM OF DFS

```
1  procedure DFS(G,v):
2      label v as discovered
3      for all directed edges from v to w that are in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```

IMPLEMENTATION USING DATA STRUCTURE

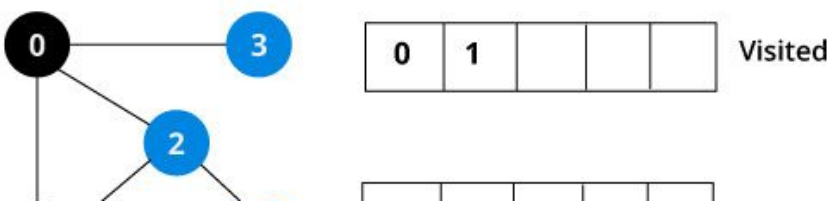
We use a **stack** to implement Depth First Search



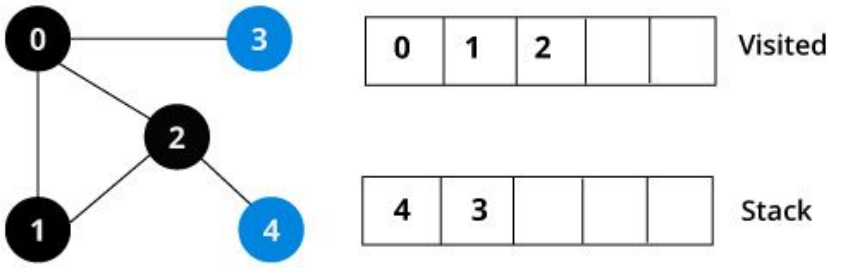
We start from vertex 0, the DFS algorithm starts by putting it in the Visited list and putting all its adjacent vertices in the stack.

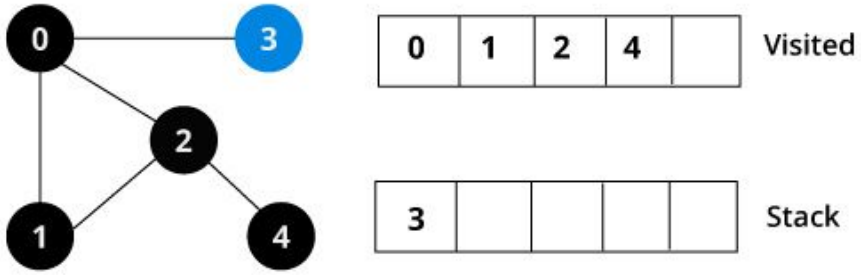
Next, we visit the element at the top of stack i.e. 1 and go to its adjacent nodes. Since 0 has already been visited, we visit 2 instead.

Vertex 2 has an

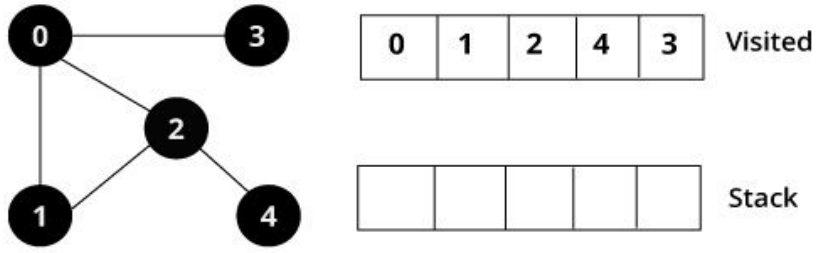


unvisited adjacent vertex in 4, so we add that to the top of the stack and visit it.





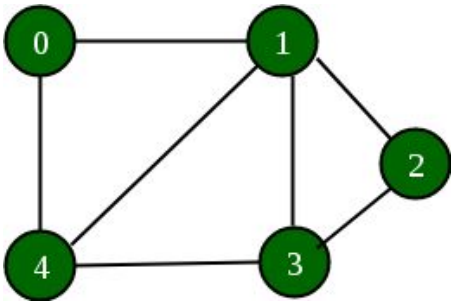
After we visit the last element 3, it doesn't have any unvisited adjacent nodes, so we have completed the Depth First Traversal of the graph.



PYTHON PROGRAM TO IMPLEMENT DFS

```
class Graph:

  def __init__(self, num):
    self.graph = [[] for i in range(num)]

  def addEdge(self, u, v, cost):
    self.graph[u].append((v, cost))
    self.graph[v].append((u, cost))
```

```
    def dfs(self, start, end):
        visited = [False for i in self.graph]
        stack = [start, ]
        visited[start] = True
        path = []

        while True:
            if len(stack) != 0:
                curr_node = stack.pop(-1)
            else:
                break
            path.append(curr_node)
            if curr_node == end:
                print("Path found from %d to %d (%s)" % (
                    start, end, ' -> '.join(map(str, path))
                ))
                break
            for node, _ in self.graph[curr_node]:
                if not visited[node]:
                    visited[node] = True
                    stack.append(node)

if __name__ == "__main__":
    graph = Graph(5)
    graph.addEdge(0, 1)
    graph.addEdge(0, 4)
    graph.addEdge(1, 2)
    graph.addEdge(1, 3)
    graph.addEdge(1, 4)
    graph.addEdge(2, 2)
    graph.addEdge(3, 4)

    graph.dfs(0, 2)
```



Output:

Path found from 0 to 2 (0 -> 4 -> 3 -> 1 -> 2)

APPLICATIONS OF DFS

1. Many problems in computer science can be thought of in terms of graphs. For example, analyzing networks, mapping routes, scheduling, and finding spanning trees are graph problems. To analyze these problems, graph-search algorithms like depth-first search are useful.
2. Depth-first searches are often used as subroutines in other more complex algorithms. For example, the matching algorithm, Hopcroft–Karp, uses a DFS as part of its algorithm to help to find a matching in a graph.
3. DFS is also used in tree-traversal algorithms, also known as tree searches, which have applications in the traveling-salesman problem and the Ford-Fulkerson algorithm.
4. Depth-first search is used in topological sorting, scheduling problems, cycle detection in graphs, and solving puzzles with only one solution, such as a maze or a sudoku puzzle.
5. Other applications involve analysing networks, for example, testing if a graph is bipartite.

ADVANTAGES AND DISADVANTAGES OF DFS

Advantages of DFS:

1. DFS consumes very less memory space.
2. It will reach at the goal node in a less time period than BFS if it traverses in a right path.
3. It may find a solution without examining much of search because we may get the desired solution in the very first go.

Disadvantages of BFS:

1. It is possible that may states keep reoccurring. There is no guarantee of finding the goal node.
2. Sometimes the states may also enter into infinite loops.

COMPLEXITY OF DFS

Depth-first search visits every vertex once and checks every edge in the graph once. Therefore, DFS complexity is O(V+E). This assumes that the graph is represented as an adjacency list.

CONCLUSION

If it is known that an answer will likely be found far into a tree, DFS is a better option than BFS. BFS is good to use when the depth of the tree can vary or if a single answer is needed—for example, the shortest path in a tree. If the entire

tree should be traversed, DFS is a better option. BFS always returns an optimal answer, but this is not guaranteed for DFS.

Why is python used instead of other languages like C++ for writing Artificial Intelligence programs?

Python has a standard library in development, and a few for **AI**. It has an intuitive syntax, basic control flow, and data structures. It also supports interpretive run-time, without standard compiler languages. This makes **Python** especially useful for prototyping algorithms for **AI**.Practically all of the most popular and widely used deep-learning frameworks are implemented in Python on the surface and C/C++ under the hood.

The main reason is that Python is widely used in scientific and research communities, because it's easy to experiment with new ideas and code prototypes quickly in a language with minimal syntax like Python.