<2T>

# Depth First Search (DFS) Description

Depth first search (also known as DFS) always expands the deepest node in the current frontier of the search tree. The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors. As those nodes are expanded, they are dropped from the frontier, so then the search "backs up" to the next deepest node that still has unexplored successors.

The depth-first search algorithm is an instance of a graph-search algorithm in similar to breadth-first-search but uses LIFO queue instead of bfs’ FIFO queue. A LIFO queue means that the most recently generated node is chosen for expansion. This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected.

As an alternative to the GRAPH-SEARCH-style implementation, it is common to implement depth-first search with a recursive function that calls itself on each of its children in turn.

# Depth First Search (DFS) Pseudocode

//DFS algorithm

function depth\_first\_search(graph G, node start, int end\_trait)

    // return if empty list

    if (start == nil)

        return nil

    var stack S

    for vertex in G

        vertex.visited = flase

    S.push(v)

    var node u

    while(!S.empty AND u.trait != end\_trait)

        u = S.pop()

        if (!u.visited)

            u.visited = true

            for vertex in u.neighbors

                if(!vertex.visited)

                    S.push(vertex)

# Depth First Search (DFS) Code

def dfs\_connected\_comps(graph, start):

    visited, stack = set(), [start]

    while stack:

        vertex = stack.pop()

        if vertex not in visited:

            visited.add(vertex)

            stack.extend(graph[vertex] - visited)

    return visited

def dfs\_paths(graph, start, goal, path=None):

    if path is None:

        path = [start]

    if start == goal:

        yield path

    for next in graph[start] - set(path):

        yield from dfs\_paths(graph, next, goal, path + [next])

graph = {'A': set(['B', 'C']),

         'B': set(['A', 'D', 'E']),

         'C': set(['A', 'F']),

         'D': set(['B']),

         'E': set(['B', 'F']),

         'F': set(['C', 'E'])}

path = list(dfs\_paths(graph, 'C', 'F')) # [['C', 'F'], ['C', 'A', 'B', 'E', 'F']]

test = dfs\_connected\_comps(graph, 'D') # {'E', 'D', 'F', 'A', 'C', 'B'}