Names: Rachel UWERA Done On 18/1/2020

Reg Number: 217205863

Department : Information Technology

# Difference between BFS and DFS

## **BFS**

Breadth First Search (BFS) algorithm traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search when a dead end occurs in any iteration.

## **DFS**

Depth First Search (DFS) algorithm traverses a graph in a depth ward motion and uses a stack to remember to get the next vertex to start a search when a dead end occurs in any iteration.

|  |  |  |
| --- | --- | --- |
| Key | BFS | DFS |
| Definition | BFS, stands for Breadth First Search. | DFS, stands for Depth First Search. |
| Data structure | BFS uses Queue to find the shortest path. | DFS uses Stack to find the shortest path. |
| Source | BFS is better when target is close to Source | DFS is better when target is far from source |

Stack and Queue both are the non-primitive data structures. The main differences between stack and queue are that stack uses LIFO (last in first out) method to access and add data elements whereas Queue uses FIFO (First in first out) method to access and add data elements.

## Breadth First Search (BFS) Algorithm

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

#### **Algorithmic Steps**

1. **Step 1**: Push the root node in the Queue.
2. **Step 2**: Loop until the queue is empty.
3. **Step 3**:Remove the node from the Queue.
4. **Step 4**: If the removed node has unvisited child nodes, mark them as visited and insert the unvisited children in the queue.

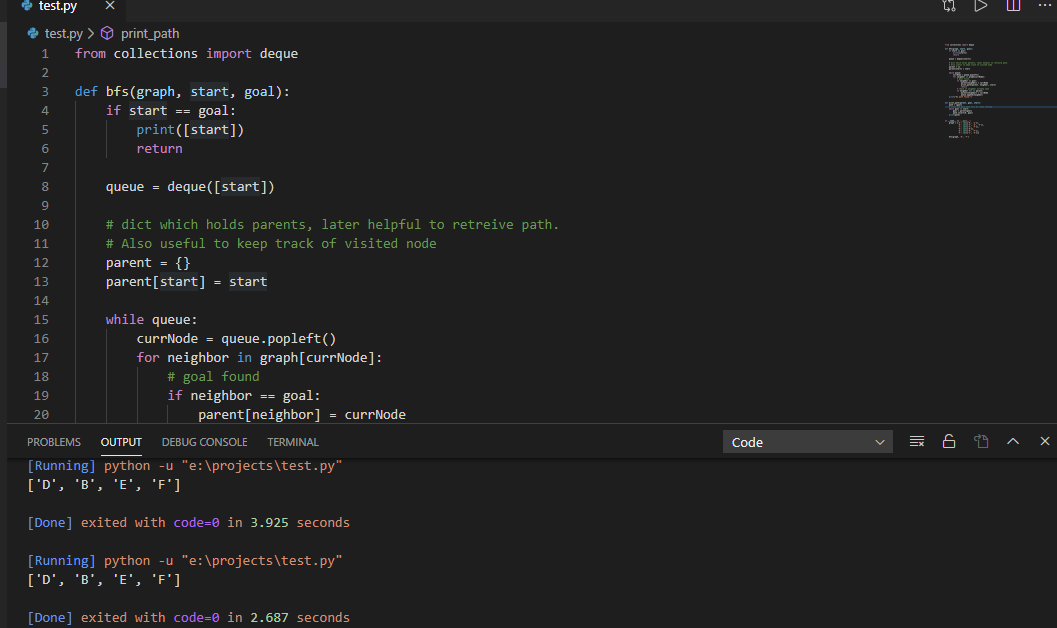
# Depth First Search (DFS) Algorithm

Depth first search (DFS) algorithm starts with the initial node of the graph G, and then goes to deeper and deeper until we find the goal node or the node which has no children. The algorithm, then backtracks from the dead end towards the most recent node that is yet to be completely unexplored.The data structure which is being used in DFS is stack. The process is similar to BFS algorithm. In DFS, the edges that leads to an unvisited node are called discovery edges while the edges that leads to an already visited node are called block edges.

#### **Algorithmic Steps**

1. **Step 1**: Push the root node in the Stack.
2. **Step 2**: Loop until stack is empty.
3. **Step 3**: Peek the node of the stack.
4. **Step 4**: If the node has unvisited child nodes, get the unvisited child node, mark it as traversed and push it on stack.
5. **Step 5**: If the node does not have any unvisited child nodes, pop the node from the stack.

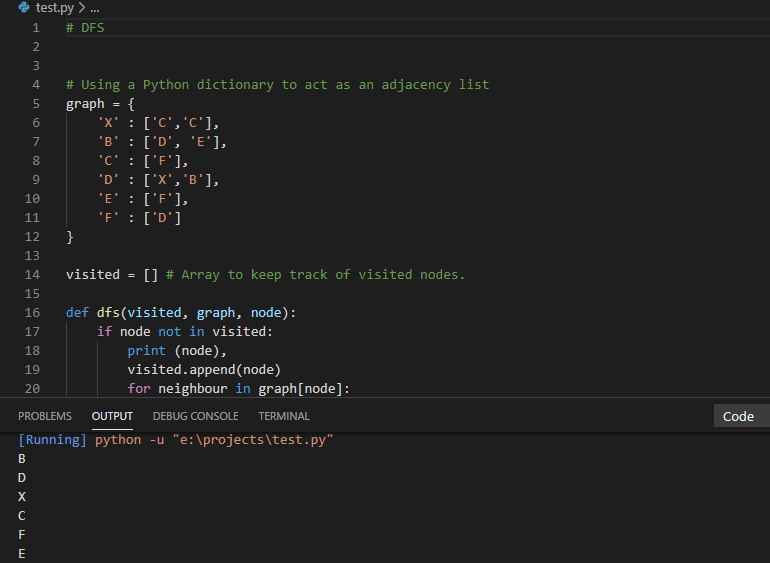
**BFS PROGRAM IN PYTHON Using vscode**

****

Source code of above program

|  |
| --- |
| from collections import deque  def bfs(graph, start, goal):  if start == goal:  print([start])  return  queue = deque([start])  # dict which holds parents, later helpful to retreive path.  # Also useful to keep track of visited node  parent = {}  parent[start] = start  while queue:  currNode = queue.popleft()  for neighbor in graph[currNode]:  # goal found  if neighbor == goal:  parent[neighbor] = currNode  print\_path(parent, neighbor, start)  return  # check if neighbor already seen  if neighbor not in parent:  parent[neighbor] = currNode  queue.append(neighbor)  print("No path found.")  def print\_path(parent, goal, start):  path = [goal]  # trace the path back till we reach start  while goal != start:  goal = parent[goal]  path.insert(0, goal)  print(path)  if \_\_name\_\_ == '\_\_main\_\_':  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'])}  bfs(graph, 'D', 'F') |

**DFS PROGRAM IN PYTHON Using vscode**



Source code of above program

|  |
| --- |
| # DFS  # Using a Python dictionary to act as an adjacency list  graph = {  'X' : ['C','C'],  'B' : ['D', 'E'],  'C' : ['F'],  'D' : ['X','B'],  'E' : ['F'],  'F' : ['D']  }  visited = [] # Array to keep track of visited nodes.  def dfs(visited, graph, node):  if node not in visited:  print (node),  visited.append(node)  for neighbour in graph[node]:  dfs(visited, graph, neighbour)  # Driver Code  dfs(visited, graph, 'B') |

Reference [https://www.codeproject.com/Articles/32212/Introduction-to-Graph-with-Breadth-First-Search-B](https://www.codeproject.com/Articles/32212/Introduction-to-Graph-with-Breadth-First-Search-BF)