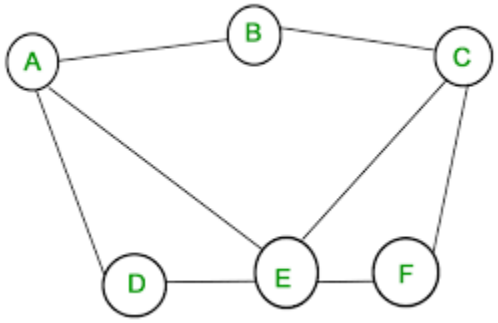
2. Represent a graph G as a combination of two sets V, E. If only these sets are used, how can DFS be implemented? Explain what additional information besides V and E may be required.

Let's take an example of the following graph. It is an undirected graph and hence every edge in E will be a bidirectional edge. So, if (u, v) is in E then it means there is an edge from u to v and from v to u.



Here, V = { A, B, C, D, E, F }
E = { (A, B), (B, C), (C, F), (C, E), (E, F), (E, D), (E, A), (D, A) }

In order to implement DFS, there should be some way of knowing all the neighbours of a given node. If we are only given sets of vertices and edges then finding all nodes that are adjacent to a given node will require us to traverse all of the edges. Also an additional array called "visited" will be required to keep track of the vertices that have been visited during the DFS so far.

Hence, for finding all the adjacent nodes of a given node, the time complexity will be O(m), where m is the number of edges in the graph. This is not optimal as in DFS, we have to find the adjacent nodes for all the nodes hence the time complexity of such DFS will be O(mn), where n is the number of vertices.

Therefore, although it is possible to implement DFS using only the set V and E(and the extra "visited" array), it is not the optimal way to do it.

We can instead create an adjacency list which can efficiently give all the neighbours of a given node. Hence, we can first create an adjacency list by traversing all the edges once which will be of O(m) complexity. Then, we can efficiently do DFS which will be of O(m + n) time complexity.

If an undirected graph is given with n nodes, labeled from 1 to n, and m edges then the algorithm for creating an adjacency list is as follows.

1. Create a variable called “adjList” which is an array of dynamic arrays.
2. Initialize “adjList” to have n empty array.
3. For edge in E:
    // edge[0], edge[1] represents the end points of the edge
    a. adjList[edge[0]].append(edge[1])
    b. adjList[edge[1]].append(edge[0])

Then to do the DFS, we need one additional array called “visited” of size n which will keep track of the vertices that have been visited so far during the DFS traversal. Algorithm of DFS with starting node V is as follows.

DFS(V):

1. visited[V] = 1
2. PRINT(V)
3. FOR i in adjList[V]:
    a. IF visited[i] == 0:
        i. DFS(i)