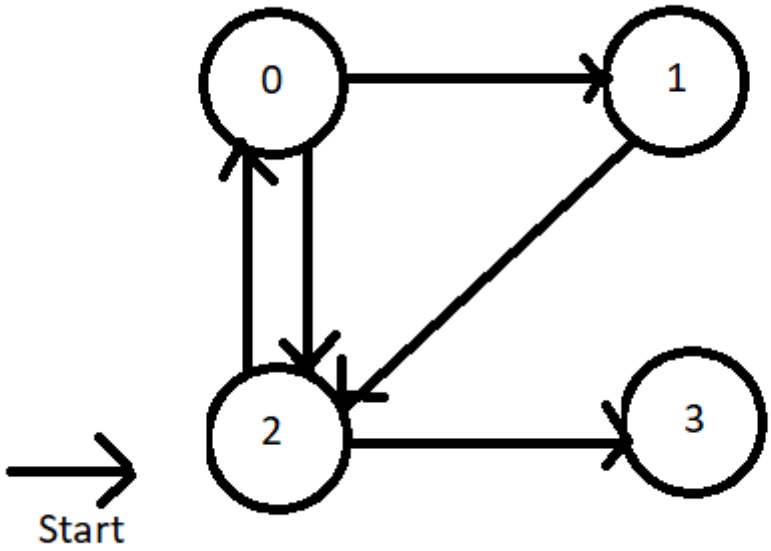
DEPTH FIRST SEARCH :

Depth First Search or DFS works on a vertical searching approach. The concept of DFS in graph is same as trees but the only difference is that unlike trees, graphs will contain cycles so we may have to visit the same node again. To avoid this case we use something called as flag or visited array.

Example:



Question:

Traverse the graph using Depth First Traversal

Solution:

As there is an arrow mentioning START we start from vertex 2. As we go to vertex 0, we look for adjacent vertices of it (which is vertex 0). But as in the above figure we can see that 2 is also an adjacent vertex to 0 which will bring us to the same point from where we started (2 will be processed again). So to avoid this kind of non-terminating process, the visiting array comes into picture. The visiting array will mark the visited nodes or vertexes as True so that the particular vertex won't be processed again but it's adjacent sides which has not been processed will be processed.

So by the above approach we can conclude

DFS of the following question:

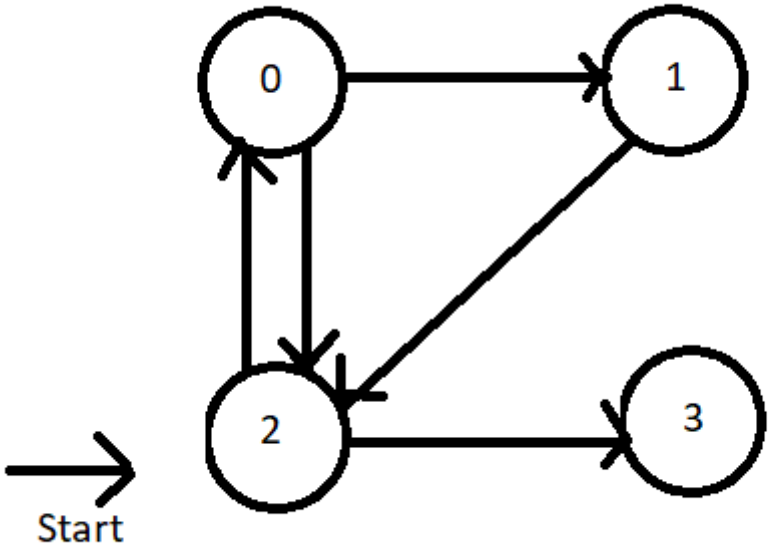
- Start from 2
- Go to 0 (adjacent)
- Instead of coming back to 2, look for it's other adjacent side if it's there and that is 1
- 1's adjacent side is 2 but as 2 is already visited we can dump that.
- The last vertex is 3 so process that via 2.

DFS = 2-0-1-3

Time Complexity:

Every code which a programmer writes should be efficient. To check efficiency 2 factors come in picture one is Space Complexity and the other is Time Complexity.

Time Complexity of the above graph:



Depth first search basically means to pick a vertex and traverse all it's edges. So by this statement:

a+2d = 2

a+2d = 0

a+d = 1

a+0d = 3

where,

a = no of vertexes

d = No of edges per vertex

Summing up we get : 4a+5d

So we can write O(V+E) = Time Complexity

Code for the same:

> `[# representing graph using adjaceny list (python dictionary)]()`
>
> `graph = {`
>
> `0: [1, 2],`
>
> `1: [0, 3, 4],`

```
 2: [0, 4],

3: [1],

 4: [1, 2]

}

# initializing visisted array

visited = [False for x in range(len(graph))]

def* dfs(*v*):

 *# runs for every vertex. This is because the condition in the loop*

 *# ensures that this line is not repeated for a vertex.*

 visited[v] = True

*# visiting node. printing vertex (end = " " means end the line with a space)*

print(v, *end*=" ")

*# runs for every edge*

for child in graph[v]:

 if not visited[child]:

  dfs(child)


 *# Time complexity: O(V+E)*

*def* fast_dfs(*v*):

 if visited[v]:

  return None

visited[v] = True

*# visiting node. printing vertex (end = " " means end the line with a space)*

print(v, *end*=" ")

for child in graph[v]:

  return dfs(child)

print("Using normal dfs: ", *end*=" ")

dfs(0)

#printing new line

print()

# reinitializing visisted array

visited = [False for x in range(len(graph))]
```

```
# running fast dfs, starting from vertex 0

print("Using fast dfs: ", *end*=" ")

fast_dfs(0)
```