Depth-First Search (DFS)

1. Definition:

- You start at a node, v, and go all the way down a path until you hit a dead-end. Then, you backtrack and go down other paths.

2. Properties:

- Like a BFS, a DFS also creates a non-unique spanning tree.
- A DFS also gives connected component information.
- Unlike a BFS, a DFS does not find the shortest path between 2 nodes. A DFS is faster than a BFS.

3. Algorithim:

- All vertices and edges start out unmarked.
- Start at vertex v and go as far as possible away from v visiting vertices.

- If the current vertex has not been visited, mark it as visited and the edge that is traversed as a DFS edge.

- If the current vertex has been visited, mark the traversed edge as a back-up edge, back up to the previous node.

- When the current vertex has only visited neighbours left, mark it as finished.

- Back-track to the first vertex that is not finished.

- Continue.

4. Implementing a DFS:

- We can use a stack (LIFO) to store the edges with the operations:

  1. push((u,v))
  2. pop()
  3. is_empty()

- Furthermore, we need to store these data in each node in order to determine whether an edge is a back-edge or a DFS edge:

  1. d[v]: Discovery time
  2. f[v]: Finish time

5. Complexity of DFS:

- Since DFS visits the neighbours of a node exactly once, the adjacency list of each vertex is visited at most once. Therefore, the total running time is $O(m+n)$.

6. DFS Edges:

- We can specify edges (u,v) in a DFS-tree according to how they are traversed during the search.

- If v is visited for the first time, then (u,v) is a tree-edge in a DFS tree.

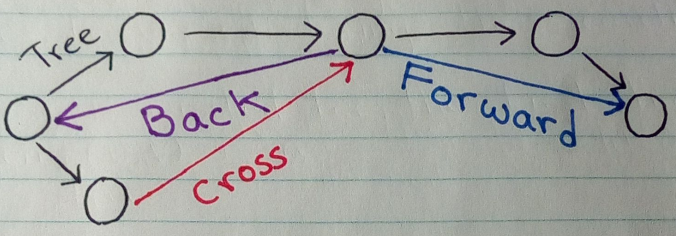
- If v has already been visited, then (u,v) is a:

  1. back-edge: An edge from a vertex u to an ancestor v in the DFS tree.

  2. forward-edge: An edge from a vertex u to a descendent v in the DFS tree.
     Note: This only applies to directed graphs.

3. Cross-edge: All the other edges that are not part of the DFS tree.
   I.e. v is neither an ancestor nor a descendant of u in the DFS tree.
   Note: This only applies to directed graphs.

- E.g.



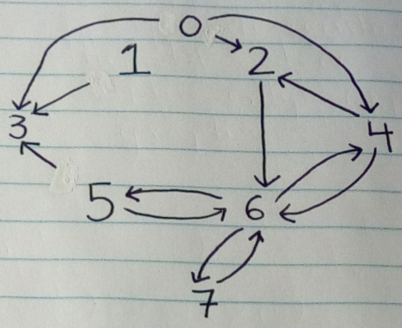
- We can use d[v] and f[v] to distinguish between the edges.

- There is a cycle in graph G iff there are any back-edges when DFS is run.

- We can detect a back-edge in a DFS if the vertex we are visiting has been visited but not finished.

7. Example of DFS

1. Consider the graph below.



1. Start at 0. Then, go to 2.

   Visited: 0, 2
   Stack: 2, 0

2. Go to 6.

   Visited: 0, 2, 6
   Stack: 6, 2, 0

3. Go to 4. Since all of 4's neighbours have been visited, we remove 4 from the stack, and back-track to 6.

   Visited: 0, 2, 6, 4
   Stack: ~~4~~, 6, 2, 0

4. From 6, we go to 5.

Visited: 0, 2, 6, 4, 5
Stack: 5, 6, 2, 0

5. From 5, we go to 3. Since we have visited all the neighbours of 3, we remove it from the stack.

Visited: 0, 2, 6, 4, 5, 3
Stack: ~~3~~, 5, 6, 2, 0

6. Since all of 5's neighbours have been visited, we remove 5 from the stack and back-track to 6.

Visited: 0, 2, 6, 4, 5, 3
Stack: ~~5~~, 6, 2, 0

From 6, we go to 7. Since all of 7's neighbours have been visited, we back-track to 6 and remove 7 from the stack.

Visited: 0, 2, 6, 4, 5, 3, 7
Stack: ~~7~~, 6, 2, 0

From 6, we back-track to 2 and remove 6 from the stack.

Visited: 0, 2, 6, 4, 5, 3, 7
Stack: ~~6~~, 2, 0

From 2, we back-track to 0 and remove 2 from the stack.

Visited: 0, 2, 6, 4, 5, 3, 7
Stack: ~~2~~, 0

Since all of 0's neighbours have been visited, we remove 0 from the stack and go to any unvisited nodes, which is 1.

Visited: 0, 2, 6, 4, 5, 3, 7
Stack: ~~0~~

From 1, there is nowhere to go because all of 1's neighbours have been visited. We remove 1 from the stack and are finished the DFS.

Visited: 0, 2, 6, 4, 5, 3, 7, 1
Stack: ~~1~~