**6\_Graph**

|  |  |  |  |
| --- | --- | --- | --- |
| Level 1 | | | |
| 1. Print Adjacency List |  | 1. Transitive Closure of a Graph |  |
| 1. BFS of Graph |  | 1. Union-Find |  |
| 1. DFS of Graph |  | 1. Detect Cycle using DSU |  |
| Level 2 | | | |
| 1. Connected Components in an Undirected Graph |  | 1. Mother Vertex |  |
| 1. Find the number of islands |  | 1. Unit Area of largest region of 1’s |  |
| 1. Detect cycle in an undirected graph |  | 1. Rotten Oranges |  |
| 1. Hamiltonian Path |  | 1. Minimum Swaps to Sort |  |
| 1. Prerequisite Tasks |  | 1. Steps by Knight |  |
| 1. Course Schedule |  | 1. Implementing Dijkstra Algorithm |  |
| 1. Circle of Strings |  | 1. Neeman’s Shoes |  |
| 1. Snake and Ladder problem |  | 1. Minimum Spanning Tree |  |
| 1. Bipartite Graph |  | 1. Strongly Connected Components (Kosaraju’s Algo) |  |
| 1. Maximum Bipartite Matching |  | 1. Bridge Edge in Graph |  |
| 1. Detect cycle in a directed graph |  | 1. Flood Fill Algorithm |  |
| 1. Find whether path exists |  | 1. Replace O’s with X’s |  |
| 1. Toplogical Sort |  | 1. Shortest Prime Path |  |
| 1. Level of Nodes |  | 1. Word Search |  |
| 1. Possible paths between 2 vertices |  | 1. Construct binary palindrome by repeated appending and trimming |  |
| 1. Find the number of ‘X’ total shapes |  | 1. Word Boggle |  |
| 1. Distance of nearest cell having 1 |  |  |  |
| Level 3 | | | |
| 1. Critical Connections |  | 1. Word Ladder I |  |
| 1. Minimum Cost Path |  | 1. Word Ladder II |  |
| 1. Strongly Connected Components (Tarjan’s Algo) |  | 1. Find number of closed islands |  |
| 1. Articulation Point – I |  | 1. Shortest Path by removing K walls |  |
| 1. Articulation Point – II |  | 1. Min Length String with All Substrings of Size N |  |
| 1. Alien Dictionary |  |  |  |

|  |
| --- |
| Link : <https://www.geeksforgeeks.org/top-50-graph-coding-problems-for-interviews/> |

**6\_Graph**

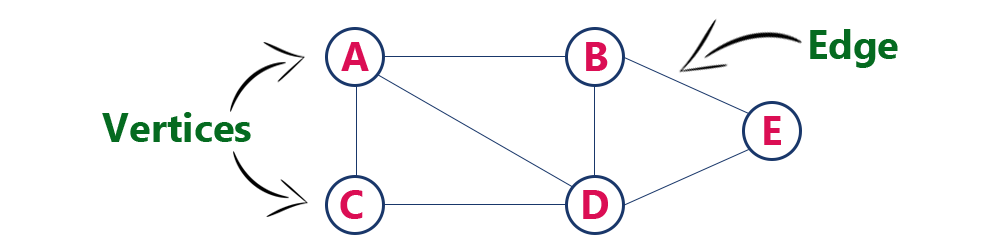
**6.1 Graph Representation in Data Structure(Graph Theory)|Adjacency Matrix and Adjacency List**

Most popular 2 method for representing graph in Computer Coding

1. Adjacency matrix
2. Adjacency list

**Adjacency matrix**

The adjacency matrix ins **n X n** matrix here **n** is number of **vertices**

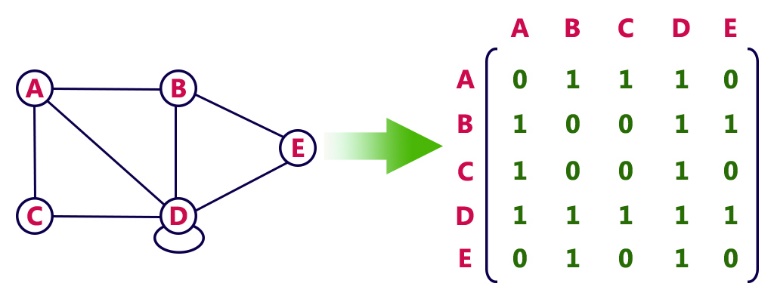
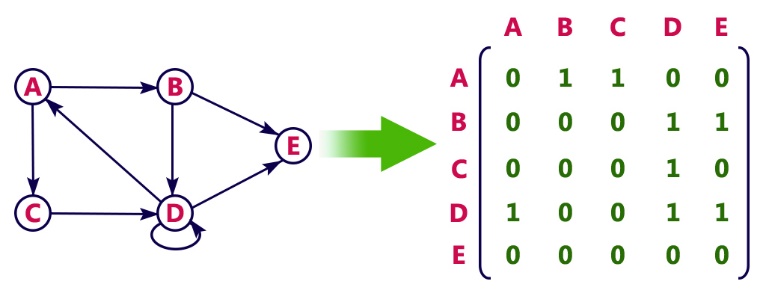


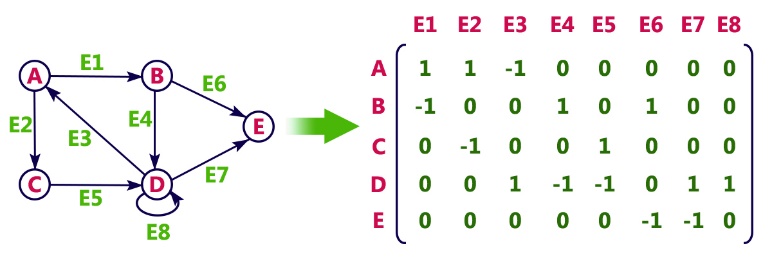
It is a matrix A[n][n]

Where n is number of vertices

& a[i][j] = 1 if i and j are adjacent

a[i][j] = 0 otherwise

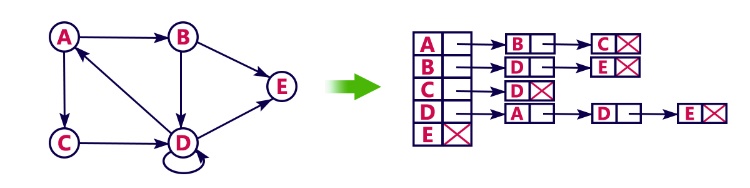
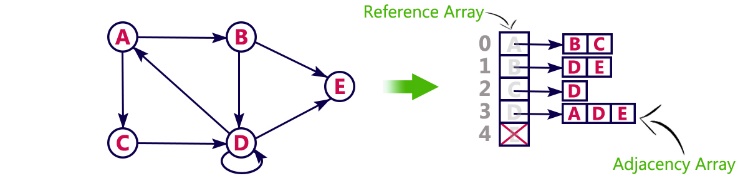


Space complexity: O(n^2)

cause the matrix is n X n

**Adjacency list**

We are going to have Linklist for each vertex one Linklist will be maintain

Space complexity: O(n + 2e)

Every edge has be written 2 times to 2e

* In case of **Dense graph** it is better to represent it using **adjacency matrix**
* In case of **Sparse graph** it is better to represent it using **adjacency list** (few number of edges are present)

**6.2 BFS and DFS Graph Traversals| Breadth First Search and Depth First Search**

BFS 🡪 Breadth First Search or Level Order Traversal

DFS 🡪 Depth First Search

**Breadth First Search**

You can start any node as root node and start the traversal

Something the root node is mentioned then start with that mentioned root

For BFS **Queue** is used

It is like exploration of one node

Numerous BFS traversal can be found

**BFS traversal** of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops. We use Queue data structure with maximum size of total number of vertices in the graph to implement BFS traversal.

We use the following steps to implement BFS traversal..

Step 1 - Define a Queue of size total number of vertices in the graph.

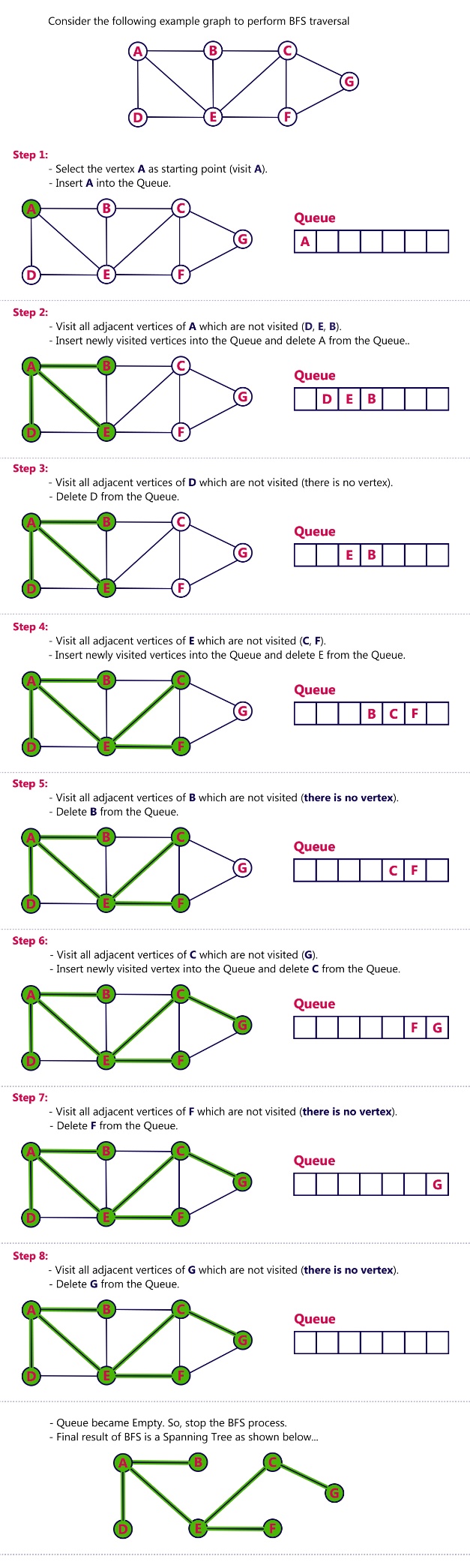
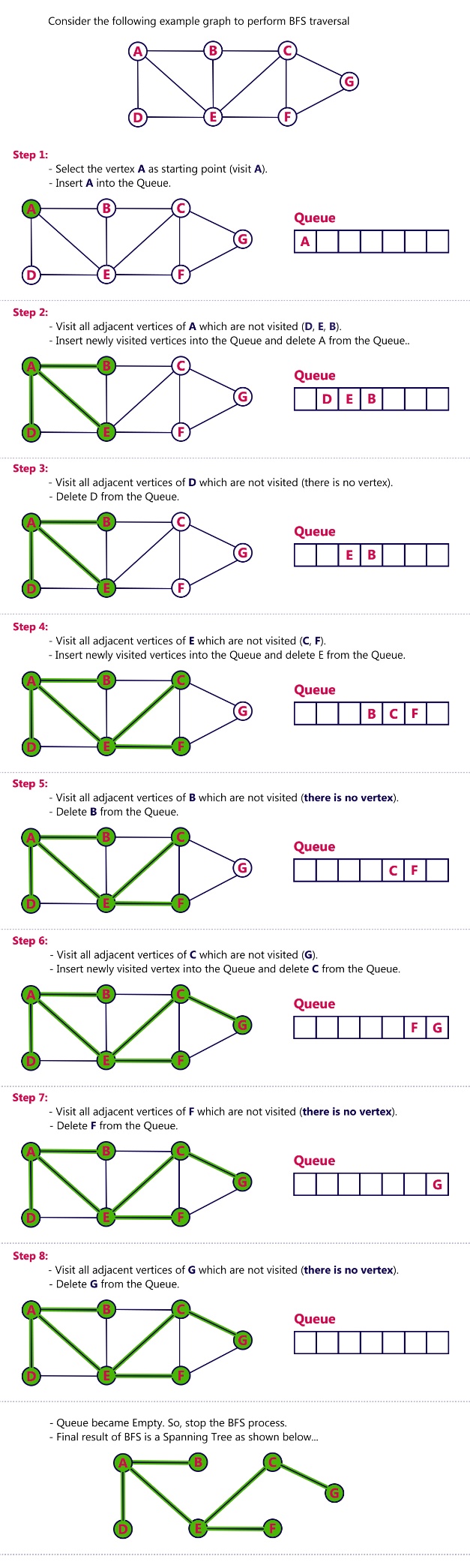
Step 2 - Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.

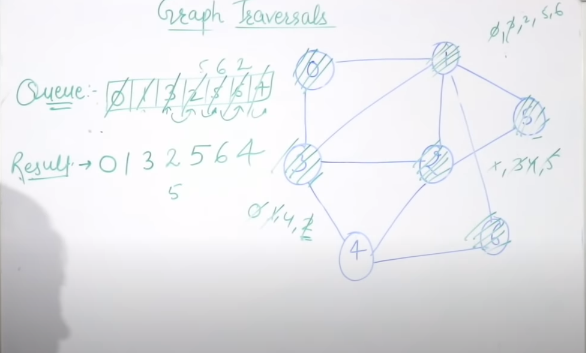
Step 3 - Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.

Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.

Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph



**Depth First Search**

You can take any node as root node and start from there

Target is to visit unvisited vertex

It has backtrack

Numerous way to do this

**DFS traversal** of a graph produces a spanning tree as final result. Spanning Tree is a graph without loops. We use Stack data structure with maximum size of total number of vertices in the graph to implement DFS traversal.

We use the following steps to implement DFS traversal...

Step 1 - Define a Stack of size total number of vertices in the graph.

Step 2 - Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.

Step 3 - Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.

Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.

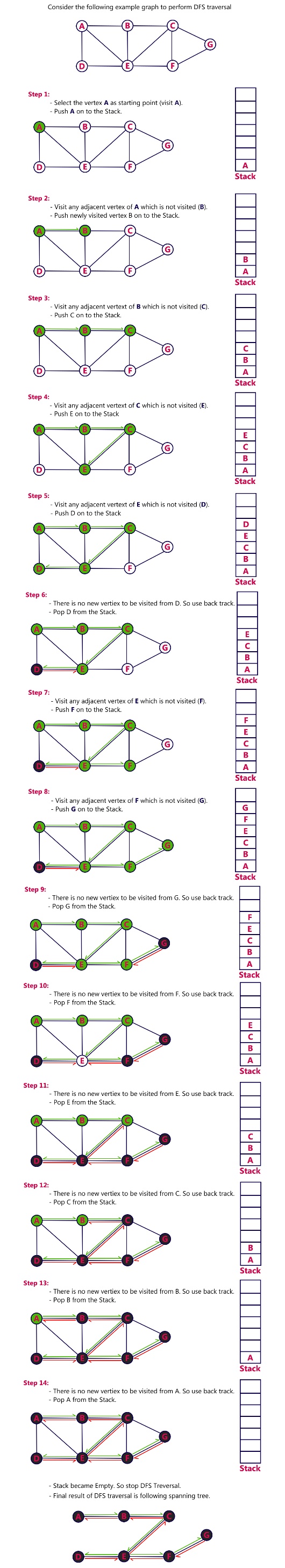
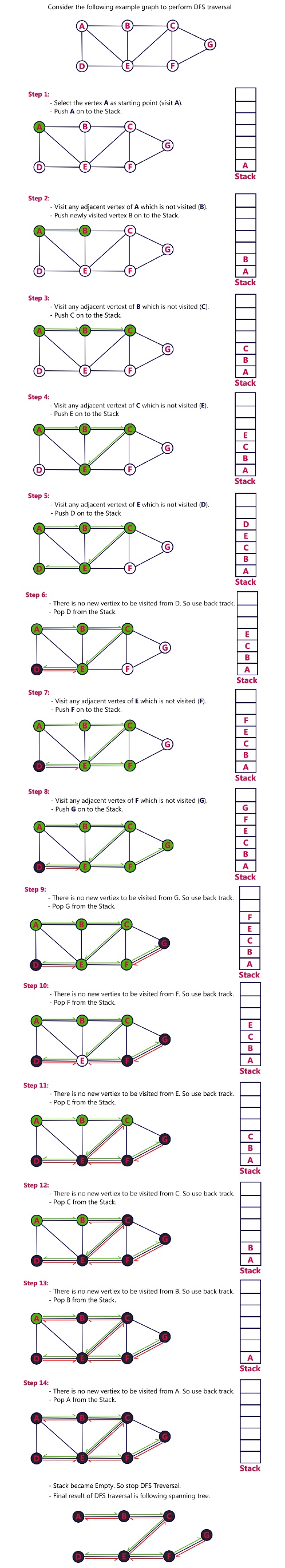
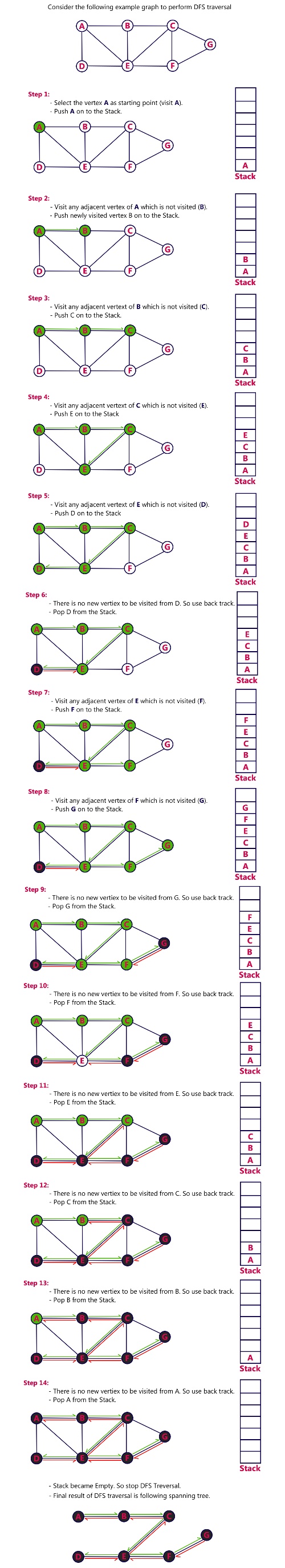
Step 5 - When there is no new vertex to visit then use back tracking and pop one vertex from the stack.

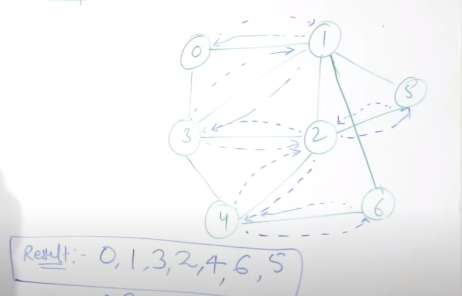
Step 6 - Repeat steps 3, 4 and 5 until stack becomes Empty.

Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

Back tracking is coming back to the vertex from which we reached the current vertex.

Example

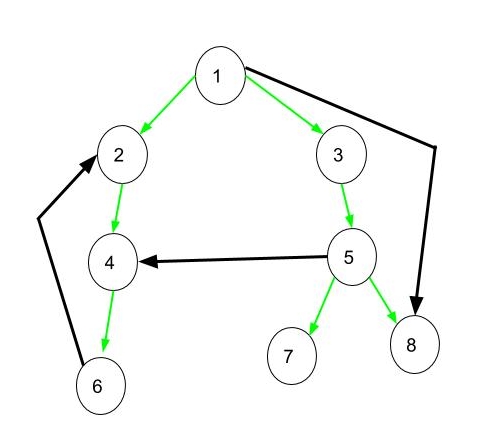
  

****

**6.3 Types of Edges in DFS | Edge Classification**

4 types of edges will be here

1. **Tree Edge:** It is an edge which is present in the tree obtained after applying DFS on the graph. All the Green edges are tree edges.
2. **Forward Edge:** It is an edge (u, v) such that v is a descendant but not part of the DFS tree. An edge from 1 to 8 is a forward edge.
3. **Back edge:** It is an edge (u, v) such that v is the ancestor of node u but is not part of the DFS tree. Edge from 6 to 2 is a back edge. Presence of back edge indicates a cycle in directed graph.
4. **Cross Edge:** It is an edge that connects two nodes such that they do not have any ancestor and a descendant relationship between them. The edge from node 5 to 4 is a cross edge.



Union of all the edges(tree, forward, back, cross) will get the total number of edges in the graph

