DFS of Graph :
https://practice.geeksforgeeks.org/problems/depth-first-traversal-for-a-graph/1

```
class Solution {
    // Function to return a list containing the DFS
traversal of the graph.
    ArrayList<Integer> ans = new ArrayList<>();
    public void dfs(int s , boolean[] vis ,
ArrayList<ArrayList<Integer>> adj){
        vis[s] = true;
        ans.add(s);
        for(int i=0 ; i<adj.get(s).size() ; i++){
            //adj.get(s) -> connection list
            //adj.get(s).get(i) -> connection list
element
            int x = adj.get(s).get(i);
            if(vis[x]==false){
                dfs(x,vis,adj);
            }
        }
    }
    public ArrayList<Integer> dfsOfGraph(int V,
ArrayList<ArrayList<Integer>> adj) {
        boolean[] vis = new boolean[V];
        dfs(0,vis,adj);
        return ans;
    }
}
```

BFS of Graph :
https://practice.geeksforgeeks.org/problems/bfs-traversal-of-graph/1

```
class Solution {
    // Function to return Breadth First Traversal
of given graph.
    public ArrayList<Integer> bfsOfGraph(int V,
ArrayList<ArrayList<Integer>> adj) {
        ArrayList<Integer> ans = new ArrayList<>();
        Queue<Integer> q = new LinkedList<>();
        boolean[] vis = new boolean[V];

        q.add(0);

        while(!q.isEmpty()){
            int v = q.remove();

            if(vis[v]==false){
                ans.add(v);
                vis[v] = true;

                //add all connections of v in q
                for(int i=0 ; i<adj.get(v).size() ;
i++){
                    //connection list = adj.get(v);
                    //element = adj.get(v).get(i)
                    q.add( adj.get(v).get(i) );
                }
```

```
            }
        }
        return ans;
    }
}
```

Dijkstra's Algorithm :
https://practice.geeksforgeeks.org/problems/implementing-dijkstra-set-1-adjacency-matrix/1

```
class Solution
{
    //Function to find the shortest distance of all
the vertices
    //from the source vertex S.
    static int[] dijkstra(int V,
ArrayList<ArrayList<ArrayList<Integer>>> adj, int
S)
    {
        boolean[] vis = new boolean[V];
        PriorityQueue<int[]> pq = new
PriorityQueue<>( (a,b) -> a[1]-b[1]);
        int[] ans = new int[V];
        Arrays.fill(ans,Integer.MAX_VALUE);
        ans[S] = 0;

        pq.add(new int[]{S,0}); //
{vertex,distance}

        while(!pq.isEmpty()){
```

```
            int[] x = pq.remove();     //0,0
            int vertex = x[0];

            if(vis[vertex] == true)
                continue;

            vis[vertex] = true;
            for(ArrayList<Integer> i :
adj.get(vertex)){
                int vi = i.get(0); //vertex
                int wt = i.get(1); //weight

                if(ans[vertex] + wt < ans[vi]){
                    ans[vi] = ans[vertex] + wt;
                }

                pq.add(new int[]{ vi , ans[vi]});
//vertex , distance
             }

        }
        return ans;

    }
}
```