Graphs

Objective

The objective of this lab is to understand the implementation of Graph data structure.

Task

1. Implement graph as adjacency list and understand its time and space complexity for each method. Create a graph where each vertex represents a person and an edge between two vertices represents their friend's relationship.

Procedure

a. A**djacency list:** Store graph as an adjacency list where one-dimensional array stores all vertices and within each vertex there is a friend's list.
b. Implement a depth-first search (DFS) algorithm and print the sequence of nodes in a DFS manner.

```
class vertex{
    String name;
    int age;
    Linkedlist friendsList=new Linklist();

   vertex(String d, int a){
          name=d;
           age=a;
 }
}

Class MyGraph{
   vertex[] adjList;
   int count;

MyGraph(int s){
   AdjList=new vertex[s];
   Count=0;
   }

public void AddVertex(String n, int a){ // add person name and age… }

public void AddEdge(String n1,String n2) {// add an edge between two person }

public void DFS() { // print all person in DFS sequence,
                        use stack that you implement earlier}

public void deleteVertex(String n){ // delete a person}

public void deleteEdge(String n1,String n2){ // delete friend's relation by
                                              removing an edge between two person }

public vertex FindVertex(String n){  // find a person  }

public String toString(){ // list all person and their friend's relationship  }

}
```

```
class node {
   vertex v;
   node next;
node(vertex ver){
   v=ver;
}

}

Class Linkedlist{

  public void insert(vertex v) { … }

  public void delete(vertex v){ … }

  public boolean find(vertex v){  … }

   public String toString() { … }

}
```

Pseudocode of DFS algorithm:

```
DFS (G, s)   //Where G is graph and s is source vertex
     let S be stack // Stack<Integer> stack = new Stack<>()
     S.push( s )          //Inserting s in stack
     mark s as visited.
     Print S
     while ( S is not empty){
        //Pop a vertex from stack to visit next
        v  =  S.pop( )
        //Push any neighbour of v in stack that is not yet visited
            w= Find unvisisted neighbour of v in Graph G
                if (w is not visited ){
                      S.push( w )
                      mark w as visited
                      print w
                  }
               else {
                    s.pop();
               }

        }// while end
```