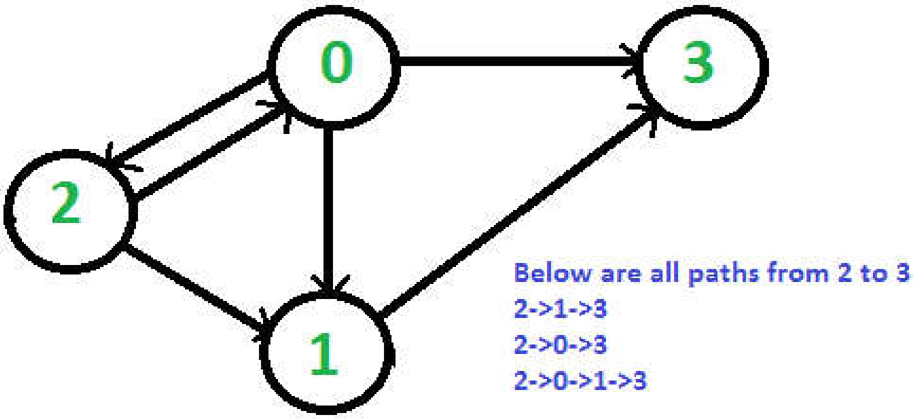
Print all paths from a given source to a destination

Given a directed graph, a source vertex ‘s’ and a destination vertex ‘d’, print all paths from given ‘s’ to ‘d’.

Consider the following directed graph. Let the s be 2 and d be 3. There are 4 different paths from 2 to 3.



Below are all paths from 2 to 3
2->1->3
2->0->3
2->0->1->3

Recommended: Please solve it on “*PRACTICE*” first, before moving on to the solution.

The idea is to do Depth First Traversal of given directed graph. Start the traversal from source. Keep storing the visited vertices in an array say ‘path[]’. If we reach the destination vertex, print contents of path[]. The important thing is to mark current vertices in path[] as visited also, so that the traversal doesn’t go in a cycle.

Following is implementation of above idea.

C/C++

```
// C++ program to print all paths from a source to destination.
#include<iostream>
#include <list>
using namespace std;

// A directed graph using adjacency list representation
class Graph
{
    int V;    // No. of vertices in graph
    list<int> *adj; // Pointer to an array containing adjacency lists

    // A recursive function used by printAllPaths()
    void printAllPathsUtil(int , int , bool [], int [], int &);

public:
    Graph(int V);   // Constructor
    void addEdge(int u, int v);
    void printAllPaths(int s, int d);
};

Graph::Graph(int V)
{
    this->V = V;
    adj = new list<int>[V];
}

void Graph::addEdge(int u, int v)
{
    adj[u].push_back(v); // Add v to u’s list.
}

// Prints all paths from 's' to 'd'
void Graph::printAllPaths(int s, int d)
{
    // Mark all the vertices as not visited
    bool *visited = new bool[V];

    // Create an array to store paths
    int *path = new int[V];
    int path_index = 0; // Initialize path[] as empty

    // Initialize all vertices as not visited
    for (int i = 0; i < V; i++)
        visited[i] = false;

    // Call the recursive helper function to print all paths
    printAllPathsUtil(s, d, visited, path, path_index);
}

// A recursive function to print all paths from 'u' to 'd'.
// visited[] keeps track of vertices in current path.
// path[] stores actual vertices and path_index is current
// index in path[]
void Graph::printAllPathsUtil(int u, int d, bool visited[],
                              int path[], int &path_index)
{
    // Mark the current node and store it in path[]
    visited[u] = true;
    path[path_index] = u;
    path_index++;

    // If current vertex is same as destination, then print
    // current path[]
    if (u == d)
    {
        for (int i = 0; i<path_index; i++)
            cout << path[i] << " ";
        cout << endl;
    }
    else // If current vertex is not destination
    {
        // Recur for all the vertices adjacent to current vertex
        list<int>::iterator i;
        for (i = adj[u].begin(); i != adj[u].end(); ++i)
            if (!visited[*i])
                printAllPathsUtil(*i, d, visited, path, path_index);
    }

    // Remove current vertex from path[] and mark it as unvisited
    path_index--;
    visited[u] = false;
}

// Driver program
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(0, 3);
    g.addEdge(2, 0);
    g.addEdge(2, 1);
    g.addEdge(1, 3);

    int s = 2, d = 3;
    cout << "Following are all different paths from " << s
         << " to " << d << endl;
    g.printAllPaths(s, d);

    return 0;
}
```

Run on IDE

Java

```
// JAVA program to print all
// paths from a source to
// destination.
import java.util.ArrayList;
import java.util.List;

// A directed graph using
// adjacency list representation
public class Graph {

    // No. of vertices in graph
    private int v;

    // adjacency list
    private ArrayList<Integer>[] adjList;

    //Constructor
    public Graph(int vertices){

        //initialise vertex count
        this.v = vertices;

        // initialise adjacency list
        initAdjList();
    }

    // utility method to initialise
    // adjacency list
    @SuppressWarnings("unchecked")
    private void initAdjList()
    {
        adjList = new ArrayList[v];

        for(int i = 0; i < v; i++)
        {
            adjList[i] = new ArrayList<>();
        }
    }

    // add edge from u to v
    public void addEdge(int u, int v)
    {
        // Add v to u's list.
        adjList[u].add(v);
    }

    // Prints all paths from
    // 's' to 'd'
    public void printAllPaths(int s, int d)
    {
```

```java
        boolean[] isVisited = new boolean[v];
        ArrayList<Integer> pathList = new ArrayList<>();

        //add source to path[]
        pathList.add(s);

        //Call recursive utility
        printAllPathsUtil(s, d, isVisited, pathList);
    }

    // A recursive function to print
    // all paths from 'u' to 'd'.
    // isVisited[] keeps track of
    // vertices in current path.
    // localPathList<> stores actual
    // vertices in the current path
    private void printAllPathsUtil(Integer u, Integer d,
                                    boolean[] isVisited,
                            List<Integer> localPathList) {

        // Mark the current node
        isVisited[u] = true;

        if (u.equals(d))
        {
            System.out.println(localPathList);
        }

        // Recur for all the vertices
        // adjacent to current vertex
        for (Integer i : adjList[u])
        {
            if (!isVisited[i])
            {
                // store current node
                // in path[]
                localPathList.add(i);
                printAllPathsUtil(i, d, isVisited, localPathList);

                // remove current node
                // in path[]
                localPathList.remove(i);
            }
        }

        // Mark the current node
        isVisited[u] = false;
    }

    // Driver program
    public static void main(String[] args)
    {
        // Create a sample graph
        Graph g = new Graph(4);
        g.addEdge(0,1);
        g.addEdge(0,2);
        g.addEdge(0,3);
        g.addEdge(2,0);
        g.addEdge(2,1);
        g.addEdge(1,3);

        // arbitrary source
        int s = 2;

        // arbitrary destination
        int d = 3;

        System.out.println("Following are all different paths from "+s+" to "+d);
        g.printAllPaths(s, d);

    }
}

// This code is contributed by Himanshu Shekhar.
```

Run on IDE

Python

```python
# Python program to print all paths from a source to destination.

from collections import defaultdict

#This class represents a directed graph
# using adjacency list representation
class Graph:

    def __init__(self,vertices):
        #No. of vertices
        self.V= vertices

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    '''A recursive function to print all paths from 'u' to 'd'.
    visited[] keeps track of vertices in current path.
    path[] stores actual vertices and path_index is current
    index in path[]'''
    def printAllPathsUtil(self, u, d, visited, path):

        # Mark the current node as visited and store in path
        visited[u]= True
        path.append(u)

        # If current vertex is same as destination, then print
        # current path[]
        if u ==d:
            print path
        else:
            # If current vertex is not destination
            #Recur for all the vertices adjacent to this vertex
            for i in self.graph[u]:
                if visited[i]==False:
                    self.printAllPathsUtil(i, d, visited, path)

        # Remove current vertex from path[] and mark it as unvisited
        path.pop()
        visited[u]= False


    # Prints all paths from 's' to 'd'
    def printAllPaths(self,s, d):

        # Mark all the vertices as not visited
        visited =[False]*(self.V)

        # Create an array to store paths
        path = []

        # Call the recursive helper function to print all paths
        self.printAllPathsUtil(s, d,visited, path)



# Create a graph given in the above diagram
g = Graph(4)
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(0, 3)
g.addEdge(2, 0)
g.addEdge(2, 1)
```

```
g.addEdge(1, 3)

s = 2 ; d = 3
print ("Following are all different paths from %d to %d :" %(s, d))
g.printAllPaths(s, d)
#This code is contributed by Neelam Yadav
```

Run on IDE

Output:

```
Following are all different paths from 2 to 3
2 0 1 3
2 0 3
2 1 3
```

This article is contributed by **Shivam Gupta**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Practice Tags : DFS Graph Backtracking

Article Tags : Backtracking Graph DFS

Login to Improve this Article

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Count all possible paths between two vertices
Find the minimum cost to reach destination using a train
Count all possible walks from a source to a destination with exactly k edges
Find if there is a path between two vertices in a directed graph
Depth First Search or DFS for a Graph
Print all paths from a given source to a destination using BFS
Backtracking | Introduction
Recursive program to generate power set
Smallest number with given sum of digits and sum of square of digits
Minimize number of unique characters in string

(Login to Rate)

3.2 Average Difficulty : **3.2/5.0**
Based on **57** vote(s)

Basic Easy Medium Hard Expert

□ Add to TODO List

□ Mark as DONE

Feedback Add Notes Improve Article

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

A computer science portal for geeks

710-B, Advant Navis Business Park,
Sector-142, Noida, Uttar Pradesh - 201305
feedback@geeksforgeeks.org