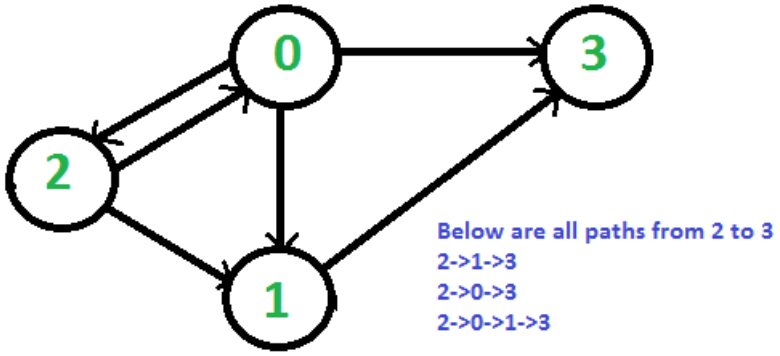
## Print all paths from a given source to a destination

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.




**We strongly recommend to minimize your browser and try this yourself first**

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 C++ implementation of above idea.

```
// 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;
}
```

Output:

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