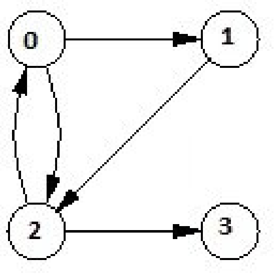
Transitive Closure of a Graph using DFS - GeeksforGeeks

Given a directed graph, find out if a vertex v is reachable from another vertex u for all vertex pairs (u, v) in the given graph. Here reachable mean that there is a path from vertex u to v. The reach-ability matrix is called transitive closure of a graph.

```
For example, consider below graph
```




```
Transitive closure of above graphs is
     1 1 1 1
     1 1 1 1
     1 1 1 1
     0 0 0 1
```

We have discussed a $O(V^3)$ solution for this here. The solution was based Floyd Warshall Algorithm. In this post a $O(V^2)$ algorithm for the same is discussed.

Below are abstract steps of algorithm.

1. Create a matrix tc[V][V] that would finally have transitive closure of given graph. Initialize all entries of tc[][] as 0.
2. Call DFS for every node of graph to mark reachable vertices in tc[][]. In recursive calls to DFS, we don't call DFS for an adjacent vertex if it is already marked as reachable in tc[][].

Below is C++ implementation of the above idea. The code uses adjacency list representation of input graph and builds a matrix tc[V][V] such that tc[u][v] would be true if v is reachable from u.

```
// C++ program to print transitive closure of a graph
#include<bits/stdc++.h>
using namespace std;

class Graph
```

```
{
    int V; // No. of vertices
    bool **tc; // To store transitive closure
    list<int> *adj; // array of adjacency lists
    void DFSUtil(int u, int v);
public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w) { adj[v].push_back(w); }

    // prints transitive closure matrix
    void transitiveClosure();
};

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

    tc = new bool* [V];
    for (int i=0; i<V; i++)
    {
        tc[i] = new bool[V];
        memset(tc[i], false, V*sizeof(bool));
    }
}

// A recursive DFS traversal function that finds
// all reachable vertices for s.
void Graph::DFSUtil(int s, int v)
{
    // Mark reachability from s to t as true.
    tc[s][v] = true;

    // Find all the vertices reachable through v
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (tc[s][*i] == false)
            DFSUtil(s, *i);
}

// The function to find transitive closure. It uses
// recursive DFSUtil()
```

```
void Graph::transitiveClosure()
{
    // Call the recursive helper function to print DFS
    // traversal starting from all vertices one by one
    for (int i = 0; i < V; i++)
        DFSUtil(i, i); // Every vertex is reachable from self.

    for (int i=0; i<V; i++)
    {
        for (int j=0; j<V; j++)
            cout << tc[i][j] << " ";
        cout << endl;
    }
}

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

    cout << "Transitive closure matrix is \n";
    g.transitiveClosure();

    return 0;
}
```

Output:

```
Transitive closure matrix is
1 1 1 1
1 1 1 1
1 1 1 1
0 0 0 1
```


References:

http://www.cs.princeton.edu/courses/archive/spr03/cs226/lectures/digraph.4up.pdf

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