Transitive Closure of a Graph using DFS

Difficulty Level : Medium • Last Updated : 11 May, 2021

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 on 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[][].

reachable from u.

C++

```
// 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];
```

```
// 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.
     if(s==v){
        if(adjList[v].contains(v))
    // 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;
}
```

Java

```
// JAVA program to print transitive
```

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

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

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

        // To store transitive closure
    private int[][] tc;

    // Constructor
    public Graph(int vertices) {

             // initialise vertex count
             this.vertices = vertices;
             this.tc = new int[this.vertices][this.vertices];

             // initialise adjacency list
             initAdjList();
    }

    // utility method to initialise adjacency list
    @SuppressWarnings("unchecked")
    private void initAdjList() {

        adjList = new ArrayList[vertices];
        for (int i = 0; i < vertices; 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);
    }

    // The function to find transitive
    // closure. It uses
    // recursive DFSUtil()
    public void transitiveClosure() {
        for (int i = 0; i < vertices; i++) {
            dfsUtil(i, i);
        }

        for (int i = 0; i < vertices; i++) {
          System.out.println(Arrays.toString(tc[i]));
        }
    }

    // A recursive DFS traversal
    // function that finds
    // all reachable vertices for s
    private void dfsUtil(int s, int v) {

        // Mark reachability from
        // s to v as true.
       if(s==v){
        if(adjList[v].contains(v))
          tc[s][v] = 1;
          }
      else
        tc[s][v] = 1;

        // Find all the vertices reachable
        // through v
        for (int adj : adjList[v]) {
            if (tc[s][adj]==0) {
                dfsUtil(s, adj);
            }
        }
    }

    // Driver Code
    public static void main(String[] args) {

        // Create a graph given
        // in the above diagram
        Graph g = new Graph(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);
        System.out.println("Transitive closure " +
                "matrix is");
```

```
}


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

Python

```
# Python program to print transitive closure of a graph
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)

        # To store transitive closure
        self.tc = [[0 for j in range(self.V)] for i in range(self.V)]

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

    # A recursive DFS traversal function that finds
    # all reachable vertices for s
    def DFSUtil(self,s,v):

        # Mark reachability from s to v as true.
         if(s==v){
        if(adjList[v].contains(v))
          tc[s][v] = 1;
          }
      else
        self.tc[s][v] = 1

        # Find all the vertices reachable through v
        for i in self.graph[v]:
            if self.tc[s][i]==0:
                self.DFSUtil(s,i)
        # Call the recursive helper function to print DFS
        # traversal starting from all vertices one by one
        for i in range(self.V):
            self.DFSUtil(i, i)
        print self.tc

# Create a graph given in the above diagram
g = Graph(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)

print "Transitive closure matrix is"
g.transitiveClosure();

# This code is contributed by Neelam Yadav
```

C#

```
// C# program to print transitive
// closure of a graph.
using System;
using System.Collections.Generic;

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

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

  // adjacency list
  private List<int>[] adjList;

  // To store transitive closure
  private int[,] tc;

  // Constructor
  public Graph(int vertices)
  {
    // initialise adjacency list
    initAdjList();
  }

  // utility method to initialise adjacency list
  private void initAdjList()
  {

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

  // add edge from u to v
  public void addEdge(int u, int v)
  {

    // Add v to u's list.
    adjList[u].Add(v);
  }

  // The function to find transitive
  // closure. It uses
  // recursive DFSUtil()
  public void transitiveClosure() {

    // Call the recursive helper
    // function to print DFS
    // traversal starting from all
    // vertices one by one
    for (int i = 0; i < vertices; i++) {
      dfsUtil(i, i);
    }

    for (int i = 0; i < vertices; i++) {
      for(int j = 0; j < vertices; j++)
        Console.Write(tc[i, j] + " ");
      Console.WriteLine();
    }
  }

  // A recursive DFS traversal
  // function that finds
  // all reachable vertices for s
```

```
     if(s==v){
        if(adjList[v].contains(v))
          tc[s][v] = 1;
          }
      else
    tc[s, v] = 1;

    // Find all the vertices reachable
    // through v
    foreach (int adj in adjList[v])
    {
      if (tc[s, adj] == 0) {
        dfsUtil(s, adj);
      }
    }
  }

  // Driver Code
  public static void Main(String[] args) {

    // Create a graph given
    // in the above diagram
    Graph g = new Graph(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);
    Console.WriteLine("Transitive closure " +
                      "matrix is");
    g.transitiveClosure();
  }
}

// This code is contributed by Rajput-Ji
```

Output:

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

Like 0

Previous

Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 Transitive closure of a graph
04, Dec 12

02 Check for transitive property in a given Undirected Graph
09, Feb 21

03 Check if a graph is strongly connected | Set 1 (Kosaraju using DFS)
02, Jun 13

05 Check if a given graph is Bipartite using DFS
03, Aug 18

06 Minimum number of edges between two vertices of a graph using DFS
29, Jan 19

07 Depth First Search or DFS for a Graph

1 (DFS of Unweighted and Undirected)
19, Jan 17

08 represented as adjacency list
22, Sep 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy Normal Medium Hard Expert

Improved By : csecec1702556, Rajput-Ji, easeit

Article Tags : Graph

Practice Tags : Graph

Improve Article Report Issue

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

Load Comments

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org