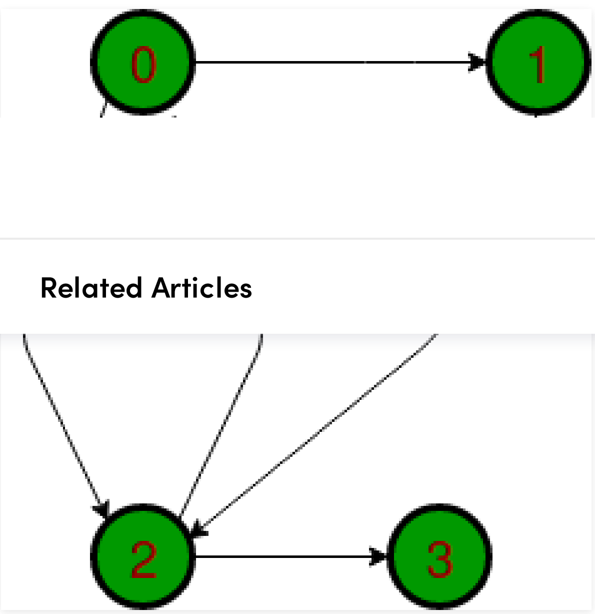
Transitive closure of a graph

Difficulty Level : Medium • Last Updated : 03 Oct, 2020

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

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

The graph is given in the form of adjacency matrix say 'graph[V][V]' where graph[i][j] is 1 if there is an edge from vertex i to vertex j or i is equal to j, otherwise graph[i][j] is 0.
Floyd Warshall Algorithm can be used, we can calculate the distance matrix dist[V][V] using Floyd Warshall, if dist[i][j] is infinite, then j is not reachable from I. Otherwise, j is reachable and the value of dist[i][j] will be less than V.
Instead of directly using Floyd Warshall, we can optimize it in terms of space and time, for this particular problem. Following are the optimizations:

1. Instead of an integer resultant matrix (dist[V][V] in floyd warshall), we can create a boolean reach-ability matrix reach[V][V] (we save space). The value reach[i][j] will be 1 if j is reachable from i, otherwise 0.
2. Instead of using arithmetic operations, we can use logical operations. For arithmetic operation '+', logical and '&&' is used, and for a min, logical or '||' is used. (We save time by a constant factor. Time complexity is the same though)

Below is the implementation of the above approach:

C++

```
// Program for transitive closure
// using Floyd Warshall Algorithm
#include<stdio.h>

// Number of vertices in the graph
#define V 4

// A function to print the solution matrix
void printSolution(int reach[][V]);

// Prints transitive closure of graph[][]
// using Floyd Warshall algorithm
void transitiveClosure(int graph[][V])
{
    /* reach[][] will be the output matrix
    // that will finally have the
       shortest distances between
```

```
    /* Initialize the solution matrix same
    as input graph matrix. Or
       we can say the initial values of
       shortest distances are based
       on shortest paths considering
       no intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            reach[i][j] = graph[i][j];

    /* Add all vertices one by one to the
    set of intermediate vertices.
      ---> Before start of a iteration,
           we have reachability values for
           all pairs of vertices such that
           the reachability values
           consider only the vertices in
           set {0, 1, 2, .. k-1} as
           intermediate vertices.
     ----> After the end of a iteration,
           vertex no. k is added to the
            set of intermediate vertices
            and the set becomes {0, 1, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as
        // source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as
            // destination for the
            // above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on a path
                // from i to j,
                // then make sure that the value
                // of reach[i][j] is 1
                reach[i][j] = reach[i][j] ||
                  (reach[i][k] && reach[k][j]);
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(reach);
}
```

```
    printf ("Following matrix is transitive");
    printf("closure of the given graph\n");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
              /* because "i==j means same vertex"
               and we can reach same vertex
               from same vertex. So, we print 1....
               and we have not considered this in
               Floyd Warshall Algo. so we need to
               make this true by ourself
               while printing transitive closure.*/
              if(i == j)
                printf("1 ");
              else
                printf ("%d ", reach[i][j]);
        }
        printf("\n");
    }
}

// Driver Code
int main()
{
    /* Let us create the following weighted graph
            10
       (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
       \|/         |
       (1)------->(2)
            3           */
    int graph[V][V] = { {1, 1, 0, 1},
                        {0, 1, 1, 0},
                        {0, 0, 1, 1},
                        {0, 0, 0, 1}
                      };

    // Print the solution
    transitiveClosure(graph);
    return 0;
}
```

Java

```java
import java.util.*;
import java.lang.*;
import java.io.*;

class GraphClosure
{
    final static int V = 4; //Number of vertices in a graph

    // Prints transitive closure of graph[][] using Floyd
    // Warshall algorithm
    void transitiveClosure(int graph[][])
    {
        /* reach[][] will be the output matrix that will finally
           have the shortest  distances between every pair of
           vertices */
        int reach[][] = new int[V][V];
        int  i, j, k;

        /* Initialize the solution matrix same as input graph
           matrix. Or  we can say the initial values of shortest
           distances are based  on shortest paths considering
           no intermediate vertex. */
        for (i = 0; i < V; i++)
            for (j = 0; j < V; j++)
                reach[i][j] = graph[i][j];

        /* Add all vertices one by one to the set of intermediate
           vertices.
          ---> Before start of a iteration, we have reachability
               values for all  pairs of vertices such that the
               reachability values consider only the vertices in
               set {0, 1, 2, .. k-1} as intermediate vertices.
          ----> After the end of a iteration, vertex no. k is
                added to the set of intermediate vertices and the
                set becomes {0, 1, 2, .. k} */
        for (k = 0; k < V; k++)
        {
            // Pick all vertices as source one by one
            for (i = 0; i < V; i++)
            {
                // Pick all vertices as destination for the
                // above picked source
                for (j = 0; j < V; j++)
                {
                    // If vertex k is on a path from i to j,
                    // then make sure that the value of reach[i][j] is 1
                    reach[i][j] = (reach[i][j]!=0) ||
```

```
        // Print the shortest distance matrix
        printSolution(reach);
    }

    /* A utility function to print solution */
    void printSolution(int reach[][])
    {
        System.out.println("Following matrix is transitive closure"+
                           " of the given graph");
        for (int i = 0; i < V; i++)
        {
            for (int j = 0; j < V; j++) {
                if ( i == j)
                  System.out.print("1 ");
                else
                  System.out.print(reach[i][j]+" ");
            }
            System.out.println();
        }
    }

    // Driver Code
    public static void main (String[] args)
    {
        /* Let us create the following weighted graph
           10
        (0)------->(3)
        |         /|\
      5 |          |
        |          | 1
        \|/        |
        (1)------->(2)
           3          */

        /* Let us create the following weighted graph

              10
         (0)------->(3)
           |         /|\
         5 |          |
           |          | 1
         \|/          |
         (1)------->(2)
            3          */
         int graph[][] = new int[][]{ {1, 1, 0, 1},
                                      {0, 1, 1, 0},
```

```
        // Print the solution
        GraphClosure g = new GraphClosure();
        g.transitiveClosure(graph);
    }
}
// This code is contributed by Aakash Hasija
```

Python

```
# Python program for transitive closure using Floyd Warshall Algorithm
#Complexity : O(V^3)

from collections import defaultdict

#Class to represent a graph
class Graph:

    def __init__(self, vertices):
        self.V = vertices

    # A utility function to print the solution
    def printSolution(self, reach):
        print ("Following matrix transitive closure of the given graph ")
        for i in range(self.V):
            for j in range(self.V):
                if (i == j):
                  print "%7d\t" % (1),
                else:
                  print "%7d\t" %(reach[i][j]),
            print ""


    # Prints transitive closure of graph[][] using Floyd Warshall algorithm
    def transitiveClosure(self,graph):
        '''reach[][] will be the output matrix that will finally
        have reachability values.
        Initialize the solution matrix same as input graph matrix'''
        reach =[i[:] for i in graph]
        '''Add all vertices one by one to the set of intermediate
        vertices.
         ---> Before start of a iteration, we have reachability value
         for all pairs of vertices such that the reachability values
          consider only the vertices in set
        {0, 1, 2, .. k-1} as intermediate vertices.
          ----> After the end of an iteration, vertex no. k is
```

```
            # Pick all vertices as source one by one
            for i in range(self.V):

                # Pick all vertices as destination for the
                # above picked source
                for j in range(self.V):

                    # If vertex k is on a path from i to j,
                       # then make sure that the value of reach[i][j] is 1
                    reach[i][j] = reach[i][j] or (reach[i][k] and reach[k][j])

        self.printSolution(reach)

g= Graph(4)

graph = [[1, 1, 0, 1],
         [0, 1, 1, 0],
         [0, 0, 1, 1],
         [0, 0, 0, 1]]

#Print the solution
g.transitiveClosure(graph)

#This code is contributed by Neelam Yadav
```

C#

```
// C# Program for transitive closure
// using Floyd Warshall Algorithm
using System;

class GFG
{
    static int V = 4; // Number of vertices in a graph

    // Prints transitive closure of graph[,]
    // using Floyd Warshall algorithm
    void transitiveClosure(int [,]graph)
    {
        /* reach[,] will be the output matrix that
        will finally have the shortest distances
        between every pair of vertices */
        int [,]reach = new int[V, V];
        int i, j, k;
```

```
    intermediate vertex. */
    for (i = 0; i < V; i++)
        for (j = 0; j < V; j++)
            reach[i, j] = graph[i, j];

    /* Add all vertices one by one to the
    set of intermediate vertices.
    ---> Before start of a iteration, we have
        reachability values for all pairs of
        vertices such that the reachability
        values consider only the vertices in
        set {0, 1, 2, .. k-1} as intermediate vertices.
    ---> After the end of a iteration, vertex no.
         k is added to the set of intermediate
         vertices and the set becomes {0, 1, 2, .. k} */
    for (k = 0; k < V; k++)
    {
        // Pick all vertices as source one by one
        for (i = 0; i < V; i++)
        {
            // Pick all vertices as destination
            // for the above picked source
            for (j = 0; j < V; j++)
            {
                // If vertex k is on a path from i to j,
                // then make sure that the value of
                // reach[i,j] is 1
                reach[i, j] = (reach[i, j] != 0) ||
                             ((reach[i, k] != 0) &&
                              (reach[k, j] != 0)) ? 1 : 0;
            }
        }
    }

    // Print the shortest distance matrix
    printSolution(reach);
}

/* A utility function to print solution */
void printSolution(int [,]reach)
{
    Console.WriteLine("Following matrix is transitive" +
                      " closure of the given graph");
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++){
            if (i == j)
```

```
            Console.WriteLine();
        }
    }

    // Driver Code
    public static void Main (String[] args)
    {
        /* Let us create the following weighted graph
        10
        (0)------->(3)
        |     /|\
    5 |     |
        |     | 1
        \|/ |
        (1)------->(2)
        3     */

        /* Let us create the following weighted graph

            10
        (0)------->(3)
        |     /|\
        5 |     |
        |     | 1
        \|/     |
        (1)------->(2)
            3     */
        int [,]graph = new int[,]{{1, 1, 0, 1},
                                  {0, 1, 1, 0},
                                  {0, 0, 1, 1},
                                  {0, 0, 0, 1}};

        // Print the solution
        GFG g = new GFG();
        g.transitiveClosure(graph);
    }
}

// This code is contributed by 29AjayKumar
```

Output

```
Following matrix is transitiveclosure of the given graph
1 1 1 1
0 1 1 1
```

Time Complexity: $O(V^3)$ where V is number of vertices in the given graph.
See below post for a $O(V^2)$ solution.
Transitive Closure of a Graph using DFS

References:
Introduction to Algorithms by Clifford Stein, Thomas H. Cormen, Charles E. Leiserson, Ronald L.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

RECOMMENDED ARTICLES

03 Graph implementation using STL for competitive programming | Set 2 (Weighted graph)
20, Jan 17

04 Detect cycle in the graph using degrees of nodes of graph
03, Apr 19

07 Convert undirected connected graph to strongly connected directed graph
21, May 20

08 Java Program to Find Independent Sets in a Graph using Graph Coloring
27, Feb 21

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : 29AjayKumar, AMBERSINGHAL

Article Tags : Graph

Practice Tags : Graph

Improve Article | Report Issue

Load Comments

GeeksforGeeks

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

feedback@geeksforgeeks.org