Tallinn Technological University

School of Information Technologies

Artem Fedorchenko 223663IVSB

Transitive closure of the Graph

Homework №6 Report

Abstract

The following work's contents are gathered from various sources which were used by the author for research purposes. The report examines the standard problem in algorithms and data structures such as transitive closure of the graph. The paper includes the definition of the problem including mathematical background of it, various solutions that have been proposed, and their efficiency rates. Additionally, the report contains the actual solution which the author has decided to opt for, its implementation, together with the pseudo-code implementation, multiple test cases with the detailed explanation of the obtained result depicted on the drawing.

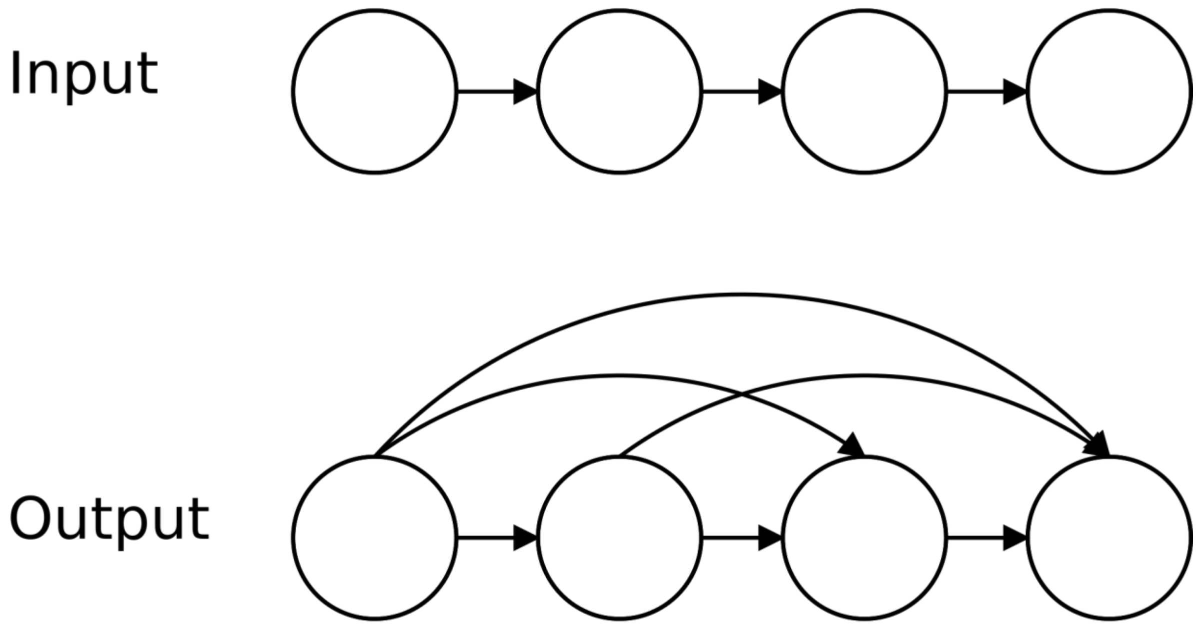
List of terms and definitions

DFS	Depth-first search
BFS	Breadth-first search
Node	Structural unit of a graph (Data Structure)
Vertex	An equivalent to the node
Arc	Structural unit of the graph which connects nodes
Edge	An equivalent to the Arc
DAG	Directed Acyclic Graph
TC	Transitive Closure

Transitive Closure

In mathematics, the transitive closure of a relation on a set is a new relation that includes all the pairs of elements that are related to each other through the original relation or a sequence of intermediate relations. In a more formal way transitive closure can be expressed as following: For any pair of elements x, y in X $(x, y) \in R+$ if and only if there exists a non-infinite sequence of elements (x1,x2, ..xn) in X such that x = x1, y = xn and $(xi, xi+1) \in R$ for all i = 1, 2, ..., n-1. In computer science or to be more specific in graph theory transitive closure defines the data structure that answers the question of reachability from one node to another. More formally, we define the transitive closure (TC) problem as follows. Given a directed graph G = (V, E) with $|V| = n$, $|E| = m$, we aim to output an $n \times n$ matrix where C (u, v) != 0 if v is reachable from u.

The notion of transitive closure in graph theory is expressed by the following picture:



Existing solutions to the problem

One of the most popular solution is to use DFS algorithm, its time complexity is O(n^3), where n is number of nodes.

The pseudo-code for this solution can be described as following:

```
function transitiveClosureDFS(graph):
    n = number of nodes in graph
    closure = 2D boolean array of size n x n
    for i from 0 to n-1:
        dfs(i, i, closure, graph)
    return closure

function dfs(start, current, closure, graph):
    closure[start][current] = True
    for neighbor in graph[current]:
        if not closure[start][neighbor]:
            dfs(start, neighbor, closure, graph)
```

Additionally, in terms of efficiency there is a better algorithm. In 1971, Fisher and Meyer proposed an algorithm which runs in O(n^ω), where ω = 2.38.

The pseudo-code for this algorithm:

```
function transitiveClosureFM(graph):
    n = number of nodes in graph
    A = copy of the adjacency matrix of graph
    for k from 1 to n:
        for i from 1 to n:
            for j from 1 to n:
                A[i][j] = A[i][j] or (A[i][k] and A[k][j])
    return A
```

My solution (Core Algorithm)

The core algorithm of my solution is the Floyd-Warshall algorithm. The Floyd-Warshall algorithm is shortest path algorithm. In fact, there are many shortest path algorithms such Dijkstra or Bellman-Ford algorithm, however, the Floyd-Warshall algorithm computes the shortest path for every pair (i, j) in the graph whereas two others compute it only for one vertex.

The pseudo-code for the Floyd-Warshall algorithm:

```
Create a |V| x |V|, M,  matrix that represents arcs between all the vertices (Adjacency matrix)
for each cell(i,j) in M:
      if i==j:
         M[i][j] = 0
      if (i,j) is an edge in M:
         M[i][j] = weight(i,j)
      else:
         M[i][j] = infinity
for k from 1 to |V|:
   for j from 1 to |V|:
      for i from 1 to |V|:
         if M[i][j] > M[i][k] + M[k][j]:
               M[i][j] = M[i][k] + M[k][j]
```

From the provided pseudo-code implementation the core algorithm of my solution will run in O(|V|^3) time. This is because of the three nested loops it has.

My solution (The algorithm to solve the task)

Since my method must return a graph object according to the predefined representation, the actual algorithm will have a few more additional steps to solve the problem correctly.

My algorithm can be described as following:

```
Create a Graph(), TC,  object with the same vertices
Create a HashMap(), vertices,  to store the Vertex IDs
Initialize the Vertex() object,V,  that indicates the first outgoing Arc from the Vertex

while (V!=null){
   vertices.put(V.id, TC.createNewVertex(V.id));
   V = V.next;
}

Initialize adjacency matrix, adjMatrix, for the given Graph()
Initialize the variable, n , that indicates the lenght of the matrix
Initialize transitive closure matrix, transitiveClosure[n][n],  of lenght n

for i from 1 to n:
   for j from 1 to n:
      if i == j or transitiveClosure[i][j] > 0:
         transitiveClosure[i][j] = 1 // path exists

Perform Floyd-Warshall algorithm for the transitiveClosure matrix

for i from 0 to n:
   for j from 0 to n:
      if transitiveClosure[i][j]:
         createVertex(i) and createVertex(j)
```

User guidelines

The method which computes transitive closure heavily depends on the set of auxiliary methods. Particularly, I used HashMap's method put(), thus, the java.util.HashMap must be imported to take the advantage of the method. Additionally, not including the predefined set of methods and constructors, I implemented the method getVertex(), which takes as an input integer which indicates the id and returns the Vertex() for the specified index.

The implementation of method getVertex():

```
Returns the vertex at the specified index.
Params: index – of the Vertex we need to get
Returns: Vertex object for the given index

2 usages   Artemmmm13
public Vertex getVertex(int index) {
    Vertex v = first;
    for (int i = 0; i < index; i++) {
        v = v.next;
    }
    return v;
}
```

Full solution (code) [1]

```java
import java.util.HashMap;
import java.util.Map;

Container class to different classes, that makes the whole set of classes one class formally.
Jaanus Poeial +2 *
public class GraphTask {

   /** Main method. */
   Jaanus Poeial
   public static void main (String[] args) {
      GraphTask a = new GraphTask();
      a.run();
   }

   /** Actual main method to run examples and everything. */
   1 usage   Artemmmm13 +1 *
   public void run() {
      // average : 118002(ms) for graph with 2000+ vertices
      Graph g = new Graph( s: "G");
      Vertex v1 = g.createVertex( vid: "1");
      Vertex v2 = g.createVertex( vid: "2");
      Vertex v3 = g.createVertex( vid: "3");
      Vertex v4 = g.createVertex( vid: "4");
      g.createArc( aid: "a1", v1, v2);
      System.out.println(g);
      Graph tc = g.transitiveClosure();
      System.out.println(tc);

   }
```

Full solution (code) [2]

```
28 usages   Jaanus Poeial +2
class Vertex {

    6 usages
    private final String id;
    8 usages
    private Vertex next;
    5 usages
    private Arc first;
    3 usages
    private int info = 0;
    // You can add more fields, if needed

    1 usage   Jaanus Poeial
    Vertex (String s, Vertex v, Arc e) {
        id = s;
        next = v;
        first = e;
    }

    1 usage   Jaanus Poeial
    Vertex (String s) { this (s, v: null, e: null); }

    13 usages   Artemmmm13
    @Override
    public String toString() {return id;}
}
```

Full solution (code) [3]

```
Arc represents one arrow in the graph. Two-directional edges are represented by two Arc objects
(for both directions).
9 usages  Jaanus Poeial +2
class Arc {

   2 usages
   private final String id;
   4 usages
   private Vertex target;
   4 usages
   private Arc next;
   no usages
   private final int info = 0;
   1 usage  Jaanus Poeial
   Arc (String s, Vertex v, Arc a) {
      id = s;
      target = v;
      next = a;
   }

   1 usage  Artemmmm13 +1
   Arc (String s) {
      this (s, v: null, a: null);}

   13 usages  Artemmmm13
   @Override
   public String toString() {return id;}
}
```

Full solution (code) [4]

```
6 usages  Jaanus Poeial +2 *
class Graph {

    3 usages
    private final String id;
    10 usages
    private Vertex first;
    4 usages
    private int info = 0;

    no usages
    private boolean[][] tc; // additional field for transitive closure matrix

    1 usage  Jaanus Poeial
    Graph (String s, Vertex v) {
        id = s;
        first = v;
    }
    2 usages  Artemmmm13
    Graph (String s) {this (s, v: null);}
```

Full solution (code) [5]

```
13 usages  Jaanus Poeial +1
@Override
public String toString() {
    String nl = System.getProperty ("line.separator");
    StringBuffer sb = new StringBuffer (nl);
    sb.append (id);
    sb.append (nl);
    Vertex v = first;
    while (v != null) {
        sb.append (v);
        sb.append (" -->");
        Arc a = v.first;
        while (a != null) {
            sb.append (" ");
            sb.append (a);
            sb.append (" (");
            sb.append (v);
            sb.append ("->");
            sb.append (a.target.toString());
            sb.append (")");
            a = a.next;
        }
        sb.append (nl);
        v = v.next;
    }
    return sb.toString();
}

6 usages  Jaanus Poeial
public Vertex createVertex (String vid) {
    Vertex res = new Vertex (vid);
    res.next = first;
    first = res;
    return res;
}
```

Full solution (code) [6]

```
6 usages  Jaanus Poeial
public Arc createArc (String aid, Vertex from, Vertex to) {
   Arc res = new Arc (aid);
   res.next = from.first;
   from.first = res;
   res.target = to;
   return res;
}

```

Create a connected undirected random tree with n vertices. Each new vertex is connected to some random existing vertex.

Params: n – number of vertices added to this graph

```
1 usage  Jaanus Poeial +1
public void createRandomTree (int n) {
   if (n <= 0)
      return;
   Vertex[] varray = new Vertex [n];
   for (int i = 0; i < n; i++) {
      varray [i] = createVertex ( vid: "v" + (n - i));
      if (i > 0) {
         int vnr = (int)(Math.random()*i);
         createArc ( aid: "a" + varray [vnr].toString() + "_"
                 + varray [i].toString(), varray [vnr], varray [i]);
         createArc ( aid: "a" + varray [i].toString() + "_"
                 + varray [vnr].toString(), varray [i], varray [vnr]);
      } else {}
   }
}
```

Full solution (code) [7]

Create an adjacency matrix of this graph. Side effect: corrupts info fields in the graph
Returns: adjacency matrix

2 usages · Jaanus Poeial

```java
public int[][] createAdjMatrix() {
   info = 0;
   Vertex v = first;
   while (v != null) {
      v.info = info++;
      v = v.next;
   }
   int[][] res = new int [info][info];
   v = first;
   while (v != null) {
      int i = v.info;
      Arc a = v.first;
      while (a != null) {
         int j = a.target.info;
         res [i][j]++;
         a = a.next;
      }
      v = v.next;
   }
   return res;
}
```

Full solution (code) [8]

Create a connected simple (undirected, no loops, no multiple arcs) random graph with n vertices and m edges.

Params: n – number of vertices
m – number of edges

no usages Jaanus Poeial +1

```java
public void createRandomSimpleGraph (int n, int m) {
   if (n <= 0)
      return;
   if (n > 2500)
      throw new IllegalArgumentException ("Too many vertices: " + n);
   if (m < n-1 || m > n*(n-1)/2)
      throw new IllegalArgumentException
            ("Impossible number of edges: " + m);
   first = null;
   createRandomTree (n);       // n-1 edges created here
   Vertex[] vert = new Vertex [n];
   Vertex v = first;
   int c = 0;
   while (v != null) {
      vert[c++] = v;
      v = v.next;
   }
   int[][] connected = createAdjMatrix();
   int edgeCount = m - n + 1;  // remaining edges
   while (edgeCount > 0) {
      int i = (int)(Math.random()*n);  // random source
      int j = (int)(Math.random()*n);  // random target
      if (i==j)
         continue;  // no loops
      if (connected [i][j] != 0 || connected [j][i] != 0)
         continue;  // no multiple edges
      Vertex vi = vert [i];
      Vertex vj = vert [j];
      createArc ( aid: "a" + vi.toString() + "_" + vj.toString(), vi, vj);
      connected [i][j] = 1;
      createArc ( aid: "a" + vj + "_" + vi, vj, vi);
      connected [j][i] = 1;
      edgeCount--;  // a new edge happily created
   }
}
```

Full solution (code) [9]

```java
1 usage   Artemmmm13 +2 *
public Graph transitiveClosure() {
    // Create a new graph with the same vertices
    Graph tc = new Graph(id);
    // Create a map to store the vertex IDs
    Map<String, Vertex> vertices = new HashMap<>();
    Vertex v = first;
    while (v != null) {
        vertices.put(v.id, tc.createVertex(v.id));
        v = v.next;
    }
    // Create the transitive closure matrix using the adjacency matrix
    int[][] adjMatrix = createAdjMatrix();
    int n = adjMatrix.length;
    int[][] transitiveClosure = new int[n][n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (i == j || adjMatrix[i][j] > 0) {
                transitiveClosure[i][j] = 1;
            }
        }
    }
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                transitiveClosure[i][j] |= (transitiveClosure[i][k] & transitiveClosure[k][j]);
            }
        }
    }

    // Create the arcs in the transitive closure graph
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (transitiveClosure[i][j] == 1) {
                tc.createArc( aid: "", vertices.get(getVertex(i).id), vertices.get(getVertex(j).id));
            }
        }
    }
    return tc;
}
```

Full solution (code) [10]

```
Returns the vertex at the specified index.
Params: index – of the Vertex we need to get
Returns: Vertex object for the given index
2 usages  Artemmmm13
        public Vertex getVertex(int index) {
            Vertex v = first;
            for (int i = 0; i < index; i++) {
                v = v.next;
            }
            return v;
        }
    }
}
```

Testing Plan

To make sure that the method works decently – test which “touch” basic and corner cases are required. In addition, the test, including a Graph with a vast number of vertices, is required.

My testing plan included such cases:

1. Graph with 3 vertices and 2 edges.
2. Graph with 5 vertices and 6 edges.
3. Graph with 4 vertices and 2 edges.
4. Graph with 4 vertices and 0 edges.
5. Graph with 2001 vertices and 2002 edges.

Test results

Case 1:

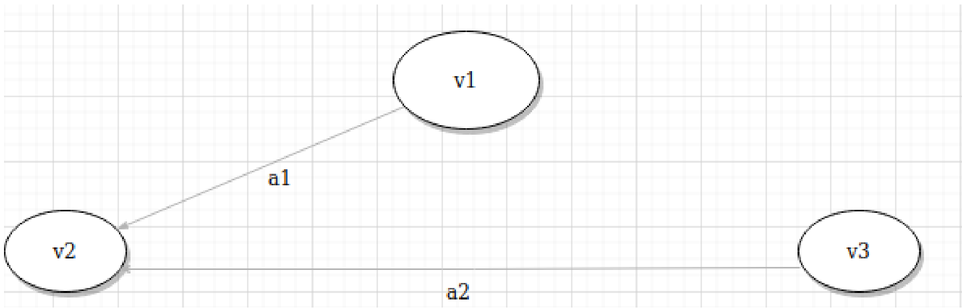
Input Graph:

3 -->

2 --> a2 (2->3)

1 --> a1 (1->2)

```
Graph g = new Graph( s: "G");
Vertex v1 = g.createVertex( vid: "1");
Vertex v2 = g.createVertex( vid: "2");
Vertex v3 = g.createVertex( vid: "3");
g.createArc( aid: "a1", v1, v2);
g.createArc( aid: "a2", v2, v3);
```

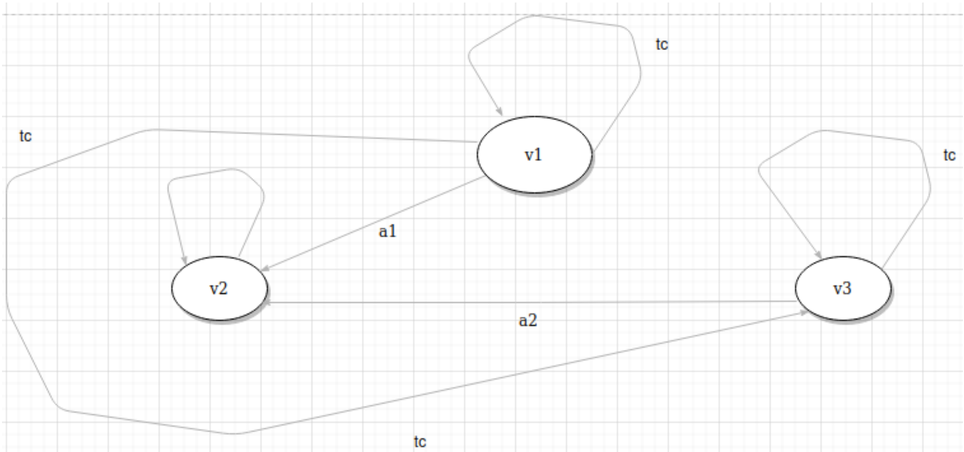


Output Graph:

1 --> (1->1) (1->2) (1->3)

2 --> (2->2) (2->3)

3 --> (3->3)



Case 2:

Input Graph:

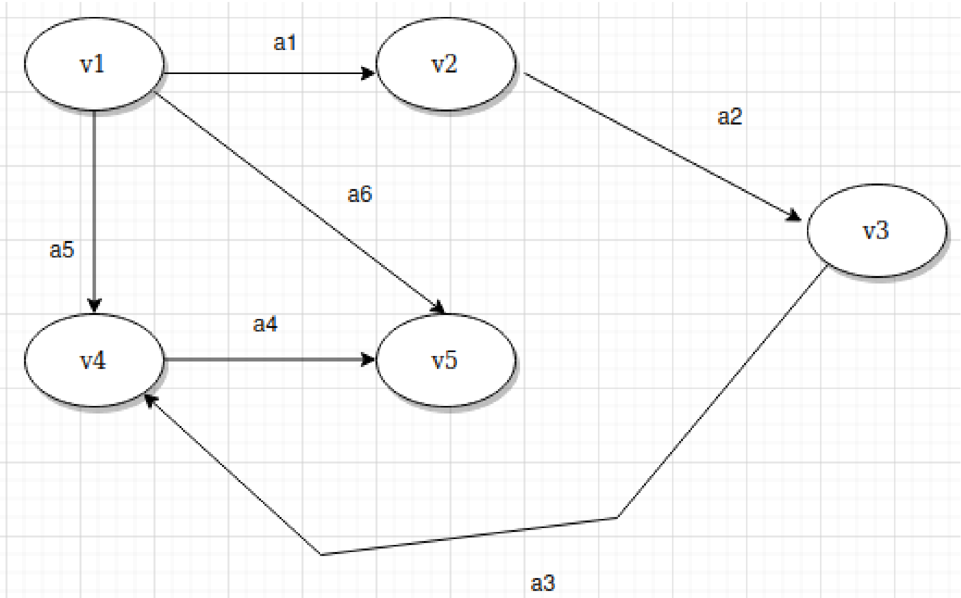
5 -->

4 --> a4 (4->5)

3 --> a3 (3->4)

2 --> a2 (2->3)

```java
Graph g = new Graph( s: "G");
Vertex v1 = g.createVertex( vid: "1");
Vertex v2 = g.createVertex( vid: "2");
Vertex v3 = g.createVertex( vid: "3");
Vertex v4 = g.createVertex( vid: "4");
Vertex v5 = g.createVertex( vid: "5");
g.createArc( aid: "a1", v1, v2);
g.createArc( aid: "a2", v2, v3);
g.createArc( aid: "a3", v3, v4);
g.createArc( aid: "a4", v4, v5);
g.createArc( aid: "a5", v1, v5);
g.createArc( aid: "a6", v1, v4);
```



Output Graph:

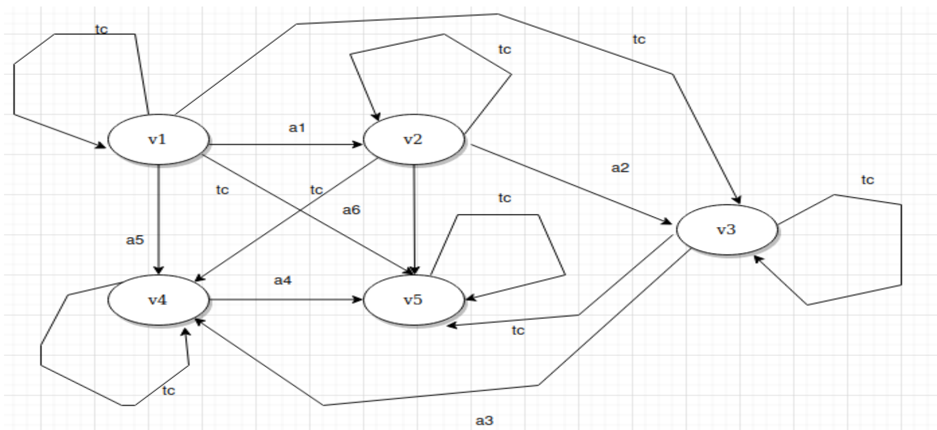
1 --> (1->1) (1->2) (1->3) (1->4) (1->5)

2 --> (2->2) (2->3) (2->4) (2->5)

3 --> (3->3) (3->4) (3->5)

4 --> (4->4) (4->5)

5 -->(5->5)



Case 3:

Input Graph:

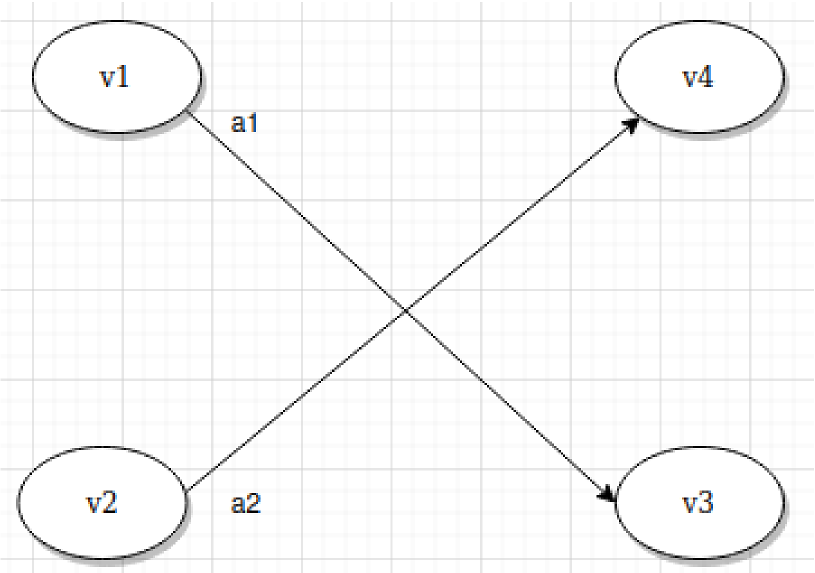
4 -->

3 -->

2 --> a2 (2->4)

1 --> a1 (1->3)

```
Graph g = new Graph( s: "G");
Vertex v1 = g.createVertex( vid: "1");
Vertex v2 = g.createVertex( vid: "2");
Vertex v3 = g.createVertex( vid: "3");
Vertex v4 = g.createVertex( vid: "4");
g.createArc( aid: "a1", v1, v3);
g.createArc( aid: "a2", v2, v4);
```



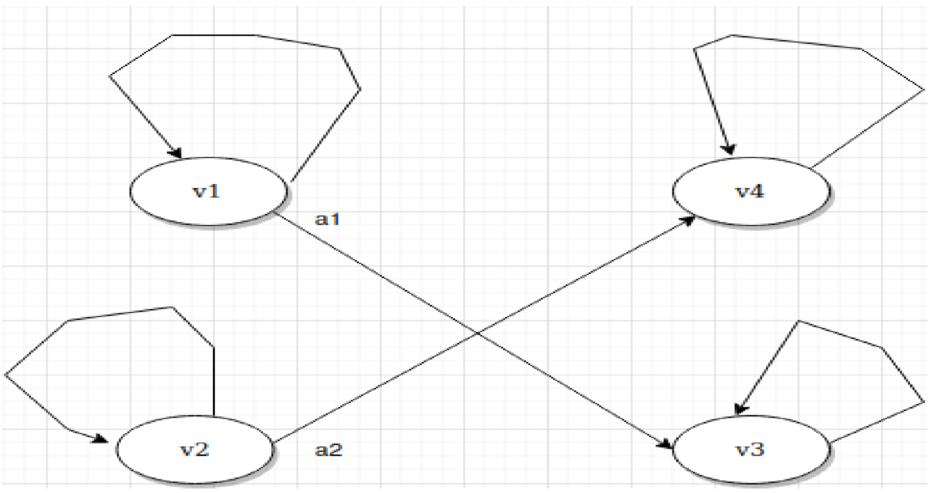
Output Graph:

1 --> (1->1) (1->3)

2 --> (2->2) (2->4)

3 --> (3->3)

4 --> (4->4)



Case 4:

Input Graph:

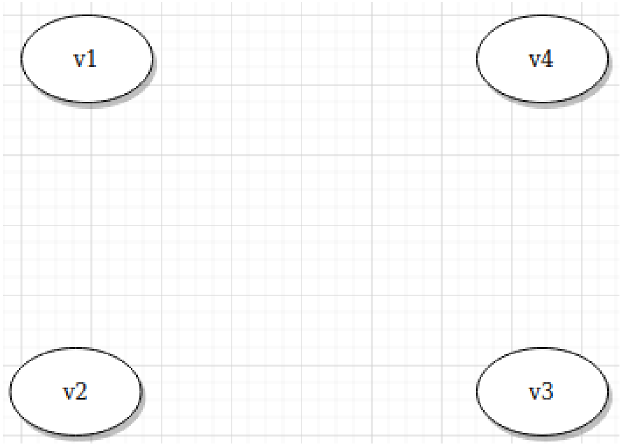
4 –>

3 –>

2 –>

1 –>

```
Graph g = new Graph( s: "G");
Vertex v1 = g.createVertex( vid: "1");
Vertex v2 = g.createVertex( vid: "2");
Vertex v3 = g.createVertex( vid: "3");
Vertex v4 = g.createVertex( vid: "4");
```



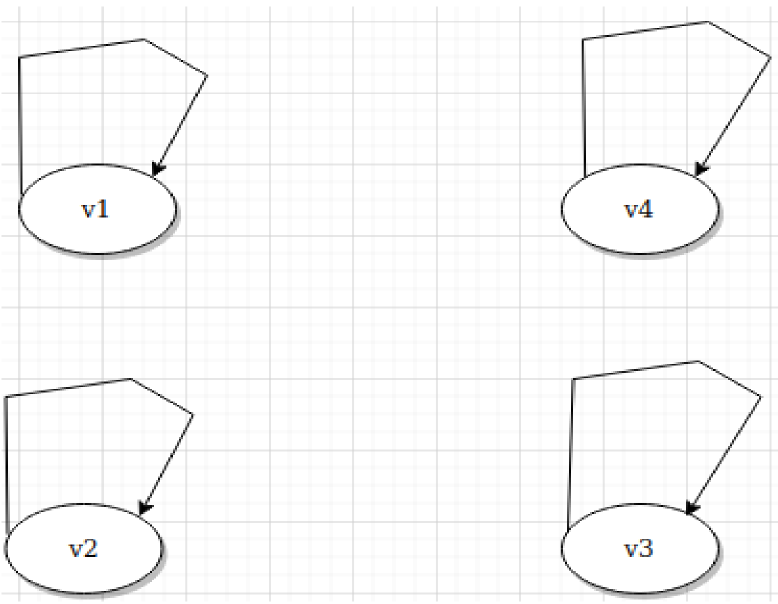
Output Graph:

1 –> (1->1)

2 –> (2->2)

3 –> (3->3)

4 –> (4->4)



Case 5:

As I have already mentioned this will include a graph with 2001 vertices and 2002 edges, thus, to show the graphical representation of it would be impossible and would not make much sense. The purpose of this test is to measure execution time and estimate the efficiency of my solution.

Here is the code for the graph creation and for measuring the execution time:

```
Graph g = new Graph( s: "G");
g.createRandomSimpleGraph( n: 2001,  m: 2002);
long start = System.nanoTime();
Graph tc = g.transitiveClosure();
long finish = System.nanoTime();
long delta = finish - start;
System.out.printf("%34s%11d%n", "Execution time (ms): ", delta / 1000000);
```

The result of the first execution is 46503 (ms).

The result of the second execution is 62789 (ms).

The result of the third execution is 44795 (ms).

The result of the fourth execution is 59655 (ms).

Therefore, the average result on my machine is **54185.5 (ms).**

References

[1]

M. Kim, “CS 267 Lecture 11 Dynamic Transitive Closure Scribe,” 2016. Accessed: Apr. 20, 2023. [Online]. Available: https://theory.stanford.edu/~virgi/cs267/lecture12.pdf

[2]

Wikipedia Contributors, “Transitive closure,” *Wikipedia*, Nov. 17, 2019.Accessed Apr.20 2023[Online]. Available:https://en.wikipedia.org/wiki/Transitive_closure

[3]

“Transitive closure of a graph,” *GeeksforGeeks*, Dec. 04, 2012. https://www.geeksforgeeks.org/transitive-closure-of-a-graph/ (Accessed Apr. 22, 2023) [Online].

[4]

M. T.Goodrich and R. Tamassia, “*Algorithms and Data Structures in Java,”* Fourth edition. 2002. Accessed: Apr.20 2023[Online].

[5]

T. H. Cormen, Charles Eric Leiserson, R. L. Rivest, C. Stein, and E. Al, *Introduction to algorithms*. MIT Press, 2009. Accessed: Apr.20 2023[Online].