1 Overview

In this final lab, you will implement two variations of an algorithm that computes the transitive closure of a relation, represented by an adjacency matrix (a 2 dimensional array of booleans). Lastly, you will experiment with both algorithms and compare their time complexities.

2 Transitive Closure Refresher

Let $A$ be a finite set of Twitter users $= \{Abby, Barry, Carla, Diego, Ellen\}$. Let $R \subseteq A \times A$ be a binary relation on $A$, representing users in $A$ who follow other users in $A$.

In other words, $R$ is a set of ordered pairs, whereas $(j, k)$ implies that user $j$ follows user $k$. Keep in mind, follows on Twitter are non-communicative; if $j$ follows $k$, that doesn’t necessarily mean that $k$ follows $j$.

Let’s say $R := \{(Abby, Barry), (Barry, Ellen), (Diego, Carla), (Diego, Ellen), (Ellen, Carla)\}$, visualized below:

We could also depict $R$ as a boolean matrix (i.e. our 2D array) where if $arr[i][j]$ is true, then $i$ follows $j$ on Twitter:
In order for our relation \( R \) to be transitive, for all \( i, j, \) and \( k \) in \( A \), if \( (i, j) \) exists in \( R \), and \( (j, k) \) exists in \( R \), then \( (i, k) \) must also exist in \( R \). In the context of our example, if Twitter user \( i \) follows user \( j \), and \( j \) follows user \( k \), then \( i \) must also follow \( k \) in order for the relation to be transitive.

Another way to think about this is in the context of the above graph: if any node on the graph can be reached from another node with a distance of \( > 1 \), it must also be able to be reached with a distance of \( 1 \). \( R \) is not transitive because, Ellen can be reached from Abby with a distance of \( 2 \) (traversing Abby \( \rightarrow \) Barry \( \rightarrow \) Ellen), but cannot be reached with a distance of \( 1 \) (there is no direct connection from Abby to Ellen).

The Transitive Closure is a relation \( R^+ \supseteq R \) that contains the minimum number of additional pairs required to make the relation transitive. Thus, the transitive closure of \( R \) would be:

\[
R^+ := \{(Abby, Barry),(Abby, Carla),(Abby, Ellen),(Barry, Ellen),(Barry, Carla),(Diego, Carla),(Diego, Ellen),(Ellen, Carla)\}
\]

The graph and boolean matrix of \( R^+ \) would be as follows:

![Graph](image)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

3. **Your Task**

Your goal for this lab is to write two variations of a function which generates the transitive closure of a given relation. Relations will be represented with a two dimensional boolean array similar to the matrices shown in the examples above. First, spend some time familiarizing yourself with the provided code and looking over the main method in `Lab09Tester.java`.

Once you have implemented your transitive closure algorithm, you are required to create and run two test relations, and analyze the time complexities of your two algorithms.

3.1 **Implementation**

There are four functions you need to implement, detailed on the following page:
1. **makeUnion**: returns a new relation matrix $T$ that is the union of the argument relations $R$ and $S$. As a refresher, this means if pair $(i, j)$ exists in $R$ and/or $S$, then it must exist in $T$.

2. **makeComposition**: returns a new relation matrix $T$ that is the composition of the argument relations $R$ and $S$ (denoted as $R \circ S$). As a refresher, that means $T$ contains all pairs $(i, k)$ such that there is a $j$ in $A$ for which $(i, j)$ exists in $S$ and $(j, k)$ exists in $R$.

3. **makeTCSimple**: lastly, you will implement two different functions which compute the transitive closure of a relation. Both functions return a new relation matrix $R^+$ that is the transitive closure of the argument relation $R$, as described above. Additionally, neither function should modify the argument relation.

Recall that, given the relation $R$ (ex: collection of "follow" pairs) on the finite set $A$ (ex: Twitter users), $R^+ = R \cup R^2 \cup R^3 \cup ... \cup R^n$ where $n$ equals the cardinality of $A$. Also recall that we can calculate $R^k$ by composing $R$ with itself $k$ times, i.e.: $R^k := R \circ R \circ ... \circ R$.

Given this logic, implement makeTCSimple following the pseudo-code representation of calculating $R^+$ described below:

(a) Initialize $T := R$
(b) Update $T$ to be union of itself with the composition of $R$. In other words, $T := T \cup (R \circ T)$
(c) Repeat Step b until one of the following occurs:
   - Step b has been repeated $(n - 1)$ times, where $n$ = the number of vertices in $R$
   - or, $T$ is unchanged after an execution of Step b

4. **makeTCWarshalls**: for this second transitive closure function, implement it using Warshall’s algorithm (a more optimized solution than the previous function).

Review the following pseudocode and analysis of Warshall’s algorithm:

```java
// Warshall's Algorithm
// Given a relation A consisting of ordered pairs [0...n, 0...n]
R begins with all relations of A
for k = 0 to n do:
    for i = 0 to n do:
        for j = 0 to n do:
            R[i,j] exists if in R: [i,j] exists OR ([i,k] and [k,j] exists)
return R
```

4 Relation Tests and Analysis

In your `Lab09Tester.java`, you must document the following:

- **Minimally**, you must create, test, and document the following two test cases to run against your two transitive closure functions:
  1. a relation which contains a chain of seven or more edges, example: <a, b>, <b, c>, <c, d> ...and so on. Each node should have 0 or 1 edges, and 0 or 1 other nodes pointing to it. If abstracted as a graph, you should be able to visualize this relation as a straight line.
  2. a relation with five or more nodes with no loop in the graph (meaning no <k, k> for any node k in its ordered pairs), but has at least one such loop (<k, k>) in its transitive closure.

- In comments at the bottom of your `main`, analyze the time complexities and describe asymptotic bounds (best and worst case behaviors) of both makeTCSimple and makeTCWarshalls along with a very brief explanation of your answers.
5  Pre-Lab Questions

After reading this document and provided code, answer the following before we meet (we will discuss in-lab):

1. Suppose we have relations, \( R \) and \( S \), each stored as a two-dimensional boolean arrays Let \( n1 \) and \( n2 \) represent the lengths of the first and second dimensions of \( R \) respectively, and \( n3 \) and \( n4 \) the dimensions of \( S \).

Are the following statements true or false per the functions you will implement in TransitiveClosures.java?

(a) To use the makeUnion(...) function to create \( R \cup S \), \( n1 \) == \( n3 \) and \( n2 \) == \( n4 \) must be true.
(b) To use the makeUnion(...) function to create \( R \cup S \), \( n1 \) == \( n2 \) and \( n3 \) == \( n4 \) must be true.
(c) To use the makeComposition(...) function to create \( R \circ S \), \( n1 \) == \( n3 \) must be true.
(d) To use the makeComposition(...) function to create \( R \circ S \), \( n2 \) == \( n3 \) must be true.

2. Is the relation on the following page transitive? If not, what edges are missing?

![Graph with edges A to B, A to C, B to A, B to D, C to A, C to D, D to A, D to B]

3. Suppose we have a non-transitive relation \( R \) and its transitive closure \( R^+ \), represented by a two-dimensional boolean arrays \( r1 \) and \( r2 \), respectively.

Is it possible that there are one or more indices \( i \) such that \( r1[i][i] \) == false and \( r2[i][i] \) == true? If so, can you come up with an example relation?

6  Submission

See the top of this document for the lab’s due date and time. When submitting your code, include only the files listed below:

- TransitiveClosures.java
- Lab09Tester.java