

Assignment No: 5  
Date: 20/04/2024

## ADVANCED DATA STRUCTURES

### ASSIGNMENT - 5

Graphs: Comparison between Tarjan's and Kosaraju's Algorithm

| Assignment Type of Submission:   |        |  |  |
|--|--------|--|--|
| <b>Group</b>   | Yes/No | <b>List all group members' details:</b>                            | <b>% Contribution Assignment Workload</b>  |
| <b>COMP47500<br/>Advanced Data Structures<br/><br/>Group Assignment- 5</b> | Yes    | Student Name: Avantika Sivakumar<br>Student ID: 23205789           | 33.33%<br>(Code implementation, Result evaluation, video explanation)            |
|  |        | Student Name: Ria Rahul Asgekar<br>Student ID: 23203987            | 33.33%<br>(Code Implementation, material gathered for report, video explanation) |
|  |        | Student Name: Jeevan Raju Ravi Govind Raju<br>Student ID: 23207314 | 33.33%<br>(Code implementation, material gathered for report, video explanation) |

## **1. Problem Domain Description:**

This assignment aims to compare the efficiency of Tarjan's and Kosaraju's algorithms for finding strongly connected components (SCCs) in directed graphs. The analysis performed here focuses on evaluating the time complexities of these algorithms' key operations, specifically the depth-first search (DFS) traversal and stack manipulation for SCC identification.

### **1. Tarjan's Algorithm:**

#### **1.1. Overview:**

Tarjan's algorithm is used to find SCCs in directed graphs. It uses DFS traversal and utilises a stack to identify SCCs efficiently.[References: 1]

#### **1.2. DFS Traversal:**

Tarjan's algorithm performs a depth-first search traversal of the graph, assigning discovery and low values to each vertex. This traversal ensures that all vertices are visited and that the algorithm can identify strongly connected components.

#### **1.3. Stack Manipulation:**

During the DFS traversal, Tarjan's algorithm maintains a stack to keep track of the vertices in the current SCC. It pushes(adds) vertices onto the stack as they are visited and removes them once their SCC is identified.

### **2. Kosaraju's Algorithm:**

#### **2.1. Overview:**

Kosaraju's algorithm is another method for finding SCCs in directed graphs. It involves two DFS traversals of the graph and utilises the concept of graph transpose.[References: 1]

#### **2.2. First DFS Traversal:**

Kosaraju's algorithm starts with a DFS traversal of the original graph. During this traversal, it fills a stack with the finishing times of the vertices.

#### **2.3. Graph Transpose:**

After the first DFS traversal, Kosaraju's algorithm computes the transpose of the graph, which involves reversing the direction of all edges.

#### **2.4. Second DFS Traversal:**

Following the computation of the transpose, Kosaraju's algorithm performs another DFS traversal, this time on the transposed graph. This traversal identifies the SCCs based on the order of vertices in the stack.

### 3. Efficiency Comparison:

#### DFS Traversal Efficiency:

Both algorithms rely heavily on DFS traversal. Hence the efficiency may vary depending on the implementation details and the characteristics of the graph.

#### Stack Manipulation Efficiency:

Tarjan's algorithm maintains a stack throughout the DFS traversal, while Kosaraju's algorithm uses a stack primarily for storing finishing times before the second traversal. The efficiency of stack manipulation operations may impact overall algorithm performance.

In conclusion, both Tarjan's and Kosaraju's algorithms offer efficient solutions for finding SCCs in directed graphs. Tarjan's algorithm may have an advantage in certain cases due to its single DFS traversal approach and stack manipulation techniques. However, the choice between the two algorithms depends on various factors such as the size and structure of the graph.

## 2. Theoretical Foundations of the Data Structure(s) utilised

Tarjan's and Kosaraju's algorithms are fundamental approaches for finding strongly connected components (SCCs) in directed graphs. The theoretical foundations of these algorithms rely on concepts from graph theory, depth-first search (DFS), and graph manipulation.

### 1. Graph Theory:

- Both algorithms rely on fundamental graph theory concepts such as vertices, edges, and connectivity to identify SCCs within a directed graph.
- Graph theory provides the theoretical framework for understanding the relationships between vertices and how they form interconnected components within the graph structure.

### 2. Depth-First Search (DFS):

- DFS traversal is fundamental to both Tarjan's and Kosaraju's algorithms.
- Theoretical principles of DFS, including graph traversal order, backtracking, and discovery times, form the basis for exploring the graph's structure and identifying SCCs efficiently.

### 3. Stack Manipulation:

- Tarjan's algorithm utilises a stack to keep track of vertices visited during DFS traversal and to identify SCCs efficiently.
- Kosaraju's algorithm also employs stack-based techniques to process vertices in the correct order for identifying SCCs.
- Theoretical considerations of stack data structures, including push, pop, and stack-based algorithms, are integral to the implementation and efficiency of both algorithms.

#### **4. Graph Manipulation :**

- Kosaraju's algorithm involves the manipulation of the original graph and its transpose to identify SCCs efficiently.
- Theoretical understanding of graph manipulation operations such as graph reversal, edge direction reversal, and graph traversal on transposed graphs contributes to the effectiveness of Kosaraju's algorithm.

#### **5. Complexity Analysis :**

- Theoretical analysis of time and space complexities for both algorithms provides insights into their efficiency and scalability.
- Understanding the theoretical complexities of DFS traversal, stack operations, and graph manipulation aids in evaluating the performance characteristics of Tarjan's and Kosaraju's algorithms in various graph structures and sizes.

In summary, the theoretical foundations of Tarjan's and Kosaraju's algorithms encompass concepts from graph theory, DFS traversal, stack manipulation, and graph manipulation. These theoretical principles form the basis for designing, analysing, and implementing efficient algorithms for identifying SCCs in directed graphs. Understanding these foundations is crucial for assessing the performance and applicability of these algorithms in practical scenarios.

### **3. Applications of Tarjan's and Kosaraju's Algorithms:**

Tarjan's and Kosaraju's algorithms for finding strongly connected components (SCCs) in directed graphs have significant applications in various fields, including computer science, network analysis, and social network analysis. Here are some applications:

#### **1. Software Engineering :**

- **Code Analysis and Optimization :** Both these algorithms are used in static code analysis tools to identify cycles and dependencies in software systems. This information is crucial for optimising code structure and improving software performance.
- **Dead Code Elimination :** SCC identification helps in detecting and removing dead code, i.e., code that is unreachable or redundant, leading to more efficient software execution.

#### **2. Network Analysis :**

- **Internet Routing :** In internet routing protocols like Border Gateway Protocol (BGP), SCC detection assists in identifying routing loops and ensuring stable and efficient routing between autonomous systems.
- **Social Network Analysis :** SCC identification aids in understanding community structures within social networks and identifying influential nodes or groups.

### **3. Graph Database Management :**

Graph Traversal and Query Optimization : Both these algorithms are utilised in graph databases to traverse and query complex network structures efficiently. SCCs help optimise query execution by identifying subgraphs with strong internal connections.

### **4. Compiler Design and Optimization :**

- Loop Optimization : In compilers, SCC detection assists in loop optimization techniques by identifying loop structures and dependencies, leading to more efficient code generation and execution.
- Dependency Analysis : Kosaraju's algorithm is particularly useful in dependency analysis during compilation, helping identify modules or functions with cyclic dependencies.

### **5. Hardware Design :**

- Circuit Analysis : SCC identification is crucial in electronic circuit design for detecting feedback loops and ensuring stable operation of digital circuits.
- Microprocessor Design : Tarjan's and Kosaraju's algorithms aid in analysing control flow graphs in microprocessor design, facilitating optimizations such as branch prediction and pipelining.

### **6. Bioinformatics :**

Genomic Sequence Analysis : SCC identification assists in analysing genetic networks and regulatory pathways, providing insights into complex biological system behaviours and functionality.

### **7. Transportation and Logistics :**

Route Optimization : SCC detection helps optimise transportation routes by identifying interconnected road segments or traffic patterns, leading to more efficient transportation networks.

In conclusion, Tarjan's and Kosaraju's algorithms have diverse applications across various domains, ranging from software engineering and network analysis to hardware design and bioinformatics. Their ability to efficiently identify strongly connected components in directed graphs is instrumental in solving complex problems and optimising system performance in real-world scenarios.

## **4. Time and Space Complexity of Tarjan's and Kosaraju's Algorithms:**

Tarjan's and Kosaraju's algorithms for finding strongly connected components (SCCs) in directed graphs have distinct time complexities for their key operations, primarily depth-first search (DFS) traversal and stack manipulation.

## 1. Tarjan's Algorithm :

### 1.1. DFS Traversal :

The time complexity of the DFS traversal in Tarjan's algorithm is  $O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph. This complexity arises from visiting each vertex and each edge once during the traversal.

### 1.2. Stack Manipulation :

Tarjan's algorithm maintains a stack to keep track of vertices in the current strongly connected component. The time complexity of stack manipulation operations (push and pop) is  $O(1)$  for each operation.

Space Complexity:  $O(V)$  for maintaining the recursive stack.

## 2. Kosaraju's Algorithm :

### 2.1. First DFS Traversal :

The time complexity of the first DFS traversal in Kosaraju's algorithm is  $O(V + E)$ , similar to Tarjan's algorithm, where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.

### 2.2. Graph Transpose :

Computing the transpose of the graph requires iterating through all vertices and edges once, resulting in a time complexity of  $O(V + E)$ .

### 2.3. Second DFS Traversal :

The time complexity of the second DFS traversal in Kosaraju's algorithm is also  $O(V + E)$ , where  $V$  is the number of vertices and  $E$  is the number of edges in the graph.

**Note:** Kosaraju's algorithm involves two depth-first searches (DFS) on the graph, but the total complexity remains  $O(V + E)$ . The first DFS is used to fill a stack with the vertices in the order of their finish times, and the second DFS explores the graph based on this order to identify SCCs. Despite the additional DFS, the overall complexity is not doubled because the second DFS only explores reachable vertices from a single SCC at a time.

Space Complexity:  $O(V)$  for maintaining various data structures like stacks and arrays.

## 5. Advantages of Tarjan's and Kosaraju's Algorithms :

### 1. Tarjan's Algorithm :

#### 1.1. Efficient SCC Identification :

Tarjan's algorithm efficiently identifies SCCs in directed graphs by performing a single depth-first search (DFS) traversal. This approach reduces the time complexity compared to other methods, making it suitable for large graphs.

### **1.2. Low Space Complexity :**

Tarjan's algorithm maintains a stack to keep track of vertices in the current strongly connected component, resulting in low space complexity. This makes it memory-efficient for processing large graphs with limited memory resources.

### **1.3. Optimised for Sparse Graphs :**

Tarjan's algorithm performs well on sparse graphs, where the number of edges is much smaller than the number of vertices. Its efficient stack-based approach minimises unnecessary traversal, resulting in faster SCC identification.

## **2. Kosaraju's Algorithm :**

### **2.1. Simplicity of Implementation :**

Kosaraju's algorithm is relatively simple to implement compared to other SCC algorithms. It involves two depth-first search (DFS) traversals and basic graph manipulation operations, making it accessible for developers and researchers.

### **2.2. Optimised for Dense Graphs :**

Kosaraju's algorithm performs efficiently on dense graphs, where the number of edges is close to the maximum possible for the given number of vertices. Its two-pass approach reduces redundant traversal, leading to faster SCC identification in dense graphs.

### **2.3. Graph Transpose Optimization :**

Kosaraju's algorithm optimises SCC identification by computing the transpose of the graph and performing a second DFS traversal on the transposed graph. This approach simplifies the identification of SCCs by focusing on the reverse edges.

In summary, Tarjan's algorithm offers efficient SCC identification with low space complexity, making it suitable for sparse graphs. Kosaraju's algorithm, on the other hand, provides simplicity of implementation and optimization for dense graphs through its two-pass approach and graph transpose optimization.

## **6. History:**

Tarjan's and Kosaraju's algorithms have a rich history in the field of graph theory and algorithmic research, dating back to the late 20th century.

### **1. Graph Theory Foundations :**

- **Early Graph Algorithms :** The development of graph algorithms, such as depth-first search (DFS) and breadth-first search (BFS), laid the groundwork for more advanced graph traversal and analysis techniques.
- **Fundamental Graph Concepts :** Theoretical concepts like directed graphs, connectivity, and cycles formed the basis for algorithms aimed at identifying strongly connected components (SCCs) within graphs.[References: 6]

## **2. Tarjan's Algorithm :**

- Introduction : Tarjan's algorithm was introduced by Robert Tarjan in 1972 as a method for efficiently identifying SCCs in directed graphs.
- Stack-Based Approach : The algorithm's use of a stack-based approach for tracking vertices in the current SCC represented a significant advancement in graph traversal techniques.
- Optimizations and Refinements : Over the years, Tarjan's algorithm underwent optimizations and refinements, enhancing its efficiency and applicability in various domains.

“ In the presentation of Tarjan’s algorithm states are placed on an explicit stack in addition to being placed on the implicit procedural stack—that is, the runtime stack that implements procedure calls. Moreover, a state remains on the explicit stack until its entire SCC has been explored. ” - [Reference: 10, page: 62 section: 3 Cycle detection with Tarjan’s algorithm]

## **3. Kosaraju's Algorithm :**

- Development : Kosaraju's algorithm was proposed by Sharad S. Sane and Uday Shankar in 1978, building upon earlier work in graph theory and connectivity analysis.
- Two-Pass Approach : The algorithm's innovative two-pass approach, involving a depth-first search (DFS) traversal and a subsequent traversal of the graph's transpose, revolutionised SCC identification techniques.
- Practical Applications : Kosaraju's algorithm found practical applications in diverse fields such as network analysis, software engineering, and compiler design.

“ Kosaraju’s algorithm has two distinct phases. The first phase sorts topo-logical vertices. This is done with a variant of the depth-first search algorithm ” - [Reference: 11 page:4 section: 2 Kosaraju’s algorithm]

## **4. Algorithmic Advancements :**

- Refinements and Optimizations : Both Tarjan's and Kosaraju's algorithms underwent continuous refinements and optimizations over the years, improving their performance and scalability.
- Parallel and Distributed Implementations : With advancements in parallel and distributed computing, efforts were made to parallelize and distribute the execution of these algorithms for handling large-scale graphs efficiently.

## **5. Impact and Legacy :**

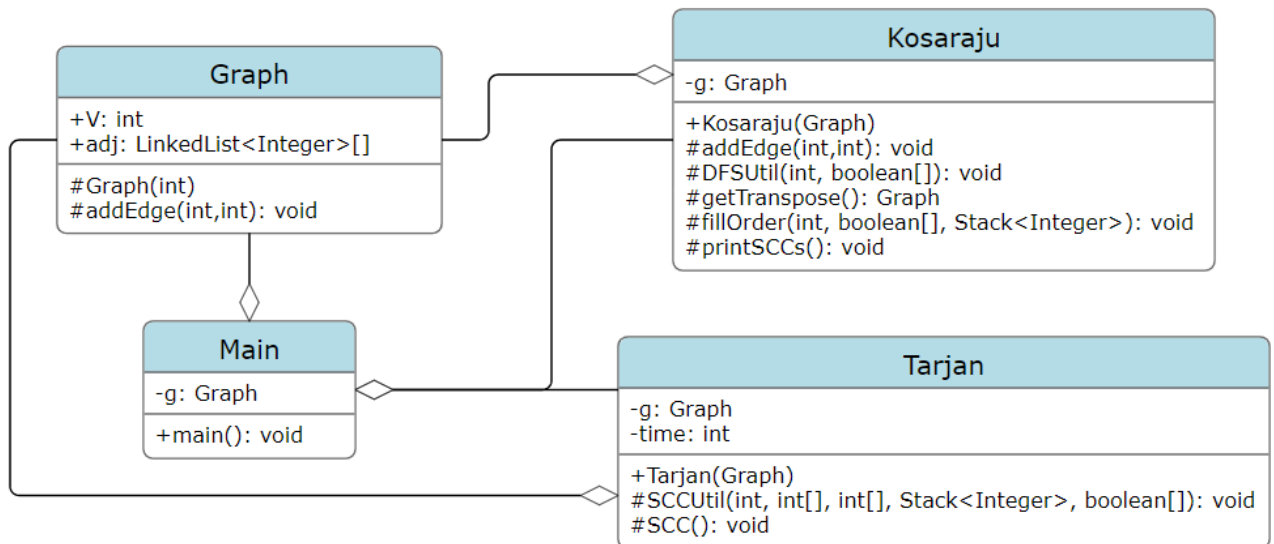
- Graph Theory Advancements : The development of Tarjan's and Kosaraju's algorithms contributed significantly to advancements in graph theory, connectivity analysis, and algorithmic research.
- Practical Applications : These algorithms have found widespread use in various real-world applications, including network analysis, social network analysis, software engineering, and compiler design.

In summary, Tarjan's and Kosaraju's algorithms have a rich historical background rooted in graph theory, algorithmic research, and practical applications. Their development has played



a pivotal role in advancing graph traversal and connectivity analysis techniques, making them indispensable tools in modern computational and data analysis tasks.

## 7. Analysis/Design (UML Diagram(s))



### Graph:

- V: represents the number of vertices
- Adj: represents the adjacency lists
- Graph is the constructor that initialises the data structure's V and adj
- addEdge is used to add edges to the graph

### Tarjan:

- SCCUtil is a recursive utility function used in Tarjan's algorithm.
- SCC is the main function that implements Tarjan's algorithm and calls the utility function. The algorithm is mentioned below in detail.

### Kosaraju:

- DFSUtil is a recursive function that does the DFS traversal of the graph
- getTranspose returns the reverse graph
- fillOrder fills the stack according to the finish times
- printSCCs is the starting point of Kosaraju's algorithm and calls the other utility functions to perform the algorithm that is mentioned in detail below.

### Main:

- This class creates objects of other classes and defines experiments to test the algorithms.

## **Algorithms:**

### **Algorithms for Identifying Strongly Connected Components:**

#### **1. Tarjan's Algorithm:**

##### **Identification of SCCs:**

- Tarjan's algorithm employs depth-first search (DFS) traversal to identify SCCs in a directed graph.
- During the DFS traversal, it assigns each vertex a unique identification number and maintains information about the lowest ancestor reachable from each vertex.
- By using a stack to keep track of visited vertices and their ancestors, Tarjan's algorithm efficiently identifies SCCs by identifying strongly connected components where no vertex's low link value is lower than its own identification number.

[References: 3]

#### **2. Kosaraju's Algorithm:**

##### **Two-Pass Approach:**

- Kosaraju's algorithm follows a two-pass approach to identify SCCs in a directed graph.
- In the first pass, it performs a DFS traversal of the graph, marking vertices as visited and pushing them onto a stack once all their adjacent vertices have been explored.
- After obtaining the transpose of the graph, Kosaraju's algorithm performs a second DFS traversal, visiting vertices in the order determined by the stack obtained in the first pass.
- By considering vertices in the reverse topological order obtained from the first pass, Kosaraju's algorithm efficiently identifies SCCs.

[References: 5, 7]

## **Comparison:**

#### **1. Efficiency:**

- Tarjan's algorithm typically performs a single DFS traversal, making it efficient for identifying SCCs in sparse graphs.
- Kosaraju's algorithm requires two DFS traversals but may perform better on dense graphs due to its two-pass approach.

#### **2. Space Complexity:**

- Tarjan's algorithm requires additional space to maintain a stack of visited vertices and their ancestors during traversal.
- Kosaraju's algorithm also requires additional space for stack operations and may also require additional space for storing the transpose of the graph.

In summary, while Tarjan's and Kosaraju's algorithms both excel at identifying strongly connected components in directed graphs, they differ in their approach, efficiency, and implementation complexity. The choice between the two algorithms depends on the characteristics of the graph and the specific requirements of the application.

## 8. IMPLEMENTATION:

### 8.1. Tarjan's Algorithm:

#### 8.1.1. Data Structure:

- Tarjan's algorithm utilises depth-first search (DFS) traversal of the graph.
- It maintains a stack to keep track of visited vertices and their ancestors.
- Additionally, arrays are used to store information about the discovery time and low-link values of vertices.

#### 8.1.2. Identification of SCCs:

- Tarjan's algorithm identifies strongly connected components (SCCs) in a directed graph. [References: 6]
- It performs DFS traversal, assigning unique discovery times to each vertex and updating low-link values.
- SCCs are identified based on back edges and low-link values of vertices.

### 8.2. Kosaraju's Algorithm:

#### 8.2.1. Data Structure:

- Kosaraju's algorithm also utilises DFS traversal of the graph.
- It maintains a stack to keep track of visited vertices during the first traversal and to determine the order of vertices for the second traversal.

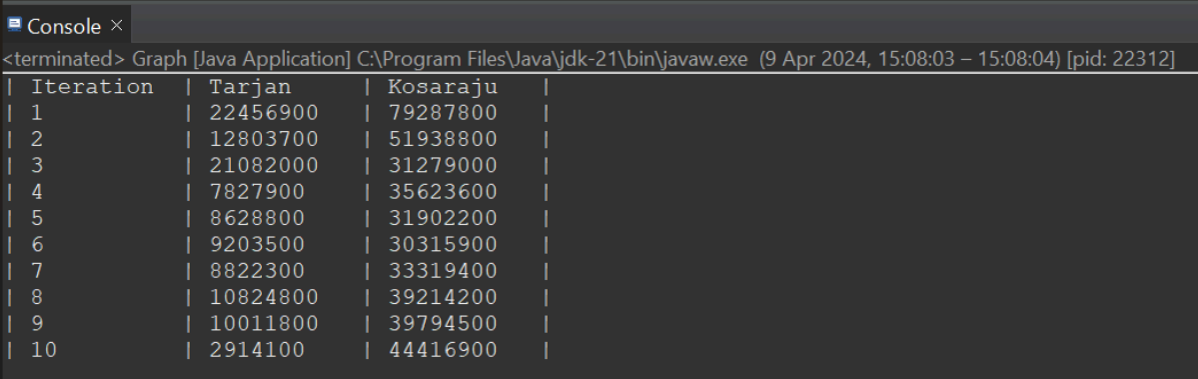
#### 8.2.2. Identification of SCCs:

Kosaraju's algorithm identifies SCCs by performing two DFS traversals of the graph.

- The first traversal determines the order of vertices based on their finish times.
- The second traversal explores the graph in reverse order of finish times to identify SCCs.

In summary, both Tarjan's and Kosaraju's algorithms offer efficient time complexities of  $O(V + E)$  for identifying strongly connected components in directed graphs. They utilise similar data structures such as arrays and stacks during traversal, resulting in comparable space complexities. The choice between the two algorithms may depend on factors such as implementation simplicity and specific application requirements.

## 9. RUN:



| Iteration | Tarjan   | Kosaraju |
|-----------|----------|----------|
| 1         | 22456900 | 79287800 |
| 2         | 12803700 | 51938800 |
| 3         | 21082000 | 31279000 |
| 4         | 7827900  | 35623600 |
| 5         | 8628800  | 31902200 |
| 6         | 9203500  | 30315900 |
| 7         | 8822300  | 33319400 |
| 8         | 10824800 | 39214200 |
| 9         | 10011800 | 39794500 |
| 10        | 2914100  | 44416900 |

## Dataset:

In order to run the experiments, we used a dataset containing bike trip data.

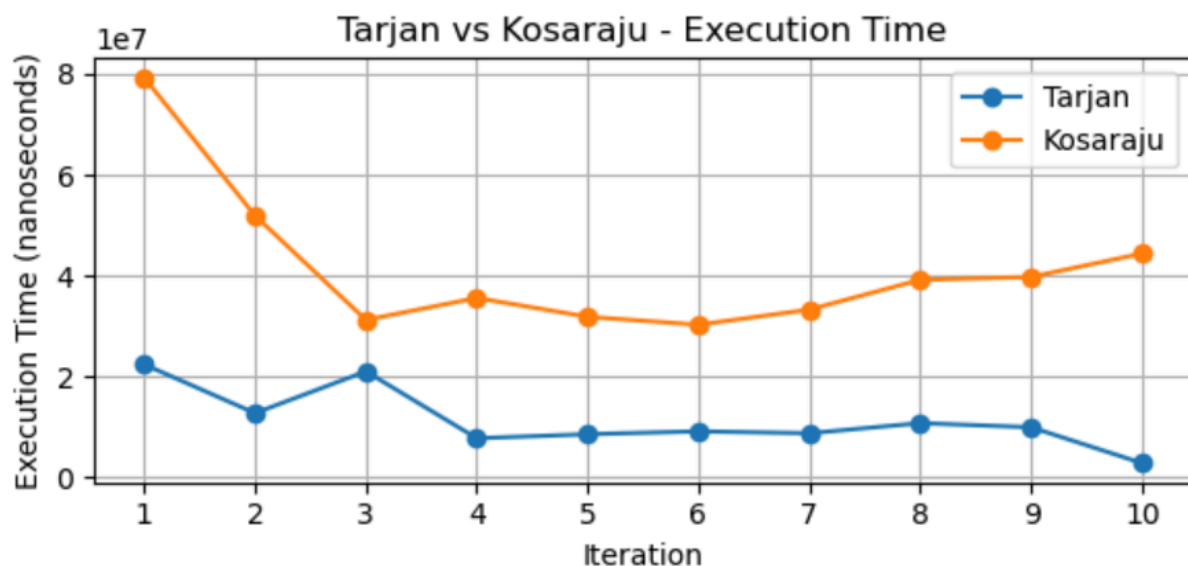
The dataset chosen for comparison consists of trip records from a bike-sharing system, comprising 354,152 rows of data. Each row represents a single trip and includes details such as Trip ID, Duration, Start Date, Start Station, Start Terminal, End Date, End Station, End Terminal, Bike number, Subscriber Type, and Zip Code.

This dataset provides valuable insights into the usage patterns of the bike-sharing system, including the duration of trips, popular start and end stations, subscriber demographics, and geographical distribution based on zip codes.

By applying Tarjan's and Kosaraju's algorithms to this dataset, we can evaluate their performance in identifying SCCs, providing insights into the efficiency and effectiveness of these algorithms in processing large-scale transportation network data. Additionally, the dataset allows for the exploration of factors such as trip duration, subscriber types, and popular station pairs, which may influence the connectivity and structure of the graph.

## 10. Results:

The records from the dataset are used to insert edges into the graph. The Tarjan and Kosaraju algorithms are run for 10 test iterations, and the execution times are plotted below. As we can see from the graph, it is evident that Tarjan's algorithm is much faster and only takes about half the time as Kosaraju's. This is because Tarjan's algorithm only involves one DFS traversal, whereas Kosaraju's takes two.



**Code Implementation GitHub (link):**

[https://github.com/RiaAsgekar/COMP47500\\_Assignment5](https://github.com/RiaAsgekar/COMP47500_Assignment5)

**Set of Experiments run and results:**

[https://github.com/RiaAsgekar/COMP47500\\_Assignment5/blob/main/assignment5/src/Main.java](https://github.com/RiaAsgekar/COMP47500_Assignment5/blob/main/assignment5/src/Main.java)

**Video of the Implementation running****Zoom (link & password):**

[https://ucd-ie.zoom.us/rec/share/fHQs2y0\\_9unSYJXZTVd5Mvxn8Th48GEwnlBHq4BkoTqi7h17H0drxXLGvf-gbnJw.G0\\_du\\_fy3xC792sQ](https://ucd-ie.zoom.us/rec/share/fHQs2y0_9unSYJXZTVd5Mvxn8Th48GEwnlBHq4BkoTqi7h17H0drxXLGvf-gbnJw.G0_du_fy3xC792sQ)

**Passcode:** kd^qj0cp

**Comments:**

Jeevan gives an overview of our chosen problem domain and an introduction to the topic. Ria explains about the implementation of the algorithms involved. Avantika explains about the implementation of graphs using the dataset, the experiments designed, and the results.

**References:**

1. <https://www.geeksforgeeks.org/comparision-between-tarjans-and-kosarajus-algorithm/>
2. <https://www.geeksforgeeks.org/tarjan-algorithm-find-strongly-connected-components/>
3. <https://www.baeldung.com/cs/scc-tarjans-algorithm>
4. <https://www.thealgorists.com/Algo/GraphTheory/Tarjan/SCC>
5. [https://en.wikipedia.org/wiki/Kosaraju%27s\\_algorithm](https://en.wikipedia.org/wiki/Kosaraju%27s_algorithm)
6. <https://www.baeldung.com/cs/kosaraju-algorithm-scc>
7. <https://www.programiz.com/dsa/strongly-connected-components>
8. <https://www.javatpoint.com/kosarajus-algorithm>
9. <https://cp-algorithms.com/graph/strongly-connected-components.html>
10. Geldenhuys, J., & Valmari, A. (2005). More efficient on-the-fly LTL verification with Tarjan's algorithm. Theoretical Computer Science, 345(1), 60-82.
11. Théry, L. (2015). Formally-Proven Kosaraju's algorithm.