Graph-Based Word Ladder

PREPARED FOR :

Miss Urooj

PREPARED BY:

- Waniya Badar 22k-4526
- Nimil Zubair 22k-4617
- Alishba Hassan 22k-4333

Table of Contents

1. Introduction

Project Overview:

The Word Ladder game solver is implemented using graph theory principles to find the shortest and longest transformation sequences between two words. Each word is modeled as a node in a graph, and edges connect words differing by only one letter. This project aims to showcase the practical applications of graph algorithms, compare efficiency between Python and MATLAB implementations, and analyze performance across different test cases.

Objectives:

- **Primary Goal:** Develop solutions to find both the shortest and longest word ladders using graph traversal algorithms.
- **Graph Representation:** Each word is a node; edges exist between words with a single letter difference.
- **Algorithm:** Breadth-First Search (BFS) for shortest path and a modified graph traversal algorithm for the longest path.
- **Comparative Analysis:** Compare execution times and performance in Python and MATLAB implementations.

2. Implementation Details

Graph Construction:

- A graph is constructed with nodes representing words and edges connecting words with one-letter differences.
- Both MATLAB and Python use dictionaries to load valid words and construct adjacency lists for graph representation.

Algorithms:

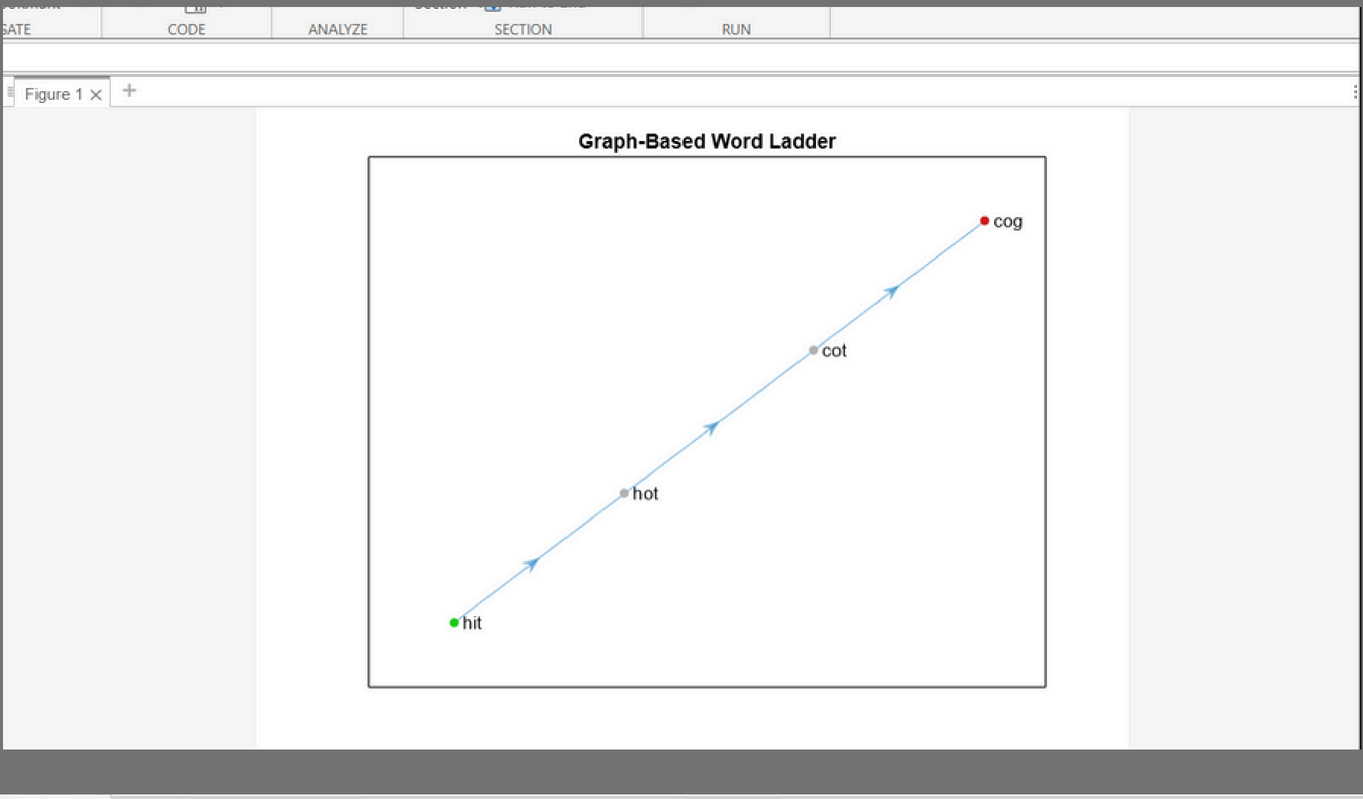
- **Shortest Path:** Implemented using BFS to ensure the minimum steps from start to end word.
- **Longest Path:** Explored all possible paths using depth-first strategies to determine the longest valid sequence.

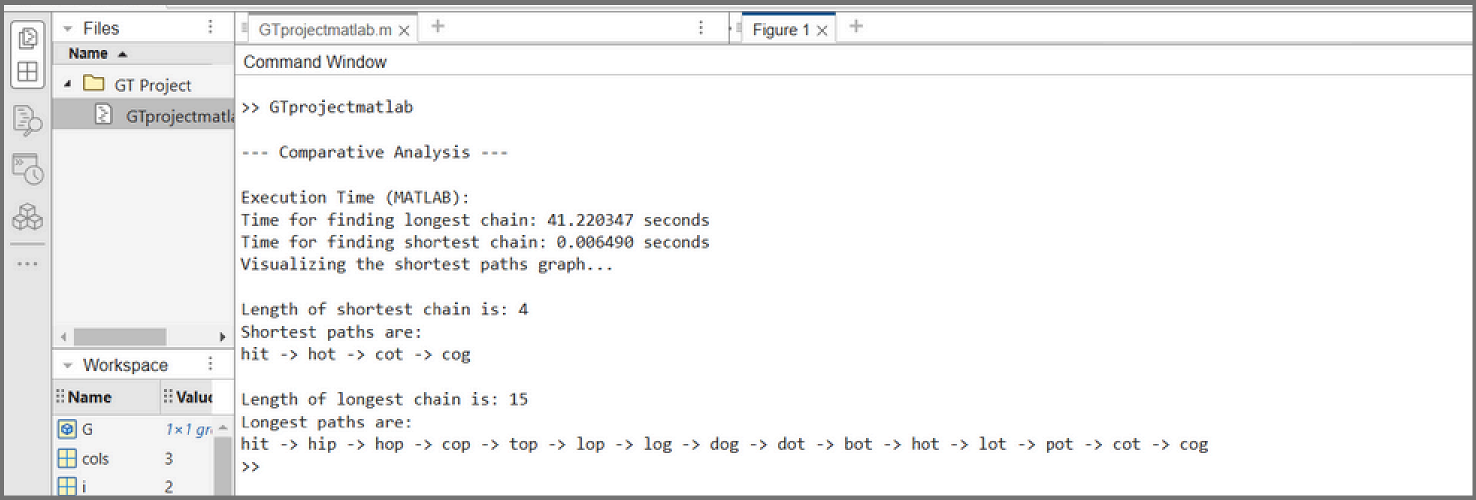
3. Test Cases and Results

1. Valid Test Case

- **Start Word:** hit
- **Target Word:** cog
- **Word Set:** {'hit', 'hot', 'dot', 'dog', 'cog', 'lot', 'log', 'hip', 'hop', 'top', 'lop', 'bot', 'pot', 'cop', 'cot'}

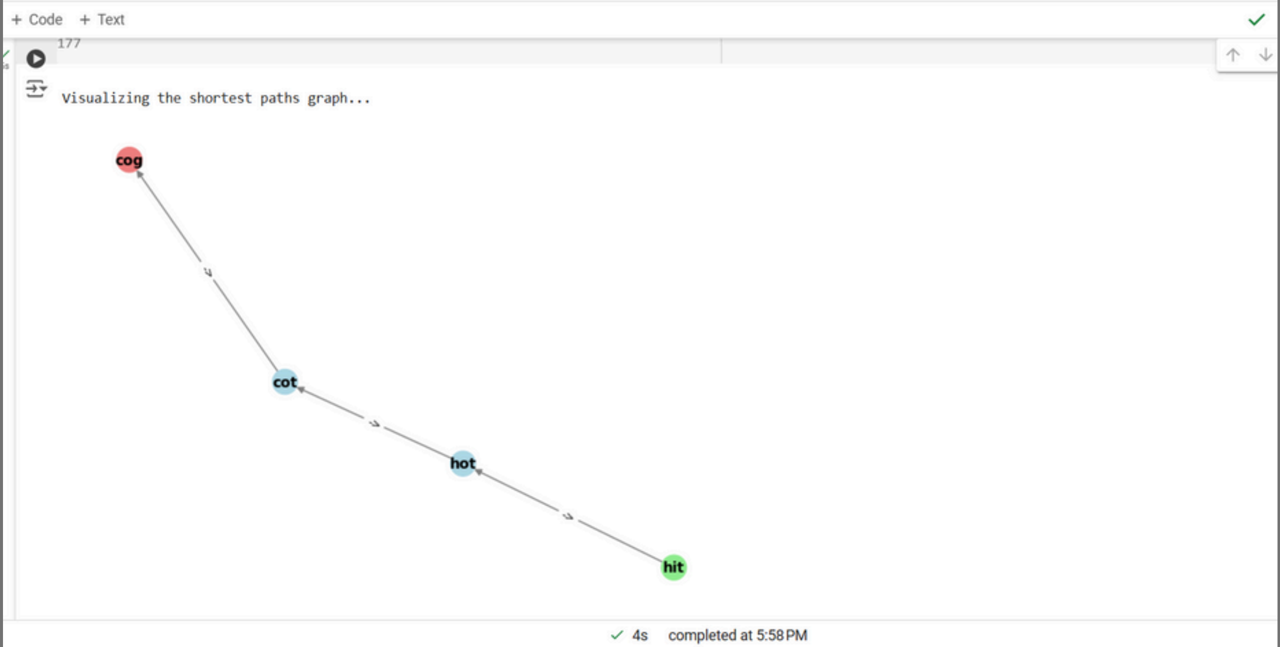
MATLAB Results:



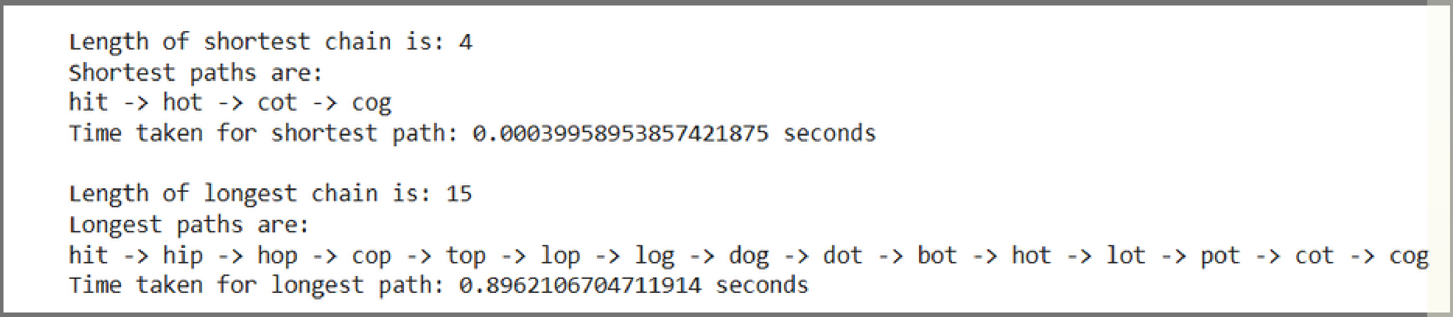


```
>> GTprojectmatlab

--- Comparative Analysis ---

Execution Time (MATLAB):
Time for finding longest chain: 41.220347 seconds
Time for finding shortest chain: 0.006490 seconds
Visualizing the shortest paths graph...

Length of shortest chain is: 4
Shortest paths are:
hit -> hot -> cot -> cog

Length of longest chain is: 15
Longest paths are:
hit -> hip -> hop -> cop -> top -> lop -> log -> dog -> dot -> bot -> hot -> lot -> pot -> cot -> cog
>>
```

Python Results:



```
Visualizing the shortest paths graph...
```

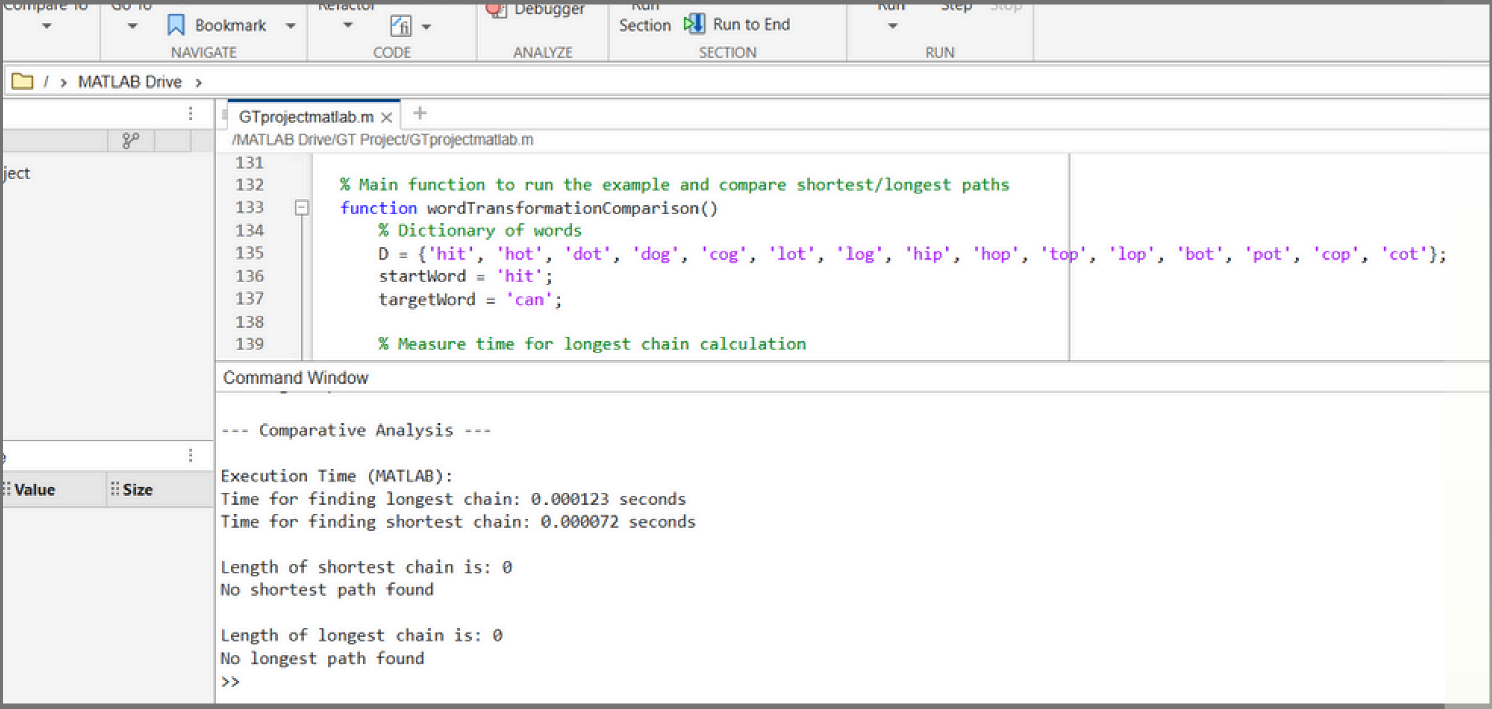


```
Length of shortest chain is: 4
Shortest paths are:
hit -> hot -> cot -> cog
Time taken for shortest path: 0.00039958953857421875 seconds

Length of longest chain is: 15
Longest paths are:
hit -> hip -> hop -> cop -> top -> lop -> log -> dog -> dot -> bot -> hot -> lot -> pot -> cot -> cog
Time taken for longest path: 0.8962106704711914 seconds
```

2. Invalid Test Case

- **Start Word:** hit
- **Target Word:** can
- **Word Set:**{'hit', 'hot', 'dot', 'dog', 'cog', 'lot', 'log', 'hip', 'hop', 'top', 'lop', 'bot', 'pot', 'cop', 'cot'}

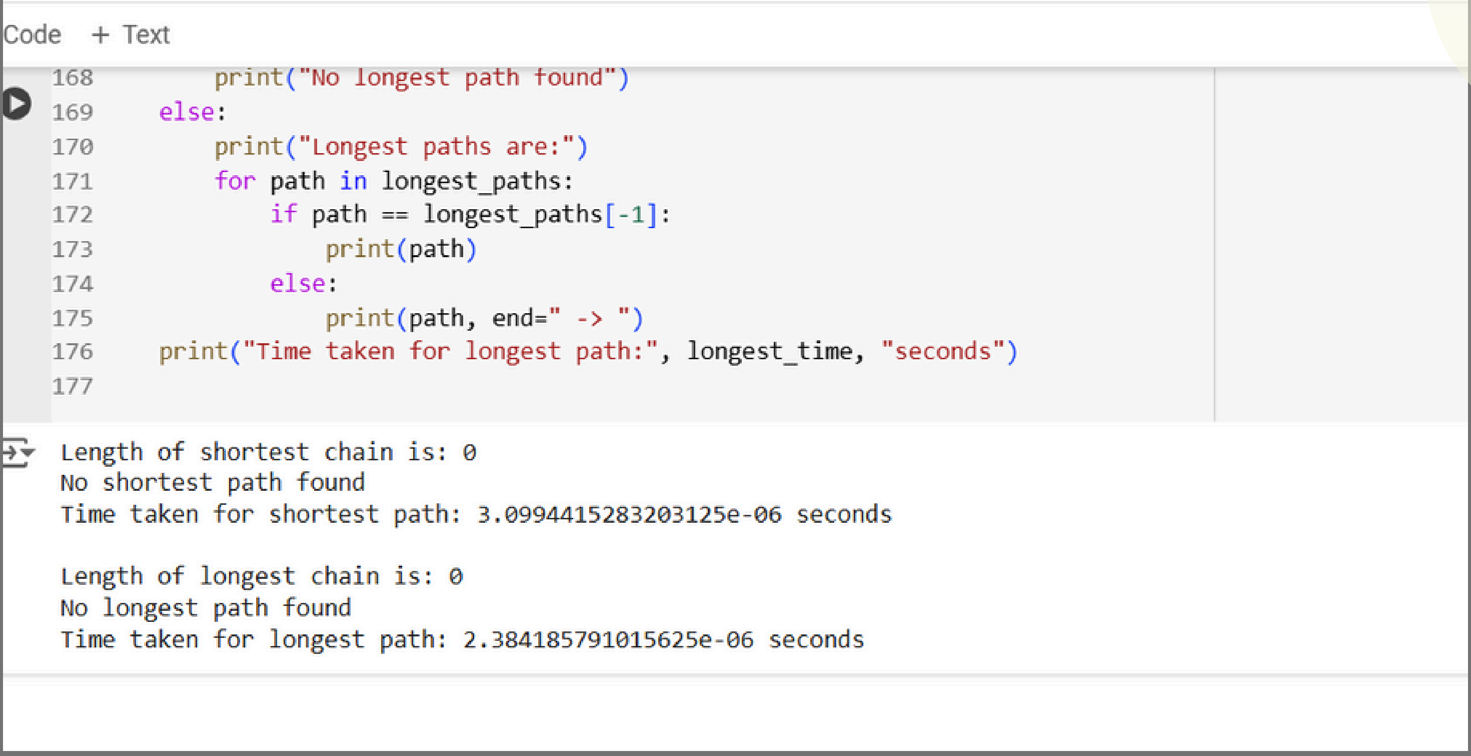
MATLAB Results:



```

% Main function to run the example and compare shortest/longest paths
function wordTransformationComparison()
    % Dictionary of words
    D = {'hit', 'hot', 'dot', 'dog', 'cog', 'lot', 'log', 'hip', 'hop', 'top', 'lop', 'bot', 'pot', 'cop', 'cot'};
    startWord = 'hit';
    targetWord = 'can';

    % Measure time for longest chain calculation
```

```
--- Comparative Analysis ---

Execution Time (MATLAB):
Time for finding longest chain: 0.000123 seconds
Time for finding shortest chain: 0.000072 seconds

Length of shortest chain is: 0
No shortest path found

Length of longest chain is: 0
No longest path found
>>
```

Python Results:



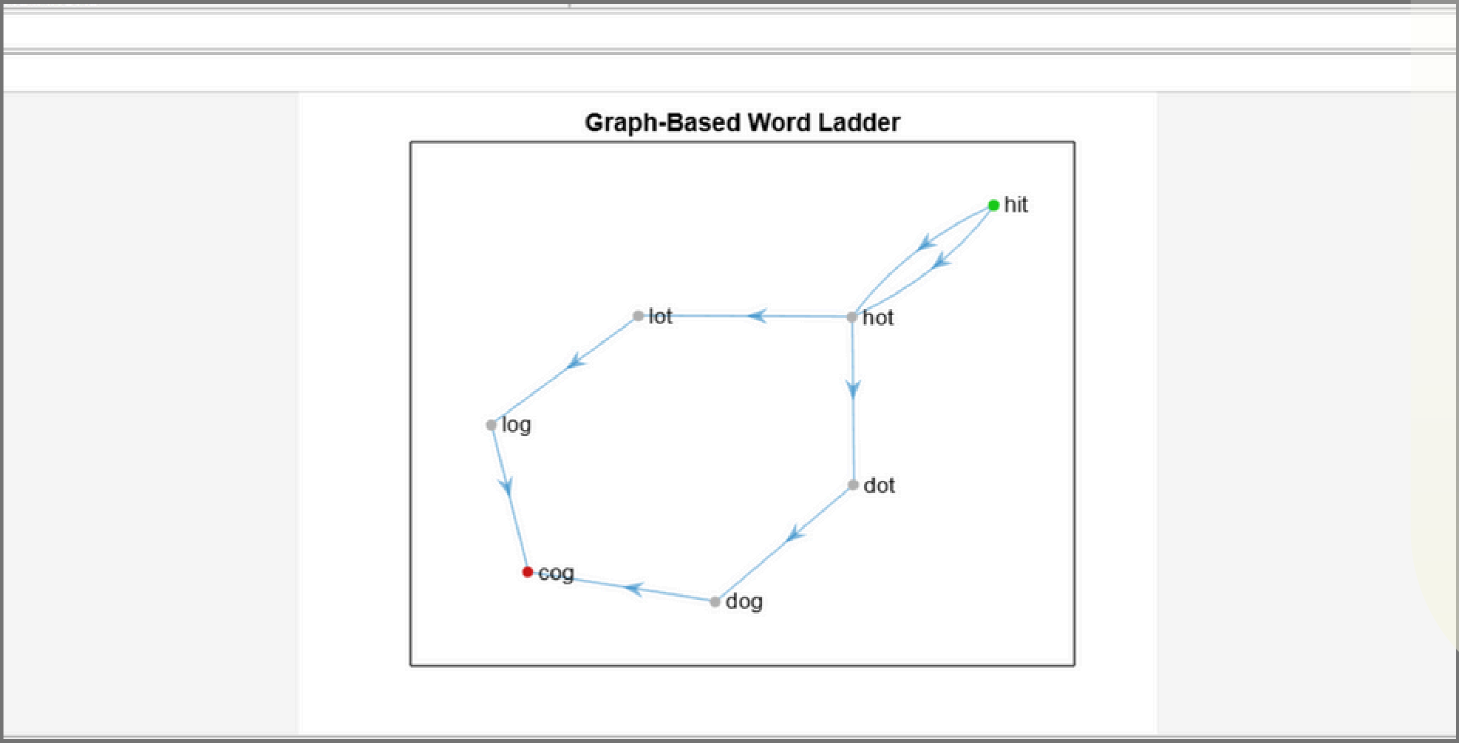
```
        print("No longest path found")
    else:
        print("Longest paths are:")
        for path in longest_paths:
            if path == longest_paths[-1]:
                print(path)
            else:
                print(path, end=" -> ")
    print("Time taken for longest path:", longest_time, "seconds")

```

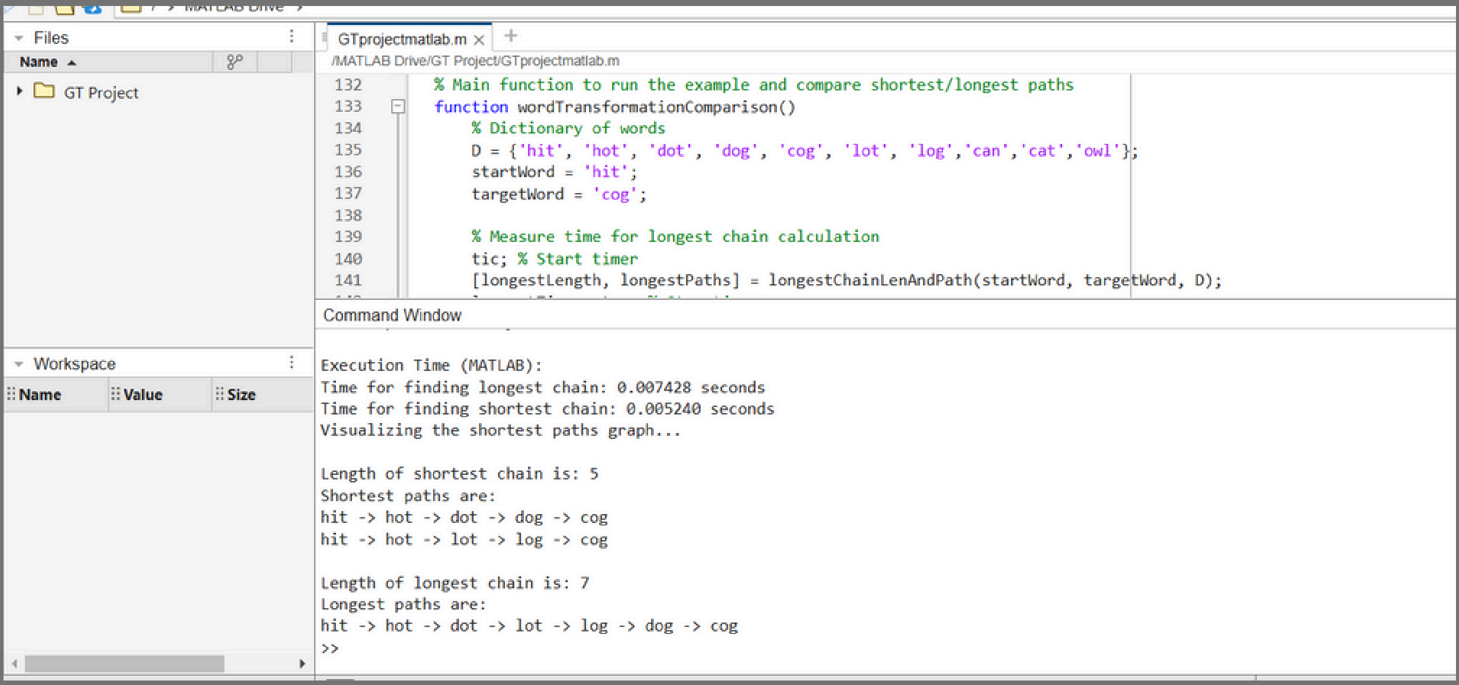
```
Length of shortest chain is: 0
No shortest path found
Time taken for shortest path: 3.0994415283203125e-06 seconds

Length of longest chain is: 0
No longest path found
Time taken for longest path: 2.384185791015625e-06 seconds
```

3. Multiple Possible Paths Test Case

- **Start Word: hit**
- **Target Word:** cog
- **Word Set:** {'hit', 'hot', 'dot', 'dog', 'cog', 'lot', 'log', 'can', 'cat', 'owl'}

MATLAB Results:

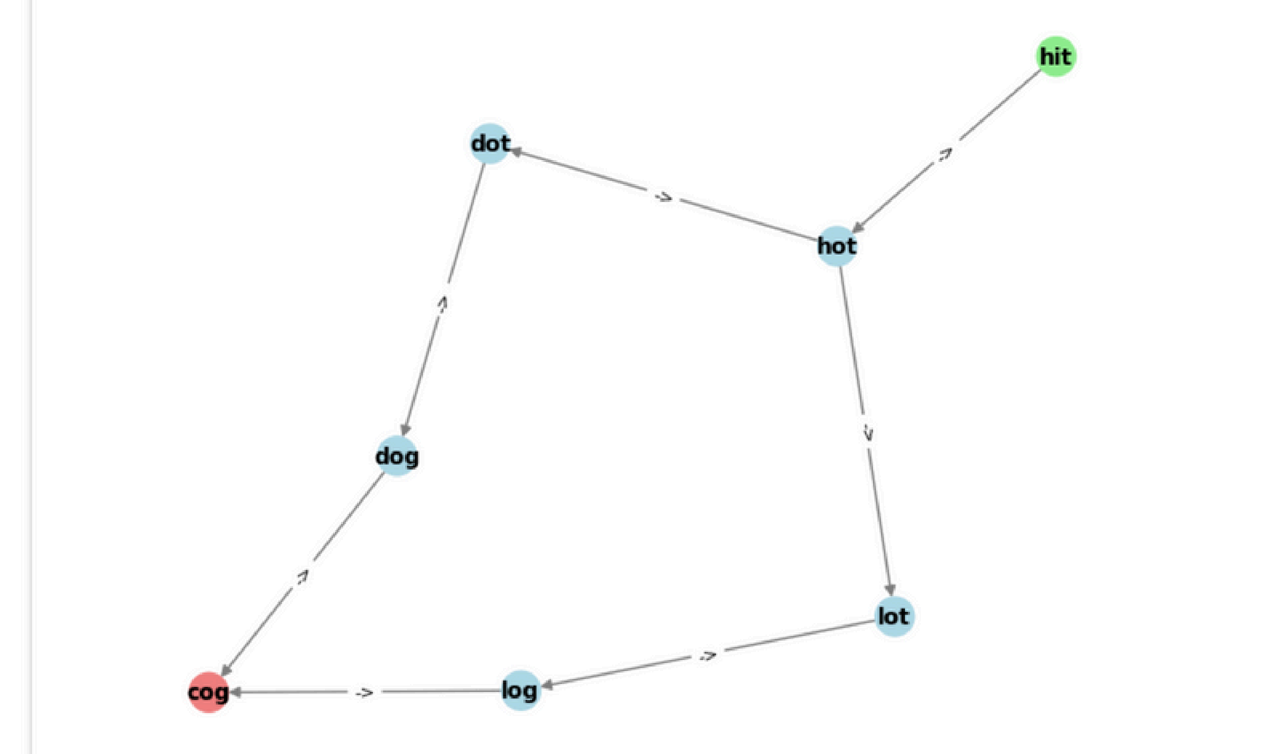


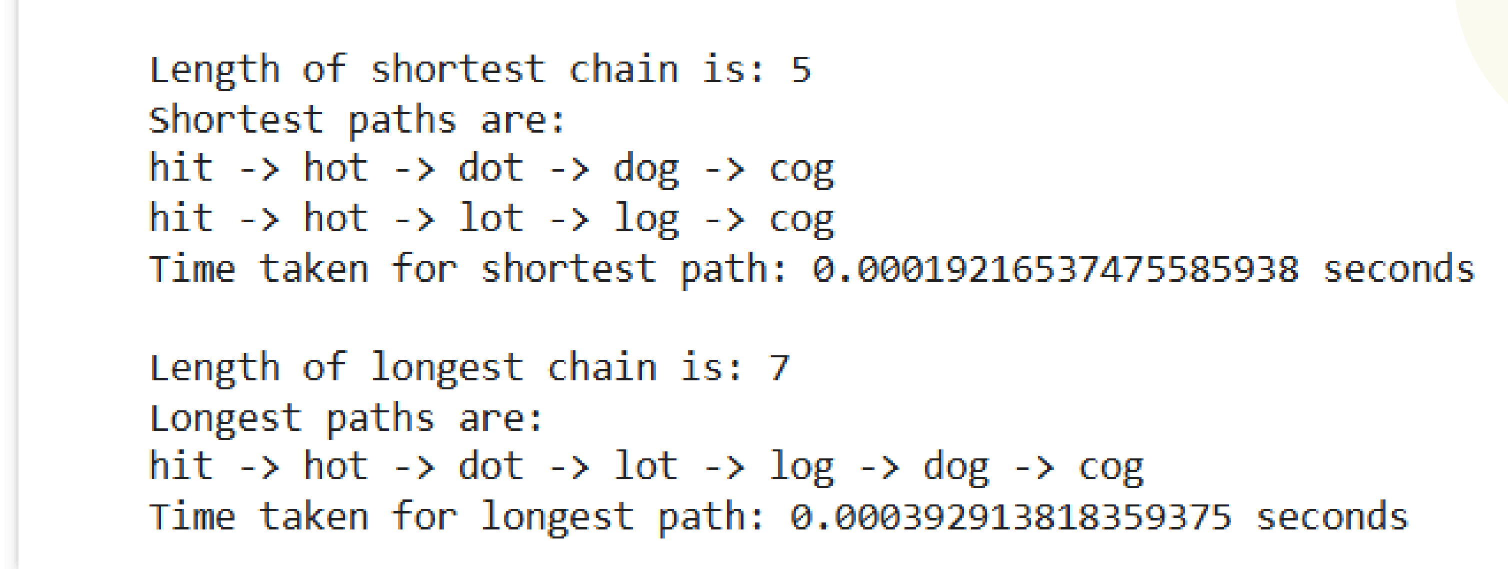


```
% Main function to run the example and compare shortest/longest paths
function wordTransformationComparison()
    % Dictionary of words
    D = {'hit', 'hot', 'dot', 'dog', 'cog', 'lot', 'log','can','cat','owl'};
    startWord = 'hit';
    targetWord = 'cog';

    % Measure time for longest chain calculation
    tic; % Start timer
    [longestLength, longestPaths] = longestChainLenAndPath(startWord, targetWord, D);
```

```
Execution Time (MATLAB):
Time for finding longest chain: 0.007428 seconds
Time for finding shortest chain: 0.005240 seconds
Visualizing the shortest paths graph...

Length of shortest chain is: 5
Shortest paths are:
hit -> hot -> dot -> dog -> cog
hit -> hot -> lot -> log -> cog

Length of longest chain is: 7
Longest paths are:
hit -> hot -> dot -> lot -> log -> dog -> cog
>>
```

Python Results:





```
Length of shortest chain is: 5
Shortest paths are:
hit -> hot -> dot -> dog -> cog
hit -> hot -> lot -> log -> cog
Time taken for shortest path: 0.00019216537475585938 seconds

Length of longest chain is: 7
Longest paths are:
hit -> hot -> dot -> lot -> log -> dog -> cog
Time taken for longest path: 0.000392913818359375 seconds
```

4. Performance Analysis

Metric	MATLAB	Python
Shortest Path	Accurate and efficient. Faster for smaller graphs.	Accurate and faster for all scenarios.
Longest Path	Accurate but slow for larger graphs due to exhaustive search.	More optimized in comparison to MATLAB.
Execution Time	Higher due to inherent overhead in graph traversal.	Generally faster with better handling of BFS and DFS.
Memory Usage	Efficient with smaller datasets.	Slightly higher memory usage due to Python’s graph representation.
Language Strengths	Easy visualization of graphs.	Better suited for rapid prototyping and scalability.

5. Conclusion

- **Efficiency:** Python outperformed MATLAB in execution times for both shortest and longest path calculations, particularly for larger graphs.
- **Accuracy:** Both implementations produced accurate results, confirming the validity of the algorithms.
- **Visualization:** MATLAB provided better tools for visualizing graphs and paths, while Python was more efficient in handling complex cases.
- **Language Choice:** Python is recommended for larger datasets or when speed is critical, while MATLAB can be used for smaller datasets or when visualization is essential.

6. Recommendations

- **Optimizations:** Implement advanced path-finding algorithms like A* or bidirectional BFS for faster results in Python.
- **Scalability:** Extend the solution to work with dynamic dictionaries for real-world applications.
- **Visualization:** Use libraries like NetworkX in Python to improve graph visualization.
- **Further Testing:** Evaluate performance on larger datasets and incorporate heuristic-based methods for faster execution.