Longest Path

Jacob Susko and Zach Kiser

Applications of Longest Path

Scheduling

Helps a team identify a critical path in their project scheduling, determining minimum completion time of the project

Networking

Finding the longest route in a network

Why not use Bellman-Ford

- First intuition is to take the negative weights on all the edges and use Bellman-Ford
- This does not work because of negative cycles
- Bellman-Ford assumes that there will be no negative cycles in the graph

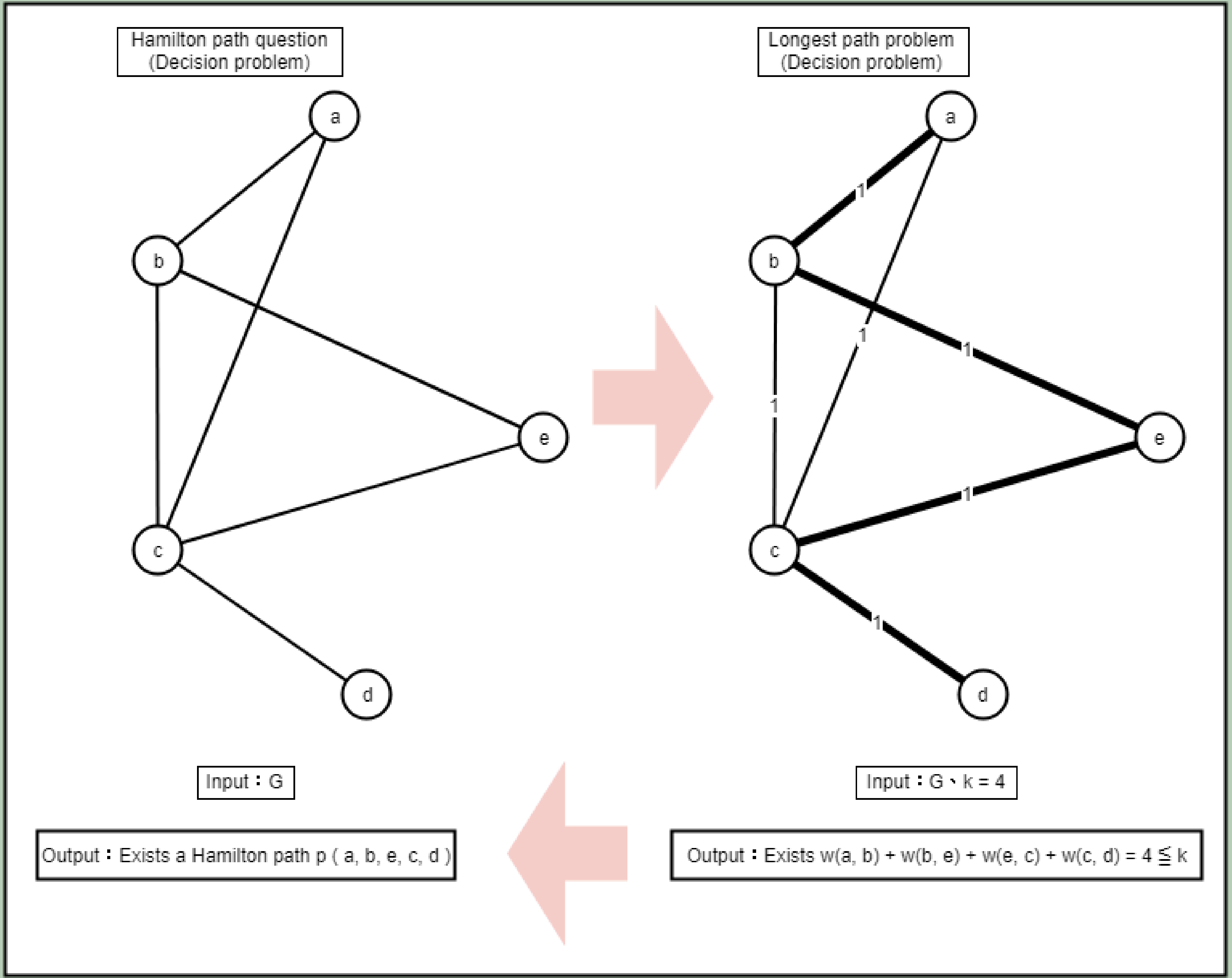
Certifier Process

- A solution can be verified in polynomial time
- Check that the path is valid -> O(E)
    - requires checking every edge along the path
- Verify that the path contains no repeated nodes

Reduction

Reducing from another NP-Complete Problem

- Can be solved with a reduction from Hamiltonian Path Problem
- A graph F has a Hamiltonian path if and only if its longest path has length n-1, where n = number of vertices in G
- Hamiltonian is NP-Complete (thus NP-Hard), so this reduction shows that the longest path problem is also NP-Complete

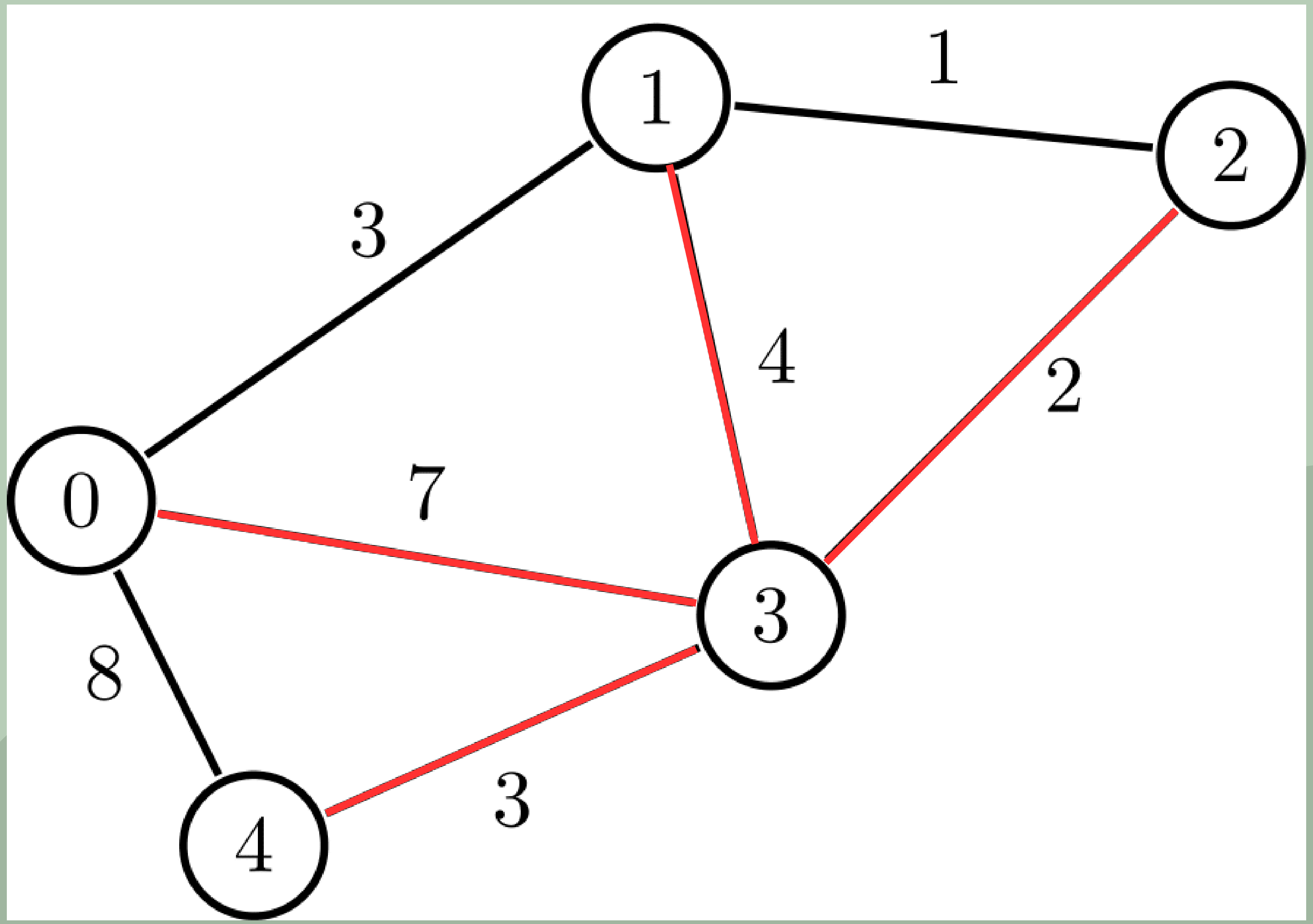


https://www.geeksforgeeks.org/optimized-longest-path-is-np-complete/
https://wangwilly.github.io/willywangkaa/2018/10/15/Algorithm-NP-completeness/
Schrijver, Alexander (2003), Combinatorial Optimization: Polyhedra and Efficiency, Volume 1, Algorithms and Combinatorics, vol. 24, Springer, p. 114, ISBN 9783540443896.

Approximate Solution

Approximate

Select locally optimal solution at each step (7 in this case)



How does it work?

A greedy algorithm that selects an initial node and iterates through all neighbors, selecting the highest edge weight at each step

Runtime and Approximation Ratio

The runtime is O(E), where E is the number of edges in the graph

The best polynomial time approximation algorithm known for this case achieves only a very weak approximation ratio

$$n/\exp(\Omega(\sqrt{\log n}))$$

Approximate

When is the solution not optimal?

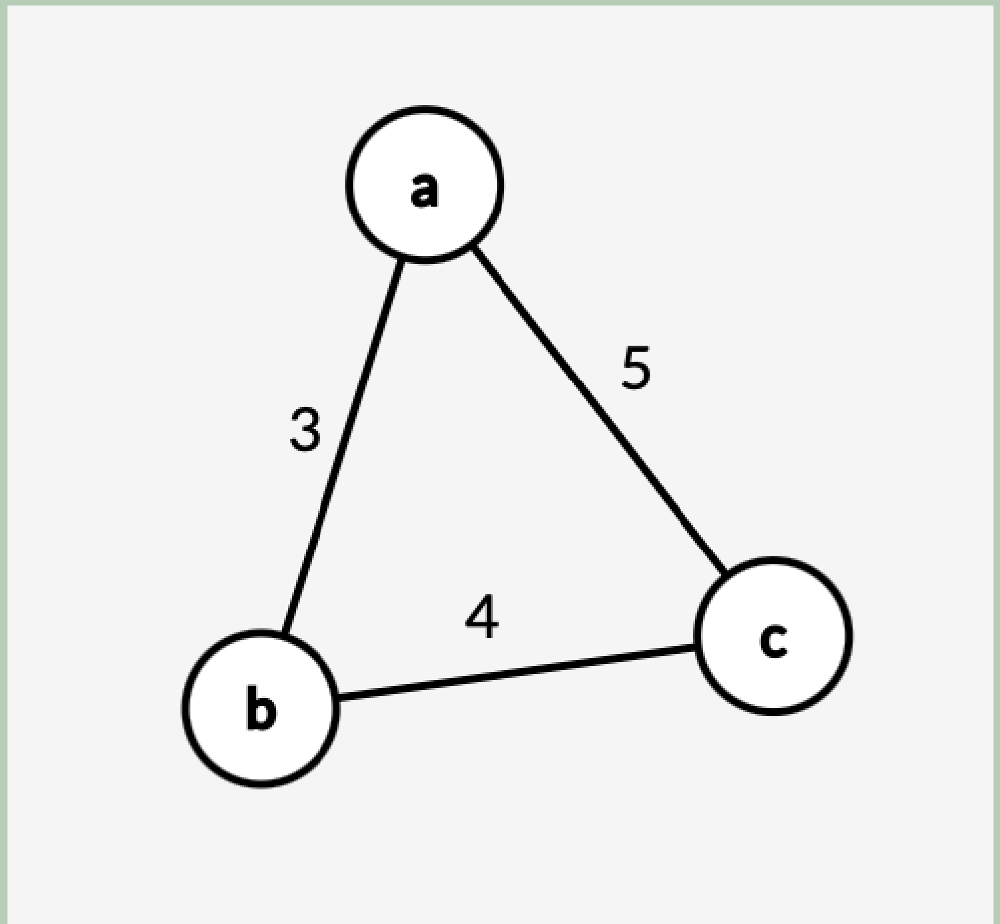
A suboptimal result occurs when there is a larger edge weight that does not lead to the longer path. In this example, the approximation algorithm will return the path a->c, when the correct result is a->b->c

Input

```
3 3
0 1 3
1 2 4
0 2 5
```

Output

```
5
0 2
```



Approximate Pseudocode

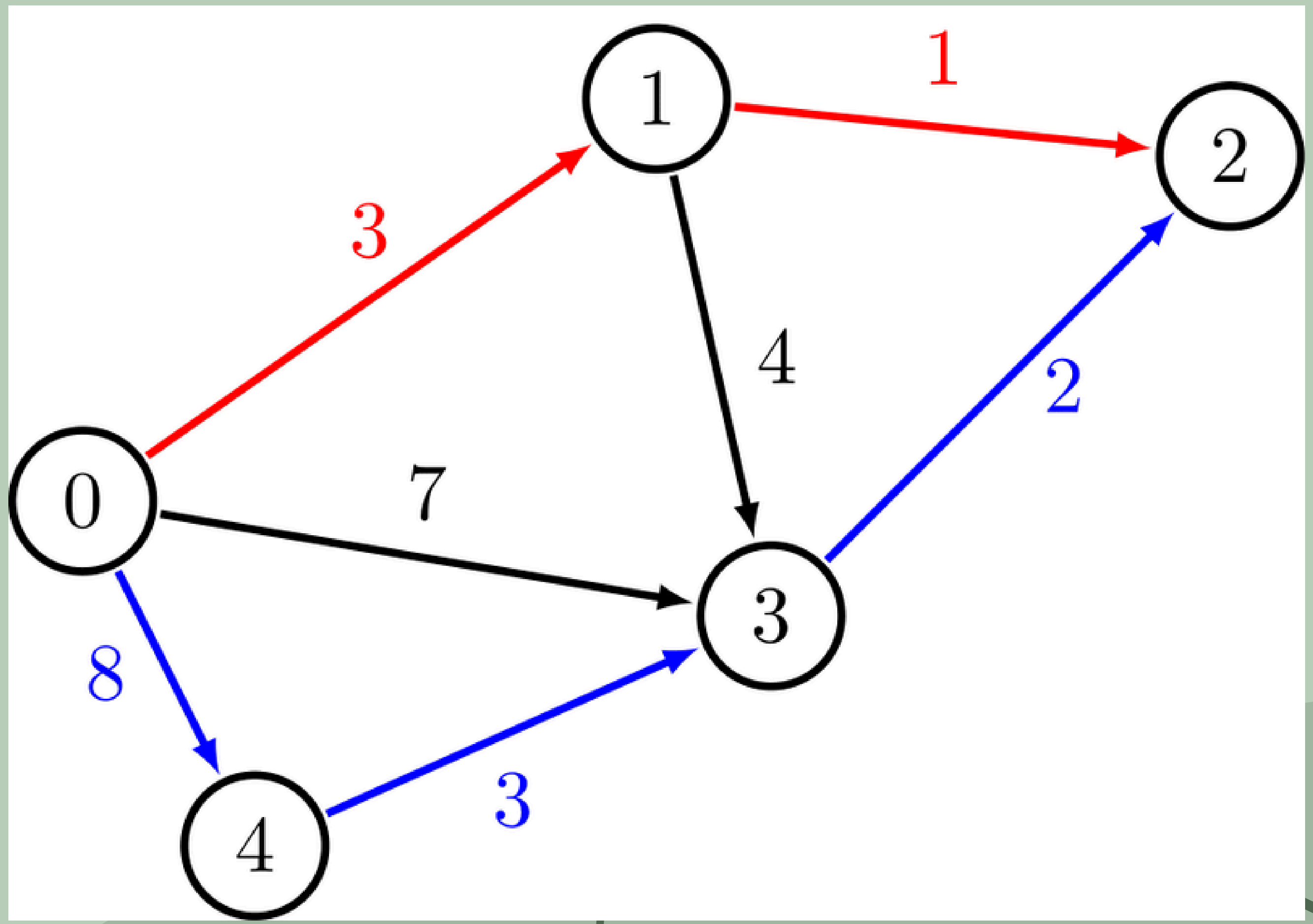
```
visited = set(start)
path = [start]
total = 0

next = None
weight = 0
while neighbors not visited:
      find the greatest neighbor for the current node

      add neighbor to visited
      add neighbor to path
      add weight to total
       next = neighbor

return total_weight, path
```

Exact Solution

Exact

How does it work?

Loops through each node as start and each node as end and runs dfs from start node to end node. Once all paths are found the one with the longest path is returned



Input

```
5 7
0 4 8
0 3 7
0 1 3
4 3 3
1 3 4
3 2 2
1 2 1
```

Output

```
13
0 4 3 2
```

Runtime

Big-O Runtime: O(V^2 * (V + E) + E)

```
def findLongest(graph):
    longest_path = None
    max_weight = 0
    for start in graph:
        for end in graph:
            paths = dfs(graph, end, start)
            for path, weight in paths:
                if weight > max_weight:
                    longest_path = path
                    max_weight = weight
    return longest_path, max_weight
```

$O(V^2)$ for main loop iterating over pairs of nodes in the graph

O(V + E) for DFS

O(E) for final loop over paths returned by DFS

Exact Pseudocode

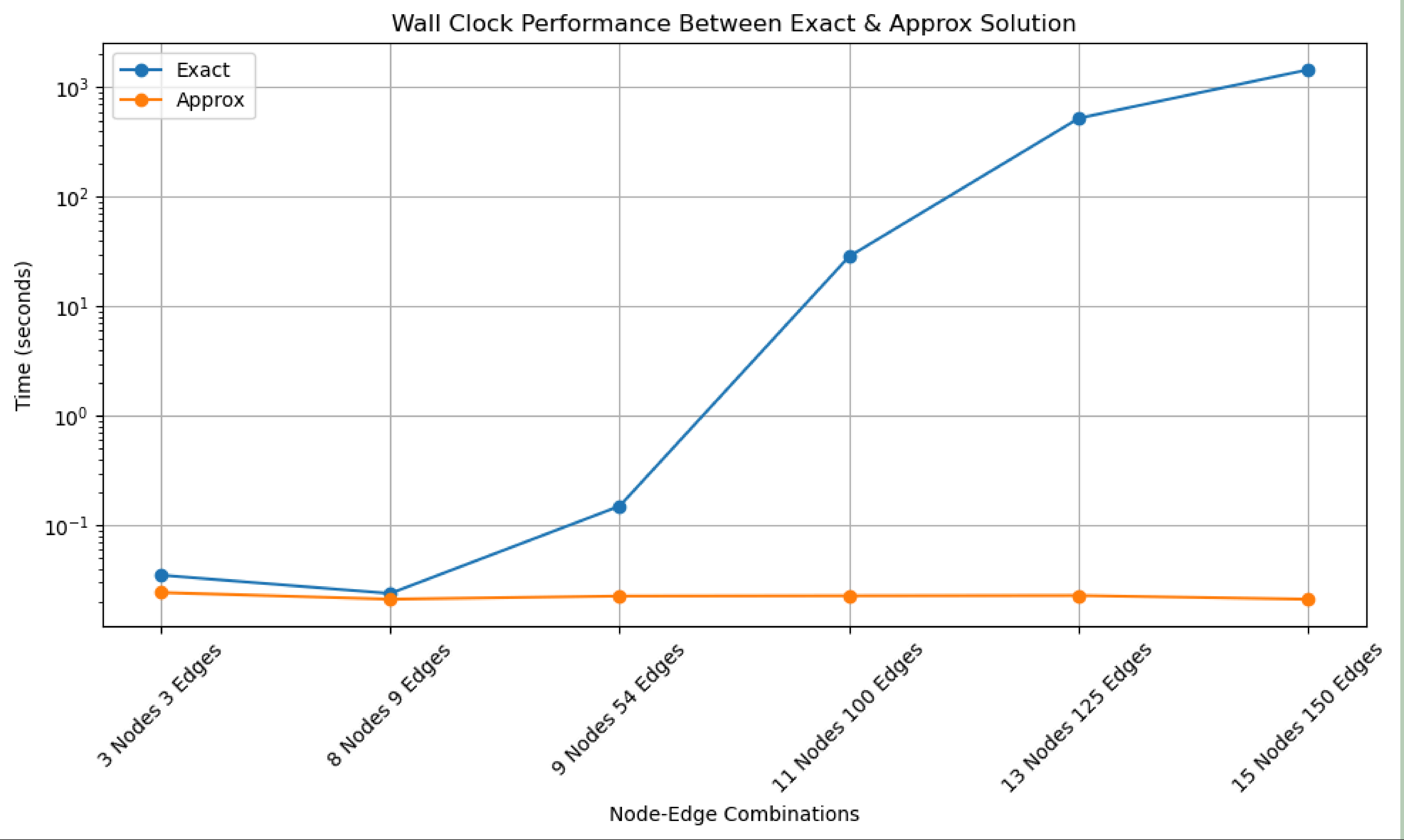
max_weight = 0

longest_path = None

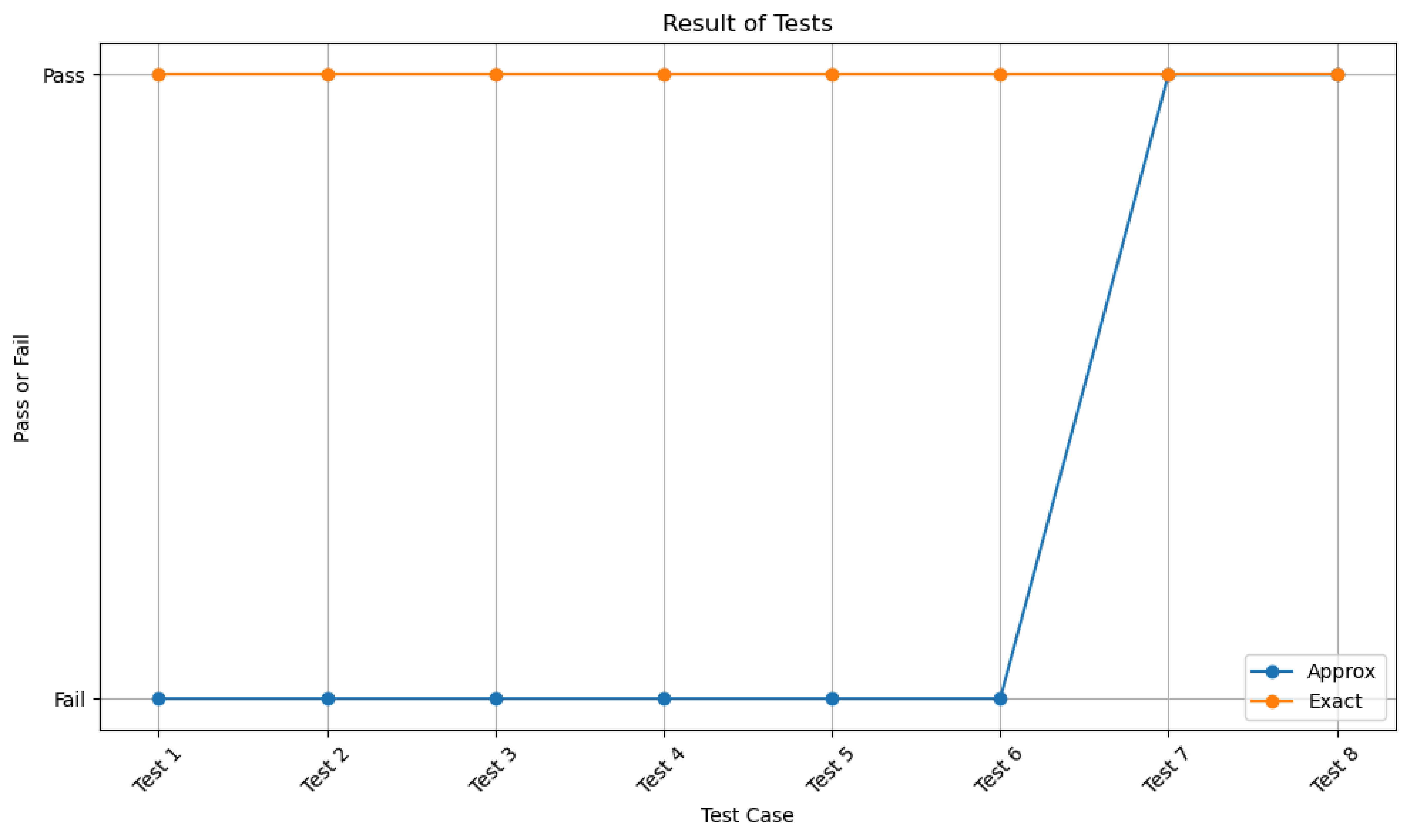
for every 2 nodes(start and end) in the graph

    perform DFS to find all paths from start to end node

    loop through every found path to update max_weight and longest_path if longer path is found



	Code Solution	3 Nodes 3 Edges	8 Nodes 9 Edges	9 Nodes 54 Edges	11 Nodes 100 Edges	13 Nodes 125 Edges	15 Nodes 150 Edges
0	Exact	0.035168	0.023893	0.150122	28.627655	523.230540	1453.948848
1	Approx	0.024355	0.021187	0.022625	0.022748	0.022866	0.021175



Example Passed Input

```
4 5
0 1 6
1 2 2
1 3 3
0 3 4
2 3 1
```

Example Approx Failed Input

```
3 3
0 1 3
1 2 4
0 2 5
```