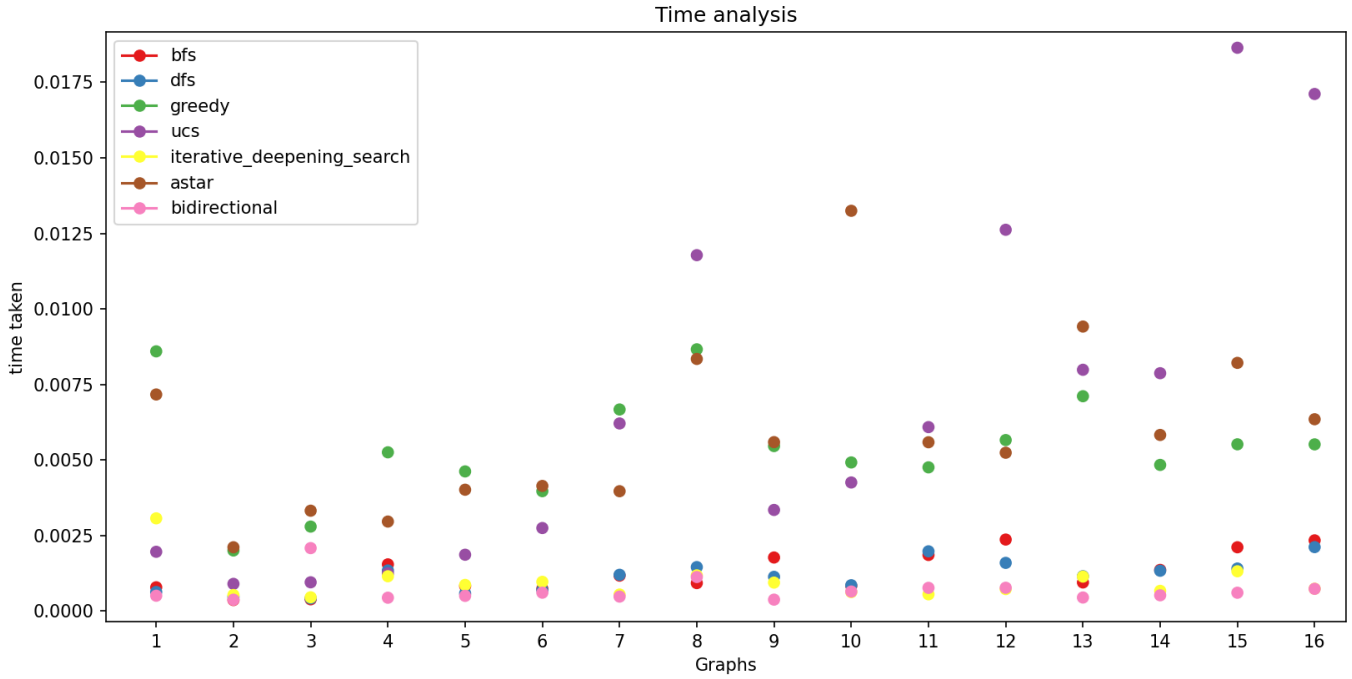
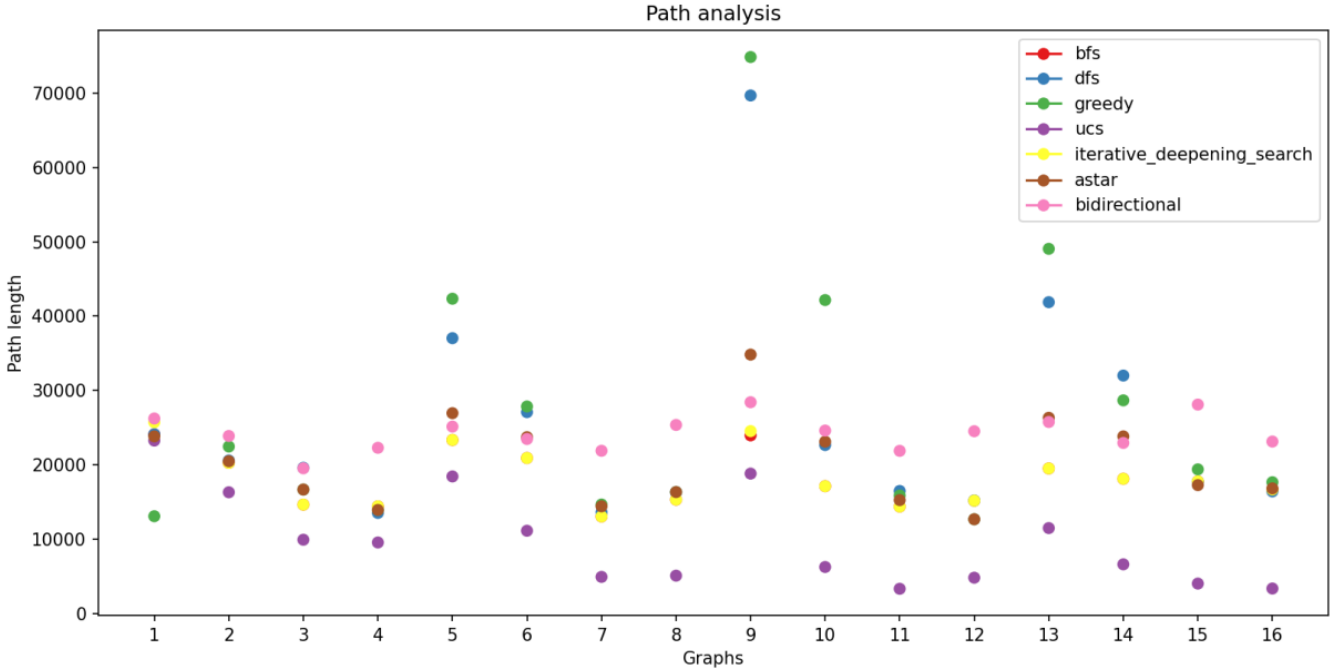

Time analysis
bfs
dfs
greedy
ucs
iterative_deepening_search
astar
bidirectional
time taken
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Graphs


Path analysis
bfs
dfs
greedy
ucs
iterative_deepening_search
astar
bidirectional
Path length
70000
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
Graphs

The first graph shows the average time taken to find the path among 10 random nodes with each other in 16 graphs using the 7 graph search algorithms given run 5 times and taking the average.

The second graph shows the length of the path each algorithm produce when they search to get the path between 10 random nodes.

Note: as the graph nodes, edge cost and location (latitude and longitude) are taken randomly they affect the cost and heuristic using functions like UCS & A* greatly.

First Graph (Time Analysis):

1. DFS (Depth-First Search):

- The DFS line illustrates how time consumption increases steadily with the depth of exploration, ideal for sparsely connected graphs.

2. BFS (Breadth-First Search):

- The BFS line reflects its consistent time consumption, making it suitable for scenarios where the search space is well-defined and limited.

3. UCS:

- UCS maintains a steady time consumption despite variations in edge density, thanks to its focus on minimizing the total cost to reach the goal.

4. A* Search:

- A* Search, while more time-consuming, balances heuristic guidance with computational cost, making it efficient for finding optimal paths in weighted graphs.

5. Bidirectional Search:

- The Bidirectional Search, not explicitly depicted, typically outperforms traditional searches by simultaneously exploring from both start and goal nodes.

6. Iterative Deepening Search:

- Iterative Deepening Search, though not visualized, gradually increases search depth, ensuring optimality without extensive memory usage.

7. Greedy Best-First Search:

- Greedy Best-First Search, not shown, rapidly explores paths based solely on heuristic values, sacrificing optimality for speed.

Second Graph (Path Analysis):

1. DFS (Depth-First Search):

- The DFS line overlaps initially with BFS, but as edges increase, it tends to produce longer paths due to its deep exploration nature.

2. BFS (Breadth-First Search):

- BFS focuses on nodes closer to the starting point, typically resulting in shorter paths compared to DFS.

3. UCS and A* Search:

- UCS and A* Search, exhibit similar path lengths due to their reliance on actual path costs in the absence of heuristic information. Further analysis might be required to distinguish their performance under different conditions.

4. Bidirectional Search:

- Bidirectional Search, extending from both start and goal nodes, tends to produce shorter paths by converging towards the midpoint of the graph.

5. Iterative Deepening Search:

- Iterative Deepening Search may result in paths of varying lengths as it incrementally explores deeper levels, possibly leading to longer paths compared to BFS but often more optimal than DFS.

6. Greedy:

- Greedy prioritizes exploration based solely on heuristic values, potentially leading to shorter paths but without guaranteed optimality.

3. Graph Centralities

The centralitites were calcutated based on the graph given as the cities of romania.

b) Comparison

Centrality measure	**Top Ranked Cities**
Degree Centrality	Sibiu, Bucharest
Closeness Centrality	Pitesti
Eigenvector Centrality	Riminicu Vilcea
Katz Centrality	Sibiu
Pagerank	Bucharest
Betweenness Centrality	Bucharest

c) Observation

Degree Centrality:

Sibiu and Bucharest have the highest number of connections, indicating that they are important nodes in terms of connectivity as they are connected to many other cities.

Closeness Centrality:

Pitesti has the highest closeness centrality, suggesting that it is the most centrally located city in terms of travel time or distance to other cities. Therefore, Pitesti is closer to all other nodes in the graph.

Eigenvalue Centrality:

Riminicu Vilcea has the highest eigenvector centrality, indicating that it is highly connected to more central nodes or cities that have high centrality scores.

Katz centrality:

Sibiu has more closer connections (immediate or otherwise) than farther ones. Katz centrality accounts for both the number of immediate neighbors a node has (like degree centrality) and the importance of those neighbors (like eigenvector centrality). It considers all paths between

nodes, giving more weight to shorter paths and discounting longer paths. Based on which Sibiu is ranked the top city.

PageRank:

Bucharest has the highest PageRank score meaning it has more popular and quality links to other cities based on google's analogy of page rank results of website links .

Betweenness Centrality:

The betweenness centrality depicts Bucharest as the most important city in the road network graph of Bucharest which makes sense given that Bucharest is its capital city. This centrality shows that Bucharest lies more often along shortest paths of other cities.

4. Comparison of the performance of different graph representations

Operation	**Adjacency List**	**Adjacency Matrix**	**Edge List**
Insertion of vertices and edges	O(1)	O(V^2)	O(1)
Deletion of vertices and edges	O(V + E)	O(V^2)	O(E)
Checking existence of an edge	O(V)	O(1)	O(E)
Finding neighbors of a vertex	O(1)	O(V)	O(E)

Representation	**Space Complexity**
Adjacency List	O(V + E)
Adjacency Matrix	O(V^2)
Edge List	O(E)

Edge list

Advantages:

- It uses less memory
- Easy to represent

Disadvantages:

- Edge look up is slow
- Hard to traverse

Adjacency list

Advantages:

- It uses less memory
- Neighbors of a node can be found pretty fast
- Best for sparse graphs

Disadvantages:

- Edge look up is slow

Adjacency matrix

Advantages:

- To represent dense graphs.
- Edge lookup is fast

Disadvantages:

- It takes more time to build and consume more memory (O(N**2) for both cases)
- Finding neighbors of a node is costly