

| **Title:** Implementation of uninformed search algorithms – BFS,DFS, DLS for the given problem |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Objective:** Examine the efficiency and performance of uninformed search algorithms in solving various problems **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Expected Outcome of Experiment:**

| **Course Outcome** | **After successful completion of the course students should be able to** |
| --- | --- |
| **CO2** | Analyse and solve problems for goal based agent architecture (searching and planning algorithms) |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. **“Artificial Intelligence: a Modern Approach” by Russell and Norving, Pearson education Publications**
2. **“Artificial Intelligence” By Rich and knight, Tata Mcgraw Hill Publications**
3. [**http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf**](http://people.cs.pitt.edu/~milos/courses/cs2710/lectures/Class4.pdf)
4. [**http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html**](http://cs.williams.edu/~andrea/cs108/Lectures/InfSearch/infSearch.html)
5. **http://www.cs.mcgill.ca/~dprecup/courses/AI/Lectures/ai-lecture02.pdf** [**http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf**](http://homepage.cs.uiowa.edu/~hzhang/c145/notes/04a-search.pdf)
6. [**http://wiki.answers.com/Q/Informed\_search\_techniques\_and\_uninformed\_search\_techniques**](http://wiki.answers.com/Q/Informed_search_techniques_and_uninformed_search_techniques)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

Problem solving, state-space trees, problem formulation, goal based agent architecture

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Historical Profile:**

### Problem-Solving Agent

A problem-solving agent is designed to find solutions to well-defined problems. This agent typically follows these steps:

1. **Formulate the Problem**: Define the initial state, goal state, and possible actions.
2. **Search for a Solution**: Use an appropriate search strategy to explore the problem space.
3. **Execute the Solution**: Apply the sequence of actions derived from the search.

### Uninformed Search Algorithms

Uninformed search algorithms, also known as blind search algorithms, are basic search strategies that explore the search space without any additional information about the goal's location beyond what is provided in the problem definition.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**New Concepts to be learned:**

Uninformed (blind) search.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Uninformed Searching Technique**

 B**readth-First Search (BFS)**:

* Explores all nodes at the present depth level before moving on to nodes at the next depth level.
* Complete and optimal if the cost is uniform.

 **Depth-First Search (DFS)**:

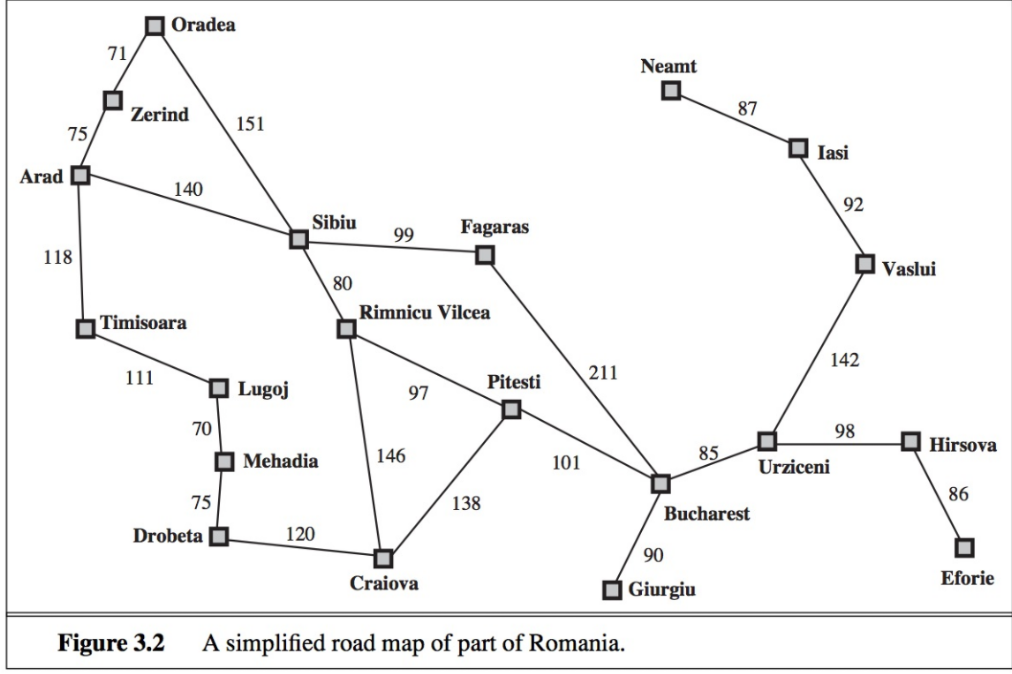
* Explores as far as possible along each branch before backtracking.
* Not complete or optimal, but requires less memory.

 **Depth-Limited Search (DLS)**:

* Depth-first search with a predetermined depth limit.
* Can overcome infinite path problems.

**Problem Statement:**

You are given a map of Romania with cities and the roads connecting them. Your task is to perform a Depth-First Search (DFS) starting from a given city and visit all the reachable cities in Romania. The goal is to simulate the DFS algorithm and return the order in which the cities are visited.



Constraints:

* The graph is bidirectional, meaning all cities are reachable from one another.
* The DFS should be implemented using a recursive approach.
* The input will always contain a valid start city.

**Solution with of chosen algorithm on the state-space tree:**

**CODE:**

**#include <bits/stdc++.h>**

**using namespace std;**

**unordered\_map<string, vector<string>> graph;**

**set<string> visited;**

**unordered\_map<string, string> parent;**

**vector<string> cities = {"Arad", "Zerind", "Oradea", "Sibiu", "Timisoara", "Lugoj", "Mehadia", "Drobeta", "Craiova", "Rimnicu Vilcea", "Fagaras", "Pitesti", "Bucharest", "Giurgiu", "Urziceni", "Hirsova", "Eforie", "Vaslui", "Lasi", "Neamt"};**

**void addEdge(const string &city1, const string &city2)**

**{**

**graph[city1].push\_back(city2);**

**graph[city2].push\_back(city1);**

**}**

**void printPath(const string &start, const string &goal)**

**{**

**vector<string> path;**

**string current = goal;**

**while (current != start && parent.find(current) != parent.end())**

**{**

**path.push\_back(current);**

**current = parent[current];**

**}**

**path.push\_back(start);**

**reverse(path.begin(), path.end());**

**for (const string &city : path)**

**{**

**cout << city << " ";**

**}**

**cout << endl;**

**}**

**bool dfs(const string &current, const string &start, const string &goal)**

**{**

**cout << "Visiting: " << current << endl;**

**if (current == goal)**

**{**

**cout << endl << "Goal found!" << endl << "Path: " ;**

**printPath(start, goal);**

**return true;**

**}**

**visited.insert(current);**

**for (const string &neighbor : graph[current])**

**{**

**if (visited.find(neighbor) == visited.end())**

**{**

**parent[neighbor] = current;**

**if (dfs(neighbor, start, goal))**

**return true;**

**}**

**}**

**return false;**

**}**

**int main()**

**{**

**addEdge("Arad", "Sibiu");**

**addEdge("Arad", "Zerind");**

**addEdge("Arad", "Timisoara");**

**addEdge("Zerind", "Oradea");**

**addEdge("Oradea", "Sibiu");**

**addEdge("Sibiu", "Fagaras");**

**addEdge("Sibiu", "Rimnicu Vilcea");**

**addEdge("Rimnicu Vilcea", "Pitesti");**

**addEdge("Rimnicu Vilcea", "Craiova");**

**addEdge("Craiova", "Drobeta");**

**addEdge("Drobeta", "Mehadia");**

**addEdge("Mehadia", "Lugoj");**

**addEdge("Lugoj", "Timisoara");**

**addEdge("Fagaras", "Bucharest");**

**addEdge("Pitesti", "Bucharest");**

**addEdge("Bucharest", "Giurgiu");**

**addEdge("Bucharest", "Urziceni");**

**addEdge("Urziceni", "Vaslui");**

**addEdge("Urziceni", "Hirsova");**

**addEdge("Hirsova", "Eforie");**

**addEdge("Vaslui", "Lasi");**

**addEdge("Lasi", "Neamt");**

**cout << "Edges in the Map of Romania: " << endl << endl;**

**for (auto &city : graph)**

**{**

**cout << city.first << " -> ";**

**for (string &neighbour : city.second)**

**{**

**cout << neighbour << " ";**

**}**

**cout << endl;**

**}**

**cout << endl;**

**cout << "Enter the start state: ";**

**string start;**

**cin >> start;**

**cout << "Enter the goal state: ";**

**string goal;**

**cin >> goal;**

**cout << endl;**

**int exists = -1;**

**for (int i = 0; i < cities.size(); ++i)**

**{**

**if (cities[i] == start)**

**exists++;**

**if (cities[i] == goal)**

**exists++;**

**}**

**if (!exists)**

**{**

**cout << "Start or Goal state not found!";**

**return 0;**

**}**

**vector<bool> visited(30, false);**

**if (!dfs(start, start, goal))**

**{**

**cout << "Goal not reachable from start." << endl;**

**}**

**return 0;**

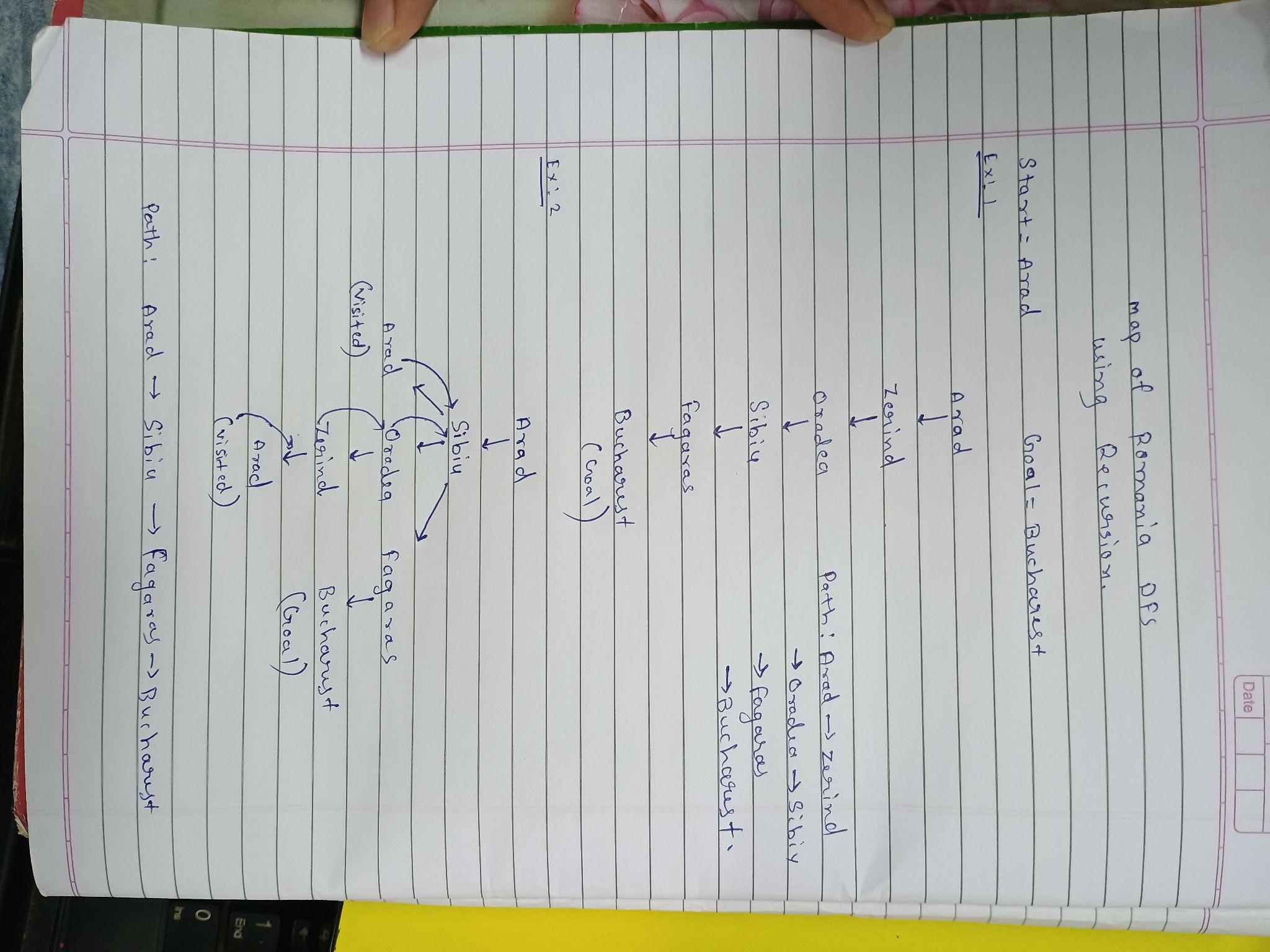
**}**

**OUTPUT:**

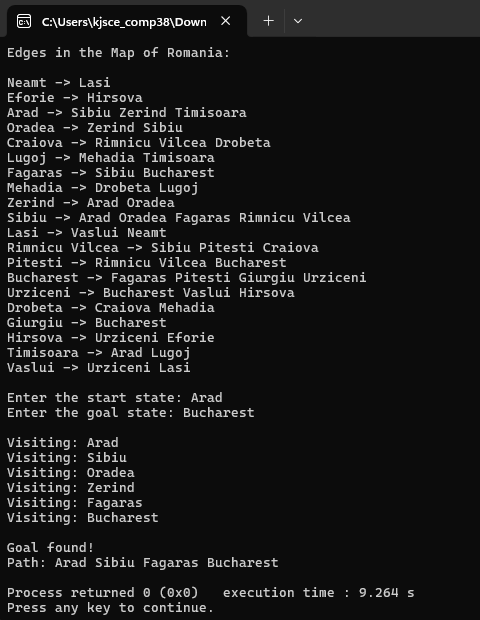
**1]**

****

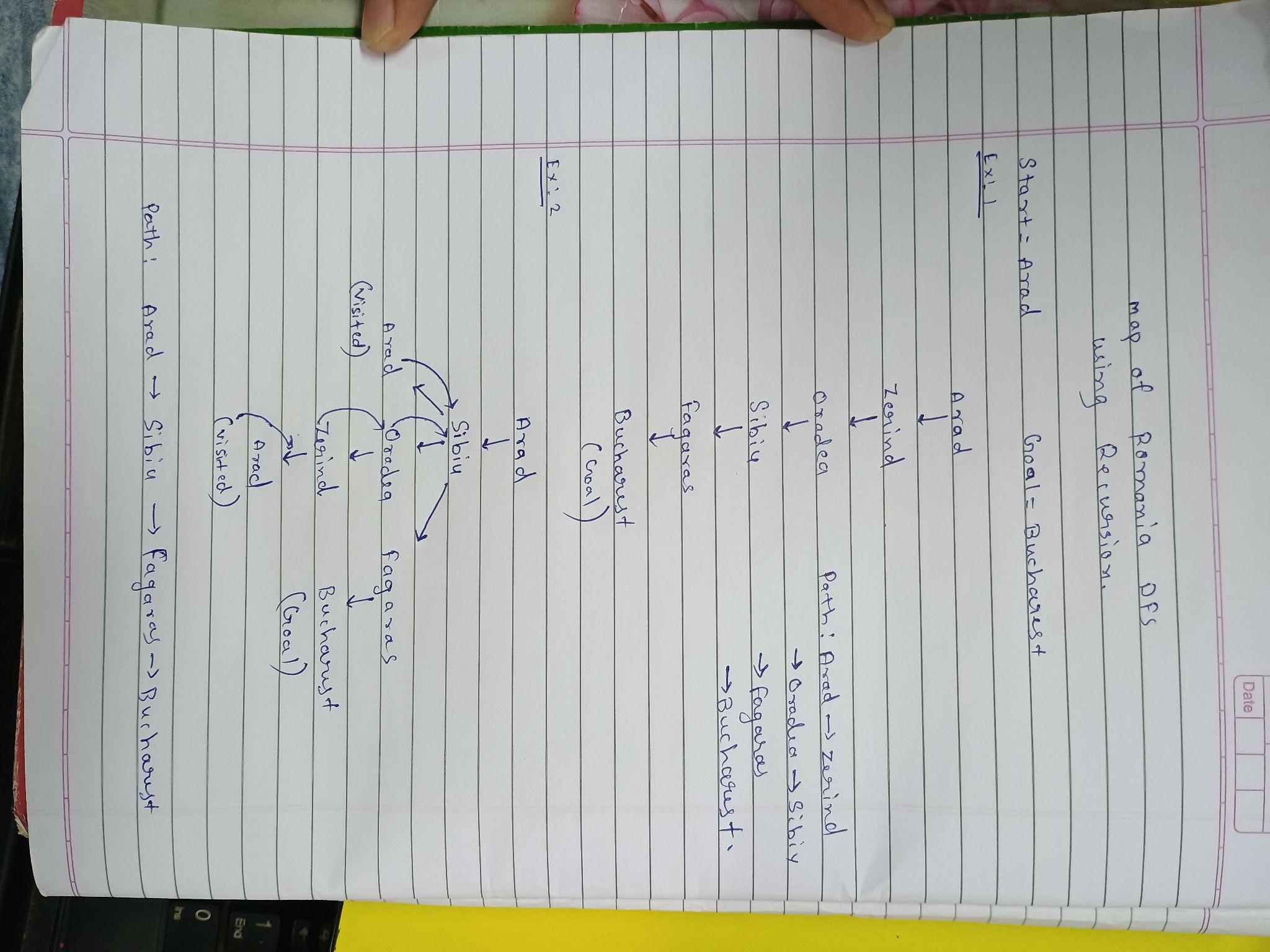
**State-Space Tree:**

****

**2]**

****

**State-Space Tree:**

****

**Post Lab Objective Questions**

1. **Which is not a Goal-based agent?**
2. Inference
3. Search
4. Planning
5. Conclusion
6. Dynamic search.

**Answer:** a. Inference

**2. Which were built in such a way that humans had to supply the inputs and interpret the outputs?**

1. Agents
2. Sensor
3. AI System
4. Actuators

**Answer:** c. AI System

**3. Which search algorithm imposes a fixed depth limit on nodes?**

* 1. Depth-limited search
  2. Depth-first search
  3. Iterative Deepening search
  4. Only (a) and (b)
  5. Only (a), (b) and (c).

**Answer:** d. Only (a) & (b)

**4. Optimality of BFS is**

* 1. When all step costs are equal
  2. When all step costs are unequal
  3. When there is less number of nodes
  4. Both a & c

**Answer:** a. When all step costs are equal

**5. What is a common application of Depth-First Search?**

* 1. Finding the shortest path
  2. Solving puzzles
  3. Web crawling
  4. Scheduling processes

**Answer:** c. Web Crawling

**Post Lab Subjective Questions:**

1. **Mention the criteria for the evaluation of search Algorithm.**

A search algorithm is evaluated based on the following four criteria:

* 1. **Completeness** – Does the algorithm always find a solution if one exists?
  2. **Optimality** – Does the algorithm guarantee the shortest or least-cost solution?
  3. **Time Complexity** – How much time does the algorithm take to find a solution in terms of nodes expanded?
  4. **Space Complexity** – How much memory does the algorithm consume during execution?

1. **State the properties of BFS,DFS, DLS.**

| **Property** | **BFS** | **DFS** | **DLS** |
| --- | --- | --- | --- |
| **Completeness** | Yes (if finite) | No (may get stuck in infinite depth) | Yes (if goal is within the limit) |
| **Optimality** | Yes (if uniform cost) | No | No |
| **Time Complexity** | *O(bd)* | *O(bd)* | *O(bd)* |
| **Space Complexity** | *O(bd)* (stores all nodes) | *O(d)* (stores path only) | *O(l)* (limited depth only) |
| **Best Use Case** | Finding the shortest path | Exploring deep structures | When depth is known |
| **Disadvantage** | High memory usage | Can get stuck in infinite depth | May miss solutions beyond the limit |

**Where:**

* **b =** Branching factor (average number of child nodes per node)
* **d =** Depth of the shallowest goal
* **l =** Depth limit set for DLS