K. J. Somaiya College of Engineering, Mumbai-77

Batch: A-4	**Roll No.:** 16010122151
Experiment / assignment / tutorial No. 4	
Grade: AA / AB / BB / BC / CC / CD /DD	
Signature of the Staff In-charge with date	

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**
4. **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**
6. **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:

DLS - MAP OF ROMANIA

You are given a map of Romania with cities connected by roads. Using Depth-Limited Search (DLS), find a path from a starting city to a destination city, but limit your search to a specified depth.

Input:

1. A list of cities and roads between them (adjacency list).
2. A starting city.
3. A destination city.
4. A depth limit.

Output:

- The path from the starting city to the destination city if reachable within the given depth.
- If the destination city is unreachable within the depth limit, return "No path found."

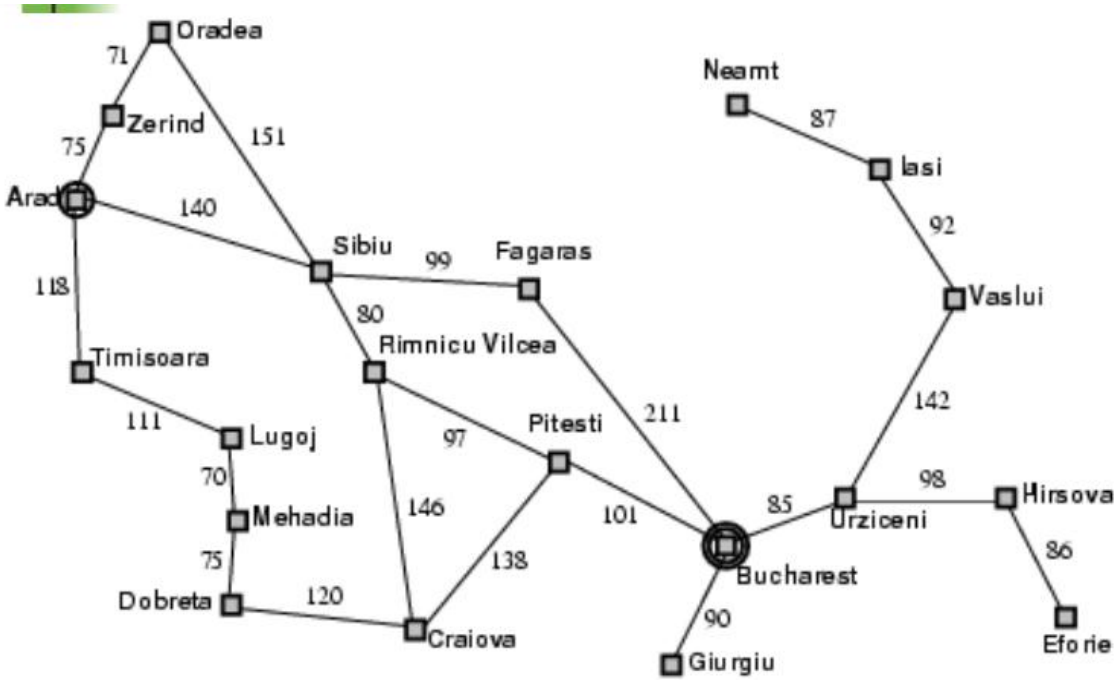
Example:

- Cities: Bucharest, Arad, Timisoara, Craiova, Pitesti, Sibiu
- Roads: Bucharest → Pitesti, Pitesti → Craiova, etc.
- Start City: Bucharest
- Goal City: Craiova
- Depth Limit: 3

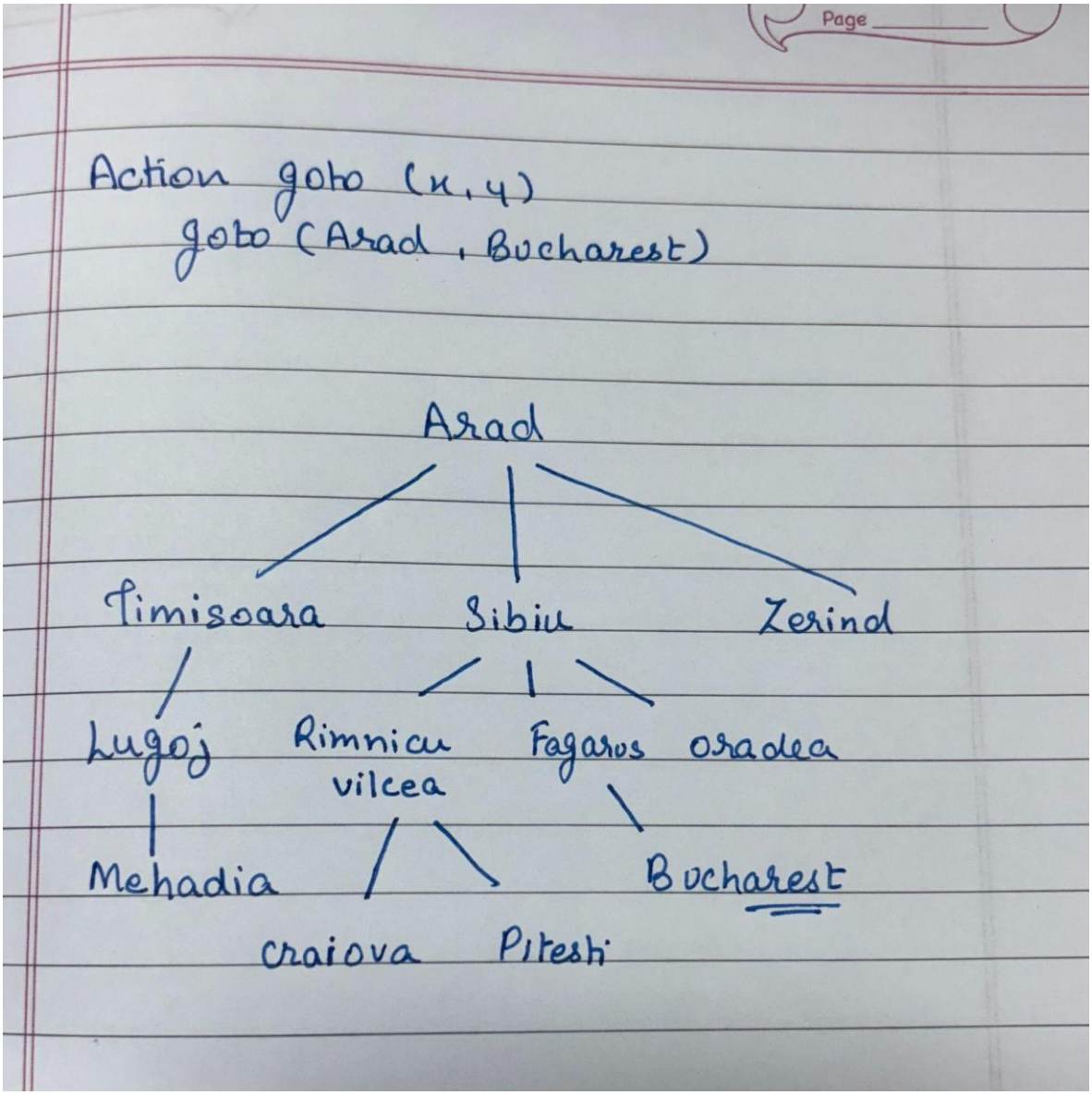
Output:
Path: Bucharest → Pitesti → Craiova

If no path is found within the limit, return "No path found."

State-space tree :



Solution of chosen algorithm on the state-space tree: *(Attach diagrams/solutions here)*



Source code for the assigned problem with assigned algorithm

```
#include<iostream>
#include<unordered_map>
#include<vector>
using namespace std;

unordered_map<string, vector<pair<string, int>>> romaniaMap = {
    {"Arad", {{"Zerind", 75}, {"Timisoara", 118}, {"Sibiu", 140}}},
    {"Zerind", {{"Oradea", 71}, {"Arad", 75}}},
    {"Oradea", {{"Zerind", 71}, {"Sibiu", 151}}},
    {"Timisoara", {{"Arad", 118}, {"Lugoj", 111}}},
    {"Lugoj", {{"Timisoara", 111}, {"Mehadia", 70}}},
    {"Mehadia", {{"Lugoj", 70}, {"Dobreta", 75}}},
    {"Dobreta", {{"Mehadia", 75}, {"Craiova", 120}}},
    {"Craiova", {{"Dobreta", 120}, {"Rimnicu Vilcea", 146}, {"Pitesti",
138}}},
    {"Sibiu", {{"Arad", 140}, {"Oradea", 151}, {"Fagaras", 99},
{"Rimnicu Vilcea", 80}}},
    {"Fagaras", {{"Sibiu", 99}, {"Bucharest", 211}}},
    {"Rimnicu Vilcea", {{"Sibiu", 80}, {"Craiova", 146}, {"Pitesti",
97}}},
    {"Pitesti", {{"Rimnicu Vilcea", 97}, {"Craiova", 138},
{"Bucharest", 101}}},
    {"Bucharest", {{"Fagaras", 211}, {"Pitesti", 101}, {"Giurgiu", 90},
{"Urziceni", 85}}},
    {"Giurgiu", {{"Bucharest", 90}}},
    {"Urziceni", {{"Bucharest", 85}, {"Vaslui", 142}, {"Hirsova",
98}}},
    {"Hirsova", {{"Urziceni", 98}, {"Eforie", 86}}},
    {"Eforie", {{"Hirsova", 86}}},
    {"Vaslui", {{"Urziceni", 142}, {"Iasi", 92}}},
    {"Iasi", {{"Vaslui", 92}, {"Neamt", 87}}},
    {"Neamt", {{"Iasi", 87}}}
};

bool limitcheck(string start,string end,int limit, vector<string>&path,
int &cost)
{
    path.push_back(start);
    if (start==end)
    {
        return true;
    }
    if (limit==0)
    {
        path.pop_back();
```

```
        return false;
    }

    for (auto next : romaniaMap[start])
    {
        cost+=next.second;
         if (limitcheck(next.first,end,limit-1,path,cost))
  {
     return true;
  }
  else
  {
  cost-=next.second;
  }
    }
   path.pop_back();
    return false;

}

int main ()
{
    string start="Arad",end ;
    int limit;

    cout <<"Enter Last Stop";
    cin >>end;

     cout << "Enter depth limit: ";
    cin >> limit;

    vector<string>path;
    int cost=0;
  if (limitcheck(start,end,limit,path,cost))
  {
      for ( auto &i:path)
      {
          cout<<i<<"->";
      }
      cout <<"end"<<endl;
      cout<<"TotalCost:-"<<cost;
  }
  else
  {
      cout <<"Path Not Possible." ;
  }
```

```
}
```

Output screenshots:

```
hyder@HyderPresswala MINGW64 ~/Downloads/w
$ g++ -o hyder hyder.cpp

hyder@HyderPresswala MINGW64 ~/Downloads/w
$ ./hyder.exe
Enter Last StopBucharest
Enter depth limit: 3
Arad->Sibiu->Fagaras->Bucharest->end
TotalCost:-450
hyder@HyderPresswala MINGW64 ~/Downloads/w
$
```

```
hyder@HyderPresswala MINGW64 ~/Downloads/w
$ ./hyder.exe
Enter Last StopBucharest
Enter depth limit: 2
Path Not Possible.
```

Comparison of performance of uninformed Algorithm:

DLS: DLS is useful when you know or can set a reasonable depth limit. It performs well in terms of memory, but its performance is sensitive to the depth limit. If the limit is too small, it won't find the solution. If the limit is too large, it becomes inefficient.

DFS: DFS may perform similarly to DLS but can be less predictable. It can easily get stuck in deep or irrelevant branches, and there's no guarantee it will find the shortest path.

BFS: BFS is the most reliable for finding the shortest path but suffers from high memory usage, especially in large or complex search spaces like the Map of Romania.

Conclusion:- Learned about uniformed Search strategies DLS Specifically.

Post Lab Objective Questions

1. Which is not a Goal-based agent?

a. Inference
b. Search
c. Planning
d. Conclusion
e. Dynamic search.

Answer: Inference (a)

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

a. Agents
b. Sensor
c. AI System
d. Actuators

Answer: AI System (c)

Post Lab Subjective Questions
Post Lab Objective questions

1. **Which search algorithm imposes a fixed depth limit on nodes?**
    a. Depth-limited search
    b. Depth-first search
    c. Iterative Deepening search
    d. Only (a) and (b)
    e. Only (a), (b) and (c).

Answer: Depth-limited search (a)

2. **Optimality of BFS is**
    a. When all step costs are equal
    b. When all step costs are unequal
    c. When there is less number of nodes
    d. Both a & c

Answer: When all step costs are equal (a)

3. What is a common application of Depth-First Search?
    a. Finding the shortest path
    b. Solving puzzles
    c. Web crawling
    d. Scheduling processes

Answer: Solving puzzles (b)

Post Lab Subjective Questions:

1. **Provide a real-world example where BFS would be preferable over DFS, and another where DFS would be a better choice. Explain your reasoning.**

    BFS is preferable in scenarios where you need to find the **shortest path**, such as navigating a city grid or map where all roads are of equal distance. Since BFS explores all nodes level by level, it guarantees that the shortest path to the goal will be found first when all steps have the same cost. On the other hand, DFS is more useful in scenarios where you need to **explore all possible paths** deeply before backtracking, such as in solving puzzles like the 8-puzzle or

navigating a maze. DFS doesn't prioritize the shortest path but is useful when you need to find any solution, especially when memory efficiency is crucial since it doesn't store all nodes in memory at once.

2. **If your search algorithm did not find a solution, what could be the possible reasons? How would adding a depth limit to DFS affect the outcome?**

   If a search algorithm doesn't find a solution, possible reasons include the goal being unreachable, an inadequate search space representation, or an incorrect algorithm choice. For instance, if you're using DFS for a shortest-path problem where BFS would be more appropriate, DFS might not yield the correct or optimal solution. Additionally, if there are infinite paths or cycles in the search space and the algorithm doesn't handle them properly, it could also fail to find a solution. Adding a depth limit to DFS helps prevent infinite loops by restricting how far the algorithm can explore, especially in cyclic graphs. However, if the solution lies beyond the set depth limit, DFS will terminate without finding it, causing potential failures.

3. **If you modified BFS to use a priority queue instead of a regular queue, how would this change the search behavior? What kind of search algorithm would it resemble?**

   If you modify BFS to use a priority queue instead of a regular queue, the search behavior would change significantly. Rather than exploring nodes level by level, the priority queue would prioritize nodes based on a cost or heuristic, expanding the least costly (or most promising) paths first. This modification would make the search resemble Uniform Cost Search (UCS) or *A search**. Both UCS and A* use a priority queue to expand nodes in order of their accumulated cost or heuristic value, making them more efficient in finding optimal solutions when dealing with costs or complex goal criteria.