National University of Computer & Emerging Sciences

Lab Manual

CS461: Artificial Intelligence Lab

Course Instructor	Dr. Hafeez-Ur-Rehman
Lab Instructor	Muhammad Hamza
Semester	Spring 2022

Un-Informed Searches

Problem Formulation

A search problem as **4 core components:**

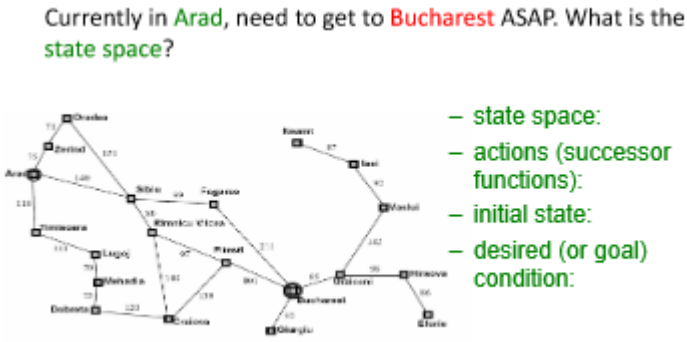
- **STATE SPACE** — the space over which to search. This involves abstracting the real problem
- **INITIAL STATE** — represents the agent's current state
- **GOAL STATE** — the desired outcome
- **ACTIONS** — the possible moves that allow the agent to get from the initial to the goal state

Optional Components:

- **COSTS** — what costs of moving from moving from each state (making an action)
- **HEURISTICS** — guides of the search processes

Solution

A solution is the algorithm that can transform your current state to the goal state.



Example of a search problem from **Arad** to **Bucharest**
In the above depiction the state space is all the available cities, the actions are moving from one city to the next, the initial state is Arad and the goal is Bucharest.

Representation

Search problems can be represented as graphs from one node to the next with vertices and edges. They can also be represented as a tree, with attributes of depth, branching factor where the same state could be represented many times. Complex probabilistic situations apply a probability to each state under uncertainty to get to the desired state.

Properties of Search

We can evaluate a search algorithm in the following ways:

- **Completeness** — whether the search will find a solution
- **Optimality** — when the actions have costs, what is the costs of the algorithm
- **Time Complexity** — the number of nodes that can be expanded or generated
- **Space Complexity** — number of nodes that must be stored in memory

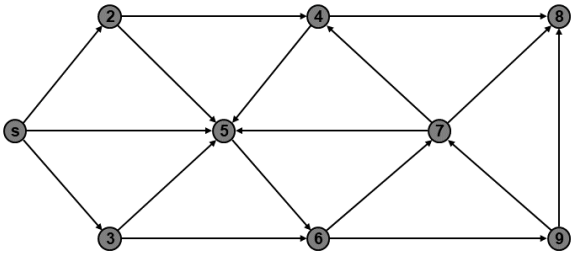
Uninformed Search Algorithms

Uninformed search is a class of general-purpose search algorithms which operates in brute force-way. Uninformed search algorithms do not have additional information about state or search space other than how to traverse the tree/graph, so it is also called blind search.

Following are the various types of uninformed search algorithms:

1. **Breadth-first Search**: Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadthwise in a tree or graph, so it is called breadth-first search.

- BFS algorithm starts searching from the root node of the tree and expands all successor node at the current level before moving to nodes of next level.
- The breadth-first search algorithm is an example of a general-graph search algorithm.
- Breadth-first search implemented using FIFO queue data structure.



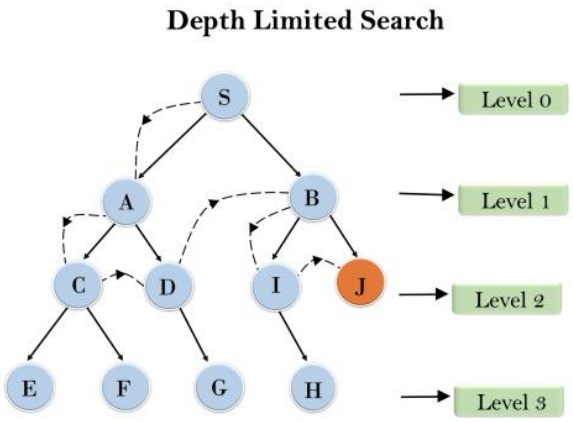
2. **Depth-first Search**: Depth-first search is a recursive algorithm for traversing a tree or graph data structure.

- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation. (LIFO)
- The process of the DFS algorithm is like the BFS algorithm.

Pseudo Code

```
DFS(G,v)   ( v is the vertex where the search starts )
      Stack S := {};   ( start with an empty stack )
      for each vertex u, set visited[u]:= false;
      push S, v;
      while (S is not empty) do
        u := pop S;
        if (not visited[u]) then
          visited[u] := true;
          for each unvisited neighbour w of u
            push S, w;
        end if
      end while
    END DFS()
```

3. **Depth-limited Search**
   A combination of the above searches where your first iterate a single branch as in depth first search, but only go to a specific depth. Truncate the search by only looking at paths d+1.Solves the infinite path length problem, however, the solution must come before the depth to be complete. Depth-limited search can be terminated with two Conditions of failure:

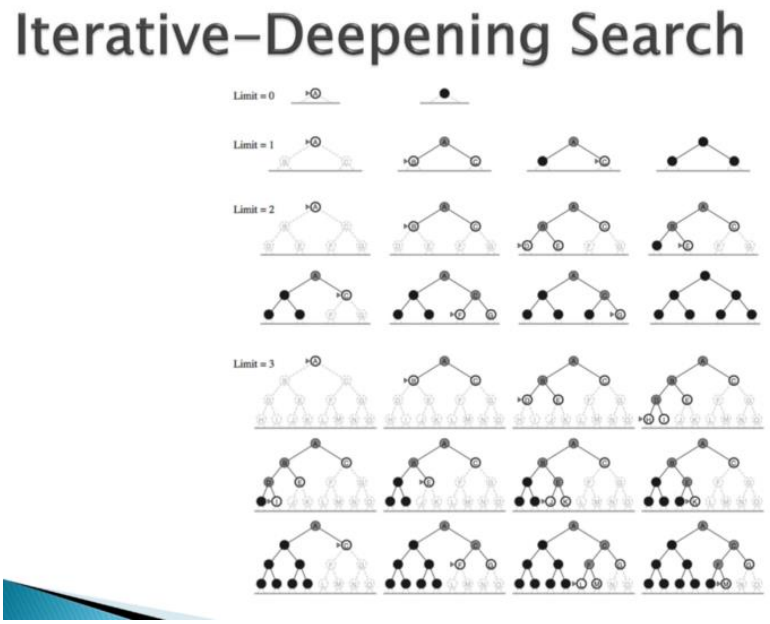


Standard failure value: It indicates that problem does not have any solution.

- **Cutoff failure value**: It defines no solution for the problem within a given depth limit.

4. **Iterative deepening depth-first search**

- Iterative deepening search is an extension of the depth limited search. Starting at the depth limit, you iteratively increase the depth until a solution is found or it has failed. If no nodes were cut off in this search, then it has exhausted all available paths.
- It is also complete as it finds a path is a solution exists at the iteratively defined depth, else it is not.
- If the costs are uniform than it will find the optimal path. You can provide a more optimal solution with costs by setting a cost bound of less than the cost for a path and expand.



Lab Task

1. Implement a Recursive Depth Limited Search function to find a specific value in a given tree. You can use the BST implementation provided in BST.py file to construct a BST.
2. Implement Iterative Deepening Search function to search a value in a given graph. Use the below given code to construct a graph.

```
# key represents a node while list in the value represent
# the child nodes
myGraph = {

    1: [2,3,5],
    2: [4,5],
    3: [5,6],
    5: [],
    6: [7,9],
    4: [5,8],
    7: [4,5,8],
    8: [],
    9: [7,8]}
```