Artificial Intelligence Lab

AL-2002

Instructor: Muhammad Saood Sarwar

Semester: Spring 2024

Artificial Intelligence Lab 03

Objective

The objective of this lab is to introduce the concept of search problems, and to teach the students how to formulate a search problem and use uninformed search algorithms such as Breadth-First Search (BFS) and Depth-First Search (DFS) to solve them.

Learning Outcomes

1. Understand the components of a search problem, including the state space, initial state, goal state, actions, costs, and heuristics.
2. Formulate a search problem by defining its components and represent it as a graph or tree.
3. Understand the properties of search algorithms, including completeness, optimality, time complexity, and space complexity.
4. Use BFS and DFS algorithms to solve search problems.

Table of Contents

Uninformed Searches

Problem Formulation

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

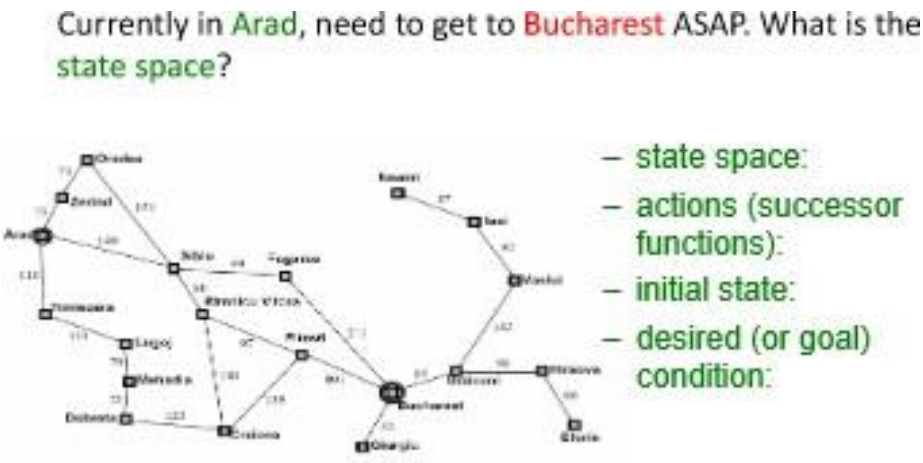
- **STATE SPACE** — the space over which to search. This involves abstracting the real problem.
- **INITIAL STATE** — represents the agent's current state.
- **GOAL STATE** — the desired outcome
- **ACTIONS** — the possible moves that allow the agent to get from the initial to the goal state.

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 have to be stored in memory.

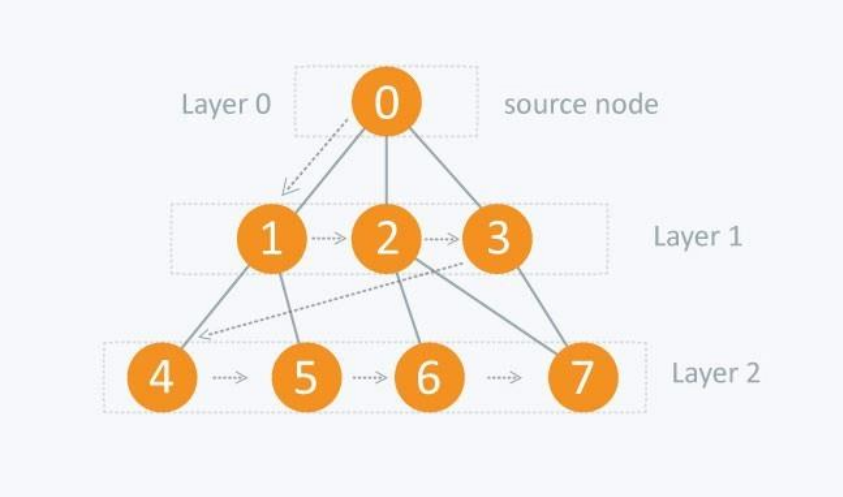
Uninformed Search Algorithms

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:

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.



Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

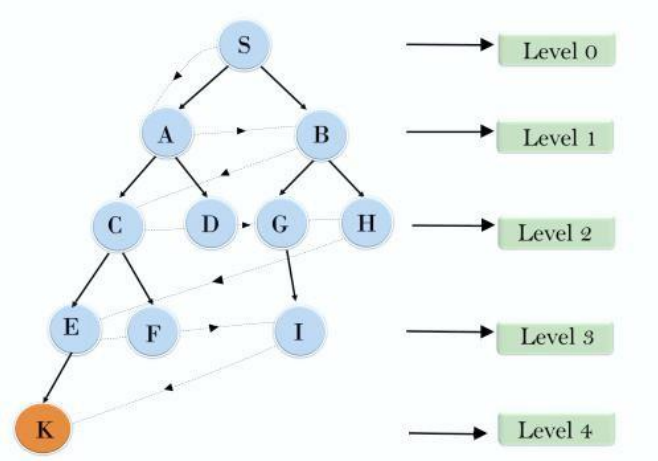
Disadvantages:

- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverse in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S---> A--->B---->C--->D---->G--->H--->E---->F---->I >K



Time Complexity: Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node. Where the d= depth of shallowest solution and b is a node at every state.

T (b) = 1+b2+b3+..... + bd= O (bd)

Space Complexity: Space complexity of BFS algorithm is given by the Memory size of frontier which is O(bd).

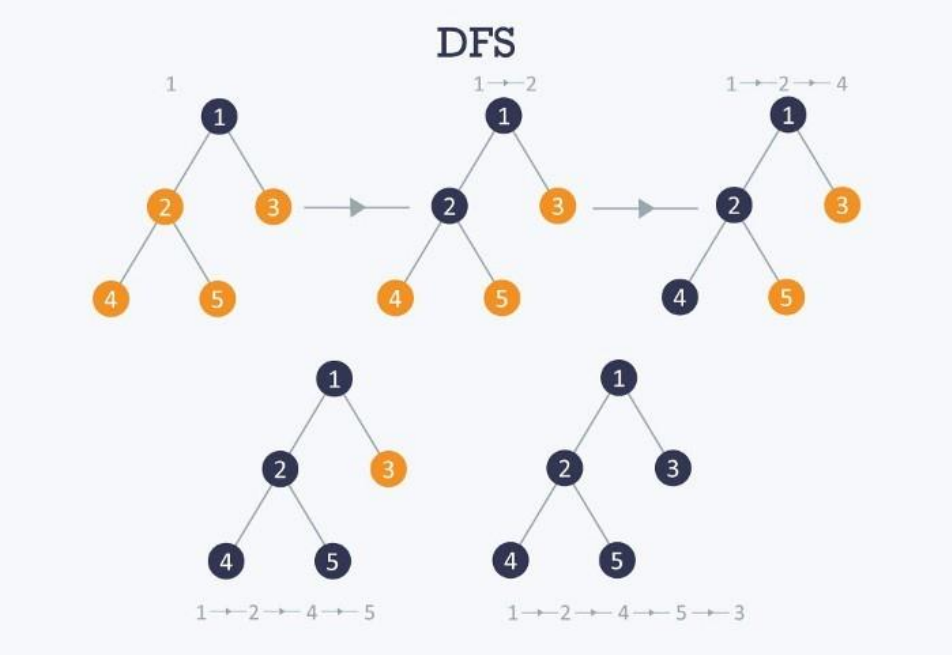
Completeness: BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality: BFS is optimal if path cost is a non-decreasing function of the depth of the node.

Depth-first Search

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

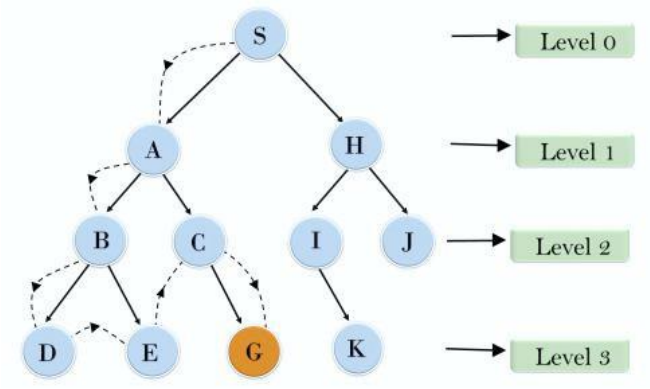
- It is called the depth-first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses a stack data structure for its implementation. (LIFO)
- The process of the DFS algorithm is like the BFS algorithm.



Example:

In the below search tree, we have shown the flow of depth-first search, and it will follow the order as:

Root node--->Left node ----> right node.



It will start searching from root node S, and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Completeness: DFS search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

T(n)= 1+ n2+ n3 +..... + nm=O(nm)

Where, m= maximum depth of any node and this can be much larger than d (Shallowest solution depth)

Space Complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is O(bm).

Optimal: DFS search algorithm is non-optimal, as it may generate many steps orhigh cost to reach to the goal node.

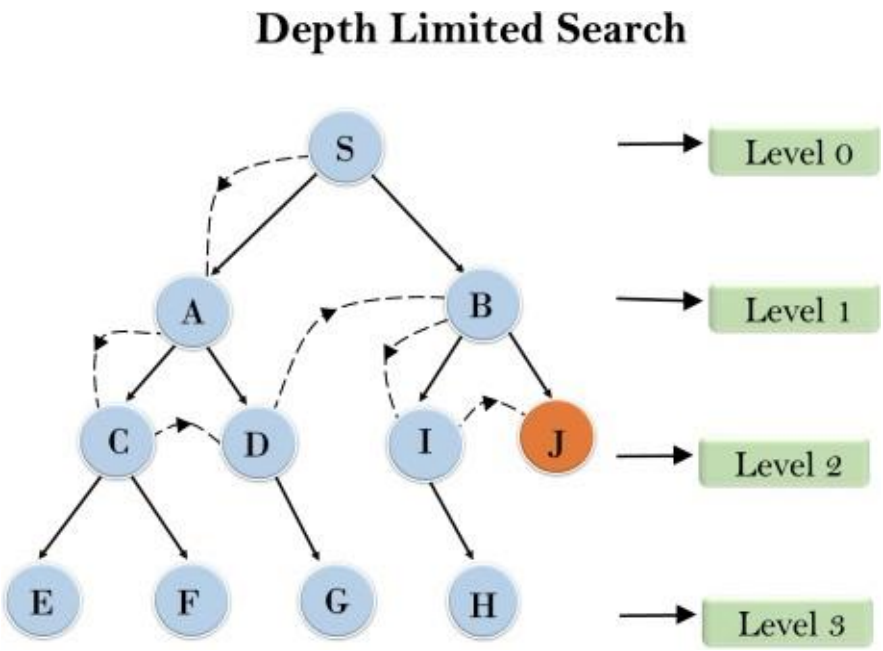
Depth-limited Search

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.

Example:



Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

Time Complexity: Time complexity of DLS algorithm is O(bℓ).

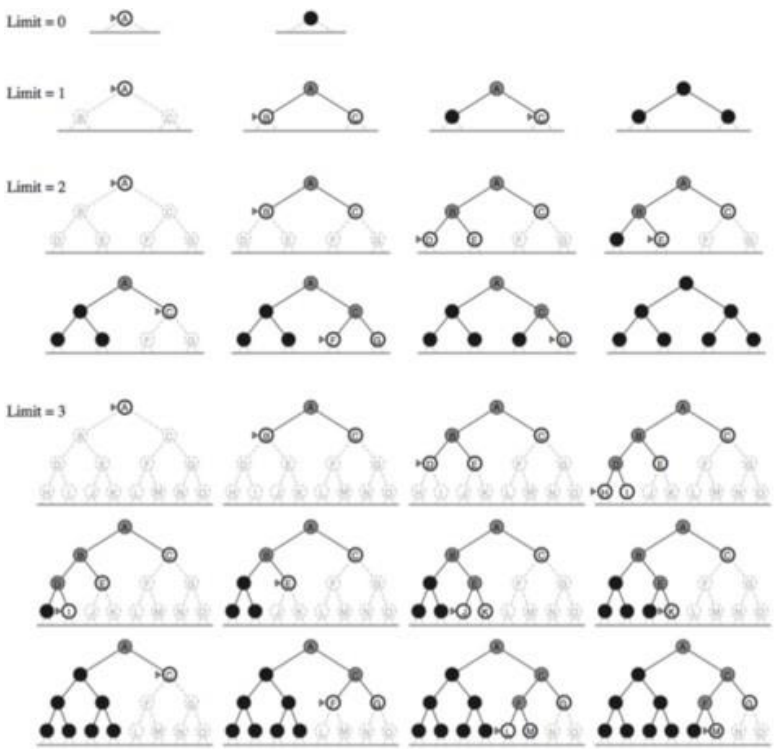
Space Complexity: Space complexity of DLS algorithm is O(b×ℓ).

Optimal: Depth-limited search can be viewed as a special case of DFS, and it is also not optimal even if ℓ>d.

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.



Advantages:

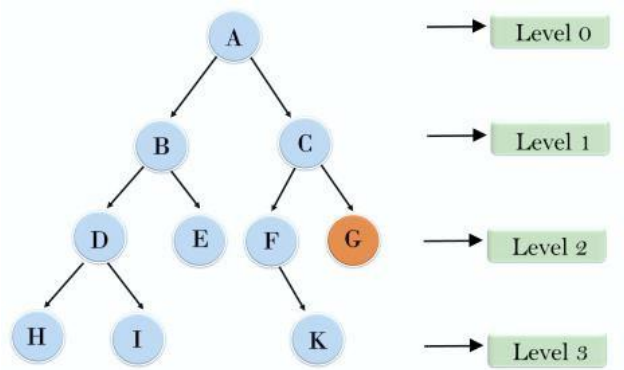
- It combines the benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency.

Disadvantages:

- The main drawback of IDDFS is that it repeats all the work of the previous phase.

Example:

Following tree structure is showing the iterative deepening depth-first search. IDDFS algorithm performs various iterations until it does not find the goal node.



The iteration performed by the algorithm is given as:

- 1'st Iteration ---- > A

- 2'nd Iteration --- > A, B, C
- 3'rd Iteration------>A, B, D, E, C, F, G
- 4'th Iteration----- >A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Completeness: This algorithm is complete is if the branching factor is finite.

Time Complexity: Let's suppose b is the branching factor and depth is d then the worst-case time complexity is O(bd).

Space Complexity: The space complexity of IDDFS will be O(bd).

Optimal: IDDFS algorithm is optimal if path cost is a non- decreasing function of the depth of the node.