National University of computer and emerging sciences

CL461-INTRODUTION TO ARTIFICIAL INTELLIGENCE

Lab 05

Search Problems in Artificial Intelligence

UN-Informed Searches

Problem Formulation

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

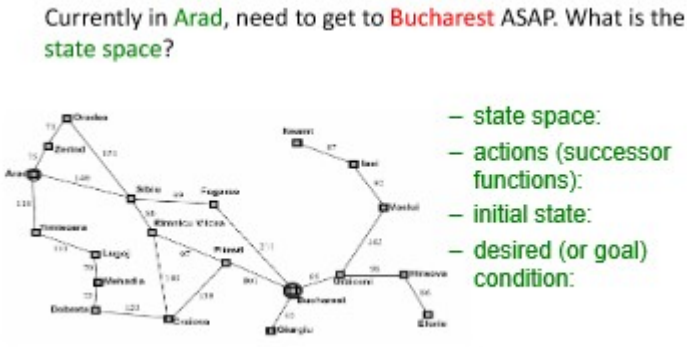
- **STATE SPACE** — the space over which to search. This involves abstracting the real problem
- **INITIAL STATE** — represents the agents current state
- **GOAL STATE** — the desired outcome
- **ACTIONS** — the possible moves that allow the agent to get from the initial to the goal state

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.

CL461-INTRODUTION TO ARTIFICIAL INTELLIGENCE

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:

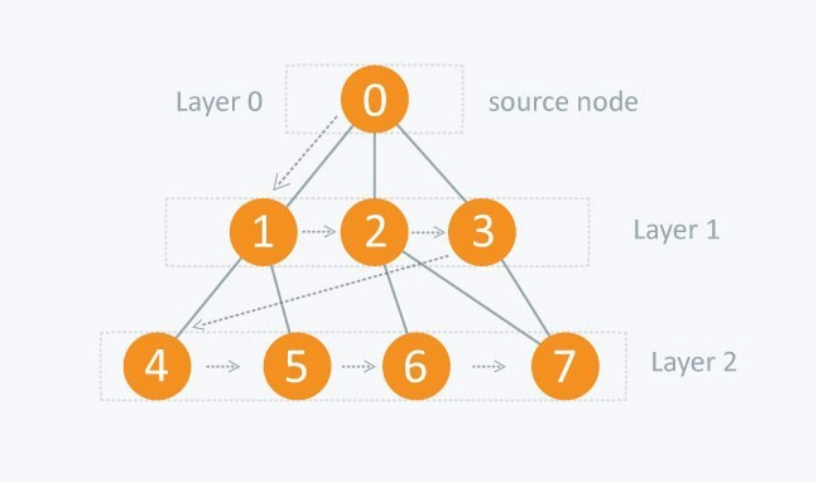
- **Completeness** — whether the search will find a solution
- **Optimality** — when the actions have costs, what is the costs of the algorithm
- **Time Complexity** — the number of nodes that can be expanded or generated
- **Space Complexity** — number of nodes that have to be stored in memory

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.

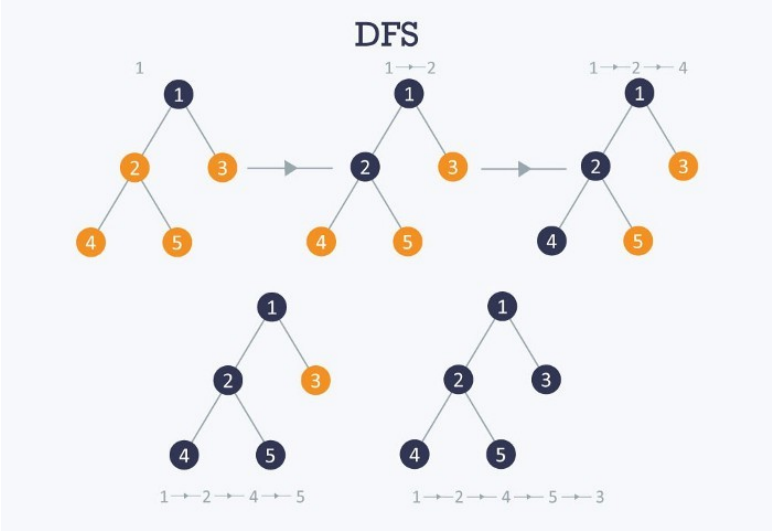
Following are the various t**ypes** of uninformed search algorithms:

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 similar to the BFS algorithm.

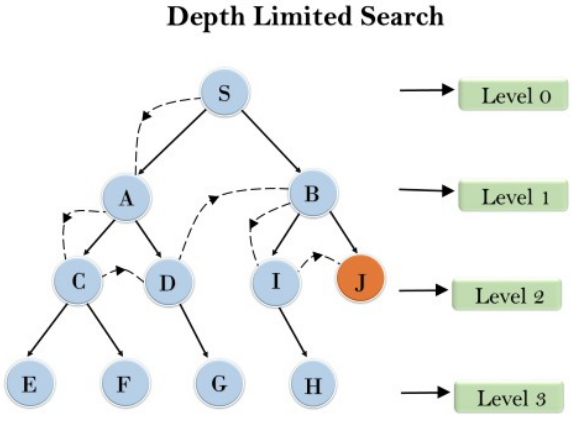


3. **Depth-limited Search**

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.

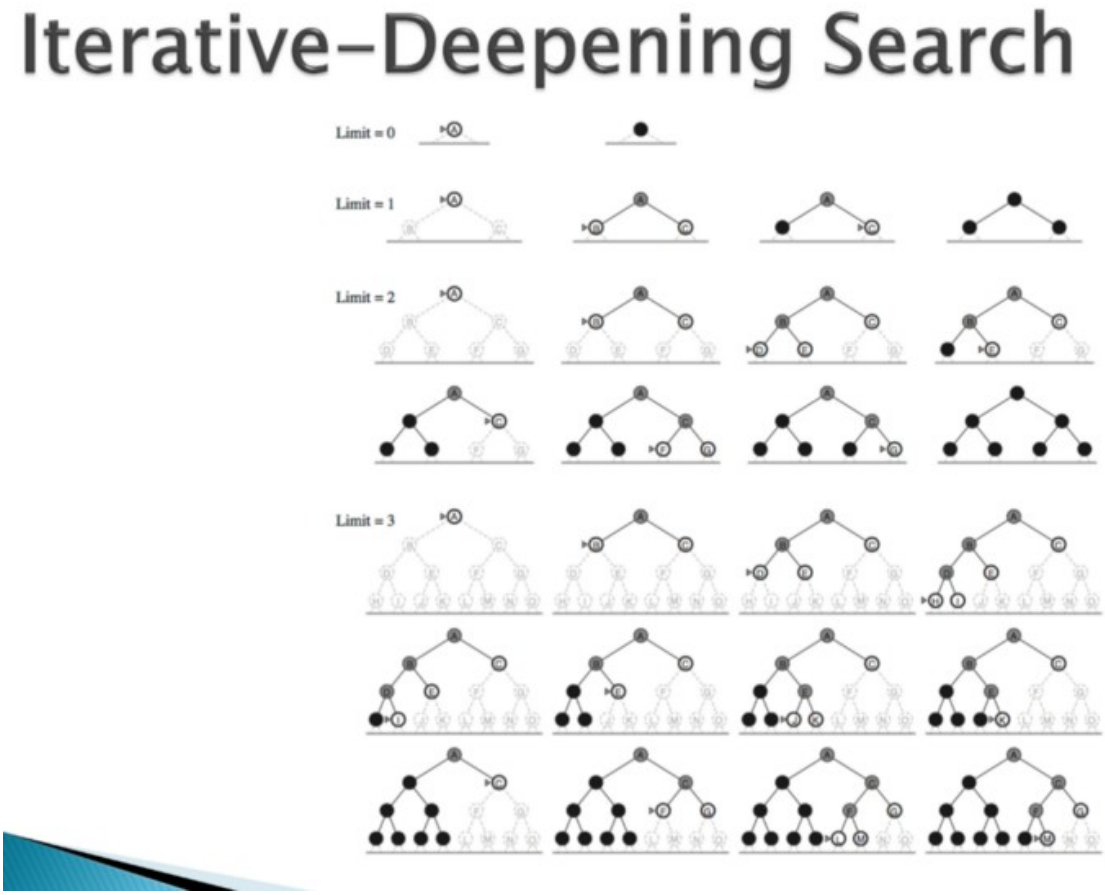
Depth-limited search can be terminated with two Conditions of failure:

- **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**

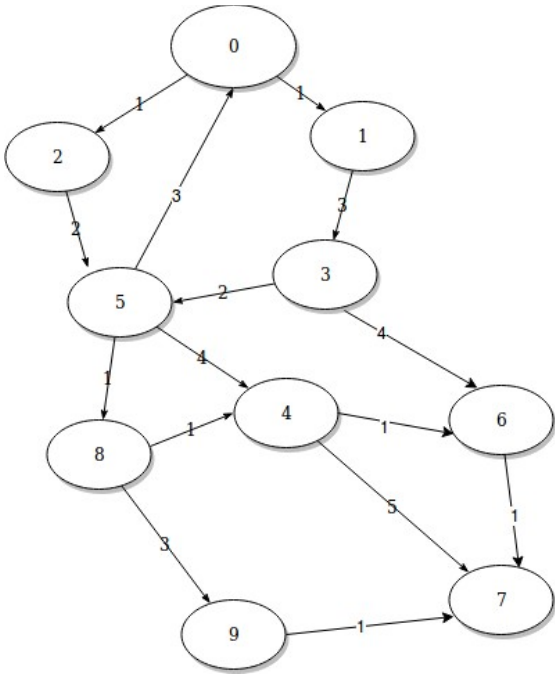
- Iterative deepening search is an extension of the depth limited search. Starting at the depth limit, you iteratively increase the depth until a solution is found or it has failed. If no nodes were cut off in this search, than it has exhausted all available paths.
- It is also complete as it finds a 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.



5. **Uniform Cost Search**

Uniform Cost Search is an algorithm used to move around a **directed weighted** search space to go from a start node to one of the ending nodes with a **minimum cumulative cost**. This search is an uninformed search algorithm, since it operates in a brute-force manner i.e it does not take the state of the node or search space into consideration. It is used to find the path with the lowest cumulative cost in a weighted graph where nodes are expanded according to their cost of traversal from the root node. This is implemented using a priority queue where lower the cost higher is its priority

Lets take the directed graph given below as example:



These are all the possible paths to reach from node 0 to node 7 with the cost of traversal
path: 0->1->3->5->4->6->7 cost: 12
path: 0->2->5->4->6->7 cost: 9
path: 0->2->5->8->4->7 cost: 10
path: 0->1->3->5->8->9->7 cost: 11
path: 0->2->5->8->9->7 cost: 8
path: 0->1->3->5->4->7 cost: 15
path: 0->1->3->6->7 cost: 9
path: 0->2->5->4->7 cost: 12
path: 0->1->3->5->8->4->7 cost: 13
path: 0->2->5->8->4->6->7 cost: 7
path: 0->1->3->5->8->4->6->7 cost: 10

best path: 0->2->5->8->4->6->7

National University of computer and emerging sciences

CL461-INTRODUTION TO ARTIFICIAL INTELLIGENCE

Lab Task

Algorithm of Uniform Cost Search

Below is the algorithm to implement Uniform Cost Search in Artificial Intelligence

- Insert RootNode into the queue.
- Repeat till queue is not empty:
- Remove the next element with the highest priority from the queue.
- if the node is destination node then print the cost and the path and exit

  else insert all the children of removed elements into the queue with their cumulative cost as their priorities.

Here rootNode is the starting node for the path and a priority queue is being maintained to maintain the path with the least cost to be chosen for the next traversal. In case 2 paths have the same cost of traversal, nodes are considered alphabetically.

Problem

Imagine You have map of state with distance mentioned between all adjacent cities now you want to travel from city 1 to city 8 but you have several different combination of routes to select. In such cases UCS can be used to select an optimal path. By considering cities as individual nodes and distance as cost of traversal between two nodes.

Code Provided:

Provided code file named UCS.ipynb contains skeleton code which generate weighted edges and store directed graph in dictionary corresponding to their cost.

self.cost_dict contains cost of each edge traversal of (u,v)

self.graph contains all the adjacent nodes of each as key and value pair in **PriorityQueue**

Tasks to Perform:

1) Implement UCS function of class Graph to find optimal path from source to destination

2) Recursively explore all paths from current city to destination city

3) Store all paths along with their cost in **self. final_dict**

4) Run the code given at end to see all paths and shortest path find by your UCS