Artificial-Intelligence

FRL Definitions

- **State-Space**: The set of all possible states.
- **Optimal Solution**: A solution that meets certain criteria, such as minimizing cost, maximizing utility, or achieving the best possible outcome, given a specific problem and its constraints. It is the most desirable solution among all possible solutions available for a given problem.
- **Polynomial problems**: These are problems for which the time complexity of the best-known algorithm is polynomial in the size of the input. In other words, the time taken to solve these problems grows at most polynomially with the size of the input.
- **Non-polynomial problems**: These are problems for which the time complexity of the best-known algorithm is non-polynomial in the size of the input. In other words, the time taken to solve these problems grows exponentially or faster with the size of the input.
- **Heuristic**: The term **Heuristic** originates from the Greek word "heuriskein," which means "to discover" or "to find." It is a strategy that helps to efficiently navigate complex situations or tasks, often by simplifying the problem or focusing attention on relevant information.

Basics

- The main goals of an artificially intelligent system are:
  - Reasoning
  - Learning
  - Problem Solving
  - Perception … like a human.
- When it comes to solving a problem, you need to represent the problem in such a way that the machine can understand it. Represent:
  - Precisely
  - In such a way that it can be analyzed

Searching Techniques

- Uninformed & Informed Search
- Difference: `+` : Informed, `-` : Uninformed

```
+ Utilizes specific information about the problem domain to guide the search process
- Lacks specific domain knowledge and relies solely on general search strategies. It
```

```
+ Generally more efficient in terms of time and space complexity.
- May be less efficient compared to informed search algorithms, especially for comple
+ Designed to provide an optimal solution.
- May or may not provide an optimal solution.
+ Examples include A* search, heuristic search, and informed hill climbing.
- Examples include depth-first search, breadth-first search, and uniform-cost search.
```

State-Space Search

- State-Space: The set of all possible states.
- Set: {S,A,Action(s), Result(s,a), Cost(s,a)}
    - S: Start, Goal
    - A: The set of all possible actions.
    - Action(s): The action we chose to execute.
    - Result(s,a): State formed as a Result of the action.
    - Cost(s,a): Cost of execute the action. The goal is to minimize the cost.

Breadth-First Search

- Type: Uninformed Search.

- Based on: FIFO (Queue).

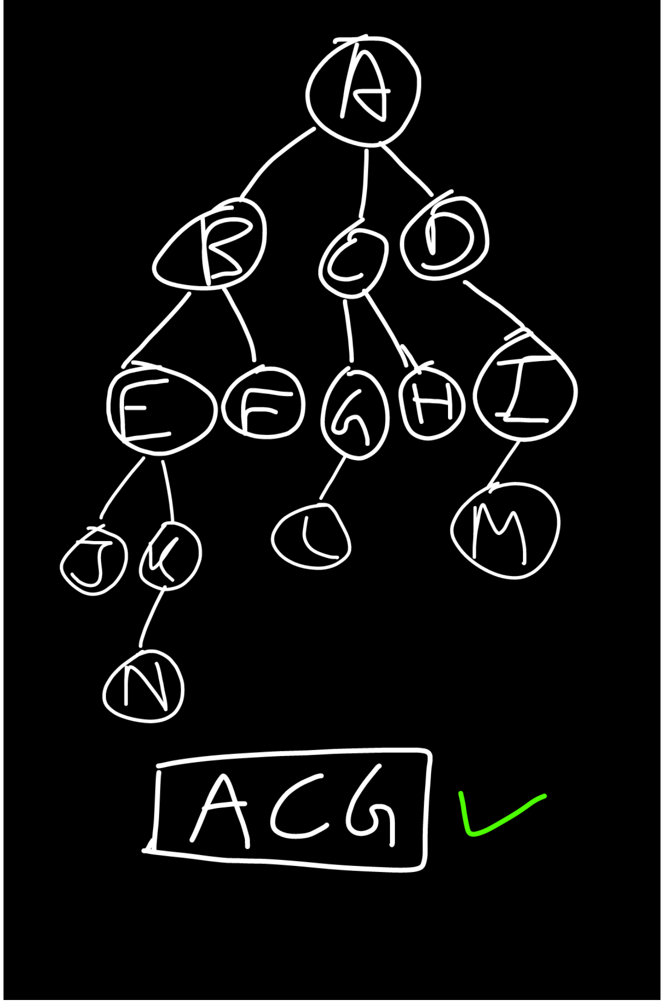
- Time complexity: $O(b^d)$

    - b: Branch factor, maximum number of children of a node.
    - d: Depth: Maximum Level of the tree, root node is at Level 0.
- Optimal, provides the best solution, if costs of all nodes is the same.

- Complete, always provides a solution.

- Example 0 (Start: A, Goal: G):

    i. A
    ii. ~~A~~BCD
    iii. ~~B~~CDEF
    iv. ~~C~~DEFGH
    v. ~~D~~EFGHI
    vi. ~~E~~FGHIJK
    vii. ~~F~~GHIJK
    viii. ~~G~~HIJKL
    - `G` was found. Result: `ACG` .
    - Note the implementation of FIFO: Elements are removed from LHS, and inserted from RHS.



Depth-First Search

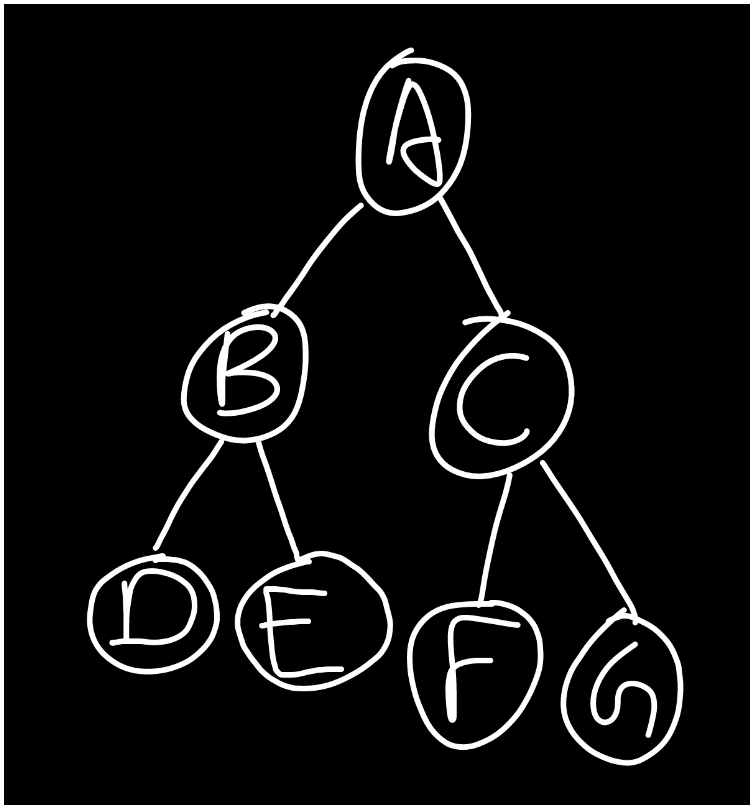
- Type: Uninformed Search.
- Based on: LIFO (Stack).
- Time complexity: $O(b^d)$
    - b: Branch factor, maximum number of children of a node.
    - d: Depth: Maximum Level of the tree, root node is at Level 0.
- Not Optimal, may not provide the best solution.
- Not Complete, may not provide a solution.
- Example 0 (Start: A, Goal: D):

    i. A
    ii. ~~A~~BC
    iii. B~~C~~FG
    iv. BF~~G~~
    v. B~~F~~
    vi. ~~B~~DE

vii. D~~E~~

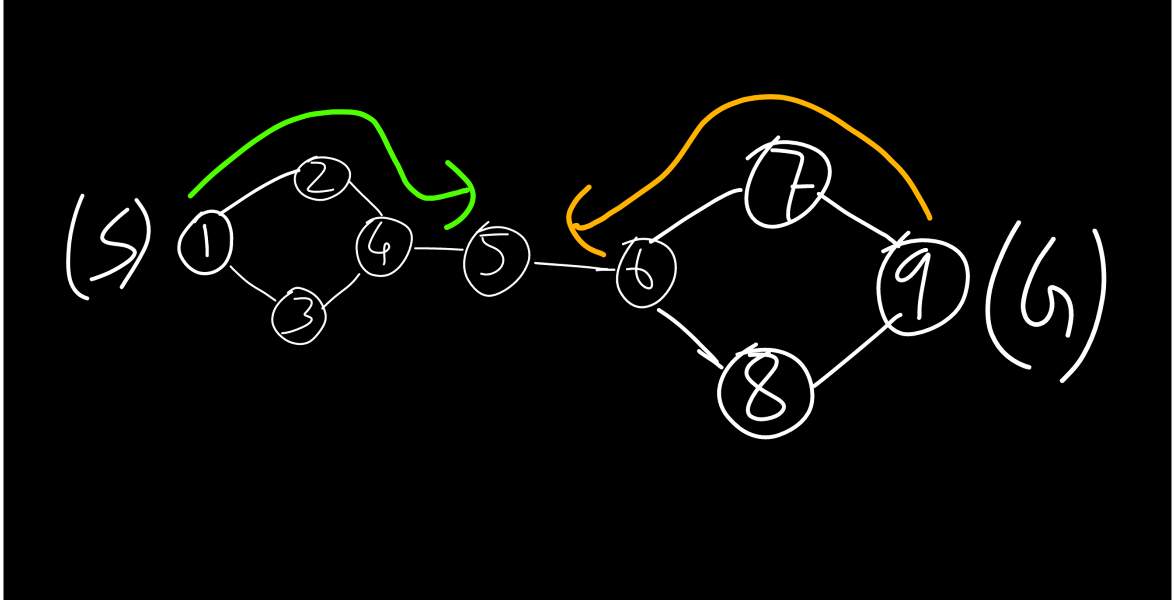
viii. ~~D~~

  - `D` was found. Result: `ACG`
  - Sequence: `ACGFBED`
  - Note the implementation of LIFO: Elements are removed from RHS, and inserted from RHS.



Bi-directional Search

- Type: Depends on algorithm used.
- 2 simultaneous search, one from initial node to goal node, another from goal node to initial node.
- Time complexity: $O(b^d + b^d) = O(2b^{d/2})$
  - b: Branch factor, maximum number of children of a node.
  - d: Depth: Maximum Level of the tree, root node is at Level 0.
- Complete only in case of Breadth-First Search.



8-Puzzle Problem without Heuristic

- Type: Blind / Uninformed Search.
- Based on: Breadth-First Search
- Time complexity: $O(b^d)$
  - b: Branch factor, maximum number of children of a node.
  - d: Depth: Maximum Level of the tree.
- Example 0:
  - Actions (A): UP (⬆), DOWN (⬇), LEFT (⬅), RIGHT (➡)
  - Start | End:

	S				G	
1	2	3	…	1	2	3
	4	6		4	5	6
7	5	8		7	8	

i. Step 1:

	0		->		⬆				➡				⬇	
1	2	3	->		2	3	…	1	2	3	…	1	2	3
	4	6	->	1	4	6		4		6		7	4	6
7	5	8	->	7	5	8		7	5	8			5	8

ii. Step 2 (from puzzle ➡):

	0		->		⬆				⬇				⬅			

	0		**->**		⬆				⬇				⬅			
1	2	3	->	1		3	…	1	2	3	…	1	2	3	…	1
4		6	->	4	2	6		4	5	6			4	6		4
7	5	8	->	7	5	8		7		8		7	5	8		7

iii. Step 3 (from puzzle ⬇):

	0		**->**		⬅				⬆				➡	
1	2	3	->	1	2	3	…	1	2	3	…	1	2	3
4	5	6	->	4	5	6		4		6		4	5	6
7		8	->		7	8		7	5	8		7	8	

iv. Step 4: Puzzle **3** is the Goal State.

- At every step, all valid moves are executed for all states, and all resultant states are determined.

Heuristic in Artificial Intelligence

- The term “heuristic” originates from the Greek word “heuriskein,” which means “to discover” or “to find.” It is a strategy that helps to efficiently navigate complex situations or tasks, often by simplifying the problem or focusing attention on relevant information.
- We use heuristic functions when we want to convert non-polynomial problems to polynomial problems (NP➡P).
    - **Polynomial problems**: These are problems for which the time complexity of the best-known algorithm is polynomial in the size of the input. In other words, the time taken to solve these problems grows at most polynomially with the size of the input.
    - **Non-polynomial problems**: These are problems for which the time complexity of the best-known algorithm is non-polynomial in the size of the input. In other words, the time taken to solve these problems grows exponentially or faster with the size of the input.
- It provides a good solution but **not an optimal solution**. This is because there may be other obstacles (like an infinite loop) in the path the algorithm has chosen. But in most cases, it provides a more efficient approach to brute-forcing our way to the solution.
- Examples of some functions:
    - **Eucledian Distance**: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$, using this we can find the shortest path from point a to point b.
    - **Manhattan Distance**:

	S		**->**		**G**	
1	3	2	->	1	2	3
6	5	4	->	4	5	6
	8	7	->	7	8	

- Manhattan Distance: $0+1+1+2+0+2+2+0=8$
- The distance is calculated by the number of steps each number needs to be moved, to reach the Goal state (G) from the Current state.

Searching Techniques

8-Puzzle Problem with Heuristic

- Type: Informed Search.
- Based on: Number of misplaced tiles
- Example 0:
  - Actions (A): UP (⬆), DOWN (⬇), LEFT (⬅), RIGHT (➡)
  - Start | End:

	S				**G**	
1	2	3	…	1	2	3
	4	6		4	5	6
7	5	8		7	8	

- Number of misplaced tiles, $h=3$

i. Step 1:

	0		**->**		**⬆**				**➡**				**⬇**	
1	2	3	->		2	3	…	1	2	3	…	1	2	3
	4	6	->	1	4	6		4		6		7	4	6
7	5	8	->	7	5	8		7	5	8			5	8

- Number of misplaced tiles, $h=4,2,4$
- We will move forward with the lowest heuristic value.

ii. Step 2 (from puzzle ➡):

	0		**->**		**⬆**				**⬇**				**⬅**			

	0		**->**		⬆				⬇				⬅			
1	2	3	->	1		3	…	1	2	3	…	1	2	3	…	1
4		6	->	4	2	6		4	5	6			4	6		4
7	5	8	->	7	5	8		7		8		7	5	8		7

- ○ Number of misplaced tiles, $h = 3, 1, 3, 3$

iii. Step 3 (from puzzle ⬇):

	0		**->**		⬅				⬆				➡	
1	2	3	->	1	2	3	…	1	2	3	…	1	2	3
4	5	6	->	4	5	6		4		6		4	5	6
7		8	->		7	8		7	5	8		7	8	

- ○ Number of misplaced tiles, $h = 2, 2, 0$

iv. Step 4: For Puzzle ➡, $h = 0$, and it is the Goal State.

v. We had to traverse through a lot less states to reach the Goal state (G), compared to a typical Uninformed Search Technique.

Generate and Test

1. Generate a possible solution.
2. Test to see if this is an actual solution.
3. If a solution is found, otherwise repeat.

- Properties of Good Generators:
    - **Complete**
    - **Non-redundant**: They must not provide solutions which have already been generated in the past.
    - **Informed**: The Generator must have atleast some basic idea which it can use to generate an efficient solution.