Solving Problems by Search

In which we see how an agent can find a sequence of actions that achieves its goals when no single action will do.

What do we want?

- Automatically solve a problem.

What do we need?

- A representation of the problem.
- Algorithms that use some strategy to solve the problem.

Problem Representation: In general,

- **State Space**: A problem is divided into a set of *resolution* steps from the *initial state* to the *goal state*.
- **Reduction to Sub-Problems**: A problem is arranged into a hierarchy of sub-problems.

A **State** can be defined as a representation of problem elements in a given moment.

Two special states are defined:

- Initial State (Starting Point).
- Final State (Goal State).

A **State Space** can be defined as all the states that are reachable from the initial state.

A **Path** is a sequence of states connected by a sequence of actions.

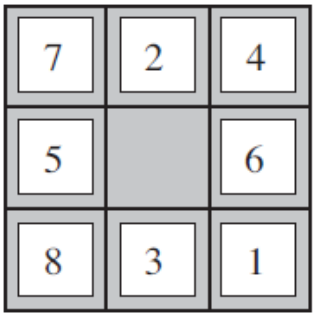
A **Solutions** is the path from the **initial** state to the **final** state.

Problem Description: When we want to describe a problem, we should specify the following elements:

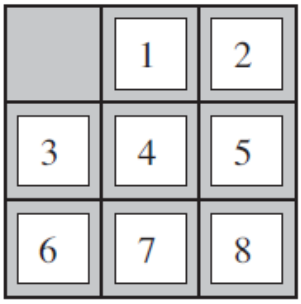
- State Space.
- Initial State.
- Goal State.
- Available Actions. Or operators to change state.
- Restrictions: like Cost.
- Type of Solution:

- Sequence of goal state.
- One optimal solution.

Example 1: The 8-puzzle Problem As shown in the figure below, it consists of a 3×3 board with eight numbered tiles and a blank space.



Initial State



Goal State

- **Problem Description**:

  State Space: Configuration of the eight tiles on the board.

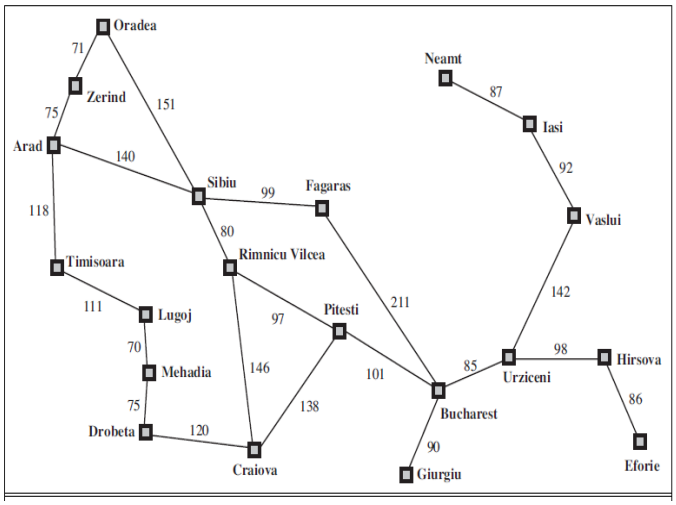
  Initial State: Any configuration.

  Goal State: Tiles in a specific order.

  Operators: blank moves Left, Right, Up, Down.

  Solution: Optimal sequence of operators.

Example 2: Travelling from Arad to Bucharest



- **Problem Description**:

  State Space: Various cities.

  Initial State: Arad.

  Goal State: Be in Bucharest.

  Operators: Drive between cities.

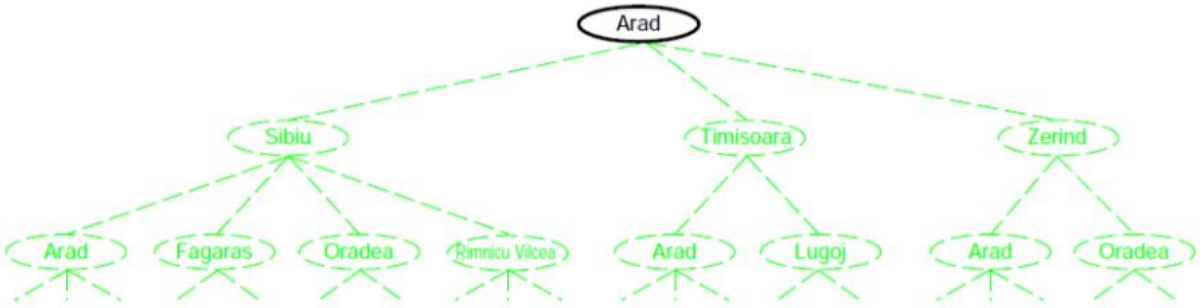
  Solution: Sequence of Cities.

- In order to solve this problem, we can use Tree Search Algorithms. Hence, successors of the states that haven't been explored are generated.

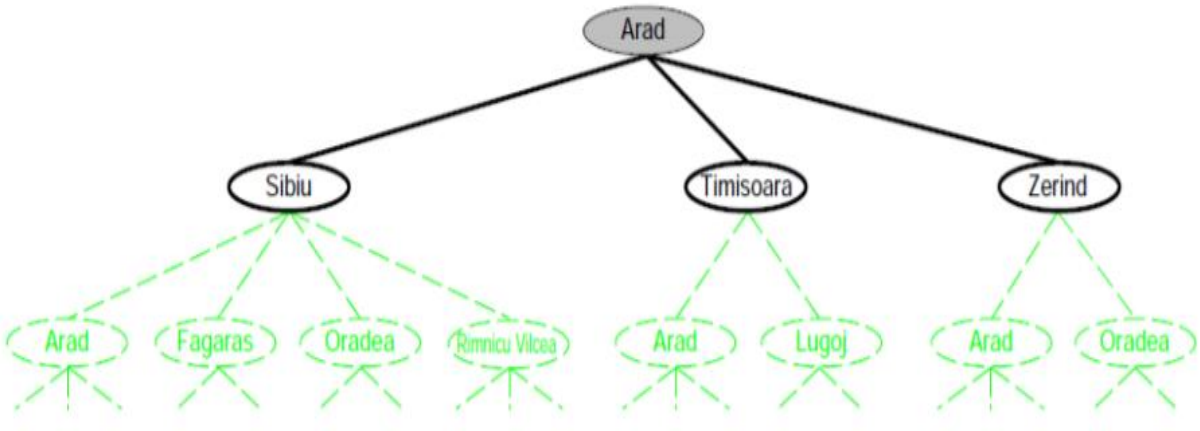
```
function TREE-SEARCH (problem, strategy)
returns a solution, or failure

    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state
            then return the corresponding solution
        expand the chosen node and add the resulting nodes to the frontier
    end
```

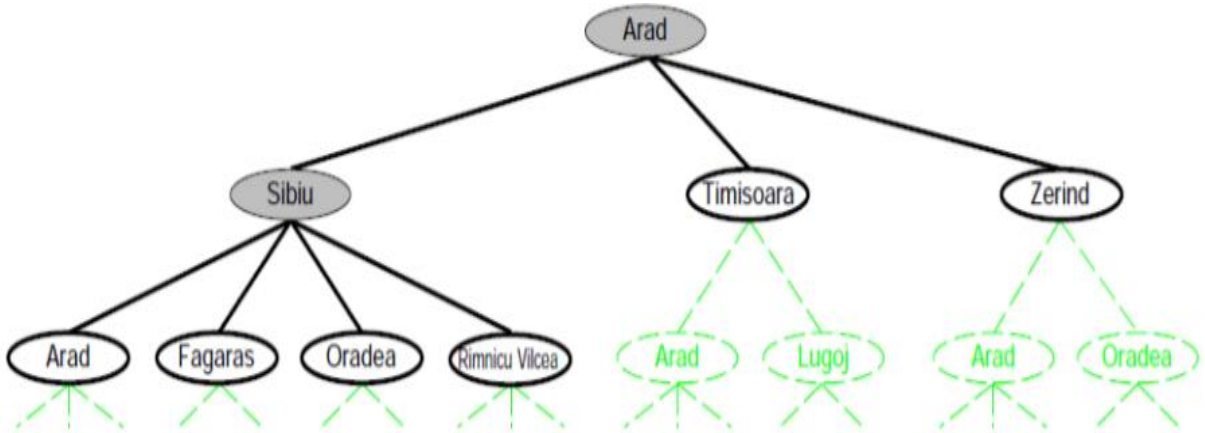
Step 1:



Step 2:



Step 3:



Search Algorithms: There are many search algorithms. The following are widely used to solve AI problems.

- Breadth-First Search.
- Depth-First Search.
- Uniform-Cost Search.
- Depth-limited search.
- Iterative deepening search.
- Bidirectional search.

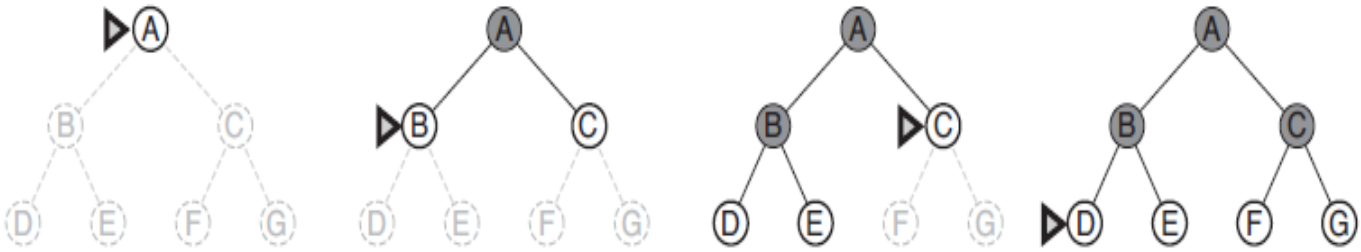
We will focus on the first two algorithms.

Breadth-First Search: is a simple strategy in which the root node is expanded first, then all the SEARCH
successors of the root node are expanded next, then their successors, and so on.

- At each stage, the node to be expanded is indicated by a marker.
    - The nodes that are already explored are gray.
    - The nodes with dashed lines are not generated yet.

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
    if EMPTY?(frontier) then return failure
    node ← POP(frontier)  /* chooses the shallowest node in frontier */
    add node.STATE to explored
    for each action in problem.ACTIONS(node.STATE) do
      child ← CHILD-NODE(problem, node, action)
      if child.STATE is not in explored or frontier then
        if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
        frontier ← INSERT(child, frontier)
```

Example:



Depth-First Search: always expands the *deepest* node in the current frontier of the search tree.