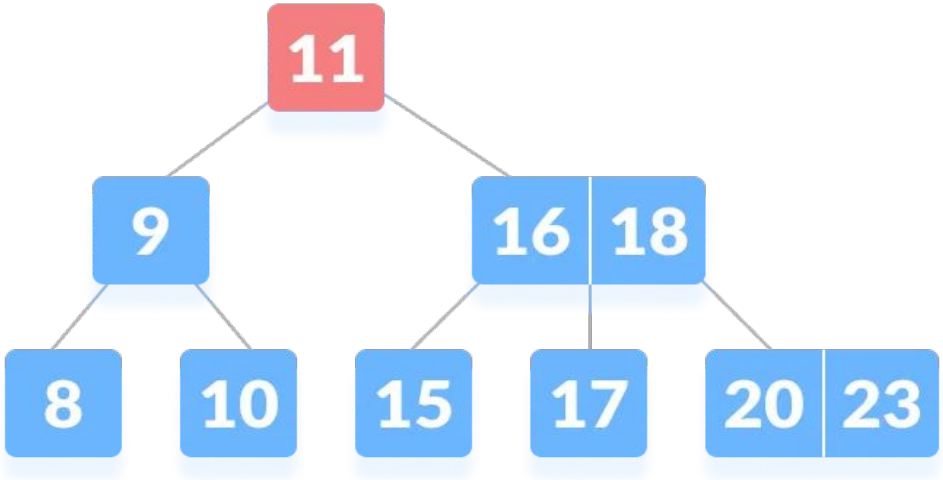
Problem Solving by Searching

BCA SEM-6 (Gujarat University)

Problem Solving Agents

The problem-solving agent performs precisely by defining problems and several solutions.

So we can say that problem solving is a part of artificial intelligence that encompasses a number of techniques such as a tree, B-tree, heuristic algorithms to solve a problem.



Problem Solving Agents

Intelligent agents are supposed to maximize their performance measure.

In Artificial Intelligence searching method is the best technique to solve the problems whether it is defined or not defined.

AI used search strategies to solve the specific problem.

To solve the problems in Artificial Intelligence it need following things:

a. Problem Define

b. Problem Analyze

c. Isolate problem and knowledge of task

d. Get techniques

e. Choose techniques to solve problems

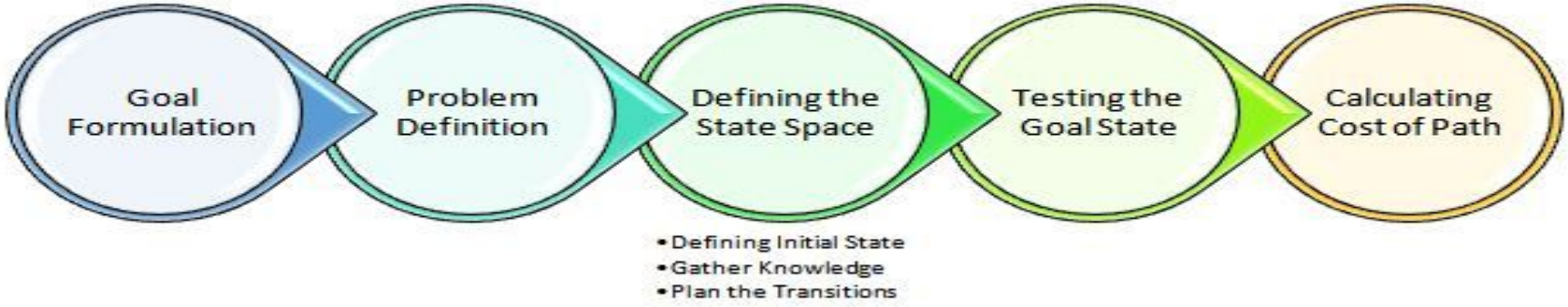
Process of Problem Solving

To build a system to solve a particular problem first we need to specify the initial system

(1) what exactly the problem is and the acceptable final solutions to the problem.

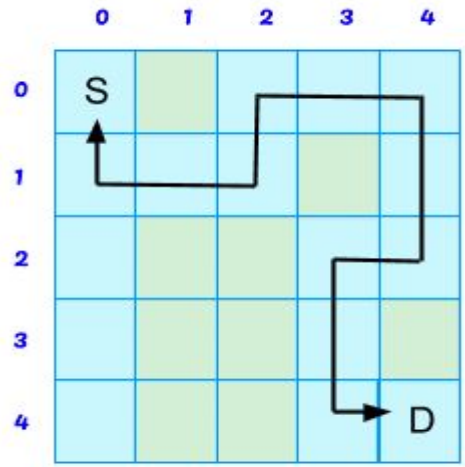
(2) Isolate the problem and read about the problems then get multiple possible techniques that are necessary to solve the problem,

(3) After that we can get the best technique and apply it (them) to the particular problem.



State Space Search

Define the problem as a state space search. **All the possible state for solving any particular problem is called as state space.** For searching the particular state from the state space we use state space search.



S: {S, A , Action(s), Result(s,a), Cost(s,a) }

State Space Search

The structure of state space is of two types:

• **In some problems set of states and rules are already defined** and **well structured organize.** We have to convert some given condition into the required condition using a set of operations which are already decided.

• In some problems **these kinds of representation occurs naturally and are not well structured.** In this **we have to make the desired structure with the help of some techniques to find the path from the initial state to goal state.** Search is an important procedure for the problems to find the, to and Forth solution no direct techniques are used in it.

Problem Solving Agents

Problem solving agents are the **goal based agents** which are also called as problem solving agents.

Simple Problem solving agents have **limited strategies to solve the problems.**

They solve the problems on current perceptions **they don't know what they want and what are the goals.**

Problem solving agents are such that **they create the sequential environment to maximize the performance of searching method to solve any problem.**

Play Chess Problem

We have to build a program which can play chess. Here the required conditions a already defined which can be the position of the **players, moves, operations, initial state** and **goal state, legal game to win and lost rules.**

In the play chess game the rules are already provided and **the starting position is defined as an 8 by 8 position where each box contains its value and has some opening rules with the legal move.** There are many ways to apply this rules because of that it is too difficult and lengthy one could not get the goal without any mistakes. **To solve this problem if possible one should look for a way to write the rules with legal moves.**

Milk Conman Problem

We are given **two conman (utensil to pour liquid), a 6 liter and 4 liter.** We don't have measuring cup or any mark. We are given a tap to fill the Conman. **How can we get exact 2 liter of milk into 6 liter of Conman ?**

4 Liters

6 Liters

One Solution of Milk Conman Problem	
Liters in the 6 Liter of Conman	**Liters in the 4 liter of Conman**
0	0
0	4
4	0
4	4
6	2
0	2
2	0

Goal Formulation

Goals help organize behavior **by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider**. Goal formulation, based on the current situation and the agent's performance measure, is the first step in problem solving.

We will **consider a goal to be a set of world states-exactly those states in which the goal is satisfied. The agent's task is to find out how to act, now and in the future, so that it reaches a goal state.** Before it can do this, it needs to decide (or we need to decide on its behalf) what sorts of actions and states it should consider.

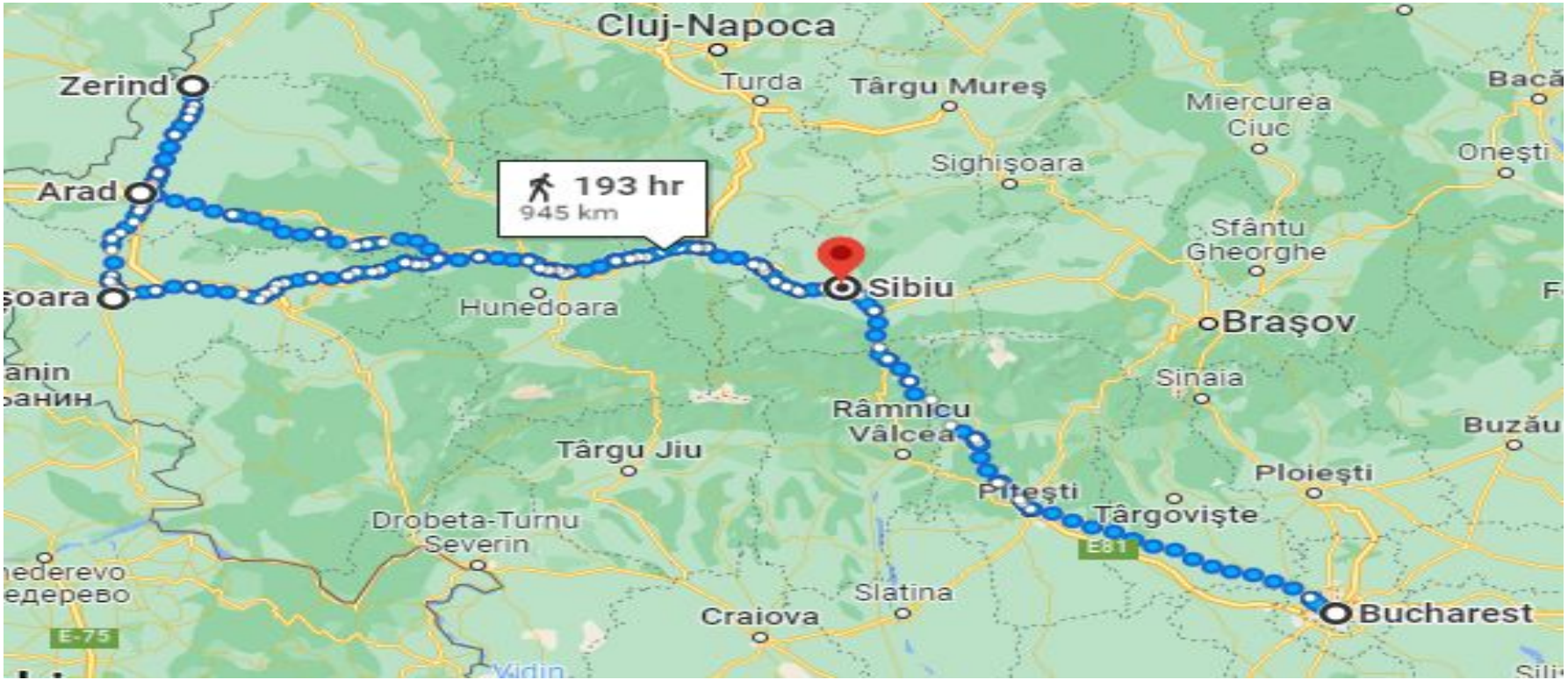
Problem Formulation

Problem formulation is the process of deciding what actions and states to consider, given a goal.

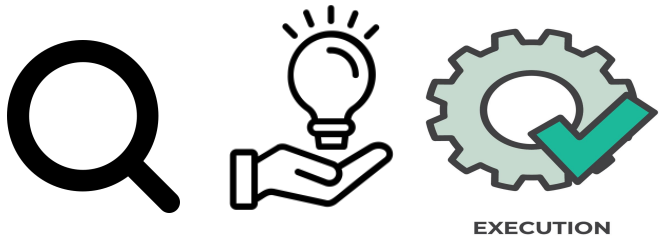
For now, let us assume that the agent will consider actions at the level of driving from one major town to another. Each state therefore corresponds to being in a particular town.

Our agent has now adopted the goal of driving to Bucharest and is considering where to go from Arad. Three roads lead out of Arad, one toward Sibiu, one to Timișoara, and one to Zerind. None of these achieves the goal, so unless the agent is familiar with the geography of Romania, it will not know which road to follow. In other words, the agent will not know which of its possible actions is best, because it does not yet know enough about the state that results from taking each action.

Problem Formulation



Search, Solution and Execution

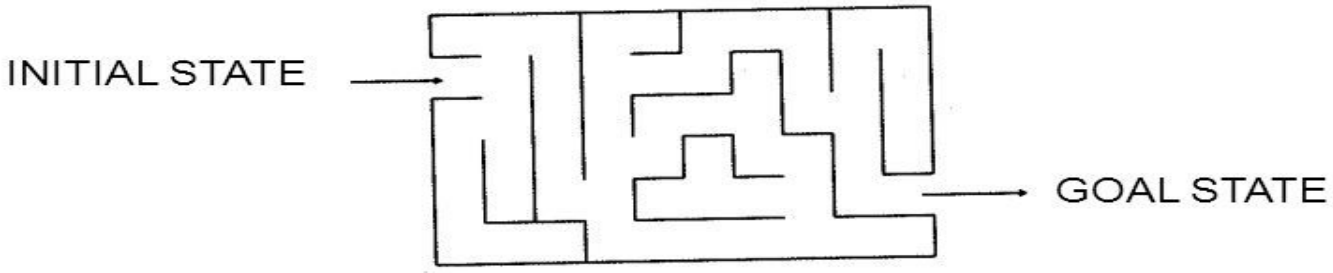


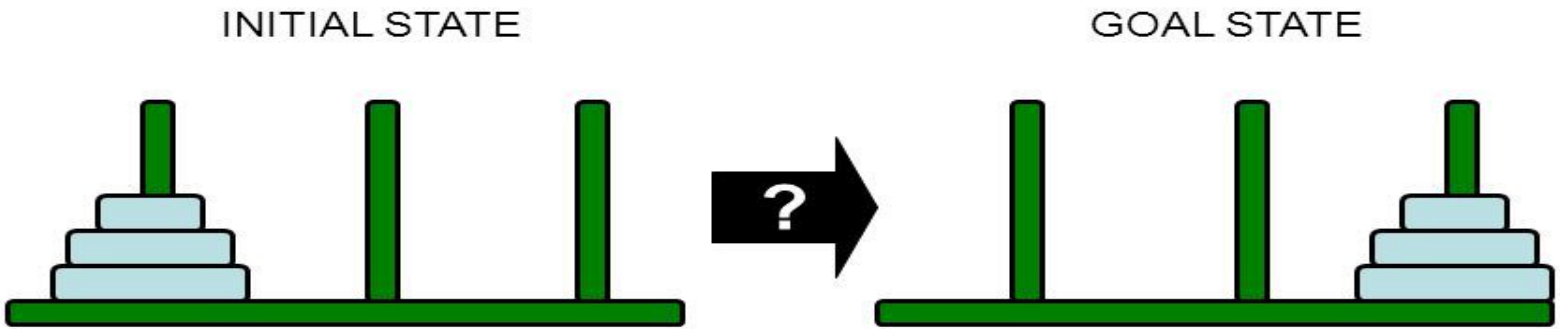
The process of looking for a sequence of actions that reaches the goal is called search.

A search algorithm takes a problem as input and returns a solution in the form of an action sequence.

Once a solution is found, the actions it recommends can be carried out. This is called the execution phase.

Problem Solving as Search





Well Defined Problems and Solutions

A Problem can be identified formally by five components:

1) Initial State
2) Applicable Actions
3) Transition Model
4) Goal Test
5) Path Cost

Well Defined Problems and Solutions

- **The Initial State that the agent starts in.** For Example, the initial state for our agent in Romania might be described as **In(Arad).**

- A description of the possible actions available to the agent. Given a particular states, **ACTIONS (s) returns the set of actions that can be executed in s.** We say that each of these actions is applicable in s. **For example, from the state In (Arad), the applicable actions are {Go (Sibiu), Go(Timișoara), Go(Zerind)}.**

A Simplified Road Map of Romania

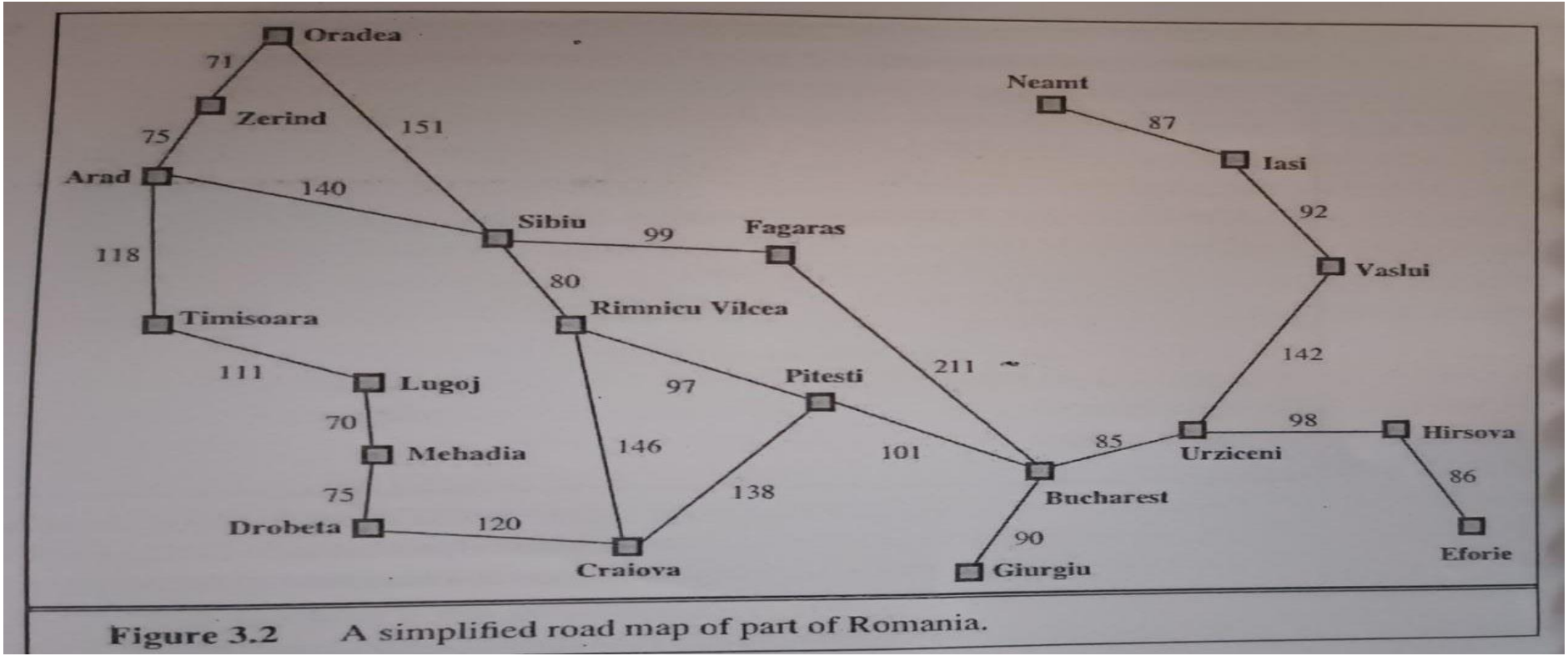


Figure 3.2 A simplified road map of part of Romania.

Well Defined Problems and Solutions

- **A description of what each action does;** the formal name for this is the transition model, **specified by a function RESULT(s, a) that returns the state that results from doing action in states.** We also use the term **successor** to refer to any state reachable from a given state by a single action. For example, we have

  RESULT (In (Arad), Go(Zerind)) = In(Zerind).

Well Defined Problems and Solutions

- Together, **the initial state, actions, and transition model implicitly define the state space of the problem** the set of all states reachable from the initial state by any sequence of actions.
- The **state space forms a directed network or graph** in which the nodes are states and the links between nodes are actions.
- **A path in the state space is a sequence of states connected by a sequence of actions.**

Well Defined Problems and Solutions

- The goal test, which determines whether a given state is a goal state. Sometimes **there is an explicit set of possible goal states,** and the test simply checks whether the given state is one of them.
- The agent's goal in Romania is the singleton set {In(Bucharest)}.

Well Defined Problems and Solutions

- **A path cost function that assigns a numeric cost to each path.**
- The problem-solving agent chooses a cost function that reflects its own performa trying to get to Bucharest, time is of the essence, so the cost of a path might be its length measure. **For the agent in kilometers.**

Formulating Problems

- In the preceding section we proposed a formulation of the problem of getting to Bucharest in terms of the initial state, actions, transition model, goal test, and path cost.
- This formulation seems reasonable, but it is still a model-an abstract mathematical description and not the real thing.
- Compare the simple state description we have chosen, In(Arad), to an actual cross country trip, where the state of the world includes so many things: the traveling companions, the current radio program, the scenery out of the window, the proximity of law enforcement officers, the distance to the next rest stop, the condition of the road, the weather, and so on.

Formulating Problems

- All these considerations are left out of our state descriptions because they are irrelevant to the problem of finding a route to Bucharest. The process of removing detail from a representation is called abstraction.

Formulating Problems

Formulating problem is depending on the knowledge, environment and perception of the problem. We can divide the problem as

a. Single state problem

b. Multiple state problem

c. Contingency problem

d. Exploration problem

Formulating Problems

Let us learn the problem with assumption before that we should understand the problem first, when the initial or present state is known with that one should get to know about the net action, sequence of action or sometimes we get goal state which is called single state problem.

If all the action are known but because of some rules, we get the limited access of the action then the possibility of happening the particular state is greater than 1, thus it can be known as multiple state problem.

Formulating Problems

When there is multiple state problem then to solving the problem the process can also be more than one.

Probability can be used to solve single problem thus this type of problem requires the sequence of action taken. It depends on the possible contingency which can also call as contingency problem. The real time problems are not exact predicted but some additional information like problem state can be known.

Toy Problem

- 8- puzzle problem-

Start State

1	2	3
4	8	-
7	6	5

Goal State

1	2	3
4	5	6
7	8	-

Down
Up
Left
Right

Toy Problem

The 8-Puzzle problem is 3×3 board with 8 tiles and a blank space in which tiles slides from one place to another only where the blank space is available.
In the puzzle we have some rules and conditions of moving the tiles in which every tiles should be move only Up, Down, Left, Right. Tiles cannot move diagonally.

We can lead following formulation:
States: A state describes the location of each eight tiles and also of the blank tiles.
Operators: Up, Down, Left, Right (blank tiles only).
Start: Describes the initial state.
Goal: Describes the goal state.

Toy Problem

Path cost: Number of steps taken to reach the goal.

Following are the conditions and states to solve the 8-puzzle problem.

Down-

{(1,2,3), (4,8,0), (7,6,5)}

{(1,2,3), (4,8,5), (7,6,0)}

Left-

{(1,2,3), (4,8,5), (7,0,6)}

Up-

{(1,2,3), (4,0,5), (7,8,6)}

Right-

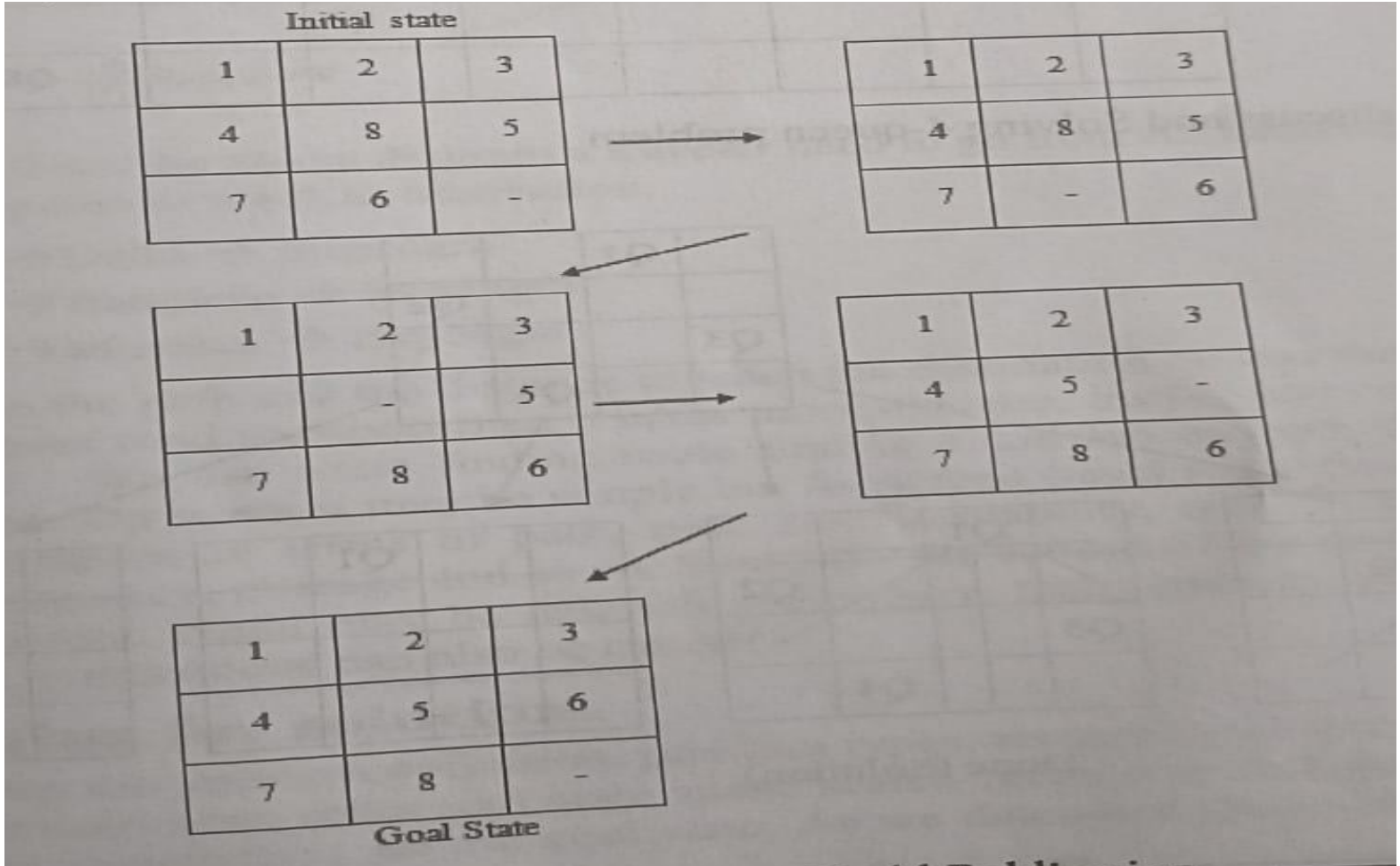
{(1,2,3), (4,5,0), (7,8,6)}

Down-

{(1,2,3), (4,5,6), (7,8,0)}

Path cost= ("5") i.e. no. of states taken to solve the problem.

Toy Problem


Initial state
1 2 3
4 8 5
7 6 -
1 2 3
4 8 5
7 - 6
1 2 3
- 5
7 8 6
1 2 3
4 5 -
7 8 6
1 2 3
4 5 6
7 8 -
Goal State

Toy Problem - 8 Puzzle

- States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.
- Initial state: Any state can be designated as the initial state. Note that any given goal can be reached from exactly half of the possible initial states (Exercise 3.4).
- Actions: The simplest formulation defines the actions as movements of the blank space Left, Right, Up, or Down. Different subsets of these are possible depending on where the blank is.

Toy Problem - 8 Puzzle

- Transition model: Given a state and action, this returns the resulting state; for example, if we apply Left to the start state in Figure 3.4, the resulting state has the 5 and the blank switched.
- Goal test: This checks whether the state matches the goal configuration shown in Figure 3.4. (Other goal configurations are possible.)
- Path cost: Each step costs 1, so the path cost is the number of steps in the path.

N Queen Problem

This is the type of constraint satisfaction problem in AI. It has NXN square grid board in which N number of queens is placed. **Some constraints/rules are :**

1. No row should contain more than one queen.
2. No column should contain more than one queen.
3. No diagonal should contain more than one queen.
4. There should be no row or column without any queen.

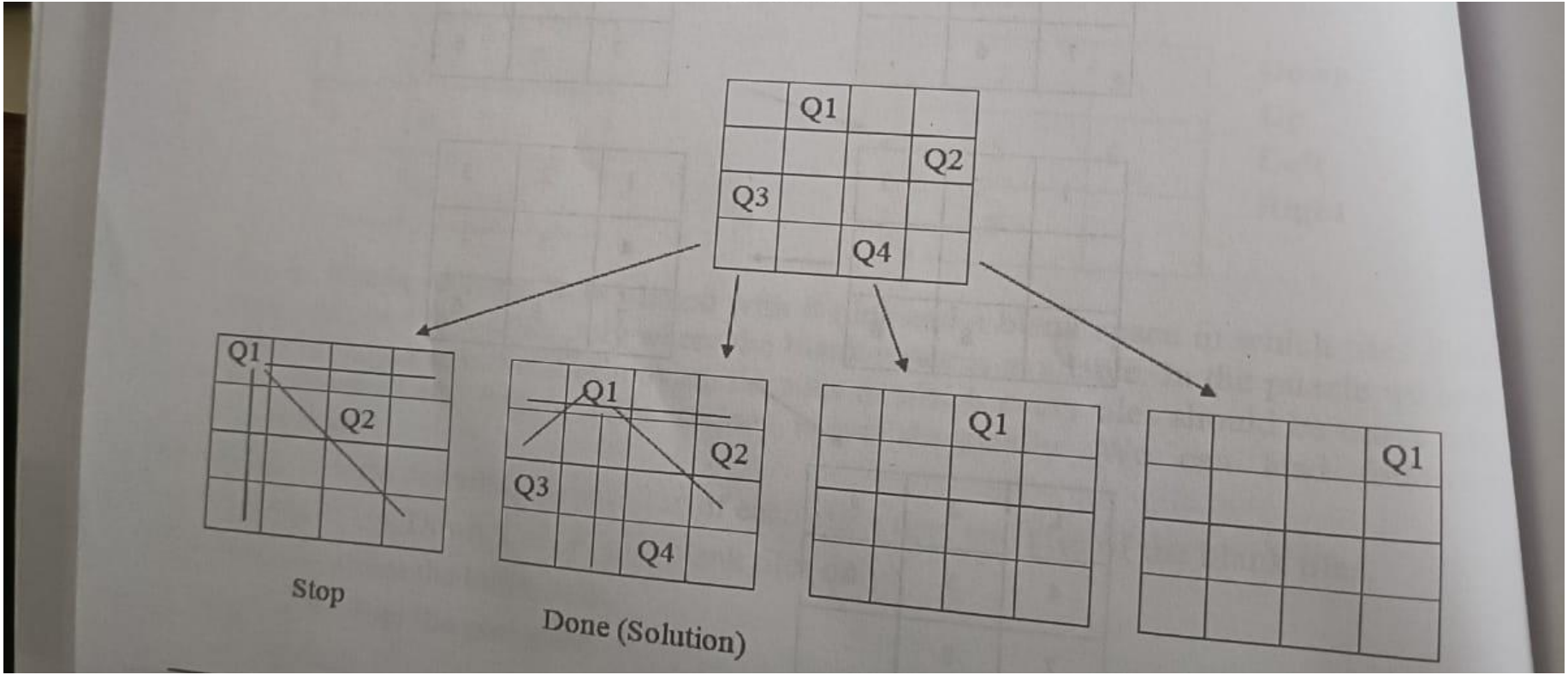
N Queen Problem

Incremental Formulation :

An incremental formulation involves operators that augment the state description, starting with an empty state; for the 8-queens problem, this means that each action adds a queen to the state.

- **States:** Any arrangement of 0 to 8 queens on the board is a state.
- **Initial state:** No queens on the board.
- **Actions:** Add a queen to any empty square.
- **Transition model:** Returns the board with a queen added to the specified square. • Goal test: 8 queens are on the board, none attacked.

N Queen Problem



N Queen Problem

Complete State Formulation :

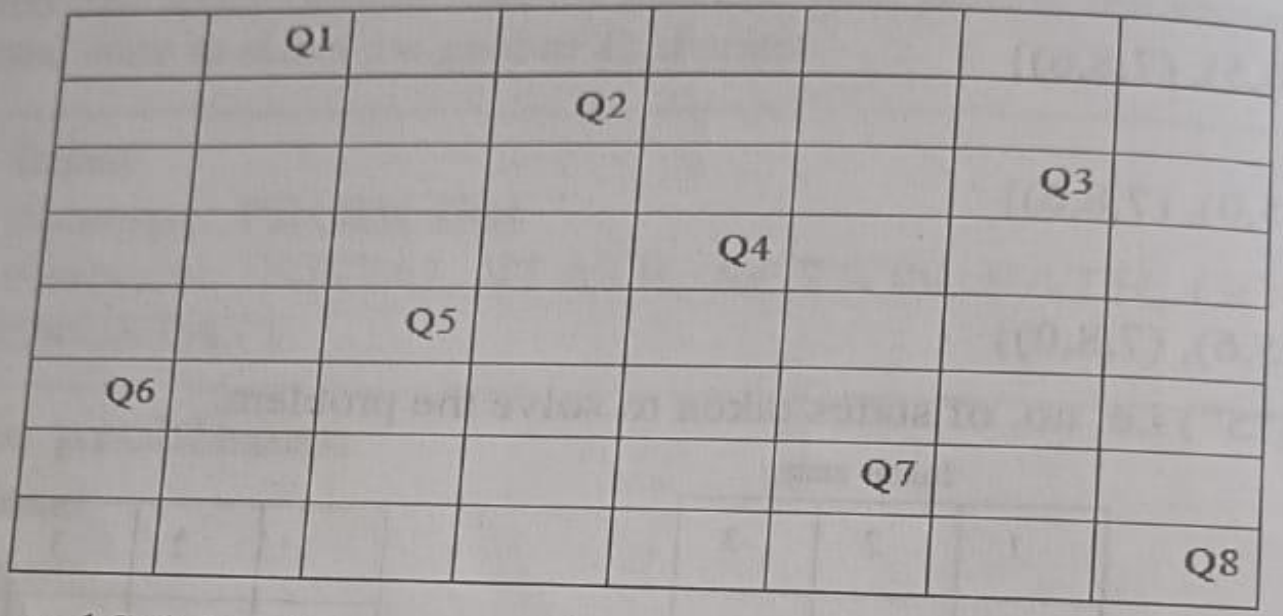
A complete-state formulation starts with all 8 queens on the board and moves them around. In either case, the path cost is of no interest because only the final state counts.

- States: All possible arrangements of n queens ($0 \leq n \leq 8$), one per column in the leftmost n columns, with no queen attacking another.

- Actions: Add a queen to any square in the leftmost empty column such that it is not attacked by any other queen.

N Queen Problem

Here is the goal state for 8-Queen problem –



Let we discuss and Solving 4-queen problem

Real World Problems

Route Finding Problem - Airline Travel Problems

- **States:** Each state obviously includes a **location** (e.g., an airport) and the **current time**. Furthermore, because the cost of an action (a flight segment) may depend on previous segments, their **fare bases**, and their **status as domestic or international**.

- **Initial state:** This is specified by the user's query.

- **Actions: Take any flight from the current location,** in **any seat class**, **leaving after the current time**, **leaving enough time for within-airport transfer if needed.**

Real World Problems

Route Finding Problem - Airline Travel Problems

- Transition model: The state resulting from taking a flight will have the **flight's destination as the current location and the flight's arrival time as the current time.**
- Goal test: **Are we at the final destination specified by the user?**
- Path cost: This depends on **monetary cost**, **waiting time**, **flight time**, **customs and immigration procedures**, **seat quality**, **time of day**,and so on.

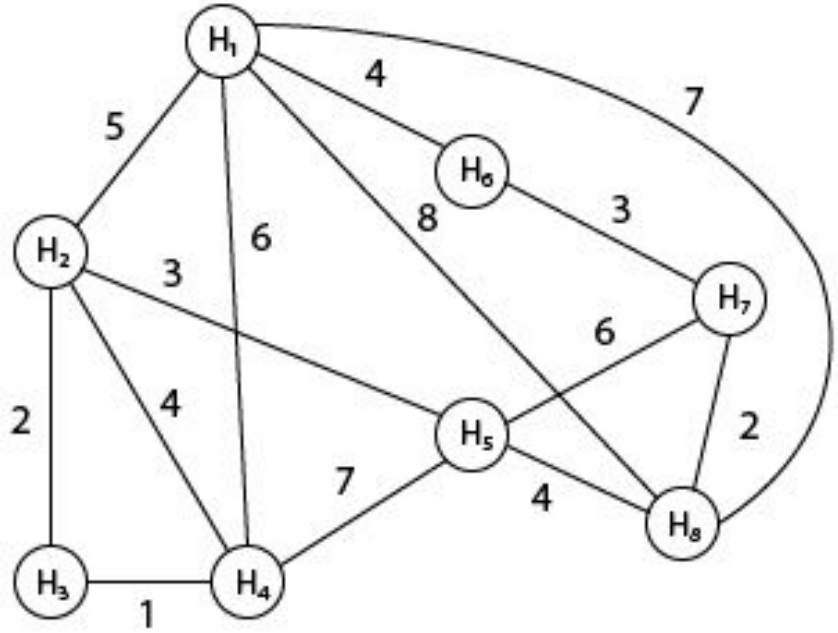
Real World Problems

Traveling Salesperson Problem (TSP)

The traveling salesman problems abide by a salesman and a set of cities. **The salesman has to visit every one of the cities starting from a certain one (e.g., the hometown) and to return to the same city.** The challenge of the problem is that the traveling salesman needs to minimize the total length of the trip.

Example: A newspaper agent daily drops the newspaper to the area assigned in such a manner that he has to cover all the houses in the respective area with minimum travel cost. Compute the minimum travel cost.

Real World Problems



Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	4	0	7
H_2	5	0	2	4	3	0	0	0
H_3	0	2	0	1	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	3	0	7	0	0	6	4
H_6	4	0	0	0	0	0	3	0
H_7	0	0	0	0	6	3	0	2
H_8	7	0	0	0	4	0	2	0

Real World Problems

	H_1	H_2	H_3	H_4	H_5	H_6	H_7	H_8
H_1	0	5	0	6	0	(4)	0	7
H_2	5	0	(2)	4	3	0	0	0
H_3	0	2	0	(1)	0	0	0	0
H_4	6	4	1	0	7	0	0	0
H_5	0	(3)	0	7	0	0	6	4
H_6	4	0	0	0	0	0	(3)	0
H_7	0	0	0	0	6	3	0	(2)
H_8	7	0	0	0	(4)	0	2	0

Real World Problems

Mark area H_4 and then select the minimum cost area reachable from H_4 it is H_1.So, using the greedy strategy, we get the following.

$$H_1 \xrightarrow{4} H_6 \xrightarrow{3} H_7 \xrightarrow{2} H_8 \xrightarrow{4} H_5 \xrightarrow{3} H_2 \xrightarrow{2} H_3 \xrightarrow{1} H_4 \xrightarrow{6} H_1.$$

Thus the minimum travel cost = 4 + 3 + 2 + 4 + 3 + 2 + 1 + 6 = 25

Searching for Solutions

SEARCH TREE NODE : The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.

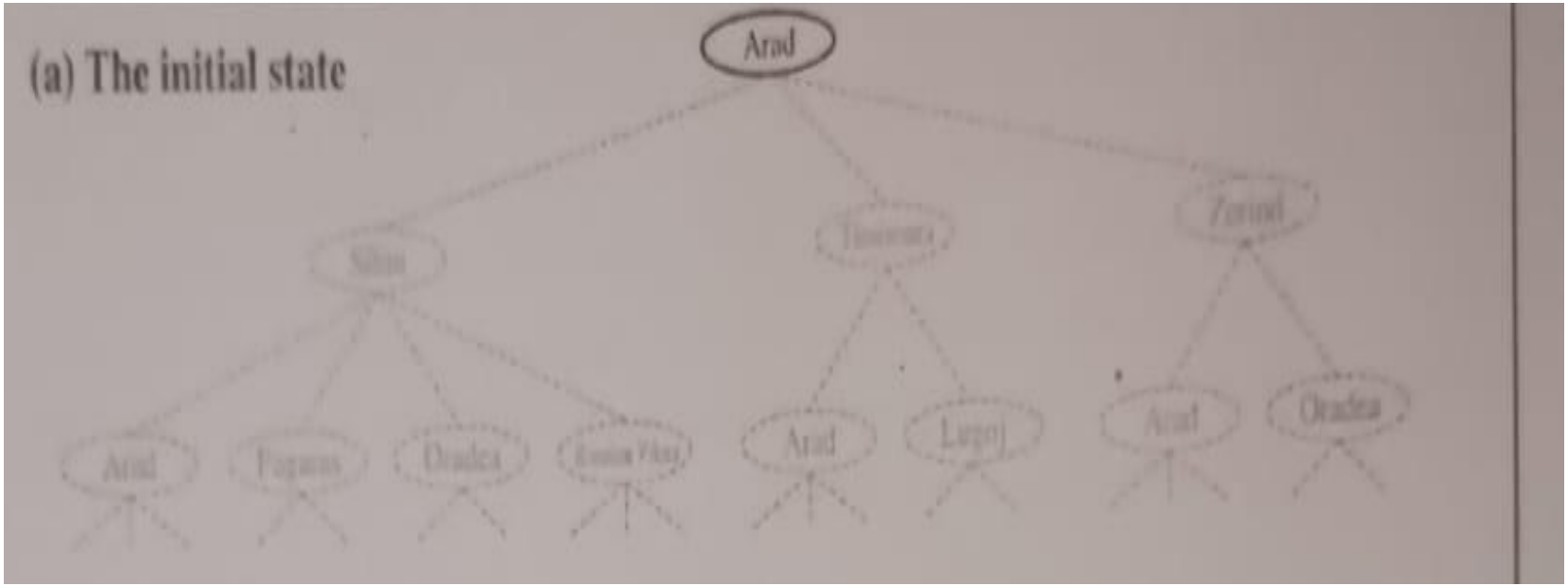
The root node of the tree corresponds to the initial state, In(Arad).

The first step is to test whether this is a goal state. (Clearly it is not, but it is important to check so that we can solve trick problems like "starting in Arad, get to Arad.")

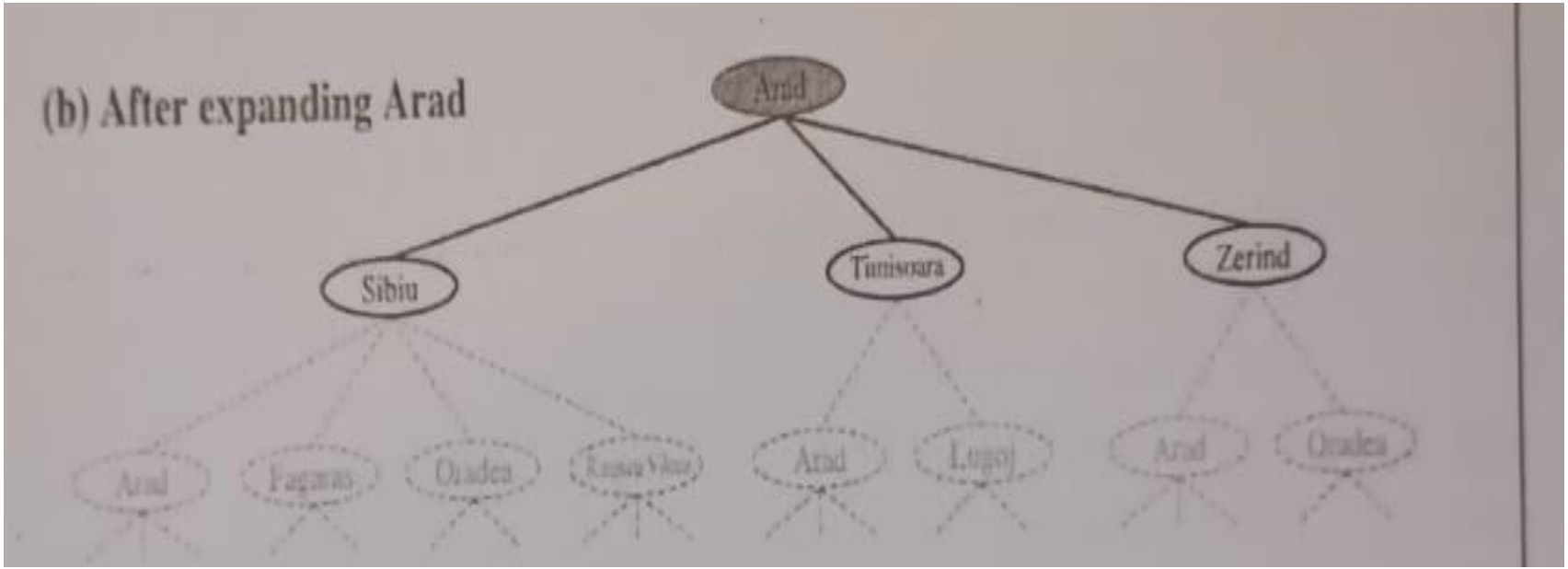
Then we need to consider taking various actions. We do this by expanding the current state; that is, applying each legal action to the current state, thereby generating a new set of states.

In this case, we add three branches from the parent node In(Arad) leading to three new child nodes: In(Sibiu), In(Timișoara), and In(Zerind).

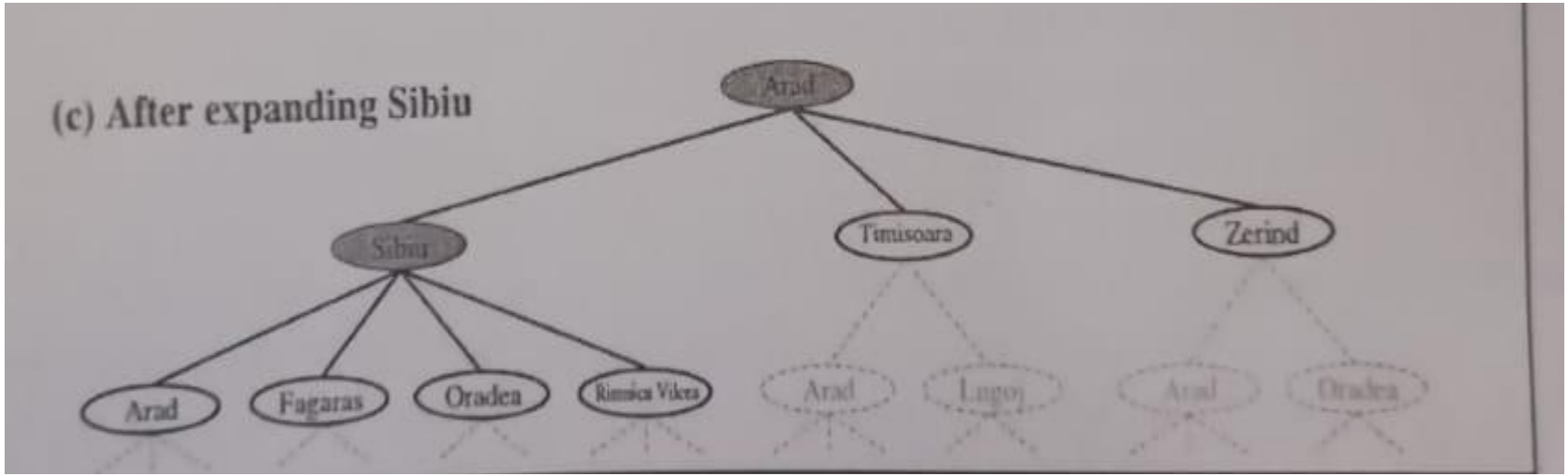
Searching for Solutions



Searching for Solutions



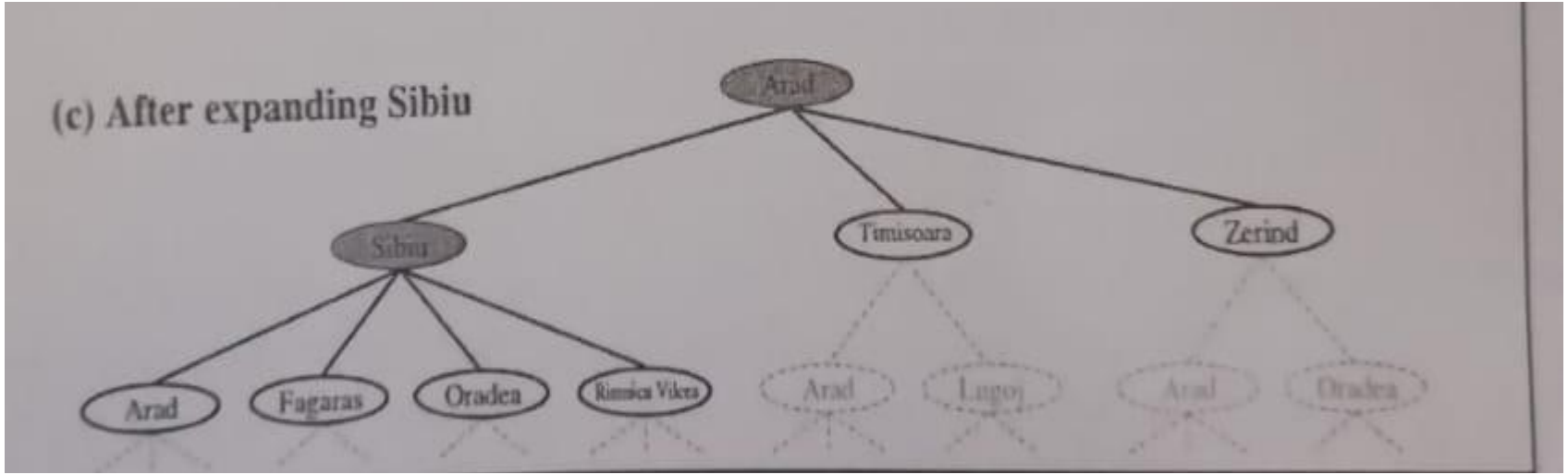
Searching for Solutions



We can then choose any of these four or go back and choose Timisoara or Zerind. Each of these six nodes is a leaf node, that is, a node with no children in the tree. The set of all leaf nodes available for expansion at any given point is called the frontier.

Searching for Solutions

Search algorithms all share this basic structure; **they vary primarily according to how they choose which state to expand next-the so-called search strategy.**



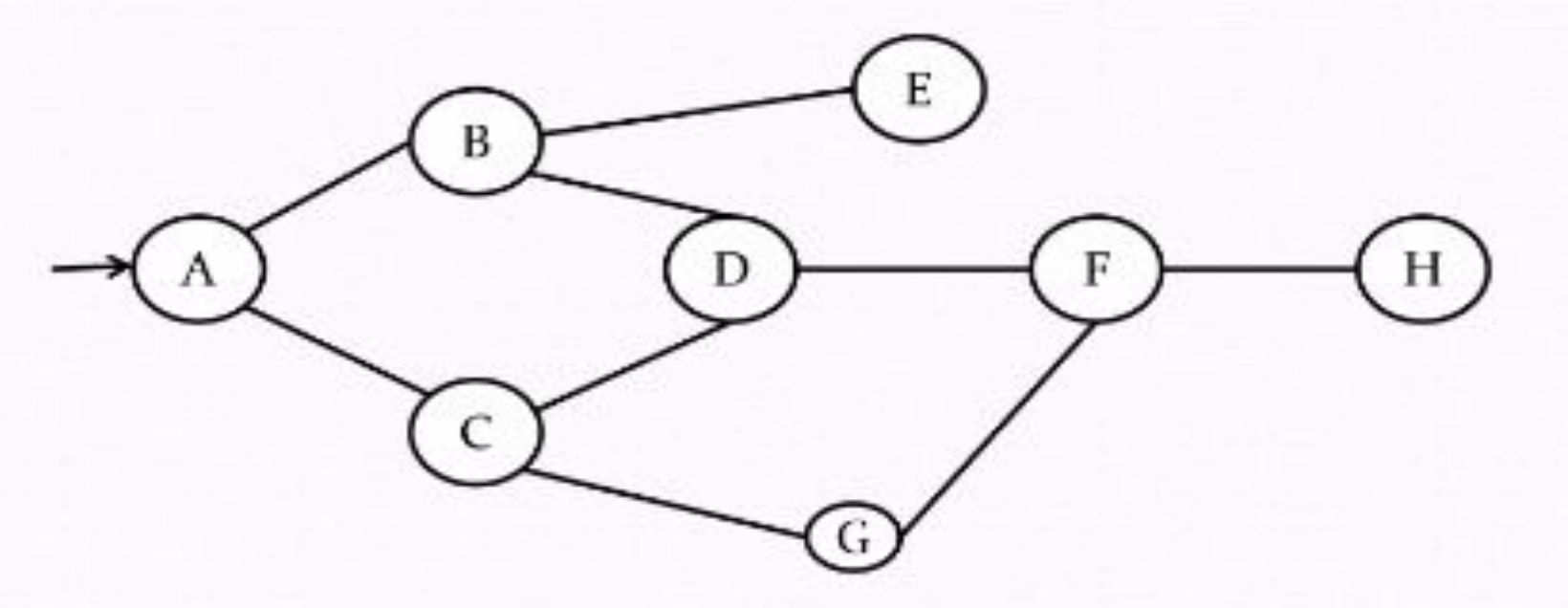
It includes the path from Arad to Sibiu and back to Arad again! We say that In(Arad) is a repeated state in the search tree, generated in this case by a loopy path.

Search Strategies

Uninformed Search	Informed Search
Search Without Information	Search With Information
Blind Search - Brute Force Search	Heuristic Search
No Knowledge	Use Knowledge to find steps to solution
Time Consuming	Quick Solution
More Complexity (Time, Space)	Less Complexity (Time, Space)
Algorithm / Methods : DFS, BFS etc.	Algorithm / Methods : A*, Heuristic DFS, Best First Search

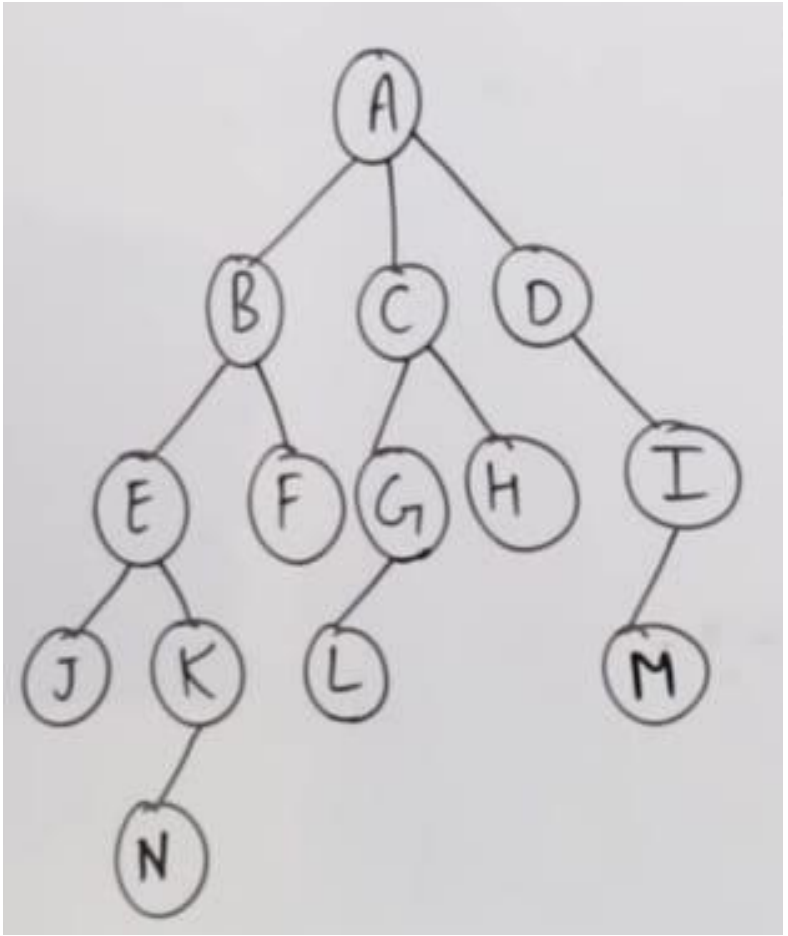
Search Strategies

Uninformed Search



Concept of BFS

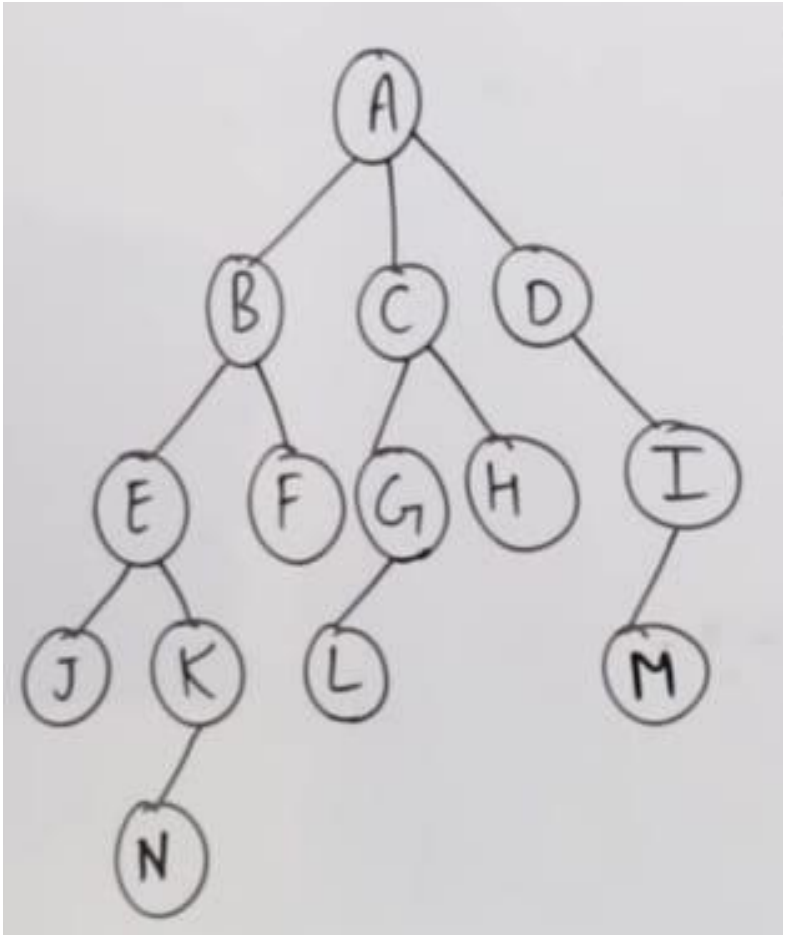
- Uninformed Search Strategy
- FIFO (Queue)
- Shallowest Node - Not Deep (Will go level by level)
- Complete - Will Surely Provide Result
- Optimal
- Time Complexity - O(b^d) where b= Branch Factor (At most Branch), d= depth
- Also Called Level Search Technique



Concept of BFS

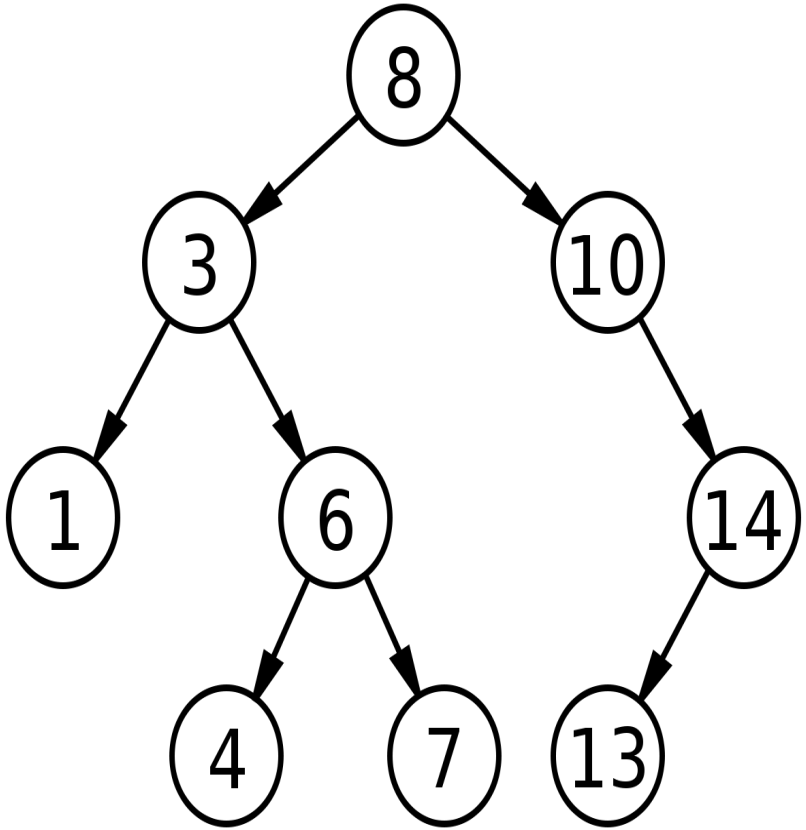
- Traverse Whole Tree Scenario
- Search for Goal (ie “G”).

?



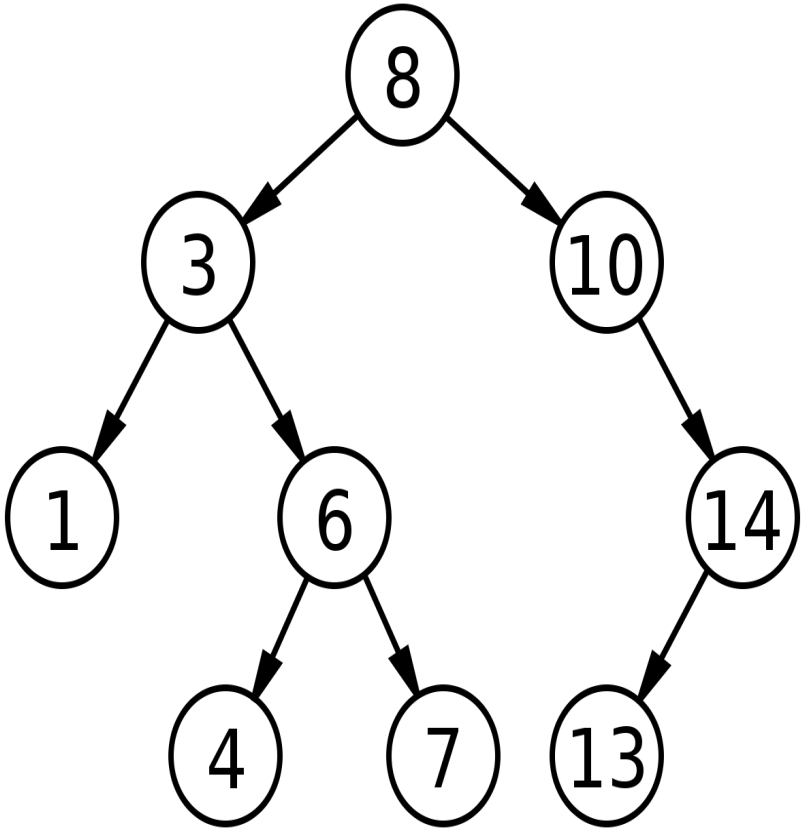
Concept of DFS

- Uninformed Search Strategy
- LIFO (Stack)
- Deepest Node - Deep (Will go in Depth)
- InComplete - When Infinite / Cyclic Graph
- Non Optimal
- Time Complexity - O(b^d) where b= Branch Factor (At most Branch), d= depth
- Also Called Depth Search Technique

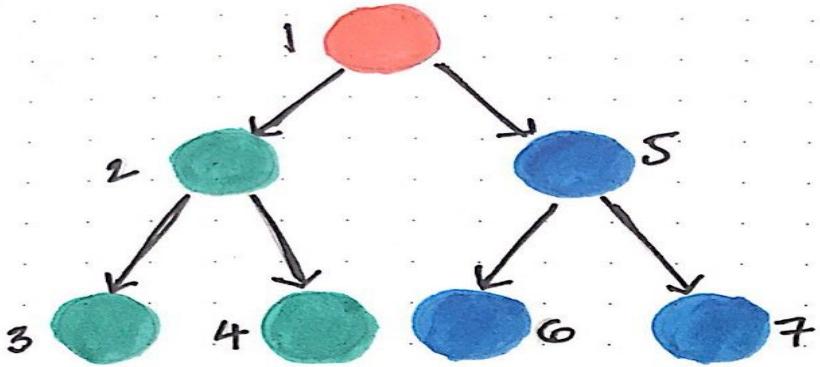


Concept of DFS

- Traverse Whole Tree Scenario
- Search for Goal (ie “1”).

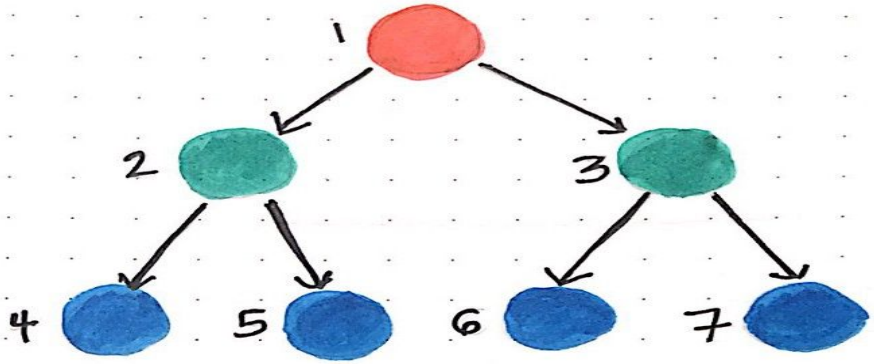


DFS and BFS



Depth-first search

- Traverse through left subtree(s) first, then traverse through the right subtree(s).



Breadth-first search

- Traverse through one level of children nodes, then traverse through the level of grandchildren nodes (and so on...).

Depth Limited Search

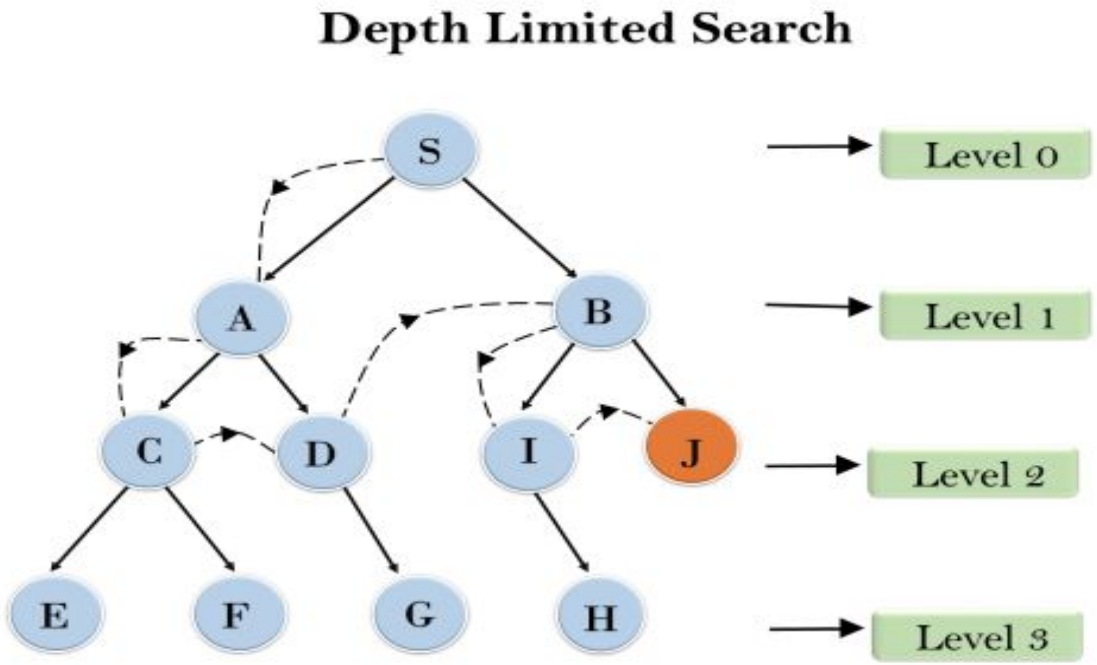
A depth-limited search algorithm **is similar to depth-first search** with a **predetermined limit.**

Depth-limited search **can solve the drawback of the infinite path** in the Depth-first search. In this algorithm, the node at the depth limit will treat as it has no successor nodes further.

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.**

Depth Limited Search

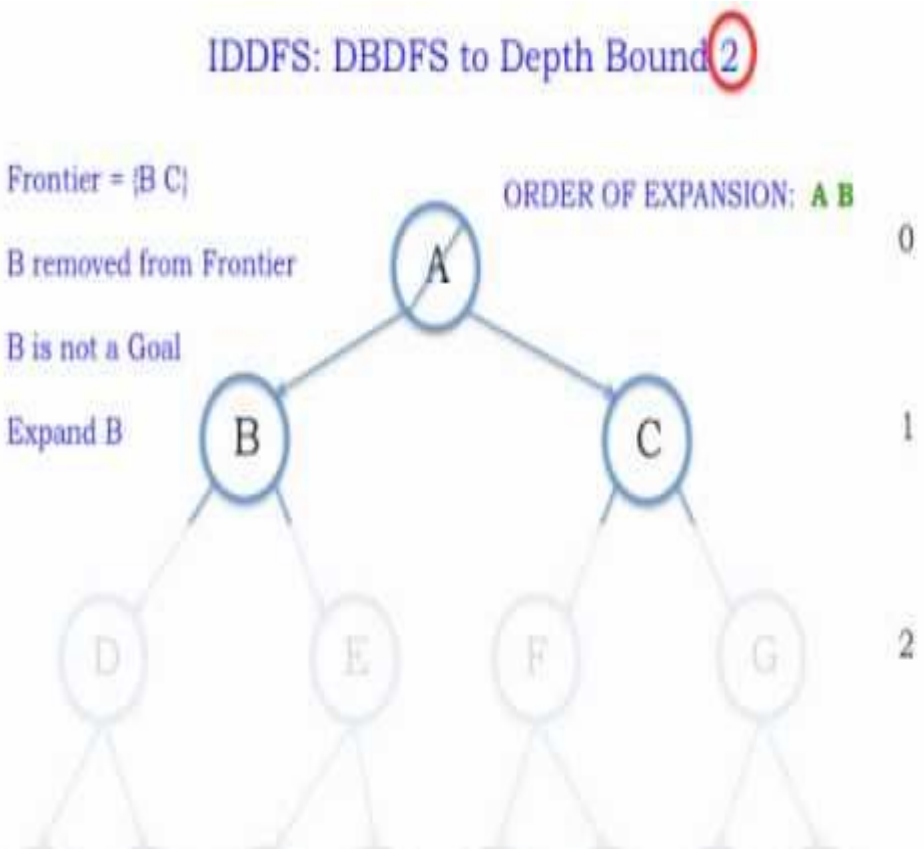


Completeness: DLS search algorithm is complete if the solution is above the depth-limit.

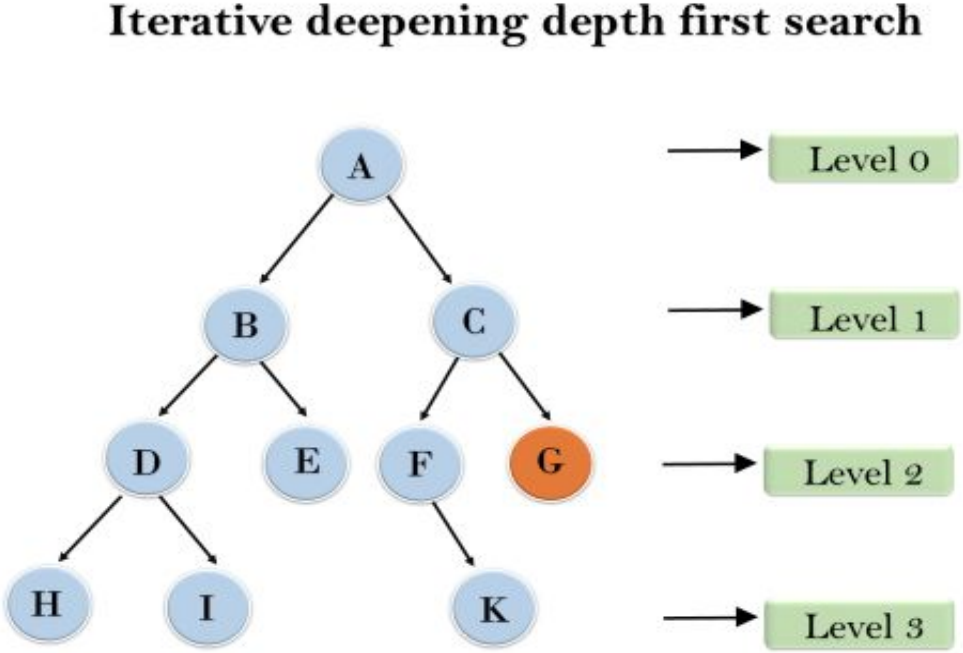
Iterative Deepening DFS

The iterative deepening algorithm is a **combination of DFS and BFS algorithms.** This search algorithm finds out the **best depth limit and does it by gradually increasing the limit until a goal is found.**

This algorithm performs depth-first search up to a certain "depth limit", and **it keeps increasing the depth limit after each iteration until the goal node is found.**



Iterative Deepening DFS



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.

Bidirectional Search

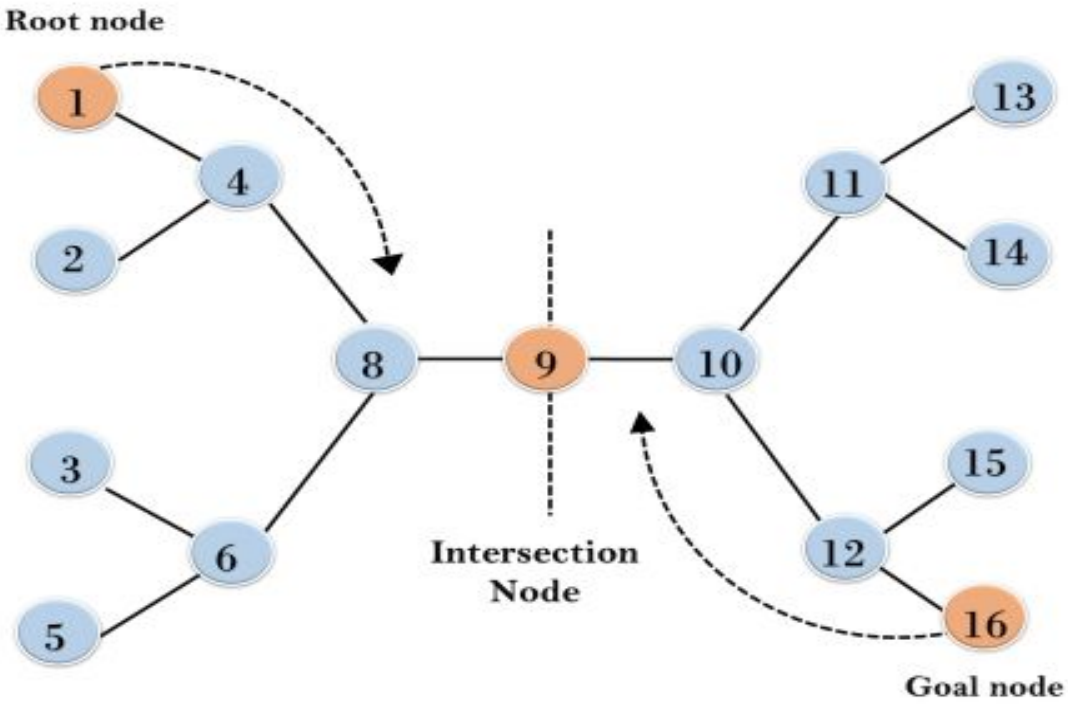
Bidirectional search algorithm runs two simultaneous searches, **one form initial state called as forward-search** and **other from goal node called as backward-search, to find the goal node.**

Bidirectional search **replaces one single search graph with two small subgraphs** in which one starts the search from an initial vertex and other starts from goal vertex.

The search stops when these two graphs intersect each other.

Bidirectional search can use search techniques such as BFS, DFS, DLS, etc.

Bidirectional Search



Completeness: Bidirectional Search is complete if we use BFS in both searches.

Time Complexity: Time complexity of bidirectional search using BFS is $\mathbf{O(b^{d/2})}$.

Space Complexity: Space complexity of bidirectional search is $\mathbf{O(b^{d/2})}$.

Informed (Heuristic) Search Strategies

Heuristic is a function which is used in Informed Search, and it finds the most promising path.

It takes the **current state of the agent as its input** and **produces the estimation of how close agent is from the goal.**

The heuristic method, however, might not always give the best solution, **but it guaranteed to find a good solution in reasonable time.**

Heuristic function **estimates how close a state is to the goal.**

It is represented by h(n), and it calculates the cost of an optimal path between the pair of states.

The value of the heuristic function is always positive.

Concept of Greedy BFS

Greedy best-first search algorithm **always selects the path which appears best at that moment.**

It is the **combination of depth-first search and breadth-first search algorithms.**

It uses the heuristic function and search.

Best-first search allows us to take the advantages of both algorithms.

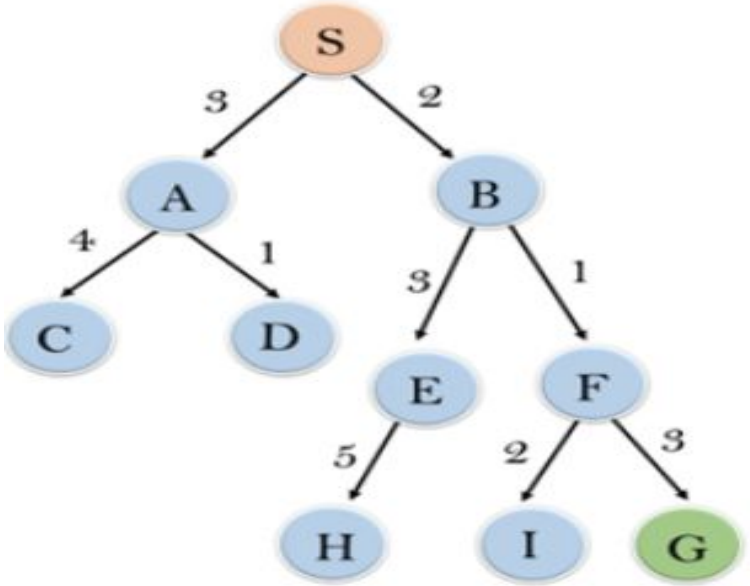
Concept of Greedy BFS

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Concept of Greedy BFS

Consider the below search problem, and we will traverse it using greedy best-first search.

At each iteration, each node is expanded using evaluation function **f(n)=h(n)** , which is given in the below table.

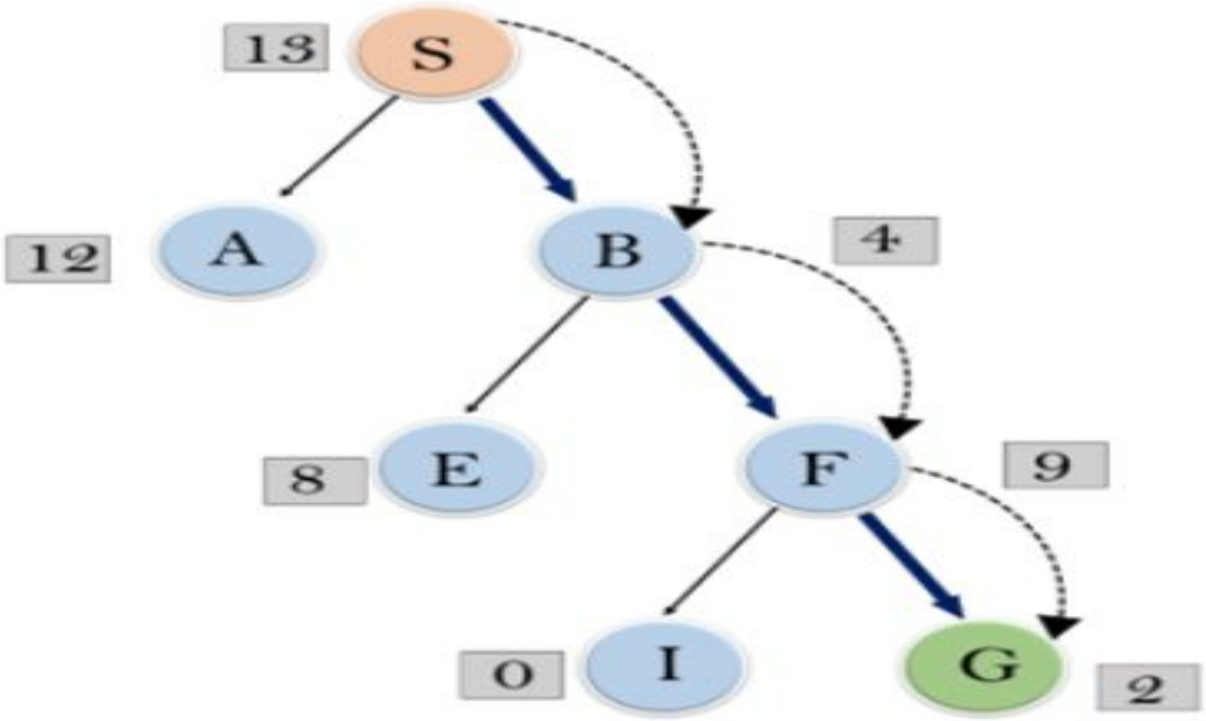


node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

Concept of Greedy BFS

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists.

Following are the iteration for traversing the above example.



Concept of Greedy BFS

Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]

: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]

: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

A* Search : Minimizing the total estimated solution cost

A* search is the most commonly known form of best-first search.

It uses **heuristic function h(n)**, and **cost to reach the node n from the start state g(n).**

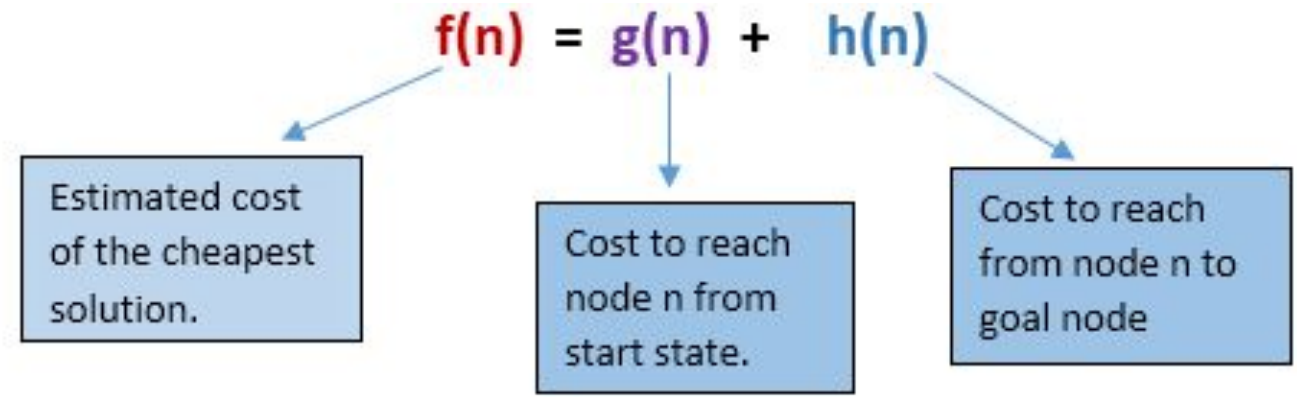
It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently.

A* search algorithm finds the shortest path through the search space using the heuristic function.

This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except **that it uses g(n)+h(n) instead of g(n).**

A* Search : Minimizing the total estimated solution cost

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**..



A* Search : Minimizing the total estimated solution cost

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

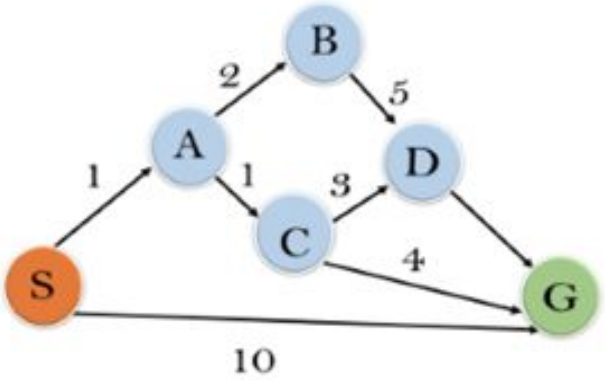
Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

A* Search : Minimizing the total estimated solution cost



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Initialization: {(S, 5)}

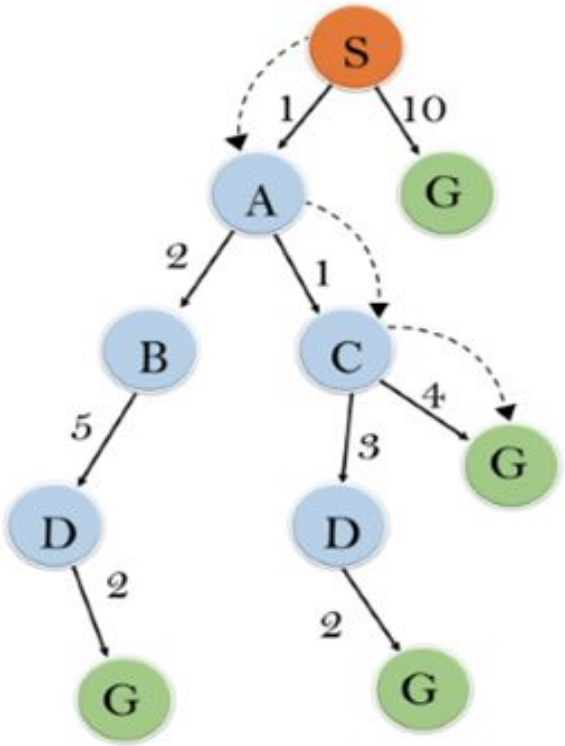
Iteration 1: {(S--> A, 4), (S-->G, 10)}

Iteration 2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration 3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

A* Search : Minimizing the total estimated solution cost



Initialization: {(S, 5)}

Iteration 1: {(S--> A, 4), (S-->G, 10)}

Iteration 2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration 3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

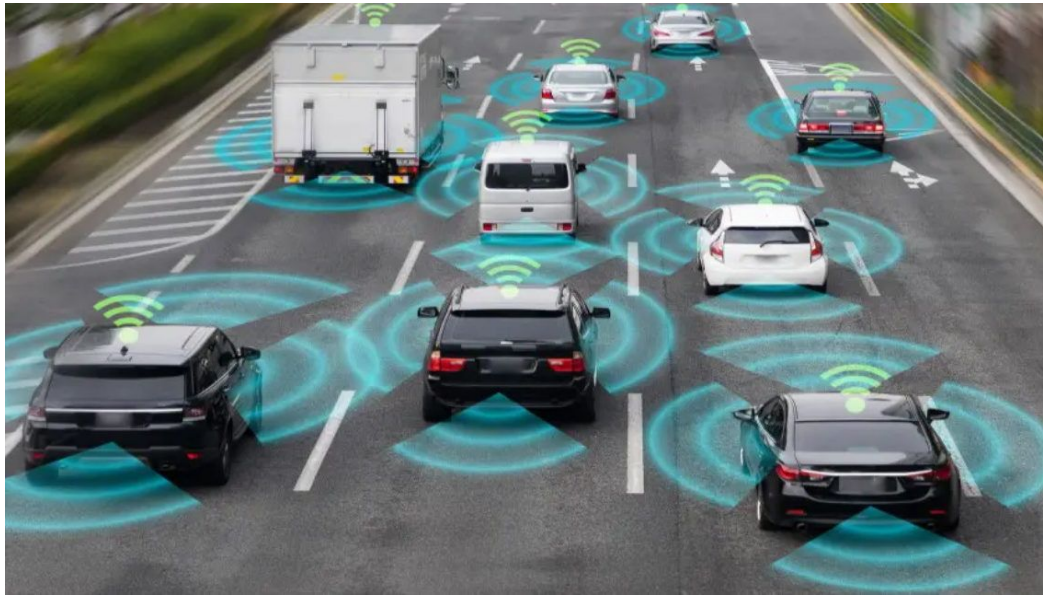
Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

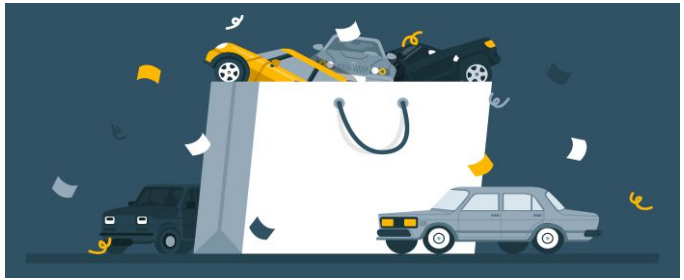
Case Study: Application of AI in Transportation

Artificial Intelligence is changing the transportation field.

It helps **cars, trucks, trains, ships and airplanes to function automatically to make traffic smooth, safe, clean, smart and more efficient.**

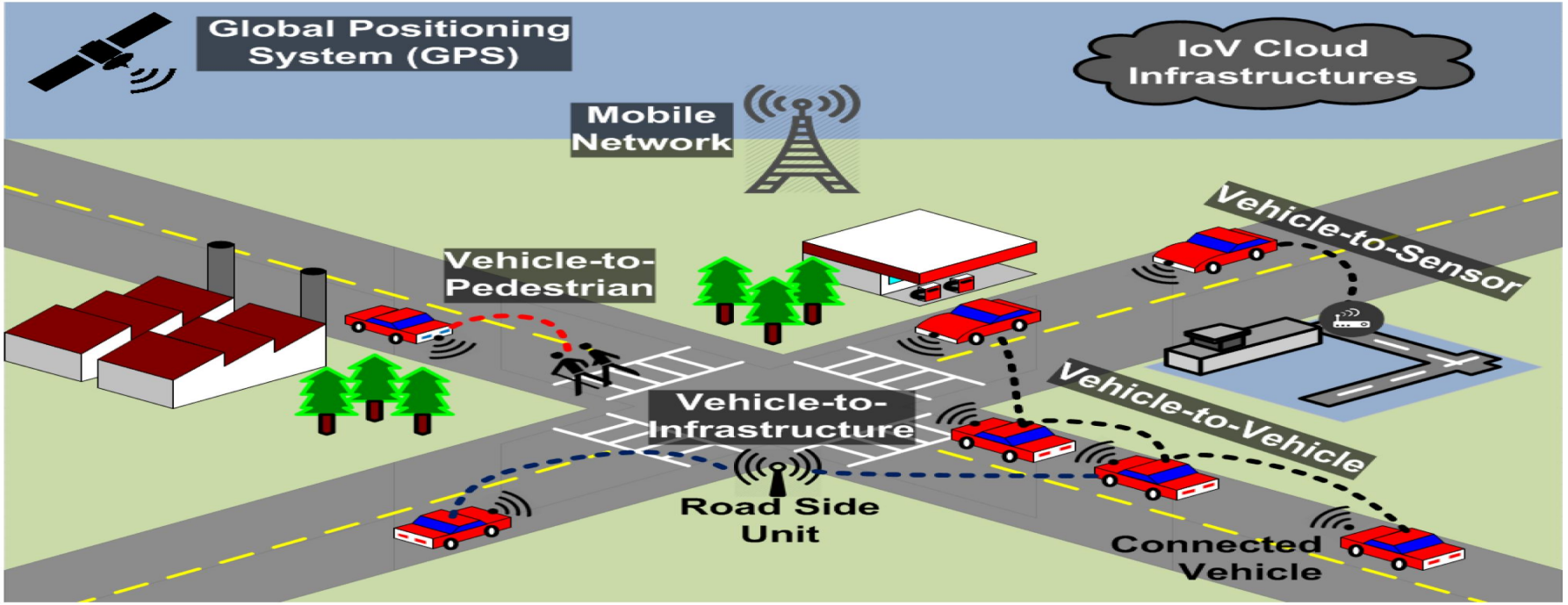
Due to increase of population, **the evolution of transportation increases and it increase the complexity to manage and to analyze the generated data in transportation.**





Case Study: Application of AI in Transportation

Intelligent transportation system (ITS)



Case Study: Application of AI in Transportation

To simplify and to maintain the complexity of such requirements AI is being used.

AI-based techniques are applied in different form and in different areas whether it is road, aviation, waters or railways. AI helps all transportation safer and smarter.

It **also reduces human errors which involve traffic accidents and many more.**

The use of AI in transportation is also known as **intelligent transportation systems (ITS)** which **include simplification of huge amount of data generated during transportation** whether it is from vehicles as well as drivers.

Case Study: Application of AI in Transportation

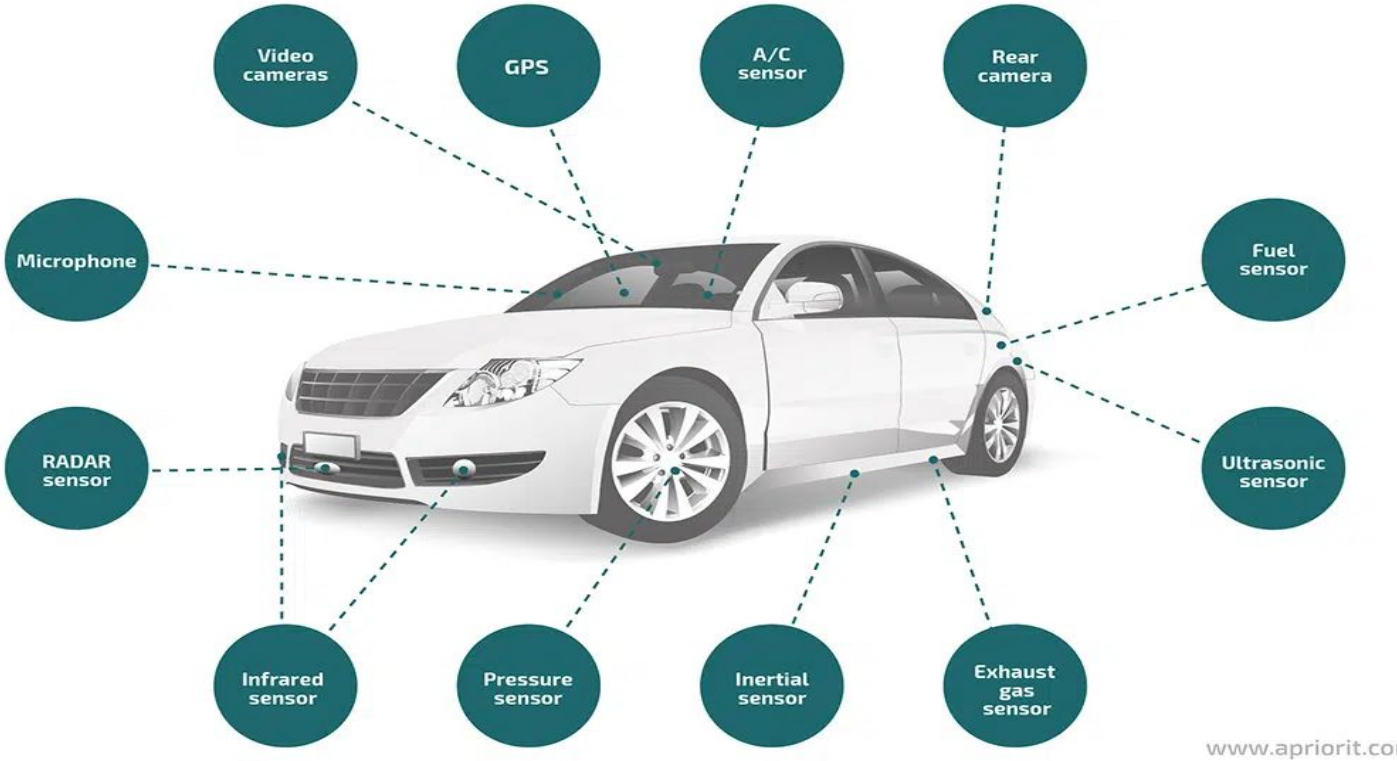
Transportation is the basic living standard for the present life. **About 305 of the humans there are many methods to spent approx 2 hours of time in a day in travelling.**

In improve the use of transportation system.

Particularly the transportation use **AI as self control vehicles, predictions of path, and predictions of traffics.**

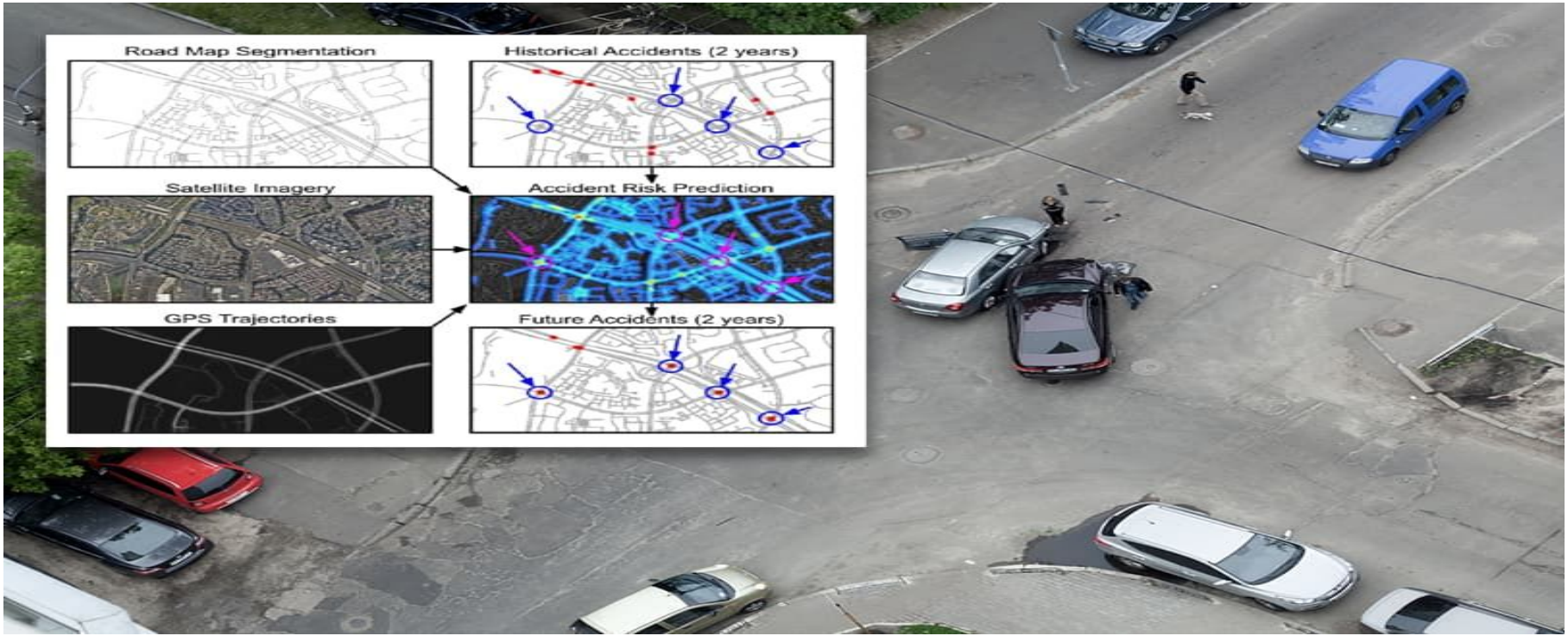
Case Study: Application of AI in Transportation

Vehicle Control System



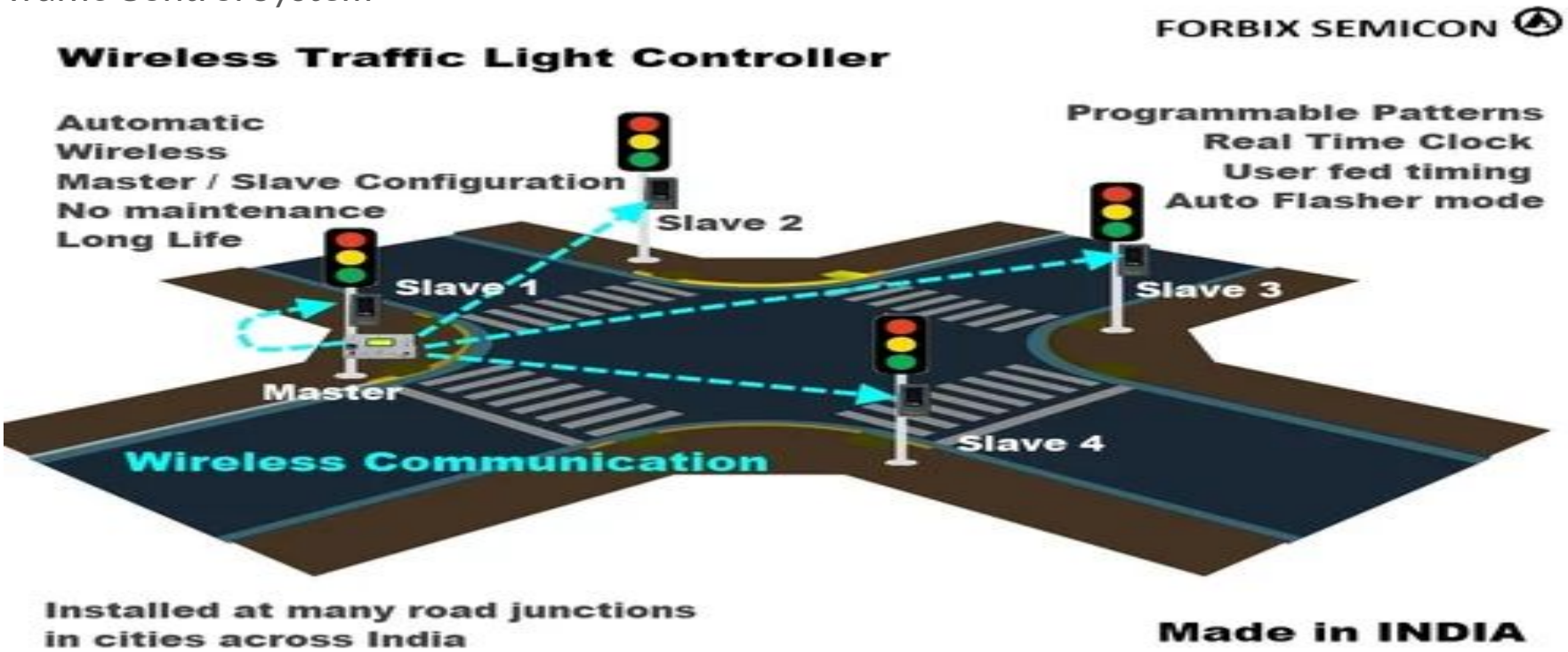
Case Study: Application of AI in Transportation

Accident Prediction System



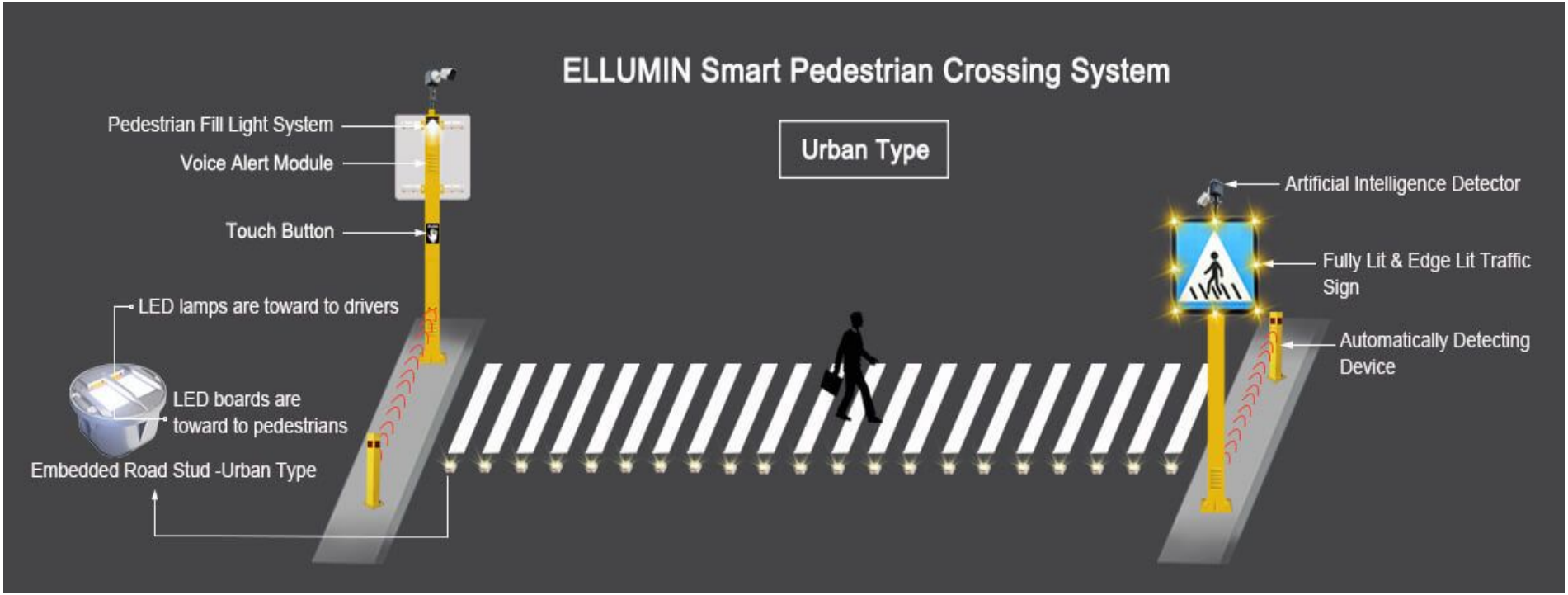
Case Study: Application of AI in Transportation

Traffic Control System



Case Study: Application of AI in Transportation

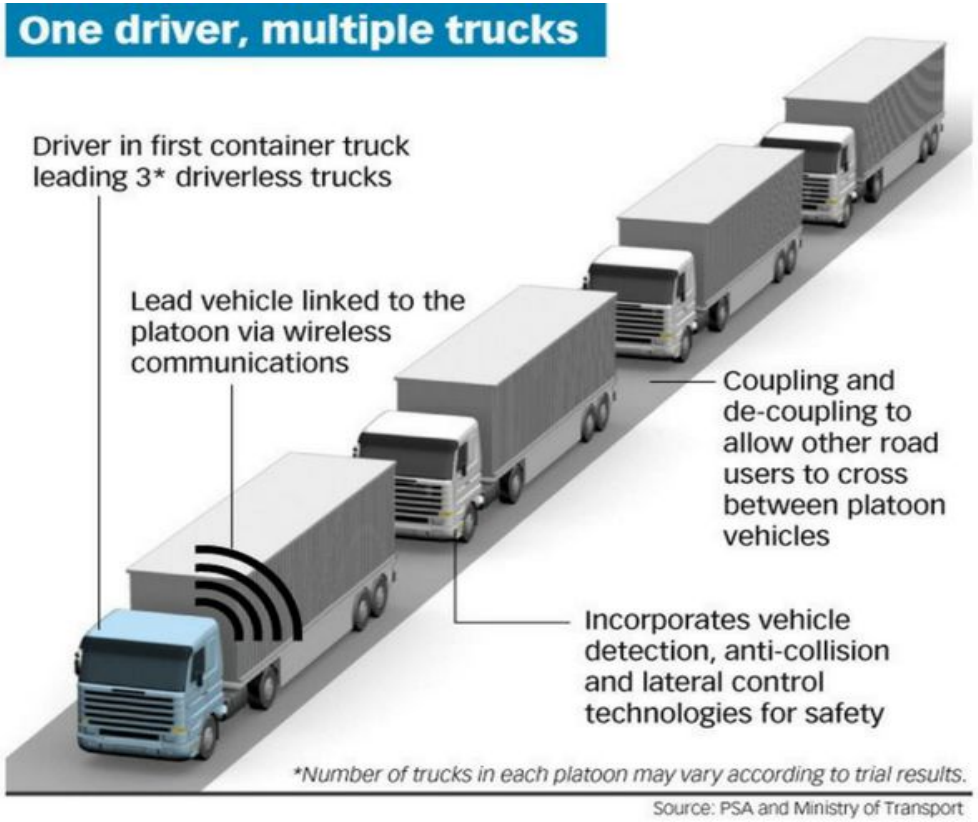
Traffic Pedestrian System



Case Study: Application of AI in Transportation

AI in Roads - illustration of truck platooning

AI makes **truck platooning**, let **4 vehicles are inline the driver of the first vehicle can control other three trucks** or allowing them to let go of the steering wheel and **let the truck steer by own or it is also called as automatic driving system. The distance between the vehicles is about to 10-12 meters. This method is very helpful which cause less congestion.**



Case Study: Application of AI in Transportation

AI in Aviation - illustration of automated pilot wars

It's about **20 million of flights in 2018**. AI is not new in the **aviation industry by automating driving, brake, traffic predictions.** AI helps aviation systems for safe and secure. **AI makes aviation business helpful with machine learning it predict the data what the customer want through google or social media and adjust the content according to customers. Some highly uses cases are fuelling, catering, loading-unloading passengers, baggage control, emails and security.**

Case Study: Application of AI in Transportation

AI in Railways

Railways are the most innovative sectors of economy and major industrial revolution. **Railways are the second biggest network after web. AI improved infrastructure, manufacture, maintenance and other work for rail transports.** It helps with lower cost, safe, effective and a big competition to other modes of transportation. **Automated metro trains is the best use case of AI in railways where the source, the distance, point of stay, minute of stay reduce the complexity of this use case.**

Thank You !!