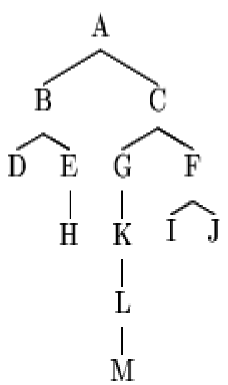
USN | | | | | | | | | | |

Internal Assessment Test 1 – June 2024

Sub:	**Artificial Intelligence**			Sub Code:	BAD402	Branch:	AIML,CSE-AIML
Date:	**26/3/2025**	Duration:	90 min	Max Marks: 50	Sem/Sec:	IV /A, B & C	OBE

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Define an intelligent agent and describe its interaction with the environment. Provide a suitable example and present the percept sequence and corresponding actions in a table.	10	CO1	L1
2a	Explain in detail breadth first search, depth first search, Iterative deepening depth first search and bi-directional search.	8	CO3	L2
2b	Compare the uninformed search strategies from Question 2a based on completeness, optimality, time, and space complexity.	2	CO3	L2
3	Illustrate with a suitable algorithm for the following:i)Uniform Cost Search ii)Depth Limited Tree Search.And measure the performance of the algorithms.	10	CO3	L2
4a)	Explain A* optimality and its required conditions.	5	CO2	L2
4b)	In the following search tree with start state A and goal state M, find the shortest path and optimal path cost using A*algorithm.	5	CO2	L3



Edge	Edge Cost
$A \rightarrow B$	5
$A \rightarrow C$	2
$B \rightarrow D$	8
$B \rightarrow E$	7
$C \rightarrow F$	9
$C \rightarrow G$	1
$E \rightarrow H$	4
$F \rightarrow I$	3
$F \rightarrow J$	4
$G \rightarrow K$	3
$K \rightarrow L$	5
$L \rightarrow M$	4

Node	Heuristic Cost
A	13
B	11
C	11
D	14
E	16
F	18
G	10
H	14
I	17
J	15
K	8
L	2
M	0

5	Explain problem-solving agents with an algorithm. Describe the five components of problem formulation using the Romania map example, where the agent starts in Arad and aims to reach Bucharest as the goal.	10	CO2	L3

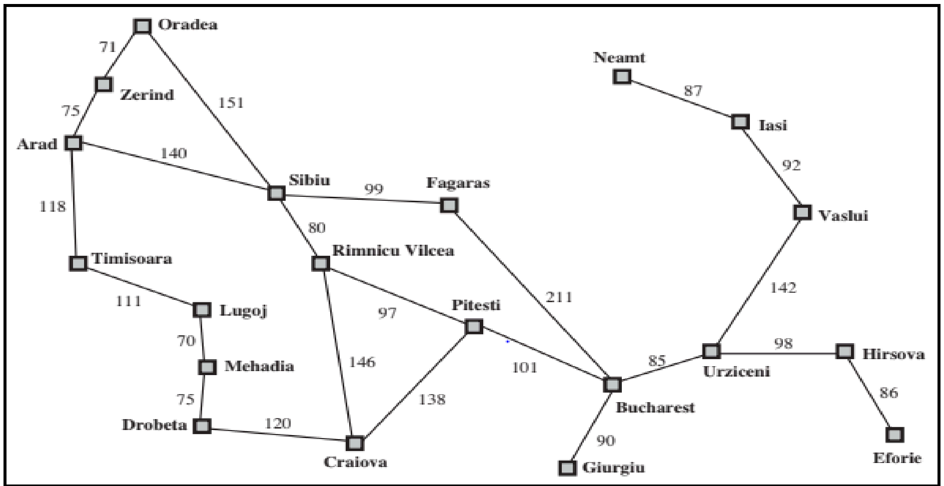
<table>
<tr>
<td></td>
<td>


Figure 3.2 A simplified road map of part of Romania.
</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Compare the following with atleast 4 differences: i)Discrete VS Continuous environments ii)static VS dynamic environments iii)Episodic VS Sequential environments iv)Deterministic vs. Stochastic environments v)Single agent VS Multi agents</td>
<td>10</td>
<td>CO2</td>
<td>L2</td>
</tr>
</table>

Faculty Signature CCI Signature HOD Signature

USN | | | | | | | | | | |

Internal Assessment Test 1 – June 2024

Sub:	**Artificial Intelligence**			Sub Code:	BAD402	Branch:	AIML,CSE-AIML	
Date:	**26/3/2025**	Duration:	90 min	Max Marks: 50	Sem/Sec:	IV /A, B & C		OBE

	Answer any FIVE FULL Questions **SCHEME AND SOLUTION**	MARKS	CO	RBT
1	Define an intelligent agent and describe its interaction with the environment. Provide a suitable example and present the percept sequence and corresponding actions in a table.	10	CO1	L1

SOLn

An intelligent agent is stated as an agent that takes the best possible action in a situation.

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.

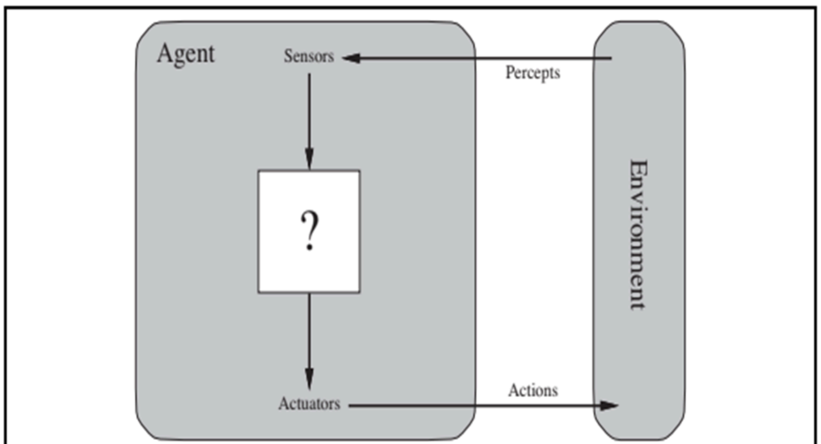


Figure 2.1 Agents interact with environments through sensors and actuators.

¢Sensors help the agent perceive the environment.Example: Eyes and ears for humans, cameras for robots.

¢Actuators help the agent act in the environment.Example: Hands and legs for humans, motors for robots.

¢A percept is what the agent senses at a single moment.A percept sequence is the complete history of everything the agent has perceived.Example: A self-driving car perceives obstacles, traffic signals, and road signs over time.

¢Agent Function: A theoretical concept—it decides what action to take based on the percept sequence.Agent Program: The actual code or system that implements the agent function in a real-world system.

The vacuum-cleaner world Example.

¢**The Vacuum World**-The world has only **two locations**, **A** and **B**. The **vacuum cleaner agent** can sense which square it is in and whether there is **dirt**.

¢**Actions**-Move left,Move right,Suck up dirt,Do nothing.

¢Simple agent function rule-If the square is **dirty**, suck up the dirt. Otherwise, move to the **other square**.

Different vacuum cleaner programs can be created by changing how the agent responds to situations-way defining a smart agent

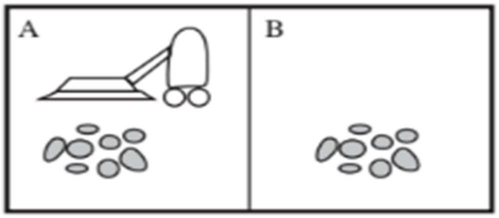


Figure 2.2 A vacuum-cleaner world with just two locations.

Percept sequence	Action
[*A*, *Clean*]	*Right*
[*A*, *Dirty*]	*Suck*
[*B*, *Clean*]	*Left*
[*B*, *Dirty*]	*Suck*
[*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮

Figure 2.3 Partial tabulation of a simple agent function for the vacuum-cleaner world shown in Figure 2.2.

2a	Explain in detail breadth first search, depth first search, Iterative deepening depth first search and bi-directional search.	8	CO3	L2

SOLn

i)**Breadth-first search** is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then *their* successors, and so on. In general, all the nodes are expanded at a given depth in the search tree before any nodes at the next level are expanded.

Breadth-first search is an instance of the general graph-search algorithm (Figure 3.7) inwhich the *shallowest* unexpanded node is chosen for expansion. This is achieved very simplyby using a FIFO queue for the frontier.
Thus, new nodes (which are always deeper than theirparents) go to the back of the queue, and old nodes, which are shallower than the new nodes,get expanded first.
There is one slight tweak on the general graph-search algorithm, which isthat the goal test is applied to each node when it is *generated* rather than when it is selected for expansion.

- **Time Complexity:** In the worst case, it is the last node generated at that level. Then the total number of nodes generated is $b + b^2 + b^3 + \cdots + b^d = O(b^d)$.
- Breadth-first search is optimal because it always expands the *shallowest* unexpanded node.
- The memory requirements are a bigger problem for breadth-first search than is execution time.
- Exponential-complexity search problems cannot be solved by uniformed methods for any but the smallest instances.

Advantages:

- BFS will provide a solution if any solution exists.
- If there are more than one solutions 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.*

ii)**Depth-first search** always expands the *deepest* node in the current frontier of the search tree.The progress of the search is illustrated in Figure 3.16.

The search proceeds immediately to the deepest level of the search tree, where the nodes have no successors.

As those nodes are expanded, they are dropped from the frontier, so then the search "backs up" to the next deepest node that still has unexplored successors.

Depth-first search uses a LIFO queue. A LIFO queue means that the most recently generated node is chosen for expansion.

This must be the deepest unexpanded node because it is one deeper than its parent—which, in turn, was the deepest unexpanded node when it was selected.

The **time complexity** of depth-first graph search is bounded by the size of the state space (which may be infinite). Generate all of the $O(b^m)$ nodes in the search tree, where m is the maximum depth of any node.

Advantages:

DFS requires very less memory as it only needs to store a stack of the nodes on the path from root node to the current node.

It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantages:

There is the possibility that many states keep re occurring, and there is no guarantee of finding the solution.

DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

iii)**Iterative deepening search(IDS)** (or iterative deepening depth-first search) is a general strategy,often used in combination with depth-first tree search, that finds the best depth limit. It doesthis by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found.This will occur when the depth limit reaches d, the depth of the shallowest goal node.

The algorithm is shown in Figure 3.18. Iterative deepening combines the benefits of depth-first and breadth- first search.

Like depth-first search, its memory requirements are modest: $O(bd)$to be precise.

Like breadth-first search, it is complete when the branching factor is finite andoptimal when the path cost is a non decreasing function of the depth of the node.

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

Disadvantages: The main drawback of IDS is states are generated multiple times.

In an iterativedeepening search, the nodes on the bottom level (depth d) are generated once, those on the next-to-bottom level are generated twice, and so on, up to the children of the root, which aregenerated d times. So the total number of nodes generated in the worst case is

$$N(\text{IDS}) = (d)b + (d-1)b^2 + \cdots + (1)b^d\,,$$

which gives a **time complexity of $O(b^d)$**—asymptotically the same as breadth-first search. There is some extra cost for generating the upper levels multiple times, but it is not large.

Forexample, if $b = 10$ and $d = 5$, the numbers are

$$N(\text{IDS}) = 50 + 400 + 3,000 + 20,000 + 100,000 = 123,450$$
$$N(\text{BFS}) = 10 + 100 + 1,000 + 10,000 + 100,000 = 111,110\,.$$

iv)**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.

2b	Compare the uninformed search strategies from Question 2a based on completeness, optimality, time, and space complexity.	2	CO3	L2

SOLn

Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Complete?	Yes[a]	Yes[a,b]	No	No	Yes[a]	Yes[a,d]
Time	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^m)$	$O(b^\ell)$	$O(b^d)$	$O(b^{d/2})$
Space	$O(b^d)$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(bm)$	$O(b\ell)$	$O(bd)$	$O(b^{d/2})$
Optimal?	Yes[c]	Yes	No	No	Yes[c]	Yes[c,d]

3	Illustrate with a suitable algorithm for the following:i)Uniform Cost Search ii)Depth Limited Tree Search.And measure the performance of the algorithms.	10	CO3	L2

SOLn i)Uniform Cost Search-

```
function UNIFORM-COST-SEARCH(problem) returns a solution, or failure
    node ←a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
      frontier ← a priority queue ordered by PATH-COST, with node as the only
element
    explored ← an empty set
    loop do
       if EMPTY?(frontier ) then return failure
       node ← POP(frontier ) /* chooses the lowest-cost node in frontier */
       if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
      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
              frontier ← INSERT(child,frontier )
         else if child.STATE is in frontier with higher PATH-COST then
              replace that frontier node with child
```

ii)Depth Limited Tree Search

```
function DEPTH-LIMITED-SEARCH(problem, limit) returns a solution, or
failure/cutoff
    return RECURSIVE-DLS(MAKE-NODE(problem.INITIAL-STATE), problem,
limit)
function RECURSIVE-DLS(node, problem, limit) returns a solution, or
failure/cutoff
```

```
 if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
else if limit = 0 then return cutoff
else
    cutoff occurred?←false
    for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        result ← RECURSIVE-DLS(child, problem, limit − 1)
        if result = cutoff then cutoff occurred?← true
       else if result = failure then return result
    if cutoff occurred? then return cutoff else return failure
```

Criterion	Uniform-Cost	Depth-Limited
Complete?	Yes^{a,b}	No
Time	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^\ell)$
Space	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b\ell)$
Optimal?	Yes	No

4a) SOLn	Explain A* optimality and its required conditions.	5	CO2	L2

The most widely known form of best-first search is called **A* search** (pronounced "A-star search"). It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the cost to get from the node to the goal:

$$f(n) = g(n) + h(n) .$$

Since $g(n)$ gives the path cost from the start node to node n, and $h(n)$ is the estimated cost of the cheapest path from n to the goal, we have

$$f(n) = \text{estimated cost of the cheapest solution through } n .$$

Thus, if we are trying to find the cheapest solution, a reasonable thing to try first is the node with the lowest value of $g(n) + h(n)$. It turns out that this strategy is more than just reasonable: provided that the heuristic function $h(n)$ satisfies certain conditions, A* search is both complete and optimal.

Optimality of A*

A* has the following properties: *the tree-search version of A* is optimal if $h(n)$ is admissible, while the graph-search version is optimal if $h(n)$ is consistent.*

The first step is to establish the following: *if $h(n)$ is consistent, then the values of*

$f(n)$ along any path are nondecreasing. The proof follows directly from the definition of consistency. Suppose n^t is a successor of n; then $g(n^t) = g(n) + c(n, a, n^t)$ for some action a, and we have

$$f(n^t) = g(n^t) + h(n^t) = g(n) + c(n, a, n^t) + h(n^t) \geq g(n) + h(n) = f(n) .$$

The next step is to prove that *whenever A* selects a node n for expansion, the optimal path to that node has been found.* Were this not the case, there would have to be another frontier node n^t on the optimal path from the start node to n, by the graph separation property of GRAPH-SEARCH; because f is nondecreasing along any path, n^t would have lower f-cost than n and would have been selected first.

The fact that f-costs are nondecreasing along any path also means that we can draw **contours** in the state space, just like the contours in a topographic map. Figure 3.25 shows an example. Inside the contour labeled 400, all nodes have $f(n)$ less than or equal to 400, and so on. Then, because A^* expands the frontier node of lowest f-cost, we can see that an A^* search fans out from the start node, adding nodes in concentric bands of increasing f-cost.

If C^* is the cost of the optimal solution path, then we can say the following:

- A^* expands all nodes with $f(n) < C^*$.
- A^* might then expand some of the nodes right on the "goal contour" (where $f(n) = C^*$) before selecting a goal node

Completeness requires that there be only finitely many nodes with cost less than or equal to

C^*, a condition that is true if all step costs exceed some finite

E and if b is finite. Notice that A^* expands no nodes with $f(n)$

$> C^*$

Algorithms that extend search paths from the root and use the same heuristic information—A^* is **optimally efficient** for any given consistent heuristic. That is, no other optimal algorithm is guaran- teed to expand fewer nodes than A^* (except possibly through tie-breaking among nodes with $f(n)= C^*$). This is because any algorithm that *does not* expand all nodes with $f(n) < C^*$ runs the risk of missing the optimal solution.

Conditions for optimality: Admissibility and consistency

1. Admissible heuristic

The first condition we require for optimality is that $h(n)$ be an **admissible heuristic**. An admissible heuristic is one that *never overestimates* the cost to reach the goal. Because $g(n)$ is the actual cost to reach n along the current path, and $f(n)= g(n) + h(n)$, we have as an immediate consequence that $f(n)$ never overestimates the true cost of a solution along the current path through n.

Admissible heuristics are by nature optimistic because they think the cost of solving the problem is less than it actually is. An obvious example of an admissible heuristic is the straight- line distance h_{SLD} that we used in getting to Bucharest.

Straight-line distance is admissible because the shortest path between any two points is a straight line, so the straight line cannot be an overestimate. In Figure 3.24, we show the progress of an A^* tree search for Bucharest. The values of g are computed from the step costs in Figure 3.2, and the values of h_{SLD} are given in Figure 3.22. Notice in particular that Bucharest first appears on the frontier at step (e), but it is not selected for expansion because its f-cost (450) is higher than that of Pitesti (417). Another way to say this is that there *might* be a solution through Pitesti whose cost is as low as 417, so the algorithm will not settle for a solution that costs 450.

2.Consistency

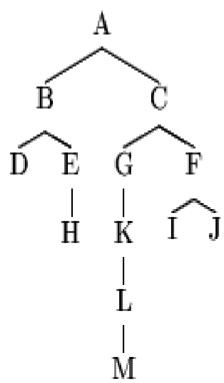
A second, slightly stronger condition called consistency (or sometimes monotonicity) is required only for applications of A∗ to graph search. A heuristic h(n) is consistent if, for every node n and every successor n^t of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n^t plus the estimated cost of reaching the goal from n^t:

$$h(n) \leq c(n, a, n^t) + h(n^t) .$$

This is a form of the general **triangle inequality**, which stipulates that each side of a triangle cannot be longer than the sum of the other two sides. Here, the triangle is formed by n, n^t, and the goal G_n closest to n.

For an admissible heuristic, the inequality makes perfect sense: if there were a route from n to G_n via n^t that was cheaper than $h(n)$, that would violate the property that $h(n)$ is a lower bound on the cost to reach G_n.

4b | In the following search tree with start state A and goal state M, find the shortest path and optimal path cost using A*algorithm. | 5 | CO2 | L3

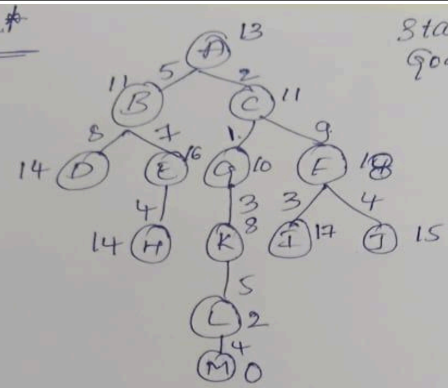


Edge	Edge Cost
$A \rightarrow B$	5
$A \rightarrow C$	2
$B \rightarrow D$	8
$B \rightarrow E$	7
$C \rightarrow F$	9
$C \rightarrow G$	1
$E \rightarrow H$	4
$F \rightarrow I$	3
$F \rightarrow J$	4
$G \rightarrow K$	3
$K \rightarrow L$	5
$L \rightarrow M$	4

Node	Heuristic Cost
A	13
B	11
C	11
D	14
E	16
F	18
G	10
H	14
I	17
J	15
K	8
L	2
M	0

SOLn)

4b) A*



Start State = A
Goal State = M

$f(n) = g(n) + h(n)$.

→ A to B

$f(B) = g(B) + h(B)$

$f(B) = 5 + 13 = 18$

→ A to C.

$f(C) = g(C) + h(C)$

$f(C) = 2 + 11 = 13$

→ Node 'C' has minimum path cost. So Choose 'C' among B and 'C'. So the Successor nodes of 'C' are 'G' and 'F'

→ ∴ $f(G) = g(G) + h(G)$

$f(G) = 2 + 1 + 10 = 13$

→ $f(F) = g(F) + h(F)$

→ $f(F) = 2 + 9 + 18 = 29$

→ Node 'G' has minimum path cost. So choose node 'G' among G & F. Therefore the Successor nodes of 'G' is K.

→ $f(K) = g(K) + h(K)$

$= 2 + 1 + 3 + 8 = 14$

→ Successor node of 'K' is 'L'.

$\therefore f(L) = g(L) + h(L)$

$= 2+1+3+5+2 = \underline{13}$

→ Successor node of 'L' is 'M'

$f(M) = g(M) + h(M)$

$= 2+1+3+5+4+0.$

$f(M) = \underline{15}$

$\because$ heuristic value of 'M' is 0, we reached Goal node.

$\boxed{\text{Shortest path: } A \to C \to G \to K \to M.}$

Optimal Path Cost = $\underline{15}$

5	Explain problem-solving agents with an algorithm. Describe the five components of problem formulation using the Romania map example, where the agent starts in Arad and aims to reach Bucharest as the goal.	10	CO2	L3

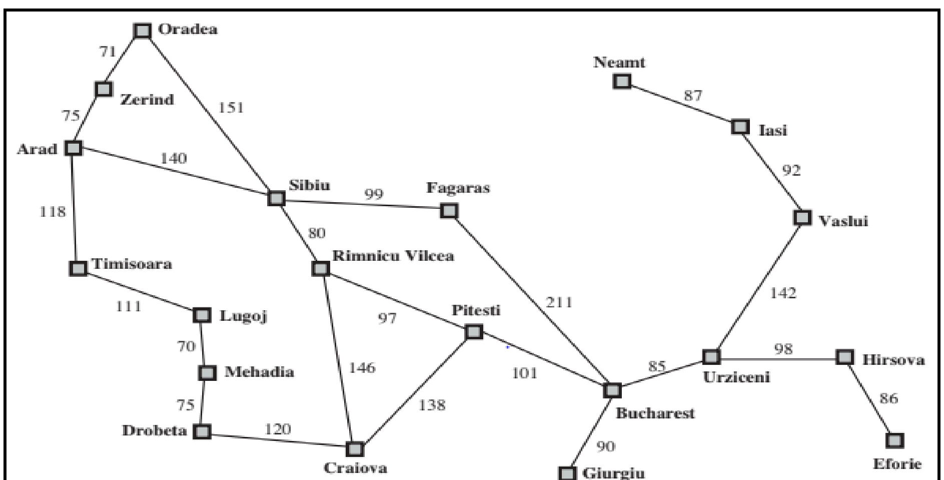


Figure 3.2 A simplified road map of part of Romania.

SOLn

Problem Solving Agent Algorithm-

function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  persistent: seq, an action sequence, initially empty
    state, some description of the current world state
    goal, a goal, initially null

```
        problem, a problem formulation
    state ← UPDATE-STATE(state, percept)
    if seq is empty then
        goal ← FORMULATE-GOAL(state)
        problem ← FORMULATE-PROBLEM(state, goal)
        seq ← SEARCH(problem)
        if seq = failure then return a null action
action ← FIRST(seq)
seq ← REST(seq)
return action
```

Problem Formulation

A problem can be defined formally by five components:

• 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 state s,
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(Timisoara), Go(Zerind)}.

• 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 a in state s. We also use the term successor to refer to any state reachable from a given state by a single action.2 For example, we have
RESULT(In(Arad),Go(Zerind)) = In(Zerind) .

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. (The map of Romania shown in Figure 3.2 can be interpreted as a state-space graph if we view each road as standing
for two driving actions, one in each direction.) A path in the state space is a sequence of states connected by a sequence of actions.

• 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)}.
Sometimes the goal is specified by an abstract property rather than an explicitly enumer-
ated set of states. For example, in chess, the goal is to reach a state called "checkmate,"
where the opponent's king is under attack and can't escape.

• A **path cost** function that assigns a numeric cost to each path. The problem-solving
agent chooses a cost function that reflects its own performance measure. For the agent
trying to get to Bucharest, time is of the essence, so the cost of a path might be its length
in kilometers. In this chapter, we assume that the cost of a path can be described as the
sum of the costs of the individual actions along the path.The step cost of taking action a in state s to reach state s is denoted by c(s, a, s). The step costs for

	Romania are shown in Figure 3.2 as route distances.			
6	Compare the following with atleast 4 differences: i)Discrete VS Continuous environments ii)static VS dynamic environments iii)Episodic VS Sequential environments iv)Deterministic vs. Stochastic environments v)Single agent VS Multi agents	10	CO2	L2
SOLn	i)Discrete VS Continuous environments- A **discrete** environment has a finite set of states, actions, and time steps, while a **continuous** environment involves smoothly changing states and time progression. The state of the environment, the way time is handled, and agents percepts & actions can be discrete or continuous ● Ex: Crossword puzzles: discrete state, time, percepts & actions ● Ex: Taxi driving: continuous state, time, percepts & actions Note: ● The simplest environment is fully observable, single-agent, deterministic, episodic, tatic and discrete. Ex: simple vacuum cleaner ii)static VS dynamic environments A **static** environment does not change while the agent is deciding, while a **dynamic** environment evolves continuously, requiring real-time decision-making. The task environment is dynamic if it can change while the agent is choosing an action, static otherwise ⇒ agent needs keep looking at the world while deciding an action ● Ex: crossword puzzles are static, taxi driving is dynamic The task environment is semidynamic if the environment itself does not change with time, but the agent's performance score does ● Ex: chess with a clock Static environments are easier to deal wrt. [semi]dynamic ones. iii)Episodic VS Sequential environments In an **episodic** environment, decisions are independent of past actions, whereas in a **sequential** environment, current actions influence future states and decisions. In an episodic task environment ● the agent's experience is divided into atomic episodes ● in each episode the agent receives a percept and then performs a single action In episodes do not depend on the actions taken in previous episodes, and they do not influence future episodes ● Ex: an agent that has to spot defective parts on an assembly line, In sequential environments the current decision could affect future decisions ⇒ actions can have long-term consequences ● Ex: chess, taxi driving, ... Episodic environments are much simpler than sequential ones ● No need to think ahead! iv)Deterministic vs. Stochastic environments A **discrete** environment has a finite set of states, actions, and time steps, while a **continuous** environment involves smoothly changing states and time			

	progression. ● A task environment is deterministic if its next state is completely determined by its current state and by the action of the agent. (Ex: a crossword puzzle). ● If not so: o A task environment is stochastic if uncertainty about outcomes is quantified in terms of probabilities (Ex: dice, poker game, component failure,...) o A task environment is nondeterministic iff actions are characterized by their possible outcomes, but no probabilities are attached to them. In a multi-agent environment we ignore uncertainty that arises from the actions of other agents (Ex: chess is deterministic even though each agent is unable to predict the actions of the others). A partially observable environment could appear to be stochastic. =⇒ for practical purposes, when it is impossible to keep track of all the unobserved aspects, they must be treated as stochastic. (Ex: Taxi driving). v)Single agent VS Multi agents A **single-agent** environment involves one decision-making entity, while a **multiagent** environment includes multiple interacting agents that can be **competitive** (e.g., chess) or **cooperative** (e.g., traffic coordination).			

Faculty Signature CCI Signature HOD Signature