ARTIFICIAL INTELLIGENCE - MAY 2014

(SEMESTER 7)

TOTAL MARKS: 80
TOTAL TIME: 3 HOURS

(1) Question 1 is compulsory.
(2) Attempt any ***three*** *from the remaining questions.*
(3) Assume data if required.
(4) Figures to the right indicate full marks.

---

1 (a) Describe robot workspace. (5 marks)

1 (b) Explain homogeneous transformation matrix. (5 marks)

1 (c) Discuss the heuristic function for 8-puzzle problem. (5 marks)

1 (d) Discuss structure of learning agent. (5 marks)

---

2 (a) Explain A* algorithm with example. (10 marks)

2 (b) Explain breadth first algorithm. (10 marks)

---

3 (a) Explain resolution refutation using suitable example. (10 marks)

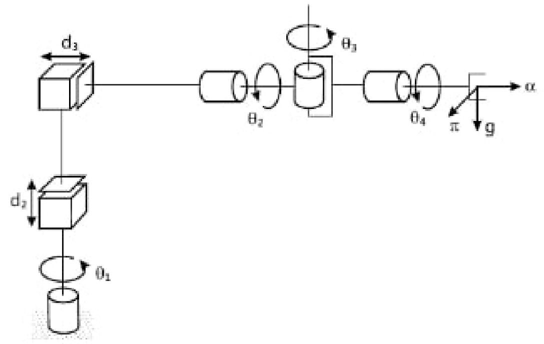
3 (b) Explain backward chaining giving suitable example. (10 marks)

---

4 (a) Discuss the application of decision tree for restaurant example. (10 marks)

4 (b) Why uncertainty occurs in AI system? How probability theory can be applied for toothache problem? (10 marks)

5 (a) Using DH algorithm, derive homogeneous transformation matrix for following robot. (15 marks)



5 (b) Explain different types of robots. (5 marks)

---

6 (a) Discuss various position sensor used in robots. (10 marks)

6 (b) Discuss partial order planning giving suitable example. (10 marks)

Write short note on

7 (a) Limitation of Hill-Climbing algorithm. (5 marks)

7 (b) Predicate Logic (5 marks)

7 (c) Properties of environment (5 marks)

7 (d) Admissibility of A* (5 marks)

1 (c) Describe robot workspace. ***--- 5 Marks***

ADD NOTE BOOKMARK

This question was repeated in Dec 14.

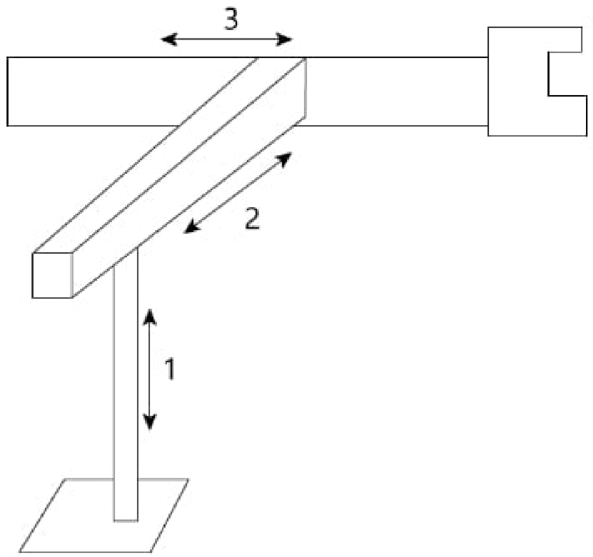
Answer:

Robot Workspace:

The workspace or configuration of a robot is defined as the locus of points in three dimensional space that can be reached by the wrist. Accordingly we get five configurations of the robot based on work envelope geometries or co-ordinate geometries which are as given as follows:

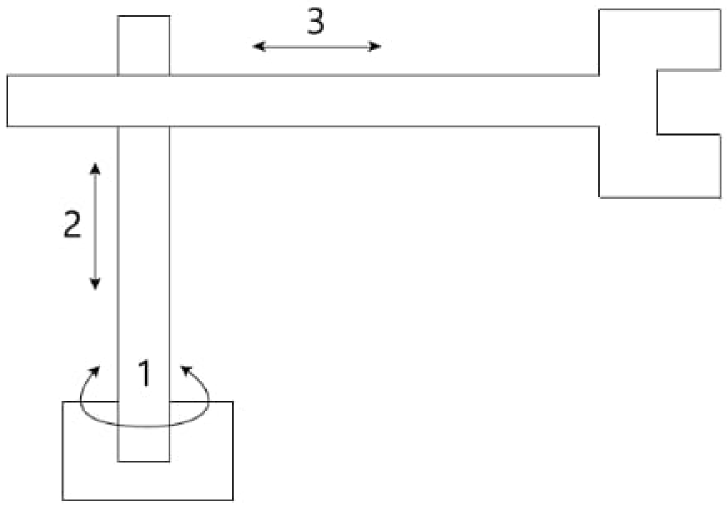
1. Cartesian robot
2. Cylindrical robot
3. Spherical robot
4. SCARA
5. Articulated robot

1. Cartesian robot:



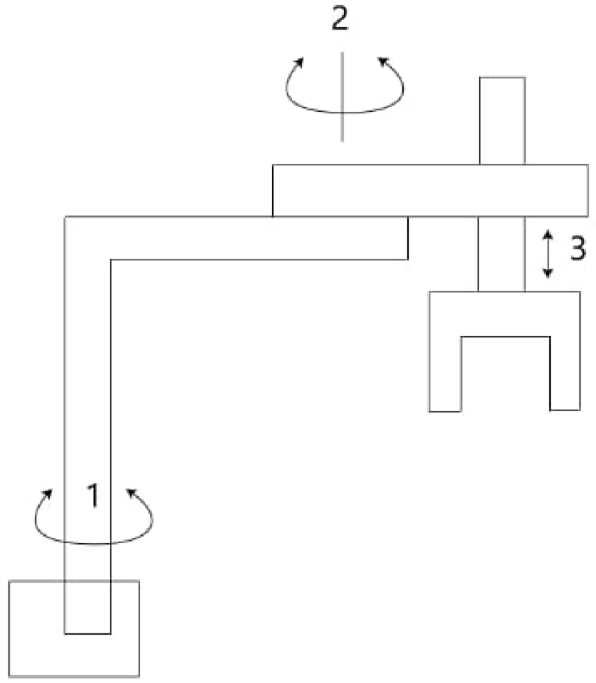
- In Cartesian robot, all three joints are prismatic joints.
- The workspace of Cartesian robot is rectangular box.

2. Cylindrical robot:



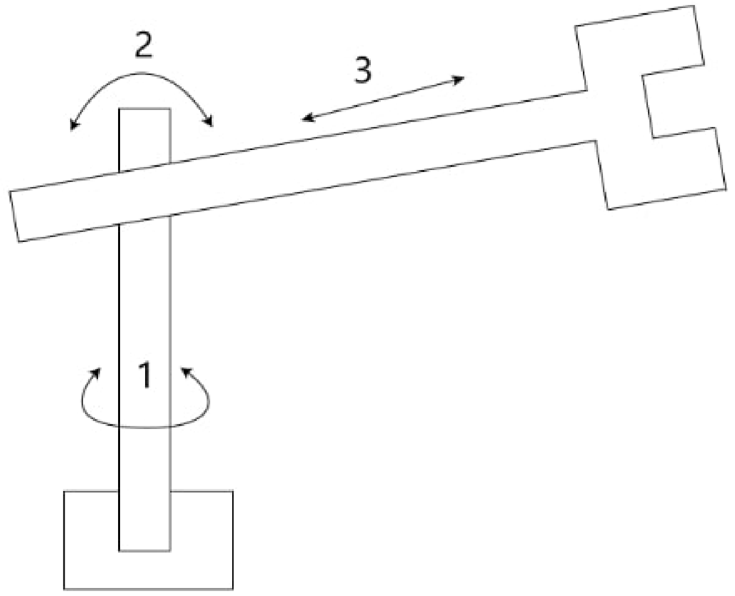
- In cylindrical robot, there is one rotary joint and two prismatic joint.
- Workspace of cylindrical robot is volume between 2 concentric cylinders.

3. Spherical robot:



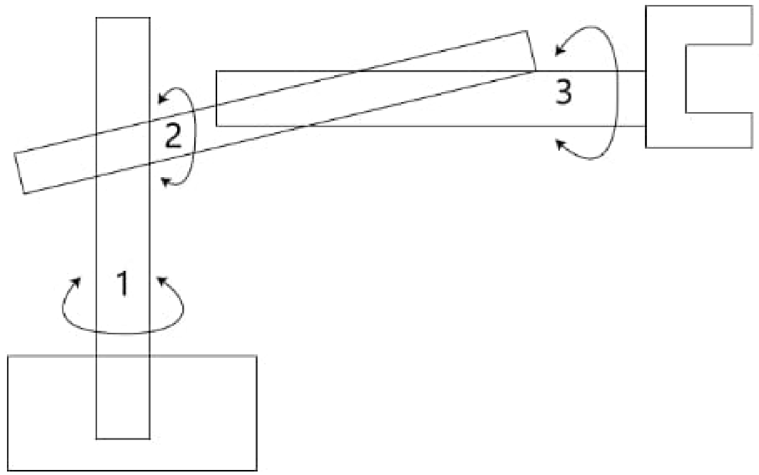
- In spherical robot, there are two rotary joints and one prismatic joint.
- Workspace of spherical robot is volume between 2 concentric cylinders.

4. SCARA robot:



- In SCARA robot, all axis are parallel, there are two rotary joints and one prismatic joint.
- Workspace of SCARA robot is complex.

5. Articulated robot:



- In articulated robot, all joints are rotary joints.
- Workspace of articulated robot is complex.

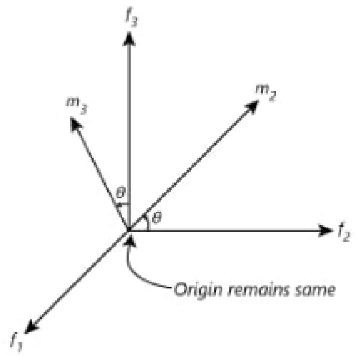
1 (b) Explain homogeneous transformation matrix. ***--- 5 Marks***

ADD NOTE BOOKMARK

Answer:

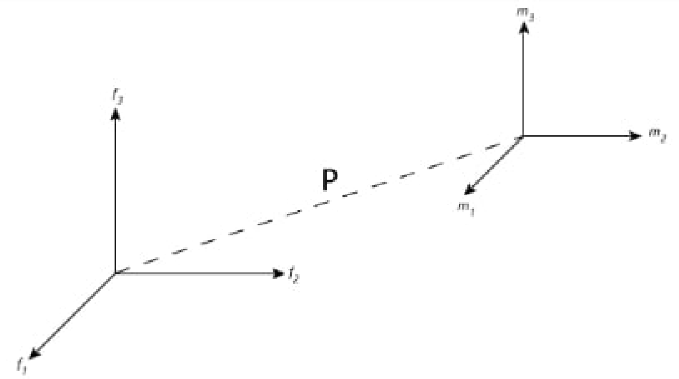
- It is homogeneous transform
- It is used to represent rotation and translation
- It is a 4x4 transformation matrix
- Formulation of homogenous transformation matrix is explained as follows:

Only rotation:



$$[q]^F = R[q]^m$$

Rotation & translation:



$$[q]^F = R[q]^m + P$$

$$\underset{\textbf{Unknown}}{\begin{bmatrix} [q]^F \\ 1 \end{bmatrix}} = \underset{\textbf{Transform}}{\begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}} \underset{\textbf{Known}}{\begin{bmatrix} [q]^m \\ 1 \end{bmatrix}}$$

$$T = \begin{bmatrix} R & P \\ 0 & 1 \end{bmatrix}$$

Rotation

Translation

$$T = \left[\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & P_1 \\ R_{21} & R_{22} & R_{23} & P_2 \\ R_{31} & R_{32} & R_{33} & P_3 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

Perspective

Scaling

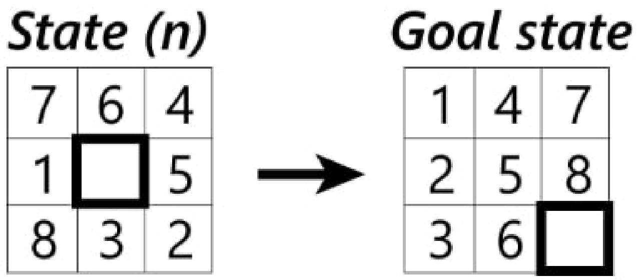
- Homogeneous co-ordinate transform matrix comprises of four parts and they are as follows:
    1. Rotation
    2. Translation
    3. Perspective
    4. Scaling
- In case of pure rotation, translation part will be zero
- In case of pure translation, rotation part will be an identity matrix

1 (c) Discuss the heuristic function for 8-puzzle problem. ***--- 5 Marks***

ADD NOTE BOOKMARK

Answer:

For an 8-puzzle problem, admissible heuristic is the number of misplaced tiles.



h(n)=8 ⇒ all 8-tiles are misplaced from goal

The 8-puzzle was one of the earliest heuristic search problems. The objective of the puzzle us to slide the tiles horizontally or vertically into empty space until configuration matched the goal configuration.

We can define two heuristic functions for 8 puzzle problem.

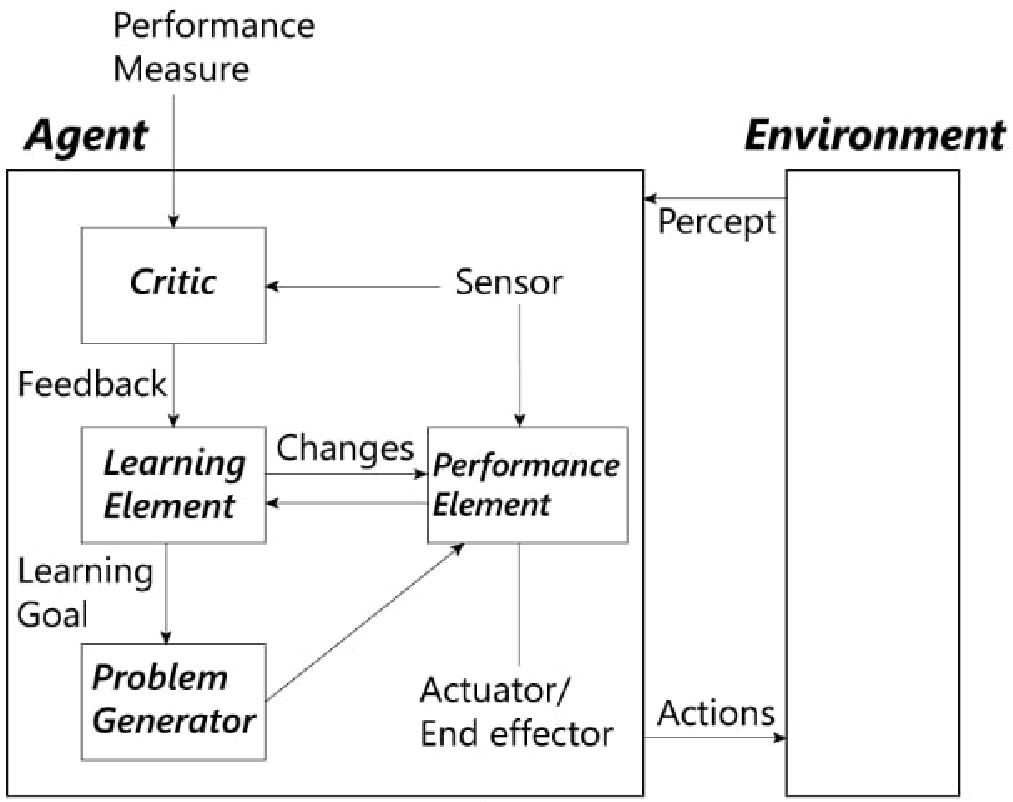
1. h_1 = the number of misplaced tiles, all of the eight tiles are out of position in the diagram above, so that the start state would have h_1=8. h_1 is admissible heuristic, because it is clear that any tile that is out of place must be moved at least once.
2. h_2 = the sum of distances of the tiles from their goal positions. Because tiles cannot move along diagonals, the distance we will count is the sum of horizontal and vertical distances. This is something called the city of block distance or Manhattan distance. h_2 is also admissible because all any move can do is move over tile one step closer to the goal.

1 (d) Discuss structure of learning agent. *--- 5 Marks*

ADD NOTE BOOKMARK

This question was repeated in Dec 13.

Answer:



Structure of Learning Agent

Components of learning agent are described as follows:

1. ***Performance element***

It is agent in itslef. It has percept to action mapping and condition - action rules.

These rules can be changed by learning element depending on the experience & feedback from the critic.

2. ***Critic:***

It compares performance measure and the percept from sensor about the environment and guides learning element. Critic plays a very important role in the structure of learning agent. Critic has access to performance measure and gets the percept from the sensor about the environment, on comparison between both performance measure and percept, it provides a feedback or actually guides the learning element and on receiving the feedback from the critic. Learning element can change the rules.

3. ***Learning element:***

   - It is the heart of learning agent
   - It learns and adapts to the changing needs to the environment
   - It can accordingly modify condition-action rules of the agent
   - It also guides problem generator
   - E.g.: supervised / unsupervised / reinforced learning

4. ***Problem generator:***

   - It generates problem to explore the world
   - E.g. automated taxi driver → wet road / brakes

2 (b) Explain A* algorithm with example. ***--- 10 Marks***

ADD NOTE BOOKMARK

This question was repeated in May 15.

Answer:

It is an uninformed search technique. It uses additional information beyond problem formulation or tree. Search is based on evaluation function f(n). Evaluation function is based on both heuristic function h(n) and g(n).

i.e. f(n)=g(n)+h(n)

$$\begin{tikzcd}\boxed{\text{Root}} \arrow[r, "g(n)"] & n \arrow[r, "h(n)"] & \boxed{\text{goal}}\end{tikzcd}$$

For its implementation it uses two queue:

i. OPEN

ii. CLOSE

OPEN queue is priority queue which is arranged in ascending order of f(n).

Algorithm:

Step 1: Create a single member queue comprising of root node

Step 2: If first member of queue is goal, then goto step 5

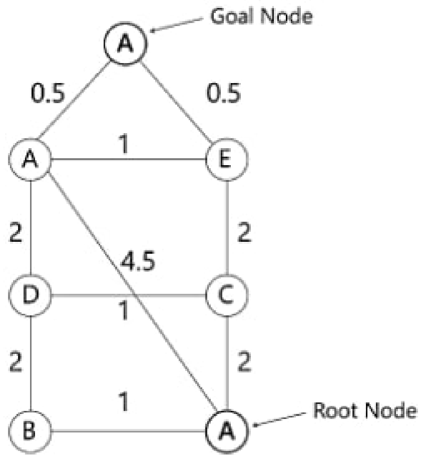
Step 3: If first member of queue is not goal, then remove t from queue and ADD to close queue. Consider it children, if any, add them into the queue, in ascending order of evaluation function f(n).

Step 4: If queue is not empty then go to step 2. If queue is empty then go to step 6

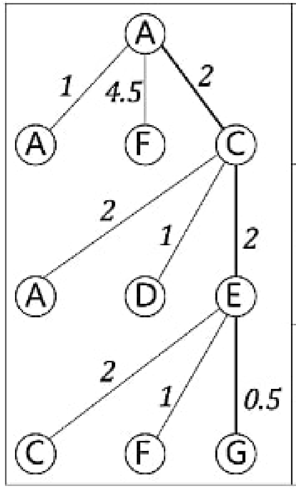
Step 5: print "SUCCESS" and stop

Step 6: print "FAILURE" and stop

Consider an example given below:



- Here we use straight line distance as heuristic h(n).
- We first convert graph into tree and then use informed search technique to solve the problem



$$\text{Pass 1 of algo}\begin{cases} f(B) = g(B) + h(B) = 1 + 4.5 = 5.5 \\ f(F) = g(F) + h(F) = 4.5 + 0.5 = 5 \\ f(C) = g(C) + h(C) = 2 + 2.5 = 4.5 \end{cases}$$

$$\text{Pass 2 of algo}\begin{cases} f(A) = g(A) + h(A) = 4 + 4.5 = 8.5 \\ f(D) = g(D) + h(D) = 3 + 2.5 = 5.5 \\ f(E) = g(E) + h(E) = 4 + 0.5 = 4.5 \end{cases}$$

$$\text{Pass 3 of algo}\begin{cases} f(C) = g(C) + h(C) = 6 + 2.5 = 5.5 \\ f(F) = g(F) + h(F) = 5 + 0.5 = 5 \\ f(G) = g(G) + h(G) = 4.5 + 0 = 4.5 \end{cases}$$

OPEN	CLOSE
$[A]$	
$4.5 \quad 5 \quad 5.5$	
$[C, \quad F, \quad B]$	$[A]$
$4.5 \quad 5 \quad 5.5 \quad 5.5 \quad 8.5$	
$[E, \quad F, \quad B, \quad D, \quad A]$	$[A, \quad C]$
$4.5 \quad 5.5 \quad 5.5 \quad 5.5 \quad 8.5 \quad 8.5$	
$[G, \quad F, \quad B, \quad D, \quad A, \quad C]$	$[A, \quad C, \quad E]$

First member of queue is goal "SUCCESS"

Advantages:

- It is a complete algorithm
- It is optimum because it considers evaluation function i.e. $f(n)=g(n)+h(n)$ where $h(n)$ is admissible heuristic

Disadvantages:

- It generates same node again and again
- Hence large memory is needed

2 (b) Explain breadth first algorithm. ***--- 10 Marks***

ADD NOTE BOOKMARK

Answer:

- BFS is uniform tree search algorithm
- It uses two queues for its implementation
    - open / fringe node queue
    - close queue
- It uses first-in-first-out [FIFO] queue
- Children are added from Back-End of queue

Algorithm:

Step-1: Create a single member queue comprising of root node

Step-2: If first member of queue is goal, then goto step-5

Step-3: If first member of queue is not goal, then remove it from queue and add to close or visited queue. Consider its children / successor, if any, add them from back / rear end.

Step-4: If queue is not empty, then go to step-2. If queue is empty, then go to step-6.

Step-5: Print "SUCCESS" and stop

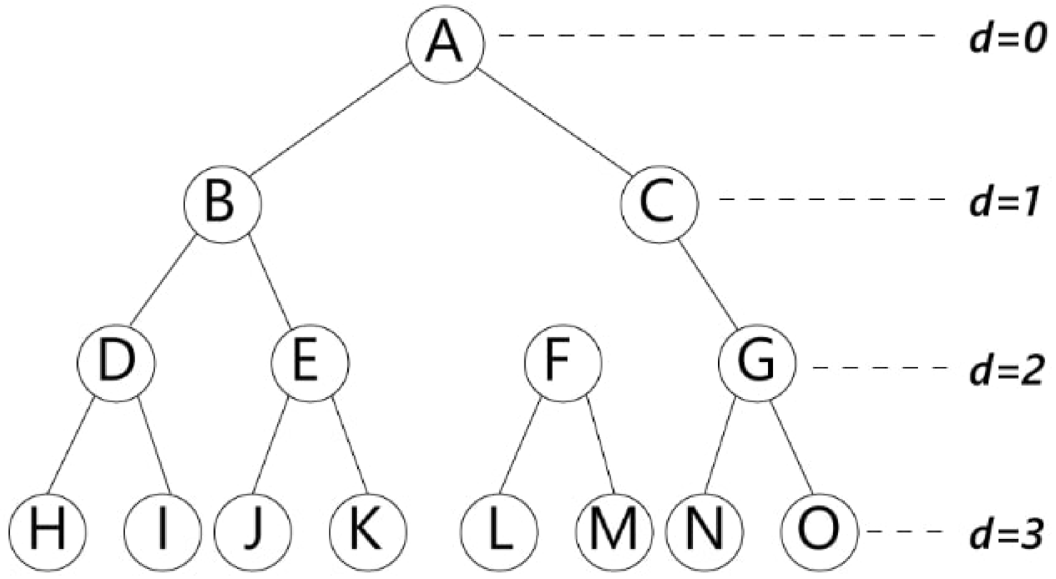
Step-6: Print "FAILURE" and stop

Example:

Consider the following example for tree search, we consider branch factor = 2.

A - root node

G - goal node



OPEN/FRINGE	CLOSE
[A]	
[B, C]	[A]
[C, D, E]	[A, B]
[D, E, F, G]	[A, B, C]
[E, F, G, H, I]	[A, B, C, D]
[F, G, H, I, J, K]	[A, B, C, D, E]
[G, H, J, K, L, M]	[A, B, C, D, E, F]

Advantages / Disadvantages:

1. ***Completeness -***
   YES.
   It gives shallowest goal.
2. ***Optimality -***
   YES.
   Provided path cost is non-decreasing
3. ***Space complexity***
   $O(b^{d+1})$
4. ***Time complexity***
   $O(b^{d+1})$

3 (a) Explain resolution refutation using suitable example. ***--- 10 Marks***

ADD NOTE BOOKMARK

Answer:

- Resolution by refutation is a powerful method of reasoning or inference.
- It tries to given fact using refutation or contradiction
- If fact 'F' is to be proved then it begins by assuming '¬F'
- It proceeds by contradicting statements in the knowledge base (KB)
- When the search ends with "NULL CLAUSE" then the proof is said to be established.
- It uses sentences in CNF form
- The procedure becomes simple if sentence is "HORN CLAUSE"

  HORN CLAUSE ⇒ atmost one literal is positive
  e.g.:
  ¬P v Q is Horn Clause
  ¬P v Q v R ***not*** is Horn Clause

- This method is based on "Modus Ponen" (latin for "mode that affirms")

  P ⇒ Q
  If P is known to be true then Q is also true
  $\left(\frac{P \Rightarrow Q,\ P}{Q}\right)$
  Q is derived inference

Rules for resolution

1.
$$\begin{tikzcd} P \arrow[dr, dash] & & \neg P \arrow[dl, dash] \\ & \boxed{\text{NULL}} & \end{tikzcd}$$

2.
$$\begin{tikzcd} P \arrow[dr, dash] & & \neg P \vee Q \arrow[dl, dash] \\ & \boxed{Q} & \end{tikzcd}$$

Example

Consider following sentences

1. It is HOT
2. If it is HOT, then it is humid
3. If it is HOT & HUMID, then it rains

A - Convert sentences in propositional logic
B - Convert in CNF form
C - prove "*it is raining*" by resolution

Solution: Convert sentences in propositional logic we first select symbiosis for given facts.

P - It is HOT
Q - It is Humid
R - It is raining

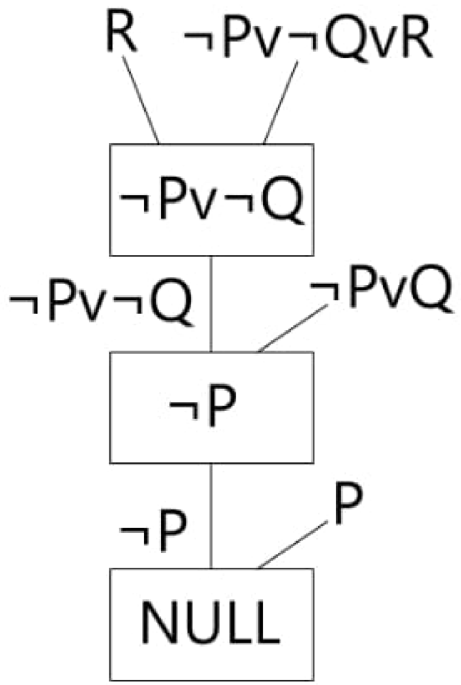
1. P
2. P ⇒ Q
3. P ∩ Q ⇒ R

Convert in CNF form:

1. P
2. $\neg P \vee Q$
3. $\neg(P \cap Q) \vee R \equiv \neg P \vee \neg Q \vee R$

Proof of "it is raining" by resolution:

We want to prove 'R' method of resolution begins with '¬R'



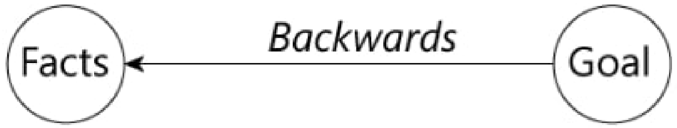
Hence 'R' is true.

3 (b) Explain backward chaining giving suitable example. ***--- 10 Marks***

ADD NOTE BOOKMARK

Answer:

- Chaining means linking or connecting.
- In backward chaining we connect known facts to arrive at some inference
- For simple problems, we make use of a powerful method like method of resolution
- In backward chaining there is no need of CNF conversion
- Following diagram represents the backward chaining



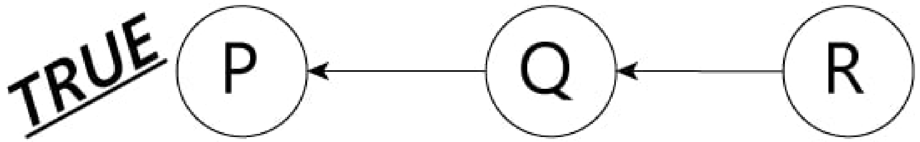
- Backward chaining is goal or query driven
- It begins from goal and moves in backward direction to establish proof

Example:

Prove "RAJA IS ANGRY" by backward chaining

Knowledge Base:

1. P
2. P ⇒ Q
3. P ⇒ R



Here in backward chaining we start from goal state i.e. R and then we will move in backward direction to establish the proof.

4 (a) Discuss the application of decision tree for restaurant example. ***--- 10 Marks***

ADD NOTE BOOKMARK

This question was repeated in Dec 13.

Answer:

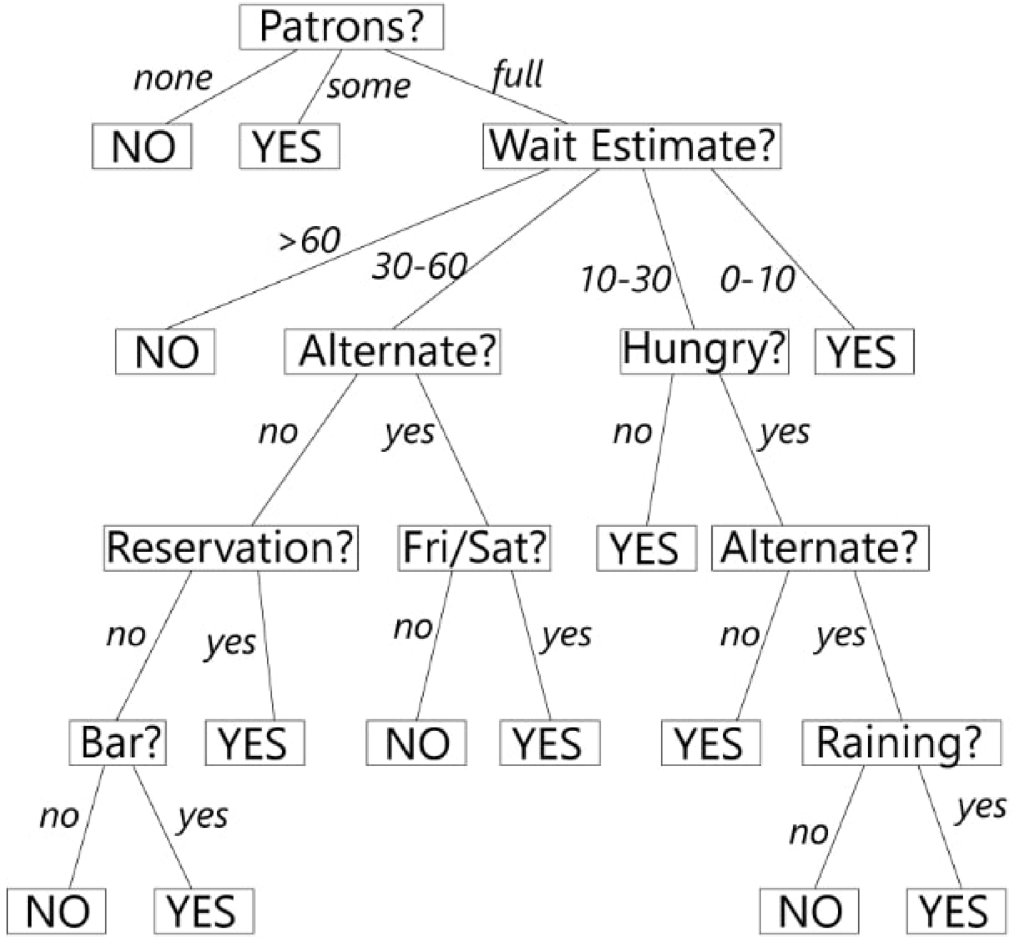
A decision tree takes as input an object or situation described by a set of properties and outputs a "YES / NO" decision. Decision tress therefore represents Boolean functions. Functions with a larger range of outputs can also be represented, but for simplicity we usually stick to the Boolean case. Each internal node in the tree corresponds to a test of the value of one of the properties and the branches from the node are labelled with the possible values of the test. Each leaf node in the tree specifies the Boolean value to be returned if that leaf is reached. As an example, consider the problem of whether to wait for a table at a restaurant. In setting this up as a learning problem, we first have to decide what properties or attributes are available to describe examples in the domain. Suppose we decide on the following list of attributes.

- *Alternate:* whether there is a suitable alternative restaurant nearby
- *Bar:* whether the restaurant has a comfortable bar area to wait in
- *Fri/Sat:* true on Friday/Saturday
- *Hungry:* whether we are hungry
- *Patrons:* How many people are in the restaurant (values - are none, some and full)
- *Price:* The restaurant's price range ($, $ $, $$$)
- *Raining:* whether it is raining outside
- *Reservation:* whether we made a reservation

- *Type:* the kind of restaurant (French, Italian, Thai or Burger)
- *Wait estimate:* the wait estimated by the host (0-10 minutes, 10-30 minutes, 30-60 minutes, >60 minutes)

The decision tree usually used for this domain is shown in figure drawn below. Logically, the tree can be expressed as a conjunction of individual implications corresponding to the paths through the tree ending in YES nodes. For example, the path for a restaurant full of patrons, with an estimated wait of 10-30 minutes when the agent is not hungry is expressed by logical sentence:

$$\forall r \ \text{Patrons}(r,\ \text{Full}) \wedge \text{wait estimate } (r, 0-10) \wedge \text{Hungry}(r,\ N) \Rightarrow \text{will wait}$$



A decision tree for deciding whether to wait for a table

4 (b) Why uncertainty occurs in AI system? How probability theory can be applied for toothache problem? ***--- 10 Marks***

ADD NOTE BOOKMARK

Answer:

Consider an example to illustrate the concepts involved in the nature of uncertain knowledge:

Diagnosis – whether for medicine or automobile repair, always involves uncertainty. Consider the below rule for dental diagnosis using first-order logic.

$$\forall p \ \mathrm{Symptom}(P, \mathrm{Toothache}) \Rightarrow \mathrm{Disease}(P, \mathrm{Cavity})$$

The above rule is incorrect since, not all patients with tooth aches, some of them have gum disease, an abscess or one of several other problems.

$$\forall p \mathrm{Symptom}(P, \mathrm{Tooth\ ache}) \Rightarrow \mathrm{Disease}(P, \mathrm{Cavity}) \lor \mathrm{Disease}(P, \mathrm{Gum\ disease}) \lor \mathrm{Disease}(P, \mathrm{Abscess}) \cdots$$

Adding almost unlimited list of possible causes do not make the rule true. However, one can try turning the rule into a causal rule.

$$\forall p \ \mathrm{Disease}(P, \mathrm{Cavity}) \Rightarrow \mathrm{Symptom}(P, \mathrm{Tooth\ ache})$$

But this rule is not right either, not all cavities cause pain. The only way to fix the rule is to make it logically exhaustive: to augment the left-hand side with all qualifications required for a cavity to cause a tooth ache. The use of first order logic to cope with a domain like medical diagnosis thus fails for three main reasons.

Reasons for uncertainty:

- *Laziness:* It is too much work to list the complete set of antecedents or consequents needed to ensure an exception less rule and too hard to use such rules
- *Theoretical Ignorance:* Medical science has no complete theory for the domain.
- *Practical Ignorance:* Even if all the rules are known, it might be uncertain about a particular patient because not all the necessary tests have been or can be run.

The connection between toothaches and cavities is just not a logical consequence in either direction. This is typical of the medical domain, as well most other judgmental domains: law, business design, automobile repair, gardening and so on. The agent's knowledge can at best provide only a degree of belief in the relevant sentences. Our main tool for dealing with degrees of belief will be probability theory, which assigns a numerical degree of belief between 0 and 1 to sentences.

Probability provides a way of summarizing the uncertainty that comes from our laziness and ignorance.

A probability of o for given sentence corresponds to an unequivocal belief that the sentence is false, while a probability of 1 corresponds to an unequivocal belief that sentence is true. Probabilities between 0 and 1 corresponds to intermediate degrees of belief in the truth of the sentence.

5 (b) Explain different types of robots. ***--- 5 Marks***

ADD NOTE BOOKMARK

Answer:

The different types of robots are mentioned as follows:

1. Cartesian [PPP]
2. Cylindrical [RPP]
3. Spherical [RRR]
4. SCARA [RRP]
5. Articulated [RRR]

Robots classification on power supply are as follows:

1. Electrical Drive Robot
2. Hydraulic Drive Robot
3. Pneumatic Drive Robot

Robots classification on motion control are as follows:

1. Point to point motion
2. Continuous path motion

6 (a) Discuss various position sensor used in robots. ***--- 10 Marks***

ADD NOTE BOOKMARK

Answer:

The various position sensors used in robots are as follows:

1. Encoders
2. Potentiometers
3. Resolvers

1. Encoders:

Encoders are used for converting the angular or linear displacement into digital signals.

Some types of encoders are:

i. *Linear encoders:* It is used to calculate the directions and positions that are in the linear form
ii. *Rotary encoders:* It helps to calculate the directions and positions that are angular form
iii. *Incremental encoders:* It is used in robots for sensing the position from its last position
iv. *Absolute encoders:* It brings a definite position that is proportional to a fixed reference position

2. Potentiometers:

Potentiometers produce an output voltage that is proportional to the position of wiper. They are the analog devices used for calculating the linear or rotary movements based on the design. It consists of a rotating wiper, which is in contact with a resistive element. This wiper is connected with an object that is motion. An AC or DC voltage is supplied to the resistive element. The wiper and ground voltage is proportional to the ratio of the wiper's one side resistance to the resistive element's total resistance.

3. Resolvers:

A resolver is also an analog device as like potentiometers and it is a rotary electrical transformer basically implemented for calculating the degress of rotation. It requires only AC signal for excitation because the use of DC current will not produce any output signal. The output signal of a resolver is proportional to the angle of rotating element with respect to the fixed element.

6 (b) Discuss partial order planning giving suitable example. --- 10 Marks

ADD NOTE BOOKMARK

Answer:

Planning is "the task of coming up with a sequence of actions that will achieve a goal".

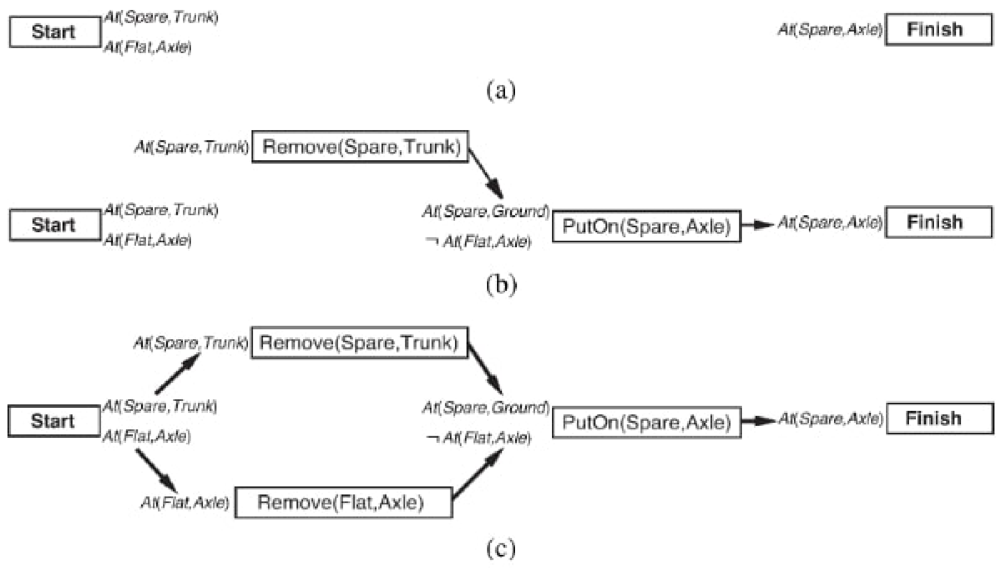
There are several types of algorithms that allow us to construct a plan such as progression planning, regression planning, and partial-order planning.

Progression planning is done with a forward state-space search, which is to say that, "we start in the problem's initial state [and consider] sequences of actions until we find a sequence that reaches a goal state". This can pose major performance problems because it considers even completely irrelevant actions.

Regression planning is the opposite - it works backwards from the goal state. This removes the problems associated with examining irrelevant actions, but not without its problems: often it is not "obvious how to generate a description of the possible predecessors of the set of goal states"

Partial-order planning is not totally-ordered as we see in progression and regression planning. Instead, partial-order planning enables us to take advantage of problem decomposition. The algorithm works on several subgoals independently, solves them with several subplans, then combines the subplans. In addition, such an approach also has the advantage of flexibility in the order in which it *constructs* the plan. That is, the planner can work on 'obvious' or 'important' decisions first, rather than being forced to work on steps in chronological order.

Partially ordered plans are created by a search through the space of plans rather than through the state space. We start with the empty plan consisting of just the initial state and
the goal, with no actions in between, as in Figure (a). The search procedure then looks for a flaw in the plan, and makes an addition to the plan to correct the flaw (or if no correction can be made, the search backtracks and tries something else). A flaw is anything that keeps the partial plan from being a solution. For example, one flaw in the empty plan is that no action achieves At(Spare,Axle). One way to correct the flaw is to insert into the plan the action PutOn(Spare,Axle). Of course that introduces some new flaws: the preconditions of the new action are not achieved. The search keeps adding to the plan (backtracking if necessary) until all flaws are resolved, as in Figure (c).



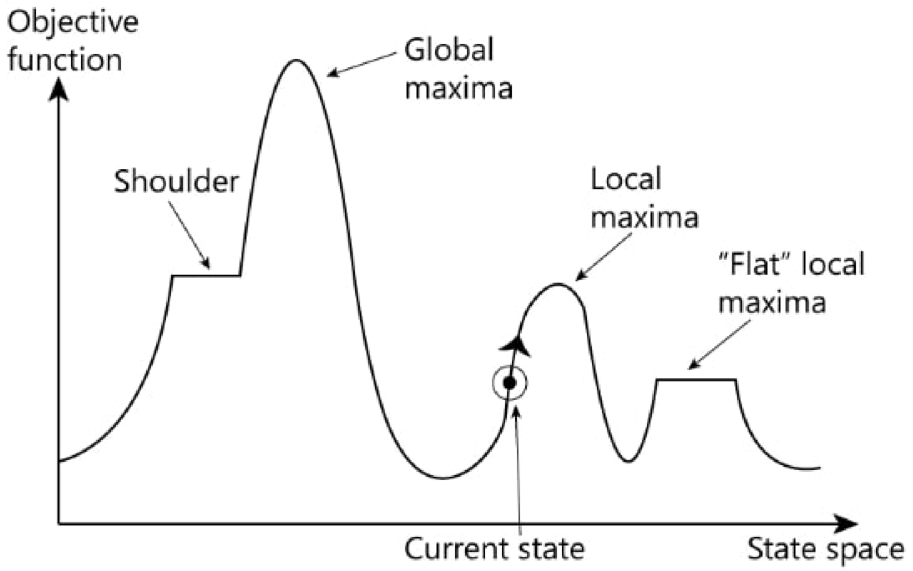
(a) the tire problem expressed as an empty plan.
(b) an incomplete partially ordered plan for the tire problem.
(c) a complete partially-ordered solution.

7 (a) Limitation of Hill-Climbing algorithm. ***--- 5 Marks***

ADD NOTE BOOKMARK

Answer:

Consider following state-space landscape



1. ***Local Maxima***

   A problem with hill climbing is that it will find only local maxima. Unless the heuristic is convex, it may not reach a global maximum. Other local search algorithms try to overcome this problem such as stochastic hill climbing, random walks and simulated appealing. This problem of hill climbing can be solved by using random hill climbing search technique.

2. ***Ridges:***

   A ridge is a curve in the search space that leads to a maximum but the orientation of the ridge compared to the available moves that are used to climb is such that each move will lead to a smaller point. In other words, each point on a ridge looks to the algorithm like a local maximum, even though the point is part of a curve leading to a better optimum

3. ***Plateau:***

   Another problem with hill climbing is that of a plateau, which occurs when we get to a "flat" part of the search space i.e. we have a path where the heurisitcs are all very close together. This kind of flatness can cause the algorithm to cease progress wander aimlessly.

7 (b) Predicate Logic *--- 5 Marks*

ADD NOTE BOOKMARK

Answer:

Predicate logic involves using standard forms of logical symbolism which have been familiar to philosophers and mathematicians for many decades. Most simple sentences, for example, "Peter is generous" or "Jane gives a painting to Sam", can be represented in terms of logical **formulae** in which a **predicate** is applied to one or more **arguments**.

PREDICATE	ARGUMENTS
generous	(peter)
gives	(jane, painting, sam)

Propositional logic combines atoms that contains no propositional connectives and have no structure (today_is_wet, john_likes_apples).

Propositional logic is fairly powerful but we must add variables and quantification to be able to reason about objects in atoms and express properties of a set of objects without listing the atom corresponding to each object.

Predicates allow us to talk about objects

- Properties: is_wet(today)
- Relations: likes(john, apples)
- Constants are objects: john, apples
- Functions transform objects: likes(john, fruit_of(apple_tree))
- Variables represent any object: likes(X, apples)
- Quantifiers qualify values of variables
    1. True for all objects (Universal): ∀X. likes(X, apples)
    2. Exists at least one object (Existential): ∃X. likes(X, apples)

7 (c) Properties of environment ***--- 5 Marks***

ADD NOTE BOOKMARK

Answer:

Properties of the task environment:

The range of task environments that might arise in AI is very vast. However, task environments can be categorized along a fairly small number of dimensions.

Fully observable vs Partially observable: A task environment is effectively fully observable if the sensors detect all aspects that are relevant to the choice of action, otherwise it is partially observable.

Single agent vs Multi agent: The environment may contain other agents which may be of the same kind as the agent, or of different kinds.

Deterministic vs Stochastic: If the next state of the environment is completely determined by the current state and the action executed by the agent, then we say the environment is deterministic; otherwise, it is stochastic.

Episodic vs Sequential: 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. When the next episode does not depend on the actions taken in previous episodes, the environment is episodic. In sequential environments, on the other hand, the current decision could affect all future decisions.

Static vs Dynamic: If the environment can change while an agent is deliberating, then we say the environment is dynamic for that agent; otherwise, it is static.

Discrete vs Continuous: The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent.

*7 (d) Admissibility of A** --- *5 Marks*

ADD NOTE BOOKMARK

Answer:

A* is admissible if it uses an optimistic heuristic estimate. There are few items which are needed to be satisfied for A* to be admissible. If a tree-search is in use, the optimality of A* is straightforward to be analyzed. In this case, A* is optimal, if h(n) is an admissible heuristic.

h*(n) = the true minimal cost to goal from n. A heuristic h is admissible if h(n) <= h*(n) for all states n.

Thus, A* is admissible if h(n) <= h*(n) for all nodes n in state space. h*(n) is the actual cost of min. cost path from n to a goal node.

Admissible heuristic: An admissible heuristic h(n) is one that never overestimates the cost to reach the goal.

Let's characterize a class of admissible heuristic search strategies, using the evaluation function:

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

g(n) represents the actual distance at which the state n has been found in the graph, and h(n) is the heuristic estimate of the distance from n to a goal state. So f(n) represents an estimate of the total cost of the path from the start, through n to the goal.

Let's define the evaluation function f^*:

$f^*(n) = g^*(n) + h^*(n)$

where $g^*(n)$ is the cost of the shortest path from the start to n and $h^*(n)$ returns the actual cost of the shortest path from n to the goal. So f^* is the actual cost of the optimal path from the start to the goal through n.

Algorithm A: In algorithm A, $g(n)$, the cost of the current path from start to n, is a reasonable estimate of g^*:

$g(n) >= g^*(n)$

$g(n)$ and $g^*(n)$ will only be equal if the search has found the optimal path to n. Similarly, in algorithm A, we replace $h^*(n)$ with $h(n)$. Although we can't actually compute $h^*(n)$, we can often determine whether $h(n)$ is bounded by $h^*(n)$ -- that is, we can often determine that $h(n)$ is less than or equal to the actual cost of a minimal path ($h^*(n)$).

Algorithm A*: is algorithm A in which

$h(n) <= h^*(n)$

for all states *n*. In other words, if an algorithm uses an evaluation function that underestimates the cost to the goal it is an A* algorithm.

All A* algorithms are admissible.