USN

Internal Assessment Test I – NOVEMBER 2024

Sub:	Artificial Intelligence			Sub Code:	BCS515B	Branch:	ISE
Date:	08-11-2024	Duration:	90 min's	Max Marks:	50	Sem/Sec:	V- A,B,C

		MARKS	OBE CO	OBE RBT
	Answer any FIVE FULL Questions			
1	a. Provide a step-by-step illustration of A* Search algorithm with an example and pseudocode.	6+4	CO3	L3

Solutions:

The most widely known form of best-first search is called **A* search** (pronounced "A search"). It evaluates nodes by combining $g(n)$, the cost to reach the node, and $h(n)$, the 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 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 i node with the lowest value of $g(n) + h(n)$. It turns out that this strategy is more than reasonable: provided that the heuristic function $h(n)$ satisfies certain conditions, A* sear both complete and optimal. The algorithm is identical to UNIFORM-COST-SEARCH ex that A* uses $g + h$ instead of g.

b.

c.

d.

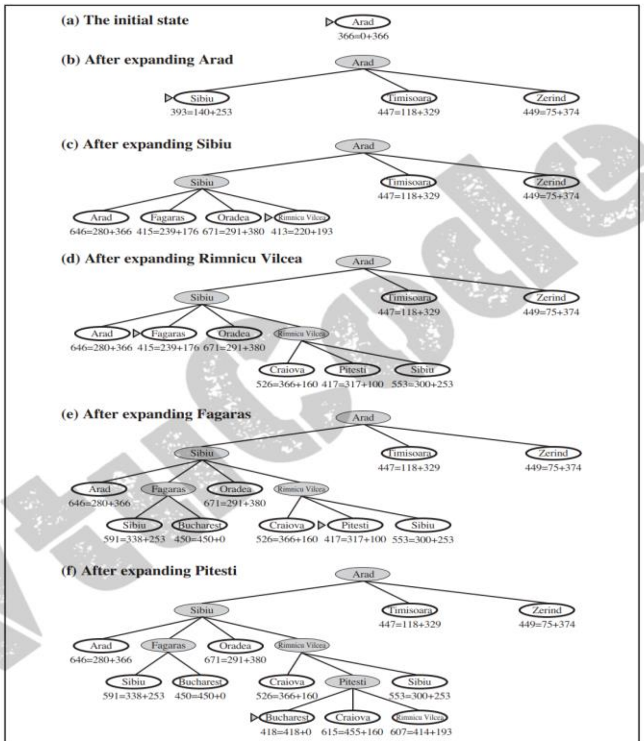

Figure 3.24 Stages in an A* search for Bucharest. Nodes are labeled with $f = g + h$. The h values are the straight-line distances to Bucharest taken from Figure 3.22.

b.

Explain about Wumpus world

computer game standards, it illustrates some important points about intelligence.

A sample wumpus world is shown in Figure 7.2. The precise definition of the task environment is given, as suggested in Section 2.3, by the PEAS description:

- **Performance measure**: +1000 for climbing out of the cave with the gold, –1000 for falling into a pit or being eaten by the wumpus, –1 for each action taken and –10 for using up the arrow. The game ends either when the agent dies or when the agent climbs out of the cave.
- **Environment**: A 4×4 grid of rooms. The agent always starts in the square labeled [1,1], facing to the right. The locations of the gold and the wumpus are chosen randomly, with a uniform distribution, from the squares other than the start square. In addition, each square other than the start can be a pit, with probability 0.2.
- **Actuators**: The agent can move *Forward*, *TurnLeft* by 90°, or *TurnRight* by 90°. The agent dies a miserable death if it enters a square containing a pit or a live wumpus. (It is safe, albeit smelly, to enter a square with a dead wumpus.) If an agent tries to move forward and bumps into a wall, then the agent does not move. The action *Grab* can be used to pick up the gold if it is in the same square as the agent. The action *Shoot* can be used to fire an arrow in a straight line in the direction the agent is facing. The arrow continues until it either hits (and hence kills) the wumpus or hits a wall. The agent has only one arrow, so only the first *Shoot* action has any effect. Finally, the action *Climb* can be used to climb out of the cave, but only from square [1,1].
- **Sensors**: The agent has five sensors, each of which gives a single bit of information:
  - In the square containing the wumpus and in the directly (not diagonally) adjacent squares, the agent will perceive a *Stench*.
  - In the squares directly adjacent to a pit, the agent will perceive a *Breeze*.
  - In the square where the gold is, the agent will perceive a *Glitter*.
  - When an agent walks into a wall, it will perceive a *Bump*.
  - When the wumpus is killed, it emits a woeful *Scream* that can be perceived anywhere in the cave.

  The percepts will be given to the agent program in the form of a list of five symbols; for example, if there is a stench and a breeze, but no glitter, bump, or scream, the agent program will get $[Stench, Breeze, None, None, None]$.

We can characterize the wumpus environment along the various dimensions given in Chapter 2. Clearly, it is discrete, static, and single-agent. (The wumpus doesn't move, fortunately.) It is sequential, because rewards may come only after many actions are taken. It is partially

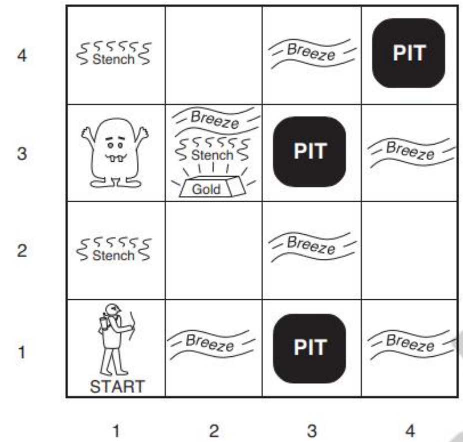

Figure 7.2 A typical wumpus world. The agent is in the bottom left corner, facing

2 a. Explain the Knowledge-Based agent with pseudocode. | 5+5 | CO3 | L2

```
function KB-AGENT(percept) returns an action
  persistent: KB, a knowledge base
              t, a counter, initially 0, indicating time

  TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
  action ← ASK(KB, MAKE-ACTION-QUERY(t))
  TELL(KB, MAKE-ACTION-SENTENCE(action, t))
  t ← t + 1
  return action
```

Figure 7.1 A generic knowledge-based agent. Given a percept, the agent adds the percept to its knowledge base, asks the knowledge base for the best action, and tells the knowledge base that it has in fact taken that action.

The central component of a knowledge-based agent is its **knowledge base**, or KB. A k edge base is a set of **sentences**. (Here "sentence" is used as a technical term. It is r but not identical to the sentences of English and other natural languages.) Each sente expressed in a language called a **knowledge representation language** and represents assertion about the world. Sometimes we dignify a sentence with the name **axiom**, wh sentence is taken as given without being derived from other sentences.

There must be a way to add new sentences to the knowledge base and a way to what is known. The standard names for these operations are TELL and ASK, respec Both operations may involve **inference**—that is, deriving new sentences from old. Inf must obey the requirement that when one ASKs a question of the knowledge base, the a should follow from what has been told (or TELLed) to the knowledge base previously. in this chapter, we will be more precise about the crucial word "follow." For now, tak mean that the inference process should not make things up as it goes along.

Figure 7.1 shows the outline of a knowledge-based agent program. Like all our a it takes a percept as input and returns an action. The agent maintains a knowledge base which may initially contain some **background knowledge**.

b. List different inference rules of Propositional logic with an example

Solutions:Inference and proofs :

Inference rules that can be applied to derive a proof—a chain of conclusi
goal.

- The best-known rule is called Modus Ponens (Latin *mode that affirms*) and is written

$$\frac{\alpha \Rightarrow \beta, \quad \alpha}{\beta}.$$

The notation means that, whenever any sentences of the form $\alpha \Rightarrow \beta$ and α are given, the sentence β can be inferred.

For example,

if $(WumpusAhead \wedge WumpusAlive) \Rightarrow Shoot$ and $(WumpusAhead \wedge WumpusAlive)$ are given, $Shoot$ can be inferred.

- Another useful inference rule is **And-Elimination**, which says that, from a conjunction of the conjuncts can be inferred:

$$\frac{\alpha \wedge \beta}{\alpha}.$$

For example, from $(WumpusAhead \wedge WumpusAlive)$, $WumpusAlive$ can be inferred.

- All of the logical equivalences in Figure 7.11 can be used as inference rules.

For example, the equivalence for biconditional elimination yields the two inference rule

$$\frac{\alpha \Leftrightarrow \beta}{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)} \quad \text{and} \quad \frac{(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)}{\alpha \Leftrightarrow \beta}$$

Let us see how these inference rules ad equivalences can be used in t
with the knowledge base containing R_1 through R_5 and show how to pro
pit in [1,2].

- First, apply biconditional elimination to R_2 to obtain

  R_6: $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.

- Then apply And-Elimination to R_6 to obtain

  R_7: $((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$.

- Logical equivalence for contrapositives gives

  R_8: $(\neg B_{1,1} \Rightarrow \neg(P_{1,2} \vee P_{2,1}))$.

- Now apply Modus Ponens with R_8 and the percept R_4 (i.e., $\neg B_{1,1}$), to obtain

  R_9: $\neg(P_{1,2} \vee P_{2,1})$.

- Finally, apply De Morgan's rule, giving the conclusion

  $R10$: $\neg P1,2 \wedge \neg P2,1$.

That is, neither [1,2] nor [2,1] contains a pit.

To apply any of the search algorithms to find a sequence of steps that constitutes a proof. Need to define a proof problem as follows:

- INITIAL STATE: the initial knowledge base.
- ACTIONS: the set of actions consists of all the inference rules applied to all the sentences that match the top half of the inference rule

Sound rules of inference

- Here are some examples of sound rules of inference
  - *A rule is sound if its conclusion is true whenever the premise is true*
- Each can be shown to be sound using a truth table

RULE	PREMISE	CONCLUSI
Modus Ponens	$A, A \rightarrow B$	B
And Introduction	A, B	$A \wedge B$
And Elimination	$A \wedge B$	A
Double Negation	$\neg\neg A$	A
Unit Resolution	$A \vee B, \neg B$	A
Resolution	$A \vee B, \neg B \vee C$	$A \vee C$

3	List elements of first-order logic and explain syntax and semantics Solution:	10	**CO4**	**L2**

First-Order logic:

- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as **Predicate logic or First-order predicate logic**. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.

- As a natural language, first-order logic also has two main parts:
  - **Syntax**
  - **Semantics**

Basic Elements of First-order logic:

Following are the basic elements of FOL syntax:

Constant	1, 2, A, John, Mumbai, cat,....
Variables	x, y, z, a, b,....
Predicates	Brother, Father, >,....
Function	sqrt, LeftLegOf, ...
Connectives	∧, ∨, ¬, ⇒, ⇔
Equality	==
Quantifier	∀, ∃

4	Explain the following concerning first-order logic with an example. a. Atomic sentences and complex sentences	10	CO4	L3

Atomic sentences:

- Atomic sentences are the most basic sentences of first-order logic. The sentences are formed from a predicate symbol followed by a parenthesis wit sequence of terms.
- We can represent atomic sentences as **Predicate (term1, term2, ..., term n)**.

Example: Ravi and Ajay are brothers: => Brothers(Ravi, Ajay).
Chinky is a cat: => cat (Chinky).

Complex Sentences:

- Complex sentences are made by combining atomic sentences using connectives.

First-order logic statements can be divided into two parts:

- **Subject:** Subject is the main part of the statement.
- **Predicate:** A predicate can be defined as a relation, which binds two atoms together in a statement.

Consider the statement: "x is an integer.", it consists of two parts, the first part x is the

b. Quantifiers

Quantifiers in First-order logic:

- A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse.
- These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier:
  - **Universal Quantifier, (for all, everyone, everything)**
  - **Existential quantifier, (for some, at least one).**

Universal Quantifier:

Universal quantifier is a symbol of logical representation, which specifies that statement within its range is true for everything or every instance of a particular thing.

The Universal quantifier is represented by a symbol ∀, which resembles an inverted A.

Note: In universal quantifier we use implication "→".

If x is a variable, then ∀x is read as:

5	Explain about backward chaining with pseudocode?	10	CO5	L2

9.4.1 A backward-chaining algorithm

Figure 9.6 shows a backward-chaining algorithm for definite clauses. FOL-BC-ASK(KB, $goal$) will be proved if the knowledge base contains a clause of the form $lhs \Rightarrow goal$, where lhs (left-hand side) is a list of conjuncts. An atomic fact like $American(West)$ is considered as a clause whose lhs is the empty list. Now a query that contains variables might be proved in multiple ways. For example, the query $Person(x)$ could be proved with the substitution $\{x/John\}$ as well as with $\{x/Richard\}$. So we implement FOL-BC-ASK as a **generator**—a function that returns multiple times, each time giving one possible result.

```
function FOL-BC-ASK(KB, query) returns a generator of substitutions
  return FOL-BC-OR(KB, query, { })

generator FOL-BC-OR(KB, goal, θ) yields a substitution
  for each rule (lhs ⇒ rhs) in FETCH-RULES-FOR-GOAL(KB, goal) do
    (lhs, rhs) ← STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ′ in FOL-BC-AND(KB, lhs, UNIFY(rhs, goal, θ)) do
      yield θ′

generator FOL-BC-AND(KB, goals, θ) yields a substitution
  if θ = failure then return
  else if LENGTH(goals) = 0 then yield θ
  else do
    first,rest ← FIRST(goals), REST(goals)
    for each θ′ in FOL-BC-OR(KB, SUBST(θ, first), θ) do
      for each θ″ in FOL-BC-AND(KB, rest, θ′) do
        yield θ″
```

Figure 9.6 A simple backward-chaining algorithm for first-order knowledge bases.

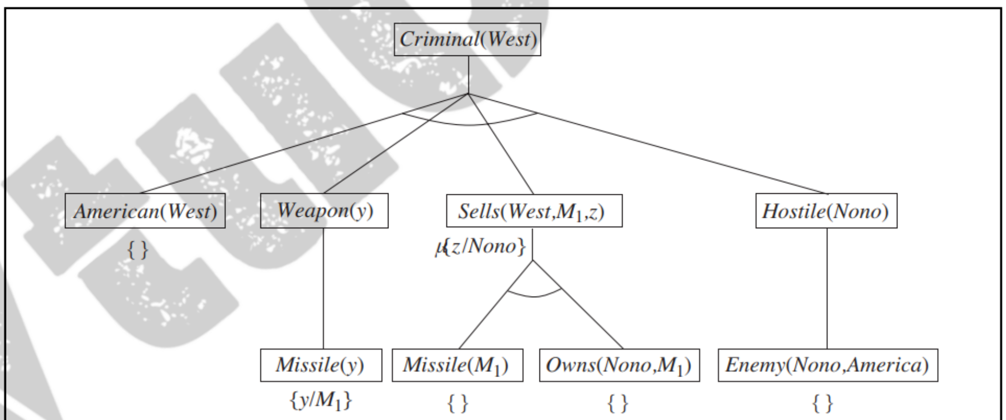

Figure 9.7 Proof tree constructed by backward chaining to prove that West is a criminal. The tree should be read depth first, left to right. To prove $Criminal(West)$, we have to prove the four conjuncts below it. Some of these are in the knowledge base, and others require further backward chaining. Bindings for each successful unification are shown next to the corresponding subgoal. Note that once one subgoal in a conjunction succeeds, its substitution is applied to subsequent subgoals. Thus, by the time FOL-BC-ASK gets to the last conjunct, originally $Hostile(z)$, z is already bound to $Nono$.

6	Explain and draw a planning graph with an example	10	CO5	L3

10.3 PLANNING GRAPHS

PLANNING GRAPH

All of the heuristics we have suggested can suffer from inaccuracies. This section shows how a special data structure called a **planning graph** can be used to give better heuristic estimates. These heuristics can be applied to any of the search techniques we have seen so far. Alternatively, we can search for a solution over the space formed by the planning graph, using an algorithm called GRAPHPLAN.

A planning problem asks if we can reach a goal state from the initial state. Suppose we are given a tree of all possible actions from the initial state to successor states, and their successors, and so on. If we indexed this tree appropriately, we could answer the planning question "can we reach state G from state S_0" immediately, just by looking it up. Of course, the tree is of exponential size, so this approach is impractical. A planning graph is polynomial-size approximation to this tree that can be constructed quickly. The planning graph can't answer definitively whether G is reachable from S_0, but it can *estimate* how many steps it takes to reach G. The estimate is always correct when it reports the goal is not reachable, and it never overestimates the number of steps, so it is an admissible heuristic.

LEVEL

A planning graph is a directed graph organized into **levels**: first a level S_0 for the initial state, consisting of nodes representing each fluent that holds in S_0; then a level A_0 consisting of nodes for each ground action that might be applicable in S_0; then alternating levels S_i followed by A_i; until we reach a termination condition (to be discussed later).

$Init(Have(Cake))$
$Goal(Have(Cake) \land Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \land Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$

Figure 10.7 The "have cake and eat cake too" problem.

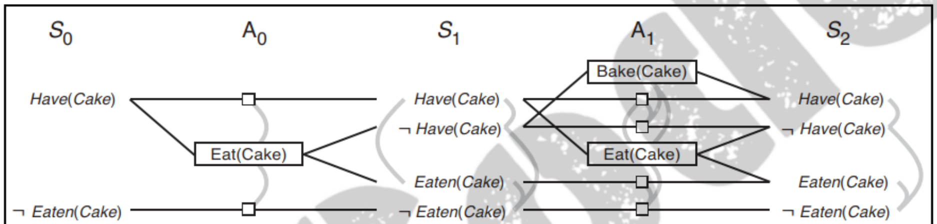

Figure 10.8 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions), and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines. Not all mutex links are shown, because the graph would be too cluttered. In general, if two literals are mutex at S_i, then the persistence actions for those literals will be mutex at A_i and we need not draw that mutex link.

Faculty Signature CCI Signature HOD Signature