Semester: 5th (Repeat/Make up)
Subject Name:- AI & Code:- CS 3011
Branch (s): - CS/IT/CSCE/CSSE

AUTUMN MID SEMESTER EXAMINATION-2023

School of Computer Engineering
Kalinga Institute of Industrial Technology, Deemed to be University
Artificial Intelligence CS 3011

Solution Scheme for Make-up Mid-Semester AI. This is a reference (base) answers to evaluate the answer scripts. Professors are requested to apply their own minds if they feel that it is necessary. Marks distribution is at the end.

1. Answer all the questions. [2 × 5 = 10 Marks]

a) Is "Turing Test" a valid test for computer vision-based AI task? What is Loebner Prize competition? *(It is to be noted that this competition has been discontinued since 2020.)*

Answer:

The Turing Test is not a valid test for computer vision-based AI tasks as it primarily evaluates natural language understanding and conversation, which is not directly related to computer vision.

The Loebner Prize competition is an annual competition in which chatbots or conversational AI systems compete in text-based conversations with human judges to determine which one can provide the most human-like responses. It's a benchmark for evaluating progress in conversational AI but doesn't assess computer vision tasks.

b) Can there be more than one agent program that implements a given agent function? Give an example or show why one is not possible.

Answer:

Yes, there can be more than one agent program that implements a given agent function. This is a fundamental concept in artificial intelligence and agent-based systems. An agent function specifies what an agent should do in response to different situations, but there can be multiple ways to implement this function. Different agent programs can achieve the same goals by employing different algorithms, strategies, or approaches.

For example, consider a simple agent function: "Find the shortest path from point A to point B in a grid." Several agent programs can implement this function, each using a different pathfinding algorithm such as Dijkstra's algorithm, A* search, or the Floyd-Warshall algorithm. These different programs would all achieve the same objective (finding the shortest path) but would do so using different computational methods.

c) Which of the following search algorithms consider both heuristic value and actual values in function calculation? Explain with a suitable example.

I. Uniform Cost Search, II. Best First Search, III. **A* Search**

Answer:

A* Search

d) Find out about the moon rover "Pragyan" in India's Chandrayaan-3.

I. Characterize the operating environment.
II. What are the actions this moon rover agent can take?
III. How can one evaluate the performance of the agent?
IV. What sort of agent architecture do you think is most suitable for this agent?

Answer:

I. **The environment (the Moon surface)**
i. partially observable,
ii. non-deterministic,
iii. sequential,
iv. dynamic,
v. continuous, and
vi. may be single-agent. If a rover must cooperate with its mother ship or other rovers, or if mischievous Martians tamper with its progress, then the environment gains additional agents

II. **The moon rover "Pragyan" has**
i. motor-driven wheels for locomotion
ii. along with a robotic arm to bring sensors close to interesting rocks and a
iii. rock abrasion tool (RAT) capable of efficiently drilling 45mm holes in hard volcanic rock.
iv. Spirit also has a radio transmitter for communication.

III. **Performance measure: A Moon rover may be tasked with**
i. maximizing the distance or variety of terrain it traverses,
ii. or with collecting as many samples as possible,
iii. or with finding life (for which it receives 1 point if it succeeds, and 0 points if it fails).

Criteria such as maximizing lifetime or minimizing power consumption are (at best) derived from more fundamental goals; e.g., if it crashes or runs out of power in the field, then it can't explore.

IV. **A model-based reflex agent is suitable for low level navigation.**
For route planning, experimentation etc, some combination of goal-based, and utility-based would be needed.

e) Which of the following is not a component of the task environment? Justify your answer.
I. Sensors, II. Actuators, III. Performance Measures, IV. Agent functions

Answer:

The task environment comprises several components, including Performance Measures, Environment, Actuators, Sensors (PEAS). However, "**Agent functions**" is not a component of the task environment. Here's the justification:

Sensors: Sensors are responsible for perceiving the environment. They gather information from the environment and provide it to the agent. Sensors are crucial for the agent to understand its surroundings and make informed decisions.

Actuators: Actuators are responsible for taking actions based on the agent's decisions. They execute the actions the agent has chosen to influence the environment. Actuators are essential for the agent to interact with and impact the environment.

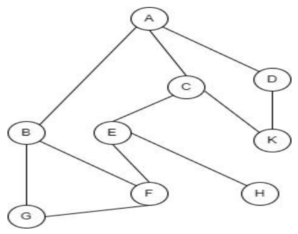
Performance Measures: Performance measures are used to evaluate how well the agent is doing in achieving its objectives or goals within the task environment. They are critical for assessing the success and efficiency of the agent's actions.

Agent functions: Agent functions refer to the algorithms or methods that the agent employs to determine its actions based on the information it receives from sensors. These functions can include decision-making processes, planning, learning, and other cognitive processes that enable the agent to choose its actions.

So, in summary, all components listed (I. Sensors, II. Actuators, III. Performance Measures, and IV. Agent functions) play a role in the overall functioning of an intelligent agent within its task environment. "**Agent functions**" is not a component of the task environment itself but represents the internal processes and algorithms used by the agent to interact with the environment effectively.

2. [5+5=10 Marks]

a) Differentiate between informed search and uninformed search. Show the working of BFS (Breadth First Search) algorithm on the following graph step by step. 'A' is the your start state and 'G' is the your goal state.



Answer:

Informed Search

1.It is also known as heuristic Search.

2.It contains additional information that helps us reach the goal state.

3.The solution can be found much quicker.

4.Example, A*, Best First Search, etc.

Uninformed Search

1.It is also known as blind search.

2.There is no additional information that helps us reach the goal state.

3.It performs an exhaustive search and, therefore may explore uninteresting parts of the problem.

4.Examples, BFS, DFS, etc. (2 Marks)

Step 1

Q = [A]

Step 2

Dequeue A

Q = [B C D]

Step 3

Dequeue B

Q = [C D G F]

Step 4

Dequeue C

Q = [D G F E K]

Step 5

Dequeue D

Q = [G F E K]

Step 6

Dequeue G

G is the Goal State (3 Marks)

b) Define and design the state-space graph "Water Jug Problem". You are given two empty jugs: one with a capacity of 'A' liters and the other with a capacity of 'B' liters. There is no measuring scale in any of these jugs. The goal is to measure a specific quantity of water, usually denoted as 'C' liters, using these two jugs. You have access to a water source, which can be used to fill the jugs, and a drain to empty the jugs. The operations you can perform are:

a) Fill a jug from the water source until it is full.

b) Empty a jug and pour its contents down the drain.

c) Pour the water from one jug into the other jug until the receiving jug is full or the source jug is empty.

The Water Jug Problem is either to find a sequence of these operations that will eventually result in having 'C' liters of water in one of the jugs or to prove that it is not possible to obtain 'C' liters of water using the given jugs and operations.

Now, design the state-space graph for "Two-Water-Jug Problem" with the given data A=5, B=3 and C=4 and find the sequence of operations or states from initial state of two empty jugs A and B to the goal state as stated above.

Answer:

Fill the 3-gallon jug and pour it into the 5-gallon jug. Fill the 3-gallon jug again and use it to fill the 5-gallon jug. You have 1 gallon. Empty the 5-gallon jug and pour the 3-gallon jug's remaining gallon in. Refill the 3-gallon jug and pour it into the 5-gallon jug.

(X, Y) corresponds to a state where X refers to the amount of water in Jug1 and Y refers to the amount of water in Jug2. Determine the path from e initial state (xi, yi) to the final state (xf, yf), where (xi, yi) is

(0, 0) which indicates both Jugs are initially empty and (xf, yf) indicates a state which could be (0, d) or (d, 0). Here, the initial state is (0,0) and final state is (0,4). The operations you can perform are:

- Empty a jug (X, 0)->(0, 0) Empty Jug 1.
- Fill a Jug, (0, 0)->(X, 0) Fill Jug 1
- Pour water from one jug to the other until one of the jugs is either empty or full, (X, Y) -> (X-d, Y+d)

Using Solution 1, successive pouring steps are:

(0,0)->(3,0)->(0,3)->(3,3)->(1,5)->(1,0)->(0,1)->(3,1)->(0,4)

Hence the no of operations you need to perform are 8.

Using Solution 2, successive pouring steps are:

(0,0)->(0,5)->(3,2)->(0,2)->(2,0)->(2,5)->(3,4)

Hence the no of operations you need to perform are 6.

Therefore, we would use solution 2 to measure 4 liters of water in 6 operations or moves.

3. [5+5=10 Marks]

a) An initial state (IS) and a goal state (GS) of the 8-Puzzle problem have been given below. Find the sequence of states from IS to GS by clearly mentioning the action that is taken for each transition from the one state to the next state. Here any action can be any one the four moving actions of the blank space i.e. UP/ DOWN/ RIGHT/ LEFT.

Initial State

2	8	
3	4	1
6	7	5

Goal State

2	4	
6	1	8
7	3	5

Answer:

The 8-puzzle is a sliding puzzle that consists of an empty space and 8 numbered tiles (usually labeled 1 through 8) placed on a 3x3 grid. The goal is to rearrange the tiles from their initial state to a goal state by sliding them one at a time into the empty space.

State Space:The state space in the 8-puzzle problem consists of all possible configurations of the 3x3 grid. Each configuration represents a state, where the position of each tile is specified.

Initial State: The initial state is the starting configuration of the puzzle.

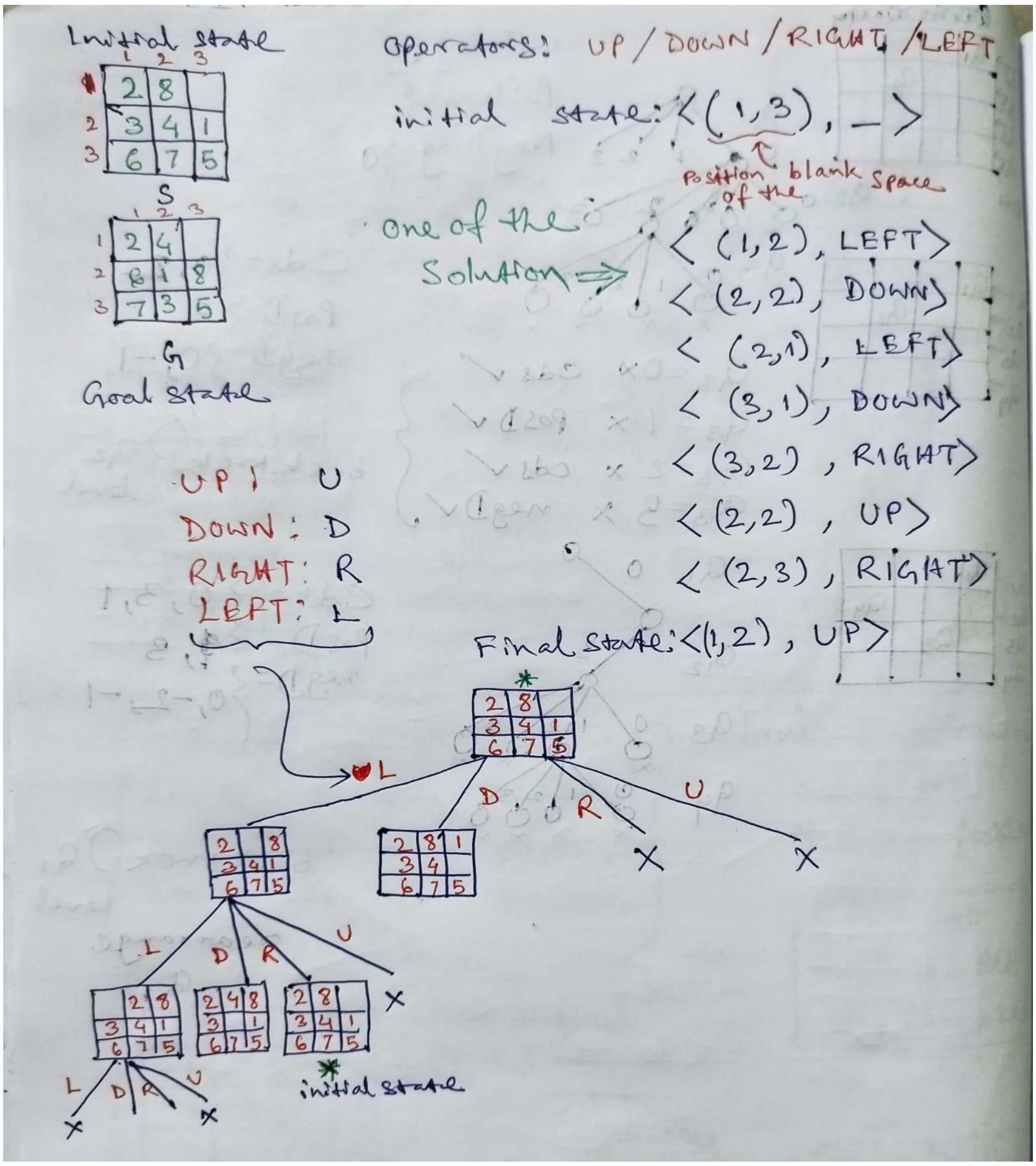
Goal State: The goal state is a specific configuration of the 3x3 grid where the tiles are arranged in a particular order (usually in ascending order from left to right, top to bottom, with the empty space at a specific location).

Operators: The operators are the actions that can be taken in the puzzle. In the case of the 8-puzzle, there are four possible actions: move a tile either up, down, left, or right into the empty space.

Transition Model: The transition model defines how the state changes when an operator is applied. When a tile is moved into the empty space, the resulting state is a new configuration of the grid.

Cost: The cost of reaching the goal state can be defined as the number of moves (operator applications) required to transform the initial state into the goal state. In many cases, each move has a uniform cost of 1, making the cost function straightforward.

State Space Tree: The state space tree represents all possible states that can be reached from the initial state by applying the defined operators. Each node in the tree represents a state, and the edges between nodes represent the application of an operator to transition from one state to another.



b) “Monotonic (i.e. consistent) heuristic function is admissible over a non-monotonic heuristic” and “Underestimation of heuristic guarantees otimality over overestimation of heuristic value”-- justify your answers in the context of A* search algorithm.

Answer:

In the context of heuristic functions used in informed search algorithms, like A* search, it's important to understand the concepts of admissibility and monotonicity.

Admissible Heuristic Function: A heuristic is admissible if it never overestimates the true cost to reach the goal from any given state. In other words, for all states, $h(n) \leq h^*(n)$, where h(n) is the estimated cost from state n to the goal, and h*(n) is the true (optimal) cost from state n to the goal.

Monotonic (Consistent) Heuristic Function: A heuristic is said to be monotonic (or consistent) if it satisfies the triangle inequality. In other words, for every state n and its successor n', the heuristic follows the inequality $h(n) \leq c(n, n') + h(n')$, where $c(n, n')$ is the cost of getting from state n to state n'. This means that the estimated cost to reach the goal from any state is not only admissible but also consistently non-decreasing along the path.

Now, let's address your statement: "**Monotonic (i.e. consistent) heuristic function is admissible over a non-monotonic heuristic.**"

This statement is not necessarily true. In fact, a non-monotonic heuristic function may or may not be admissible. Monotonicity (or consistency) and admissibility are separate properties of heuristic functions. A monotonic heuristic function is always admissible. This is because the triangle inequality ensures that the heuristic doesn't overestimate the cost to reach the goal, which is the definition of admissibility. A non-monotonic heuristic function can still be admissible if it doesn't overestimate the true cost to reach the goal for all states. Admissibility is a stricter requirement than monotonicity, and it only concerns the upper bound of the heuristic estimate.

So, in summary, a consistent (monotonic) heuristic function is not just admissible over a non-monotonic heuristic; it's admissible over any heuristic function, whether monotonic or not. However, admissibility is a fundamental property that all heuristics should possess in order to ensure the correctness of search algorithms like A*. Monotonicity is a stronger condition that, if met, can improve the efficiency of these algorithms but is not a prerequisite for admissibility.

"Underestimation of heuristic guarantees otimality over overestimation of heuristic value"

In A* search, the choice of heuristic function plays a critical role in determining the algorithm's efficiency and optimality. Specifically, A* uses a heuristic function h(n) to estimate the cost from a given state n to the goal state. Whether the heuristic underestimates or overestimates the true cost affects the algorithm's guarantees and performance. Here's why underestimation of the heuristic guarantees optimality over overestimation:

Optimality Guarantee:

Underestimation (Admissibility): If the heuristic function h(n) is admissible, meaning it never overestimates the true cost to reach the goal from any state ($h(n) \leq h^*(n)$), A* search is guaranteed to find an optimal solution. This is because A* expands nodes in increasing order of their estimated cost ($f(n) = g(n) + h(n)$), and if h(n) is admissible, then f(n) is a lower bound on the true cost to reach the goal. This ensures that when the goal state is expanded, the optimal solution is found.

Overestimation (Non-Admissibility): If the heuristic function overestimates the true cost for some states, A* may still find a solution, but it is not guaranteed to be optimal. The overestimation can lead the algorithm to explore unnecessary states, potentially resulting in suboptimal solutions.

Efficiency:

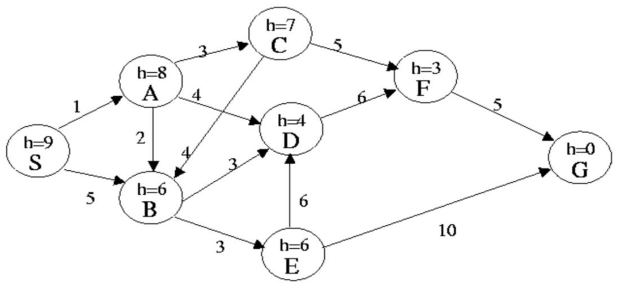
Underestimation: An underestimating heuristic is usually more informative. It guides the search towards the goal efficiently because it encourages the exploration of nodes that are closer to the goal. A* with an admissible heuristic will typically expand fewer nodes than A* with an overestimating heuristic.

Overestimation: An overestimating heuristic, while still guiding the search in the right direction, may cause A* to explore a larger portion of the search space, potentially leading to a less efficient search. It could expand more nodes, including ones that are farther from the goal.

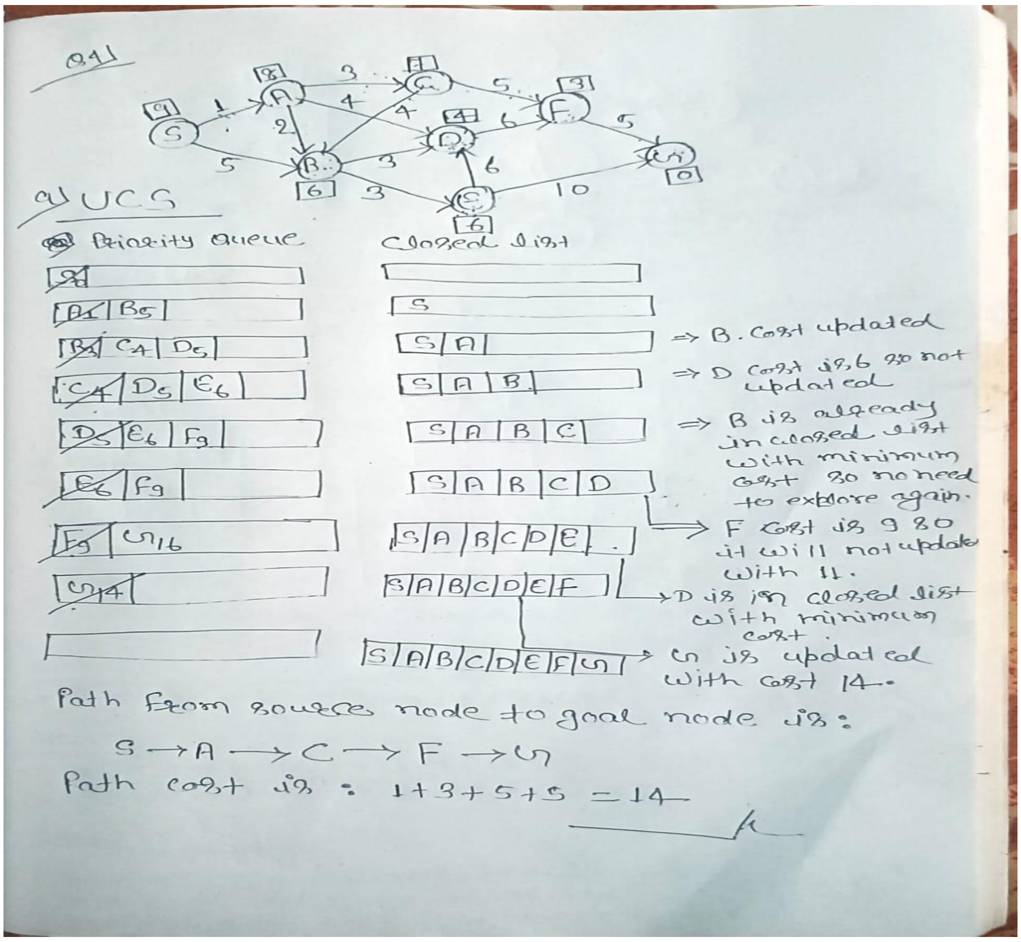
In summary, underestimation of the heuristic is preferred in A* because it guarantees optimality and generally leads to more efficient searches. Overestimation, while not inherently wrong and still useful for finding solutions, does not provide the same guarantees and can lead to less efficient search processes. Therefore, for optimal results, it's best to use admissible (underestimating) heuristics with A* when possible.

4. [5+5=10 Marks]

Apply UCS and A* search algorithms for the following graph and for each case, a) find the path from source node to the goal node and b) obtain the corresponding path cost. Here, S is the source node and G is the goal node.



Answer:


Q4)
9
8
7
4
3
6
6
0
1
5
2
3
4
4
3
3
5
6
6
5
10

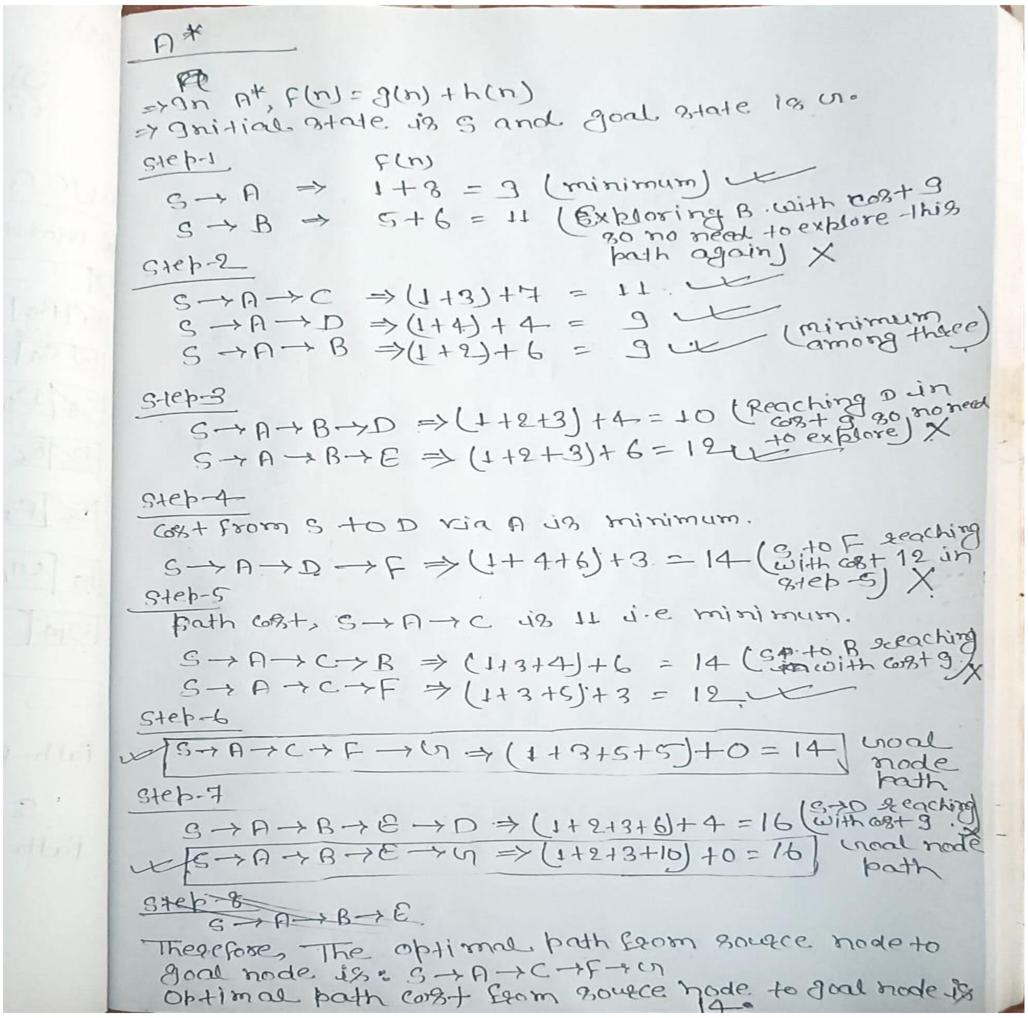
a) UCS

Priority Queue	Closed list	
~~S0~~		
~~A1~~ | B5	S	
~~B3~~ | C4 | D5	S | A	⇒ B. Cost updated
~~C4~~ | D5 | E6	S | A | B	⇒ D cost is 6 so not updated
~~D5~~ | E6 | F9	S | A | B | C	⇒ B is already in closed list with minimum cost so no need to explore again.
~~E6~~ | F9	S | A | B | C | D	→ F cost is 9 so it will not update with 11.
~~F9~~ | G16	S | A | B | C | D | E	
~~G14~~	S | A | B | C | D | E | F	→ D is in closed list with minimum cost.
	S | A | B | C | D | E | F | G	→ G is updated with cost 14.

Path from source node to goal node is:

$S \rightarrow A \rightarrow C \rightarrow F \rightarrow G$

Path cost is : $1+3+5+5 = 14$



A*

~~A~~
$\Rightarrow$ In A*, $f(n) = g(n) + h(n)$
$\Rightarrow$ Initial state is S and goal state is G.

Step-1

$f(n)$

$S \rightarrow A \Rightarrow 1 + 8 = 9$ (minimum) ✓

$S \rightarrow B \Rightarrow 5 + 6 = 11$ (Exploring B with cost 9 so no need to explore this path again) ✗

Step-2

$S \rightarrow A \rightarrow C \Rightarrow (1+3)+7 = 11$ ✓

$S \rightarrow A \rightarrow D \Rightarrow (1+4)+4 = 9$ ✓

$S \rightarrow A \rightarrow B \Rightarrow (1+2)+6 = 9$ ✓ (minimum among three)

Step-3

$S \rightarrow A \rightarrow B \rightarrow D \Rightarrow (1+2+3)+4 = 10$ (Reaching D in cost 9 so no need to explore) ✗

$S \rightarrow A \rightarrow B \rightarrow E \Rightarrow (1+2+3)+6 = 12$ ✓

Step-4

Cost from S to D via A is minimum.

$S \rightarrow A \rightarrow D \rightarrow F \Rightarrow (1+4+6)+3 = 14$ (S to F reaching with cost 12 in step-5) ✗

Step-5

Path cost, $S \rightarrow A \rightarrow C$ is 11 i.e minimum.

$S \rightarrow A \rightarrow C \rightarrow B \Rightarrow (1+3+4)+6 = 14$ (S to B reaching with cost 9) ✗

$S \rightarrow A \rightarrow C \rightarrow F \Rightarrow (1+3+5)+3 = 12$ ✓

Step-6

✓ $\boxed{S \rightarrow A \rightarrow C \rightarrow F \rightarrow G \Rightarrow (1+3+5+5)+0 = 14}$ Goal node path

Step-7

$S \rightarrow A \rightarrow B \rightarrow E \rightarrow D \Rightarrow (1+2+3+6)+4 = 16$ (S→D reaching with cost 9) ✗

✓ $\boxed{S \rightarrow A \rightarrow B \rightarrow E \rightarrow G \Rightarrow (1+2+3+10)+0 = 16}$ Goal node path

~~Step-8~~
~~$S \rightarrow A \rightarrow B \rightarrow E$~~

Therefore, The optimal path from source node to goal node is: $S \rightarrow A \rightarrow C \rightarrow F \rightarrow G$
Optimal path cost from source node to goal node is 14.

Question Number	Mandatory	Marks Distribution
1(a)	Definition and mention of text-based/non-vision-based test	1+1
1(b)	Mention of "Yes"	1, justification 1
1(c)	Correct Answer	2
1(d)	Answer based on PEAS	2, otherwise partial marking
1(e)	Agent Function	1, justification 1
2(a)	Mention of at least 3 points between informed and uninformed searches Correct solution	2 + 3 Partial marking for incomplete answers
2(b)	Definition and correct solution	5, only solution 2.5 (with out any explanation)
3(a)	Definition and correct solution + starting of the state space tree	5, only solution 2.5 (with out any explanation) With out state space tree 4
3(b)	Justification for both the statements	2.5 + 2.5 Partial marking for incomplete answers
4(a)	Brief discussion on UCS and correct implementation	5, only implementation 4, Partial marking for incomplete answer
4(b)	Brief discussion on UCS and correct implementation	5, only implementation 4, Partial marking for incomplete answer