Name:	Kathan Shah	Roll No.:	21BCP456D
Internal Assessment: 1:	Unit 1 AI	Semester: 6	Division: 6

Answer the following with appropriate diagrams where necessary.

1. **Complete the following table with relevant analysis of the systems mentioned in first column.**

System	Performance Measure (P)	Environment (E)	Actuator (A)	Sensor (S)
Lego-based House Design Assistance from available Legos	Accuracy and efficiency in creating Lego-based designs	Indoor, controlled environment with Lego components	Robotic arm or mechanism for placing Legos	- Cameras or computer vision for Lego detection
Augmented reality environment for toddler learning	Engagement, learning outcomes	Virtual environment overlaid on the real world	Display for visual augmentation	Cameras for environment perception
Water Jug Problem Solution	Optimality of solution	Physical space with water jugs	Mechanism for pouring water between jugs	Sensors to detect water levels
Tic tac toe	Strategic decision-making, win/loss rates	Virtual or physical board for playing	Mechanism for moving game pieces	Sensors for detecting player moves
Grass trimming robot	Efficiency in grass cutting	Outdoor environment with grass	Mower blades or cutting mechanism	Sensors for obstacle detection, navigation

2. **Considering the generic search algorithm mentioned below complete the given table.**

```
function Search(graph, start, goal):

        0. Initialize

                agenda = [ [start] ]

                extended_list = []

while agenda is not empty:

        1. path = agenda.pop(0) # get first element from agenda & return it

        2. if is-path-to-goal(path, goal) return path

        3. otherwise extend the current path if not already extended

          for each connected node

               make a new path (don't add paths with loops!)
```

4. add new paths from 3 to agenda and reorganize agenda

(algorithms differ here see table below)

Fail!

Search Algo	Properties	Type of problems it can be applicable to	Example of problem	Type of agent(s) generally using the algorithm	What it does with agenda in step 4
BFS	Complete, Optimal for un-weighted graphs	Shortest path in un-weighted graphs	Finding shortest path	Agents exploring unknown territory	Add new paths to the end of the queue
DFS	Not complete, not optimal	Solving puzzles, finding solutions	Maze solving, gaming trees	Agents with limited memory	Add new paths to the front of the stack
Uniform Cost	Complete, optimal for non-negative costs	Shortest path with varying edge costs	Routing in networks	Agents with cost-aware decisions	Add new paths based on path cost
Iterative Deepening	Complete, Optimal for unit step	Finding shortest path	Puzzle solving	Memory constrained agents	Add new paths up to a depth
Depth Limited Search	Not complete, not optimal	Finding solutions within depth limit	AI game playing	Agents with depth-restricted search	Add new paths up to a depth limit
Bidirectional Search	Complete, optimal for some problems	Shortest path in certain cases	Network routing, puzzle solving	Memory constrained agents	Merge paths from both directions

3. Write an algorithm for bidirectional search implementing BFS in both subtrees.

1. Input: Graph `G`, start node start , and goal node goal .
2. Create two empty queues: forward_queue and backward_queue.
   forward_queue is for the forward BFS from the start node.
   backward_queue is for the backward BFS from the goal node.
3. Create two empty sets: forward_visited and backward_visited .
   forward_visited keeps track of visited nodes in the forward search.
   backward_visited keeps track of visited nodes in the backward search.

4. Enqueue (start, None) into forward_queue and add start to forward_visited .

5. Enqueue (goal, None) into backward_queue and add goal to backward_visited .

6. While both forward_queue and backward_queue are not empty:
   Perform forward BFS:
    Dequeue a node (current_forward, parent_forward) from forward_queue .
    For each neighbor neighbor of current_forward :
     If neighbor is not in forward_visited :
      Enqueue (neighbor, current_forward) into forward_queue .
      Add neighbor to forward_visited .

   Perform backward BFS:
    Dequeue a node (current_backward, parent_backward) from `backward_queue`.
    For each neighbor `neighbor` of `current_backward`:
     If `neighbor` is not in `backward_visited`:
      Enqueue `(neighbor, current_backward)` into `backward_queue`.
      Add `neighbor` to `backward_visited`.

   Check for intersection:
    If there is an intersection between forward_visited and backward_visited, let's call it intersection_node :
     Reconstruct the path from start to goal :
      Starting from intersection_node , follow the parent_forward pointers until None (excluding).
      Starting from intersection_node , follow the parent_backward pointers until None .
     Return the combined path.

7. If the queues are empty and no intersection is found, return None as there is no path.

4. **Comment on the requirement of Heuristic search techniques. Mention its benefits and drawbacks as compared to blind search techniques.**

Benefits:

1. **Efficiency Improvement:**
    - Heuristic search techniques use domain-specific information, known as heuristics, to guide the search efficiently.
2. **Faster Convergence:**
    - Heuristics help in focusing the search on promising areas, leading to faster convergence to the goal state.
3. **Adaptability to Problem Structure:**
    - Heuristic techniques can adapt to the specific structure and characteristics of the problem

Drawbacks:

1. **Incomplete Solutions:**
   - Heuristic methods might not always find the optimal solution.
2. **Sensitivity to Heuristic Quality:**
   - The effectiveness of heuristic search depends on the quality of the heuristic function.
3. **Difficulty in Heuristic Design:**
   - Designing effective heuristics can be challenging for certain problems.

5. Give the details to complete the following heuristic search comparison table:

Technique	Properties	Required parameters	Performance Measures	What it does with agenda in step 4 of generic algo mentioned in Q-2
Greedy Best First Search	Incomplete, fast	Heuristic function	Time complexity, Solution quality	Adds new paths based on heuristic value only
Heuristic Search	Complete if consistent, Efficient	Heuristic function	Time complexity, Solution quality	Adds new paths based on heuristic value combined with path cost
A* Search	Complete, Optimal if consistent, Efficient	Heuristic function, Cost function	Time complexity, Solution quality	Adds new paths based on total estimated cost
Simulated Annealing	Probabilistic, Suitable for optimization problems	Temperature schedule, Cooling rate	Time complexity, Solution quality	Accepts worse solutions with a certain probability based on temperature