ARTIFICIAL INTELLIGENCE

ASSIGNMENT # 1

MUHAMMAD REHAN BSDSF22A001

Q#1:-

Represent the following problems as state-space.

(i) Missionaries & Cannibals:

States: A state is defined by the number of missionaries (M), cannibals (C), and the boat's location (B) on either side of the river.

Initial State: (3,3,1)

Goal State: (0,0,0)

Transitions: The only valid transitions are those that do not allow cannibals to outnumber missionaries on either side.

(ii) Tower of Hanoi:

States A state is defined by the arrangement of disks on the three pages.

Initial State: All disks are on page 1.

Goal State: All disks are on page 3.

Transitions: A disk can be moved from one page to another if it is the top disk and if it is smaller than the top disk on the destination page.

(iii) **Bridges of Königsberg:**

States: A state is defined by the set of visited and unvisited landmasses

Initial State: All landmasses are unvisited

Goal State: All landmasses are visited.

Transitions: A transition is made by crossing a bridge to an unvisited landmass.

(iv) **Traveling Salesperson:**

States: A state is defined by the set of cities visited and the current city.

Initial State A state where no cities have been visited and the current city is the starting city.

Goal State A state where all cities have been visited and the current is the starting city

Transitions: A transition is made by travelling from current city to an unvisited city.

(v) **Knapsack:**

States: A state is defined by the remaining capacity of the knapsack and the items that have been selected.

Initial State: The initial capacity of knapsack and no items selected.

Goal State: The maximum value of items in kapsack without exceeding the capacity.

Transitions A transition is made by either selecting an item if there is enough capacity or not selecting an item.

Q#2: Give pseudocodes of following algos.

(i) Beam Search:

Beam search is a greedy algorithm that explores a fixed no. of nodes at each level of search tree. It chooses the nodes with the highest heuristic values to expand.

Pseudocod:

```
function beam_search (initial_state, goal_test, successor_fn,
heuristic, beam_width)
    frontier = [initial_state]
    explored = { }
    while frontier is not empty.
        next_level = [ ]
        for each node in frontier
            if goal_test (node)
                return node
            explored.add (node)
            Successors = successor_fn (node)
            for each successor in successors
                if successor not in explored
                and next_level.size() < beam_width
                    next_level.add (successor)
                    Successor.parent = node
                    successor.g = node.g + cost_of_transition
                                  (node, successor)
                    successor.h = heuristic (successor)
                    successor.f = successor.g + successor.h
        frontier = next_level
    return failure
```

(ii) Iterative Deepening A* Search:

It is a combination of DFS and A* search. It starts with a depth limit of 1 and iteratively increases depth limit until it finds a solution.

Pseudocode

```
function i_astar (initial_state, goal_test, successor_fn, heuristic)
    for depth in 1..∞
        result = astar (initial_state, goal_test, successor_fn,
                        heuristic, depth)
        if result != failure
            return result

function astar (initial_state, goal_test, successor_fn, heuristic, depth)
    frontier = [initial_state]
    explored = {}
    while frontier is not empty
        node = frontier.pop()
        if goal_test (node)
            return node
        explored.add(node)
        if node.depth >= depth_limit
            continue
        successors = successor_fn (node)
        for each successor in successors
            if successor not in explored
                successor.parent = node
                successor.g = node.g + cost_of_transition
                              (node, successor)
                successor.h = heuristic (successor)
                successor.f = successor.g + successor.h
                frontier.add (successor)

    return failure
```

(iii) Bidirectional Search:

It searches from both initial state & goal state simultaneously, hoping to meet in the middle.

Pseudocode

```
function bidirectional (initial_state, goal_state, successor_fn, heuristic)
    forward_frontier = [initial_state]
    backward_frontier = [goal_state]
    explored_forward = { }
    explored_backward = { }
    while forward_frontier is not empty and backward_frontier
    is not empty
        // Expand forward frontier
        next_forward_level = [ ]
        for each node in forward_frontier
            if node in explored_backward
                return node
            explored_forward.add(node)
            successors = successor_fn(node)
            for each successor in successors
                if successor not_in explored_forward
                    next_forward_level.add(successor)
                    successor.parent = node
                    successor.g = node.g + cost_of_transition(node, successor)
                    successor.h = heuristic(successor)
                    successor.f = successor.g + successor.h
        forward_frontier = next_forward_level
        // Expand backward frontier
        next_backward_level = [ ]
```

```
        for each node in backward_frontier
            if node in explored_forward
                return node
        explored_backward.add(node)
        successors = successor_fn_reverse(node)
        for each successor in successors:
            if successor not in explored_backward
                next_backward_level.add(successor)
                successor.parent = node
                successor.g = node.g + cost_of_transition(node,
                                                          successor)
                successor.h = heuristic(successor)
                successor.f = successor.g + successor.h
        backward_frontier = next_
    return failure
```

(iv) Simulated Annealing Search

It is a probabilistic search algorithm to occasionally accept suboptimal solutions to explore different regions.

Pseudocode

```
function simulated_annealing(initial_state, successor_fn, heuristic, temp_schedule)
    current_state = initial_state
    best_state = current_state
    for each iteration
        temperature = temp_schedule(iteration)
        next_state = successor_fn(current_state)
        delta_e = heuristic(next_state) - heuristic(current_state)
        if delta_e < 0 or random() < math.exp(-delta_e /
                                              temperature)
            current_state = next_state
        if heuristic(current_state) < heuristic(best_state)
            best_state = current_state
    return best_state
```

Q#3: (a) Return paths for algorithms:

(i) BFS: A → B → D → F

(ii) UCS: A → C → B → D → F

(iii) DFS: A → B → D → E → F

(iv) A*: A → C → B → D → F

Q#3: (b) Admissible & monotonic

⇒ h1 is both admissible and monotonic because it always underestimates the cost at each point and heuristic value doesn't decrease as we move towards goal.

⇒ h2 is neither admissible and nor monotonic because it violates the conditions mentioned for h1 above.