Exp 1

Aim: Implementation of Toy Problem using python

Problem Statement: A person has 3000 bananas and a camel. The person wants to transport the maximum number of bananas to a destination which is 1000 KMs away, using only the camel as a mode of transportation. The camel cannot carry more than 1000 bananas at a time and eats a banana every km it travels. What is the maximum number of bananas that can be transferred to the destination using only camel (no other mode of transportation is allowed.

Algorithm: <---p1---><--------p2-----><-----p3----> A---------->B 3000km
Since there are 3000 bananas and the Camel can only carry 1000 bananas, he will have to make 3 trips to carry them all to any point in between. When bananas are reduced to 2000 then the Camel can shift them to another point in 2 trips and when the number of bananas left is <= 1000, then he should not return and only move forward. In the first part, P1, to shift the bananas by 1Km, the Camel will have to 1. Move forward with 1000 bananas – Will eat up 1 banana on the way forward 2. Leave 998 bananas after 1 km and return with 1 banana – will eat up 1 banana on the way back 3. Pick up the next 1000 bananas and move forward – Will eat up 1 banana on the way forward 4. Leave 998 bananas after 1 km and return with 1 banana - will eat up 1 banana on the way back 5. Will carry the last 1000 bananas from point a and move forward – will eat up 1 banana Note: After point 5 the Camel does not need to return to point A again. So to shift 3000 bananas by 1km, the Camel will eat up 5 bananas. After moving to 200 km the Camel would have eaten up 1000 bananas and is now left with 2000 bananas. Hence the length of part P1 is 200 Km. Now in Part P2, the Camel needs to do the following to shift the Bananas by 1km. 1. Move forward with 1000 bananas - Will eat up 1 banana on the way forward 2. Leave 998 bananas after 1 km and return with 1 banana - will eat up this 1 banana on the way back 3. Pick up the next 1000 bananas and move forward - Will eat up 1 banana on the way forward Note: After point 3 the Camel does not need to return to the starting point of P2. So to shift 2000 bananas by 1km, the Camel will eat up 3 bananas. After moving to 333 km the camel would have eaten up 1000 bananas and is now left with the last 1000 bananas. Because it is a multiple of 3, after 333 times there are 1001 bananas left. Therefore, the camel must take 1000 bananas and then take 1 back with him to pick up the last banana, leaving him with 999 bananas. This would leave the camel at 333 km + 200 km, so the merchant only needs to travel 466 km, eating 1 banana for every km. 1000 bananas - 466 bananas for each remaining km equals 534 bananas left at point B. Therefore, the merchant has 533 bananas to sell at point B.

Exp 2

Aim: Implementation of Real World Problems.

Problem Statement:

(1) TSP (Travelling salesman problem) Given a set of cities and distance between every pair of cities as an adjacency matrix, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

(2) Vacuum cleaner to remove dirt in two rooms Developed a simple reflex agent program in Python for the vacuum-cleaner world problem. This program defines the States, Goal State, Goal Test, Actions, Transition Model, and Path Cost. For each possible initial state, the program returns a sequence of actions that leads to the goal state, along with the path cost. Where A and B are the two adjacent rooms, 0 means CLEAN, and 1 means DIRTY.

TSP (Travelling salesman problem) Algorithm:
● Consider city 1 as the starting and ending point. Since the route is cyclic, we can consider any point as a starting point. ● Generate all (n-1)! permutations of cities. ● Calculate the cost of every permutation and keep track of the minimum cost permutation. ● Return the permutation with minimum cost.

Vacuum cleaner to remove dirt in two rooms Algorithm:
Initially, both the rooms are full of dirt and the vacuum cleaner can reside in any room. And to reach the final goal state, both the rooms should be clean and the vacuum cleaner again can reside in any of the two rooms. The vacuum cleaner can perform the following functions: move left, move right, move forward, move backward, and suck dust. But as there are only two rooms in our problem, the vacuum cleaner performs only the following functions here: move left, move right and suck. Here the performance of our agent (vacuum cleaner) depends upon many factors such as time taken in cleaning, the path followed in cleaning, the number of moves the agent takes in total, etc. But we consider two main factors for estimating the performance of the agent. They are: 1. Search Cost: How long the agent takes to come up with the solution. 2. Path cost: How expensive each action in the solution is.

Exp 3

Aim: Implementation of Constraint Satisfaction Problem.

Problem Statement: Given an array of strings, arr[] of size N and a string S, the task is to map integers value in the range [0,9] to every alphabet that occurs in the strings, such that the sum obtained after summing the numbers formed by encoding all strings in the array is equal to the number formed by the string S.

Algorithm:
X: It is a set of variables.
D: It is a set of domains where the variables reside. There is a specific domain for each variable.
C: It is a set of constraints which are followed by the set of variables.
The requirements to solve a constraint satisfaction problem (CSP) is:
A state-space
The notion of the solution.

Aim: Implementation of Graph Coloring Problem.

Problem Statement: The problem is, given m colors, find a way of coloring the vertices of a graph such that no two adjacent vertices are colored using the same color

Algorithm
1. Color first vertex with first color.
2. Do following for remaining V-1 vertices.
….. a) Consider the currently picked vertex and color it with the
lowest numbered color that has not been used on any previously
colored vertices adjacent to it. If all previously used colors
appear on vertices adjacent to v, assign a new color to it.

Breadth-First Search

Aim: To implement BFS(Breadth-first search) using python.

Problem Statement: Breadth-first search is a graph traversal algorithm that starts traversing the graph from the root node and explores all the neighboring nodes. Then, it selects the nearest node and explores all the unexplored nodes. While using BFS for traversal, any node in the graph can be considered as the root node.

Algorithm: ● SET STATUS = 1 (ready state) for each node in G ● Enqueue the starting node A and set its STATUS = 2 (waiting state) ● Dequeue a node N. Process it and set its STATUS = 3 (processed state). ● Enqueue all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state) ● Exit

Depth First Search

Aim: To implement DFS(Depth-first search) using python.

Problem Statement: Depth-first search (DFS) algorithm starts with the initial node of graph G, and then go to deeper and deeper until we find the goal node or the node which has no children. The algorithm then backtracks from the dead-end towards the most recent node that is yet to be completely unexplored.

Algorithm: ● SET STATUS = 1 (ready state) for each node in G ● Push the starting node A on the stack and set its STATUS = 2 (waiting state) ● Pop the top node N. Process it and set its STATUS = 3 (processed state) Push on the stack all the neighbors of N that are in the ready state (whose STATUS = 1) and set their STATUS = 2 (waiting state)

Best First Search

Aim: To implement the Best first search using python.

Problem Statement: Best First Search uses an evaluation function to decide which adjacent is most promising and then explore. Best First Search falls under the category of Heuristic Search or Informed Search. We use a priority queue to store the costs of nodes. So the implementation is a variation of BFS, we just need to change Queue to PriorityQueue.

Algorithm: ● Place the starting node into the OPEN list. ● If the OPEN list is empty, Stop and return failure. ● Remove the node n, from the OPEN list which has the lowest value of h(n), and place it in the CLOSED list. ● Expand the node n, and generate the successors of node n. ● Check each successor of node n, and find whether any node is a goal node or not. If any successor node is the goal node, then return success and terminate the search, else proceed to Step 6. ● For each successor node, the algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both lists, then add it to the OPEN list. ● Return to Step 2.

A* Best First Search

Aim: To implement the A* Best first search using python.

Problem Statement: A* search is the most commonly known form of best-first search. It uses the heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function.

Algorithm: ● Place the starting node in the OPEN list. ● Check if the OPEN list is empty or not, if the list is empty then return failure and stops. ● Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise ● Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute the evaluation function for n' and place it into the open list. ● Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value. ● Return to Step 2.

Unification

Aim: To implement Unification Algorithm.

Algorithm: ● Unification is the process of making two different logical atomic expressions identical by finding a substitution. Unification depends on the substitution process. ● It takes two literals as input and makes them identical using substitution. ● Let Ψ1 and Ψ2 be two atomic sentences and σ be a unifier such that, Ψ1σ = Ψ2σ, then it can be expressed as UNIFY(Ψ1, Ψ2)

Resolution

Aim: To implement a resolution algorithm.

Algorithm: ● Conversion of Facts into FOL - In the first step, we will convert all the given statements into their first-order logic. ● Conversion of FOL into CNF - In First-order logic resolution, it is required to convert the FOL into CNF as CNF form makes it easier for resolution proofs. ● Negate the statement to be proved - In this statement, we will apply negation to the conclusion statements, which will be written as ¬likes(John, Peanuts) ● Draw Resolution graph - Now in this step, we will solve the problem by resolution tree using substitution.

Dampster Shafer

Aim: To implement uncertain methods for Dempster Shafer's Theory.

Algorithm: ● Consider all possible outcomes. ● Belief will lead to believe in some possibility by bringing out some evidence. ● Plausibility will make evidence compatible with possible outcomes. ● Set of possible conclusion (P): {p1, p2….pn} where P is a set of possible conclusions and cannot be exhaustive, i.e. at least one (p)i must be true. ● (p)i must be mutually exclusive. ● Power Set will contain 2 n elements where n is the number of elements in the possible set.