BFS

Algorithm:

1. Assign ‘a’ as the root node and insert it into the Queue.

2. Extract node ‘a’ from the queue and insert the child nodes of ‘a’, i.e., ‘b’ and ‘c’.

3. Print node ‘a’.

4. The queue is not empty and has node ‘b’ and ‘c’. Since ‘b’ is the first node in the queue, let’s extract it and insert the child nodes of ‘b’, i.e., node ‘d’ and ‘e’.

5. Repeat these steps until the queue gets empty. Note that the nodes that are already visited should not be added to the queue again.

IDDFS

Algorithm:

1. Consider making a BFS into an Iterative depending search.

2. We can do this by having aside a DFS which will search up to a limit.It first does searching to a pre-defined limit depth to depth and then generates a route length 1.

3. This is done by creating routes by length 1 in the DFS way.Next it makes way for routes of depth limit 2,3 and onwards.

4. It even can delete all the preceding calculation all time at the beginning of the loop and iterate.Hence at some depth eventually the solution will be found if there is any in the tree because the enumeration takes place in order.

A\* Search

Algorithm:

1. Place the starting node in the OPEN list.

2. Check if the OPEN list is empty or not.If the list is empty the return failure and stop.

3. Select the node from the OPEN list which has the smallest value of evaluation function (g+n), if node n is goal node then return success and stop otherwise.

4. Expand node n and generate all of its successors and put n into the closed list. Check wheather ‘n’ is already present in OPEN or CLOSED list.

5. 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) values.

6. Return to ‘step 2’.

RBFS:

Algorithm

1. If n is goal.
2. Solution —- n ,exit ().
3. C——- expand (n).
4. If C is empty,return .∞
5. For each child n,in C.
6. If f(n)<f(n) then f(ni)——-max [f(n),f(ni)].
7. Else f(ni)——f(ni).
8. (n1,n2)——best f(c)
9. While [f(n1) <=B and f(n1)<∞]
10. F(n1)——-RBFS [n1,min (B,F(n2))]
11. (n1,n2)———best f(c)
12. Return f(n1)

Decision Tree:

Algorithm

1. Begin the tree with the root node, says S which contains the complete dataset.
2. Find the best attribute in the database using attribute selection measure (ASM).
3. Divide the S into subsets that contains possible values for the best attributes.
4. Generate the decision tree node which contain the best attributes.
5. Recursively make new decision trees using the subsets of the dataset created in step3 continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.

Feed forward back propagaton neural network:

Algorithm

1. Set the initial of weight.set a smaller random non zero number as coefficient of each layer, but set a coefficient initially.
2. Enter a sample as well as the corresponding desired output.
3. Calculate output at all layers.
4. Find the learning error for all layers.
5. Modify weights and threshold.
6. When the weights of different layers are calculated, the quality indicators can be set to determine whether the requirements are met or not. If the requirement are met the algorithm ends; or return to step3.The learning process for any given sample and the desired output should be implemented until all the input and output requirement are met.

Adaboost ensemble learning:

Algorithm

1. Initialise the dataset and assign equal weight to each of the data point.

2. Provide this as input to the model and identify the wrongly classified data points.

3. Increase the weight of the wrongly classified data points.

4. If (got required result)

Go to step 5

Else

Go to step 2

5. End