**Practical No: 01**

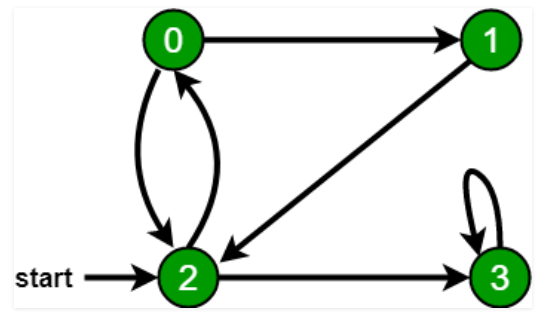
**Aim:** Implement Breadth first search algorithm for Romanian map problem.

**Theory:** Breadth First Search (BFS) Algorithm

Breadth first search is a graph traversal algorithm that starts traversing the graph from root node and explores all the neighboring nodes. Then, it selects the nearest node and explore all the unexplored nodes. The algorithm follows the same process for each of the nearest node until it finds the goal.

The algorithm of breadth first search is given below. The algorithm starts with examining the node A and all of its neighbors. In the next step, the neighbors of the nearest node of A are explored and the process continues in the further steps. The algorithm explores all neighbors of all the nodes and ensures that each node is visited exactly once and no node is visited twice.

For example, in the following graph, we start traversal from vertex 2. When we come to vertex 0, we look for all adjacent vertices of it. 2 is also an adjacent vertex of 0. If we don’t mark visited vertices, then 2 will be processed again and it will become a non-terminating process. A Breadth-First Traversal of the following graph is 2, 0, 3, 1.



**Time Complexity:** O(V+E) where V is a number of vertices in the graph and E is a number of edges in the graph.

**Advantages of BFS:**

1. The solution will definitely be found by BFS If there is some solution.

2. BFS will never get trapped in a blind alley, which means unwanted nodes.

3. If there is more than one solution then it will find a solution with minimal steps.

**Disadvantages Of BFS:**

1. Memory Constraints As it stores all the nodes of the present level to go for the next level.

2. If a solution is far away then it consumes time.

**Practical No: 02**

**Aim:** Implement Iterative deep depth first search for Romanian map problem.

**Theory:** Depth-first search is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. So the basic idea is to start from the root or any arbitrary node and mark the node and move to the adjacent unmarked node and continue this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path.

**Algorithm:**

Create a recursive function that takes the index of the node and a visited array.

1. Mark the current node as visited and print the node.
2. Traverse all the adjacent and unmarked nodes and call the recursive function with the index of the adjacent node.

**Time complexity:** O(V + E), where V is the number of vertices and E is the number of edges in the graph.

**Space Complexity:** O(V), since an extra visited array of size V is required.

**Example:**

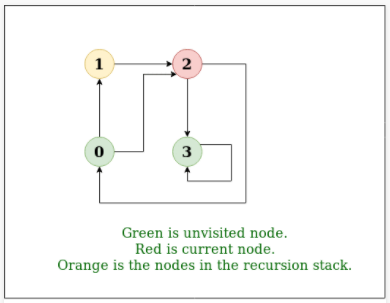
**Input:** n = 4, e = 6

0 -> 1, 0 -> 2, 1 -> 2, 2 -> 0, 2 -> 3, 3 -> 3

**Output:** DFS from vertex 1 : 1 2 0 3

**Explanation:**

**DFS Diagram:**

****

**Practical No: 03**

**Aim:** A\* (star) using romanian map problem.

**Theory:**

**What is A\* Search Algorithm?**

A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

**Why A\* Search Algorithm?**

Informally speaking, A\* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms.

This fact is cleared in detail in below sections. And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

A\* Search Algorithm

1. Initialize the open list.
2. Initialize the closed list put the starting node on the open list (you can leave its f at zero)
3. while the open list is not empty
   1. find the node with the least f on the open list, call it "q"
   2. pop q off the open list
   3. generate q's 8 successors and set their parents to q
   4. for each successor
      1. if successor is the goal, stop search
      2. else, compute both g and h for successor successor.g = q.g + distance between successor and q successor.h = distance from goal to successor (This can be done using many ways, we will discuss three heuristics- Manhattan, Diagonal and Euclidean Heuristics)

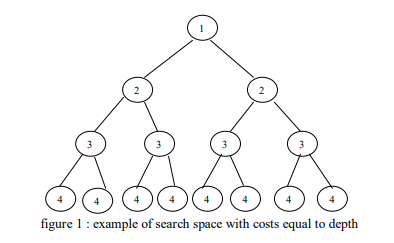
successor.f = successor.g + successor.h

* + 1. if a node with the same position as successor is in the OPEN list which has a lower f than successor, skip this successor
    2. if a node with the same position as successor is in the CLOSED list which has a lower f than successor, skip this successor otherwise, add the node to the open list end (for loop)
  1. push q on the closed list end (while loop)

**Practical No: 04**

Aim: Implement recursive best-first search algorithm for Romanian map problem

Recursive Best-First Search or RBFS, is an Artificial Intelligence Algorithmthat belongs to heuristic search algorithm. It expands fronteir nodes in best-first order. It uses the problem specific information about the enviroment to determine the preference of one node over the other. RBFS is similar to a recursive implementation of depth-first search, with the difference that it uses a special condition for backtracking that ensures that nodes are expanded in best-first order.It works by maintaing on the recursion stack the complete path to the current node being expanded, as well as all immediate siblings of nodes on that path, along with cost of the best node in the subtree explored below each sibling [4]. Whenever the cost of the current node exceeds that of some other node in the previously expanded portion of the tree, the algorithm backs up to their deepest common ancestor, and continues the search down the new path [2]. RBFS explores the search space by considering it as a tree. An example of the search space with cost equal to depth is shown in Fig. 1. RBFS explores the search space by considering it as a tree.



**Algorithm**

*{*

*function RECURSIVE\_BEST\_FIRST\_SEARCH (Problem)*

*returns a solution, or failure*

*RBFS (problem, MAKE\_NODE (INITIAL\_STATE problem), Ұ)*

*function RBFS (Problem, Node, f-limit) returns a solution or failure and a new f-cost limit*

*if GOAL\_TEST [Problem] [State] then return node*

*Successors ← EXPAND (node, problem)*

*if successors is empty then return failure, ¥*

*for each S in successors do*

*f(s) ← max (g(s) + h(s), f[node])*

*repeat*

*best ← the lowest f value node in successors*

*if [best] > f-limit then return failure, f[best]*

*alternative ← the second lowest f-value among successors*

*result, f[best] ← BFS (Problem, best, min (f-limit, alternative)*

*if result # failure then return result*

*}*

**Advantages**

* More efficient than IDA\*
* It is an optimal algorithm if h(n) is admissible
* Space complexity is O(bd).

**Disadvantages**

* It suffers from excessive node regeneration.
* Its time complexity is difficult to characterize because it depends on the accuracy of h(n) and how often the best path changes as the nodes are expanded.

**Practical No: 05**

**Aim:** Implement decision tree learning algorithm for the restaurant waiting problem.

**Theory:**

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.

The decisions or the test are performed on the basis of features of the given dataset.

It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

## **Decision Tree Terminologies**

|  |
| --- |
| * **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets. * **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node. * **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions. * **Branch/Sub Tree:** A tree formed by splitting the tree. * **Pruning:** Pruning is the process of removing the unwanted branches from the tree. * **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.   **How does the Decision Tree algorithm Work?** |

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

## **Advantages of the Decision Tree**

* It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
* It can be very useful for solving decision-related problems.
* It helps to think about all the possible outcomes for a problem.
* There is less requirement of data cleaning compared to other algorithms.

## **Disadvantages of the Decision Tree**

* The decision tree contains lots of layers, which makes it complex.
* It may have an overfitting issue, which can be resolved using the **Random Forest algorithm.**
* For more class labels, the computational complexity of the decision tree may increase.

**Practical\_no:06**

**Aim: Implement feed forward back propagation neural network learning algorithm for the restaurant waiting problem .**

**Theory:**

A feedforward neural network is an artificial neural network where the nodes never form a cycle. This kind of neural network has an input layer, hidden layers, and an output layer. It is the first and simplest type of artificial neural network. Two Types of Backpropagation Networks are: • Static Back-propagation : It is one kind of backpropagation network which produces a mapping of a static input for static output. It is useful to solve static classification issues like optical character recognition. • Recurrent Backpropagation : Recurrent Back propagation in data mining is fed forward until a fixed value is achieved. After that, the error is computed and propagated backward.

A feedforward neural network is an artificial neural network where the nodes never form a cycle. This kind of neural network has an input layer, hidden layers, and an output layer. It is the first and simplest type of artificial neural network.

Practical No 7

Aim:ANN

Theory:

Artificial neural networks (ANNs), usually simply called neural networks (NNs), are computing systems inspired by the biological neural networks that constitute animal brains. An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives a signal then processes it and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last layer (the output layer), possibly after traversing the layers multiple times.

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of [machine learning](https://www.ibm.com/cloud/learn/machine-learning) and are at the heart of [deep learning](https://www.ibm.com/cloud/learn/deep-learning) algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

**Practical No: 08**

**Aim:** Implement Naive Bayes’ learning algorithm for the restaurant waiting problem.

**Theory:**

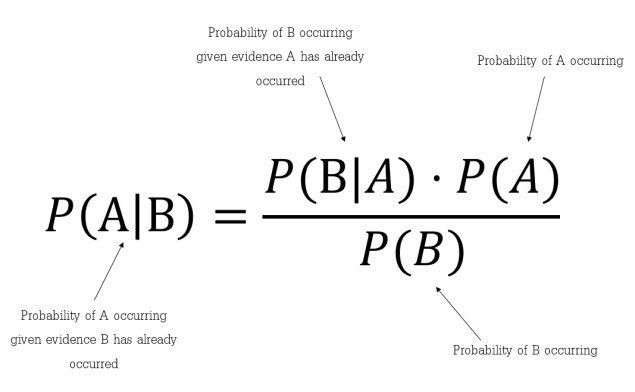
Naïve: It is called Naïve because it assumes that the occurrence of a certain feature is independent of the occurrence of other features. Such as if the fruit is identified on the basis of color, shape, and taste, then red, spherical, and sweet fruit is recognized as an apple. Hence each feature individually contributes to identify that it is an apple without depending on each other.

Bayes: It is called Bayes because it depends on the principle of Bayes' Theorem Bayes' theorem is also known as Bayes' Rule or Bayes' law, which is used to determine the probability of a hypothesis with prior knowledge. It depends on the conditional probability.

**Bayes Theorem**

Bayes’ Theorem is a simple mathematical formula used for calculating conditional probabilities.

Conditional probability is a measure of the probability of an event occurring given that another event has (by assumption, presumption, assertion, or evidence) occurred.



### **Assumptions made by Naïve Bayes**

The fundamental Naïve Bayes assumption is that each feature makes an:

* Independent: We assume that no pair of features are dependent. Hence, the features are assumed to be Independent.
* Equal: Secondly, each feature is given the same influence(or importance). So none of the attributes are irrelevant and assumed to be contributing Equally to the outcome

## **Naive Bayes algorithm**

* Step 1: Convert the data set into a frequency table
* Step 2: Create Likelihood table by finding the probabilities
* Step 3: Now, use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

**Pros:**

* It is easy and fast to predict the class of a test data set. It also perform well in multi class prediction
* When assumption of independence holds, a Naive Bayes classifier performs better compared to other models like logistic regression and you need less training data.
* It performs well in case of categorical input variables compared to numerical variable(s). For numerical variables, normal distribution is assumed (bell curve, which is a strong assumption).

**Cons:**

* If a categorical variable has a category (in test data set), which was not observed in the training data set, then the model will assign a 0 (zero) probability and will be unable to make a prediction. This is often known as “Zero Frequency”. To solve this, we can use the smoothing technique. One of the simplest smoothing techniques is called Laplace estimation.
* On the other side naive Bayes is also known as a bad estimator, so the probability outputs from predict\_proba are not to be taken too seriously.
* Another limitation of Naive Bayes is the assumption of independent predictors. In real life, it is almost impossible that we get a set of predictors which are completely independent.