CHAITANYA ENGINEERING COLLEGE

KOMMADI, VISAKHAPATNAM-530048.

This is to certify that____________________________is a student studying__________________________ Register No __________________ branch ________________________ has done ______________ number of experiments during the year _____________ in the subject ____________________________.

Lecture in-charge   Head Of the Department

External Examiner

EXPERIMENT - 1

Implement and demonstrate FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples. Read the training data from a .csv file.

Aim: To demonstrate the working of FIND-S algorithm for finding the most specific hypothesis based on training data.

Description: The Find-S algorithm is a machine learning algorithm used for concept learning in the context of supervised learning. It is part of the concept learning systems and is often employed to find the most specific hypothesis from a given set of training data.
Initialization: Start with the most specific hypothesis. In the case of attribute-value pairs, this would be a hypothesis where all attributes are set to the most specific value (e.g., '?' or 'null').

Iterative Refinement: Go through each training example one by one. If the example is positive, update the hypothesis to include only the attributes that match the positive example while keeping the rest as general as possible.
For example, if you have a positive example like (Sunny, Warm, Normal, Strong, Warm, Same), and the current hypothesis is (?, ?, ?, ?, ?, ?), after processing this positive example, the hypothesis would be updated to (Sunny, Warm, Normal, Strong, Warm, Same).

Output: The final hypothesis is the most specific hypothesis consistent with the training data.

Algorithm:

1. Initialize h to the most specific hypothesis in H

2. For each positive training instance x

```
  a. For each attribute constraint a_i in h
    i. If the constraint a_i is satisfied by x, do nothing
    ii. If the constraint a_i is not satisfied by x, replace it with the next more general constraint that is satisfied by x
3. Output the hypothesis h
```

Dataset:

sky	airTemp	humidity	wind	water	forecast	enjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import pandas as pd
import numpy as np
data= pd.read_csv("ENJOYSPORT.csv")
print(data)
a = np.array(data)[:,:-1]
print(" The attributes are: ",a)
target = np.array(data)[:,-1]
print("The target is: ",target)
def train(c,t):
 for i, val in enumerate(t):
  if val == "Yes":
    specific_hypothesis = c[i].copy()
    break
 for i, val in enumerate(c):
  if t[i] == "Yes":
    for x in range(len(specific_hypothesis)):
      if val[x] != specific_hypothesis[x]:
        specific_hypothesis[x] = '?'
      else:
        pass
    return specific_hypothesis
print(" The final hypothesis is:",train(a,target))
```

Output:

```
   Sky AirTemp Humidity    Wind Water Forecast  EnjoySport
0  Sunny    Warm   Normal  Strong  Warm     Same           1
1  Sunny    Warm     High  Strong  Warm     Same           1
2  Rainy    Cold     High  Strong  Warm   Change           0
3  Sunny    Warm     High  Strong  Cool   Change           1
 The attributes are:  [['Sunny' 'Warm' 'Normal' 'Strong' 'Warm'
'Same']
 ['Sunny' 'Warm' 'High' 'Strong' 'Warm' 'Same']
 ['Rainy' 'Cold' 'High' 'Strong' 'Warm' 'Change']
 ['Sunny' 'Warm' 'High' 'Strong' 'Cool' 'Change']]
The target is:  ['no' , 'yes', 'no', 'yes']
The Final Hypothesis is : ['Sunny', 'Warm', 'High', 'Strong',
'?', '?']
```

EXPERIMENT-2

For a given set of training data examples stored in a .csv file, implement and demonstrate the candidate elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

Aim: To demonstrate the working of candidate elimination algorithm on given training examples.

Description: The Candidate Elimination Algorithm is a concept learning algorithm used in machine learning. It was introduced by E. Mark Gold in 1967. The algorithm is designed to learn a hypothesis from a given set of training examples, where each example is a tuple of attribute values and a binary classification (positive or negative). The goal is to learn a hypothesis that generalizes well to classify new, unseen examples.
Initialization:

1. The algorithm starts with two hypotheses: the most specific hypothesis (S) and the most general hypothesis (G).
2. The specific hypothesis (S) is initialized with the most specific values, usually represented by ['0', '0', ..., '0'].
3. The general hypothesis (G) is initialized with the most general values, usually represented by ['?', '?', ..., '?'].

Iterative Refinement:

1. The algorithm iterates through each training example.
2. For a positive example, it refines the hypotheses:
    a. For each attribute where the specific hypothesis is currently more general than the example, update the specific hypothesis to include the specific attribute value.
    b. For each attribute where the general hypothesis is currently more specific than the example, update the general hypothesis to make it more general.
3. For a negative example, it refines the hypotheses:
    a. For each attribute where the specific hypothesis is inconsistent with the example, make it more general in the specific hypothesis.

b. If the general hypothesis is inconsistent with the example, keep it unchanged.

Output:

After processing all examples, the final hypotheses are considered. The specific hypothesis (S) is the most specific hypothesis consistent with the positive training examples, and the general hypothesis (G) is the most general hypothesis consistent with the negative training examples.

Dataset:

sky	airTemp	humidity	wind	water	forecast	enjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Program:

```
import numpy as np
import pandas as pd
data = pd.DataFrame(data=pd.read_csv('ENJOYSPORT.csv'))
concepts = np.array(data.iloc[:,0:-1])
target = np.array(data.iloc[:,-1])
def learn(concepts, target):
 specific_h=[0,0,0,0,0,0,0]
 print ('s0',specific_h)
 specific_h = concepts[0].copy()
 print('s1',specific_h)
 general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
 print('g0',general_h)
 for i, h in enumerate(concepts):
  if target[i] == "Yes":
    for x in range(len(specific_h)):
      if h[x] != specific_h[x]:
        specific_h[x] = '?'
        general_h[x][x] = '?'
        print(f"s{x}",specific_h)
        print(f"g{x}",general_h)
  if target[i] == "No":
    for x in range(len(specific_h)):
      if h[x] != specific_h[x]:
        general_h[x][x] = specific_h[x]
      else:
        general_h[x][x] = '?'
      indices = [i for i,val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?', '?']]
      for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?', '?'])
        print('i',indices)
  return specific_h,general_h
s_final, g_final = learn(concepts, target)
```

Output:

```
s0 [0, 0, 0, 0, 0, 0, 0]
s1 ['Sunny' 'Warm' 'Normal' 'Strong' 'Warm' 'Same']
g0 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
s2 ['Sunny' 'Warm' '?' 'Strong' 'Warm' 'Same']
g2 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s4 ['Sunny' 'Warm' '?' 'Strong' '?' 'Same']
g4 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]
s5 ['Sunny' 'Warm' '?' 'Strong' '?' '?']
g5 [['?', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

EXPERIMENT-3

Write a program to demonstrate the working of the decision tree classifier. Use appropriate dataset for building the decision tree and apply this knowledge to classify a new sample.

Aim: To demonstrate the working of decision tree classifier to classify a new sample.

Description: ID3 algorithm is a basic algorithm that learns decision trees by constructing them topdown, beginning with the question "which attribute should be tested at the root of the tree?".To answer this question, each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree. A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node (i.e., down the branch corresponding to the example's value for this attribute). The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.

Algorithm:

```
1. Create a Root node for the tree
2. If all Examples are positive, Return the single-node tree Root, with label = +
3. If all Examples are negative, Return the single-node tree Root, with label = -
4. If Attributes is empty,
    Return the single-node tree Root, with label = most common value of
    TargetAttribute in Examples
    Else
    A ← the attribute from Attributes that best classifies Examples The decision
    attribute for Root ←A
    For each possible value, vi, of A,
        Add a new tree branch below Root, corresponding to the test A = vi
    Let Examplesvi be the subset of Examples that have value vi for A
    If Examples vi is empty
```

```
            Then below this new branch add a leaf node with label = most
        common value of TargetAttribute in Examples
         Else below this new branch add the subtree ID3(Examplesvi,
        TargetAttribute, Attributes–{A}) End
 5. Return Root
```

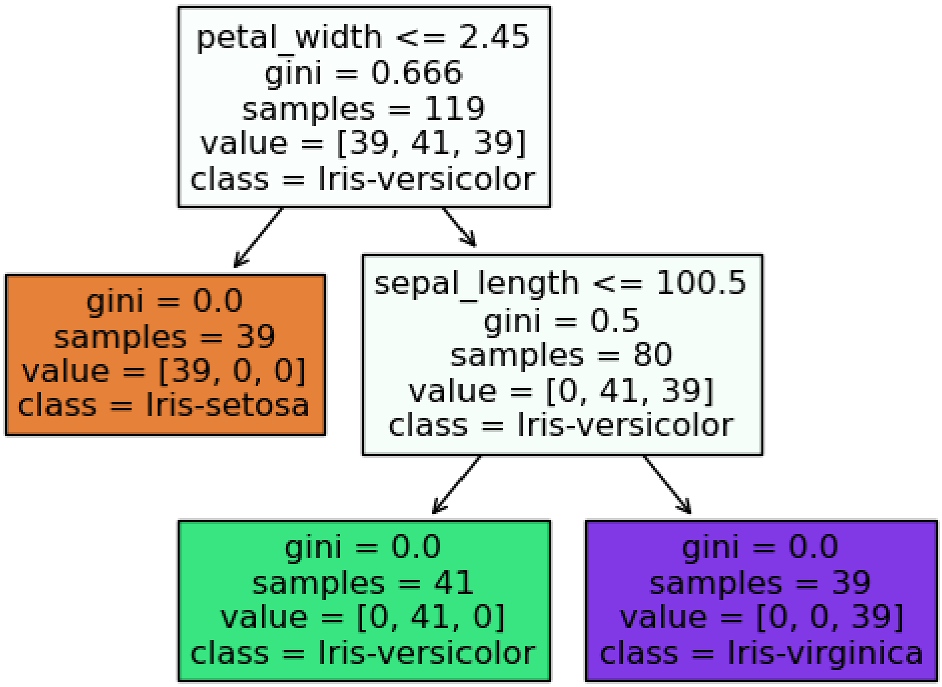
Dataset:
It contains measurements of four features (sepal length, sepal width, petal length, and petal width) for 150 iris flowers from three different species: setosa, versicolor, and virginica. Each species consists of 50 samples.

Program:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.preprocessing import LabelEncoder
# Load the Iris dataset
data = pd.read_csv('Iris.csv', skiprows=1)  # Skip the header row
# Manually specify feature names
feature_names = ["sepal_length", "sepal_width", "petal_length", "petal_width",
"class"]
# Separate features and target variable
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
# Encode the categorical labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
# Split the dataset into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
# Train the decision tree classifier
classifier = DecisionTreeClassifier()
classifier.fit(X_train, y_train)
# Plot the decision tree
```

```
plot_tree(classifier, feature_names=feature_names,
class_names=label_encoder.classes_, filled=True)
# Perform classification on a new sample
new_sample = [[5.1, 3.5, 1.4, 0.2, 0]]  # Example new sample with a class label
# Print the decision tree rules
def print_decision_rules(tree, feature_names, node=0):
    if tree.feature[node] != -2:
        feature_index = tree.feature[node]
        threshold = tree.threshold[node]
        class_index = tree.value[node].argmax()
        class_label = label_encoder.inverse_transform([class_index])[0]
        rule = f"If {feature_names[feature_index]} <= {threshold} then
{class_label}"
        print(rule)
        print_decision_rules(tree, feature_names, node=tree.children_left[node])
        print_decision_rules(tree, feature_names, node=tree.children_right[node])
print_decision_rules(classifier.tree_, feature_names[:-1])
```

Output:

```
If petal_width <= 2.449999988079071 then Iris-versicolor
If sepal_length <= 100.5 then Iris-versicolor
```


petal_width <= 2.45
gini = 0.666
samples = 119
value = [39, 41, 39]
class = Iris-versicolor
gini = 0.0
samples = 39
value = [39, 0, 0]
class = Iris-setosa
sepal_length <= 100.5
gini = 0.5
samples = 80
value = [0, 41, 39]
class = Iris-versicolor
gini = 0.0
samples = 41
value = [0, 41, 0]
class = Iris-versicolor
gini = 0.0
samples = 39
value = [0, 0, 39]
class = Iris-virginica

EXPERIMENT-4

Write a program to demonstrate the working of Decision tree regressor. Use appropriate dataset for decision tree regressor.

Aim: To demonstrate the working of decision tree regressor using salary dataset.
Description: A Decision Tree Regressor is a supervised machine learning algorithm used for predicting a continuous target variable. Unlike the Decision Tree Classifier, which is used for classification tasks, the Decision Tree Regressor predicts a numerical outcome based on the input features. Here's a theoretical overview of the Decision Tree Regressor:
Decision Trees Overview:

1. Decision Trees are hierarchical tree-like structures composed of nodes.
2. Nodes in the tree represent decisions based on features, and the leaves represent the predicted continuous values.

Splitting Criteria:

3. Decision Trees make decisions by recursively splitting the dataset into subsets based on the values of input features.
4. The algorithm chooses the best feature to split on at each node. Common splitting criteria include mean squared error reduction.

Recursive Splitting:

> The recursive process continues until a stopping criterion is met. This could be a maximum depth, a minimum number of samples per leaf, or other criteria.

Leaf Nodes:

> The leaf nodes of the tree contain the predicted continuous values for the instances that end up in that leaf.

Prediction:

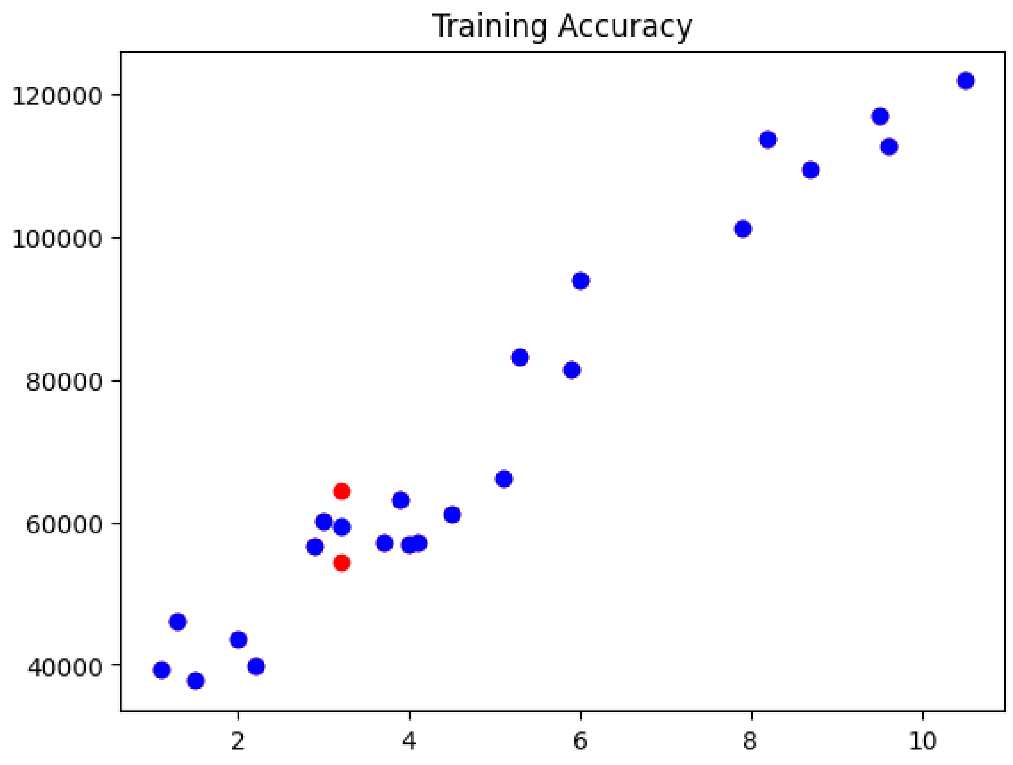
To predict the target variable for a new instance, you traverse the tree from the root to a leaf, following the decision rules at each node, and use the value stored in the leaf.

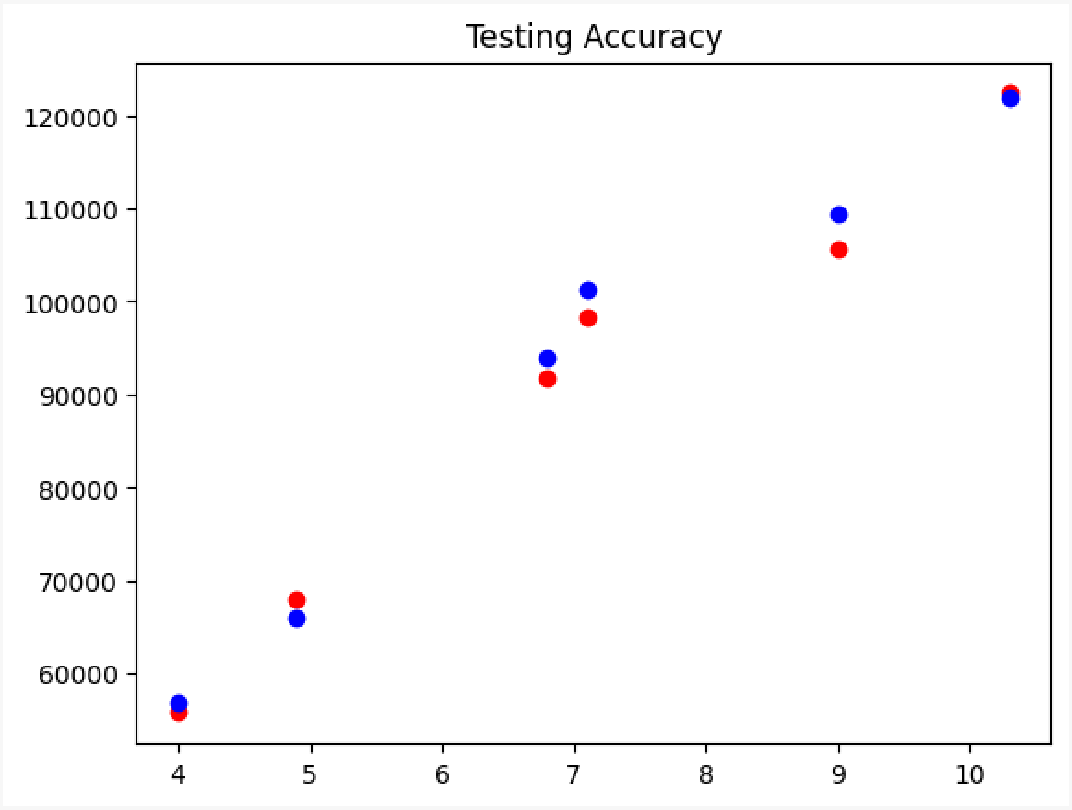
Dataset:
The dataset has 2 columns namely years of experience and salary. There are 30 instances in the dataset.

Program:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.model_selection import GridSearchCV

from sklearn.tree import DecisionTreeRegressor
df = pd.read_csv('salary.csv')
X = df.iloc[:, :-1]
y = df.iloc[:, -1]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2,
random_state = 4)
dt_regressor = DecisionTreeRegressor(random_state = 0)
dt_regressor.fit(X_train, y_train)
y_pred_train = dt_regressor.predict(X_train)
r2_score(y_train, y_pred_train)
fig, ax = plt.subplots()
plt.title('Training Accuracy')
ax.scatter(X_train,y_train, color = "red")
ax.scatter(X_train,y_pred_train, color = "blue")
y_pred_train = dt_regressor.predict(X_train)
r2_score(y_train, y_pred_train)
fig, ax = plt.subplots()
plt.title('Testing Accuracy')
ax.scatter(X_test,y_test, color = "red")
ax.scatter(X_test,y_pred, color = "blue")
y_pred = dt_regressor.predict(X_test)
r2_score(y_test, y_pred)
```

Output:

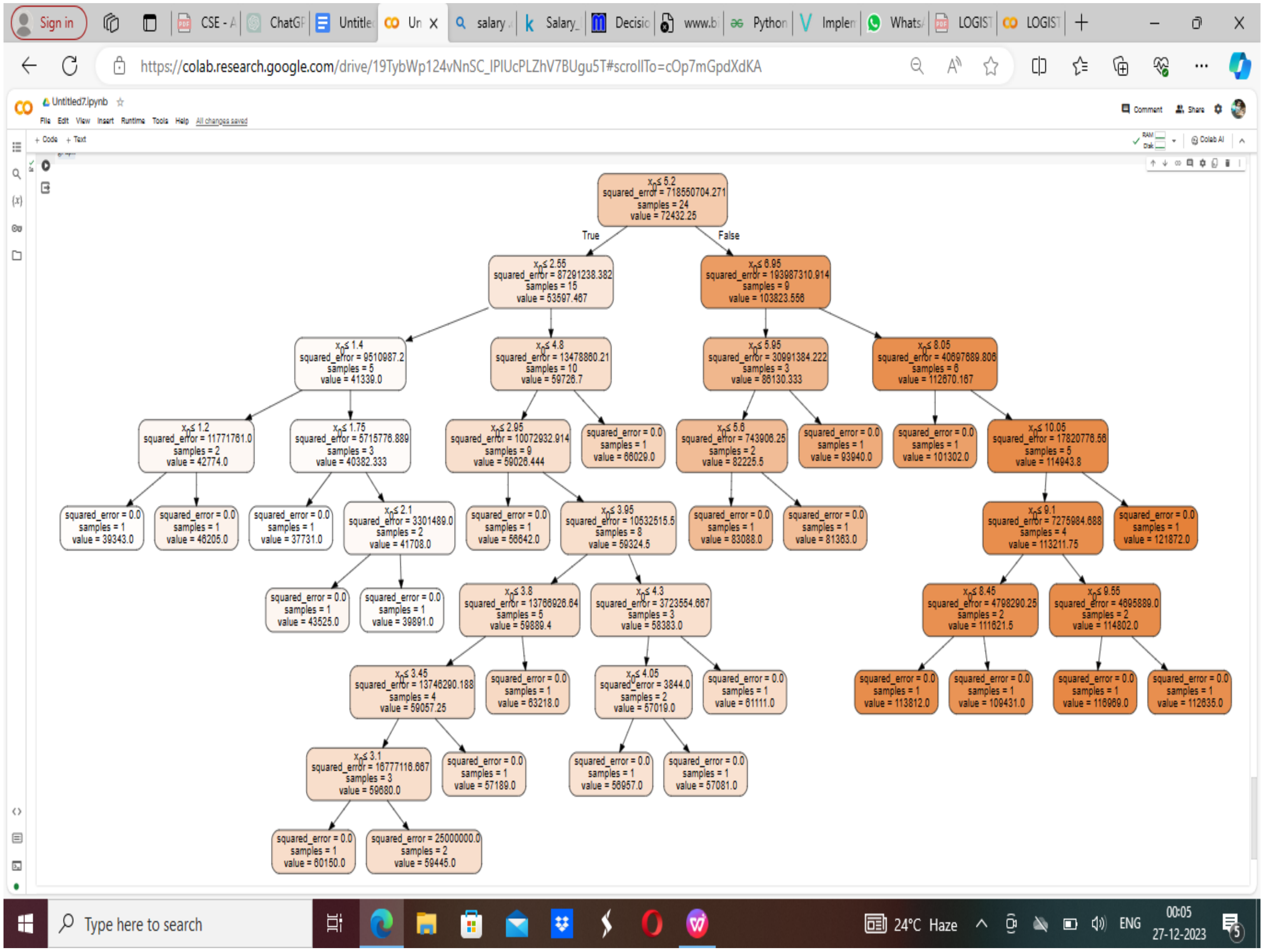
```
0.9887060585499007
```





```
from sklearn import tree
import graphviz
dot_data = tree.export_graphviz(dt_regressor,out_file=None,
                    filled=True, rounded=True,
                    special_characters=True)
graph = graphviz.Source(dot_data)
graph
```

Output:


Sign in
CSE - A
ChatGP
Untitle
Un
salary
Salary_
Decisio
www.b
Python
Implen
Whats
LOGIST
LOGIST
https://colab.research.google.com/drive/19TybWp124vNnSC_IPIUcPLZhV7BUgu5T#scrollTo=cOp7mGpdXdKA
Untitled7.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Comment Share
RAM Disk
Colab AI
$x_0 \le 5.2$ squared_error = 718550704.271 samples = 24 value = 72432.25
True
False
$x_0 \le 2.55$ squared_error = 87291238.382 samples = 15 value = 53597.467
$x_0 \le 6.95$ squared_error = 193987310.914 samples = 9 value = 103823.556
$x_0 \le 1.4$ squared_error = 9510987.2 samples = 5 value = 41339.0
$x_0 \le 4.8$ squared_error = 13478860.21 samples = 10 value = 59726.7
$x_0 \le 5.95$ squared_error = 30991384.222 samples = 3 value = 86130.333
$x_0 \le 8.05$ squared_error = 40697689.806 samples = 6 value = 112670.167
$x_0 \le 1.2$ squared_error = 11771761.0 samples = 2 value = 42774.0
$x_0 \le 1.75$ squared_error = 5715776.889 samples = 3 value = 40382.333
$x_0 \le 2.95$ squared_error = 10072932.914 samples = 9 value = 59026.444
squared_error = 0.0 samples = 1 value = 66029.0
$x_0 \le 5.6$ squared_error = 743906.25 samples = 2 value = 82225.5
squared_error = 0.0 samples = 1 value = 93940.0
squared_error = 0.0 samples = 1 value = 101302.0
$x_0 \le 10.05$ squared_error = 17820776.56 samples = 5 value = 114943.8
squared_error = 0.0 samples = 1 value = 39343.0
squared_error = 0.0 samples = 1 value = 46205.0
squared_error = 0.0 samples = 1 value = 37731.0
$x_0 \le 2.1$ squared_error = 3301489.0 samples = 2 value = 41708.0
squared_error = 0.0 samples = 1 value = 56642.0
$x_0 \le 3.95$ squared_error = 10532515.5 samples = 8 value = 59324.5
squared_error = 0.0 samples = 1 value = 83088.0
squared_error = 0.0 samples = 1 value = 81363.0
$x_0 \le 9.1$ squared_error = 7275984.688 samples = 4 value = 113211.75
squared_error = 0.0 samples = 1 value = 121872.0
squared_error = 0.0 samples = 1 value = 43525.0
squared_error = 0.0 samples = 1 value = 39891.0
$x_0 \le 3.8$ squared_error = 13766926.64 samples = 5 value = 59889.4
$x_0 \le 4.3$ squared_error = 3723554.667 samples = 3 value = 58383.0
$x_0 \le 8.45$ squared_error = 4798290.25 samples = 2 value = 111621.5
$x_0 \le 9.55$ squared_error = 4695889.0 samples = 2 value = 114802.0
$x_0 \le 3.45$ squared_error = 13746290.188 samples = 4 value = 59057.25
squared_error = 0.0 samples = 1 value = 63218.0
$x_0 \le 4.05$ squared_error = 3844.0 samples = 2 value = 57019.0
squared_error = 0.0 samples = 1 value = 61111.0
squared_error = 0.0 samples = 1 value = 113812.0
squared_error = 0.0 samples = 1 value = 109431.0
squared_error = 0.0 samples = 1 value = 116969.0
squared_error = 0.0 samples = 1 value = 112635.0
$x_0 \le 3.1$ squared_error = 16777116.667 samples = 3 value = 59680.0
squared_error = 0.0 samples = 1 value = 57189.0
squared_error = 0.0 samples = 1 value = 56957.0
squared_error = 0.0 samples = 1 value = 57081.0
squared_error = 0.0 samples = 1 value = 60150.0
squared_error = 25000000.0 samples = 2 value = 59445.0
Type here to search
24°C Haze
ENG
00:05
27-12-2023

EXPERIMENT-5

Write a program to demonstrate the working of Random Forest classifier. Use appropriate dataset for Random Forest Classifier.

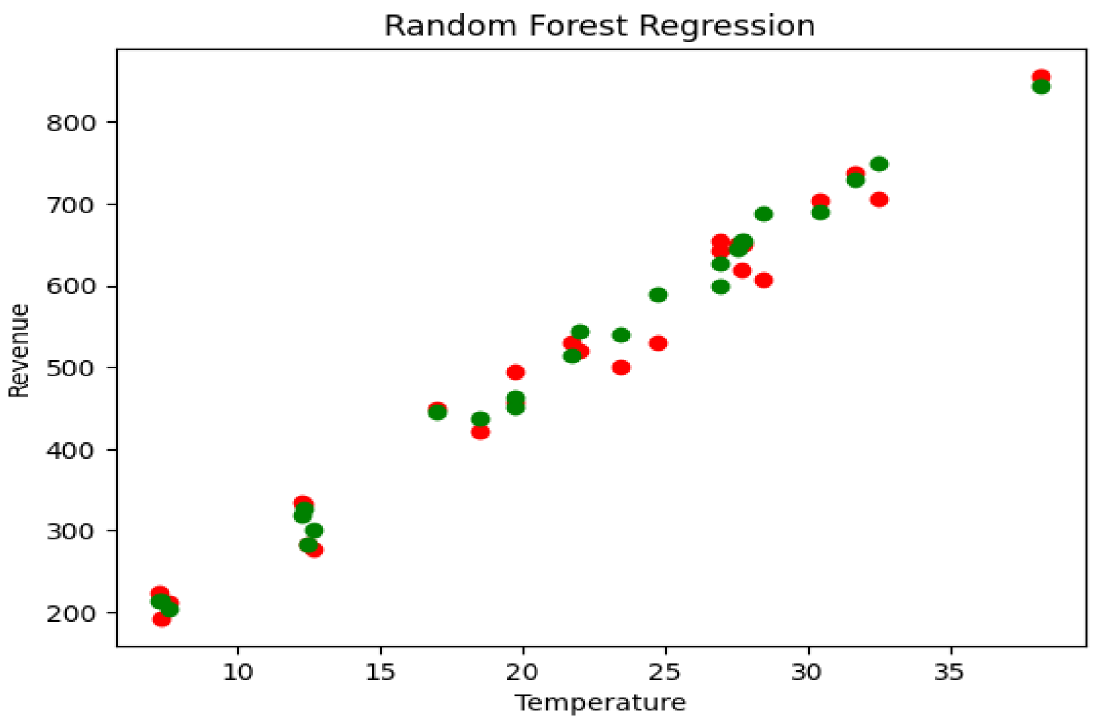
Aim: To demonstrate the working of Random Forest Classifier .
Description: Random Forest is also a “Tree”-based algorithm that uses the qualities features of multiple Decision Trees for making decisions.Therefore, it can be referred to as a ‘Forest’ of trees and hence the name “Random Forest”. The term ‘Random’ is due to the fact that this algorithm is a forest of ‘Randomly created Decision Trees’. The Decision Tree algorithm has a major disadvantage in that it causes over- fitting. This problem can be limited by implementing the Random Forest Regression in place of the Decision Tree Regression. Additionally, the Random Forest algorithm is also very fast and robust than other regression models.

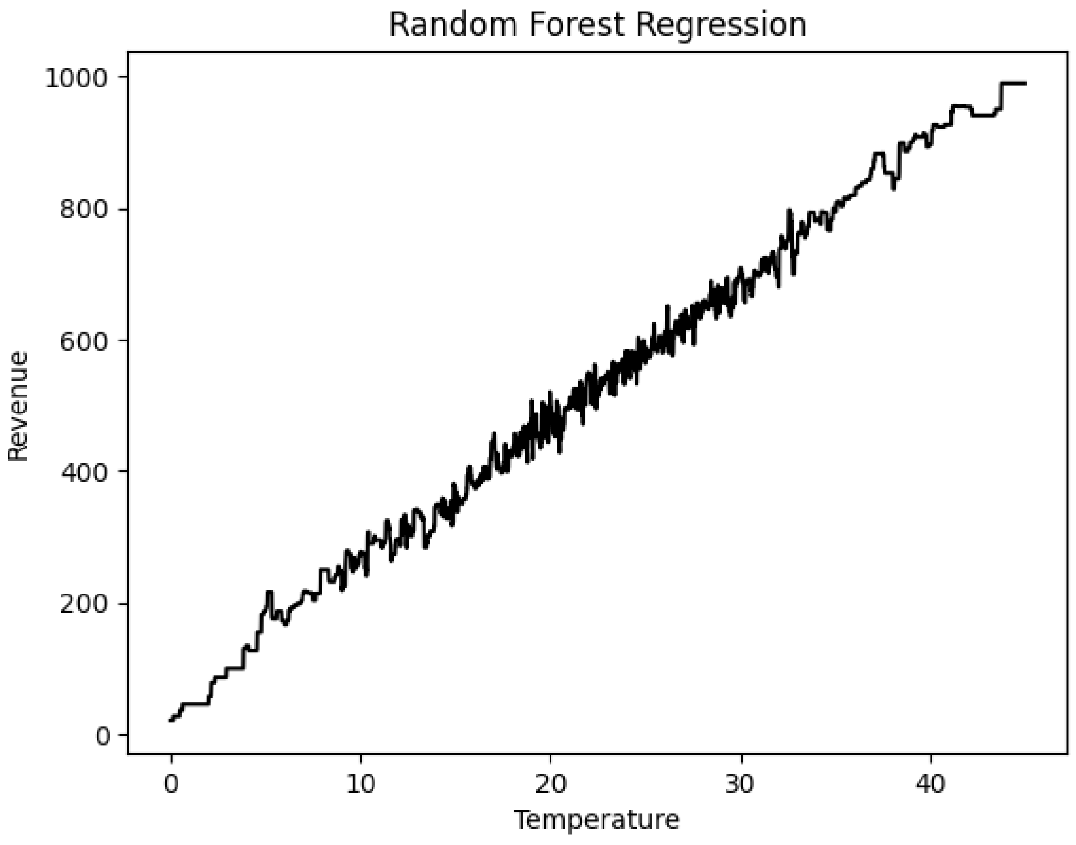
Program:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
dataset  =
pd.read_csv('https://raw.githubusercontent.com/mk-gurucharan/Regression/master/
IceCreamData.csv')
X = dataset['Temperature'].values
y = dataset['Revenue'].values
dataset.head(5)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.05)
# Fitting Random Forest Regression to the dataset
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators = 10, random_state = 0)
regressor.fit(X_train.reshape(-1,1), y_train.reshape(-1,1))
y_pred = regressor.predict(X_test.reshape(-1,1))
```

```
df = pd.DataFrame({'Real Values':y_test.reshape(-1), 'Predicted
Values':y_pred.reshape(-1)})
# # Visualizing the Random Forest Regression Results
X_grid = np.arange(min(X), max(X), 0.01)
X_grid = X_grid.reshape((len(X_grid), 1))
plt.scatter(X_test, y_test, color = 'red')
plt.scatter(X_test, y_pred, color = 'green')
plt.title('Random Forest Regression')
plt.xlabel('Temperature') plt.ylabel('Revenue')
plt.show()
plt.plot(X_grid, regressor.predict(X_grid), color = 'black') plt.title('Random Forest
Regression')
plt.xlabel('Temperature')
plt.ylabel('Revenue')
plt.show()
```

Output:


Random Forest Regression
Revenue
Temperature
200
300
400
500
600
700
800
10
15
20
25
30
35


Random Forest Regression
Revenue
Temperature
0
200
400
600
800
1000
0
10
20
30
40

EXPERIMENT-6

Write a program to demonstrate the working of Logistic Regression classifier. Use appropriate dataset for Logistic Regression.

Aim: To demonstrate the working of Logistic Regression Classifier on fish dataset.
Description: Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas Logistic regression is used for solving the classification problems

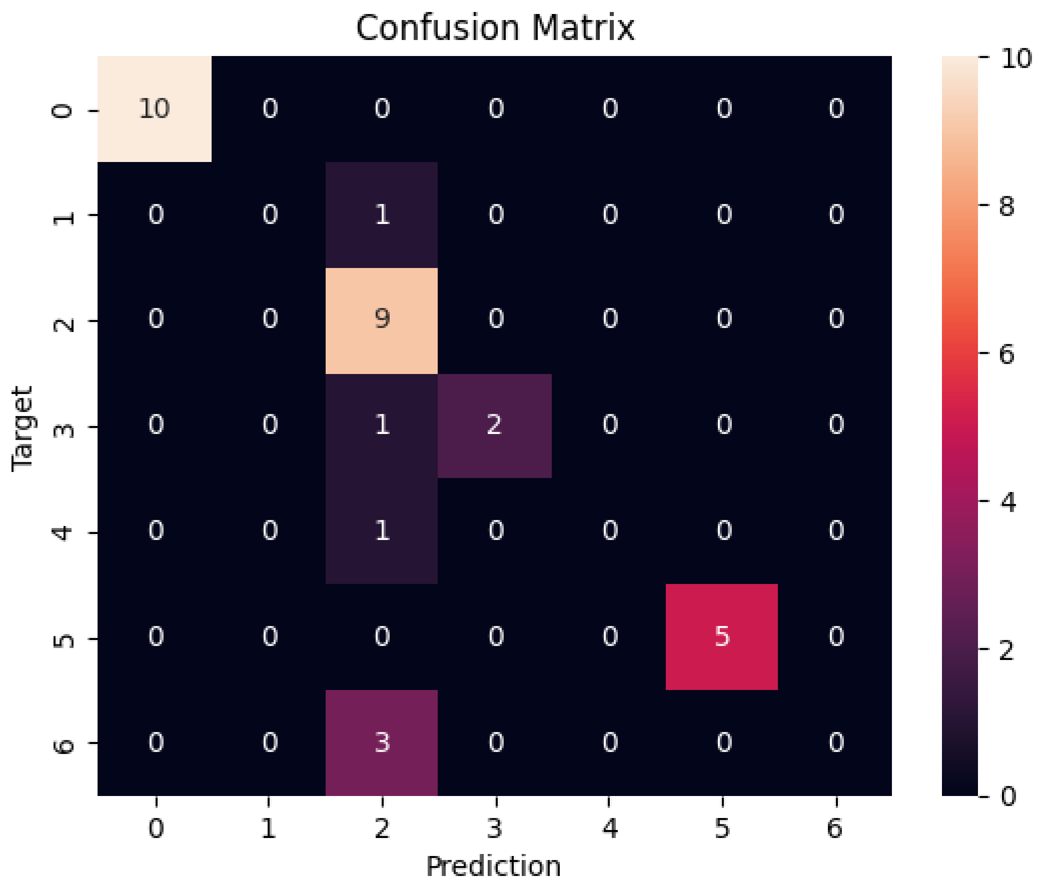
Program:

```
dataset_url="https://raw.githubusercontent.com/harika-bonthu/02-linear-regression-fish/master/datasets_229906_491820_Fish.csv"
import pandas as pd
fish = pd.read_csv(dataset_url, error_bad_lines=False)
fish.head()
X = fish.iloc[:, 1:]
y = fish.loc[:, 'Species']
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test= train_test_split(X_scaled, y, test_size=0.2,
random_state=42)
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
# training the model
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy * 100))
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred)
Import matplotlib.pyplot as plt
plt.figure()
Import seaborn as sns
sns.heatmap(cf, annot=True)
plt.xlabel('Prediction')
plt.ylabel('Target')
plt.title('Confusion Matrix')
```

Output:

```
Accuracy: 81.25%

Text(0.5, 1.0, 'Confusion Matrix')
```



Target \ Prediction	0	1	2	3	4	5	6
0	10	0	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	9	0	0	0	0
3	0	0	1	2	0	0	0
4	0	0	1	0	0	0	0
5	0	0	0	0	0	5	0
6	0	0	3	0	0	0	0

EXPERIMENT-7

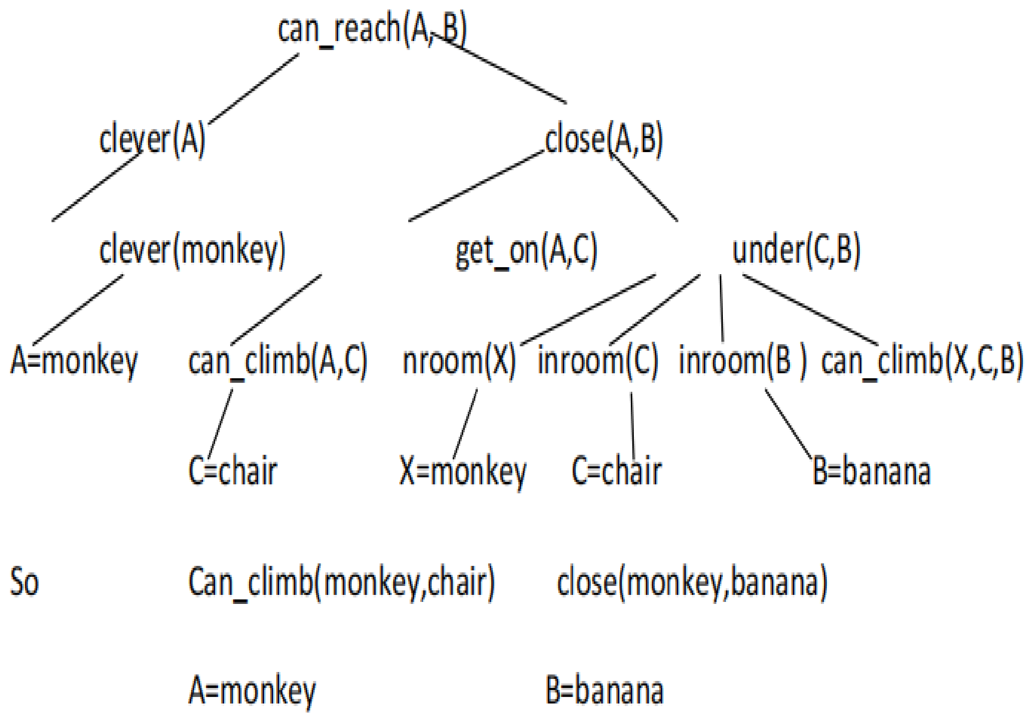
Implementation of Monkey Banana Problem using LISP/PROLOG

Aim: Write a program to solve the Monkey Banana problem.

Description: Imagine a room containing a monkey, chair and some bananas. That have been hanged from the center of ceiling. If the monkey is clever enough he can reach the bananas by placing the chair directly below the bananas and climb on the chair . The problem is to prove the monkey can reach the bananas.

Production Rules
can_reach clever,close.
get_on: can_climb.
under in room,in_room, in_room,can_climb.
Close get_on,under| tall

Parse Tree



Clauses:

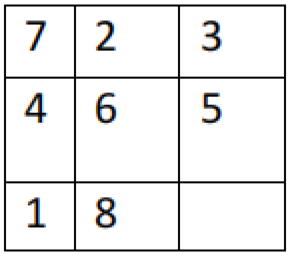
```
in_room(bananas).
in_room(chair).
in_room(monkey). clever(monkey).
can_climb(monkey, chair).
tall(chair).
can_move(monkey, chair, bananas).
can_reach(X, Y):-clever(X),close(X, Y).
get_on(X,Y):- can_climb(X,Y).
under(Y,Z):-in_room(X),in_room(Y),
in_room(Z),can_climb(X,Y,Z).
close(X,Z):-get_on(X,Y), under(Y,Z);
tall(Y).
```

Queries:

```
?- can_reach(A, B).
A = monkey.
B = banana.
?- can_reach(monkey, banana)
yes
```

EXPERIMENT-8

Implementation of Hill-climbing to solve 8- Puzzle Problem.
Aim: To solve 8 puzzle problem using hill climbing.
Description: The title of this section refers to a familiar and popular sliding tile puzzle that has been around for at least forty years. The most frequent older versions of this puzzle have numbers or letters an the sliding tiles, and the player is supposed to slide tiles into new positions in order to realign a scrambled puzzle back into a goal alignment. For illustration, we use the 3 x 3 8-tile version, which is depicted here in goal configuration**.**



7	2	3
4	6	5
1	8	

To represent these puzzle "states" we will use a Prolog term representation employing '/' as a separator. The positions of the tiles are listed (separated by '/') from top to bottom, and from left to right. Use "0" to represent the empty tile (space). For example, the goal is … goal(1/2/3/8/0/4/7/6/5)

Production Rules :-
h_function(Puzz,H) □ p_fcn(Puzz,P), s_fcn(Puzz,S),H is P + 3*S.
The 'move' productions are defined as follows.
move(P,C,left) □ left(P,C).
move(P,C,up) □ up(P,C).
move(P,C,right) □ right(P,C).
move(P,C,down) □ down(P,C).
p_fcn(A/B/C/D/E/F/G/H/I, P) □ a(A,Pa), b(B,Pb), c(C,Pc),
d(D,Pd), e(E,Pe), f(F,Pf),
g(G,Pg), h(H,Ph), i(I,Pi),

```
P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.
s_fcn(A/B/C/D/E/F/G/H/I, S) □1 s_aux(A,B,S1), s_aux(B,C,S2), s_aux(C,F,S3),
s_aux(F,I,S4),
s_aux(I,H,S5), s_aux(H,G,S6),
s_aux(G,D,S7), s_aux(D,A,S8),
s_aux(E,S9),
S is S1+S2+S3+S4+S5+S6+S7+S8+S9.
s_aux(0,0) □ cut
s_aux(X,Y,0) :- Y is X+1, !.
s_aux(8,1,0) :- !.
```

The heuristic function we use here is a combination of two other estimators: p_fcn, the Manhattan distance function, and s_fcn, the sequence function, all as explained in Nilsson (1980), which estimates how badly out-of-sequence the tiles are (around the outside).

```
h_function(Puzz,H) :- p_fcn(Puzz,P),
s_fcn(Puzz,S),
H is P + 3*S.
```

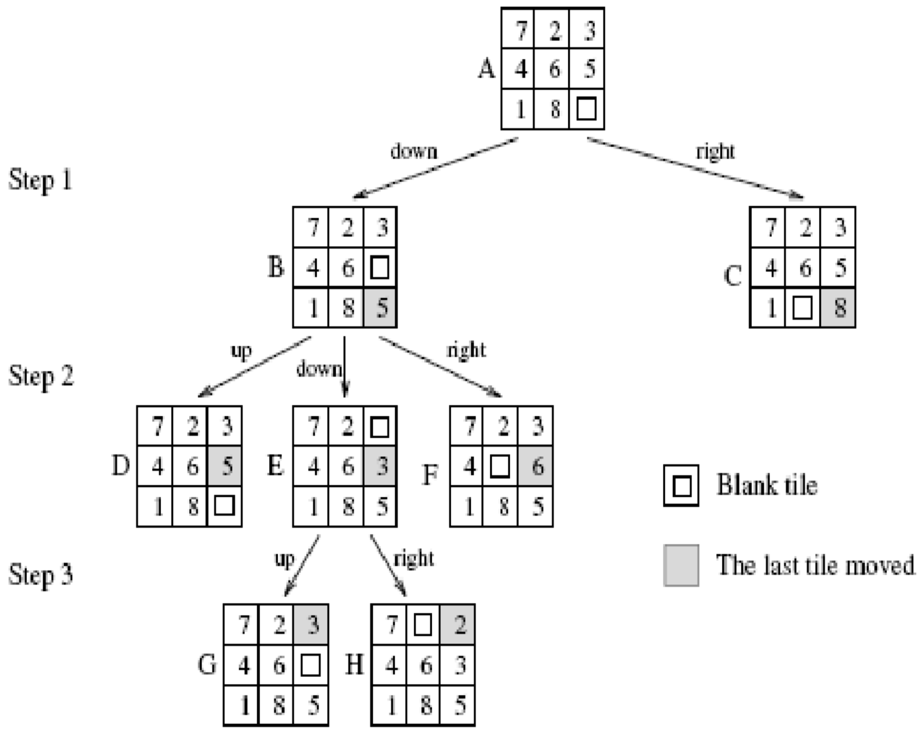
The 'move' predicate is defined as follows.

```
move(P,C,left) :- left(P,C).
move(P,C,up) :- up(P,C).
move(P,C,right) :- right(P,C).
move(P,C,down) :- down(P,C).
```

Here is the code for p and s.

```
%%% Manhattan distance
p_fcn(A/B/C/D/E/F/G/H/I, P) :-
a(A,Pa), b(B,Pb), c(C,Pc),
d(D,Pd), e(E,Pe), f(F,Pf),
g(G,Pg), h(H,Ph), i(I,Pi),
P is Pa+Pb+Pc+Pd+Pe+Pf+Pg+Ph+Pg+Pi.
a(0,0). a(1,0). a(2,1). a(3,2). a(4,3). a(5,4). a(6,3). a(7,2). a(8,1).
b(0,0). b(1,0). b(2,0). b(3,1). b(4,2). b(5,3). b(6,2). b(7,3). b(8,2).
c(0,0). c(1,2). c(2,1). c(3,0). c(4,1). c(5,2). c(6,3). c(7,4). c(8,3).
d(0,0). d(1,1). d(2,2). d(3,3). d(4,2). d(5,3). d(6,2). d(7,2). d(8,0).
e(0,0). e(1,2). e(2,1). e(3,2). e(4,1). e(5,2). e(6,1). e(7,2). e(8,1).
f(0,0). f(1,3). f(2,2). f(3,1). f(4,0). f(5,1). f(6,2). f(7,3). f(8,2).
```

```
g(0,0). g(1,2). g(2,3). g(3,4). g(4,3). g(5,2). g(6,2). g(7,0). g(8,1).
h(0,0). h(1,3). h(2,3). h(3,3). h(4,2). h(5,1). h(6,0). h(7,1). h(8,2).
i(0,0). i(1,4). i(2,3). i(3,2). i(4,1). i(5,0). i(6,1). i(7,2). i(8,3).
%%% the out-of-cycle function
s_fcn(A/B/C/D/E/F/G/H/I, S) :-
s_aux(A,B,S1), s_aux(B,C,S2), s_aux(C,F,S3),
s_aux(F,I,S4), s_aux(I,H,S5), s_aux(H,G,S6),
s_aux(G,D,S7), s_aux(D,A,S8), s_aux(E,S9), S is
S1+S2+S3+S4+S5+S6+S7+S8+S9.
s_aux(0,0) :- !.
s_aux(_,1).
s_aux(X,Y,0) :- Y is X+1, !.
s_aux(8,1,0) :- !.
s_aux(_,_,2).
?- solve(0/8/1/2/4/3/7/6/5, S).
```

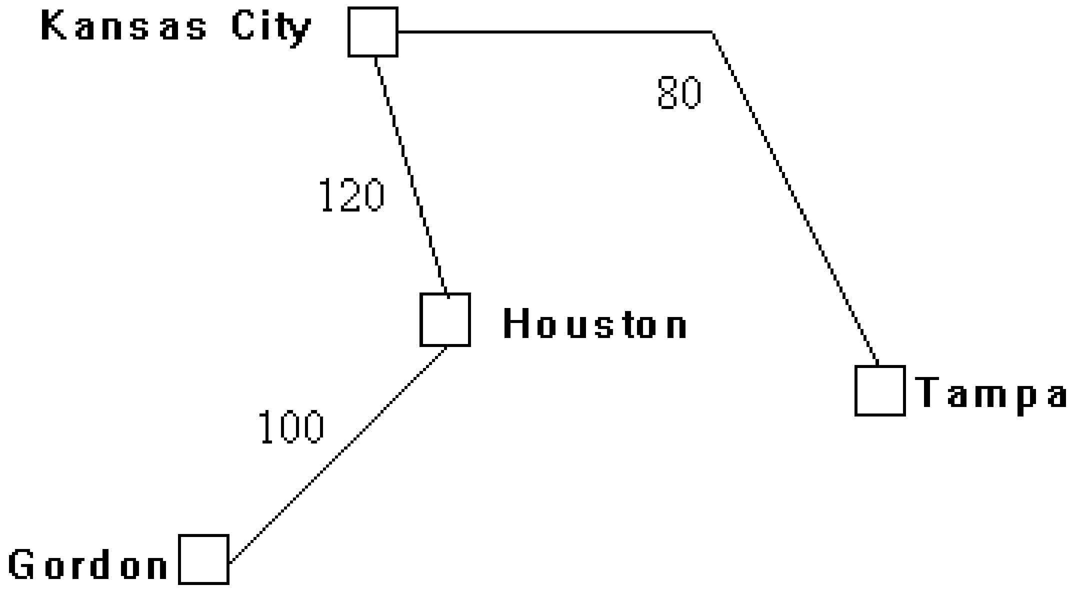


Solution:

```
left( A/0/C/D/E/F/H/I/J , 0/A/C/D/E/F/H/I/J ).
left( A/B/C/D/0/F/H/I/J , A/B/C/0/D/F/H/I/J ).
left( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/0/H/J ).
left( A/B/0/D/E/F/H/I/J , A/0/B/D/E/F/H/I/J ).
left( A/B/C/D/E/0/H/I/J , A/B/C/D/0/E/H/I/J ).
left( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/F/H/0/I ).
up( A/B/C/0/E/F/H/I/J , 0/B/C/A/E/F/H/I/J ).
up( A/B/C/D/0/F/H/I/J , A/0/C/D/B/F/H/I/J ).
up( A/B/C/D/E/0/H/I/J , A/B/0/D/E/C/H/I/J ).
up( A/B/C/D/E/F/0/I/J , A/B/C/0/E/F/D/I/J ).
up( A/B/C/D/E/F/H/0/J , A/B/C/D/0/F/H/E/J ).
up( A/B/C/D/E/F/H/I/0 , A/B/C/D/E/0/H/I/F ).
right( A/0/C/D/E/F/H/I/J , A/C/0/D/E/F/H/I/J ).
right( A/B/C/D/0/F/H/I/J , A/B/C/D/F/0/H/I/J ).
right( A/B/C/D/E/F/H/0/J , A/B/C/D/E/F/H/J/0 ).
right( 0/B/C/D/E/F/H/I/J , B/0/C/D/E/F/H/I/J ).
right( A/B/C/0/E/F/H/I/J , A/B/C/E/0/F/H/I/J ).
right( A/B/C/D/E/F/0/I/J , A/B/C/D/E/F/I/0/J ).
down( A/B/C/0/E/F/H/I/J , A/B/C/H/E/F/0/I/J ).
down( A/B/C/D/0/F/H/I/J , A/B/C/D/I/F/H/0/J ).
down( A/B/C/D/E/0/H/I/J , A/B/C/D/E/J/H/I/0 ).
down( 0/B/C/D/E/F/H/I/J , D/B/C/0/E/F/H/I/J ).
down( A/0/C/D/E/F/H/I/J , A/E/C/D/0/F/H/I/J ).
down( A/B/0/D/E/F/H/I/J , A/B/F/D/E/0/H/I/J).
```

EXPERIMENT-9

Implementation of TSP using heuristic approach using Java/LISP/Prolog.
Aim: To implement travelling sales person using prolog.
Description: The following is the simplified map used for the prototype:



Production Rules:-
route(Town1,Town2,Distance) road(Town1,Town2,Distance).
route(Town1,Town2,Distance) road(Town1,X,Dist1),
route(X,Town2,Dist2),
Distance=Dist1+Dist2,
domains
town = symbol
distance = integer
predicates
nondeterm road(town,town,distance)
nondeterm route(town,town,distance)
clauses
road("tampa","houston",200).

```
road("gordon","tampa",300).
road("houston","gordon",100).
road("houston","kansas_city",120).
road("gordon","kansas_city",130).
route(Town1,Town2,Distance):-
road(Town1,Town2,Distance).
route(Town1,Town2,Distance):- road(Town1,X,Dist1),
route(X,Town2,Dist2),
Distance=Dist1+Dist2,
!.
```

goal

```
route("tampa", "kansas_city", X),
write("Distance from Tampa to Kansas City is ",X),nl.
Distance from Tampa to Kansas City is 320
X=320
1 Solution
```

EXPERIMENT-10

Implementation of DFS for water jug problem using LISP/PROLOG

Aim: To implement water jug problem using prolog.

Description: You are given two jugs, a 4-gallon one and a 3-gallon one. Neither have any measuring markers on it. There is a tap that can be used to fill the jugs with water. How can you get exactly 2 gallons of water into the 4-gallon jug?".

Production Rules:-

R1: (x,y) --> (4,y) if x < 4
R2: (x,y) --> (x,3) if y < 3
R3: (x,y) --> (x-d,y) if x > 0
R4: (x,y) --> (x,y-d) if y > 0
R5: (x,y) --> (0,y) if x > 0
R6: (x,y) --> (x,0) if y > 0
R7: (x,y) --> (4,y-(4-x)) if x+y >= 4 and y > 0
R8: (x,y) --> (x-(3-y),y) if x+y >= 3 and x > 0
R9: (x,y) --> (x+y,0) if x+y =< 4 and y > 0
R10: (x,y) --> (0,x+y) if x+y =< 3 and x > 0

Parse Tree

```
<0,0>
/ \
R1 R2
<0,3>
/ |...\
/ |... \
R1 R4... R9
<3,0>
/ | \
/ | \
R2 R3 R5
<3,3>
/ |...\...
```

/ |... \...
R1 R3 R7
<4,2>
/ |...\...
/ |... \...
R2 R3 R5
**<0,2> / | **
**/ | **
R1 R7 R9
<2,0>

Depth First Search Algorithm

```
Successor,successor (Node, Capacities, Visited, Successor),Successors),
append( Nodes, Successors, NewNodes ),
solve_jugs( NewNodes, Capacities, [State|Visited], End, Solution )).
successor( Node, Capacities, Visited, Successor ) :-
node_state( Node, State ),
Successor = node(Action,State1,Node),
jug_transition( State, Capacities, Action, State1 ),\+ member( State1, Visited ).
jug_transition( State0, Capacities, empty_into(Source,Target), State1 ) :-
volume( Source, State0, Content0 ),Content0 > 0,
jug_permutation( Source, Target, Unused ),
volume( Target, State0, Content1 ),
volume( Target, Capacities, Capacity ),
Content0 + Content1 =< Capacity,volume( Source, State1, 0 ),
volume( Target, State1, Content2 ),
Content2 is Content0 + Content1,
volume( Unused, State0, Unchanged ),
volume( Unused, State1, Unchanged ).
jug_transition( State0, Capacities, fill_from(Source,Target), State1 ) :-
volume( Source, State0, Content0 ),Content0 > 0,
jug_permutation( Source, Target, Unused ),
volume( Target, State0, Content1 ),
volume( Target, Capacities, Capacity ),Content1 < Capacity,Content0 + Content1 >
Capacity,
volume( Source, State1, Content2 ),
volume( Target, State1, Capacity ),
Content2 is Content0 + Content1 - Capacity,
volume( Unused, State0, Unchanged ),
volume( Unused, State1, Unchanged ).
volume( small, jugs(Small, _Large, _Reservoir), Small ).
volume( large, jugs(_Small, Large, _Reservoir), Large ).
volume( reservoir, jugs(_Small, _Large, Reservoir), Reservoir ).
jug_permutation( Source, Target, Unused ) :-
select( Source, [small, large, reservoir], Residue ),
select( Target, Residue, [Unused] ).
node_state( start(State), State ).
node_state( node(_Transition, State, _Predecessor), State ).
narrative( start(Start), Capacities, End ) -->
"Given three jugs with capacities of:", newline,
literal_volumes( Capacities ),
"To obtain the result:", newline,
literal_volumes( End ),
"Starting with:", newline,
literal_volumes( Start ),
"Do the following:", newline.
narrative( node(Transition, Result, Predecessor), Capacities, End ) -->
narrative( Predecessor, Capacities, End ),
literal_action( Transition, Result ).literal_volumes( Volumes ) -->
indent, literal( Volumes ), ";", newline.
literal_action( Transition, Result ) -->
indent, "- ", literal( Transition ), " giving:", newline,
indent, indent, literal( Result ), newline.
literal( empty_into(From,To) ) -->
"Empty the ", literal( From ), " into the ",
literal( To ).
literal( fill_from(From,To) ) -->
"Fill the ", literal( To ), " from the ",
literal( From ).
literal( jugs(Small,Large,Reservoir) ) -->
```

```
literal_number( Small ), " gallons in the small jug, ",
literal_number( Large ), " gallons in the large jug and ",
literal_number( Reservoir ), " gallons in the reservoir".
literal( small ) --> "small jug".
literal( large ) --> "large jug".
literal( reservoir ) --> "reservoir".
literal_number( Number, Plus, Minus ) :-
number( Number ),
number_chars( Number, Chars ),
append( Chars, Minus, Plus ).
indent --> " ".
newline --> " ".
```

Goal

```
?- water_jugs.
Given three jugs with capacities of:
3 gallons in the small jug, 4 gallons in the large jug and 8 gallons in the reservoir;
To obtain the result:
0 gallons in the small jug, 2 gallons in the large jug and 6 gallons in the reservoir;
Starting with:
0 gallons in the small jug, 0 gallons in the large jug and 8 gallons in the reservoir;
Do the following:
- Fill the small jug from the reservoir giving:
3 gallons in the small jug, 0 gallons in the large jug and 5 gallons in the reservoir -
Empty the small jug into the large jug giving:
0 gallons in the small jug, 3 gallons in the large jug and 5 gallons in the reservoir
- Fill the small jug from the reservoir giving:
3 gallons in the small jug, 3 gallons in the large jug and 2 gallons in the reservoir
- Fill the large jug from the small jug giving:
2 gallons in the small jug, 4 gallons in the large jug and 2 gallons in the reservoir
- Empty the large jug into the reservoir giving:
2 gallons in the small jug, 0 gallons in the large jug and 6 gallons in the reservoir
- Empty the small jug into the large jug giving:
0 gallons in the small jug, 2 gallons in the large jug and 6 gallons in the reservoir.
```

EXPERIMENT-11

Implementation of Tower Of Hanoi problem using LISP/PROLOG.

Aim: To implement Tower of hanoi problem using prolog.

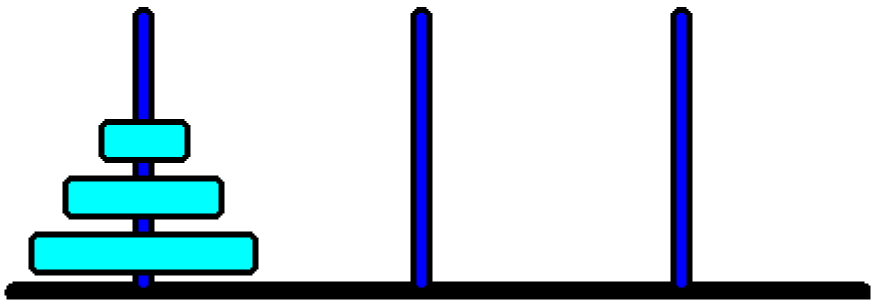
Description: This object of this famous puzzle is to move N disks from the left peg to the right peg using the center peg as an auxiliary holding peg. At no time can a larger disk be placed upon a smaller disk. The following diagram depicts the starting setup for N=3 disks.

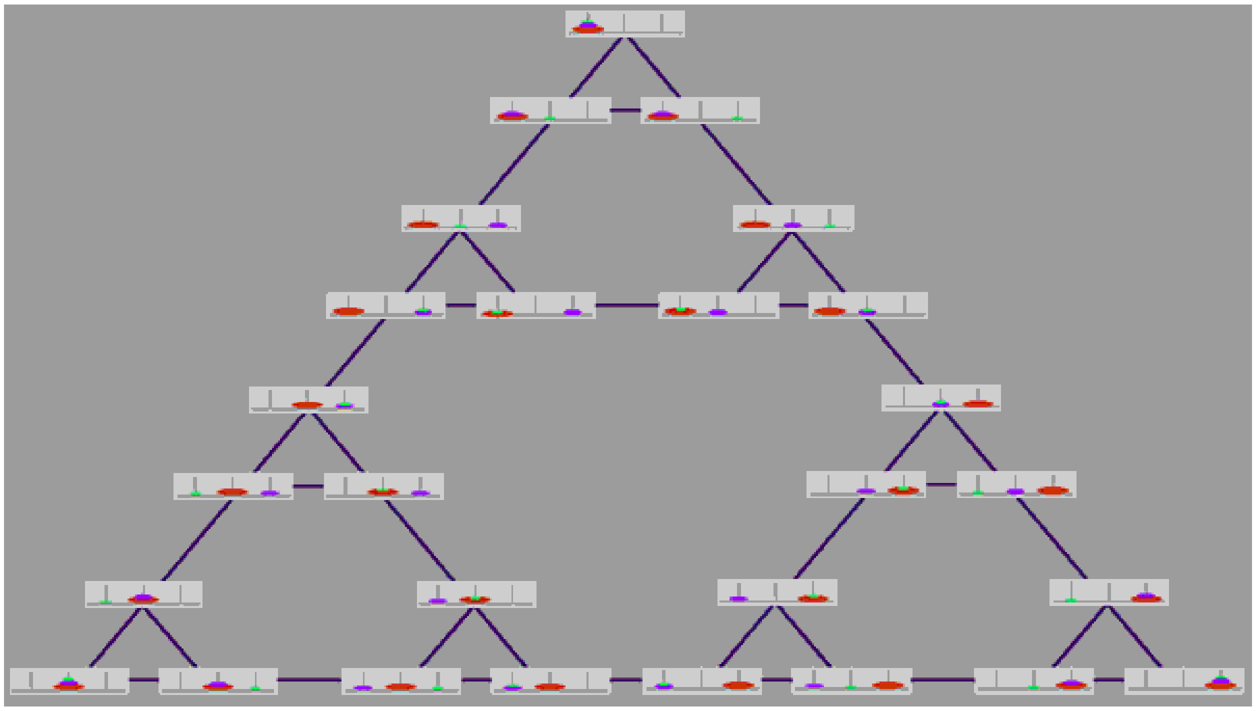
Production Rules

hanoi(N)□move(N,left,middle,right).

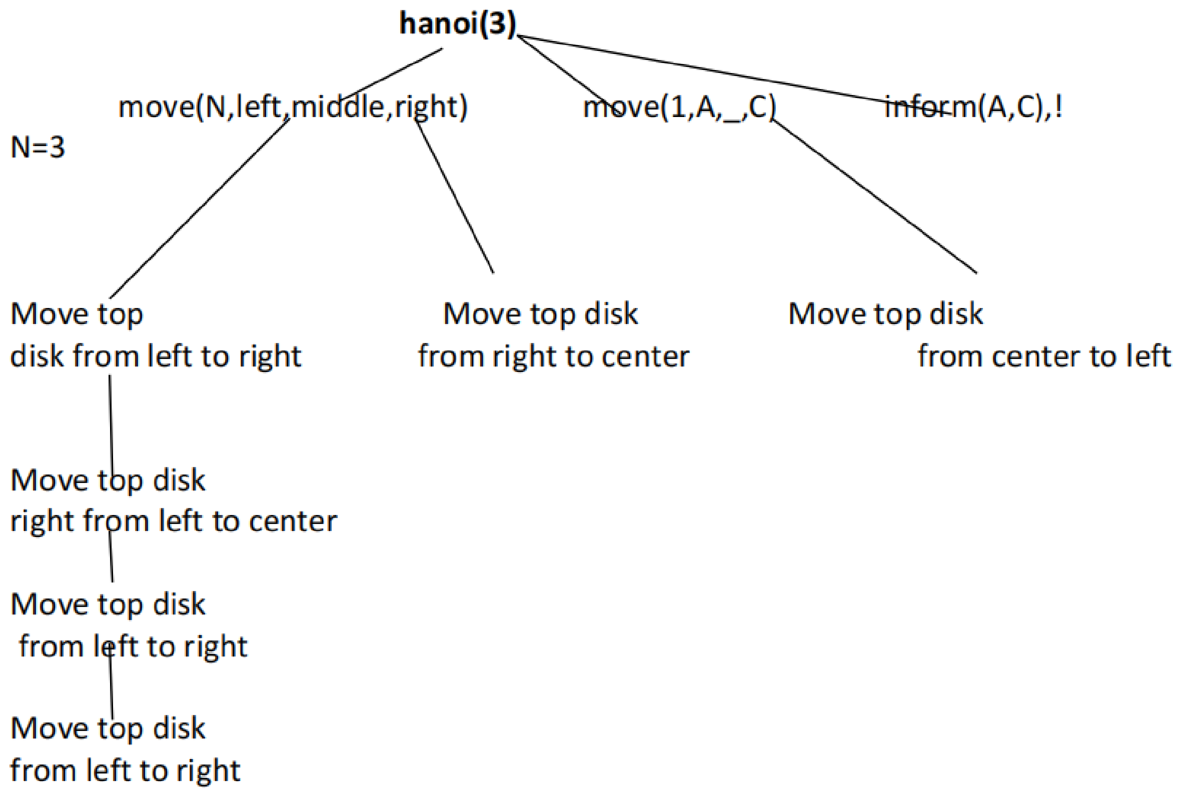
move(1,A,_,C)□inform(A,C),fail.

move(N,A,B,C)□N1=N-1,move(N1,A,C,B),inform(A,C),move(N1,B,A,C)





Parse Tree:


hanoi(3)
move(N,left,middle,right)
N=3
move(1,A,_,C)
inform(A,C),!
Move top disk from left to right
Move top disk from right to center
Move top disk from center to left
Move top disk right from left to center
Move top disk from left to right
Move top disk from left to right

Solution:-
domains

```
loc =right;middle;left
```

predicates

```
hanoi(integer)
move(integer,loc,loc,loc)
inform(loc,loc)
```

clauses

```
hanoi(N):-
move(N,left,middle,right).
move(1,A,_,C):-
inform(A,C),!.
move(N,A,B,C):-
N1=N-1,
move(N1,A,C,B),
inform(A,C),
move(N1,B,A,C).
inform(Loc1, Loc2):-
write("\nMove a disk from ", Loc1, " to ", Loc2).
```

goal

```
hanoi(3).
Move(3,left,right,center).
Move top disk from left to right
Move top disk from left to center
Move top disk from right to center
Move top disk from left to right
Move top disk from center to left
Move top disk from center to right
Move top disk from left to right
Yes
```

EXPERIMENT-12

Implement 4 queens problem using LISP/PROLOG.

Aim: To implement 4 queens problem using prolog.

Description: In the 4 Queens problem the object is to place *4* queens on a chessboard in such a way that no queens can capture a piece. This means that no two queens may be placed on the same row, column, or diagonal.

	1	2	3	4
1	1	2	3	4
2	2	3	4	5
3	3	4	5	6
4	4	5	6	7

```
domains
queen = q(integer, integer)
queens = queen*
freelist = integer*
board = board(queens, freelist, freelist, freelist, freelist)

predicates
nondeterm placeN(integer, board, board)
nondeterm place_a_queen(integer, board, board)
nondeterm nqueens(integer)
nondeterm makelist(integer, freelist)
nondeterm findandremove(integer, freelist, freelist)
```

```
nextrow(integer, freelist, freelist)
```

clauses

```
nqueens(N):-
makelist(N,L),
Diagonal=N*2-1,
makelist(Diagonal,LL),
placeN(N,board([],L,L,LL,LL),Final),
write(Final).
placeN(_,board(D,[],[],D1,D2),board(D,[],[],D1,D2)):-!.
placeN(N,Board1,Result):- place_a_queen(N,Board1,Board2),
placeN(N,Board2,Result).
place_a_queen(N,
board(Queens,Rows,Columns,Diag1,Diag2),
board([q(R,C)|Queens],NewR,NewC,NewD1,NewD2)):-
nextrow(R,Rows,NewR),
findandremove(C,Columns,NewC),
D1=N+C-R,findandremove(D1,Diag1,NewD1),
D2=R+C-1,findandremove(D2,Diag2,NewD2).
findandremove(X,[X|Rest],Rest).
findandremove(X,[Y|Rest],[Y|Tail]):-
findandremove(X,Rest,Tail).
makelist(1,[1]).
makelist(N,[N|Rest]) :-
N1=N-1,makelist(N1,Rest).
nextrow(Row,[Row|Rest],Rest).
```

goal

```
nqueens(4),nl.
board([q(1,2),q(2,4),q(3,1),q(4,3),[],[],[7,4,1],[7,4,1])
yes
```