**Exp-1 FIND-S ALGORITHM**

**Algorithm:**

1. Initialize h to the most specific hypothesis in H.
2. For each positive training instance x for each attribute constraint a in h.
3. If the constrain a, is satisfied by x then do nothing.
4. Else, replace a in h by the nest more general constraint that is satisfied by x.
5. Output hypothesis h.

**Program:**

import pandas as pd

import numpy as np

data = pd.read\_csv('Sports.csv')

concepts = np.array(data)[:,:-1]

target = np.array(data)[:,-1]

def training(con,tar):

for i,val in enumerate(tar):

if val =='yes':

specific\_h = con[i].copy( break

for i,val in enumerate (con):

if tar[i] =='yes':for x in range(len(specific\_h)):

if val[x]!=specific\_h[x]:

specific\_h[x] = '?'

else: pass

return specific\_h

print(training(concepts,target))

**Exp-2** – **CANDIDATE ELIINATION ALGORITHM**

**Algorithm:**

1. Load data set.Initialize general hypothesis and specific hypothesis For each training example.
2. If examples is positive exampleIf attribute\_value ==hypothesis value:

Do nothing Else:Replace attribute value with ‘?’.

1. If example is negative example make generdize hypothesis more specific.

**Program:**

import numpy as np

import pandas as pd

data = pd.read\_csv('sports.csv')

concepts = np.array(data.iloc[:,0:-1])

print("\nInstances are:\n",concepts)

target = np.array(data.iloc[:,-1])

def learn(concepts, target):

specific\_h = concepts[0].copy()

print("\nInitialization of specific\_h and genearal\_h")

print("\nSpecific Boundary: ", specific\_h)

general\_h = [["?" for i in range(len(specific\_h))] for i in range(len(specific\_h))]

print("\nGeneric Boundary: ",general\_h)

for i, h in enumerate(concepts):

print("\nInstance", i+1 , "is ", h)

if target[i] == "yes":print("Instance is Positive ")

for x in range(len(specific\_h)):

if h[x]!= specific\_h[x]:

specific\_h[x] ='?' general\_h[x][x] ='?'

if target[i] == "no": print("Instance is Negative ")

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] = '?'

print("Specific Bundary after ", i+1, "Instance is ", specific\_h)

print("Generic Boundary after ", i+1, "Instance is ", general\_h)

indices = [i for i, val in enumerate(general\_h) if val == ['?', '?', '?', '?', '?', '?']]

for i in indices:

general\_h.remove(['?', '?', '?', '?', '?', '?'])

return specific\_h, general\_h

s\_final, g\_final = learn(concepts, target)

print("Final Specific\_h: ", s\_final, sep="\n")

print("Final General\_h: ", g\_final, sep="\n")

**EXP-3 DECISION TREE BASED ID3 ALGORITHM**

**Algorithm:**

1. Calculate entropy for the algorithm.
2. For each node

* Calculate entropy for all its categorical values.
* Calculate information gain for the node.

1. Find the node with the highest information gain at particular level.
2. Repeat step from 1 to 3 till.
3. We reach the leaf node and have created our decision tree.

**Program:**

import pandas as pd

import math

import numpy as np

data = pd.read\_csv("kk3.csv")

features = [feat for feat in data]

features.remove("answer")

class Node:

def \_\_init\_\_(self):

self.children=[]

self.value = ""

self.isLeaf = False

self.pred = ""

def entropy(examples):

pos = 0.0

neg = 0.0

for \_, row in examples.iterrows():

if row["answer"] == "yes":

pos += 1

else:

neg += 1

if pos == 0.0 or neg == 0.0:

return 0.0

else:

p = pos / (pos + neg)

n = neg / (pos + neg)

return -(p \* math.log(p, 2) + n \* math.log(n, 2))

def info\_gain(examples, attr):

uniq = np.unique(examples[attr])

#print ("\n",uniq)

gain = entropy(examples)

#print ("\n",gain)

for u in uniq:

subdata = examples[examples[attr] == u]

#print ("\n",subdata)

sub\_e = entropy(subdata)

gain -= (float(len(subdata)) / float(len(examples))) \* sub\_e

#print ("\n",gain)

return gain

def ID3(examples, attrs):

root = Node()

max\_gain = 0

max\_feat = ""

for feature in attrs:

#print ("\n",examples)

gain = info\_gain(examples, feature)

if gain > max\_gain:

max\_gain = gain

max\_feat = feature

root.value = max\_feat

#print ("\nMax feature attr",max\_feat)

uniq = np.unique(examples[max\_feat])

#print ("\n",uniq)

for u in uniq:

#print ("\n",u)

subdata = examples[examples[max\_feat] == u]

#print ("\n",subdata)

if entropy(subdata) == 0.0:

newNode = Node()

newNode.isLeaf = True

newNode.value = u

newNode.pred = np.unique(subdata["answer"])

root.children.append(newNode)

else:

dummyNode = Node()

dummyNode.value = u

new\_attrs = attrs.copy()

new\_attrs.remove(max\_feat)

child = ID3(subdata, new\_attrs)

dummyNode.children.append(child)

root.children.append(dummyNode)

return root

def printTree(root: Node, depth=0):

for i in range(depth):

print("\t", end="")

print(root.value, end="")

if root.isLeaf:

print(" -> ", root.pred)

print()

for child in root.children:

printTree(child, depth + 1)

root = ID3(data, features)

printTree(root)

**EXP-4 - BACK PROPAGATION ALGORITHM**

**Algorithm:**

1. Input x, arrive through the preconnected path.
2. The input is modelled using tree weights w. Weights are usually chosen randomly.
3. Calculate the output of each neuron from the input layer to the hidden layer to the output layer.
4. Calculate the errors in the output.

Back propagation error = actual output – desired output.

1. From the output layer, go back to the hidden layer to adjust the weights to reduce the error.
2. Repeat the process until the desired output is achieved.

**Program:**

import numpy as np

X = np.array(([2, 9], [1, 5], [3, 6]), dtype=float)

y = np.array(([92], [86], [89]), dtype=float)

X = X/np.amax(X,axis=0) #maximum of X array longitudinally

y = y/100

#Sigmoid Function

def sigmoid (x):

return 1/(1 + np.exp(-x))

#Derivative of Sigmoid Function

def derivatives\_sigmoid(x):

return x \* (1 - x)

#Variable initialization

epoch=5 #Setting training iterations

lr=0.1 #Setting learning rate

inputlayer\_neurons = 2 #number of features in data set

hiddenlayer\_neurons = 3 #number of hidden layers neurons

output\_neurons = 1 #number of neurons at output layer

#weight and bias initialization

wh=np.random.uniform(size=(inputlayer\_neurons,hiddenlayer\_neurons))

bh=np.random.uniform(size=(1,hiddenlayer\_neurons))

wout=np.random.uniform(size=(hiddenlayer\_neurons,output\_neurons))

bout=np.random.uniform(size=(1,output\_neurons))

#draws a random range of numbers uniformly of dim x\*y

for i in range(epoch):

#Forward Propogation

hinp1=np.dot(X,wh)

hinp=hinp1 + bh

hlayer\_act = sigmoid(hinp)

outinp1=np.dot(hlayer\_act,wout)

outinp= outinp1+bout

output = sigmoid(outinp)

#Backpropagation

EO = y-output

outgrad = derivatives\_sigmoid(output)

d\_output = EO \* outgrad

EH = d\_output.dot(wout.T)

hiddengrad = derivatives\_sigmoid(hlayer\_act)#how much hidden layer wts contributed to e rror

d\_hiddenlayer = EH \* hiddengrad

wout += hlayer\_act.T.dot(d\_output) \*lr # dotproduct of nextlayererror and currentlayerop wh += X.T.dot(d\_hiddenlayer) \*lr

print ("-----------Epoch-", i+1, "Starts----------")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

print ("-----------Epoch-", i+1, "Ends----------\n")

print("Input: \n" + str(X))

print("Actual Output: \n" + str(y))

print("Predicted Output: \n" ,output)

**EXP-5 - NAÏVE BAYESIAN CLASSIFIER**

**Algorithm:**

1. Read the training dataset T.
2. Calculate the mean and standard deviation of the predictor variables in each class.
3. Repeat until the probability of all prediction variables.
4. Calculate the likelihood for each class.
5. Get the greatest likelihood.

Program:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

dataset = pd.read\_csv("Naive-Bayes-Classification-Data.csv")

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

from sklearn.model\_selection import train\_test\_split

# assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

from sklearn.preprocessing import StandardScaler

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# importing classifier

from sklearn.naive\_bayes import BernoulliNB

# import Gaussian Naive Bayes classifier

from sklearn.naive\_bayes import GaussianNB

# create a Gaussian Classifier

classifer1 = GaussianNB()

# training the model

classifer1.fit(X\_train, y\_train)

# testing the model

y\_pred1 = classifer1.predict(X\_test)

# importing accuracy score

from sklearn.metrics import accuracy\_score

# printing the accuracy of the model

print(accuracy\_score(y\_test,y\_pred1))

**EXP-6 - NAÏVE BAYESIAN CLASSIFIER MODE PRECISION, AND RECAL**

**Algorithm:**

1. Import basic libraries.
2. Importing the dataset.
3. Data preprocessing.
4. Training the model.
5. Testing and evaluation of the model.
6. Visualizing the model.

**Program:**

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import seaborn as sns

# importing the dataset

dataset = pd.read\_csv("Naive-Bayes-Classification-Data.csv")

# split the data into inputs and outputs

X = dataset.iloc[:, [0,1]].values

y = dataset.iloc[:, 2].values

# training and testing data

from sklearn.model\_selection import train\_test\_split

# assign test data size 25%

X\_train, X\_test, y\_train, y\_test =train\_test\_split(X,y,test\_size= 0.25, random\_state=0)

# importing standard scaler

from sklearn.preprocessing import StandardScaler

# scalling the input data

sc\_X = StandardScaler()

X\_train = sc\_X.fit\_transform(X\_train)

X\_test = sc\_X.fit\_transform(X\_test)

# importing classifier

from sklearn.naive\_bayes import BernoulliNB

# import Gaussian Naive Bayes classifier

from sklearn.naive\_bayes import GaussianNB

# create a Gaussian Classifier

classifer1 = GaussianNB()

# training the model

classifer1.fit(X\_train, y\_train)

# testing the model

y\_pred1 = classifer1.predict(X\_test)

# importing accuracy score

from sklearn.metrics import accuracy\_score

# printing the accuracy of the model

print(accuracy\_score(y\_test,y\_pred1))

from sklearn.metrics import accuracy\_score, confusion\_matrix, precision\_score, recall\_score

print('Accuracy Metrics: \n')

print('Accuracy: ', accuracy\_score(y\_test, y\_pred1))

print('Recall: ', recall\_score(y\_test, y\_pred1))

print('Precision: ', precision\_score(y\_test, y\_pred1))

print('Confusion Matrix: \n', confusion\_matrix(y\_test, y\_pred1))

**EXP-7 - BAYESIAN NETWORK CONSIDERING MEDICAL DATA**

**Algorithm:**

1. Identify which are the main variable in the problem to solve.
2. Fine structure of the network that is the causal relationships between all the variable(nodes).
3. Define the probability rules governing the relationship between the variables.

**PROGRAM:**

import numpy as np

import pandas as pd

import csv

from pgmpy.estimators import MaximumLikelihoodEstimator

from pgmpy.models import BayesianNetwork

from pgmpy.inference import VariableElimination

heartDisease = pd.read\_csv('kk7.csv')

heartDisease = heartDisease.replace('?',np.nan)

print('Sample instances from the dataset are given below')

print(heartDisease.head())

print('\n Attributes and datatypes')

print(heartDisease.dtypes)

model= BayesianNetwork([('age','heartdisease'),('gender','heartdisease'),('exang','heartdisease'),('cp','heartdisease'),('heartdisease','restecg'),('heartdisease','chol')])

print('\nLearning CPD using Maximum likelihood estimators')

model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)

print('\n Inferencing with Bayesian Network:')

HeartDiseasetest\_infer = VariableElimination(model)

print('\n 1. Probability of HeartDisease given evidence= restecg')

q1=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'restecg':1})

print(q1)

print('\n 2. Probability of HeartDisease given evidence= cp ')

q2=HeartDiseasetest\_infer.query(variables=['heartdisease'],evidence={'cp':2})

print(q2)

**EXP-8 EM ALGORITHM K-MEANS ALGORITHM**

**Algorithm:**

1. Import necessary libraries for implementing k-means and EM algorithm.
2. Input a dataset to preform action.
3. Process the dataset and perform EM and K-means algorithm for same dataset.
4. Print the accuracy sore of the both the algorithm as output.

**Program:**

from sklearn.cluster import KMeans

from sklearn.mixture import GaussianMixture

import sklearn.metrics as metrics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

names = ['Sepal\_Length','Sepal\_Width','Petal\_Length','Petal\_Width', 'Class']

dataset = pd.read\_csv("kk8.csv", names=names)

X = dataset.iloc[:, :-1]

label = {'Iris-setosa': 0,'Iris-versicolor': 1, 'Iris-virginica': 2}

y = [label[c] for c in dataset.iloc[:, -1]]

plt.figure(figsize=(14,7))

colormap=np.array(['red','lime','black'])

# REAL PLOT

plt.subplot(1,3,1)

plt.title('Real')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y])

# K-PLOT

model=KMeans(n\_clusters=3, random\_state=0).fit(X)

plt.subplot(1,3,2)

plt.title('KMeans')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[model.labels\_])

print('The accuracy score of K-Mean: ',metrics.accuracy\_score(y, model.labels\_))

print('The Confusion matrixof K-Mean:\n',metrics.confusion\_matrix(y, model.labels\_))

# GMM PLOT

gmm=GaussianMixture(n\_components=3, random\_state=0).fit(X)

y\_cluster\_gmm=gmm.predict(X)

plt.subplot(1,3,3)

plt.title('GMM Classification')

plt.scatter(X.Petal\_Length,X.Petal\_Width,c=colormap[y\_cluster\_gmm])

print('The accuracy score of EM: ',metrics.accuracy\_score(y, y\_cluster\_gmm))

print('The Confusion matrix of EM:\n ',metrics.confusion\_matrix(y, y\_cluster\_gmm))

**EXP-9 - K-NEAREST NEIGHBOUR ALGORITHM**

**Algorithm:**

1. Import necessary libraries.
2. Input dataset.
3. Select k numbers of the neighbours.
4. Calculate Euclidean distance among them.
5. Take the nearest neighbour k as per the distance.
6. Count number of datapoints in each category.
7. Assign new datapoint to that category in which the number of neighbour is maximum.
8. Resulted model is the output.

**Program:**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

dataset = pd.read\_csv("kk9.csv", names=names)

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

print(X.head())

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.10)

classifier = KNeighborsClassifier(n\_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0

print ("\n-------------------------------------------------------------------------")

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print ("-------------------------------------------------------------------------")

for label in ytest:

print ('%-25s %-25s' % (label, ypred[i]), end="")

if (label == ypred[i]):

print (' %-25s' % ('Correct'))

else:

print (' %-25s' % ('Wrong'))

i = i + 1

print ("-------------------------------------------------------------------------")

print("\nConfusion Matrix:\n",metrics.confusion\_matrix(ytest, ypred))

print ("-------------------------------------------------------------------------")

print("\nClassification Report:\n",metrics.classification\_report(ytest, ypred))

print ("-------------------------------------------------------------------------")

print('Accuracy of the classifer is %0.2f' % metrics.accuracy\_score(ytest,ypred))

print ("-------------------------------------------------------------------------")

**Exp-11 8 QUEENS**

**Algorithm:**

1. Initialize an empty 8x8 chessboard.
2. Start with 1st column and the 1st row([0,0}).
3. Place a queen in the current cell.
4. Move to the column and find a valid row to place the queen.
5. Check the current queen placement.
6. If all row have been checked and no valid position is found go to next step.
7. Backtrack to the previous column and move the queen to the next valid row in that column.
8. If the queen is successfully placed in the last column, a valid solution has been found.
9. Then continue the steps until all possible solution have been found.

CODE:

% render solutions nicely.

:- use\_rendering(chess).

%% queens(+N, -Queens) is nondet.

%

% @param Queens is a list of column numbers for placing the queens.

% @author Richard A. O'Keefe (The Craft of Prolog)

queens(N, Queens) :-

length(Queens, N),

board(Queens, Board, 0, N, \_, \_),

queens(Board, 0, Queens).

board([], [], N, N, \_, \_).

board([|Queens], [Col-Vars|Board], Col0, N, [|VR], VC) :-

Col is Col0+1,

functor(Vars, f, N),

constraints(N, Vars, VR, VC),

board(Queens, Board, Col, N, VR, [\_|VC]).

constraints(0, \_, \_, \_) :- !.

constraints(N, Row, [R|Rs], [C|Cs]) :-

arg(N, Row, R-C),

M is N-1,

constraints(M, Row, Rs, Cs).

queens([], \_, []).

queens([C|Cs], Row0, [Col|Solution]) :-

Row is Row0+1,

select(Col-Vars, [C|Cs], Board),

arg(Row, Vars, Row-Row),

queens(Board, Row, Solution).

/\*\* <examples>

?- queens(8, Queens).

\*/

**EXP-12 -DEPTH FIRST SEARCH**

Algorithm:

1. Initially stack and visited arrays are empty.
2. Visit 0 and put its adjacent nodes which are not yet into the stack.
3. Now node 1 at the top of the stack, so visited node 1 and pop it from the stack and put all of its adjacent nodes. which are not visited is the stack.
4. Noe node 2 at the top of the stack, so visit node 2 and pop it from the stack and put all of its adjacent nodes are not visited in the stack.
5. Now, stack becomes empty, which means we have visited all the nodes and our DFS traversal ends.

Program:

:- dynamic true/1, does/2.

role(farmer).

init(left(cabbage)).

init(left(goat)).

init(left(wolf)).

init(left(farmer)).

init(step(1)).

legal(farmer, boat(X)) :-

true(left(farmer)),

true(left(X)),

X \== farmer.

legal(farmer, boat(X)) :-

true(right(farmer)),

true(right(X)),

X \== farmer.

legal(farmer, boat(empty)).

next(left(farmer)) :- true(right(farmer)).

next(right(farmer)) :- true(left(farmer)).

next(step(N)) :- true(step(M)), N is M + 1.

next(left(X)) :-

true(right(X)),

does(farmer, boat(X)).

next(right(X)) :-

true(left(X)),

does(farmer, boat(X)).

next(left(X)) :-

true(left(X)),

does(farmer, boat(Y)),

X \== Y,

X \== farmer.

next(right(X)) :-

true(right(X)),

does(farmer, boat(Y)),

X \== Y,

X \== farmer.

goal(farmer, 100) :-

true(right(cabbage)),

true(right(goat)),

true(right(wolf)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(left(cabbage)),

true(left(goat)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(left(wolf)),

true(left(goat)),

true(right(farmer)), !.

goal(farmer, 0) :-

true(right(cabbage)),

true(right(goat)),

true(left(farmer)), !.

goal(farmer, 0) :-

true(right(wolf)),

true(right(goat)),

true(left(farmer)), !.

goal(farmer, 0) :-

true(step(8)), !.

goal(farmer, 50).

terminal :- goal(farmer, 100).

terminal :- goal(farmer, 0).

%% Heuristic predicate with auxilaries which are puzzle specific

% Must always be higher than zero and monotonic (ie never decreasing)

heuristic(State, [goal(farmer, Value)]) :-

member(step(Step), State),

countrights(State, 0, Rights),

Value is Step + Rights.

countrights([], Value, Value).

countrights([right(Item)|State], Count, Value) :-

Item \== farmer, !,

CountInc is Count + 1,

countrights(State, CountInc, Value).

countrights([\_|State], Count, Value) :-

countrights(State, Count, Value).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*\*

?- time(solve\_dfs(Path)).

Path = [

does(farmer,boat(goat)),

does(farmer,boat(empty)),

does(farmer,boat(cabbage)),

does(farmer,boat(goat)),

does(farmer,boat(wolf)),

does(farmer,boat(empty)),

does(farmer,boat(goat))

]

**Exp-13 – BREADTH FIRST SEARCH**

Algorithm:

1. Initialize a open list and closed list.
2. Create the initial state of the puzzle with a given number.

\* the empty tile is represented by a blank space.

1. Calculate heuristic value for the initial state.
2. Create a node for the initial state and add it to the open list.
3. Repeat the steps until either a solution is found or the list is empty.

* Select the node with low value or total cost.
* Remove the selected node from the open list and add it to the closed list.
* Generate child node by moving empty tile.
* For each child node.

1. Calculate h-value.
2. Calculate f- value.
3. If the child node is already in the open list with a lowest f-value.

* Sort the open list based on the f-value of the node.

1. If the open list based or empty and no solution is found the puzzle is unsolvable**.**

**Program:**

test(Plan):-

write('Initial state:'),nl,

Init= [at(tile4,1), at(tile3,2), at(tile8,3), at(empty,4), at(tile2,5), at(tile6,6), at(tile5,7), at(tile1,8), at(tile7,9)],

write\_sol(Init),

Goal= [at(tile1,1), at(tile2,2), at(tile3,3), at(tile4,4), at(empty,5), at(tile5,6), at(tile6,7), at(tile7,8), at(tile8,9)],

nl,write('Goal state:'),nl,

write(Goal),nl,nl,

solve(Init,Goal,Plan).

solve(State, Goal, Plan):-

solve(State, Goal, [], Plan).

%Determines whether Current and Destination tiles are a valid move.

is\_movable(X1,Y1) :- (1 is X1 - Y1) ; (-1 is X1 - Y1) ; (3 is X1 - Y1) ; (-3 is X1 - Y1).

solve(State, Goal, Plan, Plan):-

is\_subset(Goal, State), nl,

write\_sol(Plan).

solve(State, Goal, Sofar, Plan):-

act(Action, Preconditions, Delete, Add),

is\_subset(Preconditions, State),

\+ member(Action, Sofar),

delete\_list(Delete, State, Remainder),

append(Add, Remainder, NewState),

solve(NewState, Goal, [Action|Sofar], Plan).

act(move(X,Y,Z),

[at(X,Y), at(empty,Z), is\_movable(Y,Z)],

[at(X,Y), at(empty,Z)],

[at(X,Z), at(empty,Y)]).

is\_subset([H|T], Set):-

member(H, Set),

is\_subset(T, Set).

is\_subset([], \_).

% Remove all elements of 1st list from second to create third.

delete\_list([H|T], Curstate, Newstate):-

remove(H, Curstate, Remainder),

delete\_list(T, Remainder, Newstate).

delete\_list([], Curstate, Curstate).

remove(X, [X|T], T).

remove(X, [H|T], [H|R]):-

remove(X, T, R).

write\_sol([]).

write\_sol([H|T]):-

write\_sol(T),

write(H), nl.

append([H|T], L1, [H|L2]):-

append(T, L1, L2).

append([], L, L).

member(X, [X|\_]).

member(X, [\_|T]):-

member(X, T).

%-----------------------Output Queries---------------------------------->

?- test(Plan).

Initial state:

at(tile7,9)

at(tile1,8)

at(tile5,7)

at(tile6,6)

at(tile2,5)

at(empty,4)

at(tile8,3)

at(tile3,2)

at(tile4,1)

Goal state:

[at(tile1,1),at(tile2,2),at(tile3,3),at(tile4,4),at(empty,5),at(tile5,6),at(tile6,7),at(tile7,8),at(tile8,9)]

false.

**Exp-14 – TRAVELLING SALES PERSON PROBLEM**

**Algorithm:**

1. Define the set of cities to be visited and their co-ordinates.
2. Generate all possible permutation of the cities.
3. Initialize a variable to store all the distance(short) found so far and set it for large value.
4. For each permutation.

* Calculate the total distance of the route by summing the distance between the cities.
* If the total distance, update the shortest distance and store.

1. After all finding the shortest distance and the corresponding route represent the solution to the traveling salesman problem.

**Program:**

edge(a, b, 3).

edge(a, c, 4).

edge(a, d, 2).

edge(a, e, 7).

edge(b, c, 4).

edge(b, d, 6).

edge(b, e, 3).

edge(c, d, 5).

edge(c, e, 8).

edge(d, e, 6).

edge(b, a, 3).

edge(c, a, 4).

edge(d, a, 2).

edge(e, a, 7).

edge(c, b, 4).

edge(d, b, 6).

edge(e, b, 3).

edge(d, c, 5).

edge(e, c, 8).

edge(e, d, 6).

edge(a, h, 2).

edge(h, d, 1).

len([], 0).

len([H|T], N):- len(T, X), N is X+1 .

best\_path(Visited, Total):- path(a, a, Visited, Total).

path(Start, Fin, Visited, Total) :- path(Start, Fin, [Start], Visited, 0, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, StopLoc, Distance), NewCostn is Costn + Distance, \+ member(StopLoc, CurrentLoc),

path(StopLoc, Fin, [StopLoc|CurrentLoc], Visited, NewCostn, Total).

path(Start, Fin, CurrentLoc, Visited, Costn, Total) :-

edge(Start, Fin, Distance), reverse([Fin|CurrentLoc], Visited), len(Visited, Q),

(Q\=7 -> Total is 100000; Total is Costn + Distance).

shortest\_path(Path):-setof(Cost-Path, best\_path(Path,Cost), Holder),pick(Holder,Path).

best(Cost-Holder,Bcost-\_,Cost-Holder):- Cost<Bcost,!.

best(\_,X,X).

pick([Cost-Holder|R],X):- pick(R,Bcost-Bholder),best(Cost-Holder,Bcost-Bholder,X),!.

pick([X],X).

\*\*\*\*\*\*\*\*\*\*\*\*\*\*OUTPUT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TORUN:

run the gprolog compiler

?-["filename"].

?-shortest\_path(Path).

returns: Path- Distance -[Path]

DATA SET:

edge(a, b, 3). edge(a, c, 4). edge(a, d, 2). edge(a, e, 7). edge(b, c, 4). edge(b, d, 6). edge(b, e, 3). edge(c, d, 5). edge(c, e, 8). edge(d, e, 6). edge(b, a, 3). edge(c, a, 4). edge(d, a, 2). edge(e, a, 7). edge(c, b, 4). edge(d, b, 6). edge(e, b, 3). edge(d, c, 5). edge(e, c, 8). edge(e, d, 6). edge(a, h, 2). edge(h, d, 1).

Run:

| ?- [prolog].

yes | ?- shortest\_path(Path).

Path = 20-[a,h,d,e,b,c,a]

yes