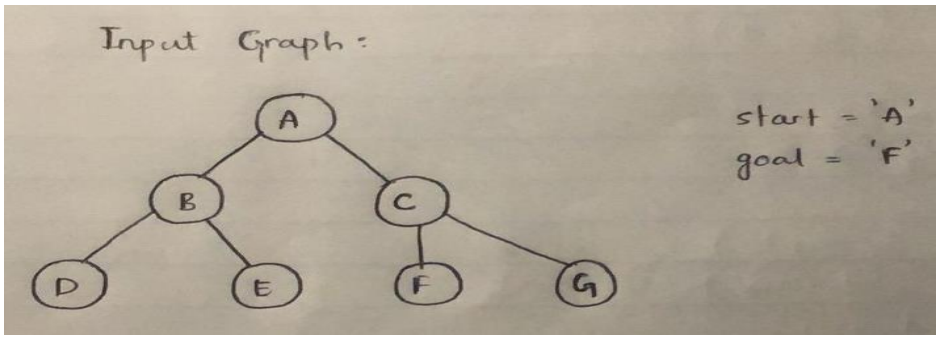
Termwork 1

Code:

```
from collections import defaultdict

graph = defaultdict(list)

def addEdge(u, v):
    graph[u].append(v)

def dfs(start, goal, depth):
    print(start, end=" ")
    if start == goal:
        return True
    if depth <= 0:
        return False
    for i in graph[start]:
        if dfs(i, goal, depth - 1):
            return True
    return False

def dfid(start, goal, maxDepth):
    print("Start node: ", start, "Goal node: ", goal)
    for i in range(maxDepth):
        print("\nDFID at level : ", i + 1)
        print("Path Taken : ", end=' ')
        isPathFound = dfs(start, goal, i)
    if isPathFound:
        print("\nGoal node found!")
        return
    else:
        print("\nGoal node not found!")

goal = defaultdict(list)

addEdge('A', 'B')
addEdge('A', 'C')
addEdge('A', 'D')
addEdge('B', 'E')
addEdge('B', 'F')
addEdge('E', 'I')
addEdge('E', 'J')
addEdge('D', 'G')
addEdge('D', 'H')
addEdge('G', 'K')
addEdge('G', 'L')
dfid('A', 'L', 4)
```

Sample Input:



Sample Output:

```
C:\Users\Lab2\Desktop\aiml>python tw1.py

DFS at level :  1
Path :  A ->
DFS at level :  2
Path :  A -> B -> C ->
DFS at level :  3
Path :  A -> B -> D -> E -> C -> G ->
DFS at level :  4
Path :  A -> B -> D -> E -> F ->
Goal node found!
```

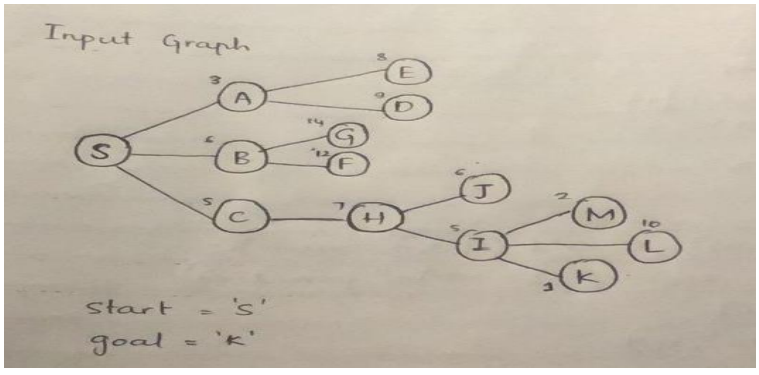
Termwork 2

Code:

```
SuccList ={ 'S':[['A',3],['B',6],['C',5]], 'A':[['E',8],['D',9]],'B':[['G',14],['F',12]], 'C':[['H',7]], 'H':[['J',6],['I',5]],'I':
[['M',2],['L',10],['K',1]]} #Graph(Tree) List

Start= input("Enter Source node >> ").upper()

Goal= input('Enter Goal node >> ').upper()

Closed = list()

SUCCESS = True

FAILURE = False

State = FAILURE

def GOALTEST(N):

   if N == Goal:

      return True

   else:

      return False

def MOVEGEN(N):

   New_list=list()

   if N in SuccList.keys():

       New_list=SuccList[N]

   return New_list

def APPEND(L1,L2):

   New_list=list(L1)+list(L2)

   return New_list

def SORT(L):

   L.sort(key = lambda x: x[1])

   return L

def BestFirstSearch():

   OPEN=[[Start,5]]

   CLOSED=list()

   global State

   global Closed

   i=1

   while (len(OPEN) != 0) and (State != SUCCESS):

      print("\n<<<<<<<<<<---({})--->>>>>>>>>>\n".format(i))

      N= OPEN[0]
```

```
        print("N=",N)
        del OPEN[0] #delete the node we picked
        if GOALTEST(N[0])==True:
            State = SUCCESS
            CLOSED = APPEND(CLOSED,[N])
            print("CLOSED=",CLOSED)
        else:
            CLOSED = APPEND(CLOSED,[N])
            print("CLOSED=",CLOSED)
            CHILD = MOVEGEN(N[0])
            print("CHILD=",CHILD)
            for val in OPEN:
                if val in CHILD:
                    CHILD.remove(val)
            for val in CLOSED:
                if val in CHILD:
                    CHILD.remove(val)
            OPEN = APPEND(CHILD,OPEN) #append movegen elements to OPEN
            print("Unsorted OPEN=",OPEN)
            SORT(OPEN)
            print("Sorted OPEN=",OPEN)
            Closed=CLOSED
            i+=1
    return State
result=BestFirstSearch()
print("Best First Search Path >>>> {} <<<{}>>>".format(Closed, result))
```

Sample Input:



Sample Output:

```
C:\Users\Lab2\Desktop\aiml>python tw2.py
Enter Source node >> S
Enter Goal node >> k

<<<<<<<<<<----(1)---->>>>>>>>>>

N:  ['S', 5]
CLOSED:  [['S', 5]]
CHILD:  [['A', 3], ['B', 6], ['C', 5]]
Unsorted OPEN:  [['A', 3], ['B', 6], ['C', 5]]
Sorted OPEN= [['A', 3], ['C', 5], ['B', 6]]

<<<<<<<<<<----(2)---->>>>>>>>>>

N:  ['A', 3]
CLOSED:  [['S', 5], ['A', 3]]
CHILD:  [['E', 8], ['D', 9]]
Unsorted OPEN:  [['E', 8], ['D', 9], ['C', 5], ['B', 6]]
Sorted OPEN= [['C', 5], ['B', 6], ['E', 8], ['D', 9]]

<<<<<<<<<<----(3)---->>>>>>>>>>

N:  ['C', 5]
CLOSED:  [['S', 5], ['A', 3], ['C', 5]]
CHILD:  [['H', 7]]
Unsorted OPEN:  [['H', 7], ['B', 6], ['E', 8], ['D', 9]]
Sorted OPEN= [['B', 6], ['H', 7], ['E', 8], ['D', 9]]
```

Termwork 3

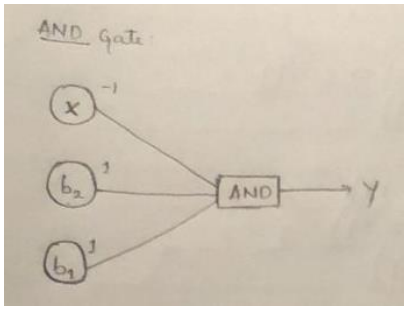
Code:

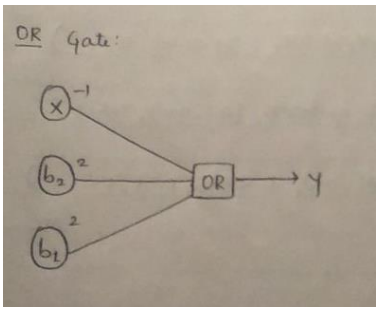
```
def OR():
    w1=0;w2=0;a=0.2;t=0
    X=[[0,0],[0,1],[1,0],[1,1]]
    Y=[0,1,1,1]
    while(True):
        Out=[]
        count = 0
        for i in X:
            step=(w1*i[0]+w2*i[1])
            if step<=t:
                O=0
                if O==Y[count]:
                    Out.append(O)
                    count+=1
                else:
                    w1=w1+(a*i[0]*1)
                    w2=w2+(a*i[1]*1)
                    print(w1,w2)
            else:
                O=1
                if O==Y[count]:
                    Out.append(O)
                    count+=1
                else:
                    w1 = w1 + (a * i[0] * 0)
                    w2 = w2 + (a * i[1] * 0)
                    print(w1,w2)
        print("------->")
        if Out[0:]==Y[0:]:
            print("Final Output of OR ::\n")
            print("Weights: w1={} and w2={} >>>> {}".format(w1,w2,Out))
            break
OR()
```

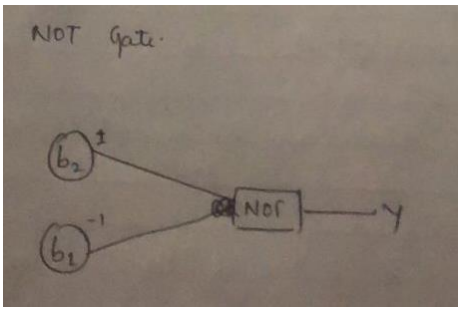
```
#AND
def AND():
    w1=0;w2=0;a=0.2;t=1
    X=[[0,0],[0,1],[1,0],[1,1]]
    Y=[0,0,0,1]
    while(True):
        Out=[]
        count = 0
        for i in X:
            step=(w1*i[0]+w2*i[1])
            if step<=t:
                O=0
                if O==Y[count]:
                    Out.append(O)
                    count+=1
                    print(w1,w2,Out)
                else:
                    print('Weights changed to..')
                    w1=w1+(a*i[0]*1)
                    w2=w2+(a*i[1]*1)
                    print("w1={} w2={}".format(round(w1,2),round(w2,2)))
                    print("------->")
            else:
                O=1
                if O==Y[count]:
                    Out.append(O)
                    count+=1
                    print(w1,w2,Out)
                else:
                    print("Weights Changed to..")
                    w1 = w1 + (a * i[0] * 0)
                    w2 = w2 + (a * i[1] * 0)
                    print("w1={} w2={}".format(round(w1,2),round(w2,2)))
                    print("------->")
        if Out[0:]==Y[0:]:
```

```
        print("\nFinal Output of AND::\n")
        print("Weights: w1={} and w2={} >>>> {}".format(round(w1,2),round(w2,2),Out))
        break
AND()
#NOT
def NOT():
   X=[0,1]
   Y=[1,0]
   weight=-1
   bias=1;Out=[]
   for i in X:
      j=weight*i+bias
      Out.append(j)
   print("\nFinal Output of NOT ::\n")
   for i in X:
      print("NOT Gate {}-->{}".format(X[i],Out[i]))
NOT()
```

Sample Input:







Sample Output:

```
Weights: w1=0.2 and w2=0.2 >>>> [0, 1, 1, 1]
0 0 [0]
0 0 [0, 0]
0 0 [0, 0, 0]
Weights changed to..
w1=0.2 w2=0.2
Final Output of NOT ::
NOT Gate 0-->1
NOT Gate 1-->0
```

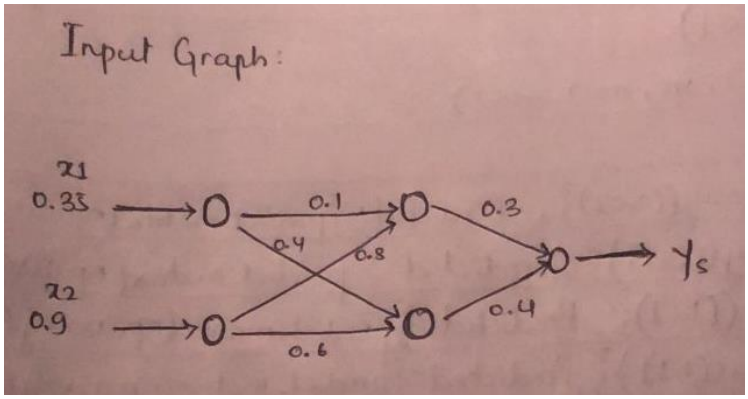
Termwork 4

Code:

```
import numpy as np

#np.random.seed(0)

def sigmoid (x):

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

def sigmoid_derivative(x):

    return x * (1 - x)

#Input datasets

inputs = np.array([[0,0],[0,1],[1,0],[1,1]])

expected_output = np.array([[0],[1],[1],[0]])

epochs = 10000

lr = 0.5

inputLayerNeurons, hiddenLayerNeurons, outputLayerNeurons = 2,2,1

#Random weights and bias initialization

hidden_weights = np.random.uniform(size=(inputLayerNeurons,hiddenLayerNeurons))
hidden_bias =np.random.uniform(size=(1,hiddenLayerNeurons))
output_weights = np.random.uniform(size=(hiddenLayerNeurons,outputLayerNeurons))
output_bias = np.random.uniform(size=(1,outputLayerNeurons))

print("Initial hidden weights: ",end='')
print(*hidden_weights)
print("Initial hidden biases: ",end='')
print(*hidden_bias)
print("Initial output weights: ",end='')
print(*output_weights)
print("Initial output biases: ",end='')
print(*output_bias)

#Training algorithm

for _ in range(epochs):

#Forward Propagation

    hidden_layer_activation = np.dot(inputs,hidden_weights)
    hidden_layer_activation += hidden_bias
    hidden_layer_output = sigmoid(hidden_layer_activation)
    output_layer_activation =np.dot(hidden_layer_output,output_weights)
    output_layer_activation += output_bias
    predicted_output = sigmoid(output_layer_activation)

#Backpropagation

    error = expected_output - predicted_output
    d_predicted_output = error * sigmoid_derivative(predicted_output)
    error_hidden_layer = d_predicted_output.dot(output_weights.T)
    d_hidden_layer = error_hidden_layer * sigmoid_derivative(hidden_layer_output)

#Updating Weights and Biases
```

```
    output_weights +=hidden_layer_output.T.dot(d_predicted_output) * lr
    output_bias += np.sum(d_predicted_output,axis=0,keepdims=True)* lr
    hidden_weights += inputs.T.dot(d_hidden_layer) * lr
    hidden_bias += np.sum(d_hidden_layer,axis=0,keepdims=True) *lr

print("Final hidden weights: ",end='')
print(*hidden_weights)
print("Final hidden bias: ",end='')
print(*hidden_bias)
print("Final output weights: ",end='')
print(*output_weights)
print("Final output bias: ",end='')
print(*output_bias)
print("\nOutput from neural network after epochs :" +str(epochs) )
print(*predicted_output)
```

Sample Input:



Sample Output:

Output: After epoch 1

Initial hidden weights: [0.57739373 0.99731969] [0.23542431 0.76683569]

Initial hidden biases: [0.37407026 0.18114935]

Initial output weights: [0.0218607] [0.07345263]

Initial output biases: [0.04597635]

Final hidden weights: [0.57739202 0.9975624] [0.23545824 0.76717274]

Final hidden bias: [0.37401636 0.18106946]

Final output weights: [0.01274522] [0.06705193]

Final output bias: [0.03174794]

Output from neural network after epochs :1

[0.52472264] [0.52823899] [0.52944441] [0.53170537]

Termwork 5:

Code:

```
x1=[1,1]
x2=[1,-1]
x3=[-1,1]
x4=[-1,-1]

xilist=[x1,x2,x3,x4]

y=[1,-1,-1,-1]

w1=w2=bw=0

b=1

def heb_learn():

    global w1,w2,bw
    print("dw1\tdw2\tdb\tw1\tw2\tb")

    i=0

    for xi in xilist:

        dw1=xi[0]*y[i]
        dw2=xi[1]*y[i]
        db=y[i]
        w1=w1+dw1
        w2=w2+dw2
        bw+=db
        print(dw1,dw2,db,w1,w2,bw,sep='\t')

        i+=1

print("Learning...")

heb_learn()

print("Learning completed")
print("Output of AND gate using obtained w1,w2,bw:")
print("x1\tx2\ty")

for xi in xilist:

    print(xi[0],xi[1],1 if w1*xi[0]+w2*xi[1]+b*bw>0 else -1,sep='\t')

print("Final weights are: w1="+str(w1) +" w2=" +str(w2))
```

Sample Output:

```
C:\Users\Lab2\Desktop\aiml>python tw5.py
dw1     dw2     y       tw1     tw2     b
1       1       1       1       1       1
1       -1      -1      0       2       0
-1      1       -1      1       1       -1
-1      -1      -1      2       2       -2
Learning complete
Output using predicted bias
x1      x2      y
1       1       1
1       -1      -1
-1      1       -1
-1      -1      -1
```

Termwork 6:

Code:

```
import csv
a = []
with open('forTw6.csv', 'r') as csvfile:
   next(csvfile)
   for row in csv.reader(csvfile):
      a.append(row)
   print(a)
print("\nThe total number of training instances are : ",len(a))
num_attribute = len(a[0])-1
print("\nThe initial hypothesis is : ")
hypothesis = ['0']*num_attribute
print(hypothesis)
for i in range(0, len(a)):
   if a[i][num_attribute] == 'yes':
      print ("\nInstance ", i+1, "is", a[i], " and is Positive Instance")
      for j in range(0, num_attribute):
         if hypothesis[j] == '0' or hypothesis[j] == a[i][j]:
            hypothesis[j] = a[i][j]
         else:
            hypothesis[j] = '?'
      print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
   if a[i][num_attribute] == 'no':
      print ("\nInstance ", i+1, "is", a[i], " and is Negative Instance Hence Ignored")
      print("The hypothesis for the training instance", i+1, " is: " , hypothesis, "\n")
print("\nThe Maximally specific hypothesis for the training instance is ", hypothesis)
```

sample input: 'forTw6.csv'

sample output:

[['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']]

The total number of training instances are: 4

The initial hypothesis is:

['0', '0', '0', '0', '0', '0']

Instance 1 is ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'yes'] and is Positive Instance

The hypothesis for the training instance 1 is: ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']

Termwork 7:

```
import pandas as pd
import numpy as np
import sklearn
from sklearn import linear_model
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Load the dataset and print it

boston = fetch_california_housing()

# Create dataframes out of the dataset
# data --> independent variables / x values
# target --> dependent variable / y value
# feature_names --> column names / features

df_x = pd.DataFrame(boston.data, columns=boston.feature_names)

df_y = pd.DataFrame(boston.target)

# Generate descriptive statistics such as mean, count, etc.

print(df_x.describe())

# Initialise the linear regression model

reg = linear_model.LinearRegression()

# Split the dataset into 67% training and 33% testing data

x_train, x_test, y_train, y_test = train_test_split(df_x, df_y, test_size=0.33, random_state=42)

# Train the model with the training data

reg.fit(x_train, y_train)

# Print the coefficients for each feature

print("\nCOEFFICIENTS", reg.coef_)

# Run the model on the test data and print the predictions

y_pred = reg.predict(x_test)

print("\nPREDICTIONS : ", y_pred)

# Print the actual / target values

print("\nACTUAL DATA : ", y_test)

# Calculate the error

# using np.mean

print("\nNP MEAN : ", np.mean(y_test - y_pred)**2)

# using mean_squared_errro from sklearn.metrics

print("\nMEAN SQUARED ERROR :", mean_squared_error(y_test, y_pred))
```

sample output:

```
[[-1.28749718e-01  3.78232228e-02  5.82109233e-02  3.23866812e+00
  -1.61698120e+01  3.90205116e+00 -1.28507825e-02 -1.42222430e+00
   2.34853915e-01 -8.21331947e-03 -9.28722459e-01  1.17695921e-02
  -5.47566338e-01]]
[[28.53469469]…
dtype: float64        20.724023437339696
```

Termwork 8

Code:

```
import numpy as np
import pandas as pd
emails = pd.read_csv('forTw8')
#emails[:10]
def process_email(text):
    text = text.lower()
    return list(set(text.split()))
emails['words'] = emails['text'].apply(process_email)
num_emails = len(emails)
num_spam = sum(emails['spam'])
print("Number of emails:", num_emails)
print("Number of spam emails:", num_spam)
print()
# Calculating the prior probability that an email is spam
print("Probability of spam:", num_spam/num_emails)
print()

model = {}
# Training process
for index, email in emails.iterrows():
    for word in email['words']:
        if word not in model:
            model[word] = {'spam': 1, 'ham': 1}
        if word in model:
            if email['spam']:
                model[word]['spam'] += 1
            else:
                model[word]['ham'] += 1
def predict_bayes(word):
    word = word.lower()
    num_spam_with_word = model[word]['spam']
    num_ham_with_word = model[word]['ham']
    return 1.0*num_spam_with_word/(num_spam_with_word + num_ham_with_word)
print("Prediction using Bayes for word sale",predict_bayes("sale"))
print("Prediction using Bayes for word lottery",predict_bayes("lottery"))
print()

def predict_naive_bayes(email):
    total = len(emails)
    num_spam = sum(emails['spam'])
    num_ham = total - num_spam
    email = email.lower()
    words = set(email.split())
    spams = [1.0]
    hams = [1.0]
    for word in words:
        if word in model:
            spams.append(model[word]['spam']/num_spam*total)
            hams.append(model[word]['ham']/num_ham*total)
```

```
    prod_spams = np.compat.long(np.prod(spams)*num_spam)
    prod_hams = np.compat.long(np.prod(hams)*num_ham)
    return prod_spams/(prod_spams + prod_hams)

print("Prediction using NaiveBayes for word lottery sale",predict_naive_bayes("lottery sale"))
print("Prediction using NaiveBayes for word asdfgh",predict_naive_bayes("asdfgh"))
print("Prediction using NaiveBayes ",predict_naive_bayes('Hi mom how are you'))
```

sample output:

```
Number of emails: 5728
Number of spam emails: 1368

Probability of spam: 0.2388268156424581

Prediction using Bayes for word sale 0.48148148148148145
Prediction using Bayes for word lottery 0.9

Prediction using NaiveBayes for word lottery sale 0.9638144992048691
Prediction using NaiveBayes for word asdfgh 0.2388268156424581
Prediction using NaiveBayes  0.12554358867164464
```