Pabna University of Science And Technology

Faculty of Engineering & Technology

Department of Information and Communication Engineering

PRACTICAL LAB REPORT

COURSE CODE: *ICE-3102*

Course Title: Artificial Intelligence and Robotics Sessional.

SUBMITTED BY:	**SUBMITTED TO:**
Name: **MD RAHATUL RABBI**	**Dr. Md. Omar Faruk**
Roll: **190609**	**Associate Professor**
Session: **2018-2019**	**Tarun Debnath**
Third year First Semester	**Lecturer**
Department of ICE,	**Department of ICE.**
Pabna university of Science and Technology	**Pabna University of Science and Technology Pabna, Bangladesh.**
	Teacher's Signature

INDEX

Experiment No: 01

Name of the Experiment: Write a Program to Implement Tower of Hanoi for n Disk.

Objectives:

- ✓ To understand the concept of Tower of Hanoi problem.
- ✓ To understand and describe how to implement a program to solve the problem for n disks.

Theory:

Tower of Hanoi : The Tower of Hanoi (also called The problem of Benares Temple or Tower of Brahma or Lucas' Tower and sometimes pluralized as Towers, or simply pyramid puzzle) is a mathematical game or puzzle consisting of three rods and a number of disks of various diameters, which can slide onto any rod. The puzzle begins with the disks stacked on one rod in order of decreasing size, the smallest at the top, thus approximating a conical shape. The objective of the puzzle is to move the entire stack to the last rod, obeying the following rules:

1. Only one disk may be moved at a time.
2. Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
3. No disk may be placed on top of a disk that is smaller than it.

With 3 disks, the puzzle can be solved in 7 moves. The minimal number of moves required to solve a Tower of Hanoi puzzle is $2^n - 1$, where n is the number of disks.

The solution to the Tower of Hanoi problem can be achieved recursively. The basic idea is to move the top n-1 disks from the source pole to the auxiliary pole, then move the nth disk from the source pole to the destination pole, and finally move the n-1 disks from the auxiliary pole to the destination pole. This process can be repeated recursively until all the disks have been moved to the destination pole.

The idea is to use the helper node to reach the destination using recursion. Below is the pattern for this problem:

1. Shift ‘N-1’ disks from ‘A’ to ‘B’ using C.
2. Shift last disks from ‘A’ to ‘B’ using C.

Shift ‘N-1’ disks from ‘B’ to ‘C’, using A

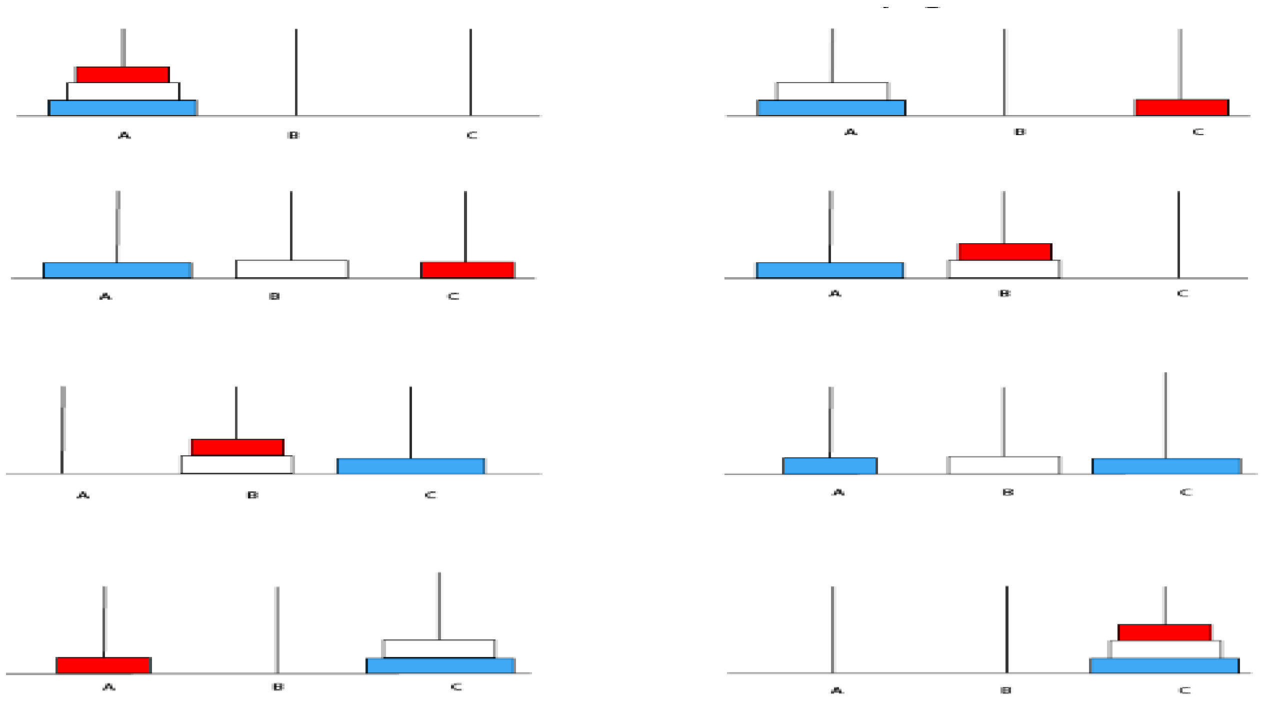


Figure-1.1: Tower of Hanoi for three disk.

Source Code in C:

```
#include <stdio.h>
 void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod) {
    if (n == 1) {
       printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
    return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
 }

 int main()
 {
    int n; // Number of disks
    printf("Enter Number of disks: ");
    scanf("%d", &n);
    towerOfHanoi(n, 'P', 'R', 'Q'); // P, Q and R are names of rods
    return 0;
 }
```

Input:

Enter Number of disks: 3

Output:

```
Move disk 1 from rod P to rod R

Move disk 2 from rod P to rod Q

Move disk 1 from rod R to rod Q

Move disk 3 from rod P to rod R

Move disk 1 from rod Q to rod P

Move disk 2 from rod Q to rod R

Move disk 1 from rod P to rod R
```

Experiment No: 02

Name of the Experiment: Write a Program to Implement Breadth First Search Algorithm.

Objectives:

- ✓ To understand the concept of Breadth First Search Algorithm (BFS).
- ✓ To implement Breadth First Search Algorithm in C programming language.
- ✓ To demonstrate the usage of Breadth First Search Algorithm on a graph.

Theory:

Breadth First Search Algorithm: Breadth First Search Algorithm (BFS) is a traversing algorithm where you should start traversing from a selected node (source or starting node) and traverse the graph layer wise thus exploring the neighbor nodes (nodes which are directly connected to source node). You must then move towards the next-level neighbor nodes.

As the name BFS suggests, you are required to traverse the graph breadthwise as follows:

1. First move horizontally and visit all the nodes of the current layer
2. Move to the next layer

Consider the following diagram

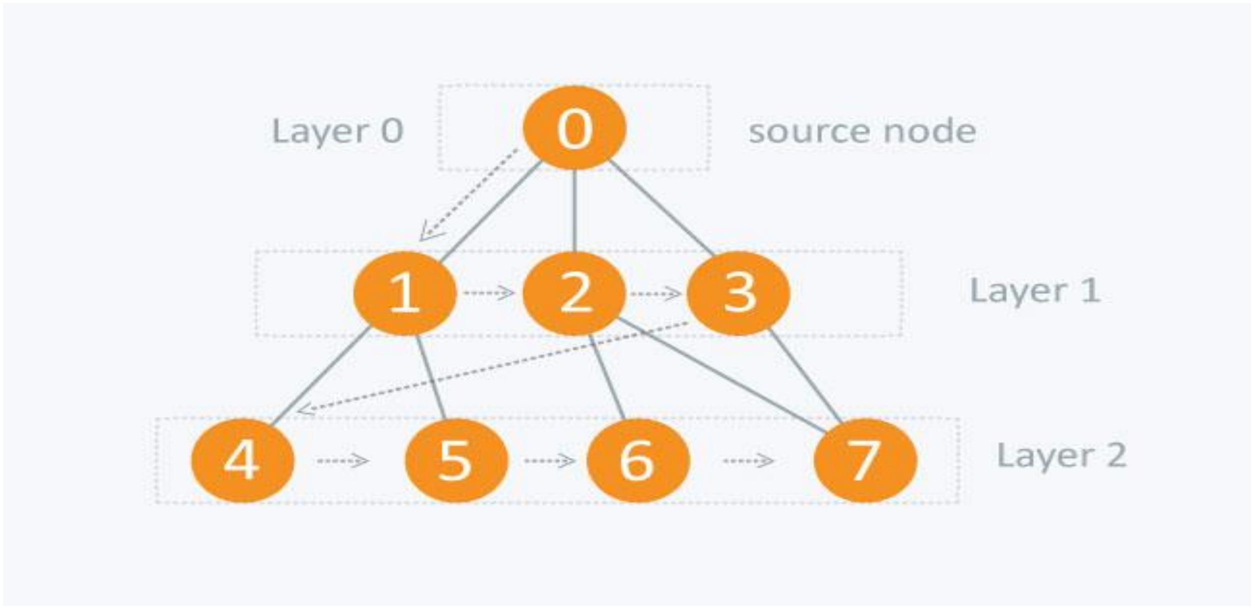


Figure-2.1: Example of BFS with 7 nodes.

The distance between the nodes in layer 1 is comparatively lesser than the distance between the nodes in layer 2. Therefore, in BFS, you must traverse all the nodes in layer 1 before you move to the nodes in layer 2.

Source Code:

```
#include<stdio.h>
int a[20][20], q[20], visited[20], n, i, j, f = 0, r = -1;
 void bfs(int v) {
    for(i = 1; i <= n; i++)
    if(a[v][i] && !visited[i])
       q[++r] = i;
    if(f <= r) {
        visited[q[f]] = 1;
        bfs(q[f++]);
    }
 }
 void main() {
    int vtx;
    printf("\nEnter the number of Vertices :");
    scanf("%d", &n);
    for(i=1; i <= n; i++) {
        q[i] = 0;
        visited[i] = 0;
    }
    printf("\nEnter graph data in matrix form:\n");
    for(i=1; i<=n; i++) {
           printf("Enter value of %d Row : \n",i);
        for(j=1;j<=n;j++) {
           scanf("%d", &a[i][j]);
        }
    }
    printf("\nEnter the starting vertex:");
    scanf("%d", &vtx);
    bfs(vtx);
    printf("\nThe node which are reachable are:\n");
    for(i=1; i <= n; i++) {
        if(visited[i])
           printf("%d\t", i);
        else {
            printf("\nNot all nodes are reachable So, BFS is not possible.");
            break;
        }
    }
 }
```

Input:

Enter the number of Vertices: 4

Enter values of 1 Row : 1 0 1 0

Enter values of 2 Row : 0 1 0 1

Enter values of 3 Row : 1 1 1 0

Enter values of 4 Row : 1 0 0 1

Enter the starting vertex: 2

Output:

The nodes which are reachable are:

```
                1        2           3          4
```

Experiment No: 03

Name of the Experiment: **Write a Program to Implement Depth First Search Algorithm.**

Objectives:

- ✓ To understand the concept of Depth First Search Algorithm (DFS).
- ✓ To implement Depth First Search Algorithm in C programming language.
- ✓ To demonstrate the usage of Depth First Search Algorithm on a graph.

Theory:

Depth First Search Algorithm: The Depth First Search Algorithm **(DFS)** is a recursive algorithm that uses the idea of backtracking. It involves exhaustive searches of all the nodes by going ahead, if possible, else by backtracking.

Here, the word backtrack means that when you are moving forward and there are no more nodes along the current path, you move backwards on the same path to find nodes to traverse. All the nodes will be visited on the current path till all the unvisited nodes have been traversed after which the next path will be selected. This recursive nature of DFS can be implemented using stacks.

The basic idea is as follows:

Pick a starting node and push all its adjacent nodes into a stack. Pop a node from stack to select the next node to visit and push all its adjacent nodes into a stack. Repeat this process until the stack is empty. However, ensure that the nodes that are visited are marked. This will prevent you from visiting the same node more than once. If you do not mark the nodes that are visited and you visit the same node more than once, you may end up in an infinite loop.

The following image shows how DFS works.

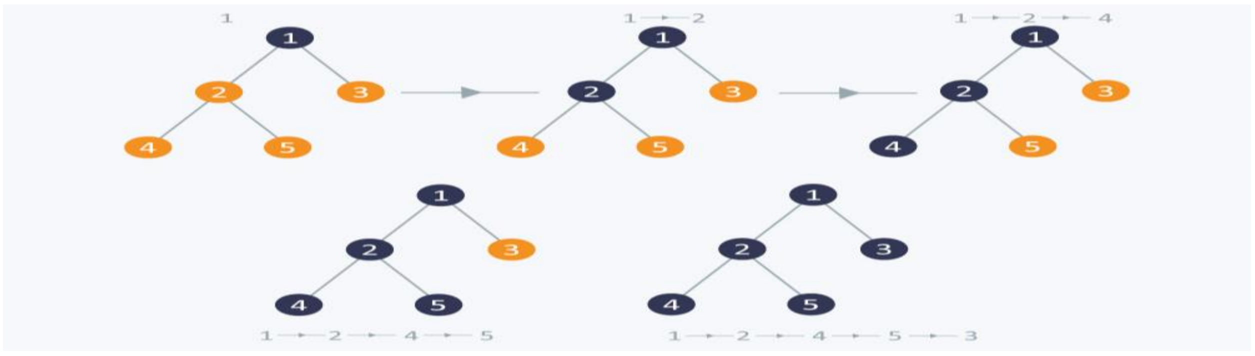


Figure-3.1: Example of DFS with 5 nodes.

Source Code:

```
#include<stdio.h>
 void DFS(int);
 int G[10][10],visited[10],n;
 void DFS(int i){
     int j;
     visited[i]=1;
     for(j=0;j<n;j++)
        if(!visited[j]&&G[i][j]==1)
            DFS(j);
 }
 void main()
 {
     int i,j,vtx;
     printf("Enter Number of Vertices's: ");
     scanf("%d",&n);
     printf("\nEnter Adjacency Matrix of The Graph:\n");
     for(i=0;i<n;i++){
         printf("Enter value of %d Row : \n",i);
         for(j=0;j<n;j++)
            scanf("%d",&G[i][j]);
     }
     for(i=0;i<n;i++)
        visited[i]=0;

    printf("\nEnter the starting vertex:"); //1
     scanf("%d", &vtx);
     DFS(vtx);
     printf("\nThe node which are reachable are:\n");
     for(i=0; i < n; i++) {
         if(visited[i])
            printf("%d\t", i);
         else {
             printf("\nNot all nodes are reachable So, DFS is not possible.");
             break;
         }
     }
     getch();
}
```

Input:

```
Enter Number of Vertices : 4


Enter Adjacency Matrix of The Graph:

Enter values of 1 Row:  0 1 0 1

Enter values of 2 Row:  1 0 1 0

Enter values of 3 Row:  1 0 0 1

Enter value of 4 Row:    1 1 1 0

Enter the starting vertex: 1
```

Output:

```
 The nodes which are reachable are:

                    0         1          2        3
```

Experiment No: 04

Name of the Experiment: Write a Program to Implement Travelling Salesman Problem.

Objectives:

- ✓ To understand the concept of Travelling Salesman Problem.
- ✓ To implement Travelling Salesman Problem in C programming language.
- ✓ To demonstrate the usage of Travelling Salesman Problem on a graph.

Theory:

The Travelling salesman problem: Travelling salesman problem is the most notorious computational problem of finding the shortest possible route that visits every city exactly once and returns to the starting city. One of the popular algorithms to solve this problem is the brute-force algorithm which generates all possible permutations of the cities and finds the shortest route. However, this approach is not feasible for large number of cities as the number of possible permutations increases factorial with the number of cities.

Another algorithm to solve this problem is the dynamic programming approach using the concept of sub problems. In this approach, we first solve the sub problems of finding the shortest route between two cities, and then gradually build up to solve the problem for all cities.

Here, we know that Hamiltonian Tour exists (because the graph is complete) and in fact, many such tours exist, the problem is to find a minimum weight Hamiltonian Cycle.

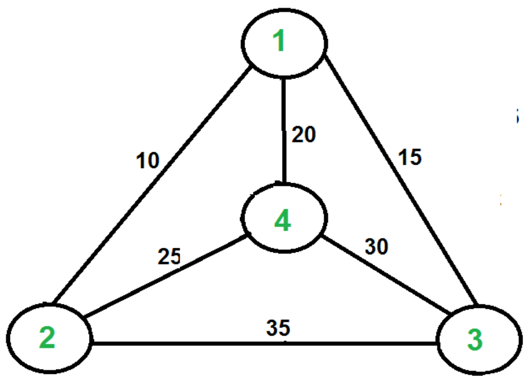


Figure-4.1: Example of TSP with 4 nodes.

For example, consider the graph shown in the figure on the right side. A TSP tour in the graph is 1-2-4-3-1. The cost of the tour is 10+25+30+15 which is 80. The problem is a famous NP-hard problem. There is no polynomial-time know solution for this problem. The following are different solutions for the traveling salesman problem.

Source Code:

```
#include<stdio.h>
 int ary[10][10],completed[10],n,cost=0;
 void mincost(int city){
     int i,ncity;
     completed[city]=1;
     printf("%d--->>",city+1);
     ncity=least(city);
     if(ncity==999){
         ncity=0;
         printf("%d",ncity+1);
         cost+=ary[city][ncity];
         return;
    }
    mincost(ncity);
 }
 int least(int c) {
     int i,nc=999;
     int min=999,kmin;
     for(i=0;i < n;i++){
         if((ary[c][i]!=0)&&(completed[i]==0))
             if(ary[c][i]+ary[i][c] < min){
                 min=ary[i][0]+ary[c][i];
                 kmin=ary[c][i];
                 nc=i;
             }
     }
     if(min!=999)
        cost+=kmin;
     return nc;
 }

 int main(){
     int i,j,vtx;
     printf("Enter the number of nodes: ");
     scanf("%d",&n);
     printf("\nEnter the Cost Matrix\n");
```

```
   for(i=0;i < n;i++){
        printf("Enter values of %d Row : ",i+1);
        for( j=0;j < n;j++)
           scanf("%d",&ary[i][j]);
        completed[i]=0;
    }
    printf("\nEnter the starting vertex:"); // 1
    scanf("%d", &vtx);
    printf("\n\nThe Path is:\n");
    mincost(vtx);
    printf("\n\nMinimum cost is %d  ",cost);
    return 0;
}
```

Input:

Enter the number of nodes: 4

Enter the Cost Matrix:

Enter values of 1 Row : 0 3 4 6

Enter values of 2 Row : 1 0 3 4

Enter values of 3 Row : 4 3 0 4

Enter values of 4 Row : 2 2 3 0

Enter the starting vertex: 0

Output:

The Path is:

```
                    1--->>2--->>4--->>3--->>1
```

Minimum cost is 14

Experiment No: 05

Name of the Experiment:

- **(i) Create and Load Different Datasets in Python.**
- **(ii) Write a Python Program to Compute Mean, Median, Mode, Variance and Standard Deviation using Datasets.**

Objectives:

- ✓ To Create and Load Different Datasets in Python.
- ✓ To understand the concept of Mean, Median, Mode, Variance and Standard deviation using Datasets.

(i) Theory:

Create and Load Datasets: When we started our data science journey the very first library that got our way was **pandas** and the very first function was **read_csv(). But do we have only a CSV file to read? No right!** This article will let you know about other methods one can use to read and access the dataset.

1. **Pandas Library:** Pandas is a powerful library that provides a data structure which is used to manipulate and analyze data. It provides a wide variety of methods to read and store data from different data sources such as CSV, Excel, JSON, HTML, and SQL.

2. **Numpy Library:** Numpy is a powerful library used for scientific computing. It provides a data structure which is used for scientific computing. Numpy is used for array manipulation, linear algebra, Fourier transforms and random number generation.

3. **Scikit-Learn Library:** Scikit-Learn is a powerful library for machine learning. It provides a wide variety of algorithms for supervised and unsupervised learning. It also provides tools for data preprocessing, feature selection, model selection, and model evaluation.

(i) Source Code in Python:

```
import pandas as pd
data= {'Name':
['RAHIMA','RAHATUL','NAYEEM','SUHAN','IMRAN','SHADHIN','BAPPY'],
    'Age': [22,23,24,25,22,23,21],
    'Gender': ['F','M','M','M','M','M','M'],
    'Marks': [85,90,'NaN',70,82,'NaN',86]}
df =pd.DataFrame(data)
df
```

Output:

	Name	Age	Gender	Marks
0	RAHIMA	22	F	85
1	RAHATUL	23	M	90
2	NAYEEM	24	M	NaN
3	SUHAN	25	M	70
4	IMRAN	22	M	82
5	SHADHIN	23	M	NaN
6	BAPPY	21	M	86

(ii) Theory:

1. **Mean:** The mean is the average of a given set of numbers. It is calculated by adding up all the numbers and dividing by the number of values in the set. To calculate the **mean** of a given data set, we use the following formula

$$\text{Mean } (\bar{x}) = \frac{\sum x}{N}$$

2. **Median:** The median is the middle value in a given set of numbers. It is found by arranging the numbers in ascending or descending order and finding the middle value. To calculate the **median** of a given data set, we use the following formula

$$\text{Median} = (\frac{n+1}{2})^{th} \text{ observation}$$

If an even number of terms are given in the data set, we use the following formula,

$$\text{Median} = \frac{(\frac{n}{2})^{th}\ observation + (\frac{n}{2}+1)^{th}\ observation}{2}$$

3. **Mode:** The mode is the most frequently occurring value in a given set of numbers. It is found by looking for the number that appears most often in the set.

$$\text{Mode} = l + (\frac{f_1 - f_0}{2f_1 - f_0 - f_2}) \times h$$

4. **Variance:** The variance is a measure of how spreads out the values in a given set of numbers are. It is calculated by finding the difference between each number and the mean, squaring each of these differences, and then finding the average of all the squared differences. To calculate the **variance** of a given data set, we use the following formula.

$$\text{Variance}(\sigma^2) = \frac{\sum(x_i - \mu)^2}{N}$$

5. **Standard Deviation:** The standard deviation is a measure of how much the values in a given set of numbers differ from the mean. It is calculated by taking the square root of the variance. To calculate the **standard deviation** of a given data set, we use the following formula

$$\text{Standard deviation}(\sigma) = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

(ii) Source Code in Python:

```
# Importing Packages
import numpy as np
# Creating Dataset
dataset = [3, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50]

# Calculating Mean
 mean = np.mean(dataset)
print("Mean of the data set is:", mean)


# Calculating Median
median = np.median(dataset)
 print("Median of the data set is:", median)

# Calculating Mode
from scipy import stats
mode = stats.mode(dataset)
print("Mode of the data set is:", mode[0][0])

# Calculating Variance
variance = np.var(dataset)
print("Variance of the data set is:", variance)
```

```
# Calculating Standard Deviation
std_dev = np.std(dataset)
print("Standard Deviation of the data set is:", std_dev)
```

Output:

Mean of the data set is: 27.5

Median of the data set is: 27.5

Mode of the data set is: 3

Variance of the data set is: 208.33333333333334

Standard Deviation of the data set is: 14.422205101855956

Experiment No: 06

Name of the Experiment: Write a Python Program to Implement Simple Linear Regression and Plot the Graph.

Objectives:

- ✓ To understand the concept of Simple Linear Regression.
- ✓ To implement Simple Linear Regression Problem in python programming language.
- ✓ To demonstrate the usage of Simple Linear Regression Problem on a graph.

Theory:

Linear Regression: Linear regression is a statistical technique to describe relationships between dependent variables with a number of independent variables. Here we discuss the basic concepts of linear regression as well as its application within Python.

In order to give an understanding of the basics of the concept of linear regression, we begin with the most basic form of linear regression, i.e., "Simple linear regression".

Simple Linear Regression: Simple linear regression (SLR) is a method to predict a response using one feature. It is believed that both variables are linearly linked. Thus, we strive to find a linear equation that can predict an answer value(y) as precisely as possible in relation to features or the independently derived variable(x).

Let's consider a dataset in which we have a number of responses **y** per feature **x**:

X	10	11	12	13	14	15	16	17	18	19
Y	11	13	12	15	17	18	18	19	20	22

For simplification, we define:

x as **feature vector**, i.e., $x = [x_1, x_2, x_3, \ldots, x_n]$,

y as **response vector**, i.e., $y = [y_1, y_2, y_3 \ldots, y_n]$

For **n** observations (in above example, $n = 10$).

A scatter plot of the above dataset looks like: -

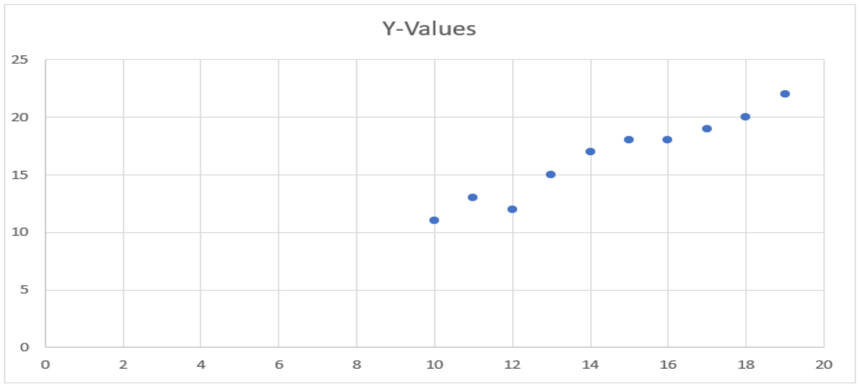


Figure-6.1: Scatter Plot of Sample Dataset.

Source Code in Python:

```
import numpy as nmp
import matplotlib.pyplot as mtplt
def estimate_coeff(p, q):
    n1 = nmp.size(p)        # To estimate the total number of points or observation.
    m_p = nmp.mean(p)  # To calculate the mean of a and b vector.
    m_q = nmp.mean(q)
# To calculate the cross deviation and deviation about a
    SS_pq = nmp.sum(q * p) - n1 * m_q * m_p
    SS_pp = nmp.sum(p * p) - n1 * m_p * m_p
    b_1 = SS_pq / SS_pp  # To calculate the regression coefficients
.   b_0 = m_q - b_1 * m_p
.   return (b_0, b_1)
.
. def plot_regression_line(p, q, b):
.   mtplt.scatter(p, q, color = "m",   # To plot the actual points as scatter plot
.           marker = "o", s = 30)
.   q_pred = b[0] + b[1] * p  # To calculate the predicted response vector
.   mtplt.plot(p, q_pred, color = "g")  # To plot the regression line
.
```

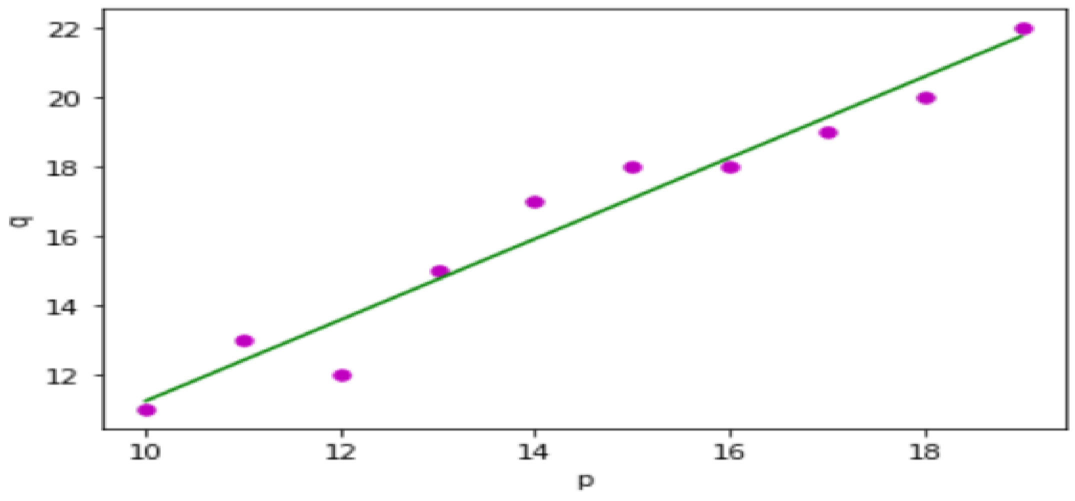
```
.      mtplt.xlabel('p')  # To put the labels
.     mtplt.ylabel('q')
.     mtplt.show()  # To define the function to show plot
. def main():
. # entering the observation points or data
.     p = np.array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
.     q = np.array([11, 13, 12, 15, 17, 18, 18, 19, 20, 22])
.     b = estimate_coeff(p, q) # To estimate the coefficients
.     print("Estimated coefficients are :\nb_0 = {} \
.         \nb_1 = {}".format(b[0], b[1]))
.     plot_regression_line(p, q, b)  To plot the regression line
. if __name__ == "__main__":
.     main()
```

Output:

Estimated coefficients are:

```
b_0 = -0.4606060606060609
b_1 = 1.1696969696969697
```

Graph:



Experiment No: 07

Name of the Experiment: Write a Python Program to Implement Find S Algorithm.

Objectives:

- ✓ To understand the concept of Find-S Algorithm.
- ✓ To implement Find-S Algorithm problem in python programming language.
- ✓ To understand how to read-write the dataset.

Theory:

The Find-S Algorithm: To understand the implementation, let us try to implement it to a smaller data set with a bunch of examples to decide if a person wants to go for a walk.

The Find-S algorithm follows the steps written below:

```
1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x
       For each attribute constraint $a_i$ in h
           If the constraint $a_i$ is satisfied by x
               Then do nothing
           Else replace $a_i$ in h by the next more general constraint that is satisfied by x

3. Output hypothesis h.
```

Now that we are done with the basic explanation of the Find-S algorithm, let us take a look at how it works.

The concept of this particular problem will be on what days do a person likes to go on walk.

Time	Weather	Temperature	Company	Humidity	Wind	Goes
Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Looking at the data set, we have six attributes and a final attribute that defines the positive or negative example. In this case, yes is a positive example, which means the person will go for a walk.

Working Algorithm:

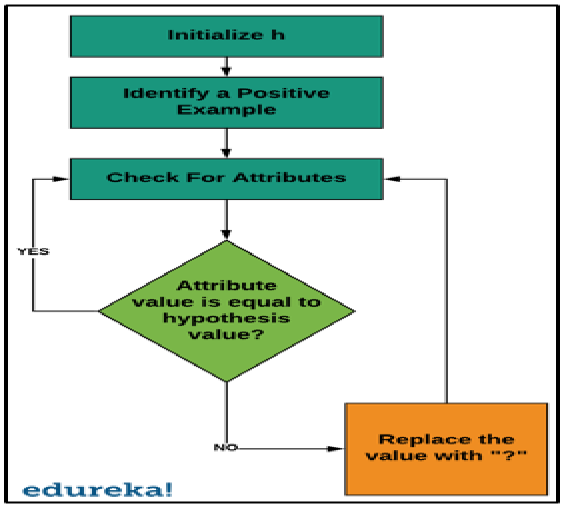

Initialize h
Identify a Positive Example
Check For Attributes
YES
Attribute value is equal to hypothesis value?
NO
Replace the value with "?"
edureka!

Figure-7.1: Working Algorithm of Find-S Algorithm.

Source Code in Python:

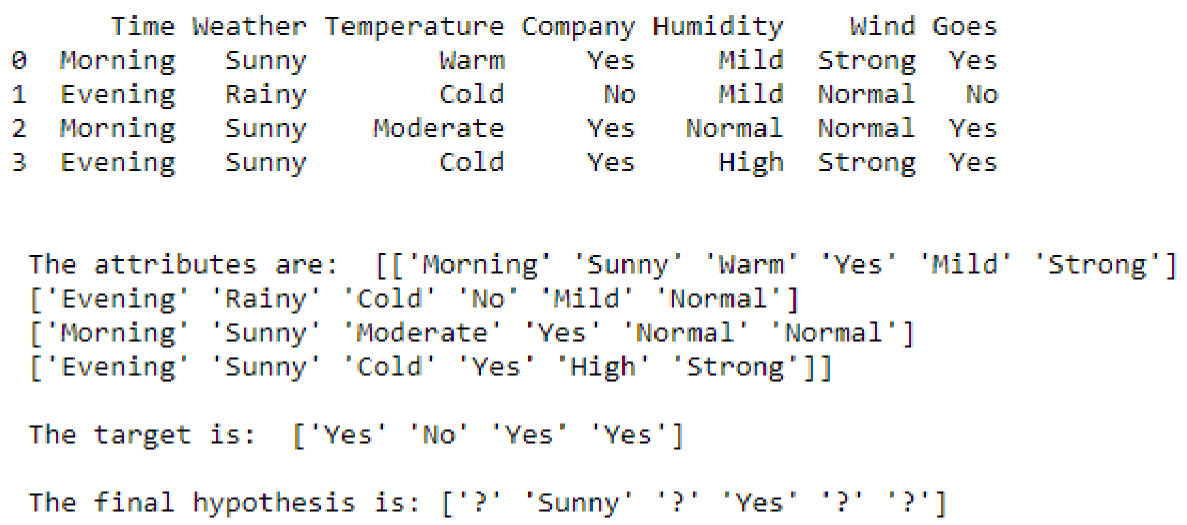
```
import pandas as pd
import numpy as np

#to read the data in the csv file
data = pd.read_csv("data.csv")
print(data,"n")
#making an array of all the attributes
d = np.array(data)[:,:-1]
print("n The attributes are: ",d)
#segragating the target that has positive and negative examples
target = np.array(data)[:,-1]
print("n The target is: ",target)
#training function to implement find-s algorithm
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
#obtaining the final hypothesis
print("n The final hypothesis is:",train(d,target))
```

Input:

Dataset:

Morning	Sunny	Warm	Yes	Mild	Strong	Yes
Evening	Rainy	Cold	No	Mild	Normal	No
Morning	Sunny	Moderate	Yes	Normal	Normal	Yes
Evening	Sunny	Cold	Yes	High	Strong	Yes

Output:



```
      Time Weather Temperature Company Humidity    Wind Goes
0  Morning   Sunny        Warm     Yes     Mild  Strong  Yes
1  Evening   Rainy        Cold      No     Mild  Normal   No
2  Morning   Sunny    Moderate     Yes   Normal  Normal  Yes
3  Evening   Sunny        Cold     Yes     High  Strong  Yes


 The attributes are:  [['Morning' 'Sunny' 'Warm' 'Yes' 'Mild' 'Strong']
 ['Evening' 'Rainy' 'Cold' 'No' 'Mild' 'Normal']
 ['Morning' 'Sunny' 'Moderate' 'Yes' 'Normal' 'Normal']
 ['Evening' 'Sunny' 'Cold' 'Yes' 'High' 'Strong']]

 The target is:  ['Yes' 'No' 'Yes' 'Yes']

 The final hypothesis is: ['?' 'Sunny' '?' 'Yes' '?' '?']
```

Experiment No: 08

Name of the Experiment: **Write a Python Program to Implement Support Vector Machine (SVM) Algorithm.**

Objectives:

- ✓ To understand support vector machine algorithm (SVM), a popular machine learning algorithm or classification.
- ✓ To learn how to implement SVM models in Python.
- ✓ To know the pros and cons of Support Vector Machines (SVM) and their different applications in machine learning (artificial intelligence).

Theory:

The Support Vector Machine: Support Vector Machine (SVM) is a supervised learning machine learning algorithm that can be used for both classification and regression challenges. However, it is mostly used in classification problems, such as text classification. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is the number of features you have), with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the optimal hyper-plane that differentiates the two classes very well (look at the below snapshot).

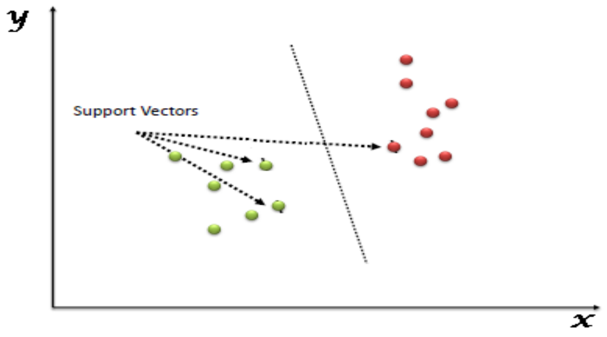


Figure-8.1: Example of SVM.

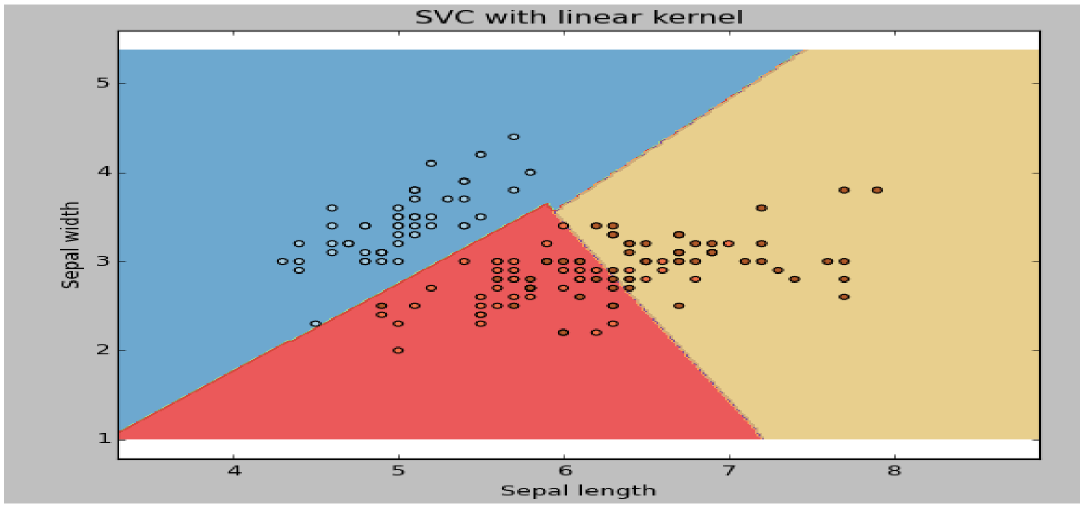
Support Vectors are simply the coordinates of individual observation, and a hyper-plane is a form of SVM visualization. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/line).

Source Code:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
# import some data to play with
```

```
iris = datasets.load_iris()
X = iris.data[:, :2] # we only take the first two features. We could
 # avoid this ugly slicing by using a two-dim dataset
y = iris.target
# we create an instance of SVM and fit out data. We do not scale our

# data since we want to plot the support vectors
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=1,gamma=0).fit(X, y)

# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
h = (x_max / x_min)/100
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
 np.arange(y_min, y_max, h))
plt.subplot(1, 1, 1)
Z = svc.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.Paired, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.xlim(xx.min(), xx.max())
plt.title('SVC with linear kernel')
plt.show()
```

Output:


SVC with linear kernel
Sepal width
Sepal length