Experiment no: 01

Name of the experiment: Write a program to implement Tower of Hanoi for n disk.

Objectives:

1. To learn about the mathematical puzzle of tower of Hanoi.
2. To learn how a recursive function works.

Theory:

The Tower of Hanoi is a classic recursive puzzle game consisting of three rods and n disks of different sizes which can slide onto any rod. The objective of the game is to move the entire stack to another rod, obeying the following rules:

1. Only one disk can 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 an empty rod.
3. No disk may be placed on top of a smaller disk.

The game was invented by the French mathematician Édouard Lucas in 1883 and is named after the city of Hanoi in Vietnam. The Towers of Hanoi puzzle is a classic example of recursion, since the solution is to decompose the problem of moving a stack of n disks into subproblems of moving a stack of n-1 disk, and so in succession, until the movements of the basic case are carried out for only one disc.

The game can be played with any number of disks, but the minimum number of moves required to solve the puzzle is 2^n-1, where n is the number of disks.

The Tower of Hanoi puzzle has applications in computer science, such as in algorithm design and optimization, and in artificial intelligence, such as in the design of search algorithms and problem-solving agents.

Source Code:

```
#include <stdio.h>
 // C recursive function to solve tower of hanoi puzzle
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
 {
7 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 = 2; // Number of disks
 towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
 return 0;
}
```

Output:

Move disk 1 from rod A to rod B

Move disk 2 from rod A to rod C

Move disk 1 from rod B to rod C

-----x-----

Experiment no: 02

Name of the experiment: Write a program to implement Breadth First Search algorithm.

Objectives:

1. To learn about Bread First Search algorithm.
2. To Find the shortest path.
3. To understand how BFS algorithm works.

Theory:

Breadth-first search (BFS) algorithm is a widely used graph traversal algorithm, which visits all vertices of the graph in first order, i.e. it visits all vertices that are the same distance from the starting vertex, then moves to the one located at a greater distance.

BFS can be implemented using either an adjacency matrix or an adjacency list. In an adjacency matrix, each row represents a vertex, and each column represents an edge. The value of the element at the i-th row and j-th column is 1 if there is an edge between vertices i and j, and 0 otherwise. In an adjacency list, each vertex has a list of its adjacent vertices.

BFS is a useful algorithm for exploring and searching graphs. It is particularly useful for finding the shortest path between two vertices in an unweighted graph.

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 v;
 printf("\n Enter the number of vertices:");
 scanf("%d", &n);

 for(i=1; i <= n; i++) {
 q[i] = 0;
 visited[i] = 0;
 }

 printf("\n Enter graph data in matrix form:\n");
 for(i=1; i<=n; i++) {
 for(j=1;j<=n;j++) {
 scanf("%d", &a[i][j]);
 }
 }

 printf("\n Enter the starting vertex:");
 scanf("%d", &v);
 bfs(v);
 printf("\n The node which are reachable are:\n");

 for(i=1; i <= n; i++) {
```

```
if(visited[i])
printf("%d\t", i);
else {
printf("\n Bfs is not possible. Not all nodes are reachable");
break;
}
}
}
```

Input and Output:

Enter the number of vertices:5

Enter graph data in matrix form:

1 0 1 0 0

0 1 1 1 0

1 0 0 1 1

1 1 1 0 0

0 1 0 1 0

Enter the starting vertex:1

The node which are reachable are:

1 2 3 4 5

-----x-----

Experiment no: 03

Name of the experiment: Write a program to implement Depth First Search algorithm.

Objectives:

1. To learn about Depth First Search algorithm.
2. To Find the shortest path using DFS.
3. To understand how DFS algorithm works.

Theory:

Depth-First Search (DFS) is a graph traversal algorithm that explores as far as possible along each branch before backtracking. In other words, DFS traverses the vertices of a graph depth-first, visiting all the vertices of a branch before moving to the next branch.

DFS can be implemented using either an adjacency matrix or an adjacency list. In an adjacency matrix, each row represents a vertex, and each column represents an edge. The value of the element at the i-th row and j-th column is 1 if there is an edge between vertices i and j, and 0 otherwise. In an adjacency list, each vertex has a list of its adjacent vertices.

DFS has several applications in computer science, including finding strongly connected components in a graph, topological sorting, and solving maze problems.

Source Code:

```
#include<stdio.h>
void DFS(int);
int G[10][10],visited[10],n;
void main()
{
int i,j;
printf("Enter number of vertices:");
scanf("%d",&n);
printf("\nEnter adjecency matrix of the graph:\n");
for(i=0;i<n;i++)
for(j=0;j<n;j++)
scanf("%d",&G[i][j]);
for(i=0;i<n;i++)
visited[i]=0;
DFS(0);
}
void DFS(int i)
{
int j;
printf("\n%d",i);
visited[i]=1;
for(j=0;j<n;j++)
if(!visited[j]&&G[i][j]==1)
```

```
 DFS(j);
 }
```

Input and Output:

Enter number of vertices:5

Enter adjecency matrix of the graph:

1 0 1 0 0

0 1 1 1 0

1 0 0 1 1

1 1 1 0 0

0 1 0 1 0

0

2

3

1

4

-----x-----

Experiment no: 04

Name of the experiment: Write a program to implement Travelling Salesman Problem.

Objectives:

1. To Find the shortest possible route.
2. To understand travelling salesman problem.
3. To minimize the total cost of traveling.

Theory:

The Traveling Salesman Problem (TSP) is a classic optimization problem in computer science and mathematics. It is a problem of finding the shortest possible route that a salesman can take to visit all the cities in a given list and return to the starting city. The problem is named after a hypothetical salesman who needs to travel to multiple cities to sell his products and wants to minimize his travel costs.

The problem is often expressed as a complete graph, where each city is a vertex, and the distance between two cities is the weight of the edge that connects them. The objective is to find a Hamiltonian cycle, i.e., a cycle that visits each vertex exactly once, and has the minimum possible total weight.

TSP has several real-world applications, such as planning delivery routes for logistics companies, optimizing traffic flow in cities, and designing integrated circuits. It is a challenging problem that has attracted the attention of researchers for several decades, and many techniques and algorithms have been proposed to solve it efficiently.

Source Code:

```
#include<stdio.h>
 int ary[10][10],completed[10],n,cost=0;
 void takeInput()
 {
 int i,j;

 printf("Enter the number of node: ");
 scanf("%d",&n);
 printf("\nEnter the Cost Matrix\n");
 for(i=0;i < n;i++)
 {
 for( j=0;j < n;j++)
 scanf("%d",&ary[i][j]);
 completed[i]=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()
 {
 takeInput();
 printf("\n\nThe Path is:\n");
 mincost(0); //passing 0 because starting vertex
 printf("\n\nMinimum cost is %d\n ",cost);
 return 0;
}
```

Input and Output:

Enter the number of node: 4

Enter the Cost Matrix

0 5 4 5

1 0 5 4

2 4 0 3

1 2 5 0

The Path is:

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

Minimum cost is 16

-----x-----

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:

1. To learn how to load dataset in python.
2. To learn how to compute Mean, Median, Mode, Variance and Standard Deviation using datasets.

Theory:

A dataset is a collection of data that is organized and stored in a specific format for ease of access, retrieval, and analysis. Datasets can come in various formats, such as a table, spreadsheet, file, or database, and can contain data in different types, such as numerical, text, categorical, or images. They are commonly used in machine learning, data mining, and statistical analysis to train models, test hypotheses, or derive insights from the data.

Mean: Mean is a measure of central tendency that represents the average value of a set of numbers. It is calculated by adding up all the numbers in the dataset and then dividing the total by the number of values in the dataset. The formula for calculating the mean is:

mean = (sum of all values) / (number of values)

Median: Median is a measure of central tendency that represents the middle value of a set of ordered numbers. It is calculated by arranging the values in the dataset in numerical order and identifying the middle value. If the dataset has an odd number of values, the median is the middle value. If the dataset has an even number of values, the median is the average of the two middle values.

Mode: the mode is a measure of central tendency that represents the most frequently occurring value in a dataset. It is the value that appears most often in a dataset.

Variance: variance is a measure of how spread out a set of data is from its mean (average) value. It quantifies the amount of variability or dispersion in a dataset.

Standard Deviation:

the standard deviation is a measure of the amount of variation or dispersion in a set of data. It is the square root of the variance.

Source Code:

```
import pandas as pd

data= {'Name': ['Mitu','Shahadat','Masud','Sourov','Mahi','Sumaiya'],

    'Age': [23,23,24,22,22,20],

    'Gender': ['F','M','M','M','F','F'],

    'Marks': [80,83,'NaN',90,92,'NaN']}

df =pd.DataFrame(data)

df

df = pd.read_csv('1.car driving risk analysis.csv')
```

speed	risk
250	95
100	20
300	98
110	60
240	88
115	40
50	8
230	85
190	45
260	91

Df

270	92
185	59

Input and Output:

Name	Age	Gender	Marks
Mitu	23	F	80
Shahadat	23	M	83
Masud	24	M	95
Sourov	22	M	90
Mahi	22	F	92
Sumaiya	20	F	90

Source Code:

```
import statistics as st

import pandas as pd

df=pd.read_csv('meanMedian.csv')

df

st.mean(df['data'])

st.median(df['data'])

st.mode(df['data'])

st.stdev(df['data'])

st.variance(df['data'])
```

Input and Output:

```
23.642857142857142
21.5
10
13.29897128963881
176.86263736263737
```

	data
0	10
1	18
2	20
3	10
4	20
5	23
6	34
7	32
8	23
9	10
10	23
11	45
12	10
13	53

-----x-----

Experiment no: 06

Name of the experiment: . Write a python program to implement simple linear regression and plot the graph.

Objectives:

1. To learn how to implement linear regression.
2. To learn how to plot a graph.

Theory:

Simple linear regression is a statistical method used to model the relationship between two quantitative variables by fitting a linear equation to the observed data. The goal of simple linear regression is to find the line of best fit that describes the relationship between the two variables.

In simple linear regression, one variable is considered as the independent variable or predictor variable, while the other variable is considered as the dependent variable or response variable. The line of best fit is calculated based on the values of the independent variable and the corresponding values of the dependent variable.

The equation for a simple linear regression model can be written as:

Y = a + bX

Where Y is the dependent variable, X is the independent variable, a is the intercept or the value of Y when X = 0, and b is the slope or the rate at which Y changes with respect to X

Input and Output:

Implement simple linear Regression and plot the graph

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
```

Out[4]:

	speed
0	300
1	250
2	100
3	300
4	110

```
In [5]: y.head()
```

```
Out[5]: 0    97
        1    95
        2    20
        3    98
        4    60
        Name: risk, dtype: int64
```

Visualization

```
In [7]: from sklearn.model_selection import train_test_split
        xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)
```

training section

60% For training and 40% for testing

```
In [8]: xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=.40,random_state=1)
```

```
xtrain
```

```
In [9]: xtest
```

Out[9]:

	speed
3	300
13	310
7	50
2	100
6	115
10	260
4	110

```
In [10]: ytrain
```

Out[10]:
```
1     95
14    18
0     97
15     2
9     45
8     85
12    59
11    92
5     88
Name: risk, dtype: int64
```

In [11]:
```
ytest
```

Out[11]:
```
3     98
13    93
7      8
2     20
6     40
10    91
4     60
Name: risk, dtype: int64
```

In [12]:
```
from sklearn.linear_model import LinearRegression
```

In [13]:
```
reg=LinearRegression() #create object
```

In [14]:
```
reg.fit(xtrain,ytrain)
```

Out[14]:
```
LinearRegression()
```

In [15]:
```
reg.predict(xtest)  #compare with ytest
```

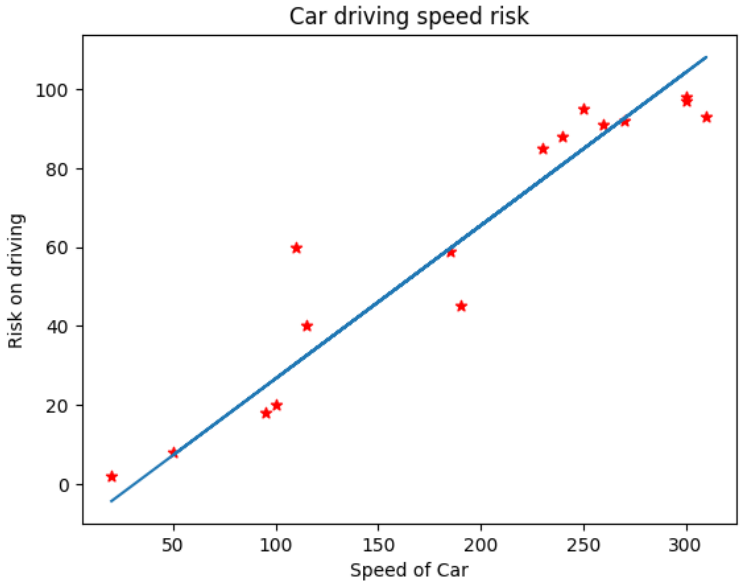
Out[15]:
```
array([104.19713512, 108.07511573,   7.24761988,  26.63752293,
        32.45449384,  88.68521268,  30.51550354])
```

In [16]:
```
ytest
```

Out[16]:
```
3     98
13    93
7      8
2     20
6     40
10    91
4     60
```

In [17]:
```
plt.scatter(df['speed'],df['risk'],marker='*',color='red')
plt.xlabel('Speed of Car')
plt.ylabel('Risk on driving')
plt.title('Car driving speed risk')
plt.plot(df.speed,reg.predict(df[['speed']]))
```

Out[17]:
```
[<matplotlib.lines.Line2D at 0x2479857ceb0>]
```




```
In [18]: reg.predict([[150]])

C:\Users\mituk\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\base.py:420: UserWarning: X does not have
valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(

Out[18]: array([46.02742598])

In [23]: reg.coef_

Out[23]: array([0.38891318])

In [24]: reg.intercept_

Out[24]: -15.62743726501703

In [25]: y=0.38891318*150-15.627437265017058
         y

Out[25]: 42.70953973498295
```

-----x-----

Experiment no: 07

Name of the experiment: Write a python program to implement Find S algorithm.

Objectives:

1. To learn how S algorithm works.
2. To find the most specific hypothesis that fits all positive instances in the training data.

Theory:

The S-algorithm, also known as the Candidate Elimination Algorithm, is a machine learning algorithm used for finding the most specific and most general hypotheses that fit the training data. The algorithm works by initializing the hypothesis space to the set of all possible hypotheses and then iteratively eliminating hypotheses that are inconsistent with the training data.

The Find-S algorithm is a very simple and efficient algorithm for learning a hypothesis from training data. It is particularly useful when the hypothesis space is small and the data is noise-free. However, it may not work well when the data is noisy or when the hypothesis space is large. In such cases, more sophisticated algorithms such as decision trees or neural networks may be more suitable.

Input and Output:

In [1]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('Book1.csv')
data
```

Out[1]:

	Unnamed: 0	sky	air temp	humidity	wind	water	forecast	enjoy sport
0	0	sunny	warm	normal	strong	warm	same	yes
1	1	sunny	cold	high	strong	warm	same	yes
2	2	rainy	cold	high	strong	warm	change	no
3	3	cloudy	warm	high	strong	cool	change	yes

In [2]:
```
# leave the last column
concept=np.array(data)[:,:-1]
# only access the last column
target=np.array(data)[:,-1]
print(concept)
target
```

```
[[0 'sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 [1 'sunny' 'cold' 'high' 'strong' 'warm' 'same']
 [2 'rainy' 'cold' 'high' 'strong' 'warm' 'change']
 [3 'cloudy' 'warm' 'high' 'strong' 'cool' 'change']]
```

Out[2]: array(['yes', 'yes', 'no', 'yes'], dtype=object)

In [3]:
```
def train(concept,target):
    for i,value in enumerate(target):
        if value.lower()=='yes':
            specific_h=concept[i].copy()
            break
    for i,value in enumerate(concept):
        if target[i].lower()=='yes':
            for x in range(len(specific_h)):
                if value[x]!=specific_h[x]:
                    specific_h[x]='?'
                else:
                    pass;
    return specific_h
```

In [4]:
```
result=train(concept,target)
print(result)
```

```
['?' '?' '?' '?' 'strong' '?' '?']
```

In [5]:
```
day=input("Enter 6 word to check:")
day=day.split()
check=True
```

```
Enter 6 word to check:sunny ok ? ? strong ok ?
```

In [6]:
```
for i in range(len(result)):
    if result[i]=='?'or result[i]==day[i]:
        check=True;
    else:
        check=False;
        break;
if check:
    print("Enjoy sport")
else:
    print("Not Possible")
```

```
Enjoy sport
```

Experiment no: 08

Name of the experiment: Write a python Program to implement Support Vector Machine (SVM) Algorithm.

Objectives:

1. To find the best hyperplane that can separate the training data.
2. To learn about SVM algorithm

Theory:

The Support Vector Machine (SVM) algorithm is a powerful machine learning algorithm used for classification and regression analysis. It is based on the concept of finding the best hyperplane that separates the data into different classes with maximum margin.

The SVM algorithm is particularly useful when the data is non-linearly separable or when the number of features is much larger than the number of samples. The algorithm can handle high-dimensional data very efficiently and avoid overfitting by regularizing the optimization problem. SVM is widely used in various fields, such as image classification, text classification, and bioinformatics.

Input and Output:

In [30]:
```
#https://www.mltut.com/svm-implementation-in-python-from-scratch/
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [31]:
```
dataset = pd.read_csv('Social_Network_Ads.csv')
dataset
```

Out[31]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
...	...	...	...
395	46	41000	1
396	51	23000	1
397	50	20000	1
398	36	33000	0
399	49	36000	1

400 rows × 3 columns

In [32]:
```
X = dataset.iloc[:, [0, 1]].values
y = dataset.iloc[:, 2].values
```

In [33]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 0)
```

In [34]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [35]:
```
from sklearn.svm import SVC
classifier = SVC(kernel = 'rbf', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[35]:
```
SVC(random_state=0)
```

In [36]:
```
y_pred = classifier.predict(X_test)
```

In [37]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

In [37]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test,y_pred)
```

```
[[64  4]
 [ 3 29]]
```

Out[37]: 0.93

In [38]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, step = 0.01),
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, step = 0.01))
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('SVM (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\mituk\AppData\Local\Temp\ipykernel_8160\2113774946.py:10: UserWarning: *c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*.  Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```