Practical No.1

Aim:- Implement Breadth First Search Algorithm for Romanian Map Problem.

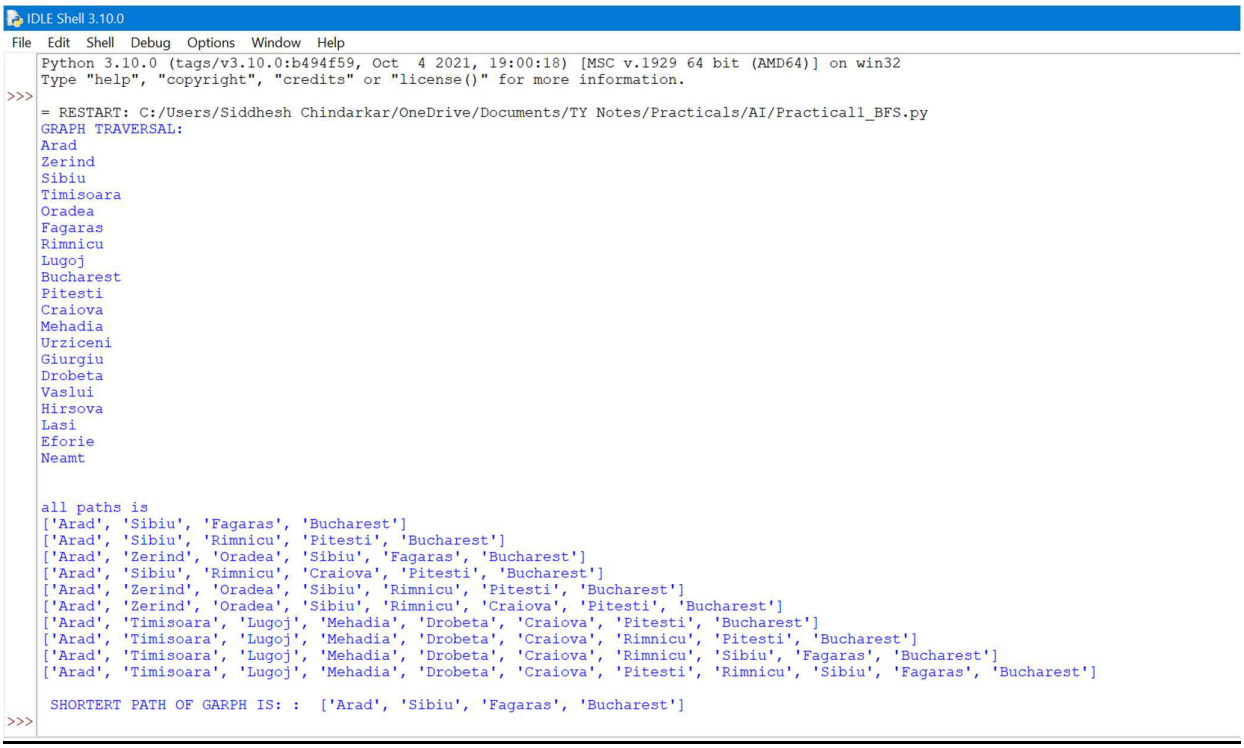
CODE:-

```
import collections
def bfs(graph,root):
    seen,queue=set([root]),collections.deque([root])
    while queue:
        vertex=queue.popleft()
        visit(vertex)
        for node in graph[vertex]:
            if node not in seen:
                seen.add(node)
                queue.append(node)

def allpath(st,end,gr):
    todo=[(st,[st])]
    while len(todo):
        node,path=todo.pop(0)
        for next_node in gr[node]:
            if next_node in path:
                continue
                print('Ideal solution')
            elif next_node==end:
                yield path + [next_node]
            else:
                todo.append((next_node,path + [next_node]))

def visit(n):
```

```
    print(n)


def bfs_shortest_path(grah, source, destination):


    checked=[]
    queue=[[source]]
    if source == destination:
        return "SOURCE IS DESTINATION :"
    while queue:
        path=queue.pop(0)
        node=path[-1]
        if node not in checked:
            neighbours = graph[node]


            for neighbour in neighbours:
                new_path=list(path)
                new_path.append(neighbour)
                queue.append(new_path)
                if neighbour == destination:
                    return new_path


            checked.append(node)
    return "PATH DOES NOT EXIST :"
graph={'Oradea': ['Zerind', 'Sibiu'],
       'Zerind':['Oradea', 'Arad'],
       'Arad': ['Zerind','Sibiu','Timisoara'],
       'Timisoara': ['Arad', 'Lugoj'],
       'Lugoj': ['Timisoara', 'Mehadia'],
       'Mehadia': ['Lugoj', 'Drobeta'],
```

```
    'Drobeta': ['Mehadia', 'Craiova'],
    'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],
    'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],
    'Sibiu': ['Oradea', 'Fagaras', 'Rimnicu','Arad'],
    'Fagaras': ['Sibiu', 'Bucharest'],
    'Rimnicu': ['Sibiu', 'Pitesti','Craiova'],
    'Bucharest': ['Urziceni','Giurgiu'],
    'Giurgiu': ['Bucharest'],
    'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],
    'Vaslui': ['Lasi', 'Urziceni'],
    'Lasi': ['Neamt', 'Vaslui'],
    'Neamt': ['Lasi'],
    'Hirsova': ['Urziceni', 'Eforie'],
    'Eforie': ['Hirsova']
    }

print("GRAPH TRAVERSAL: ")
bfs(graph,'Arad')
print('\n\nall paths is')
[print (x) for x in allpath ('Arad','Bucharest', graph)]
print("\n SHORTERT PATH OF GARPH IS: : ", bfs_shortest_path(graph, 'Arad','Bucharest'))
```

Output:-


IDLE Shell 3.10.0
File Edit Shell Debug Options Window Help
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/Siddhesh Chindarkar/OneDrive/Documents/TY Notes/Practicals/AI/Practical1_BFS.py
GRAPH TRAVERSAL:
Arad
Zerind
Sibiu
Timisoara
Oradea
Fagaras
Rimnicu
Lugoj
Bucharest
Pitesti
Craiova
Mehadia
Urziceni
Giurgiu
Drobeta
Vaslui
Hirsova
Lasi
Eforie
Neamt
all paths is
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Sibiu', 'Rimnicu', 'Pitesti', 'Bucharest']
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Sibiu', 'Rimnicu', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu', 'Pitesti', 'Bucharest']
['Arad', 'Zerind', 'Oradea', 'Sibiu', 'Rimnicu', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu', 'Pitesti', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Rimnicu', 'Sibiu', 'Fagaras', 'Bucharest']
['Arad', 'Timisoara', 'Lugoj', 'Mehadia', 'Drobeta', 'Craiova', 'Pitesti', 'Rimnicu', 'Sibiu', 'Fagaras', 'Bucharest']
SHORTERT PATH OF GARPH IS: : ['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
>>>

Practical No.2

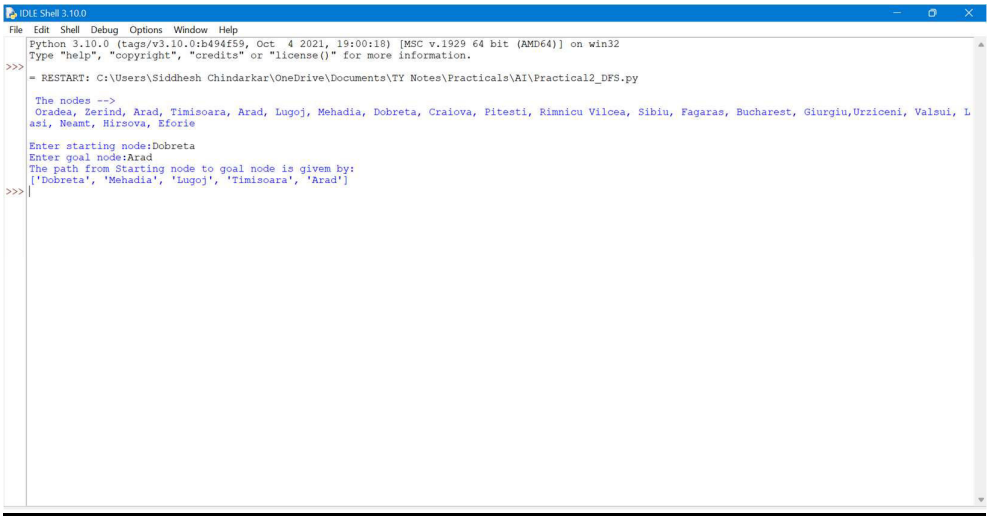
Aim:- Implement Depth First Search Algorithm for Romanian Map Problem.

CODE:-

```
graph={ 'Oradea': ['Zerind','Sibiu'],
        'Zerind': ['Oradea' , 'Arad'],
        'Arad' : ['Zerind' , 'Sibiu','Timisoara'],
        'Timisoara' : ['Arad' , 'Lugoj'],
        'Lugoj' : ['Timisoara' , 'Mehadia'],
        'Mehadia' : ['Lugoj' , 'Dobreta'],
        'Dobreta' : ['Mehadia' , 'Craiova'],
        'Craiova' : ['Dobreta' , 'Pitesti' , 'Rimnicu Vilcea'],
        'Pitesti' : ['Rimnicu Vilcea' , 'Craiova' , 'Bucharest'],
        'Rimnicu Vilcea' : ['Sibiu' , 'Pitesti' , 'Craiova' ],
        'Sibiu' : ['Oradea' , 'Rimnicu Vilcea' , 'Arad' , 'Fagaras'],
        'Fagaras' : ['Sibiu' , 'Bucharest'],
        'Bucharest' : ['Urziceni' , 'Giurgiu'],
        'Giurgiu' : ['Bucharest'],
        'Urziceni' : ['Bucharest' , 'Valsui' , 'Hirsova'],
        'Valsui' : ['Lasi' , 'Urziceni'],
        'Lasi' : ['Valsui' , 'Neamt'],
        'Neamt' : ['Lasi'],
        'Hirsova' : ['Urziceni' , 'Eforie'],
        'Eforie' : ['Hirsova'],
       }


def dfs(g, n, seen, d):
    if n not in seen:
        seen.append(n)
```

```
    for i in g[n]:
        if seen[-1] in d:
            break
        dfs(g, i, seen, d)
    return seen
print('\n The nodes --> \n Oradea, Zerind, Arad, Timisoara, Arad, Lugoj, Mehadia, Dobreta, Craiova, Pitesti, Rimnicu Vilcea, Sibiu, Fagaras, Bucharest, Giurgiu,Urziceni, Valsui, Lasi, Neamt, Hirsova, Eforie')



X=input("\nEnter starting node:")
Y=input("Enter goal node:")
print("The path from Starting node to goal node is givem by:")
print(dfs(graph, X, [], Y))
```

Output:-



Practical No.3

Aim:- Implement Depth Limited Search Algorithm for Romanian Map Problem.

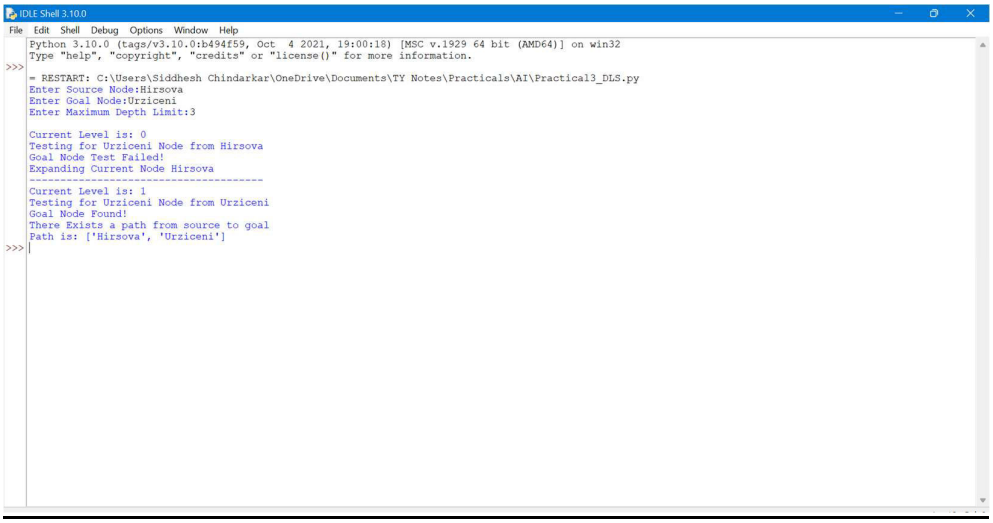
Code:-

```
graph={
 'Oradea': ['Zerind', 'Sibiu'],
 'Zerind':['Oradea', 'Arad'],
 'Arad': ['Zerind','Sibiu','Timisoara'],
 'Timisoara': ['Arad', 'Lugoj'],
 'Lugoj': ['Timisoara', 'Mehadia'],
 'Mehadia': ['Lugoj', 'Drobeta'],
 'Drobeta': ['Mehadia', 'Craiova'],
 'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],
 'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],
 'Sibiu': ['Oradea', 'Fagaras', 'Rimnicu','Arad'],
 'Fagaras': ['Sibiu', 'Bucharest'],
 'Rimnicu': ['Sibiu', 'Pitesti','Craiova'],
 'Bucharest': ['Urziceni','Giurgiu'],
 'Giurgiu': ['Bucharest'],
 'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],
 'Vaslui': ['Lasi', 'Urziceni'],
 'Lasi': ['Neamt', 'Vaslui'],
 'Neamt': ['Lasi'],
 'Hirsova': ['Urziceni', 'Eforie'],
 'Eforie': ['Hirsova']
 }


def dls(s,g,path,level,max_depth):
    print("Current Level is:",level)
```

```
    print("Testing for",g+" "+"Node from",s)
    path.append(s)
    e="Max Depth Limit Reached!"
    while True:
        if level>max_depth:
            print("Current Level Reaches Maximum Depth")
            return False
        break
    if s==g:
        print("Goal Node Found!")
        return path
    print("Goal Node Test Failed!")
    print("Expanding Current Node",s)
    print("---------------------------------------")
    for neighbor in graph[s]:
        if dls(neighbor,g,path,level+1,max_depth):
            return path
        path.pop()
        return False
    return False


s=input("Enter Source Node=")
g=input("Enter Goal Node=")
max_depth=int(input("Enter Maximum Depth Limit="))
print()
path=list()
output=dls(s,g,path,0,max_depth)
if (output):
```

```
    print("There Exists a path from source to goal")
    print("Path is:",path)
else:
    print("No Path From Source to Goal in given depth Limit")
```

Output:-



Practical No.4

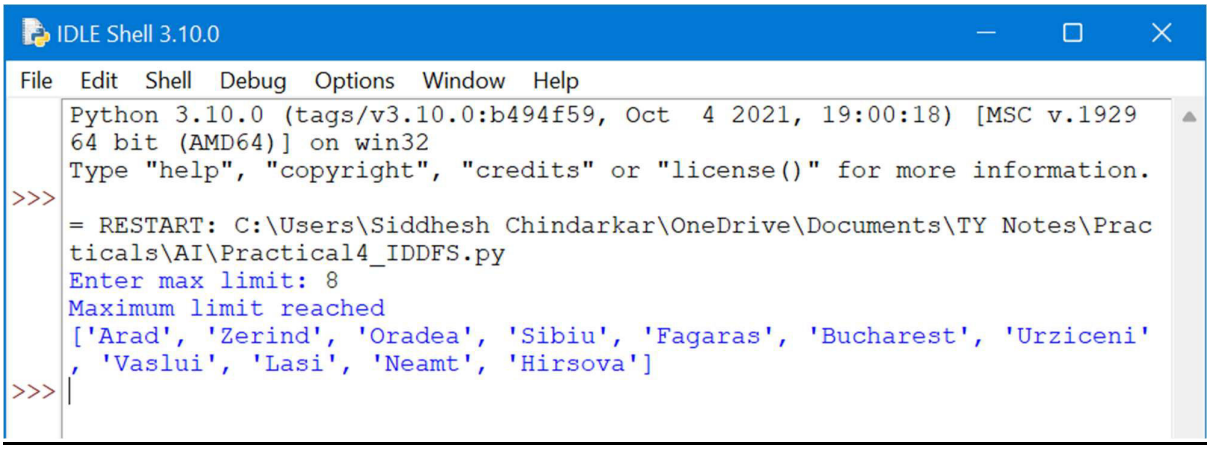
Aim:- Implement Iterative Deep Depth First Search for Romanian Map Problem.

Code:-

```
graph = {'Oradea': ['Zerind', 'Sibiu'],
        'Zerind': ['Oradea', 'Arad'],
        'Arad': ['Zerind', 'Sibiu', 'Timisoara'],
        'Timisoara': ['Arad', 'Lugoj'],
        'Lugoj': ['Timisoara', 'Mehadia'],
        'Mehadia': ['Lugoj', 'Drobeta'],
        'Drobeta': ['Mehadia', 'Craiova'],
        'Craiova': ['Drobeta', 'Rimnicu', 'Pitesti'],
        'Pitesti': ['Rimnicu', 'Craiova', 'Bucharest'],
        'Sibiu': ['Oradea', 'Fagaras', 'Rimnicu', 'Arad'],
        'Fagaras': ['Sibiu', 'Bucharest'],
        'Rimnicu': ['Sibiu', 'Pitesti', 'Craiova'],
        'Bucharest': ['Urziceni', 'Giurgiu'],
        'Giurgiu': ['Bucharest'],
        'Urziceni': ['Bucharest', 'Vaslui', 'Hirsova'],
        'Vaslui': ['Lasi', 'Urziceni'],
        'Lasi': ['Neamt', 'Vaslui'],
        'Neamt': ['Lasi'],
        'Hirsova': ['Urziceni', 'Eforie'],
        'Eforie': ['Hirsova']
        }


def iddfs(g, n, seen, dst, dep, lim):
   if n not in seen:
```

```
        seen.append(n)
        if dep <= lim:
            for i in g[n]:
                if seen[-1] is dst:
                    return seen
                iddfs(g, i, seen, dst, dep + 1, lim)
        else:
            print("Maximum limit reached")
    return None

print(iddfs(graph, 'Arad', [], 'Hirsova', 0, int(input("Enter max limit: "))))
```

Output:-



Practical No.5

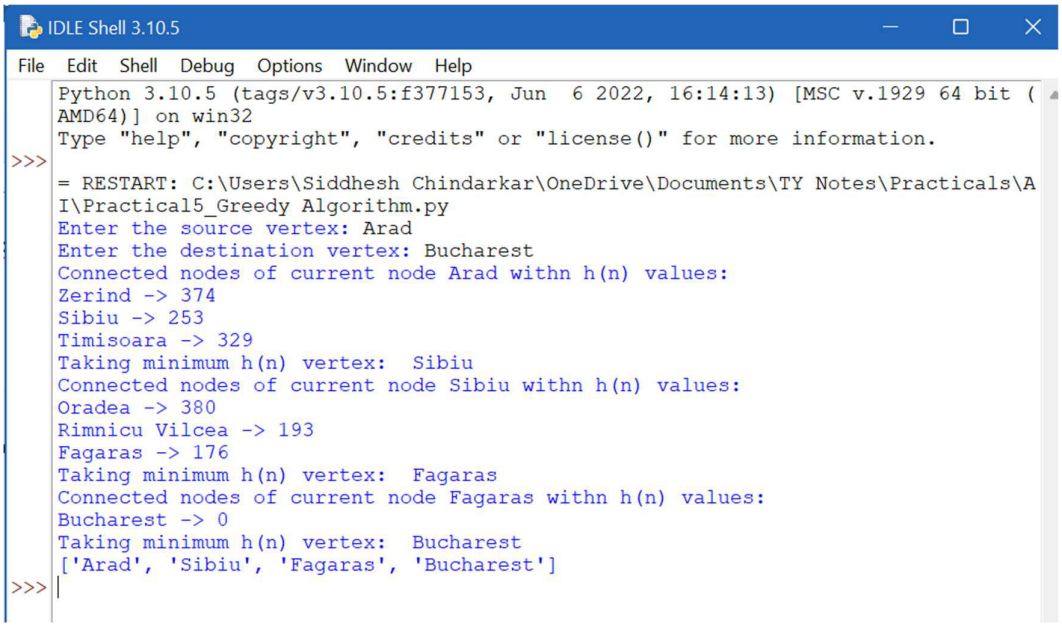
Aim:- Implement Recursive Best First Search for Romanian Map Problem.

Code:-

```
graph = {
   'Oradea': ({'Zerind':71, 'Sibiu':151},380),
   'Zerind': ({'Oradea':71, 'Arad':75},374),
   'Arad': ({'Zerind':75, 'Sibiu':140, 'Timisoara':118},366),
   'Timisoara': ({'Arad':118, 'Lugoj':111},329),
   'Lugoj': ({'Timisoara':111, 'Mehadia':70},244),
   'Mehadia': ({'Lugoj':70, 'Dobreta':75},241),
   'Sibiu': ({'Oradea':151, 'Rimnicu Vilcea':80, 'Arad':140, 'Fagaras':99},253),
   'Fagaras': ({'Sibiu':99, 'Bucharest':211},176),
   'Rimnicu Vilcea': ({'Sibiu':80, 'Pitesti':97, 'Craiova':146},193),
   'Bucharest': ({'Urziceni':85,'Fagaras':211,'Pitesti':101,'Giurgiu':90},0),
   'Dobreta': ({'Mehadia':75, 'Craiova':120},242),
   'Craiova': ({'Dobreta':120, 'Pitesti':138, 'Rimnicu Vilcea':97},160),
   'Pitesti': ({'Rimnicu Vilcea':97, 'Craiova':138, 'Bucharest':101},100),
   'Urziceni': ({'Bucharest':85, 'Valsui':142, 'Hirsova':98},80),
   'Giurgiu': ({'Bucharest':90},77),
   'Valsui': ({'Lasi':92, 'Urziceni':142},199),
   'Hirsova': ({'Urziceni':98, 'Eforie':86},151),
   'Lasi': ({'Valsui':92, 'Neamt':87},226),
   'Eforie': ({'Hirsova':86},161),
   'Neamt': ({'Lasi':87},234)
}


def greedy_search_rec(graph,prev,dst,path,q):
   # n:(h(n))
   print("Connected nodes of current node",prev,"withn h(n) values: ")
```

```
    for n in graph[prev][0]: #neighbour list prev=Arad, -> Z,S,T
        if n not in path:
            q[n] = graph[n][1]  #n=z  [1]=374
            print(n,"->",q[n])
    while q:
        mn = min(q,key=q.get)
        print("Taking minimum h(n) vertex: ",mn)
        #print(mn)
        if dst == mn:
            return path + [dst]
        #del q[mn]
        new_path = greedy_search_rec(graph,mn,dst,path + [mn],q)
        if new_path:
            return new_path
    return []

sourec = input("Enter the source vertex: ")
dest = input("Enter the destination vertex: ")
path = greedy_search_rec(graph,sourec,dest,[sourec],{})
if path:
    print(path)
else:
    print("Path not found")
```

Output:-


IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Siddhesh Chindarkar\OneDrive\Documents\TY Notes\Practicals\A
I\Practical5_Greedy Algorithm.py
Enter the source vertex: Arad
Enter the destination vertex: Bucharest
Connected nodes of current node Arad withn h(n) values:
Zerind -> 374
Sibiu -> 253
Timisoara -> 329
Taking minimum h(n) vertex: Sibiu
Connected nodes of current node Sibiu withn h(n) values:
Oradea -> 380
Rimnicu Vilcea -> 193
Fagaras -> 176
Taking minimum h(n) vertex: Fagaras
Connected nodes of current node Fagaras withn h(n) values:
Bucharest -> 0
Taking minimum h(n) vertex: Bucharest
['Arad', 'Sibiu', 'Fagaras', 'Bucharest']
>>>

Practical No.6

Aim:- Implement A* Algorithm for Romanian Map Problem.

Code:-

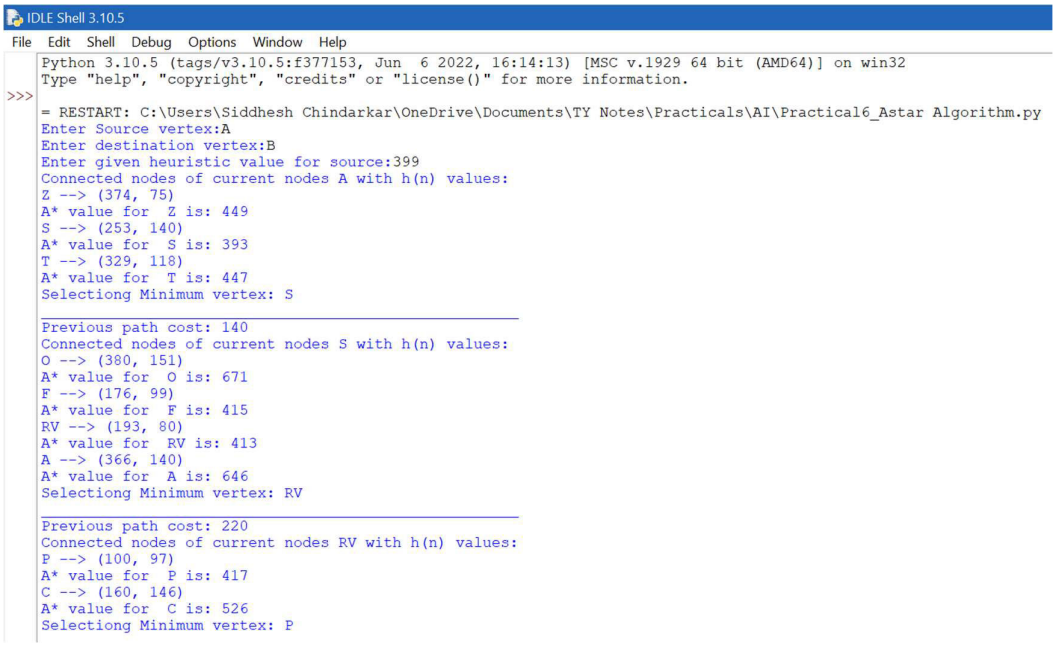
```
graph={
   "O":({"Z":71,"S":151},380),
   "Z":({"O":71,"A":75},374),
   "A":({"Z":75,"S":140,"T":118},366),
   "T":({"A":118,"L":111},329),
   "L":({"T":111,"M":70},244),
   "M":({"L":70,"D":75},241),
   "S":({"O":151,"F":99,"RV":80,"A":140},253),
   "F":({"S":99,"B":211},176),
   "RV":({"S":80,"P":97,"C":146},193),
   "B":({"F":211,"P":101,"U":85,"G":90},0),
   "P":({"RV":97,"C":138,"B":101},100),
   "C":({"RV":146,"P":138,"D":120},160),
   "D":({"M":75,"C":120},242),
   "U":({"B":85,"V":142,"":98},80),
   "G":({"B":90},77),
   "V":({"L":92,"U":142},199),
   "H":({"U":98,"E":86},151),
   "I":({"V":92,"N":87},226),
   "E":({"H":86},161),
   "N":({"L":87},234)
   }
def get_min(q):
   mn=(0,(0,float("INF")))
   for i in q:
      if sum(q[i])<sum(mn[1]):
```

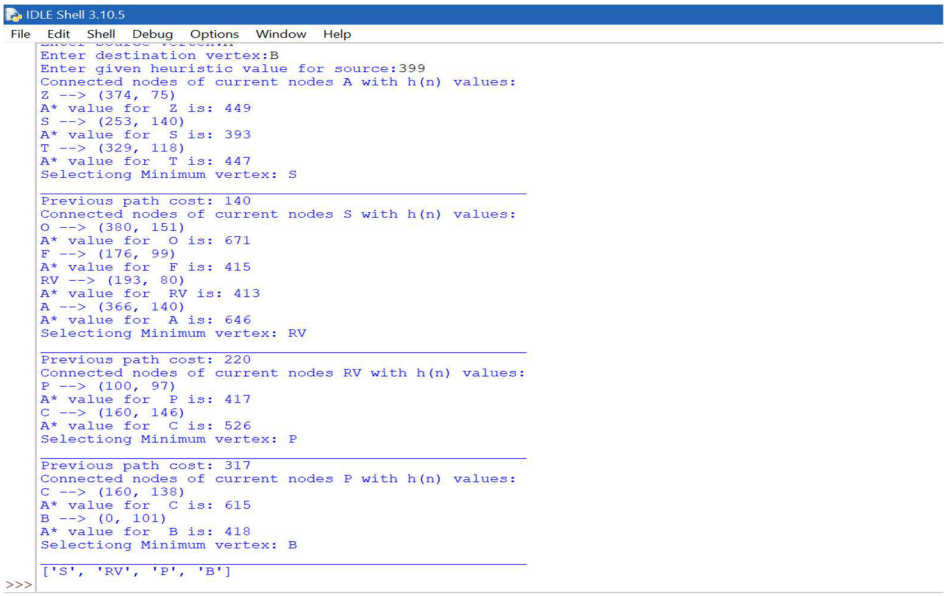
```
            mn=(i,q[i])
    return mn[0]
def a_star(graph,prev,dst,path,pcost,q):
    print("Connected nodes of current nodes",prev,"with h(n) values:")
    for n in graph[prev][0]:
        if n not in path:
            q[n]=(graph[n][1],graph[prev][0][n])
            print(n,"-->",q[n])
            add1=sum(q[n])
            path_cost=pcost+add1
            print("A* value for ",n,"is:",path_cost)
    while q:
        mn=get_min(q)
        print("Selectiong Minimum vertex:",mn)
        print("_____________________________________________________")
        if dst==mn:
            return path+[dst]
        pc=pcost+q[mn][1]
        print("Previous path cost:",pc)
        new_path=a_star(graph,mn,dst,path+[mn],pc,q)
        if new_path:
            return new_path
    return[]
source=input("Enter Source vertex:")
dest=input("Enter destination vertex:")
heuristic=int(input("Enter given heuristic value for source:"))
path=a_star(graph,source,dest,[],0,{source:(heuristic,0)})
if path:
    print(path)
```

else:

print("Path is not found")

Output:-





Practical No.7

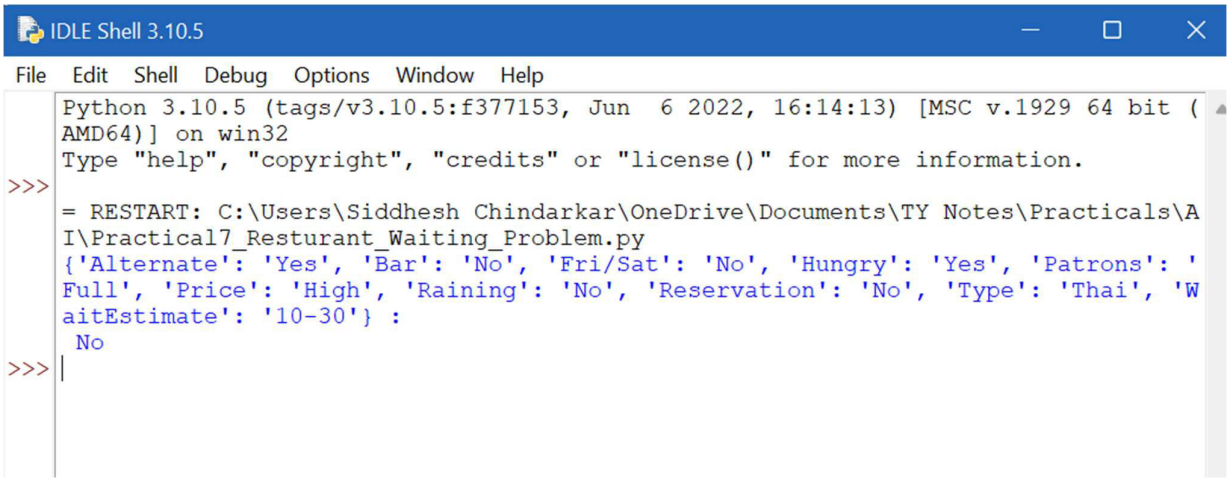
Aim:- Implement Naïve Bayes Learning Algorithm for Restaurant Waiting Problem.

Code:-

```
#import operator

#data set => already taken for prediction

dataset = {

"Ans":          ["Yes","No","Yes","Yes","No","Yes","Yes","No","Yes","No","No","Yes"],

"Alternate":      ["Yes","Yes","No","No","No","Yes","No","No","Yes","Yes","No","Yes"],

"Bar":          ["No","Yes","Yes","Yes","No","No","Yes","No","No","Yes","Yes","No"],

"Fri/Sat":       ["Yes","No","No","No","Yes","No","Yes","Yes","Yes","No","No","Yes"],

"Hungry":        ["No","Yes","No","Yes","No","Yes","No","Yes","No","Yes","No","Yes"],

"Patrons":
["Some","Full","Full","None","Full","Some","Some","Full","Full","Some","Full","Some"],

"Price":
["High","Low","Low","Low","High","High","High","Low","High","Low","High","Low"],

"Raining":       ["Yes","Yes","No","No","No","No","No","Yes","No","Yes","No","No"],

"Reservation":    ["Yes","Yes","No","Yes","No","No","Yes","No","Yes","No","Yes","Yes"],

"Type":
["French","Thai","Burger","Italian","Italian","Thai","French","Thai","Burger","Italian","Burge
r","French"],

"WaitEstimate":   ["10-30","0-10",">60","30-60","10-30","0-10",">60","30-60","30-
60",">60","10-30","0-10"]

}


#input data to test or predict

test_case={

"Alternate":     "Yes",

"Bar":          "No",

"Fri/Sat":       "No",

"Hungry":        "Yes",
```

```
"Patrons":       "Full",
"Price":         "High",
"Raining":       "No",
"Reservation":    "No",
"Type":          "Thai",
"WaitEstimate":   "10-30"
}

def build_probs(ds, test_case):
    ans = ds["Ans"]#output attribute (ans)
    length = len(ans) #total length of output(ans)
    ans_set = set(ans) #unique (individual) classes yes and no
    count_ans = {k: ans.count(k) for k in ans_set}
    calc_prob = {k: count_ans[k] / length for k in ans_set}
    for ft in ds:
        if ft != "Ans":
            counts = {attr: {k: 0 for k in ans_set} for attr in set(ds[ft])}
            for i in range(length):
                counts[ds[ft][i]][ans[i]] +=1
            for k in ans_set:
                calc_prob[k] *= counts[test_case[ft]][k]/ count_ans[k]
    print(test_case,":\n",max(calc_prob, key=calc_prob.get))

build_probs(dataset, test_case)
```

Output:-


IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Siddhesh Chindarkar\OneDrive\Documents\TY Notes\Practicals\A
I\Practical7_Resturant_Waiting_Problem.py
{'Alternate': 'Yes', 'Bar': 'No', 'Fri/Sat': 'No', 'Hungry': 'Yes', 'Patrons': '
Full', 'Price': 'High', 'Raining': 'No', 'Reservation': 'No', 'Type': 'Thai', 'W
aitEstimate': '10-30'} :
No
>>>

Practical No.8

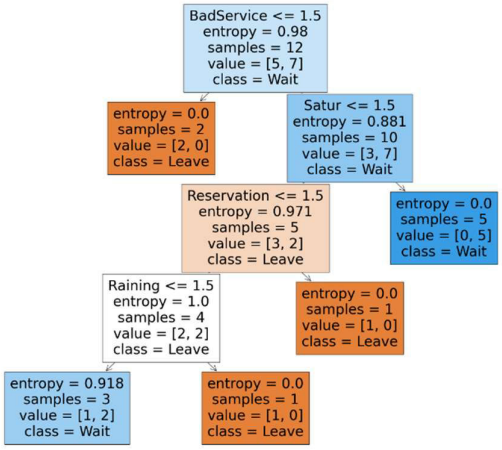
Aim:- Implement Decision Tree Algorithm for the Restaurant Waiting Problem.

Code:-

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn .model_selection import train_test_split
from sklearn import metrics
from matplotlib import pyplot as plt
from sklearn import tree

col_names=['Reservation','Raining','BadService','Satur','Result']
hoteldata=pd.read_csv("dtree.csv",header=None,names=col_names)
feature_cols=['Reservation','Raining','BadService','Satur']
X=hoteldata[feature_cols]
Y=hoteldata.Result
X_train,X_test,Y_train,Y_test,=train_test_split(X,Y,test_size=0.3,random_state=1)
print(hoteldata)

clf=DecisionTreeClassifier(criterion="entropy",max_depth=5)
clf=clf.fit(X_train,Y_train)
Y_pred=clf.predict(X_test)
print("ytest=",X_test)
print("ypred=",Y_pred)
print("Accuracy:",metrics.accuracy_score(Y_test,Y_pred))
fig=plt.figure(figsize=(25,20))
t=tree.plot_tree(clf,feature_names=feature_cols,class_names=['Leave','Wait'],filled=True)
fig.savefig("decistion_tree.png")
```

Output:-

```
IDLE Shell 3.10.5
File  Edit  Shell  Debug  Options  Window  Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:\Users\Siddhesh Chindarkar\OneDrive\Documents\TY Notes\Practicals\AI\Practical8_Decision_Tree.py
    Reservation  Raining  BadService  Satur Result
0             1        1           2      1   Wait
1             2        2           2      2   Wait
2             2        1           1      1  Leave
3             1        1           1      1   Wait
4             2        1           2      1  Leave
5             2        1           2      2   Wait
6             1        2           2      1  Leave
7             2        1           2      2   Wait
8             1        1           2      1  Leave
9             1        2           1      2  Leave
10            2        1           2      2   Wait
11            1        2           2      1  Leave
12            2        1           2      2   Wait
13            1        1           2      1  Leave
14            1        2           1      2  Leave
15            1        1           2      1   Wait
16            2        2           2      2   Wait
17            2        1           1      1  Leave
ytest=     Reservation  Raining  BadService  Satur
6             1        2           2      1
3             1        1           1      1
13            1        1           2      1
2             2        1           1      1
14            1        2           1      2
7             2        1           2      2
ypred= ['Leave' 'Leave' 'Wait' 'Leave' 'Leave' 'Wait']
Accuracy: 0.6666666666666666
>>>
```



Practical No.9

Aim:- Implement Majority Voting Classifier in Ensemble Learning.

Code:-

```
import numpy as np
from numpy import *
from sklearn import datasets
from sklearn import model_selection
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
import warnings
warnings.filterwarnings("ignore")

iris=datasets.load_iris()#iris dataset 5cols
x,y=iris.data[:,1:3],iris.target#first paramete for row, and second for col,here : in row is for all row
m1=LogisticRegression(random_state=1)
m2=RandomForestClassifier(random_state=1)
m3=GaussianNB()
m4=DecisionTreeClassifier()

labels=['Logistic Regression','Random Forest', 'Naive Bayes', 'Decision Tree']
for m, label in zip([m1,m2,m3,m4],labels):
   scores=model_selection.cross_val_score(m,x,y,cv=5,scoring='accuracy')
   print("Accuracy: %0.2f [%s]" %(scores.mean(), label))
voting_clf_hard=VotingClassifier(estimators=[(labels[0],m1),
                                 (labels[1],m2),
```

```
                    (labels[2],m3),
                    (labels[3],m4)],
                voting='hard')
scores1=model_selection.cross_val_score(voting_clf_hard,x,y,cv=5,scoring='accuracy')
print("Accuracy of combined Model Using Hard Voting: %0.2f"  %(scores1.mean()))
```

Output:-