**ĐỘC CÔ CỬU KIẾM – KIẾM PHÁP BY VUDANG**

**Phần 1 : Python kiếm phổ cơ bản – OOP trong python – BFS – DFS – A\***

|  |  |
| --- | --- |
| **# 1. Viết một chương trình Python in ra dãy số Fibonacci giữa 0 và 50.**  **def fibo\_find():**  **n1, n2 = 0, 1**  **for i in range(0, 50):**  **print(n1)**  **n = n1 + n2**  **n1 = n2**  **n2 = n**  **fibo\_find()**  **#2. Viết một chương trình Python đếm số các số chẵn và số các số lẻ của một dãy số**  **def is\_even(n: int) -> bool:**  **return True if n % 2 == 0 else False**  **def count\_number():**  **count\_odd = 0**  **count\_even = 0**  **arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**  **for i in range(len(arr)-1):**  **if is\_even(arr[i]):**  **count\_even += 1**  **else:**  **count\_odd += 1**  **print("Odd -> " + repr(count\_odd))**  **print("Even -> " + repr(count\_even))**  **count\_number()**  **3. Viết một chương trình Python xây dựng cấu trúc sau, bằng việc sử dụng vòng lặp for**  **lồng nhau.**  **\***  **\* \***  **\* \* \***  **\* \* \* \***  **\* \* \* \* \***  **def print\_star():**  **for i in range(0,6):**  **for j in range(i):**  **print("\* ",end=" ")**  **print(" ")**  **print\_star()**  **#4. Viết một chương trình Python tính tổng tất cả các số trong một danh sách.**  **def sum\_list(list: []) -> int :**  **sum = 0**  **for i in range(len(list)):**  **sum += list[i]**  **return sum**  **test = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**  **print(sum\_list(test))**  **BFS – DFS**  **#Khai báo đồ thị**  **graph={**  **'A':['B','C'],**  **'B':['D','E'],**  **'C':['E','I'],**  **'D':['F'],**  **'E':['F','J'],**  **'F':['G','H'],**  **'G':[], 'H':[],**  **'I':['K'],**  **'J':[],'K':[],**  **}**  **#Bài 1. Viết chương trình khai báo đồ thị và duyệt đồ thị theo giải thuật BreadthFirst Search (BFS) biết rằng BFS làm việc theo cơ chế Queue (hàng đợi) – FIFO**  **#(First In First Out). Sau đó xuất kết quả duyệt đồ thị ra màn hình.**  **#khai bao giai thuat BFS**  **def BFS(graph,start):**  **queue=[start]**  **visited=[]**  **while queue: #queue khac rong thi chay tiep tuc**  **node=queue.pop(0)**  **if node in visited:**  **continue**  **visited.append(node)**  **for nextNode in graph[node]:**  **queue.append(nextNode)**  **return visited**  **#call function BFS**  **print("Ket qua duyet do thi theo BFS: ")**  **print(BFS(graph,'A'))**  **#Bài 2. Dựa và giải thuật BFS. Sinh viên viết chương trình khai báo đồ thị và duyệt**  **#đồ thị theo giải thuật Depth-First Search (DFS) biết rằng DFS làm việc theo cơ chế**  **#Stack (ngăn xếp) – LIFO (Last In First Out). Sau đó xuất kết quả duyệt đồ thị ra màn hình.**  **def DFS(graph,start,visited):**  **if start not in visited:**  **visited.append(start)**  **for n in graph[start]:**  **DFS(graph,n,visited)**  **return visited**  **visited=DFS(graph,'A',[])**  **print("Duyet do thi theo DFS: ")**  **print(visited)**  **#Bài 3. Viết chương trình cải tiến giải thuật Breadth-First Search (BFS) để in ra**  **#đường đi từ vị trí start đến vị trí end trong đồ thị.**  **def BFS\_Path(graph,start,end):**  **queue=[(start,[start])]**  **while queue:**  **node,path=queue.pop(0)**  **for nextNode in graph[node]:**  **if nextNode in path:**  **continue**  **elif nextNode==end:**  **return path+[nextNode]**  **else:**  **queue.append((nextNode,path+[nextNode]))**  **#Call function BFS\_Path**  **print("DUYET DO THI THEO BFS")**  **start=input("Nhap vi tri bat dau: ")**  **end=input("Nhap vi tri ket thuc: ")**  **print("Duong di tu "+start+" den "+end+": ")**  **print(BFS\_Path(graph,start,end))**  **print("------------------------------------")**  **#Bài 4. Dựa vào Bài 3, sinh viên viết chương trình cải tiến giải thuật Depth-First**  **#Search (DFS) để in ra đường đi từ vị trí start đến vị trí end trong đồ thị.**  **def DFS\_Path(graph, start, goal):**  **stack = [(start, [start])]**  **visited = set()**  **while stack:**  **(vertex, path) = stack.pop()**  **if vertex not in visited:**  **if vertex == goal:**  **return path**  **visited.add(vertex)**  **for neighbor in graph[vertex]:**  **stack.append((neighbor, path + [neighbor]))**  **#Call functionn DFS\_Path**  **print("DUYET DO THI THEO DFS")**  **start=input("Nhap vi tri bat dau: ")**  **end=input("Nhap vi tri ket thuc: ")**  **print("Duong di tu "+start+" den "+end+": ")**  **print(DFS\_Path(graph,start,end))**  **A\***  **# Khai báo Graph**  **class Graph:**  **def \_\_init\_\_(self, adjacency\_list, H):**  **self.adjacency\_list = adjacency\_list**  **self.H = H**  **def get\_neighbors(self, v):**  **return self.adjacency\_list[v]**  **def h(self, n):**  **return self.H[n]**  **def a\_star\_algorithm(self, start\_node, end\_node):**  **open\_list = set([start\_node])**  **closed\_list = set([])**  **g = {}**  **g[start\_node] = 0**  **parents = {}**  **parents[start\_node] = start\_node**  **while open\_list:**  **n = None**  **for v in open\_list:**  **if n == None or (g[v] + self.h(v) < g[n] + self.h(n)):**  **n = v**  **if n == None:**  **print("Cannot find path")**  **return None**  **if n == end\_node:**  **path = []**  **while parents[n] != n:**  **path.append(n)**  **n = parents[n]**  **path.append(start\_node)**  **path.reverse()**  **print("Path : {}".format(path))**  **return path**  **for (m, cost) in self.get\_neighbors(n):**  **if m not in open\_list and m not in closed\_list:**  **open\_list.add(m)**  **parents[m] = n**  **g[m] = g[n] + cost**  **else:**  **if g[m] > g[n] + cost:**  **g[m] = g[n] + cost**  **parents[m] = n**  **if m in closed\_list:**  **closed\_list.remove(m)**  **open\_list.add(m)**  **open\_list.remove(n)**  **closed\_list.add(n)**  **print("Doesnt exist path")**  **return None**  **# adjacency\_list = {**  **# 'A': [('C', 9), ('D', 7), ('E', 13), ('F', 20)],**  **# 'C': [('H', 6)],**  **# 'D': [('E', 4), ('H', 8)],**  **# 'E': [('K', 4), ('I', 3)],**  **# 'F': [('I', 6), ('G', 4)],**  **# 'H': [('K', 5)],**  **# 'K': [('B', 6)],**  **# 'I': [('K', 9), ('B', 5)],**  **# }**  **#**  **# heuristic = {**  **# 'A': 14,**  **# 'B': 0,**  **# 'C': 15,**  **# 'D': 6,**  **# 'E': 8,**  **# 'F': 7,**  **# 'G': 12,**  **# 'H': 10,**  **# 'K': 2,**  **# 'I': 4**  **# }**  **my\_graph = Graph(adjacency\_list, heuristic)**  **my\_graph.a\_star\_algorithm(‘A’,’B’)**  **\*Decision Tree:**  **# Tiểu đường**  **import pandas as pd**  **from sklearn.tree import DecisionTreeClassifier**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn import metrics**  **col\_names = ['pregant','glucose','bp','skin','insulin','bmi','pedigree','age','label'**  **]**  **#load dataset**  **pima = pd.read\_csv("D:\\diabetes.csv",header=0,names = col\_names)**  **pima.head()**  **#split**  **feature\_cols=['pregant','glucose','bp','skin','insulin','bmi','pedigree','age']**  **X = pima[feature\_cols] #features**  **y = pima.label #target variable**  **print(pima[feature\_cols])**  **print(pima.label)**  **#split ,train**  **X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=1)**  **# create decision tree classifer object**  **clf = DecisionTreeClassifier()**  **#train**  **clf = clf.fit(X\_train,y\_train)**  **#predict**  **y\_pred = clf.predict(X\_test)**  **print(y\_pred)**  **#model Accuracy**  **print("Accuracy",metrics.accuracy\_score(y\_test,y\_pred))**  **# Gender Transport**  **import pandas as pd**  **import numpy as np**  **from sklearn.tree import DecisionTreeClassifier**  **from sklearn.model\_selection import train\_test\_split**  **from sklearn import metrics**  **from sklearn import preprocessing**  **col\_names =['gender','carownership','travelcost','incomelevel','transportationmode']**  **#load dataset**  **pima = pd.read\_csv("D:\\transport.csv",header = 0,names =col\_names)**  **pima.head()**  **lE = preprocessing.LabelEncoder()**  **data = pima.apply(lE .fit\_transform)**  **print(data)**  **#split**  **feature\_cols = ['gender','carownership','travelcost','incomelevel']**  **X = data[feature\_cols]**  **y =data.transportationmode**  **X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3,random\_state=1)**  **clf = DecisionTreeClassifier()**  **clf = clf.fit(X\_train,y\_train)**  **#predict**  **y\_pred = clf.predict(X\_test)**  **print(y\_pred)**  **print("Accuracy:" ,metrics.accuracy\_score(y\_test,y\_pred))**  **d1 = np.array([[0,2,1,2]])**  **d1\_pred = clf.predict(d1)**  **print(d1\_pred)**  **ketqua=lE.inverse\_transform(d1\_pred)**  **print(ketqua)**  # Perceptron Neurel Network  **from sklearn.neural\_network import MLPClassifier**  **from sklearn.linear\_model import Perceptron**  **import sklearn.metrics as metric**  **import numpy as np**  **X\_training = [ [1,1],**  **[1,0],**  **[0,1],**  **[0,0],**  **]**  **y\_training = [1,**  **1,**  **1,**  **0**  **]**  **X\_testing = X\_training**  **y\_true = y\_training**  **ptn = Perceptron(max\_iter= 500)**  **ptn.fit(X\_training ,y\_training)**  **y\_pred=ptn.predict(X\_testing)**  **print(y\_pred)**  **accuracy = metric.accuracy\_score(y\_true,y\_pred,normalize = True)**  **print('accuracy = ',accuracy)**  **#Multilayer Percepton**  **from sklearn.neural\_network import MLPClassifier**  **from sklearn.linear\_model import Perceptron**  **import sklearn.metrics as metric**  **import numpy as np**  **X\_training = [ [1,1],**  **[1,0],**  **[0,1],**  **[0,0],**  **]**  **y\_training = [1,**  **1,**  **1,**  **0**  **]**  **X\_testing = X\_training**  **y\_true = y\_training**  **mlp = MLPClassifier(solver = 'lbfgs',hidden\_layer\_sizes=[1,1],activation='logistic')**  **mlp.fit(X\_training,y\_training)**  **y\_pred=mlp.predict(X\_testing)**  **print(y\_pred)**  **accuracy = metric.accuracy\_score(np.array(y\_true).flatten(),np.array(y\_pred).flatten(),normalize = True)**  **print('accuracy = ',accuracy)**  **# In[26]:** | **import numpy as np**  **import datetime**  **# 1. Viết hàm trong chương trình Python in ra dãy số Fibonacci giữa 0 và 50 (sử dụng hàm đệ quy)**  **def print\_fibonacci(n):**  **if n <= 1:**  **return n**  **return print\_fibonacci(n - 1) + print\_fibonacci(n - 2)**  **def run\_fibo(n):**  **for i in range(n):**  **print(print\_fibonacci(i))**  **# run\_fibo(50)**  **# 2. Viết một hàm Python tính tổng tất cả các số trong một danh sách**  **def sum\_list(list\_array: []) -> int:**  **sum\_all = 0**  **for i in range(len(list\_array)):**  **sum\_all += list\_array[i]**  **return sum\_all**  **arr = [6, 9, 1, 5, 2, 6]**  **# print(sum\_list(arr))**  **# 3. Viết một hàm Python đầu vào là một số và**  **# kiểm tra xem số đó có phải là số nguyên tố hay**  **# không**  **def is\_prime(n: int) -> bool:**  **count\_div = 1**  **for i in range(1, n):**  **if n % i == 0:**  **count\_div += 1**  **if count\_div > 2:**  **return False**  **return True**  **def check\_prime(n: int):**  **if is\_prime(n):**  **print("Is prime")**  **else:**  **print("Not prime")**  **# check\_prime(12)**  **# 4. Khởi tạo 2 ma trận 3x3, áp dụng các phép toán cộng,**  **# trừ 2 ma trận và xuất ra ma trận kết quả.**  **def create\_matrix():**  **a = np.arange(2, 11).reshape(3, 3)**  **return a**  **def sum\_matrix(a, b):**  **return a + b**  **def minus\_matrix(a, b):**  **return a - b**  **matrix\_one = create\_matrix()**  **matrix\_two = create\_matrix()**  **# res\_sum = sum\_matrix(matrix\_one,matrix\_two)**  **# res\_minus = minus\_matrix(matrix\_one,matrix\_two)**  **# print(res\_sum)**  **# print(res\_minus)**  **# Lớp hình chữ nhật tam giác hình học**  **class Shape:**  **width = 0**  **height = 0**  **def \_\_init\_\_(self):**  **self.width = 0**  **self.height = 0**  **def get\_height(self):**  **return self.height**  **def get\_width(self):**  **return self.width**  **class Triangle(Shape):**  **def \_\_init\_\_(self):**  **self.height = super().get\_height()**  **self.base = super().get\_width()**  **def input(self):**  **print("Enter value of triangle")**  **self.height = int(input("Height : "))**  **self.base = int(input("Base: "))**  **def output(self):**  **print("Triangle ")**  **print("Base " + repr(self.base))**  **print("Height " + repr(self.height))**  **def area(self) -> float:**  **return (self.height \* self.base) / 2**  **def get\_class\_name(self):**  **return "Triangle"**  **class Rectangle(Shape):**  **def \_\_init\_\_(self):**  **self.height = super().get\_height()**  **self.width = super().get\_width()**  **def input(self):**  **print("Enter value of rectangle")**  **self.height = int(input("Height : "))**  **self.width = int(input("Width: "))**  **def output(self):**  **print("Rectangle ")**  **print("Width " + repr(self.width))**  **print("Height " + repr(self.height))**  **def area(self):**  **return self.height \* self.width**  **def get\_class\_name(self):**  **return "Rectangle"**  **def my\_input(n):**  **result = []**  **for i in range(0, n ):**  **print("Enter shape => " + repr(i))**  **print("Enter shape type :")**  **print(" Triangle is T")**  **print(" Rectangle is C")**  **choice = input()**  **if choice != 'C' and choice != 'T':**  **print("Invalid")**  **return**  **if choice == 'C':**  **r = Rectangle()**  **r.input()**  **result.append(r)**  **if choice == 'T':**  **t = Triangle()**  **t.input()**  **result.append(t)**  **return result**  **def my\_output(list):**  **sum\_area = 0**  **for i in range(0,len(list)):**  **list[i].output()**  **sum\_area += list[i].area()**  **print("Area of " + list[i].get\_class\_name() + " " + repr(list[i].area()))**  **print("Sum area is " + repr(sum\_area))**  **def my\_main():**  **n = int(input("Enter number of Shape : "))**  **temp = my\_input(n)**  **print("\n")**  **my\_output(temp)**  **my\_main()**  **\*PROLOG**      **\*NATIVE BAYERS**  **from \_\_future\_\_ import print\_function**  **from sklearn.naive\_bayes import MultinomialNB**  **import numpy as np**  **#train data**  **d1 = [0, 0, 0, 1]**  **d2 = [0, 0, 0, 0]**  **d3 = [1, 0, 0, 1]**  **d4 = [2, 1, 0, 1]**  **d5 = [2, 2, 1, 1]**  **d6 = [2, 2, 1, 0]**  **d7 = [1, 2, 1, 0]**  **d8 = [0, 1, 0, 1]**  **d9 = [0, 2, 1, 1]**  **d10 = [2, 1, 1, 1]**  **d11 = [0, 1, 1, 0]**  **d12 = [1, 1, 0, 0]**  **d13 = [1, 0, 1, 1]**  **d14 = [2, 1, 0, 0]**  **train\_data = np.array([d1, d2, d3, d4, d5, d6, d7, d8, d9, d10, d11, d12, d13, d14])**  **label = np.array(['No', 'No', 'Yes', 'Yes', 'Yes','No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No' ])**  **d15 = np.array([[0, 1, 0, 0]])**  **## call MultinomialNB**  **clf = MultinomialNB()**  **# training**  **clf.fit(train\_data, label)**  **# test**  **print('Predicting class of d5:', str(clf.predict(d15)[0]))**  **print('Probability of d6 in each class:', clf.predict\_proba(d15))**  **\*Email spam Native Bayers**  **import pandas as pd**  **import numpy as np**  **#đọc dữ liệu từ file**  **file\_path = 'D:/Test/SMSSpamCollection.txt'**  **df = pd.read\_csv(file\_path, delimiter='\t', header=None, skipinitialspace=True, names=['label', 'msg'])**  **#print(df)**  **#chuyển Label thành giá trị nhị phân 0: ham; 1: spam**  **df['label'] = df.label.map({'ham':0, 'spam':1})**  **df.head()**  **#chia tập dữ liệu thành 2 tập: training data và test data theo tỉ lệ 7: 3**  **from sklearn.model\_selection import train\_test\_split**  **X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(df.iloc[:,1], df.iloc[:,0], test\_size=0.3, random\_state=50)**  **print (X\_train.head(10))**  **print (Y\_train.head(10))**  **from sklearn.feature\_extraction.text import CountVectorizer**  **# Khởi tạo vectorizer**  **vect = CountVectorizer()**  **#đếm các từ trong tập dữ liệu train và chuyển thành ma trận từ**  **#(Learn vocabulary and create document-term matrix in a single step)**  **train\_dtm = vect.fit\_transform(X\_train)**  **train\_dtm**  **#Chuyển tập dữ liệu text thành ma trận từ (transform testing data into a document-term matrix)**  **test\_dtm = vect.transform(X\_test)**  **test\_dtm**  **#Chuyển tập traint\_dtm thành array**  **train\_arr = train\_dtm.toarray()**  **train\_arr**  **from sklearn.naive\_bayes import MultinomialNB**  **nb = MultinomialNB()**  **nb.fit(train\_dtm, Y\_train)**  **#DỰ đoán dữ liệu trong test(make predictions on test data using test\_dtm**  **Y\_pred = nb.predict(test\_dtm)**  **Y\_pred**  **#So sánh độ chính xác với các Label trong test có sẵn để biết bộ chính xác phân loại**  **#(compare predictions to true Labels)**  **from sklearn import metrics**  **print('accuracy= ', metrics.accuracy\_score(Y\_test, Y\_pred))**  **#Neurel network**  **from sklearn.neural\_network import MLPClassifier**  **from sklearn.linear\_model import Perceptron**  **import sklearn.metrics as metric**  **import numpy as np**  **X\_training = [ [ 1, 1, 0],**  **[ 1, -1, -1],**  **[-1, 1, 1],**  **[-1, -1, 1],**  **[ 0, 1, -1],**  **[ 0, -1, -1],**  **[ 1, 1, 1]**  **]**  **y\_training = [[1,0],**  **[0,1],**  **[1,1],**  **[1,0],**  **[1,0],**  **[1,1],**  **[1,1]**    **]**  **X\_testing = X\_training**  **y\_true = y\_training**  **mlp = MLPClassifier(solver = 'lbfgs',hidden\_layer\_sizes=[3,2],activation='logistic')**  **mlp.fit(X\_training,y\_training)**  **y\_pred=mlp.predict(X\_testing)**  **print(y\_pred)**  **accuracy = metric.accuracy\_score(np.array(y\_true).flatten(),np.array(y\_pred).flatten(),normalize = True)**  **print('accuracy = ',accuracy)**  **# In[4]:**  **# Neurel network Weather**  **import pandas as pd**  **import numpy as np**  **import sklearn.metrics as metric**  **from sklearn import metrics**  **from sklearn import preprocessing**  **from sklearn.neural\_network import MLPClassifier**  **from sklearn.linear\_model import Perceptron**  **from sklearn.model\_selection import train\_test\_split**  **col\_names = ['Outlook','Temp','Humidity','Windy','Play']**  **dt = pd.read\_csv("D:\\Play.csv",header = 0 , names = col\_names)**  **dt.head()**  **lE = preprocessing.LabelEncoder()**  **data = dt.apply(lE.fit\_transform)**  **print (data)**  **feature\_cols = ['Outlook','Temp','Humidity','Windy']**  **X = data[feature\_cols]**  **y = data.Play**  **X\_train,X\_test,y\_train,y\_test = train\_test\_split(X ,y ,test\_size = 0.3 , random\_state = 1 )**  **mlp = MLPClassifier(solver = 'lbfgs',hidden\_layer\_sizes=[3,2],activation='logistic')**  **mlp.fit(X\_train,y\_train)**  **y\_pred=mlp.predict(X\_test)**  **print(y\_pred)**  **print("Accuracy:" ,metrics.accuracy\_score(y\_test,y\_pred))**  **d1 = np.array([[2,0,0,1]])**  **d1\_pred = mlp.predict(d1)**  **print(d1\_pred)**  **ketqua=lE.inverse\_transform(d1\_pred)**  **print(ketqua)** |