**TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT TP. HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**🙠 🕮 🙢**



**MÔN HỌC: LẬP TRÌNH PYTHON**

**BÀI TẬP LẬP TRÌNH**

**Mã LHP: IPPA233277\_23\_2\_07**

**SVTH MSSV**

Bùi Lê Thủy Tiên 22162048

**GVHD:** ThS Từ Tuyết Hồng

***Câu 1:*** Khoảng cách chỉnh sửa giữa hai chuỗi là thước đo mức độ giống nhau của chúng. Khoảng cách chỉnh sửa càng nhỏ thì các chuỗi càng giống nhau về số lượng thao tác chèn, xóa và thay thế tối thiểu cần thiết để chuyển đổi chuỗi này thành chuỗi khác.  
Hãy xem xét các chuỗi kitten và sitting. Chuỗi đầu tiên có thể được chuyển đổi thành chuỗi thứ hai bằng các thao tác sau: Thay thế k bằng chữ s, thay thế e bằng chữ i và chèn g vào cuối chuỗi. Đây là số thao tác nhỏ nhất có thể được thực hiện để chuyển kitten sang sitting.  
Kết quả là khoảng cách chỉnh sửa là 3.  
Viết hàm đệ quy tính khoảng cách chỉnh sửa giữa hai chuỗi.  
Sử dụng hàm đệ quy của bạn để viết chương trình đọc hai chuỗi từ người dùng và hiển thị khoảng cách chỉnh sửa giữa chúng.

Thành phố Hồ Chí Minh, Tháng 5 năm 2024

***Solution:***

def edit\_distance(str1, str2):

    #trường hợp cơ sở: nếu str1 rỗng thì khoảng cách là len str2, ngược lại

    if not str1:

        return len(str2)

    if not str2:

        return len(str1)

#nếu kí tự đầu giống thì số lần chỉnh sửa = số lần chỉnh sửa các kí tự còn

    if str1[0] == str2[0]:

        return edit\_distance(str1[1:], str2[1:])

    else:

        insert\_cost = edit\_distance(str1, str2[1:]) + 1

        delete\_cost = edit\_distance(str1[1:], str2) + 1

        substitute\_cost = edit\_distance(str1[1:], str2[1:]) + 1

        return min(insert\_cost, delete\_cost, substitute\_cost)

#chương trình chính

str1 = input("Nhập chuỗi 1: ")

str2 = input("Nhập chuỗi 2: ")

distance = edit\_distance(str1, str2)

print(f"Khoảng cách chỉnh sửa giữa '{str1}' và '{str2}' là: {distance}")

***Câu 2:*** Trong mặt phẳng Euclid, cho điểm p có tọa độ Descartes là (p1, p2) và điểm q có tọa độ (q1, q2). Khi đó khoảng cách giữa p và p được tính như sau:



# Tính khoảng cách Euclid giữa 2 điểm p1 và p2  
import numpy as np  
p = 121  
p1 = np.array((1, 1, 1))  
p2 = np.array((1, 2, 3))  
print(np.sqrt(p))  
print(np.dot(p1,p2))  
print(np.square(p1-p2))  
kc = np.\_\_\_\_\_\_\_\_\_\_\_\_\_  
print(kc)  
Hãy điền phần lệnh bị thiếu vào chỗ trống để chương trình chạy và cho kết quả đúng.  
Output mẫu:  
**11.0  
6   
[0 1 4]  
2.23606797749979  
Chú ý: Phần trả lời không được chứa ký tự trắng và ký tự hoa thị ( \* )**

***Solution:***

# Tính khoảng cách Euclid giữa 2 điểm p1 và p2

import numpy as np

p = 121

p1 = np.array((1, 1, 1))

p2 = np.array((1, 2, 3))

print(np.sqrt(p))

print(np.dot(p1,p2))

print(np.square(p1-p2))

kc = np.sqrt(np.sum(np.square(p1-p2)))

print(kc)

***Câu 3:***

# Đếm số chuỗi con trong một mảng chuỗi  
import numpy as np  
iarr = np.array(['Teyenten', ' Teyen ', 'Teyenqubhre'])  
oarr = np.char.count(iarr,sub='en')  
print(oarr)  
arr = np.char.count(\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_)  
print(arr)

Điền phần lệnh bị thiếu vào chỗ trống để chương trình chạy và cho kết quả đúng.  
Output mẫu là:  
[2 1 1][3 2 2]Chú ý: Phần trả lời không được chứa ký tự trắng.

*Solution:*

# Đếm số chuỗi con trong một mảng chuỗi

import numpy as np

iarr = np.array(['Teyenten', ' Teyen ', 'Teyenqubhre'])

oarr = np.char.count(iarr,sub='en')

print(oarr)

arr = np.char.count(iarr,sub='e',start=0,end=7)

print(arr)

*Câu 4:*

Viết chương trình tính tích các thừa số nguyên tố duy nhất của một số nguyên dương  
Ví dụ:  
*Input*: 44  
*Output*: 22  
vì các thừa số của 44 là 2, 2, 11  
nên tích số cần tìm = 2\*11 = 22

***Solution:***

#tích các thừa số nguyên tố

def prime\_factors(n):

#trả về danh sách chứa các thừa số nguyên tố

    factors = []

    i = 2

    while i \* i <= n:

        if n % i == 0:

            factors.append(i)

            n //= i

        else:

            i += 1

    if n > 1:

        factors.append(n)

    return factors

#tính tích các thừa số nguyên tố

def product\_of\_unique\_prime\_factors(n):

    prime\_factors\_list = prime\_factors(n)

    unique\_prime\_factors = list(set(prime\_factors\_list))

    product = 1

    for factor in unique\_prime\_factors:

        product \*= factor

    return product

number = int(input())

result = product\_of\_unique\_prime\_factors(number)

print(result)

***Câu 5:***

import matplotlib.pyplot as plt  
import numpy as np  
x = np.arange(-2\*np.pi,2\*np.pi,0.01)  
y = np.sin(3\*x)/x  
y2 = np.sin(2\*x)/x  
y3 = np.sin(x)/x  
# ....  
Hãy viết tiếp phần lệnh bị thiếu vào chương trình trên để vẽ đồ thị giống như hình mẫu:

A diagram of a function

Description automatically generated

***Solution:***

import matplotlib.pyplot as plt

import numpy as np

x = np.arange(-2\*np.pi, 2\*np.pi, 0.01)

y = np.sin(3\*x)/x

y2 = np.sin(2\*x)/x

y3 = np.sin(x)/x

fig = plt.figure(figsize=(6,4))

ax = fig.add\_subplot(1, 1, 1)

#di chuyển left vào giữa và bottom ở tọa độ (0, 0)

ax.spines['left'].set\_position('zero')

ax.spines['bottom'].set\_position('zero')

#xóa top và right khỏi đồ thị

ax.spines['right'].set\_color('none')

ax.spines['top'].set\_color('none')

#thiết lập vạch trên trục x và y cách đều 1 đơn vị

x\_ticks = np.array([-2\*np.pi, -np.pi, 0, np.pi, 2\*np.pi])

y\_ticks = np.arange(-1, 4, 1)

plt.xticks(x\_ticks, ['$-2\pi$', '$-\pi$', '0', '$+\pi$', '$+2\pi$'],fontsize=9)

plt.yticks(y\_ticks, ['$-1$', '0', '$+1$', '$+2$','$+3$'],fontsize=10)

#Giới hạn giá trị cột x, y

plt.ylim(-1, 3.2)

#Vẽ đồ thị

plt.plot(x, y, color='blue')

plt.plot(x, y2, color='red')

plt.plot(x, y3, color='green')

#Hiển thị

plt.show()

***Câu 6:*** Sử dụng matplotlib, viết chương trình vẽ đồ thị giống như hình sau:

A graph with a red line

Description automatically generated

***Solution:***

import matplotlib.pyplot as plt

import numpy as np

x = np.linspace(-11, 10, 100)

y = x\*x-20

plt.plot(x, y, 'r-')

plt.grid(visible=True,c='green',ls='--')

plt.xlabel('X axis')

plt.ylabel('Y axis')

plt.title('Line graph')

plt.show()

***Câu 7:***

A close-up of numbers

Description automatically generated

Hãy điền thêm phần lệnh bị thiếu vào ô trống màu xanh trên các dòng lệnh 3, 5, 6 trong đoạn lệnh trên để chương trình chạy và cho kết quả  
đúng.  
**Output mẫu** là:  
**Ma trận:  
[[24 15 36]  
 [81 61 10]  
 [77 19 5]]  
Cột thứ 3 của ma trận: [36, 10, 5]**

***Solution:***

#Tìm cột thứ K của ma trận

import numpy as np

L = np.array([[24,15,36],[81,61,10],[77,19,5]])

print("Ma trận:",L,sep='\n')

K = 3

r = [x[K-1] for x in L]

print(f"Cột thứ {K} của ma trận: {r}")

***Câu 8:***

Một chuỗi được gọi là một chuỗi palindrome nếu đảo ngược của chuỗi giống với chuỗi đó.  
Ví dụ: “radar” là một palindrome, nhưng “radix” không phải là một palindrome (vì đảo của "radix" là "xidar") Viết chương trình kiểm tra biểu diễn nhị phân của một số nguyên dương có phải là Palindrome không?  
*Ví dụ:***Input:** 9  
**Output**: True  
vì biểu diễn nhị phân của 9 là 1001 (là palindrome)  
**Input**: 10  
**Output:** False  
vì biểu diễn nhị phân của 10 là 1010 (không là palindrome)

***Solution:***

#hàm kiểm tra chuỗi palindmore, trả về boolean

def isPalindmore(s):

    return s == s[::-1]

#hàm đổi số sang chuỗi nhị phân, trả về string

def toBinString(x):

    res = ''

    while x > 0:

        res = str(x % 2) + res

        x //= 2

    return res

#chương trình chính

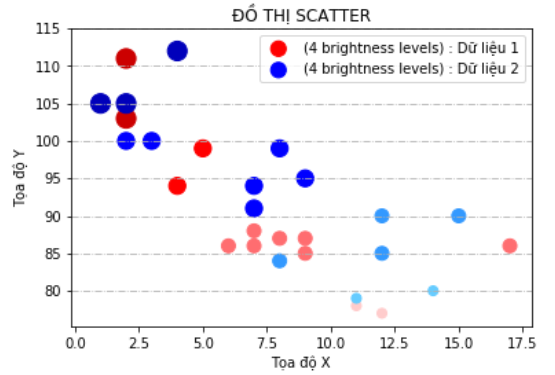
num = int(input("Nhập số nguyên dương: "))

inputString = toBinString(num)

print(isPalindmore(inputString))

***Câu 9:***

import matplotlib.pyplot as plt  
import numpy as np  
# Dữ liệu 1  
x1 = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])  
y1 = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])  
# Dữ liệu 1  
x2 = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])  
y2 = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])  
# ....  
Hãy viết tiếp phần lệnh bị thiếu vào chương trình trên để vẽ đồ thị giống như hình mẫu:



***Solution:***

import matplotlib.pyplot as plt

import numpy as np

#data 1

x1 = np.array([5, 7, 8, 7, 2, 17, 2, 9, 4, 11, 12, 9, 6])

y1 = np.array([99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86])

#data 2

x2 = np.array([2, 2, 8, 1, 15, 8, 12, 9, 7, 3, 11, 4, 7, 14, 12])

y2 = np.array([100, 105, 84, 105, 90, 99, 90, 95, 94, 100, 79, 112, 91, 80, 85])

#create colors and size for spot

sizes1 = []

sizes2 = []

colors1 = []

colors2 = []

for y in y1:

    if 100 <= y <= 115:

        sizes1.append(300)

        colors1.append("crimson")

    elif 90 <= y < 100:

        sizes1.append(250)

        colors1.append("red")

    elif 80 < y < 90:

        sizes1.append(160)

        colors1.append("salmon")

    else:

        sizes1.append(100)

        colors1.append("mistyrose")

for y in y2:

    if 100 < y <= 115:

        sizes2.append(300)

        colors2.append("mediumblue")

    elif 90 < y <= 100:

        sizes2.append(250)

        colors2.append("blue")

    elif 80 < y <= 90:

        sizes2.append(160)

        colors2.append("dodgerblue") #deepskyblue

    else:

        sizes2.append(100)

        colors2.append("skyblue")

#scatter plot for data 1

plt.scatter(x1, y1, s=sizes1, color=colors1, label='(4 brightness levels) : Dữ liệu 1')

#scatter plot for data 2

plt.scatter(x2, y2, s=sizes2, color=colors2, label='(4 brightness levels) : Dữ liệu 2')

#title

plt.xlabel('Tọa độ X')

plt.ylabel('Tọa độ Y')

plt.title('ĐỒ THỊ SCATTER')

#range for xaxis and yxis

x\_ticks = np.arange(0.0, 18.0, 2.5)

plt.xticks(x\_ticks)

plt.xlim(-0.2, 17.8)

plt.ylim(75.1, 115)

#(-. style)

plt.grid(axis='y', linestyle='-.')

#label

plt.legend(fontsize=12)

#show

plt.show()

***Câu 10:*** Bit chẵn lẻ (parity bit) là một cơ chế đơn giản để phát hiện lỗi trong dữ liệu được truyền qua kết nối không đáng tin cậy như đường dây điện thoại. Ý tưởng cơ bản là một bit bổ sung được truyền sau mỗi nhóm 8 bit để có thể phát hiện được một lỗi bit trong quá trình truyền. Các bit chẵn lẻ có thể được tính toán cho parity lẻ hoặc parity chẵn. Nếu chọn parity chẵn thì bit chẵn lẻ được truyền đi sẽ được chọn sao cho tổng số bit 1 được truyền (8 bit dữ liệu cộng với bit chẵn lẻ) là chẵn. Khi chọn parity lẻ, bit chẵn lẻ được chọn sao cho tổng số bit 1 được truyền là số lẻ. Viết chương trình tính bit chẵn lẻ cho nhóm 8 bit được người dùng nhập vào bằng cách sử dụng parity chẵn. Chương trình của bạn sẽ đọc các chuỗi chứa 8 bit cho đến khi người dùng nhập một dòng trống. Sau khi người dùng nhập chuỗi, chương trình của bạn sẽ hiển thị một thông báo rõ ràng cho biết bit chẵn lẻ phải là 0 hay 1. Hiển thị thông báo lỗi thích hợp nếu người dùng nhập thứ gì đó không phải 8 bit.

***Solution:***

#hàm kiễm tra chuỗi nhâp vào theo parity chẵn

def calculate\_parity(input\_bits):

    #kiểm tra xem đầu vào có đúng 8 bit không

    if len(input\_bits) != 8:

        return "Đầu vào không phải là 8 bit"

    #kiểm tra xem chuỗi chỉ chứa các ký tự 0 và 1

    for bit in input\_bits:

        if bit not in ['0', '1']:

            return "Chuỗi chỉ được chứa các ký tự 0 hoặc 1"

    #đếm số lượng bit 1

    count\_ones = sum(int(bit) for bit in input\_bits)

    #nếu tổng số bit 1 là chẵn

    if count\_ones % 2 == 0:

        return f"Bit chẵn lẻ của {input\_bits} phải là 0"

    else:

        return f"Bit chẵn lẻ của {input\_bits} phải là 1"

#hàm dùng dể nhận input vào

def main\_process():

    while True:

        s = input("Nhập parity bit hoặc Enter để thoát: ")

        if s == "":

            break

        print(calculate\_parity(s))

#chương trình chính

main\_process()