<https://www.onlinegdb.com/online_python_compiler>

1. **В строке найти и заменить одну подстроку на другую. Если одинаковых подстрок несколько, заменить все.**

**Вариант 1**

str = "Tree, box, chair, lamp, desk, cat, dog, grass, pig, box, lamp, shelf"

print(str)

subStrOld = input("Old substring: ")

subStrNew = input("New substring: ")

lenStrOld = len(subStrOld)

while str.find(subStrOld) > 0:

i = str.find(subStrOld)

str = str[:i] + subStrNew + str[i+lenStrOld:]

print(str)

**Вариант 2**

mystr1=input('Введите строку символов ')

print(mystr1)

subStrOld = input("Old substring: ")

subStrNew = input("New substring: ")

lenStrOld = len(subStrOld)

while mystr1.find(subStrOld) > 0:

i = mystr1.find(subStrOld)

mystr1 = mystr1[:i] + subStrNew + mystr1[i+lenStrOld:]

print(mystr1)

1. **Сортировка выбором**

Алгоритм сортировки выбором заключается в поиске на необработанном срезе массива или списка минимального значения и в дальнейшем обмене этого значения с первым элементом необработанного среза. На следующем шаге необработанный срез уменьшается на один элемент.

Найти наименьшее значение в списке.

Записать его в начало списка, а первый элемент - на место, где раньше стоял наименьший.

Снова найти наименьший элемент в списке. При этом в поиске не участвует первый элемент.

Второй минимум поместить на второе место списка. Второй элемент при этом перемещается на освободившееся место.

Продолжать выполнять поиcк и обмен, пока не будет достигнут конец списка.

**Вариант 1**

# Заполняем список из 10 элементов

# случайными числами от 1 до 99 и

# выводим неотсортированный список на экран.

from random import randint

N = 10

arr = []

for i in range(N):

arr.append(randint(1, 99))

print(arr)

# В цикле переменная i хранит индекс ячейки,

# в которую записывается минимальный элемент.

# Сначала это будет первая ячейка.

i = 0

# N - 1, так как последний элемент

# обменивать уже не надо.

while i < N - 1:

# ПОИСК МИНИМУМА

# Сначала надо найти минимальное значение

# на срезе от i до конца списка.

# Переменная m будет хранить индекс ячейки

# с минимальным значением.

# Сначала предполагаем, что

# в ячейке i содержится минимальный элемент.

m = i

# Поиск начинаем с ячейки следующей за i.

j = i + 1

# Пока не дойдем конца списка,

while j < N:

# будем сравнивать значение ячейки j,

# со значением ячейки m.

if arr[j] < arr[m]:

# Если в j значение меньше, чем в m,

# сохраним в m номер найденного

# на данный момент минимума.

m = j

# Перейдем к следующей ячейке.

j += 1

# ОБМЕН ЗНАЧЕНИЙ

# В ячейку i записывается найденный минимум,

# а значение из ячейки i переносится

# на старое место минимума.

arr[i], arr[m] = arr[m], arr[i]

# ПЕРЕХОД К СЛЕДУЮЩЕЙ НЕОБРАБОТАННОЙ ЯЧЕЙКЕ

i += 1

# Вывод отсортированного списка

print(arr)

**Вариант 2**

from random import randint

def sel\_sort(array):

for i in range(len(array) - 1):

m = i

j = i + 1

while j < len(array):

if array[j] < array[m]:

m = j

j = j + 1

array[i], array[m] = array[m], array[i]

a = []

for i in range(10):

a.append(randint(1, 99))

a = [22, 7, 2, -5, 8, 4]

print(a)

sel\_sort(a)

print(a)

1. **Пузырьковая сортировка**

**Вариант 1**

from random import randint

N = 10

a = []

for i in range(N):

a.append(randint(1, 99))

print(a)

for i in range(N-1):

for j in range(N-i-1):

if a[j] > a[j+1]:

a[j], a[j+1] = a[j+1], a[j]

print(a)

**Вариант 2**

from random import randint

N = 10

a = []

for i in range(N):

a.append(randint(1, 99))

print(a)

i = 0

while i < N - 1:

j = 0

while j < N - 1 - i:

if a[j] > a[j+1]:

a[j], a[j+1] = a[j+1], a[j]

j += 1

i += 1

print(a)

**Вариант 3**

from random import randint

def bubble(array):

for i in range(N-1):

for j in range(N-i-1):

if array[j] > array[j+1]:

buff = array[j]

array[j] = array[j+1]

array[j+1] = buff

N = 10

a = []

for i in range(N):

a.append(randint(1, 99))

print(a)

bubble(a)

print(a)

**4.Сортировка Шелла**

«Сортировка Шелла». Этот алгоритм может рассматриваться и как обобщение пузырьковой сортировки, так и сортировки вставками.

Идея метода заключается в сравнение разделенных на группы элементов последовательности, находящихся друг от друга на некотором расстоянии. Изначально это расстояние равно d или N/2, где N — общее число элементов. На первом шаге каждая группа включает в себя два элемента расположенных друг от друга на расстоянии N/2; они сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается на d/2, и количество групп, соответственно, уменьшается. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.

def shell(seq):

inc = len(seq) // 2

while inc:

for i, el in enumerate(seq):

while i >= inc and seq[i - inc] > el:

seq[i] = seq[i - inc]

i -= inc

seq[i] = el

inc = 1 if inc == 2 else int(inc \* 5.0 / 11)

data = [22, 7, 2, -5, 8, 4]

print (data)

shell(data)

print (data )# [-5, 2, 4, 7, 8, 22]

**5.Доступ к столбцам матрицы**

import numpy as np

A = np.array([[1, 4, 5, 12],

[-5, 8, 9, 0],

[-6, 7, 11, 19]])

print(A)

print("A[:,0] =",A[:,0]) # Первый столбец

print("A[:,3] =", A[:,3]) # Четвертый столбец

print("A[:,-1] =", A[:,-1]) # Последний столбец (четвертый столбец)

**6.Нахождение максимального и минимального элементов в матрице**

import random

N=int(input("Введите размер квадратной матрицы:"))

arr=[[random.randint(1,9) for i in range(N)] for j in range(N)]

min = arr[0][0]

max = 0

for i in range(N):

for j in range(N):

if(arr[i][j]<min):

min=arr[i][j]

elif(arr[i][j]>max):

max=arr[i][j]

for i in range(N):

print(arr[i])

print(max, min)

**7. Генерируется квадратная матрица. Находится сумма элементов ее главной и побочной диагоналей.**

**#**Главная диагональ идет из верхнего левого угла в правый нижний, побочная - из верхнего правого угла в левый #нижний.

from random import random

N = 5

matrix = []

for i in range(N):

row = []

for j in range(N):

row.append(int(random()\*10))

matrix.append(row)

for row in matrix:

print(row)

sumMain = 0

sumSecondary = 0

for i in range(N):

sumMain += matrix[i][i]

sumSecondary += matrix[i][N-i-1]

print(sumMain)

print(sumSecondary)

**8. Транспонирование матрицы**

**Вариант 1**

from pprint import pprint

# модуль pprint используется для удобного вывода на экран

matrix = [[0.5, 0, 0, 0, 0],

[ 1, 0.5, 0, 0, 0],

[ 1, 1, 0.5, 0, 0],

[ 1, 1, 1, 0.5, 0],

[ 1, 1, 1, 1, 0.5]]

matrix\_t = list(zip(\*matrix)) # непосредственно транспонирование

pprint(matrix)

pprint(matrix\_t)

**Вариант 2**

import numpy as np

A = np.array([[1, 1], [2, 1], [3, -3]])

print(A)

print(A.transpose())

**9. Сложение двух матриц**

import numpy as np

A = np.array([[2, 4], [5, -6]])

B = np.array([[9, -3], [3, 6]])

print(A)

print(B)

C = A + B # сложение соответствующих элементов

print(C)

**10. Умножение двух матриц:**

**Вариант 1**

import numpy as np

A = np.array([[3, 6, 7], [5, -3, 0]])

B = np.array([[1, 1], [2, 1], [3, -3]])

print(A)

print(B)

C = A.dot(B)

print(C)

**11. Изменение формы массива**

import numpy as np

a=np.array([[[ 0, 1, 2],

[ 10, 12, 13]],

[[100, 101, 102],

[110, 112, 113]]])

a.ravel() # Делает массив плоским

#array([ 0, 1, 2, 10, 12, 13, 100, 101, 102, 110, 112, 113])

a.shape = (6, 2) # Изменение формы

print(a)

#array([[ 0, 1],

# [ 2, 10],

# [ 12, 13],

# [100, 101],

# [102, 110],

# [112, 113]])

print( a.transpose() ) # Транспонирование

#array([[ 0, 2, 12, 100, 102, 112],

# [ 1, 10, 13, 101, 110, 113]])

print( a.reshape((3, 4))) # Изменение формы

#array([[ 0, 1, 2, 10],

# [ 12, 13, 100, 101],

# [102, 110, 112, 113]])

**12.**

#Постройте матрицу смежности по заданным спискам смежности.

#В первой строке входных данных содержится

#число N - количество вершин (1?N?100). Далее идут N строк.

#В i-й строке содержится описание всех ребер, исходящих из i-й вершины.

#Описание начинается количеством исходящих ребер.

#Далее следуют номера вершин, в которые эти ребра идут.

#Все вершины нумеруются натуральными числами от 1 до N.

n=int(input())

Mt=[[0]\*n for i in range(n)]

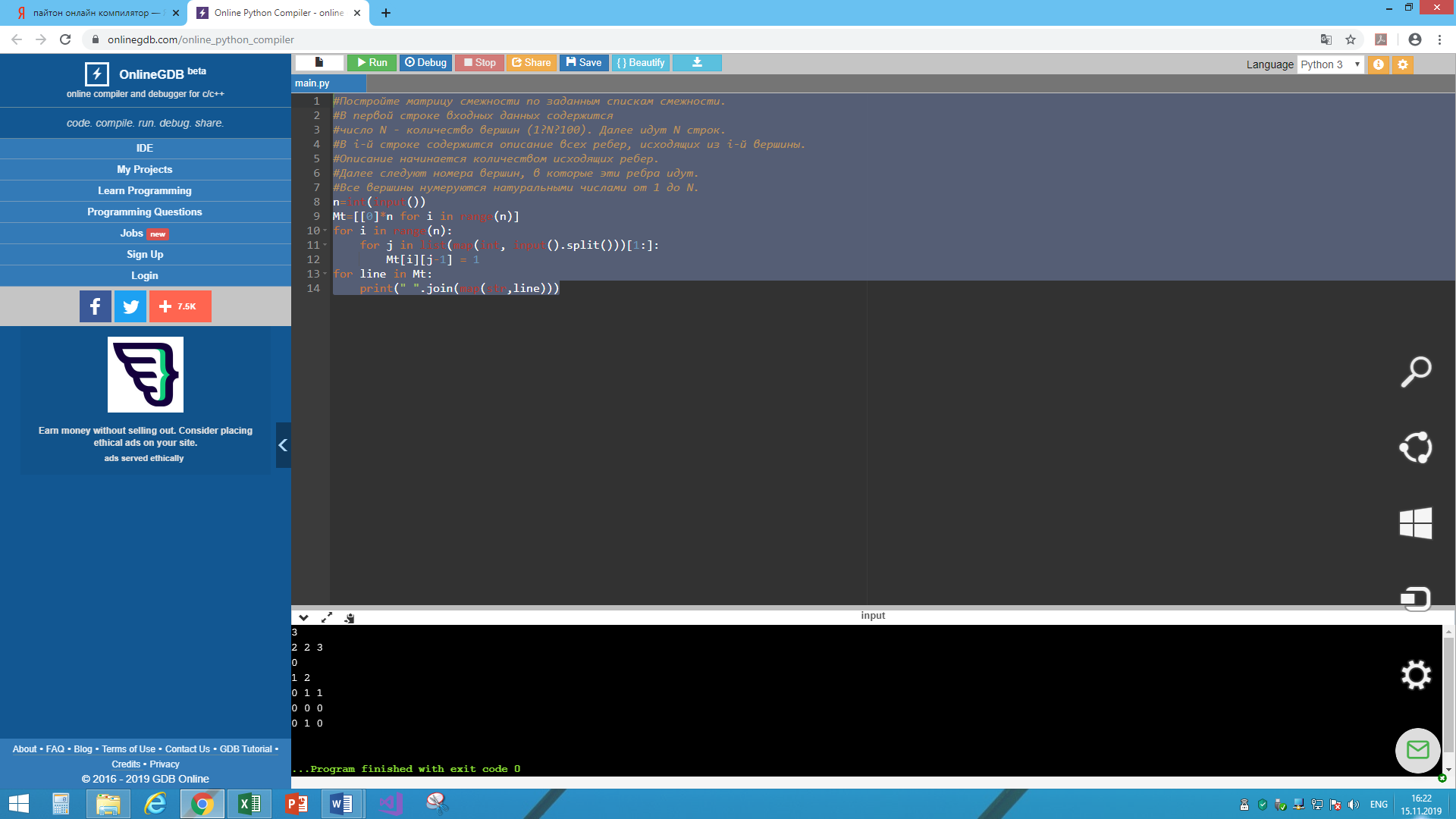
for i in range(n):

for j in list(map(int, input().split()))[1:]:

Mt[i][j-1] = 1

for line in Mt:

print(" ".join(map(str,line)))



**13. Связанная сортировка двух списков**

n = int(input("Количество пар элеменов списка"))

Sp = []

for I in range(n):

name,num= input().split()

Sp.append((int(num),name))

Sp.sort()

for elem in Sp:

print(elem[0], elem[1])

# Необходимо отсортировать элементы, которые представляют собой

# пару из номера группы и фамилии

**14.Алгоритм Флойда**

# Полный ориентированный взвешенный граф задан матрицей смежности.

# Постройте матрицу кратчайших путей между его вершинами.

# Гарантируется, что в графе нет циклов отрицательного веса.

#В первой строке вводится единственное число N (1?N?100) – количество вершин графа.

# В следующих N строках по N чисел задается матрица смежности графа

# (j-е число в i-й строке соответствует весу ребра из вершины i в вершину j).

# Все числа по модулю не превышают 100.

# На главной диагонали матрицы стоят нули.

n = int(input())

A = [list(map(int, input().split())) for i in range (n)]

for k in range(n):

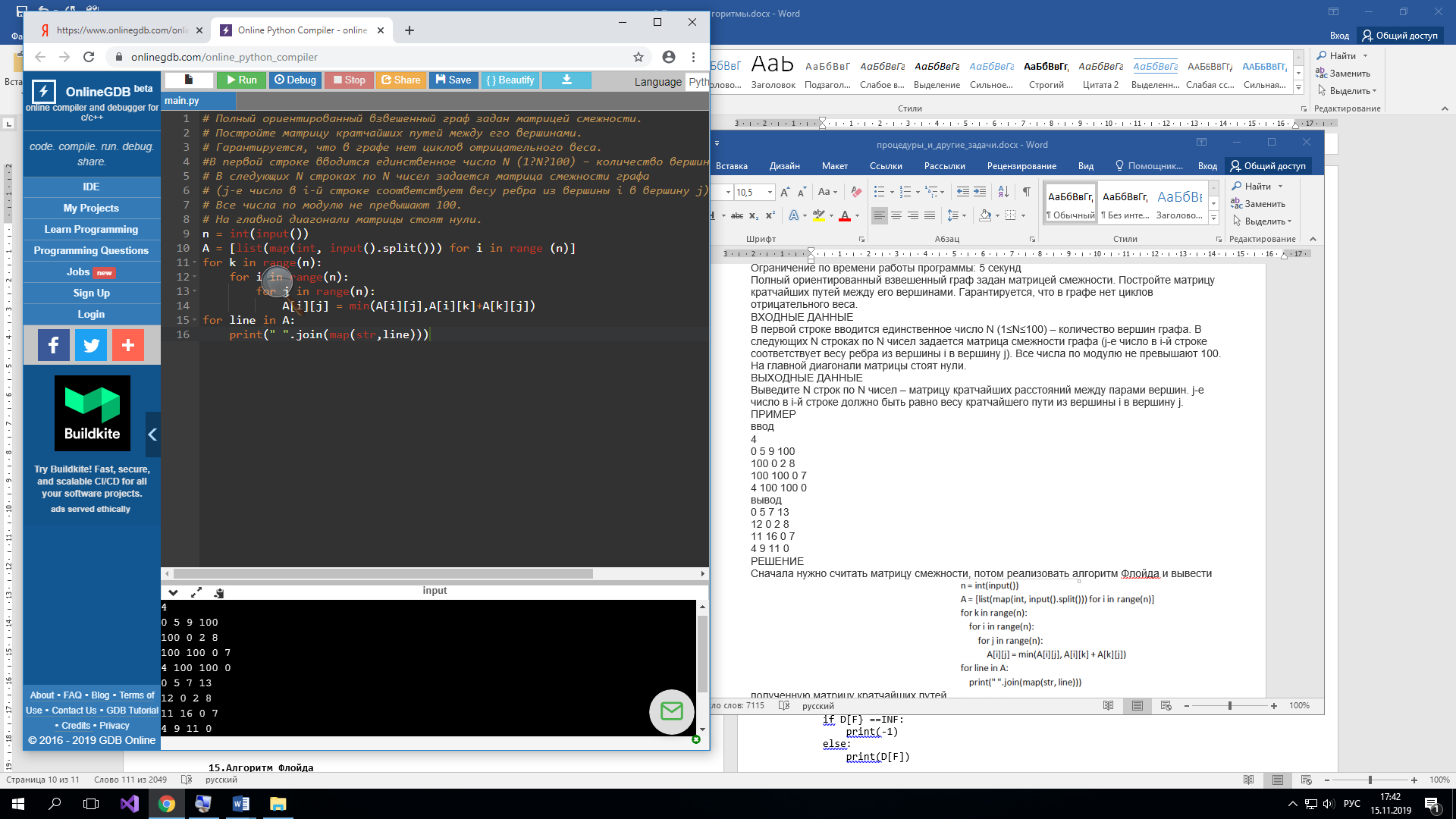
for i in range(n):

for j in range(n):

A[i][j] = min(A[i][j],A[i][k]+A[k][j])

for line in A:

print(" ".join(map(str,line)))



**15. Поиск кратчайшего пути в графе. Алгоритм Дейкстры**

# Дан ориентированный взвешенный граф.

#Найдите кратчайшее расстояние от одной заданной вершины до другой.

#В первой строке содержатся три числа: N, S и F (1?N?100, 1?S,F?N),

#где N — количество вершин графа, S — начальная вершина, а F – конечная.

#В следующих N строках вводится по N чисел,

#не превосходящих 100 — матрица смежности графа,

#где -1 означает отсутствие ребра между вершинами,

#а любое неотрицательное число — присутствие ребра данного веса.

#На главной диагонали матрицы записаны нули.

# Требуется вывести искомое расстояние или -1,

# если пути между указанными вершинами не существует.

N, S,F = map(int, input().split())

S-=1

F-=1

W=[]

for i in range(N):

W.append(list(map(int,input().split())))

INF = 10\*\*10

D = [INF]\* N

D[S] = 0

Colored = [False]\*N

while True:

min\_dist = INF

for i in range(N):

if not Colored[i] and D[i]<min\_dist:

min\_dist = D[i]

min\_vertex = i

if min\_dist ==INF:

break

i=min\_vertex

Colored[i]=True

for j in range(N):

if D[i]+W[i][j] <D[j] and W[i][j]!=-1:

D[j]=D[i]+W[i][j]

if D[F] ==INF:

print(-1)

else:

print(D[F])