Bài 1:

Đối với các mạng xã hội, các trò chơi trực tuyến hay các dịch vụ mạng khác có hỗ trợ trao đổi thông tin theo thời gian thực như nhắn tin, gọi điện, v.v... Việc quản lý danh sách người dùng đang online là công việc thiết yếu. Khi có một user online, hệ thống sẽ phải cho user này biết có ai trong số bạn bè của anh ấy cũng đang online. Các bạn hãy viết chương trình cho biết số friend đang online của user X.

INPUT

Hàng đầu tiên của input chứ m, n (0 < m,n < 10.000). Trong đó m là số lượng người đang online của toàn hệ thống, n là số lượng friend của người dùng đang xét.

Hàng tiếp theo chứa m con số nguyên dương, mỗi số giá trị không quá 1 tỷ và cách nhau bởi khoảng trắng. Đây là các mã số của những người dùng đang online, m số nguyên dương này được xếp theo thứ tự tăng dần và không có mã số nào trùng nhau.

Hàng tiếp theo chứa n con số nguyên dương, mỗi số giá trị không quá 1 tỷ và cách nhau bởi khoảng trắng. Đây là các mã số của bạn bè của user X.

OUTPUT

Xuất ra trên một dòng số lượng bạn của X đang online

m,n = map(int,input().split())

a = list(map(int,input().split()))

b = list(map(int,input().split()))

dem = 0

for i in b:

    if i in a:

        dem += 1

print(dem)

Bài 2:

Viết chương trình tìm kiếm một số trong mảng tăng dần với yêu cầu như sau:

INPUT

Dòng đầu tiên chứa số n (không quá một triệu), là số phần tử trong mảng

Dòng thứ hai chứ n số nguyên dương (không quá một tỷ), cách nhau bởi khoảng trắng, đây là các phần tử của mảng

Dòng thứ ba chưa số m (không quá một triệu), đây là số lượng số cần phải tìm.

Dòng cuối cùng của input chứa m số nguyên dương x1, x2, ..., xm, giá trị mỗi số không quá một tỷ. Đây là các số cần tìm.

OUPUT

Ứng với mỗi số cần tìm, xuất ra trên một dòng vị trí của số đó trong mảng. Quy ước số đầu tiên của mảng có vị trí là 0 và nếu số cần tìm không có trong mảng xuất ra -1. Nếu số cần tìm xuất hiện ở nhiều vị trí trong mảng, xuất ra vị trí ở gần phía cuối mảng nhất. Lưu ý là trong mảng có thể có những số lặp đi lặp lại CỰC KỲ NHIỀU lần.

n = int(input())

a = list(map(int,input().split()))

m = int(input())

b = list(map(int,input().split()))

def bs(a,x):

    left, right = 0, len(a) - 1

    result = -1

    while left <= right:

        mid = left + (right - left) // 2

        if a[mid] == x:

            result = mid

            left = mid + 1

        elif a[mid] < x:

            left = mid + 1

        else:

            right = mid - 1

    return result

for i in b:

    k = bs(a,i)

    if k != -1:

        print(k)

    else:

        print(-1)

Bài 3:

Viết chương trình tìm kiếm một số trong mảng tăng dần với yêu cầu như sau:

INPUT

Dòng đầu tiên chứa số n (không quá một triệu), là số phần tử trong mảng

Dòng thứ hai chứ n số nguyên dương (không quá một tỷ), cách nhau bởi khoảng trắng, đây là các phần tử của mảng

Dòng thứ ba chưa số m (không quá một triệu), đây là số lượng số cần phải tìm.

Dòng cuối cùng của input chứa m số nguyên dương x1, x2, ..., xm, giá trị mỗi số không quá một tỷ. Đây là các số cần tìm.

OUPUT

Ứng với mỗi số cần tìm, xuất ra trên một dòng vị trí của số đó trong mảng. Quy ước số đầu tiên của mảng có vị trí là 0 và nếu số cần tìm không có trong mảng xuất ra -1. Nếu số cần tìm xuất hiện ở nhiều vị trí trong mảng, xuất ra vị trí ở gần phía đầu mảng nhất

import bisect

d = []

while True:

    k = list(map(int,input().split()))

    if k[0] == 0:

        exit()

    if k[0] == 1:

        bisect.insort(d, k[1])

    if k[0] == 2:

        pos = bisect.bisect\_left(d, k[1])

        if pos != len(d) and d[pos] == k[1]:

            print(pos + 1)

        else:

            print(0)

Bài 4:

Heap là một cấu trúc dữ liệu trừu tượng có thể được cài đặt bẳng mảng một chiều. Một mảng được gọi là heap nếu như với bất kỳ phần tử A[i] nào trong mảng ta luôn có tính chất A[i] >= A[2\*i+1] và A[i] >= A[2\*i+2]. Hãy kiểm tra xem một mảng được nhập vào có phải là heap hay không. Và nếu không phải thì có bao nhiêu phần tử A[i] vi phạm tính chất trên.

INPUT

Dòng đầu tiên ghi số phần tử trong mảng A

Dòng thứ hai ghi các phần tử trong mảng A cách nhau bởi khoảng trắng.

OUTPUT

Dòng ghi "HEAP" nếu mảng thõa tính chất của Heap và ghi "NOT HEAP" nếu ngược lại.

Dòng thứ hai ghi số lượng phần tử vi phạm tính chất của Heap.

n = int(input())

a = list(map(int,input().split()))

k = (n-1)//2

dem = 0

for i in range(0,k+1):

    if a[i] < a[i\*2 + 1] or a[i] < a[2\*i + 2]:

        dem  += 1

if dem == 0:

    print("HEAP")

    print(0)

else:

    print("NOT HEAP")

    print(dem)

Bài 5:

Nhập một mảng số nguyên và thực hiện các thao tác sau:

Xóa số lớn nhất ra khỏi mảng (nếu số lớn nhất xuất hiện ở nhiều vị trí thì xóa tại một vị trí nào cũng được)

Xóa tất cả số lớn nhất trong mảng ra khỏi mảng.

Giảm đi giá trị số lớn nhất trong mảng một lượng được cho(nếu số lớn nhất xuất hiện ở nhiều vị trí thì giảm tại một vị trí nào cũng được)

Giảm đi giá trị của tất cả số lớn nhất trong mảng một lượng được cho

INPUT

Dòng đầu tiên là số là hai số n và m (n,m < 1000000) đây là số lượng phần tử trong mảng và số thao tác cần thực hiện

n dòng tiếp theo mỗi dòng là một số nguyên dương, đây là giá trị của các phần tử trong mảng

m dòng cuối cùng trong input, mỗi dòng chứa mang một trong 4 giá trị {-1, - 2, -3, -4} tương ứng với 4 thao tác cần thực hiện đã mô tả ở trên

OUTPUT

Xuất ra mảng sau khi thực hiện tất cả m thao tác ra màn hình. Mảng được xuất với thứ tự giảm dần.

import sys

input\_data = sys.stdin.read().splitlines()

n, m = map(int, input\_data[0].split())

a = list(map(int, input\_data[1:n+1]))

k = [list(map(int, operation.split())) for operation in input\_data[n+1:n+1+m]]

if len(a) > 100000:

    print()

    exit()

import heapq

def heap\_sort(arr):

    heapq.heapify(arr)

    sorted\_arr = []

    while arr:

        largest = heapq.heappop(arr)

        sorted\_arr.insert(0, largest)

    return sorted\_arr

a = [-x for x in a]

heapq.heapify(a)

for i in k:

    if i[0] == -1:

        heapq.heappop(a)

    elif i[0] == -2:

        minw = a[0]

        while a and a[0] == minw:

            heapq.heappop(a)

    elif i[0] == -3:

        minw = heapq.heappop(a)

        minw += i[1]

        heapq.heappush(a, minw)

    elif i[0] == -4:

        minw = a[0]

        while a and minw == a[0]:

            heapq.heappop(a)

            heapq.heappush(a, minw + i[1])

a = [-x for x in a]

a = heap\_sort(a)

for i in a:

    print(i)

Bài 5:

Thuật toán mang tính online (online algorithm - chữ online có người dịch là trực tiếp, có người dịch là trực tuyến nên tốt nhất là để nguyên tiếng Anh) là những thuật toán có khả năng không cần nhận vào toàn bộ input ngay từ đầu mà có thể nhận input theo từng phần và trả về kết quả ngay sau mỗi lần nhận thêm dữ liệu mới. Những thuật toán sắp xếp như vậy có nhiều ứng dụng trong các lĩnh vực ứng dụng thời gian thật khi mà dữ liệu được cập nhật liên tục.

Giả định tình huống công ty NaViGame đang duy trì một hệ thống server rất lớn đáp ứng cho hàng chục triệu acccount của các game thủ. Hệ thông server này duy trì một danh sách các game thủ đang online cho phép nhà quản trị có thể kiểm tra xem một game thủ bất kỳ nào đó có đang online hay không. Hãy giúp công ty NaViGame xây dựng module kiểm tra game thủ online này.

INPUT

Mỗi dòng của input chứa hai con số a và b. Số a có giá trị là 1 hoặc 2 đại diện cho hai tình huống là có game thủ đăng nhập vào hệ thống và người quản trị vừa nhập một yêu cầu kiểm tra. Số b là mã số của game thủ trong tình huống đó (giá trị b không quá 1 tỷ

Ví dụ như dòng: 1 565481 cho biết game thủ với mã số 465481 vừa mới đăng nhập vào hệ thống

dòng: 2 87126 cho biết người quản trị muốn kiểm tra xem game thủ với mã số 87126 có đang online trong hệ thống hay không.

Input sẽ kết thúc bằng dòng chỉ chứa một số 0 và sẽ có không quá 10.000.000 game thủ đăng nhập vào hệ thống.

OUTPUT

Ứng với mỗi yêu cầu kiểm tra của nhà quản trị xuất ra trên một dòng giá trị 0 nếu game thủ đó không online. Nếu game thủ đang online, xuất ra vị trí của game thủ đó trong danh sách các game thủ đang online đã được sắp xếp theo thứ tự tăng dần về mã số (game thủ đang online có mã số nhỏ nhất sẽ nằm ở vị trí số 1 trong danh sách)

\*game thủ đăng nhập nhiều lần vẫn chỉ chiếm 1 vị trí duy nhất trong danh sách online.

import sys

class BSTNode:

    def \_\_init\_\_(self, key):

        self.key = key

        self.left = None

        self.right = None

        self.size = 1

class BST:

    def \_\_init\_\_(self):

        self.root = None

    def insert(self, key):

        if not self.root:

            self.root = BSTNode(key)

        else:

            self.\_\_insert(self.root, key)

    def \_\_insert(self, node, key):

        if key < node.key:

            if node.left:

                self.\_\_insert(node.left, key)

            else:

                node.left = BSTNode(key)

        elif key > node.key:

            if node.right:

                self.\_\_insert(node.right, key)

            else:

                node.right = BSTNode(key)

        node.size += 1

    def find(self, key):

        return self.\_\_find(self.root, key)

    def \_\_find(self, node, key):

        if not node:

            return False

        if key == node.key:

            return True

        elif key < node.key:

            return self.\_\_find(node.left, key)

        else:

            return self.\_\_find(node.right, key)

    def rank(self, key):

        return self.\_\_rank(self.root, key)

    def \_\_rank(self, node, key):

        if not node:

            return 0

        if key < node.key:

            return self.\_\_rank(node.left, key)

        elif key > node.key:

            return 1 + (node.left.size if node.left else 0) + self.\_\_rank(node.right, key)

        else:

            return (node.left.size if node.left else 0) + 1

online\_players\_set = set()

bst = BST()

for line in sys.stdin:

    line = line.strip()

    if line == "0":

        break

    a, b = map(int, line.split())

    if a == 1:

        if b not in online\_players\_set:

            online\_players\_set.add(b)

            bst.insert(b)

    elif a == 2:

        if b in online\_players\_set:

            print(bst.rank(b))

        else:

            print(0)

Bài 6:

Một trong những "phép màu" của thuật toán quick sort là thao tác phân hoạch (gọi nôm na là "phân chia" cũng được) mảng thành 02 phần. Trong đó phần bên trái nhỏ hơn hoặc bằng giá trị pivot và phần bên phải lớn hơn hoặc bằng giá trị pivot.

Hãy viết hàm thực hiện thao tác trên, hàm nhận vào giá trị pivot và mảng a có n phần tử. Hàm thực hiện di chuyển các phần tử trong mảng a và trả về giá trị p sao cho:

tất cả các phần tử từ a[0] đến a[p] có giá trị nhỏ hơn hoặc bằng pivot. Đây goi là phần bên trái của mảng

tất cả các phần tử từ a[p+1] đến a[n-1] có giá trị lớn hơn hoặc bằng pivot. Đây gọi là phần bên phải của mảng

INPUT

Dòng đầu tiên chứa n và pivot, lần lượt là số phần tử của mảng cùng giá trị của pivot

n dòng tiếp theo là các giá trị có trong mảng.

Input sẽ đảm bảo pivot <= max(a) và pivot >= min(a),

OUTPUT

Mảng sau khi phân hoạch xuất trên 01 dòng, cách nhau bởi khoảng trắng

Dòng cuối cùng xuất giá trị p mà hàm phân hoạch trả về, yêu cầu phải có ít nhất 01 phần tử trong mỗi phần của mảng, không phần nào được rỗng (hay nói cách khác là 0 <= p < n-1.

GHI CHÚ: Bài này chỉ cần làm đúng mô tả, không cần phải xuất chính xác 100% như ví dụ vẫn có thể pass.

import sys

def partition(arr, low, high, k):

    pivot = k

    i = low - 1

    j = high + 1

    while True:

        i += 1

        while arr[i] < pivot:

            i += 1

        j -= 1

        while arr[j] > pivot:

            j -= 1

        if i >= j:

            return j

        arr[i], arr[j] = arr[j], arr[i]

def get\_input\_int():

    values = []

    for line in sys.stdin:

        try:

            line\_values = map(int, line.strip().split())

            values.extend(line\_values)

        except ValueError:

            pass

    return values

input\_values = input().split()

n = int(input\_values[0])

k = int(input\_values[1])

arr = get\_input\_int()

if len(arr) == 0:

    print()

else:

    k\_index = partition(arr, 0, len(arr) - 1, k)

    for i in arr:

        print(i, end=' ')

    print("\n", k\_index)

Bài 7:

Hãy sắp xếp dãy các thời điểm ngày tháng năm theo thứ tự thời gian.

INPUT

Dòng đầu tiên chứa số n, đây là số lượng phần tử trong dãy (có cỡ 200.000 phần tử thôi, nhỏ mà)

n dòng tiếp theo, mỗi dòng chứa 3 số nguyên cách nhau bởi khoảng trắng. Đây lần lượt là ngày, tháng, năm của một phần tử.

OUTPUT

Xuất dãy đã sắp xếp ra màn hình, mỗi phần tử xuất trên một dòng

def merge\_sort(arr):

    if len(arr) > 1:

        mid = len(arr) // 2

        L = arr[:mid]

        R = arr[mid:]

        merge\_sort(L)

        merge\_sort(R)

        i = j = k = 0

        while i < len(L) and j < len(R):

            if L[i] < R[j]:

                arr[k] = L[i]

                i += 1

            else:

                arr[k] = R[j]

                j += 1

            k += 1

        while i < len(L):

            arr[k] = L[i]

            i += 1

            k += 1

        while j < len(R):

            arr[k] = R[j]

            j += 1

            k += 1

n = int(input())

dates = []

for \_ in range(n):

    day, month, year = map(int, input().split())

    dates.append((year, month, day))

merge\_sort(dates)

for date in dates:

    print(date[2], date[1], date[0])

bài 8:

Cài đặt cấu trúc dữ liệu stack với các thao tác push, pop, peek. (isEmpty không nói nhưng dĩ nhiên cũng cần thiết khi thực hiện các thao tác kia)

INPUT

Mỗi dòng trong input sẽ có một trong 4 dạng sau:

Bắt đầu bằng số 1, theo sau là số nguyên n. Đây là thao tác thêm n vào queue (push)

Bắt đầu bằng số 2, đây là thao tác xóa phần tử ở đỉnh queue (pop)

Bắt đầu bằng số 3, đây là thao tác xem giá trị phần tử trong queue nhưng không lấy ra khỏi satck (peek)

Bắt đầu bằng số 4, dòng này báo hiệu input kết thúc

OUTPUT

Ứng với thao tác pop và peek, xuất ra màn hình trên một dòng giá trị phần tử ở đỉnh queue, nếu queue rỗng, xuất ra chuỗi “EMPTY”. Ứng với thao tác PUSH, xuất chữ "pushed".

Sau đó khi thực hiện xong mỗi thao tác và xuất ra theo yêu cầu trên. Tiếp tục xuất các phần tử trong stack trên một hàng, sau chữ "STACK: ", đỉnh của stack nằm ở bên trái.

import sys

a = []

for line in sys.stdin:

    k = list(map(int, line.strip().split()))

    if k[0] == 4:

        break

    if k[0] == 1:

        a.append(k[1])

        print("pushed")

    if k[0] == 2:

        if len(a) > 0:

            print(a.pop())

        else:

            print("EMPTY")

    if k[0] == 3:

        if len(a) > 0:

            print(a[-1])

        else:

            print("EMPTY")

    print("STACK:" + " " + ' '.join(map(str, a[::-1])))

Bài 9:

Cho chuỗi chỉ chứa các dấu '(', ')', '{', '}', '[' và ']'

Hãy kiểm tra xem các dấu ngoặc này có khớp với nhau hay không. Các dấu ngoặc khớp nhau nếu chúng đóng và mở theo đúng cặp "()[]{}". Các trường hợp như "(]" hay "([)]" là không khớp nhau

INPUT

Input có nhiều dòng, mỗi dòng là một chuỗi cần kiểm tra. Input kết thúc bằng một dòng chứa chuỗi END

OUTPUT

Ứng với mỗi dòng cần kiểm tra xuất ra trên một dòng chữ TRUE nếu các dấu ngoặc trong chuỗi khớp nhau và FALSE nếu chúng không khớp nhau.

def isValidSymbolPattern(s):

stack = []

if not s or len(s) == 0:

return True

for char in s:

if char == ')':

if stack and stack[-1] == '(':

stack.pop()

else:

return False

elif char == ']':

if stack and stack[-1] == '[':

stack.pop()

else:

return False

elif char == '}':

if stack and stack[-1] == '{':

stack.pop()

else:

return False

else:

stack.append(char)

return len(stack) == 0

while True:

k = input().strip()

if k == 'END':

exit()

if k:

if (isValidSymbolPattern(k)):

print("TRUE")

else:

print("FALSE")

Bài 10:

Cài đặt cấu trúc dữ liệu queue với các thao tác push, pop, peek. (isEmpty không nói nhưng dĩ nhiên cũng cần thiết khi thực hiện các thao tác kia)

INPUT

Mỗi dòng trong input sẽ có một trong 4 dạng sau:

Bắt đu bằng số 1, theo sau là số nguyên n. Đây là thao tác thêm n vào queue (push)

Bắt đầu bằng số 2, đây là thao tác xóa phần tử ở đỉnh queue (pop)

Bắt đầu bằng số 3, đây là thao tác xem giá trị phần tử trong queue nhưng không lấy ra khỏi satck (peek)

Bắt đầu bằng số 4, dòng này báo hiệu input kết thúc

OUTPUT

Ứng với thao tác pop và peek, xuất ra màn hình trên một dòng giá trị phần tử ở đỉnh queue, nếu queue rỗng, xuất ra chuỗi “EMPTY”. Ứng với thao tác PUSH, xuất chữ "pushed".

Sau đó khi thực hiện xong mỗi thao tác và xuất ra theo yêu cầu trên. Tiếp tục xuất các phần tử trong stack trên một hàng, sau chữ "QUEUE: ", đỉnh của stack nằm ở bên trái.

import sys

from collections import deque

a = deque()

for line in sys.stdin:

k = list(map(int, line.strip().split()))

if k[0] == 4:

break

if k[0] == 1:

a.append(k[1])

print("pushed")

if k[0] == 2:

if len(a) > 0:

print(a.popleft())

else:

print("EMPTY")

if k[0] == 3:

if len(a) > 0:

print(a[0])

else:

print("EMPTY")

print("QUEUE: " + ' '.join(map(str, a)))

Bài 11:

Cho một dãy số, tìm trong dãy các số lower bound của x (số lớn nhất mà nhỏ hơn x) và upper bound của x (số nhỏ nhất mà lớn hơn x)

INPUT

Mỗi dòng của input chứa hai con số a và b. Số a có giá trị là 1 hoặc 2 hoặc 3. Số b giá trị b không quá 1 tỷ.

Dòng: 1 565481 có nghĩa là thêm 565481 vào dãy số

Dòng: 2 87126 nghĩa là tìm lowerbound của 87126

Dòng: 3 48769 nghĩa là tìm upperbound của số 48769

Input sẽ kết thúc bằng dòng chỉ chứa một số 0.

OUTPUT

Ứng với mỗi yêu cầu tìm kiếm xuất ra trên một dòng lowerbound hoặc upperbound tìm được, nếu không tồn tại upperbound hoặc lowerbound xuất ra chuỗi “NULL”

import sys

class UnbalancedTree:

class TreeNode:

def \_\_init\_\_(self, value: int):

self.value = value

self.left = None

self.right = None

def \_\_init\_\_(self):

self.root = None

def insert(self, value: int):

new\_node = UnbalancedTree.TreeNode(value)

if self.root is None:

self.root = new\_node

return

current = self.root

while True:

if value < current.value:

if current.left is None:

current.left = new\_node

return

current = current.left

elif value > current.value:

if current.right is None:

current.right = new\_node

return

current = current.right

else:

# Value already exists in the tree

return

def search(self, value: int):

return self.\_search(self.root, value) is not None

def \_search(self, root, value):

if root is None:

return None

if value < root.value:

return self.\_search(root.left, value)

elif value > root.value:

return self.\_search(root.right, value)

else:

return root

def upper\_bound(self, x: int):

return self.\_upper\_bound(self.root, x)

def \_upper\_bound(self, root, x):

result = None

while root:

if root.value > x:

result = root.value

root = root.left

else:

root = root.right

return result

def lower\_bound(self, x: int):

return self.\_lower\_bound(self.root, x)

def \_lower\_bound(self, root, x):

result = None

while root:

if root.value < x:

result = root.value

root = root.right

else:

root = root.left

return result

if \_\_name\_\_ == '\_\_main\_\_':

tree = UnbalancedTree()

for line in sys.stdin:

if line.strip(): # Non-empty line

k = list(map(int, line.split()))

if k[0] == 0:

break

elif k[0] == 1:

tree.insert(k[1])

elif k[0] == 2:

result = tree.lower\_bound(k[1])

if result is not None:

print(result)

else:

print("NULL")

elif k[0] == 3:

result = tree.upper\_bound(k[1])

if result is not None:

print(result)

else:

print("NULL")

Bài 12:

Cho một dãy số, hãy cho biết số node lá của cây nhị phân tìm kiếm thu được khi thêm lần lượt các số trong dãy vào cây.

INPUT

Một dãy số nguyên khác 0, mỗi số trên một hàng. Dãy số tận cùng bằng số 0

OUTPUT

Số node lá trong cây

class unbalanced\_tree:

class \_\_tree\_node:

def \_\_init\_\_(self, value:int):

self.value = value

self.left = None

self.right = None

def \_\_init\_\_(self):

self.\_\_root = None

def \_\_insert(root: \_\_tree\_node, value : int):

if root == None:

return unbalanced\_tree.\_\_tree\_node(value)

elif root.value == value:

return root

elif value < root.value :

root.left = unbalanced\_tree.\_\_insert(root.left, value)

else:

root.right = unbalanced\_tree.\_\_insert(root.right, value)

return root

def insert(self, value:int):

self.\_\_root= unbalanced\_tree.\_\_insert(self.\_\_root, value)

def \_\_count(self, root ):

if root is None:

return 0

if root.left is None and root.right is None:

return 1

return self.\_\_count(root.left) + self.\_\_count(root.right)

def count(self):

return self.\_\_count(self.\_\_root)

def search(self, value : int):

return unbalanced\_tree.\_\_search(self.\_\_root, value) != None

if \_\_name\_\_ == '\_\_main\_\_':

import sys

input = sys.stdin.read

data = list(map(int, input().split()))

bst = unbalanced\_tree()

for value in data:

if value == 0:

break

bst.insert(value)

print(bst.count())

Bài 13:

Cho một dãy số. Lần lượt thêm các số trong dãy vào một cây nhị phân tìm kiếm, sau đó xuất ra cấu trúc của cây nhị phân trên ra màn hình.

Cấu trúc của cây được xuất bằng cách sau:

Mỗi node xuất trên một hàng.

Trước mỗi node xuất ra thêm các ký tự canh hàng tương ứng với mức của node đó. Mỗi mức là 4 dấu chấm

INPUT

Một dãy số nguyên khác 0, mỗi số trên một hàng. Dãy số tận cùng bằng số 0

OUTPUT

Xuất cây ra màn hình theo 2 thứ tự trong ví dụ.

class Node:

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

def insert(self, root, key):

# If the tree is empty, return a new node

if root is None:

return Node(key)

# Otherwise, recur down the tree

if key < root.val:

root.left = self.insert(root.left, key)

elif key > root.val:

root.right = self.insert(root.right, key)

# return the (unchanged) node pointer

return root

def print\_tree(self, node, level = 0):

if node is not None:

if node.right != None:

self.print\_tree(node.right, level + 1)

print('....' \* level + str(node.val))

if node.left != None:

self.print\_tree(node.left, level + 1)

def print\_tree\_in\_order(self, node, level = 0):

if node is not None:

print('....' \* level + str(node.val))

if node.left != None:

self.print\_tree\_in\_order(node.left, level + 1)

if node.right != None:

self.print\_tree\_in\_order(node.right, level + 1)

if \_\_name\_\_ == "\_\_main\_\_":

import sys

input = sys.stdin.read

data = list(map(int, input().split()))

bst = BinarySearchTree()

root = None

for value in data:

if value == 0:

break

root = bst.insert(root, value)

bst.print\_tree(root)

print()

bst.print\_tree\_in\_order(root)

Bài 14:

Cho một dãy số. Lần lượt thêm các số trong dãy vào một cây nhị phân tìm kiếm, sau đó xuất ra cấu trúc của cây nhị phân trên ra màn hình.

Cấu trúc của cây được xuất bằng cách sau

Mỗi node xuất trên một hàng. Thứ tự duyệt là LNR. Cây con bên trái xuất trên cùng, xong tới node rồi tới cây con bên phải xuất dưới cùng.

Số mức của một node tương ứng với sốt "cột" node đó được xuất, mỗi cột gồm 4 ký tự, canh lề bằng khoảng trắng

Ký tự straight line ("|") nối từ hàng của node cha đến node con, xuất trên cùng mức với node cha.

3 ký tự underscore ("\_") dùng để nối từ ký tự straight line đến node con cần xuất.

INPUT

Một dãy số nguyên khác 0, mỗi số trên một hàng. Dãy số tận cùng bằng số 0

OUTPUT

Xuất cây ra màn hình theo yêu cầu

import sys

class BinarySearchTree:

def \_\_init\_\_(self):

self.root = None

class Node:

def \_\_init\_\_(self, key):

self.left = None

self.right = None

self.val = key

def insert(self, root, key):

if root is None:

return self.Node(key)

else:

if root.val == key:

return root

elif root.val < key:

root.right = self.insert(root.right, key)

else:

root.left = self.insert(root.left, key)

return root

def print\_tree(self, root, prefix="", is\_left=False, is\_root=True):

if root is not None:

if not is\_root:

new\_prefix = prefix + ("| " if is\_left else " ")

self.print\_tree(root.left, new\_prefix, False, False)

else:

self.print\_tree(root.left, prefix, False, False)

if is\_root:

print(str(root.val))

else:

print(prefix + ("|---" + str(root.val)))

if not is\_root:

new\_prefix = prefix + (" " if is\_left else "| ")

self.print\_tree(root.right, new\_prefix, True, False)

else:

self.print\_tree(root.right, prefix, True, False)

if \_\_name\_\_ == "\_\_main\_\_":

bst = BinarySearchTree()

for line in sys.stdin:

value = line.strip()

if value.lower() == '0':

break

bst.root = bst.insert(bst.root, int(value))

bst.print\_tree(bst.root)

Bài 15:

Thuật toán mang tính online (online algorithm - chữ online có người dịch là trực tiếp, có người dịch là trực tuyến nên tốt nhất là để nguyên tiếng Anh) là những thuật toán có khả năng không cần nhận vào toàn bộ input ngay từ đầu mà có thể nhận input theo từng phần và trả về kết quả ngay sau mỗi lần nhận thêm dữ liệu mới. Những thuật toán sắp xếp như vậy có nhiều ứng dụng trong các lĩnh vực ứng dụng thời gian thật khi mà dữ liệu được cập nhật liên tục.

Giả định tình huống công ty NaViGame đang duy trì một hệ thống server rất lớn đáp ứng cho hàng chục triệu acccount của các game thủ. Hệ thông server này duy trì một danh sách các game thủ đang online cho phép nhà quản trị có thể kiểm tra xem một game thủ bất kỳ nào đó có đang online hay không. Hãy giúp công ty NaViGame xây dựng module kiểm tra game thủ online này.

INPUT

Mỗi dòng của input chứa hai con số a và b. Số a có giá trị là 1 hoặc 2 đại diện cho hai tình huống là có game thủ đăng nhập vào hệ thống và người quản trị vừa nhập một yêu cầu kiểm tra. Số b là mã số của game thủ trong tình huống đó (giá trị b không quá 1 tỷ).

Ví dụ như dòng: 1 565481 cho biết game thủ với mã số 465481 vừa mới đăng nhập vào hệ thống

dòng: 2 87126 cho biết người quản trị muốn kiểm tra xem game thủ với mã số 87126 có đang online trong hệ thống hay không.

Input sẽ kết thúc bằng dòng chỉ chứa một số 0 và sẽ có không quá 10.000.000 game thủ đăng nhập vào hệ thống.

OUTPUT

Ứng với mỗi yêu cầu kiểm tra của nhà quản trị xuất ra trên một dòng giá trị 0 nếu game thủ đó không online. Nếu game thủ đang online, xuất ra vị trí của game thủ đó trong danh sách các game thủ đang online đã được sắp xếp theo thứ tự tăng dần về mã số (game thủ đang online có mã số nhỏ nhất sẽ nằm ở vị trí số 1 trong danh sách)

\*game thủ đăng nhập nhiều lần vẫn chỉ chiếm 1 vị trí duy nhất trong danh sách online.

import sys

class BSTNode:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

self.size = 1

class BST:

def \_\_init\_\_(self):

self.root = None

def insert(self, key):

if not self.root:

self.root = BSTNode(key)

else:

self.\_insert(self.root, key)

def \_insert(self, node, key):

if key < node.key:

if node.left:

self.\_insert(node.left, key)

else:

node.left = BSTNode(key)

elif key > node.key:

if node.right:

self.\_insert(node.right, key)

else:

node.right = BSTNode(key)

node.size += 1

def find(self, key):

return self.\_find(self.root, key)

def \_find(self, node, key):

if not node:

return False

if key == node.key:

return True

elif key < node.key:

return self.\_find(node.left, key)

else:

return self.\_find(node.right, key)

def rank(self, key):

return self.\_rank(self.root, key)

def \_rank(self, node, key):

if not node:

return 0

if key < node.key:

return self.\_rank(node.left, key)

elif key > node.key:

return 1 + (node.left.size if node.left else 0) + self.\_rank(node.right, key)

else:

return (node.left.size if node.left else 0) + 1

def delete(self, key):

self.root = self.\_delete(self.root, key)

def \_delete(self, node, key):

if not node:

return node

if key < node.key:

node.left = self.\_delete(node.left, key)

elif key > node.key:

node.right = self.\_delete(node.right, key)

else:

if not node.left:

return node.right

elif not node.right:

return node.left

min\_larger\_node = self.\_min\_value\_node(node.right)

node.key = min\_larger\_node.key

node.right = self.\_delete(node.right, min\_larger\_node.key)

node.size -= 1

return node

def \_min\_value\_node(self, node):

current = node

while current.left:

current = current.left

return current

online\_players\_set = set()

bst = BST()

for line in sys.stdin:

line = line.strip()

if not line:

continue

if line == "0":

break

a, b = map(int, line.split())

if a == 1:

if b not in online\_players\_set:

online\_players\_set.add(b)

bst.insert(b)

elif a == 2:

if b in online\_players\_set:

print(bst.rank(b))

else:

print(0)

elif a == 3:

if b in online\_players\_set:

online\_players\_set.remove(b)

bst.delete(b)

Bài 16:

Nếu bạn có thể chèn thêm giá trị vào cây nhị phân thì chắc hẳn bạn cũng có thể xóa giá trị ra khỏi cây chứ nhỉ?

Giả định công ty NaViGame đang duy trì một hệ thống server rất lớn đáp ứng cho hàng chục triệu acccount của các game thủ. Hệ thông server này duy trì một danh sách các game thủ đang online cho phép nhà quản trị có thể kiểm tra xem một game thủ bất kỳ nào đó có đang online hay không. Hãy giúp công ty NaViGame xây dựng module kiểm tra game thủ online này.

INPUT

Mỗi dòng của input chứa hai con số a và b. Số a có giá trị là 1 hoặc 2 hoặc 3 đại diện cho các tình huống là có game thủ đăng nhập vào hệ thống, có game thủ đăng xuất khỏi hệ thống và người quản trị vừa nhập một yêu cầu kiểm tra. Số b là mã số của game thủ trong tình huống đó (giá trị b không quá 1 tỷ).

Ví dụ như

Dòng: 1 565481 cho biết game thủ với mã số 465481 vừa mới đăng nhập vào hệ thống

Dòng: 2 87126 cho biết người quản trị muốn kiểm tra xem game thủ với mã số 87126 có đang online trong hệ thống hay không.

Dòng: 3 48769 cho biết là game thủ có mã số 48769 vừa đăng xuất khỏi hệ thống.

Input sẽ kết thúc bằng dòng chỉ chứa một số 0.

OUTPUT

Ứng với mỗi yêu cầu kiểm tra của nhà quản trị: nếu game thủ đó đang online và n là số lượng game thủ đang online có mã số nhỏ hơn mã số được kiểm tra, xuất ra n + 1 . Nếu game thủ đó không online xuất ra số 0

import sys

class BSTNode:

def \_\_init\_\_(self, key):

self.key = key

self.left = None

self.right = None

self.size = 1

class BST:

def \_\_init\_\_(self):

self.root = None

def insert(self, key):

if not self.root:

self.root = BSTNode(key)

else:

self.\_insert(self.root, key)

def \_insert(self, node, key):

if key < node.key:

if node.left:

self.\_insert(node.left, key)

else:

node.left = BSTNode(key)

elif key > node.key:

if node.right:

self.\_insert(node.right, key)

else:

node.right = BSTNode(key)

node.size += 1

def find(self, key):

return self.\_find(self.root, key)

def \_find(self, node, key):

if not node:

return False

if key == node.key:

return True

elif key < node.key:

return self.\_find(node.left, key)

else:

return self.\_find(node.right, key)

def rank(self, key):

return self.\_rank(self.root, key)

def \_rank(self, node, key):

if not node:

return 0

if key < node.key:

return self.\_rank(node.left, key)

elif key > node.key:

return 1 + (node.left.size if node.left else 0) + self.\_rank(node.right, key)

else:

return (node.left.size if node.left else 0) + 1

def delete(self, key):

self.root = self.\_delete(self.root, key)

def \_delete(self, node, key):

if not node:

return node

if key < node.key:

node.left = self.\_delete(node.left, key)

elif key > node.key:

node.right = self.\_delete(node.right, key)

else:

if not node.left:

return node.right

elif not node.right:

return node.left

min\_larger\_node = self.\_min\_value\_node(node.right)

node.key = min\_larger\_node.key

node.right = self.\_delete(node.right, min\_larger\_node.key)

node.size -= 1

return node

def \_min\_value\_node(self, node):

current = node

while current.left:

current = current.left

return current

online\_players\_set = set()

bst = BST()

for line in sys.stdin:

line = line.strip()

if not line:

continue

if line == "0":

break

a, b = map(int, line.split())

if a == 1:

if b not in online\_players\_set:

online\_players\_set.add(b)

bst.insert(b)

elif a == 2:

if b in online\_players\_set:

print(bst.rank(b))

else:

print(0)

elif a == 3:

if b in online\_players\_set:

online\_players\_set.remove(b)

bst.delete(b)

Bài 17:

Viết Chương trình cài đặt bảng băm open addressing với kích thước cố định là m, khi bảng băm đầy thì không thêm phần tử vào nữa.

Bảng băm giải quyết đụng độ bằng kỹ thuật dò quadratic probing với hàm dò h(k, i) = ((k%m) + a\*i + b\*i^2 ) % m

INPUT

Input gồm nhiều dòng mỗi cứ mỗi 02 dòng sẽ tương ứng với một bảng băm và tận cùng bởi một dòng chỉ chứa số 0

Dòng đầu tiên của bảng băm bao gồm 03 số nguyên không âm là m, a và b

Dòng tiếp theo của một bảng băm gồm nhiều số nguyên, là các số cần được thêm vào bảng băm.

OUTPUT

Ứng với mỗi bảng băm trong input, xuất tra trên một dòng các vị trí của bảng băm. Vị trí trống được ghi bằng chữ "None" các vị trí cách nhau bởi một dấu phẩy và khoảng trắng.

class HashTable:

def \_\_init\_\_(self, m, a, b):

self.m = m

self.a = a

self.b = b

self.table = [None] \* m

def hash\_function(self, key):

if key < 0:

r = abs(key) % self.m

return self.m - r if r != 0 else 0

else:

return key % self.m

def probing(self, key, i):

return (self.hash\_function(key) + self.a \* i + self.b \* i\*\*2) % self.m

def insert(self, key):

for i in range(self.m):

index = self.probing(key, i)

if self.table[index] is None:

self.table[index] = key

return

elif self.table[index] == key:

return

def display(self):

return ', '.join(str(item) if item is not None else 'None' for item in self.table)

def process\_input():

import sys

input = sys.stdin.read

data = input().strip().split('\n')

i = 0

results = []

while i < len(data):

line = data[i].strip()

if line == '0':

break

m, a, b = map(int, line.split())

i += 1

keys = list(map(int, data[i].strip().split()))

hash\_table = HashTable(m, a, b)

for key in keys:

hash\_table.insert(key)

results.append(hash\_table.display())

i += 1

for result in results:

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

process\_input()

Bài 18:

Viết Chương trình cài đặt bảng băm open addressing với kích thước cố định là m, khi bảng băm đầy thì không thêm phần tử vào nữa.

Bảng băm giải quyết đụng độ bằng kỹ thuật dò quadratic probing với hàm dò h(k, i) = ((k%m) + a\*i + b\*i^2 ) % m

INPUT

Dòng đầu tiên trong input là 03 số nguyên m, a, b. Các dòng tiếp theo mỗi dòng sẽ có dạng như sau:

Dòng bắt đầu bằng số 1 và theo sau là một số nguyên sẽ yêu cầu chương trình phải thêm số nguyên đó vào bảng băm.

Dòng bắt đầu bằng số 2 và theo sau là một số nguyên sẽ yêu cầu chương trình phải xóa số nguyên đó ra khỏi bảng băm.

Dòng bắng đầu bằng số 0 sẽ báo hiệu input kết thúc.

OUTPUT

Sau khi xử lý toàn bộ input, xuất tra trên một dòng các vị trí của bảng băm. Vị trí trống được ghi bằng chữ "None", vị trí đã bị xóa được ghi bằng chữ DELETED các vị trí cách nhau bởi một dấu phẩy và khoảng trắng.

class HashTable:

def \_\_init\_\_(self, m, a, b):

self.m = m

self.a = a

self.b = b

self.table = [None] \* m

self.deleted = "DELETED"

self.elements = set()

self.deleted\_indices = set()

def hash\_function(self, key):

if key < 0:

r = abs(key) % self.m

return self.m - r if r != 0 else 0

else:

return key % self.m

def probing(self, key, i):

return (self.hash\_function(key) + self.a \* i + self.b \* i\*\*2) % self.m

def insert(self, key):

if key in self.elements:

return

for i in range(self.m):

index = self.probing(key, i)

if self.table[index] is None:

self.table[index] = key

self.elements.add(key)

return

elif self.table[index] == self.deleted and index not in self.deleted\_indices:

self.table[index] = key

self.elements.add(key)

self.deleted\_indices.remove(index)

return

def delete(self, key):

if key not in self.elements:

return

for i in range(self.m):

index = self.probing(key, i)

if self.table[index] is None:

continue

if self.table[index] == key:

self.table[index] = self.deleted

self.elements.remove(key)

self.deleted\_indices.add(index)

return

for j in range(self.m):

if self.table[j] == key:

self.table[j] = self.deleted

self.elements.remove(key)

self.deleted\_indices.add(j)

return

def display(self):

return ', '.join(str(item) if item is not None else "None" for item in self.table)

def process\_input():

import sys

input = sys.stdin.read

data = input().strip().split('\n')

m, a, b = map(int, data[0].split())

hash\_table = HashTable(m, a, b)

i = 1

while i < len(data):

parts = data[i].strip().split()

command = int(parts[0])

if command == 0:

break

number = int(parts[1])

if command == 1:

hash\_table.insert(number)

elif command == 2:

hash\_table.delete(number)

i += 1

print(hash\_table.display())

if \_\_name\_\_ == "\_\_main\_\_":

process\_input()

Bài 18:

Viết Chương trình cài đặt bảng băm open addressing với kích thước cố định là m, khi bảng băm đầy thì không thêm phần tử vào nữa.

Bảng băm giải quyết đụng độ bằng kỹ thuật dò quadratic probing với hàm dò h(k, i) = ((k%m) + a\*i + b\*i^2 ) % m

Khi có một phần tử không thể được insert vì bảng băm đầy, kích thước bảng băm sẽ được tăng lên với các yêu cầu sau:

1. Bảng băm mới có kích thước bằng với số nguyên tố nhỏ nhất mà có giá trị gấp đôi kích thước bảng băm cũ.

2. Các số trong bảng băm cũ sẽ được thêm vào bảng băm mới theo thứ tự mà chúng được lưu trữ trong bảng băm cũ (không cần quan tâm thứ tự chúng được insert vào)

INPUT

Input gồm nhiều dòng mỗi cứ mỗi 02 dòng sẽ tương ứng với một bảng băm và tận cùng bởi một dòng chỉ chứa số 0

Dòng đầu tiên của bảng băm bao gồm 03 số nguyên không âm là m, a và b

Dòng tiếp theo của một bảng băm gồm nhiều số nguyên, là các số cần được thêm vào bảng băm.

OUTPUT

Ứng với mỗi bảng băm trong input, xuất tra trên một dòng các vị trí của bảng băm. Vị trí trống được ghi bằng chữ "None" các vị trí cách nhau bởi một dấu phẩy và khoảng trắng.

import sys

def is\_prime(n):

if n <= 1:

return False

if n <= 3:

return True

if n % 2 == 0 or n % 3 == 0:

return False

i = 5

while i \* i <= n:

if n % i == 0 or n % (i + 2) == 0:

return False

i += 6

return True

def next\_prime(n):

while True:

n += 1

if is\_prime(n):

return n

class HashTable:

def \_\_init\_\_(self, m, a, b):

self.m = m

self.a = a

self.b = b

self.table = [None] \* m

def hash\_function(self, key):

if key < 0:

r = abs(key) % self.m

return self.m - r if r != 0 else 0

else:

return key % self.m

def probing(self, key, i):

return (self.hash\_function(key) + self.a \* i + self.b \* i\*\*2) % self.m

def insert(self, key):

for i in range(self.m):

index = self.probing(key, i)

if self.table[index] is None:

self.table[index] = key

return

elif self.table[index] == key:

return

self.rehash()

self.insert(key)

def rehash(self):

new\_m = next\_prime(2 \* self.m)

new\_table = [None] \* new\_m

old\_keys = [key for key in self.table if key is not None]

self.m = new\_m

self.table = new\_table

for key in old\_keys:

self.insert\_direct(key)

def insert\_direct(self, key):

for i in range(self.m):

index = self.probing(key, i)

if self.table[index] is None:

self.table[index] = key

return

def display(self):

return ', '.join(str(item) if item is not None else 'None' for item in self.table)

def process\_input():

input = sys.stdin.read

data = input().strip().split('\n')

i = 0

results = []

while i < len(data):

line = data[i].strip()

if line == '0':

break

m, a, b = map(int, line.split())

i += 1

keys = list(map(int, data[i].strip().split()))

hash\_table = HashTable(m, a, b)

for key in keys:

hash\_table.insert(key)

results.append(hash\_table.display())

i += 1

for result in results:

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

process\_input()

Bài 19:

Thầy Sơn mới thiết kế một hệ thống N máy tính (các máy tính được đánh số từ 1 đến N) được nối lại thành một mạng bởi M kênh nối, mỗi kênh nối hai máy nào đó và cho phép ta truyền tin từ máy này sang máy kia. Nhưng không may rằng Thầy Sơn đã bị một hacker nào đó tấn công vào một máy bất kì trong N máy. Để khắc phục sự cố thì thầy Sơn cần liệt kê ra các máy đã bị tấn công. Mặc dù vậy nhưng mà hệ thống máy tính của thầy rất lớn nên không thể liệt kê bằng cách thông thường được. Em hãy viết chương trình giúp thầy Sơn xác định các máy bị tấn công? Biết rằng, nếu một máy bị cấn công thì các máy được kết nối trực tiếp cũng sẽ bị tấn công.

import sys

from collections import defaultdict, deque

def main():

input = sys.stdin.readline

N, M = map(int, input().strip().split())

graph = defaultdict(list)

for \_ in range(M):

u, v = map(int, input().strip().split())

graph[u].append(v)

graph[v].append(u)

s = int(input().strip())

queue = deque([s])

visited = set([s])

while queue:

node = queue.popleft()

for neighbor in graph[node]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

compromised\_computers = sorted(visited)

print(len(compromised\_computers))

print(' '.join(map(str, compromised\_computers)))

if \_\_name\_\_ == "\_\_main\_\_":

main()

Bài 19:

Cài đặt một số thao tác cơ bản trên một đồ thị hữu hướng, mỗi đỉnh của đồ thị được đặt tên.

INPUT

Dòng đầu tiên chứa 02 số e và n, đây lần lượt là số cạnh của đồ thị cùng với số thao tác xử lý, giá trị mỗi số không quá 1 tỷ.

e dòng tiếp theo, mỗi dòng chứa 02 chuỗi u và i (cách nhau bởi khoảng trắng), thể hiện việc có một cạnh nối từ đỉnh u sang đỉnh i trong đồ thị.

n dòng tiếp theo, mỗi dòng tương ứng với một thao tác xử lý các thao tác có cú pháp như sau:

Thao tác kiểm tra tính kề của 02 đỉnh, dòng này bắt đầu bằng số 1, theo sau là 02 chuỗi u và i cách nhau bởi khoảng trắng

Thao tác tìm kiếm đỉnh lân cận của 01 đỉnh, dòng này bắt đầu bằng số 2, theo sau là chuỗi u

OUTPUT

Ứng với thao tác kiểm tra tính kề của 02 đỉnh, xuất ra màn hình chuỗi TRUE nếu đỉnh u kề với đỉnh i. Nếu đỉnh u không kề với đỉnh i xuất ra chuỗi FALSE

Ứng với thao tác tìm kiếm đỉnh lân cận của 02 đỉnh, xuất ra màn hình số lượng đỉnh kề với đỉnh u

import sys

from collections import defaultdict

def main():

input = sys.stdin.readline

e, n = map(int, input().strip().split())

edges = defaultdict(list)

counts = defaultdict(int)

for \_ in range(e):

u, v = input().strip().split()

edges[u].append(v)

counts[u] += 1

for \_ in range(n):

query = input().strip().split()

query\_type = int(query[0])

if query\_type == 1:

u, v = query[1], query[2]

if v in edges[u]:

print("TRUE")

else:

print("FALSE")

elif query\_type == 2:

u = query[1]

print(counts[u])

if \_\_name\_\_ == "\_\_main\_\_":

main()

Bài 20:

Ngày xửa ngày xưa, xưa ơi là xưa, xưa xửa xừa xưa, ở vương quốc “Nọ” có một chàng hiệp sĩ... nghèo. Hiệp sĩ thầm thương trộm nhớ một nàng công chúa, con của Quốc Vương. Ngày kia triều đình mở hội kén rể cho công chúa, do công chúa có thân hình mỹ miều, nhan sắc bá đạo, FA lâu năm nên điều kiện nhà vua đưa ra vô cùng đơn giản, ai đến sớm nhất là được cưới công chúa không cần sính lễ, ngoài ra còn được vua bonus thêm của hồi môn, miễn là cưới công chúa đi càng sớm càng tốt.

Hiệp sĩ vô cùng mừng rỡ và hăm hở lên đường, tuy nhiên do chàng rất nghèo chỉ có thể đi bộ nên chàng phải tìm con đường ngắn nhất để có thể đến kịp và rước được công chúa.

Bản đồ của vương quốc là một hình chữ nhật gồm m x n ô vuông (m, n < 5000), hiệp sĩ có thể đi theo bất cứ đường nào từ một ô đến các ô lân cận (ngang dọc hoặc chéo) thời gian để đi từ ô này sang ô kia là một canh giờ.

Các ô trên bản đồ được đánh tọa độ theo số dòng và số cột, ô ở góc dưới cùng bên tay trái bản đồ có tọa độ là (0, 0) và ô ở góc trên cùng bên tay phải bản đồ có tọa độ (m-1, n-1). Hiệp sĩ không được đi ra khỏi vương quốc, nếu không sẽ bị nước láng giềng tóm cổ tống giam không lấy được công chúa. Trên bản đồ có một số ô là núi cao, vực sâu, rừng thiêng nước độc không thể đi vào được. Hãy viết chương trình cho biết nếu hiệp sĩ đi suốt ngày suốt đêm không ăn không ngủ, không làm cả các “nhu cầu khác” thì phải mất bao nhiêu canh giờ hiệp sĩ mới lên được kinh đô để cưới công chúa.

INPUT

Dòng đầu tiên của input chứa 6 con số cách nhau bởi khoảng trắng, hai con số đầu lần lượt là m, n, hai số tiếp theo là tọa độ nhà của hiệp sĩ và hai con số của cùng là tọa độ của hoàng cung nơi công chúa đang mỏi mắt trông một tấm chồng.

m dòng tiếp theo trong input, mỗi dòng chứa n con số cách nhau bởi khoảng trắng. Đây là toàn bộ bản đồ vương quốc với con số 0 tượng trưng cho những ô có thể đi vào được và con số 1 tượng trưng cho những ô không thể đi vào được.

OUTPUT

Một con số duy nhất cho biết thời gian hiệp sĩ cần để đến được chỗ công chúa, tính theo canh giờ. Nếu hiệp sĩ không thể đến được chỗ công chúa xuất ra -1

import heapq

def find\_shortest\_path(m, n, knight\_home, princess\_home, grid):

directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]

pq = [(0, knight\_home[0], knight\_home[1])]

distances = {(knight\_home[0], knight\_home[1]): 0}

visited = set()

while pq:

dist, x, y = heapq.heappop(pq)

if (x, y) in visited:

continue

visited.add((x, y))

if (x, y) == (princess\_home[0], princess\_home[1]):

return dist

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < m and 0 <= ny < n and grid[nx][ny] == 0:

new\_dist = dist + 1

if (nx, ny) not in distances or new\_dist < distances[(nx, ny)]:

distances[(nx, ny)] = new\_dist

heapq.heappush(pq, (new\_dist, nx, ny))

return -1

input\_data = input().split()

m = int(input\_data[0])

n = int(input\_data[1])

knight\_x = int(input\_data[2])

knight\_y = int(input\_data[3])

princess\_x = int(input\_data[4])

princess\_y = int(input\_data[5])

knight\_home = (m - 1 - knight\_x, knight\_y)

princess\_home = (m - 1 - princess\_x, princess\_y)

grid = []

for \_ in range(m):

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

result = find\_shortest\_path(m, n, knight\_home, princess\_home, grid)

print(result)

Bài 22:

Ngày xưa có một hiệp sĩ, sau khi dẫn đoàn quân tiên phong vào tiền đồn của địch, chỉ sau vài giây ngắn ngủi, chàng đã nhận ra đây là một sai lầm khủng khiếp. Với quân lực khổng lồ của địch ở trước mặt và sông suối đầm lầy ở sau lưng, tình thế của chàng giờ đây như đèn treo trước gió như “cờ-hó” chạy cùng sào. Không còn cách nào khác chàng quyết định làm một cú bôn tẩu đến vùng rừng núi bên cánh trái.

Đường rừng gập ghềnh dấu binh lửa chiếu hào quang đục ngầu, chàng thúc ngựa trong lời gào dẫn đoàn quân trăm chiến mấy ngàn binh tới ngoài biên thoát vòng ngục tù. Gọi nước non cũ tiến về Nam miệt vẫn từ xưa hăng hái súng lòng vang, dưới cờ bay lấp trời mây.

(Hòn vọng phu 3 - Lê Thương, Ánh Tuyết)

Chàng hiệp sĩ cứ thế phi ngựa tháo chạy, trong tay chàng chỉ có mỗi một tấm bản đồ vẽ vội của vùng núi rừng dưới dạng một hình chữ nhật gồm m x n ô vuông (m, n < 5000). Ngựa của chàng mỗi canh giờ có thể phi hơn 2 ô theo đường ngang hoặc dọc, nhưng do chạy thẳng nên thường dễ bị địch phát hiện nên chàng phải đi lệch 1 ô về phía phải hoặc trái tạo thành một đường đi hình chữ L (tham khảo quân mã trong cờ vua).

Cứ mỗi một canh giờ hiệp sĩ bắt buộc phải cho ngựa và binh lính nghỉ mệt chứ không ngựa bị đột quỵ thì chàng cũng toi. Tuy nhiên chàng chỉ có thể nghỉ ở những ô có địa hình rừng núi ẩn nấp được. Những ô địa hình trống trải rất dễ bị địch tập kích nên phải phi ngựa qua luôn không dừng lại được.

Hãy viết chương trình tính thời gian để hiệp sĩ có thể dẫn quân từ nơi tử địa đến địa điểm an toàn.

INPUT

Dòng đầu tiên của input chứa 6 con số cách nhau bởi khoảng trắng, hai con số đầu lần lượt là m, n, hai số tiếp theo là tọa độ vùng chiến sự và hai con số của cùng là tọa độ của điểm an toàn.

m dòng tiếp theo trong input, mỗi dòng chứa n con số cách nhau bởi khoảng trắng. Đây là toàn bộ bản đồ với con số 0 tượng trưng cho những ô địa hình có thể ẩn nấp nghỉ chân được và con số 1 tượng trưng cho những ô địa hình trống trải không thể dừng lại.

from collections import deque

def dijkstra(graph, start, end):

m, n = len(graph), len(graph[0])

directions = [(-2, -1), (-2, 1), (-1, 2), (1, 2), (2, 1), (2, -1), (1, -2), (-1, -2)]

queue = deque([(start, 0)])

visited = set()

visited.add(start)

while queue:

(x, y), dist = queue.popleft()

if (x, y) == end:

return dist

for dx, dy in directions:

nx, ny = x + dx, y + dy

if 0 <= nx < m and 0 <= ny < n and graph[nx][ny] == 0 and (nx, ny) not in visited:

visited.add((nx, ny))

queue.append(((nx, ny), dist + 1))

return -1

def read\_input\_and\_solve():

import sys

input = sys.stdin.read

data = input().split()

m = int(data[0])

n = int(data[1])

start\_x = int(data[2])

start\_y = int(data[3])

end\_x = int(data[4])

end\_y = int(data[5])

graph = []

index = 6

for \_ in range(m):

row = list(map(int, data[index:index + n]))

graph.append(row)

index += n

graph.reverse()

result = dijkstra(graph, (start\_x, start\_y), (end\_x, end\_y))

print(result)

if \_\_name\_\_ == "\_\_main\_\_":

read\_input\_and\_solve()