МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

**«Московский технический университет связи и информатики»**

Кафедра

«Математическая кибернетика и информационные технологии»

**Курсовая работа**

по дисциплине

**«**Структуры и алгоритмы обработки данных**»**

Выполнил:

студент гр. БСТ2004

Шадюк М.Р.

Москва, 2022 г.

Оглавление

[**Задание 1 «Плохие цены» 1100 2**](#_Toc103304445)

[**Код 2**](#_Toc103304446)

[**Результат 3**](#_Toc103304447)

[**Вводные 3**](#_Toc103304448)

[**Вывод 4**](#_Toc103304449)

[**Задание 2 «Система регистрации» 1300 4**](#_Toc103304450)

[**Код 5**](#_Toc103304451)

[**Результат 5**](#_Toc103304452)

[**Вводные 5**](#_Toc103304453)

[**Вывод 6**](#_Toc103304454)

[**Задание 3 "Влад и отложенные дела" 1800 6**](#_Toc103304455)

[**Код 7**](#_Toc103304456)

[**Результат 8**](#_Toc103304457)

[**Вводные 8**](#_Toc103304458)

[**Вывод 9**](#_Toc103304459)

[**Задание 4 «Стартап» 1000 9**](#_Toc103304460)

[**Код 10**](#_Toc103304461)

[**Результат 10**](#_Toc103304462)

[**Вводные 10**](#_Toc103304463)

[**Вывод 10**](#_Toc103304464)

[**Задание 5 «Карточные пирамиды» 1100 10**](#_Toc103304465)

[**Код 11**](#_Toc103304466)

[**Результат 12**](#_Toc103304467)

[**Ввод 12**](#_Toc103304468)

[**Вывод 13**](#_Toc103304469)

# Задание 1 «Плохие цены» 1100

Поликарп анализирует цену на новую модель berPhone. В его распоряжении цены за nn последних дней: a1,a2,…,ana1,a2,…,an, где aiai — цена berPhone в день ii.

Поликарп считает цену в день ii плохой, если позже (то есть в день с большим номером) berPhone продавался по меньшей цене. Например, если n=6n=6 и a=[3,9,4,6,7,5]a=[3,9,4,6,7,5], то количество дней с плохой ценой равно 33 — это дни 22 (a2=9a2=9), 44 (a4=6a4=6) и 55 (a5=7a5=7).

Выведите количество дней с плохой ценой по мнению Поликарпа.

Вам необходимо ответить на tt независимых наборов входных данных.

**Входные данные**

В первой строке записано целое число tt (1≤t≤100001≤t≤10000) — количество наборов входных данных в тесте. Наборы входных данных надо обрабатывать независимо, один за другим.

Каждый набор входных данных состоит из двух строк. Первая строка содержит целое число nn (1≤n≤1500001≤n≤150000) — количество дней. Вторая строка содержит nn целых чисел a1,a2,…,ana1,a2,…,an (1≤ai≤1061≤ai≤106), где aiai — цена в ii-й день.

Гарантируется, что сумма nn по всем наборам входных данных в тесте не превосходит 150000150000.

**Выходные данные**

Выведите tt целых чисел, jj-е из них должно быть равно количеству дней с плохой ценой в jj-м наборе входных данных.

## Код

def solve(n, arr):

    min\_price\_on\_right = int(arr[n-1])

    bad\_prices = 0

    for i in range(n-2, -1, -1):

        if int(arr[i]) > min\_price\_on\_right:

            bad\_prices += 1

        else:

            min\_price\_on\_right = int(arr[i])

    return bad\_prices

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

    t = int(input())

    results = list()

    for \_ in range(0, t):

        n = int(input())

        arr = input().split(" ")

        results.append(solve(n, arr))

    for result in results:

        print(result)

## Результат

### Вводные

5

6

3 9 4 6 7 5

1

1000000

2

2 1

10

31 41 59 26 53 58 97 93 23 84

7

3 2 1 2 3 4 5

### Вывод



Рисунок 1 – Выполнение первого задания

# Задание 2 «Система регистрации» 1300

В скором времени в Берляндии откроется новая почтовая служба "Берляндеск". Администрация сайта хочет запустить свой проект как можно быстрее, поэтому они попросили Вас о помощи. Вам предлагается реализовать прототип системы регистрации сайта.

Система должна работать по следующему принципу. Каждый раз, когда новый пользователь хочет зарегистрироваться, он посылает системе запрос name со своим именем. Если данное имя не содержится в базе данных системы, то оно заносится туда и пользователю возвращается ответ OK, подтверждающий успешную регистрацию. Если же на сайте уже присутствует пользователь с именем name, то система формирует новое имя и выдает его пользователю в качестве подсказки, при этом подсказка также добавляется в базу данных. Новое имя формируется по следующему правилу. К name последовательно приписываются числа, начиная с единицы (name1, name2, ...), и среди них находят такое наименьшее i, что namei не содержится в базе данных сайта.

**Входные данные**

В первой строке входных данных задано число *n* (1 ≤ *n* ≤ 105). Следующие *n* строк содержат запросы к системе. Каждый запрос представляет собой непустую строку длиной не более 32 символов, состоящую только из строчных букв латинского алфавита.

**Выходные данные**

В выходных данных должно содержаться *n* строк — ответы системы на запросы: ОК в случае успешной регистрации, или подсказку с новым именем, если запрашиваемое уже занято.

## Код

def solve(n, names):

    db = {}

    result = []

    for i in range(0, n):

        if names[i] not in db:

            result.append("OK")

            db[names[i]] = 1

        else:

            postfix = db[names[i]]

            result.append(names[i] + str(postfix))

            db[names[i]] = postfix + 1

    return result

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

    n = int(input())

    names = []

    for i in range(0, n):

        names.append(input())

    result = solve(n, names)

    for i in range(0, n):

        print(result[i])

## Результат

### Вводные

4

abacaba

acaba

abacaba

acab

### Вывод

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 2 – Результат работы второй программы

# Задание 3 "Влад и отложенные дела" 1800

Влад и Настя живут в городе, состоящем из nn домов и n−1n−1 дороги. Из каждого дома можно добраться до другого передвигаясь только по дорогам. То есть город представляет собой дерево.

Влад живёт в доме с номером xx, а Настя в доме с номером yy. Влад решил пойти к Насте в гости. Однако он вспомнил, что отложил на потом kk дел, которые он должен сделать прежде чем прийти к Насте. Чтобы сделать ii-е дело ему необходимо прийти в aiai-й дом, дела можно делать в любом порядке. За 11 минуту он может пройти от одного дома до другого, если их соединяет дорога.

Влад не очень любит пешие прогулки, поэтому его интересует, какое минимальное количество минут он должен потратить на дорогу, чтобы сделать все дела и после этого прийти к Насте. Дома a1,a2,…,aka1,a2,…,ak он может посетить в любом порядке. Любые Дома он может посещать многократно (если захочет).

**Входные данные**

В первой строке входных данных записано целое число tt (1≤t≤1041≤t≤104) — количество наборов входных данных в тесте. Перед каждым набором входных данных в тесте содержится пустая строка.

Первая строка набора входных данных содержит два числа nn и kk (1≤k≤n≤2⋅1051≤k≤n≤2⋅105) — количество домов и дел соответственно.

Вторая строка набора входных данных содержит два числа xx и yy (1≤x,y≤n1≤x,y≤n) — номера домов, в которых живут Влад и Настя, соответственно.

Третья строка набора входных данных содержит kk целых чисел a1,a2,…,aka1,a2,…,ak (1≤ai≤n1≤ai≤n) — номера домов, в которые нужно зайти по дороге.

Следующие n−1n−1 строк содержат описание города, по два числа на строку vjvj и ujuj (1≤uj,vj≤n1≤uj,vj≤n) — номера домов, которые соединяет дорога jj.

Гарантируется, что сумма значений nn по всем наборам входных данных в тесте не превосходит 2⋅1052⋅105.

**Выходные данные**

Выведите tt строк, каждая из которых содержит ответ на соответствующий набор входных данных. В качестве ответа выведите единственное число — минимальное количество минут, необходимых Владу на дорогу, чтобы сделать все дела и прийти к Насте.

## Код

def solve(n, k, x, y, a, graph):

    a.add(x)

    a.add(y)

    stack = [(x, 0)]

    parent = (n + 1) \* [-1]

    y\_height = None

    while stack:

        node, height = stack.pop()

        if node == y:

            y\_height = height

        for child in graph[node]:

            if child != parent[node]:

                parent[child] = node

                stack.append((child, height + 1))

    lst = list(a)

    edge\_ct = 0

    for node in lst:

        if node != x and parent[node] not in a:

            a.add(parent[node])

            lst.append(parent[node])

        if node != x:

            edge\_ct += 2

    return edge\_ct - y\_height

t = int(input())

for case in range(t):

    input()

    n, k = map(int, input().split())

    x, y = map(int, input().split())

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

    graph = [[] for \_ in range(n + 1)]

    for \_ in range(n - 1):

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

        graph[u].append(v)

        graph[v].append(u)

    print(solve(n, k, x, y, a, graph))

## Результат

### Вводные

3

3 1

1 3

2

1 3

1 2

6 4

3 5

1 6 2 1

1 3

3 4

3 5

5 6

5 2

6 2

3 2

5 3

1 3

3 4

3 5

5 6

5 2

### Вывод

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – Выполнение 3 задания

# Задание 4 «Стартап» 1000

Совсем недавно стартап двух студентов государственного университета города F стал невероятно популярным. Пришло время основать новую компанию. Но какое название для нее выбрать?

Было решено применить очень хитрый маркетинговый ход: название компании должно совпадать со своим зеркальным отражением! Другими словами, если выписать название компании на листке в строчку (горизонтально слева направо) большими буквами английского алфавита, после чего поставить этот листок напротив зеркала, то отражение названия в зеркале должно в точности совпадать со строкой, написанной на листке.

Предложений, как назвать компанию, было очень много, поэтому подходить с листком к зеркалу для каждого названия было не разумно. Основатели компании решили автоматизировать этот процесс. Они попросили вас написать программу, которая по заданному слову определит, является оно «зеркальным» или нет.

**Входные данные**

В первой строке содержится непустое название, которое требуется проверить. Название содержит в себе не более 105 заглавных букв английского алфавита. Для написания названия будет использоваться следующий шрифт без засечек:



**Выходные данные**

Выведите «YES» (без кавычек), если заданное название будет совпадать со своим зеркальным отражением. Иначе, выведите «NO» (без кавычек).

## Код

def solve(s):

    symmetric\_letters = {'A', 'H', 'I', 'M', 'O', 'T', 'U', 'V', 'W', 'X', 'Y'}

    n = len(s)

    for i in range(0, (n//2)+1):

        left\_char = s[i]

        right\_char = s[n - i - 1]

        if left\_char not in symmetric\_letters \

                or right\_char not in symmetric\_letters \

                or left\_char != right\_char:

            return "NO"

    return "YES"

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

    s = input()

    print(solve(s))

## Результат

### Вводные

AHA

### Вывод

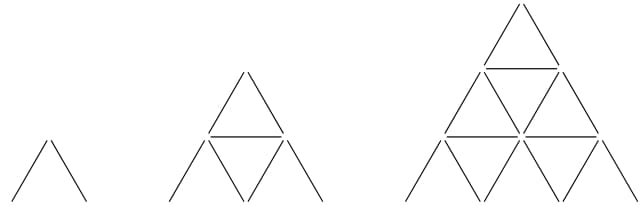
Изображение выглядит как текст

Автоматически созданное описание

Рисунок 4 – Результат

# Задание 5 «Карточные пирамиды» 1100

Карточная пирамида высоты 11 может быть построена с помощью двух карт, приставленных друг к другу. Для h>1h>1 карточная пирамида высоты hh может быть построена расположением карточной пирамиды высоты h−1h−1 на базе. База состоит из hh пирамидок высоты 11, поставленных в ряд, и h−1h−1 карты, лежащей сверху на них. Например, карточные пирамиды с высотами 11, 22 и 33 выглядят следующим образом:



Изначально у вас есть nn карт и вы строите как можно более высокую пирамиду из этих карт. Если остались лишние карты, вы снова строите как можно более высокую пирамиду из оставшихся карт. Вы продолжаете этот процесс до тих пор, пока не станет невозможно построить какую-нибудь пирамиду. Сколько карточных пирамид вы построите?

**Входные данные**

Каждый тест состоит из нескольких тестовых случаев. Первая строка содержит единственное целое число tt (1≤t≤10001≤t≤1000) — количество тестовых случаев. Следующие tt строк содержат описания тестовых случаев.

Единственная строка описания каждого тестового случая содержит единственное целое число nn (1≤n≤1091≤n≤109) — количество карт.

Гарантируется, что сумма значений nn по всем тестовым случаям не превосходит 109109.

**Выходные данные**

Для каждого тестового случая выведите единственное целое число — количество карточных пирамид, которое вы построите в конце процесса.

## Код

import math

def solve(card\_counts):

    pyramids\_possible = list()

    for card\_count in card\_counts:

        total\_pyramids = 0

        while card\_count >= 2:

            highest\_pyramid\_height = (math.sqrt((6 \* card\_count) + 0.25) - 0.5)/3

            highest\_pyramid\_cards\_needed = cards\_needed\_for\_given\_height(int(highest\_pyramid\_height))

            if card\_count >= highest\_pyramid\_cards\_needed:

                card\_count -= highest\_pyramid\_cards\_needed

            else:

                break

            total\_pyramids += 1

        pyramids\_possible.append(total\_pyramids)

    return "\n".join(str(p) for p in pyramids\_possible)

def cards\_needed\_for\_given\_height(h):

    return (h \* ((3\*h) + 1))/2

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

    t = int(input())

    card\_counts = list()

    for \_t in range(0,t):

        card\_counts.append(int(input()))

    print(solve(card\_counts))

## Результат

### Ввод

5

3

14

15

24

1

### Вывод

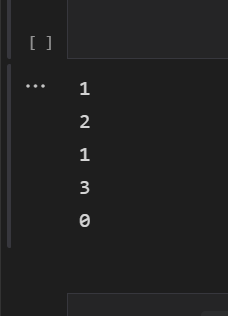


Рисунок 5 – Выполнение программы