Отборочный этап Олимпиады школьников «Шаг в будущее»

Профиль: «Инженерное дело»

Специализация: «Информатика»

Класс участия: 10-11

Задача 1 (8 баллов).

В городе Запакетск есть огромный сортировочный центр посылок. На каждой посылке наклеен штрих-код — целое число. Однажды в системе произошёл сбой: оказалось, что часть штрих-кодов печатается неправильно. После долгого расследования инженеры заметили странную закономерность:

- если какой-то код на конвейере встретился нечётное количество раз, то все посылки с таким кодом считаются бракованными и должны быть удалены из ленты;
- если код встретился чётное количество раз, то все посылки с таким кодом считаются корректными и остаются.

Инженеры хотят взять запись ленты за один день и «очистить» её: выкинуть все посылки с кодами, встречающимися нечётное число раз, а остальные оставить в том же порядке, в котором они ехали по конвейеру.

Ваша задача — по записи штрих-кодов восстановить, как будет выглядеть лента после очистки.

Если после удаления всех «нечётных» кодов на ленте не останется ни одной посылки, инженеры хотят считать, что лента стала пустой.

Входные данные

На первой строке задано целое число N — количество посылок на ленте

$1 <= N <= 200000$.

На второй строке задано целых чисел $a_1, a_2, \ldots, a_n$ – штрих-коды посылок

$|a_i| <= 10^9$.

Выходные данные

Если после очистки на ленте не останется ни одной посылки, выведите единственную строку «**Empty**», иначе:

- на первой строке выведите целое число K — количество посылок, оставшихся на ленте;
- на второй строке выведите K целых чисел — штрих-коды этих посылок в том порядке, в котором они останутся после удаления бракованных.

Примеры

Входные данные	**Вывод**
6 **5 10 5 3 10 3**	**6** **5 10 5 3 10 3**
7 **4 7 4 8 7 9 9**	**6** **4 7 4 7 9 9**
3 **7 7 7**	**Empty**

Решение:

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

cnt = {}
for x in a:
    cnt[x] = cnt.get(x, 0) + 1

ans = []
for x in a:
    if cnt[x] % 2 == 0:
        ans.append(x)

if not ans:
    print("Empty")
else:
    print(len(ans))
    print(*ans)
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	8

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.

Задача 2 (12 баллов).

В подземной библиотеке Академии стоит длинный ряд из N зачарованных свитков. Каждый свиток i даёт ученику a_i единиц знаний (это может быть и отрицательное число — тогда свиток запутывает и мешает). Ученику разрешено читать свитки только слева направо, в том порядке, в котором они стоят, и он может некоторые свитки пропустить. Но есть одно правило безопасности: **если ученик прочитает три свитка подряд без пропусков, его сознание перегреется, и эксперимент будет считаться проваленным**.

Формально, если он читает свитки с номерами $i_1 < i_2 < \ldots < i_k$, то нельзя, чтобы среди них встретилась тройка вида i, i+1, i+2, где все три свитка прочитаны.

Ученику нужно выбрать, какие свитки прочитать, чтобы:

1. он нигде не читал три подряд;
2. суммарное количество полученных знаний было максимально возможным.

Если все свитки вредные (сумма всегда получается неположительной), ученик может вообще ничего не читать и получить 0.

Входные данные

В первой строке задано целое число N — количество свитков

($1 \le N \le 200\,000$).

Во второй строке задано N целых чисел $a_1, a_2, \ldots, aN$ — количество знаний от каждого свитка

($-10^9 \le a_i \le 10^9$).

Выходные данные

Выведите одно целое число — максимальную суммарную полезность, которую может получить ученик, если будет соблюдать правило «не читать три свитка подряд».

Разрешается ничего не читать — в этом случае ответ равен 0.

Примеры

Входные данные	**Вывод**
5 3 2 5 10 7	22
4 -5 -1 -3 -4	0
6 4 -2 3 -1 2 3	12

Решение

```
n = int(input())

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



dp0 = 0
```

```
dp1 = float('-inf')

dp2 = float('-inf')



for x in a:

    new0 = max(dp0, dp1, dp2)

    new1 = dp0 + x

    new2 = dp1 + x



    dp0, dp1, dp2 = new0, new1, new2



print(max(dp0, dp1, dp2))
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	12

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.

Задача 3 (16 баллов).

В Академии магии существует временной канал, через который маги передают сообщения. Канал работает нестабильно: у него есть интервалы, когда он открыт, и интервалы, когда закрыт. Каждый интервал задаётся временем начала и конца в формате HH:MM:SS. Гарантируется, что для данного интервала канал открыт включительно от начала до конца.

Но существуют два типа интервалов:

- O — интервалы открытия (канал открыт)
- C — интервалы помех (канал закрыт и перекрывает любые открытия)

Приоритет: если на одну и ту же секунду приходится хотя бы один интервал C, то канал закрыт, даже если имеются интервалы O.

Необходимо узнать, сколько секунд за сутки канал был закрыт.

Сутки состоят из 86400 секунд — от 00:00:00 до 23:59:59.

Секунда считается закрытой, если выполняется хотя бы одно из двух условий

1. она покрыта хотя бы одним интервалом C;
2. она не покрыта ни одним интервалом O.

Формат входных данных

В первой строке заданы два целых числа N и M — количество интервалов O и количество интервалов C соответственно

($0 \le N, M \le 200000$).

Далее идут N строк вида:

O HH:MM:SS HH:MM:SS

После них идут M строк вида:

C HH:MM:SS HH:MM:SS

Гарантируется, что для каждого интервала время начала не больше времени конца.

Формат выходных данных

Выведите одно число — количество секунд, в течение которых канал был закрыт.

Примеры

Входные данные	**Вывод**
2 1 O 00:00:00 00:00:10 O 00:00:05 00:00:20 C 00:00:08 00:00:12	86384
0 1 C 23:00:00 23:59:59	86400
1 0 O 12:00:00 12:00:00	86399

Решение

```
def to_sec(s):
    h, m, sec = map(int, s.split(':'))
    return h * 3600 + m * 60 + sec
n, m = map(int, input().split())
DAY = 86400
open_diff = [0] * (DAY + 1)
close_diff = [0] * (DAY + 1)
for _ in range(n):
    typ, l, r = input().split()
    l = to_sec(l)
```

```
    r = to_sec(r)
    open_diff[l] += 1
    if r + 1 < DAY:
        open_diff[r + 1] -= 1
for _ in range(m):
    typ, l, r = input().split()
    l = to_sec(l)
    r = to_sec(r)
    close_diff[l] += 1
    if r + 1 < DAY:
        close_diff[r + 1] -= 1
closed_seconds = 0
cur_open = 0
cur_close = 0
for t in range(DAY):
    cur_open += open_diff[t]
    cur_close += close_diff[t]
    is_open = cur_open > 0
    is_closed_by_c = cur_close > 0
    if is_closed_by_c or not is_open:
        closed_seconds += 1
print(closed_seconds)
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	16

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.

Задача 4 (20 баллов)

В лаборатории магических материалов исследуют квадратную кристаллическую решётку размера N × N. Каждая ячейка содержит целое число — уровень активности кристалла. Учёные обнаружили, что если выбрать некоторый центр (i, j) и рассмотреть вокруг него квадратный участок нечётного размера, то такой участок считается стабильным, если выполняется условие: число в центральной клетке строго больше суммы всех остальных чисел этого квадрата.

Например:

- для квадрата 3×3 с центром (i, j) должно выполняться A[i][j] > сумма остальных 8 клеток;
- для квадрата 5×5 — A[i][j] > сумма остальных 24 клеток и так далее. Разрешаются размеры только 1×1, 3×3, 5×5, 7×7 и т. д.

Квадрат 1×1 считается стабильным только если A[i][j] > 0, так как сумма «внешних» клеток равна нулю.

Требуется найти общее количество стабильных квадратов по всей решётке, среди всех возможных центров и всех допустимых размеров.

Формат входных данных
Первая строка: целое число N ($1 \le N \le 700$).
Далее N строк по N целых чисел — матрица A.
Значения A[i][j] могут быть от -10^9 до 10^9.

Формат выходных данных

Выведите одно число — количество стабильных квадратов.

Входные данные	**Вывод**
3 5 1 1 1 2 1 1 1 1	9
3 0 0 0 0 10 0 0 0 0	2
5 0 0 0 0 0 0 10 0 0 0 0 0 0 4 0 0 1 0 0 0 0 0 0 3 0	7

Решение

```
def is_stable(grid, center_row, center_col, square_size,
prefix_sums):

    grid_size = len(grid)

    half_size = square_size // 2


    if (

        center_row - half_size < 0

        or center_row + half_size >= grid_size

        or center_col - half_size < 0

        or center_col + half_size >= grid_size

    ):

        return False


    square_sum = (
```

```
            prefix_sums[center_row + half_size + 1][center_col +
half_size + 1]

            - prefix_sums[center_row - half_size][center_col +
half_size + 1]

            - prefix_sums[center_row + half_size + 1][center_col
- half_size]

            + prefix_sums[center_row - half_size][center_col -
half_size]

    )



    square_sum -= grid[center_row][center_col]



    return grid[center_row][center_col] > square_sum




def count_stable_squares():

    matrix_size = int(input().strip())



    grid = []

    for _ in range(matrix_size):

        values_row = list(map(int, input().strip().split()))

        grid.append(values_row)



    prefix_sums = [[0] * (matrix_size + 1) for _ in
range(matrix_size + 1)]

    for row_idx in range(matrix_size):

        for col_idx in range(matrix_size):

            prefix_sums[row_idx + 1][col_idx + 1] = (

                    grid[row_idx][col_idx]

                    + prefix_sums[row_idx][col_idx + 1]
```

```
                    + prefix_sums[row_idx + 1][col_idx]
                    - prefix_sums[row_idx][col_idx]
            )

    stable_count = 0

    for row_idx in range(matrix_size):
        for col_idx in range(matrix_size):
            if grid[row_idx][col_idx] > 0:
                stable_count += 1

            square_size = 3
            while True:
                if not is_stable(grid, row_idx, col_idx,
square_size, prefix_sums):
                    break
                stable_count += 1
                square_size += 2
                if (
                    row_idx - square_size // 2 < 0
                    or row_idx + square_size // 2 >= matrix_size
                    or col_idx - square_size // 2 < 0
                    or col_idx + square_size // 2 >= matrix_size
                ):
                    break
    print(stable_count)
count_stable_squares()
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	20

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.

Задача 5 (20 баллов)

На исследовательском поле Академии разложены магические кристаллы. Каждый кристалл расположен в точке с целочисленными координатами (x_i, y_i) на плоскости. Учёные хотят узнать, какие два кристалла можно связать самой длинной магической нитью. Нить можно протянуть только по прямой линии от одного кристалла до другого. Требуется найти максимальное значение квадрата расстояния между какими-либо двумя кристаллами.

То есть нужно найти максимальное значение величины

$(x_i - x_j)^2 + (y_i - y_j)^2$ по всем парам различных точек (i, j).

Если кристалл всего один, то нить протянуть не между кем, и в этом случае ответ считается равным нулю.

Формат входных данных

В первой строке задано целое число N — количество кристаллов

($1 \le N \le 200000$).

В следующих N строках даны пары целых чисел x_i и y_i — координаты кристаллов.

Гарантируется, что $|x_i|, |y_i| \le 10^9$.

Формат выходных данных

Выведите одно целое число — максимальное значение квадрата расстояния между двумя кристаллами.

Если кристалл только один, выведите 0.

Входные данные	Выходные данные
3 0 0 3 4 -1 0	32
1 100 100	0
4 0 0 0 1 1 0 1 1	2

Решение

```
def cross(o, a, b):

    return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0]
- o[0])

def dist2(a, b):

    dx = a[0] - b[0]

    dy = a[1] - b[1]

    return dx * dx + dy * dy

def convex_hull(points):

    points = sorted(set(points))

    if len(points) <= 1:

        return points

    lower = []

    for p in points:

        while len(lower) >= 2 and cross(lower[-2], lower[-1], p)
<= 0:

            lower.pop()

        lower.append(p)
```

```
    upper = []
    for p in reversed(points):
        while len(upper) >= 2 and cross(upper[-2], upper[-1], p)
<= 0:
            upper.pop()
        upper.append(p)

    return lower[:-1] + upper[:-1]
def diameter_squared(points):
    n = len(points)
    if n <= 1:
        return 0
    if n == 2:
        return dist2(points[0], points[1])

    hull = convex_hull(points)
    m = len(hull)

    if m == 1:
        return 0
    if m == 2:
        return dist2(hull[0], hull[1])

    ans = 0
    j = 1

    for i in range(m):
```

```
        ni = (i + 1) % m


        while True:
            nj = (j + 1) % m
            cur = abs(cross(hull[i], hull[ni], hull[nj]))
            prev = abs(cross(hull[i], hull[ni], hull[j]))
            if cur > prev:
                j = nj
            else:
                break


        ans = max(ans, dist2(hull[i], hull[j]))
        ans = max(ans, dist2(hull[ni], hull[j]))


    return ans


n = int(input())
points = [tuple(map(int, input().split())) for _ in range(n)]
print(diameter_squared(points))
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	20

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.

Задача 6 (24 балла)

В подземной сети города Метролис есть N станций, соединённых магическими линиями разных цветов. Каждая линия — ориентированный транспортный коридор из станции u в станцию v, и у каждой линии есть цвет — маленькая латинская буква от a до z.

Ученику-хрононому нужно добраться со станции 1 до станции N. Ему разрешено ехать по линиям вдоль их направлений, переходить между

станциями и линиями сколько угодно раз, пока он не достигнет станции .

Однако есть проблема: каждый раз, когда он переходит с линии одного цвета на линию другого цвета, это вызывает временной удар по организму.

Формально, если он едет по пути

e1, e2, …, ek

и цвета этих линий равны c1, c2, …, ck, то количество ударов равно числу таких i, что

ci ¹ c_i+1

Цвет первой линии удара не вызывает (сравнивать не с чем).

Нужно выбрать такой путь из 1 в N, чтобы количество смен цвета по пути было минимально возможным.

Если добраться из 1 в N невозможно, нужно это честно признать.

Формат входных данных

Первая строка содержит два целых числа N и M— количество станций и количество линий (1 <= N <= 200000, 0 <= M <= 200000).

Следующие M строк описывают линии. Каждая строка состоит из: u, v, c, где

- u и v— целые числа, номера станций (1 <= u,v <= N),
- c — маленькая латинская буква (a–z), обозначающая цвет линии,
- линия идёт из u в v.

Между одними и теми же станциями могут быть несколько линий (в том числе разных цветов). Петли вида u->u тоже могут встречаться.

Формат выходных данных

Выведите одно целое число — **минимальное возможное число смен цвета** на пути от станции 1 до станции N.

Если добраться из 1 в N невозможно, выведите -1.

Входные данные	**Выходные данные**
4 4 1 2 r 2 3 r 3 4 r 1 3 b	0
3 2 1 2 a 2 3 b	1
3 1 2 3 a	-1

```
import sys
from collections import deque


data = sys.stdin.buffer.read().split()
if not data:
    sys.exit(0)

it = iter(data)
n = int(next(it))
m = int(next(it))

if n == 1:
    print(0)
    sys.exit(0)

state_id = {}
state_vertex = []
state_edges = []
station_states = [[] for _ in range(n)]

def get_state(v, color):
    key = v * 26 + color
    sid = state_id.get(key)
    if sid is None:
        sid = len(state_vertex)
```

```
        state_id[key] = sid
        state_vertex.append(v)
        state_edges.append([])
        station_states[v].append(n + sid)
    return sid

for _ in range(m):
    u = int(next(it)) - 1
    v = int(next(it)) - 1
    c = next(it)[0] - ord('a')

    su = get_state(u, c)
    sv = get_state(v, c)
    state_edges[su].append(sv)

total = n + len(state_vertex)
INF = 10**18
dist = [INF] * total
dist[0] = 0

dq = deque([0])

while dq:
    x = dq.popleft()
    dx = dist[x]

    if x < n:
        for y in station_states[x]:
            if dist[y] > dx + 1:
                dist[y] = dx + 1
                dq.append(y)
    else:
        sid = x - n
        v = state_vertex[sid]
        if dist[v] > dx:
            dist[v] = dx
            dq.appendleft(v)
        for to_sid in state_edges[sid]:
            y = n + to_sid
            if dist[y] > dx:
                dist[y] = dx
                dq.appendleft(y)

ans = dist[n - 1]
if ans == INF:
    print(-1)
else:
    print(ans - 1)
```

Критерии оценивания**

Критерий	Балл
Дан неверный ответ/ответ отсутствует	0
Дан верный ответ	24

** - Балл за каждую задачу проставляется в соответствии с долей выполнения участником задания. Шаг оценивания каждого задания - 1 балл. Доля выполнения вычисляется автоматической системой тестирования на основе комплекта установленных для задания тестов.