Максимальное количество баллов за олимпиаду — 600

Задание 1. Расстояние между словами

Будем называть словом любую последовательность букв русского алфавита. За одну операцию разрешается сделать одно из трёх действий:

- убрать из слова одну любую букву,
- добавить в любое место слова любую букву,
- заменить любую букву в слове на любую другую букву.

Расстоянием между словами будем называть наименьшее число операций, которое позволяет получить из одного слова другое. Сколько слов находятся на расстоянии 2 и от слова СТУК, и от слова КУСТЫ?

Ответ: 10

Критерий оценивания: точное совпадение ответа — 100 баллов

Максимальный балл за задание — 100

Решение. Если некоторое слово находится на расстоянии 2 от слов КУСТЫ и СТУК, то расстояние между словами КУСТЫ и СТУК не более 4. Изучим кратчайшие «пути» (по количеству операций) между ними.

Покажем, что из слова КУСТЫ можно получить слово СТУК минимум за 4 операции, причём эти 4 операции: убрать К, У, заменить Ы на У или К и добавить К или У. Если это утверждение доказано, то слова на расстоянии 2 от обоих могут быть только после применения двух из указанных операций, а таких слов 10:

СТЫ, УСТУ, УСТК, КСТУ, КСТК, УСТУЫ, УСТЫК, КСТУЫ, КСТЫК, КУСТУК.

Поскольку операций не более 4, то к какой-то букве не применялись операции, назовем её неподвижной. Пусть она ровно одна. Тогда из остальных букв удалили одну, а остальные поменяли. В результате неподвижная буква либо сохранила свой порядковый номер в слове, либо уменьшила его на 1. Но с такими условиями общих букв у слов КУСТЫ и СТУК нет. Следовательно, неподвижных букв как минимум две. Они должны идти в исходном и конечном слове в одинаковом порядке, поэтому это обязательно буквы С и Т. Таким образом, как минимум две операции потребуется, чтобы удалить все буквы до С и Т, ещё как минимум две — чтобы после СТ появились У и К. Итого нужно не менее 4 операций, причём равенство достигается только в случаях, которые мы описали в начале решения.

Задание 2. След степени матрицы

Матрицей $n \times m$ будем называть таблицу из чисел, состоящую из n строк и m столбцов.
Умножение матриц — это применение правила «строка на столбец». Если

$$M = \begin{pmatrix} p & q \\ r & s \end{pmatrix}, \qquad N = \begin{pmatrix} u & v \\ w & x \end{pmatrix},$$

то

$$\begin{pmatrix} pu + qw & pv + qx \\ ru + sw & rv + sx \end{pmatrix}.$$

Обычно порядок важен: как правило, $MN \neq NM$. Возведением матрицы в степень называют многократное умножение её самой на себя: $A^1 = A$, а далее $A^{k+1} = A^k A$.
Дана матрица:

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix},$$

Найдите сумму левого верхнего и правого нижнего элементов матрицы A^{20}.

Ответ: 15127

Критерий оценивания: точное совпадение ответа — 100 баллов

Максимальный балл за задание — 100

Решение. Рассмотрим умножение произвольной матрицы $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$ на матрицу $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.

Имеем

$$MA = \begin{pmatrix} a + b & a \\ c + d & c \end{pmatrix}.$$

Видно, что при умножении на A правый столбец результата совпадает с левым столбцом исходной матрицы, а новый левый столбец получается как сумма двух столбцов исходной матрицы.

Обозначим числа Фибоначчи: $F_0 = 0$, $F_1 = 1$, $F_{n+1} = F_n + F_{n-1}$. Из описанного свойства умножения следует, что при последовательном умножении на матрицу A столбцы матриц A^n эволюционируют по рекуррентному правилу Фибоначчи.

Докажем по индукции, что для всех $n \geq 1$

$$A^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

База индукции. При $n = 1$

$$A^1 = A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_2 & F_1 \\ F_1 & F_0 \end{pmatrix}.$$

Индукционный переход. Пусть формула верна для некоторого n. Тогда

$$A^{n+1} = A^n A = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} F_{n+1} + F_n & F_{n+1} \\ F_n + F_{n-1} & F_n \end{pmatrix}.$$

По определению чисел Фибоначчи это равно

$$\begin{pmatrix} F_{n+2} & F_{n+1} \\ F_{n+1} & F_n \end{pmatrix},$$

что завершает индукцию.

Следовательно,

$$A^{20} = \begin{pmatrix} F_{21} & F_{20} \\ F_{20} & F_{19} \end{pmatrix}$$

и искомая сумма равна $F_{21} + F_{19}$.

Вычислим значения: $F_{19} = 4181$, $F_{21} = 10946$. Тогда $F_{21} + F_{19} = 10946 + 4181 = 15127$.

Задание 3. ReLU

Напомним, что

$$\mathrm{ReLU}(z) = \max(0, z).$$

Рассмотрим функцию действительного аргумента x:

$$F(x) = c + w_1 \cdot \mathrm{ReLU}(x - t_1) + w_2 \cdot \mathrm{ReLU}(x - t_2) + \cdots + w_{2025} \cdot \mathrm{ReLU}(x - t_{2025}),$$

где $t_1, \ldots, t_{2025}$ — попарно различные действительные числа, каждый коэффициент w_i равен либо $+1$, либо -1, а c — действительное число.

Какое наибольшее число различных решений может иметь уравнение $F(x) = 0$, если известно, что у него конечное число решений?

Ответ: 1013

Критерий оценивания: точное совпадение ответа — 100 баллов

Максимальный балл за задание — 100

Решение. Упорядочим точки: $t_1 < t_2 < \cdots < t_{2025}$. На каждом промежутке (t_k, t_{k+1}) функция F является линейной. Её наклон равен

$$a_k = \sum_{i \leq k} w_i, \quad a_0 = 0.$$

При переходе через каждую точку t_i наклон изменяется на ± 1.

Рассмотрим уравнение $F(x) = 0$, то есть точки пересечения графика F с осью x. На каждом промежутке, где наклон a_k имеет фиксированный ненулевой знак, график может пересечь ось x не более одного раза.

Если при переходе через некоторую точку t_i наклон меняется с положительного на отрицательный, то в этой точке наклон обязательно проходит через значение 0, то есть функция достигает горизонтального положения. Если на каком-то промежутке наклон равен нулю, то соответствующий участок графика является горизонтальным; на таком отрезке либо нет решений уравнения $F(x) = 0$, либо решений бесконечно много. Последний случай запрещён по условию, поэтому горизонтальные отрезки не дают вклад в число решений.

Следовательно, число решений уравнения $F(x) = 0$ не превосходит числа групп подряд идущих промежутков, на которых наклоны a_k имеют одинаковый ненулевой знак.

Предположим, что таких групп не меньше 1014. Тогда между ними должно быть как минимум 1013 переходов, в которых наклон обращается в ноль. Это требует наличия как минимум 2026 различных точек t_i, что противоречит условию $t_1, \ldots, t_{2025}$. Следовательно, число решений не превосходит 1013.

Покажем, что это значение достижимо. Положим $c = -1$ и зададим веса периодическим блоком

$$(w_1, w_2, w_3, w_4) = (+1, -1, -1, +1),$$

повторяя его до w_{2025}. Тогда частичные суммы наклонов имеют вид

$$(a_k) = (0, 1, 0, -1, 0, 1, 0, -1, \dots),$$

то есть $a_k \neq 0$ ровно при нечётных k.

Положим $t_1 = 0$, а далее зададим

$$t_{k+1} - t_k = \begin{cases} 3, & k \text{ нечётно}, \\ 1, & k \text{ чётно}. \end{cases}$$

При таком выборе на каждом нечётном промежутке (t_k, t_{k+1}) функция F меняет знак и имеет ровно одно решение уравнения $F(x) = 0$, а на чётных промежутках решений нет.

Таким образом, число решений равно числу нечётных промежутков, то есть 1013. Это и есть наибольшее возможное число решений.

Задание 4. Чистка журнала заказов маркетплейса

Дан файл с журналом заказов маркетплейса, который вы можете скачать в форматах XLSX, ODS или CSV.
Каждая строка файла содержит три поля в порядке `delivery_hours` (вещественное число), `label` (строка) и `rating` (вещественное число).
Сначала оставьте только строки, где время доставки попадает в диапазон: $1 \leqslant$ `delivery_hours` $\leqslant 96$. Затем в столбце `label` замените все пустые значения модой — самым частым значением этого столбца после фильтрации.
Далее нормируйте `rating`: пусть среди оставшихся строк $\texttt{rating}_{\min}$ — минимальный рейтинг, а $\texttt{rating}_{\max}$ — максимальный. Для любой строки её нормированный рейтинг вычисляется по формуле

$$\frac{\texttt{rating} - \texttt{rating}_{\min}}{\texttt{rating}_{\max} - \texttt{rating}_{\min}}.$$

Сколько строк после всех преобразований одновременно удовлетворяют следующим условиям:

- `label` равен моде;
- нормированный рейтинг не меньше 0.8;
- `delivery_hours` $\leqslant 48$?

Ответ: 711

Критерий оценивания: точное совпадение ответа — 100 баллов

Максимальный балл за задание — 100

Решение. Решение выполняется последовательно по условиям задачи. Сначала из таблицы удаляются строки с нереалистичным временем доставки, остаются только значения `delivery_hours` $\in [1, 96]$. Затем в столбце `label` все пропуски (включая пустые строки) заменяются наиболее часто встречающимся после фильтрации значением. После этого столбец `rating` приводится к шкале $[0, 1]$ с помощью min–max-нормализации, результат сохраняется в новом столбце `rating_norm`. На последнем шаге подсчитывается количество строк, в которых одновременно выполняются условия: значение `label` совпадает с модой, `rating_norm` ≥ 0.8 и `delivery_hours` ≤ 48.

Решение на языке Python:

```python
import pandas as pd
import numpy as np

df = pd.read_csv("orders6_raw.csv")

df = df[(df["delivery_hours"] >= 1) & (df["delivery_hours"] <= 96)].copy()

lbl = df["label"].replace("", np.nan)
mode_label = lbl.mode(dropna=True)
mode_label = mode_label.iloc[0] if not mode_label.empty else "unknown"
df["label"] = lbl.fillna(mode_label)

r_min = df["rating"].min()
r_max = df["rating"].max()
den = (r_max - r_min)
```

```
df["rating_norm"] = (df["rating"] - r_min) / den if den != 0 else 0.0

answer = (
    (df["label"] == mode_label)
    & (df["rating_norm"] >= 0.8)
    & (df["delivery_hours"] <= 48)
).sum()

print(answer)
```

Задание 5. Сжатие размерности

Ограничение по времени: 1 секунда

Ограничение по памяти: 256 мегабайт

В машинном обучении часто возникает проблема: данных слишком много, а ресурсов слишком мало. Например, вы обучаете модель для распознавания изображений, и каждое изображение описывается миллионом признаков (по числу пикселей). Если хранить и обрабатывать все признаки, модель получится очень тяжёлой и будет работать медленно. Чтобы справиться с этим, используют приём сжатия размерности. Его смысл — это удалить или преобразовать те признаки, которые не дают новой информации. Например:

- если у всех объектов один и тот же цвет фона, то этот признак можно убрать;
- если признак не меняется, он не помогает отличать объекты друг от друга.

Рассмотрим упрощённый вариант этой идеи.

Дана выборка объектов a, состоящая из n элементов a_i. Каждый объект a_i представляется двоичной строкой длины 10. Получается таблица признаков: строки — объекты, столбцы — признаки (первый бит — первый признак, второй бит — второй признак и так далее). Все признаки бинарные: каждый бит равен либо 0, либо 1.

Если какой–то признак одинаков у всех чисел, он не несёт информации и должен быть удалён.

Сколько признаков останется после удаления одинаковых?

Формат входных данных

В первой строке входных данных даётся одно число n ($1 \leqslant n \leqslant 1000$) — количество объектов в выборке a.

В следующих n строках даётся по одному целому числу a_i ($0 \leqslant a_i < 1024$) — битовые представления объектов.

Формат выходных данных

Выведите одно целое число — количество оставшихся после удаления признаков.

Примеры

стандартный ввод	стандартный вывод
3 741 121 164	7

Замечание

В первом тестовом примере в массиве a было всего 3 числа: 741, 121, 164. В двоичном виде они выглядят как 1011100101, 0001111001, 0010100100. Удалены будут 2–й, 5–й и 9–й признаки. Останется 7 признаков.

Решение

В этой задаче нет идейной сложности, есть реализационная: нужно уметь получать 10 бит числа. Далее для каждого бита убедиться в том, правда ли что его значения во всех числах совпадают. Для этого достаточно сравнить значение этого бита у всех чисел со значением у первого числа.

Вспомогательный код выделения 10 младших битов числа x:

```
vector<int> bits;
int x;
for (int i = 0; i < 10; ++i) {
    bits.push_back(x % 2);
    x /= 2;
}
```

На практике можно сделать ещё проще: для каждого бита достаточно понять, встречается ли он среди чисел как 0 и как 1. Это удобно проверяется через побитовые операции: `v_or` содержит единицы в битах, где хотя бы у одного числа стоит 1, а `v_and` содержит единицы там, где у всех чисел стоит 1. Тогда в битах, где значения у чисел не совпадают, `v_or` и `v_and` различаются, то есть их XOR равен 1. Остаётся посчитать количество единиц в a = `v_or` $\oplus$ `v_and`.

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    cin >> n;
    vector<int> v(n);
    for (int i = 0; i < n; ++i) cin >> v[i];

    int v_or = v[0];
    int v_and = v[0];
    for (int i = 0; i < n; ++i) {
        v_or |= v[i];
        v_and &= v[i];
    }

    int a = v_or ^ v_and;

    int cnt = 0;
    while (a > 0) {
        cnt += a % 2;
        a /= 2;
    }

    cout << cnt << '\n';
    return 0;
}
```

Максимальный балл за задание — 100

Задание 6. Разнообразие разбиения

Решающие деревья — один из базовых методов машинного обучения. Они применяются для самых разных задач: от рекомендаций фильмов до диагностики по медицинским признакам. По сути, дерево — это последовательность вопросов о признаках, которые делят выборку объектов на группы. Классический алгоритм строится так, чтобы после каждого вопроса данные в ветвях становились как можно более однородными по классам — ведь чем однороднее ветка, тем проще в ней принять окончательное решение.

Однако у такого подхода есть недостаток: дерево может переобучаться, слишком подстраиваясь под конкретные данные. Чтобы изучить альтернативные принципы разбиения, исследователи решили рассмотреть противоположную идею — иногда выбирать разбиение, при котором обе ветви оказываются максимально разнообразными по составу классов. В этой задаче вам предстоит реализовать модуль, находящий порог признака, который даёт наибольшее разнообразие.

Дана обучающая выборка из n объектов. Каждый объект описывается одним числовым признаком x_i и меткой класса $y_i \in \{0, 1\}$. Рассмотрим целочисленный порог t и определим разбиение выборки на два блока:

- левый блок $L(t)$ — все объекты с номерами i, для которых $x_i \leqslant t$;
- правый блок $R(t)$ — все объекты с номерами i, для которых $x_i > t$.

Обозначим:

- $n_l = |L(t)|$ и $n_r = |R(t)|$ — количество объектов в левом и правом блоках;
- p_l — доля объектов класса 1 в левом блоке;
- p_r — доля объектов класса 1 в правом блоке;

Мера разнообразия одного блока определяется как

$$d = p(1 - p),$$

где p — доля объектов класса 1 в этом блоке. Общая оценка разбиения порогом t — это взвешенная по размерам блоков сумма мер разнообразия левого и правого блоков:

$$d_{\text{split}}(t) = \frac{n_l}{n} p_l(1 - p_l) + \frac{n_r}{n} p_r(1 - p_r),$$

где $n = n_l + n_r$.

Найдите такое целое значение порога t, при котором величина $d_{\text{split}}(t)$ максимальна. Если несколько порогов дают одинаковое значение разнообразия, выберите наименьший из них.

Формат входных данных

В первой строке входных данных дано единственное целое число n — количество объектов ($2 \leqslant n \leqslant 10^4$). В следующих n строках даны по два целых числа x_i и y_i — признак и класс i–го объекта ($|x_i| < 10^{12}$, $y_i \in 0{,}1$. Обратите внимание, что в данных могут встречаться повторяющиеся признаки и не гарантируется, что y_i будут совпадать при совпадающих x_i.

Формат выходных данных

Выведите одно целое число — оптимальное значение порога t, при котором величина $d_{\text{split}}(t)$ максимальна. Если несколько значений t дают одинаковое значение разнообразия, выведите наименьший из таких порогов.

Система оценки

Баллы за каждую подзадачу начисляются только в случае, если все тесты для этой подзадачи успешно пройдены.

Подзадача	Баллы	Дополнительные ограничения
1	20	Все метки классов одинаковы ($y_i = 0$ или $y_i = 1$)
2	80	$2 \leqslant n \leqslant 10^4$

Примеры

стандартный ввод	стандартный вывод
5 1 0 2 1 3 0 5 0	2

Замечание

Рассмотрим все возможные пороги $t \in 1, 2, 3, 4$. Для каждого порога вычислим меру разнообразия

$$d_{\text{split}} = \frac{n_l}{n} p_l(1 - p_l) + \frac{n_r}{n} p_r(1 - p_r),$$

где p_l и p_r — доли единиц в левой и правой группах соответственно.

t	Левый блок ($x_i \leqslant t$)	Правый блок ($x_i > t$)	p_l	p_r	d_{split}
1	[0]	[1, 0, 1, 0]	0	0.5	0.2
2	[0, 1]	[0, 1, 0]	0.5	$\frac{1}{3}$	0.2(3)
3	[0, 1, 0]	[1, 0]	$\frac{1}{3}$	0.5	0.2(3)
4	[0, 1, 0, 1]	[0]	0.5	0	0.2

Максимальное значение $d_{\text{split}} = 0.2333$ достигается при $t = 2$ и $t = 3$, но по условию при равенстве выбирается наименьший порог: $t = 2$.

Решение

Дана выборка из n объектов с признаком x_i и меткой $y_i \in \{0, 1\}$. Нужно найти целочисленный порог t, при котором величина разнообразия разбиения $d_{\text{split}}(t) = \frac{n_l}{n} p_l(1 - p_l) + \frac{n_r}{n} p_r(1 - p_r)$ максимальна, где левый блок задаётся условием $x_i \leq t$, правый — $x_i > t$. Рассматриваем только пороги, при которых оба блока непустые, а при равенстве значений берём минимальный t.

Перебирать все целые пороги нельзя (диапазон может быть огромным). Ключевое наблюдение: значение разбиения меняется только при переходе через какое-то значение x, поэтому достаточно рассматривать пороги вида $t = x$ для уникальных значений признака.

Отсортируем пары (x_i, y_i) по x_i и будем идти слева направо группами одинакового x. Поддерживаем количество единиц в левом блоке `l1` и общее количество элементов слева `l`. Справа аналогично: `r` $= n -$ `l` и `r1` $=$ `total_ones` $-$ `l1`. Переходя к очередному значению t, мы переносим всю группу с $x = t$ влево (так как условие $x \leq t$), после чего, если справа что-то осталось, считаем $d_{\text{split}}(t)$ и обновляем ответ. Такой проход после сортировки работает за $O(n)$, итого сложность $O(n \log n)$.

```
import sys

def solve() -> None:
    input = sys.stdin.readline
    n = int(input())
    data = [tuple(map(int, input().split())) for _ in range(n)]
    data.sort()

    total_ones = sum(y for _, y in data)

    best_t = None
    best_val = -1.0

    l = 0
    l1 = 0
    i = 0
    while i < n:
```

```
        t = data[i][0]

        grp_ones = 0
        j = i
        while j < n and data[j][0] == t:
            grp_ones += data[j][1]
            j += 1

        l += (j - i)
        l1 += grp_ones
        r = n - l
        r1 = total_ones - l1
        if l > 0 and r > 0:
            # p(1-p) = (ones/size) * (1 - ones/size)
            p_l = l1 / l
            p_r = r1 / r
            val = (l / n) * (p_l * (1 - p_l)) + (r / n) * (p_r * (1 - p_r))

            if val > best_val:
                best_val = val
                best_t = t

        i = j

    print(best_t)

if __name__ == "__main__":
    solve()
```

Максимальный балл за задание — 100