Задача A. Два альбома

(*Фольклор.*)

Для каждого альбома определим массив, в котором будем хранить уникальные номера марок этого альбома. Пусть `a[n]` и `b[m]` — массивы из этих номеров. В задаче требуется найти количество общих элементов массивов `a[n]`, `b[m]` и вывести эти элементы.

Подзадача 1. Простым перебором элементов двух массивов можно набрать 30 баллов.

Подзадача 2. Для нахождения общих элементов можно сравнить каждое число массива `a[n]` с каждым числом массива `b[m]`. Алгоритмическая сложность такого решения — $O(n \cdot m)$, оно набирает 60 баллов.

Подзадача 3. Объединим массивы `a[n]` и `b[m]` в один массив `c[n + m]` и отсортируем его по возрастанию. Тогда общие элементы исходных массивов будут встречаться в массиве `c[n + m]` ровно два раза. Осталось пройти по всему массиву и подсчитать количество *соседних* повторяющихся элементов и сами эти элементы.

Другая возможность полного решения — использование функции `set_intersection` из библиотеки шаблонов <algorithm> языка `C++`. Для этого нужно сначала отсортировать массивы `a[n]` и `b[m]`, а затем к полученным после сортировки массивам применить указанную функцию.

Алгоритмическая сложность таких решений — $O(n \log n + m \log m)$, они набирают 100 баллов.

Приведём основной фрагмент кода на языке `C++`:

```
cin >> n >> m;
vector <int> c(n + m);
for (int i = 0; i < n + m; ++i)
    cin >> c[i];
sort(c.begin(), c.end());

vector <int> v;
for (int i = 1; i < n + m; ++i)
    if (a[i] == a[i - 1]) v.push_back(a[i]);

cout << v.size() << endl;
for (int i = 0; i < v.size(); ++i)
    cout << v[i] << ' ';
return 0;
```

Задача B. Факториал

(Автор задачи — *Киндер М.И.*)

Подзадача 1. Если длина записи числа $n!$ не больше 3, решить задачу можно несложным перебором значений $n!$, где $1 \leqslant n \leqslant 6$.

Подзадача 2. Предварительно вычислим значения факториалов $n!$ для всех чисел n от 1 до 20. Затем сравним заданную строку с записью каждого вычисленного факториала. Если эти строки отличаются только одним символом, то находим цифру, соответствующую символу #.

Подзадача 3. Если запись числа $n!$ содержит более 20 символов, то для вычисления значений $n!$ можно воспользоваться длинной арифметикой, а затем опять сравнить заданную строку с записью $n!$, как это описано в решении подзадачи 2.

Подзадача 4. Полное решение задачи достаточно простое и не требует дополнительных идей, связанных с предварительным подсчётом значений $n!$ или длинной арифметики.

Пусть $n!$ — записанное на доске число. Отметим, что если $n \geqslant 6$, число $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot \ldots \cdot n$ обязательно делится на 9. По признаку делимости сумма цифр записанного на доске числа $n!$ также делится на 9. Отсюда вытекает следующий алгоритм нахождения стёртой цифры.

Подсчитаем сумму всех цифр в строке s (за исключением, разумеется, «#») и найдём остаток r при делении этой суммы на 9. Тогда ненулевая цифра, заменённая символом #, равна $9 - r$.

Если же $n < 6$, запись числа $n!$ содержит не более трёх цифр, из которых одна цифра заменена символом #. Все эти случаи легко решаются простым перебором:

— при $n \leqslant 3$ запись $n!$ состоит из одной цифры, и подойдёт любая из цифр 1, 2 или 6;

— при $n = 4$ ответом будет цифра 2, если заменили первую цифру, или 4, если заменили вторую цифру в записи $n!$;

— при $n = 5$ ответом будет цифра 1, если заменили первую цифру, или 2, если заменили вторую цифру.

Приведём основной фрагмент кода на языке C++:

```
string s;
cin >> s;

for (int i = 0; i < s.length(); ++i) {
   if (s[i] != '#') sum += s[i] - '0';
}
if (s.length() == 1) cout << 2;
if (s.length() == 2) cout << 6 - sum;
if (s.length() == 3)
   if (s[0] == '#') cout << 1; else cout << 2;
if (s.length() > 3) cout << 9 - sum % 9;
return 0;
```

Задача C. Сортировка мусора

(Автор задачи — *Киндер М.И.*)

Пусть $a[1][1]$, $a[2][1]$, $\ldots a[n][1]$ — количество отходов вида 1 в контейнерах с номерами 1, 2, $\ldots$, n соответственно. Общее количество отходов вида 1, очевидно, равно

$$s[1] = a[1][1] + a[2][1] + \ldots + a[n][1] = \sum_{k=1}^{n} a[k][1].$$

Предположим, что после сортировки все отходы вида 1 окажутся в контейнере с номером i_1. Тогда количество операций для перемещения отходов вида 1 в этот контейнер будет равно $s[1]-a[i_1][1]$. Аналогичным образом, подсчитаем количество операций для перемещения отходов вида 2, 3, $\ldots$, n в контейнеры с номерами i_2, i_3, $\ldots$, i_n соответственно. Тогда общее число операций, необходимое для сортировки всего мусора, равно

$$s = (s[1] - a[i_1][1]) + (s[2] - a[i_2][2]) + \ldots + (s[n] - a[i_n][n]) = \sum_{i=1}^{n}\sum_{k=1}^{n} a[i][k] - \sum_{k=1}^{n} a[i_k][k].$$

Двойная сумма в правой части этого равенства — это сумма всех чисел исходного массива $a[n][n]$, и она не зависит от выбора контейнеров i_1, i_2, $\ldots$, i_n. Число операций s будет наименьшим только, если сумма $\sum_{k=1}^{n} a[i_k][k]$ принимает наибольшее значение. Другими словами, в двумерной таблице-массиве $a[n][n]$ необходимо выбрать n элементов $a[i_1][1]$, $a[i_2][2]$, $\ldots$, $a[i_n][n]$ — по одному из каждой строки и из каждого столбца — так, чтобы сумма выбранных чисел была *наибольшей*.

Для оценки сложности алгоритма подсчитаем количество вариантов выбора таких n элементов в массиве $a[n][n]$. Это количество совпадает с количеством вариантов расстановки n ладей на шахматной доске $n \times n$, в которых ни одна из них не угрожает другой, то есть равно $n! = 1 \cdot 2 \cdot \ldots \cdot n$.

Подзадача 1. При $n = 2$ количество вариантов в переборном алгоритме равно $2! = 2$. Другими словами, достаточно выбрать наибольшее число из двух сумм $a[1][1] + a[2][2]$ и $a[1][2] + a[2][1]$. Это решение оценивается в 20 баллов.

Подзадача 2. При $n = 3$ количество вариантов в переборном алгоритме равно $3! = 6$ и необходимо выбрать наибольшую из шести сумм, каждая из которых состоит из трёх слагаемых. Это решение оценивается ещё в 20 баллов.

Подзадача 3. При $n = 4$ задача также решается аналогичным перебором.

Подзадача 4. Для реализации переборного алгоритма в общей ситуации сгенерируем все $n!$ перестановок n-элементного множества $\{1, 2, \ldots, n\}$. Затем для каждой перестановки $(i_1, i_2, \ldots, i_n)$ вычисляем сумму вида $\sum_{k=1}^{n} a[i_k][k]$. Теперь осталось выбрать наибольшую среди $n!$ таких сумм.

При $n \leqslant 10$ количество вариантов не превосходит $10! < 4 \cdot 10^6$.

Такое переборное решение оценивается в 100 баллов.

Задача D. Кенгурёнок и Тигра

(Автор задачи — *Киндер М.И.*)

Прежде всего, отметим, что наименьший по длине маршрут перемещения кенгурёнка из точки A в точку B будет наименьшим и по количеству прыжков.

Подзадача 1. Пусть длина d прыжка кенгурёнка больше расстояния AB. Тогда кенгурёнок сможет попасть из точки A в точку B за два прыжка — сначала он прыгает в точку C, отстоящую от точек A и B на расстоянии d, а затем уже из точки C прыгает в точку B. Для вычисления координат точки C сначала находим середину M отрезка AB, составляем вектор $\overrightarrow{MT}$, перпендикулярный вектору $\overrightarrow{AB}$. Затем в направлении вектора $\overrightarrow{MT}$ от точки M откладываем отрезок MT' (вектор) длины $\sqrt{d^2 - \frac{1}{4}AB^2}$. Осталось к координатам точки M прибавить координаты смещения MT'.

Реализация этой идеи позволяет решить задачу на 25 баллов.

Подзадачи 2 и 3. Пусть точки A и B находятся на горизонтальной или вертикальной прямой. Тогда вычислительные формулы для нахождения промежуточных точек, в которые нужно прыгать кенгурёнку, достаточно простые.

Сначала находим расстояние AB и проверяем, делится ли AB без остатка на длину прыжка d. Если это так, то количество прыжков равно частному от деления AB на d, а координаты промежуточных точек-прыжков получаются из координат точки A смещением по горизонтали (подзадача 2) или вертикали (подзадача 3) на величину $\frac{i}{q} \cdot AB$, где i — номер прыжка, q — частное от деления AB на d.

Если же AB не делится нацело на d, то общее число прыжков равно 2, если $AB < 2 \cdot d$, и равно $q = \lceil \frac{AB}{d} \rceil$, если $AB > 2 \cdot d$. (Здесь $\lceil x \rceil$ — наименьшее целое, которое не меньше чем x.) Для того чтобы попасть из точки A в точку B кенгурёнок сначала делает $q - 2$ прыжков по прямой AB, приближаясь к точке B, а затем, когда оставшееся расстояние до точки B будет уже меньше $2d$, ещё двумя прыжками, как в описано в подзадаче 1, попадает в точку B.

Реализация этих идей позволяет решить задачу на 75 баллов.

Подзадача 4. Идея решения задачи в общей ситуации такая же, как и в подзадачах 2 и 3. Формулы для вычисления координат промежуточных прыжков незначительно усложняются, поскольку приходится учитывать угол наклона прямой AB.

Реализация этих идей в общем случае позволяет решить задачу на 100 баллов.

Задача E. Простые суммы

(Автор задачи — *Киндер М.И.*)

Пусть n — количество участников соревнований. Перенумеруем участников соревнований числами от 1 до n и составим граф G, в котором вершины соответствуют номерам участников соревнований. Две вершины i и j графа G соединены ребром, если сумма исходных номеров $a[i]$ и $a[j]$ является простым числом.

Каждое разбиение игроков на пары соответствует некоторому выбору рёбер графа G. Поскольку каждый спортсмен может принимать участие не более чем в одной паре, соответствующая ему вершина может входить только в одно ребро графа G. Другими словами, любые два ребра, входящие в требуемое разбиение, не имеют общих вершин. В теории графов такие рёбра называют

попарно несмежными, а набор из попарно несмежных рёбер — *паросочетанием* (или независимым множеством рёбер графа).

Наибольшее паросочетание (или максимальное по размеру паросочетание) — это такое паросочетание, которое содержит максимальное количество рёбер. Число паросочетания $\nu(G)$ графа G — это число рёбер в наибольшем паросочетании. (У графа может быть несколько наибольших паросочетаний.)

Таким образом, в задаче требуется найти число паросочетания и само наибольшее паросочетание графа G.

Подзадача 1. Ясно, что число паросочетания $\nu(G)$ не превосходит $\frac{n}{2}$. При небольших ограничениях на количество вершин n задачу можно решить несложным перебором.

Подзадача 2. Для нахождения наибольшего паросочетания в произвольном графе можно воспользоваться алгоритмом Эдмондса или алгоритмом поиска максимального потока. Асимптотическая сложность таких решений $O(n^4)$, и они проходят тесты для всех $n \leqslant 100$.

Подзадачи 3 и 4. Разобьём множество всех вершин графа G на два подмножества (доли): в одну часть включим все *чётные* числа, во вторую — все *нечётные* числа. Очевидно, сумма любых двух чётных или двух нечётных чисел является чётной, то есть не может быть простым числом. Отсюда следует, что каждое ребро графа G соединяет вершины только из разных долей. Такой граф называется *двудольным*. В этом случае можно воспользоваться более эффективным алгоритмом Куна поиска наибольшего паросочетания, его сложность — $O(n^3)$.

В зависимости от технической реализации этого алгоритма получим решения, которые проходят тесты для всех $n \leqslant 1500$ (подзадача 3) или $n \leqslant 2500$ (подзадача 4). Для решения подзадачи 4 достаточно указать явное описание долей графа G сразу на этапе считывания данных.