Задача A. Хамелеоны

(*Фольклор.*)

Если хамелеонов какого-нибудь цвета, скажем, белого, — чётное количество, то, разбивая их на пары, добьёмся, что все они превратятся в чёрных хамелеонов. Таким образом, все хамелеоны острова будут чёрного цвета.

Если же хамелеонов каждого цвета — нечётное число, то все хамелеоны не смогут окраситься в один цвет. Действительно, после каждой встречи число хамелеонов белого или чёрного цвета либо не меняется, либо изменяется на 2, то есть остаётся *нечётным* числом, и поэтому никогда не станет равным 0.

Таким образом, в задаче нужно проверить чётность чисел b и w. Если они оба нечётные, выводим `no`; если хотя бы одно чётное, выводим `yes`. Кроме того, для корректной работы алгоритма нужно учесть, что b и w должны быть переменными типа long long.

Основной фрагмент кода на языке `C++`:

```
long long w, b;
cin >> w >> b;

if ((w % 2 == 1) && (b % 2 == 1)) cout "no" << endl;
else cout << "yes" << endl;
```

Задача B. 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 C. 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 D. 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, ..., n в контейнеры с номерами i_2, i_3, ..., 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, ..., 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 $a[i_1][1]$, $a[i_2][2]$, ..., $a[i_n][n]$ — 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.