Задача 1. Две доминошки

Две доминошки $\boxed{a}\boxed{b}$ и $\boxed{c}\boxed{d}$ можно поставить рядом только, когда одно из чисел в паре $(a; b)$ совпадает с одним из чисел в паре $(c; d)$, то есть когда $a = c$ или $a = d$ или $b = c$ или $b = d$. В каждом из этих случаев легко восстанавливается порядок расположения доминошек. Если не выполняется ни одно из этих условий, выводим -1.

```
def solve():
    a, b = map(int, input().split())
    c, d = map(int, input().split())

    if a == c:
        print(a, b, c, d)
    elif a == d:
        print(a, b, d, c)
    elif b == c:
        print(b, a, c, d)
    elif b == d:
        print(b, a, d, c)
    else:
        print(-1)
```

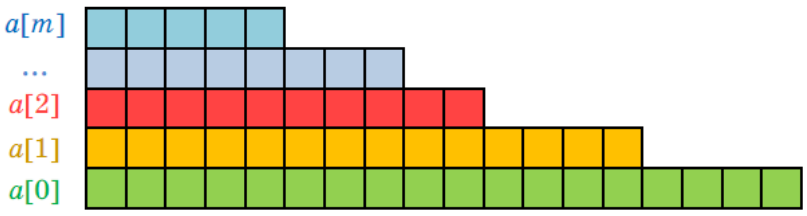
Задача 2. Потерянные этажи

Постановка задачи

- В последовательности натуральных чисел подсчитали количество чисел a_0, больших 0, количество чисел a_1, больших 1, и т.д.
- По набору положительных чисел $a_0, a_1, ..., a_{m-1}$ восстановить исходную последовательность.

Подзадача 1

Изобразим числа последовательности $a_0, a_1, ..., a_{m-1}$ в виде таблицы (диаграмма Юнга) Тогда исходная последовательность — это столбцы этой таблицы. Осталось посчитать число «этажей» в каждом столбце.



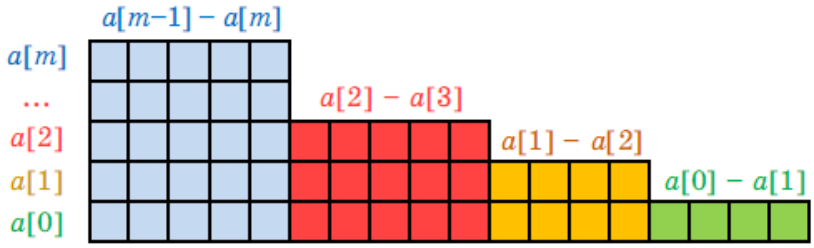
Общее количество домов в городе будет равно $a[0]$. Пусть $b[j]$ — количество этажей в j-м доме. Приведем пример кода на языке Pascal:

```
for i := 0 to m - 1 do
    for j := 1 to a[i] do inc(b[j]);
writeln(a[0]);
for i := 1 to a[0] do
    write(b[i],' ');
```

Сложность алгоритма — $O(m^2)$. Решение проходит тесты при всех $m \leqslant 40\,000$.

Подзадача 2

Как и в первом решении, сначала выведем значение $a[0]$ — общее количество домов в городе. Теперь заметим, что количество одноэтажных домов равно $a[0] - a[1]$, количество двухэтажных равно $a[1] - a[2]$ и так далее, наконец, количество m-этажных домов будет равно $a[m-1] - a[m]$.



Выводим число этажей $i+1$ в количестве $a[i]-a[i+1]$ для каждого $i=0\,..\,m-1$.
Пример кода на языке Pascal:

```
writeln(a[0]);
for i := m - 1 downto 0 do
     for j := a[i + 1] + 1 to a[i] do
          write(i + 1,' ');
```

Сложность алгоритма — $O(m)$.

Задача 3. Баскетбольный турнир

Постановка задачи

Надо научиться разделять две ситуации.

Полное решение

- Если команды играли круговой турнир, то общее количество матчей будет равно $\frac{n\cdot(n-1)}{2}$. Соответственно, суммарное количество побед будет $\frac{n\cdot(n-1)}{2}$.
- Если команды играли олимпийскую систему, то общее количество матчей будет равно $n-1$. Соответственно, суммарное количество побед будет $n-1$.

Посчитаем суммарное количество побед. Если оно будет равно $\frac{n\cdot(n-1)}{2}$, то ответом будет **Round-robin**, иначе **Olympic**.

Задача 4. Рассадка пассажиров

Подзадача 1

В текущей подзадаче можно создать двумерный массив a размером n на 3 и эмулировать процесс рассадки. Значение $a[i][j]$ равное *true* будет означать, что в ряду i на кресле j уже сидит пассажир, иначе кресло ещё пустое. При посадке очередного пассажира нужно проверить: сидит ли кто-нибудь между проходом и креслом пассажира, и если на указанных местах есть какие-либо пассажиры, то нужно увеличить ответ на количество таких пассажиров. Асимптотика решения $O(k\cdot m)$, а так как m — это константа, то имеем асимптотику $O(k)$. Количество пассажиров не больше общего количества мест, поэтому $k\leqslant n\cdot m$ и итоговая асимптотика решения $O(n)$.

Подзадача 2

Текущую подзадачу можно решить аналогично подзадаче 1, но нужно создать массив размером n на m. Различие лишь в том, что в подзадаче 1 для каждого места в ряду можно было точечно проверить остальные места в том же ряду, а в текущей подзадаче лучше использовать цикл. Асимптотика решения $O(n\cdot m^2)$.

Подзадача 3

В текущей подзадаче можно отсортировать массив по номеру ряда, при этом для одного и того же ряда нужно сохранить порядок пассажиров, который был в исходном списке. Другими словами, если пассажиры i и j имеют кресла на одном ряду и в изначальном списке пассажир i идёт до пассажира j, то и после сортировки пассажир i должен идти до пассажира j.

Так как мы отсортировали пассажиров по номеру ряда, то мы можем пойти по отсортированному массиву и обрабатывать ряд за рядом, для каждого ряда эмулируя процесс посадки. То есть в отличие от предыдущих подзадач можно создать одномерных массив размером m.

Итоговая асимптотика решения $O(k \cdot \log k) + O(k \cdot m) = O(k \cdot (\log k + m))$.

Подзадача 4

Решение основывается на решении из подзадачи 3, но теперь нужно обрабатывать каждого пассажирова быстрее чем $O(m)$. Для этого можно воспользоваться, например, деревом отрезков и при обработке очередного пассажира вычислять количество пассажиров на отрезке в ряду за $O(\log m)$. В таком случае итоговая асимптотика будет $O(k \cdot \log k) + O(k \cdot \log m) = O(k \cdot (\log k + \log m))$.

Подзадача 5

Заметим, что задача сводится к задаче о подсчёте количества инверсий. В таком случае нам не нужно хранить данные о текущем обрабатываемом ряде в одномерном массиве размером m, а можно обойтись массивом размером, равном количеству пассажиров в текущем ряду. Асимптотика решения $O(k \cdot \log k)$.

Задача 5. Лорды и гербы

Задачу можно представить в виде двудольного графа:

- левая доля — лорды (вершины $1 \dots n$);
- правая доля — гербы (вершины $1 \dots n$);
- ребро между лордом i и гербом j существует, если лорд i имеет герб j.

Тогда два лорда могут общаться напрямую, если они соединены через общую вершину-герб. Цепочка посредников соответствует пути в этом двудольном графе вида «лорд — герб — лорд — герб — ... — лорд». Минимальное число посредников равно длине такого пути, делённой на 2, минус 1 (если считать в вершинах, то количество вершин-лордов минус 1).

Поэтому задача сводится к поиску кратчайшего пути между вершинами-лордами A и B в двудольном графе, проходящего по рёбрам «лорд — герб — лорд».

Подзадача 1. $n \leqslant 30$, сумма гербов $\leqslant 3\,000$.

Можно явно построить граф лордов: две вершины соединены ребром, если у соответствующих лордов есть общий герб. Затем запустить BFS от A до B в этом графе. Проверку наличия общего герба для пары лордов можно делать двойным циклом по гербам: для каждой пары лордов (i, j) перебираем все пары гербов (h_i, h_j), где h_i принадлежит лорду i, а h_j — лорду j. Если найдём совпадение, то лорды i и j соединены ребром.

Оценим сложность алгоритма: проверка каждой пары лордов – $O(n^2)$ пар, затем для каждой пары делаем $O(g_i \cdot g_j)$ сравнений. В худшем случае $g_i, g_j \approx \frac{1}{n}\sum g_i \approx 100$, значит, общее число операций $\approx 30^2 \cdot 100^2 = 9 \cdot 10^6$, что приемлемо. Алгоритм BFS требует $O(n^2)$ операций. Таким образом,

Сложность алгоритма: $O(n^2 \cdot G^2)$, где G — среднее количество гербов у лорда. Память $O(n^2)$. При данных ограничениях работает быстро.

Подзадача 2. $n \leqslant 200$, сумма гербов $\leqslant 20\,000$.

Можно ускорить проверку наличия общего герба, отсортировав списки гербов у каждого лорда и используя метод двух указателей. Построение графа лордов всё ещё квадратично по n, но проверка общей вершины работает за $O(g_i + g_j)$.

Оценим сложность: проверка наличия общего герба имеет сложность $O(g_i + g_j)$ вместо $O(g_i \cdot g_j)$, затем сортировка списков гербов: $O(\sum g_i \log g_i)$. Далее, строим граф за $O(n^2 \cdot G)$, где G — средняя длина списка. В худшем случае нам нужно $200^2 \cdot 100 = 4 \cdot 10^6$ операций. Кроме того, BFS требует $O(n^2)$ операций. Таким образом,

Сложность алгоритма: $O(n^2 \cdot G)$, G — *среднее количество гербов*. Память $O(n^2)$.

Подзадача 3. $n \leqslant 1\,000$, сумма гербов $\leqslant 150\,000$.

Здесь уже квадратичное построение графа лордов не пройдёт. Нужно использовать двудольную структуру: BFS идёт по рёбрам «лорд → герб → лорд». Заведём списки смежности: для каждого лорда список его гербов и для каждого герба список лордов, имеющих этот герб. BFS запускаем от лорда a, чередуя переходы: из лорда во все его гербы, из герба во всех лордов, имеющих этот герб. Посещённые гербы и лорды отмечаем, чтобы не ходить по кругу.

Оценим сложность алгоритма: построение списков требует $O(\sum g_i)$ операций, затем BFS, в котором каждое ребро «лорд–герб» и «герб–лорд» будет просмотрено не более одного раза, поэтому общая сложность $O(\sum g_i)$. Таким образом,

Сложность алгоритма: $O(n+\sum g_i)$, то есть линейная от общего количества рёбер в двудольном графе, память $O(n + \sum g_i)$.

Подзадача 4. [Полное решение.] $n \leqslant 1\,500$, сумма гербов $\leqslant 300\,000$.

Решение такое же, как для подзадачи 3, поскольку алгоритм имеет линейную сложность от общего количества гербов. Ограничения увеличены, но алгоритм остаётся эффективным. Особенности реализации:

- нужно аккуратно работать с памятью, используя, например, векторы (динамические массивы) в C++.
- для быстрой проверки общего герба у A и B можно использовать булевый массив размера n.
- для восстановления пути нужно хранить для каждой вершины (лорда и герба) предыдущую вершину в BFS. Каждый путь в BFS имеет вид: A (лорд) → герб$_1$ → лорд$_1$ → герб$_2$ → лорд$_2$ → ... → B. Нас интересуют только лорды-посредники (лорд$_1$, лорд$_2$, ...). Их количество на единицу меньше, чем количество рёбер «герб–лорд» в пути (или количество рёбер «лорд–герб» минус 1). При восстановлении пути из B двигаемся назад по ссылкам, собирая только лордов (кроме A и B), затем разворачиваем порядок.

Итоговая *сложность алгоритма*: $O(\sum g_i)$, то есть линейная от общего количества рёбер в двудольном графе, память $O(n + \sum g_i)$.

Таким образом, полное решение, проходящее все подзадачи, — это двудольный BFS, описанный в подзадачах 3 и 4. Реализация алгоритма состоит из следующих шагов:

1. Считать n, списки гербов для каждого лорда.
2. Построить списки: для каждого лорда список его гербов, для каждого герба список лордов.
3. Считать A и B. Проверить, есть ли у них общий герб (вывести 0, если есть).
4. Запустить BFS от A:
   - Очередь хранит пары (*тип вершины*, *номер*), где тип указывает, лорд это или герб.
   - Из лорда переходим во все его гербы (если герб не посещён).
   - Из герба переходим во всех лордов, имеющих этот герб (если лорд не посещён).
   - Для каждой вершины запоминаем предыдущую вершину (для восстановления пути).
5. Если B достигнут, восстановить цепочку посредников и вывести ответ.
6. Иначе вывести -1.

Задача демонстрирует важность выбора подходящего представления графа. Наивное построение графа лордов работает только для маленьких n. Для больших ограничений необходимо использовать двудольную структуру, что позволяет решить задачу за линейное время от общего количества гербов.