Задача A. Светофор

Ответы в формате: $input \to output$. $jasyl \to 0$, $sary \to 1$, $qyzyl \to 2$.

Задача B. Кратчайшие пути в новом графе

Напишем алгоритм Дейкстры, но в новом графе может быть много рёбер. Заметим, что нам не обязательно каждый раз делать переход. Достаточно уметь вытаскивать минимальную не посещённую вершину.

Будем поддерживать для каждой вершины из изначального графа множество рёбер, которые ещё не посещены. Заметим, что минимальным может быть только ребро с минимальным весом. Для того чтобы посчитать значение в этом ребре, нам понадобится структура данных. Тогда чтобы найти минимальное не посещенное ребро, нам нужно будет взять минимальное среди n кандидатов.

При обработке ребра в Дейкстре будем для каждого его конца убирать это ребро из списка посещённых, добавлять само ребро в структуру и находить для нового минимума его значение.

В итоге наша структура данных должна уметь поддерживать следующие операции:

1. Найти минимальное значение для данного x среди $x \cdot w_e + dp_e$.
2. Добавить новую запись (dp_e, w_e).

Мы можем реализовать *online convex hull trick* или *li chao tree*.

Если заметим, что записи добавляются в порядке возрастания dp_e, можно написать вариацию *convex hull trick*, где для линий (k_i, b_i) b_i добавляется по возрастанию. И искать ответ бинарным поиском, или же поддерживать указатель на оптимальную линию, так как x запрашивается тоже по возрастанию.

В итоге $O(m \cdot \log(m))$.

Задача C. Восстановите массив

Так как каждое число не меньше 1 и не больше N, то только число N делится на N. Значит, B_N - это количество чисел, равных N. Если убрать их и обновить значение B_i, то оставшиеся числа будут не больше $N - 1$. Этот процесс нужно будет повторить N раз.

Решенеие на python:

```
n = int(input())
b = list(map(int, input().split()))
ans = []

for i in range(n, 0, -1):
    for j in range(1, b[i - 1] + 1):
        ans.append(i)
    for j in range(1, i):
        if i % j == 0:
            b[j - 1] -= b[i - 1]
print(*ans)
```

Задача D. Очередная задача про запросы в массиве

Давайте поддерживать массив блоками одинаковых чисел. Например, массив $[1, 1, 0, 0, 0, 3, 3, 1]$ будет представлен в виде массива троек: $[(1, 2, 1), (3, 5, 0), (6, 7, 3), (8, 8, 1)]$, где каждая тройка (l, r, x) означает, что для всех i $(l \leqslant i \leqslant r)$, $a[i] = x$.

Каждый запрос может удалить некоторые записи и добавить до трех новых. Одно значение относится к самому отрезку, и два значения добавляются, если концы отрезка «разрезают» блок.

Чтобы найти MEX чисел на отрезке, нужно извлечь все блоки и посчитать первое число, которое не встречается. Это будет работать за количество блоков на отрезке, что, казалось бы, приведет к $O(n^2)$.

Однако можно заметить что каждый блок рассматривается не более одного раза. Всего блоков не более $n + 3q$. Таким образом, суммарная сложность составит $O(n + q)$.

Для поддержки блоков можно использовать структуру данных, такую как *set* или *segment tree*. В итоге будет $O((n+q)\log n)$.

Задача E. Юрты

Если центр в точке P, то юрты можно разделить на 2 типа: юрты с симметричной (относительно P) парой и без. Для каждой юрты без симметричной пары мы должны добавить в ответ $+1$.

Давайте для каждой точки плоскости P посчитаем, сколько юрт **не будут** требовать добавления симметричной юрты если центр аула будет P. Пусть это значение $M[P]$, тогда добавить юрт потребуется $n - M[P]$.

Переберем все пары точек (A, B). Эта пара будет симметрична относительно точки $P = (A+B)/2$, поэтому мы увеличим значение $M[(A+B)/2]$ на 2. Так же для каждой точки A увеличим значение $M[A]$ на 1. Тогда ответом будет являться значение $n - \max_A M[A]$.

Для хранения значений M можно воспользоваться структурой данных *map*. Время работы: $O(n^2 \log(n))$.

Задача F. Арман и игра-конструктор

Будем считать, что $n \leqslant m$. Если $n > m$, мы всегда можем перевернуть матрицу на 90 градусов.

Заметим, что каждое простое кроме 2 нечетное. Найдем и посчитаем количество всех простых от 3 до $n \times m$. Обозначим это количество через k.

Если $k = 0$, то мы можем вывести что угодно. С этого момента будем считать, что $k > 0$.

Если $n = 1$, то у нас всегда будут столбцы с нечетной суммой. Так что ответа не существует.

Если $k = 2$, то эти два простые можно поставить в одну строку, но тогда в столбцах будет нечетная сумма. И наоборот. Так что тут ответа тоже не существует.

Если k нечетное, ответа тоже не существует. Потому что если в каждой строке сумма должна быть четной, значит сумма всей матрицы тоже должна быть четной.

Если k нацело делится на 4, можно расставить квадраты 2×2 в матрице. Поскольку каждый такой квадрат не меняет четность, это всегда работает.

Однако, что будет если необходимое количество квадратов не будет влезать в матрицу? Мы утверждаем, что такого случая никогда не будет, так как k — количество **простых** до $n \times m$. А простых не так уж и много по сравнению с общим количеством клеток в матрице (количество простых чисел до N можно приблизительно оценить как $\frac{N}{\log N}$).

А что если k не делится на 4, но делится на 2? Например $k = 6$? Мы можем построить аналогичный квадрат 3×3, заполненный следующим образом:

```
XX.
.XX
X.X
```

Здесь, «`X`» обозначает место, куда мы поставим простое число. Легко видеть, что это является решением для $k = 6$. Отсюда вылезает еще один случай.

Если $n = 2$, ответа не существует, поскольку квадрат 3×3 просто не поместится в матрицу.

Иначе, ответ всегда можно построить. Сперва поставим квадрат 3×3 в любой из углов и вычтем 6 простых из числа k. Теперь k будет точно делиться на 4, а это мы уже умеем решать.

Конечная асимптотика: $O(\sum nm)$

Задача G. Красно-черное дерево

Посчитаем ответ для фиксированной раскраски.

Подвесим дерево и посчитаем cnt_v — количество красных вершин в поддереве v. Тогда ответ будет равен $\sum \min(cnt_i, k - cnt_i)$. (доказательство остается читателю в качестве упражнения)

При перекраске одной вершины изменятся значения k и cnt_i для вершин на пути до корня.

Для каждой вершины посчитаем $dp_{v,x,y}$ — ее вклад в ответ, если k изменить на x и cnt_v на y.

Тогда при перекраске вершины v:

- Если v изначально была красной:

  - $dp_{u,k-1,cnt_i-1}$ для вершин на пути до корня
  - $dp_{u,k-1,cnt_i}$ дли всех остальных

- Если v изначально была черной:

  - $dp_{u,k+1,cnt_i+1}$ для вершин на пути до корня
  - $dp_{u,k+1,cnt_i}$ дли всех остальных

Это значит, что нужно посчитать только $O(n)$ значений dp и поддерживать их сумму на этом пути до корня.