Задача A. Смешарики: Быстрое преобразование Фурье

Так как в каждом ряду w сидений, каждый ряд после 1-го добавляет w к номеру места в нумерации. Таким образом, ответ равен $w \cdot (x - 1) + y$.

Задача B. Время суток

Чтобы решить задачу, достаточно считать строку, получить из неё текущее количество часов и минут и сравнить их с числами из условия задачи.

```
a = tuple(map(int, input().split(":")))
if (6, 36) <= a <= (11, 46):
    print("Morning")
elif (11, 47) <= a <= (17, 58):
    print("Day")
elif (0, 43) <= a <= (6, 35):
    print("Night")
else:
    print("Evening")
```

Задача C. Авантюра

Для каждой строки из входа надо проверить, что она состоит из четырёх символов и что в ней чётность символов совпадает с чётностью позиций (в 1-индексации).

```
def check(x):
    if len(x) != 4:
        return False

    for i in range(4):
        if int(x[i]) % 2 == i % 2:
            return False

    return True

s = input().split()
ans = True

for c in s:
    ans = ans and check(c)

print('Yes' if ans else 'No')
```

Задача D. Гнездовские игры

Создадим список C длины N, состоящий из списков, изначально пустых. Хотим, чтобы список C_i содержал номера мальчиков, живущих в i-й семье.

В цикле на K итераций будем считывать информацию о детях. Если очередной ребёнок — это девочка, ничего не делаем. Иначе нужно добавить номер этого ребёнка в список, относящейся к семье, в которой живёт этот ребёнок.

Заведём список ans длины N, изначально полностью заполненный значениями `False`. Хотим, чтобы в ans_i лежал ответ на вопрос, будет ли i-й в порядке ввода ребёнок участвовать в «Гнездовских играх».

Теперь нужно заполнить список ans корректными значениями. Переберём номер семьи i и определим, какие дети из этой семьи будут участвовать в соревнованиях. У нас есть список мальчиков в этой семье — C_i.

- Если список пуст, то в этой семье никто не будет участвовать.

- Если список имеет нечётную длину, то номер мальчика, представляющего эту семью на соревнованиях, — это $C[i][len(C[i])/2]$. В список ans по этому индексу нужно записать `True`.
- Если список имеет чётную длину, то эту семью на играх будут представлять двое мальчиков — $C[i][len(C[i])/2-1]$ и $C[i][len(C[i])/2]$. В список ans по этим индексам нужно записать `True`.

Будем выводить ответ для каждого ребёнка в отдельной строке. Если ans_i равен `True`, то i-й ребёнок участвует в «Гнездовских играх», и нужно вывести `Yes`. Иначе нужно вывести `No`.

Задача E. Командировки

Поскольку каждая командировка длится неделю, суммарно Поликарп был в командировках $7n$ дней. Поскольку $n \leqslant 2 \cdot 10^5$, то $7n \leqslant 1.4 \cdot 10^6$ — такое количество дней с лёгкостью поместится в структуру данных, поддерживающую быструю вставку и поиск элементов (в `Python` можно использовать `set`, в `C++` — `std::unordered_set` или `std::set`). Поместим все номера дней, когда Поликарп был в командировках, в такую структуру данных и при каждом запросе будем искать номер из запроса в этой структуре. Решение работает за $O(n+q)$ ($O((n+q) \cdot \log n)$ при использовании `std::set` в `C++`) по времени и $O(n)$ по памяти. Также возможно решение с использованием бинарного поиска.

Задача F. Такси

Переберём четыре варианта поворота (без поворота, на 90°, на 180° и на 270° по часовой стрелке), а затем проверим, что у двух матриц не существует клетки, в которой стоят одинаковые символы.

Задача G. Громкость и три бита

Заметим, что любое число x, такое что $x \geqslant 24$, можно набрать с помощью суммы какого-то количества трёхбитных чисел 7, 11, 13, 14, 19.

Из этого следует, что для любого $n \geqslant 24$ ответ `Yes`.

Найдем все $0 \leqslant n < 24$ для которых ответ `Yes`: 0, 7, 11, 13, 14, 18, 19, 20, 21, 22.

- 0 можно набрать, ничего не взяв;
- 7 можно набрать, взяв трёхбитное число 7;
- 11 можно набрать, взяв трёхбитное число 11;
- 13 можно набрать, взяв трёхбитное число 13;
- 14 можно набрать, взяв трёхбитное число 14;
- 18 можно набрать, взяв сумму трёхбитных чисел 7 и 11;
- 19 можно набрать, взяв трёхбитное число 19;
- 20 можно набрать, взяв сумму трёхбитных чисел 7 и 13;
- 21 можно набрать, взяв трёхбитное число 21;
- 22 можно набрать, взяв трёхбитное число 22.

Для всех остальных $0 \leqslant n < 24$ не из чисел 0, 7, 11, 13, 14, 18, 19, 20, 21, 22 ответ `No`.

Задача H. Добрая задача с добрым условием про арбузы

В задаче требуется найти количество подотрезков массива, сумма на которых не равна K. Будем для простоты считать X — количество подотрезков с суммой, равной K. Тогда ответ — это `количество вообще всех подотрезков` $- X = \frac{n \cdot (n+1)}{2} - X$.

Решение 1:

Нередко в задачах на поиск количества подотрезков, удовлетворяющих некоторому условию, можно применить такую идею: переберём правую границу r и для каждой правой границы быстро узнаем количество подходящих левых.

Давайте заранее насчитаем $pref_i$ — массив префиксных сумм. Дальше перебираем правую границу искомого подотрезка в порядке возрастания. Нужно узнать количество таких l, что $pref_r - pref_{l-1} = k$, то есть $pref_{l-1} = pref_r - k$. Можно хранить все уже рассмотренные $pref_i$ в словаре d. Тогда количество подходящих l равно $d[pref_r - k]$, так как $l - 1 \in [0, r - 1]$.

Сложность работы: $\mathcal{O}(n)/\mathcal{O}(n \log n)$ в зависимости от сложности работы словаря.

Решение 2:

Решим более простую задачу, когда в массиве не может быть нулей. Введём обозначение $sum(l, r) = \sum_{i=l}^{r} a_i$.

Пусть для некоторой левой границы l_1 существует некоторая правая граница r_1, что $sum(l_1, r_1) = k$. Заметим, что таких r_1 для фиксированной l_1 не более одной штуки, так как для всех $r' < r_1$ $sum(l_1, r') < sum(l_1, r_1) = k$, а для всех $r' > r_1$ $sum(l_1, r') > sum(l_1, r_1) = k$ (это следует из того, что в массиве все числа положительные).

Теперь возьмём $l_2 = l_1 + 1$, тогда, если и существует такое r_2, что $sum(l_2, r_2) = k$, то обязательно $r_2 > r_1$, так как $sum(l_2, r_1) < k$, ведь $a_{l_1} > 0$.

То есть если перебирать левую границу и двигать l от 1 до n, то потенциальная правая граница подходящего подотрезка будет двигаться только вправо. Значит, можно честно перебирать левую границу, затем двигать правую, пока $sum(l, r) < k$. Если сумма всё-таки равна k, значит, отрезок $[l, r]$ подходит.

Более простую задачу научились решать, вернёмся к исходной.

Аналогичным образом найдём для каждой левой границы минимальную правую r_1, что для всех $r \geqslant r_1$ $sum(l, r) \geqslant k$, и минимальную r_2, что для всех $r \geqslant r_2$ $sum(l, r) > k$. Тогда для фиксированной l все подходящие правые границы лежат на $[r_1, r_2)$.

Сложность работы: $\mathcal{O}(n)$.

Задача I. Гирлянда из флажков

Если бы максимальная длина гирлянды-палиндрома была чётной, было бы достаточно отсортировать для каждой буквы все возможные стоимости её взять, затем брать для каждой буквы две минимальные стоимости, если они ещё не были взяты, и среди всех возможных букв выбирать такую минимальную пару. Можно продолжать так делать, пока не кончится бюджет, и вывести в качестве ответа те буквы, которые вошли в бюджет. Это будет оптимальный ответ, так как никогда не имеет смысла брать не самую дешёвую пару. Взятые пары мы расставим симметрично относительно центра гирлянды и получим палиндром.

Ответ не всегда бывает чётной длины, поэтому случай нечётной длины нужно рассмотреть отдельно. Для этого переберём букву, которая будет стоять в центре (от `a` до `z`), и сразу же возьмём флажок минимальной стоимости с ней. Далее осталось просто решить задачу для чётной длины палиндрома, убрав выбранный флажок и переразбив флажки для выбранного символа на пары (начиная со второго по стоимости флажка, а не с первого). Если наивно повторить решение для чётного случая, получится асимптотика $\mathcal{O}(C \cdot (n \log n))$, но его можно улучшить.

Заметим, что после применения решения для чётной длины палиндрома мы можем получить отсортированный по возрастанию стоимостей массив всех пар букв (пары получены описанным способом разбиения). Теперь будем так же перебирать букву, которая будет стоять в середине, и построим новый массив пар стоимостей для этой буквы, убрав из стоимостей минимальную (её мы ставим в середину). Отсортируем новые пары для текущей буквы, и теперь задача сводится к тому, чтобы получить из двух отсортированных массивов (массива для всех букв, который мы получили ещё до перебора буквы, и массива новых пар для буквы) один отсортированный. Это можно сделать алгоритмом слияния двух массивов за сумму их длин (поддерживая по указателю на текущее место в каждом массиве, и каждый раз добавляя в новый массив число, которое меньше, и двигая соответствующий указатель). Чтобы учесть, что в массиве для всех букв остались некорректные пары для текущей перебираемой буквы, можно просто пропускать все такие вхождения в массиве в очередной итерации алгоритма слияния. Также заметим, что на самом деле можно не создавать новый отсортированный массив, а сразу применять алгоритм, описанный для случая палиндрома чётной длины, только изначально нужно добавить к стоимости набираемой гирлянды букву, которая стоит

в середине.

Сортировка всех букв вместе будет за $\mathcal{O}(n \log n)$, сортировки букв по отдельности также дадут $\mathcal{O}(n \log n)$, так как суммарно всех букв n, а перебор буквы и слияние произойдёт за асимптотику $\mathcal{O}(C \cdot n)$. Таким образом, получаем решение за $\mathcal{O}(n \cdot (\log n + C))$.

Задача J. Тимофей и подготовка к ЧеРеКОШу

Решение задачи состоит из двух замечаний:

1. Если среди всех a_i существует такое, что: $2a_i > \sum_{i=1}^{n} a_i + 1$, то мы не сможем составить строку.

   Доказательство:

   Пусть $a_i = x$, а все остальные a_i в сумме дают y. Нам известно, что $2x > y + x + 1$, то есть $x - 1 > y$. Чтобы разделить все x цифр i, нужно хотя бы $x - 1$ других цифр, а у нас их строго меньше.

   В ином случае по тем же соображениям мы сможем составить строку так, чтобы разделить одинаковые цифры.

2. Чтобы минимизировать число, нужно в каждый момент времени ставить минимальную цифру, которую можем сейчас поставить. Если мы поставим не минимальную цифру, то это гарантированно ухудшит ответ, а количество разделителей, которые нам могут понадобиться для разделения каких-то цифр, в любом случае уменьшится на 1.

Реализация:

Мы будем строить ответ поэлементно, в каждый момент времени у нас будет готов какой-то префикс ответа.

Во время построения поддерживаем $last$ — последнюю цифру числа (изначально $last = 0$, так как число не может начинаться с нуля). Когда хотим поставить очередную цифру, делаем следующее:

1. Если есть такое i, что $2a_i = \sum_{i=1}^{n} a_i + 1$, то, если можем, ставим эту цифру прямо сейчас, так как если мы поставим её на следующей итерации, то по доказанному, у нас получится некорректная строка.

2. Если первый пункт не реализовался, то ставим минимальную цифру, которую можем.

3. Если мы не можем поставить ни одну цифру, то завершаем процесс.

Если в итоге мы расставили не все цифры, то такого числа не существует.

Также отдельно нужно обработать случай, когда число равно 0.

Задача K. Спасти любой ценой

Формализуем условие. Нам дан взвешенный ориентированный граф. По i-му ребру можно начать идти в момент t, только если существует целое k такое, что $d_i + k \cdot (g_i + p_i) \leqslant t < d_i + k \cdot (g_i + p_i) + g_i$. Также существуют запрещённые отрезки времени, в которые в вершине находиться нельзя.

Последнее условие неудобно, поэтому избавимся от него так: разделим каждую вершину на две и будем считать, что первая доступна до l_i секунды не включительно, а вторая — с $r_i + 1$ секунды. Тем самым мы сохраним инвариант, что чем раньше мы окажемся в вершине, тем лучше. В этом случае мы можем применить алгоритм Дейкстры. Внутри него, рассматривая очередное ребро (v, u), посмотрим на минимальное время, за которое мы можем добраться до v, пусть это t. Пусть $x = t \bmod (g_v + p_v)$. Если $x < g_v$, то начать переход по ребру можно прямо сейчас, иначе нужно ждать начала следующего цикла, то есть $(g_v + p_v - x)$ секунд. Здесь же нужно учесть, существует ли в момент начала перехода вершина v и в момент окончания перехода вершина u.

Если до вершины t (той, что существует до l_t секунды) можно добраться, пока она жива, то нужно понять, можно ли добраться из неё до любой безопасной вершины, то есть до любой вершины,

которая существует, **начиная** с какого-то момента, а это новые n вершин, полученных после разделения. Запустим алгоритм Дейкстры второй раз из вершины t и определим, можно ли добраться до любой такой вершины. Если можно, выведем ответ, иначе выведем максимальное l_i среди всех вершин, до которых мы можем добраться — это будет время смерти. Отметим, что условие, что с момента последнего перехода до смерти должна пройти хотя бы одна секунда, означает, что если вершина недостижима из-за аномалии в ней, то не нужно пытаться продлить себе жизнь, идя по ребру в неё.

Если же до вершины t добраться в срок нельзя (или пути вообще нет), то определим, можно ли добраться до безопасной вершины из s, и, аналогично первому случаю, выведем ответ.

Задача L. Яблоневый сад

Давайте по аналогии с кодированием напишем рекурсивную функцию раскодирования.

Пусть наша функция должна построить поддерево, в котором для всех вершин выполняется $lef < x_i < rig$ (другими словами, ключи в нашем поддереве могут лежать только в интервале (lef, rig)). Если очередное рассматриваемое по порядку число не принадлежит интервалу, то наше поддерево пустое, так как первым числом всегда идёт корень. В противном случае запустим расшифровку левого и правого поддерева, изменив границы допустимых ключей.

Расшифровка на языке C++:

```
int build_tree(int &uk, int lef, int rig, int n)
{
    if (uk == n + 1) return -1;
    int val = perm[uk];
    if (val <= lef || val >= rig) return -1;

    uk++;
    left_son[val] = build_tree(uk, lef, val, n);
    right_son[val] = build_tree(uk, val, rig, n);
    return val;
}
```

Заметим, что результат расшифровки всегда — корректное бинарное дерево поиска, так как мы соблюдаем ограничения на ключи в поддеревьях.

Если нам удалось расшифровать весь массив, то найденное дерево и есть ответ. Иначе закодированная запись некорректна.

Итоговая асимптотика — $\mathcal{O}(n)$.

Задача M. Витя и метро

Чтобы решить первую часть задачи и посчитать сумму расстояний для изначального метро, можно использовать динамическое программирование на дереве с переподвешиванием. Подвесим дерево за любую вершину и посчитаем для каждой вершины размер её поддерева. Далее посчитаем для каждой вершины сумму расстояний от неё до вершин её поддерева. Чтобы это сделать, переберём детей вершины, возьмём сумму расстояний от ребёнка до всех вершин его поддерева, увеличим каждое расстояние на 1, прибавив размер поддерева ребёнка, и просуммируем эту величину по всем детям вершины. В итоге мы получим сумму расстояний от корня до всех вершин. Чтобы посчитать расстояния от других вершин, применим технику переподвешивания дерева. Пусть корнем в данный момент является вершина v. Переберём её детей. Пусть мы сейчас рассматриваем ребёнка u. «Забудем», что u являлся ребёнком v, откатив пересчёт динамики для v, и сделаем v ребёнком u, пересчитав динамику u через v. Теперь корнем является u, и мы знаем сумму расстояний от u до всех вершин. Пройдёмся по дереву обходом в глубину, каждый раз переподвешивая дерево за ту вершину, в которой мы сейчас находимся в обходе. В итоге мы узнаем сумму расстояний от любой вершины до всех остальных, и просуммировав их, получим ответ на задачу.

Чтобы решить вторую часть, посмотрим, как изменяется сумма расстояний между всеми парами вершин в дереве при добавлении цикла длины 3. Рассмотрим цикл $u - w - v$. Пусть рёбра $u - w$

и $w - v$ в дереве уже были, а мы добавили ребро $u - v$. Тогда длины всех путей, которые раньше проходили через вершины $u - w - v$, уменьшились на 1, так как теперь можно пройти по новому ребру $u - v$. То есть, уменьшились на 1 длины всех путей от какой-либо вершины из поддерева u до какой-либо вершины из поддерева v, если подвесить дерево за w, а значит, сумма расстояний уменьшилась на $2 \cdot sub_u \cdot sub_v$, где sub_v — размер поддерева v.

Чтобы найти оптимальные u и v, вновь применим переподвешивание. Пусть в данный момент корнем является w. Переберём его детей и найдём двух из них, u и v, с максимальными размерами поддеревьев. Запомним значение $2 \cdot sub_u \cdot sub_v$ и выберем из них наибольшее по всем w. Асимптотика: $\mathcal{O}(n)$.

Задача N. Далёкие университеты

Будем перебирать наибольший общий делитель d. Найдём все такие рёбра, что их веса делятся на d, такие веса будут равны: d, $2d$, $3d$, ...

Для каждого d выделим множество рёбер, которые разбивают исходный граф на компоненты связности. Нам нужно определить, существует ли среди этих компонент связности хотя бы одна, размер которой не меньше k.

Для решения этой задачи подойдёт структура данных `CHM` (прочитать про неё можно на `https://ru.algorithmica.org/cs/set-structures/dsu/`).

Для каждого значения d будем решать задачу независимо.

Будем поддерживать размер максимальной компоненты связности — max_rang.

Для каждой компоненты, корень которой равен v, будем хранить её размер $rang_v$.

Будем по порядку объединять два множества, которые связаны ребром (u, v). После объединения двух компонент, корни которых равны x и y соответственно $(x \neq y)$, появляется новая компонента, размер которой равен $rang_x + rang_y$, зная этот размер, обновим значение max_rang.

После объединения всех компонент остаётся проверить, что $max\_rang \geqslant k$, если это так, то обновим ответ.

Важное замечание: для каждого d не нужно сохранять отдельные массивы, в которых будут храниться размеры и предки. Достаточно создать отдельный массив $time_v$, в котором будет храниться последнее d, при котором обновлялись значения $rang_v$, p_v (где p_v — предок вершины v). Если для текущего d выполняется $time_v \neq d$, это означает, что вершина v не обновлялась при текущем d. В таком случае нужно вернуть $rang_v$, p_v к первоначальному состоянию, то есть проделать операции $rang_v = 1$, $p_v = v$.

Задача O. Сора против Стефани

В задаче требуется для q запросов понять, существует ли $t \in [1, k]$, такое что $Set([l, r]) = Set([t, k])$, где $Set([l, r])$ — множество чисел a_i, где $l \leqslant i \leqslant r$. Также, если t существует, то найти максимальную сумму затемнённостей комнат на $[t, k]$ для всех подходящих t.

Заметим, что все подходящие для нас t, если таковые существуют, будут лежать на одном отрезке. Это следует из того, что чем левее двигается граница t, тем сильнее размер $Set([t, k])$ может увеличиться, а мы хотим, чтобы множества, а значит, и их размеры, совпали.

Найдём максимальный и минимальный суффикс отрезка $[1, k]$ с p различными числами, где p — количество различных на отрезке $[l, r]$. Найти количество различных на отрезке $[l, r]$ — это базовая задача на персистентное дерево отрезков, но так как запросы даны в оффлайне, это можно сделать и обычным деревом отрезков, или даже деревом Фенвика. Отсортируем запросы по r и пройдём по массиву слева направо. При этом будем поддерживать для каждого x последнее найденное вхождение x, то есть наибольшее i, что $a_i = x$. Для каждого такого i, что i — последнее вхождение какого-либо числа, прибавим 1 к i-му элементу дерева Фенвика. Пусть мы прошли по первым k элементам массива. Рассмотрим все такие $[l, r]$, что $r = k$, и возьмём сумму на отрезке $[l, r]$ в дереве Фенвика. Она и будет равна количеству различных на отрезке $[l, r]$.

Далее найдём t_1, t_2 такие, что на $[t_2, k]$ — p различных, а на $[t_1, k]$ — $p + 1$ различных. Для этого ещё раз пройдёмся по массиву, как в предыдущей части, и используем спуск по персистентному ДО или Фенвику. Тогда только $t \in (t_1, t_2]$ могут нам подойти. Мы смогли проверить на совпадения количества различных, осталось проверить, что сами множества совпадают. Для этого будем

хранить какую-нибудь хеш-функцию на различных элементах, поддерживая это в персистентном дереве отрезков или в дереве Фенвика. Например, можно каждой игре сопоставить случайное число до 10^{18} и суммировать их, или ксорить их.

Теперь, если хеш и количество различных совпадают, то любое $t \in (t_1, t_2]$ удовлетворяет требованиям Стефани, осталось только максимизировать сумму на $[t, k]$. Её можно посчитать через разность двух суффиксных, тогда нам надо будет на $(t_1, t_2]$ находить наибольшую суффиксную сумму. Это можно сделать, использовав либо `sparse table`, либо обычное дерево отрезков.