Задача A. Серебряный статус в авиакомпании

При данных ограничениях задачу проще всего решать перебором вариантов. А именно, переберём, сколько пакетов по 30 000 будем менять на 5 000 квалификационных, а из оставшихся средств — сколько пакетов по 7 500 миль будем менять на 1 000 каждый. Из этих вариантов выберем тот, при котором количество квалификационных миль будет не менее 20 000, а общее число миль будет как можно больше.

Код программы на языке Python:

```
n = int(input())
k = int(input())
mx = 0
for i in range(n // 30000 + 1):
    for j in range((n - i * 30000) // 7500 + 1):
        nn = n - i * 30000 - j * 7500
        kk = k
        if nn < k:
            kk = nn #  квалификационные мили были потрачены на обмен
        kk += i * 5000 + j * 1000 #  новое число квалификационных миль
        nn += i * 5000 + j * 1000 #  новое число миль
        if kk >= 20000: #  квалификационных миль хватает для статуса
            if nn > mx:
                mx = nn
if mx > 0:
    print(mx)
else:
    print(-1)
```

Но задачу можно решить и разбором случаев. Определим, сколько пакетов по 1 000 квалификационных миль нам не хватает до достижения 20 000 миль. Разделим это количество на пять и обменяем как минимум такое количество пакетов по 5 000 миль, остаток обменяем по 1 000 миль. Но если осталось четыре пакета по 1 000, то вместо них выгоднее обменять дополнительный пакет в 5 000 миль, потому что при той же стоимости мы получим в итоге на 1 000 миль больше. При этих необходимых операциях на обмены могут расходоваться и имевшиеся квалификационные мили, что может привести к невозможности получения серебряного статуса, даже когда общего количества миль на обмены хватает.

```
n = int(input())
k = int(input())
#  используем округление вверх до 1000 при подсчете недостающих пакетов
l = (max(0, 20000 - k) + 999) // 1000
if l % 5 == 4:
    #  выгоднее обменять лишний пакет в 5000 миль вместо четырех по 1000 миль
    l5 = (l + 1) // 5
    #  квалификационные мили с учетом возможного их уменьшения при обмене
    k = min(k, n - 30000 * l5) + 5000 * l5
    n -= 25000 * l5 #  оставшиеся мили
else:
    l1 = l % 5
    l5 = l // 5
    k = min(k, n - 30000 * l5 - 7500 * l1) + 5000 * l5 + 1000 * l1
    n -= (25000 * l5 + 6500 * l1) #  оставшиеся мили
if k >= 20000: #  квалификационных миль достаточно
    print(n)
else:
    print(-1)
```

Задача B. Офшоры

Пусть мы хотим сделать i-й элемент максимумом. Посчитаем $S_i = \sum\limits_{j=1\ldots n}^{j\neq i} \lfloor \frac{a_j}{x} \rfloor$, заметим что максимально мы можем сделать именно столько переводов на i-й счет со всех остальных суммарно (так как выполняется условие $y \leqslant x$, то если мы сделаем перевод с какого-то счета u на какой-то счет v, то $\lfloor \frac{a_u}{x} \rfloor$ уменьшится ровно на 1, а $\lfloor \frac{a_v}{x} \rfloor$ увеличится не более чем на 1, поэтому промежуточные переводы между счетами не помогут увеличить S_i). Таким образом задача свелась к тому чтобы научиться быстро считать S_i для каждого i. Для этого можно посчитать $S_{all} = \sum\limits_{j=1}^{n} \lfloor \frac{a_j}{x} \rfloor$, тогда $S_i = S_{all} - \lfloor \frac{a_i}{x} \rfloor$.

Задача C. Шифровка

Подгруппа 1 ($n \leqslant 10, k \leqslant 2$).

В этой подзадаче можно было перебрать все возможные варианты расшифровки (для каждой позиции выбрать, какую из двух букв мы возьмём) для каждого варианта определить его информативность и выбрать оптимальную расшифровку. Информативность строки можно найти, честно перебрав d и проверив, верно ли, что s представляет собой префикс длины d, повторённый $\frac{n}{d}$ раз. Такую проверку можно осуществить за $O(n)$, при этом проверять имеет смысл только те d, которые являются делителями n. Поскольку делителей n не более чем $2\sqrt{n}$, то нахождение информативности будет работать за $O(n\sqrt{n})$. Всего существует k^n различных вариантов расшифровок, поэтому общая асимптотика алгоритма — $O(k^n n\sqrt{n})$.

Подгруппа 2.

В этой подзадаче можно было заметить, что оптимальной будет расшифровка, содержащая только буквы `a` и `c`. Действительно, ведь при такой расшифровке буквы совпадают на всех парах позиций, на которых они вообще могли бы совпасть, поэтому информативность в таком случае будет минимально возможной. Построить требуемую расшифровку можно за $O(nk)$.

Подгруппа 3 ($k \leqslant 3$).

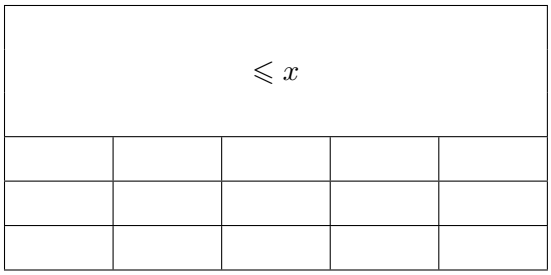
Давайте, как в первой подзадаче, будем перебирать d (делитель n) и проверять, можем ли мы получить строку с информативностью d. Для этого для каждого $1 \leqslant i \leqslant d$ надо проверить, что у позиций $i, i+d, i+2d, \ldots$ существует общая буква. Для этого переберём букву на позиции i и проверим, что на позициях $i+d, i+2d, \ldots$ эта буква также присутствует. Для позиции i проверку можно осуществить за $O\left(\frac{n}{d} \cdot k^2\right)$, для всех i это будет работать за $O(nk^2)$. Суммарно по всем d алгоритм будет работать за $O(n\sqrt{n}k^2)$. При аккуратной реализации с break такое решение может заходить на 100 баллов.

Полное решение.

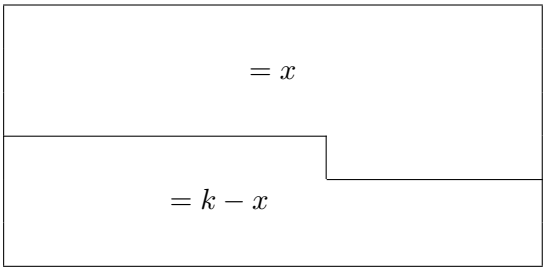
Идея: давайте для каждой позиции вместо k букв, которые на ней стоят, будем хранить массив длины 26, в котором для каждой буквы записано, присутствует она на этой позиции или нет. Теперь для каждого i мы можем за $O\left(26 \cdot \frac{n}{d}\right)$ проверить, что на позициях $i, i+d, i+2d, \cdot$ есть общая буква (перебрать одну из букв на позиции i и за $\frac{n}{d}$ проверить, что она подходит). Таким образом, общая асимптотика алгоритма составит $O(26 \cdot n\sqrt{n})$.

Задача D. Разрез таблицы

Пусть в таблице k единиц. Тогда максимально возможное произведение $\left\lfloor \frac{k}{2} \right\rfloor \cdot \left(k - \left\lfloor \frac{k}{2} \right\rfloor\right)$. Покажем, что ответ всегда достигается. Хотим выделить разрезом часть с $x = \left\lfloor \frac{k}{2} \right\rfloor$. Найдем последнюю строку, что сумма единиц на префиксе строк $\leqslant x$.



В следующей строке возьмем на суффиксе нужное количество элементов чтобы сумма сверху стала равна x.



После этого разрез восстанавливается из рисунка выше. Асимптотика $\mathcal{O}(n \cdot m)$.

Задача E. Любовь к комбинаторным объектам

В этой задаче нам нужно искать «кратчайшие» пути в графе, но рёбра на графе не являются взвешенными; время перехода по ребру из u в v зависит от момента времени, в который мы это делаем. Несмотря на это, идеи из стандартных алгоритмов всё ещё применимы*.

Пусть в момент времени T с человеком u провели воспитательную беседу, и мы хотим понять, когда с его соседом v проведут воспитательную беседу. Другими словами, мы хотим понять время перехода по ребру из u в v в момент времени T.

Нам нужно найти первый из моментов времени $a_v, a_v + t_v, \ldots$, который при этом позже T. Сдвинем нашу шкалу времени назад на a_v. Теперь нам нужно найти минимальное число, кратное t_v, большее $T - a_v$. Это будет

$$\left\lceil \frac{T - a_v + 1}{t_v} \right\rceil \cdot t_v$$

Таким образом, мы перейдём в вершину v в момент времени

$$a_v + \max\left(0, \left\lceil \frac{T - a_v + 1}{t_v} \right\rceil \cdot t_v\right)$$

Взяв $T = \mathrm{dist}_u$, мы можем пересчитать этой величиной dist_v.

В подгруппе 2 граф является деревом, то есть до каждой вершины существует единственный простой путь.

В подгруппе 3 время перехода по ребру всегда равно 1, поэтому мы можем использовать обычный поиск в ширину.

В подгруппе 5 время перехода по ребру не больше 20, поэтому мы можем использовать поиск в ширину с несколькими очередями (1-K BFS).

В подгруппах 7 и 8, а также в полном решении нужно применить алгоритм Дейкстры.

*Более формально, для алгоритмов поиска расстояний из одной вершины до всех остальных требуется выполнение трёх свойств:

- Монотонность: $\mathrm{Len}(\mathrm{path}) \leqslant \mathrm{Len}(\mathrm{path} + e)$
- Консистентность: если $\mathrm{Len}(\mathrm{path}_1) = \mathrm{Len}(\mathrm{path}_2)$, то $\mathrm{Len}(\mathrm{path}_1 + e) = \mathrm{Len}(\mathrm{path}_2 + e)$
- Оптимальность начала: если $\mathrm{Len}(\mathrm{path}_1) \leqslant \mathrm{Len}(\mathrm{path}_2)$, то $\mathrm{Len}(\mathrm{path}_1 + e) \leqslant \mathrm{Len}(\mathrm{path}_2 + e)$

Задача F. Черепашка наносит ответный удар

Для $n, m \leqslant 100$ можно написать переборное решение: перебираем, какую клетку выберет Рафаэль, и для каждого варианта считаем, сколько удовольствия получит Микеланджело. Это считается простейшей динамикой-черепашкой.

Для дальнейших решений выделим один из путей S из $(1,1)$ в (n,m), который на исходной пицце доставляет Микеланджело максимальное удовольствие. Клетки этого пути назовём $s_1, s_2, \ldots, s_{n+m-1}$. Заметим, что если Рафаэль не выберет ни одну из этих клеток, то Микеланджело выберет путь S и получит максимально возможное удовольствие (благодаря неотрицательным по удовольствию кускам пиццы Рафаэль всегда может сделать ход, который не увеличит доступное Микеланджело удовольствие). А значит, если Рафаэль претендует на что-то большее, он должен заблокировать одну из клеток $s_1, \ldots, s_{n+m-1}$.

Уже из одного этого замечания мы получаем решение за $\mathcal{O}(nm(n+m))$: давайте переберём клетку на пути, которую выберет Рафаэль, и для каждого из этих вариантов снова посчитаем ответ.

Для дальнейших продвижений заметим, что после блокировки клетки (i,j) у Микеланджело будет два варианта: получить на $2a_{i,j}$ меньше удовольствия, чем максимум, то есть $maxS - 2a_{i,j}$, или пройти из $(1,1)$ в (n,m), не посетив клетку (i,j). Первый вариант не требует дополнительных вычислений, а вот второй создаёт некоторые сложности. Давайте разберёмся с ними.

Для клетки (i,j), чтобы гарантировать, что путь не пройдёт через неё, достаточно показать, что на пути будет лежать клетка (a,b), для которой выполняется либо $a < i$ и $b > j$, либо $a > i$ и $b < j$. Отсюда появляется новое решение с той же асимптотикой, но которое уже можно будет оптимизировать: давайте переберём все клетки (a,b) и для каждой клетки s_k сохраним возможность обойти её за $dpS[a][b] + dpT[a][b] - a[a][b]$, где dpS и dpT — вспомогательные динамики, выдающие максимальное удовольствие при движении из $(1,1)$ в (a,b) и из (a,b) в (n,m) соответственно. По всем таким вариантам для каждой s_k нам нужно сохранить максимум; назовём его $dp[k]$. Тогда ответ формируется как $\min_k \big(\max\big(dp[k],\ maxS - 2a[s_k]\big)\big)$.

Теперь давайте оптимизируем и это решение. Заметим, что нам не нужно перебирать всевозможные клетки (a,b), через которые можно обойти клетку (i,j). Достаточно рассмотреть лишь клетки $(i+1, q < j)$ и клетки $(p < i, j+1)$. К сожалению, напрямую перебирать все такие клетки всё ещё долго. Однако заметим, что нам нужно уметь считать $\max\big(dpS[x][y] + dpT[x][y] - a[x][y]\big)$ для префиксов точек по x в конкретном столбце и по y в конкретной строке, а именно для наборов $(i+1,1), (i+1,2), \ldots, (i+1,j-1)$ и $(1,j+1), (2,j+1), \ldots, (i-1,j+1)$.

Значения таких префиксных максимумов можно предподсчитать. Таким образом, за $\mathcal{O}(nm)$ предподсчёта мы получаем возможность для каждой клетки (i,j) узнавать максимальное удовольствие маршрута, который не будет посещать её, за $\mathcal{O}(1)$. Итоговая асимптотика решения составляет $\mathcal{O}(nm + (n+m)) = O(nm)$.

Отметим также, что значительное количество баллов могло набирать решение за $\mathcal{O}(nm \log(nm))$, которое оптимизирует второе решение за $\mathcal{O}(nm(n+m))$ при помощи структур данных.