Задача №2025. Стенка на стенку

Бокс, каратэ, самбо... Классические боевые единоборства пресытили аудиторию. Поэтому известный спортивный канал запускает новый формат соревнований, основанный на традиционной русской забаве — боях стенка на стенку. В соревновании могут участвовать от двух до k команд, каждая из которых будет соперничать с остальными. Всего в соревновании примут участие n бойцов. Перед началом боя они должны разделиться на команды, каждый боец должен войти ровно в одну команду. За время боя два бойца сразятся, если они состоят в разных командах. Организаторы считают, что популярность соревнований будет тем выше, чем больше будет количество схваток между бойцами. Помогите распределить бойцов по командам так, чтобы максимизировать количество схваток между бойцами, и выведите это количество.

Исходные данные:

> В первой строке дано количество тестов T ($1 \le T \le 10$). В следующих T строках перечислены тесты. В каждой из них записаны целые числа n и k через пробел ($2 \le k \le n \le 10^4$).

Результат:

> Для каждого теста в отдельной строке выведите одно целое число — ответ на задачу.

Рабочий код

```python
s = int(input())

for i in range(s):
    n, k = map(int, input().split())
    a = [0] * k

    for j in range(k):
        a[j] = n // k

    for j in range(n % k):
        a[j] += 1

    res = 0
    for j in range(k-1):
        res += a[j] * (n - a[j])
        n -= a[j]

    print(res)
```

Объяснение алгоритма

Для начала нам нужно разделить наших бойцов по командам как можно более равномерно, иначе количество боев окажется меньшим. Далее достаточно лишь вычислить сумму ряда:

$$t^k + t^{k-1} + t^{k-2} + \cdots + t^0$$

Задача №1296. Гиперпереход

Гиперпереход, открытый ещё в начале XXI-го века, и сейчас остаётся основным способом перемещения на расстояния до сотен тысяч парсеков. Но совсем недавно физиками открыто новое явление. Оказывается, длительностью альфа-фазы перехода можно легко управлять. Корабль, находящийся в альфа-фазе перехода, накапливает гравитационный потенциал. Чем больше накопленный гравитационный потенциал корабля, тем меньше энергии потребуется ему на прыжок сквозь пространство. Ваша цель — написать программу, которая позволит кораблю за счёт выбора времени начала альфа-фазы и её длительности накопить максимальный гравитационный потенциал.

В самой грубой модели грави-интенсивность — это последовательность целых чисел p_i. Будем считать, что если альфа-фаза началась в момент i и закончилась в момент j, то накопленный в течение альфа-фазы потенциал — это сумма всех чисел, стоящих в последовательности на местах от i до j.

Исходные данные:

> В первой строке записано целое число N — длина последовательности, отвечающей за грави-интенсивность ($0 \le N \le 6 \cdot 10^4$). Далее идут N строк, в каждой записано целое число $p_i (-3 \cdot 10^4 \le p_i \le 3 \cdot 10^4)$.

Результат:

> Выведите максимальный гравитационный потенциал, который может накопить корабль в альфа-фазе прыжка. Считается, что потенциал корабля в начальный момент времени равен нулю.

Рабочий код

```python
count, maximal = 0, 0

s = int(input())

for i in range(s):
    elem = int(input())
    if count + elem > 0:
        count += elem
    else:
        count = 0
    if count > maximal:
        maximal = count

print(maximal)
```

Объяснение алгоритма

По сути нашей задачей является нахождение последовательности с наибольшей суммой. Для наилучшего решения нам нужно учитывать все числа, которые не обращают сумму в отрицательное значение. Если же найдется такое число, которое сделает сумму последовательности отрицательной, то начинаем счет заново. На каждом шаге нам нужно запоминать максимальную сумму.

Задача №1604. В Стране Дураков

Главный бульдог-полицейский Страны Дураков решил ввести ограничение скоростного режима на автомобильной трассе, ведущей от Поля Чудес к пруду Черепахи Тортиллы. Для этого он заказал у Папы Карло n знаков ограничения скорости. Папа Карло слабо разбирался в дорожном движении и поэтому изготовил знаки с разными ограничениями на скорость: 49 км/ч, 34 км/ч, 42 км/ч, и т.д. Всего получилось k различных ограничений: n_1 знаков с одним ограничением, n_2 знаков со вторым ограничением, и т.д. ($n_1 + \cdots + n_k = n$).

Бульдог-полицейский ничуть не расстроился, получив такие знаки, напротив, он решил извлечь из этого экономическую выгоду. Дело в том, что по Правилам дорожного движения Страны Дураков ограничение на скорость действует вплоть до следующего знака. Если на знаке написано число 60, это означает, что участок от данного знака до следующего нужно проехать ровно со скоростью 60 километров в час — не больше и не меньше. Бульдог распорядился расставить знаки так, чтобы обогатившимся на Поле Чудес автолюбителям во время своего движения по трассе приходилось как можно больше раз менять скорость. Для этого нужно расставить имеющиеся знаки в правильном порядке. Если Вы поможете бульдогу это сделать, то он готов будет поделиться с Вами частью своих доходов.

Исходные данные:

> В первой строке дано число k — количество различных типов знаков с ограничением скорости ($1 \le k \le 10000$). Во второй строке через пробел перечислены целые положительные числа $n_1, \ldots, n_k$. Сумма всех n_i не превосходит 10000.

Результат:

> Выведите n целых чисел в пределах от 1 до k — порядок, в котором нужно расставить по трассе имеющиеся знаки. Вне зависимости от того, какой знак стоит первым, считается, что, проезжая его, водитель меняет скорость, так как до этого ограничения не действовали. Если задача имеет несколько решений, выведите любое.

Рабочий код

```python
import heapq

k = int(input())
a = []
if k == 1:
    temp = input().split()
    print("1 " * int(temp[0]))
    exit()

temp = input().split()
for i in range(k):
    heapq.heappush(a, (-int(temp[i]), i + 1))
while all[0][0] != 0:
    first = heapq.heappop(a)
    second = heapq.heappop(a)
    print(first[1], end=" ")
    first = (first[0] + 1, first[1])
    if second[0] != 0:
        print(second[1], end=" ")
        second = (second[0] + 1, second[1])
    heapq.heappush(a, first)
    heapq.heappush(a, second)
```

Объяснение алгоритма

По приоритетной очереди распределяем кортежи, рассматриваем каждую пару, выводим числа, уменьшаем количество и возвращаем обратно в приоритетную очередь, пока первое слагаемое не достигнет 0.