Разбор задач

Задача 1. Треугольник из палочек

Напомним, что треугольник существует, если выполняется так называемое неравенство треугольника. Оно утверждает, что длина любой стороны треугольника всегда меньше суммы длин двух его других сторон.

Рассмотрим частный случай $a = b = c$. Очевидно, что в этом случае после каждой операции треугольник будет оставаться равносторонним (и, соответственно, существовать), пока все стороны не станут нулевой длины. Это произойдет после a операций.

Рассмотрим частный случай $a, b, c \leqslant 10^5$. Смоделируем описанный в условии процесс: будем уменьшать стороны на 1, пока выполняется неравенство треугольника.

```
a = int(input())
b = int(input())
c = int(input())
ans = 0
while a + b > c and a + c > b and b + c > a:
    a, b, c = a - 1, b - 1, c - 1
    ans += 1
print(ans)
```

Полное решение. Упорядочим a, b, c по возрастанию, тогда неравенство треугольника можно будет записать так: $c \leqslant a + b$.

Само упорядочивание удастся осуществить следующим образом: наименьшее и наибольшее из трёх данных чисел можно выразить через стандартные операции минимума и максимума. Среднее же из чисел получится найти, если из суммы трёх исходных чисел вычесть найденные значения наибольшего и наименьшего.

Спустя x операций это неравенство нарушится и превратится в равенство $= a + b$ (так как c уменьшается на 1, а $a + b$ на 2, то мы не «перескочим» равенство). Составим и решим уравнение:

$c - x = a + b - 2 \times x$

Отсюда $x = a + b - c$

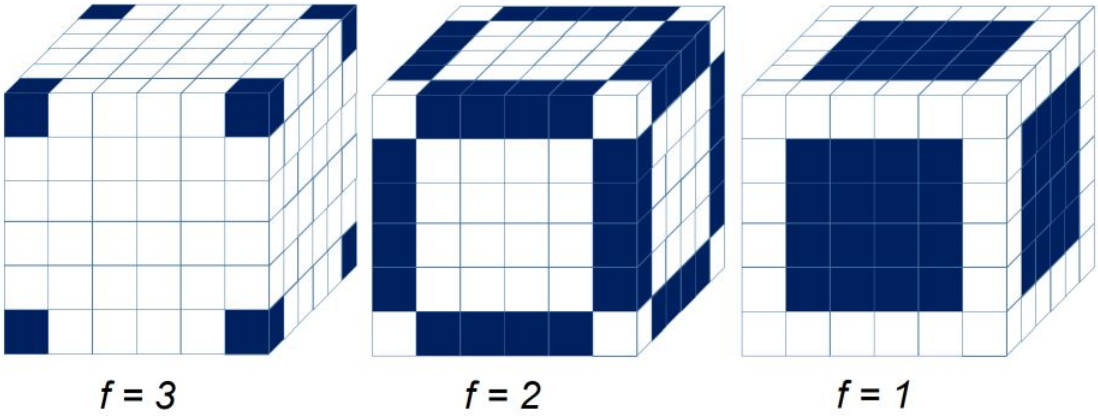
```
a = int(input())
b = int(input())
c = int(input())
a, b, c = min(a, b, c), a + b + c - min(a, b, c) - max(a, b, c), max(a, b, c)
ans = a + b - c
print(ans)
```

Задача 2. Раскрашенный куб

Самый простой случай – последний, при $f = 3$. Три грани окажутся окрашенными только у тех кубиков, которые расположены в углах исходного куба, их всегда 8.

При $f = 2$ две грани окажутся окрашенными только у тех кубиков, которые расположены на рёбрах исходного куба, на каждом ребре их будет по $n - 2$, всего рёбер у куба 12.

При $f = 1$ одна грань окажется окрашенной только у тех кубиков, которые расположены внутри каждой грани исходного куба, на каждой грани их будет по $(n - 2) \times (n - 2)$, всего граней у куба 6.



Наконец, при $f = 0$ нужно найти количество не окрашенных ни с одной из сторон кубиков. Можно из общего числа кубиков вычесть все рассмотренные в предыдущих случаях кубики:

$n^3 - 8 - 12 \times (n-2) - 6 \times (n-2) \times (n-2)$,

или сообразить, что все они образуют куб со стороной на 2 меньше, чем исходный куб, то есть $(n-2)^3$.

```
n = int(input())
f = int(input())
if f == 0:
    ans = (n - 2) ** 3
if f == 1:
    ans = 6 * (n - 2) ** 2
if f == 2:
    ans = 12 * (n - 2)
if f == 3:
    ans = 8
print(ans)
```

Задача 3. Антон и арбузы

Частичное решение

Промоделируем в явном виде процесс, описанный в условии. На каждом из d шагов добавим по единице ко всем затронутым ячейкам массива, а затем найдём в нём максимум и посчитаем количество максимумов по всему массиву.

```
n = int(input())
m = int(input())
d = int(input())

A = [[0] * m for _ in range(n)]

for i in range(d):
    x_i = int(input())
    y_i = int(input())
    for x in range(x_i):
        for y in range(y_i):
            A[x][y] += 1

ans = 0
```

```
for i in range(n):
    ans = max(ans, max(A[i]))

count = 0
for i in range(n):
    count += A[i].count(ans)

print(count, ans)
```

Сложность такого алгоритма составляет $O(nmd)$ по времени и $O(nm)$ по памяти, то есть работает он очень долго, да и хранить массив $n \times m$ мы не всегда можем, поэтому такое решение наберёт только 50 баллов.

Полное решение

Заметим, что на каждом шаге Антон поливает хотя бы одну строку и столбец. Следовательно, арбуз, лежащий в ячейке $(1, 1)$, будет полит ровно d раз, и это максимум, который является ответом на второй вопрос. Осталось понять, сколько ещё ячеек будут политы каждый день. Несложно видеть, что d раз будут политы в точности $\min x_i$ строк и $\min y_i$ столбцов, то есть их произведение и является ответом на первый вопрос.

```
n = int(input())
m = int(input())
d = int(input())

for i in range(d):
    x_i = int(input())
    y_i = int(input())
    n = min(n, x_i)
    m = min(m, y_i)
print(n * m, d)
```

Сложность такого алгоритма $O(d)$, а дополнительную память мы вообще не используем.

Задача 4. Платные музыкальные сервисы

Если $m = 1$, то подходящих сервисов нет, так как за два месяца будет заплачено не менее 2 рублей.

Если $m = 2$, то есть единственный подходящий сервис $a = 1, d = 1$.

Если $n, m \leqslant 1000$, то можно перебрать все варианты подходящих сервисов. Для этого переберём все возможные первые взносы a от 1 до $m - 1$, а для каждого такого значения a переберём все подходящие d от 1 до тех пор, пока выполняется условие задачи (значение $a+(n-1)\cdot d$ не превышает m). Если новое значение d уже не удовлетворяет этому условию, переходим к следующему a, а если удовлетворяет, то увеличиваем счетчик подходящих сервисов на 1.

Полное решение требует оптимизации перебора. Заметим, что при фиксированном d нам подойдут все такие a, что $a+(n-1)\cdot d \leqslant m$, то есть a образуют непрерывный отрезок от 1 до $m-(n-1)\cdot d$. Добавим к ответу длину этого отрезка.

Теперь переберем все такие d, что $d \cdot (n-1) \leqslant m$, и для каждого добавим к ответу $m-(n-1)\cdot d$.

Таким образом мы сможем избавиться от полного перебора всех вариантов и получить более быстрое решение, которое проходит все тесты.

Задача 5. Линейный футбол

Общий подход к решению задачи заключается в моделировании процесса перемещения мяча от игрока к игроку. Для этого нужно ввести переменную, в которой хранится текущее положение мяча i, после каждого шага изменяющееся на величину a_i. При этом необходимо хранить набор номеров уже сыгравших спортсменов и для текущего положения мяча знать: играл ли футболист,

получивший мяч, или нет. Если играл — получаем цикл, если мяч вышел за границу поля — значит, был забит гол в соответствующие ворота.

Данный процесс можно выполнять с разной эффективностью. Для тестов первой подзадачи достаточно для каждого игрока независимо промоделировать движение мяча и понять, куда тот попадет, после чего соответствующим образом пометить этого игрока. Данное решение работает неэффективно и для больших тестов получит вердикт «Превышение времени».

В тестах второй подгруппы все игроки, кроме самого правого, ударяют вправо. Промоделируем для самого правого футболиста процесс перемещения мяча от него к другим игрокам и запишем в его позицию итоговый результат. Далее, передвигаясь справа налево, для каждого игрока выясним, куда попадает мяч от него. Возможны следующие варианты: мяч попадает непосредственно в правые ворота (ставим R), либо мяч попадает к игроку, для которого уже известно, куда в итоге попадет мяч от него. Тогда для текущего футболиста ставим тот символ, которым помечен игрок, получивший от него мяч. Таким образом, за один проход получим результат игр для каждого игрока.

Для прохождения тестов третьей группы модернизируем предыдущее рассуждение. Для каждого из не более чем 10 игроков, перебрасывающих мяч на другую половину, промоделируем процесс перемещения мяча. Далее два раза пройдём по игрокам начиная от центра: сначала влево (от игрока мяч попадет либо сразу в правые ворота, либо к уже известному результатом игроку, этот результат и укажем для текущего). Затем пройдем от центра вправо с аналогичными рассуждениями. За линейное время получим ответ.

Для прохождения тестов четвертой подгруппы полезно отметить уже известный факт: если мяч от игрока A попал к игроку B, про которого известно, куда попадет от него мяч в итоге, то и игроку A нужно выставить тот же символ. Для начала найдем первого ещё не игравшего футболиста при перемещении слева направо. Перебирём игроков, к которым мяч попал от него, и сохраняем их в дополнительном массиве. После того как игра закончилась взятием ворот (а это верно для всех игроков в тестах этой группы), выставляем всем попавшим в массив игрокам этот результат и движемся далее вправо до следующего неигравшего игрока (для этого нужно отмечать, играл игрок или нет, что в данном случае выяснить просто, ведь для сыгравших игроков выставляется их результат). Так как обрабатываются только неигравшие игроки, то в итоге снова алгоритм линейный. Это решение не позволяет обрабатывать случай, когда игроки передают мяч по циклу.

Для полного решения в предыдущем случае следует еще отмечать некоторым символом (например @) всех сыгравших в данной игре (помимо помещения их в массив) и, если мяч попадает в ворота или к уже известному по своему результату игроку — выставлять всем сыгравшим в данной игре этот результат. А если мяч попадает к игроку, отмеченному символом @, то это значит, что игра зациклилась, тогда для всех сыгравших выставляем U. В любом случае после выставления результата очищаем массив для дальнейшего использования, либо заводим новый. Так как и в этом случае каждый игрок обрабатывается один раз, общая сложность полного решения снова линейна.