Альтернативный разбор задач

Этот разбор задач является шуточным. Все приведённые решения являются правильными и набирают 100 баллов, однако, используются чрезмерно «умные» алгоритмы. У всех этих задач есть и более простые решения, изложенные в основном разборе.

Задача 1. Подарки

Воспользуемся двоичным поиском по ответу. В качестве левой границы возьмём число подарков, которое точно можно собрать (0), в качестве правой границы — количество подарков, которое нельзя собрать (общее число конфет, делённое на n, увеличенное на 1).

Как проверить, можно ли собрать заданное число подарков (mid)? Сделаем это жадно: нам понадобится $mid\lfloor\frac{n}{3}\rfloor$ упаковок по 3 конфеты и $mid(n \bmod 3)$ упаковок по 1 конфете, при этом если нам не хватит упаковок по 3 конфеты, то их можно заменить на упаковки по 1 конфете.

Пример такого решения на языке Python.

```
n = int(input())
a = int(input())
b = int(input())
left = 0
right = (a + 3 * b) // n + 1
while right - left > 1:
    mid = (left + right) // 2
    count3 = n // 3 * mid
    count1 = n % 3 * mid
    if a > count1:
        count3 -= (a - count1) // 3
    if count1 <= a and count3 <= b:
        left = mid
    else:
        right = mid
print(left)
```

Задача 2. Пробежка

Воспользуемся двоичным поиском по ответу. Для этого нужно научиться проверять, сможет ли Рита пробежать mid дней так, что ей хватит заряда наушников. За mid дней суммарная величина заряда наушников равна $a \cdot mid$, а потрачено заряда было $b + (b + 1) + ... + (b + mid - 1)$. Последнее выражение по формуле суммы арифметической прогрессии можно преобразовать к виду $\frac{2b + mid - 1}{2} mid$

Таким образом, значение mid подходит в качестве новой левой границе, если величина накопленного за mid дней заряда (хранится в переменной $plus$) не меньше величины потраченного заряда (хранится в переменной $minus$).

Пример решения на языке Python.

```
a = int(input())
b = int(input())
left = 0
right = 2 * 10 ** 9 + 1
while right - left > 1:
    mid = (left + right) // 2
    plus = a * mid
    minus = (b + b + mid - 1) * mid // 2
    if plus >= minus:
        left = mid
    else:
```

```
        right = mid
print(right)
```

Задача 3. Лес

Данная задача решается двоичным поиском по ответу (количество полных циклов пройденных Мишей).

Введём систему координат, где ось OX направлена вправо (на восток), ось OY — вверх (на север), а начало координат находится в центре квадрата.

Циклом будем называть четыре перемещения: на A шагов на север, B шагов на восток, C шагов на юг и D шагов на запад. В результате перемещения по одному циклу Миша смещается на вектор (x, y), где $y = A - C$, $x = B - D$.

Если Миша совершил $n > 0$ полных циклов, и при перемещении по следующему циклу он вышел из леса, то и на кажом из последующих циклов он будет ещё дальше удаляться от начала координат и также выйдет из леса. Поэтому можно воспользоваться двоичным поиском.

Если Миша совершил n полных циклов, то он оказался в точке (nx, ny).

Функция `check(x, y)` проверяет, выйдет ли Миша из леса на очередном цикле, если в начале цикла он находился в точке (x, y). Эта функция возвращает количество шагов, которое Миша должен сделать на этом цикле, чтобы выйти из леса, а если на этом цикле Миша не выйдет из леса, то функция возвращает -1. Для этого функция последовательно выполняет четыре перемещения: $y = y + A$, $x = x + B$, $y = y - C$, $x = x - D$, и если после очередного перемещения координата точки станет по модулю не меньше K, то возвращает число пройденных шагов. Если после четырёх перемещений Миша не вышел из леса, то функция возвращает -1.

Далее двоичным поиском находится такое минимальное число циклов, после совершения которых Миша выйдет из леса на следующем цикле. Это значение будет храниться в переменной `right`. Тогда количество шагов до того, как Миша выйдет из цикла, равно $right \cdot (A + B + C + D)$ плюс результат вызова функции `check(x * right, y * right)`.

Также отметим, что на первом цикле Миша может выйти в каждом из четырёх направлений. Например, если $A \geqslant K$, то Миша сразу же выйдет из леса при первом движении на север, даже если потом окажется, что $C > A$. Поэтому отдельно нужно рассмотреть возможность выхода из леса во всех четырёх направлениях на первом цикле. Для этого вызовем функцию `check(0, 0)` в начале, если она вернула не -1, то это и будет ответом.

Пример решения на языке Python.

```
import sys
A = int(input())
B = int(input())
C = int(input())
D = int(input())
K = int(input())

def check(x, y):
    if abs(x) > K or abs(y) > K:
        return 0
    y += A
    if y >= K:
        return A - (y - K)
    x += B
    if x >= K:
        return A + B - (x - K)
    y -= C
    if y <= -K:
        return A + B + C - (-y - K)
    x -= D
```

```
    if x <= -K:
        return A + B + C + D - (-x - K)
    return -1

if check(0, 0) != -1:
    print(check(0, 0))
    sys.exit(0)

x = B - D
y = A - C

left = 0
right = 10 ** 9
while right - left > 1:
    mid = (left + right) // 2
    if check(x * mid, y * mid) == -1:
        left = mid
    else:
        right = mid
print(right * (A + B + C + D) + check(x * right, y * right))
```

Задача 4. Сериал

В этой задаче в данном числовом массиве нужно найти три таких индекса $i < j < k$, что $a_i < a_j < a_k$. То есть нам нужно найти в массиве строго возрастающую подпоследовательность длины 3.

Воспользуемся хорошо известным алгоритмом нахождения наибольшей возрастающей подпоследовательности сложности $O(n \log n)$ при помощи двоичного поиска (см., например, https://e-maxx.ru/algo/longest_increasing_subseq_log). Здесь `dp[i]` — это значение минимального элемента, которым может оканчиваться возрастающая последовательность длины `i`, а массив `prev` нужен для восстановления ответа.

Посчитав значения функции динамического программирования, посмотрим на значение элемента `dp[3]`. Если он не равен значению `INF`, то ответ существует и нужно его восстановить, иначе выведем 0.

```
import bisect

n = int(input())
a = [int(input()) for i in range(n)]

INF = 10 ** 9

dp = [0] + [INF] * n
prev = [0] * (n + 1)

for elem in a:
    i = bisect.bisect_left(dp, elem)
    if dp[i] > elem:
        dp[i] = elem
        prev[i] = dp[i - 1]

if dp[3] == INF:
    print(0)
else:
```

```
    k = n - 1
    while a[k] != dp[3]:
        k -= 1
    j = k - 1
    while a[j] != prev[3]:
        j -= 1
    i = j - 1
    while a[i] >= a[j]:
        i -= 1
    print(i + 1, j + 1, k + 1)
```

Задача 5. Длинный плакат

Эта задача решается жадным алгоритмом. Чтобы получившееся число было максимально большим, нужно вычеркнуть цифры так, чтобы на первом месте оказалась цифра 9. Это можно сделать, если среди первых $k+1$ цифр есть девятка, тогда вычеркнем все цифры до этой девятки (первой девятки на этом отрезке). Если же нет девятки, то надо взять максимальную цифру среди первых $k+1$ (первую из максимальных). После этого нужно уменьшить значение k на количество вычеркнутых цифр.

Далее будем искать следующую цифру, которую можно поставить, это также максимальная (первая из максимальных цифр) среди $k+1$ цифр, следующих за той, которую мы оставили (значение k при этом уменьшилось). Найдём эту цифру, снова уменьшим значение k и продолжим искать цифры дальше. Если k стало равно 0, то больше вычеркивать цифры нельзя и нужно добавить все оставшиеся цифры.

Чтобы эффективно реализовать это решение, нужно воспользоваться структурой данных «дерево отрезков», которая позволяет быстро (за $O(\log n)$) искать наибольший элемент на отрезке. Таким образом, сложность полученного решения будет $O(n \log n)$. Дерево отрезков должно возвращать нам первую из максимальных цифр, поэтому в дереве отрезков будем хранить не просто цифры, а значение `num[i] * M1 + M2 - i`, где `M1 = 10**7` и `M2 = 10**6`, то есть значение цифры домножается на большое число, и добавляется значение `M2 - i`, которое уменьшается с ростом `i`.

Такое решение работает в тестирующей системе 0,97 секунд, что не превосходит ограничения по времени в 1 секунду.

```
import math
import sys


def get_first_max(tree, idx, l, r, L, R):
  if r <= L or R <= l:
    return -1
  if l >= L and r <= R:
    return tree[idx]

  m = (l + r) // 2
  return max(get_first_max(tree, idx * 2 + 1, l, m, L, R),
             get_first_max(tree, idx * 2 + 2, m, r, L, R))


num = input()
k = int(input())
n = len(num)
N = 2**math.ceil(math.log2(n))

M1 = 10 ** 7
```

```
M2 = 10 ** 6
tree = [-1] * (2 * N)
for i in range(n):
  tree[N - 1 + i] = int(num[i]) * M1  + M2 - i
for i in range(N - 2, -1, -1):
  tree[i] = max(tree[2 * i + 1], tree[2 * i + 2])

i = 0
ans = ""
for _ in range(n - k):
  maximum = get_first_max(tree, 0, 0, N, i, i + k + 1)
  val = maximum // M1
  pos = M2 - maximum % M1
  ans += str(val)
  k -= pos - i
  i = pos + 1
  if k == 0:
    ans += num[i:]
    break

print(ans)
```