Разбор задач

Задача 1. Подарки

60 баллов можно набрать моделированием процесса сборки подарков «по одному», пока не кончатся упаковки по одной конфете. «Собирая» очередной подарок прежде всего постараемся потратить упаковки по три конфеты — их необходимо использовать $\lfloor \frac{n}{3} \rfloor$ для каждого подарка, но также нужно учитывать, что у нас не более b упаковок по три конфеты (значение b будем уменьшать по мере сборки подарков). Оставшиеся конфеты необходимо набрать упаковками по одной конфете. Будем продолжать этот процесс, пока число имеющихся упаковок по одной конфете (переменная a) не станет отрицательным — это означает, что последний подарок собрать не удалось. Такое решение не будет укладываться в ограничение по времени при больших входных числах. Пример такого решения на языке Python.

```
n = int(input())
a = int(input())
b = int(input())
ans = 0
while a >= 0:
    count3 = min(b, n // 3)
    count1 = n - 3 * count3
    a -= count1
    b -= count3
    ans += 1
print(ans - 1)
```

Чтобы набрать полный балл, посчитаем общее число конфет на складе, оно равно $s = a + 3b$. Количество изготовленных подарков не может быть больше $\lfloor \frac{s}{n} \rfloor$, иначе просто не хватит конфет, но также следует учесть, что если n не делится на 3, то на один подарок нужна обязательно одна или две упаковки по одной конфете (в зависимости от остатка от деления n на 3). То есть ответ не может превышать $\lfloor \frac{a}{n \bmod 3} \rfloor$. Ответом будет минимальное из этих двух чисел. Пример решения на 100 баллов на языке Python.

```
n = int(input())
a = int(input())
b = int(input())
s = a + 3 * b
ans = s // n
if n % 3 != 0:
    ans = min(ans, a // (n % 3))
print(ans)
```

Задача 2. Пробежка

Чтобы набрать 60 баллов, можно промоделировать процесс по дням. Пусть s — суммарный накопленный заряд наушников с первого дня, а *delta* — на сколько увеличивается заряд наушников каждый день. В самом начале $delta = a - b$, затем каждый день *delta* уменьшается на 1, а значение s увеличивается на *delta*. Рано или поздно *delta* станет отрицательным и значение s начнёт уменьшаться. Цикл закончится, когда s станет отрицательным, нужно вывести количество итераций цикла (количество прошедших дней), которое будем сохранять в переменной *day*. Пример решения на языке Python.

```
a = int(input())
b = int(input())
s = 0
delta = a - b
```

```
day = 0
while s >= 0:
    s += delta
    delta -= 1
    day += 1
print(day)
```

Чтобы решить задачу на полный балл, заметим, что за n дней суммарная величина заряда наушников равна an, а потрачено заряда было $b + (b + 1) + ... + (b + n - 1)$. Нам нужно найти такое минимальное n, что за n дней заряда было накоплено меньше, чем потрачено, то есть верно неравенство

$$an < b + (b + 1) + ... + (b + n - 1).$$

Правую часть преобразуем по формуле суммы арифметической прогрессии:

$$an < \frac{2b + n - 1}{2} n,$$

откуда $2a < 2b + n - 1$, $n > 2(a - b) + 1$. Минимальное n, удовлетворяющее этому неравенству, равно $2(a - b) + 2$.

Но если $a < b$, то эта формула даёт отрицательный результат, в этом случае программа должна вывести число 1 (уже на первой пробежке заряда наушников не хватит). Поэтому ответ на задачу равен $\max(1, 2(a - b) + 2)$.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
print(max(1, 2 * (a - b) + 2))
```

Получить эту формулу можно и из иного соображения «симметрии». Пусть $a > b$, тогда в первые $a - b$ дней заряд накапливается, причём в первый день добавляется $a - b$ минуты, потом $a - b - 1$ минута и т.д. В день номер $a - b$ добавится одна минута заряда. В следующий день величина дополнительного заряда равна величине пробежки (ничего не добавится), а следующие $a - b$ дней заряд будет тратиться — сначала 1 минута, потом 2 минуты и т.д. Таким образом, за $2(a - b) + 1$ дней накопленный заряд станет равен 0, а на следующий день заряда не хватит.

Задача 3. Лес

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

Чтобы набрать 32 балла, можно промоделировать весь путь Миши по одному шагу. Если координаты Миши (x, y), то он вышел из леса, когда $|x| = K$ или $|y| = K$.

Закодируем направления движения числами 0 (север), 1 (восток), 2 (юг), 3 (запад), тогда Миша меняет направления движения в порядке 0, 1, 2, 3, 0, 1, 2, 3 и изменение направления движения осуществляется инструкцией `dir = (dir + 1) % 4`. В массивах `DX` и `DY` хранятся координаты векторов перемещения на один шаг в данном направлении, то есть если направление движения есть `dir`, то по оси OX Миша сместится на значение `DX[dir]`, а по оси OY — на значение `DY[dir]`. Миша будет перемещаться, пока не дойдёт до границы леса, при этом общее число шагов подсчитывается в переменной `steps`, а число шагов, которое он совершил после последнего поворота, в переменной `last_segment`. Если значение `last_segment` стало равно числу шагов, которое Миша совершает в данном направлении (оно хранится в `count[dir]`), то нужно сделать поворот — изменяется значение `dir` и обнуляется значение `last_segment`. Пример такого решения.

```
count = [int(input()) for i in range(4)]
k = int(input())
DX = [0, 1, 0, -1]
DY = [1, 0, -1, 0]
```

```
x = 0
y = 0
steps = 0
last_segment = 0
dir = 0
while abs(x) != k and abs(y) != k:
    x += DX[dir]
    y += DY[dir]
    steps += 1
    last_segment += 1
    if last_segment == count[dir]:
        dir = (dir + 1) % 4
        last_segment = 0
print(steps)
```

Это решение можно улучшить, если моделировать перемещения не одиночными шагами, а сразу же на весь отрезок длины `count[dir]`. В этом случае к значению x будет добавляться `DX[dir] * count[dir]`, к значению y будет добавляться `DY[dir] * count[dir]`, и сразу после этого необходимо сделать поворот `dir = (dir + 1) % 4`. При таком моделировании, после очередного перемещения Миша может выйти за границы квадрата, поэтому цикл продолжается, пока обе координаты по модулю меньше k, а после окончания цикла нужно вычесть излишне пройденные вне квадрата шаги (то есть если Миша оказался в точке (x, y) и $|x| > k$, то нужно вычесть $|x| - k$, если же $|y| > k$, то нужно вычесть $|y| - k$ излишне пройденных шагов).

Пример решения на языке Python, такое решение набирает 64 балла.

```
count = [int(input()) for i in range(4)]
k = int(input())
DX = [0, 1, 0, -1]
DY = [1, 0, -1, 0]
x = 0
y = 0
steps = 0
dir = 0
while abs(x) < k and abs(y) < k:
    x += DX[dir] * count[dir]
    y += DY[dir] * count[dir]
    steps += count[dir]
    dir = (dir + 1) % 4

if abs(x) > k:
    steps -= abs(x) - k
if abs(y) > k:
    steps -= abs(y) - k
print(steps)
```

Для решения на полный балл нужно просто перебрать четыре направления, в которых Миша может выйти из леса, и для каждого из этих направлений посчитать, когда это произойдёт.

Пусть в результате перемещений на A шагов на север, B шагов на восток, C шагов на юг и D шагов на запад Миша оказался в точке (x, y), где $y = A - C$, $x = B - D$. Назовём это «циклом». Тогда после второго цикла Миша окажется в точке $(2x, 2y)$. после третьего цикла — в точке $(3x, 3y)$ и т.д. Заметим, что если $y > 0$, то Миша может выйти из леса в направлении на север, а если $y < 0$ — то в направлении на юг. Если $x > 0$, то Миша может выйти из леса в направлении на восток, а если $x < 0$ — то в направлении на запад.

Разберём случай $y > 0$, то есть когда Миша выйдет из леса в северном направлении. Посчитаем, сколько циклов пройдёт, прежде чем Миша выйдет из леса в этом направлении. На каждом

цикле Миша перемещается на север на y шагов, а затем он делает A шагов на север, прежде чем начнёт возвращаться на юг. При этом координата Миши должна оказаться не меньше чем K. Это напоминает известную задачу про улитку, которая днём поднимается по столбу на A метров, а вечером спускается на C метров и должна подняться на столб высоты K. Тогда Мише понадобится сделать $n = \lceil \frac{K-A}{A-C} \rceil$ полных циклов. А общее число шагов, которое сделает Миша, будет равно $n(A+B+C+D) + K - ny$ (за каждый из n циклов Миша проходит по $A+B+C+D$ шагов, после этого он оказывается в точке (nx, ny) и чтобы дойти до границы леса ему понадобится $K - ny$ шагов).

Рассмотрим случай $x > 0$. В этом случае Миша перемещается на восток на B шагов, потом возвращается на запад на D шагов. Количество полных циклов, которое необходимо ему сделать, прежде чем он выйдет на восток, равно $m = \lceil \frac{K-B}{B-D} \rceil$, а общее число шагов равно $m(A+B+C+D) + A + K - mx$.

Похожим образом надо рассмотреть возможность выхода из леса в южном и западном направлениях и взять минимальное количество шагов, необходимое для выхода во всех четырёх направлениях.

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 $y < 0$. Поэтому отдельно нужно рассмотреть возможность выхода из леса во всех четырёх направлениях на первом цикле.

Пример решения на языке Python.

```
import sys

count = [int(input()) for i in range(4)]
k = int(input())
DX = [0, 1, 0, -1]
DY = [1, 0, -1, 0]
x = 0
y = 0
steps = 0
for i in range(4):
    x += DX[i] * count[i]
    y += DY[i] * count[i]
    steps += count[i]
    if abs(x) >= k:
        print(steps - abs(x) + k)
        sys.exit(0)
    if abs(y) >= k:
        print(steps - abs(y) + k)
        sys.exit(0)
ans = 10 ** 19
if y > 0:
    p = (k - count[0] + y - 1) // y
    ans = min(ans, steps * p + k - p * y)
if x > 0:
    p = (k - count[1] + x - 1) // x
    ans = min(ans, steps * p + k - p * x + count[0])
if y < 0:
    p = (k - count[2] + count[0] - y - 1) // -y
    ans = min(ans, steps * p + k + p * y + count[0] * 2 + count[1])
if x < 0:
    p = (k - count[3] + count[1] - x - 1) // -x
    ans = min(ans, steps * p + k + p * x + count[0] + count[1] * 2 + count[2])
print(ans)
```

Задача 4. Сериал

В этой задаче в данном числовом массиве нужно найти три таких индекса $i < j < k$, что $a_i < a_j < a_k$.

Чтобы набрать 30 баллов можно написать перебор всех троек индексов $i < j < k$ тремя вложенными циклами и проверить для трёх элементов массива условие $a_i < a_j < a_k$. Если оно выполняется, то выведем ответ, если ни одной такой тройки не нашлось, то нужно вывести 0. Сложность такого решения составляет $O(n^3)$. Пример такого решения.

```
import sys
n = int(input())
a = [int(input()) for i in range(n)]
for i in range(n):
    for j in range(i + 1, n):
        for k in range(j + 1, n):
            if a[i] < a[j] < a[k]:
                print(i + 1, j + 1, k + 1)
                sys.exit(0)
print(0)
```

Улучшим это решение, заменив вложенные циклы на последовательные. Будем перебирать средний элемент (индекс j), а для фиксированного j найдём слева от него любой элемент a_i, который меньше a_j, а справа от j найдём любой элемент a_k, кототрый больше a_j. Это можно сделать двумя последовательными циклами, и сложность такого решения будет $O(n^2)$. Это решение набирает 60 баллов.

```
import sys
n = int(input())
a = [int(input()) for i in range(n)]
for j in range(1, n - 1):
    i = j - 1
    while i >= 0 and a[i] >= a[j]:
        i -= 1
    k = j + 1
    while k < n and a[j] >= a[k]:
        k += 1
    if i >= 0 and k < n:
        print(i + 1, j + 1, k + 1)
        sys.exit(0)
print(0)
```

Другой вариант этого решения, где в качестве a_i берётся минимальный элемент слева от a_j, а в качестве a_k — максимальный элемент справа от a_j. Для поиска минимального, максимального элементов и их индексов используются стандартные средства языка Python. Такое решение также имеет сложность $O(n^2)$ и набирает 60 баллов.

```
import sys
n = int(input())
a = [int(input()) for i in range(n)]
for j in range(1, n - 1):
    left_min = min(a[:j])
    right_max = max(a[j + 1:])
    if left_min < a[j] < right_max:
        print(a.index(left_min) + 1, j+1, j + a[j+1:].index(right_max) + 2)
```

```
        sys.exit(0)
print(0)
```

Это решение можно улучшить, если заметить, что перебирая центральный элемент нам необходимо найти минимальный элемент на некотором префиксе массива (начальной части) и максимальный элемент на некотором его суффиксе (конечной части). А минимальные и максимальные значения на всех префиксах и суффиксах можно один раз посчитать за $O(n)$ для всего массива. Ниже приведён пример такого решения, в котором в элементе массива `pref_min[i]` хранится индекс минимального элемента среди всех элементов, чьи индексы не превосходят `i`, а в массиве `suf_max[k]` хранится индекс максимального элемента среди всех элементов, чьи индексы не меньше `k`. Сначала двумя циклами заполняются эти массивы, затем циклом по переменной `j` перебирается центральный элемент, и теперь легко можно найти индекс наименьшего элемента перед ним (это `pref_min[j - 1]`) и максимального элемента после него (это `suf_max[j + 1]`). Такое решение имеет сложность $O(n)$ и набирает 100 баллов.

```
import sys
n = int(input())
a = [int(input()) for i in range(n)]
pref_min = [0] * n
suf_max = [0] * n
pref_min[0] = 0
for i in range(1, n):
    if a[i] < a[pref_min[i - 1]]:
        pref_min[i] = i
    else:
        pref_min[i] = pref_min[i - 1]
suf_max[n - 1] = n - 1
for k in range(n - 2, -1, -1):
    if a[k] > a[suf_max[k + 1]]:
        suf_max[k] = k
    else:
        suf_max[k] = suf_max[k + 1]
for j in range(1, n - 1):
    i = pref_min[j - 1]
    k = suf_max[j + 1]
    if a[i] < a[j] < a[k]:
        print(i + 1, j + 1, k + 1)
        sys.exit(0)
print(0)
```

Приведём пример ещё одного правильного решения сложности $O(n)$. Это решение работает за один проход и даже не сохраняет все элементы в массиве. Будем запоминать два значения: `i_val` — это наименьший встреченный элемент (рассматриваем его, как значение элемента массива, который будет ответом для переменной `i`) и `j_val` — минимальное значение элемента массива, слева от которого есть ещё один элемент, меньший него (это значение мы рассматриваем, как кандидата для второго элемента массива из ответа). Если новый просмотренный элемент оказывается больше, чем `j_val`, то ответ найден. Иначе обновим корректно значения `i_val` и `j_val`, если новое значение оказывается меньше них.

```
n = int(input())
i = 0
i_val = 10 ** 9
ans_i = 0
ans_j = 0
j_val = 10 ** 9
```

```
for k in range(1, n + 1):
    year = int(input())
    if year > j_val:
        print(ans_i, ans_j, k)
        break
    if i_val < year < j_val:
        j_val = year
        ans_i = i
        ans_j = k
    if year < i_val:
        i_val = year
        i = k
else:
    print(0)
```

Задача 5. Длинный плакат

30 баллов можно набрать, если перебрать все способы вычеркнуть из данного числа цифры. Пусть в числе s цифр. В группе тестов на 30 баллов $s \leqslant 9$, поэтому можно просто перебрать все подмножества цифр числа n (их будет 2^s штук, т.е. не больше 512) и взять только такие подмножества, в которых ровно $s-k$ элементов. Для таких подмножеств нужно составить соответствующую строку длины $s-k$ и выбрать из этих строк максимальную в лексикографическом порядке.

Такое решение имеет сложность $O(s2^s)$. Пример реализации такого решения на C++, для перебора подмножеств множества из s элементов используется перебор подмасок, удовлетворяющих неравеству $0 \leqslant mask < 2^s$ и битовые операции для обращения к битам числа $mask$.

```
#include<iostream>
using namespace std;
int main()
{
    string n, ans, tmp;
    int k;
    cin >> n >> k;
    int s = n.size();
    for (int mask = 0; mask < (1 << s); ++mask)
    {
        tmp = "";
        for (int i = 0; i < s; ++i)
            if ((mask >> i) & 1)
                tmp += s[i];
        if (tmp.size() == s - k && tmp > ans)
            ans = tmp;
    }
    cout << ans << endl;
}
```

Более эффективное решение просто строит ответ последовательно по одной цифре (слева направо), используя жадный алгоритм. Чтобы получившееся число было максимальным, 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.

Ниже приведён пример, в котором следующая цифра ищется вложенным циклом по переменной j. Такое решение имеет сложность $O(s^2)$ и набирает 60 баллов.

```
n = input()
k = int(input())
ans = ""
i = 0
while i + k < len(n):
    max_digit = n[i]
    max_digit_pos = i
    for j in range(i + 1, i + k + 1):
        if n[j] > max_digit:
            max_digit = n[j]
            max_digit_pos = j
    ans += max_digit
    k -= max_digit_pos - i
    i = max_digit_pos + 1
print(ans)
```

Чтобы улучшить это решение, нужна дополнительная идея — как быстро искать следующую цифру. Для этого для каждой из 10 возможных цифр `d` сохраним в списке `idx[d]` все позиции, на которых встречается эта цифра в исходном числе. Развернём все эти списки так, чтобы в начале шли максимальные (последние) индексы, а в конце — минимальные (первые). Это нужно для того, чтобы удалять из конца списков уже просмотренные позиции, потому что удаление элементов с конца списков работает быстро. Иными словами, списки `idx` являются стеками, на вершине которых хранятся минимальные индексы появления данной цифры в числе.

Следующая цифра, которую мы можем добавить к ответу, это максимальная цифра `d`, такая, что в конце списка `idx[d]` есть значение, индекс которого не меньше `i` и не больше `i + k`. Будем перебирать следующие цифры в цикле по переменной `d` от 9 до 0. Сначала выкинем из конца списка `idx[d]` все просмотренные цифры — это значения, которые меньше `i`. Если после этого в конце списка `idx[d]` оказался элемент, который меньше или равный $i + k$ (i — текущая позиция, но ещё не более k цифр можно вычеркнуть, поэтому нас интересуют цифры с индексами до $i + k$), то это означает, что мы можем вычернуть цифры вплоть до цифры `d`, и это и будет следующая цифра ответа. Её позиция будет последнему элементу в списке `idx[d]`. Добавим эту цифру в конец ответа, уменьшим значение k на количество вычернутых цифр и присвоим значение `i = idx[d][-1] + 1`, чтобы продолжить построение ответа со следующей за ней цифры.

Такое решение будет иметь сложность $O(s)$, или, если точнее, $O(s|A|)$, где A — множество цифр, которые могут использоваться в числе, то есть $|A| = 10$.

```
import sys

n = input()
k = int(input())

idx = [[] for i in range(10)]

for i in range(len(n)):
    idx[int(n[i])].append(i)

for i in range(10):
    idx[i] = idx[i][::-1]

ans = ""
```

```
i = 0
while i + k < len(n):
    for d in range(9, -1, -1):
        while idx[d] and idx[d][-1] < i:
            idx[d].pop()
        if idx[d] and idx[d][-1] <= i + k:
            ans += str(d)
            k -= idx[d][-1] - i
            i = idx[d][-1] + 1
            break
print(ans)
```

Не приводя подробную реализацию заметим, что у данной задачи есть и решение сложности $O(n)$, то есть без дополнительного множителя $|A|$. Например, можно использовать структуру данных «очередь с максимумом», позволяющую добавлять элементы в конец, удалять из начала и находить наибольший элемент в очереди, выполняя все операции за $O(1)$. Положим в очередь $K+1$ первых цифр числа. Каждую из этих цифр можно сделать первой цифрой ответа, поэтому из этих цифр можно взять максимальную (первую из максимальных, если таковых несколько). Узнаем значение этой цифры и удалим из очереди все элементы до этой цифры и эту цифру, затем добавим в очередь следующую цифру. Теперь любую цифру, находящиеся в очереди, можно поставить на следующее место в ответе, опять выберем максимальную цифру, удалим её и все предыдущие цифры, добавим в очередь ещё одну цифру и т.д.