Разбор задач

Задача 1. Разнобуквенные подстроки

Для начала оценим общее число разбиений. Заметим, что каждый символ строки (кроме первого) может как быть началом некоторой подстроки, так и не быть им (первый гарантированно является началом некоторой подстроки). Тогда общее количество вариантов будет равно 2^{n-1}, где n — длина строки. Для строки «*queue*» число возможных разбиений равно 16. В принципе, можно перебрать все разбиения и оставить только корректные.

Чтобы упростить процесс перебора, можно заметить, что подстроки, начинающиеся в позициях 1 или 2 (в единичной индексации), могут заканчиваться не далее позиции 3 (иначе будет повторение буквы «q»). Подстроки, начинающиеся в позиции 3, могут заканчиваться не далее позиции 4 (иначе будет повторение буквы «e»).

Итоговый набор корректных разбиений является таким:

q u e u e
qu e u e
q ue u e
q u eu e
q u e ue
qu eu e
qu e ue
q ue ue
que u e
que ue

Задача 2. Москва-2050

Все крайние сегменты (как у цифры 0) уже проверены, потому что первоначально отображается число 00. Рассмотрим средние сегменты. Минимальная цифра, при которой отображается средний горизонтальный сегмент — это цифра 3, она же проверяет и все остальные внутренние сегменты. Поэтому нужно постараться получить цифру 3 в старшем разряде, используя минимальное число нажатий. Для этого получим число 01, затем будем умножать на 2: 02, 04, 08, 16, 32, что соответствует последовательности нажатий «1 2 2 2 2 2». Несложно убедиться, что при этом мы проверим и все внутренние сегменты во втором разряде, например, цифрами 2 и 6.

Быстрее получить цифру 3 в старшем разряде нельзя, поэтому не получится быстрее проверить средний горизонтальный сегмент в первом разряде, значит, это наилучшее решение.

Задача 3. Последовательность

Ответы.

1) 32

2) 101

3) 75

4) 800

Решение.

1) Достаточно промоделировать построение такой последовательности, начинающейся с числа 1:
$1 \to 2 \to 4 \to 8 \to 16 \to 22 \to 24 \to 28 \to 36 \to \ldots$

Видно, что число 32 не входит в эту последовательность.

Ответ 32.

2) Для решения этой задачи построим начальные части двух последовательностей.

1 → 2 → 4 → 8 → 16 → 22 → 24 → 28 → 36 → 42 → 46 → 52 → 57 → 64 → 70 → 77 → 84 → 92 ↘
101
3 → 6 → 12 → 14 → 18 → 26 → 32 → 35 → 40 → 44 → 48 → 56 → 62 → 68 → 76 → 83 → 91 → 100 ↗

Ответ 101.

3) Применим обратный анализ: как можно получить число n? При увеличении числа на наибольшую цифру оно увеличивается не менее, чем на 1, и не более, чем на 9. Поэтому число n получится только из чисел в диапазоне $[n-9, n-1]$. Таких чисел будет не более двух, причём они обязательно должны принадлежать различным десяткам (например, 56 дают числа 48 и 51).

У числа 99 наибольшая цифра принадлежит как разряду десятков, так и разряду единиц, поэтому оно является производным от $99 - 9 = 90$.

Число 90 может получиться только из числа, первая цифра которого — 8. Очевидно, что это число 82 (других вариантов нет, поскольку все числа, подходящие под условие, принадлежат одному и тому же десятку).

Число 82 теоретически могут дать числа из диапазона $[80, 81]$ (но это невозможно — первая цифра 8 слишком большая) или из диапазона $[73, 79]$ (здесь подходящее число найти можно, это 75).

Число 75 теоретически можно получить из числа в диапазоне $[70, 74]$ (но это невозможно — первая цифра 7 слишком большая) или в диапазоне $[64, 69]$ (перебрав числа, убеждаемся, что и здесь подходящее число отсутствует).

Ответ 75.

4) Рассмотрев последовательность чисел, начинающуюся с 1, почти сразу найдём 11 подряд идущих чётных чисел:

$1 \to 2 \to 4 \to 8 \to 16 \to 22 \to 24 \to 28 \to 36 \to 42 \to 46 \to 52 \to 57 \to \ldots$

Но есть и более длинные последовательности чётных чисел. Очевидно, что числа в последовательности будут принадлежать всем десяткам без пропусков (поскольку увеличение происходит не более, чем на 9), а чётное число может произвести следующее чётное число в последовательности, только если его наибольшая цифра — чётная. При этом рано или поздно мы получим у чётного числа в разряде десятков цифру 9 и следующее число в последовательности неизбежно будет нечётным. Значит, нам нужно выбрать такую чётную цифру, чтобы она как можно дольше оставалась наибольшей цифрой для наших чисел (очевидно, что это цифра 8) и при этом стартовое число как можно позже переходило на девятый десяток, при этом желательно, чтобы в восьмом десятке встретилось два чётных числа: $*80 \to *88$.

В качестве наименьшего подходящего числа можно взять 800 (его даёт нечётное число 791, поэтому это будет первое чётное число подпоследовательности): $800 \to 808 \to 816 \to 824 \to 832 \to 840 \to 848 \to 856 \to 864 \to 872 \to 880 \to 888 \to 896$. В числе десятков – нечётная цифра 9, поэтому следующее число будет нечётным. Всего 13 чётных чисел и найти более длинную последовательность чётных нельзя.

Ответ 800.

Задача 4. Путешествие с чемоданом

Сначала убедимся в том, что хотя бы один из друзей может унести другого. Если это неверно, то $a > d$ и $c > b$. В этом случае нужно вывести -1.

Иначе рассмотрим два случая. Если Гена несёт Чебурашку с чемоданом, то они смогут унести $\min(d - a, b, z)$, а если Чебурашка несёт Гену с чемоданом, то они смогут унести $\min(b - c, d, z)$. Из этих двух минимумов следует выбрать наибольший.

Пример решения на языке Python.

```
a = int(input())
b = int(input())
c = int(input())
d = int(input())
z = int(input())
if a > d and c > b:
    print(-1)
else:
    print(max(min(d - a, b, z), min(b - c, d, z)))
```

Задача 5. Светофоры

50 баллов можно набрать, если моделировать процесс, для каждого перехода определяя время его начала и окончания, для чего сделаем цикл длины n.

Период работы светофора имеет продолжительность $a + b$ секунд. Переходить можно, если после начала периода прошло менее a секунд, то есть для определения того, какой сигнал горит на светофоре в момент времени tm, нужно сравнить остаток от деления tm на $a + b$ со значением a. Если сейчас горит (или загорается) красный, то нужно подождать начала нового периода. Для этого посчитаем число полных прошедших периодов, используя целочисленное деление на $a + b$, добавим один дополнительный период и умножим на длину периода $a + b$.

После этого прибавим к текущему времени tm продолжительность одного перехода t.

Пример такого решения.

```
n = int(input())
a = int(input())
b = int(input())
t = int(input())
tm = 0
period = a + b
for i in range(n):
    if tm % period >= a:
        tm = (tm // period + 1) * period
    tm += t
print(tm)
```

Это решение работает медленно при больших n. Чтобы набрать 100 баллов определим количество переходов, которое совершит Петя за один период. Здесь важно то, что $t < b$, то есть если Петя начал переход, и загорелся красный свет, то мальчик закончит переход на красный, то есть новый период не начнётся. Это значение равно k = (a − 1) // t + 1. (Для последнего перехода в периоде нужно «зарезервировать» минимум 1 секунду от периода горения зелёного сигнала. Тогда за $a - 1$ секунд Петя успеет начать и завершить на зелёный(a − 1) // t переходов. Добавим тот последний переход, который Петя начнёт и, возможно, завершит на красный)

Посчитаем количество полных «блоков» из k переходов и количество оставшихся переходов, которые Петя совершит за последний (неполный) период, взяв целочисленное частное и остаток от деления n на k. Один полный блок Петя переходит за $a + b$ секунд (период работы светофора), а каждый дополнительный переход занимает t секунд.

Но есть один крайний случай — если дополнительных переходов нет, то есть n делится на k нацело, то переход последнего блока займёт не $a + b$, а kt секунд, так как Пете не нужно будет ожидать зелёного сигнала после завершения всех переходов. Этот случай проще всего учесть, считая, что если n делится на k, то полных блоков на 1 меньше, а k переходов последнего блока нужно считать как неполный блок из k переходов, для чего проверим отдельное условие.

Пример такого решения.

```
n = int(input())
a = int(input())
b = int(input())
t = int(input())
k = (a - 1) // t + 1
blocks = n // k
extra = n % k
if extra == 0:
    blocks -= 1
    extra = k
print(blocks * (a + b) + extra * t)
```

Задача 6. Путь зигзагом

Будем увеличивать координаты чередуя их, пока не придём в нужную точку ($x = a$, $y = b$). Если же одна координата уже приняла необходимое значение, а её нужно изменить, уменьшим её на 1, потом она снова увеличится на 1, то есть эта координата будет меняться вблизи нужного нам значения: a, $a - 1$, a, $a - 1$ и т.д., пока вторая координата не достигнет конечного значения.

В этом случае, возможно, мы сделаем лишний ход. Чтобы избежать этого лишнего хода, первое движение нужно делать в том направлении, в котором нам нужно переместиться на большее расстояние, то есть если $a > b$, то на первом шаге нужно менять x, а если $b > a$ — то y. Тогда мы получим наилучший ответ.

Докажем это. Пусть $m = \max(a, b)$. Тогда мы должны сделать как минимум m шагов в одном направлении, и ответ не может быть меньше $m + (m - 1)$, т.к. в другом направлении мы должны сделать как минимум $m - 1$ шагов. Более того, если a и b — одной чётности, то количества выполненных шагов в каждом направлении также должны быть одной чётности, и общее число шагов будет не менее $2m$. То есть общее число шагов не меньше $2m$, если a и b одной чётности, и $2m - 1$, если разной чётности.

Пусть $a > b$. Тогда выполнив a шагов по координате x мы окажемся либо в точке (a, b), либо в точке $(a, b-1)$, так как по координате y робот будет «бегать» между $y = b$ и $y = b-1$. Поскольку мы начали движение с координаты x, по оси y мы сделали на одно перемещение меньше, и координаты робота будут разной чётности. В какой именно точке он окажется, зависит именно от чётности b, поэтому в случае разных чётностей a и b он окажется в точке (a, b), и алгоритм завершит работу. Если же a и b одной чётности, то робот окажется в точке $(a, b - 1)$, и ему понадобится сделать ещё один шаг. Случай $a < b$ рассматривается аналогично, в случае $a = b$ наше решение достигнет цели ровно за $a + b$ шагов.

Пример решения. В этом решении в переменной move_x хранится логическое значение (True или False), означающее, что робот делает очередной шаг вдоль оси OX. Это значение будет меняться на противоположное на каждом шаге цикла.

```
a = int(input())
b = int(input())
move_x = a > b
x = 0
y = 0
while x != a or y != b:
    if move_x:
        if x < a:
            x += 1
        else:
            x -= 1
    else:
        if y < b:
            y += 1
        else:
            y -= 1
    print(x, y)
    move_x = not move_x
```