Задача 3. «Поврежденный XML»

Поскольку ограничения на размер входной строки в этой задаче небольшие, ее можно решать полным перебором. Будем пытаться исправлять каждый символ по порядку на все допустимые (латинские буквы, символы '<', '>', '/'), до тех пор пока не получится корректная xml-строка.

Приведем пример кода на языке python3:

```
s = input()
alph = 'qwertyuiopasdfghjklzxcvbnm<>/'
for i in range(n):
    for c in alph:
        test = s[:i] + c + s[i + 1:]
        if correct_xml(test):
            print(test)
            exit()
```

Осталась чисто техническая часть решения: написать функцию correct_xml, которая проверяет, является ли данная строка корректным xml-фрагментом. Разобьем эту задачу на несколько шагов.

1) Убедимся, что строка начинается с символа '<', а заканчивается символом '>' и удалим начальный и конечный символы.
2) Разрежем строку по парам символов '><' (удаляя при этом эти символы). Получим список строк-тегов (открывающих или закрывающих).
3) Проверим, что в этих тегах нет символов '<' или '>', а также нет символов '/' не на первой позиции.
4) Проверим, что эти теги образуют "правильную скобочную последовательность".

Опишем подробнее выполнение последнего шага. Заведем стек тегов (строк), изначально пустой. Будем идти по тегам слева направо. Рассмотрим два случая:

I. *Очередной тег – открывающий*. В этом случае добавим новый тег в конец стека.

II. *Очередной тег* – закрывающий.

   а) Стек пуст. В этом случае мы нашли закрывающий тег, которому не соответствует открывающий тег, следовательно, строка не является корректной xml-строкой.

   б) Стек не пуст. Извлечем из стека последний элемент. Если этот тег не соответствует нашему закрывающему тегу, то xml-строка некорректная.

Если мы прошли весь список тегов, и ни на каком шаге не выяснили, что строка некорректна, нам осталось лишь проверить, что в конце стек оказался пуст (то есть что у нас нет "лишних" открывающих тегов).

Приведем пример реализации такой функции на языке python3:

```
def check(test):
    if test[0] != '<' or test[-1] != '>':
        return False
    testlist = test[1:-1].split('><')
    for s in testlist:
        if '<' in s or '>' in s or '/' in s[1:]:
            return False
```

```
    q = deque()
    for s in testlist:
        if s[0] != '/':
            q.append(s)
        elif not q or q.pop() != s[1:]:
            return False
    if q:
        return False
    return True
```

Задача 4. «Игра с числами»

Обозначим через *first* число, выбранное на первом ходу. Тогда каждое очередное число нужно выбирать так, чтобы НОД разностей всех выбранных чисел с первым числом был строго больше 1. Заметим, что если d – это НОД разностей выбранных чисел с первым числом, то оставшиеся числа делятся на две группы: числа, при выборе которых на последующих ходах НОД не изменится, и числа, которые при их выборе уменьшат НОД. При этом с одной стороны *для дальнейшей игры нам не важно, какие именно числа находятся в первой группе* (а важно лишь их количество), а с другой стороны, определить, принадлежит ли некоторое число a второй группе, можно, проверив, делится ли $|a - first|$ на d. Таким образом, позицию в игре (начавшейся выбором числа first) можно задавать двумя параметрами: количеством сделанных ходов *num* и НОД d разностей всех выбранных чисел с первым числом.

Для каждого возможного первого хода будем решать задачу отдельно. Поскольку количество теоретически возможных значений НОД может быть довольно велико, будем решать задачу рекурсией с запоминанием ("ленивым" динамическим программированием).

Рассмотрим позицию (*num*, *d*).

1) Если *num* = *n*, то позиция проигрышная (все числа уже выбраны; ход сделать нельзя).
2) Подсчитаем, количество чисел *count* в изначальном наборе чисел, разности которых с числом *first* делятся на *d*:

   а) Рассмотрим сначала числа, выбор которых уменьшает *d*, то есть такие числа *x*, для которого $|x - first|$ не делится на *d*, и при этом НОД(d, $|x - first|$) должен быть больше 1 (в противном случае выбрать число *x* уже нельзя). Позиция (*num*, *d*) выигрышная, если (*num* + 1, НОД(d, $|x - first|$) – проигрышная.

   б) если *count* > *num,* то мы можем выбрать число, делящееся на *d*, получив при этом позицию (*num* + 1, *d*). Если она проигрышная, то (*num*, *d*) – выигрышная.
3) В противном случае позиция (*num*, *d*) – проигрышная.

Осталось отметить один технический момент: если *num* = 1, то выбрано пока только число *first*. В этом случае все рассуждения останутся верными, если положить $d = 0$.

Приведем программу на языке python3:

```
#python3
from fractions import gcd
fin = open("game.in")
fout = open("game.out", "w")
n = int(fin.readline()
a = list(map(int, fin.readline().split()))
def solve(num, d):
    if (num, d) in res:
```