Хеширование и хеш-таблицы

1 A. Кубики

Разбор

Сначала разберёмся с тем, что от нас хотят в этой задаче – придумаем наивное и не очень эффективное решение. Привидение Петя стоит перед зеркалом и **не знает**, сколько кубиков перед ним. Тогда мы можем перебрать разные позиции зеркала и понять, возможна ли каждая из позиций. Пусть перед Петей лежат k настоящих кубиков. Тогда следующие k кубиков $k+1, k+2, ..., 2k$ должны являться отражением того, что видит перед собой Петя. Все остальные кубики нас **не интересуют**, т.к. Петя их не видит и может лишь догадываться о том, что у него за спиной. Т.е. нам нужно проверить, что подстрока $s_{0..2k}$ является палиндромом.

Как это быстро проверить с помощью хешей? Посчитаем хеши префиксов в прямом и обратном порядке. Затем найдём прямой хеш подстроки $s_{0..k}$ и обратный хеш подстроки $s_{k+1..2k}$, эти хеши должны быть равны.

Асимптотика решения: $O(N)$

Решение

```
n, m = map(int, input().split())
a = list(map(int, input().split()))
a_inv = list(reversed(a))

p, mod = m, 1000000007

h, h_inv, pws = [0] * (n + 1), [0] * (n + 1), [1] * (n + 1)
for i in range(n):
    h[i + 1] = (h[i] * p + a[i]) % mod
    h_inv[i + 1] = (h_inv[i] * p + a_inv[i]) % mod
    pws[i + 1] = (pws[i] * p) % mod

ans = []
for i in range(n // 2 + 1):
    re_hash = h[i]
    # n - i - (n - 2 * i) =>  n - i - n + 2 * i => 2i - i => i
    im_hash = (h_inv[n - i] - h_inv[n - 2 * i] * pws[i]) % mod
    if re_hash == im_hash:
        ans.append(n - i)

print(*reversed(ans))
```

2 B. Период строки

Разбор

Чтобы проверить периодичность строки, нам нужно перебрать возможные длины периодов и для каждого периода проверить, что значение всех периодических подстрок совпадает. Будем искать ответ прямым перебором: если длина строки кратна l и для данной длины периода l выполняется периодичность, то мы нашли ответ и дальше перебирать нет смысла.

Как быстро проверить периодичность? Давайте $\frac{n}{l}$ раз проверим, что хеши $h(s_{0..l-1})$ и $h(s_{i..i+l-1})$ совпадают для всех $i = l, 2l, 3l, ...$

Асимптотика решения: $O(N \times \sqrt{N})$

Решение

```
p, mod = 29, 1000000007

def ctoi(c):
    return ord(c) - ord('a') + 1

s = input()
n = len(s)

def check_period(h, i):
    h_base = (h[i] - h[0] * pws[i]) % mod
    k = 2 * i
    while k <= n:
        h_sub = (h[k] - h[k - i] * pws[i]) % mod
        if h_sub != h_base:
            return False
        k += i
    return True

h, pws = [0] * (n + 1), [1] * (n + 1)
for i in range(n):
    h[i + 1] = (h[i] * p + ctoi(s[i])) % mod
    pws[i + 1] = (pws[i] * p) % mod

ans = 1
for i in range(1, n // 2 + 1):
    if n % i == 0 and check_period(h, i):
        ans = n // i
        break
print(ans)
```

3 C. Палиндромные заклинания

Разбор

В этой задаче нам нужно жадно набрать как можно больше совпадающих подстрок $s_{i..i+k}$ и $s_{n-i-k..n-i}$. Для этого пойдём с двух концов и будем смотреть на подстроки. Если они не совпали - увеличим k. Иначе увеличим ответ на единицу и сделаем $k := 1$. Важно не запутаться в индексации.

Асимптотика решения: $O(T \times N)$

Решение

```
p, mod = 29, 1000000007

def ctoi(c):
    return ord(c) - ord('a') + 1

t = int(input())

for _ in range(t):
    s = input()
    n = len(s)
    h, pws = [0] * (n + 1), [1] * (n + 1)
    for i in range(n):
        h[i + 1] = (h[i] * p + ctoi(s[i])) % mod
        pws[i + 1] = (pws[i] * p) % mod
    l, r, shift = 0, n, 1
    ans = 0
    while l + shift <= r - shift:
        h_l = (h[l + shift] - h[l] * pws[shift]) % mod
        h_r = (h[r] - h[r - shift] * pws[shift]) % mod
        if h_r == h_l:
            l += shift
            r -= shift
```

```
            shift = 1
            ans += 2
        else:
            shift += 1
    if l < r:
        ans += 1
    print(ans)
```

4 D. Массивы

Разбор

Соберём в хеш-таблицу количество вхождений каждого из элементов массива a. Затем, идя по массиву b, будем выводить значение из хеш-таблицы либо 0.

Асимптотика решения: $O(N + M)$

Решение

```
_ = input()
a = list(map(int, input().split()))
cnt = {}
for v in a:
    if not v in cnt:
        cnt[v] = 0
    cnt[v] += 1

_ = input()
ans = []
b = list(map(int, input().split()))
for v in b:
    ans.append(cnt.get(v, 0))
print(*ans)
```

5 E. С днём рождения!

Разбор

Зафиксируем длину строки $l = \log p$ и число итераций $k = \sqrt{q}$. Затем k раз сгенерируем случайную строку s и запомним соответствие хеша и строки. Если ранее мы уже добавили в список другую строку с таким же хешем, то ответ найден. Иначе запустим функцию ещё раз.

Асимптотика решения: $O(\log p \times \sqrt{q})$

Решение

```
from math import sqrt, log
from random import choice

ALPHABET = 'abcdefghijklmnopqrstuvwxyz'

def sample(p, q):
    l = int(log(p) + 6)
    s, h = '', 0
    for _ in range(l):
        s += choice(ALPHABET)
        h = (h * p + ord(s[-1])) % q
    return s, h

def solve(p, q):
    k = int(sqrt(q) + 6)
    collisions = {}
    for _ in range(k):
```

```
        s, h = sample(p, q)
        if h in collisions and s != collisions[h]:
            return collisions[h], s
        collisions[h] = s
    return solve(p, q)

p, q = map(int, input().split())
a, b = solve(p, q)
print(a)
print(b)
```

6 F. Массивы-палиндромы

Разбор

Для начала научимся для заданной длины подотрезков проверять, что их сумма образует палиндром. Возьмём в качестве примера две последовательности a, b, c, d и x, y, z, w. Пусть мы хотим, чтобы $a + x = d + w$, $b + y = c + z$. Тогда мы, очевидно, можем переставить слагаемые так, чтобы получилось $a - d = w - x$ и $b - c = z - y$. Т.е. мы хотим, чтобы разности противоположных относительно центра отрезка элементов были равны.

Если бы мы каждый раз пересчитывали разности с нуля, то получили бы квадратичную сложность. Значит, нам надо придумать, как быстро считать разность отрезка. Давайте будем считать не саму разность, а её хеш! Для этого возьмём прямой хеш левой половинки отрезка $a_1 p^{k-1} + a_2 p^{k-2} + ... + a_k$ и обратный хеш правой половинки отрезка $a_{k+1} + a_{k+2} p + ... + a_{2k} p^{k-1}$ и посчитаем их разность. Если в обоих массивах для каких-то отрезков мы получили одинаковые хеши разностей, то это гарантирует нам, что сумма этих отрезков образует палиндром.

Т.к. палиндромы симметричны относительно центра, просто переберём допустимую длину палиндрома бинпоиском.

Асимптотика решения: $O(N \times \log N)$

Решение

```
p, mod = 101, 1000000007

n, m = map(int, input().split())
a = list(map(int, input().split()))
b = list(map(int, input().split()))
a_inv, b_inv = list(reversed(a)), list(reversed(b))

v, w = min(n, m), max(n, m)
h_a, h_a_inv = [0] * (n + 1), [0] * (n + 1)
h_b, h_b_inv = [0] * (m + 1), [0] * (m + 1)
pws = [1] * (w + 1)

for i in range(n):
    h_a[i + 1] = (h_a[i] * p + a[i]) % mod
    h_a_inv[i + 1] = (h_a_inv[i] * p + a_inv[i]) % mod
for i in range(m):
    h_b[i + 1] = (h_b[i] * p + b[i]) % mod
    h_b_inv[i + 1] = (h_b_inv[i] * p + b_inv[i]) % mod
for i in range(w):
    pws[i + 1] = (pws[i] * p) % mod

def check(l, odd):
    shift = int(odd)
    hashes = set()
    for i in range(2 * l + shift, n + 1):
        left_start = i - 2 * l - shift
        left_end = i - l - shift
        right_start = i - l
        right_end = i

        h_1 = (h_a[left_end] - h_a[left_start] * pws[l]) % mod
```

```
            h_2 = (h_a_inv[n - right_start] - h_a_inv[n - right_end] * pws[l]) % mod
            hashes.add(h_1 - h_2)
        for i in range(2 * l + shift, m + 1):
            left_start = i - 2 * l - shift
            left_end = i - l - shift
            right_start = i - l
            right_end = i

            h_1 = (h_b[left_end] - h_b[left_start] * pws[l]) % mod
            h_2 = (h_b_inv[m - right_start] - h_b_inv[m - right_end] * pws[l]) % mod
            if (h_2 - h_1) in hashes:
                return True
        return False

ans = 0
# odd case
l, r = -1, (v + 1) // 2 + 1
while r - l > 1:
    mid = (r + l) // 2
    if check(mid, True):
        l = mid
    else:
        r = mid
ans = max(ans, 2 * l + 1)
# even case
l, r = -1, (v + 1) // 2 + 1
while r - l > 1:
    mid = (r + l) // 2
    if check(mid, False):
        l = mid
    else:
        r = mid
ans = max(ans, 2 * l)
print(ans)
```