Задача A. Сравнения подстрок

Считаем обычный префиксный полиномиальный хэш с лекции для строки s. На каждом запросе вычисляем за $O(1)$ хэш подстроки $s[a \dots b]$, подстроки $s[c \dots d]$ и сравниваем.

Решение работает за $O(|s| + m)$.

```
s = input()
mod = 10**9 + 7
t = 31
len_s = len(s)
h = [0] * (len_s + 1)
p = [1] * (len_s + 1)
for i in range(len_s):
    h[i + 1] = (h[i] * t + (ord(s[i]) - ord('a') + 1)) % mod

for i in range(len_s):
    p[i + 1] = (p[i] * t) % mod

n = int(input())
for _ in range(n):
    l, r, l1, r1 = map(int, input().split())
    str_1 = (h[r] - h[l - 1] * p[r - l + 1]) % mod
    str_2 = (h[r1] - h[l1 - 1] * p[r1 - l1 + 1]) % mod
    if str_1 == str_2:
        print('Yes')
    else:
        print("No")
```

Задача B. Поиск подстроки

Решаем аналогично задаче A. Насчитываем префиксный полиномиальный хэш строки S и полиномиальный хэш строки T. Теперь осталось сравнить хэши всех подстрок строки S с хэшом строки T.

Решение работает за $O(|S| + |T|)$.

```
S = input()
T = input()
mod = 10 ** 18
k = 29
n = len(S)
q = len(T)
prefixHash = [0] * q
prefixHash[0] = (ord(T[0]))
for i in range(1, q):
    ch = ord(T[i])
    prefixHash[i] = (prefixHash[i - 1] * k + ch) % mod;
hashT = prefixHash[-1]
prefixHash[0] = (ord(S[0]))
for i in range(1, q):
    ch = ord(S[i])
    prefixHash[i] = (prefixHash[i - 1] * k + ch) % mod;
hashPrefixS = prefixHash[-1]
N = k ** q % mod
result = []
if hashT == hashPrefixS:
    result.append(0)
for i in range(1, n-q+1):
```

```
    chNew = ord(S[i-1+q])
    chOld = ord(S[i-1])
    hashPrefixS = (hashPrefixS * k + chNew) % mod
    hashPrefixS -= (N * chOld) % mod
    #print(hashPrefixS)
    if hashPrefixS == hashT:
        result.append(i)
print(*result)
```

Задача C. Неточное совпадение

Предпосчитаем префиксный полиномиальный хэш для строк p и t. Переберём начало вхождения p в t. Бинпоиском найдём длину максимального префикса шаблона, затем проверим, что суффикс с позиции $i+1$ (пропустили несовпадающий символ) совпадает с соответствующим суффиксом шаблона.

Решение работает за $O(|t|\log|t| + |s|)$.

```
ans = []
for i in range(len(s) - len_m + 1):
    l = 0
    r = len_m - 1
    while l < r:
        m = (l + r) // 2
        if find_hs(i + 1, i + m + 1, hs) == hs2[m]:
            l = m + 1
        else:
            r = m
    if l == len_m or l == (len_m - 1):
        ans.append(i + 1)
        continue
    elif hs2_rev[l + 1] == find_hs(i + l + 2, i + len_m, hs):
        ans.append(i + 1)
```

Задача D. Подпалиндромы

У каждой подстроки палиндрома есть центр. Относительно этого центра длина половинок, которые образуют палиндром монотонна. То есть можно фиксировать центр палиндрома, делать бинпоиск по длине половинки и проверять хэшами. К ответу надо прибавить $k+1$, если максимальный палиндром с фиксированным центром имеет длину $2k+1$ и k в противном случае.

Решение работает за $O(n\log n)$. Существуют альтернативное решение, рассматривающее только палиндромы нечетной длины и альтернативное решение с алгоритмом Манакера.

```
res = 0
for i in range(len_s):
    l = 0
    r = min(i, len_s - i - 1)

    while l < r:
        m = (l + r) // 2
        if find_hs2(len_s - i, len_s - i + m) == find_hs1(i + 1, i + m + 1):
            l = m + 1
        else:
            r = m

    if find_hs2(len_s - i, len_s - i + l) == find_hs1(i + 1, i + l + 1):
        res += 1
    res += l
```

```
for i in range(len_s - 1):
    if s[i] != s[i + 1]:
        continue
    l = 0
    r = min(i, len_s - i - 2)

    while l < r:
        m = (l + r) // 2
        if find_hs2(len_s - i, len_s - i + m) == find_hs1(i + 2, i + 2 + m):
            l = m + 1
        else:
            r = m

    if find_hs2(len_s - i, len_s - i + l) == find_hs1(i + 2, i + 2 + l):
        res += 1
    res += l
```

Задача E. Анаграммы-2

Давайте установим биекцию $\phi(x) = rand()$, чтобы различные числа переходили в различные, а одинаковые в одинаковые. Тогда на роль хэш функции множества подойдет сумма $\phi(x)$.

Давайте предпосчитаем хэш от каждого подотрезка второго массива, закинем их в словарь по длине как отсортированные последовательности (это делать необязательно, но так мы уменьшим вероятность коллизии).

Теперь будем фиксировать подотрезок первого массива и искать бинпоиском в предпосчитанной таблице такой же хэш по индексу такой же длины.

Решение работает за $O(n^2 \log n)$.

```
ans = 0
for i in range(0,n+1):
    for j in range(i,n+1):
        t.add(pref_1[i] - pref_1[j])
for i in range(0,n_2+1):
    for j in range(i,n_2+1):
        k = pref_2[i] - pref_2[j]
        if k in t:
            #print(j-i)
            ans = max(j-i,ans)
print(ans)
```