Университет ИТМО

Факультет инфокоммуникационных технологий
1 курс
Лабораторная работа №4

Выполнила:
Чагина Вероника Александровна

группа К3144

Преподаватель:

Харьковская Татьяна Александровна

Дата выполнения: 22.06.2022

Санкт-Петербург

Основная часть

4 Задача. Равенство подстрок

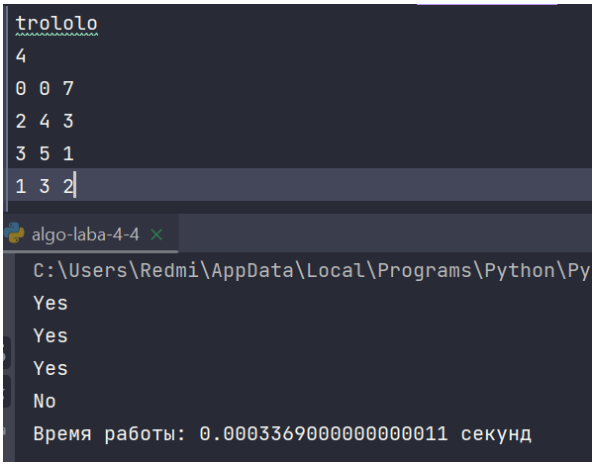
В этой задаче вы будете использовать хеширование для разработки алгоритма, способного предварительно обработать заданную строку s, чтобы ответить эффективно на любой запрос типа «равны ли эти две подстроки s?» Это, в свою очередь, является основной частью во многих алгоритмах обработки строк.

Решение:

```
def get_hashes(ss):
    n = len(ss)
    hsh = [0] * (n + 1)
    for i in range(n):
        hsh[i + 1] = (hsh[i] * x % M + ord(ss[i]) - ord('a')) % M
    return hsh


def get_pows(n):
    pw = [1] * (n + 1)
    for i in range(1, n + 1):
        pw[i] = pw[i - 1] * x % M
    return pw


def get(l, r):
    return (hsh[r] - hsh[l] * pw[r - l] % M) % M


s = input()
hsh = get_hashes(s)
pw = get_pows(len(s))
q = int(input())
for _ in range(q):
    l1, l2, sz = map(int, input().split())
    if get(l1, l1 + sz) == get(l2, l2 + sz):
        print('Yes')
    else:
        print('No')
```

Подсчитываем хэш-значения всех префиксов. С помощью этих хеш-значений вычисляем хеш-значения строки. В конце сравниваем эти значения - если равны, то выводим «Yes», иначе «No».

Тесты:



5 Задача. Префикс-функция

Постройте префикс-функцию для всех непустых префиксов заданной строки s.

Решение:

```
def prefix_function(text):
    p = [0 for a in range(len(text) - 1)]
    i, j = 1, 0
    while i < len(text)-1:
        if text[i] == text[j]:
            p[i] = j + 1
            i += 1
            j += 1
        elif j > 0:
            j = p[j-1]
        else:
            p[i] = 0
            i += 1
    return p
```

Проходясь по строке мы смотрим, может ли найденный суффикс быть расширен на следующую позицию - если нет, то уменьшаем суффикс. Если же элементы совпадают, то увеличиваем счётчик длины и добавляем итог в ответ.

Тесты:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.968	136368128	1000002	6888889
1	OK	0.031	8925184	8	11
2	OK	0.015	8998912	9	13
42	OK	0.859	136302592	1000002	6888884
43	OK	0.750	136327168	1000002	6888869
44	OK	0.703	133046272	1000002	6388894
45	OK	0.750	136278016	1000002	6883894

7 Задача. Наибольшая общая подстрока

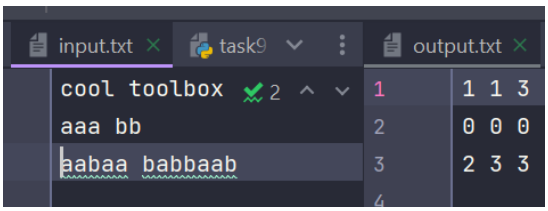
Найти строку w максимальной длины, которая является подстрокой как s, так и t, данных нам по условию.

Решение:

```
def get_hashes(ss, M):
    n = len(ss)
    hsh = [0] * (n + 1)
    for i in range(n):
        hsh[i + 1] = (hsh[i] * x % M + ord(ss[i]) - ord('a')) % M
    return hsh


def get_pows(n, M):
    pw = [1] * (n + 1)
    for i in range(1, n + 1):
        pw[i] = pw[i - 1] * x % M
    return pw


def check(k):
    if k == 0:
        return (0, 0)
    idx = dict()
    for i in range(n + 1 - k):
        h1 = (s1[i + k] - s1[i] * pow1[k]) % M1
        h2 = (s2[i + k] - s2[i] * pow2[k]) % M2
        idx[(h1, h2)] = i
    for i in range(m + 1 - k):
        h1 = (t1[i + k] - t1[i] * pow1[k]) % M1
        h2 = (t2[i + k] - t2[i] * pow2[k]) % M2
        if (h1, h2) in idx:
            return idx[(h1, h2)], i
    return -1, -1

for line in sys.stdin:
    s, t = line.split()
    n, m = len(s), len(t)
    s1 = get_hashes(s, M1)
    s2 = get_hashes(s, M2)
    t1 = get_hashes(t, M1)
    t2 = get_hashes(t, M2)
    pow1 = get_pows(max(n, m), M1)
    pow2 = get_pows(max(n, m), M2)
    l, r = 0, n + 1
    while r - l > 1:
        mid = (r + l) // 2
        i1, i2 = check(mid)
        if i1 == -1:
            r = mid
        else:
            l = mid
```

Здесь, как и в четвертой задаче, подсчитываем хэшзначения всех префиксов, с помощью которых вычисляем хешзначения строки. Далее, пробегаясь по циклам, ищем наибольшую общую подстроку.

Тесты:



Доп. Задачи (Задачи 1, 2, 3, 6, 9)

1 Задача. Наивный поиск подстроки в строке

Дан неориентированный граф и две различные вершины u и v. Проверьте, есть ли путь между u и v.

Решение:

```
n_sub = len(substring)
n_string = len(string)

match = []

for i in range(n_string):
    if string[i] == substring[0]:
        if i + n_sub <= n_string:
            if string[i:i + n_sub] == substring:
                match.append(str(i + 1))
                i += n_sub

            visited[j] = True
```

Проходясь по строке t, смотрим, не равен ли какой-либо символ из нее первому символу подстроки p. Если равен, то проверяем длину подстроки и равенство всего подстрочного выражения фрагменту строки t.

Тесты:

100	OK	0.031	9043968	11004	497
101	OK	0.031	9809920	10005	48889
102	OK	0.031	9015296	20003	6
103	OK	0.031	9031680	13337	3
104	OK	0.015	9228288	10912	10925
105	OK	0.031	8994816	10015	2041
106	OK	0.031	9842688	10008	48883
107	OK	0.031	9502720	15004	23903

2 Задача. Карта

Команда Александра Смоллетта догадалась, что сокровища находятся на x шагов восточнее красного креста, однако определить значение числа она не смогла. По возвращению на материк Александр Смоллетт решил обратиться за помощью в расшифровке послания к знакомому мудрецу. Мудрец поведал, что данное послание таит за собой некоторое число. Для вычисления этого числа необходимо было удалить все пробелы между словами, а потом посчитать количество способов вычеркнуть все буквы кроме трех так, чтобы полученное слово из трех букв одинаково читалось слева направо и справа налево. Александр Смоллетт догадывался, что число, зашифрованное в послании, и есть число x. Однако, вычислить это число у него не получилось. После смерти капитана карта была безнадежно утеряна до тех пор, пока не оказалась в ваших руках. Вы уже знаете все секреты, осталось только вычислить число x.

Решение:

```
def halfGraph(u):
    global sides, total_colours
    search_queue = deque()
    search_queue.append((u, 0))
    visited = []
    while search_queue:
        cur_node, colour = search_queue.popleft()
        if cur_node not in visited:
            total_colours[cur_node] = colour
            visited.append(cur_node)
            for node in sides[cur_node]:
                if colour == 0:
                    search_queue.append((node, 1))
                else:
                    search_queue.append((node, 0))
        elif total_colours[cur_node] != colour:
            return 0
    return 1
```

Находим числовое значение каждого символа. Далее проходимся циклом в обратном порядке по строке и по ключам в массиве словарей префикса. Ответ увеличиваем на число возможных способов вычеркивания букв.

Тесты:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.546	Превышение	300002	16
1	OK	0.031	9007104	10	1
2	OK	0.031	9011200	34	3
3	OK	0.031	9003008	5	1
4	OK	0.031	8978432	6	1
5	OK	0.015	9011200	7	1
6	OK	0.031	8998912	9	2
7	OK	0.031	9023488	7	1
8	OK	0.031	9023488	7	1
9	OK	0.062	8994816	13	2
10	OK	0.015	9043968	202	6
11	OK	0.031	9019392	202	6
12	OK	0.015	8986624	202	6
13	OK	0.015	8990720	202	6
14	OK	0.031	9179136	202	5

3 Задача. Паттерн в тексте

В этой задаче ваша цель – реализовать алгоритм Рабина-Карпа для поиска заданного шаблона (паттерна) в заданном тексте.

Решение:

```
def PolyHash(pat, p, x):
    result = 0
    for i in reversed(range(pattern_length)):
        result = (result * x + ord(pat[i])) % p
    return result % p


def PrecomputeHashes(T, p, x):
    global pattern_length, text_length
    H = [0] * (text_length - pattern_length + 1)
    S = T[text_length - pattern_length: text_length]
    H[text_length - pattern_length] = PolyHash(S, p, x)
    y = 1
    for i in range(1, pattern_length + 1):
        y = (y * x) % p
    for i in range(text_length - pattern_length - 1, -1, -1):
        H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i + pattern_length]) + p) % p
    return H


def Rabin_Karp(pattern, text):
    global pattern_length, text_length
    p = 10 ** 9 + 9
    x = 127
    count = 0
    result = []
    hash_pattern = PolyHash(pattern, p, x)
    hash_string = PrecomputeHashes(text, p, x)
    for i in range(text_length - pattern_length + 1):
        if hash_pattern != hash_string[i]:
            continue
        count += 1
        result.append(str(i + 1))
    print(str(count) + "\n" + " ".join(result))

Rabin_Karp(pattern, text)
```

Суть задачи состоит в хешировании и равнении полученных хэшей. В PolyHash получаем полиномиальный хеш подстроки, а в PrecomputeHashes подсчитываем хеши. В конце с помощью алгоритма Рабина-Карпа мы сравниваем строку с подстрокой по хешам.

Тесты:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла	Группа тестов
Max		0.984	146931712	2000003	6888904	
1	OK	0.015	9150464	14	6	
2	OK	0.015	9125888	6	4	
3	OK	0.031	9142272	6	3	
4	OK	0.062	9080832	7	6	0
5	OK	0.031	9089024	7	3	
6	OK	0.031	9121792	9	6	
7	OK	0.015	9158656	10	4	
8	OK	0.359	23060480	900004	3	
9	OK	0.406	34381824	601028	1334	
10	OK	0.375	26865664	799942	47137	5
11	OK	0.531	72302592	600005	4088889	
12	OK	0.343	11583488	1200003	6	
13	OK	0.343	23048192	900004	3	
14	OK	0.437	39804928	720004	764368	
15	OK	0.390	34287616	601009	1334	5
16	OK	0.593	90603520	600008	4088881	
17	OK	0.468	51781632	900004	1988909	

6 Задача. Z-функция

Постройте Z-функцию для заданной строки s.

Решение:

```
def maze(k, path, sides):
    room_cur = 1
    for i in range(k):
        colour = path[i]
        flag = False
        for vertex, side in sides[room_cur]:
            if side == colour:
                room_cur = vertex
                flag = True
                break
        if not flag:
            return 'INCORRECT'
    return str(room_cur)
```

Для каждой позиции i перебираем ответ для неё z[i], начиная с нуля, и до тех пор, пока мы не обнаружим несовпадение или не дойдём до конца строки. При этом в процессе вычисления Z-функции сохраняем последнее ненулевое найденное значение в виде границ отрезка [l;r], равного соответствующему префиксу.

Тесты:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.484	139309056	1000002	6888887
1	OK	0.031	8957952	8	9
2	OK	0.015	9031680	9	11
42	OK	1.312	121131008	1000002	4444442
43	OK	1.218	109404160	1000002	2977775
44	OK	1.093	101916672	1000002	2000042
45	OK	1.140	101982208	1000002	2004884

9 Задача. Декомпозиция строки

Строка `ABCABCDEDEDEF` содержит подстроку `ABC`, повторяющуюся два раза подряд, и подстроку `DE`, повторяющуюся три раза подряд. Таким образом, ее можно записать как `ABC*2+DE*3+F`, что занимает меньше места, чем исходная запись той же строки.

Ваша задача – построить наиболее экономное представление данной строки s в виде, продемонстрированном выше, а именно, подобрать такие $s_1, a_1, ..., s_k, a_k$, где s_i - строки, а a_i - числа, чтобы $s = s_1 \cdot a_1 + ... + s_k \cdot a_k$. Под операцией умножения строки на целое положительное число подразумевается конкатенация одной или нескольких копий строки, число которых равно числовому множителю, то есть, `ABC*2=ABCABC`. При этом требуется минимизировать общую длину итогового описания, в котором компоненты разделяются знаком `+`, а умножение строки на число записывается как умножаемая строка и множитель, разделенные знаком `*`. Если же множитель равен единице, его, вместе со знаком `*`, допускается не указывать.

Решение:

```
sys.stdin = open('input.txt', "r")
s = input()
n = len(s)
s += '_'
dp = [n - i for i in range(n + 1)]
to = [[n - i, n - i] for i in range(n + 1)]
for i in range(n - 2, -1, -1):
    z = z_function(s[i:])
    if dp[i] > dp[i + 1] + 2:
        dp[i] = dp[i + 1] + 2
        to[i] = [1, 1]
    for j in range(i + 1, n + 1):
        k = 1
        while k * k <= j - i:
            if (j - i) % k:
                k += 1
                continue
            if z[k] + k >= j - i:
                if dp[i] > dp[j] + step(j - i, k, j):
                    dp[i] = dp[j] + step(j - i, k, j)
                    to[i] = [j - i, k]
            if z[(j - i) // k] + (j - i) // k >= j - i:
                if dp[i] > dp[j] + step(j - i, (j - i) // k, j):
                    dp[i] = dp[j] + step(j - i, (j - i) // k, j)
                    to[i] = [j - i, (j - i) // k]
            k += 1
```

Используя префикс-функцию, проходимся по строке - если в ней есть повторения, то записываем их как произведение, попутно удаляем одинаковые элементы подстроки. Если повторений нет, убираем одну букву и либо записываем ее просто так, либо добавляем к уже записанной неповторяющейся подстроке. Повторяем алгоритм пока строка не закончится.

Тесты:

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		Превышение	9359360	817	448
1	OK	0.031	9232384	15	12
2	OK	0.031	9224192	7	5
27	OK	0.562	9338880	450	448
28	OK	0.750	9334784	498	19
29	OK	0.750	9314304	498	56
30	TL	Превышение	9359360	817	0

Вывод:

В данной лабораторной работе я изучила графы, поиск в глубину, поиск в ширину, алгоритм Дейкстры, алгоритм Беллмана-Форда. Рассмотрены различные виды графов, типы их обходов и другие алгоритмы, связанные с графами.