Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Алгоритмы и структуры данных

Лабораторная работа №2_4

Подстроки

Выполнил:

Бараканов Жаргал Мырзабекович

Факультет ИКТ

Группа К3121

Преподаватель:

Харьковская Татьяна Александровна

Санкт-Петербург

2022

Задание 1.

Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.

- **Формат ввода / входного файла (input.txt).** Первая строка входного файла содержит p, вторая – t. Строки состоят из букв латинского алфавита;
- **Ограничения на входные данные.** $1 \le |p|, |t| \le 10^4$;
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки p в строку t. Во второй строке выведите в возрастающем порядке номера символов строки t, с которых начинаются вхождения p. Символы нумеруются с единицы;
- **Ограничение по времени.** 2 сек;
- **Ограничение по памяти.** 256 Мб.

Решение:

Считываем строки p и t, заводим счетчик “count”. Далее бежим по строке t, сравнивая i-й символ с первым символом строки p. Если они совпали, то проверяем подстроку длины строки p на совпадение с этой строкой. Если они совпали, то увеличиваем счетчик и в строку результата добавляем индекс начала подстроки в строке t. В конце записываем результаты в файл.

```
with open('input.txt') as f:
    p = f.readline()[:-1]
    t = f.readline()

count = 0
res = ''
for i in range(len(t) - len(p) + 1):
    if t[i] == p[0]:
        part = t[i:i+len(p)]
        if part == p:
            count += 1
            res += str(i+1) + ' '
res = res[:-1]

with open('output.txt', 'w') as f:
    f.write(str(count) + '\n' + str(res))
```

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.078	9289728	20003	48889

Задание 3.

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

- **Формат ввода / входного файла (input.txt).** На входе две строки: паттерн P и текст T. Требуется найти все вхождения строки P в строку T в качестве подстроки;
- **Ограничения на входные данные.** $1 \leq |P|, |T| \leq 10^6$. Паттерн и текст содержат только латинские буквы;
- **Формат вывода / выходного файла (output.txt).** В первой строке выведите число вхождений строки P в строку T. Во второй строке выведите в возрастающем порядке номера символов строки T, с которых начинаются вхождения P. Символы нумеруются с единицы;
- **Ограничение по времени.** 2 сек;
- **Ограничение по памяти.** 256 Мб.

Решение:

Для начала нам понадобится хеш-функция, которая будет считать значения хеша для подстрок.

```
3   def PolyHash(P, p, x):
4       res = 0
5       for i in reversed(range(n)):
6           res = (res * x + ord(P[i])) % p
7       return res
```

Следующая функция подсчитывает массив хешей для всех подстрок текста длины паттерна.

```
10  def PrecomputeHashes(T, p, x):
11      global n, m
12      H = [0] * (m - n + 1)
13      S = T[m - n : m]
14      H[m - n] = PolyHash(S, p, x)
15      y = 1
16      for i in range(1, n + 1):
17          y = (y * x) % p
18      for i in range(m - n - 1, -1, -1):
19          H[i] = (x * H[i + 1] + ord(T[i]) - y * ord(T[i + n]) + p) % p
20      return H
```

Далее в основной функции сравниваем значения хешей подстрок текста и паттерна. При совпадении увеличиваем счетчик и добавляем в массив индекс начала подстроки, совпавшей с паттерном. В конце выводим результаты в файл.

```python
def RabinKarp(pattern, text):
    global n, m
    p = 10 ** 9 + 9
    x = random.randint(1, p-1)
    count = 0
    res = []
    hPattern = PolyHash(pattern, p, x)
    H = PrecomputeHashes(text, p, x)
    for i in range(m - n + 1):
        if hPattern != H[i]:
            continue
        count += 1
        res.append(str(i + 1))
    with open("output.txt", "w") as f:
        f.write(str(count) + "\n" + " ".join(res))
```

В основном коде считываем паттерн и текст и вызываем функцию “RabinKarp”.

```python
with open("input.txt") as f:
    P = f.readline()[:-1]
    T = f.readline()

n, m = len(P), len(T)
RabinKarp(P, T)
```

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла	Группа тестов
Max		1.015	147513344	2000003	6888904	

Задание 4.

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.

- **Формат ввода / входного файла (input.txt).** Первая строка содержит строку s, состоящую из строчных латинских букв. Вторая строка содержит количество запросов q. Каждая из следующих q строк задает запрос тремя целыми числами a, b и l;
- **Ограничения на входные данные.** $1 \le |s| \le 500000$, $1 \le q \le 100000$, $0 \le a, b \le |s|-l$ (следовательно, индексы a и b начинаются с 0);
- **Формат вывода / выходного файла (output.txt).** Для каждого запроса выведите «Yes», если подстроки $s_a s_{a+1} \ldots s_{a+l-1} = s_b s_{b+1} \ldots s_{b+l-1}$ равны, и «No» – если не равны;

- **Ограничение по времени.** 10 сек;
- **Ограничение по памяти.** 512 Мб.

Решение:

Первые две функции идентичны с предыдущим заданием.

```
def PolyHash(P, l, p, x):
    res = 0
    for i in reversed(range(l)):
        res = (res * x + ord(P[i])) % p
    return res % p
```

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

В основном коде считываем исходную строку. Далее считываем индексы начала подстрок и их длину. Формируем массив хешей подстрок длины “l” и сравниваем хеши подстрок, начинающихся с данных индексов. Если они совпадают, то записываем в файл “Yes”, в обратном случае – “No”.

```
with open('input.txt') as f:
    with open('output.txt', 'w') as f1:
        s = f.readline().strip()
        lS = len(s)
        q = int(f.readline())
        p = 10**9+7
        x = random.randint(1, p-1)
        for i in range(q):
            a, b, l = map(int, f.readline().split())
            H = PrecomputeHashes(s, lS, l, p, x)
            if H[a] == H[b]:
                f1.write('Yes\n')
            else:
                f1.write('No\n')
```

Задание 6.

Постройте Z-функцию для заданной строки s.

- **Формат ввода / входного файла (input.txt).** Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита;
- **Ограничения на входные данные.** $2 \leq |s| \leq 10^6$;

- **Формат вывода / выходного файла (output.txt).** Выведите значения Z-функции для всех индексов 1, 2, ..., |s| строки s, в указанном порядке;
- **Ограничение по времени.** 2 сек;
- **Ограничение по памяти.** 256 Мб.

Решение:

В функции “z_func” вычисляется массив значений Z-функции для всех символов строки s, кроме первого.

```
def z_func(s):
    z = [''] * (len(s)-1)
    L = 0
    R = 0
    for i in range(1, len(s)):
        if i >= R:
            j = 0
            while i+j < len(s) and s[i+j] == s[j]:
                j += 1
            L = i
            R = i+j
            z[i-1] = str(j)
        else:
            if int(z[i-L-1]) < R-i:
                z[i-1] = z[i-L-1]
            else:
                j = R-i
                while i+j < len(s) and s[i+j] == s[j]:
                    j += 1
                L = i
                R = i+j
                z[i-1] = str(j)
    return z
```

В основной программе считываем строку и вызываем вышеуказанную функция, получая искомый массив значений, который и выводим в файл.

```
with open('input.txt') as f:
    s = f.readline().strip()

Z = z_func(s)
with open('output.txt', 'w') as f:
    f.write(' '.join(Z))
```

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.062	98320384	1000002	6888887

Задание 7.

В задаче на наибольшую общую подстроку даются две строки s и t, и цель состоит в том, чтобы найти строку w максимальной длины, которая является подстрокой как s, так и t. Это естественная мера сходства между двумя строками. Задача имеет применения для сравнения и сжатия текстов, а также в биоинформатике. Эту проблему можно рассматривать как частный случай проблемы расстояния редактирования (Левенштейна), где разрешены только вставки и удаления. Следовательно, ее можно решить за время O(|s||t|) с помощью динамического программирования. Есть также весьма нетривиальные структуры данных для решения этой задачи за линейное время O(|s| + |t|). В этой задаче ваша цель – использовать хеширование для решения почти за линейное время.

- **Формат ввода / входного файла (input.txt).** Каждая строка входных данных содержит две строки s и t, состоящие из строчных латинских букв.
- **Ограничения на входные данные.** Суммарная длина всех s, а также суммарная длина всех s не превышает 100 000.
- **Формат вывода / выходного файла (output.txt). Для** каждой пары строк s_i и t_i найдите ее самую длинную общую подстроку и уточните ее параметры, выведя три целых числа: ее начальную позицию в s, ее начальную позицию в t (обе считаются с 0) и ее длину. Формально выведите целые числа $0 \le i < |s|$, $0 \le j < |t|$ и $l \ge 0$ такие, что и l максимально. (Как обычно, если таких троек с максимальным l много, выведите любую из них.)
- **Ограничение по времени.** 15 сек.
- **Ограничение по памяти.** 512 Мб.

Решение:

Функции, использующиеся в данном задании такие же, как и в задании №4.

```
def PolyHash(P, l, p, x):
    res = 0
    for i in reversed(range(l)):
        res = (res * x + ord(P[i])) % p
    return res % p


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

В основной программе считываем две данные строки. За длину k наибольшей общей возможной подстроки берем минимум их длин. Далее идем от k до 1, строя на каждой итерации массивы хешей подстрок длины i для каждой из данных строк. Далее проходимся по обоим массивам и если находим совпадение, то выводим индексы начала подстрок, их длину и останавливаем поиск. Если же к концу цикла так и не было ни одного совпадения, то выводим ‘0 1 0’.

```
with open('input.txt') as f:
    while True:
        line = f.readline()
        if not line:
            exit()
        s, t = map(str, line.split())
        lS, lT = len(s), len(t)
        k = min(lS, lT)
        p = 10**9+7
        x = random.randint(1, p-1)
        flag = False
        for i in reversed(range(1, k+1)):
            Hs = PrecomputeHashes(s, lS, i, p, x)
            Ht = PrecomputeHashes(t, lT, i, p, x)
            for j in range(len(Hs)):
                for h in range(len(Ht)):
                    if Hs[j] == Ht[h]:
                        print(j, h, i)
                        flag = True
                        break
                if flag:
                    break
            if flag:
                break
        if not flag:
            print(0, 1, 0)
```

Задание 5 (доп).

Поиск подстроки

(Время: 0,2 сек. Память: 16 Мб Сложность: 38%)

Найти все вхождения строки T в строке S.

Входные данные

В первой строке входного файла INPUT.TXT записана строка S, во второй строке записана строка T. Обе строки состоят только из английских букв. Длины строк могут быть в диапазоне от 1 до 50 000 включительно.

Выходные данные

В выходной файл OUTPUT.TXT нужно вывести все вхождения строки T в строку S в порядке возрастания. Нумерация позиций строк начинается с нуля.

Решение:

Для решения данной задачи в рамках ограничений хватило обычного наивного поиска подстроки.

```
with open('../1/input.txt') as f:
    t = f.readline()[:-1]
    p = f.readline()[:-1]

res = []
for i in range(len(t) - len(p) + 1):
    if t[i] == p[0]:
        part = t[i:i+len(p)]
        if part == p:
            res.append(str(i))

print(' '.join(res))
```

Задание 8 (доп).

Подстроки из одинаковых букв

(Время: 1 сек. Память: 16 Мб Сложность: 32%)

В заданной строке, состоящей из малых английских букв, необходимо найти пару самых длинных подстрок, состоящих из одних и тех же букв (возможно, в разном порядке). Например, в строке twotwow это будут подстроки wotwo и otwow.

Входные данные

Входной файл INPUT.TXT содержит исходную строку, длиной от 1 до 100 символов.

Выходные данные

Выходной файл OUTPUT.TXT должен содержать единственное число – длину подстрок в максимальной паре, или 0, если таких подстрок в строке нет.

Решение:

Чтобы найти две подстроки максимальной длины, состоящей из одних и тех же символов, достаточно найти самую длинную подстроку, которая

начинается и оканчивается одним и тем же символом. Тогда максимальной длинной таких подстрок будет длина найденной подстроки без одного символа. В программе мы идем счетчиком k от l-1 до 1 – максимизируем длину возможных подстрок. Далее счетчиком i сравниваем i-й и (i+k)-й символы. Если они одинаковые, то выводим k. Если в течение циклов строки с двумя одинаковыми символами на концах не нашлось, то выводим 0.

```
with open('input.txt') as f:
    s = f.readline()

l = len(s)
flag = False
for k in range(l-1, 0, -1):
    for i in range(l-k):
        if s[i] == s[i+k]:
            print(k)
            flag = True
            break
    if flag:
      break
if not flag:
    print(0)
```

17287506	16.06.2022 17:53:19	Бараканов Жаргал Мырзабекович	0361	Python	Accepted		0,046	362 Кб

Задание 10 (доп).

Abracadabra

(Время: 5 сек. Память: 64 Мб Сложность: 59%)

Строка s называется супрефиксом для строки t, если t начинается с s и заканчивается на s. Например, «abra» является супрефиксом для строки «abracadabra». В частности, сама строка t является своим супрефиксом. Супрефиксы играют важную роль в различных алгоритмах на строках.

В этой задаче требуется решить обратную задачу о поиске супрефикса, которая заключается в следующем. Задан словарь, содержащий n слов t_1, t_2, …, t_n и набор из m строк-образцов s_1, s_2, …, s_m. Необходимо для каждой строки-образца из заданного набора найти количество слов в словаре, для которых эта строка-образец является супрефиксом.

Требуется написать программу, которая по заданному числу n, n словам словаря t_1, t_2, …, t_n, заданному числу m и m строкам-образцам s_1, s_2, …, s_m вычислит для каждой строки-образца количество слов из словаря, для которых эта строка-образец является супрефиксом.

Входные данные

Первая строка входного файла INPUT.TXT содержит целое число n ($1 \le n \le 200\,000$). Последующие n строк содержат слова $t_1, t_2, \ldots, t_n$, по одному слову в каждой строке. Каждое слово состоит из строчных букв английского алфавита. Длина каждого слова не превышает 50. Суммарная длина всех слов не превышает 10^6. Словарь не содержит пустых слов.

Затем следует строка, содержащая целое число m ($1 \le m \le 200\,000$). Последующие m строк содержат строки-образцы $s_1, s_2, \ldots, s_m$, по одной на каждой строке. Каждая строка-образец состоит из строчных букв английского алфавита. Длина каждой строки-образца не превышает 50. Суммарная длина всех строк-образцов не превышает 10^6. Никакая строка-образец не является пустой строкой.

Выходные данные

В выходной файл OUTPUT.TXT выведите m чисел, по одному на строке. Для каждой строки-образца в порядке, в котором они заданы во входном файле, следует вывести количество слов словаря, для которых она является супрефиксом.

Решение:

Для решения задачи нам понадобится два словаря: “strs” и “supr”. В первом мы будем хранить все уникальные строки и их количество, а во втором – уникальные супрефиксы и их количество. Считывая все n строк, мы проверяем их на наличие суперфиксов. Если таковой уже был, то увеличиваем его количество на 1, иначе создаем ячейку в словаре “supr” с количеством, равным одному. Далее, если строка встретилась впервые, то создаем ячейку в словаре “strs”. Иначе – увеличиваем количество на 1.

```
with open('input.txt') as f:
    n = int(f.readline())
    strs = {}
    supr = {}
    for i in range(n):
        s = f.readline().strip()
        for j in range(1, len(s)):
            if s[:j] == s[len(s)-j:]:
                if s[:j] in supr:
                    supr[s[:j]] += 1
                else:
                    supr[s[:j]] = 1
        if s not in strs:
            strs[s] = 1
        else:
            strs[s] += 1
```

Далее, считываем m строк, проверяя их наличие в обоих словарях. Если строка есть в данном словаре, то добавляем к счетчику значение ячейки данного словаря. На каждой итерации выводим счетчик.

```
    m = int(f.readline())
    for i in range(m):
        count = 0
        cur_s = f.readline().strip()
        if cur_s in strs:
            count += strs[cur_s]
        if cur_s in supr:
            count += supr[cur_s]
        print(count)
```

17287892	16.06.2022 19:23:56	Бараканов Жаргал Мырзабекович	0857	Python	Accepted		1,593	38 Мб

Вывод:

В ходе данной лабораторной работы были решены различные задачи по работе с подстроками. Были изучены и применены на практике наивный поиск подстрок, алгоритм Рабина-Карпа и Z-функция.