САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе № 4
по курсу «Алгоритмы и структуры данных»
Тема: Подстроки
Вариант №4

Выполнил:
Волжева М.И.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2023 г.

Содержание отчета

Задачи по варианту

Задача №1. Наивный поиск подстроки в строке [2 s, 256 Mb, 1 балл]

- Текст задачи:
  Даны строки p и t. Требуется найти все вхождения строки p в строку t в качестве подстроки.
- Формат входного файла (input.txt).
  Первая строка входного файла содержит p, вторая – t. Строки состоят из букв латинского алфавита.
- Ограничения на входные данные.
  $1 \le |p|, |t| \le 104$
- Формат 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.
- Пример:

Input.txt	aba abaCaba
Output.txt	2 1 5

Листинг кода:

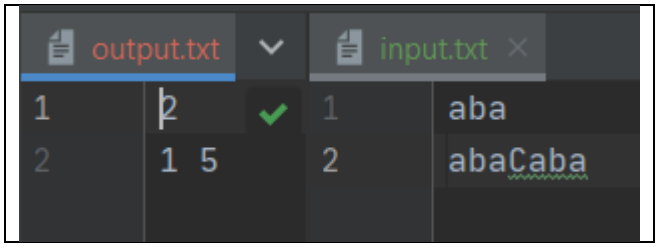
```
import time
import os, psutil

t_start = time.perf_counter()
process = psutil.Process(os.getpid())

f = open('input.txt')
p = f.readline()[:-1]
print(p)
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) + ' '

d = open('output.txt', 'w')
d.write(str(count) + '\n' + str(res))
```

```
print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Код читает данные из файла. Далее проходится по строке, если какой-то символ строки совпадает с 1 символом подстроки, то мы проверяем, есть ли там вся строка. Если да – сохраняем номер первого элемента и увеличиваем счётчик count.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0012629000000000112	14.1953125

Вывод по задаче: Мы научились находить вхождение подстроки в строку интутивным способом.

Задача №6. Z-функция [2 s, 256 Mb, 1.5 балла]

- Текст задачи:
  Постройте Z-функцию для заданной строки s.
  Тогда Z-функция ("зет-функция") от этой строки — это массив длины n, i-ый элемент которого равен наибольшему числу символов, начиная с позиции i, совпадающих с первыми символами строки s.
- Формат ввода / входного файла (input.txt). Одна строка входного файла содержит s. Строка состоит из букв латинского алфавита.
- Ограничения на входные данные. $2 \le |s| \le 10^6$
- Формат выходного файла (output.txt).
  Выведите значения Z-функции для всех индексов 1, 2, ..., |s| строки s, в указанном порядке.
- Пример:

Input.txt	aaaAAA	abacaba
Output.txt	2 1 0 0 0	0 1 0 3 0 1

Листинг кода:

```
import time
import os, psutil


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


t_start = time.perf_counter()
process = psutil.Process(os.getpid())

f = open('input.txt')
s = f.readline().strip()

Z = z_func(s)
d = open('output.txt', 'w')
d.write(' '.join(Z))


print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Данный код реализует алгоритм вычисления Z-функции строки s. Z-функция для каждой позиции i в строке s определяет наибольшую подстроку, начинающуюся с i и совпадающую с некоторым префиксом всей строки s.

Алгоритм начинает с инициализации массива z длиной на единицу меньше, чем длина строки s. Затем устанавливаются начальные значения левой и правой границ L и R, соответственно, равные нулю. Затем происходит итерация по всем индексам строки s начиная с 1.

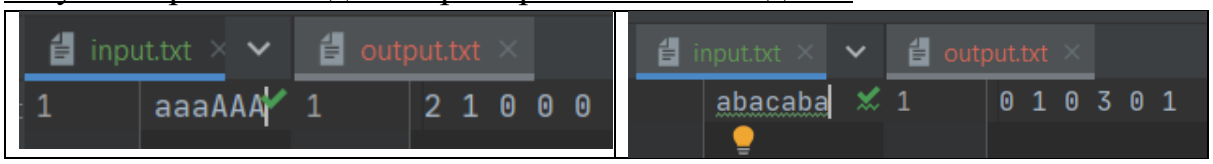
Если текущий индекс i больше или равен правой границы R, то происходит последовательный перебор всех символов строки, начиная с i-го, и сравнение их с символами, начиная с первого, пока символы совпадают. При каждом совпадении инкрементируется переменная j. Затем устанавливаются новые значения левой и правой границ L и R, соответственно, равные i и i+j, соответствующий элемент z[i-1] устанавливается равным строковому представлению числа j.

Если текущий индекс i меньше правой границы R, то вычисляется значение k, равное i-L-1, и сравнивается со значением R-i. Если k меньше R-i, то соответствующий элемент z[i-1] устанавливается равным элементу z[k]. В противном случае происходит последовательный перебор символов, начиная с i+j, и сравнение их с символами, начиная с j, пока символы совпадают. При каждом совпадении инкрементируется переменная j. Затем устанавливаются новые значения левой и правой границ L и R, соответственно, равные i и i+j, соответствующий элемент z[i-1] устанавливается равным строковому представлению числа j.

В результате работы функции z_func возвращается массив z, содержащий значения Z функции для каждой позиции строки s. Затем открывается файл 'input.txt' и считывается строка s из него. Вычисляется массив Z, вызывая функцию z_func.

Затем открывается файл 'output.txt' для записи и в него записывается строка, состоящая из элементов массива Z, разделенных пробелами.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.014278411865234375	14. 23828125
Прмер	0.0011632000000000031	14.0390625

Вывод по задаче: Мы построили z-функцию заданной строки.

Задача №7. Наибольшая общая подстрока [15 s, 512 Mb, 2 балла]

- Текст задачи:
  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.
- 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 100000.
- Формат выходного файла (output.txt).
  Для каждой пары строк si и ti 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.
- Пример:

Input.txt	cool toolbox aaa bb aabaa babbaab
Output.txt	1 1 3 0 1 0 0 4 3

Листинг кода:

```
import random
import time
import os, psutil


def GetHash(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] = GetHash(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


t_start = time.perf_counter()
process = psutil.Process(os.getpid())
f = open('input.txt')
d = open('output.txt', 'w')
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]:
                    d.write(' ' + str(j) + ' ' + str(h) + ' ' + str(i) +
'\n')
                    flag = True
                    break
            if flag:
                break
        if flag:
            break
    if not flag:
        d.write('0 1 0')


print("Time of working: %s second" % (time.perf_counter() - t_start))
print("Memory", process.memory_info().rss/(1024*1024), "mb")
```

Текстовое объяснение решения:

Этот код считывает ввод из файла, который содержит две строки s и t. Затем он использует алгоритм Рабина-Карпа, чтобы найти самую длинную общую подстроку между s и t.

Алгоритм Рабина-Карпа — это алгоритм поиска строки, который использует хеширование для поиска подстроки длины m в тексте длины n. В этой реализации алгоритм используется для поиска самой длинной общей подстроки между s и t. Алгоритм работает путем вычисления хэш значений всех возможных подстрок s и t длины k, где k начинается с

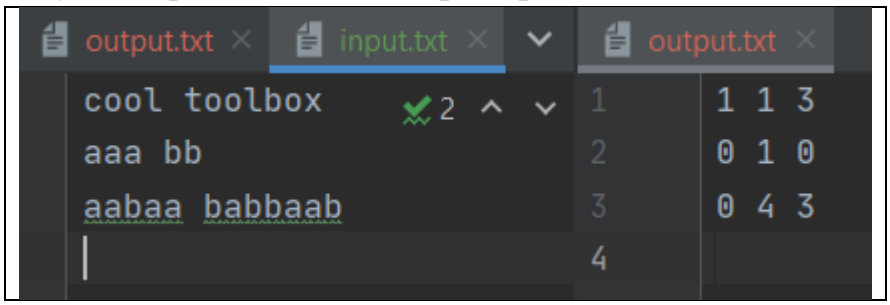
минимальной длины s и t и уменьшается до тех пор, пока не будет найдена общая подстрока.

В коде используются две вспомогательные функции: GetHash и PrecomputeHashes. GetHash вычисляет хеш-значение строки P длины l, используя хеш-функцию h(P) = P[0]x^(l-1) + P[1]x^(l 2) + ... + P [l-2]x + P[l-1] mod p, где x — случайное целое число от 1 до p-1, а p — большое простое число. PrecomputeHashes вычисляет хэш-значения всех подстрок T длины k.

Основной цикл кода перебирает все возможные значения k, начиная с минимальной длины s и t и уменьшаясь до тех пор, пока не будет найдена общая подстрока. Для каждого значения k он вычисляет хеш-значения всех подстрок s и t длины k, используя GetHash и PrecomputeHashes, и проверяет наличие общего хеш-значения. Если найдено общее хэш-значение, печатаются начальные позиции общей подстроки в s и t и длина подстроки.

Если общая подстрока не найдена, код печатает «0 1 0», чтобы указать, что общей подстроки нет.

Результат работы кода на примерах из текста задачи:



Тестирование алгоритма:

	Время выполнения (seconds)	Затраты памяти (mbs)
Пример	0.0010133000000000225	14. 23828125

Вывод по задаче: Мы научились находить наибольшую общую подстроку с помощью хеширования.

Дополнительные задачи

Задача №5.

- Текст задачи:
  Найти все вхождения строки T в строке S.
- Формат ввода.
  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 50000 включительно.
- Формат выходного файла (output.txt).
  В выходной файл OUTPUT.TXT нужно вывести все вхождения строки T в строку S в порядке возрастания. Нумерация позиций строк начинается с нуля.
- Пример:

Input.txt	abababababa aba
Output.txt	0 5 7

Листинг кода:

```
f = open('input.txt')
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))


d = open('output.txt', 'w')
d.write(' '.join(res))
```

Текстовое объяснение решения:

Код читает данные из файла. Далее проходится по строке, если какой-то символ строки совпадает с 1 символом подстроки, то мы проверяем, есть ли там вся строка. Если да – сохраняем номер первого элемента и увеличиваем счётчик count.

Результат работы кода на примерах из текста задачи:

Тестирование алгоритма:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19545375	15.06.2023 16:03:08	Волжева Мария Ильинична	0202	Python	Accepted		0,171	2230 Кб

Вывод по задаче: Мы нашли все вхождения строки T в строке S

Задача № 8.

- Текст задачи:
  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.
- Формат ввода.
  Входной файл INPUT.TXT содержит исходную строку, длиной от 1 до 100 символов.
- Формат выходного файла (output.txt).
  Выходной файл OUTPUT.TXT должен содержать единственное число – длину подстрок в максимальной паре, или 0, если таких подстрок в строке нет.
- Пример:

Input.txt	abcde	abcdea
Output.txt	0	5

Листинг кода:

```
import time
import tracemalloc

t = time.process_time()
tracemalloc.start()


f = open('input.txt')
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)


print('Время работы: %s секунд' % (time.process_time() - t))
```

```
print('Затраты памяти:', float(tracemalloc.get_tracemalloc_memory()) / (2
** 20), 'мб')
```

Текстовое объяснение решения:

Данный код читает данные из файла 'input.txt', содержащего одну строку s, и затем ищет длину самой длинной подстроки s, которая является палиндромом.

Первые две строки кода открывают файл 'input.txt' и читают первую строку из него, сохраняя ее в переменную s.

Затем создается переменная l, которая содержит длину строки s. Далее создается флаг flag, который используется для определения того, была ли найдена палиндромическая подстрока. Значение флага изначально устанавливается в False.

Затем происходит двойной цикл for, который перебирает все возможные длины k для подстроки s, начиная с максимальной длины (l-1) и заканчивая длиной 1. Для каждой длины k происходит вложенный цикл for, который перебирает все возможные начальные позиции i для подстроки длины k.

На каждой итерации внутреннего цикла происходит проверка, совпадают ли символы в позициях i и i+k. Если символы совпадают, то это означает, что найдена палиндромическая подстрока длины k, и текущее значение k записывается в файл. Также флаг flag устанавливается в True и происходит выход из внутреннего цикла.

Если после завершения всех циклов флаг flag остается равным False, то это означает, что палиндромическая подстрока не была найдена, и в файл записывается 0.

Результат работы кода на примерах из текста задачи:

Тестирование алгоритма:

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
19546192	15.06.2023 20:37:16	Волжева Мария Ильинична	0361	Python	Accepted		0,031	482 Кб

Вывод по задаче: Мы научились находить пару самых длинных подстрок, состоящих из одних и тех же букв (возможно, в разном порядке).

Задача №10.

- Текст задачи:

  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. В этой задаче требуется решить обратную задачу о

поиске супрефикса, которая заключается в следующем. Задан словарь, содержащий n слов t1, t2, ..., tn и набор из m строк-образцов s1, s2, ..., sm. 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 t1, t2, ..., tn, заданному числу m и m строкам образцам s1, s2, ..., sm вычислит для каждой строки-образца количество слов из словаря, для которых эта строка-образец является супрефиксом

- Формат ввода / входного файла (input.txt). Первая строка входного файла input.txt содержит целое число n ($1 \le n \le 200\,000$). Последующие n строк содержат слова t1, t2, ..., tn, 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 106. Словарь не содержит пустых слов. Затем следует строка, содержащая целое число m ($1 \le m \le 200\,000$). Последующие m строк содержат строки-образцы s1, s2, ..., sm, 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 106. Никакая строка-образец не является пустой строкой.
- Формат выходного файла (output.txt).
  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.
- Пример:

Input.txt	abcabca abcda	ccba accbbaa
Output.txt	ca	Ccba

Листинг кода:

```
f = open('input.txt')
d = open('output.txt', 'w')
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
m = int(f.readline())
strs1 = {}
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]
    d.write(str(count) + '\n')
```

Текстовое объяснение решения:

Данный код открывает файл 'input.txt' и читает из него данные для дальнейшей обработки.

Первая строка в файле представляет собой число n - количество строк, которые нужно обработать. Затем создаются два словаря: strs и supr, которые будут использоваться для хранения информации о строках и суперпалиндромах.

Далее происходит цикл for, который проходит по всем строкам, которые нужно обработать. Внутри этого цикла считывается строка s при помощи функции f.readline(), которая удаляет символы переноса строки и пробелы в начале и конце строки при помощи метода strip().

Затем происходит цикл for, который проходит по всем возможным длинам префикса s и суффикса s. Если префикс равен суффиксу, то это означает, что s является суперпалиндромом. В этом случае в словарь supr добавляется суперпалиндром с ключом s[:j] и значение счетчика, которое соответствует количеству суперпалиндромов с таким же префиксом.

Затем происходит проверка, существует ли уже данная строка s в словаре strs. Если данная строка не была найдена ранее, то она добавляется в словарь strs с начальным значением счетчика 1. В противном случае значение счетчика для данной строки увеличивается на 1.

Далее происходит считывание числа m из файла 'input.txt', которое представляет собой количество запросов на подсчет количества строк.

Затем создается словарь strs1 для хранения информации о количестве строк, найденных в запросах.

Затем происходит цикл for, который проходит по всем запросам на подсчет количества строк. Для каждого запроса сначала устанавливается начальное значение счетчика count в 0. Затем считывается строка cur_s из файла 'input.txt' и проверяется, есть ли данная строка в словаре strs. Если строка cur_s найдена в словаре strs, то значение счетчика count увеличивается на количество вхождений этой строки в словаре strs. Затем происходит проверка, есть ли данная строка cur_s в словаре supr. Если строка cur_s найдена в словаре supr, то значение счетчика count увеличивается на количество суперпалиндромов с префиксом cur_s, хранящихся в словаре supr.

Результат работы кода на примерах из текста задачи:

Тестирование алгоритма:

19546422	15.06.2023 21:42:40	Волжева Мария Ильинична	0857	Python	Accepted		1,687	34 Мб

Вывод по задаче: Было выяснено, что такое суперпрефикс и решена задача.

Вывод

Мы узнали некоторые термины, связанные со строками и подстроками, узнали некоторые алгоритмы и научились решать задачи.