Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет трансляционных информационных технологий
Факультет инфокоммуникационных технологий

Дисциплина: Алгоритмы и структуры данных

Отчет по Лабораторной работе №4

Выполнила: Микулина Алиса Романовна

Группа: К3143, 1 курс

Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург

29.06.2022

Описание задания

Задание 1.

Наивно найти подстроку в строке. Долго, конечно, но надо

Задание 2.

В строке найти все возможные палиндромы из трех букв, которые можно получить вычеркиванием букв в строке.

Задание 3.

Опять ищем подстроку в строке, но по оптимизированному алгоритму Рабина-Карпа.

Задание 4.

С помощью магии, приворотов и хеширования нужно сравнить две подстроки в строке, и вывести, равны ли они.

Задание 5.

Вроде как обычная префикс функция, самая обыкновенная без всяких там приколов.

Задание 6.

Нужно написать z-функцию. Тут, скорее, сложно понять, что такое эта функция, а когда поймешь уже не сложно.

Задание 7.

Бинарным методом ищем наибольшую общую подстроку двух строк, еще и с помощью хеширования. Не так сложно, как звучит на самом деле, норм.

Задание Сдвиг Текста.

Из конца строки взяли сколько-то символов, перевернули и поставили в начало. Надо определить, сколько.

Задание Поиск Подстроки.

Тоже ищем подстроку, но очень быстро. Быстрее всего (по крайней мере тесты прошло) решение с помощью префиксов.

Описание решения и исходный код

Задача 1.

Все начинается так просто… Просто наивный алгоритм для нахождения подстроки. Мы знаем, какой длины строку ищем, поэтому просто циклом проходимся по строке, смотря на все вхождения такой длины, и сравниваем их с искомым) По сути, все)))

```
def find_instances(to_find, text):
    inst = ''
    count = 0
    len_f = len(to_find)
    len_t = len(text)
    iterable = len_t - (len_f - 1)
    for i in range(iterable):
        comparable = text[i:i + len_f]
        if comparable == to_find:
            count += 1
            inst += str(i + 1) + ' '
    inst = inst[:-1]
    return [count, inst]



with open('input.txt') as f:
    to_find = f.readline().strip()
    text = f.readline().strip()

ans = find_instances(to_find, text)

with open('output.txt', 'w') as d:
    d.write(str(ans[0]) + '\n')
    d.write(ans[1])
```

Задача 2.

Тут забавная задачка. Нужно найти все существующие палиндромы длиной в 3 символа, которые можно получить путем вычеркивания остальных символов.

На самом деле тоже очень просто, мы делаем сортировку подсчетом по символам и при этом храним индекс каждого вхождения символа. Потом мы просто смотрим индексы одинаковых элементов попарно и вычисляем количество букв между ними. Так мы, как раз, и найдем все микропалиндромы))

```
def find_three(line):
    count = 0
    first = ord('a')
    last = ord('z')
    full = []
    left = []
    for i in range(first, last + 1):
        full.append(0)
        left.append(0)
    for char in line:
        id_ = ord(char)
        full[id_ - first] += 1
    for char in line:
        id_ = ord(char) - first
        left[id_] += 1
        for i in range(len(full)):
            if i == id_:
                right = full[i] - left[i]
                cur_left = left[i] - 1
                count += cur_left * right
            else:
                right = full[i] - left[i]
                count += left[i] * right
    return count



with open('input.txt') as f:
    line = f.readline().split()
    text = ''
    for word in line:
        text += word

ans = find_three(text)

with open('output.txt', 'w') as d:
    d.write(str(ans))
```

Задача 3.

Так как вариант нахождения подстроки из первой задачи далеко неэффективен, нам нужно написать алгоритм Робина-Карпа, основанный на хешировании.

Все, что я могу сказать о хешировании: это больно. Очень. Хеши можно написать, через день забыть, как написал. В итоге это почему-то работающий черный ящик, в который мы кидаем хомячка и получаем корги на выходе…

В общем, в чем суть алгоритма. Мы знаем длину подстроки. Дальше мы находим все хеши строк этой длины. Но это медленно, если считать каждый хеш отдельно, ведь там такие размеры строк могут быть!!! Для этого есть оптимизация этого алгоритма. Оно называется скользящий хеш.

Чтобы посчитать хешем все подстроки данной длины, “ручками” надо найти только первую такую подстроку, а дальше считать по формулке. Мы, получается вычитаем предыдущий элемент, домножаем все на основание и прибавляем следующий, ну и все дружно делим на Большое Простое Число)))

Строки с одинаковыми хешами будут, с огромной вероятностью, одинаковыми, вот и все)))

А теперь поговорим о боли… Я, когда находила вхождения с одинаковым хешем, добавляла циферку не в массив, а сразу в строку, ведь все равно потом сдавать в файлик таким макаром…

Да? Ну чтож, смотрю я, тесты валятся по времени. Думаю, надо оптимайзить. Стала считать степени заранее, а не каждый раз, чтобы не перегружать питон возведением в степень, убрала проверку на то, одинаковы ли строки буквенно, если они одинаковы хешами, так как шанс ошибки крайне мал… Вся равно не проходит… Спустя два часа я решила делать сначала массив, а потом уже его печатать текстово.

Разница в 0.5 секунды! Пол секунды были скушаны строками… Я даже и не думала, что они так тормозят прогресс…… Поплакали, можно решать дальше))

```
with open('input.txt') as f:
    to_find = f.readline().strip()
    text = f.readline().strip()
# print(len(text))

mult = 1019
primary = 10 ** 9 + 7
deg = [1 % primary]
find_ln = len(to_find)
for i in range(1, find_ln):
    deg.append((1019 * deg[i - 1]) % primary)
```

```
def hash(line, primary):
    num = 0
    for i in range(len(line)):
        letter = ord(line[i])
        num += letter * deg[len(line) - i - 1]
        num = num % primary
    hashed = num % primary
    return hashed


def precompute_hashes(find_ln, line, mult, primary):
    hashes = [0 for i in range(len(line) - find_ln + 1)]
    first = line[:find_ln]
    hashes[0] = hash(first, primary)
    for i in range(len(hashes) - 1):
        old_hash = hashes[i]
        old_symb = ord(line[i])
        b_degreed = deg[find_ln - 1]
        new_symb = ord(line[i + find_ln])
        all_old = ((old_hash - old_symb * b_degreed) * mult) % primary
        new_hash = all_old + new_symb
        hashes[i + 1] = new_hash
    return hashes


def RobinKarp(to_find, line):
    cnt = 0
    ans = []
    primary = 10 ** 9 + 7
    mult = 1019
    find_ln = len(to_find)
    hashed_find = hash(to_find, primary)
    hashed_all = precompute_hashes(find_ln, line, mult, primary)
    for i in range(len(hashed_all)):
        if hashed_all[i] != hashed_find:
            continue
        else:
            ans.append(str(i + 1))
            cnt += 1
    return (str(cnt), ' '.join(ans))



if len(to_find) > len(text):
    ans = 0
else:
    ans = RobinKarp(to_find, text)
```

```
with open('output.txt', 'w') as d:
    if len(to_find) > len(text):
        d.write(str(ans))
    else:
        d.write(str(ans[0]) + '\n')
        d.write(ans[1])
```

Задача 4.

Эту задачу я села решать сразу после третьей, часов в 10 вечера. Написала, она падает. Падает и все. При этом хеши считаются, вроде все адекватно (но нет).

В итоге, эту задачу я решила к 9 вечера следующего дня, при этом еще успела решить все остальные в этом промежутке. Хуже этой задачи, мне кажется, нет ничего на свете. Если в прошлый раз мы учились квакать, то тут бы надо не квакнуться пока пишешь…

Я даже специально для отчета оставила микрокомметнарий в процессе решения для осознания всей боли…

“Пятый час сижу с этой задачей. Плачу. Не понимаю как это работает, а решить надо… вот бы мне внезапно пришло понимание того где что нужно умножать возводить в степень и вычитать. Я всю ночь сидела с задачами в коворкинге и все что мне сейчас хочется это плакать. Кристина принесла мне пледик, хоть что-то хорошее в этой жизни…”

Ладно, достаточно печали, надо отчет отчетить. Берем, получается, считаем первую подстроку строки. Дальше считаем на основе первого хеща все остальные подстроки, увеличивая количество буковок. Получается что-то типа скользящий односторонний хеш. Чтобы нат хеш подстроки, нам нужно вычесть из префикса, оканчивающейся на последнюю букву подстроки, префикс, заканчивающийся на одну букву раньше, чем первый символ подстроки. Че-то там перемножить, че-то там возвести в степень, и методом волшебного тыка подбираем комбинацию, с которой это все начинает волшебным образом работать!

Очень не люблю эту задачку, я прям в расстройстве с нее, но решила.

Кстати от того, что решила ее, никакого удовлетворения. Хотя обычно я решенным задачам прям радуюсь.

```
MAX_SIZE = 2 * 10 ** 6
MULT = 1019
PRIMARY = 10 ** 9 + 7


deg = [1 % PRIMARY]
find_ln = MAX_SIZE
```

```
for i in range(1, find_ln):
    deg.append((1019 * deg[i - 1]) % PRIMARY)


def hash(line):
    ln_line = len(line)
    hashed_pefixes = [0 for i in range(ln_line + 1)]
    hashed_pefixes[0] = 0
    for i in range(1, ln_line + 1):
        hashed_pefixes[i] = (hashed_pefixes[i - 1] + ord(line[i - 1]) * deg[i]) %
PRIMARY
    return hashed_pefixes


def whether_equal(first, second, ln, full_ln, hashed):
    first_hashed = ((hashed[first + ln] - hashed[first]) * deg[full_ln - first]) %
PRIMARY
    second_hashed = ((hashed[second + ln] - hashed[second]) * deg[full_ln - second])
% PRIMARY
    if first_hashed == second_hashed:
        return 'YES'
    return 'NO'


with open('input_4.txt') as f:
    line = f.readline().strip()
    hashed = hash(line)
    num_checks = int(f.readline())
    for line in f.readlines():
        first, second, ln = list(map(int, line.split()))
        print(whether_equal(first, second, ln, len(line), hashed))
```

Задача 5.

Одна из тех задач, что я решала для морального успокоения. Описание есть полностью в лекции. Делается это все чудо за линию за счет того, что у нас есть два указателя, которые мы передвигаем.

Получается, мы либо увеличиваем значение j, если строка от j равна строке от i-1, а если нет, то j смещаем на начало строки и сравниваем заново. И так до конца.

Несложно и быстро, причем очень быстро)))

```
import time
t_start = time.perf_counter()

def prefix(line):
    stripped_line = list(line)
    len_line = len(line)
```

```
    pref = [0 for i in range(len_line)]
    j = 0

    for i in range(2, len_line + 1):
        while j > 0 and stripped_line[j] != stripped_line[i - 1]:
            j = pref[j - 1]
        if stripped_line[j] == stripped_line[i - 1]:
            j += 1
        pref[i - 1] = j

    return pref




with open('input_5.txt') as f:
    line = f.readline().strip()

ans = prefix(line)

with open('output_5.txt', 'w') as d:
    for elem in ans:
        d.write(str(elem) + ' ')

print("Время работы: %s секунд " % (time.perf_counter() - t_start))
```

Задача 6.

И ещеее одна задача, которую я решала для морального успокоения. Чем-то похожа на префикс, но не префикс.

Вообще z-функция показывает для каждого элемента с индексом i, какой длины подстрока, начинающаяся с этого элемента, может быть префиксом всей строки.

Мем какой-то конечно, но почему бы и нет, если все так просто…

Используем, получается, z-блоки. Z-блок начинается с I и длина у него z[i]. Конец и начало блока храним в R (right) и L (left). В зависимости от того, меньше i чем right или нет мы либо бегаем внутри отрезка, либо снаружи, добавляя значения ячейкам когда видим, что дальше идет подходящее значение. Так пробегаемся по всей строке, и в принципе все)))

```
def zed(line):
  zedded = [0] * len(line)
  L, R = 0, 0
  for i in range(1, len(line)):
    zedded[i] = max(0, min(zedded[i - L], R - i))
    while i + zedded[i] < len(line) and line[zedded[i]] == line[i + zedded[i]]:
```

```
      zedded[i] += 1
    if i + zedded[i] > R:
      L, R = i, i + zedded[i]
  return zedded



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

ans = zed(line)[1:]

with open('output.txt', 'w') as d:
    for elem in ans:
        d.write(str(elem) + ' ')
```

Задача 7.

Я бы сказала, что эта задача Больше красивая, чем сложная. Бинарный поиск очень хорошо влезает в эту задачу. С самими хешами нет никаких проблем, потому что они такие же, как в старых задачах, а вот binary search тут прекрасен. Мы делим с самого начала длину строки пополам, и ищем все подстроки длиной в половину строки. Если находим, то записываем координаты и идем искать больше, если нет – то меньше. Опять пополамим и ищем так пока не дойдем то ограничителей.

Очень красивая задачка, мне прям понравилась 😊

```
def hash(line, primary):
    global deg
    num = 0
    for i in range(len(line)):
        letter = ord(line[i])
        num += letter * deg[len(line) - i - 1]
        num = num % primary
    hashed = num % primary
    return hashed


def precompute_hashes(find_ln, line, mult, primary):
    global deg
    hash_dict = {}
    hashes = [0 for i in range(len(line) - find_ln + 1)]
    first = line[:find_ln]
    hashes[0] = hash(first, primary)
    hash_dict[hashes[0]] = 0
    for i in range(len(hashes) - 1):
        old_hash = hashes[i]
```

```
        old_symb = ord(line[i])
        b_degreed = deg[find_ln - 1]
        new_symb = ord(line[i + find_ln])
        all_old = ((old_hash - old_symb * b_degreed) * mult) % primary
        new_hash = all_old + new_symb
        hashes[i + 1] = new_hash
        if new_hash in hash_dict.keys():
            continue
        else:
            hash_dict[new_hash] = i + 1
    return hash_dict


def binary_search(smaller, bigger, old_dividor, very_old_dividor, border, direction):
    global CURRENT_BIGGEST
    if old_dividor == very_old_dividor:
        return
    if direction == 'right':
        if (very_old_dividor + old_dividor) % 2 != 0:
            new_splitline = (very_old_dividor + old_dividor) // 2 + 1
        else:
            new_splitline = (very_old_dividor + old_dividor) // 2
    else:
        if (very_old_dividor - (very_old_dividor - old_dividor)) % 2 != 0:
            new_splitline = (very_old_dividor - (very_old_dividor - old_dividor)) //
2 + 1
        else:
            new_splitline = (very_old_dividor - (very_old_dividor - old_dividor)) //
2

    if new_splitline == 0 or new_splitline == border:
        return
    all_smol = precompute_hashes(new_splitline, smaller, mult, primary)
    all_big = precompute_hashes(new_splitline, bigger, mult, primary)
    found = (find_same(all_smol, all_big))
    if found[0] == True:
        CURRENT_BIGGEST = [found[1], found[2], new_splitline]
        direction = 'right'
    else:
        direction = 'left'
    binary_search(smaller, bigger, new_splitline, old_dividor, border, direction)



def find_same(all_smol, all_big):
    for key in all_smol.keys():
        if key in all_big.keys():
            id_1, id_2 = all_smol[key], all_big[key]
            return [True, id_1, id_2]
    return [False]
```

```
def find_biggest_subline(first, second):
    global mult, primary, CURRENT_BIGGEST
    if len(first) < len(second):
        smaller, bigger = first, second
    else:
        smaller, bigger = second, first
    tiny_len = len(smaller)
    if tiny_len % 2 != 0:
        first_splitter = tiny_len // 2 + 1
    else:
        first_splitter = tiny_len // 2
    all_smol = precompute_hashes(first_splitter, smaller, mult, primary)
    all_big = precompute_hashes(first_splitter, bigger, mult, primary)
    first_found = (find_same(all_smol, all_big))
    if first_found[0] == True:
        CURRENT_BIGGEST = [first_found[1], first_found[2], first_splitter]
        direction = 'right'
    else:
        direction = 'left'
    binary_search(smaller, bigger, first_splitter, tiny_len, tiny_len, direction)
    return




mult = 1019
primary = 10 ** 9 + 7

with open('input_7.txt') as f:
    for line in f.readlines():
        CURRENT_BIGGEST = [0, 0, 0]
        first, second = line.split()
        len_one, len_two = len(first), len(second)
        find_ln = max(len_one, len_two)
        deg = [1 % primary]
        for i in range(1, find_ln):
            deg.append((1019 * deg[i - 1]) % primary)
        find_biggest_subline(first, second)
        print(CURRENT_BIGGEST)
```

Задача Сдвиг Текста.

Пока я писала сам отчет, я поняла, что можно было бы решить ее с помощью z-функции, но мне уже некогда писать, да и тем более это решение хоть и наивное, но тоже зашло в проверку. Тут мы буквально поочередно берем букву с конца и ставим в начало строки, а потом сравниваем с тем, что получилось в условии задачи. Считаем количество изменений, потом их выводим. Если получить такую строку такими вращениями не получается, то значит нужно вывести -1.

Z-функцией бы было быстрее, но я даже не уверена что оно работает, но мне кажется работает))))

```
def cycle(initial, final):
    if initial == final:
        return 0
    len_str = len(initial)
    temp = initial
    for i in range(len_str):
        temp = temp[-1:] + temp[:-1]
        if temp == final:
            return i + 1
    return -1




with open('input_cycle.txt') as f:
    initial = f.readline().strip()
    final = f.readline().strip()

ans = cycle(initial, final)

with open('output_cycle.txt', 'w') as d:
    d.write(str(ans))
```

Задача Поиск Подстроки.

Тут все просто и сердито. Берем р-функцию и с омощью нее решаем. Это, вроде как, самый быстрый вариант. Мы берем строку, которую ищем, после нее ставим разделитель # и саму строку в которой будем искать.

После прохождения функцией всего массива, то части, описывающей все после # будет храниться информация о том, где есть нужные нам вхождения. Так как мы знаем длину искомой нам строки, то мы буквально можем посмотреть, где лежит эта длина, и это и будут наши индексы))) Только нам нужно вычесть длину искомой подстроки, а то будет неправильно)

```
def find_subline(line, fin_len):
    stripped_line = list(line)
    len_line = len(line)
    pref = [0 for i in range(len_line)]
    j = 0

    for i in range(2, len_line + 1):
        while j > 0 and stripped_line[j] != stripped_line[i - 1]:
            j = pref[j - 1]
        if stripped_line[j] == stripped_line[i - 1]:
```

```
            j += 1
        pref[i - 1] = j
    ans = ''
    for i in range(len(pref)):
        if pref[i] == fin_len:
            ans += str(i - 2 * fin_len) + ' '
    return ans



with open('input.txt') as f:
    line = f.readline().strip()
    to_find = f.readline().strip()
    fin_len = len(to_find)
    zhaba = to_find + '#' + line # zhaba is a mix of the line we need to find and the
line itself, separated by #
    ans = find_subline(zhaba, fin_len)

with open('output.txt', 'w') as d:
    d.write(ans.strip())
```

Описание проведенных тестов.

1

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		0.078	9166848	20003	48889
1	OK	0.015	9035776	14	6
2	OK	0.031	9039872	6	4
3	OK	0.015	9043968	6	3
4	OK	0.031	9039872	7	6
5	OK	0.015	8982528	7	3
6	OK	0.031	8990720	9	6
7	OK	0.062	8978432	10	4
8	OK	0.015	8978432	3004	3
9	OK	0.015	9015296	3028	6
10	OK	0.031	9003008	2656	428
11	OK	0.015	8998912	2005	8894
12	OK	0.015	8998912	4003	6
13	OK	0.031	8970240	3004	3
14	OK	0.031	9048064	2252	1849
15	OK	0.031	9048064	2021	185
16	OK	0.031	9023488	2008	8883
17	OK	0.031	8998912	3004	3903
18	OK	0.031	8978432	2670	3
19	OK	0.031	8966144	3028	6
20	OK	0.046	8994816	2404	690
21	OK	0.031	9007104	2005	8898
22	OK	0.031	9015296	4003	6
85	OK	0.031	9048064	8022	811
86	OK	0.015	9158656	8008	38883
87	OK	0.031	9125888	12004	18903
88	OK	0.031	9117696	15004	3
89	OK	0.015	9056256	11028	16
90	OK	0.031	9056256	10925	664
91	OK	0.031	9138176	10005	48884
92	OK	0.015	9035776	20003	6
93	OK	0.031	9064448	13337	3
94	OK	0.031	9068544	12504	8255
95	OK	0.031	9048064	10020	1021
96	OK	0.046	9138176	10008	48883
97	OK	0.046	9158656	15004	23903
98	OK	0.031	9043968	15004	3
99	OK	0.031	8990720	11028	16
100	OK	0.031	9048064	11004	497
101	OK	0.031	9150464	10005	48889
102	OK	0.015	9019392	20003	6
103	OK	0.015	9056256	13337	3
104	OK	0.078	9031680	10912	10925
105	OK	0.031	9043968	10015	2041
106	OK	0.046	9129984	10008	48883
107	OK	0.031	9138176	15004	23903

2

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.578	10158080	300002	16
1	OK	0.015	9048064	10	1
2	OK	0.031	9056256	34	3
3	OK	0.031	9052160	5	1
4	OK	0.031	9060352	6	1
5	OK	0.031	9039872	7	1
6	OK	0.031	9039872	9	2
7	OK	0.031	9048064	7	1
8	OK	0.031	9048064	7	1
9	OK	0.031	9097216	13	2
10	OK	0.000	9113600	202	6
11	OK	0.015	9035776	202	6
12	OK	0.046	9060352	202	6
13	OK	0.031	9023488	202	6
14	OK	0.015	9056256	202	5
15	OK	0.015	9089024	202	5
16	OK	0.031	9052160	202	5
17	OK	0.031	9068544	202	7
18	OK	0.046	9097216	5002	11
19	OK	0.031	9064448	5002	11
20	OK	0.015	8982528	5002	11
21	OK	0.046	9101312	5002	11
22	OK	0.046	9101312	5002	11
23	OK	0.046	9072640	5002	11
24	OK	0.046	9093120	5002	11
25	OK	0.046	9043968	5002	11
26	OK	0.046	9048064	5002	11
27	OK	0.046	9076736	5002	11
28	OK	0.046	9084928	5002	9
29	OK	0.062	9035776	5002	9
30	OK	0.046	9056256	5002	9
31	OK	0.046	9113600	5002	9
32	OK	0.046	9134080	5002	9
33	OK	1.250	9965568	300002	16
34	OK	1.250	9945088	300002	16
35	OK	1.265	9998336	300002	16
36	OK	1.218	9908224	300002	16
37	OK	1.265	10014720	300002	16
38	OK	1.281	9940992	300002	16
39	OK	1.546	9990144	300002	15
40	OK	1.500	9961472	300002	15
41	OK	1.484	9945088	300002	15
42	OK	1.484	9998336	300002	15
43	OK	1.484	10149888	300002	15
44	OK	1.453	10137600	300002	15
45	OK	1.484	10117120	300002	15
46	OK	1.453	10158080	300002	15
47	OK	1.500	10100736	300002	15
48	OK	1.500	9928704	300002	15
49	OK	1.531	10006528	300002	15
50	OK	1.578	9953280	300002	15
51	OK	1.468	9981952	300002	15
52	OK	1.421	9998336	300002	15

3

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла	Группа тестов
Max		1.140	132362240	2000003	6888904	
1	OK	0.031	9179136	14	6	
2	OK	0.031	9158656	6	4	
3	OK	0.015	9199616	6	3	
4	OK	0.031	9207808	7	6	0
5	OK	0.031	9207808	7	1	
6	OK	0.015	9117696	9	6	
7	OK	0.015	9166848	10	4	
8	OK	0.531	37199872	900004	3	
9	OK	0.484	36753408	601028	1334	
10	OK	0.500	35528704	799942	47137	5
11	OK	0.562	64835584	600005	4088889	
12	OK	0.656	39972864	1200003	6	
13	OK	0.546	37298176	900004	3	
14	OK	0.531	45010944	720004	764368	
15	OK	0.468	36745216	601009	1334	5
16	OK	0.671	84115456	600008	4088881	
17	OK	0.640	61366272	900004	1988909	
18	OK	0.515	35573760	800004	3	

86	OK	1.031	120287232	900008	6188825	
87	OK	0.984	85000192	1350004	3038909	
88	OK	0.843	53145600	1333337	3	
89	OK	0.765	52727808	1001028	2249	
90	OK	0.781	52502528	1125004	119668	5
91	OK	0.890	100040704	1000005	6888891	
92	OK	1.046	59498496	2000003	6	
93	OK	0.859	53174272	1333337	3	
94	OK	0.890	69140480	1111112	1445693	
95	OK	0.718	52727808	1001026	2243	5
96	OK	1.125	132358144	1000008	6888853	
97	OK	1.125	93233152	1500004	3388909	
98	OK	0.859	53174272	1333337	3	
99	OK	0.734	52629504	1001028	2249	
100	OK	0.843	52838400	1249930	62604	5
101	OK	0.921	100069376	1000005	6888904	
102	OK	1.093	59482112	2000003	6	
103	OK	0.828	53207040	1333337	3	
104	OK	0.921	66543616	1333336	1131788	
105	OK	0.734	52404224	1001529	1120	5
106	OK	1.140	132362240	1000008	6888853	
107	OK	1.140	93171712	1500004	3388909	

4

Input	Output
aaabaaab 5 0 4 4 0 4 3 0 4 2 0 4 1 3 4 1	YES YES YES YES NO
trololo 4 0 0 7 2 4 3 3 5 1 1 3 2	YES YES YES NO

5

Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.218	60608512	1000002	6888890
1	OK	0.015	9109504	8	12
2	OK	0.031	9113600	9	14
3	OK	0.015	9117696	3	2
4	OK	0.015	9080832	4	4
5	OK	0.000	9080832	4	4
6	OK	0.031	9150464	12	20
7	OK	0.015	9150464	12	20
8	OK	0.109	12193792	92672	185340
9	OK	0.156	15679488	99998	588846
10	OK	0.140	14610432	100002	561029
11	OK	0.203	15986688	176391	352778
12	OK	0.265	20566016	199994	1288779
13	OK	0.296	20602880	199992	1190917
14	OK	0.203	15974400	172864	345724
15	OK	0.375	24600576	300002	1988890
16	OK	0.390	24506368	300002	1716599
17	OK	0.265	15872000	249367	498730
18	OK	0.500	29696000	400002	2688855
19	OK	0.484	29630464	399998	2333024
20	OK	0.484	20914176	455342	910680
21	OK	0.609	35684352	499996	3388798
22	OK	0.625	35590144	499998	2875816
23	OK	0.546	22233088	505139	1010274
24	OK	0.718	41603072	600000	4088811
25	OK	0.734	41705472	600002	3977735
26	OK	0.578	24031232	539096	1078188
27	OK	0.859	46092288	699998	4788807
28	OK	0.875	45895680	700002	4566500
29	OK	0.546	22122496	492073	984142
30	OK	0.953	50376704	799997	5488765
31	OK	0.984	50282496	800002	3984418
32	OK	0.781	28626944	763540	1527076
33	OK	1.078	55353344	899994	6188744
34	OK	1.125	55422976	900002	5662193
35	OK	0.859	30515200	836144	1672284
36	OK	1.140	60358656	1000002	6888855
37	OK	1.203	60559360	1000002	6555315
38	OK	1.046	35430400	1000002	2000000
39	OK	1.171	60346368	1000002	6888890
40	OK	1.203	60555264	1000002	6888890
41	OK	1.218	60272640	1000002	6209773
42	OK	1.156	60608512	1000002	6888885
43	OK	1.171	60563456	1000002	6888870
44	OK	1.171	57376768	1000002	6388895
45	OK	1.171	60563456	1000002	6883895

6

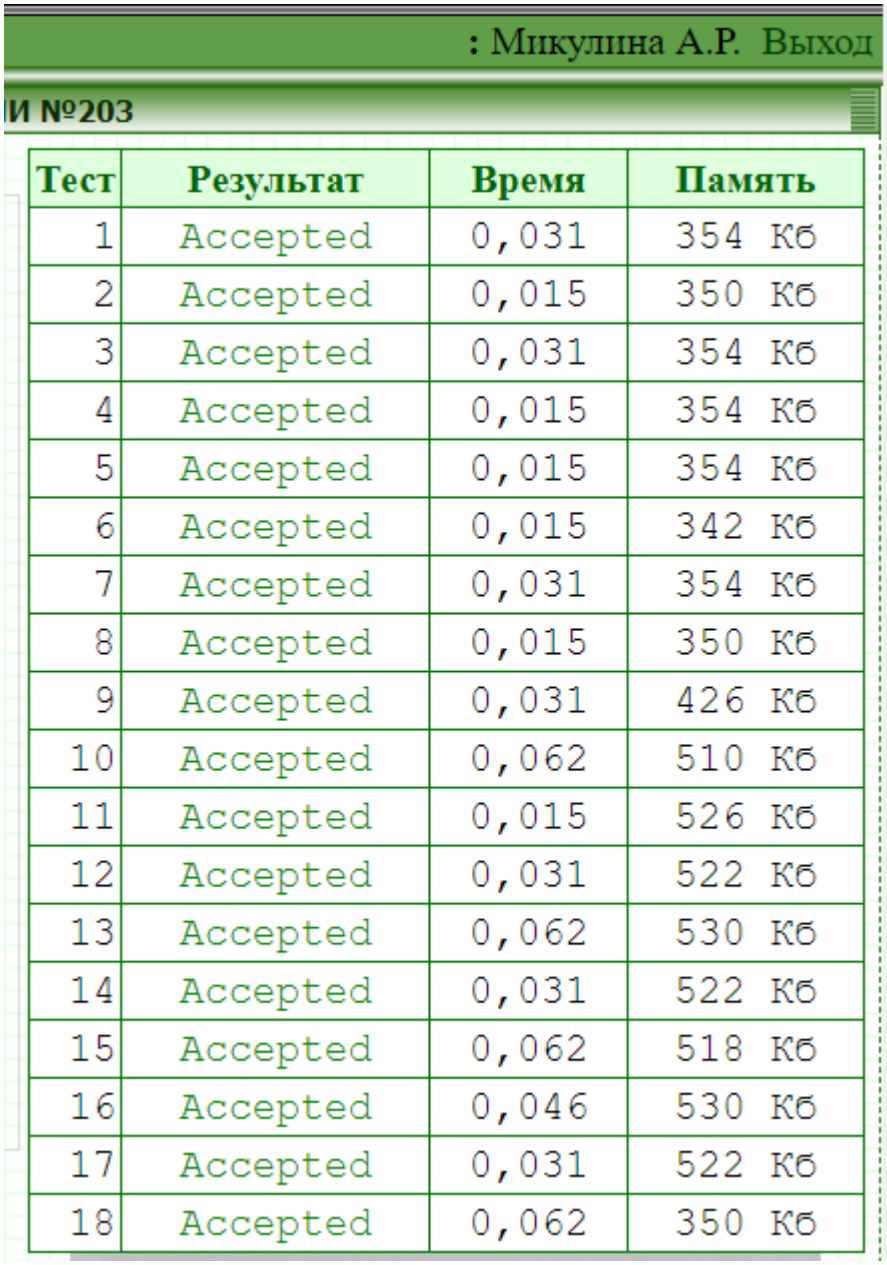
Верное решение!
Результаты работы Вашего решения

№ теста	Результат	Время, с	Память	Размер входного файла	Размер выходного файла
Max		1.859	59998208	1000002	6888888
1	OK	0.015	9023488	8	10
2	OK	0.031	8998912	9	12
3	OK	0.031	8966144	4	2
4	OK	0.015	8986624	4	2
5	OK	0.015	9011200	5	4
6	OK	0.031	8978432	12	18
7	OK	0.046	8990720	12	18
8	OK	0.171	10633216	92672	185338
9	OK	0.203	11358208	99998	264801
10	OK	0.203	11468800	100002	272211
11	OK	0.296	12050432	176391	352776
12	OK	0.421	12992512	199994	474050
13	OK	0.375	12881920	199992	456479
14	OK	0.296	12238848	172864	345722
15	OK	0.562	24055808	300002	1988888
16	OK	0.562	15921152	300002	786113
17	OK	0.406	13639680	249367	498728
18	OK	0.734	17723392	400002	1036108
19	OK	0.750	16797696	399998	885168
20	OK	0.718	17137664	455342	910678
21	OK	0.906	19353600	499996	1217153
22	OK	0.921	20029440	499998	1267986
23	OK	0.828	17952768	505139	1010272
24	OK	1.093	21352448	600000	1406340
25	OK	1.109	21843968	600002	1477776
26	OK	0.859	18882560	539096	1078186
27	OK	1.265	23605248	699998	1682393
28	OK	1.250	22851584	700002	1558331
29	OK	0.781	18272256	492073	984140
30	OK	1.453	25096192	799997	1804662
31	OK	1.515	29949952	800002	2196116
32	OK	1.218	23121920	763540	1527074
33	OK	1.625	27054080	899994	2030971
34	OK	1.640	32813056	900002	2765564
35	OK	1.359	24457216	836144	1672282
36	OK	1.859	31514624	1000002	2611108
37	OK	1.843	29118464	1000002	2227775
38	OK	1.593	27512832	1000002	1999998
39	OK	1.750	59998208	1000002	6888888
40	OK	1.734	59998208	1000002	6888888
41	OK	1.859	33923072	1000002	2841971
42	OK	1.781	43692032	1000002	4444443
43	OK	1.812	33955840	1000002	2977776
44	OK	1.828	27545600	1000002	2000043
45	OK	1.796	27590656	1000002	2004885

7

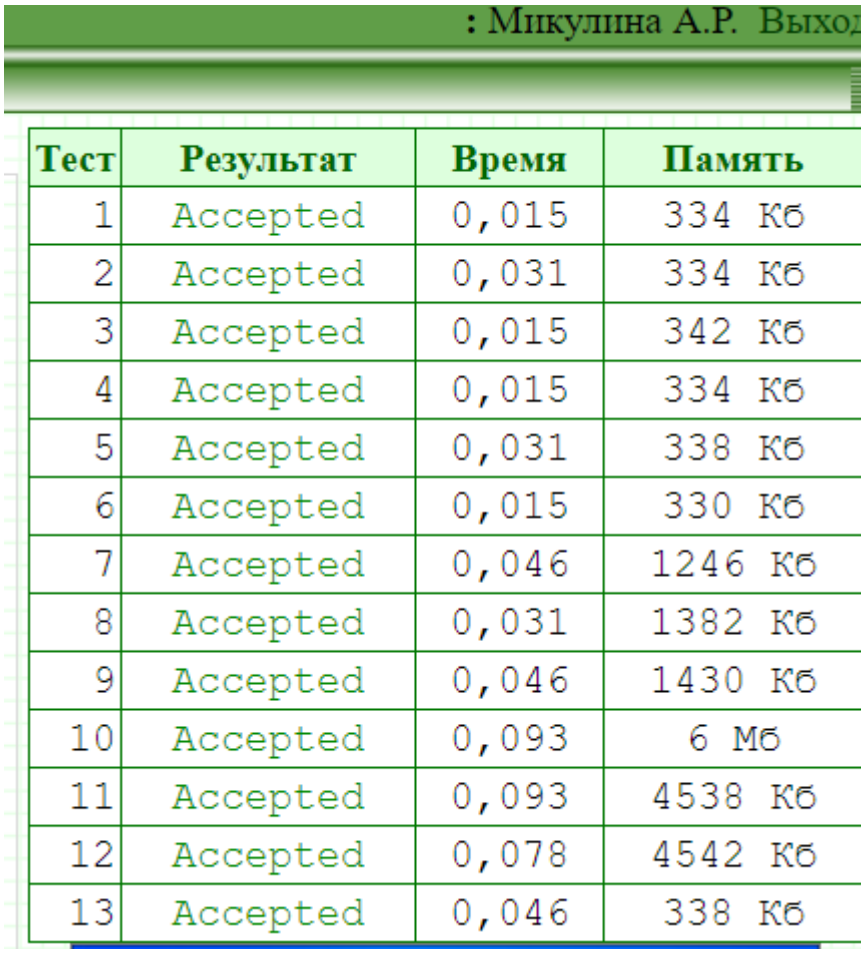
Input	Output
cool toolbox aaa bb aabaa babbaab aaaaaaaaaaaaaaaaaaaaaaaaaabbbbdddaa aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab fghjkytbnkiuygbnmkjhgfdfghjklkjhgfdfghhdsfysj cvbnmfjrdcvbjhgfdsfghjkjhgfdsdfvbnmbnbnbbbbbhhhfdd	[1, 1, 3] [0, 0, 0] [0, 4, 3] [0, 0, 25] [17, 22, 6]

Сдвиг текста



Тест	Результат	Время	Память
1	Accepted	0,031	354 Кб
2	Accepted	0,015	350 Кб
3	Accepted	0,031	354 Кб
4	Accepted	0,015	354 Кб
5	Accepted	0,015	354 Кб
6	Accepted	0,015	342 Кб
7	Accepted	0,031	354 Кб
8	Accepted	0,015	350 Кб
9	Accepted	0,031	426 Кб
10	Accepted	0,062	510 Кб
11	Accepted	0,015	526 Кб
12	Accepted	0,031	522 Кб
13	Accepted	0,062	530 Кб
14	Accepted	0,031	522 Кб
15	Accepted	0,062	518 Кб
16	Accepted	0,046	530 Кб
17	Accepted	0,031	522 Кб
18	Accepted	0,062	350 Кб

Поиск подстроки



Тест	Результат	Время	Память
1	Accepted	0,015	334 Кб
2	Accepted	0,031	334 Кб
3	Accepted	0,015	342 Кб
4	Accepted	0,015	334 Кб
5	Accepted	0,031	338 Кб
6	Accepted	0,015	330 Кб
7	Accepted	0,046	1246 Кб
8	Accepted	0,031	1382 Кб
9	Accepted	0,046	1430 Кб
10	Accepted	0,093	6 Мб
11	Accepted	0,093	4538 Кб
12	Accepted	0,078	4542 Кб
13	Accepted	0,046	338 Кб

Выводы по проделанной работе.

Такс-такс, утро перед экзаменом, я пишу отчет))) Могла бы, конечно, написать ночью, но это бы уже вторая ночь без сна, я б не выдержала. Лабу я в итоге решала два дня и полную ночь, или около того.

Вот и все)) Лаба очень красивая сама по себе, над ней бы подольше посидеть))) Хочу теперь сдвиг текста написать более эффективно, хотя бы через хеши лили мою идею с z-функцией. Прям хочется посидеть подумать))

Я была в шоке с того, как долго работает сложение строк, честно. А еще нет ничего в мире сложнее упрощения подсчета хешей :DDD

Очень грустно, что на этом все… А кому же еще я буду писать свои отчеты, зная, что они понравятся… Как же ломать мозг ночами над задачками… Радоваться каждый раз когда что-то начало работать, ходить на защиты… эх, я буду скучать, больше всего, именно по алгоритмам…