**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**КАФЕДРА МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №4**

**по дисциплине «Построение и Анализ Алгоритмов»**

**Тема: Алгоритм** **Кнута-Морриса-Пратта**

|  |  |  |
| --- | --- | --- |
| Студент гр. 1303 |  | Смирнов Д. Ю. |
| Преподаватель |  | Фирсов М. А. |

Санкт-Петербург

2023

**Цель работы.**

Реализовать алгоритм Кнута-Морриса-Пратта для нахождения вхождений одной подстроки в другую и для определения, является ли одна строка циклическим сдвигом другой.

**Задание.**

1. Реализуйте алгоритм КМП и с его помощью для заданных шаблона *P* (∣*P*∣ ≤ 15000) и текста *T* (∣*T*∣ ≤ 5000000) найдите все вхождения *P* в *T*.

Вход:

Первая строка - *P*

Вторая строка – *T*

Выход:

индексы начал вхождений *P* в *T*, разделенных запятой, если *P* не входит в *T*, то вывести −1

**Sample Input:**

ab

abab

**Sample Output:**

0,2

1. Заданы две строки *A* (∣*A*∣ ≤ 5000000) и *B* (∣*B*∣ ≤ 5000000). Определить, является ли *А* циклическим сдвигом *В* (это значит, что *А* и *В* имеют одинаковую длину и *А* состоит из суффикса *В*, склеенного с префиксом *В*). Например, defabc является циклическим сдвигом abcdef.

Вход:

Первая строка – *A*

Вторая строка – *B*

Выход:

Если *A* является циклическим сдвигом *B*, индекс начала строки *B* в *A*, иначе вывести −1. Если возможно несколько сдвигов вывести первый индекс.

**Sample Input:**

defabc

abcdef

**Sample Output:**

3

**Выполнение работы.**

Весь код программ представлен в приложении А.

**Описание алгоритма.**

Префикс-функция:

В функции создается список *result* и переменная *j* – она будет использоваться для отслеживания длины текущего совпадающего префикса и суффикса. Далее проходимся по индексам (*i*) строки, начиная с единицы, до длины паттерна. При совпадении символов *pattern[i]* и *pattern[j]* увеличиваем значение *j* на *1* и затем записываем значение *j* в список *result* по индексу *i*. В случае несовпадения символов, значение *j* обновляется на значение *result[j-1]* – это позволяет алгоритму вернуться к предыдущему совпадающему префиксу и суффиксу. Как только дошли до конца строки возвращаем список result в качестве результата функции.

Алгоритм Кнута-Морриса-Пратта:

Для искомой подстроки вызывается префикс-функция.

Инициализируем пустой список *result* для индексов вхождений образца в тексте, переменную *i*, отслеживающую длину совпадающего префикса образца и текста, *j* – индекс в тексте. Далее пока не дошли до конца текста, если текущие символы (*pattern[i]* и *text[j]*) совпадают, увеличиваем *j* и *i* на 1. В случае, если *i* равно длине подстроки, значит подстрока найдена в тексте - добавляем индекс начала вхождения подстроки в текст в список *result*, также устанавливаем *i* равным значению префикс-функции для последнего символа образца. Если же символы (*pattern[i]* и *text[j]*) не совпадают и *i* > 0, то уменьшаем *i* (устанавливаем *i* равным значению префикс-функции для предыдущего символа). В конце возвращаем список индексов вхождений образца в текст.

**Описание переменных и функций.**

Глобальные переменные:

* *DEBUG* – Флаг отвечающий за вывод дополнительной информации.

Функции:

* *prefixFunction(pattern: str) -> list* – Функция отвечает за выполнение описанной выше префикс-функции. Аргумент *pattern* – строка, для которой хотят построить префикс-функцию. Функция возвращает список из длин префиксов.
* (Для задания 1) *algorithmKMP(pattern: str, text: str) -> list –* Функция отвечает за описанный выше алгоритм КМП. Аргументы: *pattern* – строка, вхождения которой хотят найти в тексте, *text* – строка, в которой ищут все вхождения *pattern’а*. Функция возвращает список из индексов вхождения *pattern’a* в *text*.
* (Для задания 2) *algorithmKMP(pattern: str, text: str) -> list –* Функция отвечает за описанный выше алгоритм КМП, но после того как нашло первое вхождение выходит из функции, если дошли до конца строки и вхождение не было найдено, то возвращаем *-1*. Аргументы: *pattern* – строка, вхождения которой хотят найти в тексте, *text* – строка, в которой ищут все вхождения *pattern’а*. Функция возвращает список из индексов вхождения *pattern’a* в *text*.

Для решения 2-го задания алгоритм КМП выполняется для удвоенной строки *A*, так как если она является циклическим сдвигом строки *B*, то в ней точно будет содержаться подстрока *B*.

**Оценка сложности алгоритмов.**

Префикс-функция:

Для построения списка длин наибольших префиксов для каждой позиции этой строки, функция проходится по всей строке, из-за чего сложность будет O(*n*), где *n* – длина строки.

Алгоритм Кнута-Морриса-Пратта:

Для искомой подстроки выполняется префикс функция. После чего функция проходится по всему тексту из чего следует сложность O(*n* + *m*), где *n* – длина подстроки, *m* – длина текста. По памяти имеем 2 строки, из чего получаем O(*n* + *m*), где *n* – длина подстроки, *m* – длина текста.

**Тестирование.**

Пример вывода дополнительной информации для заданий 1-2, представлен на рисунках 1-2 соответственно.

Тестирование Префикс-функции:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Комментарий** |
| efefeftefe | 0 0 1 2 3 4 0 1 2 3 | Верно |
| abcde | 0 0 0 0 0 | Верно |
| aba | 0 0 1 | Верно |

Тестирование алгоритма Кнута-Морриса-Пратта:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Комментарий** |
| smth  nothing | -1 | Верно |
| aba  abababF | 0,2 | Верно |
| lip  liliput | 2 | Верно |
| abba  abba | 0 | Верно |
| avav  avavavava | 0,2,4 | Верно |
| abbb  vv | -1 | Верно |

Тестирование алгоритма определения циклического сдвига:

|  |  |  |
| --- | --- | --- |
| **Входные данные** | **Выходные данные** | **Комментарий** |
| defabc  abcdef | 3 | Верно |
| ab  abc | -1 | Верно |
| abc  ab | -1 | Верно |
| abc  bca | 1 | Верно |
| abc  cab | 2 | Верно |
| abc  abc | 0 | Верно |

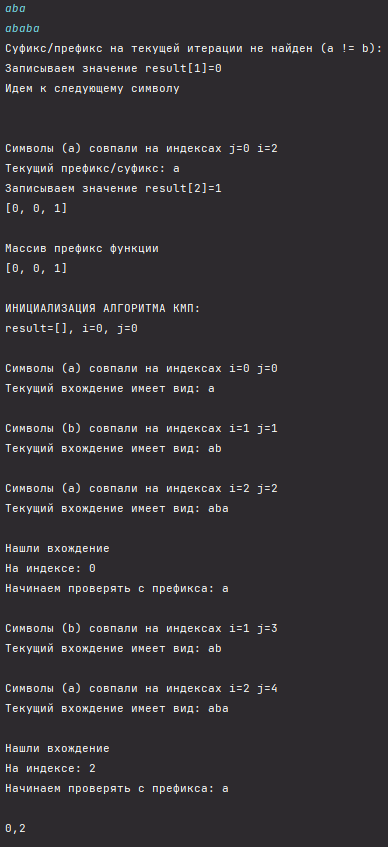
****

Рисунок 1- дополнительный вывод для задания 1

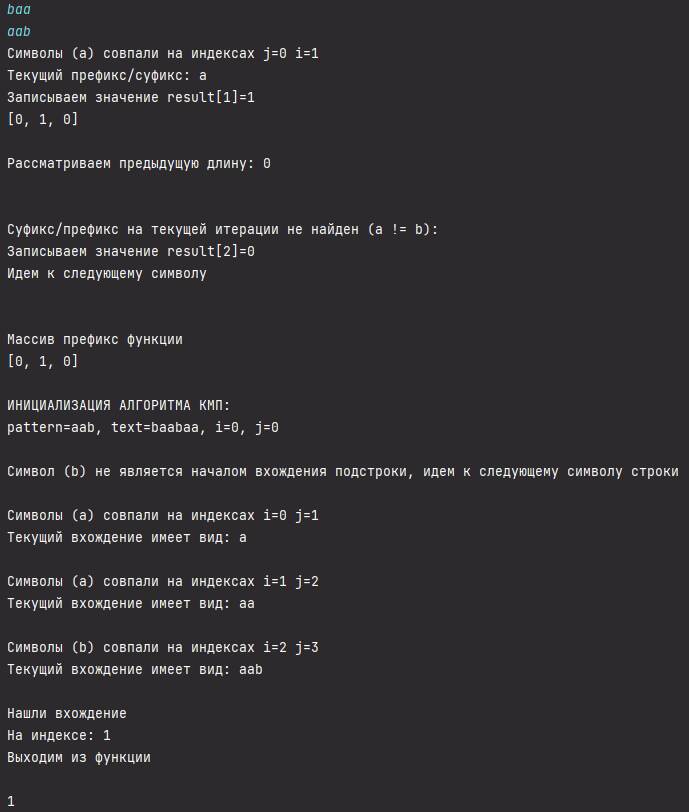
****

Рисунок 2- дополнительный вывод для задания 2

**Вывод.**

В ходе выполнения лабораторной работы изучен принцип работы префикс функции, а также алгоритм Кнута-Морриса-Пратта — поиска вхождения подстроки в строку. Разработаны две программы, первая: нахождения всех вхождений подстроки в строку, вторая: проверяющая является ли строка циклическим сдвигом другой.

**ПРИЛОЖЕНИЕ А**

**ИСХОДНЫЙ КОД ПРОГРАММ**

Название файла: KMPalgorithm.py

DEBUG = True  
  
  
def prefixFunction(*pattern*: *str*) -> *list*:  
 *"""  
 Функция принимает на вход строку и высчитывает для каждой подстроки [1…i]  
 значение префикс-функции. При этом на каждом шаге используется информация  
 о длине максимального префикса на предыдущем шаге, что ускоряет подсчёт.* ***:param*** *pattern: стока для который вычисляют префикс функцию* ***:return****: массив из элементов, обозначающих длину максимального префикса строки, совпадающего с её суффиксом  
 """* global DEBUG  
 result = [0] \* *len*(*pattern*)  
  
 j, i = 0, 1  
 while i < *len*(*pattern*):  
 if *pattern*[i] == *pattern*[j]: *# символы совпали* result[i] = j + 1 *# увеличиваем текущую длину* if DEBUG:  
 *print*(f"Символы ({*pattern*[i]}) совпали на индексах j={j} i={i}")  
 *print*(f"Текущий префикс/суфикс: {*pattern*[:j + 1]}")  
 *print*(f"Записываем значение result[{i}]={j + 1}")  
 *print*(result, *end*="\n\n")  
 j += 1 *# увеличиваем индексы* i += 1  
 else:  
 if j == 0:  
 if DEBUG:  
 *print*(f"Суфикс/префикс на текущей итерации не найден ({*pattern*[j]} != {*pattern*[i]}):")  
 *print*(f"Записываем значение result[{i}]=0")  
 *print*("Идем к следующему символу")  
 *print*("\n")  
 result[i] = 0 *# сопадений нет* i += 1 *# к следующему символу* else:  
 if DEBUG:  
 *print*(f"Рассматриваем предыдущую длину: {result[j - 1]}")  
 *print*("\n")  
 j = result[j - 1] *# возвращаемя с предыдущей длине* if DEBUG:  
 *print*("Массив префикс функции")  
 *print*(result, *end*="\n\n")  
 return result  
  
  
def algorithmKMP(*pattern*: *str*, *text*: *str*) -> *list*:  
 *"""  
 Посредством посимвольного сравнения двух строк определятся совпадение подстроки pattern  
 со строкой text. Если все символы совпали – вхождение найдено  
 и в результат записывается индекс вхождения.  
 Если первые символы не совпали, то первый символ pattern  
 сравнивается со вторым text и так далее до попадания. Если  
 символ pattern не совпадает с символом text, то следующее сравнение происходит  
 с символом подстроки P под индексом префикс-функции предыдущего символа.  
 Эти действия будут повторяться до тех пор, пока не будет  
 достигнут последний символ строки text.* ***:param*** *pattern: подстрока, вхождение которой ищут* ***:param*** *text: строка в которой ищут вхождение* ***:return****: список индексов вхождений  
 """* global DEBUG  
 prefArray = prefixFunction(*pattern*)  
 result = []  
 i = 0  
 j = 0  
 if DEBUG:  
 *print*("ИНИЦИАЛИЗАЦИЯ АЛГОРИТМА КМП:")  
 *print*(f"result={result}, i={i}, j={j}\n")  
 while j < *len*(*text*):  
 if *pattern*[i] == *text*[j]: *# символы совпали* if DEBUG:  
 *print*(f"Символы ({*pattern*[i]}) совпали на индексах i={i} j={j}")  
 *print*(f"Текущий вхождение имеет вид: {*pattern*[:i + 1]}\n")  
 i += 1 *# идем дальше* j += 1  
 if i == *len*(*pattern*): *# нашли вхождение* if DEBUG:  
 *print*("Нашли вхождение")  
 *print*(f"На индексе: {j - i}")  
 *print*(f"Начинаем проверять с префикса: {*pattern*[:prefArray[i - 1]]}\n")  
 result.append(j - i) *# сохраняем его индекс* i = prefArray[i - 1] *# востанавливаем индекс паттерна* elif j < *len*(*text*) and *pattern*[i] != *text*[j]: *# пока не дошли до конца и не совпали символы* if i == 0:  
 if DEBUG:  
 *print*(f"Символ ({*text*[j]}) не является началом вхождения подстроки, идем к следующему символу строки\n")  
 j += 1 *# идем к следующему* else:  
 if DEBUG:  
 *print*(f"Рассматриваем предыдущую длину: {prefArray[i - 1]}\n")  
 i = prefArray[i - 1] *# востанавливаем индекс паттерна* return result  
  
  
def main():  
 answer = algorithmKMP(*input*(), *input*()) *# Считываем входные данные и запускаем алгоритм КМП  
 print*(','.join(*map*(*str*, answer if answer else [-1]))) *# если нашли хоть один индекс выводим его, иначе -1*if \_\_name\_\_ == "\_\_main\_\_":  
 main()

Название файла: binaryHeap.py

class Heap:

def \_\_init\_\_(self, arr=None):

"Инициализация объектс класса"

if arr is None:

arr = []

self.\_\_heap = []

for el in arr:

self.insert(el)

@staticmethod

def getParent(index):

"Получение индекса родителя текущей вершины"

return (index - 1) // 2

@staticmethod

def getLeft(index):

"Получение индекса левого ребенка"

return 2 \* index + 1

@staticmethod

def getRight(index):

"Получение индекса правого ребенка"

return 2 \* index + 2

def \_\_siftUp(self, index):

"""

Просеивает вверх узел

:param index: индекс вершины которую хотят просеить вверх

:return: ничего возвращает

"""

if index < 0 or index >= len(self.\_\_heap):

return

parent = self.getParent(index)

while index and not self.\_\_heap[parent] < self.\_\_heap[index]:

self.\_\_heap[parent], self.\_\_heap[index] = self.\_\_heap[index], self.\_\_heap[parent]

index, parent = parent, self.getParent(index)

def \_\_siftDown(self, index):

"""

Просеивает вниз узел

:param index: индекс вершины которую хотят просеить вниз

:return: ничего возвращает

"""

if index < 0 or index >= len(self.\_\_heap):

return

minIndex = index

while True:

left, right = self.getLeft(index), self.getRight(index)

if right < len(self.\_\_heap) and self.\_\_heap[right] < self.\_\_heap[minIndex]:

minIndex = right

if left < len(self.\_\_heap) and self.\_\_heap[left] < self.\_\_heap[minIndex]:

minIndex = left

if minIndex == index:

return

else:

self.\_\_heap[index], self.\_\_heap[minIndex] = self.\_\_heap[minIndex], self.\_\_heap[index]

index = minIndex

def extract\_min(self):

"""

Достает минимальеый ставит максимальный на его место и просеивает вниз.

:return: возвращает минимальный элемент

"""

if not self.\_\_heap:

return

min\_element = self.\_\_heap[0]

self.\_\_heap[0] = self.\_\_heap[-1]

del self.\_\_heap[-1]

self.\_\_siftDown(0)

return min\_element

def insert(self, element):

"""

Добавляет элемент ставит в конец и просеивает вверх его.

:return: ничего возвращает

"""

self.\_\_heap.append(element)

self.\_\_siftUp(len(self.\_\_heap) - 1)

def size(self):

"""

:return: возвращает размер кучи

"""

return len(self.\_\_heap)

def \_\_repr\_\_(self):

representation = ""

for item in self.\_\_heap:

representation += '\t' + str(item) + "\n"

return representation

Название файла: cycleShift.py

DEBUG = True  
  
  
def prefixFunction(*pattern*: *str*) -> *list*:  
 *"""  
 Функция принимает на вход строку и высчитывает для каждой подстроки [1…i]  
 значение префикс-функции. При этом на каждом шаге используется информация  
 о длине максимального префикса на предыдущем шаге, что ускоряет подсчёт.* ***:param*** *pattern: стока для который вычисляют префикс функцию* ***:return****: массив из элементов, обозначающих длину максимального префикса строки, совпадающего с её суффиксом  
 """* global DEBUG  
 size = *len*(*pattern*)  
 result = [0] \* size  
  
 j, i = 0, 1  
 while i < size:  
 if *pattern*[i] == *pattern*[j]: *# символы совпали* result[i] = j + 1 *# увеличиваем текущую длину* if DEBUG:  
 *print*(f"Символы ({*pattern*[i]}) совпали на индексах j={j} i={i}")  
 *print*(f"Текущий префикс/суфикс: {*pattern*[:j + 1]}")  
 *print*(f"Записываем значение result[{i}]={j + 1}")  
 *print*(result, *end*="\n\n")  
 j += 1 *# увеличиваем индексы* i += 1  
 else:  
 if j == 0:  
 if DEBUG:  
 *print*(f"Суфикс/префикс на текущей итерации не найден ({*pattern*[j]} != {*pattern*[i]}):")  
 *print*(f"Записываем значение result[{i}]=0")  
 *print*("Идем к следующему символу")  
 *print*("\n")  
 result[i] = 0 *# сопадений нет* i += 1 *# к следующему символу* else:  
 if DEBUG:  
 *print*(f"Рассматриваем предыдущую длину: {result[j - 1]}")  
 *print*("\n")  
 j = result[j - 1] *# возвращаемя с предыдущей длине* if DEBUG:  
 *print*("Массив префикс функции")  
 *print*(result, *end*="\n\n")  
 return result  
  
  
def algorithmKMP(*pattern*: *str*, *text*: *str*) -> *int*:  
 *"""  
 Посредством посимвольного сравнения двух строк определятся совпадение подстроки pattern  
 со строкой text. Если все символы совпали – вхождение найдено  
 и в результат возвращается.  
 Если первые символы не совпали, то первый символ pattern  
 сравнивается со вторым text и так далее до попадания. Если  
 символ pattern не совпадает с символом text, то следующее сравнение происходит  
 с символом подстроки P под индексом префикс-функции предыдущего символа.  
 Эти действия будут повторяться до тех пор, пока не будет  
 достигнут последний символ строки text.* ***:param*** *pattern: подстрока, вхождение которой ищут* ***:param*** *text: строка в которой ищут вхождение* ***:return****: список индексов вхождений  
 """* global DEBUG  
 patternSize, textSize = *len*(*pattern*), *len*(*text*)  
 prefArray = prefixFunction(*pattern*)  
 i = 0  
 j = 0  
 if DEBUG:  
 *print*("ИНИЦИАЛИЗАЦИЯ АЛГОРИТМА КМП:")  
 *print*(f"pattern={*pattern*}, text={*text*}, i={i}, j={j}\n")  
 while j < textSize:  
 if *pattern*[i] == *text*[j]: *# символы совпали* if DEBUG:  
 *print*(f"Символы ({*pattern*[i]}) совпали на индексах i={i} j={j}")  
 *print*(f"Текущий вхождение имеет вид: {*pattern*[:i + 1]}\n")  
 i += 1 *# идем дальше* j += 1  
 if i == patternSize: *# нашли вхождение* if DEBUG:  
 *print*("Нашли вхождение")  
 *print*(f"На индексе: {j - i}")  
 *print*(f"Выходим из функции\n")  
 return j - i *# прекращаем поиск выводя индекс* else:  
 if i == 0:  
 if DEBUG:  
 *print*(f"Символ ({*text*[j]}) не является началом вхождения подстроки, идем к следующему символу строки\n")  
 j += 1 *# идем к следующему* else:  
 if DEBUG:  
 *print*(f"Нашли различные символы ({*pattern*[i]} != {*text*[j]}) и i({i})>0 из-за чего")  
 *print*(f"Рассматриваем предыдущую длину: {prefArray[i - 1]}\n")  
 i = prefArray[i - 1] *# востанавливаем индекс паттерна* if DEBUG:  
 *print*("text - не является циклической престановкой,")  
 *print*("возвращаем -1")  
 return -1  
  
  
def main():  
 *# считываем данные* text = *input*()  
 pattern = *input*()  
 *# если разная длина выводим сразу ответ* if *len*(text) != *len*(pattern):  
 *print*(-1)  
 return  
 *# иначе запускаем алгоритм КМП для склеиного текста  
 print*(algorithmKMP(pattern, text \* 2))  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()