МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №4

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Кнута-Морриса-Пратта

Студент гр. 8382 _____ Чирков С.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы вхождения подстроки в строку, а также разработать алгоритм проверки двух строк на циклический сдвиг.

Задание 1.

Реализуйте алгоритм КМП и с его помощью для заданных шаблона P($|P| \leq 15000$) и текста T($|T| \leq 5000000$) найдите все вхождения P в T.

Входные данные:

Первая строка – P

Вторая строка – T

Выходные данные:

Индексы начал вхождений P в T, разделенных запятой, если P не входит в T, то вывести -1.

Пример входных данных

aba

ababa

Пример выходных данных

0, 2

Задание 2.

Заданы две строки A ($|A| \leq 5000000$) и B$|B| \leq 5000000$). Определить, является ли A циклическим сдвигом B (это значит, что A и B имеют одинаковую длину и AA состоит из суффикса BB, склеенного с префиксом BB). Например, defabc является циклическим сдвигом abcdef.

Входные данные:

Первая строка – A

Вторая строка – B

Выходные данные:

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

Пример входных данных

defabc

abcdef

Пример выходных данных

3

Вариант дополнительного задания.

Вар. 1. Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

Описание алгоритма

Алгоритм Кнута — Морриса — Пратта (КМП-алгоритм) — эффективный алгоритм, осуществляющий поиск подстроки в строке. Время работы алгоритма линейно зависит от объёма входных данных, то есть разработать асимптотически более эффективный алгоритм невозможно. На вход алгоритма передается строка-образ, вхождения которой нужно найти, и строка-текст, в которой нужно найти вхождения. Алгоритм сначала вычисляет префикс-функцию строки-образа, с учетом запрашиваемого количества потоков. Далее сравнивается элемент строки-текста и элемент строки-образа. В случае их равенства, происходит увеличение индексов, указывающих на символ в строке-тексте и строке-образе. Затем после того как выявилось совпадение символов,

происходит проверка равенства обрабытаваемого индекса и длины строки-образа, если это верно, то значит, что вхождение найдено и происходит запись индекса начала вхождения в список с ответами ans.

В случае, когда элемент строки-текста и элемент строки-образа не совпали, то происходит проверка, не равен ли сейчас нулю индекс, указывающий на текущий элемент строки-образа. Если это верно, увеличиваем на единицу индекс, который указывает на символ в строке тексте. Иначе, если индекс не равен 0, то происходит перемещение позиции индекса при помощи префикс-функции. Алгоритм завершает работу по окончании строки-текста.

Описание функций и структур данных.

ans = [] – список для хранения ответа.

p = [] – список для хранения префикс-функции.

kmp(a,b,thread) – функция алгоритма Кнута-Морриса-Пратта, аргументы - шаблон, текст, количество потоков. Возвращаемого значения нет.

Тестирование

Ввод	**Вывод**
Вхождение подстроки	
ab abab 1	0,2
abc abcabcabcac 3	0,3,6
aba ababababa 2	0,2,4,6

aba abbba 5	слишком много потоков
Циклический сдвиг	
aba baa 2	2
abc accc 2	длины строк не равны
ab ba 3	слишком много потоков
abc def 2	-1

Тестирование с промежуточными выводами

Вхождение подстроки

```
введите шаблон
abababa
введите текст
abababababababa
введите количество потоков
5

построение префикс-функции

текущий поток 1
[0, 0, 0, 0, 0, 0, 0]

текущий поток 2
[0, 0, 0, 0, 0, 0, 0]

текущий поток 3
найдено новое значение p(2)=1
[0, 0, 1, 0, 0, 0, 0]

текущий поток 4
найдено новое значение p(3)=2
[0, 0, 1, 2, 0, 0, 0]

текущий поток 5
найдено новое значение p(4)=3
найдено новое значение p(5)=4
найдено новое значение p(6)=5

префикс-функция создана [0, 0, 1, 2, 3, 4, 5]

текущий поток 1

текущий поток 2

текущий поток 3

текущий поток 4
найдено новое решение 0

текущий поток 5
найдено новое решение 2
найдено новое решение 4
найдено новое решение 6

0,2,4,6
```

Циклический сдвиг

```
введите первую строку
abcbabc
введите вторую строку
bcbabca
введите количество потоков
2

построение префикс-функции

текущий поток 1
[0, 0, 0, 0, 0, 0, 0]

текущий поток 2
найдено новое значение p(4)=1
найдено новое значение p(5)=2
найдено новое значение p(6)=3

префикс-функция создана [0, 0, 0, 0, 1, 2, 3]

текущий поток 1

текущий поток 2
найдено новое решение 6

6
```

Сложность алгоритма

Сложность алгоритма по времени и по памяти: O (m + n), m – длина образа, n – длина текста.

Выводы.

В ходе работы был построен и анализирован алгоритм КМП на примере программ, решающих следующие задачи: нахождение индексов вхождения образца в строке и индекс циклического смещения одной строки в другой.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

Вхождение образца в строку

```
def kmp(a,b,thread): #алгоритм Кнута-Морриса-Пратта, аргументы - шаблон, текст, количество потоков
    n=len(a)
    m=len(b)
    ans=[]
    p=[0]*n
    j=0
    i=1
    print()
    print('построение префикс-функции')
    print()
    for ptr in range(thread-1): #цикл по шаблону, разделенный на thread потоков
        print('текущий поток',ptr+1)
        while i!=(ptr+1)*(round(n//thread)):     #инициализация префикс-функции
            if a[i]==a[j]:
                print('найдено новое значение p(',i,')=',j+1,sep='')
                p[i]=j+1
                i+=1
                j+=1
            elif j==0:
                p[i]=0
                i+=1
            else:
                j=p[j-1]
        print(p)
        print()
    print('текущий поток',thread)
    while i!=n:
        if a[i]==a[j]:
            print('найдено новое значение p(',i,')=',j+1,sep='')
            p[i]=j+1
            i+=1
            j+=1
        elif j==0:
            p[i]=0
            i+=1
        else:
            j=p[j-1]
    print()
    print('префикс-функция создана',p)
    k=0
    l=0
    print()
    for i in range(thread-1): #цикл по тексту, разделенный на thread потоков
        print('текущий поток',i+1)
        while k!=(i+1)*(round(m//thread)): #идем по round(len(b)/thread) символов каждый раз
            if b[k]==a[l]:            #совпал ли символ шаблон?
                if l==n-1:
                    ans.append(k-l)    #сохраняем, если шаблон пройден
                    print('найдено новое решение',ans[-1])
                    if l!=0:
                        l=p[l-1]
                    else:
                        k+=1
                else:
                    k+=1
```

```
                if l<n-1:
                    l+=1
            elif l==0:              #если в начале шаблона - следующий символ
                k+=1
            else:                   #иначе - переход по префикс-функции
                l=p[l-1]
        print()
    print('текущий поток',thread)
    while k!=m: #обработка оставшейся части слова (если не делится на равные части)
        if b[k]==a[l]:
            if l==n-1:
                ans.append(k-l)
                print('найдено новое решение',ans[-1])
                if l!=0:
                    l=p[l-1]
                else:
                    k+=1
            else:
                k+=1
                if l<n-1:
                    l+=1
        elif l==0:
            k+=1
        else:
            l=p[l-1]
    print()
    if ans==[]:
        print(-1)
    else:
        for x in ans: #вывод
            if x!=ans[len(ans)-1]:
                print(x,end=',')
            else:
                print(x)
print('введите шаблон')
a=input()
print('введите текст')
b=input()
print('введите количество потоков')
thread=int(input())
if thread>len(b):
    print('слишком много потоков')
else:
    kmp(a,b,thread)
```

Циклический сдвиг

```
def kmp(a,b,thread): #алгоритм Кнута-Морриса-Пратта, аргументы - шаблон, текст, количество потоков
    n=len(a)
    b+=b
    m=len(b)
    ans=[]
    p=[0]*n
    j=0
    i=1
    print()
    print('построение префикс-функции')
    print()
    for ptr in range(thread-1): #цикл по шаблону, разделенный на thread потоков
        print('текущий поток',ptr+1)
        while i!=(ptr+1)*(round(n//thread)):     #инициализация префикс-функции
            if a[i]==a[j]:
                print('найдено новое значение p(',i,')=',j+1,sep='')
                p[i]=j+1
                i+=1
                j+=1
            elif j==0:
                p[i]=0
                i+=1
            else:
                j=p[j-1]
        print(p)
        print()
    print('текущий поток',thread)
    while i!=n:
        if a[i]==a[j]:
            print('найдено новое значение p(',i,')=',j+1,sep='')
            p[i]=j+1
            i+=1
            j+=1
        elif j==0:
            p[i]=0
            i+=1
        else:
            j=p[j-1]
    print()
    print('префикс-функция создана',p)
    k=0
    l=0
    print()
    for i in range(thread-1): #цикл по тексту, разделенный на thread потоков
        print('текущий поток',i+1)
        while k!=(i+1)*(round(m//thread)): #идем по round(len(b)/thread) символов каждый раз
            if b[k]==a[l]:            #совпал ли символ шаблон?
                if l==n-1:
                    ans.append(k-l)    #сохраняем, если шаблон пройден
                    print('найдено новое решение',ans[-1])
                    if l!=0:
                        l=p[l-1]
                    else:
                        k+=1
                else:
                    k+=1
```

```
                if l<n-1:
                    l+=1
            elif l==0:              #если в начале шаблона - следующий символ
                k+=1
            else:                   #иначе - переход по префикс-функции
                l=p[l-1]
        print()
    print('текущий поток',thread)
    while k!=m: #обработка оставшейся части слова (если не делится на равные части)
        if b[k]==a[l]:
            if l==n-1:
                ans.append(k-l)
                print('найдено новое решение',ans[-1])
                if l!=0:
                    l=p[l-1]
                else:
                    k+=1
            else:
                k+=1
                if l<n-1:
                    l+=1
        elif l==0:
            k+=1
        else:
            l=p[l-1]
    print()
    if ans==[]: #вывод
        print(-1)
    else:
        print(ans[0])
print('введите первую строку')
a=input()
print('введите вторую строку')
b=input()
print('введите количество потоков')
thread=int(input())
if thread>len(b):
    print('слишком много потоков')
elif len(a)!=len(b):
    print('длины строк не равны')
else:
    kmp(a,b,thread)
```