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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 8382 _____ Чирков С.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Разработать программу для точного поиска вхождения набора образцов в тесте с использованием алгоритма Ахо-Корасик.

Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст ($1 \leq T \leq 100000$). Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p1, \ldots, pn\}$, $1 \leq |pi| \leq 75$. Все строки содержат символы из алфавита {A,C,G,T,N}

Выход:

Все вхождения образцов из P в T. Каждое вхождение образца в текст представить в виде двух чисел - i p, Где i - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером p (нумерация образцов начинается с 1). Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.

Sample Input:

NTAG
3
TAGT
TAG
T

Sample Output:

2 2
2 3.

Задание 2.

Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

В шаблоне встречается специальный символ, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

Например, образец ab??c? с джокером ? встречается дважды в тексте xabvccbababcax.

Символ джокер не входит в алфавит, символы которого используются в T. Каждый джокер соответствует одному символу, а не подстроке неопределённой длины. В шаблон входит хотя бы один символ не джокер, т.е. шаблоны вида ??? недопустимы.

Все строки содержат символы из алфавита {A,C,G,T,N}.

Вход:

Текст (T,$1 \leq |T| \leq 100000$)

Шаблон (P,$1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер). Номера должны выводиться в порядке возрастания.

Sample Input:

ACTANCA

A$$A$

$

Sample Output:

1

Вариант дополнительного задания.

Вариант 3. Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

Суффиксная ссылка — это ссылка на узел, соответствующий самому длинному суффиксу, который не заводит бор в тупик.

Для корневого узла суффиксная ссылка — петля. Для остальных правило таково: если последний распознанный символ — x, то осуществляется обход по суффиксной ссылке родителя, если оттуда есть дуга, нагруженная символом x, суффиксная ссылка направляется в тот узел, куда эта дуга ведёт. Иначе — алгоритм проходит по суффиксной ссылке ещё раз, пока либо не придет в корень, либо не найдёт дугу, нагруженную символом x.

Конечные ссылки связаны с суффиксными: конечная ссылка ведёт на ближайшую по суффиксным ссылкам конечную вершину; обход по конечным ссылкам даёт все совпавшие строки.

Описание алгоритма

Алгоритм Ахо-Корасик позволяет за линейное время найти все позиции вхождений набора образцов в тексте. Из образцов строится детерминированный автомат распознаватель (бор) с суффиксными ссылками. После построения бора происходит итерация по символам текста и, если из текущего состояния автомата можно перейти дальше по текущему символу текста, то происходит переход по ребру, иначе – по суффиксной ссылке. Если на очередной итерации по тексту было достигнуто конечное состояние автомата – данная позиция означает окончание вхождение данного образца.

Алгоритм Ахо-Корасик может быть использован для поиска по тексту вхождений шаблона со специальным символом (джокером), означающим любой символ алфавита. Для этого необходимо разбить шаблон по джокеру на подстроки, запомнить для каждой подстроки её смещение в исходном шаблоне. Затем найти индексы вхождения всех подстрок в тексте и, учитывая смещение подстроки относительно изначального шаблона, увеличить значение индекса

текста на один. Если в списке таких значений будет найдено число, равное количеству подстрок, этот индекс подходит для шаблона.

Описание функций и структур данных.

Для реализации узла бора написан класс AhoNode. Его поля-переменные: goto = {} – словарь, указывающий на возможные пути из данного узла, out - шаблоны, удовлетворяющие данной вершине(включая те, на которые указывают конечные ссылки), fail - суффиксная ссылка.

aho_create_bor(patterns) – функция построения бора вместе с суффиксными и конечными ссылками. Аргумент – массив строк. Возвращает корень бора.

def aho_find_all(s, root) – функция с реализацией алгоритма Ахо-Корасик. Аргументы – s - строка с текстом, root - корень бора. Возвращает массив ответов.

def print_bor(root) – функция вывода бора. Аргумент – корень бора. Возвращаемого значения нет.

Оценка сложности программы.

По памяти: O(n*q), где n – общая длина слов в словаре, q – Размер алфавита

По времени: O(n*q + H + k), где H – длина текста, k – общая длина всех совпадений.

Тестирование с промежуточными выводами

Для программы без джокера:

```
введите строку для поиска
abcbabcba
введите количество шаблонов
3
введите шаблоны
abc
ab
bc
максимальная длина цепочки суффиксных ссылок равна 2
максимальная длина цепочки конечных ссылок равна 1

вид бора (обход в ширину)

можем пойти в следующие вершины из корня : a b

можем пойти в следующие вершины из a : b
суффиксная ссылка указывает на возможные переходы ['a', 'b']
b терминальная для 2

можем пойти в следующие вершины из b : c
суффиксная ссылка указывает на возможные переходы ['a', 'b']
c терминальная для 3

можем пойти в следующие вершины из b : c
суффиксная ссылка указывает на возможные переходы ['c']
c терминальная для 1 3

можем пойти в следующие вершины из c : -
суффиксная ссылка указывает на возможные переходы ['a', 'b']

можем пойти в следующие вершины из c : -
суффиксная ссылка указывает на возможные переходы []

найдено решение на 1 символе, шаблон номер 2
найдено решение на 1 символе, шаблон номер 1
найдено решение на 2 символе, шаблон номер 3
перешли по суффиксной ссылке, обрабатывая символ b
перешли по суффиксной ссылке, обрабатывая символ b
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 5 символе, шаблон номер 2
найдено решение на 5 символе, шаблон номер 1
найдено решение на 6 символе, шаблон номер 3
перешли по суффиксной ссылке, обрабатывая символ b
перешли по суффиксной ссылке, обрабатывая символ b
перешли по суффиксной ссылке, обрабатывая символ a

1 1
1 2
2 3
5 1
5 2
6 3
```

```
введите строку для поиска
tagt
введите количество шаблонов
3
введите шаблоны
tagt
agt
gt
максимальная длина цепочки суффиксных ссылок равна 4
максимальная длина цепочки конечных ссылок равна 2

вид бора (обход в ширину)

можем пойти в следующие вершины из корня : t a g

можем пойти в следующие вершины из t : a
суффиксная ссылка указывает на возможные переходы ['t', 'a', 'g']

можем пойти в следующие вершины из a : g
суффиксная ссылка указывает на возможные переходы ['t', 'a', 'g']

можем пойти в следующие вершины из g : t
суффиксная ссылка указывает на возможные переходы ['t', 'a', 'g']
t терминальная для 3

можем пойти в следующие вершины из a : g
суффиксная ссылка указывает на возможные переходы ['g']

можем пойти в следующие вершины из g : t
суффиксная ссылка указывает на возможные переходы ['t']
t терминальная для 2 3

можем пойти в следующие вершины из t : -
суффиксная ссылка указывает на возможные переходы ['a']

можем пойти в следующие вершины из g : t
суффиксная ссылка указывает на возможные переходы ['t']
t терминальная для 1 2 3

можем пойти в следующие вершины из t : -
суффиксная ссылка указывает на возможные переходы []

можем пойти в следующие вершины из t : -
суффиксная ссылка указывает на возможные переходы []

найдено решение на 1 символе, шаблон номер 1
найдено решение на 2 символе, шаблон номер 2
найдено решение на 3 символе, шаблон номер 3

1 1
2 2
3 3
```

Для программы с джокером:

```
введите строку для поиска
nanana
введите шаблон
nun
введите символ джокера
u
максимальная длина цепочки суффиксных ссылок равна 1
максимальная длина цепочки конечных ссылок равна 0

вид бора (обход в ширину)

можем пойти в следующие вершины из корня : n

можем пойти в следующие вершины из n : -
суффиксная ссылка указывает на возможные переходы ['n']

найдено решение на 1 символе, шаблон номер 1
найдено решение на 1 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 3 символе, шаблон номер 1
найдено решение на 3 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 5 символе, шаблон номер 1
найдено решение на 5 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
перешли по суффиксной ссылке, обрабатывая символ a

1
3
```

```
введите строку для поиска
abababababa
введите шаблон
a*ab*
введите символ джокера
*
максимальная длина цепочки суффиксных ссылок равна 1
максимальная длина цепочки конечных ссылок равна 0

вид бора (обход в ширину)

можем пойти в следующие вершины из корня : a

можем пойти в следующие вершины из a : b
суффиксная ссылка указывает на возможные переходы ['a']
b терминальная для 2

можем пойти в следующие вершины из b : -
суффиксная ссылка указывает на возможные переходы ['a']

найдено решение на 1 символе, шаблон номер 1
найдено решение на 1 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 3 символе, шаблон номер 1
найдено решение на 3 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 5 символе, шаблон номер 1
найдено решение на 5 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 7 символе, шаблон номер 1
найдено решение на 7 символе, шаблон номер 2
перешли по суффиксной ссылке, обрабатывая символ a
найдено решение на 9 символе, шаблон номер 1

1
3
5
```

Выводы.

В ходе выполнения лабораторной работы была реализована программа для поиска всех вхождения образцов в текст с использованием алгоритма Ахо Корасик. Также была реализована программа для нахождения вхождений шаблона с джокером. Была получена асимптотическая оценка времени работы алгоритма.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД.

Ахо-Корасик

```
class AhoNode:  #класс для построения бора (goto - возможные пути, out - шаблоны, удовлетворяющие данной
    вершине, fail - суффиксная ссылка)
  def __init__(self):
    self.goto = {}
    self.out = []
    self.fail = None
    self.isterm = False

def aho_create_bor(patterns): #построение бора вместе с суффиксными и конечными ссылками по массиву строк
  root = AhoNode()
  count = 1
  maxcount1 = 1
  maxcount2 = 0
  for i in range(len(patterns)):  #инициализация структуры бора
    node = root
    for symbol in patterns[i]:
      node = node.goto.setdefault(symbol, AhoNode())
    node.out.append([i,len(patterns[i])])
    node.isterm = True
  queue = []
  for node in root.goto.values(): #добавление в очередь первого уровня ребер
    queue.append(node)
    node.fail = root

  while len(queue) > 0:         #установка суффиксных/конечных ссылок
    rnode = queue.pop(0)
    for key, unode in rnode.goto.items():
      queue.append(unode)
      fnode = rnode.fail
      while fnode is not None and key not in fnode.goto:
        fnode = fnode.fail
      unode.fail = fnode.goto[key] if fnode else root
      unode.out += unode.fail.out
      #if unode.isterm:
        #print(len(unode.out)-1,maxcount2,unode.goto)
      if len(unode.out)-1>maxcount2 and unode.fail.out and unode.isterm:
        maxcount2 = len(unode.out)-1
      elif len(unode.out)>maxcount2 and unode.fail.out:
        maxcount2 = len(unode.out)
        print(unode.out,unode.goto)
  queue = []
  for node in root.goto.values():
    queue.append(node)
  while len(queue) > 0:         #подсчёт количества ссылок
    rnode = queue.pop(0)
    if not rnode.isterm:
      for key, unode in rnode.goto.items():
        count = 1
```

```
            queue.append(unode)
            fnode = rnode.fail
            while fnode.fail is not None:
              count+=1
              fnode = fnode.fail

            if maxcount1<count:
              maxcount1 = count
        else:
          count = 1
          fnode = rnode.fail
          while fnode.fail is not None:
            count+=1
            fnode = fnode.fail

          if maxcount1<count:
            maxcount1 = count
    print('максимальная длина цепочки суффиксных ссылок равна',maxcount1)
    print('максимальная длина цепочки конечных ссылок равна',maxcount2)
    return root


def aho_find_all(s, root): #реализация алгоритма поиска по строке и бору
    node = root
    ans = []
    for i in range(len(s)):
      while node is not None and s[i] not in node.goto: #пока находимся не в корне и некуда идти по бору
        node = node.fail                    #переход по суффиксной ссылке
        print('перешли по суффиксной ссылке, обрабатывая символ',s[i])
      if node is None:
        node = root
        continue
      node = node.goto[s[i]]                #шаг вперед
      for pattern in node.out:              #сохранение ответа
        print('найдено решение на',i - pattern[1] + 2,'символе, шаблон номер',pattern[0]+1)
        ans.append([i - pattern[1] + 2, pattern[0]+1])
    print()
    ans.sort(key=lambda x: (x[0],x[1]))
    return ans

def print_bor(root):                    #промежуточный вывод бора
    print()
    print('вид бора (обход в ширину)')
    print()
    queue = []
    for key, node in root.goto.items():
      queue.append(node)
    print('можем пойти в следующие вершины из корня',end = ' : ')
    tmp = []
    for key in root.goto.keys():
        print(key, end=' ')
        tmp.append(key)
```

```
    print()
    print()
    while len(queue) > 0:
        rnode = queue.pop(0)
        print('можем пойти в следующие вершины из', tmp[0],end=' : ')
        tmp.pop(0)
        if not rnode.goto.items():
            print('-')
        for key, node in rnode.goto.items():
            tmp.append(key)
            print(key, end=' ')
            print()
            print('суффиксная ссылка указывает на возможные переходы',list(rnode.fail.goto.keys()))
            queue.append(node)
            if node.out:
                print(key,end=' терминальная для ')
                for term in node.out:
                    print(term[0]+1,end=' ')
                print()
        if not rnode.goto.items():
            print('суффиксная ссылка указывает на возможные переходы',list(rnode.fail.goto.keys()))
        print()

print('введите строку для поиска')
s = input()
print('введите количество шаблонов')
n = int(input())
print('введите шаблоны')
patterns = []
for i in range(n):
    patterns.append(input())
root = aho_create_bor(patterns)
print_bor(root)
for x in aho_find_all(s, root):
    print(x[0],x[1])
```

Джокер

```
class AhoNode:  #класс для построения бора (goto - возможные пути, out - шаблоны, удовлетворяющие данной
      вершине, fail - суффиксная ссылка)
    def __init__(self):
        self.goto = {}
        self.out = []
        self.fail = None
        self.isterm = False

def aho_create_bor(patterns): #построение бора вместе с суффиксными и конечными ссылками по массиву строк
    root = AhoNode()
    count = 1
    maxcount1 = 1
    maxcount2 = 0
```

```
for i in range(len(patterns)):  #инициализация структуры бора
    node = root
    for symbol in patterns[i]:
        node = node.goto.setdefault(symbol, AhoNode())
    node.out.append([i,len(patterns[i])])
    node.isterm = True
queue = []
for node in root.goto.values(): #добавление в очередь первого уровня ребер
    queue.append(node)
    node.fail = root

while len(queue) > 0:          #установка суффиксных/конечных ссылок
    rnode = queue.pop(0)
    for key, unode in rnode.goto.items():
        queue.append(unode)
        fnode = rnode.fail
        while fnode is not None and key not in fnode.goto:
            fnode = fnode.fail
        unode.fail = fnode.goto[key] if fnode else root
        unode.out += unode.fail.out

        if len(unode.out)-1>maxcount2 and unode.fail.out and unode.isterm:
            maxcount2 = len(unode.out)-1
        elif len(unode.out)>maxcount2 and unode.fail.out:
            maxcount2 = len(unode.out)
            print(unode.out,unode.goto)
queue = []
for node in root.goto.values():
    queue.append(node)
while len(queue) > 0:          #подсчёт количества ссылок
    rnode = queue.pop(0)
    if not rnode.isterm:
        for key, unode in rnode.goto.items():
            count = 1
            queue.append(unode)
            fnode = rnode.fail
            while fnode.fail is not None:
                count+=1
                fnode = fnode.fail

            if maxcount1<count:
                maxcount1 = count
    else:
        count = 1
        fnode = rnode.fail
        while fnode.fail is not None:
            count+=1
            fnode = fnode.fail

        if maxcount1<count:
            maxcount1 = count
print('максимальная длина цепочки суффиксных ссылок равна',maxcount1)
```

```
    print('максимальная длина цепочки конечных ссылок равна',maxcount2)
    return root


def aho_find_all(s, root): #реализация алгоритма поиска по строке и бору
    node = root
    ans = []
    for i in range(len(s)):
        while node is not None and s[i] not in node.goto: #пока находимся не в корне и некуда идти по бору
            node = node.fail                        #переход по суффиксной ссылке
            print('перешли по суффиксной ссылке, обрабатывая символ',s[i])
        if node is None:
            node = root
            continue
        node = node.goto[s[i]]                      #шаг вперед
        for pattern in node.out:                    #сохранение ответа
            print('найдено решение на',i - pattern[1] + 2,'символе, шаблон номер',pattern[0]+1)
            ans.append([i - pattern[1] + 2, pattern[0]+1])
    print()
    ans.sort(key=lambda x: (x[0],x[1]))
    return ans

def print_bor(root):                    #промежуточный вывод бора
    print()
    print('вид бора (обход в ширину)')
    print()
    queue = []
    for key, node in root.goto.items():
        queue.append(node)
    print('можем пойти в следующие вершины из корня',end = ' : ')
    tmp = []
    for key in root.goto.keys():
            print(key, end=' ')
            tmp.append(key)
    print()
    print()
    while len(queue) > 0:
        rnode = queue.pop(0)
        print('можем пойти в следующие вершины из', tmp[0],end=' : ')
        tmp.pop(0)
        if not rnode.goto.items():
            print('-')
        for key, node in rnode.goto.items():
            tmp.append(key)
            print(key, end=' ')
            print()
            print('суффиксная ссылка указывает на возможные переходы',list(rnode.fail.goto.keys()))
            queue.append(node)
            if node.out:
                print(key,end=' терминальная для ')
                for term in node.out:
                    print(term[0]+1,end=' ')
```

```
            print()
        if not rnode.goto.items():
            print('суффиксная ссылка указывает на возможные переходы',list(rnode.fail.goto.keys()))
        print()

print('введите строку для поиска')
s = input()
print('введите шаблон')
pat = input()
startpos = []
patterns = []
print('введите символ джокера')
joker = input()
count = [0]*len(s)
string = ''
n = 0
flag = 1
for symb in pat:
    n+=1
    if symb!=joker:
        if flag==1:
            startpos.append(n)
            string = ''
        string+=symb
        flag = 0
    else:
        flag = 1
        if string:
            patterns.append(string)
        string = ''
if string:
    patterns.append(string)
root = aho_create_bor(patterns)
print_bor(root)
for item in aho_find_all(s, root):
    if item[0]-startpos[item[1]-1]>=0:
        count[item[0]-startpos[item[1]-1]]+=1
count=count[:len(s)-len(pat)+1]
for i in range(len(count)):
    if count[i]==len(startpos):
        print(i+1)
```