# МИНОБРНАУКИ РОССИИ

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

# «ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

**Кафедра МОЭВМ**

# ОТЧЕТ

**по лабораторной работе №5**

# по дисциплине «Построение и анализ алгоритмов» Тема: Алгоритм Ахо-Корасика

| Студентка гр. 1304 | Чернякова А.Д. |
| --- | --- |
| Преподаватель | Шевелева А.М. |

Санкт-Петербург 2023

# Цель работы.

Изучить и реализовать на практике алгоритм Ахо-Корасика по поиску образцов в строке

# Задания.

**Задание 1.**

Разработайте программу, решающую задачу точного поиска набора образцов.

**Вход:**

Первая строка содержит текст. Вторая - число *n*, каждая следующая из *n* строк содержит шаблон из набора *P*. 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.

**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 *Р* в текст *Т*. 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

Шаблон Джокер **Выход:**

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

# Ход работы.

Задание 1.

Этот код реализует алгоритм *Aho-Corasick* для поиска множества паттернов в тексте.

Класс ***AhoNode*** представляет узел в дереве *Aho-Corasik*. Он содержит

следующие атрибуты:

*self.goto* - словарь, отображающий символы на следующие узлы в дереве;

self.*out* - список паттернов, которые заканчиваются в данном узле;

self.*suffix\_link* - ссылка на узел, куда следует перейти в случае неудачи.

Класс ***Tree*** представляет собой дерево *Aho-Corasik*. Для него реализованы следующие методы:

*aho\_create\_tree(self, patterns)* - строит дерево из заданных паттернов, которые принимает на вход. Для каждого паттерна происходит построение пути в дереве, а отдельные символы добавляются как узлы;

*aho\_create\_suffix\_link(self)* - создает суффиксные ссылки для каждого узла в дереве. Это выполняется с помощью алгоритма обхода в ширину. При этом вычисляются суффиксные ссылки для всех узлов, начиная от корневого узла и спускаясь по дереву.

Класс ***Aho*** выполняет основную логику алгоритма *Aho-Corasik*. Он содержит следующие методы:

*aho\_find\_occurence(self)* - осуществляет поиск паттернов в заданном тексте. При проходе по тексту, начиная с корневого узла дерева, он ищет совпадения символов и переходит к следующему узлу. Если достигнут конечный узел, то добавляет найденные паттерны в результат и возвращает результат;

*solve(self)* - основной метод, который решает задачу. Он считывает паттерны из ввода, строит дерево *Aho-Corasik*, вычисляет суффиксные ссылки и вызывает *aho\_find\_occurence* для поиска паттернов в тексте. Результаты сортируются и выводятся на экран.

При вызове *main*, создается объект класса *Aho* и вызывается метод *solve()*.

Этот код реализует алгоритм *Aho-Corasik,* который позволяет находить все вхождения заданного множества паттернов в тексте эффективно и за линейное время от длины текста.

Задание 2.

Этот код представляет расширенную версию алгоритма *Aho-Corasick* с поддержкой джокеров. Рассмотрим его поэтапно:

Класс ***AhoNode*** представляет узел в дереве *Aho-Corasick* с дополнительной информацией. Он содержит следующие атрибуты:

self.*goto* - словарь, отображающий символы на следующие узлы в дереве;

*self.out -* список индексов паттернов, которые заканчиваются в данном узле;

*self.suffix\_link* - ссылка на узел, куда следует перейти в случае неудачи.

Класс***Tree***представляет дерево *Aho-Corasick* с поддержкой джокеров. Он имеет следующие методы:

*aho\_create\_tree(self, patterns) -* строит дерево из заданных паттернов (которые принимает на вход) с учетом джокеров. Для каждого паттерна происходит построение пути в дереве, а отдельные символы добавляются как узлы. При построении пути, где встречается джокер, устанавливается ссылка на корневой узел;

*aho\_create\_suffix\_link(self) -* создает суффиксные ссылки для каждого узла в дереве с учетом джокеров. Это выполняется с помощью алгоритма обхода в ширину. При вычислении суффиксных ссылок учитывается наличие джокера.

Класс ***JokerAho*** выполняет основную логику алгоритма *Aho-Corasick* с поддержкой джокеров. Он содержит следующие методы:

*split\_joker\_pattern(self)* - разделяет паттерн с джокерами на отдельные паттерны и сохраняет их в *split\_patterns*. Также сохраняет индексы, по которым происходит разделение, в *split\_indexes;*

*aho\_find\_occurence(self)* - осуществляет поиск паттернов в заданном тексте с учетом джокеров. При проходе по тексту, начиная с корневого узла дерева, он ищет совпадения символов и переходит к следующему узлу. Если достигнут конечный узел, то добавляет найденные паттерны в результат и возвращает результат;

*solve(self) -* основной метод, который решает задачу. Он разделяет паттерн с джокерами на отдельные паттерны, строит дерево *Aho-Corasick* с учетом джокеров, вычисляет суффиксные ссылки и вызывает *aho\_find\_occurence* для поиска паттернов в тексте. Затем считает количество вхождений паттернов, учитывая джокеры, и выводит результаты на экран.

При вызове *main*, создается объект класса *JockerAho* и вызывается метод *solve()*.

Данный код реализует расширенную версию алгоритма *Aho-Corasick*, которая позволяет находить все вхождения заданного паттерна с джокерами в тексте.

Исходный код программ находится в приложении А.

# Выводы.

Рассмотрен, изучен и реализован метод поиска вхождений набора паттернов в текст с помощью алгоритма Ахо-Корасика. Для первой задачи с поиском нескольких паттернов реализован классический алгоритм по поиску вхождений набора паттернов в текст. Во второй задаче реализован поиск вхождение единственного паттерна с джокером в тексте, используя расширенную версию алгоритма Ахо-Корасика. Сложность затрат времени для алгоритма Ахо-Корасика не превышает линейное.

Программа прошла все тесты на платформе Stepic

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

название файла: task1.py

class AhoNode: # узел в дереве Aho-Corasick

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

self.goto = {} # переходы к следующим узлам

self.out = [] # паттерны, которые заканчиваются в этом узле

self.suffix\_link = link # ссылка на узел, куда перейти в случае неудачи

class Tree: # дерево Aho-Corasick

def \_\_init\_\_(self):

self.root = AhoNode()

def aho\_create\_tree(self, patterns): # строит дерево из заданных паттернов

for path in patterns:

node = self.root

for symbol in path:

node = node.goto.setdefault(symbol, AhoNode())

node.out.append(path)

def aho\_create\_suffix\_link(self): # создает суффиксные ссылки для каждого узла в дереве

queue = []

for node in self.root.goto.values():

queue.append(node)

node.suffix\_link = self.root

while queue:

current\_node = queue.pop(0)

for edge, child in current\_node.goto.items():

queue.append(child)

current\_link = current\_node.suffix\_link

while current\_link is not None and edge not in current\_link.goto:

current\_link = current\_link.suffix\_link

child.suffix\_link = current\_link.goto[edge] if current\_link else self.root

child.out += child.suffix\_link.out

class Aho: # выполняет основную логику алгоритма Aho-Corasick

def \_\_init\_\_(self):

self.tree = Tree()

self.text = input()

self.number\_of\_patterns = int(input())

self.patterns = []

self.dictionary = {}

def aho\_find\_occurence(self): # осуществляет поиск паттернов в заданном тексте

result = []

node = self.tree.root

for i in range(len(self.text)):

while node and self.text[i] not in node.goto:

node = node.suffix\_link

if not node:

node = self.tree.root

continue

node = node.goto[self.text[i]]

for pattern in node.out:

result.append([i - len(pattern) + 2, self.dictionary.get(pattern) + 1])

return result

# считывает паттерны из ввода, строит дерево Aho-Corasick,

# вычисляет суффиксные ссылки и вызывает aho\_find\_occur для поиска паттернов в тексте

def solve(self):

for i in range(self.number\_of\_patterns):

pattern = input()

self.patterns.append(pattern)

self.dictionary[pattern] = i

self.tree = Tree()

self.tree.aho\_create\_tree(self.patterns)

self.tree.aho\_create\_suffix\_link()

result = self.aho\_find\_occurence()

result.sort()

for element in range(len(result)):

print(f"{result[element][0]} {result[element][1]}")

if \_\_name\_\_ == "\_\_main\_\_":

aho = Aho()

aho.solve()

название файла: task2.py

class AhoNode: # узел в дереве Aho-Corasick

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

self.goto = {} # переходы к следующим узлам

self.out = [] # паттерны, которые заканчиваются в этом узле

self.suffix\_link = link # ссылка на узел, куда перейти в случае неудачи

class Tree: # дерево Aho-Corasick

def \_\_init\_\_(self):

self.root = AhoNode()

def aho\_create\_tree(self, patterns): # строит дерево из заданных паттернов

for indexes, path in enumerate(patterns):

node = self.root

for i in range(len(path)):

node = node.goto.setdefault(path[i], AhoNode(self.root))

node.out.append(indexes)

def aho\_create\_suffix\_link(self): # создает суффиксные ссылки для каждого узла в дереве

queue = []

for node in self.root.goto.values():

queue.append(node)

while queue:

current\_node = queue.pop(0)

for edge, child in current\_node.goto.items():

queue.append(child)

current\_link = current\_node.suffix\_link

while current\_link and edge not in current\_link.goto.keys():

current\_link = current\_link.suffix\_link

child.suffix\_link = current\_link.goto[edge] if current\_link else self.root

child.out += child.suffix\_link.out

class JokerAho: # выполняет основную логику алгоритма Aho-Corasick с джокером

def \_\_init\_\_(self):

self.tree = Tree()

self.text = input()

self.pattern\_with\_joker = input()

self.joker = input()

self.split\_patterns = []

self.split\_indexes = []

self.result = []

def split\_joker\_pattern(self): # создание списка подстрок из строки с джокером

self.split\_patterns = list(self.pattern\_with\_joker.split(self.joker))

while "" in self.split\_patterns:

self.split\_patterns.remove("")

flag = 1

for iterator, symbol in enumerate(self.pattern\_with\_joker):

if symbol == self.joker:

flag = 1

continue

if flag:

self.split\_indexes.append(iterator)

flag = 0

def aho\_find\_occurence(self): # осуществляет поиск паттернов в заданном тексте с учетом джокеров

result = []

node = self.tree.root

for i in range(len(self.text)):

while node and self.text[i] not in node.goto.keys():

node = node.suffix\_link

if not node:

node = self.tree.root

continue

node = node.goto[self.text[i]]

for pattern in node.out:

result.append([i - len(self.split\_patterns[pattern]) + 1, pattern])

return result

def solve(self): # решение задачи поиска вхождений в текст строки с джокером

self.split\_joker\_pattern()

self.tree = Tree()

self.tree.aho\_create\_tree(self.split\_patterns)

self.tree.aho\_create\_suffix\_link()

aho\_result = self.aho\_find\_occurence()

occurence\_counter = [0]\*len(self.text)

for text\_index, pattern\_index in aho\_result:

checking\_index = text\_index - self.split\_indexes[pattern\_index]

if 0 <= checking\_index < len(self.text):

occurence\_counter[checking\_index] += 1

for i in range(len(self.text) - len(self.pattern\_with\_joker) + 1):

if occurence\_counter[i] == len(self.split\_patterns):

self.result.append(i+1)

for element in self.result:

print(element)

if \_\_name\_\_ == "\_\_main\_\_":

joker\_aho = JokerAho()

joker\_aho.solve()