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

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Построение и анализ алгоритмов»
Тема: Ахо-Корасик

Студент гр. 1304 _____ Сулименко М.А.

Преподаватель _____ Шевелева А.М.

Санкт-Петербург
2023

Цель работы.

Изучить и на практике освоить метод точного поиска набора образцов в строке при помощи алгоритма Ахо-Корасика.

Задание.

Задание 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

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 xabvccbababcaxxabvccbababcax.

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}

Основные теоретические положения.

Пусть дан набор строк в алфавите размера k суммарной длины n. Алгоритм Ахо-Корасик за O(nk) времени и памяти строит префиксное дерево для этого набора строк, а затем по этому дереву строит автомат, который может использоваться в различных строковых задачах — например, для нахождения всех вхождений каждой строки из данного набора в произвольный текст за линейное время.

Для того чтобы найти все вхождения в текст заданного шаблона с масками Q, необходимо обнаружить вхождения в текст всех его безмасочных кусков. Пусть {Q1,…,Qk} — набор подстрок Q, разделенных масками, и пусть {l1,…,lk} — их стартовые позиции в Q. Например, шаблон abφφcφ содержит

две подстроки без масок ab и c и их стартовые позиции соответственно 1 и 5. Для алгоритма понадобится массив C. C[i] — количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i. Тогда появление подстроки Qi в тексте на позиции j будет означать возможное появление шаблона на позиции j−li+1.

1. Используя алгоритм Ахо-Корасик, находим безмасочные подстроки шаблона Q: когда находим Qi в тексте T на позиции j, увеличиваем на единицу C[j−li+1].

2. Каждое i, для которого C[i]=k, является стартовой позицией появления шаблона Q в тексте.

Поиск подстрок заданного шаблона с помощью алгоритма Ахо-Корасик выполняется за время O(m+n+a), где n — суммарная длина подстрок, то есть длина шаблона, m — длина текста, a — количество появлений подстрок шаблона. Далее просто надо пробежаться по массиву C и просмотреть значения ячеек за время O(m).

Выполнение работы.

В ходе работы было определено, что для обоих заданий можно скомпоновать все необходимые конструкции в один класс *AcoKarasik*. Было разработано 2 файла для решения 1 и 2 задания соответственно. Некоторые функции из 1 и 2 задания имеют одинаковые названия и функционал, поэтому они будут рассмотрены здесь лишь единожды. Приступим к разбору:

1) Подкласс *TrieNode* — подкласс узла, в котором хранятся поля с информацией о дочерних узлах данного узла (*children*), суффиксная ссылка (*sufaшчLink*) и список индексов шаблонов, для которой они являются терминальными (*patterns*).

2) Метод *create_trie* — создает лес — бор — дерево паттернов.

3) Метод *create_statemachine* — создает автомат Ахо-Корасика. Фактически создает бор и инициализирует суффиксные ссылки всех узлов, обходя дерево в ширину.

4) Метод *find_patterns* — находит все возможные подстроки из набора паттернов в строке.

5) Метод *find_patterns_with_mask* — находит все возможные подстроки из набора паттернов в строке для задачи точного поиска образца с джокером.

6) Метод *create_parts_and_indices* — создает части — паттерны — из строки с джокером и их начальные индексы в стркое с джокером.

7) Метод *input_data* — осуществляет ввод данных

8) Метод *print_answer* — выводит ответ на задачу

9) Метод *start_algorithm*— начинает решение задачи и запускает нужные методы

Разработанный программный код см. в приложении А.

Выводы.

Исследован, изучен метод точного поиска набора образцов в строке - алгоритм Ахо-Корасика. Для обоих задач все необходимы для решения конструкции определены в классе решения Solution. В рамках первой задачи используется классическая постановка задачи точного поиска набора образцов в строке, с которой алгоритм Ахо-Корасика справляется за линейное время. В рамках второй задачи используется неформальная постановка задачи, а именно точный поиск образца с джокером в строке, и для ее решения понадобились дополнительные соображения, указанные в основных теоретических положениях. В итоге, дополненный и модифицированный алгоритм Ахо-Корасика справляется и с этой задачей за линейное время. Программа успешно прошла все тесты на Stepik как для первой, так и для второй задачи. Данный алгоритм является одним из наиболее эффективных для поиска всех заданных паттернов в тексте.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: task1.py

```
        # Класс для решения задачи алгоритмом Ахо-Карасика и создания
бора
    class AchoKarasik:
        # Подкласс узла бора
        class TrieNode:
            # Инициализация полей класса узла
            def __init__(self):
                self.patterns = []
                self.children = {}
                self.suffixLink = None

        # Инициализация полей класса решения задачи
        def __init__(self):
            self.inputText = str()
            self.patterns = list()
            self.textLength = int()
            self.answer = list()

        # Инициализация бора(дерево паттернов)
        # Возвращает корень дерева
        def initialize_trie(self):
            root = self.TrieNode()
            for index, path in enumerate(self.patterns):
                node = root
                for symbol in path:
                    node       =       node.children.setdefault(symbol,
self.TrieNode())
                node.patterns.append(index)
            return root

        # Создание автомата для алгоритма Ахо-Карасика
        def create_statemachine(self):
            root = self.initialize_trie()
            queue = []
            for node in root.children.values():
                queue.append(node)
                node.suffixLink = root

            while len(queue) > 0:
                currentNode = queue.pop(0)
                for sym, child in currentNode.children.items():
                    queue.append(child)
                    currentSuffixLink = currentNode.suffixLink
                    while currentSuffixLink is not None and sym not in
currentSuffixLink.children:
                        currentSuffixLink                                =
currentSuffixLink.suffixLink
                    child.suffixLink  =  currentSuffixLink.children[sym]
if currentSuffixLink else root
                    child.patterns += child.suffixLink.patterns
```

```
            return root

        # Метод для нахождения всех подстрок из набора паттернов в
строке
        def find_patterns(self):
            root = self.create_statemachine()
            node = root
            for i in range(len(self.inputText)):
                while node is not None and self.inputText[i] not in
node.children:
                    node = node.suffixLink
                if node is None:
                    node = root
                    continue
                node = node.children[self.inputText[i]]
                for pattern in node.patterns:
                    self.answer.append((i - len(self.patterns[pattern])
+ 2, pattern + 1))
            self.answer = sorted(self.answer)

        # Метод для ввода данных
        def input_data(self):
            self.inputText = input()
            self.textLength = int(input())
            self.patterns = [input() for _ in range(self.textLength)]

        # Метод для корректного вывода ответа
        def print_answer(self):
            for item in self.answer:
                print(*item)

        # Метод для начала решения и запуска необходимых методов
        def start_algorithm(self):
            self.input_data()
            self.find_patterns()
            self.print_answer()


    solver = AchoKarasik()
```

Название файла: task2.py

```
# Класс для решения задачи алгоритмом Ахо-Карасика и создания бора
class AchoKarasik:
    # Подкласс узла бора
    class TrieNode:
        # Инициализация полей класса узла
        def __init__(self):
            self.children = {}
            self.listOfPatterns = []
            self.suffixLink = None

    # Инициализация полей класса решения задачи
    def __init__(self):
        self.inputText = str()
        self.listOfPatterns = list()
```

```
            self.pattern = str()
            self.mask = str()
            self.answer = list()

        # Инициализация бора(дерево паттернов)
        # Возвращает корень дерева
        def initialize_trie(self):
            root = self.TrieNode()
            for index, path in enumerate(self.listOfPatterns):
                node = root
                for symbol in path:
                    node = node.children.setdefault(symbol,
self.TrieNode())
                node.listOfPatterns.append(index)
            return root

        # Создание автомата для алгоритма Ахо-Карасика
        def create_statemachine(self):
            root = self.initialize_trie()
            queue = []
            for node in root.children.values():
                queue.append(node)
                node.suffixLink = root
            while len(queue) > 0:
                currentNode = queue.pop(0)
                for symbol, child in currentNode.children.items():
                    queue.append(child)
                    currentSuffixLink = currentNode.suffixLink
                    while currentSuffixLink is not None and symbol not
in currentSuffixLink.children:
                        currentSuffixLink = currentSuffixLink.suffixLink
                    child.suffixLink =
currentSuffixLink.children[symbol] if currentSuffixLink else root
                    child.listOfPatterns +=
child.suffixLink.listOfPatterns
            return root

        # Метод для нахождения всех подстрок из набора паттернов в
строке
        def find_patterns(self):
            root = self.create_statemachine()
            node = root
            for i in range(len(self.inputText)):
                while node is not None and self.inputText[i] not in
node.children:
                    node = node.suffixLink
                if node is None:
                    node = root
                    continue
                node = node.children[self.inputText[i]]
                for pattern in node.listOfPatterns:
                    self.answer.append((i -
len(self.listOfPatterns[pattern]) + 1, pattern))
            self.answer = sorted(self.answer)

        # Создает части - паттерны - из строки с джокером
        # и их начальные индексы в строке с джокером
        def create_parts_and_indices(self):
```

```
            parts = list(filter(bool, self.pattern.split(self.mask)))
            indices = list()
            isMask = True
            for index, symbol in enumerate(self.pattern):
                if symbol == self.mask:
                    isMask = True
                    continue
                if isMask:
                    indices.append(index)
                    isMask = False
            return parts, indices

        # Метод для нахождения всех подстрок из набора паттернов в
строке
        # С джокером
        def find_patterns_with_mask(self):
            parts, indices = self.create_parts_and_indices()
            self.listOfPatterns = parts
            self.find_patterns()

            text = [0] * len(self.inputText)
            for ind, pInd in self.answer:
                index = ind - indices[pInd]
                if 0 <= index < len(text):
                    text[index] += 1

            result = []
            for i in range(len(text) - len(self.pattern) + 1):
                if text[i] == len(self.listOfPatterns):
                    result.append(i + 1)
            self.answer = result

        # Метод для ввода данных
        def input_data(self):
            self.inputText = input()
            self.pattern = input()
            self.mask = input()

        # Метод для корректного вывода ответа
        def print_answer(self):
            print(*self.answer, sep="\n")

        # Метод для начала решения и запуска необходимых методов
        def start_algorithm(self):
            self.input_data()
            self.find_patterns_with_mask()
            self.print_answer()


    solver = AchoKarasik()
    solver.start_algorithm()
```