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: Поиск набора подстрок в строке. Ахо-Корасик.

Студент гр. 3343 _____ Никишин С.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2025

Цель работы.

Изучить алгоритм поиска набора подстрок в строке. Разработать программу, находящую все вхождения подстрок в строку с помощью алгоритма Ахо-Корасика.

Задание.

Вариант 2. Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска.

Задание 6.1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:
Первая строка содержит текст (T, $1 \leq |T| \leq 100000$).
Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P = \{p_1,\ldots,p_n\}$ $1 \leq |p_i| \leq 75$
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 6.2.

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
```

Описание алгоритма.

Алгоритм Ахо-Корасика — это эффективный алгоритм поиска множества подстрок в тексте. Он сочетает в себе идеи использования бора (префиксного дерева) и автомата.

Этапы алгоритма (используется в заданиях 6.1 и 6.2):

1. Построение бора (Trie). Создаём бор для всех искомых строк:

    1.1. Корень соответствует пустой строке.

1.2. Для каждого символа строки переходим по соответствующему ребру (если его нет — создаем).

1.3. В конце помечаем терминальную вершину (храним длину образца или его идентификатор).

2. Построение суффиксных и конечных ссылок:

2.1. Строим суффиксные ссылки (перебираем узлы-детей и их символы):

2.1.1. Суффиксная ссылка корня и суффиксная ссылка его потомков указывает на сам корень.

2.1.2. Иначе идем по суффиксным ссылкам родителя, пока не найдем узел-ребенка с текущим символом.

2.2. Построение конечных ссылок:

2.2.1. Если суффиксная ссылка ведет в терминальный узел, она является терминальной

2.2.2. Иначе продолжаем идти по суффиксным ссылкам, пока не найдем терминальный узел.

3. Поиск шаблонов в тексте:

3.1. Начинаем с корня.

3.2. Для каждого символа в тексте:

3.2.1. Пока символ и символ в рассматриваемом узле не совпали, идем по суффиксным ссылкам.

3.2.2. Если символы совпали, переходим в дочерний узел.

3.3. Проверяем является ли данный узел терминальным и идем п конечным ссылкам.

3.4. Сохраняем найденные вхождения.

4. В итоге получаем найденные вхождения шаблонов.

Поиск при использовании шаблона с символом-джокером (используется в задании 6.2):

1. Делим шаблон на подшаблоны игнорируя символы-джокеры.

2. Ищем вхождения подшаблонов алгоритмом Ахо-Корасика.

3. Ищем вхождения шаблона:

3.1. Создается массив, который хранит количество найденных подстрок для каждой позиции текста.

3.2. Перебираем позиции найденных вхождений подшаблонов. Увеличиваем значение ячейки массива с индексов позиции вхождения.

3.3. Если число в ячейки массива соответствует числу подшаблонов, индекс ячейки – начало вхождения полного шаблона.

Описание алгоритма индивидуализации:

1. Ищем вхождения шаблонов алгоритмом Ахо-Корасика.

2. Рассматриваем найденный массив вхождений. Рассматриваем каждое вхождения, с остальными, с которыми еще не рассматривалось это вхождение ранее.

3. Вычисляем концы рассматриваемых вхождений и проверяем не оканчивается ли хотя бы одно из них раньше, чем начинается другое.

3.1. Если оканчивается рассматриваем следующую пару

3.2. Иначе находим точку пересечения и заносим информацию о вхождениях и точке пересечения в массив найденных пересечений

4. В конце получаем массив найденных пересечений.

	Сложность по времени
Построение бора	O(L)
Построение ссылок	O(A * L)
Поиск	O(T + L + n)
Деление шаблона на подшаблоны	O(P)
Поиск пересечений	$O(n^2)$

	Сложность по памяти

Хранение автомата	O(A * L)

, где L = Σ длин всех шаблонов, T – длина текста, A – число символов в алфавите (число разных символов среди всех шаблонов), P – длина шаблона с джокерами, n – число вхождений

Исходные коды реализующие алгоритмы из заданий 6.1 и 6.2 представлены в приложении (приложении 1 и 2 соответственно).

Описание функций и структур данных.

Структуры данных:

1. *TrieNode* – структура, представляющая из себя узел бора.

Содержит поля:

id - уникальный идентификатор узла (для вывода)

isTerminal - флаг, является ли вершина терминалом

patternIndices - номера шаблонов, заканчивающихся в этом узле

patternLength - длина шаблона (для определения позиции)

childrens - указатели на дочерние узлы

suffixLink - суффиксная ссылка

finalLink - конечная ссылка

2. *AhoCorasicAlgorithm* – класс, который хранит автомат (бор с ссылками) реализует методы для работы с ним, а так же сам алгоритм Ахо-Корасика.

3. *C = [0] * (len(text) + 1)* - массив, где C[i] - количество встретившихся в тексте безмасочных подстрок шаблона, который начинается в тексте на позиции i.

Функции и методы:

1. *def __init__(self, patterns: list):* метод класса *AhoCorasicAlgorithm*, инициализирует алгоритм, создает автомат.

Содержит:

root – корень бора (тип *TrieNode*)

nodeCount – счетчик числа узлов в боре (тип *int*)

Параметры:

patterns – массив шаблонов, добавляемых в бор (тип *list*).

Сложность по времени: O(L), где L = Σ длин всех шаблонов

2. *def __add(self, pattern: str, index: int):* метод класса *AhoCorasicAlgorithm*, добавляет новый шаблон в бор.

Параметры:

pattern – массив шаблон, добавляемый в бор (тип *str*).

index – индекс добавляемого шаблона.

Сложность по времени: O(L), где L = Σ длин всех шаблонов

3. *def __makeLinks(self):* метод класса *AhoCorasicAlgorithm*, создает суффиксные и конечные ссылки.

Сложность по времени: O(A * L), где L = Σ длин всех шаблонов, A – число символов в алфавите (число разных символов среди всех шаблонов)

4. *def search(self, text):* метод класса *AhoCorasicAlgorithm*, ищет все вхождения шаблонов в тексте.

Параметры:

text – текст, в котором осуществляется поиск (тип *str*).

Возвращает:

results – отсортированный по позициям в текст массив найденных вхождений (элемент массива имеет вид (позиция начала вхождения, индекс шаблона) – (*int, int*))

Сложность по времени: O(T + L + n), где L = Σ длин всех шаблонов, T – длина текста, n – число вхождений

5. *def getNodeCount(self):* метод класса *AhoCorasicAlgorithm*, позволяет получить число узлов в автомате.

Возвращает:

nodeCount – счетчик числа узлов в боре (тип *int*)

Сложность по времени: O(1)

6. *def __printAutomat(self):* метод класса *AhoCorasicAlgorithm*, выводит информацию о построенном автомате.

Сложность по времени: O(L), где L = Σ длин всех шаблонов

7. *def findIntersectingPatterns(results: list, patterns: list):* ищет пересечения вхождений.

Параметры:

results –массив найденных вхождений (тип *list*).

patterns – массив шаблонов

Возвращает:

intersectionPairs – массив с найденными пересечениями (тип *list*).

Сложность по времени: $O(n^2)$, где n – число элементов массива *results*.

8. *def findSubPatterns(pattern: str, jokerSymbol: str):* делит шаблон с символом-джокером на отдельные части.

Параметры:

pattern – шалон с джокер-символами, который требуется разделить (тип *str*).

jokerSymbol – джокер-символ (тип *str*).

Возвращает:

subPatterns, *positions* – список подшаблонов и список их позиций (типы *list*)

Сложность по времени: O(P), где P – длина шаблона с джокерами.

Тестирование.

Тестирование программ представлено на таблице.

Таблица 1. Задание 6.1

№ Теста	Входные данные	Выходные данные	Комментарий
1	NTAG 3 TAGT TAG T	2 2 2 3	Верно
2	shershe 4 he she his hers	1 2 2 1 2 4 5 2 6 1	Верно
3	TTTTTTTTTT 2 NAN GNG		Верно
4	TATTATAT 3 TATT TAT TATTAT	1 1 1 2 1 3 4 2 6 2	Верно
5	ABBBAABAB 3 ABB ABB ABA	1 1 1 2 6 3	Верно

Таблица 2. Задание 6.2

№ Теста	Входные данные	Выходные данные	Комментарий
1	ACTANCA A$$A$ $	1	Верно
2	ABBBACBBABBBC ABB $	1 5 9	Верно
3	ABBBACBBABBBC C $	5	Верно
4	ABBA C$ $		Верно

Таблица 3. Индивидуализация

№ Теста	Входные данные	Выходные данные	Комментарий
1	NTAG 3 TAGT TAG T	TAG и T пересекаются в точке 2	Верно
2	shershe 4 he she his hers	she и he пересекаются в точке 2 she и her пересекаются в точке 2 he и her пересекаются в точке 2 her и she пересекаются в точке 5 she и he пересекаются в точке 6	Верно
3	ABBBACBBABBBC ABB $	ABB и ABB пересекаются в точке 5 ABB и ABB пересекаются в точке 9	Верно
4	ABCMMMCAB 2 ABC CAB	Пересечений нет	Верно

Примеры работ программ с выводом промежуточных результатов при введённом значении указаны в приложении (приложение 3 и приложение 4 для программ 1 и 2 соответственно)

Выводы.

Был изучен алгоритм Ахо-Корасика поиска всех вхождений набора подстрок в строку. Разработана программа, реализующая алгоритм Ахо-Корасика, а также алгоритм поиска шаблона с джокерами.

Приложение

Приложение 1

```
FROM COLLECTIONS IMPORT DEQUE
IMPORT GRAPHVIZ

CLASS TRIENODE:
    """КЛАСС УЗЛА БОРА"""
    DEF __INIT__(SELF, NODEID: INT):
        SELF.ID = NODEID            # УНИКАЛЬНЫЙ ИДЕНТИФИКАТОР УЗЛА

        SELF.ISTERMINAL = FALSE     # ФЛАГ, ЯВЛЯЕТСЯ ЛИ ВЕРШИНА ТЕРМИНАЛОМ
        SELF.PATTERNINDICES = []    # НОМЕРА ШАБЛОНОВ, ЗАКАНЧИВАЮЩИХСЯ В ЭТОМ УЗЛЕ
        SELF.PATTERNLENGTH = 0      # ДЛИНА ШАБЛОНА

        SELF.CHILDRENS = {}         # УКАЗАТЕЛИ НА ДОЧЕРНИЕ УЗЛЫ
        SELF.SUFFIXLINK = NONE      # СУФФИКСНАЯ ССЫЛКА
        SELF.FINALLINK = NONE       # КОНЕЧНАЯ ССЫЛКА


CLASS AHOCORASICALGORITHM:
    """КЛАСС, РЕАЛИЗУЮЩИЙ АЛГОРИТМ АХО-КОРАСИКА"""
    DEF __INIT__(SELF, PATTERNS: LIST):
        """ИНИЦИАЛИЗИРУЕТ АЛГОРИТМ, СОЗДАЕТ АВТОМАТ"""

        SELF.ROOT = TRIENODE(0)             # КОРНЕВОЙ УЗЕЛ
        SELF.ROOT.SUFFIXLINK = SELF.ROOT    # СУФФИКСНАЯ ССЫЛКА КОРНЯ
        SELF.NODECOUNT = 1                  # СЧЕТЧИК УЗЛОВ
        SELF.PATTERNS = PATTERNS            # СОХРАНЯЕМ ШАБЛОНЫ
        SELF.MAX_ARCS = 0                   # МАКСИМАЛЬНОЕ КОЛИЧЕСТВО ДУГ ИЗ ОДНОЙ ВЕРШИНЫ

        # СТРОИМ БОР
        FOR INDEX IN RANGE(LEN(PATTERNS)):
            SELF.__ADD(PATTERNS[INDEX], INDEX)

        # СОЗДАЕМ ССЫЛКИ, СТРОИМ АВТОМАТ
        SELF.__MAKELINKS()

    DEF __ADD(SELF, PATTERN: STR, INDEX: INT):
        """ДОБАВЛЯЕТ НОВЫЙ ШАБЛОН В БОР"""
        CURRENTNODE = SELF.ROOT
        # ПЕРЕБИРАЕМ СИМВОЛЫ ШАБЛОНА
        FOR CHAR IN PATTERN:
            IF CHAR NOT IN CURRENTNODE.CHILDRENS:
                NEWNODE = TRIENODE(SELF.NODECOUNT)
                CURRENTNODE.CHILDRENS[CHAR] = NEWNODE
                SELF.NODECOUNT += 1
                # ОБНОВЛЯЕМ МАКСИМАЛЬНОЕ КОЛИЧЕСТВО ДУГ
                SELF.MAX_ARCS = MAX(SELF.MAX_ARCS, LEN(CURRENTNODE.CHILDRENS))
            CURRENTNODE = CURRENTNODE.CHILDRENS[CHAR]

        CURRENTNODE.ISTERMINAL = TRUE
        CURRENTNODE.PATTERNINDICES.APPEND(INDEX)
        CURRENTNODE.PATTERNLENGTH = LEN(PATTERN)

    DEF __MAKELINKS(SELF):
        """СОЗДАЕТ СУФФИКСНЫЕ И КОНЕЧНЫЕ ССЫЛКИ"""
        QUEUE = DEQUE()
        QUEUE.APPEND(SELF.ROOT)

        WHILE QUEUE:
            CURRENTNODE = QUEUE.POPLEFT()
```

```
        FOR CHAR, CHILDNODE IN CURRENTNODE.CHILDRENS.ITEMS():
            QUEUE.APPEND(CHILDNODE)

            # ДЛЯ ДЕТЕЙ КОРНЯ СУФФИКСНАЯ ССЫЛКА ВЕДЕТ В КОРЕНЬ
            IF CURRENTNODE == SELF.ROOT:
                CHILDNODE.SUFFIXLINK = SELF.ROOT
            ELSE:
                # ИЩЕМ ПЕРВУЮ ВОЗМОЖНУЮ СУФФИКСНУЮ ССЫЛКУ
                TEMP = CURRENTNODE.SUFFIXLINK
                WHILE (TEMP != SELF.ROOT) AND (CHAR NOT IN TEMP.CHILDRENS):
                    TEMP = TEMP.SUFFIXLINK

                # УСТАНАВЛИВАЕМ НАЙДЕННУЮ ССЫЛКУ ИЛИ ССЫЛКУ НА КОРЕНЬ
                IF CHAR IN TEMP.CHILDRENS:
                    CHILDNODE.SUFFIXLINK = TEMP.CHILDRENS[CHAR]
                ELSE:
                    CHILDNODE.SUFFIXLINK = SELF.ROOT

            # ПОСТРОЕНИЕ КОНЕЧНОЙ ССЫЛКИ
            IF CHILDNODE.SUFFIXLINK.ISTERMINAL:
                CHILDNODE.FINALLINK = CHILDNODE.SUFFIXLINK
            ELSE:
                CHILDNODE.FINALLINK = CHILDNODE.SUFFIXLINK.FINALLINK

DEF SEARCH(SELF, TEXT):
    """ИЩЕТ ВСЕ ВХОЖДЕНИЯ ШАБЛОНОВ В ТЕКСТЕ"""
    RESULTS = []
    CURRENTNODE = SELF.ROOT

    FOR POSITION IN RANGE(LEN(TEXT)):
        CHAR = TEXT[POSITION]

        # ИСПОЛЬЗУЕМ СУФФИКСНЫЕ ССЫЛКИ ПРИ ОТСУТСТВИИ ПЕРЕХОДА
        WHILE (CURRENTNODE != SELF.ROOT) AND (CHAR NOT IN CURRENTNODE.CHILDRENS):
            CURRENTNODE = CURRENTNODE.SUFFIXLINK

        # ПЕРЕХОДИМ ПО СИМВОЛУ, ЕСЛИ ПЕРЕХОД СУЩЕСТВУЕТ
        IF CHAR IN CURRENTNODE.CHILDRENS:
            CURRENTNODE = CURRENTNODE.CHILDRENS[CHAR]

        # ПРОВЕРЯЕМ ТЕРМИНАЛЬНЫЕ УЗЛЫ
        IF CURRENTNODE.ISTERMINAL:
            FOR PATTERNINDEX IN CURRENTNODE.PATTERNINDICES:
                STARTPOSITION = POSITION - CURRENTNODE.PATTERNLENGTH + 1
                RESULTS.APPEND((STARTPOSITION, PATTERNINDEX, CURRENTNODE.PATTERNLENGTH))

        # ПРОВЕРЯЕМ КОНЕЧНЫЕ ССЫЛКИ ДЛЯ НАХОЖДЕНИЯ ВСЕХ ВЛОЖЕННЫХ ШАБЛОНОВ
        TEMP = CURRENTNODE.FINALLINK
        WHILE TEMP:
            FOR PATTERNINDEX IN TEMP.PATTERNINDICES:
                STARTPOSITION = POSITION - TEMP.PATTERNLENGTH + 1
                RESULTS.APPEND((STARTPOSITION, PATTERNINDEX, TEMP.PATTERNLENGTH))
            TEMP = TEMP.FINALLINK

    RETURN SORTED(RESULTS)

DEF GETNODECOUNT(SELF):
    """ПОЗВОЛЯЕТ ПОЛУЧИТЬ ЧИСЛО УЗЛОВ В АВТОМАТЕ"""
    RETURN SELF.NODECOUNT

DEF GETMAXARCS(SELF):
    """ВОЗВРАЩАЕТ МАКСИМАЛЬНОЕ КОЛИЧЕСТВО ДУГ ИЗ ОДНОЙ ВЕРШИНЫ"""
    RETURN SELF.MAX_ARCS
```

```
    def visualizeBOR(self, filename="bor_tree"):
        """Визуализирует бор в PNG файл"""
        dot = graphviz.Digraph(comment='BOR Tree')
        dot.attr('node', shape='circle')

        queue = deque([self.root])
        visited = set([self.root])

        # Добавляем корневой узел
        root_label = "0"
        if self.root.isTerminal:
            root_label += "\\n" + ",".join(str(i+1) for i in self.root.patternIndices)
        dot.node('0', root_label, style='filled', fillcolor='lightblue')

        while queue:
            node = queue.popleft()

            for char, child in node.childrens.items():
                if child not in visited:
                    visited.add(child)
                    queue.append(child)

                    # Создаем метку для узла
                    node_label = f"{child.id}"
                    if child.isTerminal:
                        patterns = ",".join(str(i+1) for i in child.patternIndices)
                        node_label += f"\\n{patterns}"

                    dot.node(str(child.id), node_label)

                    # Добавляем ребро
                    dot.edge(str(node.id), str(child.id), label=char)

        # Сохраняем в файл
        dot.render(filename, format='png', cleanup=True)
        print(f"Дерево бора сохранено в файл: {filename}.png")

    def printAutomatInfo(self):
        """Выводит краткую информацию об автомате"""
        print(f"Информация об автомате:")
        print(f"Количество узлов: {self.getNodeCount()}")
        print(f"Количество шаблонов: {len(self.patterns)}")
        print(f"Максимальное количество дуг из одной вершины: {self.getMaxArcs()}")

        # Подсчитываем дополнительную статистику
        terminal_count = 0
        queue = deque([self.root])

        while queue:
            node = queue.popleft()

            if node.isTerminal:
                terminal_count += 1

            for child in node.childrens.values():
                queue.append(child)

        print(f"Терминальных узлов: {terminal_count}")


def removeFoundPatterns(text, results, patterns):
    """Вырезает найденные образцы из строки и возвращает остаток"""
    if not results:
```

```
        RETURN TEXT

    # Создаем массив для отметки позиций, которые нужно удалить
    REMOVE_MASK = [FALSE] * LEN(TEXT)

    # Помечаем позиции, которые попадают в найденные образцы
    FOR START_POS, PATTERN_IDX, PATTERN_LENGTH IN RESULTS:
        PATTERN = PATTERNS[PATTERN_IDX]
        END_POS = START_POS + PATTERN_LENGTH

        # Проверяем границы и помечаем позиции для удаления
        FOR I IN RANGE(MAX(0, START_POS), MIN(LEN(TEXT), END_POS)):
            REMOVE_MASK[I] = TRUE

    # Собираем остаток строки
    REMAINDER = []
    FOR I, CHAR IN ENUMERATE(TEXT):
        IF NOT REMOVE_MASK[I]:
            REMAINDER.APPEND(CHAR)

    RETURN ''.JOIN(REMAINDER)


DEF FINDPATTERNRANGES(RESULTS, PATTERNS):
    """Находит диапазоны найденных образцов для наглядного вывода"""
    RANGES = []
    FOR START_POS, PATTERN_IDX, PATTERN_LENGTH IN RESULTS:
        PATTERN = PATTERNS[PATTERN_IDX]
        END_POS = START_POS + PATTERN_LENGTH
        RANGES.APPEND((START_POS, END_POS, PATTERN))
    RETURN SORTED(RANGES)


DEF MAIN():
    # ввод данных
    TEXT = INPUT("Введите текст: ")
    NUM = INT(INPUT("Введите количество шаблонов: "))
    PATTERNS = [INPUT(F"Введите шаблон {I + 1}: ") FOR I IN RANGE(NUM)]

    # Создаем автомат Ахо-Корасика
    AHOCORASICALGORITHM = AHOCORASICALGORITHM(PATTERNS)

    # Визуализируем бор (опционально)
    TRY:
        AHOCORASICALGORITHM.VISUALIZEBOR("BOR_VISUALIZATION")
    EXCEPT:
        PRINT("Для визуализации бора установите GRAPHVIZ: PIP INSTALL GRAPHVIZ")

    # Выводим информацию об автомате
    AHOCORASICALGORITHM.PRINTAUTOMATINFO()

    # Пункт 1 варианта 5: Максимальное количество дуг из одной вершины
    MAX_ARCS = AHOCORASICALGORITHM.GETMAXARCS()
    PRINT(F"\N=== РЕЗУЛЬТАТЫ ДЛЯ ВАРИАНТА 5 ===")
    PRINT(F"1. Максимальное количество дуг, исходящих из одной вершины: {MAX_ARCS}")

    # Поиск вхождений
    RESULTS = AHOCORASICALGORITHM.SEARCH(TEXT)

    # Вывод результатов поиска
    PRINT(F"\NНайдено вхождений: {LEN(RESULTS)}")
    IF RESULTS:
        PRINT("Найденные вхождения:")
        RANGES = FINDPATTERNRANGES(RESULTS, PATTERNS)
```

```
        FOR START_POS, END_POS, PATTERN IN RANGES:
            PRINT(F"  ПОЗИЦИИ {START_POS}-{END_POS-1}: '{PATTERN}'")

        # ПУНКТ 2 ВАРИАНТА 5: ВЫРЕЗАЕМ НАЙДЕННЫЕ ОБРАЗЦЫ
        REMAINDER = REMOVEFOUNDPATTERNS(TEXT, RESULTS, PATTERNS)
        PRINT(F"\N2. РЕЗУЛЬТАТ ВЫРЕЗАНИЯ НАЙДЕННЫХ ОБРАЗЦОВ:")
        PRINT(F"   ИСХОДНЫЙ ТЕКСТ: '{TEXT}'")
        PRINT(F"   ОСТАТОК СТРОКИ:  '{REMAINDER}'")

        # ДОПОЛНИТЕЛЬНАЯ ИНФОРМАЦИЯ О ВЫРЕЗАНИИ
        ORIGINAL_LENGTH = LEN(TEXT)
        REMAINDER_LENGTH = LEN(REMAINDER)
        REMOVED_LENGTH = ORIGINAL_LENGTH - REMAINDER_LENGTH
        PRINT(F"   СТАТИСТИКА: УДАЛЕНО {REMOVED_LENGTH} СИМВОЛОВ ИЗ {ORIGINAL_LENGTH}")

        IF REMAINDER:
            PRINT(F"   СОХРАНЕНО: {REMAINDER_LENGTH} СИМВОЛОВ")
        ELSE:
            PRINT(F"   ВЕСЬ ТЕКСТ БЫЛ УДАЛЕН (СОСТОИТ ТОЛЬКО ИЗ ШАБЛОНОВ)")
    ELSE:
        PRINT("ШАБЛОНЫ НЕ НАЙДЕНЫ В ТЕКСТЕ")
        PRINT(F"ОСТАТОК СТРОКИ: '{TEXT}'")


IF __NAME__ == "__MAIN__":
    MAIN()
```

ПРИЛОЖЕНИЕ 2

```
FROM COLLECTIONS IMPORT DEQUE
IMPORT GRAPHVIZ

CLASS TRIENODE:
    """КЛАСС УЗЛА БОРА"""
    DEF __INIT__(SELF, NODEID: INT):
        SELF.ID = NODEID            # УНИКАЛЬНЫЙ ИДЕНТИФИКАТОР УЗЛА

        SELF.ISTERMINAL = FALSE     # ФЛАГ, ЯВЛЯЕТСЯ ЛИ ВЕРШИНА ТЕРМИНАЛОМ
        SELF.PATTERNINDICES = []    # НОМЕРА ШАБЛОНОВ, ЗАКАНЧИВАЮЩИХСЯ В ЭТОМ УЗЛЕ
        SELF.PATTERNLENGTH = 0      # ДЛИНА ШАБЛОНА

        SELF.CHILDRENS = {}         # УКАЗАТЕЛИ НА ДОЧЕРНИЕ УЗЛЫ
        SELF.SUFFIXLINK = NONE      # СУФФИКСНАЯ ССЫЛКА
        SELF.FINALLINK = NONE       # КОНЕЧНАЯ ССЫЛКА


CLASS AHOCORASICALGORITHM:
    """КЛАСС, РЕАЛИЗУЮЩИЙ АЛГОРИТМ АХО-КОРАСИКА"""
    DEF __INIT__(SELF, PATTERNS: LIST):
        """ИНИЦИАЛИЗИРУЕТ АЛГОРИТМ, СОЗДАЕТ АВТОМАТ"""

        SELF.ROOT = TRIENODE(0)             # КОРНЕВОЙ УЗЕЛ
        SELF.ROOT.SUFFIXLINK = SELF.ROOT    # СУФФИКСНАЯ ССЫЛКА КОРНЯ
        SELF.NODECOUNT = 1                  # СЧЕТЧИК УЗЛОВ
        SELF.PATTERNS = PATTERNS            # СОХРАНЯЕМ ШАБЛОНЫ
        SELF.MAX_ARCS = 0                   # МАКСИМАЛЬНОЕ КОЛИЧЕСТВО ДУГ ИЗ ОДНОЙ ВЕРШИНЫ

        # СТРОИМ БОР
        FOR INDEX IN RANGE(LEN(PATTERNS)):
            SELF.__ADD(PATTERNS[INDEX], INDEX)

        # СОЗДАЕМ ССЫЛКИ, СТРОИМ АВТОМАТ
        SELF.__MAKELINKS()
```

```
DEF __ADD(SELF, PATTERN: STR, INDEX: INT):
    """Добавляет новый шаблон в бор"""
    CURRENTNODE = SELF.ROOT
    # перебираем символы шаблона
    FOR CHAR IN PATTERN:
        IF CHAR NOT IN CURRENTNODE.CHILDRENS:
            NEWNODE = TRIENODE(SELF.NODECOUNT)
            CURRENTNODE.CHILDRENS[CHAR] = NEWNODE
            SELF.NODECOUNT += 1
            # Обновляем максимальное количество дуг
            SELF.MAX_ARCS = MAX(SELF.MAX_ARCS, LEN(CURRENTNODE.CHILDRENS))
        CURRENTNODE = CURRENTNODE.CHILDRENS[CHAR]

    CURRENTNODE.ISTERMINAL = TRUE
    CURRENTNODE.PATTERNINDICES.APPEND(INDEX)
    CURRENTNODE.PATTERNLENGTH = LEN(PATTERN)

DEF __MAKELINKS(SELF):
    """Создает суффиксные и конечные ссылки"""
    QUEUE = DEQUE()
    QUEUE.APPEND(SELF.ROOT)

    WHILE QUEUE:
        CURRENTNODE = QUEUE.POPLEFT()

        FOR CHAR, CHILDNODE IN CURRENTNODE.CHILDRENS.ITEMS():
            QUEUE.APPEND(CHILDNODE)

            # для детей корня суффиксная ссылка ведет в корень
            IF CURRENTNODE == SELF.ROOT:
                CHILDNODE.SUFFIXLINK = SELF.ROOT
            ELSE:
                # ищем первую возможную суффиксную ссылку
                TEMP = CURRENTNODE.SUFFIXLINK
                WHILE (TEMP != SELF.ROOT) AND (CHAR NOT IN TEMP.CHILDRENS):
                    TEMP = TEMP.SUFFIXLINK

                # устанавливаем найденную ссылку или ссылку на корень
                IF CHAR IN TEMP.CHILDRENS:
                    CHILDNODE.SUFFIXLINK = TEMP.CHILDRENS[CHAR]
                ELSE:
                    CHILDNODE.SUFFIXLINK = SELF.ROOT

            # построение конечной ссылки
            IF CHILDNODE.SUFFIXLINK.ISTERMINAL:
                CHILDNODE.FINALLINK = CHILDNODE.SUFFIXLINK
            ELSE:
                CHILDNODE.FINALLINK = CHILDNODE.SUFFIXLINK.FINALLINK

DEF SEARCH(SELF, TEXT):
    """Ищет все вхождения шаблонов в тексте"""
    RESULTS = []
    CURRENTNODE = SELF.ROOT

    FOR POSITION IN RANGE(LEN(TEXT)):
        CHAR = TEXT[POSITION]

        # используем суффиксные ссылки при отсутствии перехода
        WHILE (CURRENTNODE != SELF.ROOT) AND (CHAR NOT IN CURRENTNODE.CHILDRENS):
            CURRENTNODE = CURRENTNODE.SUFFIXLINK

        # переходим по символу, если переход существует
        IF CHAR IN CURRENTNODE.CHILDRENS:
```

```
            currentNode = currentNode.childrens[char]

        # проверяем терминальные узлы
        if currentNode.isTerminal:
            for patternIndex in currentNode.patternIndices:
                startPosition = position - currentNode.patternLength + 1
                results.append((startPosition, patternIndex, currentNode.patternLength))

        # проверяем конечные ссылки для нахождения всех вложенных шаблонов
        temp = currentNode.finalLink
        while temp:
            for patternIndex in temp.patternIndices:
                startPosition = position - temp.patternLength + 1
                results.append((startPosition, patternIndex, temp.patternLength))
            temp = temp.finalLink

    return sorted(results)

def getNodeCount(self):
    """Позволяет получить число узлов в автомате"""
    return self.nodeCount

def getMaxArcs(self):
    """Возвращает максимальное количество дуг из одной вершины"""
    return self.max_arcs

def visualizeBOR(self, filename="bor_tree"):
    """Визуализирует бор в PNG файл"""
    dot = graphviz.Digraph(comment='BOR Tree')
    dot.attr('node', shape='circle')

    queue = deque([self.root])
    visited = set([self.root])

    # Добавляем корневой узел
    root_label = "0"
    if self.root.isTerminal:
        root_label += "\\n" + ",".join(str(i+1) for i in self.root.patternIndices)
    dot.node('0', root_label, style='filled', fillcolor='lightblue')

    while queue:
        node = queue.popleft()

        for char, child in node.childrens.items():
            if child not in visited:
                visited.add(child)
                queue.append(child)

                # Создаем метку для узла
                node_label = f"{child.id}"
                if child.isTerminal:
                    patterns = ",".join(str(i+1) for i in child.patternIndices)
                    node_label += f"\\n{patterns}"

                dot.node(str(child.id), node_label)

                # Добавляем ребро
                dot.edge(str(node.id), str(child.id), label=char)

    # Сохраняем в файл
    dot.render(filename, format='png', cleanup=True)
    print(f"Дерево бора сохранено в файл: {filename}.png")

def printAutomatInfo(self):
```

```
        """Выводит краткую информацию об автомате"""
        print(f"Информация об автомате:")
        print(f"Количество узлов: {self.getNodeCount()}")
        print(f"Количество шаблонов: {len(self.patterns)}")
        print(f"Максимальное количество дуг из одной вершины: {self.getMaxArcs()}")

        # Подсчитываем дополнительную статистику
        terminal_count = 0
        queue = deque([self.root])

        while queue:
            node = queue.popleft()

            if node.isTerminal:
                terminal_count += 1

            for child in node.childrens.values():
                queue.append(child)

        print(f"Терминальных узлов: {terminal_count}")


def removeFoundPatterns(text, results, patterns):
    """Вырезает найденные образцы из строки и возвращает остаток"""
    if not results:
        return text

    # Создаем массив для отметки позиций, которые нужно удалить
    remove_mask = [False] * len(text)

    # Помечаем позиции, которые попадают в найденные образцы
    for start_pos, pattern_idx, pattern_length in results:
        pattern = patterns[pattern_idx]
        end_pos = start_pos + pattern_length

        # Проверяем границы и помечаем позиции для удаления
        for i in range(max(0, start_pos), min(len(text), end_pos)):
            remove_mask[i] = True

    # Собираем остаток строки
    remainder = []
    for i, char in enumerate(text):
        if not remove_mask[i]:
            remainder.append(char)

    return ''.join(remainder)


def findPatternRanges(results, patterns):
    """Находит диапазоны найденных образцов для наглядного вывода"""
    ranges = []
    for start_pos, pattern_idx, pattern_length in results:
        pattern = patterns[pattern_idx]
        end_pos = start_pos + pattern_length
        ranges.append((start_pos, end_pos, pattern))
    return sorted(ranges)


def main():
    # ввод данных
    text = input("Введите текст: ")
    num = int(input("Введите количество шаблонов: "))
    patterns = [input(f"Введите шаблон {i + 1}: ") for i in range(num)]
```

```
    # Создаем автомат Ахо-Корасика
    ahoCorasicAlgorithm = AhoCorasicAlgorithm(patterns)

    # Визуализируем бор (опционально)
    try:
        ahoCorasicAlgorithm.visualizeBOR("bor_visualization")
    except:
        print("Для визуализации бора установите graphviz: pip install graphviz")

    # Выводим информацию об автомате
    ahoCorasicAlgorithm.printAutomatInfo()

    # Пункт 1 варианта 5: Максимальное количество дуг из одной вершины
    max_arcs = ahoCorasicAlgorithm.getMaxArcs()
    print(f"\n=== РЕЗУЛЬТАТЫ ДЛЯ ВАРИАНТА 5 ===")
    print(f"1. Максимальное количество дуг, исходящих из одной вершины: {max_arcs}")

    # Поиск вхождений
    results = ahoCorasicAlgorithm.search(text)

    # Вывод результатов поиска
    print(f"\nНайдено вхождений: {len(results)}")
    if results:
        print("Найденные вхождения:")
        ranges = findPatternRanges(results, patterns)
        for start_pos, end_pos, pattern in ranges:
            print(f"  Позиции {start_pos}-{end_pos-1}: '{pattern}'")

        # Пункт 2 варианта 5: Вырезаем найденные образцы
        remainder = removeFoundPatterns(text, results, patterns)
        print(f"\n2. Результат вырезания найденных образцов:")
        print(f"   Исходный текст: '{text}'")
        print(f"   Остаток строки:  '{remainder}'")

        # Дополнительная информация о вырезании
        original_length = len(text)
        remainder_length = len(remainder)
        removed_length = original_length - remainder_length
        print(f"   Статистика: удалено {removed_length} символов из {original_length}")

        if remainder:
            print(f"   Сохранено: {remainder_length} символов")
        else:
            print(f"   Весь текст был удален (состоит только из шаблонов)")
    else:
        print("Шаблоны не найдены в тексте")
        print(f"Остаток строки: '{text}'")


if __name__ == "__main__":
    main()
```