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: Алгоритм Ахо-Корасик

Студентка гр. 8383 _____ Максимова А.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Разработка программы на языке программирования C++, решающей задачу точного поиска набора образцов в строке, с помощью алгоритма Ахо-Корасик, и выводящей в качестве результата индексы вхождений и соответствующие номера строк-шаблонов.

Постановка задач.

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 $P = \{p_1, \dots, p_n\}$ $1 \leq |p_i| \leq 75$

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.

Входные данные:

NTAG

3

TAGT

TAG

T

Выходные данные:

2 2

2 3

Вар. 5. Вычислить максимальное количество дуг, исходящих из одной вершины в боре; вырезать из строки поиска все найденные образцы и вывести остаток строки поиска.

Основные теоретические положения.

Алгоритм Ахо-Корасик решает задачу поиска нахождения для каждой строки-образца всех ее вхождений в строку поиска (текст).

Бор - структура данных, используемая для хранения набора строк, представляющая из себя подвешенное дерево с символами на ребрах (т. е. ориентированный связный граф без циклов, в котором в каждую вершину, кроме корня, входит 1 ребро). Строки получаются последовательной записью всех символов, записанных на ребрах, которые лежат на пути из корня в терминальную (или конечную) вершину.

Терминальная вершина - конечная вершина (со степенью 1).

Суффиксные ссылки для каждой вершины v - это вершина, в которой окажется наидлиннейший собственный суффикс строки, соответствующий вершине v. Особый случай корень бора - суффиксная ссылка из него - в него же.

Описание структур данных.

1. *Класс* `Reader`, *необходимый для считывания данных введенных пользователем, их проверки на корректность и создания объекта класс* Bohr.

* class `Reader` см. приложение А

Поля:

- `std::string string_;` - строка, в которой осуществляется поиск вхождений шаблонов.
- `std::vector <std::string> patterns;` - набор строк-образцов (шаблонов).
- `int numbPattern;` - количество строк-образцов.

Методы:

- `Reader();` - конструктор класса, используется для инициализации полей класса. Ничего не принимает и не возвращает.
- `void check(std::string& string_);` - метод, принимающий строку `string_`, в которую будет записано значение, введенное пользователем, в случае, если длина строки не превышает заданную, в ином случае, пользователь должен будет вновь ввести данные. Метод ничего не возвращает.
- `void check(int& numbPattern);` - метод, принимающий целочисленную переменную `numbPattern`, в которую будет записано значение количества шаблонов, в случае, если пользователь ввел данные корректно, иначе, повторный запрос данных. Метод ничего не возвращает.
- `void checkData(std::string& temp);` - метод, принимающий временную переменную для хранения шаблонов `temp`, в методе проверяется, что длина шаблона не превышает длину строки поиска. Метод ничего не возвращает.
- `void startRead();` - метод, ничего не принимает. Используется для считывания данных, введенных пользователем, вывода текстовых подсказок, создания объекта класс Bohr. Метод ничего не возвращает.

2. *Класс, используемый для хранения структуры объекта вершина, а также взаимодействия с ней.*

* class Vertex см. приложение А

Поля:

- `bool endStr;` - булевая переменная, используемая для обозначения конца шаблона.
- `std::map <char, int> nextVertex;` - словарь вершин потомков и символов, по которым можно в них перейти.
- `std::map <char, int> transitions;` - словарь переходов.
- `int numbPattern;` - номер шаблона, выводимый в итоговом решении.
- `int suffixLink;` - суффиксная ссылка из вершины.
- `char nameEdge;` - символ, лежащей на ребре, ведущем в данную вершину.
- `int parent;` - номер вершины - родителя.
- `char parentGoCur;` - символ, по которому их предка можно попасть в текущюю вершину (для суффиксных ссылок).

Методы:

- `Vertex();` - конструктор класса, используется для инициализации полей класса. Ничего не принимает и не возвращает.
- `void setEndStr(bool value);` - метод, принимающий булевое значение `value`, используется для установки состояния флага `endStr` - является вершина концом шаблона или нет. Метод ничего не возвращает.
- `void setNextVertex(char symb, int value);` - метод, принимающий чаровскую переменную `symb` и целочисленное значение `value`. Добавляет вершины-потомки в словарь, с соответствующими им символами на ребрах. Метод ничего не возвращает.
- `void setNumbPattern(int value);` - метод, принимающий

целочисленное значение `value`. Используется для установки значения номера вершины. Метод ничего не возвращает.

- `void setSuffixLink(int direction);` - метод, принимающий целочисленную переменную `direction`. Устанавливает значение суффиксной ссылки для текущей вершины. Метод ничего не возвращает.
- `void setNameEdge(char name);` - метод, принимающий символ, который используется для инициализации ребра. Метод ничего не возвращает.
- `void setParent(int parent);` - метод, принимающий целочисленную переменную `parent` - номер вершины-родителя и устанавливающий данное значение в поле `parent`. Метод ничего не возвращает.
- `void setTransitions(char symb, int index);` - метод, принимающий символ `symb` и целочисленное значение `index`, используемые для установки перехода. Метод ничего не возвращает.
- `std::map <char, int>& getTransitions();` - метод, ничего не принимающий на вход. Возвращает по ссылке словарь переходов.
- `bool getEndStr();` - метод, ничего не принимающий на вход. Возвращает значение поля `endStr`.
- `std::map <char, int>& getNextVertex();` - метод, ничего не принимает. Возвращает по ссылке словарь вершин-потомков.
- `int getNumbPattern();` - метод, ничего не принимает. Возвращает значение поля `numbPattern`.
- `int getSuffixLink();` - метод, ничего не принимает. Возвращает значение суффиксной ссылки для текущей вершины.
- `char getNameEdge();` - метод, ничего не принимает. Возвращает значение поля `nameEdge`.
- `int getParent();` - метод, ничего не принимает. Возвращает значение поля `parent`.
- `bool findMapElem(char symb);` - метод, принимающий на вход символ, по которому в словаре потомков производится поиск. В случае успеха

возвращает true, иначе false.

- `bool findMapTranzit(char symb);` – принимающий на вход символ, по которому в словаре переходов производится поиск. В случае успеха возвращает true, иначе false.

3. *Класс, используемый для реализации алгоритма Ахо-Корасик.*

* class Bohr см. приложение А

Поля:

- `int maxArcs;` - максимальное количество дуг, исходящих из одной вершины.
- `int numbPattern;` - количество паттернов (шаблонов).
- `size_t maxSizeBohr;` - переменная, используемая для записи максимального (относительно введенных данных) размера бора.
- `size_t realSizeBohr;` - переменная, используемая для хранения реального, вычисленного после добавления вершин в бор, размера бора.
- `std::string string_;` - строка, для хранения текста, в котором реализуется поиск вхождений шаблонов.
- `std::vector <Vertex> bohr;` - вектор, для хранения вершин бора.
- `std::vector <std::string> patterns;` - вектор, для хранения набора паттернов.
- `std::vector <std::pair <int, int>> result;` - вектор, используемый для хранения пар (индекс вхождения шаблона в текст, номер шаблона).

Методы:

- `Bohr(std::string string_, int numbPattern, std::vector <std::string> patterns, size_t lengthPatterns);` - конструктор класса, используется для инициализации полей класса.
- `void initVertex(char curSymb, int parent);` - метод,

принимающий символ `curSymb`, используемый для перехода по ребру, и целочисленное значение родителя вершины `parent`. Метод используется для инициализации полей вершины, которая после будет добавлена в бор. Метод ничего не возвращает.

- `void addVertex(int curIndex, char curSymb);` – метод, принимающий номер создаваемой вершины `curIndex`, и символ `curSymb`, для перехода по инцидентному ей ребру. Метод используется для добавления ребер в бор. Метод ничего не возвращает.
- `void addPatterns();` – метод, ничего не принимает. Используется для добавления строк (шаблонов) в бор. Метод ничего не возвращает.
- `int setGetTransition(int indexVertex, char symbol);` – метод, принимающий номер текущей вершины и символ, по которому нужно определить переход. Метод используется для получения значения вершины, в которую нужно перейти. Если для текущей вершины переход не установлен, то кроме его вычисления, происходит также его фиксирование. Метод возвращает значение вершины, в которую нужно перейти.
- `int setGetSuffixLink(int indexVertex);` – метод, принимающий номер текущей вершины, для которой возвращается значение суффиксной ссылки, если она известна, или происходит ее вычисление и сохранение.
- `void findEntry();` – метод, ничего не принимает. Используется для реализации поиска в тексте всех шаблонов, формируется вектор результатов. Метод ничего не возвращает.
- `static int cmp(std::pair <int, int> a, std::pair <int, int> b);` – компаратор.
- `void findMaxArcs();` – метод, ничего не принимает. Используется для определения максимального значения количества дуг, исходящих из одной вершины. Метод ничего не возвращает.
- `std::string cutText();` - метод, ничего не принимает. Используется для вырезания из строки поиска всех найденных шаблонов. Возвращает

отредактированную строку.

- `void coutResult();` – метод который ничего не принимает. Необходим для вывода результатов. Метод ничего не возвращает.

Описание алгоритма.

Алгоритм можно представить в виде последовательности шагов:

Этап 1: Построение бора, который будет хранить строки-шаблоны.

Шаг 1. Создание корня (пустой символ).

Шаг 2. Поэлементно перебираем строки. Проверяем есть ли переход по символу из текущей вершины в другую.

Шаг 2.1. Есть, тогда переходим к следующему символу.

Шаг 2.2. Перехода нет - создаем новую вершину.

Шаг 3. Возвращаемся на шаг два, до тех пор, пока не закончились символы в строке. Иначе переходим на шаг 4.

Шаг 4. Если строка закончилась, отмечаем флагом конец шаблона в данной вершине.*

**Примечание*: признаком конца строки в боре является терминальная вершина, но возможно частные случаи, когда один шаблон является подстрокой другого, в таком случае, строка закончится, но текущая вершина не будет терминальной, поэтому заводим флаг.

Этап 2: Вычисление суффиксных ссылок.

У каждой вершины должна быть суффикс ссылка, у корня - это ссылка в себя же. У вершин, инцидентных одному и тому же ребру с корнем - суффикс ссылка так же ведет в корень. Для других вершин необходимо провести вычисления.

Вычисление суффиксных ссылок является рекурсивным, так как чтобы определить вершину, в которую можно перейти из текущей вершины по символу (когда прямых ребер для перехода нет), необходимо пройти по ссылке

родителя и из новой текущей вершины перейти по символу, если это возможно, иначе опять продолжить перемещение по суффикс ссылкам.

Для запоминания переходов из текущей вершины по определенному символу используется словарь переходов.

Этап 3: Поиск вхождений.

Алгоритм считывает по символу из строки поиска и осуществляет переходы по вершинам бора. При достижении конечных вершин, вхождения фиксируются. Алгоритм продолжается до тех пор, пока не будет обработана вся строка поиска.

Оценка сложности алгоритма.

Сложность алгоритма по памяти можно оценить как $O(m)$, где m - суммарная длина всех строк-образцов, если точно, то $O(m+1)$, где 1 - это корень. Так как в худшем случае ни одна из строк - шаблонов не будет иметь повторяющегося префикса.

Сложность алгоритма по времени можно оценить как $O((m+n)\cdot \log k + t)$, где k - мощность алфавита (из условия 5), n - длина строки поиска, а t - количество вхождений шаблонов в текст, $\log k -$ временная сложность вставки в map, используемом для хранения информации о потомках, для переходов.

Тестирование.

Подробный тест.

Входные данные.

NTAG

3

TAGT

TAG

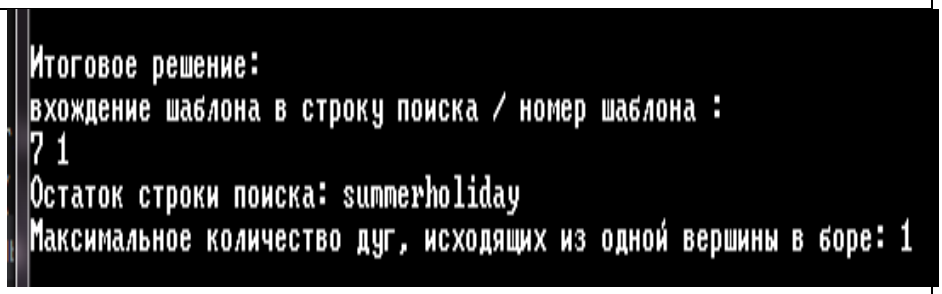
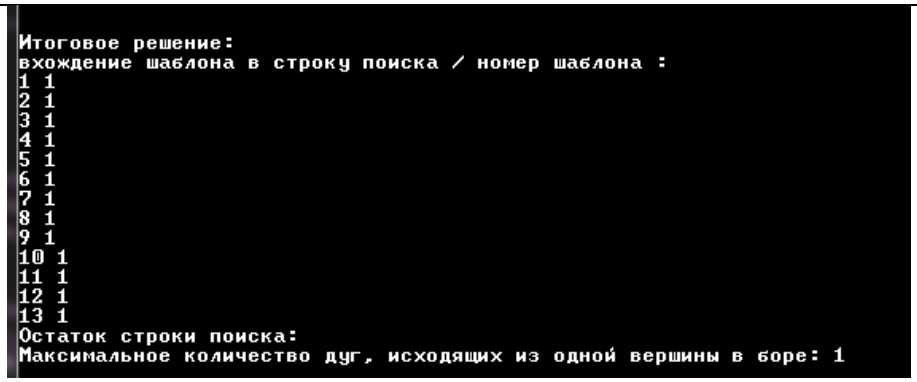
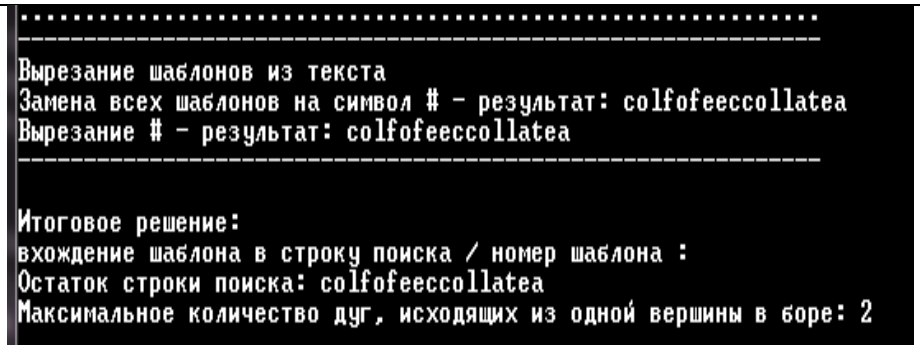
T

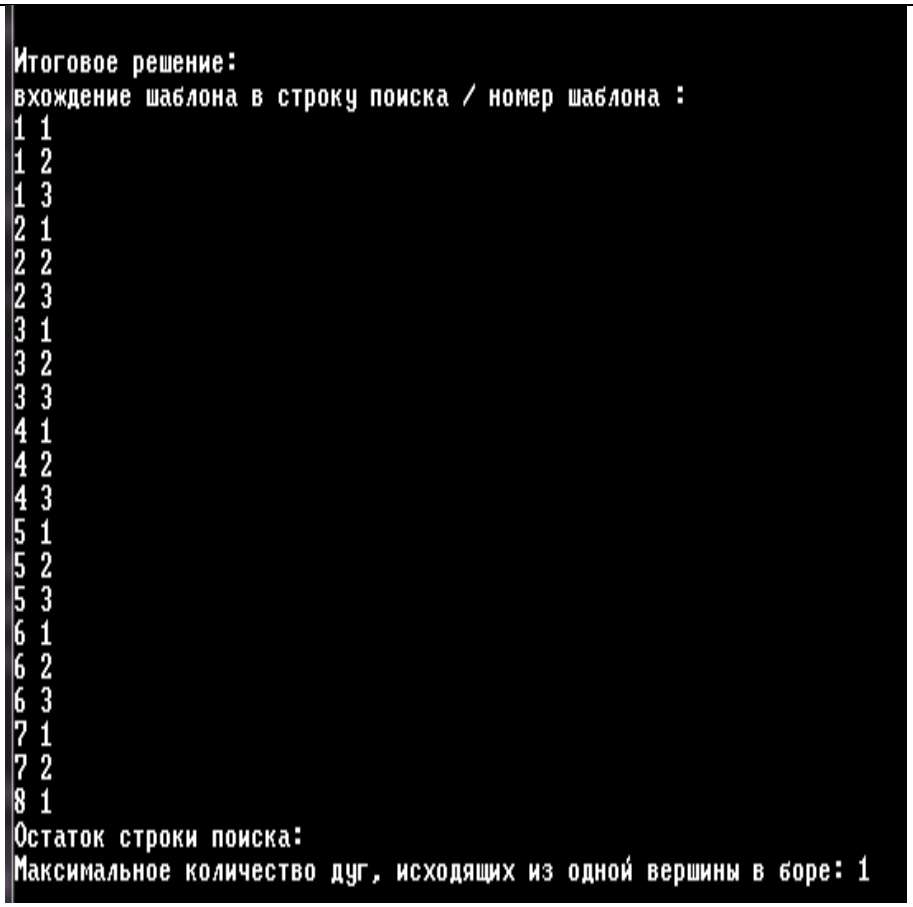
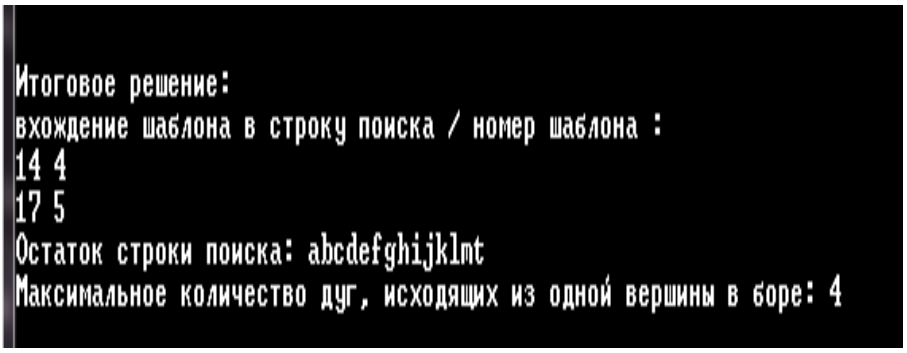
Выходные данные.

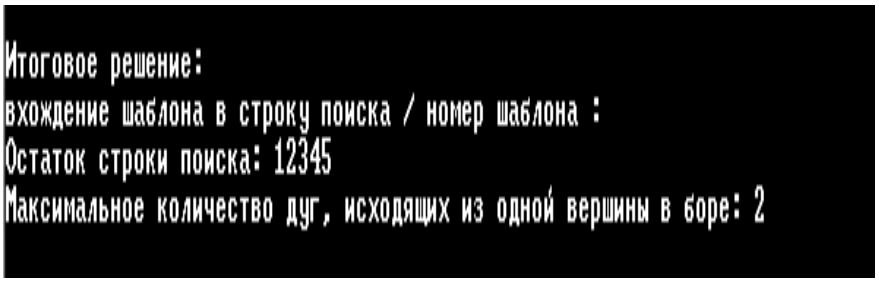
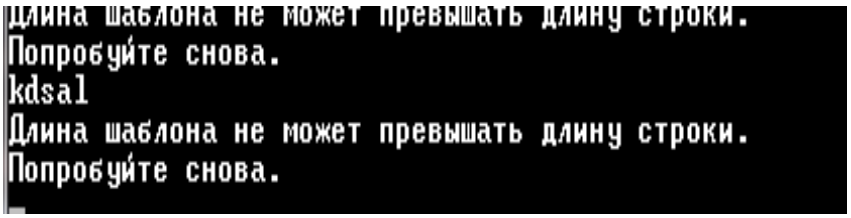
```
Ввод данных.
Введите строку, в которой будет осуществлятся поиск шаблонов.
NTAG 3 TAGT TAG T
Введите количество шаблонов.
Введите шаблон(ы).
**************************************************
Этап первый: добавление строк-образцов в бор.

Добавление шаблона TAGT в бор.
Создание ребра: --T-->
Создание ребра: --A-->
Создание ребра: --G-->
Создание ребра: --T-->
Добавление ребра завершено.

Добавление шаблона TAG в бор.
Ребро --T--> уже создано.
Ребро --A--> уже создано.
Ребро --G--> уже создано.
Добавление ребра завершено.

Добавление шаблона T в бор.
Ребро --T--> уже создано.
Добавление ребра завершено.

Количество вершин в боре = 5
**************************************************
**************************************************
Запуск поиска по строке: NTAG
Текущая вершина: 0
Переход из вершины 0 по символу N еще не вычислен.
Найден переход: 0 --N--> 0

Текущая вершина: 0
Переход из вершины 0 по символу T еще не вычислен.
Найден переход: 0 --T--> 1

Пришли в конечную вершину: 1
Найден шаблон: T
Индекс вхождения = 2
Номер шаблона = 3

Переход по суффиксной ссылке на вершину 1
Суффиксная ссылка еще не вычислена: Суффиксная ссылка уже вычислена: 1-->0

Текущая вершина: 1
Переход из вершины 1 по символу A еще не вычислен.
Найден переход: 1 --A--> 2


Переход по суффиксной ссылке на вершину 2
Суффиксная ссылка еще не вычислена.

Вычислим ее, перейдя по суффиксной ссылке родителя - 1
Суффиксная ссылка уже вычислена: 1-->0

Переход из вершины 0 по символу A еще не вычислен.
Найден переход: 0 --A--> 0

2-->0
```

```
Текущая вершина: 2
Переход из вершины 2 по символу G еще не вычислен.
Найден переход: 2 --G--> 3

Пришли в конечную вершину: 3
Найден шаблон: TAG
Индекс вхождения = 2
Номер шаблона = 2

Переход по суффиксной ссылке на вершину 3
Суффиксная ссылка еще не вычислена.

Вычислим ее, перейдя по суффиксной ссылке родителя - 2
Суффиксная ссылка уже вычислена: 2-->0

Переход из вершины 0 по символу G еще не вычислен.
Найден переход: 0 --G--> 0

3-->0


Задания для индивидулизации:
............................................................
Вычисление максимального количества дуг из одной вершины:
Текущий максимум = 0
Сравниваем количество дуг с текущим максимальным:
1 > 0
Текущий максимум = 1
Сравниваем количество дуг с текущим максимальным:
1 <= 1
Текущий максимум = 1
Сравниваем количество дуг с текущим максимальным:
1 <= 1
Текущий максимум = 1
Сравниваем количество дуг с текущим максимальным:
1 <= 1
Текущий максимум = 1
Сравниваем количество дуг с текущим максимальным:
0 <= 1
Текущий максимум = 1
Результат: 1
............................................................
------------------------------------------------------------

Вырезание шаблонов из текста
Замена всех шаблонов на символ # - результат: N###
Вырезание # - результат: N
------------------------------------------------------------


Итоговое решение:
вхождение шаблона в строку поиска / номер шаблона :
2 2
2 3
Остаток строки поиска: N

Максимальное количество дуг, исходящих из одной вершины в боре: 1

C:\Users\Александр\source\repos\PIAA5\Debug\PIAA5.exe (процесс 23500) зав
Чтобы автоматически закрывать консоль при остановке отладки, включите пар
Нажмите любую клавишу, чтобы закрыть это окно:
```

Таблица тестирования.

Входные данные	Выходные данные
summerexamholiday 1 exam	
AAAAAAAAAAAA A 1 A	
colfofeeccollatea 2 coffee cola	

BBBBBBBBB *3* *BB* *BBB* *BBBB*	
abcdefghijklmnoprst *5* *abcdf* *abggs* *dsfddf* *nop* *rs*	

12345 *2* *9* *10*	
a *3* *abc* *bds* *kdsal*	

Вывод.

В результате выполнения лабораторной работы была написана программа, реализующая алгоритм Ахо-Корасика, решающего задачу точного поиска шаблонов в строке. Также программа вычисляет максимальное значение количества дуг, исходящих из одной вершины бора и удаляет из строки поиска все вхождения шаблонов.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД АЛГОРИТМА ФОРДА-ФАЛКЕРСОНА

```
#include <iostream> //разделить на файлы

#include <vector>

#include <map>

#include <algorithm> //sort


#define MaxText    1000000

#define MaxPattern 750


class Vertex {                                  //вершина

private:


    bool endStr;                                //флаг = true, если в этой вершине оканчивается какая-либо (из имеющихся) строка-образец

    std::map <char, int> nextVertex;            //вершины-потомки: если в словаре есть, например, элемент с ключом 'A' - это означает,

                                                //что от текущей вершины можно пройти по ребру 'A'

    std::map <char, int> transitions;

    int numbPattern;                            //номер строки-образца, обозначаемого этой вершиной

    int suffixLink;                             //суффиксные ссылки


    char nameEdge;

    int parent;                                 //предок для реализации SufLink
```

```
    char parentGoCur;                          //символ, по которому
их предка можно попасть в текущюю вершину (для суффиксных ссылок)


public:

    Vertex()  :  endStr(false),  nameEdge('0'),  numbPattern(-1),
suffixLink(-1), parent(-1), parentGoCur('0') {}


    //Vertex(bool  endStr,  int  numbPattern)  :  Vertex()  {
//делегирующий конструктор

    //    this->endStr = endStr;

    //    this->numbPattern = numbPattern;

    //}


    //сеттеры

    void setEndStr(bool value) {

        endStr = value;

    }


    void setNextVertex(char symb, int value) {

        nextVertex.insert(std::make_pair(symb, value));

    }


    void setNumbPattern(int value) {

        numbPattern = value;

    }


    void setSuffixLink(int direction) {
```

```
    suffixLink = direction;
}

void setNameEdge(char name) {
    nameEdge = name;
}

void setParent(int parent) {
    this->parent = parent;
}

void setTransitions(char symb, int index) {
    transitions.insert(std::make_pair(symb, index));
}

//геттеры
std::map <char, int>& getTransitions() {
    return transitions;
}

bool getEndStr() {
    return endStr;
}

std::map <char, int>& getNextVertex() {
    return nextVertex;
```

```
}

int getNumbPattern() {
    return numbPattern;
}

int getSuffixLink() {
    return suffixLink;
}

char getNameEdge() {
    return nameEdge;
}

int getParent() {
    return parent;
}

bool findMapElem(char symb) {
    auto it = nextVertex.find(symb);
    if (it == nextVertex.end())
        return false;
    return true;
}

bool findMapTranzit(char symb) {
```

```
        auto it = transitions.find(symb);
        if (it == transitions.end())
            return false;
        return true;
    }

};

class Bohr {  //бор
private:

    int maxArcs;                                    //максимальное
количество дуг,исходящих из одной вершины
    int numbPattern;                                //кол-во
паттернов

    size_t                                          maxSizeBohr;
//максимальный размер бора
    size_t realSizeBohr;                            //реальный
размер бора

    std::string string_;                            //строка
поиска

    std::vector <Vertex> bohr;                      //бор, с
который будем работать
    std::vector <std::string> patterns;             //строки -
шаблоны
    std::vector <std::pair <int, int>> result;      //i - позиция
в тексте с 1; p - номер образца
```

```
public:


    Bohr(std::string string_, int numbPattern, std::vector
<std::string> patterns, size_t lengthPatterns) : realSizeBohr(0),
maxArcs(0) {

        this->string_ = string_;

        this->numbPattern = numbPattern;

        this->patterns = patterns;



        maxSizeBohr = lengthPatterns + 1;

        bohr.resize(maxSizeBohr);                  //в худшем случае
все-строки образцы не имеют одинаковых префиксов ->

                                                   //->        верхняя
оценка числа вершин бора = сумма длин всек строк-образцов + 1
(root)

                                                   //свободная  память
потом освобождается

        realSizeBohr = 1;                                  //корень -
инициализирован при создании

        bohr[0].setSuffixLink(0);                         //суффиксная
ссылка от корня ведет в корень

        addPatterns();

    }


    void initVertex(char curSymb, int parent) { //инициализация
вершин

        bohr[realSizeBohr].setEndStr(false);

        bohr[realSizeBohr].setNumbPattern(-1);

        bohr[realSizeBohr].setParent(parent);
```

```
        bohr[realSizeBohr].setNameEdge(curSymb);
        realSizeBohr++;
    }

    void addVertex(int curIndex, char curSymb) {  //добавление вершин в бор
        initVertex(curSymb, curIndex);
        bohr[curIndex].setNextVertex(curSymb, realSizeBohr - 1);
    }

    void addPatterns() {
        std::cout << "**************************************************\n";
        std::cout << "Этап первый: добавление строк-образцов в бор.\n";

        int curIndex;                                   //начинаем из корня бора - индекс текущий вершины в боре
        char curSymb;

        for (size_t i = 0; i < numbPattern; i++) {  //добавление в бор всех строк
            curIndex = 0;  //добавление новых строк начинается с корня
            std::cout << "\nДобавление шаблона " << patterns[i] << " в бор.\n";
            for (size_t j = 0; j < patterns[i].size(); j++) {  //проходим по строке

                curSymb = patterns[i][j];
```

```
//текущий символ


                if     (!bohr[curIndex].findMapElem(curSymb))     {
//если ребро еще не добавлено -добавляем

                    addVertex(curIndex, curSymb);

                        std::cout   <<   "Создание   ребра:   --"   <<
curSymb << "-->\n";

                }

                else {

                        std::cout << "Ребро --" << curSymb << "-->
уже создано.\n";

                }


                curIndex                                            =
bohr[curIndex].getNextVertex()[curSymb];

            }


            std::cout << "Добавление ребра завершено.\n";

            bohr[curIndex].setEndStr(true);

            bohr[curIndex].setNumbPattern(i          +          1);
//номер образца в ответ выводить с 1

        }


        bohr.resize(realSizeBohr);
//освобождаем незанимаемую память

        std::cout   <<   "\nКоличество   вершин   в   боре   =   "   <<
bohr.size() << "\n";

        std::cout                                                   <<
"**************************************************\n";
```

```
//поиск максимального количества дуг, исходящих из одной вершины
        findEntry();
    }

    int setGetTransition(int indexVertex, char symbol) {
//переходы

        if (bohr[indexVertex].findMapTranzit(symbol)) {       //у
вершины есть переход по символу
            std::cout << "Переход из вершины " << indexVertex << "
по символу " << symbol << " уже вычислен:\n";
            std::cout << indexVertex << " --" << symbol << "--> "
<< bohr[indexVertex].getTransitions()[symbol] << "\n\n";
        }
        else {                                                  //нет
перехода по символу в словаре - находим его
            std::cout << "Переход из вершины " << indexVertex << "
по символу " << symbol << " еще не вычислен.\n";

            if (bohr[indexVertex].findMapElem(symbol)) {     //есть
ли ребро с нужным символом
                bohr[indexVertex].setTransitions(symbol,
bohr[indexVertex].getNextVertex()[symbol]);
                std::cout << "Найден переход: " << indexVertex <<
"       --"       <<       symbol       <<       "-->       "       <<
bohr[indexVertex].getTransitions()[symbol] << "\n\n";
                //устанавливаем переход по символу с индексом =
номеру вершины, в которую мы передем из текущем по ребру с этим
символом
            }
            else {
```

```
                if (!indexVertex) {

                    bohr[indexVertex].setTransitions(symbol,    0);
//добавляем переход

                    std::cout << "Найден переход: " << indexVertex
<<      "      --"      <<      symbol      <<      "-->      "      <<
bohr[indexVertex].getTransitions()[symbol] << "\n\n";

                }

                else {//иначе переходим по суффиксной ссылке и
ищем переход там

                    std::cout << "Ищем переход:\n";

                    bohr[indexVertex].setTransitions(symbol,
setGetTransition(setGetSuffixLink(indexVertex), symbol));

                    std::cout << "Найден переход: " << indexVertex
<<      "      --"      <<      symbol      <<      "-->      "      <<
bohr[indexVertex].getTransitions()[symbol] << "\n\n";

                }

            }

        }

        return          bohr[indexVertex].getTransitions()[symbol];
//возвращаем переход

    }




    int         setGetSuffixLink(int         indexVertex)         {
//суффикусные ссылки

;


        if  (bohr[indexVertex].getSuffixLink()  !=  -1)  {    //есть
суффиксная ссылка

            std::cout << "Суффиксная ссылка уже вычислена: ";
```

```
            std::cout     <<     indexVertex     <<     "-->"     <<
bohr[indexVertex].getSuffixLink() << "\n\n";

        }


        else {                                                  //нужно
установить

            if  (!bohr[indexVertex].getParent())  {           //если
родитель корень

                std::cout << "Суффиксная ссылка еще не вычислена:
";

                bohr[indexVertex].setSuffixLink(0);
//устанавливаем ссылку

                std::cout << "Суффиксная ссылка уже вычислена: ";

                std::cout    <<    indexVertex    <<    "-->"    <<
bohr[indexVertex].getSuffixLink() << "\n\n";

            }


            else {                                              //нужно
вычислить

                                                                //переход
по ссылке предка, а после переход от вершины,

                                                                //в
которую пришли, по символу на ребре от предка до нашей вершины

                std::cout    <<    "Суффиксная    ссылка    еще    не
вычислена.\n";

                std::cout << "\nВычислим ее, перейдя по суффиксной
ссылке родителя - " << bohr[indexVertex].getParent() << "\n";


bohr[indexVertex].setSuffixLink(setGetTransition(setGetSuffixLink(
bohr[indexVertex].getParent()), bohr[indexVertex].getNameEdge()));

                std::cout    <<    indexVertex    <<    "-->"    <<
bohr[indexVertex].getSuffixLink() << "\n\n";
```

```
            }


        }


        return bohr[indexVertex].getSuffixLink();

    }


    void                        findEntry()                         {
//поиск вхождений в тексте

        std::cout << "Запуск поиска по строке: " << string_ <<
"\n";

        int               curIndex               =               0;
//поиск с корня


        for (size_t i = 0; i < string_.length(); i++) {
//бежим по строке

            std::cout << "Текущая вершина: " << curIndex << "\n";

            curIndex = setGetTransition(curIndex, string_[i]);
//осуществляем переход


            for (size_t j = curIndex; j > 0; j =
setGetSuffixLink(j)) { //проходим по суффикс ссылкам, пока не
попадем в корень


                if              (bohr[j].getEndStr())              {
//нашли вхождение шаблона в текст

                    std::cout << "Пришли в конечную вершину: " <<
curIndex << "\n";

                    std::cout    <<    "Найден    шаблон:    "    <<
patterns[bohr[j].getNumbPattern() - 1] << "\n";
```

```
                    std::cout << "Индекс вхождения = " << (i + 1)
- (patterns[bohr[j].getNumbPattern() - 1].length() - 1);

                    std::cout   <<   "\nНомер   шаблона   =   "   <<
bohr[j].getNumbPattern() << "\n";

                    result.push_back(std::make_pair((i   +   1)   -
(patterns[bohr[j].getNumbPattern()    -    1].length()    -    1),
bohr[j].getNumbPattern()));

                }

                std::cout << "\nПереход по суффиксной ссылке на
вершину " << curIndex << "\n";

            }

        }



        coutResult();

    }



    static int cmp(std::pair <int, int> a, std::pair <int, int> b)
{    //для сортировки

        if (a.first == b.first) return a.second < b.second;

        return a.first < b.first;

    }



    void                        findMaxArcs()                        {
//вычисление максимальнгого количества дуг

        std::cout                                                    <<
".....................................................\n";

        std::cout << "Вычисление максимального количества дуг из
одной вершины: \n";



        std::cout << "Текущий максимум = " << maxArcs << "\n";
```

```
        for (size_t i = 0; i < bohr.size(); i++) {



            std::cout << "Сравниваем количество дуг с текущим
максимальным: \n";



            if (bohr[i].getNextVertex().size() > maxArcs) {

                std::cout << bohr[i].getNextVertex().size() << " >
" << maxArcs << "\n";

                maxArcs = bohr[i].getNextVertex().size();

                std::cout << "Текущий максимум = " << maxArcs <<
"\n";

            }

            else {

                std::cout << bohr[i].getNextVertex().size() << "
<= " << maxArcs << "\n";

                std::cout << "Текущий максимум = " << maxArcs <<
"\n";

            }

        }

        std::cout << "Результат: " << maxArcs << "\n";

        std::cout                                                 <<
".....................................................\n";

    }



    std::string                    cutText()                    {
//вырезание найденных шаблонов из текста

        std::cout << "-----------------------------------------
-----------------\n";

        std::cout << "Вырезание шаблонов из текста\n";
```

```
        std::string temp;

        for (size_t i = 0; i < result.size(); i++) {

            for (size_t j = result[i].first - 1; j < (result[i].first + patterns[result[i].second - 1].size()) - 1; j++) {

                string_[j] = '#';

            }

        }

        std::cout << "Замена всех шаблонов на символ # - результат: ";

        std::cout << string_ << "\n";


        for (size_t k = 0; k < string_.size(); k++) {

            if (string_[k] != '#')

                temp.push_back(string_[k]);

        }

        std::cout << "Вырезание # - результат: " << temp << "\n";

        std::cout << "-------------------------------------------------\n";

        return temp;

    }


    void coutResult() { //печать результата


        std::cout << "\nЗадания для индивидулизации:\n";

        findMaxArcs();

        string_ = cutText();
```

```
        std::cout << "\nИтоговое решение:\n";
        std::cout << "вхождение шаблона в строку поиска / номер шаблона : \n";

        sort(result.begin(), result.end(), cmp); //сортировка

        for (size_t i = 0; i < result.size(); i++)
            std::cout << result[i].first << " " << result[i].second << "\n";

        std::cout << "Остаток строки поиска: ";
        for (size_t k = 0; k < string_.size(); k++) {
            std::cout << string_[k];
        }
        std::cout << "\nМаксимальное количество дуг, исходящих из одной вершины в боре: " << maxArcs << "\n";
    }
};

class Reader {
private:
    std::string string_; //строка, в которой осуществляется поиск вхождений
    std::vector <std::string> patterns; //строки-шаблоны
    int numbPattern; //количество строк-образцов
```

```
public:
    Reader() : string_("0"), numbPattern(0) {
        startRead();
    }

    void check(std::string& string_) {
        int exit = 1;

        do {
            std::cin >> string_;
            if (string_.length() > MaxText) {
               std::cout << "Недопустимый размер строки.\n";
               std::cout << "Попробуйте снова.\n";
            }
            else
                exit = 0;
        } while (exit);
    }

    void check(int& numbPattern)
//проверка корректность введенных данных
    {
        while (!(std::cin >> numbPattern))
        {
              std::cout << " Ошибка: данные некорректны! Ожидается
число.\n";
```

```
            std::cin.clear();
     //сброс коматозного состояния cin
            std::cin.ignore(1000, '\n');
            fflush(stdin);
//очистка потока ввода
        }

        std::cin.get();
    }

    void checkData(std::string& temp) {
        int exit = 1;

        do {
            std::cin >> temp;
            if (temp.length() > MaxPattern) {
                std::cout << "Недопустимый размер шаблона.\n";
                std::cout << "Попробуйте снова.\n";
            }
            else if (temp.length() > string_.length()) {
                std::cout << "Длина шаблона не может превышать
длину строки.\n";
                std::cout << "Попробуйте снова.\n";
            }
            else
                exit = 0;
        } while (exit);
```

```
    }

    void startRead() {
        size_t lengthPatterns = 0;

        std::cout << "Введите строку, в которой будет осуществлятся поиск шаблонов.\n";
        check(string_);

        std::cout << "Введите количество шаблонов.\n";
        check(numbPattern);

        std::cout << "Введите шаблон(ы).\n";
        std::string temp;
        for (int i = 0; i < numbPattern; i++) {
            checkData(temp);
            lengthPatterns += temp.length();
            patterns.push_back(temp);
        }

        Bohr bohr(string_, numbPattern, patterns, lengthPatterns);
    }
};

int main()
{
    setlocale(LC_ALL, "rus");
```

```
    std::cout << "Ввод данных.\n";
    Reader now_;

    return 0;
}
```