**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. 7382 Лящевская А. П.

Преподаватель Фирсов М. А.

Санкт-Петербург

2019

**Цель работы.**

Изучить алгоритм поиска множественных шаблонов в заданном тексте с и без использования символов-джокеров. Данным алгоритмом является алгоритм Ахо-Корасик. Написать реализацию алгоритма, найти асимптотики по времени и по памяти.

**Задание 1.**

Разработайте программу, решающую задачу точного поиска набора образцов.

**Вход**:

Первая строка содержит текст (T,1≤|T|≤100000 ).

Вторая - число n (1≤n≤3000), каждая следующая из n строк содержит шаблон из набора P={p1,…,pn}1≤|pi|≤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.

**Пример входных данных.**

CCCA

1

CC

***Пример выходных данных.***

*1 1*

*2 1*

**Задание 2.**

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

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

Например, образец аb??с? 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 неопределенной длины. В шаблоне входит хотя бы один символ не джокер, те шаблоны вида ??? недопустимы.

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

**Вход**:

Текст (T,1≤|T|≤100000 )

Шаблон (P,1≤|P|≤40)

Символ джокера

**Выход**:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).

Номера должны выводиться в порядке возрастания.

**Пример входных данных.**

ACT

A$

$

***Пример выходных данных.***

*1*

**Вариант 5.**

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

**Описание алгоритма.**

На вход программе подается строка для обработки, паттерны и для второй программы еще и символ-джокер.

Программа подразумевает два этапа: этап инициализации бора и нахождения всех паттернов в строке для обработки.

Первый этап.

1. Первоначальная инициализация: добавление вершины-корня в бор.
2. Добавление всех строк.

Строится бор последовательным добавлением исходных строк. Изначально есть 1 вершина, корень (root) — пустая строка. Добавление строки происходит так: начиная в корне, двигаемся по дереву, выбирая каждый раз ребро, соответствующее очередной букве строки. Если такого ребра нет, то оно создается вместе с вершиной. На Рисунке 1 приведен пример построенного бора для строк: 1)acab, 2)accc, 3)acac, 4)baca, 5)abb, 6)z, 7)ac. Красными кругами отмечены концы строк.

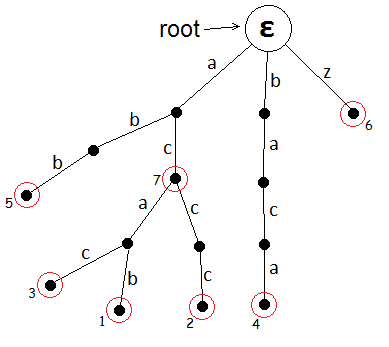


Рисунок 1.

Второй этап.

1. Нахождение всех вхождений паттернов в текст. Обработка происходит посимвольно.
2. Каждый из символов посылается в функцию автомата вместе текущим местоположением в боре(оно запоминается).
3. Автомат уже определяет следующее местоположение в боре. Если из текущей вершины ***v*** есть прямой путь до вершины соответствующей искомому символу **u**, то автомат возвращает этот путь. Если нет, то проверяется наличие джокера как **u**,иначе происходит переход по суффиксной ссылке (про них ниже) и поиск продолжается аналогично. В том случае, когда автомат пришел к корню, он останавливается. Поиск продолжается с корня. Схема представлена на Рисунке 2.

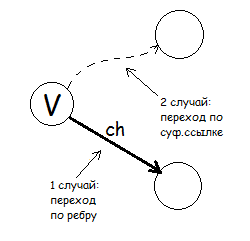


Рисунок 2.

1. Суффиксной ссылкой вершины ***v*** называется указатель на вершину ***u***, такую что строка ***u*** — наибольший cобственный суффикс строки ***v***, или, если такой вершины нет в боре, то указатель на корень. В частности, ссылка из корня ведет в него же.

В задаче получения суф. ссылки от вершины можно использовать ленивую динамику. Эвристика заключена в следующем: для получения суф. ссылки вершины ***v*** (строки *s*[*i*..*j*]) спустимся до ее предка *par*, пройдем по суф. ссылке *par* и запустим переход от текущей вершины *t* по символу *symb*, который написан на ребре от *par* до *v*. Очевидно, что сначала мы попадем в наибольший суфикс *s*[*i*..*j-1*] такой что, он имеет ребро с символом *symb*, потом пройдем по этому ребру. По определению, получившаяся вершина и есть суффикная ссылка из вершин ***v***. Схема представлена на Рисунке 3.

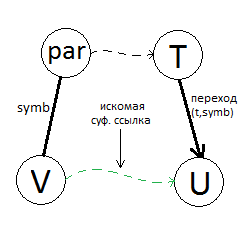


Рисунок 3.

Пример расстановки суф. ссылок для бора приведен на Рисунке 4.

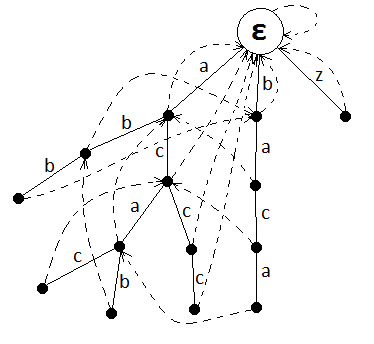


Рисунок 4.

1. После обработки для каждого символа местоположение в боре проверяется на достижение паттернов. В случае их нахождения, данные о их местоположении сохраняются в вектор.

В итоге, программа выводит собранные данные и завершается.

**Асимптотика алгоритма.**

Существующий вариант алгоритм проходит циклом по длине s (N=s.length()), откуда его уже можно оценить как O(N\*O(check)), но так как check прыгает только по заведомо помеченным вершинам, для которых flag=true, то общую ассимптотику можно оценить как O(N+t), где t — количество всех возможных вхождений всех строк-образцов в s. Если быть точным и учитывать вычисления автомата и суф. ссылок, то алгоритм работает O(M\*k+N+t), где M=bohr.size().

Память — константные массивы размера k для каждой вершины бора, откуда и выливается оценка O(M\*k).

Оказывается, другой способ хранения, а конкретно, обращения к алфавиту, способен изменить эту оценку. Будем использовать отображение map<char,int> вместо массива. Читаем здесь и здесь, видим, что структура данных map из STL реализована красно-черным деревом, а время обращения к его элементам пропорционально логарифму числа элементов. В нашем случае — двоичному логарифму размера алфавита k (что практически константа). Общее время — O((M+N)\*log k+t). На практике, это значительнее быстрее массива. Map вовсе не хранит лишних ячеек памяти для элементов, поэтому память пропорциональна количеству ребер в боре (а следовательно и количеству вершин в боре, т.к. в дереве с M вершин — M-1 ребер). Количество вычислений переходов автомата, очевидно, пропорционально длине строки. Получившаяся оценка — O(M+N).

**Описание структур**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Название структуры** | **Тип** | **Объект** | **Параметры** | **Описание** | **Возвращаемое значение** |
| struct bohr\_vrtx | - | int next\_vrtx[k]; | - | next\_vrtx[i] — номер вершины, в которую мы придем по символу с номером i в алфавите. | - |
| - | int pat\_num; | - | номер строки-образца, обозначаемого этой вершиной. | - |
| - | int suff\_link; | - | суффиксная ссылка. | - |
| - | int auto\_move[k]; | - | запоминание перехода автомата. | - |
| - | int par; | - | вершина-отец в дереве. | - |
| - | int64\_t suff\_flink ; | - | "Хорошая" суфф. Ссылка. | - |
| - | bool flag; | - | бит, указывающий на то, является ли наша вершина исходной строкой. | - |
| - | char symb; | - | символ на ребре от par к этой вершине. | - |
| - | int count\_rec; | - | количество исходящих дуг из вершины. | - |
| class Bohr | private | vector<bohr\_vrtx> bohr ; | - | Вектор вершин. | - |
| private | vector<string> pattern ; | - | Вектор паттернов. | - |
| private | vector<pair<int, int>> remove\_text ; | - | Вектор пар(индекс, длинна) нахождения в тексте паттернов для удаления из него. | - |
| public | char joker; | - | Хранит джокер. | - |
| public | bohr\_vrtx make\_bohr\_vrtx(int p, char c) | int p – номер вершины;  char c – символ вершины. | Передает номер отца и символ на ребре в боре. | Возвращает объект новой вершины. |
| private | int get\_suff\_link(int v, int count\_rec) | int v – номер вершины;  int count\_rec – счетчик рекурсии. | Переход по суф. ссылке. | Возвращает суффиксную ссылку. |
| public | Bohr() | - | Инициализация бора. | - |
| public | void add\_string\_to\_bohr(const string &s) | const string &s – добавляемая строка | Добавление строки в бор. | - |
| private | int get\_auto\_move(int v, char ch) | int v – номер вершины для перехода;  char ch – символ – куда переходим. | Функция автомата. | Возвращает индекс вершины после перехода автомата. |
| private | int get\_suff\_flink(int v, int count\_rec) | int v - номер вершины для перехода;  int count\_rec - счетчик рекурсии; | Нахождение "хорошей" суф. ссылки. | Возвращает найденную «хорошую» суффиксную ссылку. |
| private | void check(int v, int i) | int v – номер вершины для прверки;  int i – индекс элемента строки поиска. | Проверка подстроки на нахождение в боре. | - |
| public | void find\_all\_pos(const string &s) | const string &s – строка для поиска. | Нахождение всех вхождений паттернов в текст. | - |
| public | int max\_arc() | - | Нахождение максимального количества дуг в боре. | Возвращает максимальное количество дуг в боре. |
| public | string text\_without\_patterns(const string &s) | const string &s – строка для удаления. | Удаляет из текста все найденные паттерны. | Возвращает строку без входящих в нее паттернов. |
| public | void print\_bohr(int cur\_vrtx) | int cur\_vrtx – вершина для подсветки. | Распечатка бора. | - |
| private | void print\_next(int n, int count\_tab, int cur\_vrtx) | int n, int count\_tab, int cur\_vrtx | Вспомогательная функция для распечатки. | - |
| public | void print\_patterns() | - | Распечатка паттернов. | - |

**Тестирование 1.**

|  |  |  |
| --- | --- | --- |
| Номер  теста | Тест: | Результат: |
| 1 | QWERTYUI  8  QWE  QWER  QWERTY  QOKIL  POIK  TYUI  QWERKOLH  POKOJI | 1 1  1 2  1 3  5 6  max\_arc : 3  Text without patterns: |
| 2 | ASDFGHJKLERTY  5  ASD  ER  H  QWERTYUI  OPIPO | 1 1  6 3  10 2  max\_arc : 5  Text without patterns: FGJKLTY |
| 3 | AAAAAAAAAAAaaaa  2  aa  AA | 1 2  2 2  3 2  4 2  5 2  6 2  7 2  8 2  9 2  10 2  12 1  13 1  14 1  max\_arc : 2  Text without patterns: |

**Тестирование 2.**

|  |  |  |
| --- | --- | --- |
| Номер  теста | Тест: | Результат: |
| 1 | QWERTYUERY  R$  $ | 4  9  max\_arc : 1  Text without patterns: QWEYUE |
| 2 | ACT  A$  $ | 1  max\_arc : 1  Text without patterns: T |
| 3 | ACTACTTACATCATTATTCTTTAATC  AC$  $ | 1  4  8  max\_arc : 1  Text without patterns: TTCATTATTCTTTAATC |

**Вывод.**

В ходе выполнения лабораторной работы был изучен алгоритм АхоКорасик. Была написана реализация для поиска подстрок в заданном текст с использованием и без использования символов-джокеров. Были рассчитаны асимптотики работы алгоритма.

**Приложение 1. Код программы.**

* **5.1**[**.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <iostream>

#include <cstring>

#include <vector>

#include <windows.h>

using namespace std;

// Размер алфавита

const int k = 60;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

struct bohr\_vrtx

{

int next\_vrtx[k]; //next\_vrtx[i] — номер вершины, в которую мы придем по символу с номером i в алфавите

int pat\_num; //номер строки-образца, обозначаемого этой вершиной

int suff\_link; //suff\_link - суффиксная ссылка

int auto\_move[k]; //auto\_move - запоминание перехода автомата

int par; //par - вершина-отец в дереве

int64\_t suff\_flink; //suff\_flink - "хорошая" суф. ссылка

bool flag; //flag — бит, указывающий на то, является ли наша вершина исходной строкой

char symb; //символ на ребре от par к этой вершине

int count\_arc; //количество исхдящих дуг из вершины

};

class Bohr

{

vector<bohr\_vrtx> bohr;

vector<string> pattern;

vector<pair<int, int>> remove\_text;

//Функция добваления вершины в бор

bohr\_vrtx make\_bohr\_vrtx(int p, char c) //передаем номер отца и символ на ребре в боре

{

bohr\_vrtx v; //(255)=(2^8-1)=(все единицы в каждом байте памяти)=(-1 в дополнительном коде целого 4-байтного числа int)

memset(v.next\_vrtx, 255, sizeof(v.next\_vrtx));

memset(v.auto\_move, 255, sizeof(v.auto\_move));

v.flag = false; //По умолчанию: вершина - не сходная строка

v.suff\_link = -1; //изначально - суф. ссылки нет

v.par = p;

v.symb = c;

v.suff\_flink = -1;

v.count\_arc = 0;

return v;

}

public:

Bohr()

{

//добавляем единственную вершину - корень

bohr.push\_back(make\_bohr\_vrtx(0, '$'));

}

~Bohr() = default;

//Добавление строки в бор

void add\_string\_to\_bohr(const string &s)

{

int num = 0; //начинаем с корня

for (int i = 0; i < s.length(); i++)

{

cout << '\t' << s[i] << " - ";

int ch = s[i] - 'A'; //получаем номер в алфавите для символа строки

if (bohr[num].next\_vrtx[ch] == -1) //-1 - признак отсутствия ребра

{

cout << "символа в боре нет. Добавлим ребро для него." << endl;

bohr[num].count\_arc++;

bohr.push\_back(make\_bohr\_vrtx(num, ch)); //Создание вершины

bohr[num].next\_vrtx[ch] = bohr.size() - 1; //Установление ребра

}

else

cout << "символ был в боре. Переходим к нему." << endl;

num = bohr[num].next\_vrtx[ch]; //Переход в следующую вершину

}

cout << "\tПометим конец строки." << endl;

bohr[num].flag = true; //Пометка конца паттерна

pattern.push\_back(s);

bohr[num].pat\_num = pattern.size() - 1;

}

private:

//Переход по суф. ссылке

int get\_suff\_link(int v, int count\_rec)

{

for(int i = 0; i < count\_rec; i++)

cout << '\t';

cout <<"\tДля " << (v ? (char)(bohr[v].symb +'A') : '$') << " ";

if (bohr[v].suff\_link == -1) //если еще не считали

{

cout << "суфф. ссылка не определена." << endl;

for(int i = 0; i < count\_rec; i++)

cout << '\t';

if (v == 0 || bohr[v].par == 0) //если v - корень или предок v - корень

{

cout << "\tМы у корня. Идти не куда. Следовательно суфф. ссылка ведет в корень." << endl;

bohr[v].suff\_link = 0;

}else

{

cout << "\tСоздадим ее. Перейдем к родителю:" << endl;

bohr[v].suff\_link = get\_auto\_move(get\_suff\_link(bohr[v].par, count\_rec + 1), bohr[v].symb);

}

}else

cout << "суфф. ссылка уже была найдена. Она ведет к "<< (bohr[v].suff\_link ? (char)(bohr[bohr[v].suff\_link].symb +'A') : '$') << endl;

return bohr[v].suff\_link;

}

//Функция автомата

int get\_auto\_move(int v, char ch)

{

if (bohr[v].auto\_move[ch] == -1) //Если для нашего случая переход автомата еще не определен, то определим его

{

cout << "\tАвтомат перехода не был определен. Определим его:" << endl;

if (bohr[v].next\_vrtx[ch] != -1) //Если из вершины v есть путь в ch

{

cout << "\tБыла найдена дуга (" << (v ? (char)(bohr[v].symb +'A') : '$') <<", " << (char)(ch+'A') << ")" << endl;

cout << "\tТеперь автомат определен." << endl;

bohr[v].auto\_move[ch] = bohr[v].next\_vrtx[ch]; //Установим автомат перехода в нее

}else if (v == 0) //Если пришли в корень

{

cout << "\tДуга (" << (v ? (char)(bohr[v].symb +'A') : '$') <<", " << (char)(ch+'A') << ") не найдена" << endl;

cout << "\tПришли к корню. Теперь автомат определен." << endl;

bohr[v].auto\_move[ch] = 0; //Установим автомат перехода в корень

}else //Иначе

{

cout << "\tДуга (" << (v ? (char)(bohr[v].symb +'A') : '$') <<", " << (char)(ch+'A') << ") не найдена." << endl;

cout << "\tПереходим по суффиксной ссылке." << endl;

bohr[v].auto\_move[ch] = get\_auto\_move(get\_suff\_link(v, 0), ch); //Установим автомат перехода рекурсивно в соответствии с автоматом для вершины в которую ведет суфф. ссылка текушей вершины

}

}else

cout << "автомат перехода был определен.";

return bohr[v].auto\_move[ch];

}

// Нахождение "хорошей" суф. ссылки

int get\_suff\_flink(int v, int count\_rec)

{

if (bohr[v].suff\_flink == -1)

{

int u = get\_suff\_link(v, count\_rec); //Переход по суфф. ссылке

if (u == 0) //либо v - корень, либо суфф. ссылка v указывает на корень

bohr[v].suff\_flink = 0;

else

bohr[v].suff\_flink = (bohr[u].flag) ? u : get\_suff\_flink(u, count\_rec + 1); // В случае ненахождения конца паттерна, рекурсивный поиск

}

return bohr[v].suff\_flink;

}

// Проверка подстроки на нахождение в боре

void check(int v, int i)

{

for (int u = v; u != 0; u = get\_suff\_flink(u, 0)) //Переходы по суфф. ссылкам

{

if (bohr[u].flag)

{

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED);

cout << "Найдена подстрока: " << pattern[bohr[u].pat\_num] << endl;

cout << i - pattern[bohr[u].pat\_num].length() + 1 << " " << bohr[u].pat\_num + 1 << endl;

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

remove\_text.push\_back({i - pattern[bohr[u].pat\_num].length(), pattern[bohr[u].pat\_num].size()}); //Добавление в вектор для последующей обработки текста

}

}

}

public:

// Нахождение всех вхождений паттернов в текст

void find\_all\_pos(const string &s)

{

int u = 0;

for (int i = 0; i < s.length(); i++)

{

cout << endl << i+1<< ") "<< s[i] << " - ";

u = get\_auto\_move(u, s[i] - 'A'); //Переход автомата

cout << endl << "\tПроверка подстроки на нахождение в боре:" << endl;

check(u, i + 1); // Проверка подстроки на нахождение в боре

print\_bohr(u);

}

}

// Нахождение максиматльного кличества дуг в боре

int max\_arc()

{

int i\_max\_count\_arc = 0;

for (int i = 0; i < bohr.size(); i++)

if (bohr[i\_max\_count\_arc].count\_arc < bohr[i].count\_arc)

i\_max\_count\_arc = i;

return bohr[i\_max\_count\_arc].count\_arc;

}

// Удаляет из текста все найденные паттерны

string text\_without\_patterns(const string &s)

{

string final\_text;

bool remove\_ind[s.length()]; //Массив пометок об удалении буквы из строки

memset(remove\_ind, true, s.length());

for (int i = 0; i < remove\_text.size(); i++) //Область паттерна помечена false

memset(remove\_ind + remove\_text[i].first, false, remove\_text[i].second);

for (int i = 0; i < s.length(); i++)

if (remove\_ind[i]) //Для всех не помеченных букв: добавить к строке

final\_text += s[i];

return final\_text;

}

//Распечатка бора

void print\_bohr(int cur\_vrtx)

{

if(cur\_vrtx != -1)

if(bohr[cur\_vrtx].suff\_link == 0)

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << bohr[0].symb << endl;

if(cur\_vrtx!= -1)

if(bohr[cur\_vrtx].suff\_link == 0)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

print\_next(0, 0, cur\_vrtx);

}

private:

void print\_next(int n, int count\_tab, int cur\_vrtx)

{

int next;

int next\_count\_tab = 0;

for (int i = 0; i < k; i++)

{

next = bohr[n].next\_vrtx[i];

if (next != -1)

{

cout << " - ";

if(next == cur\_vrtx)

SetConsoleTextAttribute(hConsole, FOREGROUND\_GREEN);

if(cur\_vrtx != -1)

if(bohr[cur\_vrtx].suff\_link == next)

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << (char)(bohr[next].symb + 'A');

if(next == cur\_vrtx)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

if(cur\_vrtx!= -1)

if(bohr[cur\_vrtx].suff\_link == next)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

if (bohr[next].flag)

cout << "(" << bohr[next].pat\_num + 1 << ")";

print\_next(next, count\_tab + 1, cur\_vrtx);

next\_count\_tab++;

if (bohr[next].count\_arc == 0)

cout << endl;

if (next\_count\_tab != bohr[n].count\_arc)

for (int p = 0; p < count\_tab; p++)

cout << " ^";

}

}

};

public:

//Распечатка паттернов

void print\_patterns()

{

for (int i = 0; i < pattern.size(); i++)

cout << i + 1 << ") " << pattern[i] << endl;

}

};

int main()

{

setlocale(LC\_ALL, "rus");

string T, pattern;

int n;

cout << "Введите текст и количество паттернов:" << endl;

cin >> T >> n;

// Инициализация бора

cout << "Инициализация бора:" << endl;

Bohr bohr;

cout << "Добавление паттернов в бор:" << endl;

for (int i = 0; i < n; i++)

{

cout << i + 1 << ") ";

cin >> pattern;

// Добавление паттернов в бор

bohr.add\_string\_to\_bohr(pattern);

bohr.print\_bohr(-1);

}

cout << "Итоговое предсталение бора для списка паттернов:" << endl;

bohr.print\_patterns();

bohr.print\_bohr(-1);

cout << endl << endl << "Нахождение всех паттернов в строке " << T << ":" << endl;

bohr.find\_all\_pos(T);

cout << "max\_arc : " << bohr.max\_arc() << endl;

cout << "Text without patterns: " << bohr.text\_without\_patterns(T) << endl;

}

* **5.2**[**.c**](https://github.com/makometr/AiSD/pull/37/files#diff-240c2792fd92b595432e18629f6e16b6)**pp**

#include <iostream>

#include <cstring>

#include <vector>

#include <windows.h>

using namespace std;

// Размер алфавита

const int k = 90;

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

struct bohr\_vrtx

{

int next\_vrtx[k]; //next\_vrtx[i] — номер вершины, в которую мы придем по символу с номером i в алфавите

int pat\_num; //номер строки-образца, обозначаемого этой вершиной

int suff\_link; //suff\_link - суффиксная ссылка

int auto\_move[k]; //auto\_move - запоминание перехода автомата

int par; //par - вершина-отец в дереве

int64\_t suff\_flink; //suff\_flink - "хорошая" суф. ссылка

bool flag; //flag — бит, указывающий на то, является ли наша вершина исходной строкой

char symb; //символ на ребре от par к этой вершине

int count\_arc; //количество исхдящих дуг из вершины

};

class Bohr

{

vector<bohr\_vrtx> bohr;

vector<string> pattern;

vector<pair<int, int>> remove\_text;

public:

char joker;

private:

//Функция добваления вершины в бор

bohr\_vrtx make\_bohr\_vrtx(int p, char c) //передаем номер отца и символ на ребре в боре

{

bohr\_vrtx v; //(255)=(2^8-1)=(все единицы в каждом байте памяти)=(-1 в дополнительном коде целого 4-байтного числа int)

memset(v.next\_vrtx, 255, sizeof(v.next\_vrtx));

memset(v.auto\_move, 255, sizeof(v.auto\_move));

v.flag = false; //По умолчанию: вершина - не сходная строка

v.suff\_link = -1; //изначально - суф. ссылки нет

v.par = p;

v.symb = c;

v.suff\_flink = -1;

v.count\_arc = 0;

return v;

}

public:

Bohr()

{

//добавляем единственную вершину - корень

bohr.push\_back(make\_bohr\_vrtx(0, '$'));

}

//Добавление строки в бор

void add\_string\_to\_bohr(const string &s)

{

int num = 0; //начинаем с корня

for (int i = 0; i < s.length(); i++)

{

cout << '\t' << s[i] << " - ";

int ch = s[i] - ' '; //получаем номер в алфавите для символа строки

if (bohr[num].next\_vrtx[ch] == -1) //-1 - признак отсутствия ребра

{

cout << "символа в боре нет. Добавлим ребро для него." << endl;

bohr[num].count\_arc++;

bohr.push\_back(make\_bohr\_vrtx(num, ch)); //Создание вершины

bohr[num].next\_vrtx[ch] = bohr.size() - 1; //Установление ребра

}

else

cout << "символ был в боре. Переходим к нему." << endl;

num = bohr[num].next\_vrtx[ch]; //Переход в следующую вершину

}

cout << "\tПометим конец строки." << endl;

bohr[num].flag = true; //Пометка конца паттерна

pattern.push\_back(s);

bohr[num].pat\_num = pattern.size() - 1;

}

private:

//Переход по суф. ссылке

int get\_suff\_link(int v, int count\_rec)

{

for(int i = 0; i < count\_rec; i++)

cout << '\t';

cout <<"\tДля " << (v ? (char)(bohr[v].symb +' ') : '$') << " ";

if (bohr[v].suff\_link == -1) //если еще не считали

{

cout << "суфф. ссылка не определена." << endl;

for(int i = 0; i < count\_rec; i++)

cout << '\t';

if (v == 0 || bohr[v].par == 0) //если v - корень или предок v - корень

{

cout << "\tМы у корня. Идти не куда. Следовательно суфф. ссылка ведет в корень." << endl;

bohr[v].suff\_link = 0;

}else

{

cout << "\tСоздадим ее. Перейдем к родителю:" << endl;

bohr[v].suff\_link = get\_auto\_move(get\_suff\_link(bohr[v].par, count\_rec + 1), bohr[v].symb);

}

}else

cout << "суфф. ссылка уже была найдена. Она ведет к "<< (bohr[v].suff\_link ? (char)(bohr[bohr[v].suff\_link].symb +' ') : '$') << endl;

return bohr[v].suff\_link;

}

//Функция автомата

int get\_auto\_move(int v, char ch)

{

if (bohr[v].auto\_move[ch] == -1) //Если для нашего случая переход автомата еще не определен, то определим его

{

cout << "\tАвтомат перехода не был определен. Определим его:" << endl;

if (bohr[v].next\_vrtx[ch] != -1) //Если из вершины v есть путь в ch

{

cout << "\tБыла найдена дуга (" << (v ? (char)(bohr[v].symb +' ') : '$') <<", " << (char)(ch+' ') << ")" << endl;

cout << "\tТеперь автомат определен." << endl;

bohr[v].auto\_move[ch] = bohr[v].next\_vrtx[ch]; //Установим автомат перехода в нее

}else if (bohr[v].next\_vrtx[joker] != -1) //Или есть джокер

{

cout << "\tБыла найдена дуга (" << (v ? (char)(bohr[v].symb +' ') : '$') <<", " << (char)(joker + ' ') << ")" << endl;

cout << "\tТеперь автомат определен." << endl;

bohr[v].auto\_move[ch] = bohr[v].next\_vrtx[joker];

}else if (v == 0) //Если пришли в корень

{

cout << "\tДуга (" << (v ? (char)(bohr[v].symb +' ') : '$') <<", " << (char)(ch+' ') << ") не найдена" << endl;

cout << "\tПришли к корню. Теперь автомат определен." << endl;

bohr[v].auto\_move[ch] = 0; //Установим автомат перехода в корень

}else //Иначе

{

cout << "\tДуга (" << (v ? (char)(bohr[v].symb +' ') : '$') <<", " << (char)(ch+' ') << ") не найдена." << endl;

cout << "\tПереходим по суффиксной ссылке." << endl;

bohr[v].auto\_move[ch] = get\_auto\_move(get\_suff\_link(v, 0), ch); //Установим автомат перехода рекурсивно в соответствии с автоматом для вершины в которую ведет суфф. ссылка текушей вершины

}

}else

cout << "автомат перехода был определен.";

return bohr[v].auto\_move[ch];

}

// Нахождение "хорошей" суф. ссылки

int get\_suff\_flink(int v, int count\_rec)

{

if (bohr[v].suff\_flink == -1)

{

int u = get\_suff\_link(v, count\_rec); //Переход по суфф. ссылке

if (u == 0) //либо v - корень, либо суфф. ссылка v указывает на корень

bohr[v].suff\_flink = 0;

else

bohr[v].suff\_flink = (bohr[u].flag) ? u : get\_suff\_flink(u, count\_rec + 1); // В случае ненахождения конца паттерна, рекурсивный поиск

}

return bohr[v].suff\_flink;

}

// Проверка подстроки на нахождение в боре

void check(int v, int i)

{

for (int u = v; u != 0; u = get\_suff\_flink(u, 0)) //Переходы по суфф. ссылкам

{

if (bohr[u].flag)

{

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED);

cout << "Найдена подстрока: " << pattern[bohr[u].pat\_num] << endl;

cout << i - pattern[bohr[u].pat\_num].length() + 1 << " " << bohr[u].pat\_num + 1 << endl;

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

remove\_text.push\_back({i - pattern[bohr[u].pat\_num].length(), pattern[bohr[u].pat\_num].size()}); //Добавление в вектор для последующей обработки текста

}

}

}

public:

// Нахождение всех вхождений паттернов в текст

void find\_all\_pos(const string &s)

{

int u = 0;

for (int i = 0; i < s.length(); i++)

{

cout << endl << i+1<< ") "<< s[i] << " - ";

u = get\_auto\_move(u, s[i] - ' '); //Переход автомата

cout << endl << "\tПроверка подстроки на нахождение в боре:" << endl;

check(u, i + 1); // Проверка подстроки на нахождение в боре

print\_bohr(u);

}

}

// Нахождение максиматльного кличества дуг в боре

int max\_arc()

{

int i\_max\_count\_arc = 0;

for (int i = 0; i < bohr.size(); i++)

if (bohr[i\_max\_count\_arc].count\_arc < bohr[i].count\_arc)

i\_max\_count\_arc = i;

return bohr[i\_max\_count\_arc].count\_arc;

}

// Удаляет из текста все найденные паттерны

string text\_without\_patterns(const string &s)

{

string final\_text;

bool remove\_ind[s.length()];

memset(remove\_ind, true, s.length());

for (int i = 0; i < remove\_text.size(); i++)

memset(remove\_ind + remove\_text[i].first, false, remove\_text[i].second);

for (int i = 0; i < s.length(); i++)

if (remove\_ind[i])

final\_text += s[i];

return final\_text;

}

//Распечатка бора

void print\_bohr(int cur\_vrtx)

{

if(cur\_vrtx != -1)

if(bohr[cur\_vrtx].suff\_link == 0)

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << bohr[0].symb << endl;

if(cur\_vrtx!= -1)

if(bohr[cur\_vrtx].suff\_link == 0)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

print\_next(0, 0, cur\_vrtx);

}

private:

void print\_next(int n, int count\_tab, int cur\_vrtx)

{

int next;

int next\_count\_tab = 0;

for (int i = 0; i < k; i++)

{

next = bohr[n].next\_vrtx[i];

if (next != -1)

{

cout << " - ";

if(next == cur\_vrtx)

SetConsoleTextAttribute(hConsole, FOREGROUND\_GREEN);

if(cur\_vrtx != -1)

if(bohr[cur\_vrtx].suff\_link == next)

SetConsoleTextAttribute(hConsole, FOREGROUND\_RED | FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

cout << (char)(bohr[next].symb + ' ');

if(next == cur\_vrtx)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

if(cur\_vrtx!= -1)

if(bohr[cur\_vrtx].suff\_link == next)

SetConsoleTextAttribute(hConsole, FOREGROUND\_BLUE | FOREGROUND\_GREEN |FOREGROUND\_RED);

if (bohr[next].flag)

cout << "(" << bohr[next].pat\_num + 1 << ")";

print\_next(next, count\_tab + 1, cur\_vrtx);

next\_count\_tab++;

if (bohr[next].count\_arc == 0)

cout << endl;

if (next\_count\_tab != bohr[n].count\_arc)

for (int p = 0; p < count\_tab; p++)

cout << " ^";

}

}

};

public:

//Распечатка паттернов

void print\_patterns()

{

for (int i = 0; i < pattern.size(); i++)

cout << i + 1 << ") " << pattern[i] << endl;

}

};

int main()

{

setlocale(LC\_ALL, "rus");

string T, pattern;

char joker;

cout << "Введите текст, паттерн и джокер:" << endl;

cin >> T >> pattern >> joker;

joker -= ' ';

// Инициализация бора

cout << "Инициализация бора:" << endl;

Bohr bohr;

//Инициализация джокера

bohr.joker = joker;

bohr.add\_string\_to\_bohr(pattern);

cout << "Итоговое предсталение бора для списка паттернов:" << endl;

bohr.print\_patterns();

bohr.print\_bohr(-1);

cout << endl << endl << "Нахождение всех паттернов в строке " << T << ":" << endl;

// Нахождение всех паттернов в строке T

bohr.find\_all\_pos(T);

cout << "max\_arc : " << bohr.max\_arc() << endl;

cout << "Text without patterns: " << bohr.text\_without\_patterns(T) << endl;

}