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

Студент гр. 8303 _____ Кибардин А.Б.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Изучение алгоритма Ахо-Корасика для поиска всех вхождений каждой строки из данного набора.

Формулировка задачи 1.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T,$1 \le |T| \le 100000$).

Вторая - число n ($1 \le n \le 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p1,\ldots,pn\}$ $1 \le |pi| \le 75$

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:

CCCA
1
CC
Sample Output:
1 1
2 1

Формулировка задачи 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 a*b*??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 ($T, 1 \le |T| \le 100000$)
Шаблон ($P, 1 \le |P| \le 40$)
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

Индивидуализация.

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

Описание алгоритма Ахо-Корасик.

На вход подается набор строк шаблонов, по которым строится бор. Поочередно рассматривается каждый символ строки. Если в боре существует переход по данному символу, переходим в следующее состояние. При отсутствии перехода по данному символу, создается новая вершина и добавляется в бор.

Далее происходит обработка текстовой строки. В автомат, полученный из построенного бора, подается символ. При наличии перехода по текущему символу, автомат переходит в следующее состояние, счетчик текста инкрементируется. При отсутствии перехода, рассматриваются ребра бора. При наличии ребра перехода, добавляем в множество вершин перехода по данному символу это ребро. В противном случае рассматривается переход по суффиксной ссылке.

При достижении терминальной вершины, в вектор результата записывается индекс шаблона и вычисляется его позиция в тексте.

Суффиксная ссылка строится следующим образом. При отсутствии ссылки производится проверка, является ли данная вершина корнем, если она таковой является, ссылка равняется 0, то есть ссылка корня будет ссылаться на корень. Если вершина не является корнем, то производится переход по суффиксной ссылке предка. При отсутствии ссылки у предка рекурсивно производится вычисление ссылки для него.

Для вывода вывода пересекающихся образцов вычисляется позиция последнего символа накладываемого шаблона. Если данная позиция равна либо больше индекса следующего шаблона, то выводится сообщение о пересечении.

Сложности алгоритма.

Сложность алгоритма по операциям O((n+m)*log(k) + t), где n – количество символов в тексте, m – сумма длин шаблонов, k — размер алфавита, t — длина всех вхождений строк шаблонов.

Сложность алгоритма по памяти O(m + n), где n – количество символов в тексте, m – сумма длин шаблонов

Описание поиска с джокером.

Алгоритм инициализирует вектор для записи результата длины равной длине текста. Строка с джокерами разбивается на подстроки без них. Так же запоминаются из позиции в исходной строке. Далее с помощью алгоритма Ахо-Корасик ищуются все вхождения подстрок в тексте. При нахождении очередной подстроки вычисляется индекс в позиции вектора. Если индекс попадается в заданный диапазон вектора, то счетчик в векторе по данному индексу увеличивается на единицу.

Далее в векторе результата производится поиск на совпадение полей вектора с итоговым количеством подстрок. Если их числа совпадают, то в тексте с данного индекса начинается строка с джокерами, выводится сообщение о нахождении.

Сложности алгоритма.

Сложность алгоритма по операциям O((n+m)*log(k) + t + n), где n – количество символов в тексте, m – сумма длин шаблонов, k — размер алфавита, t — длина всех вхождений строк шаблонов.

Сложность алгоритма по памяти O(m + 2n + k), где n – количество символов в тексте, m – сумма длин шаблонов, k — количество подстрок.

Описание структур Ахо-Корасик.

```
struct Vertex {
std::map<char,int> next;
std::map<char,int> go;
bool leaf;
int parent;
```

```
    char parentChar;
    int link;

    int patternNumber;
};
```

Структура для хранения вершин бора.

next – контейнер для хранения переходов по ребрам в боре.

go – контейнер для хранения переходов в автомате.

leaf – флаг для проверки является ли данная вершина терминальной.

parent – индекс родительской вершины.

parentChar – символ родительской вершины.

link – суффискная ссылка.

patternNumber – номер паттерна, который находится, если вершина терминальная.

```
class Trie
{
    std::vector<Vertex> vertices;
    std::vector<std::string> patterns;
    std::vector<std::pair<int,int>> result;
}
```

Структура для хранения бора
verstices — вектор вершин бора.
patterns — вектор для хранения шаблонов
result — вектор для записи результата работы алгоритма — номер паттерна и его позиции в тексте.

Описание функций Ахо-Корасик.

bool myCmp(std::pair<int, int> i, std::pair<int, int> j) – функция для сортировки вектора результата. Возвращается true, если первый аргумент меньше второго.

i, j – значения вектора результата.

void init() - метод инициализации бора.

void addString (const std::string & s) – метод добавления новой строки в бор.
s – добавляемая строка.

int go (int v, char c) – метод перехода автомата в новое состояние. Возвращает номер нового состояния.
v – текущее состояние.
c – символ перехода.

int getLink (int v) – метод получения суффиксной ссылки. Возвращается номер вершины, на которую указывает ссылка.
v – текущая вершина.

void ahoCorasick(std::string& tex
t) – метод, реализующий алгоритм Ахо-Корасик.
text – текст, в котором ищутся совпадения.

void printResult() - метод печати результата работы алгоритма.

Описание структур поиска с джокером.

```
struct Vertex {
std::map<char,int> next;
std::map<char,int> go;
bool leaf;
int parent;
char parentChar;
int link;

std::vector<int> listPatterns;
```

};

Структура для хранения вершин бора.

next – контейнер для хранения переходов по ребрам в боре.

go – контейнер для хранения переходов в автомате.

leaf – флаг для проверки является ли данная вершина терминальной.

parent – индекс родительской вершины.

parentChar – символ родительской вершины.

link – суффискная ссылка.

listPatterns – контейнер для хранения множества индексов подстрок.

```
class Trie
{
   std::vector<Vertex> vertices;
   std::vector<std::string> patterns;
   std::vector<int> C;
        std::vector<int> patternsIndex;



}
```

Структура для хранения бора
verstices — вектор вершин бора.
patterns — вектор для хранения шаблонов
C — вектор для записи результата работы алгоритма — номер паттерна и его позиции в тексте.

patternsIndex — вектор для хранения индексов начала подстрок в исходной строке с джокерами.

Описание функций поиска с джокером.

void init() - метод инициализации бора.

void addString (const std::string & s) – метод добавления новой строки в бор.

s – добавляемая строка.

int go (int v, char c) – метод перехода автомата в новое состояние. Возвращает номер нового состояния.
v – текущее состояние.
c – символ перехода.

int getLink (int v) – метод получения суффиксной ссылки. Возвращается номер вершины, на которую указывает ссылка.
v – текущая вершина.

void ahoCorasickJoker(std::string& text, std::string& pattern) – метод, поиск с джокером на основе алгоритма Ахо-Корасик.
text – текст, в котором ищутся совпадения.
pattern – исходная строка с джокером.

void printResult() - метод печати результата работы алгоритма.

void splitString(std::string& pattern, char joker) – метод разбиения строки с джокером.
pattern – исходная строка с джокером.
joker – символ джокера.

Тестирование.

Алгоритм Ахо-Корасик

```
ACACGTNNNNAGT
3
G
AG
NNN
/home/anton/ClionProjects/piaa_5/cmake-build-debug/piaa_5
------------------
add new string in trie: G
symbol G
add new vertex from 0(G)
------------------
add new string in trie: AG
symbol A
add new vertex from 0(A)
symbol G
add new vertex from 2(G)
------------------
add new string in trie: NNN
symbol N
add new vertex from 0(N)
symbol N
add new vertex from 4(N)
symbol N
add new vertex from 5(N)
------------------
------------------
current symbol: A
next state: 2
state: 2
------------------
get link to root:
get link to state: 0
------------------
current symbol: C
------------------
get link to state: 0
------------------
current symbol: C
go through link: 0
next state: 0
go through link: 0
next state: 0
```

```
state: 0
------------------
current symbol: A
next state: 2
state: 2
------------------
get link to state: 0
------------------
current symbol: C
next state: 0
state: 0
------------------
current symbol: G
next state: 1
state: 1
found pattern #3: G
------------------
------------------
get link to root:
get link to state: 0
------------------
current symbol: T
------------------
get link to state: 0
------------------
current symbol: T
go through link: 0
next state: 0
go through link: 0
next state: 0
state: 0
------------------
current symbol: N
next state: 4
state: 4
------------------
get link to root:
```

```
get link to state: 0
------------------
current symbol: N
next state: 5
state: 5
------------------
get link through parent:
------------------
get link to state: 0
------------------
current symbol: N
next state: 4
get link to state: 4
------------------
get link to state: 0
------------------
current symbol: N
next state: 6
state: 6
found pattern #5: NNN
------------------
------------------
get link through parent:
------------------
get link to state: 4
------------------
current symbol: N
next state: 5
get link to state: 5
------------------
get link to state: 4
------------------
get link to state: 0
------------------
current symbol: N
------------------
get link to state: 5
```

```
current symbol: N
next state: 6
go through link: 6
next state: 6
state: 6
found pattern #6: NNN
------------------
------------------
get link to state: 5
------------------
get link to state: 4
------------------
get link to state: 0
------------------
current symbol: A
------------------
get link to state: 5
------------------
current symbol: A
------------------
get link to state: 4
------------------
current symbol: A
------------------
get link to state: 0
------------------
current symbol: A
next state: 2
go through link: 2
next state: 2
go through link: 2
next state: 2
go through link: 2
next state: 2
state: 2
------------------
get link to state: 0
```

```
------------------
current symbol: G
next state: 3
state: 3
found pattern #9: AG
------------------
------------------
get link through parent:
------------------
get link to state: 0
------------------
current symbol: G
next state: 1
get link to state: 1
found pattern #10: G
------------------
------------------
get link to state: 0
------------------
current symbol: T
------------------
get link to state: 1
------------------
current symbol: T
next state: 0
go through link: 0
next state: 0
state: 0
result:
5 1
7 3
8 3
11 2
12 1
#verties: 7
crossing of patterns NNN and NNN. #3,#3 at positions 7,8
crossing of patterns AG and G. #2,#1 at positions 11,12
```

```
ABCACBABACBAABACBCA
7
AB
ACA
ACB
ACC
BAB
BCA
AC
```

```
/home/anton/ClionProjects/piaa_5/cmake-build-debug/piaa_5
result:
1 1
2 6
4 7
4 3
6 5
7 1
9 7
9 3
13 1
15 7
15 3
17 6

Process finished with exit code 0
```

```
CCCA
1
CC
```

```
/home/anton/ClionProjects/piaa_5/cmake-b
result:
1 1
2 1

Process finished with exit code 0
```

Поиск с джокером

```
abcbabcbbbc
axcxxbc
x
```

```
------------------
add new string in trie: a
symbol a
add new vertex from 0(a)
------------------
add new string in trie: c
symbol c
add new vertex from 0(c)
------------------
add new string in trie: bc
symbol b
add new vertex from 0(b)
symbol c
add new vertex from 3(c)
------------------
------------------
current symbol: a
next state: 1
state: 1
found pattern
match at position: 0
------------------
------------------
get link to root:
get link to state: 0
------------------
current symbol: b
------------------
get link to state: 0
------------------
current symbol: b
next state: 3
```

```
go through link: 3
next state: 3
state: 3
------------------
get link to root:
get link to state: 0
------------------
current symbol: c
next state: 4
state: 4
found pattern
------------------
------------------
get link through parent:
------------------
get link to state: 0
------------------
current symbol: c
next state: 2
get link to state: 2
found pattern
match at position: 0
------------------
------------------
get link to root:
get link to state: 0
------------------
current symbol: b
------------------
get link to state: 2
------------------
current symbol: b
------------------
```

```
get link to state: 0
------------------
current symbol: b
next state: 3
go through link: 3
next state: 3
go through link: 3
next state: 3
state: 3
------------------
get link to state: 0
------------------
current symbol: a
------------------
get link to state: 0
------------------
current symbol: a
next state: 1
go through link: 1
next state: 1
state: 1
found pattern
match at position: 4
------------------
------------------
get link to state: 0
------------------
current symbol: b
next state: 3
state: 3
------------------
get link to state: 0
------------------
```

```
current symbol: c
next state: 4
state: 4
found pattern
match at position: 0
------------------
------------------
get link to state: 2
found pattern
match at position: 4
------------------
------------------
get link to state: 0
------------------
current symbol: b
next state: 3
state: 3
------------------
get link to state: 0
------------------
current symbol: b
------------------
get link to state: 0
------------------
current symbol: b
next state: 3
go through link: 3
next state: 3
state: 3
------------------
get link to state: 0
------------------
current symbol: b
```

```
next state: 3
state: 3
------------------
get link to state: 0
------------------
current symbol: c
next state: 4
state: 4
found pattern
match at position: 4
------------------
------------------
get link to state: 2
found pattern
------------------
------------------
get link to state: 0
result:
1
5

Process finished with exit code 0
```

```
ACTANCA
A$$A$
$
```

```
1

Process finished with exit code 0
```

```
bbebbeaas
$b$
$
```

```
1
3
4

Process finished with exit code 0
```

Вывод.

В ходе выполнения лабораторной работы были получены знания по работе с алгоритмом Ахо-Корасик. Так же были реализованы структуры вершин бора, класс бора, алгоритм Ахо-Корасик для поиска всех вхождений шаблонов в тексте и алгоритм поиска с джокером.

Приложение А. Исходный код.

ahoCorasick.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>
/*
 * Вариант 2. Подсчитать количество вершин в автомате; вывести список найденных образцов, имеющих пересечения с другими найденными образцами в строке поиска.
*/
//#define debug
bool myCmp(std::pair<int, int> i, std::pair<int, int> j)
{
    return i.second < j.second;
}

struct Vertex {
    std::map<char,int> next;              // множество ребер бора
    std::map<char,int> go;                // состояния автомата
    bool leaf;                            // флаг для проверки является ли вершина терминальной
    int parent;                           // индекс родительской вершины
    char parentChar;                      // символ родительской вершины
    int link;                             // суффиксная ссылка

    int patternNumber;                    // номер шаблона в терминальной вершине
};

class Trie
{
    std::vector<Vertex> vertices;         // контейнер для хранения бор
    std::vector<std::string> patterns;    // множество подстрок
    std::vector<std::pair<int,int>> result; // контейнер для записи результата
public:
    void init() {                         // инициализация бора
        Vertex head;
        head.parent = head.link = -1;
        head.leaf = false;
        vertices.push_back(head);
    }

    void addString (const std::string & s) {
#ifdef debug
        std::cout << "------------------\n";
#endif
#ifdef debug
        std::cout << "add new string in trie: " << s << std::endl;
#endif
        patterns.push_back(s);
        int v = 0;
        for (auto c : s) {
#ifdef debug
            std::cout << "symbol " << c << " \n";
#endif
            if (vertices[v].next.find(c) == vertices[v].next.end()) {          // если вершины в боре нет, добавляем
#ifdef debug
                std::cout << "add new vertex from " << v << "(" << c <<") \n";
#endif
                Vertex buffer;
```

```
            buffer.leaf = false;
            buffer.link = -1;
            buffer.parent = v;
            buffer.parentChar = c;

            vertices.push_back(buffer);
            vertices[v].next[c] = vertices.size() - 1;
         }
         v = vertices[v].next[c];
      }
      vertices[v].leaf = true;                                  // помечаем вершну как
терминальную
      vertices[v].patternNumber = patterns.size() - 1;          // добавляем индекс
шаблона
   }

   int go (int v, char c) {                                     // метод перехода в новое состояние
#ifdef debug
      std::cout << "------------------\n";
      std::cout << "current symbol: " << c << "\n";
#endif
      if (vertices[v].go.find(c) == vertices[v].go.end())       // переход в новое
состояние, при отсутствии
      {                                                         // в множестве переходов текущего
         if (vertices[v].next.find(c) != vertices[v].next.end())
         {
            vertices[v].go[c] = vertices[v].next[c];

         }
         else
         {
            vertices[v].go[c] = v==0 ? 0 : go (getLink(v), c);  // переход по ссылке
#ifdef debug
            std::cout << "go through link: " << vertices[v].go[c] <<" \n";
#endif
         }
      }
#ifdef debug
      std::cout << "next state: " << vertices[v].go[c] <<" \n";
#endif
      return vertices[v].go[c];
   }

   int getLink (int v) {                                        // метод получения суффиксной
ссылки
#ifdef debug
      std::cout << "------------------\n";
#endif

      if (vertices[v].link == -1)                               // при отстутствии ссылки вычисляем
ее
      {
         if (v == 0 || vertices[v].parent == 0)                 // добавлем ссылку на корень
         {
#ifdef debug
            std::cout << "get link to root: " <<" \n";
#endif
            vertices[v].link = 0;
         }
         else                                                   // вычисление ссылки через родительскую
вершину
```

```
        {
#ifdef debug
            std::cout << "get link through parent: " <<" \n";
#endif
            vertices[v].link = go (getLink(vertices[v].parent), vertices[v].parentChar);
        }
    }
#ifdef debug
    std::cout << "get link to state: " << vertices[v].link <<" \n";
#endif
    return vertices[v].link;
}

void ahoCorasick(std::string& text)
{
#ifdef debug
    std::cout << "------------------\n";
#endif
    int state = 0;
    for(int i = 0; i < text.size();i++)
    {
        state = go(state, text[i]);                                 // переход в новое состояние
#ifdef debug
        std::cout << "state: " << state <<" \n";
#endif
        for(size_t tmp = state; tmp != 0; tmp = getLink(tmp))       // цикл для поиска всех возможных
        {                                                           // вхождений
            if(vertices[tmp].leaf)
            {
                std::pair<int,int> buffer;                          // добавление в результат
                buffer.first = vertices[tmp].patternNumber;         // найденного шаблона и его
                buffer.second = i - patterns[buffer.first].size();  // позиции в тексте
                result.push_back(buffer);
#ifdef debug
                std::cout << "found pattern #" << buffer.second << ": " << patterns[buffer.first] <<" \n";
#endif
#ifdef debug
                std::cout << "------------------\n";
#endif
            }
        }
    }
}

void printResult()
{
    std::cout << "result: " <<std::endl;
    std::sort(result.begin(), result.end(), myCmp);
    for(auto iter : result)
    {
        std::cout << iter.second + 2 << " " << iter.first + 1 << std::endl;
    }
#ifdef debug
    std::cout << "#verties: " << vertices.size() << std::endl;
    for(size_t i = 0 ; i < result.size() - 1; i++) {
        for(size_t j = i + 1 ; j < result.size(); j++)
        {
            size_t first, second;
            first = patterns[result[i].first].size() - 1 + result[i].second;
```

```
                second =  result[j].second;
                if(first >= second)
                {
                    std::cout << "crossing of patterns " << patterns[result[i].first] << " and " <<
patterns[result[j].first] << ". #" << result[i].first + 1<< ",#" << result[j].first + 1 << " at
positions " << result[i].second + 2 << "," << result[j].second + 2 << "\n";
                }
            }
        }
#endif
    }
};

int main() {
    std::freopen("test", "r", stdin);
    Trie trie;
    trie.init();
    std::string text, pattern;
    int n = 0;
    std::cin >> text;
    std::cin >> n;
    for(int i = 0; i < n ;i++)
    {
        std::cin >> pattern;
        trie.addString(pattern);
    }

    trie.ahoCorasick(text);
    trie.printResult();

    return 0;
}
```

joker.cpp

```
#include <iostream>
#include <map>
#include <vector>
//#define debug

struct Vertex {
    std::map<char,int> next;              // множество ребер бора
    std::map<char,int> go;                // состояния автомата
    bool leaf;                            // флаг для проверки является ли вершина терминальной
    int parent;                           // индекс родительской вершины
    char parentChar;                      // символ родительской вершины
    int link;                             // суффиксная ссылка

    std::vector<int> listPatterns;        // множество индексов шаблонов
};

class Trie
{
    std::vector<Vertex> vertices;         // контейнер для хранения бор
    std::vector<std::string> patterns;    // множество подстрок

    std::vector<int> C;                   // контейнер для записи результата
    std::vector<int> patternsIndex;       // индексы вхождения подстрок в строке с
джокером
public:
```

```
    void init() {                       // инициализация бора
        Vertex head;
        head.parent = head.link = -1;
        head.leaf = false;
        vertices.push_back(head);
    }

    void addString (std::string s) {
#ifdef debug
        std::cout << "------------------\n";
#endif
#ifdef debug
        std::cout << "add new string in trie: " << s << std::endl;
#endif
        patterns.push_back(s);
        int v = 0;
        for (auto c : s) {
#ifdef debug
            std::cout << "symbol " << c << " \n";
#endif
            if (vertices[v].next.find(c) == vertices[v].next.end()) {           // если вершины в боре
нет, добавляем
#ifdef debug
                std::cout << "add new vertex from " << v << "(" << c <<") \n";
#endif
                Vertex buffer;
                buffer.leaf = false;
                buffer.link = -1;
                buffer.parent = v;
                buffer.parentChar = c;

                vertices.push_back(buffer);
                vertices[v].next[c] = vertices.size() - 1;
            }
            v = vertices[v].next[c];
        }
        vertices[v].leaf = true;                                        // помечаем вершну как
терминальную
        vertices[v].listPatterns.push_back(patterns.size() - 1);            // добавляем индекс
паттерна
    }

    int go (int v, char c) {                                            // метод перехода в новое состояние
#ifdef debug
        std::cout << "------------------\n";
        std::cout << "current symbol: " << c << "\n";
#endif
        if (vertices[v].go.find(c) == vertices[v].go.end())                 // переход в новое
состояние, при отсутствии
        {                                                               // в множестве переходов текущего
            if (vertices[v].next.find(c) != vertices[v].next.end())
            {
                vertices[v].go[c] = vertices[v].next[c];

            }
            else
            {
                vertices[v].go[c] = v==0 ? 0 : go (getLink(v), c);          // переход по ссылке
#ifdef debug
                std::cout << "go through link: " << vertices[v].go[c] <<" \n";
#endif
```

```
        }
      }
#ifdef debug
      std::cout << "next state: " << vertices[v].go[c] <<" \n";
#endif
      return vertices[v].go[c];
   }

   int getLink (int v) {                                    // метод получения суффиксной
ссылки
#ifdef debug
      std::cout << "------------------\n";
#endif

      if (vertices[v].link == -1)                           // при отстутствии сслыки вычисляем
ее
      {
         if (v == 0 || vertices[v].parent == 0)                  // добавлем ссылку на корень
         {
#ifdef debug
            std::cout << "get link to root: " <<" \n";
#endif
            vertices[v].link = 0;
         }
         else                                               // вычисление ссылки через родительскую
вершину
         {
#ifdef debug
            std::cout << "get link through parent: " <<" \n";
#endif
            vertices[v].link = go (getLink(vertices[v].parent), vertices[v].parentChar);
         }
      }
#ifdef debug
      std::cout << "get link to state: " << vertices[v].link <<" \n";
#endif
      return vertices[v].link;
   }

   void ahoCorasickJoker(std::string& text, std::string& pattern)
   {
#ifdef debug
      std::cout << "------------------\n";
#endif
      int state = 0;
      C.resize(text.size());                                // инициализация вектора
результата
      for(int i = 0; i < text.size();i++)
      {
         state = go(state, text[i]);                        // переход в новое состояние
#ifdef debug
         std::cout << "state: " << state <<" \n";
#endif
         for(size_t tmp = state; tmp != 0; tmp = getLink(tmp))          // цикл для поиска
всех возможных
         {                                                  // вхождений
            if(vertices[tmp].leaf)
            {
#ifdef debug
               std::cout << "found pattern "  << std::endl;
#endif
```

```
            for(auto Li : vertices[tmp].listPatterns)                    // цикл для вычисления
индексов
            {                                                            // вхождений
                int buffer = i + 1 - patterns[Li].size()- patternsIndex[Li];
                if(buffer >=0 && buffer <= text.size() - pattern.size())
                {
#ifdef debug
                    std::cout << "match at position: " << buffer << std::endl;
#endif
                C[buffer]++;
                }
            }
#ifdef debug
             std::cout << "------------------\n";
#endif
          }
        }
    }
  }

  void printResult(std::string pattern)
  {
#ifdef debug
     std::cout << "result:" << std::endl;
#endif
    for(size_t i = 0; i < C.size(); i++)
    {
        if(C[i] == patterns.size())
        std::cout << i + 1<< "\n";
    }
  }

  void splitString(std::string& pattern, char joker)              //метод разбиения строки
  {
    size_t currentPos, prevPos;
    for(size_t i = 0; i < pattern.size() && currentPos != std::string::npos;)
    {
      std::string buffer;
      while(pattern[i] == joker) i++;                            // пропуск джокеров в строке
      prevPos = i;
      currentPos = pattern.find(joker, i);                       //поиск следующего джокера
      if(currentPos == std::string::npos)                         //создание подстроки
        buffer = pattern.substr(i, pattern.size() - i);
      else
        buffer = pattern.substr( prevPos,currentPos - prevPos);
      if(!buffer.empty())
      {
        patternsIndex.push_back(prevPos);                        // запись индекса подстроки и
добавление ее в бор
        addString(buffer);
      }
      i = currentPos;
    }
  }
};

int main() {
  std::freopen("test", "r", stdin);
  Trie trie;
  trie.init();
  std::string text, pattern;
```

```
    char joker;
    std::cin >> text;
    std::cin >> pattern;
    std::cin >> joker;

    trie.splitString(pattern, joker);
    trie.ahoCorasickJoker(text, pattern);
    trie.printResult(pattern);

    return 0;
}
```