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. 8382 _____ Кобенко В.П.

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:

```
NTAG
3
TAGT
TAG
T
```

Sample Output:

```
2 2
2 3
```

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 Peak {
std::map<char,int> next;
std::map<char,int> states;
bool flag;
int parentPeak;
```

```
    char parentPeakSymb;
    int Slink;
    int patternNumber;
};
```

Структура для хранения вершин бора.

next – контейнер для хранения переходов по ребрам в боре.

states – контейнер для хранения переходов в автомате.

flag – флаг для проверки является ли данная вершина терминальной.

parentPeak – индекс родительской вершины.

parentPeakSymb – символ родительской вершины.

Slink – суффискная ссылка.

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

```
    std::vector<Peak> peaks;
    std::vector<std::string> patterns;
    std::vector<std::pair<int,int>> res;
```

Структура для хранения бора:

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

Описание функций Ахо-Корасик.

bool comp(std::pair<int, int> a, std::pair<int, int> b) – функция для сортировки вектора результата. Возвращается true, если первый аргумент меньше второго. a, b – значения вектора результата.

void NewStr (const std::string & s, std::vector<string>& patterns, std::vector<Peak>& peaks) – метод добавления новой строки в бор.

s – добавляемая строка.

int jump (int count, char symb, std::vector<Peak>& peaks) – метод перехода автомата в новое состояние. Возвращает номер нового состояния.

count – текущее состояние.

symb – символ перехода.

int getSufLink (int count, std::vector<Peak>& peaks) – метод получения

суффиксной ссылки. Возвращается номер вершины, на которую указывает

ссылка.

count – текущая вершина.

void ahoCorasick(std::string& inputStr, std::vector<Peak>& peaks, std::vector<std::pair<int,int>>& res, std::vector<std::string>& patterns) – метод, реализующий алгоритм Ахо-Корасик.
inputStr – текст, в котором ищутся совпадения.

void output(std::vector<std::pair<int,int>>& res, std::vector<Peak>& peaks, std::vector<std::string>& patterns) - метод печати результата работы алгоритма.

Описание структур поиска с джокером.

```
struct Peak {

std::map<char,int> next;

std::map<char,int> states;

bool flag;

int parentPeak;

char parentPeakSymb;

int Slink;

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

Структура для хранения вершин бора.

next – контейнер для хранения переходов по ребрам в боре.

states – контейнер для хранения переходов в автомате.

flag – флаг для проверки является ли данная вершина терминальной.

parentPeak – индекс родительской вершины.

parentPeakSymb – символ родительской вершины.

Slink – суффискная ссылка.

listPatterns – контейнер для хранения множества индексов подстрок.

std::vector<Peak> peaks;

std::vector<std::string> patterns;

std::vector<int> res;

std::vector<int> patternsIndex;

Структура для хранения бора
peaks — вектор вершин бора.
patterns — вектор для хранения шаблонов
res — 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 NewStr (const std::string & s, std::vector<string>& patterns,
std::vector<Peak>& peaks) – метод добавления новой строки в бор.
s – добавляемая строка.

int jump (int count, char symb, std::vector<Peak>& peaks) – метод перехода автомата в новое состояние. Возвращает номер нового состояния.

count – текущее состояние.

symb – символ перехода.

int getSufLink (int count, std::vector<Peak>& peaks) – метод получения

суффиксной ссылки. Возвращается номер вершины, на которую указывает

ссылка.

count – текущая вершина.

void AC_joker(std::string& inputStr, std::string& pattern, std::vector<Peak>& peaks, std::vector<std::string>& patterns, std::vector<int>& patternsIndex, std::vector<int>& res) – метод, поиск с джокером на основе алгоритма Ахо-Корасик.

inputStr – текст, в котором ищутся совпадения.

pattern – исходная строка с джокером.

void output(std::string pattern, std::vector<std::string>& patterns, std::vector<int>& res) - метод печати результата работы алгоритма.

void split(std::string& pattern, char joker, std::vector<int>& patternsIndex, std::vector<Peak>& peaks, std::vector<std::string>& patterns) – метод разбиения строки с джокером.

pattern – исходная строка с джокером.

joker – символ джокера.

Промежуточные результаты.

На рис. 1 представлены промежуточные шаги и результаты программы с алгоритмом Ахо-Корасика(без джокера).

```
NTAG
3
TAGT
Добавляем строку в контейнер: TAGT
Символ: T
Добавляем новую вершину из 0(T)
Символ: A
Добавляем новую вершину из 1(A)
Символ: G
Добавляем новую вершину из 2(G)
Символ: T
Добавляем новую вершину из 3(T)
TAG
Добавляем строку в контейнер: TAG
Символ: T
Символ: A
Символ: G
T
Добавляем строку в контейнер: T
Символ: T
Текущий символ: N
Переходим по суфф. ссылке 0
Следующее состояние: 0
Состояние: 0
Текущий символ: T
Следующее состояние: 1
Состояние: 1
Найден шаблон №0: T
Добавляем суфф. ссылку на корень:
Получаем ссылку на состояние: 0
Текущий символ: A
Следующее состояние: 2
Состояние: 2
Вычисляем суфф. ссылку через родительскую вершину:
Получаем ссылку на состояние: 0
Текущий символ: A
Переходим по суфф. ссылке 0
Следующее состояние: 0
Получаем ссылку на состояние: 0
Текущий символ: G
Следующее состояние: 3
Состояние: 3
Найден шаблон №0: TAG
Вычисляем суфф. ссылку через родительскую вершину:
Получаем ссылку на состояние: 0
Текущий символ: G
Переходим по суфф. ссылке 0
Следующее состояние: 0
Получаем ссылку на состояние: 0
Ответ:
2 2
2 3
Количество вершин: 5
Пересечение шаблонов: TAG и T. №2, №3 на индексах 2,2
```

Рис. 1 - Ахо-Корасик(без джокера).

На рис.2 представлены промежуточные шаги и результаты программы с алгоритмом Ахо-Корасика(с джокером).

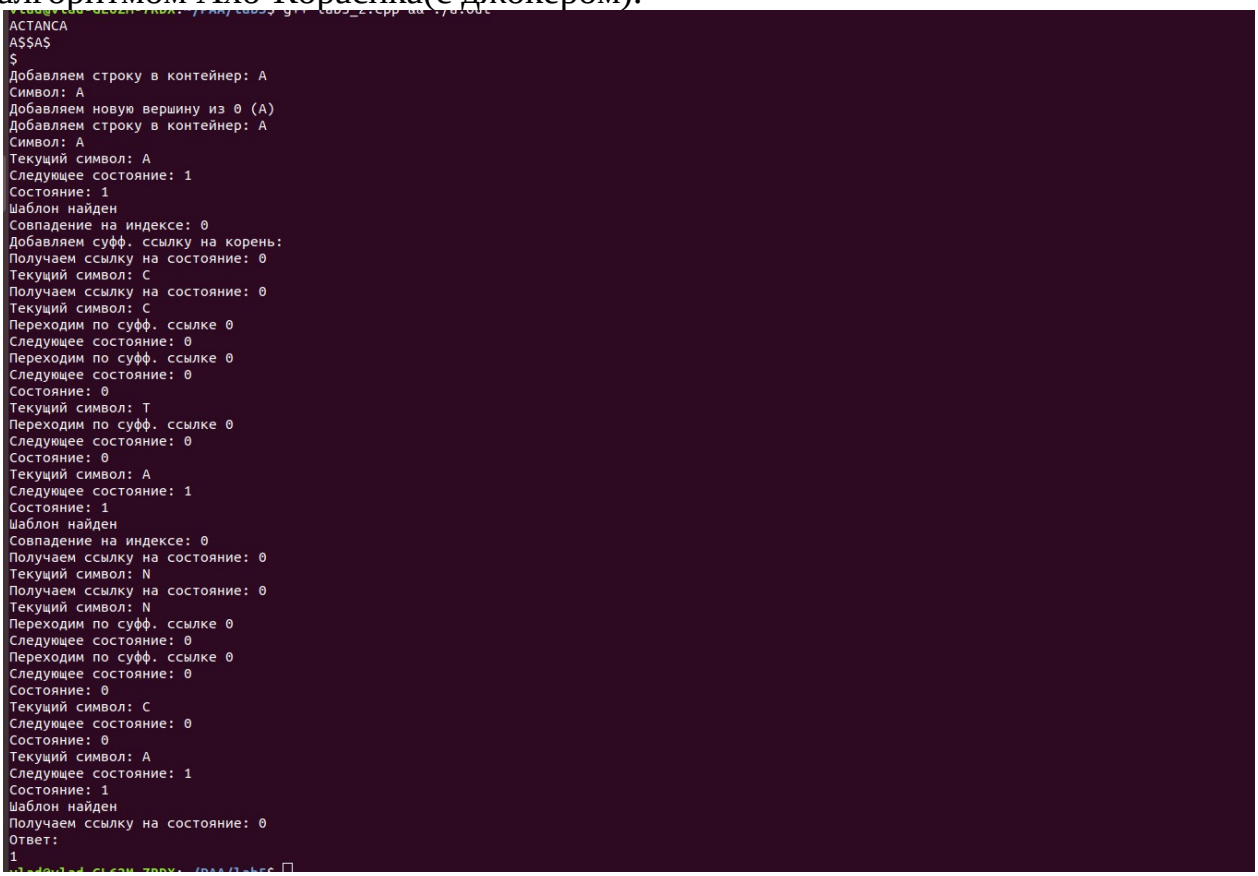

Рис. 2 – Ахо-Корасик(с джокером).

Тестирование.

Тестирование Ахо-Корасика без джокера.

№	Input	Output
1	abcsddh 3 bcsd dd fg	2 1 5 2
2	abababab 1 aba	1 1 3 1 5 1
3	abab 1 dfg	-

Тестирование Ахо-Корасика с джокером.

№	Input	Output
1	ACTANCA A$$A$ $	1
2	Vlvlv v#v #	1 3
3	anton ga $	-

Вывод.

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

Приложение А. Исходный код.

lab5_1.cpp

```
#include <iostream>
#include <map>
#include <vector>
#include <algorithm>

//#define d

struct Peak {
std::map<char,int> next;
std::map<char,int> states;
bool flag;
int parentPeak;
char parentPeakSymb;
int Slink;
int patternNumber;
};

void NewStr (const std::string& , std::vector<std::string>& , std::vector<Peak>& );
int jump(int , char , std::vector<Peak>& );
int getSufLink (int , std::vector<Peak>& );
void ahoCorasick(std::string& , std::vector<Peak>& , std::vector<std::pair<int,int>>& ,
std::vector<std::string>& );
void output(std::vector<std::pair<int,int>>& , std::vector<Peak>& ,
std::vector<std::string>& );
bool comp(std::pair<int, int>& , std::pair<int, int>& );


int main() {
std::vector<Peak> peaks;
std::vector<std::string> patterns;
std::vector<std::pair<int,int>> res;
std::string inputStr, pattern;
Peak head;
head.parentPeak = head.Slink = -1;
head.flag = false;
peaks.push_back(head);
int n = 0;
std::cin >> inputStr;
std::cin >> n;
for(int i = 0; i < n ;i++){
std::cin >> pattern;
NewStr(pattern, patterns, peaks);
}
ahoCorasick(inputStr, peaks, res, patterns);
output(res, peaks, patterns);

return 0;
}

bool comp(std::pair<int, int>& a, std::pair<int, int>& b){
if (a.second == b.second)
return a.first < b.first;
```

```
return a.second < b.second;
}

void NewStr (const std::string& s, std::vector<std::string>& patterns,
std::vector<Peak>& peaks){
#ifdef d
std::cout << "Добавляем строку в контейнер: " << s << std::endl;
#endif
patterns.push_back(s);
int count = 0;
for (auto symb : s) {
#ifdef d
std::cout << "Символ: " << symb << " \n";
#endif
if (peaks[count].next.find(symb) == peaks[count].next.end()) {
#ifdef d
std::cout << "Добавляем новую вершину из " << count << "(" << symb <<") \n";
#endif
Peak buf;
buf.flag = false;
buf.Slink = -1;
buf.parentPeak = count;
buf.parentPeakSymb = symb;
peaks.push_back(buf);
peaks[count].next[symb] = peaks.size() - 1;
}
count = peaks[count].next[symb];
}
peaks[count].flag = true;
peaks[count].patternNumber = patterns.size() - 1;
}

int jump (int count, char symb, std::vector<Peak>& peaks) {
#ifdef d
std::cout << "Текущий символ: " << symb << "\n";
#endif
if (peaks[count].states.find(symb) == peaks[count].states.end()){
if (peaks[count].next.find(symb) != peaks[count].next.end()){
peaks[count].states[symb] = peaks[count].next[symb];
}
else{
peaks[count].states[symb] = count==0 ? 0 : jump (getSufLink(count, peaks), symb,
peaks);
#ifdef d
std::cout << "Переходим по суфф. ссылке " << peaks[count].states[symb] <<" \n";
#endif
}
}
#ifdef d
std::cout << "Следующее состояние: " << peaks[count].states[symb] <<" \n";
#endif
return peaks[count].states[symb];
}

int getSufLink (int count, std::vector<Peak>& peaks) {
```

```
if (peaks[count].Slink == -1){
if (count == 0 || peaks[count].parentPeak == 0){
#ifdef d
std::cout << "Добавляем суфф. ссылку на корень: " <<" \n";
#endif
peaks[count].Slink = 0;
}
else{
#ifdef d
std::cout << "Вычисляем суфф. ссылку через родительскую вершину: " <<" \n";
#endif
peaks[count].Slink = jump(getSufLink(peaks[count].parentPeak, peaks),
peaks[count].parentPeakSymb, peaks);
}
}
#ifdef d
std::cout << "Получаем ссылку на состояние: " << peaks[count].Slink <<" \n";
#endif
return peaks[count].Slink;
}

void ahoCorasick(std::string& inputStr, std::vector<Peak>& peaks,
std::vector<std::pair<int,int>>& res, std::vector<std::string>& patterns){
int state = 0;
for(int i = 0; i < inputStr.size();i++){
state = jump(state, inputStr[i], peaks);
#ifdef d
std::cout << "Состояние: " << state <<" \n";
#endif
for(size_t tmp = state; tmp != 0; tmp = getSufLink(tmp, peaks)){
if(peaks[tmp].flag){
std::pair<int,int> buf;
buf.first = peaks[tmp].patternNumber;
buf.second = i - patterns[buf.first].size();
res.push_back(buf);
#ifdef d
std::cout << "Найден шаблон №" << buf.second << ": " << patterns[buf.first] <<" \
n";
#endif
}
}
}
}

void output(std::vector<std::pair<int,int>>& res, std::vector<Peak>& peaks,
std::vector<std::string>& patterns){
#ifdef d
std::cout << "Ответ: " <<std::endl;
#endif
std::sort(res.begin(), res.end(), comp);
for(auto iter : res){
std::cout << iter.second + 2 << " " << iter.first + 1 << std::endl;
}
#ifdef d
```

```
std::cout << "Количество вершин: " << peaks.size() << std::endl;
for(size_t i = 0 ; i < res.size() - 1; i++) {
for(size_t j = i + 1 ; j < res.size(); j++){
size_t first, second;
first = patterns[res[i].first].size() - 1 + res[i].second;
second = res[j].second;
if(first >= second){
std::cout << "Пересечение шаблонов: " << patterns[res[i].first] << " и " <<
patterns[res[j].first] << ". №" << res[i].first + 1<< ", №" << res[j].first + 1 << " на
индексах " << res[i].second + 2 << "," << res[j].second + 2 << "\n";
}
}
}
#endif
}
```

lab5_2.cpp

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

struct Peak {
std::map<char,int> next;
std::map<char,int> states;
bool flag;
int parentPeak;
char parentPeakSymb;
int Slink;
std::vector<int> listPatterns;
};

void split(std::string& , char , std::vector<int>& , std::vector<Peak>& ,
std::vector<std::string>& );
void output(std::string , std::vector<std::string>& , std::vector<int>& );
void AC_joker(std::string& , std::string& , std::vector<Peak>& ,
std::vector<std::string>& , std::vector<int>& , std::vector<int>& );
int getSufLink(int , std::vector<Peak>& );
int jump(int , char , std::vector<Peak>& );
void NewStr(std::string , std::vector<std::string>& , std::vector<Peak>& );


int main(){
std::string inputStr, pattern;
std::vector<Peak> peaks;
std::vector<std::string> patterns;
std::vector<int> res;
std::vector<int> patternsIndex;
Peak head;
head.parentPeak = head.Slink = -1;
head.flag = false;
peaks.push_back(head);
char joker;
std::cin >> inputStr;
std::cin >> pattern;
std::cin >> joker;
split(pattern, joker, patternsIndex, peaks, patterns);
AC_joker(inputStr, pattern, peaks, patterns, patternsIndex, res);
output(pattern, patterns, res);
return 0;
}

void NewStr(std::string s, std::vector<std::string>& patterns, std::vector<Peak>&
peaks){
#ifdef d
std::cout << "Добавляем строку в контейнер: " << s << std::endl;
#endif
patterns.push_back(s);
int count = 0;
for (auto symb : s) {
#ifdef d
```

```
std::cout << "Символ: " << symb << " \n";
#endif
if (peaks[count].next.find(symb) == peaks[count].next.end()) {
#ifdef d
std::cout << "Добавляем новую вершину из " << count << " (" << symb <<") \n";
#endif
Peak buf;
buf.flag = false;
buf.Slink = -1;
buf.parentPeak = count;
buf.parentPeakSymb = symb;
peaks.push_back(buf);
peaks[count].next[symb] = peaks.size() - 1;
}
count = peaks[count].next[symb];
}
peaks[count].flag = true;
peaks[count].listPatterns.push_back(patterns.size() - 1);
}

int jump(int count, char symb, std::vector<Peak>& peaks){
#ifdef d
std::cout << "Текущий символ: " << symb << "\n";
#endif
if (peaks[count].states.find(symb) == peaks[count].states.end()){
if (peaks[count].next.find(symb) != peaks[count].next.end()){
peaks[count].states[symb] = peaks[count].next[symb];
}
else{
peaks[count].states[symb] = count==0 ? 0 : jump(getSufLink(count, peaks), symb,
peaks);
#ifdef d
std::cout << "Переходим по суфф. ссылке " << peaks[count].states[symb] <<" \n";
#endif
}
}
#ifdef d
std::cout << "Следующее состояние: " << peaks[count].states[symb] <<" \n";
#endif
return peaks[count].states[symb];
}

int getSufLink(int count, std::vector<Peak>& peaks){
if (peaks[count].Slink == -1){
if (count == 0 || peaks[count].parentPeak == 0){
#ifdef d
std::cout << "Добавляем суфф. ссылку на корень: " <<" \n";
#endif
peaks[count].Slink = 0;
}
else{
#ifdef d
std::cout << "Вычисляем суфф. ссылку через родительскую вершину: " <<" \n";
#endif
```

```
peaks[count].Slink = jump (getSufLink(peaks[count].parentPeak, peaks),
peaks[count].parentPeakSymb, peaks);
}
}
#ifdef d
std::cout << "Получаем ссылку на состояние: " << peaks[count].Slink <<" \n";
#endif
return peaks[count].Slink;
}

void AC_joker(std::string& inputStr, std::string& pattern, std::vector<Peak>& peaks,
std::vector<std::string>& patterns, std::vector<int>& patternsIndex, std::vector<int>&
res){
int state = 0;
res.resize(inputStr.size());
for(int i = 0; i < inputStr.size();i++){
state = jump(state, inputStr[i], peaks);
#ifdef d
std::cout << "Состояние: " << state <<" \n";
#endif
for(size_t tmp = state; tmp != 0; tmp = getSufLink(tmp, peaks)){
if(peaks[tmp].flag){
#ifdef d
std::cout << "Шаблон найден " << std::endl;
#endif
for(auto Li : peaks[tmp].listPatterns){
int counter = i + 1 - patterns[Li].size()- patternsIndex[Li];
if(counter >=0 && counter <= inputStr.size() - pattern.size()){
#ifdef d
std::cout << "Совпадение на индексе: " << counter << std::endl;
#endif
res[counter]++;
}
}
}
}
}
}

void output(std::string pattern, std::vector<std::string>& patterns, std::vector<int>&
res){
#ifdef d
std::cout << "Ответ:" << std::endl;
#endif
for(size_t i = 0; i < res.size(); i++){
if(res[i] == patterns.size())
std::cout << i + 1 << "\n";
}
}

void split(std::string& pattern, char joker, std::vector<int>& patternsIndex,
std::vector<Peak>& peaks, std::vector<std::string>& patterns){
size_t currentPos, prevPos;
for(size_t i = 0; i < pattern.size() && currentPos != std::string::npos;){
std::string str_buf;
```

```
while(pattern[i] == joker) i++;
prevPos = i;
currentPos = pattern.find(joker, i);
if(currentPos == std::string::npos)
str_buf = pattern.substr(i, pattern.size() - i);
else
str_buf = pattern.substr( prevPos,currentPos - prevPos);
if(!str_buf.empty()){
patternsIndex.push_back(prevPos);
NewStr(str_buf, patterns, peaks);
}
i = currentPos;
}
}
```