МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр.8382 _____ Синельников М.Р

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2020

Цель работы.

Ознакомится с алгоритмом Ахо-Корасик для поиска подстроки в строке.

Задание.

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

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 из следующих n строк содержит шаблон из набора P = {p1,p2…,pn} 1<=|pi|<=75

Все строки содержат симолы 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

3

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, именуемый джокером (wild card), который "совпадает" с любым символом. По заданному содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

Например, образец ab??c?ab??c? с джокером ? встречается дважды в тексте xabvccbababcaxxabvccbababcax.

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 <= |T| <= 100000).

Шаблон(P, 1 <= |P| <= 40)

Символ джокера
Выход:
Строки с номерами позиций вхождений шаблона (каждая строка содержит 4 только один номер). Номера должны выводиться в порядке возрастания.
Пример входных данных:
ACTANCA
A$$A$
$
Пример выходных данных:
1

Описание алгоритма.

Задание№1

Создаётся корневая вершина бора. В бор добавляются введённые шаблоны.

После добавления в бор шаблонов, для всех его вершин вычисляются суффиксные ссылки. Для корня и его детей ссылка ведёт в корень. Для

остальных определяется по следующему правилу: выполняется переход по ссылке родителя. Проверяется, если ли среди потомков данной вершины переход по той же букве, что и в исходную вершину. Если есть, суффиксная ссылка исходной вершины устанавливается на найденную вершину. Если нет, то выполняется переход по суффиксной ссылке данной вершины и процесс повторяется. Если нужная вершина не найдена, суффиксная ссылка устанавливается на корень.

Поиск вхождений шаблонов в тексте осуществляется с помощью переходов в автомате, начиная с корневой вершины. На каждой итерации проверяется, терминальная ли текущая вершина. Если да, то очередное вхождение помещается в результирующий массив.

В конце результирующий массив сортируется, после чего результат печатается в нужном формате.

После вывода основного результата в цикле определяются образцы, имеющие пересечение с другими образцами. Для этого заводится дополнительный массив, который при обходе полученных образцов инкрементирует значение ячейки текущего образца, если он пересекается с другим. Если значение ячейки не равно 0, значит текущий образец пересекается хотя бы с одним из других образцов.

Задание№2

Введённый шаблон разбивается на подстроки по симолу-джокеру, полученное множество строк добавляется в бор в качестве шаблонов. Для каждого шаблона также определяется индекс его вхождения в изначальный шаблон. В функции поиска заводится дополнительный массив. Каждая встреченная терминальная вершина инкрементирует значение в массиве-счетчика на позиции, вычисленной с учетом текущей позиции в строке, длине найденной подстроки шаблона, и индексу вхождения данной подстроки в изначальный шаблон. После обработки строки происходит проход по массиву-счетчику. Если в какой-то ячейке значение счетчика равно количеству подстрок шаблона, значит, с данной позиции в тексте начинается вхождение целого

шаблона (при условии, что длина шаблона не превышает разность между длиной текста и индексом вхождения).

Оценка сложности.

Добавление в бор имеет сложность O(|p|*n), где n — длина шаблона, p — длина самого большого шаблона. Построение суффикс-ссылок также имеет линейную сложность - O(|p| * n). Обработка строки линейно зависит от ее длины — O(|T|). Общая сложность по времени первого алгоритма — O(|p| * n + |T|).

Во втором алгоритме присутствует разбиение изначального шаблона, которая линейно зависит от его длины — O(P). Сложность по времени второго алгоритма — O(|P| + |p| * n + |T|).

Сложность первого алгоритма по памяти — O(|p| * n). Сложность второго выше, так как добавляется дополнительный массив-счётчик, чей размер совпадает с длинной исходного текста. Поэтому сложность второго алгоритма по памяти — O(|p| * n + |T|).

Тестирование.

Программа1 была протестирована на следующих исходных данных:

Входные данные	Результат
NTAG 3 TAGT TAG T	NTAG 3 TAGT TAG T Создали новую вершину с символом T Создали новую вершину с символом A Создали новую вершину с символом G Создали новую вершину с символом T Шаблон TAGT полностью добавлен Переход с символом T уже существует Переход с символом A уже существует Переход с символом G уже существует Шаблон TAG полностью добавлен Переход с символом T уже существует Шаблон T полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от T в корень Установили суффикс ссылку из вершины TA на корень Установили суффикс ссылку из вершины TAG на корень Установили суффикс ссылку из вершины TAGT в вершину T Автомат построен

	Начинаем обход текста Текущий символ N Текущая вершина: Текущий символ T Текущая вершина: T Нашли вхождение шаблона с номером 3 с индекса 2 Текущий символ A Текущая вершина: TA Текущий символ G Текущая вершина: TAG Нашли вхождение шаблона с номером 2 с индекса 2 Количество вершин в автомате: 5 2 2 2 3 Список образцов,имеющих пересечение с другими образцами: TAG на позиции 2 T на позиции 2
CCCA 1 CC	Создали новую вершину с символом C Создали новую вершину с символом C Шаблон CC полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от C в корень Установили суффикс ссылку из вершины CC в вершину C Автомат построен Начинаем обход текста Текущий символ C Текущая вершина: C Текущий символ C Текущая вершина: CC Нашли вхождение шаблона с номером 1 с индекса 1 Текущий символ C Текущая вершина: CC Нашли вхождение шаблона с номером 1 с индекса 2 Текущий символ A Текущая вершина: Количество вершин в автомате: 3 1 1 2 1 Список образцов,имеющих пересечение с другими образцами: CC на позиции 1 CC на позиции 2
ABBAABBA 2 BAAB BBAABB	Создали новую вершину с символом B Создали новую вершину с символом A Создали новую вершину с символом A Создали новую вершину с символом B Шаблон BAAB полностью добавлен Переход с символом B уже существует Создали новую вершину с символом B

	Создали новую вершину с символом A Создали новую вершину с символом A Создали новую вершину с символом B Создали новую вершину с символом B Шаблон BBAABB полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от B в корень Установили суффикс ссылку из вершины BA на корень Установили суффикс ссылку из вершины BAA на корень Установили суффикс ссылку из вершины BAAB в вершину B Установили суффикс ссылку из вершины BB в вершину B Установили суффикс ссылку из вершины BBA в вершину BA Установили суффикс ссылку из вершины BBAA в вершину BAA Установили суффикс ссылку из вершины BBAAB в вершину BAAB Установили суффикс ссылку из вершины BBAABB в вершину BB Автомат построен Начинаем обход текста Текущий символ A Текущая вершина: Текущий символ B Текущая вершина: B Текущий символ B Текущая вершина: BB Текущий символ A Текущая вершина: BBA Текущий символ A Текущая вершина: BBAA Текущий символ B Текущая вершина: BBAAB Нашли вхождение шаблона с номером 1 с индекса 3 Текущий символ B Текущая вершина: BBAABB Нашли вхождение шаблона с номером 2 с индекса 2 Текущий символ A Текущая вершина: BBA Количество вершин в автомате: 10 2 2 3 1 Список образцов,имеющих пересечение с другими образцами:

	BBAABB на позиции 2 BAAB на позиции 3
AGCAC 1 Y	Создали новую вершину с символом Y Шаблон Y полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от Y в корень Автомат построен Начинаем обход текста Текущий символ A Текущая вершина: Текущий символ G Текущая вершина: Текущий символ C Текущая вершина: Текущий символ A Текущая вершина: Текущий символ C Текущая вершина: Количество вершин в автомате: 2 Список образцов,имеющих пересечение с другими образцами:

Программа 2 была протестирована на следующих входных данных:

Входные данные	Результат
ACTANCA A$$A$ $	Разбиение на построки A A Создали новую вершину с символом A Шаблон A полностью добавлен Переход с символом A уже существует Шаблон A полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от A в корень Автомат построен Размер автомата 2 Начинаем обход текста Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ C Текущая вершина: Текущий символ T Текущая вершина:

	Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ N Текущая вершина: Текущий символ C Текущая вершина: Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики проверяем счётчики Нашли вхождение на позиции 1 1
TYABUAB $$AB$ $	Разбиение на построки AB Создали новую вершину с символом A Создали новую вершину с символом B Шаблон AB полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от A в корень Установили суффикс ссылку из вершины AB на корень Автомат построен Размер автомата 3 Начинаем обход текста Текущий символ T Текущая вершина: Текущий символ Y Текущая вершина: Текущий символ A Текущая вершина: A Текущий символ B Текущая вершина: AB найден терминальный символ увеличиваем счётчики Текущий символ U Текущая вершина: Текущий символ A Текущая вершина: A Текущий символ B Текущая вершина: AB найден терминальный символ увеличиваем счётчики проверяем счётчики Нашли вхождение на позиции 1 1
ACACGACAC	Разбиение на построки

A$ $	A Создали новую вершину с символом A Шаблон A полностью добавлен Установили суффикс-ссылку от корня в корень Установили суффикс-ссылку от A в корень Автомат построен Размер автомата 2 Начинаем обход текста Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ C Текущая вершина: Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ C Текущая вершина: Текущий символ G Текущая вершина: Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ C Текущая вершина: Текущий символ A Текущая вершина: A найден терминальный символ увеличиваем счётчики Текущий символ C Текущая вершина: проверяем счётчики Нашли вхождение на позиции 1 Нашли вхождение на позиции 3 Нашли вхождение на позиции 6 Нашли вхождение на позиции 8 1 3 6 8

Выводы.

В ходе работы был реализован алгоритм Ахо-Корасик для поиска подстрок в строке, а также его модификации для поиска шаблона с «джокером». Кроме того, для каждого алгоритма выводится количество вершин в автомате, а для первой программы дополнительно печатаются только те образцы, которые пересекаются с другими.

Приложения А. Исходный код программы 1

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <time.h>

using std::cin;
using std::cout;
using std::vector;
using std::string;
using std::pair;
using std::make_pair;
using std::map;
using std::sort;

class Node{
public:
    char symbol;
    int index;
    bool ispattern = false;
    bool suffix_edge = false;
    int curr_branch_len = 0;
    int pos = -1;
    Node(char symbol,int index,bool ispattern,bool suffix_edge,int curr_branch_len,int pos)
    {
        this->symbol = symbol;
        this->index = index;
        this->ispattern = ispattern;
        this->suffix_edge = suffix_edge;
        this->curr_branch_len = curr_branch_len;
        this->pos = pos;
    }
    Node() = default;
};

void inputPatterns(vector<string> &patterns){
    for(int i = 0;i < patterns.size();i++)
        cin >> patterns[i];
}
```

```
string reverseString(string str){
    char temp;
    for(int i = 0;i < str.length() / 2;i++) {
        temp = str[i];
        str[i] = str[str.length() - 1 - i];
        str[str.length() - 1 - i] = temp;
    }

    return str;
}

string getName(vector<vector<Node>> &automation,vector<int> &prev_vertices,int curr_vertex){
    string name;
    int prev;
    while (curr_vertex){
        prev = prev_vertices[curr_vertex];
        for(int i = 0;i < automation[prev].size();i++){
            if(automation[prev][i].index == curr_vertex) {
                name += automation[prev][i].symbol;
                break;
            }
        }
        curr_vertex = prev;
    }
    return reverseString(name);
}

void constructTrie(vector<string> &patterns,vector<vector<Node>> &automation,vector<int> &prev_vertices){
    int branch_len = 0;
    int size = patterns.size();
    int number_of_vertices = 0;
    int curr_vertex;
    bool flag;
    for(int i = 0;i < size;i++){
        curr_vertex = 0;
        branch_len = 1;
        for(int j = 0;j < patterns[i].size();j++){
            flag = false;
            for(int k = 0;k < automation[curr_vertex].size();k++){
                if(automation[curr_vertex][k].symbol == patterns[i][j]){
                    branch_len++;
```

```
                cout << "Переход с символом " << patterns[i][j] << " уже существует\n";
                if(j == patterns[i].size() - 1) {
                    automation[curr_vertex][k].ispattern = true;
                    automation[curr_vertex][k].pos = i + 1;
                }
                curr_vertex = automation[curr_vertex][k].index;
                flag = true;
                break;
            }
        }
        if(flag)
            continue;
        number_of_vertices++;
        automation.emplace_back();
        if(j == patterns[i].size() - 1)
            automation[curr_vertex].emplace_back(Node(patterns[i][j],number_of_vertices,true, false,branch_len,i + 1));
        else
            automation[curr_vertex].emplace_back(Node(patterns[i][j],number_of_vertices, false, false,branch_len,-1));
        prev_vertices.emplace_back(curr_vertex);
        curr_vertex = number_of_vertices;
        branch_len++;
        cout << "Создали новую вершину с символом " << patterns[i][j] << "\n";
    }
    cout << "Шаблон " << patterns[i] << " полностью добавлен\n";
  }
}

void addSuffixEdge(vector<vector<Node>> &automation, int curr_vertex, char symbol,int main_vertex,vector<int>
&prev_vertices,vector<bool> &visited) {
  if(automation[curr_vertex].size() && automation[curr_vertex][automation[curr_vertex].size() - 1].suffix_edge)
    return;
  if(!curr_vertex || !prev_vertices[curr_vertex]){
    automation[curr_vertex].emplace_back(Node('$',0, false, true,0,-1));
    if(!curr_vertex)
      cout << "Установили суффикс-ссылку от корня в корень\n";
    else{
      cout << "Установили суффикс-ссылку от " << getName(automation,prev_vertices,curr_vertex) << " в корень"
<< "\n";
    }
    return;
  }
  curr_vertex = prev_vertices[curr_vertex];
```

```
    char curr_symbol;
    while (curr_vertex){
        for(int i = 0;i < automation[prev_vertices[curr_vertex]].size();i++)
            if(automation[prev_vertices[curr_vertex]][i].index == curr_vertex){
                curr_symbol = automation[prev_vertices[curr_vertex]][i].symbol;
                break;
            }
        addSuffixEdge(automation,curr_vertex,curr_symbol,curr_vertex,prev_vertices,visited);
        curr_vertex = automation[curr_vertex][automation[curr_vertex].size() - 1].index;

        for(int i = 0;i < automation[curr_vertex].size();i++){
            if(automation[curr_vertex][i].symbol == symbol && !automation[curr_vertex][i].suffix_edge){
                if(automation[curr_vertex][i].index != main_vertex){
                                    automation[main_vertex].emplace_back(Node(symbol,automation[curr_vertex][i].index,
automation[curr_vertex][i].ispattern,
                        true,automation[curr_vertex][i].curr_branch_len, automation[curr_vertex][i].pos));

                                                cout << "Установили суффикс ссылку из вершины " <<
getName(automation,prev_vertices,main_vertex) << " в вершину " << getName(automation,prev_vertices,
automation[curr_vertex][i].index) << "\n";
                }
                else {
                                                cout << "Установили суффикс ссылку из вершины " <<
getName(automation,prev_vertices,main_vertex) << " на корень " << "\n";
                    automation[main_vertex].emplace_back(Node('$', 0, false, true, 0, -1));
                }
                return;
            }
        }
    }
     cout << "Установили суффикс ссылку из вершины " << getName(automation,prev_vertices,main_vertex) << " на
корень " << "\n";
    automation[main_vertex].emplace_back(Node('$',0, false, true,0,-1));
}

void constructAutomation(vector<vector<Node>> &automation,int curr_vertex,vector<int>
&prev_vertices,vector<bool> &visited){
        char symbol;
        for(int i = 0;i < automation[prev_vertices[curr_vertex]].size();i++)
            if(automation[prev_vertices[curr_vertex]][i].index == curr_vertex){
                symbol = automation[prev_vertices[curr_vertex]][i].symbol;
                break;
```

```
        }
      addSuffixEdge(automation,curr_vertex, symbol,curr_vertex,prev_vertices,visited);
      for(int i = 0;i < automation[curr_vertex].size();i++)
        if(!automation[curr_vertex][i].suffix_edge)
          constructAutomation(automation,automation[curr_vertex][i].index,prev_vertices,visited);
}

void next(vector<vector<Node>> &automation, int curr_vertex,char symbol,int &new_vertex){
  for(int j = 0;j < automation[curr_vertex].size();j++)
    if(automation[curr_vertex][j].symbol == symbol && !automation[curr_vertex][j].suffix_edge){
      curr_vertex = automation[curr_vertex][j].index;
      new_vertex = curr_vertex;
      return;
    }

  if(!curr_vertex) {
    new_vertex = curr_vertex;
    return;
  }
  next(automation, automation[curr_vertex][automation[curr_vertex].size() - 1].index,symbol,new_vertex);
}

void findMatching(string &text,vector<vector<Node>> &automation,vector<pair<int,int>> &output,vector<int>
&prev_vertices){
    cout << "Начинаем обход текста\n";
    int curr_vertex = 0;
    int mediator;
    char symbol;
    int new_vertex;
    int prev = 0;
    for(int i = 0;i < text.length();i++){
      symbol = text[i];
      cout << "Текущий символ " << symbol << "\n";
      next(automation, curr_vertex,symbol,new_vertex);
      curr_vertex = new_vertex;
      cout << "Текущая вершина: " << getName(automation,prev_vertices,curr_vertex) << "\n";
      mediator = curr_vertex;
      prev = prev_vertices[curr_vertex];
      while (mediator){
        for(int j = 0;j < automation[prev].size();j++)
          if(automation[prev][j].index == mediator) {
            if (automation[prev][j].ispattern) {
```

```
                output.emplace_back(make_pair(i - automation[prev][j].curr_branch_len + 2,
                    automation[prev][j].pos));

                cout << "Нашли вхождение шаблона с номером " << automation[prev][j].pos << " " << "с индекса "
<< i - automation[prev][j].curr_branch_len + 2 << "\n";
                }
                break;
            }
          prev = mediator;
          mediator = automation[mediator][automation[mediator].size() - 1].index;
        }
    }
}

void printOutput(vector<pair<int,int>> &output,vector<string> &patterns) {
   vector<int> mediator(output.size());
   sort(output.begin(), output.end());
   for(int i = 0;i < output.size();i++)
      cout << output[i].first << " " << output[i].second << "\n";

   cout << "Список образцов,имеющих пересечение с другими образцами:\n";
   int counter = 0;
   bool flag ;
   for (int i = 0; i < output.size(); i++) {
      flag = false;
      if (mediator[i]) {
         flag = true;
      }
      counter = i + 1;
      while (1) {
         if (counter == output.size())
            break;
         if (output[counter].first == output[i].first) {
            mediator[counter]++;
            flag = true;
         }
         else if (output[i].first + patterns[output[i].second - 1].length() >= output[counter].first) {
            mediator[counter]++;
            flag = true;
         }
         else
            break;
```

```
            counter++;
        }
        if(flag)
            cout << patterns[output[i].second - 1] << " на позиции " << output[i].first << "\n";
    }
}

int main() {
    int number_of_patterns;
    string curr_string;
    string text;
    cin >> text;
    cin >> number_of_patterns;
    vector<string> patterns(number_of_patterns);
    vector<vector<Node>> automation(1);
    vector<pair<int,int>> output;
    inputPatterns(patterns);
    vector<int> prev_vertices;
    prev_vertices.push_back(0);
    constructTrie(patterns,automation,prev_vertices);
    vector<bool> visited(automation.size());
    constructAutomation(automation,0,prev_vertices,visited);
    cout << "Автомат построен" << "\n";
    cout << "Количество вершин в автомате: " << automation.size() << "\n";
    findMatching(text,automation,output,prev_vertices);
    printOutput(output,patterns);
    return 0;
}
```

Приложение Б. Исходный код программы 2

```
#include <iostream>
#include <vector>
#include <map>
#include <algorithm>
#include <time.h>
#include <sstream>
#include <fstream>

using std::cin;
using std::cout;
using std::vector;
using std::string;
using std::pair;
using std::make_pair;
using std::map;
using std::sort;
using std::istringstream;
using std::fstream;

class Node{
public:
    char symbol;
    int index;
    bool ispattern = false;
    bool suffix_edge = false;
    int curr_branch_len = 0;
    int pos = -1;
    vector<int> subpatternspos;
    Node(char symbol,int index,bool ispattern,bool suffix_edge,int curr_branch_len,int pos,int subpatternpos)
    {
        this->symbol = symbol;
        this->index = index;
        this->ispattern = ispattern;
        this->suffix_edge = suffix_edge;
        this->curr_branch_len = curr_branch_len;
        this->pos = pos;
        this->subpatternspos.push_back(subpatternpos);
    }
```

```
    Node(char symbol,int index,bool ispattern,bool suffix_edge,int curr_branch_len,int pos,vector<int> &subpatternpos)
    {
        this->symbol = symbol;
        this->index = index;
        this->ispattern = ispattern;
        this->suffix_edge = suffix_edge;
        this->curr_branch_len = curr_branch_len;
        this->pos = pos;
        this->subpatternspos = subpatternpos;
    }
    Node(char symbol,int index,bool ispattern,bool suffix_edge,int curr_branch_len,int pos)
    {
        this->symbol = symbol;
        this->index = index;
        this->ispattern = ispattern;
        this->suffix_edge = suffix_edge;
        this->curr_branch_len = curr_branch_len;
        this->pos = pos;
    }
    Node() = default;
};

int split(vector<pair<string,int>> &collection, string pattern,char wildcard){
    int size = 0;
    istringstream stream(pattern);
    std::string substr;
    int number = 0;
    cout << "Разбиение на построки\n";
    while (std::getline(stream, substr, wildcard)) {
        if (substr.length() > 0) {
            cout << substr << "\n";
            collection.push_back(std::make_pair(substr, number));
            number += substr.length() + 1;
            size++;
        }
        else {
            number++;
        }
    }


    return size;
```

```
}

string reverseString(string str){
   char temp;
   for(int i = 0;i < str.length() / 2;i++) {
      temp = str[i];
      str[i] = str[str.length() - 1 - i];
      str[str.length() - 1 - i] = temp;
   }

   return str;
}

string getName(vector<vector<Node>> &automation,vector<int> &prev_vertices,int curr_vertex){
   string name;
   int prev;
   while (curr_vertex){
      prev = prev_vertices[curr_vertex];
      for(int i = 0;i < automation[prev].size();i++){
         if(automation[prev][i].index == curr_vertex) {
            name += automation[prev][i].symbol;
            break;
         }
      }
      curr_vertex = prev;
   }
   return reverseString(name);
}

void constructTrie(vector<pair<string,int>> collection,vector<vector<Node>> &automation,vector<int>
&prev_vertices){
   int branch_len = 0;
   int size = collection.size();
   int number_of_vertices = 0;
   int curr_vertex;
   bool flag;
   for(int i = 0;i < size;i++){
      curr_vertex = 0;
      branch_len = 1;
      for(int j = 0;j < collection[i].first.size();j++){
         flag = false;
         for(int k = 0;k < automation[curr_vertex].size();k++){
```

```
            if(automation[curr_vertex][k].symbol == collection[i].first[j]){
                branch_len++;
                cout << "Переход с символом " << collection[i].first[j] << " уже существует\n";
                if(j == collection[i].first.size() - 1) {
                    automation[curr_vertex][k].ispattern = true;
                    automation[curr_vertex][k].pos = i + 1;
                    automation[curr_vertex][k].subpatternspos.push_back(collection[i].second);
                }
                //if(automation[curr_vertex][k].ispattern)
                  //  automation[curr_vertex][k].subpatternspos.push_back(collection[i].second);
                curr_vertex = automation[curr_vertex][k].index;
                flag = true;
                break;
            }
        }
        if(flag)
            continue;
        number_of_vertices++;
        automation.emplace_back();
        if(j == collection[i].first.size() - 1)
            automation[curr_vertex].emplace_back(Node(collection[i].first[j],number_of_vertices,true, false,branch_len,i
+ 1,collection[i].second));
        else
                                automation[curr_vertex].emplace_back(Node(collection[i].first[j],number_of_vertices,  false,
false,branch_len,-1));
        prev_vertices.emplace_back(curr_vertex);
        curr_vertex = number_of_vertices;
        branch_len++;
        cout << "Создали новую вершину с символом " << collection[i].first[j] << "\n";
    }
    cout << "Шаблон " << collection[i].first << " полностью добавлен\n";
  }
}

void addSuffixEdge(vector<vector<Node>> &automation, int curr_vertex, char symbol,int main_vertex,vector<int>
&prev_vertices,vector<bool> &visited) {
  if(automation[curr_vertex].size() && automation[curr_vertex][automation[curr_vertex].size() - 1].suffix_edge)
    return;
  if(!curr_vertex || !prev_vertices[curr_vertex]){
    automation[curr_vertex].emplace_back(Node('$',0, false, true,0,-1));
    if(!curr_vertex)
        cout << "Установили суффикс-ссылку от корня в корень\n";
```

```
        else
            cout << "Установили суффикс-ссылку от " << getName(automation,prev_vertices,curr_vertex) << " в корень" << "\n";
        return;
    }
    curr_vertex = prev_vertices[curr_vertex];
    char curr_symbol;
    while (curr_vertex){
        for(int i = 0;i < automation[prev_vertices[curr_vertex]].size();i++)
            if(automation[prev_vertices[curr_vertex]][i].index == curr_vertex){
                curr_symbol = automation[prev_vertices[curr_vertex]][i].symbol;
                break;
            }
        addSuffixEdge(automation,curr_vertex,curr_symbol,curr_vertex,prev_vertices,visited);
        curr_vertex = automation[curr_vertex][automation[curr_vertex].size() - 1].index;

        for(int i = 0;i < automation[curr_vertex].size();i++){
            if(automation[curr_vertex][i].symbol == symbol && !automation[curr_vertex][i].suffix_edge){
                if(automation[curr_vertex][i].index != main_vertex){
                                    automation[main_vertex].emplace_back(Node(symbol,automation[curr_vertex][i].index, automation[curr_vertex][i].ispattern,
                                                    true,automation[curr_vertex][i].curr_branch_len, automation[curr_vertex][i].pos,automation[curr_vertex][i].subpatternspos));

                                                cout << "Установили суффикс ссылку из вершины " << getName(automation,prev_vertices,main_vertex) << " в вершину " << getName(automation,prev_vertices, automation[curr_vertex][i].index) << "\n";
                }
                else {
                    automation[main_vertex].emplace_back(Node('$', 0, false, true, 0, -1));
                                                cout << "Установили суффикс ссылку из вершины " << getName(automation,prev_vertices,main_vertex) << " на корень " << "\n";
                }
                return;
            }
        }
    }
     cout << "Установили суффикс ссылку из вершины " << getName(automation,prev_vertices,main_vertex) << " на корень " << "\n";
    automation[main_vertex].emplace_back(Node('$',0, false, true,0,-1));
}
```

```
void constructAutomation(vector<vector<Node>> &automation,int curr_vertex,vector<int> &prev_vertices,vector<bool> &visited){
    char symbol;
    for(int i = 0;i < automation[prev_vertices[curr_vertex]].size();i++)
        if(automation[prev_vertices[curr_vertex]][i].index == curr_vertex){
            symbol = automation[prev_vertices[curr_vertex]][i].symbol;
            break;
        }
    addSuffixEdge(automation,curr_vertex, symbol,curr_vertex,prev_vertices,visited);
    for(int i = 0;i < automation[curr_vertex].size();i++)
        if(!automation[curr_vertex][i].suffix_edge)
            constructAutomation(automation,automation[curr_vertex][i].index,prev_vertices,visited);
}

void next(vector<vector<Node>> &automation, int curr_vertex,char symbol,int &new_vertex){
    for(int j = 0;j < automation[curr_vertex].size();j++)
        if(automation[curr_vertex][j].symbol == symbol && !automation[curr_vertex][j].suffix_edge){
            curr_vertex = automation[curr_vertex][j].index;
            new_vertex = curr_vertex;
            return;
        }

    if(!curr_vertex) {
        new_vertex = curr_vertex;
        return;
    }
    next(automation, automation[curr_vertex][automation[curr_vertex].size() - 1].index,symbol,new_vertex);
}

void findMatching(string &text,vector<vector<Node>> &automation,vector<int> &prev_vertices,int size_of_pattern,int pattern_len){
    cout << "Начинаем обход текста\n";
    vector<int> counter;
    counter.resize(text.size());
    int curr_vertex = 0;
    int mediator;
    char symbol;
    int new_vertex;
    int prev = 0;
    for(int i = 0;i < text.length();i++){
        symbol = text[i];
        next(automation, curr_vertex,symbol,new_vertex);
```

```
        curr_vertex = new_vertex;
        cout << "Текущий символ " << symbol << "\n";
        cout << "Текущая вершина: " << getName(automation,prev_vertices,curr_vertex) << "\n";
        mediator = curr_vertex;
        prev = prev_vertices[curr_vertex];
        while (automation[mediator][automation[mediator].size() - 1].index != mediator){
          for(int j = 0;j < automation[prev].size();j++)
            if(automation[prev][j].index == mediator) {
              if (automation[prev][j].ispattern) {
                cout << "найден терминальный символ\n";
                cout << "увеличиваем счётчики\n";
                for(int k = 0;k < automation[prev][j].subpatternspos.size();k++) {
                  if (i - automation[prev][j].curr_branch_len + 1 -
                    automation[prev][j].subpatternspos[k] < 0)
                    continue;
                  counter[i - automation[prev][j].curr_branch_len + 1 -
                      automation[prev][j].subpatternspos[k]]++;

                  if(counter[i - automation[prev][j].curr_branch_len + 1 -
                        automation[prev][j].subpatternspos[k]] == 6){
                  }
                }
              }
              break;
            }
          prev = mediator;
          mediator = automation[mediator][automation[mediator].size() - 1].index;
        }
      }
      cout << "проверяем счётчики\n";
      vector<int> output;
      for(int i = 0;i <= counter.size() - pattern_len;i++)
        if(counter[i] == size_of_pattern) {
          output.push_back(i + 1);
          cout << "Нашли вхождение на позиции " << i + 1 << "\n";
        }

      for(int i = 0;i < output.size();i++)
        cout << output[i] << "\n";
}

int main() {
```

```
    string pattern;
    string text;
    int size = 0;
    char wildcard;
    cin >> text;
    cin >> pattern;
    cin >> wildcard;
    vector<pair<string,int>> collection;
    size = split(collection, pattern,wildcard);
    vector<vector<Node>> automation(1);
    vector<pair<int,int>> output;
    vector<int> prev_vertices;
    prev_vertices.push_back(0);
    constructTrie(collection,automation,prev_vertices);
    vector<bool> visited(automation.size());
    constructAutomation(automation,0,prev_vertices,visited);
    cout << "Автомат построен\n";
    cout << "Размер автомата\n" << automation.size() << "\n";
    findMatching(text,automation,prev_vertices,size,pattern.length());
}
```