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. 7304 _____ Давыдов А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы.

Изучение алгоритма точного поиска набора образцов в строке – алгоритм Ахо-Корасик.

Задача.

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.

Sample Input:

```
CCCA
1
CC
```

Sample Output:

```
1 1
2 1
```

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

   В шаблоне встречается специальный символ, именуемого джокером (wildcard), который "совпадает" с любым символом. По заданному

содержащему шаблоны образцу P необходимо найти все вхождения P в текст T.

Например, образец ab??c? с джокером ? встречается дважды 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,$1 \leq |T| \leq 100000$)

Шаблон (P,$1 \leq |P| \leq 40$)

Символ джокера

Выход:

Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер). Номера должны выводиться в порядке возрастания.

Sample Input:

```
ACT
A$
$
```

Sample Output:

```
1
```

Описание алгоритма.

1. **Алгоритм Ахо-Корасик**
    1. На основе набора паттернов строим префиксное дерево.
    2. Рассматриваем префиксное дерево как конечный детерминированный автомат. Стартовая позиция в корне.
    3. Считываем первый символ текста.
    4. Переходим в следующее состояние по ребру, обозначающему этот символ. Если такого ребра нет, то идем по суффиксной ссылке. Если суффиксной ссылки нет, то запоминаем символ, по которому пришли в данный узел, берем суффиксную ссылку родителя и пытаемся перейти по этому символу. Данный шаг выполняется рекурсивно, пока не найден такой переход или не достигнут корень.
    5. Выполняем шаг 4 до тех пор, пока не найдем валидный переход по текущему символу текста или пока не достигнем корня.
    6. Проверяем является ли текущее состояние каким-либо паттерном. Для этого переходим по суффиксным ссылкам, проверяя соответствующий флаг. Если это паттерн, выводим его номер и позицию в тексте в консоль.
    7. Если не конец текста переходим к шагу 3.

2. **Алгоритм Ахо-Корасик + паттерн с джокером**
    1. Разбиваем паттерн на части, разделенные джокерами. Запоминаем их позицию в паттерне (индекс).
    2. Используя алгоритм Ахо-Корасик ищем позицию этих паттернов в тексте.
    3. Создаем массив нулей, размер которого совпадает с длиной текста.

4. Для каждого вхождения паттерна инкрементируем $i - j + 1$ позицию в массиве, где i – индекс вхождения, j – позиция данного паттерна в исходном паттерне.
5. Таким образом, в тех позициях массива, где значение совпадает с количеством паттернов, присутствует совпадение с исходным паттерном.

Вывод

В ходе лабораторной работы был изучен алгоритм Ахо-Корасик для точного поиска набора образцов в строке. Данный алгоритм был реализован на C++. Также, этот алгоритм был использован для поиска в строке паттерна, содержащего символ джокер. Для данного случая было решено разбить строку на паттерны по символам джокерам и добавить их в бор, сохраняя индексы этих шаблонов в исходной строке паттерне, также в программу добавлена структура для хранения индекса совпадения паттерна в тексте и номера паттерна. Затем применяется модифицированный алгоритм Ахо-Корасика и выводятся индексы совпадений.

Исходный код программы

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

using namespace std;


struct Vertex
{
    int next_vrtx[5];
    int auto_move[5]; //
    int par; // parrent of this vertex
    char symb; // transfer symbol from parrent
    int pat_num[40]; // number of pattern designated by this vertex
    bool flag;
    int suff_link;
    int suff_flink; // good suffix link which has flag = true
};


struct EntryPattern
{
    EntryPattern(int index, int pat_num) : index(index), pat_num(pat_num) {}

    int index;
    int pat_num;
};

vector<Vertex> bohr;
vector<string> pattern;

map<char, char> alpha_idx{
    {'A', 0},
    {'C', 1},
    {'G', 2},
    {'N', 3},
    {'T', 4},
};


//p - parrent, c - symbol of transfer
Vertex make_bohr_vrtx(int p, char c)
{
   Vertex v;

   memset(v.next_vrtx, 255, sizeof(v.next_vrtx)); // filling all bytes of
next_vrtx by -1
   memset(v.auto_move, 255, sizeof(v.auto_move));
   memset(v.pat_num, 255, sizeof(v.pat_num));
   v.flag=false;
   v.suff_link=-1; //изначально - суф. ссылки нет
   v.par=p;
   v.symb=c;
   v.suff_flink=-1;

   return v;
}


void bohr_ini()
{
   //добавляем единственную вершину - корень
   bohr.push_back(make_bohr_vrtx(-1, -1));
}
```

```
void add_string_to_bohr(const string& s)
{
   int num=0; //начинаем с корня

   for(int i = 0; i < s.length(); i++)
   {
      char ch = alpha_idx[s[i]]; //получаем номер в алфавите

      if(bohr[num].next_vrtx[ch]==-1)
      {
         bohr.push_back(make_bohr_vrtx(num, ch));
         bohr[num].next_vrtx[ch]=bohr.size()-1;
      }

      num=bohr[num].next_vrtx[ch];
   }

   bohr[num].flag=true;
   pattern.push_back(s);

   // if pattern has same subpatterns
   for(int i = 0; i < 40; ++i)
       if(bohr[num].pat_num[i] == -1)
       {
           bohr[num].pat_num[i] = pattern.size() - 1;
           break;
       }
}


int get_auto_move(int v, char ch);

int get_suff_link(int v)
{
   if (bohr[v].suff_link==-1) //если еще не считали
      if (v==0||bohr[v].par==0) //если v - корень или предок v - корень
         bohr[v].suff_link=0;
      else
         bohr[v].suff_link=get_auto_move(get_suff_link(bohr[v].par),
bohr[v].symb);

   return bohr[v].suff_link;
}


int get_auto_move(int v, char ch)
{
   if (bohr[v].auto_move[ch]==-1)
   {
      if (bohr[v].next_vrtx[ch]!=-1)
         bohr[v].auto_move[ch]=bohr[v].next_vrtx[ch];
      else
      {
         if (v==0)
            bohr[v].auto_move[ch]=0;
         else
            bohr[v].auto_move[ch]=get_auto_move(get_suff_link(v), ch);
      }
   }

   return bohr[v].auto_move[ch];
}


int get_suff_flink(int v)
{
```

```
   if (bohr[v].suff_flink==-1)
   {
      int u=get_suff_link(v);
      if (u==0) //либо v - корень, либо суф. ссылка v указывает на корень
         bohr[v].suff_flink=0;
      else
         bohr[v].suff_flink=(bohr[u].flag) ? u : get_suff_flink(u);
   }

   return bohr[v].suff_flink;
}


//implemenation of searching by automation
void check(int v, int i, vector<EntryPattern> &ep)
{
    for(int u=v; u!=0; u=get_suff_flink(u))
        if (bohr[u].flag)
            for(int j = 0; j < 40; ++j)
            {
                if(bohr[u].pat_num[j]!= -1)
                    ep.push_back(EntryPattern(i -
pattern[bohr[u].pat_num[j]].length(), bohr[u].pat_num[j]));
                else
                    break;
            }
}


void aho_corasik(const string& s, vector<EntryPattern> &ep){
    int u=0;

    for(int i=0 ; i < s.length(); i++)
    {
        u=get_auto_move(u, alpha_idx[s[i]]);
        check(u, i+1, ep);
    }
}


int main()
{
    bohr_ini();

    string text, pattern_str;
    char joker;

    cin >> text >> pattern_str >> joker;

    std::vector<int> matches(text.length(), 0);
    std::vector<std::string> patterns;
    std::vector<int> patterns_pos;



    for (int i = 0; i < pattern_str.length(); i++)
    {
        std::string pat;

        if (pattern_str[i] != joker)
        {
            patterns_pos.push_back(i + 1);

            for (int j = i; pattern_str[j] != joker && j !=
pattern_str.length(); j++)
            {
                pat += pattern_str[j];
```

```
                    i++;
                }

                patterns.push_back(pat);
            }
        }

        for(string &el : patterns)
            add_string_to_bohr(el);

        vector<EntryPattern> ep;
        aho_corasik(text, ep);

        for(int i = 0; i < ep.size(); ++i)
        {
            if(ep[i].index < patterns_pos[ep[i].pat_num] - 1)
                continue;

            //increment i + j - 1 position in vector of mathes
            matches[ep[i].index - patterns_pos[ep[i].pat_num] + 1]++;

            /*
             * if in position i + j - 1 has count of matches = count of subpatterns
of pattern_str
             * second condition check that pattern does not go beyond the text
            */
            if(matches[ep[i].index - patterns_pos[ep[i].pat_num] + 1] ==
patterns.size() &&
                    ep[i].index - patterns_pos[ep[i].pat_num] + 1 <= text.length() -
pattern_str.length())
                cout << ep[i].index - patterns_pos[ep[i].pat_num] + 2 << endl;
        }

        return 0;
}
```