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 _____ Субботин А.С.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2019

Цель работы:

Изучить и реализовать на языке программирования C++ алгоритм Ахо-Корасик, который осуществляет поиск множества подстрок в тексте с помощью построения бора.

Формулировка задачи:

- Разработайте программу, решающую задачу точного поиска набора образцов.

  Вход:
  Первая строка содержит текст (T,1≤|T|≤100000T,1≤|T|≤100000).
  Вторая - число nn (1≤n≤30001≤n≤3000), каждая следующая из nn строк содержит шаблон из набора P={p1,…,pn}1≤|pi|≤75P={p1,…,pn}1≤|pi|≤75
  Все строки содержат символы из алфавита {A,C,G,T,N}{A,C,G,T,N}
  Выход:
  Все вхождения образцов из PP в TT.
  Каждое вхождение образца в текст представить в виде двух чисел - ii pp
  Где ii - позиция в тексте (нумерация начинается с 1), с которой начинается вхождение образца с номером pp
  (нумерация образцов начинается с 1).
  Строки выхода должны быть отсортированы по возрастанию, сначала номера позиции, затем номера шаблона.
- Используя реализацию точного множественного поиска, решите задачу точного поиска для одного образца с *джокером*.

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

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

  Символ джокер не входит в алфавит, символы которого используются в TT. 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}{A,C,G,T,N}

  Вход:
  Текст (T,1≤|T|≤100000T,1≤|T|≤100000)
  Шаблон (P,1≤|P|≤40P,1≤|P|≤40)
  Символ джокера
  Выход:
  Строки с номерами позиций вхождений шаблона (каждая строка содержит только один номер).
  Номера должны выводиться в порядке возрастания.

Ход работы:

Ахо-Корасик:

1) Выбирается образец. Если образцов не осталось – переходим на шаг 3.
2) Выбранный образец добавляется в бор, переходим на шаг 1.
3) Текущий символ = первый символ текста; текущая вершина = корень.
4) Переход из текущей вершины по текущему символу с помощью хода по автомату.
5) Проверка на встретившиеся шаблоны в вершине, выбранной на шаге 4, с помощью перехода по хорошим суффиксальным ссылкам.
6) Если текст не закончился, то текущая вершина = вершина, выбранная на шаге 4, текущий символ = следующий символ текста, переход на шаг 4.

Образец с джокером:

1) Деление образца на буквенные подстроки.
2) Запоминаем количество джокеров между каждой такой подстрокой, а также в начале и конце образца.
3) Алгоритмом Ахо-Корасик находит индексы вхождения каждой подстроки
4) Сопоставляя количество джокеров между строками и их индексы, мы вычисляем, есть ли в тексте данный шаблон. Если есть – запоминаем индекс.

Результаты работы программы:

Ахо-Корасик:
Входные данные:
CCCA
1
CC
Выходные данные:
1 1
2 1

Джокер:
Входные данные:
ACT
A$
$
Выходные данные:
1

Выводы:

В ходе выполнения данной лабораторной работы был изучен и реализован на языке программирования c++ алгоритм Ахо-Корасик, результатом работы которого являются индексы вхождений подстроки в тексте и номер этой подстроки. Для работы этого алгоритма понадобилось реализовать бор, содержащий все подстроки, которые необходимо найти. Бор был реализован в качестве конечного детерминированного автомата, для прохода по которому используются суффиксальные ссылки. Вычислительная сложность данного алгоритма зависит от длины всех подстрок, размера алфавита, длины текста и длины всех совпадений, а точнее – сумма произведения первых двух и всего остального.

Приложение 1. Код программы

```
#include <iostream>
#include <vector>
#include <string>
#include <cstring>
#include <cstdint>

using namespace std;
typedef int32_t type;

struct bohr_vertex{
    type next_vertex[5]; //A, C, G, T, N
    bool endofshape;     //является ли концом подстроки
    size_t num;          //номер подстроки
    size_t parent;       //индекс родителя
    size_t sufflink;     //индекс суффиксальной ссылки наибольшего суффикса
    type gotosymbol[5];  //индекс перехода по каждому символу
    type symboltoparent; //возвращает индекс символа, по которому переходится из
родителя
    size_t goodsufflink; //индекс хорошей СС
};

bool Sort(size_t first, size_t second){
    return first < second;
}

class Bohr{
public:
    vector <bohr_vertex> bohr;
    string text;
    vector<size_t> count_j;
    vector<string> patches;
    vector<vector<size_t>> ind;
    Bohr()
    {
        string shape;
        char joker;
        cin >> text >> shape >> joker;

        size_t len = shape.length();
        size_t count_other = 0;
        for(size_t i(0); i < len; i++)
            if(shape[i] == joker)
                count_other++;
            else{
                string pat;
                while(shape[i] != joker && i < len){
                    pat.push_back(shape[i]);
                    i++;
                }
                i--;
                patches.push_back(pat);
                count_j.push_back(count_other);
                count_other = 0;
            }
        count_j.push_back(count_other);
        bohr.push_back({{-1, -1, -1, -1, -1}, false, 0, 0, 0, {-1, -1, -1, -1, -
1}, -1, 0});
        len = patches.size();
        for(size_t i(0); i < len; i++)
```

```
            MakeVert(patches.at(i), i); //i - начинаем нумерацию с 0 (удобство в
обращении с patches)
    }

    void MakeVert(string temp, size_t number)
    {
        size_t len = temp.length();
        size_t index = 0;
        size_t symbol;
        for(size_t i(0); i < len; i++){
            switch(temp.at(i))
            {
                case 'A':
                            symbol = 0;
                            break;
                case 'C':
                            symbol = 1;
                            break;
                case 'G':
                            symbol = 2;
                            break;
                case 'T':
                            symbol = 3;
                            break;
                case 'N':
                            symbol = 4;
                            break;
            }
            if(bohr[index].next_vertex[symbol] == -1){
                bohr.push_back({{-1, -1, -1, -1, -1}, i == len-1, number, index,
0, {-1, -1, -1, -1, -1}, (type)symbol, 0});
                bohr[index].next_vertex[symbol] = bohr.size() - 1;
            }
            index = bohr[index].next_vertex[symbol];
        }
    }

    size_t getSuffLink(size_t vertex)
    {
        if(bohr.at(vertex).sufflink == 0){
            if(vertex == 0 || bohr.at(vertex).parent == 0)
                bohr.at(vertex).sufflink = 0;
            else
                bohr.at(vertex).sufflink =
getLink(getSuffLink(bohr.at(vertex).parent), bohr.at(vertex).symboltoparent);
        }
        return bohr.at(vertex).sufflink;
    }

    size_t getLink(size_t vertex, size_t symbol)
    {
        if(bohr.at(vertex).gotosymbol[symbol] == -1){
            if(bohr.at(vertex).next_vertex[symbol] != -1)
                bohr.at(vertex).gotosymbol[symbol] =
bohr.at(vertex).next_vertex[symbol];
            else
                bohr.at(vertex).gotosymbol[symbol] = (vertex == 0) ? 0 :
getLink(getSuffLink(vertex), symbol);
        }
        return bohr.at(vertex).gotosymbol[symbol];
    }

    size_t getGoodSuffLink(size_t vertex)
```

```
    {
        if(bohr.at(vertex).goodsufflink == 0){
            size_t temp = getSuffLink(vertex);
            if(temp == 0)
                bohr.at(vertex).goodsufflink = 0;
            else
                bohr.at(vertex).goodsufflink = (bohr.at(temp).endofshape) ? temp :
getGoodSuffLink(temp);
        }
        return bohr.at(vertex).goodsufflink;
    }

    void check(type v, size_t i)
    {
        for(type u(v); u != 0; u = getGoodSuffLink(u)){
            if(bohr.at(u).endofshape){
                size_t delta = 0;
                type temp = u;
                while(bohr.at(temp).parent != 0){
                    temp = bohr.at(temp).parent;
                    delta++;
                }
                //cout << i - delta << " " << bohr.at(u).num << endl;  //Сделать
ту самую проверку на соотносимость массивов
                ind.at(bohr.at(u).num).push_back(i-delta);
            }
        }
    }

    void AHO()
    {
        ind.resize(patches.size());
        size_t vertex = 0;
        size_t len = text.length();
        size_t symbol;
        for(size_t i(0); i < len; i++){
            switch(text.at(i))
            {
                case 'A':
                            symbol = 0;
                            break;
                case 'C':
                            symbol = 1;
                            break;
                case 'G':
                            symbol = 2;
                            break;
                case 'T':
                            symbol = 3;
                            break;
                case 'N':
                            symbol = 4;
                            break;
            }
            vertex = getLink(vertex, symbol);
            check(vertex, i+1);
        }
    }

    vector<size_t> correct;

    void Del_wrong(size_t index){
        while(index < ind.size())
```

```
            ind.at(index++).pop_back();
    }

    void Del_more(size_t index){
        for(size_t i(0); i < index; i++)
            while(ind.at(i).back() + count_j.at(i+1) + patches.at(i).size() >
ind.at(index).back())
                if(ind.at(i).size() == 1)
                    ind.at(i).at(0)=0;
                else
                    ind.at(i).pop_back();
    }

    bool empYT(){
        for(size_t i(0); i < ind.size(); i++)
            if(ind.at(i).empty() || !ind.at(i).at(0))
                return false;
        return true;
    }

    void Fun(size_t index)
    {
        if(!empYT())
            return;
        if(index == ind.size() - 1){
            Del_more(index);
            if(ind.at(index).back() + count_j.at(index+1) +
patches.at(index).size() - 1 <= text.size()){
                if(!index){
                    if(ind.at(0).back() - count_j.at(0) > 0){
                        correct.push_back(ind.at(0).back() - count_j.at(0));
                        ind.at(index).pop_back();
                        if(empYT())
                            Fun(index);
                    }
                }
                else
                    Fun(index-1);
            }
            else{
                ind.at(index).pop_back();
                if(empYT())
                    Fun(index);
            }
        }
        else if(!index){
            if(ind.at(0).back() - count_j.at(0) > 0
                    && ind.at(0).back() + count_j.at(1) + patches.at(0).size() ==
ind.at(1).back()){
                correct.push_back(ind.at(0).back() - count_j.at(0));
                Del_wrong(0);
            }
            else
                ind.at(ind.size() - 1).pop_back();
            if(empYT())
                Fun(ind.size() - 1);
        }
        else{
            Del_more(index);
            if(ind.at(index).back() - count_j.at(index) - patches.at(index-
1).size()!= ind.at(index-1).back()){
                //Del_wrong(index);
                ind.at(ind.size() - 1).pop_back();
```

```
                if(empYT())
                    Fun(ind.size() - 1);
            }
            else
                Fun(index-1);
        }
    }

    void Out(){
        for(size_t i(0); i < patches.size(); i++)
            for(size_t j(i+1); j < patches.size(); j++)
                if(patches.at(i) == patches.at(j))
                    ind.at(j) = ind.at(i);
        if(empYT()){
            Fun(ind.size() - 1);
            for(int i(correct.size() - 1); i >= 0; i--)
                cout << correct.at(i) << endl;
        }
    }
};

int main(){
    Bohr bohr;
    bohr.AHO();
    bohr.Out();
    return 0;
}
```