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

Студентка гр. 7383 _____ Ханова Ю.А.

Преподаватель _____ Жангиров Т. Р.

Санкт-Петербург
2019

Цель работы

Исследовать и реализовать задачу точного поиска набора образцов при помощи алгоритма Ахо-Корасик.

Реализация задачи

В ходе выполнения данной работы был реализован алгоритм Ахо-Корасик(АК). Основной задачей является составление бора и построения по нему конечного детерминированного автомата.

Сначала по заданному набору шаблонов строится бор: в качестве ребер используются буквы, составляющие шаблоны. При этом вершины, обозначающие завершение какого либо шаблона отмечаются флагом.

Затем необходимо построить конечный детерминированный автомат. Состояние автомата — это какая-то вершина бора. Переход из состояний осуществляется по 2 параметрам — текущей вершине v и символу ch. по которому нам надо сдвинуться из этой вершины. Поконкретнее, необходимо найти вершину u, которая обозначает длиннейшую строку, состоящую из суффикса строки v (возможно нулевого) + символа ch. Если такого в боре нет, то идем в корень.

Назовем суффиксной ссылкой вершины v указатель на вершину u, такую что строка u — наибольший собственный суффикс строки v, или, если такой вершины нет в боре, то указатель на корень. В частности, ссылка из корня ведет в него же. Для каждой вершины определяется суффиксная ссылка.

Затем алгоритм работает следующим образом: если из текущей вершины есть ребро с символом ch, то пройдем по нему, в обратном случаем пройдем по суффиксной ссылке и запустимся рекурсивно от новой вершины.

Также добавляется понятие хороших ссылок, они представляют собой ближайший суффикс, имеющийся в боре, для которого flag=true. Число «скачков» при использование таких ссылок уменьшится и станет пропорционально количеству искомых вхождений, оканчивающихся в этой позиции.

Тестирование

Программа собрана в операционной системе Ubuntu 17.04 с использованием компилятора g++. В других ОС и компиляторах тестирование не проводилось. Результаты тестирования показали, что поставленная цель выполнена. Результаты тестирования представлены в Приложении Б.

Так же было проведено исследование алгоритма. Сложность оказывается O(N+t), где t — количество всех возможных вхождений всех строк-образцов в s, N — длина строки. Если быть точным и учитывать вычисления автомата и суф. ссылок, то алгоритм работает O(M*k+N+t), где M=bohr.size(). Память - константные массивы размера k для каждой вершины бора, откуда выливается оценка O(M*k).

Выводы

В ходе выполнения лабораторной работы была решена задача поиска набора образцов на языке C++, и исследован алгоритм Ахо-Корасик. Полученный алгоритм имеет сложность O(M*k). Код программ представлен в Приложении А.

ПРИЛОЖЕНИЕ А.
КОД ПРОГРАММЫ

```
#include <iostream>
#include <vector>
#include <string.h>

#define A 6

using namespace std;

struct bohr_vrtx{
    int next_vrtx[A];
    int path_num;
    bool flag;
    int suff_link;
    int auto_move[A];
    int par;
    char symbol;
    int suff_flink;
};

vector<bohr_vrtx> bohr;
vector <string> pattern;

bohr_vrtx make_bohr_vrtx(int par, char symbol){    //Функции создания
новой вершины и инициализации бора:
    bohr_vrtx vertex;
    memset((vertex.next_vrtx), 255, sizeof(vertex.next_vrtx));
    vertex.flag = false;
    vertex.suff_link = -1;
    memset((vertex.auto_move), 255, sizeof(vertex.auto_move));
    vertex.par = par;
    vertex.symbol = symbol;
    vertex.suff_flink = -1;
return vertex;
}

void init_bohr() {
    bohr.push_back(make_bohr_vrtx(-1, -1));
}

int find(char symbol) {
    int ch;
    switch (symbol)
    {
```

```
    case 'A':
        ch = 0;
        break;
    case 'C':
        ch = 1;
        break;
    case'G':
        ch = 2;
        break;
    case 'T':
        ch = 3;
        break;
    case 'N':
        ch = 4;
        break;
    default:
        break;
    }
    return ch;
}

void add_string_to_bohr(const string& s){
   int num=0;
//начинаем с корня
   for (int i=0; i<s.length(); i++){
      char ch = find(s[i]);
//получаем номер в алфавите
      if (bohr[num].next_vrtx[ch]==-1){
//-1 - признак отсутствия ребра
         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);
   bohr[num].path_num=pattern.size()-1;
}

bool is_string_in_bohr(const string& s){
   int num=0;
   for (int i=0; i<s.length(); i++){
      char ch = find(s[i]);
      if (bohr[num].next_vrtx[ch]==-1){
         return false;
         }
```

```
      num=bohr[num].next_vrtx[ch];
   }
   return true;
}

int get_auto_move(int v, char ch);

int get_suff_link(int v) {
    if (bohr[v].suff_link == -1) {
        if (v == 0 || bohr[v].par == 0) {
            bohr[v].suff_link = 0;
        }
        else {
            bohr[v].suff_link =
get_auto_move(get_suff_link(bohr[v].par), bohr[v].symbol);  //пройдем по
суф.ссылке предка и запустим переход по символу.
        }
    }
    return bohr[v].suff_link;
}

int get_auto_move(int v, char ch){                          //вычисляемая
функция переходов
   if (bohr[v].auto_move[ch]==-1)
      if (bohr[v].next_vrtx[ch]!=-1)    //если из текущей вершины есть
ребро с символом ch
         bohr[v].auto_move[ch]=bohr[v].next_vrtx[ch];
//то идем по нему
      else
         if (v==0)
            bohr[v].auto_move[ch]=0;
         else
//иначе перейдем по суффиксальной ссылке
            bohr[v].auto_move[ch]=get_auto_move(get_suff_link(v), ch);
   return bohr[v].auto_move[ch];
}

void check(int v,int i);

void find_all_pos(const string& s){
    int u=0;
    for(int i=0;i<s.length();i++){
        u=get_auto_move(u,find(s[i]));
        check(u,i+1);
    }
}
```

```
int get_suff_flink(int v) {
    if (bohr[v].suff_flink == -1) {
        int u = get_suff_link(v);
        if (u == 0) {
            bohr[v].suff_flink = 0;
        }
        else {
            bohr[v].suff_flink = (bohr[u].flag) ? u : get_suff_flink(u);
//если для вершины по суф.ссылке flag=true, то это искомая вершина,
иначе рекурсия.
        }
    }
    return bohr[v].suff_flink;
}

void check(int v, int i) {  //хождение по хорошим суффиксальнм ссылкам
из текущей позиции, учитывая, что эта позиция оканчивается на симво i
    for (int u = v; u != 0; u = get_suff_flink(u)) {
        if (bohr[u].flag) {
            cout << i - pattern[bohr[u].path_num].length() + 1 << " " <<
bohr[u].path_num +1<< endl;
        }
    }
}

int main(){
  string str;
  int n;
  init_bohr();
  cin >> str;
  cin >> n;
  for (int i = 0; i < n; i++) {
      string tmp;
      cin >> tmp;
      add_string_to_bohr(tmp);
  }
  find_all_pos(str);
return 0;
}
```

ПРИЛОЖЕНИЕ Б.

ТЕСТОВЫЕ СЛУЧАИ

Результаты тестов представлены в табл. 1-2.

Таблица 1 – запуск lr5.cpp.

Входные данные	Выходные данные
CCCA 1 CC	1 1 2 1

Таблица 2 – запуск lr5_2.cpp.

Входные данные	Выходные данные
ACT A$ $	1