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. 8382 _____ Наконечная А. Ю.

Преподаватель _____ Фирсов М. А.

Санкт-Петербург

2020

Цель работы.

Изучить принцип работы алгоритма Ахо-Корасик для множественного поиска индексов вхождения строк-паттернов в строку-текст. Разработать программу, которая реализует алгоритм Ахо-Корасик.

Постановка задачи.

Разработайте программу, решающую задачу точного поиска набора образцов.

Вход:

Первая строка содержит текст (T, $1 \leq |T| \leq 100000$). Вторая - число n ($1 \leq n \leq 3000$), каждая следующая из n строк содержит шаблон из набора $P=\{p1,\ldots,pn\}$ $1 \leq |pi| \leq 75$. 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 22 3

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? 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, $1 \leq |T| \leq 100000$)

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:

ACTANCA

A$$A$$

Sample Output:

1

Индивидуальное задание.

Вариант 3.

Вычислить длину самой длинной цепочки из суффиксных ссылок и самой длинной цепочки из конечных ссылок в автомате.

Описание алгоритма.

На вход алгоритму передается строка-текст и строки-образцы. Суть алгоритма заключена в использование структуры данных — бора и построения по нему конечного детерминированного автомата. По строкам-образцам строится бор - префиксное дерево, которое строится следующим образом: корневой вершиной является нулевая строка. Поочередно считываются все строки-образцы и посимвольно обрабатываются. Если из текущей вершины уже есть переход по этому символу, то он осуществляется, если же перехода нет, то создается новая вершина, которой присваивается данный символ, эта вершина становится текущей. Если этот символ конечный в строке-шаблоне, то устанавливается терминальный флаг. Бор хранится как массив вершин, где каждая вершина имеет свой уникальный номер, у корня - 0.

Построение автомата по бору: состояние автомата - какая-то вершина бора. Переход из состояний осуществляется по двум параметрам - текущей вершине v и символу u, по которому нужно сдвинуться из этой вершины. Цикл в алгоритме пробегается по всему тексту и считывает на каждой итерации символ u. Из текущей вершины v (изначально из корня) совершается переход по символу u. Если прямой переход есть, то он выполняется, иначе происходит переход по суффиксной ссылке.

Суффиксная ссылка для каждой вершины v - это индекс вершины в боре, в которой оканчивается наидлиннейший собственный суффикс строки, соответствующей вершине v. Суффиксная ссылка находится следующим образом: происходит переход в вершину по суффиксной ссылке родителя, а затем из нее совершается переход по заданному символу. После выполненного пе-

рехода происходит проверка. Из текущей вершины происходит переход по суффиксным ссылкам, пока не дойдем до корня. Если вершина терминальная - то вхождение было найдено. Алгоритм завершает работу, когда будет обработан каждый символ строки-текста.

Описание алгоритма Ахо-Корасик с джокером.

Для поиска шаблона, содержащего джокер, строка-шаблон разбивается на части по символу джокера. Для каждой такой части хранится позиция в первоначальной строке. Из полученных подстрок строится бор, их вхождение ищется в тексте с помощью алгоритма Ахо-Корасик. Создается вектор C - длины текста, инициализированный нулями. При каждом совпадении вычисляется место в этом массиве, которому соответствует текущая подстрока по формуле j – l + 1, j - позиция в тексте, l - позиция подстроки в строке-шаблоне. C [j – l + 1] увеличивается на 1. Далее в цикле перебираются все элементы массива. Если значение на каком-либо месте совпадает с количеством подстрок, то найдено соответствие по шаблону.

Описание индивидуализации.

Заведены две переменные, инициализированные нулями, для максимальной длины суффиксной ссылки и максимальной длины конечной ссылки. Во время проверки, когда из текущей вершины происходит переход по суффиксным ссылкам до корня, происходит подсчет количества ссылок. Для подсчета наибольшего числа конечный ссылок счетчик увеличивается не каждую итерацию, а только на терминальном символе. После проверки полученные значения сравниваются с найденными ранее максимальными и при необходимости заменяются.

Описание функций и структур данных.

class BohrVertex - класс необходимый для хранения всё информации о вершинах и боре.

class Bohr - класс необходимый для хранения информации о боре и работы с ним.

map <char, int> nextVertex - контейнер для вершин потомков.

map <char,int> autoMove - контейнер, который необходим для отображения возможности передвижения. Хранит в себе вершину и индекс вершины в боре, в которую можно переместиться по символу.

vector<string> patterns - контейнер всех паттернов в в боре.

map <int, vector<int>> result - результат.

Описание функций.

void addPattern(const string& pattern) - функция добавления паттерна в бор, на вход принимает паттерн, который необходимо добавить.

int getSuffLink(int index) - получение суффиксной ссылки. Принимает переменную index - номер вершины в боре, для которой нужно вычислить суффиксную ссылку. Возвращаемое значение - индекс вершины в боре, на которую указывает суффиксная ссылка.

int getAutoMove(int index, char symbol) - метод перехода из вершины vertexIndex по символу symbol. Возвращает индекс вершины в боре, в которую будет совершен переход.

void findAllPos(string textT) - метод поиска всех вхождений строк-образцов. Принимает на вход текст. В методе в цикле обрабатываются все символы исходного текста.

void printRes() - печать ответа.

void printBohr() - печать бора.

Сложность алгоритма.

По времени.

Алгоритм Ахо-Корасик строит бор за O(m * log(k)), где k - количество символов алфавита, m - длина всех строк-образцов, т.к. в худшем случае бор состоит из всех вершин, а для хранения используется map (вставка в которую имеет временную сложность O(log(k))). Алгоритм пройдется по всей длине текста t, получая переходы за log(k), после каждого перехода будут проверены все суффиксные ссылки до корня, которых максимально m штук. В итоге временная сложность алгоритма: O((t + 2m) * log(k)).

По памяти.

Алгоритм Ахо-Корасик имеет сложность по памяти O(m), m - длина всех строк образцов, т.к. в худшем случае в боре хранится каждый символ этих строк.

Тестирование.

№ теста	Тест	Результат
1	CCCA 1 CC	Bohr: Symbol Index Parent SuffLink Sons (index) 0 0 -1 0 1 C 1 0 -1 2 C 2 1 -1 Pattern entry search started. Start from root //------------------// Step by symbol in text: C Search path from 0 (0) to C Path from 0 (0) to C is found Getting sufflink for vertex C (1)

		link not yet defined go to parent Link from C (1) is 0 //----------// Step by symbol in text: C Search path from C (1) to C Path from C (1) to C is found Getting sufflink for vertex C (2) link not yet defined go to parent Getting sufflink for vertex C (1) Link from C (1) is 0 Path from 0 (0) to C is found Link from C (2) is 1 Getting sufflink for vertex C (1) Link from C (1) is 0 //----------// Step by symbol in text: C Search path from C (2) to C Go by suffLink (no child) Getting sufflink for vertex C (2) Link from C (2) is 1

		Path from C (1) to C is found Path from C (2) to C is found Getting sufflink for vertex C (2) Link from C (2) is 1 Getting sufflink for vertex C (1) Link from C (1) is 0 //----------// Step by symbol in text: A Search path from C (2) to A Go by suffLink (no child) Getting sufflink for vertex C (2) Link from C (2) is 1 Search path from C (1) to A Go by suffLink (no child) Getting sufflink for vertex C (1) Link from C (1) is 0 Search path from 0 (0) to A Go by suffLink (no child) to the root

		Path from 0 (0) to A is found Path from C (1) to A is found Path from C (2) to A is found //----------// Result: Max suff link chain length: 2 Max end link chain length: 1 Positions: Offset in text: 1 Pattern №1: CC Offset in text: 2 Pattern №1: CC
2	A 1 A	Bohr: Symbol Index Parent SuffLink Sons (index) 0 0 -1 0 1 A 1 0 -1 Pattern entry search started. Start from root //----------// Step by symbol in text: A Search path from 0 (0) to A Path from 0 (0) to A is found Getting sufflink for vertex A (1) link not yet defined go to parent Link from A (1) is 0 //----------// Result:

		Max suff link chain length: 1 Max end link chain length: 1 Positions: Offset in text: 1 Pattern №1: A
3	NTAG 3 TAGT TAG T	Bohr: Symbol Index Parent SuffLink Sons (index) 0 0 -1 0 1 T 1 0 -1 2 A 2 1 -1 3 G 3 2 -1 4 T 4 3 -1 Pattern entry search started. Start from root //----------// Step by symbol in text: N Search path from 0 (0) to N Go by suffLink (no child) to the root Path from 0 (0) to N is found //----------// Step by symbol in text: T Search path from 0 (0) to T Path from 0 (0) to T is found Getting sufflink for vertex T (1) link not yet defined go to parent Link from T (1) is 0 //----------//

		Step by symbol in text: A Search path from T (1) to A Path from T (1) to A is found Getting sufflink for vertex A (2) link not yet defined go to parent Getting sufflink for vertex T (1) Link from T (1) is 0 Search path from 0 (0) to A Go by suffLink (no child) to the root Path from 0 (0) to A is found Link from A (2) is 0 //----------// Step by symbol in text: G Search path from A (2) to G Path from A (2) to G is found Getting sufflink for vertex G (3) link not yet defined go to parent Getting sufflink for vertex A (2) Link from A (2) is 0 Search path from 0 (0) to G

		Go by suffLink (no child) to the root Path from 0 (0) to G is found Link from G (3) is 0 //------------------// Result: Max suff link chain length: 1 Max end link chain length: 1 Positions: Offset in text: 2 Pattern №2: TAG Offset in text: 2 Pattern №3: T

Выводы.

В ходе выполнения лабораторной работы был реализован алгоритм Ах-оКорасик для множественного поиска индексов вхождения строк-паттернов в строку-текст.

ПРИЛОЖЕНИЕ А

Исходный код программы

piaa_5.cpp

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


using std::string;
using std::cin;
using std::cout;
using std::endl;
using std::map;
using std::vector;
using std::pair;
using std::setw;
using std::move;
using std::sort;


class BohrVertex{
public:
    //флаг конечной, терминальной вершины
    bool flag;
    //имя вершины
    char name;
    //суффиксная ссылка
    int suffLink;
    //индекс в боре родительской вершины
    int parent;
    //длина паттерна (для конечной вершины)
    int offset;
    //номер паттерна (для конечной вершины)
    int patternNumber;


    //контейнер для вершин потомков
    //key - имя value - индекс в боре
    map <char, int> nextVertex;
    //возможность прохода из вершины
    //key - имя value - индекс в боре
    map <char, int> autoMove;
    //инициализация вершины бора
        BohrVertex(char name, int parent, int offset = -1, int
patternNumber = -1, bool flag = false)
```

```
                                    :name(name),        parent(parent),
offset(offset),patternNumber(patternNumber), flag(flag),suffLink(-1){}
};


class Bohr{
public:
    vector<BohrVertex> bohr;
    //контейнер всех паттернов в боре
    vector<string> patterns;
    string text;
    //key - индекс value - номер образца
    map<int, vector<int>> result;


    int maxSuffLink; //максимальная длина цепочки из суффиксных ссылок
    int maxEndLink; //максимальная длина цепочки из конечных ссылок


    Bohr();
    //функция добавления паттерна в бор
    void addPattern(const string& pattern);
    //функция получения суффиксной ссылки
    int getSuffLink(int index);
    //функция перехода
    int getAutoMove(int index, char symbol);
    //функция поиска всех вхождений
    void findAllPos(string textT);
    //печать ответа
    void printRes();
    //печать бора
    void printBohr();
};


//конструктор
Bohr::Bohr(){
    //push_back создание корня
    bohr.emplace_back('0', -1);
    bohr[0].suffLink = 0;
    maxEndLink = 0;
    maxSuffLink = 0;
}


//функция добавления паттерна в бор
void Bohr::addPattern(const string& pattern){
    //рассматриваемый символ в паттерне
    char name;
    //текущая вершина в боре
    int currentVertex = 0;
```

```
    //добавление паттерна в вектор всех паттернов
    patterns.push_back(pattern);
    //проходимся по всему паттерну
    for(char i : pattern){
        name = i;
        //ищем вершину для перехода
        auto it = bohr[currentVertex].nextVertex.find(name);
        //переход невозможен -> создание новой вершины
        if (it == bohr[currentVertex].nextVertex.end()){
            bohr[currentVertex].nextVertex.insert(pair<char, int>(name,
bohr.size()));
            bohr.emplace_back(name, currentVertex);
            currentVertex = bohr.size() - 1;
        }
            //переход по рассматриваемому символу возможен из текущей
вершины
        //значит просто переходим к этой вершине
        else{
            currentVertex = it->second;
        }
    }
    //после добавления паттерна, последняя вершина получает флаг терми-
нальной
    bohr[currentVertex].flag = true;
    //паттерн получает соответствующий номер
    bohr[currentVertex].patternNumber = patterns.size();
    //добавление к данным длины паттерна
    bohr[currentVertex].offset = pattern.size();
}


//функция получения суффиксной ссылки
int Bohr::getSuffLink(int vertexIndex){
         cout << endl << "Getting sufflink for vertex " <<
bohr[vertexIndex].name << " (" << vertexIndex << ")" << endl;
    //суффиксная ссылка еще не была определена
    if(bohr[vertexIndex].suffLink == -1){
        cout << "link not yet defined go to parent" << endl;
        //текущая вершина является корнем, либо же ее предок корень
        if(vertexIndex == 0 || bohr[vertexIndex].parent == 0){
            bohr[vertexIndex].suffLink = 0;
             cout << "Link from " << bohr[vertexIndex].name << " (" <<
vertexIndex << ") is " <<
            bohr[vertexIndex].suffLink << endl << endl;
            return 0;
        }
        //иначе шаг из суффиксной ссылки родителя
        else{
```

```
                                        bohr[vertexIndex].suffLink   =
getAutoMove(getSuffLink(bohr[vertexIndex].parent),
bohr[vertexIndex].name);
        }
    }
       cout << "Link from " << bohr[vertexIndex].name << " (" <<
vertexIndex << ") is " <<
    bohr[vertexIndex].suffLink << endl << endl;
    return bohr[vertexIndex].suffLink;
}


//функция перехода
int Bohr::getAutoMove(int vertexIndex, char symbol){
    auto it = bohr[vertexIndex].autoMove.find(symbol);
    //нет перехода по данному символу из вершины
    if (it == bohr[vertexIndex].autoMove.end()){
         cout << "Search path from " << bohr[vertexIndex].name << " ("
<< vertexIndex << ") to "<< symbol << endl;
        auto it2 = bohr[vertexIndex].nextVertex.find(symbol);
        //прямого перехода не было найдено
        if(it2 == bohr[vertexIndex].nextVertex.end()){
            cout << "Go by suffLink (no child)" << endl;
            if (vertexIndex == 0){//в корень
                cout << "to the root" << endl;
                            bohr[vertexIndex].autoMove.insert(pair<char,
int>(symbol, 0));
            }
            //иначе по суффиксной ссылке символа
            else{
                //ход автомата
                      int move = getAutoMove(getSuffLink(vertexIndex),
symbol);
                            bohr[vertexIndex].autoMove.insert(pair<char,
int>(symbol, move));
            }
        }
        //был найден прямой переход
        else{
                 bohr[vertexIndex].autoMove.insert(pair<char, int>(it2-
>first, it2->second));
        }
    }
       cout << "Path from " << bohr[vertexIndex].name << " (" <<
vertexIndex << ") to " << symbol << " is found " << endl;
    return bohr[vertexIndex].autoMove[symbol];
}


//функция поиска всех вхождений паттернов в текст
```

```
void Bohr::findAllPos(string textT){
    //this->text = textT
    this->text = move(textT);
    //текущая вершина в боре
    int currentVertex = 0;
    cout << "Pattern entry search started. Start from root" << endl;
    //проходимся по всему тексту
    for(int i = 0; i < text.size(); i++){
        cout << "//-------------------//" << endl << "Step by symbol in
text: " << text[i] << endl;
        //шаг по автомату
        //можно ли перейти по символу текста из вершины
        currentVertex = getAutoMove(currentVertex, text[i]);


        int currentSuffLenght = 0;
        int currentEndLenght = 0;


            for(int j = currentVertex; j != 0; j = getSuffLink(j),
currentSuffLenght++){//check
            if(bohr[j].flag){
                currentEndLenght++;
                                                        result[i+2   -
bohr[j].offset].push_back(bohr[j].patternNumber);
            }
        }
         maxEndLink = currentEndLenght>maxEndLink ? currentEndLenght :
maxEndLink;
        maxSuffLink = currentSuffLenght>maxSuffLink ? currentSuffLenght
: maxSuffLink;
    }
    cout << "//-------------------//" << endl << endl;
    printRes();
}


//функция для печати ответа
void Bohr::printRes(){
    //сортировка по номеру паттерна для одинаковых позиций в тексте
    for(auto & it : result){
        sort(it.second.begin(), it.second.end());
    }
     cout << "Result: " << endl << "Max suff link chain length: " <<
maxSuffLink <<
     endl << "Max end link chain length: " << maxEndLink << endl <<
"Positions:" << endl;
    //вывод результата
    if(!result.empty()){
        for(auto & it : result){
```

```
            for(auto it2 = it.second.begin(); it2 != it.second.end();
it2++){
                cout << "Offset in text: " << it.first << " Pattern №"
<< (*it2) << ": " << patterns[(*it2)-1] << endl;
            }
        }
    }
    else{
        cout << "there are no matches for your search!" << endl;
    }


}


//функция для печати получившегося бора
void Bohr::printBohr(){
    cout << endl << "Bohr: " << endl;
     cout << setw(8) << "Symbol" << setw(8) << "Index"<< setw(8) <<
"Parent" << setw(10) <<
    "  SuffLink  "<< setw(10) << "Sons (index)" << endl;
    for(int i = 0; i < bohr.size(); i++){
         cout << setw(8) << bohr[i].name << setw(8) << i << setw(8) <<
bohr[i].parent << setw(8)
        << bohr[i].suffLink << setw(8);
        for(auto & it : bohr[i].nextVertex){
            cout << it.second << " ";
        }
        cout << endl;
    }
}


int main() {
    //создание бора
    Bohr bor;
    //считывание информации(текст, кол-во образцв и сами образцы)
    string text;
    string pattern;
    int countP = 0;
    cin >> text;
    cin >> countP;
    //добавление паттернов к бору
    for(int i = 0; i < countP; i++){
        cin >> pattern;
        bor.addPattern(pattern);
    }
    //печать бора
    bor.printBohr();
    //поиск позиций всех образцов в тексте
```

```
    bor.findAllPos(text);
    return 0;
}
```