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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «ПиАА»

Тема: Алгоритм Ахо Корасик

Студент(ка) гр. 0000 _____ Ивченко А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Ознакомиться с алгоритмом Ахо-Корасик поиска всех вхождений в строке.

Формулировка задания.

Вход:

Первая строка содержит текст ($T, 1 \le |T| \le 100000$).

Вторая - число **n** ($1 \le n \le 3000$), каждая следующая из **n** строк содержит шаблон из набора $P = \{p1, \ldots, pn\}\ 1 \le |pi| \le 75$

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.

Вариант 4. Реализовать режим поиска, при котором все найденные образцы не пересекаются в строке поиска (т.е. некоторые вхождения не будут найдены; решение задачи неоднозначно).

Описание структур данных.

Бор представляется в виде связного списка на указателях, элементы которого имеют следующую структуру:

```
struct Vertex {
	bool root = false;//проверка, является ли текущий элемент корнем
	bool leaf = false;//проверка, является ли текущий элемент листом
	std::vector<char> suffix;// суффикс текущей вершины
	std::map<char, Vertex*> next;//словарь для перехода к соседним элементам по ключу
	std::vector<Vertex*> next_dfs;//список следующих вершин для обхода в глубину
	Vertex* auto_move = nullptr;//действие автомата на этой вершине
	Vertex* suff_link = nullptr;//суффиксная ссылка
	Vertex* parent = nullptr;//ссылка на родителя

};
```

Описание функций.

```
Vertex* automatic(Vertex* a, char ch) – возвращает действие автомата на данной вершине
Vertex* setSuffixLink(Vertex *cur) – устанавливает суффиксную ссылку для данной вершины
void dfs(Vertex* a) – обход в глубину
Vertex* makeTrieVertex(Vertex* par, char a) – создание вершины бора
void addToTrie(std::string& sample, Vertex* root) – добавление в бор
Vertex* disjoint(Vertex* a, char ch, Vertex* root) – возвращает действие автомата на данной вершине(для непересекающихся вхождений)
void check(Vertex* a, int i, std::vector<int>& res) – проверяет является ли элемент листом
void AhoCorasik(std::string& s, Vertex* root, std::ostream& out) – поиск вхождений
```

Описание алгоритма

Алгоритм строит конечный автомат, которому затем передаёт строку поиска. Автомат получает по очереди все символы строки и переходит по соответствующим рёбрам. Если автомат пришёл в конечное состояние, соответствующая строка словаря присутствует в строке поиска.

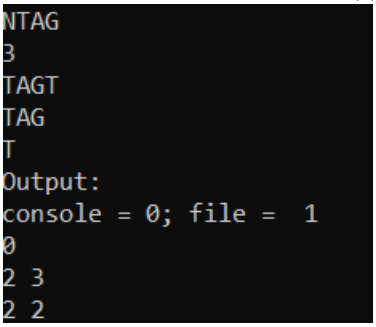
Вычислительная сложность O(nq + H + k), где H — длина текста, в котором производится поиск, n — общая длина всех слов в словаре, q — размер алфавита, k — общая длина всех совпадений.

Для выполнения задания из варианта суффиксные ссылки заменены ссылками в корень бора.

Тестирование.

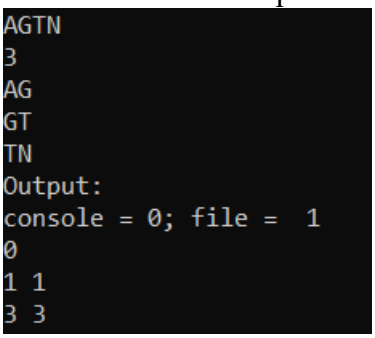
Результат работы алгоритма представлен на рис.1

Рис. 1 — поиск всех вхождений



Результат работы задания из варианта представлен на рис.2

Рис. 2 — поиск непересекающихся подстрок



Вывод.

В ходе лабораторной работы был разобран алгоритм Ахо-Корасик поиска подстроки, также была изучена структура данных бор (префиксное дерево).

Исходный код

```
#include <iostream>
#include <string>
#include <vector>
#include <map>
#include <algorithm>
#include <fstream>

std::map<char, int> alphabet = { {'A', 1},{'C', 2},{'G', 3},{'T', 4}, {'N', 5} };
std::map <std::string, int> pattern;

struct Vertex {

	bool root = false;
	bool leaf = false;
	std::vector<char> suffix;
	std::map<char, Vertex*> next;
	std::vector<Vertex*> next_dfs;
	Vertex* auto_move = nullptr;
	Vertex* suff_link = nullptr;
	Vertex* parent = nullptr;

};
Vertex* automatic(Vertex* a, char ch);

Vertex* setSuffixLink(Vertex *cur) {

	if (cur->root) return cur;

	if (cur->suff_link == nullptr) {
		if (cur->parent->root)
			cur->suff_link = cur->parent;
		else
			cur->suff_link = automatic(setSuffixLink(cur->parent), cur-
>suffix.back());
	}
	return cur->suff_link;
}

std::map<Vertex*, bool> used;

void dfs(Vertex* a) {
	used[a] = 1;

	for (auto i : a->next_dfs) {
		if (!used[i]) {
			//a->suff_link = setSuffixLink(a);
			dfs(i);
		}
	}
}

Vertex* makeTrieVertex(Vertex* par, char a) {

	Vertex* vrtx = new Vertex;
	vrtx->parent = par;
	vrtx->suffix = par->suffix;
	vrtx->suffix.push_back(a);
	return vrtx;
}
```

```
void addToTrie(std::string& sample, Vertex* root) {

	Vertex* ptr = root;
	for (int i = 0; i < sample.length(); i++) {

		if (alphabet[sample[i]]) {

			if (!ptr->next[sample[i]]) {
				Vertex* nextVertex = makeTrieVertex(ptr, sample[i]);
				ptr->next[sample[i]] = nextVertex;
				ptr->next_dfs.push_back(nextVertex);
				ptr = nextVertex;
			}
			else ptr = ptr->next[sample[i]];

		}
		else std::cout << "Incorrect";

	}ptr->leaf = true;
}

Vertex* automatic(Vertex *a, char ch) {


	if (a->auto_move == nullptr || a->root) {
		if (a->next[ch]) {
			a->auto_move = a->next[ch];
			return a->auto_move;
		}
		else {
			if (a->root)
				a->auto_move = a;
			else
				a->auto_move = automatic(setSuffixLink(a), ch);
		}
	}
	return a->auto_move;
}

Vertex* disjoint(Vertex* a, char ch, Vertex* root) {


	if (a->auto_move == nullptr || a->root) {
		if (a->next[ch]) {
			a->auto_move = a->next[ch];
			return a->auto_move;
		}
		else {
			if (a->root)
				a->auto_move = a;
			else
				a->auto_move = disjoint(root, ch, root);//ссылка в корень
		}
	}
	return a->auto_move;
}
void check(Vertex* a, int i, std::vector<int>& res) {

	if(a != nullptr) {

		if (a->leaf) {
			std::string s(a->suffix.begin(), a->suffix.end());
			res.push_back(i - s.length() + 1);
```

```
                        res.push_back(pattern[s] + 1);
                }
        }
}


void AhoCorasik(std::string& s, Vertex* root, std::ostream& out) {

        std::vector<int> result;
        Vertex* u = root;

        for (int i = 0; i < s.length() ; i++) {

                if (alphabet[s[i]]) {

                        u = disjoint(u, s[i], root);
                        //u = automatic(u, s[i]);
                        check(u, i + 1, result);

                }

        }
        for (int a = 0; a < result.size(); a += 2) {
                out << result[a] << ' ' << result[a + 1] << std::endl;
        }
}
int main(){

        int n;
        std::vector<int> result;
        std::string text;

        int input_ch, output_ch;

        std::cout << "Input: console = 0; file = 1\n";
        std::cin >> input_ch;

        Vertex* root = new Vertex;
        root->root = true;

        std::istream* input;

        if (input_ch == 1) {

                std::ifstream infile("input.txt");
                input = &infile;

        }
        else
                input = &std::cin;


        *input >> text;
        *input >> n;

        for (int i = 0; i < n; i++) {
                std::string str;
                *input >> str;
                pattern[str] = i;
                addToTrie(str, root);
        }

        std::cout << "Output:\nconsole = 0; file =  1" << std::endl;
```

```
std::cin >> output_ch;

dfs(root);//устанавливаем суффиксные ссылки в боре методом обхода в глубину

if (output_ch == 0) {
	AhoCorasik(text, root, std::cout);
}
else if (output_ch == 1) {
	std::ofstream file;
	file.open("result.txt");
	if (!file.is_open()) {
		std::cout << "Incorrect!\n";
		return 0;
	}
	AhoCorasik(text, root, file);
}

return 0;
}
```