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

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Построение и анализ алгоритмов»

Тема: Алгоритм Ахо-Корасик

Студент гр. 8304 Самакаев Д.И.

Преподаватель Размочаева Н.В.

Санкт-Петербург

2019

Вариант 2.

Цель работы.

Построение и анализ алгоритма Ахо-Корасик на основе решения задачи о поиске вхождений подстроки в строку.

Основные теоретические положения.

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 xabvccbababcax.

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}

Описание алгоритма.

Для нахождения всех вхождений образца с джокером, на место джокера подставляются все символы алфавита и добавляются в структуру для хранения слов, которые необходимо найти в тексте. По этим словам строится бор и реализуется алгоритм Ахо-Корасик. Строятся суффиксы и сжатые суффиксы. При проходе по тексту и по бору, если встречается сжатый суффикс, переходим по нему, сохраняя позицию в тексте, если можем идти дальше по бору, идём, если не можем, переходим по суффиксной ссылке. Алгоритм прекращает работу когда завершает проход

по тексту.

Функции и структуры данных.

void bor_search(std::string text, std::shared_ptr<Elem> bor, std::vector<std::string> &result) – основная функция поиска в тексте.

void compressed_found(size_t i, std::shared_ptr<Elem> tmp, std::vector<std::string> &result) – рекурсивная функция перехода по сжатым суффиксам.

void make_word_variants(std::string word, std::vector<char>& alphabet, std::vector<std::string>& words) – функция обработки слов с «джокерами».

std::shared_ptr<Elem> make_bor(std::vector<std::string>words) – функция построения бора.

struct Elem – структура элемента бора.

Реализован файловый и консольный вводы и выводы.

Вывод промежуточной информации.

Во время основной части работы алгоритма происходит вывод обхода бора.

Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
aaabbbbcbd aa ab b?	0 1 1 1 2 2 3 4 4 4 5 4 6 5 8 6
aaaaa aa	0 1 1 1 2 1 3 1

Вывод.

В ходе работы был построен и анализирован алгоритм Ахо_Корасик на основе решения задачи о поиске подстроки в строке.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

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


struct Elem {
	std::map<char, std::shared_ptr<Elem>> next;
	std::string path;
	std::shared_ptr<Elem> suffix = nullptr;
	std::shared_ptr<Elem> compressed_suffix = nullptr;
	size_t height = 0;
	char name;
	size_t is_terminal = 0;
};

bool contain(char a, std::map<char, std::shared_ptr<Elem>> next){
	for (auto it = next.begin(); it != next.end(); it++) {
		if (it->first == a)
			return true;
	}
	return false;
}

std::shared_ptr<Elem> get_ptr(std::string path, std::shared_ptr<Elem> root) {
	bool is_path;
	for (size_t i = 0; i < path.size(); i++) {
		is_path = false;
		for (auto it = root->next.begin(); it != root->next.end(); it++) {
			if (it->first == path[i]) {
				root = it->second;
				is_path = true;
				break;
			}
		}
		if (is_path == false)
			return nullptr;
	}
	return root;
}

std::shared_ptr<Elem> make_bor(std::vector<std::string>words) {

	std::shared_ptr<Elem> bor = std::make_shared<Elem>();
	std::shared_ptr<Elem> tmp = bor;

	std::string path;

	for (size_t j = 0; j < words.size(); j++) {
		for (size_t i = 0; i < words[j].size(); i++) {
			path += words[j][i];

			if (!contain(words[j][i], tmp->next)) {
				tmp->next[words[j][i]] = std::make_shared<Elem>();
				tmp->next[words[j][i]]->name = words[j][i];
				tmp->next[words[j][i]]->height = i + 1;
			}
			tmp->height = i;
```

```
                tmp = tmp->next[words[j][i]];
                tmp->path = path;
            }
            path = "";
            tmp->is_terminal = j + 1;
            tmp = bor;
        }

        std::shared_ptr<Elem> buff = bor;

        std::queue<std::shared_ptr<Elem>> q;
        q.push(bor);
        tmp = bor;

        while (!q.empty()) {
            tmp = q.front();
            q.pop();

            for (auto it = tmp->next.begin(); it != tmp->next.end(); it++) {
                q.push(it->second);
            }
            path = tmp->path;
            while (true) {
                path.erase(0,1);
                buff = get_ptr(path, bor);
                if (buff != nullptr) {
                    tmp->suffix = buff;
                    if (tmp->suffix->is_terminal)
                        tmp->compressed_suffix = buff;
                    break;
                }
            }
        }

        return bor;
}

void make_word_variants(std::string word, std::vector<char>& alphabet,
std::vector<std::string>& words) {
        for (size_t i = 0; i < word.size(); i++) {
            if (word[i] == '?') {
                for (size_t j = 0; j < alphabet.size(); j++) {
                    word[i] = alphabet[j];
                    make_word_variants(word, alphabet, words);
                }
                return;
            }
            if (i == word.size() - 1)
                words.push_back(word);
        }
        return;
}


void compressed_found(size_t i, std::shared_ptr<Elem> tmp, std::vector<std::string>
&result) {
        if (tmp->compressed_suffix) {
            compressed_found(i, tmp->compressed_suffix, result);
        }
        std::string buff;
        buff += std::to_string(i - tmp->height);
        buff.push_back(' ');
        buff += std::to_string(tmp->is_terminal);
        result.push_back(buff);
}
```

```
void bor_search(std::string text, std::shared_ptr<Elem> bor, std::vector<std::string>
&result) {

	std::shared_ptr<Elem> tmp = bor;
	size_t i = 0;
	std::string buff;

	while (i != text.length()) {

		buff = "";

		if (tmp->compressed_suffix) {
			compressed_found(i, tmp->compressed_suffix, result);
		}

		if (contain(text[i], tmp->next)) {
			tmp = tmp->next[text[i]];
			if (tmp->is_terminal) {
				buff += std::to_string(i - tmp->height + 1);
				buff.push_back(' ');
				buff += std::to_string(tmp->is_terminal);
				result.push_back(buff);
			}
			i++;
		}
		else tmp = tmp->suffix;
	}
}

void console_input() {

	std::cout << "Please, Enter the text\n";

	std::string text;
	size_t words_number;
	std::vector<std::string>words;
	std::string buff;

	std::string out_file_name = "out.txt";

	std::cin >> text;

	std::cout << "Please, Enter words to search number" << std::endl;
	std::cin >> words_number;

	std::cout << "Please, Enter "<<words_number<<" word(s)\n";
	std::vector<char> alphabet = { 'a','b','c','d' };

	for (size_t i = 0; i < words_number; i++) {

		std::cin >> buff;
		make_word_variants(buff, alphabet, words);

	}

	std::vector<std::string> result;
	std::shared_ptr<Elem> bor;

	bor = make_bor(words);

	bor_search(text, bor, result);

	std::ofstream out_file;
	out_file.open(out_file_name);
```

```
	if (!out_file.is_open()) {
		std::cout << "Error! Output file isn't open" << std::endl;
	}

	for (size_t i = 0; i < result.size(); i++) {
		out_file << result[i] << std::endl;
	}

	for (size_t i = 0; i < result.size(); i++) {
		std::cout << result[i] << std::endl;
	}
}

void file_input(char*& argv) {
	std::ifstream file;
	std::string testfile = argv;
	file.open(testfile);

	std::string out_file_name = "out.txt";

	if (!file.is_open()) {
		std::cout << "Error! File isn't open" << std::endl;
		return;
	}

	std::vector<char> alphabet = { 'a','b','c','d' };

	std::string text;

	std::vector<std::string>words;

	file >> text;

	std::string buff;

	while (!file.eof()) {
		file >> buff;
		make_word_variants(buff, alphabet, words);
	}

	std::vector<std::string> result;
	std::shared_ptr<Elem> bor;

	bor = make_bor(words);

	bor_search(text, bor, result);

	for (size_t i = 0; i < result.size(); i++) {
		std::cout << result[i] << std::endl;
	}

	std::ofstream out_file;
	out_file.open(out_file_name);

	if (!out_file.is_open()) {
		std::cout << "Error! Output file isn't open" << std::endl;
	}

	for (size_t i = 0; i < result.size(); i++) {
		out_file << result[i] << std::endl;
	}
}

int main(size_t argc, char** argv)
```

```
{
	if (argc == 1)
		console_input();
	else if (argc == 2)
		file_input(argv[1]);
}
```