МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Отчет по выполнению практического задания 11

Тема:

Поиск в тесте образца. Алгоритмы. Эффективность алгоритмов.

Выполнил студент Цемкало А. Р.
Фамилия И. О.

группа ИКБО-10-20

Москва 2021

СОДЕРЖАНИЕ

1.1. Постановка задачи 3
1.2. Описание модели (подход к решению) 3
1.3. Код программы 3
1.4. Тестирование 5
1.5. Оценка практической сложности алгоритма в зависимости от длины текста и длины образца 6
2.1. Постановка задачи 6
2.2. Описание модели (подход к решению) 6
2.3. Код программы 7
2.4. Тестирование 8
2.5. Оценка практической сложности алгоритма в зависимости от длины текста 8
ВЫВОДЫ 9
СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 10

Цель. Получить знания и навыки применения алгоритмов поиска в тексте подстрок (образца).

Вариант 11.

Отчёт по задаче 1

1.1. Постановка задачи

Дан текст, состоящий из слов, разделенных знаками препинания. Сформировать массив из слов, которые содержат заданную подстроку.

1.2. Описание модели (подход к решению)

Цикл проходит по каждому символу. Если символ – знак препинания (т.е. конец слова), проверяем булеву переменную, отвечающую за признак найденного слова. При успехе добавляем слово в вектор. Затем очищается строка, в которую записывается текущее слово.

Если символ – буква, добавляем букву в текущее слово, если подстрока в этом слове ещё не найдена, проверяем: если буква равна первой букве в подстроке, отмечаем, что можно продолжать поиск, т.е. с последующими буквами будем сравнивать не первую букву подстроки, а вторую и т.д. Если же поиск был начат, но оказалось, что остальные буквы подстроки не встречаются в тексте, отмечаем, что поиск для этого слова ещё не начат.

1.3. Код программы

```
#include <iostream>
#include <chrono>
#include <vector>
using namespace std;

vector<string> search(string text, string substring) {
	long long int comparing = 1;
	vector<string> words;
	string word = "";
	int k = 0;
	bool found = false;
	bool full_found = false;
	for (int i = 0; i < text.size(); i++) {
		comparing++;
		if (text[i] > 31 && text[i] < 48 || text[i] == 58 || text[i] == 59) {
			comparing++;
			if (full_found) {
				words.push_back(word);
				cout << "Number of comparisons: " << comparing << endl;
			}
```

```
                        found = false;
                        full_found = false;
                        word = "";
                        k = 0;
                }
                else {
                        word += text[i];
                        comparing++;
                        if (!full_found) {
                                comparing++;
                                if (text[i] == substring[k]) {
                                        k++;
                                        comparing++;
                                        comparing++;
                                        if (k == substring.size()) {
                                                full_found = true;
                                                comparing--;
                                        }
                                        else if (!found) {
                                                found = true;
                                        }
                                }
                                else {
                                        comparing++;
                                        if (found) {
                                                found = false;
                                                k = 0;
                                        }
                                }
                        }
                }
        }
        comparing++;
        if (full_found) {
                words.push_back(word);
        }
        cout << "Number of comparisons: " << comparing << endl;
        return words;
}

int main() {
        //string text = "";
        //for (int i = 0; i < 50; i++) {
        //      if (i % 5 == rand() % 5) text += rand() % (47 - 32 + 1) + 32;
        //      else text += rand() % (122 - 97 + 1) + 97;
        //}

        string text = "J Strother Moore (his first name is the alphabetic character 'J' – not
an abbreviated 'J.') is a computer scientist. He is a co-developer of the Boyer–Moore
string-search algorithm, Boyer–Moore majority vote algorithm, and the Boyer–Moore automated
theorem prover, Nqthm. He made pioneering contributions to structure sharing including the
piece table data structure and early logic programming. An example of the workings of the
Boyer–Moore string search algorithm is given in Moore's website. Moore received his Bachelor
of Science (SB) in mathematics at Massachusetts Institute of Technology in 1970 and his
Doctor of Philosophy (Ph.D.) in computational logic at the University of Edinburgh in
Scotland in 1973.";
        //string text = "njk string inf dings";
        string substring = "ing";
        using clock_t = chrono::high_resolution_clock;
        using second_t = chrono::duration<double, ratio<1> >;
        chrono::time_point<clock_t> start;
        start = clock_t::now();
        vector<string> words = search(text, substring);
        cout << chrono::duration_cast<second_t>(clock_t::now() - start).count() << endl;
        if (words.empty()) {
```

```
            cout << "no matching words";
        }
        else {
            for (string word : words) cout << word << endl;
        }

        return 0;
}
```

1.4. Тестирование

Таблица 1

Искомая подстрока: “ing”.

№	Входные данные	Ожидаемый результат	Результат выполнения программы
1	J Strother Moore (his first name is the alphabetic character 'J' – not an abbreviated 'J.') is a computer scientist. He is a co-developer of the Boyer–Moore string-search algorithm, Boyer–Moore majority vote algorithm, and the Boyer–Moore automated theorem prover, Nqthm. He made pioneering contributions to structure sharing including the piece table data structure and early logic programming. An example of the workings of the Boyer–Moore string search algorithm is given in Moore's website. Moore received his Bachelor of Science (SB) in mathematics at Massachusetts Institute of Technology in 1970 and his Doctor of Philosophy (Ph.D.) in computational logic at the University of Edinburgh in Scotland in 1973.	string pioneering sharing including programming workings string	string pioneering sharing including programming workings string
2	njk string inf dings	string dings	string dings
3	njk strshs, shtlk. ag	no matching words	no matching words
4	ing:ing string thing dings	ing ing string thing dings	ing ing string thing dings
5	/&#!	no matching words	no matching words

1.5. Оценка практической сложности алгоритма в зависимости от длины текста и длины образца

Таблица 2

n	Время выполнения	Кол-во сравнений при успешном поиске	Кол-во сравнений при безуспешном поиске
10	0.0006845	33	38
100	0.0047426	333	369
1000	0.0008792	3600	3620
10000	0.0064204	36246	36320
100000	0.0598715	362183	363087
1000000	0.534276	3622172	3630769

Время выполнения и количество сравнений не зависит от длины искомой подстроки.

В программе используется один цикл, поэтому сложность алгоритма O(n). Тестирование это подтверждает.

Отчёт по задаче 2

2.1. Постановка задачи

Назовем строку палиндромом, если она одинаково читается слева направо и справа налево. Примеры палиндромов: "abcba", "55", "q", "xyzzyx". Требуется для заданной строки найти максимальную по длине ее подстроку, являющуюся палиндромом. Реализация алгоритмом Бойера и Мура.

2.2. Описание модели (подход к решению)

Поиск палиндрома осуществляется путём поиска обращённой строки в тексте. Если эта строка найдена и является соседней с исходной, палиндром найден, затем сравнивается его длина с предыдущим найденным.

Идея алгоритма Бойера – Мура: Сканирование слева направо, сравнение справа налево. Нахождение Стоп – символа: при несовпадении первой сравниваемой буквы, шаблон сдвигается до ближайшей такой же; при

отсутствии стоп - символа шаблон сдвигается за него. Поиск совпадающего суффикса: в случае совпадения 1 или более числа символов шаблон сдвигается вправо до первого совпадения с этим суффиксом.

2.3. Код программы

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> make_shift_table(string text, string substr) {
	int n = text.size();
	vector<int> shifts(substr.size() + 1, substr.size());
	vector<int> z(substr.size(), 0);
	for (int j = 1, maxZidx = 0, maxZ = 0; j < substr.size(); ++j) {
		if (j <= maxZ) z[j] = std::min(maxZ - j + 1, z[j - maxZidx]);
		while (j + z[j] < substr.size() && substr[substr.size() - 1 - z[j]] ==
substr[substr.size() - 1 - (j + z[j])]) z[j]++;
		if (j + z[j] - 1 > maxZ) {
			maxZidx = j;
			maxZ = j + z[j] - 1;
		}
	}
	for (int j = substr.size() - 1; j > 0; j--) shifts[substr.size() - z[j]] = j;
	for (int j = 1; j <= substr.size() - 1; j++)
		if (j + z[j] == substr.size())
			for (int r = 0; r <= j; r++)
				if (shifts[r] == substr.size()) shifts[r] = j;
	return shifts;
}

int search_for_substring(string text, string substr) {
	vector<int> shift_table = make_shift_table(text, substr);
	int n = text.size();
	for (int i = 0, j = 0; i <= n - substr.size() && j >= 0; i += shift_table[j + 1]) {
		for (j = substr.size() - 1; j >= 0 && substr[j] == text[i + j]; j--);
		if (j < 0)
			return i;
	}
}

int find_max_palindrome(string text, int& n) {
	int i, j, p, k, m = 0, pos = 0;
	for (i = 0; i < text.size(); ++i) {
		for (j = i + 1; j < text.size(); ++j) {
			string first = text.substr(i, j - i);
			if (first.size() > (text.size() - j) / 2) break;
			string second = first;
			reverse(second.begin(), second.end());
			p = search_for_substring(text.substr(j), second) + j;
			if (p < text.size()) {
				if ((p == j + 1) || (p == j)) {
					k = p + j - i - i - 1;
					if (k > m) {
						m = k;
						pos = i;
					}
				}
			}
		}
	}
```

```
	n = (m > 0) ? m + 1 : m;
	return pos;
}

int main() {
	string s = "a12325bc9e9c321";
	//string s = "123456789";
	int pos, len;
	pos = find_max_palindrome(s, len);
	int i, n = pos + len;
	if (len == 0) {
		cout << "no palindrome was found";
	}
	else {
		for (i = pos; i < n; ++i) cout << s[i];
	}

	return 0;
}
```

2.4. Тестирование

Таблица 3

№	Входные данные	Ожидаемый результат	Результат выполнения программы
1	a12325bc9e9c321	c9e9c	c9e9c
2	a123gg325bc9e9c321	23gg32	23gg32
3	123456789	no palindrome was found	no palindrome was found

2.5. Оценка практической сложности алгоритма в зависимости от длины текста

n	Время выполнения	Кол-во сравнений
10	0.0002401	324
100	0.0407338	215207
1000	14.9179	180437771
10000	223.65	151285921013

Вычислительная сложность алгоритма Бойера – Мура O(n+m), где n – длина строки, в которой выполняется поиск, m – длина шаблона поиска.

В алгоритме используются 2 вложенных цикла, в которых ещё вызывается функция поиска подстроки алгоритмом Бойера – Мура. Поэтому вычислительная сложность программы – $O(n^3)$.

ВЫВОДЫ

В ходе выполнения задания получены знания и практические навыки по разработке и реализации алгоритмов поиска в тексте подстрок (образца).

Разработан алгоритм поиска подстроки в тексте. Подсчитано количество сравнений для успешного и безуспешного поисков. Оценена практическая сложность алгоритма O(n) в зависимости от длины текста, сложность не зависит от длины образца.

Разработан алгоритм поиска максимальной по длине подстроки, являющейся палиндромом с помощью реализации алгоритма Бойера и Мура.

СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Процедурное программирование Языки программирования – Сайт lizochekk! [Электронный ресурс]: URL: https://lizochekk.jimdofree.com/программирование/
2. Документация по Microsoft C/C++ | Microsoft Docs – [Электронный ресурс] URL: https://docs.microsoft.com/ru-ru/cpp/?view=msvc-160
3. Все публикации подряд / Хабр – [Электронный ресурс] URL: https://habr.com/ru/
4. Курс: Структуры и алгоритмы обработки данных (Часть 1) Скворцова Л.А. [II.20-21] – [Электронный ресурс] URL: https://online-edu.mirea.ru/course/view.php?id=6600
5. ru.wikipedia.org/wiki/Алгоритм_Бойера_—_Мура#Описание_алгоритма