Лабораторная работа №1
студента группы ИТ – 32
Курбатовой Софьи Андреевны

Выполнение:_____ Защита _____

РЕАЛИЗАЦИЯ АЛГОРИТМОВ ПОИСКА ПО ТЕКСТУ.

Цель работы: научиться реализовывать на выбранном языке программирования алгоритмы поиска по тексту: прямой поиск; алгоритм Кнута, Морриса и Пратта; алгоритм Бойера-Мура.

Содержание работы

Написать программу на выбранном языке программирования, реализующую указанные выше алгоритмы для поиска подстроки в строке. Программа должна запрашивать имя входного файла. Оценить трудоемкость рассматриваемых алгоритмов.

Ход работы

1. Идея состоит в следующем: перебрать текст и, если есть совпадение для первой буквы шаблона, проверит, все ли буквы шаблона соответствуют тексту. Если m - это количество букв в шаблоне, а n - это количество букв в тексте, временная сложность этих алгоритмов равна O (m (n-m 1)) .

Листинг 1.1. Функция реализующая алгоритм простого поиска.

```
#include <iostream>
#include <string>
#include <cstring>
#include <iostream>
#include <fstream>
#include <stdio.h>
#define _CRT_SECURE_NO_WARNINGS
char* userStrstr(const char* haystack,  char* needle)
{
	for (const char* hp = haystack; hp != haystack + strlen(haystack); ++hp)
	{
		const char* np = needle;
		const char* tmp = hp;
		for (; np != needle + strlen(needle); ++np)
		{
			if (*tmp != *np)
			{
				break;
			}
			else
			{
				++tmp;
			}
		}
		if (np == needle + strlen(needle))
		{
	std::cout <<"Прямой поиск: Найдено на " << pos+1<<"позиции от начала строки\n";
			return needle;
		}
	}
	return 0;
}
```

Преимущество этого алгоритма в том, что ценой некоторого количества предварительных вычислений над шаблоном (но не над строкой, в которой ведётся поиск), шаблон сравнивается с исходным текстом не во всех позициях — часть проверок пропускаются как заведомо не дающие результата.

Общая оценка вычислительной сложности современного варианта алгоритма Бойера — Мура —O(n+m) , где n — длина строки, в которой выполняется поиск, m — длина шаблона поиска.

Листинг 1.2. Функция реализующая алгоритм Бойера-Мура.

```
int BMSearch(char* str, char* substr)
{
	int sl, ssl;
	int res = -1;
	sl = strlen(str);
	ssl = strlen(substr);
	int i, Pos;
	int BMT[256];
	for (i = 0; i < 256; i++)
		BMT[i] = ssl;
	for (i = ssl - 1; i >= 0; i--)
		if (BMT[((short)substr[i])] == ssl)
			BMT[(short)(substr[i])] = ssl - i - 1;
	Pos = ssl - 1;
	while (Pos < sl)
		if (substr[ssl - 1] != str[Pos])
			Pos = Pos + BMT[((short)str[Pos])];
		else
			for (i = ssl - 2; i >= 0; i--) {
				if (substr[i] != str[Pos - ssl + i + 1]) {
					Pos += BMT[(short)(str[Pos - ssl + i + 1])] - 1;
std::cout << "Алгоритм Бойера-Мура: Найдено на " << Pos << " позиции от начала строки\n";

break;
				}
				else
					if (i == 0)
						return Pos - ssl + 1;
			}
	return res;
}
```

Идея алгоритма Кнута-Морриса-Пратта заключается в расчете таблицы сдвигов, которая предоставляет нам информацию, в которой мы должны искать кандидатов-паттернов.

Листинг 1.3. Функция реализующая алгоритм Кнута, Морриса, Пратта.

```
int KMPSearch(char* str, char* substr)
{
	int sl, ssl;
	int res = -1;
	sl = strlen(str);
	ssl = strlen(substr);
	int i, j = 0, k = -1;
	int* d;
	d = new int[1000];
	d[0] = -1;
	while (j < ssl - 1) {
		while (k >= 0 && substr[j] != substr[k])
			k = d[k];
		j++;
		k++;
		if (substr[j] == substr[k])
			d[j] = d[k];
		else
```

```
                        d[j] = k;
        }
        i = 0;
        j = 0;
        while (j < ssl && i < sl) {
                while (j >= 0 && str[i] != substr[j])
                        j = d[j];
                i++;
                j++;
        }
        delete[] d;
        res = j == ssl ? i - ssl : -1;
std::cout << "Алгоритм Кнута, Морриса и Пратта: Найдено на "<<i-ssl+1<<" позиции от начала
строки\n";
        return res;
}
```

Листинг 1.4. Главная функция

```
int main()
{
        setlocale(LC_ALL, "");
        //char path = "C:\\test\\tests.txt";
        char path[100];
        char line[100];
        char search[100];
        std::cout << "Enter path:\t";
        std::cin >> path;
        std::cout << "Enter search word:\t";
        std::cin >> search;
        std::ifstream input(path);

        while (!input.is_open())
        {
                std::cerr << "File error" << std::endl;
                std::cin.ignore();
                break;
        }
        if (input)
        {
                while (!input.eof())
                {
                        input.getline(line, 80);
                        std::cout << "Символов в строке-источнике: " << strlen(line) << ". Символов в
искомой строке: " << strlen(search) << std::endl;
                        if (userStrstr(line, search))
                        {
                                std::cout << "Прямой поиск:" << std::endl;
                                std::cout << "Сложность = "<< ((strlen(line) -
strlen(search)+1)*strlen(search)) <<std::endl;
                                std::cout <<"Your word: " <<search << " in "<<"' "<<line<<"
'"<<std::endl;
                        }

                        if (KMPSearch(line, search)!= -1)
                        {
                                std::cout << "Алгоритм Кнута, Морриса и Пратта:" << std::endl;
                                std::cout << "Сложность = " << (strlen(line) + strlen(search))
<<std::endl;
                                std::cout << "Your word: " << search << " in " << "' " << line << " '"
<< std::endl;
                        }
                        if (BMSearch(line, search)!= -1)
                        {
                                std::cout << "Алгоритм Бойера-Мура:" << std::endl;
                                std::cout << "Сложность = " << (strlen(line) + strlen(search)) <<
std::endl;
```

```
                        std::cout << "Your word: " << search << " in " << "' " << line << " '"
<< std::endl;
                    }
                }
            }
        input.close();
         }
```

2. Тестирование:

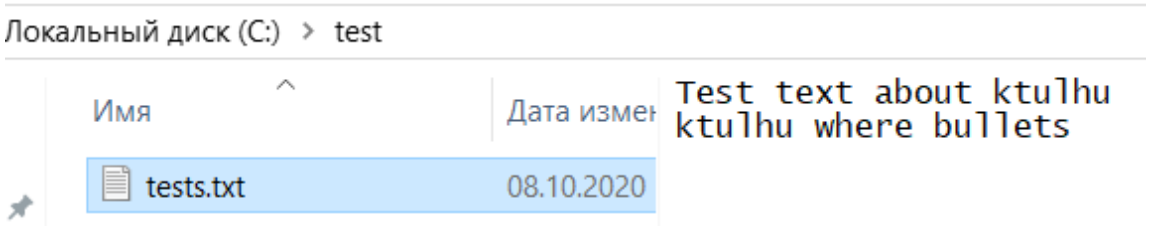

Рис. 1.1. Исходный текстовый файл

```
Enter path:     C:\\test\\tests.txt
Enter search word:      where
Символов в строке-источнике: 22. Символов в искомой строке: 5
Символов в строке-источнике: 20. Символов в искомой строке: 5
Прямой поиск:
Сложность = 80
Your word: where in ' ktulhu where bullets '
Алгоритм Кнута, Морриса и Пратта:
Сложность = 25
Your word: where in ' ktulhu where bullets '
Алгоритм Бойера-Мура:
Сложность = 25
Your word: where in ' ktulhu where bullets '
```

Рис. 1.2. Результат работы программы

```
Выбрать Консоль отладки Microsoft Visual Studio
Enter path:     C:\test\tests.txt
Enter search word:      text
Символов в строке-источнике: 22. Символов в искомой строке: 4

 Прямой поиск: Найдено на 6позиции от начала строки
Сложность = 76
Your word: text in ' Test text about ktulhu '

 Алгоритм Кнута, Морриса и Пратта: Найдено на 6 позиции от начала строки
Сложность = 26
Your word: text in ' Test text about ktulhu '

 Алгоритм Бойера-Мура: Найдено на 6 позиции от начала строки
Сложность = 26
Your word: text in ' Test text about ktulhu '
Символов в строке-источнике: 20. Символов в искомой строке: 4
```

Рис. 1.3. Результат работы программы с указанием позиции

Вывод: Таким образом, в ходе выполнения лабораторной работы на языке C/C++ были реализованы алгоритмы поиска строки, задаваемой пользователем, в текстовом файле, путь к которому также задается пользователем. Была проведена оценка сложности используемых алгоритмов. В результате было выявлено, что наиболее эффективными можно считать алгоритмы Кнута и Боейра-Мура, так как их сложность O(m+n). С другой стороны, наиболее понятен и быстро реализуем, так как

не требует дополнительной обработки текста, алгоритм простого поиска. Однако он сложнее: O((n-m+1)*m). Потому что, искомая строка сравнивается посимвольно.