**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №4**

**по дисциплине «Программирование»**

Тема: "**Поиск подстроки в строке. Работа с текстовыми строками**"

|  |  |  |
| --- | --- | --- |
| Студент(ка) гр. | Тимошенков И.М. |  |
| Преподаватель | Глущенко А. Г. |  |

Санкт-Петербург

2022

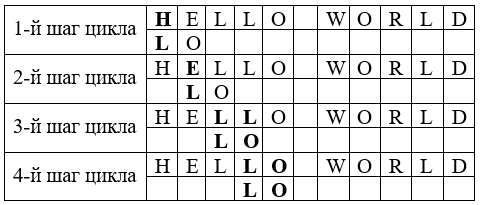
**Цель работы.**

Изучение работы со строками, редактирование строк и поиск подстроки в строке при помощи алгоритма Бойера-Мура и линейного алгоритма.

**Основные теоретические положения.**

При работе со строками часто будет возникать потребность в поиске набора символа или слов (поиска подстроки в строке). При условии, что текст может быть крайне большим, хочется, чтобы алгоритм поиска подстроки работал быстро.

Самый простой способ подстроки в строке – Линейный поиск – циклическое сравнение всех символов строки с подстрокой. Действительно, этот способ первый приходит в голову, но очевидно, что он будет самым долгим.



На первых двух итерациях цикла сравниваемые буквы не будут совпадать. На третьей же итерации, совпал символ ‘L’, это означает, что теперь нужно сравнивать следующий символ подстроки со следующим символом строки. Видно, что символы отличаются, поэтому алгоритм продолжает свою работу. На четвертой же итерации подстрока была найдена.

Если представить, что исходная строка непорядок больше и подстрока находится в конце строки (или вовсе отсутствует), то сразу видны минусы данного алгоритма.

Одним из самых популярных алгоритмов, который работает быстрее, чем приведенный выше алгоритм, является алгоритм Кнута-Морриса-Пратта (КМП). Идея заключается в том, что не нужно проходить и сравнивать абсолютно все символы строки, если известны символы, которые есть и в строке, и в подстроке.

Суть алгоритма: дана подстрока S и строка T. Требуется определить индекс, начиная с которого образец S содержится в строке T. Если S не содержится в T, необходимо вернуть индекс, который не может быть интерпретирован как позиция в строке.



Хоть алгоритм и работает быстрее, по-прежнему необходимо сначала пройти всю строку, чтобы определить префиксы или суффиксы (вхождение (индексы) символов).

Алгоритм Бойера-Мура в отличие от КМП полностью не зависим и не требует заранее проходить по строке. Этот алгоритм считается наиболее быстрым среди алгоритмов общего назначения, предназначенных для поиска подстроки в строке.

Преимущество этого алгоритма в том, что ценной некоторого количества предварительных вычислений над подстрокой (но не над исходной строкой, в которой ведётся поиск), подстрока сравнивается с исходным текстом не во всех позициях (пропускаются позиции, которые точно не дадут положительный результат).

Поиск подстроки ускоряется благодаря созданию таблиц сдвигов. Сравнение подстроки со строки начинается с последнего символа подстроки, а затем происходит прыжок, длина которого определяется по таблице сдвигов. Таблица сдвигов строится по подстроке так чтобы перепрыгнуть максимальное количество символов строки и не пропустить вхождение подстроки в строку.

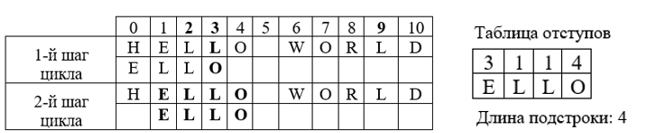
Правила построения таблицы сдвигов:

1)    Значение элемента таблицы равно удаленности соответствующего символа от конца шаблона (подстроки).

2)    Если символ встречается более одного раза, то применятся значение, соответствующее символу, наиболее близкому к концу шаблона.

3)    Если символ в конце шаблона встречается 1 раз, ему соответствует значение, равное длине образа; если более одного раза – значение, соответствующее символу, наиболее близкому к концу образа.

4)    Для символов, отсутствующих в образе, применяется значение, равное длине шаблона.



Сначала была построена таблица отступов и подсчитана длина подстроки. Затем начинается алгоритм поиска подстроки в строке. Сравнивает символ ‘L’ строки и ‘O’ подстроки. Элементы не совпадают, поэтому необходимо определить длину отступа. Символ ‘L’ присутствует в таблице отступа, длина отступа равняется 1. Подстрока смещается на 1 символ вперед. На следующей итерации подстрока найдена.

**Постановка задачи.**

Необходимо написать программу, которая:

1)    С клавиатуры или с файла (\*) (пользователь сам может выбрать способ ввода) вводится последовательность, содержащая от 1 до 50 слов, в каждом из которых от 1 до 10 строчных латинских букв и цифр. Между соседними словами произвольное количество пробелов. За последним символом стоит точка.

2)    Необходимо отредактировать входной текст:

·        удалить лишние пробелы;

·        удалить лишние знаки препинания (под «лишними» подразумевается несколько подряд идущих знаков (обратите внимание, что «…» - корректное использование знака) в тексте);

·        исправить регистр букв, если это требуется (пример некорректного использования регистра букв: пРиМЕр);

3) Выполнить задание по варианту:

4) Выполнить задание по варианту:

5)  Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КМП, а нечетные – Бойера-Мура. (\*)

**Выполнение работы.**

Код программы представлен в приложении А.

#include <iostream>

#include <string>

#include <fstream>

#include <windows.h>

using namespace std;

//возвращает настоящую длину массива, нужно для оптимизации массива

int realLength(string\* arr, int size) {

int length = 0;

for (int i = 0; i < size; i++) {

if (arr[i] != "") {

length++;

}

else {

return length;

}

}

}

//разделяет строку в массив по знаку

string\* split(string str, char delim, int& size) {

size = 1000;

str += delim;

string\* array = new string[size];

string word = "";

int k = 0;

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

if (str[i] != delim) {

word += str[i];

}

else {

array[k] = word;

k++;

word = "";

}

}

size = realLength(array, size);

string\* newArr = new string[size];

for (int i = 0; i < size; i++) {

newArr[i] = array[i];

}

delete[] array;

return newArr;

}

//объединяет массив строк в строку чреез знак

string join(string\* str, string del, int& size) {

string stroka = "";

for (int i = 0; i < size; i++) {

stroka += str[i] + del;

}

size = stroka.length();

return stroka;

}

//удаляет вообще все знаки, нужно для заданий, где выводятся только слова

void deleteAllSignesAndDots(string& str) {

while (str.find(",") != -1) {

str.replace(str.find(","), 1, " ");

}

while (str.find("?") != -1) {

str.replace(str.find("?"), 1, " ");

}

while (str.find("!") != -1) {

str.replace(str.find("!"), 1, " ");

}

while (str.find("-") != -1) {

str.replace(str.find("-"), 1, " ");

}

while (str.find(".") != -1) {

str.replace(str.find("."), 1, " ");

}

while (str.find(")") != -1) {

str.replace(str.find(")"), 1, " ");

}

while (str.find("(") != -1) {

str.replace(str.find("("), 1, " ");

}

while (str.find(" ") != -1) {

str.replace(str.find(" "), 2, " ");

}

}

//удаляет лишние знаки

void deleteSignes(string& str, int& size) {

while (str.find(" ") != -1) {

str.replace(str.find(" "), 2, " ");

}

while (str.find(",,") != -1) {

str.replace(str.find(",,"), 2, ",");

}

while (str.find("??") != -1) {

str.replace(str.find("??"), 2, "?");

}

while (str.find("!!") != -1) {

str.replace(str.find("!!"), 2, "!");

}

while (str.find("--") != -1) {

str.replace(str.find("--"), 2, "-");

}

while (str.find("((") != -1) {

str.replace(str.find("(("), 2, "(");

}

while (str.find("))") != -1) {

str.replace(str.find("))"), 2, ")");

}

size = str.length();

}

//приводит слово к нижнему регистру

void redactorOfRegister(string& str) {

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

if (str[i] <= 'Z' && str[i] >= 'A')

str[i] -= ('Z' - 'z');

else if (str[i] <= 'Я' && str[i] >= 'А')

str[i] -= ('Я' - 'я');

}

}

//делает из символа заглавную букву

void mainLetter(char& a) {

if (a <= 'z' && a >= 'a')

a -= ('z' - 'Z');

else if (a <= 'я' && a >= 'а')

a -= ('я' - 'Я');

}

//удаляет все лишние точки, кроме знака моноготочия

void deleteDots(string& str, int& size) {

while (str.find("...") != -1) {

if (str[str.find("...") + 3] != '.' and str[str.find("...") - 1] != '.' and str[str.find("...") - 1] != '<' and str[str.find("...") + 3] != '<') {

str.replace(str.find("..."), 3, "\*");

}

else {

str.replace(str.find("..."), 3, "<");

}

}

while (str.find("<") != -1) {

str.replace(str.find("<"), 1, "...");

}

while (str.find("..") != -1) {

str.replace(str.find(".."), 2, ".");

}

while (str.find("\*") != -1) {

str.replace(str.find("\*"), 1, "...");

}

size = str.length();

}

//редактирует строку в нормальный вид

string redactorOfSentances(string& str, int& size) {

deleteSignes(str, size);

deleteDots(str, size);

string\* b = split(str, ' ', size);

for (int i = 0; i < size; i++) {

redactorOfRegister(b[i]);

}

string newStr = join(b, " ", size);

b = split(newStr, '.', size);

for (int i = 0; i < size; i++) {

if (b[i][0] != ' ') {

mainLetter(b[i][0]);

}

else {

mainLetter(b[i][1]);

}

}

newStr = join(b, ".", size);

newStr.replace(newStr.length() - 1, 1, "");

return newStr;

delete[] b;

}

//проверяет является ли символ числом

bool isDigit(string str) {

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

if (str[i] <= '9' && str[i] >= '0') {

return false;

}

}

}

//функция для первого задания

void problem1(string str, int size) {

deleteAllSignesAndDots(str);

string\* b = split(str, ' ', size);

for (int i = 0; i < size; i++) {

if (isDigit(b[i])) {

redactorOfRegister(b[i]);

cout << b[i] << " ";

}

}

}

//функция для второго задания

void problem2(string str, int size) {

deleteAllSignesAndDots(str);

string\* b = split(str, ' ', size);

for (int i = 0; i < size; i++) {

redactorOfRegister(b[i]);

cout << b[i] << '\n';

}

}

int getNumberOfSize(int\* arr, string str, char a, int size) { // возвращает число сдвига согласно букве из таблицы сдвигов

for (int i = 0; i < size -1; i++) {

if (str[size - arr[i] - 1] == a) {

return arr[i];

}

}

return arr[size - 1]; // возвращает номер сдвига для любого элемента или для последнего, если он не встречается нигде кроме как в конце

}

// функция, которая проверяет есть ли проверяемый элемент в массиве уже проверенных элементов

int isElement(string str, int\* a, int j, int size) {

for (int i = 0; i < size; i++) {

if (a[i] < str.length() and a[i] >= 0) {

if (str[a[i]] == str[j]) {

return a[i];

}

};

}

return -1;

}

void findInText(string str, string text) { //поиск по алгоритму Бойера-Мура

int size = str.length();

int\* arrayOfIndexes = new int[size]; //массив, который сохраняет индексы уже обработанных символов в таблице

int\* arrayOfSizes = new int[size]; //сама таблица

// цикл формирования таблицы

for (int i = size - 2; i >= 0; i--) {

int prevIndex = isElement(str, arrayOfIndexes, i, size);

if (prevIndex != -1) {

arrayOfSizes[i] = size - (prevIndex + 1);

}

else {

arrayOfSizes[i] = size - (i + 1);

arrayOfIndexes[i] = i;

}

}

//рассчитывание отступа для последнего элемента

int prevIndexOfLast = isElement(str, arrayOfIndexes, size - 1, size);

if (prevIndexOfLast != -1) {

arrayOfSizes[size - 1] = size - (prevIndexOfLast + 1);

}

else {

arrayOfSizes[size - 1] = size;

}

int \*indexes = new int[100];

int point = 0;

int counterForIndexes = 0;

bool isFinded = false;

while ((point+ str.length()) <= text.length()) {

int k = str.length() - 1;

int numberOfMatches = 0;

for (int pointerRev = point + str.length() - 1; point <= pointerRev; pointerRev--) {

if (text[pointerRev] == str[k]) {

numberOfMatches += 1;

k--;

if (numberOfMatches == str.length()) {

indexes[counterForIndexes] = point;

counterForIndexes += 1;

point += str.length();

}

}

else {

if (str.find(text[pointerRev]) != -1) {

point += getNumberOfSize(arrayOfSizes, str, text[pointerRev], str.length());

break;

}

else {

point += str.length();

break;

}

}

}

}

for (int i = 0; i < counterForIndexes; i++) {

for (int j = 0; j < str.length(); j++) {

mainLetter(text[indexes[i] + j]);

}

}

delete[] indexes;

cout << text;

}

void lineSearch(string str, string text) { // линейный поиск

int\* indexes = new int[100];

int e = 0;

for (int i = 0; i < text.length() - str.length() + 1; i++) {

int k = 0;

for (int j = 0; j < str.length(); j++) {

if (str[j] == text[i + j]) {

k += 1;

}

}

if (k == str.length()) {

indexes[e++] = i;

}

}

for (int i = 0; i < e; i++) {

for (int j = 0; j < str.length(); j++) {

mainLetter(text[indexes[i] + j]);

}

}

cout << text;

delete[] indexes;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "");

string text;

string flag;

cout << "Как хотите прочесть строку (из файла - введите 1)/(из консоли - введите 2)\n";

getline(cin, flag);

if (flag == "1") {

ifstream fin;

string path;

cout << "Введите имя файла и расширение: ";

getline(cin, path);

fin.open(path);

if (!fin.is\_open()) {

cout << "Ошибка открытия" << '\n';

}

else {

text = "";

string line;

while (getline(fin, line)) {

text += line + " ";

}

}

fin.close();

}

else if (flag == "2") {

cout << "Введите строку: ";

getline(cin, text);

}

else {

cout << '\n';

cout << "Произошла ошибка, перезагрузите программу";

return 1;

}

cout << "------------------------------------|Задание-1|------------------------------------";

cout << '\n';

problem1(text, text.length());

cout << '\n';

cout << "------------------------------------|Задание-2|------------------------------------";

cout << '\n';

problem2(text, text.length());

cout << '\n';

int size = text.length();

text = redactorOfSentances(text, size);

cout << "Введите строку которую хотите найти: ";

string str;

getline(cin, str); cout << '\n';

cout << "---Работа линейного поиска---";

cout << '\n';

lineSearch(str, text); cout << '\n' << '\n';

cout << "---Работа алгоритма Бойера-Мура---";

cout << '\n';

findInText(str, text);

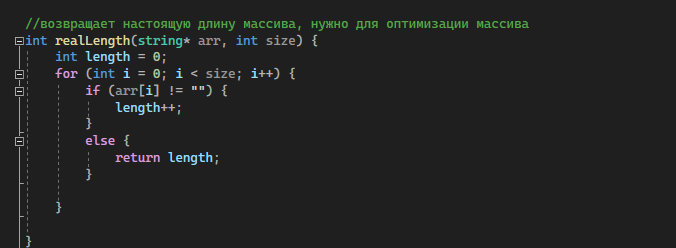
cout << '\n';

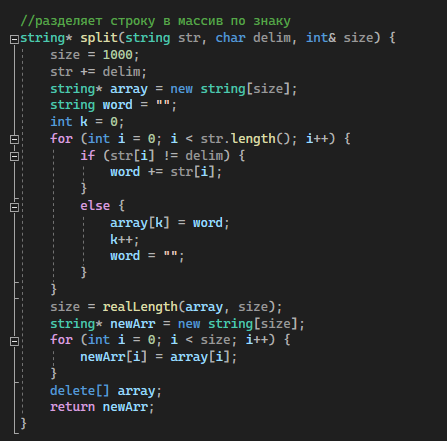
return 0;

}

**Блок описания кода и использованных алгоритмов**

Алгоритм оптимизации массива строк и split(). Удаляет лишние элементы

****



Объеденение массива в строку.

