**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 гр. 4373 |  | Музыченко И.А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

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:

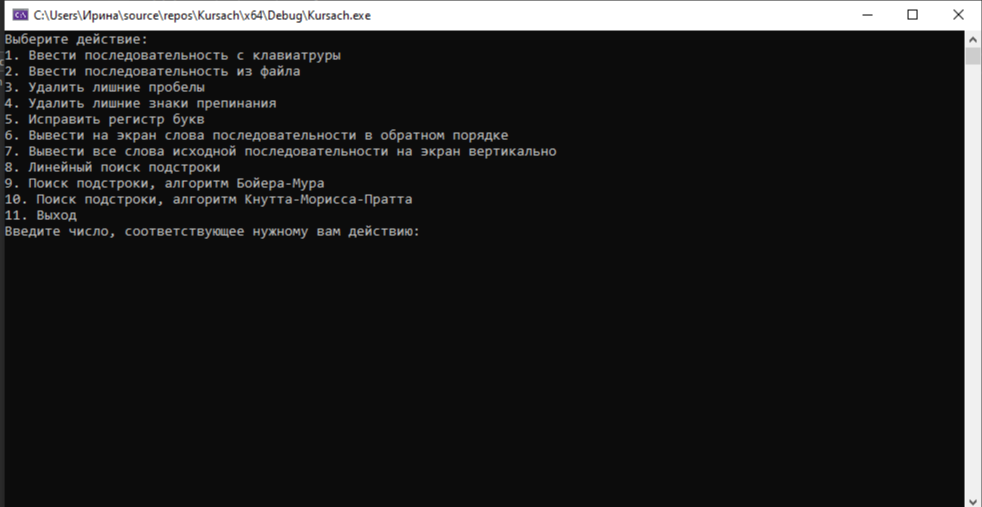
5) Необходимо найти все подстроки, которую введёт пользователь в имеющейся строке. Реализуйте два алгоритма: первый алгоритма – Линейный поиск, а второй алгоритм согласно вашему номеру в списке. Четные номера должны реализовать алгоритм КМП, а нечетные – Бойера-Мура. (\*)

**Выполнение работы.**

**Код программы представлен в приложении А**.

**Блок скриншотов работы программы**

Выбор задания



**Выводы.**

Выполнив данную работу мы поняли, как работать со строками.

Приложение А

рабочий код

#include <iostream>

#include <windows.h>

#include <fstream>

#include <stdio.h>

#include "Module1.4Func.h"

#define numberOfUniqueSymbols 256

using namespace std;

int checkInputString(int floor, int ceiling) {

int num;

cin >> num;

while (cin.fail() || cin.peek() != '\n' || num > ceiling || num < floor) {

cin.clear();

cin.ignore((numeric\_limits<streamsize>::max)(), '\n');

cout << "Введите корректное значение: ";

cin >> num;

}

return num;

}

unsigned int my\_StrLen(char\* String) {

unsigned int Length = 0;

while (String[Length])

++Length;

return Length;

}

void del(char\* arr, int length, int index) {

for (int i = index; i < length; i++)

{

arr[i] = arr[i + 1];

}

arr[length - 1] = '\0';

}

void deleteExtraSpaces(char\* txt) {

int length = my\_StrLen(txt);

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

if (txt[i] == ' ' && txt[i + 1] == ' ')

while (txt[i + 1] == ' ')

del(txt, length, (i + 1));

}

}

void deleteExtraPunctuation(char\* txt) {

int length = my\_StrLen(txt);

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

if (ispunct(txt[i]) && ispunct(txt[i + 1]))

if (txt[i] == '.' && txt[i + 1] == '.' && txt[i + 2] == '.' && txt[i + 3] == ' ') {

i += 3;

}

else {

while (ispunct(txt[i + 1]))

del(txt, length, (i + 1));

}

}

}

void correctLetterCase(char\* txt) {

for (int i = 0; i < my\_StrLen(txt); i++) {

if (isalpha(txt[i]) && (i == 0 || txt[i - 1] == ' ')) {

txt[i] = toupper(txt[i]);

}

else {

txt[i] = tolower(txt[i]);

}

}

}

void mainMenuString() {

system("cls");

cout << "Выберите действие: " << endl;

cout << "1. Ввести последовательность с клавиатруры" << endl;

cout << "2. Ввести последовательность из файла" << endl;

cout << "3. Удалить лишние пробелы" << endl;

cout << "4. Удалить лишние знаки препинания" << endl;

cout << "5. Исправить регистр букв" << endl;

cout << "6. Вывести на экран слова последовательности в обратном порядке" << endl;

cout << "7. Вывести все слова исходной последовательности на экран вертикально" << endl;

cout << "8. Линейный поиск подстроки" << endl;

cout << "9. Поиск подстроки, алгоритм Бойера-Мура" << endl;

cout << "10. Поиск подстроки, алгоритм Кнутта-Морисса-Пратта" << endl;

cout << "11. Выход" << endl;

cout << "Введите число, соответствующее нужному вам действию: ";

}

int linearSearch(char\* txt, char word[], int textLength, int wordLength) {

for (int i = 0; i <= textLength - wordLength; ++i) {

int j;

for (j = 0; j <= wordLength; ++j)

if (txt[i + j] != word[j])

break;

if (j == wordLength)

return i;

}

return -1;

}

int BoyerMoore(char\* txt, char\* subsequence, int lenTxt, int lenSub) {

int table[numberOfUniqueSymbols];

int step = 0;

for (int i = 0; i < numberOfUniqueSymbols; i++)

table[i] = -1;

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

table[subsequence[i]] = i;

while (step <= (lenTxt - lenSub))

{

int j = lenSub - 1;

while (j >= 0 && subsequence[j] == txt[step + j])

j--;

if (j < 0)

{

return step;

step += (step + lenSub < lenTxt) ? lenSub - subsequence[txt[step + lenSub]] : -1;

}

else

{

step += max(1, j - table[txt[step + j]]);

}

}

}

int print(int i) {

cout << i << " ";

return 1;

}

void prefix(char\* pattern, int\* patternIndex, int length) {

patternIndex[0] = 0;

for (int i = 1; i < length; ++i)

{

int j = patternIndex[i - 1];

while ((j > 0) && (pattern[i] != pattern[j]))

j = patternIndex[j - 1];

if (pattern[i] == pattern[j])

++j;

patternIndex[i] = j;

}

}

void KMP(char\* txt, char\* pattern, int\* patternIndex, int length, int f(int)) {

patternIndex[0] = 0;

int l = 0;

for (l = 1; l < length; ++l)

{

int j = patternIndex[l - 1];

while ((j > 0) && (pattern[l] != pattern[j]))

j = patternIndex[j - 1];

if (pattern[l] == pattern[j])

++j;

patternIndex[l] = j;

}

int j = 0;

for (int i = 0; txt[i]; ++i)

{

while ((j > 0) && (txt[i] != pattern[j]))

j = patternIndex[j - 1];

if (txt[i] == pattern[j])

++j;

if (j == l)

if (!print(i - l + 1))

return;

}

}

#define numberOfUniqueSymbols 256

using namespace std;

int strings() {

setlocale(LC\_ALL, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

const int N = 550;

char Str[N] = { ' ' };

int choice;

do {

mainMenuString();

choice = checkInputString(1, 11);

switch (choice) {

case 1: {

system("cls");

cout << "Введите последовательность с клавиатуры: ";

cin.getline(Str, N, '.');

del(Str, my\_StrLen(Str), 0);

cin.sync();

int StrLength = my\_StrLen(Str);

break;

}

case 2: {

system("cls");

cout << "Введенная вами из файла строка: ";

const char\* filePath = "C:/test/hmm.txt";

ifstream file(filePath, ios::in);

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

cout << "Открыть файл не удалось! \n";

return -1;

}

file.read((char\*)Str, sizeof(Str));

file.close();

for (int i = 0; i < my\_StrLen(Str); ++i)

cout << Str[i];

cout << endl;

break;

}

case 3: {

system("cls");

deleteExtraSpaces(Str);

int StrLength = my\_StrLen(Str);

cout << "Отредактированная строка: ";

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

cout << Str[i];

}

cout << endl;

break;

}

case 4: {

system("cls");

int StrLength = my\_StrLen(Str);

deleteExtraPunctuation(Str);

cout << "Отредактированная строка: ";

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

cout << Str[i];

}

cout << endl;

break;

}

case 5: {

system("cls");

int StrLength = my\_StrLen(Str);

correctLetterCase(Str);

cout << "Отредактированная строка: ";

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

cout << Str[i];

}

cout << endl;

break;

}

case 6: {

system("cls");

int a = 0;

int length = my\_StrLen(Str);

for (int i = length; i >= 0; i--) {

if (Str[i] == ' ') {

for (int k = i + 1; k <= (i + a); k++) {

cout << Str[k];

}

a = 0;

cout << " ";

}

a++;

}

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

if (Str[i] == ' ') break;

cout << Str[i];

}

break;

}

case 7: {

system("cls");

for (int i = 0; i <= my\_StrLen(Str); i++) {

if (Str[i] == ' ')

cout << '\n';

else

cout << Str[i];

}

break;

}

case 8: {

system("cls");

const int size = 50;

char word[N];

cout << "Введите подпоследовательность, которую хотите найти: ";

cin.clear();

cin.getline(word, size, '.');

del(word, my\_StrLen(word), 0);

cin.sync();

int textLength = my\_StrLen(Str);

int wordLength = my\_StrLen(word);

int result = linearSearch(Str, word, textLength, wordLength);

if (result == -1)

cout << "Подпоследовательность не найдена" << endl;

else

cout << "Подпоследовательность найдена на позиции " << result << endl;

cout << endl;

break;

}

case 9: {

system("cls");

const int size = 50;

char word[N];

int patternIndex[N];

cout << "Введите подпоследовательность, которую хотите найти: ";

cin.clear();

cin.getline(word, size, '.');

del(word, my\_StrLen(word), 0);

cin.sync();

int result = BoyerMoore(Str, word, my\_StrLen(Str), my\_StrLen(word));

if (result == -1)

cout << "Подпоследовательность не найдена" << endl;

else

cout << "Подпоследовательность найдена на позиции " << result << endl;

cout << endl;

break;

}

case 10: {

system("cls");

const int size = 50;

char word[N];

int patternIndex[N];

cout << "Введите подпоследовательность, которую хотите найти: ";

cin.clear();

cin.getline(word, size, '.');

del(word, my\_StrLen(word), 0);

cin.sync();

cout << "Подпоследовательность найдена на позиции: ";

KMP(Str, word, patternIndex, my\_StrLen(word), print);

cout << endl;

break;

}

}

if (choice != 11)

system("pause");

} while (choice != 11);

}