**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №4**

**по дисциплине «Построение и анализ алгоритмов»**

Тема: **Алгоритм Кнута-Морриса-Пратта**

|  |  |  |
| --- | --- | --- |
| Студентгр. 8304 |  | Чешуин Д.И. |
| Преподаватель |  | Размочаева Н.В. |

Санкт-Петербург

2020

**Цель работы.**

Реализовать алгоритм Кнута-Морриса-Пратта, найти индексы начал вхождений подстроки в строку. Реализовать алгоритм проверки двух строк на циклический сдвиг

**Постановка задачи.**

**Вариант 1.**

Подготовка к распараллеливанию: работа по поиску разделяется на k равных частей, пригодных для обработки k потоками (при этом длина образца гораздо меньше длины строки поиска).

**Описание алгоритма.**

На вход алгоритму подается две строки (первая строка - строка-образец, вторая строка – строка текст для поиска). Требуется определить вхождения первой строки во вторую соответственно.

Строка-текст считывается посимвольно, в памяти хранится текущий символ.

Вычисляется префикс-функция для строки-образца.

Далее посимвольно считывается строка-текст. Переменная-счетчик для строки-образца k изначально равна 0.

Пока k> 0 и k-ый символ строки образца не совпадает с символом текста, сдвигается переменная строки образца. Далее идет проверка, является ли k-ый символ строки-образца равным символу текста, если да, то увеличивается переменная-счетчик.

Если k = размер строки-образца, обнаружено новое вхождение.

**Анализ алгоритма.**

Сложность алгоритма по операциям: O (m + n), m–длина образца, n–длина текста.

Сложность алгоритма по памяти: O (m + n) в данном случае, т.к. для распараллеливания необходимо сразу считать весь текст, m–длина образца, n–длина текста.

**Описание функций и СД.**

Для решения задачи был реализован класс KnuthMorrisPratt

void setThreadsCount(unsigned count);

Метод устанавливает количество потоков, для разбивания изначальной строки.

void setString(const string& str);

Метод устанавливает переданную строку как текст.

void setSubString(const string& str);

Метод устанавливает переданную строку как образец.

bool findSubString();

Метод ищет вхождения образца в текст в последнем, ещё не проверенном, фрагменте текста.

const vector<unsigned> getEntries();

Метод возвращает все вхождения образца в уже проверенных фрагментах текста.

**Спецификация программы.**

Программа предназначена для нахождения вхождений подстроки в строки. Программа написана на языке C++. Входными данными является число N потоков, образец и текст, а выходными – перечень вхождений образца в текст.

**Тестирование.**

Пример работы программы вывода для текста “aaabbbcccdddaabbccddabc” и образца “ab” представлен на рис. 1.

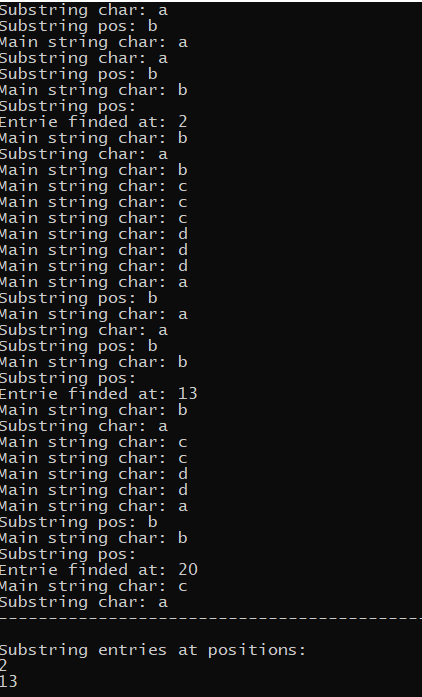


Рисунок 1 – Пример вывода для простых входных данных

**Выводы.**

В ходе выполнения лабораторной работы был реализован алгоритм КМП для поиска подстроки в строке, а также функция вычисления префикса строки.

**ПРИЛОЖЕНИЕ A.  
ИСХОДНЫЙ КОД ПРОГРАММЫ**

main.cpp.

#include <iostream>

#include <vector>

#include <cstring>

#include <fstream>

#include <algorithm>

using namespace std;

class **IOManager**

{

private:

static istream\* input;

static ostream\* output;

public:

static void **setStreamsFromArgs**(int argc, char\*\* argv)

{

if(argc > 1)

{

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

{

if(strcmp(argv[i], "-infile") == 0)

{

if(i + 1 < argc)

{

input = new ifstream(argv[i + 1]);

i += 1;

}

}

if(strcmp(argv[i], "-outfile") == 0)

{

if(i + 1 < argc)

{

output = new ofstream(argv[i + 1]);

i += 1;

}

}

}

}

}

static istream& **getIS**()

{

return \*input;

}

static ostream& **getOS**()

{

return \*output;

}

static void **resetStreams**()

{

if(input != & cin)

{

delete input;

input = &cin;

}

if(output != & cout)

{

delete output;

output = &cout;

}

}

};

class **KnuthMorrisPratt**

{

private:

unsigned threadsCount = 1;

vector<pair<string, unsigned>> threads;

string str;

string subStr;

vector<unsigned> entries;

vector<unsigned> prefixes;

private:

void **splitStrToThreads**();

public:

void **setThreadsCount**(unsigned count);

void **setString**(const string& str);

void **setSubString**(const string& str);

bool **findSubString**();

const vector<unsigned> **getEntries**();

};

int **main**(int argc, char\*\* argv)

{

IOManager::setStreamsFromArgs(argc, argv);

KnuthMorrisPratt alg;

string str, substr;

unsigned threadsCount = 1;

IOManager::getOS() << "Enter threads count." << std::endl;

IOManager::getIS() >> threadsCount;

getline(IOManager*::getIS()*, *str*, '\n');

alg.setThreadsCount(threadsCount);

IOManager::getOS() << "Enter main string." << std::endl;

getline(IOManager*::getIS()*, *str*, '\n');

alg.setString(str);

IOManager::getOS() << "Enter substring.." << std::endl;

getline(IOManager*::getIS()*, *substr*, '\n');

alg.setSubString(substr);

while(alg.findSubString());

auto entries = alg.getEntries();

IOManager::getOS() << "----------------------------------------------------------------------------" << std::endl;

if(entries.size() == 0)

{

IOManager::getOS() << "No entries.";

}

else

{

IOManager::getOS() << "Substring entries at positions:" << std::endl;

for(auto entrie : entries)

{

IOManager::getOS() << entrie << std::endl;

}

}

IOManager::getOS() << std::endl;

return 0;

}

void KnuthMorrisPratt::**setThreadsCount**(unsigned count)

{

if(count >= 1)

{

threadsCount = count;

}

if(str.size() && subStr.size())

{

splitStrToThreads();

}

IOManager::getOS() << "Threads count setted: " << count << endl;

}

void KnuthMorrisPratt::**setString**(const string& str)

{

this->str = str;

if(subStr.size())

{

splitStrToThreads();

}

IOManager::getOS() << "Main string setted: " << str << endl;

}

void KnuthMorrisPratt::**splitStrToThreads**()

{

threads.clear();

unsigned threadSize = str.size() / threadsCount;

if(str.size() % threadsCount)

{

threadSize += 1;

}

unsigned charCount = threadSize + subStr.size();

IOManager::getOS() << "Splitting main string in " << threadsCount << " parts. Parts length: " << charCount << endl;

for(unsigned i = 0; i < threadsCount; i++)

{

unsigned offset = i \* threadSize;

if(i < str.size() % threadsCount)

{

offset -= i;

}

else

{

offset -= str.size() % threadsCount;

}

threads.push\_back(make\_pair(str.substr(offset, charCount), *offset*));

IOManager::getOS() << "Part #" << i+1 << ": " << threads.back().first << endl;

}

}

void KnuthMorrisPratt::**setSubString**(const string& str)

{

prefixes.clear();

subStr = str;

//заполняем массив префиксов для подстроки

prefixes.resize(subStr.size());

prefixes[0] = 0;

IOManager::getOS() << "Calculating prefixes for substring." << std::endl;

unsigned subStrSize = subStr.size();

for(unsigned subStrIndex = 1; subStrIndex < subStrSize; subStrIndex++)

{

unsigned prefix = prefixes[subStrIndex - 1];

while(prefix > 0 && subStr[subStrIndex] != subStr[prefix])

{

prefix = prefixes[prefix - 1];

}

if(subStr[subStrIndex] == subStr[prefix])

{

prefix += 1;

}

IOManager::getOS() << "Prefix for pos " << subStrIndex << " = " << prefix << std::endl;

prefixes[subStrIndex] = prefix;

}

if(str.size())

{

splitStrToThreads();

}

IOManager::getOS() << "Substring setted: " << str << endl;

}

bool KnuthMorrisPratt::**findSubString**()

{

if(threads.size() && subStr.size())

{

string str = threads.back().first;

unsigned offset = threads.back().second;

threads.pop\_back();

unsigned subStrIndex = 0;

unsigned strSize = str.size();

unsigned subStrSize = subStr.size();

IOManager::getOS() << "Finding substring entries for string: " << str << std::endl;

for(unsigned strIndex = 0; strIndex < strSize; strIndex++)

{

IOManager::getOS() << "Main string char: " << str[strIndex] << std::endl;

while(subStrIndex > 0 && str[strIndex] != subStr[subStrIndex])

{

subStrIndex = prefixes[subStrIndex - 1];

IOManager::getOS() << "Substring char: " << subStr[subStrIndex] << std::endl;

}

if(str[strIndex] == subStr[subStrIndex])

{

subStrIndex += 1;

IOManager::getOS() << "Substring pos: " << subStr[subStrIndex] << std::endl;

}

if(subStrIndex == subStrSize)

{

entries.push\_back(strIndex - subStrSize + 1 + offset);

IOManager::getOS() << "Entrie finded at: " << entries.back() << std::endl;

}

}

return true;

}

else

{

return false;

}

}

const vector<unsigned> KnuthMorrisPratt::**getEntries**()

{

sort(entries.begin(), entries.end());

auto newEnd = unique(entries.begin(), entries.end());

entries.erase(newEnd, entries.end());

return entries;

}

istream\* IOManager::input = &cin;

ostream\* IOManager::output = &cout;