МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Казанский национальный исследовательский технический университет

им. А.Н. Туполева – КАИ»

Институт компьютерных технологий и защиты информации

Отделение СПО в ИКТЗИ (Колледж информационных технологий)

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине

Теория алгоритмов

Тема: РАЗРАБОТКА АЛГОРИТМА ПОИСКА МАКСИМАЛЬНОГО ПОТОКА В ТРАНСПОРТНОЙ СЕТИ

Работу выполнил

Студент гр.4332

Салахов К.И.

Принял

Преподаватель Валова П.А.

Казань 2024

**Цель**

Разработать алгоритм поиска максимального потока в транспортной сети.

**Задание на лабораторную работу**

Задание 1:

1. Самостоятельно задать пропускные способности дуг и построить максимальный поток в транспортной сети.

2. Найти минимальный разрез сети и проверить справедливость теоремы Форда – Фалкерсона.

Задание 2:

1. Paзpaбoтaть cxeмy aлгopитмa и пpoгpaммy, peaлизyющиe следующие мeтoды пoиcкa пoдcтpoки: алгopитм Kнyтa-Moppиca-Пpaтта, алгоритм Бойера-Мура-Хорспула

2. Cохранить фaйл “text.txt” в кaтaлoг, гдe лeжит пpoгpaммa.

3. Дoпoлнить пpoгpaммy тaк, чтoбы oнa иcкaлa вce вxoждeния cлoвa,

ввeдeннoгo c клaвиaтypы, и oбщee кoличecтвo вxoждeний, в фaйлe “text.txt ”.

5. Пepeпиcaть пpoгpaммy тaким oбpaзoм, чтoбы мexaнизм пoиcкa был peaлизoвaн oтдeльнoй фyнкциeй.

6. Cpaвнить вpeмя paбoты пpoгpaммы co вpeмeнeм paбoты пpoгpaммы пoиcкa дpyгим cпocoбoм.

**Результат выполнения работы**

|  |  |
| --- | --- |
| Сеть | Замена неориентированных дуг на ориентированные |
| Вариант 16 |  |
| Увеличивающая цепь: AD, DF;  направление дуг совпадает с направлением потока, δ = min(9-0, 4-0) = 4 | Новые потоки по дугам цепи:  AD = 0+4=4, DF=0+4=4. |
|  |  |
| Увеличивающая цепь: AE, EF;  направление дуг совпадает с направлением потока,  δ = min(7-0, 7-0) = 7 | Новые потоки по дугам цепи:  AE = 0+7=7, EF=0+7=7. |
|  |  |
| Увеличивающая цепь: AF;  направление дуг совпадает с направлением потока,  δ = min(4-0) = 4 | Новые потоки по дугам цепи:  AF = 0+4=4 |
|  |  |

Увеличивающих цепей в сети нет, поэтому максимальный поток построен и равен 15=4+4+7.

Минимальный разрез сети равен 15.

Задание 2.

Содержание текстового файла txt (Рисунок 1).

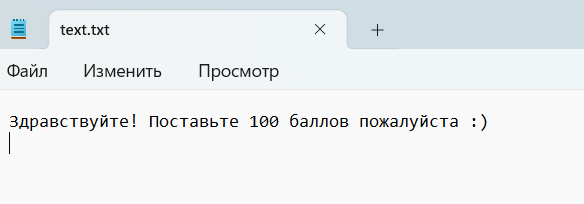


Рисунок 1 – Содержимое text.txt

Используем алгopитм Kнyтa-Moppиca-Пpaтта и алгоритм Бойера-Мура-Хорспула для поиска нужного слова в text.txt (Рисунок 2).

Изображение выглядит как текст, снимок экрана, Шрифт, черный

Автоматически созданное описание

Рисунок 2 – Результат работы программы

**Листинг**

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.IO;

class Program

{

static void Main()

{

string filePath = @"C:\Users\Камиль\OneDrive\Рабочий стол\ConsoleApp1\ConsoleApp1\ConsoleApp1\text.txt";

if (!File.Exists(filePath))

{

Console.WriteLine("Файл text.txt не найден.");

return;

}

string text = File.ReadAllText(filePath);

Console.WriteLine("Введите слово для поиска:");

string pattern = Console.ReadLine();

var words = text.Split(new[] { ' ', '.', ',', '!', '?', ';', ':', '(', ')', '\r', '\n' }, StringSplitOptions.RemoveEmptyEntries);

Stopwatch stopwatch = Stopwatch.StartNew();

var kmpMatches = FindWordPositionsKnuthMorrisPratt(words, pattern);

stopwatch.Stop();

Console.WriteLine($"Алгоритм КМП нашел {kmpMatches.Count} вхождений слова за {stopwatch.ElapsedMilliseconds} мс.");

if (kmpMatches.Count > 0)

{

Console.WriteLine("Позиция слова (КМП): " + string.Join(", ", kmpMatches));

}

stopwatch.Restart();

var bmhMatches = FindWordPositionsBoyerMooreHorspool(words, pattern);

stopwatch.Stop();

Console.WriteLine($"Алгоритм Бойера-Мура-Хорспула нашел {bmhMatches.Count} вхождений слова за {stopwatch.ElapsedMilliseconds} мс.");

if (bmhMatches.Count > 0)

{

Console.WriteLine("Позиция слова (Бойер-Мур-Хорспул): " + string.Join(", ", bmhMatches));

}

}

static List<int> FindWordPositionsKnuthMorrisPratt(string[] words, string pattern)

{

var matches = new List<int>();

int wordPosition = 0;

foreach (var word in words)

{

if (KnuthMorrisPrattSearch(word, pattern) > -1)

{

matches.Add(wordPosition);

}

wordPosition++;

}

return matches;

}

static int KnuthMorrisPrattSearch(string text, string pattern)

{

int[] prefixTable = BuildPrefixTable(pattern);

int j = 0;

for (int i = 0; i < text.Length; i++)

{

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

j = prefixTable[j - 1];

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

j++;

if (j == pattern.Length)

return i - j + 1;

}

return -1;

}

static int[] BuildPrefixTable(string pattern)

{

int[] prefixTable = new int[pattern.Length];

int j = 0;

for (int i = 1; i < pattern.Length; i++)

{

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

j = prefixTable[j - 1];

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

j++;

prefixTable[i] = j;

}

return prefixTable;

}

static List<int> FindWordPositionsBoyerMooreHorspool(string[] words, string pattern)

{

var matches = new List<int>();

int wordPosition = 0;

foreach (var word in words)

{

if (BoyerMooreHorspoolSearch(word, pattern) > -1)

{

matches.Add(wordPosition);

}

wordPosition++;

}

return matches;

}

static int BoyerMooreHorspoolSearch(string text, string pattern)

{

var shiftTable = BuildShiftTable(pattern);

int patternLength = pattern.Length;

int textLength = text.Length;

int i = 0;

while (i <= textLength - patternLength)

{

int j = patternLength - 1;

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

j--;

if (j < 0)

return i;

else

i += shiftTable.ContainsKey(text[i + j]) ? shiftTable[text[i + j]] : patternLength;

}

return -1;

}

static Dictionary<char, int> BuildShiftTable(string pattern)

{

var shiftTable = new Dictionary<char, int>();

int patternLength = pattern.Length;

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

shiftTable[pattern[i]] = patternLength - i - 1;

return shiftTable;

}

}