**Основные характеристики устройства, на котором происходило тестирование:**  
Процессор: QuadCore AMD Ryzen 5 1400, 3200 MHz  
Оперативная память: 8 ГБ DDR4-2400 1200 МГц (2 шт.)  
Загруженность ресурсов перед тестированием минимальна

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Время выполнения алгоритма** | |
| **Прямой поиск подстроки в строке** | **Алгоритм Карпа-Рабина** |
| Исходная строка – художественный текст, например,  10 000 символов из романа «Война и мир»  Искомая строка – буква «о» | Всего вхождений: 671  Время работы: 2561 тактов | Всего вхождений: 671  Время работы: 1228 тактов |
| Исходная строка – художественный текст, например,  50 000 символов из романа «Война и мир»  Искомая строка – буква «о» | Всего вхождений: 3863  Время работы: 14215 тактов | Всего вхождений: 3863  Время работы: 5707 тактов |
| Исходная строка – художественный текст, например,  100 000 символов из романа «Война и мир»  Искомая строка – буква «о» | Всего вхождений: 8126  Время работы: 25333 тактов | Всего вхождений: 8126  Время работы: 9899 тактов |

|  |  |  |
| --- | --- | --- |
| **Исходные данные** | **Время выполнения алгоритма** | |
| **Прямой поиск подстроки в строке** | **Алгоритм Карпа-Рабина** |
| Исходная строка – художественный текст, например,  10 000 символов из романа «Война и мир»  Искомая строка – «оо» | Всего вхождений: 3  Время работы: 3961 тактов | Всего вхождений: 3  Время работы: 825 тактов |
| Исходная строка – художественный текст, например,  50 000 символов из романа «Война и мир»  Искомая строка – «оо» | Всего вхождений: 8  Время работы: 5579 тактов | Всего вхождений: 8  Время работы: 2887 тактов |
| Исходная строка – художественный текст, например,  100 000 символов из романа «Война и мир»  Искомая строка – «оо» | Всего вхождений: 13  Время работы: 15495 тактов | Всего вхождений: 13  Время работы: 7552 тактов |

Проведены различные тесты с разными длинами исходной и искомой строк.

**Код программы:**

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.Diagnostics.Metrics;  using System.IO;  using System.Linq;  using System.Reflection;  using System.Text;  using System.Text.RegularExpressions;  class ExtraProblem  {  public static void Pryamoy(ref StringBuilder orig, ref StringBuilder pattern, int n, int m, StreamWriter writer)  {  //алгоритм прямого поиска строки в подстроке  List<int> list = new List<int>();  Stopwatch sw = new Stopwatch();  int count = 0; //число вхождений  writer.WriteLine("\nАлгоритм прямого поиска строки в подстроке");  sw.Restart();  int index = -1;  for (int i = 0; i <= n - m; i++)  {  int j;  for (j = 0; j < m; j++)  {  if (orig[i + j] != pattern[j])  break;  }  if (j == m)  { //все символы совпали  index = i;  count++;  list.Add(index);  }  }  sw.Stop();  if (index == -1)  writer.WriteLine("Подстрока не найдена");  else  writer.WriteLine("Всего вхождений: " + count);  for (int i = 0; i < count - 1; i++)  {  index = list[i];  writer.Write("Подстрока найдена в позиции: " + index + " ");  for (int k = index; k < index + m + 1; k++)  writer.Write(orig[k]);  writer.Write("...\n");  }  writer.WriteLine("Время работы: " + sw.ElapsedTicks + " тактов");  }  public static void Karpa\_Rabina(ref StringBuilder orig, ref StringBuilder pattern, int n, int m, StreamWriter writer)  {  //алгоритм Карпа-Рабина  Stopwatch sw = new Stopwatch();  List<int> list = new List<int>();  writer.WriteLine("\nАлгоритм Карпа-Рабина");  const long P = 67;  long[] pwp = new long[n];  pwp[0] = 1;  for (int i = 1; i < n; i++)  {  pwp[i] = pwp[i - 1] \* P;  }  int count = 0; //число вхождений  //вычисляем массив хеш-значений для всех префиксов исходной строки  long[] h = new long[n];  h[0] = (orig[0] - 'a' + 1) \* pwp[0];  for (int i = 1; i < n; i++)  {  h[i] = (orig[i] - 'a' + 1) \* pwp[i];  h[i] += h[i - 1];  }  //вычисляем хэш-значение для подстроки  long h\_s = 0;  for (int i = 0; i < m; i++)  {  h\_s += (pattern[i] - 'a' + 1) \* pwp[i];  }  int temp = 0;  sw.Restart();  //проводим поиск по хеш-значениям  for (int i = 0; i + m - 1 < n; i++)  {  //находим хэш-значение подстроки T начиная с позиции i длиною s.Length  long cur\_h = h[i + m - 1];  if (i > 0)  {  cur\_h -= h[i - 1];  }  //приводим хэш-значения двух подстрок к одной степени  if (cur\_h == h\_s \* pwp[i])  { // если хеш-значения равны, то и подстроки равны  temp = 1;  count++;  list.Add(i);  }  }  sw.Stop();  if (temp == 0)  writer.WriteLine("Подстрока не найдена");  else  {  writer.WriteLine("Всего вхождений: " + count);  for (int o = 0; o < count - 1; o++)  {  int i = list[o];  writer.Write("Подстрока найдена в позиции: " + i + " ");  for (int k = i; k < i + m + 1; k++)  writer.Write(orig[k]);  writer.Write("...\n");  }  }  writer.WriteLine("Время работы: " + sw.ElapsedTicks + " тактов");  }  static void Main(string[] args)  {  string outpuFile = "result.txt";  string inputFile = "wap.txt";  using (StreamWriter writer = new StreamWriter(outpuFile, false))  {  using (StreamReader reader = new StreamReader(inputFile))  {  char c;  int SymbolsCount = 100000;  //int StartPosition = 0;  //int Counter = 0;  //string InputString = "";  StringBuilder InputString = new StringBuilder(100000);  StringBuilder search = new StringBuilder(1);  search.Append('о');  //search.Append('о');  //int idx = 0;  while (reader.Peek() != -1)  {  c = (char)reader.Read();  SymbolsCount--;  InputString.Append(c);  if (SymbolsCount == 0)  {  break;  }  }  int m = search.Length;  int n = InputString.Length;  writer.WriteLine($"Длина строки {n}, длина искомой строки {m}");  Pryamoy(ref InputString, ref search, n, m, writer);  Karpa\_Rabina(ref InputString, ref search, n, m, writer);  writer.WriteLine();  }  }  }  } |

**Вывод:** Во всех приведенных случаях поиска подстроки алгоритм Карпа-Рабина показал лучший результат. Стоит уточнить, что этот алгоритм показал себя наиболее эффективнее (по сравнению с прямым поиском) при увеличении длины исходной строки и количества вхождений искомой подстроки.