Федеральное государственное образовательное бюджетное учреждение

высшего профессионального образования

«Нижегородский Государственный Университет им.

Н.И.Лобачевского» (ННГУ)

Институт Информационных Технологий Математики и Механики

Отчёт по лабораторной работе

Поиск фрагмента в тексте: наивный алгоритм и алгоритм Кнута-Морриса-Пратта

Выполнил:

студент группы 3821Б1ФИ1

Балясов И. А.

Проверил:

лаборант, преподаватель

Уткин Г.В

Нижний Новгород

2023г.

Содержание

[Введение 3](#_Toc150449740)

[Постановка задачи 4](#_Toc150449741)

[Руководство пользователя 5](#_Toc150449744)

[Руководство программиста 8](#_Toc150449745)

[Эксперименты 10](#_Toc150449747)

[Заключение 13](#_Toc150449748)

[Список литературы 14](#_Toc150449749)

[Приложение 15](#_Toc150449750)

# Введение

**Программирование** — это процесс превращения алгоритма в нотацию, написанную на языке программирования, которая может быть выполнена компьютером. Одной из ключевых задач компьютера является работа с данными.

Поиск фрагмента текста — является неотъемлемой частью программирования и используется во многих приложениях. Этот процесс может показаться простым, но на самом деле он требует определенных навыков и знаний.

К примеру, для поиска информации в браузере используется «поиск фрагмента в тексте». Браузер берет ваш запрос и начинает искать по множеству сайтов, чтобы выдать целевую информацию о вашем запросе.

В ходе выполнения лабораторной работой на языке программирования «С++» будет написана программа, реализующие алгоритмы поиска фрагмента в тексте.

# Постановка задачи

Для слов Y = Y1Y2 … Yn  и X = X1X2 … Xm в алфавите A найти все вхождения слова Y в X. В дальнейшем под решением этой задачи мы будем понимать такую функцию f: {1, 2, … ,m}→{0, 1, 2, … , n}, что f[j] = n, если Y1Y2 …Yn =X j-(n-1)X j-(n-2) … X j  (на j-ой букве слова X заканчивается очередное вхождение в него слова Y), и f[j]<n в противном случае. Для избавления псевдокодов алгоритмов от мешающих их пониманию деталей, мы будем в дальнейшем считать, что 1≤n≤m.

### Наивный алгоритм поиска фрагмента в тексте

Суть данного алгоритма иллюстрируется рамкой длины n, движущейся по слову X. Если фрагмент слова X, находящийся в рамке, совпадает со словом Y, это означает, что очередное вхождение слова Y в X найдено.

Временная сложность наивного алгоритма есть O(n⋅m).

### Алгоритм Кнута-Морриса-Пратта

В общем виде алгоритм Кнута-Морриса-Пратта строит функцию откатов fY: {1, 2, … ,n}→{0, 1, 2, … , n-1} для поданного на его вход слова Y = Y1Y2 …Yn. Таким образом, fY[i] является длиной наибольшего собственного суффикса слова Y = Y1Y2 …Yn, который является также и его префиксом.

Тогда исходную задачу поиска всех вхождений слова Y = Y1Y2 …Yn  в слово (текст) X = X1X2 …Xm решит построение функции откатов fZ для слова Z=Y#X, где # является произвольной буквой, не принадлежащей алфавиту A. В этом случае искомую функцию f можно найти, положив f[i]=fZ[i+n+1], i = 1, …, m. Сохраняя неизменной суть данного построения функции f, удается организовать его так, чтобы не было необходимости вводить в алфавит новую букву. Таким образом, алгоритм Кнута-Морриса-Пратта поиска всех вхождений слова Y=Y1Y2 …Yn  в X=X1X2 …Xm можно представить в следующем виде.

Временная сложность алгоритма Кнута-Морриса-Пратта есть O(n+m).

# Руководство пользователя

После запуска программа выводит в консоль интерфейс (рис 1.). Пользователь может выбрать один из предложенных вариантов действий.

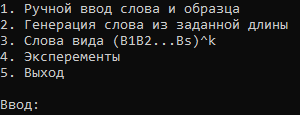


Рисунок 1. Консоль после запуска программы.

**Функционал «Ручной ввод слова и образца»**

Возможность проихвольного ввода данных и обработка их двумя алгоритмами (рис. 2).

Для начала нужно ввести сам текста, а затем нужно ввести образец, по которому будет произведен поиск. К примеру, введем в «текст»: “abcabcbbc”, а в «образец»: “abc”. После каждого введенного значения необходимо нажимать кнопку «Enter». (рис. 3)

Когда данные будут введены, в консоль будет выведен, где можно ознакомится с результатами. (рис 4)



Рисунок 3. Ввод данных.

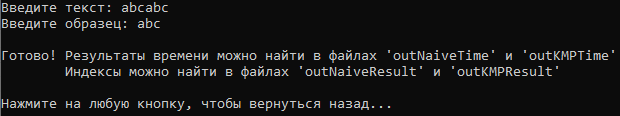


Рисунок 4. Завершение «Ручной ввод слова и образца»

**Функционал «Генерация слова из заданной длины»**

Возможность сгенерировать текст и образец по заданным параметрам (рис. 5):

**1. Введите количество букв в алфавите.** Здесь нужно задать сколько букв будет в начальном слове. К примеру, впишем 3.

**2. Введите буквы алфавита.** Здесь нужно задать, какие буквы будут в слове. К примеру, впишем “abc”.

**3. Введите длину слова.** Здесь нужно задать, какой длины будет сгенерировано слово. К примеру, впишем 10.

Для текста и образца будет предложено отдельные вводы параметров. После каждого введенного значения необходимо нажимать кнопку «Enter».

**Важно!** Длина текста, в котором будет производится поиск, не может быть меньше длины образца. Если данное условие не будет выполнено, в консоль будет выведена соответствующая ошибка и данные нужно вводить заново.

После ввода данных, будет выведено в консоль оповещение о запуске двух алгоритмов. После выполнения обоих алгоритмах будет соответствующее оповещение. С результатами обоих алгоритмов можно будет ознакомится в соответствующих файлах, которые будут указаны в консоли (рис 6).

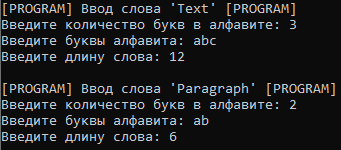


Рисунок 5. Ввод данных.

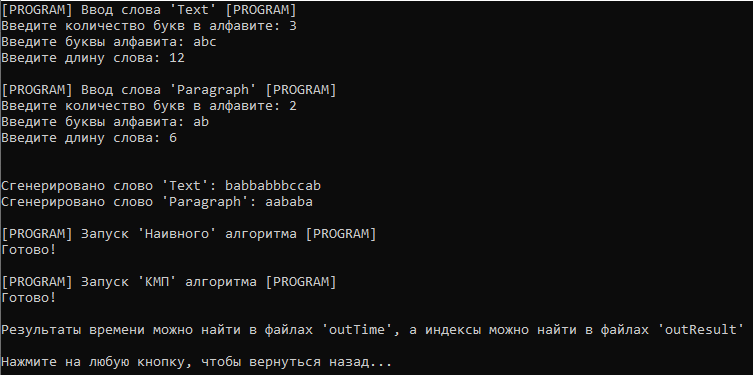


Рисунок 6. Окончание работы «Генерация слова из заданной длины»

**Функционал «Слова вида (В1В2...Вs)^k»**

Возможность сгенерировать текст и образец по заданным параметрам (рис. 7):

**1. Введите слово, которое будет представлено в виде «(В1В2...Вs)^k».** Здесь нужно будет ввести слово.

**2. Введите число k.** Здесь нужно ввести число, которое будет равняться степени повторения введенного слова.

Для текста и образца будет предложено отдельные вводы параметров. После каждого введенного значения необходимо нажимать кнопку «Enter».

**Важно!** Длина текста, в котором будет производится поиск, не может быть меньше длины образца. Если данное условие не будет выполнено, в консоль будет выведена соответствующая ошибка и данные нужно вводить заново.

После ввода данных, будет выведено в консоль оповещение о запуске двух алгоритмов. После выполнения обоих алгоритмах будет соответствующее оповещение. С результатами обоих алгоритмов можно будет ознакомится в соответствующих файлах, которые будут указаны в консоли (рис. 8).

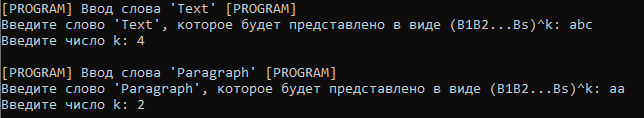


Рисунок 7. Ввод данных

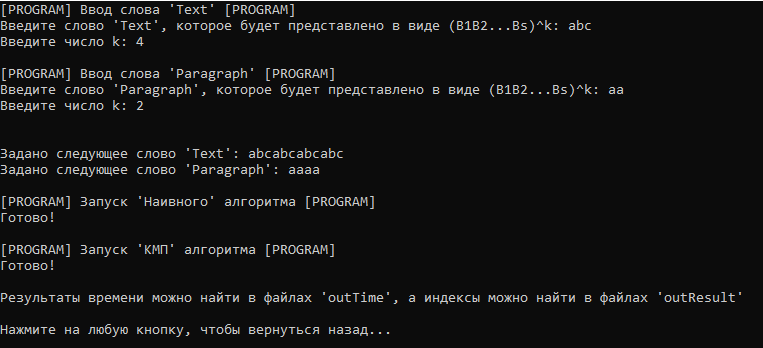


Рисунок 8. Окончание работы «Слова вида (B1B2…Bs)^k»

# Руководство программиста

## Описание алгоритмов

**Описание наивного поиска.**

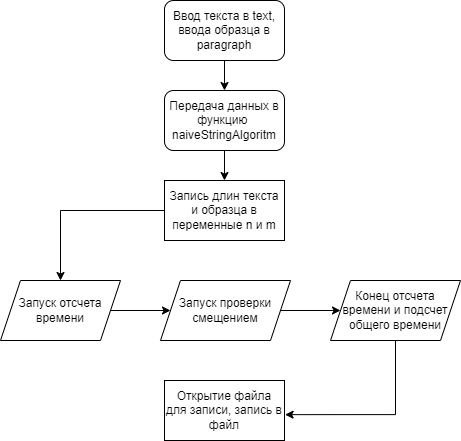


Рисунок 9. Описание алгоритма

**Описание Кнута-Морриса-Прата.**

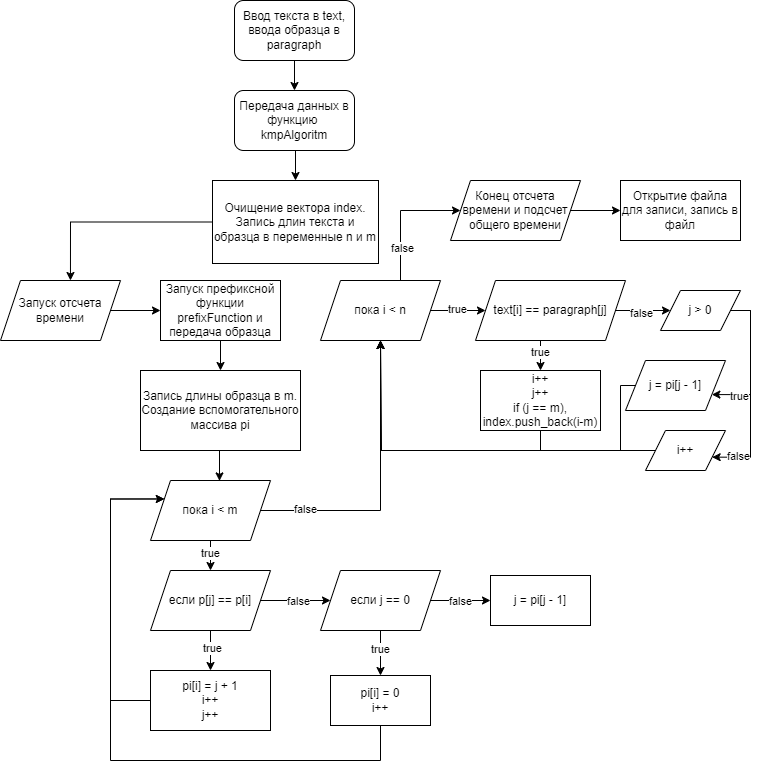


Рисунок 10. Описание алгоритма

# Эксперименты

Проведем серию экспериментов.

**Эксперимент 4.1.** Y=(ab)k и X=(ab)1000\*k, k = 1, … ,1001 с шагом 10 (нарисовать графики функций T1(k) и T2(k)).

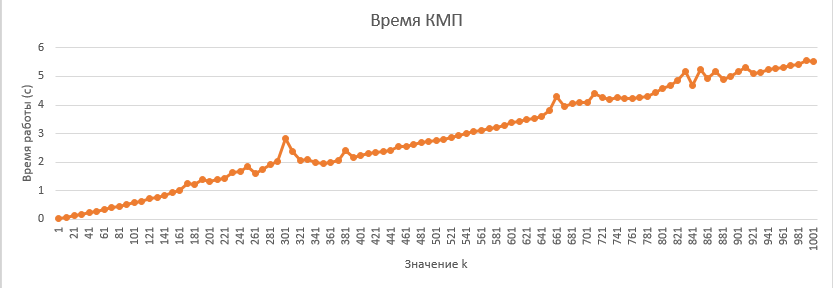


Рисунок 11. Диаграмма времени работы «КМП» алгоритма.

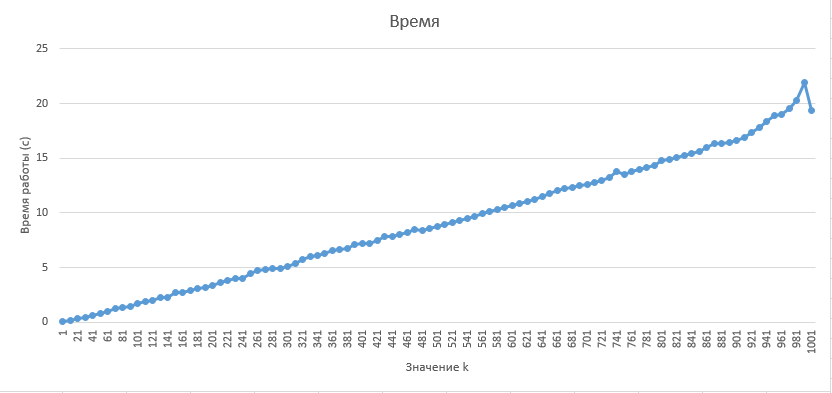


Рисунок 12. Диаграмма времени работы «Наивного» алгоритма.

**Эксперимент 4.2.** A={a,b}, Y=(a)m, слово X состоит из 106+1 букв алфавита A, встречающихся равновероятно, m = 1, … ,106+1 с шагом 104 (нарисовать графики функций T1(m) и T2(m)) .



Рисунок 13. Диаграмма времени работы “Наивного” алгоритма.

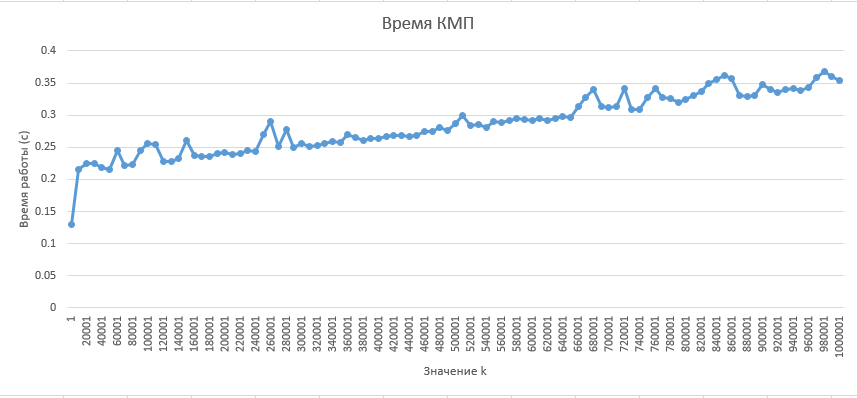


Рисунок 14. Диаграмма времени работы «КМП» алгоритма.

**Эксперимент 4.3.** Y=aaaaa и X=(aaaaab)h, h = 1, … ,106+1 с шагом 104 (нарисовать графики функций T1(h) и T2(h)).



Рисунок 15. Диаграмма времени работы “Наивного” алгоритма.

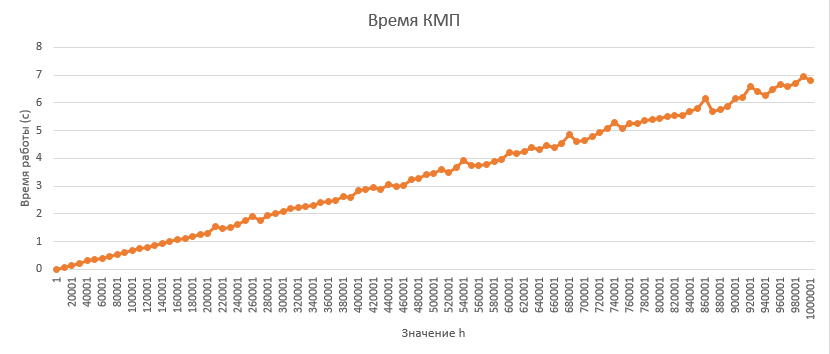


Рисунок 16. Диаграмма времени работы «КМП» алгоритма.

При тестировании программы с различными входными данными в среде разработки Visual Studio никаких ошибок или предупреждений не выявлено.

# Заключение

В ходе лабораторной работы мною была написана программа на языке «С++», которая реализует алгоритмы поиска фрагмента в тексте, такие как: «Наивный» алгоритм и алгоритм «Кнута-Морриса-Пратта».

Выполнение данной лабораторной работы показало, что у каждого алгоритма разная скорость работы. «Наивный» алгоритм стоит использовать тогда, когда внутренний цикл прерывается быстро (на 1-3 шаге, скажем, как в простом случае). Но если образец и строка содержат часто повторяющиеся вложенные куски, то следует выбирать на выполнение алгоритм «Кнута-Морриса-Пратта».

Выполнение подобных работ позволяет значительно улучшить свои навыки программирования, а также улучшить понимание работы с компьютером и средой разработки, в тоже время позволяет закрепить получаемые знания в области программирования и отработать их применения на практике.

Список литературы

1. А. Ахо, Дж. Хопкрофт, Дж. Ульман. Построение и анализ вычислительных алгоритмов.
2. В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. Лекции по теории графов..
3. Ф. Препарата, М. Шеймос. Вычислительная геометрия: Введение.
4. А. Шень. Программирование: теоремы и задачи.
5. В. Е. Алексеев, В. А. Таланов. Графы. Модели вычислений. Структуры данных: Учебник.
6. Н. Вирт Алгоритмы и структуры данных.
7. Т. Кармэн, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: Построение и анализ.
8. Методичка с описанием работ.

# Приложение

**1. main.cpp**

|  |
| --- |
| #include <iostream>  #include <conio.h>  #include "Algoritms.h"  using namespace std;  int main(){  setlocale(0, "Rus");  int answer;  bool flag = true;  string text;  string paragraph;    while (flag)  {  system("cls");  cout << "1. Ручной ввод слова и образца\n2. Генерация слова из заданной длины\n3. Слова вида (B1B2...Bs)^k"  << "\n4. Эксперементы\n5. Выход\n" << endl;  cout << "Ввод: ";  cin >> answer;  switch (answer)  {  case 1:  {  system("cls");  cout << "Введите текст: ";  cin >> text;  cout << "Введите образец: ";  cin >> paragraph;  naiveStringAlgoritm(text, paragraph);  kmpAlgoritm(text, paragraph);  cout << "\nГотово! Результаты времени можно найти в файлах 'outNaiveTime' и 'outKMPTime'"  << "\n Индексы можно найти в файлах 'outNaiveResult' и 'outKMPResult'" << endl;  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  char ch = \_getch();  break;  }  case 2:  {  system("cls");  int alphabetSizeText = 0;  int alphabetSizeParagraph = 0;  int sizeWordText = 0;  int sizeWordParagraph = 0;  string alphabetText;  string alphabetParagraph;  cout << "[PROGRAM] Ввод слова 'Text' [PROGRAM]" << endl;  cout << "Введите количество букв в алфавите: ";  cin >> alphabetSizeText;  cout << "Введите буквы алфавита: ";  cin >> alphabetText;  cout << "Введите длину слова: ";  cin >> sizeWordText;  cout << endl << "[PROGRAM] Ввод слова 'Paragraph' [PROGRAM]" << endl << "Введите количество букв в алфавите: ";  cin >> alphabetSizeParagraph;  cout << "Введите буквы алфавита: ";  cin >> alphabetParagraph;  cout << "Введите длину слова: ";  cin >> sizeWordParagraph;  if (sizeWordText < sizeWordParagraph)  {  cout << "\nОшибка! Длинна текста, в котором будет производится поиск, не может быть меньше длины образца." << endl;  cout << "Вернитесь в меню и повторите ввод с верными данными... Нажмите любую кнопку..." << endl;  char ch = \_getch();  break;  }  cout << endl << endl;  text = generateWord(alphabetSizeText, alphabetText, sizeWordText);  cout <<"Сгенерировано слово 'Text': " << text << endl;  paragraph = generateWord(alphabetSizeParagraph, alphabetParagraph, sizeWordParagraph);  cout << "Сгенерировано слово 'Paragraph': " << paragraph << endl;  cout << endl << "[PROGRAM] Запуск 'Наивного' алгоритма [PROGRAM]";  naiveStringAlgoritm(text, paragraph);  cout << "\nГотово!" << endl << endl << "[PROGRAM] Запуск 'КМП' алгоритма [PROGRAM]" << "\nГотово!" << endl;  kmpAlgoritm(text, paragraph);  cout << "\nРезультаты времени можно найти в файлах 'outTime', а индексы можно найти в файлах 'outResult'" << endl;  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  char ch = \_getch();  break;  }  case 3:  {  system("cls");  int k = 0;  string newText;  string newParagraph;  cout << "[PROGRAM] Ввод слова 'Text' [PROGRAM]" << endl;  cout << "Введите слово 'Text', которое будет представлено в виде (B1B2...Bs)^k: ";  cin >> newText;  cout << "Введите число k: ";  cin >> k;  text = genRepeatWord(newText, k);  cout << endl << "[PROGRAM] Ввод слова 'Paragraph' [PROGRAM]" << endl;  cout << "Введите слово 'Paragraph', которое будет представлено в виде (B1B2...Bs)^k: ";  cin >> newParagraph;  cout << "Введите число k: ";  cin >> k;  paragraph = genRepeatWord(newParagraph, k);  if (text.length() < paragraph.length())  {  cout << "\nОшибка! Длинна текста, в котором будет производится поиск, не может быть меньше длины образца." << endl;  cout << "Вернитесь в меню и повторите ввод с верными данными... Нажмите любую кнопку..." << endl;  char ch = \_getch();  break;  }  cout << endl << endl;  cout << "Задано следующее слово 'Text': " << text << endl;  cout << "Задано следующее слово 'Paragraph': " << paragraph << endl;  cout << endl << "[PROGRAM] Запуск 'Наивного' алгоритма [PROGRAM]";  naiveStringAlgoritm(text, paragraph);  cout << "\nГотово!" << endl << endl << "[PROGRAM] Запуск 'КМП' алгоритма [PROGRAM]" << "\nГотово!" << endl;  kmpAlgoritm(text, paragraph);  cout << "\nРезультаты времени можно найти в файлах 'outTime', а индексы можно найти в файлах 'outResult'" << endl;  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  char ch = \_getch();  break;  }  case 4:  {  char ch;  system("cls");  cout << "1. Эксперемент 4.1\n2. Эксперемент 4.2\n3. Эксперемент 4.3\n4. Назад" << endl;  cout << "Ввод: ";  cin >> answer;  switch (answer)  {  case 1:  system("cls");  cout << "[PROGRAM] Запуск эксперемента 4.1 [PROGRAM]" << endl;  cout << "Пожалуйста, дождитесь завершения эксперемента! Не закрывайте консоль!" << endl << endl;  experiment4\_1();  cout << "\nГотово. С результатами эксперемента можно ознакомится в файле 'timeForExperiment4\_1.txt'";  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  ch = \_getch();  break;  case 2:  system("cls");  cout << "[PROGRAM] Запуск эксперемента 4.2 [PROGRAM]" << endl;  cout << "Пожалуйста, дождитесь завершения эксперемента! Не закрывайте консоль!" << endl << endl;  experiment4\_2();  cout << "\nГотово. С результатами эксперемента можно ознакомится в файлах: 'timeNaiveForExperiment4\_2.txt'"  << "и 'timeKMPForExperiment4\_2.txt'";  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  ch = \_getch();  break;  case 3:  system("cls");  cout << "[PROGRAM] Запуск эксперемента 4.3 [PROGRAM]" << endl;  cout << "Пожалуйста, дождитесь завершения эксперемента! Не закрывайте консоль!" << endl << endl;  experiment4\_3();  cout << "\nГотово. С результатами эксперемента можно ознакомится в файле 'timeNaiveForExperiment4\_3.txt'"  << "и 'timeKMPForExperiment4\_3.txt'";  cout << "\nНажмите на любую кнопку, чтобы вернуться назад..." << endl;  ch = \_getch();  break;  case 4:  break;  default:  break;  }  break;  }  case 5:  {  flag = false;  cout << "\nДо новых встречь!" << endl;  break;  }  default:  break;  }  }  return 0;  } |

**2. Algoritms.h**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <chrono>  #include <string>  #include <vector>  #include <random>  #include "workOfFile.h"  using namespace std;  vector <int> index;  int\* pi;  string generateWord(const int& alphabetSize, const string& alphabet, const int& sizeWord) {  srand(time(nullptr));  random\_device random;  mt19937 gen(random());  uniform\_int\_distribution <int> dis(0, alphabetSize - 1);  string word;  for (int i = 0; i < sizeWord; i++)  {  int randomIndex = dis(gen);  word += alphabet[randomIndex];  }  return word;  }  string genRepeatWord(const string& word, const int& k) {  string result;  for (int i = 0; i < k; i++)  {  result += word;  }  return result;  }  void naiveStringAlgoritm(const string &text, const string &paragraph) {    ofstream outputFileInTime("outNaiveTimeAlgoritm.txt");  int n = text.length();  int m = paragraph.length();  auto startTime = chrono::high\_resolution\_clock::now();  for (int i = 0; i <= n - m; i++)  {  if (text.substr(i, m) == paragraph)  index.push\_back(i);  }  auto endTime = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTime = endTime - startTime;  outputFileInTime << "Время работы 'Наивного' алгоритма: " << totalTime.count() << " секунд.";  writtingResultToFile("outNaiveResultAlgoritm.txt", index);  }  void prefixFunction(const string& p) {  int m = p.length();  pi = new int[m];  pi[0] = 0;  int j = 0, i = 1;  while (i < m)  {  if (p[j] == p[i])  {  pi[i] = j + 1;  i++;  j++;  }  else  {  if (j == 0)  {  pi[i] = 0;  i++;  }  else  {  j = pi[j - 1];  }  }  }  }  void kmpAlgoritm(const string& text, const string& paragraph) {    index.clear();    int n = text.length();  int m = paragraph.length();  auto startTime = chrono::high\_resolution\_clock::now();  prefixFunction(paragraph);  int i = 0;  int j = 0;  while (i < n)  {  if (text[i] == paragraph[j])  {  i++;  j++;  if (j == m)  index.push\_back(i - m);  }  else  {  if (j > 0)  j = pi[j - 1];  else  i++;  }  }  auto endTime = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTime = endTime - startTime;  ofstream outputFileInTime("outKMPTimeAlgoritm.txt");  outputFileInTime << "Время работы алгоритма 'Кнута-Морриса-Пратта': " << totalTime.count() << " секунд.";  writtingResultToFile("outKMPResultAlgoritm.txt", index);  }  void experiment4\_1() {  ofstream outNaiveTimeExperement4\_1;  ofstream outKMPTimeExperement4\_1;    outNaiveTimeExperement4\_1.open("timeNaiveForExperiment4\_1.txt", std::fstream::app);  outKMPTimeExperement4\_1.open("timeKMPForExperiment4\_1.txt", std::fstream::app);  string text\_old = "ab", paragraph\_old = "ab";  string text\_new, paragraph\_new;  for (int k = 1; k <= 1001; k += 10)  {  int persent = k / 10;  if (persent % 10 == 0)  {  cout << "Готово на " << persent << "%..." << endl;  }  text\_new = genRepeatWord(text\_old, 1000 \* k);  paragraph\_new = genRepeatWord(paragraph\_old, k);  // =========================================================  auto startTimeNaive = chrono::high\_resolution\_clock::now();  naiveStringAlgoritm(text\_new, paragraph\_new);  auto endTimeNaive = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTimeNaive = endTimeNaive - startTimeNaive;  outNaiveTimeExperement4\_1 << k << " " << totalTimeNaive.count() << endl;  // =========================================================  auto startExperement = chrono::high\_resolution\_clock::now();  kmpAlgoritm(text\_new, paragraph\_new);  auto endExperement = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTimeExperement = endExperement - startExperement;  outKMPTimeExperement4\_1 << k << " " << totalTimeExperement.count() << endl;  // =========================================================  }  }  void experiment4\_2() {  ofstream outTimeNaiveAlgoritm;  ofstream outTimeKMPAlgoritm;  outTimeNaiveAlgoritm.open("timeNaiveForExperiment4\_2.txt", std::fstream::app);  outTimeKMPAlgoritm.open("timeKMPForExperiment4\_2.txt", std::fstream::app);  int alphabetSizeText = 2;  int wordLenghtText = 1000001;  string alphabetText = "ab";  string paragraph, paragraph\_init = "a";  string text = generateWord(alphabetSizeText, alphabetText, wordLenghtText);  for (int m = 1; m <= 1000001; m += 10000)  {  int persent = (m - 1) / 10000;  if (persent % 10 == 0)  {  cout << "Готово на " << persent << "%..." << endl;  }  paragraph = genRepeatWord(paragraph\_init, m);  // =========================================================  auto startTimeNaive = chrono::high\_resolution\_clock::now();  naiveStringAlgoritm(text, paragraph);  auto endTimeNaive = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTimeNaive = endTimeNaive - startTimeNaive;  outTimeNaiveAlgoritm << m << " " << totalTimeNaive.count() << endl;  // =========================================================  auto startTimeKMP = chrono::high\_resolution\_clock::now();  kmpAlgoritm(text, paragraph);  auto endTimeKMP = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTimeKMP = endTimeKMP - startTimeKMP;  outTimeKMPAlgoritm << m << " " << totalTimeKMP.count() << endl;  // =========================================================  }  }  void experiment4\_3() {  ofstream outTimeNaiveAlgoritm;  ofstream outTimeKMPAlgoritm;  outTimeNaiveAlgoritm.open("timeNaiveForExperiment4\_3.txt", std::fstream::app);  outTimeKMPAlgoritm.open("timeKMPForExperiment4\_3.txt", std::fstream::app);  string text;  string paragraph = "aaaaa";  for (int h = 1; h <= 1000001; h += 10000)  {  int persent = (h - 1) / 10000;  if (persent % 10 == 0)  {  cout << "Готово на " << persent << "%..." << endl;  }  text = genRepeatWord(paragraph + "b", h);  // =====================================================  auto startTimeNaive = chrono::high\_resolution\_clock::now();  naiveStringAlgoritm(text, paragraph);  auto endTimeNaive = chrono::high\_resolution\_clock::now();  chrono::duration <double> totaltimeNaive = endTimeNaive - startTimeNaive;  outTimeNaiveAlgoritm << h << " " << totaltimeNaive.count() << endl;  // ======================================================  auto startTimeKMP = chrono::high\_resolution\_clock::now();  kmpAlgoritm(text, paragraph);  auto endTimeKMP = chrono::high\_resolution\_clock::now();  chrono::duration <double> totalTimeKMP = endTimeKMP - startTimeKMP;  outTimeKMPAlgoritm << h << " " << totalTimeKMP.count() << endl;  // ======================================================  }  } |

**3. workOfFile.h**

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  void writtingResultToFile(const string& fileName, vector <int> &index) {  ofstream outputFile(fileName);  if (index.size() == 0)  {  outputFile << "Обзац не найден в строке";  }  else  {  outputFile << "Обзац был найден в строке. Индексы, начиная с которых происходит совпадение: ";  for (int k = 0; k < index.size(); k++)  outputFile << index[k] << " ";  }  } |