|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий | |
|  | |
| Кафедра Математического обеспечения и стандартизации информационных технологий | |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 10** | |
| **по дисциплине** | |
| **«**Структуры и алгоритмы обработки данных**»**  **Тема: «Поиск в тесте образца. Алгоритмы. Эффективность алгоритмов»** | |
|  | |
| Выполнил студент группы ИКБО-08-22 | Сенькевич Г.Д. |
| Принял преподаватель | Филатов А.С. |

|  |  |  |
| --- | --- | --- |
| Лабораторная работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2023

# **Цель работы**

Получить знания и навыки применения алгоритмов поиска в тексте подстрок (образца).

# **Постановка задачи**

Разработать и реализовать алгоритм поиска в тексте.

1. Включить в этап «Решение» описание алгоритма рассматриваемого метода. Разобрать алгоритм на примере. Подсчитать количество сравнений для успешного поиска первого вхождения образца в текст и безуспешного поиска.
2. Разработать и отладить функции для реализации алгоритма.
3. Сформировать таблицу тестов с указанием успешного и неуспешного поиска, используя большой по объему текст, и образец различного объема. Включить ее в этап тестирования
4. Разработать и реализовать программу тестирования алгоритма.
5. Оценить практическую сложность алгоритма в зависимости от длины текста и длины образца и отобразить результаты в таблице.
6. Оформить отчет, включив в него этапы разработки каждой задачи варианта. Сравнить эффективность алгоритма, как практическую, так и теоретическую.

Вариант №23. Условие задания:

|  |  |
| --- | --- |
| Упражнение 1 | Дан пакет из n документов. Каждый документ = это текст протокола собрания коллектива. В протоколе есть фраза: слушали сообщение: после которой через пробел следует фамилия и инициалы (записаны по формату: Иванов И.И.) выступившего. Сформировать массив данных по выступившим для каждого протокола. |
| Упражнение 2 | Определить, является ли строка номером телефона в формате +7-000-000-00-00 |

# **Решение**

Поиск в тесте образца является важной задачей в области информационного поиска и анализа данных. Он заключается в поиске определенного образца, называемого образцом или шаблоном, внутри целевого текстового или числового потока данных, называемого текстом.

Целью поиска в тесте образца является обнаружение всех вхождений образца в тексте и определение их положения. Это может быть полезно для решения различных задач, таких как поиск подстроки в тексте, сравнение двух текстовых документов, распознавание образцов в изображениях или поиск генетических последовательностей в биоинформатике.

Алгоритм Кнута-Морриса-Пратта (KMP) является эффективным алгоритмом поиска подстроки в строке. Основная идея алгоритма КМП заключается в использовании информации о самой подстроке, чтобы избежать бесполезных сравнений при поиске в тексте. Алгоритм основан на предподсчете префикс-функции (также известной как функция отката) для подстроки.

Префикс-функция для данной подстроки определяет длину наибольшего собственного суффикса этой подстроки, который также является её префиксом. Например, для подстроки "ABABC" префикс-функция будет равна [0, 0, 1, 2, 0].

Алгоритм КМП использует префикс-функцию для определения количества символов, на которые можно сдвинуть позицию проверки при несоответствии символов между подстрокой и текстом. Это позволяет пропускать более чем один символ при поиске.

Основной шаг алгоритма состоит в итеративном сравнении символов подстроки и текста. Если символы совпадают, проверка переходит к следующему символу в обеих строках. Однако, если символы не совпадают, алгоритм использует префикс-функцию, чтобы определить новую позицию для сравнения в тексте, пропуская некоторые символы подстроки.

Таким образом, алгоритм КМП обеспечивает эффективный поиск подстроки без необходимости повторного сравнения символов, которые уже были проверены. Сложность алгоритма составляет O(n + m), где n - длина текста, а m - длина подстроки.

Для решения первого упражнения были написаны функции read\_file, fill\_LPS и KMP. Функция read\_file текст из файла. Она принимает на вход имя файла и возвращает строку. Функция fill\_LPS заполняет массив для поиска с помощью алгоритма Кнута-Морисса-Прата. Функция KMP реализует алгоритм поиска Кнута-Морисса-Прата. Она принимает на вход две строки – текст и строка, которую в нём нужно найти, а также ссылку на целочисленную переменную, содержащую количество сравнений во время поиска, и возвращает вектор со всеми вхождениями строки.

|  |
| --- |
| std::string read\_file(std::string filename)  {  std::ifstream f(filename);  std::stringstream buffer;  buffer << f.rdbuf();  return buffer.str();  }  void fill\_LPS(std::string pattern, std::vector<int>& lps, int& comparisons)  {  int len = 0;  lps[0] = 0;  for (int i = 1; i < lps.size(); i++)  {  comparisons += 2;  if (pattern[i] == pattern[len])  {  len++;  lps[i] = len;  }  else  {  comparisons++;  if (len != 0)  {  len = lps[len - 1];  i--;  }  else  {  lps[i] = 0;  }  }  }  }  std::vector<int> KMP(std::string text, std::string pattern, int& comparisons)  {  std::vector<int> found\_indexes;  std::vector<int> lps(pattern.length());  fill\_LPS(pattern, lps, comparisons);  int i = 0, j = 0;  comparisons++;  while ((text.length() - i) >= (pattern.length() - j))  {  comparisons++;  if (pattern[j] == text[i])  {  i++;  j++;  }  comparisons++;  if (j == pattern.length())  {  found\_indexes.push\_back(i - j);  j = lps[j - 1];  }  else if (i < text.length() && pattern[j] != text[i])  {  comparisons += 3;  if (j != 0)  {  j = lps[j - 1];  }  else  {  i++;  }  }  }  return found\_indexes;  } |

Для решения второго упражнения была написана функция is\_phone\_number, которая принимает на вход строку и возвращает булево значение, которое указывает, является ли строка номером телефона.

|  |
| --- |
| bool is\_phone\_number(std::string input)  {  std::regex pattern("\\+7-\\d{3}-\\d{3}-\\d{2}-\\d{2}");  return std::regex\_match(input, pattern);  } |

При запуске программы пользователь видит пользовательское меню, позволяющее выбрать между двумя реализованными функциями и функцией тестирования алгоритма поиска.

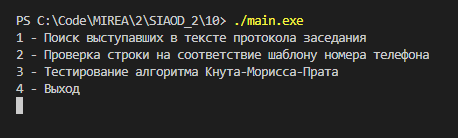


Рисунок 1. Интерфейс программы

# **Тестирование**

Протестируем выполнение программой первого упражнения. Для этого создадим файл, содержащий ключевые слова «Слушали сообщение:» и две фамилии, и выберем в меню первую функцию. На рисунке 2 видим, как программа вывела верный ответ.

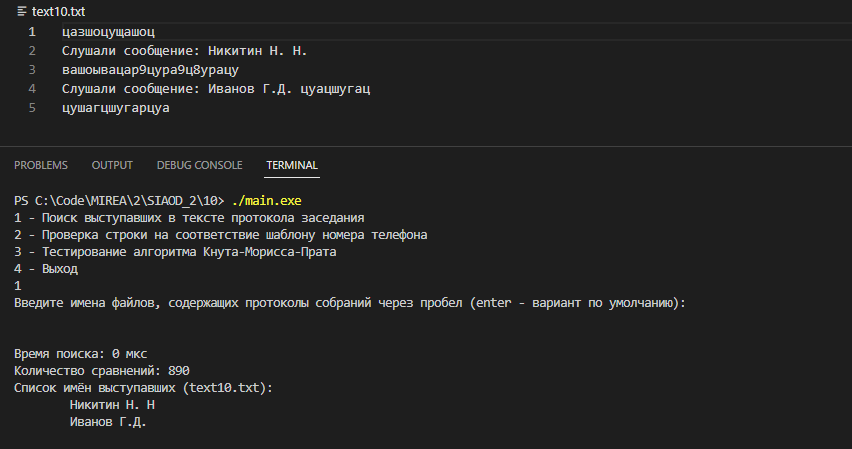


Рисунок 2. Тестирование программы

Протестируем выполнение программой первого упражнения. Для этого выберем в меню вторую функцию и введём номер телефона. На рисунке 3 видим, как программа вывела верный ответ.

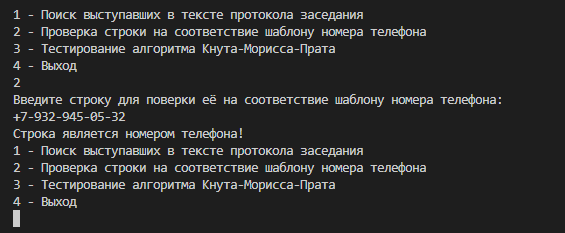


Рисунок 3. Тестирование программы

Протестируем алгоритм поиска Кнута-Морисса-Прата на разных размерах файлов.

Таблица 1. Рабочий прогон программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **n** | **m** | **T(n), мкс** | **Та=f(N+M)** | **Тэ=C** |
| 100000 | 1000 | 4222 | 101000 | 499651 |
| 100000 | 50000 | 4993 | 150000 | 503581 |
| 200000 | 2000 | 7968 | 202000 | 999293 |
| 200000 | 100000 | 10994 | 300000 | 1007057 |
| 300000 | 3000 | 8209 | 303000 | 1498870 |
| 300000 | 150000 | 16006 | 450000 | 1510968 |
| 400000 | 4000 | 14848 | 404000 | 1998361 |
| 400000 | 200000 | 12639 | 600000 | 2014209 |
| 500000 | 5000 | 13082 | 505000 | 2498194 |
| 500000 | 250000 | 18339 | 750000 | 2517447 |

Из результатов выполнения программы видно, что программа работает корректно, выполняя все поставленные задачи.

# **Вывод**

В результате выполнения работы я:

1. Освоил алгоритмы поиска в тексте образца и их реализацию на языке программирования C++;
2. Научился программировать автоматическое тестирование простых программ.

# **Исходный код программы**

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <sstream>  #include <string>  #include <vector>  #include <regex>  #include <chrono>  #include <random>  #include <ctime>  std::string read\_file(std::string filename)  {  std::ifstream f(filename);  std::stringstream buffer;  buffer << f.rdbuf();  return buffer.str();  }  void fill\_LPS(std::string pattern, std::vector<int>& lps, int& comparisons)  {  int len = 0;  lps[0] = 0;  for (int i = 1; i < lps.size(); i++)  {  comparisons += 2;  if (pattern[i] == pattern[len])  {  len++;  lps[i] = len;  }  else  {  comparisons++;  if (len != 0)  {  len = lps[len - 1];  i--;  }  else  {  lps[i] = 0;  }  }  }  }  std::vector<int> KMP(std::string text, std::string pattern, int& comparisons)  {  std::vector<int> found\_indexes;  std::vector<int> lps(pattern.length());  fill\_LPS(pattern, lps, comparisons);  int i = 0, j = 0;  comparisons++;  while ((text.length() - i) >= (pattern.length() - j))  {  comparisons++;  if (pattern[j] == text[i])  {  i++;  j++;  }  comparisons++;  if (j == pattern.length())  {  found\_indexes.push\_back(i - j);  j = lps[j - 1];  }  else if (i < text.length() && pattern[j] != text[i])  {  comparisons += 3;  if (j != 0)  {  j = lps[j - 1];  }  else  {  i++;  }  }  }  return found\_indexes;  }    bool is\_phone\_number(std::string input)  {  std::regex pattern("\\+7-\\d{3}-\\d{3}-\\d{2}-\\d{2}");  return std::regex\_match(input, pattern);  }  int main()  {  while (true)  {  std::cout << "1 - Поиск выступавших в тексте протокола заседания\n"  << "2 - Проверка строки на соответствие шаблону номера телефона\n"  << "3 - Тестирование алгоритма Кнута-Морисса-Прата\n"  << "4 - Выход\n";  int mode = 0;  std::cin >> mode;  if (mode == 1)  {  std::vector<std::string> filenames;  std::cout << "Введите имена файлов, содержащих протоколы "  << "собраний через пробел (enter - вариант по умолчанию):\n";  std::string name;  std::cin.get();  char next = std::cin.get();  while (next != '\n')  {  std::cin.putback(next);  std::cin >> name;  filenames.push\_back(name);  next = std::cin.get();  }  if (filenames.size() == 0)  {  filenames.push\_back("text10.txt");  }  std::cout << std::endl;  for (std::string filename : filenames)  {  std::string pattern = "Слушали сообщение: ";  std::string text = read\_file(filename);  int comparisons = 0;  auto start = std::chrono::high\_resolution\_clock::now();  std::vector<int> indexes = KMP(text, pattern, comparisons);  auto end = std::chrono::high\_resolution\_clock::now();  auto duration =  std::chrono::duration\_cast<std::chrono::microseconds>(end - start);  std::cout << "Время поиска: " << duration.count() << " мкс" << std::endl;  std::cout << "Количество сравнений: " << comparisons << std::endl;  std::vector<std::string> full\_names;  for (int i = 0; i < indexes.size(); i++)  {  std::string full\_name;  for (int j = indexes[i] + pattern.length();; j++)  {  if (text[j] == ' ')  {  full\_name += text.substr(j, 7);  break;  }  else  {  full\_name += text[j];  }  }  full\_names.push\_back(full\_name);  }  if (full\_names.size() == 0)  {  std::cout << "Выступавшие не найдены! (" << filename << ")\n";  }  else  {  std::cout << "Список имён выступавших (" << filename << "):\n";  for (std::string name : full\_names)  {  std::cout << '\t' << name << '\n';  }  }  std::cout << std::endl;  }  }  else if (mode == 2)  {  std::cout << "Введите строку для поверки её на соответствие "  << "шаблону номера телефона:\n";  std::string input;  std::cin >> input;  if (is\_phone\_number(input))  {  std::cout << "Строка является номером телефона!\n";  }  else  {  std::cout << "Строка не является номером телефона!\n";  }  }  else if (mode == 3)  {  std::srand(time(NULL));  std::string charset = "abcdefghijklmnopqrstuvwxyz";  int text\_length = 0;  std::cout << "Введите количество символов в тексте для поиска: ";  std::cin >> text\_length;  int wanted\_string\_length = 0;  std::cout << "Введите количество символов в искомой строке: ";  std::cin >> wanted\_string\_length;  std::string text, wanted\_string;  for (int i = 0; i < text\_length; i++)  {  text += charset[std::rand() % charset.length()];  if (i >= text\_length - wanted\_string\_length)  {  wanted\_string += text[i];  }  }  std::ofstream of("test\_file.txt");  of << text;  of.close();  int comparisons = 0;  auto start = std::chrono::high\_resolution\_clock::now();  std::vector<int> indexes = KMP(text, wanted\_string, comparisons);  auto end = std::chrono::high\_resolution\_clock::now();  auto duration =  std::chrono::duration\_cast<std::chrono::microseconds>(end - start);  std::cout << "Время поиска: " << duration.count() << " мкс" << std::endl;  std::cout << "Количество сравнений: " << comparisons << std::endl;  }  else if (mode == 4)  {  break;  }  else  {  std::cout << "Такого варианта нет!\n";  }  }  return 0;  } |