|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | 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-09-22 | Гришин А. В. |
| 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.

1. **Постановка задачи**

**Задание 1.** 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.

**Задание 2.**  Разработать алгоритм и функцию поиска образца в тексте с применением регулярных выражений для второй задачи варианта

1. Разработать регулярное выражение в соответствии с задачей варианта.
2. Разработать функцию, реализующую проверку входной строки на соответствие регулярному выражению или ее модификацию в соответствии с индивидуальным вариантом.
3. Выполнить тестирование на разработанных тестах.
4. Оформить отчет, представив результаты по пунктам задания.

|  |  |  |
| --- | --- | --- |
| 7 | 1. Строка S была записана много раз подряд, после чего из получившейся строки взяли произвольную часть строки - подстроку и передали, как входные данные. Необходимо определить минимально возможную длину исходной строки S. 2. Дан список придуманных пользователем паролей. Проверить, какие из паролей корректно составлены, т.е. удовлетворяют требованиям: в паролях могут быть только английские буквы (строчные или прописные) и цифры. Пробелы, подчеркивания и другие знаки препинания не допускаются. | Кнута-Мориса-Пратта |

# **. Решение**

* 1. **Краткая теория по теме**

Алгоритм Кнута-Мориса-Пратта (КМП) – это алгоритм для поиска подстроки в строке. Основная идея алгоритма заключается в использовании префикс-функции для определения наилучшего совпадения между префиксом подстроки и суффиксом строки. Префикс-функция для строки S – это массив P, где P[i] – это длина наибольшего собственного префикса (не равного всей строке), который также является суффиксом S[0: i].

Алгоритм КМП начинает сравнивать символы подстроки и строки слева направо. Если символы совпадают, то алгоритм переходит к следующему символу. Если символы не совпадают, то алгоритм использует префикс-функцию для определения количества символов, которые можно пропустить при следующем сравнении. Это делается путем перехода к индексу, соответствующему значению префикс-функции для предыдущего символа. Алгоритм КМП имеет линейную сложность O(n+m), где n – длина строки, а m – длина подстроки. Это делает его очень эффективным для поиска подстроки в больших текстовых файлах.

Регулярные выражения – это формальный язык, который используется для поиска и манипуляции текстом на основе шаблонов. В C++ регулярные выражения реализуются с помощью стандартной библиотеки <regex>.

* 1. **. Выполнение первого задания****.**

В первом задании необходимо разобрать алгоритм, а также подсчитать количество сравнений для успешного поиска первого вхождения образца в текст.

|  |
| --- |
| vector<int> PrefixFunction(string str)  {  int size = str.length();  vector<int> prefix(size, 0);  for (int i = 1; i < size; i++)  {  int j = prefix[i - 1];  while (j > 0 && str[i] != str[j])  {  j = prefix[j - 1];  }  if (str[i] == str[j])  {  j++;  }  prefix[i] = j;  }  return prefix;  }  int FindMinimumLength(const string& text, const string& subStr, int& comp) {  int textSize = text.length();  int subSize = subStr.length();  vector<int> prefix = PrefixFunction(subStr);  int j = 0;  for (int i = 0; i < textSize; i++) {  while (j > 0 && text[i] != subStr[j]) {  j = prefix[j - 1];  comp++;  }  if (text[i] == subStr[j]) {  j++;  comp++;  }  if (j == subSize) {  return i - j + 1;  }  }  return textSize / 5;  } |

Данный код представляет функцию FindMinimumLength, которая выполняет поиск первого вхождения подстроки subStr в строке text.

Основной алгоритм поиска базируется на использовании префикс-функции, реализованной в функции PrefixFunction. Префикс-функция вычисляет значения префиксов для каждого символа в строке subStr.

Функция FindMinimumLength принимает текст text, подстроку subStr и ссылку на переменную comp, которая используется для подсчета количества сравнений символов.

В начале функции, определяются размеры текста и подстроки. Затем вызывается функция PrefixFunction, которая вычисляет массив значений префикс-функции для подстроки subStr. Далее идет цикл, который выполняет обход текста. На каждой итерации цикла проверяется, совпадает ли символ в текущей позиции текста с символом в текущей позиции подстроки. Если символы не совпадают, значение j уменьшается до значения префикс-функции для j-1. Если символы совпадают, значение j увеличивается на 1. Если значение j становится равным длине подстроки subStr, это означает, что подстрока найдена, и функция возвращает индекс начала подстроки в тексте (i - j + 1).

Если обход текста завершается без нахождения подстроки, функция возвращает значение, равное длине всей строки text. Это условие используется в качестве индикатора, что подстрока не найдена.

Во время обхода текста, каждое сравнение символов увеличивает значение переменной comp, которая используется для подсчета количества сравнений.

* 1. **Выполнение второго задания.**

Во втором задании необходимо разработать алгоритм и функцию проверки пароля на валидность.

|  |
| --- |
| bool IsPasswordValid(const string& password)  {  for (char ch : password)  {  if (!((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9')))  {  return false;  }  }  return true;  }  void CheckPasswords(const string& fileName)  {  ifstream inputFile(fileName);  string password;  vector<string> validPasswords;  while (inputFile >> password)  {  if (IsPasswordValid(password))  {  validPasswords.push\_back(password);  }  }  inputFile.close();  cout << "Корректные пароли:" << endl;  for (const string& validPassword : validPasswords)  {  cout << validPassword << endl;  }  } |

Данная функция IsPasswordValid проверяет, является ли пароль, заданный строкой password, валидным. Функция перебирает каждый символ ch в пароле и проверяет, что символ является буквой (маленькой или большой) или цифрой. Если символ не удовлетворяет этому условию, функция возвращает false, что означает, что пароль невалидный. Если все символы пароля проходят проверку, функция возвращает true, что означает, что пароль валидный. Функция CheckPasswords открывает файл с именем fileName для чтения. Затем она создает переменную password типа string и вектор validPasswords для хранения корректных паролей. В цикле while считывается каждое слово (пароль) из файла inputFile и проверяется его валидность с помощью вызова функции IsPasswordValid. Если пароль валидный, он добавляется в вектор validPasswords. После завершения цикла while, файл inputFile закрывается. Затем функция выводит на экран строку "Корректные пароли:" и в цикле выводит каждый корректный пароль из вектора validPasswords на отдельной строке.

* 1. **Описание работы программы и её интерфейса**

Для первого задания изначально создаются и заполняются 5 файлов с текстом от 2000 до 10000 символов. Далее пользователя просят ввести строку, поиск которой осуществляется в текстах с подсчетом сравнений.

Для второй задачи изначально создаётся файл со списком различных паролей, который проверяется на валидность.

Ниже представлено меню выбора с демонстрацией выполнения второго задания (рис. 1).

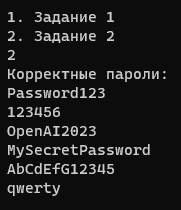


Рисунок 1. Интерфейс работы программы

1. **Тестирование**

Протестируем работу первого здания на файлах с текстом размера 2000, 4000, 5000, 7000, 8000 с учетом успешного и безуспешного поиска строки в тексте (рис. 2). Для этого поместим некоторую строку в середину текстов, за основу которых был взят текст “Lorem ipsum”.

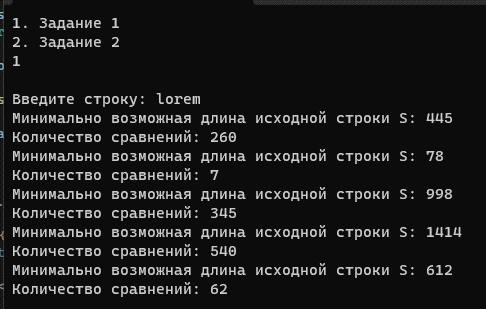


Рисунок 2. Тестирование работы первого задания с успешным поиском

Код работает исправно. По результатам тестов построим таблицы для поиска.

Таблица 1. Сводная таблица результатов

|  |  |  |  |
| --- | --- | --- | --- |
| **n** | **T(n)=N+M** | **Та=f(N+M)** | **Тэ=C** |
| 2000 | 2005 | 2005 | 260 |
| 4000 | 4005 | 4005 | 7 |
| 5000 | 5005 | 5005 | 345 |
| 7000 | 7005 | 7005 | 540 |
| 8000 | 8005 | 8005 | 62 |

Протестируем работу второго задания. Для этого возьмем список паролей в текстовом файле (рис. 3).

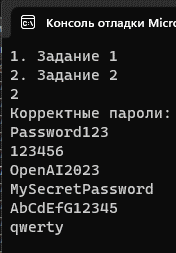


Рисунок 3. Тестирование работы второго задания

Тестирование второго задания прошло успешно. Все тесты заданий пройдены успешно.

1. **Вывод**

В результате выполнения работы я получил практический опыт, знания и навыки применения алгоритмов поиска в тексте подстрок.

1. **Исходный код программы**

Исходный код файла PractiseTask10.cpp

|  |
| --- |
| #include <iostream>  #include <fstream>  #include <string>  #include <vector>  #include <regex>  using namespace std;  vector<int> PrefixFunction(string str)  {  int size = str.length();  vector<int> prefix(size, 0);  for (int i = 1; i < size; i++)  {  int j = prefix[i - 1];  while (j > 0 && str[i] != str[j])  {  j = prefix[j - 1];  }  if (str[i] == str[j])  {  j++;  }  prefix[i] = j;  }  return prefix;  }  int FindMinimumLength(const string& text, const string& subStr, int& comp) {  int textSize = text.length();  int subSize = subStr.length();  vector<int> prefix = PrefixFunction(subStr);  int j = 0;  for (int i = 0; i < textSize; i++) {  while (j > 0 && text[i] != subStr[j]) {  j = prefix[j - 1];  comp++;  }  if (text[i] == subStr[j]) {  j++;  comp++;  }  if (j == subSize) {  return i - j + 1;  }  }  return textSize / 5;  }  bool IsPasswordValid(const string& password)  {  for (char ch : password)  {  if (!((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z') || (ch >= '0' && ch <= '9')))  {  return false;  }  }  return true;  }  void CheckPasswords(const string& fileName)  {  ifstream inputFile(fileName);  string password;  vector<string> validPasswords;  while (inputFile >> password)  {  if (IsPasswordValid(password))  {  validPasswords.push\_back(password);  }  }  inputFile.close();  cout << "Корректные пароли:" << endl;  for (const string& validPassword : validPasswords)  {  cout << validPassword << endl;  }  }  int main()  {  setlocale(LC\_ALL, "rus");  cout << "1. Задание 1" << endl;  cout << "2. Задание 2" << endl;  int choice{};  cin >> choice;  switch (choice)  {  case 1:  {  string name = "words";  string subStr;  cout << "\nВведите строку: ";  cin >> subStr;  for (int i = 1; i <= 5; i++)  {  int comp = 0;  string task = name + to\_string(i);  ifstream inputFile(task + ".txt");  string text;  string sent;  while (getline(inputFile, sent))  {  text += sent;  }  inputFile.close();  int minLen = FindMinimumLength(text, subStr, comp);  cout << "Минимально возможная длина исходной строки S: " << minLen << endl;  cout << "Количество сравнений: " << comp << endl;  }  break;  }  case 2:  {  string path = "passwords.txt";  CheckPasswords(path);  }  }  return 0;  } |