ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| доцент |  |  |  | В. А. Кузнецов |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 4.1 |
| ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ |
| по курсу: |
| ОСНОВЫ ПРОГРАММИРОВАНИЯ |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ гр. № | 4326 |  |  |  | Г. С. Томчук |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2024

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#__RefHeading___Toc801_316780651)

[2 Схема алгоритма решения 4](#__RefHeading___Toc809_316780651)

[3 Полное описание реализованной функции 5](#__RefHeading___Toc807_316780651)

[4 Листинг программы 6](#__RefHeading___Toc805_316780651)

[5 Результаты тестирования программы 7](#__RefHeading___Toc803_316780651)

[6 Результаты измерения времени работы и оценки сложности алгоритма 8](#__RefHeading___Toc801_3167806512)

1. Постановка задачи

Задача: реализовать алгоритм на языке C/С++, выполняющий поставленную задачу. Вариант задания, пример входных и выходных данных представлен в таблице 1. Глобальные параметры использовать запрещено; допустимо использование дополнительных функций.

* Разработанный алгоритм должен быть реализован в виде цельной программной функции (или нескольких функций) так, чтобы мог быть многократно применен с различными исходными данными и при этом не включал команды, не относящиеся к решаемой задаче, например, ввод и вывод исходных данных на консоль или в файл.
* Произвести теоретическую оценку количества используемых операций разработанного алгоритма.
* Произвести экспериментальную проверку времени работы разработанного алгоритма, определив его класс сложности для среднего случая. Измерить среднее время для *Test\_Count* повторений при различных размерностях входных данных.

Таблица 1 – Вариант

|  |  |  |  |
| --- | --- | --- | --- |
| N | Текст задания | Вход | Выход |
| 4 | Алгоритм нахождения всех вхождений строки A в строку B. Сложность определяется размером строк NA, NB. | A = “de”  B = “abcdede” | 3, 5 |

1. Схема алгоритма решения

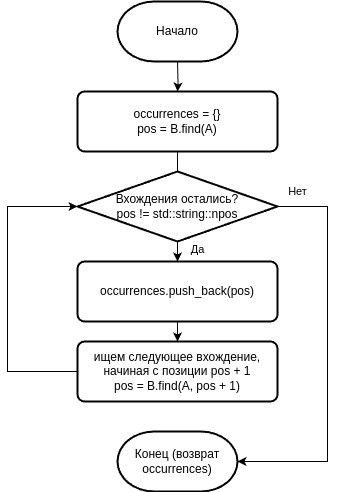


Рисунок 1 – Блок-схема алгоритма

1. Полное описание реализованной функции

Функция find\_occurrences находит индексы всех вхождений строки A в строку B. Принимает следующие аргументы:

1. A (const std::string&): подстрока для поиска.
2. B (const std::string&): строка, в которой осуществляется поиск.

Возвращает вектор std::vector<int> с индексами всех вхождений подстроки A в строку B.. Работа функции происходит следующим образом:

1. Создается пустой вектор occurrences для хранения индексов вхождений.
2. Используется метод B.find(A) для поиска первого вхождения подстроки A в строке B.
3. Пока find не вернет std::string::npos, что значит, что больше вхождений нет, текущая позиция pos добавляется в вектор occurrences.
4. Поиск продолжается с позиции pos + 1, чтобы найти следующие вхождения.
5. После завершения поиска возвращается вектор occurrences с индексами всех найденных вхождений.
6. Листинг программы

Листинг 1

#include <iostream>  
#include <string>  
#include <vector>  
  
std::vector<int> find\_occurrences(const std::string &A, const std::string &B) {  
 std::vector<int> occurrences;  
 size\_t pos = B.find(A); // находим первое вхождение  
  
 while (pos != std::string::npos) {  
 occurrences.push\_back(pos); // добавляем индекс в вектор  
 pos = B.find(A, pos + 1); // ищем следующее вхождение, начиная с позиции pos + 1  
 }  
  
 return occurrences;  
}  
  
int main() {  
 std::string A;  
 std::string B;  
  
 std::cout << "A: ";  
 std::cin >> A;  
  
 std::cout << "B: ";  
 std::cin >> B;  
  
 std::vector<int> occurrences = find\_occurrences(A, B);  
  
 for (int index : occurrences) {  
 std::cout << index << " ";  
 }  
  
 return 0;  
}

1. Результаты тестирования программы

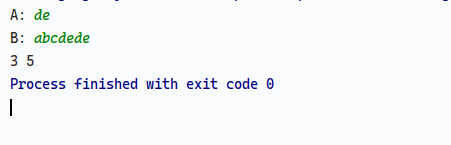


Рисунок 2

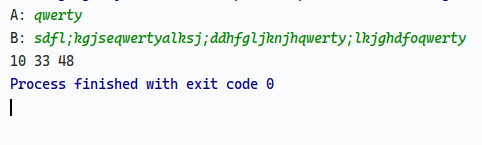


Рисунок 3

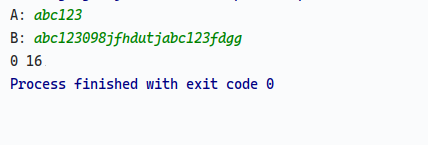


Рисунок 4

1. Результаты измерения времени работы и оценки сложности алгоритма

**Теоретическая оценка количества операций.** Алгоритм использует метод std::string::find для поиска вхождений строки A в строку B. Для большинства реализаций сложность метода find будет O(n \* m) в худшем случае. Однако в среднем случае, благодаря оптимизациям и особенностям строк, сложность может быть ближе к O(n), где n — длина строки B, а m — длина подстроки A.

Метод find вызывается несколько раз. Если в строке B есть k вхождений строки A, то сложность алгоритма будет O(k \* (n \* m)) в худшем случае.

Если подстрока A не встречается в строке B, алгоритм завершится после одной проверки, что даст сложность O(n) в лучшем случае.

Однако в среднем случае, если строки B и A разной длины и символы распределены случайным образом, сложность будет ближе к O(n), так как метод find будет делать меньше сравнений при поиске всех вхождений.

Таблица 2 – Количество операций в коде алгоритма

|  |  |
| --- | --- |
| Операция | Количество |
| Нахождение вхождения find | k+1 |
| Сравнение != | k+1 |
| Сравнение == (метод find) | N |

**Экспериментальная проверка времени работы.** Для экспериментальной проверки времени работы были написаны две дополнительные функции, который замеряют время выполнения алгоритма для случайных строк А фиксированной длины и строк B различной длины. Для измерения времени была использована библиотека <chrono>. Чтобы получить среднее время выполнения, функция вызывается test\_count=1000 раз.

Листинг 2

std::string generate\_random\_string(size\_t length) {  
 const char charset[] = "abcdefghijklmnopqrstuvwxyz";  
 std::default\_random\_engine generator;  
 std::uniform\_int\_distribution<int> distribution(0, sizeof(charset) - 2);  
  
 std::string random\_string;

Продолжение листинга 2

for (size\_t i = 0; i < length; ++i) {  
 random\_string += charset[distribution(generator)];  
 }  
 return random\_string;  
}  
  
void test\_find\_occurrences() {  
 const int test\_count = 1000;  
 std::vector<int> sizes = {10000, 100000, 1000000, 10000000};  
  
 for (int size : sizes) {  
 std::string A = generate\_random\_string(5);  
 std::string B = generate\_random\_string(size);  
  
 auto total\_duration = 0.0;  
  
 for (int i = 0; i < test\_count; ++i) {  
 auto start = std::chrono::high\_resolution\_clock::now();  
 find\_occurrences(A, B);  
 auto end = std::chrono::high\_resolution\_clock::now();  
 std::chrono::duration<double> duration = end - start;  
 total\_duration += duration.count();  
 }  
  
 double average\_duration = total\_duration / test\_count;  
 std::cout << "Среднее время выполнения при B=" << size << ": " << std::fixed << std::setprecision(12)  
 << average\_duration << " с" << std::endl;  
 }  
}

Результаты измерения среднего времени работы представлены на рис. 5. Отчетливо видно, что время выполнения алгоритма увеличивается примерно пропорционально длине строки B (в 10 раз в данном случае).

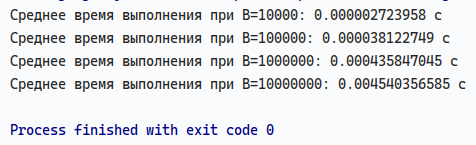


Рисунок 5