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

Задача: реализовать алгоритм на языке C/C++, 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 N_A, N_B.	A = “de” B = “abcdede”	3, 5

2 Схема алгоритма решения

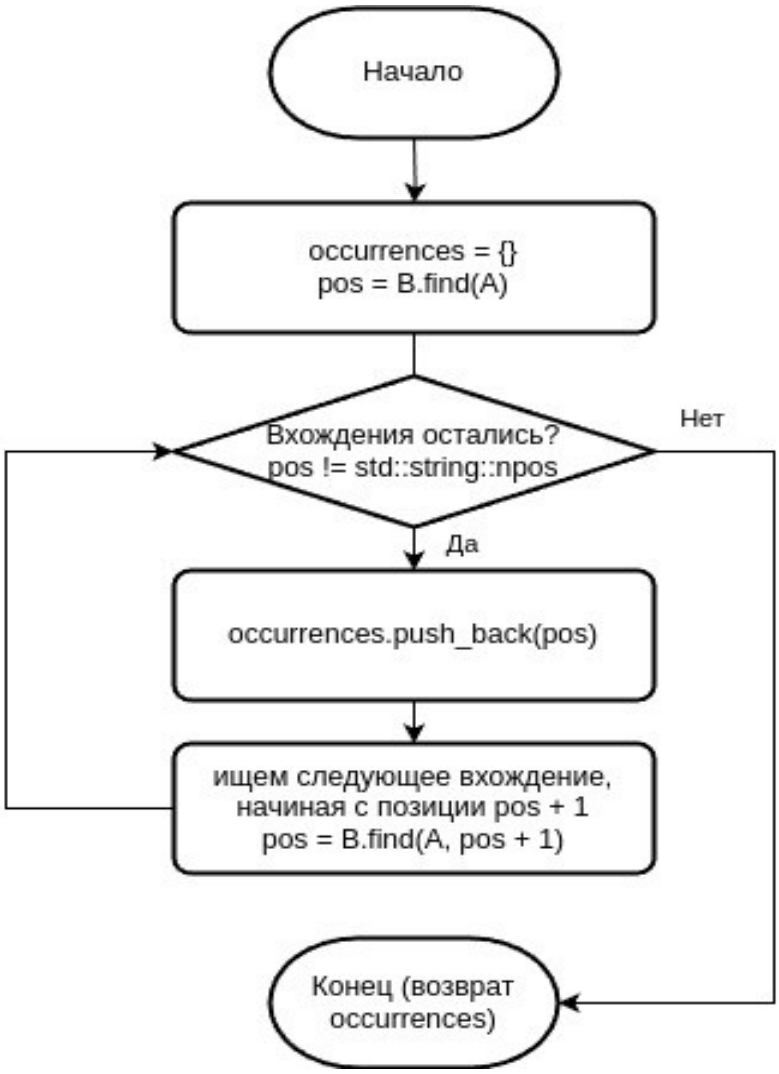

Рисунок 1 – Блок-схема алгоритма

3 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.

4 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;
}
```

5 Результаты тестирования программы

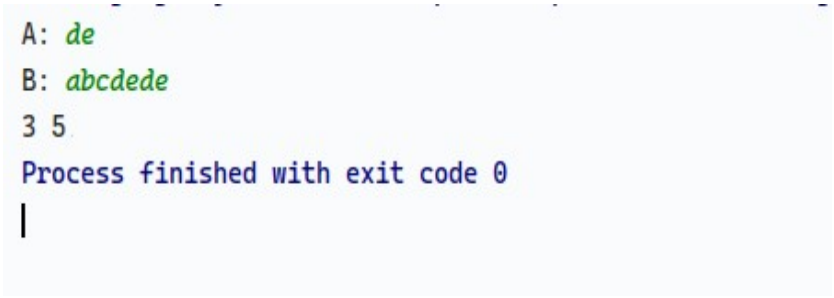

Рисунок 2

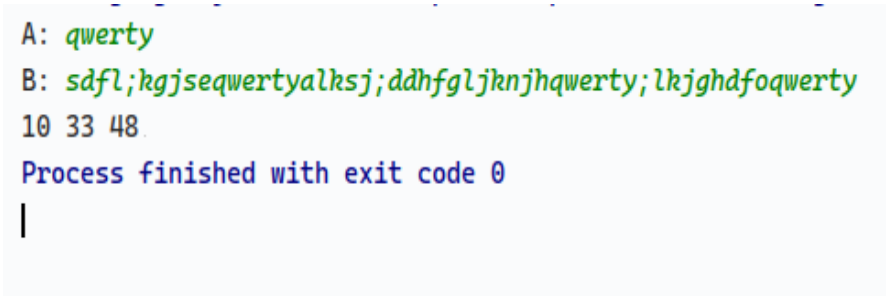

Рисунок 3

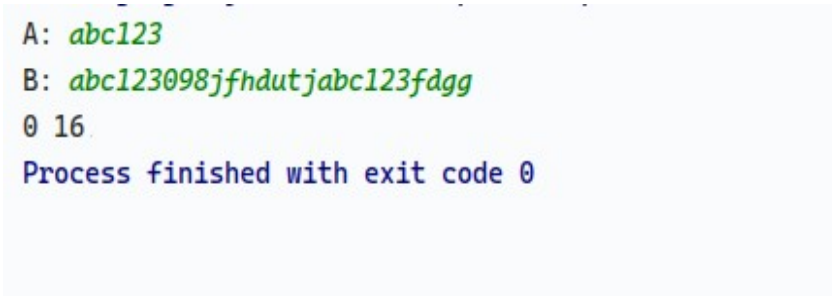

Рисунок 4

6 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 A 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 << " c" << 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).

```
Среднее время выполнения при B=10000: 0.000002723958 c
Среднее время выполнения при B=100000: 0.000038122749 c
Среднее время выполнения при B=1000000: 0.000435847045 c
Среднее время выполнения при B=10000000: 0.004540356585 c

Process finished with exit code 0
```

Рисунок 5