Бюджетное учреждение высшего образования   
Ханты-Мансийского автономного округа   
«Сургутский государственный университет»

Политехнический институт

Кафедра автоматики и компьютерных систем

**Отчет**

по лабораторной работе № 4

по дисциплине «Алгоритмы и структуры данных»

Выполнил: студент группы 609-21,

Шумилов И.Д.

Принял: старший преподаватель кафедры АиКС

Назаров Е.В.

2024 г.

**Цель работы:** изучить основные принципы работы алгоритмов поиска подстрок, исследовать их свойства, закрепить навыки структурного программирования.

**Общее задание на работу:**

1. Разработать функцию, реализующую любой из алгоритмов поиска подстрок.
2. Исследовать алгоритмы поиска реализованной функции и функции стандартной библиотеки strstr: построить и сравнить зависимости среднего времени выполнения от размера текста и размера образца. При исследовании алгоритмов обратить особое внимание на формирование текста, в котором осуществляется поиск
3. Составить отчет, в котором привести графики полученных зависимостей, анализ свойств алгоритмов и выводы по работе.

**Индивидуальное задание на лабораторную работу:**

Разработать алгоритм Рабина-Карпова.

**Код программы**

Листинг 1. Функции поиска Рабина-Карпова и генерации случайной строки:

// Алгоритм поиска подстрок Рабина-Карпа

int search(std::string sub, std::string str, int base = 256, int mod = INT\_MAX) {

int m = sub.length();

int n = str.length();

int subHash = 0;

int strHash = 0;

int power = 1;

for (int i = 0; i < m; i++)

power = (power \* base) % mod;

for (int i = 0; i < m; i++) {

subHash = (base \* subHash + sub[i]) % mod;

strHash = (base \* strHash + str[i]) % mod;

}

for (int i = 0; i <= n - m; i++) {

if (subHash == strHash) {

int j;

for (j = 0; j < m; j++) {

if (str[i + j] != sub[j]) {

break;

}

}

if (j == m)

return i;

}

if (i < n - m) {

strHash = (base \* (strHash - str[i] \* power) + str[i + m]) % mod;

if (strHash < 0)

strHash = (strHash + mod);

}

}

return -1;

}

// Генерация случайной строки

std::string randStr(int n) {

std::string res;

for (int i = 0; i < n; i++)

res += 'a' + rand() % 26;

return res;

}

// Генерация подстроки

std::string getSubstring(std::string str, int subSize) {

int margin = str.length() / 2;

std::string res;

for (int i = 0; i < subSize; i++)

res += str[margin + i];

return res;

}

**Анализ алгоритмов поиска подстрок**

Таблица 1 – время выполнения алгоритмов при постоянном размере подстроки равном 100 символов

|  |  |  |
| --- | --- | --- |
| Размер строки | Время выполнения (мс) | |
| Рабина-Карпа | strstr |
| 100000 | 0.7104 | 0.0956 |
| 200000 | 0.9431 | 0.1222 |
| 300000 | 1.2764 | 0.1797 |
| 400000 | 1.3472 | 0.2116 |
| 500000 | 1.5032 | 0.281 |
| 600000 | 1.5668 | 0.3209 |
| 700000 | 1.6592 | 0.4229 |
| 800000 | 1.8496 | 0.4212 |
| 900000 | 2.3314 | 0.501 |
| 1000000 | 2.4852 | 0.5465 |

­

Рис. 1 - график временной зависимости алгоритмов при постоянном размере подстроки в 100 символов

Таблица 2 – время выполнения алгоритмов при постоянном размере строки равном 1000000 символов

|  |  |  |
| --- | --- | --- |
| Размер подстроки | Время выполнения (мс) | |
| Рабина-Карпа | strstr |
| 10 | 2.2024 | 0.4945 |
| 20 | 2.2506 | 0.5881 |
| 30 | 2.3324 | 0.598 |
| 40 | 2.3954 | 0.6151 |
| 50 | 2.4696 | 0.6713 |
| 60 | 2.5479 | 0.7013 |
| 70 | 2.6528 | 0.8104 |
| 80 | 2.8358 | 1.0308 |
| 90 | 2.9938 | 1.4181 |
| 100 | 3.0577 | 1.5662 |

Рис. 2 – график временной зависимости алгоритмов при постоянном размере строки в 1000000

**Анализ свойств алгоритмов поиска подстрок**

Таблица 3 – сравнение алгоритмов Рабина-Карпа и strstr

|  |  |  |
| --- | --- | --- |
| Признак сравнения | Алгоритм | |
| Рабина-Карпа | strstr |
| **Скорость работы** | имеет линейную сложность *O*(*n*+*m*), где *n* - длина текста, а *m* - длина образца. Он работает эффективно для поиска нескольких вхождений образца в тексте | в худшем случае имеет сложность 𝑂(𝑛×𝑚). Он выполняет поиск только первого вхождения образца в тексте |
| **Возвращаемое значение** | часто используется, когда нужно найти все вхождения образца в тексте, или когда необходимо выполнять поиск сразу по нескольким образцам | используется для поиска первого вхождения образца в тексте |
| **Требования к памяти** | требует дополнительной памяти для хранения хэш-значений и предварительной обработки | не требует дополнительной памяти |
| **Чувствительность к размеру алфавита** | работает эффективно независимо от размера алфавита | эффективен для ASCII и UNICODE, но может иметь ограничения при работе с другими алфавитами |

**Вывод:** был проанализирован и исследован алгоритм Рабина-Карпова. Также была исследована работа функции strstr стандартной библиотеки C++.

Были изучены основные принципы работы алгоритмов поиска подстрок, исследованы их свойства, закреплены навыки структурного программирования.