

Министерство образования Республики Беларусь

Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы алгоритмизации и программирования

Отчет
по учебной (ознакомительной) практике
на тему

Алгоритм Рабина-Карпа

Сроки прохождения практики: с 22.06.2022 по 05.07.2022

Студент группы 150502

Альхимович Н.Г.

Руководитель:

Луцик Ю.А.

Минск 2022

Содержание

<i>Введение.....</i>	<i>3</i>
<i>Постановка задачи</i>	<i>4</i>
<i>Описание алгоритма</i>	<i>4</i>
<i>Код программы</i>	<i>5</i>
<i>Примеры работы программы.....</i>	<i>11</i>
<i>Заключение</i>	<i>12</i>
<i>Список использованных источников</i>	<i>13</i>

Введение

Разработанный Ричардом Карпом и Майклом Рабином алгоритм Рабина-Карпа – это алгоритм поиска строки, который использует хеширование для обнаружения совпадений между заданным шаблоном и текстом.

Простейший вариант реализации описанной задачи, так называемый наивный алгоритм, заключается в посимвольном сравнении части строки с шаблоном, что приводит к далеко не идеальной сложности времени исполнения $O(n*m)$, где n – длина текста, а m – длина шаблона.

Алгоритм Рабина-Карпа совершенствует этот подход благодаря тому, что сравнение хешей двух строк выполняется за линейное время, а переход к посимвольному сравнению происходит только в случае совпадения хешей очередного «окна» и заданной подстроки. Таким образом, алгоритм показывает лучшее время исполнения $O(n+m)$.

Практическая реализации оптимизированных алгоритмов поиска заданного шаблона в тексте не перестает быть актуальной благодаря большому разнообразию сфер применения: анализ данных, обнаружение плагиата и др. Хотя представленный метод и не является совершенным в силу риска возникновения коллизий, но простота реализации и ускорение времени работы делает его весьма полезным и оптимальным вариантом для поиска подстроки в строке.

Постановка задачи

Реализовать алгоритм Рабина-Карпа, написав код программы на языке Си. Разработать функцию, вычисляющую хеш каждого «окна» строки.

Описание алгоритма

Основные этапы алгоритма:

1. Вычисляется хеш шаблона строки.
2. Вычисляется хеш подстроки в тексте строки, начиная с индекса 0 и до $m-1$.
3. Сравнивается хеш подстроки текста с хешем шаблона.
 - а. Если они совпадают, то сравниваются отдельные символы для выявления точного совпадения двух строк.
 - б. Если они не совпадают, то окно подстроки сдвигается путём увеличения индекса и повторяется третий пункт для вычисления хеша следующих m символов, пока не будут пройдены все n символов.

Необходимо отметить, что при использовании простейшей хеш-функции, которая при продвижении по строке каждый раз вычисляет с нуля хеш нового окна, производительность алгоритма увеличится незначительно или не увеличится вовсе. Для повышения эффективности алгоритма при его реализации используется скользящая хеш-функция, которая считает новый хеш, основываясь на предыдущем с изменением пары значений (убирая из подсчета первый символ предыдущего окна и добавляя новый). Чтобы сопоставить две строки, они, по сути, превращаются в числа простым переводом в произвольную систему счисления, где позиция символа – это разряд, его код – это значение разряда, а мощность алфавита – это, например, количество символов в кодировке:

$H = (H_p - S_p \times c^{m-1}) \times c + S_n$, где H_p – предыдущий хеш, S_p и S_n – предыдущий и новый символы соответственно, c – константа.

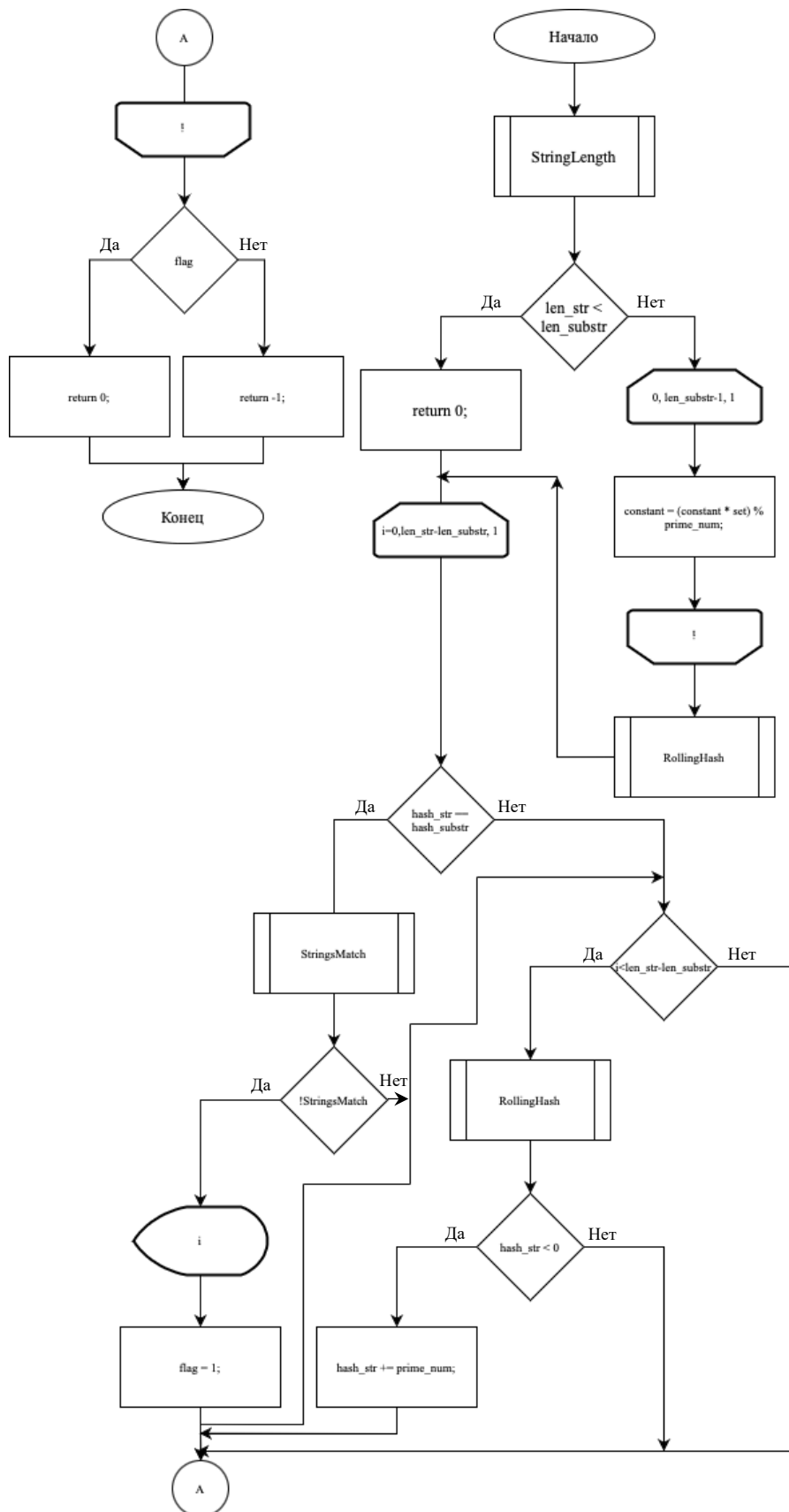
Результат всех вычислений должен браться по модулю, т.е. с использованием остатка от деления, во избежание появления больших значений хеша и целочисленных переполнений. Для этого обычно выбирается простое число. Причем, чем меньше его значение, тем выше вероятность ложных срабатываний – хеш-коллизий.

Для сравнения эффективности двух алгоритмов – наивного и Рабина-Карпа можно рассмотреть поиск подстроки «Karp» в строке «Rabin-Karp algorithm».

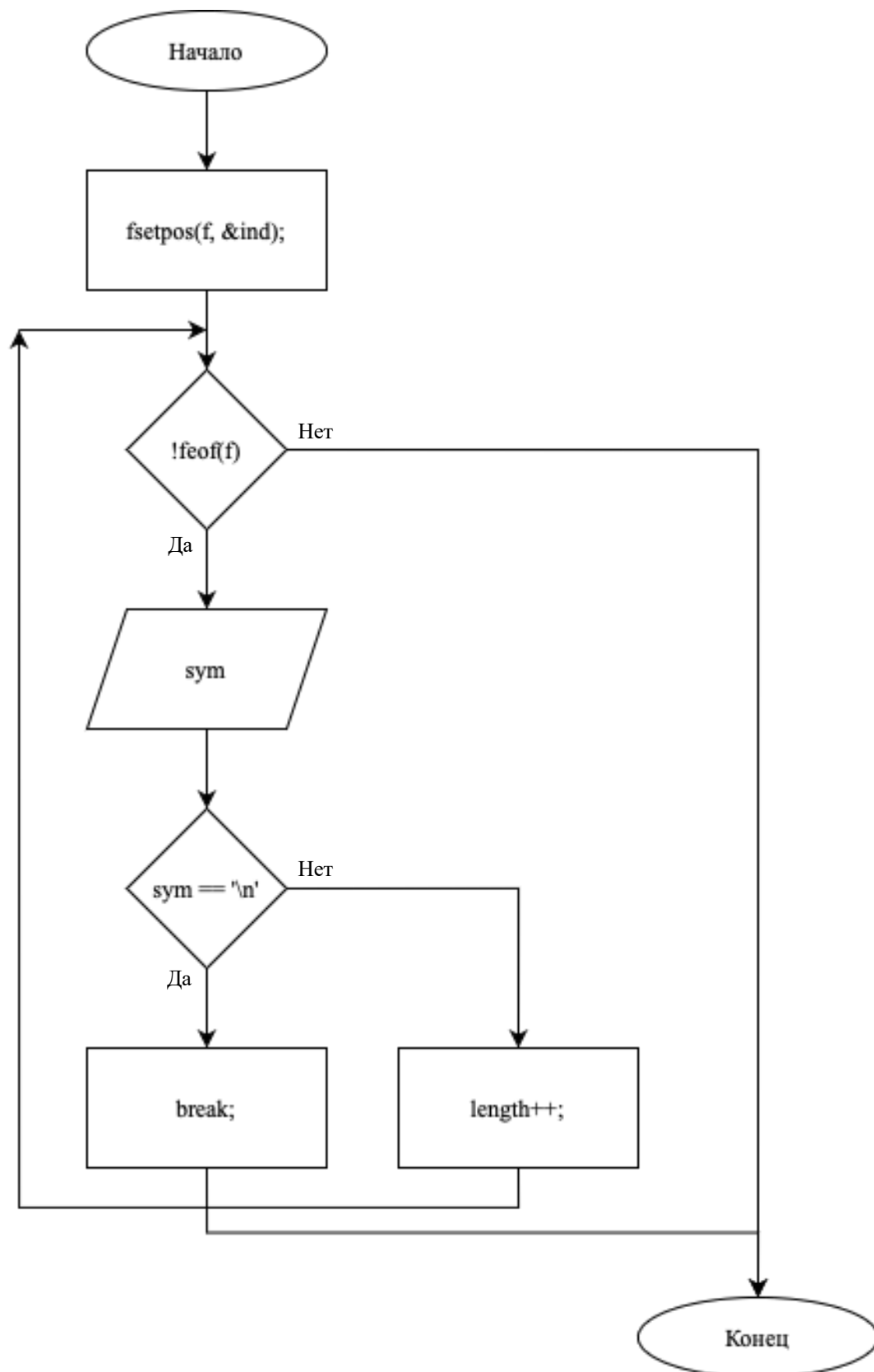
В первом случае время работы программы составляет 0,000364 с. Во втором случае – 0,000242 с. Таким образом, можно заметить, что получился заметный выигрыш во времени при использовании алгоритма Рабина-Карпа, а именно в 1,5 раза.

Блок-схемы функций

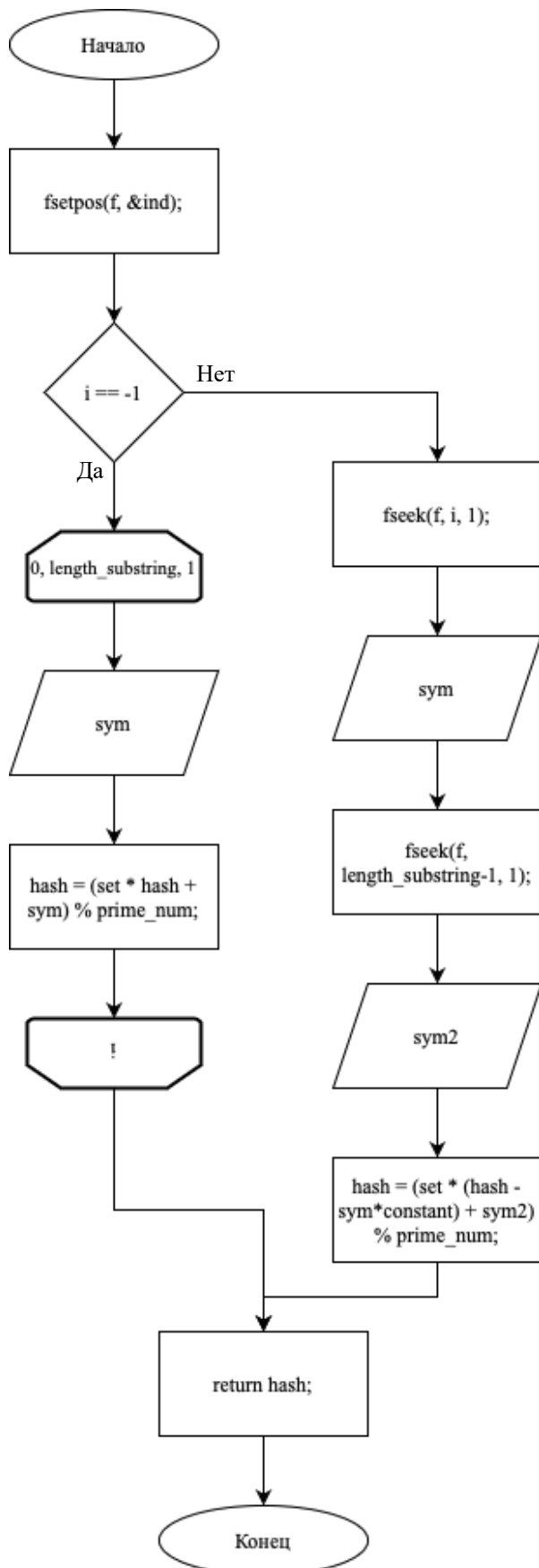
RabinKarpSearch:



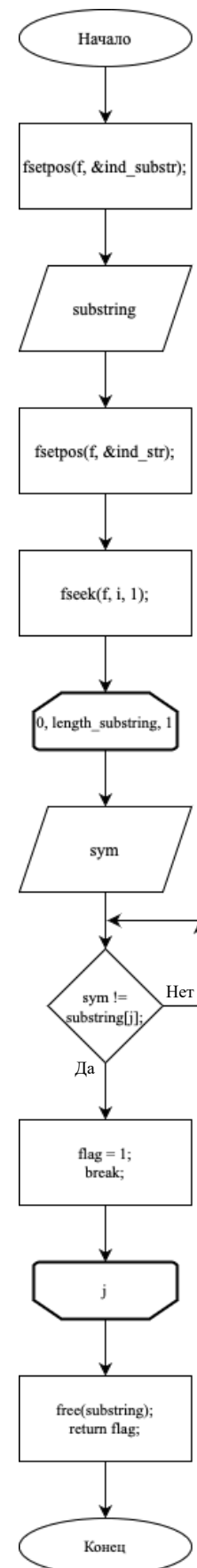
StringLength:



RollingHash:



StringsMatch:



Код программы

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define set 256
int prime_num = 101;

int main()
{
    clock_t time;
    time = clock(); //функция нахождения времени работы программы

    FILE *f;
    if(!(f = fopen("strings.txt", "rt"))) //проверка доступа к файлу
    {
        printf("\nНевозможно открыть файл");
        fclose(f);
        return -1;
    }

    fpos_t begin = 0, ind_str = 0, ind_substr = 0;
    char sym;
    int RabinKarpSearch(FILE *f, fpos_t ind_str, fpos_t ind); //прототип функции, реализующей поиск
    подстроки в строке

    if(getc(f) == EOF) //проверка, не пустой ли файл
    {
        printf("Файл пустой\n");
        return -1;
    }

    fsetpos(f, &begin);
    ind_substr = begin; //запоминание места начала подстроки

    while(!(feof(f)))
    {
        fscanf(f, "%c", &sym);
        if(sym == '\n') //если достигнут конец подстроки
        {
            fgetpos(f, &ind_str); //запоминается позиция начала строки
            break;
        }
    }
    fsetpos(f, &begin);

    if((RabinKarpSearch(f, ind_str, ind_substr)) == -1) printf("\nТребуемая подстрока не найдена");

    fclose(f);

    time = clock() - time;
    printf("\n\nВремя работы программы, реализующей алгоритм Рабина-Карпа: %f секунд\n\n",
    ((double)time)/CLOCKS_PER_SEC);

    return 0;
}

int RabinKarpSearch(FILE *f, fpos_t ind_str, fpos_t ind_substr)
```



```

{
    unsigned long int len_str = 0, len_substr = 0;
    int constant = 1, hash_substr = 0, hash_str = 0, i = -1, j = 0, flag = 0;
    unsigned long int StringLength(FILE *f, unsigned long int length, fpos_t ind); //прототип функции,
определяющей длину строк
    int RollingHash(FILE *f, fpos_t ind, unsigned long int length_substring, int constant, int hash, int i);
//прототип функции, возвращающей значение хеша окна
    int StringsMatch(FILE *f, fpos_t ind_str, fpos_t ind_substr, unsigned long int length_substring, int i);
//прототип функции, осуществляющей посимвольное сравнение строк

    len_str = StringLength(f, len_str, ind_str);
    len_substr = StringLength(f, len_substr, ind_substr);

    if(len_str < len_substr)
    {
        printf("\nПроизошла ошибка. Длина требуемой подстроки превышает длину строки");
        return 0;
    }

    for(j=0; j<len_substr-1; j++)
        constant = (constant * set) % prime_num;

    hash_substr = RollingHash(f, ind_substr, len_substr, constant, hash_substr, i); //нахождение хеша
подстроки и всей строки
    hash_str = RollingHash(f, ind_str, len_substr, constant, hash_str, i);

    for(i=0; i<=len_str-len_substr; i++) //пооконное движение по строке
    {
        if(hash_str == hash_substr) //если найдено совпадение хешей
            if(!StringsMatch(f, ind_str, ind_substr, len_substr, i)) //посимвольное сравнение окна и
шаблона
            {
                printf("\nИндекс вхождения подстроки: %d", i);
                flag = 1; //поднятие флага, если найдено хотя бы одно вхождение
            }

        if(i<len_str-len_substr)
        {
            hash_str = RollingHash(f, ind_str, len_substr, constant, hash_str, i); //вычисление
очередного хеша окна
            if(hash_str < 0)
                hash_str += prime_num;
        }
    }

    if(flag) return 0; //если искомая подстрока была найдена в строке
    else return -1;
}

unsigned long int StringLength(FILE *f, unsigned long int length, fpos_t ind)
{
    fsetpos(f, &ind);
    char sym;

    while(!feof(f)) //пока не достигнут конец файла
    {
        fscanf(f, "%c", &sym);
        if(sym == '\n') //если достигнут конец подстроки
            break;
        length++;
    }
}

```

```

    }
    return length;
}

int RollingHash(FILE *f, fpos_t ind, unsigned long int length_substring, int constant, int hash, int i)
{
    fsetpos(f, &ind);
    char sym, sym2;

    if(i == -1)
        for(i=0; i<length_substring; i++)
        {
            fscanf(f, "%c", &sym);
            hash = (set * hash + sym) % prime_num;
        }
    else
    {
        fseek(f, i, 1);
        fscanf(f, "%c", &sym);
        fseek(f, length_substring-1, 1);
        fscanf(f, "%c", &sym2);
        hash = (set * (hash - sym*constant) + sym2) % prime_num;
    }

    return hash;
}

int StringsMatch(FILE *f, fpos_t ind_str, fpos_t ind_substr, unsigned long int length_substring, int i)
{
    int j, flag = 0;
    char sym, *substring;
    fsetpos(f, &ind_substr);

    if((substring = (char *) malloc(length_substring*sizeof(char))) == NULL)
    {
        printf("\nПамять не выделена");
        return -1;
    }

    fgets(substring, length_substring+1, f);
    fsetpos(f, &ind_str);
    fseek(f, i, 1);

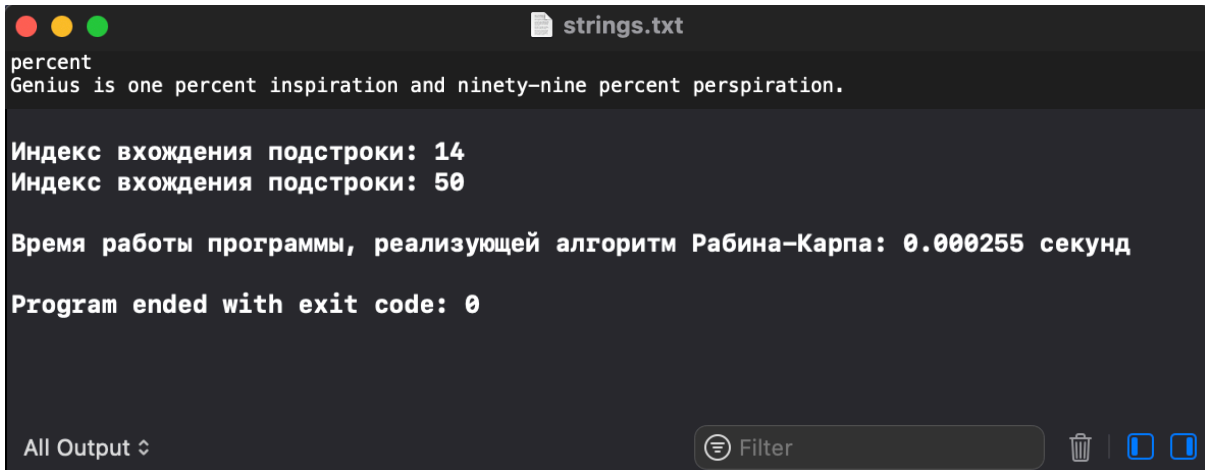
    for(j=0; j<length_substring; j++)
    {
        fscanf(f, "%c", &sym);
        if(sym != substring[j])
        {
            flag = 1;
            break;
        }
    }

    free(substring);

    return flag;
}

```

Примеры работы программы



```
strings.txt
percent
Genius is one percent inspiration and ninety-nine percent perspiration.

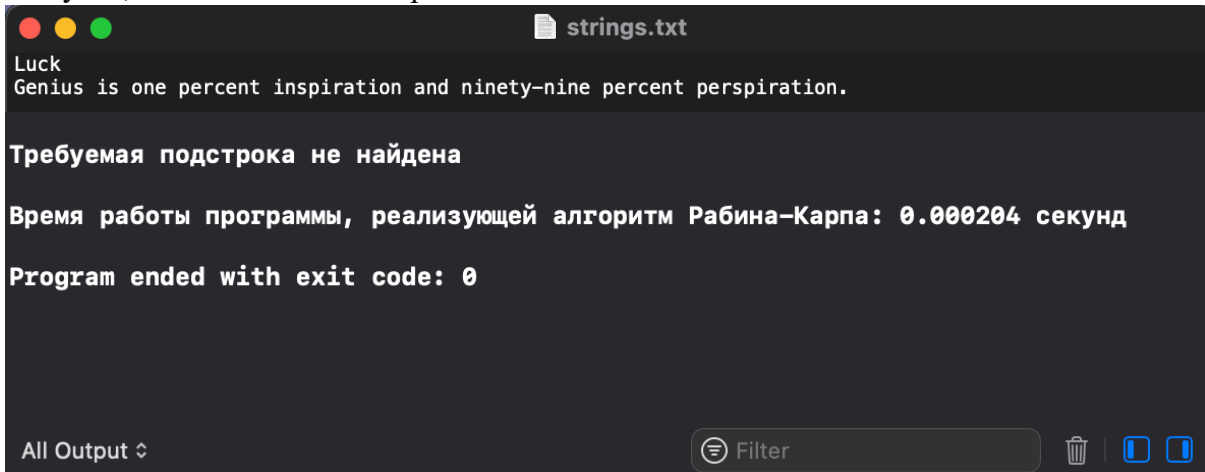
Индекс вхождения подстроки: 14
Индекс вхождения подстроки: 50

Время работы программы, реализующей алгоритм Рабина-Карпа: 0.000255 секунд

Program ended with exit code: 0
```

All Output ↕ Filter 🗑️ 📄 📄

В случае, если искомой подстроки в тексте нет:



```
strings.txt
Luck
Genius is one percent inspiration and ninety-nine percent perspiration.

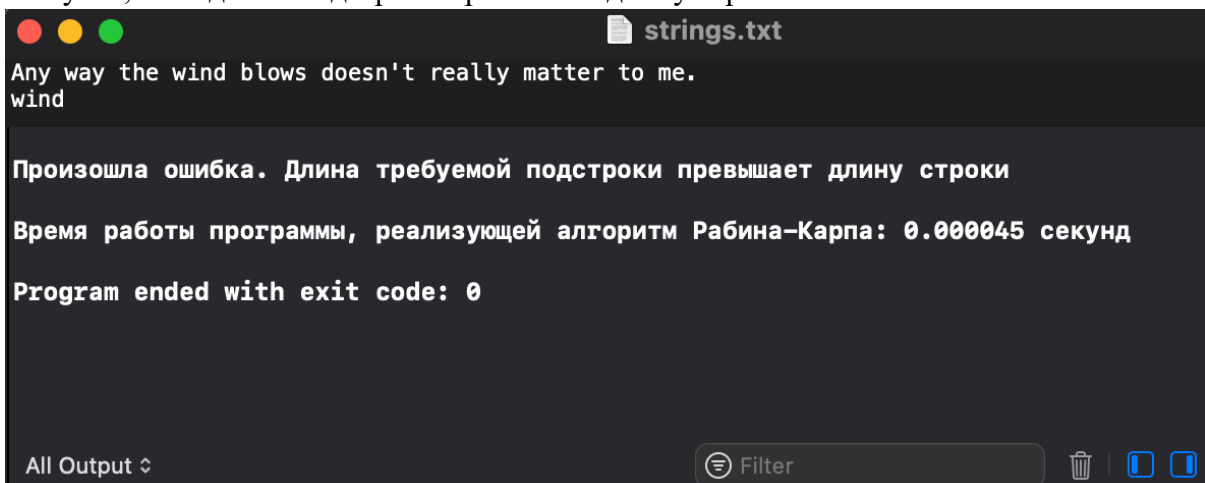
Требуемая подстрока не найдена

Время работы программы, реализующей алгоритм Рабина-Карпа: 0.000204 секунд

Program ended with exit code: 0
```

All Output ↕ Filter 🗑️ 📄 📄

В случае, если длина подстроки превышает длину строки:



```
strings.txt
Any way the wind blows doesn't really matter to me.
wind

Произошла ошибка. Длина требуемой подстроки превышает длину строки

Время работы программы, реализующей алгоритм Рабина-Карпа: 0.000045 секунд

Program ended with exit code: 0
```

All Output ↕ Filter 🗑️ 📄 📄

Заключение

Таким образом, в рамках учебной практики был изучен и практически реализован один из методов оптимизации поиска подстроки в строке – алгоритм Рабина-Карпа с применением понятия хеша и разработки функции, использующей кольцевой хеш.

Ввод данных осуществлялся через предварительно созданный текстовый файл, задействовав соответствующие функции. Отдельно были разработаны: функция, управляющая основными процессами поиска, а также функции нахождения длины подстрок внутри файла, вычисления хеша и посимвольного сравнения подстрок.

При проведении несложного сравнительного анализа удалось выяснить, что алгоритм Рабина-Карпа по сравнению с наивным алгоритмом дает выигрыш во времени по меньшей мере в полтора раза.

Необходимо отметить, что главными преимуществами изученного метода являются простота и понятность реализации, а также экспериментально подтвержденная минимизация временных затрат. К недостаткам можно отнести не 100%-ю точность срабатывания алгоритма и затруднительность работы с большими объемами данных.

Список использованных источников

- 1) Луцик Ю.А., Ковальчук А.М., Сасин Е.А. – Учебное пособие по курсу "Основы алгоритмизации и программирования". – Минск: БГУИР, 2015 г.
- 2) Строковые алгоритмы на практике. Часть 3 –Алгоритм Рабина – Карпа [Электронный ресурс]. – 2022 – Режим доступа : <https://habr.com/ru/post/662678/>.
- 3) Overview of Rabin-Karp Algorithm [Электронный ресурс]. – 2022 – Режим доступа : <https://www.baeldung.com/cs/rabin-karp-algorithm>.