ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ УФИМСКИЙ УНИВЕРСИТЕТ НАУКИ И ТЕХНОЛОГИЙ

Кафедра ВМиК

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

по предмету **«Языки программирования»**

Выполнил: студент группы МО-203б

Ярошко Е. В.

Проверил:  
ст. преподаватель

Рипатти А. В.

**Уфа 2025 г.**

**Цель работы:**

Изучить работу алгоритмов поиска подстроки в строке.

**Задачи работы:**

- освоить работу со строками в языке C++

- изучить алгоритмы поиска подстроки в строке

**Задание:**

1. Скачать книги про Гарри Поттера в формате TXT, например, отсюда

https://github.com/bobdeng/owlreader/tree/master/ERead/assets/books

Чтобы это сделать быстро, можно скачать весь репозиторий в формате ZIP отсюда

https://github.com/bobdeng/owlreader/tree/master

и затем найти все книги в каталоге ERead/assets/books

2. Реализовать следующие алгоритмы поиска подстроки в строке

- Наивный поиск: перебираем позицию начала возможного вхождения образца в текст, после чего проверяем совпадает ли нужный кусочек текста с образцом.

- Через метод find у std::string

- Через функцию strstr

- Через префикс-функцию (алгоритм Кнута-Морриса-Пратта)

- Через полиномиальное хэширование (алгоритм Рабина-Карпа)

Данные алгоритмы нужно адаптировать так, чтобы находились все вхождения образца в текст.

3. Для одной из книг про Гарри Поттера (желательно брать размером хотя бы 900КБ, текст книги загружается в программу чтением из файла) всеми реализованными алгоритмами найти количество вхождений следующих подстрок:

Harry

I dunno

What's up?

Hermione and Ron

said Professor McGonagall

и вывести на экран. Если алгоритмы реализованы корректно, они все выдадут одинаковый ответ.

4. Сделать замер времени работы для каждого алгоритма и также вывести на экран. Сравнить эффективность алгоритмов (словесно и в отчете).

**Теоретические основы:**

В рамках данной лабораторной работы реализованы несколько алгоритмов поиска подстроки в строке. Для поиска используются несколько string-переменных из книги Harry Potter and the Chamber of Secrets.

При тестировании алгоритмов было выявлено, что встроенные функции поиска в библиотеках C++ работают быстрее всего. Это связано с тем, что функции, реализуемые в библиотеках C++, работают на более низком уровне языка, тем самым выигрывая в производительности при компиляции.

Стоит отметить, что медленнее всего работает алгоритм наивного поиска (Naive Search), показывая результаты в три раза больше по времени, чем все остальные алгоритмы. Это связано с тем, что поиск производится по каждому элементу строки, что является самым неэффективным методом.

Довольно медленным способом поиска оказался и метод полиномиального хеширования (Hash Search). Связано это, скорее всего, с тем, что изначально строка и подстрока переводятся в хэш, что также отнимает время для поиска. Тем не менее, результат не сильно выше остальных алгоритмов.

Поиск по префиксу (Prefix Search) показывает хорошие результаты, но не является самым быстрым алгоритмом.

Таким образом, самыми быстрыми алгоритмами поиска являются встроенные инструменты библиотеки cstring C++: find и strstr.

**Листинг программы:**

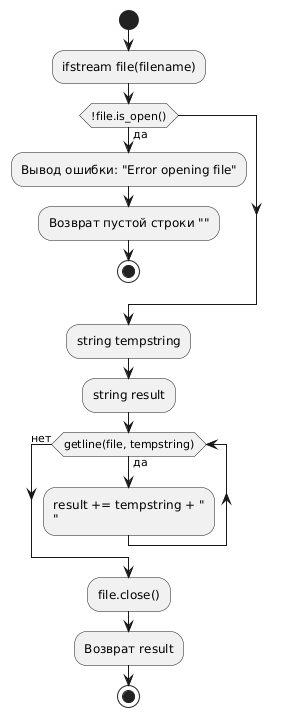
*#include* <iostream>  
*#include* <fstream>  
*#include* <cstring>  
*#include* <vector>  
*#include* <chrono>  
  
using namespace std;  
  
*string readFile*(const *string* &*filename*) {  
 *ifstream* file*(filename)*;  
 if (!file.*is\_open*()) {  
 cerr *<<* "Error opening file: " *<< filename << endl*;  
 return "";  
 }  
 *string* tempstring;  
 *string* result;  
 while (*getline*(file, tempstring)) {  
 result *+=* tempstring *+* "\n";  
 }  
 file.*close*();  
  
 return result;  
}  
  
void *naive\_search*(const *string* &*book*, const *string* &*findstr*) {  
 for (int i = 0; i < *book*.*size*(); i++) {  
 if (*book*.*substr*(i, *findstr*.*size*()) *== findstr*) {  
 cout *<<* i *<<* " ";  
 }  
 }  
}  
  
void *find\_search*(const *string* &*book*, const *string* &*findstr*) {  
 unsigned long long i = 0, last\_i = 0;  
 while (i < *book*.*size*()) {  
 last\_i = i;  
 i = *book*.*find*(*findstr*, i) + 1;  
 if (last\_i >= i) {  
 break;  
 }  
 cout *<<* i - 1 *<<* " ";  
 }  
}  
  
void *strstr\_search*(const *string* &*book*, const *string* &*findstr*) {  
 const char\* bookptr = *book*.*c\_str*();  
 const char\* findptr = *findstr*.*c\_str*();  
 unsigned long long offset = 0;  
 while (true) {  
 const char\* pos = *strstr*(bookptr + offset, findptr);  
 if (!pos) {  
 break;  
 }  
 const long long found\_pos = pos - bookptr;  
 cout *<<* found\_pos *<<* " ";  
 offset = found\_pos + *findstr*.*size*();  
 }  
}  
  
void *prefix\_search*(const *string* &*book*, const *string* &*findstr*) {  
 const *string* new\_str = *findstr +* "#" *+ book*;  
 *vector*<int> pi*(*new\_str.*length*(), 0*)*;  
 for (int i = 1; i < new\_str.*length*(); i++) {  
 int j = pi*[*i - 1*]*;  
 while (j > 0 && new\_str*[*i*]* != new\_str*[*j*]*) {  
 j = pi*[*j - 1*]*;  
 }  
 if (new\_str*[*i*]* == new\_str*[*j*]*) {  
 pi*[*i*]* = j + 1;  
 } else {  
 pi*[*i*]* = j;  
 }  
 }  
 const unsigned long long t\_len = *findstr*.*length*();  
 for (int i = 0; i < *book*.*length*(); i++) {  
 if (pi*[*t\_len + 1 + i*]* == t\_len) {  
 cout *<<* i - t\_len + 1 *<<* " ";  
 }  
 }  
}  
  
long long *mod\_pow*(long long *base*, long long *exponent*, const long long *mod*) {  
 long long result = 1;  
 *base* %= *mod*;  
 while (*exponent* > 0) {  
 if (*exponent* % 2 == 1)  
 result = (result \* *base*) % *mod*;  
 *exponent* >>= 1;  
 *base* = (*base* \* *base*) % *mod*;  
 }  
 return result;  
}  
  
long long *myhash*(const *string* &*s*) {  
 long long hash = 0;  
 long long p\_pow = 1;  
 for (const char c : *s*) {  
 hash = (hash + (static\_cast<unsigned char>(c) + 1) \* p\_pow) % 1000000007;  
 p\_pow = (p\_pow \* 257) % 1000000007;  
 }  
 return hash;  
}  
  
void *hash\_search*(const *string* &*book*, const *string* &*sub*) {  
 const unsigned long long n = *book*.*length*();  
 const unsigned long long m = *sub*.*length*();  
 const long long hashW = *myhash*(*sub*);  
 long long hashS = *myhash*(*book*.*substr*(0, m));  
  
 *vector*<long long> p\_pow*(max*(n, m)*)*;  
 p\_pow*[*0*]* = 1;  
 for (int i = 1; i < p\_pow.*size*(); ++i)  
 p\_pow*[*i*]* = (p\_pow*[*i - 1*]* \* 257) % 1000000007;  
  
 const long long inv\_p = *mod\_pow*(257, 1000000007 - 2, 1000000007);  
  
 for (int i = 0; i <= n - m; ++i) {  
 if (hashS == hashW && *book*.*substr*(i, m) *== sub*) {  
 cout *<<* i *<<* " ";  
 }  
 if (i < n - m) {  
 const long long old\_char = static\_cast<unsigned char>(*book[*i*]*) + 1;  
 const long long new\_char = static\_cast<unsigned char>(*book[*i + m*]*) + 1;  
  
 hashS = (hashS - old\_char + 1000000007) % 1000000007;  
 hashS = (hashS \* inv\_p) % 1000000007;  
 hashS = (hashS + new\_char \* p\_pow*[*m - 1*]*) % 1000000007;  
 }  
 }  
}  
  
typedef void (\**func*) (const *string*&, const *string*&);  
struct *analyze*{  
 *string* name;  
 *func* search;  
};  
  
  
int *main*() {  
 const *string* text = *readFile*("../books/Harry Potter and the Chamber of Secrets.txt");  
  
 *analyze* analyses[5] = {  
 *{*"Naive search", *naive\_search}*,  
 *{*"Find search", *find\_search}*,  
 *{*"Strstr search", *strstr\_search}*,  
 *{*"Prefix search", *prefix\_search}*,  
 *{*"Hash search", *hash\_search}* };  
 *string* finds[5] = {  
 "Harry",  
 "I dunno",  
 "What's up?",  
 "Hermione and Ron",  
 "said Professor McGonagall"  
 };  
  
 for(const *string*& findstr: finds){  
 for(const auto& [name, search]: analyses){  
 cout *<<* name *<<* " (searching for: " *<<* findstr *<<* "):" *<< endl*;  
  
 auto start = chrono::*high\_resolution\_clock*::*now*();  
 search(text, findstr);  
 auto finish= chrono::*high\_resolution\_clock*::*now*();  
  
 const auto duration = chrono::*duration\_cast*<chrono::*microseconds*>(finish *-* start).*count*();  
  
 cout *<< endl <<* "Total time: " *<<* duration *<<* " ms" *<< endl*;  
 }  
 cout *<< endl*;  
 }  
  
 *system*("pause");  
  
 return 0;  
}

**Запуск программы:**

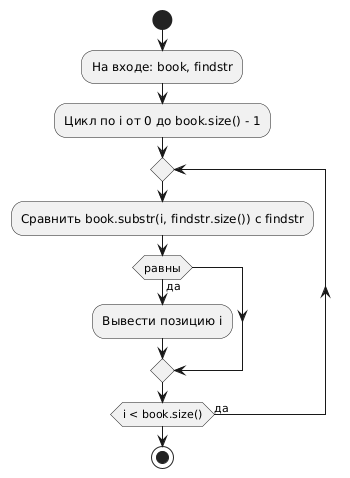


**Блок-схемы:**

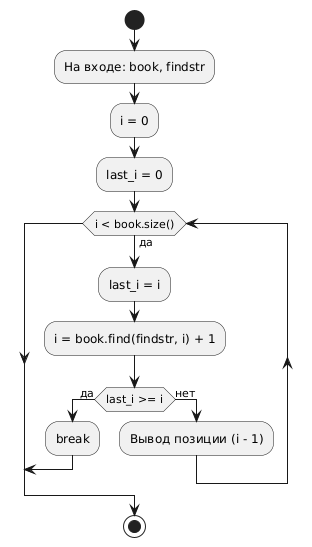
readFile:



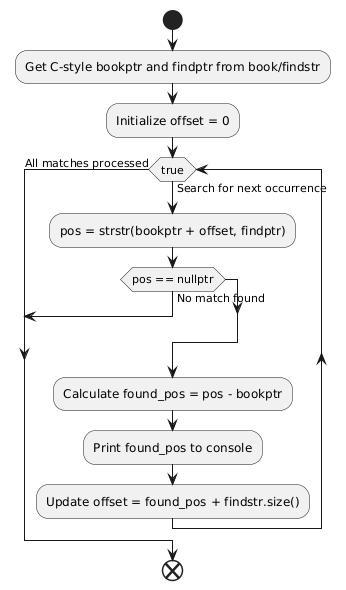
naive\_search:



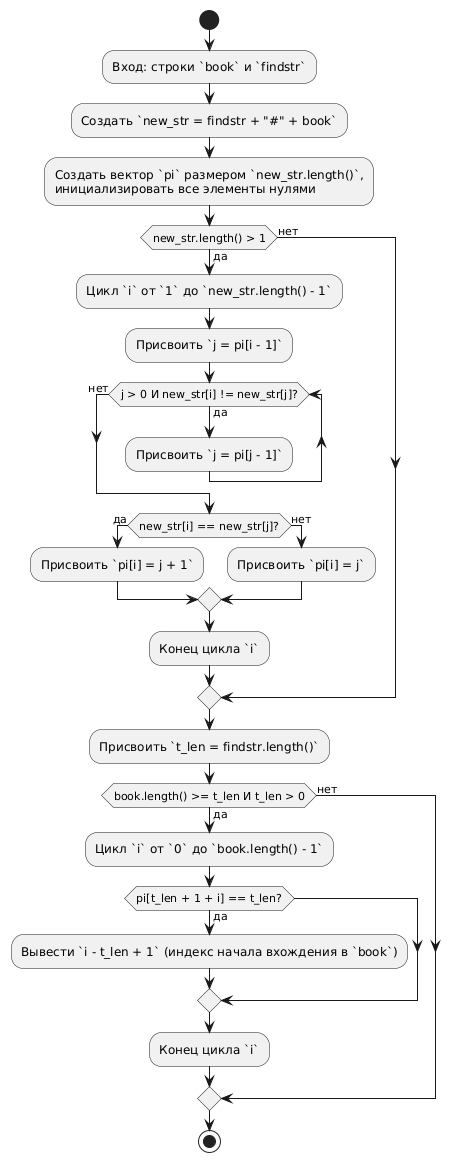
find\_search:



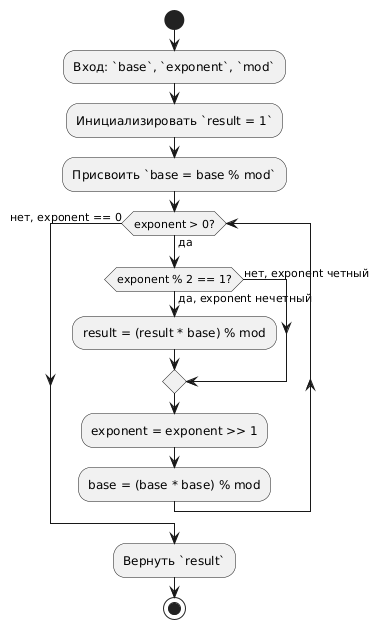
strstr\_search:



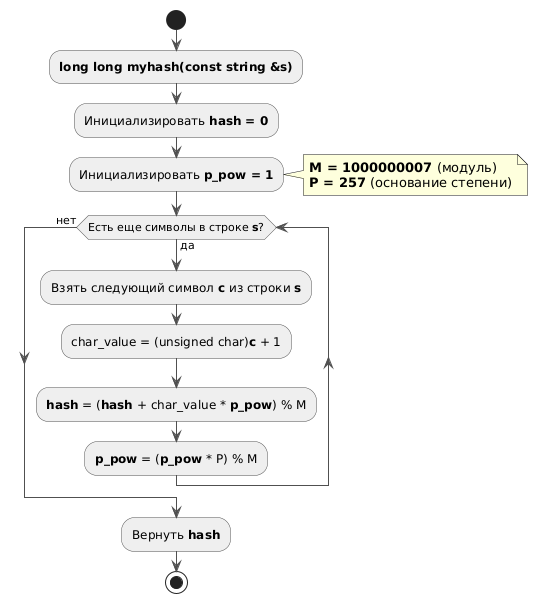
prefix\_search:



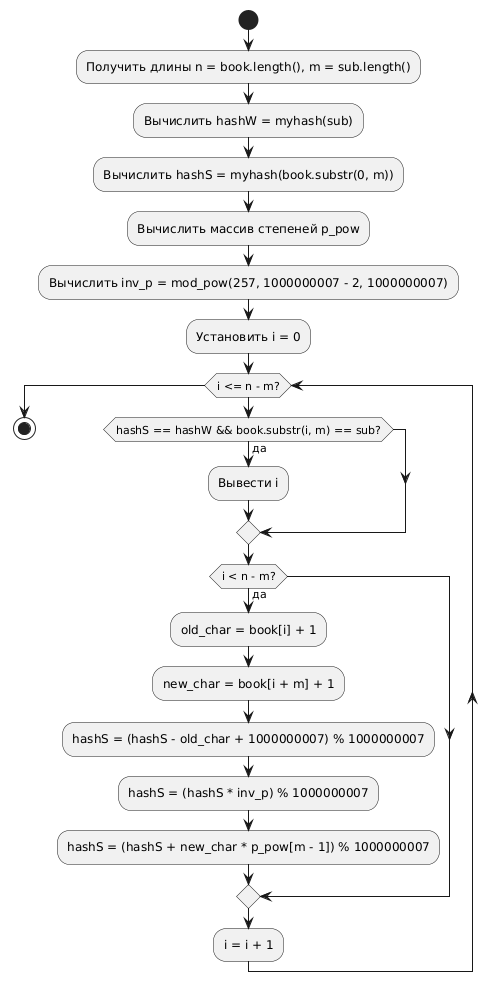
mod\_pow:



myhash:



hash\_search:



main:

