Міністерство освіти України

Національний технічний університет "ХПІ"

кафедра "Стратегічного управління"

|  |  |
| --- | --- |
|  |  |

**Звіт до лабораторної роботи №5**

з дисципліни "Алгоритми на структури даних"

Виконав: студент групи КН-321А

Бородай Д. А.

Перевірив: старший викладач

Мошко Є.О.

Харків 2022

**Тема лабораторної роботи:** Алгоритми пошуку (лінійний пошук, бінарний пошук, пошук з бар'єром, пошук підрядка в рядку, алгоритм Кнута-Морріса-Пратта, алгоритм Бойера-Мура, алгоритм Рабина – Карпа).

**Мета:** ознайомитися із основними алгоритмами пошуку даних та особливостями їх програмної реалізації. Набути практичних навичок по роботі зі алгоритмами пошуку даних

**Порядок виконання роботи**:

1. Визначити масив, в якому буде виконуватися пошук. Використовувати масив, створений в роботі №4.

2. Розробити функції лінійного пошуку в масиві, бінарного пошуку в масиві. Для забезпечення бінарного пошуку в масиві використовувати найбільш ефективний алгоритм сортування, визначений при виконанні лабораторної роботі №4:

− ввести інформацію в масив з файлу;

− виконати лінійний пошук в масиві;

− виконати пошук з бар'єром;

− упорядкувати елементи масиву функцією сортування і виконати бінарний пошук.

3. Розробити функції для пошуку підрядка в рядку:

− виконати прямий пошук підрядка;

− виконати алгоритм Кнута-Морріса-Пратта;

− виконати алгоритм Бойера-Мура;

− виконати алгоритм Рабина – Карпа.

4. Дослідити складність алгоритмів. Провести асимптотичний аналіз алгоритмів пошуку та зробити висновки.

− для порівняння алгоритмів пошуку виконати наступні кроки;

− створити таблицю асимптотичних оцінок трудомісткості алгоритмів в кращому, середньому, гіршому випадках;

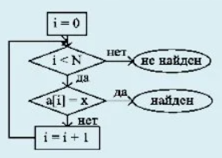
− розставити лічильники операцій у функціях пошуку;

− провести експеримент, визначити середню кількість операцій для різних алгоритмів, побудувати графіки;

− створити таблиці і представити графіки експериментальних оцінок алгоритмів

## ОПИС АЛГОРИТМІВ ПОШУКУ

1.1 Лінійний пошук



Блок-схема 1.1 – Лінійний пошук

Приклад коду пошуку:

void FindLinear(int\* arr, int size, int\* ans, int h, long key)

{

int counter = 0;

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

{

if (arr[i] == key && ans[h] != i)

ans[h++] = i;

counter++;

}

if (h != 0)

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

cout << "Ключ " << key << " находится в ячейке " << ans[i] << endl;

else

cout << "\nМы не нашли ключ " << key << " в массиве\n";

cout << "\nOperations: " << counter + 1 << endl;

}

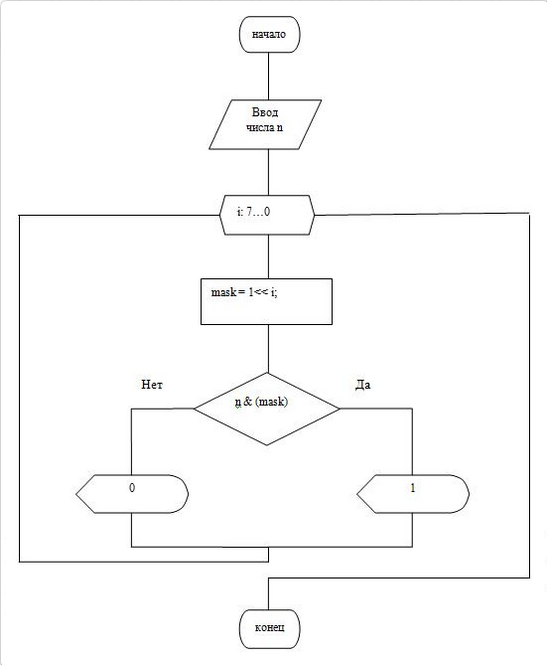
Повний код програми наведено у додатку А.

Найгірша складність: О(n)

Середня складність: О(n)

Середня продуктивність: О(n/2)

1.2 Бінарний пошук



Блок-схема 1.2 – Бінарний пошук

Приклад коду бінарного пошуку:

void binary(int\* arr, int size, int\* ans, int h, long key)

{

long\* arr1 = new long[size];

for (size\_t i = 0; i < size; i++)

arr1[i] = arr[i];

int counter = 0;

bucketsort(arr, size);

bool flag = false;

int l = 0;

int r = size - 1;

int size1 = 0;

int qw = 0;

int count = 0;

for (size\_t i = 0; i < size; i++)

cout << arr[i] << " ";

cout << endl << endl;

while ((l <= r) && (flag != true))

{

ans[count] = (l + r) / 2;

if (arr[ans[count]] == key)

{

flag = true;

size1++;

qw = ans[count];

for (size\_t i1 = ans[count] + 1; i1 <= size; i1++)

{

if (arr[i1] == key)

{

count += 1;

ans[count] = i1;

size1++;

}

counter++;

}

for (int i2 = qw - 1; i2 >= 0; i2--)

{

if (arr[i2] == key)

{

count += 1;

ans[count] = i2;

size1++;

}

else

goto S;

counter++;

}

}

if (arr[ans[count]] > key)

{

r = ans[count] - 1;

counter++;

}

else

{

l = ans[count] + 1;

counter++;

}

}

counter++;

S:

if (flag)

{

for (size\_t i = 0; i < size1; i++)

cout << "Ключ " << key << " находится в ячейке " << ans[i] << endl;

}

else cout << "Мы не нашли ключ " << key << " в массиве\n";

cout << "\nOperations: " << counter + 1 << endl;

for (size\_t i = 0; i < size; i++)

arr[i] = arr1[i];

}

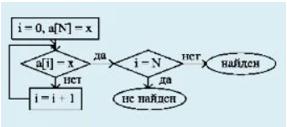
Повний текст програми наведено у додатку А.

Найгірша складність: О(log n)

Середня складність: О(log n)

Найкраща складність: О(1)

1.3 Пошук бар’єром



Блок-схема 1.3 – Пошук бар'єром

Приклад коду алгоритм пошуку бар’єром:

void barrier\_seach(int\* arr, int size, int\* ans, int h, long key)

{

int counter = 0;

int last = arr[size - 1];

arr[size - 1] = key;

int k = 0;

int p = 0;

for (size\_t j = 0; j < size; j++)

{

size\_t i = 0;

for (;;)

{

counter++;

if (p == size) goto S1;

if (arr[i] == key && i > k)

{

goto S;

}

else ++i;

p++;

}

S:

p = 0;

k = i;

arr[size - 1] = last;

if (i != (size - 1) || key == last)

{

ans[h++] = i;

counter++;

}

S1:

counter++;

}

if (h != 0)

for (int l = 0; l < h; l++)

cout << "Ключ " << key << " находится в ячейке " << ans[l] << endl;

else cout << "Мы не нашли ключ " << key << " в массиве\n";

cout << "\nOperations: " << counter + 1 << endl;

}

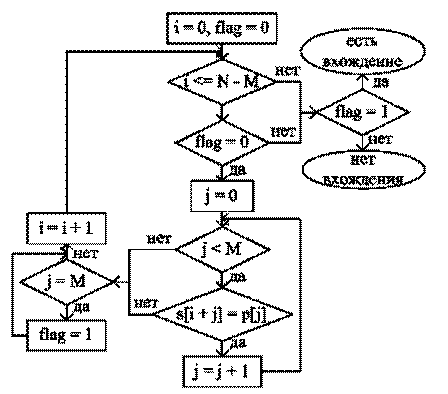
Повний Текст програми наведено у додатку А.

Найгірша складність: О(n)

Середня складність: О(n)

Середня продуктивність: О(n/2)

1.4 Алгоритм прямого пошуку підрядка в рядку



Блок-схема 1.4 ­ Прямий пошук

Приклад коду даного алгоритму:

int Forward(char\* s, char\* c, int n)

{

int i, j, lenC, lenS;

for (lenC = 0; c[lenC]; lenC++);

for (lenS = 0; s[lenS]; lenS++);

for (i = 0; i <= lenS - lenC; i++)

{

for (j = 0; s[i + j] == c[j]; j++)

counterFinding++;

if (j - lenC == 1 && i == lenS - lenC && !(n - 1)) return i;

if (j == lenC)

if (n - 1) n--;

else return i;

counterFinding++;

}

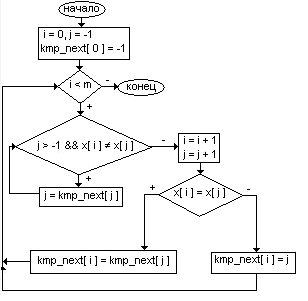
return -1;

}

Повний код програми наведено у додатку А.

Найгірший випадок. Нехай масив T→{AAA….AAAB}, довжина │T│=N, зразок W→{A….AB}, довжина │W│=M. Очевидно, що для виявлення збігу в кінці рядка потрібно зробити порядку N \* M порівнянь, тобто O (N \* M).

1.5 Алгоритм пошуку Кнута – Моріса – Пратта



Блок-схема 1.5 ­ КМП алгоритм

Приклад коду КМП алгоритму:

void KMP(const char\* text, const char\* pattern, int m, int n)

{

int counter = 0;

if (\*pattern == '\0' || n == 0) {

printf("The pattern occurs with shift 0");

counter++;

}

if (\*text == '\0' || n > m) {

printf("Pattern not found");

counter++;

}

int\* next = new int[n + 1];

for (int i = 0; i < n + 1; i++) {

next[i] = 0;

counter++;

}

for (int i = 1; i < n; i++)

{

int j = next[i + 1];

while (j > 0 && pattern[j] != pattern[i]) {

j = next[j];

counter++;

}

if (j > 0 || pattern[j] == pattern[i])

next[i + 1] = j + 1;

counter++;

}

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

{

if (\*(text + i) == \*(pattern + j))

{

counter++;

if (++j == n) {

printf("Индекс нахождения: %d\n", i - j + 1);

}

}

else if (j > 0)

{

j = next[j];

i--;

}

}

cout << "\nOperations: " << counter + 1 << endl;

}

Повний код програми наведено у додатку А.

Алгоритм КМП-пошуку фактично вимагає лише порядку N порівнянь навіть у найгіршому випадку.

1.6 Алгоритм пошуку Бойера – Мура

У більшості випадків, крім спеціально побудованих прикладів, БМ-пошук вимагає значно менше N порівнянь. У самих сприятливих обставин, коли останній символ зразка завжди потрапляє на символ, що не збігається, текст порівнянь дорівнює (N / M), у гіршому ж випадку – О((N-M+1)\*M+ p), де p – потужність алфавіту.

Приклад коду алгоритму Бойера – Мура:

int BMSearch(char\* string, char\* substring) {

int sl = 0;

int ssl = 0;

int res = -1;

while (string[sl] != NULL) {

sl++;

counterFinding6++;

}

while (substring[ssl] != NULL) {

ssl++;

counterFinding6++;

}

if (sl == 0)

printf("Некорректная строка\n");

else if (ssl == 0)

printf("Некорректная подстрока\n");

else {

int i, Pos;

int BMT[256];

for (i = 0; i < 256; i++)

BMT[i] = ssl;

for (i = ssl - 1; i >= 0; i--)

if (BMT[(short)(substring[i])] == ssl)

BMT[(short)(substring[i])] = ssl - i - 1;

counterFinding6++;

Pos = ssl - 1;

while (Pos < sl)

if (substring[ssl - 1] != string[Pos])

Pos = Pos + BMT[(short)(string[Pos])];

else

for (i = ssl - 2; i >= 0; i--) {

if (substring[i] != string[Pos - ssl + i + 1]) {

Pos += BMT[(short)(string[Pos - ssl + i + 1])] - 1;

break;

}

else

if (i == 0)

return Pos - ssl + 1;

counterFinding6++;

}

counterFinding6++;

}

printf("\n");

return res;

}

Повний код програми наведено у додатку А.

1.6 Алгоритм Рабіна – Карпа

У гіршому випадку час роботи алгоритму РК - Θ((N-M+1)\*M), в середньому він працює досить швидко - за час О(N+M).

Приклад коду алгоритма Рабіна – Карпа:

int RabinKarpMatch(char\* T, char\* P, int d, int q)

{

int i, j, p, t, n, m, h, found;

n = strlen(T);

m = strlen(P);

h = mod(d, m - 1, q);

p = t = 0;

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

{

p = (d \* p + tonum(P[i])) % q;

t = (d \* t + tonum(T[i])) % q;

counterFinding7++;

}

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

{

if (p == t)

{

found = 1;

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

if (P[j] != T[i + j])

{

counterFinding7++;

found = 0;

break;

}

if (found)

return i;

}

else

{

t = (d \* (t - ((tonum(T[i]) \* h) % q)) + tonum(T[i + m])) % q;

}

counterFinding7++;

}

return -1;

}

Повний код програми наведено у додаку А.

## АСИМПОТИЧНИЙ АНАЛІЗ

* 1. Порівняння бінарного, лінійного та бар’єрного пошуку

Таблиця 2.1 – Данні, отримані в ході тестувань

|  |  |  |  |
| --- | --- | --- | --- |
| n/Пошук | Бінарний пошук | Лінійний пошук | Пошук бар'єром |
| 10 | 7 | 11 | 12 |
| 100 | 34 | 101 | 102 |
| 1000 | 318 | 1001 | 1008 |
| 10 000 | 1894 | 10001 | 10058 |

За отриманими результатами, витікає висновок, що найкращий алгоритм пошуку – бінарний.

* 1. Порівняння алгоритмів пошуку підрядка у рядку

Вхідні данні:

kergeighbnkehjbgjkenbghfgbvuhnbrwgkjsjiohweriybgknjcviwrgjnbghwbfybwevjnwrjwirghndkvjbifgioupijrweibshdabuy8gwhbfhfbvwueiyrgnsdgf

Пошук: viwrgjnbg

# Висновок

Алгоритм пошук може бути значно ефективнішим, якщо дані будуть впорядковані. Іншим, відносно простим, методом доступу до елемента є метод бінарного (дихотомічного) пошуку, який виконується в явно впорядкованій послідовності елементів. Записи в таблицю заносяться в лексикографічному (символьні ключі) або чисельно (числові ключі) зростаючому порядку.

Основною відмінністю алгоритму Кнута-Моріса-Пратта (КМП) від алгоритму прямого пошуку є здійснення зсуву слова не на один символ на кожному кроці алгоритму, а на деяку змінну кількість символів. Таким чином, перед тим як виконувати черговий зсув, потрібно визначити величину зсуву. Для підвищення ефективності алгоритму необхідно, щоб зсув на кожному кроці був би якомога більшим.

Алгоритм Кнута-Моріса-Пратта дає справжній виграш тільки тоді, коли невдачі передувала деяка кількість збігів. Лише у цьому випадку слово зсовується більше ніж на одиницю. На жаль, це швидше виняток, ніж правило: збіги зустрічаються значно рідше, ніж незбіги. Тому виграш від практичного використання КМП-стратегії в більшості випадків пошуку в звичайних текстах досить незначний. Метод, який запропонували Р. Боуєр і Д. Мур в 975 р., не тільки покращує обробку самого поганого випадку, але й дає виграш в проміжних ситуаціях.

Алгоритм Рабіна-Карпа далеко не найшвидший алгоритм читання. Оскільки він робить порівняння символів безпосередньо, він може швидко шукати в потоковому режимі приклади входжень певного патерну.

# Додаток А

Код Source.cpp

#include <iostream>

#include <Windows.h>

#include <fstream>

#include <iomanip>

#define tonum(c) (c >= 'A' && c <= 'Z' ? c - 'A' : c - 'a' + 26)

using namespace std;

int counterFinding = 0;

int counterMerge = 0;

int counterHeap = 0;

int counterFinding6 = 0;

int counterFinding7 = 0;

void printarray(int\* arr, int size) {

cout << endl << endl;

for (size\_t i = 0; i < size; i++)

cout << arr[i] << setw(4);

cout << endl;

}

void siftDown(int\* arr, int size, int bottom)

{

int maxChild; // индекс максимального потомка

int done = 0; // флаг того, что куча сформирована

// Пока не дошли до последнего ряда

while ((size \* 2 <= bottom) && (!done))

{

if (size \* 2 == bottom) // если мы в последнем ряду,

maxChild = size \* 2; // запоминаем левый потомок

// иначе запоминаем больший потомок из двух

else if (arr[size \* 2] > arr[size \* 2 + 1])

maxChild = size \* 2;

else

maxChild = size \* 2 + 1;

// если элемент вершины меньше максимального потомка

if (arr[size] < arr[maxChild])

{

int temp = arr[size]; // меняем их местами

arr[size] = arr[maxChild];

arr[maxChild] = temp;

size = maxChild;

}

else // иначе

done = 1; // пирамида сформирована

counterHeap++;

}

}

void selectionsort(int\* arr, int size)

{

int c = 0;

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

int minz = arr[i], \* ind = &arr[i];

for (int j = i + 1; j < size; j++) {

if (arr[j] < minz) minz = arr[j], ind = &arr[j];

c++;

}

swap(arr[i], \*ind);

c++;

}

}

void Show(int\* arr, int size)

{

for (size\_t i = 0; i < size; i++)

{

cout << arr[i] << " ";

}

}

void FindLinear(int\* arr, int size, int\* ans, int h, long key)

{

int counter = 0;

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

{

if (arr[i] == key && ans[h] != i)

{

ans[h++] = i;

}

counter++;

}

if (h != 0)

{

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

{

cout << "Ключ " << key << " находится в ячейке " << ans[i] << endl;

}

}

else

{

cout << "\nМы не нашли ключ " << key << " в массиве\n";

}

cout << "\nOperations: " << counter + 1 << endl;

}

void barrier\_seach(int\* arr, int size, int\* ans, int h, long key)

{

int counter = 0;

int last = arr[size - 1];

arr[size - 1] = key;

int k = 0;

int p = 0;

for (size\_t j = 0; j < size; j++)

{

size\_t i = 0;

for (;;)

{

counter++;

if (p == size)

{

goto S1;

}

if (arr[i] == key && i > k)

{

goto S;

}

else

{

++i;

}

p++;

}

S:

p = 0;

k = i;

arr[size - 1] = last;

if (i != (size - 1) || key == last)

{

ans[h++] = i;

counter++;

}

S1:

counter++;

}

if (h != 0)

{

for (int l = 0; l < h; l++)

{

cout << "Ключ " << key << " находится в ячейке " << ans[l] << endl;

}

}

else

{

cout << "Мы не нашли ключ " << key << " в массиве\n";

}

cout << "\nOperations: " << counter + 1 << endl;

}

void binary(int\* arr, int size, int\* ans, int h, long key)

{

long\* arr1 = new long[size];

for (size\_t i = 0; i < size; i++)

{

arr1[i] = arr[i];

}

int counter = 0;

selectionsort(arr, size);

bool flag = false;

int l = 0;

int r = size - 1;

int size1 = 0;

int qw = 0;

int count = 0;

for (size\_t i = 0; i < size; i++)

{

cout << arr[i] << " ";

}

cout << endl << endl;

while ((l <= r) && (flag != true))

{

ans[count] = (l + r) / 2;

if (arr[ans[count]] == key)

{

flag = true;

size1++;

qw = ans[count];

for (size\_t i1 = ans[count] + 1; i1 <= size; i1++)

{

if (arr[i1] == key)

{

count += 1;

ans[count] = i1;

size1++;

}

counter++;

}

for (int i2 = qw - 1; i2 >= 0; i2--)

{

if (arr[i2] == key)

{

count += 1;

ans[count] = i2;

size1++;

}

else

{

goto S;

}

counter++;

}

}

if (arr[ans[count]] > key)

{

r = ans[count] - 1;

counter++;

}

else

{

l = ans[count] + 1;

counter++;

}

}

counter++;

S:

if (flag)

{

for (size\_t i = 0; i < size1; i++)

{

cout << "Ключ " << key << " находится в ячейке " << ans[i] << endl;

}

}

else

{

cout << "Мы не нашли ключ " << key << " в массиве\n";

}

cout << "\nOperations: " << counter + 1 << endl;

for (size\_t i = 0; i < size; i++)

{

arr[i] = arr1[i];

}

}

void selsort2(int\* arr, int b, int e) {

int l = b, r = e;

int piv = arr[(l + r) / 2];

while (l <= r) {

while (arr[l] > piv) // Поменяй знак

l++;

while (arr[r] < piv) // Поменяй знак

r--;

if (l <= r)

swap(arr[l++], arr[r--]);

}

if (b < r)

selsort2(arr, b, r);

if (e > l)

selsort2(arr, l, e);

}

int Forward(char\* s, char\* c, int n)

{

int i, j;

int lenC, lenS;

for (lenC = 0; c[lenC]; lenC++);

for (lenS = 0; s[lenS]; lenS++);

for (i = 0; i <= lenS - lenC; i++) // Пока есть возможность поиска

{

for (j = 0; s[i + j] == c[j]; j++)

{

// Проверяем совпадение посимвольно

counterFinding++;

}

if (j - lenC == 1 && i == lenS - lenC && !(n - 1)) return i;

if (j == lenC)

if (n - 1) n--;

else return i;

counterFinding++;

}

//Иначе вернем -1 как результат отсутствия подстроки

return -1;

}

void KMP(const char\* text, const char\* pattern, int m, int n)

{

int counter = 0;

// базовый случай 1: шаблон равен NULL или пуст

if (\*pattern == '\0' || n == 0) {

printf("The pattern occurs with shift 0");

counter++;

}

// базовый случай 2: текст равен NULL или длина текста меньше длины шаблона

if (\*text == '\0' || n > m) {

printf("Pattern not found");

counter++;

}

// next[i] сохраняет индекс следующего лучшего частичного совпадения

int\* next = new int[n + 1];

for (int i = 0; i < n + 1; i++) {

next[i] = 0;

counter++;

}

for (int i = 1; i < n; i++)

{

int j = next[i + 1];

while (j > 0 && pattern[j] != pattern[i]) {

j = next[j];

counter++;

}

if (j > 0 || pattern[j] == pattern[i]) {

next[i + 1] = j + 1;

}

counter++;

}

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

{

if (\*(text + i) == \*(pattern + j))

{

counter++;

if (++j == n) {

printf("Индекс нахождения: %d\n", i - j + 1);

}

}

else if (j > 0)

{

j = next[j];

i--; // так как `i` будет увеличен на следующей итерации

}

}

cout << "\nOperations: " << counter + 1 << endl;

}

int BMSearch(char\* string, char\* substring) {

int sl = 0;

int ssl = 0;

int res = -1;

while (string[sl] != NULL) {

sl++;

counterFinding6++;

}

while (substring[ssl] != NULL) {

ssl++;

counterFinding6++;

}

if (sl == 0)

printf("Некорректная строка\n");

else if (ssl == 0)

printf("Некорректная подстрока\n");

else {

int i, Pos;

int BMT[256];

for (i = 0; i < 256; i++)

BMT[i] = ssl;

for (i = ssl - 1; i >= 0; i--)

if (BMT[(short)(substring[i])] == ssl)

BMT[(short)(substring[i])] = ssl - i - 1;

counterFinding6++;

Pos = ssl - 1;

while (Pos < sl)

if (substring[ssl - 1] != string[Pos])

Pos = Pos + BMT[(short)(string[Pos])];

else

for (i = ssl - 2; i >= 0; i--) {

if (substring[i] != string[Pos - ssl + i + 1]) {

Pos += BMT[(short)(string[Pos - ssl + i + 1])] - 1;

break;

}

else

if (i == 0)

return Pos - ssl + 1;

counterFinding6++;

}

counterFinding6++;

}

printf("\n");

return res;

}

int mod(int a, int p, int m)

{

if (p == 0)

return 1;

int sqr = mod(a, p / 2, m) % m;

if (p & 1)

return ((a % m) \* sqr) % m;

else

return sqr;

}

int RabinKarpMatch(char\* T, char\* P, int d, int q)

{

int i, j, p, t, n, m, h, found;

n = strlen(T);

m = strlen(P);

h = mod(d, m - 1, q);

p = t = 0;

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

{

p = (d \* p + tonum(P[i])) % q;

t = (d \* t + tonum(T[i])) % q;

counterFinding7++;

}

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

{

if (p == t)

{

found = 1;

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

if (P[j] != T[i + j])

{

counterFinding7++;

found = 0;

break;

}

if (found)

return i;

}

else

{

t = (d \* (t - ((tonum(T[i]) \* h) % q)) + tonum(T[i + m])) % q;

}

counterFinding7++;

}

return -1;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int chooseFile;

int size;

long key;

int choose2;

while (true) {

cout << "1 - Массив \n2 - Cтрока \n0 - Exit \n-->";

cin >> choose2;

if (choose2 == 1)

{

cout << "Размер массива: ";

cin >> size;

int\* arr = new int[size];

int\* ans = new int[size];

int h = 0;

int choose;

int menu = -1;

int c = 0;

while (menu != 0)

{

cout << endl << endl;

cout << "1 - Заполнить рандомно\n";

cout << "2 - Чтение из файла\n";

cout << "3 - Сортировать по возрастанию\n";

cout << "4 - Сортировать по убыванию\n";

cout << "5 - Удалить массив\n";

cout << "6 - Линейный поиск\n";

cout << "7 - Поиск с барьером\n";

cout << "8 - Бинарный поиск\n";

cout << "9 - Запись в файл\n";

cout << "0 - Назад \n";

cout << "--> ";

cin >> menu;

string name = "DataForArray.txt";

switch (menu)

{

case 1:

{

for (size\_t i = 0; i < size; i++)

arr[i] = rand() % 10;

printarray(arr, size);

c = 1;

break;

}

case 2:

{

char ch;

ifstream file\_read;

file\_read.open(name);

if (!file\_read.is\_open())

{

cout << "Error" << endl;

}

else

{

for (size\_t j = 0; j < size; j++)

{

file\_read >> arr[j];

}

}

file\_read.close();

printarray(arr, size);

c = 1;

break;

}

case 3:

{

selectionsort(arr, size);

printarray(arr, size);

break;

}

case 4:

{

selsort2(arr, 0, size - 1);

printarray(arr, size);

break;

}

case 5:

{

size = 0;

delete[] arr;

c = 0;

break;

}

case 6:

{

cout << "\nВведите число: ";

cin >> key;

FindLinear(arr, size, ans, h, key);

ans = new int[size];

break;

}

case 7:

{

cout << "\nВведите число: ";

cin >> key;

barrier\_seach(arr, size, ans, h, key);

ans = new int[size];

break;

}

case 8:

{

cout << "\nВведите число: ";

cin >> key;

binary(arr, size, ans, h, key);

ans = new int[size];

break;

}

case 9:

{

ofstream fout;

fout.open(name);

for (int i = size - 1; i >= 0; i--)

{

if (i == 0)

{

fout << arr[i];

}

else

{

fout << arr[i] << " ";

}

}

fout.close();

break;

}

}

}

}

else if(choose2 == 2)

{

char arr1[10000];

int c = 0;

int menu = -1;

while (menu != 0)

{

if (c == 0)

{

}

else

{

for (size\_t i = 0; i < strlen(arr1); i++)

{

cout << arr1[i];

}

}

cout << endl << endl;

cout << "1 - Ввести строку\n";

cout << "2 - Запись из файла\n";

cout << "3 - Удалить строку\n";

cout << "4 - Прямой поиск\n";

cout << "5 - Алгоритм Кнута-Морриса-Пратта\n";

cout << "6 - Алгоритм Бойера-Мура\n";

cout << "7 - Алгоритм Рабина-Карпа\n";

cout << "8 - Запись в файл\n";

cout << "0 - Назад\n";

cout << "--> ";

cin >> menu;

string name1 = "DataForString.txt";

switch (menu)

{

case 1:

{

cin.ignore(32767, '\n');

cout << "Введите строку: ";

cin.getline(arr1, 10000);

c = 1;

break;

}

case 2:

{

char ch;

ifstream file\_read\_s;

file\_read\_s.open(name1);

if (!file\_read\_s.is\_open())

{

cout << "Error" << endl;

}

else

{

file\_read\_s >> arr1;

}

file\_read\_s.close();

c = 1;

break;

}

case 3:

{

strcpy\_s(arr1, " ");

c = 0;

break;

}

case 4:

{

int i, n = 0;

char arr2[10000];

cin.ignore(32767, '\n');

cout << "Введите подстроку: ";

cin.getline(arr2, 100);

for (i = 1; n != -1; i++)

{

n = Forward(arr1, arr2, i);

if (n >= 0)

cout << "Индекс начала искомой подстроки: " << n << endl;

}

cout << "\nOperations: " << counterFinding + 1 << endl;

break;

}

case 5:

{

char arr2[100];

cin.ignore(32767, '\n');

cout << "Введите подстроку: ";

cin.getline(arr2, 100);

int n = strlen(arr1);

int m = strlen(arr2);

KMP(arr1, arr2, n, m);

break;

}

case 6:

{

char arr2[100];

cin.ignore(32767, '\n');

cout << "Введите подстроку: ";

cin.getline(arr2, 100);

int pos = BMSearch(arr1, arr2);

if (pos != -1)

{

cout << "Позиция: " << pos << endl;

}

else

{

cout << "Нету такого рядка\n";

}

cout << "\nOperations: " << counterFinding6 << endl;

break;

}

case 7:

{

char arr2[100];

cin.ignore(32767, '\n');

cout << "Введите подстроку: ";

cin.getline(arr2, 100);

int d = 1, q = 1000;

int sovp;

sovp = RabinKarpMatch(arr1, arr2, d, q);

if (sovp != -1)

cout << "Позиция: " << sovp << endl;

else

cout << "Нету такого подрядка\n";

cout << "\nOperations: " << counterFinding7 << endl;

break;

}

case 8:

{

ofstream fout;

fout.open(name1);

for (int i = strlen(arr1) - 1; i >= 0; i--)

{

if (i == 0)

{

fout << arr1[i];

}

else

{

fout << arr1[i] << " ";

}

}

fout.close();

break;

}

}

}

}

else if (choose2 == 0)

exit(0);

else

continue;

}

}