Zad. 1

Algorytm Morrisa-Pratta

#include <iostream>

#include <string>

#include <cstdlib>

#include <time.h>

using namespace std;

const int N = 79; // długość łańcucha s

const int M = 5; // długość wzorca p

int main( )

{

string s, p;

int PI [ M + 1 ], i, b, pp;

srand ( ( unsigned )time ( NULL ) );

// generujemy łańcuch s

s = "A journey of thousand miles begins with a single step.";

for( i = 0; i < N; i++ ) s += 65 + rand( ) % 2;

// generujemy wzorzec

p = "step";

for( i = 0; i < M; i++ ) p += 65 + rand( ) % 2;

// dla wzorca obliczamy tablicę PI [ ]

PI [ 0 ] = b = -1;

for( i = 1; i <= M; i++ )

{

while( ( b > -1 ) && ( p [ b ] != p [ i - 1 ] ) ) b = PI [ b ];

PI [ i ] = ++b;

}

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch s

cout << s;

// poszukujemy pozycji wzorca w łańcuchu

pp = b = 0;

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

{

while( ( b > -1 ) && ( p [ b ] != s [ i ] ) ) b = PI [ b ];

if( ++b == M )

{

while( pp < i - b + 1 )

{

cout << " "; pp++;

}

cout << "^"; pp++;

b = PI [ b ];

}

}

cout << endl;

return 0;

}

Algorytm Knutha-Morrisa-Pratta

#include <iostream>

#include <string>

#include <cstdlib>

#include <time.h>

using namespace std;

const int N = 79; // długość łańcucha s

const int M = 5; // długość wzorca p

int main( )

{

string s, p;

int KMPNext [ M + 1 ], i, b, pp;

srand ( ( unsigned )time ( NULL ) );

// generujemy łańcuch s

s = "A ship in the harbor is safe, but that is not what a ship is for.";

for( i = 0; i < N; i++ ) s += 65 + rand( ) % 2;

// generujemy wzorzec

p = "ship";

for( i = 0; i < M; i++ ) p += 65 + rand( ) % 2;

// dla wzorca obliczamy tablicę Next [ ]

KMPNext [ 0 ] = b = -1;

for( i = 1; i <= M; i++ )

{

while( ( b > -1 ) && ( p [ b ] != p [ i - 1 ] ) ) b = KMPNext [ b ];

++b;

if( ( i == M ) || ( p [ i ] != p [ b ] ) ) KMPNext [ i ] = b;

else KMPNext [ i ] = KMPNext [ b ];

}

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch s

cout << s << endl;

// poszukujemy pozycji wzorca w łańcuchu

pp = b = 0;

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

{

while( ( b > -1 ) && ( p [ b ] != s [ i ] ) ) b = KMPNext [ b ];

if( ++b == M )

{

while( pp < i - b + 1 )

{

cout << " "; pp++;

}

cout << "^"; pp++;

b = KMPNext [ b ];

}

}

cout << endl;

return 0;

}

Algorytm uproszczony Boyera-Moore'a

#include <iostream>

#include <string>

#include <cstdlib>

#include <time.h>

using namespace std;

const int N = 79; // długość łańcucha s

const int M = 5; // długość wzorca p

const int zp = 65; // kod pierwszego znaku alfabetu

const int zk = 66; // kod ostatniego znaku alfabetu

int main( )

{

string s, p;

int Last [ zk - zp + 1 ], i, j, pp;

srand ( ( unsigned )time ( NULL ) );

// generujemy łańcuch s

s = " Better to be poor and healthy rather than rich and sick.";

for( i = 0; i < N; i++ ) s += zp + rand( ) % ( zk - zp + 1 );

// generujemy wzorzec

p = "to be";

for( i = 0; i < M; i++ ) p += zp + rand( ) % ( zk - zp + 1 );

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch

cout << s << endl;

// dla wzorca obliczamy tablicę Last [ ]

for( i = 0; i <= zk - zp; i++ ) Last [ i ] = -1;

for( i = 0; i < M; i++ ) Last [ p [ i ] - zp ] = i;

// szukamy pozycji wzorca w łańcuchu

pp = i = 0;

while( i <= N - M )

{

j = M - 1;

while( ( j > -1 ) && ( p [ j ] == s [ i + j ] ) ) j--;

if( j == -1 )

{

while( pp < i )

{

cout << " "; pp++;

}

cout << "^"; pp++;

i++;

}

else i += max ( 1, j - Last [ s [ i + j ] - zp ] );

}

cout << endl << endl;

return 0;

}

Algorytm pełny Boyera-Moore'a

#include <iostream>

#include <string>

#include <cstdlib>

#include <time.h>

using namespace std;

const int N = 79; // długość łańcucha s

const int M = 5; // długość wzorca p

const int zp = 65; // kod pierwszego znaku alfabetu

const int zk = 66; // kod ostatniego znaku alfabetu

int main( )

{

string s, p;

int Last [ zk - zp + 1 ], BMNext [ M + 1 ], Pi [ M + 1 ], b, i, j, pp;

srand ( ( unsigned )time ( NULL ) );

// generujemy łańcuch s

s = "Never put off until tomorrow what you can do today.";

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

s += zp + rand( ) % ( zk - zp + 1 );

// generujemy wzorzec p

p = "tomorrow";

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

p += zp + rand( ) % ( zk - zp + 1 );

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch

cout << s << endl;

// dla wzorca obliczamy tablicę Last [ ]

for( i = 0; i <= zk - zp; i++ ) Last [ i ] = -1;

for( i = 0; i < M; i++ ) Last [ p [ i ] - zp ] = i;

// Etap I obliczania tablicy BMNext [ ]

for( i = 0; i <= M; i++ ) BMNext [ i ] = 0;

i = M; b = M + 1; Pi [ i ] = b;

while( i > 0 )

{

while( ( b <= M ) && ( p [ i - 1 ] != p [ b - 1 ] ) )

{

if( BMNext [ b ] == 0 ) BMNext [ b ] = b - i;

b = Pi [ b ];

}

Pi [ --i ] = --b;

}

// Etap II obliczania tablicy BMNext [ ]

b = Pi [ 0 ];

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

{

if( BMNext [ i ] == 0 ) BMNext [ i ] = b;

if( i == b ) b = Pi [ b ];

}

// szukamy pozycji wzorca w łańcuchu

pp = i = 0;

while( i <= N - M )

{

j = M - 1;

while( ( j > -1 ) && ( p [ j ] == s [ i + j ] ) ) j--;

if( j == -1 )

{

while( pp < i )

{

cout << " "; pp++;

}

cout << "^"; pp++;

i += BMNext [ 0 ];

}

else i += max ( BMNext [ j+1 ], j-Last [ s [ i+j ] -zp ] );

}

cout << endl;

return 0;

}

Algorytm Karpa-Rabina

#include <iostream>

#include <string>

#include <cstdlib>

#include <time.h>

using namespace std;

const int N = 79; // długość łańcucha s

const int M = 4; // długość wzorca p

const int zp = 65; // kod pierwszego znaku alfabetu

const int zk = 67; // kod ostatniego znaku alfabetu

// Funkcja obliczająca hasz dla łańcucha x

//----------------------------------------

int h ( string & x )

{

int i, hx;

hx = 0;

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

hx = 3 \* hx + ( x [ i ] - 65 );

return hx;

}

int main( )

{

string s, p;

int pp, i, Hp, Hs;

srand ( ( unsigned )time ( NULL ) );

// generujemy łańcuch s

s = "People who live in glass houses shouldn’t throw stones at others.";

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

s += zp + rand( ) % ( zk - zp + 1 );

// generujemy wzorzec p

p = "shouldn't";

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

p += zp + rand( ) % ( zk - zp + 1 );

// wypisujemy wzorzec

cout << p << endl;

// wypisujemy łańcuch

cout << s << endl;

// obliczamy hasz wzorca

Hp = h ( p );

// obliczamy hasz okna wzorca

Hs = h ( s );

// szukamy pozycji wzorca w łańcuchu

pp = i = 0;

while( true )

{

if( ( Hp == Hs ) && ( p == s.substr ( i, M ) ) )

{

while( pp < i )

{

cout << " "; pp++;

}

cout << "^"; pp++;

}

i++;

if( i == N - M ) break;

Hs = ( Hs - ( s [ i - 1 ] - 65 ) \* 27 ) \* 3 + s [ i + M - 1 ] - 65;

}

cout << endl;

return 0;

}

| Algorytm | Tekst | Wzorzec | Wynik z konsoli |
| --- | --- | --- | --- |
| Morrisa-Pratta | A journey of thousand miles begins with a single step. | step | stepBBBAB |
| Knutha-Morrisa-Pratta | A ship in the harbor is safe, but that is not what a ship is for. | ship | shipAAAAA |
| uproszczony Boyera-Moore'a | Better to be poor and healthy rather than rich and sick. | to be | to beAABAB |
| pełny Boyera-Moore'a | Never put off until tomorrow what you can do today. | tomorrow | tomorrowBAABB |
| Karpa-Rabina | People who live in glass houses shouldn’t throw stones at others. | shouldn’t | shouldn'tCBCC |

Algorytm Karpa-Rabina jest najbardziej odmienny i skomplikowany wobec pozostałych algorytmów co czyni go mniej przystępnym do zapoznawania się w przypadku reszty. Jako jedyny posiada również 3 litery zamiast dwie oraz o wiele bardziej skomplikowany wzór działania. Postępowanie w tym algorytmie działa tak, że najpierw odwzorowujemy poszukiwany wzorzec p w liczbę całkowitą H p za pomocą tzw. funkcji haszującej, ustawiamy okno wzorca na początku tekstu i haszujemy je przy pomocy tej samej funkcji w liczbę całkowitą H s i porównujemy dwie liczby H.Jeśli są równe, to oznacza to, iż wzorzec i jego okno są do siebie podobne z dokładnością do haszów. Główną zaletą takiego sposobu hashowania jest możliwość wyznaczenia wartości skrótu kolejnego podciągu na podstawie podciągu poprzedniego poprzez wykonanie stałej liczby operacji niezależnej od długości tego podciągu.

Zad. 2

Szyfr Cezara

#include <iostream>

#include <string>

using namespace std;

int main( )

{

string s;

int i;

// odczytujemy wiersz znaków

getline ( cin, s );

// zamieniamy małe litery na duże

// i kodujemy szyfrem cezara

for( i = 0; i < s.length( ); i++ )

{

s [ i ] = toupper ( s [ i ] );

if( ( s [ i ] >= 'A' ) && ( s [ i ] <= 'Z' ) ) s [ i ] = char ( 65 + ( s [ i ] - 62 ) % 26 );

}

// wypisujemy zaszyfrowany tekst

cout << s << endl << endl;

return 0;

}

Szyfrowanie z pseudolosowym dostępem

#include <iostream>

#include <string>

using namespace std;

int main( )

{

string s;

int i;

unsigned long long X, a, m, c;

// definiujemy generator LCG

m = 3956280000ull;

a = 1978141ull;

c = 1309ull;

// odczytujemy klucz i wiersz znaków

cin >> X; cin.ignore ( 256, '\n' );

getline ( cin, s );

// szyfrujemy

for( i = 0; i < s.length( ); i++ )

{

// obliczamy kolejną liczbę pseudolosową

X = ( a \* X + c ) % m;

// szyfrujemy literkę

s [ i ] = toupper ( s [ i ] );

if( ( s [ i ] >= 'A' ) && ( s [ i ] <= 'Z' ) ) s [ i ] = 65 + ( s [ i ] - 65 + X % 26 ) % 26;

}

// wypisujemy zaszyfrowany tekst

cout << s << endl << endl;

return 0;

}

Szyfr przestawieniowy

#include <iostream>

#include <string>

using namespace std;

int main( )

{

string s;

unsigned i;

// odczytujemy tekst

getline ( cin, s );

// zamieniamy miejscami litery

for( i = 0; i < s.length( ) - 1; i += 2 )

swap ( s [ i ], s [ i + 1 ] );

// wyświetlamy wynik

cout << s << endl;

return 0;

}

Szyfr Enigmy

#include <iostream>

#include <string>

using namespace std;

// definicje elementów Enigmy

const string pierscien\_szyfr [ 5 ] = {"EKMFLGDQVZNTOWYHXUSPAIBRCJ",

"AJDKSIRUXBLHWTMCQGZNPYFVOE",

"BDFHJLCPRTXVZNYEIWGAKMUSQO",

"ESOVPZJAYQUIRHXLNFTGKDCMWB",

"VZBRGITYUPSDNHLXAWMJQOFECK"};

const string przeniesienie = "RFWKA";

const string beben\_odwr = "YRUHQSLDPXNGOKMIEBFZCWVJAT";

int main( )

{

int pierscien [ 3 ], i, j, k, n, c;

bool ruch;

string szyfr, s, lacznica;

// odczytujemy konfigurację pierścieni szyfrujących

cin >> n;

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

{

pierscien [ i ] = ( n % 10 ) - 1; // numer pierścienia na i-tej pozycji

n /= 10;

}

// odczytujemy położenia początkowe pierścieni

cin >> szyfr;

for( i = 0; i < szyfr.length( ); i++ ) szyfr [ i ] = toupper ( szyfr [ i ] );

// odczytujemy stan łącznicy wtyczkowej

cin >> s;

for( i = 0; i < s.length( ); i++ ) s [ i ] = toupper ( s [ i ] );

lacznica = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

for( i = 0; i < s.length( ) - 1; i += 2 )

{

lacznica [ s [ i ] - 65 ] = s [ i + 1 ];

lacznica [ s [ i + 1 ] - 65 ] = s [ i ];

}

// odczytujemy szyfrogram

cin.ignore ( 256, '\n' );

getline ( cin, s );

for( i = 0; i < s.length( ); i++ ) s [ i ] = toupper ( s [ i ] );

// szyfrujemy/rozszyfrowujemy szyfrogram

for( i = 0; i < s.length( ); i++ )

{

// najpierw ruch pierścieni szyfrujących

for( ruch = true, j = 2; ruch && ( j >= 0 ); j-- )

{

ruch = ( szyfr [ j ] == przeniesienie [ pierscien [ j ] ] );

szyfr [ j ] = 65 + ( szyfr [ j ] - 64 ) % 26;

}

// pobieramy znak szyfrogramu

c = s [ i ];

// przechodzimy przez łącznicę wtyczkową

c = lacznica [ c - 65 ];

// przechodzimy przez pierścienie w kierunku do bębna odwracającego

for( j = 2; j >= 0; j-- )

{

k = szyfr [ j ] - 65;

c = pierscien\_szyfr [ pierscien [ j ] ][ ( c - 65 + k ) % 26 ];

c = 65 + ( c - 39 - k ) % 26;

}

// przechodzimy przez bęben odwracający

c = beben\_odwr [ c - 65 ];

// wracamy ścieżką powrotną

for( j = 0; j < 3; j++ )

{

k = szyfr [ j ] - 65;

c = 65 + ( c - 65 + k ) % 26;

for( n = 0; pierscien\_szyfr [ pierscien [ j ] ][ n ] != c; n++ );

c = 65 + ( 26 + n - k ) % 26;

}

// przechodzimy przez łącznicę wtyczkową

c = lacznica [ c - 65 ];

// uaktualniamy szyfrogram

s [ i ] = c;

}

// wyświetlamy szyfrogram

cout << s << endl << endl;

return 0;

}

Szyfr RSA

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <time.h>

using namespace std;

// Funkcja czeka na dowolny klawisz i czyści ekran

//------------------------------------------------

void czekaj ( void )

{

char c [ 1 ];

cout << "\nZapisz te dane\n\n";

cin.getline ( c, 1 );

cin.getline ( c, 1 );

for( int i = 1; i < 500; i++ ) cout << endl;

}

// Funkcja obliczająca NWD dla dwóch liczb

//----------------------------------------

int nwd ( int a, int b )

{

int t;

while( b != 0 )

{

t = b;

b = a % b;

a = t;

};

return a;

}

// Funkcja obliczania odwrotności modulo n

//----------------------------------------

int odwr\_mod ( int a, int n )

{

int a0, n0, p0, p1, q, r, t;

p0 = 0; p1 = 1; a0 = a; n0 = n;

q = n0 / a0;

r = n0 % a0;

while( r > 0 )

{

t = p0 - q \* p1;

if( t >= 0 )

t = t % n;

else

t = n - ( ( -t ) % n );

p0 = p1; p1 = t;

n0 = a0; a0 = r;

q = n0 / a0;

r = n0 % a0;

}

return p1;

}

// Procedura generowania kluczy RSA

//---------------------------------

void klucze\_RSA( )

{

const int tp [ 10 ] = {11, 13, 17, 19, 23, 29, 31, 37, 41, 43};

int p, q, phi, n, e, d;

cout << "Generowanie kluczy RSA\n"

"----------------------\n\n";

// generujemy dwie różne, losowe liczby pierwsze

do

{

p = tp [ rand( ) % 10 ];

q = tp [ rand( ) % 10 ];

} while( p == q );

phi = ( p - 1 ) \* ( q - 1 );

n = p \* q;

// wyznaczamy wykładniki e i d

for( e = 3; nwd ( e, phi ) != 1; e += 2 );

d = odwr\_mod ( e, phi );

// gotowe, wypisujemy klucze

cout << "KLUCZ PUBLICZNY\n"

"wykladnik e = " << e

<< "\n modul n = " << n

<< "\n\nKLUCZ PRYWATNY\n"

"wykladnik d = " << d << endl;

czekaj( );

}

// Funkcja oblicza modulo potęgę podanej liczby

//---------------------------------------------

int pot\_mod ( int a, int w, int n )

{

int pot, wyn, q;

// wykładnik w rozbieramy na sumę potęg 2

// przy pomocy algorytmu Hornera. Dla reszt

// niezerowych tworzymy iloczyn potęg a modulo n.

pot = a; wyn = 1;

for( q = w; q > 0; q /= 2 )

{

if( q % 2 ) wyn = ( wyn \* pot ) % n;

pot = ( pot \* pot ) % n; // kolejna potęga

}

return wyn;

}

// Procedura kodowania danych RSA

//-------------------------------

void kodowanie\_RSA( )

{

int e, n, t;

cout << "Kodowanie danych RSA\n"

"--------------------\n\n"

"Podaj wykladnik = "; cin >> e;

cout << " Podaj modul = "; cin >> n;

cout << "----------------------------------\n\n"

"Podaj kod RSA = "; cin >> t;

cout << "\nWynik kodowania = " << pot\_mod ( t, e, n ) << endl;

czekaj( );

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// \*\* Program główny \*\*

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main( )

{

int w;

srand ( ( unsigned )time ( NULL ) );

do

{

cout << "System szyfrowania danych RSA\n"

"-----------------------------\n"

" (C)2012 mgr Jerzy Walaszek\n\n"

"MENU\n"

"====\n"

"[ 0 ] - Koniec pracy programu\n"

"[ 1 ] - Generowanie kluczy RSA\n"

"[ 2 ] - Kodowanie RSA\n\n"

"Jaki jest twoj wybor? ( 0, 1 lub 2 ) : ";

cin >> w;

cout << "\n\n\n";

switch( w )

{

case 1 : klucze\_RSA( ); break;

case 2 : kodowanie\_RSA( ); break;

}

cout << "\n\n\n";

} while( w != 0 );

return 0;

}

| Szyfr | Tekst | Wynik z konsoli | Tekst 2 | Wynik z konsoli 2 |
| --- | --- | --- | --- | --- |
| Szyfr Cezara | Silence is half consent | VLOHQFH LV D KDOI FRQVHQW | Pen is mightier than a sword | SHQ LV PLJKWLHU WKDQ D VZRUG |
| Szyfrowanie z pseudolosowym dostępem | Still water runs deep | OKMWX KZFVLR AAW SUXF | What goes around comes around | MKIK BSRK SCABHY BMZMZ VRXIMF |
| Szyfr przestawieniowy | A thing begun is a thing half done | Ahtni gebug nsia h la foden | Beauty is in the eye of the beholder | eBuayti snit ehe eyo fht eebohdlre |
| Szyfr Enigmy | Early bird catches the worm | XKDUBHFRMIEJQNWNVPXXNVTDALI | Fall seven times. Stand up eight | SKMUSHZWSBEECCMQRFWXQYHXLAWXMLBR |
| Szyfr RSA | 128 | 219 | 1024 | 302 |

Szyfr cezara jest bardzo prostym szyfrem, który sprawny umysł jest w stanie odczytać w chwilę bez użycia wzoru na ten szyfr. Jego działanie opiera się na podstawieniu pod literę 3 literę położoną od niej dalej w alfabecie co czyni też szyfr Cezara szyfrem podstawieniowym. Jest szybki w zakodowaniu jednak tak samo szybki w złamaniu ponadto w obecnych czasach dobrze rozpoznawalny i znany, dlatego nie jest bezpieczną opcją w przypadku gdy chce się mieć szyfrowaną wiadomość. Może być za to świetnym ćwiczeniem umysłowym dla małych dzieci poznających alfabet.