Algorytmy szyfrowania danych

Czasy po 1950 roku

1) Znany algorytm szyfrowania danych, tajne parametry

```
public String koduj (String x)     //zamiana znaków z
// napisu 'x' z podanego zakresu [' ', '~'](kod ASCII
//[#32,..,#126]) na ich kody zależnie od pozycji znaku
// i długości napisu
{           // koduj
String y = "";
int n = x.length();
int i;
for (i = 0; i < n; i++)
   {
    int k = (int)x.charAt(i);
                // zmiana znaku na liczbę
    if ((k > 31) && (k < 127))
    {
     k = k + n + i;   //f(k) = ...np. k + 3*n + 5*i + 7
     while (k > 126) {k = k – 94};
     y += (char)k;
               // zmiana liczby na znak
    }
   }
 return y;
}     // koduj
```

Napis:	N	I	C	Długość stringu: n = 3
Kody ASCII:	78	73	67	
F(k):	82	78	73	
Zakodowany:	R	N	I	

```
public String dekoduj (String x)     //zamiana znaków z
// napisu ‘x’ z podanego zakresu [‘ ‘, ‘~’](kod ASCII
//[#32,..,#126]) na ich kody zależnie od pozycji znaku
// i długości napisu
{           // dekoduj
String y = ””;
int n = x.length();
int i;
for (i = 0; i < n; i++)
   {
    int k = (int)x.charAt(i);
                // zmiana znaku na liczbę
    if ((k > 31) && (k < 127))
    {
     k = k – n - i; //f-1(k) = ... np. k - 3*n - 5*i - 7
     while (k < 33) {k = k + 94};
     y +=(char)k;
               // zmiana liczby na znak
    }
   }
 return y;
}     // dekoduj
```

2) Znany algorytm szyfrowania danych, tajne hasło

Wykorzystywana jest operacja **xor** na liczbach w zapisie binarnym (różne cyfry dają 1, takie same dają 0).

x : 10110010
y : 10011000
x xor y : 00101010

Własność operacji xor : (x xor y) = z ⇔ (z xor y) = x

Konwersja liczb binarnych na dziesiętne:

Liczba binarna (czyli dwójkowa) zapisana jest jako ciąg bitów:

$$(b_{k-1} \ldots b_1 b_0)_2$$

gdzie $b_i = 1$ lub $b_i = 0$

Konwersja liczby binarnej na dziesiętną polega na obliczeniu wartości wielomianu, którego współczynnikami są kolejne cyfry (najbardziej znacząca cyfra odpowiada współczynnikowi przy najwyższej potędze), dla argumentu x=2.

Wartość liczby zapisanej binarnie definiuje się:

$$(b_{k-1} \ldots b_1 b_0)_2 = b_{k-1} 2^{k-1} + \ldots + b_1 2 + b_0$$

Np.

$$(1010011)_2 = 1 * 2^6 + 0 * 2^5 + 1 * 2^4 + 0 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0 = 64 + 16 + 2 + 1 = 83$$

Algorytm:

we : liczba binarna zapisana w tablicy B[0...k-1], gdzie bit najmniej znaczący znajduje się pod indeksem 0.
wy: liczba naturalna n;

```
n = B[0];
p = 2;
FOR (i=1; i<k; i++)
 {
   n = n + p * B[i];
   p = p * 2;
  }
```

Konwersja liczb dziesiętnych na binarne:

Algorytm:
we : liczba naturalna n (int)
wy: zapis bitowy liczby w tablicy B[0...k-1], gdzie
bit najmniej znaczący znajduje się pod indeksem 0

```
i = 0;
while  (n > 0)
 {
  B[i] = n % 2; // operacja modulo - reszta z
  i = i + 1;                          dzielenia
  n = n / 2;     // dzielenie całkowite – typ int
 }
```

Przykład: tekst: **nauka informatyki** hasło: **xyz**
xyzxyzxyzxyzxyzxy

tekst zaszyfrowany **;@#$%^%&*(()_++*%**

```
public String kodekoduj (String x, String h)
// zamiana znaków z napisu ‘x’ z podanego zakresu
// [‘ ‘,‘~’] (kod ASCII [#32,..,#126]) na ich kody
// zależnie od kolejnych znaków hasła ‘h’
{           // kodekoduj
String y = ””;
int m = h.length();
if (m > 0)
 {
  int n = x.length();
  int i = 0; int j = 0;
  for (i = 0; i < n; i++)
   {if (j = m) {j = j – m};
    int k = (int)x.charAt(i); int l = (int)h.charAt(j);
    if ((k > 31)&&(k < 127)&&(l > 31)&&(l < 127))
      {
       k = k – 32; l = l – 32;
       k = k ^ l;  k = k + 32; // xor
       y +=(char)k;
      }
     j = j + 1;
    }
  }
 return y;
}         // kodekoduj
```

Kompresja danych

Kompresja pliku o długości 100 000 znaków.

Częstotliwość występowania znaków:
A - 44 000 wystąpień, B – 16 000, C – 14 000,
D – 11 000, E – 9 000, F – 6 000.

Problem: zaprojektować kod binarny, w którym każdy znak jest reprezentowany przez ustalony ciąg bitów.

1) Jednoznaczne kodowanie informacji.

Kod1:

A = 0	napis = **1011101101010010100 1**
B = 1	można interpretować jako:
C = 00	**BABBBABBABABAABABAAB**
D = 01	**E FB D EB D CB D CB**
E = 10	**E F E F D D C E E D**
F = 11	**BA FBA FAB DAA EBA D**
	E F EB EBABA DABA D

Kod jednoznaczny (Kod prefiksowy):

Kod żadnego znaku nie jest prefiksem (początkowym podsłowem) kodu innego znaku.

Fakt:
Optymalna kompresja pliku metodą kodowania znaków zawsze może być osiągnięta za pomocą pewnego kodu prefiksowego.

2) Kod2 – zawierający słowa o jednakowej długości:

A = 000 B = 001 C = 010 D = 011 E = 100
F = 101

Dla kodu stałej długości do reprezentacji 6 znaków potrzebne są 3 bity na znak ➔
długość pliku po zakodowaniu: 300 000 bitów (+ około 1000 bitów na słownik) = 37 625 bajtów

Kod3 – słowa kodowe posiadają po cztery bity:

A = 0001 B = 0010 C = 0100 D = 0111
E = 1000 F = 1011

długość pliku po zakodowaniu: 400 000 bitów (+ około 1000 bitów na słownik) = 50 125 bajtów

Kod4– zawierający słowa o zmiennej długości:
(częściej występujące znaki są reprezentowane za pomocą krótszych słów kodowych)

A = 10 B = 01 C = 110 D = 001 E = 111
F = 000

długość pliku po zakodowaniu: 60*2 + 40*3 = 240 ⇒
240 000 bitów (+1000 bitów na słownik)= 30 125 bajtów

Rozwiązanie optymalne (nie jedyne):
Kod5 - zawierający słowa o zmiennej długości:

A = 0 B = 101 C = 100 D = 111 E = 1101
F = 1100
daje długość pliku zakodowanego (skompresowanego)
44*1 + 41*3 + 15*4 = 227 ⇒ 227 000 + 1000 bitów =
28 500 bajtów.

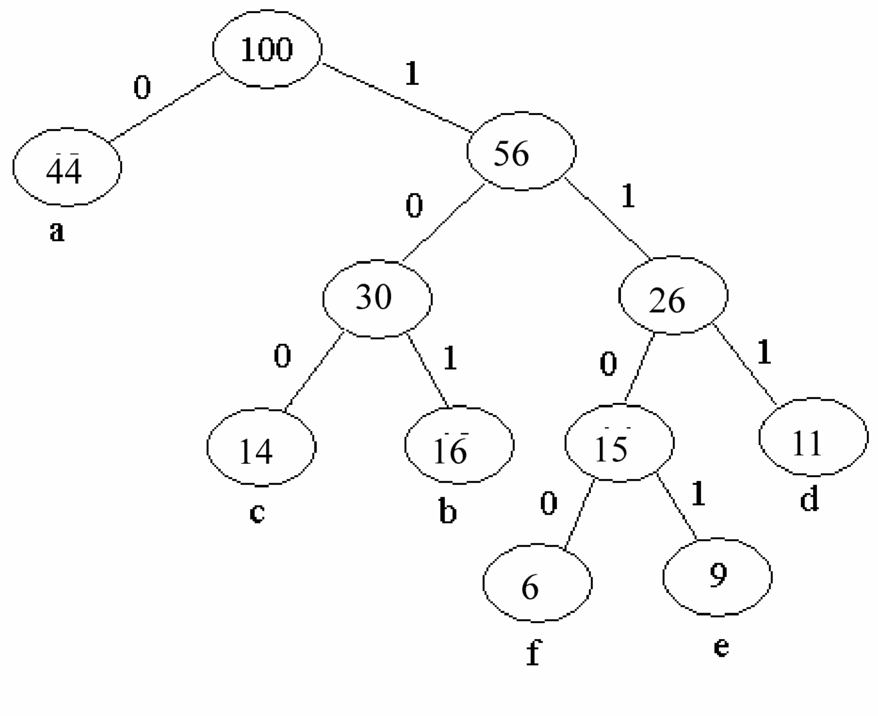
Kodowanie tekstu za pomocą kodów binarnych (prefiksowych) – konkatenacja kodów kolejnych znaków w pliku

W kodzie 5   ABC to 0101100

Dekodowanie- kolejne kody zamieniamy na znaki.

Reprezentacja kodu (ułatwia rozpoznawanie kodów podczas dekodowania):

- drzewo binarne
- liśćmi są poszczególne znaki
- krawędź idąca w lewo ma etykietę 0, w prawo: 1
- kod znaku - ciąg etykiet na ścieżce od korzenia do liścia tego znaku
- etykiety węzłów – ilość wystąpień znaków poddrzewa



a=0   b=101   c=100   d=111   e=1101   f=1100

Koszt kodu:

liczba bitów wymagana do zakodowania pliku z zadanym rozkładem liczności wystąpień znaków:

$$\mathbf{B(T) = \sum f(c)\, d_T(c),}$$

(suma po wszystkich znakach $c \in C$),

gdzie:

C – alfabet, T – drzewo kodu

f(c) – liczba wystąpień znaku 'c'

d_T(c) – głębokość (czyli odległość od korzenia) liścia-znaku 'c' – długość kodu ‵c′

Optymalny kod – reprezentowany przez regularne drzewo binarne – każdy węzeł wewnętrzny ma dwoje dzieci

Drzewo dla kodu2

A = 000 B = 001 C = 010 D = 011 E = 100 F = 101

3) tworzenie drzewa regularnego – według częstości występowania znaków

```
                         100%
                        ABCDEF
          *---------------------------------*
          50%                                50%
          AF                                BCDE
  *----------------*                    *----------*
 44%               6%                  25%          25%
  A                F                   BE           CD
                                  *-------*    *-------*
                                 16%      9%  14%      11%
                                  B       E    C        D
```

Kod6:

A = 11 B = 011 C = 001 D = 000 E = 010 F = 10

długość pliku zakodowanego 50*2 + 50*3 = 250 ⇒
250 000 + 1000 bitów = 31 375 bajtów.

tworzenie drzewa regularnego – według najczęściej występującego znaku

```
             ABCDEF
  *-------------------------*
 44%                      BCDEF
  A              *--------------------*
                16%                 CDEF
                 B               *----------*
                                14%        DEF
                                 C     *-------*
                                      11%       EF
                                       D    *------*
                                            9%     6%
                                            E      F
```

Kod7:
A = 1 B = 01 C = 001 D = 0001 E = 00001
F = 00000

długość pliku zakodowanego 44*1 + 16*2 + 14*3 + 11*4 + 15*5 = 237 ⇒ 237 000 + 1000 bitów = 29 750 bajtów

4) Kod Huffmana - optymalny kod prefiksowy
kod elementu zależy od częstości jego występowania

Idea algorytmu Huffmana:
- utworzyć listę węzłów reprezentujących znaki
- w każdym kroku pobierać dwa węzły o najmniejszej częstości
- tworzyć nowy węzeł (wewnętrzny) jako poprzednik tych dwóch
- pobrane dwa zastępować nowym, o częstości równej sumie częstości w tych dwóch

```
Dane:
C – alfabet (zbiór występujących znaków)
f(c) – liczba wystąpień znaku 'c' (lub częstość)

n = |C|;
Q = C;
for (i = 1; i < n; i++)
 {
  utwórz nowy węzeł z;
  z.left  = x = Q.extract_min;
  z.right = y = Q.extract_min;
  z.freq = x.freq + y.freq;
  Q.insert(z);
  }
return Q.extract_min;
 //zwraca korzeń drzewa – reprezentacji kodu


44%       16%       14%       11%       9%       6%
 A         B         C         D        E        F

44%       16%        15%     14%       11%
 A         B      *-------*   C         D
                  E       F

44%       25%        16%       15%
 A     *-------*      B     *-------*
       C       D            E       F

44%              31%                   25%
 A            *-----------*         *---------*
             B        *-------*    C         D
                      E       F

               56%                     44%
         *-------------*                A
    *--------*     *-------*
    B    *-----*   C       D
         E     F
```

```
                       100%
                 *---------------------*
           *---------------*           A
    *----------*        *--------*
    B       *-----*     C        D
            E     F
```

Kod Huffmana:

A = 0 B = 111 C = 101 D = 100 E = 1101 F = 1100

długość pliku zakodowanego 44*1 + 41*3 + 15*4 = 227
⇒ 227 000 + 1000 bitów = 28 500 bajtów

System kryptograficzny z kluczem jawnym RSA

Współczesne systemy kryptograficzne bazują na dużych liczbach pierwszych – można je łatwo znaleźć

Bezpieczeństwo systemów – rozkład na czynniki iloczynu dużych liczb pierwszych (czynniki – nietrywialne dzielniki liczby – np. dla 20 są to: 2, 4, 5, 10)

Liczby pierwsze i złożone – 0, 1, ujemne liczby całkowite nie są ani pierwsze ani złożone.

Twierdzenie
Dla dowolnej liczby całkowitej **a** i dowolnej dodatniej liczby całkowitej **n** istnieją jednoznacznie wyznaczone liczby całkowite **q** i **r** takie, że $\mathbf{0} \le \mathbf{r} < \mathbf{n}$ i $\mathbf{a = qn + r}$
(r = a mod n)

d - największy wspólny dzielnik a i b
(np. nwd (24, 30) = 6)

Jeśli d|a i d|b ⇒ d|(a+b), d|(a-b), d|(ax+by),
gdzie x i y – dowolne liczby całkowite

Dwie liczby całkowite a i b są względnie pierwsze – ich jedynym wspólnym dzielnikiem jest 1, nwd(a,b)=1, (np. 8 i 15)

Algorytm Euklidesa obliczania nwd dla dwóch dowolnych nieujemnych liczb całkowitych a i b (300 p.n.e.)

Własność nwd: nwd (a, b) = nwd (b, a mod b)

```
public int Euclid(int a, int b)
 {  // zwraca d = nwd (a, b)
   if (b == 0) {return a};
   else {return Euclid(b, a % b)};
 }
```

```
Euclid(30,21) = Euclid(21,9) = Euclid(9,3) =
Euclid(3,0) = 3
```

Rozszerzony algorytm Euklidesa – znajduje całkowitoliczbowe współczynniki takie, że
 d = nwd (a, b) = ax + by

```
public int[] Ext_Euclid (int a, int b)
{  // zwraca d, x, y; d = nwd (a, b)= ax + by
  if (b == 0)
   {
     int[] dxy = {a, 1, 0};
     return dxy;
    }
  int[] dxy = Ext_Euclid(b, a % b);
  int x = dxy[1];
  dxy[1] = dxy[2]; dxy[2] = x – (a/b)*dxy[2];
  return dxy;
}
```

a	b	a/b	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	-	3	1	0

nwd(99,78)=3 =99(-11)+78*14

Kryptosystem z kluczem jawnym RSA
(1977, Rivest-Szamir-Adleman)
– szyfrowanie przesyłanych informacji z dołączeniem podpisu cyfrowego (łatwy do sprawdzenia, niemożliwy do podrobienia)
Każdy użytkownik posiada klucz jawny i klucz tajny.

Zasada działania: mamy trzy liczby naturalne:
M – duża liczba, jawna, J – klucz jawny, T – klucz tajny

Liczby te mają następujące własności:
Dla każdej liczby naturalnej I, $1 < I < M$ zachodzi:

$X = I^{J} \text{ MOD } M \Rightarrow I = X^{T} \text{ MOD } M$ --> $T(J(I)) = I$

i odwrotnie:

$Y = I^{T} \text{ MOD } M \Rightarrow I = Y^{J} \text{ MOD } M$

Kodowanie tekstu polega na zamianie ciągu znaków na liczbę naturalną $I < M$.
Wysyłanie wiadomości I:

Liczby Alicji: M_A , J_A – klucz jawny, T_A – klucz tajny
Liczby Bartka: M_B , J_B – klucz jawny, T_B – klucz tajny

Bartek wysyła wiadomość I z podpisem B do Alicji.

$J_A (I) = \text{szyfr}(I, J_A , M_A) = I^{JA} \text{ MOD } M_A = X$
$J_A (B) = \text{szyfr}(B, J_A , M_A) = B^{JA} \text{ MOD } M_A = Y$
$T_B (B) = \text{szyfr}(B, T_B, M_B) = B^{TB} \text{ MOD } M_B = Z$
Z –podpis elektroniczny

Alicja deszyfruje:
$T_A (X) = \text{szyfr}(X, T_A , M_A) = X^{TA} \text{ MOD } M_A = I$
$T_A (Y) = \text{szyfr}(Y, T_A , M_A) = Y^{TA} \text{ MOD } M_A = B$
$J_B (Z) = \text{szyfr}(Z, J_B, M_B) = Z^{JB} \text{ MOD } M_B = B$ (B=B) ???

Problemy:

1) Potęgowanie modularne – efektywnie obliczyć a^b MOD m, gdzie m – duża liczba

Metoda wielokrotnego podnoszenia do kwadratu

```
public long potmod (long a, long b, long m)
{
 long y = 1; // y = a^b MOD m
 while (b > 0)
  {
   if (b%2 != 0){y = (y*a) % m};
   b = b / 2;
   a = (a*a) % m;
  }
 return y;
}
```

Przykład: 3^200 MOD 1000

y	a	b	m
1 (a^0)	3	200	1000
-	9 ^2	100	
-	81 ^4	50	
-	(6)561 ^8	25	
561 (a^8)	(314)721 ^16	12	
-	(519)841 ^32	6	
-	(707)281 ^64	3	
561*281=(157)641 (a^72)	(78)961 ^128	1	
641*961=(616)001 (a^200)	(923)521 ^256	0	

2) Wyznaczyć liczby M, J, T

- wybierz dwie duże liczby pierwsze p i q (100-200 cyfr)
- oblicz M = p*q
- oblicz F = (p-1)*(q-1)
- wybierz niewielka nieparzysta liczbę J względnie pierwszą z F (nwd (J, F) = 1)
- oblicz T tak aby (T*J) MOD F = 1
  (czyli istnieje X takie, że T*J = X*F +1)

Znając M i J niemożliwe jest znalezienie T w rozsądnym czasie – trzeba znaleźć p i q aby wyznaczyć F
Rozkładanie dużych liczb (M) na czynniki jest bardzo trudne

Przykłady:

p = 7
q = 13
M = 91
F = 72
J = 5, 7, 17
T = 29, 31, 17

p = 17
q = 23
M = 391
F = 352
J = 5, 7, 15
T= 141, 151, 47

Ext_Euclid (72, 5) (J*T = X*F +1)

F	J	F/J	d	X	T
72	5	14	1	-2	29
5	2	2	1	1	-2
2	1	2	1	0	1
1	0	-	1	1	0

nwd(72,5)= 1 = 72*(-2)+5*29

p = 47
q = 59
M = 2773
F = 2668
J = 7, 9, 13, 17, 19
T = 367, 593, 821, 157, 983

p = 1951
q = 1997
M = 3896147
F = 3892200
J = 59
T = 131939

Algorytm wyznaczania klucza tajnego tak aby T było liczbą dodatnią –wyznaczenie dwóch par współczynników
d = nwd(F, J)= F*x1 + J*y1 = F*x2 + J*y2

Jeśli d==1 i y1 > 0 to T= y1;
Jeśli d==1 i y2 >0 to T= y2;
Jeśli J i F względnie pierwsze to nie może być d>1

```
public Ext_NWD (int a, int b)
{ // największy wspólny dzielnik z rozkładem na sumy
 int z = a; int x1 = 1; int y1 = 0;
 int d = b; int x2 = 0; int y2 = 1;
 while (z != d)
  {        // z = x1*a+y1*b, d = x2*a+y2*b
   if (z > d)
    {
     z = z-d; x1 = x1-x2; y1 = y1-y2;
    }
   else
    {
     d = d-z; x2 = x2-x1; y2 = y2-y1;
    }
  }        // z = NWD(a,b) = d
}
```

a	b	z	x1	y1	d	x2	y2
72	5						
-	-	72	1	0	5	0	1
-	-	67	1	-1			
-	-	62	1	-2			
...	...						
-	-	12	1	-12			
-	-	7	1	-13			
-	-	2	1	-14			
-	-				3	-1	15
-	-				1	-2	29
-	-	1	3	-43			

72 * 3 + 5 * (-43) = 1 = 72 * (-2) + 5 * 29