Cozma Laura - Elena
244

Examen

1. Cuvântul notat cu Y, corespunzător cuvântului x=COLA, este:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
y =	0	0	0	1	1	0	1	1	1	1	0	1	1	0	0	0	0	0	0	1

Deoarece y este format din $4 \cdot 5 = 20$ biți, numărul de biți de paritate este 5.

Vom genera codul Hamming (25,20).

Scriem în baza 2 nr. de la 1 la 25

	1 →	1
	2 →	10
d_1	3 →	11
	4 →	100
d_2	5 →	101
d_3	6 →	110
d_4	7 →	111
	8 →	1000
d_5	9 →	1001
d_6	10 →	1010
d_7	11 →	1011
d_8	12 →	1100
d_9	13 →	1101
d_{10}	14 →	1110
d_{11}	15 →	1111
	16 →	10000
d_{12}	17 →	10001
d_{13}	18 →	10010
d_{14}	19 →	10011
d_{15}	20 →	10100
d_{16}	21 →	10101
d_{17}	22 →	10110
d_{18}	23 →	10111
d_{19}	24 →	11000
d_{20}	25 →	11001

Fiecare poziție putere a lui 2 e un bit de paritate. Al n-lea bit de paritate acoperă pozițiile care au bitul n cel mai semnificativ egal cu 1.

Deci:

$r_1 = d_1 + d_2 + d_4 + d_5 + d_7 + d_9 + d_{11} + d_{12} + d_{14} + d_{16} + d_{18} + d_{20}$

$r_1 = 0+0+1+1+1+1+0+1+0+0+0+1 = 0$

$r_2 = d_1 + d_3 + d_4 + d_6 + d_7 + d_{10} + d_{11} + d_{13} + d_{14} + d_{17} + d_{18}$

$r_2 = 0+0+1+0+1+1+0+1+0+0+0 = 0$

$r_3 = d_2 + d_3 + d_4 + d_8 + d_9 + d_{10} + d_{11} + d_{15} + d_{16} + d_{17} + d_{18}$

$r_3 = 0+0+1+1+1+1+0+0+0+0+0 = 0$

$r_4 = d_5 + d_6 + d_7 + d_8 + d_9 + d_{10} + d_{11} + d_{19} + d_{20}$

$r_4 = 1+0+1+1+1+1+0+0+1 = 0$

$r_5 = d_{12} + d_{13} + d_{14} + d_{15} + d_{16} + d_{17} + d_{18} + d_{19} + d_{20}$

$r_5 = 1+1+0+0+0+0+0+0+1 = 1$

Rezultă codul Hamming:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	0	0	0	0	0	1	0	1	0	1	1	1	1	0	1	1	1	0	0	0	0	0	0	1

1) Se transmite cuvântul y, iar el va fi recepționat fără erori

O recepționare fără erori ar însemna să primim:

0 0 0 0 0 0 1 0 1 0 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1

Reconstituim biții de date, extrăgându-i pe cei de paritate.

Obținem: 0 0 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 0 1

Acum verificăm paritățile. Dacă rezultatele vor fi toate 0, înseamnă că a fost recepționat fără erori. Dacă nu, înseamnă că a existat o eroare, iar rezultatul va indica poziția greșită.

Adunăm biții de paritate cu biții folosiți la construirea lor:

$p_1 + d_1 + d_2 + d_4 + d_5 + d_7 + d_9 + d_{11} + d_{12} + d_{14} + d_{16} + d_{18} + d_{20} =$
$0 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 1 + 0 + 0 + 0 + 1 = 0$

$p_2 + d_1 + d_3 + d_4 + d_6 + d_7 + d_{10} + d_{11} + d_{13} + d_{14} + d_{17} + d_{18} =$
$0 + 0 + 0 + 1 + 0 + 1 + 1 + 0 + 1 + 0 + 0 + 0 = 0$

$p_3 + d_2 + d_3 + d_4 + d_8 + d_9 + d_{10} + d_{11} + d_{15} + d_{16} + d_{17} + d_{18} =$
$0 + 0 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 = 0$

$p_4 + d_5 + d_6 + d_7 + d_8 + d_9 + d_{10} + d_{11} + d_{19} + d_{20} =$
$0 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 = 0$

$p_5 + d_{12} + d_{13} + d_{14} + d_{15} + d_{16} + d_{17} + d_{18} + d_{19} + d_{20} =$
$1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 1 = 0$

Toate valorile sunt 0, deci nu am avut nicio eroare.

2) Se transmite ~~cu erori~~ și e recepționat cu erori:

Presupunem că am recepționat:

0 0 0 0 1 0 1 0 1 0 1 1 1 1 0 1 1 1 0 0 0 0 0 0 1 (am schimbat bitul 5).

Reconstituim biții de date: 0 1 0 1 1 0 1 1 1 1 0 1 1 0 0 0 0 0 1

Verificăm paritățile:

$p_1 + d_1 + d_2 + d_4 + d_5 + d_7 + d_9 + d_{11} + d_{12} + d_{14} + d_{16} + d_{18} + d_{20} =$
$= 0 + 0 + 1 + 1 + 1 + 1 + 1 + 0 + 1 + 0 + 0 + 0 + 1 = 1$

$p_2 + d_1 + d_3 + d_4 + d_6 + d_7 + d_{10} + d_{11} + d_{13} + d_{14} + d_{17} + d_{18} =$
$0 + 0 + 0 + 1 + 0 + 1 + 1 + 0 + 1 + 0 + 0 + 0 = 0$

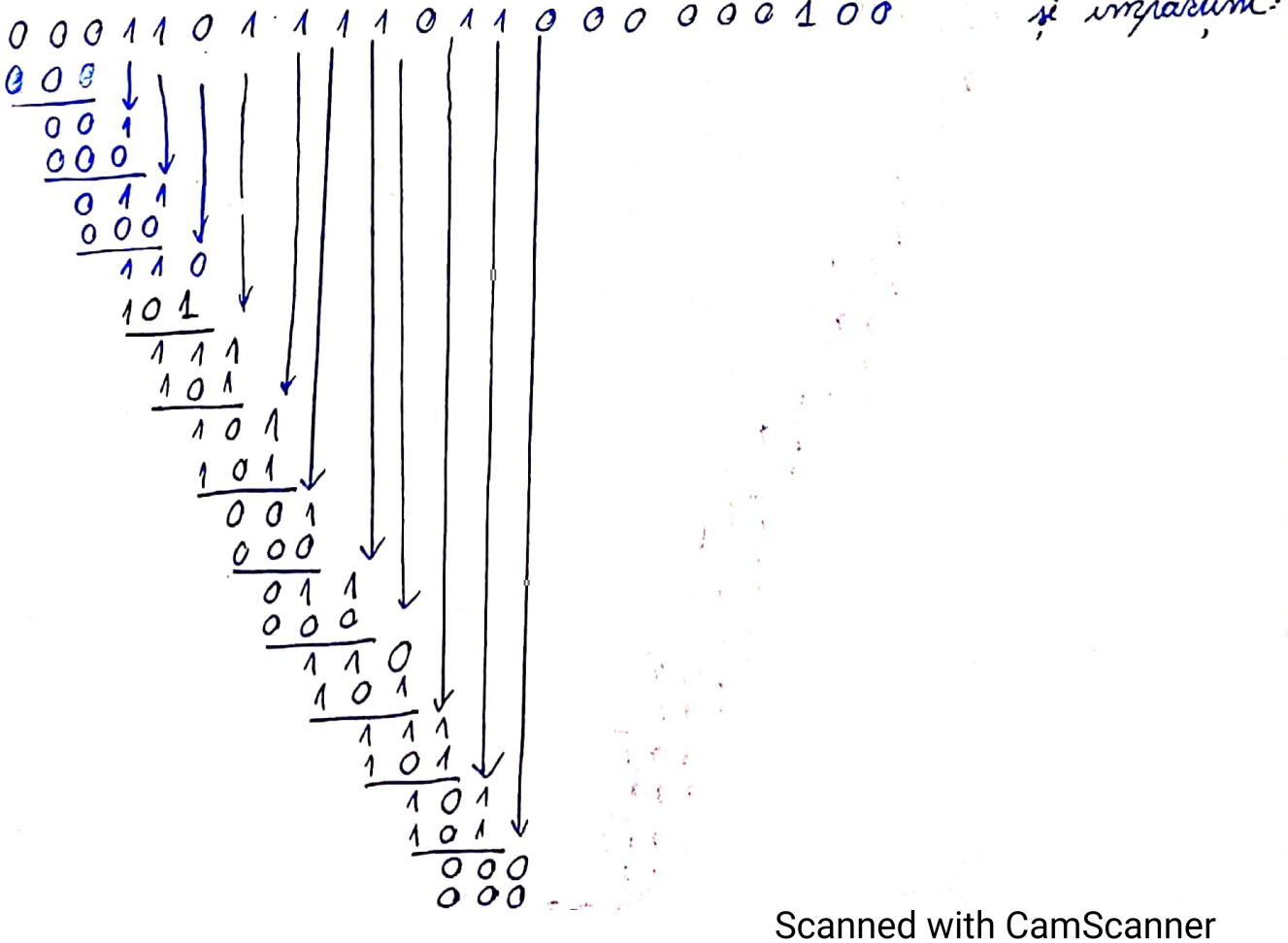
$p_3 + d_2 + d_3 + d_4 + d_8 + d_9 + d_{10} + d_{11} + d_{15} + d_{16} + d_{17} + d_{18} =$
$0 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 = 1$

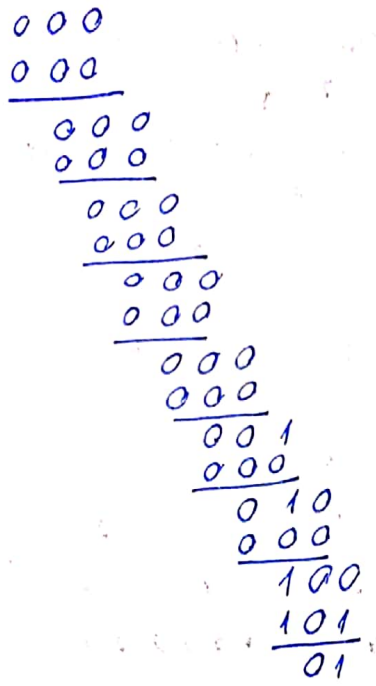
$p_4 + d_5 + d_6 + d_7 + d_8 + d_9 + d_{10} + d_{11} + d_{19} + d_{20} =$
$0 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 0 + 1 = 0$

$p_5 + d_{12} + d_{13} + d_{14} + d_{15} + d_{16} + d_{17} + d_{18} + d_{19} + d_{20} =$
$1 + 1 + 1 + 0 + 0 + 0 + 0 + 0 + 1 = 0$

Cum toate valorile sunt nule, deci am avut o eroare.
Detecție de eroare: 00101. Deci, eroarea s-a produs pe bitul 5' din codificarea Hamming. Inversăm bitul și obținem mesajul corectat: 00000010101111011100000001

2. $z = E$, deci se obține $t = 00101$
Mesajul transmis: 00011011110110000001
Generator: 101
Calculăm suma de control.
Adăugăm mesajului transmis 2 (gradul polinomului generator) biți de 0 la final:



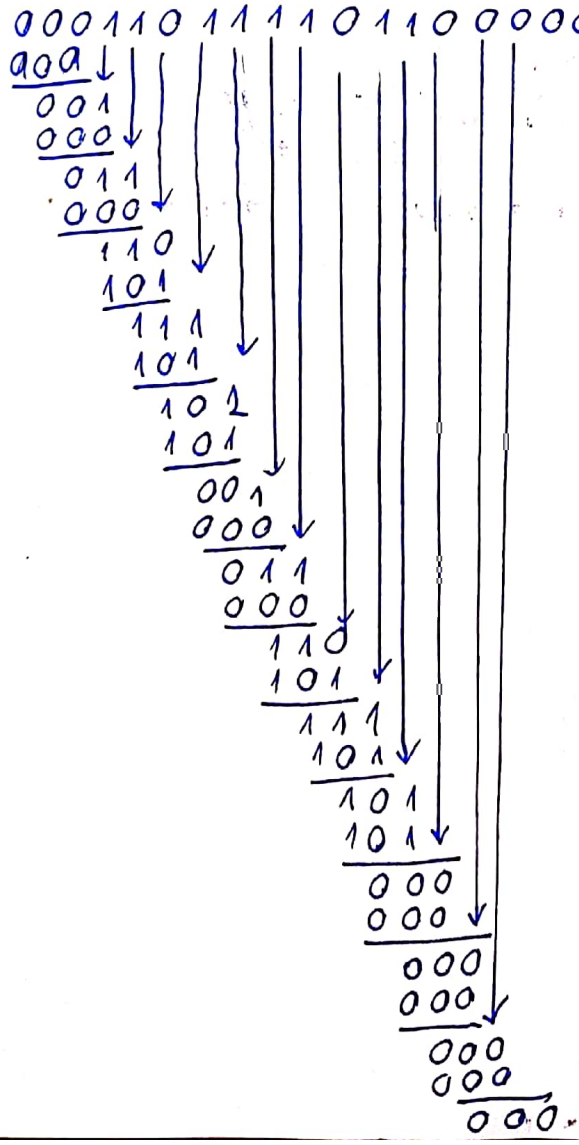


Restul este 01

Deci valoarea transmisă este: 000 1101111 01 1000000101

1. Se transmite fără erori:

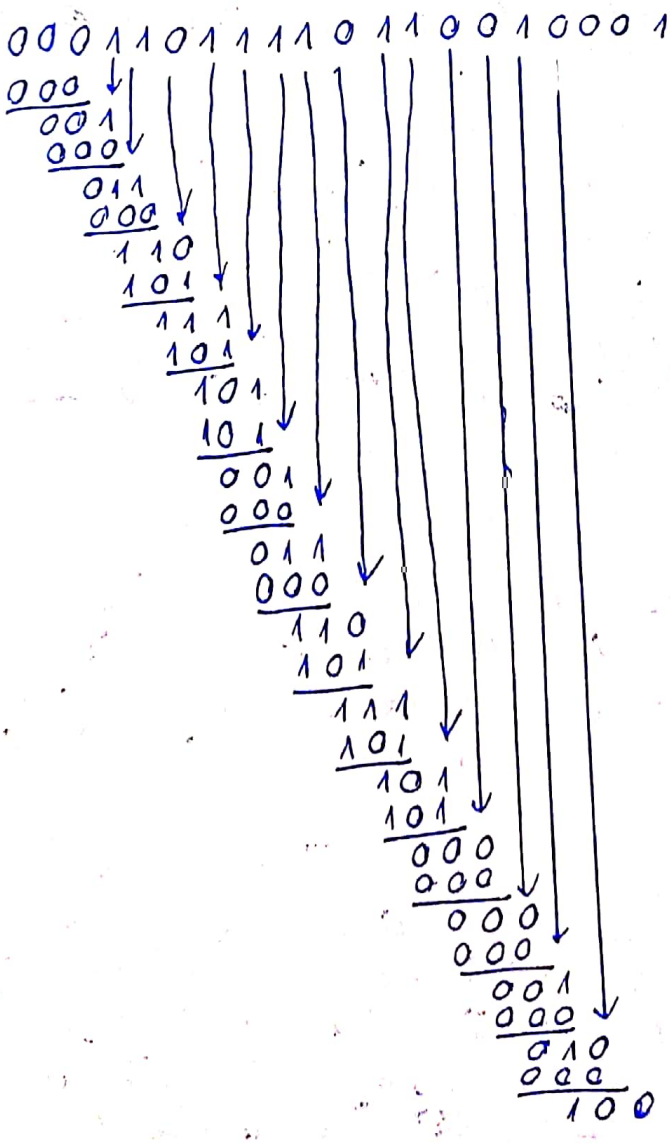
Deci destinația va primi: 000110 1111011000000101 pe care aplicăm algoritmul CRC

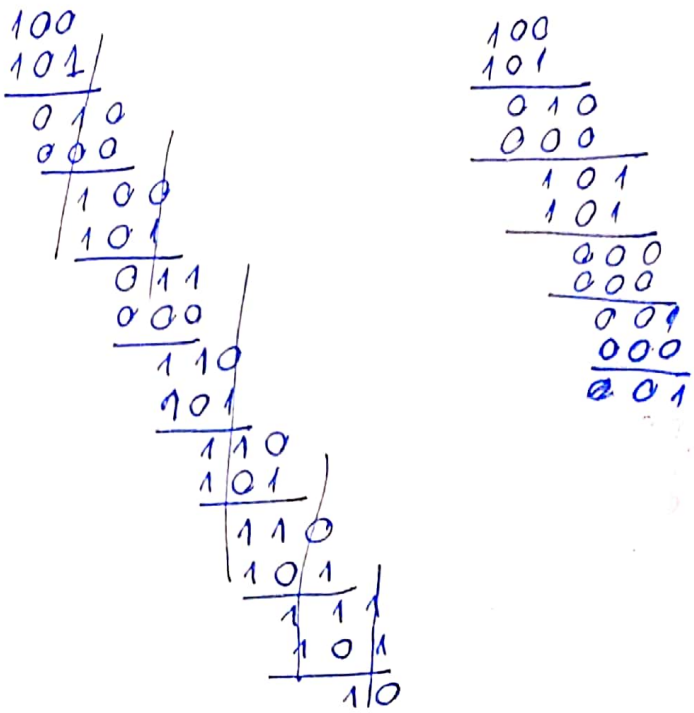


```
000
000
---
 000
 000
 ---
  000
  000
  ---
   001
   000
   ---
    010
    000
    ---
     101
     101
     ---
      000
```

Restul împărțirii este 0. Deci destinația va ști că mesajul a fost transmis fără eroare.

2. Vom transmite mesajul cu erori. Presupunem că s-a transmis: 000110111101100100000101 (am modificat bitul 16) și aplicăm algoritmul CRC





Restul este 01, deci nu e nul, așa că știm că a existat o eroare, însă algoritmul nu o și corectează.

3. Antetul IP este format dintr-o parte fixă de 20 de octeți și o parte de lungime variabilă.

i) Time to live:

Reprezintă un contor de timp folosit pentru a limita durata de viață a pachetelor. Este decrementat la fiecare salt al pachetului (la orice trecere printr-un router. Atunci când ajunge la valoarea 0, pachetul este eliminat. Permite un timp de viață maxim de 255 de secunde.

~~Identif~~

Identificare:

Reprezintă un număr care îi permite destinației să își dea seama cărei datagrame îi aparține un nou pachet primit. Ele ajută la reasamblarea mesajului. Toate fragmentele unei datagrame conțin aceeași valoare de identificare.

Când avem mai multe subrețele interconectate și trecem dintr-o subrețea în altă subrețea, este posibil ca pachetul să fie mai mare decât capacitatea noii rețele de a transmite informații. Astfel pachetul se sparge, iar noile fragmente ~~nu~~ are trecută identificarea în pachet

DF: este un câmp de un bit al cărui nume vine de la Don't Fragment. Când acest bit e activat, avem o cerință ca pachetul să nu se fragmenteze în cazul în care ar trebui să intre într-o rețea a cărui capacitate nu permitea transmiterea pachetului într-o singură bucată. Asta ar însemna că datagrama poate evita drumul optim în favoarea altui drum care îi va permite transmiterea fără fragmentare.

MF: este o prescurtare de la More Fragments, având semnificația că mai urmează fragmente. Este un câmp de un bit; toate fragmentele, cu ~~acest excepție~~ excepția ultimului vor avea acest bit activat.
Acesta ne va indica dacă au sosit toate pachetele din datagrama sau mai există. În cazul în care un pachet nu este fragmentat, el ~~e set~~ are bitul setat ca și cum ar fi ultimul pachet.

Versiune: ne spune cărei versiuni de protocol îi aparține datagrama, adică modul în care au fost implementate. Rețelele diferite pot avea versiuni diferite. Dacă se transmite un mesaj care tranzitează mai multe rețele, e important să se facă adaptarea între versiuni. În acest caz, versiunea mai veche va fi adaptată noii rețele în care intră.

Deplasament: va păstra ordinea fragmentelor în cadrul unei datagrame. Astfel, vom avea ordinea fragmentelor din cadrul unei datagrame, iar când va ieși pe o poartă, mesajul e recompus.

ii)

Timp de viață:
Este utilizat în cadrul inundării. Acesta presupune că fiecărui pachet recepționat ~~de~~ într-un router i se vor face copii și va fi transmis spre toți vecinii săi. Problema este că nu vrem să existe pachete care stau la infinit în rețea. Astfel adăugăm acest câmp în antet care la un nou salt e decrementat. Dacă nu se știe distanța până la destinație, este inițializat cu diametrul rețelei. Când ajunge la valoarea 0, vom ști că pachetul e distrus

și cel puțin o copie a ajuns la destinație. Tehnica e folosită în aplicațiile militare unde avem nevoie de robustețe.

Identificare:

Este folosit în cazul dirijării ierarhice. Analizăm următorul caz: vrem să transmitem un mesaj de la București la New York. Pachetul va trece prin diferite rețele cu capacități mai mari sau mai mici. Este esențial să știm atunci când spargem în fragmente un pachet să știm ~~fiecare~~ fiecare din ce pachet face parte. Poarta prin care vor ieși din rețea va recombina pachetele în funcție de identificare.

Deplasament:

~~Și~~ Similar identificării, ne va ajuta la reconstituirea pachetelor când vor fi sparte. Folosit la dirijare ierarhică. Este sinutil în ~~caz~~ cazul circuitelor virtuale, deoarece fragmentele acelui pachet își păstrează ordinea. Folosit la datagrama unde nu își păstrează ordinea în funcție de pachet.

DF:

Folosit la dirijare ierarhică pentru a ne spune dacă mai sunt ~~pe~~ fragmente din acest pachet dacă a ~~fost~~ spart. De exemplu, dacă vrem să trimitem un pachet important din Australia la București ~~și știm că ar trebui să~~ putem activa opțiunea pentru a ne asigura că nu se fragmentează. Fragmentarea poate duce la apariția erorilor sau pierderea unor fragmente. Este mai important să ajungă în siguranță.

MF: Tot în cazul dirijării ierarhice când ~~vrem~~ se sparge pachetul la trecerea dintr-o rețea în alta și vrem să știm când se termină fragmentele.