TRANSMISIA DATELOR - APLICATII

Detecție şi corecție de erori - codul Hamming

Conf. Dr. Ing. Avram Camelia
Asist. Dr. Ing. Radu Dan

1 Detecție și corecție de erori - codul Hamming

1.1 Obiective

1.2 Introducere

1.2.1 Noțiuni teoretice

1.2.1.1 Distanța Hamming

Distanţa Hamming, (numită după Richard Hamming), se calculează pentru doi vectori de dimensiuni egale. Distanţa Hamming este dată de numărul de poziţii în care vectorii sunt diferiţi. Reprezintă numarul de erori care transformă un vector în celalalt (ex.: 1.1).

Tabela 1.1: Distanța Hamming - exemple.

Vector 1	Vector 2	Distanţa Hamming
1101011	1001001	= 2 (diferă poziţiile 2 şi 6)
1111011	1100111	= 3 (diferă poziţiile 3, 4 şi 5)

Informaţia este reprezentată în binar (ex.: cod ASCII). Operaţii obisnuite în binar: 1.2, 1.3, 1.4.

Tabela 1.2: Adunare în binar - "SAU" / "OR".

+	0	1
0	0	1
1	1	0

Tabela 1.3: Înmulţire în binar - "ŞI" / "AND".

*	0	1
0	0	0
1	0	1

Tabela 1.4: Sau exclusiv în binar - "SAU Ex." / "XOR".

$\oplus$	0	1
0	0	1
1	1	0

1.2.1.2 CODURI LINIARE - HAMMING

În cazul codurilor liniare, cuvântul de cod se scrie uzual sub forma unui vector linie:

$$V = [a_1 a_2 a_3 ... a_n]; \tag{1.1}$$

Din cele n simboluri, k sunt de informaţie, iar m sunt de control. Prin operaţia de codare se înţelege calcularea simbolurilor de control în funcţie de cele de informaţie, astfel încât [V] să fie cuvânt de cod. În cazul codurilor liniare, simbolurile de control sunt combinaţii liniare ale simbolurilor de informaţie. Ele se obţin prin rezolvarea unui sistem liniar de m ecuaţii cu m necunoscute:

$$\begin{cases} h_{11} * a_1 \oplus h_{12} * a_2 \oplus ... \oplus h_{1n} * a_n = 0 \\ h_{21} * a_1 \oplus h_{22} * a_2 \oplus ... \oplus h_{2n} * a_n = 0 \\ \\ ... \\ \\ h_{m1} * a_1 \oplus h_{m2} * a_2 \oplus ... \oplus h_{mn} * a_n = 0 \end{cases} \tag{1.2}$$

Sistem de ecuaţii (1.2) se poate scrie sub forma:

$$[h][V]^T = [0]_{mx1} \tag{1.3}$$

În relaţiile (1.2), (1.3)), produsul este cel obişnuit, iar suma este modulo doi. Matricea:

$$H = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \\ \dots & & & \\ \\ h_{m1} & h_{m2} & \dots & h_{mn} \end{bmatrix}; \quad h_{ij} \in [0,1]; \tag{1.4}$$

reprezintă matricea de control a codului. Specificarea unui cod liniar implică precizarea matricii de control [H]. Proprietăţile de detecţie sau corecţie ale codului se stabilesc prin alegerea matricei [H] în mod corespunzător. Relaţia (1.3) reprezintă condiţia de codare: se

consideră cuvinte de cod doar acele secvenţe binare de n simboluri, care verifică relaţia (1.3). Decodorul recepţionează cuvântul eronat $[V^{'}] = [V] = [\epsilon]$, unde:

$$[\epsilon]^{'}[\epsilon_1\epsilon_2...\epsilon_n]; \tag{1.5}$$

reprezintă cuvântul eroare, definit prin:

$$[\epsilon_i] = \begin{cases} 0, \text{ dacă pe poziția "i" nu s-a introdus o eroare.} \\ 1, \text{ dacă pe poziția "i" s-a introdus o eroare.} \end{cases} \tag{1.6}$$

Operaţia de decodare constă în calcularea corectorului:

$$[Z] = [H] * [V^{'}]^T; \tag{1.7}$$

ţinând cont de legătura dintre $[V^{'}], [V]$ şi $[\epsilon]$ şi de relaţia de codare (1.3) se poate scrie:

$$[Z] = [H] * ([V]^{'} * [\epsilon])^T = [H] * [V]^T = [H] * [\epsilon]^T; \tag{1.8}$$

Dacă nu s-au introdus erori, $[\epsilon] = [0]$, deci [Z]=[0]. Dacă s-a introdus cel puţin o eroare, $[\epsilon] \neq [0]$ şi deci $[Z] \neq [0]$. În cazul codurilor corectoare de erori, din structura corectorului se pot stabili numărul şi poziţia erorilor. În cazul codurilor binare corecţia se efectuează prin negarea simbolului eronat (operaţie care se execută prin sumarea modulo doi a lui 1 la simbolul eronat).

1.2.2 APLICAȚII

1.2.2.1 CODUL HAMMING - CORECTOR DE O EROARE

Codul Hamming corector de o eroare este un cod liniar care are matricea [H] de forma:

$$H = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ & & & & \\ \dots & & & & \\ & & & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix} = [h_1 h_2 \dots h_n]; \tag{1.9}$$

unde fiecare coloană h_i reprezintă transcrierea în cod binar a numărului de ordine al coloanei utilizând n biţi.

Cuvântul de cod [V] are simbolurile de control plasate pe poziţiile 2^i, i = 0 ... N, deci pe poziţiile 1, 2, 4, 8, 16, ...:

$$[V] = [c_1 c_2 a_3 c_4 a_5 a_6 a_7]; \tag{1.10}$$

unde prin c_1, c_2, c_4, ..., c_m s-au notat simbolurile de control, iar prin a_3, a_5, a_6, ..., a_n simbolurile informaţionale.

Datorită acestei alegeri, fiecare relaţie din sistemul (1.2) va conţine un singur simbol de control, permiţând astfel rezolvarea usoară a sistemului de m ecuaţii cu m necunoscute, după cum urmează:

$$\begin{cases} c_1 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ c_2 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ c_4 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ ... \end{cases} \tag{1.11}$$

Recepţionându-se $[V^{'}]$, decodorul calculează $[Z] = [H][V^{'}]^T = [H][\epsilon]^T$. În absenţa erorii, $[\epsilon] = [0]$, deci [Z]=[0]. Dacă există o eroare pe poziţia i, atunci:

$$[Z]^{'} = [H] * [\epsilon]^T \tag{1.12}$$

În prezenţa erorii, corectorul [Z] reprezintă coloana din matricea [H] corespunzătoare poziţiei erorii, adică tocmai poziţia erorii scrisă în binar. Pe baza corectorului [Z] calculat din cuvântul recepţionat, se poate determina poziţia erorii şi apoi corecta bitul eronat.

1.2.2.2 CODUL HAMMING - CORECTOR DE O EROARE, DETECTOR DE DOUĂ ERORI

În scopul corecţiei unei erori şi detectării erorilor duble se mai adaugă un bit de control (paritate), notat c_0 şi calculat cu XOR (modulo doi) a tuturor celorlalte simboluri:

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \dots a_n; \tag{1.13}$$

Notând cu $[V_{H1}] = [c_1 c_2 a_3 c_4 a_5 \dots]$ cuvântul de cod din cazul codului Hamming corector de o eroare şi cu $[V_{H2}] = [c_0 c_1 c_2 a_3 c_4 a_5 \dots]$ cuvântul de cod din cazul codului Hamming corector de o eroare, detector de erori duble, se poate observa usor legătura dintre acestea:

$$[V_{H2}] = [c_0 V_{H1}]; \tag{1.14}$$

Matricea de control pentru noul cod provine din matricea $[H_1]$ de la codul corector de o eroare prin adăugarea unei prime coloane de m zerouri (zero scris în binar cu m biţi) şi apoi a unei ultime linii de n+1 unităţi:

$$H_2 = \begin{bmatrix} 0 & \dots & & & \\ 0 & \dots & & & \\ & & & & \\ \dots & & H_1 & & \\ & & & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & & 1 \\ 0 & 1 & \dots & & 0 \\ & & & & \\ \dots & & & & \\ & & & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}; \tag{1.15}$$

La recepţie se calculează:

$$Z_{H2} = [H_2] * [V_{H2}]^T = \begin{bmatrix} Z_{H1} \\ Z_0 \end{bmatrix}; \tag{1.16}$$

Dacă cuvântul recepţionat este $[v^{'}] = [c_0^{'} c_1^{'} c_2^{'} a_3^{'} c_4^{'} a_5^{'} ... a_n^{'}]$, din (1.16) rezultă:

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \ldots a_n; \tag{1.17}$$

Comparând (1.13) cu (1.17) rezultă:

- dacă pe canal apare un număr par de erori rezultă $z_0 = 0$;
- dacă pe canal apare un număr impar de erori rezultă $z_0 = 1$;
- dacă pe canalul de transmisiuni pot să apară cel mult două erori, atunci când $z_0 = 0$ si $[Z_{H1}] = [0]$, se decide că în cuvântul recepţionat sunt două erori, iar când $z_0 = 1$ se decide că în cuvântul recepţionat este o eroare pe poziţia rezultată transcriind în zecimal componentele corectorului $[Z_{H1}]$;
- dacă $z_0 = 0$ si $[Z_{H1}] = [0]$ se decide că ceea ce s-a recepţionat este corect.

1.2.3 Exemplificări

1.3 Desfășurarea lucrării

1.3.1 Client

Fișierul **index.html** conține:

```
1  <html>
2
3  <head>
4      <meta content="text/html;charset=utf-8" http-equiv="
           Content-Type">
5      <title>Hamming Encoder</title>
6      <link rel="stylesheet" type="text/css" href="css/style.
           css">
7  </head>
8
9  <body>
10
11     <div id="hamming-encoder">
12
```

```
13          <h1>Hamming code for
14              <select v-on:change="initDataBits()" v-model="
                    numberOfDataBits">
15                  <option value="4">4</option>
16                  <option value="8">8</option>
17              </select>
18          bits</h1>
19
20
21          <br>
22          <ul v-for="(bit, index) in dataBits">
23              <li>
24                  <input maxlength="1" :placeholder="'D'+index"
                        v-model="bit.data"
25                      v-bind:class="[validateBit(bit.data) ? '
                            valid-input' : 'invalid-input']" />
26
27              </li>
28          </ul>
29          <br>
30          Data bits: <code>{{dataBits}}</code>
31          <br>
32
33          <button v-on:click="send()">Send</button>
34
35          <p>{{ status }}</p>
36
37      </div>
38
39      <script type="text/javascript" src="libs/vue.js"></script
            >
40      <script type="text/javascript" src="libs/axios.js"></
            script>
41      <script type="text/javascript" src="js/app.js"></script>
42  </body>
43
44  </html>
```

Fișierul **app.js** conține:

```
1
2   var app = new Vue({
3       el: '#hamming-encoder',
4       data: {
```

```
5           dataBits: [],
6           status: '',
7           numberOfDataBits: 4
8       },
9       created: function () {
10          this.initDataBits(4);
11      },
12      methods: {
13          initDataBits: function(){
14              this.dataBits=[];
15
16              for(var i=0;i<this.numberOfDataBits;i++){
17                  var bit = { data: null };
18                  this.dataBits.push(bit);
19              }
20          },
21          send: function () {
22              if (this.validate(this.dataBits) === true){
23                  var encodedMessage = this.encode(this.
                        dataBits);
24
25
26                  return axios.put("http://localhost:3000/
                        message", {bits: encodedMessage}).then(
27                      response => (this.status = response.data)
28                  );
29              } else {
30                  this.status = 'Input is not valid. Please use
                        0 or 1 as data bit values';
31              }
32          },
33          encode: function(bits){
34
35
36              var c4=this.parity(parseInt(bits[1].data)+
                    parseInt(bits[2].data)+parseInt(bits[3].data))
                    ;
37              var c2=this.parity(parseInt(bits[0].data)+
                    parseInt(bits[2].data)+parseInt(bits[3].data))
                    ;
38              var c1=this.parity(parseInt(bits[0].data)+
```

```
                parseInt(bits[1].data)+parseInt(bits[3].data))
                ;
39              console.log("Control bits: "+c1+","+c2+","+c4);
40              return [c1,c2,parseInt(bits[0].data),c4,parseInt(
                    bits[1].data),parseInt(bits[2].data),parseInt(
                    bits[3].data)];
41          },
42          parity: function(number){
43              return number % 2;
44          },
45          validate: function(bits){
46              for(var i=0; i<bits.length;i++){
47                  if (this.validateBit(bits[i].data) === false)
48                  return false;
49              }
50              return true;
51          },
52          validateBit: function(character){
53              if (character === null) return false;
54              return (parseInt(character) === 0 ||
55              parseInt(character) === 1);
56          }
57      }
58  })
```

Fișierul **style.css** conține:

```
1   body {
2     margin: 30px;
3   }
4
5   select {
6     margin-top: -5px;
7     padding: 5px;
8   }
9
10  ul {
11    list-style: none;
12    display: inline-block;
13    margin: 0px;
14    margin-bottom: 20px;
15    margin-top: 0px;
16    padding: 0px;
17    padding-right: 10px;
```

```
18 }
19
20 input {
21   font-size: 16px;
22   padding: 10px;
23   width: 50px;
24   text-align: center;
25 }
26
27 .invalid-input {
28   border: 3px solid orange;
29   -webkit-transition: all 1000ms linear;
30   -ms-transition: all 1000ms linear;
31   transition: all 1000ms linear;
32 }
33
34 .valid-input {
35   border: 3px solid green;
36   background-color: green;
37   color: white;
38   -webkit-transition: all 1000ms linear;
39   -ms-transition: all 1000ms linear;
40   transition: all 1000ms linear;
41 }
42
43 button {
44   margin-top: 10px;
45   padding: 10px;
46 }
47
48 code {
49   background-color: #efffec;
50   margin: 0px;
51 }
```

1.3.2 SERVER

Fișierul **run.js** conține:

```
1 var api = require('./api.js').app;
2 var hamming = require('./hamming.js');
3
4 api.put('/message', function(request, response) {
```

```
5   var bits = request.body.bits;
6
7
8   var decoded = hamming.decode(bits);
9   if(decoded.errorCorrected){
10    response.json('One error corrected on position: '+decoded
        .errorPosition);
11  }
12  response.json('Message received without errors');
13 });
14
15 api.listen(3000, function(){
16   console.log('CORS-enabled web server is listening on port
       3000...');
17 });
18
19 function distortBit(bits, index){
20   bits[index] = (bits[index]+1) % 2;
21   return bits;
22 }
```

Fișierul **api.js** conține:

```
1  var express = require('express');
2  var cors = require('cors');
3  var app = express();
4  app.use(cors());
5
6
7  var bodyParser = require('body-parser');
8  app.use(bodyParser.urlencoded({
9    extended: true
10 }));
11 app.use(bodyParser.json());
12
13 exports.app = app;
```

Fișierul **hamming.js** conține:

```
1  function decode(bits) {
2    var z4=parity(bits[3]+bits[4]+bits[5]+bits[6]);
3    var z2=parity(bits[1]+bits[2]+bits[5]+bits[6]);
4    var z1=parity(bits[0]+bits[2]+bits[4]+bits[6]);
5
6      var errorPosition=z1*1+z2*2+z4*4;
7    var errorDetected=false;
```

```
8   if (errorPosition!=0) errorDetected=true;
9   if (errorDetected) {
10    bits[errorPosition-1]=parity(bits[errorPosition-1]+1);
11  }
12    return { errorCorrected: errorDetected, errorPosition:
          errorPosition-1, bits: bits };
13 }
14
15 parity = function(number){
16   return number % 2;
17 }
18
19 exports.decode = decode;
```

Fișierul **package.json** conține:

```
1  {
2    "name": "it-prototype",
3    "description": "Node-Express Server",
4    "private": true,
5    "version": "0.0.1",
6    "dependencies": {
7      "body-parser": "^1.12.2",
8      "cors": "^2.8.5",
9      "express": "3.x"
10   },
11   "main": "Prototype",
12   "devDependencies": {}
13 }
```

1.4 Probleme propuse

1.4.1 Problema *(5 puncte)*

- Soluție pentru 4 respectiv 8 biți de date *(2 p)*.
- Soluție pentru n biți de date *(3 p)*.

1.5 Bibliografie