

DEPARTAMENTUL DE AUTOMATICĂ
UNIVERSITATEA TEHNICĂ DIN CLUJ - NAPOCA

Transmisia Datelor

Îndrumător de laborator - volumul I

Dr. Ing. Camelia Claudia AVRAM
Dr. Ing. Dan RADU

CUPRINS

Listă de tabele	vi
Listă de figuri	vii
1 Descrierea mediului de lucru	1
1.1 Obiective	1
1.2 Introducere	1
1.2.1 Motivație	1
1.2.2 HTML	4
1.2.3 CSS	4
1.2.4 JavaScript	4
1.2.5 Unelte de dezvoltare	5
1.3 Desfășurarea lucrării	5
1.3.1 Prima aplicație JavaScript	5
1.3.1.1 Instrucțiuni uzuale JavaScript	6
1.3.2 Implementarea și testarea unei funcții în JavaScript	8
1.3.3 Proiectarea unei interfețe grafice în HTML	9
1.3.4 jQuery	11
1.4 Aplicații și probleme	13
1.4.1 Problema 1 (3 puncte)	13
1.4.2 Problema 2 (1.5 puncte)	13
1.4.3 Problema 3 (1.5 puncte)	13
1.4.4 Problema 4 (2 puncte)	13
1.4.5 Problema 5 (2 puncte)	14
1.5 Ce vom învăța în laboratorul următor?	14
Bibliografie	15
2 Sisteme de comunicații cu fir	16
2.1 Obiective	16
2.2 Introducere	16
2.2.1 Motivație	16
2.2.2 Medii de transmisie cu fir	16
2.2.2.1 Cablul coaxial	16
2.2.2.2 Cablul torsadat	19
2.2.2.3 Fibra optică	23
2.2.3 Topologii de rețea	25

2.2.4	Aplicații web	27
2.2.5	Arhitectura client-server	27
2.2.6	TCP/IP	27
2.2.7	HTTP vs. WebSocket	29
2.3	Desfășurarea lucrării	30
2.3.1	Aplicație client cu Vue.js	30
2.3.2	Aplicație client-server cu HTTP	32
2.3.2.1	Modulul server	32
2.3.2.2	Modulul client	34
2.3.3	Aplicație client-server cu WebSockets	36
2.3.3.1	Server	36
2.3.3.2	Client	37
2.4	Aplicații și probleme	39
2.4.1	Problema 1 (1 punct)	39
2.4.2	Problema 2 (4 puncte)	39
2.4.3	Problema 3 (3 puncte)	39
Bibliografie		40
3	Detectie și corecție de erori - codul Hamming	41
3.1	Obiective	41
3.2	Introducere	41
3.2.1	Noțiuni teoretice	41
3.2.1.1	Distanța Hamming	41
3.2.1.2	Coduri liniare - Hamming	42
3.2.2	Aplicații	43
3.2.2.1	Codul Hamming - corector de o eroare	43
3.2.2.2	Codul Hamming - corector de o eroare, detector de două erori	44
3.2.3	Exemplificări	45
3.3	Desfășurarea lucrării	45
3.3.1	Client	45
3.3.2	Server	49
3.4	Probleme propuse	51
3.4.1	Problema (5 puncte)	51
Bibliografie		52
4	Compresia datelor - Algoritmi statici de compresie. Algoritmul Huffman. Algoritmul Shannon-Fano	53
4.1	Obiective	53

4.2	Introducere	53
4.2.1	Noțiuni teoretice	54
4.2.1.1	Criterii de apreciere a unui cod	56
4.2.1.2	Codul Huffman	57
4.2.1.3	Codul Shannon - Fano	58
4.2.2	Aplicații	60
4.2.2.1	Algoritmul Huffman	60
4.2.2.2	Algoritmul Shannon - Fano	63
4.3	Desfășurarea lucrării	66
4.3.1	Aplicație cu D3js	67
4.3.1.1	Algoritmul Huffman	74
4.3.1.2	Algoritmul Shannon-Fano	75
4.4	Probleme propuse	77
Bibliografie		79
5	Coduri în banda de bază	80
5.1	Obiective	80
5.2	Introducere	80
5.2.1	Noțiuni teoretice	80
5.2.1.1	Coduri NRZ	82
5.2.1.2	Coduri RZ	83
5.2.1.3	Codarea bifazică	83
5.2.1.4	Coduri binare multinivel MLB ('Multi Level Binary')	84
5.2.2	Aplicații	87
5.3	Desfășurarea lucrării	91
Bibliografie		93
6	Comunicații prin satelit. Aplicații ale comunicațiilor prin satelit - GPS	94
6.1	Obiective	94
6.2	Introducere	94
6.2.1	GPS - Sistemul de poziționare globală	96
6.2.2	Determinarea poziției prin triangulație	98
6.2.3	Formatul mesajelor GPS	100
6.3	Desfășurarea lucrării	101
6.3.1	Afișarea coordonatelor GPS în browser	101
6.3.2	Afișarea locației GPS	102
6.3.2.1	în Google Maps	102
6.3.2.2	TomTom Maps	105
6.4	Aplicații și probleme	106

LISTĂ DE TABELE

3.1	Distanța Hamming - exemple.	41
3.2	Operații în binar: Adunare - "SAU" / "OR"; Înmulțire - "ȘI" / "AND"; Sau exclusiv - "SAU Ex." / "XOR".	41
6.1	Informații conținute în sub-cadre	101

LISTĂ DE FIGURI

1.1	Limbajele de programare cele mai populare pe Github în 2018 (Sursa: [4])	2
1.2	Limbajele de programare cele mai populare pe StackOverflow în 2018 (Sursa: [5])	3
2.1	Cablul coaxial	17
2.2	Cablul coaxial	18
2.3	Conectori, (Sursa: [5], [6])	18
2.4	Conectori, (Sursa: [5], [6])	19
2.5	Cablul torsadat, 4 perechi	20
2.6	Cablul UTP, 4 perechi	20
2.7	Cablul S/STP, 4 perechi	20
2.8	Conectori RJ-45, standardele 568A și 568B de mufare, (Sursa: [9], [10])	22
2.9	Fibră optică, (Sursa: [11])	23
2.10	Conectori fibră optică, (Sursa: [13])	24
2.11	Conectori fibră optică - componente, (Sursa: [12])	24
2.12	Topologii de rețea, (Sursa: [14])	26
2.13	Conectare aplicație browser (client) - server web	28
2.14	TCP, UDP, IP	29
4.1	Arbore Shannon-Fano	59
4.2	Arbore Huffman - pasul 1	60
4.3	Arbore Huffman - pasul 2	60
4.4	Arbore Huffman - pasul 3	61
4.5	Arbore Huffman - pasul 4	61
4.6	Arbore Huffman - pasul 5	61
4.7	Arbore Huffman - pasul 6	62
4.8	Arbore Huffman	62
4.9	Arbore Shannon-Fano	65
5.1	Apariția și propagarea erorii de detecție a începutului de caracter, (Sursa: [2])	82
5.2	Codare 4B/5B	86
5.3	Exemplu de codare 4B/5B	87
6.1	Mecanismul comunicației prin satelit, (Sursa: [12])	95
6.2	Comunicației prin satelit - conexiune one-way, (Sursa: [12])	97
6.3	Comunicației prin satelit - conexiune two-way, (Sursa: [12])	98
6.4	Triangulația	99

6.5 Structura unui sub-cadru	100
--	-----

1 DESCRIEREA MEDIULUI DE LUCRU

1.1 OBIECTIVE



- Introducere în tehnologiile HTML, CSS și JavaScript
- Proiectarea și implementarea interfețelor grafice
- Gestionarea evenimentelor în aplicații software

1.2 INTRODUCERE

1.2.1 MOTIVAȚIE

În prezent pachetul de tehnologii ("technology stack") **HTML, CSS și JavaScript** este extrem de popular în industria IT pentru că se adresează unui spectru larg de aplicații software. Cu ajutorul acestor tehnologii se dezvoltă în special aplicații web. Totuși, în ultima perioadă, datorită apariției unor platforme cum sunt Node.js [2] și Electron [3]), există un sprijin tot mai puternic pentru aplicații backend, desktop și mobile.

Un citat celebru din anul 2007 spune că *"orice aplicație care poate fi scrisă în JavaScript va fi în cele din urmă scrisă în JavaScript"* - Jeff Atwood, Cofondator al StackOverflow. Acest fapt este confirmat de ultimele sondaje care se referă la cele mai frecvent utilizate limbaje de programare. JavaScript este limbajul de programare cel mai popular atât pe pe Github cât și pe StackOverflow, după cum reiese din statisticile pe anul 2018 prezentate în Figurile 1.1 și 1.2.

De asemenea, multe companii importante din industria software (Facebook/React, Google/Angular, Microsoft, Mozilla, Yahoo, Alibaba) susțin și contribuie la dezvoltarea uneltelor și librăriilor necesare pentru dezvoltarea aplicațiilor pe aceste trei limbaje.

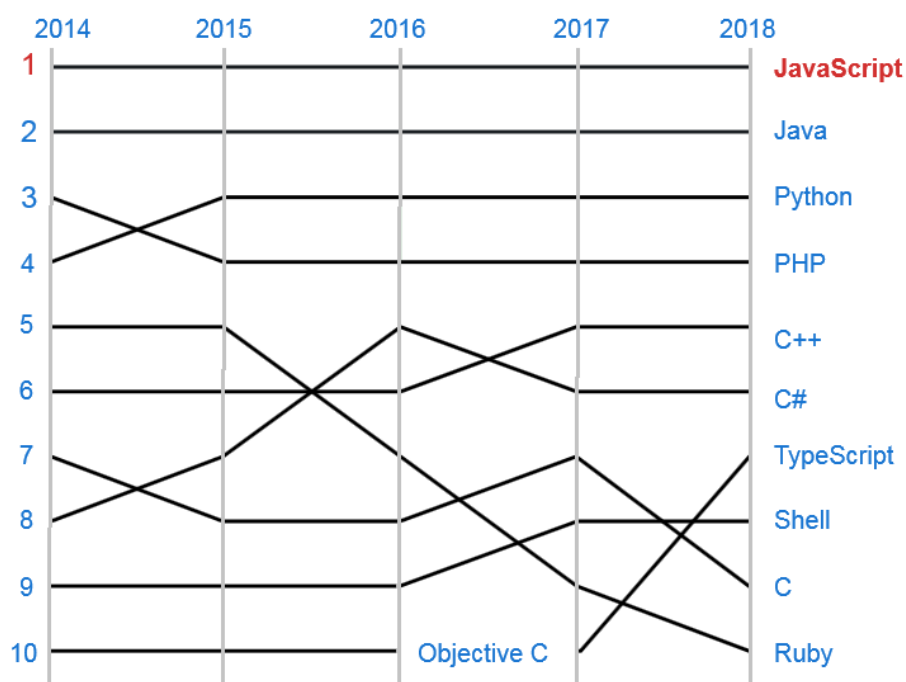


Figure 1.1: Limbajele de programare cele mai populare pe Github în 2018 (Sursa: [4])

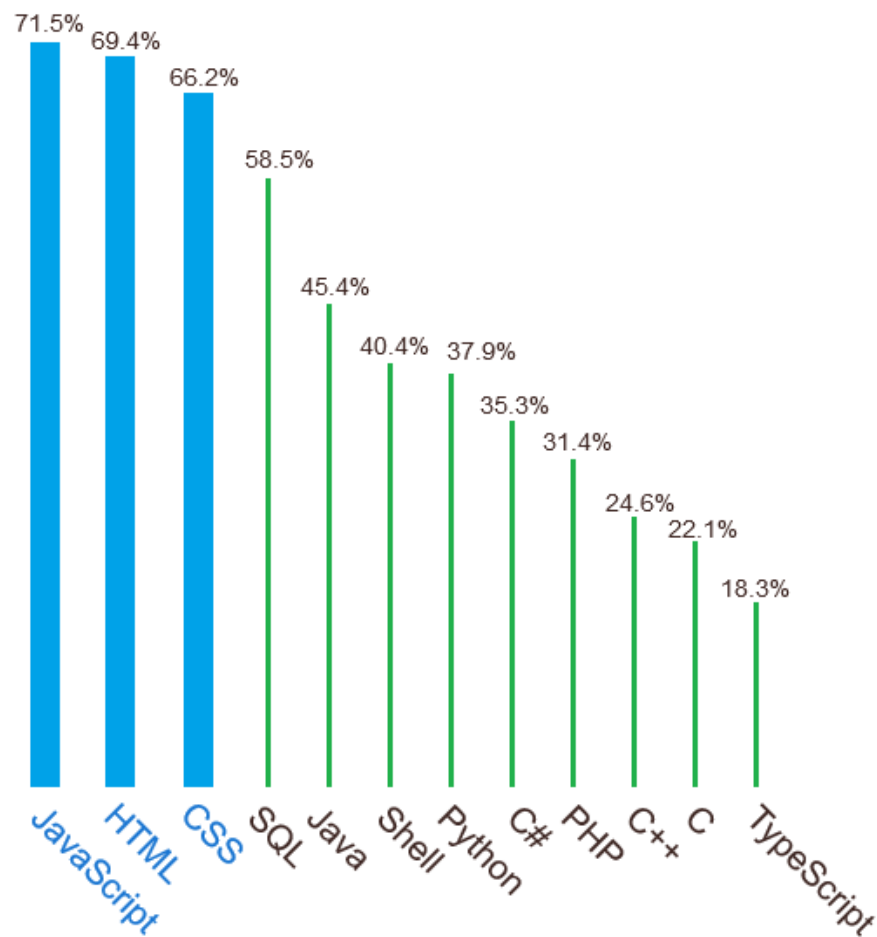


Figure 1.2: Limbajele de programare cele mai populare pe StackOverflow în 2018 (Sursa: [5])

1.2.2 HTML

Acronimul HTML vine de la "Hyper Text Markup Language" și reprezintă un limbaj care ne permite să definim interfețe grafice pentru aplicații software care sunt executate în browser.

O pagină HTML are o structură sub formă de arbore formată din elemente grafice, care are nod rădăcină elementul `<html>`. Acesta are doi copii `<head>` și `<body>`. În `<head>` se descrie pagina web (meta data, titlu, tipul de caractere folosite). De asemenea, se includ eventualele fișiere externe cu stiluri CSS și scripturile JavaScript. În `<body>` se descrie interfața grafică care într-o aplicație web include: antet, meniu și conținut.

Printre elementele HTML cele mai importante se numără: `<div>`, `<input>` și `<button>`.

1.2.3 CSS

CSS (Cascading Style Sheets) este limbajul prin care se descrie modul în care elementele HTML sunt reprezentate grafic. Elementele grafice au un stil standard de afișare care este specific browserului în care pagina este afișată. Totuși, acest stil poate fi suprascris și customizat folosind sintaxa CSS.

Câteva proprietăți CSS frecvent utilizate sunt: `font-size`, `border`, `background-color`, `color`, `width` și `height`.

Cu ajutorul stilurilor definite în CSS se poate controla modul de reprezentare grafică al aplicațiilor pe diferite medii de redare (monitoare, laptopuri, smartphone-uri și hârtie).

1.2.4 JAVASCRIPT

JavaScript este un limbaj de programare creat de către Brendan Eich în 1995 și a fost folosit pentru prima dată în browserul Netscape.

JavaScript este limbajul care controlează comportamentul interfețelor grafice web și nu numai. Chiar dacă anterior JavaScript-ul era folosit în special pentru aplicații web, în ultimii ani a devenit o alegere tot mai populară pentru aplicații desktop și mobile.

Există mai multe librării care permit crearea unor aplicații client complexe: jQuery, React, VueJS, Angular. Acestea extind capabilitățile JavaScript prin diverse funcționalități uzuale cum ar fi: sintaxă simplificată pentru referențierea elementelor HTML, legătură bidirecțională între model (variabile) și interfața grafică, arhitectură bazată pe componente.

Pentru orice tip de aplicații (frontend, backend, mobile) în general se folosește platforma NodeJS care facilitează structurarea și automatizarea multor pași necesari în dezvoltarea software (importul de librării, execuția unor servere web care servesc resurse, configurarea și execuția buildului pe client).

1.2.5 UNELTE DE DEZVOLTARE

Deși browserul și Notepad-ul sunt suficiente pentru a scrie și testa aplicații simple, o dată ce programele devin mai complexe se recomandă utilizarea următoarelor editoare de cod: Visual Studio Code, Sublime sau Atom.

1.3 DESFĂȘURAREA LUCRĂRII

1.3.1 PRIMA APLICAȚIE JAVASCRIPT


Prima aplicație conține 3 fișiere: **index.html**, **style.css** și **app.js**. Fișierul **index.html** definește structura interfeței grafice și care pentru început va afișa un simplu câmp text. În secțiunea <head> se include fișierul CSS iar la sfârșitul secțiunii <body> fișierul JavaScript.

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4      </head>
5      <body>
6          <div id="message"></div>
7
8          <script src="app.js"></script>
9      </body>
10 </html>
```

Este important ca instrucțiunile JavaScript să fie executate doar în momentul în care toate elementele grafice HTML din <body> au fost deja încărcate în browser. Acest lucru se poate realiza prin amplasarea scriptului la sfârșitul elementului <body>, având în vedere că resursele (JS, CSS, HTML) se încarcă în browser secvențial în ordina apariției lor în fișierul HTML. O altă soluție este executarea codului doar la apariția evenimentului generat când pagina s-a încărcat (ex: <body onload="run();">).

În fișierul JavaScript **app.js** se afișează la consola din browser un mesaj:

```
1 console.log("Welcome to data transmission");
```

 Pentru a executa aplicația și a vedea mesajul din JavaScript se deschide **index.html** în orice browser, se apasă tasta F12 și se selectează tabul **Console**.

În continuare se va modifica din JavaScript valoarea șirului de caractere afișat în elementul HTML astfel:

```
1 document.getElementById("message").innerHTML = "Message
    from JavaScript";
```

Cu ajutorul obiectului nativ JavaScript denumit **document** [6] se poate obține o referință spre orice element HTML pe baza unui selector, fie acesta tipul elementului, id-ul sau clasa acestuia. În momentul în care obținem această referință putem controla caracteristicile elementului HTML din JavaScript. De asemenea, putem gestiona anumite evenimente care sunt generate atunci când un utilizator folosește interfața grafică.

👉 Identificatorul folosit în JavaScript ca argument al metodei `getElementById` ("message") este identic cu identificatorul unic specificat în elementul HTML "`<div id="message"></div>`".

În momentul în care obținem referința spre elementul HTML din JavaScript, avem posibilitatea să modificăm caracteristicile acestuia. De asemenea, putem gestiona anumite evenimente care sunt generate atunci când utilizatorul interacționează cu obiectul grafic.

1.3.1.1 INSTRUCȚIUNI UZUALE JAVASCRIPT

În JavaScript variabilele se declară utilizând cuvântul cheie **var** pus în fața numelui variabilelor și pot lua diferite valori. În scriptul următor se vor declara patru tipuri de variabile (Number, String, Boolean și Object).

```
1 var sum = 10;
2 var name = "Alexandru";
3 var isActive = true;
4 var user = {id: 1, name: "Andrei", age: 21, };
```

Obiectul `user` este în format JSON (JavaScript Object Notation). În sintaxa JSON obiectele încep cu `{` și se termină cu `}`. Între acolade se specifică perechi de tipul **cheie: valoare** care definesc caracteristicile specifice ale obiectului. Obiectele JSON pot include și liste de elemente care se declară între paranteze drepte. Un exemplu de obiect JSON complex este prezentat în scriptul următor:

```
1 var user = {
2     "id": 1,
3     "name": "Alexandru Cristea",
4     "username": "acristea",
5     "email": "acristea@test.com",
6     "address": {
7         "street": "Padin",
8         "number": "Ap. 10",
9         "city": "Cluj-Napoca",
10        "zipcode": "123456",
11        "geo": {
```

```
12     "lat": "46.783364",
13     "lng": "23.546472"
14   },
15 },
16 "phone": "004-07xx-123456",
17 "company": {
18   "name": "XYZ",
19   "domain": "Air Traffic Management",
20   "cities": ["Cluj-Napoca", "Vienna", "Paris"]
21 }
22 }
```

Afișarea diferitelor caracteristici ale obiectului `user` este exemplificată în următorul script:

```
1
2 console.log(user.name);
3 console.log(user.address.geo.lat);
4 console.log(user.company.name);
5 console.dir(user.company.cities);
6 console.log(user.company.cities[0]);
7 ...
```

În continuare vom defini și apela o funcție:

```
1 function print(message){
2   console.log(message);
3 }
4 print("hello");
```

Operatorul ternar este o simplificare a instrucțiunii **if** și este exemplificat în continuare:

```
1 var password="123456";
2 console.log(password=="123456"? "ALLOW": "DENY");
```

Logica echivalentă descrisă cu instrucțiunea **if** este următoarea:

```
1 var password="123456";
2 if(password == "123456"){
3   console.log("permission accepted");
4 } else {
5   console.log("permission denied");
6 }
```

1.3.2 IMPLEMENTAREA ȘI TESTAREA UNEI FUNCȚII ÎN JAVASCRIPT

Aplicația conține 3 fișiere: **index.html**, **app.js** și **appTest.js**. Interfața grafică ne permite să introducem o valoare într-un câmp input:

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4   </head>
5   <body>
6     <input type="text" id="n" value="5"/>
7
8     <script src="app.js"></script>
9     <script src="appTest.js"></script>
10  </body>
11 </html>
```

Fișierul **app.js** conține:

```
1 document.getElementById("n").addEventListener('input',
    inputSum);
2
3 function inputSum(){
4   var inputNumber = parseInt(document.getElementById("n")
    .value);
5   sum(inputNumber);
6 }
7
8 function sum(n){
9   if (typeof n === 'undefined') return "n is undefined";
10  var sum = 0;
11  for(var i=1;i<=n;i++){
12    sum+=i;
13  }
14  return sum;
15 }
```

- Linia 1: Se specifică un event handler care va fi apelat atunci când utilizatorul modifică valoarea din input.
- Linia 4: În JavaScript variabilele se definesc folosind cuvântul cheie **var**. Se citește valoarea din câmpul input și se convertește din șir de caracter în număr întreg.
- Linia 5: Se apelează funcția care returnează suma elementelor de la 1 până la n.

Fișierul **appTest.js** conține o suită de 4 teste care validează comportamentul dorit al funcției **sum**:

```
1 function test(){
2     console.log(sum(0)==0?"Passed":"Failed");
3     console.log(sum(2)==3?"Passed":"Failed");
4     console.log(sum(4)==10?"Passed":"Failed");
5     console.log(sum()=="n is undefined?"Passed":"Failed");
6 }
7 test();
```

Sintaxa JavaScript ne permite să scriem codul de mai sus și ca funcție anonimă executată imediat astfel:

```
1 (function(){
2     console.log(sum(0)==0?"Passed":"Failed");
3     console.log(sum(2)==3?"Passed":"Failed");
4     console.log(sum(4)==10?"Passed":"Failed");
5     console.log(sum()=="invalid argument?"Passed":"Failed"
6     );
7 })();
```

1.3.3 PROIECTAREA UNEI INTERFEȚE GRAFICE ÎN HTML

Fișierul **index.html** definește structura interfeței grafice care conține un label text, și un buton.

```
1 <!DOCTYPE html>
2 <html>
3     <head>
4         <link rel="stylesheet" href="style.css">
5     </head>
6     <body>
7         <div id="counter"></div>
8         <button id="inc">+</button>
9
10        <script src="app.js"></script>
11    </body>
12 </html>
```

În continuare se va modifica stilul CSS din fișierul **style.css** astfel:

```
1 body {
2     padding: 20px;
3     text-align: center;
4     background-color: #f5f5f5;
```

```
5  }
6
7  #widget {
8      width: 60px;
9      background-color: #555;
10     margin: 0 auto;
11     padding: 10px;
12 }
13
14 #counter {
15     height: 50px;
16     width: 50px;
17     margin: 0 auto;
18     background-color: #fff;
19     color: #555;
20     text-align: center;
21     line-height: 50px;
22     font-size: 30px;
23 }
24
25 #inc {
26     margin-top: 10px;
27     height: 50px;
28     width: 50px;
29     background-color: #5b9aff;
30     color: #fff;
31     text-align: center;
32     line-height: 50px;
33     font-size: 20px;
34     border: 0px;
35 }
36
37 #inc:hover {
38     cursor: pointer;
39     background-color: #8cb8ff;
40     color: #fff;
41 }
```

Fișierul JavaScript **app.js** descrie comportamentul interfeței grafice și anume logica necesară ca valoarea contorului să fie incrementată de fiecare dată când butonul este apăsat de către utilizator.

```
1  var counter = 0;
2
```

```
3 function printValue(divId, value){
4     document.getElementById(divId).innerHTML = value;
5 }
6 printValue("counter", 0);
7
8 document.getElementById("inc").addEventListener("click",
9     increment);
10 function increment(){
11     counter++;
12     printValue("counter", counter);
13 }
```

- Linia 1: Starea aplicației.
- Linia 3: Funcția care ne permite să afișăm o valoare text într-un div cu un identificator specificat.
- Linia 6: Afișarea inițială a contorului pe interfața grafică.
- Linia 8: Specificarea unui event handler care va fi apelat atunci când utilizatorul apasă butonul de incrementare. Acest event handler conține 2 parametri: tipul evenimentului și funcția care va fi apelată la producerea evenimentului specificat.

1.3.4 JQUERY

jQuery este o bibliotecă scrisă în JavaScript care simplifică și oferă o sintaxă mai expresivă de programare și extinde totodată funcționalitățile limbajului JavaScript. jQuery se poate include în secțiunea head a paginii HTML astfel:

```
1 <head>
2     <script src="https://ajax.googleapis.com/ajax/libs/
3         jquery/3.3.1/jquery.min.js"></script>
4 </head>
```

Pentru obținerea referinței spre un element HTML se folosește sintaxa simplificată **\$("#id")**. Simbolul "\$" specifică începutul unui obiect/funcții jQuery. Codul JavaScript din secțiunea 1.3.3 poate fi rescris cu jQuery astfel:

```
1 var counter = 0;
2
3 function printValue(divId, value){
4     $("#"+divId).html(value)
5 }
6 printValue("counter", 0);
```

```
7  
8 $("#inc").on('click', increment);  
9  
10 function increment(){  
11     counter++;  
12     printValue("counter", counter);  
13 }
```

Linia 4: Pentru a scrie într-un element de tip **div** se folosește funcția **html()** din biblioteca **jQuery**.

Dacă se dorește utilizarea unui câmp de tip **input** text în aplicație, citirea (linia 1) și setarea (linia 2) valorii elementului **input** cu jQuery se face astfel:

```
1 var textValue = $('#inputTextId').val();  
2 $('#inputTextId').val('123');
```

1.4 APLICAȚII ȘI PROBLEME

1.4.1 PROBLEMA 1 (3 puncte)

Să se implementeze/testeze toate aplicațiile descrise în Secțiunea [1.3.1](#):

- Modificarea valorii unui câmp text din JavaScript folosind funcția **getElementById** accesibilă din interfața obiectului **document** (0.25 p).
- Afișare mesaj la consolă din JavaScript (0.25 p).
- Accesarea unei valori dintr-un obiect JSON (0.25 p).
- Definirea și apelul unei funcții JavaScript (0.25 p).
- Utilizarea instrucțiunii **if** ca operator ternar (0.25 p).
- Proiectarea unei interfețe grafice în HTML (0.5 p).
- Folosirea bibliotecii jQuery în aplicația JavaScript (0.25 p).

1.4.2 PROBLEMA 2 (1.5 puncte)

Să se extindă aplicația prezentată în Secțiunea [1.3.3](#) prin adăugarea unui buton de decrementare și să se restricționeze valoarea contorului pe intervalul [0-10].

1.4.3 PROBLEMA 3 (1.5 puncte)

Să se modifice logica funcției **sum** din Secțiunea [1.3.2](#) astfel încât să permită exclusiv introducerea unui număr ca argument de intrare. De exemplu, dacă se introduce un șir de caractere se așteaptă returnarea valorii "not a number".

Să se scrie două teste care verifică introducerea unui șir de caractere și o variabilă booleană ca argument al funcției **sum**.

1.4.4 PROBLEMA 4 (2 puncte)

Să se scrie o funcție JavaScript cu antetul **getFibonacci(n)** care generează elemente din șirul lui Fibonacci până la o limită impusă **n** și să se scrie teste care validează comportamentul următor:

- Funcția returnează șirul de numere [1, 1] atunci când argumentul de intrare este 2.

- Funcția returnează șirul de numere [1, 1, 2, 3, 5] atunci când argumentul de intrare este 5.
- Funcția apelată fără argument sau cu orice tip de dată în afară de număr returnează "not allowed".
- Funcția apelată cu $n < 1$ sau $n > 10$ returnează "not allowed".

1.4.5 PROBLEMA 5 (2 puncte)

Pe baza modelului prezentat în Secțiunea 1.3.2 să se implementeze cu ajutorul **jQuery** interfața grafică pentru un calculator simplu care permite operații de adunare, scădere, înmulțire, împărțire și rest [+ , - , / , * ,

Câmpuri necesare:

- Două câmpuri **input** text (cu id-urile **firstNumber** și **secondNumber**) pentru preluarea datelor de la utilizator. În jQuery se folosește funcția **val** a elementului **input** HTML (ex: `var firstNumberText = $('#firstNumber').val();`) pentru a citi valoarea câmpului text. Pentru conversia din șir de caractere în număr întreg se poate folosi funcția **parseInt** (ex: `var firstNumber = parseInt(firstNumberText)`).
- Un element **button** pentru egal (cu event handler la click).
- Afișarea rezultatului într-un **div**.

1.5 CE VOM ÎNVĂȚA ÎN LABORATORUL URMĂTOR?

Despre:

- Medii de transmisie cu fir.
- Transferarea datelor între componente software distribuite utilizând JavaScript atât pe frontend (UI) cât și pe backend (API).
- Utilizarea protocoalelor **http** și **websocket** pentru dezvoltarea de aplicații client-server.

Vom folosi librăriile și tehnologiile **NodeJS**, **VueJS** și **Express.js**.

BIBLIOGRAFIE

- [1] Adina Aștilean. *Transmisia datelor, notițe curs.*
- [2] <https://nodejs.org/en/>
- [3] <https://electronjs.org>
- [4] <https://octoverse.github.com/projects#languages>
- [5] <https://insights.stackoverflow.com/survey/2018/#technology>
- [6] https://www.w3schools.com/js/js_htmlDOM.asp

2SISTEME DE COMUNICAȚII CU FIR

2.1 OBIECTIVE



- Familiarizarea cu mediile de transmisie cu fir;
- Dezvoltarea unei aplicații client / server utilizând protocolul http;
- Dezvoltarea unei aplicații client / server utilizând protocolul websocket.

2.2 INTRODUCERE

2.2.1 MOTIVAȚIE

2.2.2 MEDII DE TRANSMISIE CU FIR

Medii ghidate Mediile de transmisie a informației se clasifică în funcție de tipul de semnale transportate. Avantajele transmisiei prin fir sunt:

- viteza de transmisie - raportul dintre cantitatea de informație transmisă și timpul necesar transmisiei;
- lățimea de bandă - cantitatea de informație pe care o poate transmite.
- distanța pe care se face transmisia - distanța dintre emițător și receptor pe care se face transmisia de date fără atenuări sau erori.

2.2.2.1 CABLUL COAXIAL

Banda de transmisie: 0 - 600 MHz

Este realizat dintr-un fir de cupru (miez) prin care informația este transmisă și o plasă de sârmă cu rol izolator la interferențe, separate printrul manșon de plastic. Informația transmisă este analogică sau digitală. figura 2.1

Cele mai uzuale cabluri coaxiale, figura 2.2, figura [3] (RG - radio guide; /U - utilizare generală):



Figure 2.1: Cablul coaxial

- RG-58 /U - utilizat pentru semnale low power si conexiuni RF, are rezistența de 50 Ohm și impedanța de 25 pF/ft (82 pF/m);
- RG-59 /U - utilizat pentru semnale video low power si conexiuni RF, are rezistența de 75 Ohm și impedanța de 20pF/ft (60pF/m); Cel mai des utilizat pentru distribuție CC TV.
- RG-62 /U - utilizat în aplicații industriale de interior pentru a transmite semnale de frecvențe înalte și de putere, are rezistența de 93 Ohm;
- RG-6/U - utilizat în aplicații rezidențiale, are rezistența de 75 Ohm; Cel mai des utilizat pentru distribuție TV.
- RG-11/U - utilizat în aplicații industriale de interior pentru a transmite semnale de frecvențe înalte și de putere, are rezistența de 75 Ohm. Cel mai des utilizat pentru distribuție HD TV (High Definition TV).

Conectori, figura 2.3, figura 2.4.

Avantaje și dezavantaje:

- + ieftin și durabil;
- + ușor de folosit;
- + rezistență bună la EMI;



Figure 2.2: Cablul coaxial



Figure 2.3: Conectori, (Sursa: [5], [6])

- + viteze de transfer până la 10 Mbps;
- mentenanța - un cablu defect, întreaga rețea este inactivă.



Figure 2.4: Conectori, (Sursa: [5], [6])

2.2.2.2 CABLUL TORSADAT

Cablul torsadat este format din mai multe perechi de fire de cupru izolate și se utilizează în comunicații. Torsadarea este o protecție electromagnetică, pentru reducerea sau anularea semnalelor parazite produse de alte circuite electrice învecinate și se obține prin răsucirea a două cabluri figura 2.5.

Un curent electric de intensitate variabilă produce la trecerea printr-un conductor, un câmp magnetic variabil. De asemenea, un câmp magnetic variabil induce într-un conductor o tensiune electrică variabilă. [7]

Dacă prin două conductoare electrice (ce închid un circuit) se transmite un semnal electric variabil, atunci curenții care le parcurg vor avea sensuri opuse. Aceștia vor genera două câmpuri electromagnetice în antifază care se vor anula reciproc.

Tipuri de cabluri torsadate (cele mai răspândite):

- FTP - Cablu cu perechi răsucite în folie - Foiled Twisted Pair;
- UTP - Cablu bifilar torsadat neecranat - Unshielded Twisted Pair, figura 2.6;
- STP - Cablu bifilar torsadat ecranat - Shielded Twisted Pair;
- S/UTP - Screened Unshielded Twisted Pair;
- S/FTP - Screened Foiled Twisted Pair;
- S/STP: Screened Shielded Twisted Pair, figura 2.7.

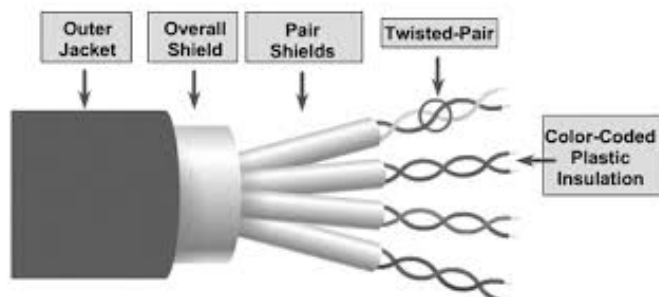


Figure 2.5: Cablul torsadat, 4 perechi

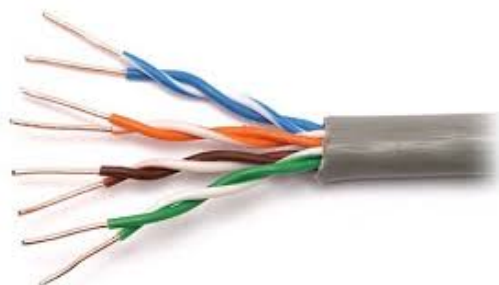


Figure 2.6: Cablul UTP, 4 perechi



Figure 2.7: Cablul S/STP, 4 perechi

Categorii de cabluri torsadate (Electronic Industries Association (EIA) a stabilit standardele UTP în 1995, [8]:

- CAT 1 - Categoria 1, se utilizează în rețele de telefonie. Se pot transmite doar semnale codificate analogic, cu frecvențe de până la 0,4 MHz și viteze de transmisie mai mici de 100 Kbps; Nu a fost proiectat pentru transmisia datelor.
- CAT 2 - Categoria 2, cablu cu 4 perechi torsadate. Se utilizează pentru semnale analogice și digitale cu viteze de transmisie de până la 4 Mbps; Utilizat în rețele LAN vechi (Token-passing ring LANs, IEEE 802.5). Lățime de bandă de 1 MHz.
- CAT 3 - Categoria 3, cablu cu 4 perechi torsadate, având 7-8 răsuciri pe metru; Se utilizează pentru viteze de transmisie Ethernet de până la 10 Mbps și semnale cu frecvențe de până la 16 MHz. Utilizat în vechile rețele Ethernet 10base-T LANs (IEEE 802.3).
- CAT 4 - Categoria 4, cablu cu 4 perechi torsadate. Se utilizează în rețele cu viteze de transmisie de până la 16 Mbps și frecvențe de până la 20 Mhz; Utilizat în rețele token-based sau 10Base-T.
- CAT 5 - Categoria 5, cea mai utilizată categorie de cablu în rețele Ethernet, având 4 perechi torsadate la un număr de 3 -4 răsuciri pe inch. Poate fi utilizat pentru viteze de transmisie de până la 155 Mbps sau frecvențe de până la 100 MHz pe distanțe de maxim 100 m. Transportă date cu viteze de până la 100 Mbps (Fast Ethernet, IEEE 802.3u) și este utilizat pentru rețele 100 base-T și 10base-T.
- Categoria 5e – cablu cu 4 perechi torsadate. Poate fi utilizat pentru viteze de transmisie de până la 1 Gbps sau frecvențe de până la 100 MHz pe distanțe de maxim 100 m. Pentru 100BASE-TX și 1000BASE-T Ethernet.
- CAT 6 - Categoria 6, cablu cu 4 perechi torsadate. Poate fi utilizat pentru viteze de transmisie de până la 10 Gbps sau frecvențe de până la 250 MHz pe distanțe de maxim 100 m. Pentru 10GBASE-T Ethernet.
- CAT 6a - Categoria 6a, cablu cu 4 perechi torsadate. Poate fi utilizat pentru viteze de transmisie de până la 10 Gbps sau frecvențe de până la 500 MHz pe distanțe de maxim 100 m. Pentru 10GBASE-T Ethernet.
- CAT 7 - Categoria 7, suportă până la 10 Gbps și frecvențe de până la 600 MHz. Pentru telefonie, CCTV, 1000BASE-TX în același cablu, 10GBASE-T Ethernet. Folosește F/FTP și S/FTP.
- CAT 7a - Categoria 7a, suportă până la 10 Gbps și frecvențe de până la 1000 MHz. Pentru telefonie, CATV, 1000BASE-TX în același cablu, 10GBASE-T Ethernet. Folosește F/UTP.

- CAT 8.1 - Categoria 8.1, suportă până la 40 Gbps și frecvențe de până la 1600 - 2000 MHz. Pentru telefonie, CATV, 1000BASE-TX în același cablu, 40GBASE-T Ethernet. Folosește F/UTP.
- CAT *.2 - Categoria 8.2, suportă până la 40 Gbps și frecvențe de până la 1600 - 2000 MHz. Pentru telefonie, CATV, 1000BASE-TX în același cablu, 40GBASE-T Ethernet. Folosește F/FTP și S/FTP.

Conectori UTP, FTP (standardele A și B de mufare), figura 2.8

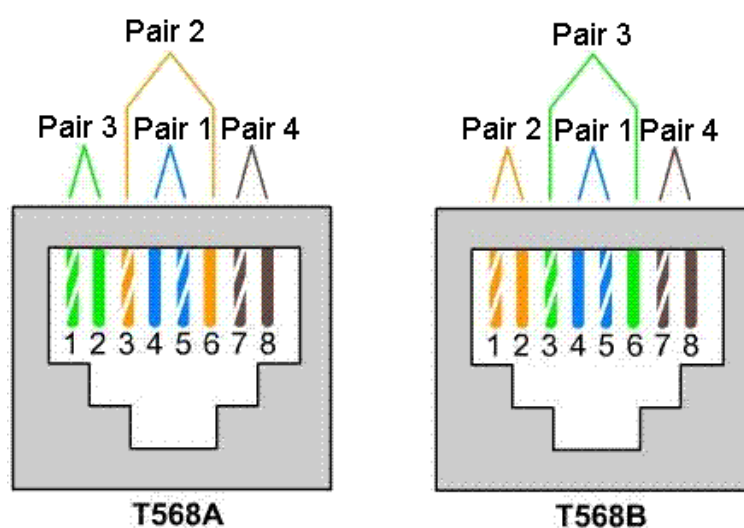


Figure 2.8: Conectori RJ-45, standardele 568A și 568B de mufare, (Sursa: [9], [10])

Avantaje și dezavantaje:

- + ieftin și ușor de instalat;
- + mai puțin sensibil la interferență cauzate de echipamentele din apropiere și la rândul lor nu provoacă interferențe;
- + STP poate transporta date la viteze mai mari;
- STP e mai voluminos și mai scump decât UTP;
- STP e mai dificil de mufat.

2.2.2.3 FIBRA OPTICĂ

Un cablu de fibră optică este format din fire de sticlă sau plastic (fibră optică); un singur cablu poate avea până la câteva sute de fire, figura 2.9. Miezul fibrei optice are dimensiuni extrem de reduse, 2 - 125 μm . Un singur fir poate transmite mai multe mii de apeluri telefonice. Capacitate de transmisie de ordinul gigabitilor pe mai multi kilometri având atenuări foarte mici (0.2 până la 1 dB/km) și o imunitate la zgomot foarte ridicată.

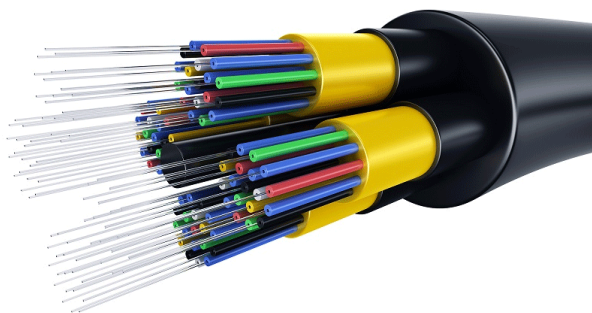


Figure 2.9: Fibră optică, (Sursa: [11])

Tipuri de fibră optică:

- fibră optică multi-mod (LED); Are un miez de 62.5 μm sau 50 μm , permițând moduri multiple de propagare a luminii.

Cablurile multi-mode pot trimite informații doar pe distanțe relativ scurte și sunt folosite pentru interconectarea a două rețele.

- fibră optică mono-mod (laser); Are un miez de diametru în jurul valorii de 9 μm și transmite doar un singur fascicul de lumină cu o lungime de undă specifică.

Televiziunea prin cablu, Internetul și semnalele telefonice sunt în general realizate prin fibre single-mode. Pot trimite informații la distanță de peste 100 km (60 mile).

Unele fibre optice premium au o degradare mult mai mică a semnalului - mai puțin de 10% / km la 1.550 nm.

Semnalele digitale sunt codificate în impulsuri analogice de lumină, NRZ – “Non-Return to Zero”. Exemple de conectori fibră optică sunt prezentate în figurile 2.11 și 2.10.

Cele mai multe fibre funcționează în duplex:

- una pentru transmisie, și

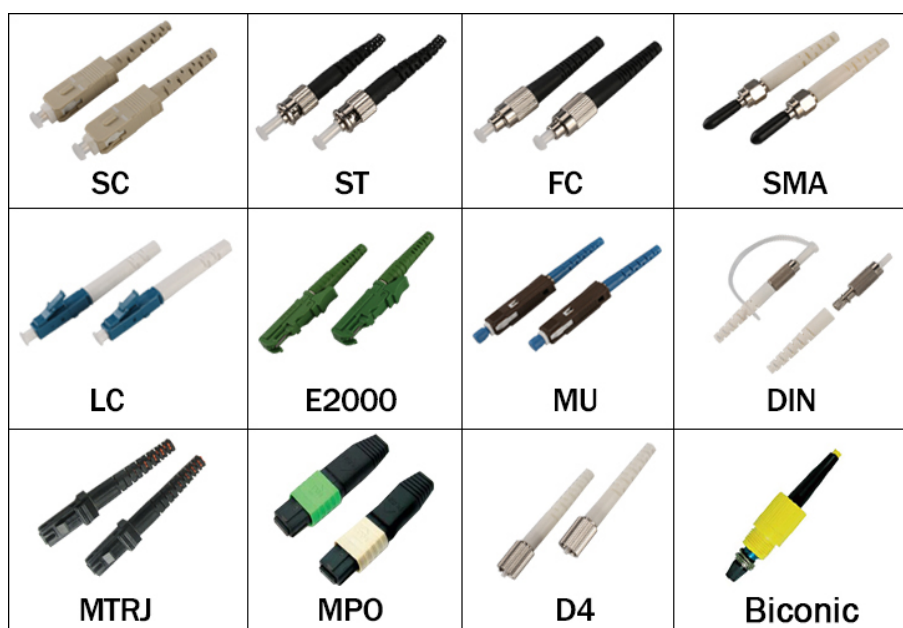


Figure 2.10: Conectori fibră optică, (Sursa: [13])

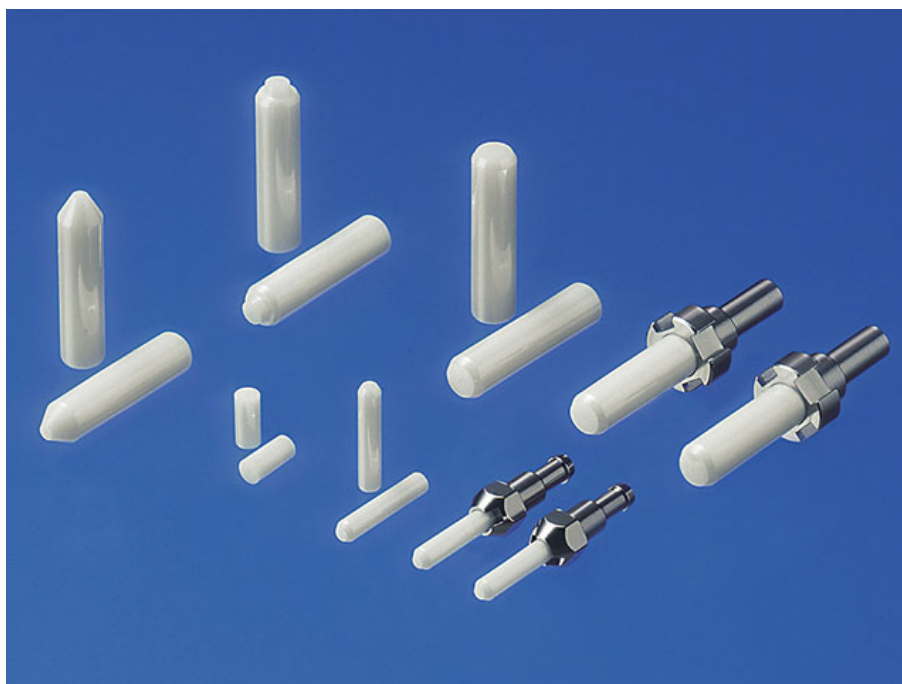


Figure 2.11: Conectori fibră optică - componente, (Sursa: [12])

- una pentru recepție.

Avantaje și dezavantaje:

- + volum redus în comparație cu celelalte cabluri;
- + viteza mai mare;
- + lățimea de bandă mult mai mare;
- + securitatea datelor;
- + imună la EMI;
- costuri ridicate;
- dificultate la cablarea unei noi rețele;
- fibra optică trebuie slefuită cu precizie pentru a transmite lumina cu pierderi mici.

2.2.3 TOPOLOGII DE REȚEA

Există mai multe topologii de rețea, figura [2.12](#):

- Magistrală - Într-o rețea de tip magistrală toate nodurile sunt conectate în linie.
- Stea - Toate nodurile sunt conectate individual cu un nod central.
- Inel - Fiecare nod este conectat cu două noduri diferite.
- Arbore - E o combinație între topologiile magistrală și stea.
- Fiecare cu fiecare - Fiecare nod este conectat cu fiecare nod în parte.
- Punct la punct - Două noduri cu drepturi egale sunt conectate între ele.
- Client / Server - Serverul este un dispozitiv cu putere de calcul mai mare și nodurile client au putere de calcul redusă în comparație cu serverul.

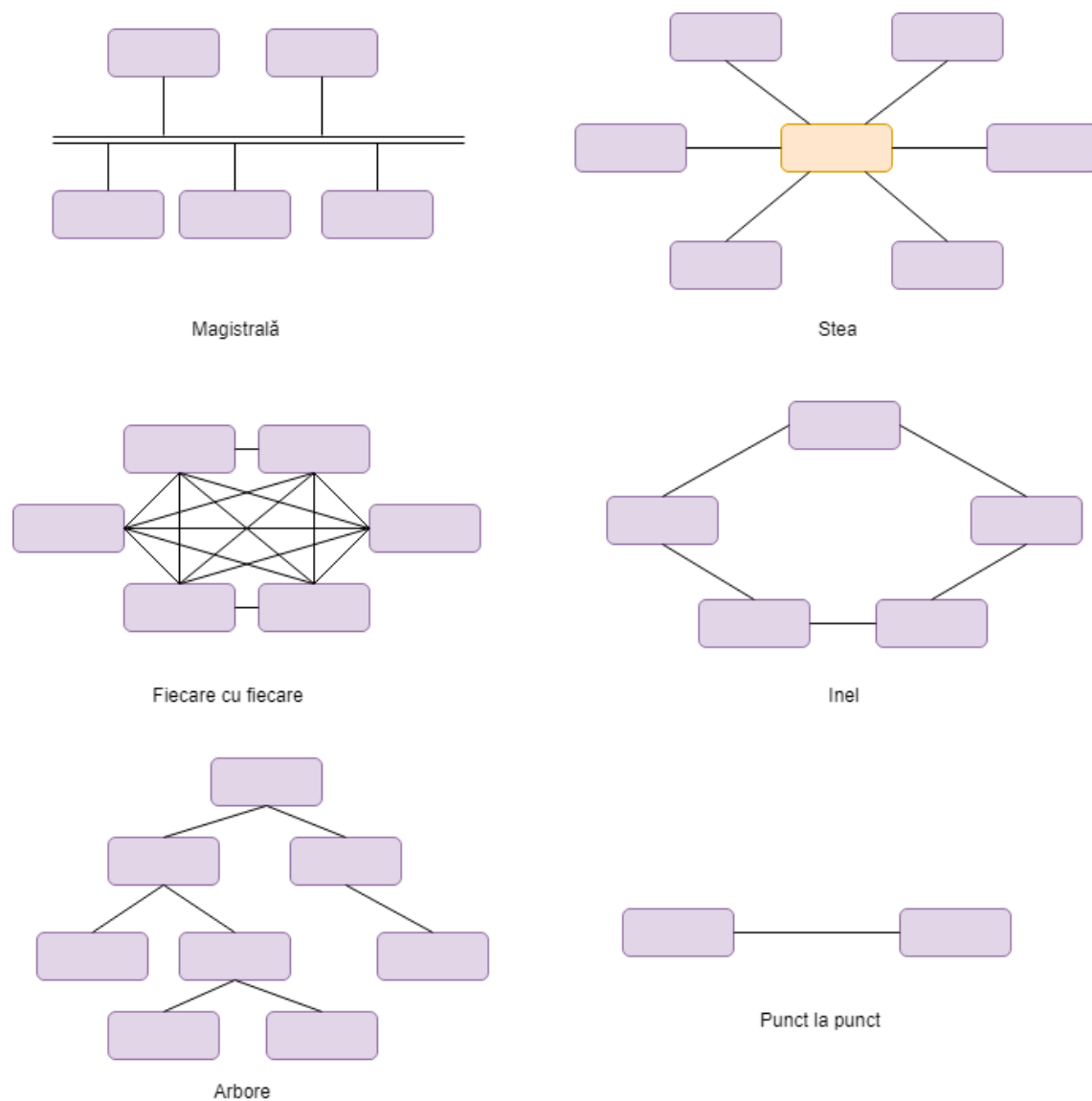


Figure 2.12: Topologii de rețea, (Sursa: [14])

2.2.4 APLICAȚII WEB

În prezent există mai multe tipuri de aplicații web, de la pagini web clasice, care utilizează o tehnologie de backend (php, java) pentru a gestiona și genera resurse web (html, css, js) până la aplicații complexe (Single page applications) care sunt executate izolat în browser și care sunt complet decuplate de modulul api care gestionează cererile acestor clienți și le expune o interfață de comunicare prin care se transferă datele. În continuare ne vom referi la al doilea tip de aplicații web pentru care există mai multe tehnologii actuale JavaScript care aduc modele arhitecturale software cât și mai multe funcționalități uzuale cum ar fi: arhitectura MVC (Model View Controller), legătură bidirecțională între modelul de date (variabile JavaScript) și interfața grafică, sintaxă HTML extinsă care facilitează afișarea datelor pe interfață.

2.2.5 ARHITECTURA CLIENT-SERVER

Arhitectura client-server este o arhitectură distribuită compusă din module client (care gestionează resurse) și servere (care oferă o interfață de acces la resurse). Clienții și serverele sunt aplicații software executate de cele mai multe ori pe mașini diferite. De asemenea, serverele operează în strânsă legătură cu diverse mecanisme de stocare a datelor (ex: server SQL, MongoDB, MySQL).

☞ Câteva exemple de resurse sunt: utilizatori ai aplicației, produsele afișate într-un magazin online, posturile/tweeturile de pe twitter/facebook, imaginile de pe instagram, emailurile.

Clienții (cunoscuți ca și aplicații frontend) gestionează datele la nivelul interfeței grafice și inițiază sesiuni de comunicație cu serverele (cunoscuți ca și aplicații backend) așteptând cereri (incoming requests) de la aceștia.

2.2.6 TCP/IP

Orice dispozitiv conectat la o rețea TCP (Transport Control Protocol)/IP are o adresă unică de identificare denumită IP (Internet Protocol).

Pentru ca o mașină (server, calculator) să poată expune mai multe aplicații și servicii pe rețea este nevoie, pe lângă adresa IPO, de mai multe porturi prin care mașina să transfere date.

IP-ul identifică serverul iar portul identifică aplicația sau serviciul care este executat pe server.

Figura următoare prezintă conectarea dintre două calculatoare pe rețeaua Internet folosind IP-uri și porturi.

Porturile iau valori pe 16 biți între 0 to 65535 și sunt grupate astfel:

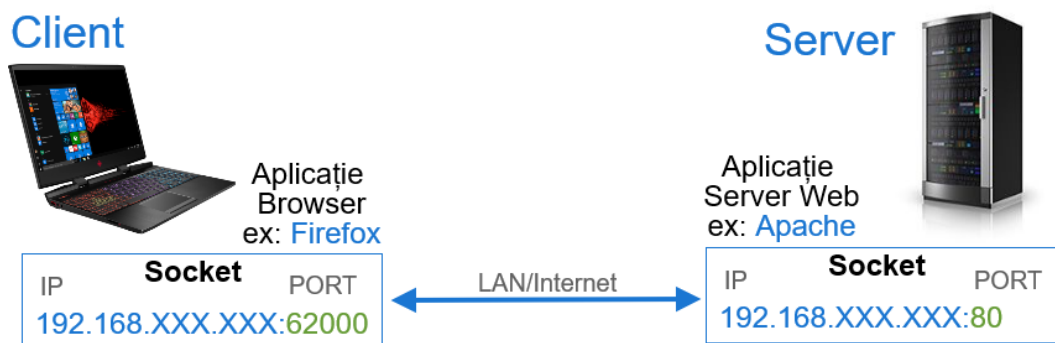


Figure 2.13: Conectare aplicație browser (client) - server web

- 0-1023: porturi alocate pentru servicii de sistem. Cele mai cunoscute porturi sunt: 80 (server web), 22 (SSH), 20 (FTP), 23 (SMTP).
- 1024-49151: porturi rezervate pentru aplicații client.
- 49152-65535: porturi dinamice/private.

O conexiune între două calculatoare folosește un socket (soclu) de comunicație care poate fi văzut ca o pereche de adresare IP:port. Așa cum reiese din figura 2.13 calculatoarele se conectează între ele folosind socketuri.

De exemplu atunci când accesăm site-ul Google se realizează o conexiune sursă-destinație între aplicația browser de pe calculatorul nostru local (care are atribuit un socket de felul 192.168.XXX.XXX:62000) și un server din infrastructura Google care servește aplicația web (216.58.XXX.XX:80). Porturile de pe calculatorul personal sunt atribuite dinamic aplicațiilor pentru fiecare sesiune de comunicație cu un socket extern și pot fi refolosite o dată ce această sesiune s-a încheiat.

TCP și UDP sunt protocoale aflate la nivelul transport de date pe când IP este implementat la nivelul rețea. Porturile sunt implementate la nivelul transport de date ca parte din antetul mesajelor TCP sau UDP, figura 2.14.

Protocolul TCP/IP suportă două tipuri de porturi TCP și UDP. TCP este un protocol de transport orietat pe conexiune care are implementat un mesaj de retransmisie a mesajelor în caz de eroare astfel încât toate packetele să ajungă la destinație. Este folosit pentru mesaje text, email, etc.

UDP este utilizat pentru transmisii multimedia (video, audio) și nu are implementat un mecanism care să asigure transferul cu succes al pachetelor de date.

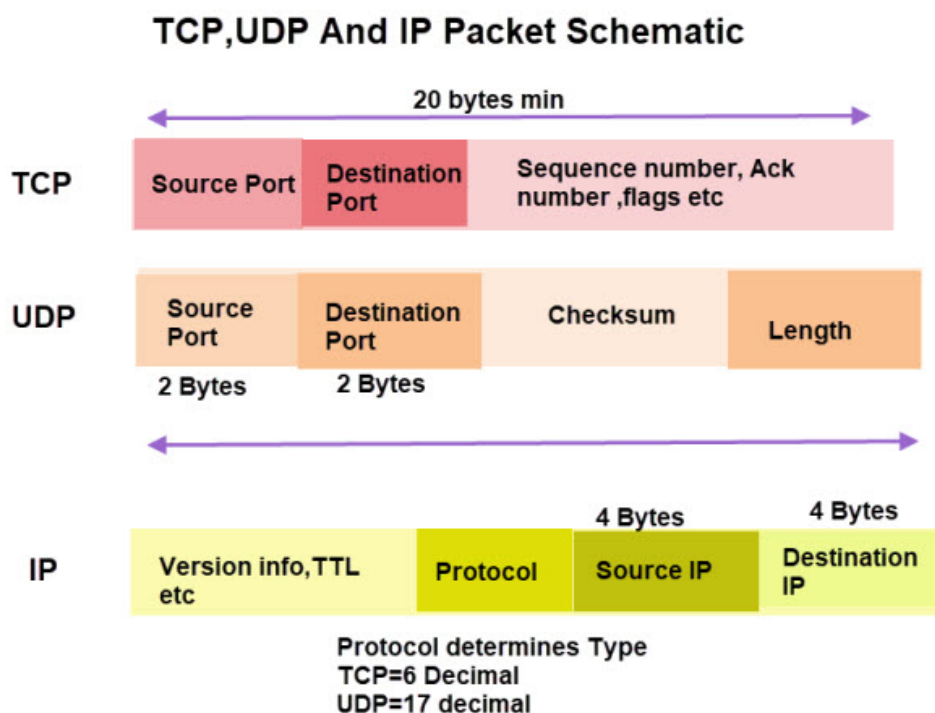


Figure 2.14: TCP, UDP, IP

2.2.7 HTTP VS. WEBSOCKET

Pentru transferul de date între diverse aplicații software conectate la rețea se folosesc în general socketuri (socluri de comunicație). Aceste socketuri pot să folosească protocoale de comunicație diverse din stiva TCP/IP cum ar fi: HTTP și WebSocket.

HTTP este un protocol unidirecțional care permite clienților să facă așa numitele cereri spre servere care gestionează resursele (eventual folosind o bază de date). Tipurile de cereri HTTP cele mai frecvent utilizate sunt: GET, PUT, POST și DELETE. Aceste 4 funcții permit citirea, salvarea, actualizarea și ștergerea resurselor.

WebSocket este o modalitate de a comunica bidirecțional între sisteme software distribuite. Cel mai simplu exemplu este un serviciu de chat în care clienții trimit mesaje spre server, care la rândul său le salvează și le trimite mai departe spre toți clienții interesați de aceste mesaje. Totuși prin websocketuri se pot trimite date audio și video.

JavaScript pune la dispoziție suport nativ pentru protocoalele HTTP și WebSockets pentru aplicații conectate la rețea.

2.3 DESFĂȘURAREA LUCRĂRII

Lucrarea constă în execuția unei aplicații compusă din două module: un modul client care expune către utilizator interfața grafică și un modul server care expune către modulul client puncte de interfațare (API). Modulele colaborează pentru a realiza împreună transmiterea de mesaje între utilizatori și gestiunea utilizatorilor care trimit mesaje prin intermediul modulului client. Comunicația între module se poate realiza în mai multe moduri și am ales să detaliem protocoalele HTTP și WebSocket.

Resurse folosite:

- biblioteca vue.js pentru modulul client
- biblioteca axios.js pentru modulul client
- mediul de execuție Node.js pentru modulul server
- npm - administratorul de pachete pentru Node.js

2.3.1 APLICAȚIE CLIENT CU VUE.JS

Fișierul **index.html** conține:

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta content="text/html; charset=utf-8" http-equiv="
        Content-Type">
6      <meta content="utf-8" http-equiv="encoding">
7      <title>Vue first app</title>
8      <link rel="stylesheet" href="style.css">
9      <script type="text/javascript" src="libs/vue/vue.js"></
        script>
10 </head>
11
12 <body>
13     <div id="app">
14
15         <input class="message" v-model="message" placeholder="
            Name">
16
17         <div class="output-empty" v-if="message == ''>The
            message is empty</div>
18         <div class="output" v-if="message != ''># {{ message
            }} #</div>
```

```
19
20     <button class="process-button" v-on:click="process()"
21         >Process</button>
22
23 </div>
24 <script src="app.js"></script>
25
26 </body>
27 </html>
```

Fișierul **app.js** conține:

```
1 var app = new Vue({
2   el: '#app',
3   data: {
4     message: ''
5   },
6   methods: {
7     process: function(){
8       console.log(this.message);
9     }
10  }
11 })
```

Fișierul **style.css** conține:

```
1 .message {
2   border: 1px solid #777;
3   padding: 7px;
4   margin: 5px;
5 }
6
7 .output-empty {
8   padding: 7px;
9   margin: 5px;
10  color: rgb(255, 104, 34);
11 }
12
13 .output {
14   padding: 7px;
15   margin: 5px;
16   color: rgb(27, 189, 27);
17 }
18
19 .process-button {
20   border: 1px solid #777;
```

```
21 background-color: #555;
22 color: white;
23 padding: 15px;
24 }
25
26 .process-button:hover{
27     background-color: #999;
28     cursor: pointer;
29 }
```

2.3.2 APLICAȚIE CLIENT-SERVER CU HTTP

Această aplicație realizează transmiterea de date folosind protocolul HTTP și, așa cum am zis, este împărțită în două module: modulul client și modulul server.

2.3.2.1 MODULUL SERVER

Modulul server este alcătuit din 4 fișiere: **api.js**, **run.js**, **users.json** și **package.json**.

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 **run.js** conține:

```
1 var api = require('./api.js').app;
2 var users = require('./users.json');
3
4 api.get('/', function(request, response) {
5     response.json("node.js backend");
6 });
7
8 api.get('/users', function(request, response) {
```



```
9   response.json(users);
10  });
11
12  api.put('/users', function(request, response) {
13    users[users.length] = request.body;
14    response.json('User was saved succesfully');
15  });
16
17
18  api.delete('/users/:index', function(request, response) {
19    users.splice(request.params.index, 1);
20    response.json('User with index ' + request.params.index +
21      ' was deleted');
22  });
23
24  api.listen(3000, function(){
25    console.log('CORS-enabled web server is listening on port
26      3000...');
```

Fișierul **users.json** conține:

```
1  [
2    {"name": "Cristina", "city": "Sebes"},
3    {"name": "Ion", "city": "Turda"},
4    {"name": "Sebastian", "city": "Bistrita-Nasaud"}
5  ]
```

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 }
```

Pentru a instala dependențele externe se execută comanda: **npm install** în linia de comandă.

Pentru a executa modulul server se execută comanda: **node run.js**.

Serverul răspunde requesturilor declarate în fișierul run.js (ex: <http://localhost:3000/users>)

2.3.2.2 MODULUL CLIENT

Modulul client este alcătuit din 4 fișiere: **index.html**, **app.js**, **users.js** și **style.css**. Modulul include librăriile externe vue.js și axios.js adaugate ca fișiere separate.

Fișierul **index.html** conține:

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5   <meta content="text/html; charset=utf-8" http-equiv="
      Content-Type">
6   <meta content="utf-8" http-equiv="encoding">
7   <title>Users manager</title>
8   <link rel="stylesheet" href="style.css">
9   <script type="text/javascript" src="libs/vue/vue.js"></
      script>
10  <script type="text/javascript" src="libs/axios/axios.js"
      ></script>
11  <script type="text/javascript" src="users.js"></script>
12 </head>
13
14 <body>
15   <div id="app">
16
17     <ul class="user-list">
18       <li v-for="(user, index) in users">
19         <div class="user-row">
20           <div class="user-name"><b>{{ user.name }}</b></
              div>
21           <div class="user-city"><i>{{ user.city }}</i></
              div>
22         </div>
23       </li>
24     </ul>
25
26   </div>
27   <script src="app.js"></script>
28 </body>
29
```

30 | `</html>`

Fișierul **app.js** conține:

```
1  var app = new Vue({
2    el: '#app',
3    data: {
4      users: [],
5      userService: null
6    },
7    created: function () {
8      userService = users();
9      userService.get().then(response => (this.users =
10         response.data));
11    },
12    methods: {
13    }
14  })
```

Fișierul **users.js** conține:

```
1  function users() {
2
3    get = function() {
4      return axios.get("http://localhost:3000/users");
5    }
6
7    return {
8      get: get
9    }
10 }
```

Fișierul **style.css** conține:

```
1  ul {
2    list-style-type: none;
3  }
4
5  .user-row {
6    margin: 20px auto;
7    background-color: rgb(255, 254, 234);
8    padding: 10px;
9    text-align: center;
10   width: 400px;
11 }
12
13 .user-row:hover {
```

```
14 background-color: rgb(255, 251, 175);
15 }
16
17 .user-name {
18     width: 150px;
19     display: inline-block;
20 }
21
22 .user-city {
23     width: 150px;
24     display: inline-block;
25 }
```

Pentru a executa modulul client este nevoie de un server care să servească resursele web. Un server simplu se poate instala pe platforma node.js executând următoarea comandă (din folderul clientului):

```
1 npm install http-server -g
```

După ce s-a instalat server-ul, aplicația client se execută cu comanda:

```
1 http-server
```

Aceasta va fi accesibilă în orice browser la adresele <http://localhost:8080> și <http://127.0.0.1:8080>.

2.3.3 APLICAȚIE CLIENT-SERVER CU WEBSOCKETS

2.3.3.1 SERVER

Fișierul **package.json** conține:

```
1 {
2   "name": "chat-server",
3   "version": "0.0.1",
4   "description": "Data Transmission websocket api",
5   "dependencies": {},
6   "devDependencies": {
7     "socket.io": "^2.2.0"
8   }
9 }
```

Fișierul **server.js** conține:

```
1 const io = require("socket.io");
2 const server = io.listen(8000);
3 let connectedClients = new Map();
4
```

```
5
6 server.on("connection", (socket) => {
7     console.info(`Client connected [id=${socket.id}]`);
8     connectedClients.set(socket);
9
10
11     socket.on("disconnect", () => {
12         connectedClients.delete(socket);
13         console.info(`Client [id=${socket.id}] disconnected`);
14     });
15
16     socket.on("message-from-client", (payload) => {
17         sendMessageToAllOtherClients(socket, payload);
18     });
19
20 });
21
22 function sendMessageToAllOtherClients(sender, message) {
23     for (const [client, sequenceNumber] of connectedClients
24         .entries()) {
25         if (sender.id !== client.id) client.emit("message-
26             from-server", message);
27     }
28 }
```

Dependințele specificate în package.json se descarcă folosind comanda:

```
1 npm install
```

Serverul se execută executând următoarea comandă în folderul în care se află serverul:

```
1 node server.js
```

2.3.3.2 CLIENT

Fișierul **package.json** conține:

```
1 {
2     "name": "chat-client",
3     "version": "0.0.1",
4     "description": "Data Transmission socket application",
5     "dependencies": {},
6     "devDependencies": {
7         "live-server": "^1.2.1",
8         "socket.io-client": "^2.2.0"
9     }
10 }
```

```
9   }  
10  }
```

Fișierul **client.js** conține:

```
1  var socket = io.connect('localhost:8000');  
2  
3  try {  
4  
5      socket.on('connect', function (data) {  
6          socket.emit("message-from-client", "Hello to  
          everyone from " + checkBrowser());  
7      });  
8  
9      socket.on('message-from-server', function (message) {  
10         console.log(message);  
11     });  
12  
13 }  
14 catch (err) {  
15     alert('ERROR: socket.io encountered a problem:\n\n' +  
        err);  
16 }  
17  
18 function checkBrowser() {  
19     var browser = 'Noname browser';  
20     if (navigator.userAgent.search("Chrome") > -1) {  
21         browser = "Chrome";  
22     }  
23     if (navigator.userAgent.search("Firefox") > -1) {  
24         browser = "Firefox";  
25     }  
26     return browser;  
27 }  
28  
29 document.getElementById("send").addEventListener("click",  
    sendMessage);  
30 function sendMessage() {  
31     var message = document.getElementById("message").value;  
32     socket.emit("message-from-client", message);  
33 }
```

Dependințele specificate în package.json se descarcă folosind comanda:

```
1  npm install
```

Clientul se pornește executând următoarea comandă în folderul clientului:

```
1 | live-server
```

Acesta va fi accesibilă în orice browser la adresele <http://localhost:8080> și <http://127.0.0.1:8080>.

2.4 APLICATII SI PROBLEME

2.4.1 PROBLEMA 1 (1 punct)

- Aplicație client VueJS (0,5 p).
- La apăsarea butonului 'Process, dacă valoarea din câmpul message este egală cu '123' atunci se afișează pe interfața grafică "Mesajul este egal cu 123" (0,5 p).

2.4.2 PROBLEMA 2 (4 puncte)

- Aplicație client-server HTTP (1 p).
- Să se implementeze ștergerea unui user (1 p).
- Să se implementeze adăugarea unui user nou (1 p).
- Să se implementeze modificarea unui user existent (pe baza indexului său în șirul de elemente) (1 p).

2.4.3 PROBLEMA 3 (3 puncte)

- Testarea aplicației client-server cu WebSocket folosind 2 clienți deschiși în browsere diferite (1 p).
- Să se implementeze cu VueJS o interfață grafică pentru un serviciu de mesagerie (ex: facebook messenger). Serverul rutează mesajul primit de la un client spre toți clienții conectați iar aceștia îl afișează în browser, nu doar la consolă. Se dorește păstrarea pe interfața grafică a mesajelor primite/transmise (2 p).

BIBLIOGRAFIE

- [1] Adina Aștilean. *Transmisia datelor, notițe curs*.
- [2] Andrew S. Tanenbaum, David J. Wetherall. *COMPUTER NETWORKS*. Prentice Hall Pub.
- [3] <https://uk.rs-online.com>
- [4] <https://tri-vcables.com/custom-assemblies/coax/>
- [5] <http://www.l-com.com/content/Article.aspx?Type=L&ID=10057>
- [6] <http://www.coax-connectors.com/>
- [7] <https://cdn.kramerav.com>
- [8] <http://www.berkteklevitontechnologies.com>
- [9] <http://www.groundcontrol.com/galileo/ch5-ethernet.htm>
- [10] <https://www.fiberoptics4sale.com>
- [11] <http://www.netcom-activ.ro/fibra-optica.php>
- [12] <https://global.kyocera.com/prdct/semicon/semi/fiber/>
- [13] <https://www.fs.com/fiber-optic-connector-tutorial-aid-341.html>
- [14] <https://www.orosk.com/>

3 DETECȚIE ȘI CORECȚIE DE ERORI - CODUL HAMMING

3.1 OBIECTIVE



- Familiarizarea cu algoritmi de detecție și corecție a erorilor;
- Dezvoltarea unei aplicații pentru detecția și corecția de erori folosind codul Hamming.

3.2 INTRODUCERE

3.2.1 NOȚIUNI TEORETICE

3.2.1.1 DISTANȚA HAMMING

Distanța Hamming, (numită după Richard Hamming), se calculează pentru doi vectori de dimensiuni egale. Distanța Hamming este dată de numărul de poziții în care vectorii sunt diferiți. Reprezintă numărul de erori care transformă un vector în celălalt (ex.: [3.1](#)).

Table 3.1: Distanța Hamming - exemple.

Vector 1	Vector 2	Distanța Hamming
1101011	1001001	= 2 (diferă pozițiile 2 și 6)
1111011	1100111	= 3 (diferă pozițiile 3, 4 și 5)

Informația este reprezentată în binar (ex.: cod ASCII). Operații obișnuite în binar: [3.2](#).

Table 3.2: Operații în binar: Adunare - "SAU" / "OR"; Înmulțire - "ȘI" / "AND"; Sau exclusiv - "SAU Ex." / "XOR".

+	0	1	*	0	1	⊕	0	1
0	0	1	0	0	0	0	0	1
1	1	0	1	0	1	1	1	0

3.2.1.2 CODURI LINIARE - HAMMING

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

$$V = [a_1 a_2 a_3 \dots a_n]; \quad (3.1)$$

Din cele n simboluri, k sunt de informație, iar m sunt de control. Prin operația de codare se înțelege calcularea simbolurilor de control în funcție de cele de informație, astfel încât $[V]$ să fie cuvânt de cod. În cazul codurilor liniare, simbolurile de control sunt combinații liniare ale simbolurilor de informație. Ele se obțin prin rezolvarea unui sistem liniar de m ecuații cu m necunoscute:

$$\begin{cases} h_{11} * a_1 \oplus h_{12} * a_2 \oplus \dots \oplus h_{1n} * a_n = 0 \\ h_{21} * a_1 \oplus h_{22} * a_2 \oplus \dots \oplus h_{2n} * a_n = 0 \\ \dots \\ h_{m1} * a_1 \oplus h_{m2} * a_2 \oplus \dots \oplus h_{mn} * a_n = 0 \end{cases} \quad (3.2)$$

Sistem de ecuații (3.2) se poate scrie sub forma:

$$[h][V]^T = [0]_{m \times 1} \quad (3.3)$$

În relațiile (3.2), (3.3), produsul este cel obișnuit, iar suma este modulo doi. Matricea:

$$H = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1n} \\ h_{21} & h_{22} & \dots & h_{2n} \\ \dots & & & \\ h_{m1} & h_{m2} & \dots & h_{mn} \end{bmatrix}; \quad h_{ij} \in [0, 1]; \quad (3.4)$$

reprezintă matricea de control a codului. Specificarea unui cod liniar implică precizarea matricii de control $[H]$. Proprietățile de detecție sau corecție ale codului se stabilesc prin alegerea matricii $[H]$ în mod corespunzător. Relația (3.3) reprezintă condiția

de codare: se consideră cuvinte de cod doar acele secvențe binare de n simboluri, care verifică relația (3.3).

Decodorul recepționează cuvântul eronat $[V'] = [V] = [\epsilon]$,

unde:

$$[\epsilon]' [\epsilon_1 \epsilon_2 \dots \epsilon_n]; \quad (3.5)$$

reprezintă cuvântul eroare, definit prin:

$$[\epsilon_i] = \begin{cases} 0, & \text{dacă pe poziția "i" nu s-a introdus o eroare.} \\ 1, & \text{dacă pe poziția "i" s-a introdus o eroare.} \end{cases} \quad (3.6)$$

Operația de decodare constă în calcularea corectorului:

$$[Z] = [H] * [V']^T; \quad (3.7)$$

ținând cont de legătura dintre $[V']$, $[V]$ și $[\epsilon]$ și de relația de codare (3.3) se poate scrie:

$$[Z] = [H] * ([V'] * [\epsilon])^T = [H] * [V]^T = [H] * [\epsilon]^T; \quad (3.8)$$

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

3.2.2 APLICAȚII

3.2.2.1 CODUL HAMMING - CORECTOR DE O EROARE

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

$$H = \begin{bmatrix} 0 & 0 & \dots & 0 & 1 \\ 0 & 0 & \dots & 1 & 0 \\ \dots & & & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix} = [h_1 h_2 \dots h_n]; \quad (3.9)$$

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

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

$$[V] = [c_1 c_2 a_3 c_4 a_5 a_6 a_7]; \quad (3.10)$$

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

Datorită acestei alegeri, fiecare relație din sistemul (3.2) va conține un singur simbol de control, permițând astfel rezolvarea ușoară a sistemului de m ecuații cu n necunoscute, după cum urmează:

$$\begin{cases} c_1 \oplus a_3 \oplus a_5 \oplus \dots \oplus 0 = 0 \\ c_2 \oplus a_3 \oplus a_5 \oplus \dots \oplus 0 = 0 \\ c_4 \oplus a_3 \oplus a_5 \oplus \dots \oplus 0 = 0 \\ \dots \end{cases} \quad (3.11)$$

Recepționându-se $[V']$, decodorul calculează $[Z] = [H][V']^T = [H][\epsilon]^T$. În absența erorii, $[\epsilon] = [0]$, deci $[Z] = [0]$. Dacă există o eroare pe poziția i , atunci:

$$[Z]' = [H] * [\epsilon]^T \quad (3.12)$$

În prezența erorii, corectorul $[Z]$ reprezintă coloana din matricea $[H]$ corespunzătoare poziției erorii, adică tocmai poziția erorii scrisă în binar. Pe baza corectorului $[Z]$ calculat din cuvântul recepționat, se poate determina poziția erorii și apoi corecta bitul eronat.

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

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

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \dots a_n; \quad (3.13)$$

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 și 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 ușor legătura dintre acestea:

$$[V_{H2}] = [c_0 V_{H1}]; \quad (3.14)$$

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 biți) și apoi a unei ultime linii de $n+1$ unități:

$$H_2 = \begin{bmatrix} 0 & \dots & & & \\ 0 & \dots & & & \\ \dots & & H_1 & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 1 \\ 0 & 1 & \dots & 0 \\ \dots & & & \\ 1 & 1 & \dots & 1 & 1 \end{bmatrix}; \quad (3.15)$$

La recepție se calculează:

$$Z_{H2} = [H_2] * [V_{H2}]^T = \begin{bmatrix} Z_{H1} \\ Z_0 \end{bmatrix}; \quad (3.16)$$

Dacă cuvântul recepționat este $[v'] = [c'_0 c'_1 c'_2 a'_3 c'_4 a'_5 \dots a'_n]$, din (3.16) rezultă:

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \dots a_n; \quad (3.17)$$

Comparând (3.13) cu (3.17) 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 $z_0 = 0$ și $[Z_{H1}] = [0]$, se decide că în cuvântul recepționat sunt două erori, iar când $z_0 = 1$ se decide că în cuvântul recepționat este o eroare pe poziția rezultată transcriind în zecimal componentele corectorului $[Z_{H1}]$;
- dacă $z_0 = 0$ și $[Z_{H1}] = [0]$ se decide că ceea ce s-a recepționat este corect.

3.2.3 EXEMPLIFICĂRI

3.3 DESFĂȘURAREA LUCRĂRII

3.3.1 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         </li>
27     </ul>
28     <br>
29     Data bits: <code>{{dataBits}}</code>
30     <br>
31
32     <button v-on:click="send()">Send</button>
33
34     <p>{{ status }}</p>
35
36 </div>
37
38 <script type="text/javascript" src="libs/vue.js"></
    script>
39 <script type="text/javascript" src="libs/axios.js"></
    script>
40 <script type="text/javascript" src="js/app.js"></
    script>
41 </body>
42
43 </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) ===
48                 false)
49                 return false;
50         }
51         return true;
52     },
53     validateBit: function(character){
54         if (character === null) return false;
55         return (parseInt(character) === 0 ||
56             parseInt(character) === 1);
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 }

```

3.3.2 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: '+
11      decoded.errorPosition);
12  }
13  response.json('Message received without errors');
14 }
15 api.listen(3000, function(){
16   console.log('CORS-enabled web server is listening on port
17     3000...');
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 }
    
```

3.4 PROBLEME PROPUSE

3.4.1 PROBLEMA (5 puncte)

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

BIBLIOGRAFIE

- [1] Adina Aștilean. *Transmisia datelor, notițe curs.*

4 COMPRESIA DATELOR - ALGORITMI STATICI DE COMPRESIE. ALGORITMUL HUFFMAN. ALGORITMUL SHANNON-FANO

4.1 OBIECTIVE



- Familiarizarea cu metode de compresie a datelor.
- Familiarizarea cu algoritmul Huffman de compresie a datelor.
- Familiarizarea cu algoritmul Shannon-Fano de compresie a datelor.

4.2 INTRODUCERE

Compresia datelor este un proces de codare prin care se urmărește micșorarea redundanței mesajului generat de o sursă, pentru a reduce resursele necesare memorării sau transmiterii acestui mesaj. Deoarece, de regulă, pentru memorarea sau transmiterea unui mesaj se folosește, eventual într-o formă intermediară, reprezentarea prin simboluri binare a acestuia, și pentru că cele mai multe metode de compresie folosesc metode de codare binar - binar, putem spune că obiectivul compresiei constă în reducerea numărului de simboluri binare necesar pentru reprezentarea mesajului.

Din punct de vedere al măsurii în care mesajul refăcut prin decompresie se aseamănă cu cel original, asupra căruia s-a acționat prin procesul de compresie, distingem două categorii de algoritmi de compresie: fără pierderi și cu pierderi.

Algoritmii de compresie fără pierderi sunt reversibili, prin decompresie obținându-se mesajul original întocmai. Algoritmii de compresie cu pierderi au ca rezultat diferențe relativ mici între mesajul rezultat prin decompresie și cel original și sunt utilizați în compresia imaginilor și a mesajelor video și audio.

În cele ce urmează ne referim la algoritmi de compresie fără pierderi, folosiți pentru texte sau date pur binare. Dacă se dispune de mesajul complet înainte de a începe procesul de compresie, se pot utiliza algoritmi statici de compresie, cum ar fi spre exemplu codarea Shannon – Fano și codarea Huffman statică, care țin seama de frecvența caracterelor în text, caracterelor cu probabilitatea mai mare de apariție în text alocându-li-se cuvinte de cod mai scurte. Cu algoritmi de compresie adaptivi, dinamici, procesul de compresie se realizează în timp real, probabilitatea de apariție a caracterelor fiind aproximată permanent, pe măsură ce sursa generează mesajul.

4.2.1 NOȚIUNI TEORETICE

Tehnicile de compresie sunt utile pentru fișierele text, în care anumite caractere apar cu o frecvență mai mare decât altele, pentru fișiere ce codifică imagini sau sunt reprezentări digitale ale sunetelor ori ale unor semnale analogice, ce pot conține numeroase motive care se repetă. Chiar dacă astăzi capacitatea dispozitivelor de memorare a crescut foarte mult, algoritmi de compresie a fișierelor rămân foarte importanți, datorită volumului tot mai mare de informații ce trebuie stocate. În plus, compresia este deosebit de utilă în comunicații, transmiterea informațiilor fiind mult mai costisitoare decât prelucrarea lor.

Una dintre cele mai răspândite tehnici de compresie a fișierelor text, care, în funcție de caracteristicile fișierului ce urmează a fi comprimat, conduce la reducerea spațiului de memorie necesar cu 20% – 90%, a fost descoperită de D. Huffman în 1952 și poartă numele de *codificare (cod) Huffman*. În loc de a utiliza un cod în care fiecare caracter să fie reprezentat pe 7 sau pe 8 biți (lungime fixă), se utilizează un cod mai scurt pentru caracterele care sunt mai frecvente și coduri mai lungi pentru cele care apar mai rar.

Definiții:

- *Compresia datelor* – metode și tehnici de reducere a volumului unui set de date reprezentate sub formă binară, cu posibilitatea reconstruirii lor fie exact, fie aproximativ.
- Set de date – șir finit de simboluri, cu reprezentare numerică (cod numeric original), în care numărul simbolurilor ce-l compun (volumul setului de date) > numărul simbolurilor distincte ce construiesc setul de date.
- $1 \text{ simbol} = 8 \text{ biți (ASCII)} \rightarrow$ setul de date se construiește din maxim $2^8 = 256$ simboluri distincte.
- *Alfabet* – mulțimea simbolurilor distincte dintr-un set de date.
- *Lungimea setului de date* – numărul total de biți pe care este reprezentat setul de date.

- *Volumul unui set de date* – numărul de simboluri care compun setul de date.
- *Compresie*
 - *conservativă* - metode de reconstrucție exactă a setului de date din versiunea sa comprimată – text, programe sursă.
 - *neconservativă* - metode de reconstrucție aproximative, inexacte, a setului de date original – semnal audio, imagine.

- *Rata de compresie* – parametru prin care se măsoară performanțele metodei de compresie utilizate:

$$\gamma = [1 - \frac{c}{o}]100\%; \quad (4.1)$$

unde:

- c - lungimea setului de date comprimat.
- o - lungimea setului de date original.
- γ :
 - = 100%, metoda de compresie ideală, (câștig maxim).
 - = 0%, metoda nu realizează compresia, (câștig nul).
 - < 0%, metoda nu realizează compresia, (expandarea datelor).
- γ - măsoară câștigul relativ de lungime obținut prin utilizarea unei metode de compresie.
- *Viteza de compresie* – $\frac{1}{\gamma}$ (depinde de complexitatea metodei, compromis între cei doi parametri).
- *Compresie*:
 - *conservativă* - rata de compresie mare, viteza de compresie scăzută.
 - *neconservativă* - rata de compresie mică, viteza de compresie mare.
- *Expandare* – creșterea lungimii setului de date procesat față de cel original (în cazul metodelor ineficiente).
- *Decompresie* – acțiunea de reconstruire (exactă sau aproximativă) a setului de date original, plecând de la setul de date original.
- *Redundanța* – se măsoară cu ajutorul unui model statistic asociat setului de date, plecând de la frecvențele de apariție ale simbolurilor alfabetului de date.

- *Frecvența de apariție* – raportul între numărul aparițiilor simbolului și numărul total de simboluri ai setului de date notate cu $v(s)$. (probabilitatea de apariție a simbolului "s", $P(s)$).
- *Entropie* – numărul minim de biți necesari pentru recodificarea simbolului "s", fără a se realiza confuzia cu alte simboluri. (poate fi fracționar, dar de obicei se rotunjește la întregul superior). Va fi cel mult egală cu numărul inițial de biți.

$$\begin{aligned} H(s) = -\log_2 v(s) \rightarrow v(s) = 2^{-H(s)} \in (0, 1]. \\ 0 < v(s) \leq 1 \rightarrow H(s) > 0. \end{aligned} \quad (4.2)$$

4.2.1.1 CRITERII DE APRECIERE A UNUI COD

Obiectivul unei metode de compresie a datelor este detectarea și eliminarea redundanțelor din setul de date. Deoarece la transmiterea mesajelor costul explorării unui sistem de transmisie crește liniar cu timpul, un criteriu convenabil de apreciere a unui cod este lungimea medie a unui cuvânt:

$$n = \sum_{i=1}^n n_i P_i. \quad (4.3)$$

unde:

- P_i sunt probabilitățile asociate alfabetului sursei.
- n_i este numărul de litere din cuvântul de cod cu indicele i .
- n este un parametru care precizează compactitatea codului, fiind evident că n trebuie să fie cât mai mic.

n este limitat inferior de condiția de asigurare a entropiei informaționale pe simbol al alfabetului de cod:

$$n \geq n_{min} = \frac{H}{\lim q} \quad (4.4)$$

unde H reprezintă entropia sursei. În aceste condiții, eficiența unui cod este definită de formula:

$$\eta = \frac{\overline{n_{min}}}{\overline{n}}. \quad (4.5)$$

4.2.1.2 CODUL HUFFMAN

D. Huffman a elaborat un algoritm Greedy care construiește un cod prefix optimal, numit cod Huffman. Prima etapă în construcția codului Huffman este calcularea numărului de apariții ale fiecărui caracter în text. Există situații în care putem utiliza frecvențele standard de apariție a caracterelor, calculate în funcție de limbă sau de specificul textului.

Fie $C = c_1, c_2, \dots, c_n$ mulțimea caracterelor dintr-un text, iar f_1, f_2, \dots, f_n , respectiv, numărul lor de apariții. Dacă l_i ar fi lungimea șirului ce codifică simbolul c_i , atunci lungimea totală a reprezentării ar fi:

$$L = \sum_{i=1}^n l_i f_i. \quad (4.6)$$

Scopul nostru este de a construi un cod prefix care să minimizeze această expresie. Pentru aceasta, construim un arbore binar complet în manieră bottom-up astfel:

- Inițial, considerăm o partiție a mulțimii $C = \{\{c_1, f_1\}, \{c_2, f_2\}, \dots, \{c_n, f_n\}\}$, reprezentată printr-o pădure de arbori formați dintr-un singur nod.
- Pentru a obține arborele final, se execută $n-1$ operații de unificare.

Unificarea a doi arbori A_1 și A_2 constă în obținerea unui arbore A , al cărui subarbore stâng este A_1 , subarbore drept A_2 , iar frecvența rădăcinii lui A este suma frecvențelor rădăcinilor celor doi arbori. La fiecare pas unificăm 2 arbori ale căror rădăcini au frecvențele cele mai mici.

Construcția arborelui pleacă de la frunze spre rădăcină → codurile simbolurilor sunt construite de la bitul cel mai puțin semnificativ la bitul cel mai semnificativ.

Pasul 1: Se parcurge secvențial D și se construiește $A^0 \rightarrow N(s)$ pentru fiecare $s \in A^0$

Pasul 2: Rearanjarea simbolurilor alfabetului: $A^0 = \{s_1, s_2, \dots, s_n\}$ cu $N(s_1) \geq N(s_2) \geq \dots \geq N(s_n)$. Se renotează și reindexează elementele alfabetului astfel:

$$A^0 = \{t_{N+k-1} = s_k\}_{k \in \overline{1, N}}$$

Simbolurile $t_n, t_{n+1}, \dots, t_{2n-1}$ sunt utilizați pentru etichetarea frunzelor arborelui binar, iar alfabetul A^0 devine primul alfabet curent al algoritmului și va fi renotat prin $A^{0, N}$. Fiecare alfabet curent al algoritmului se va nota prin $A^{0, (N-k)}$, $k = 0 \dots n-1$ $k=0 \rightarrow$ frunzele arborelui sunt etichetate cu simbolurile primului alfabet curent.

Pasul 3: Construcția arborelui binar se realizează simultan cu formarea noului alfabet curent. Noul alfabet curent, $A^{0, (n-k-1)}$ cu $k=0 \dots N-2$, se construiește din vechiul alfabet $A^{0, (N-k)}$, aplicându-se următoarea regulă:

se elimina 2 simboluri cu cele mai mici contoare din vechiul alfabet curent, înlocuindu-le cu un simbol virtual, notat prin " γ " și care are contorul egal cu suma contoarelor celor două simboluri.

Dacă există mai mult de două simboluri slabe având același contor, se aleg ultimele două din alfabet \rightarrow cardinalul noului alfabet scade cu o unitate față de cardinalul alfabetului precedent.

$$A^{0,(n-k-1)} = [A^{0,(n-k)} \{t_p, t_q\}] \cup \{t_{n-k-1}\}$$

$$t_{n-k-1} = \text{simbol virtual}$$

$$N(t_{n-k-1}) = N(t_p) + N(t_q)$$

$$\#A^{0,(n-k-1)} = \#A^{0,(n-k)} - 1 = n - k - 1$$

Simbolul virtual etichetează un nod intermediar al arborelui, având ca descendenți cele două noduri etichetate de simboluri eliminate. Fiecare nod intermediar din arbore are câte 2 descendenți, contorul fiului din stânga fiind cel puțin egal cu cel al fiului din dreapta.

Pasul 4: Se repetă pasul 3 de " $n-1$ " ori pentru fiecare alfabet curent $A^{0,(n-k)}$ până când se etichetează și rădăcina arborelui (t_1).

Pasul 5: Se reindexează nodurile arborelui astfel încât fiii oricărui nod să fie etichetați cu indici consecutivi, de la stânga la dreapta, începând de la rădăcină.

4.2.1.3 CODUL SHANNON - FANO

Metoda Shannon – Fano constă în construcția arborelui binar asociat setului de date, după următorul algoritm:

Pasul 1: Parcurgerea setului de date (D) în mod secvențial, citind fiecare simbol în parte:

- se construiește alfabetul de ordin 0 (A^0).
- se contorizează numărul de apariții ale fiecărui simbol $s \in A^0 \rightarrow$ contorul $N(s) \in N$.

$$\nu(s) = \frac{N(s)}{D}, \forall s \in A^0 \text{ (estimarea probabilității de apariție a lui s)}$$

$$D = \sum_{x \in A^0} N(s) \text{ (volumul setului de date este constant)}$$

Pasul 2: Simbolurile din A^0 se aranjează în ordine descrescătoare a valorilor contoarelor, în caz de egalitate se aranjează lexicografic.

Pasul 3: Alfabetul A^0 se divizează în două părți disjuncte, numite subalfabete (A^{0L} și A^{0R}), dar se păstrează două condiții de bază:

- se păstrează ordinea simbolurilor.
- ponderile subalfabetelor să fie cât mai apropiate.

$$A^0 = A^{0L} \cup A^{0R}.$$

$$A^{0L} \cap A^{0R} = \emptyset.$$

$$N(A^{0L}) = \sum_{s \in A^{0L}} N(s) \approx N(A^{0R}) = \sum_{s \in A^{0R}} N(s)$$

La construcția subalfabetelor se pleacă simultan din cele două capete ale alfabetului A^0 , mergând spre centru, comparând mereu ponderile parțiale ale subalfabetelor, până la epuizarea simbolurilor alfabetului A^0 .

Pasul 4: Construcția arborelui binar. Ramura dreaptă primește ponderea "1" iar ramura stângă primește ponderea "0".

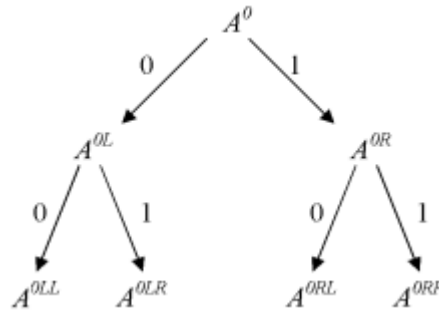


Figure 4.1: Arbore Shannon-Fano

Pasul 5: Se repetă pașii 3 și 4 pentru fiecare nod terminal al arborelui (subalfabet) care conține mai mult de un simbol.

Stop: Procesul de construcție a arborelui se oprește când toate subalfabetele nodurilor terminale (frunze) conțin doar un singur simbol. Codul unui simbol se obține parcurgând arborele de la rădăcină până la nodul ocupat de acel simbol, concatenând valorile arcelor prin care se trece.

Observație: Pentru un simbol frecvent în setul de date codul corespundent este scurt, pentru că nodul lui va fi în apropierea rădăcinii. Simbolurile rare au o lungime mai mare a codului.

4.2.2 APLICAȚII

4.2.2.1 ALGORITMUL HUFFMAN

Construcția arborelui binar

Setul de date, D: "IT IS BETTER LATER THAN NEVER."

Alfabetul asociat setului de date, D, este:

A^0		E	T	R	A	I	N	.	B	H	L	S	V
N(s)	5	5	5	3	2	2	2	1	1	1	1	1	1

Construirea arborelui se face pornind de la frunze spre rădăcină.

Pasul 1: Fiecare simbol cu frecvența de apariție este trecut într-o etichetă, 4.2.

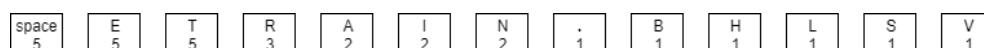


Figure 4.2: Arbore Huffman - pasul 1

Pasul 2: Se unesc simbolurile cu frecvența cea mai mică, 4.3.

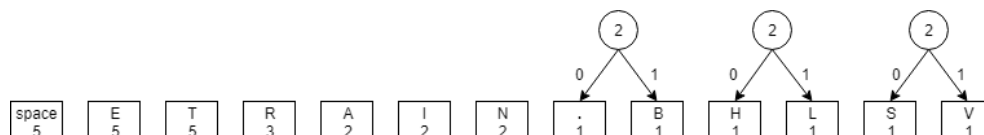


Figure 4.3: Arbore Huffman - pasul 2

Pasul 3: Se unesc simbolurile și /sau nodurile cu frecvența cea mai mică, 4.4.

Pasul 4: Se unesc simbolurile și /sau nodurile cu frecvența cea mai mică, 4.5.

Pasul 5: Se unesc simbolurile și /sau nodurile cu frecvența cea mai mică, 4.6. Dacă este nevoie se aranjează arborele.

Pasul 6: Se unesc simbolurile și /sau nodurile cu frecvența cea mai mică, 4.6. Dacă este nevoie se aranjează arborele.

Pasul 7: Arborele final, 4.8.

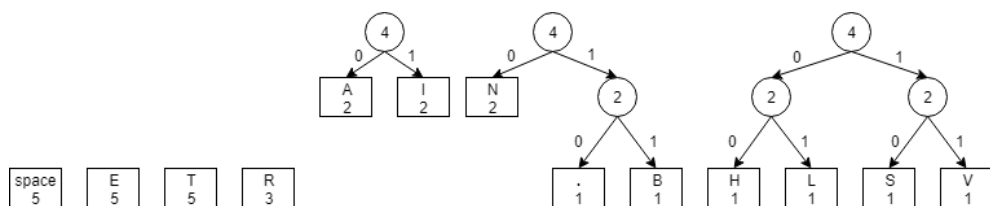


Figure 4.4: Arbore Huffman - pasul 3

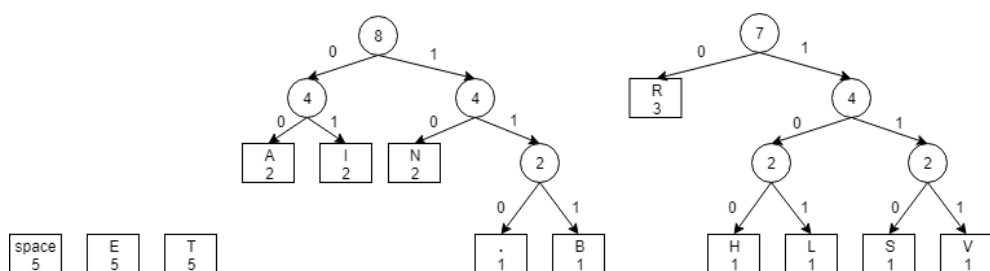


Figure 4.5: Arbore Huffman - pasul 4

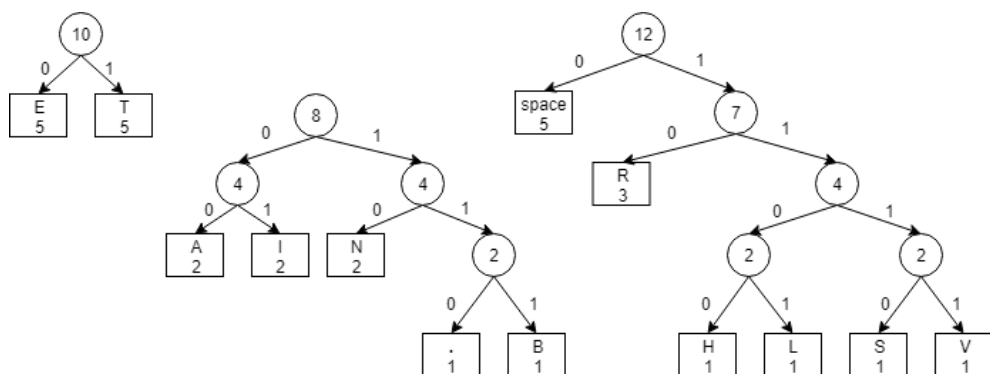


Figure 4.6: Arbore Huffman - pasul 5

Noile coduri asociate setului de date D sunt:

A^0		E	T	R	A	I	N	.	B
Cod nou	00	110	111	010	1000	1001	1010	10110	10111
Nr. biți	2	3	3	3	4	4	4	5	5

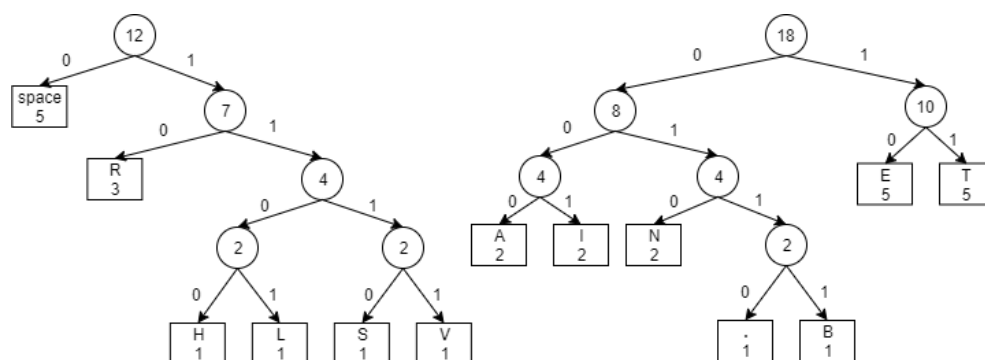


Figure 4.7: Arbore Huffman - pasul 6

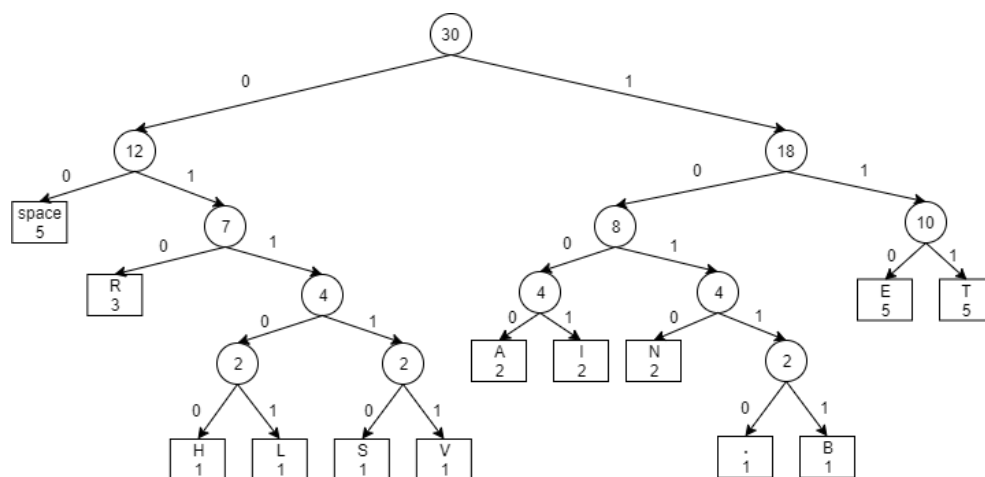


Figure 4.8: Arbore Huffman

A^0	H	L	S	V
Cod nou	01100	01101	01110	01111
Nr. biți	5	5	5	5

Codurile numerice sunt exprimate în zecimal, dar în realitate aceste coduri sunt binare și au câte o lungime de 8 biți. Aranjarea informației auxiliare poate fi realizată și în altă manieră, dar modul de aranjare trebuie cunoscut și la decompresia datelor.

Cod	1001	101	00	1001	01110	00	10111	110	111	111	110	010
Info.	I	T		I	S		B	E	T	T	E	R
nr. biti	4	3	2	4	5	2	5	3	3	3	3	3

Cod	00	01100	1000	111	110	010	00	111	01100	1000	1010
Info.		L	A	T	E	R		T	H	A	N
nr. biti	2	5	4	3	3	3	2	3	5	4	4

Cod	00	1010	110	01111	110	010	10110
Info.		N	E	V	E	R	.
nr. biti	2	4	3	5	3	3	5

4.2.2.2 ALGORITMUL SHANNON - FANO

Setul de date, D: "IT IS BETTER LATER THAN NEVER."

Construcția arborelui binar, alfabetul asociat setului de date, D, este:

A^0		E	T	R	A	I	N	.	B	H	L	S	V
N(s)	5	5	5	3	2	2	2	1	1	1	1	1	1

$N(A^0) = 30$, arborele binar se construiește în cinci iterații de vizitare în subalfabete.

Pas 1:

$$N(A^{0L}) = 15$$

A^{0L}		E	T
N(s)	5	5	5

$$N(A^{0R}) = 15$$

A^{0R}	R	A	I	N	.	B	H	L	S	V
N(s)	3	2	2	2	1	1	1	1	1	1

Simbolurile cu ponderi egale sunt aranjate în ordine lexicografică.

Pas 2:

$$N(A^{0LR}) = 10$$

A^{0LR}	E	T
N(s)	5	5

$$N(A^{0RL}) = 7$$

A^{0RL}	R	A	I
N(s)	3	2	2

$$N(A^{0RR}) = 8$$

A^{0RR}	N	.	B	H	L	S	V
N(s)	2	1	1	1	1	1	1

Simbolul “spațiu” este cel mai frecvent simbol folosit și a atins o frunză fiind recodificat pe 2 biți: 00 (în loc de codul original pe 8 biți 0010 0000, adică 32).

Pas 3:

$$N(A^{0RLR}) = 4$$

A^{0RL}	A	I
N(s)	2	2

$$N(A^{0RRL}) = 4$$

A^{0RR}	N	.	B
N(s)	2	1	1

$$N(A^{0RRR}) = 4$$

A^{0RRR}	H	L	S	V
N(s)	1	1	1	1

Simbolurile cele mai frecvente sunt recodificate în această iterație, iar noile lor coduri nu depășesc lungimea de 3 biți (în loc de 8).

Pas 4:

$$N(A^{0RRLR}) = 2$$

A^{0RR}	.	B
N(s)	1	1

$$N(A^{0RRRL}) = 2$$

A^{0RRRL}	H	L
N(s)	1	1

$$N(A^{0RRRR}) = 2$$

A^{0RRRR}	S	V
N(s)	1	1

Numai simbolurile foarte rare din setul de date au mai rămas de recodificat, dar codurile lor vor avea o lungime de 5 biți în loc de 8 biți.

Pas 5:

D = IT IS BETTER LATER THAN NEVER. Noile coduri nu depășesc 5 biți → minimizarea redundanței și deci maximizarea ratei de compresie.

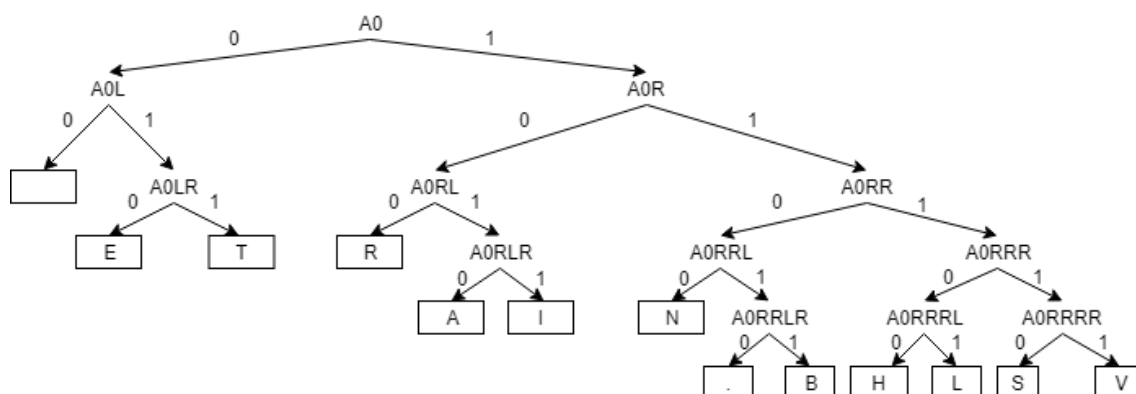


Figure 4.9: Arbore Shannon-Fano

Codurile numerice sunt exprimate în zecimal, dar în realitate aceste coduri sunt binare și au câte o lungime de 8 biți. Aranjarea informației auxiliare poate fi realizată și în altă manieră, dar modul de aranjare trebuie cunoscut și la decompresia datelor.

Cod	1011	011	00	1011	11110	00	11011	010	011	011	010
Info.	I	T		I	S		B	E	T	T	E
nr. biti	4	3	2	4	5	2	5	3	3	3	3

Cod	100	00	11101	1010	011	010	100	00	011	11100	1010
Info.	R		L	A	T	E	R		T	H	A
nr. biti	3	2	5	4	3	3	3	2	3	5	4

Cod	1100	00	1100	010	11111	010	100	11010
Info.	N		N	E	V	E	R	.
nr. biti	4	2	4	3	5	3	3	5

Codurile informației utile și cele ale informației auxiliare se succed fără separatori între ele. Între informația auxiliară și cea utilă ar putea să apară un separator sub forma unui simbol virtual, în absența numărului de simboluri ai alfabetului, N. În acest caz,

separatorul de informație trebuie să aibă un cod mai mare de 255, de exemplu 256, dar reprezentarea lui ar fi pe 16 biți, în loc de 8 biți, cât ocupă N.

Analiza performanțelor de compresie

Vom face o analiza a entropiei (numărul de biți efectiv alocați în urma compresiei). Există două tipuri de entropii: cea a informației auxiliare $H_a(D)$ și cea a informației utile $H_u(D)$.

Decompresia datelor:

- se citește informația auxiliară (5 octeți consecutivi indică o pereche simbol – contor);
- se construiește arborele binar asociat;
- se citește informația utilă secvențial, bit cu bit;
- fiecare simbol al setului de date este decriptat și se folosește un arbore binar.

Exemplu:

- Primul bit este 1 → deplasarea de la A^0 la A^{0R} .
- Al doilea bit 0 → deplasarea spre A^{0RL} .
- Al treilea bit 1 → deplasarea spre A^{0RLR} .
- Al patrulea bit 1 → atingerea simbolului I, care va fi înscris pe fluxul de ieșire.

Atingerea unei frunze reinițializează căutarea din rădăcina arborelui.

Observație: Avantajul metodei constă în faptul că nu este necesară cunoasterea dimensiunilor noilor coduri de compresie care în general variază de la un simbol la altul.

4.3 DESFĂȘURAREA LUCRĂRII

Lucrarea constă în compresia unui text folosind cei doi algoritmi prezentați mai sus.

Resurse folosite:

- biblioteca d3js pentru afisarea datelor sub forma de arbore binar
- npm - administratorul de pachete pentru Node.js

4.3.1 APLICAȚIE CU D3JS

Aplicația următoare poate fi folosită pentru reprezentarea arborilor binari și are la bază unul dintre exemplele despre "collapsible tree" din librăria D3. Nodul rădăcină este A0 și fiecare nod are maxim două ramuri.

Pentru a instala dependențele externe se execută comanda: **npm install d3** în linia de comandă.

Pentru a instala serverul http: **npm install -g http-server** în linia de comandă.

Pentru a porni serverul: **http-server &** în linia de comandă.

Fișierul **index.html** conține:

```
1
2 <!DOCTYPE html>
3 <html>
4 <head>
5   <meta content=" text/html; charset=utf -8" http-equiv="
      Content-Type">
6   <meta content ="utf-8" http-equiv="encoding">
7   <!-- load the style.css -->
8   <link rel="stylesheet" href="style.css">
9   <!-- load the d3.js library https://d3js.org/d3.v5.min.js
      -->
10  <script src="https://d3js.org/d3.v5.min.js"></script>
11 </head >
12 <body>
13   <!-- arbore.js - a function to display a tree
14        - is based on "collapsible tree" sample from d3
15        library
16   -->
17   <script src="arbore.js"></script>
18 </body>
19 </html>
```

Fișierul **style.css** conține:

```
1 .node circle {
2   fill: #fff;
3   stroke: steelblue;
4   stroke-width: 3px;
5 }
6
7 .node text {
```

```
8   font: 12px sans-serif;
9 }
10
11 .link {
12   fill: none;
13   stroke: #ccc;
14   stroke-width: 2px;
15 }
```

Fișierul **arbore.js** conține:

```
1  %
2  Exemplu de afisarea unui arbore de compresie.
3  A0 este radacina arborelui.
4  Fiecare nivel introduce un bit la finalul codului.
5  Fiecare nod va avea maximum 2 copii (L - stanga (introduce
6    un "0") si R - dreapta (introduce un "1"))
7  Numele nodului contine numele nodului vizitat anterior.
8
9  ex.: A0LR - nivel 2, cod: 01
10 %
11 var treeData = {
12   "name": "A0",
13   "children": [
14     {
15       "name": "Level 1: _",
16       "children": [
17         { "name": "Level 2: _",
18           "children": [
19             { "name": "Level 3: _",
20               "children": [
21                 { "name": "Level 4: _ - space (frequency)" },
22                 { "name": "Level 4: _ - 3 (frequency)" }
23               ]
24             },
25             { "name": "Level 3: _ - q (frequency)" }
26           ]
27         },
28         { "name": "Level 2: _ - M (frequency)" }
29       ]
30     },
31     { "name": "Level 1: _",
32       "children": [
```

```
33         { "name": "Level 2: _ - A (frequency)" },
34         { "name": "Level 2: _ - * (frequency)" }
35     ]
36 }
37 ]
38 };
39
40
41 Definirea dimensiunii de afisare a arborelui
42 var margin = {top: 20, right: 90, bottom: 30, left: 90},
43     width = 1960 - margin.left - margin.right,
44     height = 800 - margin.top - margin.bottom;
45
46
47
48
49
50 var svg = d3.select("body").append("svg")
51     .attr("width", width + margin.right + margin.left)
52     .attr("height", height + margin.top + margin.bottom)
53     .append("g")
54     .attr("transform", "translate("
55         + margin.left + "," + margin.top + ")");
56
57 var i = 0,
58     duration = 750,
59     root;
60
61
62 defineste layout pentru arbore si ii asigneaza dimensiuni
63
64 var treemap = d3.tree().size([height, width]);
65
66
67 Asigneaza parinte, copil, inaltime, adancime
68
69 root = d3.hierarchy(treeData, function(d) { return d.
    children; });
70 root.x0 = height / 2;
71 root.y0 = 0;
72
73
74 Inhide dupa al doilea nivel
```

```
75
76 root.children.forEach(collapse);
77
78 update(root);
79
80
81   Inhide nodul si toti copiii lui
82
83 function collapse(d) {
84   if(d.children) {
85     d._children = d.children
86     d._children.forEach(collapse)
87     d.children = null
88   }
89 }
90
91 function update(source) {
92
93   Asigneaza pozitiile x si y pentru noduri
94
95   var treeData = treemap(root);
96
97
98   Calculeaza noul layout.
99
100  var nodes = treeData.descendants(),
101      links = treeData.descendants().slice(1);
102
103
104  Normalizeaza pentru adancime fixa.
105
106  nodes.forEach(function(d){ d.y = d.depth * 180});
107
108
109
110  Update noduri ...
111
112  var node = svg.selectAll('g.node')
113      .data(nodes, function(d) {return d.id || (d.id = ++i)
114          ; });
114
115
```

```
116     var nodeEnter = node.enter().append('g')
117         .attr('class', 'node')
118         .attr("transform", function(d) {
119             return "translate(" + source.y0 + "," + source.x0 +
120                 ")";
121         })
122         .on('click', click);
123
124     Adauga cerc pentru fiecare nod.
125
126     nodeEnter.append('circle')
127         .attr('class', 'node')
128         .attr('r', 1e-6)
129         .style("fill", function(d) {
130             return d._children ? "lightsteelblue" : "#fff";
131         });
132
133
134     Adauga eticheta pentru fiecare nod.
135
136     nodeEnter.append('text')
137         .attr("dy", ".35em")
138         .attr("x", function(d) {
139             return d.children || d._children ? -13 : 13;
140         })
141         .attr("text-anchor", function(d) {
142             return d.children || d._children ? "end" : "start";
143         })
144         .text(function(d) { return d.data.name; });
145
146
147     var nodeUpdate = nodeEnter.merge(node);
148
149
150     nodeUpdate.transition()
151         .duration(duration)
152         .attr("transform", function(d) {
153             return "translate(" + d.y + "," + d.x + ")";
154         });
155
156
```

```
157 Update attributele si stilul nodurilor.
158
159 nodeUpdate.select('circle.node')
160   .attr('r', 10)
161   .style("fill", function(d) {
162     return d._children ? "lightsteelblue" : "#fff";
163   })
164   .attr('cursor', 'pointer');
165
166
167 Sterge orice nod care iese.
168
169 var nodeExit = node.exit().transition()
170   .duration(duration)
171   .attr("transform", function(d) {
172     return "translate(" + source.y + "," + source.x +
173       ")";
174   })
175   .remove();
176
177 La iesire se reduce dimensiunea cercului la 0.
178
179 nodeExit.select('circle')
180   .attr('r', 1e-6);
181
182
183 nodeExit.select('text')
184   .style('fill-opacity', 1e-6);
185
186
187 ***** sectiune pentru linkuri
188 *****
189
190 var link = svg.selectAll('path.link')
191   .data(links, function(d) { return d.id; });
192
193 var linkEnter = link.enter().insert('path', "g")
194   .attr("class", "link")
195   .attr('d', function(d){
196     var o = {x: source.x0, y: source.y0}
197     return diagonal(o, o)
```



```
198     });
199
200
201     var linkUpdate = linkEnter.merge(link);
202
203
204     linkUpdate.transition()
205         .duration(duration)
206         .attr('d', function(d){ return diagonal(d, d.parent)
207             });
208
209     var linkExit = link.exit().transition()
210         .duration(duration)
211         .attr('d', function(d) {
212             var o = {x: source.x, y: source.y}
213             return diagonal(o, o)
214         })
215         .remove();
216
217
218     nodes.forEach(function(d){
219         d.x0 = d.x;
220         d.y0 = d.y;
221     });
222
223
224     function diagonal(s, d) {
225         path = `M ${s.y} ${s.x}
226             C ${(s.y + d.y) / 2} ${s.x},
227               ${(s.y + d.y) / 2} ${d.x},
228               ${d.y} ${d.x}`
229         return path
230     }
231
232
233     function click(d) {
234         if (d.children) {
235             d._children = d.children;
236             d.children = null;
237         } else {
238             d.children = d._children;
```

```
239     d._children = null;
240     }
241     update(d);
242     }
243 }
```

4.3.1.1 ALGORITMUL HUFFMAN

Algoritmul Huffman - secvența de date pentru textul: "IT IS BETTER LATER THAN NEVER". Fișierul **arbore.js** conține:

```
1
2 arbore binar "IT IS BETTER LATER THAN NEVER"
3
4 var treeData =
5   {
6     "name": "A0 = 30",
7     "children": [
8       {
9         "name": "Level 1: 18",
10        "children": [
11          { "name": "Level 2: 8",
12            "children": [
13              { "name": "Level 3: 4",
14                "children": [
15                  { "name": "Level 4: 2",
16                    "children": [
17                      { "name": "Level 5: B" },
18                      { "name": "Level 5: ." }
19                    ]
20                  },
21                  { "name": "Level 4: N" }
22                ]
23              },
24              { "name": "Level 3: 4",
25                "children": [
26                  { "name": "Level 4: I" },
27                  { "name": "Level 4: A" }
28                ]
29              }
30            ]
31          },
32          { "name": "Level 2: 10",
```

```

33         "children": [
34             { "name": "Level 3: T"},
35             { "name": "Level 3: E" }
36         ]
37     }
38 ]
39 },
40 { "name": "Level 1: 12",
41   "children": [
42     { "name": "Level 2: 7",
43       "children": [
44         { "name": "Level 3: 4",
45           "children": [
46             { "name": "Level 4: 2",
47               "children": [
48                 { "name": "Level 5: V" },
49                 { "name": "Level 5: S" }
50             ]
51           },
52           { "name": "Level 4: 2",
53             "children": [
54               { "name": "Level 5: L" },
55               { "name": "Level 5: H" }
56             ]
57           }
58         ]
59       },
60       { "name": "Level 3: R" }
61     ]
62   },
63   { "name": "Level 2: spatiu" }
64 ]
65 }
66 ]
67 };

```

4.3.1.2 ALGORITMUL SHANNON-FANO

Algoritmul Shannon-Fano - secvența de date pentru textul: "IT IS BETTER LATER THAN NEVER". Fișierul **arbore.js** conține:

```

1
2 arbore binar "IT IS BETTER LATER THAN NEVER"

```

```
3 var treeData =
4   {
5     "name": "A0",
6     "children": [
7       {
8         "name": "Level 1: AOR",
9         "children": [
10          { "name": "Level 2: AORR",
11            "children": [
12              { "name": "Level 3: AORRR",
13                "children": [
14                  { "name": "Level 4: AORRRR",
15                    "children": [
16                      { "name": "Level 5: AORRRRR = V
17                        " },
18                      { "name": "Level 5: AORRRRL = S
19                        " }
20                    ]
21                  },
22                  { "name": "Level 4: AORRRL",
23                    "children": [
24                      { "name": "Level 5: AORRRRLR = L
25                        " },
26                      { "name": "Level 5: AORRRRL = H
27                        " }
28                    ]
29                  }
30                ]
31              },
32              { "name": "Level 3: AORRL",
33                "children": [
34                  { "name": "Level 4: AORRLR",
35                    "children": [
36                      { "name": "Level 5: AORRLRR = B
37                        " },
38                      { "name": "Level 5: AORRLRL = .
39                        " }
40                    ]
41                  },
42                  { "name": "Level 4: AORRL = N" }
43                ]
44              }
45            ]
46          }
47        ]
48      }
49    ]
50  }
```

```
40         },
41         { "name": "Level 2: AORL",
42           "children": [
43             { "name": "Level 3: AORLR",
44               "children": [
45                 { "name": "Level 4: AORLRR = I" },
46                 { "name": "Level 4: AORLRL = A" }
47               ]
48             },
49             { "name": "Level 3: AORLL = R" }
50           ]
51         }
52       ]
53     },
54     { "name": "Level 1: AOL",
55       "children": [
56         { "name": "Level 2: AOLR",
57           "children": [
58             { "name": "Level 3: AOLRR = T" },
59             { "name": "Level 3: AOLRL = E" }
60           ]
61         },
62         { "name": "Level 2: AOLL = spatiu" }
63       ]
64     }
65   ]
66 };
```

4.4 PROBLEME PROPUSE

1. Să se implementeze algoritmul Huffman de compresie. Să se afișeze arborele de compresie obținut.
2. Să se implementeze algoritmul Shannon-Fano de compresie. Să se afișeze arborele de compresie obținut.
3. Pentru un șir de caractere preluat de la tastatură:
 - Aplicați algoritmul Huffman de compresie;
 - Să se calculeze rata de compresie;
 - Să se afișeze noile coduri obținute pentru fiecare simbol în parte.

4. Pentru un șir de caractere preluat de la tastatură:

- Aplicați algoritmul Shannon-Fano de compresie;
- Să se calculeze rata de compresie;
- Să se afișeze noile coduri obținute pentru fiecare simbol în parte.

BIBLIOGRAFIE

- [1] Adina Aștilean. *Transmisia datelor, notițe curs.*

5 CODURI ÎN BANDA DE BAZĂ

5.1 OBIECTIVE



- Prezentarea generală a modalităților de sincronizare a receptorului cu transmițătorul în cazul transmisiei asincrone și sincrone.
- Prezentarea generală a codurilor de linie fără întoarcere la zero, cu întoarcere la zero și bifazice.

5.2 INTRODUCERE

5.2.1 NOȚIUNI TEORETICE

Transmisia unui semnal se poate realiza direct în banda sa de bază, dacă se introduce pe canal semnalul nemodificat, cu spectrul său original, așa cum este obținut de la traductorul care preia informația din mediul real și o transformă în semnal (senzori pentru diferite mărimi fizice, microfoane pentru semnal audio, camere de luat vederi pentru semnale video). Transmisia în banda de bază nu utilizează modulația pe purtătoare a semnalului util, care ar rezulta în modificarea spectrului de frecvențe inițial, prin translație.

Acest mod de transmisie a informației are avantajul simplității, deoarece semnalele nu sunt prelucrate suplimentar, dar are și o serie de dezavantaje. Banda relativă de frecvență este largă (banda relativă = banda semnalului raportată la frecvența medie) și poate fi ușor influențată de perturbații cu diverse frecvențe. Semnalele nu pot fi transmise în banda de bază la distanțe mari datorită imunității scăzute la perturbații ce apar cu precădere în zona inferioară a spectrului.

Transmisia digitală în banda de bază înseamnă, în general, transmiterea directă pe canal a semnalului dreptunghiular cu două niveluri de tensiune sau curent corespun-

zătoare valorilor logice de “0” și “1”. Deoarece transmisia datelor are loc în mod serial, echipamentul receptor va primi doar două niveluri de tensiune care se succed în timp, din care acesta trebuie să discrimineze începutul și sfârșitul fiecărei celule de bit (sincronizarea pe bit), dar și împachetarea globală a șirului de date în caractere și blocuri de date (sincronizarea pe caracter și pe cadru). Sincronizarea receptorului cu emițătorul poate fi realizată în două moduri, în funcție de relația dintre ceasul transmițătorului și cel al receptorului.

În cazul transmisiei asincrone, cele două ceasuri sunt semnale independente. Fiecare caracter este tratat independent și receptorul va trebui resincronizat la începutul acestuia.

În cazul transmisiei sincrone, ceasul receptorului este în strânsă relație de fază cu cel al transmițătorului. Pe linie se transmite un cadru în mod continuu, fără pauze între caractere, fiind necesară doar păstrarea dependenței ceasurilor la începutul fiecărui caracter. Transmisia asincronă se folosește în general în situațiile când data transferată este generată la intervale aleatoare, linia fiind inactivă perioade lungi și nedeterminate de timp.

Fiecare caracter este încadrat între un bit de “START” și 1, $\frac{1}{2}$, sau 2 biți de “STOP”. Biții de “START” și cei de “STOP” au polaritate diferită, pentru asigurarea a cel puțin unei tranziții $0 \rightarrow 1 \rightarrow 0$ între două caractere succesive care sosesc fără întârziere între ele. Prima tranziție $1 \rightarrow 0$ după perioada de inactivitate a liniei este utilizată de receptor pentru a determina momentele de timp în care se preiau eşantioanele de bit. Se pornește recepția unui caracter pe frontul negativ al bitului de “START”.

Sincronizarea pe bit se realizează prin determinarea cât mai precisă a tranziției de “START”. Ceasul receptorului trebuie să aibă perioada de cel puțin 16 ori mai mică decât durata de transmisie a unui bit, pentru a minimiza eroarea de detecție a începutului de caracter, eroare ce apare datorită asincronismului între tranzițiile de pe linia de recepție și tranzițiile tactului receptorului. Pentru exemplificarea acestei erori, în figura 5.1 s-a considerat o perioadă a ceasului de numai 4 ori mai mică decât durata de transmisie a unui bit.

Sincronizarea pe caracter, adică determinarea începutului și sfârșitului caracterului, se realizează prin simpla numărare a biților veniți pe linie, cu verificarea corectitudinii biților finali de “STOP” care trebuie să aibă valoarea logică “1”. Acest lucru este posibil deoarece numărul de biți pe caracter este stabilit la configurarea transmițătorului și receptorului, reprezentând o caracteristică a transmisiei.

Sincronizarea pe cadru (bloc de caractere) se realizează în mod distinct, în funcție de informația transmisă, adică șir de coduri ASCII (text) sau date binare.

Codurile binare pot fi grupate în patru categorii:

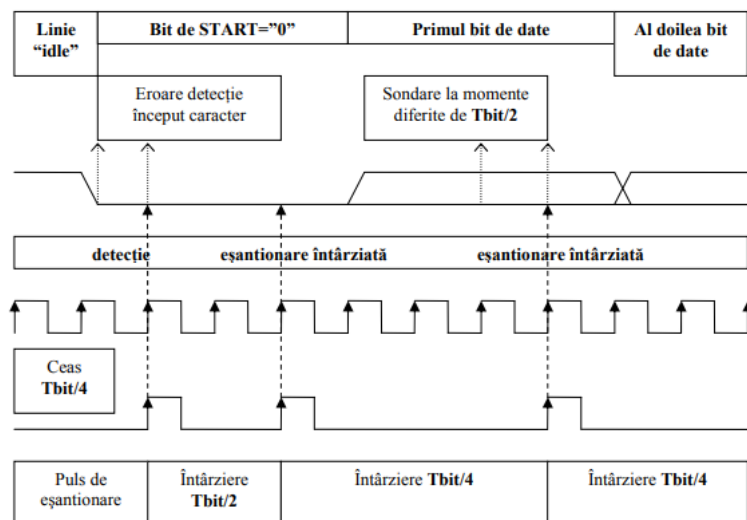


Figure 5.1: Apariția și propagarea erorii de detecție a începutului de caracter, (Sursa: [2])

1. Coduri NRZ ('Non Return to Zero' - fără întoarcere în zero).
2. Coduri RZ ('Return to Zero' - cu întoarcere prin zero).
3. Coduri PE ('Phase Encoded' sau 'Split Phase' - cu codarea fazei).
4. Coduri binare multinivel MLB ('Multi Level Binary').

O altă diferențiere a codurilor se poate face după semnul tensiunilor sau curenților asociați: dacă semnele sunt diferite pentru '0' și '1', codul este bipolar, iar pentru semne de același fel, codul se numește unipolar. Codurile bipolare permit utilizarea ca nivel de referință a potențialului pământului ('ground').

5.2.1.1 CODURI NRZ

Acest tip de codare folosește două niveluri de tensiune diferite. Astfel un „1” logic este reprezentat printr-un nivel pozitiv de tensiune (+V), în timp ce unui „0” îi corespunde fie o tensiune nulă (0V) - în varianta unipolară NRZ, fie o tensiune negativă (-V) dacă ne referim la NRZ bipolar.

Codarea NRZ cunoaște câteva variante:

1. NRZ-L (Level): echivalent cu NRZ (1 - nivel ridicat, 0 – nivel coborât).
2. NRZ-M (Mark): 1- apare o tranziție, 0 – nu apare nici o tranziție
3. NRZ-S (Space): 1 – nu apare nici o tranziție, 0 – apare o tranziție.

Debitul maxim teoretic care poate fi atins într-o transmisie NRZ este egal cu dublul benzii de frecvență ocupată de către semnal (pot fi transmiși 2 biți/Hertz). Principalul dezavantaj al codării de tip NRZ îl constituie lipsa tranzițiilor în cazul unor secvențe lungi de biți identici, ceea ce poate duce la pierderea sincronizării la receptor.

Utilizări: Fast Ethernet (100 Base Fx), FDDI.

5.2.1.2 CODURI RZ

În acest caz se include o informație suplimentară despre ceasul transmisiei, prin întoarcerea la zero a tensiunii după o semiperioadă de ceas (la jumătatea duratei unui bit). Codul de linie RZ-unipolar include pulsuri pozitive de ceas pentru biți de „1”, iar în cazul biților de „0” aceste pulsuri lipsesc complet. La codul de linie RZ-bipolar, pentru codificarea biților de „0” se includ în semnal pulsuri negative de ceas. Semnalul RZ-bipolar are dezavantajul că necesită 2 praguri de decizie pentru discriminarea a 3 niveluri distincte de tensiune, dar recuperarea ceasului de bit la recepție este aproape directă, deoarece atât șirurile lungi de „0” cât și cele de biți „1” au pulsuri de ceas înglobate.

5.2.1.3 CODAREA BIFAZICĂ

Se utilizează trei variante ale acestui tip de codare:

1. Biphase - level - $BI\Phi - L$. Mai este cunoscută și sub denumirea de codare Manchester.
2. Biphase - mark - $BI\Phi - M$. Această codare presupune apariția unei tranziții la începutul oricărui interval de bit. Dacă bitul este de „1”, atunci o a doua tranziție va apare la mijlocul intervalului de bit. Pentru transmisia unui „0” nu se va mai produce nici un fel de tranziție.
3. Biphase - space - $BI\Phi - S$. Este exact inversa codării $BI\Phi - M$ (o tranziție la începutul intervalului de bit, urmată de o altă tranziție la jumătatea acestui interval dacă se transmite „0”, sau fără tranziție dacă se transmite „1”).

Codarea Manchester

Ideea care stă la baza codării Manchester este aceea de a determina o tranziție pentru semnalul emis, tranziție care să apară la mijlocul perioadei de bit. Astfel, un „1” este reprezentat printr-o tranziție de la nivelul $+V$ la nivelul $-V$, în timp ce unei tranziții de la nivelul $-V$ la nivelul $+V$ îi corespunde un „0”. Este evident că în acest fel se asigură sincronizarea între emițător și receptor, chiar și în cazul transmiterii unor secvențe lungi de „0” sau „1”. Mai mult decât atât, întrucât simbolurile binare sunt reprezentate prin

tranziții și nu prin niveluri constante (stări) ca la codarea de tip NRZ, scade drastic probabilitatea apariției unor erori cauzate de mediul de transmisie. Un zgomot care afectează semnalul poate modifica nivelurile transmise, dar este puțin probabil că el va duce la inversarea tranziției sau la lipsa ei, conducând astfel la erori la recepție.

Dezavantajul codării Manchester constă în faptul că, pentru a transmite cu un anumit debit binar, este nevoie de o bandă de frecvențe disponibilă dublă față de cea pe care am utiliza-o în cazul altor tipuri de codare (de exemplu pentru a transmite cu un debit de 10Mbps avem nevoie de o lățime de bandă de 10MHz). Acest inconvenient face codarea Manchester dificil de utilizat pentru debite ridicate.

Utilizări: Ethernet 10Base5, 10Base2, 10BaseT, 10 BaseFL

Codarea Manchester diferențială

La baza codării Manchester diferențiale stă prezența sau absența unei tranziții la începutul intervalului de tact. Astfel, un bit de „1” este reprezentat prin lipsa unei tranziții, în timp ce fiecare bit de „0” este semnatificat prin prezența unei tranziții. Avantajele, respectiv dezavantajele acestui tip de codare sunt în general aceleași ca la codarea Manchester nediferențială.

Utilizări: Rețele de tip Token-Ring.

5.2.1.4 CODURI BINARE MULTINIVEL MLB ('MULTI LEVEL BINARY')

Coduri MLB:

1. Codarea AMI bipolară (AMI-Alternate Mark Inversion).
2. Codarea HDB-3 (High Density Bipolar Order 3).
3. Codarea B8ZS (Bipolar with 8 Zeros Substitution).
4. Codarea 4B/5B NRZI.
5. Codare MLT-3 (MultiLevel Transmission-3).

Codarea AMI bipolară (AMI-Alternate Mark Inversion)

Principiu: zerourile sunt reprezentate printr-un potențial nul (absența semnalului electric pe linie), în timp ce biții de „1” sunt reprezentați alternativ prin tensiuni pozitive (+V), respectiv negative (-V). În acest tip de codare pot exista intervale lungi de lipsă semnal (pentru secvențe lungi de „0”), lucru care poate duce la pierderea sincronizării.

Există și varianta inversată a acestei codări, anume codare pseudoternar, unde lipsa semnalului simbolizează un bit de „1”, iar „0” este reprezentat alternant prin potențiale pozitive și negative. Plecând de la codarea AMI, s-au dezvoltat o serie de tehnici de codare care tind să-l înlocuiască pe acesta în sistemele moderne de transmisii de date.

Utilizări: Transmisia ADSL (Additional Digital Subscriber Loop)

Codarea HDB-3 (High Density Bipolar Order 3)

Se dorește evitarea desincronizărilor ce ar putea apare la secvențe de "0" lungi. Acest inconvenient este combătut astfel: dacă apare un șir de 4 zerouri consecutive, ultimul bit este înlocuit cu o tensiune de aceeași polaritate cu a ultimului bit de "1" introdus. Această măsură ar putea duce însă la apariția unei componente continue semnificative. De exemplu, șirul 100000000, ar putea fi codat astfel : +000+000+. Pentru a evita asemenea situații, fiecare bit modificat trebuie ales de semn schimbat față de precedentul.

Tot pentru a evita introducerea unei componente continue în semnal trebuie respectate regulile:

- dacă numărul de „1” de după ultima secvență modificată este par, atunci un grup de 4 zerouri consecutive se înlocuiește cu secvența „+00+” în cazul în care ultimul nivel nenul de dinaintea acestei secvențe a fost negativ, respectiv cu „-00-” în caz contrar;
- dacă numărul de „1” ce urmează ultimei secvențe modificate este impar, atunci un grup de 4 zerouri consecutive se înlocuiește cu secvența „000+” în cazul în care ultimul nivel nenul de dinaintea acestei secvențe a fost pozitiv, respectiv cu „000-” în caz contrar.

Utilizări: Standardele E1, E3

Codarea B8ZS (Bipolar with 8 Zeros Substitution)

Are la bază codul AMI, se înlocuiesc secvențele de 8 zerouri consecutive, cu secvențe în care să apară tranziții pentru a se evita astfel pierderea sincronismului.

Astfel:

- dacă impulsul anterior acestei secvențe de „0” este de nivel pozitiv, atunci codul corespunzător este 000+-0-+;
- dacă impulsul anterior acestei secvențe de 8 zerouri este de nivel negativ, atunci codul corespunzător este 000-+0+-;

Utilizarea acestui tip de codare va produce două modificări ale alternanței "+-“, situație care este improbabil să fie cauzată de către un zgomot.

Utilizări: Standardul T1 (transmisie rapidă de voce, date pe fire torsadate sau cablu coaxial).

Codarea 4B/5B NRZI

Aceast cod este o combinație de 2 algoritmi de codare. Pentru a înțelege semnificația acestei alegeri, să considerăm pentru început alternativa simplă a schemei NRZ. Cu

NRZ, o stare a semnalului reprezintă „1” binar, și o altă stare reprezintă „0” binar. Dezavantajul acestei abordări îl reprezintă absența sincronizării. În schema de codare 4B/5B, codarea se face cu 4 biți odată. Fiecare 4 biți de semnal sunt codati într-un cuvânt de 5 biți de cod. Eficiența acestei codări este de 80%: pentru a transmite date cu un debit de 100Mbps este necesară o lățime de bandă disponibilă de 125MHz. De remarcat că modul de transmisie al biților cuvântului de cod nu se specifică. Se utilizează de obicei codarea NRZI (ajungându-se la un necesar de bandă de 62.5 MHz) sau codarea MLT-3 (lățime de bandă necesară de 31.25MHz).

Deoarece pentru a coda cele 16 combinații posibile de câte 4 biți utilizăm doar 16 din cele 32 de combinații posibile de câte 5 biți, rămân alte 16 grupuri care nu sunt utilizate. Cuvintele de cod sunt astfel alese încât să nu existe mai mult de două zerouri succesive și mai puțin de două tranziții (doi de „1” - în codaj NRZI) în cuvântul de cod de 5 biți. Pentru codare se folosește următorul tabel:

Date	Cod	Date	Cod	Simboluri speciale	Cod
0000	11110	1000	10010	Q (Quiet)	00000
0001	01001	1001	10011	I (Idle)	11111
0010	10100	1010	10110	H (Halt)	00100
0011	10101	1011	10111	J (Start delimiter)	11000
0100	01010	1100	11010	K (Start delimiter)	10001
0101	01011	1101	11011	T (End delimiter)	01101
0110	01110	1110	11100	S (Set)	11001
0111	01111	1111	11101	R (Reset)	00111

Figure 5.2: Codare 4B/5B

Să considerăm de exemplu secvența binară de intrare 10000101111. Ea este împărțită în grupuri de câte 4 biți, care se codează conform tabelii de codare:

Grupurile de cod care rămân nefolosite sunt fie declarate invalide, fie capătă rolul de simboluri de control a transmisiei, așa cum se observă în tabelul de codare.

Utilizări: FDDI, 100Base-X

Codare MLT-3 (MultiLevel Transmission-3)

Cu toate că 4B/5B-NRZI este eficient pe fibră optică, el nu poate fi folosit și în transmisiile pe fire torsadate. Cablul UTP acționează asupra semnalului transmis asemănător cu un filtru trece-jos, atenuând componentele de înaltă frecvență și distorsionând astfel puternic secvența de date.

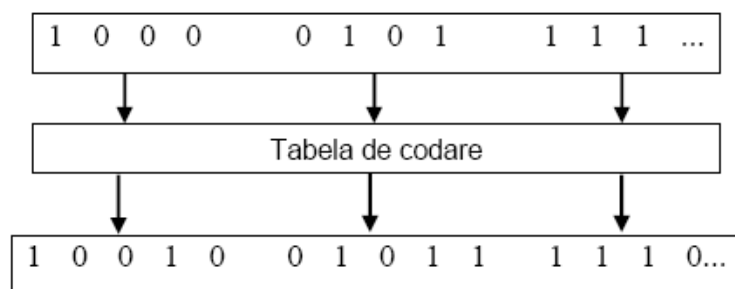


Figure 5.3: Exemplu de codare 4B/5B

Principiul codării MLT este următorul:

- Biții de „1” produc o schimbare de stare a semnalului transmis în linie, în timp ce biții de „0” lasă nemodificată starea corespunzătoare bitului anterior.
- Biții de „1” sunt codati succesiv prin 3 stări: +V, 0, -V.
- Marele avantaj al acestui tip de codare este reducerea semnificativă a benzii necesare pentru un debit cerut, grație folosirii celor 3 stări: pentru un debit de 100Mbps lățimea de bandă necesară este de doar 25MHz.

Utilizări: Fast Ethernet 100BaseTx, 100BaseT4

5.2.2 APLICAȚII

Lucrarea constă în reprezentarea unui șir de biți folosind librării js și pachete.

Resurse folosite:

- librăria axios
- librăria vue
- npm - administratorul de pachete pentru Node.js

Pentru a instala dependențele externe se execută comanda: **npm install ...** în linia de comandă.

Pentru a instala serverul http: **npm install -g http-server** în linia de comandă.

Pentru a porni serverul: **http-server &** în linia de comandă.

Fișierul **index.html** conține:

```
1 | <html>
2 | <head>
```

```

3     <meta content="text/html; charset=utf-8" http-equiv="
      Content-Type">
4     <title>Baseband Encoder</title>
5     <link rel="stylesheet" type="text/css" href="css/style.
      css">
6 </head>
7 <body>
8     <div id="baseband-encoder">
9         <ul v-for="(bit, index) in bits">
10             <li>
11                 <input
12                     v-on:change="encode()"
13                     maxlength="1" :placeholder="'B'+index"
14                     v-model="bit.value"
15                     v-bind:class="[validateBit(bit.value) ?
16                         'valid-input' : 'invalid-input']"
17                     />
18             </li>
19         </ul>
20         <p>Bitstream data model: <code>{{bits}}</code></p>
21         <p>Baseband code representation: <code class="
22             encoded">{{encodedBits.join('')}}</code></p>
23     </div>
24     <script type="text/javascript" src="libs/vue.js"></
25     script>
26     <script type="text/javascript" src="libs/axios.js"></
27     script>
28     <script type="text/javascript" src="js/helpers.js"></
29     script>
30     <script type="text/javascript" src="js/baseband-codes.
31     js"></script>
32     <script type="text/javascript" src="js/app.js"></
33     script>
34 </body>
35 </html>

```

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: 0px;
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: 2px solid green;
29     -webkit-transition: all 1000ms linear;
30     -ms-transition: all 1000ms linear;
31     transition: all 1000ms linear;
32 }
33
34 .valid-input {
35     border: 2px 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 }
52
53 .encoded{
54   letter-spacing: -6px;
55   font-size: 30px;
56   font-weight: bold;
57 }
```

Fișierul **app.js** conține:

```
1  var app = new Vue({
2    el: '#baseband-encoder',
3    data: {
4      bits: [],
5      encodedBits: [],
6      status: '',
7      numberOfBits: 8,
8      validateBit: validateBit
9    },
10   created: function () {
11     this.bits = getBitstream(this.numberOfBits);
12   },
13   methods: {
14     encode: function(){
15       this.encodedBits = getManchesterLevelEncoding(
16         this.bits);
17     }
18   })
```

Fișierul **helpers.js** conține:

```
1  var validate = function (bits) {
2    for (var i = 0; i < bits.length; i++) {
3      if (this.validateBit(bits[i].value) ===
4        false)
5        return false;
6    }
7    return true;
8  }
9  var validateBit = function (character) {
10   if (character === null) return false;
```

```
11         return (parseInt(character) === 0 ||
12                 parseInt(character) === 1);
13     }
14
15     function getBitstream(n) {
16         result = [];
17         for (var i = 0; i < n; i++) {
18             let bit = { value: null };
19             result.push(bit);
20         }
21         return result;
22     }
```

Codul Manchester este reprezentat în **baseband-codes.js**:

```
1 function getManchesterLevelEncoding(bits) {
2     var result = [];
3     for (var i = 0; i < bits.length; i++) {
4         let symbol = '|--|--|';
5         if (parseInt(bits[i].value) == 1) symbol = '|__|--|';
6         if (parseInt(bits[i].value) == 1 && i > 0 &&
7             parseInt(bits[i - 1].value) == 1) symbol = '|__
8             |--|';
9         if (parseInt(bits[i].value) == 0) symbol = '|--|__|';
10        if (parseInt(bits[i].value) == 0 && i > 0 &&
11            parseInt(bits[i - 1].value) == 0) symbol = '|--|
12            __|';
13        result.push(symbol);
14    }
15    return result;
16 }
```

5.3 DESFĂȘURAREA LUCRĂRII

1. Considerând șirul de biți următor: 1100100001000 să se codeze folosind toate tehnicile prezentate anterior.
2. Pentru șirul de biți următor: 1000010000110000 să se reprezinte șirul de biți rezultat în urma codării folosind tehnica AMI.

3. Pentru șirul de biți următor: 1001010110100100 să se reprezinte șirul de biți rezultat în urma codării folosind tehnica MLT.
4. Pentru șirul de biți următor: 0000110000100001 să se reprezinte șirul de biți rezultat în urma codării folosind tehnica 4B/5B NRZI.

BIBLIOGRAFIE

- [1] Adina Aștilean. *Transmisia datelor, notițe curs.*
- [2] Ioan, Aleodor Daniel. *Transmisia datelor: considerații teoretice și experimente de laborator.*

6 COMUNICAȚII PRIN SATELIT. APLICAȚII ALE COMUNICAȚIILOR PRIN SATELIT - GPS

6.1 OBIECTIVE



- Prezentarea generală a comunicațiilor prin satelit.
- Prezentarea generală a sistemului GPS (Global Positioning System).
- Familiarizarea cu aplicații Google de localizare.

6.2 INTRODUCERE

Comunicația prin satelit presupune comunicația dintre două stații folosind un satelit [12]. În figura 6.1 este prezentat mecanismul comunicației prin satelit.

Semnalele primite de la stația de emisie este procesată de către satelit cu ajutorul unui modul numit transponder (transmitter + responder). Un satelit are la dispoziție mai multe transpondere, fiecare lucrând pe o anumită frecvență. Un transponder poate funcționa și în regim de semitransponder (half – transponder). Semnalul transmis către stațiile de recepție (antenele) de pe pământ poartă numele de downlink. Semnalul transmis de la stațiile terestre către satelit poartă numele de uplink. De obicei, frecvențele purtătoare ale celor 2 fascicule – uplink și downlink - sunt diferite. Suprafața terestră asupra căreia radiază antenele de emisie ale satelitului se numește “zonă de acoperire” sau footprint. Zonele de acoperire sunt reprezentate pe hărți prin curbe de nivel constant, indicând puterea (EIRP) sau densitatea ei la suprafața solului (PFD).

Clasificarea generală a sateliților geostaționari:

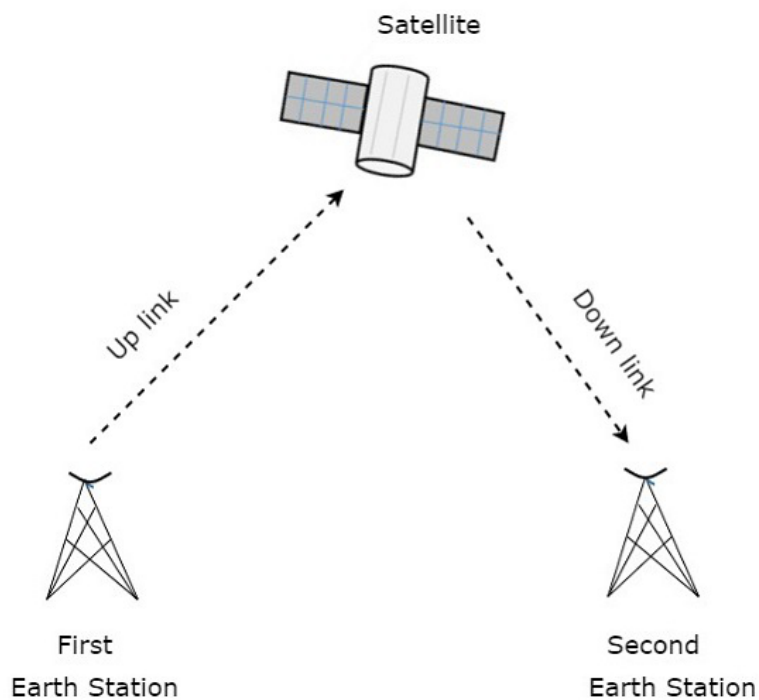


Figure 6.1: Mecanismul comunicației prin satelit, (Sursa: [12])

- Sateliți de telecomunicații;
- Sateliți de radiodifuziune;
- Sateliți meteorologici;
- Sateliți științifici;
- Sateliți militari.

Avantajele și dezavantajele comunicației prin satelit.



- + Zonă de acoperire mare.
- + Lățime de bandă mare.
- + Posibilitatea de a transmite mesaje broadcast.
- Cost mare (lansarea pe orbită și întreținerea sateliților; depanare).

Comunicațiile prin satelit se pot clasifica funcție de tipul conexiunii:

- conexiune "one-way" de comunicație cu satelitul, [6.2](#).
Comunicația este inițiată de către prima stație (transmitter) și semnalul este recepționat de către cea de a doua stație (receiver). Aplicații ale comunicațiilor prin satelit one-way:
 - Radio, TV.
 - Internet.
 - Servicii de telemetrie, tracking și comandă.
 - Servicii de localizare.
- conexiune "two-way" de comunicație cu satelitul. refimg-satelit-3.
În cazul comunicației two-way informația este schimbată între oricare două stații de pe pământ prin intermediul satelitelui. Aplicații ale comunicațiilor prin satelit two-way:
 - Telefonie fixă, fax.
 - Internet.
 - Telefonie mobilă, comunicații maritime și aeriene.

6.2.1 GPS - SISTEMUL DE POZIȚIONARE GLOBALĂ

Sistemul de poziționare globală (GPS - Global Positioning System) este un sistem global de radio-navigare cu ajutorul căruia se poate determina locația oricărui punct de pe glob și timpul exact.

Sistemul GPS cunoscut și ca NAVSTAR (Navigation Satellite Timing and Ranging), a fost propus de către Departamentul american al apărării încă de la începutul anilor 1970. Acesta a fost destinat aplicațiilor militare și a avut drept scop central furnizarea

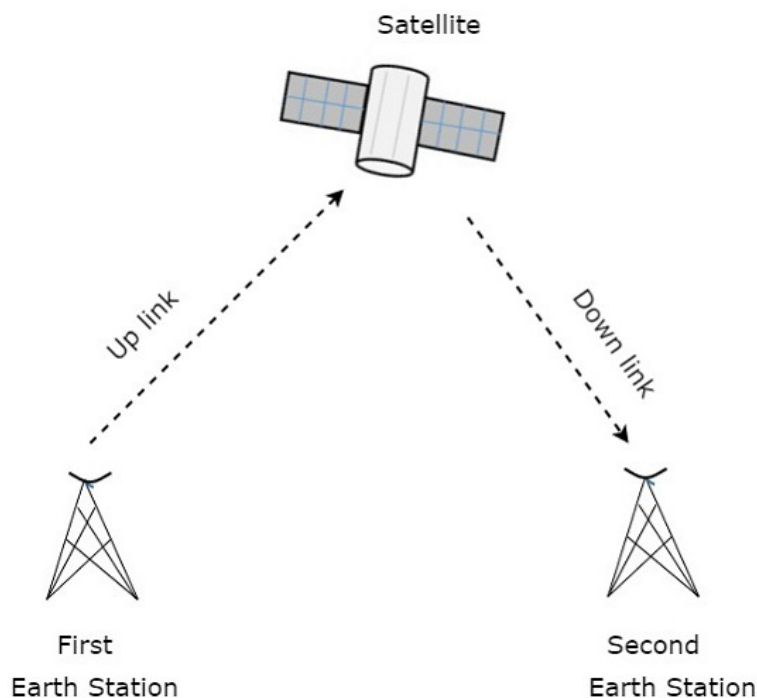


Figure 6.2: Comunicației prin satelit - conexiune one-way, (Sursa: [12])

poziției unei persoane aflate în orice punct geografic de pe suprafața Terrei, în orice condiții meteo și la orice moment de timp.

La sfârșitul anului 1993 sistemul GPS a devenit complet operațional, punând la dispoziția utilizatorilor o constelație formată din 24 de sateliți. Aceștia sunt amplasați pe orbite aflate la o înălțime medie de 20180 km deasupra Pământului și se deplasează cu o viteză de aproximativ 3.9 km/s [7].

Tehnologia GPS este compusă din trei componente [2], [3] [4] [5] [6] [7], [8], [9], [10], [11]:

- *Segmentul spațial* format din 24 de sateliți.
- *Segmentul de control* format din 5 stații aflate pe Pământ care gestionează buna funcționare a sateliților.
- *Segmentul utilizatorilor* (civili sau din departamentul militar).

Din motive strategice, Segmentul de Control a fost responsabil cu bruiatul semnalelor GPS (politica DS - Disponibilitate Selectivă) pentru a scădea precizia sistemelor de poziționare folosite în scopuri civile. În anul 2000 restricțiile DS au fost ridicate iar

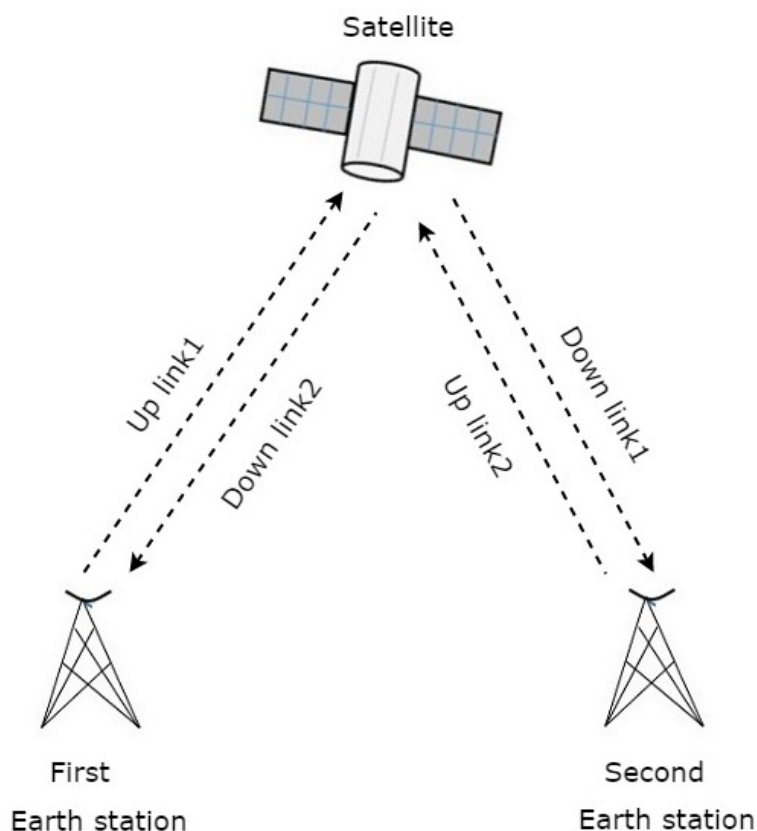


Figure 6.3: Comunicații prin satelit - conexiune two-way, (Sursa: [12])

precizia poziționării a ajuns pentru prima dată sub limita impusă de 100 m [9].

Pe lângă Sistemul GPS mai există și alte sisteme de navigație printre care GLONASS (Rusia), GALILEO (Europa) și Beidou/Compass (China).

În prezent GPS-ul pune la dispoziție 32 de sateliți iar precizia poziționării este de ordinul metrilor în sistemele GPS standard și poate ajunge la câțiva milimetri în cele mai noi versiuni de GPS diferențial.

6.2.2 DETERMINAREA POZIȚIEI PRIN TRIANGULAȚIE

Triangulația este o metodă de calcul a unei poziții de pe Pământ utilizând trei sateliți ca și puncte de referință (Figura 6.4). Aceasta se bazează pe estimarea cât mai precisă a distanței dintre receptorul GPS și cei trei sateliți. Pentru a putea calcula o locație de

pe glob, mai este necesară cunoașterea coordonatelor exacte ale celor trei puncte de referință din spațiu.

Distanța dintre un satelit și un receptor de pe Pământ (d) se calculează cu formula:

$$d = \Delta_t \cdot c \quad (6.1)$$

Unde Δ_t este întârzierea semnalului recepționat de la satelit iar c este viteza luminii.

Sateliții sunt echipați cu ceasuri atomice care măsoară foarte precis timpul. Totuși există erori care apar datorită ceasurilor din dispozitivele receptoare. O nesincronizare de $1 \mu s$ duce la o eroare de poziționare de $300 m$.

Există mai multe surse de eroare care pot afecta acuratețea poziționării:

- Erori cauzate de propagarea semnalului pe căi multiple.
- Precizia ceasurilor atomice de pe sateliți și nesincronizările ceasului din receptorul GPS.
- Numărul de sateliți vizibili și poziționarea acestora în spațiu.
- Efectele ionosferei și troposferei asupra semnalului GPS.
- Precizia datelor de tip ephemeris (pozițiile sateliților în spațiu).

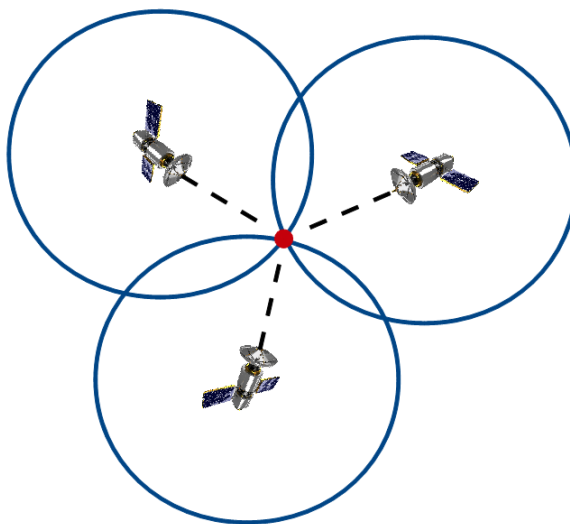


Figure 6.4: Triangulația

6.2.3 FORMATUL MESAJELOR GPS

Fiecare satelit transmite în mod continuu spre Pământ un mesaj de navigație, cu viteza de 50 b/s, care cuprinde timpul sistemului și valorile de corecție ale ceasului, poziția sa cu precizie ridicată (*ephemeris*), pozițiile aproximative ale celorlalți sateliți (*almanac*) și starea sistemului. Mesajul de navigație este necesar pentru determinarea întârzierii semnalului recepționat și a poziției exacte a unui satelit în spațiu. Un întreg mesaj de navigație este compus din 25 de cadre (pagini) independente cu lungimea de 1500 de biți fiecare. Acestea la rândul lor sunt divizate în 5 sub-cadre de 300 de biți (Figura 6.5).

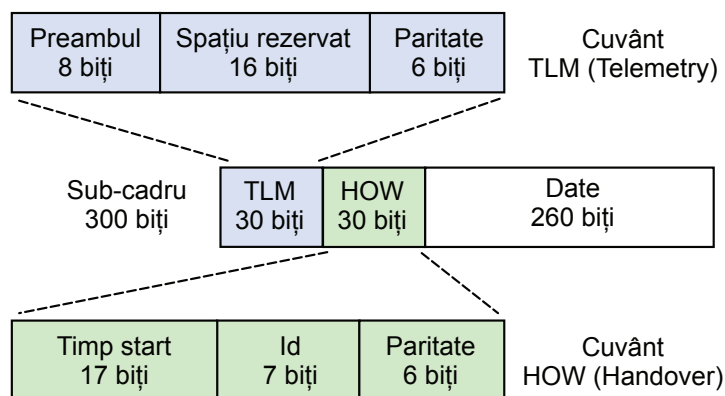


Figure 6.5: Structura unui sub-cadru

Fiecare sub-cadru este compus dintr-un antet (TLM, HOW) care are un format comun pentru toate sub-cadrele transmise. Acest antet este utilizat de către receptoarele GPS la identificarea sub-cadrelor și de asemenea la verificarea parității datelor recepționate.

Primul cuvânt al fiecărui sub-cadru, cunoscut ca și TLM, este compus dintr-un preambul de 8 biți (10001011) utilizat pentru sincronizarea cu receptorul GPS, urmat de un segment de 16 biți rezervat utilizatorilor autorizați. Fiecare cuvânt dintr-un sub-cadru se termină cu 6 biți de paritate utilizați pentru verificarea datelor recepționate. În cazul în care biții de paritate nu corespund datelor din cuvântul TLM, receptorul va trece la următorul preambul.

Cuvântul HOW începe cu 17 biți care reprezintă timpul de start pentru următorul sub-cadru (sincronizarea receptorului cu satelitul). Acest timp începe cu valoarea 0 care corespunde cu începutul săptămânii GPS (Duminică 00:00:00) și este incrementat cu o unitate la fiecare 6 secunde (durata de transmisie a unui sub-cadru). Următorii 7 biți conțin identificatorul sub-cadrelui curent.

Pe lângă antet, sub-cadrele conțin și o secțiune de date. În Tabelul 6.1 este specificată

informația transportată în cele 5 sub-cadre.

Table 6.1: Informații conținute în sub-cadre

Sub-cadru	Date
1	Timpul satelitului și starea satelitului. Estimarea preciziei de localizare a satelitului. 10 biți pentru numărul săptămânii GPS (0-1023).
2,3	Informații precise despre orbita satelitului transmițător (<i>ephemeris</i>).
4,5	Date de tip <i>almanac</i> pentru ceilalți sateliți, starea de funcționare a acestora și diferența dintre timpul GPS și UTC.

Sateliții din sistemul GPS transmit semnale purtătoare de informație pe două frecvențe principale:

- Frecvența L1 (1575,42 MHz) - poartă mesajul de navigație și semnalele de cod SPS (Serviciul de Poziționare Standard).
- Frecvența L2 (1227,60 MHz) - este folosită pentru măsurarea întârzierilor provocate de ionosferă.

6.3 DESFĂȘURAREA LUCRĂRII

6.3.1 AFIȘAREA COORDONATELOR GPS ÎN BROWSER

Aplicația cere permisiunea utilizatorului să afișeze coordonatele GPS. Pentru o localizare precisă este necesar un dispozitiv cu senzor GPS.

Fișierul **index.html** conține:

```

1 <!DOCTYPE html>
2 <html>
3 <body>
4 <p>Click the button to get your coordinates.</p>
5 <button onclick="getLocation()">Try It</button>
6 <p id="demo"></p>
7 <script src="app-gps.js"></script>
8 </body>
9 </html>
```

Fișierul **app-gps.js** conține:

```
1 var x = document.getElementById("demo");
2 function getLocation() {
3     if (navigator.geolocation) {
4         navigator.geolocation.getCurrentPosition(showPosition);
5     } else {
6         x.innerHTML = "Geolocation is not supported by this
7             browser.";
8     }
9 }
10 function showPosition(position) {
11     x.innerHTML = "Latitude: " + position.coords.latitude +
12     "<br>Longitude: " + position.coords.longitude;
```

6.3.2 AFIȘAREA LOCAȚIEI GPS

Sunt mai multe companii producătoare de hărți, fiecare impunând mai multe condiții de utilizare. În continuare sunt exemplificate utilizarea hărților din Google Maps și TomTom.

6.3.2.1 ÎN GOOGLE MAPS

Pentru folosirea Google Maps este necesară obținerea de un API key în prealabil. Pentru obținerea API key se parcurg următorii pași:

1. Se accesează [Google Cloud Platform Console](#) (e necesar un cont google). Pentru fiecare user sunt permise 12 proiecte gratuite.
2. Din meniul dropdown, se poate alege un proiect deja existent sau se poate crea un nou proiect prin apăsarea butonului "create project" din partea dreaptă a ferestrei de lucru.
3. Din meniu se accesează: APIs & Services > Credentials.
4. Din meniul Credentials se accesează Create credentials > API key.
Se deschide o fereastră de dialog în care se afișează noile credențiale.
5. se dă click pe Close.
noua cheie API key este afișată în pagina **Credentials** la secțiunea **API keys**.
6. API key astfel obținut este adăugat în request:

```
1 <script async defer src="https://maps.googleapis.
    com/maps/api/js?key=YOUR_API_KEY&callback=
```

```
2      initMap"  
    type="text/javascript"></script>
```

Fișierul **index.html** conține:

```
1 <!DOCTYPE html>  
2 <html>  
3   <head>  
4     <link rel="stylesheet" href="style.css">  
5   </head>  
6   <body>  
7  
8     <div id="map"></div>  
9     <!-- Replace the value of the key parameter with your own  
10        API key. -->  
11     <script async defer  
12       src="https://maps.googleapis.com/maps/api/js?  
13         key=YOUR_API_KEY&callback=initMap">  
14     </script>  
15     <script src="app-gps.js"></script>  
16   </body>  
17 </html>
```

Fișierul **app-gps.js** conține:

```
1 var map, infoWindow;  
2 function initMap() {  
3   map = new google.maps.Map(document.getElementById('map'),  
4     {  
5       center: {lat: -34.397, lng: 150.644},  
6       zoom: 6  
7     });  
8   infoWindow = new google.maps.InfoWindow;  
9  
10  if (navigator.geolocation) {  
11    navigator.geolocation.getCurrentPosition(function(  
12      position) {  
13      var pos = {  
14        lat: position.coords.latitude,  
15        lng: position.coords.longitude  
16      };  
17      infoWindow.setPosition(pos);
```

```
18     infoWindow.setContent('Location found.');
```

```
19     infoWindow.open(map);
```

```
20     map.setCenter(pos);
```

```
21 }, function() {
```

```
22     handleLocationError(true, infoWindow, map.getCenter()
```

```
23     );
```

```
24 } else {
```

```
25
```

```
26     handleLocationError(false, infoWindow, map.getCenter())
```

```
27     ;
```

```
28 }
```

```
29
```

```
30 function handleLocationError(browserHasGeolocation,
```

```
31     infoWindow, pos) {
```

```
32     infoWindow.setPosition(pos);
```

```
33     infoWindow.setContent(browserHasGeolocation ?
```

```
34         'Error: The Geolocation service
```

```
35         failed.' :
```

```
36         'Error: Your browser doesn\'t
```

```
37         support geolocation.');
```

```
38     infoWindow.open(map);
```

```
39 }
```

Fișierul **style.css** conține:

```
1
```

```
2
```

```
3 #map {
```

```
4     height: 100%;
```

```
5 }
```

```
6
```

```
7 html, body {
```

```
8     height: 100%;
```

```
9     margin: 0;
```

```
10    padding: 0;
```

```
11 }
```


6.3.2.2 TOMTOM MAPS

Pentru a utiliza hărțile TomTom este necesară procurarea unei API Key. În continuare sunt prezentați pașii necesari pentru obținerea unei API Key:

1. Se accesează contul de dezvoltare [TomTom](#).
2. Din meniul **My Dashboard**, utilizatorul poate vizualiza cheile existente și poate crea altele noi. Pentru fiecare API Key pot fi incluse mai multe produse.
3. Din meniu utilizatorul va selecta: APIs & Services > Credentials.
4. Pentru a instala biblioteca Maps SDK pentru Web - Maps, se introduce următoarea comandă din command prompt:

npm i @tomtom-international/web-sdk-maps

În continuare este prezentat un exemplu, având la bază tutorialul TomTom, de afișare a unei hărți [13]. Acest program folosește http-server. Instalarea http-server se face introducând următoarea comandă din command prompt.

npm install -global http-server

Pornirea serverului http-server:

http-server

Fișierul **my-map.html**:

```
1 <!DOCTYPE html>
2 <html class='use-all-space'>
3 <head>
4     <meta http-equiv='X-UA-Compatible' content='IE=Edge' />
5     <meta charset='UTF-8'>
6     <title>My Map</title>
7     <meta name='viewport'
8         content='width=device-width,initial-scale=1,
9             maximum-scale=1,user-scalable=no' />
10    <!-- Replace version in the URL with desired library
11         version -->
12    <link rel='stylesheet' type='text/css' href='https://
13        api.tomtom.com/maps-sdk-for-web/cdn/5.x/<version>/
14        maps/maps.css' />
15    <style>
16        #map {
17            width: 100vw;
18            height: 100vh;
19        }
```

```
16     </style>
17 </head>
18 <body>
19     <div id='map' class='map'></div>
20     <!-- Replace version in the URL with desired library
        version -->
21     <script src='https://api.tomtom.com/maps-sdk-for-web/
        cdn/5.x/<version>/maps/maps-web.min.js'>
22     <script>
23         tt.setProductInfo('Your Product Name', '<version>');
24         tt.map({
25             key: 'Your API Key',
26             container: 'map',
27             style: 'tomtom://vector/1/basic-main '
28         });
29     </script>
30 </body>
31 </html>
```

Următorii parametrii se vor personaliza:

- **your-product-name:** Numele produsului software;
- **your-product-version:** Versiunea produsului software;
- **your-tomtom-API-Key:** TomTom API Key generată anterior;
- **version:** Versiunea de bibliotecă software - toate versiunile sunt disponibile în secțiunea download de pe site-ul producătorului.

6.4 APLICAȚII ȘI PROBLEME

BIBLIOGRAFIE

- [1] Adina Aşticlean. *Transmisia datelor, notițe curs*.
- [2] Elliott D. Kaplan and Christopher Hegarty. *Understanding GPS: Principles and Applications*. Artech House Publishers, 2 edition, 2005. ISBN 1580538940.
- [3] Tracy M. L. Brown, Steven A. McCabe, and Charles Wellford. *Global Positioning System (GPS) Technology for Community Supervision: Lessons Learned*. NOBLIS Technical Report, 2007.
- [4] Robert Grisso, Mark Alley, and Conrad Heatwole. *Precision Farming Tools: Global Positioning System (GPS)*. Virginia Polytech, 2009.
- [5] Domenico A Maisano, Jafar Jamshidi, Fiorenzo Franceschini, Paul Maropoulos, Luca Mastrogiacomo, Antony Mileham, and Geraint Owen. *Indoor GPS: System functionality and initial performance evaluation*. International Journal of Manufacturing Research, 3(3):335-349, 2008.
- [6] Elvis Ngah. *GPS Technology - Optimizing Car Navigation.*, 2006.
- [7] Frederic Snider. *GPS: Theory, Practice and Applications*. PDH Online Course, 2012.
- [8] Jean-Marie Zogg. *GPS Basics Introduction to the system - Application overview*. u-blox Online Publication, 2002.
- [9] Jean-Marie Zogg. *GPS Essentials of Satellite Navigation - Compendium*. u-blox Online Publication, 2009.
- [10] Trimble. *GPS Tutorial*. Online: http://www.trimble.com/gps_tutorial.
- [11] Norman Bonnor. *A Brief History of Global Navigation Satellite Systems*. Journal of Navigation - Cambridge University Press, 65:1-14, 2012.
- [12] <https://www.tutorialspoint.com>
- [13] <https://developer.tomtom.com/maps-sdk-web-js/functional-examples>