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

LISTĂ DE FIGURI

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 librariilor necesare pentru dezvoltarea aplicațiilor pe aceste trei limbaje.

1.2.2 HTML

Acronimul HTML vine de la "Hyper Text Markup Language" si reprezintă un limbaj care ne permite sa definim interfete grafice pentru aplicații software care sunt executate în

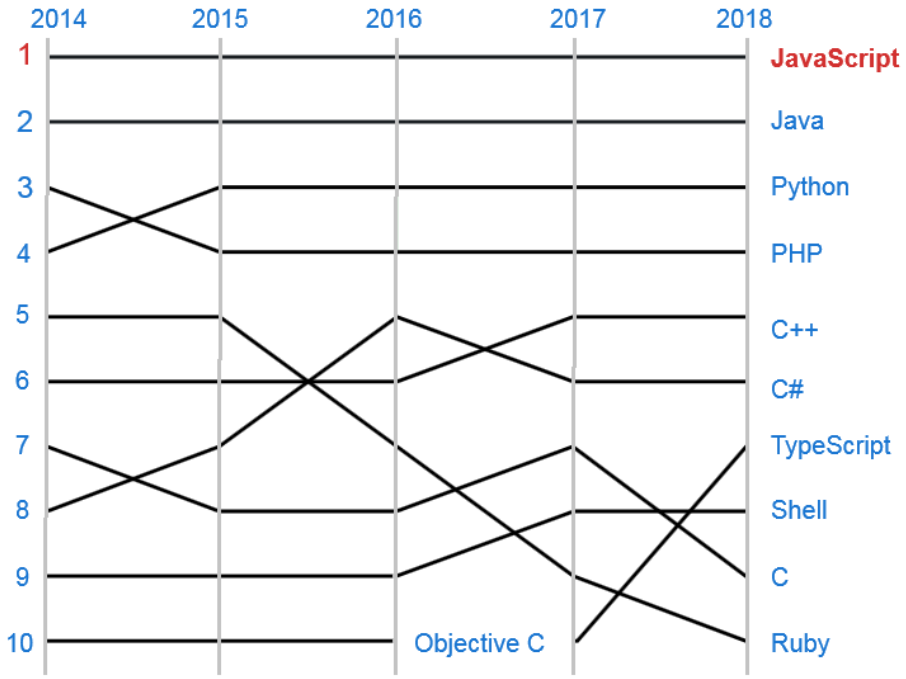

Figure 1.1: Limbajele de programare cele mai populare pe Github în 2018 (Sursa: [4])

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: , <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, hârtie).

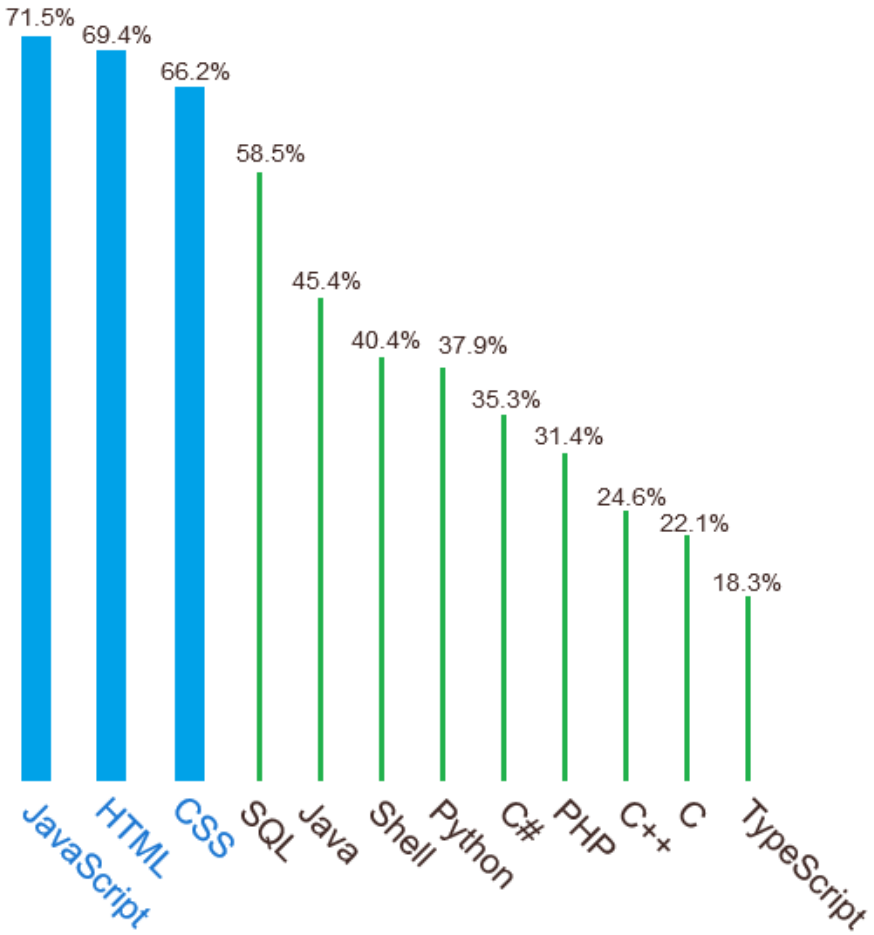

Figure 1.2: Limbajele de programare cele mai populare pe StackOverflow în 2018 (Sursa: [5])

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 librarii 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 referințâ 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 " ".

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 accepted");
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 })();
```

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",
       increment);
9
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 librarie scrisă în JavaScript care simplifică și oferă o sintaxa 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/jquery
          /3.3.1/jquery.min.js"></script>
3 </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 libraria **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:

- 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 librariei 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 returneaza "not allowed".
- Funcția apelată cu n<1 sau n>10 returneaza "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 aplicaţii client-server.

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

BIBLIOGRAFIE

[1] Adina Aştilean. *Transmisia datelor, notiţe 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

2 SISTEME DE COMUNICAŢII CU FIR

2.1 OBIECTIVE

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

2.2 INTRODUCERE

2.2.1 MOTIVAŢIE

2.2.2 MEDII DE TRANSMISIE CU FIR

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

- viteza de transmisie - raportul dintre cantitatea de informaţie transmisă şi timpul necesar transmisiei;
- lăţimea de bandă - cantitatea de informaţie pe care o poate transmite.
- distanţa pe care se face transmisia - distanţa dintre emiţător şi receptor pe care se face transmisia de date fară 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 informaţia este transmisă şi o plasă de sârmă cu rol izolator la interferenţe, separate printrul manşon de plastic. Informaţia transmisă este analogică sau digitală. 2.1

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

Figure 2.1: Cablul coaxial

- RG-58 /U - utilizat pentru semnale low power si conexiuni RF, are rezistenţa de 50 Ohm şi impedanţa de 25 pF/ft (82 pF/m);
- RG-59 /U - utilizat pentru semnale video low power si conexiuni RF, are rezistenţa de 75 Ohm şi impedanţa de 20pF/ft (60pF/m); Cel mai des utilizat pentru distribuţie CC TV.
- RG-62 /U - utilizat în aplicaţii industriale de interior pentru a transmite semnale de frecvenţe înalte şi de putere, are rezistenţa de 93 Ohm;
- RG-6/U - utilizat în aplicaţii rezidenţiale, are rezistenţa de 75 Ohm; Cel mai des utilizat pentru distribuţie TV.
- RG-11/U - utilizat în aplicaţii industriale de interior pentru a transmite semnale de frecvenţe înalte şi de putere, are rezistenţa de 75 Ohm. Cel mai des utilizat pentru distribuţie HD TV (High Definition TV).

Figure 2.2: Cablul coaxial

Conectori, 2.3, 2.4
Avantaje şi dezavantaje:

+ ieftin şi durabil;
+ uşor de folosit;
+ rezistenţă bună la EMI;
+ viteze de transfer până la 10 Mbps;
- mentenanţa - un cablu defect, întreaga reţea este inactivă.

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

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 şi se utilizează în comunicaţii. Torsadarea este o protecţie electromagnetică, pentru reducerea sau anularea semnalelor parazite produse de alte circuite electrice învecinate şi se obţine prin răsucirea a două cabluri 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 curenţii care le parcurg vor avea sensuri opuse. Aceştia vor genera două câmpuri electromagnetice în antifază care se vor anula reciproc.

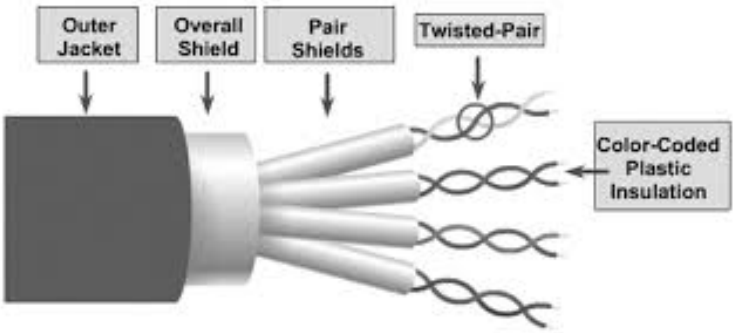

Figure 2.5: Cablul torsadat, 4 perechi

Tipuri de cabluri torsadate (cele mai răspândite):

- FTP - Cablu cu perechi rasucite în folie - Foiled Twisted Pair;
- UTP - Cablu bifilar torsadat neecranat - Unshielded Twisted Pair, 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, 2.7.

Categorii de cabluri torsadate (Electronic Industries Association (EIA) a stabilit standard-ele UTP in 1995, [8]:

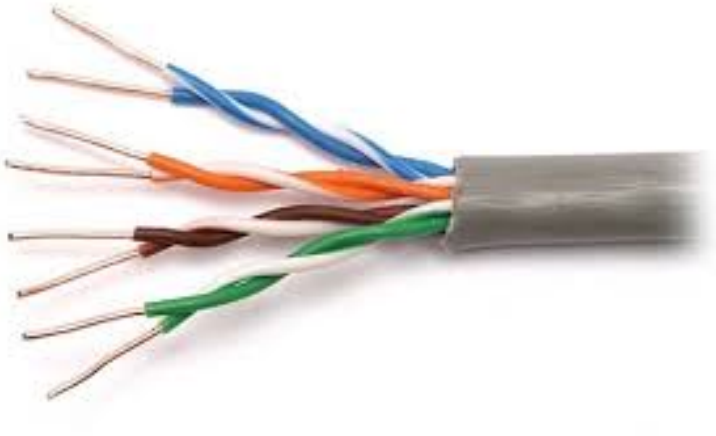

Figure 2.6: Cablul UTP, 4 perechi

Figure 2.7: Cablul S/STP, 4 perechi

- CAT 1 - Categoria 1, se utilizează în reţele de telefonie. Se pot transmite doar semnale codificate analogic, cu frecvenţe de până la 0,4 MHz şi 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 şi digitale cu viteze de transmisie de până la 4 Mbps; Utilizat în reţele LAN vechi (Token-passing ring LANs, IEEE 802.5). Lăţime 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 şi semnale cu frecvenţe de până la 16 MHz. Utilizat în vechile reţele Ethernet 10base-T LANs (IEEE 802.3).
- CAT 4 - Categoria 4, cablu cu 4 perechi torsadate. Se utilizează în reţele cu viteze de transmisie de până la 16 Mbps şi frecvenţe de până la 20 Mhz; Utilizat în reţele token-based sau 10Base-T.
- CAT 5 - Categoria 5, cea mai utilizată categorie de cablu în reţele 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 frecvenţe de până la 100 MHz pe distanţe de maxim 100 m. Transportă date cu viteze de până la 100 Mbps (Fast Ethernet, IEEE 802.3u) şi este utilizat pentru reţele 100 base-T şi 10base-T.
- Categoria 5e – cablu cu 4 perechi torsadate. Poate fi utilizat pentru viteze de transmisie de până la 1 Gbps sau frecvenţe de până la 100 MHz pe distanţe de maxim 100 m. Pentru 100BASE-TX şi 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 frecvenţe de până la 250 MHz pe distanţe 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 frecvenţe de până la 500 MHz pe distanţe de maxim 100 m. Pentru 10GBASE-T Ethernet.
- CAT 7 - Categoria 7, suportă până la 10 Gbps şi frecvenţe de până la 600 MHz. Pentru telefonie, CCTV, 1000BASE-TX în acelaşi cablu, 10GBASE-T Ethernet. Foloseşte F/FTP şi S/FTP.
- CAT 7a - Categoria 7a, suportă până la 10 Gbps şi frecvenţe de până la 1000 MHz. Pentru telefonie, CATV, 1000BASE-TX în acelaşi cablu, 10GBASE-T Ethernet. Foloseşte F/UTP.

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

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

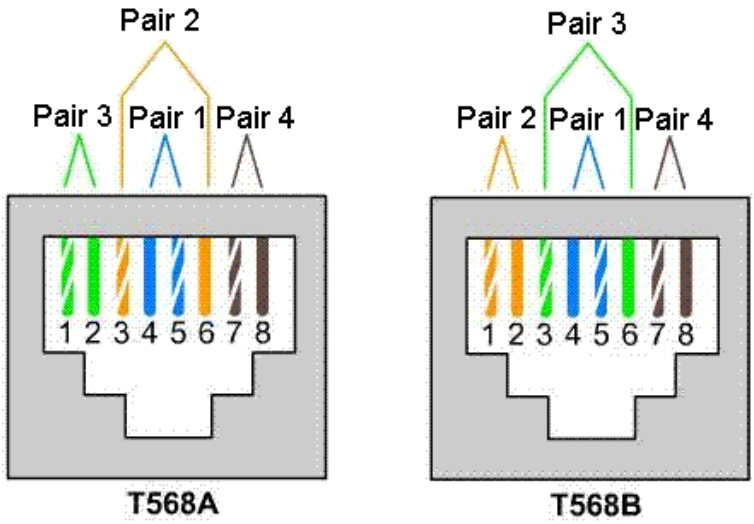

Figure 2.8: Conectori RJ-45, standardele 568A şi 568B de mufare, (Sursa: [9], [10])

Avantaje şi dezavantaje:

+ ieftin şi uşor de instalat;
+ mai puţin sensibil la interferenţ cauzate de echipamentele din apropiere şi la rândul lor nu provoacă interferenţe;
+ STP poate transporta date la viteze mai mari;
- STP e mai voluminos şi 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, 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) şi o imunitate la zgomot foarte ridicată.

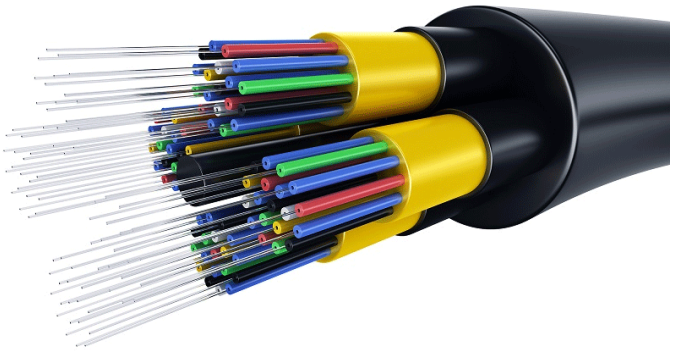

Figure 2.9: Fibră optică, (Sursa: [11])

Tipuri de fibră optică:

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

  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 9µm 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 şi 2.10.

Cele mai multe fibre funcționează în duplex:

- una pentru transmisie, şi
- una pentru recepție.

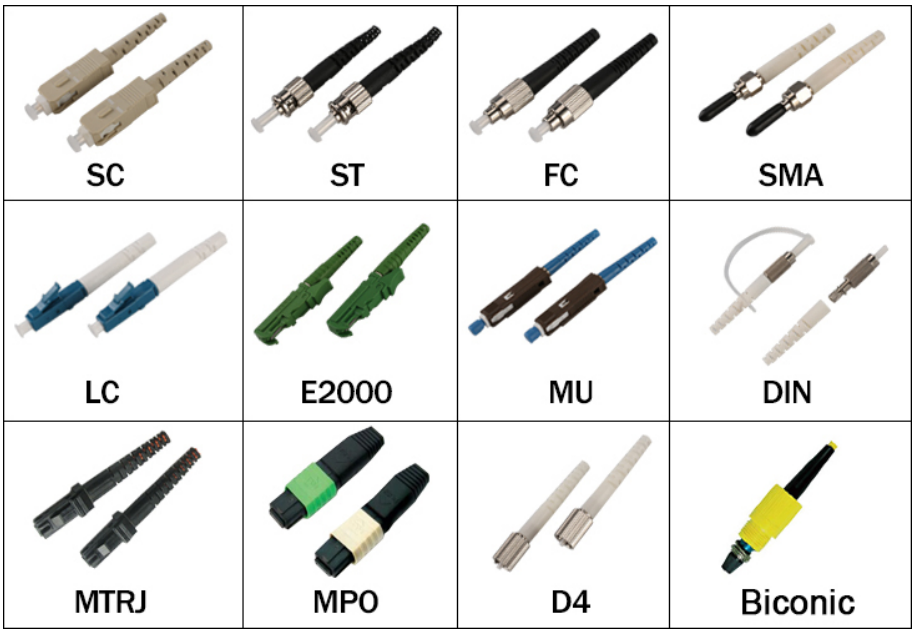

Figure 2.10: Conectori fibră optică, (Sursa: [13])

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

Avantaje şi dezavantaje:

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

2.2.3 TOPOLOGII DE RETEA

Există mai multe topologii de reţea, 2.12:

- Magistrală - Într-o reţea 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 combinaţie între topologiile magistrală şi 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 şi nodurile client au putere de calcul redusă în comparaţie cu serverul.

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

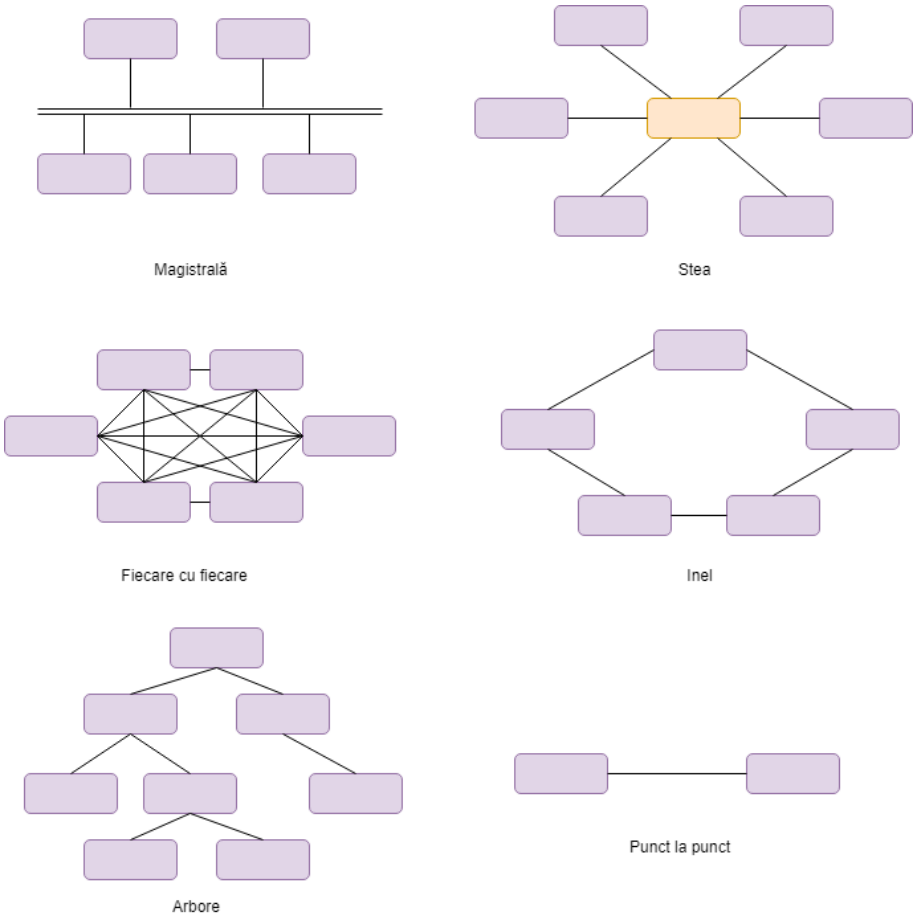

Figure 2.12: Topologii de reţea, (Sursa: [14])

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, sintxă 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 ofer[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, pprodusele 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) așteaptă cereri (incoming requests) de la aceștia.

2.2.6 TCP/IP

Orice dispozitiv conectat la o rețea TCP (Transport Control Protocol)/IP are o adresa unica 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:

- 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.

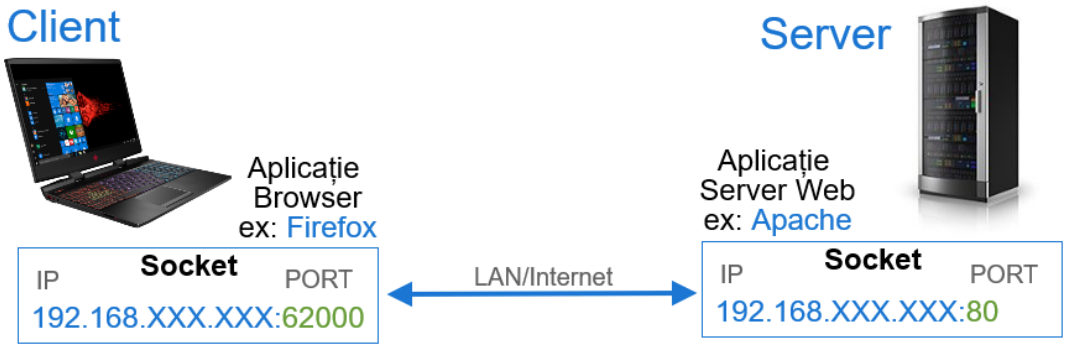

Figure 2.13: Conectare aplicație browser (client) - server web

O conexiune între două calculatoare folosește un socket (soclu) de comunicație care poate fi vazut 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 incheiat.

TCP și UDP sunt protocoale aflate la nivelu 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 orinetat 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.

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 foloseasca 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

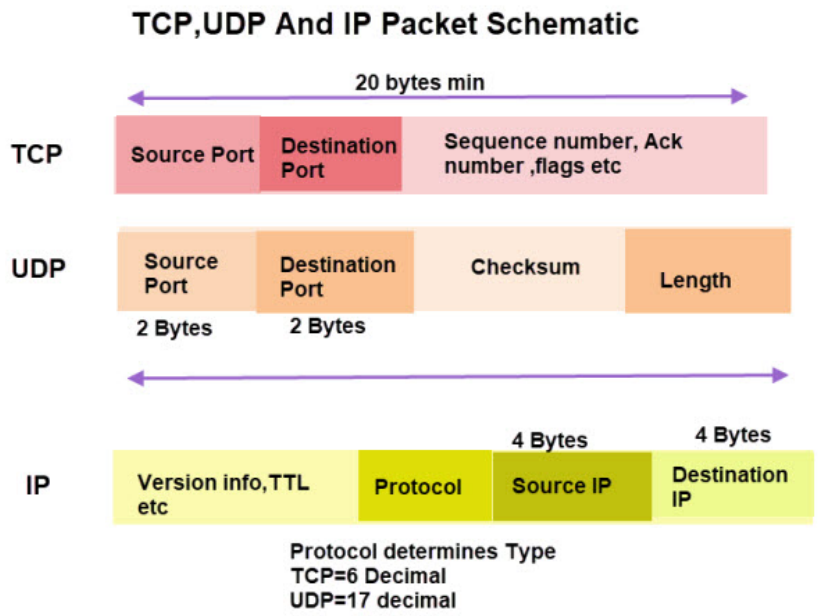

Figure 2.14: TCP, UDP, IP

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 să 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:

- libraria vue.js pentru modulul client
- libraria 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
```

```
<body>
  <div id="app">

    <input class="message" v-model="message" placeholder="
       Name">

    <div class="output-empty" v-if="message == ''">The
       message is empty</div>
    <div class="output" v-if="message != ''">#  {{ message }}
        #</div>


    <button class="process-button" v-on:click="process()"
       >Process</button>

  </div>
  <script src="app.js"></script>
</body>

</html>
```

Fișierul **app.js** conține:

```
var app = new Vue({
    el: '#app',
    data: {
        message: ''
    },
    methods: {
        process: function(){
            console.log(this.message);
        }
    }
})
```

Fișierul **style.css** conține:

```
.message {
  border: 1px solid #777;
  padding: 7px;
  margin: 5px;
}

.output-empty {
  padding: 7px;
  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 transmisia 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 + '
       was deleted');
21 });
22
23 api.listen(3000, function(){
24   console.log('CORS-enabled web server is listening on port
       3000...');
25 });
```

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 dependințele 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          usersService: null
6      },
7      created: function () {
8          usersService = users();
9          usersService.get().then(response => (this.users =
               response.data));
10     },
11     methods: {
12     }
13 })
```

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
```

```
.user-row {
  margin: 20px auto;
  background-color: rgb(255, 254, 234);
  padding: 10px;
  text-align: center;
  width: 400px;
}

.user-row:hover {
  background-color: rgb(255, 251, 175);
}

.user-name {
  width: 150px;
  display: inline-block;
}

.user-city {
  width: 150px;
  display: inline-block;
}
```

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):

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

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

```
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:

```
{
  "name": "chat-server",
  "version": "0.0.1",
  "description": "Data Transmission websocket api",
  "dependencies": {},
  "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.
            entries()) {
24          if (sender.id != client.id) client.emit("message-from
                -server", message);
25      }
26  }
```

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:

```
{
  "name": "chat-client",
  "version": "0.0.1",
  "description": "Data Transmission socket application",
  "dependencies": {},
  "devDependencies": {
    "live-server": "^1.2.1",
    "socket.io-client": "^2.2.0"
  }
}
```

Fișierul **client.js** conține:

```
var socket = io.connect('localhost:8000');

try {

    socket.on('connect', function (data) {
        socket.emit("message-from-client", "Hello to everyone
            from " + checkBrowser());
    });

    socket.on('message-from-server', function (message) {
        console.log(message);
    });

}
catch (err) {
    alert('ERROR: socket.io encountered a problem:\n\n' + err
       );
}

function checkBrowser() {
    var browser = 'Noname browser';
    if (navigator.userAgent.search("Chrome") > -1) {
        browser = "Chrome";
    }
    if (navigator.userAgent.search("Firefox") > -1) {
        browser = "Firefox";
    }
    return browser;
}

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 Aştilean. *Transmisia datelor, notiţe 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

3.2 INTRODUCERE

3.2.1 NOȚIUNI TEORETICE

3.2.1.1 DISTANȚA HAMMING

Distanţa Hamming, (numită după Richard Hamming), se calculează pentru doi vectori de dimensiuni egale. Distanţa Hamming este dată de numărul de poziţii în care vectorii sunt diferiţi. Reprezintă numarul de erori care transformă un vector în celalalt (ex.: 3.1).

Table 3.1: Distanța Hamming - exemple.

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

Informaţia este reprezentată în binar (ex.: cod ASCII). Operaţii obişnuite în binar: 3.2, 3.3, 3.4.

Table 3.2: Adunare în binar - "SAU" / "OR".

+	0	1
0	0	1
1	1	0

Table 3.3: Înmulţire în binar - "ŞI" / "AND".

*	0	1
0	0	0
1	0	1

Table 3.4: Sau exclusiv în binar - "SAU Ex." / "XOR".

$\oplus$	0	1
0	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:

$$V = [a_1 a_2 a_3 ... a_n]; \tag{3.1}$$

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

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

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

$$[h][V]^T = [0]_{mx1} \tag{3.3}$$

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

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

reprezintă matricea de control a codului. Specificarea unui cod liniar implică precizarea matricii de control [H]. Proprietăţile de detecţie sau corecţie ale codului se stabilesc prin alegerea matricei [H] în mod corespunzător. Relaţia (3.3) reprezintă condiţia de codare: se

consideră cuvinte de cod doar acele secvenţe binare de n simboluri, care verifică relaţia (3.3). Decodorul recepţionează cuvântul eronat $[V^{'}] = [V] = [\epsilon]$, unde:

$$[\epsilon]^{'}[\epsilon_1\epsilon_2...\epsilon_n]; \tag{3.5}$$

reprezintă cuvântul eroare, definit prin:

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

Operaţia de decodare constă în calcularea corectorului:

$$[Z] = [H] * [V^{'}]^T; \tag{3.7}$$

ţinând cont de legătura dintre $[V^{'}]$, $[V]$ şi $[\epsilon]$ şi de relaţia de codare (3.3) se poate scrie:

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

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

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:

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

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

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

$$[V] = [c_1 c_2 a_3 c_4 a_5 a_6 a_7]; \tag{3.10}$$

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

Datorită acestei alegeri, fiecare relaţie din sistemul (3.2) va conţine un singur simbol de control, permiţând astfel rezolvarea usoară a sistemului de m ecuaţii cu m necunoscute, după cum urmează:

$$\begin{cases} c_1 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ c_2 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ c_4 \oplus a_3 \oplus a_5 \oplus ... \oplus = 0 \\ ... \end{cases} \tag{3.11}$$

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

$$[Z]^{'} = [H] * [\epsilon]^T \tag{3.12}$$

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

3.2.2.2 Codul Hamming - corector de o eroare, detector de două erori

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

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \dots a_n; \tag{3.13}$$

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

$$[V_{H2}] = [c_0 V_{H1}]; \tag{3.14}$$

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 biţi) şi apoi a unei ultime linii de n+1 unităţi:

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

La recepţie se calculează:

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

Dacă cuvântul recepţionat este $[v^{'}] = [c_0^{'} c_1^{'} c_2^{'} a_3^{'} c_4^{'} a_5^{'} ... a_n^{'}]$, din (3.16) rezultă:

$$c_0 c_1 c_2 a_3 c_4 a_5 a_6 \ldots a_n; \tag{3.17}$$

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 $z_0 = 0$ si $[Z_{H1}] = [0]$, se decide că în cuvântul recepţionat sunt două erori, iar când $z_0 = 1$ se decide că în cuvântul recepţionat este o eroare pe poziţia rezultată transcriind în zecimal componentele corectorului $[Z_{H1}]$;
- dacă $z_0 = 0$ si $[Z_{H1}] = [0]$ se decide că ceea ce s-a recepţionat este corect.

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) === false)
48                  return false;
49              }
50              return true;
51          },
52          validateBit: function(character){
53              if (character === null) return false;
54              return (parseInt(character) === 0 ||
55              parseInt(character) === 1);
56          }
57      }
58  })
```

Fișierul **style.css** conține:

```
1  body {
2    margin: 30px;
3  }
4
5  select {
6    margin-top: -5px;
7    padding: 5px;
8  }
9
10 ul {
11   list-style: none;
12   display: inline-block;
13   margin: 0px;
14   margin-bottom: 20px;
15   margin-top: 0px;
16   padding: 0px;
17   padding-right: 10px;
18 }
```

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

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: '+decoded
          .errorPosition);
11    }
12    response.json('Message received without errors');
13  });
14
15  api.listen(3000, function(){
16    console.log('CORS-enabled web server is listening on port
        3000...');
17  });
18
19  function distortBit(bits, index){
20    bits[index] = (bits[index]+1) % 2;
21    return bits;
22  }
```

Fișierul **api.js** conține:

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

Fișierul **hamming.js** conține:

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

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

Fișierul **package.json** conține:

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

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 Aştilean. *Transmisia datelor, notiţe curs.*

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

4.1 OBIECTIVE

- Prezentarea generală a modalităţilor de compresie a datelor.
- Prezentarea generală a algoritmului Huffman de compresie a datelor.
- Prezentarea generală a algoritmului Shannon-Fano de compresie a datelor.

4.2 INTRODUCERE

Compresia datelor este un proces de codare prin care se urmăreşte micşorarea redundanţei 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 foloseşte, eventual într-o formă intermediară, reprezentarea prin simboluri binare a acestuia, şi 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 acţionat prin procesul de compresie, distingem două categorii de algoritmi de compresie: fără pierderi şi cu pierderi.

Algoritmii de compresie fără pierderi sunt reversibili, prin decompresie obţinându-se mesajul original întocmai. Algoritmii de compresie cu pierderi au ca rezultat diferenţe relativ mici între mesajul rezultat prin decompresie şi cel original şi sunt utilizaţi în compresia imaginilor şi a mesajelor video şi audio.

În cele ce urmează ne referim la algoritmii de compresie fără pierderi, folosiţi 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 şi codarea Huffman statică, care ţin seama de frecvenţa caracterelor în text, caracterelor cu probabilitatea mai mare de apariţie în text alocându-li-se cuvinte de cod mai scurte. Cu algoritmii de compresie adaptivi, dinamici, procesul de compresie se realizează în timp real, probabilitatea de apariţie a caracterelor fiind aproximată permanent, pe măsură ce sursa generează mesajul.

4.2.1 NOŢIUNI TEORETICE

Tehnicile de compresie sunt utile pentru fişierele text, în care anumite caractere apar cu o frecvenţă mai mare decât altele, pentru fişiere ce codifică imagini sau sunt reprezentări digitale ale sunetelor ori ale unor semnale analogice, ce pot conţine numeroase motive care se repetă. Chiar dacă astăzi capacitatea dispozitivelor de memorare a crescut foarte mult, algoritmii de compresie a fişierelor rămân foarte importanţi, datorită volumului tot mai mare de informaţii ce trebuie stocate. În plus, compresia este deosebit de utilă în comunicaţii, transmiterea informaţiilor fiind mult mai costisitoare decât prelucrarea lor.

Una dintre cele mai răspândite tehnici de compresie a fişierelor text, care, în funcţie de caracteristicile fişierului ce urmează a fi comprimat, conduce la reducerea spaţiului de memorie necesar cu 20% – 90%, a fost descoperită de D. Huffman în 1952 şi 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 biţi (lungime fixă), se utilizează un cod mai scurt pentru caracterele care sunt mai frecvente şi coduri mai lungi pentru cele care apar mai rar.

Definiţii:

- *Compresia datelor* – metode şi tehnici de reducere a volumului unui set de date reprezentate sub formă binară, cu posibilitatea reconstruirii lor fie exact, fie aproximativ.
- Set de date – şir 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 simbol* = 8 biţi (ASCII) → setul de date se construieşte din maxim 28 = 256 simboluri distincte.
- *Alfabet* – mulţimea simbolurilor distincte dintr-un set de date.
- *Lungimea setului de date* – numărul total de biţi pe care este reprezentat setul de date.
- *Volumul unui set de date* – numărul de simboluri care compun setul de date.

- *Compresie*
  - *conservativă* - metode de reconstrucţie exactă a setului de date din versiunea sa comprimată – text, programe sursă.
  - *neconservativă* - metode de reconstrucţie aproximative, inexacte, a setului de date original – semnal audio, imagine.
- *Rata de compresie* – parametru prin care se măsoară performanţele metodei de compresie utilizate:

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

  unde:
  - c - lungimea setului de date comprimat.
  - o - lungimea setului de date original.
  - γ:
    - = 100%, metoda de compresie ideală, (câştig maxim).
    - = 0%, metoda nu realizează compresia, (câştig nul).
    - < 0%, metoda nu realizează compresia, (expandarea datelor).
- γ - măsoară câştigul relativ de lungime obţinut 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* – creşterea lungimii setului de date procesat faţă de cel original (în cazul metodelor ineficiente).
- *Decompresie* – acţiunea de reconstruire (exactă sau aproximativă) a setului de date original, plecând de la setul de date original.
- *Redundanţa* – se măsoară cu ajutorul unui model statistic asociat setului de date, plecând de la frecvenţele de apariţie ale simbolurilor alfabetului de date.
- *Frecvenţa de apariţie* – raportul între numărul apariţiilor simbolului şi numărul total de simboluri ai setului de date notate cu v(s). (probabilitatea de apariţie a simbolului "s", P(s)).

- *Entropie* – numărul minim de biţi necesari pentru recodificarea simbolului "s", fără a se realiza confuzia cu alte simboluri. (poate fi fracţionar, dar de obicei se rotunjeşte la întregul superior). Va fi cel mult egală cu numărul iniţial de biţi.

$$H(s) = -log_2 v(s) \rightarrow v(s) = 2^{-H(s)} \in (0,1]. \\ 0 < v(s) \leqslant 1 \rightarrow H(s) > 0. \tag{4.2}$$

4.2.1.1 Criterii de apreciere a unui cod

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

$$n = \sum_{i=1}^{n} n_i P_i. \tag{4.3}$$

unde:

- P_i sunt probabilităţile 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 condiţia de asigurare a entropiei informaţionale pe simbol al alfabetului de cod:

$$n \geq n_{min} = \frac{H}{\lim q} \tag{4.4}$$

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

$$\eta = \frac{\overline{\mathrm{n}_{min}}}{\overline{\mathrm{n}}}. \tag{4.5}$$

4.2.1.2 Codul Huffman

D. Huffman a elaborat un algoritm Greedy care construieşte un cod prefix optimal, numit cod Huffman. Prima etapă în construcţia codului Huffman este calcularea numărului de apariţii ale fiecărui caracter în text. Există situaţii în care putem utiliza frecvenţele standard de apariţie a caracterelor, calculate în funcţie de limbă sau de specificul textului.
Fie $C = c_1, c_2, ..., c_n$ mulţimea caracterelor dintr-un text, iar $f_1, f_2, ..., f_n$, respectiv, numărul lor de apariţii. Dacă l_i ar fi lungimea şirului ce codifică simbolul c_i, atunci lungimea totală a reprezentării ar fi:

$$L = \sum_{i=1}^{n} l_i f_i. \tag{4.6}$$

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:

- Iniţial, considerăm o partiţie a mulţimii $C = \{\{c_1, f_1\}, \{c_2, f_2\}, ..., \{c_n, f_n\}\}$, reprezentată printr-o pădure de arbori formaţi dintr-un singur nod.
- Pentru a obţine arborele final, se execută n-1 operaţii de unificare.

Unificarea a doi arbori A_1 şi A_2 constă în obţinerea unui arbore A, al cărui subarbore stâng este A_1, subarbore drept A_2, iar frecvenţa rădăcinii lui A este suma frecvenţelor rădăcinilor celor doi arbori. La fiecare pas unificăm 2 arbori ale căror rădăcini au frecvenţele cele mai mici.

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

Pasul 1: Se parcurge secvenţial D şi se construieşte $A^0 \rightarrow N(s)$ pentru fiecare $s \in A^0$

Pasul 2: Rearanjarea simbolurilor alfabetului: $A^0 = \{s_1, s_2, ..., s_n\}$ cu $N(s_1) \geq N(s2) \geq ... \geq N(s_n)$. Se renotează şi reindexează elementele alfabetului astfel:

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

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

Pasul 3: Construcţia arborelui binar se realizează simultan cu formarea noului alfabet curent. Noul alfabet curent, $A^{0,(n-k-1)}$ cu k=0 ... N-2, se construieşte 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" şi care are contorul egal cu suma contoarelor celor două simboluri.
Dacă există mai mult de două simboluri slabe având acelaşi contor, se aleg ultimele două din alfabet → cardinalul noului alfabet scade cu o unitate faţă de cardinalul alfabetului precedent.

$$\begin{aligned} A^{0,(n-k-1)} &= [A^{0,(n-k)} \{t_p, t_q\}] \cup \{t_{n-k-1}\} \\ t_{n-k-1} &= \text{simbol virtual} \\ N(t_{n-k-1}) &= N(t_p) + N(t_q) \\ \#A^{0,(n-k-1)} &= \#A^{0,(n-k)} - 1 = n-k-1 \end{aligned}$$

Simbolul virtual etichetează un nod intermediar al arborelui, având ca descendenţi cele două noduri etichetate de simboluri eliminate. Fiecare nod intermediar din arbore are câte 2 descendenţi, contorul fiului din stânga fiind cel puţin 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 şi rădăcina arborelui (t_1).

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

4.2.1.3 CODUL SHANNON - FANO

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

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

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

$$v(s) = \frac{N(s)}{D}, \forall s \in A^0 \text{(estimarea probabilităţii de apariţie a lui s)}$$

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

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

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

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

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

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

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

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

Pasul 4: Construcţia arborelui binar. Ramura dreaptă primeşte ponderea "1" iar ramura stângă primeşte ponderea "0".

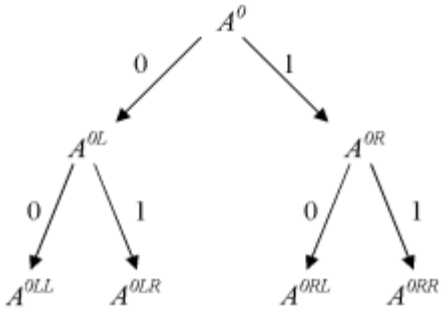

Figure 4.1: Arbore Shannon-Fano

Pasul 5: Se repetă paşii 3 şi 4 pentru fiecare nod terminal al arborelui (subalfabet) care conţine mai mult de un simbol.

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

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

4.2.2 APLICAŢII

4.2.2.1 ALGORITMUL HUFFMAN

Construcţia 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 rădacină.
Pasul 1: Fiecare simbol cu frecvenţa de apariţie este trecut într-o etichetă, 4.2.

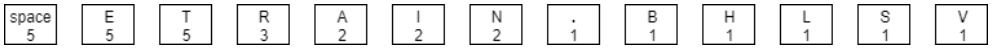

Figure 4.2: Arbore Huffman - pasul 1

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

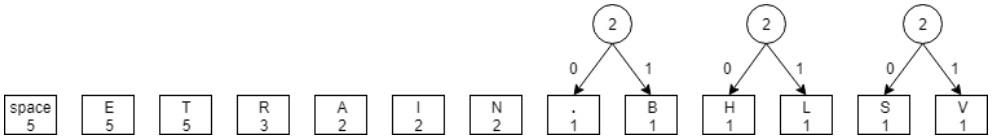

Figure 4.3: Arbore Huffman - pasul 2

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

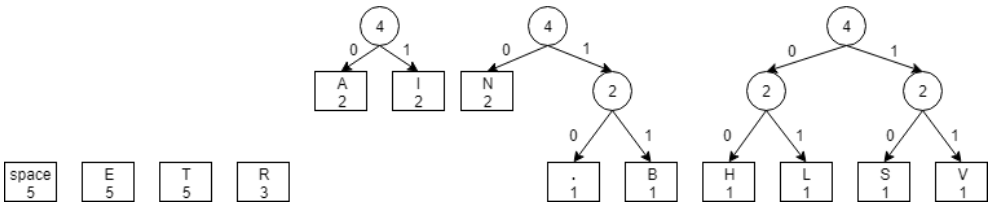

Figure 4.4: Arbore Huffman - pasul 3

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

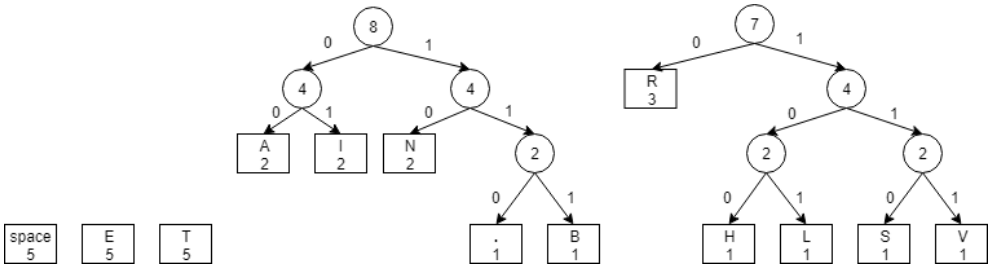

Figure 4.5: Arbore Huffman - pasul 4

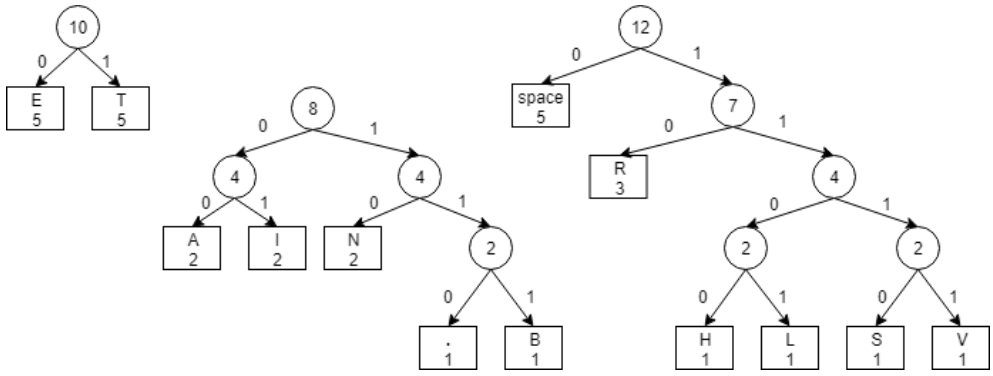

Figure 4.6: Arbore Huffman - pasul 5

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

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

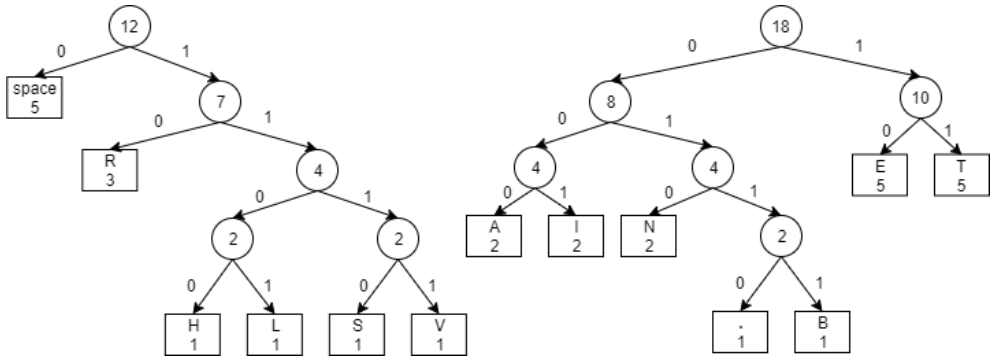

Figure 4.7: Arbore Huffman - pasul 6

Pasul 7: Arborele final, 4.8.

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. biţi	2	3	3	3	4	4	4	5	5

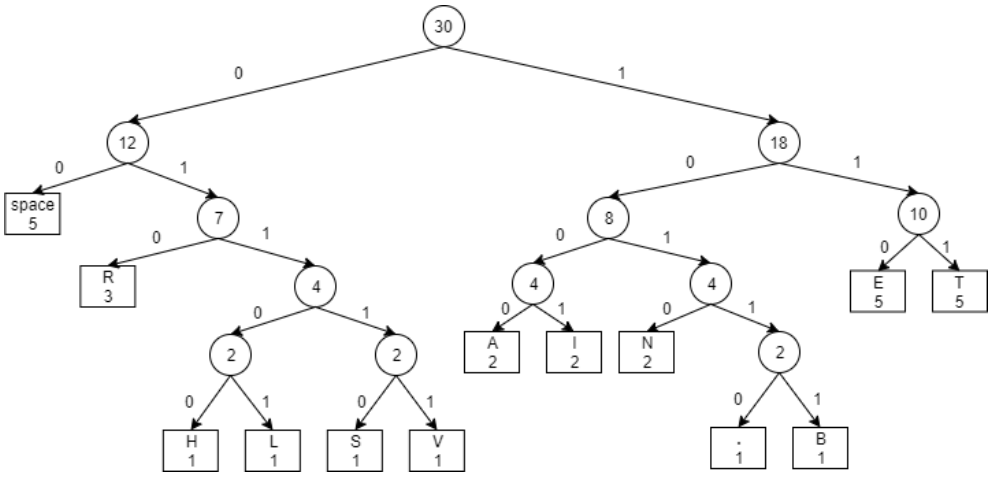

Figure 4.8: Arbore Huffman

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

Codurile numerice sunt exprimate în zecimal, dar în realitate aceste coduri sunt binare şi au câte o lungime de 8 biţi. Aranjarea informaţiei auxiliare poate fi realizată şi în altă manieră, dar modul de aranjare trebuie cunoscut şi 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."
Construcţia 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 construieste în cinci iteraţii 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 “spaţiu” este cel mai frecvent simbol folosit şi a atins o frunză fiind recodificat pe 2 biţi: 00 (în loc de codul original pe 8 biţi 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 iteraţie, iar noile lor coduri nu depăsesc lungimea de 3 biţi (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 biţi în loc de 8 biţi.

Pas 5:

D = IT IS BETTER LATER THAN NEVER. Noile coduri nu depăsesc 5 biţi → minimizarea redundanţei şi deci maximizarea ratei de compresie.

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

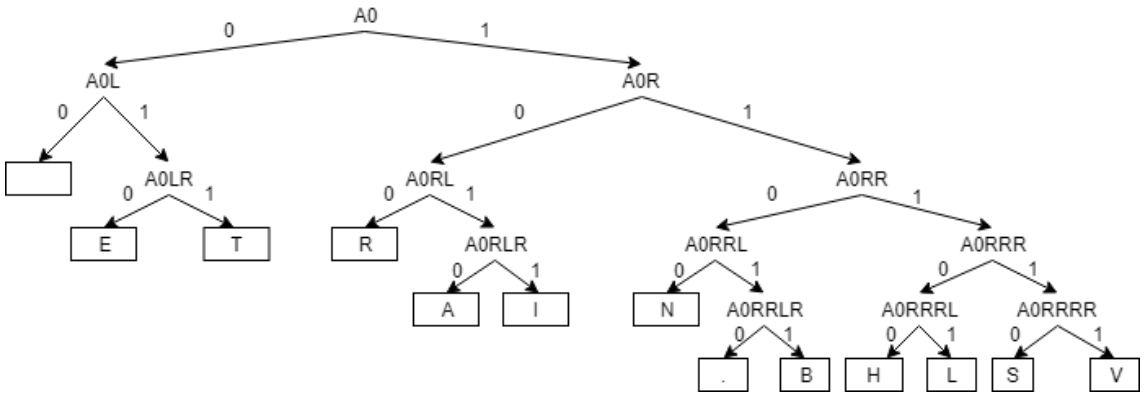

Figure 4.9: Arbore Shannon-Fano

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 informaţiei utile şi cele ale informaţiei auxiliare se succed fără separatori între ele. Între informaja auxiliară şi cea utilă ar putea să apară un separator sub forma unui simbol virtual, în absenţa numărului de simboluri ai alfabetului, N. În acest caz, separatorul de informaţie trebuie să aibă un cod mai mare de 255, de exemplu 256, dar reprezentarea lui ar fi pe 16 biţi, în loc de 8 biţi, cât ocupă N.

Analiza performanţelor de compresie

Vom face o analiza a entropiei (numărul de biţi efectiv alocaţ în urma compresiei). Există două tipuri de entropii: cea a informaţiei auxiliare $H_a(D)$ şi cea a informaţiei utile $H_u(D)$.

Decompresia datelor:

- se citeste informaţia auxiliară (5 octeţi consecutivi indică o pereche simbol – contor);

- se construieşte arborele binar asociat;
- se citeşte informaţia utilă secvenţial, bit cu bit;
- fiecare simbol al setului de date este decriptat şi se foloseşte 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 ieşire.

Atingerea unei frunze reiniţializează căutarea din rădăcina arborelui.

Observaţie: 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 Desfăşurarea lucrării

Lucrarea constă în compresia unui text folosind cei doi algoritmi prezentaţi mai sus.
Resurse folosite:

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

4.3.1 Aplicație cu D3js

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

Pentru a instala dependințele 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>
```

```
<head>
  <meta content=" text/html;charset=utf -8" http-equiv="
     Content-Type">
  <meta content ="utf-8"http-equiv="encoding">
  <!-- load the style.css -->
  <link rel="stylesheet" href="style.css">
  <!-- load the d3.js library https://d3js.org/d3.v5.min.js
     -->
  <script src="https://d3js.org/d3.v5.min.js"></script>
</head >
<body>
    <!-- arbore.js  - a function to display a tree
            - is based on "collapsible tree" sample from d3
               library
    -->
    <script src="arbore.js"></script>
</body>
</html>
```

Fișierul **style.css** conține:

```
.node circle {
  fill: #fff;
  stroke: steelblue;
  stroke-width: 3px;
}

.node text {
  font: 12px sans-serif;
}

.link {
  fill: none;
  stroke: #ccc;
  stroke-width: 2px;
}
```

Fișierul **arbore.js** conține:

```


```

```
6
7
8
9  var treeData = {
10   "name": "A0",
11     "children": [
12       {
13         "name": "Level 1: _",
14         "children": [
15           { "name": "Level 2: _",
16       "children": [
17         { "name": "Level 3: _",
18         "children": [
19             { "name": "Level 4: _ - space (frequency" },
20             { "name": "Level 4: _ - 3 (frequency)" }
21           ]
22         },
23         { "name": "Level 3: _ - q (frequency)" }
24       ]
25       },
26           { "name": "Level 2: _ - M (frequency)" }
27         ]
28       },
29       { "name": "Level 1: _",
30         "children": [
31           { "name": "Level 2: _ - A (frequency)" },
32           { "name": "Level 2: _ - * (frequency)" }
33         ]
34     }
35     ]
36 };
37
38
39 var margin = {top: 20, right: 90, bottom: 30, left: 90},
40     width = 1960 - margin.left - margin.right,
41     height = 800 - margin.top - margin.bottom;
42
43
44
45
46 var svg = d3.select("body").append("svg")
47     .attr("width", width + margin.right + margin.left)
```

```
48      .attr("height", height + margin.top + margin.bottom)
49    .append("g")
50      .attr("transform", "translate("
51            + margin.left + "," + margin.top + ")");
52
53  var i = 0,
54      duration = 750,
55      root;
56
57
58  var treemap = d3.tree().size([height, width]);
59
60
61  root = d3.hierarchy(treeData, function(d) { return d.children
        ; });
62  root.x0 = height / 2;
63  root.y0 = 0;
64
65
66  root.children.forEach(collapse);
67
68  update(root);
69
70
71  function collapse(d) {
72    if(d.children) {
73      d._children = d.children
74      d._children.forEach(collapse)
75      d.children = null
76    }
77  }
78
79  function update(source) {
80
81    var treeData = treemap(root);
82
83
84    var nodes = treeData.descendants(),
85        links = treeData.descendants().slice(1);
86
87
88    nodes.forEach(function(d){ d.y = d.depth * 180});
89
```

```
90
91
92    var node = svg.selectAll('g.node')
93        .data(nodes, function(d) {return d.id || (d.id = ++i);
              });
94
95
96    var nodeEnter = node.enter().append('g')
97        .attr('class', 'node')
98        .attr("transform", function(d) {
99          return "translate(" + source.y0 + "," + source.x0 + "
                )";
100     })
101     .on('click', click);
102
103
104   nodeEnter.append('circle')
105       .attr('class', 'node')
106       .attr('r', 1e-6)
107       .style("fill", function(d) {
108           return d._children ? "lightsteelblue" : "#fff";
109       });
110
111
112   nodeEnter.append('text')
113       .attr("dy", ".35em")
114       .attr("x", function(d) {
115           return d.children || d._children ? -13 : 13;
116       })
117       .attr("text-anchor", function(d) {
118           return d.children || d._children ? "end" : "start";
119       })
120       .text(function(d) { return d.data.name; });
121
122
123   var nodeUpdate = nodeEnter.merge(node);
124
125
126   nodeUpdate.transition()
127     .duration(duration)
128     .attr("transform", function(d) {
129         return "translate(" + d.y + "," + d.x + ")";
```

```
130       });
131
132
133     nodeUpdate.select('circle.node')
134       .attr('r', 10)
135       .style("fill", function(d) {
136           return d._children ? "lightsteelblue" : "#fff";
137       })
138       .attr('cursor', 'pointer');
139
140
141     var nodeExit = node.exit().transition()
142         .duration(duration)
143         .attr("transform", function(d) {
144             return "translate(" + source.y + "," + source.x + "
                    )";
145         })
146         .remove();
147
148
149     nodeExit.select('circle')
150       .attr('r', 1e-6);
151
152
153     nodeExit.select('text')
154       .style('fill-opacity', 1e-6);
155
156
157
158     var link = svg.selectAll('path.link')
159         .data(links, function(d) { return d.id; });
160
161
162     var linkEnter = link.enter().insert('path', "g")
163         .attr("class", "link")
164         .attr('d', function(d){
165           var o = {x: source.x0, y: source.y0}
166           return diagonal(o, o)
167         });
168
169
170     var linkUpdate = linkEnter.merge(link);
```

```
171
172
173   linkUpdate.transition()
174       .duration(duration)
175       .attr('d', function(d){ return diagonal(d, d.parent) })
              ;
176
177
178   var linkExit = link.exit().transition()
179       .duration(duration)
180       .attr('d', function(d) {
181         var o = {x: source.x, y: source.y}
182         return diagonal(o, o)
183       })
184       .remove();
185
186
187   nodes.forEach(function(d){
188     d.x0 = d.x;
189     d.y0 = d.y;
190   });
191
192
193   function diagonal(s, d) {
194     path = `M ${s.y} ${s.x}
195             C ${(s.y + d.y) / 2} ${s.x},
196               ${(s.y + d.y) / 2} ${d.x},
197               ${d.y} ${d.x}`
198     return path
199   }
200
201
202   function click(d) {
203     if (d.children) {
204         d._children = d.children;
205         d.children = null;
206       } else {
207         d.children = d._children;
208         d._children = null;
209       }
210     update(d);
211   }
```

```
212 }
```

4.3.1.1 Algoritmul Huffman

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

```
1
2  var treeData =
3    {
4      "name": "A0 = 30",
5      "children": [
6        {
7          "name": "Level 1: 18",
8          "children": [
9            { "name": "Level 2: 8",
10               "children": [
11                 { "name": "Level 3: 4",
12                     "children": [
13                       { "name": "Level 4: 2",
14                           "children": [
15                             { "name": "Level 5: B" },
16                             { "name": "Level 5: ." }
17                           ]
18                       },
19                       { "name": "Level 4: N"}
20                     ]
21                 },
22                 { "name": "Level 3: 4",
23                     "children": [
24                       { "name": "Level 4: I"},
25                       { "name": "Level 4: A" }
26                     ]
27                 }
28               ]
29           },
30           { "name": "Level 2: 10",
31               "children": [
32                 { "name": "Level 3: T"},
33                 { "name": "Level 3: E" }
34               ]
35           }
36         ]
```

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

4.3.1.2 Algoritmul Shannon-Fano

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

```
1
2  var treeData =
3    {
4      "name": "A0",
5      "children": [
6        {
7          "name": "Level 1: A0R",
8          "children": [
```

```
9           { "name": "Level 2: A0RR",
10              "children": [
11                { "name": "Level 3: A0RRR",
12                    "children": [
13                      { "name": "Level 4: A0RRRR",
14                          "children": [
15                            { "name": "Level 5: A0RRRRR = V"
                                  },
16                            { "name": "Level 5: A0RRRRL = S"
                                  }
17                          ]
18                      },
19                      { "name": "Level 4: A0RRRL",
20                          "children": [
21                            { "name": "Level 5: A0RRRLR = L"
                                  },
22                            { "name": "Level 5: A0RRRLL = H"
                                  }
23                          ]
24                      }
25                    ]
26                },
27                { "name": "Level 3: A0RRL",
28                    "children": [
29                      { "name": "Level 4: A0RRLR",
30                          "children": [
31                            { "name": "Level 5: A0RRLRR = B"
                                  },
32                            { "name": "Level 5: A0RRLRL = ."
                                  }
33                          ]
34                      },
35                      { "name": "Level 4: A0RRLL = N" }
36                    ]
37                }
38              ]
39          },
40          { "name": "Level 2: A0RL",
41              "children": [
42                { "name": "Level 3: A0RLR",
43                    "children": [
44                      { "name": "Level 4: A0RLRR = I" },
45                      { "name": "Level 4: A0RLRL = A" }
```

```
46                      ]
47                    },
48                    { "name": "Level 3: A0RLL = R" }
49                  ]
50              }
51            ]
52        },
53        { "name": "Level 1: A0L",
54          "children": [
55            { "name": "Level 2: A0LR",
56                "children": [
57                  { "name": "Level 3: A0LRR = T" },
58                  { "name": "Level 3: A0LRL = E" }
59                ]
60             },
61            { "name": "Level 2: A0LL = spatiu" }
62          ]
63      }
64      ]
65  };
```

4.4 PROBLEME PROPUSE

1. Să se implementeze algoritmul Huffman de compresie. Să se afişeze arborele de compresie obţinut.
2. Să se implementeze algoritmul Shannon-Fano de compresie. Să se afişeze arborele de compresie obţinut.
3. Pentru un şir de caractere preluat de la tastatură:
   - Aplicaţi algoritmul Huffman de compresie;
   - Să se calculeze rata de compresie;
   - Să se afişeze noile coduri obţinute pentru fiecare simbol în parte.
4. Pentru un şir de caractere preluat de la tastatură:
   - Aplicaţi algoritmul Shannon-Fano de compresie;
   - Să se calculeze rata de compresie;
   - Să se afişeze noile coduri obţinute pentru fiecare simbol în parte.

4.5 BIBLIOGRAFIE

BIBLIOGRAFIE

[1] Adina Aştilean. *Transmisia datelor, notiţe curs.*

5 CODURI ÎN BANDA DE BAZĂ

5.1 OBIECTIVE

- Prezentarea generală a modalităţilor de sincronizare a receptorului cu transmiţătorul în cazul transmisiei asincrone şi sincrone.
- Prezentarea generală a codurilor de linie fără întoarcere la zero, cu întoarcere la zero şi bifazice.

5.2 INTRODUCERE

5.2.1 NOŢIUNI TEORETICE

Transmisia unui semnal se poate realiza direct în banda sa de bază, dacă se introduce pe canal semnalul nemodificat, cu spectrul său original, aşa cum este obţinut de la traductorul care preia informaţia din mediul real şi 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 modulaţia pe purtătoare a semnalului util, care ar rezulta în modificarea spectrului de frecvenţe iniţial, prin translatare.

Acest mod de transimisie a informaţiei are avantajul simplităţii, deoarece semnalele nu sunt prelucrate suplimentar, dar are şi o serie de dezavantaje. Banda relativă de frecvenţă este largă (banda relativă = banda semnalului raportată la frecvenţa medie) şi poate fi uşor influenţată de perturbaţii cu diverse frecvenţe. Semnalele nu pot fi transmise în banda de bază la distanţe mari datorită imunităţii scăzute la perturbaţii 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 corespunzătoare valorilor logice de “0” şi “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 şi sfârşitul fiecărei celule de bit (sincronizarea pe bit), dar şi împachetarea globală a şirului de date în caractere şi blocuri de date (sincronizarea pe caracter şi pe cadru). Sincronizarea receptorului cu emiţătorul poate fi realizată în două moduri, în funcţie de relaţia dintre ceasul transmiţătorului şi cel al receptorului.

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

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

Fiecare caracter este încadrat între un bit de ”START” şi 1, ½, sau 2 biţi de “STOP”. Biţii de “START” şi cei de “STOP” au polaritate diferită, pentru asigurarea a cel puţin unei tranziţii 0→1→0 între două caractere succesive care sosesc fără întârziere între ele. Prima tranziţie 1→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 porneşte recepţia unui caracter pe frontul negativ al bitului de “START”.

Sincronizarea pe bit se realizează prin determinarea cât mai precisă a tranziţiei de “START”. Ceasul receptorului trebuie să aibă perioada de cel puţin 16 ori mai mică decât durata de transmisie a unui bit, pentru a minimiza eroarea de detecţie a începutului de caracter, eroare ce apare datorită asincronismului între tranziţiile de pe linia de recepţie şi tranziţiile 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 şi sfârşitului caracterului, se realizează prin simpla numărare a biţilor veniţi pe linie, cu verificarea corectitudinii biţilor finali de “STOP” care trebuie sa aibă valoarea logică “1”. Acest lucru este posibil deoarece numărul de biţi pe caracter este stabilit la configurarea transmiţătorului şi receptorului, reprezentând o caracteristică a transmisiei.

Sincronizarea pe cadru (bloc de caractere) se realizează în mod distinct, în funcţie de informaţia transmisă, adică şir de coduri ASCII (text) sau date binare.

Codurile binare pot fi grupate în patru categorii:

1. Coduri NRZ (’Non Return to Zero’ - făară 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’).

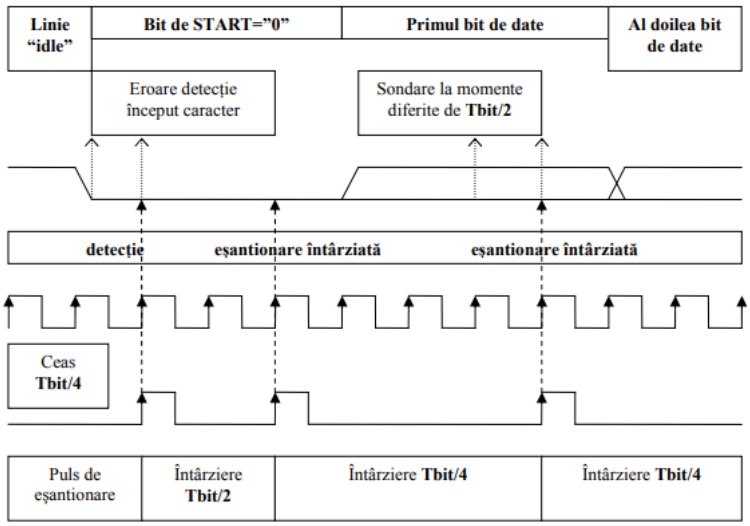

Figure 5.1: Apariţia şi propagarea erorii de detecţie a începutului de caracter, (Sursa: [2])

O altă diferenţiere a codurilor se poate face după semnul tensiunilor sau curenţilor asociaţi: dacă semnele sunt diferite pentru '0' şi '1', codul este bipolar, iar pentru semne de acelaşi fel, codul se numeşte unipolar. Codurile bipolare permit utilizarea ca nivel de referinţă a potenţialului pământului ('ground').

5.2.1.1 CODURI NRZ

Acest tip de codare foloseşte 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 cunoaşte câteva variante:

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

Debitul maxim teoretic care poate fi atins într-o transmisie NRZ este egal cu dublul benzii de frecvenţă ocupată de către semnal (pot fi transmişi 2 biţi/Hertz). Principalul dezavantaj al codării de tip NRZ îl constituie lipsa tranziţiilor în cazul unor secvenţe lungi de biţi 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 informaţie 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 biŃi de "1", iar în cazul biţilor de "0" aceste pulsuri lipsesc complet. La codul de linie RZ-bipolar, pentru codificarea biţilor 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 recepţie este aproape directă, deoarece atât şirurile lungi de "0" cât şi cele de biţi "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 şi sub denumirea de codare Manchester.
2. Biphase - mark - $BI\Phi - M$. Această codare presupune apariţia unei tranziţii la începutul oricărui interval de bit. Dacă bitul este de „1", atunci o a doua tranziţie va apare la mijlocul intervalului de bit. Pentru transmisia unui „0" nu se va mai produce nici un fel de tranziţie.
3. Biphase - space - $BI\Phi - S$. Este exact inversa codării $BI\Phi - M$ (o tranziţie la începutul intervalului de bit, urmată de o altă tranziţie la jumătatea acestui interval dacă se transmite „0", sau fără tranziţie dacă se transmite „1").

Codarea Manchester

Ideea care stă la baza codării Manchester este aceea de a determina o tranziţie pentru semnalul emis, tranziţie care să apară la mijlocul perioadei de bit. Astfel, un „1" este reprezentat printr-o tranziţe de la nivelul +V la nivelul –V, în timp ce unei tranziţii de la nivelul –V la nivelul +V îi corespunde un „0". Este evident că în acest fel se asigură sincronizarea între emiţător şi receptor, chiar şi în cazul transmiteri unor secvenţe lungi de „0" sau „1". Mai mult decât atât, întrucât simbolurile binare sunt reprezentate prin tranziţii şi nu prin niveluri constante (stări) ca la codarea de tip NRZ, scade drastic probabilitatea apariţiei unor erori cauzate de mediul de transmisie. Un zgomot care afectează semnalul poate modifica nivelurile transmise, dar este puţin probabil că el va duce la inversarea tranziţiei sau la lipsa ei, conducând astfel la erori la recepţie.

Dezavantajul codării Manchester constă în faptul că, pentru a transmite cu un anumit debit binar, este nevoie de o bandă de frecvenţe disponibilă dublă faţă 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 lăţime de bandă de 10MHz). Acest inconvenient face codarea Manchester dificil de utilizat pentru debite ridicate.

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

Codarea Manchester diferenţială

La baza codării Manchester diferenţiale stă prezenţa sau absenţa unei tranziţii la începutul intervalului de tact. Astfel, un bit de „1" este reprezentat prin lipsa unei tranziţii, în timp ce fiecare bit de „0" este semnificat prin prezenţa unei tranziţii. Avantajele, respectiv dezavantajele acestui tip de codare sunt în general aceleaşi ca la codarea Manchester nediferenţială.

Utilizări: Reţele 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 potenţial nul (absenţa semnalului electric pe linie), în timp ce biţii de „1" sunt reprezentaţi alternativ prin tensiuni pozitive (+V), respectiv negative (-V). În acest tip de codare pot exista intervale lungi de lipsă semnal (pentru secvenţe lungi de „0"), lucru care poate duce la pierderea sincronizării.

Există şi varianta inversată a acestei codări, anume codare pseudoternar, unde lipsa semnalului simbolizează un bit de „1", iar „0" este reprezentat alternant prin potenţiale pozitive şi 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 doreşte evitarea desincronizărilor ce ar putea apare la secvenţe de "0" lungi. Acest inconvenient este combătut astfel: dacă apare un şir de 4 zerouri consecutive, ultimul bit este înlocuit cu o tensiune de aceeaşi polaritate cu a ultimului bit de "1" introdus.

Această măsură ar putea duce însă la apariţia unei componente continue semnificative. De exemplu, şirul 100000000, ar putea fi codat astfel : +000+000+. Pentru a evita asemenea situaţi, fiecare bit modificat trebuie ales de semn schimbat faţă de precedentul.

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

- dacă numărul de „1” de după ultima secvenţă modificată este par, atunci un grup de 4 zerouri consecutive se înlocuieşte cu secvenţa „+00+” în cazul în care ultimul nivel nenul de dinaintea acestei secvenţe a fost negativ, respectiv cu „-00-” în caz contrar;
- dacă numărul de „1” ce urmează ultimei secvenţe modificate este impar, atunci un grup de 4 zerouri consecutive se înlocuieşte cu secvenţa „000+” în cazul în care ultimul nivel nenul de dinaintea acestei secvenţe 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 secvenţele de 8 zerouri consecutive, cu secvenţe în care să apară tranziţii pentru a se evita astfel pierderea sincronismului.

Astfel:

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

Utilizarea acestui tip de codare va produce două modificări ale alternanţei “+-“, situaţie 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 combinaţie de 2 algoritmi de codare. Pentru a înţelege semnificaţia acestei alegeri, să considerăm pentru început alternativa simplă a schemei NRZ. Cu NRZ, o stare a semnalului reprezintă „1” binar, şi o altă stare reprezintă „0” binar. Dezavantajul acestei abordări îl reprezintă absenţa sincronizării. În schema de codare 4B/5B, codarea se face cu 4 biţi odată. Fiecare 4 biţi de semnal sunt codaţi într-un cuvânt de 5 biţi de cod. Eficienţa acestei codări este de 80%: pentru a transmite date cu un debit de 100Mbps este necesară o lăţime de bandă disponibilă de 125MHz. De remarcat că modul de transmisie al biţilor cuvântului de cod nu se specfică. Se utilizează de obicei codarea NRZI (ajungându-se la un necesar de bandă de 62.5 MHz) sau codarea MLT-3 (lăţime de bandă necesară de

31.25MHz).

Deoarece pentru a coda cele 16 combinaţii posibile de câte 4 biţi utilizăm doar 16 din cele 32 de combinaţii posibile de câte 5 biţi, rămân alte 16 grupuri care nu sunt utilizate. Cuvintele de cod sunt astfel alese încât să nu existe mai mult de două zerouri succesive şi mai puţin de două tranziţii (doi de „1”- în codaj NRZI) în cuvîntul de cod de 5 biţi. Pentru codare se foloseşte 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 secvenţa binară de intrare 10000101111. Ea este împărţită în grupuri de câte 4 biţi, care se codează conform tabelei de codare:

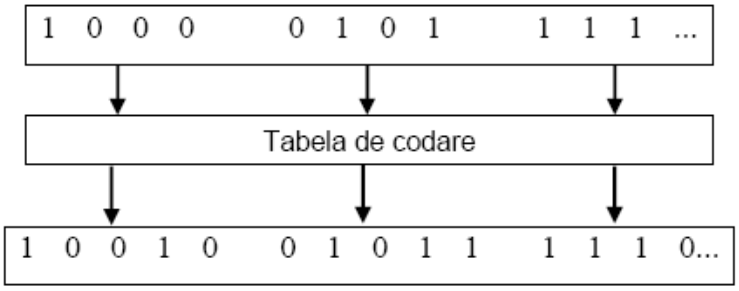

Figure 5.3: Exemplu de codare 4B/5B

Grupurile de cod care rămân nefolosite sunt fie declarate invalide, fie capătă rolul de simboluri de control a transmisiei, aşa 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 şi în transmisiile pe fire torsadate. Cablul UTP acţionează asupra semnalului transmis asemănător cu un filtru trece-jos, atenuând componentele de înaltă frecvenţă şi distorsionând astfel puternic secvenţa de date.

Principiul codării MLT este următorul:

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

Utilizări: Fast Ethernet 100BaseTx, 100BaseT4

5.2.2 Aplicaţii

Lucrarea constă în reprezentarea unui şir de biţi folosind librării js şi pachete.
Resurse folosite:

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

Pentru a instala dependințele 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:

```
<html>
    <head>
        <meta content="text/html;charset=utf-8" http-equiv="
            Content-Type">
        <title>Baseband Encoder</title>
        <link rel="stylesheet" type="text/css" href="css/style.
            css">
    </head>
    <body>
        <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"
                            v-model="bit.value"
14                      v-bind:class="[validateBit(bit.value) ? '
                            valid-input' : 'invalid-input']" />
15              </li>
16          </ul>
17          <p>Bitstream data model: <code>{{bits}}</code></p>
18          <p>Baseband code representation: <code class="encoded
              ">{{encodedBits.join('')}}</code></p>
19      </div>
20
21      <script type="text/javascript" src="libs/vue.js"></
            script>
22      <script type="text/javascript" src="libs/axios.js"></
            script>
23      <script type="text/javascript" src="js/helpers.js"></
            script>
24      <script type="text/javascript" src="js/baseband-codes.js"
            ></script>
25      <script type="text/javascript" src="js/app.js"></script>
26  </body>
27  </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:

```
var app = new Vue({
    el: '#baseband-encoder',
    data: {
        bits: [],
        encodedBits: [],
        status: '',
        numberOfBits: 8,
        validateBit: validateBit
    },
    created: function () {
        this.bits = getBitstream(this.numberOfBits);
    },
    methods: {
        encode: function(){
            this.encodedBits = getManchesterLevelEncoding(
               this.bits);
        }
    }
})
```

Fișierul **helpers.js** conține:

```
var validate = function (bits) {
        for (var i = 0; i < bits.length; i++) {
                if (this.validateBit(bits[i].value) === false
                   )
                        return false;
        }
        return true;
}

var validateBit = function (character) {
        if (character === null) return false;
        return (parseInt(character) === 0 ||
                parseInt(character) === 1);
}

function getBitstream(n) {
        result = [];
        for (var i = 0; i < n; i++) {
                let bit = { value: null };
                result.push(bit);
        }
```

```
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 && parseInt
               (bits[i - 1].value) == 1) symbol = '|__|--|';
7          if (parseInt(bits[i].value) == 0) symbol = '|--|__|';
8          if (parseInt(bits[i].value) == 0 && i > 0 && parseInt
               (bits[i - 1].value) == 0) symbol = '|--|__|';
9          result.push(symbol);
10     }
11     return result;
12 }
```

5.3 DESFĂŞURAREA LUCRĂRII

1. Considerând şirul de biţi următor: 1100100001000 să se codeze folosind toate tehnicile prezentate anterior.
2. Pentru şirul de biţi următor: 1000010000110000 să se reprezinte şirul de biţi rezultat în urma codării folosind tehnica AMI.
3. Pentru şirul de biţi următor: 1001010110100100 să se reprezinte şirul de biţi rezultat în urma codării folosind tehnica MLT.
4. Pentru şirul de biţi următor: 0000110000100001 să se reprezinte şirul de biţi rezultat în urma codării folosind tehnica 4B/5B NRZI.

BIBLIOGRAFIE

[1] Adina Aştilean. *Transmisia datelor, notiţe curs.*

[2] Ioan, Aleodor Daniel. *Transmisia datelor: consideraţii teoretice şi experimente de laborator.*