Seminarski rad

iz predmeta Teorija kodiranja

Tema: Primjena Furijeove transformacije u teoriji kodiranja

Student: Penjić Safet
PMF, Sarajevo, decembar 2010.

Sažetak

Seminarski rad se sastoji iz deset dijelova. I, II i III dio je posvećen osnovnim pojmovima iz Algebre kao što su podgrupa, red grupe, red elementa, ciklus, ciklička grupa, dekokmpozicija na klase, Galoa polje, konačno polje bazirano nad polinomijalnim prstenom i multiplikativna grupa dobijena iz GF(q). Ovi pojmovi su bitni da bi se nesmetano mogao pratiti ostatak seminarskog rada, i bez čijeg znanja se ne bi mogao napisati X dio (C++ program). U IV dijelu definišemo Reed-Solomonov kod preko matrice provjere parnosti i dokazujemo teoremu koja kaže da je Reed-Solomonov kod MDS kod. V dio je posvećen diskretnoj Furijeovoj transformaciji, koja nam u VI dijelu rada pomaže da detaljnije uronimo u osobine Reed-Solomonovog koda, i pomoću koje na osnovu matrice provjere parnosti pronalazimo generator polinom (a time i generator matricu) Reed-Solomonovog koda. U VII dijelu upoznajemo se sa pojmom LFSR-om i sa pojmom Linearne kompleksnosti niza. Ove pojmove uvodimo radi VIII dijela -Blahutove Teoreme, koja nam povezuje DFT sa Linearnom kompleksnošću niza. Ova teorema nam pomaže da u IX dijelu nađemo algoritam za dekodiranje Reed-Solomonovog koda. Važno je napomenuti i to da smo u VIII dijelu dali Berlkamp-Massey-ev algoritam za pronalaženje (jednog od) najkraćeg LFSR za dati niz.

Zadnji dio rada (X dio) je posvećen programu napisanom u C++ programskom jeziku, koji obuhvata svu teoriju napisanom u radu, a koji enkodira i dekodira datu napisanu rečenicu primjenom $RS(8, 7, \alpha, 1, 3)$ koda.

Recimo još nešto o Reed-Solomonovim (RS) kodovima. RS kodovi imaju ogromnu primjenu u digitalnoj pohrani podataka i komunikaciskim sistemima. Primjer uključuje poznati RS(255,223,33) za NASA komunikaciju u svemiru, skraćeni RS kod nad $GF(2^8)$ za CD-ROM, DVD i HDTV prenosne aplikacije, kao i prošireni RS(128,122,7) kod nad $GF(2^7)$ za modeme između ostalog.

I Podgrupa, red elementa, red grupe, ciklus, ciklička grupa, dekompozicija na klase, Galoa polje

Neka je G zajedno sa operaciom $*$ (zvjezdica) grupa i neka je H podskup skupa G. Ako G ima konačan broj elemenata, zovemo ga konačna grupa, a broj elemenata u G zovemo *red grupe* G. H zovemo *podgrupa* grupe G ako je H grupa u odnosu na operaciju $*$ (zvjezdica). Da bi dokazali da je neprazan skup H podgrupa grupe G, potrebno je samo provjeriti da je $a * b \in H$ kadgod su a i b u H (zatvorenost), i da je inverzni element svakog a u H, također u H. Ostale osobine potrebne za grupu će biti naslijeđene iz grupe G. Ako je grupa konačna, tada je čak i osobina za inverzni element automatski zadovoljena ako je osobina zatvorenosti zadovoljena, kao što ćemo ubrzo vidjeti u diskusiji cikličke podgrupe.

Kao primjer, u skupu cijelih brojeva (pozitivnih, negativnih i nule) pod operacijom sabiranja, skup parnih cijelih je podgrupa, kao i što je skup mnolžilaca broja 3.

Jedan od načina da dobijemo podgrupu H konačne grupe G je da uzmemo element $h \in G$, i definišemo da je H skup svih elemenata dobijen množenjem broja h samim sobom, proizvoljan broj puta. To jest, formiramo niz elemenata

$$h,\ h * h,\ h * h * h,\ h * h * h * h,\ ...,$$

označavajući ove elemente jednostavnije sa h, h^2, h^3, h^4,... . Kako je G konačna grupa, samo konačan broj ovih elemenata može biti različito, pa se niz eventualno mora ponoviti. Prvi element ponavljanja mora biti h, jer ako bi neka druga dva elementa h^i i h^j bila jednaka, oni bi se mogli pomnožiti sa inverznim elementom od h, pa bi iz toga dobili da su h^{i-1} i h^{j-1} također jednaka. Nastavljajući ovaj proces (ukupno $i-1$ puta) dobli bi da je $h = h^{j-i+1}$, pa bi opet imali da je prvi element ponavljanja h. Sljedeće, primjetimo da ako je $h^j = h$, tada je $h^{j-1} = 1$, jedinični element grupe. Skup h zovemo podgrupa generisana elementom h. Broj c elemenata u skupu H zovemo *red elementa* h. Skup elemenata h, h^2, h^3,..., $h^c = 1$ zovemo *ciklus*. Ciklus je podgrupa, zato što proizvod dva elementa iz ciklusa je treći elment koji je iz ciklusa, a inverz od h^i je h^{c-i}, gdje je h^{c-i} također element iz ciklusa. Grupa koja sadrži sve stepene jednog svog elementa zovemo *ciklička grupa*.

Za datu konačnu grupu G i podrupu H, postoji važna konstrukcija poznata kao *dekompozicija na klase* skupa G, koja ilustrira određenu relaciju između H i G. Elemente skupa H označimo sa h_1, h_2, h_3, ..., i izaberimo h_1 da bude jedinični element. Konstruišimo niz na sljedeći način: Prvi red sadrži elemente iz H, i to sa jediničnim elementom kao prvi element na lijevoj strani, a iza njega su drugi elementi iz H koji se pojavljuje jednom i samo jednom. Izaberimo element skupa G koji se nije pojavio u prvom redu. Nazovimo ga g_2 i iskoristimo ga kao prvi element drugog reda. Ostale elemente drugog reda ćemo dobiti tako što ćemo množiti svaki element podgrupe H (prvi red), ovim prvim elementom sa lijeve strane. Nastavljajući na ovaj način konstrukciju trećeg, četvrtog, i narednih redova, ako je moguće, svaki put izabirući prethodni nekorišten element grupe koji ćemo staviti u prvu kolonu. Ovaj proces se mora zaustaviti zato što je skup G konačan. Konačan niz izgleda ovako:

$$\begin{array}{cccccc}
h_1 = 1 & h_2 & h_3 & h_4 & ... & h_n \\
g_2 * h_1 = g_2 & g_2 * h_2 & g_2 * h_3 & g_2 * h_4 & ... & g_2 * h_n \\
g_3 * h_1 = g_2 & g_3 * h_2 & g_3 * h_3 & g_3 * h_4 & ... & g_2 * h_n \\
. & . & . & . & . & . \\
. & . & . & . & . & . \\
. & . & . & . & . & . \\
g_m * h_1 = g_m & g_m * h_2 & g_m * h_3 & g_m * h_4 & ... & g_m * h_n
\end{array}$$

Prvi element na lijevoj strani svakog reda je poznat po imenu *lider klase*. Svaki red u nizu je poznat kao *lijeva klasa*, ili jednostavnije kao klasa kad je grupa Abelova. Slično, ako je dekompozicija na klase definisana sa elementom iz G pomnožen sa desne strane, redovi našeg niza su poznati pod nazivom *desna klasa*. Dekompozicija na klase je uvijek pravougaonog oblika, sa svim popunjenim redovima, zato što je koonstruisana na taj način. Dokazat ćemo da uvijek dobijamo niz u kojem svaki element iz G se pojavljuje tačno jednom.

Teorema 1 *Svaki element grupe G pojavljuje se jednom i samo jednom u klasi dekompozicije grupe G.*

Dokaz: Svaki element pojavljuje se najmanje jednom, zato što u suprotnom, konstrukcija ne bi bila zaustavljena. Pokazat ćemo da se isti element ne može pojaviti dva puta u istom redu, a poslije toga ćemo pokazati da se isti element ne može pojaviti u dva različita reda.

Pretpostavimo da su dva člana u istom redu, $g_i * h_j$ i $g_i * h_k$ jednaka. Množeći svaki element sa g_i^{-1} dobijamo $h_j = h_k$. Ovo je kontradikcija zato što svaki element podgrupe se pojavljuje tačno jednom u prvom redu.

Pretpostavimo da su dva člana u različitim redovima, $g_i * h_j$ i $g_k * h_l$, jednaka i da je $k < i$. Množeći ove elemente sa desne strane sa h_j^{-1} dobijamo $g_i = g_k * h_l * h_j^{-1}$. Znamo da su h_l i h_j u podgrupi H a time i h_j^{-1}. Prema tome možemo zaključiti da je g_i u k-toj klasi. Ovo je u kontradikciji sa pravilom konstrukcije da lider klase ne smije biti element koji se ranije koristio ($k < i$).

Pretpostavka suprotna tvrdnji nas vodi u kontradikciju pa nije tačna. Prema tome svaki element iz G pojavljuje se jednom i samo jednom u svakoj klasi dekompozicije grupe G.

◯◁

Posljedica 2 *Ako je H podgrupa grupe G, tada broj elemenata u H djeli broje elemenata u G. Drugim riječima,*
(Red podgrupe H)(Broj klasa iz G u odnosu na H)=(Redu grupe G).

Dokaz: Slijedi odmah iz pravougaone strukture dekompozicije grupe G na klase.

◯◁

Teorema 3 (Lagrangeov teorem) *Red konačne grupe je djeljiv sa redom bilo kojeg svog elementa.*

Dokaz: Neka je G data konačna grupa. Uzmimo proizvoljni element $h \in G$, i neka je red tog elementa recimo c. Pomoću ovog elementa možemo generisati cikličku podgrupu (recimo H), pa prema prethodnoj posljedici broj elemenata u H djeli broj elemenata u G. Drugim riječima, red grupe G je djeljiv sa c. Kako je h proizvoljan element, zaključujemo: red konačne grupe je djeljiv sa redom bilo kojeg svog elementa. *q.e.d.*

◯◁

Polje $(F, \circ, *)$ u kojem je F konačan skup zovemo konačno polje ili *Galoa polje*. Galoa polje čemo označavati sa $GF(q)$. Konačna polja su daleko najvažniji algebarski sistemi u savremenoj teoriji kodiranja!

II Konačno polje bazirano nad polinomijalnim prstenom

Konačno polje može se dobiti iz polinomijalnog prstena koristeći konstrukciju koja će imitirati istu konstrukciju pomoću koje se može dobti konačno polje iz prstena cijelih. Pretpostavimo da imamo $F[x]$, prsten polinoma nad poljem F. Konstruišimo prsten količnika u $F[x]$. Izaberimo neki polinom $p(x)$ iz $F[x]$, i definišimo prsten količnika koristeći $p(x)$ kao modul za polinomijalnu aritmetiku. Ograničit ćemo diskusiju samo na monik polimome, zato što ovo ograničenje eliminiše nepotrebnu dvosmislenost u konstrukciji.

Definicija 4 *Za svaki monik polinom $p(x)$ nenula stepena nad poljem F, prsten polinoma modulo $p(x)$ je skup svih polinoma sa stepenom manjim od $p(x)$, zajedno sa polinomijalnim sabiranjem i polinomijalnim množenjem modulo $p(x)$. Ovaj prsten konvencionalno označavamo sa $F[x]/\langle p(x)\rangle$.*

Svaki element $r(x)$ iz prstena $F[x]$ može biti preslikan u $F[x]/\langle p(x)\rangle$ sa $r(x) \to R_{p(x)}[r(x)]$, gdje oznaka $R_{p(x)}[r(x)]$ označava ostatak dijeljenja polinoma $r(x)$ sa polinomom $p(x)$. Za dva elementa $a(x)$ i $b(x)$ prstena $F[x]$ koja se preslikavaju u isti element $F[x]/\langle p(x)\rangle$ kažemo da su kongruentna:

$$a(x) \equiv b(x)\,(mod\, p(x))\,.$$

Tada je $b(x) = a(x) + Q(x)p(x)$ za neki polinom $Q(x)$.

Teorem 5 *$F[x]/\langle p(x)\rangle$ je prsten.*

Dokaz: Dokaz nije težak i samo zbog dugog dokaza - pisanja, ćemo ga izostaviti.

◯◁

Primjer 6 U prstenu polinoma nad poljem $GF(2)$, izaberimo $p(x) = x^3 + 1$. Tada prsten polinoma modulo $p(x)$ je $GF(2)[x]/\langle x^3 + 1\rangle$. Ovaj prsten sadrži skup $\{0,\ 1,\ x,\ x+1,\ x^2,\ x^2+1,\ x^2+x,\ x^2+x+1\}$. Navedimo primjer množenja u ovom prstenu:

$$(x^2+1)\cdot(x^2) = R_{x^3+1}[(x^2+1)\cdot(x^2)] = R_{x^3+1}[x(x^3+1)+x^2+x] =$$
$$= x^2 + x$$

Koristili smo redukciju $x^4 = x(x^3+1) + x$.

◯◁

U sljedećoj teoremi iskoristit ću sljedeće tvrdnje koje ću ovdje samo navesti:
a) $R_{d(x)}[a(x)+b(x)] = R_{d(x)}[a(x)] + R_{d(x)}[b(x)]$
b) $R_{d(x)}[a(x)\cdot b(x)] = R_{d(x)}(R_{d(x)}[a(x)]\cdot R_{d(x)}[b(x)])$
c) $NZD(r(x), s(x)) = a(x)r(x) + b(x)s(x)$ gdje su $a(x)$ i $b(x)$ polinomi nad poljem $GF(q)$.

Teorema 7 *Prsten polinoma modulo monik polinom $p(x)$ je polje ako i samo ako je $p(x)$ prost polinom.*
Napomena: Prost polinom ćemo definisati da je oboje i monik i nesvodljiv. Da bi dobili polje, dovoljno je da $p(x)$ bude nesvodljiv, ali po našoj definiciji insistirat ćemo da

korišteni polinom bude monik.

Dokaz: " $\Rightarrow$ " : Pretpostavimo da je polinom $p(x)$ prost. Da bi pokazali da je prsten polje, moramo pokazati da svaki nenula element ima multiplikativni inverz. Neka je $s(x)$ nenula element prstena. Tada

$$stepen(s(x)) < stepen(p(x)).$$

Kako je $p(x)$ prost polimon, $NZD(s(x), p(x)) = 1$. Tada (iz posljedice Euklidovog algoritma polinome) postoje polinom $a(x)$ i $b(x)$ takvi da vrijedi

$$1 = a(x)p(x) + b(x)s(x)$$

Prema tome

$$\begin{aligned} 1 = R_{p(x)}[1] = R_{p(x)}[a(x)p(x) + b(x)s(x)] = R_{p(x)}[a(x)p(x)] + R_{p(x)}[b(x)s(x)] = \\ = R_{p(x)}[b(x)s(x)] = R_{p(x)}[R_{p(x)}[b(x)]R_{p(x)}[s(x)]] = \\ = R_{p(x)}[R_{p(x)}[b(x)]s(x)] \end{aligned}$$

Prema tome $R_{p(x)}[b(x)]$ je multiplikativni inverz za polinom $s(x)$ u prstenu polinoma modulo $p(x)$.
" $\Leftarrow$ " : Sad pretpostavimo da polinom $p(x)$ nije prost, i da ima stepen najmanje 2. Tada $p(x) = r(x)s(x)$ za neki $r(x)$ i $s(x)$, svaki od njih stepena najmanje 1. Ako je prsten polje, tada $r(x)$ ima inverzni nad operaciom množenja, koji čemo označiti sa $r^{-1}(x)$. Prema tome

$$\begin{aligned} s(x) = R_{p(x)}[s(x)] = R_{p(x)}[r^{-1}(x)r(x)s(x)] \\ = R_{p(x)}[r^{-1}(x)p(x)] = 0. \end{aligned}$$

Ovo je kontradikcija sa činjenicom da je $s(x) \neq 0$. Prema tome prsten nije polje.

◯◁

Koristeći teoriju ove lekcije, kadgod možemo naći prost polinom stepena m nad $GF(q)$, tada možemo konstruisati konačno polje sa q^m elemenata u polju. U ovoj konstrukciji, elementi su predstavljeni pomoću polinoma nad poljem $GF(q)$ stepena manjeg od m. Postoji q^m takvih polinoma, pa prema tome q^m elemenata u polju.

Primjer 8 Konstruišimo $GF(4)$ sa $GF(2)$, korištenjem primitivnog polinoma $p(x) = x^2 + x + 1$. Nesvodljivost ovog polinoma nije teško provjeriti testirajući sve moguće faktorizacije. Elementi polja su predstavljeni skupom polinoma $\{0,\ 1,\ x,\ x+1\}$. Tabele sabiranja i oduzimanja, prikazane na Slici 1 su lagano konstruisane. Naravno, jednom kad su tabele aritmetike konstruisane, polinome možemo zamjeniti drugačijim simbolima kao npr. cijelim brojevima ili nekim drugim oznakama.

+	0	1	x	x+1		·	0	1	x	x+1
0	0	1	x	x+1		0	0	0	0	0
1	1	0	x+1	x		1	0	1	x	x+1
x	x	x+1	0	1		x	0	x	x+1	1
x+1	x+1	x	1	0		x+1	0	x+1	1	x

Slika 9. Tabela aritmetike u $GF(4) = \{0,\ 1,\ x,\ x+1\}$

◯◁

III Multiplikativna grupa iz GF(q)

Nenula elementi F^* bilo kojeg polja F formiraju Abelovu grupu. U slučaju konačnog polja $GF(q)$, multiplikativna grupa $GF(q)^*$ ima red $q-1$. Iz činjenice da svaki element konačne grupe ima konačan red i generiše cikličku grupu sa ovim redom i iz činjenice da red podgrupe od konačne grupe dijeli red ove grupe (Lagrangeov teorem), slijedi da svaki element iz $GF(q)^*$ je korijen polinomijalne jednačine $x^{q-1}=1$, ili, što je ekvivalentno, svi nenula elementi od $GF(q)$ kojih ukupno ima $q-1$, su nule polinoma $x^{q-1}-1$. Ali primjetimo da polinom stepena d (gdje je $d \geq 0$) sa koeficijentima iz polja F može imati najviše d nula u F ili u proširenom polju E od F. Prema tome, β je nula polinoma $x^{q-1}-1$ (ili, ekvivalentno, $x-\beta$ je djelilac $x^{q-1}-1$) ako i samo ako je β nenula element iz $GF(q)$. Time smo dokazali sljedeću tvrdnju:

Teorema 10 (Faktorizacija od $x^{q-1}-1$) *Polinom $x^{q-1}-1$ (gdje su koeficijenti 1 i -1 elementi polja $GF(q)$) se može potpuno faktorisati u linearne faktore na sljedeći način:*

$$x^{q-1}-1=\prod_{\beta\in GF(q)^*}(x-\beta)$$

Slično, ako obe strane prethodne jednakosti pomnožimo sa $x=x-0$ dobili bi

$$x^q-x=\prod_{\beta\in GF(q)}(x-\beta)$$

Primjer 11 Posmatrajmo sljedeći proizvod u GF(5) (čiji su elementi 0, 1, 2, 3 i 4)

$$(x+1)(x+2)(x+3)(x+4)=(x+1)(x+2)(x^2+2x+2)=(x+1)(x^3+2x^2+2x+2x^2+4x+4)=$$
$$=(x+1)(x^3+4x^2+x+4)=x^4+4x^3+x^2+4x+x^3+4x^2+x+4=$$
$$=x^4+4$$

Kako je u $GF(5)$: $4=-1$, $3=-2$, $2=-3$, $1=-4$ to je

$$(x-1)(x-2)(x-3)(x-4)=x^4-1$$

◯◁

Neka je n najveći (multiplikativni) red elementa iz $GF(q)^*$. Kako red svakog elementa dijeli red elementa maksimalnog reda kad god je ovaj red konačan (ovo nije teško pokazati), slijed da (multiplikativni) red svakog elementa iz $GF(q)^*$ djeli n. Prema tome, svi elementi iz $GF(q)^*$, kojih ukupno ima $q-1$ su nule polinoma x^n-1. Kako n dijeli $q-1$ tada je sigurno $n \leq q-1$. Sa druge strane, x^n-1 ima najmanje $q-1$ različitih nula pa je $n \geq q-1$. Prema tome možemo zaključiti da je $n=q-1$. Prema tome $GF(q)^*$ sadrži element reda $q-1$ pa je prema tome ciklička grupa reda $q-1$. Bilo koji generator cikličke grupe, tj bilo koji element iz $GF(q)^*$ sa (multiplikativnim) redom $q-1$, zovemo *primitivni element* $GF(q)$. Drugim riječima primitivni element polja $GF(q)$ je element α takav da se svi elementi polja osim nule mogu izraziti kao stepen od α. Sad možemo donijeti sljedeći zaključak:

Teorema 12 (Fundamentalne osobine multiplikativne grupe iz $GF(q)$) *Grupa nenula elemenata iz $GF(q)$ pod operacijom množenja je ciklička grupa reda $q-1$. Ako je α primitivni element $GF(q)$ i $\beta=\alpha^i$, tada β ima (multiplikativni) red*

$(q-1)/NZD(q-1,i)$. *U stvari, postoji element (multiplikativnog) reda* N *u* $GF(q)^*$ *ako i samo ako je* N *pozitivan cijeli koji djeli* $q-1$.

Primjer 13 Posmatrajmo prsten svih polinoma nad poljem $GF(2)$. Razmotrimo konstrukciju $GF(8)$ koristeći prosti polinom $h(x) = x^3 + x + 1$. Znamo da je $GF(2)[x]/\langle x^3 + x + 1\rangle$ polje. Primitivni element je x, koji ću označiti sa β, pa računajući $x^i \, mod \, h(x)$ dobićemo:

binarni zapis riječi	polinomijalni zapis ($x^i \, mod \, h(x)$)	eksponencijalni zapis
000	0	-
001	1	β^0
010	x	β^1
100	x^2	β^2
011	$x+1$	β^3
110	x^2+x	β^4
111	x^2+x+1	β^5
101	x^2+1	β^6

Sa ovakvom prezentacijom elemenata $GF(8)$ množenje je lagano. Npr $\beta^4 * \beta^5 = \beta^7 * \beta^2 = \beta^2$.

◯◁

Primjer 14 Posmatrajmo konstrukciju polja $GF(16)$ koristeći prosti polinom $p(x) = x^4 + x + 1$. Ako stavimo da je $\gamma = x$ primitivni element imamo da je ($GF(16)$,$*$) ciklička grupa:

binarni zapis riječi	polinomijalni zapis ($x^i \, mod \, p(x)$)	eksponencijalni zapis
0000	0	-
0001	1	γ^0
0010	x	γ^1
0100	x^2	γ^2
1000	x^3	γ^3
0011	$x+1$	γ^4
0110	x^2+x	γ^5
1100	x^3+x^2	γ^6
1011	x^3+x+1	γ^7
0101	x^2+1	γ^8
1010	x^3+x	γ^9
0111	x^2+x+1	γ^{10}
1110	x^3+x^2+x	γ^{11}
1111	x^3+x^2+x+1	γ^{12}
1101	x^3+x^2+1	γ^{13}
1001	x^3+1	γ^{14}

Npr. Ako podjelimo x^{10} sa x^4+x+1 iz dolje napisane računice vidimo da je ostatak djeljenja x^2+x+1.

$$
\begin{array}{r}
x^{10} : (x^4+x+1) = x^6+x^3+x^2+1 \\
\underline{+\ x^{10}+x^7+x^6} \\
x^7+x^6 \\
\underline{+\ x^7+x^4+x^3} \\
x^6+x^4+x^3 \\
\underline{+\ x^6+x^3+x^2} \\
\end{array}
$$

$$\begin{array}{r} x^4+x^2 \\ \underline{+\ x^4+x+1} \\ x^2+x+1 \end{array}$$

Isto tako podjelimo x^{15} sa x^4+x+1.

$$\begin{array}{l}
x^{15} : (x^4+x+1) = x^{11}+x^8+x^7+x^5+x^3+x^2+x+1 \\
\underline{+\ x^{15}+x^{12}+x^{11}} \\
\qquad x^{12}+x^{11} \\
\qquad \underline{+\ x^{12}+x^9+x^8} \\
\qquad\quad x^{11}+x^9+x^8 \\
\qquad\quad \underline{+\ x^{11}+x^8+x^7} \\
\qquad\qquad x^9+x^7 \\
\qquad\qquad \underline{+\ x^9+x^6+x^5} \\
\qquad\qquad\quad x^7+x^6+x^5 \\
\qquad\qquad\quad \underline{+\ x^7+x^4+x^3} \\
\qquad\qquad\qquad x^6+x^5+x^4+x^3 \\
\qquad\qquad\qquad \underline{+\ x^6+x^3+x^2} \\
\qquad\qquad\qquad\quad x^5+x^4+x^2 \\
\qquad\qquad\qquad\quad \underline{+\ x^5+x^2+x} \\
\qquad\qquad\qquad\qquad x^4+x \\
\qquad\qquad\qquad\qquad \underline{+\ x^4+x+1} \\
\qquad\qquad\qquad\qquad\quad 1
\end{array}$$

◯◁

IV Reed-Solomonovi kodovi

Razmotrit ćemo konstrukciju najvažnijih linearnih kodova, najvažnijih i teoriski i praktično koji su do sad pronađeni, takovzvane Reed-Solomonovi (RS) kodovi.

Neka je dato bilo koje konačno polje $GF(q)$ sa $q \geq 3$ elemenata i neka je N djelilac od $q-1$ koji je $N \geq 2$. Uobičajen izbor je $N = q-1$ ali nekad je zanimljivo i uzeti manji N.) Neka je α element (multiplikativnog) reda N u $GF(q)$ (i primjetimo da, kao što je objašnjeno u prethodnoj lekciji, da takvo α uvijek postoji.) Neka je m_0 bilo koji cijeli broj koji zadovoljava $0 \leq m_0 < N$; m_0 je drugi parametar od najveće važnosti i često ima vrijednost 0 ili 1. Neka je d cijeli koji je $2 \leq d \leq N$. Tad *Reed-Solomonov kod* $RS(q, N, \alpha, m_0, d)$ je q-arni linearni kod dužine bloka N za koji je

$$H = \begin{bmatrix} (\alpha^{m_0})^{N-1} & (\alpha^{m_0})^{N-2} & \ldots & \alpha^{m_0} & 1 \\ (\alpha^{m_0+1})^{N-1} & (\alpha^{m_0+1})^{N-2} & \ldots & \alpha^{m_0+1} & 1 \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ (\alpha^{m_0+d-2})^{N-1} & (\alpha^{m_0+d-2})^{N-2} & \ldots & \alpha^{m_0+d-2} & 1 \end{bmatrix}$$

matrica provjere parnosti.

Primjer 15 Razmotrimo kako bi izgledala matrica za provjeru parnosti Reed-Solomonovog koda u slučaju kada je $m_0 = 0$ i $m_0 = 1$. Za $RS(q, N, \alpha, 0, d)$ kod imamo

$$H = \begin{bmatrix} 1 & 1 & \ldots & 1 & 1 \\ \alpha^{N-1} & \alpha^{N-2} & \ldots & \alpha & 1 \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ (\alpha^{d-2})^{N-1} & (\alpha^{d-2})^{N-2} & \ldots & \alpha^{d-2} & 1 \end{bmatrix}.$$

Matrica za provjeru parsnosti $RS(q, N, \alpha, 1, d)$ koda izgleda ovako:

$$H = \begin{bmatrix} \alpha^{N-1} & \alpha^{N-2} & \ldots & \alpha & 1 \\ (\alpha^2)^{N-1} & (\alpha^2)^{N-2} & \ldots & \alpha^2 & 1 \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ \cdot & \cdot & & \cdot & \cdot \\ (\alpha^{d-1})^{N-1} & (\alpha^{d-1})^{N-2} & \ldots & \alpha^{d-1} & 1 \end{bmatrix}$$

◯◁

Primjer 16 Posmatrajmo polje $GF(8)$ koje smo formirali u primjeru 13. Razmotrimo konstrukciju matrice za provjeru parnosti Reed-Solomonovog koda nad ovim poljem. Kako je dato $GF(8)$ time možemo zaključiti da je $q = 8$. Kako trebamo izabrati $N > 1$ takvo da je N djelilac od $q-1$ jedini izbor je $N = 7$. Iz primjera 13 vidimo da α ima (multiplikativni) red 7, što nam i treba. m_0 i d možemo izabrati proizvoljno tako da zadovoljavaju uslove $0 \leq m_0 < N$, $2 \leq d \leq N$. Pa izaberimo $m_0 = 1$ i $d = 3$. Matrica provjere parnosti $RS(8, 7, \alpha, 1, 3)$ koda izgleda ovako:

$$H = \begin{bmatrix} \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \\ \alpha^{12} & \alpha^{10} & \alpha^8 & \alpha^6 & \alpha^4 & \alpha^2 & 1 \end{bmatrix} = \begin{bmatrix} \alpha^6 & \alpha^5 & \alpha^4 & \alpha^3 & \alpha^2 & \alpha & 1 \\ \alpha^5 & \alpha^3 & \alpha & \alpha^6 & \alpha^4 & \alpha^2 & 1 \end{bmatrix} \bigcirc \triangleleft$$

Sljedeći problem ćemo iskoristiti u dokazu teoreme 18.

Problem 17 Vandermond matrica je kvadratna $m \times m$ matrica oblika

$$V_m = \begin{bmatrix} 1 & 1 & ... & 1 \\ \alpha_1 & \alpha_2 & ... & \alpha_m \\ \alpha_1^2 & \alpha_2^2 & ... & \alpha_m^2 \\ \alpha_1^3 & \alpha_2^3 & ... & \alpha_m^3 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{m-1} & \alpha_2^{m-1} & ... & \alpha_m^{m-1} \end{bmatrix}$$

gdje su $\alpha_1, \alpha_2, ..., \alpha_m$ elementi polja $\mathbb{F}$ (koje ne mora biti konačno). Dokazati da je

$$det(V_m) = \prod_{j=1}^{m-1} \prod_{i=j+1}^{m} (\alpha_i - \alpha_j)$$

i prema tome V_m je nesingularna matrica ako i samo ako su $\alpha_1, \alpha_2, ..., \alpha_m$ različiti.

Rješenje: Dokaz ćemo izvesti indukcijom po m. Prije toga primjetimo da ako npr imamo $m = 4$

$$V_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ \alpha_1 & \alpha_2 & \alpha_3 & \alpha_4 \\ \alpha_1^2 & \alpha_2^2 & \alpha_3^2 & \alpha_4^2 \\ \alpha_1^3 & \alpha_2^3 & \alpha_3^3 & \alpha_4^3 \end{bmatrix}$$

tada, prema tvrdnji, determinantu matrice V_4 možemo izračunati preko formule:

$$det(V_4) = \prod_{j=1}^{3} \prod_{i=j+1}^{4} (\alpha_i - \alpha_j) =$$

$$= (\alpha_2 - \alpha_1)(\alpha_3 - \alpha_1)(\alpha_4 - \alpha_1)(\alpha_3 - \alpha_2)(\alpha_4 - \alpha_2)(\alpha_4 - \alpha_3)$$

Baza indukcije:
Pokažimo da je tvrdnja tačna za $m = 1$ i $m = 2$. Za $m = 1$, imamo $V_1 = [1]$ i $det(V_1) = 1$, što odgovara praznom proizvodu. Za $m = 2$, imamo da je $det(V_2) = \alpha_2 - \alpha_1$, što je i trebalo dobiti.
Korak indukcije:
Predpostavimo da je jednakost tačna za sve k, $1 \leq k < m$, to jest da $det(V_k) = \prod_{j=1}^{k-1} \prod_{i=j+1}^{k} (\alpha_i - \alpha_j)$ vrijedi za sve k od 1 do $m - 1$. Primjetimo da ova pretpostavka uključuje i $det(V_{m-1}) = \prod_{j=1}^{m-2} \prod_{i=j+1}^{m-1} (\alpha_i - \alpha_j)$. Na osnovu ove pretpostavke pokažimo da je jednakost tačna za m.

Definišimo polinom $D(x)$ na sljedeći način:

$$D(x) = \begin{vmatrix} 1 & 1 & ... & 1 & 1 \\ \alpha_1 & \alpha_2 & ... & \alpha_m & x \\ \alpha_1^2 & \alpha_2^2 & ... & \alpha_m^2 & x^2 \\ \alpha_1^3 & \alpha_2^3 & ... & \alpha_m^3 & x^3 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{m-1} & \alpha_2^{m-1} & ... & \alpha_m^{m-1} & x^{m-1} \end{vmatrix}$$

Koristeći zadnju kolonu za razvijanje napisane determinante, možemo primjetiti da je

$$D(x) = d_{m-1}x^{m-1} + d_{m-2}x^{m-2} + ... + d_1x + d_0$$

za neke koeficijente d_0, d_1, ..., d_{m-1} gdje je npr

$$d_{m-1} = \begin{vmatrix} 1 & 1 & ... & 1 \\ \alpha_1 & \alpha_2 & ... & \alpha_m \\ \alpha_1^2 & \alpha_2^2 & ... & \alpha_m^2 \\ \alpha_1^3 & \alpha_2^3 & ... & \alpha_m^3 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{m-2} & \alpha_2^{m-2} & ... & \alpha_m^{m-2} \end{vmatrix}, \quad d_0 = \begin{vmatrix} \alpha_1 & \alpha_2 & ... & \alpha_m \\ \alpha_1^2 & \alpha_2^2 & ... & \alpha_m^2 \\ \alpha_1^3 & \alpha_2^3 & ... & \alpha_m^3 \\ \alpha_1^4 & \alpha_2^4 & ... & \alpha_m^4 \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{m-1} & \alpha_2^{m-1} & ... & \alpha_m^{m-1} \end{vmatrix}$$

Primjetimo da je

$$\begin{aligned} d_{m-1} &= det(V_{m-1}) \\ &= \prod_{j=1}^{m-2} \prod_{i=j+1}^{m-1} (\alpha_i - \alpha_j) \end{aligned} \tag{1}$$

prema indukciskoj pretpostavci. Računajući vrijednost $D(x)$ u tačkama α_1, α_2, ..., α_{m-1}, vidimo da je $D(\alpha_i) = 0$ za sve ove elemente, sobzirom da u svakom slučaju imamo dvije iste kolone u determinanti. Prema tome, pronašli smo $m-1$ korijena od $D(x)$, i kako je $stepen(D(x)) = m-1$, ovo su svi korijeni od $D(x)$. Prema tome, $D(x)$ možemo podjeliti u linearne faktore nad poljem $\mathbb{F}$ na sljedeći način:

$$D(x) = d_{m-1} \prod_{i=1}^{m-1} (x - \alpha_i)$$

Sad imamo,

$$det(V_m) = D(\alpha_m) = d_{m-1} \prod_{i=1}^{m-1} (\alpha_m - \alpha_j) = \prod_{j=1}^{m-1} \prod_{i=j+1}^{m} (\alpha_i - \alpha_j)$$

gdje zadnja napisan jednakost slijedi iz (1), što smo i trebali dobiti.

Zaključak:
Jednakost vrijedi za sve prirodne brojeve. Kako je zadnji napisani izraz nenula, V_m je nesingularna matrica ako i samo ako su svi $\alpha_1, \alpha_2, ..., \alpha_m$ različiti.

◯◁

Odredimo dimenziju K i minimalnu udaljenost d_{min} $RS(q, N, \alpha, m_0, d)$ koda.
Primjetimo da H

$$H = \begin{bmatrix} (\alpha^{m_0})^{N-1} & (\alpha^{m_0})^{N-2} & ... & \alpha^{m_0} & 1 \\ (\alpha^{m_0+1})^{N-1} & (\alpha^{m_0+1})^{N-2} & ... & \alpha^{m_0+1} & 1 \\ . & . & & . & . \\ . & . & & . & . \\ . & . & & . & . \\ (\alpha^{m_0+d-2})^{N-1} & (\alpha^{m_0+d-2})^{N-2} & ... & \alpha^{m_0+d-2} & 1 \end{bmatrix}$$

ima $d-1$ redova. Posmatrajmo kvadratnu podmatricu formiranu biranjem kolona $i_1 = N - k_1$, $i_2 = N - k_2$, ..., $i_{d-1} = N - k_{d-1}$ iz H (gdje je $N \geq k_1 > k_2 > ... > k_{d-1} > 0$). Vidimo da ova matrica ima oblik

$$\tilde{H} = \begin{bmatrix} (\alpha^{m_0})^{i_1} & (\alpha^{m_0})^{i_2} & ... & (\alpha^{m_0})^{i_{d-1}} \\ (\alpha^{m_0+1})^{i_1} & (\alpha^{m_0+1})^{i_2} & ... & (\alpha^{m_0+1})^{i_{d-1}} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ (\alpha^{m_0+d-2})^{i_1} & (\alpha^{m_0+d-2})^{i_2} & ... & (\alpha^{m_0+d-2})^{i_{d-1}} \end{bmatrix}$$

Cilj nam je sad pokazati da je ova matrica nesingularna, tj. da ima linearne nezavisne redove (ili što je ekvivalentno, linearno nezavisne kolone). Svaku kolonu ove podmatrice možemo podijeliti sa $(\alpha^{m_0})^{i_j}$ $(\neq 0)$, bez uticana na linearnu zavisnost ili nezavisnost kolona. Radeći tako, dobijamo novu podmatricu

$$\hat{H} = \begin{bmatrix} 1 & 1 & ... & 1 \\ \alpha^{i_1} & \alpha^{i_2} & ... & \alpha^{i_{d-1}} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ (\alpha^{i_1})^{d-2} & (\alpha^{i_2})^{d-2} & ... & (\alpha^{i_{d-1}})^{d-2} \end{bmatrix}$$

Koju možemo prepoznati kao Vandermonde matrica reda $d-1$. Kako α ima multiplikativni red N i kako imamo $0 < i_{d-1} < ... < i_2 < i_1 \leq N$, slijedi da su elementi α^{i_1}, α^{i_2}, ..., $\alpha^{i_{d-1}}$ različiti pa je i Vandermonde matrica nesingularna (vidjeti Problem 16). Prema tome $d-1$ proizvolojnih kololna iz H su uvijek linearno nezavisne. Ali d kolona izabranih iz H moraju biti linearno zavisne, s obzirom da ove kolone imaju samo $d-1$ komponentu. Slijedi da $d_{min} = d$ za RS kod. Naš argument iznad povlači da su redovi od H linearno nezavisni. Prema tome dimenzija K našeg RS koda zadovoljava $N - K = d - 1$ ili što je ekvivalentno, $d_{min} = N - K + 1$, što pokazuje da naš RS kod je MDS (maximum-distance-separable) kod. Rezimiramo naš pronalazak.

Teorema 18 (Parametri Reed-Solomonovog koda) *$RS(q, N, \alpha, m_0, d)$ kod je q-arni linearni (N, K) kod sa minimalnom udaljenošću $d_{min} = d$ i one je MDS, tj. $d_{min} = N - K + 1$.*

Sad želimo da pronađemo generator matricu RS koda i efikasan algoritam za dekodiranje RS koda. Za ovu svrhu, predstavit ćemo diskretnu Furijeovu transformaciju (DFT), koja će nam pomoći da detaljnije shvatimo strukturu RS koda.

V Diskretna Furije-ova transformacija

Diskretnu Furijeovu transformaciju, za prvi susret sa njom, tretirat ćemo kao aproksimaciju. Slučaj najveće zanimljivosti je kad razmatramo Furijeovu transformaciju funkcije $f \in L_1(\mathbb{R})$ ($L_1(\mathbb{R}) = \{f : \mathbb{R} \to \mathbb{C} \mid \int_R |f(x)|dx < \infty\}$) koja se ne može svesti na nešto jednsotavno, nego je neki komplikovan signal takav da

$$\widehat{f}(\omega) = \int_{-\infty}^{\infty} f(t)e^{-i\omega t}\,dt$$

se ne može izračunati u zatvorenom obliku. Za dovoljno velike $a < 0$ i $b > 0$, za bilo koju $f \in L_1(\mathbb{R})$, integral

$$\int_a^b f(t)e^{-i\omega t}\,dt$$

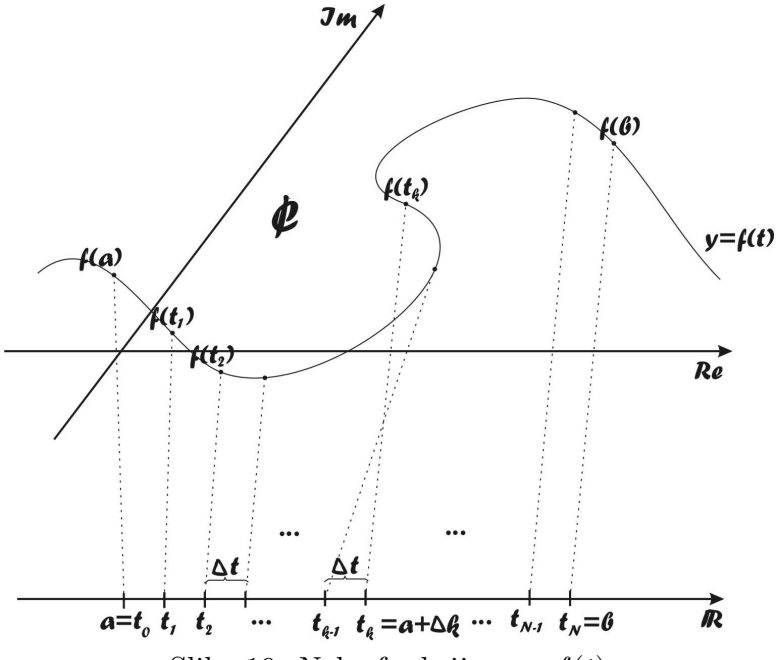
je dobra aproksimacija za $\widehat{f}$. Da bi aproksimirali ovaj integral, uzimamo uzorak na konačnom broju jednako vremenski razdvojenih tački $t_0 = a < t_1 < ... < t_N = b$, $a < 0$, $b > 0$. Neka je

$$\Delta t = \frac{b-a}{N} \quad i \quad t_k = a + k\Delta t,\ k = 0, 1, 2, ..., N.$$

Tada aproksimacija ϕ od $\widehat{f}$ je data sa

$$\phi(\omega) = \sum_{k=0}^{N-1} f(t_k)e^{-i\omega t_k}\Delta t = \sum_{k=0}^{N-1} f(t_k)e^{-i\omega(a+k\Delta t)}\Delta t = \sum_{k=0}^{N-1} f(t_k)e^{-i\omega a}e^{-i\omega k\Delta t}\Delta t$$

$$= e^{-i\omega a}\sum_{k=0}^{N-1} f(t_k)e^{-i\omega k\frac{(b-a)}{N}}\Delta t.$$



Slika 19: Neka funkcija $y = f(t)$

Dalje, izvadimo vremensko trajanje $[a, b]$ iz prikaza, fokusirajući se samo na tačke (frekvenciju)

$$\omega_n = \frac{2\pi n}{b-a}$$

gdje je n cijeli broj. U ovim tačkama

$$\phi(\omega_n) = e^{-i\omega_n a}\sum_{k=0}^{N-1} f(t_k)e^{-i\omega_n k\frac{(b-a)}{N}}\Delta t = e^{-i\omega_n a}\sum_{k=0}^{N-1} f(t_k)e^{-i\frac{2\pi n}{b-a}k\frac{(b-a)}{N}}\Delta t$$

$$= e^{-i\omega_n a}\sum_{k=0}^{N-1} f(t_k)e^{-i\frac{2\pi nk}{N}}\Delta t$$

Zanemarivanjem multiplikativne konstante $e^{-i\omega_n a}\Delta t$ i fokusiranje pažnje samo na N-periodičnu funkciju $Df : \mathbb{Z} \to \mathbb{C}$,

$$Df(n) = \sum_{k=0}^{N-1} f(t_k)e^{-i\frac{2\pi nk}{N}}, \quad n \in \mathbb{Z},$$

ili (što je ista stvar)

$$Df(n) = \sum_{k=0}^{N-1} f(t_k)\omega^{-nk}, \ gdje\ je\ \omega = e^{i\frac{2\pi}{N}}, \quad n \in \mathbb{Z}.$$

Pogledajmo sad na ove stvari sa "diskretne" perspektive. Zaboravimo da je funkcija f definisana na $\mathbb{R}$. Kako radimo samo sa vrijednostima funkcije f u konačnom broju tački, pretpostavimo da je f definisana na skupu $\{0, 1, ..., N-1\}$. Drugim riječima posmatrajmo funkciju f kao n-torku kompleksnih brojeva:
$f = (f(0), f(1), ..., f(N-1)) \in \mathbb{C}^N$. Ili razmotrimo funkciju f definisanu na cikličkoj grupi cijelih modulo pozitivni cijeli N,

$$\mathbb{Z}_N = \mathbb{Z}/N \quad (\mathbb{Z}\, modulo\, N)$$

i

$$f: \quad \mathbb{Z}_N \quad \to \quad \mathbb{C},$$
$$k + (N) \to f(k),$$

gdje je $(N) = \{kN : k \in \mathbb{Z}\}$. Druga mogučnost da posmatramo f je kao N-periodičnu funkciju na $\mathbb{Z}$ uzimajući

$$f(k + nN) = f(k) \text{ na } \mathbb{Z},\ k = 0, 1, 2, ..., N-1,\ n \in \mathbb{Z}.$$

Definicija 20 (Diskretna Furijeova transformacija) *Pretpostavimo da je konačan skup $\mathbb{Z}_N$ opremljen sa diskretnom mjerom (svi podskupovi su mjerljivi i mjera svakog podskupa je broj elemenata u njemu). Kako je $\mathbb{Z}_N$ konačan skup, svaka funkcija definisana na njemu je integrabilna.*

Diskretna Furijeova transformacija (DFT) funkcije $f : \mathbb{Z}_N \to \mathbb{C}$ je:

$$Df(n) = \sum_{k=0}^{N-1} f(k)e^{-i\frac{2k\pi n}{N}} = \sum_{k=0}^{N-1} f(k)\omega^{-kn},\ n \in \mathbb{Z}_N,\ \omega = e^{i\frac{2\pi}{N}}$$

Primjer 21 Neka je data funkcija $g : \mathbb{Z}_4 \to \mathbb{C}$ takva da $g(0, 1, 2, 3) \to (1, 2, 2, 1)$. Pronađimo DFT funkcije g. Prema definiciji imamo

$$Dg(n) = \sum_{k=0}^{3} g(k) e^{-i\frac{2k\pi n}{4}} = \sum_{k=0}^{3} g(k) e^{-i\frac{nk\pi}{2}}$$

Prema tome

$$Dg(0) = \sum_{k=0}^{3} g(k) e^{-i\frac{0\,k\pi}{2}} = g(0) + g(1) + g(2) + g(3) = 1 + 2 + 2 + 1 = 6$$

$$Dg(1) = \sum_{k=0}^{3} g(k) e^{-i\frac{1\,k\pi}{2}} = g(0)\cdot 1 + g(1)\cdot e^{-\frac{\pi}{2}} + g(2)\cdot e^{-\pi} + g(3)\cdot e^{-\frac{3\pi}{2}} = 1 - 2i - 2 + i = -1 - i$$

$$Dg(2) = \sum_{k=0}^{3} g(k) e^{-i\frac{2\,k\pi}{2}} = g(0)\cdot 1 + g(1)\cdot e^{-\pi} + g(2)\cdot e^{-2\pi} + g(3)\cdot e^{-3\pi} = 1 - 2 + 2 - 1 = 0$$

$$Dg(3) = \sum_{k=0}^{3} g(k) e^{-i\frac{3\,k\pi}{2}} = g(0)\cdot 1 + g(1)\cdot e^{-\frac{3\pi}{2}} + g(2)\cdot e^{-3\pi} + g(3)\cdot e^{-\frac{9\pi}{2}} = 1 + 2i - 2 - i = -1 + i$$

Prema tome $Dg = (6, -1 - i, 0, -1 + i)$.

◯◁

Diskretna Furijeova transformacija u konačnom polju

Bilo koji vektor $\mathbf{b} = [b_1 \;\; b_2 \;\; ... \;\; b_N] \in \mathbb{F}^N$ možemo identificirati kao polinom

$$\begin{aligned} b(X) &= b_1 X^{N-1} + b_2 X^{N-2} + ... + b_{N-1} X + b_N \\ &= \sum_{n=1}^{N} b_n X^{N-n} \end{aligned} \tag{2}$$

reda manjeg od N sa koeficijentima iz polja $\mathbb{F}$ i možemo pričati naizmjenično o vektoru $\mathbf{b}$ ili o polinomu $b(X)$. Pretpostavimo da postoji element α (multiplikativnog) reda N u polju $\mathbb{F}$. Tada *Diskretna Furijeova transformacija (DFT)* vektora iz "vremenskog-domena" $\mathbf{b}$ je vektor iz "frekvenciskog-domena" $\mathbf{B} = [B_1 \;\; B_2 \;\; ... \;\; B_N] \in \mathbb{F}^N$ definisan sa

$$B_i = b(\alpha^i), \;\; i = 1, 2, ..., N. \tag{3}$$

Koristeći (2), (3) možemo napisati eksplicitno pomoću komponenti od $\mathbf{b}$ kao

$$B_i = \sum_{n=1}^{N} b_n \cdot (\alpha^i)^{N-n}$$

što, zbog činjenice da je $\alpha^{iN} = 1$, možemo napisati u obliku

$$B_i = \sum_{n=1}^{N} b_n \alpha^{-in}, \;\; i = 1, 2, ..., N$$

što nam je već poznato iz Definicije 20.

Lagano je pokazati da je DFT prava "transformacija", tj da je inverzibilna. Pretpostavimo da su $\mathbf{b}$ i $\tilde{\mathbf{b}}$ vektori sa istom DFT, tj. da je $\mathbf{B} = \tilde{\mathbf{B}}$. Iz (3) slijedi da je α^i nula polinoma $b(X) - \tilde{b}(X)$ za $i = 1, 2, ..., N$. Dalje, polinom $b(X) - \tilde{b}(X)$ je reda manjeg od N, i nule α^1, α^2, ..., α^N ovog polinoma su sve različite. Možemo zaključiti da je $b(X) - \tilde{b}(X)$ nula polinom, tj. $b(X) = \tilde{b}(X)$, ili ekvivalentno, da je $\mathbf{b} = \tilde{\mathbf{b}}$.

Teorema 22 (Teorema inverzije) *Za funkciju $f : \mathbb{Z}_N \to \mathbb{F}$ sa DFT definisanom sa*

$$Df(i) = B_i = \sum_{n=1}^{N} f(n)\alpha^{-in} = \sum_{n=1}^{N} b_n \alpha^{-in},\ i = 1, 2, ..., N$$

$$(Df : \mathbb{Z}_N \to \mathbb{F})$$

gdje je α element iz $\mathbb{F}$ (multiplikativnog) reda N imamo

$$f(i) = b_i = \frac{1}{((N))} \sum_{i=1}^{N} Df(i)\alpha^{+ni} = \frac{1}{((N))} \sum_{i=1}^{N} B_i \alpha^{+ni}$$

gdje je $((N))$ suma N jedinica iz polja $\mathbb{F}$.

Dokaz: Da bi pokazali kako ćemo vratiti vektor $\mathbf{b}$ iz DFT $\mathbf{B}$ posmatramo polinom

$$B(X) = B_1 X^{N-1} + B_2 X^{N-2} + ... + B_{N-1} X + B_N = \sum_{i=1}^{N} B_i X^{N-i}$$

koji se može poistovjetiti sa vektorom $\mathbf{B}$ i izračunajmo ovaj polinom u $\alpha^{-n} = \alpha^{N-n} (= 1 \cdot \alpha^{-n})$. Dobićemo

$$B(\alpha^{-n}) = \sum_{i=1}^{N} B_i (\alpha^{-n})^{N-i} = \sum_{i=1}^{N} B_i \alpha^{-nN+in} = \sum_{i=1}^{N} B_i (\alpha^N)^{-n} \alpha^{in} =$$

$$= \sum_{i=1}^{N} (\sum_{j=1}^{N} b_j \alpha^{-ij}) \alpha^{in} = \sum_{i=1}^{N} b_j \sum_{j=1}^{N} (\alpha^{n-j})^i.$$

Za $j = n$, zadnja napisana suma na desnoj strani će biti samo suma od N jedinica u polju $\mathbb{F}$, što ćemo označiti simbolom $((N))$. Za $1 \le n \le N$ i $1 \le j \le N$, imamo $-N < n - j < N$ pa prema tome $\alpha^{n-j} \neq 1$ (ako je $j \neq n$). Ali, iz ovog razloga zadnja napisana suma mora biti 0, što ćemo pokazati. Neka je $\beta = \alpha^{n-j} \neq 1$ i primjetimoda je $\beta^N = 1$. Tada zadnja napisana suma S postaje

$$S = \beta + \beta^2 + ... + \beta^N = \beta(1 + \beta + ... + \beta^{N-1});$$

množeći sa $(1 - \beta)$ dobijamo

$$(1 - \beta)S = \beta(1 - \beta^N) = 0.$$

Ali $1 - \beta \neq 0$ pa je $S = 0$ kao što je i tvrđeno. Prema tome pokazali smo da se $B(\alpha^{-n}) = \sum_{i=1}^{N} b_j \sum_{j=1}^{N} (\alpha^{n-j})^i$ svodi na

$$B(\alpha^{-n}) = ((N))b_n.$$

Prema tome inverzna DFT je data sa

$$b_n = \frac{1}{((N))} B(\alpha^{-n}), \;\; n = 1, 2, ..., N.$$

Koristeći $B(X) = B_1 X^{N-1} + B_2 X^{N-2} + ... + B_{N-1} X + B_N = \sum_{i=1}^{N} B_i X^{N-i}$ zadnje napisano možemo napisati eksplicitno pomoću komponenti vektora $\mathbf{B}$ kao

$$b_n = \frac{1}{((N))} \sum_{i=1}^{N} B_i \alpha^{+ni}, \;\; n = 1, 2, ..., N$$

što je i trebalo dobiti.

◯◁

Smisao ove lekcije je da se DFT može koristiti u bilo kojem polju. Preciznije, ako možemo naći element (multiplikativnog) reda N u polju F, tada možemo definisati DFT dužine N za to polje. Iz diskusije Lekcije III slijedi da možemo podesiti DFT dužine N za konačno polje $GF(q)$ ako i samo ako je N djelilac od $q-1$.
Sljedeću teoremu ćemo iskoristiti u VI lekciji:

Teorema 23 *Neka je* $\boldsymbol{B} = (B_1, B_2, ..., B_N)$ *DFT koja odgovara vektoru* $\boldsymbol{b}$=$(b_1, b_2, ..., b_N)$. *Tada:*

a) DFT koja odgovara komponenti $b_{((n-1))}$ *je* $\alpha^{-n} B_n$;

b) DFT koja odgovara komponenti $\alpha^n b_n$ *je* $B_{((n-1))}$.

Dokaz:
a) Posmatrajmo vektore $\mathbf{b}$=$(b_1, b_2, ..., b_N)$ i $\mathbf{g}$=$(g_1, g_2, ..., g_N)$. Neka je $\mathbf{g}$=$(g_1, g_2, ..., g_N)$=$(b_N, b_1, ..., b_{N-1})$ tj. $g_n = b_{((n-1))}$. DFT komponente g_n je

$$G_n = \sum_{k=1}^{N} g_k \alpha^{-kn} = \sum_{k=1}^{N} b_{((k-1))} \alpha^{-kn} =$$

$$= b_N \alpha^{-n} + b_1 \alpha^{-2n} + ... + b_{N-1} \alpha^{-Nn} = \alpha^{-n} (b_N \cdot 1 + b_1 \alpha^{-n} + ... + b_{N-1} \alpha^{-(N+1)n})$$

$$= \alpha^{-n} \sum_{k=1}^{N} b_k \alpha^{-kn}$$

što je i trebalo dobiti.

b) Označimo sa $d_n = \alpha^{-n} b_n$. DFT od g_n je

$$D_n = \sum_{k=1}^{N} d_k \alpha^{-kn} = \sum_{k=1}^{N} \alpha^{-k} b_k \alpha^{-kn} = \sum_{k=1}^{N} b_k \alpha^{-k(n-1)} = B_{((n-1))}$$

što je i trebalo dobiti.

◯◁

Prije nego napustimo ovu lekciju, napravimo još jednu napomenu. Pretpostavimo da smo koristili (3) da bi definisali B_i za sve cijele $i \geq 0$, ne samo za $1 \leq i \leq N$. U ovom slučaju imali bi da je $B_{i+N} = b(\alpha^{i+N}) = b(\alpha^i) = B_i$ za sve $i \geq 0$, tj. niz iz frekventnog-domena $B_0, B_1, ..., B_N, ...$ će biti N-periodičan (ali njegov pravi period može biti djelilac od N). Slično, ako koristimo Teoremu inverzije da bi proširili definiciju niza iz vremenskog-domena na sve $n \geq 0$, imali bi da je
$b_{n+N} = \frac{1}{((N))} B(\alpha^{-n-N}) = \frac{1}{((N))} B(\alpha^{-n}) = b_n$ i prema tome prošireni niz iz vremenskog domena $b_0, b_1, b_2, ...,$ će također biti N-periodično. Dalje, često ćemo naći da je prikladno

pretpostaviti da imamo prošireni *jednostrano-beskonačne N-periodične nizove iz vremenskog ili frekventnog domena*

$$\underset{\sim}{\mathbf{b}} = [b_0, b_1, b_2, ...]$$

i

$$\underset{\sim}{\mathbf{B}} = [B_0, B_1, B_2, ...]$$

definisani iz $\mathbf{b}$ i $\mathbf{B}$ pomoću relacija

$$b_n = b_{n+N}, \quad za\, sve\, n \geq 0,$$
$$B_i = b_{i+N}, \quad za\, sve\, n \geq 0.$$

Zbog zadnje napisanog, slijedi da možemo pustiti svim našim indeksima iznad da idu od 0 do $N-1$ umjesto od 1 do N, bez promjene u bilo kojoj relaciji u ovoj lekciji, i u stvari ovo je obično urađeno u procesiranju digitalnih signala.

VI Generator Reed-Solomonovg koda i DFT

Iz matrice provjere parnosti H definisanoj u IV lekciji koju smo koristili za definisanje $RS(q, N, \alpha, m_0, d)$ koda, vidimo da

$$bH^T = [b(\alpha^{m_0}) \;\; b(\alpha^{m_0+1}) \;\; ... \;\; b(\alpha^{m_0+d-2})]$$

gdje je $b(X) = b_1X^{N-1} + b_2X^{N-2} + ... + b_{N-1}X + b_N$. Iz (3), na strani 15, slijedi da je

$$bH^T = [B_{m_0} \;\; B_{m_0+1} \;\; ... \;\; B_{m_0+d-2}].$$

Prema tome, *alternativna definicija* $RS(q, N, \alpha, m_0, d)$ *koda* je kao skup vektora **b** iz vremenskog domena $GF(q)^N$ čija DFT nestaje u granicama frekvencije od m_0 do $m_0 + d - 2$ uključivo.

Pokažimo da je RS kod ciklički kod. Neka je **d** kodna riječ iz RS koda. Prema Teoremi 23 (tvrdnja pod a)) ako je kodna riječ **d**= $[d_1 \; d_2 \; ... \; d_N]$ ciklički pomjerena za jedno mjesto, npr. poslije pomjeranja dobijena riječ je
e = $[e_1 \; e_2 \; ... \; e_N] = [d_N \; d_1 \; ... \; d_{N-1}]$, tada njezine komponente u frekventnom domenu D_j su pomnožene sa α^{-j}. Drugim riječima **E**= $[\alpha^{-1}D_1 \;\; \alpha^{-2}D_2 \;\; ... \;\; \alpha^{-N}D_N]$. Kako je $\alpha^{-j}D_j$ nula kadgod je D_j nula, cikličko pomjeranje od **d** je također kodna riječ. Prema tome RS kod je ciklički kod.

Za trenutak udaljimo se od RS kodova i posmatrajmo neki linearni kod C sa dužinom kodne riječi N. Izaberimo nenula kodni polinom najmanjeg stepena iz C, i označimo njegov stepen sa $N - k$ (mora biti manji od N). Pomnožimo ga sa elementom polja i načinimo ga monik polinomom. Dobijena riječ mora također biti u kodu C zato što je kod linearan. Ni jedan drugi monik polinom stepena $N - k$ nije u kodu zato što, u suprotnom, razlika dva monik polinoma bi bila u kodu i imali bi stepen manji od $N - k$, što je kontradiktorno sa našim izborom polinoma. Jedinstven nenula monik polinom najmanjeg stepena iz linearnog koda C zovemo <u>*generator polinom*</u> koda C i označavamo ga sa $g(x)$.
Navedimo dvije teoreme bez dokaza (dokaz ovih teorema se može naći u [3] na strani 101):

Teorema 24 *Ciklički kod sadrži sve množitelje generator polinom* $g(x)$ *sa polinomima stepena* $k - 1$ *i manje.*

Teorema 25 *Postoji ciklički kod blokdužine* N *sa generator polinomom* $g(x)$ *ako i samo ako* $g(x)$ *djeli* $x^N - 1$.

Prost polinom $f(x)$ najmanjeg stepena nad poljem $GF(q)$ sa osobinom $f(\beta) = 0$ se zove minimalni polinom od β nad $GF(q)$ (β je element iz proširenog polja $GF(q)$).

Vratimo se RS kodovima. Kako je RS kod ciklički kod, postoji generator polinom, $g(x)$, koji se može izračunati. Minimalni polinom nad $GF(q)$ elementa β u istom polju je $f_\beta = x - \beta$. Primjetimo da su svi minimalni polinomi prvog stepena. Prema alternativnoj definiciji RS koda, **b** je kodna riječ ako i samo ako vrijedi $b(\alpha^{m_0}) = b(\alpha^{m_0+1}) = ... = b(\alpha^{m_0+d-1}) = 0$. Primjetimo da elementi $\{\alpha^{m_0}, \alpha^{m_0+1}, ..., \alpha^{m_0+d-1}\}$ potpuno definišu nenula monik polinom $g(x) = (x - \alpha^{m_0})(x - \alpha^{m_0+1})...(x - \alpha^{m_0+d-1})$ koji je najmanjeg stepena u RS kodu (Zašto?). Drugim riječima polinom $g(x)$ je generator polinom za RS kod.

Posljedica Teoreme 24 i 25 je

Posljedica 26 *Neka polinom $g(x)$ ima red $N-k$. Ako $g(x)$ generiše linearan ciklički kod C nad $GF(2^r)$ dužine $N=2^r-1$, i dimenzije k tada*

$$G=\begin{bmatrix} g(x) \\ xg(x) \\ \vdots \\ x^{k-1}g(x) \end{bmatrix}$$

je generator matrica za C, i broj kodnih riječi u C je $(2^r)^k$.

Primjer 27 Konstruišimo $GF(8)$ korištenjem prostog polinoma $h(x)=x^3+x+1$ (vidjeti primjer 13) sa β kao primitivnim elementom. Neka je $g(x)=(x+\beta)(x+\beta^2)=x^2+(\beta+\beta^2)x+\beta^3=x^2+\beta^4x+\beta^3$. Tada $g(x)$ generiše linearni ciklički kod C nad $GF(8)$ dužine 7. Generator matrica za C je

$$G=\begin{bmatrix} 1 & \beta^4 & \beta^3 & 0 & 0 & 0 & 0 \\ 0 & 1 & \beta^4 & \beta^3 & 0 & 0 & 0 \\ 0 & 0 & 1 & \beta^4 & \beta^3 & 0 & 0 \\ 0 & 0 & 0 & 1 & \beta^4 & \beta^3 & 0 \\ 0 & 0 & 0 & 0 & 1 & \beta^4 & \beta^3 \end{bmatrix}.$$

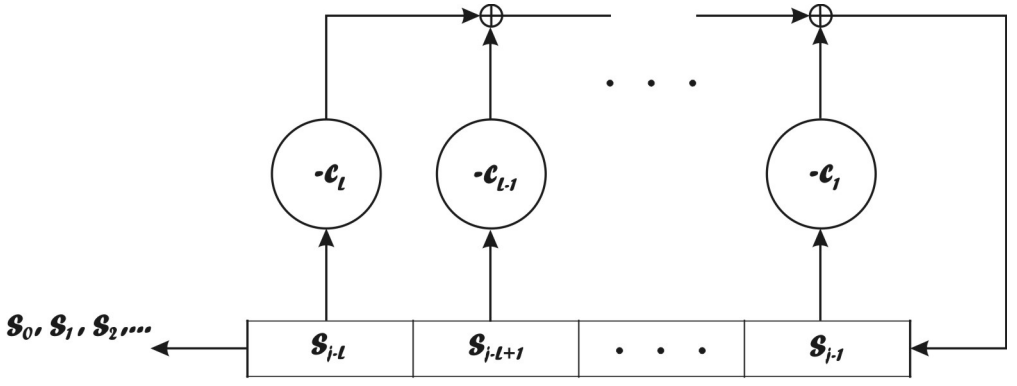
C ima 8^5 kodnih riječi. Kodna riječ koja odgovara polinomu $m(x)=\beta^3x^4+\beta x+1 \leftrightarrow \beta^3 00\beta 1=m$, na primjer je $m(x)g(x) \leftrightarrow mG=\beta^3 1\beta^6\beta\beta^4 0\beta^3$. Elementarnim transformacijama matrica G se može svesti na sljedeći oblik:

$$G'=\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \beta^4 & \beta \\ 0 & 1 & 0 & 0 & 0 & \beta^5 & \beta \\ 0 & 0 & 1 & 0 & 0 & \beta^5 & \beta^3 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & \beta^4 & \beta^3 \end{bmatrix}.$$

Zašto smo matricu G sveli na ovaj oblik?

◯◁

Linear Feedback Shift Registar je uređaj, prikazan na Slici 23, sastavljen od čelija sabiranja, množitelja sa konstantom i adresa.



Slika 28: Linear Feedback Shift Registar

Podrazumjeva se da su u LFSR inicijalno učitani brojevi s_0, s_1, ..., s_{L-1} (L je dužina LFSR), gdje su brojevi elementi nekog određenog polja $\mathbb{F}$. Množitelji sa konstantom $-c_0$, $-c_1$, ..., $-c_{L-1}$ su elementi istog polja $\mathbb{F}$. LFSR, počinje u vrementu 0, proizvodeći jednostrano-beskonačan izlazni niz

$$\underset{\sim}{\mathbf{s}} = [s_0 \;\; s_1 \;\; s_2 \;\; ...]$$

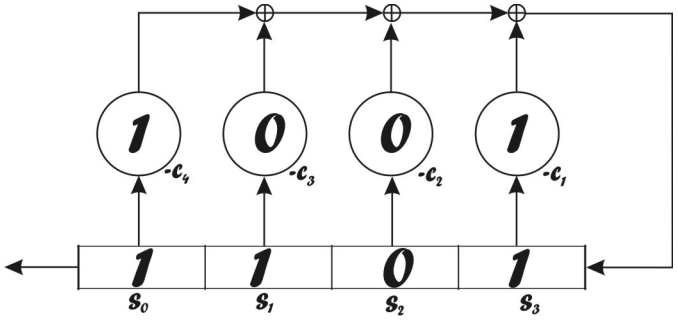
elemenata polja $\mathbb{F}$ po pravilu rekurzije

$$s_j = -c_1 s_{j-1} - c_2 s_{j-2} - ... - c_L s_{j-L} = \sum_{i=1}^{L} (-c_i) s_{j-i}, \;\; j = L, L+1, ...$$

što možemo napisati i u drugačijem obliku:

$$s_j + c_1 s_{j-1} + c_2 s_{j-2} + ... + c_L s_{j-L} = 0, \;\; j = L, L+1, ...$$

Primjer 29 Posmatrajmo LFSR u kojem množitelji sa konstantom imaju vrijednost $-\mathbf{c} = [-c_1 \;\; -c_2 \;\; -c_3 \;\; -c_4] = [1 \;\; 0 \;\; 0 \;\; 1]$, a inicijalno su učitani brojevi $\mathbf{s} = [s_1 \;\; s_2 \;\; s_3 \;\; s_4] = [1 \;\; 1 \;\; 0 \;\; 1]$. LFSR sa datim vrijednostima izgleda ovako

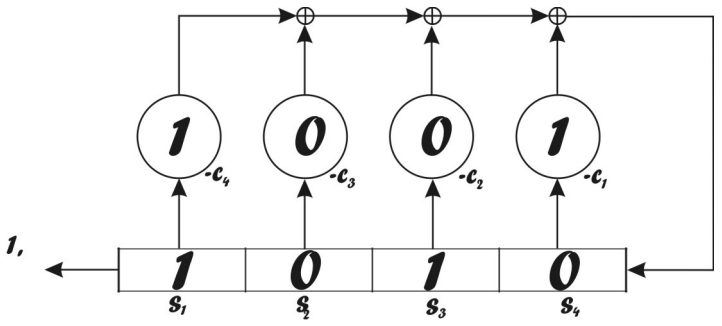


Usvojimo binarnu aritmetiku i izračunajmo ostale elemente niza.

$$s_4 = \sum_{i=1}^{4} (-c_i) s_{4-i} = (-c_1) s_3 + (-c_2) s_2 + (-c_3) s_3 + (-c4) s_0 =$$

$$= 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 1 = 0$$

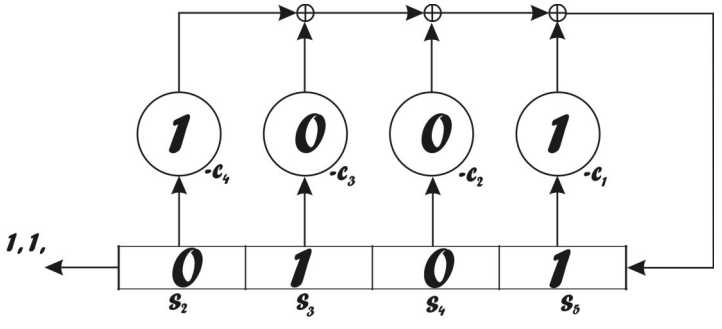
Dalje



$$s_5 = \sum_{i=1}^{4}(-c_i)s_{4-i} = (-c_1)s_4 + (-c_2)s_3 + (-c_3)s_2 + (-c4)s_1 =$$

$$= 1 \cdot 0 + 0 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 1$$

Izračunajmo još jedan član niza

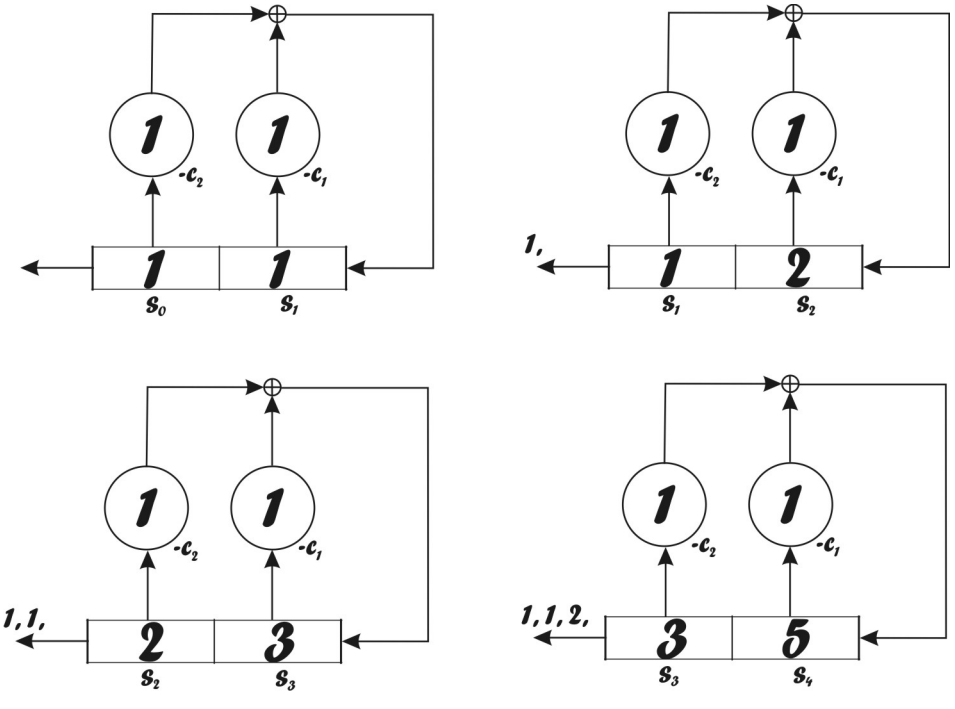


$$s_6 = \sum_{i=1}^{4}(-c_i)s_{4-i} = (-c_1)s_5 + (-c_2)s_4 + (-c_3)s_3 + (-c4)s_2 =$$

$$= 1 \cdot 1 + 0 \cdot 0 + 0 \cdot 1 + 1 \cdot 0 = 1$$

◯◁

Primjer 30 Fibonačijev niz opisan pomoću LFSR bi izgledao ovako



Pokušajmo sad naći algebarski opis izlaznog niza $\underset{\sim}{\mathbf{s}}$.

Počećemo tako što ćemo niz $\underset{\sim}{\mathbf{s}}$ poistovjetiti sa stepenim redom

$$S(D) = s_0 + s_1 D + s_2 D^2 + ... + s_j D^j + s_{j+1} D^{j+1} + ...$$

gdje je izbor za D napravljen djelom da upozori čitaoca da $S(D)$ ne mora biti polinom (može sadržavati beskonačno mnogo ne-nula članova) i djelom zato što je zgodno misliti o D kao operatoru sabiranja.

Da bi opisali LFSR, koristićemo *polinom veze* koji ćemo definisati kao

$$C(D) = 1 + c_1 D + c_2 D^2 + ... + c_L D^L.$$

Primjetimo da je red $C(D)$ najviše L, ali može biti manji. Prema tome $C(D)$ sam nije dovoljan da opiše LFSR; također treba da postavimo dužinu L eksplicitno. Pisaćemo $\langle C(D), L\rangle$ za oznaku *LFSR sa polinomom veze* $C(D)$ *i dužinom* L.

Sad primjetimo rekurziju $s_j + c_1 s_{j-1} + c_2 s_{j-2} + ... + c_L s_{j-L} = 0,\ \ j = L, L+1, ...$ određenu tako da nema ne-nula članova stepena L ili većeg u prozvodu od $C(D)$ i $S(D)$, tj. da

$$C(D)S(D) = P(D)$$

Ako ovaj proizvod raspišemo detaljno imamo:

$$C(D)S(D) = (1 + c_1 D + c_2 D^2 + ... + c_L D^L)(s_0 + s_1 D + s_2 D^2 + ...s_L D^L + ...) =$$

$$= s_0 + (c_1 s_0 + s_1)D + (c_2 s_0 + c_1 s_1 + s_2)D^2 + ... + (c_{L-1} s_0 + c_{L-2} s_1 + ... + s_{L-1})D^{L-1} +$$
$$\underbrace{(c_L s_0 + c_{L-1} s_1 + ... + s_L)}_{=0} D^L + \underbrace{(c_{L+1} s_1 + c_L s_2 + ... + s_{L+1})}_{=0} D^{L+1} + ...$$

$P(D)$ je polinom stepena strogo manjeg od L. Napišimo

$$P(D) = p_0 + p_1 D + ... + p_{L-1} D^{L-1}$$

i izjednačimo članove stepena i, $i < L$, sa obe strane jednakosti $C(D)S(D) = P(D)$ čime dobijamo matričnu jednakost

$$\begin{bmatrix} p_0 \\ p_1 \\ . \\ . \\ . \\ p_{L-1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & ... & ... & 0 \\ c_1 & 1 & ... & 0 & 0 \\ . & . & ... & & . \\ . & ... & . & & . \\ c_{L-2} & & ... & . & 0 \\ c_{L-1} & c_{L-2} & ... & c_1 & 1 \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ . \\ . \\ . \\ s_{L-1} \end{bmatrix}$$

što pokazuje da za svaki izbor od $P(D)$ postoji jedinstveno odgovarajuće inicijalno stanje $[s_0\ \ s_1\ \ ...\ \ s_{L-1}]$ od LFSR. Sumirajmo naš pronalazak.

Teorema 31 *LFSR* $\langle C(D), L\rangle$ *može proizvesti jednostrano-beskonačan izlazni niz* $\underset{\sim}{\boldsymbol{s}}$ *ako i samo ako se stepeni red* $S(D)$ *može napisati u obliku*

$$S(D) \ = \ \frac{P(D)}{C(D)} \tag{4}$$

gdje je $P(D)$ polinom stepena strogo manjeg od L.

Primjer 32 Neka je dato polje $\mathbb{F}_3[x]$, svih polinoma nad poljem $GF(3)$. Neka je

$$f(x) = 1, \;\; g(x) = 2 \cdot x^4 + x + 1.$$

Tada možemo konstruisati LFSR koji će proizvesti jednostrano beskonačan niz čiji su članovi koeficijenti polinoma $h(x)$, drugim riječima

$$h(x) = \frac{f(x)}{g(x)} = 1 + 2 \cdot x + x^2 + 2 \cdot x^3 + 2 \cdot x^4 + x^6 + x^7 + x^8 + ...$$

Kako trebamo izračunati $\frac{f(x)}{g(x)}$ to je polinom veze $g(x)$. Iz polinoma veze nije teško dobiti $LFSR$. Problem predstavlja inicijalno stanje LFSR. Inicijalno stanje dobijemo na sljedeći način. Primjetimo da je

$$h(x) = h_0 + h_1 x + h_2 x^2 + ...$$

Uvedimo oznaku $p(x) = h_0 + h_1 x + h_2 x^2 + h_3 x^3$, pa sad imamo

$$h(x) = p(x) + h_4 x^4 + h_5 x^5 + ...$$

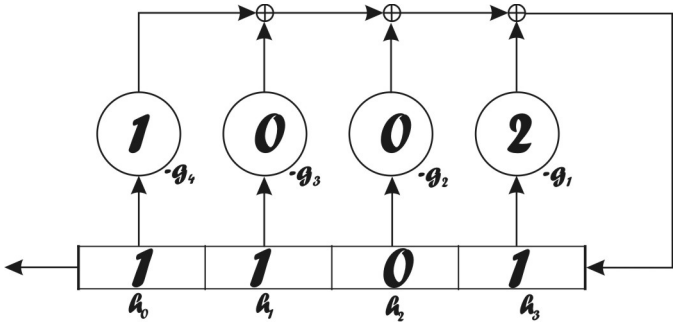
Kako želimo iskoristiti rekurzivnu formulu

$$h_j + g_1 h_{j-1} + g_2 h_{j-2} + g_3 h_{j-3} + g_4 h_{j-4} = 0, \;\; j = 4, 5, ...$$

primjetimo da je tada

$$h(x)g(x) = p(x)g(x) = 1$$

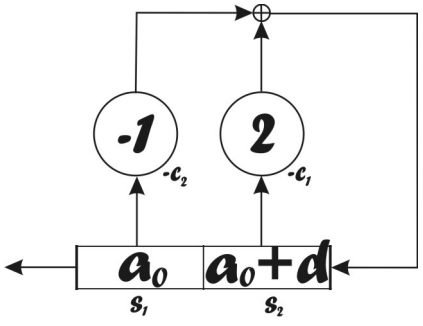
Iz čega dobijemo da su $h_0 = 1$, $h_1 = 2$, $h_2 = 1$, $h_3 = 2$. Početno stanje LFSR koji proizvodi koeficijente 2, 1, 2, 2, 0, 1, 1, 1, 2, 2, 2, 2, 0, 2, 0, 2, 1, 1, 2, 0, 1... izgleda



◯◁

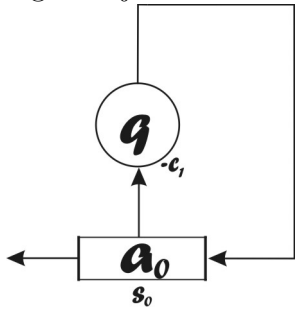
Linearna kompleksnost jednostrano-beskonačnog niza $\underset{\sim}{\mathbf{s}}$, što ćemo označiti sa $L(\underset{\sim}{\mathbf{s}})$, je najmanji L takav da niz $\underset{\sim}{\mathbf{s}}$ može biti proizveden sa LFSR dužine L, i on je ∞ ako takav $LFSR$ ne postoji. Po dogovoru, za sve nula nizove $\underset{\sim}{\mathbf{0}} = [0 \;\; 0 \;\; 0 \;\; ...]$ kažemo da imaju linearnu kompleksnost 0, tj. $L(\underset{\sim}{\mathbf{0}}) = 0$. Također, po dogovoru definišemo linearnu kompleksnost konačnog niza, recimo, $\underset{\sim}{\mathbf{s}}^{(n)} = [s_0 \;\; s_1 \;\; s_2 \;\; ...s_{n-1} \;]$. Linearna kompleksnost od $\underset{\sim}{\mathbf{s}}^{(n)}$ se definiše kao najmanja linearna kompleksnost od svih jednostrano-beskonačnih nizova koji imaju $\underset{\sim}{\mathbf{s}}^{(n)}$ kao prefiks. Ekvivalentno, $L(\underset{\sim}{\mathbf{s}}^{(n)})$ je dužina najkraćeg LFSR $\langle C(D), L\rangle$ koji može proizvesti $\underset{\sim}{\mathbf{s}}^{(n)}$ kao niz svojih prvih brojeva (gdje naravno inicijalno stanje mora biti prvih L brojeva od $\underset{\sim}{\mathbf{s}}^{(n)}$).

Primjer 33
Linearna kompleksnost aritmetičkog niza je 2.



◯◁

Primjer 34
Linearna kompleksnost geometriskog niza je 1.



◯◁

Sljedeća teorema je direktna posljedica našeg opisa iz Teoreme 29. LFSR izlaznog niza i naše definicije polinoma veze $C(D)$ kao polinoma za koji vriiedi $C(0) = 1$.

Teorema 35 (Linearna kompleksnost jednostrano beskonačnog niza) *Ako se stepeni red $S(D)$ jednostrano beskonačnog niza $\underset{\sim}{s}$ može napisati u obliku*

$$S(D) = \frac{P(D)}{C(D)}$$

gdje su $P(D)$ i $C(D)$ relativno prosti polinomi (tj. nemaju zajedničkog faktora stepena 1 ili većeg) i $C(D) = 1$, tada imamo

$$L(\underset{\sim}{s}) = max\{stepen(C(D)), 1 + stepen(P(D))\}.$$

Štaviše, $C(D)$ je polinom veze jedinstvenog LFSR dužine $L = L(\underset{\sim}{s})$ koji može proizvesti $\underset{\sim}{s}$.

U praksi, naravno radimo samo sa konačnim nizovima. Prema tome, sljedeći rezultat je od velike važnosti.

Teorema 36 (Linearna kompleksnost konačnog niza) *Ako je $L(\underset{\sim}{s}) = L > 0$ i ako $\underset{\sim}{s}^{(n)}$ označava prvih n brojeva od $\underset{\sim}{s}$ tada*

$$L(\underset{\sim}{s}^{(n)}) = L, \ za\, sve\, n \geq 2L$$

Štaviše, jedinstvenost od LFSR dužine L koji može proizvesti $\underset{\sim}{s}$ je također jedinstven LFSR dužine L koji može proizvesti $\underset{\sim}{s}^{(n)}$ za svaki $n \geq 2L$.

Dokaz: Pretpostavimo da $\langle C_1(D), L_1\rangle$ i $\langle C_2(D), L_2\rangle$, gdje su $L_1 \leq L$ i $L_2 \leq L$,

proizvode $\underset{\sim}{\mathbf{s}}^{(n)}$ za neko $n \geq 2L$. Jednakost (4) tada određuje polinome $P_1(D)$ i $P_2(D)$, sa stepenom manjim od L_1 i L_2, redom, takvim da

$$\frac{P_1(D)}{C_1(D)} - \frac{P_2(D)}{C_2(D)} = D^n \Delta(D)$$

gdje je $\Delta(D)$ neki stepeni red, što slijedi iz činjenice da nizovi koji su proizvedeni se ova dva LFSR imaju $\underset{\sim}{\mathbf{s}}^{(n)}$ kao prefiks i pa se razlikuju u vremenu n najranije. Množeći sa $C_1(D)C_2(D)$ prethodno napisanu jednakost dobićemo

$$P_1(D)C_2(D) - P_2(D)C_1(D) = D^n \Delta(D) C_1(D) C_2(D).$$

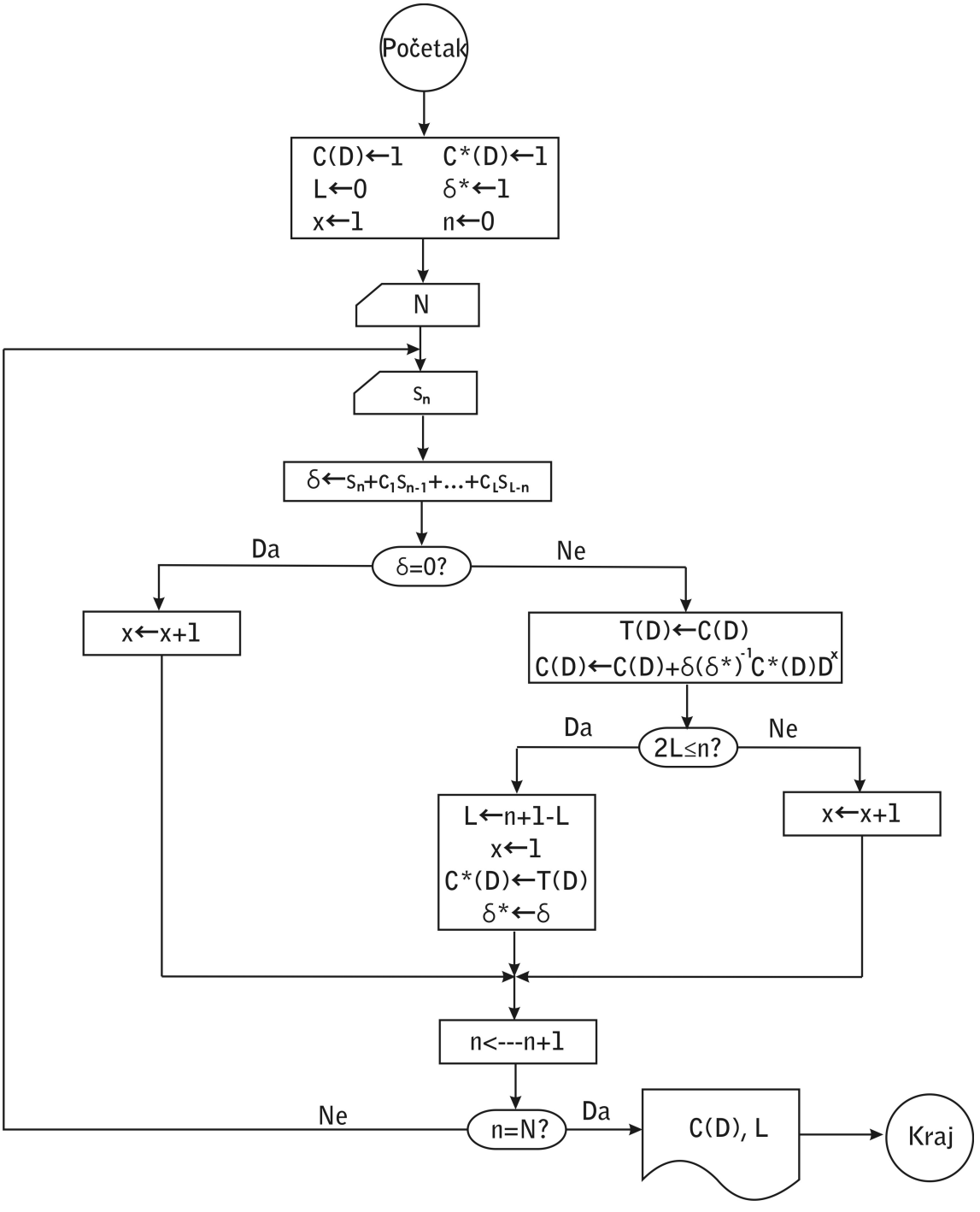
Lijeva strana prethodne jednakosti ima stepen manji od $L_1 + L_2 \leq 2L$, dok desna strana nema članova čiji je stepen manji od n. Kako je $n \geq 2L$ ovo je moguće samo u slučaju kada su obe strane jednake 0, pa prema tome

$$\frac{P_1(D)}{C_1(D)} = \frac{P_2(D)}{C_2(D)}.$$

Slijedi da oba LFSR proizvode isti jednostrano-beskonačan niz. Prema tome, pokazali smo da svi LFSR-i dužine L ili manje, koji proizvode $\underset{\sim}{\mathbf{s}}^{(n)}$, proizvode isti jednostrano-beskonačan niz, koji onda mora biti niz $\underset{\sim}{\mathbf{s}}$, zato što znamo da se $\underset{\sim}{\mathbf{s}}$ može proizvesti pomoću nekog LFSR dužine L. Prema tome, bilo koji LFSR dužine L ili manje koji proizvodi $\underset{\sim}{\mathbf{s}}$ mora biti u stvari jedinstveni LFSR dužine L koji proizvodi $\underset{\sim}{\mathbf{s}}$, pa je $L(\underset{\sim}{\mathbf{s}}^{(n)}) = L$.

◯◁

Postoji efikasan algoritam za pronalaženje (jednog od) najkraćeg LFSR-a koji proizvodi određen niz $\underset{\sim}{\mathbf{s}}^{(n)}$ dužine n. Ovaj
LFSR sintesa algoritam (ili Berlekamp-Massey algoritam, kako se često zove) je dat u obliku dijagrama toka na Slici 37. Dokaz da algoritam će uistinu rješiti problem nalaženja (jednog od) najkraćeg LFSR-a koji proizvodi $\underset{\sim}{\mathbf{s}}^{(n)}$ se može naći u [5].



Slika 37: LFSR Sinteza Algoritam (Berlkamp-Massey Algoritam) za pronalaženje (jednog od) najkraćeg LFSR koji može generisati niz $s_0, s_1, \ldots s_{N-1}$.

VIII Blahut-ova teorema - Linearna kompleksnost i DFT

Sad ćemo pokazati veoma zanimljivu vezu između linearne kompleksnosti i diskretne Furijeove transformacije, naime Haming težina u jednom domenu jednaka je linearnoj kompleksnosti u drugom.

Teorema 38 (Blahut-ova teorema) *Ako je* $\boldsymbol{B} \in \mathbb{F}^N$ *DFT od vektora* $\boldsymbol{b} \in \mathbb{F}^N$*, i ako su* $\underset{\sim}{\boldsymbol{b}}$ *i* $\underset{\sim}{\boldsymbol{B}}$ *jednostrano beskonačni nizovi* $\underset{\sim}{\boldsymbol{b}}$= $[b_0\ \ b_1\ \ b_2...]$, $\underset{\sim}{\boldsymbol{B}}$= $[B_0\ \ B_1\ \ B_2...]$ *koji su definisani iz* $\boldsymbol{b}$ *i* $\boldsymbol{B}$ *relacijom* $b_n = b_{n+N}$ *i* $B_i = B_{i+N}$ *za svako* $n \geq 0$ *i* $i \geq 0$ *(ili drugačije napisano* $\underset{\sim}{\boldsymbol{B}} = (\boldsymbol{B}, \boldsymbol{B}, \boldsymbol{B}, ...) = (B_0, B_1, B_2, ..., B_N, ..., B_{i+N}, ...)$ *i* $\underset{\sim}{\boldsymbol{b}} = (\boldsymbol{b}, \boldsymbol{b}, \boldsymbol{b}, ...) = (b_0, b_1, b_2, ..., b_N, ..., b_{i+N}, ...)$ *) tada*

$$L(\underset{\sim}{\mathbf{B}}) = w(\mathbf{b})$$

i

$$w(\mathbf{B}) = L(\underset{\sim}{\mathbf{b}}).$$

Dokaz: Ako bi bilo $\mathbf{b} = 0$, tvrdnja je trivijalna„ pa pretpostavimo da je $\mathbf{b} \neq 0$, pa prema tome i $\mathbf{B} \neq 0$. Koristeći definiciju DFT $B_i = \sum_{n=1}^{N} b_n \alpha^{-in},\ i = 1, 2, ..., N$, stepeni red za $\underset{\sim}{\mathbf{B}}$ možemo napisati kao

$$B(D) = \sum_{i=0}^{\infty} B_i D^i = \sum_{i=0}^{\infty} (\sum_{n=1}^{N} b_n \alpha^{-in}) D^i$$

$$= \sum_{n=1}^{N} b_n \sum_{i=0}^{\infty} (\alpha^{-n} D)^i$$

ili

$$B(D) = \sum_{n=1}^{N} b_n \frac{1}{1 - \alpha^{-n} D} =$$

$$= \frac{b_1}{1 - \alpha^{-1} D} + \frac{b_2}{1 - \alpha^{-2} D} + ... + \frac{b_N}{1 - \alpha^{-N} D}$$

Kako α, koja definiše DFT, ima (multiplikativni) red N, slijedi da su α^{-1}, α^{-2}, ..., α^{-N} različiti. Prema tome, desna strana zadnje jednakosti je odgovarajuća parcijalni razlomak sa $w(\mathbf{b})$ članova pa je

$$B(D) = \frac{P(D)}{C(D)}$$

gdje su $P(D)$ i $C(D)$ relativni prosti polinomi takvi da je
$stepen(P(D)) < stepen(C(D)) = w(\mathbf{b})$ i

$$C(D) = \prod_{n=1\, i\, b_n \neq 0}^{N} (1 - \alpha^{-n} D).$$

Iz ove jednakosti slijedi da je $C(0) = 1$ pa prema tome $\langle C(D), L = w(\mathbf{b}) \rangle$ je jedinstveni najkraći LFSR koji proizvodi $\underset{\sim}{\mathbf{B}}$. U stvari, pokazali smo da je L($\underset{\sim}{\mathbf{B}}$)= $w(\mathbf{b})$. Dokaz da je $w(\mathbf{B}) = L(\underset{\sim}{\mathbf{b}})$ je potpuno sličan.

◯◁

IX Dekodiranje Reed-Solomonovog koda

Sada je jednostavno formulirati algoritam dekodiranja za naš RS kod koji će ispraviti sve uzorke od t ili manje greški, gdje je $2t < d = d_{min}$. Neka je $\mathbf{b}$ tranzmitovana riječ i neka je $\mathbf{e}$ napravljeni uzorak greške, tako da je

$$\mathbf{r} = \mathbf{b} + \mathbf{e}$$

primljena riječ. Uzimajući DFT dobijamo

$$R_i = B_i + E_i, \ za\, sve\, i.$$

Ali kako je $B_i = 0$ za $i = m_0, m_0 + 1, \ldots, m_0 + d - 2$ imamo

$$R_i = E_i, \ \ m_o \leq i \leq m_0 + d - 2.$$

Možemo primjetiti da je $(R_{m_0}, R_{m_0+1}, ..., R_{m_0+d-2})$ sindrom greške od $\mathbf{r}$ u odnosu na matricu provjere parnosti H RS koda, tj računanje sindroma je operacija računanja $d-1$ članova DFT od $\mathbf{r}$. Sad možemo primjetiti da znamo prvih $d-1$ članova jednostrano-beskonačnog niza

$$\underset{\sim}{\mathbf{E}}' = [E_{m_0} \ \ E_{m_0+1} \ \ \ldots \ \ E_{m_0+d-2} \ \ E_{m_0+d-1} \ \ E_{m_0+d} \ \ \ldots]$$

gdje "prim" na $\underset{\sim}{\mathbf{E}}'$ nas podsjeća da ovaj niz nije isti kao $\underset{\sim}{\mathbf{E}}$ čiji je prvi član E_0. Ali $\underset{\sim}{\mathbf{E}}$ je periodičan niz pa je $L(\underset{\sim}{\mathbf{E}}')$=$L(\underset{\sim}{\mathbf{E}})$ s obzirom da svaki od ovih jednostrano-beskonačnih periodičnih nizova sadrži drugi kao podniz. Dalje, Blahutova teorema nam govori da je $L(\underset{\sim}{\mathbf{E}})$= $w(\mathbf{e})$, iz čega možemo zaključiti da je

$$L(\underset{\sim}{\mathbf{E}}') = w(\mathbf{e}).$$

Iz naših rezultata o Linearnoj kompleksnosti konačnih nizova, slijedi da ako je

$$2w(\mathbf{e}) \leq d - 1$$

tada možemo pronaći najkraći LFSR koji proizvodi $\underset{\sim}{\mathbf{E}}'$ primjenjujući LFSR sinteza algoritam na prvih $d-1$ poznatih članova od $\underset{\sim}{\mathbf{E}}'$. Poslije toga možemo koristiti ovaj LFSR zajedno sa poznatim članovima od $\underset{\sim}{\mathbf{E}}'$ da bi proizveli N-torku $\mathbf{E}'$ koja je samo cikličko pomjeranje od $\mathbf{E}$. Poznavajući $\mathbf{E}$, možemo pronaći $\mathbf{e}$ pomoću inverzne DFT i tad povratiti kodnu riječ $\mathbf{b}$ kao **r-e**. S druge pak strane, ako dobijemo da je $2w(\mathbf{e}) \geq d$ ovaj algoritam će dati netačan rezultat; ovu mogučnost ćemo označiti stavljajući "kapu" na izračunati iznos, koji će *garantovano biti tačan samo kad stvarna greška ima Hamming težinu koja zadovoljava* $2w(\boldsymbol{e}) < d = d_{min}$. Sumirajmo ovaj efikasni algoritam dekodiranja.

1. korak: Izračunati E_i za $m_0 \leq i \leq m_0 + d - 2$ tako što ćemo izračunati DFT primljene riječi $\mathbf{r}$ nad ovom granicom frekvencije.

2. korak: Koristeći LFSR sinteza algoritam pronaći (jedan od) najkraćih LFSR-a $\langle C(D), L\rangle$ koji će proizvesti konačan niz E_{m_0}, E_{m_0+1}, ..., E_{m_0+d-2}. Ako dobijemo da je $2L \geq d$, stani i emitiraj **?**, tj. objavi da je pronađen uzorak greške.

3. korak: Učitaj LFSR $\langle C(D), L\rangle$ sa zadnjih L brojeva niza E_{m_0}, E_{m_0+1}, ..., E_{m_0+d-2} i pokreni LFSR $N - (d-1)$ put da bi proizveli
$\hat{\mathbf{E}}' = [E_{m_0} \ \ E_{m_0+1} \ \ \ldots \ \ E_{m_0+d-2} \ \ \hat{E}_{m_0+d-1} \ \ \hat{E}_{m_0+N-1}]$.

4. korak: Ciklički, odgovarajuće, pomjerimo $\hat{\mathbf{E}}'$ (u zavisnosti od vrijednosti m_0) da bi dobili $\hat{\mathbf{E}} = [\hat{E}_1 \quad \hat{E}_2 \quad ... \quad \hat{E}_N]$.

5. korak: Izračunati inverznu DFT $\hat{\mathbf{e}}$ od $\hat{\mathbf{E}}$.

6. korak: Oduzmimo $\hat{\mathbf{e}}$ od $\mathbf{r}$ da bi dobili dekodirajuću odluku $\hat{\mathbf{b}}$ za tranzmitovanu riječ.

X Program napisan u C++ za enkodiranje i dekodiranje $RS(8, 7, \alpha, 1, 3)$ koda koji obuhvata svu teoriju napisanu u prethodnim lekcijama

```
#include <iostream>
#include <iomanip>
#include <fstream>
using namespace std;

/*
Podprogram za zapis slova u oktalnom obliku duđine 5
"a" æu oznaèiti sa 13645
"b" æu oznaèiti sa 13642
"c" æu oznaèiti sa 13655
"d" æu oznaèiti sa 13652
"e" æu oznaèiti sa 13745
"f" æu oznaèiti sa 13742
"g" æu oznaèiti sa 13755
"h" æu oznaèiti sa 13752
"i" æu oznaèiti sa 15645
"j" æu oznaèiti sa 15642
"k" æu oznaèiti sa 15655
"l" æu oznaèiti sa 15652
"m" æu oznaèiti sa 15745
"n" æu oznaèiti sa 15742
"o" æu oznaèiti sa 15755
"p" æu oznaèiti sa 15752

"q" æu oznaèiti sa 33645
"r" æu oznaèiti sa 33642
"s" æu oznaèiti sa 33655
"t" æu oznaèiti sa 33652
"u" æu oznaèiti sa 33745
"v" æu oznaèiti sa 33742
"w" æu oznaèiti sa 33755
"x" æu oznaèiti sa 33752
"y" æu oznaèiti sa 35645
"z" æu oznaèiti sa 35642
" " æu oznaèiti sa 35655
"1" æu oznaèiti sa 35652
"2" æu oznaèiti sa 35745
"3" æu oznaèiti sa 35742
"4" æu oznaèiti sa 35755
"5" æu oznaèiti sa 35752
*/
void zapisi_u_oktalnom_obliku_slovo(char slovo, int izlazni_niz[15])
{
     int pauza;
     if (slovo=='a')
     {
                    izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=6;
                    izlazni_niz[4]=4; izlazni_niz[5]=5;
     }
     else if (slovo=='b')
     {
                    izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=6;
                    izlazni_niz[4]=4; izlazni_niz[5]=2;
     }
     else if (slovo=='c')
     {
                    izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=6;
                    izlazni_niz[4]=5; izlazni_niz[5]=5;
     }
     else if (slovo=='d')
     {
                    izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=6;
                    izlazni_niz[4]=5; izlazni_niz[5]=2;
     }
     else if (slovo=='e')
     {
                    izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=7;
                    izlazni_niz[4]=4; izlazni_niz[5]=5;
     }
     else if (slovo=='f')
```

```
{
                izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
}
else if (slovo==’g’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
}
else if (slovo==’h’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
}
else if (slovo==’i’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
}
else if (slovo==’j’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
}
else if (slovo==’k’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
}
else if (slovo==’l’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
}
else if (slovo==’m’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
}
else if (slovo==’n’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
}
else if (slovo==’o’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
}
else if (slovo==’p’)
{
                izlazni_niz[1]=1; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
}

else if (slovo==’q’)
{
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
}
else if (slovo==’r’)
{
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
}
else if (slovo==’s’)
{
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
}
else if (slovo==’t’)
{
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
}
else if (slovo==’u’)
{
```

```
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
    }
    else if (slovo=='v')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
    }
    else if (slovo=='w')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
    }
    else if (slovo=='x')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=3; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
    }
    else if (slovo=='y')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
    }
    else if (slovo=='z')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
    }
    else if (slovo==' ')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
    }
    else if (slovo=='1')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=6;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
    }
    else if (slovo=='2')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=5;
    }
    else if (slovo=='3')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=4; izlazni_niz[5]=2;
    }
    else if (slovo=='4')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=5;
    }
    else if (slovo=='5')
    {
                izlazni_niz[1]=3; izlazni_niz[2]=5; izlazni_niz[3]=7;
                izlazni_niz[4]=5; izlazni_niz[5]=2;
    }
    else
    {
    cout<<"Program nece raditi isravno. U fajl ulaz kucati samo mala slova i brojeve od 1 do 5.";
    cin>>pauza;
    }
}

/*
Podprogram za dekodiranje selova
13645 dekodiraj kao "a"
13642 dekodiraj kao "b"
13655 dekodiraj kao "c"
13652 dekodiraj kao "d"
13745 dekodiraj kao "e"
13742 dekodiraj kao "f"
13755 dekodiraj kao "g"
13752 dekodiraj kao "h"
15645 dekodiraj kao "i"
15642 dekodiraj kao "j"
```

```
15655 dekodiraj kao "k"
15652 dekodiraj kao "l"
15745 dekodiraj kao "m"
15742 dekodiraj kao "n"
15755 dekodiraj kao "o"
15752 dekodiraj kao "p"

33645 dekodiraj kao "q"
33642 dekodiraj kao "r"
33655 dekodiraj kao "s"
33652 dekodiraj kao "t"
33745 dekodiraj kao "u"
33742 dekodiraj kao "v"
33755 dekodiraj kao "w"
33752 dekodiraj kao "x"
35645 dekodiraj kao "y"
35642 dekodiraj kao "z"
35655 dekodiraj kao " "
35652 dekodiraj kao "1"
35745 dekodiraj kao "2"
35742 dekodiraj kao "3"
35755 dekodiraj kao "4"
35752 dekodiraj kao "5"
*/
char dekodiraj_u_slovo(int ulazni_niz[15])
{
     if(ulazni_niz[1]==1)
     {
           if(ulazni_niz[2]==3)
           {
                if(ulazni_niz[3]==6)
                {
                    if(ulazni_niz[4]==4)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return 'a';
                            }
                            else if(ulazni_niz[5]==2)
                            {
                                    return 'b';
                            }
                            else return '?';
                    }
                    else if(ulazni_niz[4]==5)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return 'c';
                            }
                            else if(ulazni_niz[5]==2)
                            {
                                    return 'd';
                            }
                            else return '?';
                    }
                    else return '?';
                }
                else if(ulazni_niz[3]==7)
                {
                    if(ulazni_niz[4]==4)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return 'e';
                            }
                            else if(ulazni_niz[5]==2)
                            {
                                    return 'f';
                            }
                            else return '?';
                    }
                    else if(ulazni_niz[4]==5)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return 'g';
```

```
                    }
                    else if(ulazni_niz[5]==2)
                    {
                       return 'h';
                    }
                    else return '?';
                }
                else return '?';
            }
            else return '?';
        }
        else if(ulazni_niz[2]==5)
       {
            if(ulazni_niz[3]==6)
            {
                if(ulazni_niz[4]==4)
                {
                        if(ulazni_niz[5]==5)
                        {
                                return 'i';
                        }
                        else if(ulazni_niz[5]==2)
                        {
                                return 'j';
                        }
                        else return '?';
                }
                else if(ulazni_niz[4]==5)
                {
                        if(ulazni_niz[5]==5)
                        {
                                return 'k';
                        }
                        else if(ulazni_niz[5]==2)
                        {
                                return 'l';
                        }
                        else return '?';
                }
                else return '?';
            }
            else if(ulazni_niz[3]==7)
            {
                if(ulazni_niz[4]==4)
                {
                        if(ulazni_niz[5]==5)
                        {
                                return 'm';
                        }
                        else if(ulazni_niz[5]==2)
                        {
                                return 'n';
                        }
                        else return '?';
                }
                else if(ulazni_niz[4]==5)
                {
                        if(ulazni_niz[5]==5)
                        {
                                return 'o';
                        }
                        else if(ulazni_niz[5]==2)
                        {
                                return 'p';
                        }
                        else return '?';
                }
                else return '?';
           }
           else return '?';
        }
        else return '?';
}
else if(ulazni_niz[1]==3)
{
        if(ulazni_niz[2]==3)
        {
```

```
        if(ulazni_niz[3]==6)
        {
            if(ulazni_niz[4]==4)
            {
                    if(ulazni_niz[5]==5)
                    {
                            return 'q';
                    }
                    else if(ulazni_niz[5]==2)
                    {
                            return 'r';
                    }
                    else return '?';
            }
            else if(ulazni_niz[4]==5)
            {
                    if(ulazni_niz[5]==5)
                    {
                            return 's';
                    }
                    else if(ulazni_niz[5]==2)
                    {
                            return 't';
                    }
                    else return '?';
            }
            else return '?';
        }
        else if(ulazni_niz[3]==7)
        {
            if(ulazni_niz[4]==4)
            {
                    if(ulazni_niz[5]==5)
                    {
                            return 'u';
                    }
                    else if(ulazni_niz[5]==2)
                    {
                            return 'v';
                    }
                    else return '?';
            }
            else if(ulazni_niz[4]==5)
            {
                    if(ulazni_niz[5]==5)
                    {
                            return 'w';
                    }
                    else if(ulazni_niz[5]==2)
                    {
                            return 'x';
                    }
                    else return '?';
            }
            else return '?';
        }
        else return '?';
    }
    else if(ulazni_niz[2]==5)
    {
        if(ulazni_niz[3]==6)
        {
            if(ulazni_niz[4]==4)
            {
                    if(ulazni_niz[5]==5)
                    {
                            return 'y';
                    }
                    else if(ulazni_niz[5]==2)
                    {
                            return 'z';
                    }
                    else return '?';
            }
            else if(ulazni_niz[4]==5)
            {
                    if(ulazni_niz[5]==5)
```

```
                        {
                                return ' ';
                        }
                        else if(ulazni_niz[5]==2)
                        {
                                return '1';
                        }
                        else return '?';
                    }
                    else return '?';
                }
                else if(ulazni_niz[3]==7)
                {
                    if(ulazni_niz[4]==4)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return '2';
                            }
                            else if(ulazni_niz[5]==2)
                            {
                                    return '3';
                            }
                            else return '?';
                    }
                    else if(ulazni_niz[4]==5)
                    {
                            if(ulazni_niz[5]==5)
                            {
                                    return '4';
                            }
                            else if(ulazni_niz[5]==2)
                            {
                                    return '5';
                            }
                            else return '?';
                    }
                    else return '?';
                }
                else return '?';
           }
           else return '?';
     }
     else return '?';
}
//kraj podprogram za enkodiranje


//podprogram za binarno sabiranje
int saberi_binarno(int a, int b)
{
    if((a==0 && b==0) || (a==1 && b==1))
           return 0;
    else if ((a==0 && b==1) || (a==1 && b==0))
           return 1;
    else cout<<"Brojevi nisu binarni ";
}
//Kraj podprograma za binarno sabiranje

//podprogram za sabiranje brojeva iz GF(8), u kojem su oznake sljedeæe:
/*
000 sam oznaèio sa 0      ----
001 sam oznaèio sa 1      alfa^0
010 sam oznaèio sa 2      alfa^1
100 sam oznaèio sa 3      alfa^2
011 sam oznaèio sa 4      alfa^3
110 sam oznaèio sa 5      alfa^4
111 sam oznaèio sa 6      alfa^5
101 sam oznaèio sa 7      alfa^6
GF(8) je napravljeno koriŽtenjem polinoma
f(x)=x^3+x+1
*/
int saberi(int a, int b)
{
    int a_mojzapis[4], b_mojzapis[4], c_mojzapis[4];
```

```
 if(a==0)
 {
         a_mojzapis[1]=0;
         a_mojzapis[2]=0;
         a_mojzapis[3]=0;
 }
 else if(a==1)
      {
              a_mojzapis[1]=0;
              a_mojzapis[2]=0;
              a_mojzapis[3]=1;
      }
              else if(a==2)
              {
                   a_mojzapis[1]=0;
                   a_mojzapis[2]=1;
                   a_mojzapis[3]=0;
              }
                   else if(a==3)
                   {
                        a_mojzapis[1]=1;
                        a_mojzapis[2]=0;
                        a_mojzapis[3]=0;
                   }
else if(a==4)
 {
         a_mojzapis[1]=0;
         a_mojzapis[2]=1;
         a_mojzapis[3]=1;
 }
else if(a==5)
 {
         a_mojzapis[1]=1;
         a_mojzapis[2]=1;
         a_mojzapis[3]=0;
 }
else if(a==6)
 {
         a_mojzapis[1]=1;
         a_mojzapis[2]=1;
         a_mojzapis[3]=1;
 }
else if(a==7)
 {
         a_mojzapis[1]=1;
         a_mojzapis[2]=0;
         a_mojzapis[3]=1;
 }
 else
 {
      cout<<"Nije unesen broj iz GF(8)";
 }


 if(b==0)
 {
         b_mojzapis[1]=0;
         b_mojzapis[2]=0;
         b_mojzapis[3]=0;
 }
 else if(b==1)
 {
         b_mojzapis[1]=0;
         b_mojzapis[2]=0;
         b_mojzapis[3]=1;
 }
 else if(b==2)
 {
         b_mojzapis[1]=0;
         b_mojzapis[2]=1;
         b_mojzapis[3]=0;
 }
else if(b==3)
 {
         b_mojzapis[1]=1;
         b_mojzapis[2]=0;
         b_mojzapis[3]=0;
```

```
    }
   else if(b==4)
    {
            b_mojzapis[1]=0;
            b_mojzapis[2]=1;
            b_mojzapis[3]=1;
    }
   else if(b==5)
    {
            b_mojzapis[1]=1;
            b_mojzapis[2]=1;
            b_mojzapis[3]=0;
    }
   else if(b==6)
    {
            b_mojzapis[1]=1;
            b_mojzapis[2]=1;
            b_mojzapis[3]=1;
    }
   else if(b==7)
    {
            b_mojzapis[1]=1;
            b_mojzapis[2]=0;
            b_mojzapis[3]=1;
    }
    else
    {
         cout<<"Nije unesen broj iz GF(8)";
    }

    c_mojzapis[1]=saberi_binarno(a_mojzapis[1],b_mojzapis[1]);
    c_mojzapis[2]=saberi_binarno(a_mojzapis[2],b_mojzapis[2]);
    c_mojzapis[3]=saberi_binarno(a_mojzapis[3],b_mojzapis[3]);


    if(c_mojzapis[1]==1)
    {
                        if(c_mojzapis[2]==1)
                        {
                                            if(c_mojzapis[3]==1)
                                            return 6;
                                            else return 5;
                        }
                        else
                        {
                            if(c_mojzapis[3]==1)
                            return 7;
                            else return 3;
                        }
    }
    else
    {
        if(c_mojzapis[2]==1)
        {
                            if(c_mojzapis[3]==1)
                            return 4;
                            else return 2;
        }
        else
        {
            if(c_mojzapis[3]==1)
            return 1;
            else return 0;
        }
    }
}
//Kraj podprograma za sabiranje brojeva iz GF(8)

//podprogram za množenje brojeva iz GF(8), u kojem su oznake sljedeæe:
/*
000 sam oznaèio sa 0      ----
001 sam oznaèio sa 1      alfa^0
010 sam oznaèio sa 2      alfa^1
100 sam oznaèio sa 3      alfa^2
011 sam oznaèio sa 4      alfa^3
110 sam oznaèio sa 5      alfa^4
111 sam oznaèio sa 6      alfa^5
```

```
101 sam oznaèio sa 7      alfa^6
GF(8) je napravljen korištenjem polinoma
f(x)=x^3+x+1
*/
int pomnozi(int a, int b)
{
    if(a==0 || b==0)
            return 0;
    else
         if(a==1 || b==1)
                return a*b;
         else
                return (((a-1)+(b-1))%7)+1;
}
//Kraj podprograma za mnozenje

//podrprogram za pridruđivanje jednog polinoma drugom
/*Prvi napisani polimom postaje drugi.
Stepen prvog polinoma postaje stepen od drugog polinoma.
*/
void prvom_polinomu_pridruzi_drugi(int polinom_A[15], int &stepen_od_A, int polinom_B[15], int stepen_od_B)
{
    for(int i=0; i<=stepen_od_B; i++)
        polinom_A[i]=polinom_B[i];

    stepen_od_A=stepen_od_B;
}
//kraj podprograma za pridruzivanje polinoma

//Podprogram za trađenje multiplikativnog elementa od a
/*GF(8) smo konstruisali koristenjem polinoma x^3+x+1
sa alfom kao primitivnim elementom.
000 sam oznaèio sa 0      ----
001 sam oznaèio sa 1      alfa^0==1==alfa^7
010 sam oznaèio sa 2      alfa^1==alfa
100 sam oznaèio sa 3      alfa^2
011 sam oznaèio sa 4      alfa^3
110 sam oznaèio sa 5      alfa^4
111 sam oznaèio sa 6      alfa^5
101 sam oznaèio sa 7      alfa^6
Generator polinom je
g(x)=(alfa+x)(alfa^2+x)=x^2+alfa^4x+alfa^3=x^2+5x+4
*/
int multiplikativni_inverz_od(int a)
{
    if(a==1)
       return 1;
    else if((a>1) && (a<8))
       return 9-a;
    else
    {
        cout<<"Nije unesen element iz GF(8) ili je unesena nula koja nema inverz";
        cout<<endl;
        return 0;
    }
}
//kraj podprograma za inverzni element

//podprogram za stepenovanje elementa iz GF(8)
int stepenuj_broj(int alfa, int stepen)
{
    if(stepen==1)
        return alfa;
    else
    {
       int rezultat=0;
       rezultat=pomnozi(alfa, alfa);
       for(int i=1; i<=(stepen-2); i++)
       {
           rezultat=pomnozi(rezultat, alfa);
       }
    return rezultat;
    }
}
//kraj podprograma za stepenovanje

//podprogram za
```

```
//Množenje polinoma skalarom
void pomnozi_polinom_brojem(int skalar, int polinom[15])
{
    for(int i=0; i<=15; i++)
        polinom[i]=pomnozi(skalar, polinom[i]);
}
//kraj podprograma za mnozenje pollinoma skalarom

//podprogram za
//množenje polinoma sa promjenjivom na neki stepen
/*program ce pogresno raditi ako je proizvod polinoma i promjenjive na stepen
reda veæeg od 15
*/
void pomnozi_polinom_promjenjivom_na_stepen(int stepen_promjenjive, int polinom[15], int stepen_polinoma)
{
     if(stepen_promjenjive+stepen_polinoma>15)
          cout<<"Program nece raditi ispravno"<<endl;
     for(int i=(stepen_polinoma+stepen_promjenjive); i>=stepen_promjenjive; i--)
          polinom[i]=polinom[i-stepen_promjenjive];
     for(int i=0; i<=stepen_promjenjive-1; i++)
          polinom[i]=0;
}
//kraj podprograma

//podprogram za
//sabiranje dva polinoma
/*program ce pogresno raditi ako je jedan od polinoma
reda veæeg od 15
*/
void saberi_dva_polinoma(int prvi_polinom[15], int drugi_polinom[15], int rezultat_sabiranja[15])
{
     for(int i=0; i<=15; i++)
     {
             rezultat_sabiranja[i]=saberi(prvi_polinom[i],drugi_polinom[i]);
     }

}
//kraj podprograma za sabiranje dva polinoma

//LFSR Algoritam spajanja
/*
Ovo je podprogram za LFSR Synthesis Algoritam (Barlekamp-Massey Algoritam) za
nalaðenje (jednog od) najkraæeg LFSR-a koji generiŽe niz s_0, s_1, .... s_N-1

U ovom podprogramu ulaz je niz s_0, s_1, .... s_N-1 koji predstavlja
sindrom greŽke a izlaz je polinom C(D) i L pomoæu kojeg moðemo konstruisati LFSR
kompleksnosti L

ulaz su: sindrom[15] plinom sidnroma i stepen sidroma
Izlaz je polinom_veze_CD i kompleksnost_L
*/
int LFSR_algoritam_sinteze(int sindrom[15], int stepen_sindroma, int polinom_veze_CD[15], int &stepen_polinoma_veze_CD, int &kom
{
    polinom_veze_CD[0]=1;

    int polinom_veze_CD_zvjezdica[15];
    polinom_veze_CD_zvjezdica[0]=1;
    int stepen_polinoma_veze_CD_zvjezdica=0;

    int polinom_TD[15];
    int stepen_polinoma_TD=0;

    int pomocni_polinom[15];
    for(int i=0; i<=15; i++)
        pomocni_polinom[i]=0;
    int stepen_pomocnog_polinoma=0;

    kompleksnost_L=0;
    int delta_zvjezdica=1;
    int delta;
    int x=1;
    int n=0;

    int N=stepen_sindroma;

    for(int i=0; i<=N; i++)
    {
```

```
            delta=sindrom[i];
            for(int k=1; k<=kompleksnost_L; k++)
            {
                    delta=saberi(delta,pomnozi(polinom_veze_CD[k],sindrom[n-k]));
            }

            if(delta==0)
                        x=x+1;
            else
            {
                prvom_polinomu_pridruzi_drugi(polinom_TD, stepen_polinoma_TD, polinom_veze_CD, stepen_polinoma_veze_CD);
                prvom_polinomu_pridruzi_drugi(pomocni_polinom, stepen_pomocnog_polinoma, polinom_veze_CD_zvjezdica, stepen_polin

                pomnozi_polinom_promjenjivom_na_stepen(x, pomocni_polinom, stepen_pomocnog_polinoma);
                stepen_pomocnog_polinoma=stepen_pomocnog_polinoma+x;

                pomnozi_polinom_brojem(pomnozi(delta,multiplikativni_inverz_od(delta_zvjezdica)), pomocni_polinom);

                saberi_dva_polinoma(polinom_veze_CD, pomocni_polinom, polinom_veze_CD);
                stepen_polinoma_veze_CD=stepen_pomocnog_polinoma;


                if(2*kompleksnost_L<=n)
                {
                          kompleksnost_L=n+1-kompleksnost_L;
                          x=1;
                          prvom_polinomu_pridruzi_drugi(polinom_veze_CD_zvjezdica, stepen_polinoma_veze_CD_zvjezdica, polinom_TD
                          delta_zvjezdica=delta;
                }
                else
                {
                    x=x+1;
                }


            }
            n=n+1;
    }
}
//Kraj LFSR Algoritma spajanja

//Podprogram za enkodiranje Reed-Solomonovog koda.
/*GF(8) smo konstruisali koristenjem polinoma x^^3+x+1
sa alfom kao primitivnim elementom.
000 sam oznaèio sa 0      ----
001 sam oznaèio sa 1      alfa^0
010 sam oznaèio sa 2      alfa^1
100 sam oznaèio sa 3      alfa^2
011 sam oznaèio sa 4      alfa^3
110 sam oznaèio sa 5      alfa^4
111 sam oznaèio sa 6      alfa^5
101 sam oznaèio sa 7      alfa^6
Generator polinom je
g(x)=(alfa+x)(alfa^2+x)=x^2+alfa^4x+alfa^3=x^2+5x+4
*/
//podprogram za enkodiranje informacije
//koristim RS kod èiji je generator g(x)=x^2+alfa^4x+alfa^3
int enkodiraj(int informacija[15],int enkodirana_rijec[15])
{
    /*
    int G[6][8];
    G[1][1]=1; G[1][2]=5; G[1][3]=4; G[1][4]=0; G[1][5]=0; G[1][6]=0; G[1][7]=0;
    G[2][1]=0; G[2][2]=1; G[2][3]=5; G[2][4]=4; G[2][5]=0; G[2][6]=0; G[2][7]=0;
    G[3][1]=0; G[3][2]=0; G[3][3]=1; G[3][4]=5; G[3][5]=4; G[3][6]=0; G[3][7]=0;
    G[4][1]=0; G[4][2]=0; G[4][3]=0; G[4][4]=1; G[4][5]=5; G[4][6]=4; G[4][7]=0;
    G[5][1]=0; G[5][2]=0; G[5][3]=0; G[5][4]=0; G[5][5]=1; G[5][6]=5; G[5][7]=4;
    */
//prethodnu generator matricu elementarnim transformacijama svedimo na:
    int G[6][8];
    G[1][1]=1; G[1][2]=0; G[1][3]=0; G[1][4]=0; G[1][5]=0; G[1][6]=5; G[1][7]=2;
    G[2][1]=0; G[2][2]=1; G[2][3]=0; G[2][4]=0; G[2][5]=0; G[2][6]=6; G[2][7]=2;
    G[3][1]=0; G[3][2]=0; G[3][3]=1; G[3][4]=0; G[3][5]=0; G[3][6]=6; G[3][7]=4;
    G[4][1]=0; G[4][2]=0; G[4][3]=0; G[4][4]=1; G[4][5]=0; G[4][6]=1; G[4][7]=1;
    G[5][1]=0; G[5][2]=0; G[5][3]=0; G[5][4]=0; G[5][5]=1; G[5][6]=5; G[5][7]=4;

    for(int i=0; i<=11; i++)
        enkodirana_rijec[i]=0;
```

```
    for(int i=1; i<=7; i++)
        for(int j=1; j<=5; j++)
             enkodirana_rijec[i]=saberi(enkodirana_rijec[i],pomnozi(informacija[j],G[j][i]));
}
//kraj podprograma za enkodiranje

//podprogram za Diskretna Furijeova transformacija
//DFT
//koristim RS kod duđine 7 kod èiji je generator g(x)=x^2+alfa^4x+alfa^3
int DFT(int pristigla_rijec[15], int diskretna_FT[15])
{
    for(int i=1; i<=7; i++)
       for(int n=1; n<=7; n++)
       diskretna_FT[i]=saberi(diskretna_FT[i], pomnozi(pristigla_rijec[n],multiplikativni_inverz_od(stepenuj_broj(2,i*n))));


/****** provjera da li se dobije odgovarajuæa matrica DFT
    for(int i=1; i<=7; i++)
    {
       for(int n=1; n<=7; n++)
       {
            cout<<multiplikativni_inverz_od(stepenuj_broj(2,i*n))<<"   ";
       }
       cout<<endl;
    }
**********/
}
//kraj podprograma za diskretnu FT

// podprogram za LFSR
// program vraæa sljedeæu vrijednost iz niza koji se uèita
// èita brojeve redom niz_brojeva[1], niz_brojeva[2],niz_brojeva[3]
// koristi polinom veze
int LFSR(int polinom_veze_CD[15], int niz_brojeva[15], int kompleksnost_L)
{
    int rezultat=0;
    for(int i=0; i<kompleksnost_L; i++)
    {
            rezultat=saberi(rezultat,pomnozi(polinom_veze_CD[(kompleksnost_L-i)],niz_brojeva[(i+1)]));
    }
    return rezultat;
}
//kraj podprograma za LFSR


//podprogram za ciklièko pomjeranje za jedan unaprijed
int ciklicki_pomjeri_za_jedan(int niz[15], int stepen_niza)
{
    for(int i=stepen_niza; i>=0; i--)
            niz[i+1]=niz[i];
}
//kraj podprograma za ciklicko pomjeranje

// podprogram za inverznu DFT
/*GF(8) smo konstruisali koristenjem polinoma x^3+x+1
sa alfom kao primitivnim elementom.
000 sam oznaèio sa 0      ----
001 sam oznaèio sa 1      alfa^0==1==alfa^7
010 sam oznaèio sa 2      alfa^1==alfa
100 sam oznaèio sa 3      alfa^2
011 sam oznaèio sa 4      alfa^3
110 sam oznaèio sa 5      alfa^4
111 sam oznaèio sa 6      alfa^5
101 sam oznaèio sa 7      alfa^6
Generator polinom je
g(x)=(alfa+x)(alfa^2+x)=x^2+alfa^4x+alfa^3=x^2+5x+4
*/
int inverzna_DFT(int rijec[15], int inverzija[15])
{
    for(int i=1; i<=7; i++)
       for(int n=1; n<=7; n++)
       inverzija[i]=saberi(inverzija[i], pomnozi(rijec[n],(stepenuj_broj(2,i*n))));
}
//kraj podprograma za inverziju

//podprogram za dekodiranje RS(8,7,3) koda
```

```
void dekodiraj_rijec(int pristigla_rijec[15], int dekodirana_rijec[15])
{

//+++++++++++++++ prvi korak

    int transformacija[15];
    for(int i=0; i<=15; i++)
        transformacija[i]=0;

    DFT(pristigla_rijec, transformacija);

//++++++++++++++++++++++++++++
//+++++++++++++++ drugii korak

    int E[15];
    for(int i=0; i<=15; i++)
        E[i]=0;

    E[0]=transformacija[1];
    E[1]=transformacija[2];

    int polinom_veze_CD[15];
    for(int i=0; i<=15; i++)
        polinom_veze_CD[i]=0;

    int stepen_polinoma_veze_CD=0;
    int kompleksnost_L=0;

    LFSR_algoritam_sinteze(E, 1, polinom_veze_CD, stepen_polinoma_veze_CD, kompleksnost_L);


//  USLOV HOÆE LI ALGORITAM RADITI ISPRAVNO
    if(2*kompleksnost_L>=3)
       cout<<"Algoritam nece raditi ispravno - napravljeno je vise od 1 greske "<<endl;

//+++++++++++++++ treæi korak

    int brojevi_koji_idu_u_LFSR[15];
    for(int i=0; i<=15; i++)
        brojevi_koji_idu_u_LFSR[i]=0;

    for(int i=1; i<=5; i++)
    {
        for(int j=0; j<=kompleksnost_L-1; j++)
              for(int k=1; k<=kompleksnost_L; k++)
              {
                      brojevi_koji_idu_u_LFSR[k]=E[i-j];
              }
        E[i+1]=LFSR(polinom_veze_CD, brojevi_koji_idu_u_LFSR, kompleksnost_L);
    }


//+++++++++++++++ èetvrti korak
    ciklicki_pomjeri_za_jedan(E,6);
//+++++++++++++++ peti korak

    int e[15];
    for(int i=0; i<=15; i++)
        e[i]=0;

    inverzna_DFT(E, e);


//+++++++++++++++ Žesti korak


    saberi_dva_polinoma(pristigla_rijec, e, dekodirana_rijec);

}
//kraj podprograma za dekodiranje

int main()
{
  //deklaracija promjenjivih
  ofstream izlaz;
  ifstream ulaz;
  int pauza;
```

```
  FILE *pFile;
  int c, broj_znakova_u_ulaznom_fajlu=1;
  char znakovi_iz_fjla[50], x;
  char slovo;
  int enkodirana_rijec[15];
  for(int i=0; i<=15; i++)
        enkodirana_rijec[i]=0;
  int dekodirana_rijec[15];

  for(int i=0; i<=15; i++)
        dekodirana_rijec[i]=0;

//Ulazni fajl
  pFile=fopen ("ulaz.txt","r");
  if (pFile==NULL) perror ("Error opening file");
  else
  {
    do {
      c = getc (pFile);
      znakovi_iz_fjla[broj_znakova_u_ulaznom_fajlu] = ( (int) c);
      broj_znakova_u_ulaznom_fajlu++;
    } while (c != EOF);

    broj_znakova_u_ulaznom_fajlu=broj_znakova_u_ulaznom_fajlu-1;
    fclose (pFile);
  }
//Citav ulaz je zapisan u niz znakovi_iz_fjla[] koji ima znakova "broj_znakova_u_ulaznom_fajlu"

//++++++++++++ kodiraj informaciju u RS kod i zapisi je u fajl komunikaciski_kanal

    int informacija[15];
    for(int i=0; i<=15; i++)
        informacija[i]=0;

    izlaz.open("komunikaciski_kanal.txt");
    for(int i=1; i<broj_znakova_u_ulaznom_fajlu; i++)
    {
            for(int j=0; j<=7; j++)
                    informacija[j]=0;

            zapisi_u_oktalnom_obliku_slovo(znakovi_iz_fjla[i],informacija);
            enkodiraj(informacija, enkodirana_rijec);

            for(int i=1; i<=7; i++)
                    izlaz<<enkodirana_rijec[i];

            izlaz<<endl;
    }
    izlaz.close();
//++++++++++++++++++++++++++++++++++++++++++++

cout<<"Otvorite fajl komunikaciski_kanal.txt i izvrsite promjene"<<endl;
cout<<"Kad zavrsite unesite neki broj sa tastature i pritisnite enter";
cin>>pauza;

//++++++++ucitaj brojeve iz informacija.txt i dekodiraj


    int privremeni_niz[15];
    for(int i=0; i<=15; i++)
        privremeni_niz[i]=0;
    int temp=0;


//otvori ulaz za izlaz podataka u njih
    izlaz.open ("izlaz.txt");
//++++++++++++++++++++++++++++++

    ulaz.open("komunikaciski_kanal.txt");
    if (!ulaz)
    {
        cout << "Unable to open file";
        cin>>pauza;
        exit(1);  // izadji iz programa sa greskom
    }
```

```
    while (ulaz >> x)
    {
          temp++;
          privremeni_niz[temp]=x-48;
          if(temp==7)
          {
              temp=0;
              dekodiraj_rijec(privremeni_niz, dekodirana_rijec);
              izlaz<<dekodiraj_u_slovo(dekodirana_rijec);
          }
    }
    ulaz.close();



//zatvori izlaz
          izlaz.close();
//++++++++++++++++++++++++++++++

//++++++++++++++++++++++++++++++++++++++++++++++++++

    cout<<endl<<"Kraj programa (unesite neki broj sa tastature i pritisnite enter)"<<endl;
    cin>>pauza;

    return 0;
}
```

Sadržaj

Literatura

[1] J. L. Massey, *"Applied Digital Information Theory II"*, Lecture notes hold by author from 1981 until 1997 at the ETH Zurich, str. 49-60.

[2] J. L. Massey, *"Discrete Fourier Transform in Coding and Cryptography"*, In Proc. IEEE Information Theory Workshop, San Diego, CA, 8-11, str. 1-2, Feb. 1998.

[3] R. E. Blahut, *"Algebraic Codes for Data Transmission"*, Cambridge University Press, str. 67-163, 2003.

[4] J. L. Massey, *"Shift-Register Synthesis and BCH Decoding"*, IEEE Trans. on Info. Th., Vol. IT-15, str. 122-127, Jan. 1969.

[5] J. L. Massey i T. Schaub, *"Linear Complexity in Coding Theory"*, Coding theory and Applications (Eds G. Cohen and Ph. Godlewski), Lecture notes in Computer Science, No. 311. Heidelberg and New York: Springer, str. 19-32, 1998.

[6] G. Bachman, L. Narici i E. Beckenstein, *"Fourier and Wavelet Analysis"*, Springer-Verlag New York, str. 383-406, 2000.

[7] D.G. Hoffman, D.A. Leonard, C.C. Lindner, K.T. Phelps, C.A. Rodger i J.R. Wall, *"Coding Theory - The Essentials"*, Marcel Dekker, str. 97-164, 1991.