

POGLAVLJE

6

Integritet podataka

Iako je tačno da su današnja deca izložena većoj količini informacija i raznovrsnijim iskustvima nego njihovi vršnjaci u prošlosti, to ne znači automatski da su sofisticirani. Uvek znamo više nego što razumemo i sa gomilom informacija koje su predstavljene mladim ljudima praznina između znanja i razumevanja i između iskustva i učenja postaje veća nego što je bila u prošlosti.

—David Elkind (1709-1784), američki psiholog

6.1 Uvod

U prethodnim poglavljima su opisani mehanizmi koji su neophodni za smeštanje i prenos informacija. Svi ti metodi, bez obzira koliko su sofisticirani, ne mogu da garantuju efikasne i sigurne komunikacije. Uzmimo za primer sledeću poruku koja je preneta elektronskom poštom:

Your brothel in New Orleans needs money. (*Vašem bordelu u Nju Orleansu je neophodan novac.*)

Vaša reakcija može da bude raznolika, od zbumjenosti do krajnjeg užasa. Šta ako Vaša supruga prva vidi ovu poruku? Ona ne zna da je originalna poruka glasila:

Your brother in New Orleans needs money. (*Vašem bratu iz Nju Orleansa je neophodan novac.*)

Šta se desilo? Poslata poruka je bila krajnje bezazlena, a problem je nastao u toku prenosa. Slovo *r* iz reči *brother* je bilo ASCII kodirano kao 1110010. Nažalost, zbog nekih električnih smetnji, došlo je do promene četiri srednja bita 1001 u 0110, tako da je primljena kombinacija 1101100, što predstavlja kod za slovo *l*.

Mislim da je nesporno da sistem koji dopušta isporučivanje ovakvo izmenjenih poruka nije poželjan. Međutim, činjenica je da se greške javljaju. Svaka poruka koja se prenosi elektronskim putem je podložna smetnjama. Jaka sunčeva svetlost, električni udari, fluktuacije u napajanju, ili udarac ašovom u kabl mogu da izazovu neverovatne i nepredvidljive "stvari" u toku prenosa. Međutim, ne možemo da dopustimo da, na primer, astronauti prime netačne instrukcije za navigaciju, ili da neka švajcarska banka na neki račun deponuje milion dolara više nego što je potrebno (ako nije reč o mom računu!).

Mogućnost detektovanja promena u toku prenosa naziva se **detekcija grešaka**. U većini slučajeva, kada se greške detektuju, poruka se odbacuje, obaveštava se pošiljalac i poruka se ponovo šalje. Naravno, nema nikakvih garancija da ponovo neće doći do oštećenja poruke i zato je neophodno razviti protokole koji omogućavaju razmenu poruka (bez obzira na njihov status) između pošiljaoca i primaoca. U stvari, nema nikakvih garancija da neće biti oštećene i same informacije o statusu poruke. Ovo zahteva razradu detaljnih protokola, ali o njima će biti više reči u Poglavlju 8.

Ponovno slanje poruke ponekad nije praktično. Možda nema dovoljno vremena, kao u slučaju real-time aplikacija. Jedan primer su sonde za svemirska istraživanja. Mogu da proteknu sati dok se ne prenesu značajni telemetrijski podaci iz udaljene tačke u svemiru. Osim toga, ako je sonda veoma daleko, signali mogu da budu veoma slabici, samim tim, podložni smetnjama. Dok sonda primi zahtev za ponovno slanje, može da se pomeri na neku drugu tačku sa koje originalni podaci više ne mogu da se pribave. Osim toga, verovatnoća da će doći do oštećenja signala je veoma visoka kod prenosa na velikim udaljenostima i velika je verovatnoća da poruka nikada neće biti primljena bez neke greške. Sledeći primer je gledanje, ili slušanje multimedijalnih zapisa u realnom vremenu. Ako gledate video i nešto se desi u nekoliko kadrova, nije praktično da se vraćate nazad i ponovo gledate te kadrove. To bi bilo kao da gledate film na DVD-ju i neko neprestano premotava film unazad. U nekim slučajevima, kada se greška detektuje, moguće ju je ispraviti bez ponovnog prenosa. Ovo se naziva korekcija grešaka. Pošiljalac nikada neće saznati da je došlo do oštećenja poruke i da je ona ispravljena. Krajnji zaključak je da se poruke eventualno mogu tačno isporučiti.

U ovom poglavlju se bavimo integritetom podataka, mogućnošću da se utvrди kada je došlo do njihovog oštećenja. U odeljku 6.2 predstavićemo proveru parnosti, metod za detektovanje grešaka u određenim bitovima. To je jednostavan i prilično naivan pristup i, mada se obično ne implementira samostalno, ipak igra važnu ulogu u nekim složenijim šemama. U odeljku 6.3 upoznaćete cikličnu proveru redundantnosti, komplikovani metod koji se zasniva na interpretiranju nizova bitova kao polinoma, koji se nakon prijema dele da bi se utvrdilo da li postoje greške. Videćete da je ovaj metod izuzetno tačan i da, i pored složenih izračunavanja, može efikasno da se implementira. Osim toga, ovo je često korišćeni metod za detektovanje grešaka.

Konačno, u odeljku 6.4 predstavićemo metod za korekciju grešaka pod nazivom Hamingov kod. To je metod koji koristi višestruke provere parnosti na takav način da se greška, ako je došlo do promene jednog bita, detektuje u jedinstvenoj kombinaciji provere parnosti. Kada se utvrdi lokacija na kojoj je došlo do greške, bit može da se promeni i na taj način je poruka ispravljena.

6.2 Jednostavne tehnike za detekciju grešaka

Provera parnosti

Tehnike za detektovanje grešaka zahtevaju slanje dodatnih bitova čije vrednosti zavise od podataka koji su poslati. Ako se podaci promene, vrednosti dodatnih bitova neće odgovarati novim podacima (bar ne teorijski). Verovatno najčešće korišćeni pristup je provera parnosti, koja uključuje brojanje bitova sa vrednošću 1, a zatim se postavlja jedan dodatni bit, tako da ukupan broj bitova sa vrednošću 1 bude paran (**parna parnost**), ili neparan (**neparna parnost**).

Dodatni bit se naziva bit parnosti. Našu raspravu zasnivamo na parnoj parnosti, a čitaoci mogu da konstruišu analognu diskusiju za neparnu parnost.

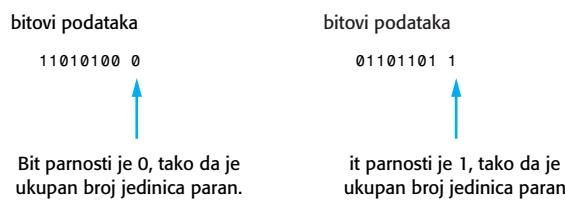
Ilustracije radi, pretpostavimo da je ukupan broj bitova sa vrednošću 1 neparan. Ako se bit parnosti definiše kao 1, ukupan broj 1 bitova je sada paran. Slično tome, ako je ukupan broj 1 bio paran, bit parnosti je 0. Razmotrite nizove bitova sa slike 6.1. Prvi niz ima četiri 1 bita. Zato je bit parnosti 0. Drugi niz ima pet 1 bitova, pa je njegov bit parnosti 1.

ANALIZA PROVERE PARNOSTI Provera parnosti detektuje samo greške u jednom bitu. Bit parnosti je prenet zajedno sa bitovima podataka, a prijemnik proverava parnost. Ako prijemnik pronađe neparan broj bitova, došlo je do greške. Ipak, greške u samo jednom bitu su veoma retke kod električnih prenosa. Na primer, pretpostavimo da je došlo do greške zbog kratkog strujnog udara, ili statickog elektriciteta, čije je trajanje izraženo u hiljaditim delovima sekunde. Sa ljudskog stanovišta, to je skoro neprimetno. Ali, ako su podaci prenošeni brzinom od 1 Mbps, u tom hiljaditom delu sekunde 10.000 bitova je izloženo nastalim smetnjama. Kada je oštećeno više bitova, kažemo da je došlo do **navalne greške** (burst error).

Kako provera parnosti funkcioniše u slučaju proizvoljnih navalnih grešaka? Pretpostavimo da se u toku prenosa menjaju vrednosti dva bita. Ako su oba bila 0, menjaju se u 1. Dve dodatne jedinice i dalje u zbiru daju paran broj jedinica. Slično tome, da su oba bila 1, promenili bi se u 0, tako da bi postojale dve jedinice manje, ali bi ukupan broj i dalje bio paran. Da su bitovi imali suprotne vrednosti i promenili ih, i dalje bi imali suprotne vrednosti. I ovoga puta ukupan broj jedinica ostaje isti. Krajnji zaključak je da provera parnosti ne može da detektuje dvostrukе greške.

U opštem slučaju, provera parnosti može da detektuje greške koje se javljaju na neparnom broju bitova. Ako se promeni vrednost parnog broja bitova, provera parnosti neće moći da detektuje grešku. Zaključak je da provera parnosti detektuje greške sa verovatnoćom od 50 odsto za navalne greške, ali to nije dovoljan procenat za komunikacione mreže.

Da li je zbog ovoga provera parnosti beskorisna? Odgovor je negativan, i to zbog dva razloga. Prvo, neke organizacije kompjuterske memorije smeštaju bitove iz bajta, ili reči na različitim čipovima. Tako se prilikom pristupa reči koriste različite putanje. U takvим slučajevima smetnje na jednoj putanji mogu da izazovu grešku u jednom bitu. Takve arhitekture često koriste dodatne memorijske čipove za proveru parnosti (detaljnija rasprava o njima ne može da se uklopi u prevideni obim ovog teksta). Drugi razlog je to što provera parnosti predstavlja osnovu za tehniku korekcije grešaka, koju ćemo predstaviti u odeljku 6.4.



SLIKA 6.1 Detektovanje grešaka u jednom bitu pomoći provere parnosti

Čeksume

Sledeći pristup deli sve bitove podataka u 32-bitne* grupe i svaku tretira kao celobrojnu vrednost. Te vrednosti se, zatim, sabiraju, tako da daju čeksumu (checksum). Svaki prenos u sabiranju koji bi zahtevao više od 32 bita se ignoriše. Zatim, proces koristi mod 2³² vrednosti sume. Dodatna 32 bita predstavljaju čeksumu, koja se dodaje podacima pre nego što se pošalju. Prijemni uređaj deli rezultat sa onim što je smešteno u dodatna 32 bita. Ako se rezultati ne poklapaju, došlo je do greške.

Ovaj pristup je efikasniji od jednostavne provere parnosti i detektuje sve vrste navalnih gresaka, jer nasumična promena brojeva obično menja vrednost njihove sume. Ipak, ovaj pristup ne detektuje sve greske. Jednostavan primer je greska koja izaziva da se jedna od 32-bitnih vrednosti poveća za određeni iznos, a neka druga 32-bitna vrednost se smanji za isti iznos. Suma tih vrednosti ostaje nepromenjena, i pored toga što je došlo do greške.

6.3 Detekcija grešaka pomoću ciklične provere redundantnosti

U ovom odeljku upoznaćete metod poznat pod nazivom **ciklična provera redundantnosti** (CRC-cyclic redundancy check), koji je mnogo tačniji i od provere parnosti i od metoda koji koristi čeksume. Osim toga, pokazaćemo kako se može efikasno implementirati.

CRC je neuobičajen, ali "pametan" metod koji proveru grešaka izvodi deljenjem **polinoma**. Vaša prva reakcija je verovatno: "U kakvoj je vezi deljenje polinoma sa prenosom niza bitova?". U stvari, metod interpretira svaki niz bitova kao polinom. U opštem slučaju, niz bitova

$$b_{n-1}b_{n-2}b_{n-3}\dots b_2b_1b_0$$

interpretira se kao polinom

$$b_{n-1}x^{n-1} + b_{n-2}x^{n-2} + b_{n-3}x^{n-3} + \dots + b_2x^2 + b_1x + b_0$$

Na primer, niz bitova 10010101110 interpretira se kao

$$x^{10} + x^7 + x^5 + x^3 + x^2 + x^1$$

Pošto je svaki koeficijent b_i ili 0, ili 1, pišemo x^i kada je b_i jednako 1, a ne zapisujemo ih kada je b_i jednako 0.

U sledećim koracima opisan je CRC metod. Pretpostavljamo da je kod svih izračunavanja korišćen mod 2.

1. Na dati niz bitova dodajte nekoliko nula na kraj niza (kasnije ćemo reći koliko i zašto) i nazovite ga B . Neka $B(x)$ bude polinom koji odgovara nizu B .
2. $B(x)$ se deli sa nekim ugovorenim polinomom $G(x)$ (**generator polinoma**) i utvrđuje se ostatak $R(x)$.

* Može da se koriste i grupe od osam, ili 16 bitova.

3. Definiše se $T(x) = B(x) - R(x)$. Kasnije ćemo pokazati da $T(x)/G(x)$ generiše ostatak 0 i da se oduzimanje može izvesti zamenom prethodno dodatih 0 bitova nizom bitova koji odgovara polinomu $R(x)$.
4. Prenosi se T , niz stringova koji odgovara polinomu $T(x)$.
5. Neka T predstavlja niz bitova koje primalac dobija i neka je $T'(x)$ pridruženi polinom. Primalac deli $T'(x)$ sa $G(x)$. Ako je ostatak 0, primalac zaključuje da je $T = T'$ i da nije bilo grešaka. U suprotnom, on zaključuje da je došlo do greške i zahteva ponovni prenos.

Pre nego što počnete da očajavate, moramo da damo odgovore na neka pitanja. Zašto izvršavamo sve ove korake? Da li postoji ikakva validacija donošenja zaključaka u prijemniku nako deljenja $T'(x)$ sa $G(x)$? Koliko je ovaj metod tačan? Da li i pošiljalac i primalac moraju da prođu kroz sve ove korake prilikom slanja svakog okvira? Međutim, ne možemo da damo odgovore na ova pitanja bez nekoliko preliminarnih objašnjenja. Prepostavljamo da posedujete predznanja o polinomima i njihovim operacijama korišćenjem realnih brojeva, ali ćemo, ipak, dati kratko objašnjenje o deljenju polinoma po modulu 2.

Deljenje polinoma

Na slici 6.2 prikazan je primer deljenja polinoma $T(x)/G(x)$, gde je

$$T(x) = x^{10} + x^9 + x^7 + x^5 + x^4$$

i

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

Ovo je slično običnom deljenju polinoma koje poznajete iz algebre, osim što se koristi aritmetika modula 2. Sabiranje i oduzimanje po modulu 2 definiše se na sledeći način:

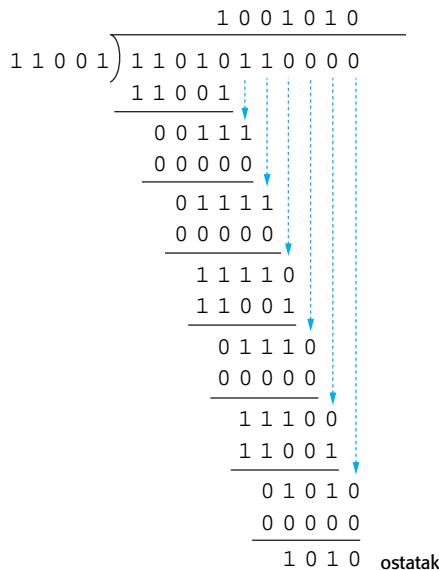
$$0 + 0 = 0 \quad 1 + 0 = 1 \quad 0 + 1 = 1 \quad 1 + 1 = 0$$

i

$$0 - 0 = 0 \quad 1 - 0 = 1 \quad 0 - 1 = 1 \quad 1 - 1 = 0$$

$$\begin{array}{r}
 & & x^6 & + x^3 & + x \\
 & & \hline
 x^4 + x^3 + 1) & x^{10} + x^9 & + x^7 & + x^5 + x^4 \\
 & x^{10} + x^9 & + x^6 & \hline
 & x^7 + x^6 + x^5 + x^4 & & \\
 & x^7 + x^6 & + x^3 & \hline
 & x^5 + x^4 + x^3 & & \\
 & x^5 + x^4 & + x & \hline
 & x^3 & + x & \text{ostatak}
 \end{array}$$

SLIKA 6.2 Izračunavanje $(x^{10} + x^9 + x^7 + x^5 + x^4)/(x^4 + x^3 + 1)$



SLIKA 6.3 Sintetičko deljenje $(x^{10} + x^9 + x^7 + x^5 + x^4) / (x^4 + x^3 + 1)$

Primećujete da su sabiranje i oduzimanje po modulu 2 identični operaciji isključivo ILI (exclusive OR). Ovo je značajna činjenica koju ćemo koristiti kasnije u raspravi o implementaciji CRC-a.

Na slici 6.3 prikazano je sintetičko deljenje istih polinoma. Možda se sećate iz algebre da postoji jedna prečica koja koristi samo koeficijente polinoma (u ovom slučaju nizove bitova). Zapamtite da treba da koristite nule na mestima gde nedostaju članovi polinoma, tako da je 11010110000 lista koeficijenata za polinom $x^{10} + x^9 + x^7 + x^5 + x^4$, a 11001 za polinom $x^4 + x^3 + 1$.

Način kako CRC funkcioniše

Pogledajte sada kako CRC funkcioniše. Pretpostavimo da želite da pošaljete niz bitova 1101011, a da je generator polinoma $G(x) = x^4 + x^3 + 1$; kasnije ćemo predstaviti neke kriterijume za izbor $G(x)$.

1. Dodajte nule na kraj niza. Broj 0 je isti kao i stepen generatora polinoma (u ovom slučaju 4). Tako dobijamo niz 11010110000.
2. Podelite $B(x)$ sa $G(x)$. Na slikama 6.2 i 6.3 pokazan ju rezultat za ovaj primer, sa ostatkom $R(x) = x^3 + x$, ili bitskim ekvivalentom 1010. Napomenimo da ovo može da se predstavi jednačinom

$$\frac{B(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

gde $Q(x)$ predstavlja količnik. Ekvivalentno, možemo da zapišemo

$$B(x) = G(x) \times Q(x) + R(x)$$

3. Definišite $T(x) = B(x) - R(x)$. Pošto oduzimanje uzima razlike između koeficijenata odgovarajućih članova, razliku izračunavamo oduzimanjem bitova koji su povezani svakom polinomu. U ovom slučaju imamo

$$\begin{array}{r} 11010110000 \quad \text{niz bitova } B \text{ (bit string } B) \\ - 1010 \quad \text{niz bitova } R \text{ (bit string } R) \\ \hline 11010111010 \quad \text{niz bitova } T \text{ (bit string } T) \end{array}$$

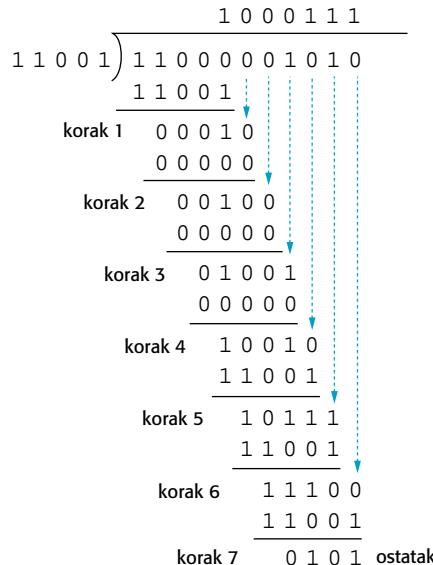
Primećujete da je niz T , u stvari, isti kao i string B sa dodatim nulama zamenjenim sa R . Sledeća važna činjenica, koja je pokazana na slici 6.4, je da dobijamo ostatak 0 ako podelimo $T(x)$ sa $G(x)$.* Nakon toga, pošiljalac prenosi niz T .

4. Ako niz T stigne bez grešaka, deljenje sa $G(x)$ će dati ostatak 0. Ali, prepostavimo da je niz T oštećen u toku prenosa. Na primer, neka su neki bitovi na sredini promenjeni u 0, tako da je primljen niz 11000001010. Primalac vrši sintetičko deljenje sa $G(x)$ i ostatak je različit od 0 (slika 6.5). Pošto je ostatak različit od 0, primalac zaključuje da je došlo do greške.

$$\begin{array}{r} 1001010 \\ \hline 11001) 11010111010 \\ 11001 \\ \hline 00111 \\ 00000 \\ \hline 01111 \\ 00000 \\ \hline 11111 \\ 11001 \\ \hline 01100 \\ 00000 \\ \hline 11001 \\ 11001 \\ \hline 00000 \\ 00000 \\ \hline 00000 \text{ ostatak} \end{array}$$

SLIKA 6.4 Deljenje $T(x)$ sa $G(x)$

* Postoji analogija sa korišćenjem celih brojeva - ako su p i q celi brojevi i ako je r celobrojni ostatak dobijen deljenjem p sa q , onda je $p - r$ deljivo sa q . Na primer, $8/3$ generiše ostatak 2, a $8 - 2$ je deljivo sa 3.



SLIKA 6.5 Deljenje primljenog polinoma sa $G(x)$

(Napomena: Ovo nije isto kao da kažemo da deljenje oštećenog niza sa $G(x)$ uvek daje ostatak različit od 0. To može da se desi, ali, ako je $G(x)$ pažljivo odabran, to će se desiti veoma retko. Ovo je sledeća tema koju ćemo obraditi.)

Analiza CRC-a

Mehanizam CRC-a je sasvim jednostavan. Još uvek nismo odgovorili na pitanje ima li ikakve koristi od ovog metoda. Da li će primalac moći da detektuje oštećeni okvir? Oslanjamо se na prepostavku da će deljenje oštećenog okvira sa $G(x)$ dati ostatak različit od nule. Ali, da li je to uvek tačno? Da li je moguće da se bitovi u nizu T promene tako da posle deljenja sa $G(x)$ ostatak bude nula?

Kompletan i detaljan dokaz zahteva poznavanje svojstava faktorizacije prstena polinoma (oblast apstraktne matematike) i ovde se nećemo baviti time. Umesto toga, sledi kraća rasprava koja će Vam pomoći da steknete osećaj za način na koji funkcioniše. Za početak, definišimo preciznije šta tražimo. Promena bitova u nizu T je analogna sabiranju nekog nepoznatog polinoma sa $T(x)$. Dakle, ako T' predstavlja primljeni niz i ako je $T'(x)$ pridruženi polinom, onda je $T'(x) = T(x) + E(x)$, gde je $E(x)$ nepoznat primaocu niza T . U prethodnom primeru

$$\text{niz } T = 11010111010 \text{ odgovara polinomu } T(x) = x^{10} + x^9 + x^7 + x^5 + x^4 + x^3 + x$$

$$\text{niz } E = 00010110000 \text{ odgovara polinomu } E(x) = x^7 + x^5 + x^4$$

$$\text{niz } T' = 11000001010 \text{ odgovara polinomu } T'(x) = x^{10} + x^9 + x^3 + x$$

Ne zaboravite da se sabiranje izvodi pomoću operacije isključivo ILI. Na primer, sabiranje članova x^7 iz $E(x)$ i $T(x)$ daje $x^7 + x^7 = (1 + 1) \times x^7 = 0$.

Moramo da damo odgovor na pitanje kada $T(x) + E(x)/G(x)$ daje ostatak nula. Pošto je $(T(x) + E(x))/G(x) = T(x)/G(x) + E(x)/G(x)$ i pošto prvi član daje ostatak nula, drugi član određuje ostatak. Zbog toga, postavljeno pitanje može da se preformuliše tako da glasi za koje polinome $E(x)$ deljenje $E(x)/G(x)$ daje ostatak nula.

Sada možemo da tvrdimo sledeće:

Nedetektovane greške u toku prenosa odgovaraju greškama za koje je $G(x)$ faktor polinoma $E(x)$.

Sledeće pitanje je pod kojim je uslovima $G(x)$ faktor polinoma $E(x)$. Proučićemo najpre najjednostavniji primer, gde se menja samo jedan bit u nizu T . U ovom slučaju $E(x)$ ima samo jedan član - x^k za neki celi broj k . Jedini način da $G(x)$ bude faktor x^k je da je $G(x)$ jednak x podignut na neki stepen. Sve dok biramo $G(x)$ sa najmanje dva člana, to neće biti moguće. Tako će CRC moći da detektuje sve jednostrukе greške.

Zatim, razmotrimo navalnu grešku dužine $k \leq r = \text{stepen } G(x)$.^{*} Prepostavimo da je polinom $T(x)$ predstavljen kao

$$t_n t_{n-1} \dots \underbrace{t_{i+k-1} t_{i+k-2} \dots t_i}_{k \text{ bitovi na koje smetnje utiču}} t_{i-1} \dots t_1 t_0$$

a $t_{i+k-1} \dots$ i t_i su prvi i poslednji bit koji će biti oštećeni. Bitovi između ova dva bita se proizvoljno menjaju. To znači da je

$$E(x) = x^{i+k-1} + \dots + x^i = x^i \times (x^{k-1} + \dots + 1)$$

Zato je

$$\frac{E(x)}{G(x)} = \frac{x^i \times (x^{k-1} + \dots + 1)}{G(x)}$$

Prepostavimo sada da je $G(x)$ izabранo tako da x nije faktor $G(x)$. Zbog toga, $G(x)$ i x^i iz prethodnog razlomka nemaju zajedničke faktore. Ako je $G(x)$ faktor brojčoca, onda mora da bude faktor $(x^{k-1} + \dots + 1)$. Pošto smo izabrali $k \leq r$, onda je $k-1 < r$ i $G(x)$ ne može da bude faktor polinoma sa manjim stepenom.

Zato donosimo sledeći zaključak:

Ako x nije faktor $G(x)$, onda se detektuju sve navalne greške koje imaju dužinu manju, ili jednaku stepenu $G(x)$.

Razmotrite sledeću navalnu grešku bilo koje dužine u kojoj je oštećen neparan broj bitova. Pošto $E(x)$ ima član za svaki oštećeni bit, sadrži neparan broj članova. Zbog toga, $E(1)$ (isključivo ILI neparnog broja jedinica) daje 1. Sa druge strane, prepostavimo da je $x+1$ faktor polinoma $G(x)$. U tom slučaju, možemo da zapišemo $G(x) = (x+1) \times H(x)$, gde je $H(x)$ neki izraz.

* Stepen polinoma je najveći eksponent za x .

Pogledajte sada šta se dešava ako se pretpostavi da su se javile neke nedetektovane greške. Sećate se da nedetektovana greška znači da je $G(x)$ faktor polinoma $E(x)$. Zamenom $G(x)$ sa $(x + 1) \times H(x)$ dobija se da je $E(x) = (x + 1) \times H(x) \times K(x)$. Sada, ako se ova jednačina izračuna za $x = 1$, faktor $x + 1$ izjednačava $E(1)$ sa 1.

Jasno je da oboje ne može da se javi istovremeno. Ako i dalje pretpostavljamo da je $x + 1$ faktor polinoma $G(x)$, onda pretpostavka o nedetektovanoj grešci koja oštećuje neparan broj bitova nije opravdana. Drugim rečima:

Ako je $x + 1$ faktor $G(x)$, onda su sve navalne greške koje oštećuju neparan broj bitova detektovane.

Poslednji slučaj koji razmatramo je navalna greška sa dužinom većom od stepena polinoma $G(x)$. Na osnovu prethodne rasprave zaključujemo da je

$$\frac{E(x)}{G(x)} = \frac{x^i \times (x^{k-1} + \dots + 1)}{G(x)}$$

Međutim, pošto ovoga puta pretpostavljamo da je $k \geq 1 = r =$ stepen $G(x)$, moguće je da je $G(x)$ faktor $(x^{k-1} + \dots + 1)$. Kolike su šanse da će se ovo desiti? Razmotrimo najpre slučaj $k - 1 = r$. Pošto je stepen polinoma $G(x)$ takođe r , onda činjenica da je $G(x)$ faktor $(x^r + \dots + 1)$ znači da je $G(x) = (x^r + \dots + 1)$. Sada članovi između $x^r + 1$ definišu bitove na kojima je došlo do grešaka. Pošto postoji $r - 1$ takvih članova, postoji 2^{r-1} mogućih kombinacija oštećenih bitova. Ako pretpostavimo da se sve kombinacije javljaju sa podjednakom verovatnoćom, postoji verovatnoća od $1/2^{r-1}$ da kombinacije tačno odgovaraju članovima polinoma $G(x)$. Drugim rečima, verovatnoća da neke greške neće biti detektovane iznosi $1/2^{r-1}$.

Slučaj za $k - 1 > r$ je složeniji i ovde ga nećemo analizirati. Međutim, moguće je pokazati da verovatnoća da neke greške neće biti detektovane iznosi $1/2^r$. U referencama [Pe72] i [Mo89] možete da pronađete detaljniju analizu kodova za detekciju grešaka.

CRC se široko koristi u lokalnim mrežama (LAN mrežama), gde postoje standardni polinomi za $G(x)$, poput sledećih:

CRC-12:	$x^{12} + x^{11} + x^3 + x^2 + x + 1$
CRC-16:	$x^{16} + x^{15} + x^2 + 1$
CRC-ITU:	$x^{16} + x^{12} + x^5 + 1$
CRC-32:	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11}$ $+ x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

U opštem slučaju, CRC je veoma efikasan ako se $G(x)$ ispravno izabere. Specifično, $G(x)$ može da se izabere tako da x nije faktor, ali $x+1$ jeste. U ovom slučaju CRC detektuje sledeće greške:

- \$ Sve navalne greške dužine r manje od stepena polinoma $G(x)$
- \$ Sve navalne greške koje utiču na neparan broj bitova
- \$ Sve navalne greške čija je dužina jednaka $r + 1$, sa verovatnoćom $(2^{r-1} - 1)/2^{r-1}$
- \$ Sve navalne greške čija je dužina veća od $r + 1$, sa verovatnoćom $(2^r - 1)/2^r$

Na primer, CRC-32 polinom detektuje sve navalne greške čija je dužina veća od 33, sa verovatnoćom $(2^{32} - 1)/2^{32}$. Ovo je ekvivalentno 99,9999998 odsto tačnosti. Nije loše.

Implementacija CRC-a pomoću cikličnih pomeranja

Pronalaženje metoda za detekciju grešaka sa zadovoljavajućom tačnošću predstavlja samo jednu polovinu "bitke". Drugu predstavlja pronalaženje načina za efikasnu implementaciju. Ako uzmete u obzir bezbroj okvira koji se prenose preko mreže, shvatićete da je efikasna implementacija od suštinskog značaja.

Nakon što savladate osnovne principe CRC-a, Vaša prva reakcija može da bude pisanje programa koji vrši deljenje polinoma. Međutim, dok se takav program izvrši, verovatno će pristći još nekoliko okvira. Dok se provere svi ti okviri, verovatno će pristći još veći broj okvira i nastiće pravo "usko grlo". To bi bilo isto kao da kasirka traži proveru cene za svaki otkucani artikl; u međuvremenu, iza Vas bi se stvorio dugačak red ljudi sa zajedljivim komentarima, a istopili bi se i svi sladoledi koje ste kupili!

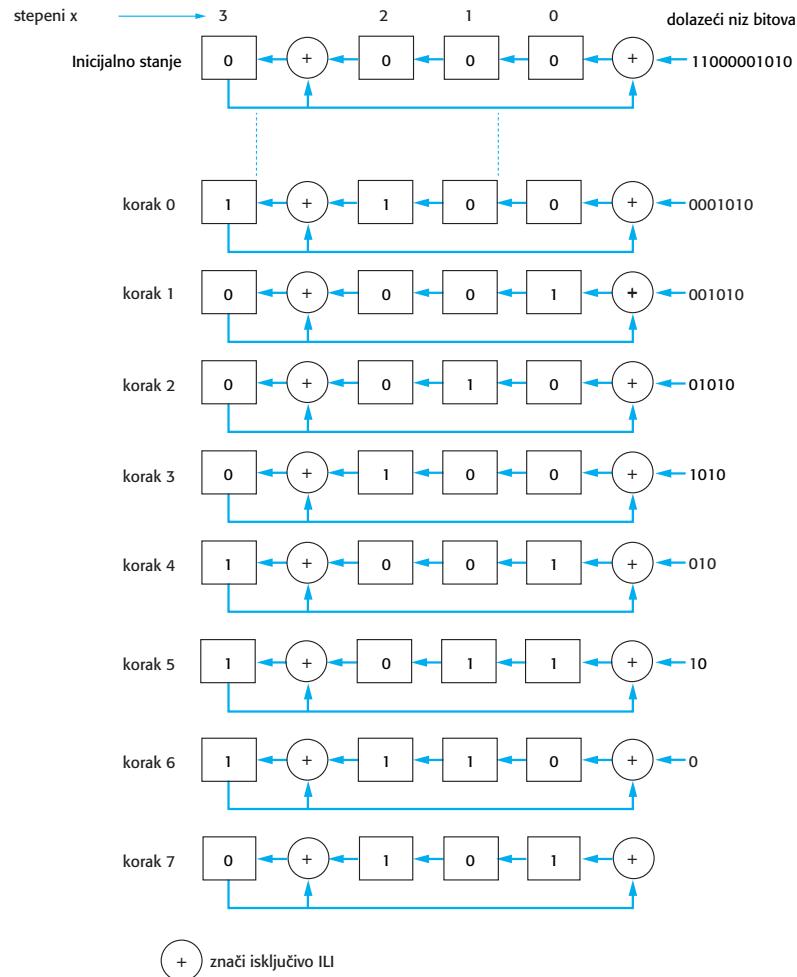
Postoji li način da podelimo dva polinoma i brzo dobijemo ostatak? Da li uopšte moramo da prolazimo kroz kompletan proces deljenja ako nam je potreban samo ostatak? Količnik se nigde ne koristi. Pogledajmo pažljivije sliku 6.5, na kojoj je prikazano sintetičko deljenje. Ceo proces može da se zamisli kao sekvenca pomeranja i isključivih ILI operacija između delitelja i delova deljenika.

Česta implementacija CRC-a koristi kolo koje se konstruiše u zavisnosti od generatora polinoma $G(x)$. Pošto postoje standardni polinomi, ova kola mogu masovno da se proizvode. Kolo sadrži pomerački registar i izvodi operaciju isključivo ILI u skladu sa sledećim pravilima:

- $\$ G(x) = b_r x^r + b_{r-1} x^{r-1} + \dots + b_2 x^2 + b_1 x + b_0$ gde je b_i ili 0 ili 1, a $i = 0, \dots, r$. Broj bitskih pozicija u registru je r . Krajnja desna pozicija odgovara članu b_0 , a krajnja leva $b_{r-1} x^{r-1}$.
- $\$$ Isključivo ILI kolo se nalazi desno od bilo koje pozicije kojoj je pridružena vrednost b_i jednak 1.
- $\$$ Niz bitova se uvodi u registar jedan po jedan, počevši od krajnje desne pozicije.
- $\$$ Kada se uvede novi bit, svi koji se već nalaze u registru pomeraju se uлево за jednu poziciju. Svaki bit se propušta kroz isključivo ILI kolo gde postoji, formirajući jedan operand za operaciju isključivo ILI.
- $\$$ Bit sa krajnje leve pozicije se propušta kroz svako isključivo ILI kolo, formirajući drugi operand svake operacije isključivo ILI.

Na slici 6.6 prikazani su registar i kola isključivo ILI za polinom $G(x) = x^4 + x^3 + 1$. Primećujete simbol operacije isključivo ILI desno od pozicija koje odgovaraju članovima x^3 i 1, a nema ih desno od pozicija koje odgovaraju članovima x^2 i x . Inicijalno, u registru se nalaze sve nule.

Na ovoj slici prikazana su ista izračunavanja kao i na slici 6.5. U koraku 0 prvi bit dolazećeg niza (deljenik sa slike 6.5) pomeren je na krajnju levu poziciju u registru. U koraku 1 krajnji levi bit se propušta kroz svako isključivo ILI kolo i sve se pomera ulevо.

**SLIKA 6.6** Deljenje korišćenjem cikličnih pomeraja

Primećujete da su sadržaji registra identični rezultatu prve operacije isključivo ILI iz koraka 1 sa slike 6.5.

Svaki korak definiše isti proces pomeranja uлево и izvođenja operacije isključivo ILI. Sadržaji registra u svim koracima odgovaraju rezultatima slično označenih koraka sa slike 6.5. Kada bitovi iz dolazećeg niza budu propušteni kroz registar, u registru će se naći ostatak (korak 7 na slikama 6.5 i 6.6).

Detektovanje grešaka pomoću CRC-ja je tačan i široko korišćen metod. Može efikasno da se implementira, a potrebno vreme je proporcionalno dužini niza. Standardni generator polinomi omogućavaju hardversku implementaciju celokupnog metoda (u čipovima), čime je dalje poboljšana njegova efikasnost.

6.4 Hamingovi kodovi: Korekcija grešaka

Kao što smo ranije istakli, kada se greške detektuju, obično postoje dve opcije: ponovno slanje originalnog okvira i ispravljanje oštećenog okvira. Druga opcija pored metoda za detektovanje grešaka zahteva i precizno utvrđivanje bitova na kojima su se greške desile. Ovo nije moguće izvesti jednostavnom proverom parnosti.

Korigovanje jednostrukе greške

Metod koji je razvio R. W. Hamming uključuje kreiranje specijalnih kodnih reči na osnovu podataka koji se šalju. Hamingov kod zahteva umetanje višestrukih bitova parnosti u niz bitova pre nego što se pošalje. Bitovi parnosti proveravaju parnost na strateškim lokacijama. Ideja je sledeća: ako se bitovi promene, njihove pozicije određuju jedinstvene kombinacije grešaka prilikom provere parnosti. Kada se okvir pošalje, primalac ponovo izračunava bitove parnosti. Ako dođe do bilo koje greške, kombinacija grešaka može da pokaže koji je bit promenjen. Nakon toga, primalac može da postavi bitove na tačne vrednosti. Ova tehnologija je prilično česta prilikom adresiranja memorije i prenosa bitova iz registara u RAM i nazad.

Ilustrovaćemo primenu Hamingovog koda na najjednostavnijem slučaju detektovanjem i korigovanjem greške na jednom bitu. Pretpostavimo da okviri sadrže osam bitova. Označimo ih kao m^1 m^2 m^3 m^4 m^5 m^6 m^7 m^8 . Sledeći korak je definisanje bitova parnosti za proveru parnosti na selektovanim pozicijama. Nameću se logična pitanja koliko ćemo provera parnosti koristiti i koje su pozicije obuhvaćene jednom proverom.

Ako koristimo jednu proveru parnosti, ona će ili uspeti, ili neće uspeti. Na osnovu toga, možemo da zaključimo da se greška javlja, ili ne javlja. Ako koristimo dve provere tačnosti, moguća su četiri ishoda: obe će biti neuspšne, obe će biti uspšne, prva neće biti uspšna, a druga hoće, ili druga neće biti uspšna, a prva hoće. Ova četiri ishoda mogu da se koriste za predstavljanje četiri događaja: nema greške, ili postoji greška u jednom bitu na tri moguće pozicije. Pošto postoje više od tri bitske pozicije, dve provere parnosti nisu dovoljne.

U opštem slučaju, ako se koristi n provera parnosti, postoji 2^n mogućih kombinacija neuspeha i uspeha. Svakoj bitskoj poziciji moramo da pridružimo jedinstvenu kombinaciju koja će primaocu omogućiti analiziranje provera parnosti i donošenje zaključka o poziciji na kojoj je došlo do greške (ako je došlo do greške). Međutim, da bi se uzele u obzir sve bitske pozicije, potrebno nam je n tako da je 2^n veće od broja poslatih bitova. Osim toga, moramo da zapamtimo da svaka dodatna provera parnosti zahteva slanje još jednog bita.

U tabeli 6.1 prikazana je relacija između n i broja poslatih bitova, uz pretpostavku da se šalje 8-bitni okvir. Kao što je pokazano, ako se koriste četiri provere parnosti, postoji 16 mogućih kombinacija za uspšne i neuspšne provere parnosti.

Tabela 6.1: Broj kombinacija za uspešne i neuspešne provere parnosti u funkciji od n

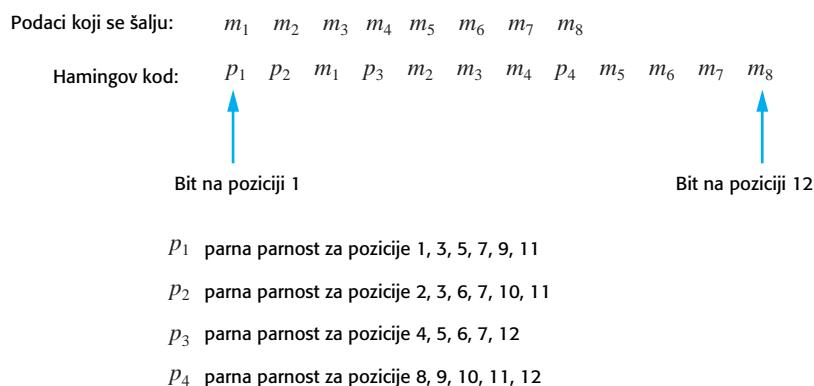
n (Broj provera parnosti)	Broj poslatih bitova	2^n (broj mogućih kombinacija uspešnih i neuspešnih parnosti)
1	9	2
2	10	4
3	11	8
4	12	16

Četiri dodatna bita parnosti sa osam originalnih bitova podrazumevaju stvarno slanje 12 bitova. Dakle, moguće je da će se desiti 13 različitih događaja: nema greške, ili greška postoji u jednom bitu na nekoj od 12 pozicija.

Sledeći korak je pridruživanje kombinacije jedinstvenom događaju. Da bi se to izvelo, konstruišu se četiri bita parnosti p^1, p^2, p^3 , i p^4 i umeću se u kadar na način koji je prikazan na slici 6.7. Svaki bit parnosti uspostavlja parnu parnost za selektovane pozicije naznačene na slici. Zašto se bitovi parnosti postavljaju na tim pozicijama? Kako da utvrdimo pozicije za sve provere parnosti?

Da bismo dali odgovore na ova pitanja, proučićemo pozicije uključene u svaku proveru parnosti. Prva provera uključuje sve bitove na neparnim pozicijama. Te pozicije, ako se zapišu u binarnom obliku, imaju 1 kao bit najmanje težine. Ako u binarnom obliku zapišete pozicije obuhvaćene drugom proverom, one će imati 1 na pozicijama bita druge najmanje težine. Slično tome, treća i četvrta provera imaju 1 na pozicijama koje odgovaraju bitovima treće i četvrte najniže težine, respektivno.

Kako nam ovo može pomoći? Kreirajte 4-bitni binarni broj koji se sastoji od b^4 , b^3 , b^2 , i b^1 gde je $b_i = 0$ ako je provera parnosti za p_i uspešna, a u suprotnom je $b_i = 1$ ($i = 1, 2, 3$, ili 4). U tabeli 6.2 prikazana je relacija između pozicija sa pogrešnim bitovima, nevalidnim proverama parnosti i 4-bitnim brojem.



SLIKA 6.7 Hamingov kod za jednostrukе greške

Tabela 6.2: Pozicije bitova sa greškom i pribrožene greške parnosti

Pozicija bita sa greškom	Netačna provera parnosti	b4, b3, b2 i b1
Nema greške	Ni jedna	0000
1	p ₁	0001
2	p ₂	0010
3	p ₁ i p ₂	0011
4	p ₃	0100
5	p ₁ i p ₃	0101
6	p ₂ i p ₃	0110
7	p ₁ , p ₂ , i p ₃	0111
8	p ₄	1000
9	p ₁ i p ₄	1001
10	p ₂ i p ₄	1010
11	p ₁ , p ₂ , i p ₄	1011
12	p ₃ , i p ₄	1100

Kao što je pokazano u tabeli, postoji poklapanje između 4-bitnog binarnog broja i pozicija sa pogrešnim bitom.

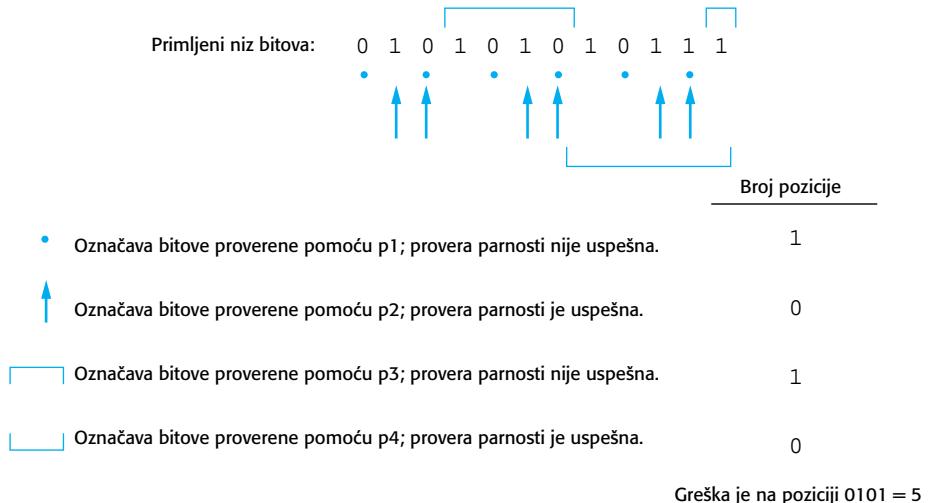
Kada primalac dobije preneti okvir, izvršava provere parnosti. Kombinacija neuspeha i uspeha određuje da li je došlo do greške i, eventualno, na kojoj se poziciji greška pojavila. Kada primalac zna gde se greška javila, on je ispravlja tako što menja vrednost bita na toj poziciji.

Ilustracije radi, razmotrite primer sa slike 6.8. Ovde vidimo da je inicijalni okvir bio 0110-0111, a da je preneti Hamingov kod 0101-1101-0111. Ako želite da se uverite da li bitovi parnosti daju parnu parnost na odgovarajućim pozicijama, morate sami da izvršite potrebna izračunavanja.

Na slici 6.9 pokazano je da je primljeni okvir 0101-0101-0111. Sada, ako izvršite proveru parnosti, videćete da su provere za p₁ i p₃ netačne, tj. postoji neparan broj jedinica na pozicijama 1, 3, 5, 7, 9 i 11 i na pozicijama 4, 5, 6, 7 i 12. Znači, prema tabeli 6.2, greška je u bitu 5. Pošto bit 5 ima vrednost 0, primalac je menja u 1 i okvir je ispravljen.

Podaci:	0	1	1	0	0	1	1	1
	m ₁	m ₂	m ₃	m ₄	m ₅	m ₆	m ₇	m ₈
Hamingov kod:	0	1	0	1	1	1	0	1
	p ₁	p ₂	m ₁	p ₃	m ₂	m ₃	m ₄	p ₄
								m ₅ m ₆ m ₇ m ₈

SLIKA 6.8 Niz bitova pre prenosa



SLIKA 6.9 Provera parnosti u okviru nakon prenosa

Korigovanje višestrukih grešaka

O kodovima za korekciju jednostrukih grešaka možemo da damo sličan komentar kao i o kodovima za detekciju jednostrukih grešaka. Greške u samo jednom bitu su retke prilikom razmene podataka. Takvi kodovi postoje, ali ih nećemo ovde predstaviti. Broj dodatnih bitova može da postane toliko veliki da se koriste samo u specijalizovanim slučajevima. Ako ste zainteresovani, raspravu o tim kodovima možete da pronađete u referencama [Ko78] i [Ha80].

Postoji još jedna klasa metoda za korekciju grešaka koju vredi pomenuti: Bose-Chaudhuri-Hocquenghem (BCH-iz očiglednih razloga) kodovi i Reed-Solomonovi kodovi, koji predstavljaju potklasu BCH kodova. Oba metoda koriste koncept kodnih reči, kolekcije od N bitova, iza koje sledi M kontrolnih bitova greške, koji se izračunavaju na osnovu bitova podataka. Kodna reč ima $N + M$ bitova i postoji 2^N mogućih kodnih reči. Samo su N bitovi proizvoljni, a preostalih M bitova je strogo određeno. To znači da, iako postoji 2^{N+M} mogućih bitskih sekvenci, samo 2^N tih kombinacija (manji broj u poređenju sa ukupnim brojem) predstavlja legitimne kodne reči. Na primer, ako je $N = 8$ i $M = 4$, postoji $2^8 = 256$ legitimnih kodnih reči od mogućih $2^{12} = 4096$ bitskih sekvenci.

Ključni koncept ovih kodova podrazumeva da postoji određeno *rastojanje* između dve kodne reči, tj. da postoje različiti bitovi u tim ključnim rečima. Na primer, dva Hamingova koda 0101-1101-0111 i 1001-1001-0111 imaju rastojanje 3, jer se razlikuju u tri bita (prvi, drugi i šesti bit). Svaki skup kodnih reči ima *minimalno rastojanje*. Da bi se ono utvrdilo, izračunavaju se rastojanja između svih parova kodnih reči. Najmanja izračunata vrednost predstavlja minimalno rastojanje tog skupa kodnih reči.

Minimalno rastojanje je značajno, zato što je direktno povezano sa brojem oštećenih bitova koje je moguće detektovati i ispraviti. U opštem slučaju, ako je d minimalno rastojanje, ovaj metod može da detektuje sve greške koje utiču na manje od d bitova (takva promena daje neispravnu kodnu reč) i može da ispravi sve greške koje utiču na manje od $d/2$ bitova.

Na primer, pretpostavimo da je minimalno rastojanje skupa kodnih reči $d = 10$. Znači, bilo koje dve legitimne kodne reči moraju da se razlikuju najmanje u 10 bitova. Greške koje utiču na manje od $d/2 = 5$ bitova mogu da se isprave. Da biste ovo videli na konkretnom primeru, pretpostavitićemo da je kodna reč poslata i da je došlo do greške u četiri bita. U tom slučaju, rezultat neće predstavljati validnu kodnu reč (da bi se kreirala validna kodna reč, mora da se promeni najmanje 10 bitova). Osim toga, smatraćemo da primalac pretpostavlja da će sve greške uticati na manje od pet bitova. Primalac neispravne kodne reči mora da pronađe najbližu legitimnu kodnu reč; kada je pronađe, zaključuje da je to ispravna poslata kodna reč. Sve druge kodne reči bi morale da imaju namanje šest pogrešnih bitova da bi podsećale na primljenu reč.

Naravno, trik je u tome da se kodne reči izaberu tako da minimalno rastojanje bude što veće. Na taj način se maksimizira broj bitova koje je moguće detektovati i ispraviti. BCH kodovi, a posebno Reed-Solomonovi kodovi, upravo to i rade. Ipak, za detaljnije opise ovih metoda neophodno je poznavanje poljāa Galoa (Galois), teorije o matricama i generator polinoma, tako da ćemo zainteresovane čitaocе uputiti na reference [Gr01], [Wi95], i [Sw02]. Međutim, ne dopustite da Vas složenost ovih metoda navede na pomisao da je reč o nekim apstraktnim teorijama visokog nivoa, koje imaju samo nekoliko primena. Kada sledeći put budete slušali muziku sa CD-a, ili gledali film sa DVD-ja, setite se da oni koriste prednosti Reed-Solomonove tehnologije.

6.5 Zaključak

U ovom poglavljiju su prvenstveno obradene dve teme: detektovanje i korigovanje grešaka nastalih prilikom prenosa informacija. Detekcija jednostavno podrazumeva utvrđivanje da li je došlo do greške; ako je došlo, onda se oslanja na druge protokole za ugovaranje dodatnih razmena da bi bile dobijene tačne informacije. Korekcija podrazumeva izvršavanje promena nakon što se podaci prime bez dodatnih prenosa.

Predstavljena su tri metoda.

- **Bitovi parnosti** Ovaj metod detekcije grešaka namenjen je prvenstveno detekciji jednostrukih grešaka, a navalne greške detektuje sa verovatnoćom 50 odsto. Međutim, može da bude koristan kada se bitovi prenose zasebno, kao kod nekih memorijskih arhitektura. Osim toga, predstavlja osnovu za tehnike korekcije grešaka.
- **Ciklična provera redundantnosti** Ovaj metod detekcije grešaka zasnovan je na teoriji deljenja polinoma. Nizovi bitova se interpretiraju kao polinomi. CRC bitovi su kreirani tako da, kada se poruka podeli sa generator polinomom, dobijate ostatak deljenja jednak nuli. Deljenjem primljene poruke sa generator polinomom i proverom ostatka deljenja sa velikom tačnošću može da se utvrdi da li je došlo do grešaka u toku prenosa. Ovaj metod se često koristi i lako se implementira korišćenjem cikličnih pomeračkih kola i registra. Određeni polinomi su proglašeni standardima.

- **Hamingov kod** Ovaj kod za korekciju grešaka uspostavlja kolekciju bitova parnosti na strateškim pozicijama. Ako dođe do jednostrukne greške, njegova pozicija utiče na jednistvenu kombinaciju provera parnosti. Ovo ne samo da omogućava detektovanje greške, već i utvrđivanje pozicije bita na kome je došlo do greške. Ako se zna pozicija greške, onda se lako može i otkloniti.

Dakle, šta je bolje: detektovanje, ili korekcija grešaka? Kao što ste mogli i da očekujete, odgovor je da ni jedno nije bolje, bar u opštem smislu. Tehnike za korekciju u opštem slučaju zahtevaju veće troškove i ne mogu u svim primenama da opravdaju svoje postojanje, ukoliko se greške retko javljaju. Obično je mnogo jeftinije jednostavno zatražiti ponovni prenos oštećenih informacija. Tipično, većina kompjuterskih mreža se ubraja u ovu kategoriju.

Ako se greške češće javljaju, dodatni troškovi zbog ponovljenih prenosa počinju da predstavljaju problem. U takvim slučajevima je možda jeftinije uključiti dodatne bitove korekcije, umesto da se medijum opterećuje preteranim redundantnim prenosima.

Učestalost pojave grešaka nije jedini faktor koji treba uzeti u obzir. I ponovno slanje okvira zahteva određeno vreme. Trajanje zavisi od brojnih faktora, kao što su količina saobraćaja, brzina prenosa podataka i rastojanje. U većini slučajeva, kratko kašnjenje kod prijema email poruka, ili fajla sa LAN servera nije "strašno", a ponekad nije ni primetno. Međutim, u real-time okruženjima kod kojih se poruke moraju isporučiti pravovremeno da bi se izbegle katastrofe, čak ni ovako mala kašnjenja nisu dopuštena. Real-time aplikacije, kao što su gledanje, ili slušanje multimedijalnih zapisa, ne mogu sebi da "priušte luksuz" ponovnog slanja oštećenih podataka. Podaci moraju da se vide, ili čuju čim se pošalju. Sonda za svemirska ispitivanja, kod kojih signali putuju po nekoliko sati do svojih odredišta, ozbiljno su hendikepirane ako se poruka mora ponovo preneti, posebno ako postoji velika verovatnoća da će se smetnje ponovo pojaviti. Zamislite astronauta koji kaže: "Halo, NASA, ne čujemo vas. Šta ste rekli u vezi pretećeg sudara sa vanzemaljskim brodom?".

Pitanja i zadaci za proveru

1. Šta je bit parnosti?
2. Koja je razlika između parne i neparne parnosti.
3. Koja je razlika između korekcije grešaka i detekcije grešaka.
4. Šta je navalna greška?
5. Da li su sledeće tvrdnje tačne, ili netačne (zašto)?
 - a. Nije neuobičajeno da se izgube jedan, ili dva bita u toku prenosa.
 - b. Iako je tačna tehnika, CRC je vremenski složena tehnika, jer zahteva dodatne troškove.
 - c. Generator polinom može da se izabere proizvoljno sve dok i pošiljalac i primalac znaju o kom polinomu je reč.
 - d. CRC će detektovati navalnu grešku proizvoljne dužine sve dok je broj bitova koji se menjaju neparan.
 - e. Kodovi za korekciju grešaka su efikasniji od kodova za detekciju grešaka, jer ne zahtevaju rentransmisiju podataka.
6. Šta je ciklična provera redundantnosti?

7. Pod kojim uslovima CRC detektuje sledeće greške?
 - a. jednostrukе greške
 - b. dvostrukе greške
 - c. navalne greške čija je dužina manja, ili jednaka stepenu generator polinoma
 - d. navalne greške čija je dužina veća od stepena generator polinoma
8. Koje uslove mora da ispuni generator polinom? Zašto?
9. Klasifikujte greške koje će CRC metod uvek detektovati.
10. Klasifikujte greške koje CRC metod neće detektovati.
11. Šta je pomerački registar?
12. Šta je Hamingov kod?
13. Definišite rastojanje između dve kodne reči.

Vežbe

1. Navedite argument koji pokazuje da jednostavna provera parnosti detektuje greške samo u slučajevima kada se promeni vrednost neparnog broja bitova.
2. Prepostavite da neki staticki elektricitet koji traje 0,01 sekundu utiče na komunikacionu liniju za 56 Kbps modem. Na koliko bitova ova smetnja može da utiče?
3. Prepostavite da se 128 bitova podataka proverava pomoću metoda za detekciju grešaka koji koristi čeksumu. Navedite primer koji pokazuje da je moguće da se promene vrednosti dva bita i da greške ne budu detektovane. Navedite još jedan primer koji pokazuje da je moguće promeniti tri bita podataka, a da greške ostanu nedetektovane. Da li je moguće da se promene svi bitovi, a da greške i dalje ne budu detektovane?
4. Zašto se za $0 - 1 = 1$ koristi oduzimanje po modulu 2?
5. Koji polinom odgovara sledećem nizu bitova?

0110010011010110

6. Koristeći metode opisane na slikama 6.2 i 6.3, izračunajte ostatak deljenja sledećih polinoma:
$$\frac{x^{12} + x^{10} + x^7 + x^6 + x^5 + x^3 + x^2}{x^7 + x^4 + x^2 + x^1}$$
7. Prepostavite da želite da pošaljete podatke 100111001 i da je generator polinom $x^6 + x^3 + 1$. Koji je niz bitova stvarno poslat?
8. Nacrtajte ciklični pomerački registar i isključiva ILI kola za CRC-12 i CRC-16 standardne polinome.
9. Koristeći ciklična pomeranja, izračunajte ostatak deljenja sledećih polinoma:

$$\frac{x^{12} + x^{10} + x^7 + x^6 + x^5 + x^3 + x^2}{x^7 + x^4 + x^2 + x^1}$$

10. Proučite dokumentaciju za LAN na Vašem univerzitetu, ili u Vašoj kompaniji i utvrdite koji se metod za detekciju grešaka (ako postoji) koristi.
11. Prepostavite da generator polinom ima član x kao faktor. Navedite primer greške koja neće biti detektovana.
12. Prepostavite da želite da generišete Hamingov kod za korekciju jednostrukih grešaka za 16-bitni niz podataka. Koliko je bitova parnosti neophodno? Šta se dogada ako se koristi 32-bitni niz podataka?
13. Primljeni su sledeći 12-bitni Hamingovi kodovi (za korekciju jednostrukih grešaka). Koje ASCII kodirano slovo predstavlja ovaj niz?

110111110010

14. Konstruišite Hamingove kodove za svako sledeće slovo: A, 0 i {
15. Prepostavite da pošiljalac ima sledeće okvire podataka:

Broj okvira	Podatak
1	0 1 1 0 1 0 0 1
2	1 0 1 0 1 0 1 1
3	1 0 0 1 1 1 0 0
4	0 1 0 1 1 1 0 0

Prepostavite da pošiljalac konstruiše Hamingov kod za svaki okvir, formirajući dvodimenzionalni niz (svaki red sadrži jedan Hamingov kod), i da šalje jednu po jednu kolonu. Šta primalac dobija ako se zbog greške u četvrtoj koloni prime sve nule? Primenite metod za korekciju grešaka na primljene podatke i ispravite ih.

16. Razvijte Hamingov kod koji može da ispravi sve jednostrukе greške i da detektuje sve dvostrukе greške u 8-bitnom nizu podataka.
17. Prepostavite da 4-bitni broj $b_4 b_3 b_2 b_1$, koji je opisan u tabeli 6.2, formira broj veći od 12. Šta to znači?
18. Napišite program koji uzima osam bitova podataka i kreira 12-bitni Hamingov kod.
19. Koliko iznosi minimalno rastojanje između dve kodne reči koje su definisane dodavanjem bita parne parnosti?
20. Koliko iznosi minimalno rastojanje Hamingovog koda definisanog u odeljku 6.4?
21. Prepostavite da se osam bitova dodaje na 32 bita radi kreiranja 40-bitne kodne reči. Koji procenat ukupnog broja 40-bitnih kombinacija predstavlja legitimne kodne reči?
22. Koliko bitova u okviru greške može da detektuje Hamingov kod?

Reference

[Gr01] Gravano, S. *Introduction to Error Control Codes*. Oxford and New York: Oxford University Press, 2001.

- [Ha80] Hamming, R. W. *Coding and Information Theory*. Englewood Cliffs, NJ: Prentice Hall, 1980.
- [Ko78] Kohavi, Z. *Switching and Finite Automata Theory*, 2nd ed. New York: McGraw-Hill, 1978.
- [Mo89] Moshos, G. *Data Communications: Principles and Problems*. St. Paul, MN: West, 1989.
- [Pe72] Peterson, W.W., and E.J. Weldon. *Error Correcting Codes*, 2nd ed. Cambridge, MA: MIT Press, 1972.
- [Sw02] Sweeney, P. *Error Control Coding: From Theory to Practice*. New York: Wiley, 2002.
- [Wi95] Wicker, S. *Error Control Systems for Digital Communications and Storage*. Englewood Cliffs, NJ: Prentice Hall, 1995.