Contents

1 Bevezetés

Bármely kommunikációs folyamat során az adótól a vevőhöz továbbítandó üzenet megsérülhet a kommunikációs csatorna okozta torzítások, zajok miatt. Ilyen sérülések jelzésére illetve javítására alkalmaznak csatornakódolást vagy más néven hibajavító kódolást. A kódolás szót más terminológiában is használják: titkosítás, forráskódolás, ASCII kódolás, vonali kódolás, kódolt moduláció, kódosztásos többszörös hozzáférés (CDMA). Tulajdonképpen mindegyik esetben egyfajta leképzés vagy megfeleltetés történik az üzenet és az annak megfelelő kód között.

Bináris üzenetek esetén, csatornakódolásnak (kódolásnak) hívjuk azt a folyamatot, amely során az átvitelre szánt bitfolyamot módosítva, extra bitekkel (redundanciával) kiegészítve egy olyan új bitfolyamot hozunk létre, amely kedvező tulajdonságokkal bír a vevő oldal szempontjából. A kódolás célja, hogy a csatorna hatása által okozott bithibákat detektálja. A vevőoldal, a kódolás ismeretében a vett bitfolyamból megpróbálja jelezni, hogy történt-e hiba az átvitel során. Komplexebb kódolások esetén a vevő nem csak a hibázás tényét tudja jelezni, hanem akár javítani is képest azt.

Miért alkalmazunk kódolást? A rendelkezésre álló erőforrások végesek. Kommunikációs szempontból megkötés számunk a rendelkezésre álló adatátviteli sávszélesség illetve a kibocsájtható teljesítmény. Kommunikációs rendszereknél általában követelmény egy adott bithibaarány elérése ($P_b = 10^{-5}$), vagy megbízhatóság (99.99%). Ha számunkra sem a sávszélesség sem a teljesítmény nem növelhető, kénytelenek vagyunk az adatátviteli sebesség csökkentésével elérni a kívánt bithibaarány javulást. Ezért alkalmazunk csatornakódolást.

Először néhány fontos kódolási alapfogalmat tárgyalunk, aztán rátérünk a különféle kódolási eljárások tárgyalására.

Az adatátvitel során történő kódolás egyszerűsített folyamata látható az 1.1.ábrán.

Figure 1.1: Kódolás és dekódolás

A bináris forrástól kapott **k** darab információs bitet (üzenetet) a kódoló leképez egy **n** ($n < k$) hosszúságú kódolt bitfolyamba (kódszóba). Ez a leképzés a **q** elemszámú kódábécé alapján történik (bináris esetben $q \in \{0, 1\}$). A kódszó és az üzenet hosszának arányát nevezzük kódaránynak: $\mathbf{R} = \mathbf{k}/\mathbf{n}$.

2 Hibadetektáló kódok

A hibadetektálás lényege, hogy bár a hibás bit helyét nem ismerjük, a vevő számára jelezze, hogy hiba történt az átvitel során az adott blokkban. A hiba érzékelése után egy automatikus újraküldés kérést (ARQ - Automatic Repeat reQuest) továbbít az adó felé, hogy a hibás blokkot küldje el ismételten. A következőkben ilyen hibadetektálási módszereket ismertetünk röviden.

2.1 Ismétlő kódok

A legegyszerűbb hibadetektálási módszer az ún. ismétlő módszer. Ennek a technikának egyszerű esete az előző fejezetben mutatott példa. Légyege, hogy minden hasznos adatot többször küldünk el az adótól a vevőhöz. Ha a többször elküldött adatsorok nem egyeznek meg, akkor tudhatjuk, hogy közülük valamelyik hibás. Nagy ismétlésszám és kis hibavalószínűség esetén a vevőben akár statisztikai alapon is megállapíthatjuk, hogy mi a helyes adatsor. Kevés ismétlés esetén viszont nem elhanyagolható esélye van annak, hogy többször is ugyanott hibázzon az átvitel, s így a többségi döntés elve alapján a hibás adatot jónak ítéljük. Nyilvánvalóan, minél többször küldünk el egy adatot, a csatorna hasznos átviteli kapacitása annál kisebb lesz - nagyszámú átküldés esetén drasztikusan lecsökken az adatsebesség - emiatt ezt a módszert csak a legegyszerűbb rendszerekben alkalmazzák.

2.2 Paritás kódok

Az ismétlő módszernél kissé összetettebb a paritásbitek módszere. Lényege, hogy az adatfolyamot meghatározott hosszúságú (általában 1-2 bájt hosszú) részekre bontjuk, majd ezekhez a részekhez egyenként fűzünk hozzá egy paritásbitet. A paritásbit értéke egy, ha az adott adatrészen belül páros számú 1 értékű bit van, egyébként nulla. A megoldás hátránya, hogy egy adatblokkon belül csak páratlan számú bithibát képes detektálni, mivel páros számú bithiba esetén a paritásjelző bit értéke megfelelő lesz. A paritásbites megoldás már sokkal kevesebb redundanciát vagyis extra információt ad hozzá az adatsorhoz, az ismétléses módszerhez képest. A leggyakoribb eljárás során minden bájthoz rendelnek egy paritásbitet. A paritásbites módszernek léteznek összetettebb változatai is. A paritásbittel ellátott adatrészek egyszeres vagy többszörös átlapolásával lehetővé válhat a hiba helyének pontosabb behatárolása vagy akár javítása is.

2.3 Ellenőrző-összegek

Ellenőrző-összeget számoló kódolás esetén az adatot nem bontják fel kisebb blokkokra, hanem nagyobb adatrészeket kezelnek egységként, és ezekhez rendelnek hozzá egy-egy több bitből álló sorozatot. Lehetséges algoritmus ilyen ellenőrző összeg elkészítésére az, ha az átvitt adatbájtokat binárisan összegezzük, majd az összeg néhány meghatározott számjegyét tekintjük ellenőrző összegnek. Előnye, hogy segítségével viszonylag nagy valószínűséggel lehet hibát detektálni kevés hozzáadott információ segítségével. Hátránya, hogy a nulla-értékű bájtokban fellépő, valamint egymást nullára kiegészítő hibák esetén ez a módszer nem jelzi a problémát, továbbá két bájt felcserélődése sem érzékelhető. E hátrányok kiküszöbölésére több módszert is kidolgoztak, amelyek nem csak az adatok valamilyen módon képzett összegét veszik figyelembe, hanem az egyes adategységek (bájtok) sorrendjét is. Ezek közül a legnépszerűbb az ún. ciklikus redundancia ellenőrzés (Cyclic Redundancy Check, CRC). Az ellenőrző összeg számításának elméleti alapja az, hogy az átvitt n-bites adatfolyamot egy n-ed fokú polinomként kezeljük. A polinomban adott fokszámú tag akkor szerepel, ha a neki megfelelő sorszámú bit értéke 1. Az így előállított polinomot elosztjuk egy előre meghatározott, általában sokkal kisebb fokszámú polinommal, amit generátor-polinomnak nevezünk. Az osztás végeredményét eldobjuk, a maradék lesz az adat ciklikus redundanciakódja, ezt kell a vevőhöz továbbítani. A vevőben szintén kiszámítjuk ezt a redundanciakódot, így a számított és kapott érték összehasonlítható. A CRC eljárás segítségével minden, a generátorpolinom fokszámánál kisebb egymást követő (burst-jellegű) bithiba detektálható. Minél hosszabb a használt adatsor és minél rövidebb a generátorpolinom, annál nagyobb a valószínűsége annak, hogy az ellenőrző összeg két különböző adatsor esetén megegyezik. Az ellenőrző összeg kiszámítására többféle módszert is alkalmaznak a gyakorlatban. A legegyszerűbb, de legnagyobb számításigényű a bitenkénti eljárás, ennél összetettebb, de sokkal gyorsabb az ún. keresőtáblás módszer.

3 Hibajavító kódok

A hibajavítás túlmutat a detektáláson, ebben az esetben nem csak azt szeretnénk tudni, hogy történt-e hiba, hanem azt is, hogy a vett kódszó melyik pozícióiban. A hibajavító kódolási módszereknek két nagy csoportja van: a lineáris blokk kódok, illetve a konvolúciós kódok. A hibajavító módszerek természetesen hibadetektáló módszerek is egyben.

3.1 Lineáris Blokk kódok

A blokk kódok lényegében egy k hosszúságú üzenetblokkhoz rendelnek hozzá egy n hosszúságú kódszót. Lineárisnak nevezünk egy kódot, ha annak két kódszavát összegezve ugyancsak egy, a kódábécéhez tartozó kódszót kapunk. Abban az esetben, ha a kódszó tartalmazza a k hosszúságú üzenetet, vagyis a kód az üzenetet csupán kiegészíti $n-k$ darab bittel, akkor szisztematikus blokk kódról beszélünk. A lineáris blokk kódoknak sok fajtája van, ebben a fejezetben a legfontosabb és legelterjedtebb kódokat mutatjuk be. A 3.1 ábra mutatja a blokk kódok legfontosabb fajtáit.

A lineáris blokk kódoknak két alcsaládja van: a ciklikus blokk kódok illetve a nem

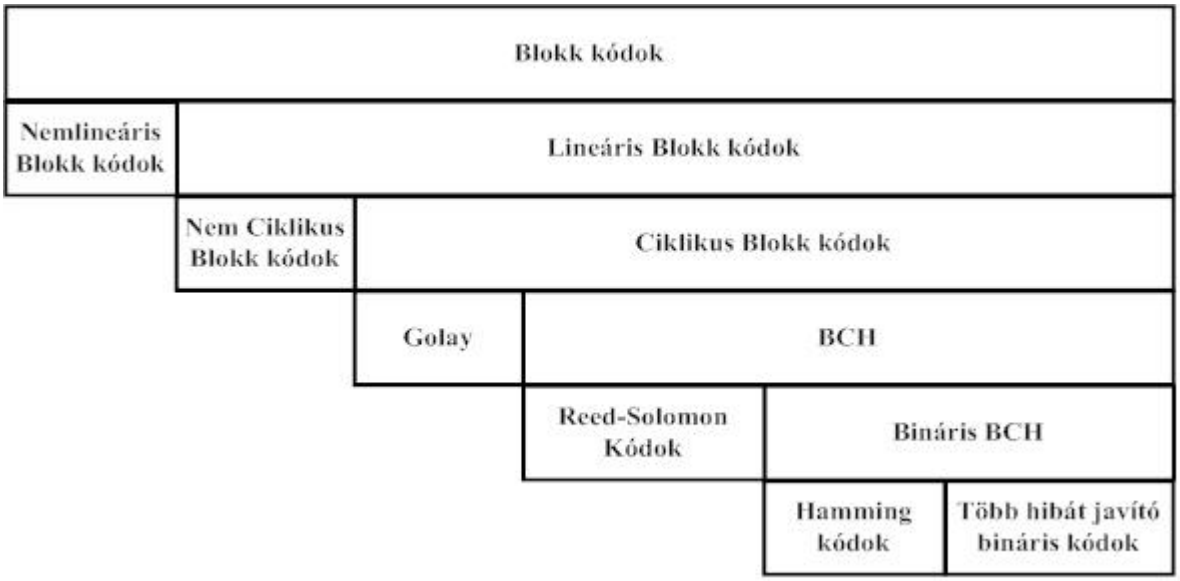

Figure 3.1: Blokk kódok fajtái

ciklikus blokk kódok. Kódok esetén a ciklikusság azt jelenti, hogy a kódszavak cirkuláris

eltoltjai is érvényes kódszavakat eredményeznek. A ciklikus blokk kódokat ugyancsak két családra lehet osztani: a BCH (Bose-Chaudhuri-Hocquenqhem) kódok és a Golay-kódok. A BCH kódok két típusát különböztetjük meg: a már jól ismert Reed-Solomon kódokat, amelyek nem binárisak, illetve a bináris BCH kódokat. A bináris BCH kódok tovább oszthatóak két fajtára: az egy hibát javító Hamming kódokra és a több hibát javító bináris kódokra. A következő fejezetekben először a lineáris kódok néhány alapvető tulajdonságát, a kódokkal kapcsolatos alapfogalmakat ismertetjük, majd rátérünk az egyes kódosztályok rövid ismertetésére.

Hamming távolságnak hívjuk két bináris sorozat esetén azon pozíciók számát, amelyekben eltérnek egymástól. Pl. [0 0 1 1 1] és [0 1 0 1 1] sorozat Hamming távolsága 2, mivel a második és harmadik pozícióban térnek el egymástól. A dekódolás folyamata során, a vett **c'** bináris sorozathoz megpróbáljuk megkeresni a Hamming távolságban hozzá legközelebbi **c** kódszót, majd a következő lépésesben a **c** kódszó alapján a küldött **b** bináris információsorozatot meghatározni.

Egy adott kód **kódtávolsága** (d_{min}) alatt egy adott kódhoz tartozó **M** darab lehetséges kódszó közötti minimális Hamming távolságot értjük. Egy adott n hosszúságú kód kódábécéje **q** elemből áll, akkor a lehetséges kódszavak száma ennek megfelelően q^n. Egy adott kódot az n, k, d_{min}, kódparaméterekkel lehet jellemezni mint (n,k,d_{min}) paraméterű kód. Egyszerű hibázás esetén belátható, hogy a javítható hibák száma a kódtávolság d_{min} függvényében a $(d_{min}-1)/2$ kifejezés egészrésze. Egy másik hibázási mód lehet, amikor tudjuk, hogy egy pozícióban hiba lehet, és többi helyen nem történt hiba. Ezt nevezzük törléses hibának. Belátható, hogy törléses hiba estén $d_{min}-1$ hiba javítható.

Egyes rendszerben alkalmaznak ún. átszövést, mivel tapasztalat alapján a bithibák sok esetben nem egyesével, elvétve lépnek fel, hanem csomókban, ezért küldés előtt a biteket "összekeverik". Így, a vevőben a bitek újrarendezésével (az átszövés visszarendezésével) a csomókban fellépő bithibák eloszlanak a vett adatblokkban.

A dekódolás folyamata nem egyértelmű folyamat. Kisméretű kódszavak esetén megoldható táblázatos keresés alapján, azonban nagyobb kódok esetében ez már nem kifizetődő megoldás. Gyorsabb megoldásokat, röviden, az adott kódok tárgyalásánál fogunk bemutatni.

3.1.1 Konstrukciós szabályok, határok

Javítható és jelezhető hibák

Egy adott kód esetén ha ismerjük a minimális kódtávolságot, akkor garantáltan képesek leszünk $d_{min}-1$ hibát jelezni:

$$t_{jel} = d_{min} - 1. \tag{3.1}$$

mivel d_{min} hiba esetén már egy másik, érvényes kódszóba mennénk át. A javítható hibák számához pedig az szükséges, hogy ha hibázunk, akkor se kerüljünk közelebb egy másik kódszóhoz. Ehhez pedig az szükséges, hogy **t** hibázás esetén is fenn álljon a következő

összefüggés:

$$t_{jav} = \left\lfloor \frac{d_{min-1}}{2} \right\rceil, \tag{3.2}$$

ahol $\lfloor \cdot \rceil$ az egészrész függvényt jelenti.

Singleton korlát

Egy adott n hosszúságú kódhossz és d_{min} kódtávolság mellett nem lehet akármilyen méretű kódot generálni. A Singleton korlát szerint a lehetséges kódszavak száma

$$M \leq q^{n-d_{min}+1} \tag{3.3}$$

Bizonyítás: a k hosszúságú üzenetek száma legfeljebb q^k lehet

$$M \leq q^k \tag{3.4}$$

Ha az egyenlőség fennáll akkor az összes kódszót felhasználtuk. Egy újabb elemmel bővítve az üzenetünket, a kódábécé elemeinek számát q szorására növeljük. k elemből n elemre bővítés esetén q^{n-k}-szor több elemünk lesz. A bővített ABC Hamming távolsága legfeljebb n-k értékével bővült, az eredeti 1-hez képest. Vagyis

$$d_{min} \leq n - k + 1 \rightarrow k \leq n - d_{min} + 1 \tag{3.5}$$

Amit visszahelyettesítva (3.4)-be

$$M \leq q^k \leq q^{n-d_{min}+1} \tag{3.6}$$

Egyenlőség esetén a kódot **maximális távolságú** kódnak nevezzük.

Hamming korlát

A Hamming korlát arra a kérdés ad választ, hogy t hiba javításához mekkora n és k szükséges. Adottak a kódszavaink, egy hiba esetén a kódszavaink $n(q-1)$ új kódszóba juthatnak át, két hiba esetén $\binom{n}{2}(q-1)(q-1)$ új kódszó kell, hogy ne hibázzunk. Vagyis ezt általánosítva t hibára:

$$1 + n(q-1) + \binom{n}{2}(q-1)(q-1)... = \sum_{i=0}^{t} \binom{n}{i}(q-1)^i \tag{3.7}$$

Ha a forrás ABC nem egyezik a kód ABC elemszámával, akkor

$$m^k \sum_{i=0}^{t} \binom{n}{i}(q-1)^i \leq q^n. \tag{3.8}$$

Azonos méretű forrás és kód ABC esetén m^k és q^n összevonható, így megkapjunk a hamming korlátot:

$$\sum_{i=0}^{t} \binom{n}{i} (q-1)^i \leq q^{n-k} \tag{3.9}$$

Jól látható, hogy ez az összefüggés nem explicit fejezi ki a paramétereket, csupán arra ad lehetőséget, hogy a kód konstrukciós szabályainak kitalálása után meghatározzuk, hogy elértük-e az elvi maximumot a kódunkkal, vagy lehet-e ennél hatékonyabb kódot generálni. Ha a Hamming-korlát teljesül, vagyis a fenti képletben az egyenlőség áll fenn, akkor a kódot **perfekt** kódnak nevezzük. Bináris kódok ($q = 2$) esetén a következőképpen módosul a képlet:

$$\sum_{i=0}^{t} \binom{n}{i} \leq 2^{n-k}. \tag{3.10}$$

3.1.2 Példák

1. Példa: Ismétléses kódok
Ez talán a legegyszerűbb kódolási módszer. Egy információs bitet többször ismételve küldünk. Kétszres ismétlésre látható egy példa a 3.1 táblázatban.

Table 3.1: Ismétléses kódok

Bináris üzenet		Kódszó		
b	$\longrightarrow$	c_1	c_2	c_3
1	$\longrightarrow$	1	1	1
0	$\longrightarrow$	0	0	0

Jól látható, hogy a kód két kódszóból áll: 111 és 000. A kód $k = 1$ információs bithez $n = 3$ hosszúságú kódszót rendel, vagyis a kódarány: $R = 1/3$. Ezen kód Hamming kódtávolsága 3, mivel csak két kódszóból áll és azok között a Hamming távolság 3. A kód legfeljebb két hibát képes jelezni, mivel ilyen esetekben a vett kódszavak nem érvényes kódszavak. Ezen felül, ez a kód képes 1 hibát javítani is, mivel ilyenkor a vett szó az elküldött kódszótól 1 Hamming távolságra van, míg a másik kódszótól 2 a távolsága. Páratlan számú ismétlés esetén a kód $(n-1)/2$ hibát képes javítani és $n-1$ hibát jelezni. Látható az is, hogy a kód lineáris és ciklikus.

2. Példa: Paritás kód
Paritás kód esetén egy üzenetblokkban lévő bináris 1-esek számának megfelelően látja el egy extra bittel. Ha az egyesek száma páros, akkor 0-val, ha pedig páratlan, akkor 1-essel egészíti ki a blokkot. Vegyünk egy egyszerű példát, amikor az üzenetblokk két bitből áll, ez látható a 3.2 táblázatban. Látható, hogy a kód lineáris és ciklus, kódaránya $R = 2/3$, 4 kódszót használ amelyek Hamming távolsága 2. A Hamming távolságból adódóan, ezzel a kóddal hibát javítani nem tudunk, csupán egy hibát detektálni.

Table 3.2: Paritás kódok

Bináris üzenet			Kódszó		
b_1	b_2		c_1	c_2	c_3
0	0	$\longrightarrow$	0	0	0
0	1	$\longrightarrow$	0	1	1
1	0	$\longrightarrow$	1	0	1
1	1	$\longrightarrow$	1	1	0

3.1.3 Alapfogalmak lineáris blokk kódok esetén

Ahhoz hogy a kódalkotást, kódolási folymatot matematikailag tárgyalni tudjuk, néhány matematikai fogalmat be kell vezetnünk. Egy adott kód esetén az n elemű c kódszó kifejezhető a k elemű u üzenetvektor segítségével egy mátrixszorzás formájában:

$$c = Gu, \tag{3.11}$$

ahol G a kód $n \times k$ generátormátrixa. Szisztematikus kódok esetén a generátor mátrix egyértelműen meghatározható:

$$G = [I_k, B], \tag{3.12}$$

ahol I_k egy $k \times k$ méretű egységmátrix (olyan mátrix, melynek főátlójában csupa 1-es elem van, a többi helyen 0-s elemek szerepelnek), B pedig egy $k \times n-k$ méretű mátrix. Vagyis egy u üzenet esetén a kódszó így épül fel:

$$c = [u_0, u_0, ..., u_{k-1}, c_k, c_{k+1}, ..., c_{n-1}]. \tag{3.13}$$

A c vektor első k elemét üzenetszegmensnek, az utolsó $n-k$ elemét pedig paritásszegmensnek nevezzük. Minden kódhoz, a G generátormátrixán kívül definiálunk még egy $n-k \times n$ méretű H mátrixot, amelyet paritásellenőrző mátrixnak hívunk és a következő tulajdonsággal bír:

$$Hc^T = 0. \tag{3.14}$$

Ezen H vektor segítségével meg tudjuk állapítani, hogy a vett kódszó valóban kódszó-e. Továbbá egy kód H és G mátrixára igaz a következő tulajdonság:

$$HG^T = 0 \tag{3.15}$$

Ezen tulajdonság könnyen bizonyítható a következő átalakításokkal, (3.11)-t behelyettesítve (3.14)-be :

$$Hc^T = H(uG)^T = HG^T u^T = 0, \tag{3.16}$$

mivel u^T nem lehet nulla, az egyelőség csak akkor állhat fenn, ha $HG^T = 0$.
Egy kód paritásellenőrző mátrixa könnyen előállítható a generátormátrix ismeretében. Legyen H mátrix alakja a következő:

$$H = [A, I_{n-k}]. \tag{3.17}$$

Ezt az alakot behelyettesítve a (3.18) képletbe, illetve G mátrixot is feloldva az (3.12) egyenlet segítségével, a következő alakot kapjuk:

$$HG^T = [A, I_{n-k}][I_k, B]^T = A + B^T = 0, \tag{3.18}$$

azaz $A = -B^T$, amely bináris esetben $-B^T = B^T$.
Ezen vektorok, valamint mátrixok tulajdonságai nagyban elősegítik a dekódolás folyamatát. A vett kódszó és a küldött kódszó különbségvektorát, mint hibavektort definiáljuk a következőképpen:

$$e = v - c. \tag{3.19}$$

A H mátrix segítségével képesek vagyunk a dekódolásra. Ha a vett kódszót megszorozzuk a H mátrixszal, a következő kifejezésre jutunk:

$$Hv^T = H(c + e)^T = Hc^T + He^T = He^T. \tag{3.20}$$

Mivel korábban megmutattuk, hogy $Hc^T = 0$, így Hv^T értéke csak a hibavektortól függ, a küldött kódszótól nem. A dekódoláshoz használt szindrómavektort a következőképpen definiáljuk:

$$s = eH^T. \tag{3.21}$$

A dekódolási módszerek közül a leggyakoribb a szindrómadekódolás. Első lépésben kiszámítjuk a vett v vektorból a szindrómát. A szindróma alapján becslést tudunk adni a hibavektorra, majd ezt levonva a vett kódszóból, a küldött kódszóra is.

3.1.4 Alapfogalmak lineáris ciklikus blokk kódok esetén

A lineáris ciklikus blokk kódok tulajdonsága, hogy ha a kódszavaikat cirkulárisan eltoljuk, ugyancsak egy érvényes kódszót kapunk. A ciklikus kódok tárgyalásához elengedhetetlen fogalom a véges test fogalma. Végest testnek, vagy Galois-térnek nevezzük azokat a q elemet tartalmazó halmazokat, melyek elemeire igazak a következő tulajdonságok:

- Két műveletet definiálunk az elemek között: összeadás és szorzás. A két elem közötti művelet esetén az eredmény ugyancsak eleme a véges testnek. Másképpen fogalmazva a test a műveletekre nézve zárt (nem vezet ki a testből).
- Egy test minden esetben tartalmazza az egységelemet és a nullelemet. Az egységelem esetén, egy másik "a" elemmel való szorzás esetén az "a" elemet kapjuk eredményként. Nullelem esetén pedig, egy másik "a" elemmel való összeadás esetén szintén az "a" elemet kapjuk: $a + 0 = a$.

- Minden elem esetén létezik egy inverz elem, szorzás (a^{-1}) illetve összeadás műveletére ($-a$). Az elemet összeszorozva az inverz elemmel az egységelemet kapjuk. Összeadás esetén egy elemet összeadva annak inverz elemével a nullelemet kapjuk.
- A műveletekre érvényesek az asszociativitás, a kommutativitás és disztributivitás tulajdonságai.

A q elemű véges testet GF(q)-val jelöljük. A véges testek elemszáma prímszám vagy prímszám valamely hatványa. Matemikailag felírva:

$$q = p^n, \tag{3.22}$$

ahol p prímszám. Ha q prímszám, akkor a véges test elemei a $0, 1 \ldots q-1$ egész számok, és a szorzás illetve összeadás művelete megegyezik a szokványos összeadás illetve szorzás modulo q műveletekkel (modulo aritmetika eredménye a q-val való osztás maradéka). Példaként tekintsük a GF(5) test esetén a szorzás és összeadás műveletét. A GF(5) test elemei a 0,1,2,3,4 számok.

Table 3.3: Szorzás GF(5)-ben

$*$	0	1	2	3	4
0	0	0	0	0	0
1	0	1	2	3	4
2	0	2	4	1	3
3	0	3	1	4	2
4	0	4	3	2	1

Table 3.4: Összeadás GF(5)-ben

$+$	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	0
2	2	3	4	0	1
3	3	4	0	1	2
4	4	0	1	2	3

Minden GF(q)-beli nem nulla elem esetén létezik egy legkisebb m természetes szám, amit az adott elem rendjének hívunk, melyre igaz, hogy:

$$a^m = 1 \bmod q. \tag{3.23}$$

Azt az α elemet, amelynek rendje $q-1$, a GF(q) a test primitívelemének nevezzük. Ez GF(5) esetén:

Table 3.5: GF(5) elemeinek rendje

Elem	Hatványai	Rendje
1	1	1
2	2,4,3,1	4 (primív elem)
3	3,4,2,1	4 (primív elem)
4	4,1	2

A későbbiekben a primitív elem fontos szerepet fog játszani a kódalkotásban.
Az üzenetblokk és a kódblokk eddig használt vektoriális megjelenítése helyett bevezetjük a polinomiális ábrázolást:

$$u = [u_0, u_1, \ldots, u_{k-1}] \rightarrow u(x) = u_0 + u_1 x + u_2 x^2 + \cdots + u_{k-1} x^{k-1} \tag{3.24}$$

Ennek megfelelően egy kódszó a következőképpen néz ki:

$$c(x) = [c_0, c_1, \ldots, c_{n-1}] \rightarrow c(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1} \tag{3.25}$$

Ciklikus kódok generálása a $g(x)$ generátorpolinom segítségével történik:

$$c(x) = g(x)u(x). \tag{3.26}$$

Minden n hosszúságú lineáris, ciklikus kód esetén, annak $g(x)$ polinomjára igaz, hogy osztója az $x^n + 1$ kifejezésnek. Egy $g(x)$ generátorpolinomhoz tartozó lineáris, ciklikus kód esetén a $h(x) = \frac{x^n-1}{g(x)}$ polinomot paritásellenőrző polinomnak hívjuk. A kódszavak cikuláris tulajdonsága miatt, ha $c(x)$ kifejezést megszorozzuk x-el, akkor ismét kódszót kell hogy kapjunk:

$$xc(x) = c_{k-1}(x^n - 1) + xc_0 + c_1 x^2 + c_2 x^3 + \cdots + c_{n-2} x^{n-1} + c_{n-1} \tag{3.27}$$

Ebből következik, hogy

$$xc(x) \bmod (x^n - 1) = c_{n-1} + c_0 x + c_1 x^1 + \ldots c_{n-2} x^{n-1} \tag{3.28}$$

Mivel $c(x) = g(x)u(x)$ ezért,ahhoz hogy a paritásellenőrző mátrixxal történő szorzás után nulla maradékot kapjunk a következő egyenlőségnek kell fent állnia:

$$g(x)h(x) = 0 \mod (x^n - 1) \tag{3.29}$$

Vagyis generátorpolinómnak az $x^n - 1$ kifejezés polinomális felbontásának valamely elemnét használhatjuk. Például a GF(2) feletti polinómok esetén: $x^3 - 1 = (x+1)(x^2+x+1)$ vagy a $x^7 - 1 = (1+x)(1+x+x^3)(1+x^2+x^3)$. A második példa a (7,4)-es Hamming kódot adja ha a második tagot vesszük generátorpolinómnak. A ciklikus kódszavak szisztematikus előállítása viszonylag egyszerű: Először vegyük a korábbi üzenetblokk polinomiális ábrázolásának módosított verzióját:

$$u = [u_0, u_2, \ldots, u_{k-1}] \rightarrow u(x) = u_1 x^{k-1} + u_2 x^{k-2} + \cdots + u_{k-1} x + u_k \tag{3.30}$$

és képezzük a c(x) kódszavakat a következőképpen:

$$c(x) = u(x)x^{n-k} - [u(x)x^{n-k} \bmod g(x)] \tag{3.31}$$

Ennek következtében az előállított $c(x)$ biztosan kódszó lesz, mert a $g(x)$-el való osztás eredménye 0, így, a vevőben az ellenőrzés a vett kódszó $g(x)$-el való osztásával történik. Ha az eredmény 0, akkor nem történt hiba és az üzenetblokk a kódszó első k eleme. Jól látható, hogy a $c(x)$-et definiáló egyenlőség jobb oldának első tagja az üzenetszegmenst, a második tagja pedig a paritásszegmenst definálja.

3.1.5 Golay kódok

A Golay kódok olyan speciális lineáris, ciklikus blokk kódok, amelyek egyben perfekt kódok is. bináris golay-kód paraméterei a következők: $n = 23, k = 12, d_{min} = 7$. A bővített bináris Golay kód esetén pedig $n = 24, k = 12, d_{min} = 8$. A bővített Golay kód a perfekt bináris Golay kódból származtatható egy paritásbit hozzáadásával. Jól látható, hogy mindkét kód 3 hibát képes javítani. Ilyen kódokat a Voyager 1 és 2 űrhajó színes képeinek továbbításánál alkalmazták, mivel viszonylag magas adatsebességet lehet elérni. Generátorpolinomja:

$$g(x) = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{11}. \tag{3.32}$$

3.1.6 BCH kódok (Reed-Solomon, Hamming)

A BCH kódok lineáris ciklikus blokk kódok, amelyeket egyszeres vagy többszörös hiba javítására használjuk. A BCH kódokon belül nagyon fontos osztályt képeznek a nem bináris Reed-Solomon kódok, és a bináris kódok, azon belül is a külön tárgyaljuk a Hamming kódokat.

Reed-Solomon kódok

Több hibát javító, nembináris, lineáris, ciklikus kódok esetén többnyire Reed-Solomon kódokat használnak. Előnye, hogy maximális távolságú kód valamint a dekódolásra effektív módszereket dolgoztak ki. Reed-Solomon kódoknál n hosszúságú kódszó esetén a generátorpolinom a következőképpen írható fel:

$$g(x) = \prod_{i=1}^{n-k} (x - \alpha^i) \tag{3.33}$$

ahol α a GF(q) test primitíveleme. Egy tetszőleges (n,k,q) paraméterű Reed-Solomon kód esetén a konstrukció a következőképpen zajlik: egy k hosszúságú üzenetet mint polinomot tekintve:

$$u = [u_0, u_1, \ldots, u_k] \rightarrow u(x) = u_0 + u_1 x + \cdots + u_{k-1}x^{k-1}, \tag{3.34}$$

ebből a kódszó komponenseit a következőképpen kapjuk:

$$c_0 = u(\alpha^0), c_2 = u(\alpha^1), \ldots, c_{n-1} = c(\alpha^{n-1}), \tag{3.35}$$

ahol α a GF(q) primitív eleme. Vagyis egy Reed-Solomon kód generátormátrixa a következőképpen írható fel:

$$G = \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{k-1} & \alpha^{2(n-1)} & \ldots & \alpha^{(k-1)(n-1)} \end{bmatrix} \tag{3.36}$$

A paritásellenőrző mátrixa pedig:

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \ldots & \alpha^{n-1} \\ 1 & \alpha^2 & \alpha^4 & \ldots & \alpha^{2(n-1)} \\ 1 & \alpha^3 & \alpha^6 & \ldots & \alpha^{3(n-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \alpha^{n-k} & \alpha^{2(n-k)} & \ldots & \alpha^{(n-k)(n-1)} \end{bmatrix} \tag{3.37}$$

Jól látható, hogy az így konstruált G generátormátrix nem szisztematikus kódot határoz meg, vagyis a kódszó nem tartalmazza az üzenetvektort. RS kódok esetén a G generátormátrixból ún. Gauss-eliminációval segítségével könnyen generálhatunk szisztematikus kódot. Erre lássunk most egy példát. Legyen egy Reed-Solomon kód a következő paraméterekkel $k = 2, n = 4, q = 5$. Mint már korábban beláttuk, GF(5) esetén $\alpha = 2$ primitívelem. A kód generátormátrixa a következőképpen adható meg:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 3 \end{bmatrix} \tag{3.38}$$

Gauss-eliminációval (sorokat egymásból kivonva) átalakítva:

$$G' = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -2 & -1 \\ 0 & 1 & 3 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 3 & 4 \\ 0 & 1 & 3 & 2 \end{bmatrix} \tag{3.39}$$

A kód paritásellenőrző-mátrixa kifejezhető:

$$H = \begin{bmatrix} 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \end{bmatrix} \tag{3.40}$$

Legyen az üzenetvektor:

$$u = [1\ 2\ ] \tag{3.41}$$

Ekkor a küldött c kódszó kifejezhető

$$c = uG = [1\ 2\ 4\ 3]. \tag{3.42}$$

Látható, hogy a kódszó eleje tartalmazza az üzenetet, az utolsó két érték pedig paritásellenőrzésre szolgál. A kód, a paramétereiből adódóan 2 törléses hibát, és egy egyszerű hibát képes javítani. Nézzük meg egy egyszerű hiba javításának menetét. A hiba történjen az első helyen, vagyis az 1-es érték helyett a vevőbe érkezzen egy 4-es érték. Mátrixszorzás segítségével kiszámolva a szindrómavektort:

$$s = vH^T = [4\ 2\ 4\ 3] \begin{bmatrix} 1 & 2 & 4 & 3 \\ 1 & 4 & 1 & 4 \end{bmatrix}^T = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{3.43}$$

Természetesen a szindrómavektor elemeit kiszámolhatjuk polinomiálisan is:

$$s_1 = v(\alpha) = c(\alpha) + e(\alpha) = 0 + e(\alpha) = \quad e\alpha^i \tag{3.44}$$

$$s_2 = v(\alpha^2) = c(\alpha^2) + e(\alpha^2) = 0 + e(\alpha^2) = \quad e\alpha^{2i} \tag{3.45}$$

Ha nem lenne hiba, akkor a szindrómavektor mindkét értéke nulla lenne. Ebből a következő egyenletrendszer adódik:

$$3 = e\alpha^i = \quad e2^i \tag{3.46}$$

$$3 = e\alpha^{2i} = \quad e2^{2i} \tag{3.47}$$

ahol i a hiba helye és e a hiba értéke. A hiba helyét megkaphatjuk, ha a két egyenletet elosztjuk egymással:

$$1 = 2^i \rightarrow i = 0 \tag{3.48}$$

vagyis az első helyen történt a hiba, mivel az a polinom 0 kitevővel rendelkező tagja. A hiba értéke visszahelyettesítéssel számolható:

$$3 = e2^0 \rightarrow e = 3 \tag{3.49}$$

Ezt kell levonnunk a kapott értékből, vagyis a valós érték a 4 helyett az 1 lesz, ami megfelel a küldött üzenetnek.

Hamming kódok

A Hamming kódolás viszonylag egyszerű kódolási mód. Ez egy olyan perfekt bináris, lineáris, ciklikus blokk kód, amely pontosan egy hibát képes javítani és kettőt észlelni. A Hamming kódok kódparaméterei a következők: $n = 2^m - 1, k = 2^m - m - 1$, ahol m egy egész szám valamint $d_{min} = 3$. Példaként nézzük meg a (7,4)-es paraméterű Hamming kódot. A blokk hossza n = 7, ehhez megnézzük a $x^7 + 1$ polinom felbontását:

$$x^7 + 1 = (1 + x)(1 + x^2 + x^3)(1 + x + x^3) \bmod 2 \tag{3.50}$$

Válasszuk ki a $g(x) = 1 + x + x^3$ generátorpolinómot. A generátormátrix felírható a g(x) és annak cirkuláris eltoltjai segítségével.

$$G = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] \tag{3.51}$$

Gauss eliminációval átalakítható szisztematikussá a generátormátrix:

$$G_{szisztematikus} = \left[\begin{array}{cccc|cccc} 1 & 1 & 0 & | & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & | & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & | & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & | & 0 & 0 & 0 & 1 \end{array}\right] \tag{3.52}$$

A (3.18) egyenletet alkamazva, a kód H paritásellenőrző mátrixa kifejezhető a generátor mátrix segítségével:

$$H = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & | & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & | & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & | & 0 & 1 & 1 & 1 \end{array}\right] \tag{3.53}$$

A kódhoz tartozó szindróma dekódolási táblázat az alábbiak táblázatban látható.

Table 3.6: (7,4) paraméterű Hamming kód szindróma dekódolási táblázata

Szindróma	Hiba helye
0 0 0	0 0 0 0 0 0 0
1 0 0	1 0 0 0 0 0 0
0 1 0	0 1 0 0 0 0 0
0 0 1	0 0 1 0 0 0 0
1 1 0	0 0 0 1 0 0 0
0 1 1	0 0 0 0 1 0 0
1 1 1	0 0 0 0 0 1 0
1 0 1	0 0 0 0 0 0 1

3.1.7 Egyéb kód családok

Reed-Müller kódok és a Hadamard kódok
A Reed-Müller kódok talán az egyik legrégebbi bináris, lineáris kódcsalád. 1972-ben a Mariner 9-es űrszonda Marsról küldött fekete-fehér képeinek továbbításánál használták. Reed Müller kódok előnye, hogy egyszerű a dekódolása és az első rendű Reed-Müller kódok kifejezetten effektívek. Egy r-ed rendű Reed-Müller kód, amely kódszavának hossza $n = 2^m$ mint $RM(r, m)$ definájuk. A következő egyeztetések állnak fent az RM kódok esetén:

- RM$(0, m)$ kódok egyenértékűek az ismétlő kódokkal $(2^m, 1, 2^m)$
- RM$(1, m)$ kódok a Hadamard kódok $(2^m, m + 1)$, amelyek konstrukciójánál a Hadamard mátrix játszik fontos szerepet. Nagyméretű m-ek esetén alacsony adatátviteli sebességük van, de sok hibát képesek javítani.
- RM$(m - 1, m)$ kódok a paritás bitet használó kódok $(2^m, 2^m - 1)$

Maximimális hosszúságú kódok
A maximális hosszúságú kódok a Hamming kódok komplementer kódjai. Vagyis ha $x^n - 1 = g(x)h(x)$ esetén a választott $g(x)$ generátorpolinóm helyett a $h(x)$ paritásellenőrző polinómot választjuk generátorpolinómnak. Az így kapott kód egy $(2^m, m)$ paraméterű kód lesz.

3.2 Konvolúciós kódok

3.2.1 Bevezetés

A blokk kódok mellett gyakran alkalmazunk konvolúciós kódokat, hibajavító képességük miatt. A blokk kódokkal ellentétben, a konvolúciós kódolók "végtelen hosszú" adatfolyamot képesek kódolni. A pillanatnyi kimenet a kódoló bemenetének aktuális értékétől és annak korábbi értékeitől függ. Egy egyszerű konvolúciós kódoló látható a 3.2. ábrán.

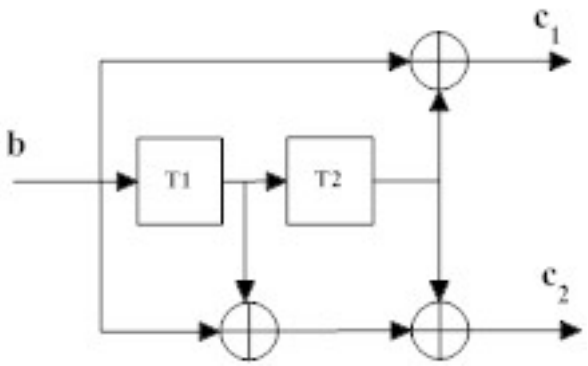

Figure 3.2: Konvolúciós kódoló

Jól látható, hogy ez az egyszerű konvolúciós kódoló, egy bementi bit hatására két kimeneti bitet (kódbitet) generál. A generálás során az összegzők bináris összegzők, vagyis modulo 2 függvényt valósítanak meg. Az is látható, hogy a kódoló két tároló elemet tartalmaz (T1, T2), vagyis az aktuális kimenet a bemenettől és annak két korábbi értékétől függ. A kódoló működését egy táblázatban össze lehet foglalni, amelyben feltüntetjük a bejövő bit értékét, a tárolók értékét és a kimeneti biteket.

Table 3.7: A konvolúciós kódot leíró táblázat

Bemeneti bit	Tároló értéke		Kimeneti bitek	
0	0	0	0	0
0	0	1	1	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	1
1	0	1	0	0
1	1	0	1	0
1	1	1	0	1

A kódoló állapotát a kódolóban lévő tárolók tartalma írja le. A kódoló állapotátmenetét úgynevezett állapot-átmeneti gráffal írjuk le, ezt hívjuk trellisnek (Trellis jelentése rózsakerítés, mivel az állapot-átmeneti gráf hasonlít erre). A fent leírt kódoló

trellis diagrammja alább látható. Az állapotok közötti nyilakon a bemeneti bit és az

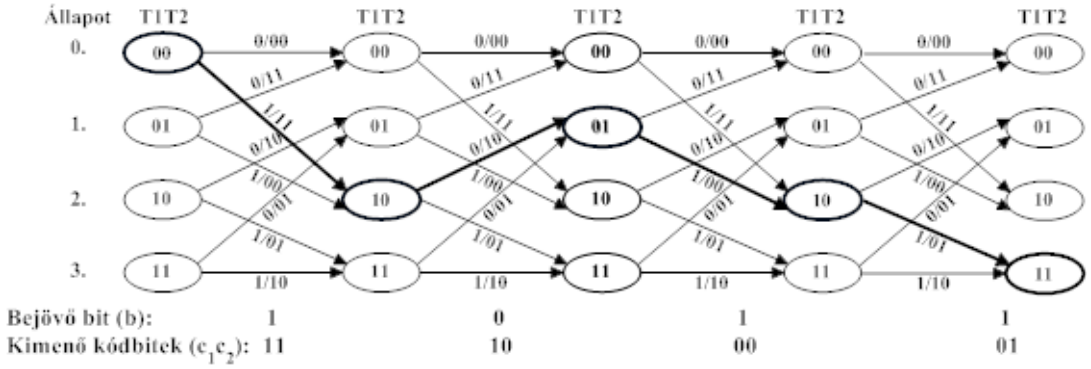

Figure 3.3: A konvolúciós kódoló trellise

ennek hatására generált kimeneti kódbitek vannak feltüntetve. A kódolás kezdetekor a tárolók tartalma 0. A trellisen vastagítva egy lehetséges állapot-átmeneti út látható az 1011 bemeneti bitsorozat hatására.

Egy konvolúciós kódoló kényszerhosszát a tárolók számánál eggyel nagyobb szám adja. Ez azt határozza meg, hogy az aktuális kimenet hány bittől függ. A szabad úthossz (d_{free}) ad választ a kódoló hibajavító képességére, csakúgy, mint a blokk kódok esetén a minimális Hamming-távolság a generált kódszavak között.

Természetesen az ilyen kód is szisztematikussá tehető, ha a kimenő kódbitek között szerepel a bejövő bit is.

3.2.2 Viterbi dekódolás

A konvolúciós dekódoló nem blokkonként dönt, akármilyen hosszú kódszót képes dekódolni. Hamming-távolság alapján, megpróbálja a trellis segítségével a lehető legvalószínűbb (legkisebb) Hamming-távolságú utat megtalálni. A Viterbi algoritmus előnye, hogy egy egyszerű felismerés segítségével nem kell az összes lehetséges utat kiszámolni, hanem leszűkítjük azon utak számát, amelyekhez a vett kódszó távolságát ki kell számolnunk. Lényegében elindulunk az első állapotból, kiszámoljuk a következő állapotba történő átmenet esetén a Hamming-távolságot, majd ha egy állapotba már kétféleképpen juthatunk el, akkor a kisebb Hamming-távolságú utat vesszük figyelembe, a másik utat eldobjuk. Végül az utak közül azt választjuk, amelynek a legkisebb a Hamming-távolsága, és ezen út alapján döntünk a vett vektorból az eredeti üzenetbitekre.

Vegyük a fenti példánkat, ám a küldött kódszó (11-10-00-01) hibázzon a harmadik bitkettősben egy helyen, és legyen a vett kódszó a 11-10-01-01. Az alábbi ábrán látható a különböző utak Hamming-távolsága a vett kódszótól, állapotról állapotra haladva.

A trellisen való haladás lépései a következők. A trellis állapotaiban, az állapotbitek helyett most az addigi út Hamming-távolsága van feltűntetve. Az állapot-átmenetek nyilain pedig a vett kódszó-részlet és az állapot-átmenethez tartozó kódszó közötti

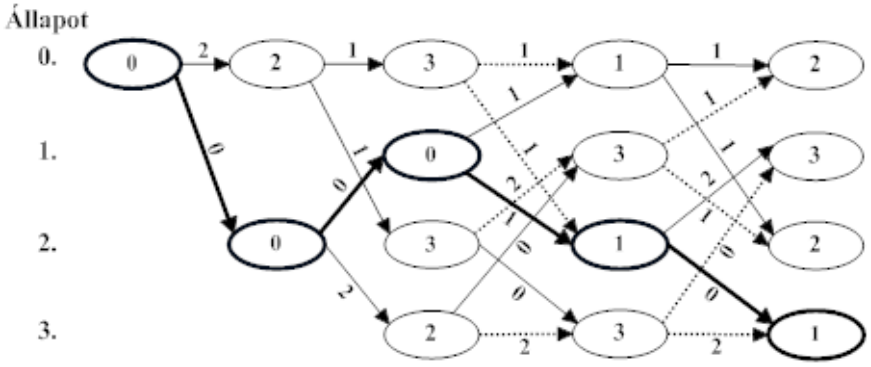

Figure 3.4: Trellis alapján történő dekódolás

Hamming-távolság látható. A dekódolás folyamán a 0-s állapotból indulunk. Vesszük a kódszó első két értékét (11). Ahhoz, hogy a nullás állapotban maradjunk, 2 hibának kellett történnie mivel a kimenő kódszó ebben az esetben 00 lett volna. Ahhoz viszont, hogy a 2-es állapotba menjünk át, nem kellett hibának történnie. Beírjuk mindkét állapotba az addigi hibák összegét, majd nézzük a vett kódszó következő szavát (10), megint megnézzük a hibák számát és beírjuk az adott állapothoz az út addigi hibaösszegét. A következő vett kódrészlet (01) esetén már egy állapotba több módon is eljuthatunk, ilyenkor a kisebb hibaösszegű utat vesszük figyelembe, a másikat eldobjuk. Azt az utat, amit eldobunk, szaggatott vonallal jelöljük. A teljes kódszó feldolgozása után vastag vonallal van jelezve a legkisebb Hamming-távolsággal rendelkező út. Ezt az utat figyelembe véve javítottuk a kódszó hibás bitjét. Természetesen hosszabb kódszavak, hosszabb utak esetén a Viterbi-algoritmus több hibát is képes javítani.

3.2.3 Pontozás

Általános esetekben a konvolúciós kódolók kódaránya: 1/n, vagyis egy bitből a kódoló egy n hosszú kódszót generál. Pontozás segítségével előállíthatók más kódarányok is. A pontozás lényege, hogy a kódszó bizonyos bitjeit szisztematikusan elhagyjuk, majd a dekódolóban az elhagyott bitek helyeit feltöltjük ismeretlen értékekkel ("don't care"), és így hajtjuk végre a dekódolást. Az előzőekben bemutatott esetben, ha a kódszó minden negyedik bitjét elhagyjuk, akkor 2 bemeneti bitre 3 kimeneti bit keletkezik, így a kódarány 2/3.

3.2.4 Lágy és kemény döntés

A bemutatott dekódoló bit szinten kemény döntésekkel dolgozik. Fontos megjegyezni, hogy manapság a számítási kapacitás növekedése miatt, sokkal effektívebb módszer a kemény döntések helyett a lágy döntéseken alapuló módszer használata, amely bitértékek helyett valószínűségekkel dolgozik, figyelembe véve a csatorna zaj-karakterisztikáját. Az

ilyen elven működő dekódolók hátránya a komplexitás, előnyük viszont, hogy sokkal jobb hibaarány érhető el velük, mint kemény döntések használata esetén.

3.3 Turbo kódok

A Turbo kódolás módszerét 1993-ban fedezték fel. Ekkor publikálta Berrou, Glavieux, és Thitimajshima a "Near Shannon Limit Error-correcting Coding and Decoding: Turbo-codes" című cikkét. Ezen kódolási technika sokkal közelebb került a Shannon korláthoz (lásd későbbi fejezet), mint eddig bármely más kódolási módszer. A kódoló maga két kódolóból áll (lehet lineáris blokk kódoló vagy konvolúciós kódoló is), amelyek párhuzamosan vannak kapcsolva. A Turbo kódoló és dekódoló blokkvázlata az alábbi ábrán látható.

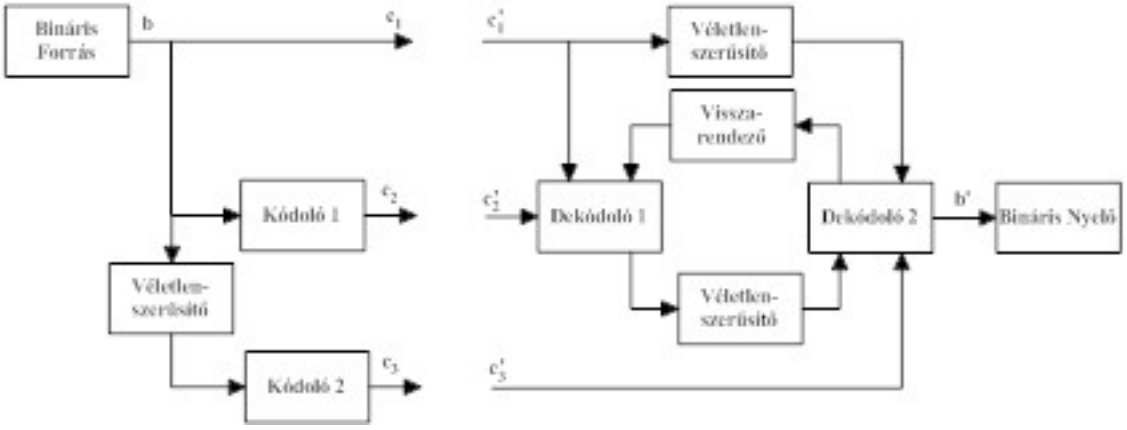

Figure 3.5: Turbo kódoló és dekódoló

Többnyire a két kódoló azonos paraméterekkel rendelkezik, a véletlenszerűsítő fokozat azért szükséges, hogy c_2 és c_3 kimenetek egymástól független, különböző kódsorozatok legyenek. Mindkét dekódoló lágy döntésekkel dolgozik, vagyis valószínűségekkel, amelyekhez elengedhetetlen a csatorna jel/zaj viszonyának ismerete. A két dekódoló megpróbál a vett kódszó egyes elemeire egy becslést adni a kódolatlan c_1, a neki megfelelő kódolt c kódszó, valamint a másik kódolótól kapott valószínűségek alapján. Ez egy iterációs folyamat: a dekódolóban a két dekódoló egymásnak adogatva a b üzenetvektorra tett valószínűségi becsléseiket (úgynevezett dekódolási nyereséget), megpróbál "megegyezni", és iterációról iterációra egyre biztosabb és biztosabb döntést hozni a b üzenetszóra. Ezzel a lágy döntésekre alapuló iterációs folyamattal alacsony hibaarányok érhetőek el. Ez az iterációs folyamat mindig konvergál, vagyis egy adott számú iteráció után már nem érhető el további nyereség.

Az iterációs folyamat könnyebb kezelhetősége érdekében dolgozták ki az EXIT-Chartot (Extrinsic Information Transfer Chart - Külső információcsere-táblázat). Ezen táblázat segítségével a két dekódoló közötti iterációs folyamat egyszerűen nyomonköve-

thető. A táblázat a két dekódoló közötti kölcsönös információcserét követi nyomon iterációról iterációra. A kölcsönös információ értéke 0 és 1 között lehet, ami tulajdonképpen azt mutatja meg, hogy a valós és becsült üzenet "mekkora biztonsággal hasonlít" - az 1 érték az jelenti, hogy teljesen biztos, a 0 érték pedig, hogy teljesen bizonytalan. Minden dekódolónak van egy EXIT-függvénye, amely megmutatja, hogy a kódolási paraméterek, a csatorna jel/zaj viszonya és a másik dekódolótól kapott valószínűség alapján, mekkora kimeneti kölcsönös információt képes produkálni a bemeneti kölcsönös információ függvényében. A mérnökök ilyen táblázat alapján tervezhetik a jel/zaj viszony függvényében a két kódoló felépítését, hogy az iterációs folyamat minél jobban konvergáljon az 1 értékű kölcsönös információ felé. Egy ilyen konvergencia folyamat látható az alábbi ábrán.

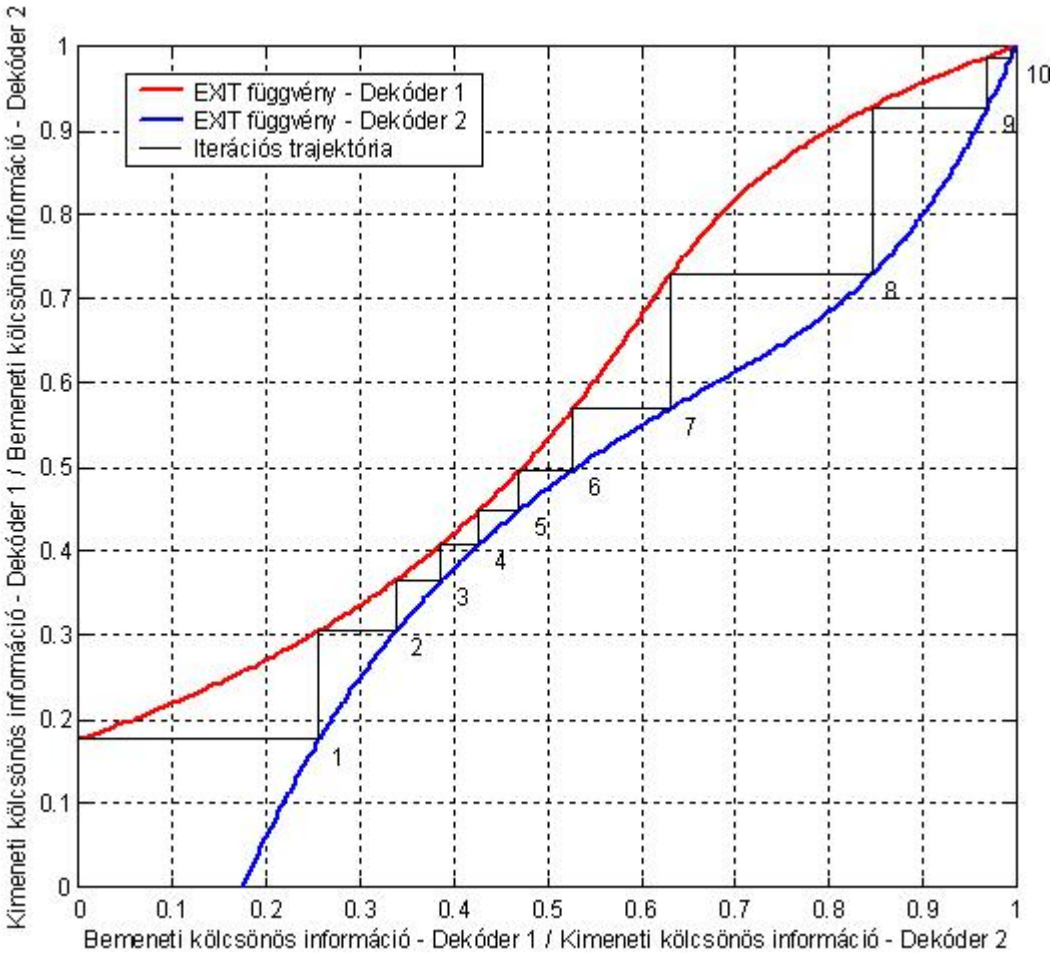

Figure 3.6: Exit táblázat

A második dekódoló EXIT-függvénye felcserélt x-y tengelyekkel van rárajzolva az első dekódoló EXIT-függvényére. A két dekódoló közötti iterációs trajektóriát (iterációs folya-

mat lépései) megvizsgálva jól látható, hogy 10 lépés után konvergál az 1 értékű kölcsönös információhoz. Ez a Turbo kód lassan konvergál, 10 iteráció szükséges a konvergencia eléréséhez. A Turbo elven működik a sorosan kapcsolt kódolók által létrehozott kódolási módszer. Ilyen kódoló rendszer látható az alábbi ábrán. A bináris üzenetet kódolja az

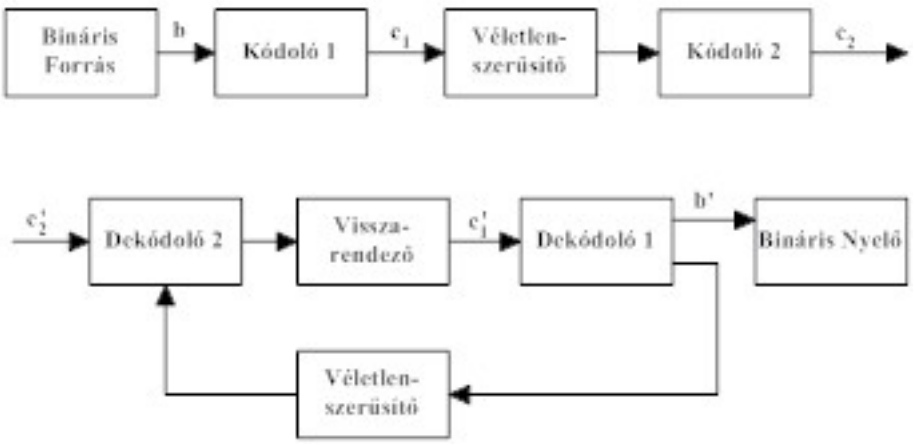

Figure 3.7: Sorosan kapcsolt kódoló és dekódoló

első kódoló, majd a kódolt bitfolyamot véletlenszerűsítik, és ismét kódolják. Jól látható, hogy a dekódolás hasonló, iteratív módon történik, mint a párhuzamos Turbo kódoknál. Természetesen több, 3-4 kódoló is összekapcsolható, de lényeges nyereséget 2 kódoló párhuzamos/soros kapcsolásával lehet elérni. Az utóbbi időben egyre több rendszerben terjedtek el a Turbo kódok. Alkalmazásuk hátránya a komplexitásban rejlik, valamint sok dekódolási algoritmust szabadalom véd. Ezt a kódolási eljárást 3G mobil telefonos szabványban, illetve űrszondák adat-továbbításánál alkalmazzák, valamint a NASA 1997-es Mars expedíciója során is ezt használták képtovábbításra.

3.4 LDPC kódok

Az LDPC (Low-Density Parity-Check - alacsony-sűrűségű paritásellenőrző) kódok komoly konkurensei a Turbo kódoknak a Shannon-határ elérésében. Már 1960-ban felfedezte őket Robert Gallagher, de sokáig feledésbe merültek, főleg nagy számítási komplexitásuk miatt, csak a legutóbbi időben kerültek ismét előtérbe. Az LDPC kódok nagyméretű blokk kódok, amelyek nagy kódtávolsággal (d_{min}) rendelkeznek. A kód G generátormátrixa nagyméretű akár több ezer sorból és oszlopból is állhat. A generátormátrixban jóval kevesebb 1-es mint 0-s található, erre utal az "alacsony sűrűségű" kifejezés. Egy k hosszúságú üzenetszóhoz rendel hozzá paritásbiteket. A paritásbitek előállítása véletlenszerűen történik, a k hosszúságú üzenetszóból l darab bit kiválasztásával. Az LDPC kódokat két paraméter határozza meg: egy bit az üzenetblokkból hány paritásbit meghatározásában vesz részt, illetve egy paritásbit meghatározásához hány bit szükséges az üzenetblokkból. A paritásbitek generálása az alábbi ábrán jól látható. Az ilyen gráfot nevezik Tanner gráfnak is. Ennél a kódnál 4 üzenetbit határoz meg egy par-

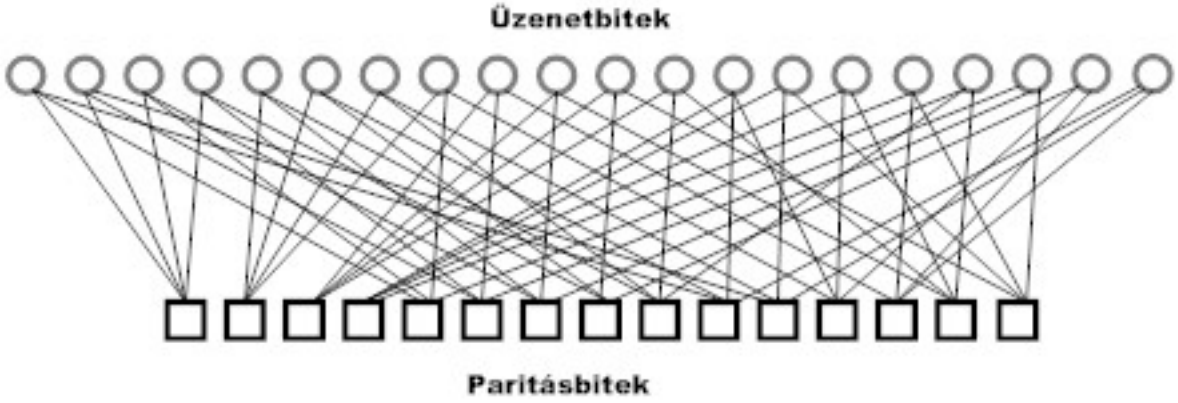

Figure 3.8: Egy LDPC kód összerendelési diagrammja

itásbitet, valamint egy üzenetbit 3 paritásbit kialakításában vesz részt. Egy LDPC kód paraméterei nem határozzák meg a konkrét összerendelési szabályt. Az összerendelési szabály a paraméterek alapján többnyire véletlenszerűen lesz generálva. Szabálytalan LDPC kódoknak nevezzük azokat a kódokat, amelyek esetén a paraméterek nem állandóak, például némely paritásbitet 4 üzenet bit határoz meg, a másikat pedig 3. A szabálytalan LDPC kódok előnye, hogy jobb bithibaarány érhető el mint a szabályos LDPC kódokkal.

Az LDPC kódok dekódolása lineáris komplexitású a kódhossz a kódhossz függvényében, míg a kódolási folyamat négyzetes komplexitású. A dekódolás, hasonlóan a turbo kódokhoz, iteratív módszerekkel történik, lágy döntések alapján. Hátrányuk a már korábban is említett komplex dekódolási algoritmusok. Az LDPC kódok dekódolási módszerét "feltételezés továbbterjedés algoritmus"-nak (belief propagation algorithm) hívják.

Lássunk egy példát az algoritmusra, amely bár kemény döntéssekkel lesz bemutatva,

a lépések lágy döntésekkel is hasonlóak. A továbbiakban tárgyalt LDPC kód összerendezési diagramja látható a 3.9 ábrán.

Legyen a küldött kódszó: $c = [10010101]$. A csatorna hatására a kódszó második elemén

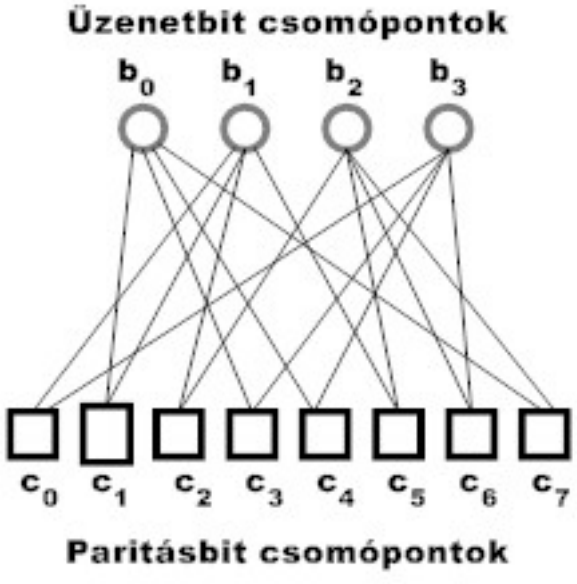

Figure 3.9: A példa LDPC kód összerendezési diagrammja

történjen hibázás, így a vett kódszó: $c' = [11010101]$.
A dekódolási algoritmus lépései a 3.9 ábrán látható gráf alapján a következőek:

1. Az első lépésben a gráfban a paritásbit csomópontok küldenek egy "üzenetet" az üzenetbit csomópontok felé, hogy milyen bit, amit ők tartalmaznak. Jelen esetben c_0 küld egy 1 értékű "üzenetet" b_1 és b_3 felé.

2. A második lépésben minden kódbit csomópont elküldi az összes paritásbit csomópontnak az általa kapott biteket. Az 1. és 2. lépés látható az alábbi táblázatban.

   Az algoritmus leáll, ha minden paritásbit stimmel az üzenetbitek alapján.

3. Minden kódbit csomópont megkapja a hozzá rendelt paritásbit csomópontoktól a paritásbiteket. A bináris döntés "többségi" alapon zajlik. A kapott paritásbitek és az érkezett üzenet alapján döntés születik az adott kódbitre. Jelen esetben az alábbi táblázatban látható, ahogy a hibás c_1 bit javítva lesz, mivel a c_0 és c_1 0 bitet javasolt az adott kódbitre.

4. Az algoritmus folyatása a 2. lépéstől.

Jelen esetben az algoritmus egy iteráció esetén leállna, mivel kijavította a hibás bitet, és az üzenetbitek alapján minden paritásbit megfelel a kódszóban.

Table 3.8: 1. és 2. lépést leíró táblázat

Üzenetbit csomópont	Küldött/Kapott "üzenet"
b_0	Kapott: $c_1 \mapsto 1$ $c_3 \mapsto 1$ $c_4 \mapsto 0$ $c_7 \mapsto 1$ Küldött: $0 \mapsto c_1$ $0 \mapsto c_3$ $1 \mapsto c_4$ $0 \mapsto c_7$
b_1	Kapott: $c_0 \mapsto 1$ $c_1 \mapsto 1$ $c_2 \mapsto 0$ $c_5 \mapsto 1$ Küldött: $0 \mapsto c_0$ $0 \mapsto c_1$ $1 \mapsto c_2$ $0 \mapsto c_5$
b_2	Kapott: $c_2 \mapsto 0$ $c_5 \mapsto 1$ $c_6 \mapsto 0$ $c_7 \mapsto 1$ Küldött: $0 \mapsto c_2$ $1 \mapsto c_5$ $0 \mapsto c_6$ $1 \mapsto c_7$
b_3	Kapott: $c_0 \mapsto 1$ $c_3 \mapsto 1$ $c_4 \mapsto 0$ $c_6 \mapsto 0$ Küldött: $1 \mapsto c_0$ $1 \mapsto c_3$ $0 \mapsto c_4$ $0 \mapsto c_6$

Table 3.9: Döntési táblázat

Kódbit	Érkezett üzenetérték	"üzenet" a paritásbitektől	döntés
c_0	1	$b_1 \mapsto 0$ $b_3 \mapsto 1$	1
c_1	1	$b_0 \mapsto 0$ $b_1 \mapsto 0$	0
c_2	0	$b_1 \mapsto 1$ $b_2 \mapsto 0$	0
c_3	1	$b_0 \mapsto 1$ $b_3 \mapsto 0$	1
c_4	0	$b_0 \mapsto 0$ $b_3 \mapsto 1$	0
c_5	0	$b_1 \mapsto 0$ $b_2 \mapsto 1$	1
c_6	0	$b_2 \mapsto 0$ $b_3 \mapsto 0$	0
c_7	1	$b_0 \mapsto 1$ $b_2 \mapsto 1$	1

Tényleges rendszerekben a dekódolási algoritmus valószínűségek (lágy döntések) alapján történik. Az algoritmus pedig speciális, a valószínűségek tulajdonságának megfelelő műveleteket használ. Így a bithibaarány iterációról iterációra javul. Ilyen kódokat használnak például a DVB-S2 műholdas rendszerekben.

3.5 Shannon-korlát

A Shannon-korlát egy elméleti alsó korlátot határoz meg Gaussi fehér zajjal terhelt csatornák esetén a hibajavításra. Adott zajszint és adatsebesség mellett elérhető minimális bithibaarányra ad korlátot. Arra vonatkozóan azonban, hogy hogyan kell ilyen kódokat generálni, amelyek segítségével ez az alsó korlát elérhető, nem ad választ.