Információelmélet

Kivonatos jegyzet

Veszprémi Egyetem, Műszaki Informatika Szak

Készítette:

Dr. Vassányi István, © 2002-2005.
vassanyi@irt.vein.hu
(Kérem, hogy a jegyzetben talált bármilyen hibát ezen a címen jelezzék!)

Letölthető a http://www.irt.vein.hu/~vassanyi/info/infojegyzet.pdf címről,
A Veszprémi Egyetem hálózatáról

Tartalom

Kérded, hol forog a nagy kerék?
A mérnök veszi a komputerét
Ő olyat sohase mond,
Hogy rajtad áll, mert te vagy benne a középpont…
(Omega)

I. FORRÁSKÓDOLÁS

1. Entrópia és kölcsönös információ

1.1. Az információ filozófiai, köznapi és mérnöki fogalma

Az információ filozófiai fogalma, az ismeretelmélet alapjai[1]

- Platón: **Phaidón** (i.e. V. század). „*Ha valaha is tisztán akarunk tudni valamit, el kell a testtől szakadnunk, és csupán a lélekkel kell szemlélnünk a dolgokat önmagukban*" A tiszta és közvetlen, értelmi megismerés elmélete, mely valószínűleg nagy hatással lehetett Arisztotelészre. A lélek eszerint, amint el tudja különíteni magát a testtől, szellemi tekintetével közvetlenül meglátja a dolgok lényegét, ami a fajalkotó, belső **forma**, az idea, eidosz (forma, alak).
- Arisztotelész: **A lélekről**. Arisztotelésznek, Platón tanítványának e műve két évezredre meghatározta az antik, keresztény, középkori zsidó és arab lélekfilozófiát. Radikális ismeretelméleti tézise szerint „*a lélek bizonyos módon azonos valamennyi létezővel*", mégpedig úgy, hogy az értelmes lélekrész alsóbbik része, az ún. passzív értelem, a megismerés során **azonosul** a megismert dolog „***belső formájával***", szubsztanciájával. Ez az azonosulás a lélek (mely maga is forma, a test belső formája) belső megformálódása, az **in-formáció**. Ezt az információ-fogalmat a filozófia ma is használja.
- Eriugena: **A természet fölosztása**/I. (IX. századi neoplatonikus keresztény misztikus). A „**forma**" keresztény teológiai-antropológiai értelemben az emberi faj belső formája, ami Eriugena szerint nem más, mint Krisztus. Az ember feladata ezen isteni, belső formához **konformálódni**.
- Ficino: **Öt kérdés a lélekről** (XV. század vége). Marsilio Ficino a XV. század végi Firenze korszakalkotó filozófusa, az olasz reneszánsz legnagyobb gondolkodója. Főműve a *Theologia platonica de immortalitate animorum* (Platonikus teológia a lélek halhatatlanságáról). Ficino változatlanul az Arisztotelész megszabta ismeretelméleti pályán mozogva e rövid traktátusában többek között arról beszél, hogy a lélek miként látja Istent értelmi látással. Az arisztoteliánus lélekfilozófiával kombinált újplatonikus miszticizmus hagyományainak megfelelően ez a látás a lélek Istenbe való **transzformáció**jával jár: Isten mint **forma** hozzákapcsolja magát a lélekhez („tamquam forma … animae sese iungit").
- John Locke: **Értekezés az emberi értelemről** (1690). Metafizikai álláspontja Descartes nyomán a dualizmus. Locke már tagadja az arisztoteliánus lélekfilozófiát és a platonikus ideatant. Locke szerint a dolgoknak ugyan nincsen belső formája, de az **information** szó e szövegben mégis előfordul

[1] Az idézett források eredetiben és magyar fordításban megtalálhatók a tárgy előadójánál.

egyszer-egyszer, amikor még nem a modern értelemben vett, manipulálható adatot jelenti, hanem valakinek a belső, erkölcsi megformálását, nevelését.

Hír, adat, információ, tudás. Az információ köznapi fogalma.
Az adat és a hír az információ hordozója, a hordozott információ mennyiségére jellemző az, hogy „mennyivel növeli a tudásunkat". Azt is szokták mondani, hogy az információ „értelmezett adat". Az információ intuitív megközelítéséhez érdemes szemügyre venni az információval kereskedők (pl. napilapok) módszereit. Ezekből látható, hogy a hír értéke két, egymással összefüggő forrásból származik:

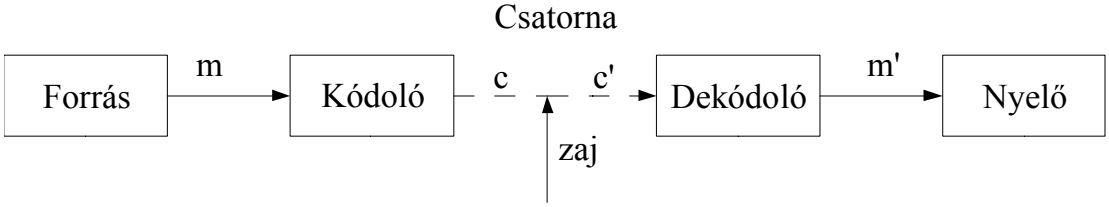
- Váratlanság (mennyire meglepő a hír)
- Relevancia (mennyire vonatkozik a befogadóra).

Egy példa a mindenkit érintő hírre: „Holnap reggel nem kel fel a Nap."
Egy kevésbé releváns hír: „Hosszú Péter *végre* átment a szigorlaton." (azért a család belső újságjában, ha lenne, biztos a címoldalra kerülne!)
A váratlanság megemeli az információ értékét: „Holnap csótányirtás lesz a menzán." —legfeljebb az egyetemi újságig jut el, de „Holnap denevérirtás lesz a menzán." — akár országos hír is lehet.

A köznapi értelemben vett információ két fenti aspektusából az információ **mérnöki fogalma** és az *információ elmélete* csak a váratlanságra épül. Az elmélet alapjait a 2002. februárjában elhunyt C.E. Shannon rakta le 1948-ban.

1.2. Diszkrét források jellemzése, az entrópia

Célunk alapvetően az információ megragadása és továbbítása térben (pl. műsorszórás) és/vagy időben (pl. könyv, CD), a forrástól a befogadóig (nyelő). A továbbítás közege sokféle lehet, pl. rádióhullám, drót, füstjelek, rovásbotok, stb. A közeget csatornának fogjuk hívni. A valóságos csatornán sajnos mindig számolnunk kell a **zaj** jelenlétével.



1.1. ábra Az információtovábbítás általános modellje

Az 1.1. ábrán m a továbbítandó üzenet, c a csatornára kerülő, kódolt üzenet, c' a vett üzenet, és m' az eredeti üzenet becslője.

Az információtovábbítás két legfontosabb célkitűzése:

- egyrészt cél, hogy az üzenet minél kisebb torzítással érkezzen meg a nyelőhöz (a torzítás a csatorna zajának a következménye),
- másrészt, hogy a csatornát (melynek használatáért általában fizetni kell) a lehető legjobban kihasználjuk.

A jó minőségű információtovábbítás érdekében a csatorna elé és mögé **kódoló** ill. **dekódoló** egységet iktatunk be. Ezeket a fenti két szempont alapján tervezzük meg.

Az üzenetet úgy fogjuk fel, mint a forrás által kibocsátott szimbólumok sorozatát. A továbbítási folyamat leírásához meg kell különböztetnünk **diszkrét** (pl. betűk) és **folytonos** (pl. énekhangok) szimbólumkészleteket. A szimbólumok ütemezése is lehet **diszkrét** (pl. írott szöveg) vagy **folyamatos** (pl. beszéd). A diszkrét

ütemezésű forrás azonos időszeletenként mindig kibocsát egyet a szimbólumai közül. A szimbólumok készlete és ütemezése különféle mintavételezési, kvantálási technikákkal mindig diszkrétté alakítható. (Erre egy példa egy beszélgetés lejegyzése egy könyvben.) Ezért az információ elmélete, és a továbbiakban ez a jegyzet is, elsősorban a diszkrét idejű és szimbólumkészletű forrásokkal foglalkozik.

A forrást az általa generált szimbólumok készletével (**forrásABC**) és a szimbólumokhoz tartozó elemi események valószínűségeivel (**forráseloszlás**) jellemezhetjük:

$$A = \{a_0, a_1, \ldots, a_{M-1}\},$$
$$P\{A\} = \{p_0, p_1, \ldots, p_{M-1}\}, \quad \sum_i p_i = 1, \quad p_i \neq 0$$

ahol M a szimbólumkészlet számossága (|A|-kel is jelölik), az elemi események pedig teljes eseményrendszert alkotnak (valami mindig történik), ezért a valószínűségeik 1-re egészítik ki egymást. Az általunk tárgyalt forrásoktól ezen kívül általában elvárjuk, hogy a forrás legyen:

- **stacionárius**, azaz a p_i valószínűségek ne függjenek az időtől
- **emlékezet nélküli**, azaz a szimbólumokhoz tartozó elemi események legyenek függetlenek egymástól.

Ezek után definiáljuk az i-edik esemény bekövetkezése által hordozott **információ** fogalmát, mint a váratlanság mértékét:

$$I(a_i) = \log \frac{1}{p_i} \text{ [bit]}$$

Az információ mértékegysége a **bit** (binary unit)[2], mivel 2-es alapú logaritmust használunk. Látható, hogy független események információtartalma összeadódik.

A forrás által egy időszeletben kibocsátott **átlagos információ** mennyiségét **forrásentrópiának** nevezzük:

$$H(A) = \sum_i p_i \log \frac{1}{p_i} \text{ [bit]}$$

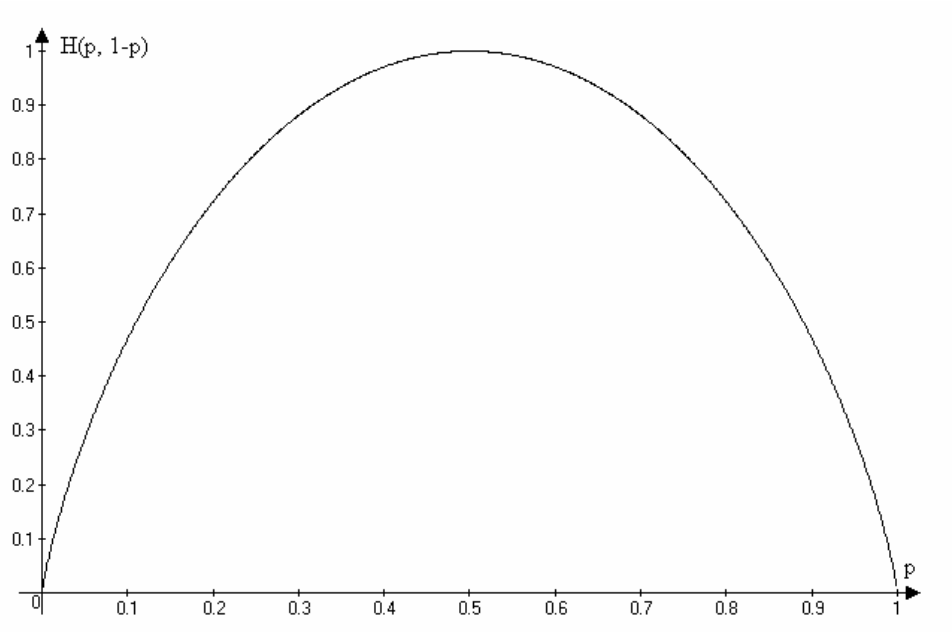
Például a P(A)={0.5, 0.5} forrás entrópiája 1 bit. Ha az elemi valószínűségek között a 0 illetve 1 szélső értékeket is megengedjük, úgy a P(A)={0, 1} forrás entrópiája 0 bit lesz[3] (nagyon unalmas forrás, mivel egy lehetetlen és egy biztos eseményből áll). Megmutatható az is, hogy az egyenletes eloszlás **maximalizálja** az entrópiát, és a maximum értéke log$M$ (lásd az F1 függelékben). Ezek szerint tehát tetszőleges, M szimbólummal rendelkező forrásra $0 \leq H(A) \leq \log M$.

Egy példa: a P{A}={0.5, 0.3, 0.15, 0.05} forrásra H(A)=1.65 bit. A továbbiakban P{A}={…} helyett röviden A={…} jelölést is fogunk használni, ha A szimbólumai a probléma szempontjából nem lényegesek.

---

[2] Nem keverendő össze a kettes számrendszerbeli számok egy helyiértékével (digitjével), melyet ugyanígy hívunk. A kettő között az a kapcsolat, hogy egy kettes számrendszerbeli digit által hordozott átlagos információ mennyisége *maximum* 1 bit lehet.

[3] mivel $\lim_{x \to 0} x \log \frac{1}{x} = 0$



1.2. ábra A bináris forrás entrópiája és eloszlása közti összefüggés

1.3. A feltételes entrópia és a kölcsönös információ

Ezeket a fogalmakat két vagy több forrás **együttes, átlagos** információtartalmának a jellemzésére használjuk. Szerepük lesz a kódolási eljárások értékelésénél és a kriptográfiai résznél.

Legyen két forrásunk, A és B, például két, egymás mellé tett TV-készülék által mutatott jelenet vagy ábra. Jelölje $p_{i,j}$ annak a valószínűségét, hogy egy adott időszeletben az A forrás az *i*-edik, a B pedig a *j*-edik szimbólumot bocsátja ki. Ekkor definiáljuk A és B **kölcsönös (együttes) entrópiáját** ill. **feltételes entrópiáját** a következő módon:

$$H(A,B) = \sum_{i,j} p_{i,j} \log \frac{1}{p_{i,j}}$$

$$H(B \mid A) = \sum_{i} p_i \sum_{j} p_{j|i} \log \frac{1}{p_{j|i}}$$

Itt $p_{j|i}$ lehet 0 is, ezzel kapcsolatban lásd a 3. lábjegyzetet. A H(A,B) kölcsönös entrópia a két forrás által együttesen szolgáltatott, időszeletenkénti átlagos információ mennyisége. Intuitíven érezhető, hogy ha a két forrás független egymástól, akkor ez nem lehet más, mint a két forrás külön-külön vett entrópiájának az összege.

Ha a H(A,B) fenti definícióját kifejtjük és ismételten alkalmazzuk a

$$p_{i,j} = p_i p_{j|i}$$

azonosságot, akkor a következő összefüggéshez jutunk:

$$\mathrm{H}(A, B) = \mathrm{H}(A) + \mathrm{H}(B|A),$$

ahol H($B|A$) a fenti képlettel adott, a B forrásnak az A forrásra vonatkozó feltételes entrópiája[4]. Ez megadja, hogy az *A forrás ismeretében, ahhoz képest* átlagosan mennyi

[4] A $H(A,B)= H(A)+H(B|A)$ képletet könnyű megjegyezni, hasonlít a feltételes és együttes valószínűségek összefüggésére, de az információ logaritmikus definíciója miatt a szorzás helyett itt összeadás áll.

meglepetéssel (azaz mennyi információval) szolgál a B forrás. A képletből látható, hogy a H($B|A$) feltételes entrópia nem más, mint a B forrásnak az A forrás egyes elemi eseményeihez tartozó entrópiáiból képzett súlyozott átlaga. Ugyanígy természetesen az is igaz, hogy H(A,B) = H(B) + H($A|B$).

Ha a B forrás **teljesen független** A-tól (pl. az egyik TV-n folyamatosan focimeccsek, a másikon kalandfilmek mennek), akkor azt várjuk, hogy a H(A,B) együttes entrópia a két forrás külön-külön vett entrópiájának összegével egyenlő (a tudásunk mindkét forrásból egymástól függetlenül gyarapodik). Valóban, az elemi események függetlensége miatt a feltételes valószínűségek feltétel nélkülivé alakulnak, miáltal H($B|A$) képlete átalakul H(B)-vé. Ekkor tehát H(A,B)= H(A)+H(B).

Másrészt, ha a B forrás az A által **teljesen meghatározott**, akkor nem várunk új információt a B-ből (ha mindkét informátorunk ugyanazt mondja, akkor mindent tudunk akkor is, ha csak az elsőt hallgatjuk meg). Azaz: H(A,B)= H(A). Valóban, a B forrás meghatározottsága miatt az összes elemi esemény $p_{j|i}$ feltételes valószínűsége 0 vagy 1 lesz, és H($B|A$)=0 adódik. Összegzésképpen megállapíthatjuk, hogy

$$0 \le \mathrm{H}(B|A) \le \mathrm{H}(B),$$

azaz a „mellékinformáció nem növeli az entrópiát." Minderre egy példát szolgáltat a paritásbit, mint egyértelműen meghatározott, de kisebb entrópiájú forrás (1.1. példa).

Az A és B közti **kölcsönös információ, I(B,A)** fogalmát ezek után úgy definiáljuk, mint „a B forrás átlagos információtartalmából az a rész, amely az A által meghatározott". Ezt úgy állíthatjuk elő, hogy H(B)-ből levonjuk azt a részt, ami *nem* az A által meghatározott, vagyis a B-ben az A-hoz képest kapott átlagos meglepetést, azaz információt. Ez pedig nem más, mint a H($B|A$):

$$I(B,A) = H(B) - H(B \mid A)$$

Könnyen megmutatható, hogy H(A, B) kétféle felírása miatt

$$I(B,A) = I(A,B)\,.$$

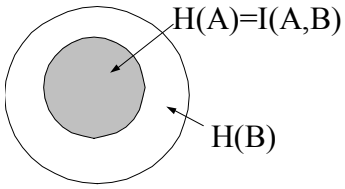
Ha a két forrás **teljesen független egymástól**[5], akkor a H($B|A$) feltételes entrópia H(B)-vé alakul, azaz a kölcsönös információ 0. Ez megfelel elvárásainknak. Ha viszont a B az A által **teljesen meghatározott**, akkor H($B|A$)=0, azaz I(B,A)=H(B): a teljes H(B) az A-ból származik. Ez azonban nem azt jelenti, hogy szükségképpen H(A)=H(B). Ha pl. |A|>|B|, azaz a A több eseménye (szimbóluma) is tartozik a B egy szimbólumához, akkor H(A)>H(B), azaz B kisebb átlagos információtartalmú. Tehát, ha az A forrás egy kommunikációs folyamat elején történt megfigyelésből származik, a B forrás pedig a folyamat végéről, akkor a folyamat során információ veszett el.

Ugyanez a helyzet minden olyan esetben is, amikor, bár H(B) nem kisebb, mint H(A), de I(B,A)<H(A), mivel az eredetileg A-ban meglévő információ egy része „nem jött át". Ennek egy klasszikus esete a *falusi pletyka*. Az általunk hallott történet általában érdekesebb (meglepőbb, azaz, mint forrás, nagyobb entrópiájú), mint az eredeti, igaz verzió, de az igazság (az eredeti információtartalom) egyes lényeges részei mégis elvesztek… Honnan származik hát a pletykában lévő plusz információ? A csatornából!

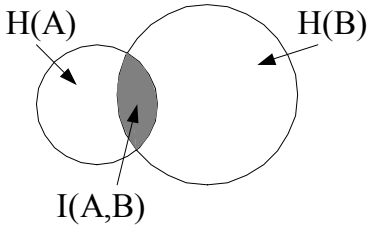
[5] A két forrás nyilvánvaló függetlensége a feladat feltételeiből derülhet ki. Pl. az isztambuli főimám esti imádsága (A forrás) és egy vezérlési hibás veszprémi közlekedési lámpa (B forrás) függetlenek.

1.1. példa: Az információtovábbítás néhány alapesete. Feltesszük, hogy az információ A-tól B felé halad. A kölcsönös információ árnyékolva van. Az 1. és a 3. példa diszkrét, a 2., 4. és 5. folytonos értékkészletű.

1. Parkinson-kóros távírász a Titanicon, akinek időnként beremeg a keze (információ nem veszett el, de új információ keletkezett, ami nem az üzenetre, hanem a távírászra jellemző).
"A" forrás: a kapitány által megfogalmazott üzenet morzejelekkel
"B" forrás: a szomszéd hajón vett jelsorozat

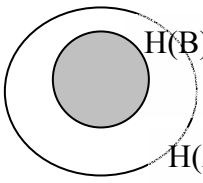


2. falusi pletyka (információ veszett el, és új információ keletkezett)
"A" forrás: az eredeti hír
"B" forrás: a hír a falu másik végén



, H(A) < H(B)

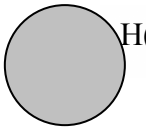
3. 2+1 *paritásbit (teljesen meghatározott forrás, információ veszett el)*
"A" forrás: A={a0, a1, a2, a3}={0.25, 0.25, 0.25, 0.25}
"B" forrás: B={b0, b1}. *B –ben* b0 *következik be, ha A-ban* a0 *vagy* a2 *következett be, különben* b1.



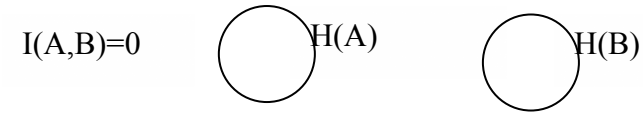
H(B)=I(A,B)=1 bit
(ellenőrizzük számítással!)

H(A)=2 bit

4. tökéletes fordítás (teljesen meghatározott forrás, információ nem veszett el)
"A" forrás: a könyv magyarul
"B" forrás: a könyv angolul



5. független források:
"A" forrás: az isztambuli imám esti imádsága
"B" forrás: egy vezérlési hibás veszprémi közlekedési lámpa
(független források—ha ugyanazon információs folyamatról származnak, akkor a folyamat során minden információ elveszett)



Végül még egy számpélda egy osztálykirándulásról.

1.2. példa: Egy vízitúrázó társaság 75%-a lányokból áll (a többiek fiúk). A lányok közül 10% szeretne focizni, a többiek inkább röpizni. A fiúknál 50-50% ez az arány.
Kérdés, hogy mit mond (mennyi információt hordoz) a sportolási készség az illető neméről?

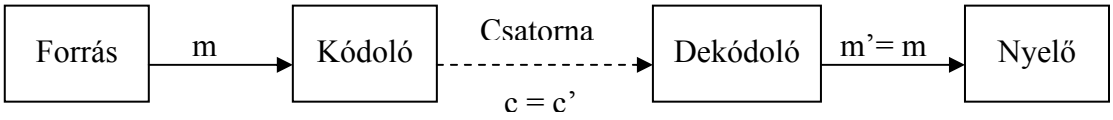
A helyes megoldás a kölcsönös információ definíciója alapján
I(nem, sport) = 0.12 bit.

2. Források kódolása

2.1. Alapok

Az 1.1. ábra szerinti kódoló ill. dekódoló egységek tervezését az elméletben (és gyakorlatban is), két lépésben oldjuk meg, az ott ismertetett két minőségi paraméter teljesítése céljából. A **forráskódolás** célja a csatorna kihasználtságának a javítása, a forrás tartalmának az információvesztés nélküli **tömörítése**. A zajos csatornán történő, minél kisebb torzítású információátvitellel a **csatornakódolás** foglalkozik. A forráskódolási algoritmusok tárgyalása során ezért feltesszük, hogy a csatornán **nincs zaj**. Ennek megfelelően az 1.1. ábra a 2.1. ábra szerint módosul.

A gyakorlati esetek túlnyomó részében a csatorna **bináris**, tehát „0" és „1" szimbólumok tárolására/továbbítására van lehetőség. Ezért a továbbiakban a bináris csatornára készülő kódolási eljárásokkal foglalkozunk. Feltesszük, hogy a csatornán egyszerre egy bináris szimbólumot továbbíthatunk, a továbbítás **költsége** pedig arányos a továbbított szimbólumok, azaz a csatornahasználatok számával. A kódolás során minden forrásszimbólumhoz hozzárendelünk egy „0" és „1" szimbólumokból álló füzért, azaz **kódszót**. Magát a hozzárendelési szabályt röviden **kódnak** fogjuk hívni.



2.1. ábra A forráskódolás általános modellje

A forráskódolás **alapgondolata**, hogy a kevésbé valószínű forrásszimbólumokhoz hosszabb, a valószínűbbekhez rövidebb szimbólumsorozatot rendelünk. Ezzel—huzamos idejű csatornahasználat alatt—csökkenteni tudjuk a használat költségét. A különböző hosszúságú kódszavakat használó kódot **változó hosszúságú kódnak** nevezzük. A változó hosszúságú kódok esetén definiáljuk az L(A) **átlagos kódszóhosszt**:

$$L(A) = \sum_i p_i l_i$$

ahol l_i a i-edik szimbólumhoz tartozó kódszó hossza, p_i pedig a szimbólum valószínűsége. Egy kódszó elküldése a csatornán átlagosan L(A) költséget jelent, ezért a célunk az L(A) csökkentése lesz.

Megfejthetőnek nevezünk egy kódot, ha az előállított kódszó-sorozat minden esetben egyértelműen meghatároz egy forrásszimbólum-sorozatot. Ekkor nem lehetnek kétségeink a dekódolás során. A megfejthetőségnek egy elégséges[6] feltétele az, ha a kód **prefix**, azaz

- Minden forrásszimbólumhoz más és más kódszó tartozik, és
- Egyik kódszó sem folytatása egy másiknak.

2.1. példa 3 elemű forrás kódolására

Szimbólum	*Egy prefix kód kódszavai*	*Egy nem prefix kód szavai*
„a”	„0”	„1”
„b”	„10”	„10”
„c”	„11”	„11”

A nem prefix kód használata mellett bajban vagyunk a vevő oldalon, ha például az „111111” *csatornaszimbólum-sorozatot kell dekódolni: a kód nem megfejthető.*

A feltétel elégséges, mert a vevőoldalon az érvényes kódszavak ismeretében egyértelműen meg tudjuk határozni a kódszavak határait, és mivel minden kódszó különböző, ezért az eredeti szimbólum-sorozatot is.

Nem megfejthető az olyan kód, amelyben a vevőoldalon nem tudjuk biztosan különválasztani az egyes kódszavakat. Ezt esetenként ki is használták. Egy távolról idevágó példa a magyar történelemből (Bánk bán) a királynő-ellenes összeesküvés kapcsán szándékosan központozás nélkül íródott következő üzenet:

„A KIRÁLYNŐT MEGÖLNI NEM KELL FÉLNETEK JÓ LESZ HA MINDNYÁJAN BELEEGYEZTEK ÉN NEM ELLENZEM”

Ezt kétféleképpen is ki lehet olvasni, pont az ellentétes értelemmel: az ellenző nyilatkozatnál 4, a támogatónál 3 „kódszóra” lehet felbontani a szöveget. Az eset történelmi hátteréről bővebben lásd az F2 Függelékben.

[6] A feltétel nem szükséges, pl. megfejthető az ún. *postfix* kód is.

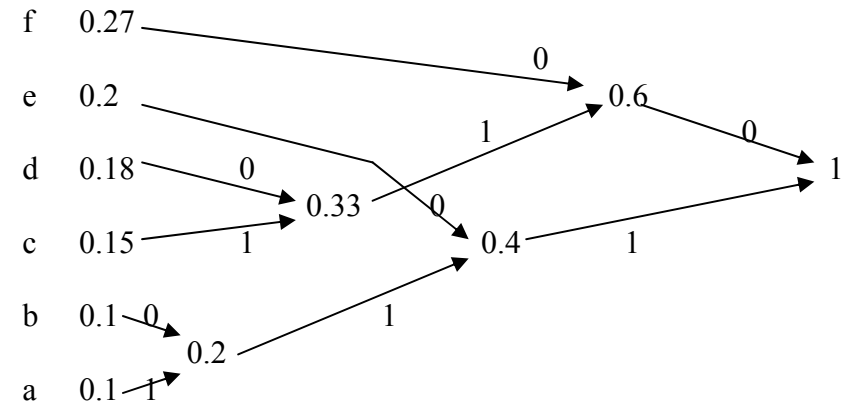
2.2. A Huffman-kód

A Huffman-kód prefix, változó hosszúságú kód, 1952-ből származik. A kód képzéséhez ismernünk kell (pl. statisztikai vizsgálatokból) a forrásszimbólumok valószínűségeit. A kódszavakat a következő módon állítjuk elő:

- A szimbólumokat növekvő sorba rendezzük a valószínűségük szerint,
- A két legkisebb valószínűségűt összevonjuk egy „kombinált" szimbólumba, melynek a valószínűsége a két alkotó szimbólum valószínűségének az összege lesz.

Mindezt addig folytatjuk, amíg már csak 1 darab, 1 valószínűségű szimbólumunk lesz. Ekkor az összevonások által képzett bináris fán (melynek levelei az elemi szimbólumok) a gyökértől visszafelé haladva meghatározzuk a kódszavakat olyan módon, hogy ha a fán balra fordulunk, egy „1"-t, ha jobbra, egy „0"-t illesztünk a levélen található szimbólum kódszavának a végéhez.

2.2. példa: az öt elemű A={a,b,c,d,e, f}={0.1, 0.1, 0.15, 0.18, 0.2, 0.27}
forrás kódszavai:

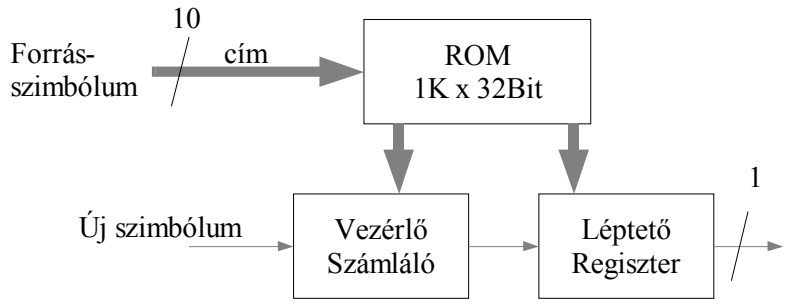


Ha helyesen végeztük el a kódolást, a következőt kapjuk:
a→„111", b→„110", c→„011", d→„010", e→„10", f→„00"

Látható, hogy a „bináris fa" képzési szabály miatt a kód prefix, azaz megfejthető.

A Huffman-kód **hardver** megvalósításában (pl. telefax) a kódolóban egy ROM-ban tároljuk a kódszavakat és az egyes kódszavak hosszát jelentő bináris számot. (Például az első 10 biten a kódszó hosszát, utána 22 biten magát a kódszót.) A csatornára a kódszavak egy léptetőregiszterből kerülnek ki, melyet egy vezérlő vezérel az éppen adott kódszó hossza alapján. A 2.2. ábra egy lehetséges megvalósítás vázlatát mutatja.

A dekódoló a prefix tulajdonságot használja ki: ha a vett kódszóból eddig érkezett darab pontosan egy érvényes kódszóra illeszkedik, akkor elérkeztünk a kódszó végéhez. Mindez megoldható egy asszociatív memóriával (CAM).



2.2. ábra Huffman-kódoló célhardver vázlata

A Huffman-kód egy konkrét alkalmazási területe a hagyományos (Group 3) fekete-fehér telefax-szabvány.

2.3. Forráskódolási tételek

A „**Shannon 1. tétele**”, vagy a „**forráskódolási tétel**” néven ismert tétel megadja a forráskódolás elméleti lehetőségeit és egyben korlátait is.

Legyen A egy diszkrét, emlékezet nélküli, stacionárius forrás, és rendeljünk az egyes a_i szimbólumaihoz egy-egy l_i hosszúságú bináris kódszót!

Ekkor *található* olyan kódolási szabály, amelyre

$$L(A) < H(A)+1,$$

de *nem található* olyan, amelyre

$$L(A) < H(A)$$

Összefoglalva:

$$H(A) \le L(A) < H(A) + 1$$

elérhető, ahol a bal oldali reláció a „negatív”, a jobb oldali a „pozitív” állítása a tételnek. Látható, hogy a forrásentrópia a forrás tömöríthetőségének az alapvető alsó korlátja, ugyanakkor 1 biten belül mindig megközelíthető alkalmas kódolással. Az előbbiekben ismertetett Huffman-kód is rendelkezik ezzel a tulajdonsággal. Ennek az állításnak, és a tételnek is az igazolása megtalálható az irodalomban.

A tétel alapján definiáljuk egy kód *h* **hatékonyságát**.

$$h = H(A) / L(A) \ [\%]$$

A hatékonyság 0 és 100% között változik, a 100%-ot ideális esetben el is érheti. Belátható, hogy ez az eset áll elő például Huffman-kódolásnál, ha az összes forrásvalószínűség kettő negatív hatványa. A 100%-os hatékonyságú kódot **optimálisnak** nevezzük.

Ha a forrás kevés szimbólumot tartalmaz, vagy a forráseloszlás távol van az egyenletestől, akkor még a H(A)+1 átlagos kódszóhossz is nagyon rossz hatékonyságú lehet, például A={0.001, 0.999} esetén h = 0.011/1 = 1.1%. Ekkor N, időben egymás után következő szimbólumot összeolvasztva, egy szimbólumnak tekintve N-szeresére kiterjeszthetjük a forrást. A **kiterjesztett forrás** szimbólumainak száma $|A_{kit}| = |A|^N$ lesz, az egyes szimbólumokhoz tartozó valószínűségek pedig az őket alkotó elemi események valószínűségeinek a szorzatai. Pl. $P(a_1, a_3, a_1) = p_1^2 \cdot p_3$ stb., ha a forrás emlékezet nélküli. Megmutatható, és a szemléletből is adódik, hogy az N-szeresére

kiterjesztett emlékezet nélküli forrás entrópiája a forrásentrópia *N*-szerese[7]. Ha most a kiterjesztett forrásra alkalmazzuk a forráskódolási tételt, és osztunk az *N* blokkmérettel, akkor látható, hogy a forráskiterjesztés révén a forráskódolási tétel által adott elvi korlát az *N* növelésével tetszőlegesen megközelíthető:

$$H(A_{kiterj}) \le L(A_{kiterj}) < H(A_{kiterj}) + 1$$
$$N \cdot H(A_{ered}) \le L(A_{kiterj}) < N \cdot H(A_{ered}) + 1$$
$$H(A_{ered}) \le L(A_{ered}) < H(A_{ered}) + \frac{1}{N}$$

ahol $H(A_{ered})$ az eredeti, nem kiterjesztett forrás entrópiája, $L(A_{ered})$ pedig az eredeti forrás egy szimbólumára jutó átlagos kódszó-hossz. Ez az eredmény a **forráskiterjesztési tétel**.

2.3. példa:

Legyen $A=\{ a_0, a_1, a_2, a_3\}=\{0.5, 0.3, 0.15, 0.05\}$*, ekkor* $H(A)=1.64$ *bit*

Naiv kódolással, négy egyenlő hosszúságú kódszóval:

00, 01, 10, 11, $L(A)=2$, **h=85 %**

Huffman-kód: 0, 10, 110, 111*,* $L(A)=1.70$*,* **h=96 %**

Huffman-kód kétszeresen kiterjesztve:

pl. $P(a_0\, a_0)=0.25$, $P(a_0\, a_1)=0.15$*, stb.*

$L(A\times A)=3.32$, $H(A\times A)=3.29$ *bit, ezzel* **h=99 %**

A forráskiterjesztésnek, mint módszernek a legfőbb hátránya a kezelhetetlenül nagyra növő szimbólumkészlet (lassú és drága hardver), és az, hogy a nagy szimbólumkészlethez nem emlékezet nélküli forrás[8] esetén nehéz a forráseloszlási statisztikákat elkészíteni. Ezért, ha a forrást kiterjesztjük, vagy valamilyen okból nem ismert pontosan a forráseloszlás, akkor a Huffman-kód alapját képező forrásvalószínűségeket a forrás figyelésével futás (kódolás) közben szükséges lehet adaptívan módosítani.

Megmutatható, hogy a becsült forrásvalószínűségekben lévő kisebb hibákra a Huffman-kód nem nagyon érzékeny, azaz a generált kód nem sokkal kisebb hatékonyságú, mint a pontos valószínűségekkel generált változat.

2.4. A Lempel-Ziv-kód

A Lempel-Ziv-kód (röviden: LZ-kód vagy szótár-kód) egészen más elven működik, mint a Huffman-kód. Sok számítógépes **tömörítő** program (pl. unix compress) alapja. Nagy előnye, hogy egyáltalán nem igényli a forráseloszlás ismeretét. A kódoláshoz csak a forrásszimbólumok számának az ismeretére van szükség. Ennek ellenére megmutatható, hogy *elég hosszú kódolt üzenet esetén* az egy szimbólumra jutó átlagos kódszóhossz a forrásentrópiához közelít, azaz a kód optimális.

[7] Hasonlóan a H(A, B)=H(A)+H(B)-hez, független A és B forrás esetén. Ha az A emlékezet nélküli, ez éppen azt jelenti, hogy két egymás utáni (azaz egy kompozit eseményt alkotó) eseménye független egymástól: $H(A\times A)=H(A)+H(A)=2H(A)$.

[8] mint például a természetes nyelvű szöveg

Az LZ-kód kódolás közben folyamatosan egy olyan láncolt lista-szerkezetet épít, amely az elemi szimbólumokban gyökerező fákból áll („erdő”). Ez az erdő a szótár, amelyre az egyes kódszavak hivatkoznak. Az algoritmus lépései:

1. A szótár, az n mutató és az m címregiszterek inicializálása:
   - A szótár oszlopai a cím, a mutató és a szimbólum, induláskor |A|+1 sort tartalmaz, az *i*-edik sor tartalma pedig (i, 0, a_{i-1}). A 0. sor tartalma (0, 0, nil).
   - Induláskor n=0, m=|A|+1, vagyis a címregiszter a szótár végére mutat.
2. Következő szimbólum /x/ beolvasása, kilép, ha nincs több szimbólum
3. *Ha* (n,x) van a szótárban, akármelyik címen
   akkor n = cím(n,x)
   ha nincs, akkor KÜLD(n)
   (n,x) tárolása az m által mutatott címre
   m = m+1
   n = cím(0,x)
4. GOTO 2

Az algoritmusban KÜLD helyett természetesen TÁROL is állhat, ha ez a cél. Ha a kódolandó szimbólumsorozat végére értünk, még az n regiszter tartalmát is el kell küldeni.

Látható, hogy az algoritmus megpróbál a forrás szimbólumsorozatában olyan részsorozatot keresni, amilyen már van a szótárban. Ha a bejövő szimbólum már nem illik rá az eddig követett lánc végére, vagy a láncnak vége van, akkor küldi csak el a sorozat végének a címét, ami (a szótár birtokában) valóban meghatározza az egész sorozatot. Ezután visszaáll a fa gyökerére: n = cím(0,x). Ezért, ha legközelebb megint előfordul ez a sorozat (például egy gyakori szó egy szövegben), már *egy szimbólummal tovább* lesz képes követni a szótár bővítése és információküldés nélkül. Elég hosszú idejű kódolás után pedig akár Az ember tragédiája teljes szövege is egyetlen n mutatóval tárolható.

Hogyan kezelhető ez az algoritmus a 2.1 fejezet fogalmaival? Mivel egyetlen kódszavat állít elő tetszőlegesen hosszú bemeneti szimbólumsorozatból, ezért azt mondhatjuk, hogy a forrást tetszőleges mértékben kiterjeszti, és a hiányzó forráseloszlást a kódolás módosításával (adaptivitással) pótolja.

Dekódolás

A vett címsorozatból a forrásABC ismeretében rekonstruálható mind a szótár, mind pedig a kódolt szimbólumsorozat.

1. A szótár, az n mutató és az m címregiszterek inicializálása:
   - A szótár oszlopai a cím, a mutató és a szimbólum, induláskor |A|+1 sort tartalmaz, az *i*-edik sor tartalma pedig (i, 0, a_{i-1}). A 0. sor tartalma (0, 0, nil).
   - Induláskor n=0, m=|A|+1, vagyis a címregiszter a szótár végére mutat.
2. Következő címmutató /n/ beolvasása, kilép, ha nincs több
3. A szótárbejegyzések hivatkozásainak a gyökérig visszafelé követésével meghatározzuk a gyökérszimbólumot (a_x).
4. A szótár m-edik helyének a mutató oszlopába beírjuk n-et. A bejegyzés szimbólum oszlopát egyelőre üresen hagyjuk.
5. A szótár m-1-edik helyének a szimbólum oszlopába beírjuk a_x–et. Ezt a lépést a legelső vett cím feldolgozásakor értelemszerűen ki kell hagyni.

6. dekódoljuk azt a szimbólumsorozatot, amelyik az n címen kezdődik.
7. m = m+1
8. GOTO 2

2.4. példa 3 elemű forrás LZ-kódolására:

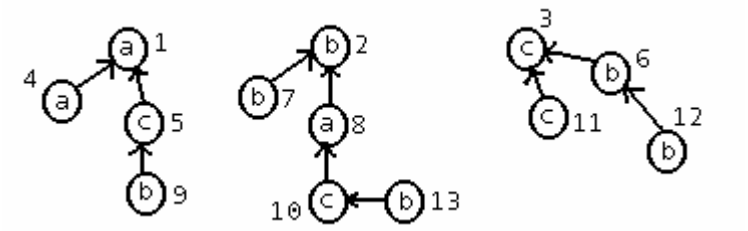
Legyen A={a,b,c}*, és kódoljuk az „*a a c b b a c b a c c b b a c b*” sorozatot!*

Az elküldött címsorozat: 1 1 3 2 2 **5 8** 3 **6 10**, *n* = 2 *(vastagon szedve a több szimbólumból álló sorozatot kódoló címek)*

A kódolás közben felépített szótár:

Cím (m)	*mutató (n)*	*szimbólum*
0	0	nil
1	0	a
2	0	b
3	0	c
4	1	a
5	1	c
6	3	b
7	2	b
8	2	a
9	5	b
10	8	c
11	3	c
12	6	b
13	10	b*, és ekkor* n=2

A szótár tartalma grafikusan (az „erdő”):



Gyakorlati megfontolások

Ha a címeket b biten tároljuk (az m regiszter hossza b), akkor a szótár maximális mérete 2^b. Tehát ha b túl kicsi, ez problémát okozhat kódolás közben. Ha viszont túl nagy, akkor feleslegesen pazaroljuk a tárolási/küldési kapacitásunkat, hiszen a ténylegesen elküldött kódolt üzenet a szótár címeiből áll össze.

2.5. Az aritmetikai kód

Ez a kódolási módszer hasonlít a Huffman-kódhoz, amennyiben a forráseloszlás a priori ismeretét igényli, illetve az LZ-kódhoz, amennyiben a forrásszimbólumok egy hosszú sorozatához egyetlen kódszavat (egy valós számot) rendel hozzá. Gyakorlati alkalmazásai többek közt a képtömörítés és a mozgókép-tömörítés.

Az algoritmus alapötlete az, hogy minden forrásszimbólumnak a valószínűsége arányában megfeleltetjük a [0,1) intervallum valamekkora részét. Ha például A={a, b,

c}={0.4, 0.4, 0.2}, akkor három részintervallumot képezünk: [0, 0.4), [0.4, 0.8) és [0.8, 1). Az első szimbólum (legyen például b) kiválasztja valamelyik részintervallumot (jelen esetben a [0.4, 0.8)-at). Ezután már ezt tekintjük alap-intervallumnak, és ugyanúgy felosztjuk részintervallumokra (a valószínűségek arányában), mint a [0,1) intervallumot. A példa szerint azt kapjuk, hogy [0.4, 0.56), [0.56, 0.72) és [0.72, 0.8). Ezek közül a második szimbólum (például c) választ egyet (a [0.72, 0.8)-at), és így tovább. A feldolgozott szimbólumok számával az aktuális részintervallum egyre szűkül, ha kis valószínűségű szimbólumok jönnek, akkor gyorsabban. Végül, ha már nincs több kódolandó szimbólumunk, akkor az utolsó részintervallumból választott bármelyik valós szám meghatározza az egész addig kódolt sorozatot.

2.5. példa

Legyen A={a, b, c, d}={0.5, 0.3, 0.15, 0.05}

*Ekkor a „*b a a c b*” szimbólumsorozatnak megfelelő részintervallum:*

[0.57425, 0.5748125)

Bármelyik, ebbe az intervallumba eső számot választhatjuk.

Az eredményül kapott intervallumból olyan számot célszerű választanunk, amely bináris csatornán vagy adathordozón kis költséggel továbbítható. Ezért az intervallum legrövidebb (legkevesebb digitet tartalmazó) kettedes törtjét választjuk. A fenti [0.72, 0.8) intervallumban ez a 0.75 lenne. Ezt aztán kettedes törtként („0.11”, illetve a kezdő 0 megtakarításával „11” alakban) továbbíthatjuk.

2.6. példa: Legyen A={a, b}={2/3, 1/3}*, és határozzuk meg a 3 hosszúságú szimbólumsorozatok intervallumhatárait!*

szimb. sorozat	*intervallum*	*választott szám*	*kettedes tört hossza*
bbb	[0, 0.0366)	0.03125	5
bba	[0.0366, 0.11)	0.06125	4
bab	[0.11, 0.183)	0.125	3
baa	[0.183, 0.33)	0.25	2
abb	[0.33, 0.4033)	0.375	3
aba	[0.4033, 0.55)	0.5	1
aab	[0.55, 0.7)	0.625	3
aaa	[0.7, 1)	0.75	2

A példa azt próbálta demonstrálni, hogy minél valószínűbb egy szimbólumsorozat, annál szélesebb intervallum tartozik hozzá, és annál inkább található abban rövid kettedes tört, azaz annál rövidebb kódszó fog hozzá tartozni.

A kód visszafejtése értelemszerűen a legelsőnek kódolt szimbólum meghatározásával kezdődik.

Megfigyelhető, hogy egy szimbólum kódolásának a hatása egy szám hozzáadása az intervallumhatárokhoz, ami a határok közül végül választott kettedes tört több helyiértékén (digitjén) is okozhat változást. Ilyen módon az aritmetikai kód

megosztja a kódbiteket az egyes szimbólumok között, tehát mentes a Huffman-kód kvantáló hatásától.

Gyakorlati megfontolások

A dekódolásnál problémát okoz, hogy ha a szimbólumok egy adott hosszúságú sorozatát kódoljuk mindig egy kódszóba, akkor mi garantálja azt, hogy az így kapott változó hosszúságú kód megfejthető lesz? Ha viszont azonos hosszúságú kódszavakat alakítunk ki, azaz akkor hagyjuk abba a kódolást, mikor egy adott kettedestört-hosszat elértünk, akkor honnan tudja a dekóder, hogy mikor kell leállni? Erre a problémára egy új STOP szimbólum beiktatása lehet a megoldás. Ezt a forrásABC egyéb szimbólumaival azonos módon kell kezelni, valószínűséget kell hozzá rendelni.

Egy másik probléma az, hogy a véges numerikus pontosság (aritmetikai túlcsordulás az intervallumhatárok számításánál) miatt nem kódolhatunk akármilyen hosszú szimbólumsorozatot egy kódszóba. A „végtelen" pontosságú aritmetikát számítástechnikai módszerekkel lehet szimulálni. Tegyük fel például, hogy már valahonnan tudjuk, hogy az intervallumhatárok első néhány digitje biztos nem fog változni. Ha pl. 0.2 és 0.4 között vagyunk, azaz mindkét határ kisebb, mint 0.5, akkor az első digit biztosan 0. Ilyenkor a leendő kódszó stabilizálódott helyiértékeit már el lehet küldeni, és csak a végével számolni tovább.

2.6. A forráskódolási eljárások értékelése

A bemutatott három forráskódolási eljárás alapvető célja volt a források veszteség nélküli „**tömörítése**", azaz olyan bináris kódszavak meghatározása, melyekkel a forrásszimbólumokra jutó átlagos kódszóhossz minél jobban megközelíti az elméleti minimumot, a forrásentrópiát. Optimális esetben, ha sikerül elérni a minimumot, a tömörített üzenet minden bináris csatornaszimbóluma *átlagosan* 1 bit információt hordoz.

A legrégebbi és legegyszerűbb módszer a **Huffman-kódolás**. Előnye, hogy egyszerű, gyors hardverrel megvalósítható (csak egy táblázatból kell a kódszavakat kiolvasni), és hogy a forrás kiterjesztése esetén egyre inkább optimális kódot szolgáltat. Hátránya, hogy kicsi, vagy szélsőséges eloszlású forrásABC esetén gyakran rossz hatékonyságú kódot szolgáltat, éppen azért, mert a kódszókészlet rögzítése után egy adott szimbólum előfordulásakor *mindig* ugyanazt az *egész* számú bináris szimbólumból álló kódszót küldjük a csatornára (kvantáló hatás). Hatékonysága tehát nem javul a kódolt sorozat hosszával. Ugyanakkor—ha a forrás emlékezet nélküli—forráskiterjesztéssel a kód az optimálishoz közelít, a kódoló egység drágábbá és lassabbá válásának az árán.

A Huffman-kód alternatívájának is tekinthető a szintén a priori ismert forráseloszlással dolgozó **aritmetikai kód**, ha a forrásABC túl kicsi. Előnye, hogy a kódbitek megosztása révén elkerüli a kvantáló hatást és aszimptotikusan optimális kódot generál, hátránya a viszonylagos bonyolultsága.

Végül egészen más elvek alapján működik az **LZ-kód**, amely nem igényli a forráseloszlás ismeretét. A jó hatékonyságú kód eléréséhez hosszú szimbólumsorozatra van szükség.

Fontos megjegyezni, hogy *nincs* olyan veszteségmentes tömörítő algoritmus, amely egy bináris forrás tetszőleges N hosszúságú szimbólumsorozatát *minden esetben* akár csak 1 bittel tömörebben, azaz legfeljebb N-1 bináris szimbólummal kódolni

tudná. Ez könnyen belátható a doboz-elv segítségével (lásd az F3 függeléket). A kódok hatékonyságára vonatkozó megállapításaink csak ***átlagosan*** érvényesek.

2.7. példa:

A 2.3. példában a forrás naiv kódolásával L(A)=2, Huffman-kóddal L(A)=1.7 adódott. Ha azonban a forrás hosszú időn keresztül mindig csak az a2 és a3 szimbólumot bocsátja ki, melyekhez a Huffman-kód 3-hosszú kódszót rendelt, akkor Huffman-kóddal 3, naiv kóddal 2 csatornahasználatba kerül egy szimbólum továbbítása.

Hogyan lehet ez? Úgy, hogy kódot átlagos *(várható) esetre terveztük, nem erre a speciális szimbólumsorozatra.*

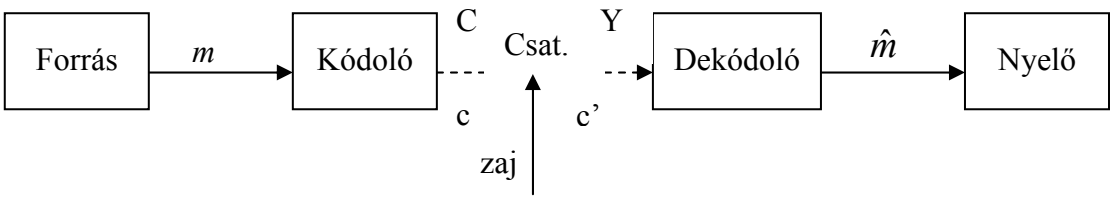
A forráskódolás során feltételeztük, hogy a csatorna zaj nélküli, és hogy a kódolás során nem veszhet el információ (a kód megfejthető). A **veszteséggel** történő kódolás fő alkalmazási területei a kép-, hang-, és mozgókép-tömörítés. Ezek az emberi érzékelés sajátosságait használják ki, ebben a jegyzetben bővebben nem tárgyaljuk őket.

Ha figyelembe vesszük a csatorna zaját, akkor a **csatornakódolás** területére lépünk át.

II. CSATORNAKÓDOLÁS

3. Csatornakódolási alapok

A zajos csatornán való információtovábbításra a következő modellt használjuk:



3.1. ábra Az információtovábbítás általános modellje

A csatornakódolás során tehát feltesszük, hogy a forrás, ill. a belőle érkező m üzenet már **eleve kódolt** (tömörített), és feladatunk ezt úgy átjuttatni a nyelőhöz (mely magában foglalja a forrás-dekódolót is), hogy minél kevesebb információ vesszen el a zaj következtében. Feltételezzük, hogy az m üzenet **bináris** szimbólumokból áll.

A zaj hatásának leküzdésére az alapvető módszerünk m kellőképpen redundánssá tétele lesz, mivel a **redundancia** felhasználásával a dekódolóban lesz esély a hibák javítására, azaz a legvalószínűbb m' üzenet-becslő meghatározására. Tehát: a forráskódolás során *csökkentettük* (optimális esetben és átlagosan meg is szüntettük) a redundanciát, most pedig *növeljük*, de ezúttal a csatorna és a probléma jellegének megfelelő módon és mértékben. A redundancia növelésének a módja a leggyakoribb ún. *blokk-kódok* esetén a bemeneti folyamatos bitfolyam (megfejthető kód egymás utáni kódszavai) K hosszú blokkokra osztása, és ezekből N hosszúságú (N>K) csatornakód-blokkok számítása lesz.

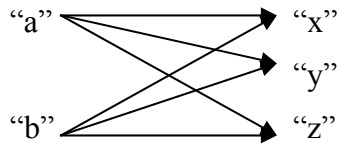
Az N és K számokat **kódparamétereknek** nevezzük, és a velük jellemzett kódot „(N, K) paraméterű kódnak" vagy röviden „(N, K) kódnak" nevezzük.

3.1. Csatornák jellemzése

Az információtovábbítás közegéül definiáljuk a **diszkrét emlékezet nélküli csatornát** (angol rövidítéssel DMC) a következőképpen:

- a csatorna bemenetén ütemenként egy szimbólumot fogad, és ütemenként egy szimbólum jelenik meg a kimenetén (**szinkron** működés)
- a bemeneti és a kimeneti szimbólumkészlet nem feltétlenül azonos, de mindkettő rögzített és véges számú elemet tartalmaz (**diszkrét**)
- ha a bemeneti szimbólumok egymástól függetlenek, akkor a kimeneti szimbólumok is függetlenek lesznek (**emlékezet nélküli**)

3.1. példa: csatorna(átmeneti) gráf



A csatornának 2 bemeneti és 3 kimeneti szimbóluma van

Jelölje c_i a csatorna i-edik bemeneti szimbólumát, y_j pedig a j-edik kimeneti szimbólumot. Ekkor egy valódi csatornán akár mérésekkel (empirikus úton), akár máshogy meghatározhatjuk, hogy a csatorna mekkora valószínűséggel fogja produkálni c_i bemenet hatására az y_j-t, azaz meghatározhatjuk a $p(y_j|c_i)$ feltételes valószínűséget. Ha $p(y_j|c_i)$-t az összes lehetséges i-re és j-re meghatározzuk, ezekkel információelméleti szempontból teljesen jellemezni tudjuk a csatornát. Annak a valószínűsége, hogy a kimeneti oldalon egy ütemben y_j-t veszünk:

$$p(y_j) = \sum_i p(y_j \mid c_i)\, p(c_i)$$

A $p(y_j|c_i)$ feltételes valószínűségekből összeállíthatjuk a $\mathbf{P}(Y|C)$ ún. **csatornamátrixot**, melynek j-edik sorában és i-edik oszlopában $p(y_j|c_i)$ áll. A csatornamátrix i-edik oszlopa megadja a kimeneti valószínűségek eloszlását, ha a bemenetre c_i érkezett, ezért a mátrix minden oszlopának összege 1. Az átmeneti feltételes valószínűségeket rá lehet írni a csatorna-gráf éleire is.

Ha a csatornán nincs zaj, akkor a bemeneti szimbólum egyértelműen meghatározza a kimeneti szimbólumot, és minden $p(y_j|c_i)$ valószínűség vagy 0 vagy 1. Ha a zaj növekszik, a valószínűségek ettől eltérnek, és a kimeneti oldal szimbólumából egyre kevésbé tudjuk kitalálni a bemeneti szimbólumot. A csatorna használhatóságát pedig alapvetően a bemenetről a kimenetre átlagosan átvihető információ mennyisége, tehát a bemenet és a kimenet, mint két forrás közti **kölcsönös információ** határozza meg.

3.2. példa: *Az előbbi csatorna mátrixa legyen:*

$$\mathbf{P}(Y \mid C) = \begin{bmatrix} 0.8 & 0.05 \\ 0.15 & 0.15 \\ 0.05 & 0.8 \end{bmatrix}$$

Tegyük fel, hogy a bemenet $C=\{0.5, 0.5\}$ *Mekkora a bemenet ill. a kimenet entrópiája, és kettejük kölcsönös információja?*

A definíció alapján: $H(C) = 1$ bit, $H(Y) = 1.45$ bit, $I(C,Y) = 0.575$ bit

Figyeljük meg, hogy a $p(y_1|c_0)=p(y_1|c_1)$ *miatt egy kimeneti* "b" *szimbólum nem hordoz semmi információt a C-vel kapcsolatban, függetlenül a C eloszlásától.*

A példa olyan esetet mutat, amikor csatorna zaja miatt a bemeneti 1 bit átlagos információtartalomból a kimenetre csak 0.575 bit jutott el, bár a kimeneten 1.45 bit átlagos információtartalmat mérünk. (A meddő információ a csatornáról magáról informál minket, lásd falusi pletyka.) Ha az átvitt információ mennyiségét nem kötjük egy konkrét bemeneti eloszláshoz, definiálhatjuk a **csatorna C kapacitását**, mint a

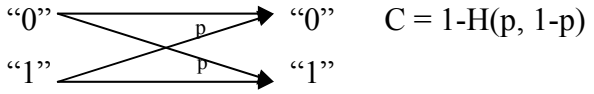
csatornahasználatonként átlagosan átvihető maximális információ mennyiségét *tetszőleges* bemeneti eloszlás mellett:

$$C = \max_{P(C)} I(C,Y)$$

A csatornakapacitás elméleti meghatározása nem könnyű feladat, de létezik rá tetszőlegesen pontos eredményt szolgáltató numerikus algoritmus (az Arimoto-Blahut algoritmus) is. Bővebben lásd az irodalomban.

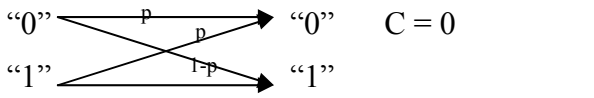
3.3. példa: néhány fontos csatorna és kapacitása

***bináris szimmetrikus csatorna**, BSC: a tévesztés valószínűsége* p



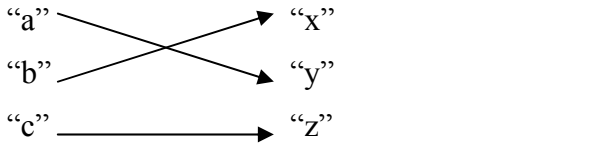
C = 1-H(p, 1-p)

0-átvitelű csatorna

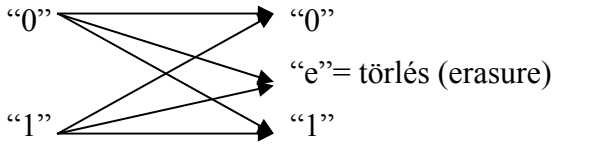


C = 0

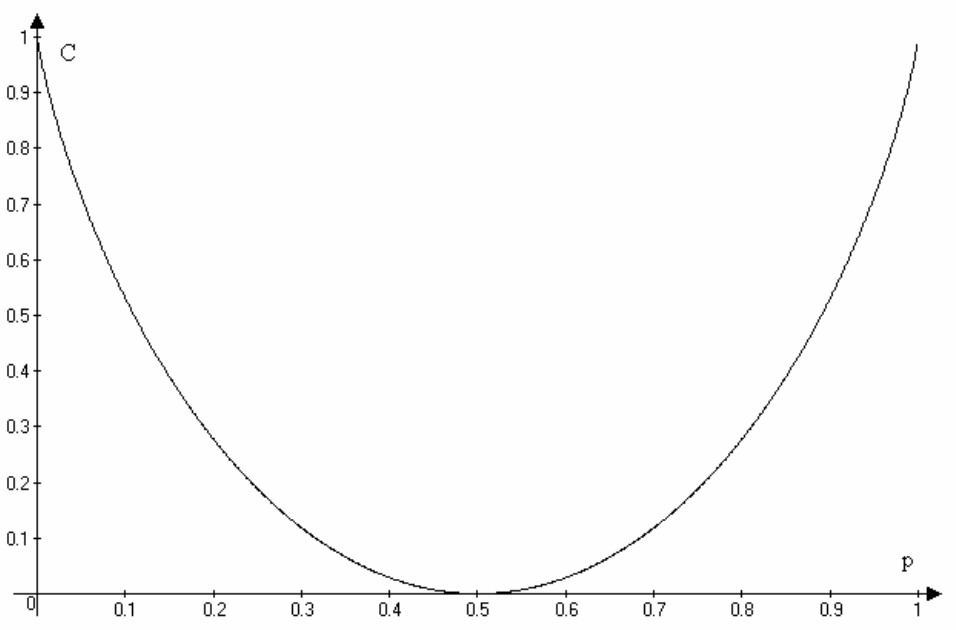
zajmentes csatorna



bináris törléses csatorna



A BSC kapacitása külön vizsgálatot érdemel (3.2. ábra). Látható, hogy a p=0-val jellemzett csatorna épp olyan zajmentes, mint a p=1-gyel jellemzett. Ez utóbbit *invertáló* csatornának is nevezik, mivel megfelel annak az embernek, aki *mindig* rossz tanácsot ad. Ha mindig az ellenkezőjét tesszük annak, amit mond, nem kerülhetünk bajba.



3.2. ábra A bináris szimmetrikus csatorna C kapacitása a p hibavalószínűség függvényében

A csatorna kapacitása hasonló az út szélességéhez: csak a csatorna műszaki konstrukciójától függ, nem pedig az éppen rajta továbbított információtól (mint ahogy egy út szélessége sem függ attól, hogy éppen milyen rajta a forgalom). Mivel a kapacitást éppen a maximális kölcsönös információval definiáltuk, ezért világos, hogy a kapacitást meghaladó információmennyiség *átlagosan* nem szállítható a csatornán. Ez persze egyedi esetekre nem vonatkozik, hiszen előfordulhat, hogy pl. egy eredeti, csatornakódolás nélküli üzenet a csatorna zaja ellenére is torzítatlanul átmegy.

Kérdés, hogy hogyan lehet, lehet-e olyan kódolási módszert találni, amelyik teljesen **kihasználja a kapacitást**, és átlagosan mégis **veszteség nélküli** információtovábbítást garantál?

3.2. A csatornakódolási tétel

A választ **Shannon II. vagy fő tétele** (más néven **a csatornakódolási tétel**) adja meg:

Bővítsük ki egy bináris forrás K hosszúságú szimbólumblokkjait N hosszúságúra redundáns szimbólumok hozzáadásával, és továbbítsuk ezeket a blokkokat egy C kapacitású zajos csatornán!

Ekkor K/N<C esetén **található** olyan kibővítési szabály (azaz csatornakódolási és -dekódolási algoritmus), amellyel a vevőoldalon a hibás blokkdekódolás valószínűsége tetszőlegesen kicsi ε>0 küszöb alá szorítható, ha K elegendően nagy.

Ha pedig K/N>C, akkor **nem található** olyan kódolás, amely tetszőlegesen kicsi vevőoldali hibavalószínűséget tenne lehetővé.

Mit jelent a tételben említett **vevőoldali hibás blokkdekódolás**? Ha a blokkméretet redundáns szimbólumokkal megnöveljük, (pl. úgy, hogy megismételjük a kódszót), akkor az ilyen módon „felhizlalt” blokk kerül a csatorna bemenetére. A csatorna bizonyos valószínűséggel bármelyik „0” vagy „1” szimbólum továbbítása közben hibázhat (nem az jön ki, ami bement). A beépített redundancia segítségével van esélyünk, hogy a vevőoldalon észrevesszük a hibát, és ki is tudjuk javítani. Ekkor $\hat{m} = m$. Ha azonban már a hiba megtörténtét sem vesszük észre, például mert a csatorna egy másik érvényes kódszóba vitte át a bemenet, akkor nincs esélyünk a hiba

javítására. Az is előfordulhat, hogy észrevesszük, javítunk, de nem találjuk el az eredeti kódszót. Ha ***ezek*** közül valamelyik megtörténik, akkor beszélünk vevőoldali hibás blokkdekódolásról.

A tétel **pozitív** állítása ellenére

- nem adja meg a kérdéses csatornakódolási algoritmust,
- az ε csökkentésével a K blokkméret rohamosan növekvő értékeit írja elő,
- és az információveszteség nélküli, hibamentes dekódolást csak átlagosan (nem konkrét esetekre) garantálja.

A tétel **negatív** állítását úgy is meg lehetne fogalmazni, hogy "a továbbítandó információ sűrűségét a csatornakapacitás alá kell higítani a redundáns szimbólumok blokkbéli arányának a növelésével, ha a vevőoldali hibák valószínűségét korlátozni akarjuk". Ha pedig a vevőoldali hibák valószínűségét *nem* tudjuk korlátozni, akkor a *kommunikációs rendszerünk használhatóságát* ***még átlagos esetben sem*** *tudjuk biztosítani*.

A kódsebesség

Vezessük be az R kódsebességet a következőképpen:

$$R = \lim_{n\to\infty} \frac{H(c_0, c_1, \ldots c_{n-1})}{n},$$

ahol $H(c_0, c_1, \ldots c_{n-1})$ az n-szeresére kiterjesztett C forrás entrópiája. (A C forrás alatt itt a csatornára kerülő csatornaszimbólumokat értjük, tehát a csatornakódoló kimenetét. A 0…n-1 index itt időbeli sorozatot jelöl.) A 2.3. fejezetben leírtak szerint R≤H(C), az egyenlőség pedig azt jelenti, hogy a C forrás emlékezet nélküli, az egymás után következő szimbólumai statisztikailag függetlenek egymástól. R tehát az *egy csatornára küldött szimbólumra jutó* átlagos információt méri.

A csatornakódolási tételt a kódsebesség használatával is kimondhatjuk, ha K/N helyére R-et helyettesítünk. Így a tétel már nemcsak blokk-kódokra lesz alkalmazható. A blokk-kódok speciális esetében minden N hosszúságú blokk csak annyi információt tartalmaz, amennyit a K hosszúságú üzenetblokk, hiszen a kiterjesztés nem jár információ-hozzáadással (az üzenetblokk egyértelműen meghatározza a kódszót). Az üzenetblokk pedig, optimálisan tömörített forrás esetén, maximum K bit információt tartalmazhat. Mivel a zajos csatornán továbbítás szempontjából ez a legkényesebb eset, ezért a csatornakódolás során feltesszük, hogy a kódoló bemenetén valóban K bit az információtartalma egy K hosszúságú blokknak. Így tehát az N db csatornára küldött szimbólumra jutó átlagos információ K bit, az egy szimbólumra jutó átlagos információ (az R kódsebesség) pedig K/N bit lesz.

3.4. példa: csatornakódolás 2+1 paritásbittel, az 1.1. példa 3 esete

A kódparaméterek: (3, 2)

A kód: (páratlan számú 1 *esetén* 1 *a redundáns rész, egyébként* 0*)*

m_i	c_i
00	000
01	011
10	101
11	110

Látható, hogy a kiterjesztés nem növelte az üzenet entrópiáját,
$H(C) = H(M) = 2$ bit, $R=2/3$ bit.

3.3. Hibajavítás és -jelzés

A dekóderben az N hosszúságú kódszóba épített redundancia segítségével próbáljuk észrevenni és javítani a hibát. A redundancia azt jelenti, hogy a 2^N lehetséges kódszóból csak 2^K érvényes, mivel az eredetileg kódolt üzenetblokkok hossza K. A dekóder természetesen tudja, melyek az érvényes kódszók, és ha a vett kódszó nem érvényes, akkor ezek közül próbálja a „legvalószínűbbet" kiválasztani és továbbítani. Ha jól választott, akkor sikeres volt a **hibajavítás**.

Az is elképzelhető, hogy a dekóder érvénytelen kódszó vételekor nem javít, csak **jelzi** az adó oldalnak a hibát, és a kódszó **újraküldését** kéri egészen addig, míg érvényes kódszót nem kap. Újraküldéssel egy kódszót esetleg többször is el kell küldenünk, ami rontani fogja a csatorna kihasználtságát, a kódsebességet.

Természetesen az is előfordulhat, hogy a csatorna a bemeneti kódszót egy másik érvényes kódszóba viszi át. Ilyenkor a javításra vagy jelzésre nincs lehetőség. Ezért nem is érhető el a csatornakódolási tételben az $\varepsilon=0$.

A kódszavak közötti választás megkönnyítésére vezessük be két kódszó d_H **Hamming-távolságát**, mint a két kódszó egymástól különböző helyiértékeinek a számát.

3.5. példa

A $c_1=[011010]$ *és a* $c_2=[010110]$ *kódszavakra* $d_H(c_1, c_2) = 2$

A Hamming-távolság **távolságmérték**, mivel, ha a és b N hosszúságú bináris kódszavak, akkor:

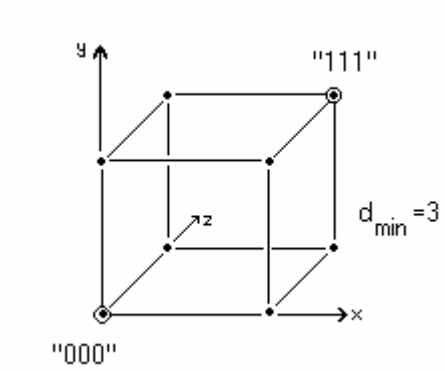
1. ha $d_H(a, b) = 0$, akkor $a = b$
2. $d_H(a, b) = d_H(b, a)$
3. $d_H(a, c) \leq d_H(a, b) + d_H(b, c)$

Tegyük fel, hogy dekóderünk hibajavításkor az érvényes kódszavak közül a vett kódszóhoz d_H szerint **legközelebbit** választja, ha több ilyen is van, akkor ezek közül véletlenszerűen választ. Megmutatható, hogy ez a döntési stratégia azt a kódszót választja ki, melyet a forrás statisztikai értelemben véve a legvalószínűbben küldött (maximum likelihood döntés), mégpedig a következő két feltétellel:

1. A használt BSC hibavalószínűsége $p<0.5$
2. A csatorna bemenetén minden kódszó (azaz a kódoló bemenetén minden K hosszúságú üzenet) egyforma valószínűséggel fordul elő.

Ez utóbbit feltehetjük, hiszen a csatornakódolási erőfeszítéseink alapfeltevése az, hogy a forrás már közel optimálisan kódolva (tömörítve) van.

*3.6. példa: a **Hamming-kocka** a hibajavítás szemléltetésére*



A kocka (3,1) *kódot mutat, a lehetséges 8 kódszóból 2 érvényes. Akkor tudunk a legjobban hibát javítani, ha ezek egy testátló két végén vannak (bekarikázott csúcsok). Bármelyik érvénytelen kódszóhoz egyértelműen megtalálható a hozzá legközelebbi érvényes kódszó: ebbe fogunk javítani.*

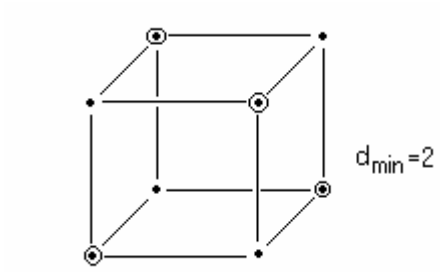
*Ha az ábrán látható átlót választjuk, akkor az ún. **háromszoros ismétléses** kódot kapjuk.*

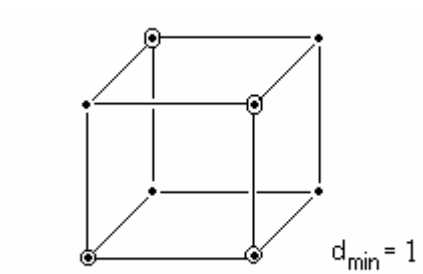
A javítás és jelzés kompromisszuma

A kód javíthatósága szempontjából alapvető fontosságú, hogy mekkora az érvényes kódszavak között előforduló Hamming-távolságok minimuma, a d_{min} **kódtávolság**. Érezhető, (a Hamming-kockán látható is) hogy nagy kódtávolság több hiba javítását teszi lehetővé.

3.7. példa:

A [0 1 1 1], [1 0 0 0], [0 1 1 0], [1 0 1 1] *kódszavakból álló kódkészlet kódtávolsága* d_{min}=1.

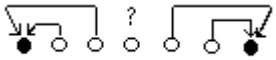




A 3.4. példa paritáskódja, és egy rosszabb kód

Legyen a javítani kívánt hibák száma t_{jav}! Könnyen belátható, hogy ekkor minden kódszónak minden kódszótól legalább $2t_{jav}$ távolságra kell lennie ($d_{min} \geq 2t_{jav}+1$), máskülönben rossz kódszóba fogunk javítani. Ezt szemlélteti a 3.8. példa ábrája, melyben az üres körök érvénytelen, a telik érvényes kódszavakat jelentenek. „Középen” pedig nem tudjuk eldönteni, melyik kódszóra tippeljünk.

3.8. példa: hibajavítás d_{min}=6 *esetén*



Az is látható, hogy *csak* jelzés esetén $d_{min} \geq t_{jel}+1$ szükséges. Ha a fenti példában 2 hibát javítunk, akkor a középső esetben, mikor mindkét kódszótól egyenlő távol, 3-ra vagyunk, csak jelzünk. Ekkor azt mondjuk, hogy 3 hibát jeleztünk. Hibajavítás nélkül 5 hibát is tudtunk volna jelezni. 6 hiba pedig érvényes kódszót jelent.

Sok esetben bizonyos hibaszámig javítunk, afelett pedig csak jelzünk. Ennek nyilván csak akkor van értelme, ha $t_{jav} < t_{jel}$. Ha azonban egy hibát már javítunk, annak a jelzésére nincs lehetőség! Ha pl. a fenti esetben 1 hibát javítunk, akkor jelezni már csak 4-et tudunk, mivel az öt hibás kódszót—tévesen—belejavítjuk a másik kódszóba. Összefoglalva:

$$\text{javítás}: \; d_{\min} \geq 2t_{jav}+1$$

$$\text{jelzés}: \; d_{\min} \geq t_{jel}+1, \text{ hiba esetén újraküldés}$$

$$\text{javítás } \textit{és} \text{ jelzés}: \; d_{\min} > 2t_{jav}+1 \; \textit{és}$$

$$d_{\min} \geq t_{jav}+t_{jel}+1$$

3.9. példa

Legyen egy kódra $d_{min}=6$. *Mik a hibajavítás és –jelzés lehetőségei?*

5 hiba jelzés VAGY
1 hiba javítás *és* 4 hiba jelzés, VAGY
2 hiba javítás *és* 3 hiba jelzés

Ha csatornánk a 3.3. példa szerinti bináris törléses csatorna, és a vett kódszóban $t_{tör}$ számú **törlés**-szimbólum, azaz törléses hiba van (de másmilyen hiba nincs), akkor egészen $d_{min} \geq t_{tör}+1$-ig biztosan meg tudjuk mondani az eredeti kódszót, mivel a törléses hibák helye ismert.

A választott stratégiától függően több-kevesebb esélyünk van a rossz kódszóba javítás, azaz a vevőoldali hibás blokk-dekódolás „elkövetésére”. Ez az esély a felhasználó szempontjából igen fontos jellemzője a kódolási módszerünknek, és attól is függ, hogy az adott kódszóban ténylegesen hány hiba történt. Könnyen belátható, hogy a **hibák számának** a várható értéke N blokkméret mellett egy p hibavalószínűségű BSC-n Np, a pontosan **t darab hiba bekövetkezésének a valószínűsége** pedig:

$$P(t,p,N)=\binom{N}{t}p^t(1-p)^{N-t}.$$

Az alábbi példa az újraküldés miatt csökkenő kódsebesség, viszont javuló hibásblokk-dekódolási valószínűség kompromisszumát mutatja be.

3.9. példa

Tekintsük a (7,1) *ismétléses kódot egy* p=0.05 *hibavalószínűségű BSC-n! Vessük össze a 3 hiba javításos ill. az 1 hiba javítás/5 hiba jelzéses vevőoldali stratégia kódsebességét és hibásblokk-dekódolási valószínűségét!*

t_{jav} = 3 *esetén* P(hbd) = P(4,5,6 vagy 7 hiba) = **1.94e-4**,
R=K/N=1/7=**0.143** bit

$t_{jav} = 1$, $t_{jel} = 5$ *esetén* P(hbd) = P(6 vagy 7 hiba) = **1.05e-7**,
R = K/N', ahol N' = N/(1-P(újraküldés)) = N/(1-P(2,3,4 vagy 5 hiba)),
azaz R = **0.136** bit
N'-ben vettük figyelembe a blokk-újraküldések hatását.

A csatorna kapacitása **0.71 bit**, *jóval nagyobb, mint az elért kódsebességek, pedig a hibásblokk-dekódolási valószínűségek meglehetősen nagyok. A* (7,1) *ismétléses kód mindkét stratégiával* igen gyengén *szerepel, bár a kombinált javítás/jelzés jobb, mint a puszta javítás. Azonos hibavalószínűség mellett jobb kódsebesség eléréséhez* növelni *kell a K blokkméretet.*

3.4. A kódtér kitöltése

A Singleton-korlát
A redundáns szimbólumokat azért csatoljuk a blokkhoz, hogy növeljük a kódtávolságot. Optimális esetben minden egyes hozzácsatolt szimbólum eggyel növelheti a kódtávolságot. Például a 3.7. példa bal oldali kockáján a hozzáadott paritásbit 2-re növelte a kódtávolságot, a jobb oldalin a kibővítés eredmény nélkül maradt.

Jelölje r = N-K a redundáns szimbólumok számát! Ekkor a fentiek szerint:

$$d_{\min} \le r + 1$$

Ezt átrendezve, és hatványra emelve kapjuk az ún. **Singleton**-féle korlátot:

$$2^K \le 2^{N - d\min + 1},$$

melynek a bal oldalán a d_{min} mellett maximálisan lehetséges kódszavak száma áll. Egyenlőség esetén, mikor minden redundáns szimbólum sikeresen növelte a kódtávolságot, **maximális távolságú**, vagy **MDS** (maximum distance separable) kódról beszélünk.

3.10. példa: az (N,1) *ismétléses kód és a* (K+1, K) *paritáskód MDS*

Magyarázat: az ismétléses kód esete triviális. A paritáskódnál csak azt kell belátni, hogy ha két K *hosszúságú üzenetszó között* 1 *volt a távolság, akkor a hozzájuk adott paritásbit különböző lesz, miáltal a távolság* 2*-re nő. Ez pedig azért teljesül, mert ha a két üzenetszó között* 1 *a távolság, akkor az egyikben pontosan eggyel kevesebb* 1*-es van, mint a másikban, tehát ha az egyikhez* 1 *paritásbitet fűzünk hozzá, akkor a másikhoz* 0*-t.*

A gömbpakolási korlát
A javítás és a jelzés alapkoncepciója, hogy az érvényes kódszavak körül az N dimenziós térben t_{jav} sugarú, egymást érintő gömböket képzelünk el, és hibajavításkor a gömb belsejét javítjuk a középpontba. A kód „takarékosságára" jellemző, hogy mennyire töltik ki ezek a gömbök az N dimenziós teret.

Mivel a hibák bármelyik pozíción előfordulhatnak, ezért egy érvényes kódszóhoz tartozó, t hibás kódszavak száma

$$\binom{N}{t}$$

A t_{jav} sugarú gömbben lévő kódszavak száma az érvényes kódszóval együtt:

$$\sum_{i=0}^{t_{jav}} \binom{N}{i}$$

Ha mind a 2^K érvényes kódszó gömbjében lévő pontokat összeszámoljuk, ezek száma nem lépheti túl az összes kódszó számát. Ezt felírva az ún. **Hamming-korlátot**, vagy **gömbpakolási** korlátot kapjuk:

$$\sum_{i=0}^{t_{jav}} \binom{N}{i} \leq 2^{N-K}$$

Egyenlőség esetén, mikor a gömbök maradéktalanul kitöltik a kódteret, **perfekt** kódról beszélünk. Ismert perfekt kódok az ismétléses kódok, a Hamming-kódok (lásd a 4.4. alatt) és a Golay-kódok (lásd az irodalomban). Az N és K megfelelő értékei:

$t_{jav} = 1$ ($d_{min} = 3$)	$t_{jav} = 2$ ($d_{min} = 5$)	kód típusa
(3,1)	(5,1)	ismétléses kód
(4,7)		Hamming-kód
(11,15)		Hamming-kód
…		

Az, hogy N és K a fenti egyenlőtlenséget egyenlőséggel elégíti ki, még nem jelenti azt, hogy az adott N, K paraméterekkel tényleg konstruálható t_{jav}-nak megfelelő kódtávolságú kód. Például $t_{jav} = 2$ esetén a (90, 78) megfelelő lenne, de bebizonyítható, hogy ilyen kód (melyre $t_{jav} = 2$ miatt $d_{min} \geq 5$ lenne) nem létezik.

Az eddigiek során két egyszerű csatornakódot ismertünk meg, a paritás- és az ismétléses kódot. Az ismétléses kód azonban kis blokkmérete, a paritáskód pedig a redundáns szimbólumok kis száma miatt csak igen korlátozott gyakorlati jelentőséggel bír.

4. Bináris lineáris blokk-kódok

Egy általános blokk-kódot legegyszerűbben egy olyan táblázattal adhatunk meg, amely mind a 2^K lehetséges üzenethez hozzárendel egy-egy különböző, N hosszúságú kódszót. A dekódolás ugyanezen **kódtáblázat** segítségével történik. Ha a táblázatban nem találjuk a kapott kódszót, tehát az hibás, akkor az érvényes, táblázatban lévő kódszavak közül választjuk a hozzá legközelebb lévőt[9].

A gyakorlatban a blokkméretet általában növelnünk kell a két lényeges célkitűzés, a hibás blokkdekódolás valószínűségének a csökkentése és C megközelítése, együttes kielégítése céljából. K növelésével azonban a táblázat mérete (2^K), ezzel együtt a kódoló/dekódoló egység komplexitása is gyorsan nő, és a legközelebbi kódszó megkeresése is egyre nehezebb.

A **lineáris** blokk-kódok nagy előnye, hogy a kód generálása és javítása ennél **egyszerűbben** és olcsóbban (mátrix-vektor szorzással) lesz megvalósítható. A lineáris blokk-kódok tárgyalásához bevezetjük a kódszavak, mint vektorok által generált N dimenziós vektorteret.

[9] Feltéve, hogy a kód által javítható összes hibát javítani kívánjuk, azaz nem alkalmazunk hibajelzést/újraküldést.

4.1. Kódszavak, mint vektorok

A csatornaszimbólumok matematikai leírásához bevezetjük a **GF[2] testet** az összeadás (+) és a szorzás (*) műveletekkel, a {0, 1} elemekkel és a következő tulajdonságokkal:

- Zártság: a két művelet nem vezet ki a halmazból;
- Asszociativitás a két műveletre: (a+b)+c = a+(b+c), (a*b)*c = a*(b*c);
- Létezik additív egységelem: a+0=a, és multiplikatív egységelem: a*1=a;
- A test bármely eleméhez található additív inverz (-a), melyre a+(-a)=0;
- A test bármely eleméhez, kivéve a 0 elemet, található multiplikatív inverz a^{-1}, melyre a* a^{-1}=1;
- Kommutativitás az összeadásra: a+b=b+a;
- Disztributivitás a két műveletre: a(b+c)=ab+ac ill. (a+b)c=ac+bc;

Megmutatható, hogy a két elemű GF[2] test rendelkezik ezekkel a tulajdonságokkal, ha az összeadás és a szorzás műveletet a szokásos algebrai módon értjük, kivéve, hogy 1+1=0 (az 1 önmaga additív inverze).

A test elemeiből n-eseket (**vektorokat**) képezünk a **kódszavak** leírásához:

$$\mathbf{a} = [a_0 \quad a_1 \quad \ldots \quad a_{N-1}], a_i \in GF[2]$$

A vektorokon két műveletet definiálunk:

- Szorzás skalárral

$$c \cdot \mathbf{a} = [c * a_0 \quad c * a_1 \quad \ldots \quad c * a_{N-1}], c \in GF[2]$$

- Vektorok összeadása

$$\mathbf{a} + \mathbf{b} = [a_0 + b_0 \quad a_1 + b_1 \quad \ldots \quad a_{N-1} + b_{N-1}]$$

A kódszóvektorok alatt a jelölések egyszerűsítése céljából sorvektorokat fogunk érteni, például:

$$\mathbf{e} = [1\ 0\ 1\ 0\ 0\ 0\ 0].$$

A vektorokat félkövér kisbetűvel, a belőlük képzett mátrixokat félkövér nagybetűvel fogjuk jelölni.

A fenti két művelettel a vektorokból és GF[2] elemeiből **vektorteret** ($GF[2]^N$) konstruálunk a következő tulajdonságokkal:

1. Zártság: az összeadás nem vezet ki a halmazból;
2. Kommutativitás az összeadásra: **a**+**b**=**b**+**a**;
3. Asszociativitás mindkét műveletre;
4. Létezik additív null-elem, **0**, mellyel minden $GF[2]^N$-beli **a** vektorra **a**+**0**=**a**;
5. A vektortér bármely vektorához található additív inverz (-**a**), mellyel **a**-**a**=**0**;
6. Minden GF[2]–beli a elemhez és $GF[2]^N$–beli **b** vektorhoz az a**b** is eleme $GF[2]^N$–nek;
7. Disztributivitás a skaláris szorzásra és vektoriális összeadásra;
8. GF[2] multiplikatív egységének az 1-et választva, tetszőleges $GF[2]^N$–beli **b** vektorra 1***b**=**b**.

4.2. Lineáris blokk-kódok tulajdonságai

Egy (N, K) kód értékkészletét az N dimenziós vektortér egy K dimenziós altere adja, amelyet K darab lineárisan független bázisvektor ($\mathbf{g}_i$) határoz meg. Ezekkel generáljuk a kódszavakat. Legyen az üzenet

$$\mathbf{m} = [m_0 \;\; m_1 \;\ldots\; m_{K-1}], \text{ ahol } m_i \text{ a GF[2] eleme.}$$

Ekkor a kódszót a bázisvektorok **m** által generált lineáris kombinációjaként állítjuk elő:

$$\mathbf{c} = m_0\mathbf{g}_0 + m_1\mathbf{g}_1 + \ldots + m_{K-1}\mathbf{g}_{K-1}$$

A bázisvektorokat a **G generátormátrixba** foglaljuk, melynek sorai a $\mathbf{g}_i$ vektorok. Ekkor a kódolás mátrix-vektor szorzássá alakul:

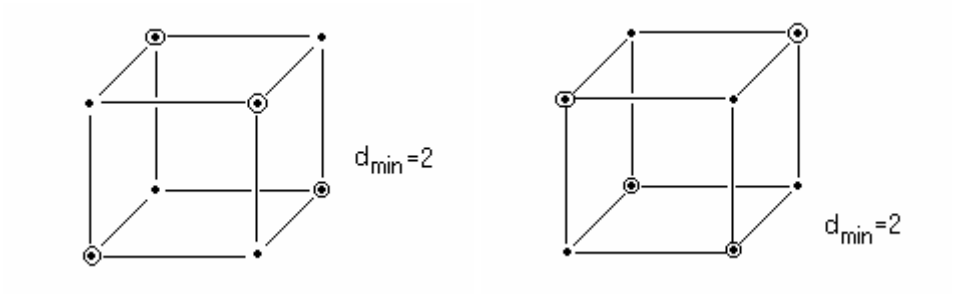
$$\mathbf{c} = \mathbf{mG}$$

Mivel **G** sorai K dimenziós bázist alkotnak, ezért könnyen belátható, hogy

1. a 2^K darab különböző **m** üzenethez 2^K darab különböző **c** kódszót kapunk, (ha két kódszó azonos lenne, akkor a bázis nem volt K dimenziós);
2. két érvényes kódszó összege is érvényes kódszó;
3. mindegyik bázisvektor maga is érvényes kódszó;
4. a **0** nem lehet bázisvektor;
5. a **0** érvényes kódszó (ezért a Hamming-kocka két bemutatott (3, 2) kódja közül csak az egyik lineáris) ;
6. az érvényes kódszavak az N dimenziós kódtér egy K dimenziós alterében vannak. Az altér **alakja** határozza meg a kódtávolságot, ami a lényeg a hibajavítás szempontból—ezért a *hibajavítás* szempontjából mindegy, *melyik* K darab nem **0**, lineárisan független kódszóvektort választjuk bázisul, bár az üzenet-kódszó hozzárendelés természetesen függ ettől.

4.1. Példa:

Két ugyanolyan alakú, azonos kódtávolságú, (3,2) *kód, melyek közül az egyik lineáris, a másik nem, mivel nem tartalmazza a* **0** *kódszót:*



lineáris *nem lineáris kód*

A (3, 1) *ismétléses kód, mint lineáris blokk-kód generátormátrixa:*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$$

Ennek a kódnak a K=1 dimenziós altere a [0 0 0] *ponton átmenő testátló.*

A (3,2) *paritáskód két lehetséges generátormátrixa:*

$$\mathbf{G}_1 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}, \mathbf{G}_2 = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

A kód altere a kockában a fenti ábra bal oldalán látható, az 1+1=0 *miatt „visszahajló” sík.*

Szisztematikus kódról beszélünk akkor, ha a generált kódszó végén, annak utolsó K pozícióján mindig megjelenik az eredeti üzenet. Ekkor a kódszóban lévő r = N-K darab redundáns szimbólum jól elkülöníthető, a dekódolás (de nem a hibajavítás) pedig triviális. A szisztematikus kódok generátormátrixának a végén megjelenik az $E_{K\times K}$ egységmátrix.

Mivel **G** sorai lineárisan függetlenek egymástól, ezért sor-oszlop transzformációkkal tetszőleges **G** szisztematikus alakra hozható. Ezután természetesen más lesz egy adott üzenethez tartozó kódszó, viszont az altér **alakja**, a kódszókészlet és a kódtávolság nem változik.

4.2. Példa:

A (3,1) *ismétléses kód fenti* **G** *mátrixa és a* (3,2) *paritáskód* $\mathbf{G}_1$ *mátrixa szisztematikus kódot határoznak meg.*

Hamming-súlynak nevezzük a kódszóban lévő „1”-es szimbólumok számát. Mivel lineáris blokk-kódokban a **0** mindig érvényes kódszó, és két érvényes kódszó összege is érvényes kódszó, ezért indirekt úton könnyen megmutatható, hogy lineáris blokk-kódokra a kódtávolság egyezik a legkisebb súlyú, nem **0** kódszó Hamming-súlyával, azaz

$$d_{min} = \min(w_H(\mathbf{c}_i)),\ \mathbf{c}_i \neq \mathbf{0}$$

Ennek alapján egyszerűen úgy megállapítható egy generátormátrixával adott lineáris kód kódtávolsága, hogy az összes kódszó generálása után megkeressük közöttük a legkisebb súlyú, nem **0** kódszót, ennek súlya lesz d_{min}.

4.3. Példa:
Ha a fenti (3,2) *paritáskód négy érvényes kódszavát felsoroljuk (ezek a 4.1. példa szerinti* $\mathbf{G}_1$, $\mathbf{G}_2$ *mátrixok sorai és a* **0***), azt tapasztaljuk, hogy a* **0***-t kivéve mindegyikben 2 darab egyes van. Ezek alapján a kódtávolság* d_{min} = 2.

4.4. Példa

Tekintsük a következő kódszókészletet:

$$\mathbf{c}_0 = [0\quad 0\quad 0], \mathbf{c}_1 = [1\quad 0\quad 0], \mathbf{c}_2 = [1\quad 1\quad 1], \mathbf{c}_3 = [0\quad 1\quad 1]$$

Mik a kódparaméterek? Lineáris-e a kód? Mekkora a kódtávolság?

A kódszavak száma és hossza alapján a kódparaméterek: (3,2). A kód lineáris, mivel tartalmazza a **0** *kódszót, és bármely két kódszó összege is érvényes kódszó. Mivel* $\mathbf{c}_1$ *súlya* 1*, ezért a kódtávolság* d_{min} = 1, *vagyis a kóddal* egy hibát sem *lehet jelezni*.

4.3. A paritásellenőrzési tétel

A kódszavak generálása a kódolóban az eddigiek alapján a hardveresen egyszerűen, hatékonyan megvalósítható **mátrix-vektor szorzással** megoldható. A kódolóban csak a **G** mátrixot kell tárolnunk, nem egy 2^K sorú táblázatot. A vevőoldalon az egyszerű, hatékonyan megvalósítható hibajavítás és –detektálás alapja a **paritásellenőrzési tétel**. Ennek kimondásához először a szisztematikus

$$\mathbf{G} = \left[\begin{array}{c|c} \mathbf{P}_{K\times r} & \mathbf{E}_{K\times K} \end{array} \right]$$

generátormátrixhoz készítsük el a hozzá tartozó $\mathbf{H}^T$ **paritásellenőrző mátrixot** az alábbi módon:

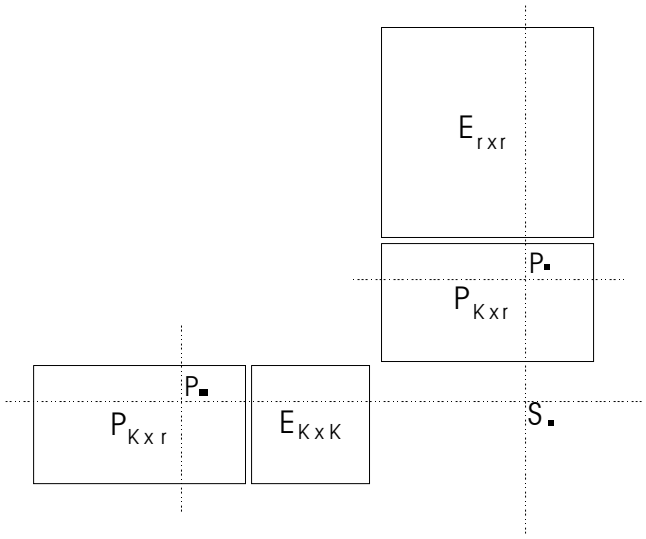
$$\mathbf{H}^T = \left[\begin{array}{c} \mathbf{E}_{r\times r} \\ \hline \mathbf{P}_{K\times r} \end{array} \right]$$

A vevőoldalon a hibadetektálás céljából minden vett **c'** kódszót szorozzunk meg a $\mathbf{H}^T$ mátrixszal, és jelölje az $\mathbf{s} = \mathbf{c'H}^T$ **szindróma** a szorzás eredményét!

Ekkor a **paritásellenőrzési tétel** szerint:

1. az érvényes kódszavak szindrómája **0**,
2. az érvénytelen kódszavak szindrómája nem **0**,
3. a nem **0** szindrómák egyértelműen azonosítják a *kód által javítható* hibákat.

Az *első* állítás olyan módon látható be, hogy az $\mathbf{s} = \mathbf{c}\ \mathbf{H}^T = (\mathbf{mG})\mathbf{H}^T = \mathbf{m}(\mathbf{GH}^T)$-ben felírjuk az $\mathbf{S}=\mathbf{GH}^T$ mátrix egy általános $s_{i,j}$ elemét:



, ezért

$$s_{i,j} = p_{i,j} \cdot 1 + 1 \cdot p_{i,j} = 0 .$$

Látható, hogy a $\mathbf{H}^T$ mátrix szerkezete miatt **G** sorai (a bázisvektorok) ortogonálisak $\mathbf{H}^T$ oszlopaira, ezért a szindróma minden érvényes, tehát **c** = **mG** alakban felírható kódszóra **0**.

A *második* állítás szemléltetéséhez azt fontoljuk meg, hogy ha egy **c** kódszóra $\mathbf{cH}^T = \mathbf{0}$, ez azt jelenti, hogy **c** ortogonális a $\mathbf{H}^T$ összes oszlopvektorára, amelyek viszont a K dimenziós, **G** által meghatározott altérre a fentiek miatt ortogonális, N-K dimenziós alteret határoznak meg. Ezért **c** benne kell, hogy legyen a **G** által meghatározott altérben, vagyis nem lehet érvénytelen kódszó.

A tétel *harmadik* állításának bizonyításához vezessünk be egy **hibamodellt**. Modellezzük úgy a csatornán bekövetkező hibákat, mintha a kódszóhoz egy **e** **hibavektor** adódott volna hozzá. Ha például N = 7 és az első és a harmadik pozíción lévő csatornaszimbólum küldésekor hibázott a csatorna, akkor

$$\mathbf{e} = [1\ 0\ 1\ 0\ 0\ 0\ 0].$$

Mivel mind a hibavektor, mind az eredeti kódszó az N dimenziós tér vektorai, melyekre érvényes a skalár szorzás disztributivitása, ezért a paritás ellenőrzésekor *csak az* **e** *hibavektor* határozza meg a szindrómát:

$$\mathbf{s} = \mathbf{c'H}^T = (\mathbf{c}+\mathbf{e})\mathbf{H}^T = \mathbf{cH}^T + \mathbf{eH}^T = \mathbf{0} + \mathbf{eH}^T$$

Ezt a fontos tulajdonságot a gyakorlati megvalósítás során is fel fogjuk használni.

A harmadik állítás indirekt bizonyításához tegyük fel, hogy két különböző hibavektor azonos szindrómát generál! Elég megmutatnunk, hogy ekkor legalább az egyik nem javítható, tehát a súlya nagyobb, mint a kódtávolság alapján javítható hibák száma, hiszen egy hibavektor súlya egyezik az általa előidézett hibák számával. Ehhez tegyük fel, hogy $\mathbf{e}_1\mathbf{H}^T = \mathbf{e}_2\mathbf{H}^T = \mathbf{s}$, *ahol* $\mathbf{s} \neq \mathbf{0}$ *és* $\mathbf{e}_1 \neq \mathbf{e}_2$. Ekkor $(\mathbf{e}_1+\mathbf{e}_2)\mathbf{H}^T = 0$, tehát $\mathbf{e}_1+\mathbf{e}_2$ érvényes kódszó kell, hogy legyen. Ekkor viszont a kód kódtávolságára a 4.1. fejezetben foglaltak (Hamming-súly) alapján $d_{\min} \le w_H(\mathbf{e}_1+\mathbf{e}_2) \le w_H(\mathbf{e}_1) + w_H(\mathbf{e}_2)$. (Egyenlőség csak akkor áll fenn ha $\mathbf{e}_1$-ben és $\mathbf{e}_2$-ben nincsenek egyesek azonos pozíción.)

Másrészt viszont $d_{\min} \ge 2t_{jav} + 1$ miatt $2t_{jav} + 1 \le w_H(\mathbf{e}_1) + w_H(\mathbf{e}_2)$, amiből látszik, hogy vagy $\mathbf{e}_1$, vagy $\mathbf{e}_2$ súlya nagyobb kell, hogy legyen t_{jav}-nál; ez pedig feltevésünkkel ellentétben nem javítható hibát jelent. □

A paritásellenőrzési tétel alapján a vevőoldali hibajavításhoz elegendő a javítható vagy javítani kívánt hibavektorok szindrómáit tartalmazó táblázatot elkészíteni, és a kapott szindróma hibavektorát ebben kikeresni. A javított kódszót a kapott kódszó és a hibavektor összegeként kapjuk. A szindróma-hibavektor táblázat lényegesen kisebb méretű, mint az az összes érvényes kódszót tartalmazó táblázat, melyre egy nem lineáris kód esetén lett volna szükségünk.

4.5. Példa:
A háromszoros bináris ismétléses kód paritásellenőrző mátrixa:

$$\mathbf{H}^T = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix}$$

Az ehhez tartozó szindrómatábla:

s	**e**
10	100
01	010
11	001

Ha kombinált hibajavítás/jelzéses stratégiát használunk, akkor a szindrómatábla eleve nem tartalmazza a javítani már nem kívánt hibák szindrómáit. Ha a számított szindrómát nem találjuk a táblában, hibát jelzünk, azaz újraküldetjük a kódszót.

4.4. A Hamming-kód

Hamming-kódnak nevezzük az 1 hibát javító perfekt kódokat. A kód konstrukciójához a $\mathbf{H}^T$ paritásellenőrző mátrix szerkezetét adjuk meg, ebből a $\mathbf{H}^T$ -hez tartozó **G** a paritásellenőrzési tétel szerint könnyen megadható.

Adott N és K kódparaméterekhez (melyek kielégítik a perfektségből fakadó $2^{N-K} = 1+N$ feltételt) a következőképpen készítjük el a $\mathbf{H}^T$ mátrixot:

- $\mathbf{H}^T$ tetejére elhelyezzük az $\mathbf{E}_{(N-K)x(N-K)}$ egységmátrixot
- A fennmaradó K sorban tetszőleges sorrendben felsoroljuk az összes 1-nél nagyobb súlyú, N-K elemű bináris vektort. A felsorolandó vektorok száma: $2^{N-K} - (N-K) - 1$, (a -1 a **0** vektor kihagyása miatt szerepel), ennek éppen K-t kell adnia, ami perfektség feltétele miatt teljesül is.

Az így előállított $\mathbf{H}^T$ és az ebből képzett **G** mátrix szisztematikus kódot határoz meg.

4.6. Példa:

A (7,4) *Hamming-kód egy lehetséges* $\mathbf{G}$ - $\mathbf{H}^T$ *párja:*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{H}^T = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

A szindróma generálásakor az 1 súlyú hibavektorok $\mathbf{H}^T$ illető sorát választják ki, például a [0 0 0 0 0 0 1] hibavektor az [1 0 1]-et. Mivel $\mathbf{H}^T$ minden sora különböző és nem **0**, ezért az 1 súlyú hibavektorok mind különböző, nem **0** szindrómát generálnak, tehát a kóddal valóban lehet 1 hibát javítani.

4.7. Példa:

A 4.6. példa Hamming-kódjának alkalmazása 1 illetve 3 hiba esetén.

Kiinduló adatok

Legyen $\mathbf{m} = [0 \quad 1 \quad 0 \quad 0]$,

ekkor $\mathbf{c} = \mathbf{mG} = [1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0]$,

a hibátlan kódszó szindrómája $\mathbf{s} = \mathbf{c'H}^T = \mathbf{cH}^T = \mathbf{0}$,

a küldött kódszó becslője $\hat{\mathbf{c}} = \mathbf{c'}$ *az üzenet becslője:* $\hat{\mathbf{m}} = [0 \quad 1 \quad 0 \quad 0]$

1 hiba javítása

Legyen $\mathbf{e_1} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0]$.

Ekkor $\mathbf{c'} = \mathbf{c} + \mathbf{e_1} = [1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0]$

$\mathbf{s} = \mathbf{c'H}^T = [0 \quad 1 \quad 1]$, *ami* $\mathbf{H}^T$ *hatodik sora, ezért a becsült hiba:*

$\hat{\mathbf{e}} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0] = \mathbf{e_1}$

$\hat{\mathbf{c}} = \mathbf{c'} + \hat{\mathbf{e}} = [1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0]$

$\hat{\mathbf{m}} = [0 \quad 1 \quad 0 \quad 0] = \mathbf{m}$ *(1 hiba sikeres javítása)*

3 hiba javítása

$\mathbf{e}_2 = [1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0]$, $\mathbf{c'} = [0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 0]$,

$\mathbf{s} = [1 \quad 0 \quad 1]$, *ez* $\mathbf{H}^T$ *hetedik sora.*

$\hat{\mathbf{e}} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1]$

$\hat{\mathbf{c}} = [0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1]$

$\hat{\mathbf{m}} = [1 \quad 0 \quad 0 \quad 1] \neq \mathbf{m}$ *(hibás blokk dekódolás, rossz kódszóba javítottunk)*

Mint a 4.7. példán látható, a szindróma alapján mindig meghatározható $\mathbf{H}^T$ egy sora, ehhez a becsült hibavektor, melyet a kapott hibás kódszóhoz adva megkapjuk a küldött kódszó becslőjét. Természetesen akkor, ha valójában nem 1, hanem több hiba történt, akkor rossz kódszóba fogunk javítani, azaz hibásan fogjuk dekódolni a blokkot.

A Hamming-kódok gyakorlati jelentősége csekély, mivel a perfektség feltétele erős megszorítást jelent az alkalmazható N, K paraméterekre[10].

5. Ciklikus kódok

A ciklikus kódok széles körben használt lineáris blokk-kódok. A ciklikus kódokat úgy definiáljuk, mint olyan lineáris blokk-kódokat, melyekben minden érvényes **c** kódszó $\mathbf{c}^e$ ciklikus eltoltja is érvényes kódszó:

$$\mathbf{c} = [c_0 \quad c_1 \quad \dots \quad c_{N-2} \quad c_{N-1}], \quad \mathbf{c}^e = [c_{N-1} \quad c_1 \quad c_2 \quad \dots \quad c_{N-2}]$$

A ciklikus kódokban mindig található $[g_0\ g_1 \dots g_{N-K}\ 0 \dots 0]$ alakú érvényes kódszó, ezért ezen kódok generátormátrixa mindig sávmátrix alakban is felírható:

$$\mathbf{G} = \begin{bmatrix} g_0 & g_1 & \cdots & g_{N-K} & 0 & \cdots & \cdots & 0 \\ 0 & g_0 & g_1 & & \ddots & & & \\ 0 & & g_0 & g_1 & & \ddots & & \vdots \\ \vdots & & & \ddots & \ddots & & \ddots & 0 \\ 0 & & \cdots & 0 & g_0 & g_1 & \cdots & g_{N-K} \end{bmatrix}$$

Példa egy egyszerű ciklikus kódra:

$\mathbf{c}_0 = [1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0]$
$\mathbf{c}_1 = [0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1]$
$\mathbf{c}_2 = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]$
$\mathbf{c}_3 = [1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]$

Látható, hogy a sávmátrix-szerkezet miatt a **c=mG** vektor komponensei az $(m_0, m_1, \dots, m_{K-1})$ és a $(g_0, g_1, \dots, g_{N-K})$ sorozatok konvolúciós szorzatösszegei. (Figyeljük meg, hogy a $(g_0, g_1, \dots, g_{N-K})$ sorozat fordított index-sorrendben függőlegesen is megjelenik a mátrix oszlopaiban!)

Ezt a tulajdonságot kihasználva bevezetjük az *x* bitpozíció-operátort, melynek kitevője a szimbólum pozícióját jelzi, és vektorok helyett *x* polinomjaival fogunk

[10] Érdekesség: a paritásellenőrzés még egyszerűbbé tehető $\mathbf{H}^T$ oszlopainak, **G** sorainak az átrendezésével, bár az eredmény nem szisztematikus.
Más: *Duális* kódokban (N, K) helyett (N, r) például a (7,4) Hamming-kód duálisában $d_{min}=4$.

dolgozni. (A kitevők a pozitív egész számok halmazából, az együtthatók GF[2]-ből származnak.) Mivel a polinomszorzás együtthatói a polinomok együtthatóinak, mint sorozatoknak a konvolúciós szorzatösszegei, ezért az *x*-szel végzett lineáris transzformáció után a mátrix-vektor szorzásból polinomszorzás lesz. Ez a művelet pedig, mint látni fogjuk, még a mátrix-vektor szorzásnál is egyszerűbben valósítható meg hardveresen.

$$\mathbf{c} = [c_0 \quad c_1 \quad c_2 \quad \dots \quad c_{N-1}] \xrightarrow{x} c(x) = c_0 + c_1 x + c_2 x^2 + \dots + c_{N-1} x^{N-1}$$

$$\mathbf{c} = \mathbf{mG} \xrightarrow{x} c(x) = m(x)g(x)$$

ahol $g(x) = g_0 + g_1 x + g_2 x^2 + \dots + g_{N-1} x^{N-1}$. A polinomok tagjait *x* kitevői szerint növekvő sorrendben írjuk.

Könnyen megmutatható, hogy a szokásos maradékos polinomosztással

$$x \cdot c(x) = c^e(x) \bmod (1 + x^N)$$

vagyis az *x*-szel, a bitpozíció-operátorral való szorzás elvárásainknak megfelelően ciklikus eltolásnak felel meg, de csak az $1 + x^N$ polinomra vett maradékát tekintve. Ezért bevezetjük a GF(2)[x]$|1 + x^N$ kommutatív, egységelemes gyűrűt, melynek elemei az N-nél kisebb fokszámú polinomok. A gyűrűben két műveletet definiálunk, a polinomiális összeadást és a szorzást, a szokásos módon. Ha a szorzás kivezetne a gyűrűből, akkor az eredményt maradékosan osztjuk az alap-polinommal ($1 + x^N$), és a maradékot tartjuk meg.

A továbbiakban ki fogjuk használni, hogy két polinom összegének a maradéka könnyen igazolható módon a maradékok összege.

5.1. Ciklikus kódok konstrukciója

Válasszuk meg g(x)-et úgy, hogy teljesüljön $h(x) \cdot g(x) = 1 + x^N$, és generáljuk az érvényes kódszavakat c(x) = m(x)g(x) szerint. Ekkor a polinomgyűrűben az érvényes kódszavakra valóban

$$s(x) = c(x) \cdot h(x) = m(x) \cdot g(x) \cdot h(x) = 0 \bmod x^N + 1$$

A generált ciklikus kódra hasonló gondolatmenettel igazak a paritásellenőrzési tételben foglalt állítások. Ezek a kódok közvetlenül megvalósíthatók az adó és a vevő oldalon is polinomszorzó áramkörök használatával, melyek egy léptetőregiszterből és a szorzó polinom együtthatóit tartalmazó kombinációs logikából állnak (bővebben lásd a konvolúciós kódoknál). A hibajavítás a javítható hibákra szindróma-táblázattal lehetséges.

Szisztematikus ciklikus kódok

A kód szisztematikussá tételéhez legyen most is $h(x) \cdot g(x) = x^N + 1$, de az érvényes kódszavakat az alábbi módon generáljuk:

$$c(x) = x^r m(x) + d(x), \text{ ahol}$$

$$d(x) = x^r m(x) \mid g(x)$$

A kód valóban szisztematikus: mivel deg(d(x)) < N-K, ezért a kódszó első K pozícióját csak $x^r m(x)$, utolsó N-K pozícióját pedig csak d(x) határozza meg. A vevőoldalon a szindrómát a g(x)-re adott maradék adja:

$$s(x) = c'(x) \mid g(x)$$

Ekkor az érvényes kódszavakra valóban

$$s(x) = (x^r m(x) + d(x)) \mid g(x) = x^r m(x) \mid g(x) + x^r m(x) \mid g(x) = 0\,,$$

mivel a polinomok együtthatói GF[2] elemei (és emiatt 1+1 = 0).

Az így előállított kód is ciklikus, mivel megfeleltethető egy polinomszorzással g(x)-ből képzett kódnak, ugyanis minden érvényes kódszó g(x) többszöröse. Ennek belátásához vezessük be a(x)-et a következő módon:

$$x^r m(x) = a(x)g(x) + x^r m(x) \mid g(x)\,,$$

ekkor a szisztematikus c(x) kódszó is felírható

$$c(x) = a(x)g(x) + x^r m(x) \mid g(x) + x^r m(x) \mid g(x) = a(x)g(x)$$

alakban, tehát valóban minden érvényes kódszó g(x) többszöröse.

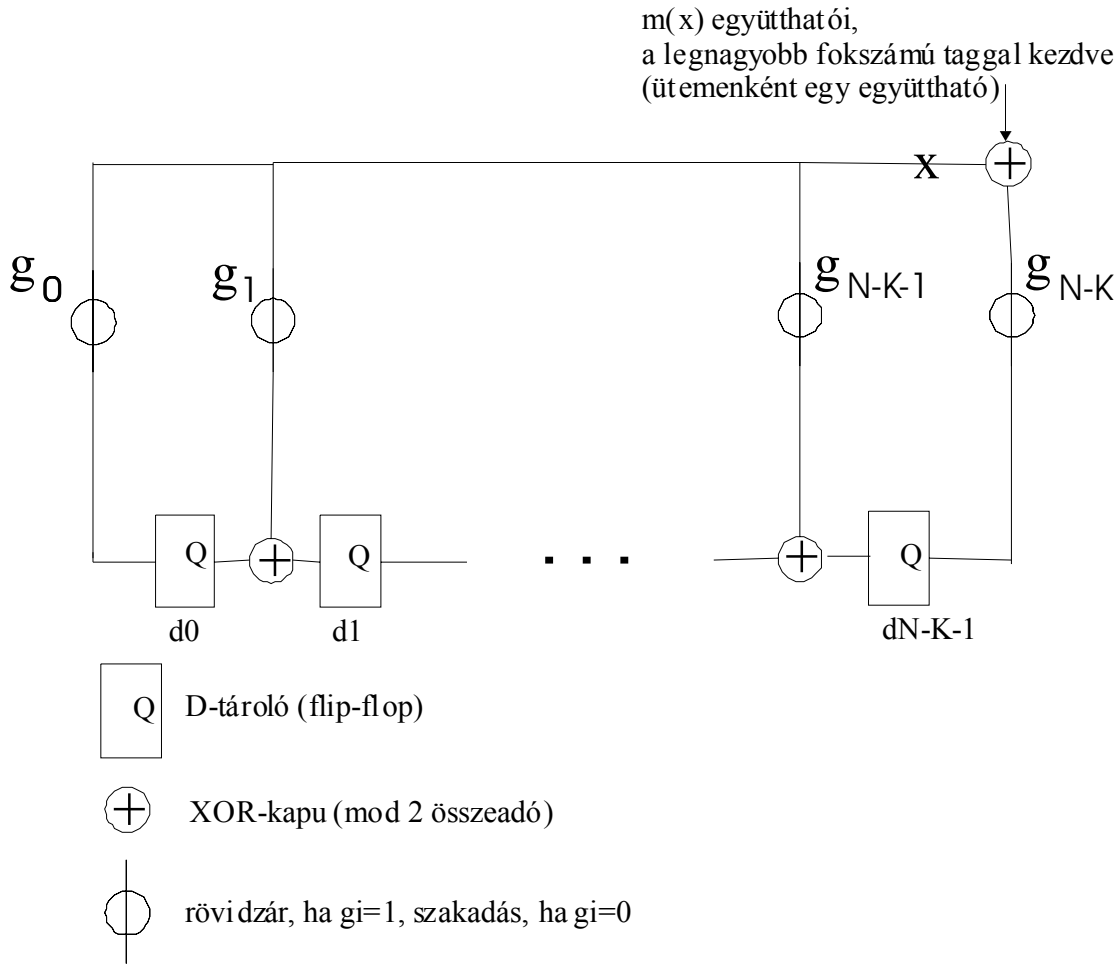
Az is megmutatható, hogy a $h(x) \cdot g(x) = x^N + 1$ feltétel nemcsak elégséges, de szükséges feltétele annak, hogy a g(x)-szel generált kód ciklikus legyen. Az is feltétel, hogy g(x)-ben a legmagasabb és a legalacsonyabb fokú tag együtthatója 1 legyen.

Szisztematikus ciklikus kódok megvalósítása polinomosztó áramkörrrel

Az eddigiek alapján a szisztematikus ciklikus kódok adóoldali generálásához és a vevőoldali szindróma képzéséhez ugyanaz a berendezés, egy g(x)-szel osztó áramkör elegendő. Az áramkör működésének az az alapja, hogy tetszőleges w(x)-re:

$$w(x) \mid g(x) = w_0 \mid g(x) + w_1 x \mid g(x) + w_2 x^2 \mid g(x) + \ldots + w_{N-1} x^{N-1} \mid g(x)\,,$$

azaz a maradékok tagonként számolhatók. Az adó-oldali polinomosztó áramkör szerkezetét a 4.1. ábra mutatja.



4.1. ábra: a polinomosztó áramkör szerkezete, jelmagyarázat az ábrán

Induláskor a D-tárolók tartalma 0. K ütem után az áramkör tárolóiban a g(x)-szel való osztás maradékának az együtthatói találhatók ($d_0 \ldots d_{N-K-1}$). A szisztematikus kód előállításához az osztó áramkört két üzemmódban használjuk:

1. Először K ütemen keresztül beléptetjük m(x) együtthatóit, miközben az együtthatókat a csatornára is továbbítjuk (a kód szisztematikus, ezért ez a rész már biztos.). K ütem után előáll a d(x) maradéka a tárolókban.
2. Megszüntetjük a visszacsatolást az áramkörön X-szel jelzett ponton, és a XOR-kapu kimenete helyett a visszacsatoló ágat 0-ra kötjük. Ezután N-K ütemen keresztül a csatornára küldjük a jobbszélső tároló kimenetét, azaz kiürítjük a léptetőregisztert. „Mellékhatásként" az áramkör alapállapotba kerül, készen az újabb kódszó előállítására.

A **dekóder** K ütemen keresztül végzi az osztást ugyanezzel az áramkörrel, majd a kapott szindrómát a szindrómatáblázat (ROM) címzésére használva meghatározza a becsült hibavektort, amit a kapott kódszóhoz hozzáad. A helyes ütemezéshez a bejövő kódszót N ütemig bufferelni kell egy léptetőregiszterben.

4.8. Példa: ciklikus kód előállítása és paritásellenőrzése N=7 esetén

az alappolinom felbontása: $1+x^7=(1+x)(1+x^2+x^3)(1+x+x^3)$

Legyen:

$g(x)=1+x^2+x^3$, *innen K=4, r=3*

$m(x)=1+x+x^3$

Ekkor

$c(x)=x^3m(x)+x^3m(x)\,|\,g(x)=x^3+x^4+x^6+x^2=[0\quad 0\quad 1\quad 1\quad 1\quad 0\quad 1]$

Ugyanezt az osztást a polinomosztóval elvégezve:

ütem	$d_0d_1d_2$	*„X" pont*	*Bemenet*
0	0 0 0	1	1
1	1 0 1	1	0
2	1 1 1	0	1
3	0 1 1	0	1
4	0 0 1	$=x^2$ *a maradék*	

Innen a kódszó: $c(x)=x^2+x^3+x^4+x^6$, *ezt ellenőrizve:*

ütem	$d_0d_1d_2$	*„X" pont*	*Bemenet*
0	0 0 0	1	1
1	1 0 1	1	0
2	1 1 1	0	1
3	0 1 1	0	1
4	0 0 1	0	1
5	0 0 0	0	0
6	0 0 0	0	0
7	0 0 0	*a maradék* 0 (érvényes kódszó)	

A kód 16 db érvényes kódszavát generálva látható, hogy

a) a kód valóban ciklikus
b) a kódtávolság 3, tehát 1 hibát tudunk javítani.

Hasonló módon kiszámolható, hogy például az 1 *súlyú* $e(x)=x^6$ *hibavektorral rontott* $c'(x)=x^2+x^3+x^4$ *hibás kódszó szindrómája* x^2

Ugyanezt a szindrómát kapjuk, ha az x^6 *maradékát számítjuk ki, mivel a szindróma valóban csak a hibától függ.*

Megjegyezzük, hogy a fenti kód egy (7,4) *Hamming-kód, mivel 1-hiba javító és perfekt. A Hamming kódok a ciklikus kódok alosztályát alkotják.*

4.9. Példa: a (K+1, K) paritáskód mint lineáris blokk-kód, és mint ciklikus kód

lineáris blokk-kód

A redundáns rész 1 *bit, mely az üzenetszó bináris digitjeinek* mod 2 *összege:*

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 1 & 0 & 1 & 0 & 0 & & \\ 1 & 0 & 0 & 1 & 0 & \cdots & 0 \\ \vdots & & & & 1 & & \\ & & & & & \ddots & \\ 1 & 0 & & & \cdots & 0 & 1 \end{bmatrix}$$

Megvalósítás: K bemenetű XOR-kapu, időben párhuzamos összeadás

ciklikus kód

az előbbi ***G*** *mátrix sorok összeadásával könnyen ciklikus alakra hozható:*

$$\mathbf{G}_{cilk} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 1 & 0 & 0 & & \\ 0 & 0 & 1 & 1 & 0 & \cdots & 0 \\ \vdots & & & 1 & 1 & & \\ & & & & \ddots & \ddots & \\ 0 & 0 & & & \cdots & 1 & 1 \end{bmatrix}$$

*Ez alapján g(x)=*1+*x. Valóban, ez a g(x) minden N-re osztója* $x^N + 1$*-nek.*

Megvalósítás: A polinomosztó a 4.1. ábra alapján mindössze egy XOR-kapuval visszacsatolt D-tárolóból áll. Ez időben sorosan (K ütem alatt, a részeredmény tárolásával) végzi el ugyanazt az összeadási műveletet, amit egy K bemenetű kapu tranziens idő alatt végez el.

Gyakorlatban használt ciklikus kódok

A ciklikus kódoknak csak bizonyos családjait lehet (itt nem részletezett módon) adott N, K, d_{min} paraméterekre tervezni. A legfontosabbak a következők:

- Hamming kódok
- BCH kódok. A javítható hibák számával nő a kódolás/dekódolás komplexitása. Egy példa: a 3551 polinom[11] által generált (31, 21) paraméterű kód 2 hibát tud javítani.

A CRC kódok

A CRC (Cyclic Redundancy Check) kódok csak hibajelzésre alkalmasak, tipikusan valamilyen hibajavító kódolással kombinálva alkalmazzák őket. A CRC kódok

[11] Ha g(x)-et ezen a módon, oktális (nyolcas számrendszerű) számmal adjuk meg, akkor ebből úgy kaphatók meg g(x) együtthatói, hogy minden egyes számjegyet átírunk kettes számrendszerbe. Ilyen módon minden számjegy 3 együtthatót kódol, például a 4 az 100-t. Bal oldalon a legmagasabb fokú tag együtthatója áll. Példa: $g(x) = 57 = x^5 + x^3 + x^2 + x + 1$

speciális ciklikus kódok, melyek nagyméretű (pl. K=1023) blokkhoz készítenek rövid (pl. r=24) ellenőrző összeget. Ha ezután akár a blokkban, akár az ellenőrző összegben változás történik, akkor az ismételten kiszámított CRC összeg nagy valószínűséggel nem fog egyezni a blokk végén található CRC összeggel. A CRC kódokat, illetve a szindrómát éppúgy polinomosztással lehet generálni, mint a többi ciklikus kódot. Az alkalmazás lépései:

1. A forráskódolás után, a hibajavító csatornakódolás előtt blokkonként a CRC generálása és a blokkhoz illesztése
2. Újbóli blokkokra osztás a hibajavító kód blokkmérete szerint, a hibajavító kód generálása
3. A vevőoldalon a paritásellenőrzés/javítás után CRC-blokkonként a CRC-összeg ellenőrzése. Ha a számított és a vett CRC összeg nem egyezik, akkor a blokkban hiba van, tehát a közben történt hibajavítás nem volt eredményes. Ekkor hibát jelzünk, és újra küldetjük/olvassuk a CRC-blokkot.

A CRC alkalmazása által viszonylag kis kódsebesség-romlás árán jó eséllyel detektálni tudjuk a vevőoldali hibás blokk-dekódolás eseteit (mikor a hibajavító kód sok hiba mellett rossz kódszóba javít, vagy a csatorna érvényes kódszóba ront), különösen, mivel a hibajavító kód blokkmérete jellemzően kisebb, mint a CRC blokkméret, tehát egy CRC-blokk több hibajavítókód-blokkot fog át.

A CRC alkalmazása sikertelen, ha a CRC-blokkban hiba történik, de nem kapunk CRC-hibát. Ez két esetben fordulhat elő:

1. Az ellenőrző összeg nem változott, de a blokk éppen egy másik olyan blokkba ment át, amelyik ugyanazt az összeget adja
2. Mind a blokk, mind az összeg változott, épp olyan módon, hogy a blokknak megfelel az összeg.

A hibajelző képesség ezért nemcsak a hibák számától, hanem azok helyétől (mintázatától) is függ. A CRC-kódok minőségi paraméterei közül a legfontosabbak:

- A **hibajelző képességet** a kódok egyedi analízisével lehet felderíteni.
- **Hibalefedés**: annak a valószínűsége, hogy egy véletlenszerűen választott CRC-blokk és egy véletlenszerűen választott ellenőrző összeg nem felel meg egymásnak, vagyis CRC-hibát ad. Mivel a CRC-hibát *nem* adó párosok száma 2^K, ezért a hibalefedés: $\lambda = 1 - \frac{1}{2^{N-K}}$, csak g(x) fokszámától függ.
- **Hibacsomó**-jelzés. A csatorna különböző fizikai folyamatai miatt (pl. folt a CD-n, rövid üzemzavar a hálózaton. stb) a hibák a gyakorlatban sok esetben nem véletlenszerűen eloszolva, hanem *hibacsomókban* (burst), egymáshoz közel jelentkeznek[12]. A CRC-kódokra jellemző, hogy ha b a hibacsomó hossza, akkor a kód jelez minden hibacsomót, ha $b \le N - K$, illetve a hibacsomók $1 - \frac{1}{2^{N-K}}$-ad részét, ha $b > N - K + 1$.

[12] A hibacsomót blokk-kódokra definiáljuk olyan módon, mint az N hosszúságú kódszó-blokkban az első és az utolsó hibás csatornaszimbólum pozíciója közötti különbséget, függetlenül attól, hogy a kettő között vannak-e más hibás szimbólumok. Ha például csak a legelső és a legutolsó szimbólum hibás, a hibacsomó hossza akkor is N.

5.2. Összefésülés és szétválogatás (interleaving)

Hardver redundancia alkalmazása
A hibacsomók elleni védekezés kézenfekvő módja, ha a kódolandó csatornaszimbólum-folyamot páros/páratlan időütemek szerint két folyamra bontjuk, és a két folyamon egymástól függetlenül (akár különböző kódolási eljárással, kódparaméterekkel), időben párhuzamosan végezzük el a csatornakódolást. A két kódoló kimenetét 2 ütemenként összefésüljük, úgy bocsátjuk a csatornára. A vevőoldalon megint szétválogatjuk a két folyamhoz tartozó csatornaszimbólumokat, és a két folyamot külön-külön a hozzá tartozó dekóderre vezetjük. Végül ezek kimenetét megint összefésüljük egy folyamba.

Az eljárás (2 utas interleaving) végrehajtásához ugyan két kódoló/dekódoló párra van szükség, azonban:

- Ezek fele akkora órajel-frekvenciával működhetnek, mint a csatorna, ami gyors csatorna és lassú kódolók (bonyolult algoritmus) esetén előnyös
- A csatornán előálló *b* hosszúságú hibacsomó az összefésülés-szétválogatás miatt úgy jelentkezik, mint két, *b*/2 hosszúságú hibacsomó a két folyamon. Ezért elég a kommunikációs rendszert fele akkora hibacsomókra tervezni.

Ugyanez az elv nemcsak kétszeres, hanem n-szeres hardver redundanciával is megvalósítható, ezzel a hibacsomók mérete n-edrészére csökkenthető.

Blokkos interleaving
A blokkos interleaving megoldás nem igényli a kódolók többszörözését. A csatornakóddal kódolt szimbólumfolyammal *oszloponként* feltöltünk egy D×D méretű táblázatot, majd mikor betelt, *soronként* olvassuk ki, úgy küldjük a csatornára. A vevőoldalon a dekódolás előtt egy hasonló táblázatot *soronként* töltünk fel, majd ha betelt, *oszloponként* olvasunk ki. Látható, hogy ha a csatornán hibacsomó keletkezik, akkor egy bizonyos méretig (D^2-D) a vevőoldalon a blokkos szétválogatás után ez D darabra törve jelentkezik, D különböző blokkban.

A módszer hátránya, hogy a táblázat betelésére D^2 ütemig várni kell, ezt a késleltetést bufferelléssel lehet kompenzálni.

Alkalmazás: zene digitális tárolása ciklikus kóddal
A zenei CD-k kódolására a várható hibacsomók (például karc) elleni védekezésként blokkos interleaving technikát alkalmaznak. A zenét 44 KHz-en mintavételezik, két csatornán, és minden mért értéket kvantálással 2 bájtos egész számmá alakítanak. Ilyen módon 4 bájt ír le egy mintát. A bájtokat *oszloponként* egy 24×24 bájtot tartalmazó táblázatba írják, amely tehát oszloponként 6, összesen 144 mintát tartalmaz. Ekkor hibajavító kóddal *soronként* 28 oszloposra bővítik a táblázatot, majd ugyanazzal a kóddal *oszloponként* 28 sorosra bővítik az immár 28 oszlopos táblázatot. Ezt a 28×28 bájtos táblázatot *soronként* írják a CD-re. Az alkalmazott kódolás egy (28, 24) paraméterű, 5 kódtávolságú Reed-Solomon kód GF[256] felett (mivel nem bináris kód, ezért ezt nem részletezzük). A módszer a hibacsomókat 28-adrészére töri.

A vevőoldalon a hibajavító kód alkalmazásával javítják az 1-hibákat. Bár a kód 2 hibát is tudna javítani, 2 vagy több hiba esetén a kódszót eldobják, és a környezet alapján következtetnek a hiányzó értékre.

5.3. Blokk-kódok hibaaránya (záró megjegyzések)

Az alábbi megállapítások minden hibajavító blokk-kódra érvényesek. Tegyük fel, hogy a rendszerünkben p hibavalószínűségű bináris szimmetrikus csatornát használunk, a kód t_{jav} számú hibát tud javítani, és javítjuk az összes javítható hibát! Ekkor annak a valószínűsége, hogy a vevőoldalon hibás blokk-dekódolás (rossz kódszóba javítás) történik:

$$P(\text{hbd}) = 1 - P(\text{sikeres javítás}) = 1 - \sum_{i=0}^{t_{jav}} \binom{N}{i} p^i (1-p)^{N-i} \xrightarrow[N>>i]{} \approx 1 - e^{-Np} \sum_{i=0}^{t_{jav}} \frac{(Np)^i}{i!}$$

A kérdéses valószínűség tehát közelíthető úgy, mint Np és t_{jav} egy függvénye. Ezek közül Np a blokkon belül várható hibák száma, t_{jav} a javítható hibák száma. Kérdés, hogy használható kommunikációs rendszerhez a javítható hibák számának mennyivel kell felülmúlni a hibák várható számát. Erre a kérdésre a fenti P(hbd) függvényt t_{jav}/Np függvényében vizsgálva kapunk választ. Ha előírjuk, hogy $P(\text{hbd}) < 10^{-10}$ legyen (ami szerény követelmény), akkor Np=2 esetén azt kapjuk, hogy körülbelül 8-szor ennyi hibát kell tudni javítani a kódunknak. Általában véve t_{jav}-nak nemcsak nagyobbnak kell lennie Np-nél, hanem Np sokszorosának kell lennie a rendszer használhatóságához.

A zajos csatornán való információtovábbítással kapcsolatban a 3.1. fejezetben megfogalmazott két fő célkitűzésünk egyrészt a csatornakapacitás jó kihasználása (nagy kódsebesség), másrészt a minél kisebb vevőoldali hibás blokk-dekódolási valószínűség biztosítása. Ezt a kettős célt a kódparaméterek és a kódolási módszer választásakor kétféleképpen próbálhatjuk elérni:

- N legyen kicsi, hogy az Np is kicsire adódjék (ne kelljen sok hibát javítani). Sajnos az ismert blokk-kódoknál csak nagy N értékekhez tartozik N-hez közeli K érték (például a 2 hibát javító a BCH kódoknál N=15-höz R=K/N=0.46, N=127-hez R=0.89 tartozik), tehát ilyen módon nehéz a nagy kódsebesség elérése.
- N legyen nagy, hogy K kellően kicsire választásával elegendő redundáns jegyünk legyen a kívánt számú hiba javítására. A kódsebesség (K/N) így is kicsire adódik.

Látható, hogy a két célkitűzést nehéz egyszerre elérni. A blokk-kódok mégis igen széles körben elterjedtek, elsősorban egyszerű generálásuk és paritásellenőrzésük miatt. Ezt túlszárnyaló hibajavítási teljesítményt a blokkokra osztást nem használó konvolúciós kódok tudnak nyújtani.

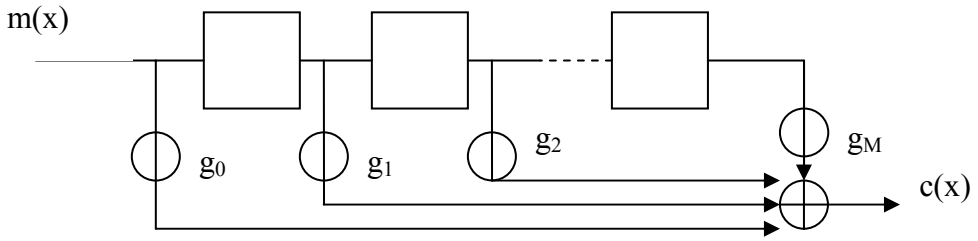
6. Konvolúciós kódok

A konvolúciós kódok jó hibajavítási tulajdonságokkal rendelkeznek, de a generált kód nem szisztematikus, és meglehetősen bonyolult a dekódolása. Tetszőlegesen hosszú bemeneti csatornaszimbólum-sorozathoz egyetlen kódszavat rendelnek hozzá (mint a forráskódolásnál az aritmetikai kód), tehát kimarad a szimbólumok blokkokra osztásának a lépése. Ezért ezen kódok leírásához használt fogalmak és módszerek csak részben feleltethetők meg a blokk-kódok hasonló fogalmainak, módszereinek.

6.1. Konvolúciós kódok generálása

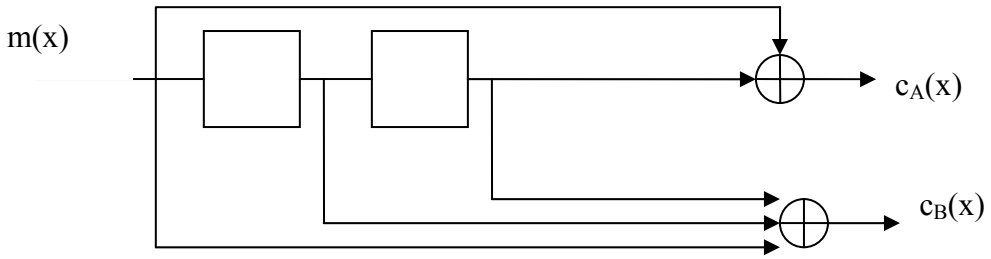
A konvolúciós kódokat a ciklikus kódok alapesetéhez hasonlóan a bemenetet polinomnak tekintve a kód generátor-polinomjaival való szorzással generálják, itt

azonban egyszerre több generátor-polinommal dolgozunk. A kódoló berendezés ennek megfelelően egy több kimenetű ún. FIR-szűrő (ablak-szűrő, konvolver), szerkezetét a 6.1. ábra mutatja.



6.1. ábra: Polinomszorzó áramkör vázlata, g_i-k a szorzó polinom együtthatói, M a léptetőregiszter mélysége (jelölések: mint a 4.1. ábrán)

A 6.1. ábrán látható áramkör a bemenetére ütemezetten érkező tetszőleges hosszúságú m(x) polinom és a g(x) polinom szorzatát állítja elő a c(x) kimeneten (ütemenként egy együtthatót). A konvolúciós kódoló ugyanazt a léptetőregisztert felhasználva egyszerre több különböző polinommal szorozza a bemenetet (szorzó polinomonként egy-egy sokbemenetű XOR-kapu és kimenet). A kódot tehát a szorzó polinomok határozzák meg.



6.2. ábra: Két kimenetű konvolúciós kódoló (jelölések: mint a 4.1. ábrán)

Az ábrán látható konvolúciós kód szorzó polinomjai:

$$g_A(x) = 1 + x^2$$

$$g_B(x) = 1 + x + x^2$$

Ütemenként egy csatornaszimbólum lép be a kódolóba, minden kimeneten egy szimbólum lép ki minden ütemben. Ezért, ha a kimenetek számát N_0-lal jelöljük, akkor a kód névleges kódsebessége:

$$R_{névl} = \frac{1}{N_0}$$

Ha a bemeneti m(x) csatornaszimbólum-sorozat nem végtelen, hanem K hosszú, akkor a valódi kódsebesség ennél kisebb, mert a léptetőregiszter kiürüléséig még M ütemig várni kell:

$$R_{eff} = \frac{K}{KN_0 + MN_0} = \frac{R_{névl}}{1 + \frac{M}{N_0}} < R_{névl}$$

A generált kódot vektoriális formában is felírhatjuk:

$$c_A(x) = m(x)g_A(x),\ c_B(x) = m(x)g_B(x), \text{ innen } \mathbf{c}(x) = m(x)\mathbf{g}(x)$$

A konvolúciós kódok fontos tulajdonsága a linearitás, melyhez szükséges, hogy két érvényes kódszó összege is érvényes legyen. A fentiek alapján ez teljesül:

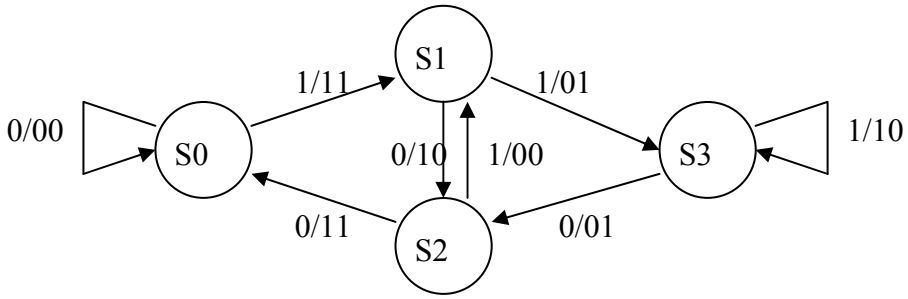
$$\mathbf{c}_1(x) = m_1(x)\mathbf{g}(x)$$
$$\mathbf{c}_2(x) = m_2(x)\mathbf{g}(x)$$
$$\mathbf{c}_1(x) + \mathbf{c}_2(x) = (m_1(x) + m_2(x))\mathbf{g}(x)$$

6.2. Az állapotgép-modell

A generált kód dekódolásához követnünk kell tudni, hogy a kódoló adott pillanatban éppen milyen belső állapotban van. Ez a kódoló **állapot-átmeneti gráfja** (sorrendi hálózat) segítségével lehetséges. Ehhez az egyes állapotokat a léptetőregiszter tartalma alapján azonosítjuk, például a 6.2. ábra szerinti kódra:

$$S0 \rightarrow 00$$
$$S1 \rightarrow 10$$
$$S2 \rightarrow 01$$
$$S3 \rightarrow 11$$

Az állapot-átmeneti gráf a bemenetek hatására bekövetkező következő állapotot, illetve a kimenet mutatja, például a fenti kódra:



6.3. ábra. A 6.2. ábra szerinti kódoló állapot-átmeneti gráfja (bemenet/(C_BC_A))

Az eddigiek alapján könnyen láthatók a konvolúciós kódok következő sajátosságai:

- a csupa 0 bemenetből csupa 0 kimenet lesz
- a bemeneti 0-k maximum M ütem után visszavisznek a 0…0 állapotba
- minden állapotba pontosan két másik állapot vezet, melyek LSB-je különböző
- minden állapotból két másik állapotba lehet eljutni, melyek MSB-je különböző

A konvolúciós kód egy **alternatív ösvényének** nevezünk egy tetszőleges hosszúságú, olyan állapotátmenet-sorozatot (illetve az ehhez tartozó kimenetet), amely a 0…0 állapotból indul, legalább egyszer elhagyja azt, és oda is tér vissza. Az összes lehetséges alternatív ösvény, hozzávéve a csupa 0 ösvényt is, kimenetei közötti páronként vett Hamming-távolságok minimuma, a d_f **szabad kódtávolság** a kód fontos jellemzője és a hibajavítás alapja. Mivel a fentiek szerint a kód lineáris, ezért a 4.2. fejezetben foglaltak alapján a szabad kódtávolság megállapításához elegendő megkeresni a legkisebb súlyú kimenetet generáló alternatív ösvényt, az ehhez tartozó kimenet súlya lesz a szabad kódtávolság. Ez gyakran az állapotátmeneti gráfról is könnyen leolvasható, például a 6.3. ábra gráfján ez az S0-S1-S2-S0 ösvény, ahonnan $d_f = 5$.

Az átviteli függvény

A konvolúciós kódok átviteli függvénye teljesen jellemzi a kód hibajavító tulajdonságait, és leírja az összes lehetséges alternatív ösvényt. Előállításához az állapotátmeneti gráf alapján a J: időindex, N: a bemenet súlya, és D: kimenet súlya operátorokkal formálisan felírjuk, hogyan juthatunk el az egyik állapotból a másikba. Egy több állapot-átmenetet tartalmazó ösvény hossza a J kitevőjéről, a hozzá tartozó kimenet súlya a D kitevőjéről olvasható le. Ahhoz, hogy az összes 0…0-ból induló és 0…0-ban végződő ösvényt fel tudjuk írni, a 0…0 (S0) állapotot két állapotra bontjuk: egy $S0_k$ kezdő- és egy $S0_v$ végállapotra. Ezek után a 6.3. ábra gráfjára a következőképpen írhatjuk fel az állapotátmenetek egyenleteit:

$$X_1 = JND^2 \cdot X_{0k} + JN \cdot X_2$$
$$X_2 = JD \cdot X_3 + JD \cdot X_1$$
$$X_3 = JND \cdot X_1 + JND \cdot X_3$$
$$X_{0v} = JD^2 \cdot X_2,$$

ahol X_i az i állapothoz vezető ösvényt jelöli. Az $S0_k$ állapotot úgy vehetjük kiindulópontnak, hogy $X_{0k}=J^0N^0D^0=1$-et helyettesítünk, és X_{0v}-re megoldjuk az egyenletrendszert. Az így kapott T(J,N,D) függvény a kód átviteli függvénye. A fenti példa esetén ez a következő alakra hozható:

$$T(J,N,D) = J^3ND^5 + J^3ND^5 \cdot \sum_{i=1}^{\infty} (JND(1+J))^i$$

A kifejezésben egy-egy tag egy ösvényt jelent, például a J^3ND^5 az S0-S1-S2-S0 ösvényt. Látható, hogy a legkisebb súlyú ösvény súlya, azaz D legkisebb kitevője valóban 5, és ennek az ösvénynek a hossza 3.

Konvolúciós kódoknál nem beszélhetünk hibás blokk-dekódolásról olyan értelemben, mint a lineáris blokk-kódoknál, hiszen nem blokkonként kódolunk. Ehelyett bevezetjük a P_b **bithiba-valószínűséget**, mint annak a valószínűségét, hogy a vevőoldalon, a (későbbiekben tárgyalt módon) dekódolt és javított szimbólumsorozatban egy szimbólum (bit) hibás. Megmutatható, hogy a bithiba-valószínűségre a T(J,N,D) függvény és a szabad kódtávolság alapján alsó és felső korlát adható, tehát az átviteli függvény valóban teljesen leírja a kód hibajavító képességét.

A jó hibajavító képességű konvolúciós kódok konstrukciójára kevés módszer ismeretes. A jó kódokat számítógéppel keresik. Néhány példa a következő táblázatban látható.

$R_{névl}$	M	Generátor-polinomok[13]	d_f
1/3	3	(13, 15, 17)	10
1/3	5	(47, 53, 75)	13
1/2	5	(53, 75)	8
1/4	4	(25, 27, 33, 37)	16

6.3. Dekódolás és hibajavítás egy lépésben: a Viterbi-algoritmus

Ez az algoritmus arra szolgál, hogy a vevőoldalon a vett, tetszőleges hosszúságú kódszóból meghatározzuk az m(x) üzenet becslőjét. Mivel a kód nem szisztematikus, ezért a feladat nem egyszerű.

[13] A ciklikus kódoknál használt oktális jelöléssel

Az algoritmus annyi különálló processzort használ, amennyi állapot van az állapotátmeneti gráfon (2^M). Ezeket a gráf állapotaihoz rendeljük, és a gráf szerint vannak összekötve, vagyis minden processzor két bemeneti és két kimeneti kapcsolattal rendelkezik (melyek természetesen mutathatnak saját magára is). Minden processzor megkapja minden ütemben a bemenetet, azaz az adott ütemhez tartozó N_0 darab csatornaszimbólumot.

Minden processzor rendelkezik egy egész szám tárolására alkalmas regiszterrel, melyben a j **jósági tényező** éppen aktuális értékét tárolja. Ezen kívül rendelkezik egy Ö ösvényregiszterrel, amely tetszőleges számú egybites bejegyzést, vagy üres értéket (0/1/X) tartalmazhat.

A dekóder folyamatos üzemben működik, azaz feltesszük, hogy a kapott kód végtelen hosszú, de induláskor a kódoló az S0 állapotban volt. Induláskor, azaz az első csatornaszimbólum-N_0-s vételekor az S0 processzorban j=0, a többiben j=∞. Az ösvényregiszter induláskor minden processzorban csupa üres (X) bejegyzést tartalmaz.

Ezután minden processzor minden ütemben a következő programot hajtja végre:

1. Elküldi a két hozzá kötött processzornak az ösvényregiszterét és a jósági tényezőjét, ezzel egyidejűleg fogadja a két hozzá vezető processzor ösvényregiszterét ($ö_1$ és $ö_2$) és a jósági tényezőjét (j_1 és j_2), és beolvassa az adott ütemhez tartozó N_0 darab csatornaszimbólumot (röviden: bemenetet).
2. A bemenet és az állapotátmeneti gráf alapján kiszámítja a két hozzá vezető útra vonatkozó b_1 és b_2 büntetést. A büntetés a gráf alapján az adott, hozzá vezető állapotátmenethez[14] tartozó kód és a megfigyelt bemenet közötti Hamming-távolság. A példánkban az S1-S2 átmenethez 10 kód tartozik, ha ehelyett 11-et figyelünk meg, akkor 1 a büntetés.
3. A kapott két jósági tényezőt megnöveli a hozzájuk számított büntetéssel, és kiválasztja a kisebbet. Ez kijelöli az elfogadott állapotátmenetet. Egyenlőség esetén véletlenszerűen választ a két átmenet között.
4. Saját jósági tényezőjét lecseréli az elfogadott (már büntetett) jósági tényezőre, az ösvényregiszterét pedig az elfogadott processzor által küldött ösvényregiszterre. Az ösvényregiszter végére (balról az első X helyett) beírja az elfogadott átmenethez az állapotátmeneti gráfon tartozó bemenetet (üzenetbitet). A példában az S1-S2 átmenethez 0 üzenetbit tartozik.

A dekódert működtető vezérlő minden ütem végén összehasonlítja a processzorok ösvényregisztereit. Ha balról nézve egy vagy több pozíción *minden processzorban* ugyanaz az üzenetbit található, akkor az egyező üzenetbiteket dekódoltnak és javítottnak tekinti és balra kilépteti az összes ösvényregiszterből.

Megmutatható, hogy az algoritmus a maximum likelihood elvet követi, azaz hogy a kapott, tetszőlegesen hosszú kódhoz megtalálja a hozzá legnagyobb valószínűséggel tartozó, lehetséges bemenetet. A jósági tényező számszerű értéke egy adott ütemben és egy adott processzorban nem más, mint a processzorhoz vezető, a kódolás elején az S0 állapotban indult ösvényhez tartozó, állapotátmeneti gráf szerint tartozó kód és a ténylegesen kapott, esetleg hibákat tartalmazó kód (a „bemenet") közötti összes távolság. Ilyen módon az illető ösvényre, mint a kódolóval történt eseménysorozatra vonatkozó, adott processzor által adott időben képviselt tippnek a *jóságát* fejezi ki (minél kisebb, annál jobb), innen a neve.

[14] Ne felejtsük, hogy minden processzor egy állapotot testesít meg!

6.4. Konvolúciós kódok továbbfejlesztései

Ebben a fejezetben a fent ismertetett alapalgoritmus problémáit, illetve ezek lehetséges megoldásait tárgyaljuk.

Az ösvényregiszter hossza
Az ösvényregiszter az algoritmus gyakorlati (VLSI) megvalósításában természetesen egy véges hosszúságú, pl. 32 bites regiszter, mérete fontos tervezési kérdés. Ha az ösvényregiszter teljesen betelt, de az ösvényregiszterek a legelső bitben sem egyeznek, akkor nem tudjuk a következő ütemben a végükre írni az üzenetbitre vonatkozó tippet. A folyamatos üzem miatt hibajelzésnek nincs értelme. Ezért ilyenkor kiválasztjuk a legkisebb jósági tényezőjű processzort, ennek ösvényregiszteréből levágjuk az első dekódolt bitet, és az összes regisztert balra toljuk egy pozícióval.

Túlcsordulás a jósági tényezőben
Elegendően hosszú, sok hibát tartalmazó kód feldolgozásakor a jósági tényezőt tartalmazó regiszterben aritmetikai túlcsordulás léphet fel. Ez kivédhető a legnagyobb (legrosszabb) jósági tényező ütemenkénti figyelésével. Ha a legnagyobb ábrázolható számot megközelíti, akkor mindegyik jósági tényezőből levonjuk a legkisebb jósági tényező értékét, az algoritmus helyes működése szempontjából úgyis csak azok *különbsége* számít. (Igaz, ezzel elveszik a jósági tényező számszerű értékének fent említett konkrét jelentése.)

A kommunikáció csökkentése (traceback módszer)
A megvalósításkor jelentkező probléma nagyobb mélységű kódoknál a nagyszámú processzor (2^M) közti kommunikáció, elsősorban az ösvényregiszterek ütemenkénti elküldése. Ez mellőzhető az ún. traceback (visszakövetéses) algoritmus-változatban. Ennek az alapja, hogy egy processzorba vezető két másik processzor állapotkódjának az LSB-je szükségképpen különböző. Ebben a változatban csak a jósági tényezőket küldjük el minden ütemben, az ösvényregiszterbe pedig az elfogadott processzor kódjának az LSB-je kerül minden ütemben. Az ösvényregiszter vizsgálatával így is visszakövethető az az ösvény, amely az adott ütemben az adott processzorhoz vezet. A processzorok egyetértését azonban már nem lehet az ösvényregiszterek elejének az összehasonlításával vizsgálni. Ehelyett attól a ponttól tekintünk dekódoltnak egy állapotátmenet-sorozatot, ahonnan az összes processzortól induló ösvény összefut. Innentől kezdve egész a regiszterek elejéig követjük ezt a közös ösvényt, és az állapotátmeneti gráf alapján meghatározzuk az egyes átmenetekhez tartozó üzenetbiteket. Ez kerül a dekóder kimenetére.

A döntetlen helyzetek kivédése
Két azonos jóságú tipp között a Viterbi-algoritmus véletlenszerűen választ, esetleg rosszul. Ez elkerülhető úgy, hogy a vevőoldalon a csatornáról a bináris szimbólumnak megfelelő mintavételezett jelet (ütemenként N_0 darab valós számot) kvantálás nélkül, közvetlenül vezetünk a dekóderbe. Ezekre természetesen nem lehet Hamming-távolságot számolni (az algoritmus 2. lépésében), ehelyett euklideszi távolságot számítunk. Ha például a csatornán +5V az ideális „1” és 0V az ideális „0”, és egy adott ütemben (1/2 kódsebességű kódnál) az állapotgráf szerinti kód (10), a vett értékek pedig 3.4V és 2.9V, akkor a büntetés:

$$b = (5 - 3.4)^2 + (0 - 2.9)^2$$

Természetesen ekkor a jósági tényező sem egész, hanem valós szám. Ezt a módszert a kvantálás elhagyása miatt **lágy dekódolású** változatnak nevezik. A módszer nemcsak a

döntetlenek lehetőségét szünteti meg (két valós érték gyakorlatilag sohasem egyezik), hanem a kvantálási információveszteség kiküszöbölése miatt bizonyíthatóan lényegesen (akár nagyságrendekkel) kisebb bithiba-valószínűséget tesz lehetővé.

A kódsebesség növelése

Az eddig tárgyalt kódok névleges kódsebessége maximum ½ volt. Kevésbé zajos csatornán természetesen igény van nagyobb kódsebességű kódokra is, konvolúciós kódoknál ezt a kód **kilyukasztásával** érhetjük el. A kilyukasztott kódok generálása az alapváltozattal azonos módon történik, de az ütemekből periódusokat képezünk, és a csatornára periódusonként csak a generált kód egy részét engedjük. Ilyen módon nagy kódsebességek is elérhetők. Az eljárást a korábbi (7,5) kóddal, illetve kilyukasztott változatával, 3 ütemet átfogó periódussal szemléltetjük. A periódus második ütemében csak a 7, harmadikban pedig csak az 5 polinom kimenetét engedjük tovább.

Idő (óraütemek)	1	2	3	4	5	6
Periódusok	←	1.per.	→	←	2.per.	→
Üzenetbitek	1	1	0	1	0	0
(7,5) alapkód, R=1/2	11	01	01	00	10	11
(7,5),7,5 kilyukasztott kód, R=3/4	11	0	1	00	1	0

A kilyukasztott kódok állapotgráfja P-szer annyi állapotot tartalmaz, mint az alapváltozat, ha P a periódus hossza, de a dekóderbe elegendő 2^M darab processzor. Ezek programja azonban bonyolultabb, mivel a processzornak nyilván kell tartania, hol tart a periódusban, és eszerint kell számítania a büntetéseket.

A kilyukasztással növelhető a kódsebesség, azonban romlik a kód hibajavító képessége. Néhány kilyukasztott kódra mutat példát az alábbi táblázat (érdemes összevetni a nem kilyukasztott kódok paramétereivel).

$R_{névl}$	M	Generátor-polinomok[15]	d_f
2/3	3	(13, 15), 15	4
3/4	5	(61, 53), 53, 53	5
4/5	5	(61, 53), 47, 47, 53	4
5/6	4	(37, 23), 23, 23, 25, 25	4

[15] A ciklikus kódoknál is használt oktális jelöléssel

III. KRIPTOGRÁFIA

7. A kriptográfia alapkérdései

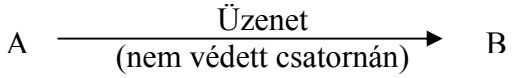
A kriptográfia régebbi tudomány, mint az információelmélet, hiszen az alkalmazásai az ókorba nyúlnak vissza. A történet lebilincselő áttekintése megtalálható az irodalomban (Singh, 2001). azonban a számos korábbi gyakorlati eredményt, módszert rendszerező elmélet csak a XX. században született meg, és jórészt C.E. Shannon érdeme.

7.1. Adatbiztonság és lehetséges támadások

Az adatbiztonság (információs rendszerek biztonsága) nagyon tág problémakör, felöleli a védendő információt tartalmazó és közvetítő berendezések fizikai védelmét, a vállalat szervezeti felépítését és emberi erőforrásait, és az alkalmazott számítógépes információs rendszerek, kriptorendszerek biztonságát. Bármelyik területen is van hiányosság, az adatbiztonság veszélybe kerül. Ezen problémakörök közül a kriptorendszerek minőségi jellemzőivel, lehetséges megoldásaival foglalkozunk. A valóságban használt kriptorendszerek tervezésének alapelvei:

- a rendszer feltörésének a költsége legyen nagyobb, mint az információ aktuális maximális[16] piaci értéke, vagy
- a feltöréshez a jelenlegi eszközökkel szükséges idő alatt a titok évüljön el (veszítse el az értékét).

Zárt rendszerek védelme általában megfelelő fizikai, szervezeti és humán tervezéssel megoldható. A kriptográfiát érdeklő probléma az, hogy a külvilággal kommunikáló, nyílt rendszert hogyan lehet a rosszindulatú behatolás ellen megvédeni, illetve még inkább, hogy az ellenséges (nem védett) csatornán keresztül hogyan tud két információs rendszer biztonságosan kommunikálni. Jó tervezés esetén a szervezeti, fizikai és informatikai rendszer-határ egybeesik. A kriptorendszerek alapsémája hasonló az információtovábbítás eddig használt modelljéhez:



Az üzenetet a csatornán titkosított (rejtett) formában továbbítjuk. Az üzenetet eredeti formájában **nyílt szövegnek** nevezzük (jele X, plaintext), titkosítva **rejtett szövegnek** (jele Y, ciphertext). A rejtett szöveget a nyíltból a rejtés (encryption, E) révén kapjuk meg, melynek a nyílt szövegen kívül egy K **kulcs** is a paramétere. A vevőoldalon a fejtés (decryption, D) állítja vissza ugyanazon kulcs és a rejtett szöveg alapján a nyílt szöveget. Összegezve:

$$Y = E_K(X)$$
$$X = D_K(Y)$$

[16] „maximális” alatt azt értjük, amennyit az információ a számunkra legkellemetlenebb támadónak ér. Háborúban ez az ellenfél, az üzleti világban a konkurencia, más esetben egy napilap, vagy az adóhatóság…

Általában (a történelmi tapasztalatok tükrében is) feltehető, hogy a támadó fél (C) ismeri az üzenet nyelvét és a titkosítás módját (E és D algoritmusát), de nem ismeri az éppen használt K kulcsot, ezért nem tudja kitalálni X-et. Az ilyen alapsémájú kriptorendszereket **titkos kulcsú** rendszereknek nevezzük.

A kriptorendszert támadó külső fél (C) a következő alapvető módszereket használhatja:

1. Hallgatózás, vagy „rejtett szöveg" típusú támadás. A támadó a csatorna forgalmának a figyelésével, az üzenet nyelvének, esetleg témájának az ismeretében próbálja megfejteni az aktuális üzenetet vagy kitalálni a K kulcsot.
2. „Nyílt szöveg" típusú támadás. A támadó ismeri egy korábbi üzenet nyílt és rejtett formáját is, ebből próbálja kitalálni a kulcsot, melyet későbbi üzenetek megfejtésére használhat.
3. „Választott nyílt szöveg" típusú támadás. C valahogyan rá tudta venni A-t, hogy egy általa választott nyílt szöveget rejtsen, és küldjön el B-nek. A nyílt és rejtett üzenet-párból könnyebb következtetni a kulcsra.
4. „Választott rejtett szöveg" típusú támadás. C valahogyan rá tudta venni A-t (pl. ideiglenesen hozzáfért a fejtő berendezéshez), hogy egy általa konstruált rejtett szöveghez a nyílt szöveget állítsa elő K használatával, és ezt a nyílt szöveget is megszerezte. A rejtett szöveg ügyes konstrukciójával a nyílt és rejtett üzenet-párból könnyebb következtetni a kulcsra.

Ezek a támadások alapvetően a kulcs megszerzésére irányulnak. A támadó azonban akkor is jelentős kárt tud okozni, ha manipulálni tudja az A-t B-vel összekötő hálózati elemeket:

1. Közbeékelődés („man in the middle" támadás). A valójában nem B-vel, hanem C-vel van összeköttetésben., B valójában nem A-tól, hanem C-től kapja az üzenetet, de ezt A és B nem veszi észre. Nyilvános kulcsú rendszerekben, egy hosszú üzenetváltás-sorozat során C hamis identitással mindvégig A és B közé ékelődik, minden üzenetet megfejt, és a megfelelő pillanatban leadja a számára előnyös üzenetet. Hasonló mesterkedés okozta a skótok királynője, Queen Mary vesztét 1586-ban.
2. Átírás (forging). Az üzenet C-n keresztül halad, aki elfogja azt, és bár megfejteni nem tudja, egyes részeit megváltoztatja. Például korábban megfigyelt rejtett-nyílt üzenetpárok alapján egy korábbi rejtett üzenetből egy ismert értelmű darabot betesz egy rejtett üzenet megfelelő darabja helyére („cut-and-paste" támadás). Esetleg egy korábban megfigyelt ismert tartalmú teljes üzenetet küld el („replay attack"), ami C-nek előnyös reakciót vált ki B-ből.

Információelméleti szempontból az alapprobléma a 1. támadás, a hallgatózás. Lehet-e, és ha igen, hogyan, olyan „tökéletes" kriptorendszert készíteni, amelyben a kulcs ismerete nélkül lehetetlen a nyílt szöveg meghatározása a rejtett szöveg megfigyelésével? Ezt a kérdést vizsgálta Shannon 1949-ben.

7.2. A tökéletes titkosítás és megvalósítása

Tökéletesnek nevezzük azt a titkosítási rendszert, amelyben a rejtett szöveg átlagosan semmi információt nem árul el a nyílt szöveggel kapcsolatban, vagyis amelyre

$$I(X,Y) = 0$$

Ha most alkalmazzuk a kölcsönös információ 1.3. alatti definícióit, akkor azt kapjuk, hogy ez akkor és csak akkor teljesül, ha

$$H(X \mid Y) = H(X) \text{ és } H(Y \mid X) = H(Y), \text{ azaz ha}$$
$$p_{i|j} = p_i \text{ és } p_{j|i} = p_j \ \forall i, j - \text{re}$$

Eszerint egy adott rejtett szöveghez ugyanakkora eséllyel kell tartoznia bármelyik nyílt szövegnek (és fordítva). Mivel a kettőt a kulcs választása rendeli össze, és a lehetséges rejtett szövegek számának legalább akkorának kell lennie, mint a lehetséges nyílt szövegek számának, ezért a kölcsönös információ hiányára vonatkozó elvárásunkat csak úgy tudjuk teljesíteni, ha

- $|X| = |Y| = |K|$, tehát a lehetséges kulcsok, nyílt és rejtett szövegek száma egyenlő, és
- $P(K_i) = \frac{1}{|K|}$, tehát a kulcsok közül minden egyes üzenet rejtésekor véletlenszerűen választunk.

Ezek nehezen teljesíthető elvárásoknak tűnnek, azonban az úgynevezett **one-time pad** (eldobó kulcsú titkosítás) esetén teljesülnek. E módszer szerint a nyílt szöveget bitenként egy kétbemenetű XOR kapura vezetjük, a másik bemenetre pedig egy, a nyílt szöveg hosszával egyező hosszúságú véletlen bitsorozatot vezetünk (ez a kulcs). A rejtett szöveg a kapu kimenete. A vevőoldalon ugyanezt a műveletet hajtjuk végre a rejtett szöveggel és ugyanazzal a kulccsal, így visszakapjuk a nyílt szöveget. Látható, hogy a fenti feltételek teljesülnek a sorozatok egyenlő hosszúsága miatt.

Azt, hogy ekkor nincs a nyílt és rejtett szöveg között kölcsönös információ, a fenti tételtől függetlenül is könnyű belátni. Annak a valószínűsége ugyanis, hogy a rejtett szöveg egy bitje például „0”,

$$P(Y=0) = P(K=0)P(X=0) + P(K=1)P(X=1) = \frac{1}{2}(P(X=0) + P(X=1)) = \frac{1}{2},$$

függetlenül a nyílt szövegtől. Így a kriptorendszer 3.1. fejezetben leírt 0 átvitelű csatorna egy speciális megvalósításának fogható fel.

Fontos megjegyezni, hogy mihelyt egy már használt kulcsot még egyszer használunk egy másik szöveg rejtésére, tökéletes titkosításunk igen sebezhetővé válik a nyílt szöveg típusú támadással szemben, mivel a kulcs meghatározása egy rejtett-nyílt üzenetpárból triviális. Minden egyes üzenethez egy, a rejtett szöveg méretével egyező méretű véletlen kulcsot kell generálni, és azt biztonságosan eljuttatni a vevőhöz a kommunikáció kezdete előtt. Ez a gyakorlati probléma igen erősen korlátozza a módszer használhatóságát. Ezért csak különleges esetekben alkalmazzák, mint például az amerikai és orosz elnököt összekötő telefonvonal, a „forró drót”.

Általában véve, a titkos kulcsú kriptorendszerek egyik nagy problémája a titkos kulcsok biztonságos eljuttatása az összes résztvevőhöz, amit **kulcskezelési problémának** (key management problem) neveznek.

A gyakorlatban használt kriptorendszerek nem tökéletesek, hanem a 7.1. fejezet elején ismertetett alapelvek szerint tervezik őket. A kriptográfiában az elv alkalmazását a **számítási teljesítményre alapozott biztonságnak** (computational secrecy) nevezik.

7.3. Néhány egyszerű kriptorendszer

A titkos kulcsú kriptorendszerek nagy része néhány egyszerű módszer kombinációjára alapul. Az alábbiakban feltesszük, hogy a nyílt szöveg magyar nyelvű és a teljes 44

betűs magyar ABC-t használja. Az egyes betűkhöz az ABC-ben elfoglalt helyzetük alapján 0…43 közötti számokat rendelünk hozzá.

Eltolás (Ceasar-módszer)
A kulcs egy betű. A nyílt szöveg minden betűjéhez mod 44 hozzáadjuk a kulcsot. Ekkor |K|=44.

S-doboz (helyettesítés)
A kulcs egy olyan táblázat, amely minden betűhöz egy másik betűt (vagy saját magát) rendel hozzá. A rejtés során a nyílt szöveg összes betűjét lecseréljük a táblázat szerinti betűre. A lehetséges kulcsok száma: |K|=44!

A fenti két módszert, mivel alapvetően egy kritpto-ABC-t használnak, monoalfabetikus módszereknek is hívják.

A következő két módszer közös jellemzője, hogy a nyílt szöveget rejtés előtt blokkokra osztjuk, és a rejtést blokkonként, ugyanazzal a kulccsal végezzük el.

Eltolás kulcsszóval
A kulcs egy szó, melynek hossza L. A nyílt szöveget L hosszúságú blokkokra osztjuk. Minden blokkban az első betűt a kulcs első betűjével toljuk el (lásd fent), a másodikat a másodikkal, stb. A lehetséges kulcsok száma: $|K|=44^L$.

P-doboz (permutáció)
A kulcs egy L×L-es permutáló mátrix, melynek minden sorában és oszlopában pontosan 1 darab egyes van, a többi eleme 0, amellyel a nyílt szöveg egy L hosszúságú blokkját, mint vektort szorozzuk. A lehetséges kulcsok száma: |K|=L!

8. A nyelvi entrópiára alapozott támadás

A fenti egyszerű kriptorendszereket az üzenet nyelvének az ismeretében könnyen fel lehet törni a lehetséges kulcsok próbálgatásával, hiszen a nyílt szövegnek értelmesnek kell lennie az adott nyelven.

Ha a nyílt és a rejtett szöveg, vagy kulcs és a rejtett szöveg egyes betűi között közvetlen kapcsolat van, mint például az S-doboz vagy az eltolás esetén, akkor **betűgyakoriság-statisztikák** alapján a támadó könnyen le tudja szűkíteni a valószínű kulcsok körét. Sőt, nem is kell az összes valószínű kulcsot végigpróbálni, hiszen például a kulcsszavas eltolás esetén a kulcs darabjait külön-külön is ki lehet találni. Ha például a levél első szavából már megvan annyi, hogy „ked*es”, akkor a hiányzó betű valószínűleg v, ez alapján pedig a kulcs negyedik betűje próbák nélkül is meghatározható. Ezekben a támadásokban a természetes nyelvek „beépített” redundanciáját lehet kihasználni.

8.1. A nyelvi entrópia és redundancia

A természetes nyelvek nem tekinthetők emlékezet nélküli forrásnak, amennyiben egy forrásszimbólumnak egy betűt tekintünk, mivel egy-egy betű (és szó) előfordulási valószínűsége erősen függ a környezetétől, tehát a szöveg távolról sem véletlenszerű betűhalmaz. Ezt a jelenséget számszerűen a nyelvi entrópiával, vagyis a végtelen mértékben kiterjesztett forrás esetén az egy betűre jutó entrópiával lehet kifejezni:

$$H_L = \lim_{n\to\infty} \frac{H(A_1, A_2, \ldots A_n)}{n}$$

ahol $H(A_1, A_2, \ldots A_n)$ az n-szeresen kiterjesztett forrás, amelyben tehát egy betű-n-est tekintünk egy forrásszimbólumnak. A természetes nyelvekre a szókészlet korlátozottsága miatt $H(A_1, A_2, \ldots A_n) < n \cdot H(A)$. Például az angol nyelvre statisztikai vizsgálatokkal megállapított értékek:

n	$\frac{H(A_1, A_2, \ldots A_n)}{n}$
1	4 bit
2	3.56 bit
3	3.30 bit

A tényleges nyelvi entrópiát angolra 1.5 és 1 bit közé lehet tenni—az egy angol betű által ténylegesen hordozott átlagos információ tehát lényegesen kevesebb, mint a betűnkénti feldolgozás alapján várnánk. A nyelvi entrópia alapján definiálhatjuk a nyelvi redundanciát:

$$R_L = 1 - \frac{H_L}{\log |A|},$$

amely az angol nyelvre 0.75-re adódik, vagyis a szöveg körülbelül 75%-a redundáns. Ezt a redundanciát lehet a támadás során kihasználni olyan módon, hogy egyszerre **sok** lehetséges kulcsot zárunk ki egyetlen próba során egy kulcsrészlettel megfejtett üzenetrészlet képtelensége alapján. Például biztos, hogy az üzenet nem kezdődik három Q betűvel. Megmutatható, hogy a nyelvi redundancia miatt a lehetséges kulcsok száma 0-ra csökken, vagyis a titkosítás statisztikai értelemben fel van törve, ha elegendően hosszú rejtett szöveget van alkalmunk megfigyelni. A feltöréshez szükséges betűk száma:

$$n_0 = \frac{\log |K|}{R_L \log |A|},$$

amely a fenti egyszerű módszerek esetén meglepően alacsony érték[17]. A nyelvi redundanciára alapozott támadásokat természetesen ki lehetne védeni forráskiterjesztéssel, például n>10 esetén már eléggé megközelíthető a nyelvi entrópia. A kiterjesztéssel azonban a forrás-ABC is kezelhetetlenül nagyra nő. Hasonló okokból nem jó megoldás az sem, hogy a nyelv szavait, esetleg szócsoportjait tekintsük forrásszimbólumoknak.

8.2. Kriptorendszerek konstrukciója

Shannon javaslata szerint nem tökéletes, titkos kulcsú kriptorendszerek esetén az egyszerű módszereket több lépésben úgy kell kombinálni, hogy minél inkább megszűnjön a nyílt szöveg, a rejtett szöveg és a kulcs egyes betűi közötti közvetlen kapcsolat, és a kulcs egyes részeit ne lehessen egymástól függetlenül kipróbálni és kitalálni. A kulcsok közül véletlenszerűen választunk, a lehetséges kulcsok száma pedig olyan nagy legyen, hogy az összes kulcsot nem lehessen végigpróbálni. Shannon

[17] angol nyelvre a kulcsszavas eltolás esetén, L=5 esetén $n_0 = 7$.

konkrétan a következő megoldásokat ajánlja nyílt szöveg, a rejtett szöveg és a kulcs részei közötti kapcsolat elmosására:

- **Diffúzió**: egy rejtési lépés előtt a szöveg kezdetétől indulva annak minden betűjét helyettesítjük a környezetében található betűk mod 44-es összegével. A szöveg végétől indulva a művelet könnyen invertálható. Hatása az, hogy „elmossa" a szöveg statisztikai szerkezetét, elegendően nagy környezet felhasználása esetén használhatatlanná teszi a betűgyakoriság-statisztikákat.
- **Konfúzió:** a szöveg blokkjain nemlineáris transzformációt, jellemzően helyettesítést hajtunk végre, melyhez a helyettesítési táblázatot egy fix (támadó számára is ismert) táblázat-készletből a kulcs és a szöveg egyes darabjaitól függő módon választjuk. Ezáltal meggátoljuk, hogy a kulcs egyes részeit egymástól függetlenül meg lehessen határozni.

A legelőnyösebb, ha a diffúziót és a konfúziót titkos kulcsú módszerrel együtt (pl. one-time pad) iteratív módon használjuk. Az iteráció minden lépésében diffúziós és konfúziós lépésekkel vesszük körül a titkos kulcsú lépést, melyhez a kulcsot folyamatosan változtatjuk.

Ezen módszerekkel el lehet érni, hogy a támadónak gyakorlatilag az összes lehetséges kulcsot végig kelljen próbálnia a kriptorendszer feltöréséhez, mert ha csak egyetlen bit is hibás a kipróbált kulcsban, a szöveg tökéletesen értelmetlen lesz. Az összes kulcs végigpróbálását nevezzük a **nyers erő** (brute force) módszerének. Ha pedig a lehetséges kulcsok száma emellett elegendően nagy, akkor a rendszer biztonságos. Mivel a számítógépek sebességének jövőbeli növekedését nehéz megjósolni, ezért az elegendően nagy kulcsszám megbecsléséhez egyre inkább a termodinamikai korlátokból indulnak ki. Ha egy kulcs kipróbálásához akár egyetlen 0/1 átmenet elég is egy szuperszámítógépen, akkor is szükséges ehhez az átmenethez energia. Ez alapján az energia kvantált természete miatt alsó korlátot lehet adni a nyers erő alkalmazásának az energiaszükségletére. Ha például a kulcs 192 bites (a lehetséges kulcsok száma 2^{192}), akkor a feltöréshez szükséges energia nagyobb, mint a Nap által egy év alatt kisugárzott összes energia.

8.3. A Data Encryption Standard (DES)

A DES klasszikus titkos kulcsú kriptorendszer, amerikai szabvány, amelyet jelenleg is igen széles körben alkalmaznak. Titkos kulcsa 56 bites, melyet a nyílt szöveg 64 bites (bináris) blokkjaira blokkonként alkalmazva kapjuk a rejtett szöveg 64 bites blokkjait. Alapelve a konfúzióval, diffúzióval és permutációval kombinált one-time pad módszer 16 iterációs ciklusban, melyhez a kulcsokat az alapkulcs alapján permutációkkal minden ciklusban külön készítik el. A fejtés ugyanazzal a kulccsal és berendezéssel történik; a kulcs másodszori alkalmazása megszünteti a kulcs hatását.

A számítástechnika fejlődésével az eredeti DES (melyet 1992-ben még 20 millió dollár költséggel 6 óra alatt lehetett feltörni) egyre inkább támadhatóvá vált a nyers erő módszerével. Ezért jelenleg 3DES néven úgy alkalmazzák, hogy a rejtés 3 egymás utáni DES lépésben történik, 3 különböző kulccsal, vagy az első és a harmadik lépésben ugyanazzal a kulccsal. Ezáltal a kulcsméret 56 bitről 168, illetve 112 bitre nő, ami már elegendő védelmet jelent.

8.4. Blokkos kriptorendszerek láncolása

A blokkos kriptorendszerek egyik gyengesége, kulcsmérettől függetlenül, hogy a nyílt szöveg ismétlődő blokkjai a rejtett szövegben is azonos blokkokat eredményeznek. Ismétlődő blokkok, szövegmintázatok pedig gyakran fordulnak elő, különösen a szoftverrendszerek közötti automatikus elektronikus kommunikációban. A nyílt szöveg/átírás típusú támadás során a támadó lecserélhet egyes rejtettszöveg-blokkokat korábbi üzenetekből ismert tartalmú rejtettszöveg-blokkokra („**cut and paste**" módszer). Ha a titkos kulcsot nem változtatták közben, akkor a vevőoldalon nem lehet a cserét észrevenni.

A támadás elleni védekezés alapja a kriptorendszer olyan módon való használata, hogy az egyes blokkok rejtését a korábbi rejtettszöveg-blokkoktól tesszük függővé. A DES úgynevezett CBC (Cipher Block Chaining, rejtettblokk-láncolás) üzemmódjában minden nyíltszöveg-blokkot az előző rejtettszöveg-blokkal kombinálnak (bitenkénti XOR művelettel), az ennek eredményeképpen előálló blokkot rejtik a DES algoritmusával és a titkos kulccsal. Az első blokk rejtéséhez egy véletlenszerűen generált blokkot, az úgynevezett inicializáló vektort használnak fel, amelyet nyíltszövegben küldenek el. A módszer előnye, hogy a nyíltszöveg mintái nem eredményeznek mintákat rejtett szövegben.

8.5. Titkos kulcsú kriptorendszerek megvalósítási kérdései

A titkos kulcsú rendszerek két nagy problémája a kulcskezelési probléma és a titkos kulcsok előállítása. Ezekre automatizált és gyors megoldást kell találni, mert a kulcsokat (nemcsak informatikai meggondolásokból) alkalmazási területtől függően minél gyakrabban cserélni kell.

A titkos kulcsoknak valóban véletlen bitsorozatot kell tartalmazniuk, ezen kívül az is fontos, hogy az eddig már felhasznált kulcsok sorozata alapján ne lehessen következtetni a következő kulcsra. A gyakorlatban használt megoldás egy álvéletlen generátor, melyet egy valóban véletlen maggal (seed) inicializálnak. A generátor lehet maga a DES, melynek egyik bemenete a mag, a másik pedig az előbb generált kimenet. A kimenetet blokkokra osztva kapjuk a kulcsok sorozatát. A mag forrásai különbözőek lehetnek:

- Fizikai folyamat mérése, pl. naptevékenység, kozmikus fehérzaj
- Felhasználói interakció, pl. a billentyű-leütések közti idő
- A számítógép aktuális rendszerparamétereinek a kombinációja, mint pl. a fájlok, bufferek, folyamatok száma, mérete, stb.

A gyakori kulcscsere egy lehetséges megoldása, hogy a kommunikáló felek csak egy fő-kulcsot juttatnak el védett módon egymáshoz, ezek után minden munkamenethez (session) az egyik fél álvéletlen generátorral helyben generálja a titkos kulcsot, majd azt a fő-kulcs (key encrypting key, KEK) segítségével DES-sel rejti, és eljuttatja a másik félhez, aki azt szintén a KEK segítségével megfejti. Ezután az új munkamenet-kulcsot használják.

9. Nyilvános kulcsú kriptorendszerek

Az internet térhódításával szükségessé vált olyan módszerek kifejlesztése, melyek nem igénylik titkos kulcsok védett leosztását, mivel a kommunikációban résztvevők köre előre nem ismert (pl. internetes boltok, szolgáltatások). A nyilvános kulcsú rendszerek

olyan módon kerülik meg a kulcskezelési problémát, hogy a kulcs titkos részét olyan formában (úgynevezett **egyirányú művelet** alkalmazásával) hozzák nyilvánosságra, hogy abból a kulcsra a jelenleg ismert módszerekkel csak nagyon nagy számítási teljesítmény felhasználásával lehet visszakövetkeztetni. Megmutatható, hogy bármilyen nyilvános kulcsú kriptorendszerhez szükség van egyirányú művelet alkalmazására.

9.1. A Diffie-Hellman módszer

Az eljárás pontosan két partner számára teszi lehetővé egy közös titkos kulcs létrehozását nyílt hálózaton. Paraméterei az a alap (kis egész szám), és az n nagy prímszám, melyeket a két résztvevő (A és B) nyilvánosan megoszt. Ezen kívül mindkét fél generál egy-egy titkos véletlen kulcsot, K_A-t és K_B-t, melyek nagy egész számok. Ezek után A a nyilvános csatornán elküldi B–nek az

$$m_A = a^{K_A} \mid n$$

maradékot, B pedig A-nak az

$$m_B = a^{K_B} \mid n$$

maradékot, melyekből a titkos kulcsokat csak nagy számítási teljesítménnyel lehet visszafejteni. Ezután A és B is előállítja helyben a munkamenet

$$m = \left(a^{K_A}\right)^{K_B} \mid n = \left(a^{K_B}\right)^{K_A} \mid n$$

titkos kulcsát.

Az n, K_A és K_B számokat olyan nagyra kell választani, hogy a titkos kulcs visszafejtése termodinamikai korlátba ütközzön. Ez a Diffie-Hellman módszer és általában a nyilvános kulcsú rendszerek esetében több száz decimális jegyet jelent.

9.2. Az RSA[18] módszer

Míg a Diffie-Hellman módszer egy közös titkos kulcs létrehozását tette lehetővé két partner számára, addig az RSA módszer bevezeti a nyilvános kulcs fogalmát, mellyel bárki tud olyan rejtett üzenetet készíteni, melyet csak az adott nyilvános kulcshoz tartozó titkos kulcs birtokosa tud elolvasni.

A módszert az előbbinél szélesebb körben alkalmazzák a munkamenet-kulcsok átvitelére és digitális aláírások készítésére. Az eljárás alapja az, hogy két nagy prímszám szorzatának a faktorizálására nem ismert gyors módszer[19].

Az RSA küldő fele (A) két nagy véletlen prímszámot generál (p_1 és p_2), és ezekből előállítja az $n= p_1p_2$ és az $m=(p_1-1)(p_2-1)$ számokat. Ezután keres egy olyan e számot, amelyre (e, m)=1, és meghatározza ennek m-re vonatkozó multiplikatív inverzét, d-t. Mindezek közül az n és e számokat nyilvánosságra hozza, ezek alkotják a nyilvános kulcsot. A titkos kulcs, melyet A senkinek nem küld el, a d szám. A küldő fél a nyíltszöveget egész számokká alakítja, ezután számonként a nyilvános kulccsal (melyet A-tól vagy a nyilvános kulcstárból szerez be) előállítja a rejtett üzenetet:

$$Y = X^e \mid n,$$

melyet csak A tud elolvasni a d titkos kulcs használatával:

$$X = Y^d \mid n$$

[18] A módszer feltalálóiról (Rivest, Shamir, Adelman) kapta a nevét.

[19] 1993-ban 1600 hálózatba kapcsolt munkaállomáson a szabad processzoridő felhasználásával 8 hónap alatt faktorizáltak egy 192 decimális jegyű számot.

Az eljárás alapja az a számelméleti módon könnyen bizonyítható tény, hogy

$$X = X^{ed} \mid n$$

Az RSA ellen számos különböző típusú támadás lehetséges (pl. választott rejtett szöveg), melyek ellen az ezt szabályozó biztonsági szabvány körültekintő alkalmazásával lehet védekezni. Gondot okoz az is, hogy a véletlenszerűen generált p_1 és p_2 számokról eldöntsük, hogy valóban prímek-e. Ha nem, esetleg sokkal könnyebb a szorzatukat faktorizálni. Ezek faktorizálása azonban éppúgy gyakorlati nehézségekbe ütközik, mint n-é. Ezért statisztikai tesztekkel korlátozzák annak a valószínűségét, hogy nem prímek.

A nyilvános kulcsú kriptorendszerek megoldják ugyan a kulcskezelési problémát, de érzékenyebbek a különböző típusú támadásokra, mint a titkos kulcsú rendszerek. Az RSA rendszer legfontosabb alkalmazása az, hogy az internetes munkamenet elején az egyik fél a másiknak az RSA segítségével át tudja adni azt az általa generált titkos (DES) kulcsot, amelyet a munkamenetben ezután használni fognak. Az RSA módszert használja a széles körben elterjedt SSL protokoll is, melyben az átvitt titkos kulcs mérete 40 és 168 bit között állítható. Az RSA bővebb leírása, egyben működésének a demonstrációja megtalálható a tárgy honlapján.

9.3. A digitális aláírás

Az internetes alkalmazások másik fontos elvárása a nyilvános dokumentumok (pl. árajánlatok) sértetlenségének és letagadhatatlanságának a biztosítása. Erre szolgál a nyilvános dokumentum végéhez csatolt digitális aláírás, a nyilvános kulcsú kriptorendszerek másik fő alkalmazási területe. Az aláírás az RSA alkalmazásával a következő lépésekben készíthető el:

1. Az aláírni kívánt nyilvános dokumentumhoz az aláíró (A) ellenőrző összeget (hash) készít egy szabványos, CRC kódokhoz hasonló algoritmussal, mint például az MD5 vagy az SHA. Az algoritmus úgy van konstruálva, hogy ha a dokumentum megváltozik, akkor a változott dokumentumhoz tartozó hash összeg igen nagy valószínűséggel nem fog egyezni az eredetivel.
2. A hash értéket és az alkalmazott hash módszer nevét (azonosítóját) A a saját titkos kulcsával rejti. Az így kapott **digitális aláírást** a dokumentum végéhez csatolja.

Az aláírás ellenőrzésének a lépései:

1. A dokumentumot ellenőrizni kívánó fél (B) beszerzi A nyilvános kulcsát, és ezzel kinyeri a digitális aláírásból a hash értéket és típust.
2. Elkészíti a dokumentumhoz a jelzett típusú hash értéket, és összehasonlítja az aláírásból kivett értékkel. Ha a kettő egyezik, a dokumentum nem változott (hiteles), és A írta alá.

A módszer alapja az, hogy az RSA konstrukciója miatt a nyilvános és a titkos kulcs szerepe felcserélhető ($X^{de} = X^{ed}$), és amit az egyikkel rejtettek, csak a másikkal lehet megfejteni.

Az aláírással vissza lehet élni olyan módon, ha a támadó (megfelelő erőforrásokkal) már előre két különböző tartalmú, de azonos hash értéket szolgáltató dokumentumot készít, az egyiket aláíratja A-val, utána az aláírást átmásolja a másik dokumentum végére (születésnap-támadás). Ez a támadás kivédhető úgy, ha A aláírás előtt a tartalmat nem érintő, lényegtelen változtatásokat tesz a dokumentumban.

9.4. A digitális tanúsítványok

A nyilvános kulcsú kriptorendszerek, így az RSA legérzékenyebb pontja a 9.2. fejezetben vázolt problémákon túl a nyilvános kulcs **hitelessége**. Ha a támadó (C) pl. a hálózati eszközök manipulációjával el tudja hitetni A-val, hogy a saját nyilvános kulcsa B-től származik, B-vel pedig azt, hogy A-tól, akkor a kommunikáció kezdetétől folyamatosan el tudja olvasni és át tudja írni akármelyik A és B közötti üzenetet („man in the middle" támadás). Hasonlóképpen veszélyben vannak a nyilvános kulcstárak, a támadó ebben is lecserélhet egy kulcsot a sajátjára. Ezért a nyilvános kulcs felhasználása előtt meg kell győződni arról, hogy az valójában kihez tartozik. Erre szolgálnak az úgynevezett **tanúsítványok** (certificate), melyekkel a nyilvános kulcsú rendszer (RSA) alkalmazása a következő lépésekben lehetséges:

1. A nyilvános kulcsot meghirdetni szándékozó fél (A) elkészíti saját nyilvános-titkos kulcspárját.
2. A egy külső testülethez vagy hatósághoz (**certificate authority**, CA) elviszi vagy elküldi (biztonságos módon!) a nyilvános kulcsát. A CA ellenőrzi (nem feltétlenül informatikai módszerekkel!) A személyazonosságát, majd kiállít egy tanúsítványt, amely tartalmazza A nevét és nyilvános kulcsát, és aláírja a tanúsítványt a saját (hatósági) titkos kulcsával.
3. Ha valaki A-nak kíván küldeni egy RSA-val rejtett levelet, a CA-tól, vagy magától A-tól elkéri A tanúsítványát, és megszerzi a CA nyilvános kulcsát. Ez utóbbi segítségével ellenőrzi a tanúsítvány digitális aláírását. Ha rendben van, akkor a tanúsítványon található nyilvános kulcs valóban A-hoz tartozik.
4. A nyilvános kulcsával B rejti az üzenetet, és elküldi A-nak.

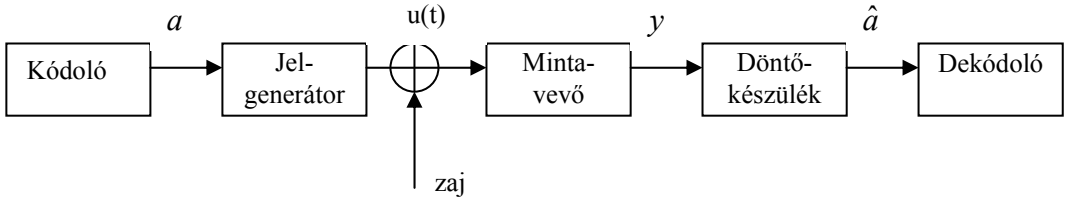
A módszer gyenge pontja természetesen a CA nyilvános kulcsának a hitelessége, ami megint csak egy, a CA feletti hatóság által kibocsátott tanúsítvány segítségével lehetséges stb. A CA-k tehát ebben a megoldásban hierarchikusan hivatkoznak egymásra, a legmagasabb szintű CA nyilvános kulcsa pedig eleve ismert kell, hogy legyen minden résztvevő információs rendszer, szoftver számára.

A hierarchikus megoldás helyett elképzelhető hálózatos rendszerű megoldás is, amelyben a kommunikáló felek maguk is aláírják mások tanúsítványait. Ez az alapja a népszerű PGP (Pretty Good Privacy) rendszernek.

IV. DÖNTÉS- ÉS HÍRKÖZLÉSELMÉLET

10. A bináris hírközlési feladat

Ebben a részben visszatérünk a kiinduló csatornamodellünkhöz, és megvizsgáljuk a csatorna fizikai felépítését. A következő ábra a kódoló és a dekódoló közötti diszkrét, zajos csatorna belső modelljét mutatja.



10.1. ábra. A diszkrét, zajos csatorna modellje

Az ábrán a valamelyik csatornaszimbólum, bináris csatornán „0” vagy „1”, $\hat{a}$ ennek a vevőoldali becslője. A kommunikációt ütemezzük, a mintavevő t_0 időnként vesz mintát a csatorna u(t) fizikai jeléből (mely folytonos, folyamatos feszültségfüggvény). A csatornaszimbólumhoz a jelgenerátor rendeli hozzá az adott időszelet jelalakját a alapján. Bináris csatorna esetén két elemi jelre (egy ún. **elemi jelpárra**) van szükség. Néhány elemi jelpár:

- On-off keying, OOK. Az „1’-hez konstans magas, a „0”-hoz konstans alacsony feszültségértéket rendelünk
- Phase-shift keying, PSK. Az „1’-hez 0, a „0”-hoz $\pi/2$ kezdőfázisú színuszos jelet rendelünk.
- Frequency-shift keying, FSK. Az „1’-hez kisebb, a „0”-hoz nagyobb frekvenciájú színuszos jelet rendelünk.

A csatornán fellépő additív zaj természetesen eltorzítja az elemi jel alakját. A döntőkészüléknek az időszelet adott pontjáról (pl. a közepéről) vett minta alapján kell meghatároznia $\hat{a}$-t.

10.1. A standard döntési szabály

A döntés alapja a vett y értékkészletének a particionálása az egyes a_i szimbólumokhoz tartozó A_i tartományokra (a particionálás maga a döntési szabály). A döntési szabály akkor megfelelő (racionális alapokon), ha

$$P(a_i \mid y) \geq P(a_j \mid y),\ \forall i, j,\ i \neq j\ \text{és}\ y \in A_i - ra$$

A $P(a_i \mid y)$ valószínűséget a posteriori feltételes valószínűségnek nevezik. Mivel ezt szeretnénk a döntéssel maximalizálni, ezért ezt a döntési szabályt **maximum a posteriori** (MAP), vagy **standard** döntésnek hívjuk. A hibás döntés valószínűsége:

$$P_{hiba} = P(y \in A_i,\ a \neq a_i)$$

Bizonyítható, hogy ha a standard döntési szabály szerint particionáltuk y értékkészletét, akkor a hibás döntés valószínűsége minimális.

A döntési feladat **Bayes-típusú**, ha

- A hibás döntések összevontan kezelhetők, tehát $\hat{a} = a_i$, $a = a_j$, $i \neq j$ esetén a hiba költsége i-től és j-től független, és
- Létezik és meghatározható az f(a) úgynevezett **a priori** valószínűség-eloszlás.

Ha ezen feltételek valamelyike nem teljesül, akkor a feladat az általános (statisztikai) hipotézisvizsgálat módszereivel oldható meg. Erre egy példa az ellenséges repülőgépek detektálása esetén a légiriadó elrendelése, ahol a vaklárma és az elmulasztott riasztás (a kétféle hiba) költsége nem összemérhető.

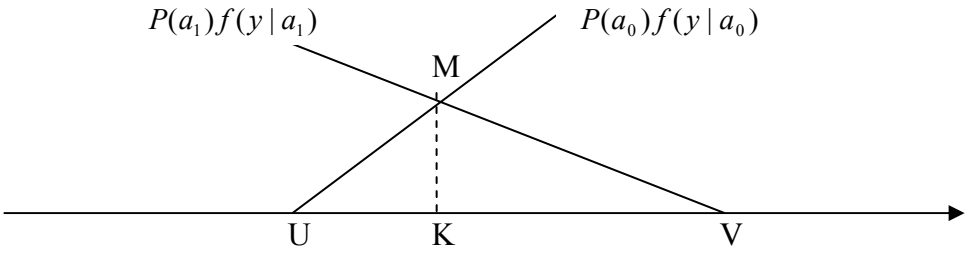
10.2. A bináris hírközlési feladat

A feladat a normális eloszlású zajjal terhelt, bináris szimmetrikus csatornán, OOK elemi jelpárral továbbított csatornaszimbólum (a_0=„0” vagy a_1=„1”) detektálása, azaz a vett folytonos y érték értékkészletének a particionálása. A fenti kritériumok szerint a feladat Bayes típusú (mindegy, hogy „0” helyett „1”-et, vagy „1” helyett „0”-t veszünk). A döntéshez a standard döntési szabály kifejezését a diszkrét eloszlású csatornaszimbólumokra és a folytonos y-ra való tekintettel a feltételes valószínűségek beírásával átalakítjuk az alábbiak szerint:

$$f(a_i \mid y) = \frac{P(a_i) f(y \mid a_i)}{f(y)}$$

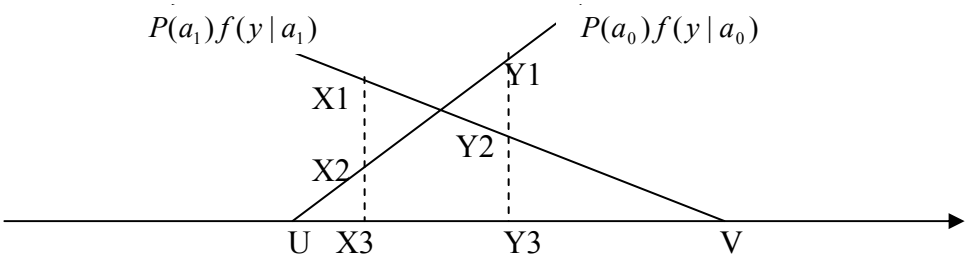
Ebben a kifejezésben $P(a_i)$ az a priori valószínűség-eloszlás, mely a kódoló kimenetén mérhető, $f(y \mid a_i)$ pedig a jelgenerátor és a zaj karakterisztikája, mely a_i folyamatos adásával a mintavevő kimenetén szintén mérhető vagy számítható. A döntési szabály meghatározásához elegendő a kifejezés számlálóit i=0, illetve i=1-re összehasonlítani, mivel a nevező ugyanaz minden i-re. A döntési küszöböt a , illetve a $P(a_1) f(y \mid a_1)$ görbék metszéspontja fogja kijelölni. A metszéspont (a 10.2. ábrán K) egyik oldalán, ahol $P(a_0) f(y \mid a_0) < P(a_1) f(y \mid a_1)$, „1”-re fogunk dönteni, a másik oldalon „0”-ra.

Törlési sáv bevezetése a hiba valószínűségének korlátozására
A bináris hírközlési feladatban a vizsgált csatorna p hibavalószínűsége az alábbi ábrán a KMV és KMU háromszögek területének összegével egyenlő. Ez a hibavalószínűség a csatorna tervezésének fontos paramétere.



10.2. ábra. A hiba valószínűsége

Ha a hibavalószínűség az alkalmazás szempontjából túl nagy, a döntési küszöb körüli törlési sáv bevezetésével átalakíthatjuk a csatornát a 3.1. fejezet szerinti bináris törléses csatornává (10.3. ábra).



10.3. ábra. Törlési sáv bevezetése

A törlési sáv (X3 és Y3 közötti sáv) bevezetése utáni hibavalószínűség már csak az X2X3U és Y2Y3V háromszögek területének összege. A döntőkészülék törlés szimbólumot fog detektálni akkor, ha y X3 és Y3 közé esik. Ekkor vagy újra kell küldeni a törléses hibát okozó szimbólumot (ami csökkenti a kódsebességet), vagy hibajavító kód alkalmazása esetén a vevőoldalon javítani tudjuk a törléses hibát, ha a törlések száma kisebb, mint a kódtávolság. A törléses hiba valószínűsége az ábrán az X1Y2Y3X3 és az X2Y1Y3X3 trapézok területének az összege, szintén fontos tervezési szempont.

10.3. A szimbólumközi áthallás

A csatornán való áthaladáskor az elemi jelek nemcsak a csatorna zaja, hanem annak aluláteresztő jellege miatt is szükségképpen torzulnak. Például az OOK jelekből a jelformáló által képzett, az időszeletek határainál ugrásokat tartalmazó eredeti időfüggvény a csatornán való áthaladás során a nagy frekvencia-összetevők levágása miatt ellaposodik, az ugrások meredeksége csökken. Ennek eredményeképpen minél kisebb az időszelet, az elemi jelek annál inkább „átlógnak" a szomszédos időszeletekbe. Ennek a csatorna zajától függetlenül, elkerülhetetlenül fellépő jelenségnek **szimbólumközi áthallás** (crosstalk) a neve.

A szimbólumközi áthallás ellen olyan módon is lehet védekezni, hogy az elemi jeleket nem igyekszünk az időszeletre korlátozni, hanem inkább arra törekszünk, hogy saját időszeletük kivételével minden időszelet közepén, a mintavételi pontban 0 legyen az értékük (Nyquist módszere).

11. Analóg átvitel

Ha a kommunikációs folyamat során a forrásból származó információt nem alakítjuk át forrásszimbólumok sorozatává, hanem az eredeti, folytonos fizikai jelet (pl. a beszédhangot leíró feszültségfüggvényt) továbbítjuk a nyelő felé, akkor analóg átvitelről beszélünk. Alkalmazási területei a földi rádió és TV műsorszórás.

11.1. Alapsávi átvitel

Az átvinni kívánt jelhez alsó és felső frekvenciakorlátot (b és B) rendelhetünk, amit a kommunikációs rendszer tervezése során kihasználhatunk. (A telefonszabvány szerint például b=300Hz, B=3400Hz.) Az alapsávi átvitel során a jel frekvenciatartományban elfoglalt helyét nem változtatjuk meg. Ha például beszédhangot mikrofonnal elektromos jellé alakítunk, és ezt a jelet továbbítjuk, akkor az elektromos jel is a beszédhang frekvenciatartományát foglalja el.

Az alapsávi átvitelt a gyakorlatban igen korlátozottan használják a következők miatt:

- Az átvitelhez használt fizikai közegnek B-nél sokkal nagyobb kihasználható sávszélessége is lehet
- A kommunikációs rendszer megvalósításában nehézségek lépnek fel, ha B/b >> 1 (pl. az antenna tervezésekor)

Ezért a jelet a B-hez képest nagy frekvenciájú vivőjel segítségével frekvenciatartományban eltoljuk a továbbítás előtt. Ezt az eljárást **modulációnak**, a vevőoldalon az eredeti jel visszaállítását **demodulációnak** nevezik.

11.2. Modulált átvitel

Amplitúdómoduláció

A modulációnak ebben a formájában a továbbítandó jelet (s(t)) szorozzuk a koszinuszos vivővel (v(t)). A modulált jel spektrumának feltérképezéséhez legyen s(t) koszinuszos mérőjel:

$$s(t) = M\cos\mu t$$

$$v(t) = V\cos\Omega t,\ \mu << \Omega$$

Ekkor a modulált jel a koszinuszra vonatkozó azonosság felhasználásával:

$$e(t) = s(t)v(t) = MV\cos\mu t\cos\Omega t = \frac{MV}{2}\left(\cos(\Omega+\mu)t + \cos(\Omega-\mu)t\right)$$

Látható, hogy a jelet tartalmazó, eredetileg b…B sáv a moduláció után két, Ω-ra szimmetrikus, Ω-B…Ω-b, illetve Ω+b…Ω+B **oldalsávként** jelenik meg, de a vivőfrekvencia (Ω) körüli 2b frekvenciasávot nem használjuk. Ezért ennek a modulációs eljárásnak a neve AM-DSB-SC (amplitude modulated, double side band, supressed carrier). A modulált jel alakja az időtartományban olyan Ω frekvenciájú jel, melynek burkolóját a mérőjel adja.

A demoduláció során a vett e(t) jelet újra szorozzuk a vivővel, majd aluláteresztő szűrést alkalmazunk:

$$\hat{e}(t) = e(t)v(t) = MV^2\cos\mu t\cdot\cos^2\Omega t = \frac{MV^2}{2}\cos\mu t\cdot\left(1+\cos 2\Omega t\right)$$

Látható, hogy ha a μ frekvencia feletti komponenseket aluláteresztő szűréssel levágjuk, akkor az eredeti mérőjellel arányos jelet kapunk.

A módszer hátránya, hogy a csatornán elfoglalt sávszélesség a jel sávszélességének (B) kétszerese. A továbbfejlesztésként kialakított AM-SSB-SC (Single Side Band) módszernél az adás előtt Ω frekvenciánál vágó aluláteresztő szűrésnek vetik alá a modulált jelet. A demoduláció a DSB módszerhez hasonló. Az AM-SSB-SC modulációt használják a rádióamatőrök, mivel kis adóteljesítménnyel (≈100 W) gyakorlatilag az egész világot el lehet érni. Az átvitel minősége gyenge, emiatt inkább csak beszéd átvitelére alkalmas a módszer. A gyenge minőség oka egyrészt, hogy a zaj közvetlenül az információt hordozó amplitúdóhoz adódik hozzá, másrészt pedig, hogy nagy távolságokon a földfelszín és a mozgó ionoszféra között ide-oda verődő rádióhullám fázisa a vevőnél időben nem állandó, ezért a demoduláció változó erősségű és minőségű jelet eredményez.

Frekvencia- és fázismoduláció

A frekvencia- és fázismoduláció (FM/PM) alapgondolata az, hogy az átvinni kívánt információt a modulált jel frekvenciájában ill. fázisában tároljuk, ezáltal nagyobb védelmet biztosítunk a rendszernek a csatorna additív zajával szemben. E módszereket az 1930-as évektől alkalmazzák, jelenleg elsősorban a földi műsorszórásban.

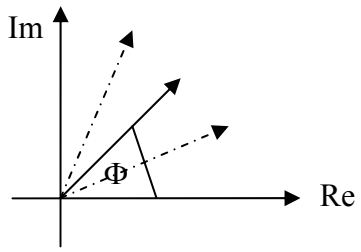
Legyen a koszinuszos mérőjel, a vivő és a modulált jel az előbbiekhez hasonlóan:

$$s(t) = M \cos \mu t$$
$$v(t) = V \cos(\Omega t + \Phi),\ \mu << \Omega$$
$$e(t) = V \cos(\Omega t + \Phi + m(t)),$$

ahol *m(t)* az átvinni kívánt jel függvénye:

$$m(t) = C \cdot s(t) \text{ fázismoduláció,}$$
$$\frac{d\, m(t)}{d\, t} = C \cdot s(t) \text{ frekvenciamoduláció esetén.}$$

Az időtartományban a modulált jel amplitúdója állandó (V), pillanatnyi frekvenciája a moduláció hatására változik (fázismodulációval a fenti koszinuszos mérőjellel a változási tartomány 2MC). Ha a modulált jelet fazorábrán ábrázoljuk, a jel vektorának „bólogatása” hordozza az információt:



AM és FM jelek esetén a névleges Φ fázistól való maximális szögeltérést (az ábrán a két szaggatott nyíl közti szögtartomány felét) **fázislöketnek** (D_Φ) nevezzük.

A modulált jel spektruma egyetlen koszinuszos mérőjel esetén is végtelen sok Bessel-függvény összegeként fejthető sorba. A spektrum 2B-nél nagyobb sávot foglal el. Ha nem kerül továbbításra a teljes tartomány, akkor a vevőoldalon nemlineáris torzítást lehet tapasztalni (minőségi romlás). Mind a gyakorlati sávszélesség, mind a zajtűrés függ a fázislöket nagyságától:

- ha a fázislöket nagy, nagyobb a gyakorlati sávszélesség, viszont
- jobb a moduláció zavartűrése (jobb a vevőoldali jel/zaj viszony).

Ezért a fázislöket megválasztása fontos tervezési szempont.

Az FM/PM moduláció megvalósításában feszültségvezérelt oszcillátorokat (VCO) illetve fáziszárt hurkokat (PLL) használnak. A megvalósítás részleteit a jegyzet nem tárgyalja.

Köszönetnyilvánítás

Köszönöm *Vassányi Miklós* segítségét az ismeretelméleti áttekintés összeállításában, *Gaál Balázs* és *Nemetz András* segítségét a szerkesztésben és a képletek, ábrák megrajzolásában.

Irodalom

1. Richard B. Wells: *Applied Coding and Information Theory for Engineers*, Prentice Hall, 1999. **Az információelméleti anyag alapja.**
2. Linder Tamás, Lugosi Gábor: *Bevezetés az információelméletbe*, Műegyetemi Kiadó, 1997, jegyzetszám: 51445. Precízebb matematikai alapok.
3. Steven Roman: *Introduction to Coding and Information Theory*, Springer, 1997. Kiegészítő anyag.
4. Simon Singh: *Kódkönyv*, Park Könyvkiadó, 2001. A kriptográfia folklórja.
5. Wenbo Mao: *Modern Cryptography*, Prentice Hall, 2004. **A kriptográfiai rész alapja.**
6. R.B.Ash: *Information Theory*, Dover Publications, 1990. Kiegészítő anyag.
7. R.E. Smith: *Internet security*, Addison-Wesley 1997. **A kriptográfiai rész alapja.**
8. Csibi Sándor (szerk.): *Információ közlése és feldolgozása*, Tankönyvkiadó, 1986. **A hírközléselméleti anyag alapja.**
9. Dallos György: *Feladatgyűjtemény az Informácó közlése és feldolgozása című tárgyhoz*, Műegyetemi Kiadó, 1998, jegyzetszám: 55000. A hírközléselméleti részhez.
10. Háttéranyagok, linkgyűjtemény, demoprogramok a tárgy honlapján: http://www.irt.vein.hu/~vassanyi/info/

FÜGGELÉK

F1. Az egyenletes forráseloszlás maximalizálja a forrásentrópiát.

Bizonyítás: Legyen $A = \{p_0, p_1, \ldots\}$ és $|A| = M$. A címben szereplő állítást bebizonyítottuk, ha megmutatjuk, hogy: $H(A) - \log(M) \le 0$, és egyenlőség csak akkor áll fenn, ha $p_i = \frac{1}{M}$, mivel $\log(M)$ az egyenletes eloszlású forrás entrópiája.

Átalakítva:

$$\sum_i p_i \log \frac{1}{p_i} - \log(M) =$$

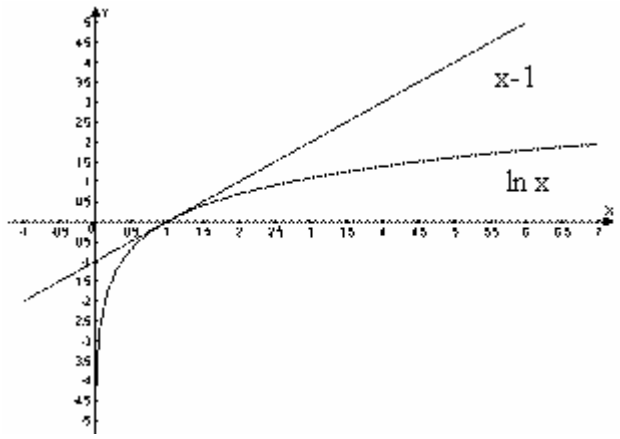
$$= \sum_i p_i \log \frac{1}{p_i M} =$$

$$= \frac{1}{\ln 2} \sum_i p_i \ln \frac{1}{p_i M} \qquad (1)$$

Itt kihasználjuk, hogy $x = \frac{1}{p_i M}$ helyettesítéssel az összegzés összes tagjára alkalmazható a következő egyenlőtlenség:

$$\ln x \le x - 1, \text{ és egyenlőség akkor áll fenn, ha } x = 1$$

Ez az egyenlőtlenség a görbék analízisével bebizonyítható.



Az egyenlőtlenséget (1)-be írva:

$$H(A) - \log M \le \frac{1}{\ln 2} \sum_i p_i \left(\frac{1}{p_i M} - 1 \right) = 0$$

Tehát valóban $H(A) - \log(M) \le 0$ és egyenlőség akkor áll fenn, ha $x = 1$, vagyis $p_i = \frac{1}{M}$ minden *i*-re, ami az egyenletes eloszlás esete.

F2. A kód megfejthetősége egy történelmi példán

„Soha sem tesz annyi kárt egy gyönge népben egy erős zsarnok, mint egy erőteljes népben egy gyönge király.

Jeruzsálemi András egész tizenhét évi uralkodása alatt nem tett egyebet a magyar nemzet, mint saját sírját ásta.

A király pazar, a nemzet koldus, kívül szükségtelen harc, belül pártháború.

Az önakarattalan királyt majd büszke, nagyravágyó nők, majd önző, alacsony lelkű tanácsosok kormányozzák, s ha mindeniktől megszabadult – saját önállástalan lelke.

Első neje, meráni Gertrud, kit a természet nem királynénak, hanem királynak teremte, de semmi esetre sem a magyar számára: büszkesége, pazarlása s idegen udvara által ellenségévé tette trónjának az ország minden rendjeit.

A főpapokat és nemeseket bosszantá, hogy minden rokonát, tanítóját, udvarmesterét, gyóntatóját érseki, báni, főispáni hivatalokba rakta, s a köznép nyögött a vérét sajtoló adó terhe alatt.

A magyar zúgolódva látta magát mindenében megraboltatni: hivatalaiban, rangbüszkeségében, vagyonában és életében, csak egy csepp kelle még a bosszú poharához, hogy kicsorduljon.

E csepp volt a női erény könnye.

Ami a Tarquiniusokat megbuktatá, az lőn Gertrudnak veszte is.

Az akkori nádor Bór Benkének, kit ismertebb néven Bánk bánnak neveznek, csudaszép neje volt a királyné udvarában, ki iránt Ottó, Gertrud testvére, tiltott szerelmet kezde érzeni.

A szép nő szebb volt erényei miatt. A magyar nők egyik főtulajdona volt eleitől fogva a hűség, szűziesség, és itt a választás nem volt nehéz a délceg, daliás nádor s az idétlen meráni herceg között.

Ottót Németországból az igazság keze üldözé nénje udvarába. Fülöp király orgyilkosai közt őt is megismerék. S aki értett az orgyilokhoz, értett a méregkeveréshez is. Egy este, midőn a királynéval s a szép Melindával együtt vacsorált, a nádor nejének poharába szerelemitalt vegyíte, s a királyné saját szobájában egyedül hagyta a herceget Melindával.

A nádor, ki éppen akkor tért vissza ítéletosztó körútjából, a kétségbeesés könnyei közt, félőrülten találta hitvesét, s míg szemei kiolvasták e könnyekből a helyrehozásra nem, csak megtorlásra váró eseményt, nejének rokonai, Mikhál, Simon és Petur bánok ujjal mutattak a bosszú tárgyára.

Ez Gertrud volt.

Rég el volt határozva a királyné halála az összeesküvők által. Az általuk felszólított esztergomi érsek, János, kérdésükre e dodonai kétértelműségű feleletet adta:

„**Reginam occidere nolite timere bonum est; si omnes consentiunt ego non contradico**.”

Melyet a különböző megszakítás szerint így is lehet érteni:

„A királynét megölni nem kell – félnetek jó lesz; ha mindenki beleegyez – én nem – ellenzem.”

De emígy is lehet magyarázni:

„A királynét megölni nem kell félnetek – jó lesz; ha mindenki beleegyez, én nem ellenzem.”

De a megsértett férj bosszúja nem kérd és nem hallgat meg tanácsot. Midőn a király éppen Halicsban volt hadat viselni s országát azalatt Bánk bánra bízta, ez a királynét saját palotájában meggyilkolá. Ottó megmenekült, meggyilkolt testvére kincseit is magával elrabolva.

A visszatérő király az összeesküvőket családostul kiirtatá; egyedül Bánkot, neje gyilkosát nem volt bátorsága megöletni. Érzé: hogy a meggyalázott nő miatti keserv nagyobb, mint a megölt miatti. *(Bánk bán történetét örökíté meg Katona József hasonló című drámájában, mely elsőrendű remekműve a magyar irodalomnak.)"*

Forrás:

http://www.mek.iif.hu/porta/szint/human/szepirod/magyar/jokai/osszes/html/070/jokai36.htm

F3. „Nincs olyan veszteségmentes (megfejthető) tömörítő algoritmus, amely egy bináris forrás tetszőleges N hosszúságú szimbólumsorozatát *minden esetben* akár csak 1 bittel tömörebben, azaz legfeljebb N-1 bináris szimbólummal kódolni tudná”

Bizonyítás:
Az állítás a doboz-elvből következik. Indirekt módon tegyük fel, hogy létezik a fent megfogalmazott kódoló egység. Ennek összesen 2^N darab különböző N hosszúságú bináris bemenete lehet, a lehetséges kimenetek száma viszont 2^{N-1}, ha a kimenet N-1 hosszú, 2^{N-2}, ha a kimenet N-2 hosszú, stb. Ezek mindegyike előfordulhat, de az összes lehetséges kimenetek száma így is kisebb, mint a lehetséges bemenetek száma:

$$2^{N-1} + 2^{N-2} + \ldots + 2^1 + 1 < 2^N \quad (= 2^N - 1)$$

Ezért a kódoló szükségképpen ugyanabba a kimenetbe visz át legalább két bemenetet, tehát a kód nem megfejthető.