EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

AZ MP3 MATEMATIKÁJA

Szakdolgozat

Mészáros Mária

Matematika BSc, Alkalmazott matematikus szakirány

Témavezető:

Dr. Szabó Csaba, egyetemi docens

Algebra és Számelmélet Tanszék

Budapest, 2010

Tartalomjegyzék

1. Bevezető....1
2. MPEG-1 Layer III....2
3. A hang fizikai jellemzői....4
4. Mintavételezés....6
5. Fourier-transzformációk....8
6. Kvantálás....11
7. Forráskódolás....16
8. Huffman-kódolás....18
9. Adaptív Huffman-kódolás....23
10. Hibajelző és hibajavító kódolás....25
11. CRC kódolás....28

Irodalomjegyzék....30

1. fejezet

Bevezető

Szakdolgozatom célja az egyik legnépszerűbb hangkódolási eljárás, az MP3 tömörítés matematikai bemutatása. Azért választottam ezt a témát, mert érdekes végigvezetni, hogy egy természeti jelenségből, jelen esetünkben egy hangból, hogyan lesz digitális, számítógéppel lejátszható adat minél tömörebben.

Dolgozatomat az MP3 történetével és adataival kezdem. Majd a hang fizikai jellemzőivel folytatom, mivel ezeket figyelembe veszi digitalizáláskor az MP3 eljárás. Ezután értelemszerűen a digitalizálás lépéseivel (mintavételezés, Fourier-transzformációk, kvantálás) folytatom, kiemelve a kvantálás szép matematikáját. (Ezen fejezetekhez felhasznált irodalom: [1], [2], [6], [7].)

A digitalizálás után az eljárás még egy veszteségmentes tömörítést is alkalmaz, a Huffman-kódolást (felhasznált irodalom: [3]). Szakdolgozatomban erre fektettem a legnagyobb hangsúlyt, hiszen személy szerint nekem ez a rész tetszett a legjobban, ezért írtam egy kis kitekintést is az adaptív Huffman-kódolásról (felhasznált irodalom: [8], [9]). Végül, de nem utolsó sorban a leggyakrabban használt CRC hibajelző kódolás elméletéről esik szó (felhasznált irodalom: [4], [5], [10], [11]), melyet az állomány épségének ellenőrzésére használnak.

Ezúton is szeretnék köszönetet mondani témavezetőmnek, Dr. Szabó Csabának türelméért, hasznos tanácsaiért, valamint hogy bármikor fordulhattam hozzá, ha segítségre volt szükségem a szakdolgozatom írásával kapcsolatban.

2. fejezet

MPEG-1 Layer III

1988 májusában négy munkacsoportot alakítottak az ISO (Nemzetközi Szabványügyi Hivatal) és az IEC (Nemzetközi Elektrotechnikai Bizottság) elnevezésű szervezetek közösen. A munkacsoportokat azzal a feladattal hozták létre, hogy egységes szabványokba foglalják a különböző tömörítési algoritmusokat. Ezek egyike volt az MPEG (Moving Picture Experts Group) mely a digitális video- és audiotömörítés szabványosítását tűzte ki célul. Az MPEG nem határozza meg a tömörítés algoritmusát. Ez teszi lehetővé a tömörítőprogramok állandó fejlődését és a különleges alkalmazásokhoz való alkalmazkodását.

Az MP3 az MPEG-1 Layer III (más nevén MPEG Audio Layer III) rövidítése, mely az MPEG szabvány audio alcsoportját jelöli. A Fraunhoffer Gesellschaft, egy német társaság dolgozta ki az MP3-at, s rendelkezik a technológia alapvető szabadalmával.

Az audio tömörítési eljárásnak három különböző rétegét definiálták. A három réteg, az MPEG-1 Layer I, II, III, (ilyen sorrendben keletkeztek egymás után) főként a kódoló algoritmus bonyolultságában tér el egymástól, méghozzá a réteget jelölő római szám nagyságához mérten.

Az adattömörítés célja, hogy nagyobb adatmennyiséget a tárolás vagy továbbítás hatásfokának növelése céljából minél kisebb méretűre zsugorítson. Az MP3 eljárás kihasználja a hang tulajdonságait és az emberi fül hallásmechanizmusát.

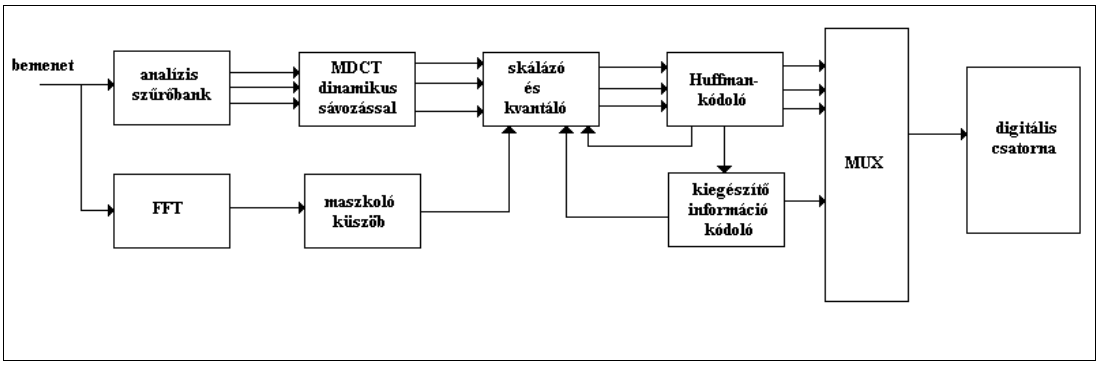
A munkacsoport célja tehát az volt, hogy olyan hatékony tömörítő eljárást keressenek, ami olyan adatokat „ejt” ki az állományból, amelyek nem befolyásolják dominánsan a kapott hang élvezhetőségét. Másik fontos szempont lehet a kitömörítés sebessége, hiszen hangot úgy érdemes tömörítetten tárolni, hogy azt kicsomagolás nélkül, a lejátszó programokkal tudjuk meghallgatni.

Az MP3 hangtömörítő eljárás hangvázlata az 1. ábrán látható. Az egyes lépéseket és matematikai jelentésüket az egyes fejezetekben tárgyalom.

Az egyszerűbb Layer I és II eljárás analízis szűrőbankja a bemeneti jelet 32 részsávba képezi le, melynek szélessége egyenként a mintavételi frekvencia 64-ed része. A mintavételezés 32, 44.1 vagy 48 kHz-el történhet. Ezután részsávonként egy transzformációs kódolás következik, melyek a következők: Layer I esetén egy 512 pontos, Layer II esetén egy 1024 pontos gyors Fourier-transzformáció (FFT). Majd kiválasztja a 15 kvantáló egyikét minden egyes részsáv számára úgy, hogy optimális legyen.

A napjainkban inkább népszerű Layer III eljárás az elődeihez képest lényeges, a hangminőséget javító eltéréseket tartalmaz. A 32 részsáv mindegyikét további 18 összetevőre bontja módosított

diszkrét koszinusz transzformáció (MDCT) felhasználásával. A frekvenciakomponensekre ezután egy nemegyenletes kvantálót alkalmaz, majd a kimenetet Huffman-kódolással tömöríti.



1. ábra. *Az MPEG-1 Layer III hangtömörítő eljárás vázlata*.

3. fejezet

A hang fizikai jellemzői

Mielőtt a hangtömörítésről lenne szó, szükséges néhány hanggal kapcsolatos fogalom felvázolása és az emberi fül korlátainak bemutatása.

Fizikai jellegét tekintve a hang valamilyen rugalmas közeg mechanikai rezgéséből áll. A rezgések a részecskék ide-oda mozgása révén keletkeznek és amennyiben ez a mozgás szinuszos, harmonikus rezgésről beszélünk. Az emberi fül által hallható hang konkrétabban pedig nem más, mint a rezgést végző részecskék által létrehozott levegőben terjedő nyomáshullám.

A későbbiekben fontos fogalom lesz a *frekvencia*, ami az egy másodperc alatti rezgések száma. Frekvenciáról csak akkor beszélünk, ha harmonikus rezgésről van szó, mértékegysége az 1/s, vagy más nevén a Hz (hertz, ez határozza meg a hangmagasságot). Az ember a különböző frekvenciájú hangokat különböző hangmagasságú hangoknak érzékeli. A hang frekvenciája különböző típusú közegekben más és más tartományokra osztható:

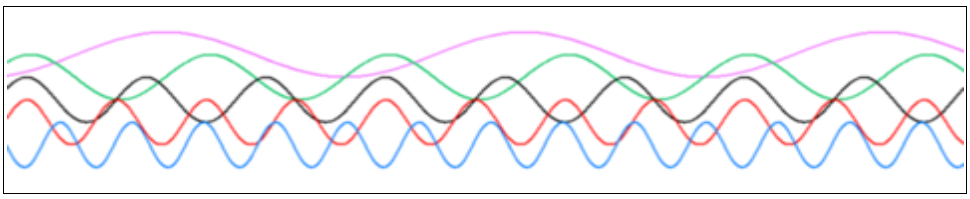
- infrahangok: 0-tól 20 Hz-ig,
- hallható hangok: 20Hz-től 20kHz-ig,
- ultrahangok: 20kHz-től 1GHz-ig,
- hyperhangok: 1GHz-től 10THz-ig.

Az *amplitúdó* a részecske legnagyobb kitérése nyugalmi helyzetéből, ez határozza meg a hangerőt.

A hanghullámok számunkra fontos tulajdonsága, hogy a természetes hang – amit röviden hangnak nevezünk – szinuszhangok összegéből áll.

Hallásunk frekvencia és szintfüggő, a teljes frekvenciatartományban szinte lehetetlen matematikai összefüggéssel leírni. Ezért inkább megfelelő méretű kis részekre osztjuk a frekvenciasávot, és ezeket a kis tartományokat már tudjuk jellemezni egyszerűen, matematikai eszközökkel.

Fourier francia matematikus felismerésének köszönhetően ma már tudjuk, hogy minden folytonosan változó, periodikus jel felbontható állandó frekvencián rezgő szinuszhullámok összegére, vagyis egy összetett jelet egyszerűbben, frekvenciakomponenseire bontva tudunk vizsgálni. A hangoknál a rezgések maguk mindig periodikusak, ellenkező esetben zörejről vagy zajról beszélünk.



2. ábra. *Különböző frekvenciájú szinuszhullámok, a lentebbiek magasabb frekvenciájúak.*

Megvizsgálva az emberi hallás fizikai jellemzőit, tapasztalatból tudjuk, hogy hallásunk mind frekvencia, mind pedig a hangnyomás tartományokban korlátozott. A fül érzékenysége a frekvenciával együttesen változik. A 4 kHz-es tartományban a legnagyobb, kisebb és nagyobb frekvenciákon pedig leromlik. Fülünk frekvenciatartománya 20Hz és 20KHZ közé esik.

Frekvenciában közeli hangoknál fellép a *hangelfedés* jelensége, ami akkor jön lére, amikor a hang egy másik hang által a fülünk számára takarásba kerül, nem hallhatóvá válik. Hangelfedés időben is létezik: a hangos jelek megszűnését követően bizonyos ideig eltart, amíg az általa addig elfedett összetevők ismét hallhatóvá nem válnak. A hallható hangok tartományában különböző frekvenciákon a hangelfedés mértéke a magasabb frekvenciákon erősebben jelentkezik.

Fülünk hallásbeli korlátait a hangtömörítésnél, és így az MP3 eljárásnál igazán ki tudjuk használni. Ennek következtében a dekódolás során az eredeti fizikai jelenség nem, csak annak érzékelhető része állítható vissza.

4. fejezet

Mintavételezés

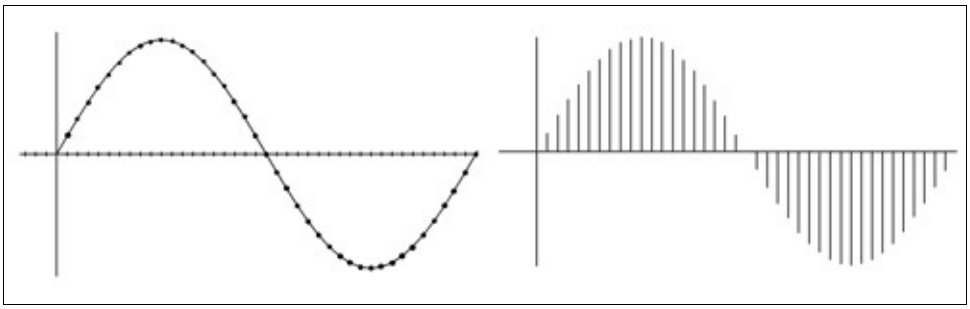
A természeti jelenségek, a fény- vagy hanghullámok folytonos, harmonikus rezgéseket, analóg jeleket hoznak létre. Hogy egy analóg jelzést egy számítógépen használjunk, először diszkrétté kell alakítanunk, mert az csak digitális ábrázolásra képes. Így ha gépen szeretnénk hanganyagot rögzíteni, át kell alakítanunk azt digitális jelek sorozatává. Tehát a hanghoz olyan kódot kell hozzárendelni, amely amplitúdóban és időben diszkréten értelmezett (véges idő alatt véges információt hordoz). Ezt a hangkártya végzi el.

Az első lépés a digitalizáláshoz, hogy a folytonos jelnek csak egy-egy pillanatban vizsgáljuk az értékét, és csak ezeket az értékeket tároljuk el. Ha ezek a pillanatok elég sűrűn követik egymást, vagyis olyan sűrűn, hogy az eredeti jelnek közben nincs ideje nagyon megváltozni, akkor ezek a folytonos jelből vett minták egész jól megközelítik az eredeti jel alakját. Ezt a műveletet hívjuk mintavételezésnek. Tehát a folytonos értékű jelet csak meghatározott időpillanatokban vesszük fel, azaz időben diszkrét függvényt kapunk.

Definíció:

Legyen $T>0$ a *mintavételi idő* és az $X(t)$ folyamat *mintavételezésének* nevezzük az $\{X(kT): k=0,\pm1,\pm2,...\}$ diszkrét idejű folyamatot.

Vagyis az eredeti folytonos jel helyett bizonyos időközönként (ez lesz a T) mintákat veszünk, és csak ezekkel dolgozunk. Egy mintavételezés látható a 3. ábrán.



3. ábra. *A szinuszhullám egy mintavételezése.*

Az első felmerülő kérdés, hogy milyen sűrűn vegyük ezeket a mintákat, ha ugyanis gyakran vesszük őket, akkor a folytonos jelek biztosan elég precízen le fogják írni a mintát, viszont nagyon sok adatot kell tárolnunk, míg ha ritkábban veszünk mintákat, könnyen előfordulhat, hogy azok nem tudják követni a jel változását. Azt kell tehát tudnunk, hogy a jel milyen gyorsan változik, így a mintavétel sűrűségét is meg tudják választani úgy, hogy a leggyorsabb változást is követni tudják mintáink. A kérdésre Shannon, az információelmélet megalapozója adta meg a választ.

Shannon-tétel:

Ha egy jelsorozat szinuszos összetevői közül a legnagyobb rezgésszámúnak frekvenciája *f*, akkor a folytonos jelet *2f* mintavételi frekvenciával mintavételezve a kapott diszkrét jelsorozat egyértelműen leírja az eredeti jelet.

A *mintavételezési frekvencia* az az adott frekvencia, amely megadja, hány mintát veszünk az adott adattárolón található jelből másodpercenként. Mivel az emberi hallástartomány felső határát 20kHz-ben állapították meg, ezt az értéket tekinthetjük a határfrekvenciának, vagyis elegendő ennek kétszeresével, 40kHz-zel mintavételeznünk.

A fenti tételből azonban az is kiderül, hogy a határfrekvenciánál nagyobb értékeket nem tudják majd visszaadni a diszkrét minták, ezért azokat valamilyen módon el kell távolítanunk az eredeti jelből. A szűrőkről érdemes tudni, hogy csak bizonyos fokú meredekséggel képesek vágni, és bár számunkra az lenne az ideális, ha 20 kHz alatt mindent, felette pedig semmit sem engednének át, ez gyakorlatilag lehetetlen. Ez az oka annak, hogy a 40 kHz-es mintavételi frekvencia helyett 44.1 vagy 48 kHz-et használnak az MP3 eljárásban. Ez például a 48 kHz-es mintavételezésnél tehát azt jelenti, hogy másodpercenként 48000 alkalommal veszünk mintát.

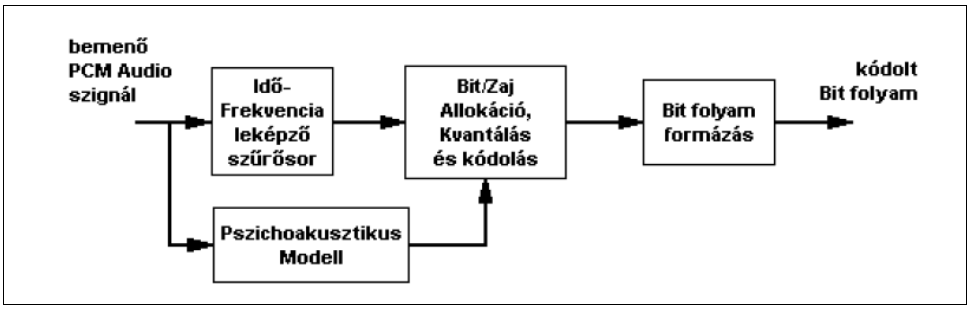
5. fejezet

Fourier-transzformációk

A mintavételezés után a három réteg különbözőképpen használja fel a hang jellemzőit azért, hogy minél jobban tömörítsenek. Értelemszerűen a Layer III eljárás vesz a legtöbb mindent figyelembe.

Minden Layer azonos szűrő eljárást használ gyors Fourier-transzformáció (FFT) segítségével. A Layer III plusz alkalmaz egy ún. módosított diszkrét koszinusz transzformációt (MDCT) a frekvencia felbontás növelésére.

A kódoló egységbe beérkező audio jel egyidejűleg halad át a szűrő soron és a pszichoakusztikus modellen a 4. ábra alapján.



4. ábra. *Az MPEG-1 Audio digitalizáló eljárása.*

A szűrők a beérkező jelet több előre meghatározott terjedelmű frekvencia sávra bontják szét. A szűrőt követő MDCT transzformáció tovább osztja a szűrők kimeneteit a frekvencia tartományban. Például 18 pontos MDCT-t alkalmazva a teljes hanganyagot $32 \cdot 18 = 576$ frekvencia alsávra bontjuk, ezzel is növelve a tömörítés hatásfokát, mivel pontosabban szétválaszthatók a lényeges és a lényegtelen összetevők. Például ha egy sávban nincs hang, a kódoló a sávot nem kódolja. További bitszámcsökkentés érhető el még oly módon is, hogy három mintacsoporton a léptéktényezőket összehasonlítjuk, s csak akkor visszük át mind a hármat, ha közöttük jelentős különbség van. Ellenkező esetben csak kettőt vagy egyet továbbítunk.

Az időbeli átfedések számításához szükség van a különböző összetevők szétválasztására, amelyet az FFT-vel végzünk el oly módon, hogy a jelet az időtartományból a frekvencia tartományba transzformáljuk át. Az FFT azért jó ebből a szempontból, mert meglehetősen nagy pontossággal tudja szétválasztani a frekvenciákat, és így jól tudjuk elemezni az egyes részsávok tartalmát.

A pszichoakusztikus modellben a kódoló a hangjelek frekvencia elosztását elemzi, majd a hangelfedés jelenségét és hallásunk „küszöb” szintjét figyelembe véve törli az ember által nem

hallott részeket a hangállományból.

Fourier francia matematikus mutatta ki, hogy minden folytonos, periodikus jel előállítható különböző frekvenciájú és fázisú szinuszos jelek összegeként. A Fourier-transzformáció lényegében függvények (pl. időjelek) szinuszos összetevőkre bontását végzi. A diszkrét változatát (DFT, diszkrét Fourier-transzformáció) akkor használjuk, ha a transzformálandó függvény korlátos intervallumon értelmezett. A gyakorlatban mindig csak véges intervallumokkal van dolgunk. Az FFT a DFT kiszámításának egy gyorsabb módszere.

Most pedig térjünk át ezen transzformációk definiálására.

Definíciók:

Legyen $f \in L^1_{2\pi}$,vagyis 2π szerint periodikus L^1 -beli függvény. Ekkor az f függvény *Fourier-sora*:

$$Sf(x) = c_0 + \sum_{n=1}^{\infty} [a_n \cos(kx) + b_n \sin(kx)].$$

$$ahol \quad c_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,dt, \quad a_n = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)\,dt \quad és \quad b_n = \frac{1}{\pi}\int_0^{\pi} f(t)\sin(kt)\,dt.$$

Állítás:

Ha az $f(x)$ függvény az x pontban differenciálható, akkor $Sf(x) = f(x)$.

Az $f \in L^1(\mathbb{R})$ függvény *Fourier-transzformálja* a következő: $F(\omega) = \int_{-\infty}^{\infty} f(t)\,e^{-2\pi j \omega t}$.

Az f függvény L^1-belisége gyakorlati gondokat nem okoz, hiszen a hangfeldolgozásban előforduló függvények mindig ilyenek.

Jelöljük T-vel a két mintavétel között eltelt időt. Ekkor az *f(t)* analóg jelből a mintavételezett jel: $f[n] = f(nT), \quad n \in \mathbb{Z}$. Tegyük fel, hogy jelünk T szerint periodikus, és T hosszúságú periódusra N darab minta esik. Ekkor a diszkrét mintavételezett n-edik elem: $f[n] = \frac{n}{N \cdot t_n}, ahol\, t_n = nT$ (t_n az *n*-edik mintavételi időpontig eltelt idő.

Ezek alapján az alábbi összefüggés a diszkrét Fourier-transzformáció (DFT):

$$F(f[n]) = T\sum_{k=0}^{N-1} f[k]\,e^{-j\frac{2\pi}{N}kn}.$$

Az inverz függvénye pedig a következő: $f[n] = \sum_{n=0}^{N-1} F[n]\,e^{j\frac{2\pi}{N}kn}$.

Ennek gyorsított algoritmusa a gyors Fourier-transzformáció (FFT), mely az alábbiak szerint működik. Alapelve, hogy egy DFT-vel kapott *F*[*n*] függvény felírható két DFT-vel kapott

függvénnyel, amelyeknek elemszáma már csak $\frac{N}{2}$. Az egyik függvény $f[n]$ páros, míg a másik függvény $f[n]$ páratlan sorszámú elemeinek DFT transzformáltjából jön létre. Ugyanígy rekurzívan végiggondolva az algoritmus végén kapunk N darab 1 elemű DFT-vel kapott függvényt. Majd ezekből állítjuk elő a 2 eleműeket, majd a 4 eleműeket, és így tovább ugyancsak rekurzívan előállítjuk a transzformáltakat. Nem probléma, ha N nem kettő egy hatványa, hiszen ekkor kiegészítjük megfelelő számú 0-val a transzformáltakat.

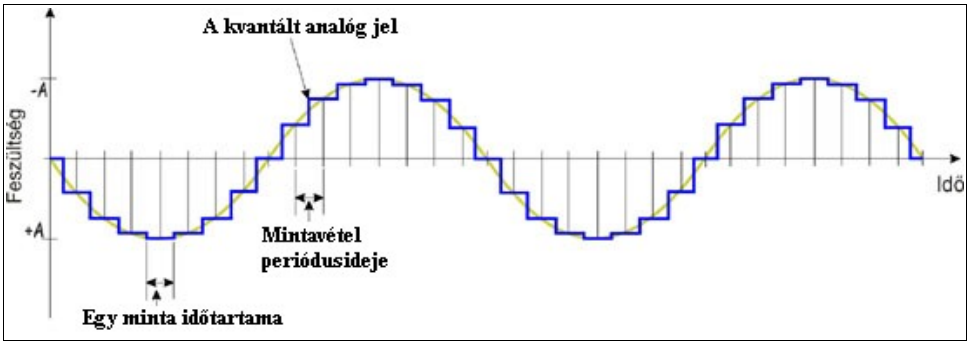
A DCT (diszkrét koszinusz transzformáció) a DFT testvére, de nem szinuszos és koszinuszos jelekre bontja fel a „periodikus” jeleket, hanem koszinuszos komponensekre. Az MDCT pedig egy egy dimenziós DCT.

6. fejezet

Kvantálás

A mintavételezett jel még végtelen sok értéket felvehet, ehhez még nem rendelhető véges kódsorozat. Egy-egy ilyen értéket sokszor csak nagyon sok számjeggyel lehetne nagy pontossággal kifejezni. A kvantálás a végtelen sok lehetséges érték olyan átalakítása, melynek során az értékeket egy-egy kiválasztott értékre kerekítjük, így az értékkészlet elemszámát lecsökkentjük.

Így a minta alapján megállapított érték ennek során adott esetben csökken, más esetben növekszik, azaz a kvantálással az eredeti jelhez zajt adunk. Így minél több kiválasztott érték van, melyre kerekítünk, annál pontosabban lehet visszajátszáskor a digitális jelsorozatból visszaállítani az analóg jelet. A kvantált jel nem hordoz információt arról, hogy ez a hozzáadott zaj mekkora volt. Egy mintavételezett, majd kvantált jelet láthatunk az 5. ábrán.



5. ábra. *Egy mintavételezett, majd kvantált jel.*

<u>Definíciók</u>:

Legyen $X = X_1, X_2, \ldots$ valószínűségi változók sorozata. Az X egydimenziós *kvantáltján* egy véges értékkészletű $Q : \mathbb{R} \to \mathbb{R}$ leképezéssel kapott $Q(X_1), Q(X_2), \ldots$ diszkrét valószínűségi változó sorozatot értünk. A $Q(\cdot)$ függvényt *kvantálónak* nevezzük.

A célunk az, hogy X-et minél pontosabban reprezentáljuk. A kódolás hűségét egy speciális hűségmértékkel, a D(Q) négyzetes torzítással mérjük n hosszú blokkokra:

$D(Q) = \frac{1}{n} E(\sum_{i=1}^{n} [X_i - Q(X_i)]^2)$, ami, mivel X_i-k azonos eloszlásúak, egyenlő a következővel:

$D(Q) = E([X - Q(X)]^2)$, ahol X ugyanolyan eloszlású mint az X_i-k. Természetesen más hűségmértéket is választhattunk volna, de a gyakorlatban ez a legelterjedtebb.

Legyen a Q kvantáló értékkészlete az $\{x_1, x_2, \ldots, x_N\}$ halmaz, ahol az x_i-k $(i = 1, 2, \ldots, N)$ valós számok. Az x_i $(i = 1, 2, \ldots, N)$ számokat *kvantálási szinteknek* nevezzük. Vegyük észre, hogy Q egyértelműen meghatározható az $\{x_1, x_2, \ldots, x_N\}$ kvantálási szintekből és a B_i *kvantálási*

tartományokból, ahol $B_i=\{x\in\mathbb{R}:\ Q(x)=x_i,\ \ i=1,2,...,N\}$. A B_i halmazokról könnyen látszik, hogy diszjunktak és egyesítésük kiadja az egész valós egyenest. Vagyis a kvantáló működése a következőképpen írható le: $Q(x)=x_i,\ \ ha\ x\in B_i,\ \ i=1,2,...,N$.

Tegyük most fel, hogy a kvantálót leíró adatok közül most csak a kvantálási szinteket ismerjük. Az MP3 eljárásban is ez a helyzet, hiszen a megszűrt és mintavételezett jelet egy előre meghatározott értékekhez kerekítjük, ezek a kvantálási szintek. Ekkor az $\{x_1, x_2, ..., x_N\}$ kvantálási szinteket tartalmazó kvantálók közül az a Q kvantáló lesz a legkisebb torzítású, amelyre a B_i $(i=1,2,...,N)$ tartományok a következők lesznek: $B_i=\{x:|x-x_i|\le|x-x_j|,\ j=1,2,...,N\}$. Ezt nevezzük legközelebbi szomszéd feltételnek. Itt a döntési szabályt úgy tehetjük egyértelművé, (vagyis a B_i halmazokat diszjunktakká), hogy ha egy adott x kettő vagy több B_i halmazba tartozna, akkor a legkisebb indexűhöz soroljuk.

Lemma:

Az adott kvantálási szintekkel ez a Q kvantáló a legkisebb négyzetes torzítású.

Bizonyítás:

Ha Q^* egy másik kvantáló ugyanezen $\{x_1, x_2, ..., x_N\}$ kvantálási szintekkel, akkor bármely x-re $Q(x)=x_i$ (tehát $x\in B_i$) és $Q^*(x)=x_j$ valamely $1\le i, j\le N$ indexekre, de ekkor a B_i halmazok definíciója szerint igaz, hogy $|(x-x_i)|\le|(x-x_j)|$, tehát $(x-Q(x))^2\le(x-Q^*(x))^2$ teljesül $\forall x\in\mathbb{R}$ -re, amiből pedig $D(Q)\le D(Q^*)$ következik. □

Ezért a következőkben csak a B_i definíciója szerinti kvantálókkal foglalkozunk.

Az ilyen kvantálók B_i kvantálási tartományai nagyon egyszerűen néznek ki. Tegyük fel most, hogy a kvantálási szintek nagyság szerint rendezve vannak, vagyis $x_1<x_2<...<x_N$. Ekkor a B_i halmazok a következő intervallumok lesznek:

$$B_1=(-\infty, y_1],\ \ B_i=(y_{i-1}, y_i], i=2,3,...,N-1,\ \ B_N=(y_N,\infty),\ \ ahol\ y_i=\frac{x_i+x_{i+1}}{2}, i=1,2,...,N-1.$$

Az optimális kvantálási szint értelemszerűen egy adott tartományhoz annak súlypontja:

$$x_i=\frac{\int_{B_i} x\,f(x)\,dx}{\int_{B_i} f(x)\,dx}.$$

A technikai nehézségek elkerülése végett a további vizsgálataink során feltesszük, hogy a kvantált X valószínűségi változó eloszlása abszolút folytonos f sűrűségfüggvénnyel.

A legegyszerűbb kvantáló az *egyenletes kvantáló*. A Q_N N-szintű egyenletes kvantálót úgy kapjuk, hogy az X lehetséges értékeinek halmazát, a [-A,A] intervallumot, N egyenlő nagyságú intervallumra osztjuk (ezek lesznek most a B_i intervallumok), és a kvantálási szinteket ezen

intervallumok közepén helyezzük el. Formálisan tehát:

$$Q_N(x) = -A + (2\mathrm{i} - i)\frac{A}{N}, \quad ha \quad -A + 2(i-1)\frac{A}{N} < x \le -A + 2\mathrm{i}\frac{A}{N}, \quad i = 1, 2, \ldots, N$$

Bebizonyítható, hogy az N szintű egyenletes kvantáló négyzetes torzítása nagy N esetén közelítőleg $\frac{1}{12}\left(\frac{2\mathrm{A}}{N}\right)^2$. Mivel az N-szintű egyenletes kvantálásnál egy 2A hosszú intervallumot osztottunk fel N részre, egy kvantálási intervallum hossza $q_N = \frac{2\mathrm{A}}{N}$. Tehát azt állítjuk, hogy nagy N-ekre $D(Q_N) \approx \frac{q_N^2}{12}$.

Nem egyenletes kvantáló tervezésénél értelemszerűen azt a technikát alkalmazzuk, hogy az átlagos torzítás csökkentéséhez a nagyobb valószínűségű tartományokban pontosabban közelítjük a bemenetet, míg a kis valószínűségű tartományokban rosszabb közelítést engedünk meg, mint egyenletes kvantálás esetén. Ezt természetesen úgy tehetjük meg, hogy a nagyobb valószínűségű helyeken a kvantálási intervallumokat kisebbnek választjuk. Így a kisebb valószínűségű helyeken nagyobb intervallumhosszokkal dolgozunk.

Tehát egy adott X valószínűségi változóhoz keressük az N szintű, optimális *Q* kvantálót. Feladatunk az $\{x_1, x_2, \ldots, x_n\}$ kvantálási szintek és a *Q(x)* függvény meghatározása úgy, hogy a *D(Q)* négyzetes torzítás minimális legyen.

Bebizonyítható, hogy egy optimális kvantáló kielégíti az alábbi két szükséges feltételt:

1. Legközelebbi szomszéd feltétel:

$$|x - Q(x)| = \min_{1 \le i \le N} |x - x_i| \qquad \forall x \in \mathbb{R}$$

vagyis minden valós *x* kvantált *Q(x)* értéke legalább olyan közel van *x*-hez, mint bármely más kvantálási szint.

2. Súlypont feltétel:

Minden x_j kvantálási szint megegyezik azon X_i minták átlagával, amelyeket erre a szintre kvantálunk ($Q(X_i) = x_j$).

Az első és második feltételt együtt *Lloyd–Max-feltételnek* nevezzük, az ezt kielégítő kvantálót pedig *Lloyd–Max-kvantálónak*. Ha egy kvantáló nem elégíti ki a Lloyd–Max-feltétel bármelyik részét, akkor van olyan kvantáló, amelynek négyzetes torzítása kisebb, tehát egy nem Lloyd–Max-kvantáló nem lehet optimális, de létezik nem optimális Lloyd–Max-kvantáló is.

<u>Példa</u>:

Legyen az X valószínűségi változó az {1,2,3,4} halmazon egyenletes eloszlású. Pontosan háromféle kétszintű Lloyd–Max-kvantálót alkalmazhatunk erre:

$$Q_1(1)=1,\ Q_1(2)=Q_1(3)=Q_1(4)=3$$
$$Q_2(1)=Q_2(2)=Q_2(3)=2,\ Q_2(4)=4$$
$$Q_3(1)=Q_3(2)=1.5,\ Q_3(3)=Q_3(4)=3.5$$

A Q_1 és Q_2 kvantálók négyzetes torzítása egyaránt 0.5, míg Q_3 kvantálóé 0.25. Bár mindhárom Lloyd–Max-kvantáló, csak Q_3 optimális.

Tegyük fel, hogy a kvantálandó X valós valószínűségi változó eloszlása az abszolút folytonos f sűrűségfüggvénnyel adott. Gyakorlati példákban legtöbbször ez az f függvény adott. Ekkor a Lloyd–Max-feltétel az alábbiak szerint alakul:

1. Legközelebbi szomszéd feltétel:

$$y_i=\frac{x_i+x_{i+1}}{2},\quad i=1,2,\dots,N-1,$$

ahol y_{i-1} és y_i annak az intervallumnak a két végpontja, amelyet az x_i szintre kvantálunk.

2. Súlypont feltétel:

$$x_i=\frac{\int_{y_{i-1}}^{y_i} x\,f(x)\,dx}{\int_{y_{i-1}}^{y_i} f(x)\,dx},\quad i=1,2,\dots,N,$$

vagyis minden kvantálási szint a saját kvantálási intervallumának súlypontja.

Fleischer-tétel:

Legyen az *f(x)* sűrűségfüggvény pozitív értékű és logaritmikusan konkáv (vagyis $\log f(x)$ legyen konkáv). Ekkor egyetlen N-szintű Lloyd–Max-kvantáló létezik az *f(x)* függvényre, így ez egyben az egyetlen optimális kvantáló is az *f(x)*-re.

Tehát például ha *f(x)* az egyenletes eloszlás sűrűségfüggvénye [*a,b*]-n, akkor mivel az egyenletes N-szintű kvantáló kielégíti a Lloyd–Max-feltételt az *f(x)*-re, és az egyenletes eloszlás logaritmikusan konkáv, az N-szintű egyenletes kvantáló az egyetlen optimális N-szintű kvantáló az egyenletes eloszlásra.

Az algoritmus:

A kvantálót tehát egyértelműen jellemzik az x_i kvantálási szintek és a $B_i=(y_{i-1},y_i]$ tartományok. Célunk a szintek és az intervallumhatárok javítása lépésről lépésre.

1. Vegyünk fel egy közelítést a kvantálási szintekre.
2. Optimalizáljuk a kvantálót a kvantálási szintek szerint, vagyis határozzuk meg az intervallumhatárokat a legközelebbi szomszéd feltétel kielégítésével.
3. Számítsuk ki, hogy mennyivel csökkent a torzítás, és ha ez egy előre meghatározott, küszöbértéknél kisebb, akkor készen vagyunk.

4. Optimalizáljuk a kvantálót az imént kapott intervallumokhoz, vagyis alkalmazzuk a súlypont feltételt, és folytassuk az algoritmust a 2. ponttól.

A hangtechnikánál azonban figyelembe kell venni, hogy az emberi hallás érzékenysége növekvő görbével írható le. Ez azt jelenti, hogy a halk hangok között sokkal kisebb különbségeket vagyunk képesek meghallani, mint a nagyobb hangerőnél. Emiatt az egyenletes kvantálás esetén a hangos részeknél alig lesz torzítás, mert az emberi fül léptékeihez képest nagyon kis lépcsőket használtak a leképezésre. Az alacsony szinteken viszont durva leképzési hibákat tapasztalhatunk. Ezért használ az MP3 eljárás egy nem egyenletes, vagyis logaritmikus kvantálást.

7. fejezet

Forráskódolás

Definíciók:

Szimbólumok egy véges halmazát *ábécének* nevezzük. Például a magyar ábécé is egy ilyen ábécé. A továbbiakban az X és Y halmazok legyenek ábécék. Az X illetve Y halmaz elemeiből képzett véges sorozatok halmazát (a magyar ábécénél ezek a különböző értelmes vagy értelmetlen szavak halmazai) jelöljük X*-gal illetve Y*-gal. *Nyelvnek* nevezzük ezek egy részhalmazát. Például a magyar ábécé értelmes szavai egy nyelvet, a magyar nyelvet alkotják. Legyen $L \subseteq Y^*$ egy nyelv.

A *kódolás* az X ábécé feletti szövegek (szavak egymásutánja) leképezése Y feletti szövegekre. A *tömörítés* az Y feletti szövegek leképezése Y feletti szövegekre úgy, hogy az új szöveg rövidebb az eredeti szövegnél. A gyakorlatban a két eljárást egymás után, egy függvényként alkalmazzuk. Ezek alapján vezetjük be az alábbi fogalmakat.

Esetünkben az X halmazt *forrásábécének*, elemeit *karaktereknek* vagy *betűknek*, X* elemeit *üzeneteknek* nevezzük. Az X* halmaz elemeit szeretnénk tömöríteni. Az Y halmazt *kódábécének*, Y* elemeit *kódszavaknak* hívjuk. Az Y* halmaz elemeiből épül majd fel a tömörített szöveg.

Az $f : X^* \to Y^*$ mindenütt értelmezett függvény a *kód* vagy teljes nevén a *kódfüggvény*. Ezek alapján a *kódolás* az *f* függvény meghatározása és minden $u \in X^* - ra$ *f(u)* meghatározása. Ha az *f* függvény kölcsönösen egyértelmű (bijektív függvény), *veszteségmentes kódolásról* beszélünk.

Az *f* függvényt *betűnkénti* kódfüggvénynek hívjuk, ha homomorfizmus, azaz $\forall u, v \in X^* - ra$ *f(uv) = f(u)f(v)*. (Itt az *f(u)f(u')* a két kódszó, *u* és *u'* egymás után írását, konkatenációját jelenti.) Értelemszerűen egy betűnkénti kódolást elég X elemein megadni. Mivel mostantól csak betűnkénti kódolást használunk, ezért innen egy $f : X \to Y^*$ függvényről beszélünk.

Az $f : X \to Y^*$ függvény *egyértelműen dekódolható,* ha minden kódszó legfeljebb egy üzenet kódolásával állhat elő, formálisan ha $u \in X^*, v \in X^*, \underline{u} = u_1 u_2 \ldots u_k, \underline{v} = v_1 v_2 \ldots v_m, \underline{u} \neq \underline{v}$, akkor $f(u_1) f(u_2) \ldots f(u_k) \neq f(v_1) f(v_2) \ldots f(v_m)$. Azaz az egyértelműen dekódolható kódok esetén csak egyféleképp bontható fel az üzenet kódszavak konkatenációjára. Az egyértelmű dekódolhatóság több, mint az invertálhatóság.

Az *f* kódot *prefix* kódnak hívjuk, ha a lehetséges kódszavak közül egyik sem folytatása a másiknak, vagyis bármely kódszó elejéből bármekkora szeletet levágva nem kapunk egy másik kódszót. Formálisan ha *x* és *y* két különböző betű kódja, akkor nem létezik olyan *z* bitsorozat, amelyre *xz = y*. Egy prefix kód egyértelműen dekódolható is, mivel nincs olyan kódszó, amely

kezdőszelete lenne a másiknak, így egyértelmű, hogy a kódolt állományban melyik kódszó a következő, amit dekódolni szeretnénk.

Tétel:

Minden egyértelműen dekódolható kódhoz létezik vele ekvivalens prefix kód, vagyis olyan prefix kód, amelynek kódszavai ezen kód kódszavaival azonos hosszúságúak.

Tehát nem vesztünk semmit, ha az egyértelmű dekódolhatóság helyett a speciálisabb, könnyebben kezelhető prefix tulajdonságot követeljük meg.

Példák:

1) $X=\{a,b,c,d\}, \quad Y=\{0,1\}, \quad f(a)=0, \; f(b)=1, \; f(c)=01, \; f(d)=10.$

Ebben az esetben az *f* függvény invertálható, de nem egyértelműen dekódolható, hiszen a 01 kódszót dekódolhatjuk *ab*-nek, illetve *c*-nek is a prefix tulajdonság megsértése miatt. Hasonló a helyzet az 10 kódszóval. Tehát a kód se nem prefix, se nem egyértelműen dekódolható.

2) $X=\{a,b,c\}, \quad Y=\{0,1\}, \quad f(a)=0, \; f(b)=10, \; f(c)=110.$

Ekkor az *abc* üzenetet a 010110 kódszóval kódoljuk. A visszafejtés a prefix tulajdonság miatt nagyon egyszerű, hiszen mindig tudjuk, hogy a következő kódszó hol végződik. Mivel prefix kód, ezért egyértelműen dekódolható is. A prefix kódok gyorsan dekódolhatók.

3) $X=\{a,b,c\}, \quad Y=\{0,1\}, \quad f(a)=0, \; f(b)=01, \; f(c)=001.$

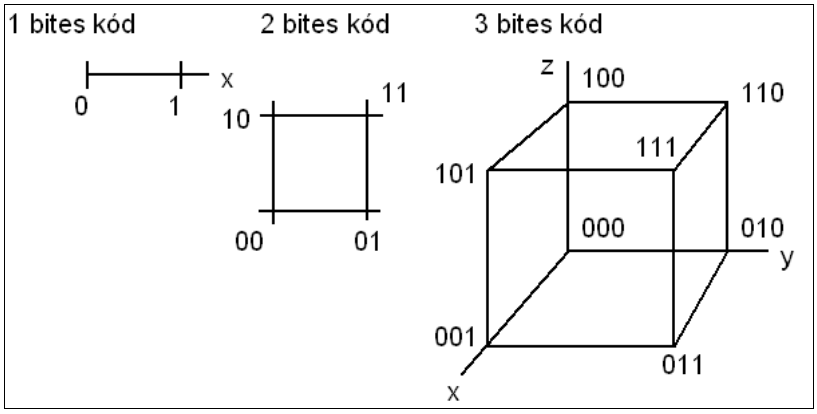
Erről a kódról könnyen látszik, hogy nem prefix, de egyértelműen dekódolható, mivel a 0 karakter egy új kódszó kezdetét jelzi. De a tétel alapján létezik vele ekvivalens prefix kód, például: $f(a)=1, \; f(b)=01, \; f(c)=001$ vagy akár a 2-es példában szereplő kód.

8. fejezet

Huffman-kódolás

A mintavételezés, szűrés és kvantálás után kapunk egy véges jelsorozatot, mely egy véges szimbólumhalmazból veszi fel értékeit. Ezt tömöríti tovább az Mpeg-1 Layer III eljárás.

A számítógép minden karaktert egy bináris jelsorozattal ábrázol. Ha fix hosszúságú kódot szeretnénk használni, akkor *n*-féle karakter kódolásához minimum $k=\lceil \log_2 n \rceil$ bitre van szükségünk, hiszen értelemszerűen 2-féle karakterhez a {0,1} kódokat, 4 féléhez a {00,01,10,11} kódokat, stb. használjuk. Másképpen a fix hosszúságú kódolásnál a kódteret egy k-dimenziós kockával ábrázolhatjuk. Itt minden bitnek egy koordinátát feleltetünk meg, minden koordinátahelyen csak két érték lehetséges, a 0 és az 1. Minden kódnak egy pont felel meg a koordináta-rendszerben.

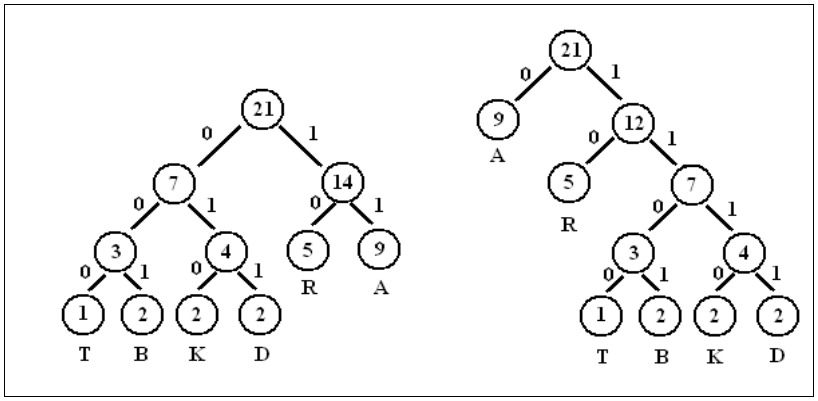


6. ábra. *Kódszavak k dimenzióban*.

Ha tömöríteni szeretnénk, akkor változó kódszóhosszúságú kódot kell használnunk. Mivel az továbbítandó karakterek gyakorisága különböző, ezért tömörítést érhetünk el azzal, ha a gyakrabban előforduló karaktereket rövidebb kóddal ábrázoljuk, mint a ritkábban előfordulókat. Gyakran alkalmazott és hatékony módszer az ilyen tömörítésre a Huffman-kód, amely alkotójáról lett elnevezve, aki már 1952-ben publikálta az eljárást. Ez egy optimális prefix kód. A kód optimális olyan szempontból, hogy legkisebb átlagos kódszóhosszú kód, ha adott a karakterek gyakorisága.

Tehát a probléma a következő: ha adott egy szöveg, amelyben $g_1, g_2, \ldots, g_n$ az $x_1, x_2, \ldots, x_n$ karakterek gyakorisága, akkor keresünk egy olyan fát (Huffman-fa), amelyre a $\sum_{i=1}^{N} g_i d(x_i)$ összeg minimális, ahol egy *x* csúcsra a *d(x)* a gyökértől *x*-ig vezető úton bejárt élek száma. Természetesen az optimális kód nem egyértelmű, hiszen az azonos gyakoriságokhoz rendelt kódszavak felcserélhetők. A továbbiakban optimális prefix kódot keresünk.

A prefix kódokat kódfával szemléltetjük, amivel a dekódolásnál a kódszót könnyen azonosítani tudjuk. A kódfa egy teljes bináris fa, amelynek levelei a kódolandó karakterek. A teljes bináris fa definíciója szerint egy olyan fa, amelyben minden nem levél pontnak két fia van. Egy teljes bináris fának ha pontosan n levele van, akkor pontosan n-1 belső pontja van. A bináris fa éleihez rendeljünk 0-t, ha a bal oldali gyerekre mutat és 1-t, ha a jobb oldali gyerekre mutat. Egy karakter kódját a fa gyökerétől az adott karakterig vezető út adja. Ez alapján az egyes karakterekhez változó hosszúságú, a fix hosszúságú kódnál rövidebb kódot tudunk rendelni. Például a későbbiekben példaként tárgyalt *„abrakadabraarraraktad”* szó egy-egy kódfája fix hosszúságú kóddal és Huffman-kóddal:

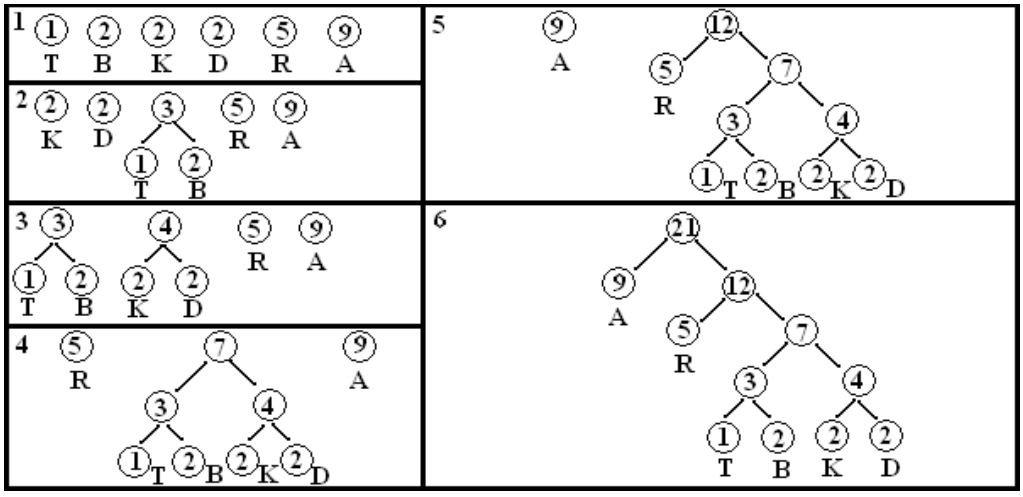


7. ábra. *Az „abrakadabraarraraktad” szó kódfája bal oldalon fix hosszúságú kóddal jobb oldalon Huffman-kóddal.*

Ha már adott egy *f* prefix kód T kódfája, akkor egyszerűen kiszámítható, hogy az adatállomány kódolásához hány bit szükséges. Az X ábécé minden *x* elemére jelölje *g(x)* az *x* elem gyakoriságát az állományban, $d_T(x)$ pedig az *x*-hez tartozó levél mélységét a T fában. Ekkor a kódoláshoz szükséges bitek száma: $B(T)=\sum_{x\in X} g(x)\,d_T(x)$. Ezt az értéket a T fa költségének nevezzük.

A Huffman-kódot létrehozó algoritmus úgy indul, hogy kezdetben meghatározzuk az egyes karakterek gyakoriságát. A karakterek száma legyen *n*. Ebből *n* pontot készítünk (a csúcsokban a karakterek gyakorisága áll), amelyek mindegyikét úgy tekintjük, mint egy egy levélből álló fát. Minden lépésben két fát összevonunk, mégpedig úgy, hogy a két minimális gyakoriságú fát kapcsoljuk össze úgy, hogy létesítünk egy új gyökeret, melynek bal oldali részfája az egyik, a jobb oldali részfája a másik legkisebb gyakoriságú fa, és ennek az új fának a gyakorisága a két minimális gyakoriságú fa gyakoriságának az összege. Az nem lényeges, hogy a két minimális fa közül melyik lesz a bal és melyik a jobb részfa, mert bár a fa különböző lesz, de a kódoláshoz szükséges bitek száma azonos marad. Így *n*-1 összehasonlítás után megkapjuk a keresett kódfát, hiszen a fák száma minden lépésben eggyel csökken. Minden karakter kódja a gyökértől a megfelelő levélig vezető úton lévő élek címkéinek sorozata.

Például az *„abrakadabraarraraktad”* szóban a gyakoriságok: $g(a)=9, g(r)=5, g(b)=2,$ $g(k)=2, g(d)=2, g(t)=1,$ és az algoritmus menete a 7. ábrán követhető nyomon. Ebben az esetben mivel hatféle karakter van, ezért fix hosszúságú kódolásnál 3 bitre van szükség minden egyes karakter kódolásához. Tehát mivel 21 darab karakterünk van, 63 bitnyi helyet foglal. A Huffmann-kódolással viszont a karakterek kódjai a következők lesznek: $A=0$, $R=10$, $T=1100$, $B=1101$, $K=1110$, $D=1111$. Könnyen ellenőrizhető, hogy a kód, mint vártuk, prefix kód, vagyis egyértelműen dekódolható. Ebben az esetben tehát $9*1+5*2+1*4+$ $+2*4+2*4+2*4=47$ bitre van szükségünk. Ez körülbelül 75%-os tömörítési arány, ami már jónak mondható. A Huffman-kódolás 20 – 90%-os tömörítési arányt tud elérni.



8. ábra. *Az „abrakadabraarraraktad” szó Huffman-algoritmussal tömörítve.*

Ha a karakterek száma *n*, akkor *n*-1 összehasonlítást végzünk, és a minimum kiválasztása $O(\log_2 n)$ időt igényel. Tehát a Huffman-algoritmus időigénye $O(n*\log_2 n)$ minden *n* darab karaktert tartalmazó ábécére.

A Huffman-algoritmus helyességét az alábbi 2 tétel bizonyítja.

<u>Tétel</u>:

Legyen X ábécé, *g(x)* az $x \in X$ karakter gyakorisága. Legyen *a* és *b* a két legkisebb gyakoriságú karakter X-ben. Ekkor létezik olyan optimális prefix kód, amely esetén az *a*-hoz és *b*-hez tartozó kódszó hossza megegyezik, és a két kódszó csak az utolsó bitben különbözik.

<u>Bizonyítás</u>:

Az alapötlet az az, hogy vegyünk egy optimális prefix kódot ábrázoló T kódfát és módosítsuk úgy, hogy a fában *a* és *b* a két legmélyebben fekvő testvér pont legyen. Ha ezt meg tudjuk csinálni, akkor a hozzájuk tartozó kódszavak valóban azonos hosszúságúak lesznek és csak az utolsó bitben különböznek.

A T fában a két legmélyebben fekvő pont legyen *u* és *v*. Feltehető, hogy $g(u) \le g(v)$ és

$g(a) \leq g(b)$. Mivel *g(a)* és *g(b)* a két legkisebb gyakoriság, valamint *g(u)* és *g(v)* tetszőleges gyakoriságok, következik, hogy $g(a) \leq g(u)$ és $g(b) \leq g(v)$. Ha felcseréljük a T fában az *u* és az *a* pontok helyét, ezzel kapjuk a T' fát, majd ebből a fából, felcserélve a *v* és a *b* pontok helyét, kapjuk a T" fát. Ekkor a T és a T' fák költségének különbsége:

$$B(T) - B(T') = \sum_{y \in Y} g(v) d_T(v) - \sum_{y \in Y} g(v) d_{T'}(v) =$$
$$= g(a) d_T(a) + g(u) d_T(u) - g(a) d_{T'}(a) - g(u) d_{T'}(u) =$$
$$= g(a) d_T(a) + g(u) d_T(u) - g(a) d\,T'(u) - g(u) d\,T'(a) =$$
$$= (g(u) - g(a))\,(d_T(u) - d_T(a)) \geq 0$$

Az egyenlőtlenség azért teljesül, mert $g(u) - g(a)$ és $d_T(u) - d_T(a)$ nemnegatív. Pontosabban, $g(u) - g(a)$ nemnegatív, mert *a* egy legkisebb gyakoriságú karakter, és $d_T(u) - d_T(a)$ azért nemnegatív, mert *u* maximális mélységű a T fában. Hasonlóan bizonyítható, hogy *v* és *b* felcserélése esetén sem növekszik a költség, így $B(T) - B(T'')$ nemnegatív. Tehát $B(T'') \leq B(T)$, és mivel T optimális így $B(T) \leq B(T'')$, tehát $B(T'') = B(T)$. Tehát T" olyan optimális fa, amelyben *a* és *b* maximális mélységű testvér pontok, amiből a tétel állítása következik. □

A tételből következik, hogy az optimális fa felépítése, az általánosság megszorítása nélkül kezdhető a mohó választással, azaz a két legkisebb gyakoriságú elem összevonásával. Miért tekinthető ez mohó választásnak? Azért, mert tekinthetjük a két összevont elem gyakoriságának összegét egy összevonás költségeként. Észrevehető, hogy a felépített fa teljes költsége megegyezik az összevonási lépések költségeinek összegével.

Tétel:

Legyen T olyan teljes bináris fa, amely az X ábécének a g(x) gyakoriságok melletti optimális prefix kódját ábrázolja. Tekintsük két olyan *a* és *b* pontját a T fának, amelyek testvérek, és legyen *c* ezek szülője. Ekkor, ha *c*-t úgy tekintjük, mint olyan karaktert, amelynek gyakorisága $g(c) = g(a) + g(b)$, akkor a $T' = T - \{a, b\}$ fa az $X' = X - \{a, b\} \cup \{c\}$ ábécé optimális prefix kódját ábrázolja.

Bizonyítás:

Először megmutatjuk, hogy a T fa B(T) költsége kifejezhető a T' fa B(T') költségével. Minden $x \in X - \{a, b\}$ esetén $d_T(x) = d_{T'}(x)$, így $g(x) d_T(x) = g(x) d_{T'}(x)$. Mivel $d_T(a) =$ $= d_{T'}(b) + 1$, így azt kapjuk, hogy $g(a) d_T(a) + g(b) d_T(b) = (g(a) + g(b))(d_{T'}(c) + 1) =$ $= g(c) d_{T'}(c) + g(a) + g(b)$, amiből következik, hogy $B(T) = B(T') + g(a) + g(b)$.

Ha T' nem optimális prefix kódfa az X' ábécére, akkor létezik olyan T" fa, amelynek levelei X'-beli karakterek és $B(T'') < B(T')$. Mivel *c* X'-beli karakternek tekinthető, így előfordul a T"

fában levélként. Ha a T'' fában *c* két gyerekeként hozzávesszük az *a* és *b* pontokat, akkor olyan prefix kódot kapunk, amelynek költsége $B(T'')+g(a)+g(b) < B(T)$, ami ellentmond a T bináris fa optimalitásának. Tehát T' optimális kell legyen az X' ábécére. □

E két tétel alapján tehát a Huffmann eljárás optimális prefix kódot állít elő.

A Huffman-algoritmus két menetben tömörít. Az első menet során meghatározza a karakterek eloszlását, majd a már meglévő gyakoriságokat felhasználva lekódolja a tömörítendő adatsort. Az eljárás kifejezett előnye a sebessége, amely elég jónak mondható. De egyik hátránya, hogy a tömörítendő adatot kétszer kell végigolvasni. Ez a hátrány kiküszöbölhető úgy, hogy az algoritmus blokkokra bontja a tömörítendő adatot, majd az első menetben nemcsak a gyakoriságokat olvassa ki, hanem a blokkot betölti a memóriába és előkészíti azt a második menetben végrehajtandó kódolásra. Másik hátrányos tulajdonsága az, hogy a tömörített adat mellé (és lehetőleg elé, hogy a dekódolónak legyen mi alapján kitömörítenie) le kell tárolni a gyakoriságokat is (vagyis át kell küldeni a kódfát).

9. fejezet

Adaptív Huffman-kódolás

Az előző fejezetben tárgyalt Huffman-kódolást alkalmazza az MPEG-1 Layer III kódolás, de érdekes továbbfejlesztéseként megemlíteném az adaptív Huffman-algoritmust is. Az eredeti Huffman-kódolás egyik hátránya tehát az, hogy el kell küldeni a gyakoriságokat is. Az adaptív Huffman-algoritmus nagy előnye, hogy nem kell elküldeni a gyakoriságokat, mivel egy forrásbetűt az előző forrásbetűk előfordulásai alapján kódolunk, s ezzel együtt lépésenként változik maga a kód is, így a gyakoriságokat a kitömörítés közben hasonlóan elő lehet állítani. Tehát az aktuális forrásbetű kódolását egy, az előzőleg feldolgozott forrásbetűkre optimális kóddal hajtjuk végre. Ezt az eljárást nevezzük adaptív Huffman-kódolásnak.

Az adaptív Huffman-kódolást Faller és Gallager találta ki és Knuth továbbfejlesztette. A legújabb algoritmus pedig Vittertől származik. Sok algoritmus létezik, de én csak az általuk kitalált algoritmusokat elemzem: az FGK és Vitter-algoritmust.

Mindkét algoritmushoz szükségünk lesz a *testvérpár tulajdonság* fogalmára. Amennyiben a fa testvér (vagyis közös szülővel rendelkező) pontpárjait a gyökértől a levelek felé haladva szintenként fel tudjuk sorolni súlyuk szerint nemnövekvő sorrendben, a fa rendelkezik a testvérpár tulajdonsággal.

Tétel (Gallager):

Egy Huffman-fa akkor és csak akkor optimális, ha rendelkezik a testvérpár tulajdonsággal.

Az **FGK algoritmus** tehát a következő:

Mivel a pontos gyakoriságok itt nem ismertek, kiindulásként felépítünk egy olyan Huffman-fát, amelyben minden forrásszimbólum azonos, 0 gyakorisággal szerepel. Elküldjük a dekódolónak sorrendben (pl. a gyökerektől a levelek felé, szintenként, balról jobbra) a lehetséges karaktereket, amelyből az szintén felépíti a Huffman-fát. Amikor beolvassuk a *k*-adik forrásszimbólumot, akkor a korábban beolvasott *k*-1 betű feldolgozásával kialakult, lokálisan optimális kódfával kódoljuk, majd eggyel növeljük a karakter gyakoriságszámlálóját. Majd aktualizáljuk a fát, vagyis megvizsgáljuk, hogy fennáll-e még a testvérpár tulajdonság. Amennyiben nem, helyreállítjuk, így biztosítva az optimalitást. Ezeket a lépéseket a dekóder is elvégzi, tehát a következő kódbetű dekódolásakor ugyanazt a fát fogja használni, mint amelyet a kódoló használt annak előállításakor.

A következő két szabály használható a fa aktualizálása során a testvérpár tulajdonság helyreállítására:

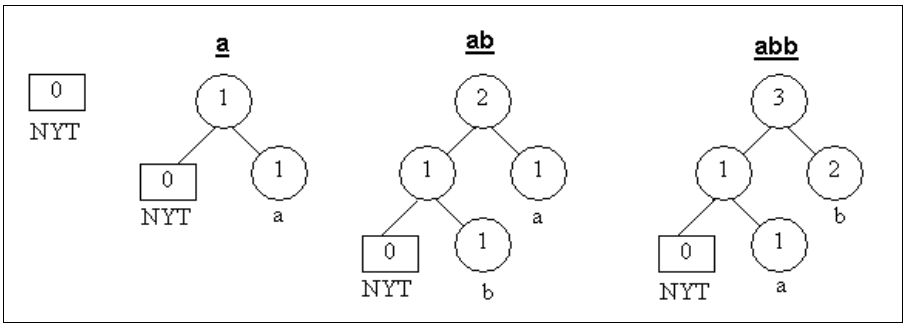
1. Két, azonos súlyú csúcs a hozzá tartozó részfákkal együtt megcserélhető. Ez természetesen nem befolyásolja a súlyokat.
2. Két levél a súlyukkal együtt megcserélhető.

A szabályok kombinálásával átalakítható egy nem megfelelő kódfa egy olyanná, amely teljesíti a testvérpár tulajdonságot.

Ügyelni kell arra, hogy hosszú bemenet esetén előfordulhat a szimbólumok gyakoriságszámlálójának túlcsordulása. Ezt megfelelő technikával orvosolni lehet, például, ha a karakterek gyakoriságát lefelezzük.

A **Vitter-algoritmus** pedig a következő:

Kezdetben a kódfa egy levélből áll, ez a 0 pont. Ez egy speciális pont, ez reprezentálja a fel nem használt szimbólumokat. Minden karakter beolvasásakor a karakter gyakoriságát eggyel meg kell növelni, és biztosítani kell a kódfa testvérpár tulajdonságát. Amikor a t-edik karaktert már továbbítottuk, és k-féle karakter már benne van a kódfában, és ha a t-edik nem az utolsó karakter, akkor a Huffmann-fának ekkor k+1 levele van, k darab a k-féle karakternek, és 1 darab a 0 pontnak. Ha a (t+1)-edik karakter már benne van az aktuális Huffmann-fában, akkor az algoritmus továbbítja a (t+1)-edik karakter aktuális kódját, majd növeli a számlálót, azután helyreállítja a testvérpár tulajdonságot. Ha egy olyan karakter érkezik, amely nincs benne a Huffmann-fában, akkor a (t+1)-edik karakter kódjaként a 0 pont kódját továbbítja, majd a 0 pontot felosztja két levélre, az új (t+1)-edik karakterre, és a 0 pontra. Majd a testvérpár tulajdonságot ismét helyreálltjuk. A 9. ábrán egy példa látható.



9. ábra. *A Vitter-algoritmus működése „abb” bemenetre.*

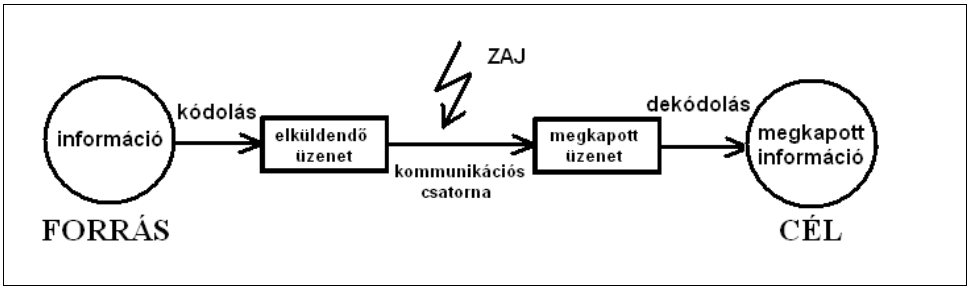
Az FGK algoritmus előnye, hogy könnyű végrehajtani. A Vitter-algoritmus előnye, hogy a fa magasságát a minimumon tartja, de elég lassú a folyamat, nehéz megvalósítani.

Az adaptív Huffman-kódolás hátránya a tömörítés sebessége, ugyanis a karakterekhez rendelt bitsorozatot minden lekódolt karakter után újra kell számolni, amely viszonylag időigényes folyamat.

10. fejezet

Hibajelző és hibajavító kódolás

Ha már be van tömörítve az állományunk, akkor egy csatornán keresztül tovább szeretnénk küldeni a 10. ábra alapján. Ekkor mivel a csatorna zajos, sérül a fájl. A továbbítás a zajos csatornán tehát szükségszerűen együtt jár különböző átviteli hibák megjelenésével. A hibajavító kódolás az üzenet zajos csatornán való áthaladása után is képes bizonyos sérülések javítására, azaz az átvitt információ (bizonyos korlátok mellett) helyreállítható belőle A célunk tehát az, hogy a címzett képes legyen visszaállítani az eredeti üzenetet. Az ilyen hibák kiküszöbölésére hosszabb üzenetet kell elküldenünk. Minél hosszabb üzeneteket küldünk, annál több hibát tudunk javítani, de annál több hiba is keletkezhet.



10. ábra. *Csatornán keresztüli információtovábbítás modellje.*

Tehát a cél a következő: minél rövidebb kódokkal minél több hibát tudjunk javítani.

Definíciók:

Legyen Q egy ábécé, amelynek elemszáma q. A Q elemeiből készített k hosszú sorozatokat *szavaknak* nevezzük. A belőlük felépülő halmazt Q^k-val jelöljük. A *kódolás* egy $f: Q^k \to Q^n$ injektív függvény. Legyen $f(Q^k) = C \subseteq Q^n$ az f függvény értékkészlete, amit *kódnak* nevezünk, és amelynek elemei a *kódszavak.*

Tehát az eredeti üzenet egy k dimenziós vektor, a kódolt üzenet pedig egy n dimenziós vektor. Fontos, hogy $n > k$. A kód megadásakor sokszor csak a C halmaz szerepel, az f kódoló függvény nem. Legtöbbször $Q = \{0,1\}$, vagyis az üzenet egy n hosszúságú bináris sorozat, a C halmazban pedig értelemszerűen azok az n hosszú 0-kból és 1-esekből felépülő vektorok találhatók, amelyeket az üzenetküldés során hibátlan szavaknak minősítünk. Így itt a hiba tulajdonképpen egy vagy több bit invertálódását jelenti.

A hibák kiküszöbölésének legegyszerűbb, de korántsem leghatékonyabb módja az, mikor az elküldendő üzenetet háromszor juttatjuk át a vevőnek (például minden bitet ismételünk háromszor). Ekkor ha a három szomszédos betűből maximum egy hibás, akkor az eredeti üzenet visszakapható,

ha a hibás szavakhoz a hozzájuk legközelebb álló kódszót rendeljük. Itt formálisan $Q=\{0,1\}$, $k=1$, $n=3$, $f(x)=xxx$ $\forall x\in Q^k$ -ra. Ekkor $C=\{000,111\}\subseteq Q^3$. Ha érkezik egy $u\in Q^3$ szó ami nem kódszó, akkor hiba történt. Ekkor keresünk egy kódszót, amitől az *u* hibás szó a legkevesebb helyen tér el.

Definíció:

Legyen $t\geq 1$ egész szám. A $C\subseteq Q^n$ kódot *t-hibajelzőnek* nevezzük, ha bármely kódszót legfeljebb *t* helyen megváltoztatva az eredmény nem lehet kódszó. A C kódot *t-hibajavítónak* hívjuk, ha bárhogy veszünk két $v\neq w$ kódszót, és ha *v*-t és *w*-t is legfeljebb *t* helyen megváltoztatjuk, (ezek a helyek mások lehetnek *v*, mint *w* esetében) akkor nem kaphatjuk Q^n-nek ugyanazt az elemét.

Például a háromszorozós példában *C* 2-hibajelző kód, hiszen 2 bitet megváltoztatva még nem kapunk egy másik kódszót, de már hármat megváltoztatva igen. Ez a kód 1-hibajavító, hiszen ha mindkét kódszót megváltoztatjuk egy helyen, akkor még nem kapunk azonos Q^3-beli elemeket, de ha már két helyen változtatjuk meg, akkor már kaphatunk.

Vagyis hibajelzés során a vevőtől a hibázás tényét kérdezzük, hogy egy adott *u* szó kódszó-e, azaz *u* eleme-e C-nek. Hibajavítás esetén pedig azt kérdezzük, hogy ha *t* a hibák száma, akkor mi biztosítja, hogy a *v* vett szóból az *u* küldött szó egyértelműen visszaállítható legyen, azaz minden más *w* kódszóra $d(v,w)>d(v,u)$ teljesüljön. Vagyis egy olyan *u* kódszót keresünk, amely a legközelebb van *v*-hez.

Definíció:

Az *u* szó *w(u) súlya* azon koordináták száma, amelyek 0-tól különböznek. A $v,w\in Q^n$ vektorok *Hamming-távolságán* azoknak a koordinátáknak a számát értjük, ahol a két szó eltér. Jele: *d*(*v*,*w*). Ez lesz az átküldött üzenet hibáinak a száma. A $C\subseteq Q^n$ kód *minimális távolsága* a különböző kódszavak Hamming-távolságainak a minimuma: $d_{min}=min\{d(v,w):v,w\in C, v\neq w\}$ Jele: $d_{min}(C)$. Vagyis ez az érték a két legközelebbi kódszó távolsága.

A példánkban $d(000,111) = 3$, és mivel csak ez a két kódszó van, ezért $d_{min} = 3$.

Könnyen bizonyítható, hogy *d* metrikát alkot, vagyis:

$$d(x,y)>=0 \text{ és } d(x,y) = 0 \iff x = y$$

$$d(x,y) = d(y,x)$$

$$d(x,z) <= d(x,y) + d(y,z)$$

Egy *v* vett szó esetén akkor tudjuk a hibázást észrevenni, ha *u* nem kódszó, ami bizonyos akkor, ha az *u* küldött kódszó esetén $d_{min} > d(u,v)$.

A dekódolás ezek szerint a Hamming-távolságok alapján történik: egy adott $v\in Q^n$ szó

dekódoltja az az $u \in C$ kódszó, amely a Hamming-távolságra nézve a legközelebb van hozzá. Ahol egy szó egy-egy (különböző) kódszótól való minimális Hamming-távolsága megegyezik, ott önkényesen döntünk a dekódoltról.

Ezek alapján a célunk tehát a egy kódkészleten belül a minimális távolság (d_{min}) maximalizálása, hogy minél több hibát tudjunk észrevenni, jobb esetben javítani is.

Sokkal átfogóbb hibajelző és hibajavító módszert tesznek lehetővé a véges testek és a felettük értelmezett polinomok bizonyos tulajdonságainak kihasználása. Ezért szükségünk lesz még a következőkre is:

Definíciók:

Ha *Q* egy véges test és *C* altere a *Q* feletti Q^n vektortérnek, akkor *C*-t *lineáris kódnak* nevezzük. A $C \subseteq Q^n$ kódot *ciklikusnak* nevezzük, ha minden $a_1a_2...a_n$ kódszóra az $a_2...a_na_1$ szó szintén kódszó. Vagyis azok a kódok, amelyeknél a kódszó ciklikus eltolása ismét kódszót eredményez.

Lineáris kód esetén $d(u, v) = d(u\text{-}v, 0) = w(u\text{-}v)$.

Az $\underline{u} = u_1u_2...u_k$ szavakat azonosítsuk az $u_1x^{k-1} + u_2x^{k-2} + ... + u_{k-1}x + u_k$ polinommal. Tehát a kódokat alkotó biteket egy polinom együtthatóiként kezeljük. Legyen $g \in Q[x]$, *n-k* fokú. Ekkor a $C = \{g(x)u(x): \ deg(u) \leq k\}$ kódot a *g* generátorú *polinomkódnak* nevezzük. Ez lineáris kód, a kódolás pedig a generátorpolinommal való szorzás. A kommunikációban résztvevőknek meg kell egyezniük ebben a közös generátorpolinomban. Fontos, hogy ennek a kitüntetett polinom kódjának a legelső és legutolsó bitjének 1-esnek kell lennie, mivel irreducibilis polinomokkal szerünk dolgozni. És természetesen a továbbítandó üzenetnek hosszabbnak kell lennie, mint a generátorpolinom.

Például $Q = \{0, 1\}$ és $g(x) = x^2 + x + 1$. Ez *k=1*, *n=3* esetén a háromszorozó kódolás, hiszen $g(x)u = ux^2 + ux + u \Leftrightarrow uuu$. A Q kételemű test esetében például $P(x) = x^5 + x^2 + x + 1$ alakban írható fel a 100111 szó.

11. fejezet

CRC kódolás

Miután betömörítettük az állományunkat, gyakran egy zajos csatornán szeretnénk átküldeni. Erre a leggyakrabban használt hibaellenőrző mód a CRC vagyis a Ciklikus Redundancia Ellenőrzés. Ez az adatblokkot egy polinom együtthatóinak tekinti, amelyet eloszt egy előre meghatározott, állandó polinommal.

A kódolás algoritmusa a következő:

Az elküldendő bináris kódot írjuk fel polinom alakban az előző fejezetben tárgyaltak mintájára. Legyen ez a polinom $M(x)$. Legyen az előre megadott generátorpolinom $P(x)$, melynek fokszáma n. A $P(x)$ polinom irreducibilis polinom kell, hogy legyen. Ez azt jelenti, hogy nem bontható fel két, nála alacsonyabb fokú polinom szorzatára.

Az adó oldali algoritmus:

A küldendő információt egészítsük ki a generátorpolinom fokszámának (ami n) megfelelő számú zérussal. Ezt algebrailag egy x^n-nel való szorzással kapjuk: $M(x)\cdot x^n$. Ezután az így kapott polinomot (az n darab nullával megtoldott bináris kódot) a generátorpolinommal, $P(x)$-el elosztjuk:

$$\frac{M(x)\cdot x^n}{P}(x)=Q(x)+\frac{R(x)}{P(x)}.$$

Az így kapott $R(x)$ maradékpolinomot az eltolt $M(x)\cdot x^n$ végén lévő zérusok helyére beillesztjük. Algebrailag ez összeadást jelent:

$$T(x)=M(x)\cdot x^n+R(x).$$

A kapott $T(x)$ lesz az üzenet polinom, ez kerül a csatornára.

A vevő oldali algoritmus:

A beérkezett $T'(x)$ polinomot osszuk el a generátorpolinommal:

$$\frac{T'(x)}{P(x)}=\frac{M(x)\cdot x^n+R(x)}{P(x)}=\frac{M(x)\cdot x^n}{P(x)}+\frac{R(x)}{P(x)}=Q(x)+\frac{R(x)}{P(x)}+\frac{R'(x)}{P(x)}=Q(x)+\frac{R(x)+R'(x)}{P(x)}.$$

Ahol $R(x)$ az adó oldalon, $R'(x)$ a vevő oldalon számított maradékpolinom. Végül, a modulo 2 összeadás szabályai szerint, ha $R=R'$, vagyis az adó és a vevő oldalon képzett maradékok egyenlőek, akkor $\frac{R(x)+R'(x)}{P(x)}=0$.

Tehát ha az átvitel során nem sérült meg az üzenet ($T(x) = T'(x)$), akkor a teljes algoritmus végén zérus maradékot kapunk:

$$\frac{T'(x)}{P(x)}=\frac{T(x)}{P(x)}=Q(x).$$

Ha az üzenet bármely része (akár az információs rész, akár a maradék rész) megsérül (ami $R\neq R'$ -ben nyilvánul meg), a vevő oldali maradék nem fog egyezni az adó oldali maradékkal. Így modulo 2 összegük sem lesz zérus. Ezzel jelzi, hogy az átvitel során hiba keletkezett.

Példa:

Legyen a továbbítandó üzenet: 1101, az ennek megfelelő polinom alak: $M(x)=x^3+x^2+1$. Legyen a generátorpolinom: $P(x)=x^3+x+1$ (bináris képe: 1011). A generátorpolinom fokszáma három, ezért $M(x)\cdot x^3=x^6+x^5+x^3$ (bináris képe: 1101000), és ezt kell osztani a generátorpolinommal. Így megkapjuk: *R(x) = 1*, melynek bináris képe 001. Ekkor az elküldött polinom a következő: $T(x)=M(x)\cdot x^n+R(x)=x^6+x^5+x^3+1$ (bináris képe: 1101001).

Először tegyük fel, hogy hibátlanul érkezett az üzenet. Ekkor a vétel oldali algoritmust elvégezve a maradék 0. Most pedig legyen egy hiba: a második legnagyobb helyiértéken 1-es helyett 0. Ekkor $T'(x)\neq T(x)$ miatt azt várjuk, hogy $R(x)\neq 0$ legyen. Az algoritmust elvégezve $R(x)=111-et$ kapunk, tehát a hibát mint vártuk, jelezte. Tovább folytatva így a gondolatsort, megkapható, hogy 2 hibát is tud jelezni.

A CRC kódolás különböző fajtái különböző generátorpolinomokat részesítenek előnyben. Például a teljesség igénye nélkül kiemelve néhány típust:

CRC-12: $x^{12}+x^{11}+x^2+x+1$

CRC-16: $x^{16}+x^{15}+x^2+1$

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

CRC-CCITT: $x^{16}+x^{12}+x^5+1$

Előnyük, hogy nagyon jó hibajelző kódolások. Például a 16 bites CRC kódok képesek felismerni minden 1 bites, 2 bites, minden páratlan hibás bitet tartalmazó hibát, minden 16 vagy kevesebb bitnyi csoportos hibát, a 17 bites csoportos hibák 99,97%-át, a 18 vagy magasabb bitszámú hibák 99,998%-át. Fontos, hogy egy hibajelző vagy hibajavító kódolás csoportos hibákat is tudjon kezelni, mivel a hibák általában így helyezkednek el.

Hátrányuk, hogy csak hibajelzésre alkalmasak.

Az MP3 eljárásban el lehet helyezni egy CRC hibajelzést, ami az állomány épségének ellenőrzésére szolgál. Ez fontos lépés a dekódolás előtt, hiszen mielőtt a lejátszóprogramunk elindítaná a tömörített fájlt, ezzel már képes ellenőrzi, hogy hibás-e.

Irodalomjegyzék

[1] L. Györfi, S. Győri, I. Vajda, *Információ- és kódelmélet*, Typotex kiadó (2005)

[2] R. Csercsa, A. Magony: *A/D átalakítás* (2004), vizsgadolgozat

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest, *Algoritmusok*, Műszaki Könyvkiadó (2001), 290 – 296.

[4] E. Kiss, *Bevezetés az algebrába*, Typotex kiadó (2007), 559 – 580.

[5] R. Freud, *Lineáris algebra*, ELTE Eötvös Kiadó (2007), 287 – 309.

[6] http://itl7.elte.hu/html/jelfel/jelfeld.htm

[7] http://e-oktat.pmmf.hu/kepeshang_2_fejezet

[8] http://www.ics.uci.edu/~dan/pubs/DC-Sec4.html

[9] http://www.cs.duke.edu/csed/curious/compression/adaptivehuff.html

[10] http://bartosz.com/Science/CrcMath.html

[11] http://www.hackersdelight.org/crc.pdf