9. KÓDOLÁS

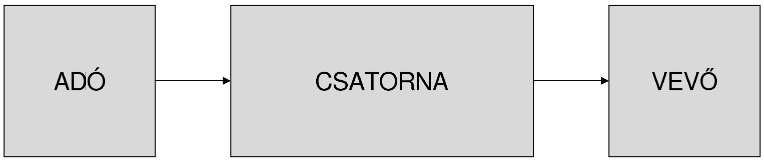
Lángné-Gonda: O.K.; Gonda kiegészítés; Gonda bevmat3; Gonda kódoláselmélet; Jákó; Salomon; Iványi Informatikai algoritmusok1; Cormen–Leiserson–Rivest: O.K.; Rónyai–Ivanyos–Szabó: O.K.; Demetrovics–Denev–Pavlov; Knuth TAOCP; Birkhoff–Bartee.

Ebben a fejezetben a kódolás alapjaival foglalkozunk. Elsőként megnézzük, hogy hogyan történik az adatok átvitele. Ezután két kérdéssel foglalkozunk részletesebben, a gazdaságos, valamint a hibakorlátozó kódolással.

9.1. Kommunikáció és kódolás

Lángné-Gonda: O.K.; Gonda kiegészítés; Gonda bevmat3; Gonda kódoláselmélet; Jákó; Salomon; Iványi Informatikai algoritmusok1; Cormen–Leiserson–Rivest: O.K.; Rónyai–Ivanyos–Szabó: O.K.; Demetrovics–Denev–Pavlov; Knuth TAOCP; Birkhoff–Bartee.

A *kommunikáció* során információt hordozó *adat*okat viszünk át egy *csatorná*n keresztül az *információforrás*tól, az *adó*tól az információ címzettjéhez, a *vevő*höz. Ezt nagy vonalakban az 9.1. ábra mutatja.



9.1. ábra

Az ábra szerint az információ átvitele térben történik. Valójában minden információátvitel térben és időben történik, és egyes esetekben az egyik, míg más esetekben a másik dimenzió a domináns. Ha telefonálunk, akkor az idő nem lényeges, ugyanakkor az információ lemezre való rögzítése, és egy későbbi időpontban való visszaolvasása esetén a helyváltozás kevésbé fontos. A továbbiakban általában úgy beszélünk az adatátvitelről, mintha az térben történne, de ebbe mindig beleértjük az időbeli átvitelt is.

A modellel kapcsolatban felmerül néhány kérdés:

(1) mi az információ, és hogyan mérhető az információ;

(2) hogyan történik az információ átvitele;

(3) milyen kapcsolat van az elküldött és a vett adat között.

9.1.1. Információ, bit, entrópia. Az információról mindenkinek van valamilyen intuitív fogalma. A nagyszámú definíció közül itt azt említjük meg, amely szerint az *információ* új ismeret. Az információt Shannon nyomán az általa megszüntetett bizonytalansággal mérjük. Tegyük fel, hogy egy információforrás nagy számú, összesen n üzenetet bocsájt ki. Az összes ténylegesen előforduló különböző üzenet legyen $a_1, a_2, \ldots, a_m$ ($m \in \mathbb{N}^+$). Ha az a_i üzenet k_i-szer fordul elő, akkor azt mondjuk, hogy *gyakorisága* k_i, *relatív gyakorisága* pedig $p_i = k_i/n$. A $p_1, p_2, \ldots, p_m$ szám-m-est az üzenetek *eloszlásának* nevezzük. Nyilván $\sum_{i=1}^m p_i = 1$. Az a_i üzenet *egyedi információtartalma* $I_i = -\log_r p_i$, ahol r egy 1-nél nagyobb valós szám. A logaritmus alapja az információ egységét határozza meg. Amennyiben az alap 2, akkor az információ egysége a *bit*, és ilyenkor $\log_r p_i$ helyett egyszerűen $\log p_i$-t írunk. Többnyire nem az egyes üzenetek egyedi információtartalma, hanem az üzenetforrás által kibocsátott üzenetek $H_r(p_1, \ldots, p_m) = -\sum_{i=1}^m p_i \log_r p_i$ *átlagos információtartalma* érdekes, ez a forrás *entrópiája*, amely csak az üzenetek eloszlásától függ.

Általánosabban, egy m tagú *eloszlás* egy pozitív valós számokból álló $p_1, p_2, \ldots, p_m$ sorozat, amelyre $\sum_{i=1}^m p_i = 1$. Az eloszlás *entrópiáját* a $H_r(p_1, \ldots, p_m) = -\sum_{i=1}^m p_i \log_r p_i$ összefüggéssel értelmezzük.

°* **9.1.2. Segédtétel.** *Legyen $p_1, \ldots, p_m$ egy eloszlás. Egy $I \subset \mathbb{R}$ intervallumon szigorúan konvex $f : I \to \mathbb{R}$ függvényre*

$$f\big(\textstyle\sum_{i=1}^m p_i q_i\big) \le \displaystyle\sum_{i=1}^m p_i f(q_i), \quad \text{ha} \quad q_1, \ldots, q_m \in I,$$

és egyenlőség pontosan akkor teljesül, ha $q_1 = q_2 = \cdots = q_m$.

Bizonyítás. Ha $m = 1$, akkor nincs mit bizonyítani. Az $m = 2$ esetben az állítás a szigorú konvexség definíciójából következik. Indukcióval, legyen $p = \sum_{i=1}^m p_i$. Ekkor

$$\begin{aligned}\sum_{i=1}^{m+1} p_i f(q_i) &= p \sum_{i=1}^m \frac{p_i}{p} f(q_i) + p_{m+1} f(q_{m+1}) \\ &\ge p f\big(\textstyle\sum_{i=1}^m p_i q_i / p\big) + p_{m+1} f(q_{m+1}) \\ &\ge f\big(\textstyle\sum_{i=1}^{m+1} p_i q_i\big),\end{aligned}$$

és egyenlőség pontosan akkor teljesül, ha $q_1 = q_2 = \cdots = q_m$ és $q_{m+1} = \sum_{i=1}^{m} p_i q_i / p = q_m$. $\square$

9.1.3. Tétel. *Bármilyen eloszláshoz tartozó entrópiára $H_r(p_1, p_2, \ldots, p_m) \leq \log_r m$, és egyenlőség pontosan akkor teljesül, ha $p_1 = p_2 = \cdots = p_m = 1/m$.*

Az átlagos információtartalom maximuma tehát $\log m$ bit.

* **Bizonyítás.** Mivel $-\log_r$ szigorúan konvex függvény,

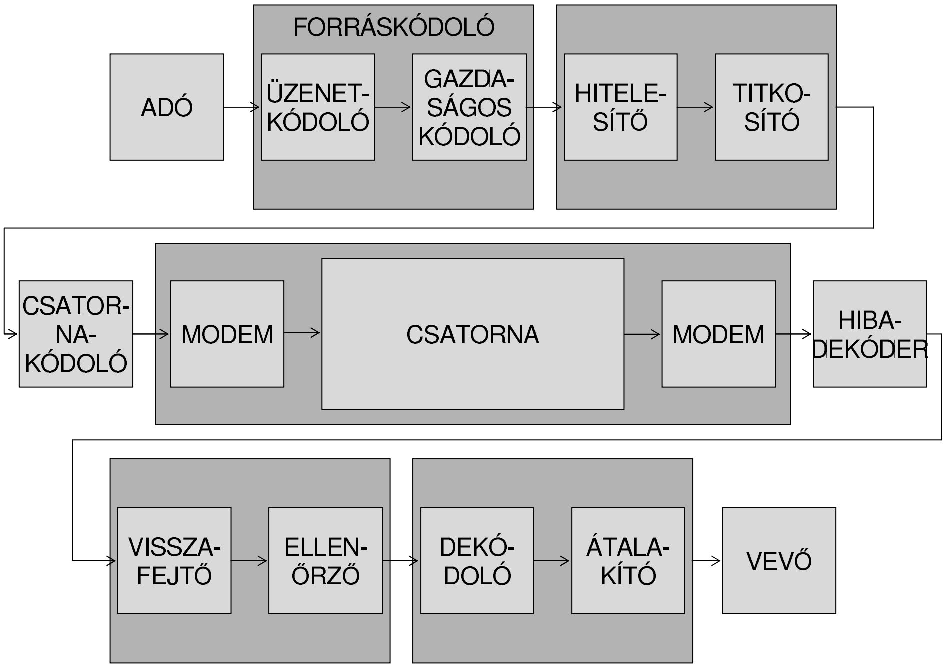
$$
\begin{aligned}
-H_r(p_1, \ldots, p_m) &= \sum_{i=1}^{m} p_i \log_r p_i = -\sum_{i=1}^{m} p_i \log_r (1/p_i) \\
&\geq -\log_r \Big(\sum_{i=1}^{m} p_i / p_i\Big) = -\log_r m. \quad \square
\end{aligned}
$$

9.1.4. Kódolás. A következő két kérdést együtt tárgyaljuk. Ahhoz, hogy az adatot továbbítani tudjuk, először is olyan alakra kell hozni, hogy képesek legyünk a céljainknak megfelelően manipulálni. A *kódolás* a legáltalánosabb értelemben az üzenetek halmazának egy másik halmazba való leképezését jelenti. Ha a leképezés injektív, akkor azt mondjuk, hogy a kódolás *felbontható* vagy *egyértelműen dekódolható*, egyébként *veszteséges*.

Gyakran az üzenetet valamilyen karakterkészlet elemeiből alkotott sorozattal adjuk meg. Ez az eljárás is egy kódolás. Az írás például egy kódolás. Kódolás tehát a kínai írás, és hasonlóan kódolás a latin betűs írás. Mindkét esetben arról van szó, hogy egy kódolandó üzenetet meghatározott módon felbontunk egymáshoz csatlakozó, előre rögzített olyan elemi részekre — nem teljesen pontosan fogalmazva az első esetben szavakra, a másodikban hangokra — , hogy minden üzenet előálljon ilyen elemi részek egymás után fűzésével, de egyetlen üzenetet se lehessen kétféleképpen felbontani ezekre az elemi részekre. Az ilyen részeknek megadjuk a kódját, és a teljes üzenetet úgy kódoljuk, hogy az előbbi részek kódját egymás után láncoljuk. A kódoláshoz tehát megadjuk az elemi részek kódját, amelyet egy szótár tartalmaz, és ezek segítségével kódoljuk az üzeneteket. Az ilyen kódolást betűnkénti kódolásnak nevezzük (léteznek más kódok is). A betűket gyakran számokká kódoljuk: jelenleg elterjedt kód az úgynevezett *ASCII-kód*, amellyel 128 különböző jel kódolható, illetve ennek a kódnak a kiterjesztései, amelyekkel 256 vagy 65536 jelet lehet kódolni.

A valóságban az egyes üzenetek különböző gyakorisággal fordulnak elő. Ez a felismerés vezetett ahhoz a gondolathoz, hogy célszerű a gyakrabban előforduló üzeneteket rövidebb kóddal megadni. Egy ilyen kódolás *tömörítés*t végez, amelynek során csökkentjük az eredeti jelsorozatban meglévő *redundanciát*. A redundancia, vagy más néven a *terjengősség* azt fejezi ki, hogy a mondandónkat lényegesen hosszabban fejezzük ki, mint amennyire az szükséges lenne. Például a `nyilvánosság` szó 12 betűből áll, holott legfeljebb öt betű az összes magyar szó kódolására elég lenne.

Az elsőként említett kódolást nevezzük *üzenetkódolás*nak, míg a másodikat *gazdaságos kódolás*nak. A két lépést együttesen *forráskódolás*nak nevezik.



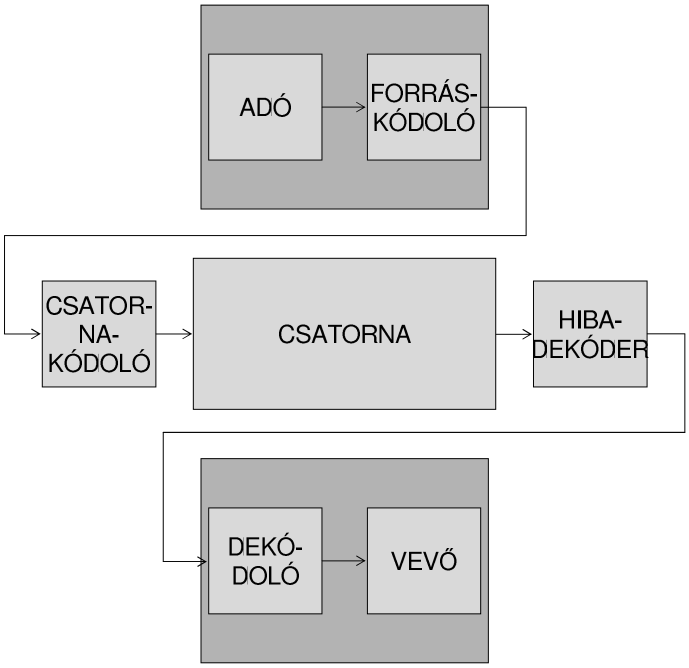
9.2. ábra

A csatornán áthaladó jel torzulást szenved, így a vétel helyére nem pontosan az a jelsorozat érkezik, mint amit a csatorna bemenetére betápláltak. Amennyiben bármilyen jelsorozat egy lehetséges üzenet, akkor a módosulást nem vesszük észre, a vétel helyén semmilyen módon nem tudjuk eldönteni egy vett jelsorozatról, hogy az megegyezik-e az eredetileg elküldött üzenettel, vagy egy másik üzenet változott meg, és így kaptuk a vett jelsorozatot. A biztonságosabb átvitel érdekében úgynevezett *hibakorlátozó kódolás*t végzünk a csatorna bemenetén. Ezzel a kódolással szándékosan viszünk redundanciát a kódolt üzenetbe. Gondoljuk meg, hogy az élő nyelvek redundanciájának köszönhető, hogy zajos körülmények között is megértjük, amit egy előadó mond, jóllehet az általunk hallott szöveg nem teljesen azonos azzal, ami eredetileg elhangzott. A megfelelő hibakorlátozó kód kiválasztása függ a csatornától is, ezért ezt a kódolást *csatornakódolás*nak nevezik.

Végezetül a kódolt üzenetet ténylegesen, fizikailag is át kell vinni a csatornán. Az átvitelhez olyan alakra kell transzformálni az üzenetet, amely forma fizikailag alkalmas a nagy távolságra történő továbbításra. Ilyen például, ha leírjuk az üzenetet, és levél formájában küldjük el, vagy ha elektromos jellé alakítjuk át, és például telefonvonalon keresztül továbbítjuk. Ezt a berendezést most — kissé pontatlanul — *modem*nek fogjuk nevezni.

Bár a közvetlen témánkhoz nem szorosan kapcsolódik, ám mindenképpen a kommunikációval kapcsolatos a titkosság és a hitelesség kérdése, vagyis az adatátvitel során, amennyiben szükséges, gondoskodni kell az üzenet titkosításáról és hitelesítéséről is. Az utóbbit például aláírással lehet megvalósítani.

Az előbbiek alapján a kommunikáció részletesebben a 9.2. ábra szerint valósítható meg. Az ábrán is jelzett összevonások, valamint a rejtjelezéssel kapcsolatos részek elhagyása után a 9.3. ábra szerinti egyszerűbb modellt kapjuk.



9.3. ábra

9.2. Forráskódolás

Lángné-Gonda: O.K.; Gonda kiegészítés; Gonda bevmat3; Gonda kódoláselmélet; Jákó; Salomon; Iványi Informatikai algoritmusok1; Cormen–Leiserson–Rivest: O.K.; Rónyai–Ivanyos–Szabó: O.K.; Demetrovics–Denev–Pavlov; Knuth TAOCP; Birkhoff–Bartee.

9.2.1. Betűnkénti kódolás. Mint már említettük, a betűnkénti kódolás során az eredeti üzenetet meghatározott módon egymáshoz átfedés nélkül csatlakozó részekre bontjuk, egy-egy ilyen részt egy szótár alapján kódolunk, és az így kapott kódokat az eredeti sorrendnek megfelelően egymáshoz láncoljuk. Az általánosság csorbítása nélkül feltehetjük, hogy a szótár alapján kódolandó elemi üzenetek egy *A ábécé* (a kódolandó ábécé) *betűi*, és egy-egy ilyen betű kódja egy másik (az előbbitől nem feltétlenül különböző) B ábécé, a kódábécé betűivel felírt *szó*, vagyis ezen ábécéből vett betűk véges hosszúságú sorozata, a sorozat elemeit egyszerűen egymás mellé írva. A továbbiakban mindkét ábécéről feltesszük, hogy nem üres és véges. Ha a B ábécé egyelemű, kételemű, háromelemű, stb., akkor rendre unáris, bináris, ternáris, stb., kódról beszélünk. Az A ábécé betűivel felírható összes (legalább egy betűt tartalmazó) szó halmazát A^+, míg az egyetlen betűt sem tartalmazó *üres szó*val, (jele: $\emptyset$ vagy λ) kibővített halmazt A^* jelöli. Az előbbiek alapján a betűnkénti kódolást egy $\varphi : A \to B^*$ leképezés határozza meg, amelyet úgy terjesztünk ki egy $\psi : A^* \to B^*$ leképezéssé, hogy ha $a_1 a_2 \dots a_n = \alpha \in A^*$ (tehát $n \in \mathbb{N}$, és $a_i \in A$, ha $1 \leq i \leq n$), akkor α kódja $\psi(\alpha) = \varphi(a_1)\varphi(a_2)\dots\varphi(a_n)$. Nyilván ha φ nem injektív, vagy az üres szó benne van az értékkészletében, akkor a kapott ψ kódolás nem felbontható, ezért betűnkénti kódolásnál mindig fel fogjuk tenni, hogy φ injektív és B^+-ba képez.

9.2.2. Példa. Tipikus példa betűnkénti kódolásra a 9.1. táblázatban látható *Morse-kód*.

betű	kód	betű	kód	betű	kód
A	·—	J	·———	S	···
B	—···	K	—·—	T	—
C	—·—·	L	·—··	U	··—
D	—··	M	——	V	···—
E	·	N	—·	W	·——
F	··—·	O	———	X	—··—
G	——·	P	·——·	Y	—·——
H	····	Q	——·—	Z	——··
I	··	R	·—·		
szám	**kód**	**szám**	**kód**	**írásjel**	**kód**
0	—————	5	·····	.	·—·—·—
1	·————	6	—····	,	——··——
2	··———	7	——···	?	··——··
3	···——	8	———··	:	———···
4	····—	9	————·	-	—····—

9.1. táblázat

Itt a kódolandó ábécé az angol ábécé, míg a kódoló ábécé két betűt tartalmaz, a pontot és a vonást, és például az `ad` szó kódja „·——··". Ezt így nem tudnánk dekódolni, mert nem injektív a megfeleltetés. Például az `emi` szó kódja megegyezik az előbbi kóddal. A valóságban természetesen az egyes jelek, továbbá az egyes betűk adása között meghatározott szünetet tartanak, és így már a kódolás egyértelmű lesz. Konkrétan egy vonás háromszor annyi ideig tart, mint egy pont, és minden jel után egy egységnyi szünet következik, majd a betű kódja végén még további két egységnyi szünet van (egy pont átviteléhez szükséges időt tekintve egységnek). Ha egy egységnyi hosszúságú jelet egy 1-es, és az egységnyi hosszúságú szünetet 0 jelöli, akkor például a `B` kódja, amely az eredeti, ponttal és vonással megadott rendszerben „—···", most `111010101000` lesz.

Így már egyértelműen dekódolható a vett üzenet, hiszen minden betű kódja három nullára végződik, és nincs olyan betű, amelynél a kód belsejében, vagy a kódszó elején lenne három nulla. Ennél a kódnál a három nullának önmagában is van jelentése, ez a szóköz kódja, vagyis ez választja el egymástól az egymás után következő szavak kódját (így egy szó kódjának a végén hat darab nulla áll). Ekkor `ad` kódja `10111000111010100000`, míg `emi` kódja `1000111011100010100000`, így a két kód jól megkülönböztethető és a kód egyértelműen megfejthető.

9.2.3. Prefix, szuffix, infix. Legyen α, β és γ az A ábécével felírt három szó. Ekkor α *prefixe* (vagy *előtagja*) és γ *szuffixe* (vagy *utótagja*) az $\alpha\gamma$ szónak, β pedig *infixe* (vagy *belső tagja*) $\alpha\beta\gamma$-nak. Szavak egy halmaza *prefixmentes halmaz*, ha nincs benne két olyan különböző szó, hogy egyik a másik prefixe.

Ha α egy szó, akkor az üres szó és α mind prefixe, mind szuffixe, mind pedig infixe α-nak. Ezek az α *triviális prefix*ei, *triviális szuffix*ei és ei. A prefix, szuffix illetve infix *valódi prefix*, *valódi szuffix*, illetve , ha nem egyezik meg α-val.

9.2.4. Kódfa. A betűnkénti kódolás jól szemléltethető irányított fa segítségével. Legyen $\varphi : A \to B^*$ egy injektív leképezés, ahol A a kódolandó ábécé és B a kódábécé, mindkettő véges. A φ értékkészletében lévő kódszavak összes prefixeinek halmaza részbenrendezett a „prefixe" relációra. Készítsük el ennek a relációnak a Hasse-diagramját. Nyilván egy irányított fát kapunk, amelynek gyökere az üres szó, és minden szó a hosszának megfelelő szinten van. A gráf éleit színezzük úgy, hogy ha $\beta = \alpha b$ valamely $b \in B$-re, akkor a β-ból α-ba vezető él színe legyen b. Nyilván bármely csúcs esetén a csúcsból kivezető élek mind különböző színűek. A kódfa valamely csúcsát, amely nem levél, *teljes csúcs*nak illetve *csonka csúcs*nak nevezzük, attól függően, hogy minden színhez tartozik-e a csúcsból kiinduló, az adott színnel színezett él. A kódfa csúcsait is kiszínezhetjük: az $a \in A$ kódjának megfelelő csúcs színe legyen a, azon csúcsok színe, amelyek nincsenek φ értékkészletében, legyen üres. Vegyük észre, hogy minden levél ki van színezve.

Az előbbi konstrukciót meg is fordíthatjuk. Tekintsünk egy véges élszínezett irányított fát, ahol a színek halmaza B. Tegyük fel, hogy az egy csúcsból kiinduló élek mind különböző színűek. Képezzük le az A véges ábécét a fa csúcsaira kölcsönösen egyértelműen úgy, hogy minden levél fellépjen képként. Az $a \in A$ betű kódja legyen az a szó, amelyet úgy kapunk, hogy a gyökértől az a-nak megfelelő csúcsig haladó irányított úton összeolvassuk az élek színeit.

9.2.5. Prefix kód, egyenletes kód és vesszős kód. Tegyük fel, hogy a $\varphi : A \to B^+$ injektív leképezés $\mathrm{rng}(\varphi)$ értékkészlete B^+ prefixmentes részhalmaza. A φ által meghatározott $\psi : A^* \to B^*$ betűnkénti kódolás nyilván könnyen dekódolható, mert ha egymás után érkeznek a kódábécé betűi, és nézzük az addig beérkezett szimbólumokból összeálló szót, akkor amint ez a kódolandó ábécé valamely betűjének kódja, azonnal dekódolható, hiszen a folytatásával kapott jelsorozat már egyetlen betűnek sem lehet a kódja. Ezen dekódolási módszer miatt szokás az ilyen kódot *prefix kód*nak nevezni. (A *prefixmentes kód* elnevezés is használatos, mivel a dekódolás $\mathrm{rng}(\varphi)$ prefixmentességén múlik.) Pefix kód nyilván felbontható.

Egy betűnkénti kód *egyenletes kód*, vagy *fix hosszúságú kód*, ha a betűk kódjainak hossza azonos. Mivel egy ilyen kód nyilván prefix, ezért felbontható.

Egy betűnkénti kód *vesszős kód*, ha van olyan ϑ szó, a *vessző*, hogy ϑ minden kódszónak szuffixe, de egyetlen kódszó sem áll elő $\alpha\vartheta\beta$ alakban nem üres β szóval. Egy vesszős kód prefix kód, mert a vessző egyértelműen jelzi a beérkezett jelsorozatban egy-egy kódszó végét, és ha ezt folytatjuk, akkor már biztosan nem kapunk kódszót, hiszen ebben a meghosszabbított sorozatban a vessző infix lenne. Ha egy betűnkénti kódban nincs vessző, akkor a kód *vesszőmentes*.

A Morse-kódnál láttuk, hogy az eredeti, a csak a ponttal és vonással megadott betűnkénti kód nem felbontható, míg az 1-gyel és 0-val való átírás után kapott szabály vesszős kód, ahol a 000 jelsorozat a vessző.

Egy betűnkénti kód pontosan akkor prefix kód, ha a kódfájának csak a levelei kódszavak.

9.2.6. Példák. Az fenti jelölésekkel legyen $A = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$, és $B = \{\mathtt{0}, \mathtt{1}\}$.

(1) Legyen $\varphi(\mathtt{a}) = \mathtt{10}$, $\varphi(\mathtt{b}) = \mathtt{1}$ és $\varphi(\mathtt{c}) = \mathtt{00}$. Ez a kód egyértelműen fejthető, tehát ψ injektív. Tegyük ugyanis fel, hogy valameddig már dekódoltuk a beérkezett kódolt üzenetet, és most érkezik egy újabb jel. Ha ez a jel `0`, akkor utána csak egy `0` következhet, és ez a `00` csupán a `c` kódja lehet. Amennyiben az előző jel `1`, akkor ezt még nem tudjuk dekódolni. Amennyiben ez után az `1` után ismét `1` következik, akkor már tudjuk, hogy az első `1` egy `b` kódja, de a második `1`-et nem tudjuk dekódolni. Mindaddig, amíg csak `1`-ek követik egymást, az utolsóként beérkező `1` kivételével valamennyit egyértelműen dekódolunk `b`-be. Abban az esetben, ha az `1` után vége van az adásnak, akkor ezt az utolsóként maradt `1`-et szintén egyértelműen `b`-ként dekódoljuk. Ha viszont az `1` után `0` következik, akkor mindaddig, amíg vagy egy `1` nem érkezik, vagy vége nincs az adásnak, nem tudunk dekódolni. Amikor viszont vagy egy `1` érkezik, vagy vége van az adásnak, akkor hátulról visszafelé végrehajtható a dekódolás: minden két-két `0`-t `c`-nek dekódolunk, és a végén vagy egy egyedül álló `1` marad, ami a `b` kódja, vagy `10` lesz a maradék, ami viszont az `a` kódja. Könnyen ellenőrizhető, hogy másként az előbbi jelsorozat nem dekódolható. Ez tehát egy felbontható kód.

(2) Legyen $\varphi(\mathtt{a}) = \mathtt{00}$, $\varphi(\mathtt{b}) = \mathtt{1}$ és $\varphi(\mathtt{c}) = \mathtt{01}$. A kódszavak halmaza prefixmentes, és így ez egy prefix kód. Az előbbi példában nem ez volt a helyzet, mert az `1`-et önmagában nem tudtuk dekódolni. Minden prefix kód egyértelműen dekódolható kód.

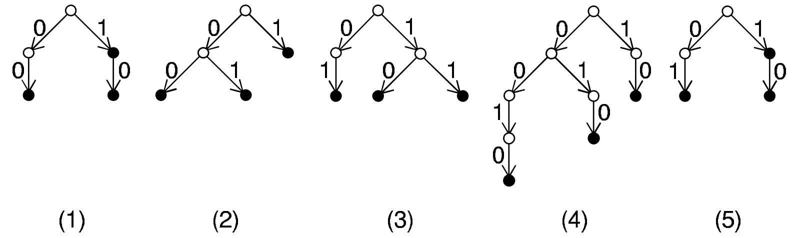
(3) Legyen $\varphi(\mathtt{a}) = \mathtt{10}$, $\varphi(\mathtt{b}) = \mathtt{11}$ és $\varphi(\mathtt{c}) = \mathtt{01}$. Ez egyenletes kód, így egyben prefix kód.

(4) Legyen $\varphi(\mathtt{a}) = \mathtt{0010}$, $\varphi(\mathtt{b}) = \mathtt{10}$ és $\varphi(\mathtt{c}) = \mathtt{010}$. Ennél a kódnál minden kódszó `10`-ra végződik, vagyis `10` mindegyiknek szuffixe, de az `10` egyetlen kódszónak sem valódi prefixe, és nem is valódi infixe. Ez a kód tehát vesszős kód, így prefix kód.

(5) Legyen $\varphi(\mathtt{a}) = \mathtt{10}$, $\varphi(\mathtt{b}) = \mathtt{1}$ és $\varphi(\mathtt{c}) = \mathtt{01}$. Ekkor $\psi(\mathtt{ab}) = \varphi(\mathtt{a})\varphi(\mathtt{b}) = \mathtt{101} = \varphi(\mathtt{b})\varphi(\mathtt{c}) = \psi(\mathtt{bc})$, tehát ez a kód nem dekódolható.

A 9.4. ábra mutatja a fenti kódokhoz tartozó kódfákat. A teli körök jelölik a kódszavakat. A három középső kód mindegyike prefix, és ezeknél a kódoknál csak a levelek kódszavak.

9.2.7. Tétel: McMillan-egyenlőtlenség. *Legyen $A = \{a_1, \dots, a_n\}$ és B két ábécé, B elemeinek a száma $r \geq 2$, és $\varphi : A \to B^+$ injektív leképezés. Ha a φ által meghatározott betűnkénti kódolás felbontható, akkor $l_i = \left|\varphi(a_i)\right|$ jelöléssel $\sum_{i=1}^{n} r^{-l_i} \leq$*



9.4. ábra

1. *Fordítva, ha $l_1, \ldots, l_n$ olyan pozitív egész számok, hogy $\sum_{i=1}^{n} r^{-l_i} \le 1$, akkor van az A-nak a B elemeivel való olyan prefix kódolása, hogy az a_i betű kódjának hossza l_i.*

A fenti tételben szereplő egyenlőtlenség a *McMillan-egyenlőtlenség*, amely lényegében véve azt fejezi ki, hogy egy felbontható kódban az átlagos szóhosszúság nem lehet túlságosan kicsi (mert ha a szóhosszúság kicsi, akkor r megfelelő negatív kitevős hatványa nagy, viszont az összeg értéke nem haladhatja meg az 1-et). A tétel második része azt mutatja, hogy egy nem prefix, de felbontható kódnak nincs nagy jelentősége, hiszen ugyanolyan hosszakkal készíthető prefix kód is. A második rész bizonyítása konstruktív: algoritmust ad a kód megkonstruálására.

* **Bizonyítás.** Tekintsük a $\varphi : A \to B^+$ injektív leképezés által meghatározott felbontható kódot. Jelöljük C-vel $\mathrm{rng}(\varphi) \subset B^+$-t, M-mel a $\sum_{i=1}^{n} r^{-l_i}$ összeget. Indukcióval megmutatjuk, hogy $\sum_{\alpha \in C^i} r^{-|\alpha|} = M^i$. Ha $\alpha \in C^{i+1}$, akkor α megadható $\alpha = \alpha_1\alpha_2$ alakban, ahol $\alpha_1 \in C^i$ és $\alpha_2 \in C$, és ha a kód felbontható, akkor ez a felbontás egyértelmű.

$$\begin{aligned}\sum_{\alpha \in C^{i+1}} r^{-|\alpha|} &= \sum_{\alpha_1 \in C^i} \sum_{\alpha_2 \in C} r^{-|\alpha_1\alpha_2|} = \sum_{\alpha_1 \in C^i} \sum_{\alpha_2 \in C} r^{-\left(|\alpha_1|+|\alpha_2|\right)} \\ &= \sum_{\alpha_1 \in C^i} \sum_{\alpha_2 \in C} r^{-|\alpha_1|} r^{-|\alpha_2|} = \sum_{\alpha_1 \in C^i} r^{-|\alpha_1|} \sum_{\alpha_2 \in C} r^{-|\alpha_2|} \\ &= M^i M = M^{i+1},\end{aligned}$$

ahol az első egyenlőségnél használtuk ki, hogy a kód felbontható.

Ezt az összeget másképp is ki tudjuk számítani, ha az összegben szereplő tagokat a kitevőkben szereplő hosszak növekvő sorrendje szerint írjuk, és az azonos kitevőhöz tartozó tagokat összevonjuk. Legyen $w_k^{(i)}$ a C^i-beli k hosszúságú szavak száma (a szavak hosszát mint B^+-beli szavak hosszát értve), és L a C-beli szavak hosszának maximuma. Ekkor a C^i-beli szavak hosszának maximuma iL, és

$$M^i = \sum_{\alpha \in C^i} r^{-|\alpha|} = \sum_{k=1}^{iL} w_k^{(i)} r^{-k} \le \sum_{k=1}^{iL} r^k r^{-k} = iL,$$

mert r szimbólummal maximum r^k különböző k hosszúságú szó írható fel. A fentiek szerint minden pozitív egész i-re $M^i \le iL$, vagyis $M^i/i \le L$. Ha $M > 1$, akkor $i \ge 2$

esetén

$$L \geq \frac{M^i}{i} = \frac{\big(1+(M-1)\big)^i}{i} = \frac{1}{i}\sum_{j=0}^{i}\binom{i}{j}(M-1)^i > \frac{1}{i}\binom{i}{2}(M-1)^2 = \frac{i-1}{2}(M-1)^2,$$

ami ellentmondás.

A tétel második állításának bizonyításához feltehetjük, hogy $l_1 \leq l_2 \leq \ldots \leq l_n$, és legyen $1 \leq k \leq n$-re $c_k = \sum_{i=1}^{k-1} r^{-l_i}$. Ez a sorozat szigorúan monoton nő, az első eleme 0, és a $\sum_{i=1}^{n} r^{-l_i} \leq 1$ feltétel alapján minden tagja kisebb 1-nél. Mivel $r^{l_k}c_k = \sum_{i=1}^{k-1} r^{l_k-l_i} < r^{l_k}$, és a bal oldalon egy nem negatív egész szám áll, hiszen a hosszak rendezése következtében az összeg tagjainak kitevője nem negatív egész szám, $r^{l_k}c_k$ felírható az r alapú számrendszerben pontosan l_k hosszban (a felírás kezdődhet nullával is). Ez lesz az a_k kódja, a számjegyeket kicserélve B betűire. (Másként, az 1-nél kisebb c_k nem negatív valós szám r-alapú számrendszerben pontosan felírható törtrészében l_k jeggyel, és ezek a jegyek adják az a_k kódját.) Amennyiben $1 \leq j < k \leq n$, akkor

$$r^{l_k}(c_k - c_j) = \sum_{i=1}^{k-1} r^{l_k-l_i} - \sum_{i=1}^{j-1} r^{l_k-l_i} = \sum_{i=j}^{k-1} r^{l_k-l_i} \geq r^{l_k-l_j},$$

ezért a megfelelő kódok első l_j számjegye közül legalább egy különböző, tehát a megadott kód prefix kód. □

9.2.8. Átlagos szóhosszúság, optimális kód. A gazdaságos kódolás szempontjából döntő a kód átlagos szóhosszúsága. Legyen $A = \{a_1, \ldots, a_n\}$ a kódolandó ábécé, $p_1, \ldots, p_n$ a betűk eloszlása egy kódolás során, $\varphi : A \to B^+$ egy betűnkénti kódolás, l_i az a_i kódjának hossza. Ekkor $\bar{l} = \sum_{i=1}^{n} p_i l_i$ a *kód átlagos szóhosszúsága*. Ha egy felbontható betűnkénti kód átlagos szóhosszúsága minimális, akkor *optimális kód*nak nevezzük. Első pillantásra nem világos, hogy van-e optimális kód, mivel valós számok egy részhalmazában nem feltétlenül van minimális elem. Válasszunk azonban egy tetszőleges felbontható kódot, és legyen ennek átlagos szóhosszúsága l. Mivel $p_i l_i > l$ esetén a kód nem lehet optimális, elég azon kódokat tekintenünk, amelyekre $l_i \leq l/p_i$, ha $i = 1, \ldots, n$. Ilyen kód csak véges sok van, így van köztük minimális átlagos hosszúságú. Mint tudjuk, ez választható prefix kódnak is.

9.2.9. Shannon tétele zajmentes csatornára. *Az előző definíció jelöléseivel, legyen B elemeinek száma r. Ha a betűnkénti kódolás felbontható, akkor $H_r(p_1, \ldots, p_n) \leq \bar{l}$, ahol H_r az eloszlás entrópiája.*

* **Bizonyítás.** A McMillan-egyenlőtlenséget, $-\log_r$ szigorú konvexségét és a 9.1.2.

segédtételt használva,

$$\begin{aligned}
\bar{l} - H_r(p_1, \dots, p_n) &= \sum_{i=1}^{n} p_i l_i + \sum_{i=1}^{n} p_i \log_r p_i \\
&= -\sum_{i=1}^{n} p_i \log_r r^{-l_i} - \sum_{i=1}^{n} p_i \log_r \frac{1}{p_i} \\
&= -\sum_{i=1}^{n} p_i \log_r \frac{r^{-l_i}}{p_i} \geq -\log_r \Big(\sum_{i=1}^{n} r^{-l_i}\Big) \\
&\geq -\log_r 1 = 0. \quad \square
\end{aligned}$$

9.2.10. Tétel: Shannon-kód létezése. *Az előző tétel jelöléseivel, $n > 1$ esetén van olyan prefix kód, amelyre $\bar{l} < H_r(p_1, \dots, p_n) + 1$.*

A bizonyítás konstruktív, algoritmust ad a kód meghatározására.

* **Bizonyítás.** Válasszunk olyan $l_1, \dots, l_n$ természetes számokat, amelyekre $r^{-l_i} \leq p_i < r^{-l_i+1}$, ha $i = 1, \dots, n$. Ekkor $\sum_{i=1}^{n} r^{-l_i} \leq \sum_{i=1}^{n} p_i = 1$, így a 9.2.7. tétel szerint van prefix kód az adott hosszakkal. Mivel $l_i < 1 - \log_r p_i$, erre

$$\sum_{i=1}^{n} p_i l_i < \sum_{i=1}^{n} p_i (1 - \log_r p_i) = 1 + H_r(p_1, \dots, p_n). \quad \square$$

9.2.11. Tétel: optimális kód konstrukciója. *Az előző tétel jelöléseivel legyen $n > 1$ és a kódszavak hosszának maximuma L. Egy optimális prefix kódra*

(1) *ha $p_i > p_j$, akkor $l_i \leq l_j$;*

(2) *a kódfában legfeljebb csak az $L-1$-edik szinten van csonka csúcs;*

(3) *a kódfában minden nem levél csúcsból legalább két él indul ki.*

Továbbá

(4) *van olyan optimális prefix kód, amelynek kódfájában legfeljebb egy csonka csúcs van;*

(5) *ha egy optimális prefix kód kódfájában nincs csonka csúcs, akkor*

$$2 + \big((n-2) \bmod (r-1)\big) = r,$$

ha pedig egy csonka csúcs van, akkor annak kifoka

$$2 + \big((n-2) \bmod (r-1)\big);$$

(6) *ha $n \leq r$, akkor egybetűs kódszavakat választva, optimális prefix kódot kapunk;*

(7) *legyen* $\beta_1, \ldots, \beta_n$ *az* r*-elemű kódábécével megadott, a* $p_1, \ldots, p_n$ *eloszláshoz tartozó optimális kód, amelynek a kódfájában nincs csonka csúcs. Ha* $2 \le m \le r$*, a* $p_{n+1}, \ldots, p_{n+m}$ *valós számokra* $\sum_{i=1}^{m} p_{n+i} = p_k$*, és*

$$\max\{p_{n+1}, \ldots, p_{n+m}\} \le \min\{p_1, \ldots, p_n\},$$

akkor

$$\beta_1, \ldots, \beta_{k-1}, \beta_{k+1}, \ldots, \beta_n, \beta_k b_1, \ldots, \beta_k b_m,$$

ahol $b_1, \ldots, b_m$ *a* B *különböző elemei, a*

$$p_1, \ldots, p_{k-1}, p_{k+1}, \ldots, p_n, p_{n+1}, \ldots, p_{n+m}$$

eloszláshoz tartozó optimális kód.

Bizonyítás. Ha (1) nem teljesül, akkor

$$0 < (p_i - p_j)(l_i - l_j) = (p_i l_i + p_j l_j) - (p_i l_j + p_j l_i),$$

és innen $p_i l_j + p_j l_i < p_i l_i + p_j l_j$, azaz a két betű kódját felcserélve csökken az átlagos szóhosszúság.

Ha (2) nem teljesül, és a t-edik szinten van csonka csúcs, ahol $t < L-1$, akkor egy L-edik szinten lévő levelet a hozzá vezető éllel áthelyezve erre a csonka csúcsra, és esetleg az áthelyezett élt átszínezve, a levélhez tartozó betű kódjának hossza L-ről $t+1$-re változik, vagyis csökken az átlagos szóhosszúság.

Ha (3) nem teljesül, és egy csúcsot csak egy él hagy el, akkor elhagyva ezt az élt, és az ennek az élnek a másik végén lévő csúcsból kiinduló részfát áthelyezve az előbbi csúcsba, az áthelyezett részfa leveleihez tartozó kódszavak hossza eggyel csökken, és így az átlagos szóhosszúság is csökken.

Legyen egy optimális kód kódfájában két különböző csonka csúcs. Az előbbiek szerint ezek azonos szinten vannak, így egyikükről az egyik élhez tartozó levelet az éllel együtt áthelyezve a másik csonka csúcsra, és esetleg ezt az élt átszínezve, az átlagos szóhosszúság nem változik. Ilyen áthelyezésekkel viszont vagy a fogadó csúcs telítődik, tehát a csonka csúcsok száma csökken, vagy a másik csonka csúcson legfeljebb egy él maradna, ami az előbbi pont alapján optimális kód kódfáján lehetetlen. Ezzel beláttuk (4)-et.

Hogy belássuk (5)-öt, tekintsünk egy n levelet tartalmazó kódfát. Töröljük ebben a fában az egyik csúcshoz tartozó valamennyi levelet, a csonka csúcshoz tartozó levelekkel kezdve, ha van csonka csúcs. Ha t számú levelet töröltünk, akkor az új fában $t-1$-gyel kevesebb levél lesz. Folytassuk az eljárást mindaddig, amíg a végén csak a gyökér marad meg. Ha összesen s lépést hajtottunk végre, akkor $t-1+(s-1)(r-1) = n-1$, vagyis $n-2 = (s-1)(r-1)+(t-2)$, és $0 \le t-2 < r-1$.

A (6)-ban megadott kód nyilván optimális prefix kód. (7)-ben egy levélhez kapcsolódó kódszót helyettesítünk m kódszóval, amelyek szintén levélen helyezkednek el. Jelölje a kapott kódot φ. Tegyük fel, hogy az állítás nem igaz, és legyen φ^* a

$$p_1, \ldots, p_{k-1}, p_{k+1}, \ldots, p_n, p_{n+1}, \ldots, p_{n+m}$$

eloszláshoz tartozó optimális prefix kód. Ekkor φ^* átlagos hossza kisebb, mint φ átlagos hossza. Az általánosság csorbítása nélkül feltehetjük, hogy φ^* kódfájában is legfeljebb csak egy csonka csúcs van. A φ^* kód konstrukciójából, illetve (5)-ből mindkét kódban egyszerre van csonka csúcs, és ha van, akkor a kifoka ugyanannyi, m, ha pedig nincs, akkor $m = r$. Mivel φ^* optimális kód, ezért az m darab legkisebb valószínűséghez tartozó kódszó a legmagasabb szinten van a kódfában, és ugyanezen a szinten vannak azok a kódszavak, amelyek az esetleges csonka csúcshoz tartoznak. Mivel kódszavak cseréje, ha azok azonos szinten vannak, nem változtatja meg a kód átlagos hosszát, ezért feltehetjük, hogy mindkét kódban az m legkisebb valószínűségű kódszóhoz tartozó levél ugyanahhoz a csúcshoz tartozik. Ezeket törölve, mindkét kód átlagos hossza ugyanannyival változik. Így a φ^*-ból kapott kód átlagos hossza kisebb lesz, mint a φ-ből kapott kódé, ami lehetetlen, hiszen feltettük, hogy a φ-ből kapott kód optimális kód. $\square$

9.2.12. Huffman-kód. Optimális kódot ad az úgynevezett *Huffman-kód*, amelyet az előző tétel (6)–(8) pontjai alapján tudunk megszerkeszteni. Rendezzük a relatív gyakoriságok csökkenő sorrendjében a betűket, majd osszuk el $n-2$-t $r-1$-gyel, és legyen t a maradék plusz 2. Első lépésben helyettesítsük a sorozat t utolsó betűjét egy újabb betűvel, amelyhez az elhagyott betűk relatív gyakoriságainak az összegét rendeljük, és az így kapott gyakoriságoknak megfelelően helyezzük el az új betűt a sorozatban. Ezek után ismételjük meg az előző redukciót, de most már minden lépésben r betűvel csökkentve a kódolandó halmazt, mígnem már csak r betű marad (feltehetjük, hogy induláskor több, mint r betű volt, ellenkező esetben a redukció elmarad). Most a redukált ábécé legfeljebb r betűt tartalmaz, és ha volt redukció, akkor pontosan r betűt. Ezeket a kódoló ábécé elemeivel kódoljuk, majd a redukciónak megfelelően visszafelé haladva, az ott összevont betűk kódját az összevonásként kapott betű már meglévő kódjának a kódoló ábécé különböző betűivel való kiegészítésével kapjuk.

9.2.13. Példa. Mutatunk egy példát a Huffman- és a Shannon-kódra. Mindkét esetben ugyanazon forrást fogjuk kódolni, így összehasonlítható a két kód hatékonysága.

Legyen a kódolandó ábécé 10-elemű (mondjuk az angol ábécé első 10 betűjével jelölve), az egyes betűk relatív gyakorisága rendre 0,17, 0,02, 0,13, 0,02, 0,01, 0,31, 0,02, 0,17, 0,06, 0,09, és a kódoló ábécé $\{\mathtt{0}, \mathtt{1}, \mathtt{2}\}$. Mivel $10-2 = 4\cdot(3-1)+0$, az első lépésben $0+2 = 2$ betűt kell összefogni, majd a további lépésekben hármat-hármat (9.2. táblázat). Amikor már csak 3 betűből áll az ábécé, akkor ezt a három betűt rendre `0`-val, `1`-gyel és `2`-vel kódoljuk, majd visszafelé haladva előállítjuk a kívánt kódot (9.3. táblázat).

A kód átlagos szóhosszúsága 1,79, míg az entrópia értéke 1,73, így a kód átlagos szóhosszúsága igen jól megközelíti az elméletileg elérhető legkisebb értéket.

Ugyanezen ábécét fogjuk most a Shannon-kóddal kódolni. Most is sorba kell rendezni az ábécét a relatív gyakoriságok csökkenő sorrendjében. Meghatározzuk a szükséges szóhosszúságokat. Mivel 0,31, 0,17, 0,13 kisebbek mint 1/3, de nem kisebbek, mint 1/9, `f`, `a`, `h` és `c` kódhossza 2. Hasonlóan `j` és `i` kódhossza 3, míg `b`, `d` és `g` kódhossza 4, végül `e` kódhossza 5. Az `f` kódja `00`, az `a` kódja `01`, a `h` kódja `02`, és ehhez a hármas alapú számrendszerben 1-et adva kapjuk `c` kódját, amely így `10`. Ehhez 1-et adva 11-et kapunk, de `j` kódjának hossza 3, ezért ezt még jobbról ki kell egészíteni egy darab 0-val, tehát `j`

f	0,31	f	0,31	f	0,31
a	0,17	a	0,17	a	0,17
h	0,17	h	0,17	h	0,17
c	0,13	c	0,13	c	0,13
j	0,09	j	0,09	j	0,09
i	0,06	i	0,06	$\big((\mathtt{g},\mathtt{e}),\mathtt{b},\mathtt{d}\big)$	0,07
b	0,02	$(\mathtt{g},\mathtt{e})$	0,03	i	0,06
d	0,02	b	0,02		
g	0,02	d	0,02		
e	0,01				

f	0,31	$(\mathtt{a},\mathtt{h},\mathtt{c})$	0,47
$\Big(\mathtt{j},\big((\mathtt{g},\mathtt{e}),\mathtt{b},\mathtt{d}\big),\mathtt{i}\Big)$	0,22	f	0,31
a	0,17	$\Big(\mathtt{j},\big((\mathtt{g},\mathtt{e}),\mathtt{b},\mathtt{d}\big),\mathtt{i}\Big)$	0,22
h	0,17		
c	0,13		

9.2. táblázat

$(\mathtt{a},\mathtt{h},\mathtt{c}) \mapsto 0$	$\mathtt{a} \mapsto 00$		
	$\mathtt{h} \mapsto 01$		
	$\mathtt{c} \mapsto 02$		
$\mathtt{f} \mapsto 1$			
$\Big(\mathtt{j},\big((\mathtt{g},\mathtt{e}),\mathtt{b},\mathtt{d}\big),\mathtt{i}\Big) \mapsto 2$	$\mathtt{j} \mapsto 20$		
	$\big((\mathtt{g},\mathtt{e}),\mathtt{b},\mathtt{d}\big) \mapsto 21$	$(\mathtt{g},\mathtt{e}) \mapsto 210$	$\mathtt{g} \mapsto 2100$
			$\mathtt{e} \mapsto 2101$
		$\mathtt{b} \mapsto 211$	
		$\mathtt{d} \mapsto 212$	
	$\mathtt{i} \mapsto 22$		

9.3. táblázat

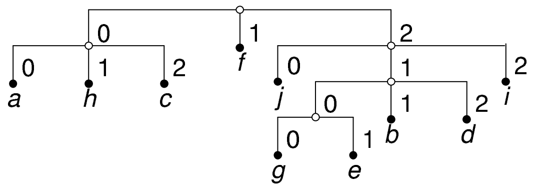
kódja 110. Hasonlóan haladva megkapjuk a teljes kódot, amelyet a 9.4. táblázat mutat (bal oldalon a szavak növekvő sorrendjében, jobbról a betűk sorrendjében). Most a kód átlagos szóhosszúsága 2,3, amely nagyobb, mint a Huffman-kódnál volt, de kisebb, mint az entrópia plusz 1.

Az 9.5. és 9.6. ábra mutatja a két kód kódfáját.

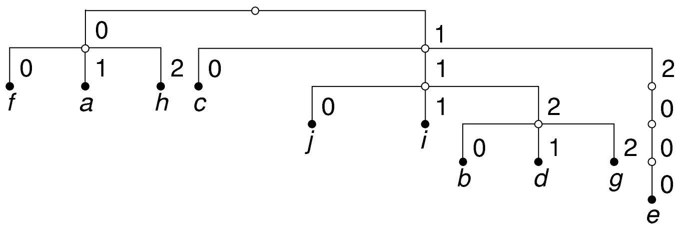
Megadjuk az ékezet nélküli betűk magyar nyelvben való előfordulásának relatív gyakoriságaihoz tartozó bináris Huffman- és Shannon-kódot (9.5. táblázat) (a Shannon-kódnál a Q hosszát az egyébként előforduló hosszak maximumának vettük). Az adott eloszláshoz tartozó entrópia értéke 4,1289, míg a Huffman-kód átlagos szóhosszúsága 4,1528, a Shannon-kódé pedig 4,5989. Ugyanakkor a Morse-kód 0-val és 1-gyel való kó-

betű	kód	betű	kód
f	00	a	01
a	01	b	1120
h	02	c	10
c	10	d	1121
j	110	e	12000
i	111	f	00
b	1120	g	1122
d	1121	h	02
g	1122	i	111
e	12000	j	110

9.4. táblázat



9.5. ábra



9.6. ábra

dolásánál az átlagos szóhosszúság ugyanezekkel a gyakoriságokkal (csak a betűk kódját figyelembe véve) 9,299. Látható, hogy mekkora javulás érhető el. Ennek egyik oka, hogy a vessző határozottan növeli az átlagos szóhosszúságot.

9.2.14. A kódolandó ábécé kiterjesztése. Egyszerű módon el tudjuk érni, hogy egy felbontható kódban az egy betűre jutó átlagos szóhosszúság tetszőlegesen megközelítse az entrópia értékét. Ehhez az eredeti ábécéből elkészítjük az összes kétbetűs, hárombetűs, stb. szót, és egy-egy ilyen szót tekintünk egy új ábécé egy-egy betűjének, ahol egy-egy ilyen új betűhöz a benne szereplő betűk relatív gyakoriságainak szorzatát rendelve kapjuk a megfelelő eloszlást. Az így kapott kód az eredeti kód *kétszeres*, *háromszoros*, stb. *kiterjesztése*. Ha m betű egybefogásából áll az ábécé, akkor van olyan

betű	valószínűség	Huffman-kód	Shannon-kód
A	0,1247	100	0010
B	0,0190	000011	101100
C	0,0065	01101110	10111101
D	0,0197	000010	101011
E	0,1407	001	000
F	0,0078	0110110	10111100
G	0,0326	01100	10011
H	0,0165	011010	101101
I	0,0444	1110	01111
J	0,0105	0000000	1011100
K	0,0541	1100	01110
L	0,0605	1010	01100
M	0,0352	01001	10010
N	0,0638	0111	0101
O	0,0683	0101	0100
P	0,0096	0000001	1011101
Q	0,0000	0110111111	1011111000 1001
R	0,0367	01000	10001
S	0,0600	1011	01101
T	0,0761	0001	0011
U	0,0226	11011	101001
V	0,0199	000001	101010
W	0,0009	011011110	10111110000
X	0,0001	0110111110	1011111000 1000
Y	0,0261	11010	101000
Z	0,0437	1111	10000

9.5. táblázat

kód, amelynek egy betűre jutó átlagos szóhosszúsága

$$\bar{l} < -\sum_{i=1}^{n} p_i \log_r p_i + \frac{1}{m},$$

ahol n az eredeti ábécé betűinek száma, és a p_i-k az eredeti ábécé betűinek relatív gyakoriságai. A módszer szépséghibája, hogy például a kétszeres kiterjesztésnél azonos számot rendel `ty`-hoz és `yt`-hez, holott a magyar nyelvben a két betűkombináció még megközelítőleg sem azonos gyakorisággal fordul elő. Ha a párok valódi gyakoriságait tekintjük, még jobb kódot kapunk.

9.2.15. Szótárkódok. A betűnkénti kódolás akkor igazán hatékony, ha az egymás után következő betűk egymástól függetlenek, vagyis egy betű megjelenése független attól, hogy előtte milyen betűket bocsájtott ki az adó. Egy élő nyelven megadott szöveg általában nem ilyen: a magyar nyelvben a `t` után sokkal gyakoribb az `y`, mint például az `r` után. A korábban említett kiterjesztés sem megoldás erre a problémára. Egy másik probléma, hogy amennyiben egy üzenet nem tipikus szöveg, vagyis benne a betűk előfordulásának valószínűsége eltér a szokásostól, akkor egy rögzített kódtábla, amely a tipikus szövegek betűgyakorisága alapján épül fel, nem biztosít optimális kódot még a Huffman-kód esetén sem. Ha viszont egy ilyen esetben a konkrét szöveg statisztikája alapján hozzuk létre a kódtáblát, akkor ezt is továbbítani kell, ami növeli a kódolt szöveg hosszát.

A *szótárkód*ok alapgondolata, hogy egy $\varphi \in A^* \to B^*$ *szótár*at használunk fel a kódolásra, amelynek értelmezési tartománya tartalmazza A-t, azaz a kódolandó ábécét. A szótár lehet állandó (statikus) vagy változó (dinamikus).

* **9.2.16. Futamkódolás.** Bizonyos szövegállományokban gyakran fordulnak elő ismétlődő karakterek (például ismétlődő szóközök). Ilyenkor célszerű lehet úgy tömöríteni, hogy a karaktert csak egyszer adjuk meg, majd megadjuk az ismétlések számát. A gondot az okozza, hogy az ilyen számokat el kell választani a szöveg karaktereitől. Egyik lehetőség, hogy a legalább (mondjuk) háromszor ismétlődő karaktert háromszor leírjuk, majd utána írunk egy decimális számjegyet, hogy még hányszor ismétlődik. Például a 40 karakter hosszú

`␣␣␣␣␣␣␣␣gugogogogol␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣`

sor így tömörítve

`␣␣␣5gugogogogol␣␣␣9␣␣␣6`

lesz; a 21 ismétlődést csak „két részletben” tudtuk leírni, mert nincs 9-nél nagyobb decimális számjegy. (Persze, a decimális számjegy helyett egy egész bájt is jelezhetné az ismétlődést, de az elv nem változna.)

Ha van olyan betű az ábécében, amely biztosan nem fordul elő a szövegben, például a `@`, akkor azt használhatjuk „escape” karakternek, jelezve, hogy az utána következő számjegy nem része a szövegnek, hanem az előző betű ismétlődéseinek számát kódolja (0 számjegy egy ismétlés, stb.). Ezzel a szöveg

`␣@6gugogogogol␣@9␣@8`

lesz. Ha `#` egy másik „escape” karakter, akkor például felhasználhatjuk annak jelzésére, hogy utána két számjegy írja le az eddig még nem kódolható számú ismétlődéseket (00 tizenegy ismétlés, stb.). Így a szöveg

`␣@6gugogogogol␣#09`

lesz. Egy további lehetőség, hogy ismert szöveghossz esetén az egyik „escsape” azt jelzi, hogy a szöveg végéig már csak egy „triviális” karakter — jelen esetben a szóköz — ismétlődik. A fenti példából így

`␣@6gugogogogol#`

lesz.

Látszólag gondot okoz „escape” karakterek keresése. Ha azonban a futamkódolást egy prefix kóddal kombináljuk, akkor annyi „escape” karaktert gyárthatunk, amennyit akarunk. Sőt, az escape karaktert és az utána következő számot egybefoghatjuk, és ezekhez a párokhoz rendelhetünk egy prefix kódszót, vagy akár a betűt, az escape karaktert és az utána következő számot fogjuk össze, és ehhez a hármashoz rendelünk prefix kódszót. Fax kódolására elterjedt eljárás egy ilyen kód. Csak fekete és fehér képelemek vannak, egy sor 1664 képelem, soronként olvasunk. A futamhosszak: $0, 1, 2, \ldots, 63$ és $64, 128, 192, 256, \ldots, 2560$. A kódhosszakat a 9.7. táblázat tartalmazza.

fehér futam hossza	bináris kód hossza	fekete futam hossza	bináris kód hossza	tetszőleges futam hossza	bináris kód hossza
0	8	0	10	1792...1920	11
1	6	1	3	1984...2560	12
2...7	4	2...3	2		
8...10	5	4	3		
12...17	6	5...6	4		
18...28	7	7	5		
29...63	8	8,9	6		
64...128	5	10...12	7		
192	6	13...14	8		
256	7	15	9		
320...640	8	16...18	10		
704...1600	9	19...25	11		
1664	6	26...63	12		
1728	9	64	10		
		128...448	12		
		512...1728	13		

9.6. táblázat

* **9.2.17. LZ77.** Ennek a szótár típusú kódolásnak az ötletét Lempel és Ziv egy 1977-ben megjelent cikke tartalmazza. A még nem kódolt szövegből egy l hosszúságú „ablakot” figyelünk, míg a már kódolt szövegnek az ablak előtti L hosszúságú része a „szótár”. Az ablakban elhelyezkedő szöveg egy legalább m hosszú prefixét keressük a szótárban, a szótár végétől kezdve. Ha több prefixet is találunk, a leghosszabbat választjuk, és ha az több helyen is előfordul, akkor a szótár végéhez legközelebb álló előfordulást. Ilyenkor az (n, d) pár kerül a kódolt szövegbe, ahol n a talált prefix hossza, $m \leq n \leq l$, a d pedig a távolság a szótár utolsó betűjétől, $0 \leq d < L$. Ha nem találunk elég hosszú prefixet, akkor az ablak első betűje kerül a kódszövegbe. Például ha a kódolandó ASCII szöveg `gugogogogol`, $l = 8$, $L = 16$, $m = 2$, akkor a kód `gugo`$(1,8)$`l` lesz. Ha n helyett $n - m + 128$ értékét írjuk a szövegbe, akkor a kód hexadecimális `6775676FF186C` lesz. A kódszöveget általában tovább kódoljuk egy prefix kódolással.

* **9.2.18. A gzip parancs.** Ez a parancs a „deflate” tömörített adatformátumot használja egyes blokkokra, de ha nem sikerül tömörítést elérni, a program úgy is dönthet, hogy egy részt csak átmásol. A deflate adatformátum részletes leírását lásd: Network Working Group RFC 1951. (RFC: Request for Comment.) Sok más hasonló program és a zlib tömörített könyvtárformátum (lásd RFC 1950) használja a deflate formátumot. A deflate formátum az LZ77 eljárást kombinálja két prefix kóddal.

A deflate első lépése egy LZ77-változat, amely bájtsorozaton dolgozik $l = 258$, $L = 2^{15}$, $m = 3$ paraméterekkel. Az eljárás annyiban módosul, hogy az ablak második betűjével kezdődő infixre is végeznek keresést a szótárban, és ha erre nagyobb n hosszt sikerül elérni, akkor az első bájtot egyedül viszik ki. (A parancs paraméterezésével „lebutíthatjuk” a keresést, és a program gyorsabb lesz, de a dekódolás nem változik.)

A második lépés a prefix kódolások alkalmazásából áll. A kódtáblák lehetnek előre definiáltak vagy változóak. Az n hosszat az $n \mapsto n - m + 257$ transzformációval a $257 \dots 512$ tartományba viszik át; 256 kimarad, az a blokk végét jelzi. Az előre definiált kódtáblákat a 9.7. táblázat tartalmazza. Az `xx...x`-szel jelölt rész bináris számláló. Ha változó kódtáblát használunk, akkor azt is át kell vinni. Ezt úgy oldják meg hatékonyan, hogy az `xx...x`-szel jelölt bináris számlálókat nem vonják be a kódba. A kódfa helyett csak a kódhosszakat tárolják sorrendben, az elő nem forduló esetekhez nulla hosszat adva meg. Ebből az információból a 9.2.7. tétel bizonyításánál megadott algoritmussal a kódfa már felépíthető. A kódhosszakat maximum 15-re korlátozzák és futamkódolják, majd prefixkódolják, és csak ennek a prefix kódnak a kódhossztábláját tárolják.

* **9.2.19. LZW-kódok.** Lempel és Ziv módszereinek egyik továbbfejlesztése Welchtől származik. A kódszavak prefix kódot alkotnak, például fix hosszúságúak. Kiinduláskor a kódtábla csak a kódolandó szöveg ábécéjének betűihez tartozó kódszavakat tartalmazza, és ez bővül a kódolás valamint a dekódolás során úgy, hogy bizonyos betűsorozatok kódját adjuk meg. Tegyük fel, hogy valameddig már kódoltuk az eredeti szöveget, és kezdjük el olvasni a kódolandó szöveget ettől a ponttól kezdve. Az első olvasott betű kódját tartalmazza a kódtábla. Olvassuk ki a következő betűt. Ha az első két betűből álló szöveg is szerepel a kódtáblában, akkor olvassuk ki a harmadik betűt, és ezt folytassuk mindaddig, amíg egy olyan betűhöz nem érünk, hogy a megelőző betűig terjedő szövegnek már van kódja, de az utolsó betűvel kiegészített szövegnek még nincs. Legyen a kiolvasott szöveg $a_1 \dots a_n a_{n+1}$, ahol $n \in \mathbb{N}^+$. Az előbbiek szerint tehát $a_1 \dots a_n$-nek már létezik kódja, de $a_1 \dots a_n a_{n+1}$-nek nem. Ekkor elküldjük az n-hosszúságú szöveg ismert kódját, és amennyiben a kódtáblában még van szabad hely, akkor a soron következő kódszót $a_1 \dots a_n a_{n+1}$-hez rendeljük, a kódtáblához pedig hosszáadjuk az $a_1 \dots a_n a_{n+1}$-ből és a kódjából álló rendezett párt. Amennyiben betelt a kódtábla, akkor nem bővítjük tovább.

A dekódolás hasonlóan történik. Az első vett kódszó biztosan egyetlen betű kódja, amelynek a vétel helyén is ismert a megfejtése. Ismét tegyük fel, hogy már valameddig eljutott az adás, és egy újabb kódszó érkezik. Ha az előző kódszónak megfelelő string $a_1 \dots a_n$, és a most vett kódszó dekódolásában az első betű u, akkor az adónál a most elküldött kódszó előtt a kódtábla soron következő eleme az $a_1 \dots a_n u$ betűsorozatot fogja kódolni, vagyis a vevő is ezt a jelsorozatot rendeli a kódtábla soron kö-

betű vagy hossz	bináris kód	távolság	bináris kód
0...143	00110000...10111111	0...3	00000...00011
144...255	110010000...111111111	4...5	00100x
256...264	0000000...0001000	6...7	00101x
265...266	0001001x	8...11	00110xx
267...268	0001010x	12...15	00111xx
269...270	0001011x	16...23	01000xxx
271...272	0001100x	24...31	01001xxx
273...276	0001101xx	32...47	01010xxxx
277...280	0001110xx	48...63	01011xxxx
281...284	0001111xx	64...95	01100xxxxx
285...288	0010000xx	96...127	01101xxxxx
289...296	0010001xxx	128...191	01110xxxxxx
297...304	0010010xxx	192...255	01111xxxxxx
305...312	0010011xxx	256...383	10000xxxxxxx
313...320	0010100xxx	384...511	10001xxxxxxx
321...336	0010101xxxx	512...767	10010xxxxxxxx
337...352	0010110xxxx	768...1023	10011xxxxxxxx
353...368	0010111xxxx	1024...1535	10100xxxxxxxxx
369...384	11000000xxxx	1536...2047	10101xxxxxxxxx
385...416	11000001xxxxx	2048...3071	10110xxxxxxxxxx
417...448	11000010xxxxx	3072...4095	10111xxxxxxxxxx
449...480	11000011xxxxx	4096...6143	11000xxxxxxxxxxx
481...511	11000100xxxxx	6144...8191	11001xxxxxxxxxxx
512	11000101	8192...12287	11010xxxxxxxxxxxx
		12288...16383	11011xxxxxxxxxxxx
		16384...24575	11100xxxxxxxxxxxxx
		24576...32767	11101xxxxxxxxxxxxx

9.7. táblázat

vetkező eleméhez, és így a továbbiakban már ezt a kódot is tudja dekódolni. Egyetlen esetben lehet probléma a dekódolásnál. Ha a kódolásnál a már kódolt szöveg után az $a_1 \ldots a_n a_{n+1} \ldots a_{n+m}$ szöveg olyan, hogy $a_1 \ldots a_n$-nek már van kódja, de $a_1 \ldots a_n a_{n+1}$-nek még nincs, és $a_{n+1} \ldots a_{n+m} = a_1 \ldots a_n a_{n+1}$, akkor az $a_1 \ldots a_n$ kódjának elküldése után az adónál bekerül a kódtáblába $a_1 \ldots a_n a_{n+1}$ kódja, és így a következő kódolandó szöveghez, $a_1 \ldots a_n a_{n+1}$-hez már lesz bejegyzés, tehát tudjuk kódolni (de a következő betűvel meghosszabbított szöveg még biztosan nem kódolható, hiszen ellenkező esetben az algoritmusunk alapján már az előző lépésben tudtuk volna $a_1 \ldots a_n a_{n+1}$-et kódolni). Ugyanakkor a vétel helyén $a_1 \ldots a_n a_{n+1}$ kódját az előző, $a_1 \ldots a_n$-hez tartozó kód dekódolása, valamint a most beérkezett, az $a_1 \ldots a_n a_{n+1}$ kódjának visszafejtésekor kapott szöveg első betűje alapján tudnánk meghatározni, ami lehetetlen, hiszen az aktuális de-kódoláshoz a még ismeretlen, éppen most meghatározandó szöveget kellene ismerni. A dekódolás azonban mégis elvégezhető. Ha a vett kódszó még nem szerepel a vétel helyén a kódtáblában, az csak azért lehetséges, mert a kódolandó szöveg a fent leírt alakú, tehát $a_{n+1}\dots a_{n+m}=a_1\dots a_n a_{n+1}$, amiből viszont következik, hogy $a_{n+1}=a_1$, tehát a kódtáblában még nem található, és éppen most beérkezett, dekódolandó szöveg $a_1\dots a_n a_1$, így el tudjuk végezni a dekódolást, és tovább tudjuk építeni a vevőnél is a kódtáblát.

* **9.2.20. Példa az LZW-kódra.** Legyen a kód hexadecimális, a kódszavak hossza 2 (így a kódtábla maximális mérete 256 bejegyzés), és az alaphelyzetben használjunk ASCII-kódot. Például `c` kódja `63`, így a kódtáblában az egyik bejegyzés a (`c`, `63`) pár. Kódoljuk a `gugogogogol` szöveget.

Mivel `g` kódja `67`, ugyanakkor `gu` még nem szerepel a kódtáblában, így elküldjük a `67`-es kódszót, és a kódtáblába beírjuk (`gu`, `80`)-at. A továbbiakban az `ugogogogol` szöveget kell kódolnunk. Az `u` kódja ismert, de `ug` kódja nem. Küldjük `u` kódját, ami `75`, és a kódtáblába beírjuk az (`ug`, `81`) párt. Maradt a `gogogogol` szöveg. Most ismert `g` kódja, de ismeretlen a `go` szöveg, tehát megy a `67`-es kódszó, és a táblába beírjuk a (`go`, `82`) párt. Most az `ogogogol` szöveg bal szélénél tartunk. Tudjuk `o` kódját, de még ismeretlen az `og` kódja. A `6F`-es kód elküldésén kívül az (`og`, `83`) bejegyzéssel megadjuk `og` kódját. A `gogogol` szöveg maradt. Tudjuk `g` és `go` kódját, de még nincs kódja a `gog` hármasnak, így elküldjük `82`-t, és a táblába bekerül a (`gog`, `84`) pár. Következik a `gogol` kódolása. Már tudjuk kódolni `g`-t, `go`-t, `gog`-ot, de még nem találkoztunk `gogo`-val, így el tudjuk küldeni a `84`-es kódszót, és felvesszük a kódtáblába (`gogo`, `85`)-öt. Még `ol` van hátra. Ebből csak az `o`-t tudjuk kódolni, az elküldött kód `6F`, és bővül a kódtábla, ahová (`ol`, `86`) kerül. Végül egyetlen karakter, az `l` maradt, amelynek a kódja `6C`. Ezt elküldjük, és a táblába már semmit nem kell bejegyeznünk.

Most nézzük a dekódolást. Az érkező kódszavak sorozata `6775676F82846F6C`. Mivel `67`-ből `g`-t, `75`-ből `u`-t dekódolunk, a kódtáblába bejegyezzük a (`gu`, `80`) párt. Az `67`-ből ismét `g`-t dekódolunk, a táblába bekerül az (`ug`, `81`) pár. Mivel `6F` az `o` kódja, tehát az eddig visszafejtett szöveg `gugo`, és a táblába (`go`, `82`)-t írunk. Következik a `82` dekódolása. A kódtáblában a (`go`, `82`) bejegyzés található, így az eddig dekódolt teljes szöveg `gugogo`. Mivel az előző kódszó `6F`, míg a most dekódolt szöveg első karaktere `g` volt, ezért a kódtáblába (`og`, `83`) kerül. Most `84`-et kell dekódolnunk, ám a táblánkban ilyen kód nincs. Ebből arra következtetünk, hogy a `84` megfejtése az utolsó dekódolt szöveg, jobbról kiegészítve az előbb dekódolt karaktersorozat bal szélső karakterével, vagyis a konkrét esetben `84` megfejtése `gog`, és gyorsan beírjuk a kódtáblába a (`gog`, `84`) párt. A hátralévő kód fejtése már könnyű, az elsőé az `o`, és a kódtábla bővül a (`gogo`, `85`) beírással, majd `6C` megfejtése `l`, és a kódtáblába feljegyezzük az (`ol`, `86`) párt.

Az előbbi kódolást és dekódolást mutatja a 9.8. és 9.9. táblázat.

A 9.8. táblázatban az 1. oszlopban a még nem kódolt szöveg, a 2. oszlopban az adott menetben kódolt szövegrész található. A 3. oszlop mutatja az elküldött kódszót, míg a 4. a táblába kerülő bejegyzést.

A 9.9. táblázat 1. oszlopában az éppen vett kódszó látható, utána jön a tábla megfelelő bejegyzése, majd a dekódolt szövegrész. A 4. oszlopban található a tábla aktuális bejegyzése.

gugogogogol	g	67	(gu, 80)
ugogogogol	u	75	(ug, 81)
gogogogol	g	67	(go, 82)
ogogogol	o	6F	(og, 83)
gogogol	go	82	(gog, 84)
gogol	gog	84	(gogo, 85)
ol	o	6F	(ol, 86)
l	l	6C	

9.8. táblázat

67	(g, 67)	g	
75	(u, 75)	u	(gu, 80)
67	(g, 67)	g	(ug, 81)
6F	(o, 6F)	o	(go, 82)
82	(go, 82)	go	(og, 83)
84	(gog, 82)	gog	(gog, 84)
6F	(o, 6F)	o	(gogo, 85)
6C	(l, 6C)	l	(ol, 86)

9.9. táblázat

Programozásnál zavaró lehet, hogy a párok első koordinátája változó hosszúságú sztring. Ezt könnyen kiküszöbölhetjük úgy, hogy csak az utolsó betűt tároljuk, és egy pointert a prefixre, ami már a táblában van. Ha a kód egyenletes kód, a dekódolásnál a φ függvény inverzét táblázatban tárolhatjuk. A kódolás során a φ függvényt mint párok halmazát kezeljük. Ez hash-transzformációval valósítható meg hatékonyan.

* **9.2.21. A compress parancs.** LZW-típusú kódot használ a compress parancs. Az újdonság, hogy a kódhosszat dinamikusan változtatjuk. Kezdetben a kódhossz 9 bit, és a bájtok kódjait tartalmazza. Ha betelik a szótár, 10 bitre növeljük a kódhosszat. A kódhossz maximális értéke paraméterezhető, de maximum 16 bit. Ha elértük a maximális kódhosszat és betelt a szótár, a program akkor is figyeli, hogy mekkora tömörítést sikerült elérni. Ha ez egy adott érték alá megy, eldobjuk a szótárat, és új szótár építésébe kezdünk.

* **9.2.22. Digitalizálás.** Egy valós értékű függvényt, *analóg jel*et számítógépben közvetlenül nem tudunk reprezentálni, ezért mintavételezéssel sorozattá alakítjuk, majd a kapott valós számokat egész számokká alakítva kvantáljuk. A két lépés együtt a *digitalizálás*.

Egy $t \mapsto F(t)$ függvényt (amit legtöbbször az idő függvényének gondolunk, bár ez nem szükségszerű) *mintavételezés*sel alakíthatunk (mindkét irányba végtelen) sorozattá: legyen $f_k = F(\tau_0 + k\tau)$, ha $k \in \mathbb{Z}$, ahol $\tau_0 \in \mathbb{R}$ és $0 < \tau \in \mathbb{R}$ rögzítettek. A mintavételezés *periódusideje* τ, ennek reciproka, $1/\tau$ a *mintavételi frekvencia*. Természetesen, ha a jel csak egy véges intervallumon van értelmezve, akkor a minta is véges sorozat lesz.

Tekintsük példaként a $t \mapsto \sin\bigl(2\pi(t-\tau_0)/T\bigr)$ jelet, amelynek periódusa T (azaz $f(t+T) = f(t)$ minden $t \in \mathbb{R}$-re), frekvenciája így $1/T$. Ennek mintavételezésével az

$f_k = \sin(2\pi k/T)$, $k \in \mathbb{Z}$ sorozatot kapjuk. Észrevehetjük, hogy ha a jel frekvenciája a mintavételi frekvencia fele, azaz ha $2/T = 1/\tau$, akkor $f_k = \sin(\pi k) = 0$, ami megegyezik az azonosan nulla jel mintavételezésével kapott sorozattal. Ez azt mutatja, hogy a mintavételi frekvencia felénél nem kisebb frekvenciájú jelösszetevők a mintavételnél elveszhetnek. Ezért a mintavételi frekvencia az átvinni kívánt jel legmagasabb frekvenciájú összetevője frekvenciájának több mint kétszerese kell legyen: ez *Shannon mintavételi törvénye*.

Kétdimenziós $(x, y) \mapsto F(x, y)$ jelnél, például képnél kétdimenziós mintavételezést alkalmazunk:

$$f_{i,j} = F(\xi_0 + i\xi, \eta_0 + j\eta).$$

Háromdimenziós $(x, y, t) \mapsto F(x, y, t)$ jelnél, például mozgóképnél a mintavételezés is háromdimenziós:

$$f_{i,j,k} = F(\xi_0 + i\xi, \eta_0 + j\eta, \tau_0 + k\tau).$$

A mintavételezéssel kapott jel még nem tárolható a számítógépben: kerekítéssel digitalizálni kell. Általában egészre kerekítünk valamilyen szabály szerint. A kerekítés előtt rendszerint egy $v \mapsto q(v)$ *kvantálási függvény*t alkalmazunk a mintavételezéssel kapott értékekre. A kvantálási függvény gyakran lineáris, $q(v) = av + b$ alakú. Minél nagyobb a, annál kisebb változás elég v-ben, hogy másik egész számot kapjuk a kerekítés után, tehát annál finomabb a kvantálás. A b konstanssal például elérhetjük, hogy a kvantált értékek nemnegatívak legyenek, b tehát szinteltolást jelent.

A kvantálás sajnos azt is jelenti, hogy „zajt" viszünk be a jelbe. A hang (egységnyi felületre eső) teljesítménye a hangnyomás négyzetével arányos. A mikrofon a hangnyomással arányos feszültséget hoz létre, a teljesítmény ennek a négyzetével arányos. Fülünk érzékenysége logaritmikus jellegű: két hang között akkor érzünk ugyanolyan hangosságkülönbséget, ha teljesítményük aránya ugyanaz az érték. Ezért hangjel kvantálásánál logaritmikus jellegű kvantálási függvényt érdemes alkalmazni, mert ekkor ugyan hangosabb jelnél a zaj is hangosabb lesz, de a teljesítményük aránya ugyanaz marad, így egyformán érezzük zajosnak a különböző részeket. Például a telefontársaságok az USA-ban és Japánban a

$$v \mapsto a \operatorname{sgn}(v) \log\bigl(1 + \mu|v|\bigr)$$

függvény használják alkalmas a és μ választással. Az is lehet, hogy egy már (lineárisan, finoman) kvantált jelet kvantálunk újra nemlineáris kvantálással. (Például digitális telefonnál a fenti függvénnyel tulajdonképpen 14 bites lineárisan kvantált értéket kvantálnak újra, hogy 8 bites értéket kapjanak.) Természetesen amikor a digitalizált jelet visszaalakítjuk analóg jellé, a kvantálási függvény inverzét kell alkalmaznunk.

Általában a digitalizálás mindkét lépésében elvész az információ egy része, így általában minden digitalizálást használó kódolás eleve veszteséges.

Néhány tömörítő eljárás *vektorkvantálás*t használ: összetartozó mennyiségeket (például egy színes képpont színkoordinátáit) együtt vektornak tekintünk, és egy vektorokat tartalmazó kódtáblából keressük ki a hozzá legközelebbi vektort.

* **9.2.23. DFT és IDFT.** Tekintsünk egy $T > 0$ periódus szerint periódikus valós változós F függvényt (azaz $F(t + T) = f(t)$ minden $t \in \mathbb{R}$-re), legyen a mintavételi

frekvencia az $1/T$ „alapfrekvencia” n-szerese, ahol $n > 2$ egy természetes szám, azaz legyen $\tau = T/n$, és legyen $\tau_0 = 0$. A mintavételezéssel kapott f_j, $j \in \mathbb{Z}$ sorozat n szerint periodikus, azaz $f_{j+n} = f_j$ minden $j \in \mathbb{Z}$-re, így egy $\mathbb{Z}_n$-en értelmezett f függvénynek is tekinthető, vagyis az $f_0, f_1, \dots, f_{n-1}$ értékek egyértelműen jellemzik. Ezt a függvényt szeretnénk „eltolt szinuszos” jelek lineáris kombinációjaként előállítani. Tekintsük a $H_k(t) = \cos(2\pi kt/T) + i\sin(2\pi kt/T)$, $k \in \mathbb{Z}$ ugyancsak T szerint periódikus komplex értékű függvényeket. A H_k függvény mintavételezésével az ω_n^{jk}, $j \in \mathbb{Z}$ sorozatot kapjuk, ahol $\omega_n = \cos(2\pi/n) + i\sin(2\pi/n)$ az első n-edik egységgyök. Legyen $\hat{f}_k = \sum_{j=0}^{n-1} f_j\omega_n^{-jk}$ (azaz az

$$(f_0, f_1, \dots, f_{n-1})$$

és az

$$\left(\omega_n^0, \omega_n^k, \dots, \omega_n^{(n-1)k}\right)$$

$\mathbb{C}^n$-beli vektorok belső szorzata), ha $k \in \mathbb{Z}$. A $\hat{f}_k$, $k \in \mathbb{Z}$ sorozat is n szerint periodikus, hiszen $\omega_n^n = 1$, így

$$\hat{f}_{k+n} = \sum_{j=0}^{n-1} f_j\omega_n^{-j(k+n)} = \sum_{j=0}^{n-1} f_j\omega_n^{-jk} = \hat{f}_k.$$

Tehát az f sorozathoz az $\hat{f}$ sorozatot rendelő leképezés a $\mathbb{Z}_n$-en értelmezett komplex értékű függvények n-dimenziós komplex vektorterét önmagába képezi le. Ezt a leképezést nevezzük *diszkrét Fourier transzformáció*nak, *DFT*-nek. A diszkrét Fourier-transzformáció olyan fontos leképezés, hogy gyors végrehajtására integrált áramköröket gyártanak, számos műszaki alkalmazással. A DFT nyilván lineáris.

Vegyük észre, hogy

$$\sum_{j=0}^{n-1} (\omega_n^k)^j = \frac{(\omega_n^k)^n - 1}{\omega_n^k - 1}$$

0, ha $\omega_n^k \neq 1$, és nyilván n, ha $\omega_n^k = 1$. Ebből következik, hogy a $H_0, H_1, \dots, H_{n-1}$ függvények mintavételezésével kapott $h_k : \mathbb{Z}_n \to \mathbb{C}$ vektorok páronként ortogonálisak és mindegyiknek az önmagával vett belső szorzata n. Innen a

$$t \mapsto \frac{1}{n}\sum_{k=0}^{n-1} \hat{f}_k H_k(t)$$

függvény a mintavételezés $t = j\tau$ pontjaiban megegyezik az F függvénnyel. Speciálisan, a DFT invertálható, és „majdnem” saját maga az inverze: $t = j\tau$ helyettesítéssel

$$f_j = \frac{1}{n}\sum_{k=0}^{n-1} \hat{f}_k\omega_n^{jk} = \hat{\hat{f}}_{-j}$$

A DFT inverzét *inverz diszkrét Fourier transzformáció*nak, *IDFT*-nek nevezzük.

További hasznos észrevétel, hogy

$$\hat{\bar{f}}_k = \sum_{j=0}^{n-1} \bar{f}_j \omega_n^{-jk} = \overline{\sum_{j=0}^{n-1} f_j \omega_n^{jk}} = \overline{\hat{f}_{-k}}.$$

Így ha az f_j sorozat valós, akkor $\hat{f}_k = \hat{\bar{f}}_k = \overline{\hat{f}_{-k}}$. Valós f_j sorozat esetén az $f_0 H_0(t)$ tag konstans, ha pedig $0 < k < n/2$, $f_k = r(\cos\varphi + i\sin\varphi)$, akkor

$$\begin{aligned} f_k H_k(t) + f_{-k} H_{-k}(t) &= r(\cos\varphi + i\sin\varphi)\big(\cos(2\pi kt/T) + i\sin(2\pi kt/T)\big) \\ &\quad + r(\cos\varphi - i\sin\varphi)\big(\cos(2\pi kt/T) - i\sin(2\pi kt/T)\big) \\ &= 2r\cos(2\pi kt/T + \varphi), \end{aligned}$$

azaz egy $2r$ amplitúdóju koszinuszos jel φ fáziseltolással, tehát az eredeti jelet lényegében harmonikus jelek összegére bontottuk.

Végül vegyük még észre, hogy ha az f_j sorozat páros, akkor

$$f_j = f_{-j} = \frac{1}{n}\hat{\hat{f}}_j,$$

ahonnan

$$\hat{f}_k = \frac{1}{n}\hat{\hat{\hat{f}}}_k = \hat{f}_{-k},$$

azaz $\hat{f}$ is páros. Ha az f_j sorozat valós és páros, akkor a $\hat{f}_k$ sorozat páros és $\overline{\hat{f}_k} = \overline{\hat{\bar{f}}_k} = \hat{f}_{-k} = \hat{f}_k$, azaz valós is. Hasonlóan, ha az f_j sorozat páratlan, akkor

$$-f_j = f_{-j} = \frac{1}{n}\hat{\hat{f}}_j,$$

ahonnan

$$-\hat{f}_k = \frac{1}{n}\hat{\hat{\hat{f}}}_k = \hat{f}_{-k},$$

azaz $\hat{f}$ is páratlan. Ha az f_j sorozat valós és páratlan, akkor a $\hat{f}_k$ sorozat páratlan és $\overline{\hat{f}_k} = \overline{\hat{\bar{f}}_k} = \hat{f}_{-k} = -\hat{f}_k$, azaz képzetes.

* **9.2.24. FFT.** A gyors Fourier-transzformáció (Fast Fourier Transform, *FFT*) a diszkrét Fourier-transzformált gyors kiszámítására szolgál. Legyen n rögzített, és az egyszerűség kedvéért vezessük be az $\omega = \overline{\omega_n} = \omega_n^{-1}$ jelölést. Vegyük észre, hogy az előző pontban használt jelölésekkel $\hat{f}_k$ nem más, mint az $f^{(0)}(x) = \sum_{j=0}^{n} f_j x^j$ polinom $x_k = \omega^k$ helyen felvett értéke. Az $\hat{f}_k$ értékek gyors kiszámításához vezető trükk: ha $n = n_1 n_2$, akkor $y = x^{n_1}$ jelöléssel

$$f^{(0)}(x^{n_1 j_2 + j_1}) = \sum_{j_1=0}^{n_1-1} x^{j_1} f_{j_1}^{(1)}(y),$$

ahol

$$f_{j_1}^{(1)}(y) = \sum_{j_2=0}^{n_2-1} f_{n_1 j_2 + j_1} y^{j_2}, \qquad j_1 = 0, 1, \ldots, n_1 - 1.$$

Ha $x = \omega_n^{j_1}, \omega^{j_1+n_2}, \ldots, \omega^{j_1+(n_1-1)n_2}$, akkor y ugyanaz, így minden kiszámított $f_{j_1}^{(1)}(y)$ érték többször is felhasználható. Például, ha $n_1 = 2$, akkor az $f^{(0)}$ polinom ω^j és $\omega^{j+n/2}$ helyen felvett értékeit egyszerre számolhatjuk ki (azt is felhasználva, hogy $\omega^{n/2} = -1$) az alábbi pillangó művelettel:

$$\begin{aligned} f^{(0)}(\omega^k) &= f_0^{(1)}(\omega^{2k}) + \omega^k f_1^{(1)}(\omega^{2k}) \\ f^{(0)}(\omega^{k+n/2}) &= f_0^{(1)}(\omega^{2k}) - \omega^k f_1^{(1)}(\omega^{2k}) \end{aligned}$$

Például, ha $n = 8$, $n_1 = 2$, akkor $x = \omega^k$ és $x = \omega^{k+4}$ esetén y értéke ω^{2k}, így az

$$f_0 + f_2 y + f_4 y^2 + f_6 y^3$$

és

$$f_1 + f_3 y + f_5 y^2 + f_7 y^3$$

értékeket kétszer is tudjuk használni, így a munkát nagyjából megfeleztük. Természetesen rekurzívan is alkalmazhatjuk ezt a trükköt. Az alábbi, együtthatóikkal adott polinomok értékeit kell kiszámolnunk a megadott helyeken:

$$\begin{pmatrix} f_0, & f_1, & f_2, & f_3, & f_4, & f_5, & f_6, & f_7 \\ \omega^0, & \omega^1, & \omega^2, & \omega^3, & \omega^4, & \omega^5, & \omega^6, & \omega^7 \end{pmatrix}$$

$$\begin{pmatrix} f_0, & f_2, & f_4, & f_6 \\ \omega^0, & \omega^2, & \omega^4, & \omega^6 \end{pmatrix} \qquad \begin{pmatrix} f_1, & f_3, & f_5, & f_7 \\ \omega^0, & \omega^2, & \omega^4, & \omega^6 \end{pmatrix}$$

$$\begin{pmatrix} f_0, & f_4 \\ \omega^0, & \omega^4 \end{pmatrix} \quad \begin{pmatrix} f_2, & f_6 \\ \omega^0, & \omega^4 \end{pmatrix} \quad \begin{pmatrix} f_1, & f_5 \\ \omega^0, & \omega^4 \end{pmatrix} \quad \begin{pmatrix} f_3, & f_7 \\ \omega^0, & \omega^4 \end{pmatrix}$$

$$\begin{pmatrix} f_0 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_4 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_2 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_6 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_1 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_5 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_3 \\ \omega^0 \end{pmatrix} \begin{pmatrix} f_7 \\ \omega^0 \end{pmatrix}$$

A fentiek alapján rekurzív programot írhatunk a diszkrét Fourier-transzformáció elvégzésére. Az $n = 2^m$ esetben célszerűbb azonban a rekurziót ciklussá kibontani. Vezessük be az $[d_s d_{s-1} \ldots d_2 d_1] = \sum_{j=1}^{s} d_j 2^{j-1}$ jelölést. Legyen

$$f_{j_1, j_2, \ldots, j_s}^{(s)} = \sum_{j=0}^{2^{m-s}-1} f_{[j j_s j_{s-1} \ldots j_2 j_1]} x^j, \quad \text{ha} \quad j_1, \ldots, j_s \in \{0, 1\}.$$

Az $f^{(m)}_{j_1,j_2,\ldots,j_m}$ polinomok konstansok, értékük $f_{[j_m j_{m-1}\ldots j_2 j_1]}$. Az

$$\begin{aligned} f^{(s)}_{j_1,j_2,\ldots,j_s}(\omega^{k2^s}) &= f^{(s+1)}_{j_1,j_2,\ldots,j_s,0}(\omega^{k2^{s+1}}) + \omega^{k2^s} f^{(s+1)}_{j_1,j_2,\ldots,j_s,1}(\omega^{k2^{s+1}}) \\ f^{(s)}_{j_1,j_2,\ldots,j_s}(\omega^{k2^s+n/2}) &= f^{(s+1)}_{j_1,j_2,\ldots,j_s,0}(\omega^{k2^{s+1}}) - \omega^{k2^s} f^{(s+1)}_{j_1,j_2,\ldots,j_s,1}(\omega^{k2^{s+1}}) \end{aligned}$$

pillangó-művelet segítségével rendre az $s = m-1, m-2, \ldots, 1, 0$ értékekre számítjuk ki a függvényértékeket. Egyetlen n elemű A tömböt használunk, $f^{(s)}_{j_1,j_2,\ldots,j_s}(\omega^{2^s[k_{m-s}k_{m-s-1}\ldots k_2 k_1]})$ értéke $A[k_1 k_2 \ldots k_{m-s} j_1 j_2 \ldots j_s]$-be kerül, ahol $k_1, k_2, \ldots, k_{m-s}, j_1, \ldots, j_s \in \{0,1\}$. Így az alábbi algoritmust kapjuk:

* **9.2.25. FFT algoritmus.** Az előző pont jelöléseivel, az algoritmus az

$$A[j_m j_{m-1} \ldots j_2 j_1] = f_{[j_m j_{m-1}\ldots j_2 j_1]}, \quad j_1, \ldots, j_m \in \{0,1\}$$

sorozat Fourier transzformáltját számolja ki. Az eredmény az A tömbben keletkezik, de

$$\hat{f}_{[k_m k_{m-1}\ldots k_2 k_1]} = A[k_1 k_2 \ldots k_{m-1} k_m], \quad \text{ha} \quad k_1, \ldots, k_m \in \{0,1\}.$$

A számítás használ egy T segédtáblázatot, amely az ω hatványait tartalmazza, alkalmas sorrendben:

$$T[j_{m-1} \ldots j_2 j_1] \leftarrow \omega^{[j_1 j_2 \ldots j_{m-1}]}, \quad \text{ha} \quad j_1, \ldots, j_m \in \{0,1\}.$$

(1) [Inicializálás.] Legyen $l \leftarrow 2^{m-1}$.

(2) [Menet kezdete.] Legyen $j \leftarrow 0$ és $t \leftarrow 0$.

(3) [Pillangósorozat kezdete.] Legyen $k \leftarrow j + l$ és $w \leftarrow T[t]$.

(4) [Pillangó.] Legyen $x \leftarrow A[j]$ és $y \leftarrow wA[j+l]$, majd legyen $A[j] \leftarrow x + y$ és $A[j+l] \leftarrow x - y$, végül legyen $j \leftarrow j + 1$.

(5) [Pillangósorozat vége?] Ha $j < k$, menjünk vissza (4)-re.

(6) [Menet vége?] Ha $k + l < n$, legyen $j \leftarrow k + l$, $t \leftarrow t + 1$ és menjünk vissza (3)-ra.

(7) [Vége?] Legyen $l \leftarrow \lfloor l/2 \rfloor$. Ha $l > 0$, menjünk vissza (2)-re, egyébként az algoritmus véget ért.

Az eredmény kiolvasásához, illetve a T tábla feltöltéséhez szükséges „bitfordítás” eltolásokkal történhet: a számot balra tolva, egyenként megkapjuk bitjeit, és azokat jobbra tolással összerakjuk fordított sorrendben. Még egyszerűbb a következő szám bitfordítottját az előző bitfordítottjából számítani, a legnagyobb helyiértékű bithez 1-et hozzáadva, és az átvitelt az alacsonyabb helyiértékek (sic!) felé végezve.

Vegyük észre, hogy ugyanaz a T táblázat különböző m értékekre is megfelel, ha a maximális m értékre számoljuk ki.

* **9.2.26. IFFT algoritmus.** Az előző pontban megadott algoritmus megfordítható: ha a meneteket fordított sorrendben hajtjuk végre, $l = 1, 2, \ldots, 2^{m-1}$-re, és a (4) pillangó műveletet invertáljuk, akkor a transzformáció inverzét kapjuk:

(4') [Inverz pillangó.] Legyen $x \leftarrow A[j]$, $y \leftarrow A[j+l]$, majd $A[j] \leftarrow (x+y)/2$, $A[j+l] \leftarrow (x-y)/(2w)$, végül $j \leftarrow j+1$.

A felhasznált T táblázat ugyanaz. A kettővel való osztásokat elhagyhatjuk, ha az egész algoritmus elején vagy végén az A tömb minden elemét osztjuk 2^m-mel. Vegyük észre azt is, hogy $1/w = \overline{w}$, így nincs szükség osztásra, szorzással is dolgozhatunk.

* **9.2.27. Gyors szorzás FFT-vel.** Legyen $f = \sum_{j=0}^{n-1} f_j x^j$, $g = \sum_{j=0}^{n-1} g_j x^j$, és h az f és g polinomok szorzata. Mivel $\hat{f}_k = f(\omega_n^{-k})$, $\hat{g}_k = g(\omega_n^{-k})$, azt kapjuk, hogy $h(\omega_n^{-k}) = \hat{f}_k \hat{g}_k$, vagyis ha h fokszáma kisebb, mint n, azaz $h = \sum_{j=0}^{n-1} h_j x^j$, akkor $\hat{h}_k = \hat{f}_k \hat{g}_k$, $0 \le k < n$. Innen h meghatározható IFFT-vel. Így gyors algoritmust kaptunk polinomszorzásra.

A q alapú számrendszerben felírt $f(q) = \sum_{j=0}^{n-1} f_j q^j$ és $g(q) = \sum_{j=0}^{n-1} g_j q^j$ számok szorzata $h(q) = \sum_{j=0}^{n-1} h_j q^j$, ha a megfelelő f és g polinomok szorzata az n-nél alacsonyabb fokú $h = \sum_{j=0}^{n-1} h_j x^j$ polinom. Így gyors algoritmust kaptunk számok szorzásra.

* **9.2.28. DCT és IDCT.** A mérnöki gyakorlatban jobban szeretnek valós számsorozathoz valós transzformált sorozatot rendelni. Tekintsünk egy valós változós valós értékű $2T$ szerint periodikus páros F függvényt. Legyen a mintavételi frekvencia az $1/(2T)$ „alapfrekvencia” $2n$-szerese, ahol $n > 2$ egy természetes szám, azaz legyen $\tau = T/n$, és legyen $\tau_0 = \tau/2$. A mintavételezéssel kapott f_j, $j \in \mathbb{Z}$ sorozatot az $f_0, f_1, \ldots, f_{n-1}$ értékek egyértelműen jellemzik, mert a sorozat $2n$ szerint periodikus, azaz $f_{j+2n} = f_j$ minden $j \in \mathbb{Z}$-re, és $f_{-j-1} = f_j$, ha $0 \le j < n$. A F függvényt szeretnénk „koszinuszos” jelek lineáris kombinációjaként előállítani. Tekintsük a $H_k(t) = \cos(\pi k t/T)$, $k \in \mathbb{Z}$ ugyancsak $2T$ szerint periódikus páros függvényeket. A H_k függvény mintavételezésével a

$$\cos\frac{\pi k(j+1/2)}{n}, \quad j \in \mathbb{Z}$$

sorozatot kapjuk. Legyen

$$c_k = \sum_{j=0}^{n-1} f_j \cos\frac{\pi k(j+1/2)}{n}$$

(azaz az $(f_0, f_1, \ldots, f_{n-1})$ és a

$$\left(\cos\frac{\pi k}{2n}, \cos\frac{3\pi k}{2n}, \ldots, \cos\frac{(2n-1)\pi k}{2n}\right)$$

$\mathbb{R}^n$-beli vektorok belső szorzata), ha $k \in \mathbb{Z}$. Az

$$(f_0, f_1, \ldots, f_{n-1}) \mapsto (c_0, c_1, \ldots, c_{n-1})$$

leképezés $\mathbb{R}^n$-et önmagába képezi le. Ezt a leképezést nevezzük *diszkrét koszinusz transzformáció*nak, *DCT*-nek. A DCT nyilván lineáris.

Vegyük észre, hogy

$$\sum_{j=0}^{2n-1}(\omega_{4n}^k)^{2j+1}=\omega_{4n}^k\frac{(\omega_{4n}^{2k})^{2n}-1}{\omega_{4n}^{2k}-1}=0,$$

ha $-2n<k<2n$, $k\neq 0$. Mivel a koszinusz páros és periodikus,

$$\begin{aligned}\sum_{j=0}^{n-1}\cos\frac{\pi k(j+1/2)}{n}&=\frac{1}{2}\sum_{j=-n}^{n-1}\cos\frac{\pi k(j+1/2)}{n}=\frac{1}{2}\sum_{j=-n}^{n-1}\cos\frac{\pi k(j+1/2)}{n}\\&=\frac{1}{2}\sum_{j=0}^{2n-1}\cos\frac{\pi k(j+1/2)}{n}=\frac{1}{2}\sum_{j=0}^{2n-1}\cos\frac{\pi k(j+1/2)}{n}\\&=\sum_{j=0}^{2n-1}\left(\omega_{4n}^{k(2j+1)}+\omega_{4n}^{-k(2j+1)}\right)=0,\end{aligned}$$

ha $-2n<k<2n$ és $k\neq 0$. Ebből következik, hogy a $H_0,H_1,\ldots,H_{n-1}$ függvények mintavételezésével kapott $h_k\in\mathbb{R}^n$ vektorok belső szorzata

$$\begin{aligned}\langle h_{k_1},h_{k_2}\rangle&=\sum_{j=0}^{n-1}\cos\frac{\pi k_1(j+1/2)}{n}\cos\frac{\pi k_2(j+1/2)}{n}\\&=\frac{1}{2}\sum_{j=0}^{n-1}\Big(\cos\frac{\pi(k_1+k_2)(j+1/2)}{n}+\cos\frac{\pi(k_1-k_2)(j+1/2)}{n}\Big),\end{aligned}$$

azaz ha $0\leq k_1,k_2<n$, $k_1\neq k_2$, akkor 0, ha $0=k_1=k_2$, akkor n, ha pedig $0\neq k_1=k_2<n$, akkor $n/2$. Innen a

$$t\mapsto\frac{c_0}{n}+\frac{2}{n}\sum_{k=0}^{n-1}c_kH_k(t)$$

függvény a mintavételezés $t=\tau/2+j\tau$ pontjaiban megegyezik az F függvénnyel. Speciálisan, a DCT invertálható: $t=\tau/2+j\tau$ helyettesítéssel

$$f_j=\frac{c_0}{n}+\frac{2}{n}\sum_{k=0}^{n-1}c_k\cos\frac{\pi k(j+1/2)}{n}.$$

A DFT inverzét *inverz diszkrét koszinusz transzformáció*nak, *IDCT*-nak nevezzük.

További hasznos észrevétel, hogy a DCT kifejezhető a DFT-vel: Legyen $g_{2j+1}=f_j$ és $g_{2j}=0$, ha $j\in\mathbb{Z}$. Ekkor a g sorozat valós, páros, és $4n$ szerint periodikus. A diszkrét

Fourier-transzformáltja így valós és páros, tehát

$$\begin{aligned}
\hat{g}_k &= \frac{\hat{g}_k + \hat{g}_{-k}}{2} = \frac{1}{2}\sum_{j=0}^{4n-1} g_j(\omega_{4n}^{-jk} + \omega_{4n}^{jk}) \\
&= \sum_{j=0}^{2n-1} f_j \frac{\omega_{4n}^{-(2j+1)k} + \omega_{4n}^{(2j+1)k}}{2} \\
&= \sum_{j=0}^{2n-1} f_j \cos\frac{\pi k(j+1/2)}{n} = \sum_{j=-n}^{n-1} f_j \cos\frac{\pi k(j+1/2)}{n} \\
&= 2\sum_{j=0}^{n-1} f_j \cos\frac{\pi k(j+1/2)}{n} = 2c_k.
\end{aligned}$$

* **9.2.29. Kétdimenziós DFT és DCT.** A kétdimenziós DFT mátrixhoz mátrixot rendel. Definíciója:

$$\hat{f}_{k_1,k_2} = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1} \omega_{n_1}^{-j_1k_1}\omega_{n_2}^{-j_2k_2} f_{j_1,j_2}, \quad \text{ha} \quad 0 \le k_1 < n_1,\ 0 \le k_2 < n_2.$$

A definícióból

$$\hat{f}_{k_1,k_2} = \sum_{j_1=0}^{n_1-1} \omega_{n_1}^{-j_1k_1} \sum_{j_2=0}^{n_2-1} \omega_{n_2}^{-j_2k_2} f_{j_1,j_2} = \sum_{j_2=0}^{n_2-1} \omega_{n_2}^{-j_2k_2} \sum_{j_1=0}^{n_1-1} \omega_{n_1}^{-j_1k_1} f_{j_1,j_2},$$

azaz a kétdimenziós DFT kiszámolható úgy, hogy előbb a mátrix soraira, majd az oszlopaira (vagy fordítva) alkalmazunk egydimenziós DFT-t.

A kétdimenziós DCT definíciója

$$c_{k_1,k_2} = \sum_{j_1=0}^{n_1-1}\sum_{j_2=0}^{n_2-1} \cos\frac{\pi k_1(j_1+1/2)}{n_1}\cos\frac{\pi k_2(j_2+1/2)}{n_2} f_{j_1,j_2}, \quad \text{ha} \quad 0 \le k_1 < n_1,\ 0 \le k_2 < n_2.$$

Ez is kiszámolható úgy, hogy előbb a mátrix soraira, majd az oszlopaira (vagy fordítva) alkalmazunk egydimenziós DCT-t.

Természetesen mindkét transzformáció általánosítható magasabb dimenziós tömbökre is.

* **9.2.30. Hangtömörítés.** A digitalizált hangjel tömörítésére általában veszteséges eljárásokat használnak. Az alapgondolat diszkrét Fourier transzformáció felhasználása a hangjel transzformálására az „időtartományból” a „frekvenciatartományba”, azaz a jel felbontása eltolt szinuszos jelek összegére. Mivel egy zenei hang csak néhány szinuszos

jel összege, a transzformáció után nagyon jól tömöríthető. Mivel a fülünk nem érzékeny a szinuszos jelek eltolására, ez az információ is eldobható. További tömörítést tesz lehetővé, hogy egy erős hang mellett megszólaló közeli frekvenciájú, de gyengébb hangot nem érzékelünk, így a közeli frekvenciájú hangok jóval durvábban kvantálhatók, mert a kvantálási zajt nem halljuk. Hasonlóan, röviddel az erős hang előtt vagy után megszólaló hasonló frekvenciájú gyengébb hangot sem hallunk. Mindezek jelentős tömörítést tesznek lehetővé. Durván a tömörítést úgy képzelhetjük, hogy a hangot DFT-vel folyamatosan analizáljuk, a kapott frekvenciaspekrumból kihagyjuk amit úgysem hallanánk, majd prefix kód és más kódolási eljárások segítségével a maradékot tömörítjük. Visszajátszásnál a dekódolt jelet folyamatosan visszatranszformáljuk: ez nagyjából annak felel meg, mintha egy nagyon jó szintetizátoron játszanánk vissza a jelet. A vázolt eljárás természetesen korán sem ilyen egyszerű a gyakorlatban. A szabványok általában csak a kódot adják meg pontosan. A kódoló és dekódoló szoftver illetve hardver tervezése nagyon nehéz hangmérnöki feladat.

A jelenlegi egyik legkorszerűbb eljárás az AAC (Advanced Audio Coding). Ennek különböző változatai többek közt a mozgóképtömörítésre (lásd ott) kifejlesztett MPEG-2 illetve MPEG-4 kódolás hangrészét képezik. Az eljárás az MPEG-1 Audio Layer III eljárás (elterjedt nevén MP3) továbbfejlesztésének tekinthető. DFT helyett a DCT egy módosított változatát használja 2048 alapponttal, vagy gyorsan változó hangokat tartalmazó jel (általában beszéd) esetén 256 alapponttal. A mintavételi frekvencia 8 kHz és 96 kHz között mozoghat, a jel 1-48 csatornás lehet, és beállítható a kódolt jelfolyam sebessége 2 kbit/s-tól (beszéd) 64 kbit/s-ig vagy feljebb csatornánként, ami már jó minőségű zeneátvitelt tesz lehetővé. Ha kis sebességet kívánunk, a jelveszteség természetesen megnő, a minőség romlik. A kódolás általában nagyon processzorigényes, a dekódolás egyszerűbb.

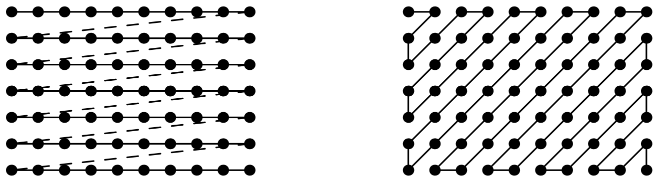
Elsősorban beszéd valós idejű átvitelére kifejlesztett eljárás az iLBC (Internet Low Bit Rate Codec). A mintavételi frekvencia 8 kHz, a kvantálás 16 bites. A sebesség rögzített, a kódolás és a dekódolás is gyors. Az alapelv más: az átvitt jelfolyam lényegében egy beszédszintetizátor működtetéséhez szükséges jel. Az eljárás jól tűri a kieső csomagokat. A pontos dokumentáció: RFC 3951. Az iLBC-t használja a Googletalk.

* **9.2.31. Képtömörítés.** Képtömörítésnél felhasználjuk azt, hogy a közeli képelemek között erős a függőség. Ezt bármilyen szokásos tömörítő eljárással jobban kihasználhatjuk, ha az egyszerű sorfolytonos bejárás helyett a még mindig nagyon egyszerű cikcakk bejárást alkalmazzuk: lásd a 9.7. ábrát.

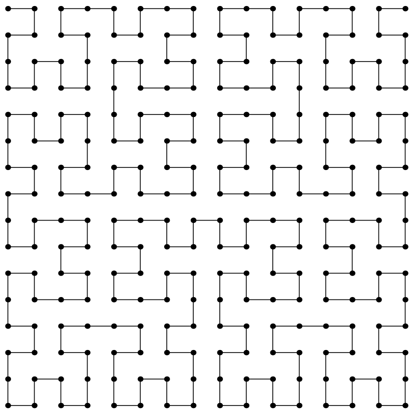
További javulás érhető el, ha sokkal speciálisabb bejárás választunk, például a Hilbert-görbén alapuló, a 9.8. ábrán látható bejárást.

Igazán hatékony tömörítést azonban csak a speciálisan képtömörítésre kifejlesztett eljárások adnak. Három elterjedt képtömörítésel fogunk megismerkedni.

* **9.2.32. PNG.** A PNG (ejtsd: ping; Portable Network Grafics) fájlformátumot digitalizált képek veszteségmentes átvitelére tervezték. Csak a tömörítéssel kapcsolatos részletekkel foglalkozunk (a specifikáció 1.0 verziója alapján) de annyit megjegyzünk, hogy a fájl olyan, elengedhetetlenül szükséges információkon kívül, mint például a sorok és oszlopok hossza és a kép típusa, nagyon sok további, az eredeti képpel kapcsolatos



9.7. ábra: sorfolytonos és cikcakk bejárás.



9.8. ábra: Hilbert-bejárás.

információt tartalmaz, például gamma, a fehérpont és az alapszínek kromacitásai, fizikai képelem (pixel) dimenziók, stb. A további részleteket lásd a specifikációban. Nem színes képek esetén az egyes képelemek kódja 1, 2, 4, 8 vagy 16 bit lehet. Színes képek RGB-formátumban adhatók meg (Red, Green, Blue, azaz vörös, zöld, kék alapszínek intenzitását megadva), 8–8–8, illetve 16–16–16 biten. Mindkét esetben megadhatunk egy opacitás értéket is, ugyanannyi biten. Ha ez maximális, a kép nem átlátszó, egyébként átlátszik a háttér, nullánál már csak a háttér látszik. Az egy képelemhez tartozó összes érték egymás után jön. A kép „palettával" is megadható: ez egy előre megadott színtáblázat, ekkor a képelemek helyén a táblaindexek vannak.

A sorfolytonosan olvasott kép minden sorát külön-külön egy „jóslás" segítségével jobban tömöríthetővé konvertáljuk. A jósláshoz a képelem feletti képelem megfelelő bájtját, valamint az előző megfelelő bájtot és az a feletti bájtot használhatjuk. Úgy kell tekinteni, hogy az első oszlop előtt és az első sor felett csupa nulla van. A fájlba mindig a valódi bájt és a jósolt bájt különbsége kerül. Ha a jóslási mód kódja 0, a jósolt érték nulla, nincs jóslás; ha 1, akkor a jóslás a, az előző megfelelő bájt értéke; ha 2, akkor a képelem feletti képelem megfelelő bájtja, b a jóslás; ha 3, akkor $\lfloor (a+b)/2 \rfloor$ a jóslás; végül, ha 4, akkor a Paeth-jóslást használjuk: legyen a képelem feletti képelem előtti megfelelő bájt c; a jóslás a, b és c közül az, amelyik legközelebb van $a+b-c$-hez. A jóslás segítségével konvertált sorokat egymás után írjuk és gzip-ben is használt „deflate” tömörítéssel tömörítjük.

Lehetőség van „előhívódó” tömörítésre is: a képet a bal felső sarokból kiindulva (maximum) 8×8 képelemes részekre osztjuk, majd a

$$\begin{array}{cccccccc}
1 & 6 & 4 & 6 & 2 & 6 & 4 & 6 \\
7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
5 & 6 & 5 & 6 & 5 & 6 & 5 & 6 \\
7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
3 & 6 & 4 & 6 & 3 & 6 & 4 & 6 \\
7 & 7 & 7 & 7 & 7 & 7 & 7 & 7 \\
5 & 6 & 5 & 6 & 5 & 6 & 5 & 6 \\
7 & 7 & 7 & 7 & 7 & 7 & 7 & 7
\end{array}$$

táblázattal megadott sorrend szerint olvassuk: először az egész képből az 1-esek helyén álló képelemeket, majd a 2-esek helyén álló képelemeket, stb. Dekódolásnál amikor az 1-es helyén álló képelem megérkezik, az egész (maximum) 8×8-as blokkot ennek megfelelő színűre festjük. Amikor a 2-es helyén álló képelem megérkezik, a blokk jobb felét átfestjük a megfelelő színre, stb.

A PNG formátum a GIF (Graphics Interchange Format) fájl formátum továbbfejlesztésének tekinthető, amely csak szürkeárnyalatos és „paletta”-színes képeket kezel, nem használ jóslást, és a compress parancshoz nagyon hasonló LZW-típusú tömörítést használ.

* **9.2.33. JPEG.** A Joint Photographic Experts Group által kidolgozott szabványból csak a digitalizált képek veszteséges tömörítésére kidolgozott változat, annak is az alapverziója terjedt el. (A veszteségmentes változat a PNG-hez hasonló.) Csak az egyik (veszteséges) alapváltozatot ismertetjük vázlatosan, a sokféle paraméterezési lehetőséget nem. Jó minőségű képeknél 10–20-szoros tömörítés érhető el.

A tömörítés első lépése — csak színes képeknél — az RGB koordináták transzformációja egy másik koordinátarendszerbe az

$$\begin{aligned}
Y &= 77R/256 + 150G/256 + 29B/256, \\
C_b &= -44R/256 - 87G/256 + 131B/256 + 128, \\
C_r &= 131R/256 - 110G/256 - 21B/256 + 128
\end{aligned}$$

transzformációval. Ez a koordináta-rendszer azért előnyösebb, mert az Y fizikai jelentése a képpont világossága, C_b illetve C_r jelentése pedig, hogy a szín a fehértől mennyire tér el a kék illetve vörös felé. A szemünk sokkal érzékenyebb a világosság változásaira, mint a színárnyalatok változásaira, így az Y „világosság” koordinátát nagyobb pontossággal fogjuk átvinni, mint a C_b és C_r „színesség” koordinátákat.

Második lépésként színes képnél — csak vízszintesen vagy függőlegesen is — felére csökkentjük a C_b és C_r képek felbontását. A továbbiakban a három képet egymás után, függetlenül kezeljük. (Persze, ha a kép nem színes, csak az Y kép van.) Mindegyiket 8×8 képelem méretű blokkokra bontjuk, szükség szerint az utolsó ismétlésével további oszlopokkal és sorokkal egészítve ki.

Minden blokkra kétdimenziós diszkrét koszinusz transzformációt alkalmazunk. A kapott transzformáltakat lineárisan kvantáljuk, de a kvantálás a $c_{j,k}$ együtthatóra (nagyjából) $j + k$ növekedésével egyre durvul. A kvantálás finomságát az Y képre, illetve a C_b és C_r képekre táblázat adja meg. Általánosságban azt mondhatjuk, hogy a kvantálás finomabb az Y, mint a C_b és C_r képre. A kvantálás után az együtthatók mátrixában általában csak néhány nem nulla elem marad.

Először sorfolytonosan olvasva a $c_{0,0}$ együtthatókat visszük át. Ez az együttható az adott blokk átlagát reprezentálja. Mivel általában a szomszédos blokkok átlaga nem nagyon tér el, az első blokk együtthatója után a többi blokkra csak az együttható előzőtől való eltérését visszük át. Ezután a többi együttható átvitele következik: az együtthatók mátrixát cikcakkban járjuk be, majd a következő blokkra térünk át, sorfolytonosan. Az átvitel előtt futamkódolást, majd prefix kódot alkalmazunk.

A dekódolás a kódolás fordítottja. Nyilván alkalmazható „előhívódó” dekódolás: már az Y kép $c_{0,0}$ együtthatói egy durva és nem színes képet adnak.

* **9.2.34. DjVu.** Az AT&T terméke a DjVu (ejtsd: dézsa vü) képtömörítő módszer. Igen nagy, szkennelt képek esetében akár 1000-szeres tömörítést lehet vele elérni. Az alapgondolat, hogy a képet három képre bontjuk fel: „háttér”, „előtér” és „maszk”. A háttér és az előtér árnyalatos és általában színes képek, a maszk pedig egy bináris kép, bitjei azt mondják meg, hogy az adott pont a háttérhez vagy az előtérhez tartozik. Mindhárom kép külön-külön jól tömöríthető: a háttér és az előtér felbontását vízszintesen és függőlegesen is általában harmadára csökkenthetjük a minőség romlása nélkül, és mindkettő nagy homogén területeket tartalmaz, míg a maszk bináris, ezért tömöríthető jól. A háttér és az előtér tömörítésére más eljárást használunk, mint a maszk tömörítésére. Az algoritmus, amely a képet háttérre, előtérre és maszkra bontja, az alábbi:

Kezdetben tekintsük az egész képet egy blokknak, a háttér színe legyen kezdetben fehér, az előtéré fekete, majd minden képpontot soroljunk be annak megfelelően, hogy a színe a háttéréhez vagy az előtéréhez áll-e közelebb, a kettő közül az egyikhez: ez adja a maszkot. Az új háttér szín a háttérhez tartozó képpontok színének átlaga, míg az előtér színe az előtérhez tartozó képpontok színének átlaga. Addig ismételjük az eljárást, amíg a háttér, illetve az előtér színe stabilizálódik (azaz már elegendően keveset változik). Ez az első lépés. Már ez a lépés felbontja a képet, ha csak két szín van, de általában nem ez a helyzet.

Osszuk fel a blokkot kisebb blokkokra, majd minden blokkra ismételjük meg a fenti eljárást, de most a nagyobb blokk háttér és előtér színével indulva, és a kisebb blokk háttér illetve előtér színét úgy számolva, hogy 0,2 súllyal a nagyobb blokk, 0,8 súllyal pedig az oda sorolt képelemek színét vesszük figyelembe. Ismételjük ezt a lépést is, amíg a háttér és az előtér színe stabilizálódik. Ez a második lépés.

Minden blokkot kisebb blokkokra osztva, ismételjük meg minden kisebb blokkra a második lépést. Ezt addig folytatjuk, amíg elég finom nem lesz a felosztás.

Az, hogy a második lépésben mindig figyelembe vesszük a nagyobb blokk háttér illetve előtér színét is, akkor válik fontossá, ha egy kisebb blokk csak előtér, vagy csak háttér pontokat tartalmaz. Ekkor a súlyozás nem engedi, hogy az előtér, illetve a háttér színe „elmásszon”.

A pontos algoritmus további „szűrőket” tartalmaz, amelyek a fenti alapalgoritmus hibáit küszöbölik ki. A dekódolás nyilvánvaló.

* **9.2.35. Mozgókép-tömörítés: MPEG.** Mozgókép-tömörítésre általánosan elterjedt az MPEG (Moving Pictures Experts Group) eljárás 1-es változata, és terjed a 2-es és 4-es változat is. Csak az MPEG-1-ben használt alapelveket ismertetjük vázlatosan.

Nyilván egy filmnél vagy videófelvételnél két egymás utáni kép (frame) általában nagyon hasonlít egymásra, legtöbbször csak elmozdulnak egy kicsit a képrészletek. Az első képet mint állóképet visszük át. A további képek átviteléhez az előző képet használjuk „jóslásra”: a képet kis részekre bontjuk, majd megkeressük a leginkább hasonló képrészletet az előző képen, átvisszük az elmozdulásvektort, és a képrészletek különbségét.

Ha minden további képre ezt az eljárást használnánk, a felhalmozódó hibák miatt a képek hamar használhatatlanná válnának. Ezért néhány megjósolt (predicted, P) kép után újra egy teljes bemenő (intra, I) képet kell átvinnünk. További tömörítés érhető el, ha észrevesszük, hogy bizonyos képrészletek csak egy későbbi képből jósolhatók, például egy mozgó tárgy mögött feltűnő háttér. Az I illetve P képek közé tehát kétirányú (bidirectional, B) képeket célszerű beiktatni, amelyek részleteit egy előttük vagy mögöttük lévő I illetve P kép segítségével jósoljuk. Természetesen ezeket csak akkor lehet kódolni illetve dekódolni, ha már rendelkezésre áll nemcsak az előttük, hanem a mögöttük lévő P vagy I kép is. Tehát a képek sorozata például a következő lehet: IBBBPBBBPBBBPBBBIBBBPB....

Konkrétabban, az MPEG-1 szabvány szerint a hang és kép továbbítása lényegében függetlenül, bár szinkronizálva történik. A hangátvitellel már foglalkoztunk. A képátvitel legkisebb egysége a makróblokk. Ez 16×16 képelem világosság adatait tartalmazza négy 8×8-as blokkra bontva, valamint mindkét irányban fele akkora felbontással a kétféle színességi adatot két 8×8-as blokként. Az I képek átvitele a JPEG-nél megismerthez nagyon hasonlóan történik, ezt nem részletezzük.

A P képek átvitelénél néhány makróblokk lehet I típusú, ezeket úgy kódoljuk, mint az I képeknél. A makróblokkok nagy része azonban P típusú: tartozik hozzá egy elmozdulásvektor, és csak az előző I vagy P kép így megadott részétől való eltérés kódoljuk. A kódolás egyébként hasonló, mint az I típusú makróblokknál, de a kvantálási tábla más. Mivel az elmozdulásvektorok nagyon hasonlóak, csak a különbségüket tároljuk, azaz a jóslás mindig az előző makróblokk elmozdulásvektora: A kép sorfolytonosan olvasott makróblokkjainak folyamát szeletnek nevezett részekre bontjuk. A szelet elején, és még néhány más esetben, például I típusú makróblokk esetén, a jósolt elmozdulásvektor nullázódik. Mivel a makróblokkok nagy része nulla lesz, ezért minden makróblokk tartalmaz egy „ugrás” kódot, hogy utána hány nulla makróblokk következik. Még a nem nulla makróblokkoknál is az egyes blokkok gyakran nullák, ezért a makróblokk tartalmaz egy hat bites maszkot, amely megmondja, mely blokkok szerepelnek, a többi nulla.

A B képek átvitele nagyon hasonló a P képek átviteléhez, de itt minden blokkhoz a P képnél szereplő „előre” elmozdulásvektoron kívül egy „hátra” típusú elmozdulásvektor is tartozhat. Ha csak „előre” vektor szerepel, a blokkhoz a jóslást az előző I vagy P kép adja, ha csak „hátra” vektor szerepel, akkor a következő I vagy P kép, ha mindkettő, akkor az előző és a következő I illetve P képből adódó jóslás számtani közepének alsó egész része.

Az összes fent említett paraméter (elmozdulásvektor, hat bites maszk, ugráskód, stb.) egyenként vagy csoportosan külön az adott célra optimalizált prefix kóddal van kódolva. A dekódolás elég gyors, a kódolás viszont meglehetősen időigényes: az ideális elmozdulásvektorok keresése nem egyszerű. Léteznek más eljárások gyorsabb kódolással, de kisebb teljesítménnyel például videotelefonhoz.

9.3. Hibakorlátozó kódolás

Lángné-Gonda: O.K.; Gonda kiegészítés; Gonda bevmat3; Gonda kódoláselmélet; Jákó; Salomon; Iványi Informatikai algoritmusok1; Cormen–Leiserson–Rivest: O.K.; Rónyai–Ivanyos–Szabó: O.K.; Demetrovics–Denev–Pavlov; Knuth TAOCP; Birkhoff–Bartee.

A *hibakorlátozó kód*okat két csoportba sorolhatjuk: hibajelző kódok és hibajavító kódok. Mindkét típussal foglalkozunk.

9.3.1. Paritásbites kód. A hibajelző kódok feladata, hogy a vétel helyén észrevegyük, ha az átvitel során az adat megváltozott. Minden hibát természetesen nem lehet felderíteni, hiszen abban az esetben, ha egy elküldött kódszó úgy változik, hogy a vétel helyére egy másik kódszó érkezik, akkor a vevőnek semmi joga nincs afelől kételkedni, hogy az elküldött kódszó más volt, mint ami megérkezett. Minden olyan esetben azonban, amikor az érkező jelsorozat különbözik valamennyi lehetséges kódszótól, a vevő biztos lehet benne, hogy az átvitel során sérült a jelsorozat. A legegyszerűbb hibajelző kód a *paritásbites kód*. Legyen például az üzenethalmaz az n-bites bináris jelsorozatok halmaza, és egészítsük ki ezeket a jelsorozatokat egy $n+1$-edik bittel, az úgynevezett *paritásbit*tel: amennyiben egy üzenetben az 1-esek száma páratlan, akkor írjunk a bitsorozat végére egy 0-t, míg az ellenkező esetben egy 1-et (vagy fordítva, de egy adott kódban mindig ugyanazon szabály szerint). Az így kiegészített, $n+1$-bites szavak mindegyikében páratlan sok 1-es van. Ha most egy ilyen kódszót elküldünk, és a vevőhöz olyan szó érkezik, amelyben az egyesek száma páros, akkor biztos, hogy hiba történt az átvitel során. Ha viszont az egyesek száma páratlan, akkor feltesszük (de nem állíthatjuk), hogy nem történt hiba. Könnyen beláthatjuk, hogy az előbbi eset akkor következik be, ha az átvitel során páratlan sok helyen sérül a kódszó, míg az utóbbi akkor, ha páros sok helyen történik változás. Ez azt jelenti, hogy minden olyan esetben észrevesszük a hibát, ha egy hiba történik, de van olyan eset, amikor két hiba történik, és ezt nem vesszük észre (sőt, a konkrét esetben két hiba esetén mindig ez a helyzet). Ez indokolja az alábbi definíciót.

9.3.2. Hibajelző kód. Egy kód *t-hiba jelző*, ha *minden* olyan esetben jelez, amikor egy elküldött kódszó legfeljebb t helyen változik meg. A kód *pontosan t-hiba jelző*, ha t-hiba jelző, de nem $t+1$-hiba jelző, azaz van olyan $t+1$ hiba, amelyet a kód nem jelez.

A hibakorlátozó kódokkal kapcsolatban mindig feltesszük, hogy az egyes kódszavak hossza azonos, és az átvitel során nincs *szinkronhiba*, vagyis a vétel helyére ugyanannyi szimbólum érkezik meg, mint amennyit elküldtünk. Ekkor könnyű olyan feltételt adni, amely jellemzi a t-hiba jelző, illetve a pontosan t-hiba jelző kódokat.

9.3.3. Kódok távolsága és súlya. A kódábécé két egyforma hosszú szavának, u-nak és v-nek a *Hamming-távolsága*, $d(u,v)$, az azonos pozícióban lévő különböző jegyek száma, és a *kód távolsága*, $d(C)$, a különböző kódszópárok távolságainak minimuma. A kód távolsága csak akkor van értelmezve, ha legalább két kódszó van. Ha a kódábécé additív Abel-csoport, akkor a kódábécé egy u szavának a $w(u)$ *Hamming-súlya* a nullától különböző jegyeinek száma, míg a *kód $w(C)$ súlya* a nem nulla kódszavak súlyainak minimuma (ha van nem nulla kódszó).

A Hamming-távolság rendelkezik a távolság szokásos tulajdonságaival, vagyis bármely u, v, z-re

(1) $d(u,v) \geq 0$;

(2) $d(u,v) = 0$ akkor és csak akkor, ha $u = v$;

(3) $d(u,v) = d(v,u)$ (*szimmetria*);

(4) $d(u,z) \leq d(u,v) + d(v,z)$ (*háromszög-egyenlőtlenség*).

Azt is könnyű belátni, hogy $d(u,v) = w(u-v)$, és $w(u) = d(u,0)$. Ha még az is igaz, hogy a kódszavak a koordinátánkénti művelettel maguk is Abel-csoportot alkotnak, akkor $d(C) = w(C)$, ahol C a kódszavak halmaza.

A továbbiakban egy kód távolságát d és a súlyát w jelöli. (A megadott jelölések kissé pontatlanok, hiszen ugyanaz a betű jelöli a függvényt, mint két kódszó valamint a kód távolságát, és hasonló igaz a súlyra is, de a környezet alapján mindig világos lesz, hogy éppen minek a jelöléséről van szó). Mivel két kódszó legalább d helyen különbözik, így, ha egy elküldött kódszó az átvitel során ennél kevesebb helyen sérül, akkor az így kapott szó biztosan nem kódszó. Ugyanakkor van a kódban két olyan kódszó, amelyek pontosan d helyen különböznek, és ha az egyiket küldik, és ez úgy változik, hogy éppen a másik érkezik meg, akkor d hiba történt, de ezt a vétel helyén nem vesszük észre, nem tudunk jelezni. Ebből következik, hogy a bevezetett távolságfogalommal egy kód akkor és csak akkor t-hiba jelző, ha $t < d$, és akkor és csak akkor pontosan t-hiba jelző, ha $t = d - 1$.

A paritásbites kód esetén a kód távolsága 2. Az eredeti üzenethalmaz elemei közötti távolság ugyanis legalább 1, és vannak olyan n-bites sorozatok, amelyek éppen egy helyen különböznek. Ekkor az egyik üzenetben az 1-esek száma páros, a másikban páratlan, vagyis a kiegészítő bit különbözni fog a két kódszóban, és így a két kódszó éppen két helyen különbözik egymástól, a két kódszó távolsága 2. Ugyanakkor azok a kiegészített bitsorozatok, amelyek eredetileg legalább két helyen tértek el egymástól, a kiegészítés után is legalább két helyen fognak különbözni, vagyis ezek távolsága ismét minimum 2, és így a kód távolsága valóban pontosan 2. Ez azt jelenti, hogy ez a kód pontosan 1-hiba jelző, ahogy azt fentebb már megállapítottuk.

A paritásbites kóddal a hibát csak érzékelni, jelezni tudjuk, de a hibás jelsorozatot nem tudjuk kijavítani. A hiba javításához tudni kellene, hogy a beérkezett jelsorozat melyik pozíción sérült, és a sérült helyen mi volt eredetileg (bináris kód esetén, és csak ekkor, a hiba javításához elegendő a hiba helyét ismerni). Abból, hogy az egyesek száma páros, tehát tudjuk, hogy történt hiba, még nem tudjuk megállapítani, hogy melyik bit hibásodott meg, ugyanis bármelyik (egyetlen, vagy páratlan számú) bit hibásodik meg, az ugyanúgy azt eredményezi, hogy az egyesek száma páros lesz.

9.3.4. Minimális távolságú dekódolás. Csupán a vett adatból nem mindig tudjuk megállapítani, történt-e hiba, hiszen bármely elküldött kódszó megváltozhat úgy az átvitel során, hogy bármely másik szó, például akármelyik kódszó érkezzen meg a vétel helyére. Mindazonáltal az megállapítható, hogy egy adott jelsorozat milyen valószínűséggel származik egy adott átviteli csatorna és adott üzenethalmaz esetén valamely kódszóból. A hibajavításhoz meg kell adni egy úgynevezett *döntési függvényt*, amely bármely lehetséges jelsorozathoz hozzárendel egy és csak egy kódszót. (Az is lehetséges, hogy nem minden szó esetén akarunk dönteni.) Ezt a döntési függvényt kell úgy meghatározni, hogy a *döntési hiba*, vagyis az a hiba, hogy egy beérkezett jelsorozathoz nem a ténylegesen elküldött kódot rendeljük, a lehető legkisebb legyen. Az előbbi feltételnek megfelelő legjobb függvény nem csupán az átviteli csatornától függ, ezért helyette általában egy másik döntési függvényt alkalmazunk. Elég természetesnek tűnik (bár ez nem mindig igaz), hogy egy vett szóban előforduló kevesebb hiba valószínűbb, mint a több hiba, vagyis nagyobb valószínűséggel küldtek egy olyan kódszót, amely a vett szótól kevesebb helyen tér el, mint amely több helyen különbözik. Másként szólva, egy vett szó esetén arra a kódszóra döntünk, amely tőle a lehető legkevesebb helyen tér el, azaz a távolsága a vett szótól minimális. Az ilyen döntési függvény által meghatározott dekódolást *minimális távolságú dekódolás*nak mondjuk. Ilyenkor előfordulhat, hogy egy adott szóhoz egynél több minimális távolságra lévő kódszó van. Ekkor vagy kiválasztunk ezek közül egyet, és az lesz a döntés eredménye (de egy adott döntési függvény esetén az ilyen vett szó esetén mindig ugyanarra a kiválasztott, minimális távolságra lévő kódszóra döntünk!), vagy az ilyen szó esetén nem döntünk, csupán jelezzük a hibát.

Legyen például $\{\texttt{0010}, \texttt{0100}, \texttt{1111}\}$ egy bináris kód. Ekkor az egyes négybites szavaktól minimális távolságra lévő kódszavakat a 9.10. táblázat mutatja. Látható, hogy négy esetben nem egyértelmű a helyzet. Egy lehetőség, hogy ezekben az esetekben is dekódolunk, és például az elől álló kódszavakra döntünk. A másik lehetőség, hogy amennyiben a vett szó az előbbi négy szó valamelyike, akkor nem döntünk, csak jelezzük a hibát. Ha például a három kódszó három színt, mondjuk a fehéret, feketét és pirosat kódolja a példában megadott sorrendben, és mondjuk a kódolt üzenet a fekete volt, míg a vétel helyén a `0000` szót kell dekódolni, akkor a táblázat alkalmazásával a fehérre dekódolunk, ami adott esetben igen rossz döntés lehet. Ebben az esetben célszerűbb nem dönteni, hanem jelezni a hibát, és ha lehet, akkor megismételtetni ezt az üzenetet, vagy valamilyen más módon, például a környező pontok színe alapján dönteni. Azt azért megismételjük, hogy a döntési függvényt a rendszer tervezésekor rögzítettük, és a továbbiakban mindig ugyanúgy kell eljárnunk az adott rendszer keretein belül. A döntési függvény táblázattal való megadása egyszerű, de rendkívül helyigényes.

0000 $\mapsto$ 0010 0100
0001 $\mapsto$ 0010 0100
0010 $\mapsto$ 0010
0011 $\mapsto$ 0010
0100 $\mapsto$ 0100
0101 $\mapsto$ 0100
0110 $\mapsto$ 0100 0010
0111 $\mapsto$ 1111
1000 $\mapsto$ 0100 0010
1001 $\mapsto$ 1111
1010 $\mapsto$ 0010
1011 $\mapsto$ 111
1100 $\mapsto$ 0100
1101 $\mapsto$ 1111
1110 $\mapsto$ 1111
1111 $\mapsto$ 1111

9.10. táblázat

9.3.5. Hibajavító kód. Egy kód *t-hiba javító*, ha minden olyan esetben helyesen javít, amikor egy elküldött kódszó legfeljebb t helyen változik meg. A kód *pontosan t-hiba javító*, ha t-hiba javító, de nem $t+1$-hibajavító, azaz van olyan $t+1$ hiba, amelyet a kód helytelenül javít, vagy nem javít.

Egy d távolságú kód esetén minimális távolságú dekódolással $t < d/2$ hiba esetén biztosan jól döntünk, hiszen a háromszög-egyenlőtlenség következtében az eredetileg elküldött kódszótól különböző bármely más kódszó biztosan $d/2$-nél több helyen tér el a vett szótól. Viszont $t \geq d/2$ esetén nincs olyan döntési függvény, amely t-hiba javító, mert a kódban van két olyan kódszó, mondjuk u és v, amelyek pontosan d helyen különböznek. Írjuk u-ban ebből a d számú pozícióból t helyre a v adott pozícióján található jegyet, és jelöljük az így kapott szót z-vel. A z az u-tól t helyen különbözik, míg v-től $d-t \leq d/2 \leq t$ helyen. Ha a dekódolás t-hiba javító lenne, akkor z-t egyrészt u-ra, másrészt v-re kellene javítani. Tehát egy d-távolságú kód minimális távolságú dekódolással minden $t < d/2$-re t-hiba javító, és pontosan $\lfloor (d-1)/2 \rfloor$-hiba javító.

A továbbiakban feltesszük, hogy minimális távolságú dekódolás esetén a döntési függvény teljes, vagyis minden vett szóhoz hozzárendel egy kódszót.

9.3.6. Ismétléses kód. Legyen egy binárisan kódolt üzenethalmazunk, és küldjük el az üzenetet úgy, hogy minden egyes bitet megháromszorozunk, azaz ugyanazt a bitet háromszor egymás után küldjük. A vétel helyén a három összetartozó bit közül legalább kettő azonos lesz, így a minimális távolságú dekódolás esetén a vett három bithez a többségi döntés alapján rendelünk egy bitet. A döntési hiba csökkenthető, ha az eredeti egy bit helyett nem három, hanem $5, 7, 9, \ldots, 2n+1$, stb., az eredeti bittel azonos jegyet küldünk. Meghatározott feltételek esetén igazolható, hogy n növekedésével a döntési hiba valószínűsége a 0-hoz tart. Ebből azonban hiba lenne arra következtetni, hogy megtaláltuk a szinte biztosan hibátlan adatátvitel módját. Egyrészt, mint említettük, az előbbi eredmény csak bizonyos feltételek esetén teljesül. Másrészt nézzük meg, hogy mi ennek az ára. Az egyes bitek átviteléhez adott, 0-nál hosszabb időre van szükség, így n növekedésével az üzenet egy-egy bitjének átviteléhez szükséges idő is nő, és tart a végtelenhez, vagyis az 1 valószínűséggel hibátlan átvitel esetén egyetlen bitnyi üzenetet sem tudunk továbbítani. Az átviteli idő növekedése egyben az átvitel költségét is növeli, az is tart a végtelenhez. Szerencsére a helyzet nem ennyire rossz. *Shannon* egy tétele szerint bizonyos feltételek teljesülése esetén lehet az üzeneteket úgy kódolni, hogy az átvitel sebessége egy, csak a csatornától függő értéket, a *csatornakapacitás*t tetszőlegesen megközelítsen, miközben a dekódolás hibája tetszőlegesen kis érték alá szorítható. A tétel elméleti jelentőségű, ugyanis ilyen kódot nem sikerült konstruálni, és, még ha sikerülne is, akkor is használhatatlan lenne a gyakorlatban, olyan nagy lenne a kódszavak hossza, és olyan nagy lenne a kód mérete. A lényege a tételnek mégis óriási, ugyanis azt igazolja, hogy lehetséges olyan kódot konstruálni, amelynél mind a sebesség, mind a döntési hiba kielégít egy reális elvárást.

9.3.7. Kétdimenziós paritásellenőrzés. A korábban tárgyalt paritáselemes kód ismeretében könnyen tudunk egy minimális távolságú dekódolással 1-hiba javító kódot konstruálni. Legyenek az üzenetek n-bites szavak. Egészítsünk ki minden kódszót egy paritásbittel mondjuk páratlan paritással, majd m ilyen kódszóból alkossunk egy blokkot. Írjuk egymás alá a blokk $n+1$-bites kódszavait, és most az egy-egy oszlopban álló m-bites sorozatokat egészítsük ki egy-egy paritásbittel például páros paritásúvá. Az így kapott $n+1$-bites szóval kiegészítve a blokkot kapjuk az eredeti m üzenet kódját, amelyet a 9.11. táblázat mutat. (A gyakorlatban az egyes szavak nyolc bitből álló bájtok, vagyis $n = 8$. Ilyenkor szokás az egyes bájtok ellenőrzését végző paritásbiteket *VRC*-nek nevezni, ami a *Vertical Redundancy Check*, keresztirányú ellenőrzés kezdőbetűiből álló rövidítés, míg az ellenőrző bájt az *LRC*, azaz *Longitudinal Redundancy Check*, a hosszirányú ellenőrzés.)

$$
\begin{array}{ccccc|c}
b_{0,0} & \dots & b_{0,j} & \dots & b_{0,n-1} & b_{0,n} \\
\vdots & & \vdots & & \vdots & \vdots \\
b_{i,0} & \dots & b_{i,j} & \dots & b_{i,n-1} & b_{i,n} \\
\vdots & & \vdots & & \vdots & \vdots \\
b_{m-1,0} & \dots & b_{m-1,j} & \dots & b_{m-1,n-1} & b_{m-1,n} \\
\hline
b_{m,0} & \dots & b_{m,j} & \dots & b_{m,n-1} & b_{m,n}
\end{array}
$$

9.11. táblázat

Ez a rendszer egyetlen hiba esetén képes azt javítani. Ha ugyanis $b_{i,j}$ és csak ez a bit hibás, akkor pontosan egy hiba van az i-edik szóban, tehát ennek ellenőrzése során hibajelzést kapunk. Az összes többi szó ellenőrzése azt adja, hogy azokban nincs hiba, és hasonlóan, a j-edik és csak a j-edik oszlop ellenőrzésénél hibajelzésre kerül sor, vagyis a két eredményből azt kapjuk, hogy az i-edik szó j-edik bitje és csak ez a bit hibás, amit ennek a bitnek az invertálásával kijavíthatunk. Minden olyan esetben azonban, amikor az i-edik és csak az i-edik szóban valamint a j-edik oszlopban és csak a j-edik oszlopban kapunk hibajelzést, azt gondoljuk, hogy ez a bit hibásodott meg, és ezt „javítjuk”, vagyis változtatjuk az ellenkezőjére. Pedig könnyen beláthatjuk, hogy ellenőrzéskor ugyanerre az eredményre jutunk akkor is, ha minden oszlopban és minden szóban, kivéve a j-edik oszlopot és i-edik szót, páros számú hiba lép fel (ebbe beleértve a hibátlan esetet is 0 hibával), míg a kitüntetett i-edik szóban és j-edik oszlopban a hibák száma páratlan. Ilyen például, ha az i-edik szóban a $j+1$-edik bit, valamint az $i+1$-edik szó j-edik és $j+1$-edik bitje hibásodik meg, és más hiba nem történik. A vétel helyén semmilyen módszerrel nem tudjuk eldönteni, hogy valóban egy hiba történt-e, és így helyesen javítunk, vagy a megfigyelt hibajelzés több hiba együttes hatása, aminek következtében vagy egy addig helyes bitet javítunk helytelenre, vagy egy tényleg hibás bitet javítunk, de még további, felderítetlen hiba is van a vett üzenetben. Hasonlóan előfordulhat, hogy volt hiba, de mi nem vesszük észre, nevezetesen, ha minden szóban és minden oszlopban a hibák száma páros. Ennek tipikus példája, amikor négy hiba egy téglalap négy sarkában lép fel, ahol a téglalap két éle egy-egy szóban van. Végül előfordulhat, hogy egynél több szóban illetve egynél több oszlopban kapunk hibajelzést, amikor biztosan tudjuk, hogy volt hiba, de a hibák száma egynél nagyobb, így javításra ebben a rendszerben nincs lehetőségünk. Ennek legegyszerűbb példája, amikor pontosan két hiba lép fel. A most ismertetett rendszert nevezhetjük *kétdimenziós paritásellenőrzés*nek. Ilyet használnak nagy mágnesszalagos tárolóknál (egy bizonyos rögzítési mód esetén, mert többféle is létezik), ahol egymás mellé írják ki egy kiegészített bájt 9 bitjét, és az adatokat mindig blokkosan tárolják. Ebben az esetben keresztirányban páratlanra, míg hosszirányban párosra egészítenek ki.

9.3.8. Lineáris kód. Ahhoz, hogy a kódolás és a dekódolás minél egyszerűbb legyen, a kódoláshoz olyan rendszereket célszerű alkalmazni, amelyek rendelkeznek valamilyen belső struktúrával. Jó kódok konstruálhatóak algebrai rendszerek segítségével. A gyakorlatban alkalmazott kódok jelentős része lineáris kód. Ha K test, akkor a K elemeiből alkotott rendezett n-esek a komponensenkénti összeadással, valamint az n-es minden elemének ugyanazzal az elemmel való szorzásával egy K feletti n-dimenziós lineáris teret alkotnak. Ennek a térnek bármely altere egy *lineáris kód*. Ha az altér k-dimenziós, a kód távolsága d, és a test elemeinek száma q, akkor az ilyen kódot $[n,k,d]_q$ kódnak nevezzük. Ha nem lényeges a megadása, akkor elhagyható a jelölésből d illetve q. Lineáris kódnál a szavak tekinthetők polinomoknak is, a betűket nullától indexelve.

A paritásellenőrző kód általában nem lineáris, de ha párosra egészítünk ki, akkor már lineáris. További egyszerű lineáris kódok az úgynevezett *CRC* vagyis *Cyclic Redundancy Check*, ciklikus ellenőrzés kódok. A test a kételemű test. A k bites kódolandó szó elejére írunk $m=n-k$ darab nullát, majd a megfelelő polinomot osztjuk egy adott m-ed fokú polinommal, a *kódpolinom*mal. Végül a kapott maradékkal kicseréljük a nullákat. Mivel a kételemű test felett minden polinom megegyezik az ellentettjével, a kódszavakat az jellemzi, hogy oszthatók a kódpolinommal. Néhány gyakran használt CRC-kódpolinom:

CRC-1 (paritásbit)	$x+1$
CRC-5-USB	x^5+x^2+1
CRC-8	x^8+x^2+x+1
CRC-16-IBM	$x^{16}+x^{15}+x^2+1$
CRC-16-CCITT	$x^{16}+x^{12}+x^5+1$
CRC-32 (Ethernet, FDDI, gzip, PNG)	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$
CRC-64-ISO	$x^{64}+x^4+x^3++x+1$

9.3.9. Generátormátrix, ellenőrző mátrix. A K véges test feletti $[n,k]$ lineáris kódnál célszerű a kódolási eljárást egy K^k-t $C\subset K^n$-re képező lineáris leképezésnek választani, ahol C a kódszavak altere. Ezt a mátrixával jellemezhetjük, ez a kódolás *generátormátrix*a. A dekódolásra használható egy $H:K^n\to K^k$ szürjektív lineáris leképezés, amelynek a magja C. Egy ilyen leképezés mátrixát a kód *ellenőrző mátrix*ának nevezzük. Ha $v\in K^n$, akkor az $s=Hv$ *szindróma* pontosan akkor nulla, ha v kódszó.

9.3.10. Szindróma-dekódolás. Az előző pont jelöléseivel, ha $s\in K^{n-k}$, legyen $e(s)$ a $\{v: Hv=s\}$ halmaz egy olyan vektora, amelynek súlya minimális. Ezt *mellékosztályvezető*nek fogjuk nevezni. Ha $c\in K^n$ egy kódszó, $v\in K^n$ a vett szó, $e=v-c$ a hiba, és $w(e)<d/2$, akkor $He(s)=s=Hv=He$, így $w\bigl(e(s)\bigr)\le w(e)$, ahonnan $w\bigl(e-e(s)\bigr)<d$. De $H\bigl(e-e(s)\bigr)=0$, így a különbség kódszó, tehát $e=e(s)$, a hiba javítható. A szindróma-dekódolás tárigenye sokkal kisebb, mint a táblázattal való dekódolásé, de még mindig nagyon nagy lehet.

9.3.11. Singleton-korlát. Ha adott egy q elemű ábécé betűiből álló n hosszú kódszavakat tartalmazó C kód, amelynek távolsága d, akkor minden kódszóból elhagyva $d-1$ betűt (ugyanarról a $d-1$ helyről) még mindig különböznek a kódszavak, de csak $n-d+1$ hosszúak. Innen a kódszavak számára azt kapjuk, hogy $\natural(C)\le q^{n-d+1}$, ez a *Singleton-korlát*. Mindkét oldal logaritmusát véve $d+\log_q\natural(C)\le n+1$. Egy lineáris $[n,k,d]_q$-kódnál azt kapjuk, hogy $d+k\le n+1$. Ha egyenlőség áll, a lineáris kódot *maximális távolságú szeparábilis kód*nak, *MDS-kód*nak nevezzük. A szeparábilis (elválasztható) kód elnevezést az indokolja, hogy bármely rögzített $d-1=n-k$ helyen álló betűket elhagyva a kódszavakból, q^k különböző szó marad, ezért a lineáris kódolást választhatjuk úgy, hogy bármely adott k helyen a kódolandó szó betűi álljanak, így az ellenőrző betűk elválaszthatók a kódolandó betűktől.

* **9.3.12. Hamming-kód.** Az úgynevezett *Hamming-kód* egy hiba javítására alkalmas lineáris kód. Csak a bináris, azaz a kételemű test feletti speciális esettel foglalkozunk. Legyen $r\ge 2$ egész szám, $n=2^r-1$, és $k=n-r$. Készítsünk egy $r\times n$ méretű mátrixot, amelyben az $0<j\le n$ indexre a j-edik oszlopban a j szám kettes számrendszerbeli felírásának jegyei találhatóak, vagyis a mátrix i-edik sorának j-edik oszlopában $h_{i,j}$ áll ($0\le i<r$, $0<j\le n$), ahol $j=\sum_{i=0}^{r-1}h_{i,j}2^i$. (Mivel $1\le j\le n<2^r$, ezért j biztosan felírható r jeggyel a bináris számrendszerben). Legyen például $r=3$, akkor $n=7$ és

$k = 4$, továbbá

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

Mivel $r > t \in \mathbb{N}$ esetén 2^t kettes számrendszerben való felírása olyan, amelyben a 0-tól kezdődő indexelés mellett a t-edik és csak a t-edik jegy 1, az összes többi 0, ezért a mátrix r darab oszlopa, amelyek a 2^t indexhez tartoznak, egy egységmátrixot adnak (az előbbi példában az 1., a 2. és 4.), így a mátrix rangja r, tehát a megfelelő H lineáris leképezés szürjektív. Legyen C azon n-dimenziós bináris vektorok halmaza, amelyekre $Hc = 0$. Az ilyen vektorokban k komponens szabadon választható. (Persze nem bármelyik k, csak azok, amelyekkel a kimaradt komponensekhez tartozó indexek a mátrix reguláris részmátrixát határozzák meg, például a fenti példában nem jó választás az utolsó négy komponens, mert a mátrix első három oszlopa lineárisan összefüggő.) Legyenek ezért a kódolandó üzenetek k bitesek, és egy-egy ilyen üzenetet egészítsünk ki r bittel úgy, hogy a kapott n-bites vektor eleme legyen a C halmaznak. Tegyük fel, hogy egy ilyen n-bites üzenet az átvitel során megsérül. Ez azt jelenti, hogy bizonyos bitek az ellenkezőjükre változnak, amit úgy is megkaphatunk, ha ezekhez az eredeti bitekhez 1-et adunk modulo 2, vagyis tegyük fel, hogy c az eredeti n-bites vektor, $e = e_1 \dots e_n$ a hibavektor, és a vétel helyére $v = c + e$ érkezik. Ha a H mátrix j-edik oszlopát h_j-vel jelöljük, akkor

$$Hv = H(c + e) = Hc + He = He = \sum_{e_j = 1} h_j,$$

hiszen e_i értéke csak 0 és 1 lehet. Ha pontosan egy hiba lépett fel, akkor egy és csak egy indexre, mondjuk s-re lesz e_j nullától különböző, és ekkor $Hv = h_s$. De H konstrukciója következtében h_s éppen s kettes számrendszerbeli felírása, vagyis Hv pontosan a hiba helyét adja.

Legyen például az előbbi 3×7-es mátrixhoz $c = 0100101$, akkor ellenőrizhető, hogy $Hc = 0$, vagyis c kódszó, és tegyük fel, hogy $e = 0000100$, vagyis az 5. bit és csak ez a bit az üzenet átvitele során megsérül. Ekkor a vétel helyén a $v = 0100001$ bitsorozatot kapjuk, és $s = Hv = 101$, ami mint bináris szám éppen 5, a hiba helye. Megváltoztatva a vett üzenetben az 5. bitet, a 0100101 bitsorozatot kapjuk, egyezésben az elküldött bitsorozattal. Amennyiben $e = 0000101$, akkor $s = 010$, és „javítás” után a 0000000 bitsorozatot kapjuk, ami nem egyezik az eredeti sorozattal, vagyis rosszul javítottunk. A Hamming-kód pontosan 1-hiba javító kód. Érdemes megnézni, hogy mi a helyzet, ha a hibavektor $e = 1110000$.

9.3.13. Reed–Solomon-kódok. Legyen K egy véges test, a test egy nem nulla α elemének multiplikatív rendje n és $0 < k < n$. Ekkor az α^i, $0 \le i < n$ elemek páronként különböznek, és mindegyik gyöke az $x^n - 1 \in K[x]$ polinomnak, ezért megadják ezen polinom összes gyökét. Így $x^n - 1 = \prod_{i=0}^{n-1} (x - \alpha^i)$.

Legyen $m = n - k$ és $g = \prod_{i=1}^{m} (x - \alpha^i)$. Ez a polinom egy K fölötti, m-edfokú főpolinom, és osztója az $x^n - 1$ polinomnak. A $C = \{ag : a \in K[x], \deg(a) < k\}$ halmaz a g (vagy az α) által generált *Reed–Solomon-kód*, és g a kód *generátorpolinom*ja. A C

elemei n-nél alacsonyabb fokú polinomok. A $\varphi : a \mapsto ag$ leképezés K^k injektív lineáris leképezése C-re, amiből az is következik, hogy C a K^n egy k-dimenziós altere.

Most tekintsük a C egy c elemét. Ekkor g osztója c-nek, tehát g minden gyöke gyöke c-nek, vagyis $c(\alpha^i) = 0$, ha $1 \le i \le m$. Fordítva, ha $u \in K^n$, és minden $1 \le i \le m$-re $u(\alpha^i) = 0$, akkor valamennyi i-re $x - \alpha^i$ osztója u-nak, de akkor ezek legkisebb közös többszöröse, azaz a szorzatuk, tehát g is osztója u-nak, vagyis ez esetben u a kódhoz tartozik. Ez azt jelenti, hogy $u \in K^n$ akkor és csak akkor eleme a kódnak, ha g valamennyi gyöke egyben u-nak is gyöke, vagyis ha minden $1 \le i \le m$-re $\sum_{j=0}^{n-1} (\alpha^i)^j u_j = 0$. Így a $h_{i,j} = \alpha^{ij}$, $1 \le i \le m$, $0 \le j < n$ mátrix egy ellenőrző mátrix, a hozzá tartozó H lineáris leképezésre $Hu = 0$ akkor és csak akkor, ha $u \in C$. Legyen $0 \le j_1 < \ldots j_m < n$, és nézzük a mátrix j_l indexű oszlopait. Ezek a mátrix egy m-edrendű kvadratikus részmátrixát adják, amelynek l-edik oszlopában az i-edik elem $h_{i,j_l} = (\alpha^i)^{j_l} = (\alpha^{j_l})^i$. Most nézzük ezen részmátrix determinánsát. A determináns l-edik oszlopában minden elemből kiemelhető α^{j_l}. Mivel a kiemelt elem nem nulla, ezért az eredeti determináns akkor és csak akkor 0, ha a kiemelés után kapott determináns értéke 0. A kapott determináns l-edik oszlopában α^{j_l} egymás után következő hatványai állnak, a 0 kitevős hatvánnyal kezdve, vagyis ez egy úgynevezett Vandermonde-típusú determináns. A méret szerinti indukcióval nem nehéz látni, hogy a determináns értéke $\prod_{0 \le s < t < m} \left(\alpha^{j_s} - \alpha^{j_t}\right) \ne 0$ (vonjuk ki minden sorból az felette álló α^{j_1}-szeresét). Ez viszont azt jelenti, hogy a mátrix bármely m oszlopa lineárisan független. Ebből egyrészt az következik, hogy rangja $m = n - k$, tehát a kód egy ellenőrző mátrixát kaptuk, másrészt, hogy egy legfeljebb $m = n - k$ súlyú e vektorra $He \ne 0$, tehát a kód távolsága $d = n - k + 1$, azaz a kód maximális távolságú. Ez mutatja, hogy megfelelő kóddal egynél több hiba is javítható.

9.3.14. A Reed–Solomon-kód dekódolása. A Reed–Solomon-kód lineáris, tehát a hiba javítható például a szindróma-dekódolással, de mutatunk egy ennél lényegesen praktikusabb hibajavítást.

Legyen adott egy $[n, k]_q$ Reed–Solomon-kód, $m = n - k$, $g = \prod_{i=1}^{m} (x - \alpha^i)$ a kód generátorpolinomja, e a hibavektor, és $L = \prod_{e_j \ne 0} (1 - \alpha^j z)$ az úgynevezett *hibahelypolinom*. Ennek ismeretében a hibák helye meghatározható: megkeressük, hogy mely α^{-j}-k gyökei L-nek, és a j-k megadják a hibák helyét. Legyen $E = \sum_{e_j \ne 0} \alpha^j e_j L_j$ az úgynevezett *hibaérték-polinom*, ahol $L_j = L/(1 - \alpha^j z)$, ha $e_j \ne 0$. Ha még ezt is ismerjük, akkor a hiba javítható, mert rögzített j esetén $L_i(\alpha^{-j})$ akkor és csak akkor nem nulla, ha $i = j$, ezért

$$e_j = \frac{E(\alpha^{-j})}{\alpha^j L_j(\alpha^{-j})}.$$

A következő tétel lehetővé teszi a két polinom meghatározását.

9.3.15. Tétel. *Az előző pont jelöléseivel legyen s a szindrómához tartozó polinom. Tegyük fel, hogy a hibahelyek száma, azaz L fokszáma legfeljebb $m/2$. Végezzünk bővített euklidészi algoritmust az $a = z^r$ és $b = s$ polinomokkal. Az ottani jelölésekkel legyen l a legkisebb index, amelyre $\deg(r_k) < m/2$, és legyen $r_l = ax_l + by_l$. Ekkor $y_l(0) \ne 0$ és $L = y_l/y_l(0)$, $E = r_l/y_l(0)$.*

* **Bizonyítás.** Először megmutatjuk, hogy x^m osztja az $E - Ls$ polinomot. Valóban,

$$\begin{aligned}
E - Ls &= \sum_{e_j \neq 0} \alpha^j e_j L_j - L \sum_{i=0}^{m-1} s_i z^i = \sum_{e_j \neq 0} \alpha^j e_j L_j - \sum_{i=0}^{m-1} L \Big(\sum_{j=0}^{n-1} \left(\alpha^{i+1}\right)^j e_j \Big) z^i \\
&= \sum_{e_j \neq 0} \alpha^j e_j h_j - \sum_{j=0}^{n-1} e_j L \sum_{i=0}^{m-1} \left(\alpha^{i+1}\right)^j z^i = \sum_{e_j \neq 0} \Big(\alpha^j e_j L_j - \alpha^j e_j L \sum_{i=0}^{m-1} \left(\alpha^j\right)^i z^i \Big) \\
&= \sum_{e_j \neq 0} \Big(\alpha^j e_j L_j - \alpha^j e_j L \frac{1 - (\alpha^j z)^m}{1 - \alpha^j z} \Big) \\
&= \sum_{e_j \neq 0} \Big(\alpha^j e_j L_j - \alpha^j e_j L_j \big(1 - \alpha^j z\big) \frac{1 - (\alpha^j z)^m}{1 - \alpha^j z} \Big) \\
&= z^m \sum_{e_j \neq 0} \alpha^{j(m+1)} e_j L_j .
\end{aligned}$$

Ez azt jelenti, hogy alkalmas f polinommal $E = fz^m + Ls$. Így z^m és s legnagyobb közös osztója osztója E-nek, és fokszáma legfeljebb annyi, mint E fokszáma, ami kisebb, mint $m/2$. Így van olyan maradék az euklidészi algoritmus alkalmazása során, amelynek fokszáma kisebb, mint $m/2$. Az $E = fz^m + Ls$ egyenlet y_l-szereséből kivonva az

$$r_l = ax_l + by_l = z^m x_l + sy_l$$

egyenlet L-szeresét, azt kapjuk, hogy

$$Ey_l - Lr_l = (fy_l - Lx_l) z^m. \tag{1}$$

Azt akarjuk megmutatni, hogy a zárójelben álló polinom nulla.

A bővített euklidészi algoritmus az $r_0 = a$, $r_1 = b$, $x_0 = 1$, $x_1 = 0$, $y_0 = 0$, $y_1 = 1$ kezdőértékkel indul. Megmutatjuk, hogy

$$\deg(y_j) = \deg(a) - \deg(r_{j-1}) \tag{2}$$

és

$$x_{j-1} y_j - x_j y_{j-1} = (-1)^{j-1},$$

ha $j > 0$. Mindkét állítás teljesül $j = 1$-re. Indukcióval

$$\begin{aligned}
\deg(y_{j+1}) &= \deg(y_{j-1} - q_j y_j) = \deg(q_j y_j) = \deg(q_j) + \deg(y_j) \\
&= \deg(r_{j-1}) - \deg(r_j) + \deg(a) - \deg(r_{j-1}) \\
&= \deg(a) - \deg(r_j)
\end{aligned}$$

és

$$x_j y_{j+1} - x_{j+1} y_j = x_j (y_{j-1} - q_j y_j) - (x_{j-1} - q_j x_j) y_j = (-1)^j .$$

Térjünk vissza (1)-hez. Mivel l választása és (2) miatt

$$\deg(y_l) = \deg(a) - \deg(r_{l-1}) \leq m - m/2 = m/2,$$

$\deg(E) < m/2$, $\deg(r_l) < m/2$, $\deg(L) \leq m/2$, a bal oldal foka kisebb, mint m, így (1)-ben a zárójelben csak nulla állhat. Innen $Ey_l = Lr_l$ és $fy_l = Lx_k$. Mivel definíciójuk szerint E és L relatív prímek, valamint (3) szerint x_l és y_l is, innen az következik, hogy L osztója y_l-nek és y_l osztója L-nek, tehát asszociáltak, azaz $L = cy_l$ valamely nem nulla konstanssal. Innnen $E = cr_l$ ugyanazzal a konstanssal. Mivel L főpolinom, kapjuk az állítást.

* **9.3.16. Példa.** Legyen $K = \mathbb{Z}_{11}$, $n = 10$, $k = 6$ tehát $m = n - k = 4$. A $\mathbb{Z}_{11}$ test elemei helyett a legkisebb nem negatív reprezentánsukat fogjuk írni hexadecimálisan. $\mathbb{Z}_{11}$-ben $2^2 = 4 \neq 1$, és $2^5 = (2^2)^2 2 = 4^2 2 = \mathrm{A} \neq 1$, ezért 2 rendje 10. Mivel $m = n-k = 4$, a kód generátorpolinomja $\prod_{i=1}^{4}(z - 2^i) = z^4 + 3z^3 + 5z^2 + 8z + 1$. A kód elemeit úgy kapjuk, hogy a generátorpolinomot megszorozzuk a $\mathbb{Z}_{11}$ feletti legfeljebb ötödfokú polinomokkal. Legyen egy ilyen polinom $2z^4 + 5z + 4$, ekkor a megfelelő kódszó

$$u = 2z^8 + 6z^7 + \mathrm{A}z^6 + \mathrm{A}z^5 + \mathrm{A}z^4 + 4z^3 + 5z^2 + 4z + 4,$$

vagyis $u = 4454\mathrm{AAA}620$. A kód távolsága $d = 5$, tehát a kód képes 2 hibát helyesen javítani. Legyen például $e = 0030200000$ a hibavektor, ekkor a vett szó $v = 44841\mathrm{AA}620$.

Az ellenőrző mátrix most

$$\begin{pmatrix} 1 & 2 & 4 & 8 & 5 & \mathrm{A} & 9 & 7 & 3 & 6 \\ 1 & 4 & 5 & 9 & 3 & 1 & 4 & 5 & 9 & 3 \\ 1 & 8 & 9 & 6 & 4 & \mathrm{A} & 3 & 2 & 5 & 7 \\ 1 & 5 & 3 & 4 & 9 & 1 & 5 & 3 & 4 & 9 \end{pmatrix},$$

és a szindróma

$$Hv = 0\mathrm{A}25,$$

így az átvitel során hiba keletkezett. Végezzünk euklideszi algoritmust a $z^m = z^4$ valamint a szindrómához tartozó $s = 5z^3 + 2z^2 + 10z$ polinomra addig, amíg a maradékpolinom foka először lesz kisebb, mint $m/2$, és közben számítsuk ki sorban az y_j polinomokat, ahol y_j a kiterjesztett euklideszi algoritmusban meghatározott együtthatója az s polinomnak, vagyis amivel $r_j = z^m x_j + sy_j$. Mint ismert, ha $y_0 = 0$ és $y_1 = 1$, akkor $y_{j+1} = y_{j-1} - q_j y_j$, ahol q_j a j-edik osztás hányadosa.

A konkrét esetben

$$z^4 + 0z^3 + 0z^2 + 0z + 0 : 5z^3 + 2z^2 + \mathrm{A}z + 0 = 9z + 3$$
$$3z^2 + 0z + 3$$

$$y_1 = 0 - (9z + 3) \cdot 1 = 2z + 8$$

$$5z^3 + 2z^2 + \mathrm{A}z + 0 : 3z^2 + 0z + 3 = 8z + 7$$
$$2z + 0$$

$$y_2 = 1 - (8z+7)(2z+8) = 4z^2 + 7z + 9,$$

vagyis $r_2 = 2z$ és $y_2 = 4z^2 + 7z + 9$. Szorozzuk meg y_2-t $y_2(0) = 9$ inverzével, 5-tel: $L = 9z^2 + 2z + 1$. Mivel $L = (1-4z)(1-5z)$, a hiba a vett szó 2 és 4 indexű pozíciójában van, ugyanis $4 = 2^2$, és $5 = 2^4$. Meg kell még határozni a hibák értékét: $E = 5r_2 = \mathrm{A}z$, $L_2 = 1 - 5z$, és $L_4 = 1 - 4z$, így

$$e_2 = \frac{E(4^{-1})}{2^2 L_2(4^{-1})} = \frac{8}{4 \cdot 8} = 3,$$

és

$$e_4 = \frac{E(5^{-1})}{2^4 L_4(5^{-1})} = \frac{2}{5 \cdot 9} = 2,$$

vagyis a kiszámolt hibavektor $e = 0030200000$.

A példa rávilágít a matematika szépségére. Egy olyan régi algoritmusnak, mint az euklideszi algoritmus, egy igen fontos és hasznos alkalmazását láthatjuk, amelyre akkor, amikor magát az algoritmust megalkották, még gondolni sem lehetett.

* **9.3.17. Kódrövidítés.** Néhány kód, például a Reed–Solomon-kód csak meghatározott hosszakban konstruálható. Ilyenkor segít a *kódrövidítés*. Tetszőleges kódra gyűjtsük össze mindazokat a kódszavakat, amelyekben egy adott helyen egy adott betű áll. Csak ezeket fogjuk használni, és a kódolás után az adott helyen álló adott betűt kihagyjuk, a dekódolás előtt pedig újra beírjuk. Nyilvánvaló, hogy a kódrövidítés nem csökkenti a távolságot (ha a rövidített kódnak egyáltalán még van távolsága).

Lineáris kódnál azokat a kódszavakat érdemes felhasználni a rövidített kódban, amelyekben az adott helyen nulla áll, mert ekkor a rövidített kód is lineáris lesz. Ha egy $[n,k,d]_q$ lineáris kódot így rövidítünk, akkor azon kódszavak C' altere, amelyekben az adott helyen nulla áll, az eredeti kódszavak C alterének egy alterét, így a C Abel-csoportnak egy részcsoportját alkotja, továbbá bármely két C-beli kódszó különbsége, amelyekben az adott helyen ugyanaz a betű áll, a C'-ben van, így C' indexe C-ben legfeljebb q, ahonnan C'-nek legalább q^{k-1} eleme van. Így k vagy nem változik, vagy eggyel csökken. Ha az eredeti kód MDS-kód volt, és $k > 1$, akkor a rövidített is az marad, mert a $d = n - k + 1$ egyenlőségben a bal oldalon d csak nőhet, a jobb oldalon n eggyel csökken, k viszont legfeljebb eggyel csökkenhet, így a Singleton-korlát miatt egyik oldal sem változhat.

* **9.3.18. Kódok direkt szorzata.** Úgy, mint a kétdimenziós paritásellenőrzésnél, két kód felhasználásával készíthetünk egy harmadik kódot: előbb keresztirányban kódolunk az első kóddal, majd hosszirányban a másodikkal, vagy fordítva. Az így kapott kód a *kódok direkt szorzata*. Ha az egyes kódok távolsága d_1 illetve d_2, akkor a direkt szorzatuk távolsága legalább $d_1 d_2$; valóban, két különböző kódszó ha valamely sorban különbözik, akkor ott legalább d_1 helyen különbözik, és legalább d_2 ilyen sornak kell lennie, mert egyébként nem lehetne olyan oszlop, amelyben a két kódszó legalább d_2 helyen különbözik.

Például a DVD-n az adatokat (egyebek közt) egy $[208, 192]_{256}$ és egy $[182, 172]_{256}$ rövidített Reed–Solomon-kód direkt szorzata védi.

* **9.3.19. Kaszkád kódok.** Egy további kódötvöző eljárás a *kaszkád kód*. Tegyük fel, hogy egy kód betűi megfeleltethetők egy másik kód adott hosszúságú szavainak. Kódoljunk először az első kóddal, majd a kapott kódszó betűit kódoljuk a második kóddal. Ha az első kód távolsága d_1, a másodiké d_2, akkor a kaszkád kód távolsága legalább $d_1 d_2$, hiszen két különböző kódszó az első kódolás után legalább d_1 betűben, és ezek mindegyikének a kódja a második kódolás után legalább d_2 helyen különbözik.

A legegyszerűbb példa kaszkád kódra, amikor egy $[n, k]_{256}$ Reed–Solomon-kód után bájtonként paritásbitet képezünk.

* **9.3.20. Adatátszövés.** Az átvitel során a hibák gyakran *hibacsomó*kban, idegen szóval *burst*-ökben jelentkeznek. Bár például a Reed–Solomon-kódok alkalmasak rövidebb hibacsomók javítására is, a hosszabb hibacsomók túlterhelik a kódot. Ez ellen úgy védekezhetünk, hogy a kódolás után az egyes blokkok adatait szétszórjuk: ez az *adatátszövés*. A legegyszerűbb esetben valahány adott hosszúságú blokkot sorfolytonosan helyezünk el egy mátrixban, majd az adatokat oszlopfolytonosan olvassuk ki onnan, dekódolás előtt pedig fordítva végrehajtva ezt, visszaállítjuk az eredeti sorrendet.