5.

Információtömörítés

Arra törekszem, hogy tíz mondatban mondjam el mindazt, amire másoknak egy egész könyv kell. FRIEDRICH NIETZSCHE

Manapság óriási mennyiségű információt tárolunk a gépeinkben, és küldünk a szélrózsa minden irányába a hálózatok mentén. A tárolás és különösen a továbbítás költsége erősen függ a szóban forgó adatok mennyiségétől. Emiatt természetes az igény, hogy bizonyos esetekben adatainkat minél tömörebb, kompaktabb formában ábrázoljuk. Az *adattömörítés* mára valóságos tudományággá vált. Se szeri, se száma a különböző információfajták (pl. szöveg, kép, hang) összenyomására, illetve kibontására szolgáló módszereknek. Itt a legelső, legegyszerűbb eljárást, a ma már klasszikusnak számító Huffman-kódolást ismertetjük, majd a széles körben használatos Lempel–Ziv–Welch-módszer ötletét mutatjuk be.

5.1. A Huffman-kód

Tegyük fel, hogy egy a $b_1, \ldots, b_n$ betűkből álló szöveget szeretnénk bitsorozatként kódolni. A b_i betűk valamely abc jelei, és egy ezekből szerkesztett hosszú (többnyire n-nél sokkal hosszabb) sorozat gazdaságos kódolása a célunk. Az egyes betűk kódjait (a nekik megfelelő bitsorozatokat) úgy akarjuk megválasztani, hogy a szöveg kódjának hossza minimális legyen. Ismert még az egyes betűk $q_1, \ldots, q_n$ gyakorisága a szövegben. Ha *uniform*, tehát ugyanolyan hosszú sorozatokat használunk a kódolásra, akkor nem sok játékterünk marad, hiszen a gyakoriságoktól függetlenül (feltéve, hogy ezek mind pozitívak) $\lceil \log_2 n \rceil$ a legrövidebb lehetséges kódhosszúság egy betűre. Ha megengedünk eltérő hosszú kódszavakat, akkor a dekódolással, a szövegnek a bitsorozatból való visszanyerésével lehet probléma. Olyan kódot szeretnénk tehát, ahol a betűk kódszavainak hossza nem feltétlenül

azonos, és a kód egyértelműen visszafejthető. Egy lehetséges megoldást kínál a következő definíció:

Definíció: *Egy bitsorozatokból álló kód* prefix kód, *ha egy betű kódja sem prefixe (kezdőszelete) egy másik kódjának. Formálisan: ha x és y két különböző betű kódja, akkor nincs olyan z bitsorozat, melyre $xz = y$.*

Egy prefix-tulajdonságú kóddal leírt üzenet egyértelműen visszafejthető. Az üzenet elejétől addig megyünk a bitek mentén, amíg egy betű kódszavát kapjuk. A prefix-feltétel miatt csak ez lehetett az első betű. Innen ugyanígy folytatva sorban megfejthetjük a további betűket.

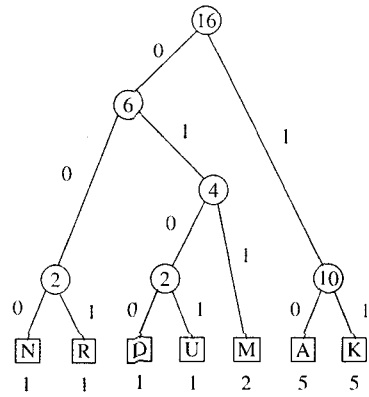
Egy prefix kódhoz természetes módon bináris fát rendelhetünk. Vegyünk először egy elég nagy teljes bináris fát. Ennek balra menő éleit címkézzük nullával, a jobbra menőket pedig egyessel. Ebben a fában egy a gyökérből lefelé menő út kölcsönösen egyértelműen leírható egy bitsorozattal, az érintett élek címkéinek sorozatával. Nézzük meg a b_i betűk kódjainak megfelelő utakat. Ezek alsó végpontjait címkézzük meg b_i-vel. A prefix-tulajdonság azt jelenti, hogy b_i nem őse b_j-nek, ha $i \neq j$. Ezért ha az előző utak által nem érintett csúcsokat és az ezekhez illeszkedő éleket eldobjuk a fából, akkor egy olyan bináris fát kapunk, melynek levelei pontosan a $b_1, \ldots, b_n$ címkéjű csúcsok. Megfordítva: egy bináris fából, melynek levelein a b_i címkék vannak (különböző leveleken különbözők), a kézenfekvő módon egy prefix kódot kapunk.

Az előző megfeleltetést figyelembe véve a szövegtömörítési probléma prefix kód alkalmazása esetén így fogalmazható: azon bináris fák között, melynek levelei $b_1, \ldots, b_n$, keressünk olyat, amelyre a $\sum q_i h(b_i)$ összeg minimális, ahol egy x csúcsra $h(x)$ a gyökértől x-ig vezető úton bejárt élek száma. Valóban, $h(b_i)$ éppen a b_i betű bitkódjának a hossza, így az összeg éppen a kódolt szöveg hosszát méri.

Az előző értelemben optimális konstrukció a *Huffman-fa*. Az algoritmus egy bináris fát épít, melynek a csúcsait pozitív valós számokkal címkézzük. Kezdetben n izolált csúcspontunk van. Ezek felelnek meg a b_i betűknek, és ezek lesznek végül a Huffman-fa levelei. A b_i-nek megfelelő csúcs címkéje q_i. Az általános lépés a következő. Tegyük fel, hogy már megépítettük az $S_1, \ldots, S_k$ fákat, ezek gyökérpontjai $x_1, \ldots, x_k$, utóbbiak címkéi az $r_1, \ldots, r_k$ számok. Ekkor vesszük a két minimális címkéjű gyökeret (legyenek ezek x_i és x_j). Ezek fölé egy új y gyökérpontot teszünk, melynek fiai x_i és x_j. Az y címkéje $r_i + r_j$. Ezzel a fák száma eggyel csökken. Megállunk, ha már csak egy fa marad. Összesen tehát $n - 1$ ilyen összevonó lépés szükséges.

Példaként tekintsük a KAKUKKMADARAMNAK szó kódolását. Ebben hét betű szerepel, a legrövidebb lehetséges uniform kódhossz betűnként 3 bit. A 16 betűs szó kódolása így 48 bitet igényel. Nézzük, mit ad a Huffman-kód. A betűk

gyakoriságai (a q_i értékek): A: 5, K:5, M: 2, U: 1, D: 1, R: 1, N: 1. Egy lehetséges kezdés, hogy először az N, és R csúcsokat (fákat) vonjuk össze, majd a D és U csúcsokat, utána az utóbbi fa gyökerét az M csúccsal. Az ábra egy ilyen fát mutat.



A betűk kódjai: K: 11, A: 10, M: 011, U: 0101, D: 0100, N: 000, R: 001. A szó kódjának a hossza így 40 bit, ami megtakarítást mutat az uniform kódoláshoz képest.

A módszer egyszerűsége mellett figyelemre méltó, hogy a prefix kódok közül a lehető legjobb tömörítést eredményezi. Nevezetesen igaz a következő:

Tétel: *A Huffman-fa optimális. Pontosabban fogalmazva, a Huffman-fa esetén az* $I = \sum q_i h(b_i)$ *összeg minimális azon bináris fák között, amelyek levelei* $b_1, \ldots, b_n$.

Bizonyítás: A Huffman-fa által adott I érték legyen $H(q_1, q_2, \ldots, q_n)$. Tegyük fel még, hogy $q_1 \leq q_2$ a két legkisebb gyakoriság. A fa konstrukciójából adódik, hogy

$$H(q_1, \ldots, q_n) = H(q_1 + q_2, q_3, \ldots, q_n) + q_1 + q_2.$$

Indoklásul megjegyezzük, hogy az első összevonó lépés után $n-1$ gyökérpont van, melyek címkéi rendre $q_1 + q_2, q_3, \ldots, q_n$. Az ezek feletti fa éppen egy Huffman-fa, melynek I-értéke $H(q_1+q_2, q_3, \ldots, q_n)$, amiből az eredeti fa I-értéke $q_1 + q_2$ hozzáadásával kapható. Jelölje $Opt(q_1, q_2, \ldots, q_n)$ a q_i gyakoriságok esetén elérhető optimális I-értéket.

Lemma: *Legyen továbbra is* $q_1 \leq q_2$ *a két legkisebb gyakoriság. Ekkor*

$$Opt(q_1, q_2, \ldots, q_n) = Opt(q_1 + q_2, q_3, \ldots, q_n) + q_1 + q_2.$$

A lemma bizonyítása: Tegyünk először néhány megállapítást az optimális fákról. Egy ilyenben feltehető, hogy minden belső csúcsnak két fia van. Ha ugyanis az y csúcsnak csak egy fia volna, akkor y-t törölhetnénk az egy szem fiát téve a helyére, ezzel csökkentve I értékét. Legyen most x egy levél az optimális fánkban, melyre $h(x)$ a lehető legnagyobb. Az előbbi fejtegetés szerint x-nek van a fában egy y testvére. Feltehető ezután, hogy x és y címkéi q_1 és q_2, hiszen a megfelelő cserék nem növelik I értékét. Töröljük egy pillanatra az x és y csúcsokat, és írjuk a keletkező új levélbe a $q_1 + q_2$ címkét. A fa I-értéke legyen J. Kapjuk azonnal, hogy $Opt(q_1, q_2, \ldots, q_n) = J + q_1 + q_2$, amiből az optimalitás miatt következik, hogy az új fának is optimálisnak kell lennie a $q_1 + q_2, q_3, \ldots, q_n$ gyakoriságokra vonatkozóan. Ez azért igaz, mert ha J-nél kisebb értékű megoldás is lenne, akkor ezzel az eredeti fán is tudnánk javítani. Mindebből arra jutunk, hogy

$$Opt(q_1, q_2, \ldots, q_n) = Opt(q_1 + q_2, q_3, \ldots, q_n) + q_1 + q_2,$$

ami éppen a lemma állítása. □

Ezután a tétel könnyű indukció n szerint. Az $n = 2$ esetben a Huffman-fa nyilván optimális, vagyis tetszőleges p és q gyakoriságokkal teljesül, hogy $Opt(p, q) = H(p, q)$. Tegyük fel mármost, hogy legfeljebb $n - 1$ betű esetén igaz a tétel állítása, és legyen $b_1, \ldots, b_n$ egy n elemű betűkészlet, melyre a gyakoriságok $q_1 \leq q_2 \leq \ldots \leq q_n$. Az indukciós feltevés szerint $Opt(q_1 + q_2, q_3, \ldots, q_n) = H(q_1 + q_2, q_3, \ldots, q_n)$. A H-ra vonatkozó kiemelt formula és a lemmabeli formula jobboldalán levő mennyiségek tehát egyenlők. Ebből következik, hogy a baloldalak is megegyeznek:

$$Opt(q_1, q_2, q_3, \ldots, q_n) = H(q_1, q_2, q_3, \ldots, q_n).$$

□

A módszer hátránya, hogy alkalmazásához kétszer kell végigolvasni a kódolandó szöveget. Az első olvasatban meghatározzuk a $q_1, q_2, \ldots, q_n$ gyakoriságokat, ezek alapján felépítjük a Huffman-fát, és a második menetben sorra elkódoljuk a szöveg betűit. Úgy is fogalmazhatunk, hogy a kódolás csak akkor kezdődhet, ha már az egész szöveg a rendelkezésünkre áll. Ez egy sor alkalmazásnál eléggé kényelmetlen. Vannak a Huffman-módszernek *dinamikus* változatai. Ezek a szöveg eddig olvasott részéből számolnak gyakoriságokat, és az így kapott fa szerint kódolják a következő betűt. Ennél a megközelítésnél a kódszavak fája betűről betűre változhat. Az ilyen egymenetes módszerek nem érnek el ugyan optimális összenyomást, de a tömörítő képességük még mindig elég kedvező. Ugyanakkor gyorsak, és megfelelnek az interaktív alkalmazások igényeinek. Ilyen megoldás van néhány régebbi UNIX-változat (mint pl. a 4.2 BSD UNIX) *compact* eljárásában.

5.2. A Lempel–Ziv–Welch-módszer

A Huffman-algoritmusoktól gyökeresen eltérő szemléletű módszercsaládot javasolt A. Lempel és J. Ziv a hetvenes évek második felében. Ötleteiknek egy szép és egyszerű megvalósítását dolgozta ki T. Welch 1984-ben. A Lempel–Ziv–Welch-módszernek nevezett eljárás ma az egyik legnépszerűbb információtömörítő megoldás. A módszer a Huffman-algoritmussal szemben nem betűnként kódolja az üzenetet. A szöveg bizonyos szavaiból *szótárat* épít. Ez szavak egy halmaza, amit S-sel fogunk jelölni. A szótárról három dolgot tételezünk fel:

– az egybetűs szavak, azaz Σ elemei mind benne vannak S-ben;
– ha egy szó benne van a szótárban, akkor annak minden kezdődarabja is benne van;
– a szótárban tárolt szavaknak fix hosszúságú kódjuk van; az $x \in S$ szó kódját $c(x)$ jelöli.

A gyakorlatban a kódok hosszát 12-15 bitnek érdemes választani. Az algoritmus az összenyomni kívánt szöveget S-beli szavak egymásutánjára bontja. A kódolás eredménye, az összenyomott szöveg az így kapott szavak kódjainak a sorozata. Az eredeti szöveg olvasásakor gyakorlatilag egyidőben épül/bővül az S szótár és alakul ki a felbontás. A helymegtakarítás, a tömörítés abból adódik, hogy sokszor helyettesítünk hosszú szavakat a rövid kódjaikkal. Az S építését és a szöveg felbontását is szinte banálisan egyszerű mohó stratégiával kezeli a módszer. Nyoma sincs benne teljes körű optimalizálásnak, ami a Huffman-algoritmus jellemzője. Pusztán az éppen vizsgált kis szövegdarab ismeretében alakulnak ki az érdemi döntések. A megoldás mégis rendkívül hatékony akár a sebességet tekintjük, akár az összenyomás mértékét.

A szótár egyik szokásos tárolási módja a szófa adatszerkezet. Az $x \in S$ szó $c(x)$ kódja úgy is tekinthető, mint egy hivatkozás (mutató) az x-nek az S-beli előfordulására. A szöveg összenyomása úgy történik, hogy amikor az olvasás során egy $x \in S$ szót találunk, aminek a következő Y betűvel való folytatása már nincs S-ben, akkor $c(x)$-et kiírjuk a kódolt szövegbe. Az xY szót felvesszük az S szótárba. A szó $c(xY)$ kódja a legkisebb még eddig az S-ben nem szereplő kódérték lesz. Ezután az Y betűvel kezdődően folytatjuk a bemeneti szöveg olvasását. Az algoritmus kicsit pontosabb megfogalmazásához legyen z egy szó típusú változó, K egy betű típusú változó. A z változó értéke kezdetben az összenyomni szánt állomány első betűje. Az eljárás futása során mindig teljesül, hogy $z \in S$. Az általános lépés a következő:

(0) Olvassuk a bemenő állomány következő betűjét K-ba.
(1) Ha az előző olvasási kísérlet sikertelen volt (vége a bemenetnek), akkor írjuk ki $c(z)$-t, és álljunk meg.
(2) Ha a zK szó is S-ben van, akkor $z \leftarrow zK$, és menjünk vissza (0)-ra.
(3) Különben (azaz ha $zK \notin S$) írjuk ki $c(z)$-t, tegyük a zK szót S-be. Legyen $z \leftarrow K$, majd menjünk vissza (0)-ra.

A módszer S-beli szavak egymásutánjára bontja a bemeneti szöveget. A felbontásban szereplő következő szónak mindig a leghosszabb lehetséges S-beli szót választja. Ez az ötlet, amit *mohó elemzésnek* neveznek, számos más szövegkezelő eljárásban is használható.

Ahogy korábban említettük, a $c(x)$ kódok rögzített hosszúságúak. Ha például ez a hosszúság 12 bit, akkor az S szótárba összesen 4096 szó kerülhet. Mi történjék akkor, ha a kódolás során kimerül ez a keret? Ha a bemeneti szöveg elég hosszú, akkor ez könnyen megtörténhet. Ebben az esetben a Lempel-Ziv-Welch-algoritmus felhagy a szótár további bővítésével. A (3) lépésben a zK szó S-be illesztését előíró rész többé nem hajtódik végre. A mohó elemzés a „betelt" S szótárra alapozva folyik tovább.

Kövessük az algoritmus működését egy példán. Az egyszerűség kedvéért legyen $\Sigma = \{a, b, c\}$. Kezdetben ezek az egybetűs szavak alkotják a szótárat. A szavak kódjai pozitív egészek legyenek; $c(a) = 1$, $c(b) = 2$, $c(b) = 3$, és az S-be kerülő új szavakhoz mindig a legkisebb még nem használt pozitív egészet rendeljük kódként. Legyen a bemenő szöveg $ababcbababaaaaaaa$. Kezdetben z értéke a. A K első értéke b lesz. Mivel $zK = ab$ nincs S-ben, először kiírjuk az eredmény első jeleként a $c(a) = 1$-et, az ab szót S-be illesztjük, és hozzá a négyest rendeljük kódként. A z értéke a lépés befejeztével b lesz. A következő lépésben $K = a$. Mivel $zK = ba$ nincs S-ben, az előzőhöz hasonlóan járunk el. Kiírjuk $c(b) = 2$-t, ba az S-be kerül, és a kódja 5 lesz, végül pedig $z = a$. Az általános lépés következő végrehajtásakor kerül először (2)-re a vezérlés, mivel ekkor már $ab \in S$. A következő input betű olvasása után $K = c$, $z = ab$. Mivel $zK = abc \notin S$, kiírjuk ab kódját, ami 4, az abc szót S-be illesztjük. A szó kódja 6 lesz, és végül $z = c$. Innen a folytatást az olvasóra bízzuk. A mohó elemzés által kapott felbontás $a|b|ab|c|ba|bab|a|aa|aaa|a$, a szöveg kódja pedig 1 2 4 3 5 8 1 10 11 1 lesz. Az S végső helyzetét és a benne levő szavak kódjait tartalmazza a következő táblázat:

a	1
b	2
c	3
ab	4
ba	5
abc	6
cb	7
bab	8
$baba$	9
aa	10
aaa	11
$aaaa$	12

A dekódolás, az összenyomott szöveg visszafejtése is elég egyszerű, de a kódoló eljárásnál azért valamivel bonyolultabb. A dekódolás első pillantásra ördöngösnek tetsző vonása, hogy *elvégezhető pusztán az egybetűs szavak kódjainak, valamint a kódok képzési szabályának ismeretében.* Nem kell tehát az egész S-et és a kódtáblát tárolni illetve elküldeni. Nem ismertetjük itt a teljes eljárást, csak az ötlet érzékeltetésére szorítkozunk. A lényeges mozzanat az, hogy a kódolt szöveget olvasva az egyes rész-szavak visszafejtése mellett rekonstruáljuk az S halmazt az elemeihez tartozó kódértékekkel együtt. Lássuk, hogyan boldogulunk az előbb kapott 1 2 4 3 5 8 1 10 11 1 sorozattal! Az első két jel betű kódja, ezeket hivatalból ismerjük. A szöveg első két betűje tehát ab. Ez a szó nem volt a kezdeti S-ben, de az eleje igen; innen rájövünk, hogy a kódoló algoritmus ezt $c(ab) = 4$ értékkel S-be tette. Ezt tudva a harmadik jel is megfejthető, így már a bemeneti szöveg első négy betűjét ismerjük: $abab$. Ebből a szövegdarabkából a kódoló algoritmus „fejével gondolkodva" kiderül, hogy ba volt a következő szó, ami S-be került, és ezáltal $c(ba) = 5$. Ebben a stílusban haladunk tovább. Legtöbbször a már megismert kódok közül kerül ki a következő visszafejtendő szám. Ennek megfejtése után pedig a bemeneti szöveg egy hosszabb szeletét ismerjük meg, amivel pedig az S szótárat építhetjük tovább.

Egyetlen eset van, amikor látszólag elakadunk, amikor a megfejtendő kód nem lesz az eddig kibontott szövegrész kódolása során kapott kódok között. Az összenyomást végző algoritmus ismeretében ekkor is tovább tudunk lépni. A helyzetet egy példával illusztráljuk. Tegyük fel, hogy az előbbi 1 2 4 3 5 8 1 10 11 1 sorozatot szeretnénk kibontani, és az ötöst már feldolgoztuk. Az eddigi kibontott szöveg $ababcba$, amin a kódoló munkáját utánozva felépíthetjük az S szótárat egészen a hetes kódértékű cb szóig. Ekkor látszólag bajban vagyunk, hiszen nyolcas kódú szó még nincs S-ben. Mi lehet ez az x szó? Az összenyomó módszer működését ismerve megállapíthatjuk, hogy az x szó $ba*$ alakú, ahol $*$ egy betű. Indoklásul

emlékeztetünk arra, hogy a kódoló algoritmus a mohó elemzés során utoljára leválasztott S-beli szót (ami esetünkben az ötös kódú ba) egészíti ki egy betűvel, és ezt teszi S-be.

Az x elejéből ezután kiderül az eredeti szöveg további két betűje: az ismert rész $ababcbaba$ lesz, amiből látjuk, hogy a $*$ betű csak b lehet, vagyis $x = bab$. A megfejtést tehát ekkor is folytatni tudjuk.

A Lempel–Ziv–Welch-módszer programja rövid és viharsebes. Mind a kódolás, mind pedig a dekódolás során csak egyszer kell olvasnunk a bemeneti jelsorozatot. A módszer gyorsabb a legjobb dinamikus Huffman-változatoknál, és erőteljesebb tömörítést is ér el azoknál a tipikus alkalmazások során (programszövegek, természetes nyelvű szövegek, grafika feldolgozása). A tömörítő képesség elemzése komoly statisztikai-információelméleti eszközöket igényel, így attól itt eltekintünk. Empirikus adatként megemlítjük, hogy a szokásos alkalmazásoknál $50 - 60\%$-ra teszik a módszerrel elérhető hosszmegtakarítást. A Lempel–Ziv–Welch-algoritmust használja több ismert és népszerű tömörítő program, így például az újabb keletű UNIX változatok *compress* függvénye. Hasonló ötleteken alapulnak a széles körben használatos *zip* és *gzip* eljárások is.