Algoritmuselmélet és bonyolultságelmélet MSc hallgatók számára

Adattömörítés, LZW algoritmus

2018. Előadó: Hajnal Péter

1. Kódolás

Gondolatainkat szeretjük egymásnak átadni. Ehhez a nyelvet használjuk. Ha közvetlenül nem kommunikálunk, akkor az írást használjuk. A gondolatainkat karaktersorozat formájába öntjük. Egy Σ véges ábécéből (elemeire mint karakterekre hivatkozunk) felépített szavak (véges karaktersorozatok) halmaza Σ^*. Az írás által, gondolataink „kódja" Σ^* egy eleme lesz.

Gyakran Σ^* elemeit tovább kódoljuk. Ekkor karaktersorozathoz egy másik karaktersorozatot rendelünk. Miért tesszük ezt? Több okunk is lehet. Néhányat felsorolok:

- **Egységesítés.** A mai világban előnye van annak, ha standard karakterkészletű eszközökkel dolgozunk.
- **Titkosítás.** A kezdeti $\sigma \in \Sigma^*$ üzenet értelmezése nyilvánvaló, mi pedig szeretnénk ezt titokban tartani. Ha van egy $\tau : \Sigma^* \to \Sigma^*$ titkosító kódolálás és egy $\delta : \Sigma^* \to \Sigma^*$ dekódoló eljárás, akkor σ helyett $\kappa = \tau(\sigma)$ karaktersorozatot küldjük el. A kapott karaktersorozatból $\delta(\kappa)$ adja az eredeti üzenetet. Ha nem ismert δ, akkor nehéz lehet „feltörni" a titkosítást.
- **Hiba javítás.** Az elküldött $\sigma \in \Sigma^*$ üzenetbe hiba csúszhat. A fogadó oldalon $\widetilde{\sigma}$ jelsorozat érkezik. Tudjuk, hogy adott hosszú üzenet esetén mennyi hiba csúszhat be. Ezt tudva Σ^* egy ritka, szétszórt részhalmazát választva ki mint lehetséges üzenetek reményünk marad a hibák tolerálására.
- **Tömörítés.** A mindennapos nyelvben redundanciák vannak. Adott hosszban jóval kevesebb értelmes szöveg van, mint amit a hossz megenged. Így σ rövidebb helyre is leírható, tárolható. A manapság használt kép, video, hang file-ok többsége nemcsak kódolt, de tömörített is.

A különböző célok különböző módszerekhez vezettek. A középső két cél külön kurzust érdemel. Mi itt érintjük az egységesítést és egy algoritmust említünk a tömörítés témaköréből.

2. Karakter-kódolás fix hosszal

Minden karakter kapjon egy fix hosszú 01 sorozatot mint kódot. Ha ℓ karaktert tárolunk így egy file-ban, akkor a file ℓ-lel arányos bitet tartalmaz. Ha az egyes karakterekhez használt bitsorozat hossza rövid, akkor kisebb file-ban tároljuk az adatot, de kevesebb karaktert használhatunk.

Példa. ASCII. A név az angol *American Standard Code for Information Interchange* elnevezésből ered. 1960 óta használják és elterjedése után a legelterjedtebb kódolássá vált. 128 karaktert 7 hosszú bitsorozatokkal kódoltak.

Több karakter használata (például az Euro, Font jel) volt kívánatos. A 2000-es években elterjedtek a 8 bites karakterkészletek.

A fenti standard kódolások (az ezeket leíró táblázatok) könnyen elérhetők az interneten. Céljuk a standardizálás. Folyó szöveg esetén a nyelvben lévő redundanciákat nem használják ki. Szakszöveg esetén a gyakran ismétlődő szavakat újra és újra leírjuk. A kódolt szövegben ezek az ismétlések megmaradnak.

Vannak jobb kódoló eljárások, ahol valódi tömörítés történik.

3. Karakter-kódolás változó hosszal

Az ötlet: a karaterek 01 kódjának hossza változzon. A gyakoribb karakterek kódjának hossza rövidebb legyen.

Példa. Morse-kódolás. A távíró elterjedése után, a távíróhoz készült kódolás. Nem tökéletes bináris kód. A nyilvánvaló rövid és hosszú jel között rövid és hosszú szünetek is szerepelnek, hogy látszódjon egy-egy karaktert kódoló hosszú/rövid sorozat mikor is ér véget. Az angol ábéce két leggyakoribb karakteres `e` és `t`. Morse-kódjuk egyetlen Morse-karakter (rövid, illetve hosszú).

Érdekességként megemlítjük, hogy Morse célja a tömörítés is volt, de nem volt optimális. A végül elterjed nemzetközi Morse-ábécé nem Morse eredeti javaslata volt, hanem egy későbbi német javításból keletkezett.

A Morse-kódolás egy problémája, hogy különböző hosszú 01 sorozatok használata miatt problémás a teljes kód 01 folyamának szétszabdalása karakterkódokra. Ezt Morse különböző hosszú szünetek bevezetésével oldotta meg. A számítógépekhez jobban illik a prefix kódok használata. Ekkor két karakter-kód esetén nem fordul elő, hogy az egyik a másik prefixe (kezdőszelete) legyen. Ez a tulajdonság megoldja a szétszabdalás problémáját.

Példa. UTF-8. A név az angol *Unicode (or Universal Coded Character Set) Transformation Format – 8-bit* leírásból ered.

Vették az ASCII kódokat és egy 0-val kiegészítettek (előlről). Ezzel 8-bites — azaz byte-os/bájt-os — hosszt kaptak. Ha itt megállunk, akkor az ASCII egy pazarló változatát kapjuk. Tovább mentek: Két, három és négy byte-os karaktereket is hozzáadtak a készlethez. Az első byte kezdete mindig 0, 110, 1110 vagy 11110 aszerint, hogy 1, 2, 3 vagy 4 byte-os a kód. Ha van további byte egy karakter mögött, akkor a későbbi byte-ok kezdete 10. Így a bitsorozat szabdalása egyszerű lett.

Az érdeklődő olvasó kiszámolhatja összesen hány karakter kódolására képes az UTF-8.

Példa. A $ egy ASCII karakter, kódja decimálisan írva 26, binárisan 0011010.

€ nem ASCII karakter. Az UTF-8-ban természetesen szerepel. Kódja három byte-os, az érdekesség kedvéért kiírjuk: 11100010:10000010:10101100 (az : jelek nem része a kódnak, csak a byte-struktúrára mutatnak rá).

Manapság a file-ok többségét UTF-8-ban kódolják (az ASCII karaktereket használó file-ok is kompatibilisek az UTF-8-cal).

Példa. Hufmann-kódolás. Egy gyökeres bináris fát építünk fel. Minden nemlevél csúcsnak két gyereke lesz, az egyikhez vezető él címkéje 0, a másik gyerekhez vezető élen 1 szerepel. A karaktereket megfeleltetjük a levelekkel. Minden karakterhez tartozik egy gyökér-karakternek megfelelő út, amelyen szereplő címkék adják a karakter kódját. A mélyen fekvő levelek kódja hosszabb mint a „magasan lévő" levelek kódja.

Ha a fánk egy teljes, kiegyensúlyozott fa, akkor visszakapjuk a fix hosszú kódolási rendszert. Célunk, hogy fánk ne legyen kiegyensúlyozott. A gyakori karakterek magasabb levelekhez legyenek rendelve.

A legegyszerűbb kódolás úgy kedődik, hogy a kódolandó szövegről egy statisztikai felmérés készül. Az egyes betűk gyakorisága szabja meg hogyan épülnek be a készülő fába. A Hufmann-kódolás standard BSc-s anyag.

Itt csak azt használjuk ki, hogy bizonyos karaktereket sokszor használunk, másokat nem annyiszor gyakran. Azt a tényt, hogy bizonyos karakter-kombinációk, szavak gyakran együtt szerepelnek az nem jelenik meg a fenti megállapodásokban/algoritmusokban.

4. Szótár alapuló kódolás, LZW

Necsak karakterek kapjanak kódot, de hosszabb karaktersorozatok is. Azaz a kódolást ne egy táblázat írja le, aminek bal oldalán a karakterek a jobb oldalán 01 kódjuk álljon. A táblázat bal oldalán engedjünk meg hosszabb karakterkombinációkat is. Az ilyen kódolásokat szótár alapú kódolásoknak nevezzük. Nyilvánvaló a haszon. Magyarban az 'AZ' névelő, angolban a THÉ nagyon gyakran szerepel. Ez és sok más hasonló gyakoriság miatt a fenti ötlet természetes és hassznos.

Az első hatékony szótár alapú kódolást Lempel—Ziv fejlesztette ki, amit néhány évvel később Welch tökélesített. Az így kapott algoritmust Lempel—Ziv—Welsch-algoritmusnak nevezzük (röviden mint LZW kódolásként hivatkozunk rá). Mi az LZW algoritmus egy egyszerű változatát ismertetjük.

Kezdeti szótár. Kiindulunk egy kezdeti szótárból. Ez tartalmazhatja mondjuk az ASCII karaktereket és mellette kódjukat, amelyeket 0-127 decimális jelöléssel írunk le. A START és STOP is benne lesz szótárunkban a 128 és 129. Az ASCII kódok valójában 7 hosszú bitsorozatok. A mi szótárunk „ezen túlcsorog". Egy egyszerűsítés, hogy a szótárban levő kódokat fix hosszúnak (L) gondoljuk, mondjuk 10 hosszú bitsorozatnak. Persze ez azt jelenti, hogy a kódolható szavak száma csak 2^L (mondjuk 1024) lehet. Tehát az 'A' betű ASCII kódja 65, ami a 1000001 bithetes, de szótárunkban mint 0001000001 (három bittel hosszabban) jelenik meg. Üzenetünket 10 hosszú bitsorozatok fogják alkotni. Szótárunkban az első $128 + 2 = 130$ foglalt hely és $1024 - 130 = 894$ szabad hely van.

Kódolás I: Szöveg feldolgozás. Az üzenet egy ASCII karaktersorozat. A sorozat elemeit sorban olvassuk. Mindig számon tartjuk azt a pontot, ameddig az üzenetet feldolgoztuk. Tegyük fel, hogy üzenetünk az, hogy 'UZENET'. Ebben egy függőleges vonás jelzi ezt a pontot. Azaz, ha a feldolgozás 'UZE|NET', akkor az üzenet UZE részét már elküldtük, a NET továbbítása még hátra van. Kezdeti állapotunk '|UZENET'. A szöveg feldolgozás aktuális része mindiga |-nél kezdődik.

Kódolás II: A következő kódrészlet kiszámítása. Kezdetben elküldük a START-ot kódoló 128-at (megállapodásunk szerint mint 10 bit) és a feldolgozaottságot '|UZENET'-re állítjuk. A továbbiakban a | jel utáni részt olvassuk el mint egy karakter, mint két karakter és így tovább. Azaz, ha a feldolgozottság 'UZE|NET', akkor a folytatás N, NE, illetve NET. Megnézzük ezeket a szótárunkban. Az következő karakter (egy hosszú folyatás, fenti mintapéldában N') biztos benne lesz. A többi (NE, illetve NET) kezdetben nem volt szótárunkban, de később belekerülhetett. Tegyük fel, hogy szótárunk tartalmazza, hogy NE (kódja κ egy 10 hosszú bitsorozat), de NET nincs benne. Ekkor „örülünk", hogy megtaláltuk NE-t és elküldjük κ-t. A feldolgozottságot át állítjuk UZENE|T-re. Továbbá „szomorúak vagyunk", mert NET nem volt benne a szótárunkban. Ezt egy nagyon fontos része az LZW algoritmusnak könyveli el.

Kódolás III: Szótárbővítés. Az első folytatás, amit nem találtunk meg a szótárban azt **belerakjuk**. Esetünkben ez NET volt. A következő üres sor ez lesz (az első bővítésnél a bővítő karaktersorozat (legalább kettő hosszú) kódja 130 lesz). Egyetlen kivétel, amikor elérjük üzenetünk végét, a teljes feldolgozottságot. Ekkor a 129 elküldése zárja a kommunikációt.

Nagyon fontos, hogy lássuk „mi történik a másik oldalon".

Dekódolás I: A kapott üzenettöredék értelmezése. Kapunk egy 10 hosszú bitsorozatot. Az első sorozat mindig 128, a protolkol elindulását jelenti. Elővesszük az alap szótárt. A továbbiakban ezt az üzenettöredéket sose kapjuk újra. 129 jelentése is világos. Most feltesszük, hogy ettől eltérő az amit kaptunk.

Mindig egy olyan bit-tizest kapunk, amely a szótárunkban már benne van (ez egy **garantált** tulajdonsága az LZW algoritmusnak) és így ki tudjuk olvasni a jelentését. Az üzenet eddig elolvasott része után írjuk. Egyre hosszabb lesz az elolvasott rész.

Dekódolás II: A szótár virtuális bővítése. A szótárunkat „virtuálisan" kibővítjük. Tudjuk, hogy a túl oldalon az elégedetlenség szótár bővítsében nyilvánult meg nyitunk egy új sort. Tudjuk, hogy az utoljára elküldött szöveg egy ASCII karaktererrel való meghosszabbítása bekerülta szótárba. Nem tudjuk melyik, de a szótár következő sora, egy újabb 10 hosszú kóddal megnyitásra kerül.

Dekódolás III: A szótár bővítése. A korábbi virtuális sorban volt egy korábbi szó egy karakterrel való kibővítése az új töredék jelentését már tudjuk. Ennek első karaktere az eddig ismeretlen karakter. Az előző virtuális sort értelmezzük (valódi/teljes szótár sorrá válik)

Az algoritmust akkor értjük meg, ha látjuk futás közben.

Példa. Legyen az üzenetünk

EMMA MAMA MA EMEL

Nézzük meg mit csinál LZW.

(0) Elküldjük, hogy 128. Feldolgozottság |EMMA MAMA MA EMEL.

(1) 'E' egy ASCII karakter benne van már az kiinduló szótárban, kódja 69. 'EM' nincs a szótárban. Elküldjük, hogy 69, szótárunkhoz egy új sort adunk: 130 ≡ EM. Feldolgozottság E|MMA MAMA MA EMEL.

(2) ‘M’ egy ASCII karakter benne van már az kiinduló szótárban, kódja 77. ‘MM’ nincs a szótárban. Elküldjük, hogy 77, szótárunkhoz egy új sort adunk: $131 \equiv$ MM. Feldolgozottság EM|MA MAMA MA EMEL.

(3) ‘M’ egy ASCII karakter benne van már az kiinduló szótárban, kódja 77. ‘MA’ nincs a szótárban. Elküldjük, hogy 77, szótárunkhoz egy új sort adunk: $132 \equiv$ MA. Feldolgozottság EMM|A MAMA MA EMEL.

(4) ‘A’ egy ASCII karakter benne van már az kiinduló szótárban, kódja 65. ‘A_’ (A, majd egy szóköz) nincs a szótárban. Elküldjük, hogy 65, szótárunkhoz egy új sort adunk: $133 \equiv$ A_. Feldolgozottság EMMA| MAMA MA EMEL.

(5) ‘_’ egy ASCII karakter benne van már az kiinduló szótárban, kódja 32. ‘_M’ (egy szóköz, majd M) nincs a szótárban. Elküldjük, hogy 32, szótárunkhoz egy új sort adunk: $134 \equiv$ _M. Feldolgozottság EMMA |MAMA MA EMEL.

(6) ‘MA’ benne van már a bővített szótárban, kódja 132. (‘MA’ egy gyakori karakterkettes, már külön kódja van. A 2×7 bitnél — ami az ASCII leíráshoz kell — kevesebb, csak 10 bittel kódoljuk. Jól látszik a spórolás kezdete.) ‘MAM’ nincs a szótárban. Elküldjük, hogy 132, szótárunkhoz egy új sort adunk: $135 \equiv$ MAM. Feldolgozottság EMMA MA|MA MA EMEL.

(7) ‘MA’ benne van már a bővített szótárban, kódja 132. ‘MA_’ nincs a szótárban. Elküldjük, hogy 132, szótárunkhoz egy új sort adunk: $136 \equiv$ MA_. Feldolgozottság EMMA MAMA| MA EMEL.

(8) ‘_M’ benne van már a bővített szótárban, kódja 134. ‘_MA’ nincs a szótárban. Elküldjük, hogy 134, szótárunkhoz egy új sort adunk: $137 \equiv$ _MA. Feldolgozottság EMMA MAMA M|A EMEL.

(9) ‘A_’ benne van már a bővített szótárban, kódja 133. ‘A_E’ nincs a szótárban. Elküldjük, hogy 133, szótárunkhoz egy új sort adunk: $138 \equiv$ A_E. Feldolgozottság EMMA MAMA MA |EMEL.

(10) ‘EM’ benne van már a bővített szótárban, kódja 130. ‘EME’ nincs a szótárban. Elküldjük, hogy 130, szótárunkhoz egy új sort adunk: $139 \equiv$ EME. Feldolgozottság EMMA MAMA MA EM|EL.

(11) ‘E’ benne van már az eredeti szótárban, kódja 69. ‘EL’ nincs a szótárban. Elküldjük, hogy 69, szótárunkhoz egy új sort adunk: $140 \equiv$ EL. Feldolgozottság EMMA MAMA MA EME|L.

(12) ‘L’ benne van már az eredeti szótárban, kódja 76. Elértünk az üzenet végére. Elküldjük, hogy 76, nincs szótár bővítés. Feldolgozottság EMMA MAMA MA EMEL|.

(13) Elküldjük, hogy 129.

Nézzük meg a „másik oldalt”.

(0) Megkaptuk, hogy 128. „Kinyitjuk a szótárunkat.”

(1) Megkapjuk, hogy 69, Az eredeti (ASCII) szótár szerint ez ‘E’-t jelent, leírjuk. Szótárunkhoz egy új virtuális sort adunk: 130 ≡ E?.

(2) Megkapjuk, hogy 77, szótárunk alapján ez ‘M’. Az eddigiek után írjuk: EM Egy új virtuális sort adunk: 131 ≡ M?. Az előző virtuális sort teljessé tesszük az új üzenet első karakterével: 130 ≡ EM.

(3) Megkapjuk, hogy 77, értelmezzük és hozzáírjuk a korábbi szövegrészlethez: EMM. Szótárunkhoz egy új virtuális sort adunk: 132 ≡ M?. A korábbi sort teljessé tesszük: 131 ≡ MM.

(4) Megkapjuk, hogy 65, értelmezzük és hozzáírjuk a korábbi szövegrészlethez: EMMA. Szótárunkhoz egy új sort adunk: 133 ≡ A?. A korábbi sort teljessé tesszük: 132 ≡ MA.

(5) Megkapjuk, hogy 32, értelmezzük a szótásunk alapján és hozzáírjuk a korábbi szövegrészlethez: EMMA_. Szótárunkhoz egy új sort adunk: 134 ≡ _?. A korábbi sort teljessé tesszük: 133 ≡ A_.

(6) Megkapjuk, hogy 132, szótárunk alapján ez ‘MA’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MA. Szótárunkhoz egy részleges sort adunk hozzá: 135 ≡ MA?. A korábbi sort teljessé tesszük: 134 ≡ _M.

(7) Megkapjuk, hogy 132, szótárunk alapján ez ‘MA’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA. Szótárunkhoz egy új sort adunk: 136 ≡ MA?. A korábbi sort teljessé tesszük: 135 ≡ MA_.

(8) Megkapjuk, hogy 134, szótárunk alapján ez ‘_M’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA_M. Szótárunkhoz egy új sort adunk: 137 ≡ _M?. A korábbi sort teljessé tesszük: 136 ≡ _MAM.

(9) Megkapjuk, hogy 133, szótárunk alapján ez ‘A_’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA_MA_. Szótárunkhoz egy új sort adunk: 138 ≡ A_?. A korábbi sort teljessé tesszük: 137 ≡ _MAM.

(10) Megkapjuk, hogy 130, szótárunk alapján ez ‘EM’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA_MA_EM. Szótárunkhoz egy új sort adunk: 139 ≡ EM?. A korábbi sort teljessé tesszük: 138 ≡ A_E.

(11) Megkapjuk, hogy 69, ez egy ASCII karakter, az eredeti szótár is benne volt már, jelentése ‘E’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA_-MA_EME. Szótárunkhoz egy új sort adunk: 140 ≡ E?. A korábbi sort teljessé tesszük: 139 ≡ EME.

(12) Megkapjuk, hogy 76, ez egy ASCII karakter, az eredeti szótár is benne volt már, jelentése ‘L’ és utánaírjuk az eddigi szövegrészletnek: EMMA_MAMA_-MA_EMEL. Szótárunkhoz egy új sort adunk: 140 ≡ L?. A korábbi sort teljessé tesszük: 139 ≡ EL.

(13) Megkapjuk, hogy 129. A kommunikáció teljes. A szótárunkkal van egy kis baj. Az ‘L’ megkapásakor azt „vizionáltuk”, hogy a másik oldal elégedetlen. Nem volt az, látta hogy a teljes üzenet átment. A téves gondolat miatt a szótásból

a 140 ≡ L? jegyzést törölhetjük (különösen fontos, ha rendmániások vagyunk). Az üzenet

EMMA_MAMA_MA_EMEL

Megjegyezzük, hogy egy nyilvánvaló probléma, hogy egy szöveg esetén nehéz megbecsülni milyen nagy szótárra van szükségünk. A fenti példa esetén természetesen az üzenettöredékeket elég lett volna nyolc hosszúnak választani (1 plusz bit az ASCII-hoz képest). Egy hosszú szöveg esetén szótárunk kinőtte volna az 894 szabad helyet.

Ezt a problémát lehet kezelni, a szöveg olvasása közben megváltoztatni az üzenettöredékek hosszát. Ez azonban technikai problémákat vet fel: Ha tömörítendő szövegünk ASCII kódokat használ, akkor START, STOP és az első szótárbővitéseknél 8-hosszú bitsorozatokkal kódolunk (ez egy uniform hossz, az ASCII karakterek szokásos bitsorozat-kódját is egy 0-val kiterjesztjük). Eljutunk oda, hogy küldenünk kell egy csomagot és szótárunkat kibővítené az algoritmus egy szóval, aminek a 256 kód jut. Kifutunk a 8 bitből. Semmi baj. Mostantól kezdve bitkilencesekre kell szabdalni az átküldött sorozatot. De éppen esedékes egy csomagküldés. Ez nyolc (késői váltás) vagy kilenc bites (korai váltás) legyen? A fogadó fél egy kis késéssel „képben van". Azt például pontosan tudja, hogy a küldő oldalon mi az aktuális szótár hossza. Azaz tudja, hogy döntés előtt van a küldő oldal. Ha tudja, hogy a kódoló algoritmus a késői vagy korai megállapodás szerint vált csomaghosszt, akkor jól értelmezi a kapott információt.

Megjegyezzük, hogy az első hosszváltás után ASCII karaktereket és a szótárba rakott korai szavakat is a kilencbites csomagokban küldjük. Más eseteben kibogozhatatlan lenne a kapott bitsorozat.

A gyakorlatban a korai és késői váltás keverten jelenik meg az interneten is nagyon gyakran használt LZW alapú tömörítések között.