

Algoritmusok és adatszerkezetek II. gyakorlati segédlet

Tömörítés

Nagy Ádám - nagyadam888@gmail.com

2020. szeptember 7.

1. Adattömörítés

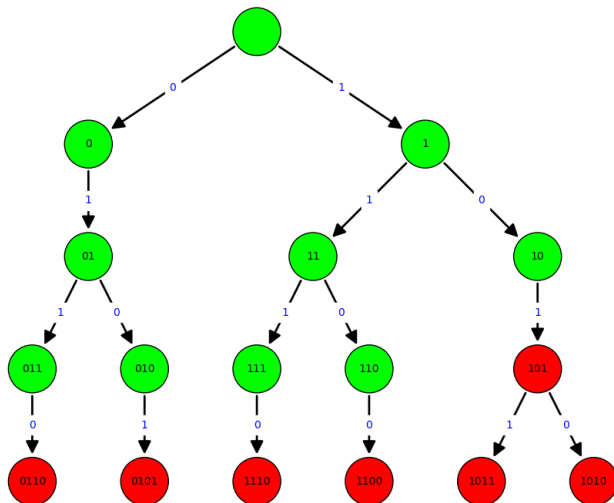
Informatikában a **kódoláselmélet** adatok különböző reprezentációjával és azok közötti átalakításokkal foglalkozik. Ennek egyik ága, a **forráskódolás** az adott alak hosszát vizsgálja; vagyis azt a kérdést, hogy az adott mennyiségű információt mekkora mennyiségű adattal lehet tárolni. Legtöbb esetben a cél a rövidebb reprezentáció, tehát beszélhetünk információ- vagy adattömörítésről.

A kódolás során fontos kérdés, hogy az adat teljes egészében visszaállítható-e. Tömörítés esetében ennek megfelelően használhatunk **vesztésmentes** vagy veszteséggel járó eljárásokat (pl. JPEG, MPEG, MP3, ...). Mi csak az előbbivel foglalkozunk.

A kódoláselméletnél meg kell adnunk az **információ alapegységét**, azaz azt, mennyi információtartalma van az atomi „tárolási egységnek”. Mivel a jelenlegi számítógépek bináris elven működnek, ez $r = 2$ és így a kódszavaink a $\Gamma = \{ '0', '1' \}$ ábécé feletti szavak lesznek.

Kódnak nevezzük a Γ feletti véges szavak (kódszavak) egy tetszőleges nem üres halmazát. Például bináris kód a

$$C = \{ '0110', '1010', '1100', '1011', '101', '0101', '1110' \}.$$



1. ábra. Kódfa a C kód esetén, kódszavak pirossal jelölve

Egy kód szemléletesebb ábrázolásához elkészíthetjük annak **kódfáját**. Ebben a fában a fa csúcsai szavak (nem feltétlenül kódszavak), az éleit pedig a kódszavak lehetséges karaktereivel címkézzük. A fa gyökerében az üres szó

szerepel és egy szóhoz tartozó csúcs leszármazottai azok a szavak, amelyeket úgy kapunk, hogy a szó után írjuk az élen szereplő karaktert. A kódhoz tartozó kódfa az a legkevesebb csúcsot tartalmazó ilyen tulajdonságú fa, ami tartalmazza az összes kódszót.

A kódfa szemléltetés mellett más szempontból is hasznos lehet. Egyrészt a fa tulajdonságaiból következtethetünk a kód tulajdonságaira, másrészt a kódfa segítségével egy bitsorozat hatékonyan dekódolható: A gyökérből indulva a bitek szekvenciájának megfelelően járjuk be a fát, az élek mentén kódszavat keresve és találat esetén ismételve a bejárást megkapjuk a dekódolt adatot¹.

A kódolást **betűnkénti kódolás**nak nevezzük, ha az eredeti Σ ábécé feletti adatot betűnként egy $\Sigma \mapsto C \subset \Gamma^*$ kölcsönösen egyértelmű (bijektív) leképezéssel készítjük el. Például az ASCII kódolás is ilyen, hiszen a megfelelő táblázat alapján betűnként történik a kódolt adat kiszámolása.

1.1. Egyenletes kód – Naiv módszer

Egy kódot **egyenletes kód**nak nevezzük, ha a kód szavainak hossza egyenlő. A **naiv módszer** egyenletes kódot használó betűnkénti kódolás.

A Σ ábécé feletti kódolt adat akkor lesz a legkisebb, ha a kódszavak közös hossza a legkisebb. Mivel $|\Gamma| = r$ és $|\Sigma| = d$ ez azt jelenti, hogy az egyes karakterek legkevesebb $\lceil \log_r d \rceil$ hosszal kódolhatóak naiv módszer segítségével.

1.1.1. Feladatok

Tekintsük a $S = \text{AABCAADEAAB}$ szöveget.

1. Naiv módszert alkalmazva hány bittel tudjuk ábrázolni az S -t?
2. Adjuk meg S egy kódolt alakját naiv módszer esetén!
3. Adjunk meg betűnkénti kódolást, amely tömörebb eredményt ad mint a naiv módszer!
4. Adjunk meg egy egyenletes kódolást, amely tömörebb eredményt ad mint a naiv módszer!
5. Rajzoljuk fel a korábban használt kód kódját! Milyen tulajdonsággal rendelkezik egy egyenletes kódhoz tartozó kódfa?

¹Természetesen ez csak akkor igaz, ha a dekódolás egyáltalán lehetséges és egyértelmű.

Gyakorlatban, ha a tömörítés nem igazán fontos szempont, egyszerűsége miatt sok helyen alkalmazzák, például a 8 bit hosszúságú kódszavakat használó ASCII kód is ilyen.

1.2. Huffman-kódolás

Intuitíven, betűnkénti kódolás esetén akkor kapunk rövidebb kódolt adatot, ha a gyakori betűkhöz rövid kódszót, a ritkákhoz pedig hosszabbakat rendelünk.

Ez a tulajdonság jellemezhető matematikailag is. A kódolt és eredeti üzenet karakterei önmagukban is, de leginkább az üzenetben elfoglalt helyük segítségével információt tárolnak. Arra hogy mennyi információt hordoz egy üzenet a hosszához viszonyítva a karakterek rendezetlensége utal. Például egy egyetlen karaktert ismételtető forrás által küldött üzenet információ-mennyisége kisebb mint egy olyané ami ugyanolyan hosszú, több karaktert használ és a karaktereket valamilyen bonyolultabb szabály szerint fűzi egymás után. Erre a rendezetlenségre és így a relatív információ-mennyiségre utal az **entrópia**, amely ha az egyes karakterek előfordulásának valószínűsége p_1, p_2, \dots, p_d S -ben, akkor értéke

$$H_r(S) = - \sum_{i=1}^d p_i \log_r p_i.$$

Az entrópia értéke akkor a legkisebb (0), ha az üzenet csak egy karaktert tartalmaz; és akkor a legnagyobb ($\log_r d$), ha minden karakter azonos valószínűséggel szerepel, azaz ugyanannyi információt tárol. Ebből közvetlenül tudunk következtetni, hogy egy szöveg mennyire tömör, mivel az entrópia megadja hogy a karaktereket mennyire „jól” alkalmazzuk adott hosszban.

Shannon tétele (zajmentes csatornára) kimondja, hogy betűnkénti kódolást alkalmazva egy S ℓ hosszú adat kódolt változata legalább $H_r(S) * \ell$ hosszú kell legyen. Másképpen ha az S -ben található karakterekhez tartozó kódszavak átlagos szóhossza $\bar{\ell}$, akkor az S adat esetén $H_r(S) \leq \bar{\ell}$. Következésképpen egy szöveg entrópiája alulról korlátozza az átlagos szóhosszt. Egy kódolás annál tömörebb minél jobban sikerül ezt az értéket megközelítenie.

Egy betűnkénti kódolást **optimálisnak** nevezünk egy S adat kódolásánál, ha

$$H_r(S) \leq \bar{\ell} < H_r(S) + 1.$$

Ez azt jelenti, hogy az eredeti szöveg kódolásánál használt karakterekhez tartozó kódszavak hossza kevesebb mint 1-el térhet el átlagosan az elérhető elméleti határtól.

A Huffman-kódolás egy betűnkénti optimális kódolás, azaz az ilyen kódolások között szinte a legjobb tömörítés érhető el vele adott adat esetén. Ezt úgy érzük el, hogy a kódhoz tartozó kódfat alulról felfelé építjük az eredeti szöveg karaktereinek gyakorisága alapján. Bináris ($r = 2$) esetben a következőképpen járunk el:

1. Olvassuk végig a szöveget és határozzuk meg az egyes karakterekhez tartozó gyakoriságokat.
2. Hozzunk létre minden karakterhez egy csúcsot és helyezzük el azokat egy (min) prioritásos sorban a gyakoriság mint kulcs segítségével.
3. (i) Vegyünk ki két csúcsot a prioritásos sorból és hozzunk létre számukra egy szülő csúcsot.
(ii) A szülő-gyerek éleket címkézzük nullával és eggyel a gyakoriságnak megfelelő sorrendben.
(iii) Helyezzük el a szülő csúcsot a prioritásos sorba gyerekei gyakoriságának összegét használva kulcsként.

4. Ismételjük meg az előző pontot, ha több mint egy csúcs szerepel a sorban.
5. Olvassuk ki a karakterekhez tartozó kódszavakat a kódfából.
6. Olvassuk végig újra a bemenetet és kódoljuk azt karakterenként.

1.2.1. Megjegyzések

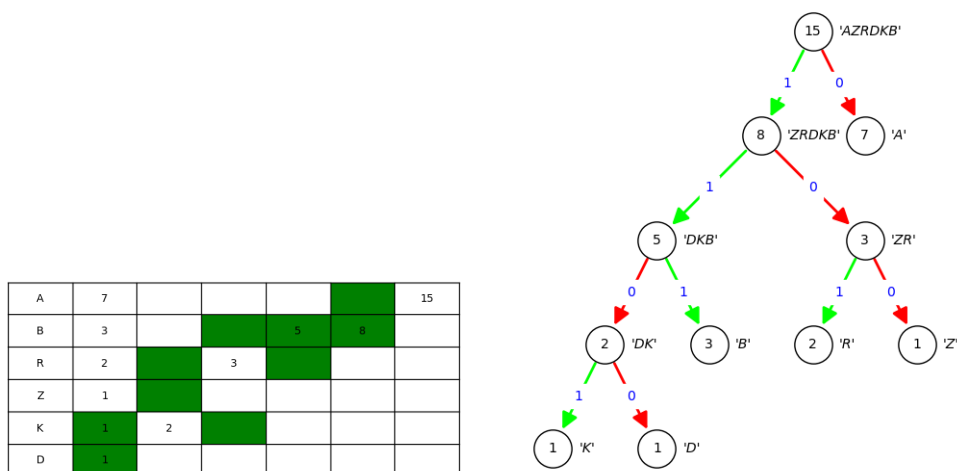
- A Huffman-kód mindig egyértelműen dekódolható (a kódfa segítségével), mivel egy prefix-kód. A **prefix-kód** egy olyan kód, amely esetén a kódszavak halmaza prefixmentes, azaz nincs két olyan kódszó, ami esetén az egyik a másiknak valódi prefixe lenne. Ez a tulajdonság a kódfára azt jelenti, hogy minden kódszóhoz tartozó csúcs levél.
- Általában a Huffman-kódolás nem egyértelmű. Egyrészt ha több azonos gyakoriság van, akkor bármelyiket választva Huffman-kódolást kapunk; másrészt a 0 és 1 szerepe felcserélhető.
- $r > 2$ esetben is hasonlóan kapjuk meg a Huffman kódot annyi különbséggel, hogy
 - az éleket 0-tól $r - 1$ -ig címkézzük és minden lépésben r csúcsot vonunk össze, kivéve
 - az első lépésben $k = d \bmod (r - 1)$ csúcsot kell összevonni, ha k egyenél nagyobb, különben $k + r - 1$ -et.
- Mivel a kódolás minden adathoz különböző, a dekódoló oldalon is ismertnek kell lennie. Ez gyakorlatban azt jelenti, hogy
 - a kódfát vagy kódtáblát is csatolnunk kell a kódolt adathoz (ront a tömörítési arányon), vagy
 - a Huffman-kódot általánosított adathoz készítjük el. Például magyar szöveg kódolásánál a magyar nyelv karaktereinek általános gyakorisága alapján (ált. tömörít, de nem az optimális kód).

1.2.2. Szemléltetés

Tekintsük az $S = \text{AZABBRAKADABRAA}$ szöveget. A gyakoriságok

| | | | | | |
|---|---|---|---|---|---|
| A | Z | B | R | K | D |
| 7 | 1 | 3 | 2 | 1 | 1 |

Program segítségével a prioritásos sor alapján azonnal építhetnénk a fát, azonban papíron számolva nem tudjuk előre, hogy a fa hogyan fog kinézni, így nem túl sok karakter esetén érdemes lehet előtte egy táblázatban az összevonásokat regisztrálni (2. ábra).



2. ábra. Huffman-kódoláshoz használt segédtábla és kódfa az S szöveghez.

A táblázat alapján könnyebben fel lehet rajzolni a fát (2. ábra), de a táblázat nélkül is ugyanazt a kódfát kapnánk.

A karakterek kódjai leolvashatóak a kódfából

| Char | Code |
|------|------|
| Z | 100 |
| R | 101 |
| K | 1101 |
| D | 1100 |
| B | 111 |
| A | 0 |

Az $S = \text{AZABBRADABRAA}$ kódolt alakja:

010001111111010110101100011110100.

Ez 33 bitet jelent és így az átlagos szóhossz 2.2. Ha a naiv módszert alkalmaznánk, akkor a kódszavak hossza minimálisan is 3 lenne és a szöveget így legalább 45 bittel ábrázolhatnánk.

Az entrópia (elméleti legkisebb átlagos szóhossz) értéke: $H_2(S) = 2.14646579850407 > 2.2 - 1$, így a kód optimálisnak nevezhető

1.2.3. Feladatok

1. A mintához hasonlóan végezzük el a Huffman-kódolást egy általunk választott tetszőleges szöveg esetén!

2. Keressünk olyan szöveget, ahol a Huffman-kódolás pontosan annyi bit-tel történik mint a naiv módszerrel!
3. Írjunk programot (struktogramot), amely a bemenetként kapott kódolt szöveg és hozzá tartozó Huffman-kód kód fája alapján helyreállítja az eredeti szöveget!
4. Írjuk meg a Huffman-kódolást megvalósító programot. Bemenet egy szöveg, kimenet a kódolt szöveg és a kód fája!
5. Számoljuk ki a ternáris ($r = 3$) Huffman-kódot a korábbi példára!
6. Verseny a csoportban: Keressük azt a 10 hosszú szöveget, amelyet a legnagyobb mértékben lehetett tömöríteni a minimális szóhosszt használó naiv módszerhez képest.

1.3. Szótárkódok – LZW (Lempel-Ziv-Welch)

A betűnkénti kódolás tömörítési tulajdonsága ismert és elmondható, hogy hatékonysága korlátozott is. Könnyen tudunk olyan adatot adni, amit sokkal tömörebben formában lehet reprezentálni, ha a kódolás nem karakterenként történik. Ezt az észrevételt használják ki a **szótárkódok** úgy, hogy egy kódszó nem csak egy karakter képe lehet, hanem egy szóé is.

Az LZW kódolás egy kezdeti kódtáblát bővít lépésről-lépésre úgy, hogy egyre hosszabb már „látott” szavakhoz rendel új kódszót. Ezzel a valós adatok azt a tulajdonságát használjuk ki, hogy abban relatív rövid részek sűrűn ismétlődnek. Például gondoljunk élő nyelvben milyen sűrűn fordulnak elő névelők, kötőszavak, stb.

Shannon tétele viszont itt is igaz marad és továbbra is korlátozza a lehetséges tömörítés mértékét, de itt r növekszik.

Az r a kódszavak későbbi számától függ és növekedésével csökken az entrópia és a tömöríthetőség alsó korlátja is.

1.3.1. Szemléltetés és működés

Legyen $S = \text{ABABABAACAACCBBA}\text{AAAAAAAAA}$ a példaszövegünk és így a kezdeti kódtáblánk (a szöveg karakterei alapján):

| Char | Code |
|------|------|
| A | 1 |
| B | 2 |
| C | 3 |

A bemenetet pontosan egyszer fogjuk olvasni a kódolás során úgy, hogy mindig a már ismert (kóddal rendelkező) leghosszabb következő szót keressük. Ha megtaláltuk, akkor egyrészt

- kiírjuk a talált szó kódját a kimenetre (eredmény), másrészt
- bővítjük a kódszavak halmazát az αc szó képével, ahol az α a talált szó és c a következő karakter.

Például az S esetén kezdetben a leghosszabb „ismert” szó az **A**, ennek kódja 1 és az új kóddal rendelkező szó **AB** a 4 kóddal. A kódolás folyamán ezt az lépést ismétljük, amíg a szöveg végére nem érünk.

Egy szemléltetése a teljes kódolásnak a következő táblázat, ahol az egyes lépések soronként szerepelnek, a kimenetet a Code oszlop tartalmazza soronként és az új kód mindig a megfelelő sor második és harmadik oszlopban szereplő értékek konkatenációjához tartozik.

| Code | Actual word | Next char | New code |
|------|-------------|-----------|----------|
| 1 | A | B | 4 |
| 2 | B | A | 5 |
| 4 | AB | A | 6 |
| 6 | ABA | A | 7 |
| 1 | A | C | 8 |
| 3 | C | A | 9 |
| 1 | A | A | 10 |
| 8 | AC | C | 11 |
| 3 | C | B | 12 |
| 2 | B | B | 13 |
| 5 | BA | A | 14 |
| 10 | AA | A | 15 |
| 15 | AAA | A | 16 |
| 15 | AAA | - | - |

A kódolt üzenet tehát

[1, 2, 4, 6, 1, 3, 1, 8, 3, 2, 5, 10, 15, 15]

A dekódoláshoz ugyanazt a kezdeti, csak a karakterek kódját tartalmazó kódtáblát használjuk és szemléltethetjük ugyanazzal a táblázattal. Az első és utolsó oszlopot teljes egészében ki tudjuk tölteni, majd soronként haladunk. Jelen esetben az első két sorhoz tartozó aktuális szó nyilván ismert.

| Code | Actual word | Next char | New code |
|------|-------------|-----------|----------|
| 1 | A | ? | 4 |
| 2 | B | ? | 5 |
| 4 | ? | ? | 6 |
| 6 | ? | ? | 7 |
| ⋮ | ⋮ | ⋮ | ⋮ |

Mivel a második sor szavának első karaktere éppen az első sorban szereplő következő karakter, így az ismert.

| Code | Actual word | Next char | New code |
|------|-------------|-----------|----------|
| 1 | A | B | 4 |
| 2 | B | ? | 5 |
| 4 | ? | ? | 6 |
| 6 | ? | ? | 7 |
| : | : | : | : |

Most már ismerjük mit kódolt az első sor új kódja, így folytathatjuk a kitöltést:

| Code | Actual word | Next char | New code |
|------|-------------|-----------|----------|
| 1 | A | B | 4 |
| 2 | B | A | 5 |
| 4 | AB | ? | 6 |
| 6 | ? | ? | 7 |
| : | : | : | : |

Itt viszont látszólag elakadunk. Eddig mindig ismert volt a kódszóhoz tartozó szó mielőtt használni szeretnénk volna. Azonban most azt a kódszót szeretnénk használni, aminek visszaállításához szükség lenne annak inverz kékére. Nyilván amíg nem ismert a kódhoz tartozó szó addig nem is használhatjuk. Szerencsére csak az első karakterére van szükség, ami ismert.

| Code | Actual word | Next char | New code |
|------|-------------|-----------|----------|
| 1 | A | B | 4 |
| 2 | B | A | 5 |
| 4 | AB | A | 6 |
| 6 | ? | ? | 7 |
| : | : | : | : |

A dekódolás ez alapján már egyszerűen befejezhető, a táblázat meg fog egyezni a kódolásnál már bemutatottal és a dekódolt szöveg kiolvasható a második oszlopból.

1.3.2. Megjegyzések

1. Fontos észrevenni, hogy egy hosszú szöveg esetén az ismertetett eljárás annyi új kódszót is bevezethet, hogy az azok közötti keresés összemérhető lenne a teljes szöveg végigolvasásával. Természetesen ezt nem

szeretnénk, ezért gyakorlatban korlátozzuk a kódszavak halmazát. Ez történhet például

- a kódszavak számának korlátozásával;
 - a kódszavakhoz tartozó szavak hosszának korlátozásával;
 - azzal, hogy a bemenet csak egy kezdőszeletén építjük a szótárat, utána csak kódolunk.
2. Mivel a Huffman-kódolás csak a betűnkénti kódolások között optimális az LZW eljárás könnyen eredményezhet rövidebb kódolt alakot, annak ellenére is, hogy az itt használt kódszavakat még binárisan kódolni kell.
 3. Az LZW eljárás egyszerűnek nevezhető (összehasonlítva például a Huffman kódolással) és mivel csak egyszer kell olvasni a bemenetet, hatékony is (amennyiben a kódszavak tárolása hatékony).

1.3.3. Feladatok

1. Szemléltessük a ABCABCABCAAABBCCAABABA kódolását.
2. Dekódoljuk az 1,3,4,5,6 üzenetet, ha a kezdeti tábla $A \rightarrow 1$ és $B \rightarrow 2$.
3. Adjunk meg olyan 20 hosszú három karaktert tartalmazó (legalább egyszer mindegyiket) szöveget, ami esetén a kódolt (LZW) szöveg pontosan 13 hosszú.
4. Keressünk olyan szöveget, aminek LZW kódolása rövidebb eredményt ad a Huffman-kódolással összehasonlítva.