Tömörítés

Alapfogalmak

- **Ábécé**: $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_d\}$
- **Ábécé mérete**: d
- **Tömörítendő szöveg hossza**: n
- **Kódszó hossza**: $L = \lceil \log_2 d \rceil$ (kettes alapon vett logaritmus, felfelé kerekítve)
- **Kódtáblázat**: Az ábécé karaktereihez hozzárendelt, fix hosszúságú bináris kódokat tartalmazza
- **Kódfa:**
    - levelei az ábécé karaktereihez tartoznak
    - minden élhez egy bináris címkét rendelünk: bal él → 0, jobb él → 1
    - egy levélhez vezető úton az élek címkéit összeolvasva kapjuk meg az adott karakter kódszavát
    - a kódfában nincs olyan kódszó, ami egy másik kódszó előtagja (prefixmentes kód)

Naiv módszer

A naiv egy olyan veszteségmentes tömörítési eljárás, amelyben minden karaktert azonos hosszúságú, fix bitsorozattal kódolunk. A kódhossz a karakterkészlet méretének függvényében a legkisebb olyan egész szám, amelyen minden karakter külön bináris kódot kaphat.

Tömörítési eljárás

1. Meghatározzuk az ábécé elemeit: Σ = { … }
2. Meghatározzuk az ábécé méretét: d = …
3. Meghatározzuk a tömörítendő szöveg hosszát: n = …
4. Meghatározzuk kódszó hosszát: L = $\lceil \log_2 d \rceil$
5. Minden karakterhez egyedi, L bites kódszót rendelünk
6. A bemeneti szöveget karakterenként helyettesítjük a kódszavával
7. A kimeneti fájl tartalmazza:
    a. a kódtáblázatot
    b. a tömörített kódsorozatot

Kitömörítés

Minden L-bites szakaszhoz a kódtáblázat alapján hozzárendeljük a megfelelő karaktert

Példa 1

- Tömörítendő szöveg: ABRAKADABRA

1. Σ = {A, B, R, K, D}
2. d = 5
3. n = 11
4. L = $\lceil \log_2 5 \rceil = 3$
5.

Karakter	Kód
A	000
B	001
D	010
K	011
R	100

6. 000 001 100 000 011 000 010 000 001 100 000
   A B R A K A D A B R A

- Tömörített szöveg mérete: n * L = 11 · 3 = 33 bit
- Kódtábla mérete: d * 8 + d * 3 = 5 * 8 + 5 * 3 = 40 + 15 = 55
- Teljes tömörített méret: 33 + 55 = 88 bit *(plusz meta adatok)*
- Eredeti méret: 11 * 8 = 88 bit *(plusz meta adatok)*

Példa 2

- Tömörítendő szöveg: ABRAKADABRAABRAKADABRAABRAKADABRA

1. Σ = {A, B, R, K, D}
2. d = 5
3. n = 33
4. L = $\lceil \log_2 5 \rceil = 3$
5.

Karakter	Kód
A	000
B	001
D	010
K	011
R	100

- Tömörített szöveg mérete: n * L = 33 · 3 = 99 bit
- Kódtábla mérete: d * 8 + d * 3 = 5 * 8 + 5 * 3 = 40 + 15 = 55
- Teljes tömörített méret: 99 + 55 = 154 bit *(plusz meta adatok)*
- Eredeti méret: 33 * 8 = 264 bit *(plusz meta adatok)*

Előnye

- Egyszerű
- Az eredeti adat teljesen visszaállítható, semmilyen információ nem vész el
- Minden kódszó fix hosszúságú → a bitsorozatot egyszerűen szakaszokra lehet bontani
- Nem szükséges bonyolult fa vagy keresés (ellentétben pl. a Huffman-kóddal)

Hátránya

- A gyakori karakterek ugyanannyi bitet kapnak, mint a ritkán előfordulók, ezért a tömörítési arány sokkal rosszabb, mint változó hosszúságú kódoknál (pl. Huffman)
- A fájlban a kódtáblát is el kell menteni, ami rövid szövegeknél nagy többletet jelenthet.
- Ha az ábécé túl nagy (d nagy), akkor a szükséges kódszóhossz ($\lceil \log_2 d \rceil$) is megnő, és a tömörítés értelme elvész
- Egy 90%-ban „A”-ból álló szöveg ugyanannyiba kerül, mint egy teljesen vegyes szöveg (ugyanakkora ábécé mellett)

Huffman-kód

A Huffman-kódolás egy olyan veszteségmentes tömörítési eljárás, amelyben a gyakrabban előforduló karakterek rövidebb, a ritkábbak hosszabb kódszót kapnak. A kódfát a karakterek gyakorisága alapján építjük, így a teljes kódolt üzenet hossza optimálisan minimális lesz a prefixmentes kódok között.

Tömörítési eljárás

1. Határozzuk meg a karakterekhez tartozó gyakoriságokat
2. Hozzunk létre minden karakterből egy fát a gyakoriság, mint kulcs segítségével (tehát itt lesz sok-sok 1 csúcsból álló fánk)
3. Tegyük be az összes így kapott fát, egy min-prioritásos sorba (tehát itt a kis fák egy csúcsként fognak viselkedni)
4. Vegyünk ki két csúcsot (fát) a sorból
5. Készítsünk egy szülőt a két fa gyökércsúcsának, ahol a kulcs a két fa gyökércsúcsában lévő kulcs összege lesz, továbbá a bal élt címkézzük „0”-val, a jobb élt pedig „1”-el
6. Az így 2 fából és egy harmadik csúcsból alkotott fát tegyük vissza a min-prioritásos sorba
7. Ismételjük meg az egészet a 4. lépéstől, ameddig a min-prioritásos sorban legalább 2 elem van
8. Ha szerencsénk van, itt már csak egy fa szerepel a sorban, ezt nevezzük kódfának
9. A kódfában balra vagy jobbra egészen a levélig haladva ki tudjuk olvasni az adott karakterhez (ami ugye a levélben van) tartozó kódszót az élek címkéiről
10. Építsük fel a kódtáblázatot a 9. pont alapján
11. A kimeneti fájl tartalmazza:
    a. a kódtáblázatot
    b. a tömörített kódsorozatot

Kitömörítés

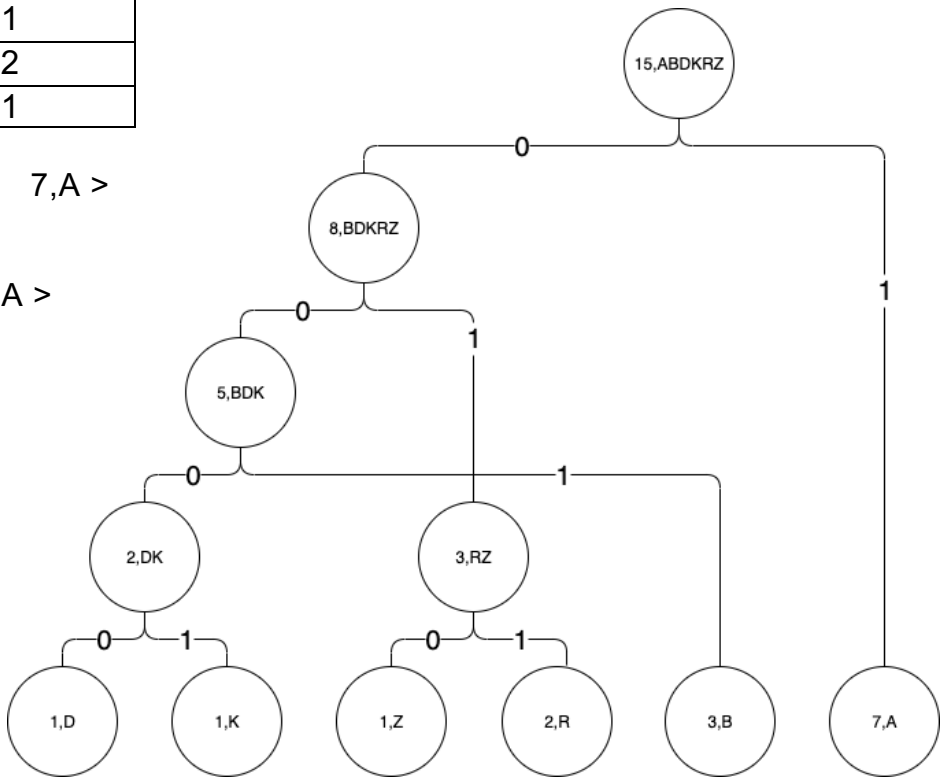
A tömörített szöveget elkezdjük olvasni, és ha találunk olyan sorozatot ami a kódtáblában szerepel, akkor az a sorozat a kódtáblában lévő karakternek fog megfelelni

Példa 1

- Tömörítendő szöveg: AZABBRAKADABRAA
- Σ = {A, Z, B, R, K, D}

Karakter	Előfordulás
A	7
B	3
D	1
K	1
R	2
Z	1

1. < 1,D 1,K 1,Z 2,R 3,B 7,A >
2. < 1,Z 2,DK 2,R 3,B 7,A >
3. < 2,DK 3,B 3RZ 7,A >
4. < 3RZ 5,BDK 7,A >
5. < 7,A 8,BDKRZ >
6. < 15,ABDKRZ >



Karakter	Kód	Előfordulás
A	1	7
B	001	3
D	0000	1
K	0001	1
R	011	2
Z	010	1

- 1 010 1 001 001 011 1 0001 1 0000 1 001 011 1 1
  A Z A B B R A K A D A B R A A

- Tömörített szöveg mérete: 7*1 + 3*3 + 1*4 + 1*4 + 2*3 + 1*3 = 33 bit
- Kódtábla mérete: 6*8 + 1 + 3 + 4 + 4 + 3 + 3 = 66 bit
- Teljes tömörített méret: 33 + 66 = 99 bit *(plusz meta adatok)*
- Eredeti méret: 15 * 8 = 120 bit *(plusz meta adatok)*

Példa 2

- Tömörítendő szöveg:
  AZABBRAKADABRAAAZABBRAKADABRAAAZABBRAKADABRAA
- Σ = {A, Z, B, R, K, D}

Karakter	Előfordulás
A	21
B	9
D	3
K	3
R	6
Z	3

7. < 3,D 3,K 3,Z 6,R 9,B 21,A >

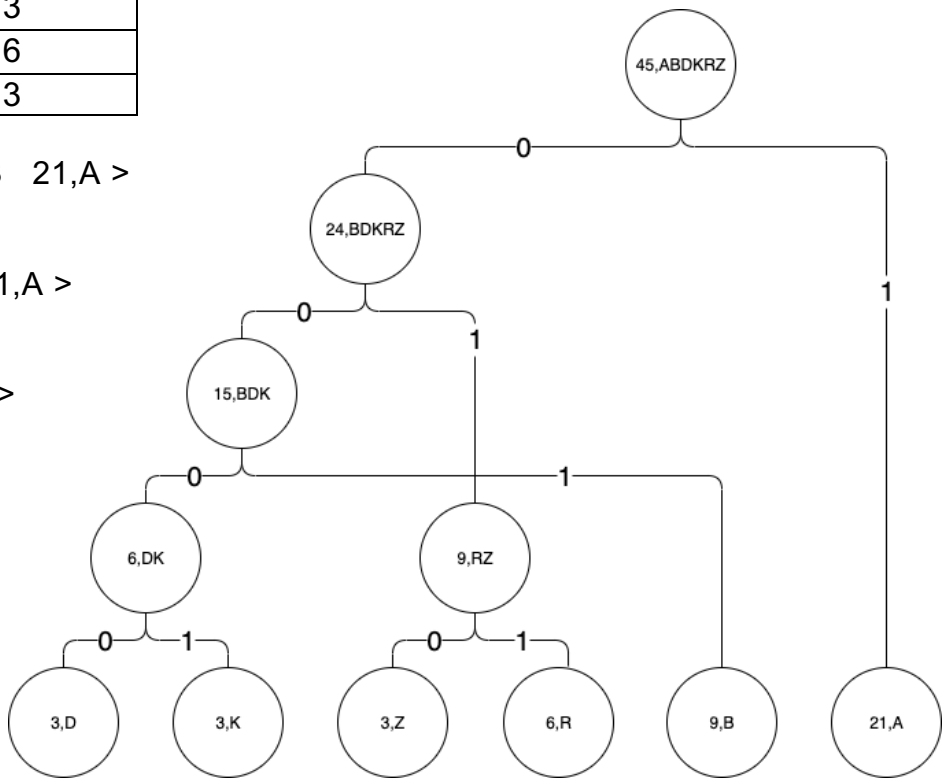
8. < 3,Z 6,DK 6,R 9,B 21,A >

9. < 6,DK 9,B 9,RZ 21,A >

10. < 9,RZ 15,BDK 21,A >

11. < 21,A 24,BDKRZ >

12. < 45,ABDKRZ >



Karakter	Kód	Előfordulás
A	1	21
B	001	9
D	0000	3
K	0001	3
R	011	6
Z	010	3

- Tömörített szöveg mérete: 21*1 + 9*3 + 3*4 + 3*4 + 6*3 + 3*3 = 99 bit
- Kódtábla mérete: 6*8 + 1 + 3 + 4 + 4 + 3 + 3 = 66 bit
- Teljes tömörített méret: 99 + 66 = 165 bit *(plusz meta adatok)*
- Eredeti méret: 45 * 8 = 360 bit *(plusz meta adatok)*

Előnye

- Mindig a lehető legrövidebb átlagos kódhosszt adja meg a karaktergyakoriságok alapján
- Az eredeti adat teljesen visszaállítható, semmilyen információ nem vész el
- Minél nagyobb az eltérés a gyakori és ritka karakterek előfordulásában, annál hatékonyabb

Hátránya

- A dekódoláshoz ismerni kell a kódfát vagy a kódtáblát, ami plusz helyet foglal
- A kódszavak változó hosszúságúak, ezért nem lehet egyszerűen szakaszokra vágni a bitsorozatot

Szorgalmi feladatok

1. Tömörítsd a Naiv módszerrel az alábbi szöveget (1 pont)
   - HUMBABUMBLAKUMPALUMPABUUUUU

   Továbbá add meg a
   - Kódtáblát
   - Kódtábla méretét
   - Tömörített szöveg méretét
   - A szöveg eredeti méretét

2. Tömörítsd a Huffman módszerrel az alábbi szöveget (1 pont)
   - HUMBABUMBLAKUMPALUMPABUUUUU

   Továbbá add meg a
   - Kódfát
   - Kódtáblát
   - Kódtábla méretét
   - Tömörített szöveg méretét
   - A szöveg eredeti méretét