# Algoritmusok és adatszerkezetek I.
# 11. Előadás

## Veszteségmentes adattömörítés.

# Tartalom

# Naiv módszer

- A tömörítendő szöveget karakterenként, $x$ hosszúságú bitsorozatokkal kódoljuk.
- $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_d \rangle$ az ábécé.
- Egy-egy karakter $\lceil \lg d \rceil$ bittel kódolható
  - $\lceil \lg d \rceil$ biten $2^{\lceil \lg d \rceil}$ különböző bináris kód ábrázolható
  - $2^{\lceil \lg d \rceil} \geq d > 2^{\lceil \lg d \rceil - 1}$ => $\lceil \lg d \rceil$ biten ábrázolható $d$-féle különböző kód, de eggyel kevesebb biten már nem.
- $In$: $\Sigma\langle\rangle$ a tömörítendő szöveg. $n = |In|$ jelöléssel $n * \lceil \lg d \rceil$ bittel kódolható
- A tömörített fájl a kódtáblázatot is tartalmazza
- Kitömörítés:
  - A kódtáblázat alapján $\lceil \lg d \rceil$ bites szakaszokra bontható
- A kódtáblázat mérete miatt a gyakorlatban csak hosszabb szövegeket érdemes így tömöríteni

# Példa:

- ABRAKADABRA szöveg
  - $d = 5$ és $n = 11$
  - ➢ a tömörített kód hossza $11 * \lceil \lg 5 \rceil = 11 * 3 = 33$ bit.
  - (A 3-bites kódok közül tetszőleges 5 kioszthető az 5 betűnek)
  - A fenti ABRAKADABRA szöveg kódtáblázata lehet pl. a következő:
  - A fenti kódtáblázattal a tömörített kód a következő lesz:
    - 000001100000011000010000001100000
    - $\lceil \lg d \rceil$ Ez a tömörített fájlba foglalt kódtáblázat alapján könnyedén 3 bites szakaszokra bontható és kitömöríthető

| karakter | kód |
|---|---|
| $A$ | 000 |
| $B$ | 001 |
| $D$ | 010 |
| $K$ | 011 |
| $R$ | 100 |

# Huffman-kód

- Karakterenkénti kódoló
  - Huffman-kód hossza minimális
- Változó hosszúságú bitsorozatok
  - A gyakrabban előforduló karakterek kódja rövidebb
  - A ritkábban előfordulóké hosszabb
- **Prefix-mentes kód:** Egyetlen karakter kódja sem prefixe semelyik másik karakter kódjának sem
- A szöveghez többféle kódfa és hozzátartozó kódtáblázat építhető
  - Mindegyik segítségével az input szövegnek ugyanolyan hosszú a tömörített kódja
- Betömörítés:
  - a kódtáblával
- Kitömörítés
  - a kódfával

➢ A tömörített fájl a kódfát is tartalmazza

# Huffman-kód

- A tömörítendő fájlt, illetve szöveget kétszer olvassa végig
  1. Olvasásnál:
     - Meghatározza a szövegben előforduló karakterek halmazát
     - az egyes karakterek gyakoriságát
     - majd ennek alapján kódfát
     - abból pedig kódtáblázatot épít
  2. Olvasásnál:
     - A kódtábla alapján kiírja az output fájlba sorban a karakterek bináris kódját
- **Kódfa:** szigorúan bináris fa
  - Levél: az ábécé karakterei és annak gyakoriságai (előfordulásainak száma)
  - A belső csúcsok: a csúcshoz tartozó részfa leveleit címkéző karakterek gyakoriságainak összegével címkézzük.
  - ➢ A kódfa gyökere: a tömörítendő szöveg hossza

# A kódfa felépítése:

- Kiindulás: egy csúcsú fák – egy csúcs és annak gyakorisága
- Minimum prioritásos sor: a fák gyökerét címkéző gyakoriságértékek szerint
- Ciklus, amíg a kupac még legalább kettő fából áll:
  - Kiveszünk a kupacból egy olyan fát, amelyeknek gyökerét a legkisebb gyakoriság címkézi
  - Ezután a maradék kupacra ezt még egyszer megismételjük
  - Összeadjuk a két gyakoriságot
  - Az összeggel címkézünk egy új csúcsot, amelynek bal és jobb részfája az előbb kiválasztott két fa lesz
    - A bal ágat a 0, a jobb ágat az 1 címkézi
  - Az így képzett új fát visszatesszük a minimumprioritásos sorba
- A minimum-prioritásos sorban maradó egyetlen bináris fa: a Huffman-féle kódfa

# Kódtáblázat, kódolás

- A kódfából -> kódtáblázat
- Egy karakter kódja:
  - a kódfa gyökerétől elindulva és a karakterhez tartozó levélig lefelé haladva a kódfa éleit címkéző biteket összeolvassuk
  - (pl. a kódfa preorder bejárásával, az aktuális csúcshoz vezető bitsorozat folyamatos nyilvántartásával, és levélhez érve, a kódtáblázatba írásával.)
- Kódolás:
  - a tömörítendő szöveg 2. végigolvasása
  - a kódtáblázat segítségével sorban mindegyik karakter bináris kódját a (kezdetben üres) tömörített bitsorozat végéhez fűzzük
- A tömörített fájl a kódfát is tartalmazza

➢ A gyakorlatban Huffman-kódolással is csak hosszabb szövegeket érdemes tömöríteni

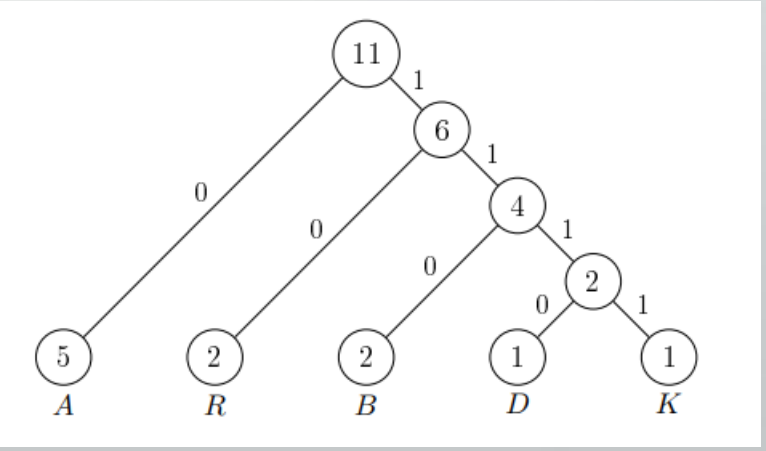
# Kitömörítés

- Karakterenként
  - Mindegyik karakter kinyeréséhez a kódfa gyökerétől indulunk
  - majd a tömörített kód sorban olvasott bitjeinek hatására
    - 0: balra lépünk
    - 1: jobbra lépünk lefelé a fában
  - Amíg levélcsúcshoz érünk
  - Ekkor kiírjuk a levelet címkéző karaktert
  - Folytatjuk az eljárást a következő bittől és újra a kódfa gyökerétől folytatjuk
  - amíg a tömörített kódon végig nem érünk.

# Huffman-kódolás szemléltetése

| szöveg: | A | B | R | A | K | A | D | A | B | R | A |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $A$ | 1 | | | 2 | | 3 | | 4 | | | 5 |
| $B$ | - | 1 | | | | | | | 2 | | |
| $D$ | - | - | - | - | - | - | 1 | | | | |
| $K$ | - | - | - | - | 1 | | | | | | |
| $R$ | - | - | 1 | | | | | | | 2 | |

- A szöveg: ABRAKADABRA
- Gyakoriságtáblázat:
- Prioritásos sor: (5 egycsúcsú fa): (levél/gyakoriság)
  - ⟨(D/1),(K/1), [B/2], (R/2),(A/5)⟩ => levél és gyökér is
- (Azonos gyakoriságok esetén a betűk alfabetikus sorrendje szerint rendezünk. )
- Fa építés:
  - Kivesszük a két legkisebb gyakoriság-értékű fát
  - Egy új gyökércsúcs alá tesszük őket bal- és jobboldali részfának
  - Az új gyökércsúcs: a két fa-gyakoriság-érték összegével címkézzük
  - Visszatesszük az új fát a minimum-prioritásos sorba.



# Huffman-kódolás szemléltetése

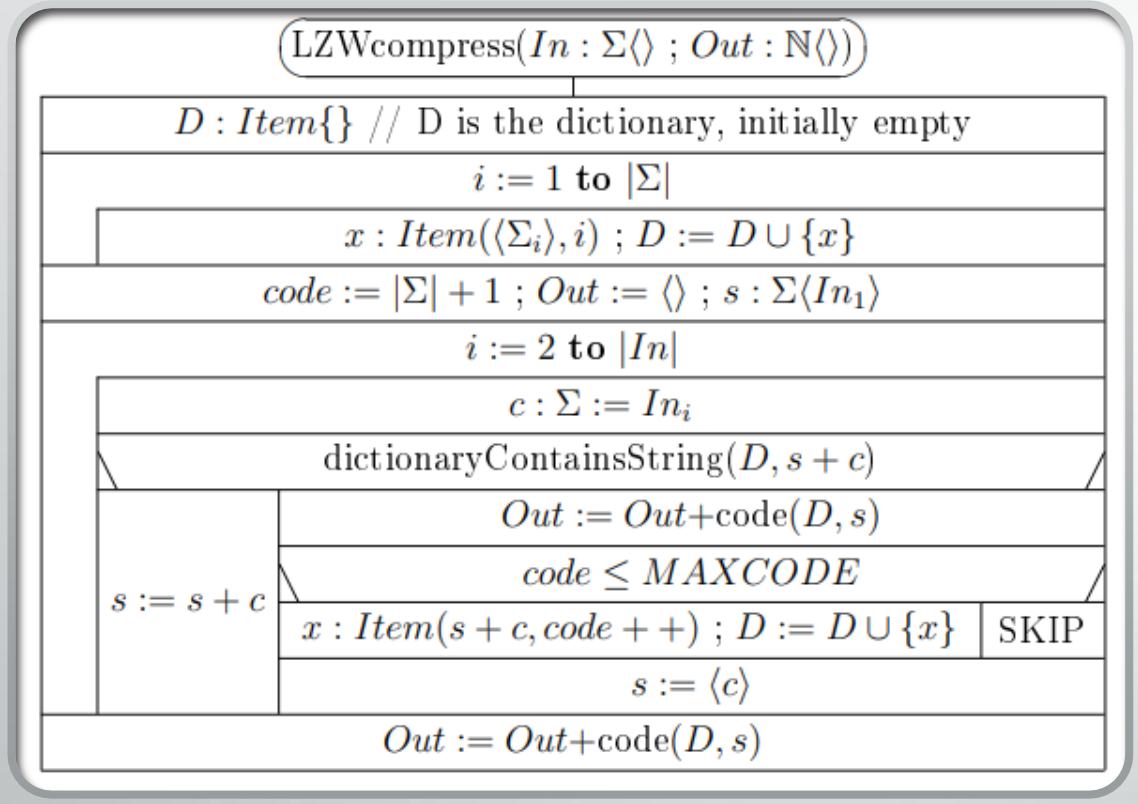
- Kódtáblázat:
  - Az út éleit címkéző biteket összeolvasva adódik a levelet címkéző karakter Huffman-kódja
- Szöveg Huffman kódja:
  - 011010011111011100110100
  - 23 bit
- Kitömörítés:
  - Huffman-kód és a kódfa alapján:
    - 0=>A
    - 110 => B
    - 10 => R
    - 0 => A
    - 1111 =>K
    - 0 => A
    - 1110 =>D
    - 0 => A
    - 110 => B
    - 10 => R
    - 0 => A

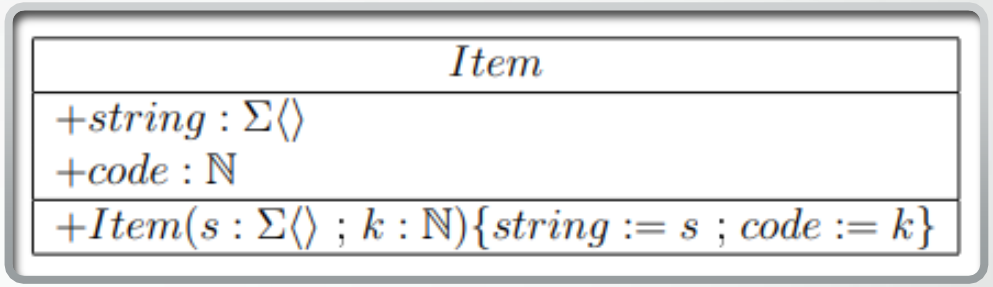
| karakter | kód |
|---|---|
| $A$ | 0 |
| $B$ | 110 |
| $D$ | 1110 |
| $K$ | 1111 |
| $R$ | 10 |

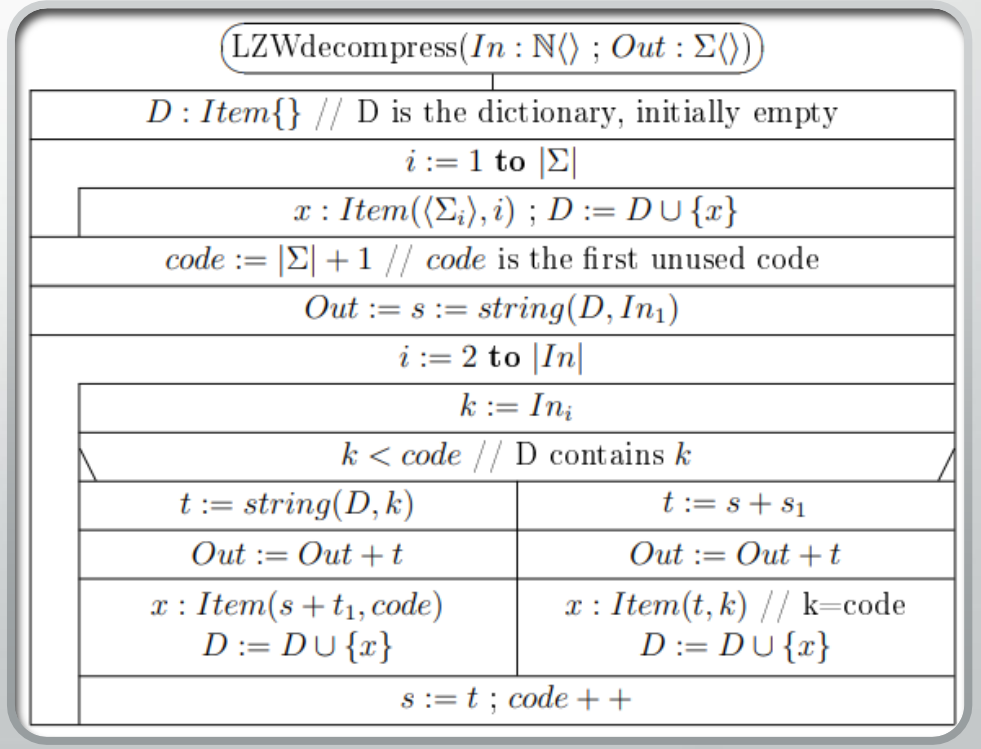
# Lempel-Ziv-Welch (LZW) módszer

- Az input szöveget ismétlődő mintákra (sztringekre) bontja
- Mindegyik mintát ugyanolyan hosszú bináris kóddal helyettesíti
- A tömörített fájl a kódtáblázatot nem tartalmazza
  - A kitömörítés rekonstruálja az ábécé és a tömörített kód alapján
- Jelölések az absztrakt struktogramokhoz:
  - Ha a kódok $b$ bitesek => MAXCODE = $2^b-1$ globális konstans a kódként használható legnagyobb számérték
  - Pl. $b = 12$ => MAXCODE = $2^{12} - 1 = 4095$
  - $\Sigma = <\sigma_1, \sigma_2, \ldots, \sigma_d>$ sorozat tartalmazza az ábécé karaktereit $(d \in \mathbb{N} \wedge d > 0)$
  - Tömörítésnél:
    - *In*: a tömörítendő szöveg
    - *Out*: a tömörítés eredménye: kódok sorozata
  - Kitömörítésnél:
    - *In – Out* fordítva
    - *D*: szótár, ami (string, code) rendezett párok, azaz Item-ek halmaza

# LZW algoritmus stuktogramja



# Ellenőrző kérdések: Huffman kód

1. Szemléltesse a Huffman kódolás működését az ÁBRÁBANÁBRA szövegen!
   - Adja meg a kódfát és a szótárat! Mekkora a Huffman kódolással tömörített kód hossza?
   - Mekkora lenne a tömörített kód x hosszú karakterkódok esetén?
   - Hogyan dekódolható egy Human kóddal tömörített szöveg?
   - Milyen értelemben optimális a Human kód? Azt jelenti-e ez, hogy a Huffman kódolás a lehető legjobb tömörítés? Miért?

# Ellenőrző kérdések: Lempel-Ziv-Welch (LZW)

1. Adott az {A;B;C} ábécé. Szemléltesse az LZW tömörítő algoritmus
   - Tömörítő algoritmus  működését a CBABABABCBABABAB szövegen
   - Majd a megfelelő kitömörítő algoritmusét az 1; 2; 3; 4; 6; 5; 9; 7; 11 kódon!
   - Mindkét esetben adja meg
     - A generált szótárat és
     - A tömörítetlen szövegen a részszavak és a kódok megfeleltetését!
   - Hogyan kezeli a kitömörítő algoritmus azt az esetet, amikor nem találja meg a szótárban az aktuális kódhoz tartozó sztringet?

# Köszönöm a figyelmet!

**Pusztai Kinga**

A bemutató Ásványi Tibor: Algoritmusok és adatszerkezetek II. eladásjegyzet:Mintaillesztés, tömörítés alapján készült.