Algoritmusok és adatszerkeztek II.

1. gyakorlat

Az anyag készítésénél felhasználásra került Nagy Ádám által készített segédlet.

Alap információk

- Minden fontos infó megtalálható a Canvas tematika felületén. (+elérhetőségek)

Tartalom:
Veszteség mentes adattömörrítés

Kódolás elmélet

- Informatikában a **kódolás elmélet** 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 **adat tömörítésről**.

Veszteségmentes tömörítés

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

Információ alapegysége

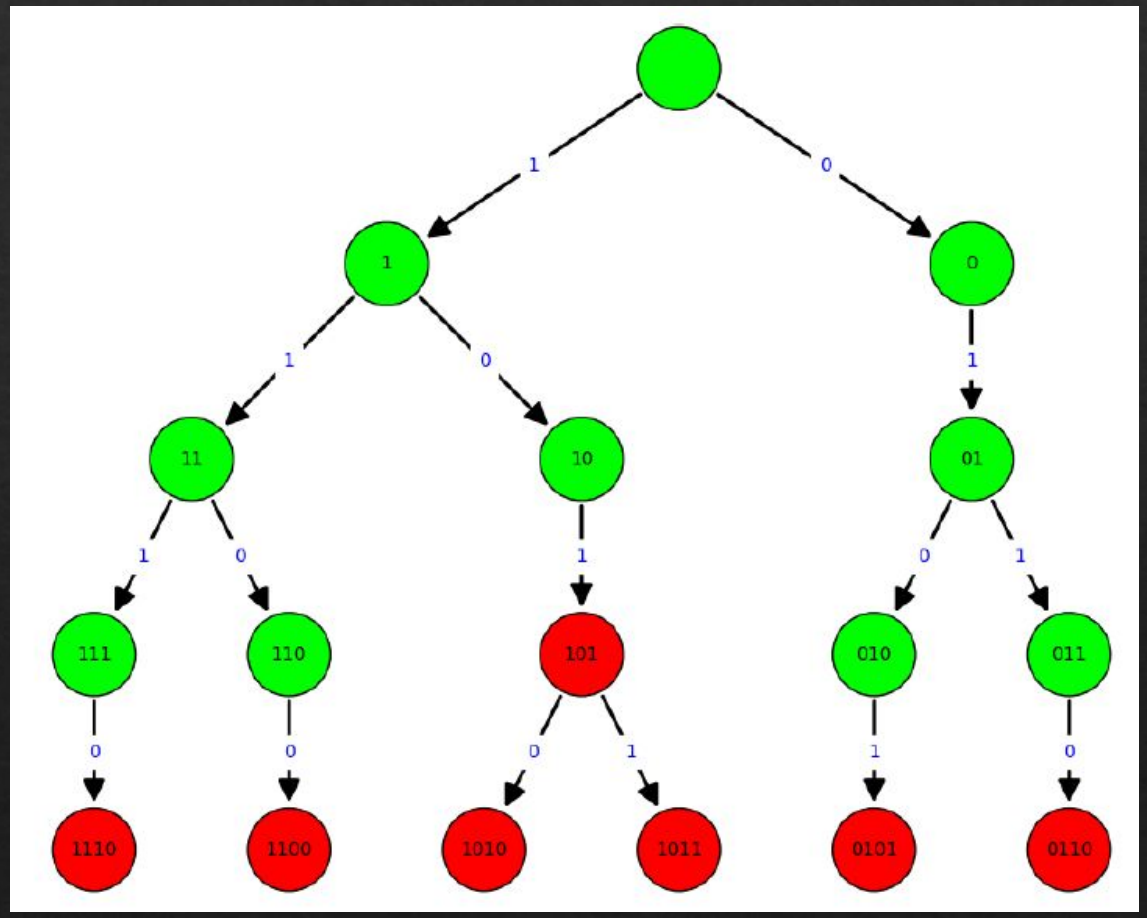
- A kódoláselméletnél meg kell adnunk az **információ alapegységét,** azaz azt, mennyi információtartalma van az atomi „tárolási egységnek".
- Mivel a jelenlegi számítógépek bináris elven működnek, ez *r=2* és így a kódszavaink a Γ={'0','1'} ábécé feletti szavak lesznek.

Kód, kódfa

- **Kódnak** nevezzük a Γ ábécé feletti véges szavak (kódszavak) egy tetszőleges nem üres halmazát.
- Például egy bináris kód a kövtkező halmaz:
  C = {'1011', '1100', '0110', '1110', '1010', '0101', '101’}
- 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.

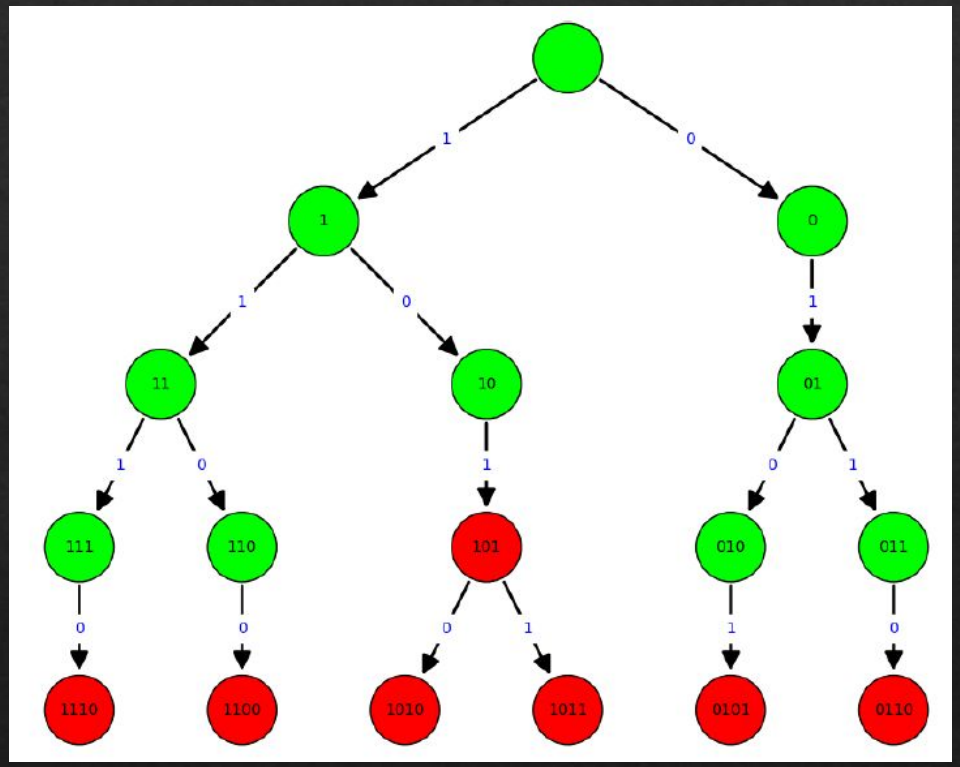
Kódfa

- C = {'1011', '1100', '0110', '1110', '1010', '0101', '101’} kód kódfája.
- Kódszavak pirossal vannak jelölve.



Kódfa

- A kódfa a 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 a dekódolt adatot.
- Természetesen ez csak akkor igaz, ha a dekódolás egyáltalán lehetséges és egyértelmű.



Betűnkénti kódolás

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

ASCII kódtábla: 7biten 128 karakter kódolása

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Egyenletes kód, **naiv módszer**

- Egy kódot **egyenletes kódnak** nevezünk, 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.
- Gyakorlatban, ha a tömörítés nem igazán fontos szempont, egyszerűsége miatt sok helyen alkalmazzák, például a 8 bit hosszúságú kódszavakat használó ASCII kód is ilyen.

Naiv módszer példa

- Tekintsük a S = AABCAADEAAB szöveget.
- Σ={A,B,C,D,E} $|\Sigma|$=5 Γ={0,1} $|\Gamma|$=2
- $\lceil \log_2 5 \rceil = 3$, azaz 3 bites lesz a kód.
- Lehetséges kódtábla:
  A=000 B=001 C=010 D=011 E=100
- Kódolt szöveg:
  000 000 001 010 000 000 011 100 000 000 001
- Kódolt szöveg hossza: 11*3 = 33 bit

Huffman kód

- Ezt az algoritmust David A. Huffman (1925–1999) írta le először egy mesteri vizsgadolgozatban, és 1952-ben publikálta.
- Intuitíven, betűnkénti kódolás esetén akkor kapunk rövidebb kódolt adatot, ha a gyakori betűkhöz rövid kódszót, a ritkákhoz pedig hoszabbakat rendelünk.
- A **Huffman-kódolás** egy betűnkénti optimális kódolás, azaz az ilyen kódolások között szinte a legjobb tömörítés érhető el vele adott adat esetén.
- A kódolás a Huffman kódfa alapján történik.
- A kódfa felépítése előtt meg kell határozni az ábécé betűinek gyakoriságát a tömörítendő szövegben.
- A kódhoz tartozó kódfát alulról felfelé építjük az eredeti szöveg karaktereinek gyakorisága alapján.

Huffman tömörítési algoritmus mente (betömörítés)

- Bináris r=2 esetben a következőképpen járunk el:

1. Olvassuk végig a szöveget és határozzuk meg az egyes karakterekhez tartozó gyakoriságokat.
2. Építsük fel a kódfát: hozzunk létre minden karakterhez egy csúcsot és helyezzük el azokat egy (minimum) prioritásos sorban a gyakoriság mint kulcs segítségével.
3. Vegyünk ki két csúcsot a prioritásos sorból és hozzunk létre számukra egy szülő csúcsot.
4. Helyezzük el a szülő csúcsot a prioritásos sorba gyerekei gyakoriságának összegét használva kulcsként.
5. Ismételjük az algoritmust a 3. ponttól, ha egynél több csúcs van a prioritásos sorban.
6. Címkézzük fel szisztematikusan a kapott kódfa éleit ,’0’ és ,’1’ címkével, például a bal gyerek legyen mindig ’0’, a jobb gyerek pedig ,’1’.
7. Olvassuk ki a karakterekhez tartozó kódszavakat a kódfából. (Gyökérből indulva, a levélig tartó út címkéi – szokták a levél szelektorának is nevezni.)
8. Olvassuk végig újra a bemenetet és kódoljuk azt karakterenként.

Huffman algoritmussal kódolt szöveg dekódolása (kitömörítés)

- A dekódoláshoz szükségünk van a kódfára, ezt a kódolt szöveg mellé szokták helyezni.
- A dekódolásnál a kódolt szöveget bitenként dolgozzuk fel az alábbi lépéseket követve:

1. Álljunk a kódfa gyökerébe.
2. Az olvasott bitek szerint lépegessünk a kódfában balra illetve jobbra, amíg levélhez nem érünk.
3. Levélhez érve olvassuk ki, milyen betűt tartalmaz a levél, és írjuk ki a kimenetre.
4. Ismételjük 1-től, amíg el nem fogynak a kódolt szöveg bitjei.

Megjegyzések a Huffman kódhoz

- 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 (több fa azonos értékű lehet). Egyrészt ha több azonos gyakoriság van, akkor bármelyiket választva Huffman-kódolást kapunk, másrészt a ’0’ és ’1’ szerepe felcserélhető.
- A betömörítés 2 menetes, azaz kétszer olvassa végig a bemenetet, első menetben az ábécé betűinek gyakoriságát határozza meg, majd a kódfa felépítése után még egyszer végig olvassa a szöveget, és előállítja a kódot. Így a gyakorlatban (nagy bemenetre) lassabban működik, mint az egy menetes LZW algoritmus.

Megjegyzések a Huffman kódhoz

- Mivel a kódszavak hossza különböző, a kódnak a dekódoló oldalon is ismertnek kell lennie. Ez gyakorlatban azt jelenti, hogy:
- a kódfát 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 (általában tömörít, de nem az optimális kódot kapjuk).
- Figyelem! A dekódoláshoz használhatnánk a betűk kódjait tartalmazó kódtáblát is, de az abban való keresgélés sokkal időigényesebb lenne, mint a kódfa alapján történő dekódolás, így nem ezt használják a gyakorlatban.

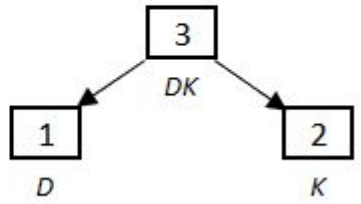
Példa a Huffman algoritmusra

- Legyen a tömrítendő szöveg az S= ABBRRAKKADABBRA
- Az ábécé betüi és gyakoriságuk:
  A – 5
  B – 4
  D – 1
  K – 2
  R – 3
  összesen: 15 karakter
- **Naív** módszert használva ($\lceil \log_2 5 \rceil = 3$) 15*3 = **45 bit** lenne a tömörített hossz.

Kódfa építése

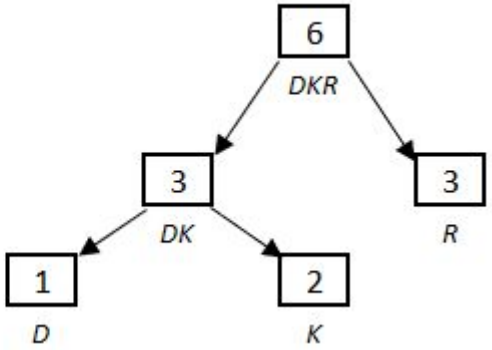
- Rendezzük gyakoriság szerint a betűket, válasszuk a két legkisebb gyakoriságút:

D	1	
K	2	3
R	3	3
B	4	4
A	5	5



- Válasszuk megint a két legkisebb gyakoriságú csúcsot (az egyik levél, a másik egy belső csúcs)

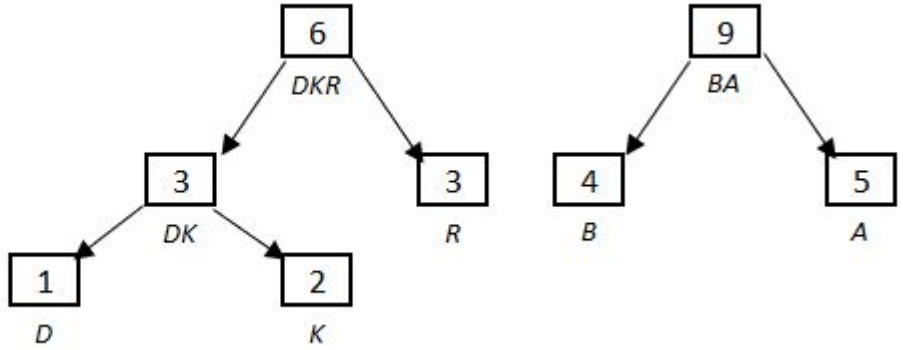
D	1		
K	2	3	
R	3	3	6
B	4	4	4
A	5	5	5



Kódfa építése

- Így folytatjuk, most a ’B’ és ’A’ leveleket kötjük össze:

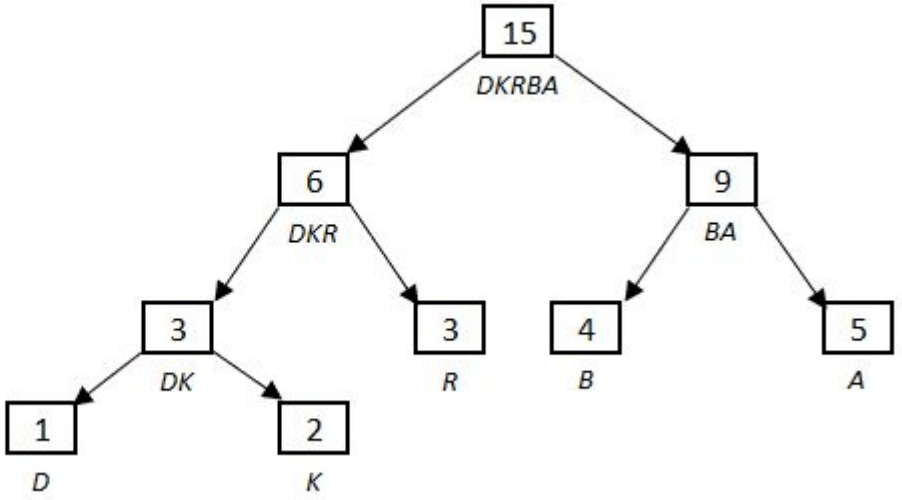
D	1			
K	2	3		
R	3	3	6	6
B	4	4	4	
A	5	5	5	9



Kódfa építése

- Végül a két belső pontot kötjük össze:

D	1				
K	2	3			
R	3	3	6	6	
B	4	4	4		
A	5	5	5	9	15

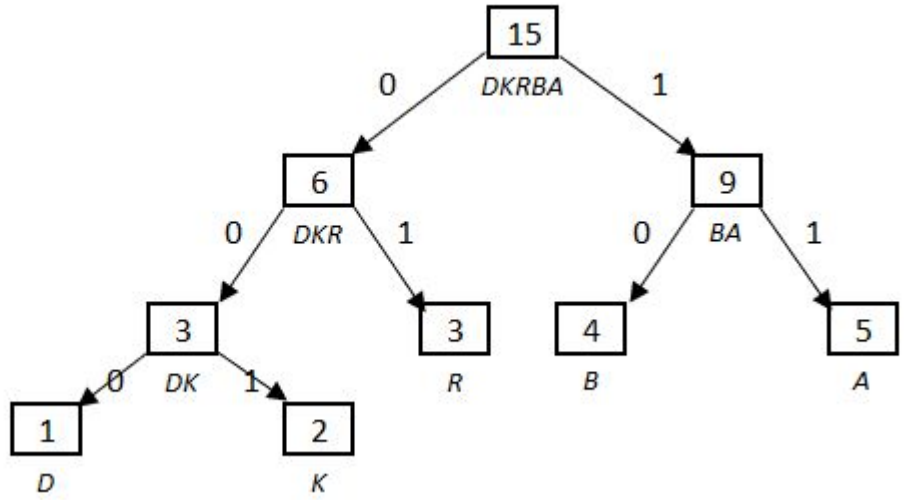


Kódtábla előállítása

- Felcímkézzük az éleket, és kiolvassuk a levélhez vezető úton a cimkéket:

Kódtábla

A	11
B	10
D	000
K	001
R	01

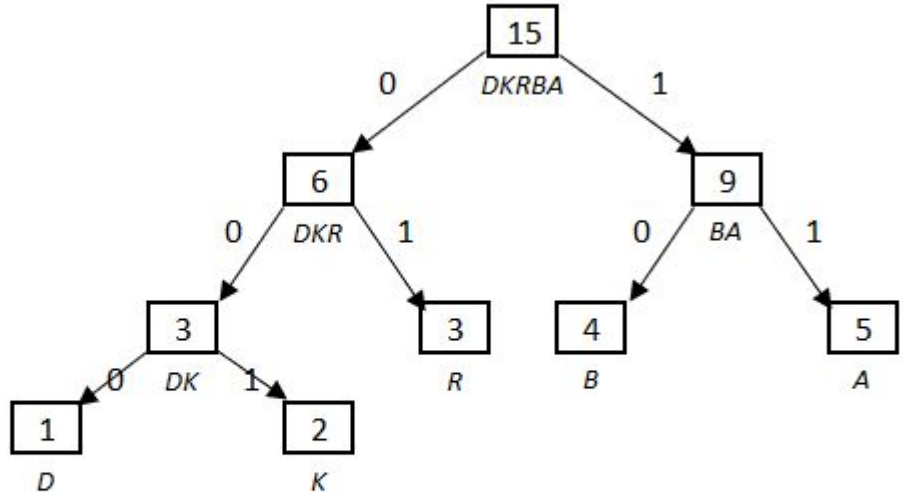


Kódolás

- Kódtábla alapján kódoljuk a szöveg betűit, kiírjuk az outputra.

Kódtábla

A	11
B	10
D	000
K	001
R	01



Kódolt szöveg:

A	B	B	R	R	A	K	K	A	D	A	B	B	R	A
11	10	10	01	01	11	001	001	11	000	11	10	10	01	11

Kapott kód hosszának kiszámítása

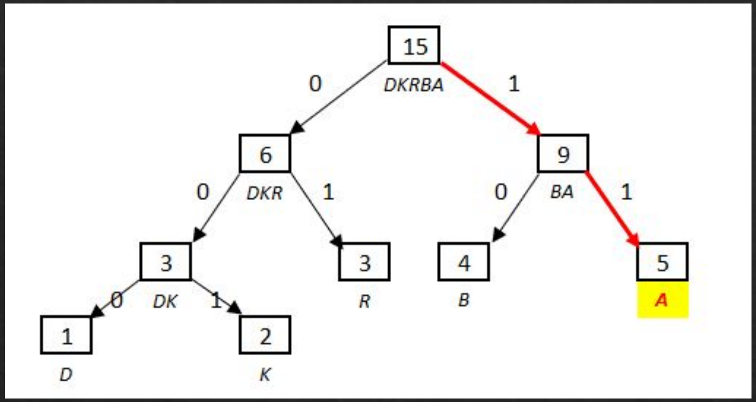
- Egy alkalmas táblázattal könnyen kiszámíthatjuk a kódolt szöveg hosszát.
- Emlékeztetőül, a naív tömörítéssel kapott hossz 45 bit volt.
- Megjegyzés: a tömörített fájl mérete nagyobb lesz, hiszen a kicsomagoláshoz szükséges kódfát is tartalmazza.

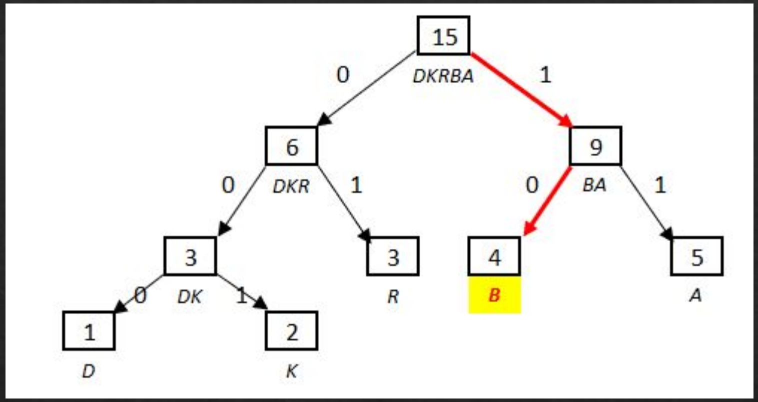
Kódolt szöveg hosszának kiszámítása:

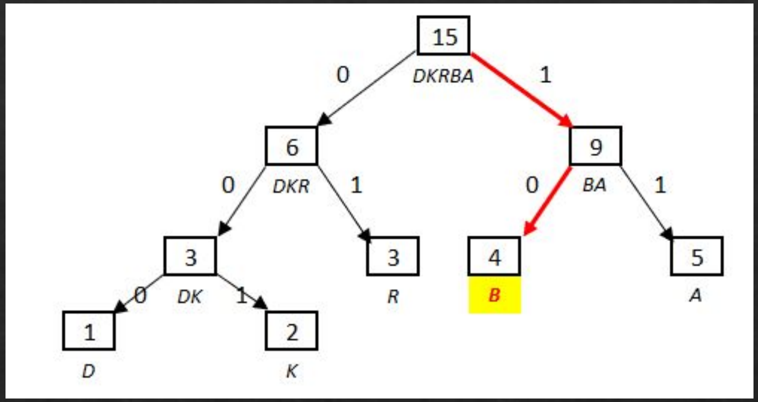
Betű	db	kódhossz	hossz	
A	5	2	10	bit
B	4	2	8	bit
D	1	3	3	bit
K	2	3	6	bit
R	3	2	6	bit
Összesen:			**33**	**bit**

Kicsomagolás szemléltetése

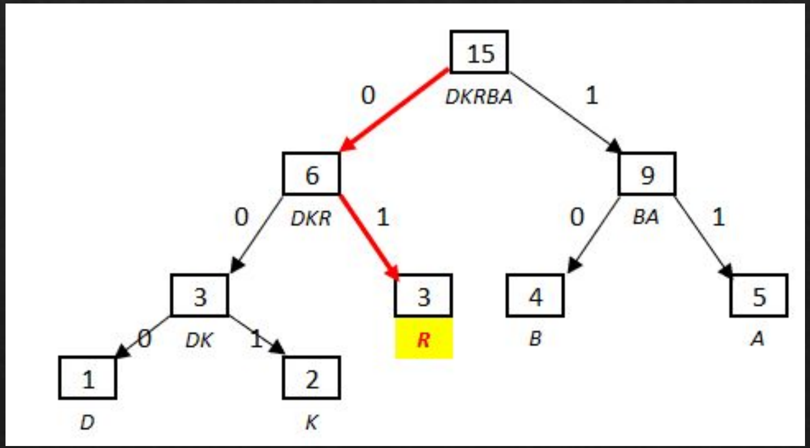
A tömrített bitsorozatot olvasva, mindig a kódfa gyökeréből indulunk, és levélig lépegetünk a bitek szerint: 11101001... 11101001... 11101001...







11101001...

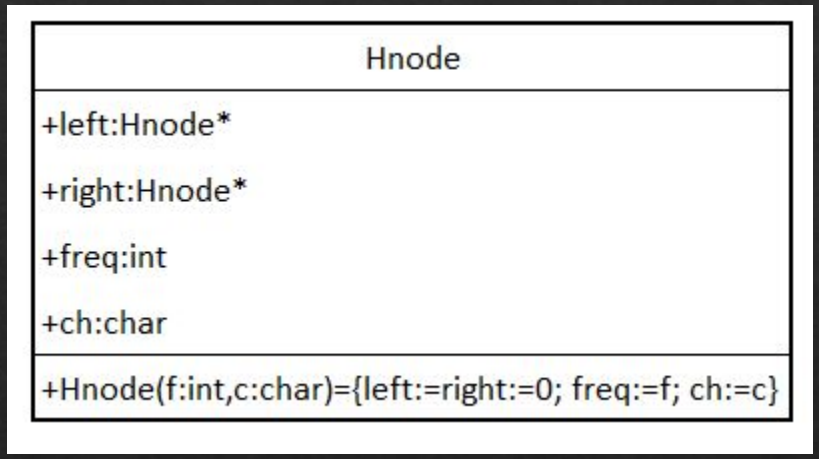


Gyakorló feladat

- Hány bit hosszúságban tömörítené a Naív algoritmus és a Huffman algoritmus a következő szöveget:
- #MESE_EMESE_SEMMISE#
- (Az aláhúzás és kettőskereszt is az ábécé eleme, a szöveg 20 betűből áll!)
- Ellenőrzés végett a helyes kódfával kapott eredmény: 48 bit lesz.

Huffman kódfa felépítésének algoritmusa

- Mielőtt az algoritmust felírjuk, egyezzünk meg a következőkben:
- A kódfa csúcsai Hnode típusúak lesznek. A node-ban freq tárolja a gyakoriságot, ch levelek esetén a betűt. A belső pontokban ch –t nem használjuk.
- A kényelmes használat végett legyen Hnode-nak az UML ábrán található konstruktora is.
- A prioritásos sorban Hnode-ra mutató pointerek lesznek. Freq adja a prioritást.
- A szöveg ábécéjének előállítását, és a betűk gyakoriságának meghatározását végző lépést nem részletezzük, programozásból ismert algoritmus.



Huffman kódfa felépítésének algoritmusa

Huffmankódfa(S:string):Hnode*

```
Előállítjuk az S szöveg ábécéjét (Σ) és az ábécé betűinek
gyakoriságát. A gyakoriságok egy, az ábécé betüivel
indexelhető f[] tömbben keletkeznek.
Q: minprQueue
for all σ ∈ Σ
    p:=new Hnode(f[σ], σ)
    Q.add(p)
Q.length() > 1
    p:=Q.remMin()    q:=Q.remMin()
    s:=new Hnode(p->freq+q->freq,' ')
    s->left:=p       s->right:=q
    Q.add(s)
return Q.remMin()
```

Feltesszük, hogy a szöveg nem üres, azaz legalább egy betűt tartalmaz.

minimum prioritásos sor létrehozása

kódfa leveleinek előállítása, és elhelyezése a pr.sorban

amíg a sor több mint egy elemű

kivesszük a két legkisebb prioritású elemet,

létrehozunk egy új belső pontot,

befűzzük a bal és jobb gyereket,

berakjuk a pr.sorba

visszatérés a fa gyökerére mutató pointerrel.

A törökök hány török tört törtek el?

LZW algoritmus - szótárkód

- Az LZW elnevezés Lempel-Ziv-Welch szerzőhármas nevének kezdőbetűiből származik. Az informatikában széles körben használt eljárást Terry Welch publikálta 1984-ben az Abraham Lempel és Jacob Ziv által 1978-ban közzétett LZ78 algoritmus továbbfejlesztéseként.
- 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 szótárt (kódtáblát, sztringtáblát) bővít lépésről-lépésre úgy, hogy egyre hosszabb már „látott" szavakhoz rendel új kódszót. Ezzel a valós adatoknak azt a tulajdonságát használjuk ki, hogy abban relatív rövid részek sűrűn ismétlődnek. Például gondoljunk élő nyelvben milyen sűrűn fordulnak elő névelők, kötőszavak, stb.

LZW szemléltetése

- Tömörítendő szövegünk legyen az S= ABABABAACAACCBBAAAAAAAAA
- Induló szótárunk az szöveg ábécéjét tartalmazza

Sztring	Kód
A	1
B	2
C	3
. . .	

- A bemenetet pontosan egyszer fogjuk végig olvasni a kódolás során úgy, hogy mindig a már ismert (kóddal rendelkező) leghosszabb következő szót keressük. Ha megtaláltuk, akkor:
  - kiírjuk a talált szó kódját a kimenetre (eredmény),
  - bővítjük a szótárt a *wc* szó képével, ahol *w* a szótárban már szereplő szó, *c* pedig a következő karakter.

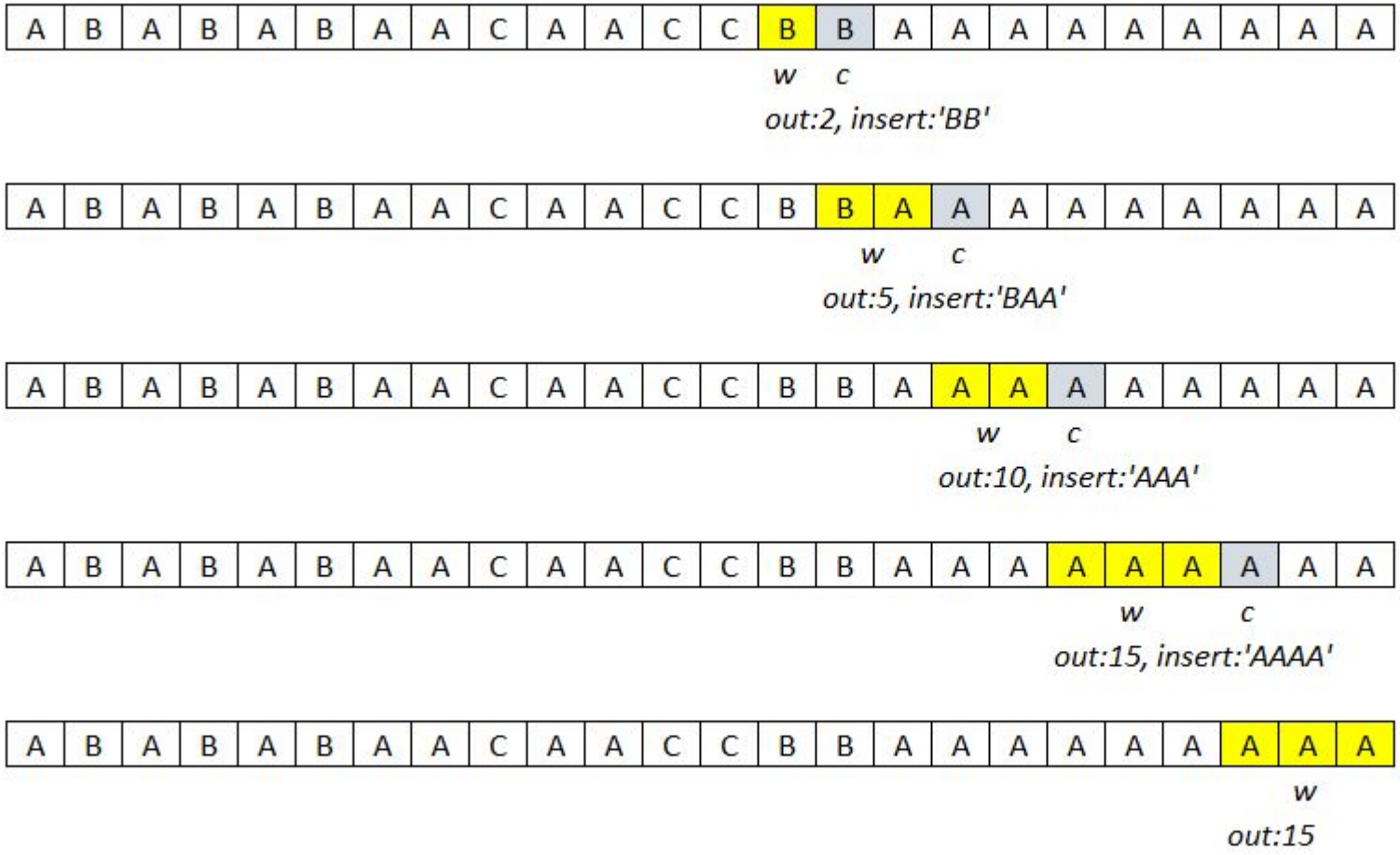
LZW szemléltetése

Sztring	Kód
A	1
B	2
C	3
AB	4
BA	5
ABA	6
ABAA	*7*

LZW szemléltetése

Sztring	Kód
A	*1*
B	*2*
C	3
AB	4
BA	5
ABA	6
ABAA	*7*
AC	8
CA	9
AA	10
ACC	11
CB	12

LZW szemléltetése



Sztring	Kód
A	*1*
B	*2*
C	3
AB	4
BA	5
ABA	6
ABAA	*7*
AC	8
CA	9
AA	10
ACC	11
CB	12
BB	13
BAA	14
AAA	15
AAAA	16

A	B	A	B	A	B	A	A	C	A	A	C	C	B	B	A	A	A	A	A	A	A	A	A

Tömörebb lejátszás

Kód	Aktuális szó	Következő betű	Új kód
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	---	---

output

Szting	Kód
A	1
B	2
C	3
AB	4
BA	5
ABA	6
ABAA	7
AC	8
CA	9
AA	10
ACC	11
CB	12
BB	13
BAA	14
AAA	15
AAA	16

Induló szótár

szótárba felvett szavak

LZW tömörítés algoritmusa

LZW_Compress(In,Out,Σ)

D:={<t_1,1> <t_2,2>.....<t_d,d>} kód:=d+1		
w:=get_char(In)		
¬ eof(In)		
	c:=get_char(In)	
	w ⊕ c ∈ D	
	w:= w ⊕ c	Write(Out, *kód* (D,w))
		D:= D ∪ {<w ⊕ c,kód>}
		w:=c
		kód++
Write(Out, kód(D,w))		

Kezdeti szótár létrehozása, kód segéd változó.

Egy betű olvasása, ez lesz az aktuális szó.

Van még az inputon betű?

Az inputról olvassuk a következő betűt.

w és c konkatenációja szerepel a szótárban?

ha igen, bővítjük az aktuális szót az utolónak beolvasott betűvel

ha nem, az aktuális szó kódját kiírjuk, felvesszük

w és c konkatenációját a szótárba,

az utolsónak olvasott betű lesz az aktuális szó.

In	tömörítendő szöveg
Out	kimenet
Σ	Input ábécé, betűi: t_1, ..., t_d
D	szótár, <sztring, kód> párokból áll
w	aktuális szó (ami benne van már a szótárban)
c	utolsónak olvasott betű

Megjegyzések

- 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;
  - kódszavakhoz tartozó szavak hosszának korlátozásával;
  - azzal, hogy a bemenet csak egy kezdőszeletén építjük a szótárat, utána csak kódolunk.
- 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.
- 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).

Gyakorló feladatok

- Szemléltessük a ABCABCABCAAABBCCAABABA kódolását.
- Keressünk olyan szöveget, aminek LZW kódolása rövidebb eredményt ad a Huffman-kódolással összehasonlítva.

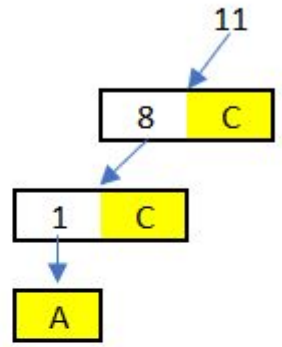
LZW kicsomagolás

- Kicsomagolás első lehetséges módja, hogy a teljes szótárt beletesszük a csomagba. Mivel a szótárban szereplő sztringek igen hosszúak is lehetnek, ez nagyon megnöveli a csomag méretét.
- Welch ötlete, a szótár tömörítésére (csak érdekesség, nem vizsga anyag):
- Végül megszületett a kicsomagoló algoritmus, melynek csak a kezdeti szótárra van szüksége!
- Itt folytatjuk...

Szting	Kód
A	*1*
B	*2*
C	3
AB	4
BA	5
ABA	6
ABAA	*7*
AC	8
CA	9
AA	10
ACC	11
CB	12
BB	13
BAA	14
AAA	15
AAAA	16

Szting	Kód
A	*1*
B	*2*
C	3
1B	4
2A	5
4A	6
6A	*7*
1C	8
3A	9
1A	10
8C	11
3B	12
2B	13
4A	14
10A	15
15A	16

például a 11 dekódolása:



kapott szó: ACC

a szavak az ismert prefix kódjából és az utolsó betűből állnak

Szorgalmi házi feladatok

- A szorgalmi házi feladatokat mindig a következő heti gyakorlat előtt lehet beküldeni a CANVAS felületen keresztül. Lehetőleg elektronikusan készítsük, de beküldhető kézzel írt, lefényképezett megoldás is, ez esetben ügyeljünk az olvashatóságra és a kép minőségére.

1. Válasszunk egy nem túl hosszú értelmes szöveget és mutassuk be rajta a Huffman algoritmust (gyakoriság, kódfa, kódtábla, kódhossz, kódhossz összehasonlítása a naív tömörítéssel). Az ábécé max. 10 féle betűből álljon, a szöveg hossza 20-30 betűből álljon, sokféle, érdekes gyakoriság legyen benne.
2. Készítsen rekurzív algoritmust, mely a Huffman kódfát bejárva, elkészíti a kódtáblát. A tavaly tanult, megfelelő bináris fa rekurzív bejáró algoritmusra támaszkodva járjuk be a fát, és leveleknél írjuk ki a szelektort. A szelektor előállításához egy paraméterként megadott verem, vagy string használható.