LZW

- **Lempel-Ziv-Welchův algoritmus (1978)**
- Kompresní/dekompresní metoda, využití – přenos dat, GIF, TIFF, postscript
- (+) rychlá metoda, (-) horší kompresní poměr (cca o 30% horší než nejlepší met.)
- Slovník – ukládají se opakující se znaky
- Pokud se vyskytne několik stejných posloupností znaků – nahrazení stejným číslem
- Slovník se neukládá, ale je znovu vytvořen ze zakódovaného souboru

LZW

- **Lempel-Ziv-Welchův algoritmus (1978) - komprese**

- Př. řetězec abcabcabcbcba (13 znaků), vstupní abeceda (použité znaky): a b c
přiřazení čísel a = 1, b = 2, c = 3
1. Nalezení nejdelší fráze ze slovníku shodné se vstupem, index na výstup, odebrání
2. Nová fráze = nalezená fráze + jeden další znak

Krok	text vstupu	nalezená fráze	výstup	nová fráze	Index nov. fráze
1	abcabcabcbcba	a	1	ab	4
2	bcabcabcbcba	b	2	bc	5
3	cabcabcbcba	c	3	ca	6
4	abcabcbcba	ab	4	abc	7
5	cabcbcba	ca	6	cab	8
6	bcbcba	bc	5	bcb	9
7	bcba	bcb	9		
8	a	a	1		

Výstup: řetězec 12346591 (8 čísel), kompresní poměr 61,5%

LZW

- **Lempel-Ziv-Welchův algoritmus (1978) - dekomprese**

- Př. řetězec 12346591, přiřazení čísel a = 1, b = 2, c = 3

1. K číslu přiřazená fráze
2. Nová fráze = fráze z předchozího kroku + první znak fráze výstupu

Pokud číslo na vstupu je větší než současné číslo ve vytvářeném slovníku: výstup = nová fráze = minulá fráze + její první znak

Krok	vstup	výstup	nová fráze	Index nov. fráze
1	1	a		
2	2	b	ab	4
3	3	c	bc	5
4	4	ab	ca	6
5	6	ca	abc	7
6	5	bc	cab	8
7	9	bcb	bcb	9
8	1	a		

Výstup: řetězec abcabcabcbcba

Huffmanovo kódování

- **Huffmanovo kódování**
- Algoritmus navržen Davidem Huffmanem (1952)
- Využití prefixového kódu - kód žádného znaku není prefixem jiného znaku, neprefixový kód: Morseova abeceda: A (.-), M(--), J(.---)
- Proměnná délka kódových slov >>> „znaky“, které jsou nejvíce četné mají nejkratší délku a naopak: 111011011, A (0), E(10), G(11)
- **Algoritmus kódování:**

1. Zjištění četnosti jednotlivých „znaků“
2. Vytvoření jednotlivých kódů na základě četností
3. Nahrazení jednotlivými znaků v datovém souboru nalezenými kódy

(+) Výhody – rychlá komprese a dekomprese, nenáročné na paměť

(-) Nevýhody – nutnost nalezených kódů, menší kompresní poměr

Huffmanovo kódování

- **Příklad:** znakový řetězec ABRAKADABRA
- 1) četnosti A (5x – 0,46), R (2x – 0,18), B (2x – 0,18), K (1x – 0,09), D (1x – 0,09)
- 2) vytvoření tabulky dle četností
- 3) poslední dvě četnosti se sečtou a zařadí se do tabulky, sčítá se až do 1

A 0,46		A 0,46		A 0,46		KDBR 0,54	1
R 0,18		R 0,18		KDB 0,36	1	A 0,46	0
B 0,18		B 0,18	1	R 0,18	0		
K 0,09	1	KD 0,18	0				
D 0,09	0						

- 4) Posledním dvěma slovům v každém sloupci tabulce přiřadíme 1 (vyšší četnost) a 0 (nižší četnost)
- 5) Výsledný kód znaku >>> posloupnost 0 a 1 dle toho jak se znak seskupoval s dalšími znaky, např. pro znak K: (1) – KDBR, (1) KDB, (0) KD a (1) K. Výsledné kódy A (0), R (10), B (111), K (1101), D (1100)
- 6) Výsledný řetězec pro ABRAKADABRA: 0 111 10 0 1101 0 1100 0 111 10 0, tj.: 01111001101011000111100
- Kompresní poměr (pokud 1 znak = 8 bit.): 0,26 (26%)

Aritmetické kódování

- **Aritmetické kódování**

- Huffmanův kód >>> problém při stejné pravděpodobnosti výskytu >>> možné řešení >>> aritmetické kódování

- Pro bezztrátovou kompresi dat, proměnná délka kódových slov jako u Huffmanova kódování, pří kódování se však vstupní znak nenahrazuje specifickým kódem, ale výsledek, tj. vstupní datový řetězec se nahradí reálným číslem z intervalu <0,1).

- **Algoritmus kódování:**

1. Zjištění četnosti (pravděpodobnosti výskytu) jednotlivých „znaků“
2. Dle pravděpodobnosti výskytu znaků se umístí znak v intervalu <0,1)
3. Celý interval <0,1) je postupně omezován z obou stran na základě přicházejících znaků.
4. Každý znak vybere z aktuálního intervalu odpovídající poměrnou část >>> nový základ pro následující symbol.
5. Po průchodu (načtení) všech znaků dostáváme podinterval z intervalu <0,1), výsledkem je pak libovolné reálné číslo z tohoto intervalu.
6. Na konec kódované zprávy dáme speciální znak, jinak při dekódování není možné určit konec datového toku, nebo uložíme délku původní posloupnosti znaků

Aritmetické kódování

- **Aritmetické kódování** – př.
- Datový řetězec CBAABCADAC (10 znaků)
- 1) Pravděpodobnosti výskytu A – 0.4 (P1), B – 0.2 (P2), C – 0.3 (P3), D – 0.1 (P4)
- 2) rozdělení v intervalu <0, 1):

<0, P1), <P1, P1 + P2), <P1 + P2, P1 + P2 + P3), <P1 + P2 + P3, P1 + P2 + P3 +P4)

Kumulativní pravděpodobnosti:

Q0 = 0, Q1= P1, Q2 = P1 + P2, … ,QN = P1 + P2 + … + PN = 1

<0, Q1), <Q1, Q2), <Q2, Q3), <Q3, Q4)
<0, 0.4), <0.4, 0.6), <0.6, 0.9), <0.9, 1)
A B C D

Aritmetické kódování

- **Aritmetické kódování** – př.
- Datový řetězec CBAABCADAC (10 znaků)
- Rozdělení intervalu

<0, 0.4), <0.4, 0.6), <0.6, 0.9), <0.9, 1)
A **B** **C** **D**

- 3) Kódování >>> postupné omezování intervalu I = <0, 1), I = <L, H)

>>> Postupně jsou brány znaky z datového řetězce, k nim známe IZ = <ZL, ZH)
>>> Nová hodnota intervalu IN = <L + ZL*(H - L), L + ZH*(H - L))

C >>> I = <0, 1), IN = <0 + 0.6*(1 – 0), 0 + 0.9*(1 - 0)) = <0.6, 0.9)
B >>> I = <0.6, 0.9), IN = <0.6 + 0.4*(0.9 – 0.6), 0.6 + 0.6*(0.9 – 0.6)) = <0.72, 0,78)
A >>> I = <0.72, 0.78), IN = <0.72 + 0*(0.78 – 0.72), 0.72 + 0.4*(0.78 – 0.72)) = <0.72, 0,744)
A >>> I = <0.72, 0.744), IN = <0.72 + 0*(0.744 – 0.72), 0.72 + 0.4*(0.744 – 0.72)) = <0.72, 0,7296)
B >>> I = <0.72, 0.7296), IN = <0.72 + 0.4*(0.7296 – 0.72), 0.72 + 0.6*(0.7296 – 0.72)) = <0.72384, 0.72576)
....
....
B >>> I = <0.72519936, 0.725208576), IN = <0.7252048896, 0.7252076544), C = 0.725205

Aritmetické kódování

- **Aritmetické dekódování** – př.
- Datový řetězec CBAABCADAC (10 znaků)
- Rozdělení intervalu

<0, 0.4), <0.4, 0.6), <0.6, 0.9), <0.9, 1)
A B C D

- Výsledek kódování >>> C = 0.725205

1) Počáteční hodnota intervalu dekódování I = <0, 1)
2) dekódování znaku: K = ((C – L) / (H – L)); ZL <= K < ZH >>> nalezneme odpovídající znak
3) počítáme nový interval IN = <L + ZL*(H - L), L + ZH*(H - L))
I = <0, 1), K = 0.725205, znak = C, IN = <0 + 0.6*(1 – 0), 0 + 0.9*(1 – 0) = <0.6, 0.9)
I = <0.6, 0.9), K = 0.41735, znak = B, IN = <0.6 + 0.4*(0.9 – 0.6),0.6 + 0.6*(0.9 – 0.6)) =<0.72, 0.78)
I = <0.72, 0.78), K = 0.08675, znak = A, IN = <0.72, 0.744)
I = <0.72, 0.744), K = 0.216875, znak = A, IN = <0.72, 0.7296)
….

Filtr Move-To-Front (MTF)

- 1980 (B. Ryabko), 1986 (J.K. Bentley) Přesuň na začátek, po provedení Burrowsovy-Wheelerovy transformace
- Dlouhé sekvence symbolů nahrazeny nulami, málo se vyskytuje: velké číslo
- Nahrazení symbolů ze vstupního řetězce jejich indexy ze zásobníku
- zásobník s čísli (pořadí, ASCII…), *i*-tá pozice – kód *i* z pořadí, ASCII
- Aktuálně kódovaný znak přesunut na počátek (dopředu)
- A až Z (26 čísel v zásobníku, + české znaky 42 písmen)
- ANANAS bude zakódován jako 1, 14, 2, 2, 2, 19
- A – 1 (poté: ABCDEFGHIJKLMNO...)
- N – 14 (poté: NABCDEFGHIJKLMO...)
- A – 2 (poté: ANBCDEFGHIJKLMO…)
- N – 2 (poté: NABCDEFGHIJKLMO...)
- A – 2 (poté: ANBCDEFGHIJKLMO...)
- S – 19 (poté: SANBCDEFGHIJKLMO...)

1	A	14	N
2	B	15	O
3	C	16	P
4	D	17	Q
5	E	18	R
6	F	19	S
7	G	20	T
8	H	21	U
9	I	22	V
10	J	23	W
11	K	24	X
12	L	25	Y
13	M	26	Z

CV11

Filtr Move-To-Front (MTF)

- ANANAS – ASCII

- A – 65, (poté A = 0)
- N – 78, (poté N = 0, A = 1)
- A – 1 (poté A = 0, N = 1)
- N – 1 (poté N = 0, A = 1)
- A – 1 (poté A = 0, N = 1)
- S – 83 (poté S = 0, A = 1…)

000	00		043	2B	+	086	56	V	129	81	ü	172	AC	¼	215	D7	╫
001	01	☺	044	2C	,	087	57	W	130	82	é	173	AD	¡	216	D8	╪
002	02	☻	045	2D	-	088	58	X	131	83	â	174	AE	«	217	D9	┘
003	03	♥	046	2E	.	089	59	Y	132	84	ä	175	AF	»	218	DA	┌
004	04	♦	047	2F	/	090	5A	Z	133	85	à	176	B0	░	219	DB	█
005	05	♣	048	30	0	091	5B	[	134	86	å	177	B1	▒	220	DC	▄
006	06	♠	049	31	1	092	5C	\	135	87	ç	178	B2	▓	221	DD	▌
007	07	•	050	32	2	093	5D	]	136	88	ê	179	B3	│	222	DE	▐
008	08	◘	051	33	3	094	5E	^	137	89	ë	180	B4	┤	223	DF	▀
009	09		052	34	4	095	5F	_	138	8A	è	181	B5	╡	224	E0	α
010	0A		053	35	5	096	60	`	139	8B	ï	182	B6	╢	225	E1	ß
011	0B	♂	054	36	6	097	61	a	140	8C	î	183	B7	╖	226	E2	Γ
012	0C	♀	055	37	7	098	62	b	141	8D	ì	184	B8	╕	227	E3	π
013	0D		056	38	8	099	63	c	142	8E	Ä	185	B9	╣	228	E4	Σ
014	0E	♫	057	39	9	100	64	d	143	8F	Å	186	BA	║	229	E5	σ
015	0F	☼	058	3A	:	101	65	e	144	90	É	187	BB	╗	230	E6	µ
016	10	►	059	3B	;	102	66	f	145	91	æ	188	BC	╝	231	E7	τ
017	11	◄	060	3C	<	103	67	g	146	92	Æ	189	BD	╜	232	E8	Φ
018	12	↕	061	3D	=	104	68	h	147	93	ô	190	BE	╛	233	E9	Θ
019	13	‼	062	3E	>	105	69	i	148	94	ö	191	BF	┐	234	EA	Ω
020	14	¶	063	3F	?	106	6A	j	149	95	ò	192	C0	└	235	EB	δ
021	15	§	064	40	@	107	6B	k	150	96	û	193	C1	┴	236	EC	∞
022	16	▬	065	41	A	108	6C	l	151	97	ù	194	C2	┬	237	ED	ø
023	17	↨	066	42	B	109	6D	m	152	98	ÿ	195	C3	├	238	EE	∈
024	18	↑	067	43	C	110	6E	n	153	99	Ö	196	C4	─	239	EF	∩
025	19	↓	068	44	D	111	6F	o	154	9A	Ü	197	C5	┼	240	F0	≡
026	1A		069	45	E	112	70	p	155	9B	¢	198	C6	╞	241	F1	±
027	1B	←	070	46	F	113	71	q	156	9C	£	199	C7	╟	242	F2	≥
028	1C	∟	071	47	G	114	72	r	157	9D	¥	200	C8	╚	243	F3	≤
029	1D	↔	072	48	H	115	73	s	158	9E	₧	201	C9	╔	244	F4	⌠
030	1E	▲	073	49	I	116	74	t	159	9F	ƒ	202	CA	╩	245	F5	⌡
031	1F	▼	074	4A	J	117	75	u	160	A0	á	203	CB	╦	246	F6	÷
032	20		075	4B	K	118	76	v	161	A1	í	204	CC	╠	247	F7	≈
033	21	!	076	4C	L	119	77	w	162	A2	ó	205	CD	═	248	F8	°
034	22	"	077	4D	M	120	78	x	163	A3	ú	206	CE	╬	249	F9	∙
035	23	#	078	4E	N	121	79	y	164	A4	ñ	207	CF	╧	250	FA	·
036	24	$	079	4F	O	122	7A	z	165	A5	Ñ	208	D0	╨	251	FB	√
037	25	%	080	50	P	123	7B	{	166	A6	ª	209	D1	╤	252	FC	ⁿ
038	26	&	081	51	Q	124	7C	¦	167	A7	º	210	D2	╥	253	FD	²
039	27	'	082	52	R	125	7D	}	168	A8	¿	211	D3	╙	254	FE	■
040	28	(	083	53	S	126	7E	~	169	A9	⌐	212	D4	╘	255	FF	
041	29	)	084	54	T	127	7F	⌂	170	AA	¬	213	D5	╒			
042	2A	*	085	55	U	128	80	Ç	171	AB	½	214	D6	╓			

Filtr Move-To-Front (MTF)

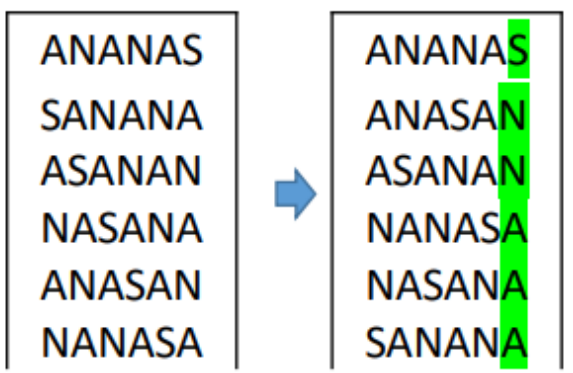
- **Dekódování**

- 1 - A (poté: ABCDEFGHIJKLMNO...)
- 14 - N (poté: NABCDEFGHIJKLMO...)
- 2 - A (poté: ANBCDEFGHIJKLMO…)
- 2 - N (poté: NABCDEFGHIJKLMO...)
- 2 - A (poté: ANBCDEFGHIJKLMO...)
- 19 - S (poté: SNABCDEFGHIJKLMO...)

1	A	14	N
2	B	15	O
3	C	16	P
4	D	17	Q
5	E	18	R
6	F	19	S
7	G	20	T
8	H	21	U
9	I	22	V
10	J	23	W
11	K	24	X
12	L	25	Y
13	M	26	Z

Burrows-Wheelerova Transformace (BWT)

- Michael Wheeler (1984) + David Burrows (1994)
- Přeskupí symboly tak, že shodné budou s vysokou pravděpodobností vedle sebe, pak můžeme použít RLE…
- Bzip2 – BWT >>> MTF >>> Huffmanovo kódování
- Vstupní řetězec se zpracovává po blocích o konstantní velikosti, větší bloky – větší efektivita, blok cca desítky tisíc symbolů (Block Sorting Algorithm)
- 1) blok o délce N (např. ANANAS), N cyklických posuvů o jeden symbol doprava >>> matice N * N, řádky => cyklické posuny
- 2) lexikografické (podle abecedy) setřídění matice
- 3) poslední sloupec je výsledek transformace
- 4) přidáme číslo řádku v setříděné matici, kde se nachází vstupní řetězec: SNNAAA, 1. ŘÁDEK



CV11

Burrows-Wheelerova Transformace (BWT)

- **Dekódování:** SNNAAA, 1. ŘÁDEK

1	sort	2	sort	3	sort	4	sort	5	sort	6	sort
S	A	SA	AN	SAN	ANA	SANA	ANAN	SANAN	ANANA	SANANA	ANANAS
N	A	NA	AN	NAN	ANA	NANA	ANAS	NANAS	ANASA	NANASA	ANASAN
N	A	NA	AS	NAS	ASA	NASA	ASAN	NASAN	ASANA	NASANA	ASANAN
A	N	AN	NA	ANA	NAN	ANAN	NANA	ANANA	NANAS	ANANAS	NANASA
A	N	AN	NA	ANA	NAS	ANAS	NASA	ANASA	NASAN	ANASAN	NASANA
A	S	AS	SA	ASA	SAN	ASAN	SANA	ASANA	SANAN	ASANAN	SANANA

- **VÝSTUP:** ANANAS

IDK bylo to v přednášce tak jsem to sem raději dal

BWT + MTF

- **BWT Dekódování:** SNNAAA, 1. ŘÁDEK

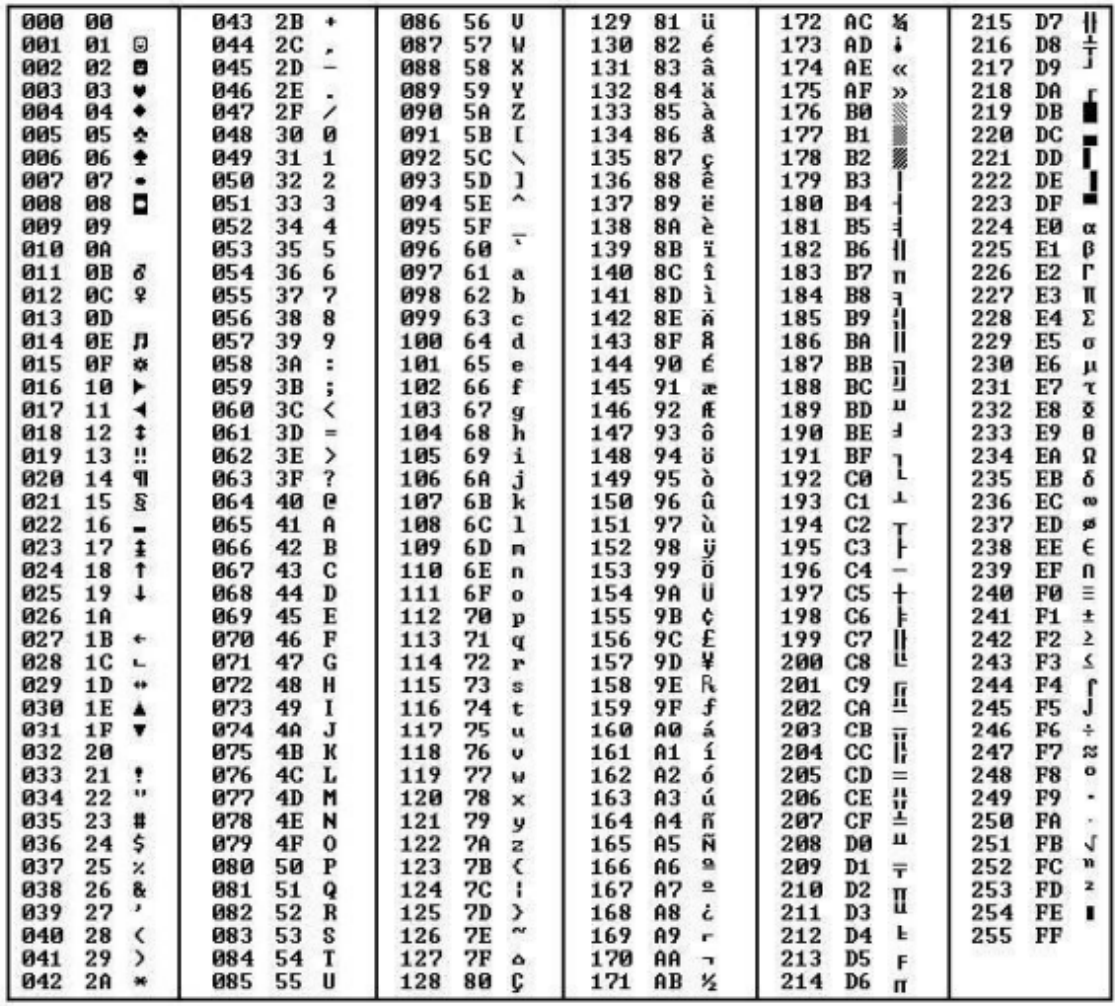
- **MTF:**

S – 83 (poté: S = 0)
N – 78 (poté: N = 0, S = 1)
N – 0
A – 65 (poté: A = 0, N = 1, S = 2)
A – 0
A – 0

pro Huffmanovo kódování:

0 – 3x
65 – 1x
78 – 1x
83 – 1x

U delších řetězců – vyšší počet 0



CV12

Bezpečnostní kódování

- **Inverzní kód**

- Vysílaný kód se při vysílání opakuje beze změn, pokud je v něm sudý počet 0, pokud máme lichý počet, tak se opakovaný kód invertuje, D = 4, lze opravit jednoduchou chybu

sudý poč. 0: 01011 01011
lichý poč. 0: 10111 01000

vysláno:	01011 01011	01001 10110	10111 01000
přijato:	00011 01011	01011 10110	10111 00000
překódováno:	00011 10100	01011 10110	10111 11111
oprava:	00011 ⊕ 10100 = 10111 chyba infor. části kódu	01011 ⊕ 10110 = 11101 chyba infor. části kódu	10111 ⊕ 11111 = 01000 chyba zab. části kódu

- **Dekódování**
- Přijatý kód >>> kontrola >>> není správně >>> oprava – dekódování,Není to inverzní postup ke kódování

- Máme kódy 000000, 111111, alespoň 4 znaky jsou správně 100100 >>> 000000