Berouza - Vīlera algoritms (Burrows-Wheeler)

1990. gadi - samērā nesens algoritms. (Izmantots bzip2 arhivētājā - piezīme)

Algoritma soļi:
Simbolu pārkārtošana → Move to front → Huffman/aritmētiskā kodēšāna.

Kodēšana ar pārvietošanu uz priekšu (Move-to-front)

1. Alfabēta burtus liek stekā, kura pozīcijas tiek numurētas sākot ar 0.
2. Katru burtu kodē ar tā pozīciju stekā, vienlaikus pārvietojot to uz steka augšu.

↓ steks → simbolu virkne

	a	b	a	c	c	a	b
a	a	b	a	c	c	a	b
b	b	a	b	a	a	c	a
c	c	c	c	b	b	b	c
Izeja :	0	1	1	2	0	1	2

Ideja: biežāk lietoti burti atradīsies steka augšā, izejā būs galvenokārt mazi skaitļi.

Piemērs

Nokodēt abababca

	a	b	a	b	a	b	c	a
a	a	b	a	b	a	b	c	a
b	b	a	b	a	b	a	b	c
c	c	c	c	c	c	c	a	b
Izeja :	0	1	1	1	1	1	2	2

Atkodēt 0 1 0 0 1 0

	0	1	0	0	1	0
a	a	b	b	b	a	a
b	b	a	a	a	b	b
c	c	c	c	c	c	c
Izeja :	a	b	b	b	a	a

Bērouza - Vīlera transformācija

Panākt, lai vienu un tie paši burti atkārtojas pēc neliela burtu skaita vēl biežāk.
Parasti pielieto teksta blokiem ∼1000 simbolu garumā.
Sadala tekstu blokos, garumā k.
a a b a b a a b c - viens bloks garumā 9

No katra bloka ar cikliskām nobīdēm izveido k simbolu virknes

a	a	b	a	b	a	a	b	c
c	a	a	b	a	b	a	a	b
b	c	a	a	b	a	b	a	a
a	b	c	a	a	b	a	b	a
a	a	b	c	a	a	b	a	b
b	a	a	b	c	a	a	b	a
a	b	a	a	b	c	a	a	b
b	a	b	a	a	b	c	a	a
a	b	a	b	a	a	b	c	a

Sakārto šīs virknes alfabētiski, pēc priekšpēdējā simbola. (Ja priekšpēdējie simboli ir vienādi, tad pēc trešā no beigām. Tad ceturtā, u.t.t.)

c	a	a	b	a	b	a	a	b
a	b	a	a	b	c	a	a	b
a	a	b	c	a	a	b	a	b
b	c	a	a	b	a	b	a	a
b	a	b	a	a	b	c	a	a
a	a	b	a	b	a	a	b	c
b	a	a	b	c	a	a	b	a
a	b	c	a	a	b	a	b	a
a	b	a	b	a	a	b	c	a

Transformācijas rezultāts - pēdējā kolonna
b b b a a c a a a

Vienādi burti ir blakus!

	b	b	b	a	a	c	a	a	a
a	b	b	b	a	a	c	a	a	a
b	a	a	a	b	b	a	c	c	c
c	c	c	c	c	c	b	b	b	b
Izeja :	1	0	0	1	0	2	1	0	0

Līdz ar to, izejā visbiežāk ir mazie skaitļi.

Kāpēc šī metode dod labus rezultātus?

Piemēram blokā 5 reizes atkārtojas "algoritms". Pēc BV transformācijas:

		.	.	.	
...	a	l	g	o	r
...	a	l	g	o	r
...	a	l	g	o	r
...	a	l	g	o	r
...	a	l	g	o	r
		.	.	.	

Blakus nonāk visas vietas tekstā, kur ir "algo". Nākošais burts ar lielu vārbūtību ir "r". Blakus nonāks arī visas vietas tekstā, kur ir "alg". Nākošais burts ar lielu vārbūtūbu būs "o". Utt.

Atkodēšana

1. Sakārtojot simbolus pēc alfabēta, iegūst priekšpēdējo kolonnu.
2. Priekšpēdējā + pēdējā kolonna ⇒ visas divu burtu virknes. Piekārtojot tās ar 2. burtu priekšpēdējai kolonnai un sakārtojot alfabētiski, iegūst 2. kolonnu no beigām.
Tās jau ir visas trīs burtu virknes. Piekārtojot tās ar 2. un 3. burtiem 2. un priekšpēdējai kolonnai un sakārtojot alfabētiski, iegūst 3. kolonnu no beigām. Utt.
3. Atkodēšanai jāzin, kura rinda ir sākotnējais teksts, tāpēc vajag papildus glabāt tās numuru.

Piemēram:
Pēdējā kolonna "b b b a a c a a a" dod priekšpēdējo kolonnu "a a a a a b b b c". Tās abas kopā ir:

a	b
a	b
a	b
a	a
a	a
b	c
b	a
b	a
c	a

Piekārtojot šos pārus ar otro simbolu priekšpēdējai kolonnai un sakārtojot alfabētiski pēc pirmā simbola, iegūst:

$a\ a\ b$
$a\ a\ b$
$b\ a\ b$
$b\ a\ a$
$c\ a\ a$
$a\ b\ c$
$a\ b\ a$
$a\ b\ a$
$b\ c\ a$

Utt.

Vai jebkurai "nokodētai virknei" atbilst kāds izejas teksts?

k izejas virknes garumā k (cikliski nobīdītas) kodējas pret 1 virkni garumā k. Līdz ar to ir jābūt arī virknēm garumā k, pret kurām nekodējas nekas.
Piemēram, pret virkni "a b" nekodējas nekas, jo cikliskās nobīdes ir acīmredzot "a b", "b a"; pēc pēdējās kolonnas ir jābūt $\begin{matrix} b & a \\ a & b \end{matrix}$, bet tie nav sakārtoti alfabētiski (pēc priekšpēdējā simbola).

Nobeigums

Šis ir viens no labākiem zināmiem tekstu saspiešanas algoritmiem. Ir vēl zināms cits algoritms, kas saspiež par 10% labāk, bet ir daudz lēnāks.

Saspiešana ar zudumiem (Attēli, skaņa, u.c.)

1. Skalārā kvantizācija.

Attēlā skaitlis 0-255 apzīmē krasu balta - melna
Visus 256 toņus acs neatšķir, tāpēc var attēlot 256 krāsas uz mazāku skaitu.
Vienkāršākais attēlojums, piem. $f(x) = \lfloor \frac{x}{4} \rfloor$. $f : \{0..255\} \to \{0..63\}$
Praksē lieto sarežģītāku funkciju, kas kopā sagrupē krāsas, kuras acs sliktāk atšķir.

2. Vektoru kvantizācija

Piemēram, krasaini attēli.
Punkts = 3 vērtības (Red, Green, Blue). $\{0..255\}^3$.
$f(x_1, x_2, x_3) = y$, tā, lai dažādi trijnieki (x_1, x_2, x_3), kas attēlojas par vienu y, būtu grūti atšķirami.

3. Transformāciju kodēšana

Ieejas dati: (x_1,x_2,...,x_n)

$$\begin{pmatrix} a_{11} & ... & a_{1n} \\ a_{21} & ... & a_{2n} \\ ... & ... & ... \\ a_{\mathrm{n1}} & ... & a_{\mathrm{nn}} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ ... \\ x_n \end{pmatrix} = \begin{pmatrix} x'_1 \\ x'_2 \\ ... \\ x'_n \end{pmatrix}, \text{ kur}$$

$$\begin{pmatrix} a_{11} & ... & a_{1n} \\ a_{21} & ... & a_{2n} \\ ... & ... & ... \\ a_{\mathrm{n1}} & ... & a_{\mathrm{nn}} \end{pmatrix} = A \text{ - transformācijas matrica, } X \text{ - ieejas dati, } X' \text{ - rezultāts.}$$

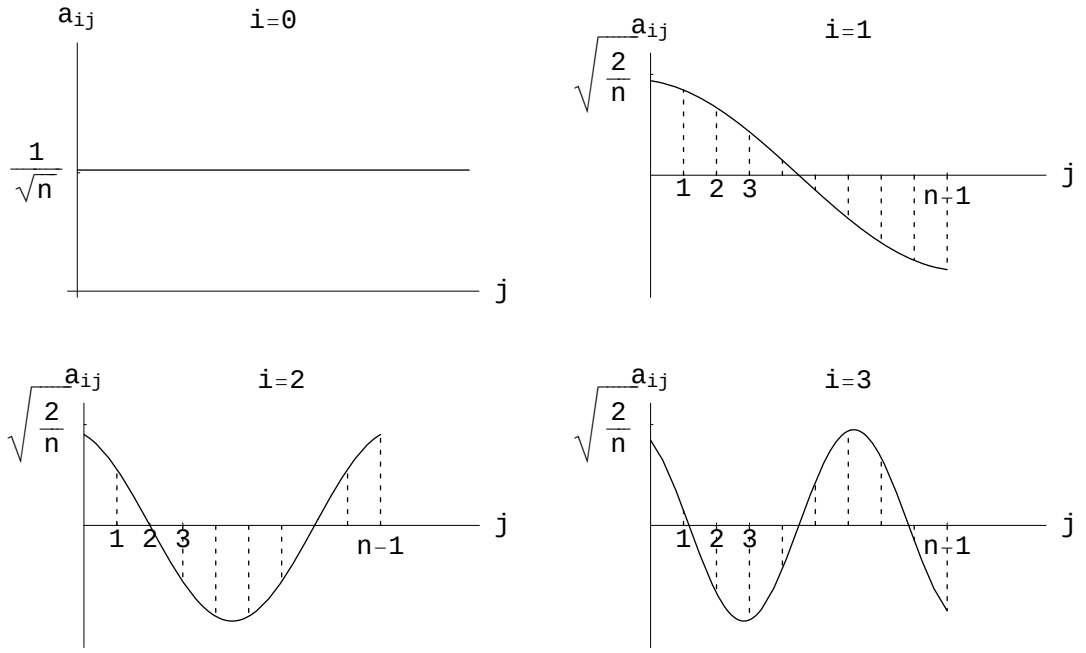
Atkodēšana:
$A^{-1} \cdot X' = A^{-1}(AX) = (A^{-1}A)X = X$.
Pēc transformāciju kodēšanas parasti lieto kvantizāciju.

Transformāciju piemēri

Discrete Cosine Transform.

$a_{\mathrm{ij}}=\begin{cases} \sqrt{1/n}, & i=0 \\ \sqrt{\frac{2}{n}}\cos\frac{(2j+1)i\pi}{2n}, & i\neq 0 \end{cases}$

x'_i izsaka līdzību starp $(x_1,...,x_n)$ un $(a_{\mathrm{i1}}, a_{\mathrm{i2}}, ..., a_{\mathrm{in}})$.



Kosinusa vietā var ņemt arī citu funkciju, tad koordinātas mērīs līdzību starp datiem un to funkciju. Piemēram, wavelet (viļņfunkcijas):