Datu saspiešana III

Lempela - Ziva algoritms.

Šis algoritms parasti tiek lietots tekstu saspiešanai.

Algoritma pamatideja: izveidot vārdnīcu ar fragmentiem, kas atkārtojas tekstā un attiecīgo fragmentu vietās raksta tiem atbilstošo vārdnīcas fragmentu identifikatorus.

Šim algoritmam ir 2 varianti: LZ77 un LZ78 (zināms arī kā Lempela - Ziva - Welch algoritms). Abi šie varianti atšķiras ar to, kādā veidā tiek iegūta vārdnīca.

LZ78 algoritms.

```
1.  w = null
2.  while (k = readchar())
3.  begin
4.    if wk in dictonary then
5.      w = wk
6.    else
7.    begin
8.      output code for w
9.      add wk to dictionary
10.     w = k
11.   end
12. end
13. output code for w
```

Sākumā vārdnīcā atrodas visi burti.

Piemērs 1.

Dota virkne 'abcabcabcdabcaba', kura jānokodē, izmantojot Lempela - Ziva algoritmu.

Soļa nr.	w (garākais fragments, kas atrodams vārdnīcā)	k	Izvads	Pievieno vārdnīcai
1.	a	b	a	ab
2.	b	c	b	bc
3.	c	a	c	ca
4.	ab	c	ab	abc

5.	ca	b	ca	cab
6.	bc	d	bc	dbc
7.	d	a	d	da
8.	abc	a	abc	abca
9.	ab	a	ab	aba
10.			a	

šajā tabulā ir izlaisti visi tie soļi, kad izrādās, ka virkne wk jau atrodas vārdnīcā.

Parasti kodē burtus par burtiem, bet garākas virknes aizstāj ar tā soļa numuru, kurā šī virkne ir ievietota vārdnīcā. Tas nozīmē, ka 1. piemērā virkne 'ab' tiktu kodēta kā 1, bc kā 2, ca kā 3, u.t.t. Tas nozīmē, ka izvadā tiktu izvadīta virkne a,b,c,1,3,2,d,4,1,a.

Piemērs 2.
Atkodēt virkni: a,a,b,1,2,4,2

Soļa nr.	w (garākais fragments, kas atrodams vārdnīcā)	K	Izvads	Pievieno vārdnīcai
1.	a	A	a	aa[1]
2.	a	B	a	ab[2]
3.	b	A	b	ba[3]
4.	aa	A	aa	aaa[4]
5.	...	...	...	...

Rezultāts : 'aabaaabaaaab'.

Piemērs 3.
Atkodēt virkni a,b,a,3,4

Soļa nr.	w (garākais fragments, kas atrodams vārdnīcā)	K	Izvads	Pievieno vārdnīcai
1.	a	B	a	ab[1]
2.	b	A	b	ba[2]
3.	a	A	a	aa[3]
4.	aa	A	aa	aaa[4]
5.		...	...	...

Rezultāts: ‘abaaaaaa’.
Šajā piemērā interesantākais ir 3. solis, kurā jāpielieto simboli no vēl neatkodētā simbola 3, kas apzīmē 3. solī iegūto simbolu virkni. Šajā gadījumā ir iespējams noskaidrot

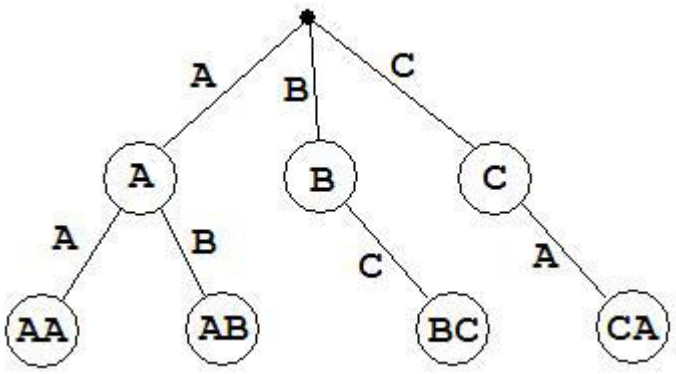
simbolu, kurš seko aiz 3. vietā esošā ‘a’ burta. Tā kā ir zināms, ka izvadā būs viens pats ‘a’ burts tas nozīmē, ka w sastāv no 1 simbola ‘a’.
No tā var secināt, ka 3. solī iegūtā simbolu virkne sākas ar ‘a’ un tā kā pēc ‘a’ saspiestajā virknē seko ‘3’, tas nozīmē, ka simbols k ir ‘a’.

Iepriekšējos piemēros demonstrētais atkodēšanas algoritms formālā pierakstā:

```
1.  w = lookup(readcode())
2.  output w
3.  while (c = readcode())
4.  begin
5.    if c in dictionary then
6.      wn = lookup(c)
7.    else
8.      wn = stringcat(w, w[0])
9.    output wn
10.   k = wn[0]
11.   add wk to dictionary;
12.   w = wn
13. end
```

Datu struktūras vārdnīcai.

1. Trie = īpaša veida koks. Piemēram, vārdnīca A, B, C, AB, BC, CA, AA.



Virsotnes dziļumā i atbilst virknēm garumā i. Katram burtam atbilst šķautne, bet lapās ierakstīts vārdam atbilstošais vārdnīcas kods.

LZ78 implementācija: potenciālās problēmas.

1. Vārdnīca var kļūt pārāk liela.
2. Vārdnīca var kļūt neadekvāta šobrīd sastopamajam tekstam (ja teksta beigās bieži sastopamās virknes ir citādas, kā sākumā).

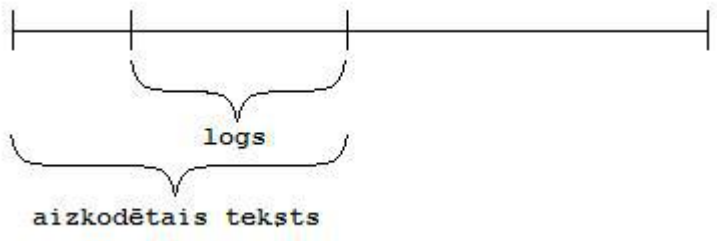
Risinājumi:

1. Pēc noteikta simbolu skaita vārdnīcu iznīcina un sāk no tās aizpildīšanu no gala.
2. Vārdnīcu aizstāj ar tukšu, ja tā kļūst neadekvāta.
3. Katrā solī izmet reti sastopamu simbolu virkni.

LZ77

LZ77 algoritma pamatā ir tā pati ideja, kas LZ78, tikai bez vārdnīcas izveidošanas.

Kā vārdnīcu izmanto jau saspiesto (vai atspiesto) teksta daļu.



Par logu sauksim pēdējos k simbolus no saspiestā teksta.

Algoritmam ir divu veida izejas dati:

(0,x,y) - teksta sākums sakrīt ar to, kas ir logā no x-tās vietas y simbolu garumā.
(1,s) - simbols s.

Piemērs.
Aizkodēt virkni 'abcabcabcdabc', ja loga garums k = 6;
(1,a),(1,b),(1,c),(0,1,6),(1,d),(0,3,3)

Piemērs.
k = 6, 'aababcab' (1,a),(0,1,1),(1,b),(0,2,2),(1,c),(0,2,2)

Saspiešana vairākos soļos.

Uzskatāms pielietojums saspiešanai vairākos soļos ir datu pārraidei ar faksa palīdzību.

Datu pārraidē ar faksu ir:

attēls ar izmēru x*y punkti, kur x, y ⌐╝ 1000

katram punktam ir viena no 2 krāsām - balta vai melna.

Pirmais solis saspiešanā ir 'Run - length encoding'

Ieejā ir bitu virkne, piemēram, '00011101011'

Aizkodējot virkni pirmo simbolu, liek 1. aizkodējamās virknes simbolu, bet tālāk tiek rakstīti skaitļi, kas apzīmē vienādo simbolu skaitu pēc kārtas - dotā virkne aizkodēta būtu 0,3,3,1,1,1,2.

Otrais solis – ņem Run-length encoding algoritma rezultātu un pielieto Hofmana vai aritmētisko algoritmu – uztver skaitļus kā simbolus un nosaka to biežumus.

Aritmētiskais algoritms ir ļoti lietderīgs, kā pēdējais solis.
To pašu var darīt ar LZ77 izejas datiem, ja katram reģistram LZ77 algoritma izejā pielieto atsevišķi Hofmana vai aritmētisko metodi.