HAFMENA SASPIEŠANAS ALGORITMS
HUFFMAN COMPRESSION ALGORITHM

Autors: **Edgars Kairišs**, e-pasts: edgars.kairish@gmail.com, +37120093673
Zinātniskā darba vadītājs: **Mihails Kijaško, Mg.sc.comp.,** e-pasts: Mihails.Kijasko@rta.lv
Rēzeknes Tehnoloģiju akadēmija, Atbrīvošanas aleja 115, Rēzekne

Abstract. *In modern IT world, we are returning to the problem of low storage space. May be typically users do not see this problem and still there is more data that companies collect and should store. In this work author analyzed Huffman compression algorithm it's effectiveness, working principles and examples of usage.*

Keywords: *Compression algorithm, Huffman.*

Ievads

Hafmena saspiešanas algoritms pirmo reizi tika aprakstīts Deivida Hafmena rakstītajā publikācijā 1952. gadā[1]. Hafmena saspiešanas algoritms (dažreiz saukts par Hafmena kodu) ir plaši izplatīts algoritms, kurš paredzēts bezzudumu datu saspiešanai. Hafmena saspiešanas algoritma būtība ir kodēt tekstā biežāk sastopamos simbolus ar mazāku bitu skaitu un retāk sastopamos simbolus ar lielāku bitu skaitu. Hafmena saspiešanas algoritms veic datu saspiešanu divos etapos, pirmajā etapā tiek veidota simbolu biežumu tabula un ģenerēti prefikss kodi (prefix codes) katram simbolam no simbolu biežuma tabulas, otrajā etapā teksts tiek kodēts[2].

Prefikss kodi ir kodi kuri nekalpo kā sakuma daļa priekš citiem kodiem, šo kodu izmantošana atvieglo dekodēšanu procesu. Tāpēc ka, neviens no koda vārds nav cita koda vārda prefikss, simbols ar kursu sākas kodētais fails tiek noteikts viennozīmīgi. Kaut arī ir pagājis ilgs laiks, Hafmena saspiešanas algoritms joprojām tiek izmantots, kā pamats vai daļa no cietiem saspiešanas algoritmiem, tas tiek lietots arī JPEG algoritmā grafisko attēlu saspiešanai.

Hafmena koda piemērs

Lai attēlotu tiek izmantota tabula(sk. 1. tabula), kas satur simbolu, to cik bieži simbols sastopams teksta rindā un to bināro apzīmējumu.

1. tabula

Hafmena koda piemēra attēlošanas tabula

Simbols	a	b	c	d
Biežums	25	45	20	10
Fiksēta garuma binārais apzīmējums	000	001	010	100
Mainīga garuma binārais apzīmējums (Prefikss kodi)	10	0	110	111

Šāda teksta rinda ar fiksēta garuma bināro apzīmējumu, kur apzīmējums katram simbolam ir 3 bitus garš, kopumā aizņems 300 bitus, ja tiktu izmantota standarta ASCII kodēšanas tabula, kur katrs simbols tiek apzīmēts 8 bitus garu bināro apzīmējumu, teksta rinda kopumā aizņemtu 800 bitus. Taču ja izmantot mainīga garuma binārus apzīmējumus un piešķirt īsākus apzīmējumus tiem simboliem, kas sastopami biežāk tad teksta rinda aizņems tikai 185 bitus. Tādā veidā izmantojot mainīga garuma binārus apzīmējumus, nevis ASCII 8 bitu binārus apzīmējums izdevās ieekonomēt 76.9% atmiņas un 38.3% atmiņas, ja tiktu izmantoti fiksēta garuma 3 bitus gari binārie apzīmējumi. Protams saspiešanas efektivitāte samazināsies ja teksta rindā būs sastopami vairāk simboli un ar mazāku biežumu.

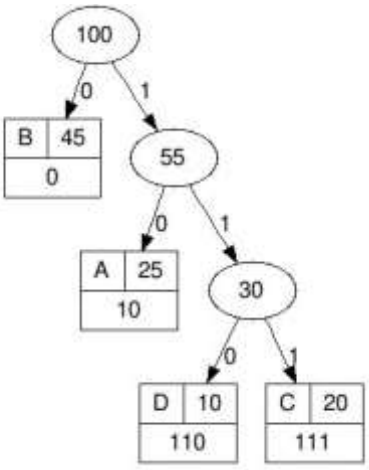
http://dx.doi.org/10.17770/het2017.21.3594

Prefikss kodi

Kā prefikss kodi tiek izskatīti tie binārie apzīmējumi, kas nav prefiksi kādam citam mainīga garuma binārajam apzīmējumam[3]. Jebkura simbolu koda teksta nokodēšana it ļoti vienkāršs process, viss ko vajag izdarīt savienot bināro apzīmējumus, piemēram izmantojot mainīga garuma bināros apzīmējumus, kas attēloti tabulā (sk. 1. tabula) simbolu virkne „abcd" binārajā formā izskatīsies šādi: 10 0 110 111. Kā jau tika minēts galvenā prefikss kodu priekšrocība ir atvieglot dekodēšanas procesu.

Viens no veidiem kā attēlot prefikss kodu piešķiršanu ir binārais koks un tā lapas ir kodējamie simboli[4]. Prefikss koda binārais apzīmējums tiek attēlots kā ceļš no binārā koka saknes līdz lapai kas satur simbolu. Veidojot šādu interpretāciju 0 apzīmē „pāreju pie kreisā meitas mezgla" un 1 apzīmē „pāreju pie labā meitas mezgla". Zemāk redzamajā attēlā(sk. 1 attēls) ir redzams binārais koks, kas izmanto prefikss kodēšanas metodi. Šāda binārā koka nolasīšana notiek no saknes elementa pārvietojoties līdz binārā koka lapām, simbolam binārais kods tiek sastādīts iegaumējot ceļu līdz binārā koka laipai, piemēram, (sk. 1. attēls)lai no saknes elementa nonākt līdz binārā koka lapai, kas satur simbola „a" vērtību no saknes elementa jāpāriet līdz „labajam meitas mezglam" un šī pāreja apzīmēta ar 1, tālāk no šī elementa „jāpāriet līdz kreisajam meitas elementam" ka apzīmēt ar 0. Saliekot klāt apzīmējumus veidojas simbola binārais kods 10

Tādā veidā var apstiprināt ja pieņemt ka C ir alfabēts ko kura tiek ņemti visi kodējamie simboli tad var teikt ka binārais koks kas satur optimālu prefikss kodu satur |C| lapas viena priekš katra simbola un |C|-1 iekšējos mezglus[5].



1.attēls **Hafmena binārais koks ar 1. tabulas datiem**

Hafmena binārā koka izveide

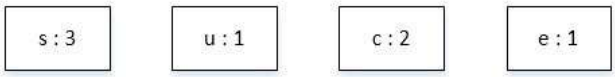
Lai izveidot Hafmena bināro koku, piemēram, teksta rindas „success", pirmajā etapā tiek izveidota simbolu biežumu tabula(sk. 1. tabula) kurā, tiek attēloti simboli, kas ir sastopami teksta rindā un to atkārtošanās biežums.

1. tabula

Simbolu biežumu tabula

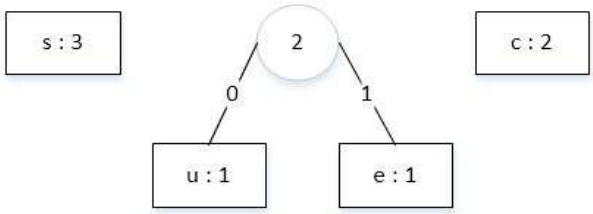
Simbols	Biežums
s	3
u	1
c	2
e	1

Tālāk tiek uzsākta Hafmena binārā koka izveide, sākumā tiek izveidotas četras binārā koka lapas kas nav saistītas savā starpā(sk. 2. attēls).



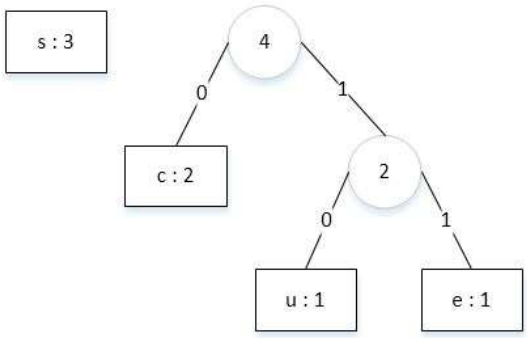
2.attēls **Hafmena binārais koks izveides pirmais solis**

Tālāk tiek savienotas divas lapas kurām ir zemākais simbolu biežums ar iekšējā mezgla palīdzību un mezglā tiek ierakstīta simbolu biežumu summa(sk. 3. attēls).

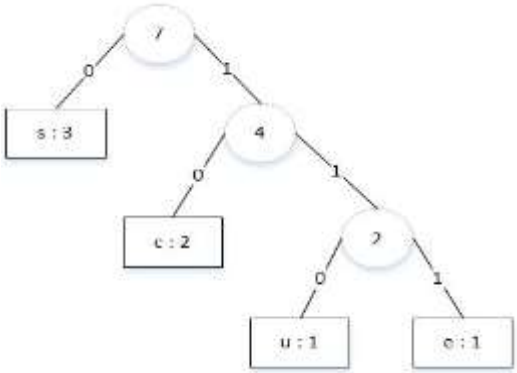


3.attēls **Hafmena binārais koks izveides otrais solis**

Nākamajos soļos atkārtojas otrais solis, atkal tiek savienotas divas lapas vai lapa un iekšējais mezgls, vai arī divi iekšējie mezgli kuriem ir mazākais simbolu biežums ar vel viena iekšējā mezgla palīdzību un tajā tiek ierakstīta apvienoto elementu simbolu biežumu summa(sk. 4. attēls). Darbība atkārtojas kamēr netiek apvienoti visas binārā koka lapas(sk. 5. attēls).



4.attēls **Hafmena binārais koks izveides trešais solis**



5.attēls **Hafmena binārais koks izveides pēdējais solis**

Secinājumi

1. Hafmena saspiešanas algoritms ir viens no pirmajiem bez zuduma saspiešanas algoritmiem, kas joprojām izmantojas mūsdienās.
2. Hafmena saspiešanas algoritms ir efektīvāks ja ar to saspiež datus kuros simboli vai elementi(pikseļi attēlos) bieži atkārtojas.
3. Izmantojot prefikss kodus var sasniegt optimālo saspiešanas pakāpi.

Summary

Huffman compression algorithm is one of first lossless data compression algorithms and become one of most popular lossless data compression algorithms. Also lot of data processing research was based on it. Main idea of algorithm is to assign smallest binary codes to characters that are more frequent and the least frequent characters get larger binary codes.

To get more optimal compressing and decrypting was easier Huffman algorithm use prefix codes. Prefix codes means that binary codes are assigned in such way that binary code assigned to one character is not prefix for other character binary code. Huffman compression algorithm can perform data compression from 20% up to 90 % of file original size, mostly compression effectiveness depends form file content.

Literatūra

1. Inna Pivkina, Discovery of Huffman Codes Sk. internetā (11.04.2017.) http://www.maa.org/press/periodicals/convergence/discovery-of-huffman-codes
2. Aashish Barnwal, Greedy Algorithms Sk. internet (31.03.2017.) http://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/
3. Huffman Coding – Base of JPEG Image Compression Sk. internet (01.04.2017.) https://www.print-driver.com/stories/huffman-coding-jpeg
4. Lydia Sinapova, Chapter 9: Huffman Codes Sk. internetā (02.04.2017.) http://faculty.simpson.edu/lydia.sinapova/www/cmsc250/LN250_Levitin/L19-Huffman.htm
5. Dave Marshall Lossless Compression Algorithms (31.03.2017.) https://users.cs.cf.ac.uk/Dave.Marshall/Multimedia/node207.html