COMPRESIA datelor

Claudiu Soroiu

Termenul de comprimare a datelor a apărut în contextul în care se manifesta o necesitate evidentă de a atinge rate mari de transfer în rețele sau de a stoca o cantitate cât mai mare de informații folosind cât mai puțin spațiu. Compresia datelor este necesară în zilele noastre și este foarte des folosită, mai ales în domeniul aplicațiilor multimedia. De-a lungul câtorva episoade vă vom prezenta cele mai des utilizate metode de compresie a datelor.

Istoria compresiei datelor a început pe la jumătatea secolului al *XX*-lea. În perioada care a urmat, cercetători ca *Claude E. Shannon*, *David Huffman*, *Newton Faller*, *Robert G. Gallager*, *Donald E. Knuth*, *Jeffrey Scott Vitter*, *Michael Burrows*, *David J. Wheeler* etc. au adus contribuții importante acestui subdomeniu al matematicii.

Introducere

Pentru a putea "pătrunde" în interesantul domeniu al compresiilor de date avem nevoie de câteva elemente utile, unele dintre ele fiind preluate din cadrul teoriei statisticii și probabilităților.

În cele ce urmează vom prezenta termenii pe care îi vom utiliza în descrierea mecanismelor necesare pentru comprimarea datelor și pentru analiza performanțelor unui algoritm de compresie.

Noțiuni teoretice

Teoria informației este acel domeniu al matematicii care se ocupă cu transmiterea, stocarea și reprezentarea informațiilor.

Compresia datelor este unul dintre subdomeniile *teoriei informației*.

Folosind un sistem digital, anumite informații, cum ar fi secvențele audio și video, care în realitate sunt secvențe continue, nu se pot reprezenta în sistemului digital la fel ca în lumea reală, fapt pentru care acestea trebuie cuantificate. Prin cuantificarea datelor anumite informații se pierd. Cuantificarea datelor constă în eliminarea sau modificarea unor secvențe de date care nu sunt percepute de către om sau care nu modifică prea mult funcționarea unor aplicații. În cazul unor secvențe de date, prin procesul de cuantificare se păstrează numai cadrele care se află la o distanță d unele de altele (distanța d nu este constantă, ea poate să varieze de la cadru la cadru în funcție de tipul de cuantificare care se aplică și de proprietățile pe care le au datele care trebuie cuantificate). Cu cât distanța dintre două cadre consecutive este mai mică, cu atât calitatea secvenței este mai bună.

Cuantificarea datelor reprezintă un prim pas în cadrul compresiei de date deoarece se reduce cantitatea de informație care trebuie stocată. Cu cât distanța dintre două cadre consecutive obținute prin cuantificare este mai mare, cu atât cantitatea de informație care trebuie stocată este mai mică. Cuantificarea datelor este necesară numai pentru datele multimedia și are ca dezavantaj faptul că se pierd date.

Un ***eveniment*** reprezintă rezultatul produs în urma efectuării unui experiment. Un *eveniment* se numește ***eveniment elementar*** dacă nu poate fi definit ca fiind o reuniune de două evenimente distincte.

Probabilitatea este o măsură definită pe un câmp (mulțime) de evenimente, p: $\Omega \rightarrow [0, 1]$. Probabilitatea $p(A)$ de realizare a unui mesaj (eveniment) A, determinată experimental, este egală cu raportul dintre numărul de cazuri favorabile evenimentului A (n_A) și numărul total de cazuri (n):

$$p(A) = n_A / n.$$

În cazul în care cunoaștem faptul că două evenimente A și B au apărut simultan de n_{AB} ori, iar evenimetul A a apărut de n_A ori și dorim să aflăm probabilitatea de apariție a evenimentului B dacă apare evenimentul A, atunci această probabilitate este egală cu raportul dintre numărul de apariții ale celor două evenimente și numărul de apariții ale evenimentului A și se notează prin $p(B/A)$:

$$p(B/A) = (n_{AB} / n) / (n_A / n) = p(A, B) / p(A),$$

unde $p(A, B)$ reprezintă probabilitatea de apariția simultană a evenimentelor A și B. Rezultă că, dacă avem două eve-

nimente A și B, atunci $p(A, B) = p(A) \cdot p(B/A) = p(B) \cdot p(A/B)$ (relația lui *Bayes*).

Dacă A_i, $1 \le i \le m$, sunt evenimentele elementare ale unui experiment probabilistic, atunci $\sum_{i=1}^{m} p(A_i)=1$. Această formulă poartă numele de *formula fundamentală a probabilităților evenimentelor elementare*.

Fie A_i, $1 \le i \le m$, evenimentele elementare ale unui experiment probabilistic și B un eveniment oarecare pentru același experiment. În aceste condiții avem:

$$p(B)=\sum_{i=1}^{m} p(A_i)\cdot p(B/A_i)$$

Se spune despre două evenimente că sunt ***echiprobabile*** dacă au aceeași probabilitate de apariție. De exemplu, evenimentele A = "*la următoarea aruncare a unei monede, aceasta va cădea cu fața în sus*" și B = "*la următoarea aruncare a unei monede, aceasta va cădea cu fața în jos*" au aceeași probabilitate de apariție: $p(A) = p(B) = 0,5 = 50\%$.

O ***sursă de informație*** este un experiment prin care se selectează un eveniment dintre n evenimente posibile, folosind un criteriu aleator.

Cantitatea de informație (h) furnizată de un eveniment A este egală cu: $h(A) = \log_2 (1/p(A)) = -\log_2 p(A)$ biți, unde $p(A)$ este probabilitatea de apariție a evenimentului A. Această mărime a fost introdusă de *Shannon*.

Bitul, pe lângă faptul că reprezintă unitatea atomică (care nu mai poate fi divizată) de măsură a informației, mai poate fi definit ca fiind o variabilă ale cărei valori sunt doar 0 și 1.

În teoria informației evenimentele sunt constituite de apariția unor simboluri (caractere).

O sursă de informație este ***discretă*** dacă între elementele generate de ea nu există continuitate. De exemplu, o funcție este discretă dacă nu are nici un punct de continuitate. De exemplu, funcția următoare este discretă:

$$f : R \to R,\ f(x)=\begin{cases}0, x\in Q\\ 1, x\in R/Q\end{cases}.$$

Despre o sursă de informație se spune că este ***fără memorie***, dacă apariția unui simbol nu depinde de nici unul dintre simbolurile care au apărut anterior. În cazul în care apariția unui simbol depinde de apariția ultimelor m simboluri anterioare, atunci despre sursa de informație se spune că este o ***sursă cu memorie*** (***Markov***) ***cu m pași***.

După cum se știe din fizică, ***entropia*** reprezintă o mărime ce caracterizează gradul de nedeterminare asupra unui sistem. În teoria informație ***entropia*** unei ***surse discrete fără memorie*** (***SDFM***) este definită ca fiind cantitatea medie de informație furnizată de apariția unui simbol a generat de sursă. Entropia unei *SDMF* se notează cu $H(S)$, unde S este o *SDMF* și are următoarea expresie:

$$H(S)=\sum_{i=1}^{m} p(A_i)\cdot \log_2 \frac{1}{p(A_i)} = -\sum_{i=1}^{m} p(A_i)\cdot \log_2 p(A_i),$$

unde m reprezintă numărul de simboluri ale sursei S, A_i ($1 \le i \le m$) reprezintă simbolurile sursei S; $p(A_i)$ reprezintă frecvența (probabilitatea) de apariție a simbolului A_i. Ca și cantitatea de informație, entropia se măsoară în biți.

Entropia maximă a unei surse este $H_{max}(S) = \log_2 m$, unde m reprezintă numărul de simboluri ale sursei S.

Se poate observa foarte ușor că entropia unei surse verifică următoarele relații:

$$0 \le H(S) \le H_{max}(S),$$
$$H(S) = H_{max}(S) \Leftrightarrow p(A_i) = 1/m,\ \forall\ 1 \le i \le m.$$

Eficiența unei surse se definește ca fiind raportul dintre entropia sursei și entropia maximă a sursei:

$$\eta_S = H(S) / H_{max}(S).$$

Redundanța unei surse reprezintă excesul de informație față de strictul necesar și este dată de diferența dintre entropia maximă a sursei și entropia "reală" a sursei:

$$R_S = H_{max}(S) - H(S).$$

Redundanța relativă a unei surse se definește ca fiind raportul dintre redundanța sursei și entropia maximă a sursei:

$$\rho_S = R(S) / H_{max}(S).$$

Algoritmi de compresie a datelor

Un compresor de date este o aplicație care, pe baza unuia sau mai multor algoritmi de compresie, diminuează spațiul necesar stocării informației utile conținute de un anumit set de date. Pentru orice compresor de date este necesară condiția de existență a cel puțin unui decompresor care, pe baza informațiilor furnizate de compresor, să poată reconstitui informația care a fost comprimată. În cazul în care nu există un decompresor, atunci datele comprimate devin inutile pentru utilizator deoarece acesta nu mai are acces la informația stocată în arhivă (o ***arhivă*** reprezintă rezultatul obținut în urma utilizării unui compresor).

Un compresor de date este format din următoarele elemente:

- una sau mai multe surse de informație;
- unul sau mai mulți algoritmi de compresie.
- una sau mai multe transformări.

Sursa de informație pentru un compresor poate fi constituită de unul sau mai multe fișiere sau de un flux de date care a fost transmis compresorului prin intemediul unei rețele. Datele arhivate urmează să fie stocate într-o arhivă care urmează să fie păstrată local sau urmează să fie transmisă mai departe.

Un model simplu de algoritm care poate sta la baza unui compresor de date este următorul:

1. *Fie S sursa de informație*
2. *Fie D destinația datelor comprimate*
3. *Fie A algoritmul de compresie utilizat*
4. *Se inițializează algoritmul de compresie*
5. *Pentru fiecare simbol i furnizat de sursa S execută*:
   5.1 *Fie R rezultatul obținut prin codificara simbolului i folosind algoritmul A*
   5.2 *Dacă rezultatul R nu este vid, transmite R către D*
6. *Transmite algoritmului A faptul că s-a încheiat recep-*

ționarea de simboluri de la sursa S și fie R ultimul rezultat reținut de algoritmul de compresie
7. Dacă rezultatul R nu este vid, transmite R către D

Semnificația iterației 5.2 este următoarea: în cazul în care algorimtul de compresie nu codifică simbolurile individual ci codifică un șir de simboluri folosind un singur identificator, atunci rezultatul obținut la iterația 5.1 este vid atât timp cât nu s-a ajuns la numărul de simboluri necesare pentru furnizarea unui rezultat. Pentru a optimiza această clasă de algorimi de compresie, propozițiile de la iterația 5, respectiv iterația 5.1 pot fi înlocuite cu "*pentru fiecare set de n simboluri furnizate de sursa S*", respectiv "*fie R rezultatul obținut prin codificara setului de n simboluri folosind algoritmul A*".

La un moment dat, algoritmul de compresie așteaptă apariția unui număr de simboluri pentru a putea furniza un rezultat, dar sursa de informație nu mai emite simboluri. Datorită acestui fapt, pentru a nu se pierde informație, este necesar ca algoritmului de compresie să i se comunice terminarea compresiei și acesta la rândul său trebuie să furnizeze ultimul rezultat obținut pe baza simbolurilor pe care nu le-a codificat. În cazul formatelor de date multimedia, o transformare poate fi identificată cu un cuantificator și poate duce la pierderea anumitor date (de exemplu, un punct gri pe o imagine complet albă poate deveni alb deoarece nu este perceput de ochiul uman).

De obicei, compresoarele de date avansate (cum sunt aplicațiile *RAR*, *ACE*, *BZIP* etc.) folosesc o înlănțuire de algoritmi similari celui prezentat anterior cu precizarea că în afară de primul algoritm din înlănțuire, sursa de informații pentru un alt algoritm este constituită de rezultatul furnizat de algoritmul anterior.

Una dintre cele mai importante întrebări care se pun cu privire la compresie datelor este: "*Cum să facem ca datele să se comprime mai bine?*"

Pentru anumite seturi de date particulare s-au găsit răspunsuri, dar despre aceasta vom vorbi în detaliu în alt episod. Pentru a face datele "*mai comprimabile*" se aplică transformări. Pentru aplicarea unei transformări, în algoritmul prezentat anterior se înlocuiește algoritmul de compresie *A* cu transformarea *T* și se folosește în combinație cu algoritmii de compresie.

În cele ce urmează vom prezenta câțiva algoritmi folosiți pentru comprimarea datelor.

Există două categorii principale de algoritmi pentru compresia datelor:

- algoritmi de compresie fără pierderi de date
  - algoritmi entropici de compresie a datelor (algoritmi care elimină redundanța din cadrul seturilor de date)
    - algoritmul *Huffman*
    - algoritmi de *compresie aritmetică* (acesta este cel mai performant algoritm entropic cunoscut).
  - algoritmi bazați pe dicționare
    - *Lempel Ziff* 77 (*LZ*77), *Lempel Ziff* 78 (*LZ*78)
    - variante ale algoritmilor din categoria *LZ*
  - transformări fără pierderi de date
    - *Run Length Encoding* (*RLE*)
    - *Burrow-Wheeler Transform* (*BWT*)
    - *transformarea delta*
- algoritmi de compresie cu pierderi de date
  - algoritmi folosiți pentru compresia audio care se bazează pe proprietățile undelor sonore și perceperea lor de către om;
  - algoritmi utilizați pentru compresia imaginilor digitale
  - algoritmi utilizați pentru compresia datelor video.

Algoritmii de compresie aparținând celei de-a doua clase (cea cu pierderi de date) se folosesc împreună cu cei din prima pentru atingerea unor rate mari de compresie.

Algoritmul Huffman

Ideea care stă la baza acestui algoritm este aceea de a atașa coduri de lungime mică (exprimată în biți) simbolurilor furnizate de o sursă de informație și a căror frecvență de apariție este mare, și de a atașa coduri de lungime mai mare simbolurilor a căror frecvență de apariție este mică.

În concluzie, folosind acest algoritm se reduce lungimea medie a codurilor folosite pentru a reprezenta simbolurile alfabetului.

Codurile generate de acest algoritm sunt optime în cazul în care toate frecvențele de apariție ale simbolurilor sunt puteri întregi ale lui 1/2.

Algoritmul lui *Huffman* constă în construirea unui arbore binar în care fiecare nod are fie nici un fiu, fie exact 2 fii, în care frunzele reprezintă simbolurile generate de sursa de informație și au atașat un cost egal cu frecvența de apariție a acestora, iar nodurile interne au atașat un cost egal cu suma costurilor fiilor. Prin urmare, rădăcina arborelui are costul 1.

Atunci când se cunosc frecvențele de apariție ale simbolurilor generate de o sursă de informație *S* și faptul că sursa a generat în total *m* simboluri distincte ale unui alfabet, construcția arborelui binar devine foarte simplă.

Primul pas constă în construirea unei păduri formată din *m* arbori. Fiecare arbore din pădure are un singur element care conține un simbol și frecvența lui de apariție. Toți cei *m* arbori din pădure sunt distincți (nu există doi arbori al cărui element să conțină același simbol).

În continuare, la fiecare pas se elimină câte doi arbori din pădure pentru care costul rădăcinii este minim. Acești doi arbori vor fi subarborii unui nod al cărui cost va fi egal cu suma frecvențelor celor doi arbori eliminați din pădure. Arborele nou creat se adaugă în pădure.

Algoritmul se termină în momentul în care în pădure mai rămâne un singur arbore, adică după efectuarea a *m* - 1 iterații.

Din analiza acestui algoritm rezultă faptul că drumul de la rădăcină până la frunzele care au costul mai mic este

mai mare decât drumul de la rădăcină până la frunzele care au costul mai mare.

În figura 1 se poate observa modul de construire al arborelui pentru șirul "*To be or not to be*" generat de o sursă de informație *S*. Șirul are lungimea $n = 18$ și $m = 8$ simboluri distincte. Simbolurile distincte sunt "*b*", "*e*", "*n*", "*o*", "*r*","*t*","*T*" respectiv "_"(spațiu) și au frecvențele de apariție 2/18, 2/18, 1/18, 4/18, 1/18, 2/18, 1/18, 5/18.

Datorită faptului că toate fracțiile au același numitor, îi vom elimina, deci la sfârșitul algoritmului rădăcina arborelui obținut va avea costul *n*, unde *n* este numărul total de simboluri generate de sursa *S*. Eliminând numitorul comun se optimizează execuția unui program care implementează acest algoritm datorită faptului că operațiile cu numere întregi sunt executate mai rapid decât operațiile cu numere reale.

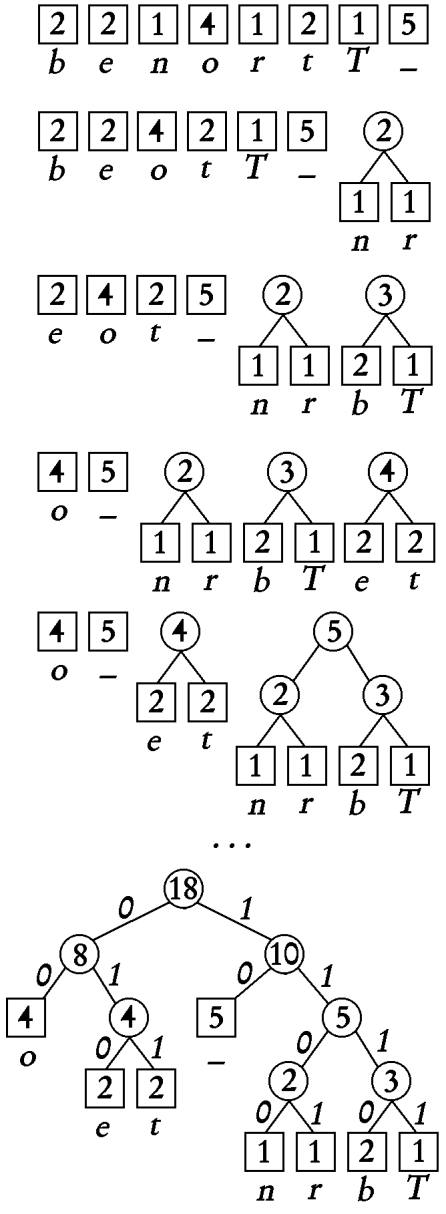

Figura 1

Lungimea codului pentru un simbol generat de sursa *S* va fi egală cu lungimea drumului de la frunza corespunzătoare unui simbol până la rădăcina arborelui obținut folosind algoritmul lui *Huffman*.

În arborele obținut în final, vom eticheta fiecare muchie dintre două noduri cu 0 dacă nodul fiu se află în stânga părintelui și cu 1 dacă nodul fiu se află în dreapta. Pentru a codifica simbolurile generate de sursa *S*, se va parcurge șirul generat și pentru fiecare simbol se va transmite la o ieșire *E* drumul de la rădăcină până la frunza corespunzătoare simbolului curent astfel: se transmit la ieșire etichetele muchiilor întâlnite pe drum.

Simbolurile din șirul dat anterior ca exemplu au codurile:

- "b" 1110
- "e" 010
- "n" 1100
- "o" 00
- "r" 1101
- "t" 011
- "T" 1111
- "_" 10

Având codurile simbolurilor, șirul din exemplu se codifică astfel:

111100101110010100011011011000001110011001011100 10

Lungimea codificării exemplului este de 50 biți (7 *bytes*). Dacă sursa de informație folosește alfabetul *ASCII*, atunci lungimea în bytes a mesajului este 18, deci folosind această metodă se diminuează spațiul utilizat cu 11 bytes. Pentru a putea decodifica acest mesaj, trebuie transmis și arborele creat, sau informațiile necesare pentru a putea fi construit. Dacă aceste informații lipsesc, nu se poate reconstitui mesajul inițial.

Arborele de codificare obținut cu agoritmul lui *Huffman* poartă numele de *dicționar*. *Dicționarul* este un termen general după cum vom vedea în cazul altor tehnici de compresie. ***Dicționarul*** desemnează orice structură de date (sau informațiile necesare construirii ei) care se atașează șirului comprimat pentru ca programul de decompresie să poată reconstitui informația inițială.

În cazul acestui algoritm o reprezentare eficientă a arborelui are dimensiunea de $2 \cdot m$ biți.

Există mai multe metode de a transmite dicționarul. Cel mai bine este ca acesta să fie și el comprimat pentru a elimina o parte din redundanță.

Pentru a determina care este redundanța rămasă după compresie, trebuie mai întâi calculată entropia șirului de intrare care este $H = 2{,}747167$ biți, deci ar fi trebuit să se folosească în medie *H* biți pentru a codifica fiecare simbol. Acest lucru nu este posibili deoarece algoritmul lui *Huffman* atașează un număr întreg de biți pentru fiecare caracter în parte. Lungimea medie a codului unui simbol este 2,777778 biți deci pentru fiecare simbol se transmite o redundanță de aproximativ 0,23 biți la care se adaugă redundanța dată de transmiterea dicționarului.

Entropia maximă a oricărui șir obținut din simbolurile alfabetului *ASCII* este $H_{max} = \log_2 m = \log_2 256 = 8$ biți, unde $m = 256$ este mărimea alfabetului, deci orice simbol *ASCII* se poate reprezenta folosind maxim 8 biți.

În acest caz, lungimea medie a unui cod este mai mică decât entropia maximă, deci șirul se poate comprima.

În cazul în care frecvențele de apariție ale simbolurilor sunt egale, iar numărul de simboluri distincte generate de o sursă *S* este egal cu numărul de simboluri ale alfabetului, atunci șirul simbolurilor nu se poate comprima folosind această metodă în felul cum a fost prezentată mai sus, deoarece lungimea medie a codului unui simbol devine egală cu entropia maximă la care se mai adaugă și informația conținută de dicționar; în concluzie, cantitatea de informație transmisă va fi mai mare decât cea inițială. În acest caz, se folosesc variante mai complexe ale algoritmului.

Variante ale algoritmului lui Huffman

Există trei variante des aplicate ale acestui algoritm:
- algoritmul *Huffman static*;
- algoritmul *Huffman semi-static*;
- algoritmul *Huffman dinamic*.

Diferențele dintre cele trei variante sunt următoarele:
- în cazul variantei statice, atât compresorul, cât și decompresorul dețin același arbore de compresie calculat pe baza unor frecvențe fixe de apariție și nu mai este necesară calcularea unui arbore nou și nici transmiterea acestuia. Dezavantajul acestei metode este că, dacă frecvențele de apariție ale simbolurilor generate de o sursă diferă foarte mult de cele fixe utilizate, atunci s-ar putea ca pentru simboluri cu frecvență mare de apariție să fie transmise coduri foarte lungi și astfel cantitatea de informație comprimată poate să depășească cu mult cantitatea de informație care a fost generată de o sursă.
- varianta semi-statică utilizează algoritmul de construire a arborelui de compresie prezentat anterior. Are ca dezavantaj faptul că șirul simbolurilor generate de o sursă de informație trebuie parcurse de două ori, o dată pentru calcularea frecvențelor necesare construirii arborelui și o dată pentru codificarea șirului simbolurilor; în cazul în care simbolurile au aproximativ aceeași frecvență de apariție, algoritmul nu oferă o compresie bună. Parcurgerea de două ori a șirului simbolurilor generate de o sursă de informație este un incovenient deoarece acesta trebuie stocat, iar dimensiunea datelor care trebuie comprimate, în zilele noastre, este foarte mare și calculatoarele personale (de cele mai multe ori) nu dețin resursele necesare stocării datelor.
- pentru varianta *dinamică* există doi algoritmi performanți: ***FGK*** *(**F**aller*, ***G**allager*, *K**n**uth)* și ***V*** *(**V**itter)*. Aceștia au în comun faptul că arborele se construiește dinamic pe măsură ce o sursă de informație generează simboluri, deci este necesară o singură parcurgere a șirului simbolurilor și nu este necesară stocarea lor. Doar o mică parte dintre ele sunt stocate cu scopul de a optimiza procesul de compresie. Rata de compresie a acestor doi algoritmi variază. În anumite cazuri rezultatele obținute sunt cu mult mai bune decât cele date de varianta statică, dar în cazul cel mai rău pentru varianta statică, rezultatele variantelor dinamice sunt optime de cele mai multe ori.

Primele două variante nu mai necesită explicații suplimentare, așadar, în continuare, vom prezenta în detaliu varianta dinamică ***FGK***, varianta dinamică ***Vitter*** fiind similară.

Algoritmul FGK

Algoritmul *FGK* precum și algoritmul *V* se bazează pe *proprietatea fraților* enunțată și demonstrată de *Gallager* în anul 1978:

"*Un arbore binar cu p frunze care au greutăți (în cazul de față frecvențe) nenegative este arbore Huffman dacă și numai dacă următoarele afirmații sunt adevărate:*
- *cele p frunze au greutățile nenegative w_1, w_2, ..., w_p și greutatea fiecărui nod intern este egală cu suma greutăților celor doi fii;*
- *nodurilor interne li se pot atașa numere de ordine în ordinea crescătoare a greutăților, astfel încât nodurile cu numerele $2 \cdot j - 1$ și $2 \cdot j$ să fie frați pentru $1 \le j \le p - 1$ și părintele lor comun să aibă un număr de ordine mai mare.*"

Dacă un arbore este construit pe baza algoritmului prezentat la secțiunea anterioară, atunci acesta este un arbore *Huffman* și respectă *proprietatea fraților*.

La început vom avea un arbore *A* care va conține un singur nod a cărui pondere (greutate) este 0 sau 1 în funcție de implementarea folosită. Acest nod ține locul tuturor simbolurilor care pot fi generate de o sursă de informație *S* și care nu au fost generate încă până la un pas *k*. Pe acest nod și îl vom numi ***nodul zero***. Vom considera că greutatea acestui nod este 0, deci simbolurile care nu au fost încă generate de sursa *S* au apărut de 0 ori (sau în 0% din cazuri). În concluzie, după un pas *k*, acest nod este frunză, iar ponderea rădăcinii este egală cu *k* (sau cu 1 în cazul în care se folosesc frecvențele de apariție ale simbolurilor generate până atunci).

La început arborele *A*, care conține doar nodul zero, este un arbore *Huffman*, deci respectă *proprietatea fraților*.

Considerăm că la al *k*-lea simbol generat de o sursă avem un arbore *Huffman*. Sursa *S* generează al $(k + 1)$-lea simbol care trebuie codificat.

În cazul în care simbolul nu a mai fost încă generat, se va transmite la ieșire codul nodului *zero* și simbolul care tocmai a fost generat. Nodul *zero* va fi înlocuit cu un nod cu ponderea 1 și ai cărui fii vor fi nodul *zero* și un alt nod corespunzător simbolului nou apărut, a cărui pondere va fi tot 1.

În cazul în care sursa a emis un simbol care a mai fost generat, la ieșire se transmite codul acestuia și se incrementează ponderea nodului corespunzător simbolului generat.

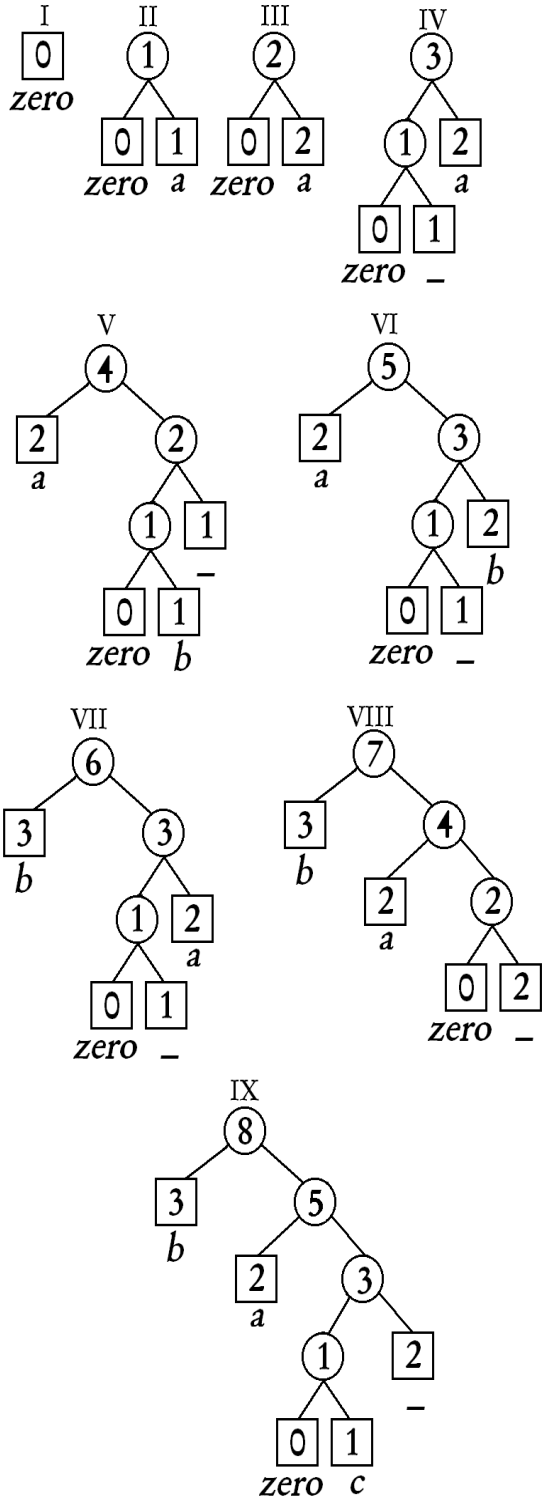

Figura 2

Codurile simbolurilor din arbore sunt construite la fel ca în cazul variantelor *semi-statice*, adică sunt date de drumul parcurs de la rădăcină până la frunza corespunzătoare simbolului care trebuie codificat.

În continuare trebuie actualizate ponderile celorlalte noduri interne deaorece după adăugarea unui nod nu mai este respectată *proprietatea fraților* fiindcă ponderea rădăcinii nu mai este egală cu suma ponderilor frunzelor subarborilor ei.

Aceste două cazuri de apariție ale unui simbol sunt tratate în mod identic de algoritmii *FGK* și *V*.

În continuare vom prezenta modul în care se actualizează ponderile nodurilor interne în cadrul algoritmului *FGK*.

Pentru a menține proprietatea fraților, trebuie parcurs drumul de la frunza actualizată ultima dată până la rădăcină. Ponderea fiecărui nod din drum trebuie incrementată cu 1 pentru a fi respectată prima condiție a proprietății fraților, iar pentru a doua condiție, la fiecare pas trebuie interschimbat nodul curent cu un nod din arbore care are aceeași pondere cu cea a nodului curent și care are cel mai mare număr de ordine. Părintele nodului curent devine părintele noului nod și invers. Trebuie avut în vedere faptul că un nod nu se poate interschimba cu un strămoș de-al său.

În figura alăturată este prezentat modul de construcție și actualizare a arborelui folosind algoritmul *FGK*, pentru șirul "*aa bbb c*", unde simbolul *spațiu* este reprezentat prin "_".

La fiecare pas, arborii din figură sunt arbori *Huffman*, deci respectă și a doua condiție a proprietății fraților. Pentru a verifica acest lucru se numerotează nodurile din arbore în ordine creascătoare începând de la ultimul nivel până la primul și de la stânga la dreapta.

Algoritmul V

Algoritmul *V* diferă de algoritmul *FGK* prin faptul că, în momentul în care se realizează actualizarea arborelui, se încearcă minimizarea expresiilor **SUM**(l_i) și **MAX**(l_i), unde **SUM**(l_i) reprezintă suma lungimilor drumurilor de la rădăcină până la frunze, iar **MAX**(l_i) reprezintă lungimea drumului de la rădăcină până la cea mai îndepărtată frunză. Cu alte cuvinte, se încearcă minimizarea adâncimii maxime a arborelui și a lungimii drumului extern.

Complexitatea celor doi algoritmi este aceeași, dar algoritmul *V* este mai performant în cazul în care probabilitățile de apariție ale simbolurilor sunt aproximativ aceleași.

Jeffrey S. Vitter a demonstrat faptul că în cel mai rău caz, algoritmul *V* transmite la fiecare simbol un bit în plus față de metoda *semi-statică*, în timp ce algoritmul *FGK* transmite în cel mai rău caz de două ori mai mulți biți pe simbol relativ la metoda *semi-statică*.

În episodul următor...

Algoritmii de compresie *Huffman* atașează câte un cod fiecărui simbol generat de o sursă *S*. În numărul următor vom prezenta o metodă de compresie care codifică un șir de simboluri cu ajutorul unui singur cod. Este vorba despre cel mai performat algoritm entropic de compresie și anume algoritmul de *compresie aritmetică*.

Claudiu Soroiu este redactor al GInfo. Poate fi contactat prin e-mail la adresa csoroiu@yahoo.com.