CODIFICARE SHANNON-FANO

In acest algoritm se obtin cuvinte de cod pentru compresie / decompresie utilizind coduri prefix (codarea fiind simbol cu simbol)

Se porneste de la urmatoarea observatie:
Informatia transmisa este maxima daca bitii ce se transmit au probabilitatea

$$p(0) = p(1) = 1/2$$

Se incearca sa se realizeze acest lucru in felul urmator:

Fie $S = \{ s_1, s_2, \ldots s_n \}$ multimea simbolurilor alfabetului sursei ordonata dupa probabilitati
$P = \{ p_1, p_2, \ldots, p_n \}$.probabilitatile de aparitie ale acestor simbolurilor

Se imparte aceasta multime in doua submultimi S1 si S2 avind probabilitatile de aparitie cit mai apropiate intre ele (la primul pas de 1/2) atribuind unei multimi bitul 0 si celeilalte bitul 1. Se repeta aceasta impartire pentru fiecare din multimile S1 si S2 si asa mai departe. Algoritmul continua in acest fel recursiv pina cind multimile au un singur element.

O descriere algoritmica este urmatoarea:

```
Shanon_Fano (multime S)        // multimea de elemente (si probabilitatile acestora)
{
   N=cardinal(S);
   if(N==1)
      return;                                   // final apeluri recursive

   afla k ∈ [1, N) astfel incit
     |Σ_{i=1}^{k} P[i] − Σ_{i=k+1}^{N} P[i]|  sa fie minima   // imparte lista cu simb. in doua subliste
                                                // cu probabilit cit mai egale

    S1 = { S[1],    S[2],...    S[k]  };       // cele doua subliste
    S2 = { S[k+1], S[k+2],...,S[N] };

    Shanon_Fano(S1);                            // apel recursiv, imparte cele doua subliste
    Shanon_Fano(S2);
}
```

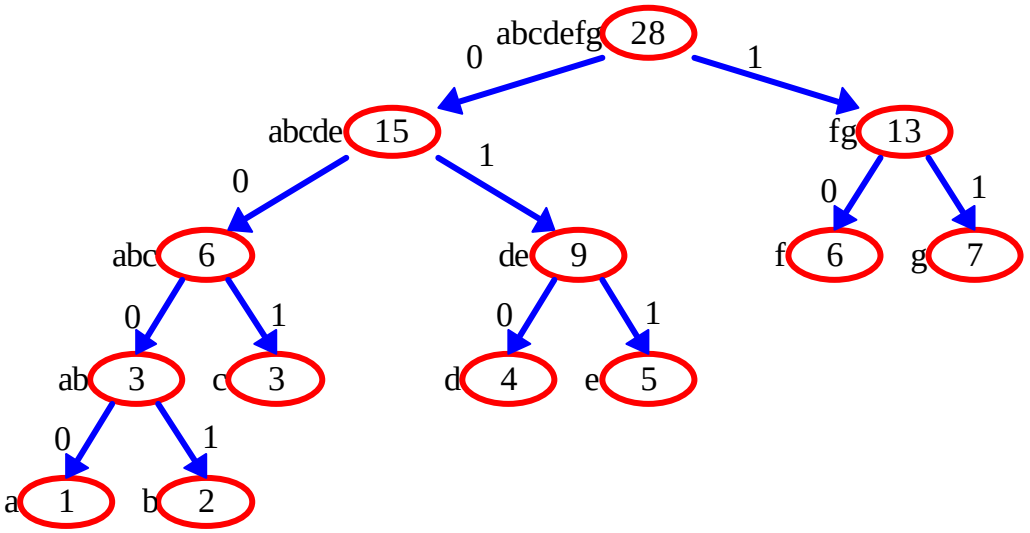

Figura 2.1

Codul asociat fiecarui simbol este format din 0 sau 1 dupa cum la fiecare impartire simbolul a facut parte din o multime sau alta. Deoarece lista poate fi uneori impartita in mai multe feluri in subliste cu probabilitati cit mai apropiate codul Shanon-Fano pentru o distributie de probabilitati nu este unic.

Un exemplu de functionare a algoritmului este dat in figura 2.1 in care am considerat simbolurile alfabetului sursei literele a,b,c,d,e,f,g cu frecventele de aparitie 1,2,3,4,5,6,7. Arborele rezultat este cel din figura.

Pentru obtinerea codului unei frunze se parcurge arborele dinspre frunza spre radacina punind de fiecare data in stiva 0 sau 1 dupa cum sintem fiul sting sau drept al parintelui. In final extragind din stiva caracterele depuse obtinem codul asociat simbolului respectiv.

CODIFICARE HUFFMAN STATICA

Si in acest algoritm se porneste de la

$S = \{ s_1, s_2, ...s_n \}$ multimea simbolurilor alfabetului sursei ordonata crescator dupa probabilitati
$P = \{ p_1, p_2, ..., p_n \}$.probabilitatile de aparitie ale acestor simbolurilor

si se incearca atribuirea bitilor 0 si 1 cu o probabilitate cit mai egala. In acest caz se combina primele doua simboluri in lista rezultind un simbol (nod) avind probabilitatea egala cu suma care va inlocui cele doua simboluri anterioare. Algoritmul continua pina ajungem la un singur simbol (nod) de probabilitate 1.

Codul generat de algoritm este un cod prefix ceea ce permite punerea sa in corespondenta cu un arbore binar pe baza caruia se va face si codarea respectiv decodarea.

Un algoritm practic de generare a arborelui este

```
Algoritm_Huffman (multime S, P)          // multimea S si probabilitatile P
{
    pentru fiecare simbol
        genereaza cite un arbore cu
            probabilit = P[i];
            fiu_sting = fiu_drept = NULL;        // initial fara fii
        si insereaza-l in lista arborilor

    cit timp ( sint mai multi arbori )
            {
            cauta arborele A cu probabilitatea minima
            cauta arborele B cu urmatoarea probabilitate
            construieste un arbore C cu
                probabilitatea = P(A) + P(B)
                fiu_sting  = A;
                fiu_drept = B;
            sterge A si B din lista arborilor;
            insereaza C in lista arborilor
            }
}
```

In final codul rezultat pentru fiecare frunza este dat de calea dinspre acea frunza spre radacina. Si in acest caz se foloseste o stiva temporara pentru a inversa ordinea bitilor.

Si acest algoritm a fost descris in termeni de probabilitati ale simbolurilor dar el se foloseste aproape exclusiv referitor la frecventele de aparitie ale simbolurilor (nu intereseaza practic probabilitatea absoluta ci doar cea relativa intre doua noduri - interne sau frunze - iar aceasta se obtine mai simplu din frecvente de aparitie. Aceasta observatie va fi importanta in cazul versiunilor dinamice unde actualizarea frecventelor de aparitie este mult mai usoara decit calcularea probabilitatilor)

In cazul algoritmului Huffman arborele este dezvoltat de la frunze spre radacina avind de a face cu o abordare "**bottom-up**" a constructiei acestuia.

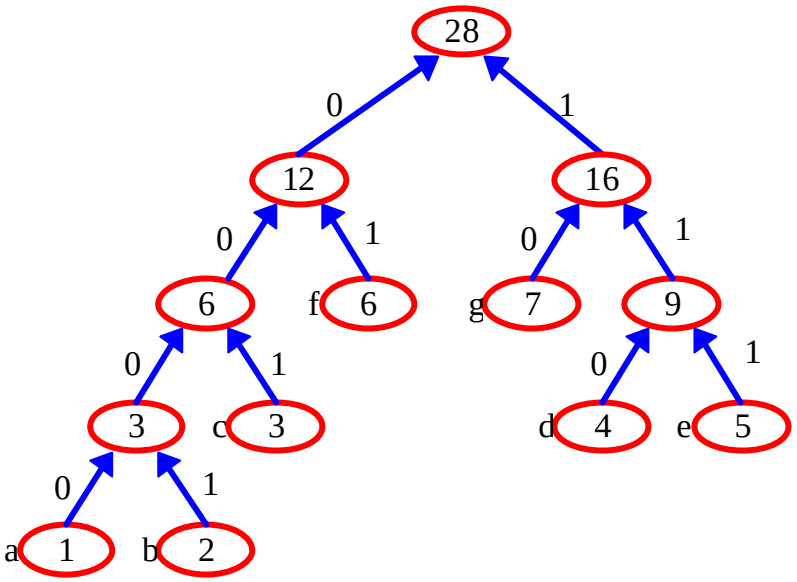

Figura 2.2

Exemplul considerat in cazul codificarii Shannon-Fano este reluat in figura 2.2 pentru cazul codificarii Huffman. Codurile asociate simbolurilor sint a = 0000, b = 0001, c = 001, d = 110, e = 111, f = 01, g =10.

Algoritmul lui Huffman in principiu nu impune nici o regula de decizie in cazul in care avem mai multe noduri cu aceeasi pondere, putind rezulta astfel diferite moduri de codare pornind de la acelasi set de frecvente de aparitie. Cea mai simpla implementare insereaza noul nod in capul listei arborilor astfel incit el va fi ales primul intre cei cu ponderi egale cu el (chiar eventual frunze). In acest mod exista tendinta ca arborele sa fie dezechilibrat.

Schwartz propune o varianta a algoritmului in care nodul rezultat se plaseaza in coada listei arborilor si se va alege ultimul in cazul in care avem mai multe noduri cu ponderi egale cu a lui. Codul astfel obtinut are valori minime in ceea ce priveste lungimea maxima a cuvintelor de cod (max l_i) si lungimea totala a cuvintelor de cod (Σl_i). Aceste caracteristici se justifica si intuitiv deoarece arborele realizat este mai echilibrat.

Algoritmul lui Huffman produce un cod optim in sensul obtinerii unei lungimi medii minime a cuvintelor de cod in raport cu orice alt cod care codifica sursa simbol cu simbol.

Algoritmul, asa cum a fost descris pina aici, se incadreaza in cadrul **metodelor semistatice**. Astfel arborele este construit o singura data la inceputul transmisiei (pe baza frecventei de aparitie a caracterelor in cadrul blocului de date) si apoi folosit pentru codificarea fiecarui simbol al blocului de date. De aici decurg principalele sale dezavantaje:

- necesita trecerea de doua ori peste datele de codificat ceea ce scade performantele sale de viteza.
- necesita transmiterea corespondentei intre mesaje si cuvinte de cod (eventual a setului frecventelor de aparitie) ceea ce supraincarca datele de transmis si determina scaderea raportului de compresie global. Acest dezavantaj poate fi redus daca se accepta un cod suboptim si se folosesc variante predefinite ale arborilor ca si carti de coduri, caz in care emitatorul si receptorul aleg codul care se foloseste la un moment dat. Experimentele au aratat ca lungimea codului obtinut prin aceasta metoda este cu cel mult 6.5% mai mare decit cea data de algoritmul Huffman.

Pentru eliminarea ambelor dezavantaje prezentate anterior se folosesc versiuni dinamice ale codurilor Huffman.