Structuri de date arborescente

Arbori de compresie Huffman

În analiza algoritmilor pe care i-am studiat până acum prioritară a fost complexitatea timp. Acum ne punem problema elaborării unor algoritmi care să micşoreze spaţiul necesar memorării unui fişier. Tehnicile de compresie sunt utile pentru fişiere text, în care anumite caractere apar cu o frecvenţă mai mare decât altele, pentru fişiere ce codifică imagini sau sunt reprezentări digitale ale sunetelor ori ale unor semnale analogice, ce pot conţine numeroase motive care se repetă. Chiar dacă astăzi capacitatea dispozitivelor de memorare a crescut foarte mult, algoritmii de compresie a fişierelor rămân foarte importanţi, datorită volumului tot mai mare de informaţii ce trebuie stocate. În plus, compresia este deosebit de utilă în comunicaţii, transmiterea informaţiilor fiind mult mai costisitoare decât prelucrarea lor.

Una dintre cele mai răspândite tehnici de compresie a fişierelor text, care, în funcţie de caracteristicile fişierului ce urmează a fi comprimat, conduce la reducerea spaţiului de memorie necesar cu 20%-90%, a fost descoperită de D. Huffman în 1952 şi poartă numele de *codificare (cod) Huffman*. În loc de a utiliza un cod în care fiecare caracter să fie reprezentat pe 7 sau pe 8 biţi (lungime fixă), se utilizează un cod mai scurt pentru caracterele care sunt mai frecvente şi coduri mai lungi pentru cele care apar mai rar.

Să presupunem că avem un fişier de 100.000 de caractere din alfabetul {a,b,c,d,e,f}, pe care dorim să-l memorăm cât mai compact. Dacă am folosi un cod de lungime fixă, pentru cele 6 caractere, ar fi necesari câte 3 biţi. De exemplu, pentru codul:

	a	b	c	d	e	f
cod fix	000	001	010	011	100	101

ar fi necesari în total 300.000 biţi.

Să presupunem acum că frecvenţele cu care apar în text cele 6 caractere sunt:

	a	b	c	d	e	f
frecvenţă	45	13	12	16	9	5

Considerând următorul cod de lungime variabilă :

	a	b	c	d	e	f
cod variabil	0	101	100	111	1101	1100

ar fi necesari doar 224.000 biţi (deci o reducere a spaţiului de memorie cu aproximativ 25%).

Problema se reduce deci la a asocia fiecărui caracter un cod binar, în funcţie de frecvenţă, astfel încât să fie posibilă decodificarea fişierului comprimat, fără ambiguităţi. De exemplu, dacă am fi codificat a cu `1001` şi b cu `100101`, când citim în fişierul comprimat secvenţa `1001` nu putem decide dacă este vorba de caracterul a sau de o parte a codului caracterului b.

Ideea de a folosi separatori între codurile caracterelor pentru a nu crea ambiguităţi ar conduce la mărirea dimensiunii codificării.

Pentru a evita ambiguităţile este necesar ca nici un cod de caracter să nu fie prefix al unui cod asociat al unui caracter (un astfel de cod se numeşte *cod prefix*).

Codul Huffman

D. Huffman a elaborat un algoritm *Greedy* care construieşte un cod prefix optimal, denumit cod Huffman. Prima etapă în construcţia codului Huffman este calcularea numărului de apariţii ale fiecărui caracter în text. Există situaţii în care putem utiliza frecvenţele standard de apariţie a caracterelor, calculate în funcţie de limbă sau de specificul textului.

Fie C={c_1,c_2,...,c_n} mulţimea caracterelor dintr-un text, iar f_1,f_2,...,f_n, respectiv, numărul lor de apariţii. Dacă l_i ar fi lungimea şirului ce codifică simbolul c_i, atunci lungimea totală a reprezentării ar fi :

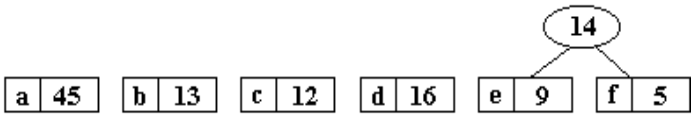
$$L = \sum_{i=1}^{n} l_i \cdot f_i$$

Scopul nostru este de a construi un cod prefix care să minimizeze această expresie. Pentru aceasta, construim un arbore binar complet în manieră *bottom-up* astfel :

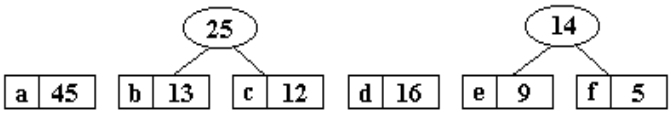
1. Iniţial, considerăm o partiţie a mulţimii C={ {c_1,f_1},{c_2,f_2}, ..., {c_n,f_n}}, reprezentată printr-o pădure de arbori formaţi dintr-un singur nod.
2. Pentru a obţine arborele final, se execută `n-1` operaţii de unificare. Unificarea a doi arbori A_1 şi A_2 constă în obţinerea unui arbore A, al cărui subarbore stâng este A_1, subarbore drept A_2, iar frecvenţa rădăcinii lui A este suma frecvenţelor rădăcinilor celor doi arbori. La fiecare pas unificăm 2 arbori ale căror rădăcini au frecvenţele cele mai mici.

De exemplu, arborele Huffman asociat caracterelor {a,b,c,d,e,f} cu frecvenţele {45,13,12,16,9,5} se construieşte pornind de la cele cinci noduri din figura 1:

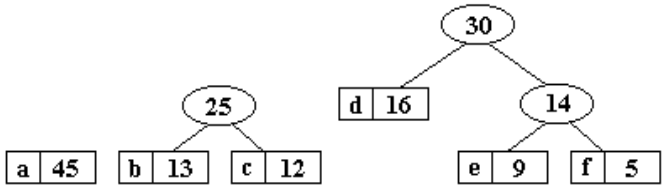
Pas 1: Unific arborii corespunzători lui e şi f, deoarece au frecvenţele cele mai mici:



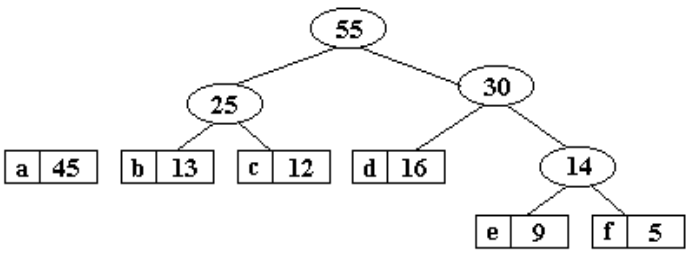
Pas 2: Unific arborii corespunzători lui b şi c:



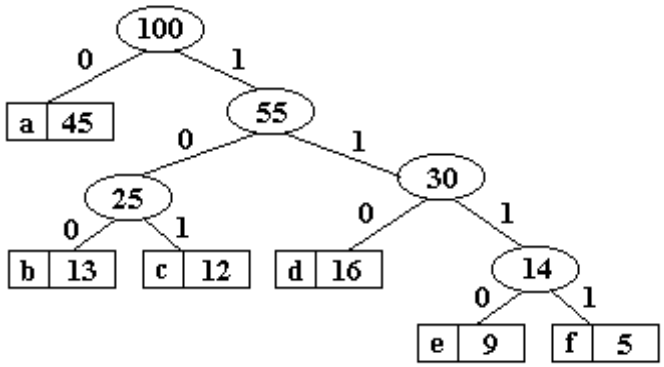
Pas 3: Unific arborele corespunzător lui d şi arborele obţinut la primul pas, cu rădăcina ce are frecvenţa 14:



Pas 4: Unific arborii ce au frecvenţele 25 şi 30.



Pas 5: Unificând ultimii doi arbori, obțin arborele Huffman asociat acestui set de caractere cu frecvențele specificate inițial.



Nodurile terminale vor conține un caracter şi frecvența corespunzătoare caracterului; nodurile interioare conțin suma frecvențelor caracterelor corespunzătoare nodurilor terminale din subarbori.

Arborele Huffman obținut va permite asocierea unei codificări binare fiecărui caracter. Caracterele fiind frunze în arborele obținut, se va asocia pentru fiecare deplasare la stânga pe drumul de la rădăcină la nodul terminal corespunzător caracterului un 0, iar pentru fiecare deplasare la dreapta un 1.

Obținem codurile :

	a	b	c	d	e	f
cod	0	100	101	110	1110	1111

Observații

– Caracterele care apar cel mai frecvent sunt mai aproape de rădăcină şi astfel lungimea codificării va avea un număr mai mic de biți.

– La fiecare pas am selectat cele mai mici două frecvențe, pentru a unifica arborii corespunzători. Pentru extragerea eficientă a minimului vom folosi un min-heap. Astfel timpul de execuție pentru operațiile de extragere minim, inserare şi ştergere va fi, în cazul cel mai defavorabil, de *O(log n).*

Algoritm de construcție a arborelui Huffman

Pas 1. Inițializare :

- fiecare caracter reprezintă un arbore format dintr-un singur nod;

- organizăm caracterele ca un min-heap, în funcție de frecvențele de apariție;

Pas 2. Se repetă de n-1 ori :

- extrage succesiv X şi Y, două elemente din heap

- unifică arborii X şi Y :

- creează Z un nou nod ce va fi rădăcina arborelui

- Z->st := X

- Z->dr := Y

- Z->frecv := X->frecv+Y->frecv

- inserează Z în heap;

Pas 3. Singurul nod rămas în heap este rădăcina arborelui Huffman. Se generează codurile caracterelor, parcurgând arborele Huffman.

Analiza complexităţii

Iniţializarea heapului este liniară. Pasul 2 se repetă de n-1 ori şi presupune două operaţii de extragere a minimului dintr-un heap şi de inserare a unui element în heap, care au timpul de execuţie de O(log n). Deci complexitatea algoritmului de construcţie a arborelui Huffman este de *O(n log n)*.

Corectitudinea algoritmului

Trebuie să demonstrăm că algoritmul lui Huffman produce un cod prefix optimal.

Lema 1.

Fie x, y ∈ C, două caractere care au frecvenţa cea mai mică. Atunci există un cod prefix optimal în care x şi y au codificări de aceeaşi lungime şi care diferă doar prin ultimul bit.

Demonstraţie:

Fie T arborele binar asociat unui cod prefix optimal oarecare şi fie a, b două noduri de pe nivelul maxim, care sunt fiii aceluiaşi nod. Putem presupune fără a restrânge generalitatea că f[a] ≥ f[b] şi f[x] ≥ f[y]. Cum x şi y au cele mai mici frecvenţe rezultă că f[x] ≥ f[a] şi f[y] ≥ f[b]. Transformăm arborele T în T', schimbând pe a cu x :

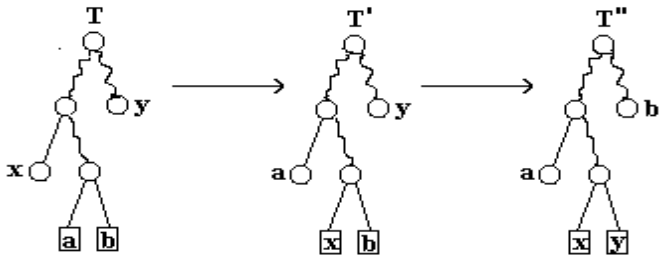


Fig. 7.

Notăm cu C(T) costul arborelui T:

$$C(T) = \sum_{c \in C} f(c) \cdot niv_T(c)$$

$$C(T) - C(T') = f(x) \cdot niv_T(x) + f(a) \cdot niv_T(a) - f(x) \cdot niv_{T'}(x) - f(a) \cdot niv_{T'}(a) \Leftrightarrow$$

$$C(T) - C(T') = (f(a) - f(x)) \cdot (niv_T(a) - niv_T(x)) \geq 0, \text{ pentru că } \begin{cases} niv_{T'}(x) = niv_T(a) \\ niv_{T'}(a) = niv_T(x) \end{cases}$$

În mod analog, construim arborele T, schimbând pe x cu y. Obţinem:

$$C(T') - C(T'') = (f(b) - f(y)) \cdot (niv_T(b) - niv_T(y)) \geq 0.$$

Deci C(T) ≥ C(T"), dar cum T era optimal deducem că T″este arbore optimal şi în plus x şi y sunt noduri terminale situate pe nivelul maxim, fii ai aceluiaşi nod.

Observaţie

Din lemă deducem că pentru construcţia arborelui optimal putem începe prin a selecta după o strategie Greedy caracterele cu cele mai mici frecvenţe.

Lema 2.

Fie T un arbore binar complet corespunzator unui cod prefix optimal pentru alfabetul C. Dacă x şi y sunt două noduri, fii ai aceluiaşi nod z în T, atunci arborele T' obținut prin eliminarea nodurilor x şi y este arborele binar complet asociat unui cod prefix optimal pentru alfabetul

C' = (C-{x,y}) ∪ { z}, f[z] fiind f[x]+f[y].

Demonstrație:

$$\forall c \in C - \{x, y\}, niv_T(c) = niv_{T'}(c) \\ niv_T(x) = niv_T(y) = niv_T(z) + 1 \Rightarrow f(x) \cdot niv_T(x) + f(y) \cdot niv_T(y) =$$

$$= (f(x) + f(y)) \cdot (niv_T(z) + 1) = f(z) \cdot niv_T(z) + f(x) + f(y) =$$

$$= f(z) \cdot niv_{T'}(z) + (f(x) + f(y)).$$

Deci,

$$C(T) = \sum_{c \in C} f(c) \cdot niv_T(c) = C(T') + (f(x) + f(y))$$

Dacă presupunem prin reducere la absurd că arborele T' nu este optimal pentru alfabetul C' = (C-{x,y}) ∪ {z} ⇒ ∃ T″, ale cărui frunze sunt caractere din C', astfel încât C(T″) < C(T).

Cum z este nod terminal în T″, putem construi un arbore T_1 pentru alfabetul C, atârnând pe x [i y ca fii ai lui z.

$C(T_1) = C(T'')+f(x)+f(y) < C(T)$, ceea ce contrazice ipoteza că T este arbore optimal pentru C. Deci, T' este un arbore optimal pentru C'.

Corectitudinea algoritmului lui Huffman este o consecință imediată a celor două leme.

Observație

Metoda de compresie bazată pe arbori Huffman statici are o serie de dezavantaje:

1. Fişierul de compactat trebuie parcurs de două ori: o dată pentru a calcula numărul de apariții ale caracterelor în text, a doua oară pentru a comprima efectiv fişierul. Deci metoda nu poate fi folosită pentru transmisii de date pentru care nu este posibilă reluarea.

2. Pentru o dezarhivare ulterioară a fişierului, aşa cum este şi firesc, este necesară memorarea arborelui Huffman utilizat pentru comprimare, ceea ce conduce la mărirea dimensiunii codului generat.

3. Arborele Huffman generat este static, metoda nefiind capabilă să se adapteze la variații locale ale frecvențelor caracterelor.

Aceste dezavantaje au fost în mare parte eliminate în metodele care folosesc arbori Huffman dinamici, ideea fiind ca la fiecare nouă codificare, arborele Huffman să se reorganizeze astfel încât caracterul respectiv să aibă eventual un cod mai scurt.