Structuri de Date

Laboratorul 8 - Codificare Huffman folosind arbori binari

Responsabil: Stefan Bucur

Data publicarii: 16.04.2007
Data ultimei modificari: 16.04.2007

Obiective

In urma parcurgerii acestui laborator, studentul va fi capabil:

- sa inteleaga modul in care informatia este masurata in biti
- sa inteleaga utilitatea diverselor forme de codificare
- sa implementeze algoritmul Huffman de generare a arborelui de simboluri, avand la baza un sir de date de intrare
- sa parcurga un sir de date comprimate, si pe baza arborelui Huffman, sa obtina sirul original

Notiuni teoretice

1. Informatia. Codificarea informatiei.

Notiunea de *informatie* poate fi privita din mai multe puncte de vedere, precum cel lexicografic (ce se refera la notiuni si fapte), sau filosofic. In inginerie, totusi, informatia are o definitie matematica riguroasa. Ea reprezinta o *inlaturare a incertitudinii*, si este o marime a carei unitate de masura este *bitul*. Astfel pentru reprezentarea unei informatii, vom avea nevoie de un anumit numar de biti de informatie (nu neaparat intreg, dupa cum vom vedea mai jos!).

Intuitiv, ne putem gandi ca odata ce avem mai multi biti de informatie, stim mai multe despre entitatea respectiva, deci incertitudinea este mai bine inlaturata. De exemplu, un numar real este mai bine reprezentat de tipul `double` (pe 64 de biti), decat de tipul `float` (pe 32 de biti), iar precizia calculelor este mai buna. Deci putem spune ca tipul `double` contine mai multa informatie decat tipul `float`. (Ca un fapt divers, pentru a inlatura in intregime incertitudinea, am avea nevoie de un numar infinit de biti pentru un numar real oarecare.)

Bine-nteles, in inginerie este nevoie de definitii riguroase, iar definitia cantitatii de informatie este legata de teoria probabilitatilor. Aceasta spune ca pentru o multime formata din n obiecte (simboluri), cu proprietatea ca fiecare obiect i poseda probabilitatea p_i de aparitie, incertitudinea (cantitatea de informatie necesara reprezentarii) pentru acest sistem este definita ca:

$$H = - p_1 * \log_2(p_1) - ... - p_n * \log_2(p_n)$$

Astfel, de exemplu, daca avem o urna cu 8 bile, si extragem o bila intamplare, stiind ca fiecare bila poate fi extrasa cu o probabilitate egala $p_i = 1/8$, atunci cantitatea de informatie, conform formulei de mai sus, este:

$$H = - 8 * 1/8 * \log_2(1/8) = - (-3) = 3$$

Deci pentru reprezentarea rezultatului acestui eveniment, avem nevoie de 3 biti de informatie. Acest rezultat era oarecum de asteptat, daca ne gandim ca putem numerota fiecare bila de la 0 la 7, si pentru acest interval de numere putem folosi 3 biti de date.

Un alt exemplu, de data aceasta mai abstract si mai apropiat de lumea calculatoarelor, este legat de cantitatea de informatie asociata unei litere a alfabetului, care poate aparea intr-un sir de litere. Stiind ca poate fi vorba de oricare dintre cele 26 de litere, cu aceeasi probabilitate de aparitie (1/26), informatia asociata unei litere este:

$$H = - 26 * 1/26 * \log_2(1/26) = 4.7 \text{ biti}$$

Deci pentru a reprezenta o litera din alfabet, avem nevoie de 4.7 biti de informatie. Insa este clar ca nici o masina numerica nu poate lucra cu un numar fractional de biti, astfel ca pentru a reprezenta o litera din alfabet, vor fi necesari minim 5 biti numerici. In acest moment se poate sesiza diferenta clara intre informatia *teoretica*, si *codificarea* acesteia, adica modul in care aceasta este reprezentata pe masina numerica.

De exemplu, pentru numerele de la 0 la 9, informatia asociata unei cifre este 3.32 biti, insa o cifra este reprezentata de obicei pe 4 biti, iar codificarea este asociata ordinii numerice in baza 2: 0000 pentru cifra 0, 0001 pentru 1, 0010 pentru 2, pana la 1001 pentru cifra 9. Acest tip de codificare este numit, din motive evidente, *cu lungime fixa*. Fiecare cifra (simbol), are asociat acelasi numar de biti. Avantajul major al acestui tip de codificare este simplitatea cu care pot fi manipulate reprezentarile simbolurilor, care pot fi aliniate foarte usor in memorie, iar acest lucru il face foarte popular in informatica. Numerele intregi sunt reprezentate in lungime fixa pe 8, 16, 32 sau 64 de biti, numerele reale pe 32 sau 64 de biti, s.a.m.d.

Cu toate acestea, reprezentarea pe lungime fixa are un dezavantaj: numarul de biti folositi pentru reprezentare nu este neaparat minim! De exemplu, pentru o cifra sunt folositi 4 biti (numerici) in loc de 3.32 (teoretici). Pentru un sir de 1000 de cifre, vor fi folositi 4000 de biti, in loc de aproximativ 3320. Teoria afirma ca numarul de biti de informatie (teoretici) este minimul posibil care poate fi atins de orice codificare aleasa. Teoria compresiei datelor se ocupa tocmai cu acest lucru: gasirea de metode de codificare a simbolurilor, astfel incat numarul de biti necesari pentru reprezentarea datelor (vazute ca siruri de simboluri), sa fie cat mai mic si mai aproape de limita teoretica.

2. Codificarea Huffman

Este evident, din expunerea precedenta, ca folosirea unei codificari cu lungime fixa nu va fi optima in orice situatie. O varianta mai buna de codificare, desi mai costisitoare din punct de vedere computational, este cea *cu lungime variabila*, in care lungimea fiecarui simbol poate diferi.

O prima problema care se ridica la codificarea cu lungime variabila este cum se face delimitarea simbolurilor intr-un sir de date. La codificarea cu lungime fixa lucrurile stateau foarte simplu: la fiecare n biti gaseam un nou simbol. Insa in cazul lungimii variabile, secretul sta in gasirea unui algoritm de generare al codificarilor pentru fiecare simbol, astfel incat sa ne putem da seama cand se termina un simbol si putem sa trecem la citirea urmatorului.

Codificarea Huffman este o varianta foarte populara de codificare cu lungime variabila. Ea presupune ca datele sunt reprezentate, in general, ca un sir de simboluri (octeti, cifre, litere, de exemplu), unde fiecare simbol ai din multimea tuturor simbolurilor posibile (alfabetul $A = \{a_1, a_2, ..., a_n\}$) , are asociata o anumita probabilitate de aparitie p_i. Probabilitatile se pot determina fie estimativ (de exemplu pe baza tipului sirului de date: text, imagine, cod executabil), fie inspectand sirul de la cap la coada si numarand aparitiile fiecarui simbol, apoi impartind fiecare contor la numarul total de simboluri.

Vom considera in continuare, ca exemplu, codificarea Huffman a sirului de date

```
"ana are mere"
```

Alfabetul A este `A = {'a', 'n', ' ', 'r', 'e', 'm'}`. Sirul de date are lungimea totala 12, iar pe baza frecventei fiecarui simbol in sir, putem asocia fiecarui simbol probabilitatile de mai jos:

Simbol	'a'	'n'	' '	'r'	'e'	'm'
Probabilitate	1/4	1/12	1/6	1/6	1/4	1/12

Conform formulei cantatii de informatie necesare reprezentarii unui simbol, in cazul nostru un simbol se va reprezenta teoretic prin

$$H = 1/4*\log_2(4) + 1/12*\log_2(12) + 1/6*\log_2(6) + 1/6*\log_2(6) + 1/4*\log_2(4) + 1/12*\log_2(12) = 2.45 \text{ biti}$$

Daca vom asocia fiecarui simbol un numar de biti (numerici), $I_1, \ldots, I_n$, atunci numarul mediu real de biti al unui simbol va fi media ponderata a lungimilor acestora, cu ponderea egala cu probabilitatile de aparitie:

$$I_{med} = I_1 * p_1 + \ldots + I_n * p_n$$

Algoritmul Huffman incearca sa apropie valoarea I_{med} cat mai mult de cea teoretica, H. Ideea de la baza compresiei Huffman este de a asocia simbolurilor cu ponderea cea mai mare (adica care apar cel mai frecvent in text), coduri de lungime cat mai mica, si viceversa pentru simbolurile cu aparitie foarte rara, si in acelasi timp codurile sa poata fi usor delimitate intr-un sir continuu de simboluri (a se vedea problematica de la codificarile cu lungime variabila).

Acest lucru este realizat intr-un mod foarte ingenios plasand fiecare simbol intr-un mod convenabil ales in frunzele unui arbore binar, iar codificarea simbolurilor sa fie de fapt calea parcursa prin arbore, de la radacina spre fiecare simbol. Algoritmul Huffman reprezinta algoritmul de constructie al acestui arbore, pornind de la o multime de simboluri A, fiecare avand asociat probabilitatea pi.

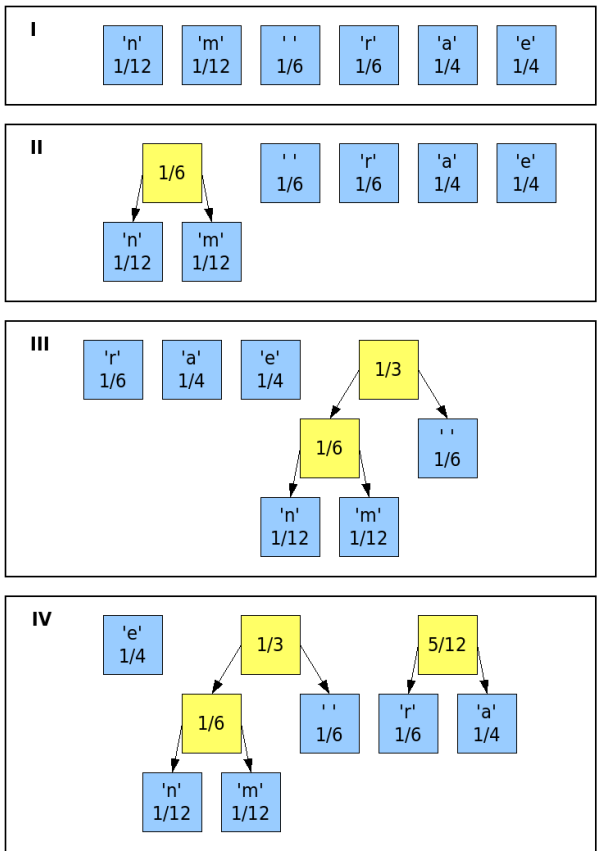
3. Algoritmul Huffman de construire a arborelui binar de simboluri

Arborele binar de simboluri contine in fiecare frunza un simbol, alaturi de probabilitatea sa de aparitie. Fiecare nod intern al arborelui va contine doar un numar, si anume suma probabilitatilor simbolurilor descendente. Astfel radacina arborelui va contine probabilitatea 1, intrucat toate simbolurile din alfabet ii sunt descendente. Parcurcand arborele de la radacina catre o frunza, se poate genera codul asociat simbolului din frunza respectiva: la fiecare pas al parcurgerii, pentru o alegere a nodului stang se adauga un '0', iar pentru o alegere a nodului drept, se adauga un '1'.

Mai jos este prezentat algoritmul Huffman de constructie a arborelui de simboluri:

1. Construieste cate un nod frunza pentru fiecare simbol si probabilitate asociata.
2. Adauga nodurile frunza intr-o coada, sortandu-le in ordinea crescatoare a probabilitatilor, de la cele mai rare simboluri la cele mai frecvente.
3. Atata timp cat coada contine mai mult de un nod, executa:
   3.1. Extrage primele 2, cele mai infrecvente noduri, si creeaza un alt nod, cu probabilitatea egala cu suma probabilitatilor celor doua noduri.
   3.2. Adauga cele 2 noduri extrase ca descendenti stang, respectiv drept, la noul nod creat.
   3.3. Insereaza nodul creat inapoi in coada, pastrand ordinea de sortare in functie de probabilitatile nodurilor
4. Coada va contine un singur nod, si anume radacina arborelui Huffman, care va avea probabilitatea 1.

Figura de mai jos prezinta procesul de constructie al arborelui Huffman pentru exemplul considerat:



Pe baza arborelui Huffman generat, fiecare simbol va avea urmatoarea codificare:

Simbol	'a'	'n'	' '	'r'	'e'	'm'
Codificare	01	1100	111	00	10	1101

Astfel, sirul "ana are mere" va fi codificat astfel:

01 1100 01 111 01 00 10 111 1101 10 00 10

iar numarul total de biti este 30. Intrucat teoria prezicea un numar de biti minim de `H*lungime = 2.45 * 12 = 29.4 biti`, putem afirma cu incredere ca algoritmul Huffman a comprimat bine datele.

Daca am fi folosit un cod cu lungime fixa (in cazul nostru lungimea fixa minima ar fi fost de 3 biti), numarul total de biti necesari pentru sirul nostru ar fi fost 3*12 = 36 de biti, semnificativ mai mare decat rezultatul obtinut. Mai mult decat atat, daca am fi reprezentat sirul de caractere ca un vector de coduri ASCII (8 biti), ar fi fost necesar un numar de 8*12 = 96 de biti pentru reprezentare!

Astfel, pentru a decomprima un fisier arhivat folosind compresia Huffman, avem nevoie de arborele Huffman (codificat cumva la inceputul arhivei), si de sirul de biti de procesat. Observati ca putem parcurge sirul si sa identificam fiecare simbol in parte: ne pozitionam in radacina arborelui, si pentru fiecare 0 sau 1 citit, avansam in descendentul stang, respectiv drept, pana cand ajungem la o frunza. Atunci stim ca am parcurs un cod intreg, si simbolul codat este continut in fruza respectiva. Apoi ne resetam pozitia pe nodul radacina, pentru a citi urmatorul simbol, si asa mai departe.

Aparent s-ar putea spune ca bitii castigati la compresie sunt cheltuiti pentru reprezentarea arborelui Huffman din arhiva. Acest lucru este adevarat pentru fisiere mici, unde compresia nu este asa de eficienta. Insa odata cu cresterea dimensiunii fisierului, arborele de simboluri ramane practic constant, insa datele sunt micsorate semnificativ.

Aplicatii

Scrieti un program care parcurge un fisier text, construieste o lista de simboluri si de probabilitati asociate, si apoi pe baza acestei liste construieste arborele Huffman corespunzator. Pentru aceasta, veti avea in vedere urmatoarele:

a) Veti citi fisierul text caracter cu caracter, si veti cauta in lista de caractere, caracterul curent; daca acesta exista, incrementati numarul de aparitii, iar daca acesta este absent il adaugati, cu numarul de aparitii 1. Probabilitatea fiecarui caracter este numarul de aparitii impartit la numarul total de caractere citite.

b) Veti construi apoi o lista de noduri, pe care la fiecare pas al algoritmului Huffman o veti sorta crescator dupa probabilitatea nodului radacina de pe fiecare pozitie. Algoritmul de sortare poate fi oarecare (bubble sort, pentru simplitate).

c) Programul vostru va afisa urmatoarele:

- numarul total de caractere din fisierul text
- numarul total de biti din fisierul text, necomprimat
- fiecare caracter din lista de simboluri, alaturi de codul Huffman generat
- numarul total de biti din fisierul text, comprimat

Pentru implementarea listelor puteti folosi vectori sau biblioteca Cobble. Nu este nevoie sa implementati optimizari, avand in vedere timpul limitat de lucru.

Bibliografie

- Cursul 5 de Structuri de Date (arbori binari) - http://learningcenter.ncit.pub.ro/mod/resource/view.php?id=875
- Huffman Coding (Wikipedia) - http://en.wikipedia.org/wiki/Huffman_coding