Laborator PS
Algoritmi de compresie a datelor

Lucrarea 1: Introducere în Compresia Datelor

Lucrarea 2: Algoritmul Shannon-Fano

Lucrarea 3: Algoritmul Huffman static

Lucrarea 4: Algoritmul Huffman dinamic

Prof. Dan Ştefănoiu <dan.stefanoiu@acse.pub.ro>

Ş.l. Alexandru Dumitraşcu <alexandru.dumitrascu@acse.pub.ro>

A. Obiective

Introducerea principalelor concepte din domeniul Compresiei Datelor. Însuşirea unui un mic vocabular de termeni specifici, care va fi folosit şi în lucrările următoare.

B. Semnale utilizate

Pentru a testa algoritmii din acest capitol, sunt disponibile trei categorii de fişiere de date:

- **Documente.** Din această categorie fac parte fişierele:
  - **Documentatie_UMAPID.doc**;
  - **Documentatie_UMAPID.pdf**;
  - **Prefata_Undine.txt**;
  - **show_audio.m** (sursă MATLAB);
  - **Y04.m** (serie de date în format ASCII).

  Acestea pot fi extrase din arhiva **Documents.rar**.

- **Semnale audio reale.** Arhiva **Audio.rar** include 3 semnale înregistrate în formatul **wave**:
  - **instr_01.wav** – sunet emis de un instrument muzical;
  - **sound_01.wav** – sunet fabricat artificial;
  - **speech_01.wav** – semnal *vocal* (emis de vocea umană).
- **Executabile.** Fişierele din această categorie sunt incluse în arhiva **Executables.rar** şi conțin cod executabil sau date împachetate în formatul **mat** din MATLAB. Ele sunt următoarele:
  - **KARMA_DATA482#1_5_V7.mat**;
  - **quartz.dll**;
  - **WinRar.exe**;
  - **WINZIP32.EXE**.

C. Suport teoretic

Compresia Datelor (CoDa), este un domeniu de cercetare care face parte din câmpul Teoriei Informației – ramură a Matematicii născută către sfârşitul anilor ’40, odată cu lucrările publicate de către cercetătorul american C.E. Shannon (de la Bell Laboratories) şi matematicianul ungur D. Gabor (unul dintre pionierii Teoriei Comunicației, laureat al premiului Nobel). Aceste lucrări prezintă o serie de soluții (mai mult de ordin teoretic) ale unor probleme legate de emisia, culegerea, prelucrarea, transmisia, recepția şi interpretarea informației cu ajutorul unui mijloc automat de calcul. În această lucrare, sunt prezentate principalele concepte care definesc domeniul CoDa, fără a intra în detalii de natură teoretică.

Concepte de bază

Obiectul de studiu al CoDa îl constituie metodele şi tehnicile destinate reducerii volumului unui set de date reprezentate sub formă binară, cu posibilitatea reconstituirii acestuia fie exact, fie aproximativ (conform unor criterii definite a priori). Posibilitatea de a reduce volumul unui set de date se bazează pe câteva fenomene importante legate de reprezentarea binară a informației.

Astfel, setul de date este privit ca un şir finit de *simboli*. Acesta este un termen consacrat (nu foarte diferit de termenul colocvial de ***simboluri***). Fiecare dintre simboli are o reprezentare numerică unică numită *cod (numeric) original*. Numărul simbolilor ce compun un set de date, adică ***volumul*** acestuia, este, în general, mult mai mare decât numărul de simboli distincți ce participă la construcția sa. **De regulă, un simbol este reprezentat în baza 2, pe 8 biți (1 octet), astfel că setul de date se construieşte folosind maximum $2^8 = 256$ simboli distincți.** Mulțimea simbolilor distincți dintr-un set de date se numeşte ***alfabet***, denumire sugerată de conceptul uzual de *alfabet literar*, din vorbirea colocvială. Alfabetul din CoDa include literele curente (A, B, C etc.), dar nu se reduce doar la acestea; el poate conține şi alți simboli (cifre numerice, semne de punctuație, semne speciale care nu au o interpretare directă etc.). Reprezentarea simbolilor pe 8 biți este realizată

conform unui standard internațional numit *ASCII*. Tabela ASCII pune în corespondență biunivocă mulțimea celor 256 de simboli cu mulțimea celor 256 de coduri numerice (de la 0 la 255). De exemplu, codul (numeric) original asociat literei 'A' este 65, în bază zecimală. Acest cod poate fi exprimat prin **0x41** în baza 16 (prin convenție, numerele precedate de **0x** sunt reprezentate în baza 16 sau în *cod hexazecimal*) sau prin **0100 0001** în baza 2 sau în cod binar.

Un al doilea fenomen important al reprezentării binare a informațiilor este cel al frecvenței diferite de apariție a simbolilor alfabetului de-a lungul setului de date. Nu toți simbolii participă în mod uniform la construcția setului de date: unii sunt mai frecvenți decât alții. Acest fapt a sugerat ideea că **alfabetul ar putea fi recodificat cu ajutorul unor coduri de lungime variabil**ă, în aşa fel încât simbolilor ce apar frecvent să li se aloce coduri mai scurte decât simbolilor ce apar mai rar în setul de date.

Rată de compresie, factor de compresie

Un prim parametru care priveşte performanțele metodei de compresie utilizate este ***rata de compresie***. Aceasta se notează cu ρ şi are următoarea exprimare:

$$\rho = \left[1 - \frac{\#dc}{\#do}\right] \times 100\ [\%], \qquad (1.1)$$

unde: $\#dc$ este lungimea setului de date comprimat, iar $\#do$ este lungimea setului de date original.

Această definiție arată că:

- dacă $\rho = 100\ \%$, atunci metoda de compresie aferentă este *ideală*, câştigul obținut prin compresie fiind maxim; în practică, însă, $\rho < 100\ \%$;

- dacă $\rho = 0\ \%$, atunci metoda nu realizează nici o compresie efectivă, câştigul fiind nul; în practică, se spune, în acest caz, că setul de date a fost *povestit* (*stored*), adică redat ca atare, fără a suferi o compresie efectivă;
- dacă $\rho < 0\ \%$, atunci metoda aferentă nu realizează o compresie efectivă, ci o *expandare* a datelor (adică o reprezentare a acestora cu mai mulți biți decît originalul), fiind ineficientă.

Termenul de *rată de compresie* este adesea înlocuit de cel de ***factor de compresie***, notat cu φ. Acesta se defineşte ca raportul procentual dintre lungimea setului de date comprimat şi cea a setului de date original:

$$\varphi = 1 - \rho = \frac{\#dc}{\#do} \times 100\ [\%]. \qquad (1.2)$$

Interpretarea sa este considerată mai directă decât cea a ratei de compresie. *O metodă de compresie este cu atât mai eficientă cu cât factorul de compresie este mai mic.*

Entropie, cantitate de informație, redundanță

Un concept care leagă CoDa de Teoria Informației este cel de ***redundanță***. Înainte de a-l defini, trebuie subliniat faptul că obiectivul principal al unei metode de compresie a datelor este detectarea, localizarea şi eliminarea *redundanțelor* (adică a repetabilității informațiilor) din setul de date.

O măsură a *redundanței* unui set de date nu este dată de rata de compresie (aşa cum am fi tentați să credem), deoarece rate diferite de compresie se pot obține pentru acelaşi set de date, dacă se apelează la metode de compresie diferite. *Redundanța* este o constantă a setului de date şi nu depinde de metoda de compresie utilizată.

Volumul unui set de date este numărul simbolilor (nu neapărat distincți) care îl compun. ***Lungimea*** unui set de date este numărul total de biți pe care este el reprezentat. Cele două noțiuni sunt diferite.

Vom nota prin $\nu(s)$ frecvența de apariție a simbolului s. Ea constituie o estimație a *probabilității de apariție* a simbolului s, notată cu $\mathfrak{P}(s)$. Plecând de la $\nu(s)$, se poate calcula numărul minim de biți suficient pentru a recodifica simbolul s, fără a confuzie cu alți simboli. Acest număr se numeşte ***entropie*** şi poate fi fracționar (nu neapărat întreg). Practic, însă, el va fi adesea rotunjit la întregul superior, din motive lesne de înțeles. Entropia va fi cel mult egală cu numărul de biți ai codului original (de exemplu, 8). Diferența dintre ele constituie o măsură a *redundanței* intrinseci transportate de acel simbol în cadrul setului de date (adică numărul de biți inutili prezenți în codul original).

Entropia unui simbol s dintr-un set de date (adică *entropia informatică*) se notează prin $\mathfrak{H}(s)$ şi se defineşte astfel:

$$\mathfrak{H}(s) = -\log_2 \nu(s). \tag{1.3}$$

În definiția (1.3) se observă imediat că: $\nu(s) \in (0,1]$, ceea ce arată că $\mathfrak{H}(s) > 0$. În consecință:

$$\nu(s) = 2^{-\mathfrak{H}(s)} \in [0,1]. \tag{1.4}$$

Dacă, de exemplu, litera 'a' apare de 38 de ori într-un text cu volumul de 100 de caractere, atunci $\nu('a') = 0.38$, ceea ce conduce la entropia informatică $\mathfrak{H}('a') = -\log_2 0.38 \cong 1.4$ biți. Rezultă că, în codarea inițială a lui 'a', 6.6 biți (din cei 8 alocați) sunt inutili, adică *redundanți*. Practic, va fi suficient ca literei 'a' să i se aloce doar 2 biți, pentru a obține o primă compresie a textului.

Un alt concept direct legat de frecvența de apariție a unui simbol s este ***cantitatea de informație***, notată cu $\imath(s)$. Ea este invers proporțională cu frecvența $\nu(s)$:

$$\imath(s) = \frac{C}{\nu(s)}, \tag{1.5}$$

unde $C > 0$ este o constantă de proporționalitate.

Un simbol ce apare frecvent în setul de date transportă o cantitate de informație mică şi va primi un număr mic de biți de recodificare. Acest fenomen este mai bine ilustrat de următoarea relație evidentă (obținută prin combinarea definițiilor (1.3) şi (1.5)):

$$\mathcal{H}(s) = -\log_2 C + \log_2 \imath(s), \tag{1.6}$$

în care se poate renunța la termenul constant $-\log_2 C$.

Se notează, mai departe, prin $\mathcal{D}$ setul de date asupra căruia se operează (mulțimea tuturor simbolilor (nu neapărat distincți) ce alcătuiesc setul de date). Reamintim că *alfabetul* setului de date este mulțimea tuturor simbolilor *distincți* ce alcătuiesc setul de date şi, în general, diferă de mulțimea $\mathcal{D}$. *Redundanța* setului de date $\mathcal{D}$ se măsoară prin numărul total de biți utilizați, dar inutili, calculat astfel:

$$\mathcal{R}(\mathcal{D}) = \sum_{s \in \mathcal{D}} [8 - \mathcal{H}(s)]. \tag{1.7}$$

(Am considerat că fiecărui simbol i s-a alocat inițial un cod pe 8 biți.) Redundanța depinde direct de entropiile tuturor simbolilor mesajului. Ea este cu atât mai pronunțată cu cât aceste entropii sunt mai mici. Dacă ținem cont de definiția (1.3) a entropiei, atunci redundanța (1.7) se poate exprima şi astfel (în funcție de frecvențele de apariție):

$$\mathcal{R}(\mathcal{D}) = \sum_{s \in \mathcal{D}} [8 - \log_2 \nu(s)]. \qquad (1.8)$$

Astfel, frecvențe mari de apariție ale simbolilor setului de date conduc la o redundanță importantă a acestuia.

Relația (1.8) poate fi utilizată şi pentru evaluarea redundanței unor subseturi de date din setul original. De exemplu, şirul de 5 caractere 'aaaaa' (numit ad hoc *cuvânt*) este reprezentat inițial pe un număr de $5 \times 8 = 40$ de biți. Dacă el face parte dintr-un set de date mai larg, în care $\nu('a') = 1/16$, atunci: $\mathcal{H}('a') = -\log_2 1/16 \cong 4$ şi $\mathcal{R}('aaaaa') = 40 - 5 \times 4 = 20$ biți. Rezultă că, în cuvîntul 'aaaaa', doar 20 de biți de reprezentare sunt utili, adică 50%. Mai mult, acest cuvânt ar putea fi codificat în mod ideal cu numai 7 biți (în loc de 20), reprezentînd codul literei 'a' (pe 4 biți) şi numărul de apariții consecutive (5, număr care ocupă 3 biți: 101). Cu toate acestea, o reprezentare pe un număr total de biți scăzut este dificil de realizat, căci, în procesul de decompresie, o serie de informații auxiliare trebuie de asemenea să se regăsească în setul de date comprimat. De exemplu, este necesar să se cunoască numărul de biți alocați simbolului 'a', poziția cuvîntului 'aaaaa' în setul de date etc. Ori, această informație poate anula câştigul obținut prin calculul entropiei şi redundanței.

D. Ghid C++

În continuare, sunt prezentate câteva funcții standard ce pot fi folosite în conceperea programelor C++. Pentru programele MATLAB, funcțiile sunt similare şi pot fi interogate cu ajutorul facilităților **`Help`**, **`Doc`**, etc.

Pentru a deschide un fişier, se poate utiliza apelul:

```
FILE * fopen ( const char * filename, const char * mode );
```

unde: ieşirea funcției este pointerul către o variabilă de tip **`FILE`**, **`filename`** este numele complet al fişierului, scris sub forma unui şir de caractere, iar **`mode`** este modul de deschidere (unul sau mai multe caractere). Un fişier poate fi deschis în mai multe moduri, dar cele mai frecvente sunt:

- **'r'** = modul *citire* (*read*); nu se poate înscrie nimic în fişierul deschis;
- **'w'** = modul *scriere* (*write*); nu se poate citi nimic din fişierul deschis; scrierea se face peste datele deja existente în fişier (dacă există), de la prima poziție a acestuia; modul scriere permite deschiderea un nou fişier, care nu a mai fost creat;
- **'a'** = modul *adăugare* (*append*); dacă fişierul nu există, el va fi creat şi deschis în modul scriere; altfel, el va fi deschis astfel încît la datele deja existente să se poată adăuga altele noi;
- **'r+'** = modul *citire-scriere*; pentru acest mod, fişierul trebuie să fi fost creat anterior, altfel el nu poate fi creat; odată deschis, fişierul poate fi citit şi/sau supra-înscris (adică informația nouă o supra-înscrie pe cea veche);
- **'a+'** = modul *citire-adăugare*; dacă fişierul nu a fost creat anterior, el va fi creat acum şi deschis; odată deschis, fişierul poate fi citit şi/sau extins (adică informația nouă se adaugă la cea veche).

Dacă funcția returnează pointerul **`NULL`** , utilizatorul este informat că fişierul nu a putut fi deschis, din diferite cauze (cel mai frecvent, pentru că nu a putut fi găsit pe disc în vreuna dintre locațiile de căutare). Pentru utilizarea acestei funcții, precum şi a celorlalte prezentate mai jos, este nevoie de introducerea bilbiotecii asociate antetului **`stdio.h`** (*standard input-output functions*).

Utilizatorul poate accesa informația dintr-un fişier deschis în diferite moduri. Există un pointer virtual care indică locația din fişier ce poate fi citită sau înscrisă. Dupa citirea/scrierea fiecărei date, pointerul se incrementează automat. Utilizatorul poate totuşi repoziționa acest pointer astfel:

```
int fseek ( FILE * stream, long int offset, int origin );
```

unde:

- **`stream`** este indicatorul de fişier, aşa cum a fost el returnat de către funcția **`fopen`**;
- **`offset`** este un întreg care arată că pointerul trebuie deplasat cu valoarea absolută a acestuia către sfîrşitul fişierului (dacă e strict pozitiv), către începutul fişierului (dacă e strict negativ) sau deloc (dacă este nul);
- **`origin`** este o etichetă de tip şir de caractere, care poate avea una dintre următoarele forme: **`SEEK_SET`**, **`SEEK_CUR`** sau **`SEEK_END`**; ea arată de unde trebuie să plece pointerul: **`SEEK_SET`** – de la începutul fişierului, **`SEEK_CUR`** – de la poziția curentă în fişier, respectiv **`SEEK_END`** – de la sfîrşitul fişierului.

Funcția **`fseek`** returnează un indicator de stare, care arată dacă repoziționarea s-a efectuat cu succes (**0**) sau nu (o valoare diferită de 0).

Dacă poziția indicată de pointerul virtual nu (mai) este cunoscută, ea poate fi (re)găsită astfel:

```
long int ftell ( FILE * stream );
```

Argumentul de ieşire este un întreg care măsoară poziția în octeți, începînd cu numărul 0 (începutul de fişier). Dacă există o eroare, este returnată valoarea **`-1L`**, iar variabila globală **`errno`** este setată cu o valoare pozitivă. Această valoare poate fi interpretată de funcția **`perror`**.

Citirea de date dintr-un fişier deschis se poate realiza, în general, cu ajutorul funcției **`fread`**. Declarația uzuală a acesteia este următoarea:

```
size_t fread ( void * ptr, size_t size, size_t count, FILE * stream );
```

unde:

- **ptr** este un pointer către un bloc de memorie cu dimensiunea minimă de (**size*count**) octeți;
- **size** este dimensiunea în octeți a fiecărui element de citit;
- **count** este un întreg care arată cîte date de dimensiune **size** trebuie citite;
- **stream** este indicatorul de fişier (returnat de către funcția **fopen**).

Biblioteca asociată antetului **stdio.h** conține o colecție de funcții importante ce pot fi folosite în realizarea unui program C++. Lista completă de funcții este prezentată mai jos. Pentru mai multe informații cu privire la utilizarea lor se poate accesa pagina web :

http://www.cplusplus.com/reference/clibrary/cstdio/

- **clearerr**
- **fclose**
- **feof**
- **ferror**
- **fflush**
- **fgetc**
- **fgetpos**
- **fgets**
- **fopen**
- **fprintf**
- **fputc**
- **fputs**
- **fread**
- **freopen**
- **fscanf**
- **fseek**
- **fsetpos**
- **ftell**
- **fwrite**
- **getc**
- **getchar**
- **gets**
- **perror**
- **printf**
- **putc**
- **putchar**
- **puts**
- **remove**
- **rename**
- **rewind**
- **scanf**
- **setbuf**
- **setvbuf**
- **sprintf**
- **sscanf**
- **tmpfile**
- **tmpnam**
- **ungetc**
- **vfprintf**
- **vprintf**
- **vsprintf**

E. Sarcini de lucru

Tema 1 (Operarea cu fişiere, evaluarea dimensiunii acestora)

Scrieți un program prin care să determinați dimensiunea în octeți şi în perechi de octeți pentru fiecare fişier din corpusul celor 12 descris în subsecțiunea de semnale utilizat. Comparați dimensiunea în octeți obținută cu cea afişată de exploratorul de fişiere (*Windows Explorer*, *Norton Commander*, etc.). Completați apoi programul cu un set de instrucțiuni care să măsoare numărul de biți ai fiecărui fişier. Se potriveşte raportul dintre acest număr şi cifra 8 cu cel de octeți? Argumentați riguros răspunsul.

Tema 2 (Histograme)

Scrieți un program prin care să citiți datele din fiecare fişier al corpusului sub formă de simboli reprezentați întâi pe 8 biți, apoi pe 16 biți. După citire, evaluați şi afişați cele două histograme ale datelor fiecărui fişier. (Histograma se calculează prin detectarea numărului de apariții ale unui simbol. Dacă simbolul este reprezentat pe 8 biți, atunci alfabetul conține cel mult 256 de simboli distincți. Lungimea setului de date fiind, de regulă, mai mare, este clar că unii dintre ei (dacă nu toți) sunt obligați să se repete. Un fenomen similar are loc şi în cazul simbolilor reprezentați pe 16 biți, numai că, aici, frecvența de apariție a simbolilor este mai mică.) Observați cu atenție aceste 24 de histograme şi indicați 3 dintre ele care par apropiate de distribuția gaussiană. Care credeți că este explicația pentru acest tip de distribuție? Există vreo histogramă care sugerează că datele sunt distribuite uniform? Dacă da, pentru care seturi de date? De ce credeți că acestea sunt distribuite cavsi-uniform? Argumentați răspunsurile în manieră riguroasă.

💻 Tema 3 (Entropii, redundanțe)

Utilizând programul de la tema anterioară, scrieți un nou program care evaluează şi trasează grafic entropiile informatice ale simbolilor fiecărui set de date. Pe fiecare pereche de grafice (simboli de 8 biți şi simboli de 16 biți), indicați redundanța setului corespunzător de date. Efectuați o comparație între cele 24 de seturi de date privind redundanța. Enumerați seturile de date în ordinea descrescătoare a compresibilității. (Teoretic, cele mai compresibile sunt şi cele mai redundante.)