Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare

ARBORI DE COMPRESIE HUFFMAN

Introducere

- Tehnicile de **compresie** sunt utile pentru:
  - **fișiere text**, în care anumite caractere apar cu o frecvență mai mare decât altele
  - **fișiere** care **codifică imagini** sau sunt reprezentări **digitale ale sunetelor** sau ale unor **semnale analogice**, care pot conține numeroase secvențe care se repetă

Introducere

- Chiar dacă astăzi capacitatea dispozitivelor de memorare a crescut foarte mult, algoritmii de compresie a fișierelor rămân foarte importanți, datorită volumului tot mai mare de informații care trebuie stocate

Introducere

- De asemenea, compresia este deosebit de utilă în comunicații, transmiterea informațiilor fiind mult mai costisitoare decât prelucrarea lor

Introducere

- Una dintre cele mai răspândite tehnici de compresie a fișierelor text, care, în funcție de caracteristicile fișierului care urmează să fie comprimat, conduce la reducerea spațiului de memorie necesar cu 20-90%, a fost descoperită de D. Huffman în 1952 și se numește **codificare (cod) Huffman**

Observații

- În locul utilizării unui cod în care fiecare caracter să fie reprezentat pe 8 biți (lungime fixă), se folosește un cod mai scurt pentru caracterele care sunt mai frecvente și coduri mai lungi pentru cele care apar mai rar

Exemplu

- Un fișier 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 biți
- 100.000 caractere * 3 biți/caracter = 300.000 biți

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

Observații

- Presupunem că frecvențele cu care apar în text cele 6 caractere sunt:
- **a** (45), **b** (13), **c** (12), **d** (16), **e** (9), **f** (5)
- Considerând codul de lungime variabilă, ar fi necesari doar 224.000 biți, adică o reducere a spațiului de memorie cu aproximativ 25%

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

a	b	c	d	e	f
0	101	100	111	1101	1110

Observații

- Problema se reduce la a asocia fiecărui caracter un cod binar, în funcție de frecvență, astfel încât să fie posibilă decodificarea fișierului comprimat, fără ambiguități

Observații

- Ideea de a folosi separatori între codurile caracterelor pentru a nu crea ambiguități ar conduce la mărirea dimensiunii codificării
- Pentru a evita ambiguitățile, este necesar ca niciun cod de caracter să nu fie prefix al unui cod asociat unui caracter – un astfel de cod se numește **cod prefix**

Codul Huffman

- Huffman a elaborat un algoritm **greedy** care construiește un cod prefix optimal, denumit **cod Huffman**
- Prima etapă în construcția codului Huffman este calcularea numărului de apariții ale fiecărui caracter în text
- Există situații în care putem utiliza frecvențele standard de apariție a caracterelor, calculate în funcție de limbă sau de specificul textului

Etapele codificării

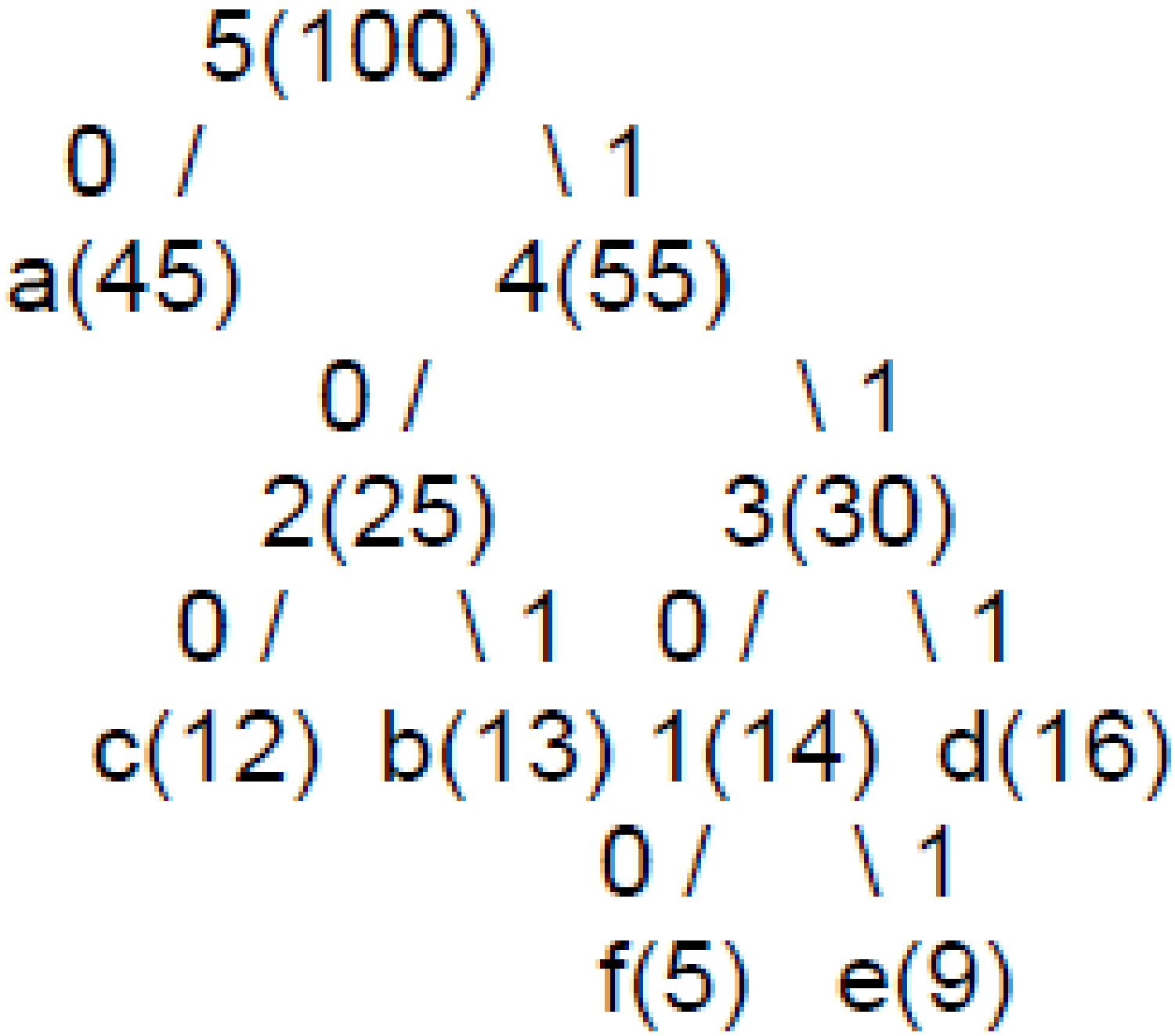
- Codificarea Huffman are 2 etape:
- 1) Construirea unui **arbore binar**, în care fiecare caracter (și frecvența asociată) apare ca o **frunză**
- 2) Parcurgerea arborelui pentru **generarea codurilor Huffman** ale caracterelor

Observații

- Nodurile terminale vor conține un caracter și frecvența corespunzătoare caracterului
- Nodurile interne conțin suma frecvențelor caracterelor corespunzătoare nodurilor terminale din arbore

Observații

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


5(100)
0
1
a(45)
4(55)
0
1
2(25)
3(30)
0
1
0
1
c(12)
b(13)
1(14)
d(16)
0
1
f(5)
e(9)

Observații

- Caracterele care apar cel mai frecvent sunt mai aproape de rădăcină și astfel lungimea codificării va avea un număr mai mic de biți
- La fiecare pas se selectează cele mai mici două frecvențe, pentru a unifica subarborii corespunzători

Observații

- Pentru extragerea eficientă a minimului se folosește o **movilă Min-Heap**
- Timpul de execuție pentru operațiile de extragere minim, inserare și ștergere va fi, în cazul cel mai defavorabil, **O(log n)**

Exemplu

- Evoluția **movilei Min-Heap** cu caracterele și frecvențele asociate este:
- **f** (5), **e** (9), **c** (12), **b** (13), **d** (16), **a** (45)
- **c** (12), **b** (13), **1** (14), **d** (16), **a** (45)
- **1** (14), **d** (16), **2** (25), **a** (45)
- **2** (25), **3** (30), **a** (45)
- **a** (45), **4** (55)
- **5** (100)

Algoritm de construcție a arborelui Huffman

- Pas **1**. *Inițializare*
- **1.1**. Fiecare **caracter** reprezintă un **subarbore** format dintr-un **singur nod**
- **1.2**. Se organizează caracterele ca o **movilă Min-Heap**, în funcție de **frecvențele** de apariție

Algoritm de construcție a arborelui Huffman

- Pas **2**. *Se repetă de **n-1** ori:*
- **2.1**. Se extrag succesiv **X** și **Y**, două elemente din **movila Min-Heap**
- **2.2**. Se unifică subarborii **X** și **Y**
  - **2.2.1**. Se creează **Z**, un nod nou, care va fi rădăcina subarborelui
  - **2.2.2**. **Z->left = X**; **Z->right = Y**;
  - **2.2.3**. **Z->frecv = X->frecv + Y->frecv**;
- **2.3**. Se inserează **Z** în **movila Min-Heap**

Algoritm de construcție a arborelui Huffman

- Pas **3**. Singurul nod rămas în **movila Min-Heap** este rădăcina arborelui Huffman. Se generează codurile caracterelor parcurgând arborele Huffman

Analiza complexității

- Inițializarea movilei Min-Heap este liniară **O(n)**
- Pasul 2 se repetă de n-1 ori și presupune două operații de extragere a minimului dintr-o movilă Min-Heap și o operație de inserare a unui element într-o movilă Min-Heap, care au timpul de execuție de **O(log n)**
- Complexitatea algoritmului de construcție a arborelui Huffman este de **O(n*log n)**

Decodificare Huffman

- Este necesar atât fișierul codificat, cât și arborele folosit la codificare
- Se parcurge arborele de la rădăcină spre stânga pentru o cifră **0** și la dreapta pentru o cifră **1**

Decodificare Huffman

- Se consideră un cursor așezat inițial pe nodul rădăcină al arborelui și se citesc șiruri de biți
- Pentru fiecare bit **0** se execută un avans al cursorului la fiul stâng, respectiv pentru fiecare bit **1** un avans al cursorului la fiul drept al nodului curent

Decodificare Huffman

- Când cursorul ajunge la un nod care nu mai are fii, se transmite în fișierul destinație caracterul asociat acelui nod și se repoziționează cursorul pe nodul rădăcină al arborelui

Observații

- Arborele Huffman poate fi privit ca un **dicționar**, care asociază fiecărui **caracter** (cheia), un **cod Huffman** (valoarea asociată cheii)
- La codificare se caută după cheie, iar la decodificare se caută după valoare – este un **dicționar bidirecțional**

Concluzii

- Codificarea Huffman reprezintă o tehnică eficientă de compactare a datelor, spaţiul economisit fiind cuprins între 20% şi 90%
- Biții câștigați la compresie sunt utilizați pentru reprezentarea arborelui Huffman

Concluzii

- Acest lucru este adevărat pentru fișiere mici, unde compresia nu este așa de eficientă
- Odată cu creșterea dimensiunii fișierului, arborele de codificare rămâne constant, însă datele sunt micșorate semnificativ