Algoritmul lui Huffman. Aplicații la codificare

Mincu Radu-Ștefan

Cuprins

- **Noțiuni preliminare**
  - Coduri binare
  - Codificare și Decodificare
  - Arbori binari stricți cu ponderi
  - Decodificare cu ABSP
  - Lungimea externă ponderată
- **Algoritmul lui Huffman**
- **Aplicații**
  - Interclasarea optimală a mai mult de două șiruri
  - Calculul entropiei informației
  - Compresia MP3

Coduri binare

- Problema de rezolvat: Avem la dispoziție un fișier ce contine o secvență de 100.000 de caractere. *Cum putem să-l stocăm cât mai compact?*
- Dintre metodele de a reprezenta un astfel de fișier, să ne concentrăm atenția asupra **codurilor binare**.

Coduri binare

- Dat V, un alfabet finit, putem defini o funcție ***c:V->{0,1}*,*** *injectivă.* Această funcție poartă numele de **cod binar**.
- Prin extensia canonică a funcție peste șiruri de caractere din V, putem defini ***c:V*->{0,1}****,* astfel încât pentru orice mesaj alcătuit prin concatenarea caracterelor din V, obținem o codificare acătuită din concatenarea codurilor respective din *{0,1}**, ale fiecărui caracter din mesaj.

Coduri binare

- Codurile se pot clasifica în
  - coduri de lungime variabilă
  - coduri de lungime fixă.
- Definim **frecvența** de apariție a unui caracter într-o secvență ca fiind **numărul de apariții** al caracterului în secvență (eventual exprimată ca și procent din numărul total de caractere al secvenței).

Coduri binare

- Să presupunem că secvența noastră de 100.000 de caractere conține doar simbolurile **a,b,c,d,e,f**.
- O primă idee de codificare ar fi să asociem fiecărui simbol un cod de 3 biți, spre exemplu:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime fixă	000	001	010	011	100	101

Exemplu de cod de lungime fixă [1].

- Observăm că lungimea fișierului codificat în acest mod va fi de 3 biți * 100.000 caractere = 300.000 biți. *Există oare și o soluție mai eficientă?*

Coduri binare

- Am putea utiliza un cod de lungime variabilă în care asociem caracterelor frecvente coduri de lungime mai scurtă, în detrimentul obținerii de coduri mai lungi pentru caracterele mai puțin frecvente. Scopul nostru este minimizarea lungimii totale a mesajului codificat.

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime fixă	000	001	010	011	100	101
Cod asociat de lungime variabilă	0	101	100	111	1101	1100

Exemplu de cod de lungime fixă, respectiv variabilă preluat din [1].

- Un simplu calcul arată că acest cod este mai bun decât cel de lungime fixă: (***45 * 1 + 13 * 3 + 12 * 3 + 16 *3 + 9 * 4 + 5 * 4***) *1,000 = ***224.000 biți***

Codificare și Decodificare

- În ceea ce privește codurile ne interesează două operații:
  - **Codificarea** (pentru **m** din ***V**** *obținerea lui* ***c(m)*** *din* ***{0,1}****)
  - **Decodificarea** (pentru ***c(m)*** *din* ***{0,1}**** *obținerea lui* **m** din ***V****)

- Codificarea se obține mereu prin concatenarea codurilor asociate, însă
- Dacă pentru codurile de lungime fixă **decodificarea** este o operație trivială
- (Pentru decodificare mesajul este segmentat în bucăți date de lungimea unui cuvintelor cod și aplicată inversa funcției **c**),
- Nu se poate spune același lucru și despre **decodificarea mesajelor codificate de tip cod variabil**.

Codificare și Decodificare

- Exemplu: **aabe**
- Codificare:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime fixă	000	001	010	011	100	101

- 000.000.001.100
- 000000001100

Codificare și Decodificare

- Exemplu: **000000001100**
- Decodificare:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime fixă	000	001	010	011	100	101

- 000.000.001.100
- a.a.b.e
- aabe

Codificare și Decodificare

- Exemplu: **aabe**
- Codificare:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime variabilă	0	101	100	111	1101	1100

- 0.0.101.1101
- 001011101

Codificare și Decodificare

- Exemplu: **001011101**
- Decodificare:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime variabilă	0	101	100	111	1101	1100

- Putem segmenta? Cum decodificăm?

Codificare și Decodificare

- Proprietatea **prefix**:
  - Un cod se bucură de proprietatea prefix ddacă niciun cuvânt cod nu reprezintă prefixul altui cuvânt cod.
  - Altfel spus, *pentru oricare x,y din V cu x ≠ y, c(x) nu este prefix al lui c(y).*
  - *Sau pentru oricare x,y din V cu x ≠ y și l(x)≤l(y): c(y) ≠ c(x)·w, cu w din {0,1}*.*
- Un cod care satisface proprietatea prefix se numește **cod prefix** (mai corect, cod liber de prefixe).
- Codurile prefix ne rezolvă problema anterioară, întrucât secvența codificată nu mai este ambiguă.
- Decodificarea se reduce la identificarea primului cuvânt cod și găsirea caracterului asociat urmată de aplicarea aceleiași operații pe restul secvenței.

Codificare și Decodificare

- Exemplu: **001011101**
- Decodificare:

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod asociat de lungime variabilă	0	101	100	111	1101	1100

- 0.01011101 = a.01011101
- a.0.1011101 = a.a.1011101
- a.a.101.1101 = a.a.b.1101
- a.a.b.1101 = a.a.b.e
- aabe

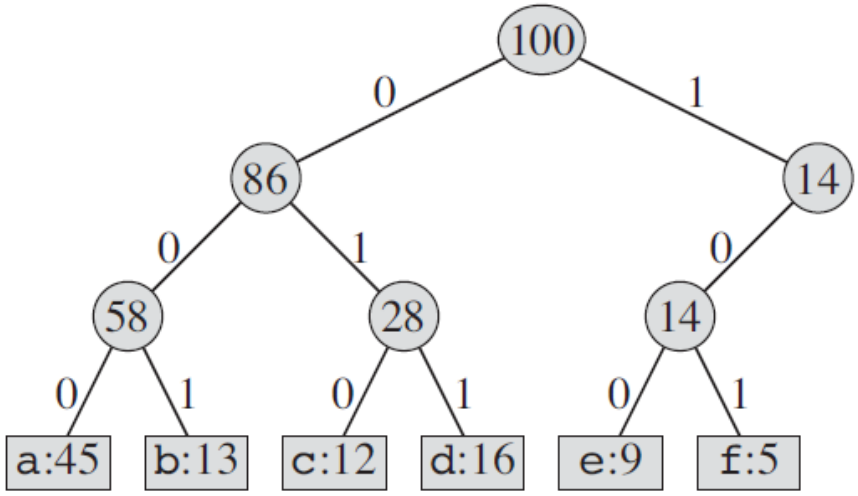
Codificare și Decodificare

- Observăm că avem nevoie de o reprezentare mai bună pentru codurile prefix în așa fel încât să fie ușor de identificat cuvintele cod.
- (Căutarea fiecărei secvențe de biți din cod în întreaga mulțime de cuvinte cod este ineficientă)
- Această reprezentare este dată de **arborele binar strict cu ponderi**.

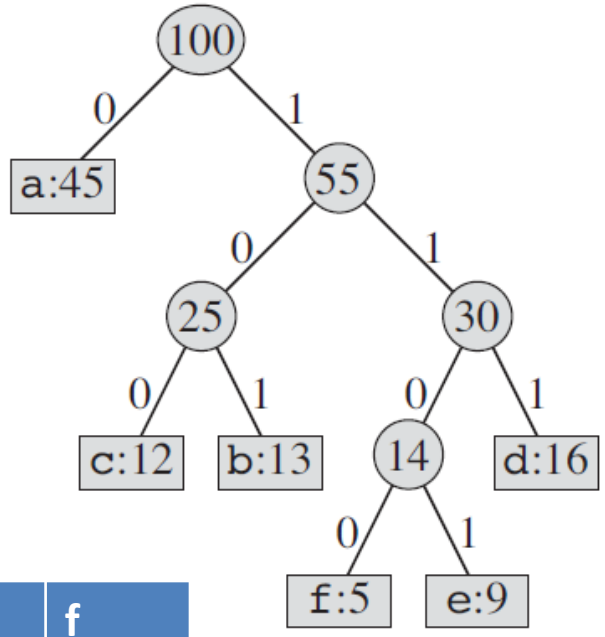
Arbore binar strict cu ponderi

- Arborele binar strict cu ponderi
  - **Strict** = fiecare nod are 0 sau doi fii
  - **Cu ponderi** = frunzele arborelui dețin în plus câmpuri de informație denumite ponderi.
  - În cazul nostru,
    - informația din **frunze** = **caracterele** ce trebuiesc codificate
    - **Ponderile** = **frecvențele** apariției acestor caractere
  - În această reprezentare, cuvintele cod sunt determinate de parcurgerea **drumului dinspre rădăcină către frunze**, unde **0** înseamnă continuarea drumului către **fiul stâng**, iar **1** către **fiul drept**.

Arbore binar strict cu ponderi



(a)



(b)

	a	b	c	d	e	f
Frecvență (mii)	45	13	12	16	9	5
Cod de lungime fixă	000	001	010	011	100	101
Cod de lungime variabilă	0	101	100	111	1101	1100

Imagini din [1].

Arbore binar strict cu ponderi

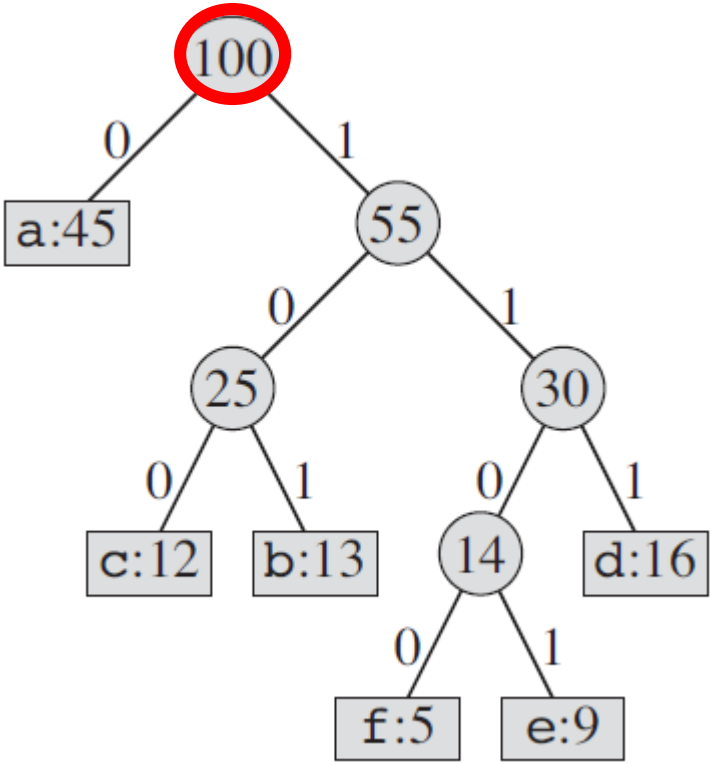
- Având la dispoziție arborele binar strict cu ponderi, decodificarea unui cod prefix revine la citirea secvenței codificate și parcurgerea arborelui dinspre rădăcină către frunze conform secvenței de biți.
- În momentul atingerii unei frunze înseamnă că am decodificat caracterul respectiv și algoritmul se reia pentru restul secvenței.

Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- 001011101

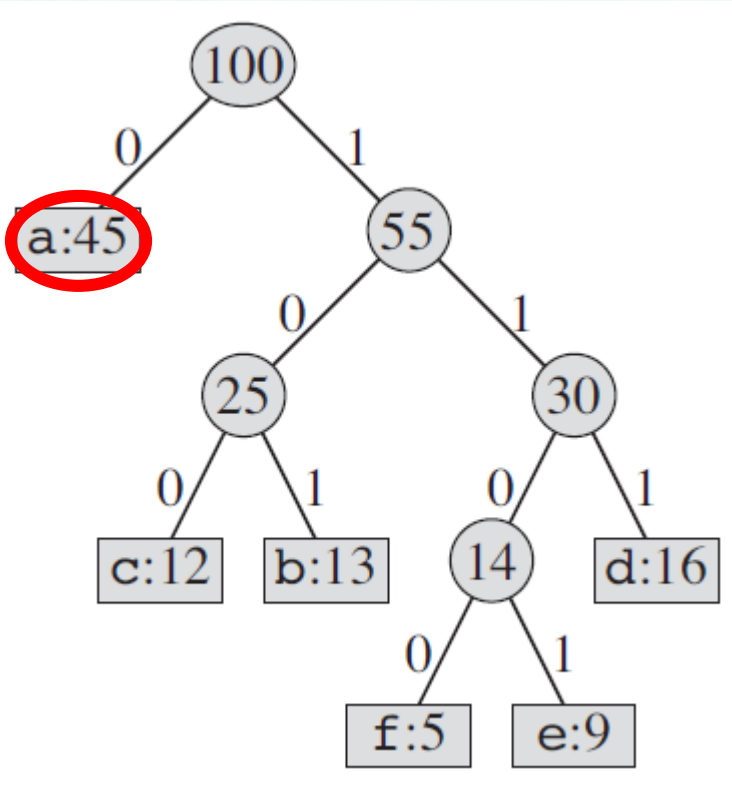


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **0**01011101
- a

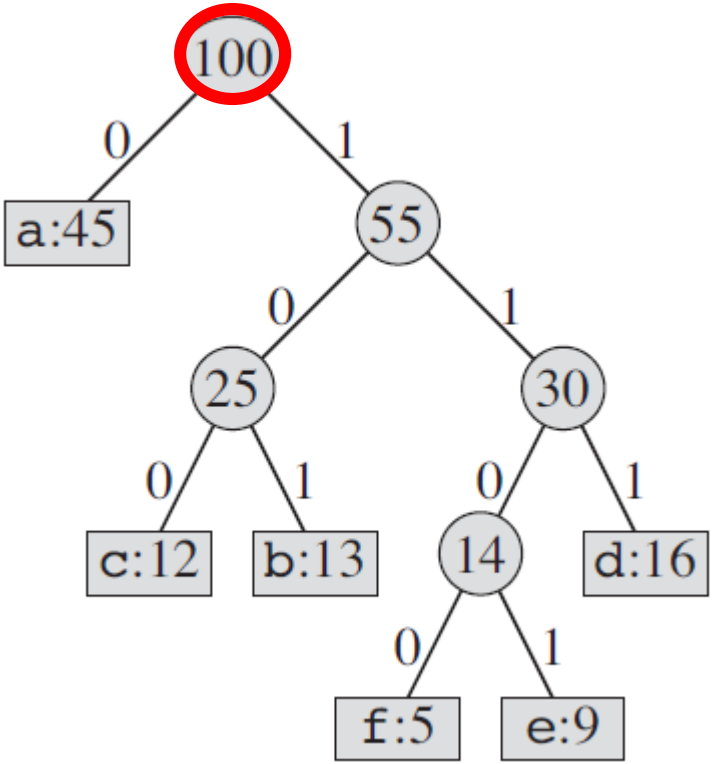


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **0**01011101
- a

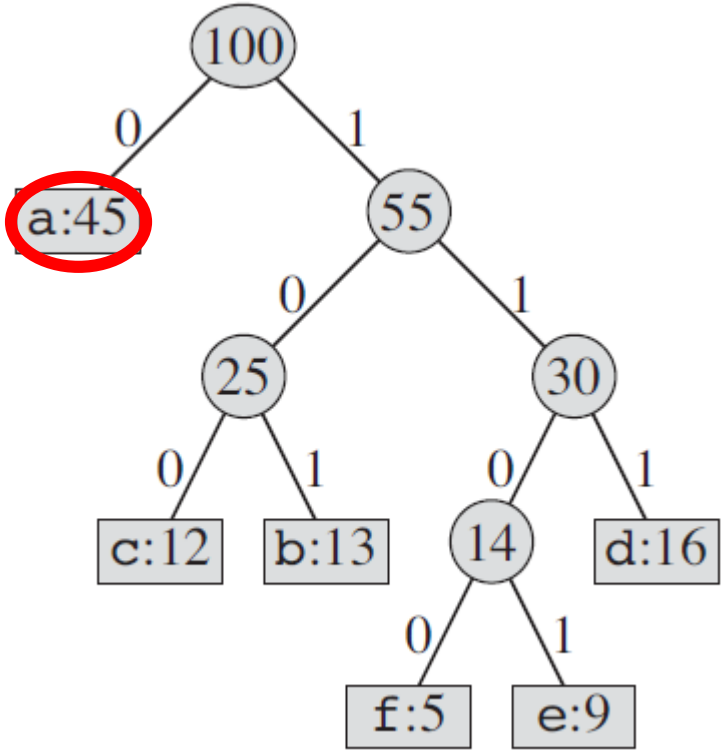


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **00**1011101
- aa

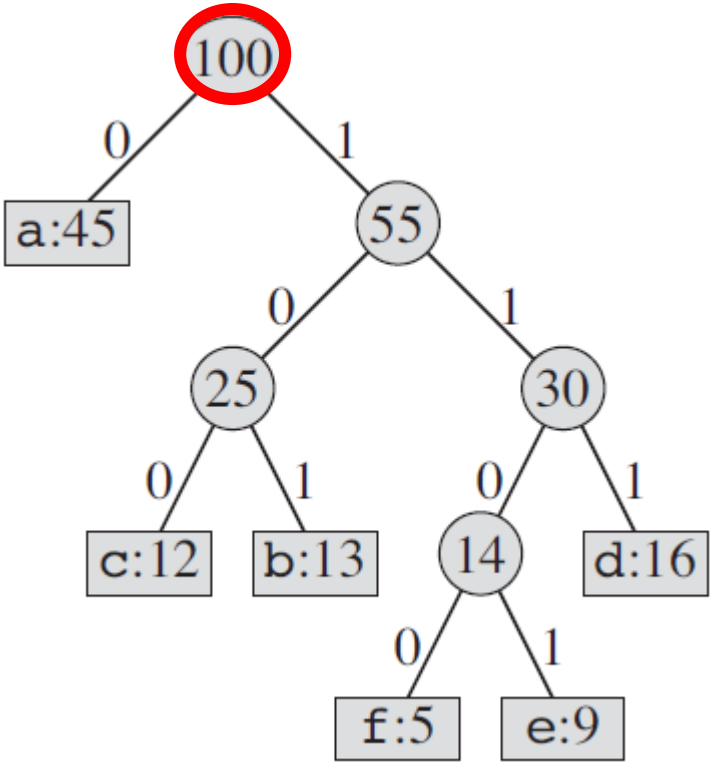


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **00**1011101
- aa

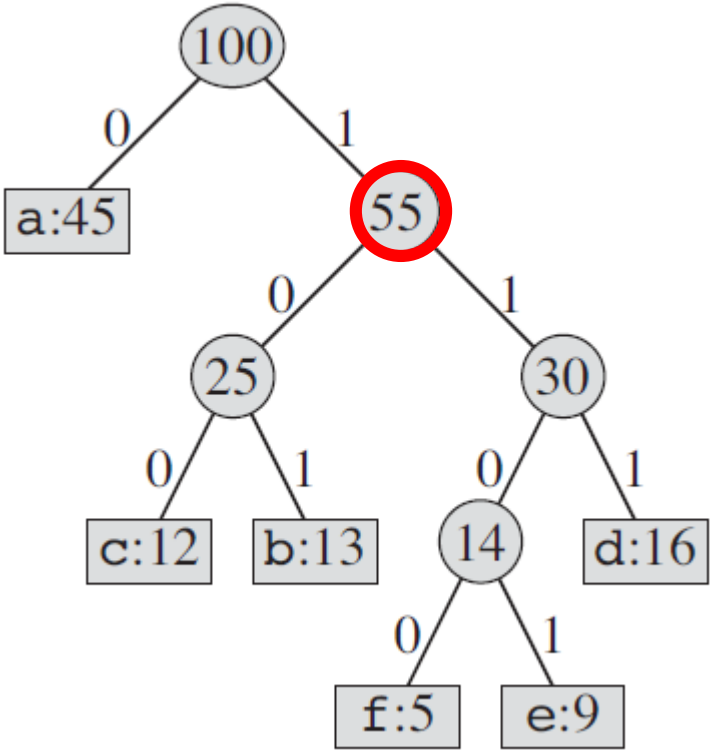


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **001**011101
- aa

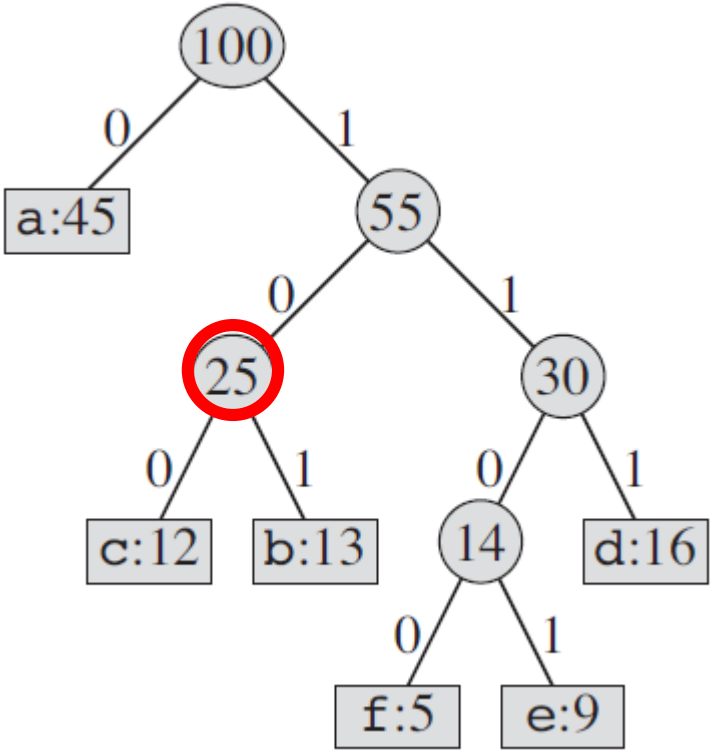


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **0010**11101
- aa

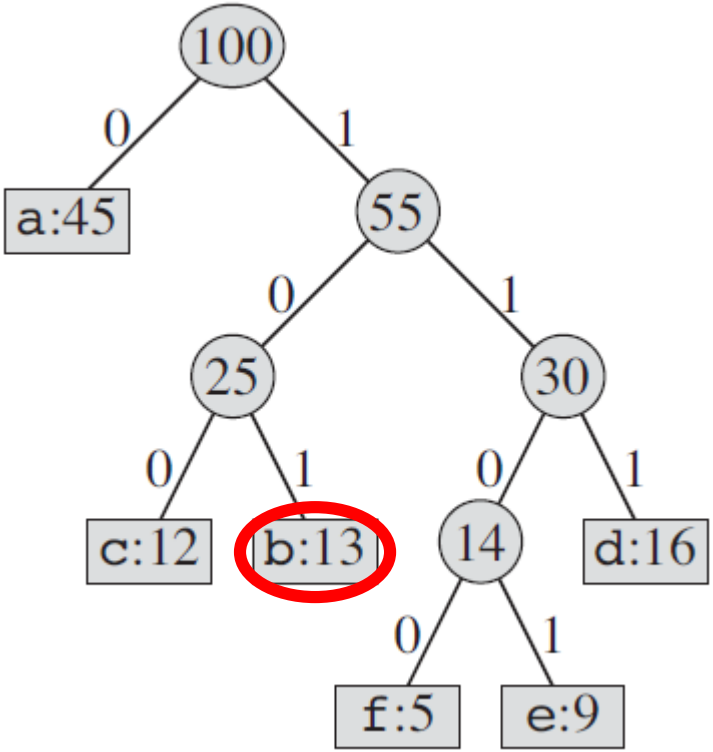


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **00101**1101
- aab

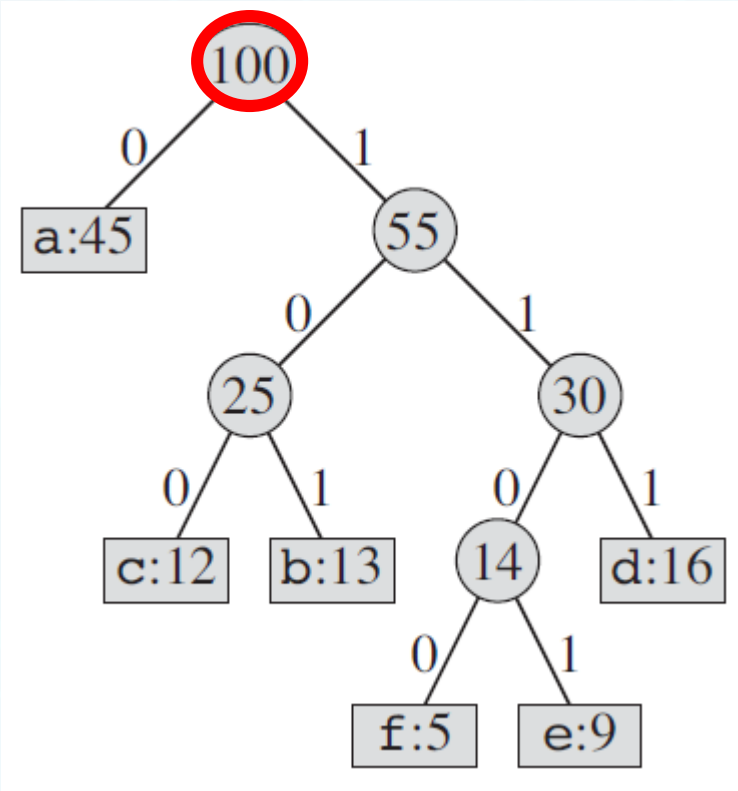


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **00101**1101
- aab

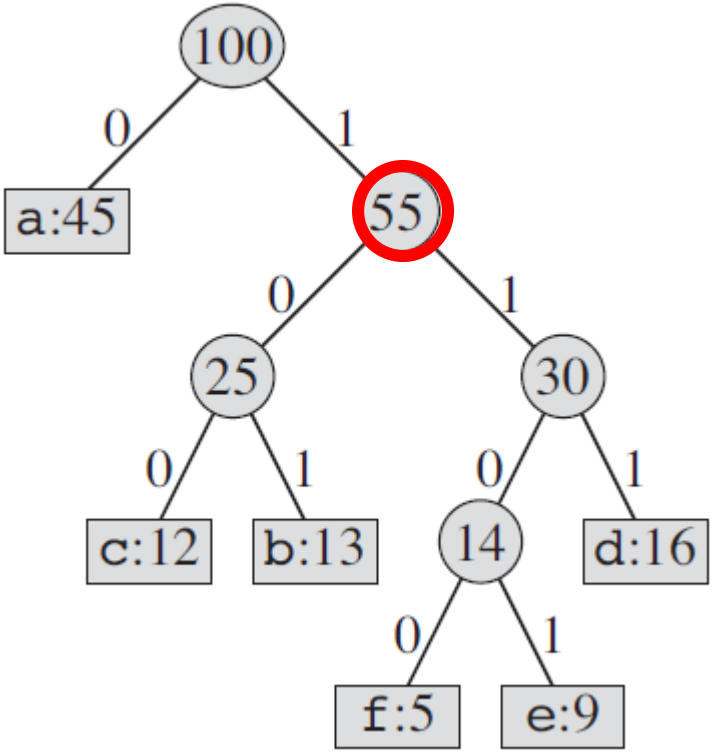


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **001011**101
- aab



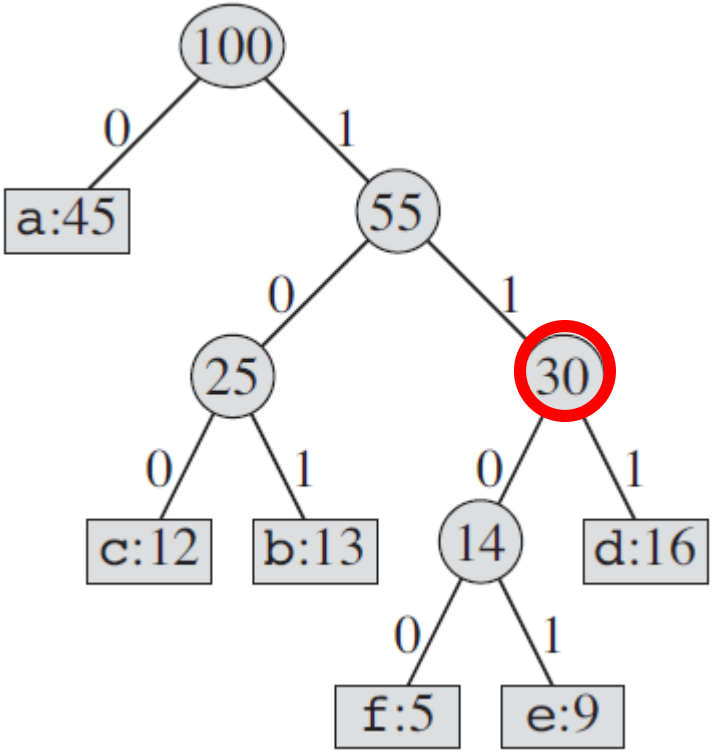
Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **001011101**

- aab

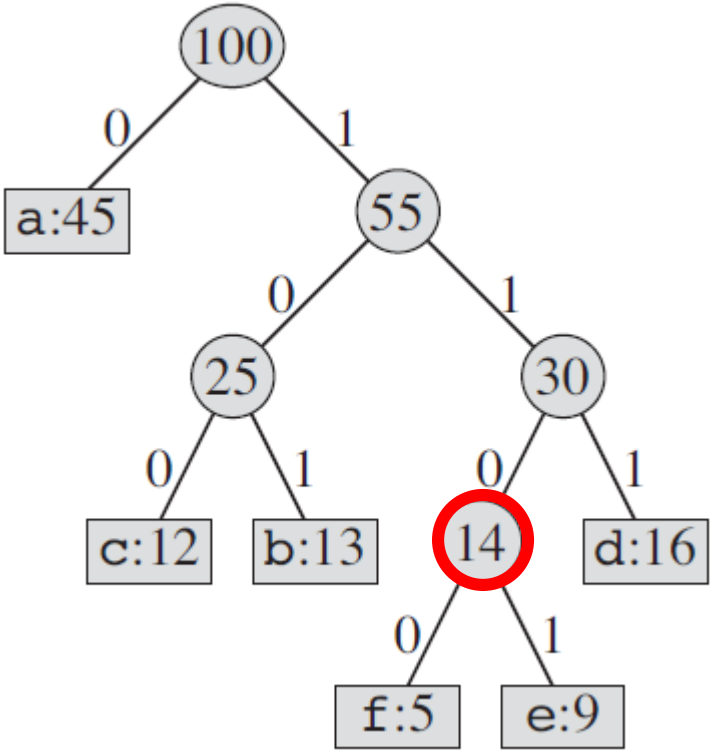


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **001011101**
- aab

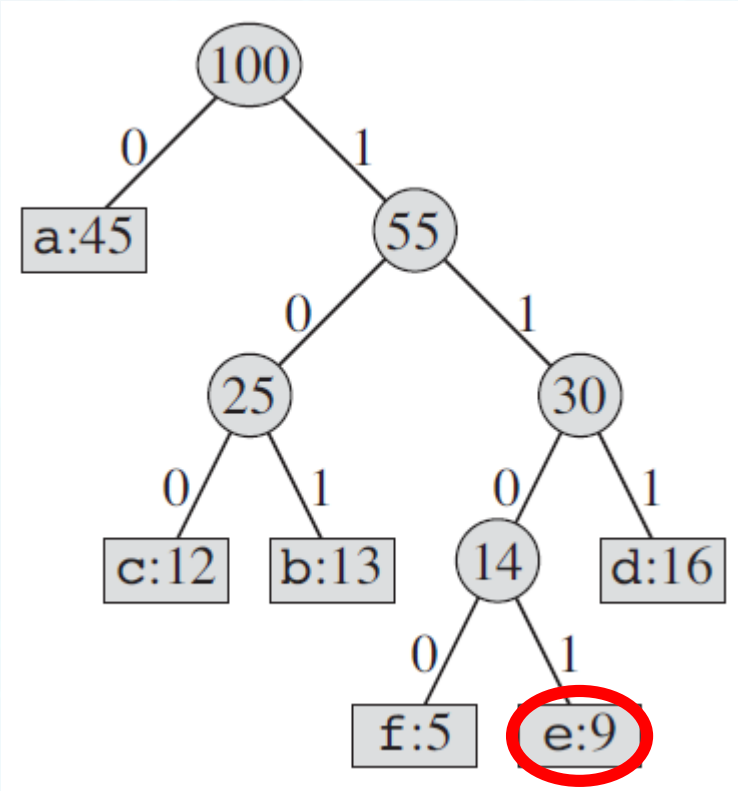


Decodificare cu ABSP

- Exemplu: **001011101**
- Decodificare:

a	b	c	d	e	f
45	13	12	16	9	5
0	101	100	111	1101	1100

- **001011101**
- aabe



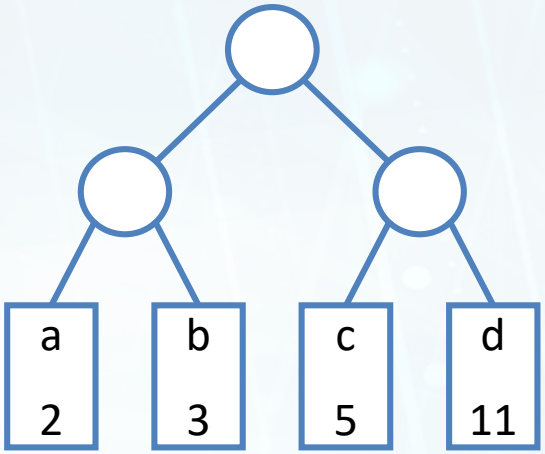
Lungimea externă ponderată

- Dat un arbore binar strict T, corespunzător unui cod prefix, putem calcula numărul de biți necesar codificării unui fișier.
- Considerăm funcția **frecvență(x)** care produce frecvența caracterului **x** și funcția **lungime(c(x))**, reprezentând lungimea codului asociat lui **x**
  - (care este și lungimea drumului dinspre rădăcină către frunza etichetată cu **x**).
- Cu aceste notații, definim
  - **L**(T) = **Σ** frecvență(x) * lungime(c(x)),

  pt. fiecare x din V

  Această mărime poartă numele de ***lungime externă ponderată***.

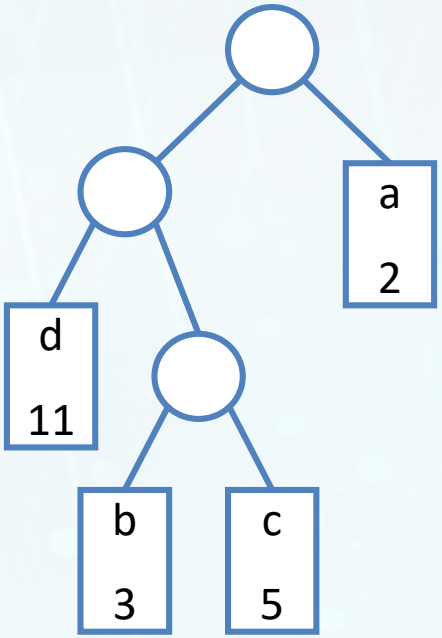
Lungimea externă ponderată



2*2 + 3*2 + 5*2 + 11*2

4 + 6 + 10 + 22

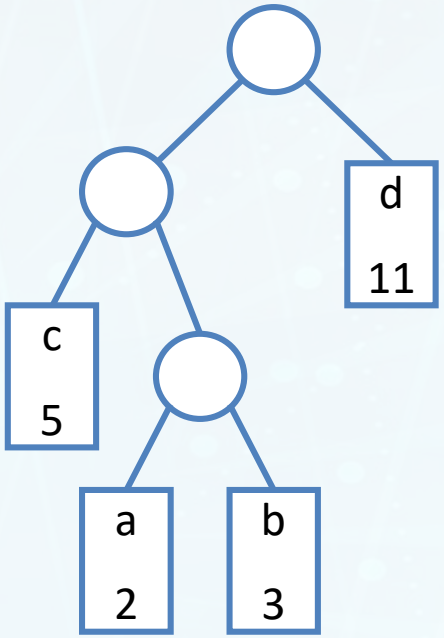
(A) L=42



11*2 + 3*3 + 5*3 + 2*1

22 + 9 + 15 + 2

(B) L=48



5*2 + 2*3 + 3*3 + 11*1

10 + 6 + 9 + 11

(C) L=36

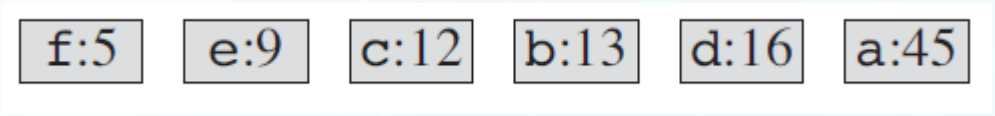
Lungimea externă ponderată

- Găsirea codificării de lungime minimă pentru o secvență inițial dată înseamnă găsirea unui arbore binar strict de ***lungime externă ponderată minimă***.
- Acest arbore nu este unic.

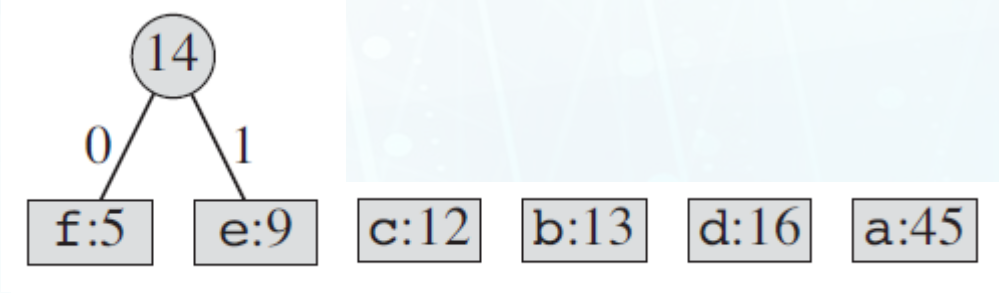
Algoritmul lui Huffman

- Un algoritm Greedy inventat de Huffman găsește un astfel de arbore binar strict de ***lungime externă ponderată minimă***.
- Pașii algoritmului:
  1. *Se creează o pădure de **n** arbori binari stricţi, fiecare de tip frunză, cu câte o pondere w_i asociată.*
  2. *Se leagă doi subarbori T_i și T_j de pondere minimă, devenind fii unui nod intern cu ponderea $w_i + w_j$.*
  3. *Se continuă algoritmul cu pasul 2 până când există un singur arbore. Acesta este arborele Huffman rezultat și este obținut în n-1 pași.*

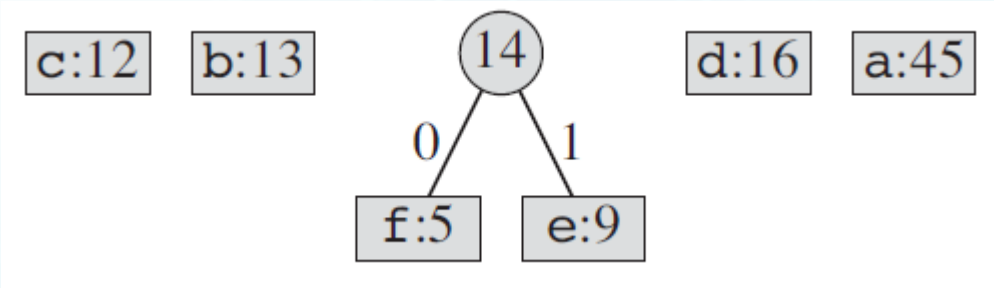
Algoritmul lui Huffman



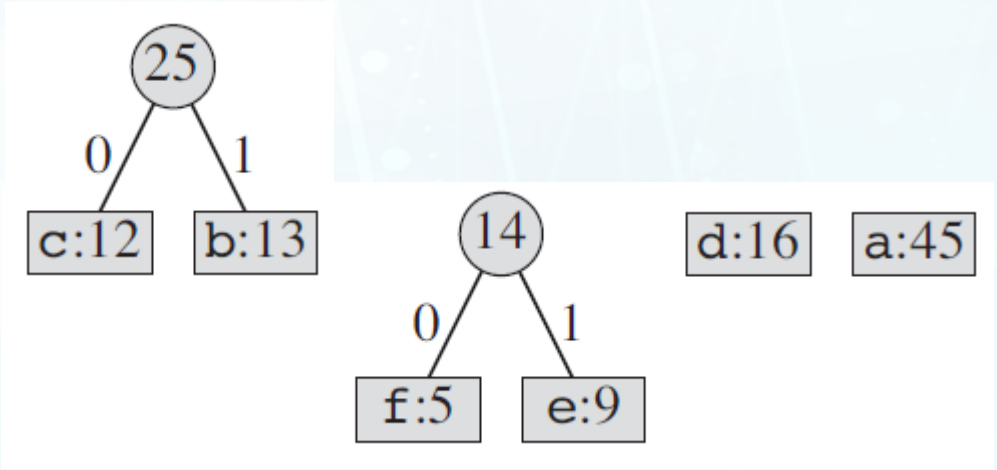
Algoritmul lui Huffman



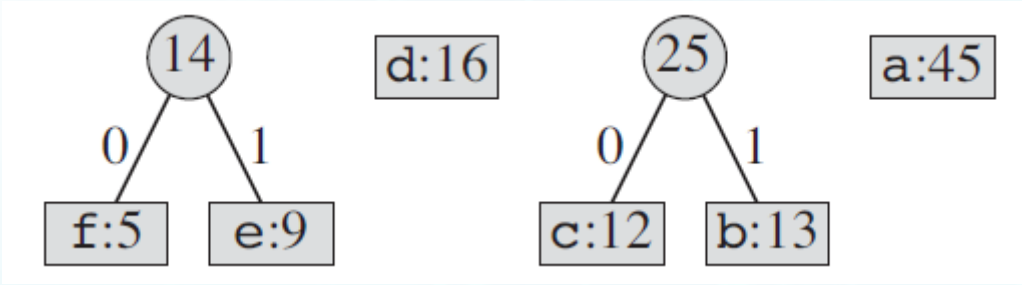
Algoritmul lui Huffman



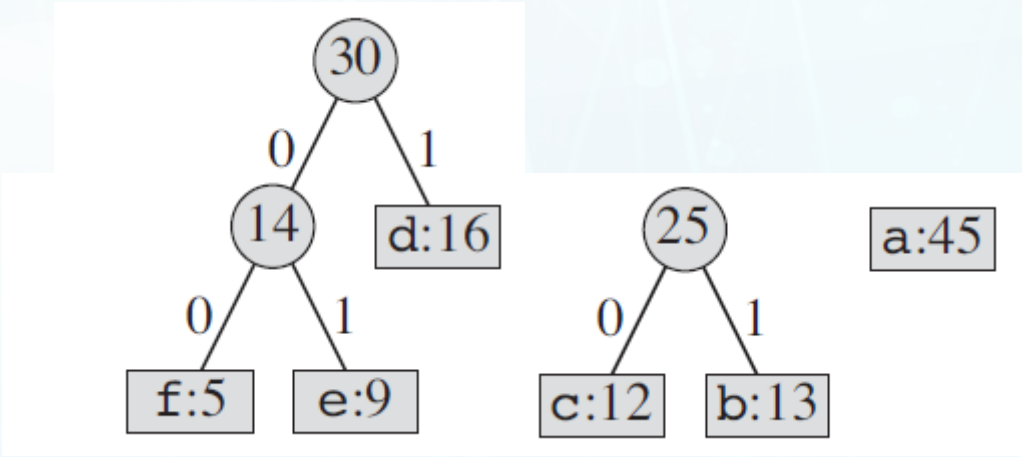
Algoritmul lui Huffman



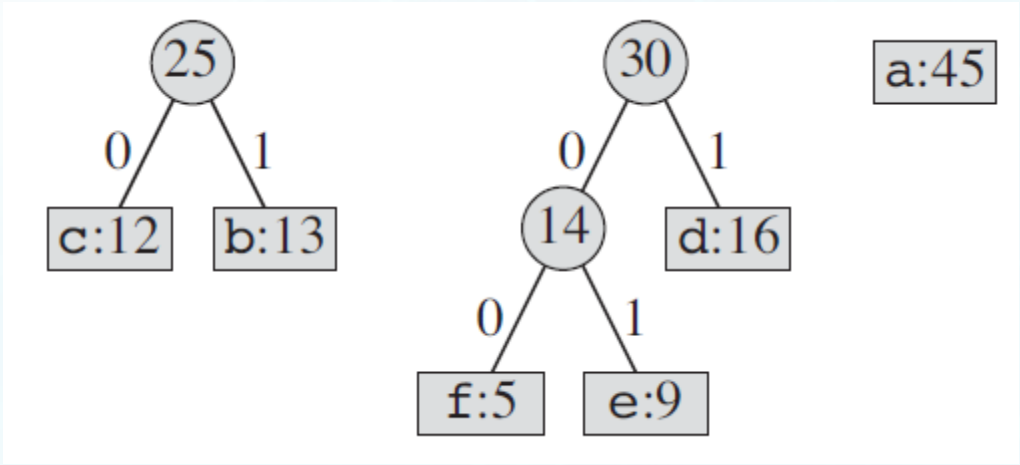
Algoritmul lui Huffman



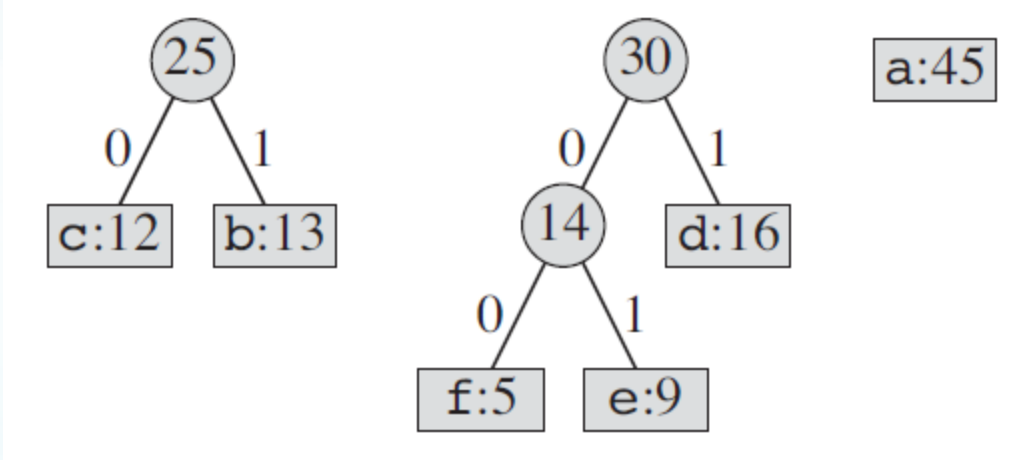
Algoritmul lui Huffman



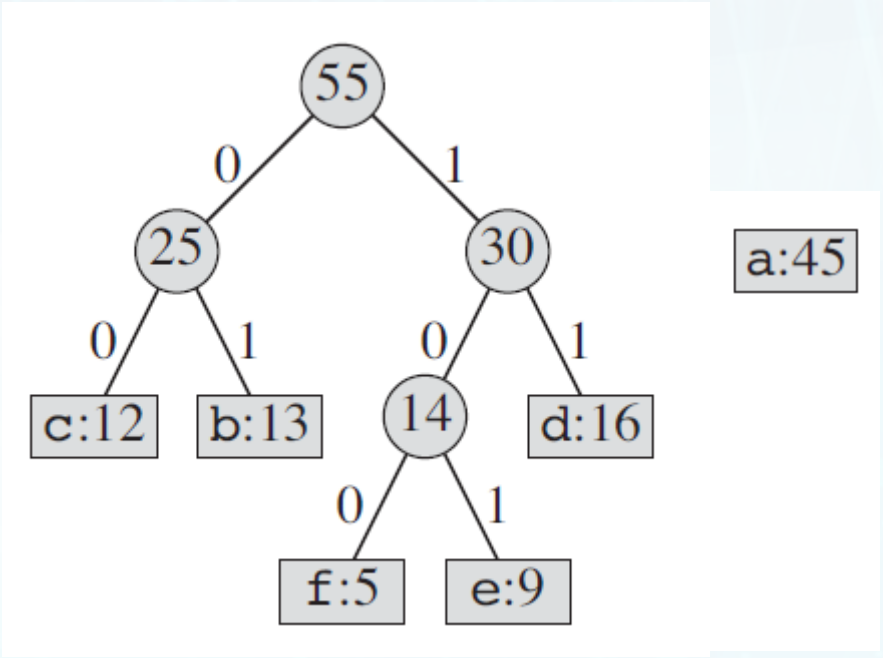
Algoritmul lui Huffman



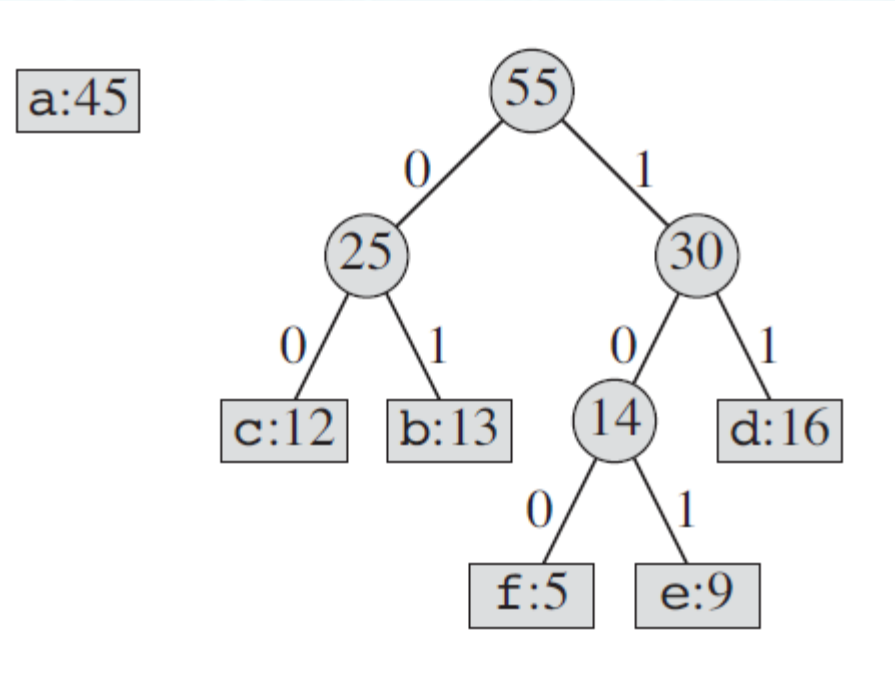
Algoritmul lui Huffman



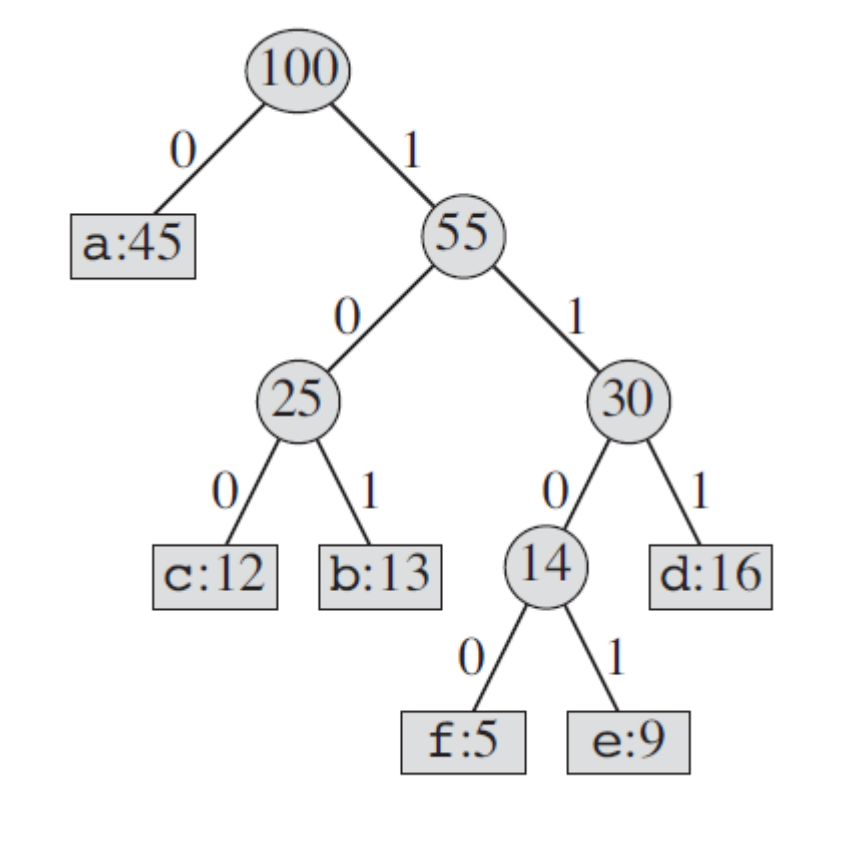
Algoritmul lui Huffman



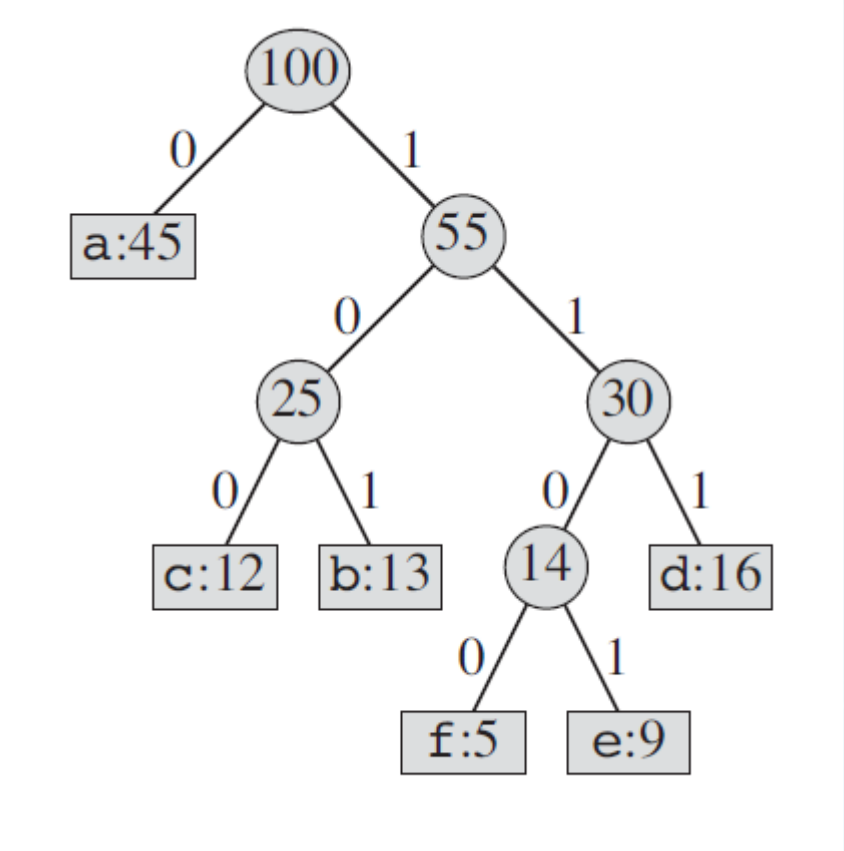
Algoritmul lui Huffman



Algoritmul lui Huffman



Algoritmul lui Huffman



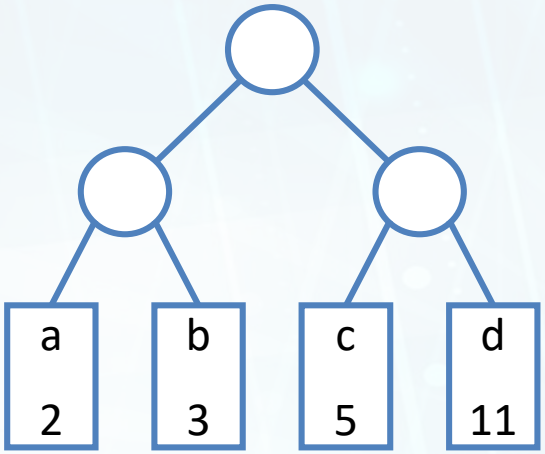
a	0
b	101
c	100
d	111
e	1101
f	1100

Exemplu din [1].

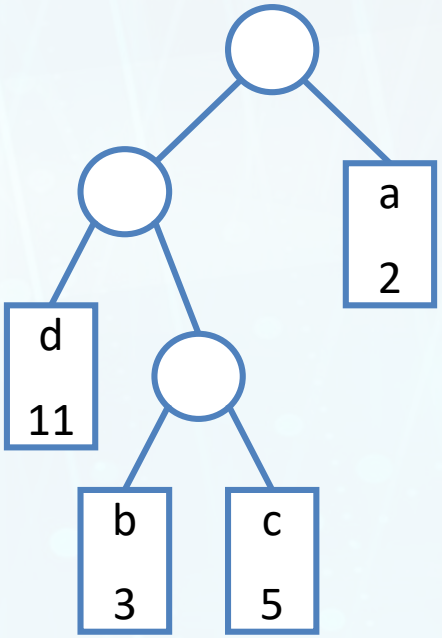
Algoritmul lui Huffman

- Observăm că modul de construcție al arborelui este **bottom-up**, dinspre frunze spre rădăcină.
- Partea de greedy a algoritmului constă în alegerea la fiecare pas a subarborilor de pondere minimă.
- Timpul de rulare al algorimului depinde de implementarea structurii prin care la fiecare pas extragem subarborii de pondere minimă. Pentru aceasta putem utiliza o structură de coadă de priorități implementată printr-un min-heap.
- Din moment ce operațiile pe min-heap rulează în *O(lg n)* și există *n-1* pași, timpul de construcție al arborelui Huffman este **O(n lg n)** folosind această strategie.

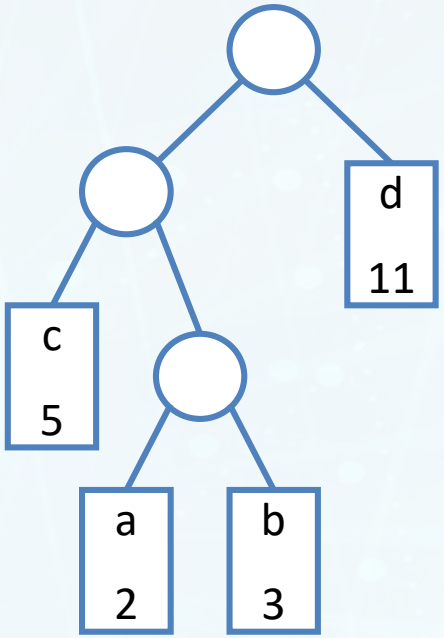
Algoritmul lui Huffman



(A) L=42

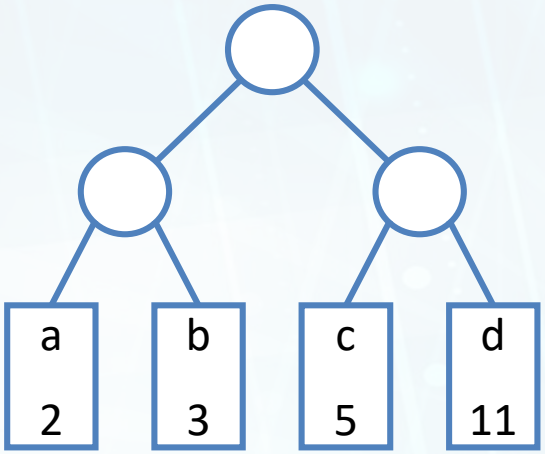


(B) L=48

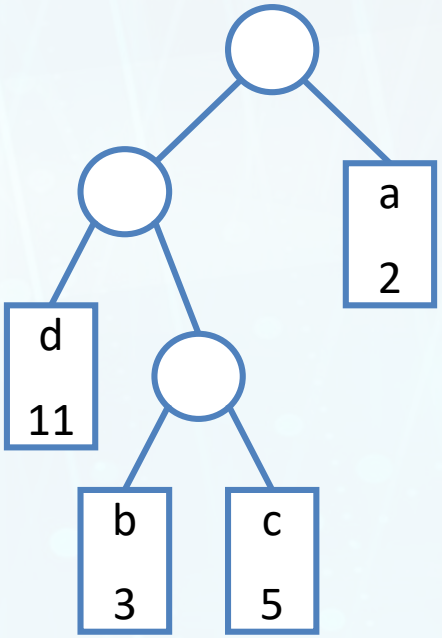


(C) L=36

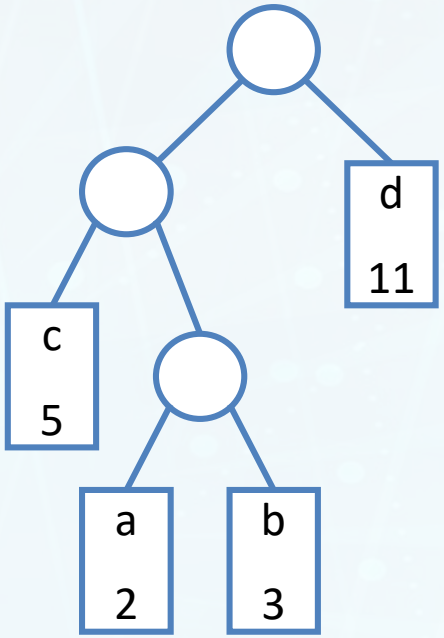
Algoritmul lui Huffman



(A) L=42



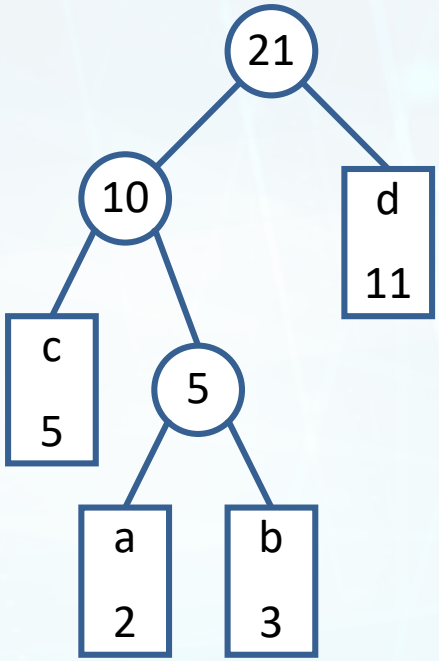
(B) L=48



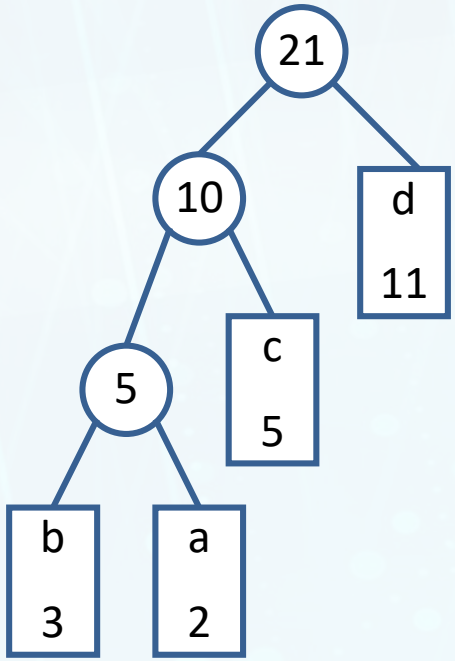
Arbore Huffman

(C) L=36

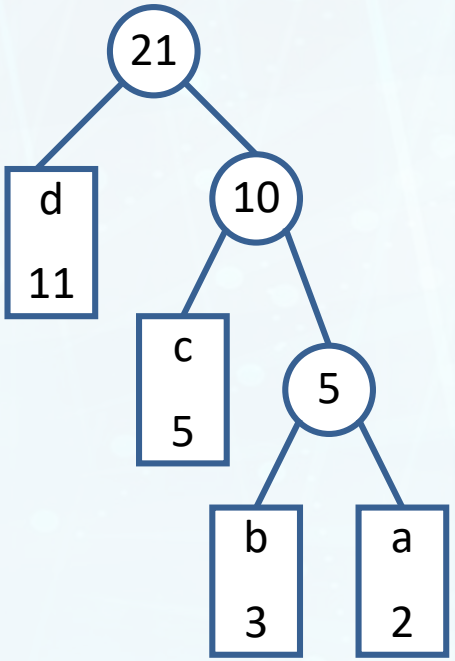
Algoritmul lui Huffman



(C) L=36



(D) L=36



(E) L=36

Exemple din [3].

Aplicații

- **Interclasarea optimală a mai mult de două șiruri**
- **Calculul entropiei informației**
- **Compresia MP3**

Interclasarea optimală a mai mult de două șiruri

- Conform [3], dacă se dau *n (n>2)* şiruri ordonate crescător, S_1, S_2,…, S_n , cu lungimile respective l_1, l_2,…, l_n. Vrem să le interclasăm pe toate, obţinând un şir *S* ordonat crescător ce conţine toate elementele şirurilor iniţiale.
- Putem face aceasta interclasând pe rând câte două şiruri (operaţia de **interclasare cu două surse**). La fiecare pas se reduce cu *1* numărul şirurilor sursă, deci vom obţine în cele din urmă un singur şir.
- Fiecare operaţie de interclasare este costisitoare în termeni de **mutări**: interclasarea lui S_i cu S_j (să notăm rezultatul cu $S_i\&S_j$) ne costă l_i+l_j mutări.

Interclasarea optimală a mai mult de două șiruri

- Fie şirurile S_1, S_2, S_3 de lungime

90, 40 şi respectiv 10.

- Să considerăm următoarele două strategii de interclasare, cu costurile lor:
  1. $(S_1 \& S_2) \& S_3$ are costul total

  (90+40)+((90+40)+10)=(90+40)*2+10=270.

  2. $(S_2 \& S_3) \& S_1$ are costul total

  (40+10)+((40+10)+90)=(40+10)*2+90=190.
- Deci, a doua strategie este mult mai performantă decât prima.

Interclasarea optimală a mai mult de două șiruri

- Fie şirurile S_1, S_2, S_3 de lungime

90, 40 şi respectiv 10.

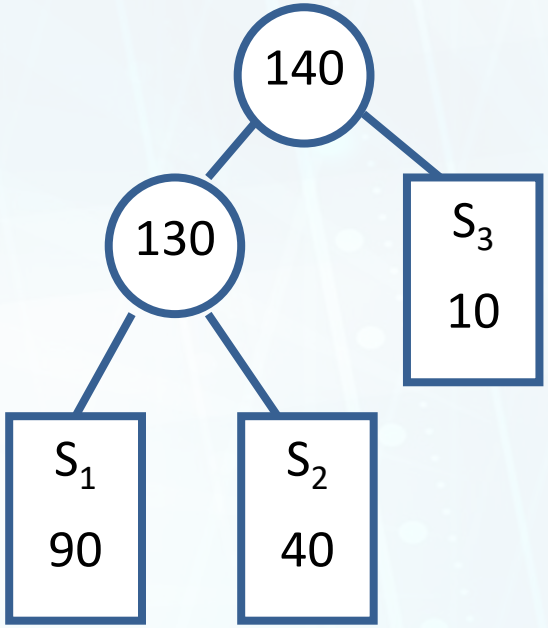
- Să considerăm următoarele două strategii de interclasare, cu costurile lor:
  1. $(S_1 \& S_2) \& S_3$ are costul total

  (90+40)+((90+40)+10)=(90+40)*2+10=270.

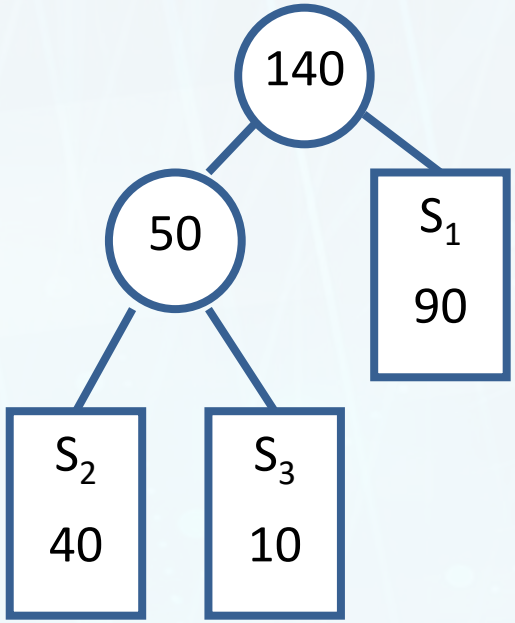
  2. $(S_2 \& S_3) \& S_1$ are costul total

  (40+10)+((40+10)+90)=(40+10)*2+90=190.
- Deci, a doua strategie este mult mai performantă decât prima.

Interclasarea optimală a mai mult de două șiruri



L_E =2*90+2*40+10=270



L_E=2*40+2*10+90=190

Ordinea optimă de interclasare este dată de algoritmul lui Huffman [3].

Calculul entropiei informației

- În [2] apare următorul scenariu:
- 3 cai de la hipodrom nu au concurat niciodată unul împotriva celuilalt. Având la dispoziție informația ultimelor 200 de curse, care este calul cel mai **predictibil**?

Outcome	Aurora	Whirlwind	Phantasm
first	0.15	0.30	0.20
second	0.10	0.05	0.30
third	0.70	0.25	0.30
other	0.05	0.40	0.20

- O încercare de răspuns se bazează pe proprietatea de **compresibilitate**.
- Vom scrie istoricul fiecărui cal utlizând un șir de 200 de caractere, iar numărul total de biți utilizat poate fi calculat conform **algoritmului lui Huffman**.
- Lungimea externă ponderată va fi:
  - 290 biți pentru Aurora
  - 380 biți pentru Whirlwind
  - 420 biți pentru Phantasm

Calculul entropiei informației

- În [2] apare următorul scenariu:
- 3 cai de la hipodrom nu au concurat niciodată unul împotriva celuilalt. Având la dispoziție informația ultimelor 200 de curse, care este calul cel mai **predictibil**?

Outcome	Aurora	Whirlwind	Phantasm
first	0.15	0.30	0.20
second	0.10	0.05	0.30
third	0.70	0.25	0.30
other	0.05	0.40	0.20

- Lungimea externă ponderată va fi:
  - 290 biți pentru Aurora => caracterul previzibil cel mai pronunțat
  - 380 biți pentru Whirlwind
  - 420 biți pentru Phantasm
- Caracterul aleator poate fi cuantificat pe baza compresibilității.
- Mai compresibil = mai puțin aleator = mai predictibil.

Compresia MP3

- Conform [2], algoritmul de codificare a semnalului audio MP3 este:
  1. Se **eșantionează** semnalul la **intervale echidistante**, producând o secvență de **numere reale**.
  2. Se **cuantifică** fiecare **valoare reală** printr-o **aproximare** dintr-un set finit, set ales în așa fel încât exploatează limitările percepției umane. Scopul este ca secvența rezultată să fie greu de distins de urechea umană față de secvența originală.
  3. Se aplică o **codificare binară** peste această nouă secvență.
- La ultimul pas se folosește algoritmul lui Huffman.

Bibliografie

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. ***Introduction to Algorithms, Third Edition*** **(3rd ed.)**. The MIT Press.
2. Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh Vazirani. 2006. ***Algorithms*** **(1 ed.)**. McGraw-Hill, Inc., New York, NY, USA.
3. R. Ceterchi, **Curs Algoritmi și structuri de date**, 2016