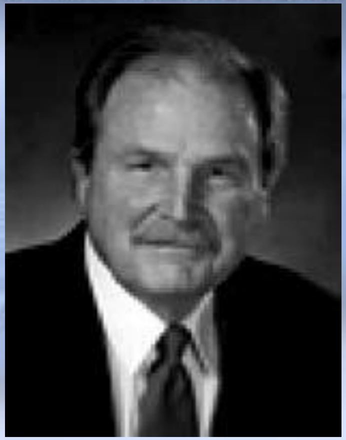
Algoritmul lui Huffman.
Aplicatii la Codificarea Binara.

Liliana Cojocaru

Facultatea de Informatica

Universitatea Alexandru Ioan Cuza

Algoritmul lui Huffman *I. Generalitati*



- A fost introdus de David Huffman in 1952.
- Este un algoritm greedy care construieste o codicare prefix optima numita codificare Huffman.
- Este un algoritm de codificare si compresie a datelor .
- Se bazeaza pe construirea unui arbore binar ponderat (optim) folosind metoda greedy.

Definitie Fie un arbore binar continand cheile $k_1, k_2, \ldots, k_n$, astfel incat fiecare cheie k_i are asociata o "probabilitate de aparitie" p_i, $0 \leq p_i \leq 1$, unde $i = 1 \ldots n$ si $p_1 + p_2 + \ldots + p_n = 1$, se numeste arborele binar optim daca: $D_P = p_1 \cdot d(k_1) + \ldots + p_n \cdot d(k_n)$ este minima

D_P = "drum ponderat", fiind suma lungimilor drumurilor de la radacina la fiecare nod, ponderate cu probabilitatile de acces la noduri.

Algoritmul lui Huffman *I. Generalitati*

Intuitiv, pentru ca drumul ponderat $D_P = p_1 \cdot d(k_1) + \ldots + p_n \cdot d(k_n)$ sa fie minim, trebuie ca o probabilitate de acces (p_i) mai mare (adica o cheie cautata des) sa fie insotita de o distanta mai mica $d(k_i)$ de la radacina arborelui la cheie $\Rightarrow$ o probabilitate de acces (p_i) mai mica (o cheie cautata mai rar) va fi insotita de o distanta mare $d(k_i)$ de la radacina la cheie (Lemma 1).

Aceasta idee sta la baza construirii arborelui de codificare Huffman (aplicatie a arborilor binari optimi).

Algoritmul lui Huffman *I. Generalitati*

- Sa presupunem ca dorim sa codificam un mesaj, fiecare caracter in acest mesaj are o frecventa de apariti.

- In codificarea Huffman caracterele care au o frecventa mai mare de aparitie vor primi o codificare mai scurta, codificarea fiind data de lungimea drumului de la radacina la frunza etichetata cu caracterul respectiv (intr-un arbore Huffman).

- Datorita proprietatii de optimalitate Algoritmul lui Huffman conduce la o codificarea a unui sir de caractere (fisier) pe un numar de biti semnificativ mai mic decat cel initial, de unde si aplicabilitatea acestui algoritm in compresia datelor.

Metoda Greedy

- Este folosita pentru rezolvarea problemelor de optimizare in care optimul global se determina din estimari succesive ale optimului local.

Exemplu Dintr-o multime *A* se cere sa se determine o submultime *B* (a lui A) care sa satisfaca anumite conditii.

- O problema este rezolvabila prin metoda Greedy daca satisface proprietatea: *Daca B este o solutie si* $C \subset B$*, atunci si C este o solutie.*

- Initial se considera B = ∅,
- Se adauga succesiv elemente din *A* in *B*, pana la un optim local.
  Daca se ajunge la concluzia ca succesiunea de optimuri locale conduce la un optimul global, atunci metoda Greedy este aplicabila.

Se pleaca de la solutia vida pentru *B* si se ia pe rand cate un element din *A*. Daca elementul ales indeplineste conditia de optim local, el este introdus in *B*.

Algoritmi Greedy

Pentru a rezolva problema de *optimizare*, cautam o *solutie posibila* care sa *optimizeze* valoarea *functiei obiectiv*. *Functia obiectiv* da valoarea unei solutii. Ex. lungimea drumului pe care l-am gasit, costul arborelui, si pe care urmarim sa o optimizam (minimizam/maximizam)

Initial, multimea candidatilor selectati B este vida. La fiecare pas, se adauga la B cel mai promitator candidat, conform unei *functii de selectie*. Daca, dupa o astfel de adaugare, multimea de candidati selectati nu mai este *fezabila* , se elimina ultimul candidat.

Daca, dupa adaugare, multimea de candidati selectati este *fezabila*, ultimul candidat adaugat va ramane in ea. De fiecare data cand largim multimea candidatilor selectati, verificam daca aceasta multime nu constituie o *solutie posibila* a problemei. Daca algoritmul greedy functioneaza corect, prima solutie gasita va fi o *solutie optima* a problemei.

Solutia optima nu este in mod necesar unica: se poate ca *functia obiectiv* sa aiba aceeasi valoare optima pentru mai multe *solutii posibile*.

Algoritmi Greedy

Algoritm greedy - pseudocod

function *greedy*(C) // C este multimea candidatilor
$S \leftarrow \emptyset$ // S este multimea in care construim solutia
  while not *solutie*(S) **and** $C \neq \emptyset$ **do**
    $x \leftarrow$ un element din C care maximizeaza/minimizeaza
    select(x)
    $C \leftarrow C - \{x\}$
    if *fezabil* ($S \cup \{x\}$) **then** $S \leftarrow S \cup \{x\}$
 if *solutie* (S) **then return** S
 else return "nu exista solutie"

Algoritmul lui Huffman *II. Etape*

Algoritmul construieste arborele corespunzator codificarii optime, numit arbore Huffman, pornind de jos in sus. Se incepe cu o multime C de frunze si se realizeaza o secventa de |C|- 1 operatii de fuzionare pentru a crea arborele optim (dupa metoda Greedy).

1. Dat un sir de caractere se calculeaza frecventa aparitiilor fiecarui caracter.
2. Se construieste arborele Huffman optim astfel:
   - Pentru fiecare caracter se considera arborele format numai dintr-un singur nod- radacina, etichetata cu caracterul considerat si cu frec-venta aparitiei lui. Se aranjeaza arborii in ordinea crescatoare a frecventelor (stocate in radacini).
   - La fiecare pas se aleg 2 dintre arborii binari optimi disponibili r_st si r_dr, se conecteaza la o aceeasi radacina (nou creata). Se calculeaza f(r) = f(r_st)+ f(r_dr) se aduga noul arbore in multimea arborilor, si se elimina arborii cu radacinile r_st si r_br din aceasta multime.

Algoritmul lui Huffman *II. Etape*

- Radacina arborelui obtinut prin conectarea a 2 subarbori minimali va contine suma frecventelor frunzelor, subarborilor combinati, si intra calcul pentru urmatoare combinare a arborilor ramasi.
- Daca sunt mai multi arbori cu aceeasi suma a frecventelor minimala (stocata in radacina) atunci se vor alege arborii in mod arbitrar.
- Nodul radacina (si nici oricare alt nod intern) nu este etichetat cu nici un caracter (caracterele sunt stocate in frunze corespondenta 1-1).

Obs. 1. Ca o consecinta arborele minimal poate sa nu fie unic.
Pot exista codificari multiple (minimale) pentru un acelasi text.
2. Nu este important cum se combina subarborii (stang sau drept).
Schimbarea ordinii (stg. dreapta) conduce la codificari (minimale) diferite.

Algoritmul lui Huffman *II. Etape*

Obs. 3. Prin combinarea la fiecare pas a doi arbori, numarul arborilor ramasi se micsoreaza cu o unitatea. In final se va ajunge la un arbore Huffman de pondere minima, in care radacina va fi etichetata cu suma tuturor frecventelor frunzelor , adica 1.

3. Dupa construirea arborelui optimal, fiecare arc stang va fi etichetat cu 0 si fiecare arc drept va fi etichetat cu 1. Citirea etichetelor arcelor de la radacina la frunza va da codificarea caracterului stocat in frunza.

Algoritmul lui Huffman *II. Etape*

Costul arborelui optim obtinut T se calculeaza prin

$$Cost(T) = f(x_1)\cdot d(x_1) + \ldots + f(x_n)\cdot d(x_n),$$

unde x_i sunt caractere distincte ce eticheteaza frunzele arborelui, iar $f(x_i)$ frecventa aparitiei caracterului in textul de codificat.

Alte notiuni: *lungimea medie a codificarii*

$$l(T) = l = p(x_1)\cdot d(x_1) + \ldots + p(x_n)\cdot d(x_n)$$

si *dispersia*

$$D(T) = (l_1 - l)^2 + \ldots + (l_n - l)^2,$$ unde l_i este

lungimea cuvantului-cod (lungimea codificarii lui) x_i .

Exemplu Codificarea Huffman (Statica)

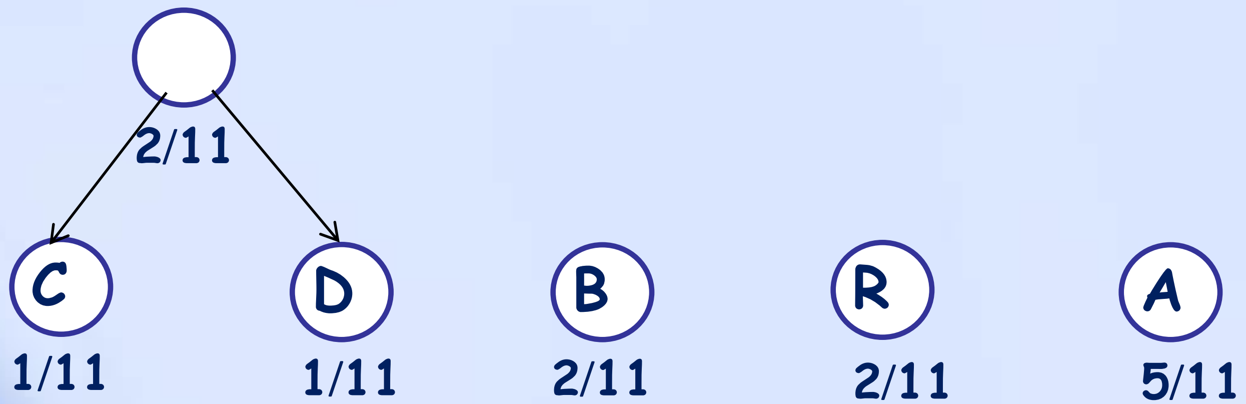
ABRACADABRA

p(A) =5/11, p(B) =2/11, p(C) =1/11, p(D) =1/11, p(R) =2/11,

Exemplu Codificarea Huffman (Statica)

ABRACADABRA

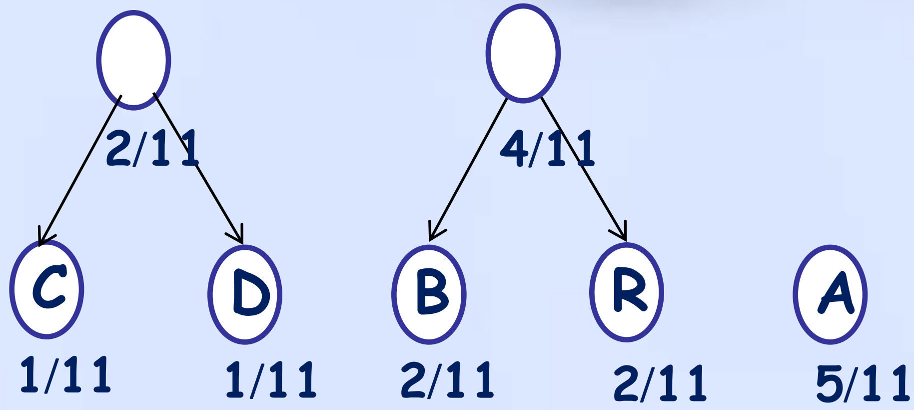
p(A) =5/11, p(B) =2/11, p(C) =1/11, p(D) =1/11, p(R) =2/11,



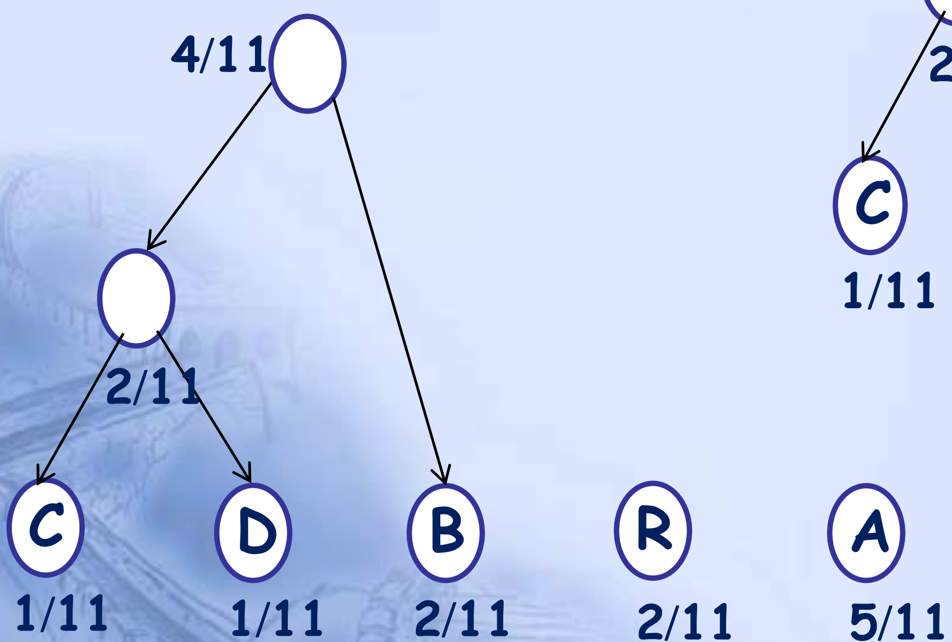
Exemplu Codificarea Huffman   ABRACADABRA

p(A) =5/11, p(B) =2/11, p(C) =1/11, p(D) =1/11, p(R) =2/11,

Arb 2



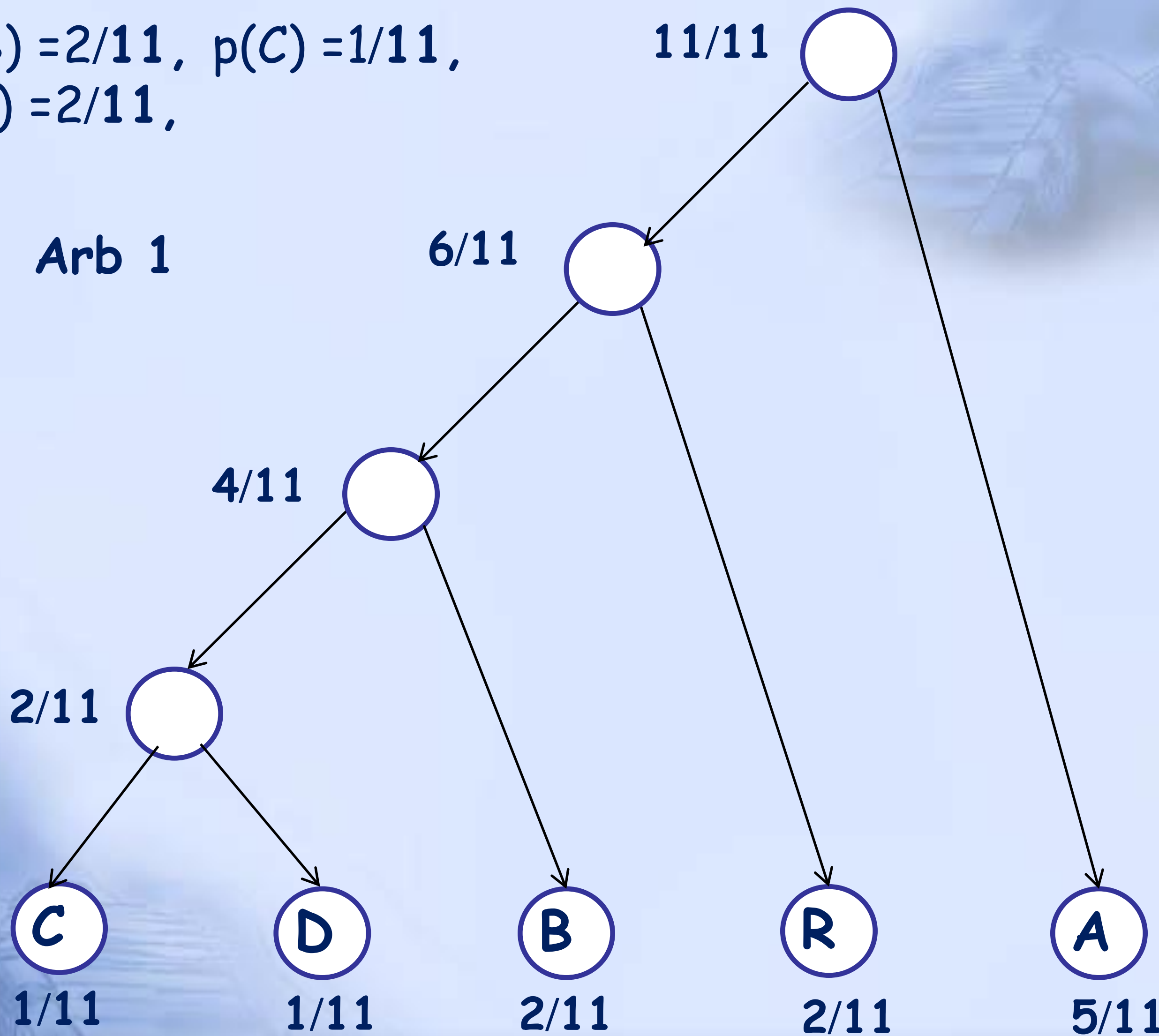
Arb 1



Exemplu Codificarea Huffman ABRACADABRA

p(A) =5/11, p(B) =2/11, p(C) =1/11,
p(D) =1/11, p(R) =2/11,

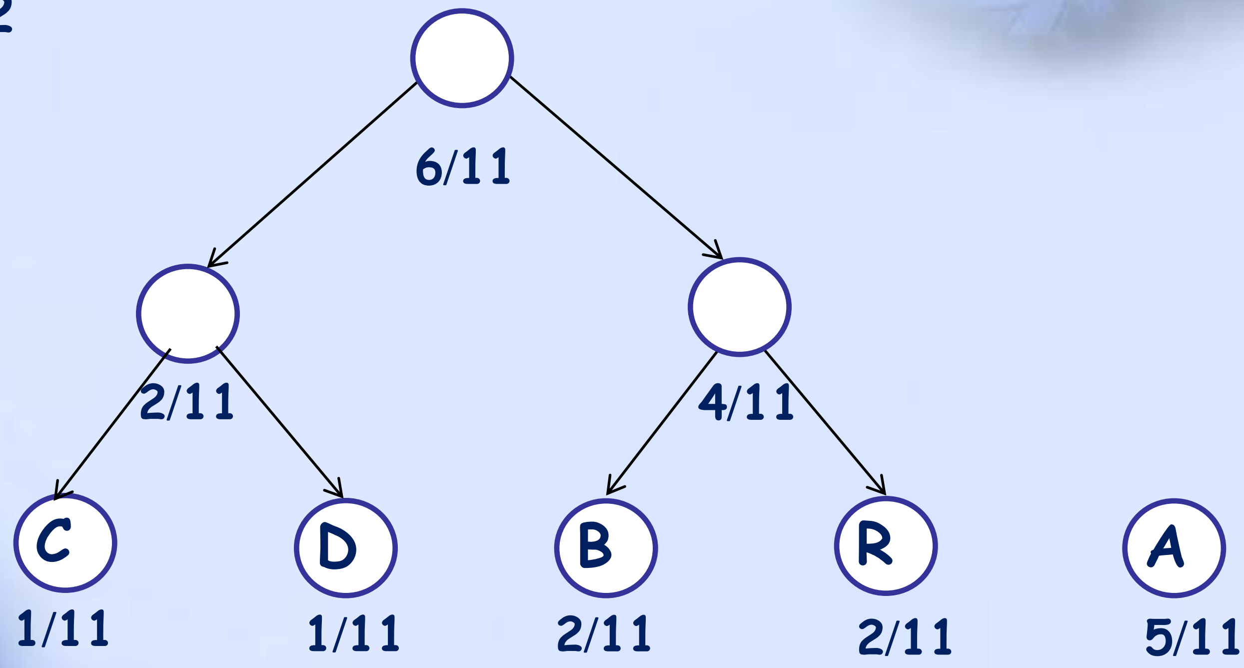
Arb 1



Exemplu Codificarea Huffman ABRACADABRA

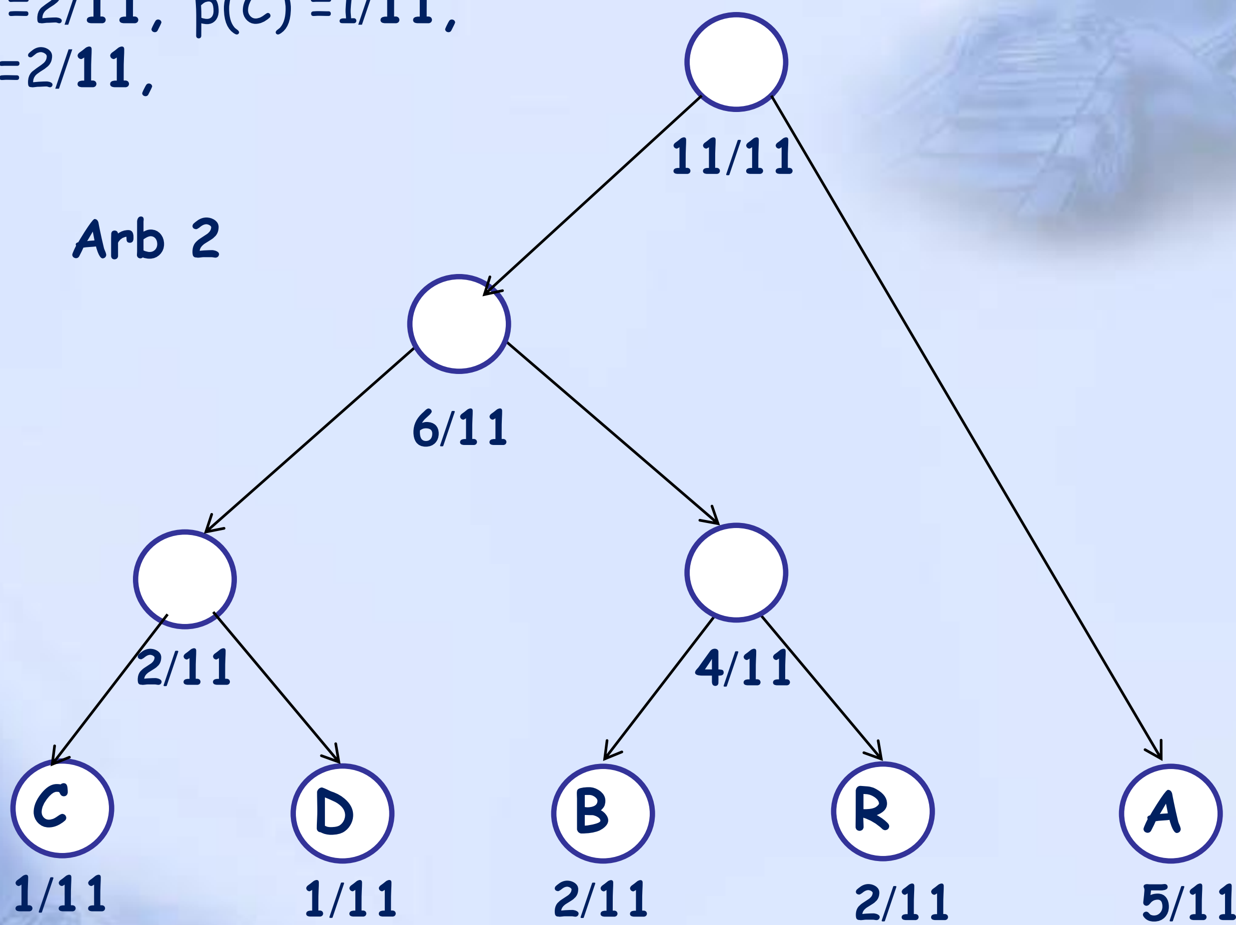
p(A) =5/**11**, p(B) =2/**11**, p(C) =1/**11**, p(D) =1/**11**, p(R) =2/**11**,

Arb 2

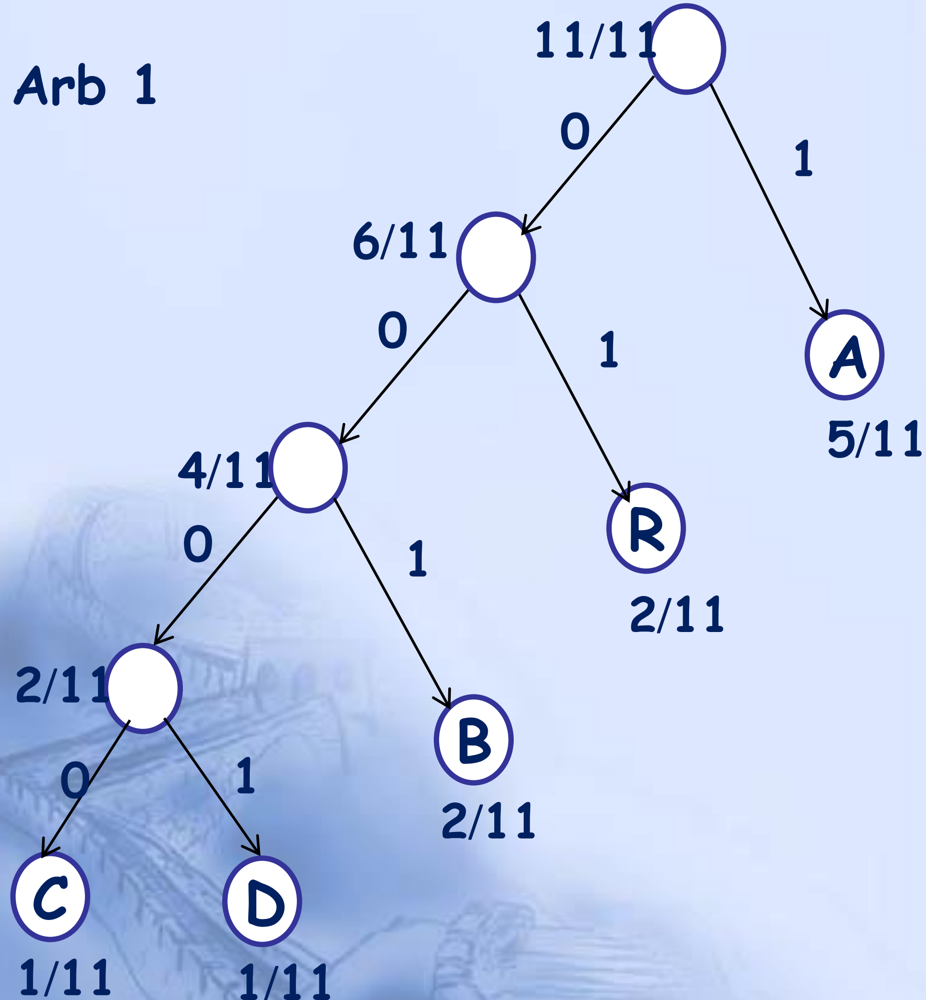


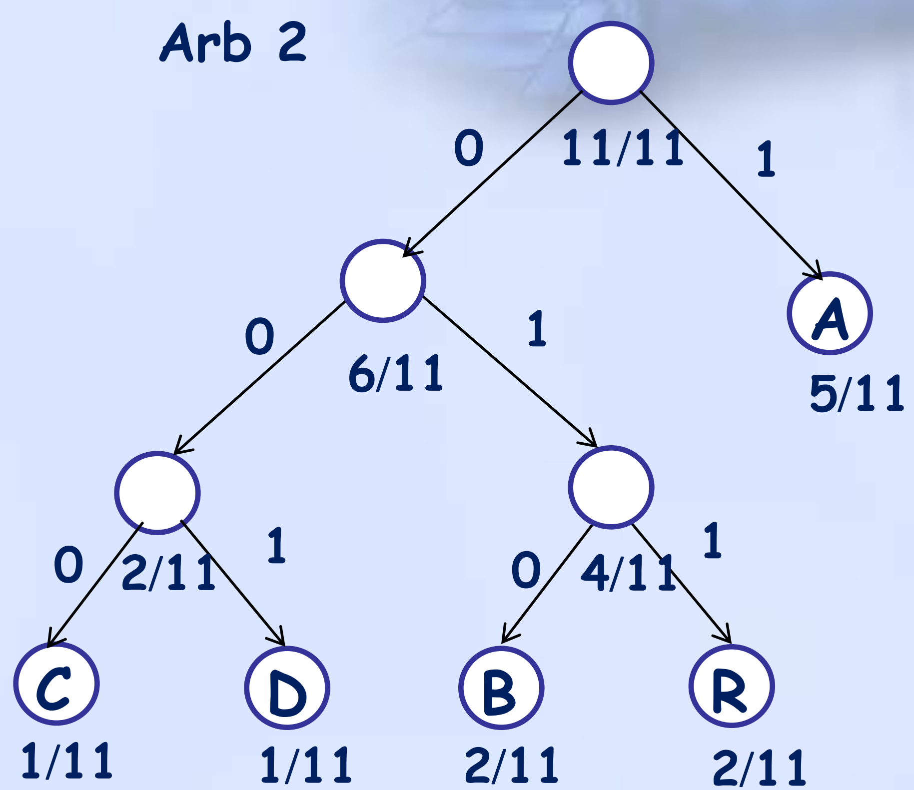
Exemplu Codificarea Huffman ABRACADABRA

p(A) =5/11, p(B) =2/11, p(C) =1/11,
p(D) =1/11, p(R) =2/11,

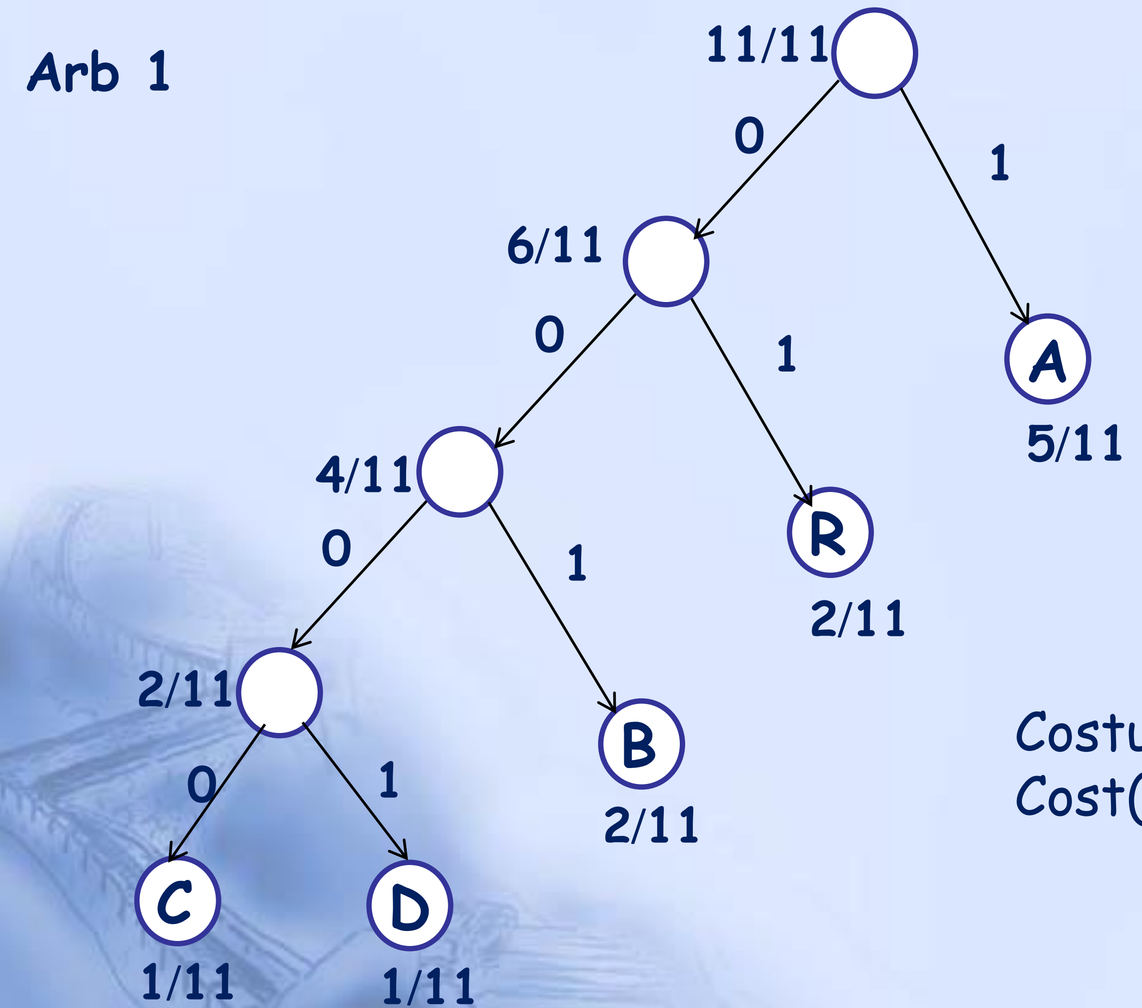


Exemplu Codificarea Huffman ABRACADABRA





Exemplu Codificarea Huffman ABRACADABRA



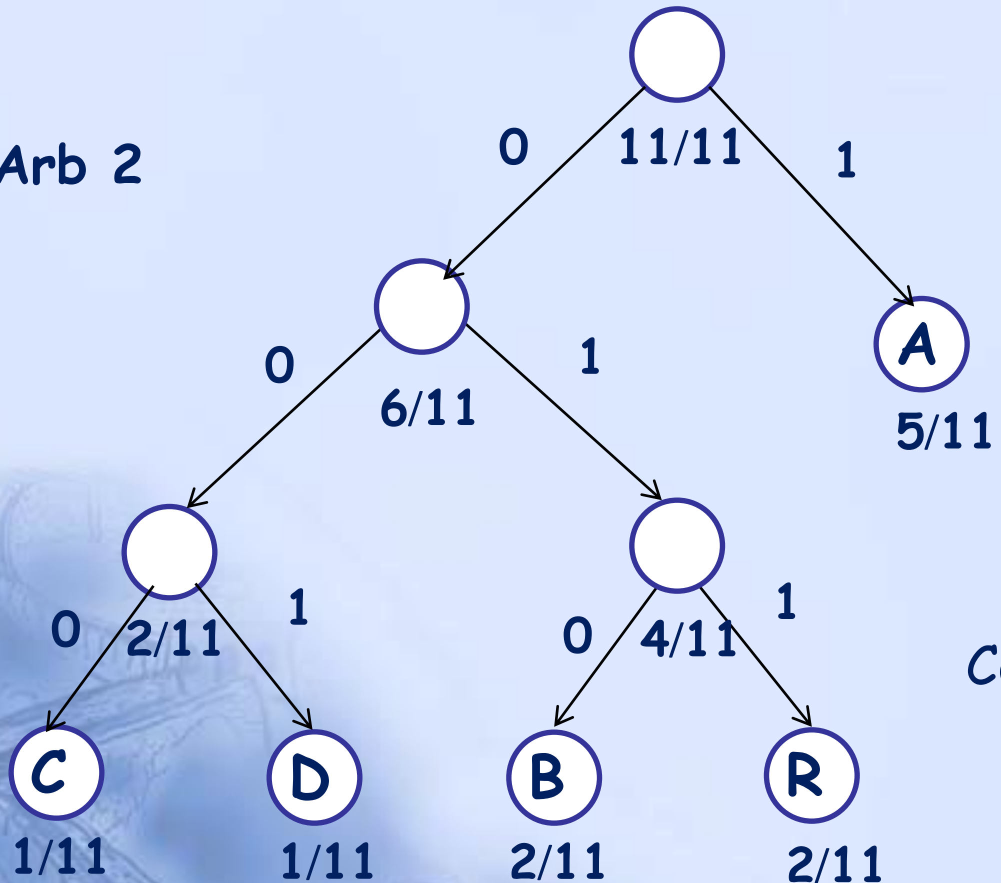
A → 1
B → 001
C → 0000
D → 0001
R → 01

Costul codificarii

$$Cost(Arb1) = 5*1 + 2*2 + 2*3 + 1*4 + 1*4 = 23$$

Exemplu Codificarea Huffman ABRACADABRA

Arb 2



$A \rightarrow 1$

$B \rightarrow 010$

$C \rightarrow 000$

$D \rightarrow 001$

$R \rightarrow 011$

$$Cost(Arb2) = 5*1 + 2*3 + 2*3 + 1*3 + 1*3 = 23$$

$$D(Arb1) > D(Arb2)$$

Codificarea Huffman Statica

Obs. 1. Deoarece la fiecare pas au fost selectati 2 arbori care au frecventele minime, caracterele cu frecvente de aparitie mai mari au fost conectate la arborele optim cat mai tarziu (cat mai aprope de radacina) lucru care atrage optimalitatea arborelui. Datorita acestei proprietati, codificarea Huffman realizeaza si o compresie a datelor.

Obs. 2. Codurile Huffman au ***proprietatea de prefix-free***

"Nici un cuvant-cod nu este prefix pentru un alt cuvant-cod"

Rezulta din constructia arborelui, drumurile in arbore se suprapun ... pana la frunze (frunza isi spune ultimul cuvant – "delimitator in decodificare").

Codificarea Huffman Statica

Obs. 3. Datorita proprietatii de prefix decodificare se face foarte usor (codificarea este neambigua, unica) proprietate care caracterizeaza un cod valid.

Obs. 4. ***Codificarea Huffman este de lungime variabila*** (caracterele sunt inlocuite de secvente de biti (cuvinte-cod) de lungimi diferite.
O codificare de lungime fixa necesita cel putin 3 biti $\Rightarrow$ o codificare Huffman este mult mai economica. Ex. Cost(Arb) = 23, L(3) = 33.

Algoritmul de Codificare Huffman - pseudocod

```
Huffman(A)
{ n:= |A|;
for  i = 1 to n
  {calculeaza  f [xi] (frecventa
    aparitiei caracterul xi in text)
   }
 PQ: = A; (inserare in PQ in ordinea
crescatoare a frecventelor din radacini)
 while PQ not empty do
   { r:= new node;
     left[r] =Extract-Min(PQ);
     right[r] =Extract-Min(PQ);
     f[r] = f[left[r]] +f[right[r]];
     Insert(PQ; r); (inserare lui r in PQ
                          cu prioritatea f(r))
    }
  return Extract-MinP(Q)
 }
```

Extragerea si inserarea in PQ necesita O(log n) timp, si se efectueaza de O(n) ori. Asadar, complexitatea algoritmului este O(n log n).

Optimalitatea Arborelui Huffman

Lemma 1 Fie x si y doua caractere cu o frecventa minima. Atunci exista un arbore optim in care aceste doua caractere sa fie frati situati pe cel mai de jos nivel al arborelui.

Dem. Fie T un arbore optim, si b si c doua frunze (frati) situate pe cel mai de jos nivel al lui T. Pp. $f(x) \leq f(y)$ si $f(b) \leq f(c)$ $\Rightarrow$

$$f(x) \leq f(b),\ f(y) \leq f(c),$$
$$d(b) \geq d(x),\ d(c) \geq f(y).$$

Schimbam pozitia lui x cu b in T si vedem daca arborele ramane optim.

$$Cost(T) \leq Cost(T') = Cost(T) - d(b)f(b) + d(b)f(x) - d(x)f(x) + d(x)f(b) =$$
$$= Cost(T) - (d(b) - d(x))(f(b) - f(x)) \leq Cost(T)$$

$Cost(T) = Cost(T') \Rightarrow$ prin interschimbare arborele ramane optim.

Analog daca se schimba y cu c (in T') se obtine un arbore T'' astfel incat $Cost(T') = Cost(T'')$

Optimalitatea Arborelui Huffman

Consecinta Fie T un arbore de codificare optim pentru un alfabet A. Daca $f(b) < f(x)$ atunci $d(b) \geq d(x)$.

Lemma2 Arborele optim asociat oricarei codificari prefix trebuie sa fie complet. (Orice nod intern are exact doi fii).

Dem. Se inlocuieste nodul cu singurul sau fiu, fara a schimba costul arborelui.

Optimalitatea Arborelui Huffman

Teorema Algoritmul lui Huffman este optim.

Dem. Prin inductie dupa n. Pentru $n = 2$ trivial.
Presupunem ca pentru un alfabet A format din n-1 litere Algoritmul lui Huffman construieste un arbore optim. Demonstram ca proprietatea se pastreaza pentru n litere.

Fie A' un alfabet format din n litere, cu proprietatea ca x si y sunt doua litere cu frecvente minimale. Conf. Lemmei 1 exista un arbore T' optim care sa aiba cele doua caractere ca frati situati pe cel mai de jos nivel.

Fie T arborele obtinut din T ' prin eliminarea celor doua frunze x si y, dar cu pastrarea parintelui, etichetat cu o noua litera z, a carui frecventa este $f(z) = f(x) + f(y)$ si $d(z) = d(x) - 1 = d(y) - 1$.
Astfel T va fi un arbore Huffman pentru un alfabet $A = (A' \cup \{z\}) - \{x,y\}$ format din n-1 litere, caruia i se poate aplica ipoteza inductiva.

Optimalitatea Arborelui Huffman

Teorema Algoritmul lui Huffman este optim.

Dem. Asadar

$Cost(T) = Cost(T') - d(x)f(x) - d(y)f(y) + d(z)f(z) =$
$= Cost(T') - d(z)\, f(x) - d(z)\, f(y) - f(x) - f(y) + d(z)(f(x) + f(y)) =$
$= Cost(T') - (f(x) + f(y))$ (1)

Conform ipotezei inductive, algoritmul Huffman construieste un arbore de codificare optim pentru A, |A| = n-1 litere.

Fie T_A acest arbore optim.
Adaugam la T_A, ca fii ai lui z frunzele x si y. Fie T '_A noul arbore.

Din (1) rezulta $Cost(T\_A) = Cost(T'\_A) - (f(x) + f(y))$ deci
$Cost(T'\_A) = Cost(T\_A) + (f(x) + f(y)) \leq Cost(T) + f(x) + f(y) = Cost(T')$

Cum T ' era arbore optim, iar T '_A s-a construit prin refacerea arborelui Huffman $\Rightarrow$ T '_A este optim.

Huffman Decoding

- Pentru a putea decodifica mesajul receptorul trebuie sa primeasca eventual probabilitatile frecventelor caracterelor, si fie codificarea caracterelor (cuvintele cod) sau arborele de codificare.

- In ambele cazuri se folosest proprietatea de prefix-free (codificarea este unica).

- Daca se primeste arborele Huffman, atunci se citeste arborele de la radacina spre frunze , ghidat de bitii din text (se merge spre stanga in arbore pt. orice bit 0 in text, si la dreapta pt. orice bit 1) pana la identificarea unei frunze . Se returneaza caracterul gasit in frunza.

- Se repeta procedeul pana la sfarsit.

Huffman Coding - Aplicatie

Se considera un text peste {a, b, c, d, e, f, g}, in care fiecare litera apare cu probabilitatea $p(a) = 0.1$, $p(b) = 0.1$, $p(c) = 0.05$, $p(d) = 0.25$, $p(e) = 0.20$, $p(f) = 0.15$, $p(g) = 0.15$.

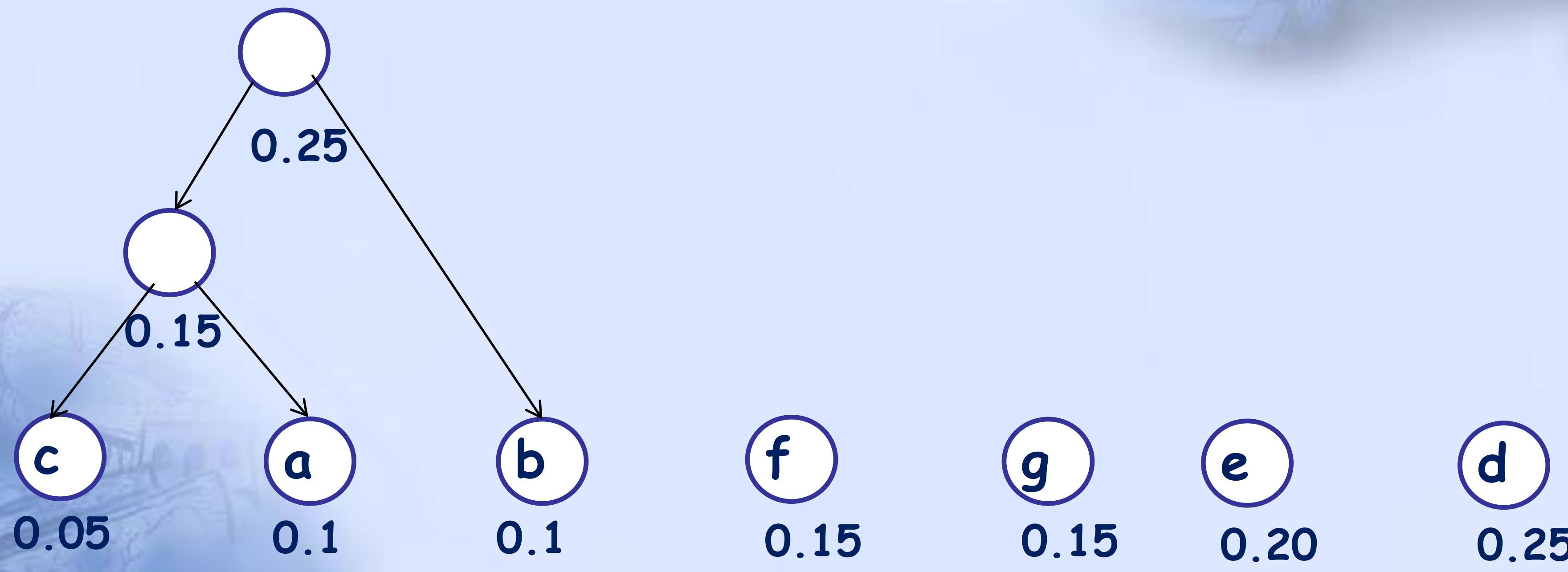
Se considera codificarea $h(a)=101$, $h(b)=110$, $h(c)=111$,
$h(d)=00$, $h(e)=101$, $h(f)=011$, $h(g)=100$.
Este h o codificare Huffman?

Rezolvare

1. Se construieste un Arbore Huffman pentru frecventele date.
2. Se calculeaza costul Arborelui.
3. Se verifica daca pentru codificarea h este satisfacuta proprietatea de optimalitatea (are acelasi cost ca si arborele construit) si are proprietatea de prefix.

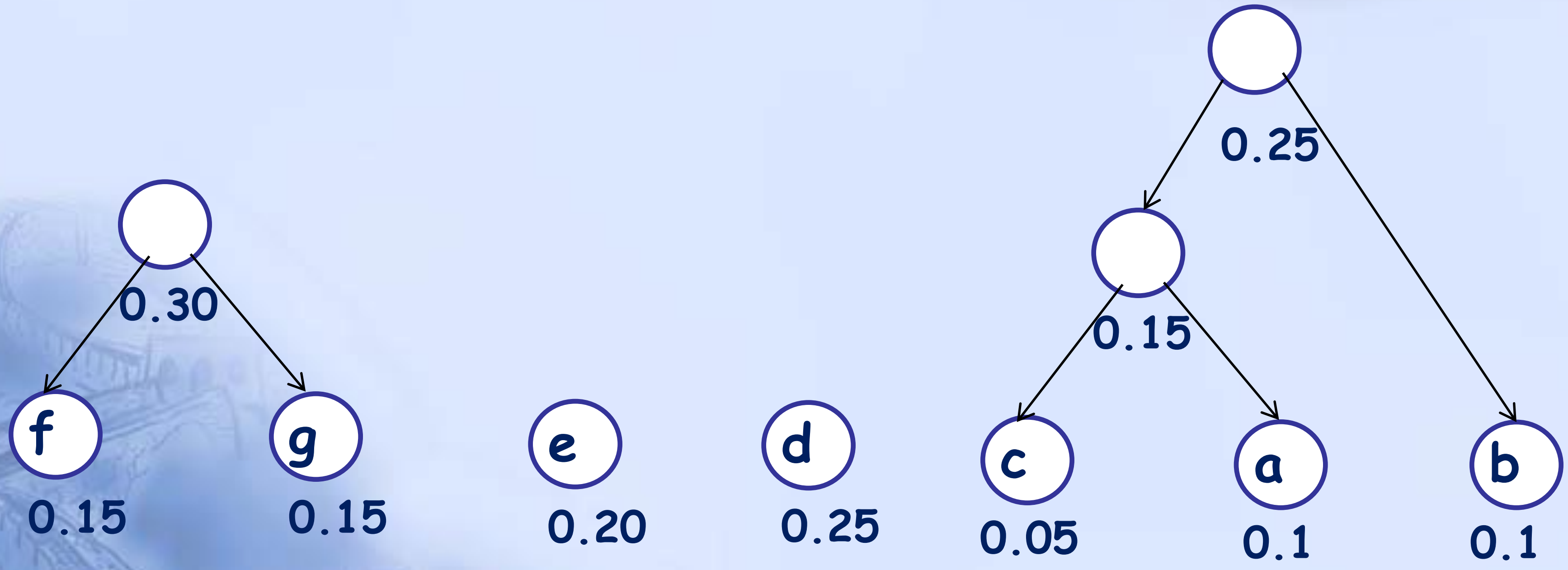
Huffman Decoding - Aplicatie

Rezolvare probabilitati $p(a) = 0.1$, $p(b) = 0.1$, $p(c) = 0.05$, $p(d) = 0.25$, $p(e) = 0.20$, $p(f) = 0.15$, $p(g) = 0.15$.



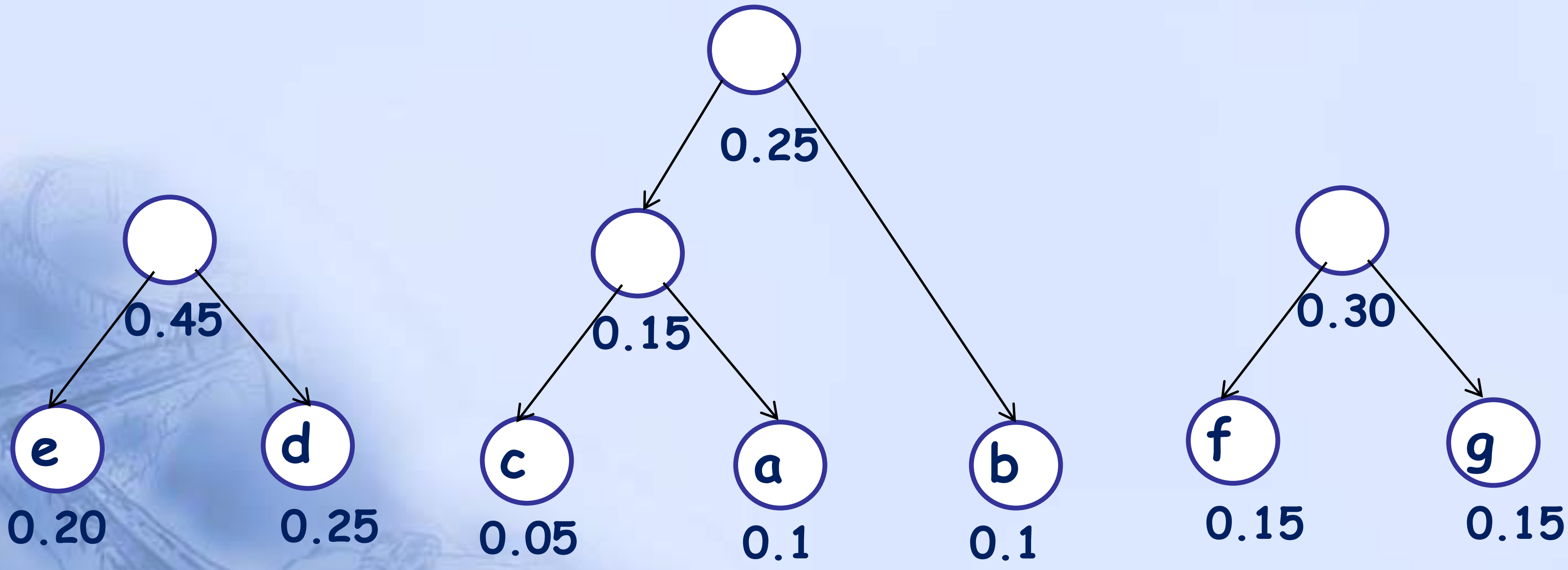
Huffman Decoding - Aplicatie

Rezolvare probabilitati $p(a) = 0.1$, $p(b) = 0.1$, $p(c) = 0.05$, $p(d) = 0.25$, $p(e) = 0.20$, $p(f) = 0.15$, $p(g) = 0.15$.



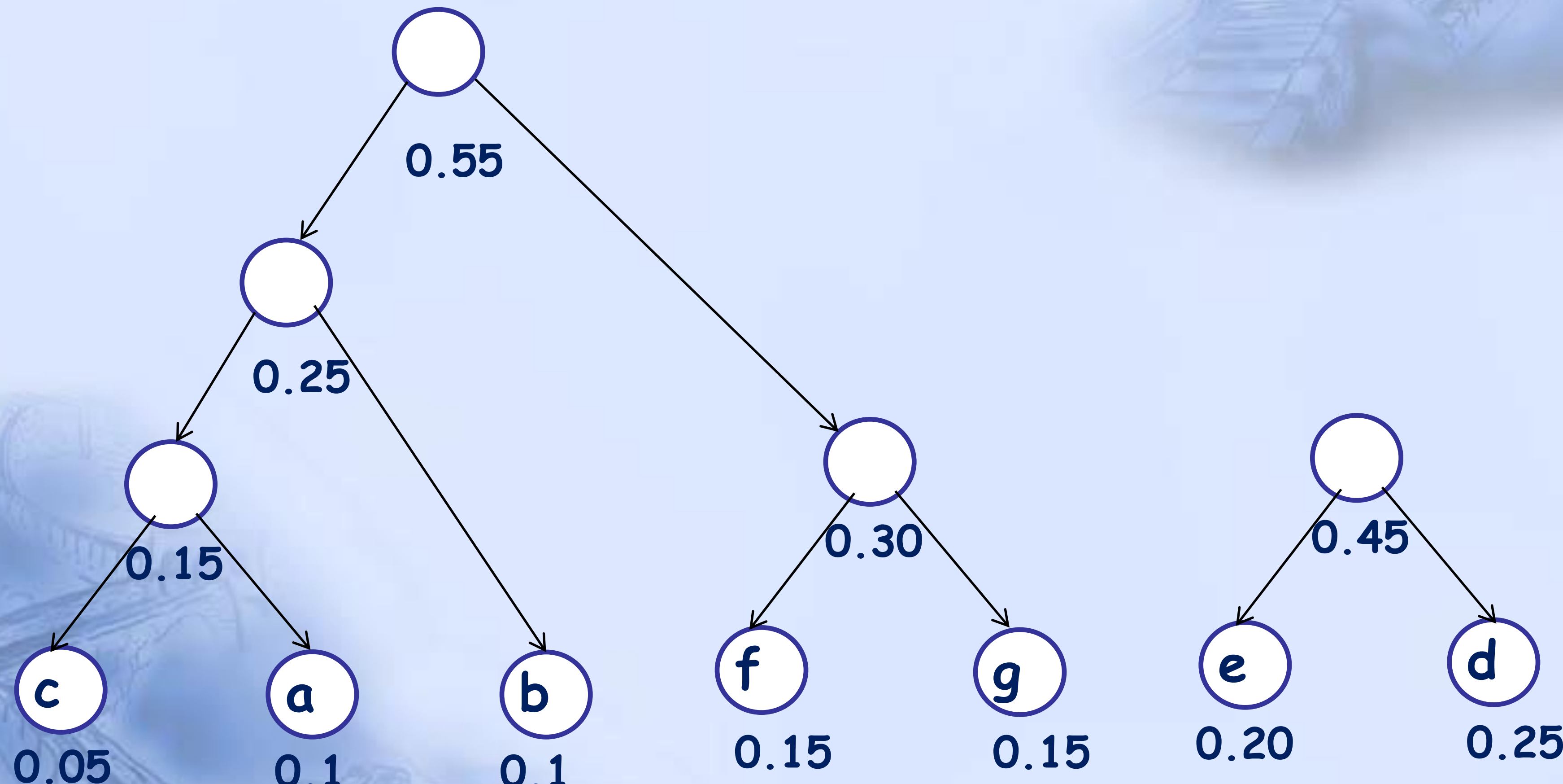
Huffman Decoding - Aplicatie

Rezolvare



Huffman Decoding - Aplicatie

Rezolvare



Huffman Decoding - Aplicatie

Rezolvare

Cost(T)=40+30+20+50+
40+45+45=270

h nu e codificare Huffman

h(a)=101,
h(b)=110,
h(c)=111,
h(d)=00,
h(e)=101,
h(f)=011,
h(g)=100
Cost(h)= 275

a → 1001
b → 101
c → 1000
d → 01
e → 00
f → 110
g → 111

Codificare si Compresie - dezavantaje

1. Algoritmul lui Huffman nu poate fi aplicat pentru un alfabet cu 2 numar 2 litere. Algoritmul realizeaza de fapt o rescriere $A \rightarrow 1$, $B \rightarrow 0$ (sau invers) … nici compresie nici codificare.

2. Algoritmul devine eficient (in ceea ce priveste compresia) in special cand probabilitatile sunt diferite. Probabilitati egale implica lungimi egale ale cuvintelor cod $\rightarrow$ nu conduce la o compresie a textului.

3. Care dintre arborii Huffman este cel mai bun. Costul este acelasi dar in practica convenabila este codficarea cu dispersia mai mica (inaltimea arborelui Huffman mai mica). Ex: Situatia in care mesajul codificat este trimis unui receptor (conduce la supra-incarcarea bufferului). Ex: Arb2 este mai convenabil.

Codificare si Compresie - dezavantaje

4. Cati arbori pot fi generati ?
Un arbore Huffman cu n caractere are cel mult n-1 noduri interioare. Arcele fiind etichetate cu 0 sau 1. Prin inversarea fratilor intre ei (nod stang cu nod drept) conduce la alta codificare (alt arbore) . Deci vor exista cel mult 2^(n–1) arbori.

5. Stacarea unui fisier arhivat pe un disc, necesita o lungime multiplu de 8. Prin stocare, un mesaj care nu are codificarea un multiplu de 8 biti va primi in mod aleatoriu un numar de biti 0 sau 1 pana la un multiplu de 8. Solutia: *pseudo-end-of-file character - pseudo-EOF*, la sfarsitul mesajului de codificat se insereaza un marcar (care prin codificare acopera minusul de biti.)

Codificarea Huffman Dinamica

- Codificarea adaptiva Huffman a fost conceputa in mod independent de Faller (1973) si Gallager (1978).

- In 1985 D. Knuth adus o imbunatatire algoritmului, realizand ceea ce se numeste algoritmul FGK (Faller-Gallager-Knuth)

Codificarea Huffman Dinamica

Codificarea Huffman Dinamica realizeaza codificarea unui text in timpul "transmisiei " , asadar fara a se cunoaste de la inceput dispersia elementelor in textul final de transmis.

Construirea arborelui Huffman, in acest caz, se face din mers, acesta "adaptandu-si " stuctura la fiecare pas al transmisiei (citirii textului) in functie de frecventa elementelor la momentul dat.

Ideea de baza este construirea unui arbore Huffman optim la fiecare citire a unui simbol (pt. mesajul citit pana la momentul transmisiei).

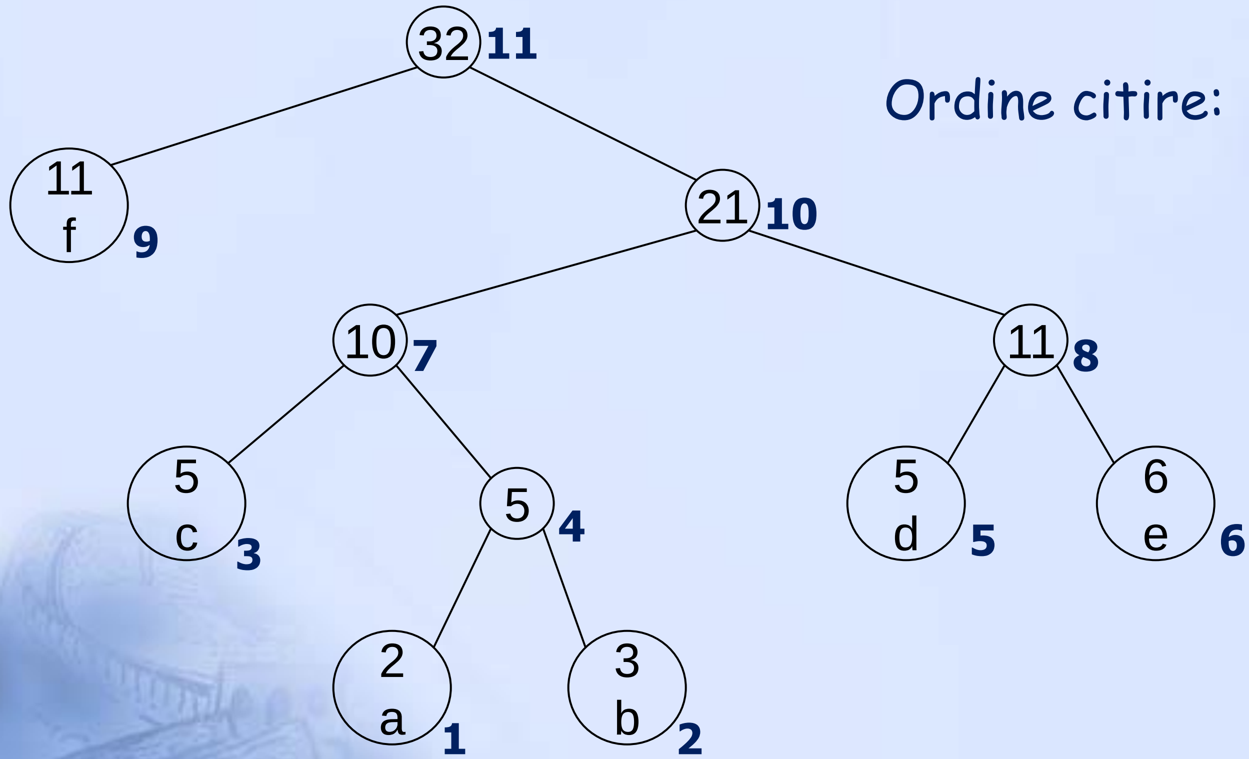
Codificarea Huffman Dinamica

- Construirea arborelui se bazeaza pe proprietatea de "fratie" (sibling). Frecventa fratelui stang trebuie sa fie mai mica (sau egala) decat frecventa fratelui drept, aceasta implica o ordonare a frecventelor nodurilor, in ordine crescatoare, atunci cand acestea sunt citite pe nivele, incepand de la cel mai de jos nivel, in ordine stanga-dreapta.

- Nodul parinte sumeaza frecventele fiilor sai deci va avea intotdeauna o frecventa mai mare.

- Daca aceasta ordine nu este indeplinita prin citirea unui nou caracter din text, atunci se interschimba subarborii cu radacinile unde s-au inregistrat dezacordurile.

Codificarea Huffman Dinamica



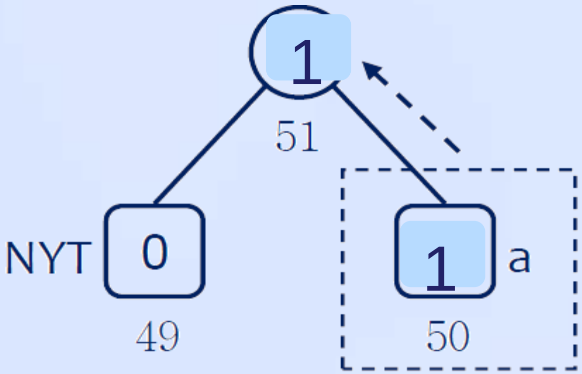
Ordine citire: 1,2,3,4, . . . , 9, 10, 11

Ordine crescatoare a frecventelor: 2,3,5,5, 5, 6, 10, 11, 11,21,32

Codificarea Huffman Dinamica Exemplu

Input: aardvark

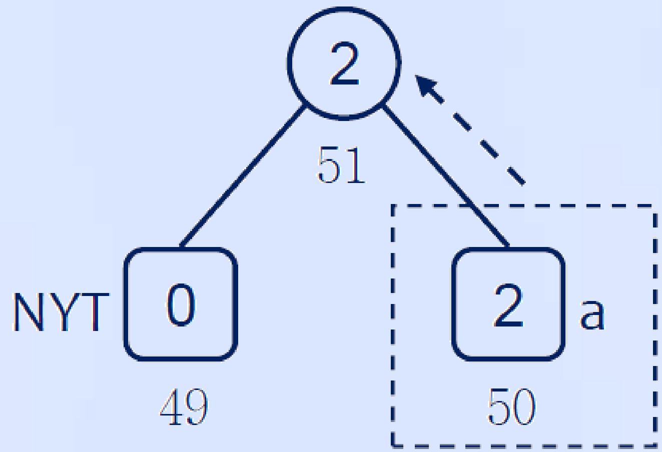
(Algoritmul FGK)



Ordinea frecventelor: 0,1,1

Simbol	Cod
NYT	0
a	1
r	
d	
v	
k	

aardvark

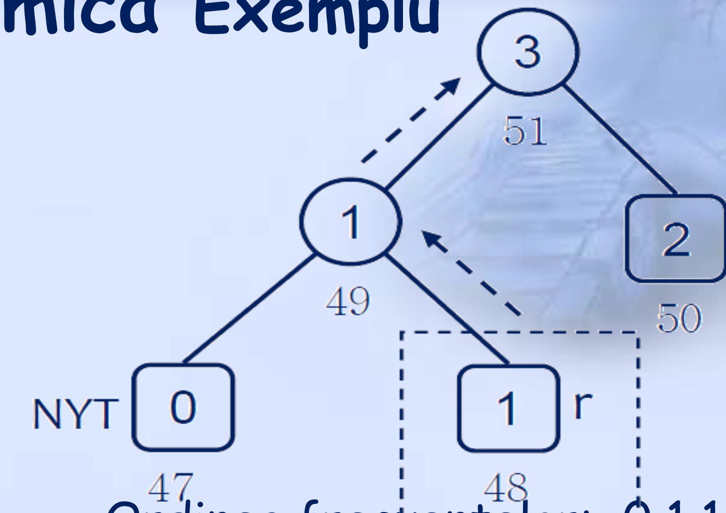


Ordinea frecventelor: 0,2,2

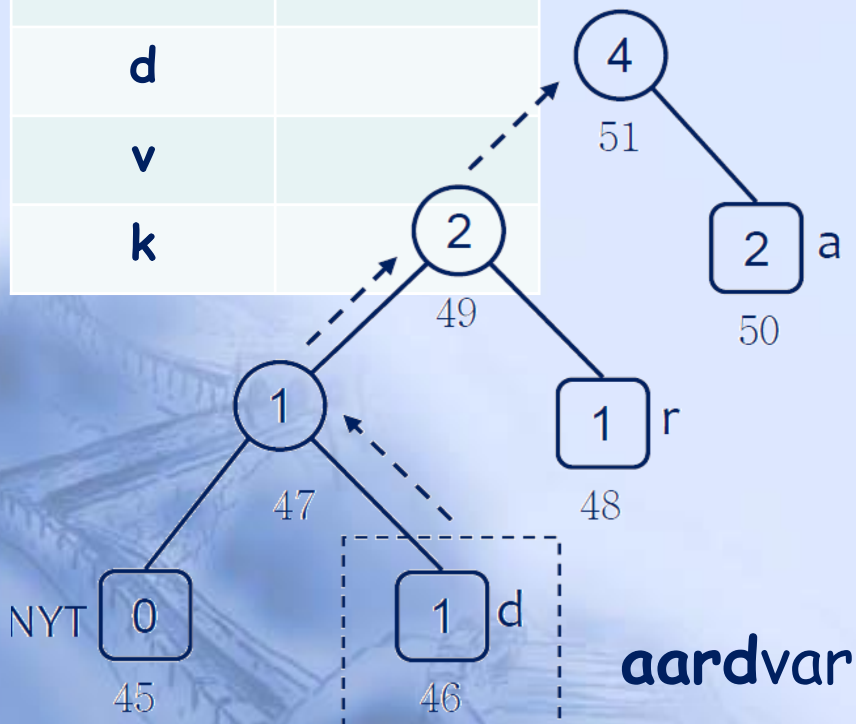
Codificarea Huffman Dinamica Exemplu

Simbol	Cod
NYT	00
a	1
r	01
d	
v	
k	

aardvark



Ordinea frecventelor: 0,1,1,2,3



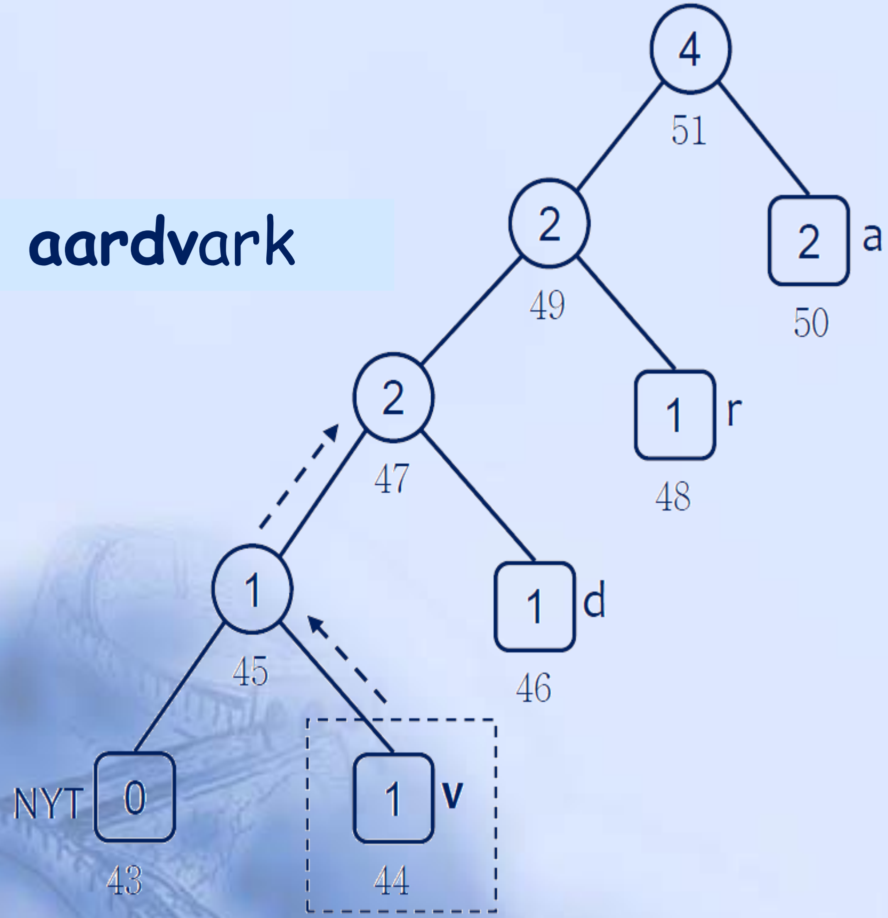
aardvark

Ordinea frecventelor: 0,1,1,1,2,2,4

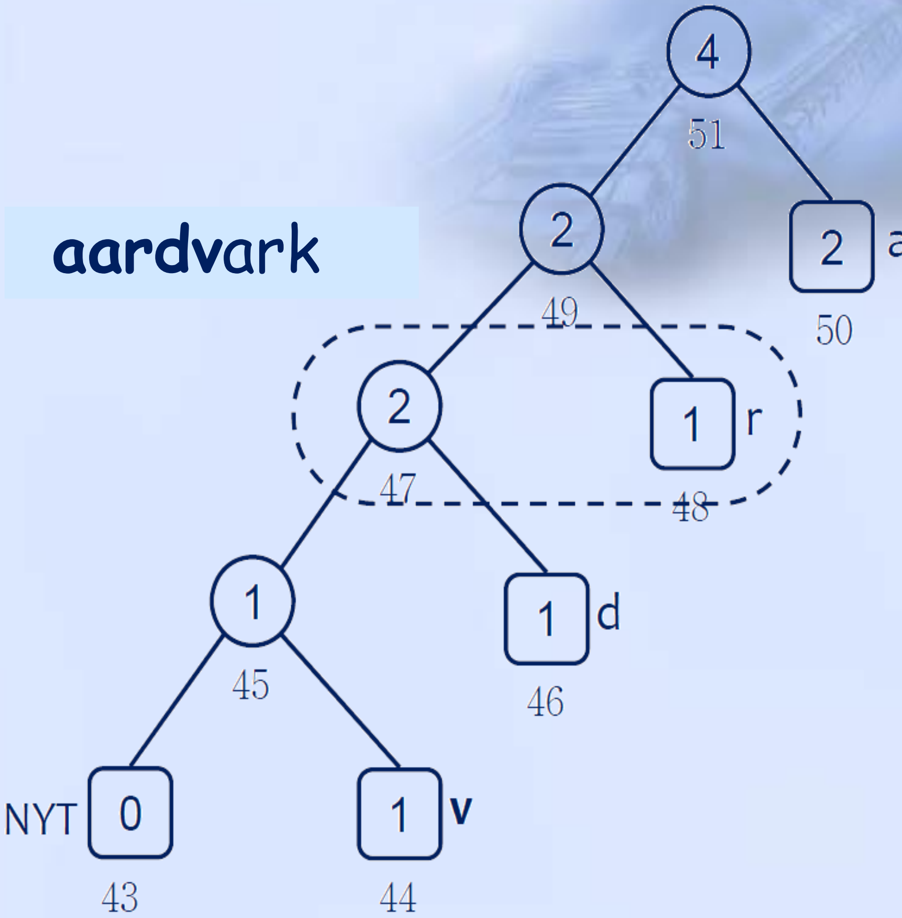
Simbol	Cod
NYT	000
a	1
r	01
d	001
v	
k	

Codificarea Huffman Dinamica Exemplu

aardvark



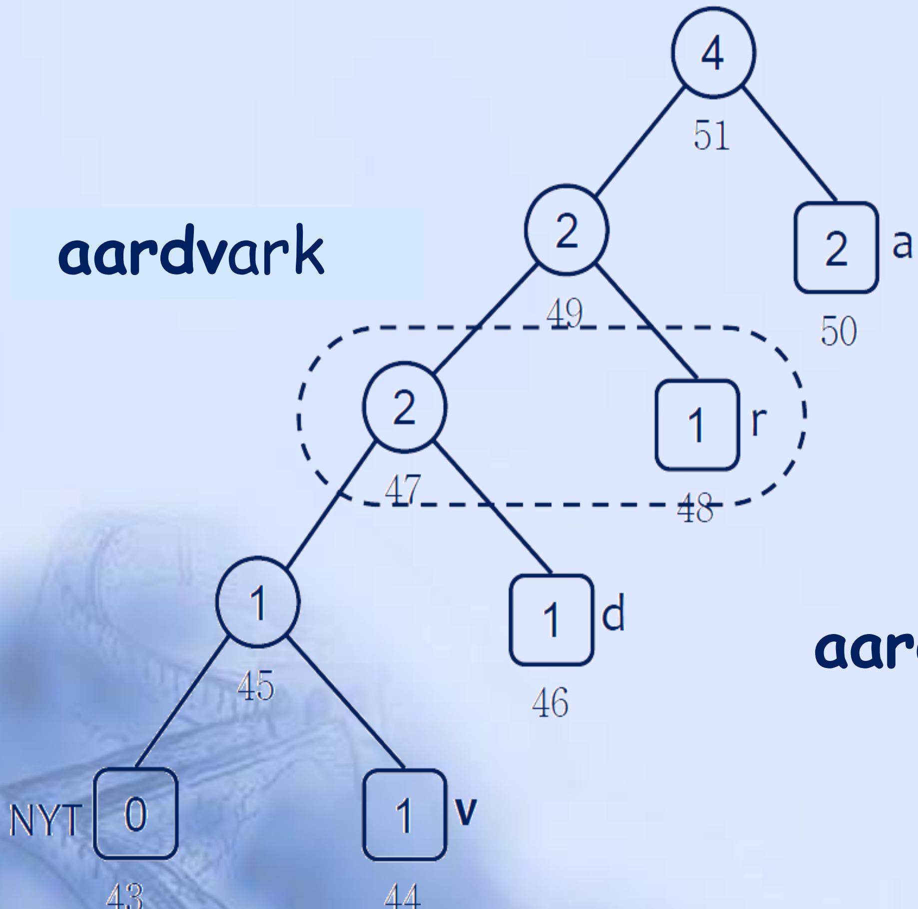
aardvark



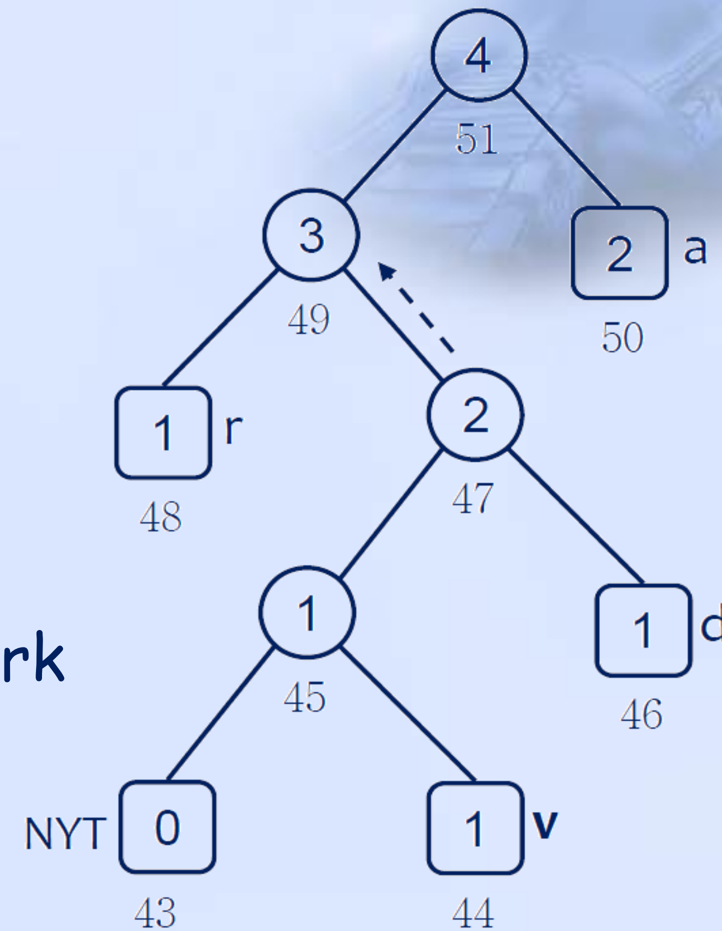
Ordinea frecventelor: 0,1,1,1,2,1,

Codificarea Huffman Dinamica Exemplu

aardvark

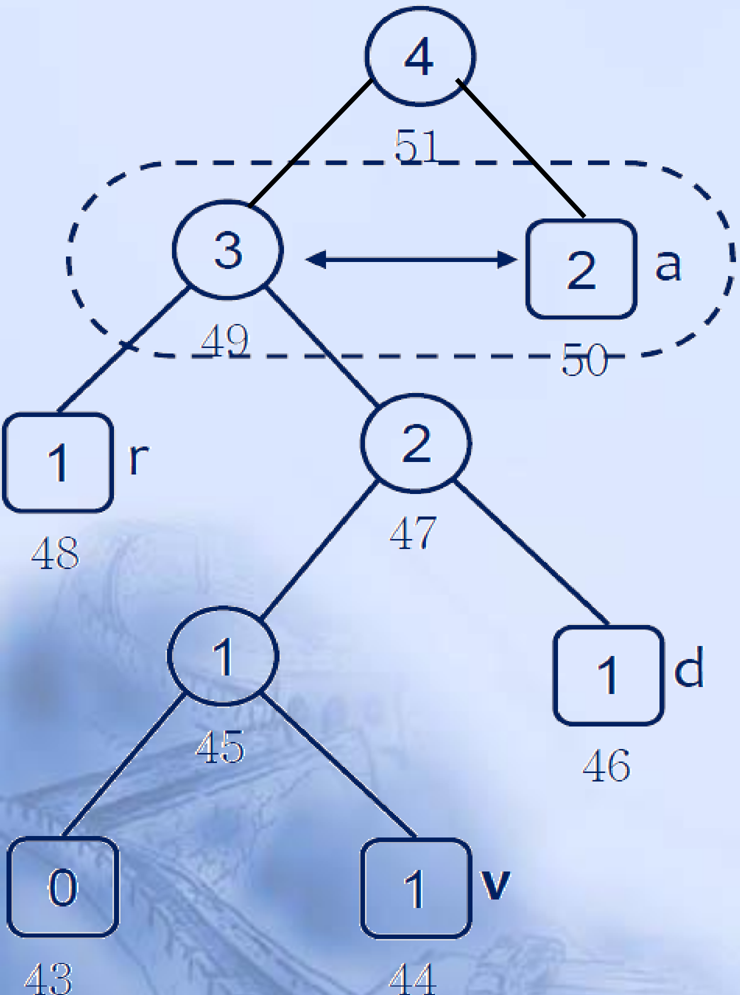


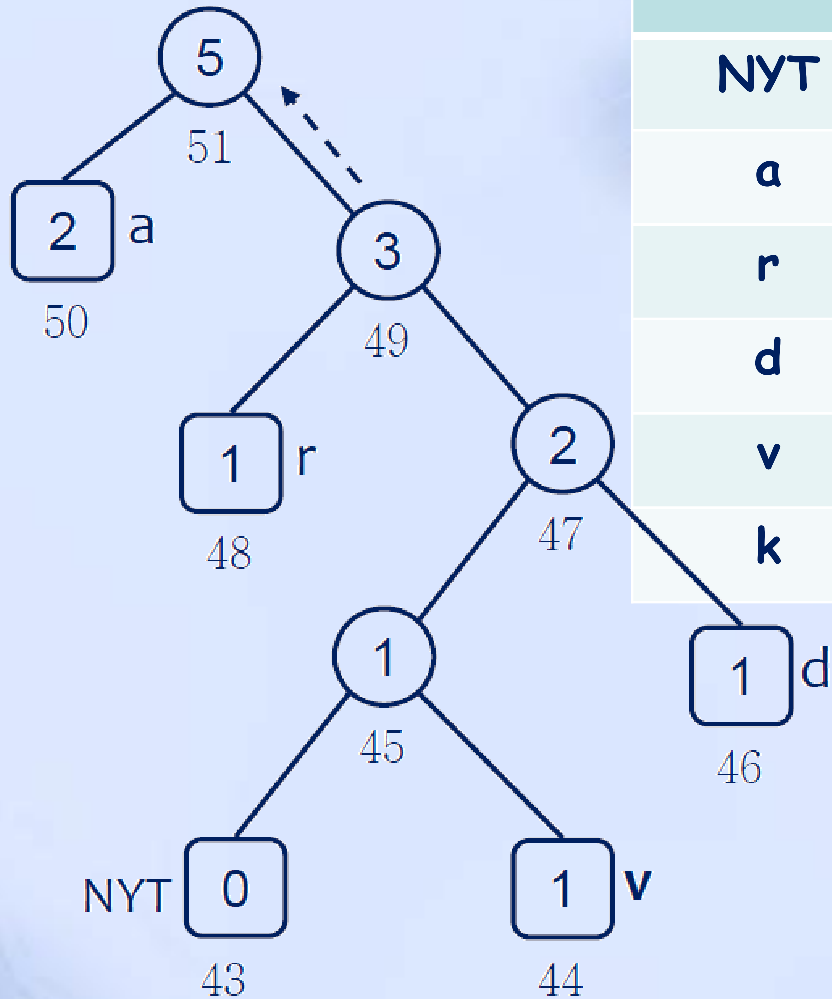
aardvark



Noua ordine a frecventelor: 0,1,1,1,2,3

Codificarea Huffman Dinamica Exemplu



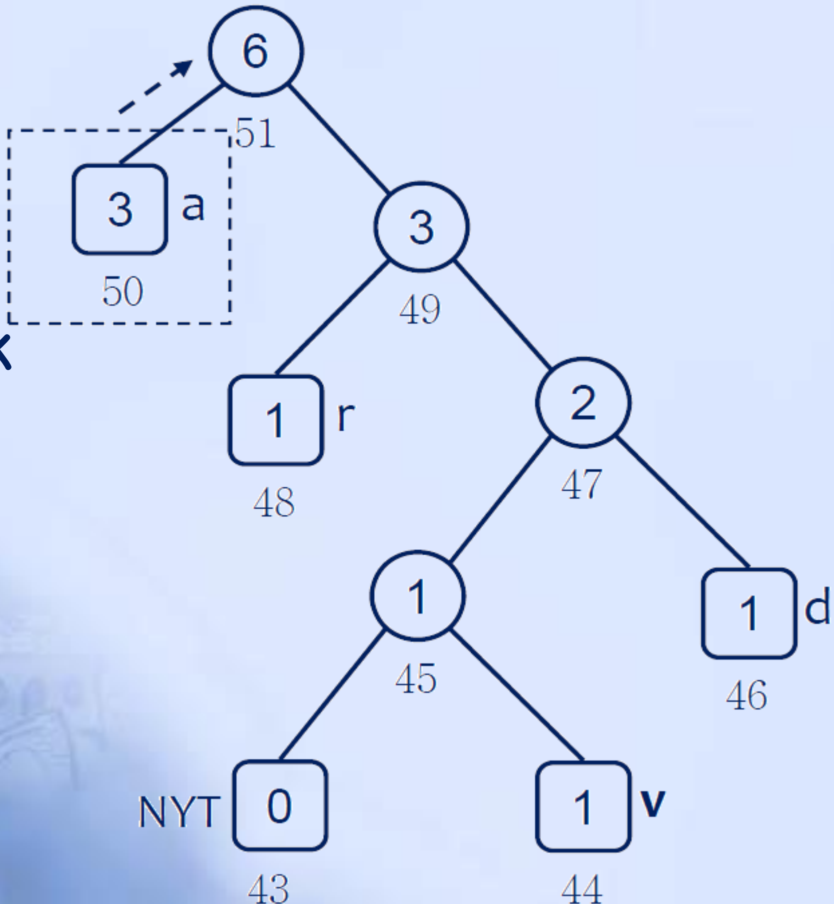


Simbol	Cod
NYT	1100
a	0
r	10
d	111
v	1101
k	

Noua ordine a frecventelor: 0,1,1,1,1, 2,2,3,5

Codificarea Huffman Dinamica Exemplu

aardvark

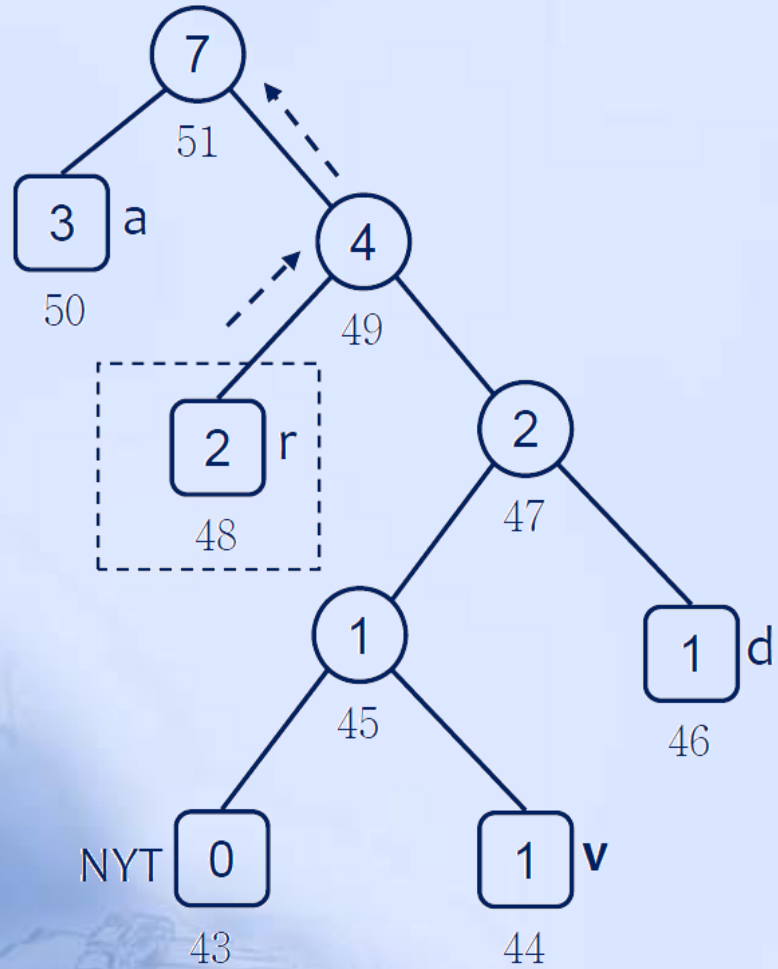


Simbol	Cod
NYT	1100
a	0
r	10
d	111
v	1101
k	

Ordinea frecventelor: 0,1,1,1,1, 2,3,3,6

Codificarea Huffman Dinamica Exemplu

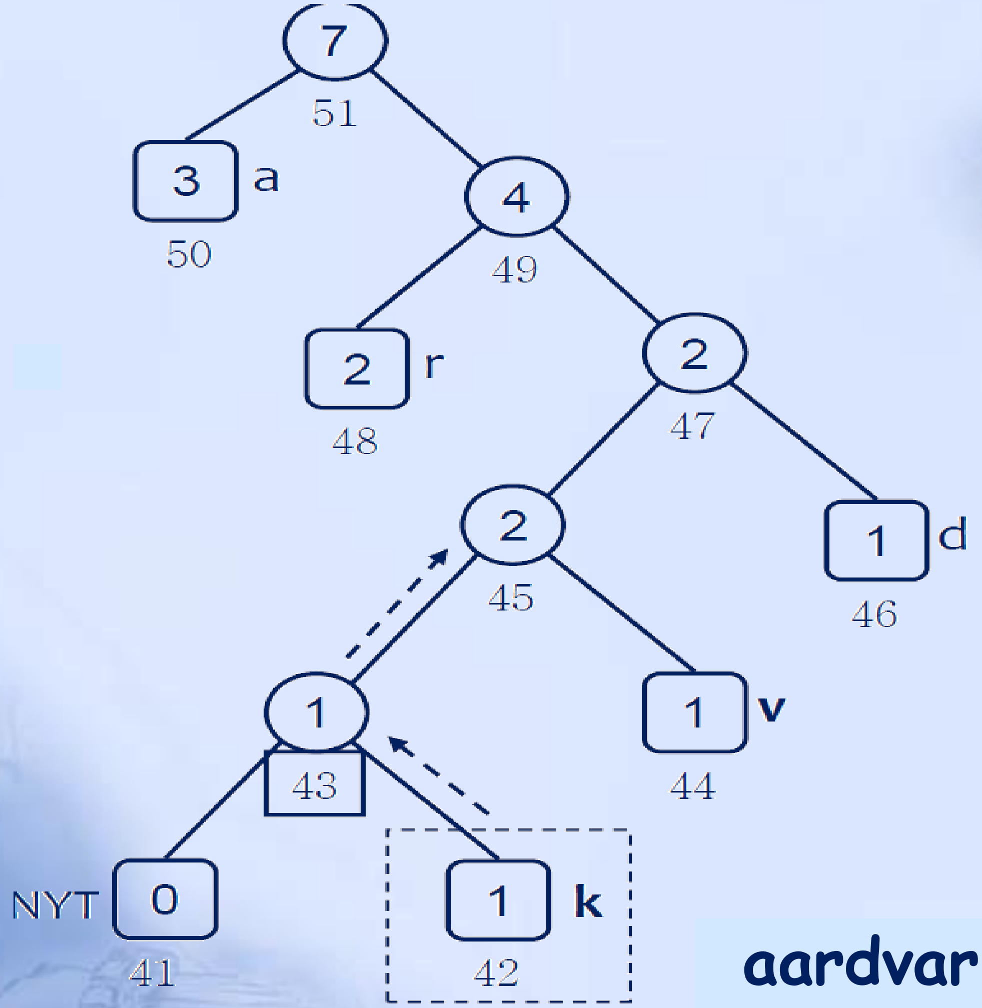
aardvark



Simbol	Cod
NYT	1100
a	0
r	10
d	111
v	1101
k	

Ordinea frecventelor: 0,1,1,1,2, 2,3,4,7

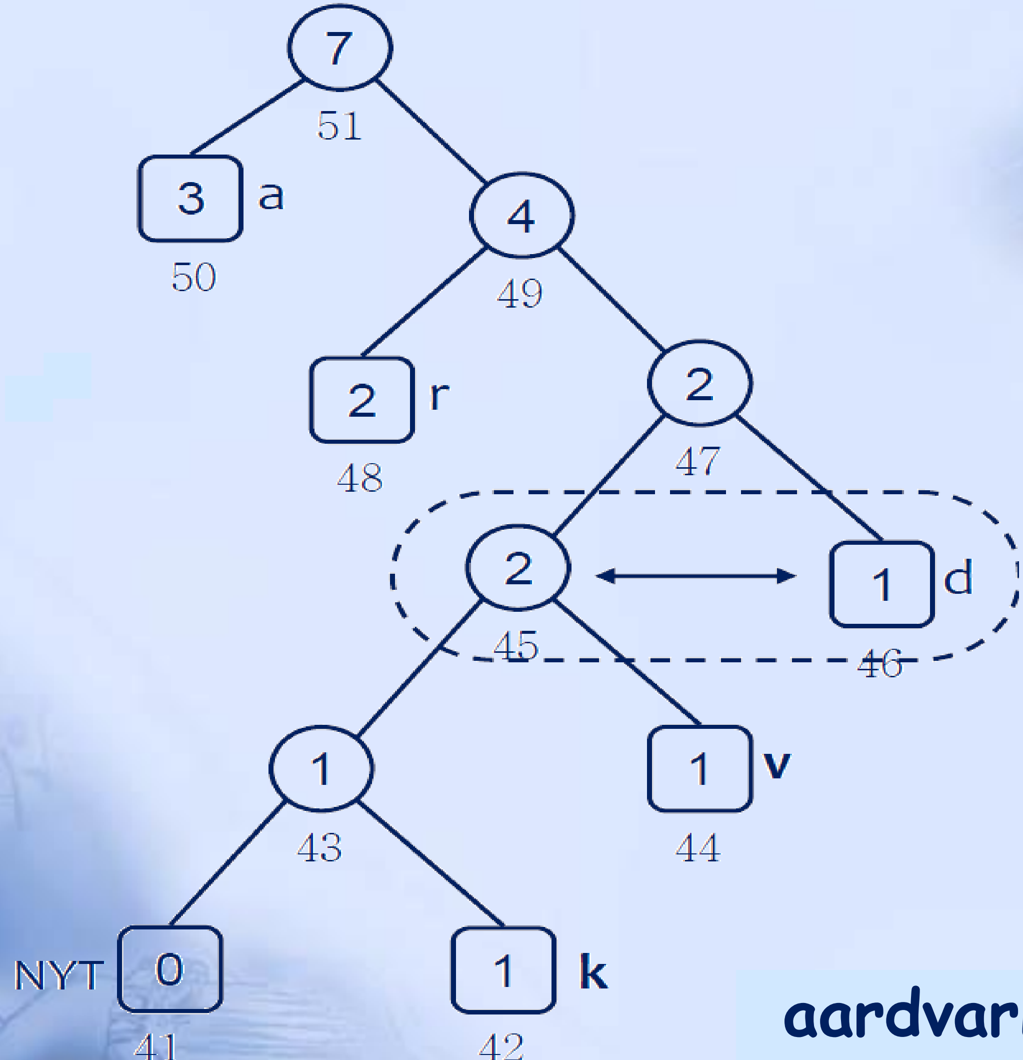
Codificarea Huffman Dinamica Exemplu



aardvark

Ordinea frecventelor: 0,1,1,1,2,1

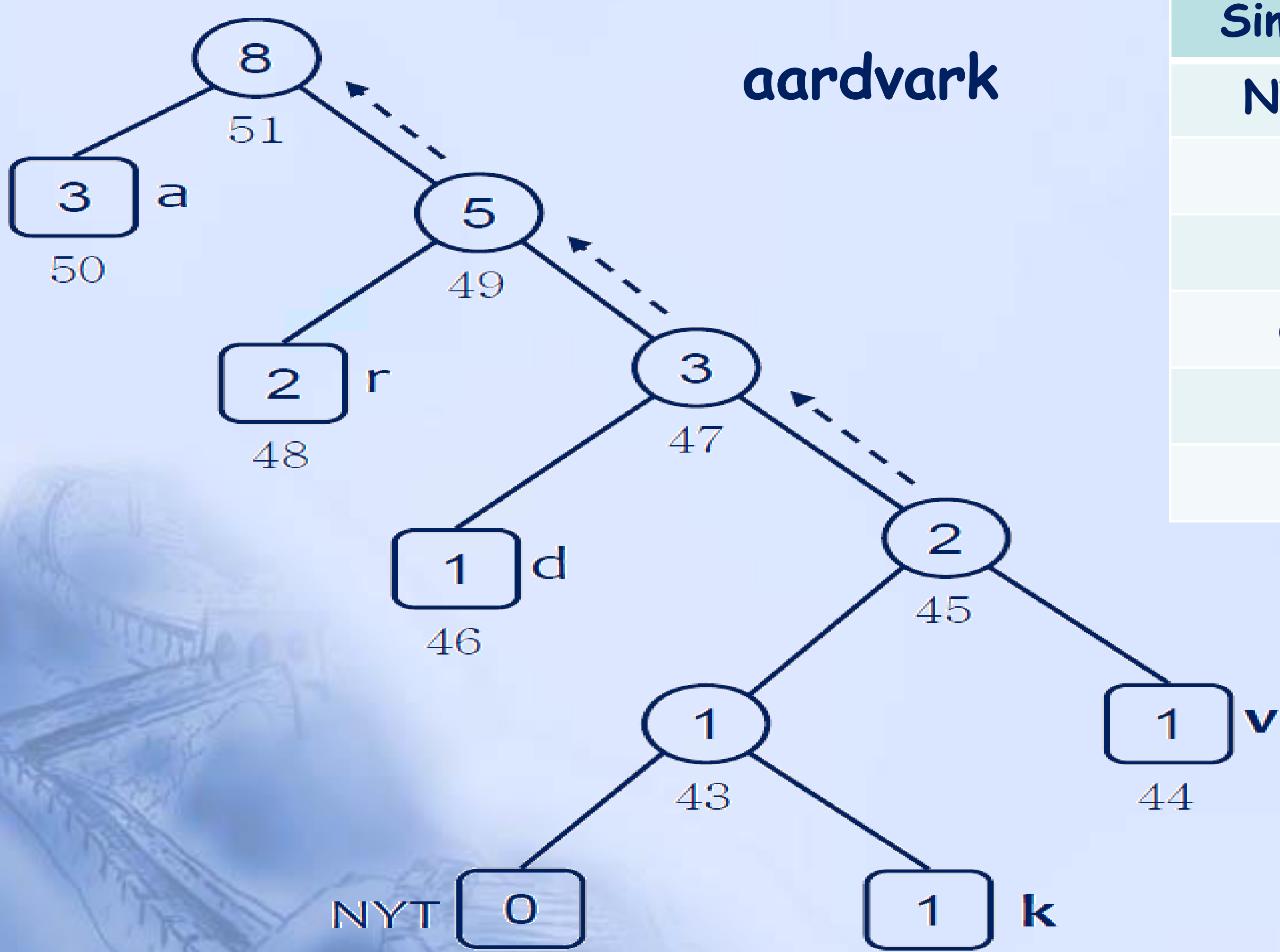
Codificarea Huffman Dinamica Exemplu



aardvark

Ordinea frecventelor: 0,1,1,1,2,1,

Codificarea Huffman Dinamica Exemplu



aardvark

Simbol	Cod
NYT	11100
a	0
r	10
d	110
v	1111
k	11101

Ordinea frecventelor: 0,1,1,1,1,2,2,3,3,5,8

Decodificarea Huffman Dinamica

- Se face litera cu litera pe masura receptionarii mesajului, NYT-ul poarta rol de delimitator.
- Transmitatorul si receptorul sunt sincronizati, in orice moment receptorul poate “vedea” litera trimisa si reconstrui arborele (asignand codul literelor in orice moment), rezulta o decodificare in timp real.

Codificarea Huffman Dinamica vs Statica

- In general codificarea Huffman dinamica este mai performanta decat cea statica (lungimea media a codificarii si costul arborelui sunt mai bune).

- Lungimea media a codificarii este mai buna cu cat textul este mai mare pentru codificarea dinamica.

- Dezavantajul principal al codificarii statice este ca nu accepta modificari in codificare la modificarea textului initial.

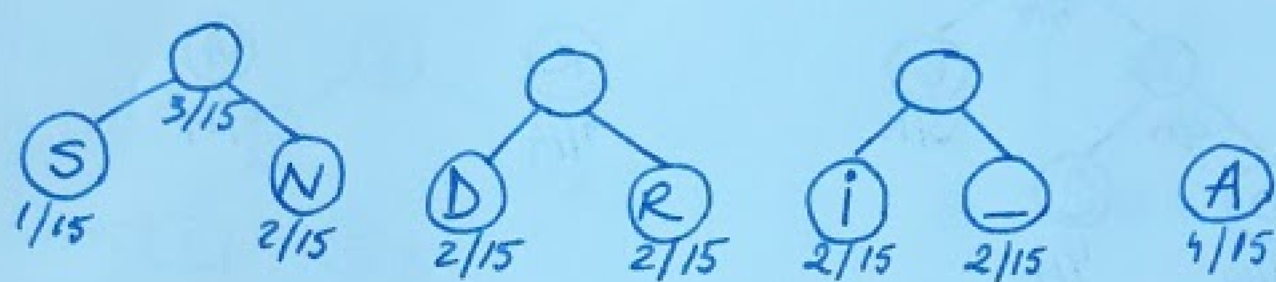
- In codificarea dinamica receptorul are posibilitatea de a-si reconstrui arborele, pe cand in cea dinamica acesta trebuie transmis.

Tema 6 FAi

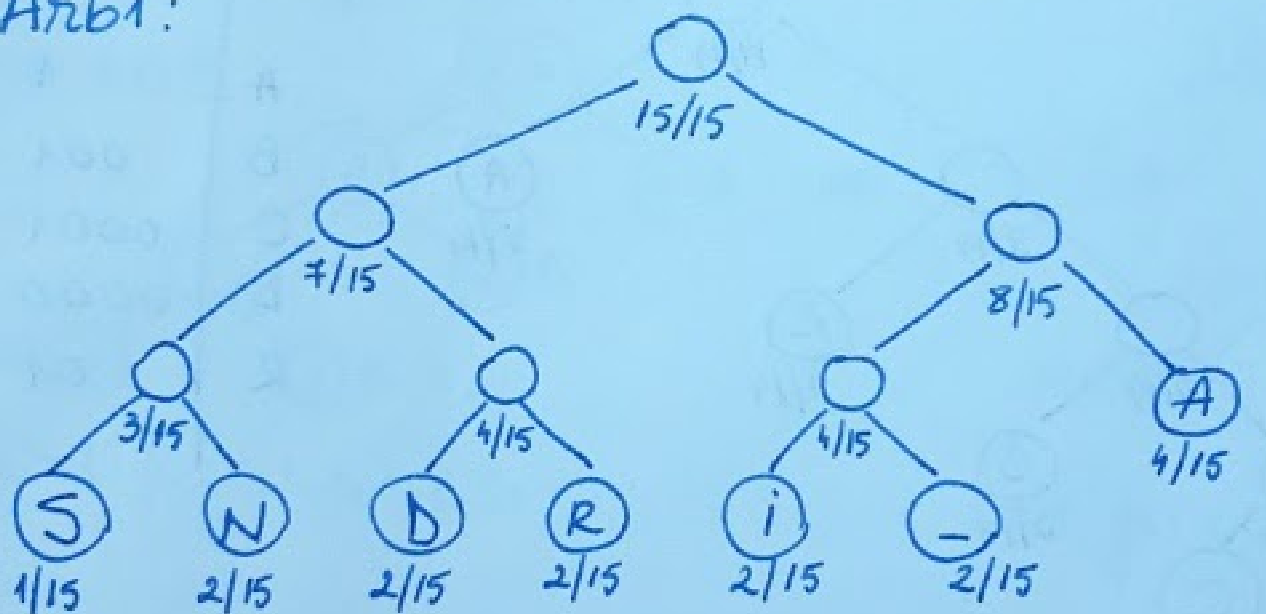
① ANDRA_Si_ADRiAN

$p(A)=4/15,\ p(N)=2/15,\ p(D)=2/15,\ p(R)=2/15,\ p(i)=2/15,\ p(\_)=2/15$

$p(S)=1/15$



Arb1:

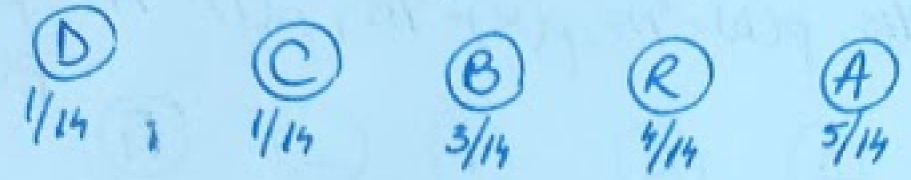


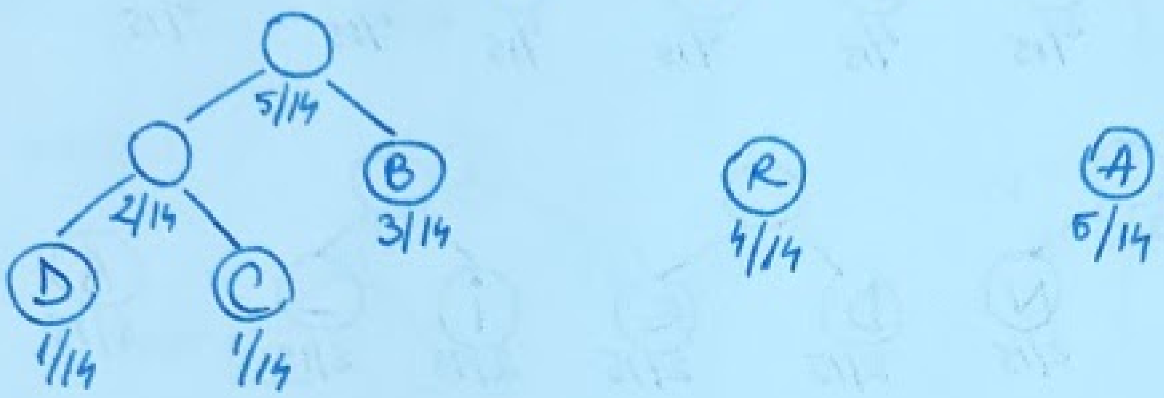
$Cost\ (Arb1) = 4\cdot 2 + 2\cdot 3 + 2\cdot 3 + 2\cdot 3 + 2\cdot 3 + 2\cdot 3 + 1\cdot 3 = 41$

A	11
D	101
i	100
N	001
R	011
S	000
_	101

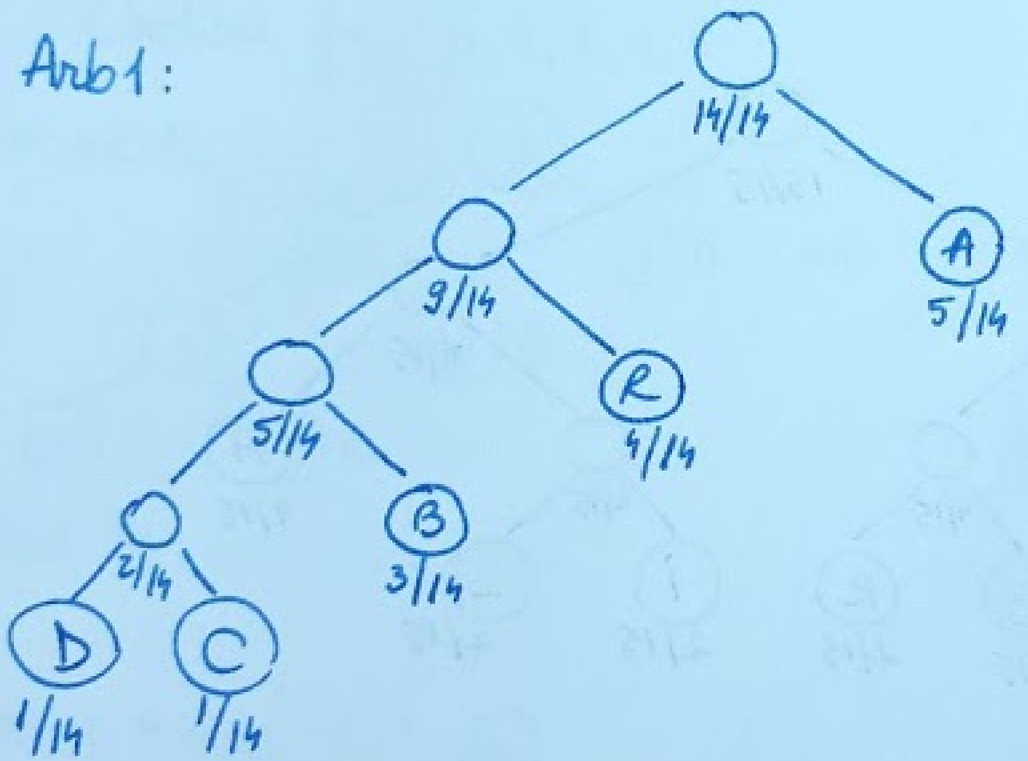
② ABRACADABBRRRRA

$p(A) = 5/14$, $p(R) = 4/14$, $p(B) = 3/14$, $p(C) = 1/14$, $p(D) = 1/14$





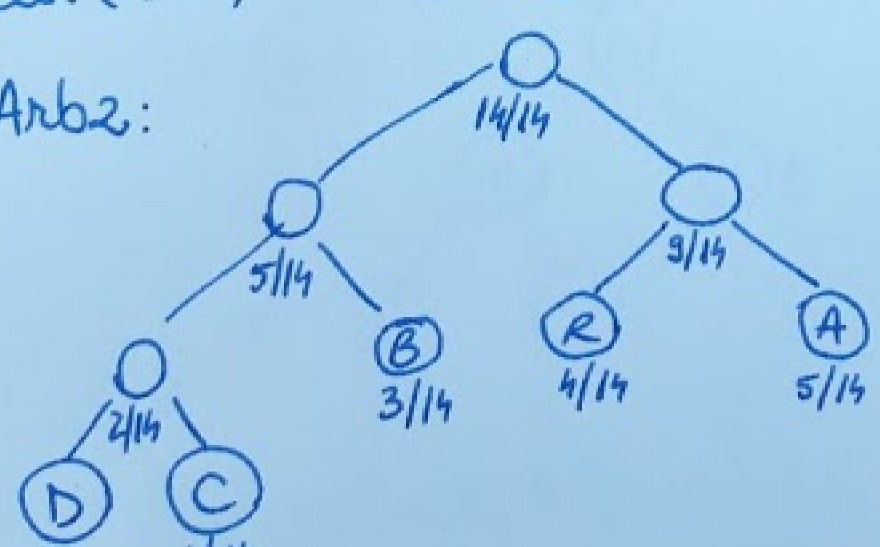
Arb1:



A	1
B	001
C	0001
D	0000
R	01

$Cost(Arb1) = 5\cdot 1 + 2+4 + 3\cdot 3 + 1\cdot 4 + 1\cdot 4 = 30$

Arb2:



A	11
B	01
C	001
D	000
R	10

$Cost(Arb2) = 1\cdot 3 + 1\cdot 3 + 3\cdot 2 + 4\cdot 2 + 5\cdot 2 = 30 = Cost(Arb1)$

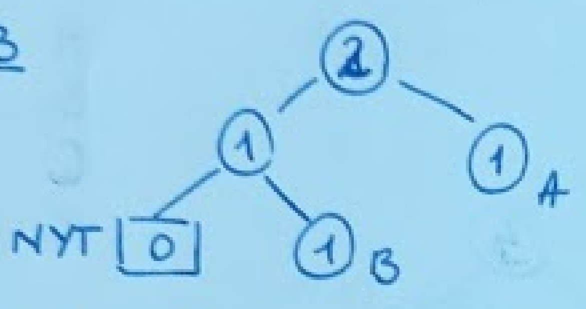
Tema 7 FAI

① ABRACADABBRRRRA (codificare Huffman dinamică)

→ A

Simbol	Cod
A	1
NYT	0

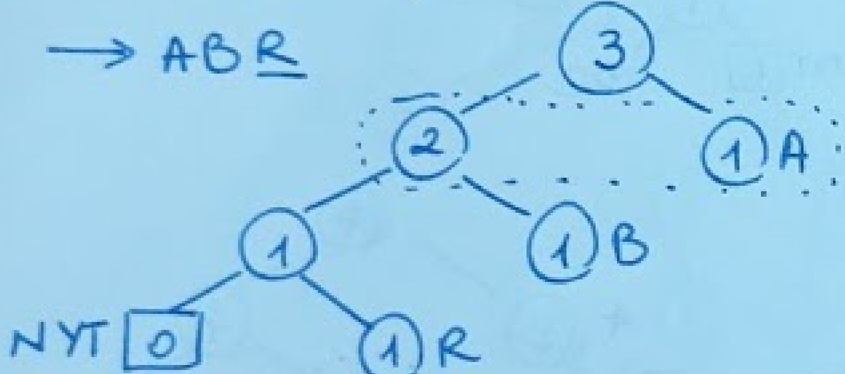
→ AB



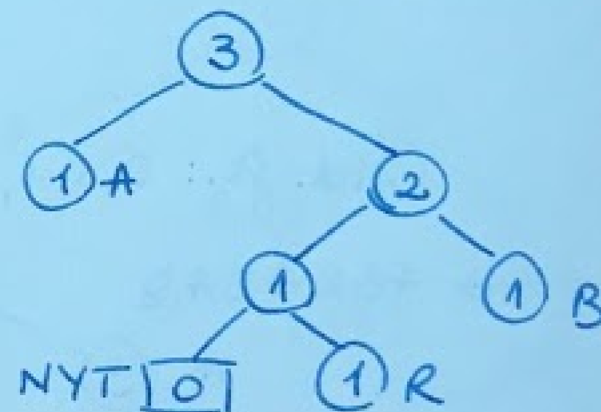
NYT	00
A	1
B	01

Ordine fr.: 0, 1, 1, 1, 1

→ ABR



⇒

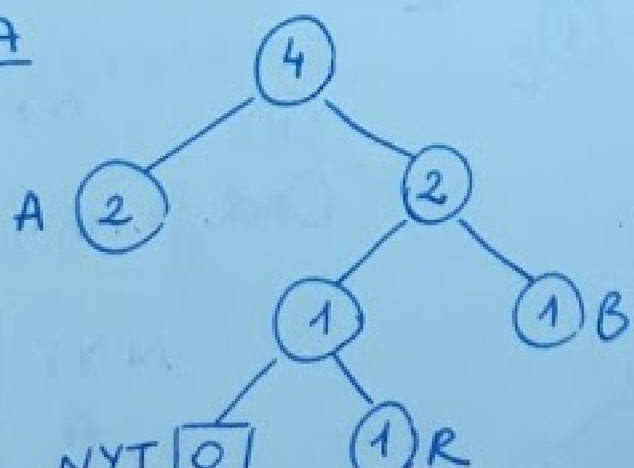


Ord. fr: 0, 1, 1, 1, 2, 1, …

Ord. fr: 0, 1, 1, 1, 1, 2, 3

NYT	100
A	0
B	11
R	101

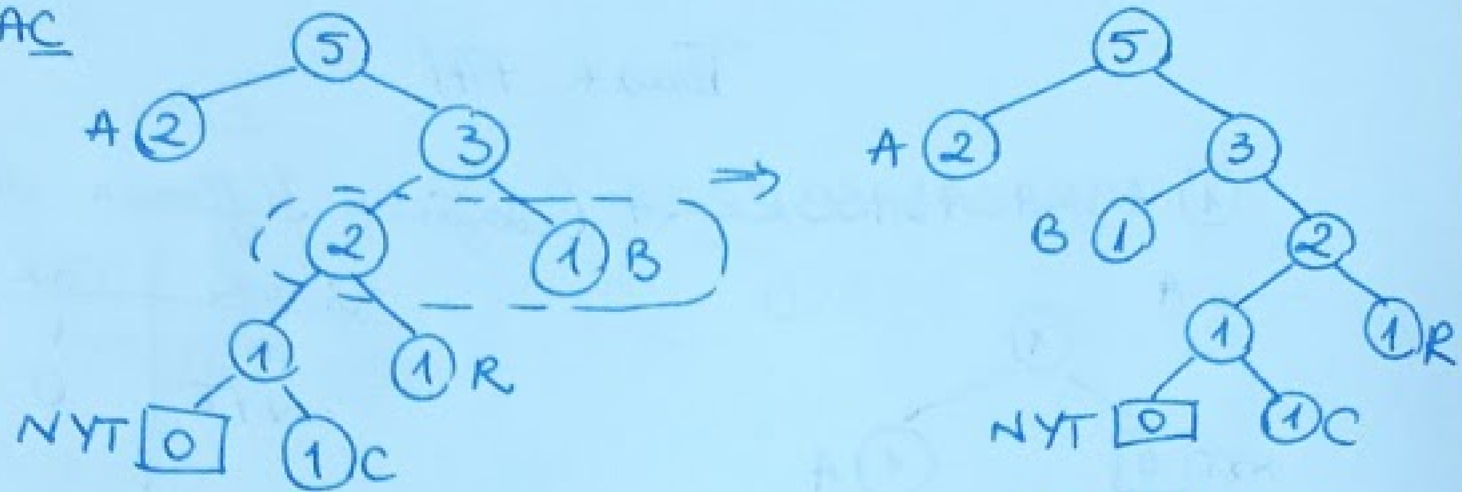
→ ABRA



NYT	100
A	0
B	11
R	101

Ord. fr.: 0, 1, 1, 1, 2, 2, 4

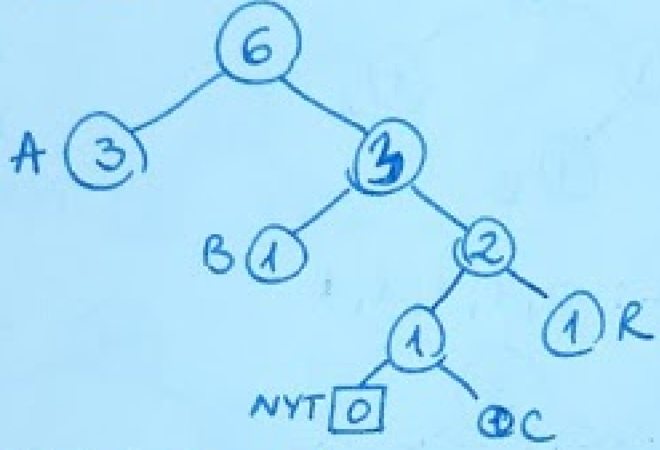
→ ABRA$\underline{C}$



Ord. fr.: 0, 1, 1, 1, $\underline{2, 1}$, …

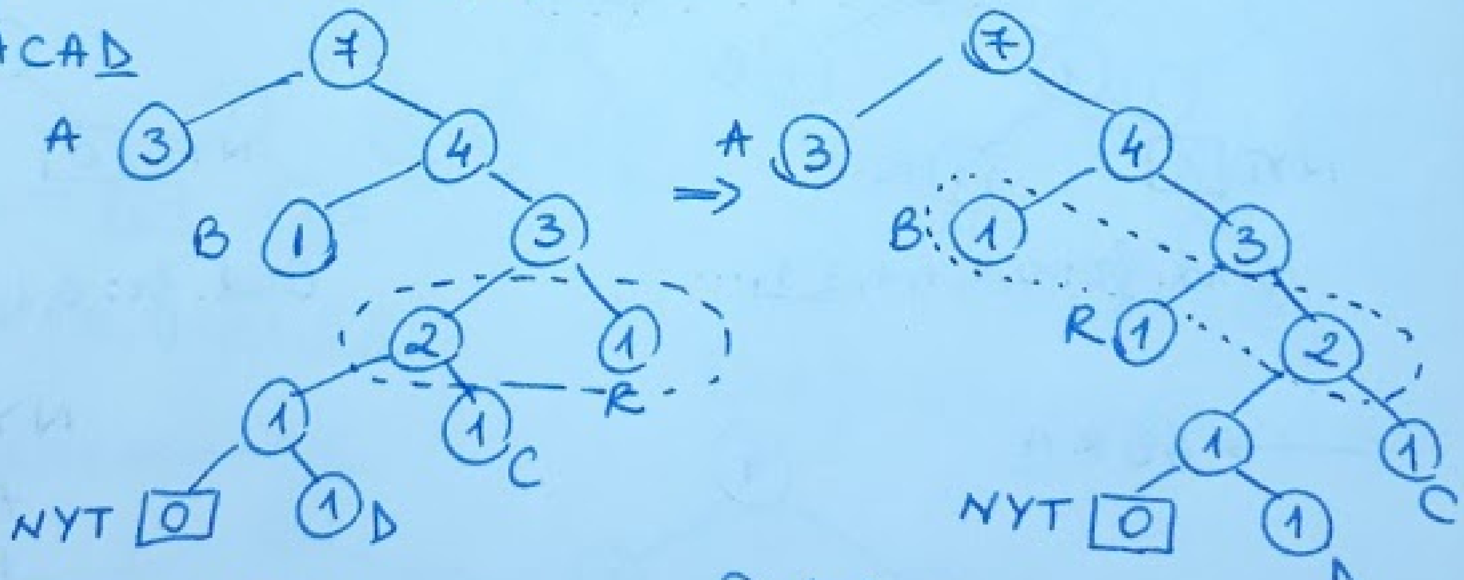
NYT	1100
A	0
B	10
C	1101
R	111

→ ABRAC$\underline{A}$



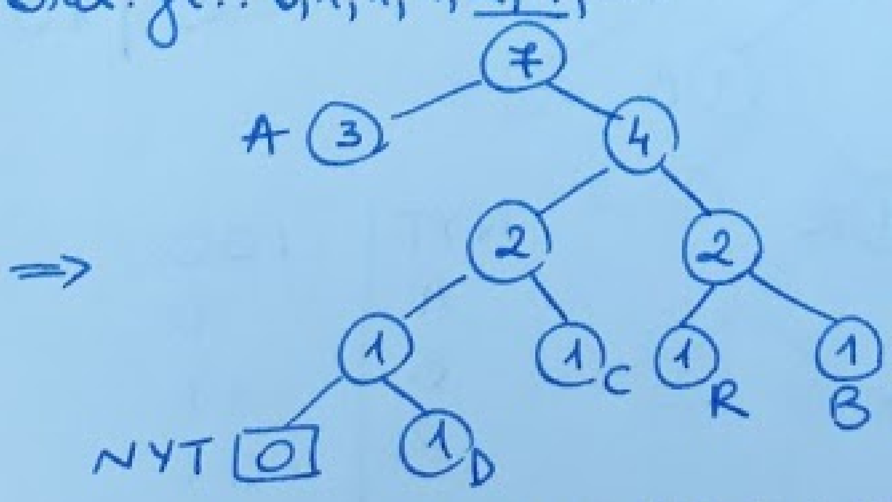
Ord. fr.: 0, 1, 1, 1, 1, 2, 3, 3, 6

→ ABRACA$\underline{D}$



Ord. fr.: 0, 1, 1, 1, $\underline{2, 1}$, …

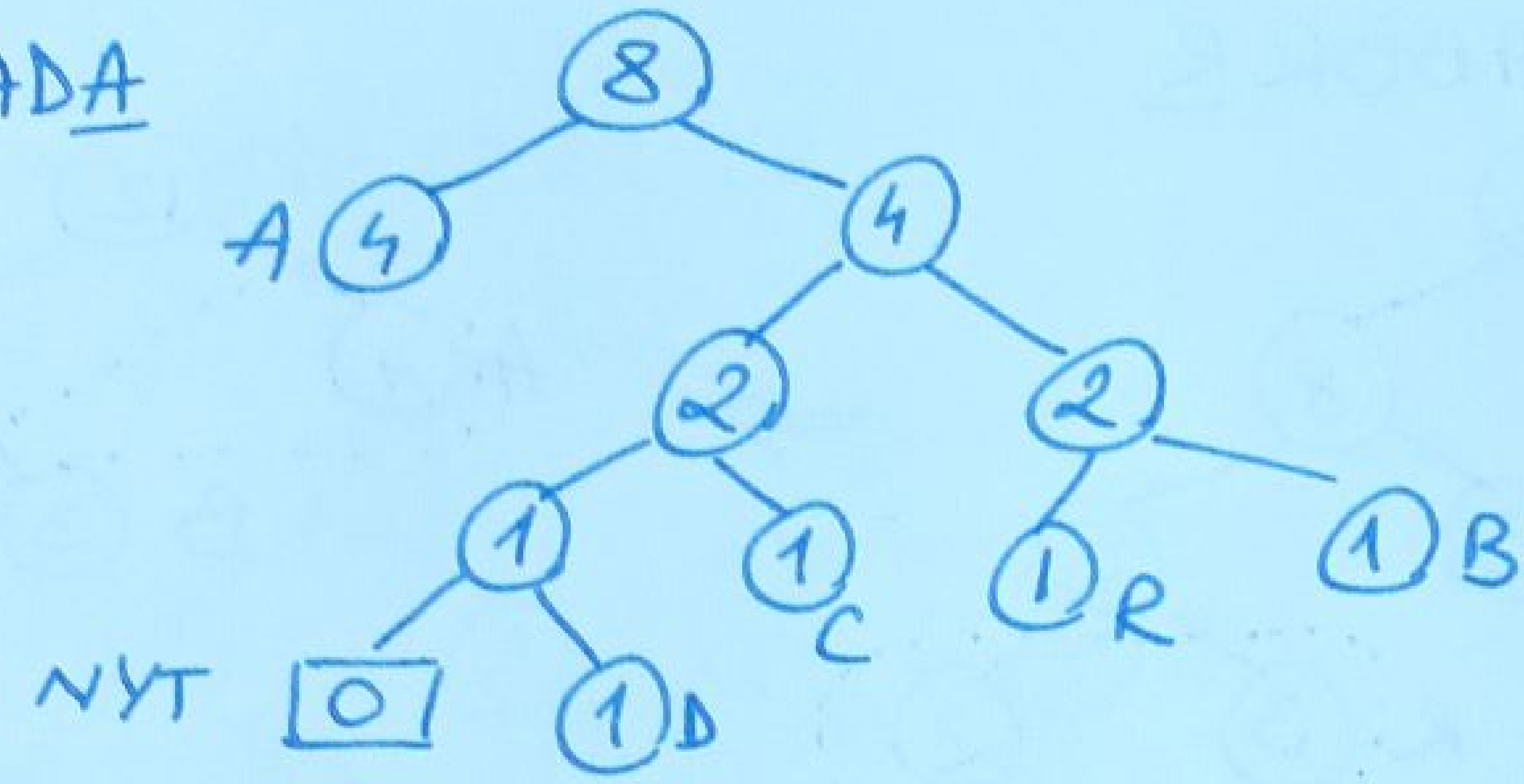
Ord. fr.: 0, 1, 1, 1, 1, $\underline{2, 1}$, …



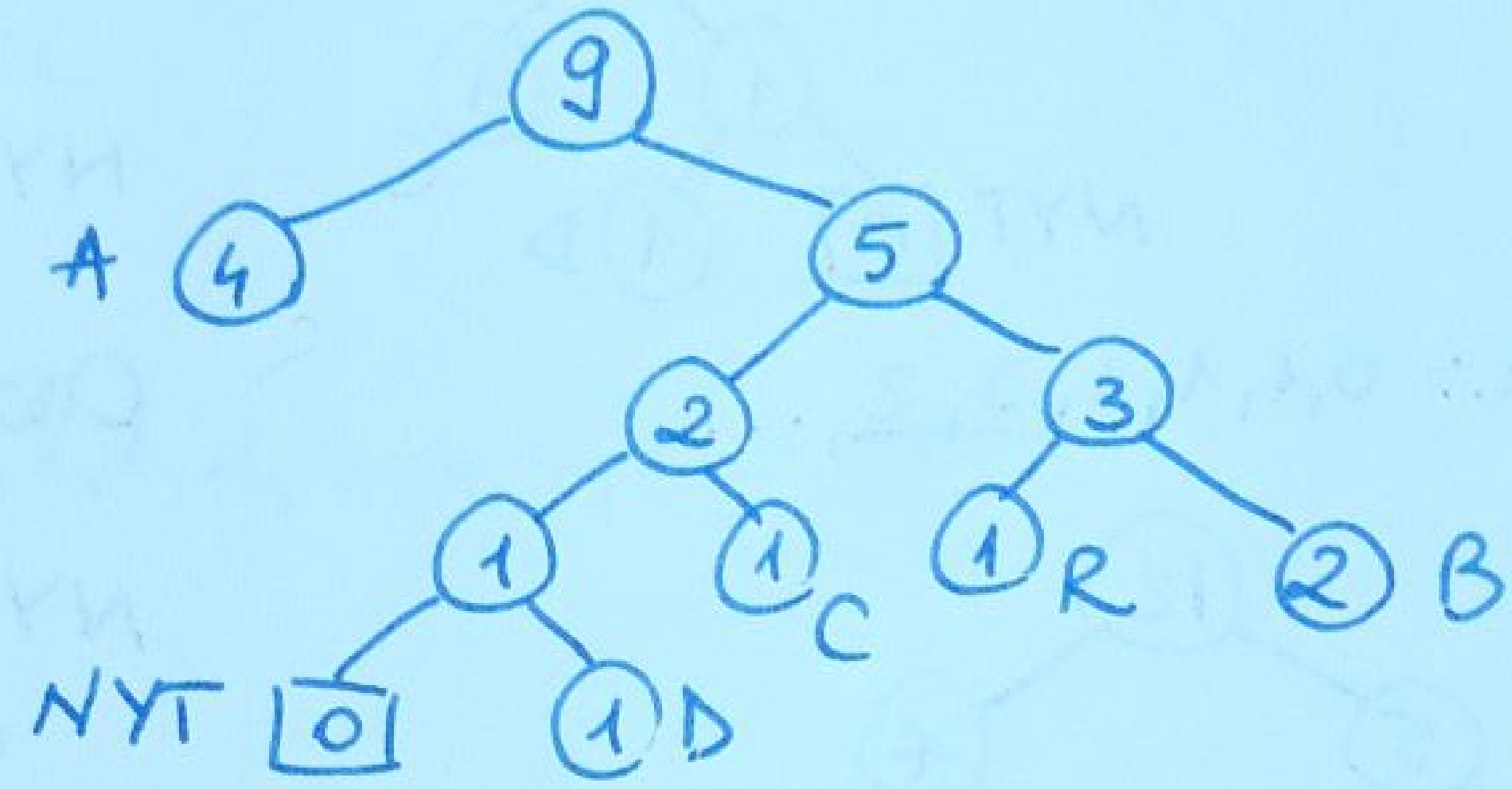
Ord. fr.: 0, 1, 1, 1, 1, 1, 2, 2, 3, 4, 7

NYT	1000
A	0
B	111
C	101
D	1001
R	110

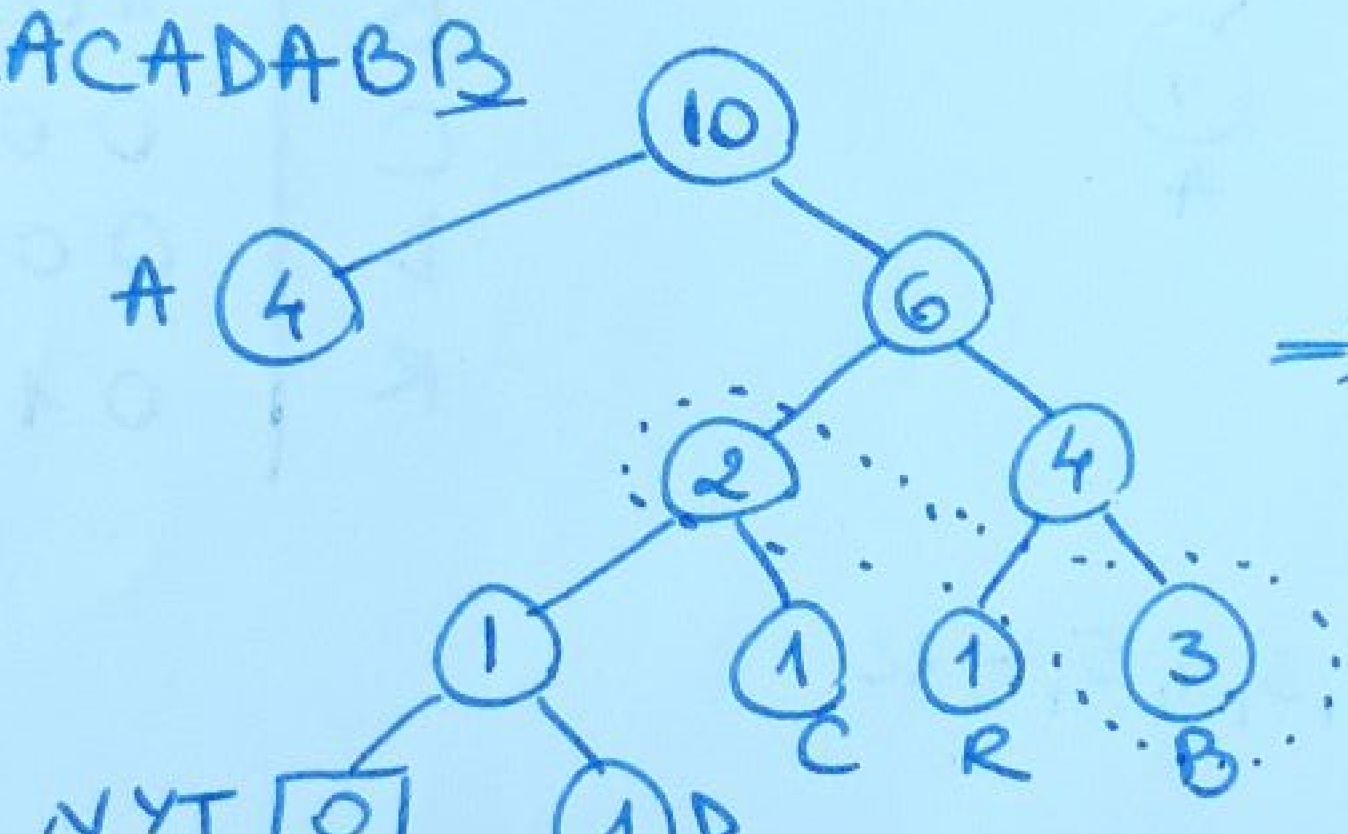
→ ABRACADA



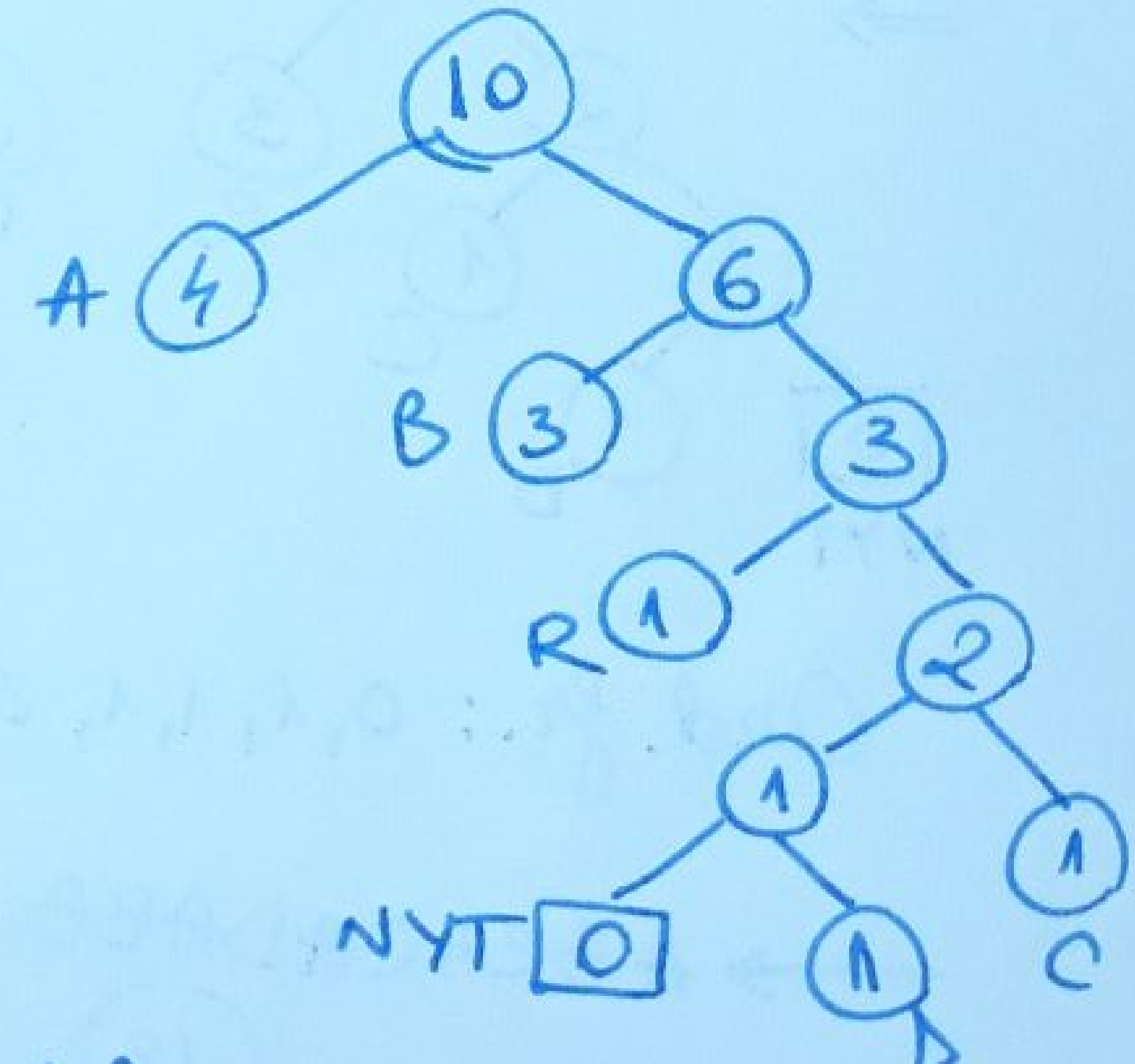
→ ABRACADAB



→ ABRACADABB



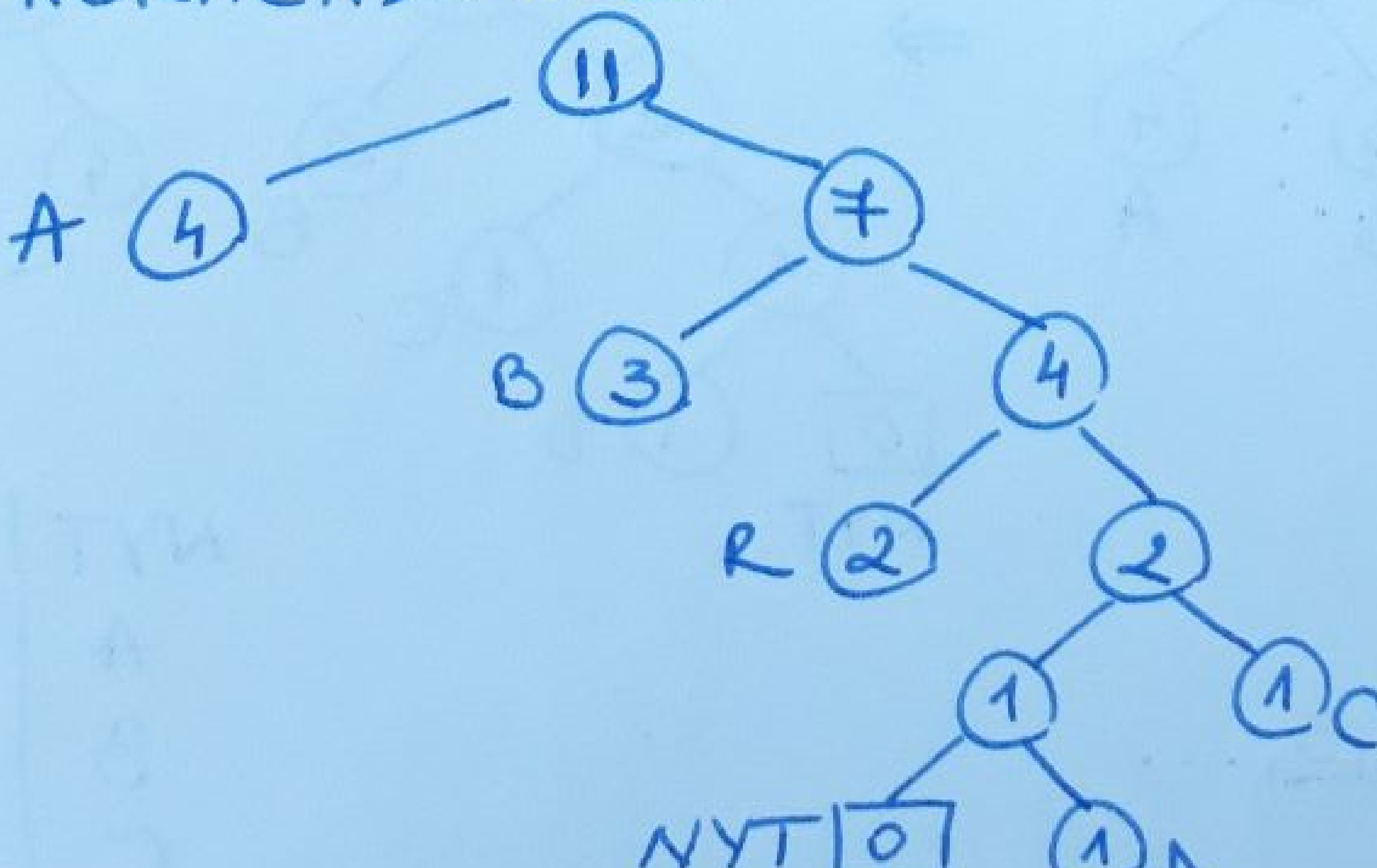
⇒



Ord. fr.: 0, 1, 1, 1, 1, 3, 2, …

Ord. fr.: 0, 1, 1, 1, 1, 2, 3, 3, 4, 6, 10

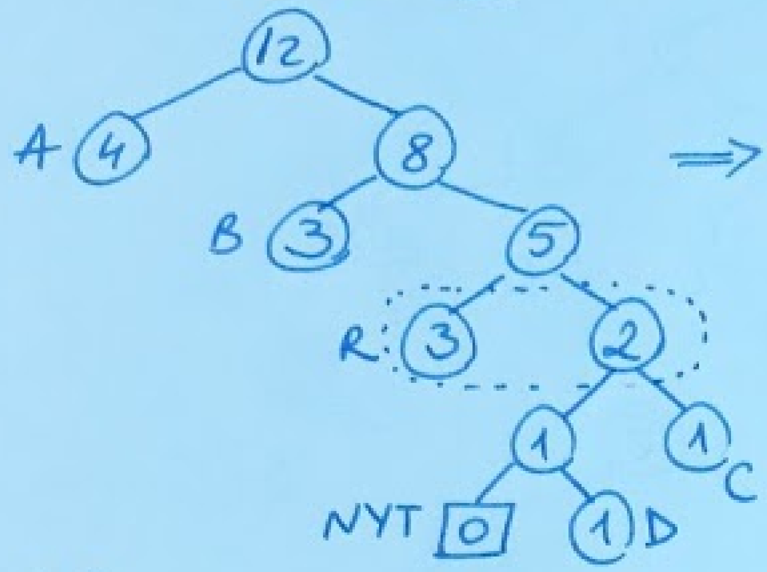
→ ABRACADABBR



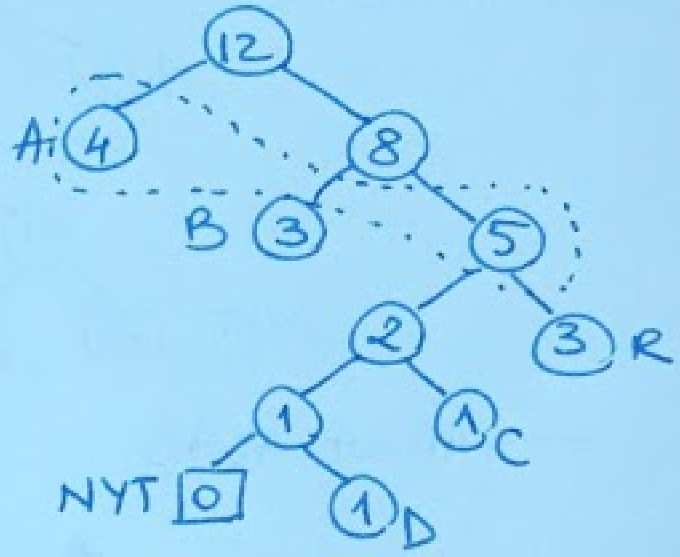
NYT	111 00
A	0
B	10
C	1111
D	11101
R	110

Ord. fr.: 0, 1, 1, 1, 2, 2, 3, 4, 4, 7, 11

→ ABRACADABBR<u>R</u>



⇒

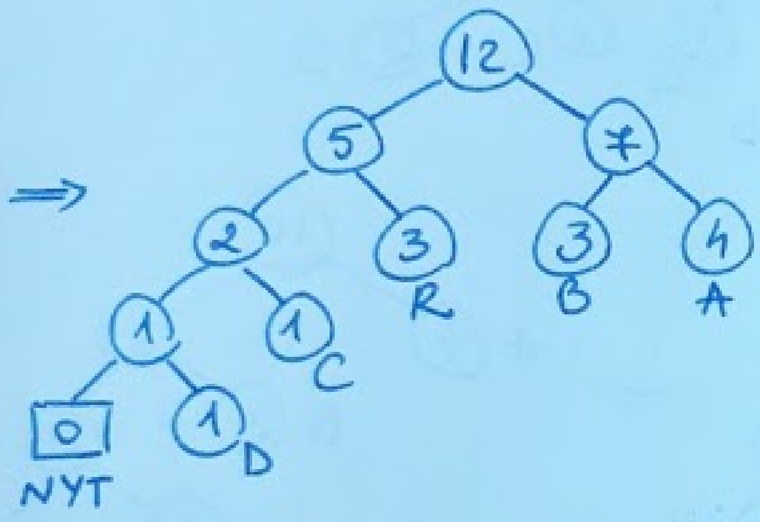


Ord. fr.: 0, 1, 1, 1, <u>3, 2</u>, ...

Ord. fr.: 0, 1, 1, 1, 2, 3, 3, <u>5, 4</u>

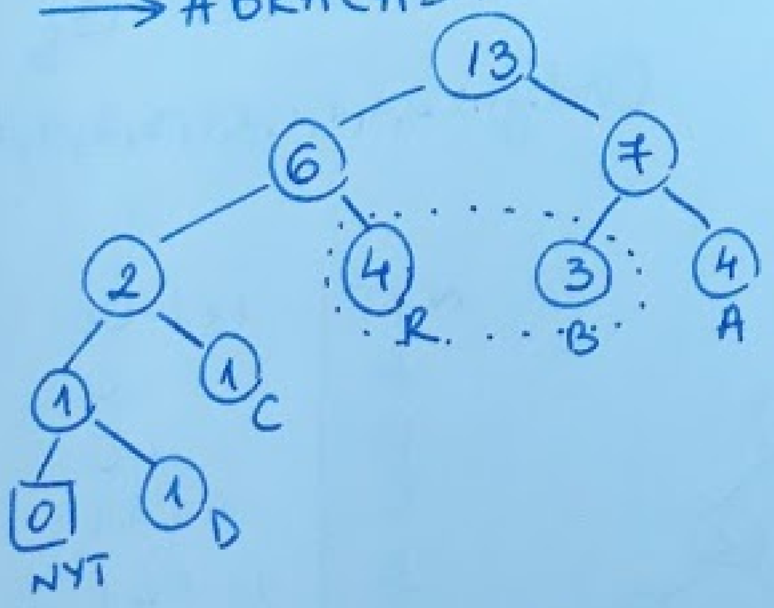
NYT	0000
A	11
B	10
C	001
D	0001
R	01

⇒

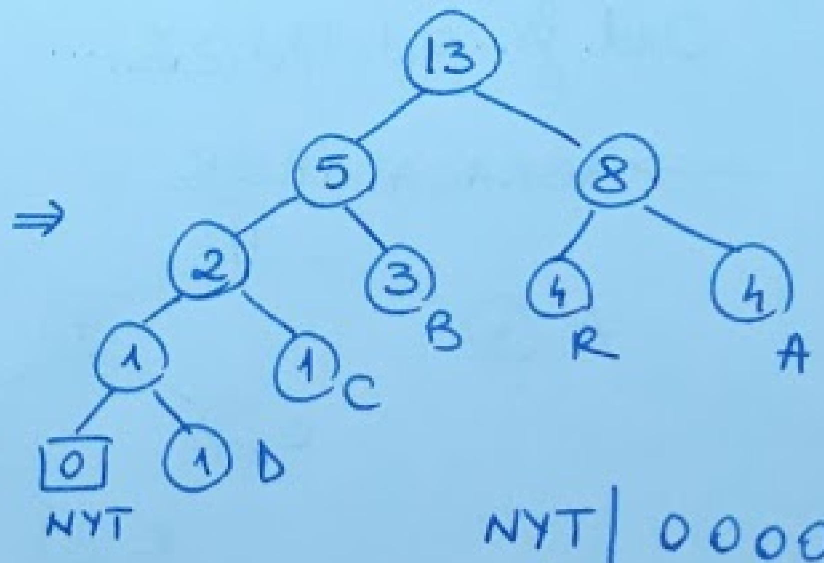


Ord. fr.: 0, 1, 1, 1, 2, 3, 3, 4, 5, 7, 12

→ ABRACADABBRR<u>R</u>



⇒

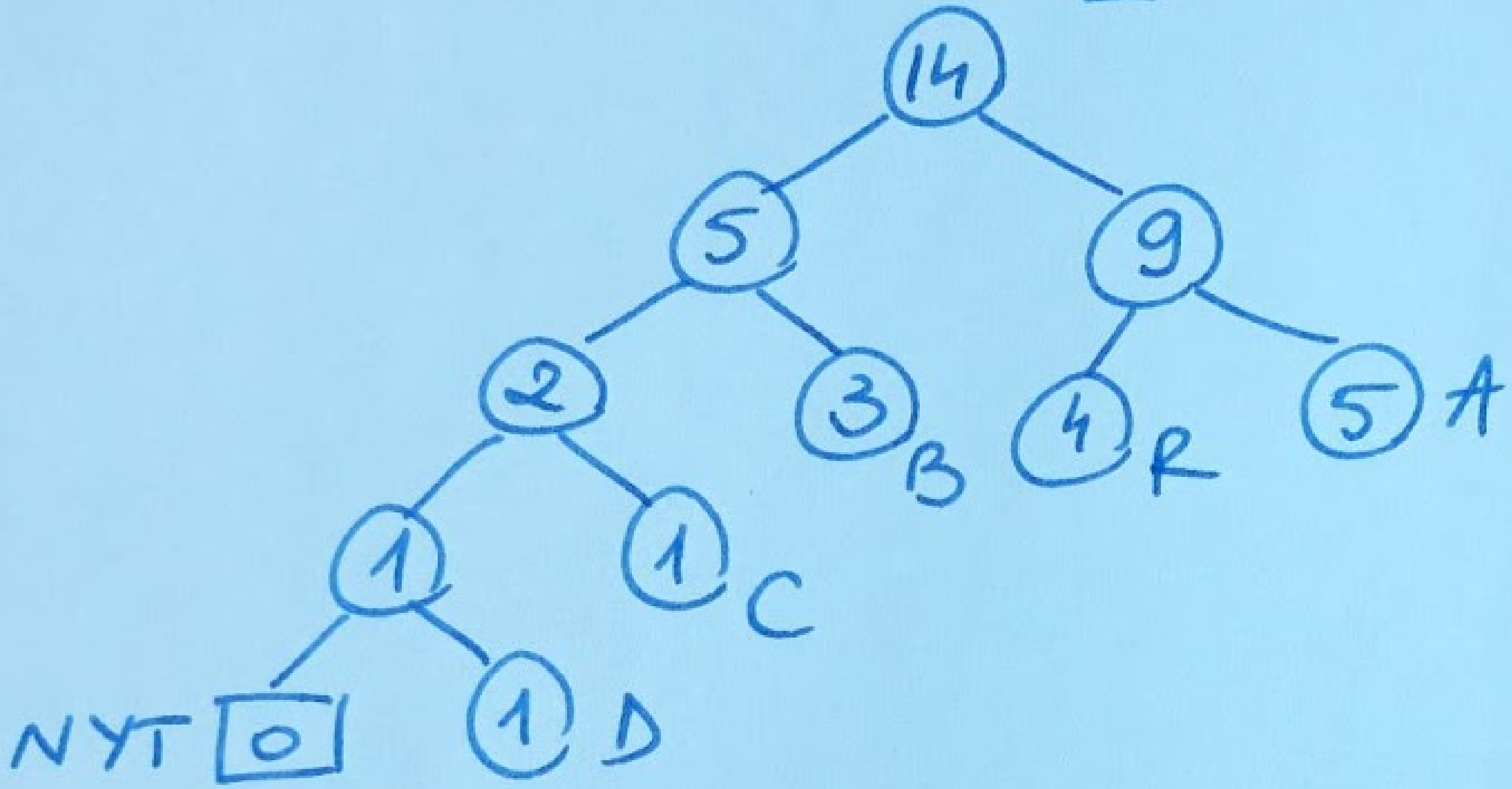


Ord. fr.: 0, 1, 1, 1, 2, <u>4, 3</u>, ...

Ord. fr.: 0, 1, 1, 1, 2, 3, 4, 4, 5, 8, 13

NYT	0000
A	11
B	01
C	001
D	0001
R	10

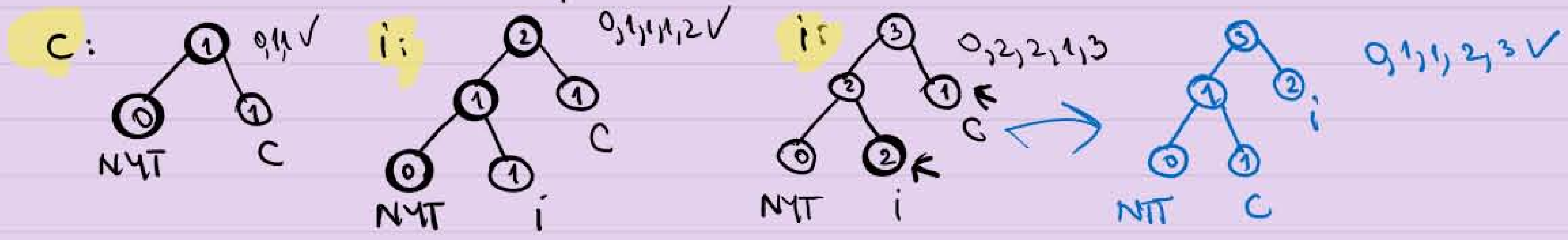
→ ABRACADABBRRR<u>A</u>

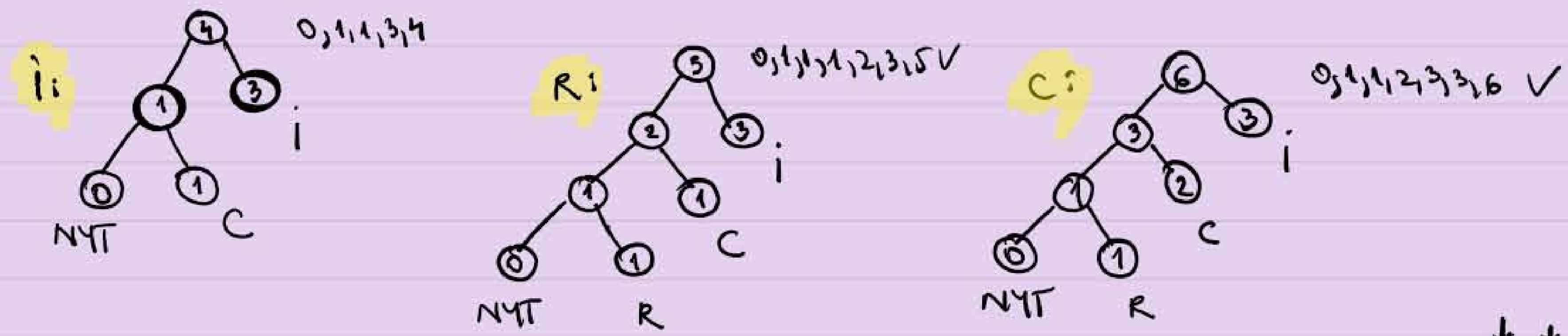


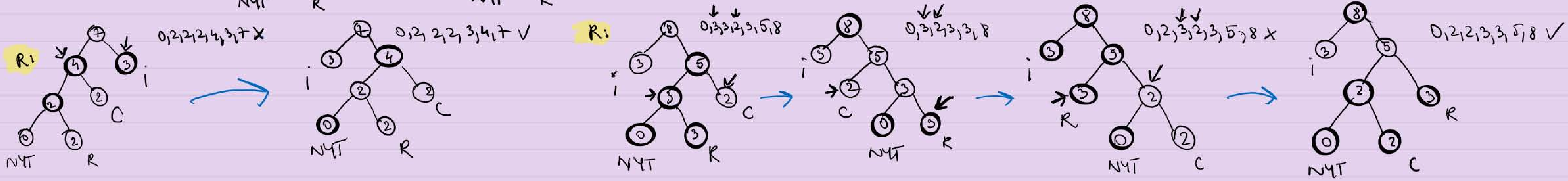
NYT	0000
A	11
B	01
C	001
D	0001
R	10

Ord. fr.: 0, 1, 1, 1, 2, 3, 4, 5, 5, 9, 14

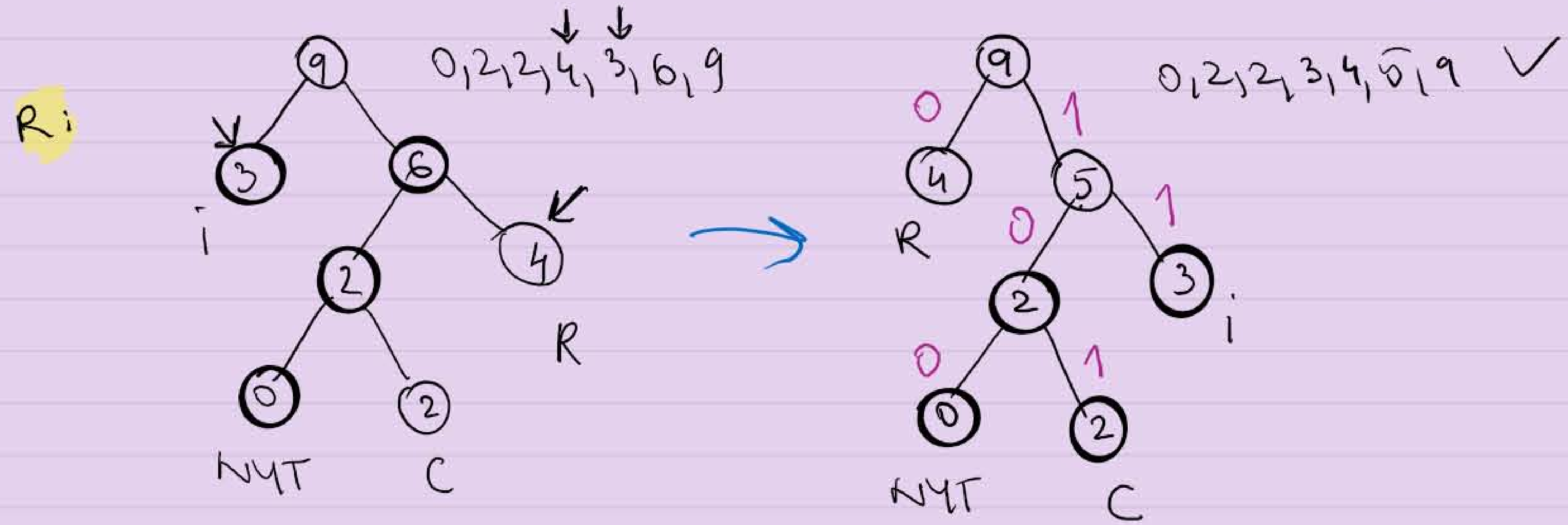
TEXT: CiiiRCRRR







SYMBOL	COD
NYT	100
C	101
i	11
R	0



CODIFICAREA TEXTULUI:

1011111110101000

COSTUL: $2*3 + 3*2 + 4*1 = 6+6+4 = 16$