Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Structuri de Date si Algoritmi

- suport de curs -

Dobrovat Anca - Madalina

An universitar 2019 – 2020
Semestrul I
Seriile 21 + 25

Curs 9

26/11/2019

Curs 9 - Cuprins

5. Arbori binari stricti cu ponderi

Algoritmul lui Huffman.

Aplicatii la codificarea binara.

Aplicatii la interclasarea optimala a mai multor siruri.

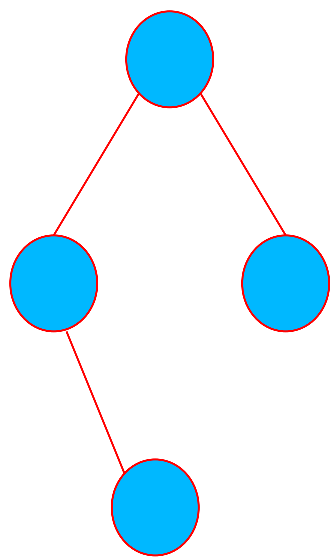
Sursa: – R. Ceterchi: "Structuri de date si Algoritmi. Aspecte matematice si aplicatii", Editura Univ. din Bucuresti, 2001

Arbori binari stricti

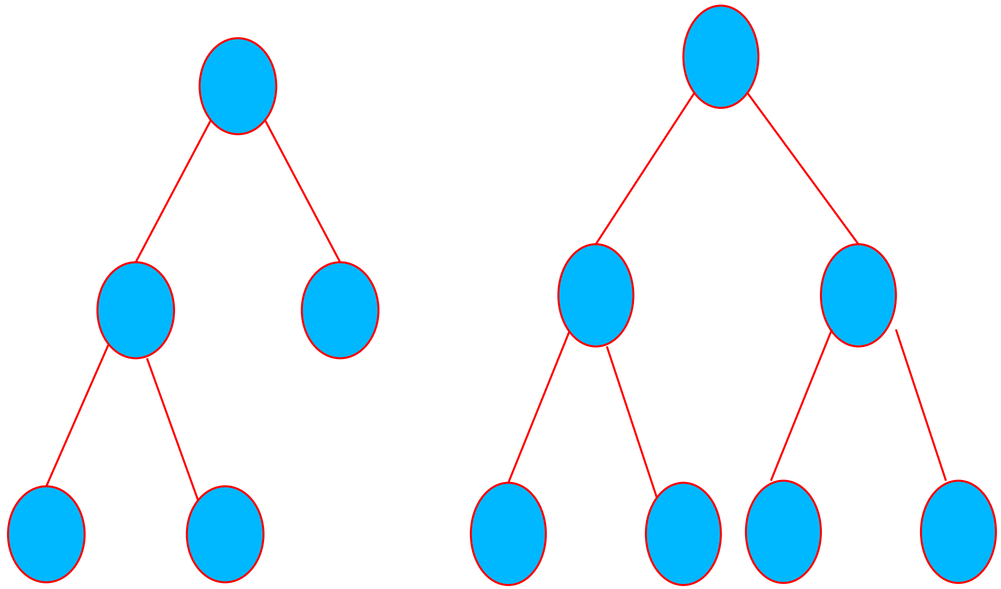
Un arbore binar strict este un arbore binar ın care fiecare nod are fie nici un fiu, fie exact doi fii.

Nodurile cu doi copii se vor numi noduri interne, iar cele fara copii se vor numi noduri externe sau frunze.

Nodurile externe pot fi de alt tip decat nodurile interne



Arbore binar nestrict

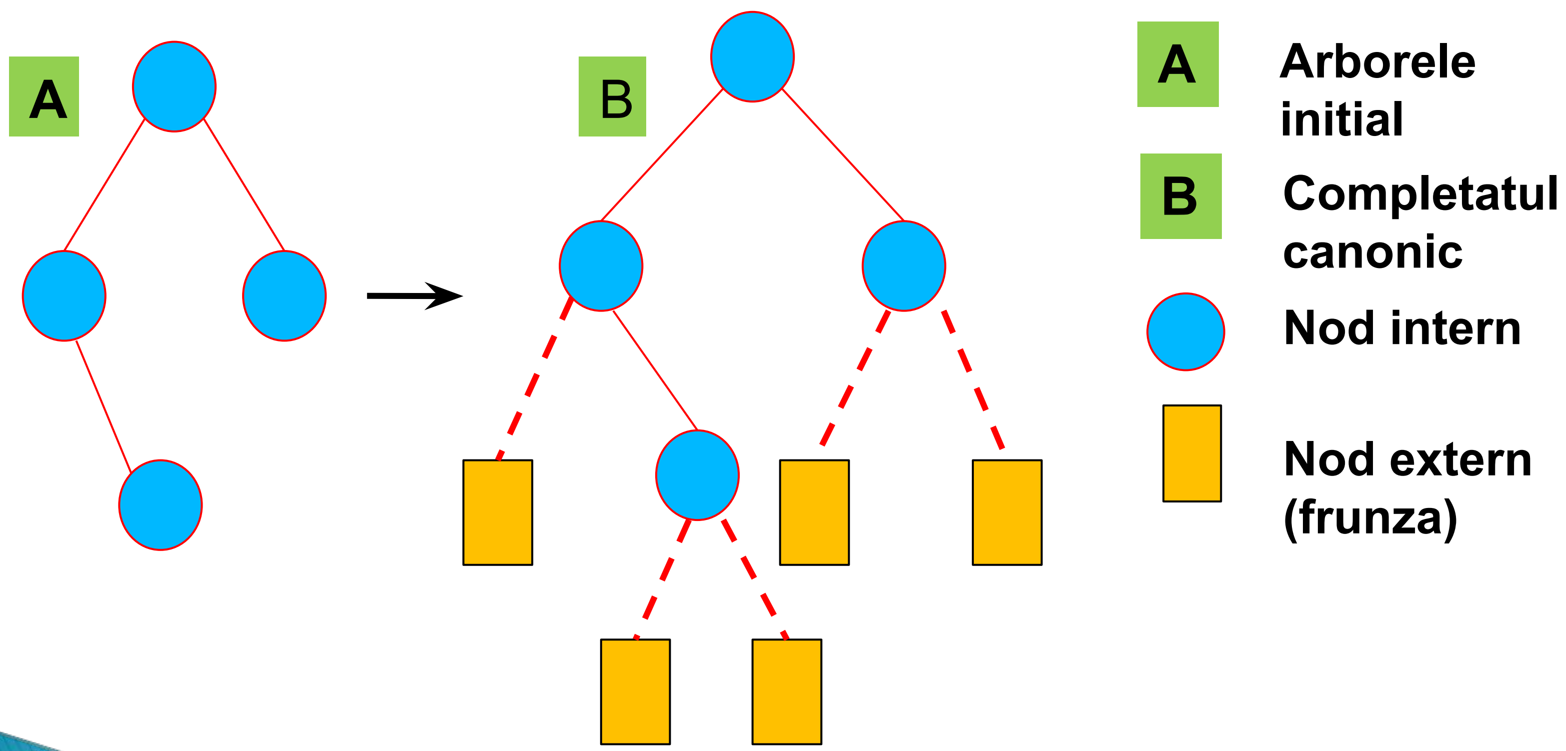


Arbori binari stricti

Arbori binari stricti

Completare canonica a unui arbore binar oarecare la unul strict

Fiecare fiu vid se inlocuieste cu un nod de tip special nodurile arborelui initial devin toate noduri interne, iar cele adaugate canonic, vor fi frunze.

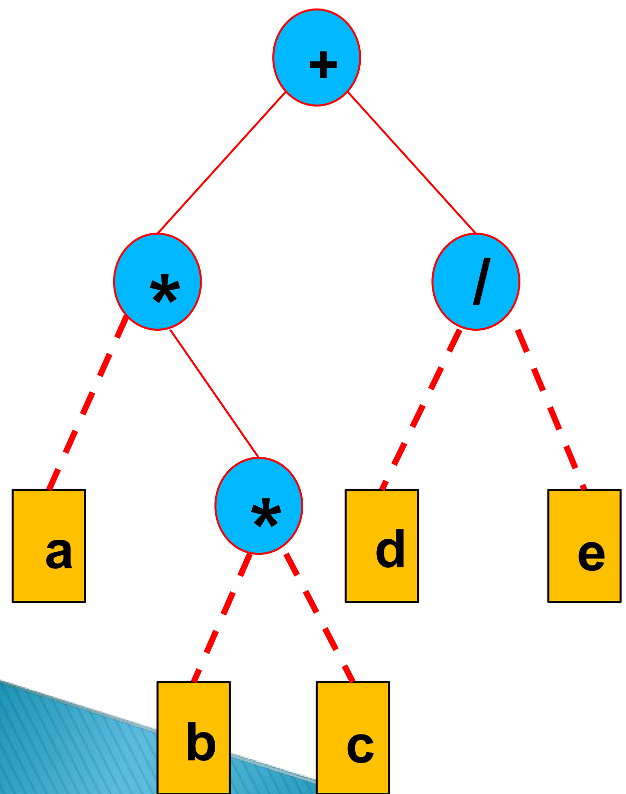


Arbori binari stricti

Exemple de aplicatii ale structurii de arbore binar strict (ABS)

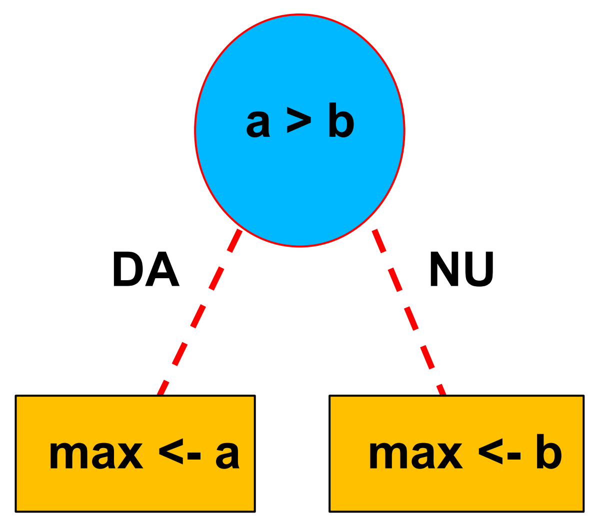
1) Reprezentari de expresii aritmetice cu operatori binari

E = a * b * c + d / e



2) Reprezentarea procedurilor de decizie

Daca (a>b) atunci
max <- a
Altfel
max <- b



Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Arbori binari stricti

Exemple de aplicatii ale structurii de arbore binar strict (ABS)

3) Reprezentarea algoritmilor

Ordonarea a 3 numere

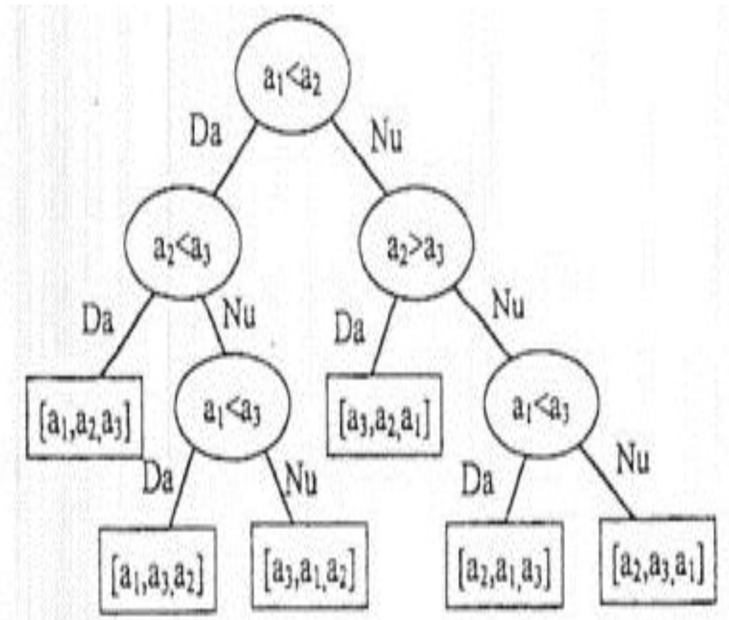
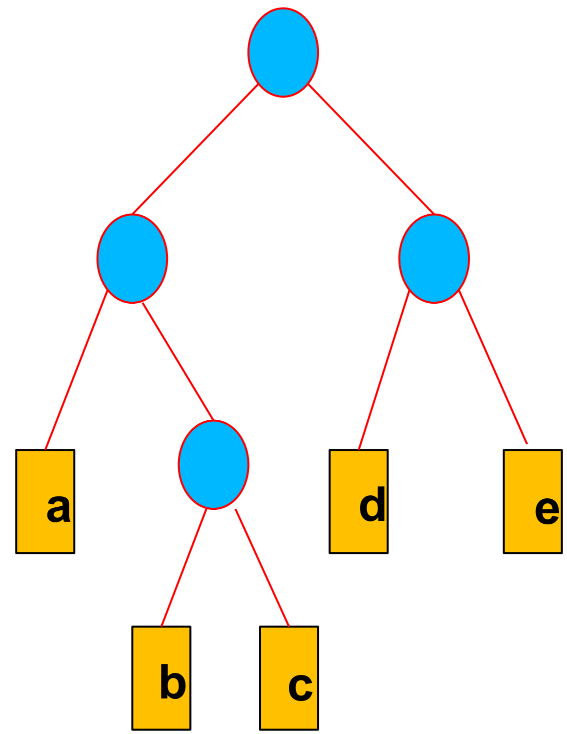


Fig.4.1.6 Arbore binar strict asociat sortării mulţimii $\{a_1, a_2, a_3\}$.

4) Aplicatii la Codificarea Binara

Exemple de Coduri binare peste alfabetul {a, b, c, d, e} asociat ABS



a – 00
b – 010
c – 011
d – 10
e – 11

Sursa imagine: R. Ceterchi - "Structuri de date si Algoritmi. Aspecte matematice si Aplicatii (2001)"

Arbori binari stricti

Notatii

N_E = numărul nodurilor externe ale unui arbore binar strict
N_I = numărul nodurilor interne
E = mulţimea frunzelor
I = mulţimea nodurilor interioare
r = rădăcina
$l(r, x)$ = lungimea drumului de la r la nodul x (măsurat în număr de arce)

L_E = ***Lungime externă*** a unui arbore binar strict = suma lungimilor drumurilor de la rădăcină până la fiecare nod extern.

$$L_E = \sum_{x \in E} l(r, x)$$

L_I = ***Lungime internă*** a unui arbore binar strict = suma lungimilor drumurilor de la rădăcină la toate nodurile interioare.

$$L_I = \sum_{y \in I} l(r, y) .$$

Arbori binari stricti

Proprietati

Propoziţia 1. Într-un arbore binar strict, numărul nodurilor externe şi al celor interne sunt legate prin relaţia: $N_E = N_I + 1.$

Propoziţia 2. Într-un arbore binar strict este adevarata relaţia: $L_E = L_I + 2N_I$

Propoziţia 3. Într-un arbore binar strict de adâncime *d* avem următoarea inegalitate.

$$N_E \leq 2^d \ .$$

Corolar. Într-un arbore binar strict de adâncime *d* avem inegalitatea

$$d \geq \lceil log_2 N_E \rceil.$$

Arbori binari stricti

Proprietati

Propoziţia 4. Dintre toţi arborii binari stricţi cu acelaşi număr de frunze, fixat, N_E, au **lungime externă minimă** aceia cu proprietatea că **frunzele** lor **sunt repartizate pe cel mult două niveluri adiacente.**

Propoziţia 5. Lungimea externă minimă a unui arbore binar strict cu N_E frunze este dată de formula:

$$L_E^{min} = N_E \lfloor \log_2 N_E \rfloor + 2(N_E - 2^{\lfloor \log_2 N_E \rfloor}).$$

Propoziţia 6. Într-un arbore binar strict avem următoarea inegalitate

$$L_E^{medie} \geq \lfloor \log_2 N_E \rfloor.$$

Arbori binari stricti cu ponderi

Def: Fie T un arbore binar strict si mulţimea frunzelor *E = {a_1,a_2, …, a_n}*.

***T* se va numi *cu ponderi* dacă există o funcţie cu valori reale *w:E R*, cu alte cuvinte, dacă fiecare frunză a_i are asociat un număr real w_i numit *ponderea* ei.**

L_E = ***Lungime externă*** a unui arbore binar strict = suma lungimilor drumurilor de la rădăcină până la fiecare nod extern.

$$L_E = \sum_{x \in E} l(r, x)$$

Lungimea externă ponderata

$$L_E^w = \sum_{i=1}^{n} w_i l_i = \sum_{i=1}^{n} w_i l(r, a_i).$$

Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Arbori binari stricti cu ponderi

Probleme

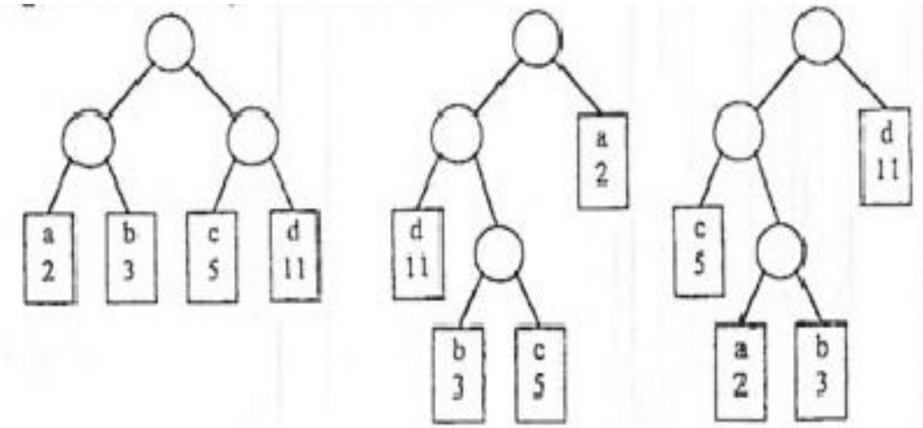
1) **Q:** Pentru care din arborii binari stricti neponderati se minimizeaza lungimea externa?

   A: (Prop 4 pt ABS) – La un numar fixat de frunze, lungimea externa se minimizeaza pentru acei ABS care au frunzele distribuite pe maxim 2 niveluri adiacente.

2) **Q:** Date fiind *n* ponderi, notate $w_1, w_2, .., w_n$, să se găsească printre toţi arborii binari stricţi cu *n* frunze, unul, nu neapărat unic, care să aibă cea mai mică lungime externă ponderată.

   A: ALGORITMUL LUI HUFFMAN

Exemplu: variatia lungimii externe ponderate



(A) L=42 (B) L=48 (C) L=36

Sursa imagine: R. Ceterchi - "Structuri de date si Algoritmi. Aspecte matematice si Aplicatii (2001)"

Algoritmul lui Huffman

Problema

Date fiind *n* ponderi, notate $w_1, w_2, .., w_n$, să se găsească printre toţi arborii binari stricţi cu *n* frunze, unul, nu neapărat unic, care să aibă cea mai mică lungime externă ponderată

Rezolvare

Presupunem date *n* ponderi, în ordine descrescătoare

$$w_1 >= w_2 >= \ldots >= w_{n-1} >= w_n.$$

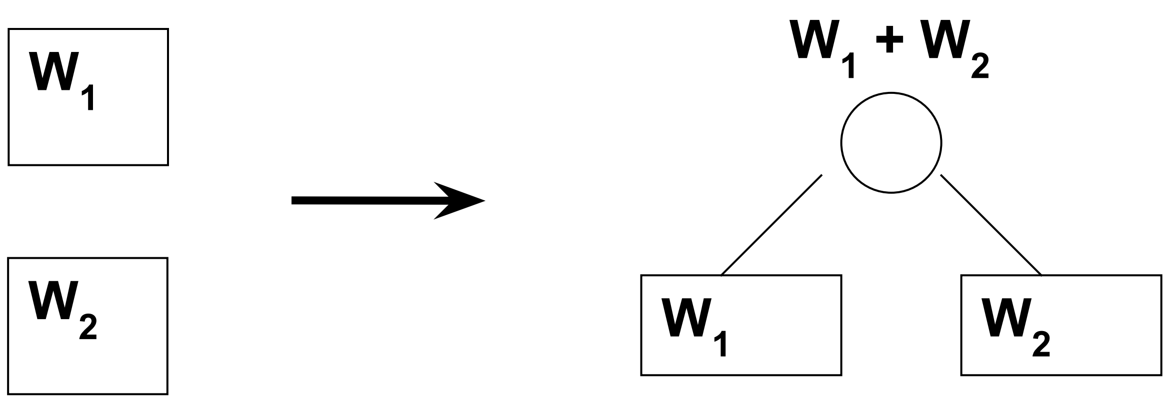
***Pasul 1*. Algoritmul formează *n* arbori binari stricţi, fiecare de tip frunză, cu câte o pondere w_i asociată ei. Avem o *pădure* cu *n* arbori.**

Algoritmul lui Huffman

Rezolvare

Pasul 2. Se „leagă“ doi subarbori cu ponderi minime (presupunem, fara a restrange generalitatea, ca cele 2 ponderi minime sunt W_1 si W_2 cu ajutorul unui nod interior pentru care ei devin cei doi fii,

- acest arbore va fi arbore binar strict cu ponderea asociată egală cu suma ponderilor arborilor pe care i-am legat, pondere pe care o vom asocia nodului intern.



Algoritmul lui Huffman

Rezolvare

Pasul 2 **reduce cu 1 numarul subarborilor din padure.**

Se reia pasul iterativ (2) pînă când obţinem un singur arbore.

Un arbore binar strict obtinut prin aplicarea algoritmului lui Huffman se va numi *arbore Huffman* asociat ponderilor $\{w_1, w_2, \ldots, w_n\}$.

Obs. 1: *algoritmul* ia in considerare doar *ponderile*. *Informatia* continuta in frunze este *importanta* pentru aplicatii, natura ei *difera* de la o aplicatie la alta.

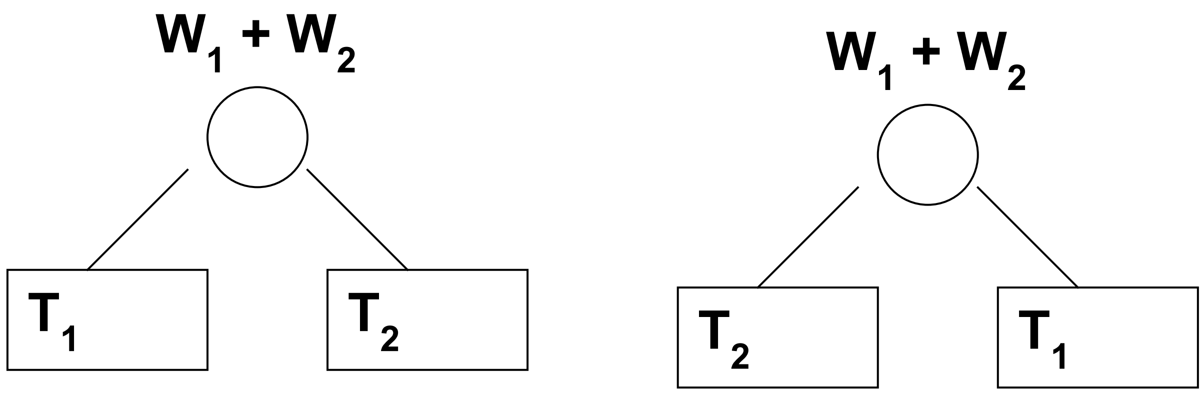
Obs. 2: arborele Huffman NU este in general unic.

Algoritmul lui Huffman

Rezolvare

Neunicitatea arborelui Huffman

La fiecare legare a 2 arbori T1 si T2 avem 2 posibilitati:

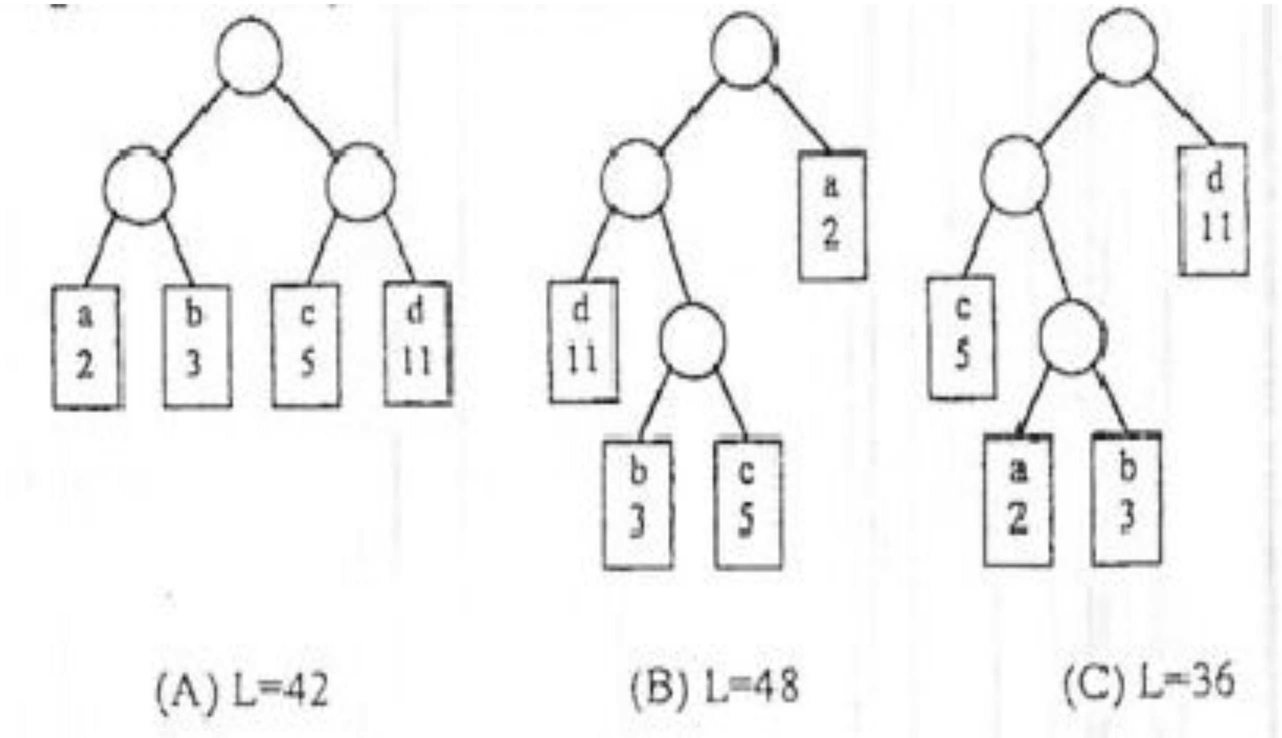


Ambiguitatea poate fi rezolvata, de exemplu, prin conventia ca, la fiecare legare a 2 arbori DE PONDERI DIFERITE, arborele de pondere minima sa devina fiu stang

Algoritmul lui Huffman

Rezolvare

Expl. Trei a.b.s. pt. frunzele cu ponderi: *(a, 2), (b, 3), (c, 5), (d, 11)*.



Arborele din fig. (C) este Huffman, L = 36 minima.

Aplicatii ale ABS la codificare binara

Notatii

V - un alfabet finit;

V* - multimea "cuvintelor" (sirurilor de caractere) formate peste V, plus cuvantul vid;

{0,1}* - multimea sirurilor ce contin doar 0 si 1;

- poate fi identificata cu multimea tuturor sirurilor de biti

Def. **cod binar pe V o functie c:V--->{0,1}*, injectiva. Codul unei litere *a* din V: c(a) – contine doar 0 si 1.**

Injectivitatea asigura ca 2 litere distincte din V au coduri distincte.

Se extinde in mod canonic la o functie c:V*--->{0,1}*:

$$c(a_1 a_2 \dots a_n) = c(a_1)c(a_2) \dots c(a_n).$$

Prop. **Si extinderea canonica este injectiva.**

Aplicatii ale ABS la codificare binara

Def. **Coduri cu proprietatea prefix: pt. oricare x,y din V, c(x) nu este prefix al lui c(y).**

Unui cod binar pe V, cu proprietatea prefix, i se poate asocia un arbore binar, cu arce etichetate cu 0 (fiu stg.) si 1 (fiu dr.). Codul literei x, c(x) = sirul de etichete al drumului unic de la radacina pina la un nod ce contine x.

Un astfel de cod:

este *injectiv* – orice 2 litere se afla in frunze diferite ;

are *proprietatea prefix* – caracterele un sunt reprezentate in noduri interioare

Def. **Un cod este de lungime fixa daca toate literele din V se codifica cu siruri de biti de aceeasi lungime.**

Un cod de lungime variabila nu are proprietatea de mai sus.

Aplicatii ale ABS la codificare binara

Coduri Hufmann - observatii

Tehnica foarte utilizata si eficienta pentru compactarea datelor;

In functie de caracteristicile fisierului care trebuie comprimat, spatiul economisit este între 20% si 90%.

Algoritmul greedy utilizeaza un tabel cu frecventele de aparitie ale fiecarui caracter.

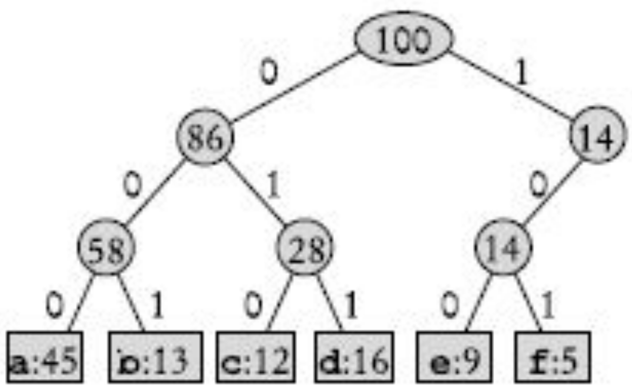
Ideea este de a utiliza o modalitate optima pentru reprezentarea fiecarui caracter sub forma unui sir binar.

Expl. **Fisier ce contine 100.000 de caractere, pe care dorim sa îl memoram într-o forma compactata. Exista doar sase caractere diferite si numarul de aparitii al fiecarui caracter este diferit.**

Aplicatii ale ABS la codificare binara

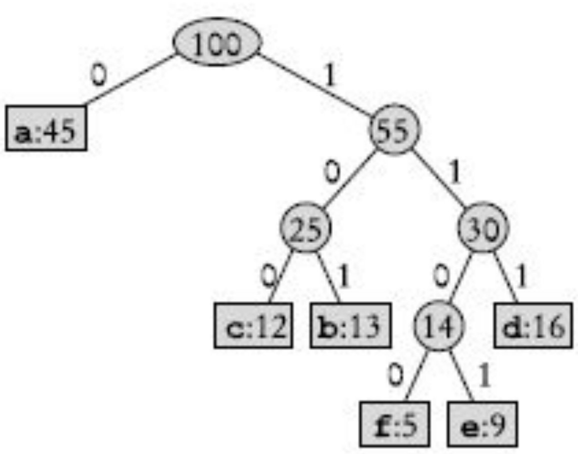
Coduri Hufmann - observatii

Expl. **Fisier ce contine 100.000 de caractere, pe care dorim sa îl memoram într-o forma compactata. Exista doar sase caractere diferite si numarul de aparitii al fiecarui caracter este diferit.**



Arborele corespunzator codificarii de lungime fixa.

Codificarea necesita 300000 biti (lungimea externa ponderata)



Arborele asociat codificarii prefix optime.

Codificarea necesita 224000 biti (lungimea externa ponderata)

Aplicatii ale ABS la codificare binara

Coduri Hufmann - observatii

Algoritm.

V = o multime de n caractere

Fiecare caracter *c din V este un obiect având o frecventa data f[c].*

***Q = coada de prioritati* pentru a identifica cele doua obiecte cu frecventa cea mai redusa care vor fuziona.**

```
Huffman(V)
n <- dimensiunea lui V
Q <- V
pentru i <- 1; n ; 1 executa
z <- Aloca-Nod()
x <- f[z] <- Extrage-Min(Q)
y <- f[z] <- Extrage-Min(Q))
f[z] <- f[x] + f[y]
Insereaza(Q; z)
returneaza Extrage-Min(Q)
```

Aplicatii ale ABS la codificare binara

Coduri Hufmann - observatii

Corectitudinea algoritmului

Pentru a demonstra ca algoritmul de tip greedy al lui Huffman este corect, vom arata ca problema determinarii unei codificari prefix optime implica alegeri greedy si are o substructura optima.

Lema 1 ***(*****se refera la proprietatea alegerii greedy*****)***
Fie V un alfabet în care fiecare caracter c din V are frecventa f[c]. Fie x si y doua **caractere din V** ***având cele mai mici frecvente. Atunci exista o codificare prefix optima pentru V în care cuvintele de cod pentru x si y au aceeasi lungime si difera doar pe ultimul bit.***

Dem **- Ideea demonstratiei este de a lua arborele** ***T reprezentând o codificare prefix*** **optima si a-l modifica pentru a realiza un arbore reprezentând o alta codificare prefix optima. În noul arbore, caracterele** ***x si y vor apare ca frunze cu acelasi tata si se vor afla pe nivelul maxim*** **în arbore.**

Aplicatii ale ABS la codificare binara

Coduri Hufmann - observatii

Corectitudinea algoritmului

Lema 2

Fie *T un arbore binar complet reprezentând o codificare prefix optima peste un* alfabet *V, unde frecventa f[c] este definita pentru fiecare caracter c din V. Consideram doua* caractere *x si y oarecare care apar ca noduri terminale frati în T, si fie z tatal lor. Atunci,* considerând *z ca un caracter având frecventa f[z] = f[x] + f[y], arborele T' = T - {x,y}* reprezinta o codificare prefix optima pentru alfabetul *V' = V - {x,y} U {z} .*

Dem - Ideea demonstratiei = metoda reducerii la absurd.

Daca *T' reprezinta o codificare prefix care nu este optima pentru alfabetul V', atunci exista* un arbore *T" ale carui frunze sunt caractere în V' astfel încât costul arborelui T" < costul arborelui T.*

Cum z este tratat ca un caracter în *V', el va apare ca frunza în T".*

Daca adaugam x si y ca fiind fiii lui z în T", atunci vom obtine o codificare prefix pentru *V având costul arborelui T"+f[x]+f[y] < costul arborelui T, ceea ce* intra în contradictie cu optimalitatea lui *T. Deci T' trebuie sa fie optim pentru alfabetul V'.*

Teorema: Procedura Huffman realizeaza o codificare prefix optima.

Aplicatii ale ABS la codificare binara

Cerinte:

1) Constructia unui *cod* binar cu proprietatea prefix peste un alfabet dat *V*.

2) Codificarea cuvintelor peste *V* adică: dat fiind orice cuvânt peste *V*, $a_1a_2 \ldots a_n$ *din* V^*, să producem codul asociat lui, din codurile literelor. Prin formula

$$c(a_1a_2 \ldots a_n) = c(a_1)c(a_2) \ldots c(a_n).$$

3) Decodificarea şirurilor de biţi, adică: dat fiind orice cuvânt *u* din $\{0, 1\}^*$, să decidem dacă există sau nu un cuvânt *x din* V^* al cărui cod este *u* şi, în cazul afirmativ, să spunem care este acest *x* (unic) cu proprietatea $c(x) = u$.

Aplicatii ale ABS la codificare binara

Cerinte:

1) **constructia unui cod binar cu proprietatea prefix -> constructia unui ABS cu literele din *V* in frunze.**

   - generare de coduri de lungime fixa -> constructia AB sa aiba toate frunzele pe acelasi nivel -> vor trebui adaugate frunze care un vor corespunde niciunui carácter din V.

2) Codificarea cuvintelor peste -> pentru fiecare frunza, drumul unic de la radacina la ea.

- deziderat: lungimea mesajelor codificate sa fie minima (pentru a scurta timpul de transmitere) -> metoda: lungimea codului unui caracter sa fie invers proportionala cu frecventa ei de aparitie in text.

3) Decodificarea sirurilor de biti -> drumuri succesive de la radacina pana la frunze.

Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Aplicatii ale ABS la codificare binara

1) ***Construcţia codului***: aplicarea algoritmului lui Huffman pentru construirea unui abs cu lungime externă ponderată minimă pentru mulţimea de frunze ponderate

$$E = \{(a, w(a)) | \ a \text{ din } V\}.$$

2) ***Codificarea***. Fiecărui caracter *a din V* i se asociază codul constând din şirul de etichete al arcelor ce compun drumul de la rădăcină la frunza asociată caracterului ***a***.

3) **Decodificarea unui şir de biţi** revine la parcurgeri repetate ale câte unui drum în arborele lui Huffman, începând de la rădăcină, conform convenţiei: în cazul în care caracterul (bitul) curent este *0*, parcurgerea continuă pe fiul stâng, dacă este *1*, parcurgerea continuă pe fiul drept. De fiecare dată când ajungem într-o frunză, am terminat de decodificat un caracter şi reluăm parcurgerea de la rădăcină pentru restul şirului de biţi. Dacă o asemenea parcurgere se termină într-un nod interior, atunci şirul *u din {0, 1}** de decodificat nu este valid, adică nu există nici un cuvânt *x din V*,* astfel încât *c(x)* = *u*.

Aplicatii ale ABS la codificare binara

Exercitiu

Se dau frunzele cu ponderi: (A, 40%), (R, 9%), (E, 22%), (T, 10%), (N, 4%), (C, 15%).

a) Sa se construiasca arborele Huffman ce contine aceste frunze, cu conventiile:

- la legarea a doi arbori, cel cu pondere mai mica se leaga la stanga
- arcele de tip fiu-stang se eticheteaza cu 0, iar cele fiu-drept cu 1.

b) Se considera cuvantul *CATREN* peste alfabetul dat. Care din urmatoarele siruri de biti reprezinta codificarea lui binara ce foloseste codul Huffman asociat arborelui construit?

A. 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 0 1 0
B. 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0
C. 1 1 0 0 1 0 0 1 0 1 1 1 1 0 1 0 1 0
D. 1 1 0 0 1 0 0 1 0 1 1 1 1 1 1 0 1 0
E. 1 1 0 0 1 0 0 1 0 0 1 1 1 0 1 0 1 0

c) Sa se decodifice, daca e posibil, urmatoarele secvente binare.

A. 1 0 1 0 1 1 1 1 1 0 1 0 0 0 1 0 1 1
B. 1 0 0 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1
C. 1 1 0 1 1 1 1 0 0 0 1 0 0 1 1 1
D. 1 1 0 0 1 0 0 0 1 0 1 0 1 1 1
E. 1 0 0 1 0 1 1 0 1 1 0 1 0 0 1 0 0 1 1

Aplicatii ale ABS la codificare binara

Lucrare de licenta – Absolvent Stancu Mihai

Titlu: **Packere. Metode de ofuscare a executabilelor.**

Coordonator: **Lector Univ. Dr. Paul Irofti.**

Packer: **tool care transformă un executabil într-un alt executabil ce prezintă aceeași funcționalitate dar este mult mai dificil de analizat prin tehnici de reverse engineering**

Ofuscarea payload-ului: **Conținutul executabilului original este modificat prin algoritmi de compresie și criptare.**
- Algoritm folosit pentru criptare si decriptare: HUFFMAN

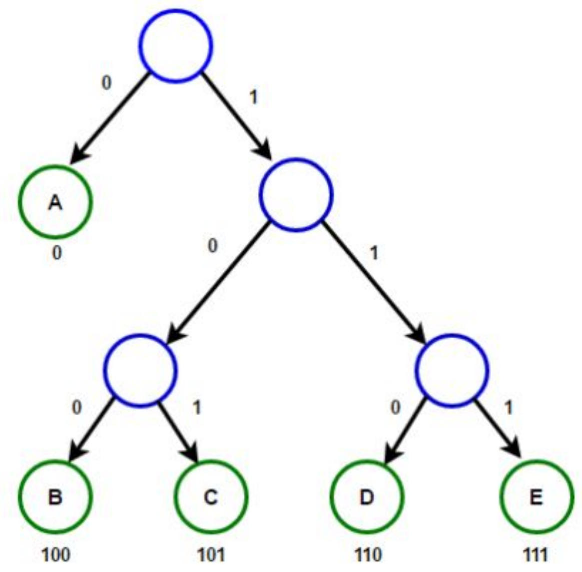
Facultatea de Matematica si Informatica
Universitatea din Bucuresti

Aplicatii ale ABS la codificare binara

Algoritmul Huffman aplicat:

Algorithm 1 Huffman Coding

```
function GENERATETREE( map < char, int > Caracters)
    priority_queue HuffmanQueue;
    for each c in Caracters do
        HuffmanQueue.push(c);
    end for
    while HuffmanQueue.size() != 1 do
        leftTree = HuffmanQueue.top();
        HuffmanQueue.pop();
        rightTree = HuffmanQueue.top();
        HuffmanQueue.pop();
        sum_frequency = leftTree.count + rightTree.count;
        new_node = HuffmanNode(sum_frequency);
        new_node.left = leftTree;
        new_node.right = rightTree;
        HuffmanQueue.push(new_node)
    end while
    root = HuffmanQueue.top();
    BuildCodes(root);
end function
```



Exemplu comprimare:

$$payload = \underbrace{A}_{0}\underbrace{C}_{101}\underbrace{E}_{111}\underbrace{Padding}_{0}$$

Exemplu decomprimare:

$$payload = \underbrace{0}_{\text{A}}\underbrace{101}_{\text{C}}\underbrace{111}_{\text{E}}\underbrace{0}_{\text{Padding}}$$

Aplicatii ale ABS la codificare binara

Eficienta compresiei

1) S-au considerat initial 6 caractere: a,r,u,x,m,h : 6 octeti initial in memorie.

2) Codurile Huffman obtinute: a = 010, r = 11000, u = 00111, x = 10010, m = 0111, h = 1010

3) S-a definit payload = c(aruxmh) si I se adauga 6 biti de 0 pentru a avea octeti completi

4) Payload comprimat, rescris in hexa:
payload = 01011000 00111100 10011110 10000000 = 58 3C 9E 80

5) Reducerea celor 6 octeti initiali la 4.

6) In medie, rata de compresie – 25%, iar viteza algoritmului de compresie – 3 MB/s.

Arbori binari stricti cu ponderi

Aplicatie a arborilor Huffman la interclasarea optimala a mai multor siruri

Interclasarea a două şiruri ordonate.

Se dau două şiruri ordonate crescător *A[1..dimA]* şi *B[1..dimB]*. Ne punem problema să construim şirul *C[1..dimA + dimB],* ordonat crescător, ce conţine toate elementele lui *A* şi *B*.

A[1..dimA] şi *B[1..dimB]* s.n **surse** ale operatiei de interclasare

C[1..dimA + dimB] s.n. **destinatie**

Arbori binari stricti cu ponderi

Interclasarea a două şiruri ordonate

```
k = 0;
i = 0;
j = 0;

while(i<dimA && j<dimB)
        if (a[i]<b[j])
            c[k++] = a[i++];
        else
            c[k++] = b[j++];

while (i<dimA)
    c[k++] = a[i++];

while (j<dimB)
    c[k++] = b[j++];
```

C:\Users\Ank\Desktop\inter

```
3
10 20 30
4
20 30 40 50
10 20 20 30 30 40 50
```

Interclasarea a două şiruri ordonate – Complexitate timp

Obs: - nr de comparatii

Completarea tuturor locatiilor din C -> cel mult **dimA + dimB - 1** comparaţii (ultima componentă se mută în C fără a mai fi comparată)

Cazul cel mai favorabil, cu număr **minim** de comparaţii

$$\boldsymbol{C_{min} = min(dimA, dimB)},$$

caz care se atinge când vectorul sursă de dimensiune mai mică are toate componentele mai mici decât cele din a doua sursă

Numărul de *mutări*, este constant şi egal cu $M = dimA + dimB$.

Arbori binari stricti cu ponderi

Interclasarea a două şiruri ordonate – Complexitate spatiu

***-spaţiu* în plus**, *C[1..dimA + dimB]*, egal ca dimensiune cu spaţiul necesar datelor de intrare.

(comparat cu alţi algoritmi de sortare, spaţiul utilizat este dublu)

Arbori binari stricti cu ponderi

Interclasarea (optimala) a mai mult de doua siruri

Se dau n $(n>2)$ şiruri ordonate crescător, S_1, S_2,…, S_n , cu lungimile respective l_1, l_2, …, l_n.

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;

iar dacă rezultatul se interclasează cu S_k, adică facem operaţia $(S_i\&S_j)\&S_k$ costul total va fi $2*(l_i+l_j)+l_k$.

Arbori binari stricti cu ponderi

Interclasarea (optimala) a mai mult de doua siruri

S. n. *strategie de interclasare* pentru sirurile S_1, S_2,…, S_n , **ordinea** în care le interclasăm două câte două.

Exemplu: variatia costului cu strategia de interclasare

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

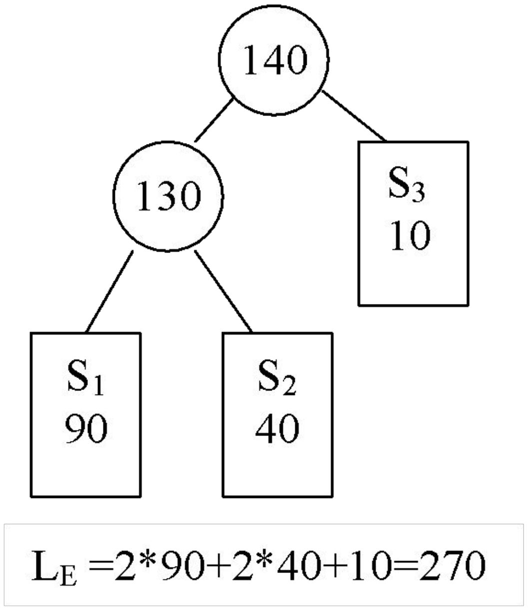
(2) $(S_2\&S_3)\&S_1$ are costul total
$(40+10)+((40+10)+90)=(40+10)*2+90=190.$

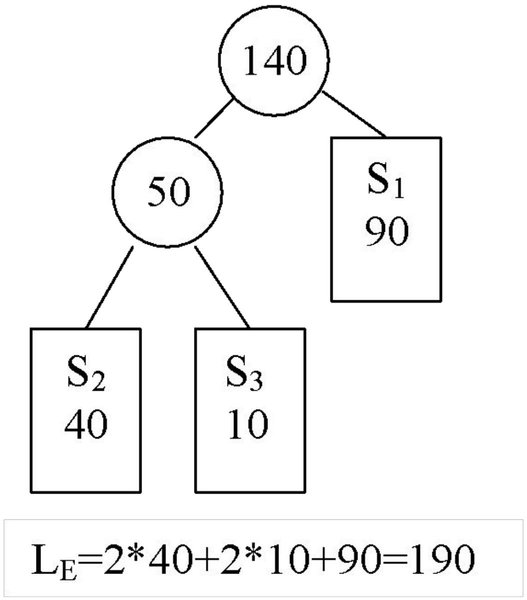
Deci, a doua strategie este mult mai performantă decât prima.

Arbori binari stricti cu ponderi

Interclasarea (optimala) a mai mult de doua siruri

fiecărei strategii îi asociem un arbore binar strict cu ponderi...





$(S_1\&S_2)\&S_3$ $(S_2\&S_3)\&S_1$

costul total al strategiei = lungimea externă ponderată a arborelui asociat.

Arbori binari stricti cu ponderi

Interclasarea (optimala) a mai mult de doua siruri

Lungimea externă ponderată a arborelui asociat unei strategii va fi exact **costul total în număr de mutări** al respectivei strategii.

Acest cost se minimizează dacă arborele asociat strategiei este arborele Huffman.

Curs 10

Algoritmi de sortare pentru multimi statice (vectori)

Clasa algoritmilor de sortare bazati pe comparatii intre chei.

Sortarea ShellSort.

Sortarea rapida (QuickSort).

Sortarea prin interclasare (MergeSort).

Analiza eficientei algoritmilor - rezolvarea relatiilor de recurenta

Teorema Master