

Structura fizica a bazelor de date relationale

Ioana Ciuciu

oana@cs.ubbcluj.ro

<http://www.cs.ubbcluj.ro/~oana/>

Planificare

Saptamana	Curs	Seminar	Laborator
S1	1. Concepte fundamentale ale bazelor de date	1. Modelul Entitate-Relatie. Modelul relational	1. Modelarea unei BD in modelul ER si implementarea ei in SQL Server
S2	2. Modelare conceptuala		
S3	3. Modelul relational de organizare a bazelor de date	2. Limbajul SQL – definirea si actualizarea datelor	2. Interogari SQL
S4	4. Gestiunea bazelor de date relationale cu limbajul SQL		
S5-6	5-6. Dependente functionale, forme normale	3. Limbajul SQL – regasirea datelor	3. Interogari SQL avansate
S7	7. Interogarea bazelor de date relationale cu operatori din algebra relationala	4. Proceduri stocate	
S8	8. Structura fizica a bazelor de date relationale		
S9-10-11	9-10-11. Indeksi. Arbori B. Fisiere cu acces direct	5. View-uri. Functii definite de utilizator. Executie dinamica	4. Proceduri stocate. View. Trigger
S12	12. Evaluarea interogarilor in bazele de date relationale	6. Formele normale ale unei relatii. Indeksi	
S13	13. Extensii ale modelului relational si baze de date NoSQL	7. Probleme	Examen practic
S14	14. Aplicatii		

Planul cursului

▶ Curs 8

- ▶ Structura fisierelor
 - ▶ Probleme ce trebuie rezolvate
- ▶ Tipuri de interogari
- ▶ Cautarea secventiala
- ▶ Cautarea in colectii de date ordonate dupa valorile unei chei

▶ Curs 9-10

- ▶ 2,3-arbore
- ▶ B-arbore
- ▶ Organizarea directa
- ▶ Index pentru un atribut oarecare
- ▶ Gestiunea indexurilor



Structura fișierelor

Tipuri de memorie:

- **internă:** nepersistentă
- **externă:**
 - persistentă
 - acces secvențial și direct (disc magnetic, memorie adresabilă), numai acces secvențial (benzi magnetice)

Caracteristici ale memoriei externe:

- **capacitatea de stocare**
- **timp de acces:** "timpul ce se consumă din momentul emiterii unei cereri de citire sau scriere și până în momentul când începe efectiv transferul de date"
- **rata de transfer:** "cantitatea de informație ce se transferă într-o secundă"
- **dimensiunea unui bloc** (înregistrare fizică). Blocul este unitatea de transfer dintre memoria internă și cea externă.

Bază de date:

- **nivel logic** (în aplicații): conform unui model de organizare
- **nivel fizic:** colecție de fișiere
- un fișier este
 - o colecție de înregistrări logice (de exemplu, pot corespunde la liniile unui tabel din modelul relațional dacă tabelul se memorează într-un fișier)
 - o colecție de blocuri pe suport (succesiune de octeți)



Structura fișierelor

Probleme privind gestiunea fișierelor:

1. Cum se realizează ștergerea unei înregistrări?

- pe locul ocupat se mută înregistrări
- marcarea zonelor șterse și păstrarea unei liste a lor. La următoarea operație de adăugare se pot folosi înregistrări din această listă (dacă lista nu e vidă).

2. Gruparea înregistrărilor în blocuri

- schimbul dintre memoria internă și suport se face prin **blocuri**
- un bloc poate conține mai multe înregistrări (toate sunt transferate simultan)
- dacă o înregistrare se memorează în mai multe blocuri, atunci poate crește timpul de acces

3. Memorarea înregistrărilor de lungime variabilă (provenite din memorarea coloanelor de lungime variabilă)

- fiecare înregistrare se memorează pe spațiul de memorie necesar (există multe metode de regăsire a valorilor coloanelor) - apar probleme la acces și modificare
- se rezervă o zonă de memorie maximă pentru a se putea memora orice înregistrare
- formatarea spațiului în zone de lungime fixă și memorarea unei înregistrări într-o mulțime de astfel de zone (consecutive, înlănțuite)
- formatarea spațiului în două zone:
 - o zonă secundară unde se memorează valorile coloanelor de lungime variabilă
 - o zonă principală care conține valorile coloanelor de lungime fixă și adresele valorilor (eventual și lungimea valorilor) pentru coloanele de lungime variabilă. Fiecare înregistrare din zona principală va avea o lungime fixă.



Structura fisierelor/Tipuri de interogari

4. Gestiunea zonelor tampon (zonă din memoria internă unde se poate memora un bloc)

- dacă o înregistrare necesară se află în memoria internă (într-o zonă tampon), atunci se micșorează timpul de acces (nu mai trebuie transferată de pe suport)
- modificarea unei înregistrări se face în memoria internă. Modificările făcute se trec în baza de date la anumite momente.
- sunt necesari algoritmi care caută o zonă tampon în care se va citi un nou bloc. Dacă zona tampon care se înlocuiește conține înregistrări modificate, atunci aceasta trebuie rescrisă pe suport înainte de a se citi alte date. **Probleme ce pot să apară.**

Tipuri de interogări într-un tabel memorat sau tabel temporar: se cer înregistrările pentru care o condiție are valoarea logică de **true**

- **Condiție elementară** cu un atribut de căutare (atribut cheie sau atribut oarecare)
 - **atribut** *operator_relațional* valoare
 - **expresie** *operator_relațional* valoare
 - **expresie** $\in [v_1, v_2]$
 - **expresie** $\in \{v_1, v_2, \dots, v_n\}$
 - **expresie** *LIKE* șablon
 - se definește o distanță **d** peste valorile atributului și se cer înregistrări pentru care valoarea atributului:
 - $d(\text{atribut}, v_0) \leq d_0$
 - cea mai apropiată valoare de v_0 ,
 - cele mai apropiate **n** valori de v_0
- **condiție compusă** cu operatorii logici: **not**, **and**, **or**



Tipuri de interogari

Pentru fiecare tip de interogare trebuie să existe cel puțin un algoritm care determină răspunsul.

Parametri importanți la determinarea răspunsului unei interogări (care depind de structura fizică):

- **timpul (mediu) de răspuns** pentru o mulțime de interogări
- **spațiul suplimentar de memorie folosit** la memorarea sursei de date

Accesul la înregistrări

Pentru a regăsi o înregistrare trebuie ca ea să fie localizată pe suport. Această localizare (accesul la înregistrare) se poate face:

- **secvențial**: localizarea se face prin parcurgerea înregistrărilor care o preced pe suport
- **direct**: localizarea se face prin intermediul unei adrese calculate (adresa calculată poate fi folosită pentru a preciza de unde se începe căutarea înregistrării)
- **indirect**: adresa înregistrării este memorată împreună cu alte date (în altă zonă de memorie, într-un index)

Organizarea fișierelor: stabilirea unei relații între înregistrările fișierului și poziția lor pe suport

- Fie $F = \{r_1, r_2, \dots, r_n\}$ colecția de înregistrări dintr-un fișier (privit la nivel logic, ca liniile unui tabel/relații). În acest fișier trebuie efectuate operațiile următoare: **adăugarea, ștergerea, modificarea valorilor, căutarea, parcurgerea (totală sau parțială)**.
- Există multe metode de organizare a fișierelor, care permit micșorarea timpului de răspuns pentru anumite tipuri de operații.



Cautarea secventiala

Determinarea răspunsului la diverse tipuri de interogări

1. Căutare secvențială pentru interogarea $C = K_0$, unde C este o cheie

Deoarece C este cheie, răspunsul poate fi:

- o singură înregistrare ("*căutare cu succes*")
- mulțimea vidă ("*căutare fără succes*")

După descrierea algoritmilor de căutare este necesară **complexitatea** lor (pentru a putea face o comparație între ei). Din această cauză se poate presupune că toată colecția (tot fișierul) este în memoria internă. Algoritmii se pot ușor modifica pentru a se citi înregistrările dintr-un fișier.

Fie: K_i = valoarea cheii **C** pentru înregistrarea r_i , deci $K_i = \Pi_C(r_i)$, $i=1, \dots, n$.

Algoritm care determină răspunsul la interogarea amintită: se compară K_0 cu $K[1]$, $K[2]$,



Cautarea secventiala

Pentru **complexitatea algoritmului** se poate lua **numărul de comparații efectuate între chei**.

Observație. Căutare secvențială după valoarea unui atribut cere consultarea întregului fișier.

Probleme asemănătoare:

- ordinea de memorare a înregistrărilor de lungime variabilă, pentru o căutare secvențială, dacă o înregistrare r_i , $i=1, \dots, n$, are **lungimea L_i și probabilitatea de a fi căutată p_i** .
- ordinea de testare a condițiilor C_i din condiția compusă **C_1 and C_2 and ... and C_n** , dacă C_i necesită un timp T_i pentru evaluare și are probabilitatea P_i de a fi adevărată.



Cautarea in colectii de date ordonate

2. Căutarea în colecții de date ordonate pentru interogarea $C = K_0$, unde C este o cheie

Deoarece C este cheie, toate valorile acesteia sunt distincte: $K_i \neq K_j, i \neq j$.

Presupunem că înregistrările sunt ordonate după valorile atributului C, deci:

$$K_1 < K_2 < \dots < K_n.$$

Pentru acest tip de colecție, cu algoritmul de **căutare secvențială** se micșorează numărul de comparații pentru "căutarea fără succes".

Algoritmul de căutare binară (logaritmică, bisectare)



Cautarea in colectii de date ordonate

Pentru a determina complexitatea algoritmului de căutare binară, se construiește un arbore binar cu n noduri interioare (în desen aceste noduri apar încercuite) și $(n+1)$ noduri terminale (încadrate):

- dacă $n=0$, atunci arborele este: $\boxed{0}$
- dacă $n \neq 0$, atunci nodul rădăcină este $\lceil n/2 \rceil$, subarborele stâng este arborele binar cu $\lceil n/2 \rceil - 1$ noduri, iar subarborele drept este arborele binar cu $\lfloor n/2 \rfloor$ noduri la care etichetele se măresc cu $\lceil n/2 \rceil$. Cei doi subarbori se construiesc după același procedeu.

Arborele binar care se construiește va avea nodurile interioare corespunzătoare valorilor i (poziția în vector) din comparațiile $K[i]=K_0$ care se fac în algoritm.

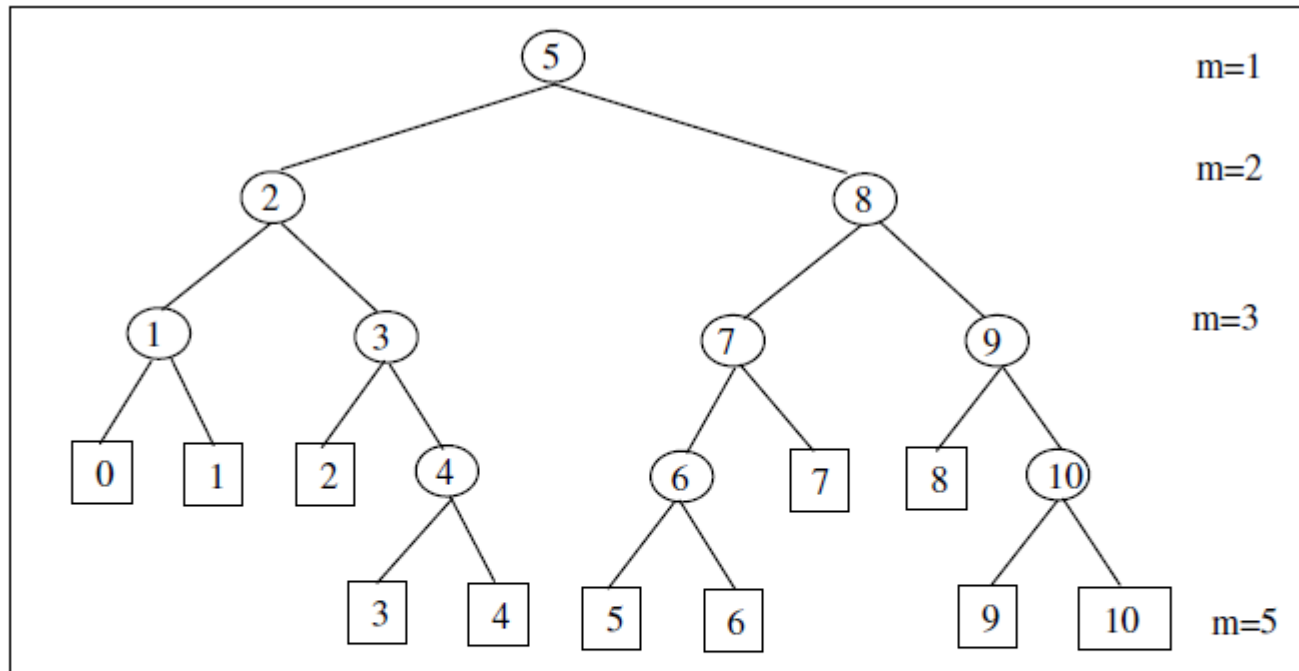
Algoritmul de căutare binară se poate termina:

- într-un nod interior (căutare cu succes)
- într-un nod terminal (căutare fără succes)

De exemplu, pentru $n=10$ se obține arborele din figura următoare.



Cautarea in colectii de date ordonate



Dacă datele nu încap pe $m-1$ nivele $\Rightarrow n > 2^0 + 2^1 + \dots + 2^{m-2} = 2^{m-1} - 1$, dar încap pe m nivele:

$$2^{m-1} \leq n < 2^m,$$

atunci arborele binar are nodurile interioare situate pe m nivele (pentru figură, numărul de nivele este 4), iar nodurile terminale pe cel mult două nivele. Din relația precedentă se obține:

$$m-1 \leq \log_2(n) < m \Rightarrow m-1 = \lfloor \log_2(n) \rfloor \Rightarrow m = \lfloor \log_2(n) \rfloor + 1$$

Cautarea in colectii de date ordonate

Observație. Algoritmul de căutare binară necesită:

- **minimum 1** comparație
- **maximum** $\left\lceil \log_2(n) \right\rceil + 1$ comparații
- un **număr mediu de comparații**: $O(\log n)$

Comparație între căutarea secvențială și căutarea în colecții ordonate.

Alți algoritmi de căutare în colecții ordonate:

- **căutarea Fibonacci**: indicii folosiți pentru comparații sunt calculați folosind numerele lui Fibonacci
- **căutarea prin interpolare**: indicii folosiți pentru comparații sunt calculați folosind o interpolare liniară



Cautarea in colectii de date ordonate

Memorarea colecției de date ca un arbore binar

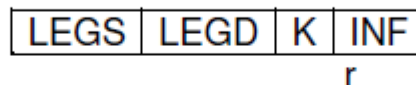
Algoritmul de căutare binară în colecții ordonate:

- foarte rapid
- greu de respectat condiția cerută: **înregistrările sunt ordonate** (mai ales pentru colecții dinamice)

Soluție: memorarea colecției ca un arbore binare:

- fiecare înregistrare se memorează într-un nod al arborelui binar
- **subarborele stâng** pentru un nod cu valoarea **v** pentru cheie va conține înregistrări cu valori mai mici decât **v** pentru cheie
- **subarborele drept** pentru un nod cu valoarea **v** pentru cheie va conține înregistrări cu valori mai mari decât **v** pentru cheie

Structura unui nod (valorile din înregistrare se memorează în K și INF):



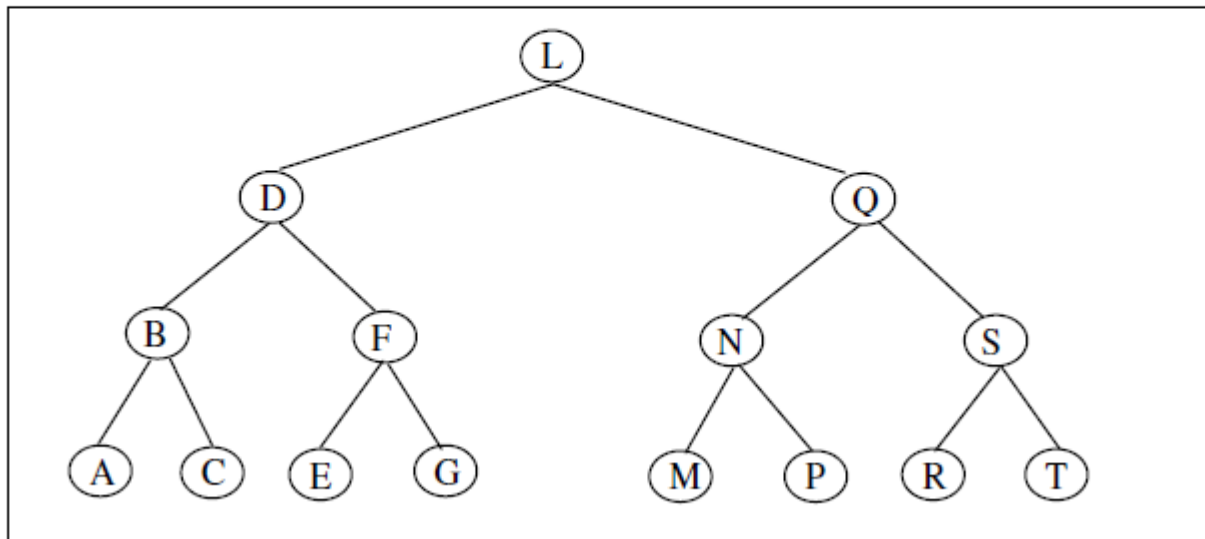
Operații necesare pentru acest arbore binar:

- **căutarea** unei înregistrări cu valoarea **K0** pentru cheie - **algoritm**
- **adăugarea** unei înregistrări - **pe exemple**
- **ștergerea** unei înregistrări
- **parcurgerea arborelui** (parțial: între două valori, total)



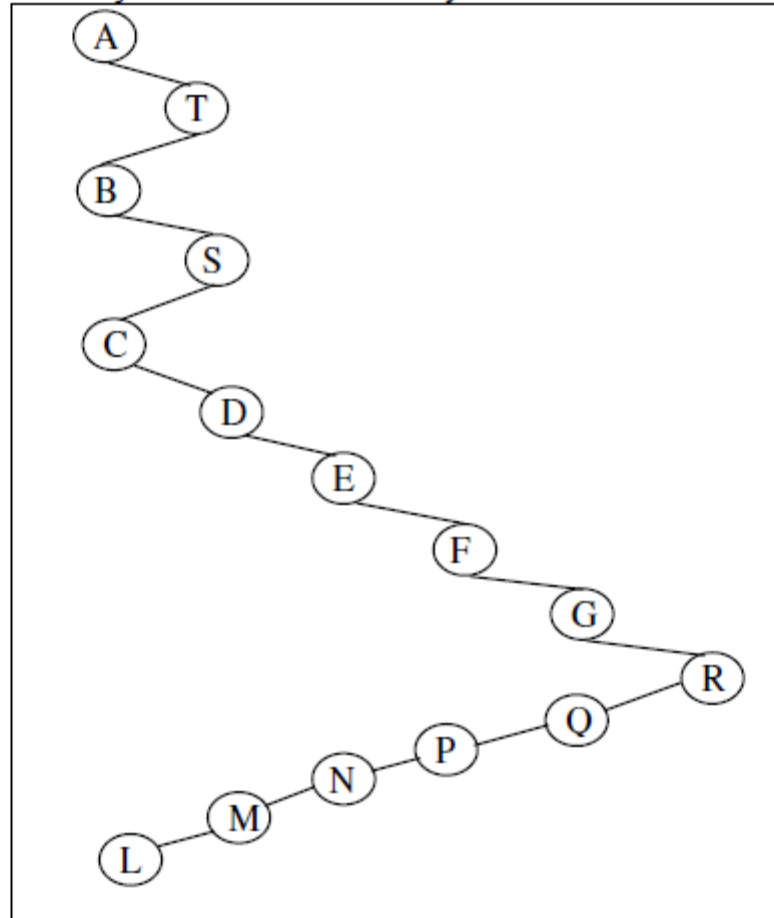
Cautarea in colectii de date ordonate

În continuare se vor da arborii binari obținuți **după adăugarea unor date** (se pun în evidență numai valorile pentru cheie). Considerăm că valorile pentru cheie sunt din mulțimea următoare: {L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C}, iar adăugarea în arbore se face în ordinea în care valorile se precizează în mulțime. Arborele obținut după ce s-au efectuat toate operațiile de adăugare este cel din figură.



Cautarea in colectii de date ordonate

Dacă valorile pentru cheie sunt aceleași, dar furnizate în altă ordine: {A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L}, atunci se obține arborele de mai jos.



Cautarea in colectii de date ordonate

Observații:

- Arborele din ultima figură este "degenerat", căutarea în el se reduce la o căutare secvențială.
- Forma arborelui, deci și timpul de căutare în acești arbori, depinde de ordinea în care datele s-au adăugat la colecție. Timpul minim se obține pentru un **arbore optimal** (în care terminalele sunt memorate pe cel mult două nivele consecutive), asemănător cu arborele construit pentru căutarea binară.

Definiții.

1. **Înălțimea unui arbore** este lungimea celui mai lung drum de la rădăcină la terminale (lungimea unui drum este dat de numărul de vârfuri).
2. Un **arbore binar este echilibrat** dacă pentru fiecare nod diferența dintre înălțimea celor doi subarbori este 0, 1 sau -1.

Observație. Dacă prin operații de adăugare sau ștergere un arbore echilibrat se transformă în unul care nu mai este echilibrat, atunci printr-un număr mic de transformări el poate să fie echilibrat.

Exemplu. La arborele A_3 din figura următoare se adaugă o valoare care modifică înălțimea (se mărește cu 1). Înălțimea subarborilor este trecută între paranteze, iar la nodurile A și B apare diferența dintre înălțimea celor doi subarbori. În nodul A se micșorează diferența dintre înălțimea celor doi subarbori, deci după adăugare arborele nu va mai fi echilibrat. Prin transformarea ce apare în partea dreaptă a figurii se face o echilibrare pentru arborele cu rădăcina în A. Această echilibrare (transformare) necesită puține atribuiri în zona de memorie unde se memorează arborele (LEGD pentru A, LEGS pentru B, legătura care face referire la A se schimbă la B). Alte modificări pot să intervină pe drumul din rădăcină la B.

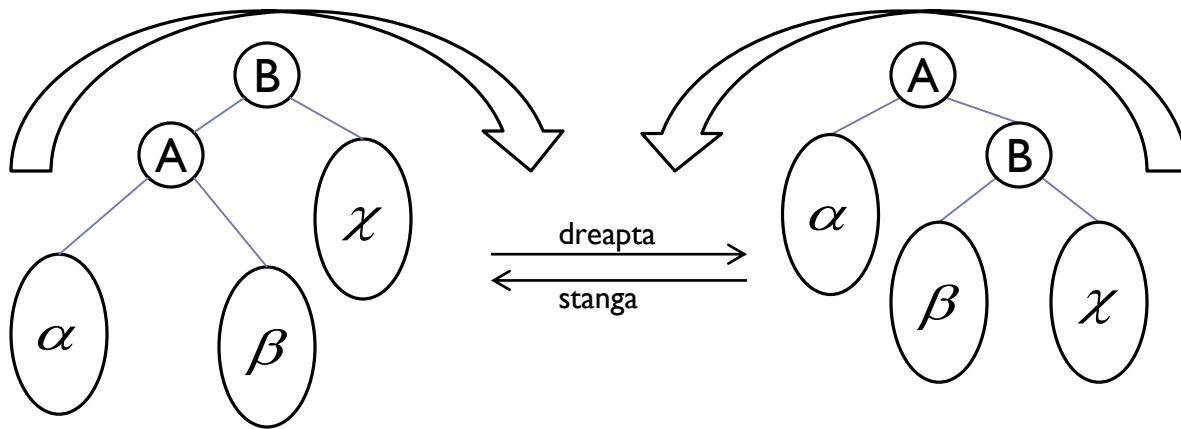


Cautarea in colectii de date ordonate

▶ Rotirea unui arbore

▶ Conditii:

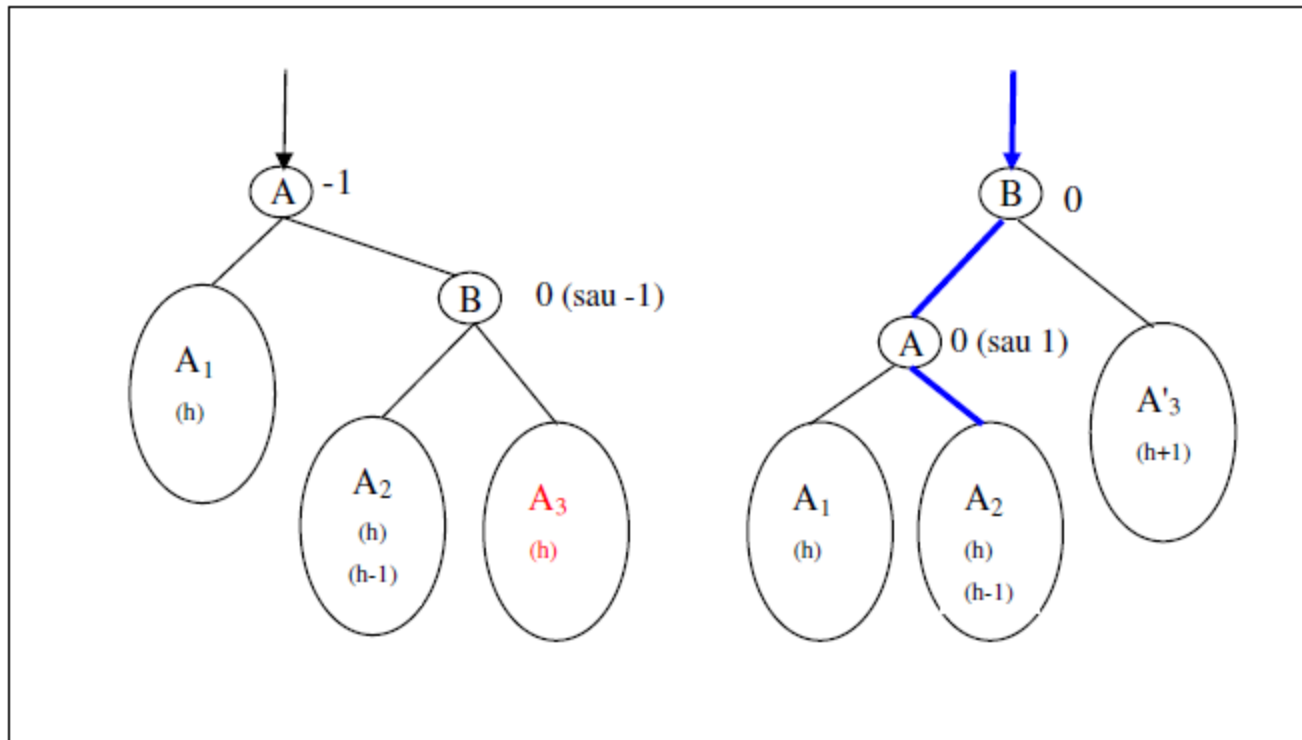
- ▶ Ordinea nodurilor terminale nu se schimba
- ▶ Sub-arborele drept e mai mare decat radacina, iar subarborele stang e mai mic decat radacina



Cautarea in colectii de date ordonate

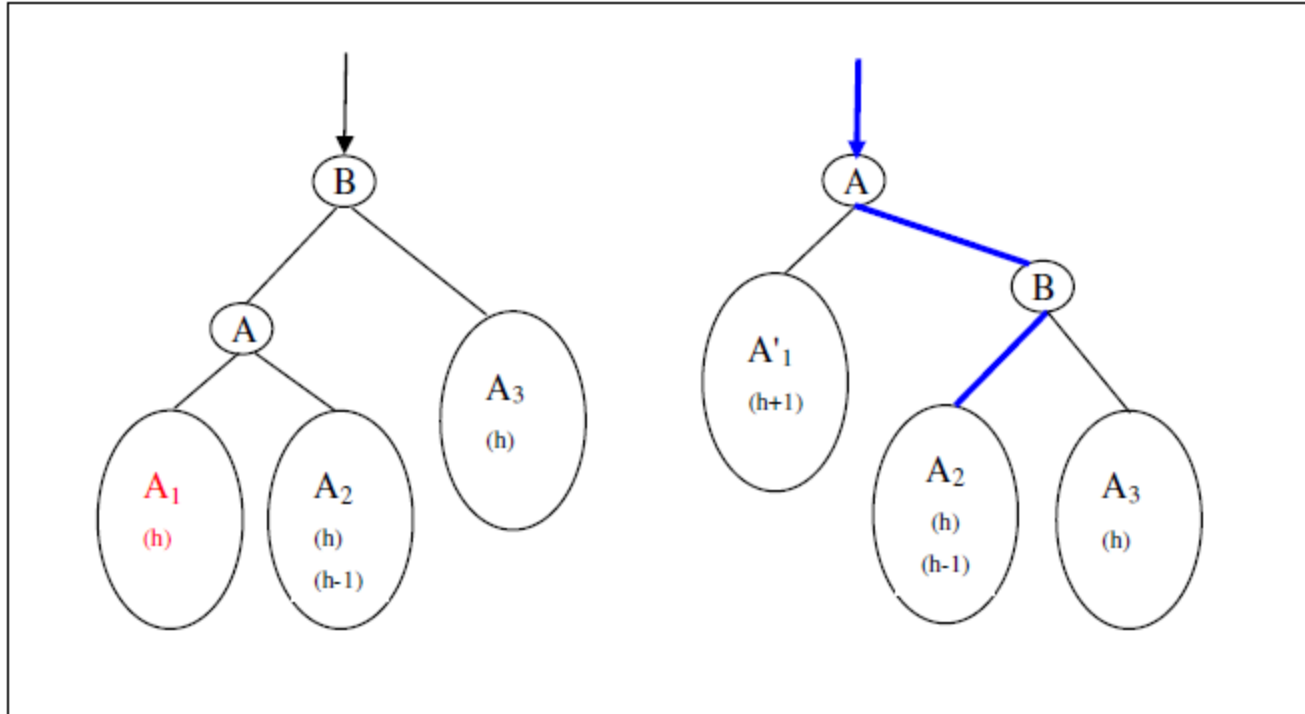
Precizare. In [Kn76] sunt enumerate toate cele **șase** transformări existente pentru echilibrare. S-a marcat cu culoarea roșie subarboarele la care se adaugă un nod și la care se mărește înălțimea.

Varianta 1:



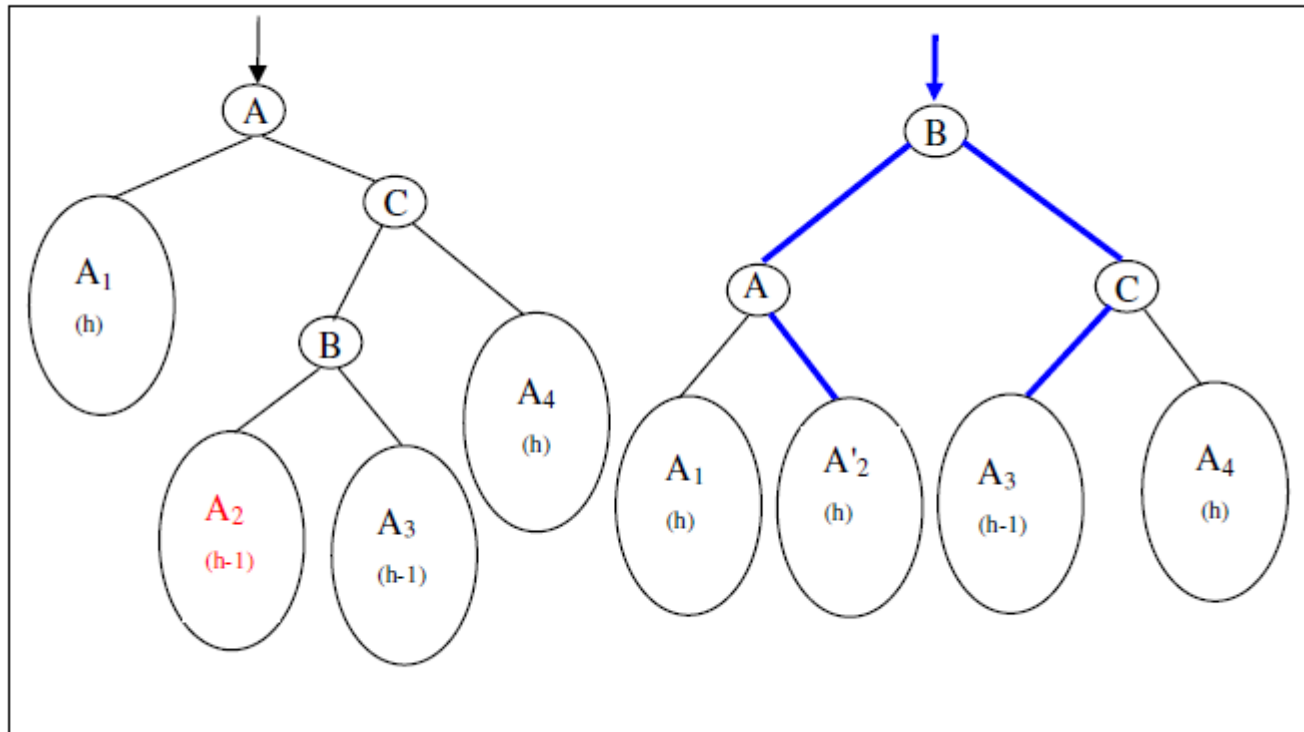
Cautarea in colectii de date ordonate

Varianta 2:



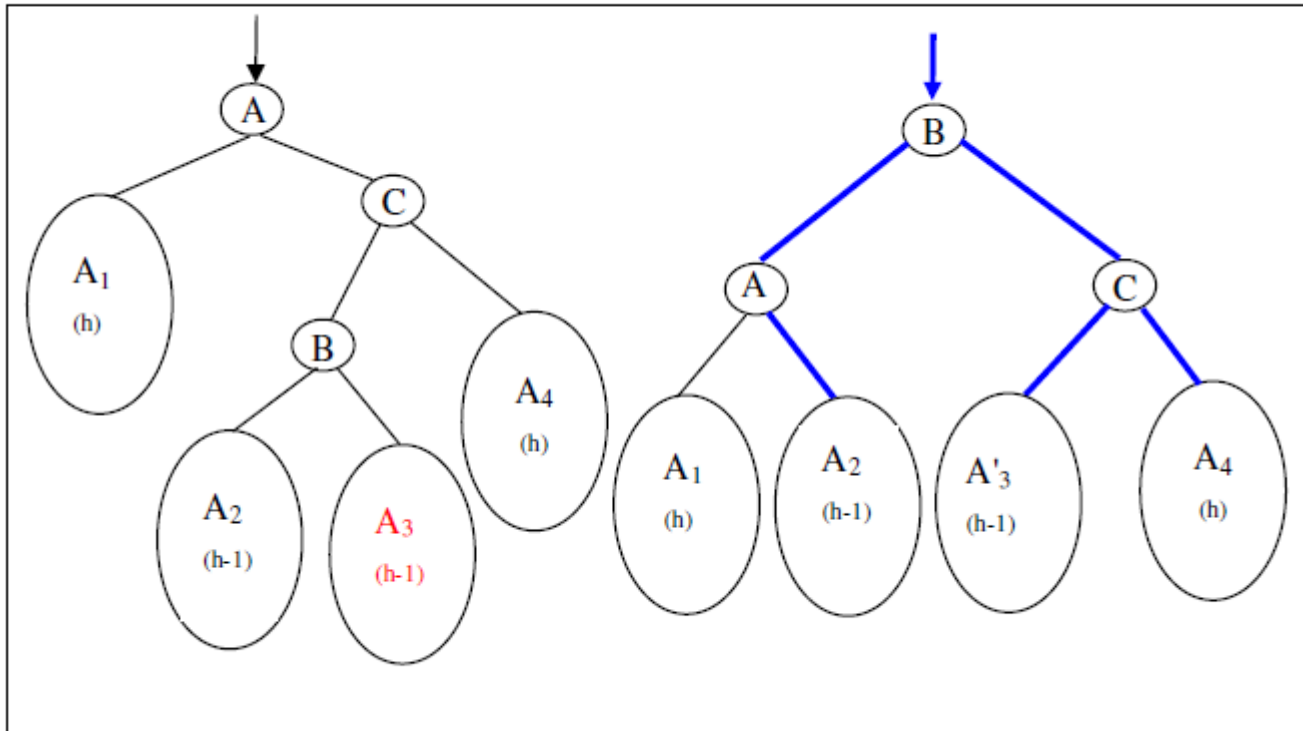
Cautarea in colectii de date ordonate

Varianta 3:



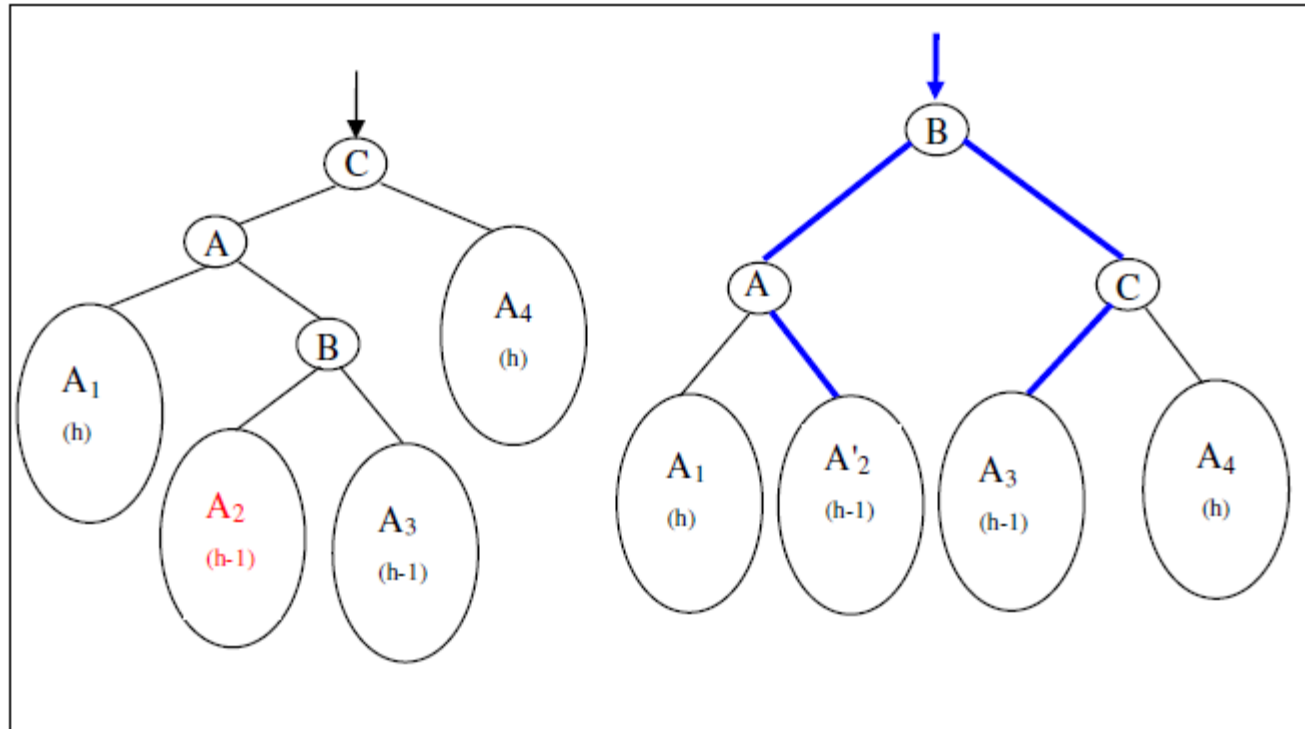
Cautarea in colectii de date ordonate

Varianta 4:



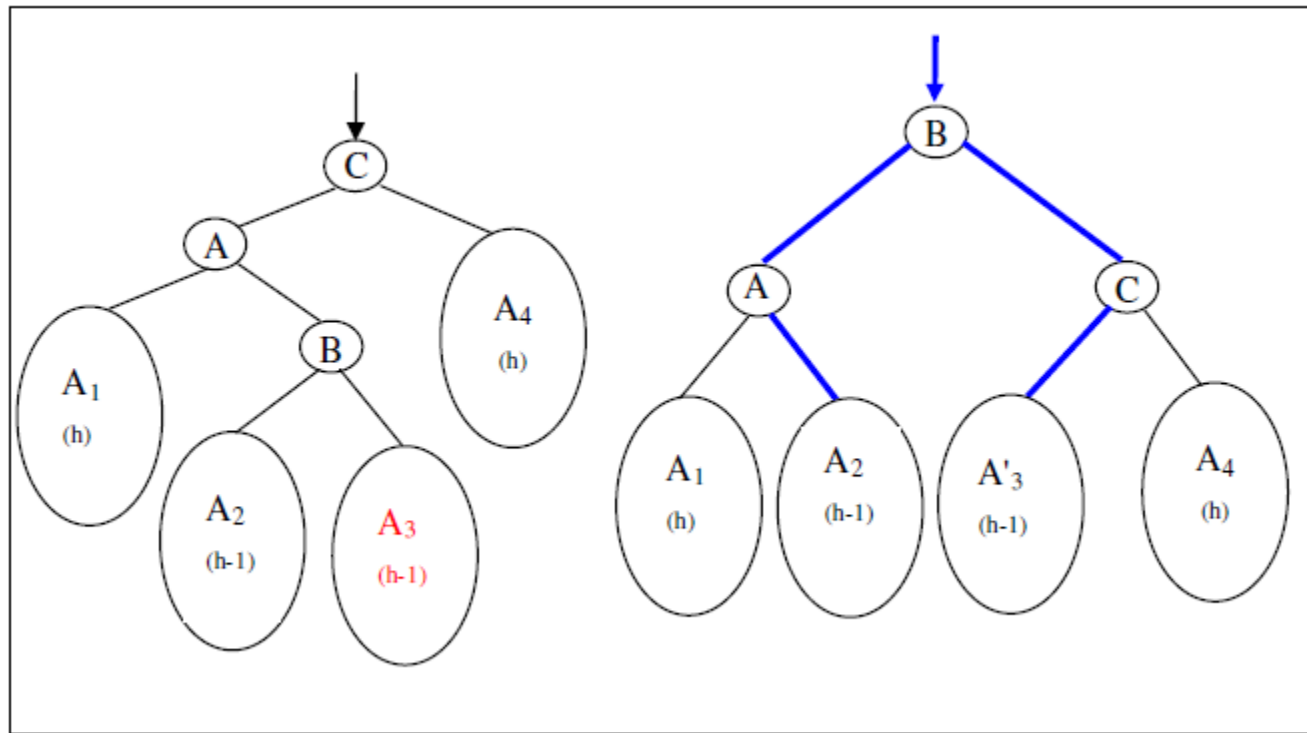
Cautarea in colectii de date ordonate

Varianta 5:



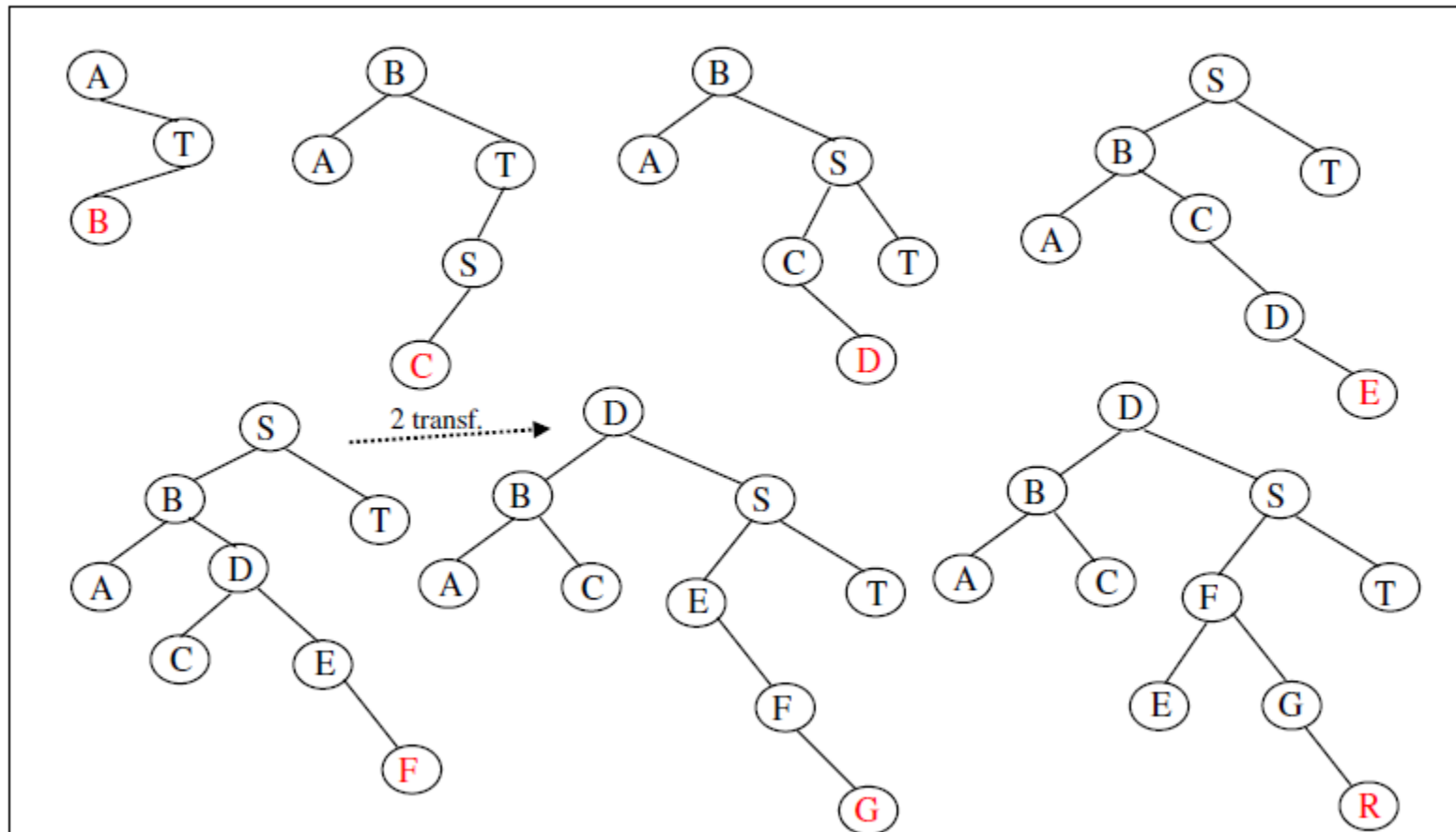
Cautarea in colectii de date ordonate

Varianta 6:

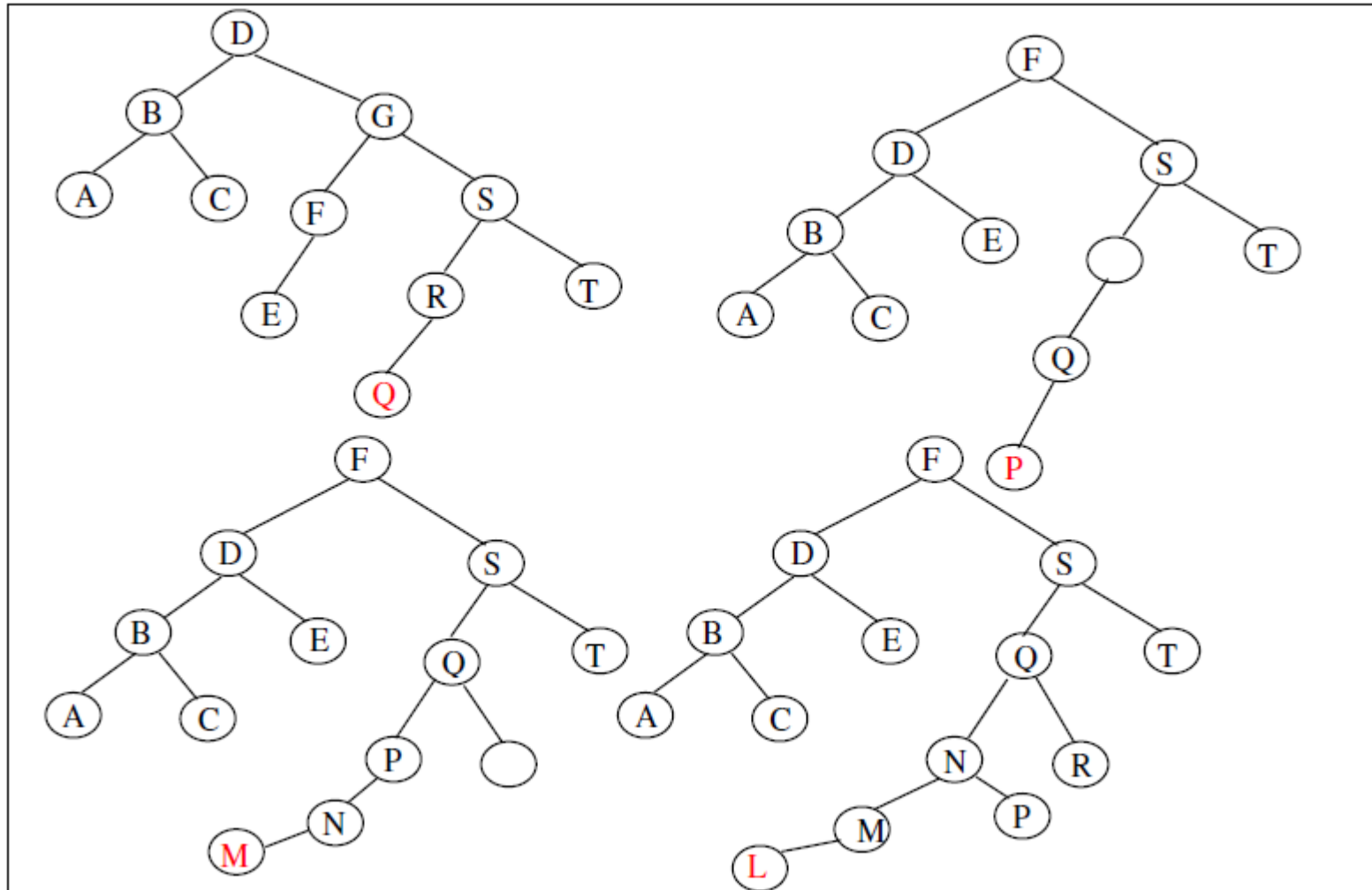


Cautarea in colectii de date ordonate

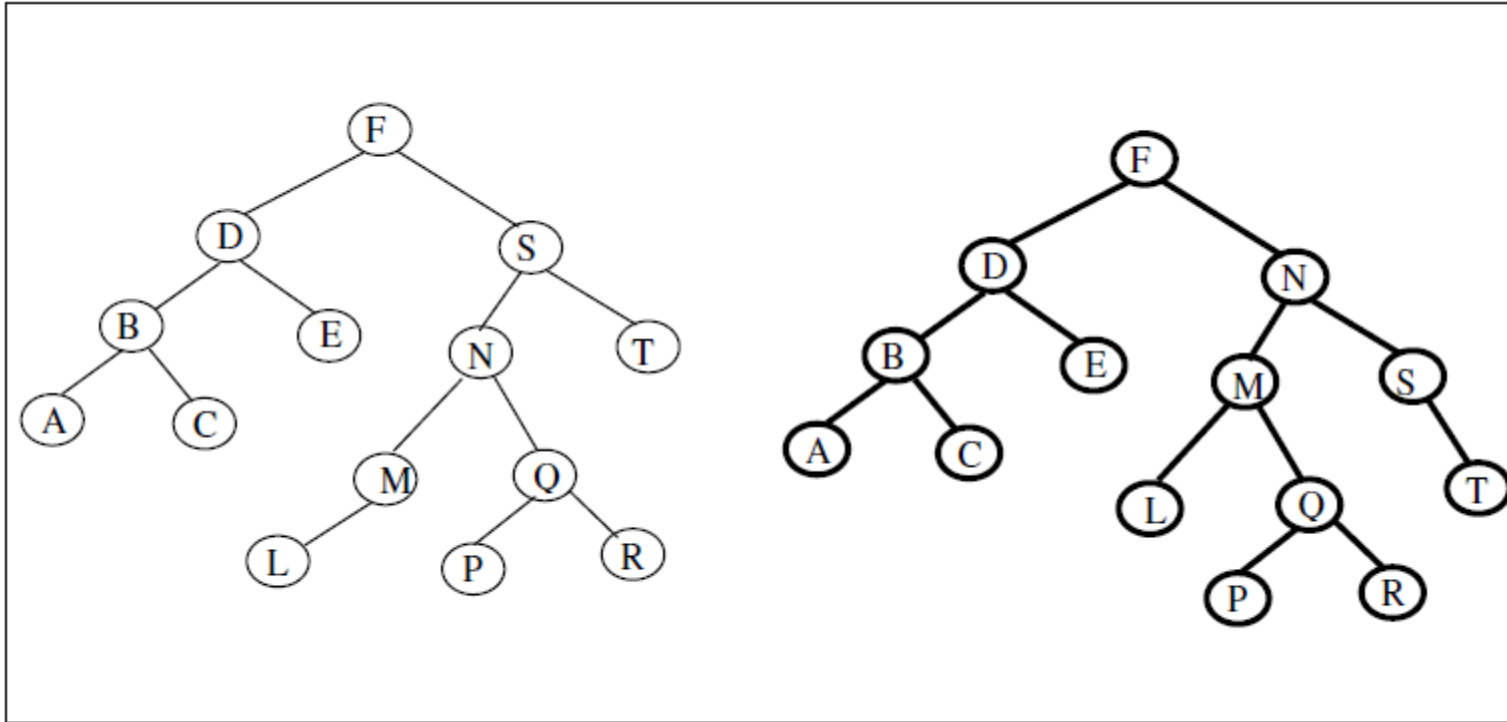
In continuare se vor arăta modurile de echilibrare pentru ultima colecție folosită:
{A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L}



Cautarea in colectii de date ordonate



Cautarea in colectii de date ordonate



Bibliografie

- ▶ [Si10] SILBERSCHATZ A., KORTZ H., SUDARSHAN S., *Database System Concepts*, McGraw-Hill, 2010, cap. 10, 11
- ▶ [Ra07] RAMAKRISHNAN, R., *Database Management Systems*. McGraw-Hill, 2007, cap. 7-10
- ▶ [Kn76] KNUTH, D.E., *Tratat de programare a calculatoarelor. Sortare si cautare*. Ed.Tehnica, Bucuresti 1976
- ▶ [Ga08] GARCIA-MOLINA, H., ULLMAN, J., WIDOM, J., *Database Systems: The Complete Book*, Pearson Prentice Hall, 2008], cap. 12-14.
- ▶ Leon Tambulea, UBB, curs de baze de date

