Structura fizica a bazelor de date relationale (partea 2)

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. Indecsi. Arbori B. Fisiere cu acces direct	5. View-uri. Functii definite de utilizator. Executie dinamica	4. Proceduri stocate. View. Trigger
		6. Formele normale ale unei relatii. Indecsi	
S12	12. Evaluarea interogarilor in bazele de date relationale		
S13	13. Extensii ale modelului relational si baze de date NoSQL	7. Probleme	Examen practic
S14	14. Aplicatii		

Planul cursului

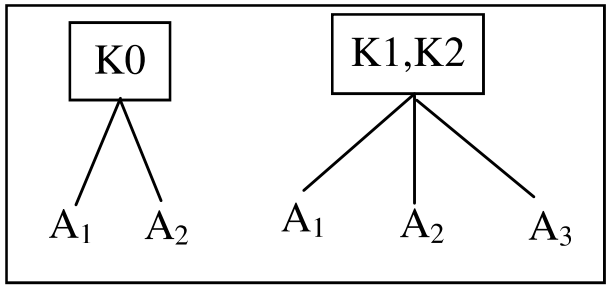
- Curs 9
  - 2,3-arbore
  - B-arbore

- Curs 10-11
  - Organizarea directa
  - Index pentru un atribut oarecare
  - Gestiunea indexurilor

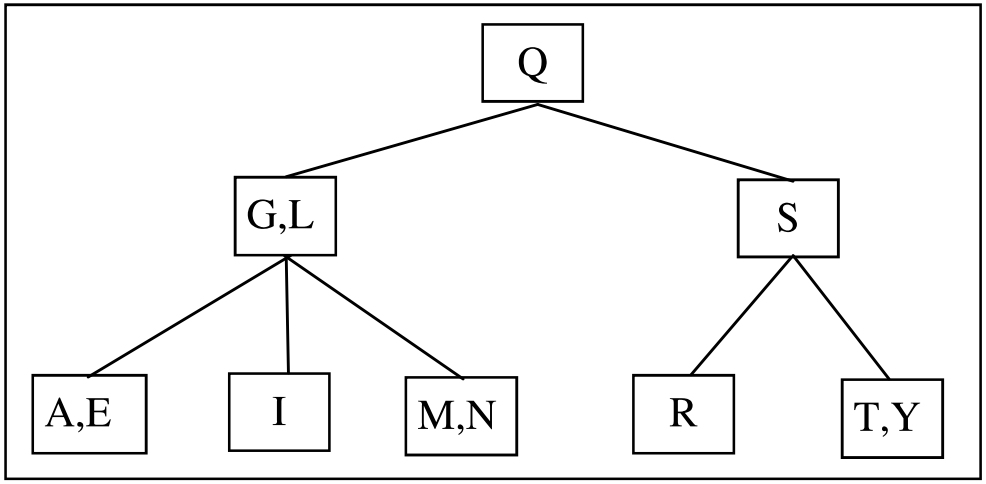
2,3-arbore

Memorează valorile unei chei (colecţie de valori distincte):

- **toate nodurile terminale sunt pe acelaşi nivel**
- **fiecare nod** conţine **una** sau **două** valori ale cheii
  - un nod neterminal cu o valoare K0 va avea doi subarbori, unul cu valori mai mici decât K0 şi unul cu valori mai mari decât K0
  - un nod neterminal cu două valori K1 şi K2, K1<K2, va avea trei subarbori, unul cu valori mai mici decât K1, unul cu valori între K1 şi K2 şi un subarbore cu valori mai mari decât K2



Exemplu (valorile pentru cheie sunt litere):

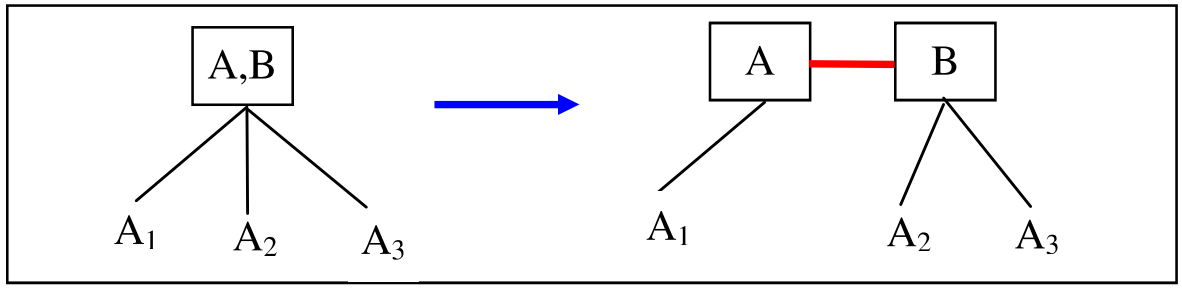


Memorarea unui 2,3-arbore

Un **index** construit ca un **2,3-arbore** se foloseşte pentru memorarea valorilor unei chei. In arbore se memorează **valorile cheii şi adresa înregistrării** (din fişier, din baza de date) care conţine valoarea respectivă pentru cheie.

Arborele se poate memora în două variante:

1. Să se **transforme într-un arbore binar**. Nodurile cu două valori se vor transforma (după cum se vede în figura următoare), iar cele cu o singură valoare vor rămâne nemodificate.



Structura unui nod va fi:

K	ADR	LEGS	LEGD	IND

unde:

- **K** memorează o valoare pentru cheie
- **ADR** este adresa înregistrării cu valoarea curentă pentru cheie (adresă în fişier)
- **LEGS**, **LEGD** sunt adresele celor doi subarbori (adresă în arbore)
- **IND** este un indicator, care precizează tipul legăturii spre dreapta (cele două valori posibile se observă din figura de mai sus)

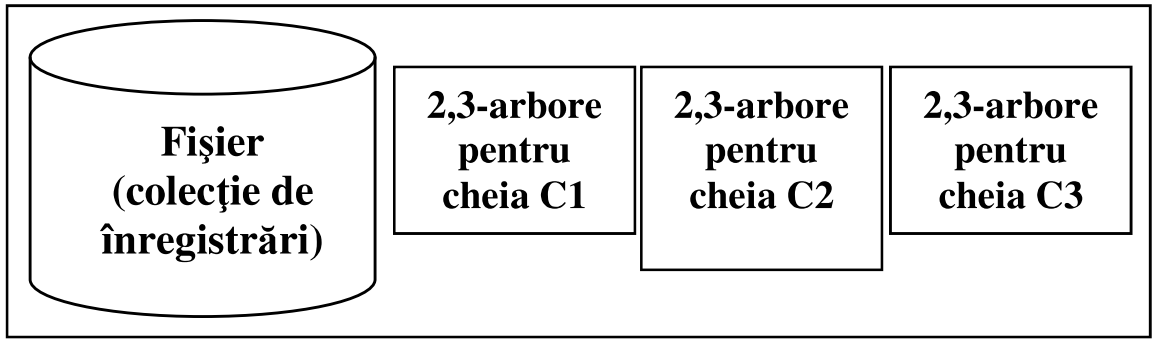
2. Zona de memorie alocată pentru un nod poate păstra două valori şi trei adrese de subarbore (maximum de valori posibile).

NV	K1	ADR1	K2	ADR2	LEG1	LEG2	LEG3

unde:

- **NV** este numărul de valori din nod (1 sau 2)
- **K1** şi **K2** memorează valori ale cheii
- **ADR1** şi **ADR2** sunt adrese de înregistrări (corespunzătoare lui K1 şi K2)
- **LEG1**, **LEG2** şi **LEG3** sunt adresele celor trei subarbori

Observaţie. La un **fişier** (relaţie dintr-o bază de date relaţională) se pot **asocia** mai mulţi 2,3-arbori (câte un 2,3-arbore pentru o cheie).

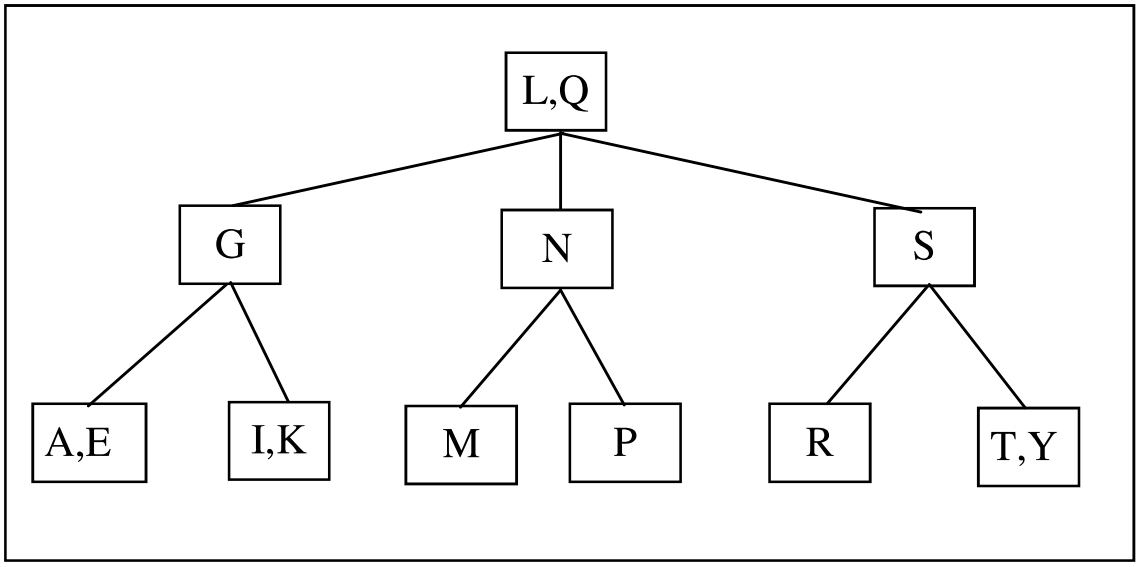


Operaţii în 2,3-arbore:

- **căutarea** unei înregistrări cu valoarea **K0** pentru cheie - **algoritm**
- **adăugarea** unei înregistrări - **descriere mod de efectuare**
- **ştergerea** unei înregistrări - **descriere mod de efectuare**
- **parcurgerea arborelui** (parţial: între două valori, total)

Adăugarea unei noi valori în 2,3-arbore:

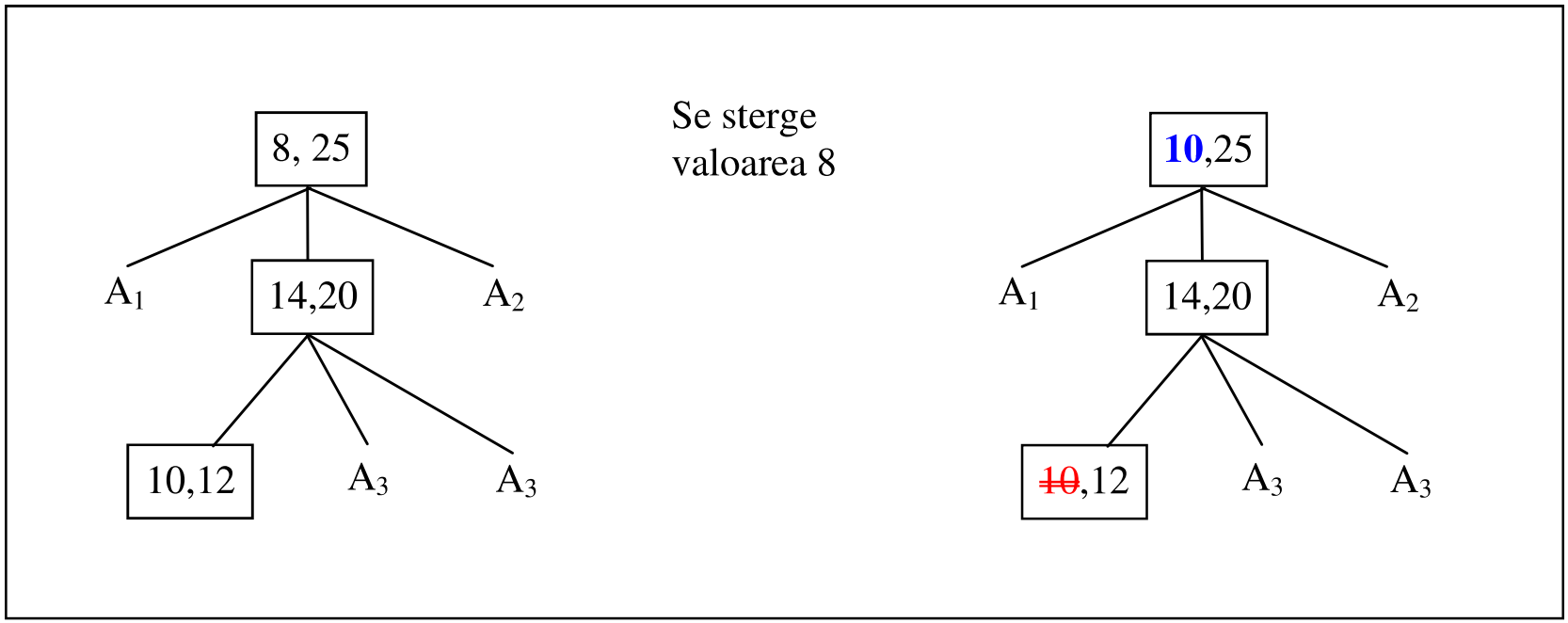
- Valorile din arbore trebuie să fie distincte, deci valoarea care se adaugă nu trebuie să se afle deja în arbore. Pentru a testa această situaţie, se face o căutare. Dacă operaţia de adăugare se poate efectua, atunci **ultima căutare are loc într-un nod terminal**.
- Dacă nodul terminal la care se ajunge are o singură valoare, atunci noua valoare se poate memora în acest nod. De exemplu, dacă în arborele din figura precedentă se adaugă valoarea **K**, atunci această valoare se memorează împreună cu valoarea **I**.
- Dacă nodul terminal la care se ajunge are două valori, atunci noua valoare se ***ataşează*** acestui nod, cele trei valori obţinute se sortează, nodul se divide în două noduri: un nod va conţine valoarea cea mai mică, al doilea nod va avea valoarea cea mai mare, iar valoarea din mijloc se ***ataşează*** nodului părinte. Situaţia din acest nod părinte se tratează asemănător. De exemplu, dacă în arborele precedent, după adăugarea valorii **K**, se mai adaugă valoarea **P**, atunci se obţine arborele din figura alăturată.



Stergerea unei valori **K0** din 2,3-arbore

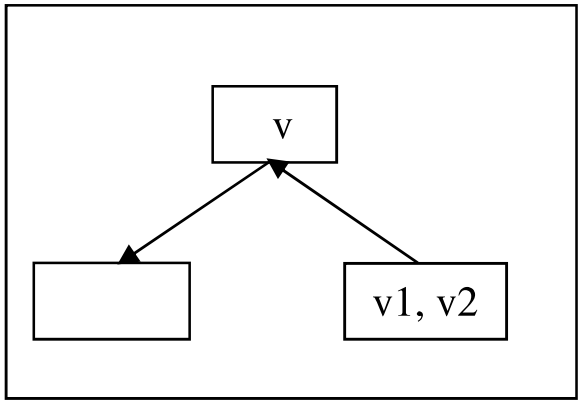
1. **Se caută** valoarea **K0** în arbore. Dacă apare într-un nod interior, atunci se ***schimbă*** această valoare cu una ***vecină* K1** aflată într-un nod terminal din arbore (nu există o altă valoare între K0 şi K1). Poziţia ocupată anterior de K1 (dintr-un nod terminal) se elimină.

   Exemplu.



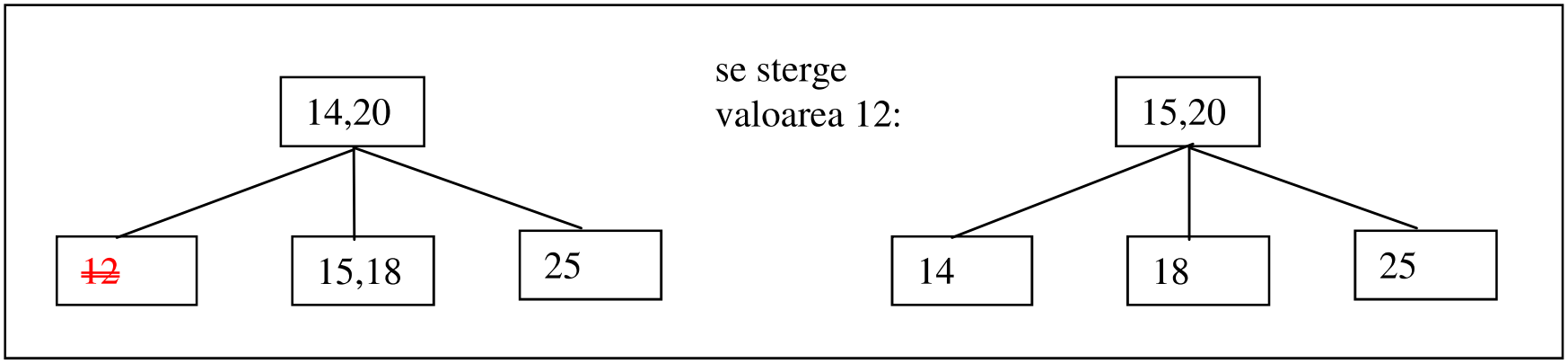
2. Acest pas se execută până apare una din situaţiile de la punctele **a** sau **b**.

a. Dacă nodul care conţine **valoarea (poziţia) v0 ce se elimină** este rădăcina sau un nod în care mai rămâne o valoare, atunci valoarea **v0** (poziţia ocupată de **v0**) se elimină şi algoritmul se termină.



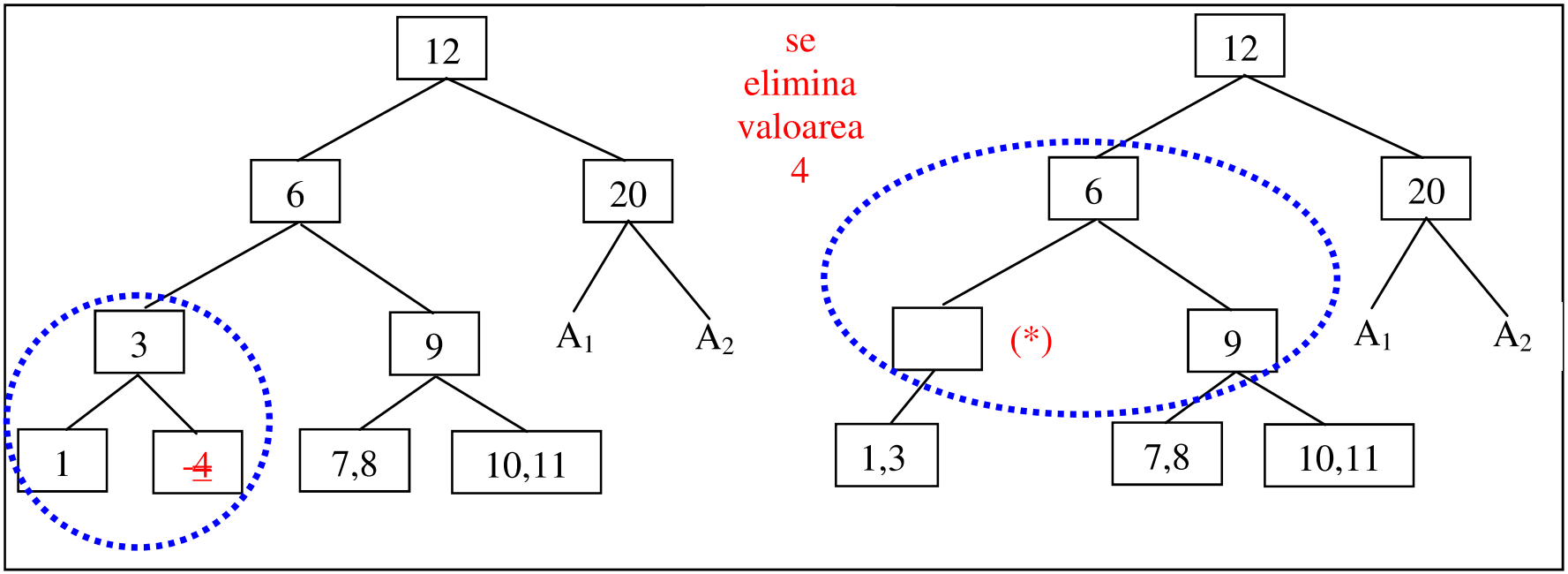
b. Dacă prin eliminare apare un nod fără valori, dar în unul din ***nodurile-frate*** vecine (stâng sau drept) există două valori, atunci una dintre valori se transferă în nodul superior şi de aici se transferă o valoare în nodul curent. După aceste transferuri algoritmul se termină.

Exemplu.

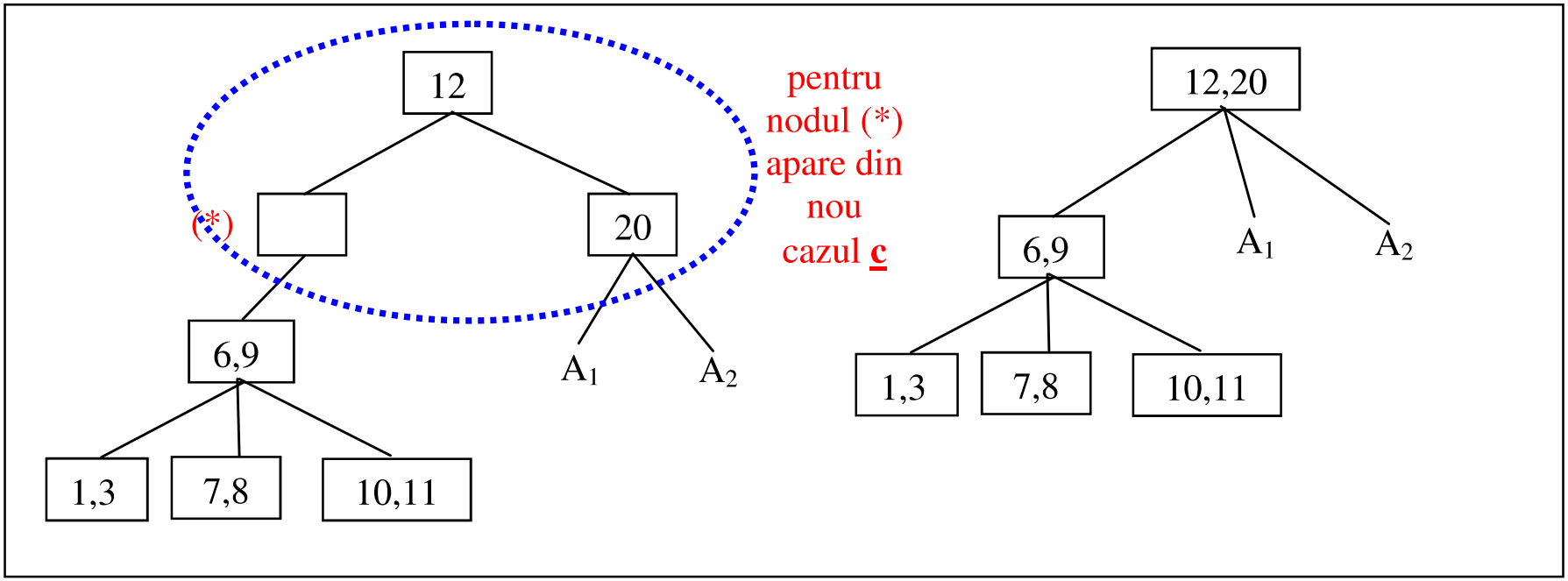


c. Dacă situaţiile precedente nu apar (nodul curent nu mai are valori şi nodurile ***frate-vecine*** au numai câte o valoare), atunci **nodul curent** (fără valori) **se uneşte** cu un nod vecin (acesta are o singură valoare) şi cu o valoare din nodul părinte (se face o concatenare de noduri), după care se analizează din nou cazul 2 din acest algoritm pentru nodul părinte (se analizează poziţia care s-a transferat). Dacă se ajunge la nodul rădăcină şi acesta rămâne fără valori, atunci el se va elimina şi nodul curent devine nod rădăcină.

Exemplu.



Pentru nodul marcat cu (*), de unde s-a eliminat valoarea poziţia ocupată de 3, apare cazul **c**.

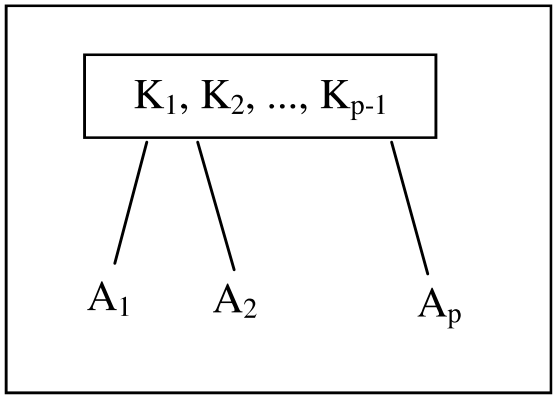

12
(*)
20
6,9
A1
A2
1,3
7,8
10,11
pentru
nodul (*)
apare din
nou
cazul c
12,20
6,9
A1
A2
1,3
7,8
10,11

B-arbore

Generalizare a 2,3-arborilor

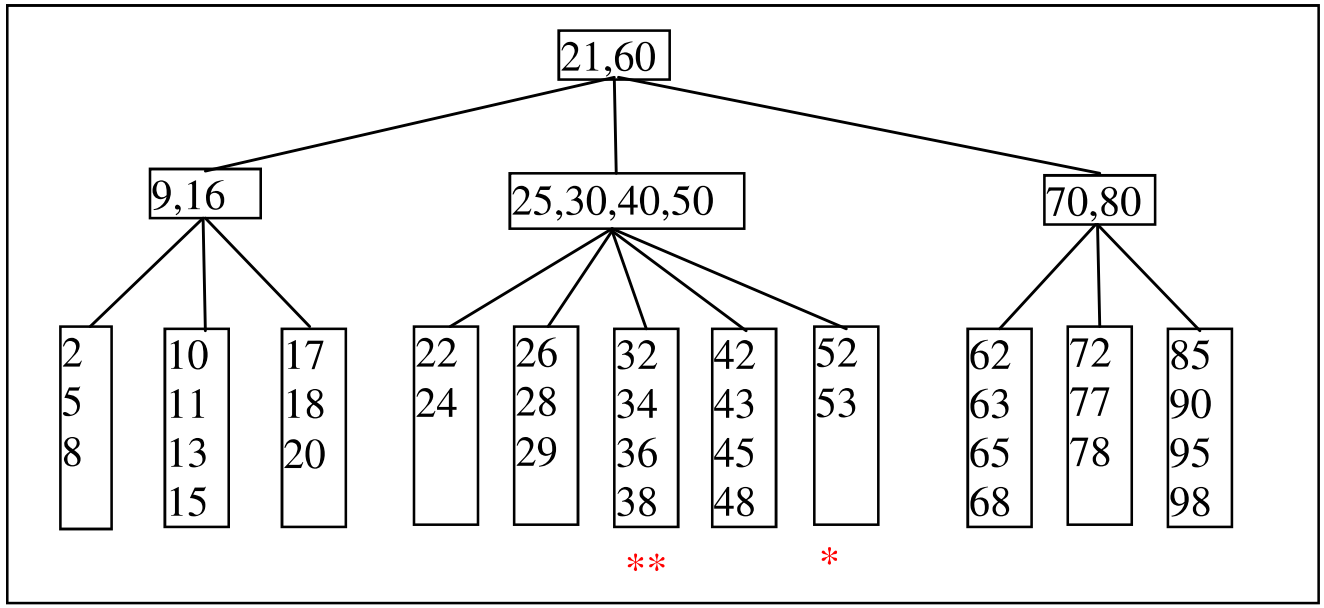
B-arbore de ordin m:

1. Dacă rădăcina nu e terminal, atunci are cel puţin 2 subarbori
2. Toate nodurile terminale sunt pe acelaşi nivel
3. Fiecare nod neterminal are **cel mult m** subarbori
4. Un nod cu **p** subarbori conţine **p-1** valori ordonate crescător: $K_1<K_2<...<K_{p-1}$. In primul subarbore sunt valori mai mici decât K_1, în subarborele A_i sunt valori cuprinse între K_{i-1} şi K_i, i=2,...,p-1, în A_p sunt valori mai mari decât K_{p-1}.
5. Fiecare nod neterminal are **cel puţin** $\lceil m/2 \rceil$ subarbori



Observaţie. Limitele pentru numărul de subarbori (şi implicit numărul de valori) dintr-un nod rezultă din modul de efectuare a operaţiilor de adăugare şi ştergere astfel încât cerinţa a doua din definiţie să fie îndeplinită.

Exemplu pentru un B-arbore de **ordin 5**: la un nod diferit de rădăcină şi neterminal pot apare **cel mult 5 subarbori** şi **cel puţin 3 subarbori**, sau nodul poate conţine **între 2 şi 4 valori**.



Obs. Nodurile marcate cu (*) şi (**) se vor referi la descrierea operaţiei de adăugare.

Memorarea unui B-arbore de ordin m

La fel ca un 2,3-arbore, B-arborele se foloseşte pentru memorarea valorilor unei chei (un index pentru baza de date). In arbore se memorează valorile cheii şi adresa înregistrării care conţine valoarea respectivă pentru cheie.

B-arborele se poate memora în două variante:

1. Să se transforme într-un **arbore binar**. Nodurile cu mai mult de două valori se vor transforma (după aceeaşi metodă ca la 2,3-arbore) în noduri cu o singură valoare.

2. Zona de memorie alocată pentru un nod poate păstra **maximum posibil de valori şi adrese de subarbore**.

NV	K_1	ADR_1	.....	K_{m-1}	ADR_{m-1}	LEG_1	....	LEG_m

unde:

- **NV** este numărul de valori din nod
- **K_1, ..., K_{m-1}** memorează valori ale cheii
- **ADR_1, , ADR_{m-1}** sunt adrese de înregistrări (corespunzătoare valorilor cheii)
- **LEG_1, ..., LEG_{m-1}** sunt adresele de subarbori

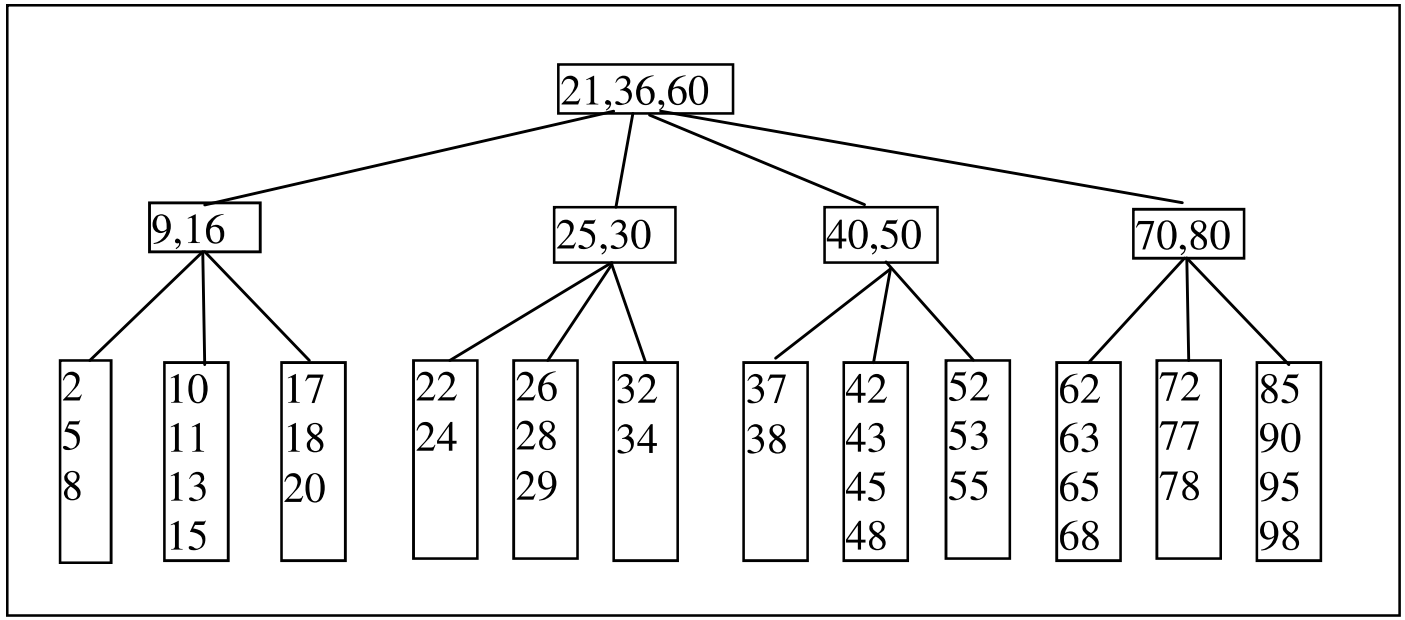
Operaţii (algoritmi) utili pentru un 2,3-arbore:

- **căutarea unei valori - algoritm**
- adăugarea unei valori - **descriere mod de efectuare**
- ştergerea unei valori - **descriere mod de efectuare**
- parcurgerea arborelui (parţial, total)

Adăugarea unei noi valori:

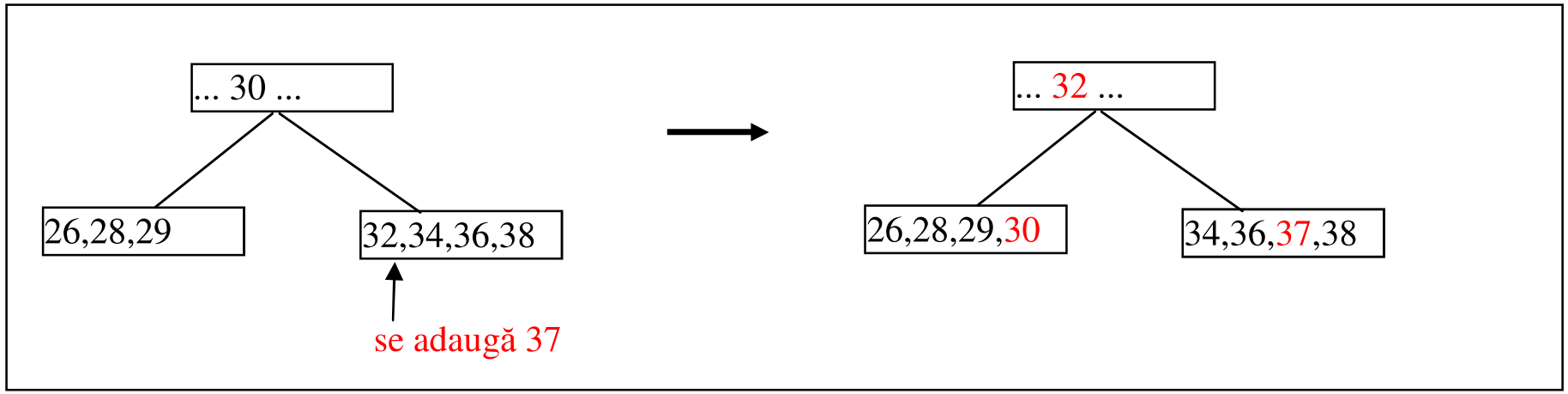
1. Valorile din arbore sunt distincte, deci valoarea care se adaugă nu trebuie să se afle în arbore. Pentru a testa această situaţie, se face o căutare în arbore. Dacă operaţia se poate efectua, atunci **ultima căutare are loc într-un nod terminal**.
2. Dacă nodul terminal la care se ajunge are **mai puţin de (m-1) valori**, atunci noua valoare se poate memora în acest nod. De exemplu, dacă în arborele din figura precedentă se adaugă valoarea 55, atunci se ajunge (cu căutarea) la nodul marcat cu (*). In nodul curent se mai pot adăuga date, deci valoarea 55 se memorează în acest nod.
3. Dacă nodul terminal la care se ajunge are deja maximum (**m-1**) valori, atunci noua valoare se ***ataşează*** acestui nod, cele **m** valori obţinute se sortează, nodul se divide în două noduri, iar valoarea din mijloc (mediana) se ***ataşează*** nodului părinte. Dacă, de exemplu, se adaugă valoarea 37, atunci se ajunge la nodul marcat cu (**), care ar trebui să conţină valorile: 32, 34, 36, 37, 38. Deoarece se depăşeşte capacitatea nodului (de maximum 4 valori), acest nod se va diviza în nodurile 32, 34 şi 37, 38, iar valoarea 36 se va ataşa nodului părinte cu valorile 25,30,40,50. Prin ataşarea valorii 36 şi acest nod va trebui să fie divizat în nodurile cu valorile 25,30 şi 40,50, iar valoarea 36 trece la nodul părinte.

Arborele care se obţine după aceste două operaţii de adăugare este următorul.

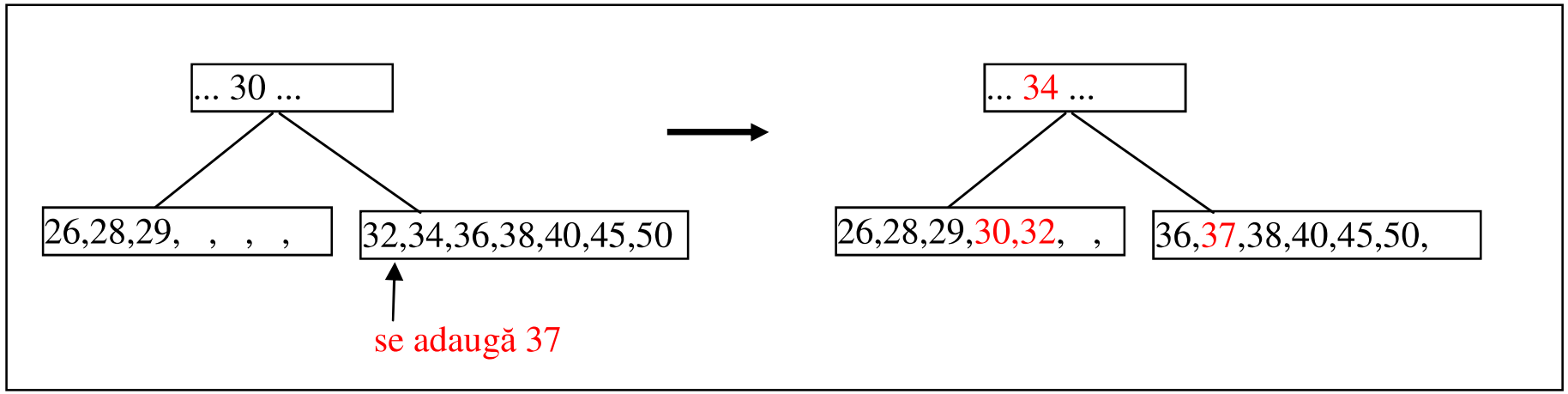

21,36,60
9,16
25,30
40,50
70,80
2
5
8
10
11
13
15
17
18
20
22
24
26
28
29
32
34
37
38
42
43
45
48
52
53
55
62
63
65
68
72
77
78
85
90
95
98

Observaţie pentru optimizarea operaţiei de adăugare. Dacă la adăugare se ajunge într-un nod care trebuie divizat, înainte de divizare se poate analiza şi varianta de a transfera **una sau mai multe valori** din nodul curent (care conţine m-1 valori) într-un nod ***frate-vecin***. Exemple apar în figurile următoare.

Exemplul 1 pentru un B-arbore de ordin 5 (într-un nod neterminal pot să fie între 2 şi 4 valori, sau între 3 şi 5 subarbori).



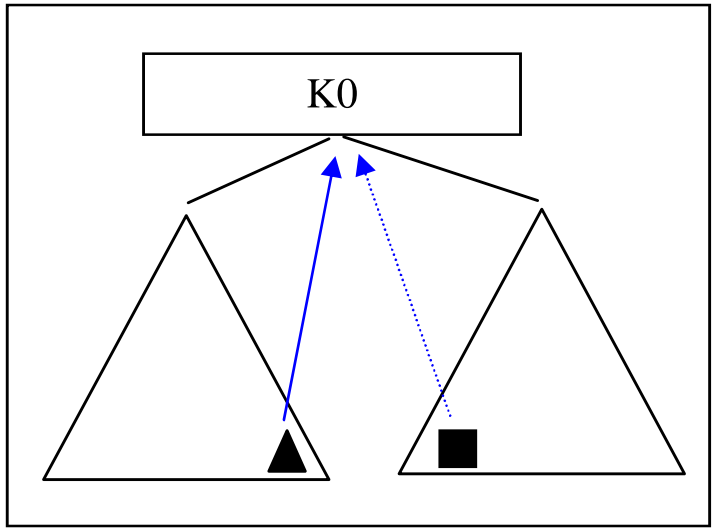
Exemplul 2 pentru un B-arbore de ordin 8 (într-un nod neterminal pot să fie între 3 şi 7 valori, sau între 4 şi 8 subarbori).



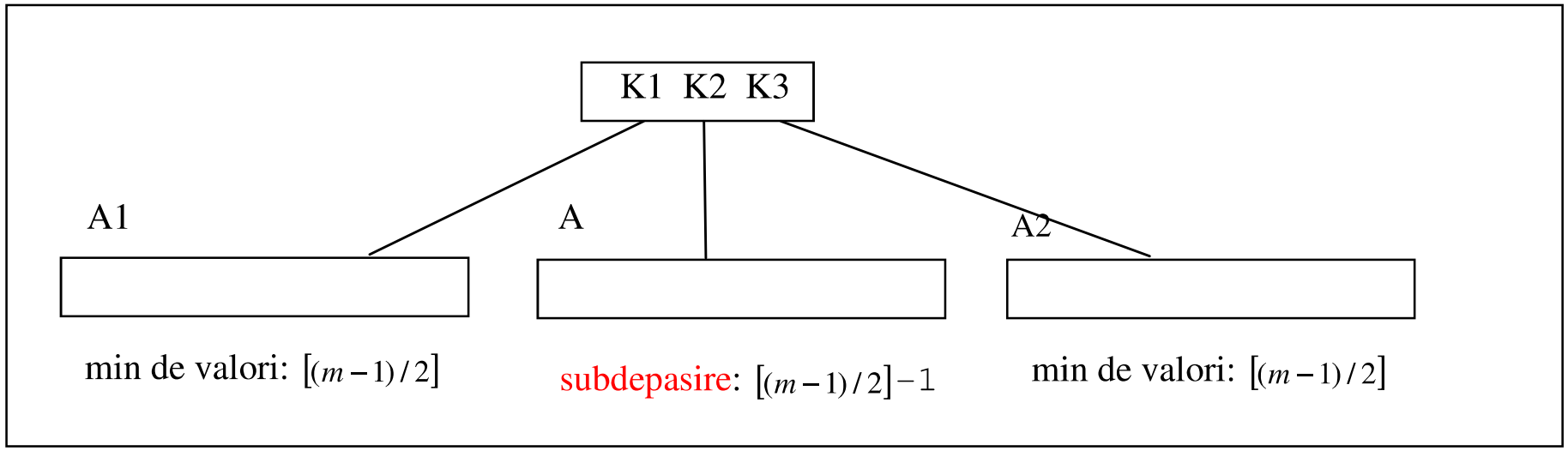
Stergerea unei valori dintr-un **B-arbore de ordin m**. Intr-un nod putem avea un **maxim** de **m** subarbori (sau **m-1** valori) şi un **minim** de $\lceil m/2 \rceil$ subarbori (sau $\lceil m/2 \rceil - 1 = \left\lceil \frac{m-1}{2} \right\rceil$ valori). Prin eliminarea unei valori dintr-un nod **se poate produce *subdepăşire***, adică prin eliminarea valorii pot să rămână mai puţine valori decât este permis.

Presupunem că se cere eliminarea valorii **K0.**

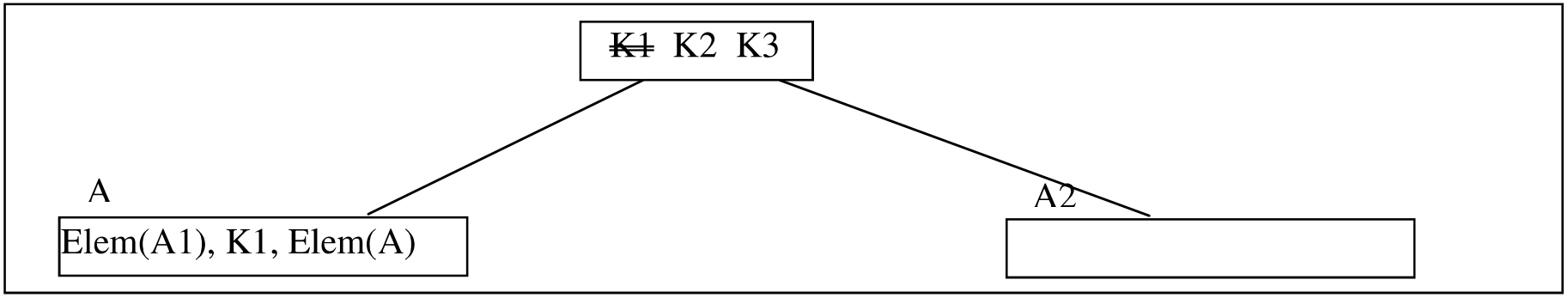
1. Se caută valoarea K0. Dacă această valoare nu este, atunci algoritmul se termină.
2. Dacă nodul unde se află K0 **nu este un nod terminal** (după cum este sugerat în figura alăturată), atunci valoarea K0 **se înlocuieşte** cu o ***valoare vecină*** aflată într-un nod terminal (alegerea se poate face între cel mult două valori, **din arborii cu valori separate de K0**.
3. Se ajunge la un **nod terminal** de unde se elimină o valoare. Fie:
   A = adresa acestui nod;
   R = adresa nodului rădăcină pentru arbore
   Cont = true (variabilă folosită la pasul următor din algoritm)



```
While Cont do {
 if (A=R) or (în A nu apare subdepăşire)
 then {se elimină valoarea din nodul A; Cont = false}
 else{     //prin eliminarea unei valori se produce subdepăşire în A
     if (există un nod frate-vecin B [A1 sau A2] care are cu cel puţin
     o valoare mai mult decât minimum)
     then {se transferă valori între A şi B prin intermediul nodului
     rădăcină, se actualizează legăturile la subarbori}
     else { //este necesară o concatenare de noduri
```

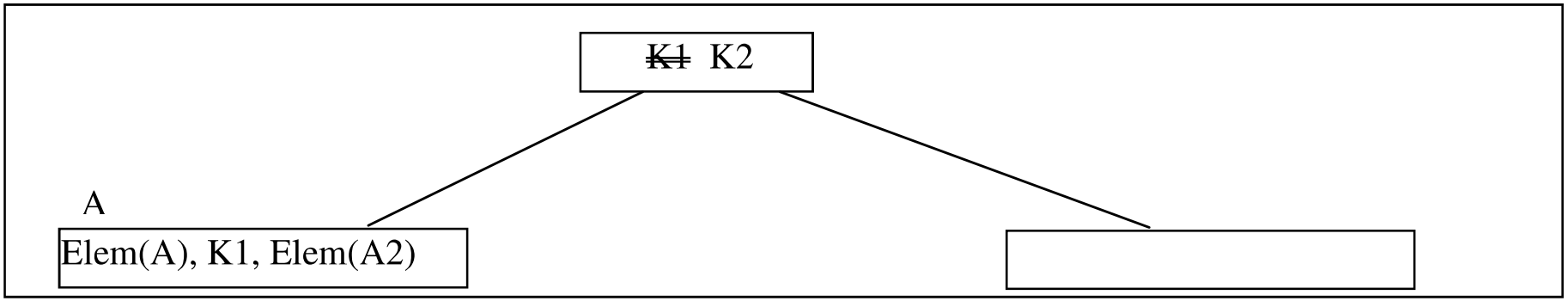


a) **Există A1:** se **unește** A1 cu A și valoarea K1 (care separă A1 de A)



Nodul de la adresa A1 este *redat spațiului liber*.

b) **Nu există A1** (A este primul subarbore pentru nodul parinte): se unește A cu A2 și valoarea K2 (care separă A de A2)



Nodul de la adresa A2 este redat spațiului liber.

}

A=nodul parinte;

}

Optimizări.

Obs.1. Pentru ca o operaţie fizică de citire de pe disc să aducă în memoria internă cât mai multe date necesare în algoritmi, este bine ca într-un bloc (înregistrare fizică) să se memoreze un singur nod din B-arbore.

De exemplu, **dacă**:

- lungimea unei chei: **10b**
- adresa unei înregistrări sau adresa unui nod din arbore: **10b**
- valoarea **NV** (numărul de valori din nod): **2b**
- lungimea unui bloc: **1024b**, din care **10b** pentru antetul blocului

atunci: 2+(m-1)*(10+10)+m*10=1024-10, de unde se obţine: **m=34**

- Dacă dimensiunea unui bloc este de **2048b** şi celelalte valori rămân neschimbate, atunci ordinul arborelui va fi **m=68** (într-un nod pot să fie între 33 şi 67 valori).

Numărul maxim de blocuri (din fişierul care memorează B-aroborele) necesare la căutarea unei valori este dat de numărul maxim de nivele din arbore. Pentru cazul cu m=68, dacă numărul de valori este **1.000.000**, atunci:

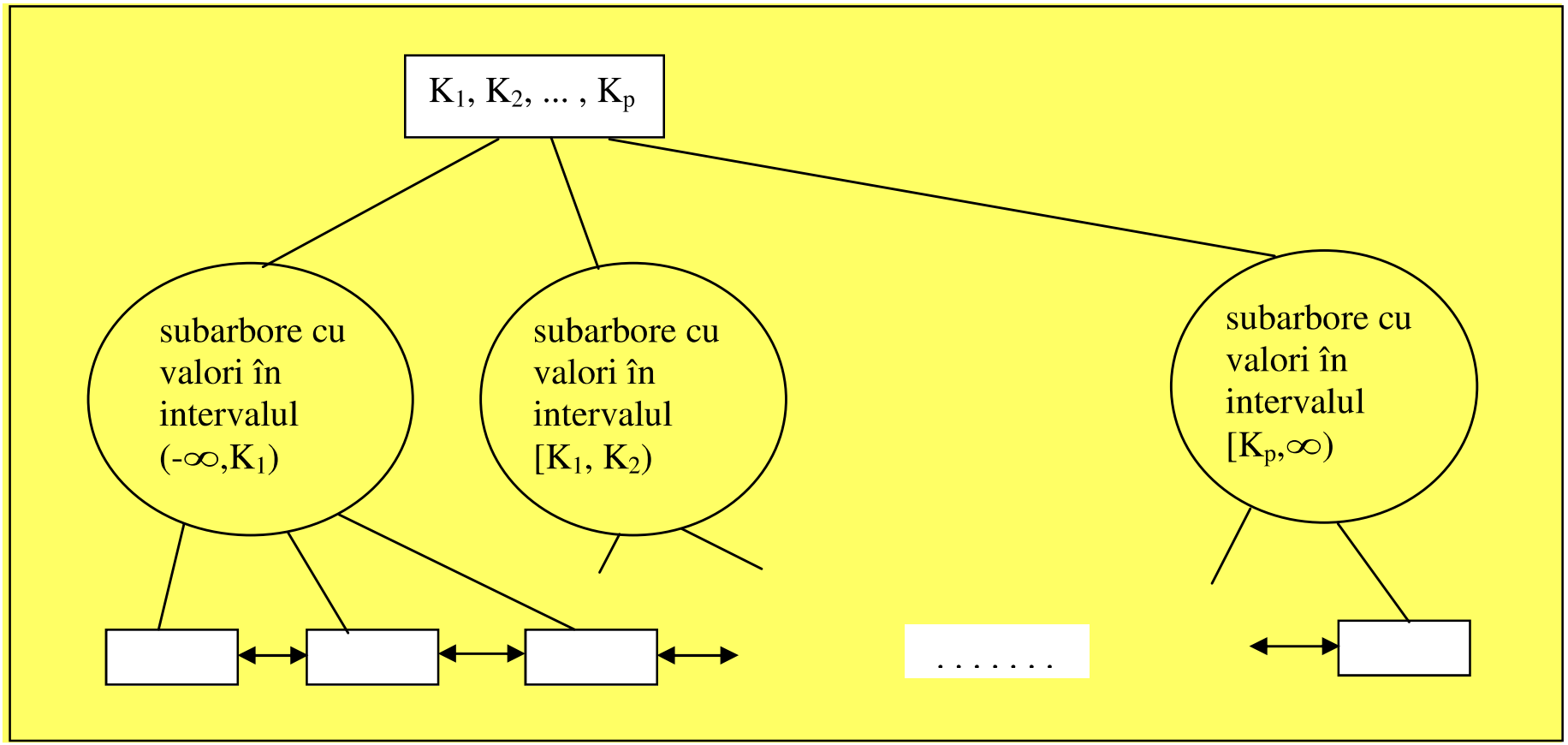
- nodul rădăcină (apare pe nivelul 0) conţine cel puţin o valoare (2 subarbori)
- pe nivelul următor (nivel 1) putem avea minimum 2 noduri * 33 valori/nod = 66 valori
- pe nivelul următor (nivel 2) putem avea minimum 2*34 noduri * 33 valori i/nod =2.244 valori
- pe nivelul următor (nivel 3) putem avea minimum 2*34*34 noduri * 33 valori i/nod = 76.296 valori
- pe nivelul următor (nivel 4) putem avea minimum 2*34*34*34 noduri * 33 valori i/nod = 2.594.064 valori, mai mult decâte valori există, deci acest nivel nu va mai apare

Se ajunge la concluzia că numărul de nivele din B-arbore este maximum 4, deci după maximum 4 citiri de blocuri (citirea de pe suportul extern este operaţia cea mai costisitoare - ca timp) şi un număr de comparaţii în memoria internă se poate determina dacă valoarea căutată există (şi care este adresa înregistrării cu valoarea dată pentru cheie) sau căutarea s-a terminat "fără succes".

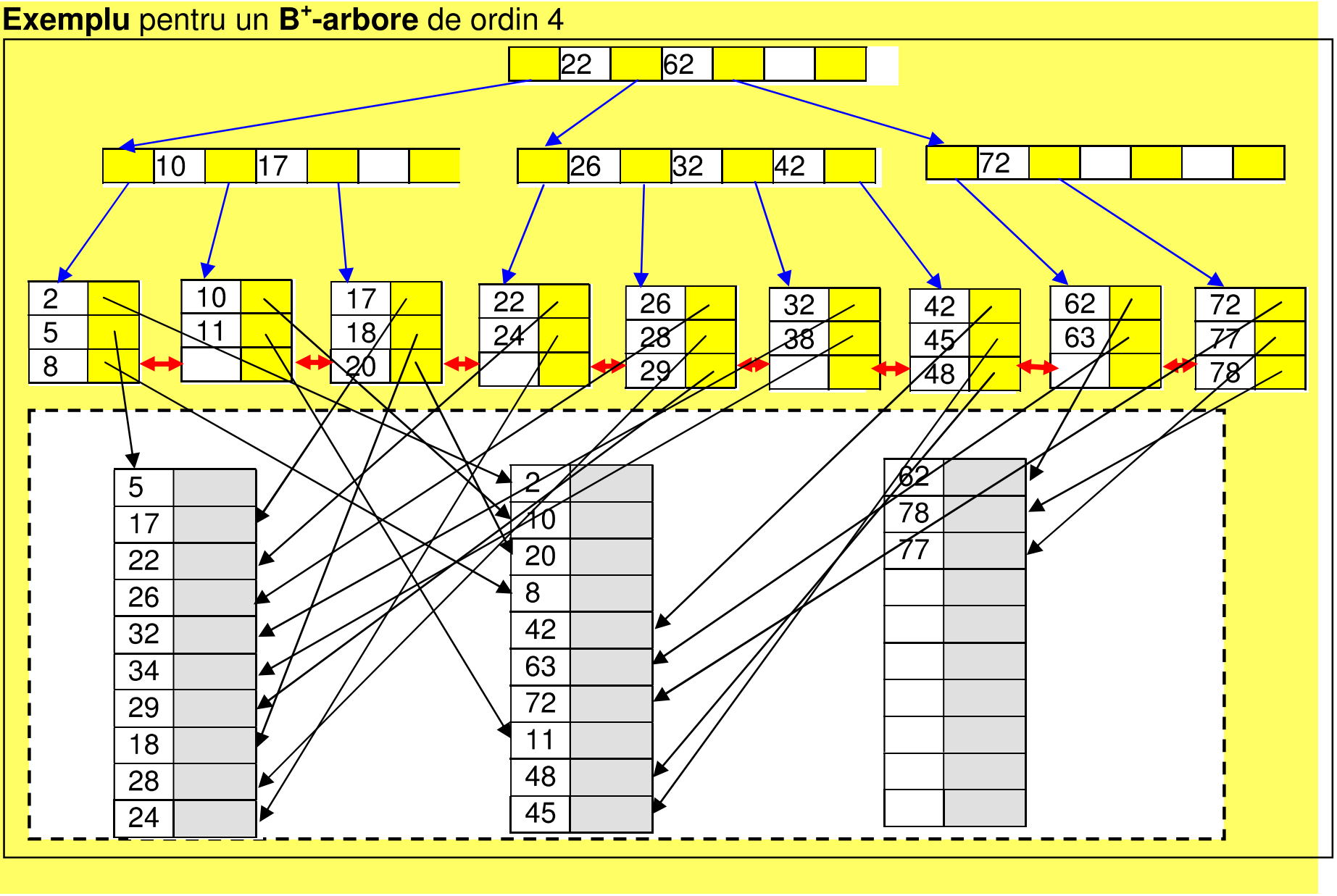
Obs.2. In nodurile terminale **adresele pentru subarbori sunt nule**, deci spaţiul alocat pentru aceste adrese **s-ar putea folosi pentru perechi suplimentare (K, Adr).**

Există multe **variante** de B-arbore, dintre care amintim **B$^+$-arbore**:

- pe ultimul nod apar toate valorile posibile (valori ale cheii şi adresele de înregistrări corespunzătoare)
- unele dintre valorile cheii pot apare şi în nodurile neterminale, fără adresele înregistrărilor, cu scopul de se separa valorile din terminale ("*dirijează*" căutarea)
- toate nodurile terminale se păstrează înlănţuit (se poate uşor efectua parcurgerea arborelui folosind aceste legături)
- un nod intern are numărul minim şi numărul maxim de valori la fel ca la B-arbore
- un nod terminal are cel puţin $\lceil m/2 \rceil$ valori (în loc de $\lceil (m-1)/2 \rceil$ ca la B-arbore) şi cel mult (m-1) valori. Pentru numărul maxim şi minim de valori se poate vedea şi observaţia 2 de la optimizarea unui B-arbore.


K1, K2, ... , Kp
subarbore cu valori în intervalul (-∞,K1)
subarbore cu valori în intervalul [K1, K2)
subarbore cu valori în intervalul [Kp,∞)
. . . . . .

Exemplu pentru un **B$^+$-arbore** de ordin 4



Memorarea unui B^+-arbore:

- metodele de la B-arbore
- se pot face mai multe optimizări (adresele din nodurile interioare sunt adrese de noduri din arbore, adresele din nodurile terminale sunt adrese de înregistrări de pe suport)

Operaţii (algoritmi) în B^+-arbore: ca la un B-arbore

Bibliografie

- Leon Tambulea, UBB, curs de baze de date
- [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, 8, 9,10
- [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, 13, 14