

Structura fizica a bazelor de date relationale (partea 2)

Ioana Ciuciu

ioana.ciuciu@ubbcluj.ro

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

Planificare

Sunt posibile mici
modificari ale planificarii in
timpul semestrului

Saptama na	Curs	Seminar	Laborator
S1	1. Concepte fundamentale ale bazelor de date. Modelare conceptuala	1. Modelul Entitate-Relatie. Modelul relational	1. Modelarea unei BD in modelul ER si implementarea ei in SQL Server
S2	2. Modelul relational de organizare a bazelor de date. Modelare conceptuala		
S3	3. Gestiunea bazelor de date relationale cu limbajul SQL (DDL)	2. Limbajul SQL – definirea si actualizarea datelor	2. Interogari SQL
S4	4. Gestiunea bazelor de date relationale cu limbajul SQL (DML)		
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	4. Proceduri stocate. View. Trigger
S8	8. Structura fizica a bazelor de date relationale		
S9	9. FARA CURS (30 nov.)	5. View-uri. Functii definite de utilizator. Trigger	
S10-11	10-11. Indecsi. Arbori B. Fisiere cu acces direct	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 8

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

▶ Curs 9 : NU SETINE (30 nov.)

▶ Curs 10-II

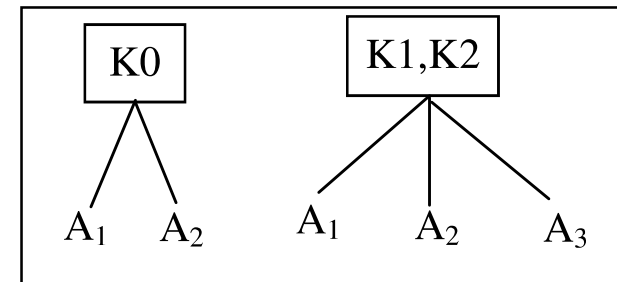
- ▶ 2,3-arbore
- ▶ B-arbore
- ▶ Organizarea directa
- ▶ Index pentru un atribut oarecare
- ▶ Gestiunea indexurilor



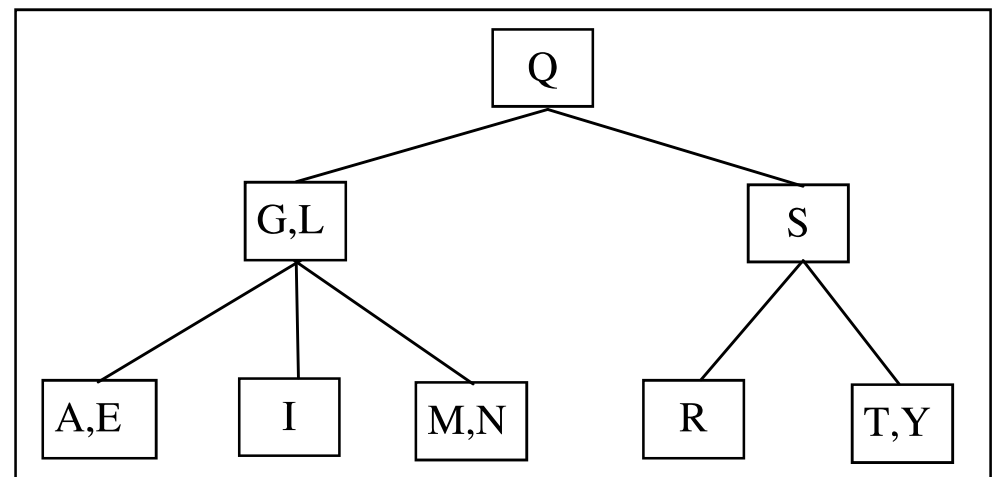
2,3-arbore

Memorează valorile unei chei (colecție de valori distincte):

- **toate nodurile terminale sunt pe același nivel**
- **fiecare nod** conține **una** sau **două** valori ale cheii
 - un nod neterminal cu o valoare K_0 va avea doi subarbori, unul cu valori mai mici decât K_0 și unul cu valori mai mari decât K_0
 - un nod neterminal cu două valori K_1 și K_2 , $K_1 < K_2$, va avea trei subarbori, unul cu valori mai mici decât K_1 , unul cu valori între K_1 și K_2 și un subarbore cu valori mai mari decât K_2



Exemplu (valorile pentru cheie sunt litere):

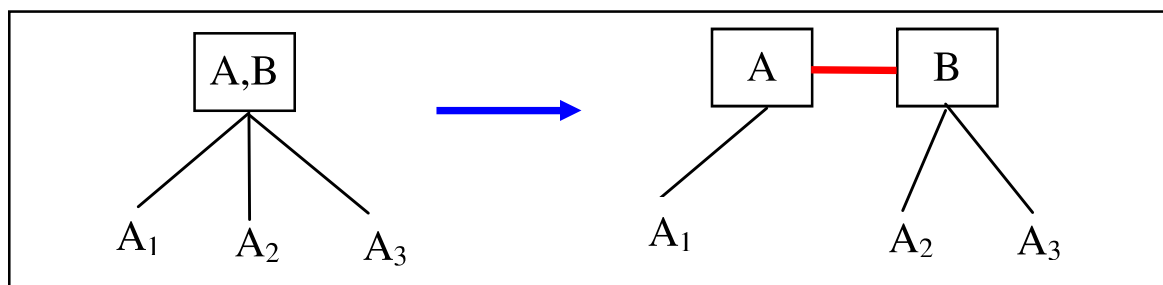


Memorarea unui 2,3-arbore

Un **index** construit ca un **2,3-arbore** se folosește pentru memorarea valorilor unei chei. În arbore se memorează **valorile cheii și adresa înregistrării** (din fișier, din baza de date) care conține 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 fișier)
- **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)

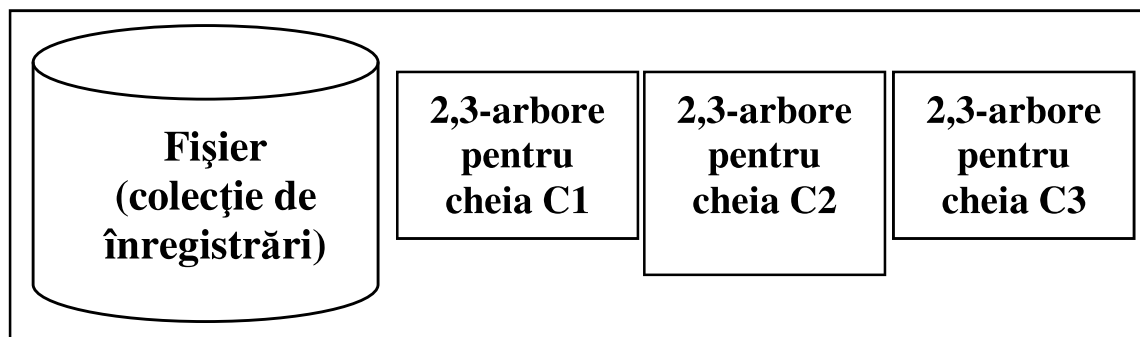
2. Zona de memorie alocată pentru un nod poate păstra două valori și trei adrese de subarbore (maximum de valori posibile).

NV	K1	ADR1	K2	ADR2	LEG1	LEG2	LEG3
-----------	-----------	-------------	-----------	-------------	-------------	-------------	-------------

unde:

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

Observație. La un **fișier** (relație dintr-o bază de date relațională) se pot **asocia** mai mulți 2,3-arbori (câte un 2,3-arbore pentru o cheie).

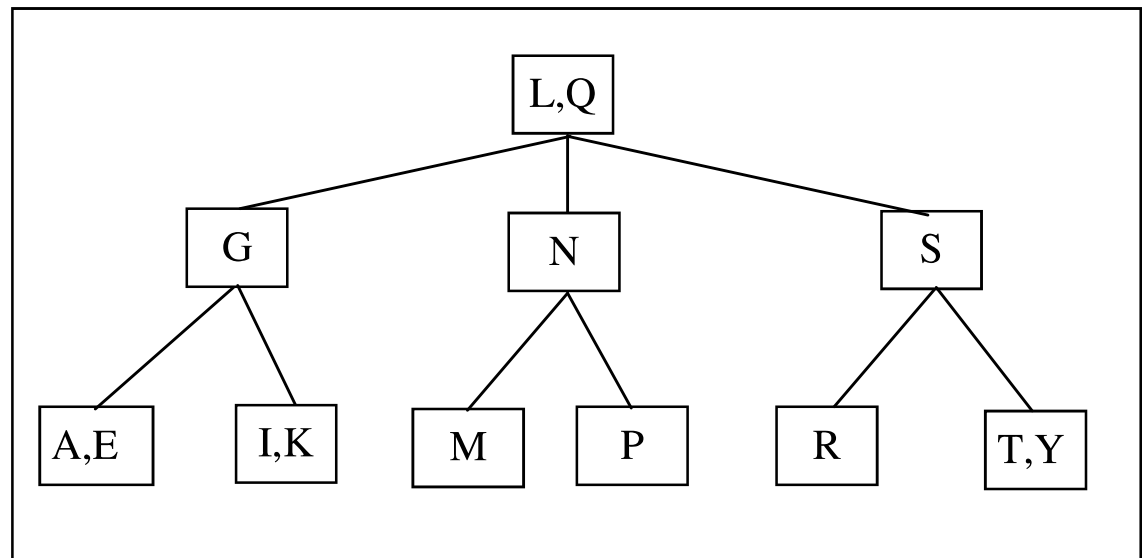


Operații în 2,3-arbore:

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

Adăugarea unei noi valori în 2,3-arbore:

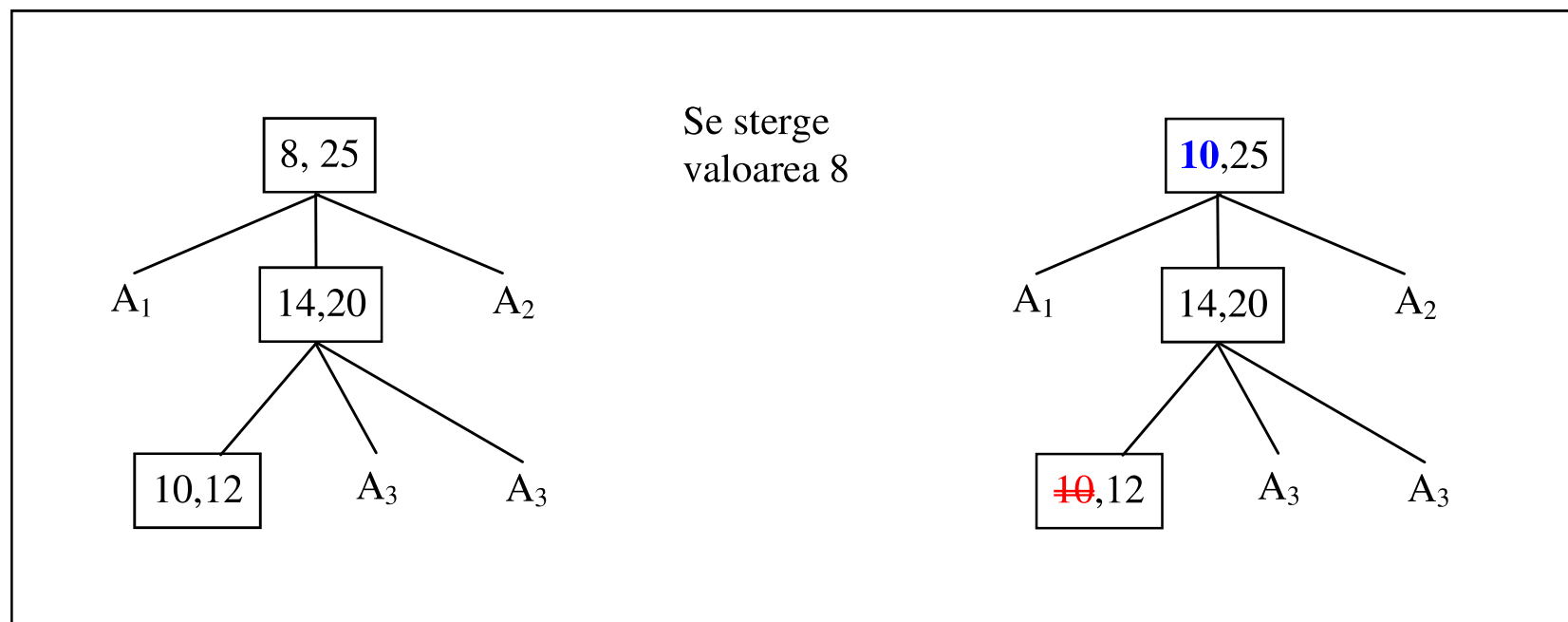
- Valorile din arbore trebuie să fie distincte, deci valoarea care se adaugă nu trebuie să se afle deja în arbore. Pentru a testa această situație, se face o căutare. Dacă operația de adăugare se poate efectua, atunci **ultima căutare are loc într-un nod terminal**.
- Dacă nodul terminal la care se ajunge 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 **atașează** acestui nod, cele trei valori obținute se sortează, nodul se divide în două noduri: un nod va conține valoarea cea mai mică, al doilea nod va avea valoarea cea mai mare, iar valoarea din mijloc se **atașează** nodului părinte. Situația din acest nod părinte se tratează asemănător. De exemplu, dacă în arborele precedent, după adăugarea valorii **K**, se mai adaugă valoarea **P**, atunci se obține 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 și K1). Poziția ocupată anterior de K1 (dintr-un nod terminal) se elimină.

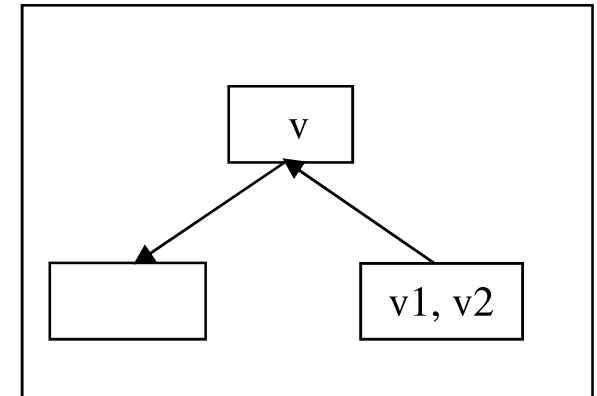
Exemplu.



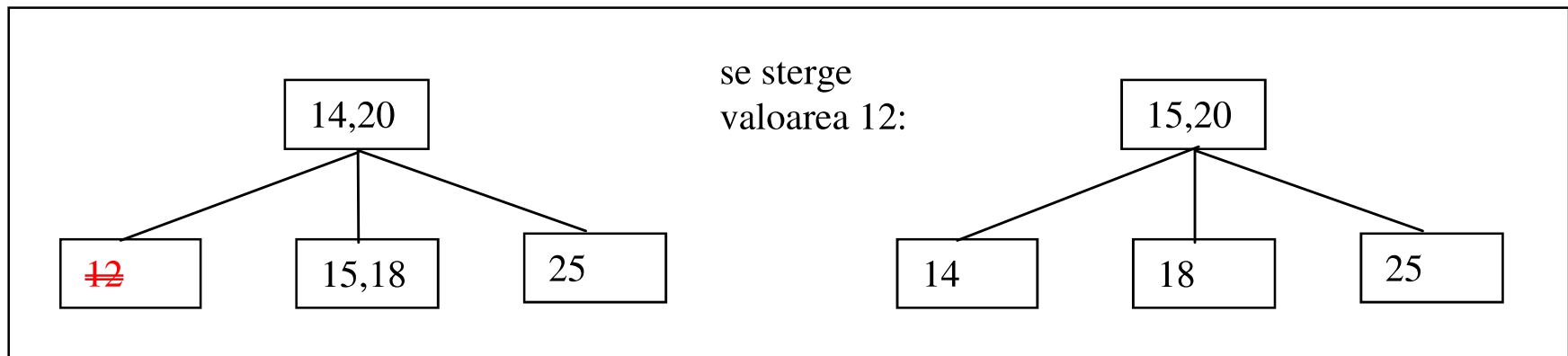
2. Acest pas se execută până apare una din situațiile de la punctele **a** sau **b**.

a. Dacă nodul care conține **valoarea (poziția) v0** ce se **elimină** este rădăcina sau un nod în care mai rămâne o valoare, atunci valoarea **v0** (poziția ocupată de **v0**) se elimină și 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 și de aici se transferă o valoare în nodul curent. După aceste transferuri algoritmul se termină.

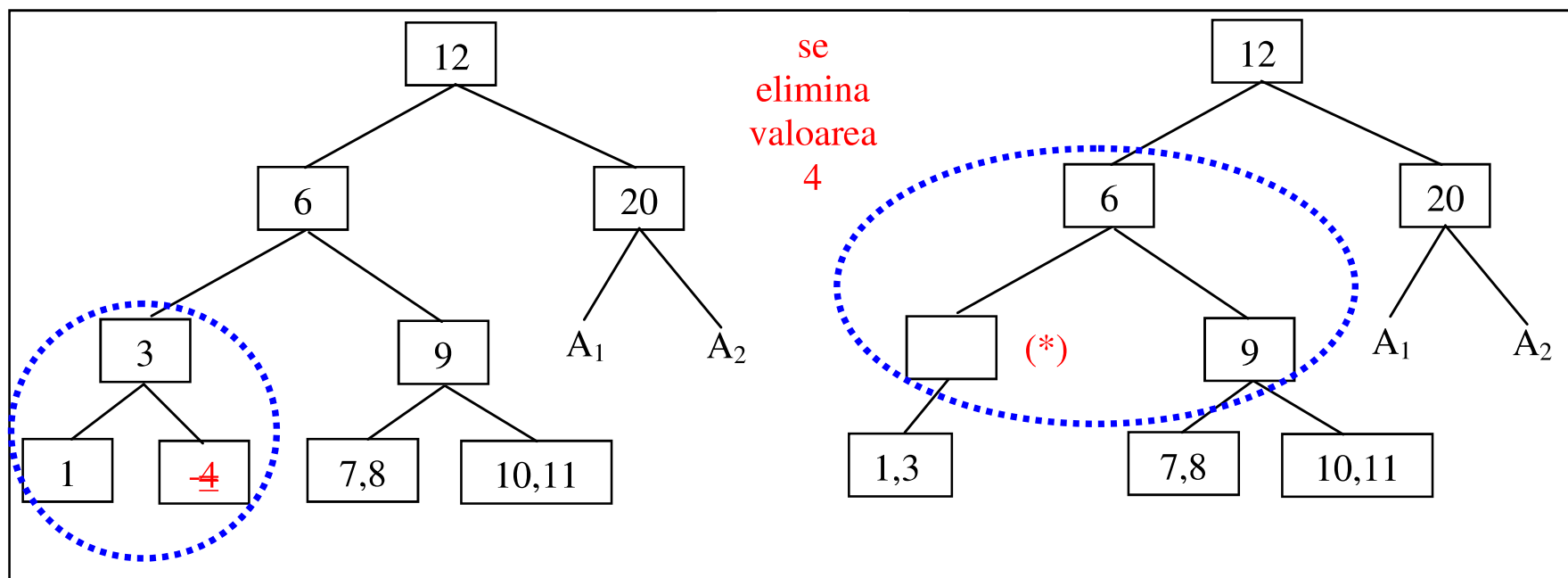


Exemplu.

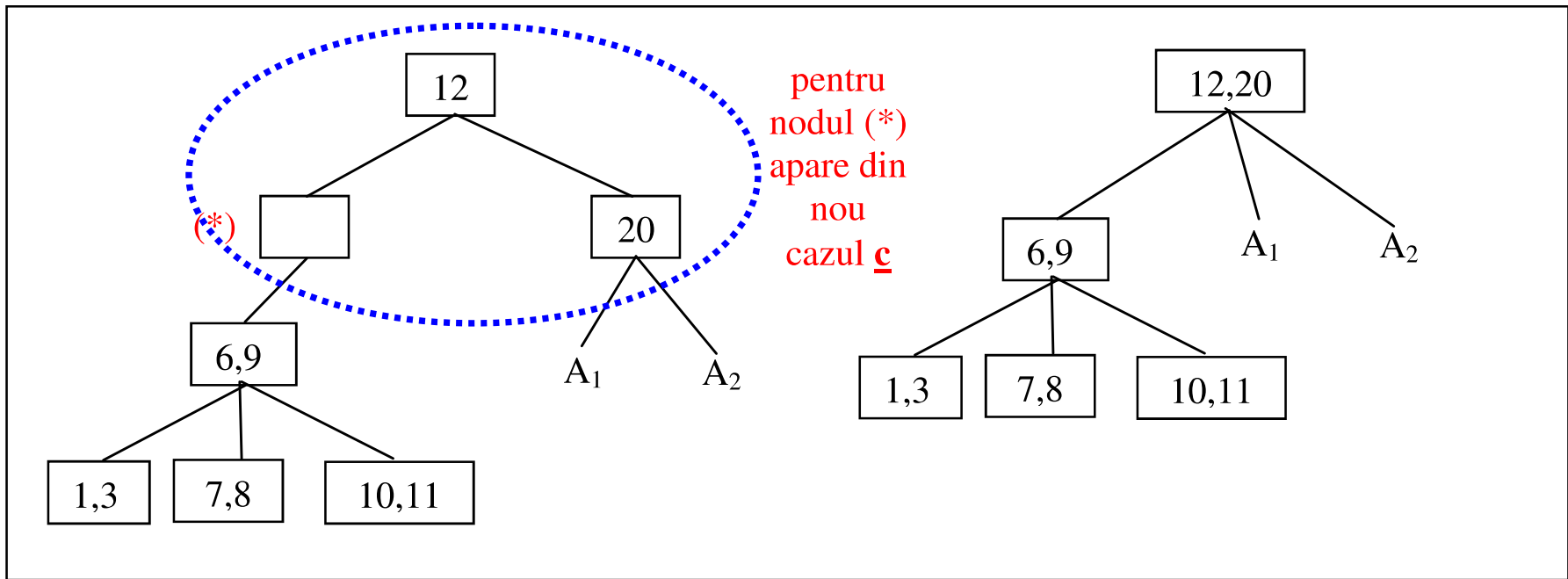


c. Dacă situațiile precedente nu apar (nodul curent nu mai are valori și nodurile *frate-vecine* au numai câte o valoare), atunci **nodul curent** (fără valori) **se unește** cu un nod vecin (acesta are o singură valoare) și cu o valoare din nodul părinte (se face o concatenare de noduri), după care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

Exemplu.



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

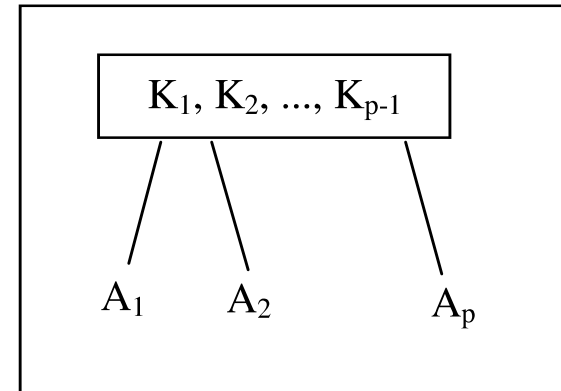


B-arbore

Generalizare a 2,3-arborilor

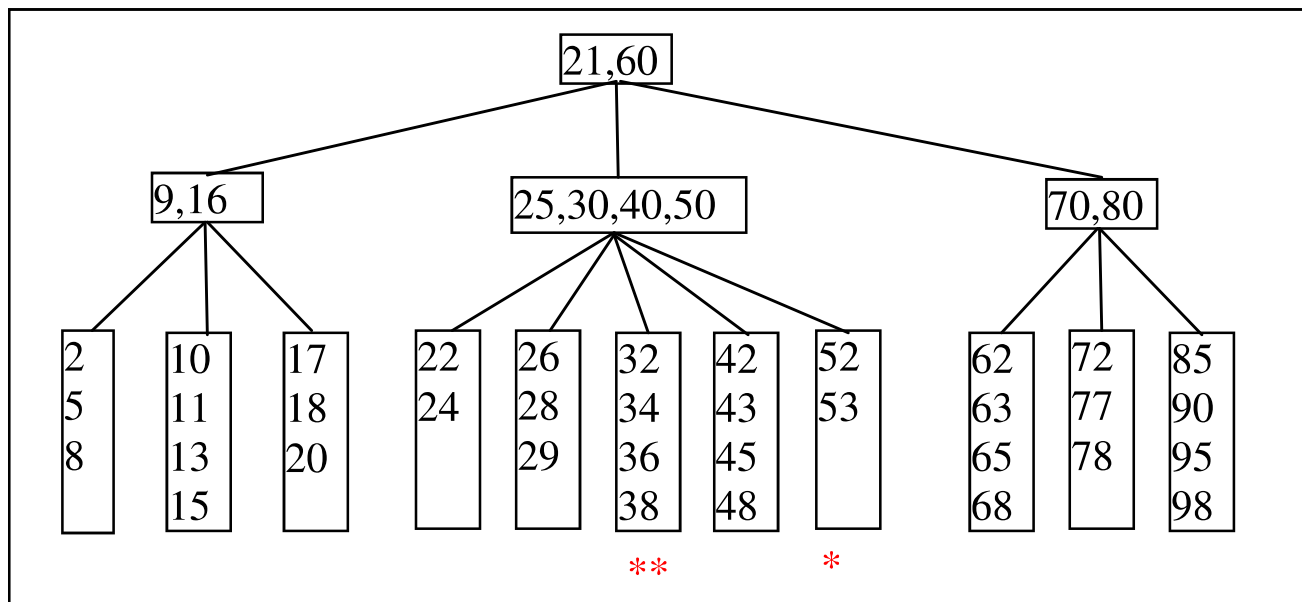
B-arbore de ordin m :

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



Observație. Limitele pentru numărul de subarbori (și implicit numărul de valori) dintr-un nod rezultă din modul de efectuare a operațiilor de adăugare și ștergere astfel încât cerința a doua din definiție să fie îndeplinită.

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



Obs. Nodurile marcate cu (*) și (**) se vor referi la descrierea operației de adăugare.

Memorarea unui B-arbore de ordin m

La fel ca un 2,3-arbore, B-arborele se folosește pentru memorarea valorilor unei chei (un index pentru baza de date). În arbore se memorează valorile cheii și adresa înregistrării care conține 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 aceeași 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 și 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, \dots, K_{m-1} memorează valori ale cheii
- ADR_1, \dots, ADR_{m-1} sunt adrese de înregistrări (corespunzătoare valorilor cheii)
- LEG_1, \dots, LEG_{m-1} sunt adresele de subarbori

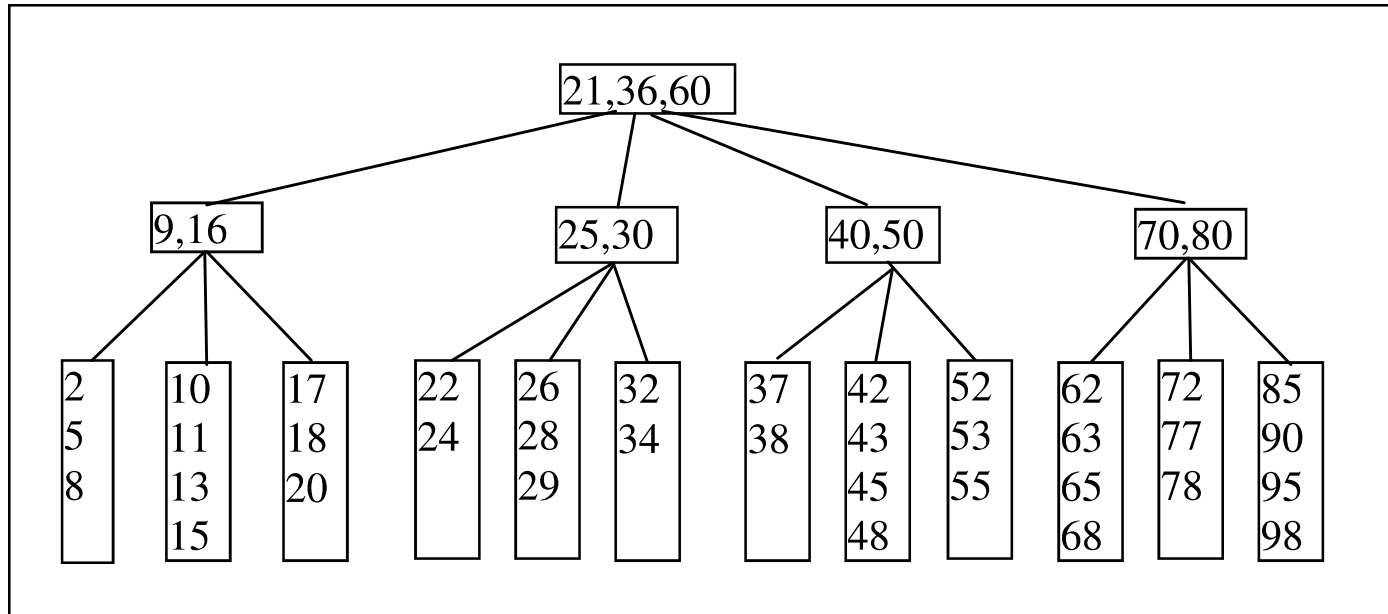
Operații (algoritmi) utili pentru un 2,3-arbore:

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

Adăugarea unei noi valori:

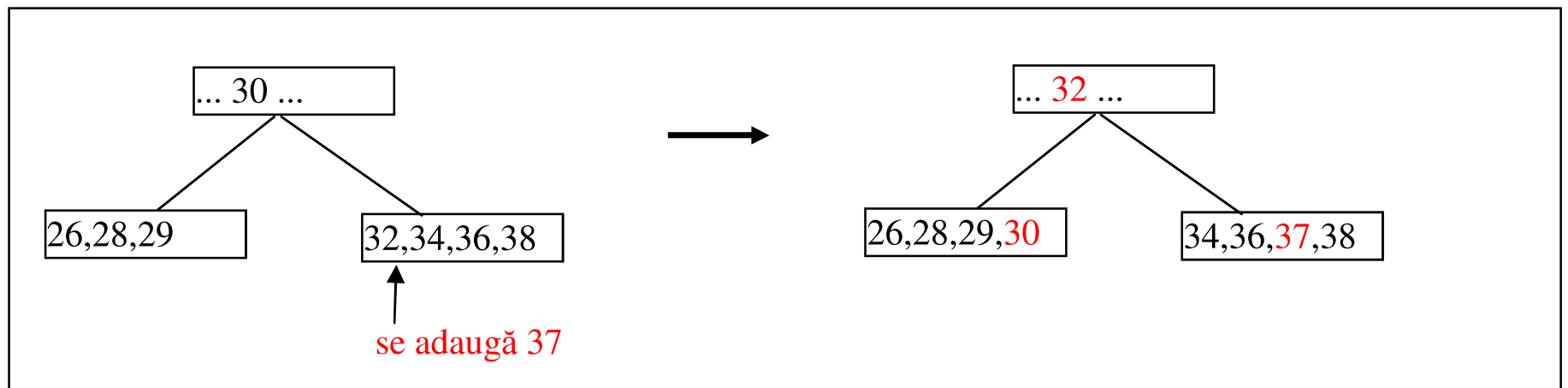
1. Valorile din arbore sunt distincte, deci valoarea care se adaugă nu trebuie să se afle în arbore. Pentru a testa această situație, se face o căutare în arbore. Dacă operația se poate efectua, atunci **ultima căutare are loc într-un nod terminal**.
2. Dacă nodul terminal la care se ajunge are **mai puțin 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 (*). În 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 **atașează** acestui nod, cele **m** valori obținute se sortează, nodul se divide în două noduri, iar valoarea din mijloc (mediana) se **atașează** nodului părinte. Dacă, de exemplu, se adaugă valoarea 37, atunci se ajunge la nodul marcat cu (**), care ar trebui să conțină valorile: 32, 34, 36, 37, 38. Deoarece se depășește capacitatea nodului (de maximum 4 valori), acest nod se va diviza în nodurile 32, 34 și 37, 38, iar valoarea 36 se va atașa nodului părinte cu valorile 25,30,40,50. Prin atașarea valorii 36 și acest nod va trebui să fie divizat în nodurile cu valorile 25,30 și 40,50, iar valoarea 36 trece la nodul părinte.

Arborele care se obține după aceste două operații de adăugare este următorul.

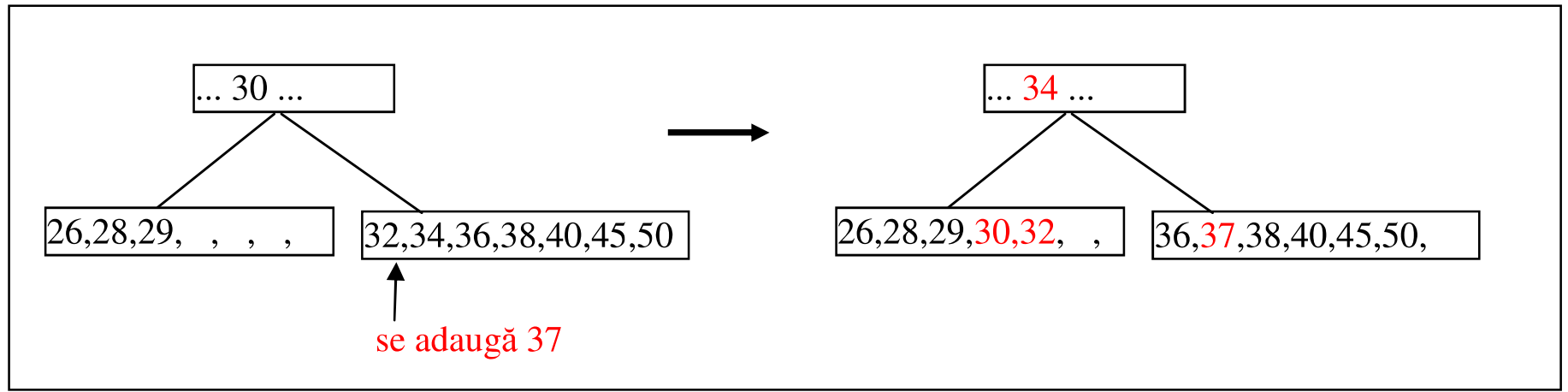


Observație pentru optimizarea operației de adăugare. Dacă la adăugare se ajunge într-un nod care trebuie divizat, înainte de divizare se poate analiza și varianta de a transfera **una sau mai multe valori** din nodul curent (care conține $m-1$ valori) într-un nod **frate-vecin**. Exemple apar în figurile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

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



Exemplul 2 pentru un B-arbore de ordin 8 (într-un nod neterminal pot să fie între 3 și 7 valori, sau între 4 și 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) și 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 subdepășire**, adică prin eliminarea valorii pot să rămână mai puține valori decât este permis.

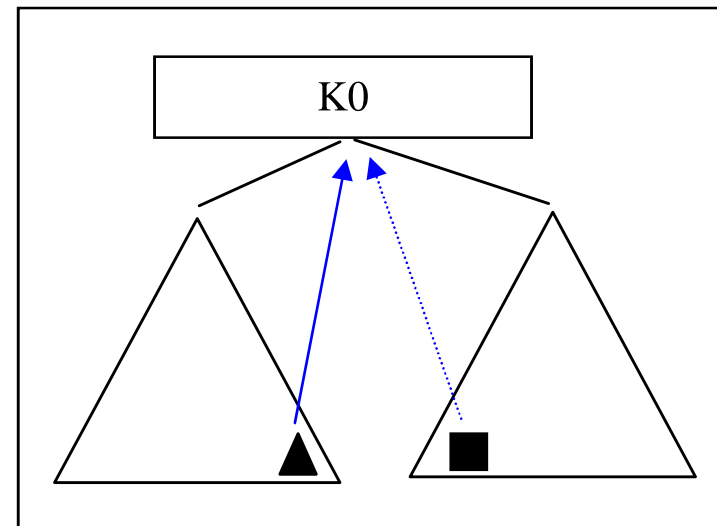
Presupunem că se cere eliminarea valorii **K0**.

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 înlocuiește** 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

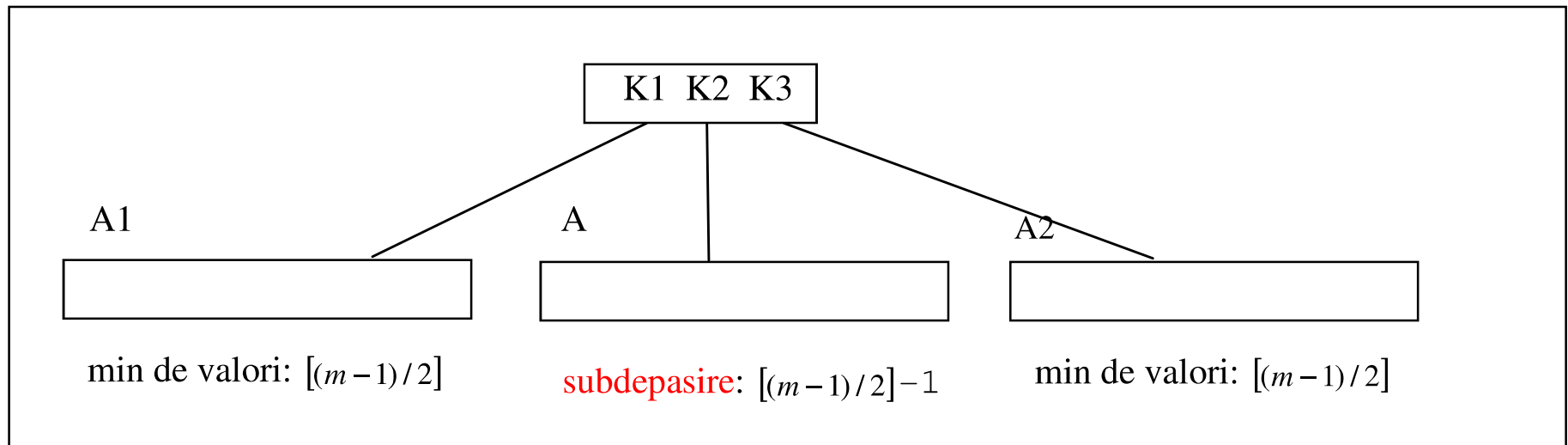
Cont = true (variabilă folosită la pasul următor din algoritm)



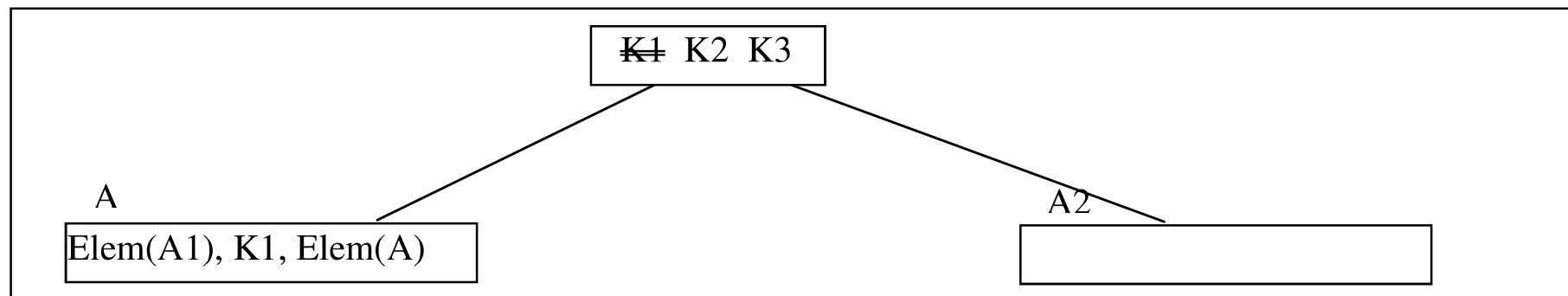
```

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

```

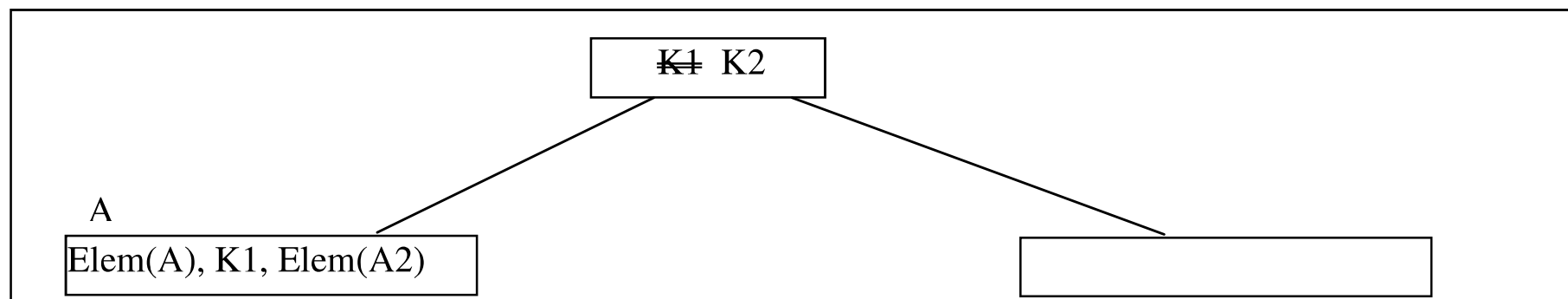


a) **Există A1**: se **unește** A1 cu A și valoarea K1 (care separă A1 de A)



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 operație 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: $2 + (m-1) \cdot (10 + 10) + m \cdot 10 = 1024 - 10$, de unde se obține: **m=34**

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

Numărul maxim de blocuri (din fișierul 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:

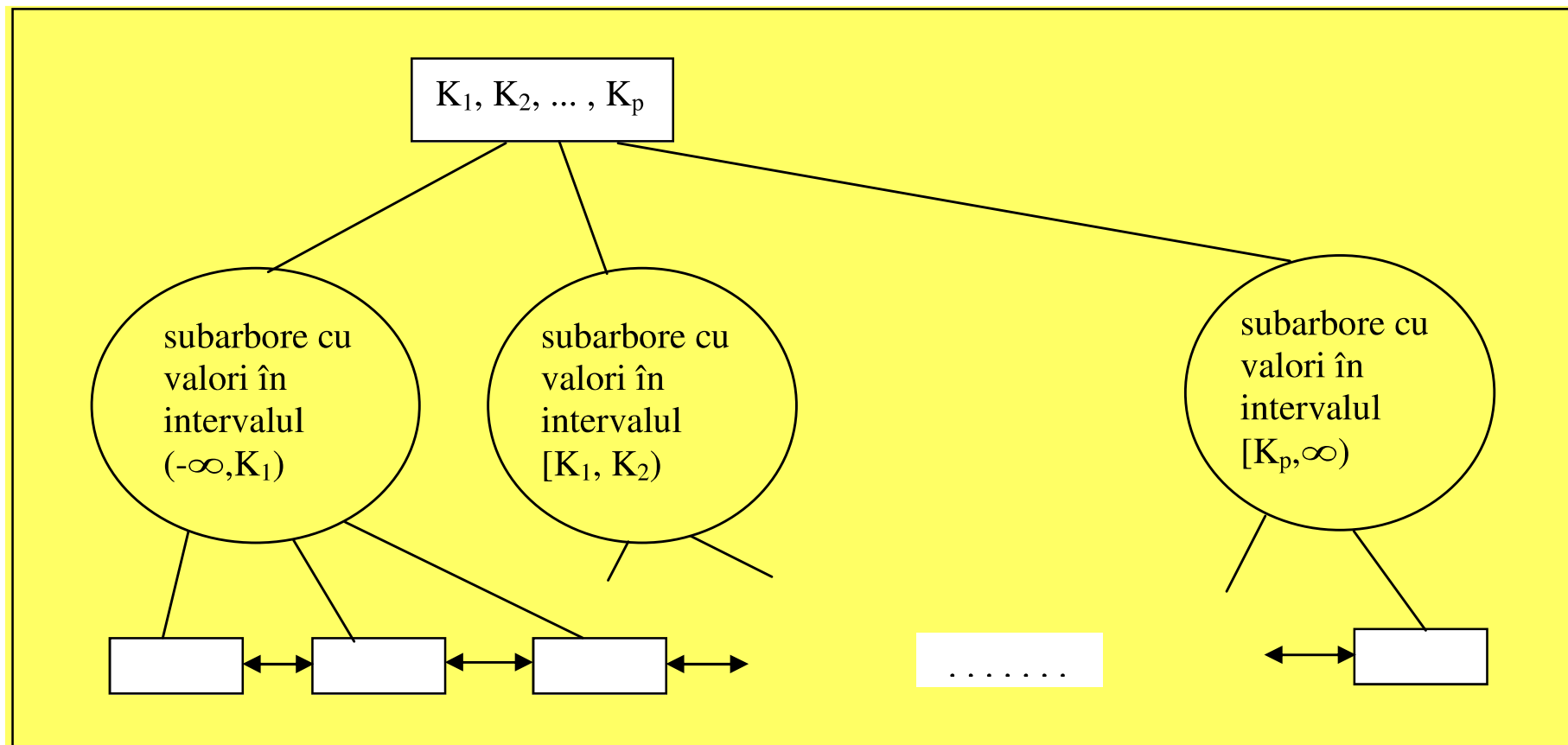
- nodul rădăcină (apare pe nivelul 0) conține cel puțin o valoare (2 subarbori)
- pe nivelul următor (nivel 1) putem avea minimum 2 noduri * 33 valori/nod = 66 valori
- pe nivelul următor (nivel 2) putem avea minimum $2*34$ noduri * 33 valori i/nod = 2.244 valori
- pe nivelul următor (nivel 3) putem 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 decât 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 operația cea mai costisitoare - ca timp) și un număr de comparații în memoria internă se poate determina dacă valoarea căutată există (și care este adresa înregistrării cu valoarea dată pentru cheie) sau căutarea s-a terminat "fără succes".

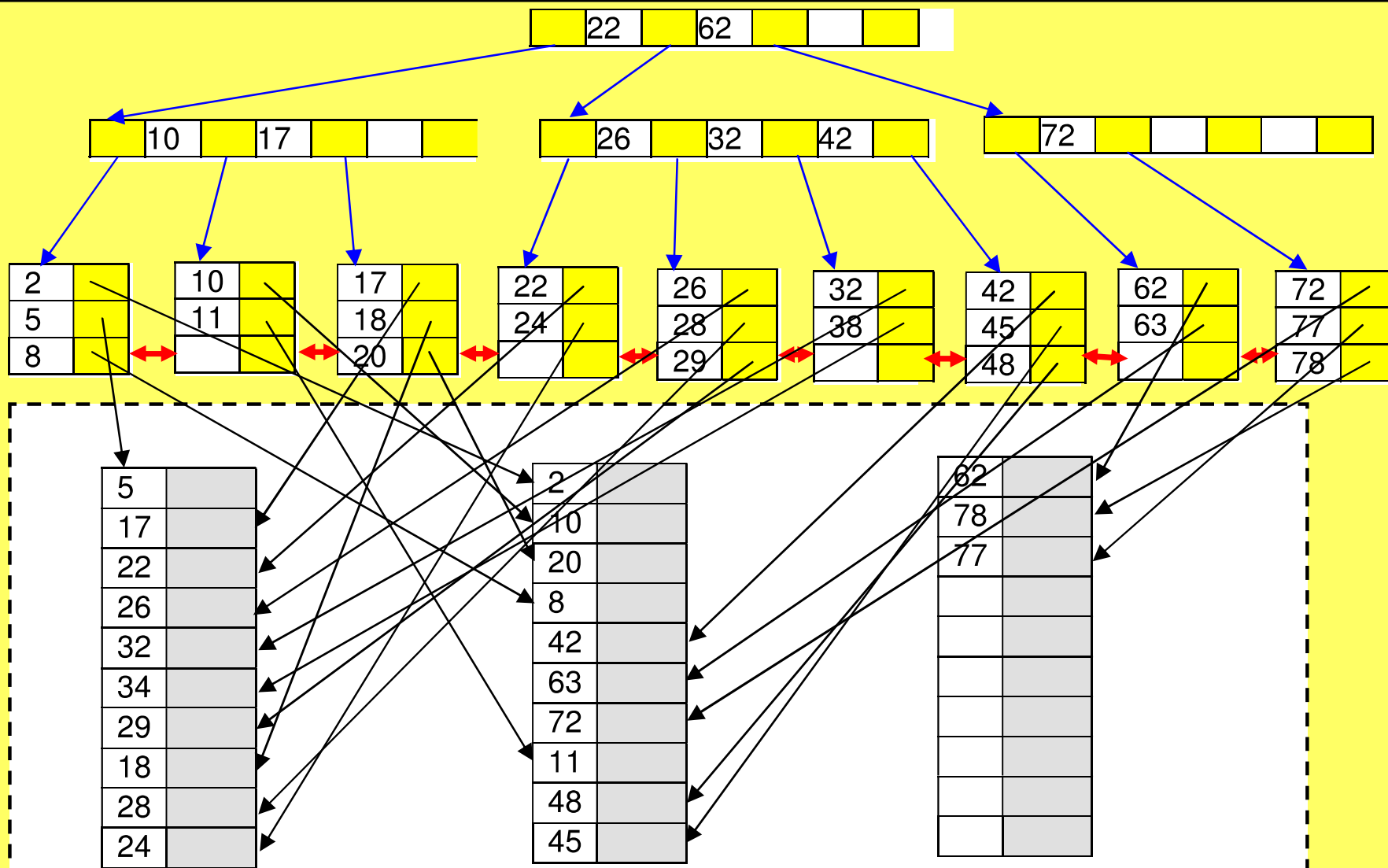
Obs.2. În nodurile terminale **adresele pentru subarbori sunt nule**, deci spațiul alocat pentru aceste adrese **s-ar putea folosi pentru perechi suplimentare (K, Adr)**.

Există multe **variente** de B-arbore, dintre care amintim **B⁺-arbore**:

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



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)

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

Referințe:

- [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 și căutare*. 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