Fişiere cu structură arborescentă

9

Structură de arbore binar

- Fişiere *heap* sau sortate - utile pt tabele statice
- Arbori binari:
  - eficienţi pentru inserare /ştergere inregistrări
  - utilizare algoritmi de căutare binară
- Structura memoriei pentru un nod:

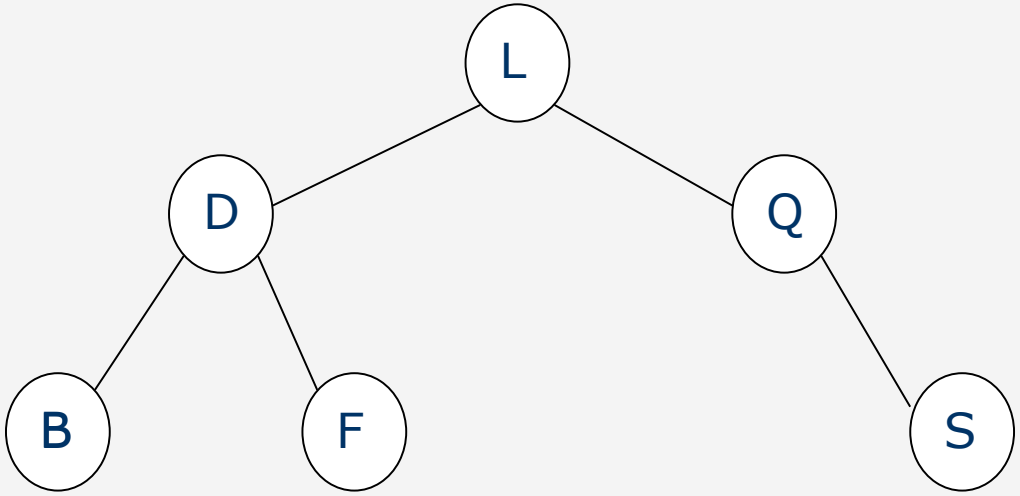
- Structura memoriei pentru arbore binar:
  - colecţie de noduri; referință rădăcină
  - listă de noduri libere (înlănţuite prin *Pointer*$_{Left}$)

Structură de arbore binar

- *Root* – referință rădăcină
- *Free* – referință cap listă noduri libere
- Exemplu:

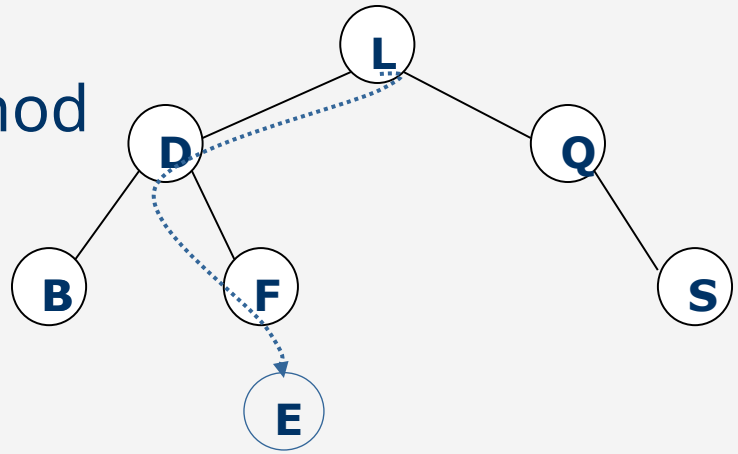
Root = 1
Free = 3

1	L	$Data_L$	2	4
2	D	$Data_D$	8	7
3			-6	NULL
4	Q	$Data_Q$	NULL	5
5	S	$Data_S$	NULL	NULL
6			-9	NULL
7	F	$Data_F$	NULL	NULL
8	B	$Data_B$	NULL	NULL
9			NULL	NULL

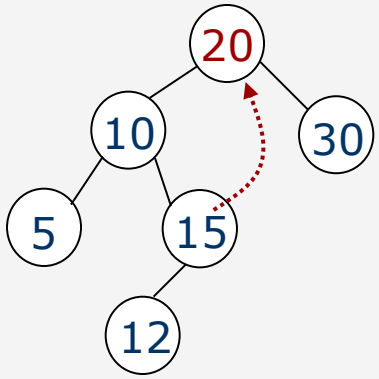


Inserare / Eliminare Înregistrări

- Inserare înregistrare:
  - detectare poziţie înregistrare
  - stocare înregistrare nouă într-un nod liber
  - legare nod la părinte

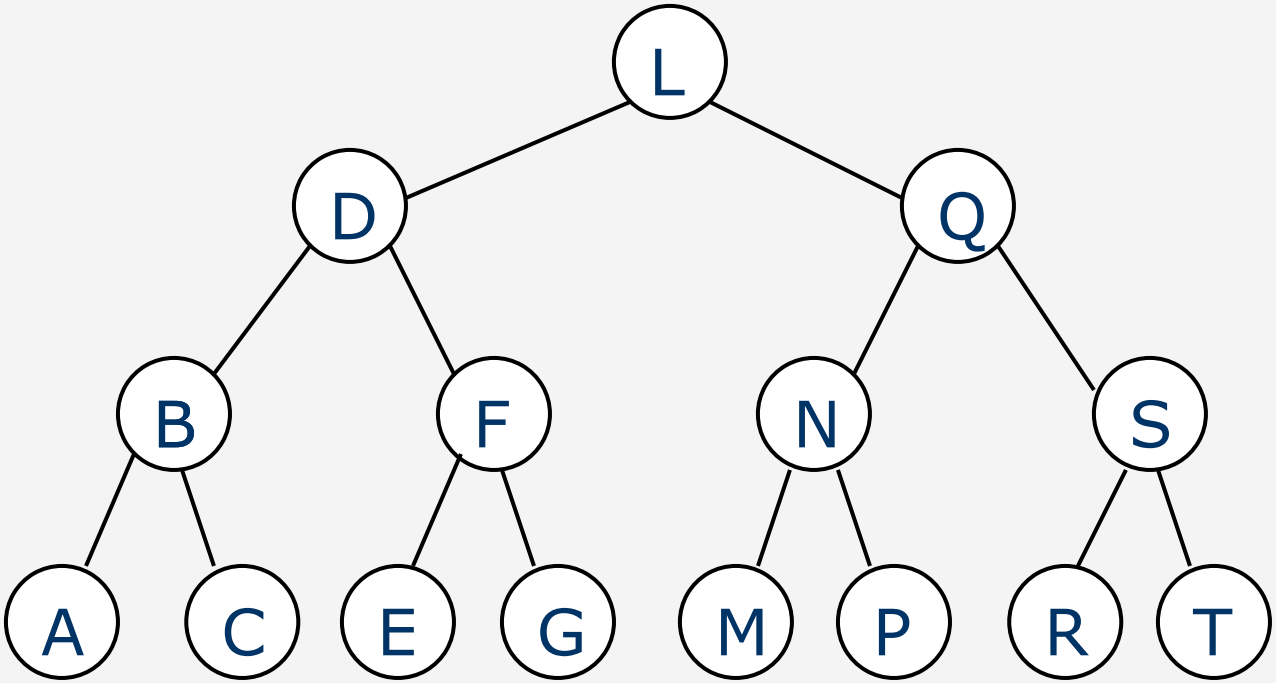


- Eliminare înregistrare:
  - căutare înregistrare
  - 3 cazuri:
    - fără copii: pointer părinte= NULL
    - 1 copil: ataşează nod copil la părinte
    - 2 copii: înlocuieşte cu cel mai apropiat vecin
  - adaugă nodul în lista de noduri libere



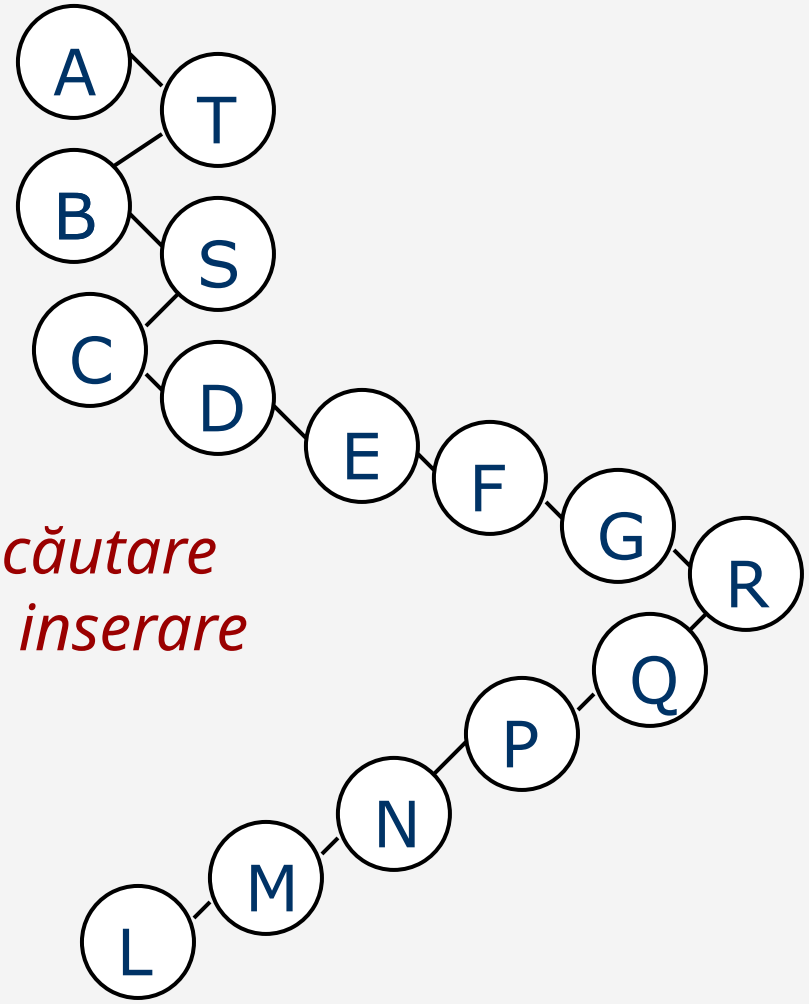
Anomalie de inserare în arbore binar

- L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C



Anomalie de inserare în arbore binar

- A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L



Dezavantaj: timpul de căutare depinde de ordinea de inserare a înregistrărilor

Arbori binari optimali

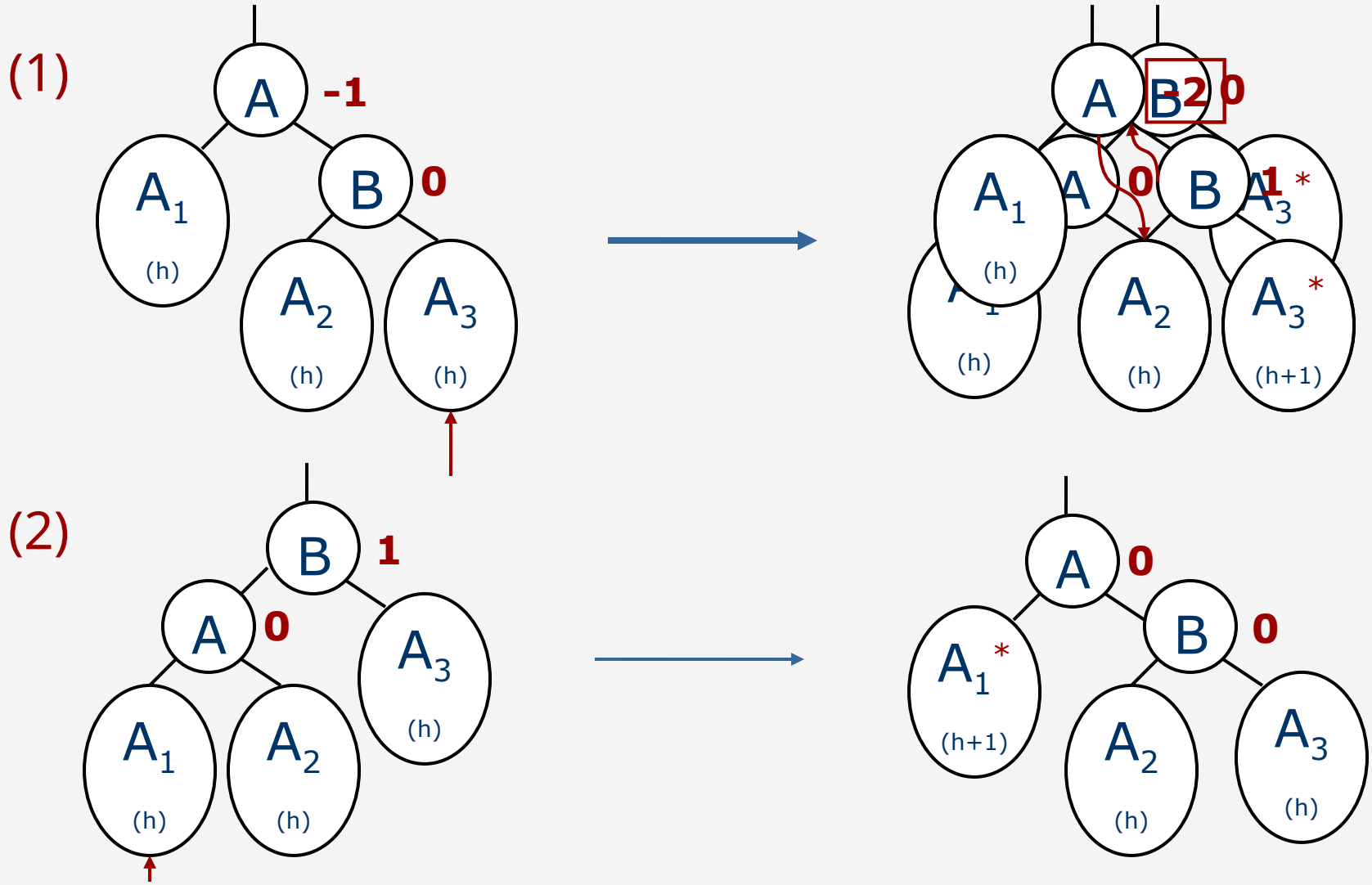
- frunzele sunt poziţionate pe cel mult 2 nivele

- întreţinerea e dificilă → consumă timp pentru adăugare, ștergere sau actualizare

Arbori binari echilibraţi

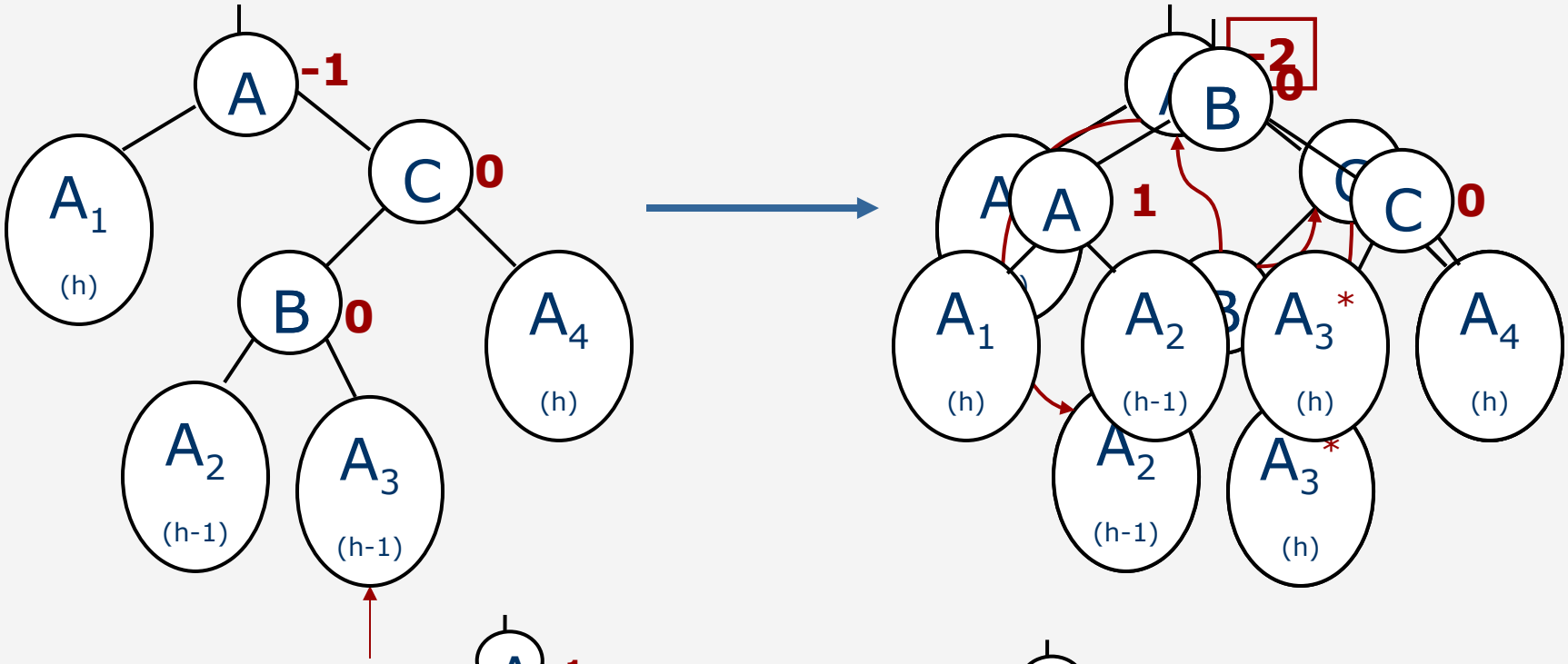
- pentru fiecare nod diferenţa dintre înălţimile sub-arborilor săi este 0, 1 sau –1 (*înălţime arbore*: dimensiunea celui mai lung drum de la rădăcină la frunze)
- număr redus de operaţii de întreţinere
- 6 cazuri distincte de “dezechilibrare” a unui arbore după o inserare

Întreţinere arbori echilibraţi

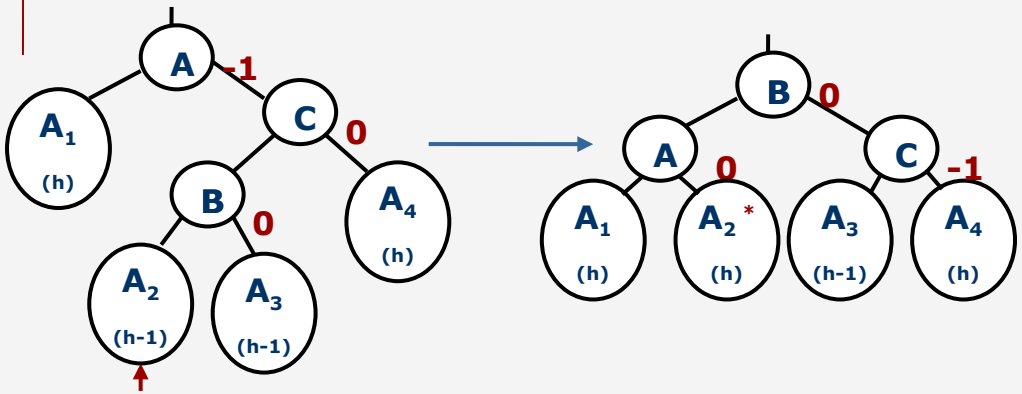


Întreţinere arbori echilibraţi

(3)

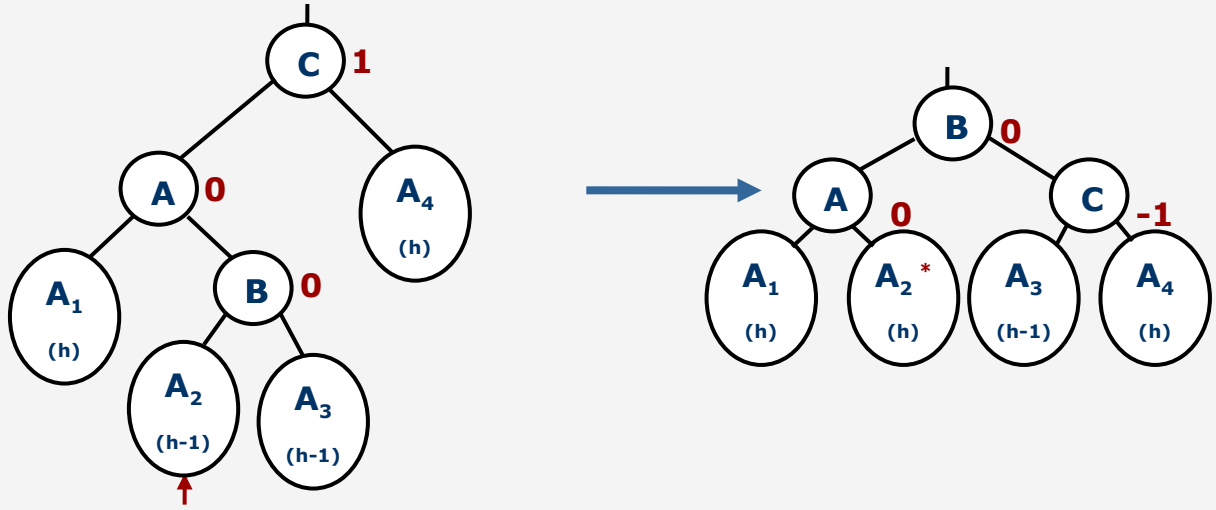


(4)

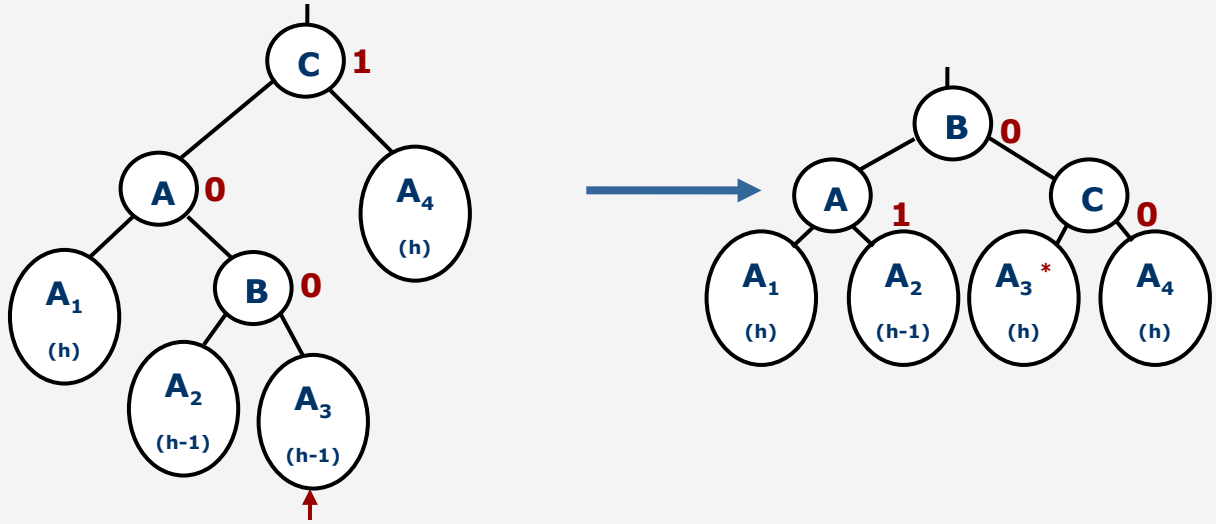


Întreţinere arbori echilibraţi

(5)

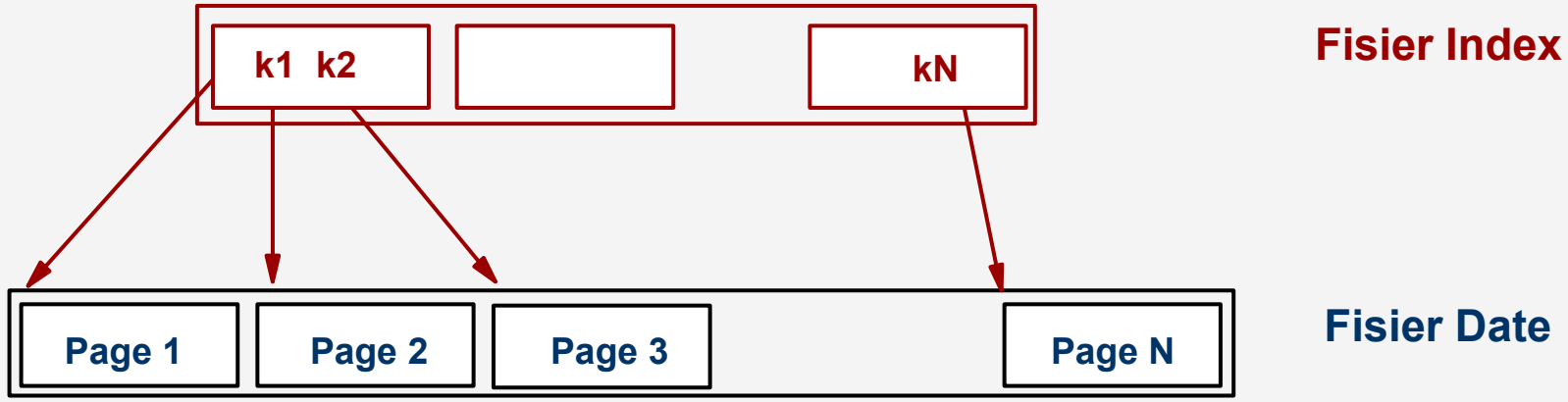


(6)



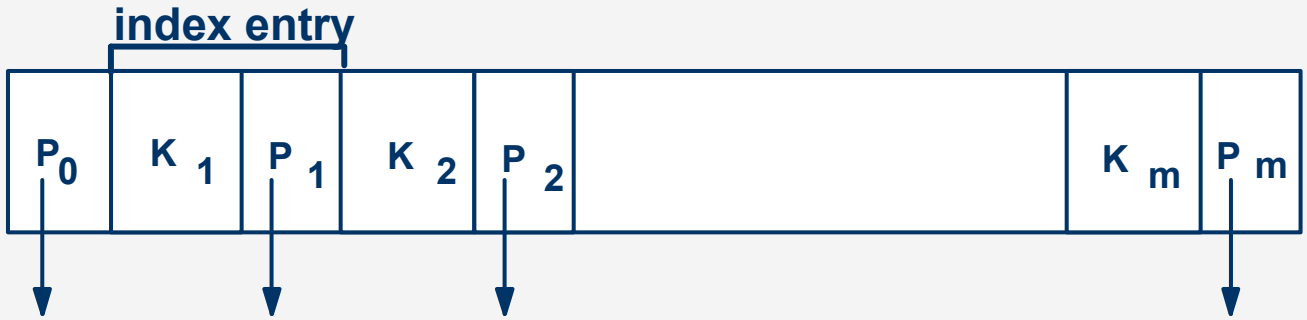
Căutări pe intervale

- ``*Returnează toţi studenţii cu* **grade** > *8.0*”
  - Dacă datele sunt stocate într-un fişier ordonat, prin căutare binară se detectează primul student, şi apoi sunt citite toate înregistrările următoare.
  - Costul căutarii/întreţinerii poate fi ridicat
- Idee: creare fisier `index’

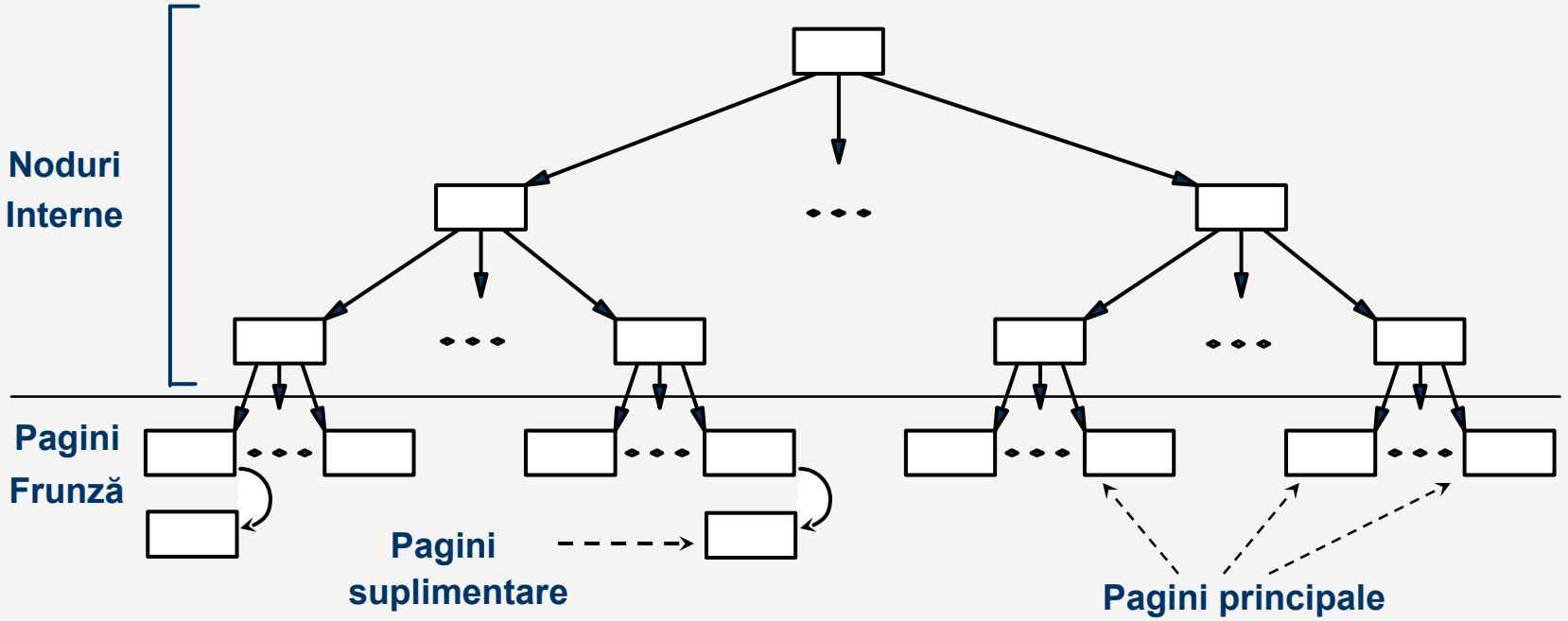


☛ *Cautarea binară se realizeaza pe fisierul index (mai mic)!*

ISAM (*Indexed Sequential Access Method*)



- Fisierul index poate fi mare, dar aceeași idee se poate aplica repetitiv



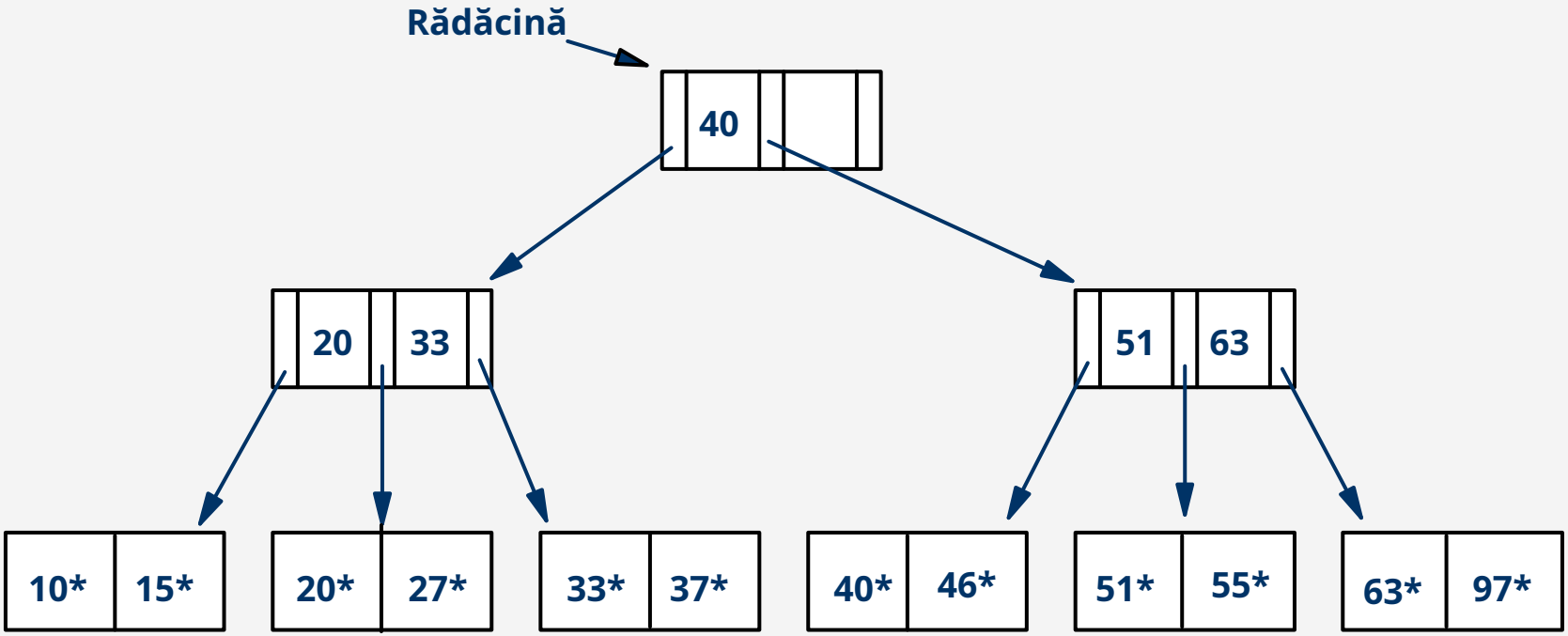
Observații asupra ISAM

- *Creare fisier*: paginile (de date) frunză sunt alocate secvential, sortate dupa cheia de căutare; apoi se memorează paginile index, apoi spațiul pentru paginile suplimentare.
- *Intrări index* : <val cheie căutare, id pagină>; vor `direcționa' căutarea către *paginile de date* (frunze).
- *Căutare*: Comparări ale cheii pornind de la rădăcina spre frunze. Cost $\log_F N$; F = număr intrări/pg index, N = numărul paginilor frunză
- *Inserare*: Căutare pagină frunză și adăugare înregistrare.
- *Ștergere*: Găsire și eliminare înregistrare din frunză; dacă e cazul, se dealocă o pagina suplimentară (*overflow*)

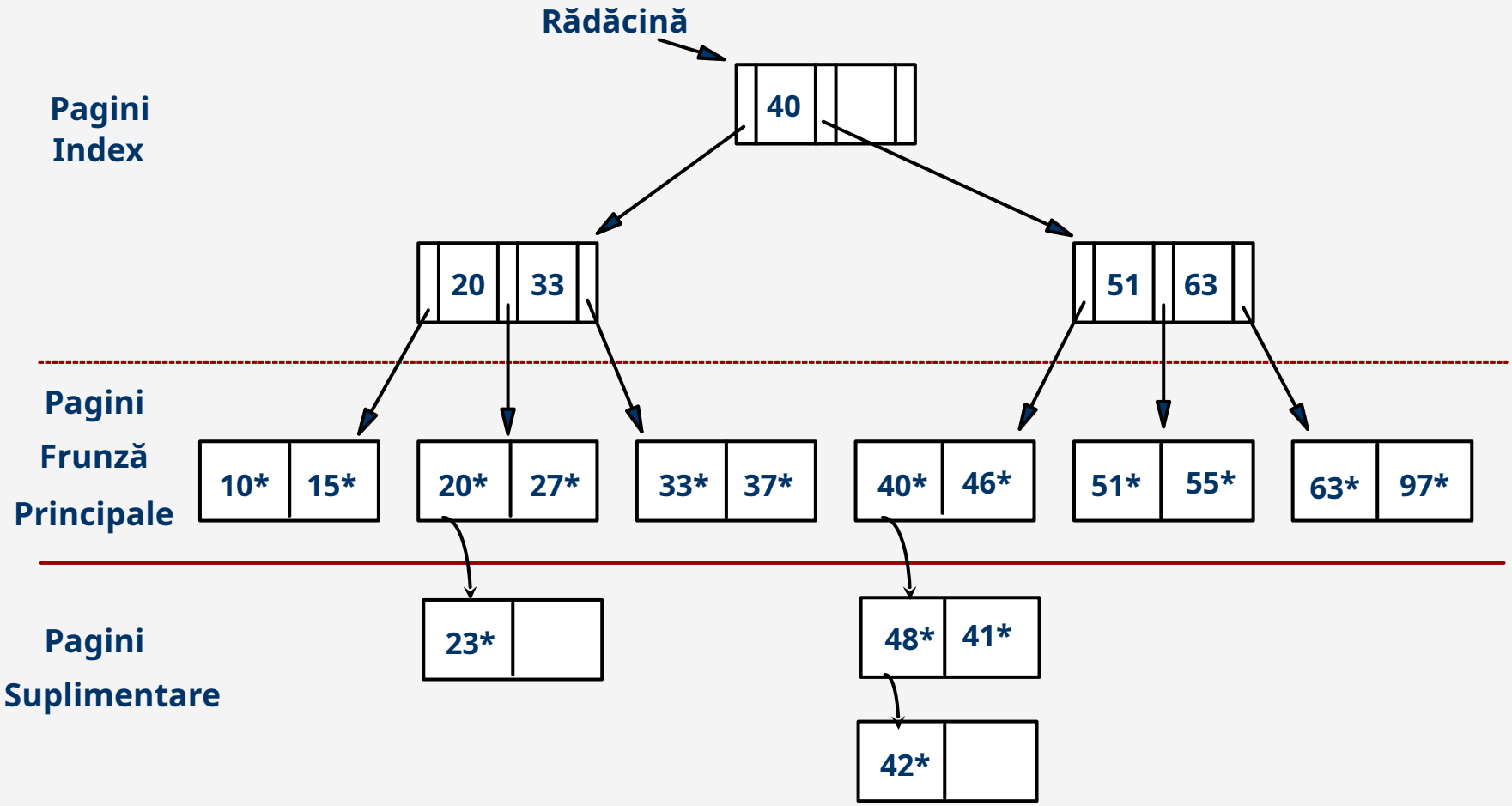
Structura arborescenta statica:
inserarea/ștergerea
afectează doar frunzele!

Exemplu arbore ISAM

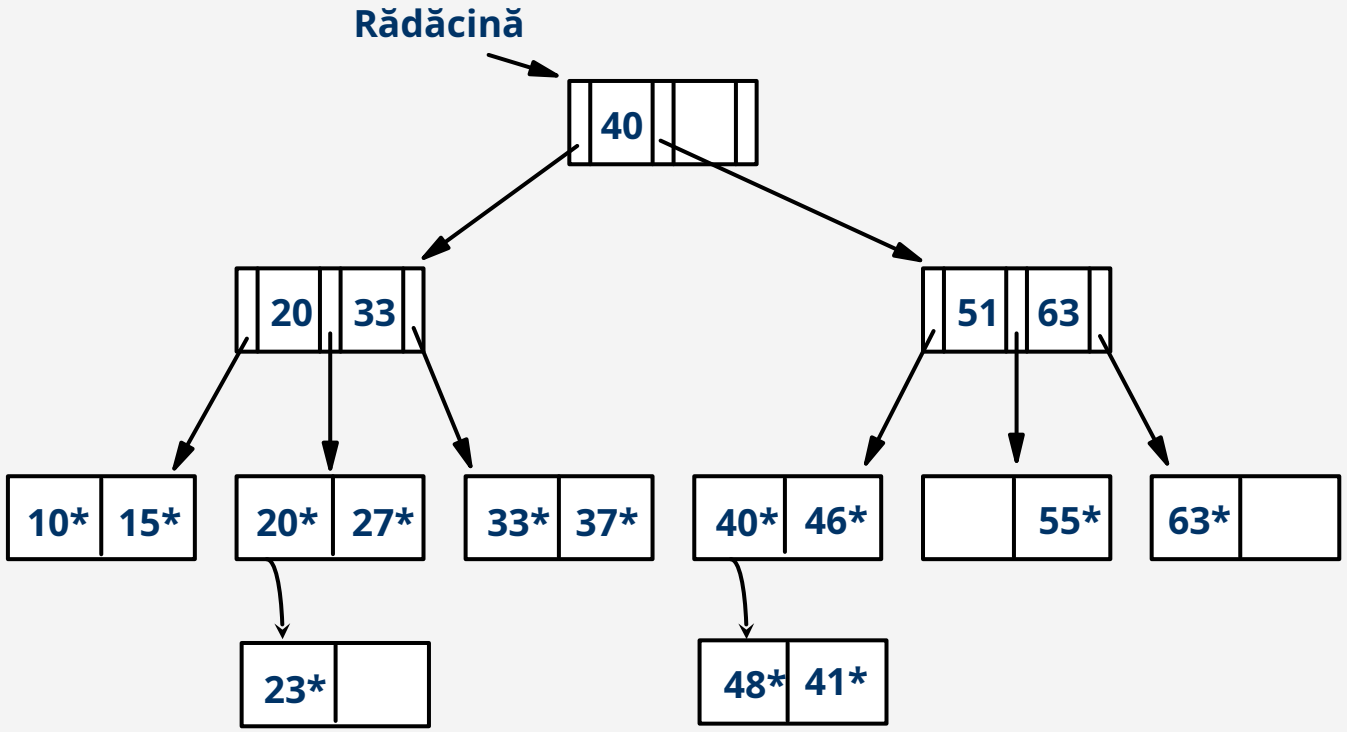
- Fiecare nod poate stoca 2 inregistrări;



După inserare 23*, 48*, 41*, 42* ...



… apoi ștergere 42*, 51*, 97*



☛ *Obs. 51* apare la nivelul indexului, dar nu mai există în frunze*

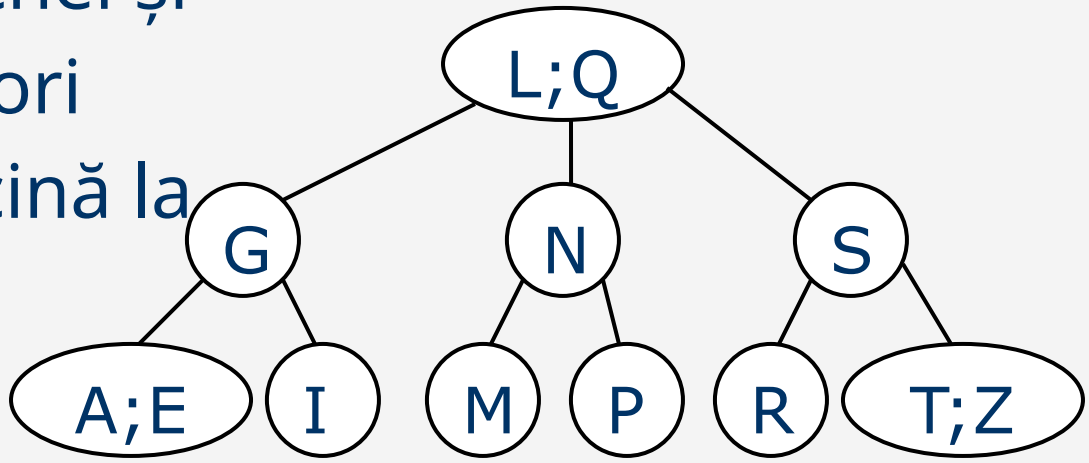
Avantaje & dezavantaje ale ISAM

- Se dezechilibrează - în urma a multor inserări & ștergeri (→ timp de căutare neuniform)
- Înregistrările din paginile de rezervă nu sunt sortate de obicei (dar ar putea fi)
- Inserări & ștergeri rapide (fără echilibrări de arbore)
- Acces concurent îmbunătățit (nodurile arborelui nu sunt blocate nicodată)

potrivite doar pentru tabele al căror conținut se modifică rar

Organizare sub formă de Arbore-B

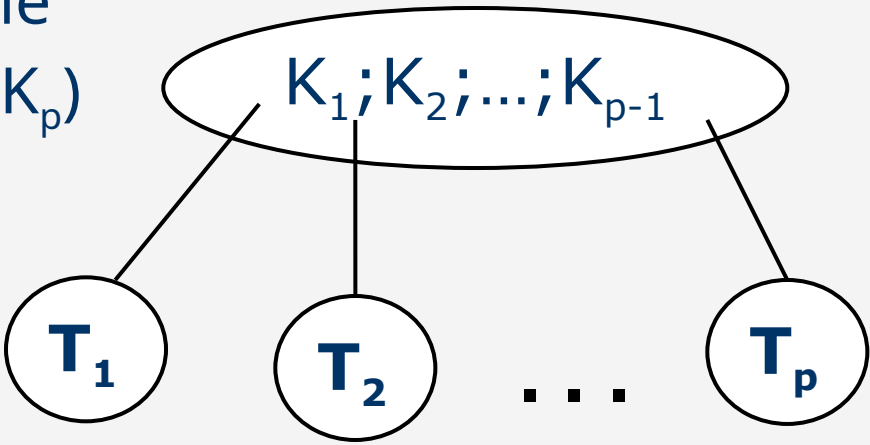
- Cea mai populară metodă de organizare a indecșilor în baze de date
- "B" vine de la "*balanced*" sau "*broad*"
- Arbore-B – arbore ordonat; un nod are mai mulți sub-arbori
- Un nod conține chei și pointeri la sub-arbori
- Căile de la rădăcină la frunze au aceeași lungime



Proprietăți Arbori-B

- Arbore-B de ordin *m*:
  - Dacă nu e frunză, rădăcina va avea mereu cel puțin 2 subarbori
  - Fiecare nod intern are cel puțin [*m/2*] sub-arbori (mai puțin rădăcina)
  - Fiecare nod intern are cel mult *m* sub-arbori
  - Toate frunzele sunt la același nivel
  - Un nod cu p sub-arbori conține

    p-1 chei ordonate (K_1, K_2, ... K_p)
    - T_1 conține valori $<K_1$
    - T_i conține valori între K_{i-1} si K_i
    - T_p conține valori $> K_p$

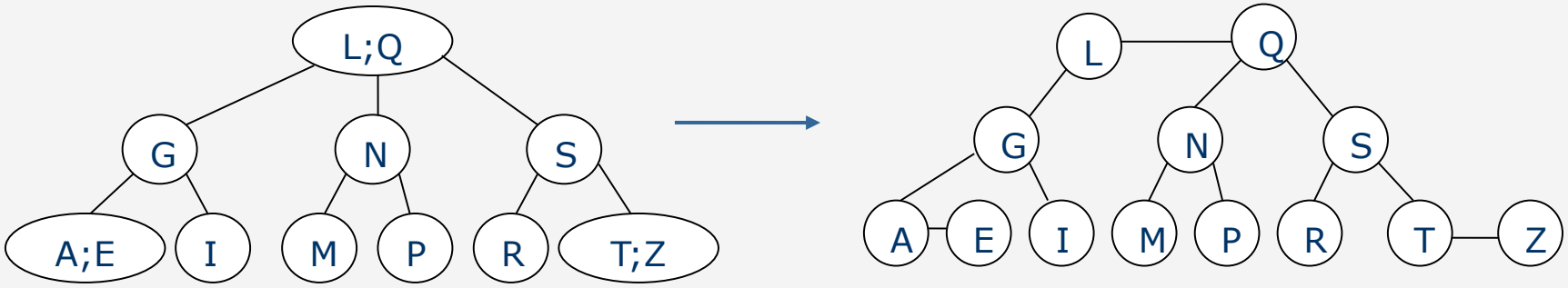


Organizarea în memorie a Arborilor-B

- Ca arbore binar

K	Data	$Pointer_{Left}$	$Pointer_{Right/H}$

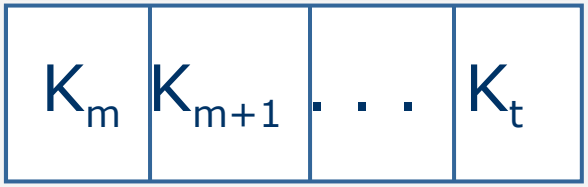
  - *Pointer*$_{Left}$
    - referă prima cheie a sub-arborelui stâng din Arborele-B
  - *Pointer*$_{Right/H}$
    - referă vecinul din dreapta în nodul Arborelui-B
    - referă prima cheie a sub-arborelui drept din Arborele-B (dacă e ultima valoare a cheii în nodul Arborelui B)
  - Flag adițional / utilizare semn pentru *pointer* dreapta



Organizarea în memorie a Arborilor-B

- Alocare memorie pentru a stoca m-1 chei per no
  - N – număr de chei stocate
  - K_i – valoarea cheii, AD_i – adresa înregistrării
  - $Point_i$ – referă un sub-arbore

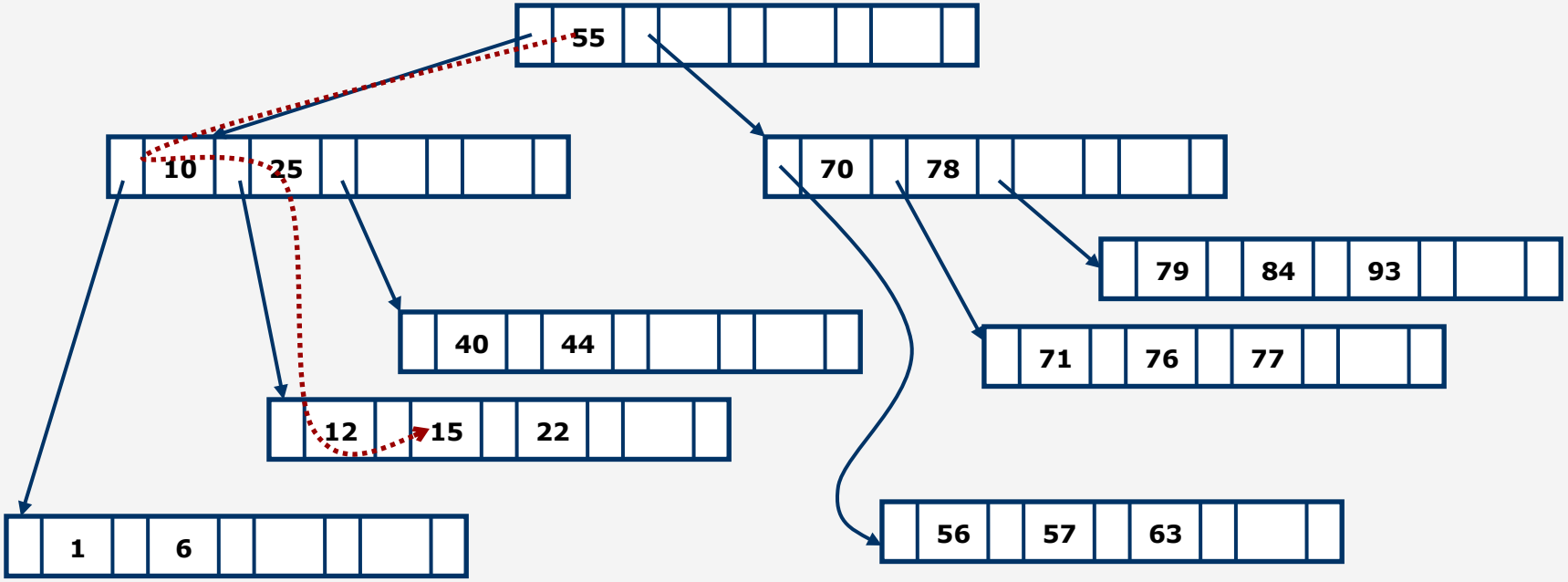
  - Utilizarea spațiului de memoriei pentru *pointeri* la stocarea cheilor în frunze



    - m/2 ⫟ N ⫟ m-1 – pentru noduri interne
    - m/2 ⫟ N ⫟ t – pentru noduri terminale
    - flag adițional/ semn pentru N

Căutarea într-un Arbore-B

- Exemplu: căutare ’15’



Inserarea înregistrărilor în Arbore-B

- Pași de inserare:
  - localizare nod un cheia trebuie să fie inserată
  - se inserează noua cheie
  - se aplică o procedură de *echilibrare* în cazul în care este depășit numărul maxim de chei ce pot fi stocate în nod

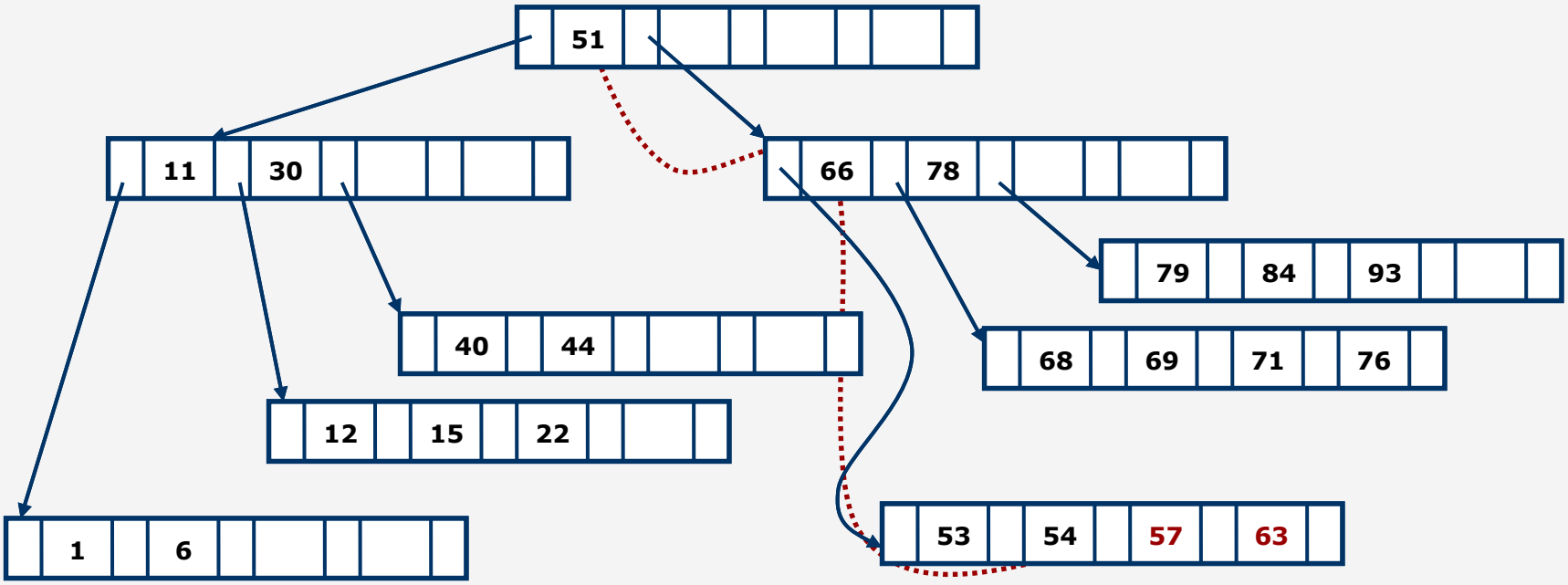
Inserarea înregistrărilor în Arbore-B

Algoritmul procedurii de inserare

1. Localizare nod pentru inserare
2. Inserare cheie
3. Dacă nodul e plin (dimensiunea e depășită):
   A) Se crează un nou nod în care se mută cheile mai mari decât valoarea cheii mediane
   B) Se inserează cheia mediană în nodul părinte
   C) *Pointerul* din dreapta cheii va referi noul nod, iar cel din stanga referă vechiul nod ce conține valorile mai mici
4. Daca și nodul părinte e plin:
   A) Dacă nodul părinte e radacină atunci se crează o radacină nouă
   B) Se repetă pasul 3 pentru nodul părinte

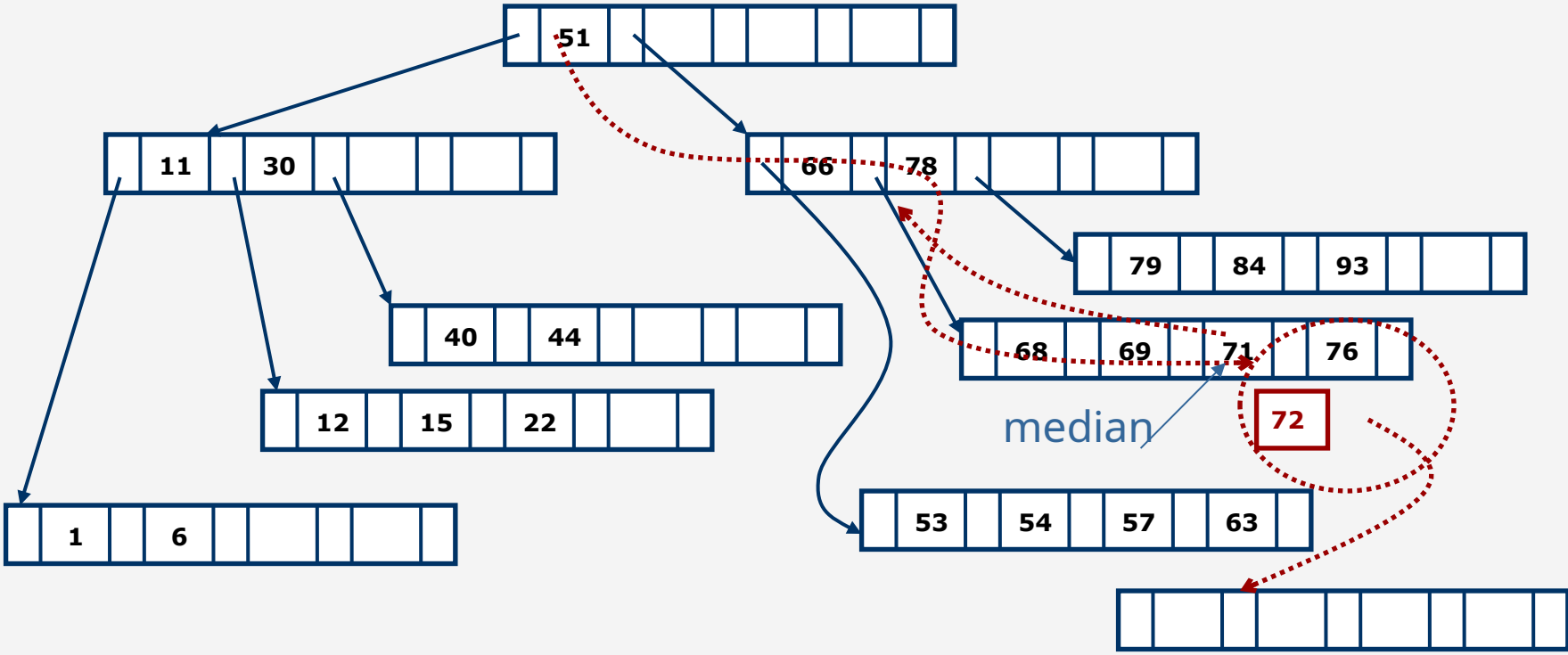
Inserarea înregistrărilor în Arbore-B

- Inserare înregistrare cu cheia ’57’



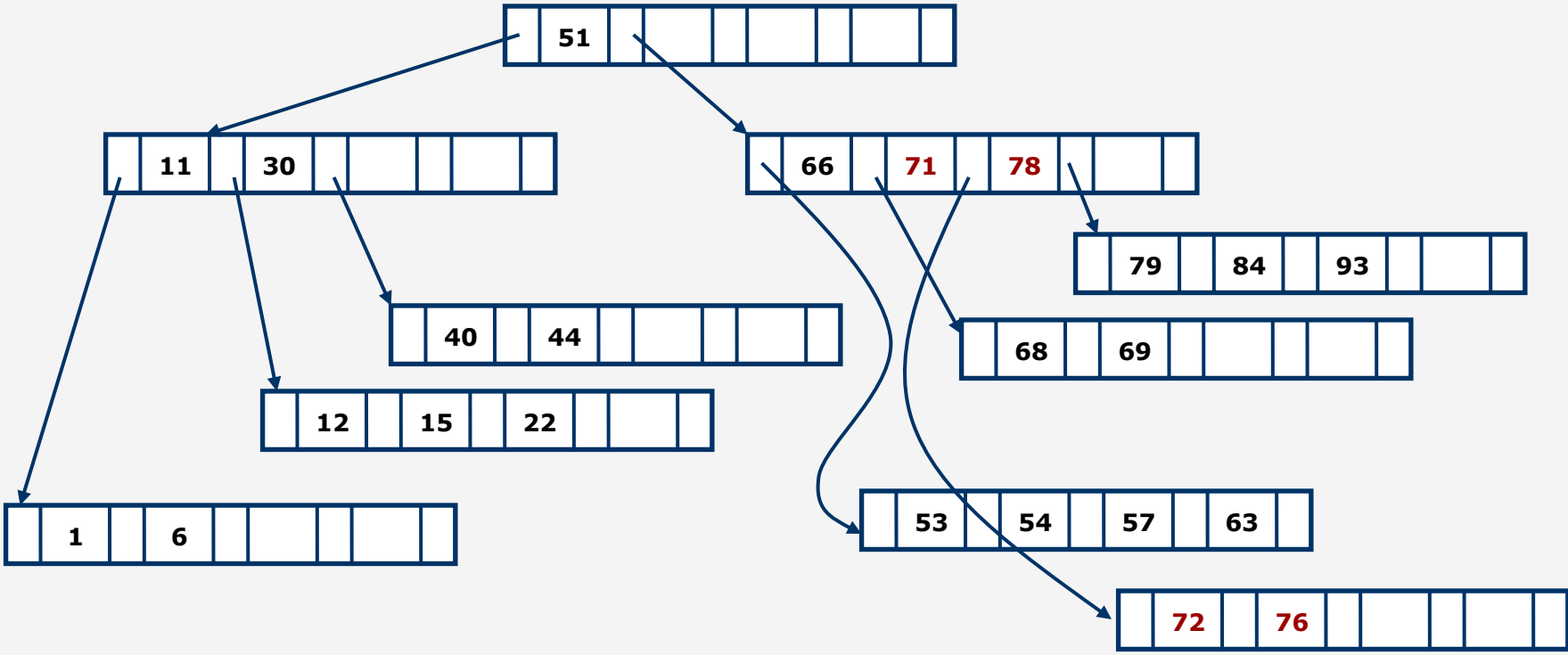
Inserarea înregistrărilor în Arbore-B

- … apoi se inserează înregistrarea cu cheia ’72’



Inserarea înregistrărilor în Arbore-B

- … apoi se inserează înregistrarea cu cheia ’72’



Ștergerea înregistrărilor în Arbore-B

- Pași de ștergere:
  - se indentifică nodul ce conține valoarea ce trebuie ștearsă
  - dacă e un nod intern, *se transferă* o valoare din frunze
  - în caz de *subdimensionare*, se realizează o *redistribuire* sau o *concatenare*

Ștergerea înregistrărilor în Arbore-B

Algoritm de ștergere

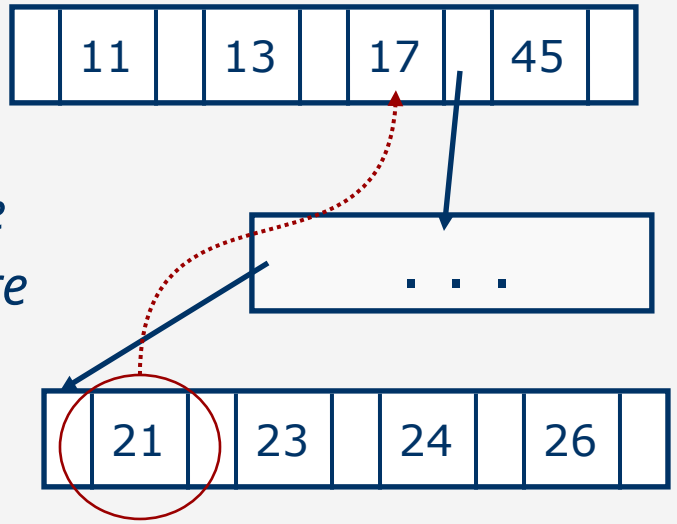
1. Se caută valoarea ce trebuie ștearsă.

Dacă se află într-un nod intern
se înlocuiește cu valoarea *vecină mai mare*

(adică cu *cea mai din stânga valoare*

a *celei mai din stânga frunze*

a *subarborelui drept*)

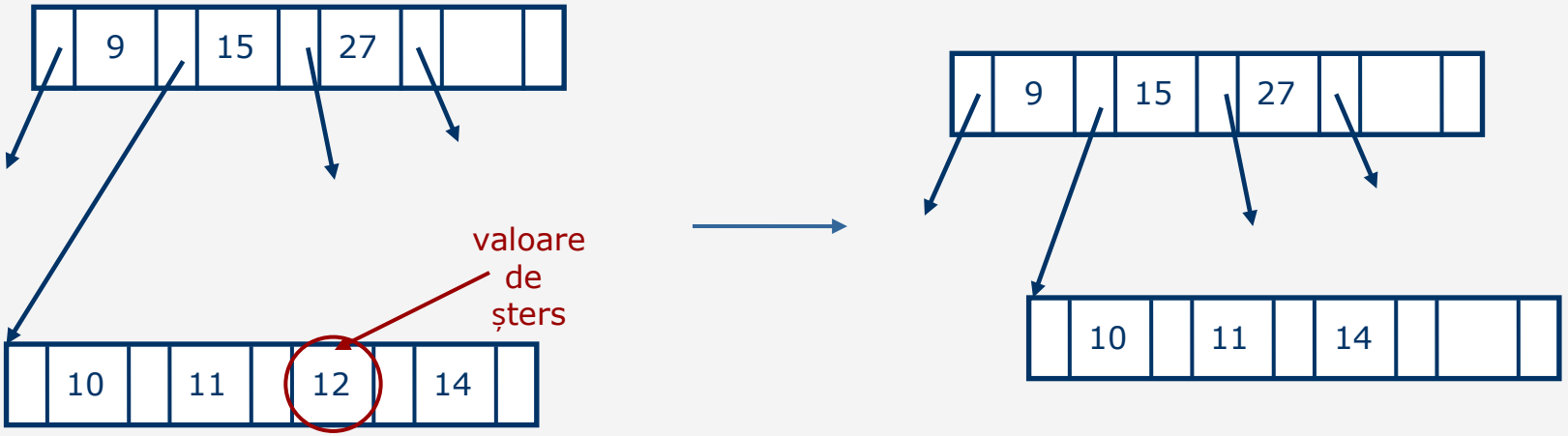


Ștergerea înregistrărilor în Arbore-B

2. Se repetă acest pas până se ajunge în cazurile A) sau B)

A) Dacă nodul ce conține valoarea de șters este rădăcina sau numărul valorilor rămase în nod este $\geq \lceil m/2 \rceil$:

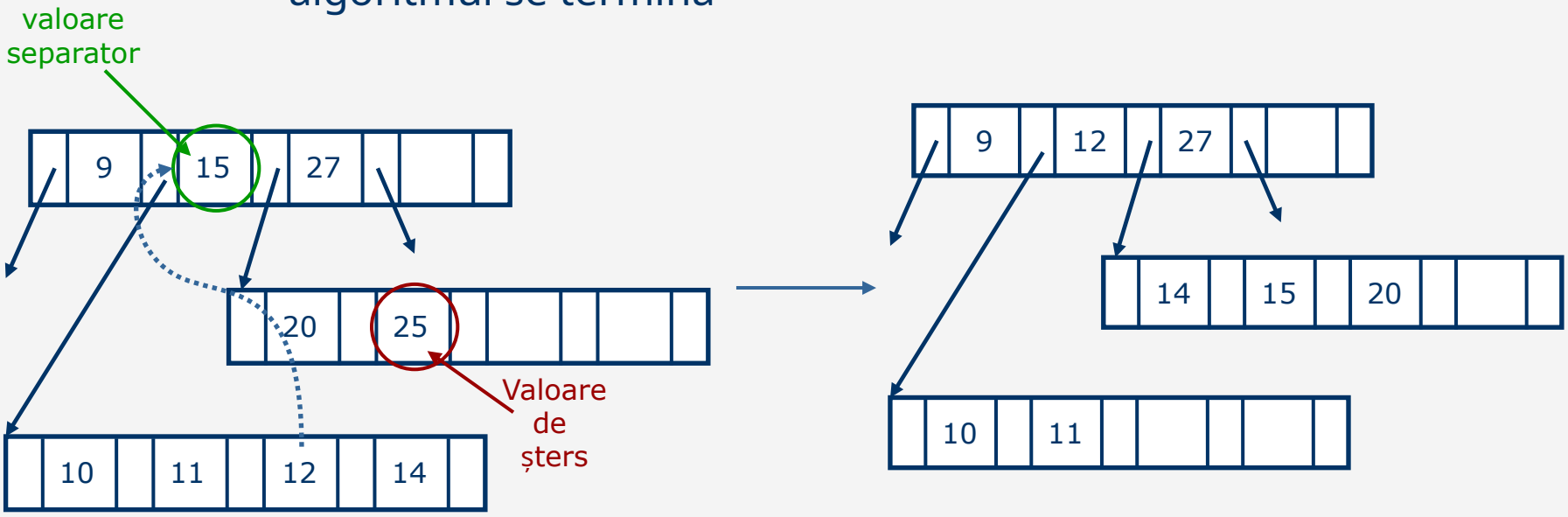
- se elimină valoarea
- se *re-aranjează valorile și pointerii* din nod
- se termină algoritmul



Ștergerea înregistrărilor în Arbore-B

B) Dacă numărul valorilor rămase în nod este < [m/2] unul dintre nodurile vecine conține > [m/2] valori ⎯ *redistribuire*

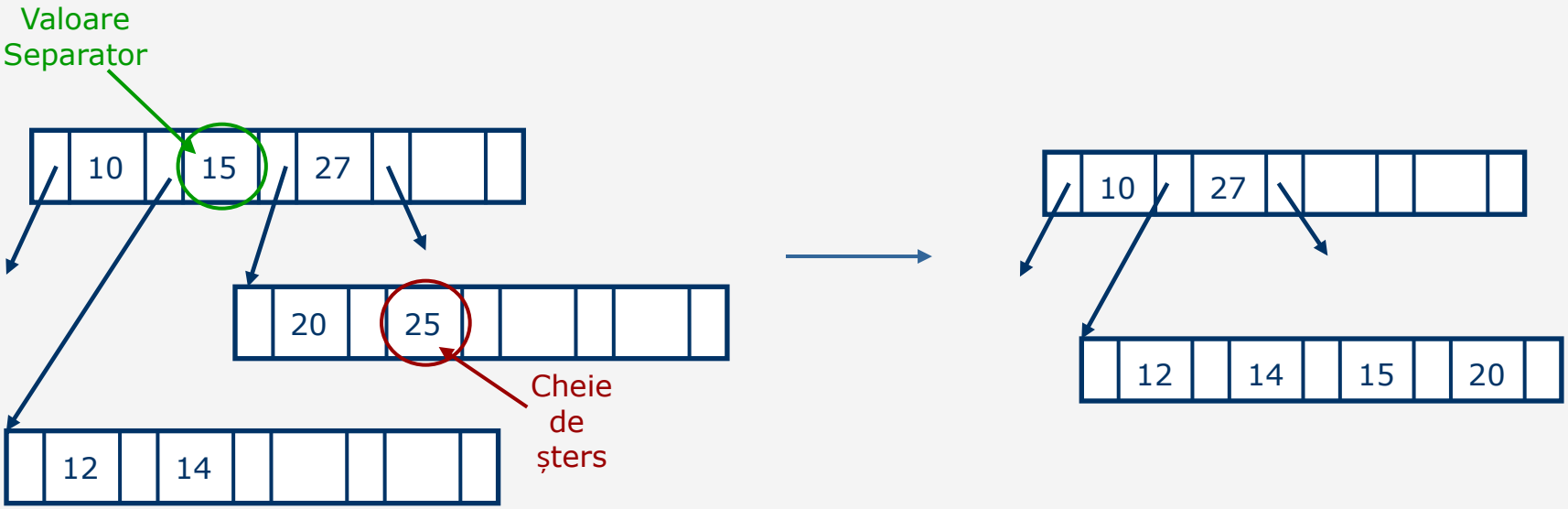
- se ordonează valorile din ambele noduri + valoarea separator din părinte
- se alege valoarea mediană și se adaugă la nodul părinte, iar celelalte valori se inserează în nodul stâng, respectiv drept
- algoritmul se termină



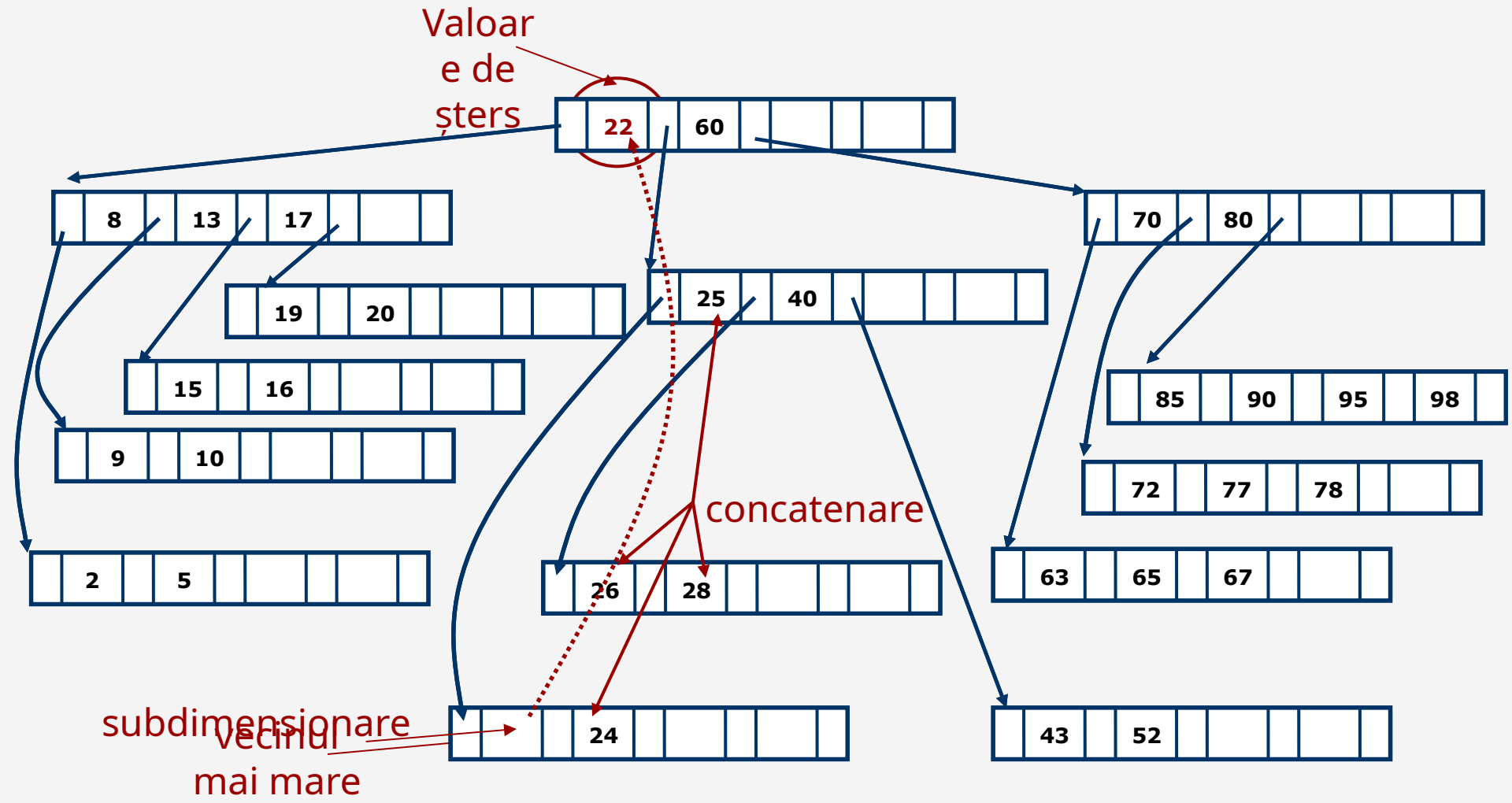
Ștergerea înregistrărilor în Arbore-B

C) Dacă suma valorilor din nodul din care s-a eliminat valoarea și a valorilor unuia din vecini este < m ⇒ *concatenare*

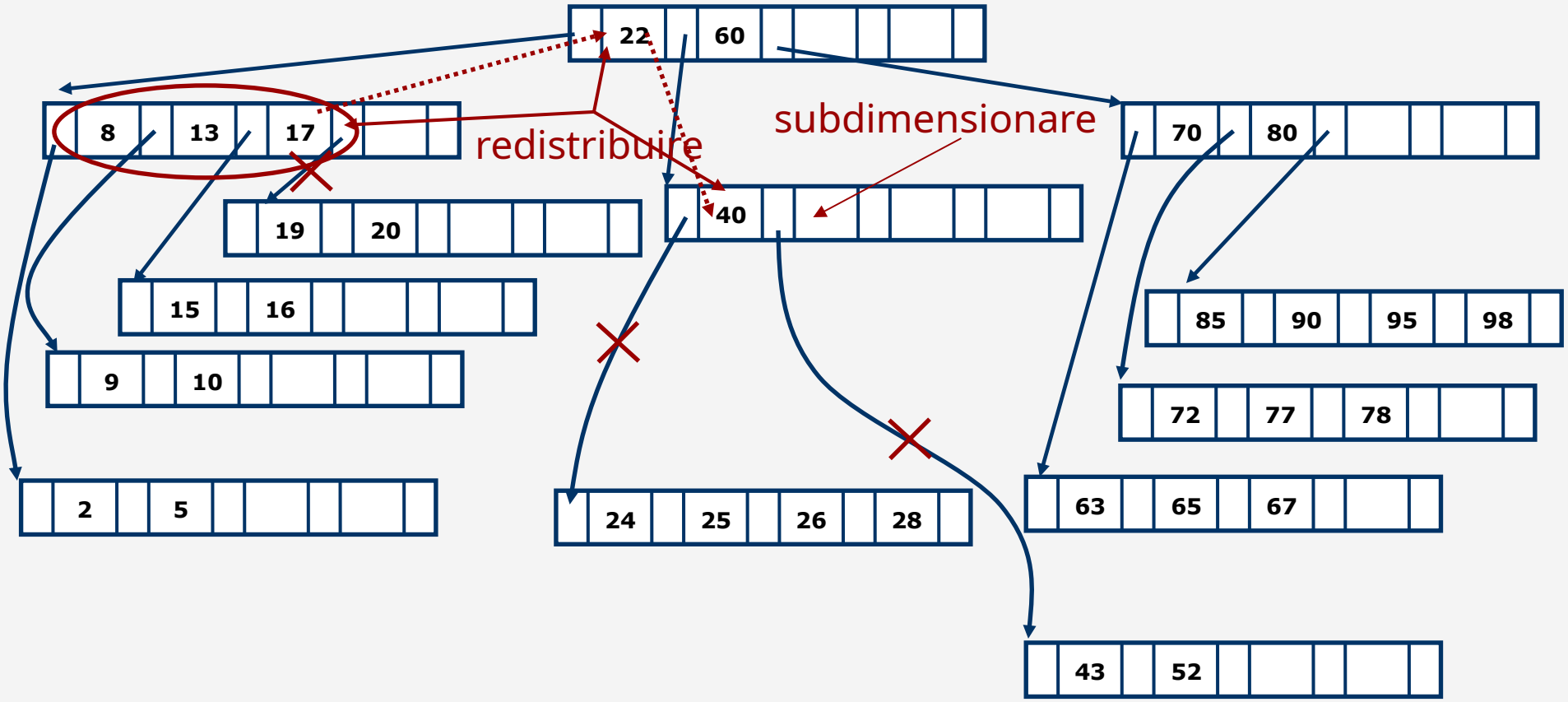
- se inserează valorile ambelor noduri + valoarea separator din nodul părinte într-un singur nod
- se repetă pasul **2.** pentru nodul părinte (din care s-a eliminat valoarea separator)
- dacă nodul părinte este rădăcina și nu mai conține valori ⇒ nodul curent devine rădăcina



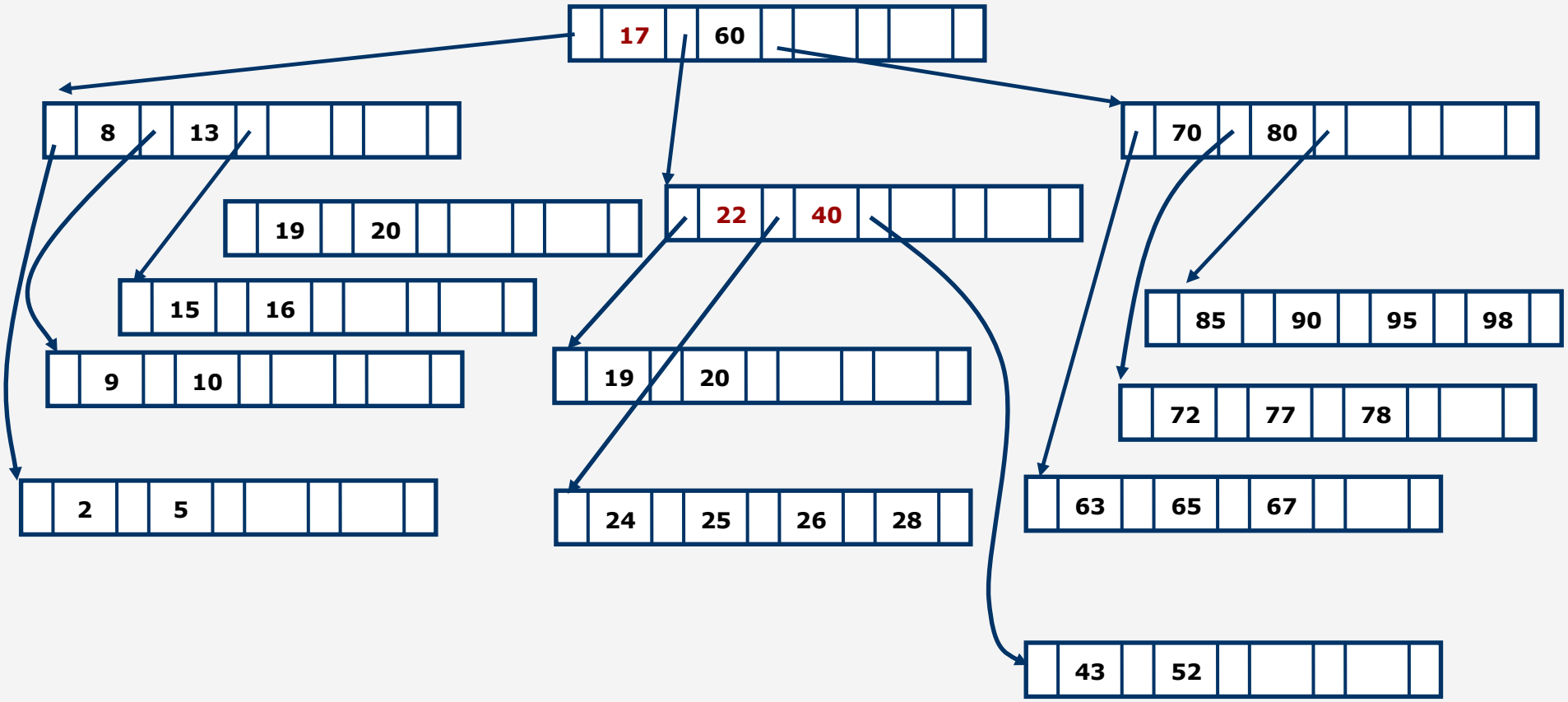
Ștergerea înregistrărilor în Arbore-B



Ștergerea înregistrărilor în Arbore-B

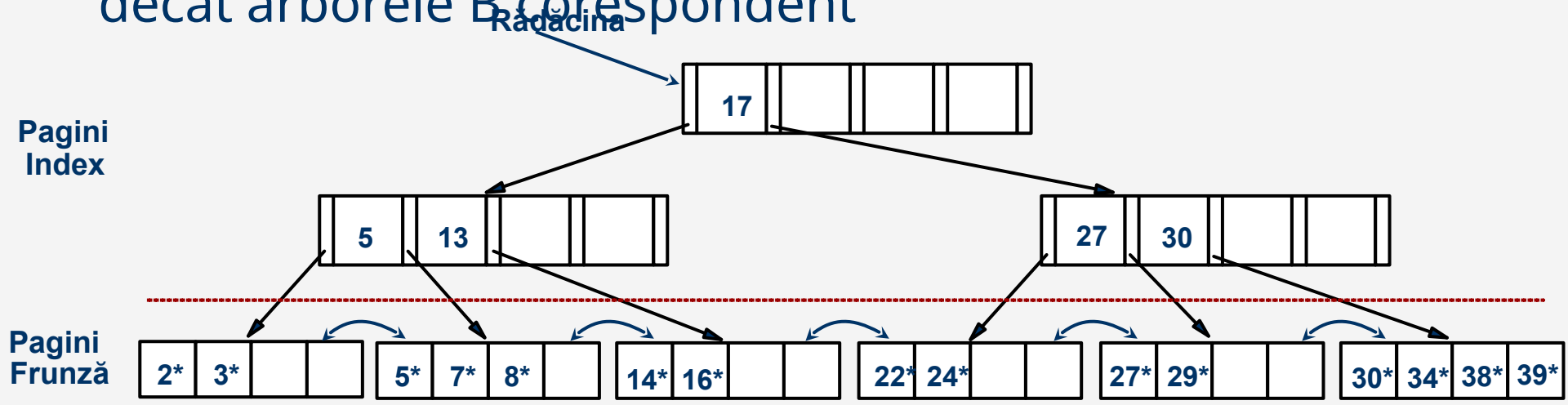


Ștergerea înregistrărilor în Arbore-B



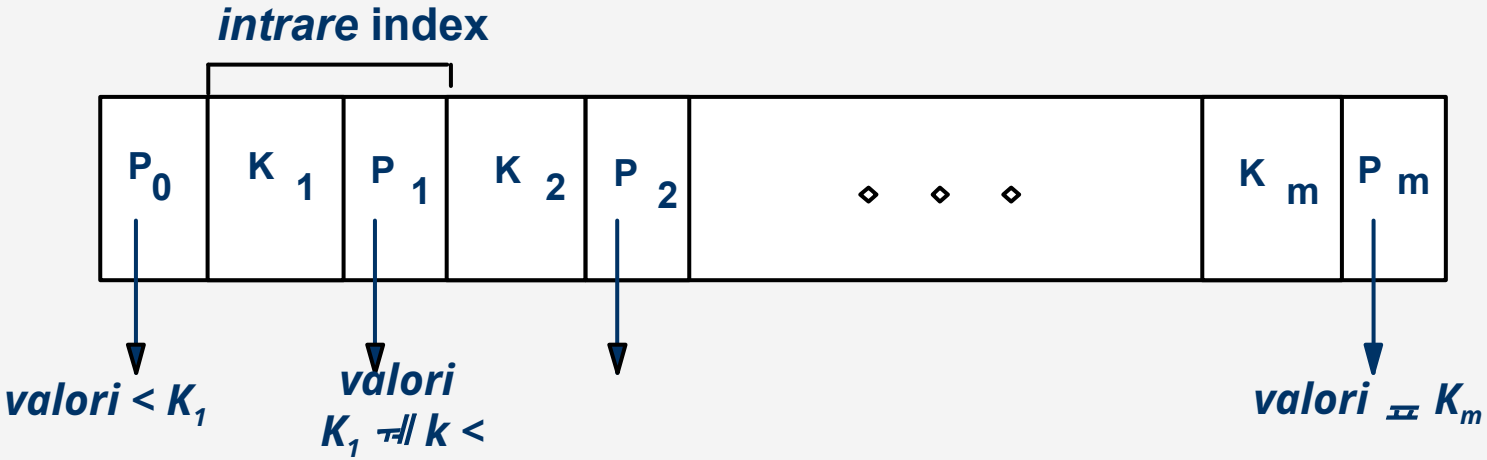
Arbori B+

- Combinație între Arbori-B și ISAM:
  - Căutarea pornește de la rădăcină, și va fi direcționată prin comparații către o frunză
  - Într-un Arbore B+ toți pointerii către înregistrări din tabele se află doar la nivelul nodurilor frunză
- Un arbore B+ poate avea mai puține nivele (sau o capacitate mai mare pentru stocarea cheilor de căutare) decât arborele B corespondent

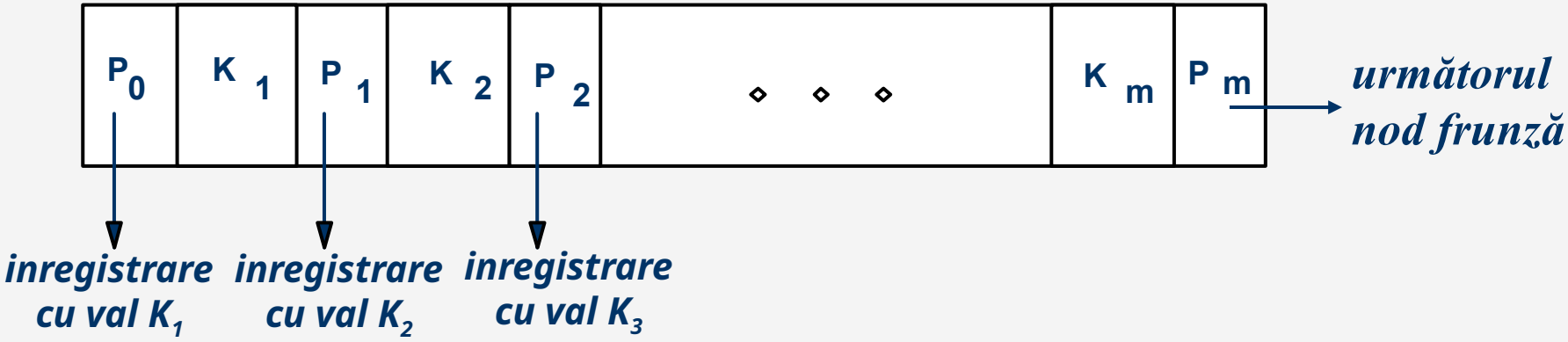


Structura nodurilor

Noduri interne



Noduri frunză



Arborii B+ în practică

- Ordin tipic: 200. Factor de acoperire: 67%.
  - În medie, (nr de valori)/(nr pagini de index) = 133
- Capacităţi tipice:
  - Înălțime 4: 133^4 = 312,900,700 înregistrări
  - Înălțime 3: 133^3 = 2,352,637 înregistrări
- În general poate fi memorat în *buffer*-ul din memoria internă:
  - Nivel 1 = 1 pagină = 8 Kbytes
  - Nivel 2 = 133 pagini = 1 Mbyte
  - Nivel 3 = 17,689 pagini = 133 MBytes

Avantaje & Dezavantaje ale Arborilor-B+

- Indecșii rămân echilibrați → timp de căutare uniform
- Rareori sunt mai mult de 3 - 5 nivele → primele nivele sunt păstrate în RAM astfel încât o căutare va necesita doar 2 sau 3 I/O (citiri/scrieri).
- În general nodurile au o ocupare de 67% (deci se folosește cu 50% mai mult spațiu decât este necesar)
- Datorită versatilităţii sale este cel mai utilizat mod de a structura indecşii în SGBD-uri relaționale. E una dintre cele mai optimizate componente ale unui SGBD.
- Arborii B+ pot fi utilizați pentru indecși *clustered*, rari (dacă tabela e sortată) sau pentru indecși *un*

Bulk Loading

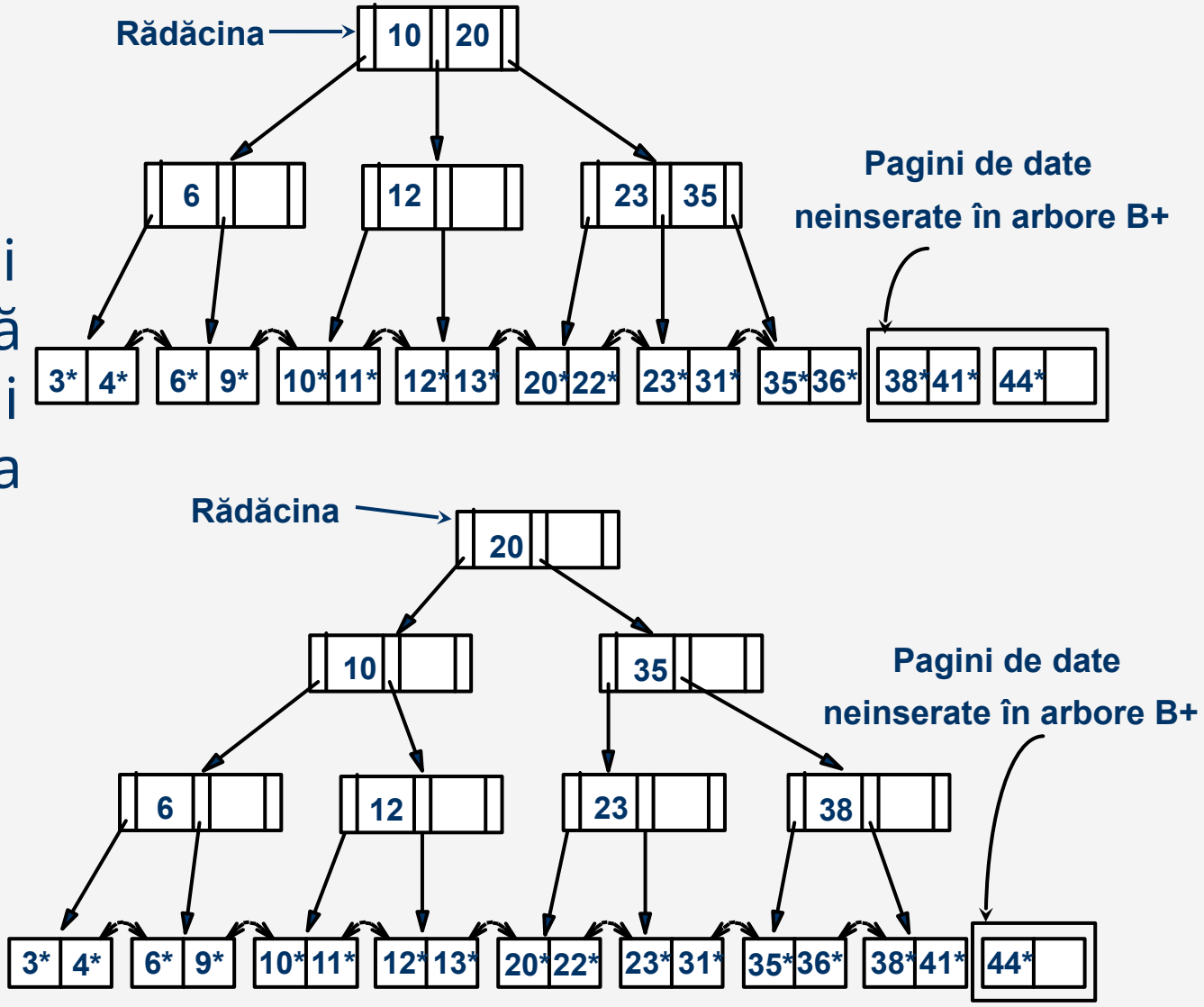
- În cazul unei colecţii mari de inregistrări crearea unui arbore B+ prin inserarea repetată a fiecarei înregistrări este lentă.
- Procedura de *Bulk Loading* este mult mai eficientă.

Initializare: Se sorteaza toate inregistrarile tabelei și se inserează un pointer către prima pagină într-o nouă pagină a arborelui (rădăcina).

Bulk Loading

Intrările din index sunt întotdeauna inserate în cea mai din dreapta pagină deasupra ultimului nivel. Când pagina devine plină se împarte în două.

Mult mai rapid decât prin inserări repetate!



Sumar *Bulk Loading*

- Opțiune 1: inserări multiple.
  - Lent.
  - Frunzele nu sunt stocate secvențial.

- Opțiune 2: *Bulk Loading*
  - Indexul poate fi utilizat concurent.
  - Mai puține citiri/scrieri (I/Os) în timpul construcției
  - Frunzele sunt stocate secvențial (și înlănțuite).
  - Se poate controla “gradul de umplere” a unei pagini

Arbore B+ Prefix (compresie de cheie)

- Conduce la creșterea numărului de valori stocate într-un nod
- Valorile din index sunt folosite doar pentru a direcționa "traficul" comparațiilor, și de aceea pot fi comprimate
  - Ex. Dacă avem intrări adiacente în index cu următoarele valori pentru cheia de căutare *Dan Yogurt*, *David Smith* și *Demy Moore*, putem abrevia *David Smith* cu *Dav*. (și de asemenea celelelalte chei ...)
- Inserările/ștergerile se pot modifica corespunzător.