



ARBORI

Șl. Dr. Ing. Șerban Radu

Departamentul de Calculatoare

Facultatea de Automatică și Calculatoare



Necesitatea structurii de arbore

- Un arbore combină avantajele oferite de alte două structuri: tablourile și listele înlanțuite
- Arborii permit, pe de o parte, executarea unor căutări rapide, la fel ca și tablourile ordonate, iar pe de altă parte, inserarea și ștergerea rapidă a elementelor, la fel ca o listă înlanțuită



Operații cu un tablou ordonat

- Să presupunem că avem un tablou ale cărui elemente sunt ordonate
- Un astfel de tablou permite căutarea rapidă a unui anumit element, utilizând metoda căutării binare
- De asemenea, parcurgerea unui tablou ordonat este o operație rapidă, fiecare element fiind vizitat în ordine

Inserarea într-un tablou ordonat este lentă

- Pe de altă parte, dacă dorim să inserăm un nou element într-un tablou ordonat, trebuie mai întâi să găsim locul unde vom insera elementul, după care să deplasăm toate elementele mai mari cu o poziție spre dreapta, pentru a-i face loc
- Ștergerea unui element presupune aceeași operație de deplasare multiplă, fiind deci la fel de lentă



Operații cu o listă înlănțuită

- Operațiile de inserare și ștergere se efectuează repede în cazul utilizării listelor înlănțuite
- Acestea se reduc la modificarea pointerilor și se efectuează într-un timp constant
- Căutarea unui anumit element într-o listă este o operație lentă

Căutarea într-o listă înlănțuită este lentă

- Trebuie să pornim de la începutul listei și să vizităm fiecare element, până când îl găsim pe cel dorit
- În cazul mediu, căutarea presupune vizitarea a $N/2$ elemente și compararea cheii fiecăruia cu valoarea dorită

Căutarea într-o listă înlănțuită este lentă

- O soluție posibilă, care să permită căutarea unui element într-un timp mai scurt, este de a utiliza o listă înlănțuită ordonată, în care elementele sunt așezate în ordine
- Dar căutarea presupune parcurgerea elementelor, începând cu primul din listă și vizitând în ordine toate elementele, deoarece singura modalitate de a avea acces la un element este de a urma lanțul de legături care conduc la el



Arbori

- Arborii, care reprezintă una dintre cele mai interesante structuri de date, oferă ambele facilități:
 - Inserarea și ștergerea rapidă – din cazul listelor înlănțuite
 - Căutarea rapidă – din cazul tablourilor



Arbori

- Arborii sunt utilizați în descrieri ierarhice, pentru modelarea unor obiecte sau fenomene
- Descrierea ierarhică se referă la descompunerea unei entități în subentități, fiecare dintre ele putând fi caracterizate printr-un set de attribute sau însușiri



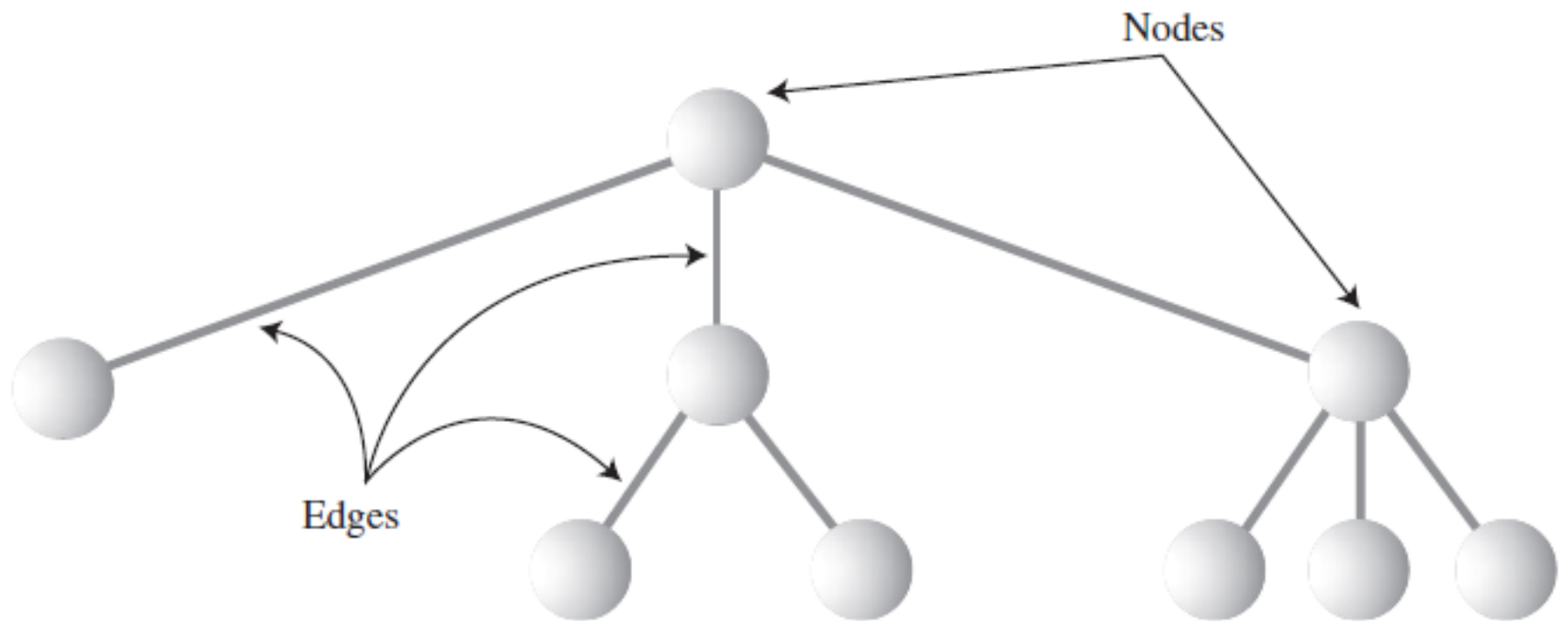
Arbori

- Utilizând o astfel de descriere, se realizează o ierarhizare a părților unei entități pe unul sau mai multe niveluri
- Organizarea ierarhică este întâlnită în diverse domenii: organizarea administrativă a unei companii, planificarea meciurilor în cadrul unui turneu sportiv, evaluarea unor expresii aritmetice



Ce este un arbore

- Un arbore constă din **noduri**, care sunt unite prin **arce**
- Vom reprezenta nodurile prin cercuri, iar arcele, prin linii care unesc cercurile



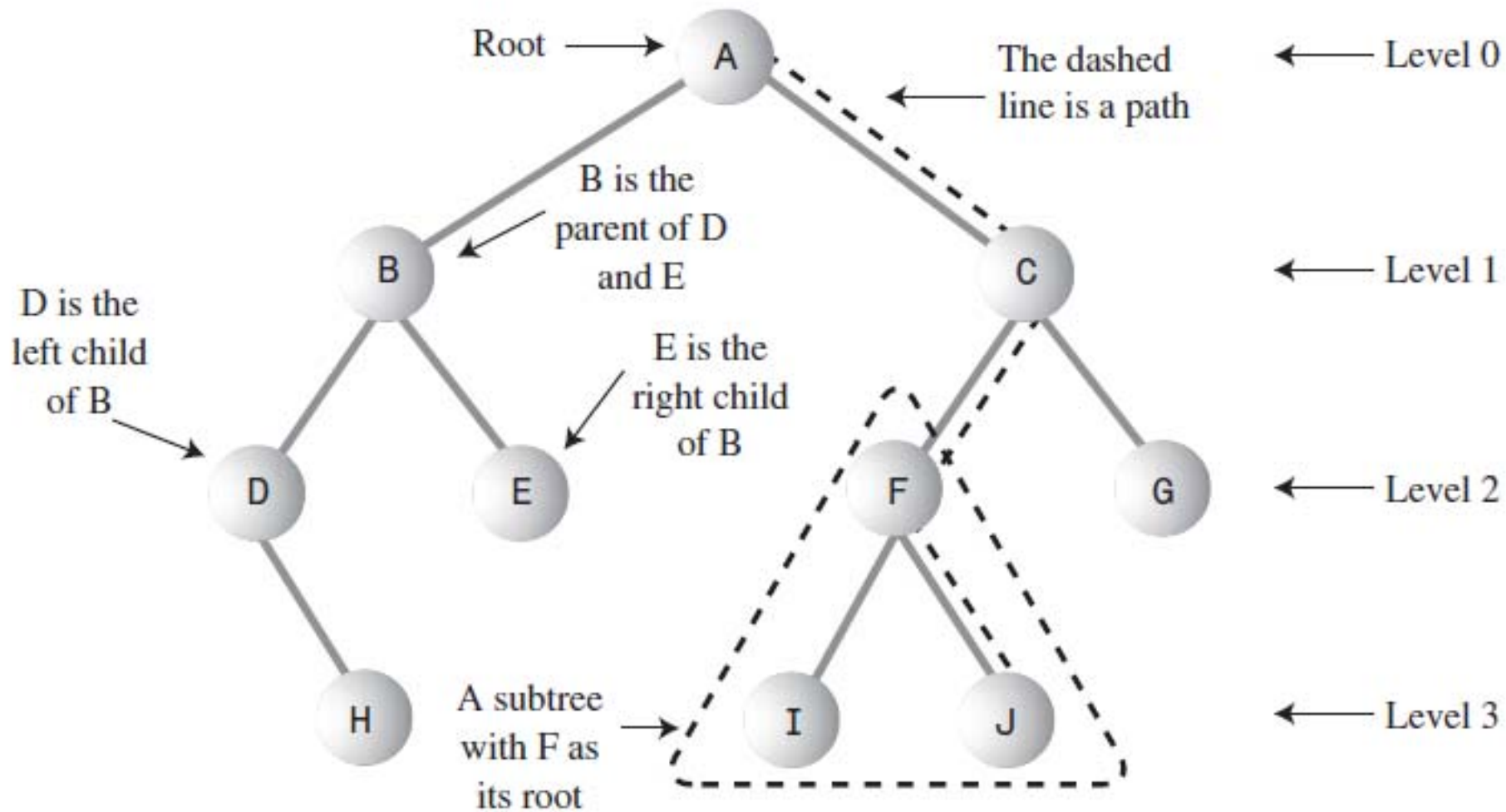


Noduri

- Nodurile reprezintă, de regulă, entități din lumea reală, cum ar fi valori numerice, persoane, părți ale unei mașini, rezervări de bilete de avion
- Nodurile sunt elemente obișnuite pe care le putem memora în orice altă structură de date

Arce

- Arcele dintre noduri reprezintă modul în care nodurile sunt conectate
- Este ușor și rapid să ajungem de la un nod la altul dacă acestea sunt conectate printr-un arc
- Arcele sunt reprezentate în C și C++ prin **pointeri**




H, E, I, J, and G are leaf nodes



Înălțimea unui arbore

- Valoarea **maximă** de pe nivelurile nodurilor terminale



Ordin (grad)

- Numărul de descendenți direcți ai unui nod reprezintă **ordinul** sau **gradul nodului**
- **Ordinul** sau **gradul arborelui** este valoarea maximă luată de gradul unui nod component al arborelui

Tipuri de arbori

- **Arbori binari**, în care fiecare nod dintr-un arbore are cel mult doi fii
- **Arbori multicăi**, în care nodurile pot avea mai mulți fii
 - Arbori 2-3-4
 - Arbori B
 - Arbori AVL (Adelson-Velskii și Landis)

Arbori binari

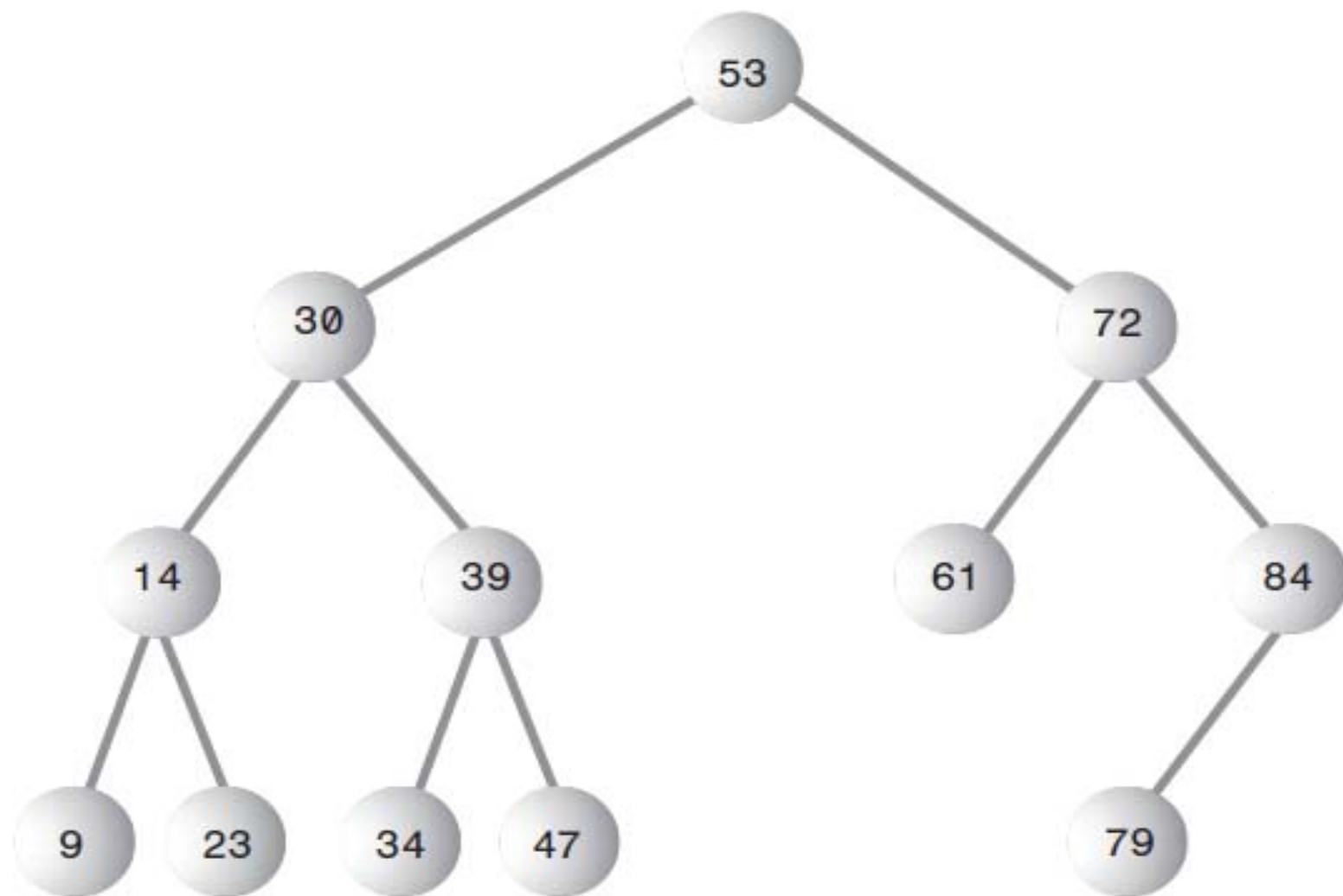
- Dacă orice nod din arbore poate avea cel mult doi fii, atunci arborele se numește **arbore binar**
- Cei doi fii ai unui nod se numesc **fiu stâng** și, respectiv, **fiu drept**, în funcție de poziția lor în reprezentarea grafică a arborelui
- Un nod care nu are niciun fiu se numește **frunză**

Arbori binari

- Între numărul N de noduri al unui arbore binar și înălțimea sa H există relațiile:
- $H \leq N \leq 2^H - 1$
- $\log_2 N \leq H \leq N$

Arbori binari de căutare

- Cheia fiului stâng trebuie să fie **mai mică** decât cea a părintelui, iar cheia fiului drept trebuie să fie **mai mare** sau egală cu cea a părintelui



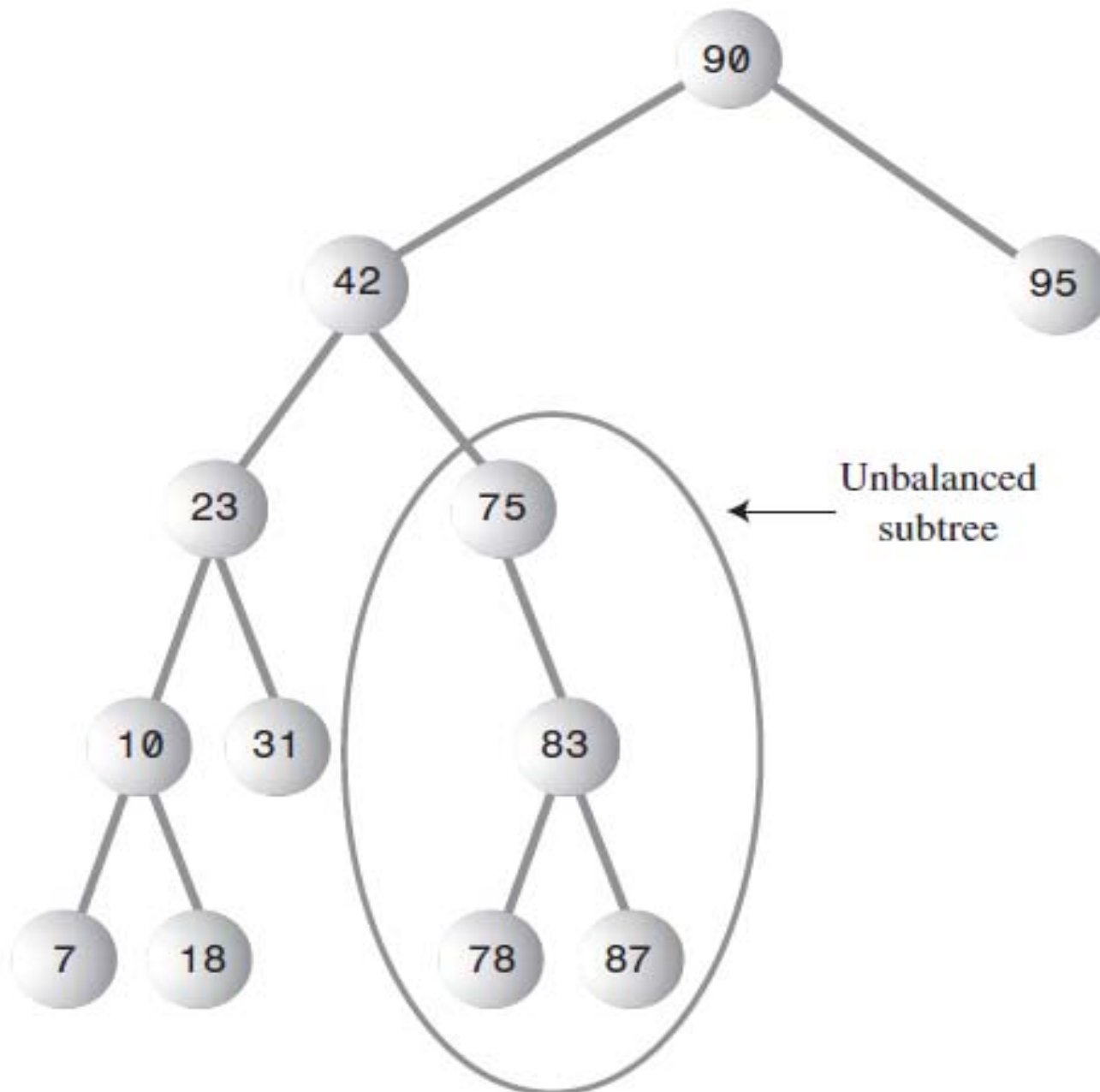
Reprezentarea unui arbore binar

- Se memorează nodurile la adrese oarecare din memorie, utilizând pointeri pentru a conecta fiecare nod cu fiii săi
- `typedef struct tnod {`
- `void * val; // valoare memorată în nod`
- `struct tnod * st; // succesori la stânga`
- `struct tnod * dr; // succesori la dreapta`
- `} tnod;`



Arbori neechilibrați

- Într-un arbore neechilibrat, majoritatea nodurilor sunt situate de o singură parte a rădăcinii, iar anumiți subarbori pot fi, la rândul lor, neechilibrați
- Dezechilibrarea arborilor poate proveni de la ordinea în care sunt inserate cheile





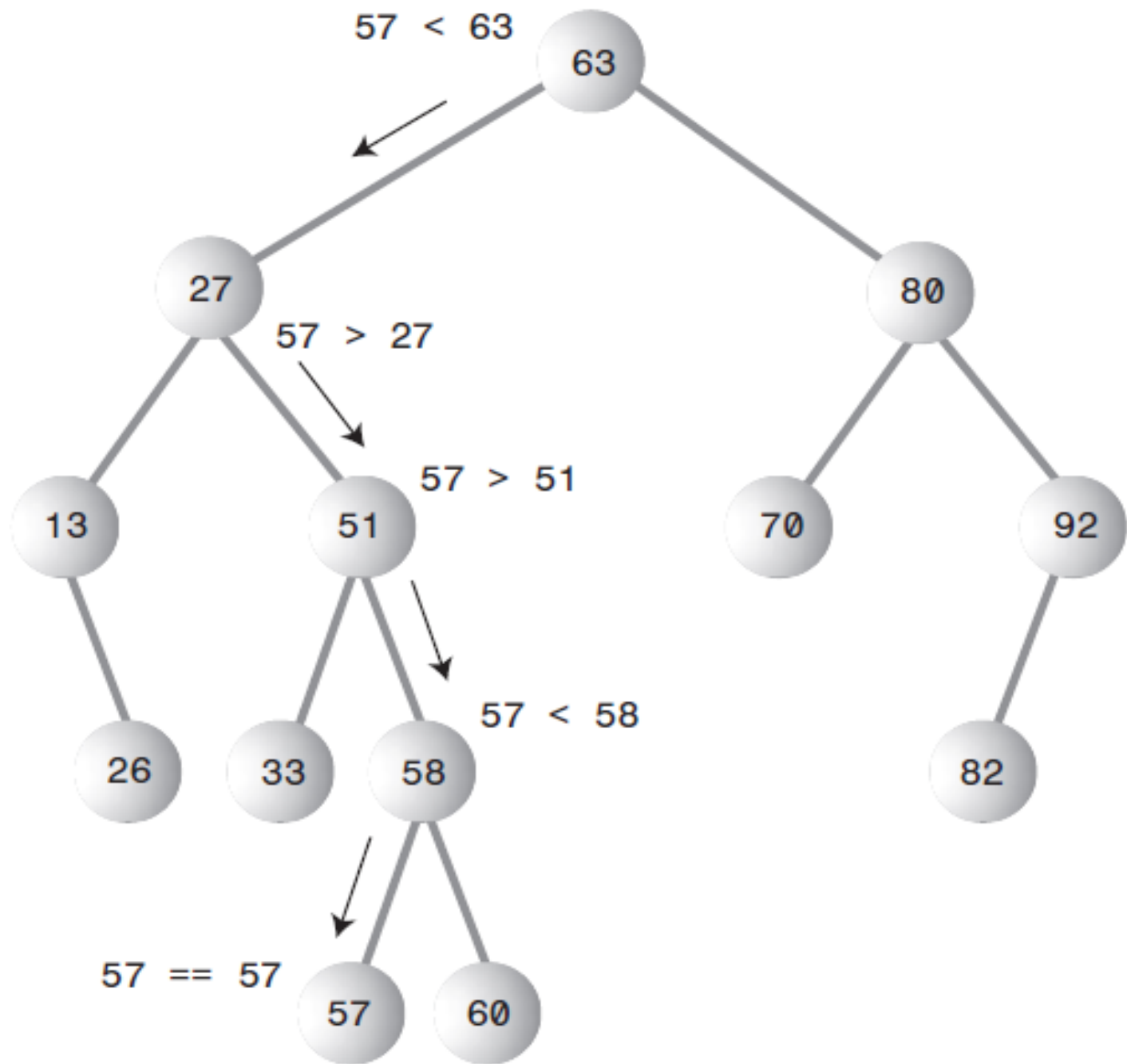
Operații frecvente asupra arborilor

- Căutarea unui nod
- Inserarea unui nod
- Traversarea arborelui
 - În preordine
 - În inordine
 - În postordine
- Ștergerea unui nod



Căutarea unui nod

- Căutarea unui nod cu o anumită cheie este cea mai simplă operație



```
tnod * find ( tnod * r, int x) { // cauta x in arborele cu radacina r
    tnod * p;
    if (r==NULL || x == r->val) // daca arbore vid sau x in nodul r
        return r;                // poate fi si NULL
    p= find (r->st,x);             // rezultat cautare in subarbore stanga
    if (p != NULL)                // daca s-a gasit in stanga
        return p;                // rezultat adresa nod gasit
    else                          // daca nu s-a gasit in stanga
        return find (r->dr,x);    // rezultat cautare in subarbore dreapta
}
```



Eficiența operației de căutare

- Timpul necesar pentru găsirea unui nod depinde de numărul de niveluri parcurse până când acesta este găsit
- Timpul este direct proporțional cu logaritmul binar al numărului de noduri



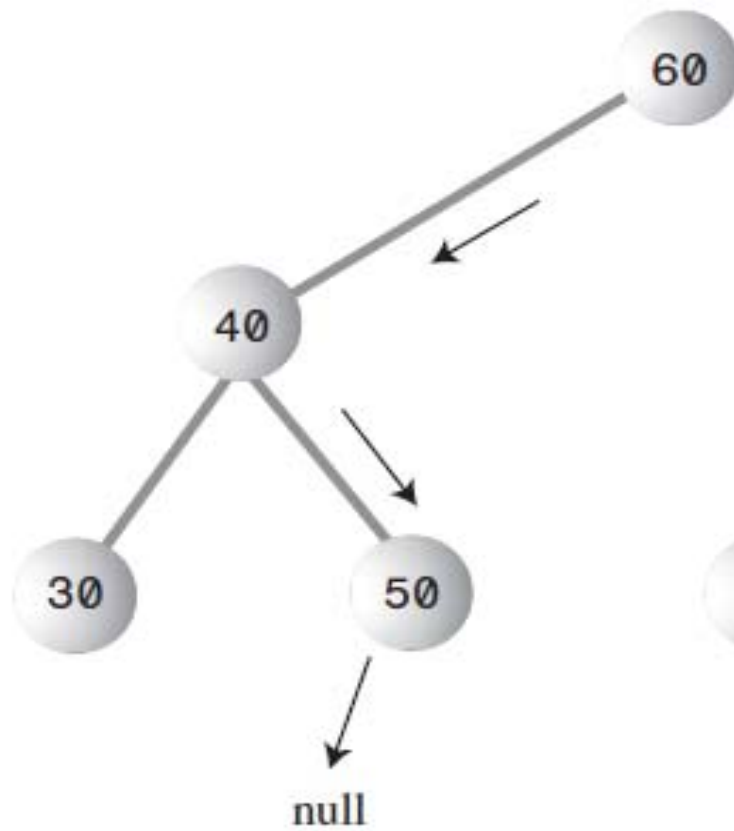
Inserarea unui nod

- Pentru a insera un nod, trebuie mai întâi să găsim locul în care îl vom insera
- Această operație este similară cu cea de căutare a unui nod care nu există
- Se parcurge calea de la rădăcină până la nodul respectiv, care va fi părintele noului nod

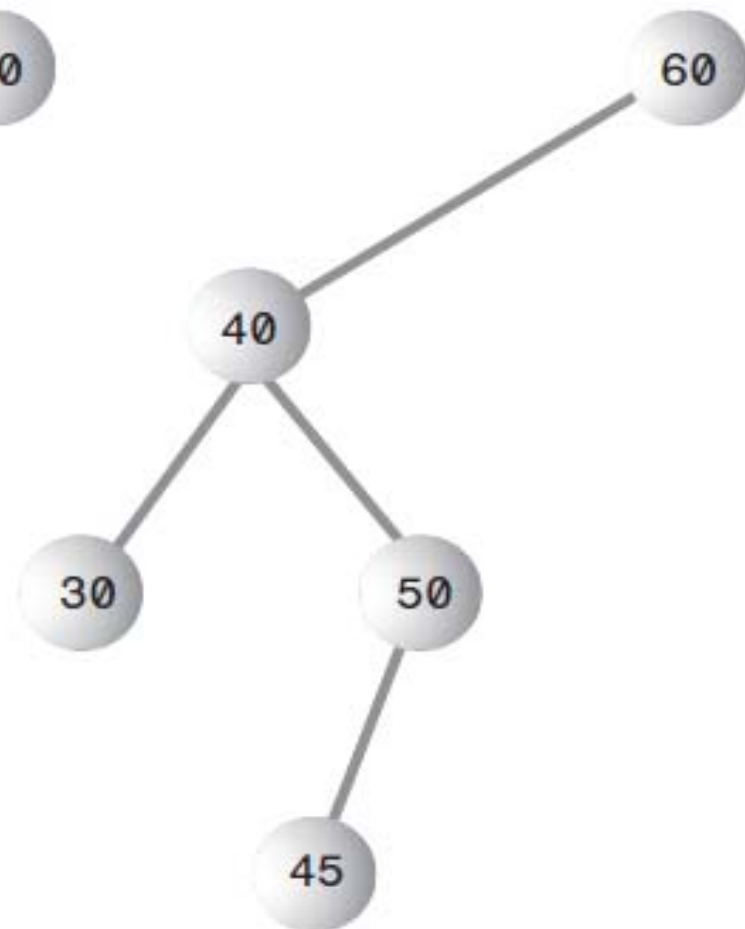


Inserarea unui nod

- După ce am găsit nodul părinte, noul nod va fi conectat ca fiu stâng sau drept al acestuia, după cum valoarea cheii sale este mai mică sau mai mare decât cea a nodului părinte



a) Before insertion



b) After insertion

Inserarea unui nod

- 1. Creează un nod nou
- 2. Dacă arborele este vid atunci noul nod devine rădăcină
- 3. Altfel
 - 3.1. Începând de la rădăcină
 - 3.2. Dacă valoarea nodului de inserat este mai mică decât valoarea rădăcinii atunci ne deplasăm recursiv la stânga
 - 3.3. Dacă nu mai este nimic la stânga, se inserează nodul

Inserarea unui nod

- 3.4. Dacă valoarea nodului de inserat este mai mare decât valoarea rădăcinii atunci ne deplasăm recursiv la dreapta
- 3.5. Dacă nu mai este nimic la dreapta, se inserează nodul



Traversarea arborelui

- Traversarea unui arbore presupune vizitarea tuturor nodurilor, într-o anumită ordine
- Traversarea nu este o operație rapidă, dar este utilă în anumite situații
- Există trei moduri de traversare a unui arbore: **inordine**, **preordine**, **postordine**



Traversarea în inordine

- Traversarea în inordine a unui arbore va conduce la vizitarea nodurilor, în ordinea crescătoare a valorilor cheilor acestora
- Dacă dorim să creăm o listă sortată care să cuprindă toate elementele unui arbore binar de căutare, parcurgerea în inordine este o soluție

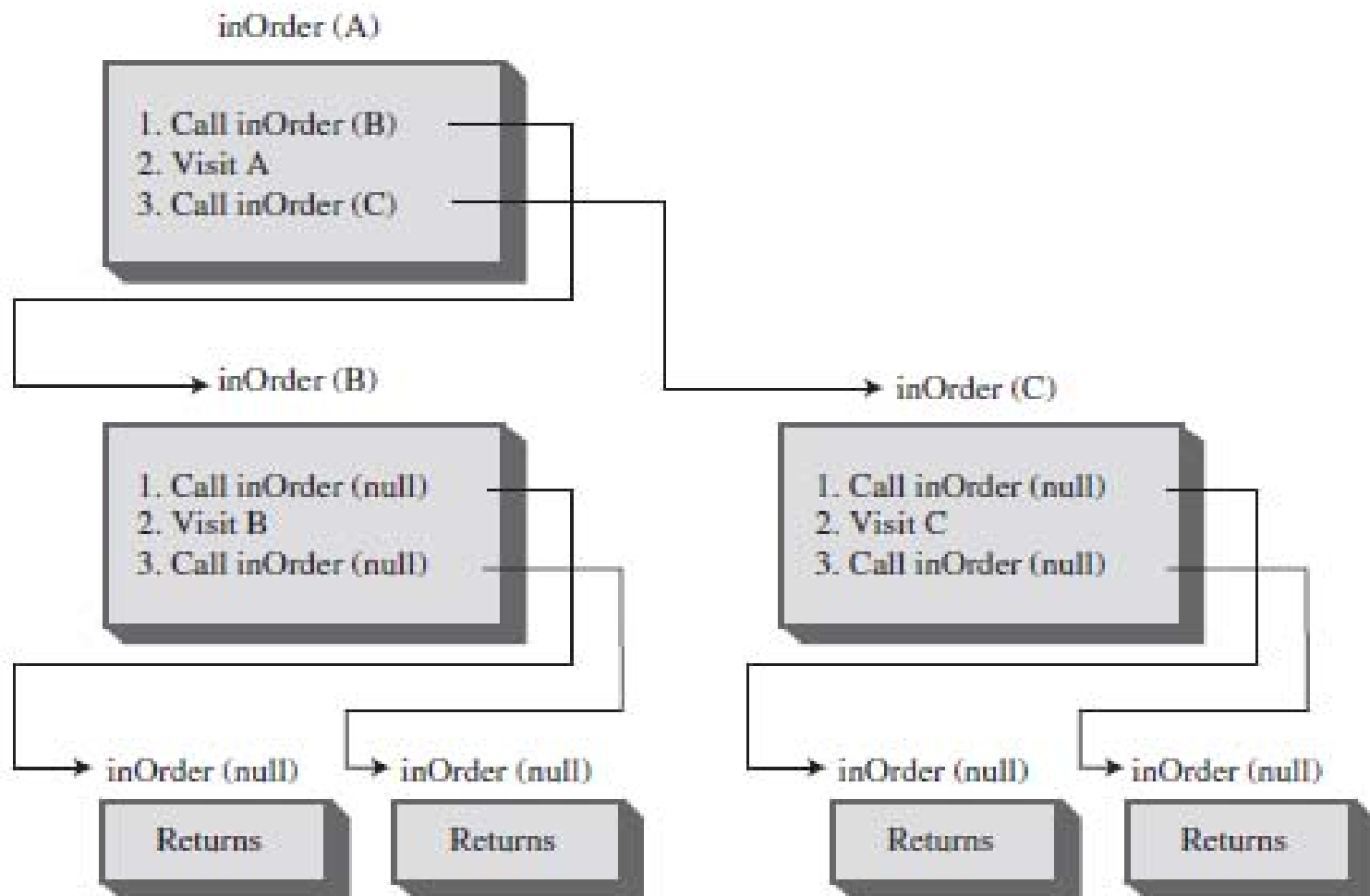
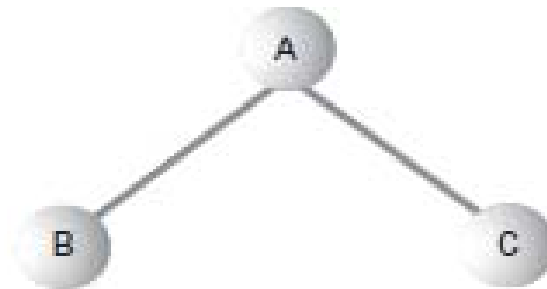
Traversarea în inordine

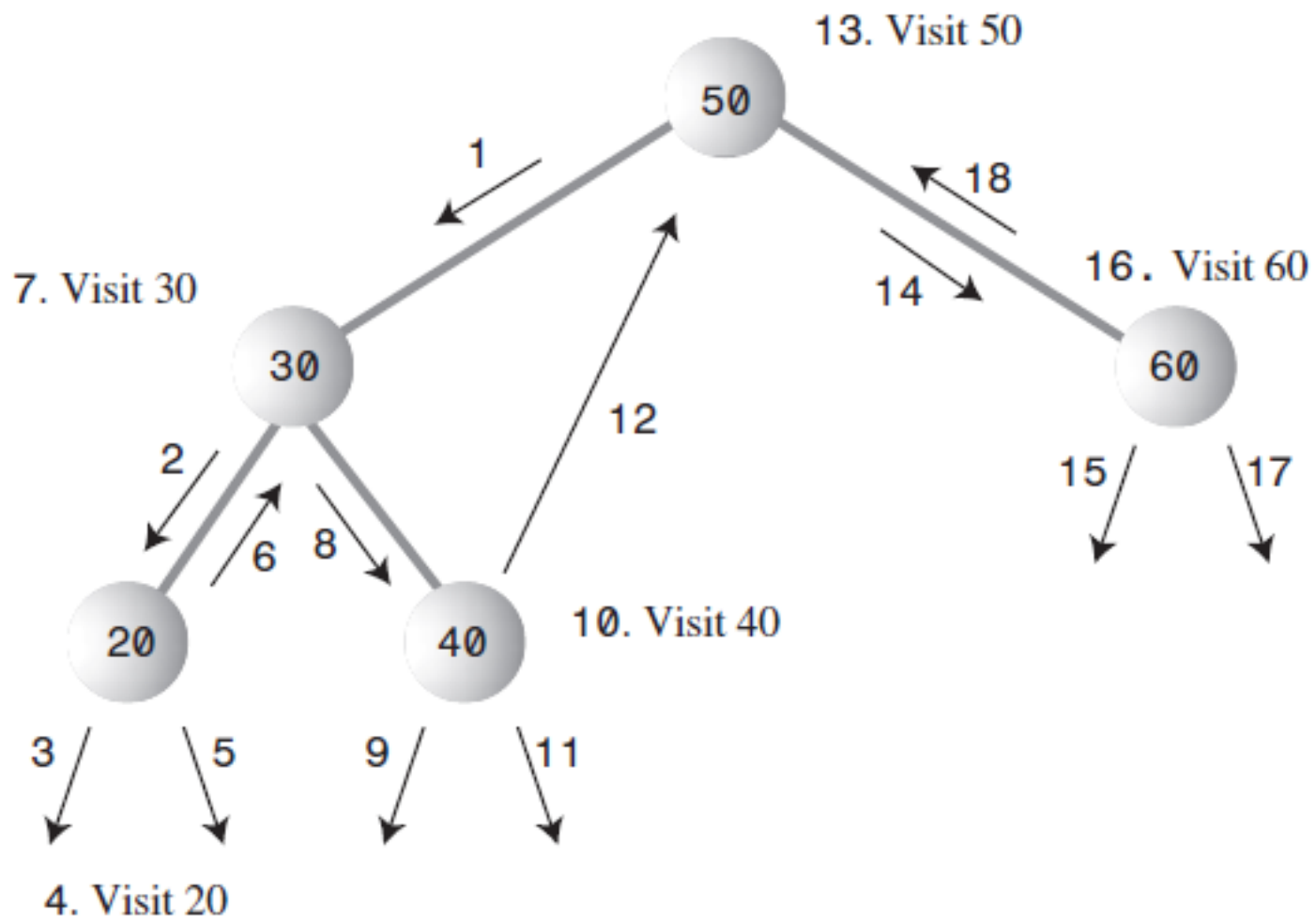
- Funcția recursivă care traversează un arbore este apelată cu un parametru reprezentând o structură de tip nod, care este chiar rădăcina arborelui
- Funcția trebuie să efectueze trei operații:
 - Un apel recursiv pentru a traversa subarborele stâng al nodului
 - Vizitarea nodului curent
 - Un apel recursiv pentru a traversa subarborele drept al nodului

```
void infix (tnod * r) {  
    if ( r == NULL) return;    // nimic daca (sub)arbore vid  
    infix (r→st);              // afisare subarbore stânga  
    printf ("%d ",r→val);      // afisare valoare din radacina  
    infix (r→dr);              // afisare subarbore dreapta  
}
```


Exemplu – Traversarea unui arbore cu trei noduri

- Presupunem că avem de traversat un arbore cu trei noduri: rădăcina A, cu fiul stâng B și fiul drept C



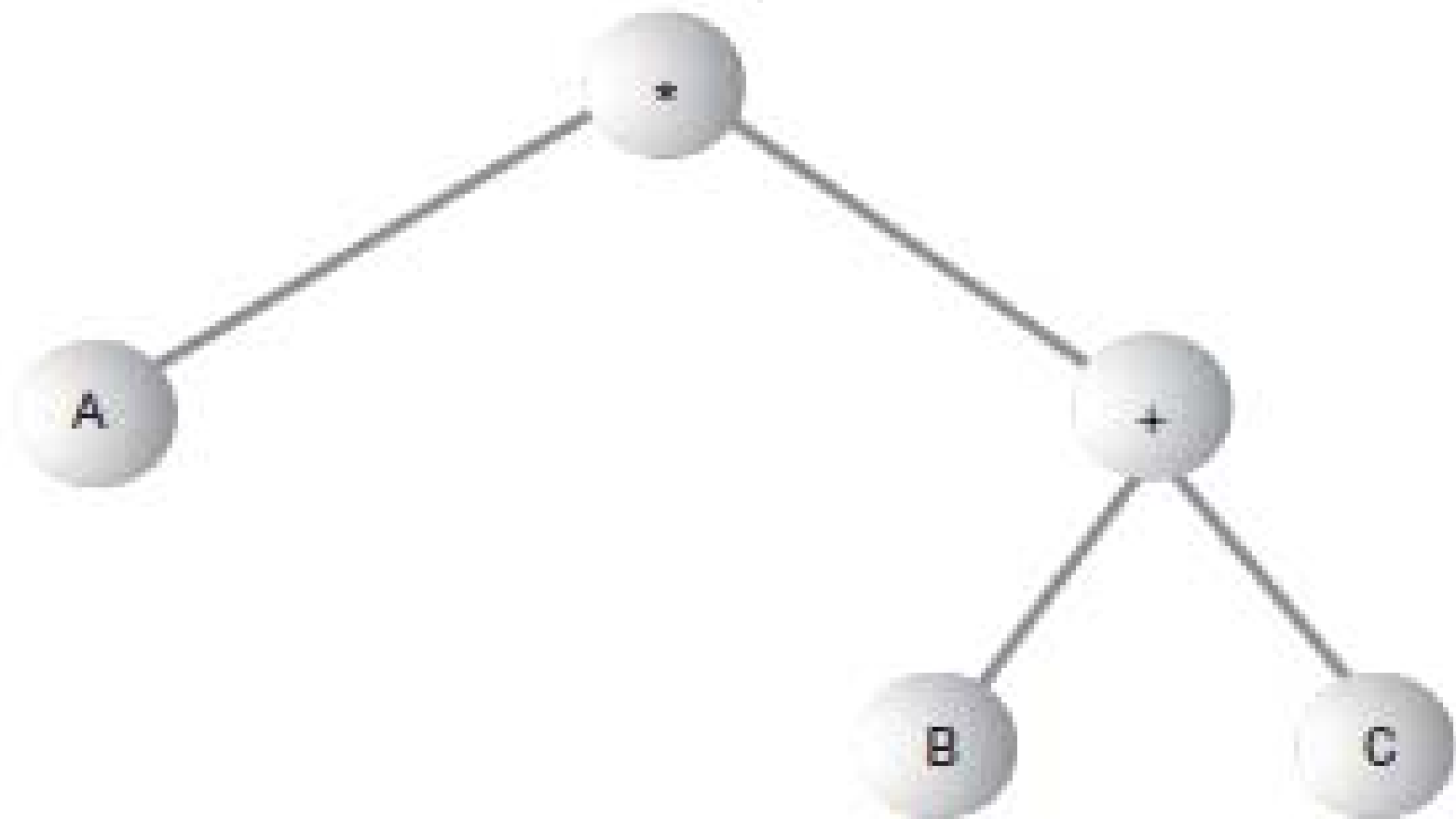


Step Number	Red Arrow on Node	Message	List of Nodes Visited
1	50 (root)	Will check left child	
2	30	Will check left child	
3	20	Will check left child	
4	20	Will visit this node	
5	20	Will check right child	20
6	20	Will go to root of previous subtree	20
7	30	Will visit this node	20
8	30	Will check right child	20 30
9	40	Will check left child	20 30
10	40	Will visit this node	20 30
11	40	Will check right child	20 30 40
12	40	Will go to root of previous subtree	20 30 40
13	50	Will visit this node	20 30 40
14	50	Will check right child	20 30 40 50
15	60	Will check left child	20 30 40 50
16	60	Will visit this node	20 30 40 50
17	60	Will check right child	20 30 40 50 60
18	60	Will go to root of previous subtree	20 30 40 50 60
19	50	Done traversal	20 30 40 50 60



Traversarea în preordine și postordine

- Aceste traversări sunt utile pentru programe care analizează expresii aritmetice
- O expresie aritmetică poate fi reprezentată printr-un arbore binar
- Nodul rădăcină conține un operator, iar fiecare din subarborii săi reprezintă fie numele unei variabile, fie o altă expresie




Infix: $A*(B+C)$

Prefix: $*A+BC$

Postfix: $ABC+*$

Traversarea în preordine


- Succesiunea pentru cazul traversării în preordine este :
 - Vizitează nodul
 - Apel recursiv pentru traversarea subarborelui stâng
 - Apel recursiv pentru traversarea subarborelui drept



```
// traversare prefixata arbore binar
void prefix (tnod * r) {
    if ( r == NULL) return;
    printf ("%d ",r→val);           // radacina
    prefix (r→st);                  // stânga
    prefix (r→dr);                  // dreapta
}
```

Traversarea în postordine

- Traversarea în postordine conține tot cei trei pași, în succesiunea:
 - Apel recursiv pentru traversarea subarborelui stâng
 - Apel recursiv pentru traversarea subarborelui drept
 - Vizitează nodul

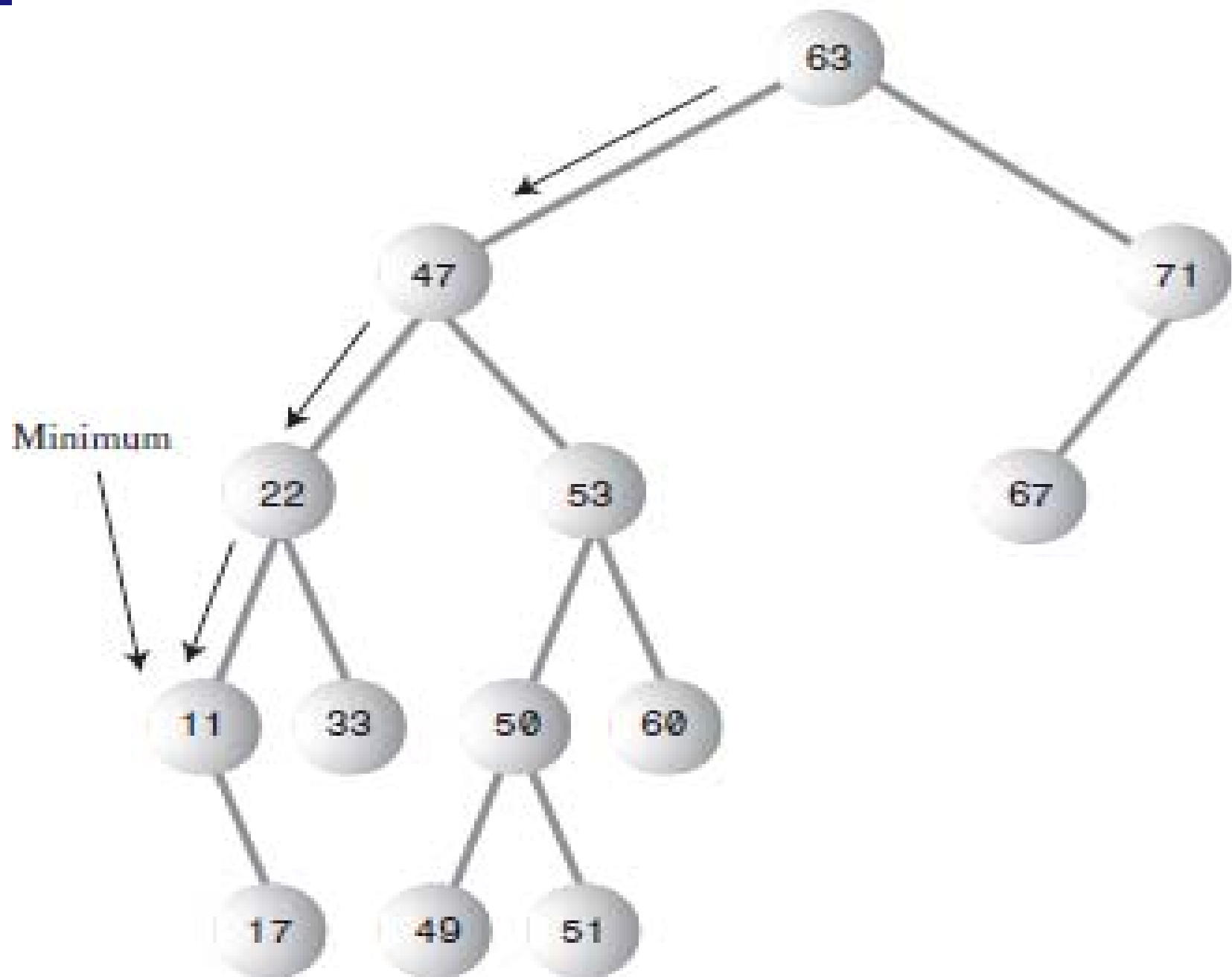


```
//traversare postfixata arbore binar  
void postfix(tnod * r) {  
    if (r == NULL) return;  
    postfix(r->st); // stanga  
    postfix(r->dr); // dreapta  
    printf("%d ", r->val); //radacina  
}
```



Valorile minime și maxime

- Pentru determinarea valorii minime, ne deplasăm în fiul stâng al rădăcinii
- De acolo, în fiul stâng al acelui fiu ș.a.m.d., până când ajungem la un nod care nu mai are niciun fiu stâng
- Acest nod conține valoarea minimă din arbore





Valorile minime și maxime

- Va fi necesar să știm cum se determină minimul, pentru a șterge un nod din arbore
- Pentru determinarea nodului cu valoarea maximă procedăm similar, avansând mereu spre fiul drept, până când găsim un nod care nu mai are niciun fiu drept
- Acest nod va conține valoarea maximă

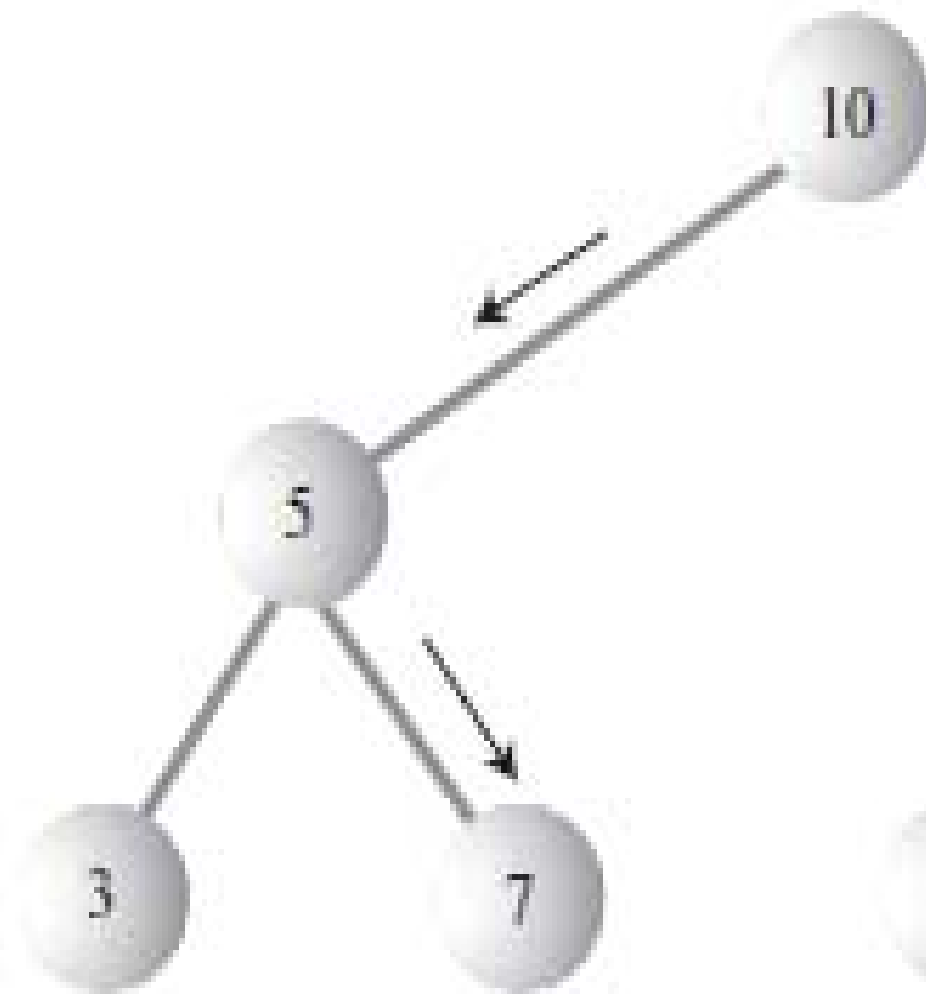


Ștergerea unui nod

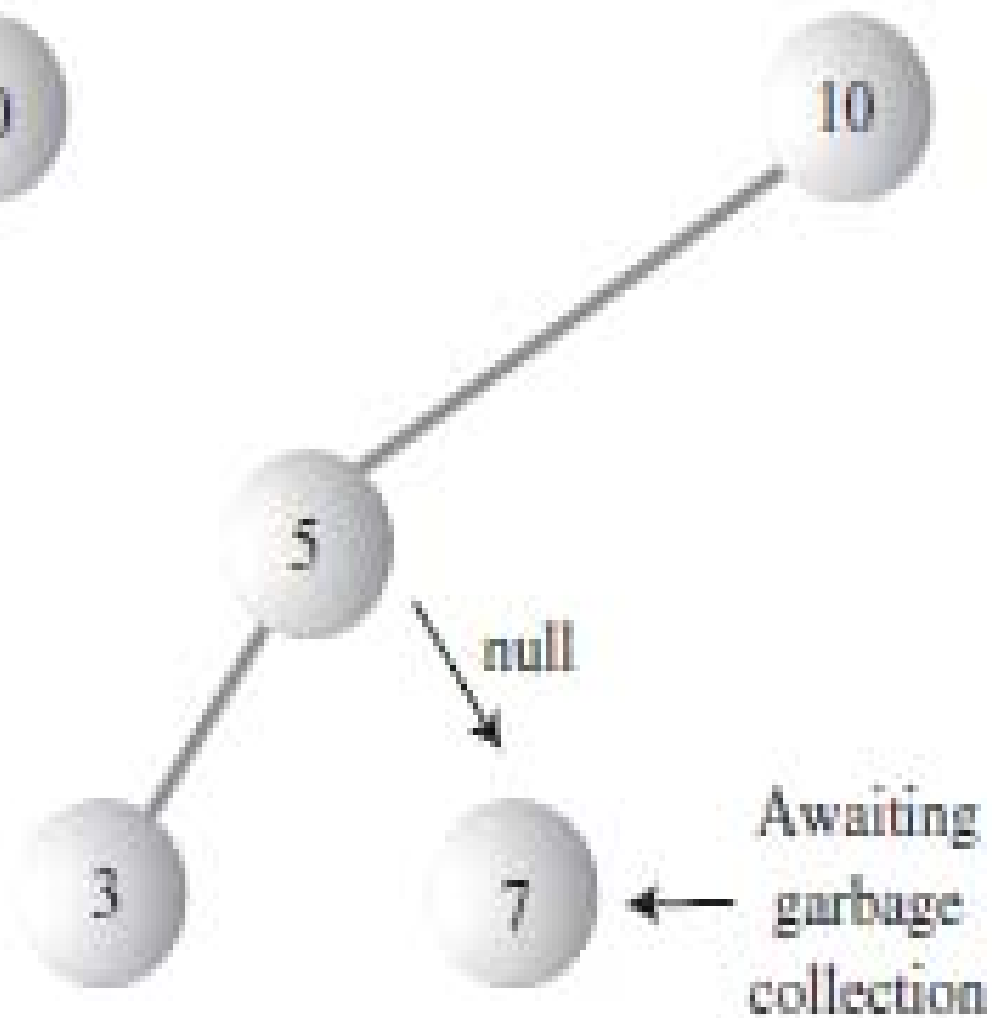
- Mai întâi se caută nodul care va fi șters
- După ce am găsit nodul, apar trei cazuri care trebuie considerate separat:
 - 1. Nodul care va fi șters este o frunză (nu are fii)
 - 2. Nodul are un singur fiu
 - 3. Nodul are doi fii

Cazul 1. Nodul șters nu are fii

- Pentru a șterge un nod frunză se modifică pointerul către nodul părinte, atribuindu-i acestuia valoarea NULL
- Trebuie să eliminăm explicit nodul din memorie, apelând funcția `free()`



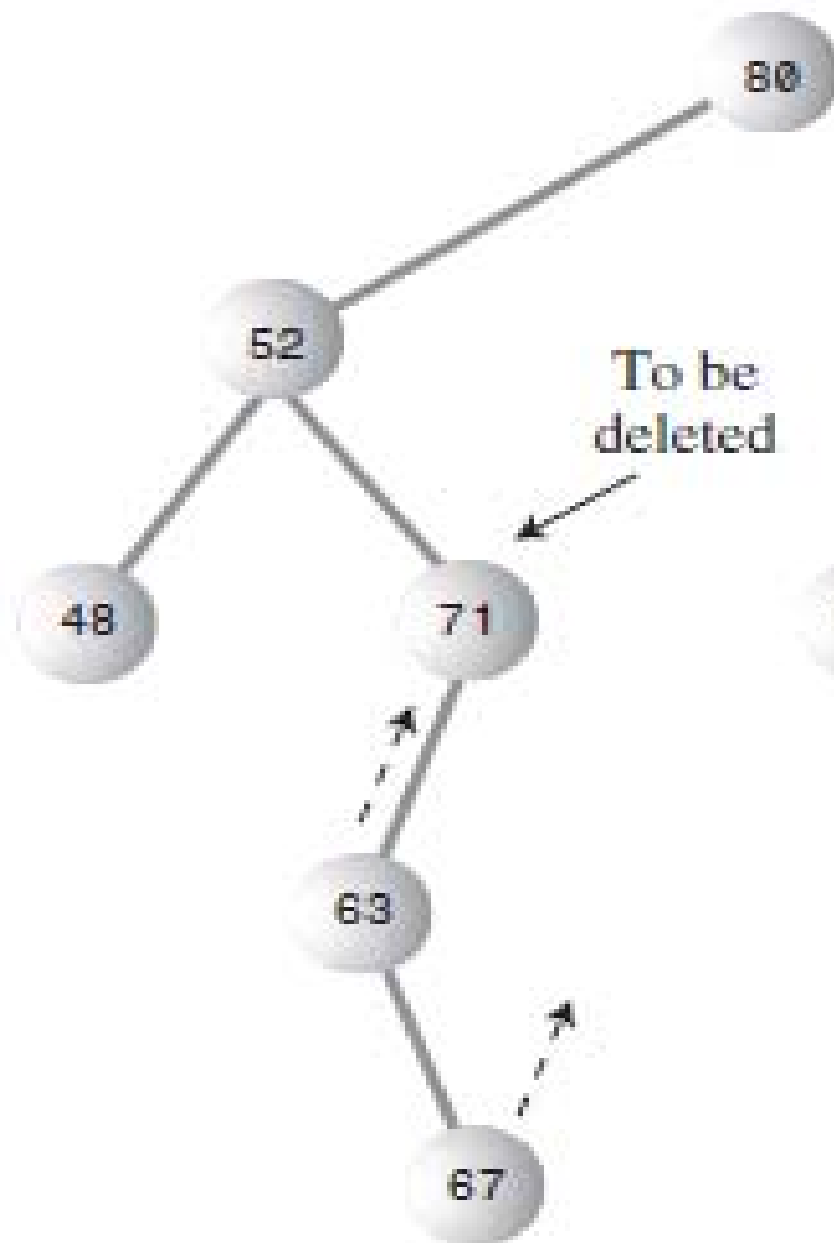
a) Before deletion



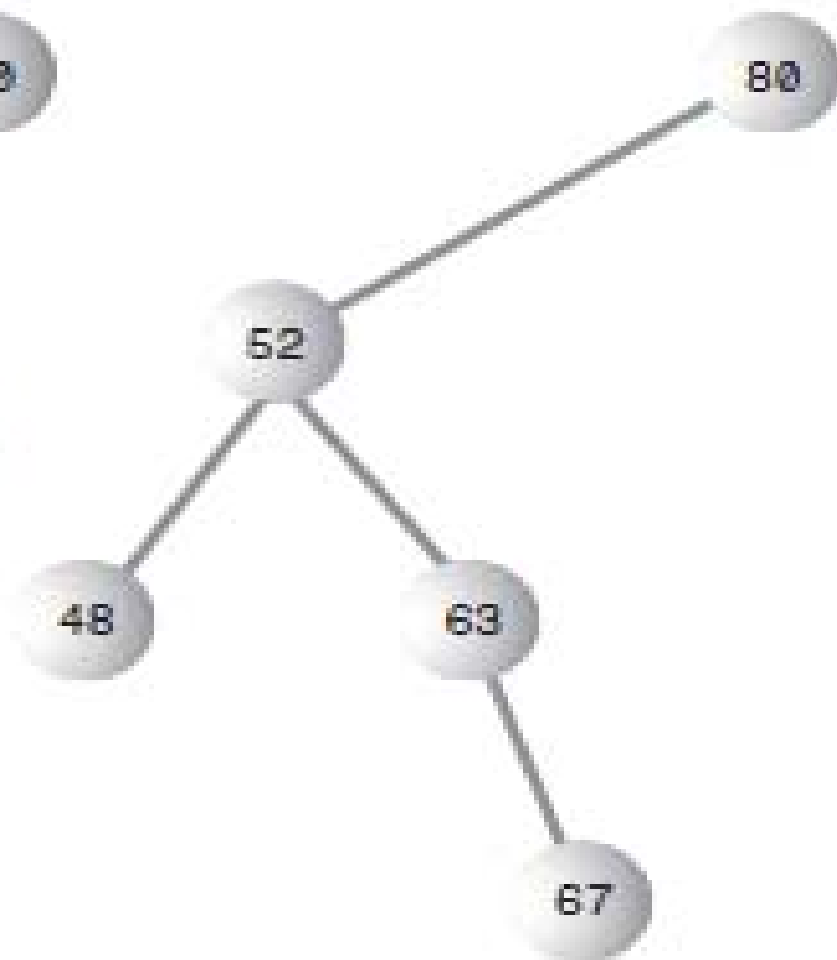
b) After deletion

Cazul 2. Nodul șters are un singur fiu

- Nodul șters are două legături: una către părinte și cealaltă către unicul descendent
- Vom tăia nodul din această secvență, conectând direct singurul fiu al nodului cu nodul părinte
- Pointerul corespunzător al părintelui va fi modificat, astfel încât să indice spre fiul nodului șters



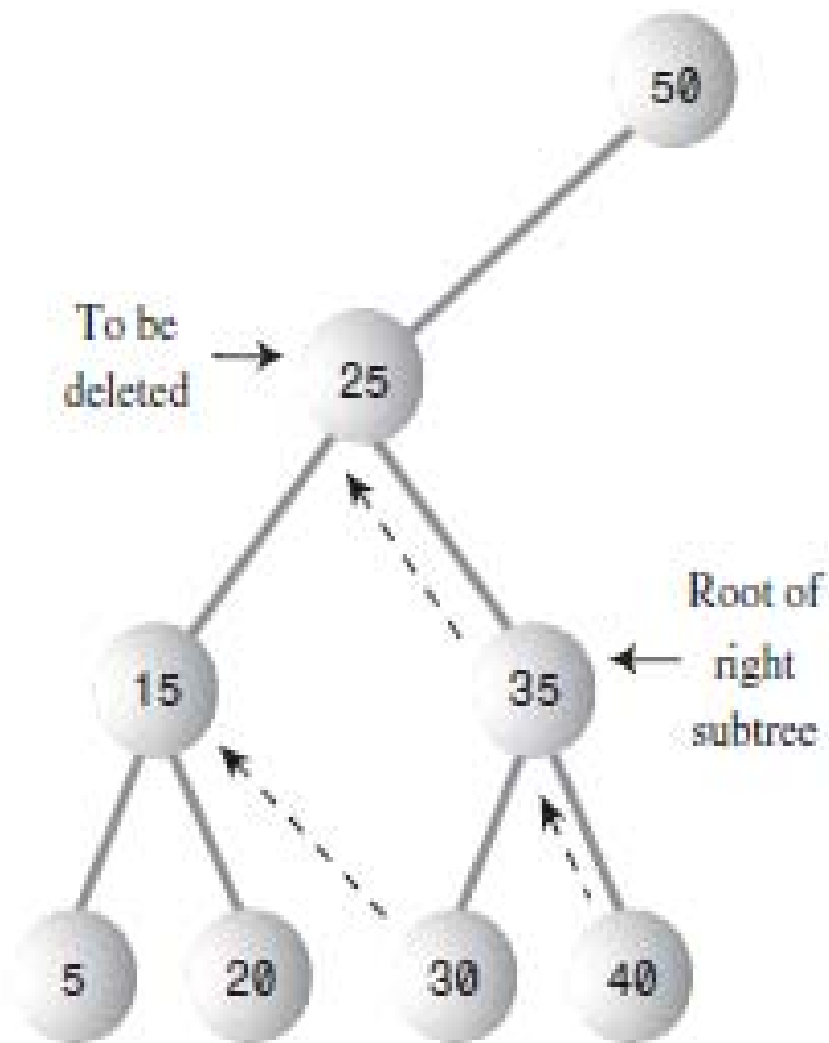
a) Before deletion



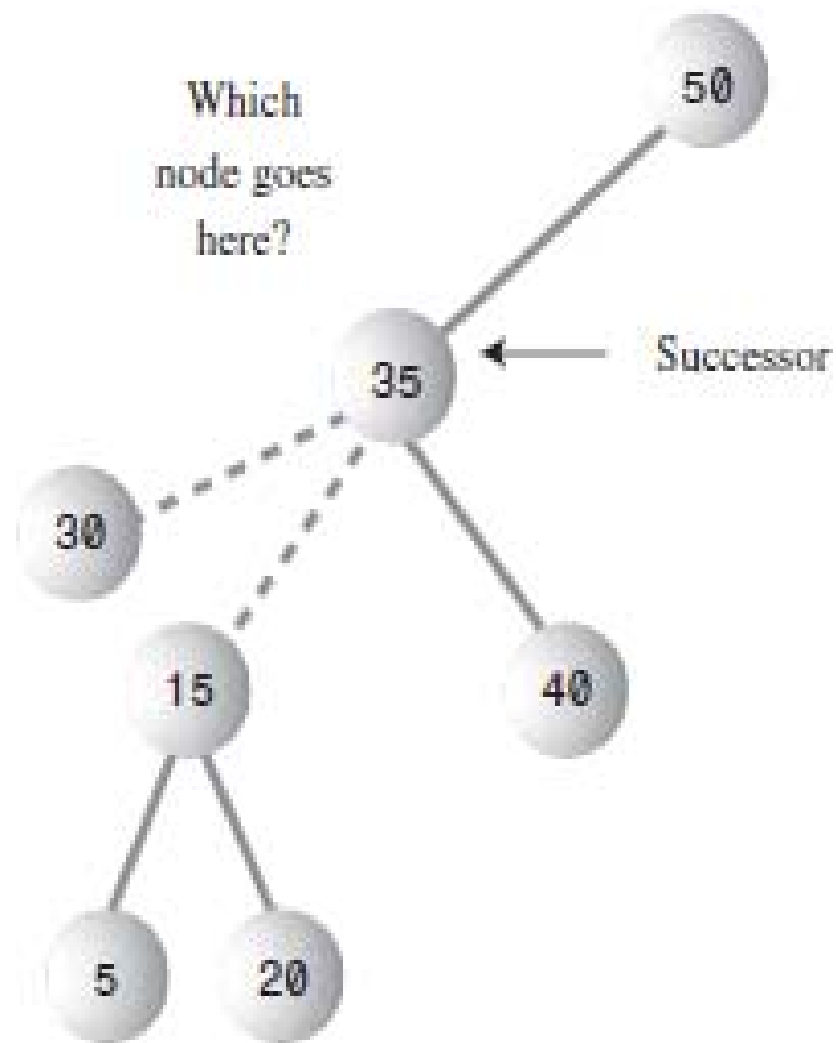
b) After deletion

Cazul 3. Nodul șters are doi fii

- Dacă nodul șters are doi fii, nu îl putem înlocui pur și simplu cu unul dintre aceștia, cel puțin în cazul în care fiul are la rândul său alți fii



a) Before deletion



b) After deletion

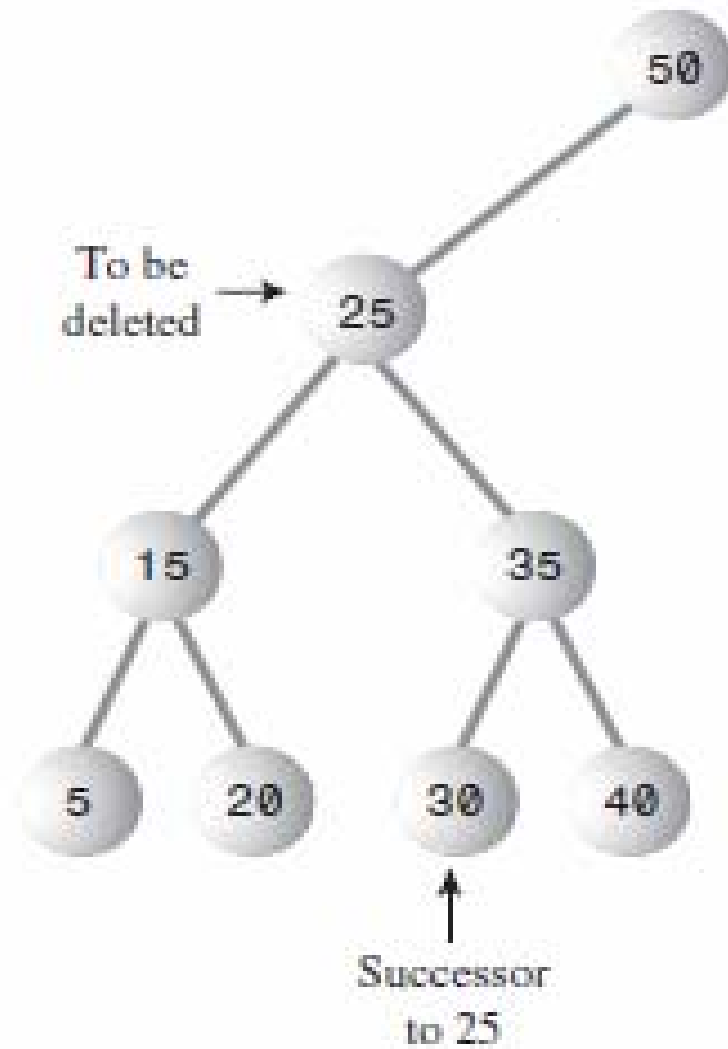
Căutarea succesorului

- Într-un arbore binar de căutare, nodurile sunt în ordinea crescătoare a cheilor
- Pentru un anumit nod, nodul cu cheia imediat superioară se numește **succesorul în inordine** al nodului, sau pur și simplu **succesorul** nodului

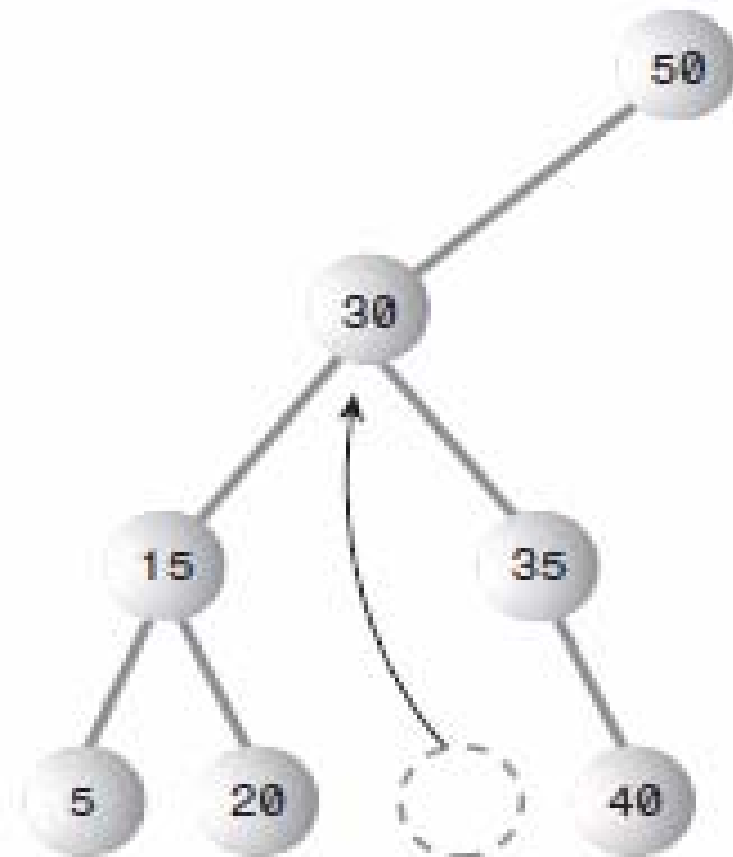


Căutarea succesoriului

- Pentru a șterge un nod cu doi fii, vom înlocui nodul șters cu succesorul său
- Această operație păstrează ordinea elementelor
- Operația se complică dacă succesorul nodului șters are la rândul său succesori



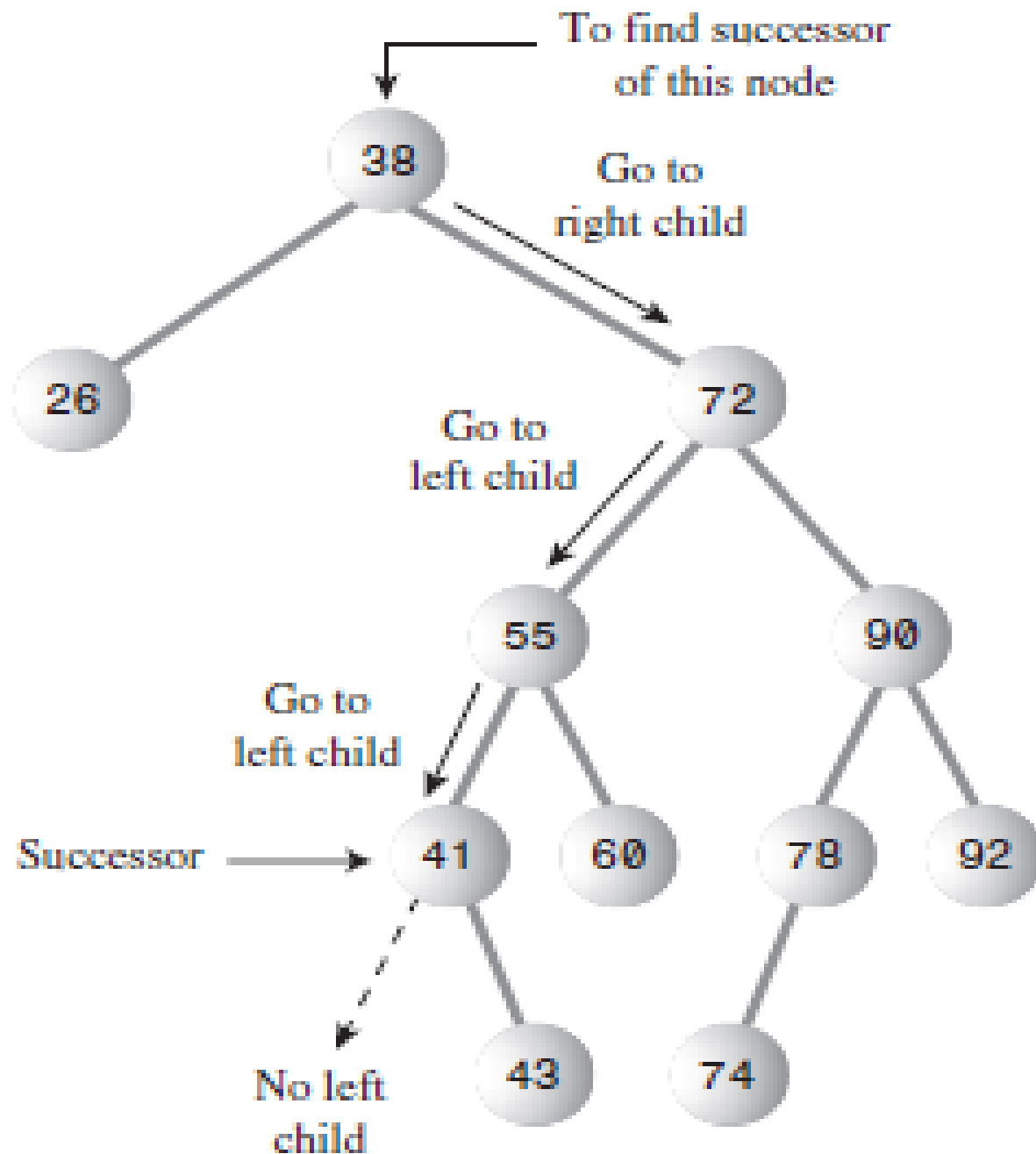
a) Before deletion



b) After deletion

Căutarea succesorului

- Algoritmul se deplasează în fiul drept al nodului șters, a cărui cheie este mai mare
- În următorul pas, se efectuează o deplasare în fiul stâng al fiului drept (dacă există un astfel de fiu), continuându-se pe cât posibil deplasarea pe o cale alcătuită numai din fii stângi
- Ultimul nod din această cale este succesorul nodului inițial





Căutarea succesoriului

- Nodul pe care îl căutăm este minimul mulțimii de noduri care sunt mai mari decât nodul original
- Când ne deplasăm în subarborele drept, toate nodurile de acolo sunt mai mari decât cel inițial, aceasta rezultând din modul de definire a unui arbore binar de căutare

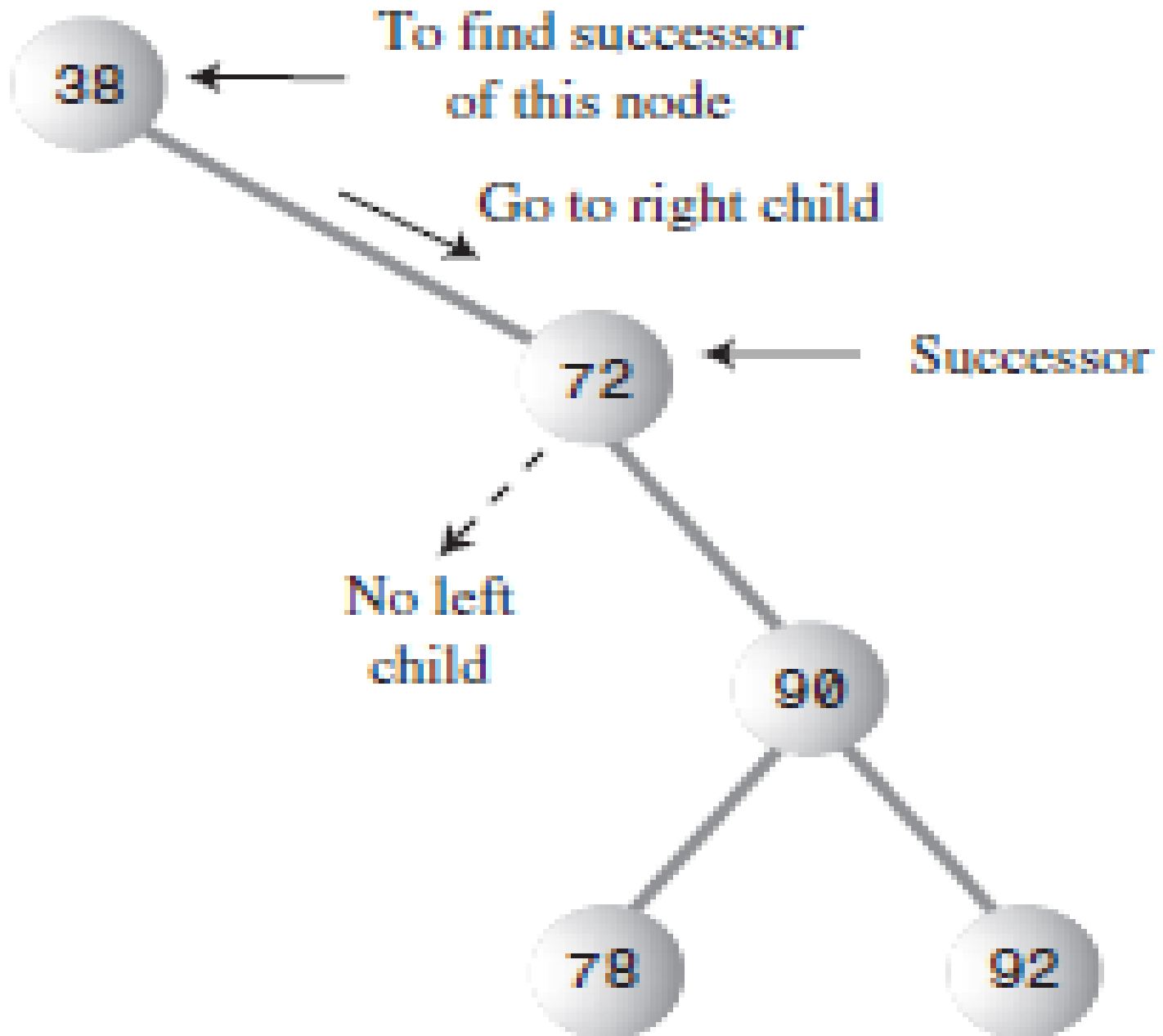
Căutarea succesoriului

- Dorim să determinăm cea mai mică valoare din acest subarbore
- Valoarea minimă dintr-un arbore se poate determina urmând calea care pornește de la rădăcină și merge numai prin fiii stângi
- Algoritmul va determina valoarea minimă care este mai mare decât nodul original
- Această valoare este chiar succesoriul căutat



Căutarea succesorului

- Dacă fiul drept al nodului nu are fii stângi, succesorul este el însuși





Algoritmul pentru determinarea succesorului

- Funcția `getSuccesor()` întoarce succesorul unui nod `delNode`, pe care îl primește ca parametru
- Funcția presupune existența subarborelui drept al nodului `delNode`, ceea ce este adevărat, din moment ce nodul șters are doi fii



Algoritmul pentru determinarea succesorului

- Funcția întoarce nodul cu valoarea imediat mai mare decât delNode
- Funcția se deplasează în fiul drept, apoi în descendenții stângi ai acestuia
- Funcția parcurge, într-o buclă while, calea formată din descendenții stângi ai acestui fiu drept
- La terminarea buclei while, variabila succesor conține succesorul nodului delNode

Algoritmul pentru determinarea succesorului

- După ce am determinat succesorul, trebuie să avem acces și la părintele acestuia
- Vom memora și părintele nodului curent, în variabila `succesorParent`
- Nodul succesor poate fi situat în două poziții distincte, în raport cu `current`, nodul care va fi șters
- Succesorul este fie fiul drept al lui `current`, fie unul dintre descendenții stângi ai acestuia

Pseudocod getSuccessor

■ 1. $\text{successorParent} \leftarrow \text{delNode}$

■ 2. $\text{successor} \leftarrow \text{delNode}$

■ 3. $\text{current} \leftarrow \text{delNode} \rightarrow \text{dr}$

//deplasare în fiul drept

■ 4. while ($\text{current} \neq \text{NULL}$)

//până când nu mai sunt fii stângi

□ 4.1. $\text{successorParent} \leftarrow \text{successor}$

□ 4.2. $\text{successor} \leftarrow \text{current}$

□ 4.3. $\text{current} \leftarrow \text{current} \rightarrow \text{st}$

//deplasare în fiul stâng

Pseudocod getSuccesor

■ 5. if (succesor \neq delNode \rightarrow dr)

//daca succesorul nu este chiar fiul drept

//se modifica pointerii

□ 5.1. succesorParent \rightarrow st \leftarrow succesor \rightarrow dr

□ 5.2. succesor \rightarrow dr \leftarrow delNode \rightarrow dr

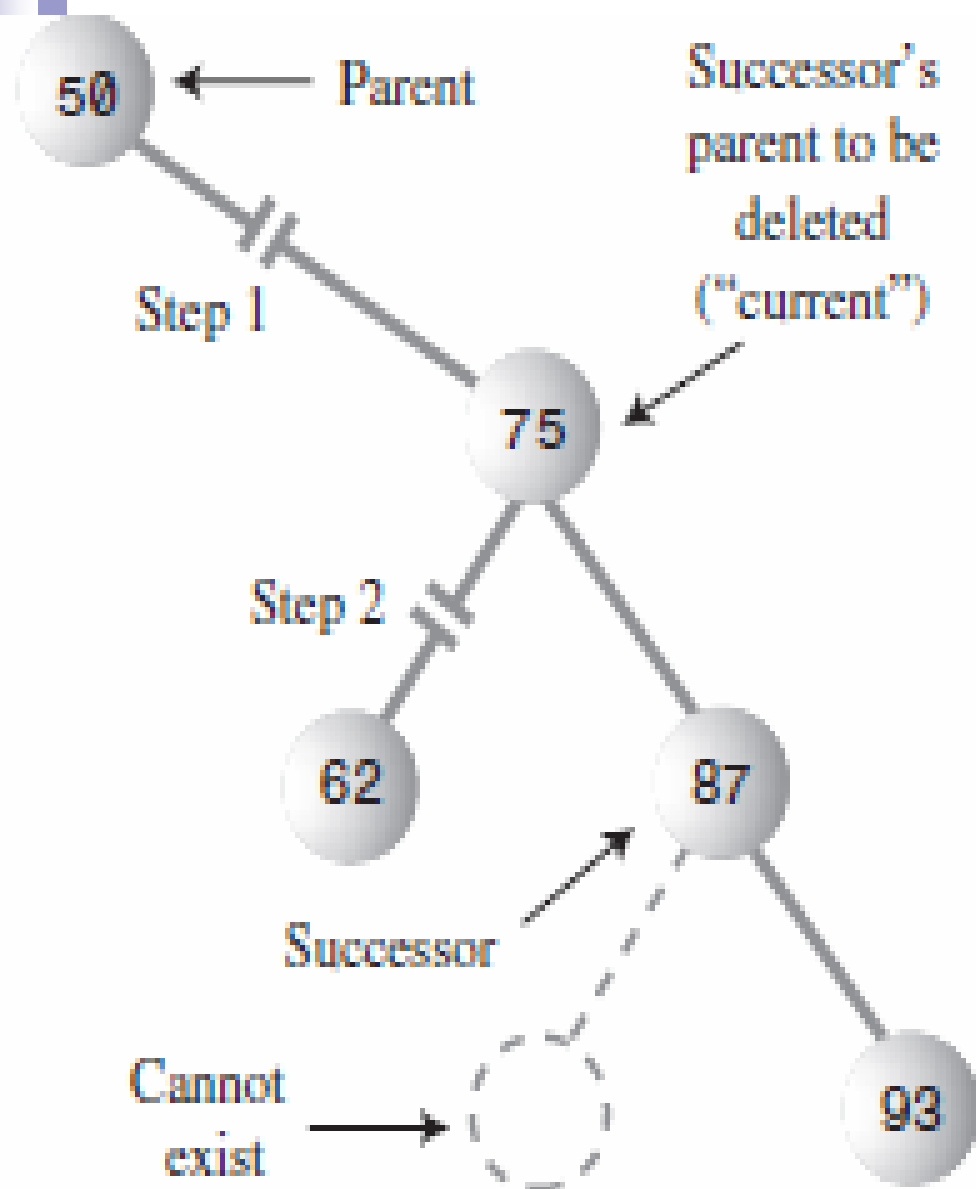
■ 6. return succesor

succesor este fiul drept al lui delNode

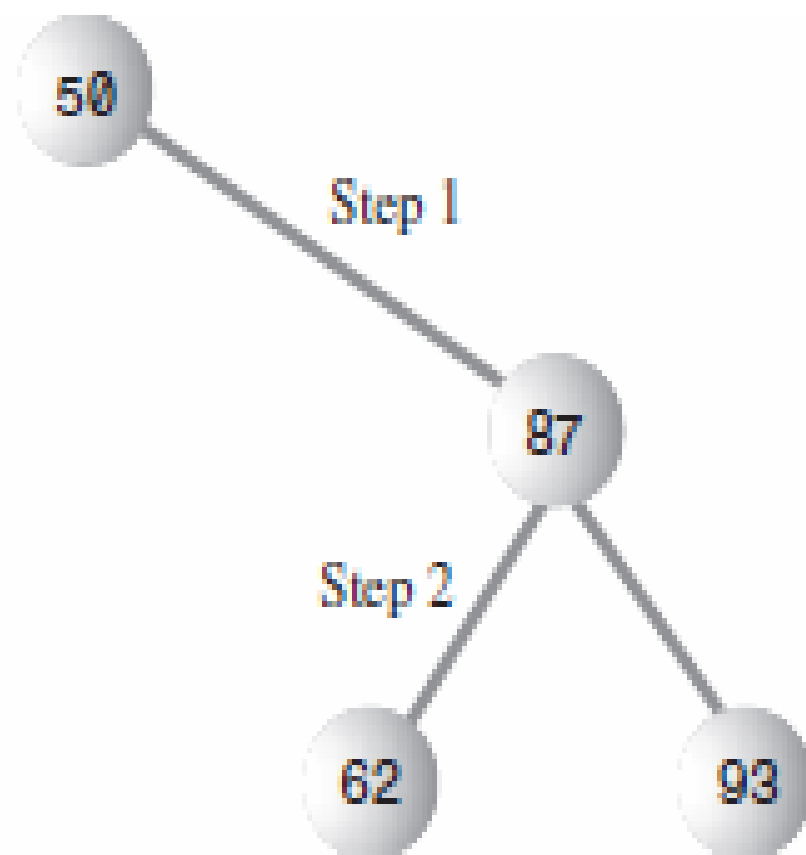
- Dacă succesor este chiar fiul drept al lui delNode, lucrurile se simplifică, deoarece putem deplasa subarborele cu rădăcina succesor (de fapt, subarborele drept al acestuia, cel stâng fiind vid) în locul nodului șters
- Operația se efectuează în doi pași:

succesor este fiul drept al lui delNode

- 1. Se elimină legătura dintre câmpul dr (sau st, după caz) al nodului părinte și nodul șters, atribuindu-se acestui câmp valoarea succesor
- 2. Subarborele stâng al lui current este conectat ca subarbore stâng al lui succesor, după ce, în prealabil, a fost deconectat de la current



a) Before deletion



b) After deletion

Pseudocod pentru ștergerea nodului

//determină succesorul nodului șters(current)

- 1. $\text{succesor} \leftarrow \text{getSuccesor}(\text{current})$

//conectează părintele lui current cu succesor

- 2. if (current == root) then root = succesor

Pseudocod pentru ștergerea nodului

■ 3. else dacă există fiul stâng then

$\text{parent} \rightarrow \text{st} \leftarrow \text{succesor}$

else $\text{parent} \rightarrow \text{dr} \leftarrow \text{succesor}$

//conectează succesorul cu fiul stâng al lui
current

■ 4. $\text{succesor} \rightarrow \text{st} \leftarrow \text{current} \rightarrow \text{st}$

Observații

- Pasul 1: Dacă nodul șters, current, este chiar rădăcina, care nu are părinte, va fi necesar să atribuim valoarea succesor rădăcinii arborelui
- În caz contrar, nodul șters este fie fiul stâng, fie cel drept al unui alt nod
- Câmpul corespunzător al nodului părinte va fi modificat pentru a indica spre nodul succesor

Observații

- Pasul 2: Fiul stâng al nodului șters (current) este conectat ca fiu stâng al lui succesor
- Proprietate: Un nod succesor nu va avea niciodată un fiu stâng
- Această proprietate este adevărată, indiferent dacă succesorul este chiar fiul drept al nodului șters sau unul din descendenții stângi ai acestuia



Observații

- Algoritmul utilizat pentru determinarea succesorului parcurge mai întâi fiul drept, apoi descendenții stângi ai acestuia
- Algoritmul se oprește când ajunge la un nod fără niciun fiu stâng deci, din definiția algoritmului, este clar că succesorul nu poate avea un astfel de fiu
- Dacă ar fi avut, acel fiu stâng ar fi fost succesor în locul său

Observații

- Succesorul poate avea un fiu drept
- Existența fiului drept nu este o problemă dificilă, atunci când succesorul este fiul drept al nodului șters
- Când deplasăm succesorul, subarborele său drept se deplasează împreună cu el
- Nu există o suprapunere cu fiul drept al nodului șters, deoarece acest fiu este succesorul însuși



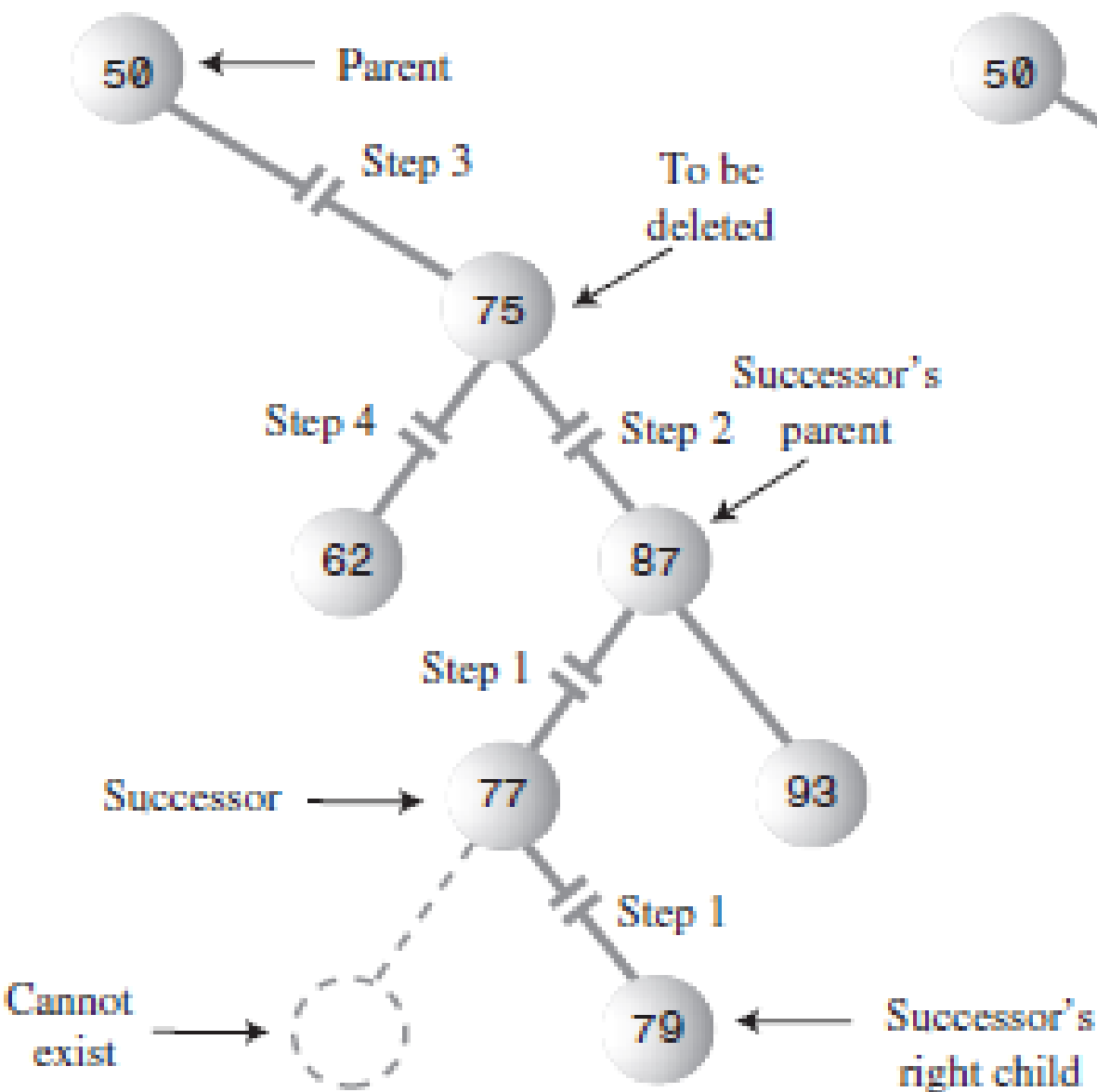
succesor este un descendent
stâng al fiului drept al lui delNode

- Dacă succesor este un descendent stâng al fiului drept al nodului care va fi șters, operația de ștergere va necesita 4 pași:
- 1. Fiul drept al succesorului devine fiul stâng al părintelui succesorului
- 2. Fiul drept al nodului șters devine fiul drept al succesorului

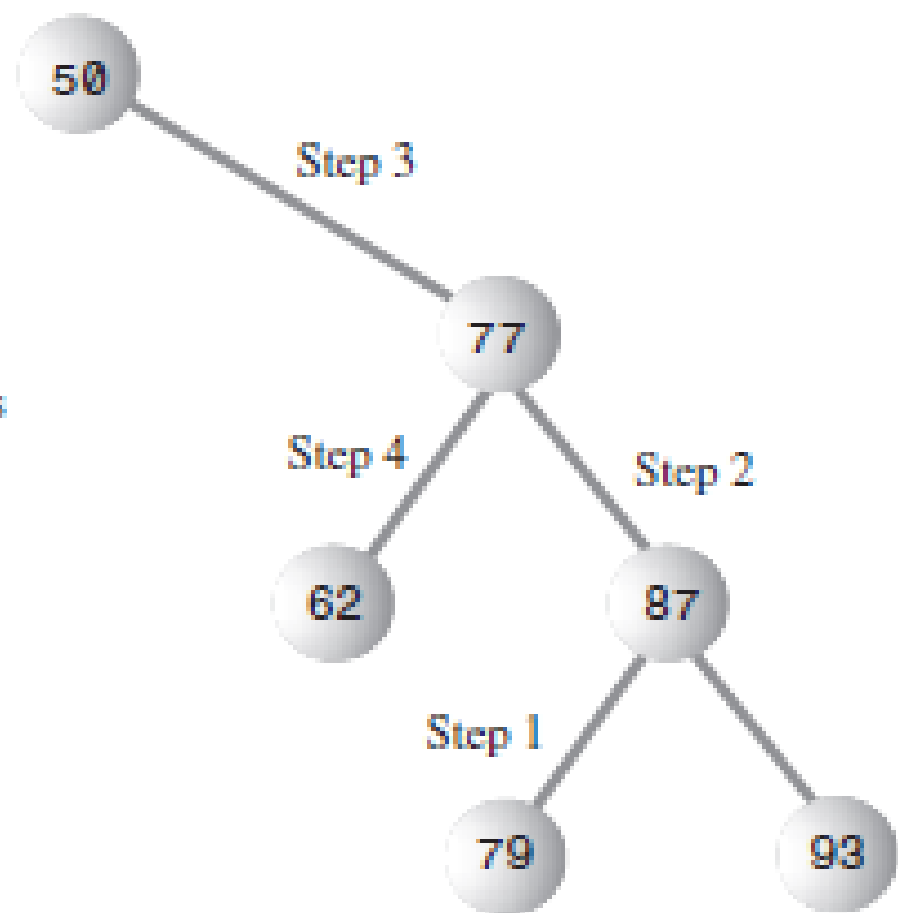


succesor este un descendent stâng al fiului drept al lui delNode

- 3. Se șterge legătura care îl conectează pe current ca fiu drept (sau stâng) al părintelui său, conectându-se succesor în locul lui current
- 4. Subarborele stâng al nodului șters devine subarborele stâng al lui succesor
- Primii doi pași sunt efectuați în funcția getSuccesor(), iar ultimii doi, în funcția de ștergere



a) Before deletion



b) After deletion

Instrucțiunile din cei 4 pași

- 1. $\text{succesorParent} \rightarrow \text{st} \leftarrow \text{succesor} \rightarrow \text{dr}$
- 2. $\text{succesor} \rightarrow \text{dr} \leftarrow \text{delNode} \rightarrow \text{dr}$
- 3. $\text{parent} \rightarrow \text{dr} \leftarrow \text{succesor}$
- 4. $\text{succesor} \rightarrow \text{st} \leftarrow \text{current} \rightarrow \text{st}$

Observații

- Pasul 1 are ca efect înlocuirea succesorului cu subarborele său drept
- Pasul 2 păstrează fiul drept al nodului șters în locul potrivit (ceea ce se întâmpla automat atunci când succesorul era fiul drept al nodului șters)
- Pașii 1 și 2 sunt executați în instrucțiunea if din finalul funcției `getSuccesor()`




Eficiența arborilor binari

- Majoritatea operațiilor asupra arborilor presupun o parcurgere descendentă pentru căutarea unui anumit nod
- Cât timp durează o astfel de căutare într-un arbore?
- Într-un arbore complet, aproximativ jumătate din noduri se află pe ultimul nivel



Eficiența arborilor binari

- Pe parcursul unei căutări, vom vizita câte un nod de pe fiecare nivel
- Prin urmare, durata necesară operației de căutare va fi direct proporțională cu numărul de niveluri
- Presupunând că arborele este complet, în tabel se prezintă numărul de niveluri necesar pentru memorarea anumitor numere de noduri



Number of Nodes	Number of Levels
1	1
3	2
7	3
15	4
31	5
...	...
1,023	10
...	...
32,767	15
...	...
1,048,575	20
...	...
33,554,432	25
...	...
1,073,741,824	30

Eficiența arborilor binari

- Dacă notăm numărul de noduri din prima coloană cu N , iar numărul de niveluri din coloana a doua cu L , observăm că N este inferior cu o unitate față de 2 ridicat la puterea L :
- $N = 2^L - 1$
- Adunând 1 în ambii membri ai ecuației, rezultă:
- $N + 1 = 2^L$

Eficiența arborilor binari

- Această notație este echivalentă cu:
- $L = \log_2(N + 1)$
- Prin urmare, timpul necesar pentru executarea operațiilor elementare asupra arborilor binari este proporțional cu logaritmul binar al lui N



Comparații între arbori și celelalte structuri de date


- Într-un tablou neordonat sau într-o listă cu 1.000.000 de elemente, sunt necesare, în medie, 500.000 de comparații pentru a căuta un anumit element
- Într-un arbore cu același număr de noduri, numărul de comparații efectuate este însă cel mult 20

Comparații între arbori și celelalte structuri de date

- Într-un tablou ordonat, putem căuta rapid un element, dar inserarea unui nou element va necesita, în medie, deplasarea a 500.000 de elemente
- Inserarea unui nod într-un arbore cu 1.000.000 de elemente necesită numai 20 de comparații și un timp scurt pentru a conecta noul nod în arbore

Comparații între arbori și celelalte structuri de date

- Ștergerea unui element dintr-un tablou cu 1.000.000 de elemente necesită, în medie, deplasarea a 500.000 de elemente
- Ștergerea unui nod dintr-un arbore cu același număr de noduri presupune cel mult 20 de comparații pentru a căuta elementul, plus alte câteva, necesare căutării succesorului său, și o durată scurtă de timp pentru a realiza efectiv ștergerea elementului și înlocuirea sa cu succesorul



Comparații între arbori și celelalte structuri de date

- Arborii asigură o implementare eficientă a tuturor operațiilor elementare asupra structurilor de date
- Traversarea unui arbore nu este însă la fel de rapidă ca și celelalte operații, dar traversările nu sunt utilizate foarte frecvent în aplicațiile care presupun memorarea unor baze mari de date



Concluzii

- Arborii sunt alcătuiți din noduri, unite prin muchii
- Rădăcina este nodul cel mai din vârf al unui arbore; nu are un nod părinte
- În orice arbore binar, un nod are cel mult doi fii

Concluzii

- Într-un arbore binar de căutare, toate nodurile care sunt descendenți stângi ai unui nod A au chei mai mici decât A ; toate nodurile care sunt descendenți dreپți ai lui A au chei mai mari (sau egale) față de A
- Arborii permit executarea căutărilor, inserărilor și ștergerilor într-un timp $O(\log N)$



Concluzii

- Nodurile reprezintă entități de date memorate în cadrul arborelui
- Muchiile se reprezintă prin pointeri către fiii fiecărui nod
- Traversarea unui arbore presupune vizitarea tuturor nodurilor sale, într-o anumită ordine
- Cele mai simple traversări sunt cele în preordine, inordine și postordine

Concluzii

- Un arbore este neechilibrat dacă rădăcina are mai mulți descendenți stânga decât dreapta, sau invers
- Căutarea unui nod presupune compararea valorii căutate cu cheia dintr-un anumit nod și avansul la fiul stâng al nodului, dacă valoarea căutată este mai mică, respectiv la fiul drept, dacă valoarea este mai mare decât cheia nodului

Concluzii

- Inserarea unui nod necesită găsirea locului potrivit și apoi modificarea unuia din pointerii nodului părinte, astfel încât noul nod să devină fiul acestuia
- Traversarea în inordine realizează vizitarea nodurilor dintr-un arbore de căutare în ordine crescătoare
- Traversările în preordine și postordine sunt utile la evaluarea expresiilor aritmetice

Concluzii

- Dacă un nod nu are fii, poate fi șters, atribuind pointerului corespunzător părintelui valoarea NULL
- Dacă un nod are un singur fiu, poate fi șters, conectând fiul la pointerul corespunzător părintelui aceluși fiu
- Dacă un nod are doi fii, ștergerea se efectuează prin înlocuirea acestuia cu succesorul său

Concluzii

- Succesorul nodului A este nodul cu valoarea minimă din subarborele drept al lui A (a cărei rădăcină este deci fiul drept al lui A)
- La ștergerea unui nod cu doi fii, apar mai multe cazuri, după cum succesorul este chiar fiul drept al nodului șters sau un descendent stâng al acestuia