



Universitatea Politehnica din București
Facultatea de Automatică și Calculatoare
Departamentul de Calculatoare



ARBORI B ȘI MEMORAREA EXTERNĂ A DATELOR

Memorarea externă a datelor

- Arborii 2-3-4 sunt un exemplu de arbori multicăi, care au mai mult de doi fii și, totodată, mai mult de un singur element într-un nod
- Un alt tip de arbori multicăi, **arborii B**, sunt utili la memorarea datelor pe suport extern
- Prin suport extern se înțelege un hard disk

Accesul la date cu suport extern

- Dezavantajul principal al memorării externe este cel al vitezei mult mai scăzute de acces față de memoria RAM
- Această diferență de viteză impune utilizarea unor tehnici eficiente de memorare externă

Exemplu

- O aplicație de baze de date pentru a prelucra datele despre 500.000 de abonați, unde fiecare abonat este identificat prin nume, adresă, număr de telefon și alte informații
- Se presupune că memorarea unui abonat se realizează cu ajutorul unei înregistrări care ocupă 512 octeți

Exemplu

- Dimensiunea totală a fișierului este $500.000 * 512 = 256.000.000$ octeți, adică 256 MB
- Se presupune că acest fișier este prea mare pentru a fi stocat integral în memoria RAM, dar poate fi memorat pe disc

Acces lent la disc

- Datele sunt memorate pe piste circulare ale unui disc aflat în rotație
- Pentru a accesa o anumită informație de pe disc, capul de citire-scriere trebuie mai întâi deplasat la pista pe care se găsește acea informație

Acces lent la disc

- După localizarea pistei corecte, capul de citire-scriere trebuie să aștepte rotirea informației în poziția corectă
- Aceasta durează, în medie, o jumătate de rotație
- După poziționarea capului de citire-scriere, începe operația de citire (sau de scriere) propriu-zisă
- Timpul de acces la disc este de ≈ 10 ms

Accesul la disc

- După ce este corect poziționat și operația de citire (sau scriere) a început, unitatea de disc permite transferul unor blocuri mari de date în memorie
- Datele sunt memorate în părți numite blocuri, pagini sau unități de alocare, în funcție de sistemul de operare

Accesul la disc

- Unitatea de disc citește sau scrie întotdeauna minimum un bloc la un moment dat
- Dimensiunea blocului variază în funcție de sistemul de operare, dimensiunea discului și alți factori, fiind de obicei o putere a lui 2

Exemplu

- Se presupune că un bloc are dimensiunea de 8.192 octeți (2^{13})
- Baza de date va necesita un număr de blocuri egal cu câtul dintre dimensiunea totală în octeți (256.000.000) și dimensiunea unui bloc (8.192), adică 31.250 de blocuri

Exemplu

- Se asigură o eficiență maximă când se efectuează o operație de citire sau de scriere a unui număr întreg de blocuri
- Dacă se dorește citirea a 100 de octeți, sistemul de operare va citi un bloc de 8.192 de octeți, păstrând apoi doar cei 100 de octeți necesari

Exemplu

- Dacă se dorește citirea a 8.200 de octeți, se vor citi două blocuri, adică 16.384 de octeți, renunțând apoi la aproape jumătate din informația citită
- Organizând programul astfel încât să lucreze cu un bloc de date la un moment dat, se pot optimiza performanțele acestuia

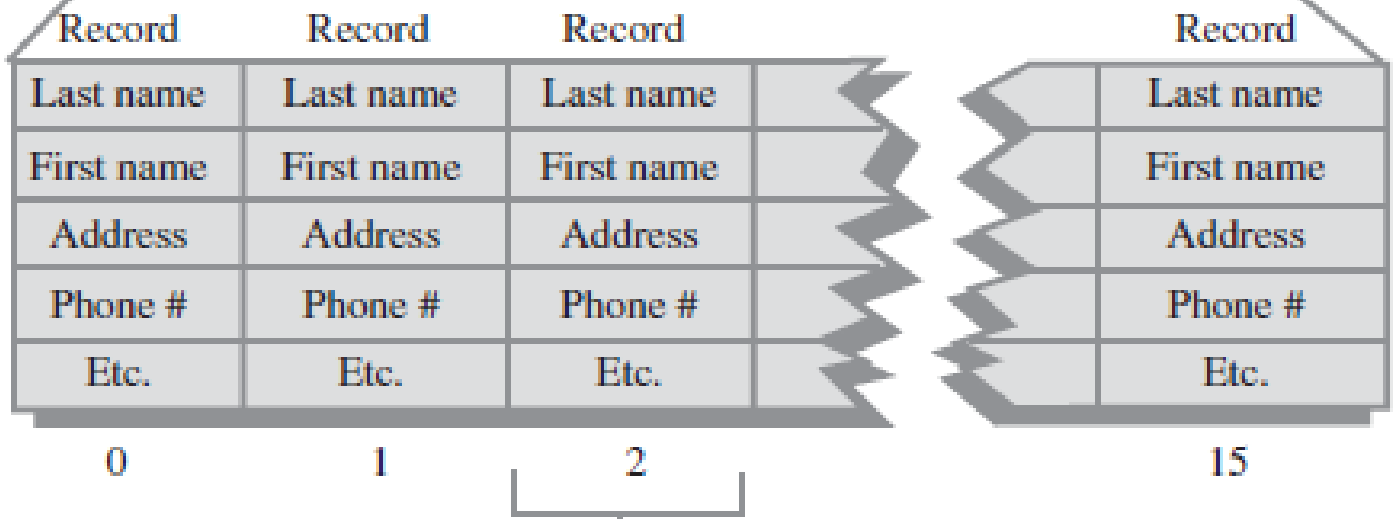
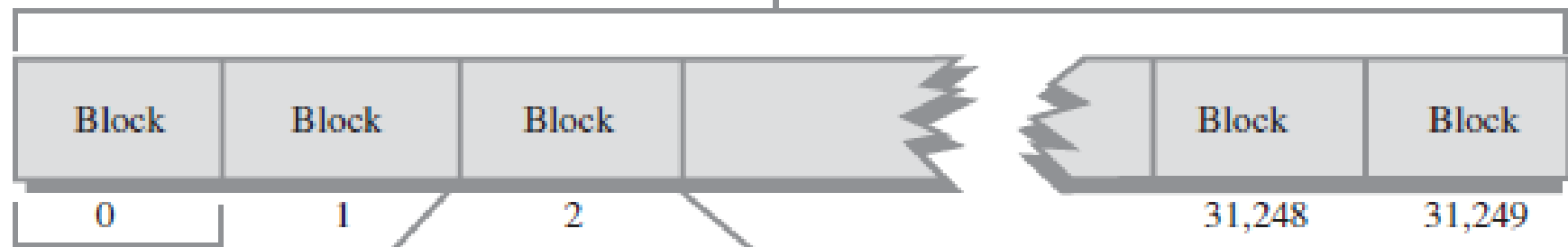
Exemplu

- Se pot memora 16 înregistrări în fiecare bloc ($8.192 / 512 = 16$)
- Pentru o eficiență maximă, trebuie citite câte 16 înregistrări deodată (sau multipli ai lui 16)
- Este util ca dimensiunea unei înregistrări să fie o putere a lui 2
- Astfel, se asigură potrivirea unui număr întreg de înregistrări într-un bloc

Observații

- După poziționarea capului de citire-scriere, citirea unui bloc este destul de rapidă, executându-se în câteva milisecunde
- Timpul de acces necesar pentru citirea sau scrierea unui bloc de pe disc nu depinde foarte mult de dimensiunea blocului
- Eficiența operației de citire sau de scriere a unei singure înregistrări pe disc crește proporțional cu dimensiunea blocului

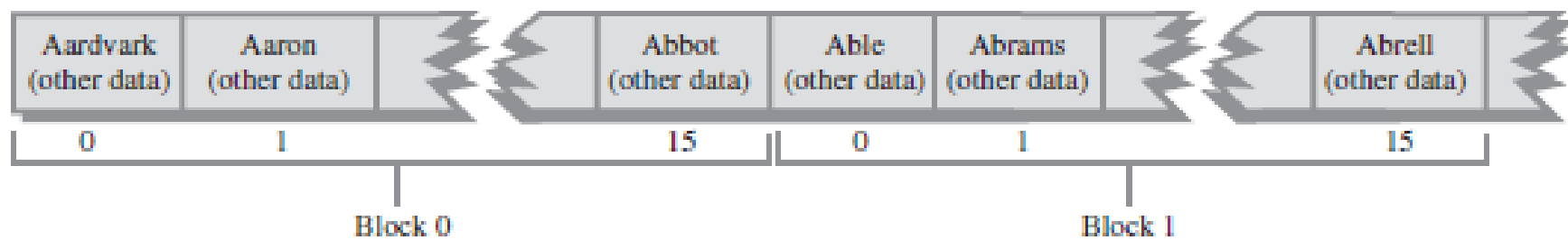
File = 256,000,000 bytes



512 Bytes

Ordonare secvențială

- O modalitate de a dispune informația din baza de date într-un fișier este de a ordona toate înregistrările după un anumit criteriu, de exemplu ordinea alfabetică



Căutarea

- Pentru a căuta o anumită înregistrare într-un fișier ordonat secvențial, se poate utiliza metoda de căutare binară
- Se citește un bloc de înregistrări din mijlocul fișierului
- Cele 16 înregistrări sunt temporar păstrate într-o zonă de 8.192 de octeți din memoria RAM

Exemplu

- Dacă valorile cheilor din aceste înregistrări sunt (în ordinea alfabetică) înaintea cheii căutate, ne vom deplasa la punctul care marchează trei sferturi din lungimea totală a fișierului, citind un bloc de acolo
- Dacă însă cheile apar după cea căutată, deplasarea se va efectua la punctul care corespunde unui sfert din lungimea fișierului

Exemplu

- Prin înjumătățirea repetată a domeniului, se detectează cheia căutată
- Căutarea binară în memorie presupune efectuarea a $\log_2 N$ comparații
- Pentru 500.000 de chei, se obțin 19 comparații
- Dacă fiecare comparație se efectuează în $10 \mu s$, timpul total va fi de $190 \mu s$

Exemplu

- Dacă datele sunt memorate pe disc, din cauză că fiecare acces la disc durează mult, este mai relevant să se urmărească numărul acceselor la disc, decât numărul total de înregistrări
- Timpul necesar pentru a citi un bloc de înregistrări este mult mai mare decât timpul în care se poate căuta o cheie în cele 16 înregistrări din bloc, după transferul blocului în memorie

Observații

- Accesul la disc este mult mai lent decât accesul la memorie
- Se citește un bloc întreg deodată, iar numărul blocurilor este mult mai mic decât cel al înregistrărilor
- Pentru exemplul cu 31.250 de blocuri, $\log_2 31250 \approx 15$
- Sunt necesare 15 accese la disc, pentru a găsi o înregistrare dată

Observații

- În practică, acest număr se micșorează, din cauză că se citesc 16 înregistrări dintr-o dată
- În primii pași ai căutării binare, înregistrările multiple din memorie nu sunt de prea mare folos, deoarece următorul acces la fișier se va efectua la o distanță foarte mare de cel curent

Observații

- Când se ajunge aproape de cheia căutată, următoarea înregistrare dorită se poate afla deja în memorie, fiind parte a aceluiași bloc de 16
- Numărul necesar de comparații se va reduce cu aproximativ 2
- Se efectuează aproximativ 13 accese la disc, care durează fiecare 10 ms
- Timpul total este de aproximativ 130 ms

Inserarea

- Când se dorește inserarea (sau ștergerea) unui element dintr-un fișier ordonat secvențial, ambele operații presupun deplasarea, în medie, a jumătate din înregistrări și, în consecință, a jumătate din blocuri

Inserarea

- Deplasarea unui bloc necesită două accese la disc, unul pentru citirea blocului și celălalt pentru scriere
- După localizarea punctului de inserare, blocul în care acesta se găsește este transferat într-o zonă de memorie

Inserarea

- Ultima înregistrare din bloc este salvată, după care se deplasează numărul corespunzător de înregistrări, pentru a face loc noii înregistrări, care va fi inserată
- Conținutul zonei este scris înapoi pe disc
- Se citește apoi următorul bloc în memorie

Inserarea

- Ultima sa înregistrare este salvată, iar toate celelalte se deplasează cu o poziție spre dreapta, după care ultima înregistrare a blocului precedent este inserată la începutul zonei
- Conținutul zonei este scris din nou pe disc
- Operația va continua până când toate blocurile, situate după locul de inserare, vor fi scrise înapoi pe disc

Exemplu

- Se presupune că sunt 31.250 de blocuri
- Va trebui să citim și să rescriem, în medie, 15.625 dintre ele
- La 10 ms pentru fiecare citire sau scriere, rezultă că fiecare inserare durează mai mult de 5 minute

Observații

- O altă problemă a ordonării secvențiale este că permite căutarea rapidă după o singură cheie
- Dacă se dorește aflarea persoanei cu un anumit număr de telefon, nu se poate utiliza căutarea binară, deoarece datele sunt ordonate după nume

Observații

- Trebuie parcurs întregul fișier, bloc cu bloc, utilizând accesul secvențial
- Aceasta presupune citirea, în medie, a jumătate dintre blocuri, operație care durează aproximativ 150 s, o performanță foarte slabă pentru o căutare
- Este necesar un mod mai eficient de memorare a informației pe disc

Arbori B

- Arborii utilizați pentru memorarea externă se numesc **arbori B**
- Arborii B au fost concepuți ca structuri eficiente pentru memorarea externă de R. Bayer și E. M. McCreight, în 1972
- Un arbore B este asemănător cu un arbore 2-3-4, dar are mai multe elemente în fiecare nod

Un bloc în fiecare nod

- Accesul la disc are o eficiență maximă când se citește sau se scrie un bloc întreg dintr-o dată
- Într-un arbore, entitatea care conține elemente este un nod
- Se memorează un întreg bloc de date în fiecare nod al unui arbore

Observații

- Citirea unui nod va asigura accesul la o cantitate maximă de informație, într-un timp minim
- Într-un arbore, trebuie memorate și legăturile către alte noduri (deci către alte blocuri, deoarece un nod corespunde unui bloc)

Observații

- Într-un arbore stocat în memorie, aceste legături sunt pointeri către noduri aflate în alte zone de memorie
- Pentru un arbore stocat într-un fișier pe disc, legăturile vor fi numerele unor blocuri din fișier (cuprinse între 0 și 31.249)

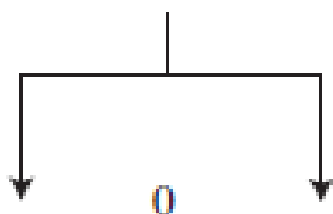
Observații

- Nu pot fi compactate 16 înregistrări de 512 octeți într-un bloc, din cauza spațiului ocupat de legăturile către nodurile fii
- Se poate reduce numărul înregistrărilor din bloc la 15, pentru a crea loc pentru legături, dar este mai eficient ca numărul înregistrărilor dintr-un nod să fie par, deci se reduce dimensiunea unei înregistrări la 507 octeți

Observații

- Nodul va conține 17 legături către noduri fii (cu una mai mult decât numărul de elemente), care vor ocupa 68 de octeți ($17 * 4$)
- Astfel, rămâne suficient loc pentru 16 înregistrări de câte 507 octeți, 12 octeți rămânând neutilizați ($507 * 16 + 68 = 8.180$)
- Figura prezintă un bloc dintr-un arbore B, împreună cu nodul corespunzător acestuia

Block
numbers

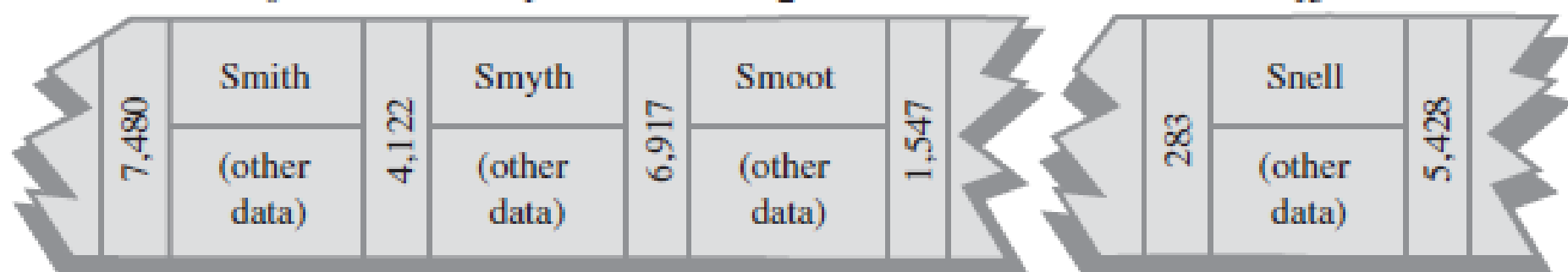


0

1

2

15



0

1

2

3

15

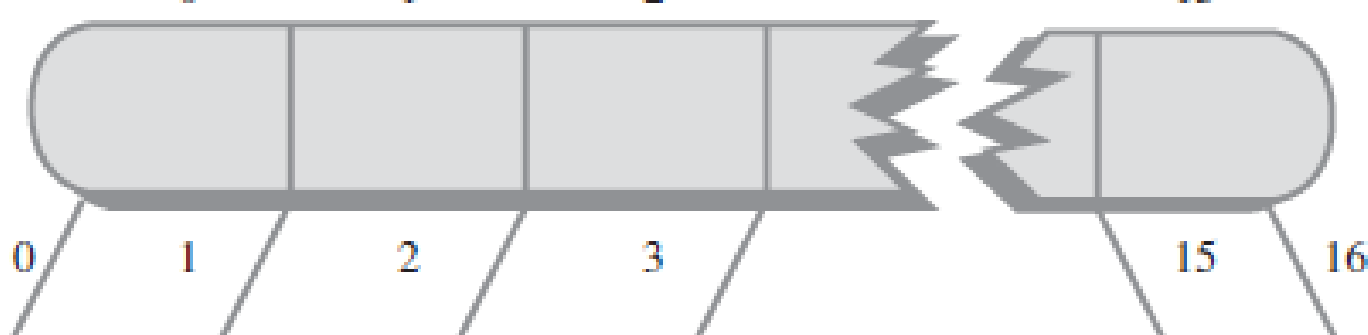
16

0

1

2

15



0

1

2

3

15

16

Observații

- În fiecare nod, datele sunt ordonate după chei, ca într-un arbore 2-3-4
- Structura unui arbore B este similară cu cea a unui arbore 2-3-4, cu excepția faptului că există mai multe elemente în fiecare nod și mai multe legături către fii

Definiție

- **Ordinul** unui arbore B reprezintă **numărul maxim de fii** pe care îi poate avea un anumit nod
- În exemplu, arborele B are ordinul 17

Proprietăți

- Un arbore **B** de ordinul **n** are următoarele proprietăți:
- **1.** Rădăcina fie nu are succesori, fie are cel puțin doi succesori
- **2.** Fiecare nod interior (altele decât rădăcina și frunzele) au între $n/2$ și n succesori
- **3.** Toate căile de la rădăcină la frunze au aceeași lungime

Căutarea

- Căutarea unei înregistrări cu o anumită cheie se efectuează la fel ca într-un arbore 2-3-4
- Mai întâi, se citește într-o zonă de memorie blocul care conține nodul rădăcină

Căutarea

- Se examinează pe rând fiecare din cele 15 înregistrări (sau, dacă nodul nu este complet, se examinează toate înregistrările existente), începând cu cea numerotată cu 0
- Când se găsește o cheie mai mare decât valoarea căutată, se continuă cu fiul a căruia legătură se află între înregistrarea curentă și cea precedentă

Căutarea

- Procesul continuă până la găsirea nodului căutat
- Dacă se ajunge la un nod frunză fără a găsi cheia, aceasta nu se află în arbore

Inserarea

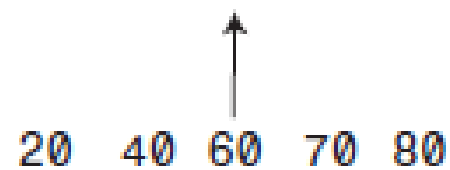
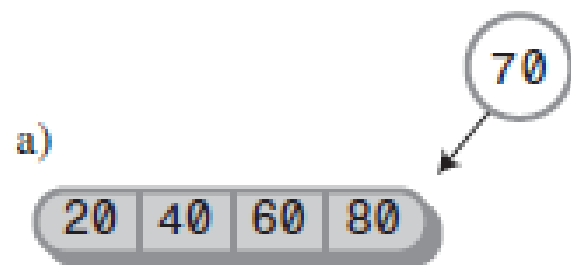
- Operația de inserare într-un arbore B este diferită față de inserarea într-un arbore 2-3-4
- Într-un arbore 2-3-4, există multe noduri incomplete, care conțin un singur element
- Divizarea unui nod generează întotdeauna două noduri cu câte un singur element
- Această soluție nu este convenabilă în cazul arborilor B

Inserarea

- Într-un arbore B, este foarte important ca nodurile să fie cât mai pline posibil, astfel încât fiecare acces la disc, care va citi un nod întreg, să acceseze o cantitate maximă de informație
- În acest scop, operația de inserare diferă de inserarea în arborii 2-3-4 prin 3 aspecte

Inserarea

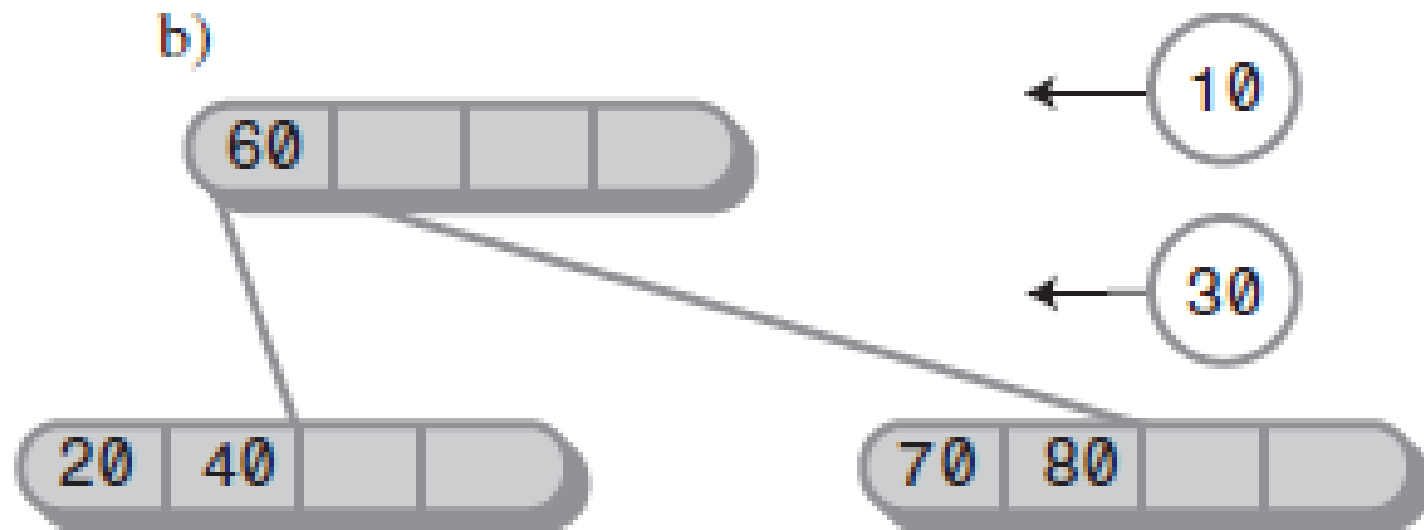
- **1.** Divizarea unui nod împarte elementele în mod egal – jumătate sunt distribuite nodului nou creat, iar cealaltă jumătate rămân în cel vechi
- **2.** Divizările de noduri se efectuează de jos în sus
- **3.** Elementul care va fi inserat în nodul părinte nu este cel din mijlocul nodului divizat, ci cel din mijlocul secvenței formate din elementele nodului și noul element



Observații

- Nodul rădăcină este deja complet
- Elementele cu cheile 20, 40, 60 și 80 au fost deja inserate în arbore
- Se inserează nodul cu cheia 70, fiind necesară divizarea rădăcinii
- Din cauză că nodul divizat este chiar rădăcina, se creează două noduri noi: o rădăcină nouă și un nod aflat la dreapta celui divizat

b)



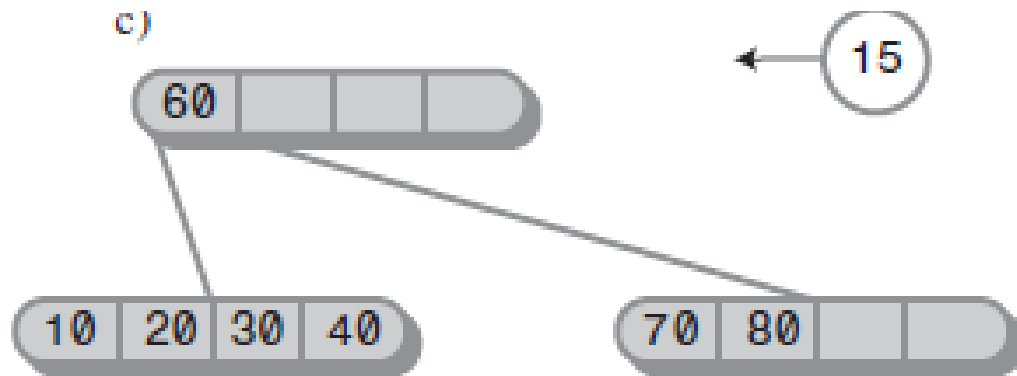
Observații

- Pentru a stabili locul în care se deplasează cele 5 elemente, algoritmul ordonează crescător valorile acestora, într-o zonă de memorie internă
- Patru dintre acestea provin de la nodul divizat, iar a cincea este valoarea inserată

Observații

- Elementul central al secvenței, 60, este inserat în noua rădăcină
- Toate elementele situate la stânga celui central rămân în nodul care a fost divizat, iar cele de la dreapta sunt inserate în nodul din dreapta
- Dacă se inserează încă două elemente, 10 și 30, acestea vor completa fiul stâng

c)



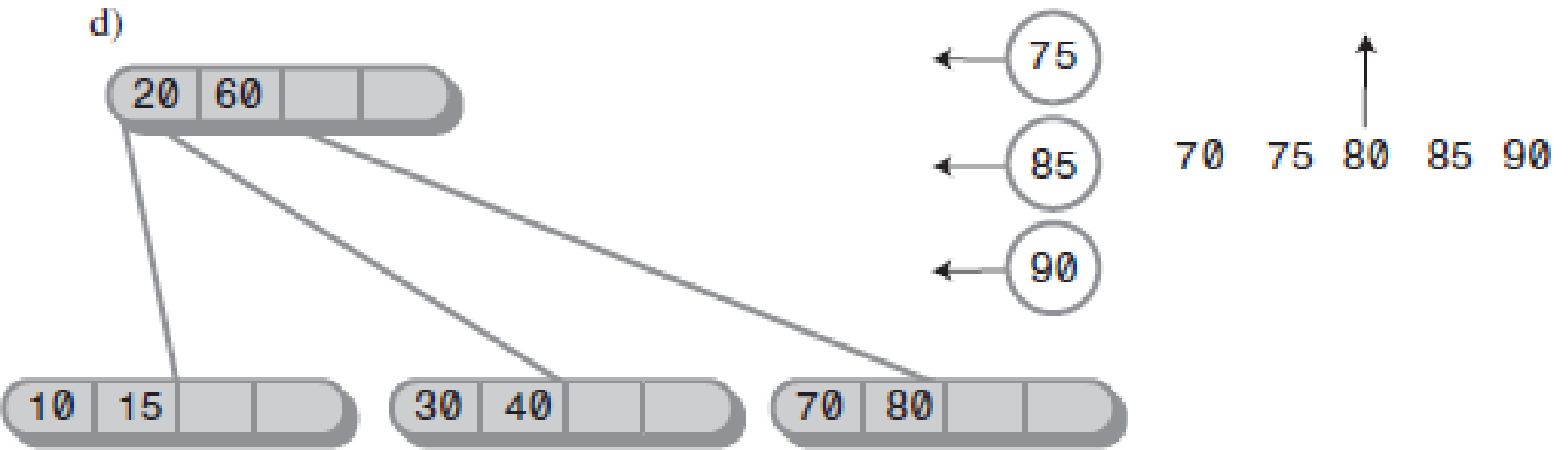
10 15 20 30 40

A horizontal sequence of numbers: 10, 15, 20, 30, 40. An arrow points upwards from the number 20.

Observații

- Următorul element inserat, 15, va diviza fiul stâng
- Elementul 20 se deplasează în sus, în nodul rădăcină

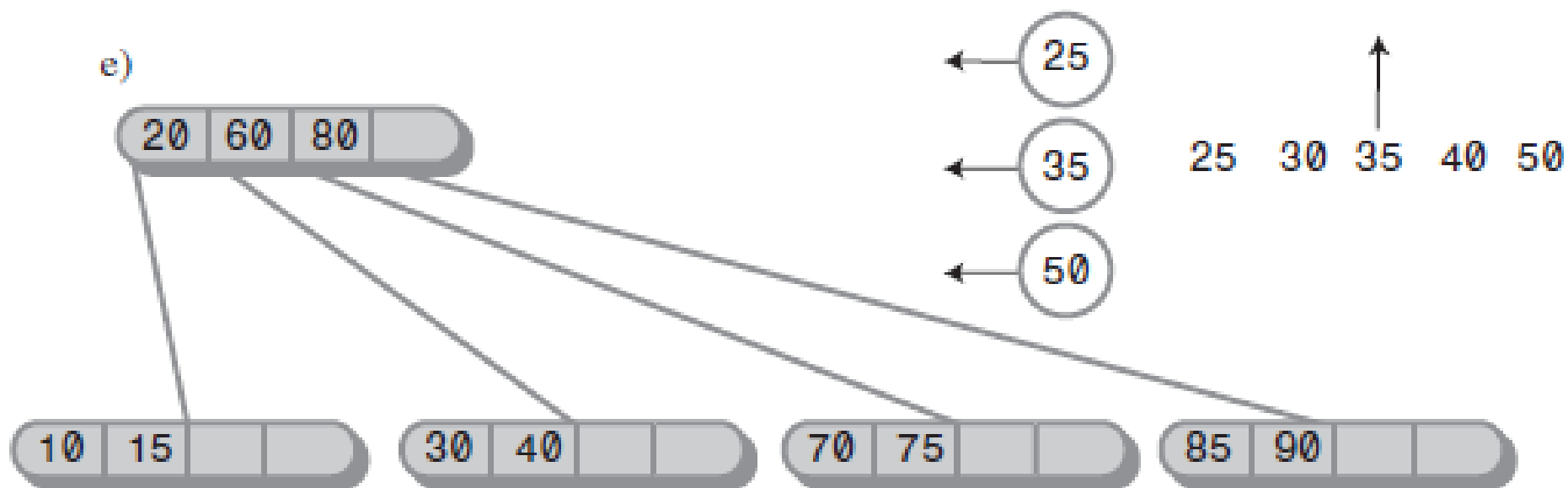
d)



Observații

- La inserarea a încă trei elemente, 75, 85 și 90, primele două completează cel de-al treilea fiu, iar al treilea determină divizarea acestuia, operație în urma căreia se creează un nod nou, iar elementul central, 80, se deplasează în rădăcină

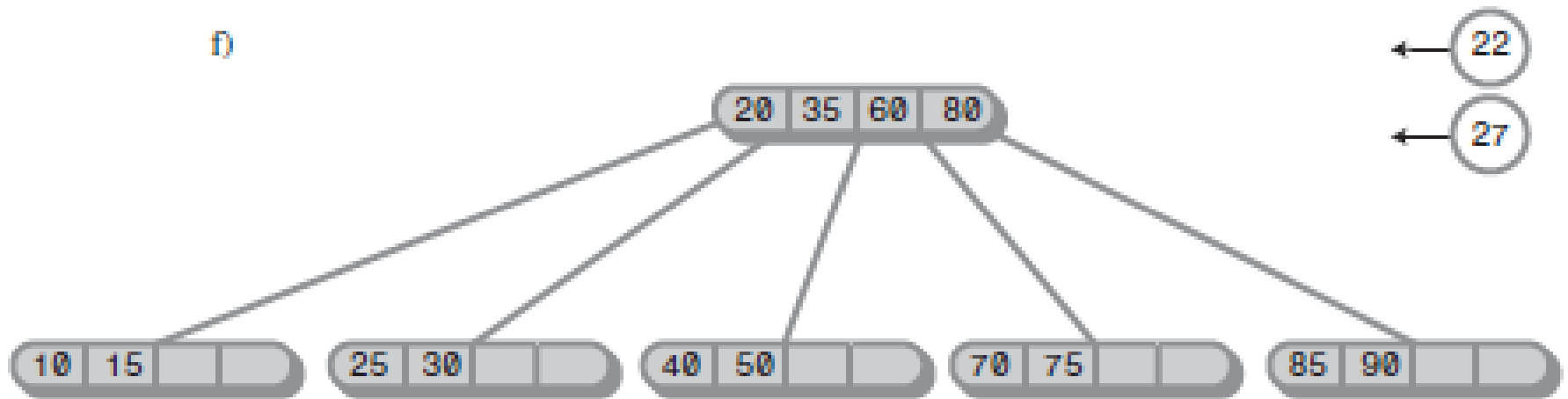
e)



Observații

- Dacă se adaugă încă trei elemente, cu valorile 25, 35 și 50, primele două completează al doilea fiu, iar al treilea îi determină divizarea, în urma căreia se creează un nod nou, iar elementul central, 35, se deplasează în rădăcină

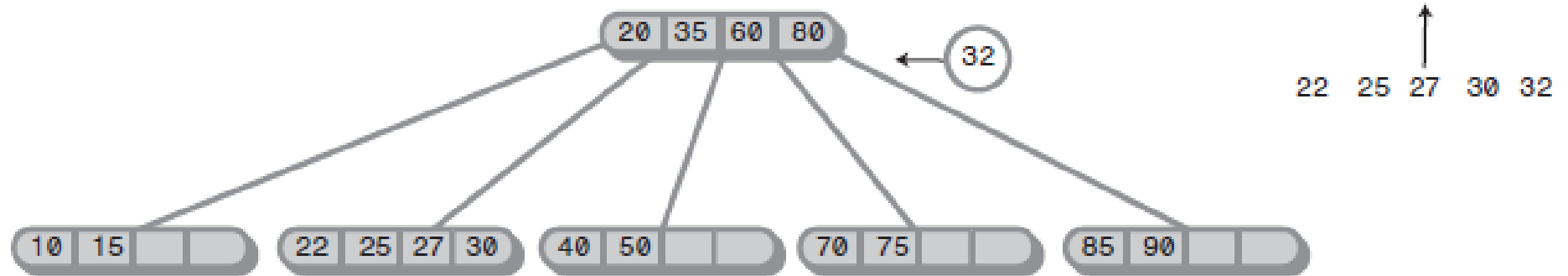
f)



Observații

- Rădăcina este completă
- Inserările ulterioare nu vor determina neapărat o divizare, deoarece nodurile se divid numai la inserarea unui element într-un nod complet, nu și atunci când se întâlnesc noduri complete, la parcurgerea descendentă a arborelui
- 22 și 27 se inserează în cel de al doilea fiu, fără a mai produce divizări

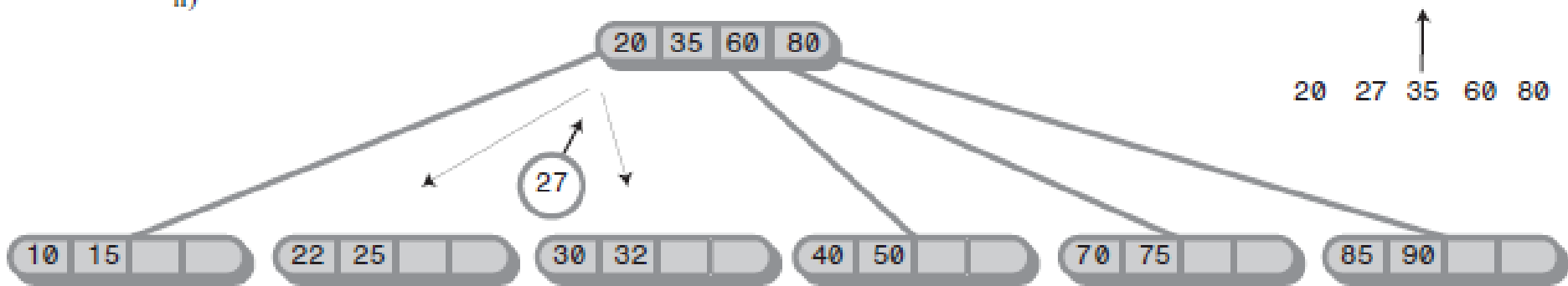
g)



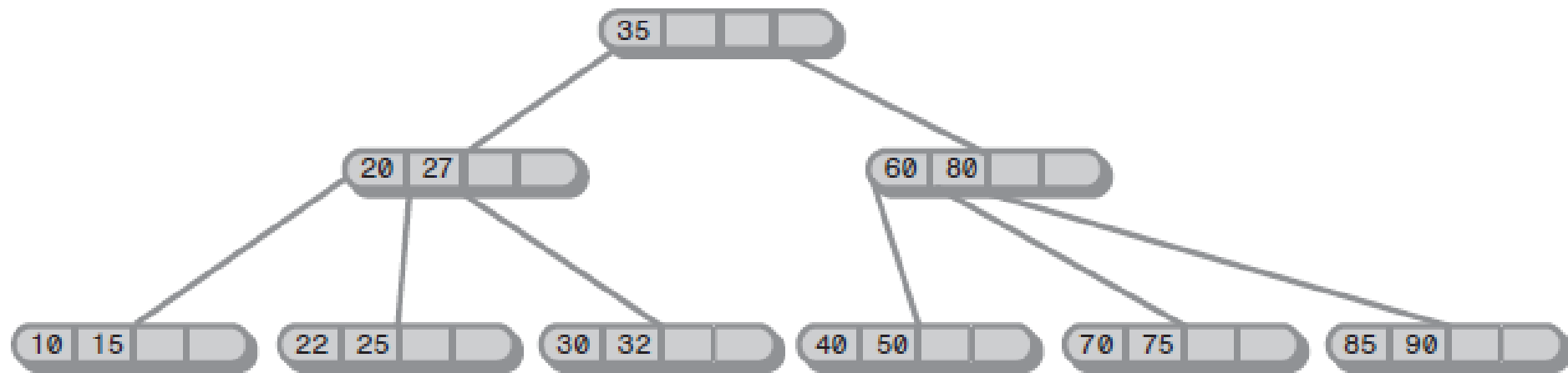
Observații

- Următorul element inserat, 32, produce două divizări
- Al doilea nod fiu este complet, deci va fi divizat
- Elementul 27, care se deplasează în sus în urma divizării, nu poate fi inserat în rădăcina completă
- Rădăcina trebuie să fie divizată

h)



i)



Observații

- Pe parcursul inserării, niciun nod (cu excepția rădăcinii) nu poate conține mai puțin de jumătate din numărul maxim de elemente
- Multe dintre noduri sunt aproape complete
- Aceasta conduce la o creștere a eficienței, din cauză că la accesul care citește un nod, se citește întotdeauna o cantitate substanțială de informații

Eficiența arborilor B

- Din cauză că fiecare nod conține multe înregistrări, iar fiecare nivel conține multe noduri, operațiile asupra arborilor B sunt rapide, ținând cont că datele sunt memorate pe disc

Exemplu

- Baza de date are 500.000 de înregistrări
- Toate nodurile din arborele B sunt cel puțin pe jumătate complete, deci conțin cel puțin 8 înregistrări și 9 legături către fii
- Înălțimea arborelui este mai mică decât $\log_9 N$, unde $N = 500.000$
- $\log_9 500.000 = 5,972$
- Arborele necesită numai 6 niveluri

Observații

- Utilizând un arbore B, sunt suficiente 6 accese la disc, pentru a găsi orice înregistrare dintr-un fișier cu 500.000 de înregistrări
- Dacă fiecare acces durează 10 ms, totalul este de 60 ms
- Timpul este mult mai bun decât în cazul căutării binare printr-un fișier ordonat secvențial

Observații

- Cu cât fiecare nod conține mai multe înregistrări, cu atât numărul nivelurilor din arbore va fi mai mic
- Arborele B din exemplu are 6 niveluri, chiar dacă fiecare nod conține maxim 16 înregistrări
- Comparativ, un arbore binar cu 500.000 de elemente are 19 niveluri, iar un arbore 2-3-4 are 10 niveluri

Observații

- Utilizând blocuri cu sute de înregistrări, se poate reduce numărul nivelurilor din arbore, îmbunătățind astfel timpii de acces
- Căutarea este mult mai rapidă în arborii B decât în cazul fișierelor secvențiale, iar avantajul arborilor B se observă la operațiile de inserare și de ștergere

Observații

- Pentru inserarea într-un arbore B, în cazul în care nu se divizează niciun nod, 6 accese sunt suficiente pentru găsirea punctului de inserare
- Se mai efectuează încă un acces pentru a scrie înapoi pe disc blocul cu noua înregistrare inserată
- Se obțin în total 7 accese

Observații

- Dacă nodul trebuie divizat, acesta este mai întâi citit, jumătate din înregistrările sale sunt apoi șterse, după care blocul este scris înapoi pe disc
- Nodul nou creat trebuie scris pe disc, după care se citește nodul său părinte, în care se inserează o înregistrare, blocul fiind scris înapoi pe disc
- Operația presupune 5 accese în plus, față de cele 6 necesare pentru a găsi locul de inserare

Concluzii

- Prin memorare externă se înțelege stocarea datelor în afara memoriei RAM, de regulă pe un disc
- Suportul extern are capacitate de memorare mai mare, este mai ieftin (raportat la un octet), dar mai lent decât memoria RAM

Concluzii

- Datele de pe suport extern sunt de regulă transferate în memorie, câte un bloc la fiecare moment de timp
- Datele pot fi aranjate pe suport extern în ordine secvențială
- Această soluție conduce la timpi buni de căutare, dar la inserări și ștergeri foarte lente

Concluzii

- Un arbore B este un arbore multicăi, în care fiecare nod poate avea zeci sau sute de chei și noduri fii
- Fiecare nod al unui arbore B are un număr de fii și un număr de chei mai mic cu o unitate decât numărul de fii