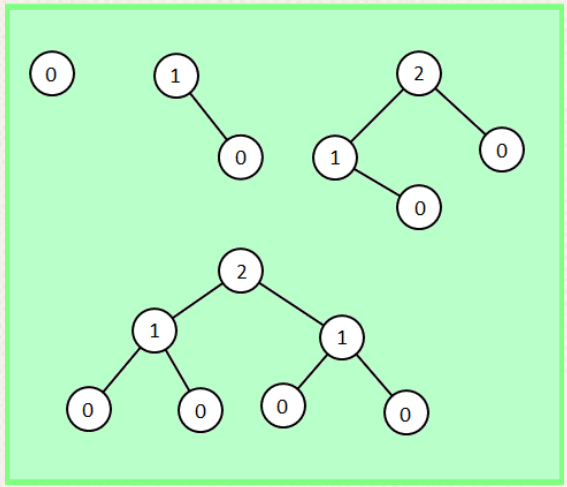
B - Arbori

Arbori echilibrați

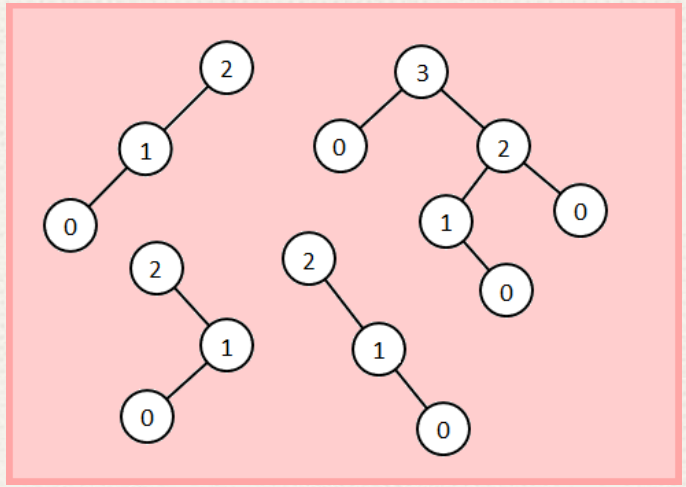
Un **arbore echilibrat** este un arbore în care, **pentru orice nod**, diferența dintre înălțimile subarborilor stâng și drept este de maxim 1.

Exemple:

Echilibrați



Neechilibrați



B-Arbori

Un **B-Arbore** este un arbore echilibrat, destinat căutării eficiente de informație.

Un B-Arbore poate avea mai mult de 2 fii pentru un nod (se poate ajunge și la ordinul sutelor).

Totuși, înălțimea arborelui rămâne O(log n), datorită unei baze a logaritmului convenabilă.

În practică, B-Arborii sunt folosiți pentru baze de date și sisteme de fișiere, pentru citirea și scrierea eficientă pe discul de memorie.
O proprietate importantă a lor este faptul că rețin multă informație. De aceea, B-Arborii reduc numărul de accesări ale discului (accesarea discului este o operație costisitoare).

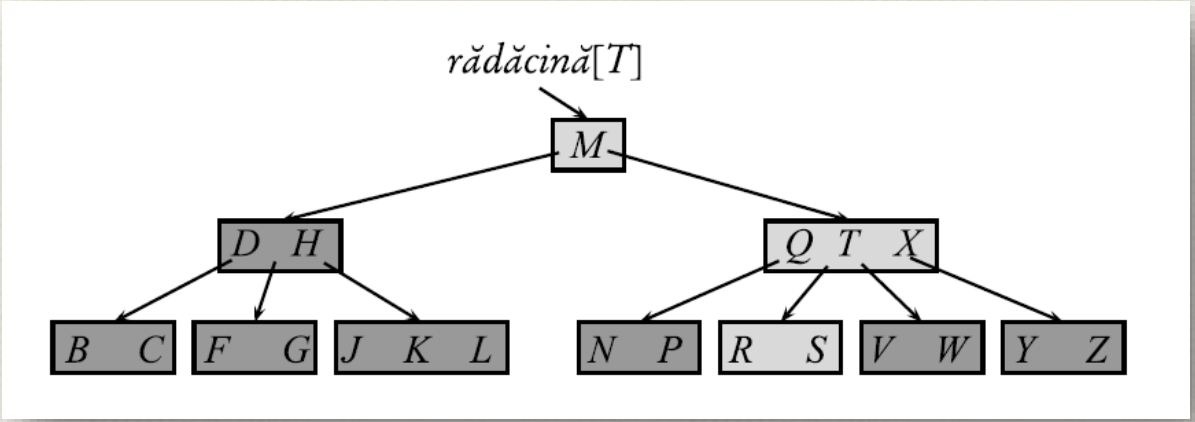
B-Arbori

Proprietăți:

1. Un nod poate să conțină mai mult de o cheie
2. Numărul de chei ale unui nod **x** este **n[x]**
3. Un nod **x** are **n[x] + 1** fii
4. Toate frunzele unui B-Arbore se află pe același nivel

B-Arbori

Exemplu:



Cheile acestui arbore sunt consoanele din alfabetul latin.
Observăm că un nod poate avea mai multe chei, iar fiecare nod **x** care are **n[x]** valori va avea **n[x] + 1** fii.

Căutarea literei ”R” este exemplificată pe traseul hașurat cu o culoare mai deschisă.

B-Arbori

Câmpurile unui nod:

1. **n[x]** – numărul de chei memorate în nodul **x**

2. cele **n[x]** chei, memorate în ordine crescătoare:
$$cheie_1[x] \leq cheie_2[x] \leq cheie_3[x] \leq ... \leq cheie_{n[x]}[x]$$

3. o valoare booleană **frunză[x]** – **True**, dacă nodul **x** este frunză, **False**, dacă nodul **x** este nod intern

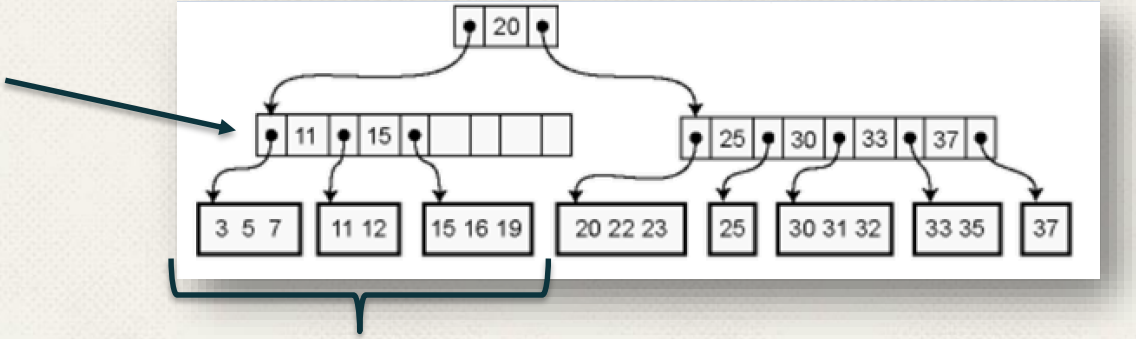
Dacă **x** este un nod intern, atunci el conține **n[x] + 1** pointeri către fiii săi. Nodurile frunză nu au fii, deci nu au aceste câmpuri definite.

B-Arbori

Cheile nodului **x** separă domeniile de chei aflate în fiecare subarbore astfel:

- dacă $\mathbf{k_i}$ este o cheie oarecare memorată într-un subarbore cu rădăcina $\mathbf{fiu_i[x]}$, atunci
  $k_1 \le cheie_1[x] \le k_2 \le cheie_2[x] \le ... \le cheie_{n[x]}[x] \le k_{n[x]+1}$

Nodul x



3, 5, 7 ≤ 11
11 ≤ 11, 12
11, 12 ≤ 15
15 ≤ 15, 16, 19

B-Arbori

Gradul unui B-Arbore

Există o limitare inferioară și una superioară a numărului de chei ce pot fi conținute într-un nod. Exprimăm aceste margini printr-un întreg fixat $t \geq 2$, numit ***grad minim*** al B-Arborelui.

B-Arbori

Restricții:

1. Fiecare nod, cu excepția rădăcinii, trebuie să aibă **cel puțin *t – 1* chei**.
   Consecință: fiecare nod intern trebuie să aibă **cel puțin *t*** fii.

2. Dacă arborele este nevid, atunci rădăcina trebuie să aibă cel puțin o cheie.

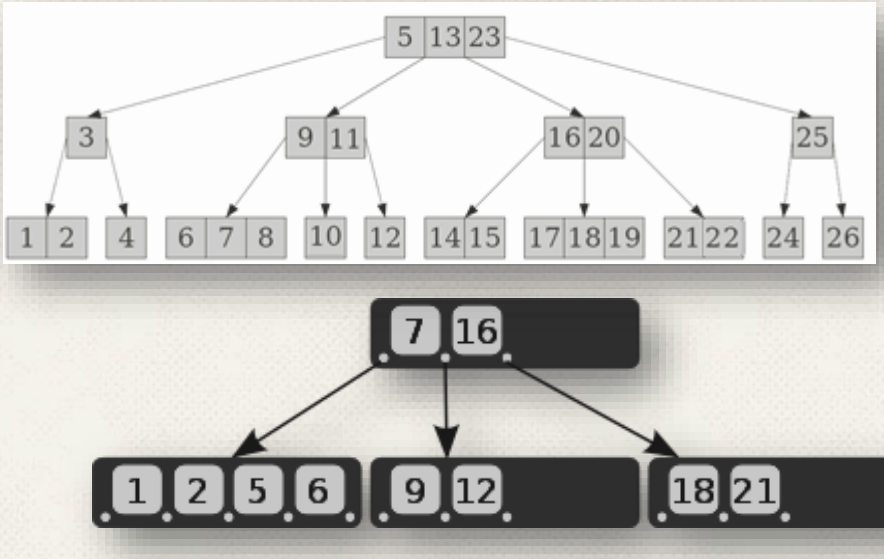
3. Fiecare nod poate să aibă **cel mult *2t – 1*** chei.
   Consecință: orice nod intern poate să aibă **cel mult *2t*** fii.

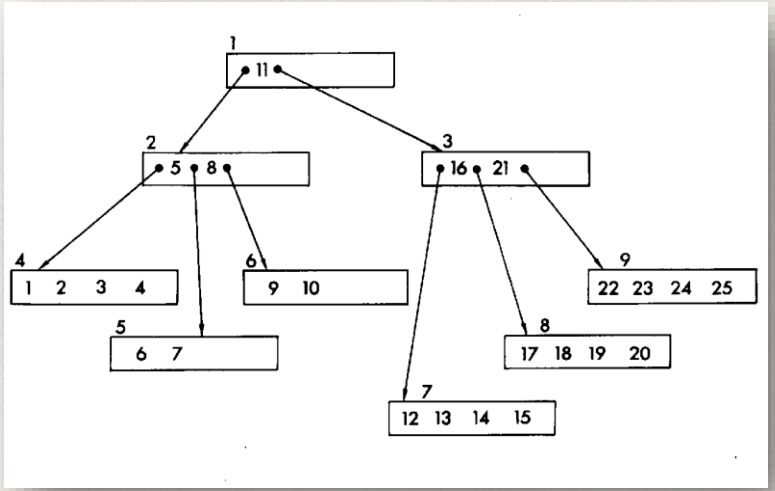
Un nod cu *2t – 1* chei se numește **nod plin**.

B-Arbori

Exemple de B-Arbori:

B-Arbore de ordin 2 (arbore 2-3-4)





Înălțimea unui B-Arbore

Teoremă: Dacă n ≥ 1, atunci, pentru orice B-Arbore T cu **n** chei și grad minim *t* ≥ *2:*

$$h \leq \log_t \frac{n+1}{2}$$ - înălțimea arborelui

Demonstrație: Dacă un B-Arbore are înălțimea h, atunci va avea număr minim de chei **dacă** rădăcina conține o singură cheie, iar toate celelalte noduri câte t – 1 chei.
În acest caz, există:

- pe nivelul 1: 2 noduri
- pe nivelul 2: 2t noduri
- pe nivelul 3: $2t^2$ noduri
- ...
- pe nivelul h: $2t^{h-1}$

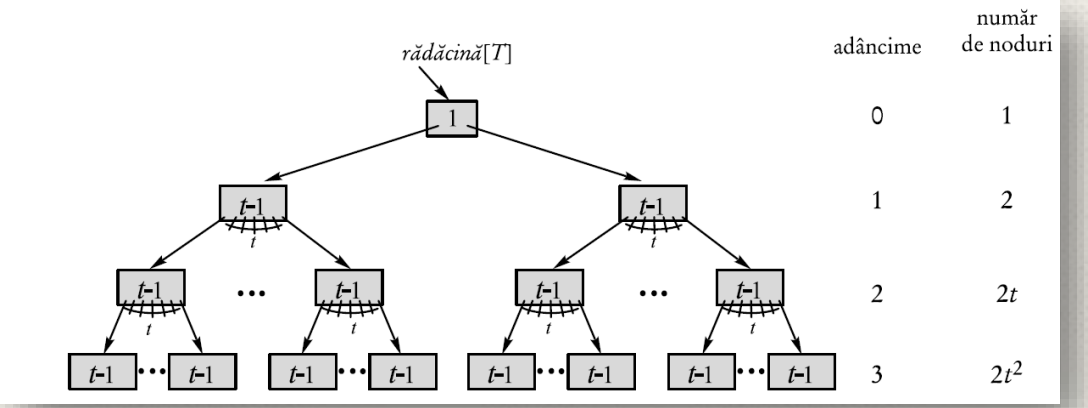
Deci:

$$n \geq 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1)\left(\frac{t^h - 1}{t-1}\right) = 2t^h - 1$$

Înălțimea unui B-Arbore

Exemplu:

Pentru un B-Arbore de înălțime 3, cu număr minim de chei, care are, în fiecare nod, numărul de chei reținute **n[x]**, avem următorul desen:



Înălțimea unui B-Arbore

Concluzie:

Înălțimea unui arbore este

$$h \leq \log_t \frac{n+1}{2}$$

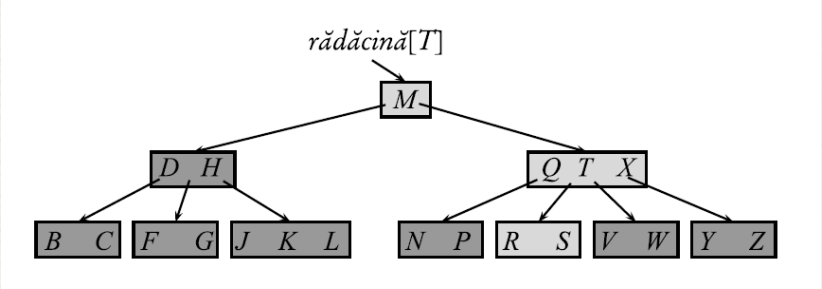
Deci, înălțimea unui B-Arbore crește proporțional cu **O(log n).**

Discuție

Exerciții:

1. De ce nu putem permite gradul minim $t = 1$?

2. Pentru ce valori ale lui t, arborele de mai jos este un B-Arbore, conform definiției?



1. Desenați toți B-Arborii corecți cu grad minim 2 care să reprezinte mulțimea {1, 2, 3, 4, 5}.

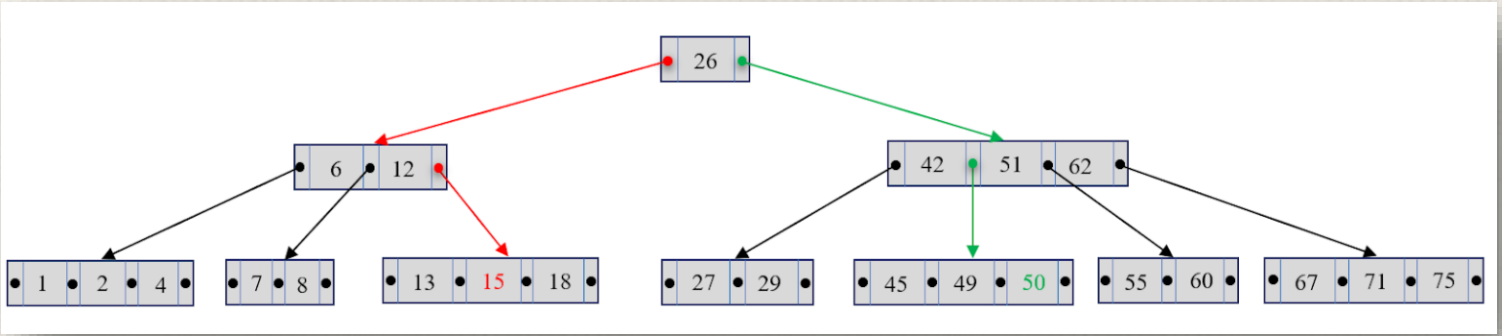
Operații de bază

Căutarea în B-Arbore

Căutarea într-un B-Arbore este asemănătoare cu o căutare într-un arbore binar.

Într-un B-Arbore, căutarea se realizează comparând cheia căutată **x** cu cheile nodului curent, plecând de la nodul rădăcină.

Căutare reușită pentru 50 și nereușită pentru 17.

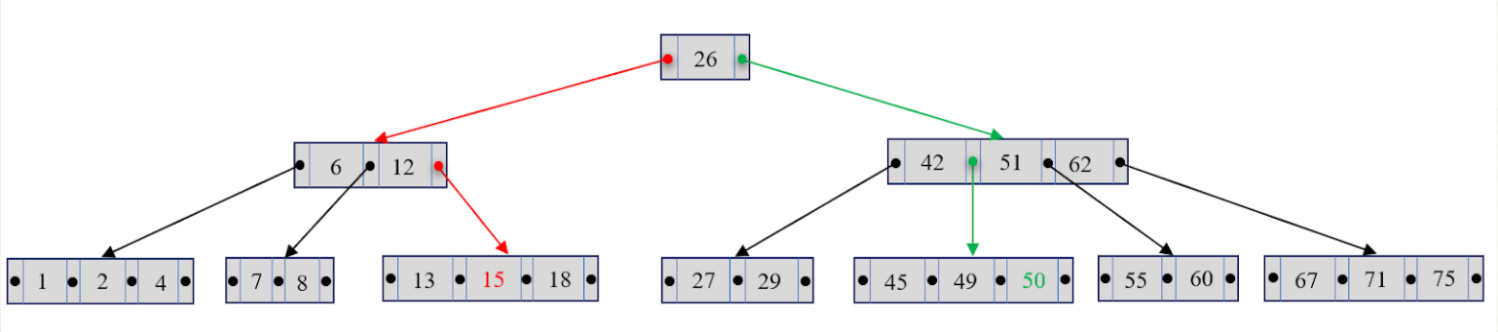


Căutarea în B-Arbore

Algoritm:

1. Căutăm cheia **x** în rădăcină
2. Dacă nu o găsim, atunci continuăm căutarea în fiul corespunzător valorii **x**
3. Dacă găsim cheia, returnăm perechea de valori (y, i), reprezentând nodul, respectiv poziția în nod pe care s-a găsit valoarea **x**.

Putem afla indicele fiului care trebuie explorat în continuare la pasul 2 folosind **căutarea binară dacă numărul de valori din fiecare nod este mare**.



Căutarea în B-Arbore

Complexitate:

Procesul se repetă de cel mult **O(h)** ori, în cazul în care valoarea căutată se află într-o frunză.

Căutarea valorii într-un nod se realizează (folosind căutarea binară) în **O(log t)**.

Complexitate finală:

$O(h * \log t) = O(\log t * \log_t n)$

$= O(\log t * (\log n / \log t))$

$= \mathbf{O(\log n)}$

Inserarea în B-Arbore

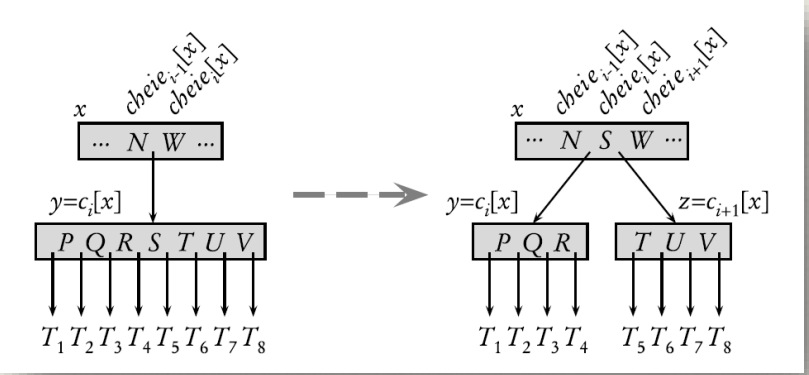
Pentru a insera o cheie **x** într-un B-Arbore, trebuie distinse două cazuri: când nodul unde trebuie introdus are mai puțin de *2t-1* chei, respectiv, când are *2t-1* chei.

Algoritm:

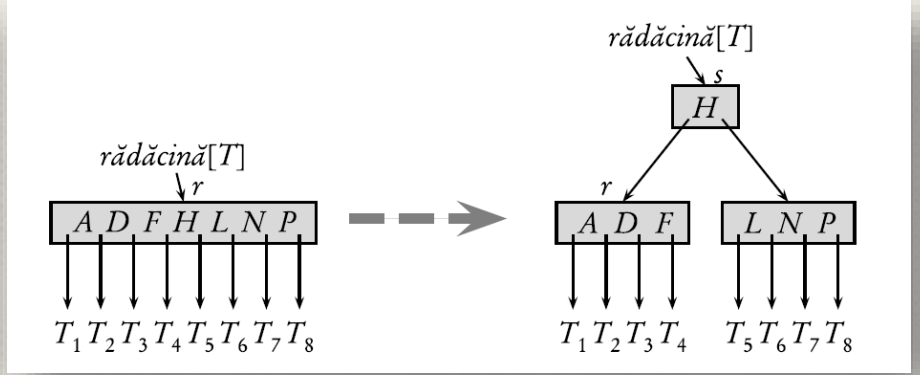
1. Aplicăm operația de căutare pentru a găsi nodul unde trebuie introdusă cheia. Notăm acest nod cu **X** și va fi o **frunză**.
2. Dacă **X** are mai puțin de 2t-1 chei, atunci inserarea se efectuează fără a modifica structura arborelui.
3. Dacă **X** are 2t-1 chei, atunci acesta trebuie divizat. Rezultă, astfel, două noduri noi, $\mathbf{F_s}$ (fiul din stânga) și $\mathbf{F_d}$ (fiul din dreapta).
4. Eliminăm cea mai mare cheie din $\mathbf{F_s}$ (cheia mediană). O notăm cu **M**.
5. **M** devine părintele celor două noduri $\mathbf{F_s}$ și $\mathbf{F_d}$.
6. Se încearcă (recursiv) adăugarea lui **M** în părintele lui **X**.

Inserarea în B-Arbore

Divizarea unui nod (t = 4):



Nod intermediar



Rădăcină

Inserarea în B-Arbore

Complexitate:

Căutarea nodului în care trebuie introdusă cheia: **$O(\log_t n)$**
Pentru un nivel: **$O(t)$**
Recursivitatea: $O(h)$ = **$O(\log_t n)$**

Complexitatea finală: **$O(t \log_t n)$**

B - Arbori

Ștergerea din B-Arbore

La ștergere, numărul de chei dintr-un nod scade cu 1, deci e posibil să ajungem să avem mai puțin de *t* chei. În acest caz, este nevoie de un proces de **fuziune** (invers divizării).

Trebuie să ne asigurăm că toate nodurile rămân cu cel puțin *t* chei.

În procesul de fuziune, o cheie coboară într-un fiu înainte de a i se aplica acestuia (recursiv) procesul de ștergere.

Ștergerea din B-Arbore

Dacă, în urma ștergerii, rădăcina rămâne fără nicio cheie (deci, cu un singur nod fiu), atunci acest nod rădăcină gol este eliminat, iar noua rădăcină devine unicul fiu al rădăcinei vechi. Astfel, înălțimea arborelui scade cu 1.

Ștergerea din B-Arbore

Avem trei cazuri:

1. Cheia de șters **k** este într-un nod **frunză x**. Avem 2 subcazuri:
   a) Dacă, după ștergere, nodul rămâne cu suficiente chei, atunci se șterge cheia **k**, fără nicio altă modificare.

   a) Dacă, după ștergere, nu rămân suficiente chei, atunci:
      - Încercăm să împrumutăm o cheie de la fratele din stânga.
      - Dacă nu putem (rămâne și stânga cu prea puține chei), atunci încercăm împrumutul la dreapta.
      - Dacă nu putem împrumuta nici de la stânga, nici de la dreapta, atunci aplicăm fuziunea pe unul din frați și cheia părinte corespunzătoare.

Ștergerea din B-Arbore

Avem trei cazuri:

2. Cheia de șters **k** este într-un nod **intern x**. Se disting 3 subcazuri:
   a) Dacă fiul **y** din stânga lui **k** are cel puțin *t* chei, atunci se caută predecesorul **k'** al lui **k** în subarborele de rădăcină **y**. Se șterge **k'** și se înlocuiește **k** din **x** cu **k'**. Se aplică mai departe procesul, recursiv.

   a) Analog pentru fiul **z** din dreapta lui **k**, căutându-se succesorul **k'** al lui **k**.

   a) Dacă și **y**, și **z** au *t-1* chei, se aplică fuziunea pe cele două noduri (adăugându-se și cheia **k** în **y**). În urma procesului, nodul **y** va avea *2t-1* chei. Se aplică mai departe procesul de ștergere recursivă a lui **k** din **y**.

Ștergerea din B-Arbore

Avem trei cazuri:

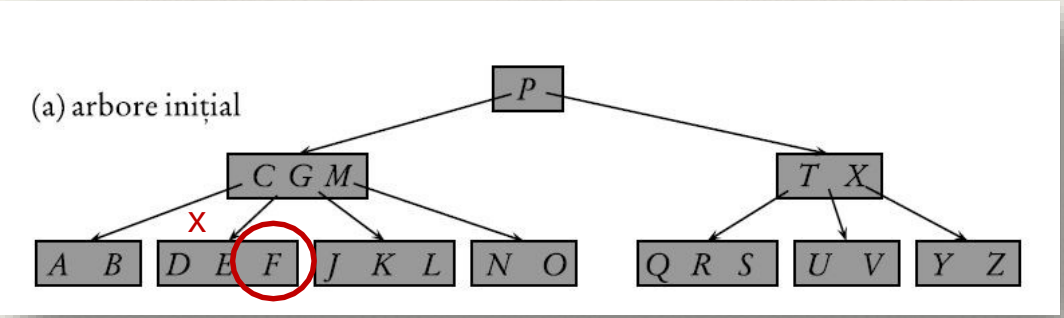
3. Cheia de șters **k** nu se găsește în nodul intern **x**
   - Determinăm rădăcina **r'** (fiu al lui **x**) care indică subarborele în care se găsește **k**.

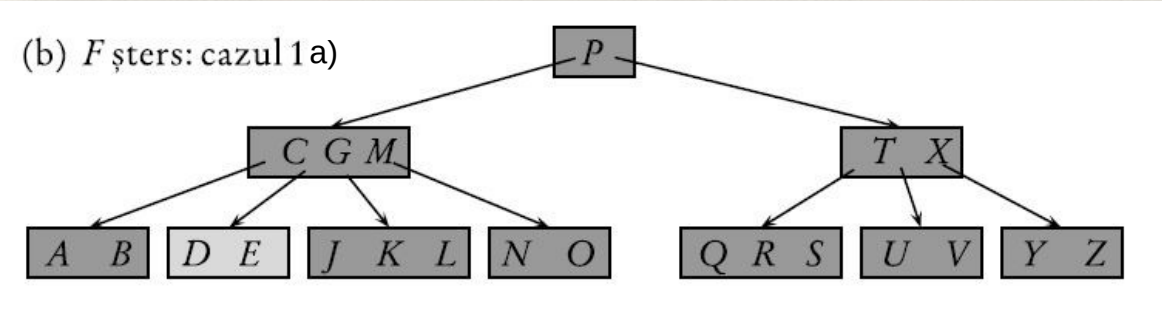
   a) Dacă **r'** are *t-1* chei, dar are un frate în stânga sau în dreapta care are *t* chei, atunci mutăm o cheie din **x** în **r'**, apoi mutăm o cheie din unul din cei 2 frați înapoi în **x**.

   a) Dacă **r'** și cei 2 frați din stânga și din dreapta au câte *t-1* chei, aplicăm procesul de fuziune pe **r'** și unul din cei 2 frați. Se mută, de asemenea, o cheie din **x** în noul nod rezultat, ca și cheie mediană.

Ștergerea din B-Arbore

Exemplu (t = 3):

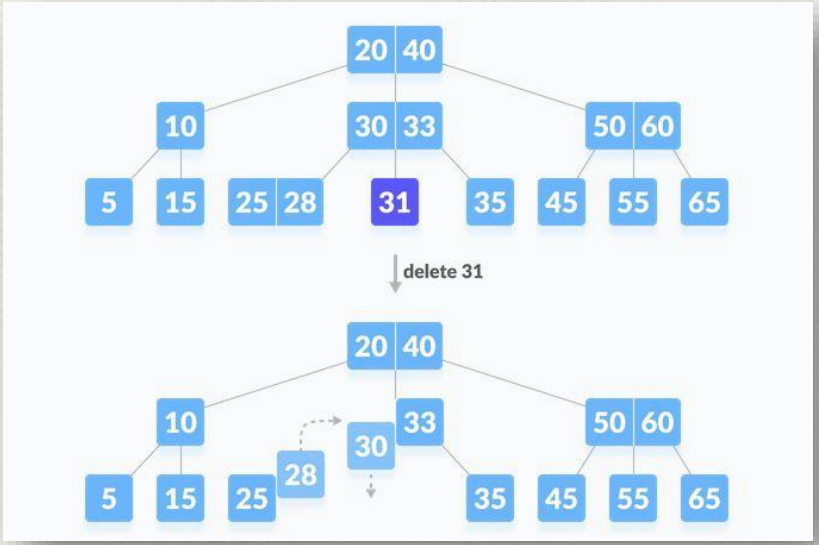


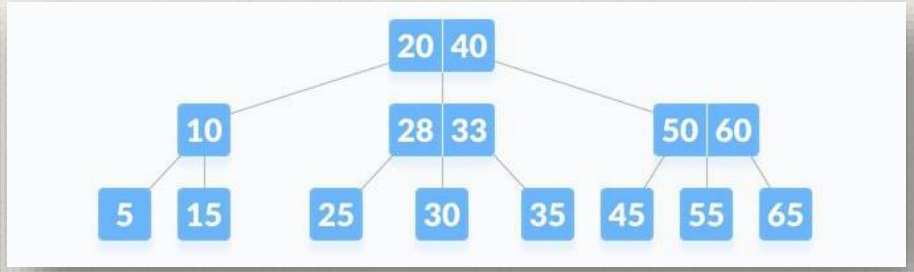


Cheia de șters se află în frunza **x** și putem șterge ușor

Ștergerea din B-Arbore

Exemplu (t = 2):

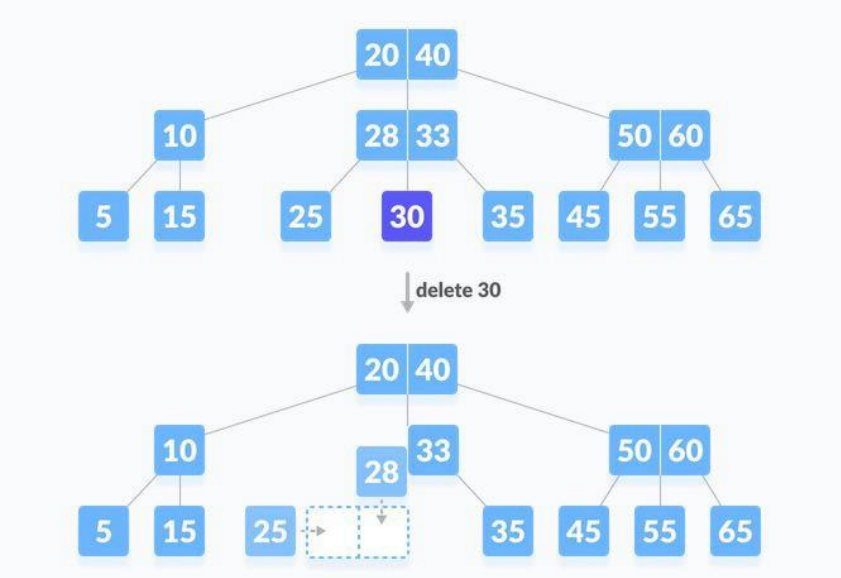


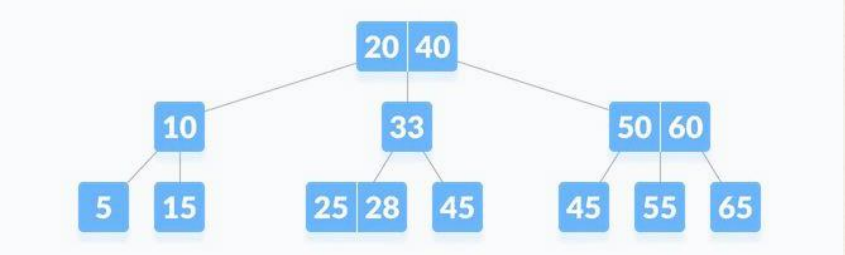


Cheia de șters se află în frunză
și putem împrumuta din stânga

Ștergerea din B-Arbore

Exemplu (t = 2):

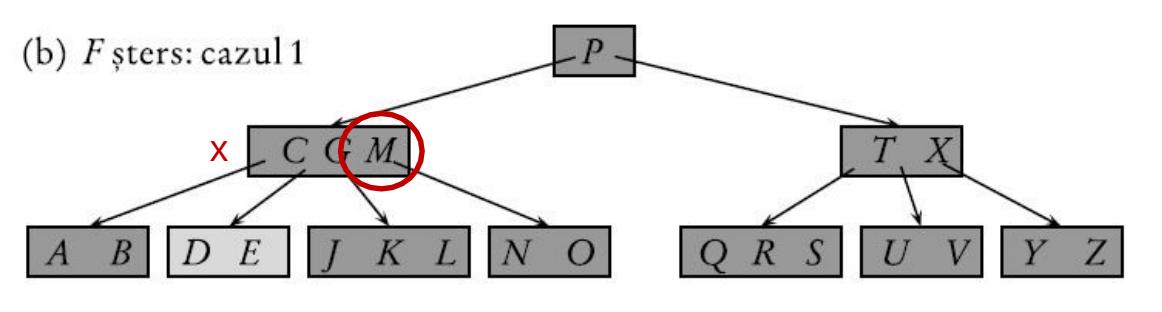


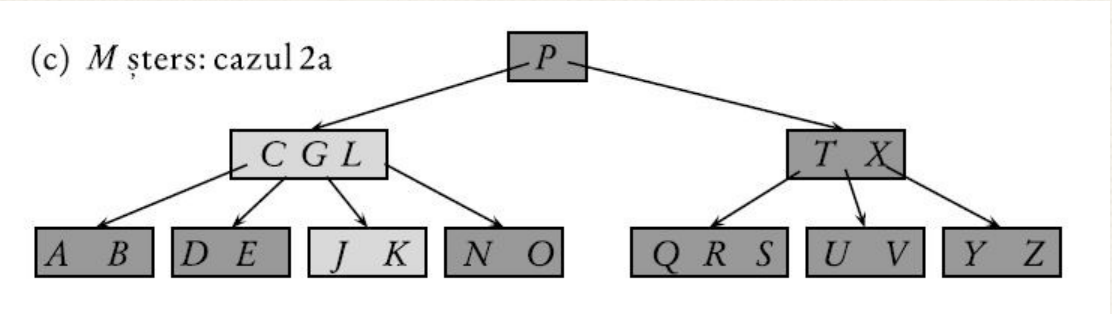


Cheia de șters se află în frunză și **nu** putem împrumuta de nicăieri

Ștergerea din B-Arbore

Exemplu (t = 3):

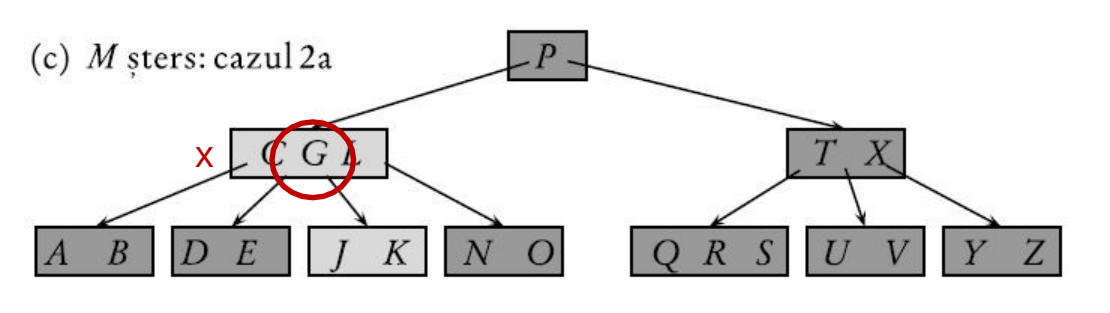


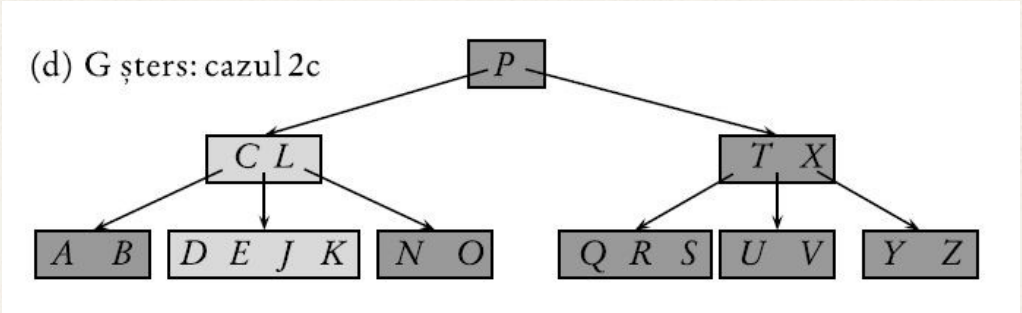


Cheia de șters se află în nodul intern **x** ȘI
fiul stâng **y** are cel puțin *t* chei

Ștergerea din B-Arbore

Exemplu (t = 3):

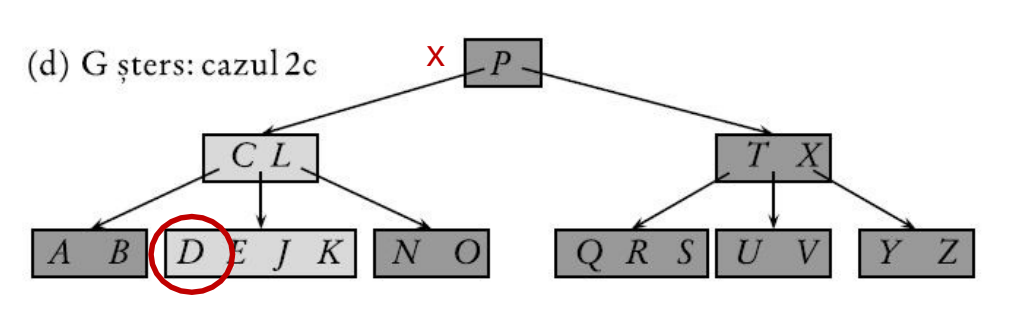


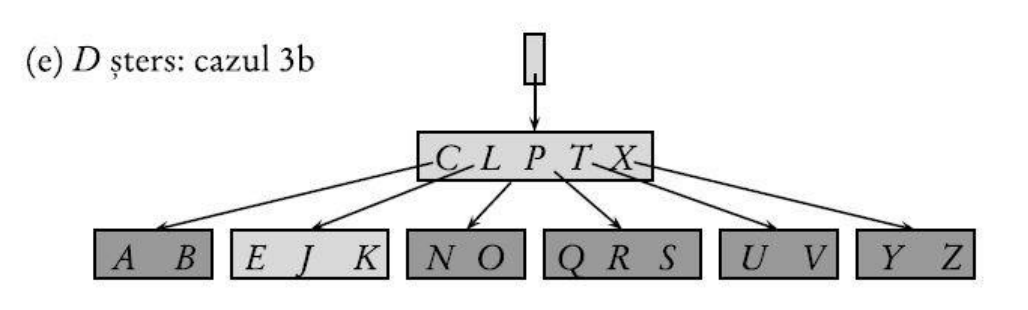


Cheia de șters se află în nodul intern **x**
ȘI
cei 2 fii **y** și **z** au *t-1* chei

Ștergerea din B-Arbore

Exemplu (t = 3):

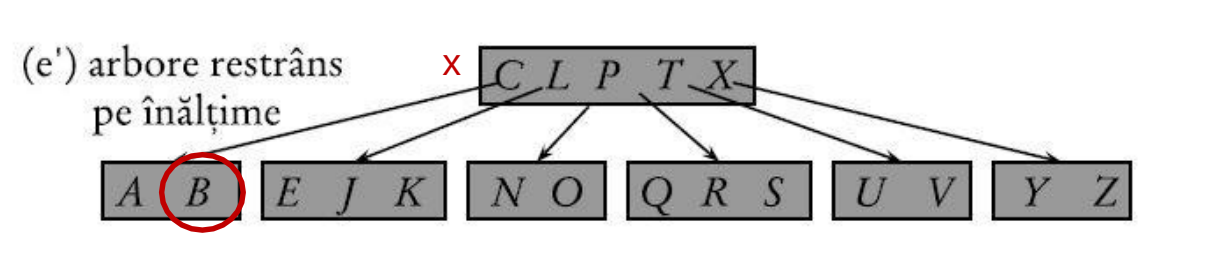


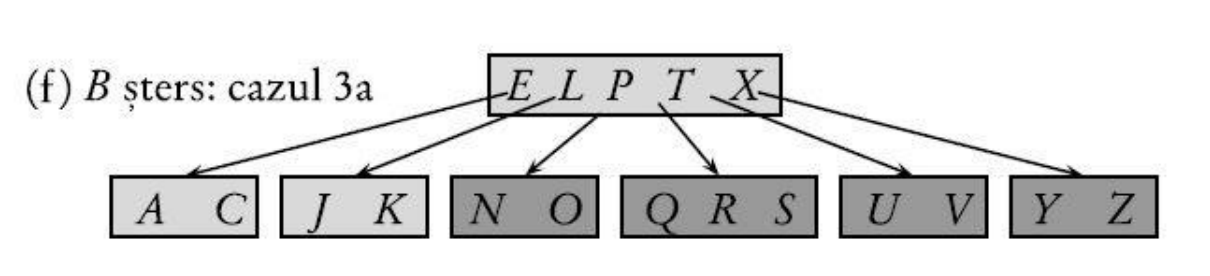


Cheia de șters NU se află în nodul intern **x**
ȘI
r' și frații lui au *t-1* chei

Ștergerea din B-Arbore

Exemplu (t = 3):





Cheia de șters **NU** se află în nodul intern **x** ȘI
r' are *t-1* chei și un frate cu *t* chei

Ștergerea din B-Arbore

Complexitate:

Procedura de ștergere într-un B-Arbore are loc descendent și fără reveniri.
La ștergerea unei chei dintr-un nod intern, are loc o serie de înlocuiri, pentru ca ștergerea efectivă să se realizeze în frunză.

Parcurgerea unui B-Arbore descendent fără reveniri: **O(h)**
Operații pe nivel: **O(t)**

Complexitate finală:

$O(t * h) = \mathbf{O(t \log_t n)}$

Exerciții

I. Căutare
   1. Cum se poate găsi cheia minimă din arbore? Cum putem găsi succesorul / predecesorul unei chei?

II. Inserare
   1. Arătați pașii intermediari și rezultatul inserării cheilor V, D, I, R, B, G, M, C, U, P, S, L, Y, E, T, W, J, Z, O, H, K, în această ordine, într-un B-Arbore inițial vid.
   2. Inserăm cheile {1, 2, ..., n} într-un B-Arbore inițial vid. Gradul arborelui este 2. Câte noduri va avea în final acest arbore?

III. Ștergere
   1. Plecând de la ultima configurație calculată în curs la algoritmul de ștergere, prezentați rezultatele obținute prin eliminarea, în ordine, a cheilor C, P, V.

Bibliografie

Cormen – Introducere în algoritmi, Ediția 3 Ion Ivan – Arbori B
Kerttu Pollari-Malmi – B+-trees
Stanford University - Balanced Trees
https://infolab.usc.edu/csci585/Spring2010/den_ar/indexing.pdf
https://en.wikipedia.org/wiki/B-tree
http://www.cs.cornell.edu/courses/cs312/2008sp/recitations/rec25.html https://www.programiz.com/dsa/deletion-from-a-b-tree

Final