B - Arbori

Universitatea "Transilvania" din Brașov

17 mai 2022

B- Arbori

B-Arborii:

- sunt arbori balansați în care toate frunzele au aceeași adâncime

B- Arbori

B-Arborii:

- sunt arbori balansați în care toate frunzele au aceeași adâncime
- reprezintă o generalizare a arborilor binari de căutare.

B- Arbori

B-Arborii:

- sunt arbori balansați în care toate frunzele au aceeași adâncime
- reprezintă o generalizare a arborilor binari de căutare.
- fiecare nod poate stoca mai multe chei, de la câteva, la mai multe mii.

B- Arbori

B-Arborii:

- sunt arbori balansați în care toate frunzele au aceeași adâncime
- reprezintă o generalizare a arborilor binari de căutare.
- fiecare nod poate stoca mai multe chei, de la câteva, la mai multe mii.
- Alături de chei, un nod al arborelui poate conține și alte informații de interes stocate în structura de date.

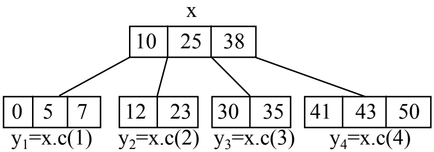
B- Arbori

B-Arborii:

- sunt arbori balansați în care toate frunzele au aceeași adâncime
- reprezintă o generalizare a arborilor binari de căutare.
- fiecare nod poate stoca mai multe chei, de la câteva, la mai multe mii.
- Alături de chei, un nod al arborelui poate conține și alte informații de interes stocate în structura de date.

Observație: O variantă utilizată frecvent a B-arborilor sunt arborii B+, care sunt B-arbori cu proprietatea că informațiile suplimentare diferite de chei sunt stocate doar în frunze. Nodurile interne conțin doar chei.

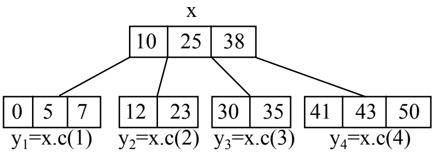
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod *x* are următoarele atribute (câmpuri):

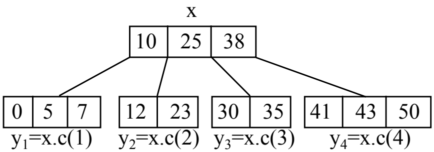
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.

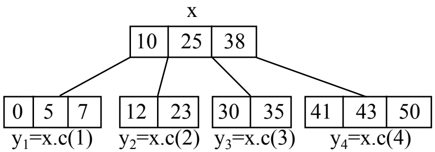
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.
   (b) Vectorul de chei $x.key$, cu cheile sortate crescător.

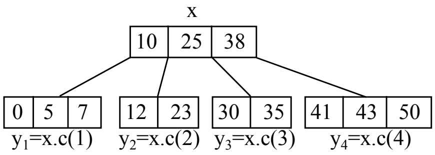
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.
   (b) Vectorul de chei $x.key$, cu cheile sortate crescător.
   (c) $x.leaf$ $=$ un câmp boolean care este TRUE, dacă x este frunză și FALSE altfel.

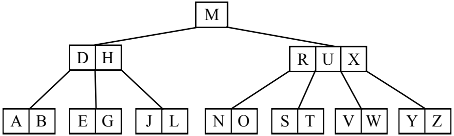
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.
   (b) Vectorul de chei $x.key$, cu cheile sortate crescător.
   (c) $x.leaf$ $=$ un câmp boolean care este TRUE, dacă x este frunză și FALSE altfel.
   (d) Vectorul $x.c$ de legături către fiii lui x

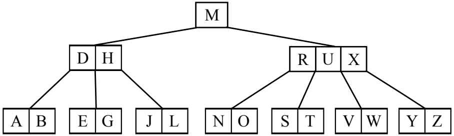
B-Arbori - Proprietăți



Proprietăți

1. Fiecare nod x are următoarele atribute (câmpuri):
   (a) $x.n$ numărul de chei stocate în nodul x.
   (b) Vectorul de chei $x.key$, cu cheile sortate creascător.
   (c) $x.leaf$ $=$ un câmp boolean care este TRUE, dacă x este frunză și FALSE altfel.
   (d) Vectorul $x.c$ de legături către fiii lui x
2. Fiecare nod intern x are $x.n + 1$ fii.

B-Arbori - Proprietăți



3. Există o ordonare a cheilor în arbore. Dacă notăm cu $y_i = x.c(i), i = 1, \ldots, x.n + 1$ atunci:

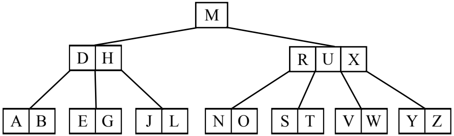
$$y_1.key(1) \leq y_1.key(2) \leq \ldots \leq y_1.key(y_1.n) \leq x.key(1) \leq$$

$$\leq y_2.key(1) \leq y_2.key(2) \leq \ldots \leq y_2.key(y_2.n) \leq x.key(2) \leq$$

$$\ldots$$

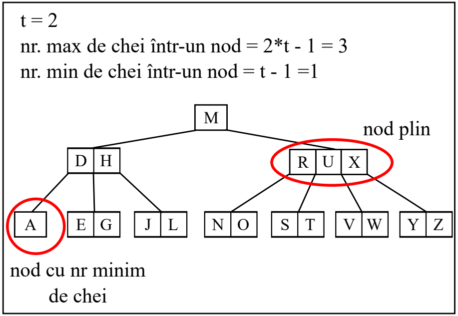
$$\leq y_{x.n+1}.key(1) \leq y_{x.n+1}.key(2) \leq \ldots \leq y_{x.n+1}.key(y_{x.n+1}.n)$$

B-Arbori - Proprietăți



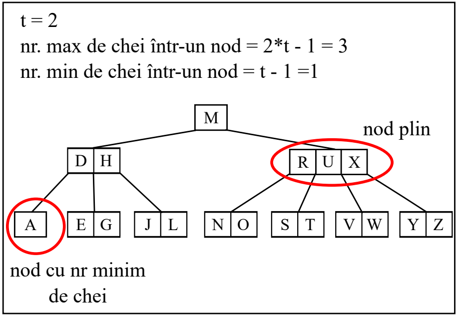
4. Toate frunzele au aceeași adâncime = adâncimea/înălțimea *h* a arborelui.

B-Arbori - Proprietăți



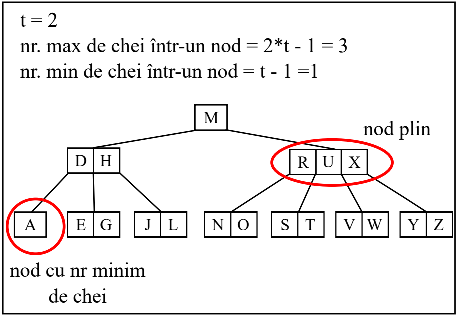
4. Toate frunzele au aceeași adâncime = adâncimea/înălțimea h a arborelui.
5. Numărul de chei ale unui nod este limitat inferior și superior pe baza unei constante t, $t \geq 2$, numită ***gradul minim*** al arborelui:

B-Arbori - Proprietăți



4. Toate frunzele au aceeași adâncime = adâncimea/înălțimea h a arborelui.
5. Numărul de chei ale unui nod este limitat inferior și superior pe baza unei constante t, $t \geq 2$, numită ***gradul minim*** al arborelui:
   - Fiecare nod x cu excepția rădăcinii, conține cel puțin $t - 1$. Rădăcina conține cel puțin o cheie.

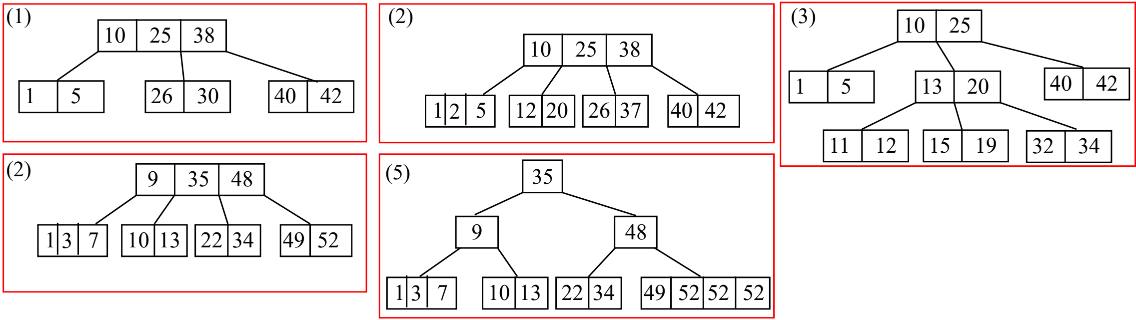
B-Arbori - Proprietăți



4. Toate frunzele au aceeași adâncime = adâncimea/înălțimea h a arborelui.
5. Numărul de chei ale unui nod este limitat inferior și superior pe baza unei constante t, $t \geq 2$, numită ***gradul minim*** al arborelui:
   - Fiecare nod x cu excepția rădăcinii, conține cel puțin $t - 1$. Rădăcina conține cel puțin o cheie.
   - Fiecare nod x conține cel mult $2t - 1$ chei. Un nod care conține $2t - 1$ chei se numește ***nod plin***.

B-Arbori - Exercițiu

Care dintre următorii arbori sunt un B-arbori corecți, considerând gradul minim $t = 2$?



B-Arbori - Înalțime

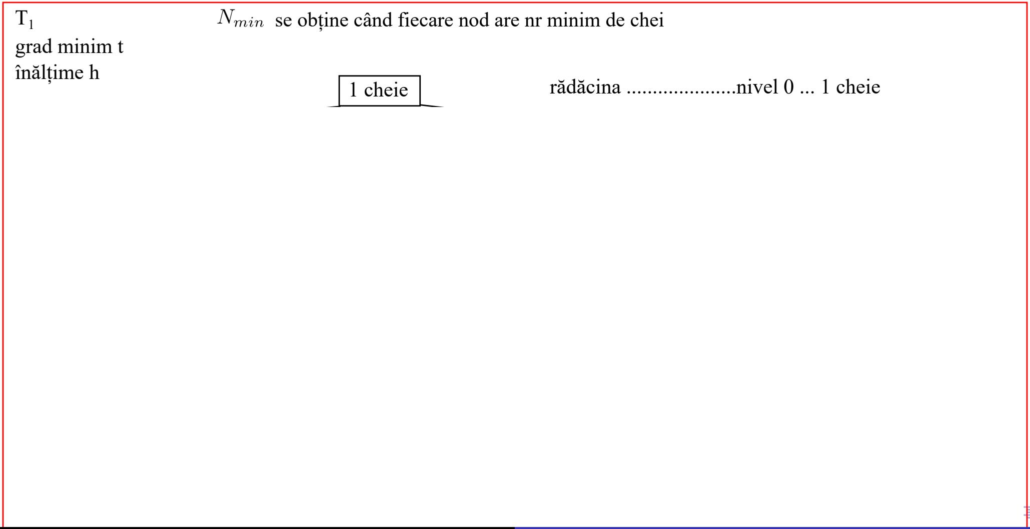
Teoremă

Într-un B-arbore de grad minim $t \geq 2$ cu n chei înălțimea h a arborelui respectă inegalitatea:

$$h \leq \log_t \left(\frac{n+1}{2} \right)$$

B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.



B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei

1 cheie

t-1 chei   t-1 chei

rădăcina ....nivel 0 ... 1 cheie

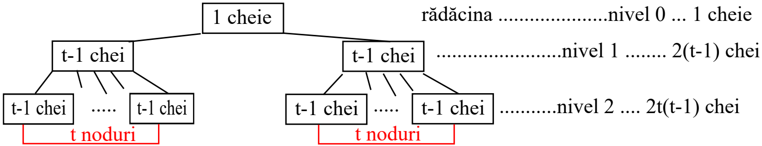
........................nivel 1 .... 2(t-1) chei

B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei

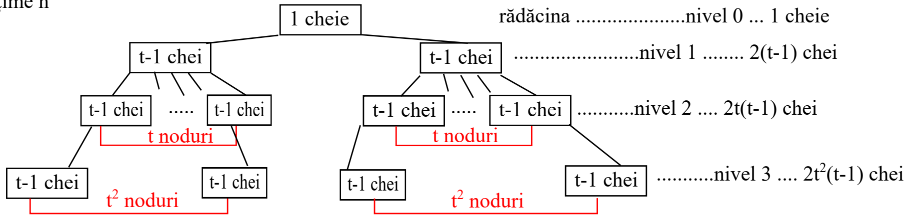


B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

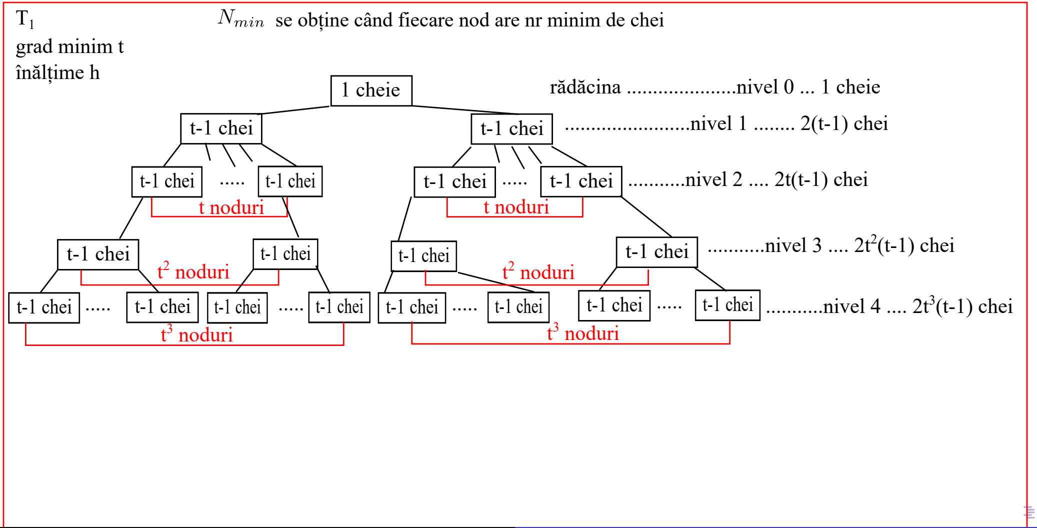
T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei



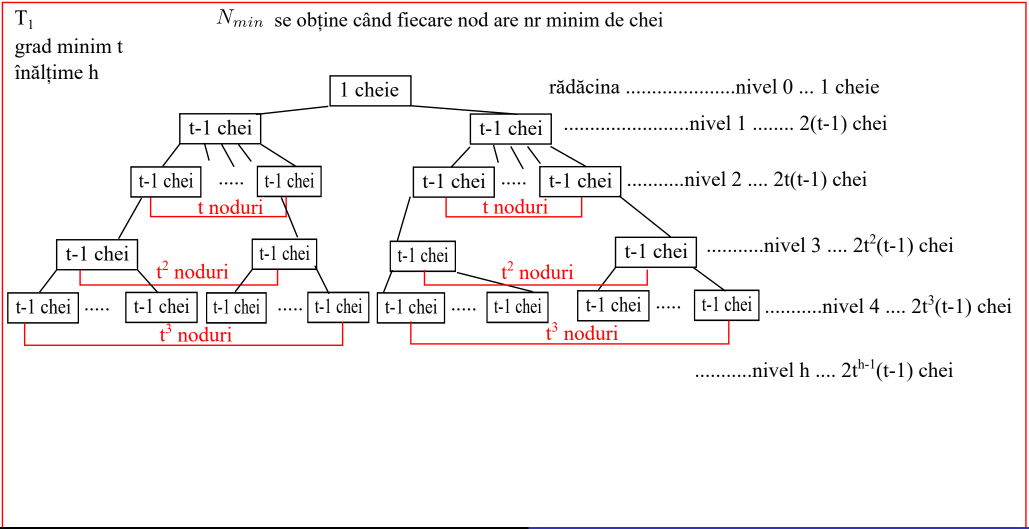
B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.



B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

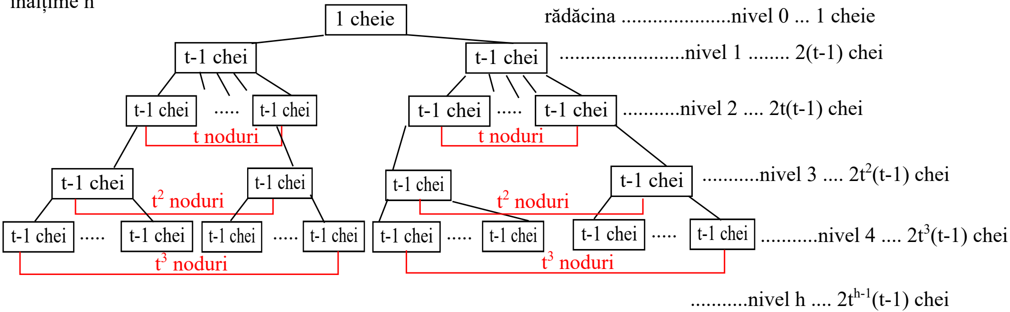


B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei



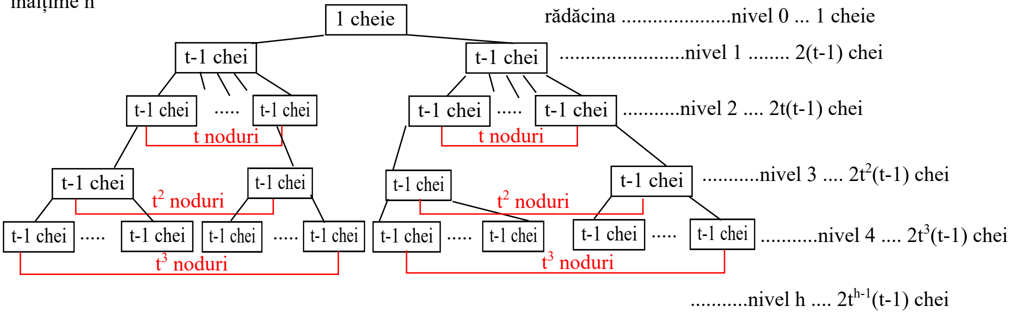
Total: $N_{min} = 1 + 2(t-1) + 2t(t-1) + \ldots + 2t^{h-1}(t-1)$

B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei



Total: $N_{min} = 1 + 2(t-1) + 2t(t-1) + \ldots + 2t^{h-1}(t-1)$

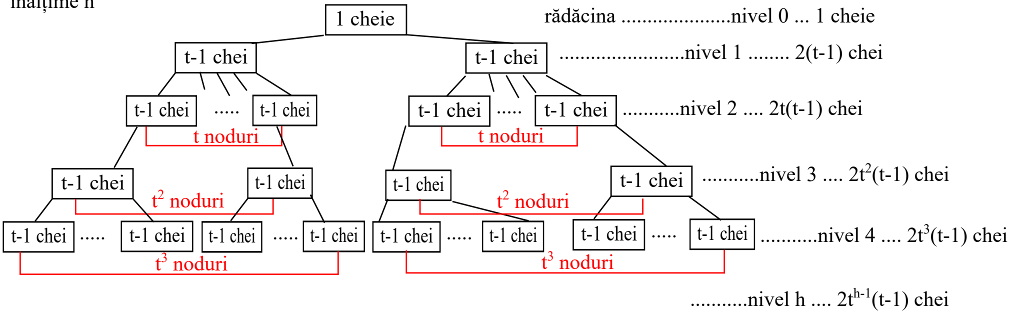
$= 1 + 2(t-1)\left(1 + t + t^2 + \ldots t^{h-1}\right)$

B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei



Total: $N_{min} = 1 + 2(t-1) + 2t(t-1) + \ldots + 2t^{h-1}(t-1)$

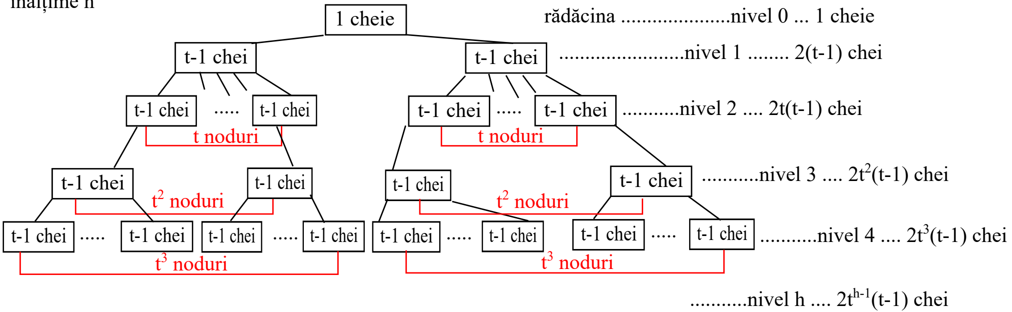
$$= 1 + 2(t-1)\left(1 + t + t^2 + \ldots t^{h-1}\right) = 1 + 2(t-1)\frac{(t^h - 1)}{(t-1)} = 2t^h - 1$$

B-Arbori - Înălțime

Demonstrație: Fie T un B-arbore cu n chei și înălțimea h și T_1 un B-arbore cu aceeași înălțime, dar cu număr minim de chei $N_{min} \Rightarrow N_{min} \leq n$.

T_1
grad minim t
înălțime h

N_{min} se obține când fiecare nod are nr minim de chei



Total: $N_{min} = 1 + 2(t-1) + 2t(t-1) + \ldots + 2t^{h-1}(t-1)$

$$= 1 + 2(t-1)\left(1 + t + t^2 + \ldots t^{h-1}\right) = 1 + 2(t-1)\frac{(t^h-1)}{(t-1)} = 2t^h - 1$$

Deci: $n \geq N_{min} = 2t^h - 1 \quad \Rightarrow h \leq \log_t \left(\frac{n+1}{2}\right)$

B-Arbori

Observații:

- Înălțimea unui B-arbore este cu atât mai mică, cu cât gradul minim t este mai mare.

B-Arbori

Observații:

- Înălțimea unui B-arbore este cu atât mai mică, cu cât gradul minim t este mai mare.
- Complexitatea operațiilor de căutare, inserție și ștergere depinde de înălțimea arborelui, deci de t

B-Arbori

Observații:

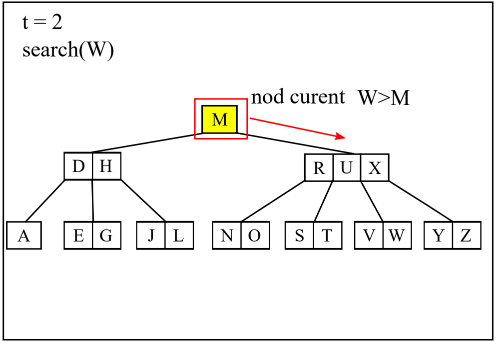
- Înălțimea unui B-arbore este cu atât mai mică, cu cât gradul minim t este mai mare.
- Complexitatea operațiilor de căutare, inserție și ștergere depinde de înălțimea arborelui, deci de t
- **Aplicații:** - implementarea bazelor de date pentru acces rapid la datele de pe disc

B-Arbori - Căutare

Căutarea într-un B-arbore:

- generalizare a căutării binare
- pentru fiecare nod nu avem doar o decizie între $n + 1$ ramuri, n fiind numărul de chei ale nodului curent.

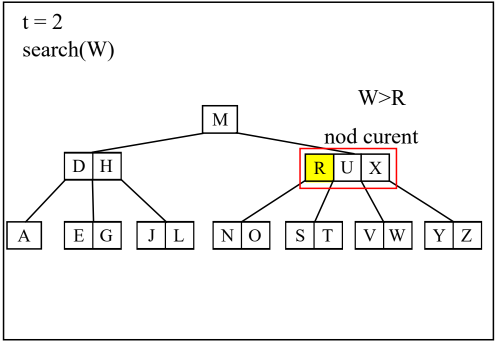
B-Arbori - Căutare



Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ si $nod.key(i) < key$
  executa
    $i \leftarrow i + 1$
daca $i \leq nod.n$ si $nod.key(i) = key$
  atunci
    RETURN(nod, i)
altfel
    daca $nod.leaf = true$ **atunci**
        RETURN NIL
    altfel
        DiskRead($nod.c(i)$)
        RETURN B_Search($nod.c(i), key$)

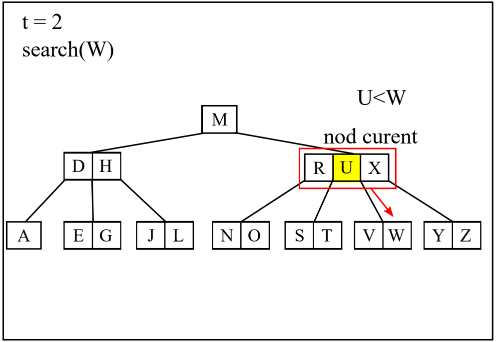
B-Arbori - Căutare



Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ si $nod.key(i) < key$
  executa
    $i \leftarrow i + 1$
daca $i \leq nod.n$ si $nod.key(i) = key$
  atunci
    RETURN(nod, i)
altfel
  daca $nod.leaf = true$ **atunci**
    RETURN NIL
  altfel
    DiskRead($nod.c(i)$)
    RETURN B_Search($nod.c(i), key$)

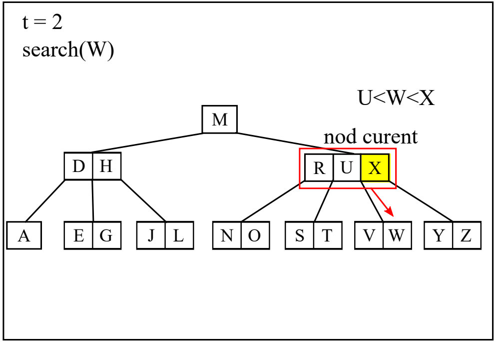
B-Arbori - Căutare



Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ *si* $nod.key(i) < key$ **executa**
  $i \leftarrow i + 1$
daca $i \leq nod.n$ *si* $nod.key(i) = key$ **atunci**
  RETURN(nod, i)
altfel
  daca $nod.leaf = true$ **atunci**
    RETURN NIL
  altfel
    DiskRead($nod.c(i)$)
    RETURN B_Search($nod.c(i)$, key)

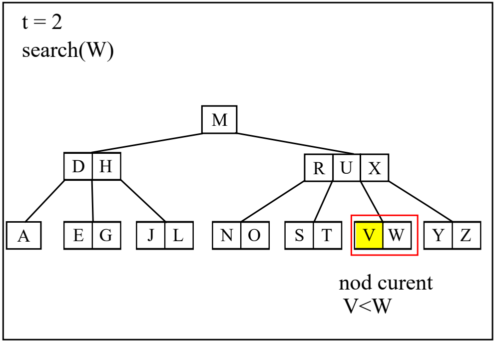
B-Arbori - Căutare



Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ si $nod.key(i) < key$ **executa**
  $i \leftarrow i + 1$
daca $i \leq nod.n$ si $nod.key(i) = key$ **atunci**
  RETURN(nod, i)
altfel
  daca $nod.leaf = true$ **atunci**
    RETURN NIL
  altfel
    DiskRead($nod.c(i)$)
    RETURN B_Search($nod.c(i), key$)

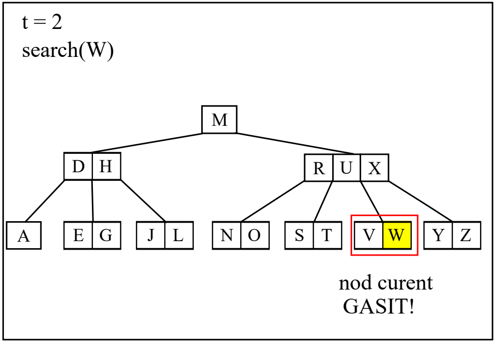
B-Arbori - Căutare



Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ *si* $nod.key(i) < key$ **executa**
  $i \leftarrow i + 1$
daca $i \leq nod.n$ *si* $nod.key(i) = key$ **atunci**
  RETURN(nod, i)
altfel
  daca $nod.leaf = true$ **atunci**
    RETURN NIL
  altfel
    DiskRead($nod.c(i)$)
    RETURN B_Search($nod.c(i)$, key)

B-Arbori - Căutare

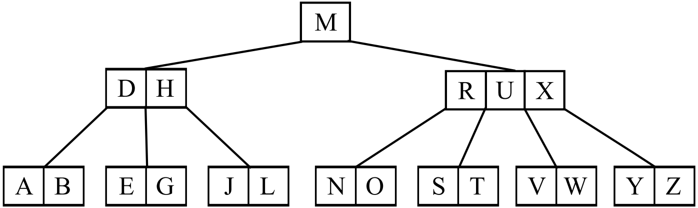


Algoritm 1: B_Search(nod, key)

$i \leftarrow 1$
cat_timp $i \leq nod.n$ *si* $nod.key(i) < key$ **executa**
  $i \leftarrow i + 1$
daca $i \leq nod.n$ *si* $nod.key(i) = key$ **atunci**
  RETURN(nod, i)
altfel
  daca $nod.leaf = true$ **atunci**
    RETURN NIL
  altfel
    DiskRead($nod.c(i)$)
    RETURN B_Search($nod.c(i)$, key)

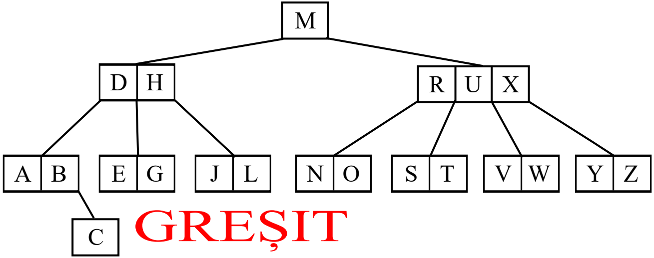
B-Arbori - Inserarea unei chei

Cum se inserează cheia C în arborele din figură? (t=2)



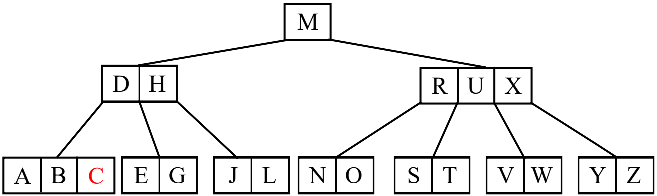
B-Arbori - Inserarea unei chei

Cum se inserează cheia *C* în arborele din figură? (t=2)
NU putem pur și simplu să creăm o frunză nouă!!!



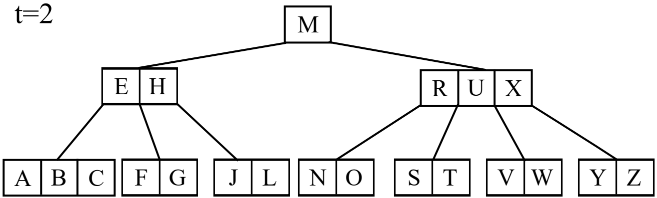
B-Arbori - Inserarea unei chei

Cum se inserează cheia *C* în arborele din figură? (t=2)
NU putem pur și simplu să creăm o frunză nouă!!!
Inserția se face mereu într-o frunză existentă!



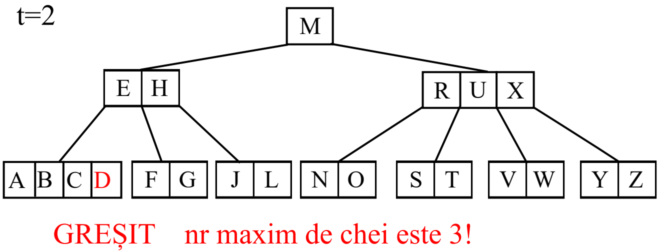
B-Arbori - Inserarea unei chei

Cum se inserează cheia D în arborele din figură?



B-Arbori - Inserarea unei chei

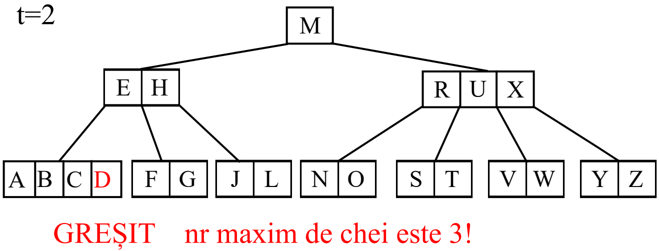
Cum se inserează cheia *D* în arborele din figură?
Nu pot insera *D* în frunza (A,B,C), pentru că este plină!! Ce fac?



B-Arbori - Inserarea unei chei

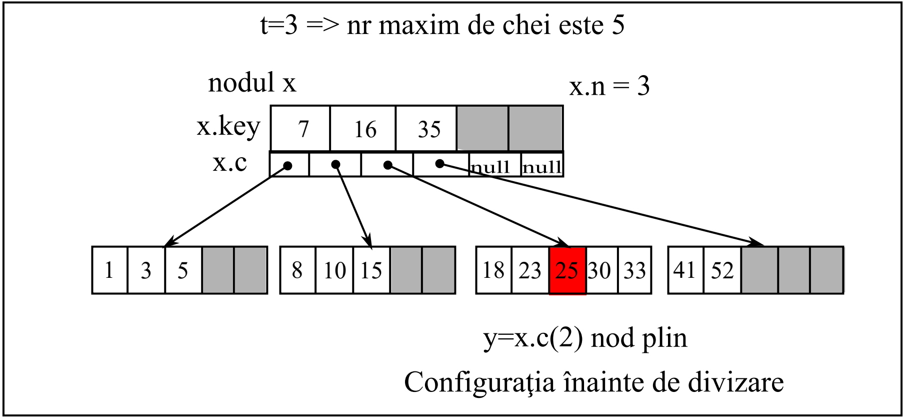
Cum se inserează cheia *D* în arborele din figură?
Nu pot insera *D* în frunza (A,B,C), pentru că este plină!! Ce fac?

Se aplică operația de DIVIZARE a frunzei!



B-Arbori - Divizarea unei chei

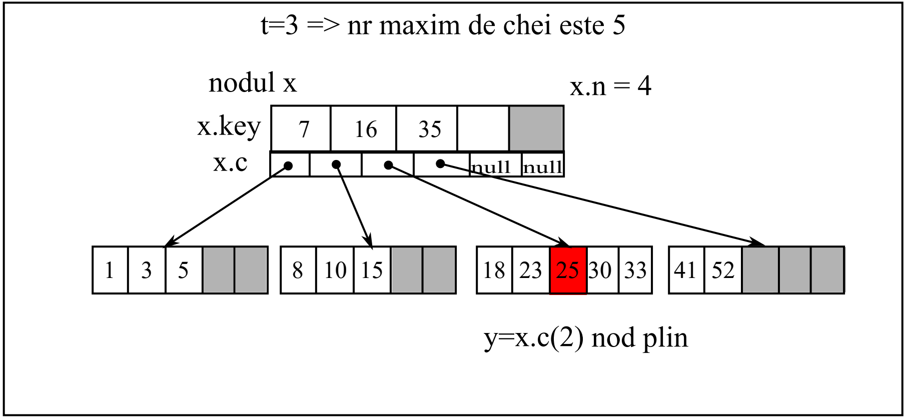
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



.

B-Arbori - Divizarea unei chei

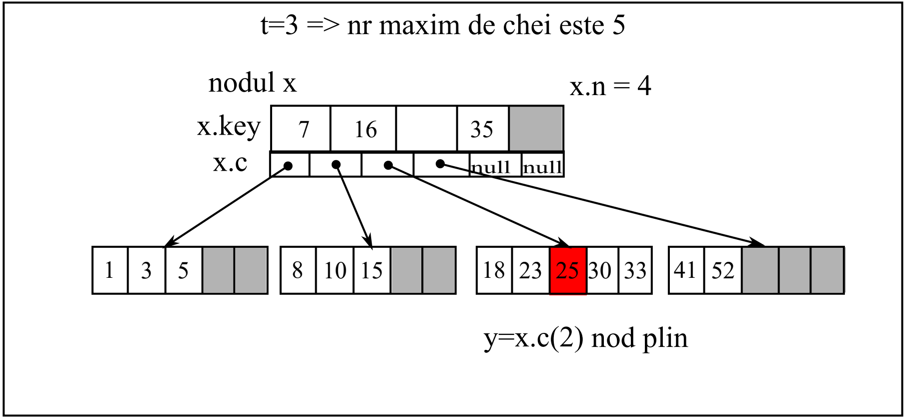
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



1. Crește numărul de chei al lui x: $x.n \leftarrow x.n + 1$.   .

B-Arbori - Divizarea unei chei

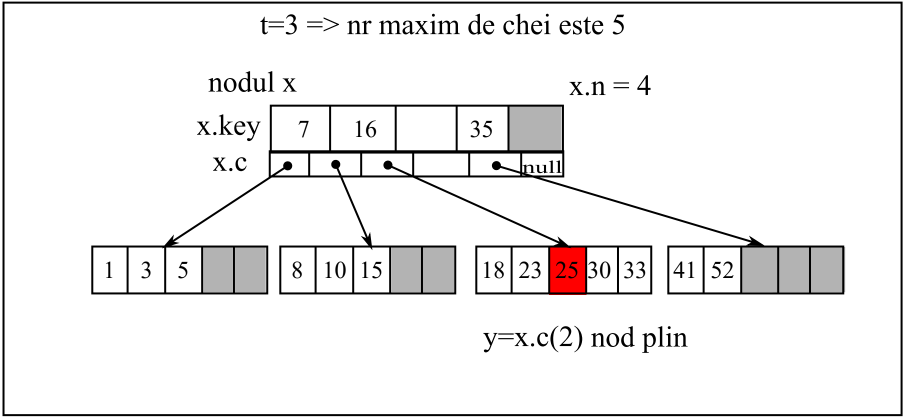
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



2. Se deplasează cheile lui x începând de la poziția i cu o poziție la dreapta:
pentru $k = \overline{x.n, i+1}$ $x.key(k) \leftarrow x.key(k-1)$.

B-Arbori - Divizarea unei chei

Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$

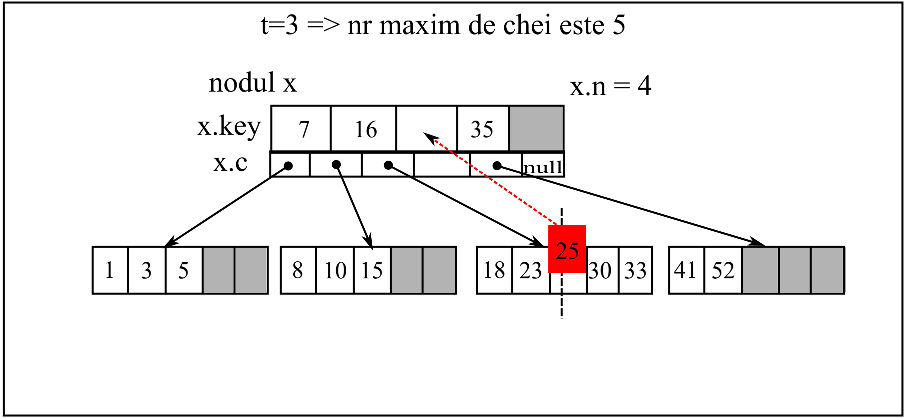


. 3. Se deplasează în x pointerii către descendenții $x.c(i+1), \ldots, x.c(x.n)$ cu o poziție spre dreapta:

pentru $k = \overline{x.n+1, i+2}$ $x.c(k) \leftarrow x.key(k-1)$

B-Arbori - Divizarea unei chei

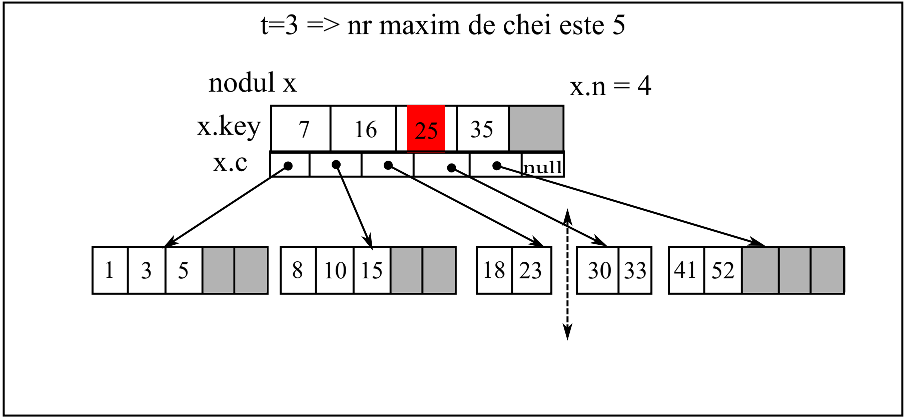
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



. 4. cheia $y.key(t)$ urcă în nodul x pe poziția i

B-Arbori - Divizarea unei chei

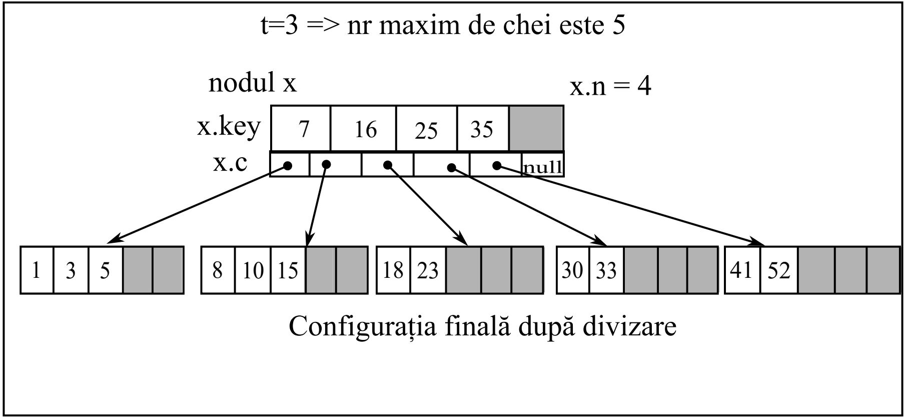
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



. 5. nodul y se divizează în două noduri noi care conțin câte $t - 1$ chei. Noile noduri create devin descendenții $x.c(i)$ și $x.c(i + 1)$.

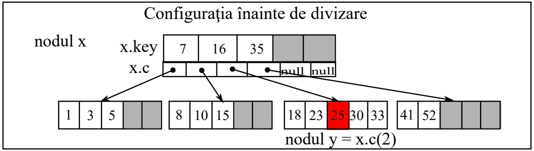
B-Arbori - Divizarea unei chei

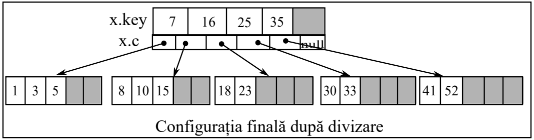
Divizarea nodului $y = x.c(i)$ în jurul cheii mediane $y.key(t)$. În exemplu $t = 3, i = 2$



.

B-Arbori - Divizarea unei chei





Algoritm 2: Split(x, i)

$y \leftarrow x.c(i)$
$x.n \leftarrow x.n + 1$
pentru $k = \overline{x.n, i+1}$ **executa**
  $x.key[k] \leftarrow x.key[k-1]$
$x.key[i] \leftarrow y.key(t)$
pentru $k = \overline{x.n, i+1}$ **executa**
  $x.c[k+1] \leftarrow x.c[k]$
aloca nod nou z
$z.n = t - 1$
pentru $k = \overline{1, t-1}$ **executa**
  $z.key(k) \leftarrow y.key(k+t)$
daca $y.leaf = false$ **atunci**
  pentru $k = \overline{1, t}$ **executa**
    $z.c(k) \leftarrow y.c(k+t)$
$z.leaf \leftarrow y.leaf$
$x.c(i+1) \leftarrow z$

B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.

B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.
2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:
   Divizarea rădăcinii

B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.

2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:
   Divizarea rădăcinii

   - Se creează un nou nod vid, care va fi noua rădăcină.

B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.
2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:
   Divizarea rădăcinii
   - Se creează un nou nod vid, care va fi noua rădăcină.
   - Se leagă vechea rădăcină ca descendent al noii rădăcini.

B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.
2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:
   Divizarea rădăcinii
   - Se creează un nou nod vid, care va fi noua rădăcină.
   - Se leagă vechea rădăcină ca descendent al noii rădăcini.
   - Se realizează operația de divizare.

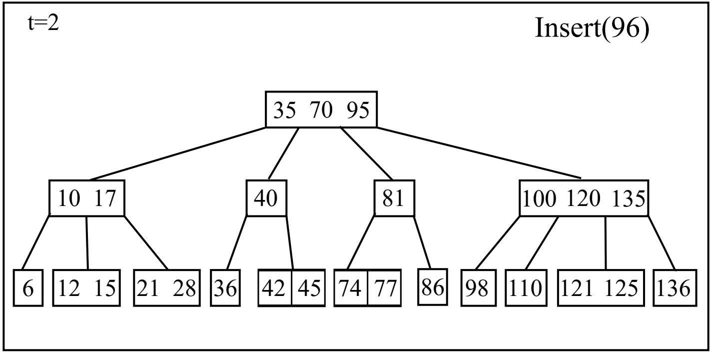
B-Arbori - Inserarea unei chei

Observații:

1. Dacă x este la rândul său plin, nu putem efectua divizarea $\rightarrow$ trebuie să ne asigurăm deja pe parcursul căutării frunzei în care se realizează inserția că pe drum nu întâlnim noduri pline. În caz contrar se realizează o divizare.
2. Dacă nodul plin y este rădăcina, atunci nu există un părinte x, în care să se insereze cheia mediană din y. Astfel, dacă nodul care trebuie divizat este chiar rădăcina trebuie procedat în modul următor:
   Divizarea rădăcinii
   - Se creează un nou nod vid, care va fi noua rădăcină.
   - Se leagă vechea rădăcină ca descendent al noii rădăcini.
   - Se realizează operația de divizare.
3. Înălțimea unui B-arbore crește doar prin divizarea rădăcinii.

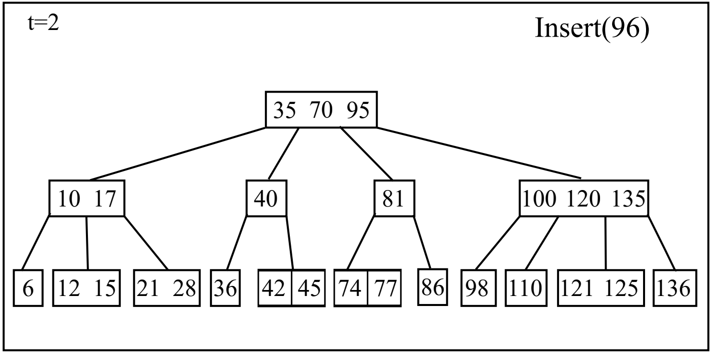
B-Arbori - Inserție

Cum inserăm cheia 96 în arborele din figură?



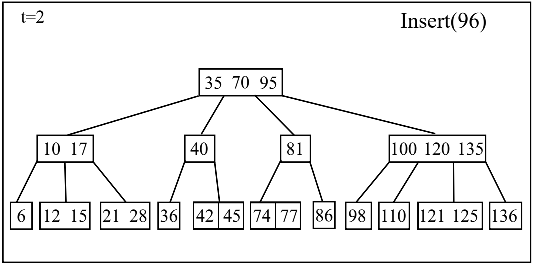
B-Arbori - Inserție

Cum inserăm cheia 96 în arborele din figură?



NU putem insera direct în frunză! Trebuie divizate pe drum nodurile pline!
Rădăcina este plină $\Rightarrow$ trebuie divizată.

B-Arbori - Inserție



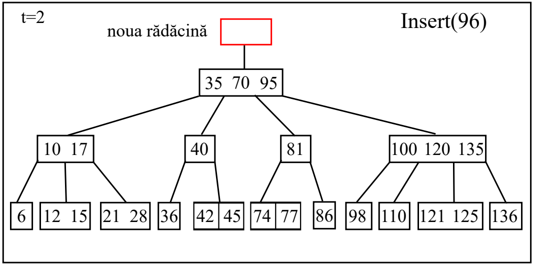
Algoritm 3: BTree_Insert(T, key)

---

daca $T.rad.nr = 2t - 1$ **atunci**
- aloca mem pentru nod_nou
- $nod\_nou.leaf = false$
- $nod\_nou.n = 0$
- $nod\_nou.c(1) = T.rad$
- $T.rad \leftarrow nod\_nou$
- Split($T.rad$, 1)

B_Insert_Rec($T.rad$, key)

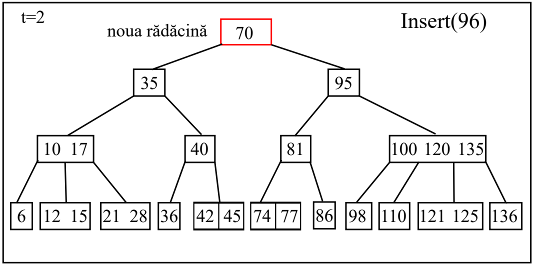
B-Arbori - Inserție



Algoritm 3: BTree_Insert(T, key)

daca $T.rad.nr = 2t - 1$ **atunci**
  aloca mem pentru nod_nou
  $nod\_nou.leaf = false$
  $nod\_nou.n = 0$
  $nod\_nou.c(1) = T.rad$
  $T.rad \leftarrow nod\_nou$
  Split($T.rad, 1$)
B_Insert_Rec($T.rad, key$)

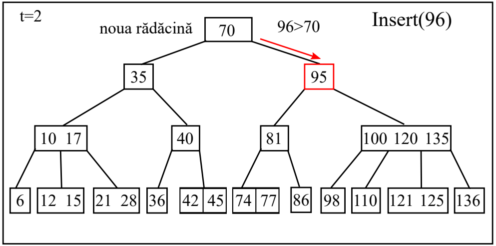
B-Arbori - Inserție



Algoritm 3: BTree_Insert(T, key)

daca $T.rad.nr = 2t - 1$ **atunci**
  aloca mem pentru nod_nou
  $nod\_nou.leaf = false$
  $nod\_nou.n = 0$
  $nod\_nou.c(1) = T.rad$
  $T.rad \leftarrow nod\_nou$
  Split($T.rad$, 1)
B_Insert_Rec($T.rad$, key)

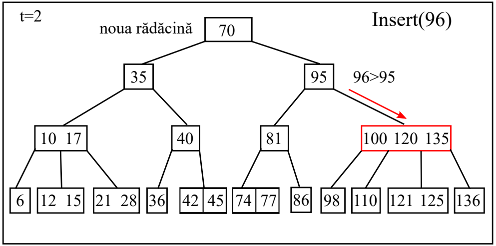
B-Arbori - Inserție



Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$
daca $nod.leaf = true$ **atunci**
  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**
    $nod.key(i+1) \leftarrow nod.key(i)$
     //translatez cheile
    $i \leftarrow i - 1$
  $nod.key(i) \leftarrow key$
  $nod.n \leftarrow nod.n + 1$
altfel
  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**
    $i \leftarrow i - 1$
  $i \leftarrow i + 1$
  daca $nod.c(i).n = 2t - 1$ **atunci**
    Split(nod, i)
    daca $key > x.key(i)$ **atunci**
      $i \leftarrow i + 1$
  B_Insert_Rec($nod.c(i), key$)

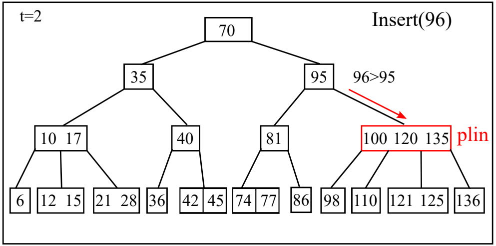
B-Arbori - Inserție



Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$
daca $nod.leaf = true$ **atunci**
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $nod.key(i+1) \leftarrow nod.key(i)$
     //translatez cheile
    $i \leftarrow i - 1$
  $nod.key(i) \leftarrow key$
  $nod.n \leftarrow nod.n + 1$
altfel
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $i \leftarrow i - 1$
  $i \leftarrow i + 1$
  daca $nod.c(i).n = 2t - 1$ **atunci**
    Split(nod, i)
    daca $key > x.key(i)$ **atunci**
      $i \leftarrow i + 1$
  B_Insert_Rec($nod.c(i), key$)

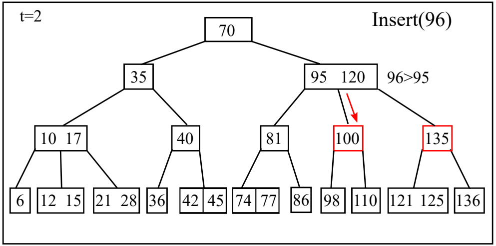
B-Arbori - Inserție



Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$
daca $nod.leaf = true$ **atunci**
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $nod.key(i+1) \leftarrow nod.key(i)$
     //translatez cheile
    $i \leftarrow i - 1$
  $nod.key(i) \leftarrow key$
  $nod.n \leftarrow nod.n + 1$
altfel
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $i \leftarrow i - 1$
  $i \leftarrow i + 1$
  daca $nod.c(i).n = 2t - 1$ **atunci**
    Split(nod, i)
    daca $key > x.key(i)$ **atunci**
      $i \leftarrow i + 1$
  B_Insert_Rec($nod.c(i), key$)

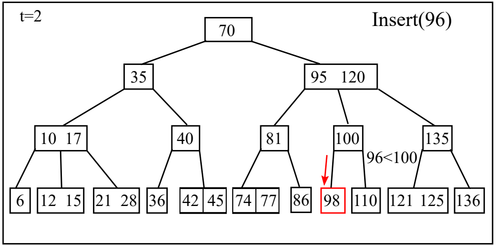
B-Arbori - Inserție



Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$
daca $nod.leaf = true$ **atunci**
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $nod.key(i+1) \leftarrow nod.key(i)$
     //translatez cheile
    $i \leftarrow i - 1$
  $nod.key(i) \leftarrow key$
  $nod.n \leftarrow nod.n + 1$
altfel
  cat_timp $i \geq 1$ *si* $key < nod.key(i)$ **executa**
    $i \leftarrow i - 1$
  $i \leftarrow i + 1$
  daca $nod.c(i).n = 2t - 1$ **atunci**
    Split(nod, i)
    daca $key > x.key(i)$ **atunci**
      $i \leftarrow i + 1$
  B_Insert_Rec($nod.c(i), key$)

B-Arbori - Inserție



Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$

daca $nod.leaf = true$ **atunci**

  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**

    $nod.key(i+1) \leftarrow nod.key(i)$

     //translatez cheile

    $i \leftarrow i - 1$

  $nod.key(i) \leftarrow key$

  $nod.n \leftarrow nod.n + 1$

altfel

  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**

    $i \leftarrow i - 1$

  $i \leftarrow i + 1$

  daca $nod.c(i).n = 2t - 1$ **atunci**

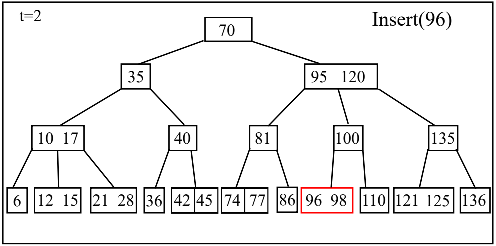
    Split(nod, i)

    daca $key > x.key(i)$ **atunci**

      $i \leftarrow i + 1$

  B_Insert_Rec($nod.c(i), key$)

B-Arbori - Inserție

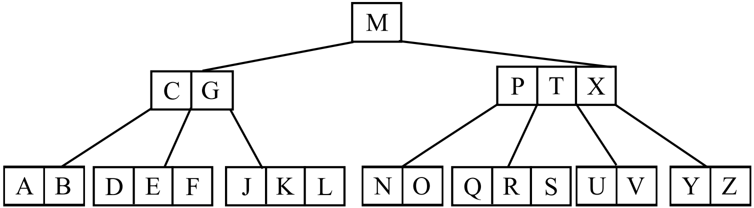


Algoritm 4: B_Insert_Rec(nod, key)

$i \leftarrow x.n$
daca $nod.leaf = true$ **atunci**
  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**
    $nod.key(i+1) \leftarrow nod.key(i)$
     //translatez cheile
    $i \leftarrow i - 1$
  $nod.key(i) \leftarrow key$
  $nod.n \leftarrow nod.n + 1$
altfel
  cat_timp $i \geq 1$ si $key < nod.key(i)$ **executa**
    $i \leftarrow i - 1$
  $i \leftarrow i + 1$
  daca $nod.c(i).n = 2t - 1$ **atunci**
    Split(nod, i)
    daca $key > x.key(i)$ **atunci**
      $i \leftarrow i + 1$
  B_Insert_Rec($nod.c(i), key$)

B-Arbori - Ștergerea unei chei

Problemă: Considerăm B-arborele din figură cu gradul minim $t = 3$. Cum ștergem cheia K? Dar cheia P? Dar cheia M?



B-Arbori - Ștergerea unei chei

Ștergerea unei chei este mai delicată decât inserția deoarece:

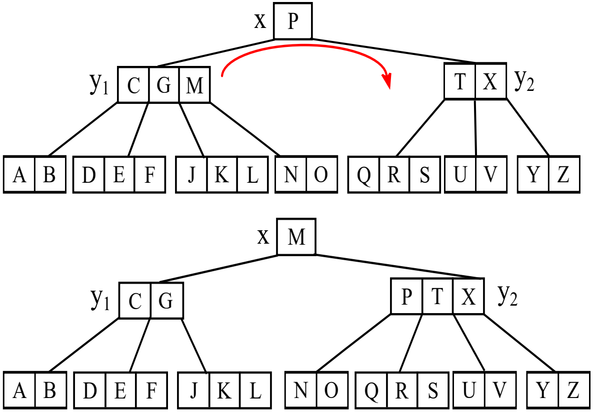
- poate fi necesară ștergerea dintr-un nod care nu e frunză $\Rightarrow$ se reduce numărul de chei $\Rightarrow$ se încalcă regula cu numărul de fii $=$ numărul de chei $+1$
- se poate ajunge la situația în care numărul de chei scade sub $t-1$ - nu e permis!s

B-Arbori - Ștergerea unei chei

Pentru ștergerea cheii *key*

- pe drumul de la rădăcină spre nodul x, care conține *key* ne asigurăm că, orice nod pe care urmează să coborâm are cel puțin t chei
- pentru a asigura numărul de t chei într-un nod se folosesc două operații speciale:
  - Rotația
  - Fuziunea

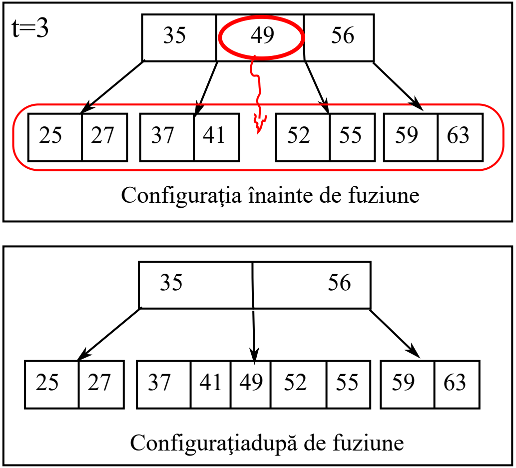
B-Arbori - Rotație



Se consideră nodul x.

- O rotație la dreapta în jurul cheii $k = x.key(i)$:
  - mută ultima cheie a fiului $y_i = x.c(i)$ în nodul x în locul cheii k
  - mută cheia k pe prima poziție în nodul $y_{i+1} = x.c(i+1)$
  - mută ultimul fiu al lui y_i ca prim fiu al lui y_{i+1}.
- O rotație la stânga în jurul cheii $k = x.key(i)$:
  - mută prima cheie a fiului $y_{i+1} = x.c(i+1)$ în nodul x în locul cheii k
  - mută cheia k pe ultima poziție din nodul $y_i = x.c(i)$
  - mută primul fiu al lui y_{i+1} ca ultim fiu al lui y_i.

B-Arbori - Fuziune



Fuziunea:

- nu poate fi realizată decât între doi fraţi vecini, fii ai aceluiaşi nod x
- fiecare dintre cele două noduri care fuzionează are exact $t - 1$ chei
- nodul x are cel puţin t chei
- nodul rezultat este plin

B-Arbori - Ștergerea unei chei

Observații:

- Înălțimea unui arbore se micșorează atunci când rădăcina are doi fii care fuzionează
- O fuziune se poate realiza doar dacă părintele nodurilor care fuzionează are cel puțin t chei $\Rightarrow$ pe parcursul căutării cheii, toate nodurile pline întâlnite pe parcurs trebuie divizate.

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și *key* este cheie a lui x $\Rightarrow$ șterge *key* din x STOP

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' = \text{predecesor}(key)$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' =$ predecesor(key)
    - $x.key[i] = k'$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' = \text{predecesor}(key)$
    - $x.key[i] = k'$
    - $\text{STERGE}(y, k')$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui x $\Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' =$ predecesor(key)
    - $x.key[i] = k'$
    - STERGE(y, k')

  (b) **Altfel dacă** $y = x.c[i+1]$ are $y.n \geq t$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui x $\Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' =$ predecesor(key)
    - $x.key[i] = k'$
    - STERGE(y, k')

  (b) **Altfel dacă** $y = x.c[i+1]$ are $y.n \geq t$
    - $k' =$ succesor(key)

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
    - $k' = \text{predecesor}(key)$
    - $x.key[i] = k'$
    - STERGE(y, k')

  (b) **Altfel dacă** $y = x.c[i+1]$ are $y.n \geq t$
    - $k' = \text{succesor}(key)$
    - $x.key[i] = k'$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul I**: x este frunză și key este cheie a lui $x \Rightarrow$ șterge key din x STOP
- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (a) **Dacă** $y = x.c[i]$ are $y.n \geq t$ atunci
  - $k' =$ predecesor(key)
  - $x.key[i] = k'$
  - STERGE(y, k')

  (b) **Altfel dacă** $y = x.c[i+1]$ are $y.n \geq t$
  - $k' =$ succesor(key)
  - $x.key[i] = k'$
  - STERGE(y, k')

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.

Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  - (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei
    - y = Fuziune(x.c[i], x.c[i+1], key)

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei

  - y = Fuziune(x.c[i], x.c[i+1], key)
  - STERGE(y, key)

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei

    - y = Fuziune(x.c[i], x.c[i+1], key)
    - STERGE(y, key)

- **Cazul III**: x NU e frunză și $x.key(i-1) < key < x.key(i)$

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$

  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei

  - y = Fuziune(x.c[i], x.c[i+1], key)
  - STERGE(y, key)

- **Cazul III**: x NU e frunză și $x.key(i-1) < key < x.key(i)$

  (a) Dacă $x.c[i]$ are cel puțin t chei STERGE(x.c[i], key)

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei
    - y = Fuziune(x.c[i], x.c[i+1], key)
    - STERGE(y, key)
- **Cazul III**: x NU e frunză și $x.key(i-1) < key < x.key(i)$
  (a) Dacă $x.c[i]$ are cel puțin t chei STERGE(x.c[i], key)
  (b) Dacă $x.c[i]$ are $t-1$ chei, dar are un frate vecin cu cel puțin t chei $\Rightarrow$ rotație dinspre acel vecin, apoi STERGE(x.c[i], key)

B-Arbori - Ștergerea unei chei

Algoritm STERGE(x, key) - x=nodul curent, key - cheia care trebuie ștearsă.
Observație: x are cel puțin t chei!

- **Cazul II**: x NU e frunză, dar $key = x.key[i]$
  (c) **Altfel** - adică ambii copii $x.c[i], x.c[i+1]$ au exact $t-1$ chei
    - y = Fuziune(x.c[i], x.c[i+1], key)
    - STERGE(y, key)
- **Cazul III**: x NU e frunză și $x.key(i-1) < key < x.key(i)$
  (a) Dacă $x.c[i]$ are cel puțin t chei STERGE(x.c[i], key)
  (b) Dacă $x.c[i]$ are $t-1$ chei, dar are un frate vecin cu cel puțin t chei $\Rightarrow$ rotație dinspre acel vecin, apoi STERGE(x.c[i], key)
  (b) Dacă $x.c[i]$ are $t-1$ chei și frații (fratele) vecini au tot $t-1$ chei $\Rightarrow$ fuziune cu un frate, apoi STERGE(x.c[i], key)

B-Arbori - Exerciții

1. Desenați toți B-arborii cu $t = 2$ care conțin exact cheile A, B, C, D, E. Fiecare cheie apare o singură dată în fiecare arbore.

2. Care este numărul maxim de chei, care poate fi stocat într-un B-arbore cu $t = 3$ și înălțimea $h = 3$?

3. Inserați într-un B-arbore inițial vid cu $t = 2$ următoarele chei: 10, 7, 9, 24, 25, 31, 14, 44, 17, 0, 8, 12, 11