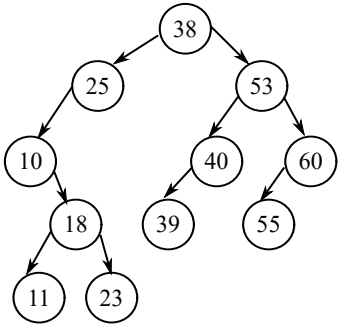
Arbori binari de căutare

Universitatea "Transilvania" din Brașov

4 aprilie 2022

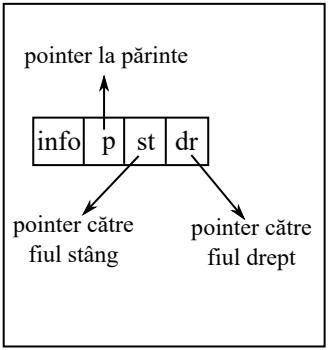
Arbore binar de căutare



Definiție: un arbore binar de căutare este un arbore binar cu următoarele proprietăți.

- Fiecare nod are o valoare numită cheie
- Pentru fiecare nod este valabil:
  - Toate nodurile din subarborele stâng au cheile mai mici decât cheia părintelui.
  - Toate nodurile din subarborele drept au cheile mai mari decât cheia părintelui.

Reprezentarea unui arbore binar



Reprezentarea unui arbore binar ca structură de pointeri:

Fiecare nod $x$este considerat o structură cu 4 câmpuri

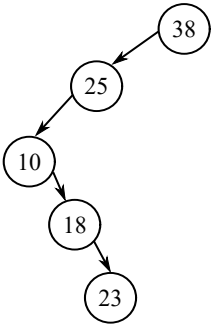
- $x.info$ - cheia nodului
- $x.p$ - legătura către nodul părinte
- $x.st$ - legătura către fiul stâng
- $x.dr$ - legătura către fiul drept

Reprezentarea unui arbore binar - cod C++

```
struct nod{
        int info;
        nod* p, *st, *dr;
};

struct ARB{
        nod* rad;
        //functii corespunzatoare
};
```

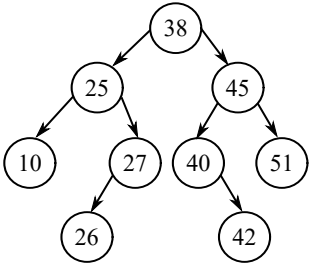
Arbore binar de căutare - Înălțime



Înălțimea maximă a unui arbore binar de căutare cu n noduri este $n - 1$

Exemplu: în figură $n = 5$, $h = 4$.

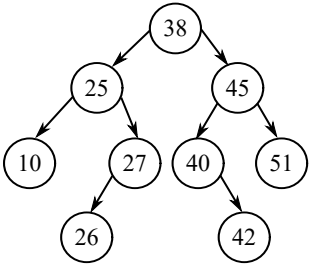
Arbore binar de căutare - Înălțime



Înălțimea minimă a unui arbore binar de căutare cu n noduri este $[\log_2 n]$

Exemplu: în figură $n = 9$, $h = 3$.

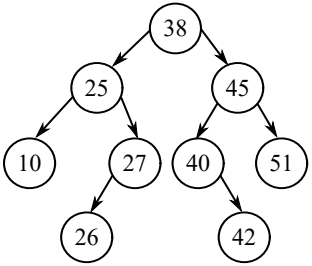
Arbore binar de căutare - Parcurgeri



Parcurgeri:

- RSD (Preordine):
- SRD (Inordine):
- SDR (Postordine):

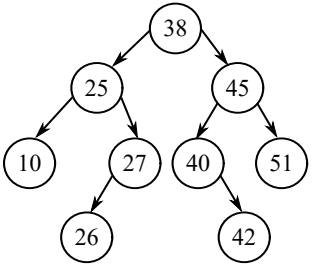
Arbore binar de căutare - Parcurgeri



Parcurgeri:

- RSD (Preordine): 38, 25, 10, 27, 26, 45, 40, 42, 51
- SRD (Inordine):
- SDR (Postordine):

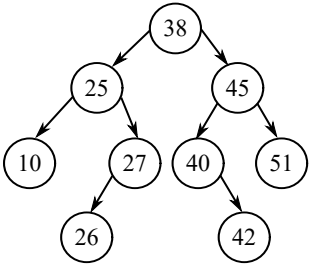
Arbore binar de căutare - Parcurgeri



Parcurgeri:

- RSD (Preordine): 38, 25, 10, 27, 26, 45, 40, 42, 51
- SRD (Inordine): 10, 25, 26, 27, 38, 40, 42, 45, 51
- SDR (Postordine):

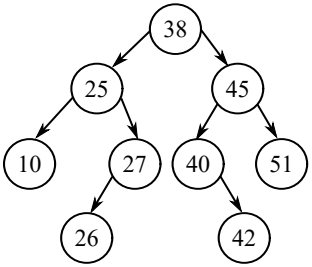
Arbore binar de căutare - Parcurgeri



Parcurgeri:

- RSD (Preordine): 38, 25, 10, 27, 26, 45, 40, 42, 51
- SRD (Inordine): 10, 25, 26, 27, 38, 40, 42, 45, 51
- SDR (Postordine): 10, 26, 27, 25, 42, 40, 51, 45, 38

Arbore binar de căutare - Parcurgeri



Parcurgeri:

- RSD (Preordine): 38, 25, 10, 27, 26, 45, 40, 42, 51
- SRD (Inordine): 10, 25, 26, 27, 38, 40, 42, 45, 51
- SDR (Postordine): 10, 26, 27, 25, 42, 40, 51, 45, 38

Observație: Parcurgerea în inordine a unui arbore binar de căutare are ca rezultat parcurgerea în ordine sortată a cheilor din arbore.

Arbore binar de căutare - Operații

Operațiile principale:

- Căutarea binară a unui nod.

Arbore binar de căutare - Operații

Operațiile principale:

- Căutarea binară a unui nod.
- Determinarea minimului/maximului.

Arbore binar de căutare - Operații

Operațiile principale:

- Căutarea binară a unui nod.
- Determinarea minimului/maximului.
- Căutarea binară a succesorului / predecesorului

Arbore binar de căutare - Operații

Operațiile principale:

- Căutarea binară a unui nod.
- Determinarea minimului/maximului.
- Căutarea binară a succesorului / predecesorului
- Inserție/ștergere a unui nod

Arbore binar de căutare - Operații

Operațiile principale:

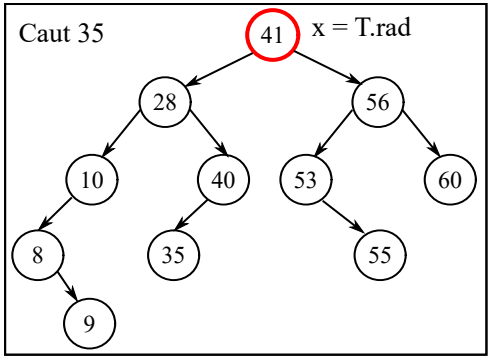
- Căutarea binară a unui nod.
- Determinarea minimului/maximului.
- Căutarea binară a succesorului / predecesorului
- Inserție/ștergere a unui nod
- Sortarea cheilor arborelui, prin parcurgerea acestuia în inordine a arborelui.

Arbore binar de căutare - Operații

Căutarea binară.
Problematică: Se consideră un și sortat de elemente. Să se verifice eficient, dacă un element x face parte din șir.

Exemplu: $v = \{3, 7, 9, 11, 15, 24, 27, 31\}$. $x = 9$.

Arbore binar de căutare - Operații



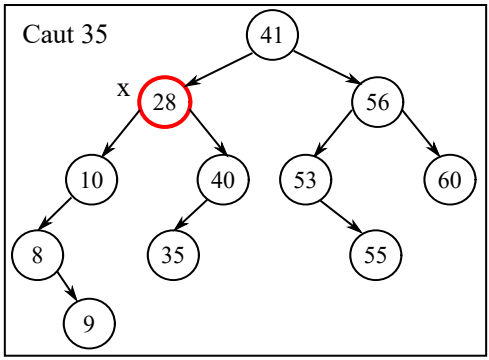
1. Căutarea binară în arbore.

Se caută în arborle T e valoarea k. În exemplu: $k = 35$.

Se pornește cu $x = T.rad$ și cât timp mai pot căuta ($x \neq Nil$):

- se compară k cu $x.info$
- dacă $k = x.info \Rightarrow$ succes
- altfel dacă $k < x.info$ caută în subarborele stâng $x.st$
- altfel caută în subarborele drept $x.dr$

Arbore binar de căutare - Operații



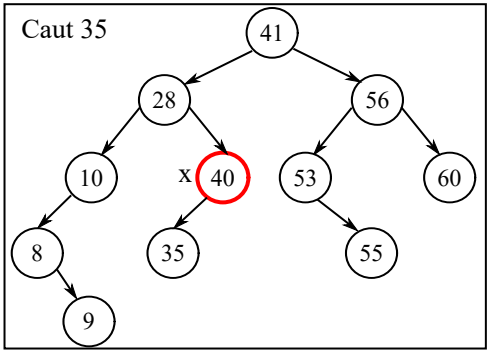
1. Căutarea binară în arbore.

Se caută în arborle T e valoarea k. În exemplu: $k = 35$.

Se pornește cu $x = T.rad$ și cât timp mai pot căuta ($x \neq Nil$):

- se compară k cu $x.info$
- dacă $k = x.info \Rightarrow$ succes
- altfel dacă $k < x.info$ caută în subarborele stâng $x.st$
- altfel caută în subarborele drept $x.dr$

Arbore binar de căutare - Operații



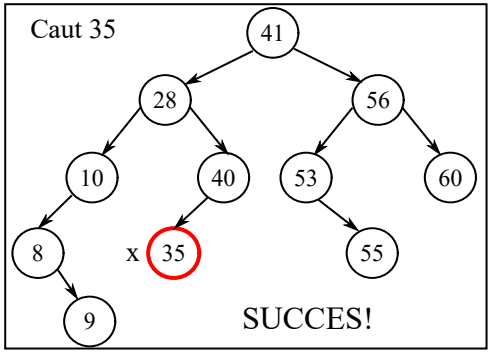
1. Căutarea binară în arbore.

Se caută în arborle T e valoarea k. În exemplu: $k = 35$.

Se pornește cu $x = T.rad$ și cât timp mai pot căuta ($x \neq Nil$):

- se compară k cu $x.info$
- dacă $k = x.info \Rightarrow$ succes
- altfel dacă $k < x.info$ caută în subarborele stâng $x.st$
- altfel caută în subarborele drept $x.dr$

Arbore binar de căutare - Operații



1. Căutarea binară în arbore.

Se caută în arborle T e valoarea k. În exemplu: $k = 35$.

Se pornește cu $x = T.rad$ și cât timp mai pot căuta ($x \neq Nil$):

- se compară k cu $x.info$
- dacă $k = x.info \Rightarrow$ succes
- altfel dacă $k < x.info$ caută în subarborele stâng $x.st$
- altfel caută în subarborele drept $x.dr$

Căutare binară în arbore - Algoritm

Algoritm 1: Căutare binară

Intrare: Arborele binar de căutare T și elementul k care se caută

Iesire: nodul x cu cheia k sau Nil

$x \leftarrow T.rad$

cat_timp $x \neq Nil$ *și* $x.info \neq k$ **executa**

    daca $k < x.info$ **atunci**

        $x \leftarrow x.st$

    sfarsit_daca

    altfel

        $x \leftarrow x.dr$

    sfarsit_daca

sfarsit_cat_timp

return x

Arbori binari de căutare - Operații

Observații:

1. Complexitatea operațiilor într-un arbore binar de căutare cu n noduri este proporțională cu înălțimea h a arborelui.

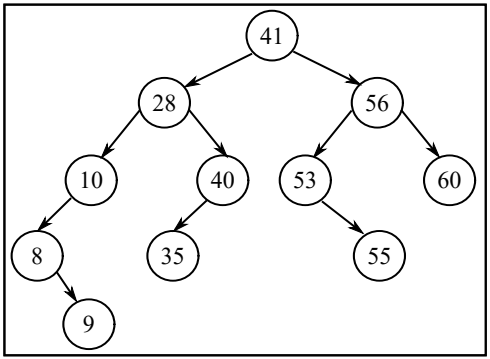
   Notăm cu $T(n)$ complexitatea algoritmului utilizat, atunci:

   $$O(log_2 n) \leq T(n) \leq O(n)$$

2. În cazul unui arbore binar de căutare oarecare nu poate fi garantată complexitatea căutării binare, adică $O(log_2 n)$.

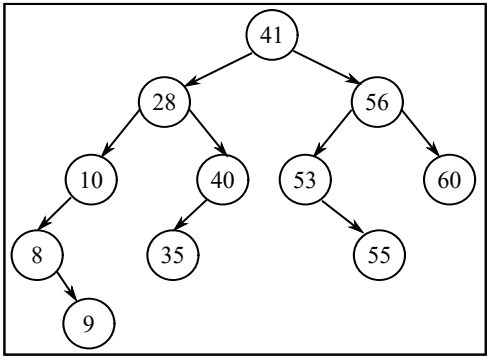
Există arbori binari de căutare care se autobalansează, de exemplu arborii AVL și arborii roșu-negru. Pentru aceștia se demonstrează faptul că au complexitatea operațiilor de ordinul $O(\log_2 n)$.

Arbori binari de căutare - Operații



2. Căutarea minimului într-un subarbore de rădăcină x.

Arbori binari de căutare - Operații



2. Căutarea minimului într-un subarbore de rădăcină x.

Algoritm 2: Minim

Intrare: Nodul x

Iesire: nodul cu cheia minimă din subarbore

$y \leftarrow x$

cat_timp $y.st \neq Nil$ **executa**

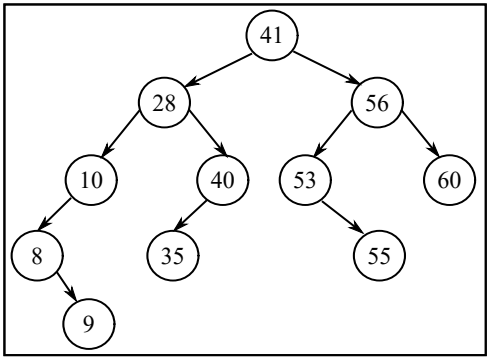
| $y \leftarrow y.st$

sfarsit_cat_timp

return y

Complexitate: $O(h)$

Arbori binari de căutare - Operații

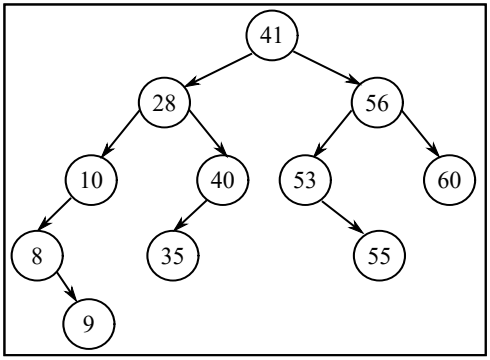


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = ?
SUCCESOR(28) = ?

Arbori binari de căutare - Operații

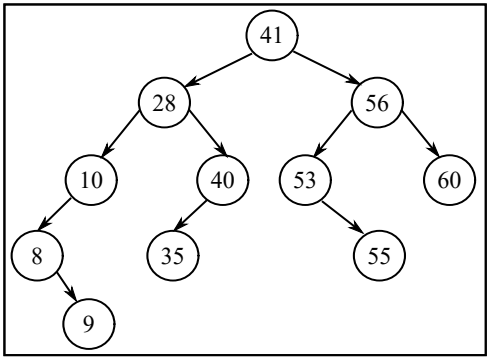


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = ?

Arbori binari de căutare - Operații

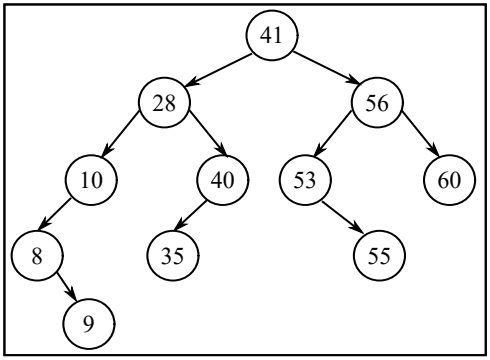


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = 35

Arbori binari de căutare - Operații

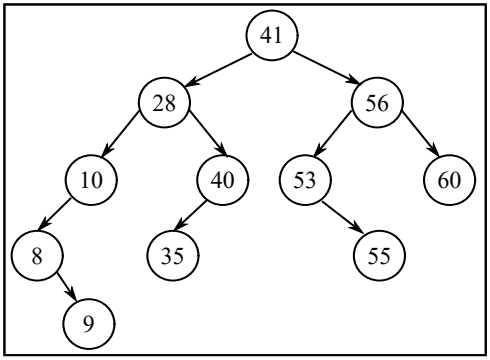


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = 35
SUCCESOR(9) = ?

Arbori binari de căutare - Operații

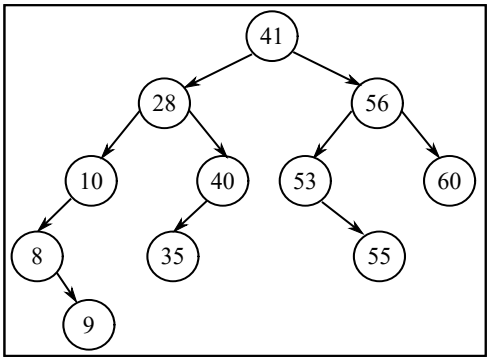


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = 35
SUCCESOR(9) = 10
SUCCESOR(55) = ?

Arbori binari de căutare - Operații

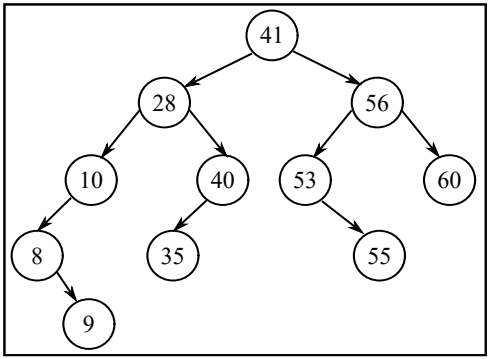


3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = 35
SUCCESOR(9) = 10
SUCCESOR(55) = 56
SUCCESOR(60) =?

Arbori binari de căutare - Operații



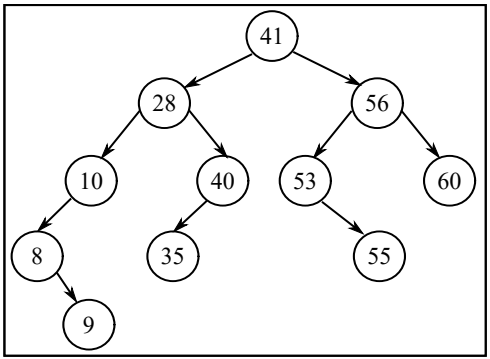
3. Succesorul binar al unui nod x = acel nod y a cărui cheie are valoarea imediat următoare cheii lui x în șirul sortat al valorilor din arbore.

Exemplu:

SUCCESOR(41) = 53
SUCCESOR(28) = 35
SUCCESOR(9) = 10
SUCCESOR(55) = 56
SUCCESOR(60) = NU are. Este maximul.

Observație: dacă x are fiu drept, atunci succesorul său NU are fiu stâng.

Arbori binari de căutare - Operații



3. Succesorul binar al unui nod x

- Poate fi determinat prin comparații
- Dacă există, este:
  - dacă $x.dr \neq Nil \Rightarrow$ succesorul este cel mai mic element din $x.dr$,
  - altfel - un nod părinte y pentru care x se află în subarborele stâng al lui y.
- Dacă x este nodul cu cea mai mare cheie, atunci x nu are succesor.
- Dacă x are fiu drept, atunci uccesorul său NU are fiu stâng.

Succesorul binar al unui nod - Algoritm

Algoritm 3: Succesor binar

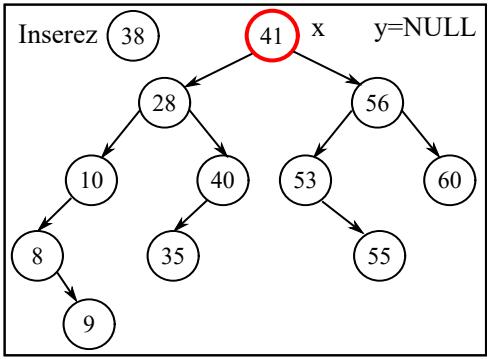
Intrare: Arborele binar de căutare T și nodul x

Iesire: nodul y, sucesor binar al lui x

daca $x.dr \neq Nil$ **atunci**
  $y = \text{MINIM}(x.dr)$
sfarsit_daca
altfel
  $y \leftarrow x.p$
  cat_timp $y \neq Nil$ și $x = y.dr$ **executa**
    $x \leftarrow y$
    $y \leftarrow y.p$
  sfarsit_cat_timp
sfarsit_daca
return y

Complexitate: $O(h)$

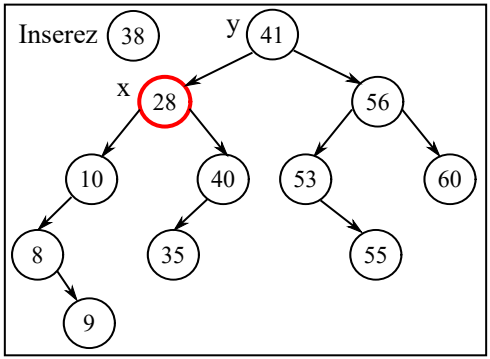
Arbori binari de căutare - Operații



4. Inserția unui nod z

- se pornește de la rădăcină
- se coboară în arbore, până la un nod y, care are cel mult un fiu și care poate fi părintele nodului z.
- se legă z de y

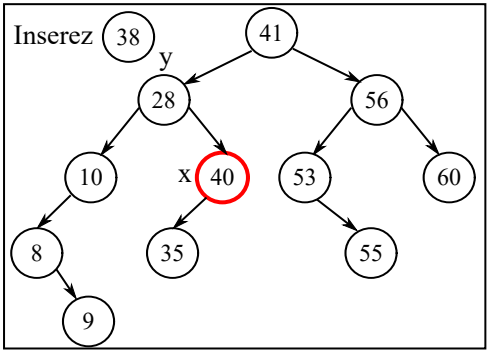
Arbori binari de căutare - Operații



4. Inserția unui nod z

- se pornește de la rădăcină
- se coboară în arbore, până la un nod y, care are cel mult un fiu și care poate fi părintele nodului z.
- se leagă z de y

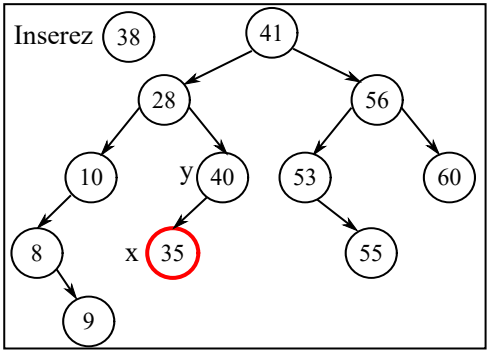
Arbori binari de căutare - Operații



4. Inserția unui nod z

- se pornește de la rădăcină
- se coboară în arbore, până la un nod y, care are cel mult un fiu și care poate fi părintele nodului z.
- se legă z de y

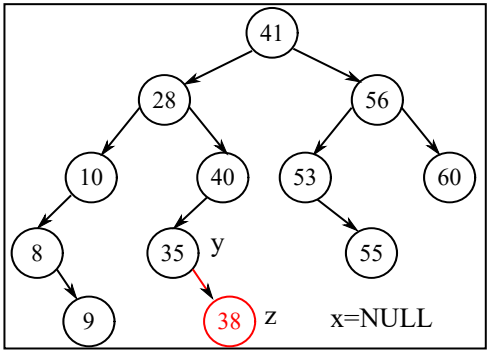
Arbori binari de căutare - Operații



4. Inserția unui nod z

- se pornește de la rădăcină
- se coboară în arbore, până la un nod y, care are cel mult un fiu și care poate fi părintele nodului z.
- se legă z de y

Arbori binari de căutare - Operații



4. Inserția unui nod z

- se pornește de la rădăcină
- se coboară în arbore, până la un nod y, care are cel mult un fiu și care poate fi părintele nodului z.
- se legă z de y

Inserarea unui nod - Algoritm

Algoritm 4: Insert

Intrare: Arborele binar de căutare T și nodul z

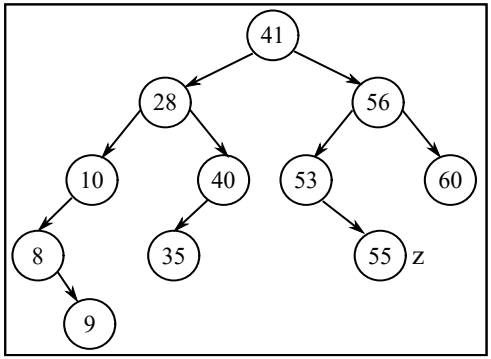
Iesire: Arborele în care s-a inserat z

$x \leftarrow T.rad$
$y \leftarrow Nil$
cat_timp $X \neq Nil$ **executa**
    $y \leftarrow x$
    daca $z.info < x.info$ **atunci**
        $x \leftarrow x.st$
    sfarsit_daca
    altfel
        $x \leftarrow x.dr$
    sfarsit_daca
sfarsit_cat_timp
$z.p \leftarrow y$
daca $y = Nil$ **atunci**
    $T.rad \leftarrow z$
altfel
    daca $z.info < y.info$ **atunci**
        $y.st \leftarrow z$
    altfel
        $y.dr \leftarrow z$

Complexitate: $O(h)$

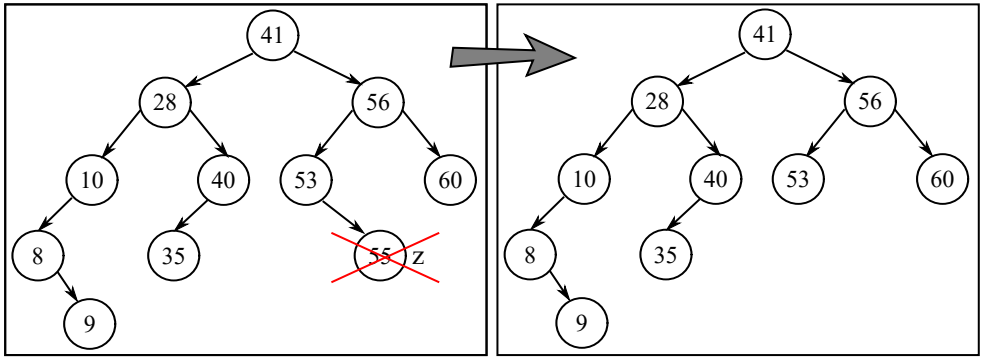
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 55?



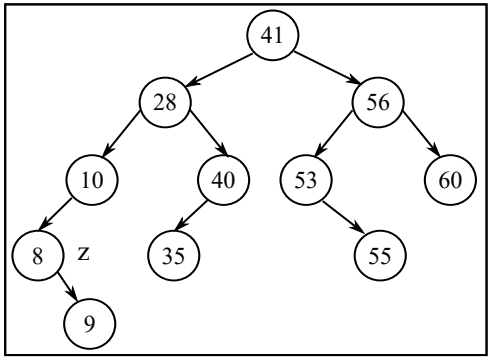
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 55?



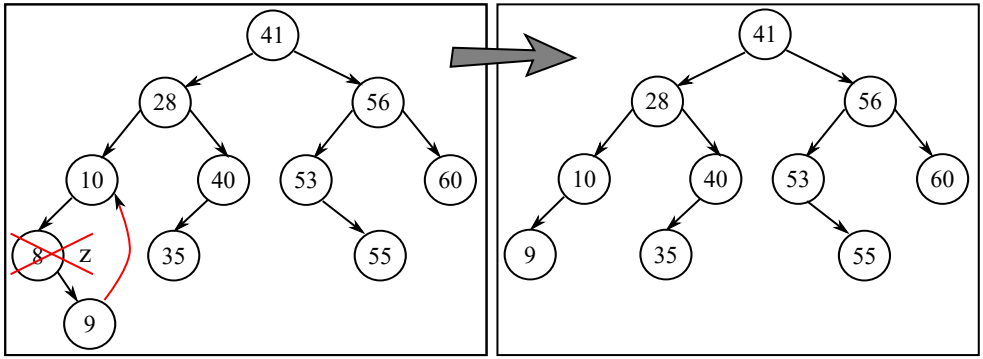
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 8?



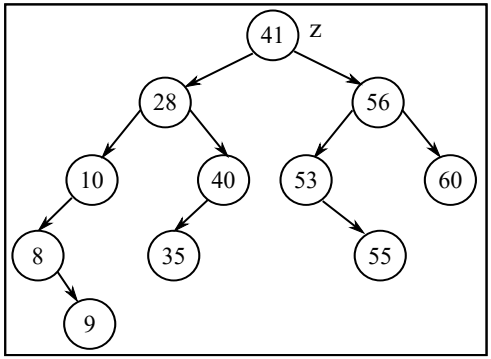
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 8?



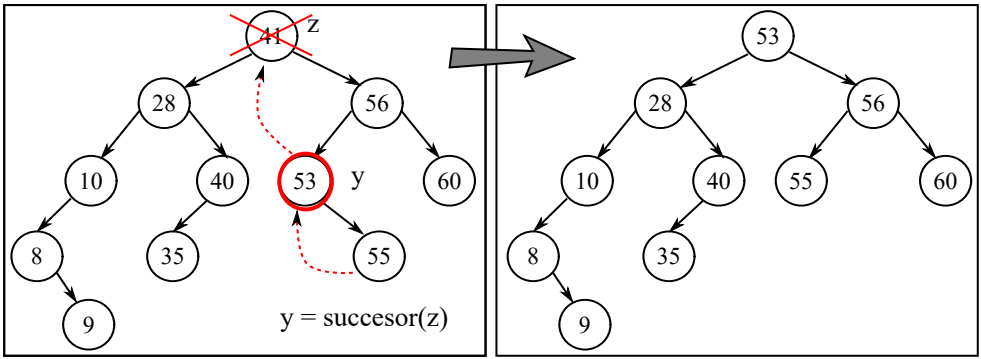
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 41?



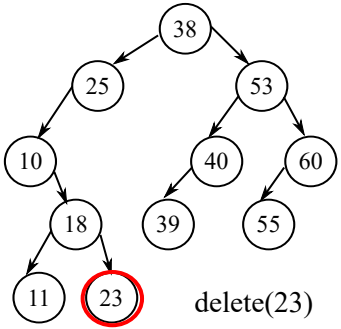
Arbori binari de căutare - Operații

5. Ștergerea unui nod. Cum șterg din arborele din figură nodul cu cheia 41?



1. Caut succesorul y al lui z
2. Înlocuiesc nodul z cu nodul y
3. Înlocuiesc nodul y cu fiul său drept

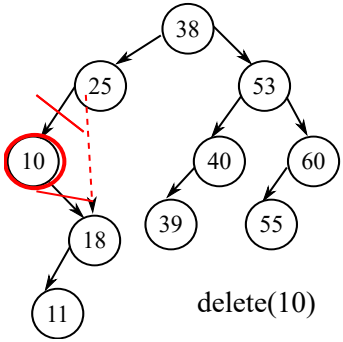
Arbori binari de căutare - Operații



5. Ștergerea unui nod. Sunt luate în considerare următoarele cazuri:

1. z nu are fii și atunci este pur și simplu înlocuit cu Nil

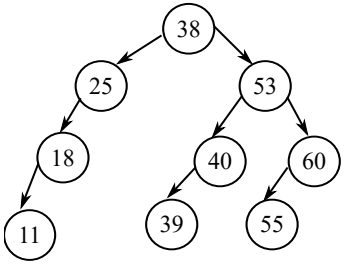
Arbori binari de căutare - Operații



5. Ștergerea unui nod. Sunt luate în considerare următoarele cazuri:

1. z nu are fii și atunci este pur și simplu înlocuit cu Nil
2. z are un singur fiu nenul. Atunci se înlocuiește z cu acel fiu

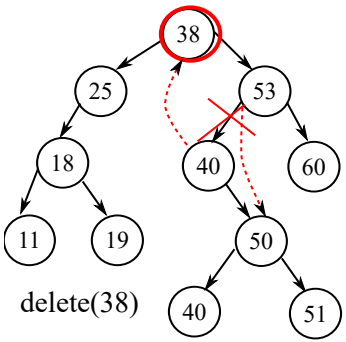
Arbori binari de căutare - Operații



5. Ștergerea unui nod. Sunt luate în considerare următoarele cazuri:

1. z nu are fii și atunci este pur și simplu înlocuit cu Nil
2. z are un singur fiu nenul. Atunci se înlocuiește z cu acel fiu

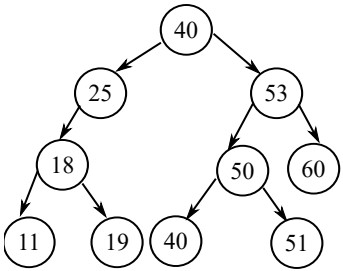
Arbori binari de căutare - Operații



5. Ștergerea unui nod. Sunt luate în considerare următoarele cazuri:

1. z nu are fii și atunci este pur și simplu înlocuit cu Nil
2. z are un singur fiu nenul. Atunci se înlocuiește z cu acel fiu
3. z are doi fii nenuli. Atunci se determină succesorul y al lui z care se află în subarborele drept al lui z și evident nu are descendent stâng. Apoi se înlocuiește nodul z cu nodul y, iar y se înlocuiește cu fiul său drept.

Arbori binari de căutare - Operații

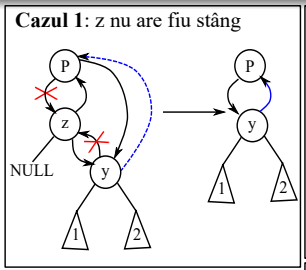


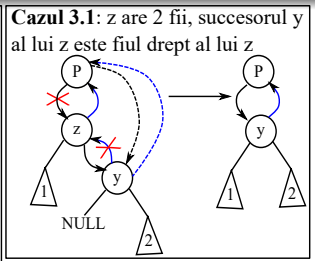
5. Ștergerea unui nod. Sunt luate în considerare următoarele cazuri:

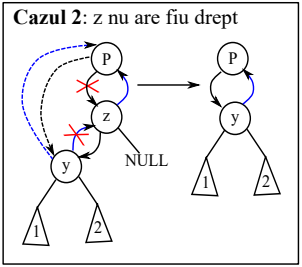
1. z nu are fii și atunci este pur și simplu înlocuit cu Nil
2. z are un singur fiu nenul. Atunci se înlocuiește z cu acel fiu
3. z are doi fii nenuli. Atunci se determină succesorul y al lui z care se află în subarborele drept al lui z și evident nu are descendent stâng. Apoi se înlocuiește nodul z cu nodul y, iar y se înlocuiește cu fiul său drept.

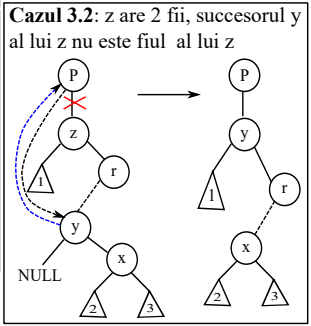
Observație: În cazul în care z are doi descendenți nenuli, el poate fi înlocuit și cu predecesorul său.

Arbori binari de căutare - Operații







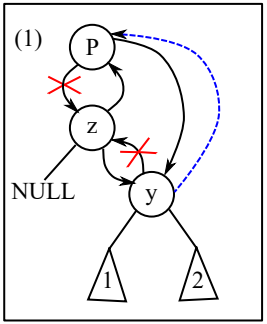


Algoritm 5: Transplant(T,z,y)

daca $z.p = Nil$ **atunci**
  $T.rad \leftarrow y$
sfarsit_daca
altfel
  daca $z = z.p.st$ **atunci**
    $z.p.st \leftarrow y$
  sfarsit_daca
  altfel
    $z.p.dr \leftarrow y$
  sfarsit_daca
sfarsit_daca
daca $y \neq Nil$ **atunci**
  $y.p \leftarrow z.p$
sfarsit_daca

Arbori binari de căutare - Operații - Ștergerea nodului z

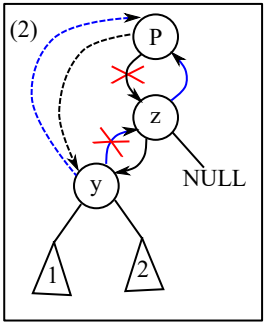
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

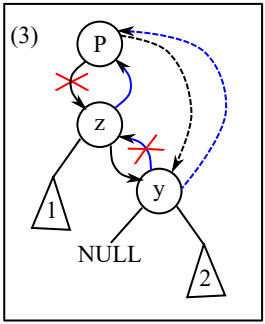
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

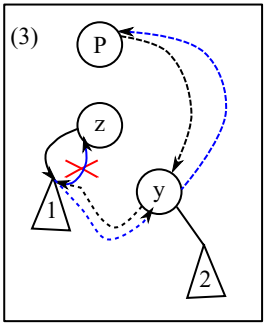
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

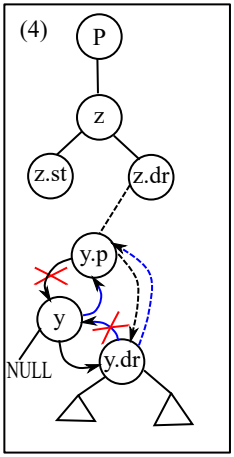
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

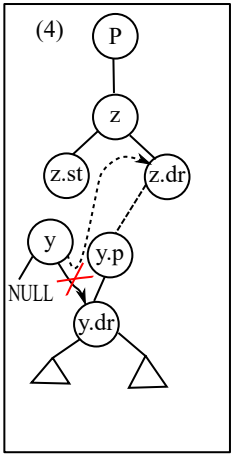
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

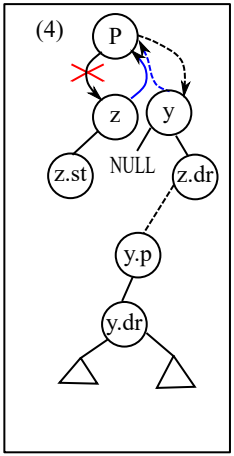
Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

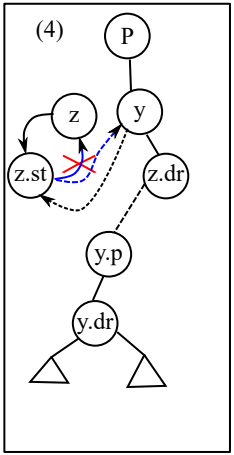
Algoritm 7: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Arbori binari de căutare - Operații - Ștergerea nodului z

Algoritm 6: Delete(T, z)



daca $z.st = Nil$ **atunci**
  Tansplant(T,z, $z.dr$)
altfel
  daca $z.dr = Nil$ **atunci**
    Tansplant(T, z, $z.st$)
  altfel
    $y \leftarrow succesor(z)$
    daca $y \neq z.dr$ **atunci**
      Transplant(T, y, $y.dr$)
      $y.dr \leftarrow z.dr$
      $z.dr.p \leftarrow y$
    Transplant(T, z, y)
    $y.st \leftarrow z.st$
    $z.st.p \leftarrow y$

Exercții

1. Care este numărul minim de chei într-un arbore binar (de căutare sau nu) cu înălțimea h?
2. Care este numărul maximde chei într-un arbore binar (de căutare sau nu) cu înălțimea h?
3. Să se insereze pe rând într-un arbore binar de căutare următoarele chei: 15, 7, 10, 3, 2, 9, 24, 20, 18, 35. Cum arată rborele final?
4. Cunoscându-se parcurgerea în preordine a unui arbore binar de căutare, să se refacă arborele. Exemplu: RSD: 23, 17, 10, 15, 19, 35, 26, 24, 30, 37.

Containere asociative sortate - Set/ Map

```
template<class Key, class Compare = std::less<Key>, class Allocator =
std::allocator<Key>> class set;
template<class Key, class T, class Compare = std::less<Key>, class
Allocator = std::allocator<std::pair<const Key,T> >> class map;
```

- mulțime sortată de elemente, care nu permit dubluri

Containere asociative sortate - Set/ Map

```
template<class Key, class Compare = std::less<Key>, class Allocator =
std::allocator<Key>> class set;
template<class Key, class T, class Compare = std::less<Key>, class
Allocator = std::allocator<std::pair<const Key,T> >> class map;
```

- mulțime sortată de elemente, care nu permit dubluri
- sortarea se face pe baza clasei **Compare**

Containere asociative sortate - Set/ Map

```
template<class Key, class Compare = std::less<Key>, class Allocator =
std::allocator<Key>> class set;
template<class Key, class T, class Compare = std::less<Key>, class
Allocator = std::allocator<std::pair<const Key,T> >> class map;
```

- mulțime sortată de elemente, care nu permit dubluri
- sortarea se face pe baza clasei **Compare**
- *Key* - trebuie să fie un tip primitiv - int, double, string, etc (tipuri care permit comparația cu $<$). în caz contrar trebuie definit un comparator pentru chei.

Containere asociative sortate - Set/ Map

```
template<class Key, class Compare = std::less<Key>, class Allocator =
std::allocator<Key>> class set;
template<class Key, class T, class Compare = std::less<Key>, class
Allocator = std::allocator<std::pair<const Key,T> >> class map;
```

- mulțime sortată de elemente, care nu permit dubluri
- sortarea se face pe baza clasei **Compare**
- *Key* - trebuie să fie un tip primitiv - int, double, string, etc (tipuri care permit comparația cu $<$). în caz contrar trebuie definit un comparator pentru chei.
- implementat printr-un arbore binar de căutare care se auto-balansează - de obicei Roșu - Negru

Containere asociative sortate - Set/ Map

```
template<class Key, class Compare = std::less<Key>, class Allocator =
std::allocator<Key>> class set;
template<class Key, class T, class Compare = std::less<Key>, class
Allocator = std::allocator<std::pair<const Key,T> >> class map;
```

- mulțime sortată de elemente, care nu permit dubluri
- sortarea se face pe baza clasei **Compare**
- *Key* - trebuie să fie un tip primitiv - int, double, string, etc (tipuri care permit comparația cu $<$). în caz contrar trebuie definit un comparator pentru chei.
- implementat printr-un arbore binar de căutare care se auto-balansează - de obicei Roșu - Negru
- operații în timp logaritmic

Containere asociative sortate - Set/ Map

```
std::map<int, int> m;
```

Funcții membre

dimensiune	modificare	căutare
size empty	insert / emplace erase / clear	find count

Containere asociative sortate- Map

Exemplul 1

```
#include <iostream>
#include <map>
void main (){
    std::map<char, int> mymap;
    std::map<char, int>::iterator it;
    mymap['a'] = 50; mymap['b'] = 100;
    mymap['c'] = 150; mymap['d'] = 200;
    it = mymap.find('b');
    if (it != mymap.end())
      mymap.erase(it);
    // print content:
    std::cout << "elements in mymap:" << endl;
    std::cout << "a => " << mymap.find('a')- >second <<endl;
    std::cout << "c => " << mymap.find('c')- >second <<endl;
    std::cout << "d => " << mymap.find('d')- >second <<endl;
}
```

Containere asociative sortate - Map

Exemplul 2

```
#include <iostream>
#include <map>
void main (){
    std::map<std::string, int> mymap2 = {{ "alpha", 0 },
                                         { "beta", 0 }, { "gamma", 0 }};
    mymap2.at("alpha") = 10;
    mymap2.at("beta") = 20;
    mymap2.at("gamma") = 30;
    for (auto x : mymap2) {
        std::cout << x.first <<": "<<x.second << endl;
}
```

Containere asociative sortate - Map

Exemplu de clasa comparator pentru obiecte de tip pointer la nod

```
class Compare
{
public:
    bool operator() (nod* a, nod* b)
    {
        if (a->frecquency>b->frecquency)
            return true;
        return false;
    }
};

nod* Construct(int CountChar[])
{
    std::priority_queue<nod*, std::vector<nod*>, Compare> Coada;

    /// linii de cod
    ///
}
```