ARBORI. Continuare.

6. ARBORI. Continuare.

Definitii:
- arborescenta
- arbore ordonat
- descendentii directi ai radacinii
- fii nodului, tatal/parintele descendentilor
- noduri frati , terminal sau frunza
- Inaltime arbore
- vector de tati
- Parcurgerea arborilor cu radacina
- Parcurgere in Preordine
- Parcurgere postordine
- arbore binar
- arbore binar complet
- arbore de cautare

Proprietati. Teoreme

- Proprietati arbori binari (1-5)

-parcurgerea arborilor binari

-in latime (BFS)

-in adancime (preordine,inordine, ostordine) (DFS)

-Arbori de cautare: Organizare, reprezentare

- Arbori cu radacina
- Parcurgerea arborilor cu radacina
- Arbori binari
- Parcurgerea arborilor binari
- Reprezentarea arborilor binari
  - Reprezentarea statica

    a) Reprezentarea standard

    a') pt. nod se precizeaza si parintele, nu doar fii

    b) Legaturi de tip tata

    c) Reprezentarea cu paranteze
  - Reprezentarea dinamica
- Arbori de cautare
- Operatii asupra arborilor de cautare (Interogari)
  - o cautarea unui nod care memoreaza o cheie data - SearchKey
  - o Det.cheii minime - MINKey
  - o Det.cheii maxime - MAXKey
  - o Det. valorii ce succede/precede o cheie -SuccPred

Algoritmi (descriere+pseudocod+ex.)
- SearchKey: Recursiv si iterativ: Cautare in arbore de cautare
- MINKey: Recursiv si iterativ: Det. nod cu valoarea cheii minima
- MAXKey:Recursiv si iterativ: Det. nod cu valoarea cheii maxima

Succesorul si predecesorul unui nod
- Successor: alternative: subarbore drept nod este sau nu vid
- Predecessor

- Tinsert: cheie data k arbore cu radacina r: Recursiv si iteratív
- Tdelete: Stergere in functie de nr. de fii

Arbori cu rădăcină

- fie $T = (X, U)$ un arbore (graf conex și fără cicluri)

- un nod $r \in X$ al acestui arbore se poate alege ca nod special, numit **rădăcina arborelui**

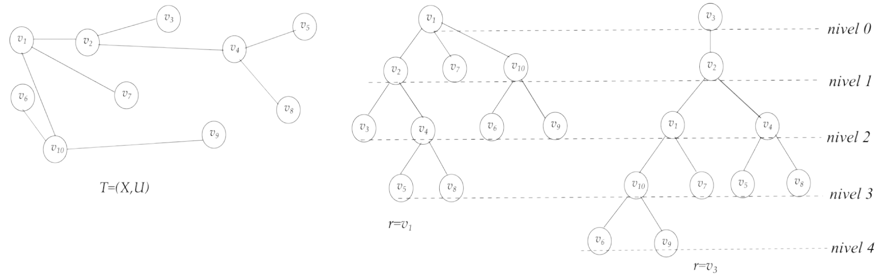
- există un lanț unic de la rădăcina r la celelalte noduri ale grafului (conform teoremei de caracterizare a unui arbore)

- alegerea rădăcinii duce la așezarea arborelui pe niveluri astfel:

  rădăcina este nod pe nivelul 0

  pe fiecare nivel k ($k \geq 1$) se plasează acele vârfuri pentru care lungimea lanțurilor care le leagă de rădăcină este k

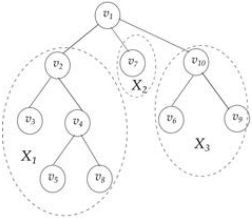
- un astfel de arbore se va numi **arbore cu rădăcină** sau **arborescență**



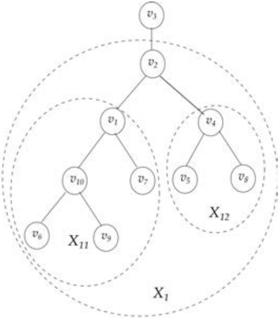
Arbori cu rădăcină

Definiție

Se numește arborescență un arbore care are un vârf r numit rădăcină, iar celelalte noduri pot fi repartizate în mulțimi disjuncte $X_1, X_2, \dots, X_k$, $X_i \cap X_j = \emptyset$, $k > 0$, $1 \le i < j \le k$, $X_1 \cup X_2 \cup \dots \cup X_k = X \setminus \{r\}$ astfel încât în fiecare din aceste mulțimi există un nod adiacent cu rădăcina, iar subgrafurile generate de acestea sunt la rândul lor arborescențe.



$r = v_1$



$r = v_3$

Arbori cu rădăcină

- în particular, o arborescență cu un singur nod este formată doar din nodul rădăcină

- dacă ordinea relativă a arborescențelor generate de $X_1, X_2, \ldots., X_k$ are importanță, arborescența se numește arbore ordonat

- nodurile adiacente cu rădăcina se numesc descendenții direcți ai rădăcinii

- descendenții direcți ai unui nod $x \in X$ se numesc **fiii** nodului x, nodul x este **tatăl/părintele** descendenților săi direcți

- nodurile care au același tată se numesc **frați**

- un nod fără fii se numește **nod terminal sau frunză**

- se numește **înălțime** a unui arbore cu rădăcină, lungimea celui mai lung lanț de la rădăcină la un nod terminal

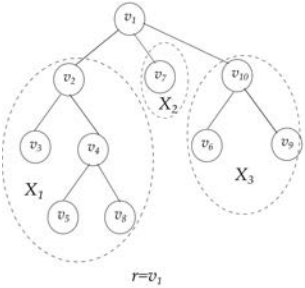
- numărul nivelurilor arborelui minus 1, este înălțimea arborelui

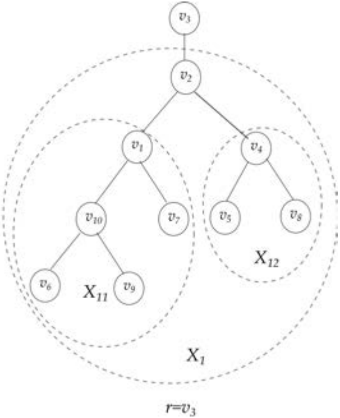
Definiție

Se numește **vector de tați** pentru arborele cu rădăcină $T = (X, U)$, cu mulțimea nodurilor $X = \{v_1, v_2, ..., v_n\}$, un vector t cu $n = |X|$ elemente definit prin

$$t[i] = \begin{cases} j, & daca\ v_i\ este\ fiu\ al\ nodului\ v_j \\ 0, & daca\ v_i = radacina \end{cases} \quad , \ i = \overline{1, n}$$

Arbori cu rădăcină





$$t_1 = (0,\ 1,\ 2,\ 2,\ 4,\ 10,\ 1,\ 4,\ 10,\ 1)$$

$$t_3 = (2,\ 3,\ 0,\ 2,\ 4,\ 10,\ 1,\ 4,\ 10,\ 1)$$

Parcurgerea arborilor cu rădăcină

- parcurgere: vizitarea în mod sistematic a nodurilor arborelui în scopul prelucrării informației atașate nodurilor sau a liniarizării nodurilor

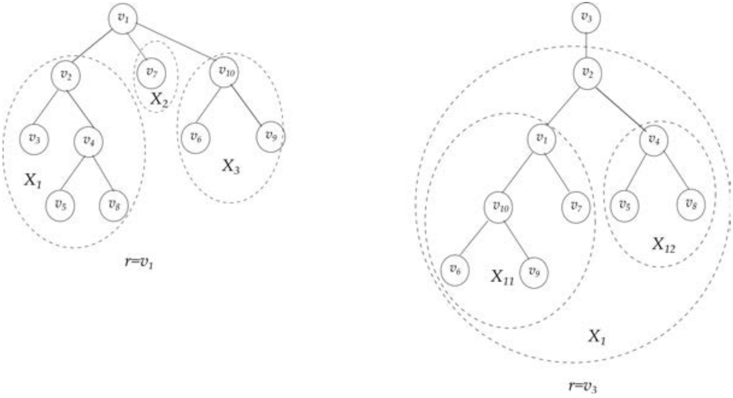
- un arbore oarecare poate fi parcurs astfel:

a) în preordine: se vizitează rădăcina și apoi subarborii de la stânga la dreapta în prordine

c) în postordine: se parcurg de la stânga la dreapta subarborii rădăcinii în postordine și apoi se vizitează rădăcina

- definirea celor două moduri de parcurgere este o definire recursivă

Parcurgerea arborilor cu rădăcină



preordine $v_1, v_2, v_3, v_4, v_5, v_8, v_7, v_{10}, v_6, v_9,$

postordine $v_3, v_5, v_8, v_4, v_2, v_7, v_6, v_9, v_{10}, v_1$

Arbori binari

Definiție

Un arbore binar $T = (X, U)$ este un graf vid, $X = \emptyset$ sau un arbore ordonat în care fiecare nod are cel mult doi fii.

- un arbore binar conține cel mult doi subarbori, pe care îi numim **subarbore stâng**, respectiv **subarbore drept**; aceștia pot obține prin suprimarea rădăcinii și a muchiilor incidente cu aceasta

Definiție

Un arbore binar $T = (X, U)$ se numește **complet**, dacă și numai dacă orice nod are 0 sau 2 fii.

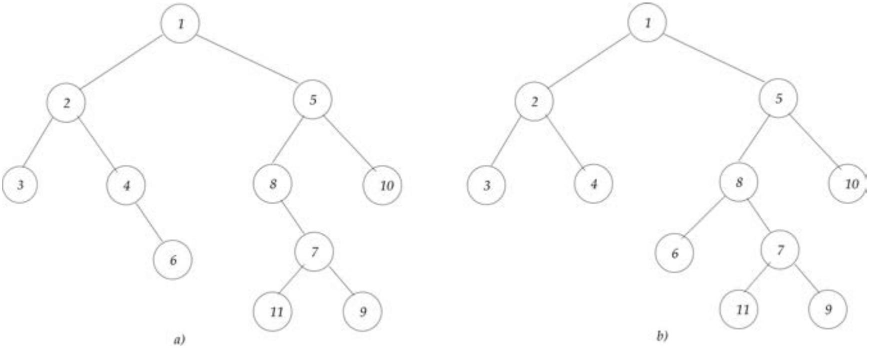


Figure: Arbori binari: a) binar oarecare b) binar complet

Arbori binari

Proprietati:

Fie $T = (X, U)$ un arbore binar cu $|X| = n$ noduri și cu înălțimea h. Atunci

1) numărul maxim de noduri de pe nivelul k al unui arbore este 2^k (rădăcina pe nivelul 0)

2) numărul maxim de noduri dintr-un arbore binar cu înălțimea h este $2^{h+1} - 1$

3) un arbore binar cu n noduri are înălțimea mai mare sau egală cu $[log_2 n]$

4) un arbore binar complet care are k noduri terminale, toate situate pe același nivel, are în total $n = 2 \cdot k - 1$ noduri

5) - un arbore binar complet are un număr impar de noduri

Parcurgerea arborilor binari

- parcurgerea arborilor binari de poate face în lățime (BFS) sau în adâncime (DFS)

- parcurgerea în lățime presupune vizitarea nodurilor pe niveluri, de la primul la ultimul, iar pe fiecare nivel nodurile sunt vizitate de la stânga la dreapta

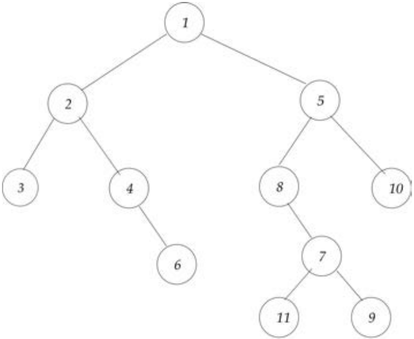
- parcurgerea în adâncime se poate face în trei moduri

  în preordine: se vizitează rădăcina, apoi subarborele stâng în preordine, apoi subarborele drept în preordine;

  în inordine: se vizitează subarborele stâng în inordine, apoi rădăcina, apoi subarborele drept în inordine;

  în postordine: se vizitează subarborele stâng în postordine, apoi subarborele drept în postordine, apoi rădăcina.

Parcurgerea arborilor binari



- în lățime: 1 2 5 3 4 8 10 6 7 11 9
- preordine: 1 2 3 4 6 5 8 7 11 9 10
- inordine: 3 2 4 6 1 8 11 7 9 5 10
- postordine: 3 6 4 2 11 9 7 8 10 5 1

Reprezentarea arborilor binari

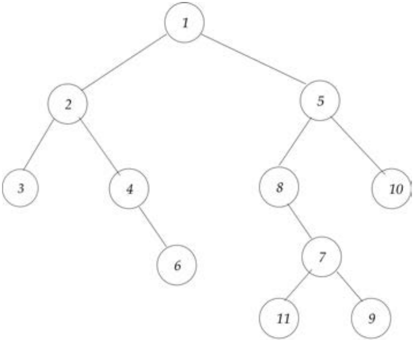
- reprezentarea prin vector de tați
- reprezentarea statică
- reprezentarea dinamică

a) **Reprezentarea standard**

- pentru fiecare nod în parte se precizează, dacă există, fiul stâng și fiul drept
- dacă un nod este terminal, atunci acest lucru se precizează punând 0 în locul fiilor săi
- se utilizează fie doi vectori numiți, de exemplu, *left*-pentru fiii din stânga și *right*-pentru fiii din dreapta
- pentru fiecare nod $x \in X$ componenta $left[x]$ conține fiul stâng al nodului x, iar componenta $right[x]$ conține fiul drept al nodului x
- rădăcina este acel nod care nu apare ca fiu al niciunui alt nod

Reprezentarea statică

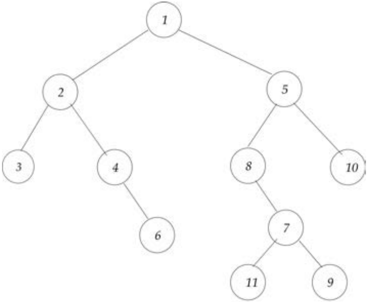
a)



i	1	2	3	4	5	6	7	8	9	10	11
left	2	3	0	0	8	0	11	0	0	0	0
right	5	4	0	6	10	0	9	7	0	0	0

Reprezentarea statică

a') pentru fiecare nod se precizează și părintele, nu doar fii



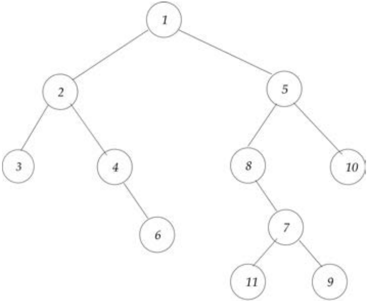
i	1	2	3	4	5	6	7	8	9	10	11
left	2	3	0	0	8	0	11	0	0	0	0
right	5	4	0	6	10	0	9	7	0	0	0
parent	0	1	2	2	1	4	8	5	7	5	7

b) **Legături de tip tată**

- se folosesc doi vectori: *parent* și *desc*
- pentru fiecare nod $i \in X$, $parent[i] =$ nodul părinte al nodului i
- $desc[i]$ poate lua două valori: -1 dacă i este fiu stâng pentru $parent[i]$, respectiv 1 dacă este fiu drept pentru acesta
- pentru nodul rădăcină r, care nu are un nod părinte asociat, valoarea corespunzătoare este $parent[r] = desc[r] = 0$

Reprezentarea statică

b) **Legături de tip tată**



i	1	2	3	4	5	6	7	8	9	10	11
parent	0	1	2	2	1	4	8	5	7	5	7
desc	0	-1	-1	1	1	1	1	-1	1	1	-1

Reprezentarea statică

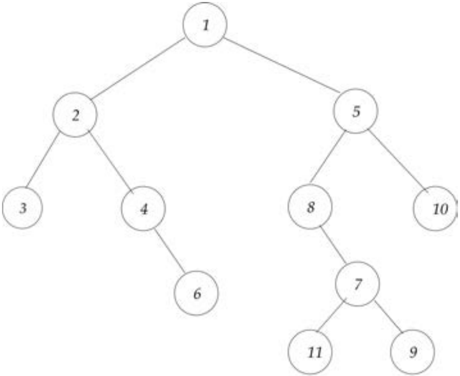
c) Reprezentarea cu paranteze

se scrie nodul rădăcină

fiecare nod al arborelui va fi urmat de:

- paranteză rotundă deschisă
- fiu stâng
- virgulă
- fiu drept
- paranteză rotundă închisă

c) Reprezentarea cu paranteze



1 (2 (3 , 4 (, 6)) , 5 (8 (, 7 (11 , 9)) , 10))

Reprezentarea dinamică

- fiecare nod este o structură cu trei câmpuri: informația atașată nodului, adresa fiului stâng și adresa fiului drept

- absența unui fiu este marcată cu pointerul nul - similar cu reprezentarea standard de la alocare statică

```
struct nod{
        int inf;          //sau orice alt tip pentru informație
        nod *left, *right;        //adresele fiilor
    } T;
```

- crearea unui arbore binar alocat dinamic:

  se generează un nod- se alocă spațiu în heap și se încarcă informația

  pentru fiecare nod se construiește subarborele stâng, subarborele drept și se completează adresele descendenților nodului cu adresele acestor subarbori

  un descendent vid se marchează printr-o proprietate stabilită asupra informației (de exemplu valoarea 0 ca informație)

Organizare, reprezentare:

- un arbore binar de căutare este organizat sub formă de arbore binar

- se poate reprezenta printr-o structură de date înlănțuită, în care fiecare nod este un obiect

- fiecare obiect nod conține câmpurile *left*, *right* și eventual *p* care punctează spre nodurile corespunzătoare fiului stâng, fiului drept, respectiv părintelui nodului și câmpul *key* ce conține informația nodului

- informația conținută de noduri este unică- două noduri nu pot conține aceeași informație

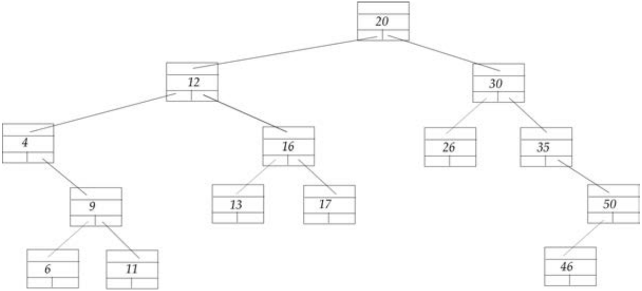
- dacă un fiu sau un părinte lipsește, câmpul corespunzător acestuia va conține valoarea nil

- nodul rădăcină este singurul nod din arbore care are valoarea nil pentru câmpul părinte *p*

Arbori de căutare

Definiție

Un arbore de căutare este un arbore binar $T = (X, U)$ în care nodurile sunt astfel memorate astfel încât: $\forall\ y \in X$ nod din subarborele stâng al nodului $x \in X$, $key[y] < key[x]$ și $\forall\ y \in X$ nod din subarborele drept al nodului $x \in X$, $key[y] > key[x]$.



Interogări

- căutarea unui nod care memorează o cheie dată - *SearchKey*
- determinarea cheii minime - *MINKey*
- determinarea cheii maxime - *MAXKey*
- determinarea valorii ce succede/precede o cheie - *Successor*/*Predecessor*

Căutarea nodului cu o cheie dată SearchKey

- fiind dat un arbore de căutare printr-un pointer x la rădăcina sa și o valoare k a cheii, se determină un pointer x la nodul având cheia k dacă există un asemenea nod în arbore, respectiv *nil* dacă nu există nod x cu $key[x] = k$

Căutare recursivă în arbore de căutare

$SearchKey(x, k)$

$\quad if((x = nil)\ or\ (k = key[x]))$

$\qquad\qquad return\ x;$

$\quad if(k < key[x])$

$\qquad\qquad return\ SearchKey(left[x], k);$

$\qquad else$

$\qquad\qquad return\ SearchKey(right[x], k);$

Operații asupra arborilor de căutare

- aceeași procedură se poate scrie iterativ- ciclu câttimp

Căutare iterativă în arbore de căutare

$SearchKey1(x, k)$

$\quad while((x \neq nil)\ and\ (k \neq key[x]))$

$\quad\quad if(k < key[x])$

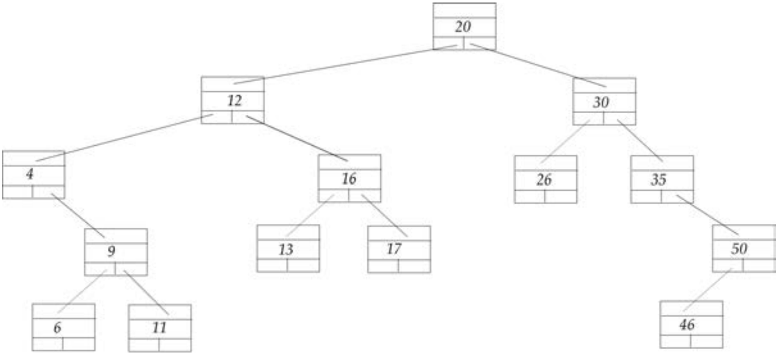
$\quad\quad\quad\quad x = left[x];$

$\quad\quad\quad else$

$\quad\quad\quad\quad x = right[x];$

$\quad return\ x;$

Operații asupra arborilor de căutare



- se caută nod cu cheia 13: $20 \stackrel{left}{\rightarrow} 12 \stackrel{right}{\rightarrow} 16 \stackrel{left}{\rightarrow} 13$
- se caută nod cu cheia 14: $20 \stackrel{left}{\rightarrow} 12 \stackrel{right}{\rightarrow} 16 \stackrel{left}{\rightarrow} 13 \stackrel{right}{\rightarrow} nil \Rightarrow$ nu există în arbore nod cu cheia 14.

Determinarea nodului cu valoarea cheii minimă

- nodul cu valoarea minimă a cheii este primul nod în parcurgerea în inordine

- se obține pornind de la rădăcină și traversând arborele pe subarborele stâng la fiecare pas

- fiind dat un arbore de căutare printr-un pointer x la rădăcina sa, se determină un pointer x la nodul având cheia k cu proprietatea că $k = min\{key[nod]|\ nod \in X\}$

Operații asupra arborilor de căutare

$MINKey(x)$ //recursiv

$if(left[x] = nil)$

$return\ x;$

$return\ MINKey(left[x]);$

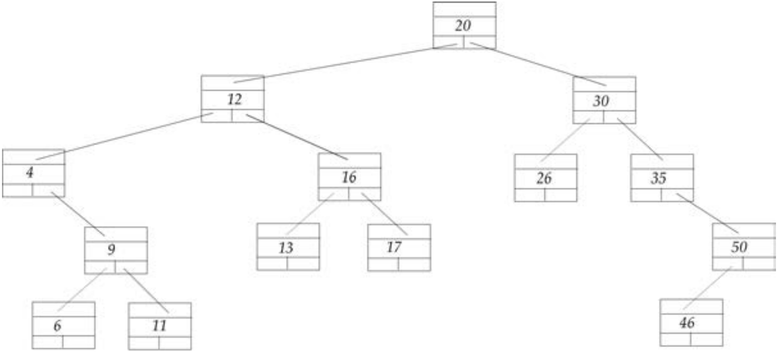
$MINKey1(x)$ //iterativ

$while(left[x] \neq nil)$

$x = left[x];$

$return\ x;$

Operații asupra arborilor de căutare



- valoarea minimă a cheii este 4 și se obține parcurgând drumul $20 \stackrel{left}{\rightarrow} 12 \stackrel{left}{\rightarrow} 4$.

Determinarea nodului cu valoarea cheii maximă

- nodul cu valoarea maximă a cheii este ultimul nod în parcurgerea în inordine
- se obține pornind de la rădăcină și traversând arborele pe subarborele drept la fiecare pas.

$MAXKey(x)$ // recursiv

$\quad if(right[x] = nil)$

$\qquad\qquad return\ x;$

$\quad return\ MAXKey(right[x]);$

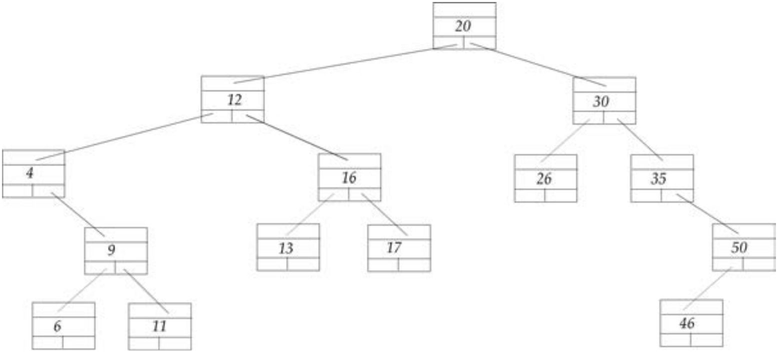
$MAXKey1(x)$ // iterativ

$\quad while(right[x] \neq nil)$

$\qquad\qquad\qquad x = right[x];$

$\quad return\ x;$

Operații asupra arborilor de căutare



$$20 \overset{right}{\rightarrow} 30 \overset{right}{\rightarrow} 35 \overset{right}{\rightarrow} 50$$

- ambele proceduri traseaza drumuri în jos în arbore - se execută într-un timp $O(h)$, unde h înălțimea arborelui

Succesorul și predecesorul unui nod

- succesorul unui nod x este nodul având cea mai mică cheie mai mare decât $key[x]$

- structura de arbore binar de căutare permite determinarea succesorului unui nod chiar și fără compararea cheilor

$Successor(x)$

$if(right[x] \neq nil)$

$return\ MINKey(right[x])$;

$succx = p[x]$;

$while\ (succx \neq nil\ and\ x = right[succx])$

$x = succx$;

$succx = p[succx]$;

$return\ succx$

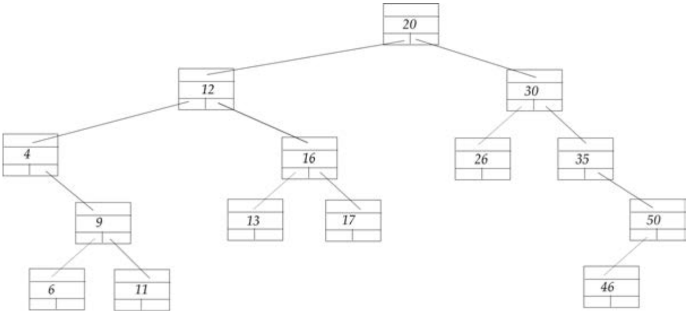
- se tratează două alternative:

dacă subarborele drept al nodului x nu este vid, atunci succesorul lui x este cel mai din stânga nod din acest subarbore drept, care este determinat în linia 2 prin apelul $MINKey(right[x])$

dacă subarborele drept al nodului x este vid, atunci succesorul său *succx* este cel mai de jos strămoș al lui x al cărui fiu din stânga este de asemenea strămoș al lui x

- pentru a determina *succx*, se traversează arborele de la x în sus până când se întâlnește un nod care este fiul stâng al părintelui său; acest lucru se realizează în liniile 3–7 ale codului procedurii *Successor*.

Operații asupra arborilor de căutare



- succesorul nodului cu cheia 12 este nodul cu cheia 13- cheia minimă din subarborele drept al nodului cu cheia 12
- succesorul nodului cu cheia 11 este nodul cu cheia 12; pentru a determina *succx*, se traversează arborele de la x în sus până când se întâlnește un nod care este fiul stâng al părintelui său

$Predecessor(x)$

$if(left[x] \neq nil)$

$return\ MAXKey(left[x])$;

$predx = p[x]$;

$while\ (predx \neq nil\ \ and\ \ x = left[predx])$

$x = predx$;

$predx = p[predx]$;

$return\ predx$;

- pentru arbore de înălțime h, timpul de execuție al procedurilor *Successor*, *Predecessor* este $O(h)$ - se urmează fie un drum în josul arborelui, fie unul în susul arborelui

Operații asupra arborilor de căutare

Inserarea

- fie dat un arbore de căutare prin rădăcina lui, r

- se va realiza inserarea în arbore a unui nod v cu o cheie data k

- se presupune că nu există deja în arbore un nod cu cheia k, altfel se poate face o căutare înainte de inserare

- se pornește de la rădăcină

> dacă cheia k este mai mare decât a nodului curent x, se inserează cheia în subarborele drept dacă acesta există sau nodul cu cheia k devine fiul drept al nodului curent dacă acesta nu are fiu drept
>
> dacă cheia k este mai mică decât a nodului curent x, se inserează nodul cu cheia k cheia în subarborele stâng dacă acesta există sau nodul cu cheia k devine fiul stâng al nodului curent dacă acesta nu are fiu stâng
>
> nodul inserat va fi întotdeauna nod frunză în arbore

Operații asupra arborilor de căutare

Inserarea unui nod nou v cu cheie dată k într-un arbore cu rădăcina r

$TInsert(r, k)$

$p[v] = nil;$

$if(r = nil)\ \ r = v;$

else

$x = r;$

$while(x \neq nil)$

$y = x;$

$if(k < key[x])\ \ x = left[x];$

$else\ \ x = right[x];$

$p[v] = y;;$

$return\ v;$

Inserare recursivă a nodului v cu cheia k într-un arbore cu rădăcina r

$TInsertRec(r, k)$

$\quad p[v] = r;$

$\quad if(r = nil)$

$\qquad\qquad return\ v;$

$\quad\quad$ else

$\qquad\qquad if(k < key[r])$

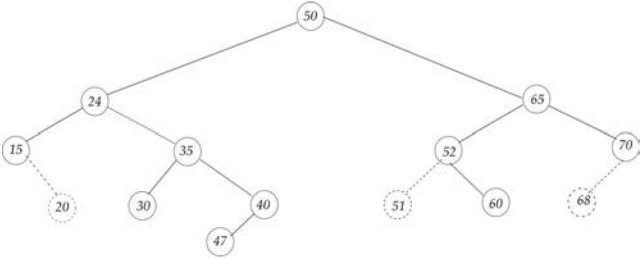
$\qquad\qquad\qquad return\ TInsertRec(left[x], k);$

$\qquad\qquad else$

$\qquad\qquad\qquad return\ TInsertRec(right[x], k);$

- procedura $TInsert$ se execută în timp $O(h)$ pe un arbore de înălțime h

Operații asupra arborilor de căutare



- inserate nodurile cu cheile 20, 51, 68

Ștergerea

- fiind dat un arbore de căutare prin rădăcina lui, r să se șteargă din arbore nodul v

- daca v nu are fii, se va modifica părintele său $p[v]$ pentru a-i înlocui fiul v cu nil

- dacă nodul v are un singur fiu, v va fi eliminat din arbore prin inserarea unei legături de la părintele lui v, $p[v]$, la fiul lui v

- dacă nodul v are doi fii, se va elimina din arbore succesorul y al lui v, care nu are fiu stâng și apoi se vor înlocui cheia și datele adiționale ale lui v cu cheia și datele adiționale ale lui y

Operații asupra arborilor de căutare

Algoritm pentru ștergerea unui nod dintr-un arbore de căutare

$TDelete(r, v)$
$\quad if\,(left[v] = nil \;\; or \;\; right[v] = nil) \;\; y = v;$
$\qquad\qquad else \;\; y = Successor(v);$
$\quad if\,(left[y] \neq nil) \;\; x = left[y];$
$\qquad\qquad else \;\; x = right[y];$
$\quad if\,(x \neq nil) \;\; p[x] = p[y];$
$\quad if\,(p[y] = nil) \;\; r = x;$
$\qquad\qquad else$
$\qquad\qquad\quad if\,(y = left[p[y]]) \;\; left[p[y]] = x;$
$\qquad\qquad\qquad else \;\; right[p[y]] = x;$
$\quad if\,(y \neq v)$
$\qquad\qquad key[v] = key[y]; \;\; p[v] = p[y];$
$\qquad\qquad left[v] = left[y]; \;\; right[v] = right[y];$
$\quad return \; y;$

Operații asupra arborilor de căutare