

7. gyakorlat

Téma:

Bináris fák láncolt ábrázolása. A három nevezetes rekurzív bejáró algoritmus bináris fákra. Kifejezés fa. Egyszerűbb algoritmusok bináris fákra.

Feladatok a gyakorlatra:

Bináris fa láncolt ábrázolása:

Két pointeres csúcs. Node

| Node |
|---------------------------------------------------------------------------------------------------------|
| + <i>key</i> : \mathcal{T} // \mathcal{T} valamilyen ismert típus |
| + <i>left, right</i> : Node* |
| + Node() { <i>left</i> = <i>right</i> = \emptyset } // egycsúcsú fát képez belőle |
| + Node(<i>x</i> : \mathcal{T}) { <i>left</i> = <i>right</i> = \emptyset ; <i>key</i> = <i>x</i> } |

Három pointeres csúcs: Node3

| Node3 |
|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| + <i>key</i> : \mathcal{T} // \mathcal{T} valamilyen ismert típus |
| + <i>left, right, parent</i> : Node3* |
| + Node3(<i>p</i> :Node3*) { <i>left</i> = <i>right</i> = \emptyset ; <i>parent</i> = <i>p</i> } |
| + Node3(<i>x</i> : \mathcal{T} , <i>p</i> :Node3*) { <i>left</i> = <i>right</i> = \emptyset ; <i>parent</i> = <i>p</i> ; <i>key</i> = <i>x</i> } |

Ismétlésképpen írjuk fel a három rekurzív algoritmust, ahogyan a jegyzetben szerepel:

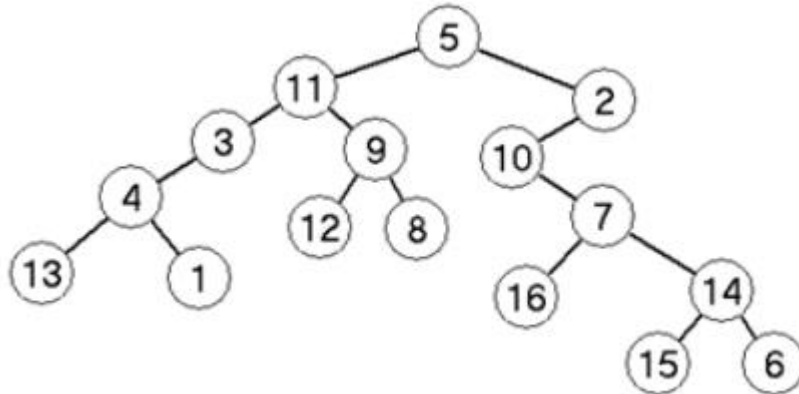
BinTree – absztrakt bináris fa típus, $t \rightarrow \text{left}$ és $t \rightarrow \text{right}$ helyett szokás $\text{left}(t)$ és $\text{right}(t)$ jelölést is használni. Az üres fát szokták Ω -val jelölni, így $t = 0$ helyett $t = \Omega$ is használható. Láncolt ábrázolású bináris fák esetén a bejáró algoritmusok paramétere lehetne $t:\text{Node}^*$, vagy $t:\text{Node3}^*$.

| preorder(<i>t</i> : BinTree) | inorder(<i>t</i> : BinTree) | postorder(<i>t</i> : BinTree) |
|-------------------------------------|------------------------------------|--------------------------------------|
| $t \neq \emptyset$ | $t \neq \emptyset$ | $t \neq \emptyset$ |
| process(<i>t</i>) | inorder(<i>t</i> → <i>left</i>) | postorder(<i>t</i> → <i>left</i>) |
| preorder(<i>t</i> → <i>left</i>) | process(<i>t</i>) | postorder(<i>t</i> → <i>right</i>) |
| preorder(<i>t</i> → <i>right</i>) | inorder(<i>t</i> → <i>right</i>) | process(<i>t</i>) |
| SKIP | SKIP | SKIP |

Megjegyzés: A jegyzetben megtalálható a preorder és az inorder bejárás egy másik alakja: az utolsó rekurzív hívást ciklus helyettesíti – így hatékonyabb.

1. feladat

Egy konkrét bináris fa bejárása a három tanult rekurzív algoritmussal, ha a feldolgozás a kulcs kiírása, milyen sorrendben írná ki a kulcsokat?



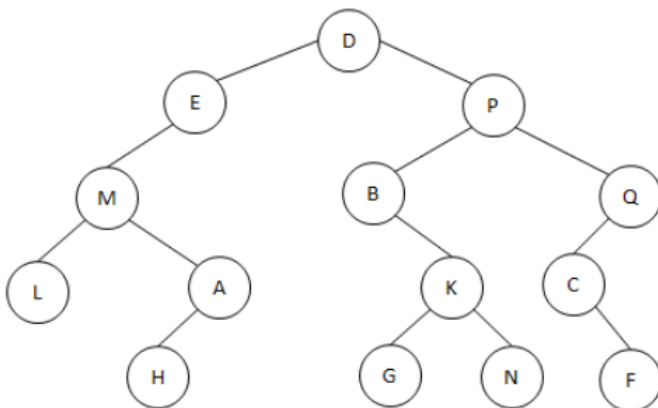
2. feladat

Egy nem teljes bináris fa preorder + inorder, vagy postorder + inorder bejárásából rekonstruáljuk, hogyan nézett ki a fa. Miért nem lehet rekonstruálni a preorder+postorder bejárásból?

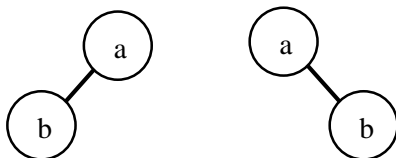
PREORDER: D E M L A H P B K G N Q C F

INORDER: L M H A E D B G K N P C F Q

Megoldás:



Miért nem jó a preorder és postorder? Ha egy-gyerekes a csúcs, nem lehet tudni belőle, hogy az egy-gyerek melyik irányban van:



PRE: a b

POST: b a

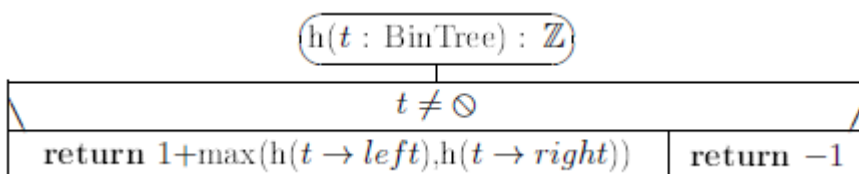
a b

b a

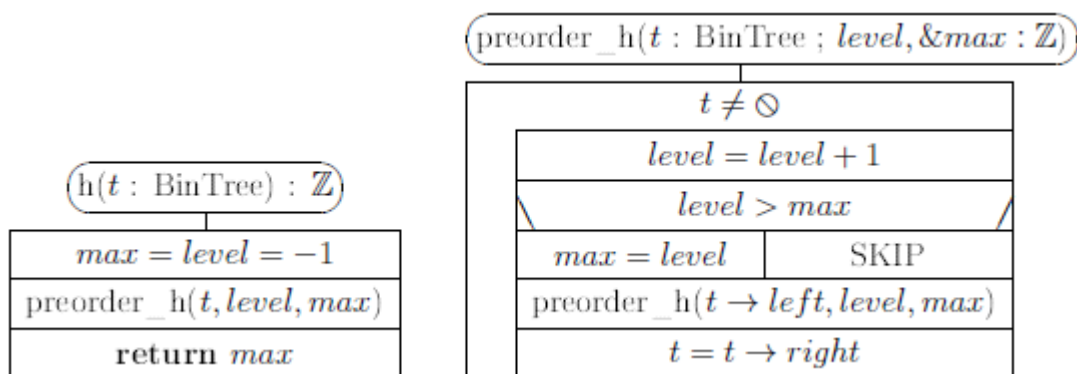
Rekurzív algoritmusok készítése bináris fákkal kapcsolatos feladatokhoz:

Előadáson szerepel a magasság függvény kétféle módon: (a) a függvény visszatérési értéke a magasság, postroder bejárásra támaszkodva, (b) preorder bejárás+segédváltozó, ebben az esetben egy nem rekurzív algoritmus kerül a rekurzív fölé, ami gondoskodik a belső rekurzív algoritmus megfelelően paraméterezett meghívásáról és az abból kapott eredmény visszaadásáról. Így néznek ki (ezt nem kell felírni az órán, de ezekre támaszkodhatunk a feladatok megoldásánál):

(a)



(b)



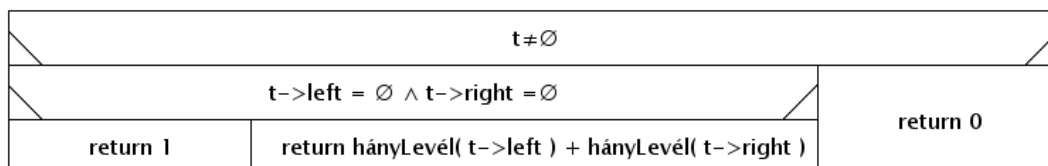
Megjegyzés: az itt közölt megoldásokban előnyben részesítjük az (a) szerinti megoldásokat, de ZH-n, vizsgán (b) módszerrel készített megoldás is elfogadható. Érdeklőség: a (b) alakú algoritmusokat össze lehet vetni a programozásból tanult felsorolós tételekkel: a kiválasztott bejáró algoritmust a fa csúcsainak felsorolására használjuk, a referencia szerint vitt paraméter pedig a progátétel eredmény változója. (Épp most tanulják programozásból a felsorolós programozási tételeket.)

3. feladat

Hány levele van a fának?

Megoldás:

hányLevél(t:BinTree):N



4. feladat

Mi a legnagyobb kulcsa egy bináris fának? A fa esetleg lehet üres, azaz $t=0$ eset is előfordulhat. Oldjuk meg (a) és (b) módszerrel is! Itt most (a) esetben is kell egy „indító” algoritmus, mert le kell ellenőrizni, hogy a fa nem üres-e? Ne használjuk a maximum kezdőértékének $-\infty$ értéket! Ha nem üres a fa, induljunk a gyökér kulcsával, ha üres a fa, jelezzünk hibát!

Megoldás:

(a) *Nem használ segéd paramétert. Üres fát le kell ellenőrizni!*

maxKulcs(t:BinTree):T

| t = 0 | |
|----------------|-------------------------|
| EmptyTreeError | return(maxBejár(t)) |

maxBejár(t:BinTree):T

| t->left = 0 \wedge t->right = 0 | | | |
|-----------------------------------|---------------------------------------------|----------------------------------------|----------------------------------------|
| return t->key | t->left \neq 0 \wedge t->right \neq 0 | t->left \neq 0 \wedge t->right = 0 | t->left = 0 \wedge t->right \neq 0 |
| | max1 := maxBejár(t-> left) | max1:=maxBejár(t-> left) | max1:=maxBejár(t-> right) |
| | max2 :=maxBejár(t-> right) | | |
| | max1 := max(max1, max2) | | |
| | return max(max1, t->key) | | |

Megjegyzések: maxBejár algoritmus előfeltétele, hogy t nem lehet üres! Ezért a rekurzív hívásokat szét kell bontani a t fa alakja szerint:

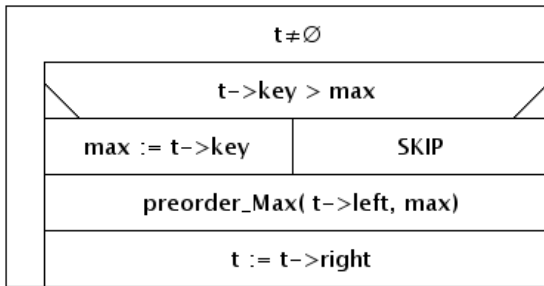
- t lehet levél, vagy belső pont
- belső pont esetei: két gyerekes, csak bal-, illetve csak jobb gyerekekkel rendelkező.

(b) A másik megoldási lehetőséget választva a fő eljárás ellenőrzi az előfeltételt, és ha a fa nem üres, akkor gondoskodik a max változó kezdőértékéről, majd elindít egy bejáró algoritmust. A bejárás végeztével max változó a legnagyobb kulcs értékét fogja tartalmazni, így azzal visszatér.

maxKulcs(t:BinTree) : T

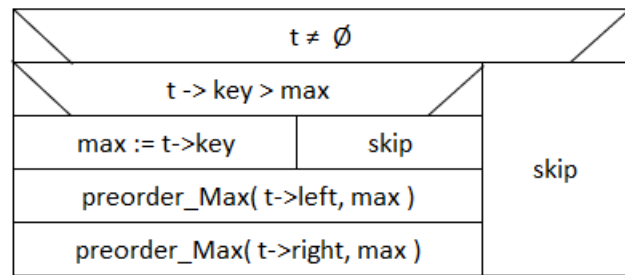
| t \neq \emptyset | |
|------------------------|----------------|
| max := t->key | EmptyTreeError |
| preorder_Max(t, max) | |
| return max | |

preorder_Max(t: BinTree, &max : T)



Végrekurzió helyett ciklus (jegyzet ajánlása)

preorder_Max(t: BinTree, &max : T)



„Klasszikus preorder bejárással”

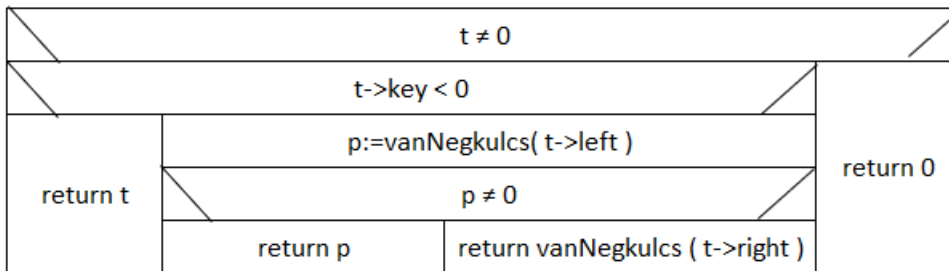
A bejáró algoritmus viszi magával a max változót, ha kell, módosítja. FONTOS, hogy max cím szerint átadott paraméter legyen!

5. feladat

Keressünk egy negatív kulcsot a fában. (A fa kulcsai egész számok.) Ha találtunk, adjuk vissza a címét, ha nem találtunk, adjunk 0 értéket vissza. Ügyeljünk a hatékonyságra, ha sikeres a keresés, minél hamarabb térjen vissza a rekurzió!

Megoldás:

vanNegKulcs(t.BinTree): Node*



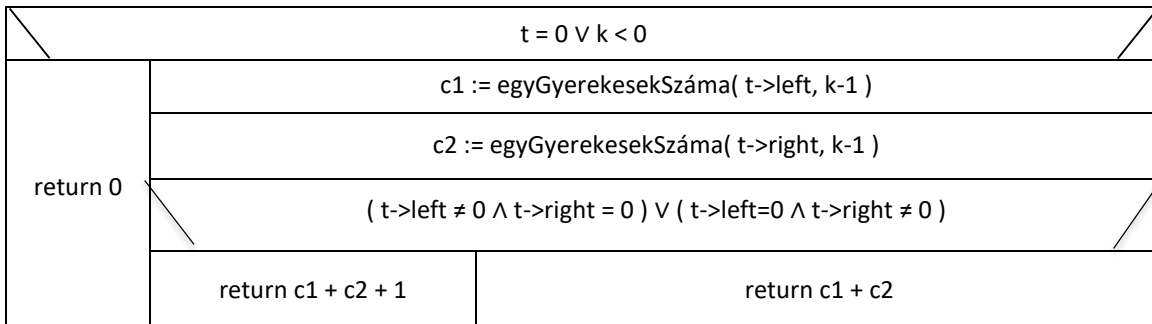
6. feladat

Bevezetjük a szint fogalmát. 0. szint a gyökér, majd 1. szint, 2. szint stb. A feldolgozás csak egy adott szintig történik, vagy adott szinttől a teljes mélységig. Feladatok: (a) hány egy-gyerekes csúcsa van a fának a 0..k szinteken, vagy (b) a k-nál nagyobb szinteken?

Megoldás:

(a)

egyGyerekesekSzama(t:BinTree, k: Z):N



(b)

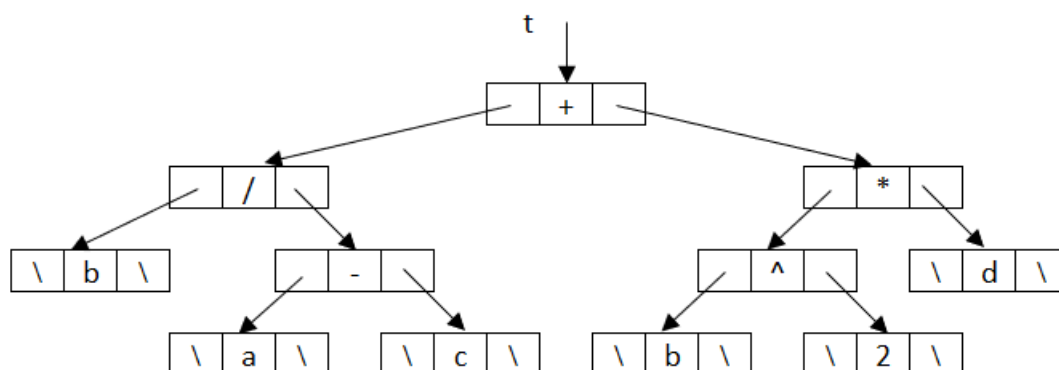
egyGyerekesekSzáma(t:BinTree, k: Z):N

| t = 0 | |
|----------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| return 0 | c1:= egyGyerekesekSzáma(t->left, k-1) |
| | c2 := egyGyerekesekSzáma(t->right, k-1) |
| | $k < 0 \wedge ((t \rightarrow \text{left} \neq 0 \wedge t \rightarrow \text{right} = 0) \vee (t \rightarrow \text{left} = 0 \wedge t \rightarrow \text{right} \neq 0))$ |
| | <div>return c1 + c2 + 1</div> <div>return c1 + c2</div> |

7. feladat

Kifejezés fa. Az egy és két operandusú műveletekből álló kifejezést ábrázolhatjuk egy bináris fával. Például egy aritmetikai kifejezés fája:

A $b/(a-c)+b^2*d$ kifejezést ábrázoló kifejezés fa.



A kifejezés fa postorder bejárása épp a kifejezés lengyel formáját adja:

$b \ a \ c \ - \ / \ b \ 2 \ ^ \ d \ * \ +$

- Találják ki, hogy melyik bejárás adja a lengyel formát? (Mit ad a másik kettő?)
- Hogyan rajzolhatjuk fel a kifejezés fát egy tetszőleges kifejezés esetén? (A legutoljára elvégzendő művelet lesz a fa gyökere, majd ezt rekurzívan alkalmazzuk a bal és jobb oldalán álló kifejezésre.)
- Ha egy operandusú műveleteket is megengedünk (például: -x, vagy ++x), akkor ezeket hogyan ábrázolná a fa? Bal, vagy jobb gyerek lesz az operandus? (Preorder és postorder szerint bármelyik lehetne, de az inorder akkor ad helyes kifejezést, ha az operátornak jobb gyereke az operandus.)

Feladat: adott egy kifejezés lengyel formája, építsük fel a kifejezés fát. Ez egy iteratív algoritmus, vermet használ, a lengyel forma kiértékelés képezheti az alapját a megoldásnak.

X bemenet legyen egy kifejezés lengyel formája, elemei operandusok és operátorok (típust most nem jelölünk). Hogy egyszerűbb legyen az algoritmus, feltesszük, hogy minden operátor két operandussal rendelkezik.

Természetesen könnyen átalakítható, hogy egy operandusú műveleteket is fel tudjon dolgozni.

Megoldás:

kifFaÉpít(X): Node*

| | |
|--------------------------|--------------|
| V: Stack(); x := read(X) | |
| x ≠ ε | |
| t := new Node(x) | |
| Operator(x) | Operandus(x) |
| t->right := V.pop() | V.push(t) |
| t->left := V.pop() | |
| V.push(t) | |
| x := read(X) | |
| return V.pop() | |

Házi feladatok:

1 feladat

Készítsünk rekurzív algoritmust, mely egy láncoltan ábrázolt bináris fa leveleiben található kulcsok minimumát adja vissza. Üres fára jelezzen hibát.

Ez egy „feltételes minimum keresés” feladat. Megoldhatnánk az ismert programozási tétel segítségével: *minBejar(t:BinTree, &l:B, &min:T)* *minLevelKulcs* az *l:=hamis* kezdőértékkel elindítja a bejáró algoritmust, a *min* változó majd az első levélnél kap kezdőértéket. Amikor a bejárás az első levélhez ér, a logikai változó igazra áll, és innentől kezdve a leveleknél folyik a minimum kiválasztás. Egy elegánsabb, segéd paraméterek nélküli megoldás készíthető a maximális kulcsot meghatározó algoritmus ötletének felhasználásával:

minLevelKulcs(t:BinTree): T

| | |
|----------------|---------------------|
| t = 0 | |
| EmptyTreeError | return(minBejar(t)) |

minBejar(t:BinTree): T

| | | | |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| t->left = 0 és t->right = 0 | | | |
| return t->key | t->left ≠ 0 és t->right ≠ 0 | t->left ≠ 0 és t->right = 0 | t->left = 0 és t->right ≠ 0 |
| | min1:=minBejár(t->left) | return minBejár(t->left) | return minBejár(t->right) |
| | min2:=minBejár(t->right) | | |
| | return min(min1,min2) | | |

2. feladat

Készítsünk rekurzív algoritmust, mely egy láncoltan ábrázolt bináris fáról megadja, hogy melyik az a legkisebb szint, ahol van levele a fának. Üres fára -1 –et adjon.

minLevélSzint(t:BinTree):Z

| | | | | | |
|-----------|-----------------------------------|------------------------------|--------------|-----------------------|--|
| t = 0 | | | | | |
| return -1 | t->left = 0 \wedge t->right = 0 | | | | |
| | return 0 | sz1:=minLevSzint(t->left) | | | |
| | | sz2:=minLevSzint(t->right) | | | |
| | | sz1 = -1 | | | |
| | | return sz2+1 | sz2 = -1 | | |
| | | | return sz1+1 | return min(sz1,sz2)+1 | |

üres fa esete

levél esete

belső pontok esete:

üres részfából feljövő -1

értékre a minimum miatt

külön figyelni kell!

Megjegyzések:

- Észrevehető, hogy a levél esetre nem lenne szükség, mert $sz1 = -1$ ág helyesen kezelné.
- Viszont logikailag segíti a megértést: ugyanis $t:BinTree$ paraméternek az esetei alak szerint:
 - üres fa, levél, belső pont.
- A rekurziót ezek mentén érdemes végig gondolni!

3. feladat

Adott egy láncoltan ábrázolt kifejezés fa. Készítsünk rekurzív algoritmust, mely kiírja a kifejezés teljesen zárójelezett alakját. Például a fenti fa esetén: $((b/(a-c))+((b^2)*d))$ lenne a kimenet.

kiírTeljZj(t:BinTree)

| | | |
|-------|-----------------------------------|------------------------|
| t = 0 | | |
| skip | t->left = 0 \wedge t->right = 0 | |
| | write(t->key) | write("(") |
| | | kiírTeljZj(t->left) |
| | | write(t->key) |
| | | kiírTeljZj(t->right) |
| | | write(")") |

A megoldásban arra kell ügyelni, hogy zárójeleket csak akkor írjunk, ha operátornál – azaz belső pontnál – járunk.

4. feladat

Adott két láncoltan ábrázolt bináris fa: t_1 és t_2 . Készítsünk rekurzív logikai függvényt, mely eldönti, hogy a két fa alakja megegyezik-e. A tárolt kulcsoknak nem kell egyenlőnek lenniük, csak az alakjuknak!

azonosAlaku(t_1 :BinTree, t_2 :BinTree):B

| | | |
|------------------|--------------------------------------------------------------------------------------|--------------|
| t1 = 0 és t2 = 0 | | |
| return true | t1 ≠ 0 és t2 ≠ 0 | |
| | return (azonosAlaku(t1->left, t2->left) ∧ azonosAlaku(t1->right, t2->right)) | return false |

Készült az „Integrált kutatói utánpótlás-képzési program az informatika és számítás-tudomány diszciplináris területein” című EFOP 3.6.3-VEKOP-16-2017-00002 azonosítójú projekt támogatásával.