7. gyakorlat

Téma:

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

Feladatok a gyakorlatra:

Bináris fa láncolt ábrázolása:

Két pointeres csúcs. Node

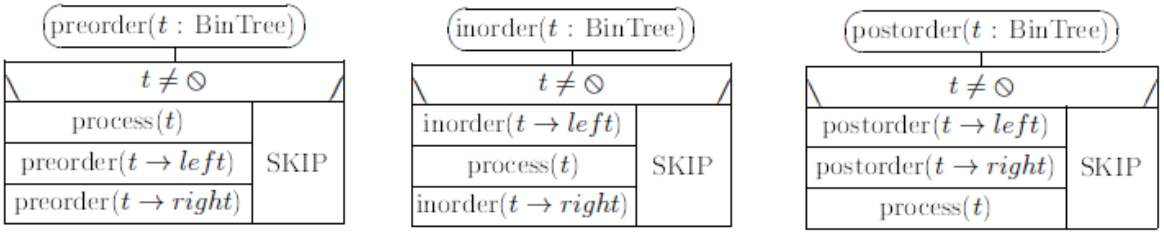
Node
+ $key : \mathcal{T}$ // $\mathcal{T}$ valamilyen ismert típus + $left, right$: Node*
+ Node() { $left = right = \oslash$ } // egycsúcsú fát képez belőle + Node($x : \mathcal{T}$) { $left = right = \oslash$; $key = x$ }

Három pointeres csúcs: Node3

Node3
+ $key : \mathcal{T}$ // $\mathcal{T}$ valamilyen ismert típus + $left, right, parent$: Node3*
+ Node3(p:Node3*) { $left = right = \oslash$; $parent = p$ } + Node3($x : \mathcal{T}$, p:Node3*) { $left = right = \oslash$; $parent = p$; $key = x$ }

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

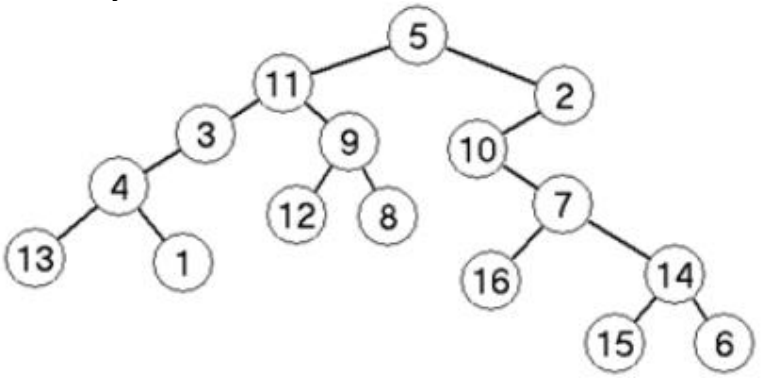
BinTree – absztrakt bináris fa típus, t→left és t→right helyett szokás left(t) és right(t) jelölést is használni. Az üres fát szokták Ω-val jelölni, így t = 0 helyett t = Ω is használható. Láncolt ábrázolású bináris fák esetén a bejáró algoritmusok paramétere lehetne t:Node*, vagy t:Node3*.



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

1. **feladat**

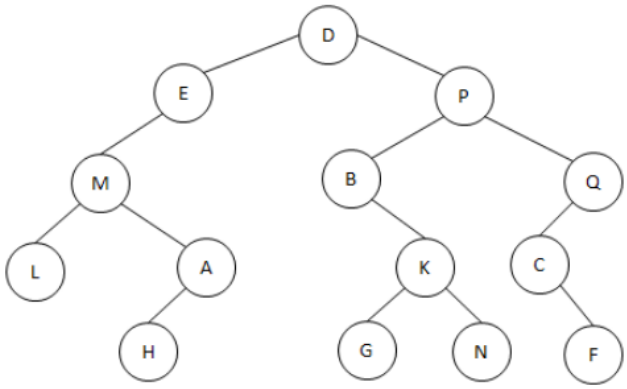
Egy konkrét bináris fa bejárása a három tanult rekurzív algoritmussal, ha a feldolgozás a kulcs kiírása, milyen sorrendben írná ki a kulcsokat?



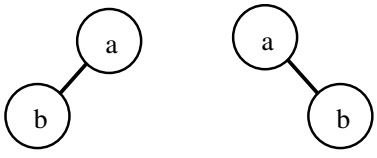
2. **feladat**

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:

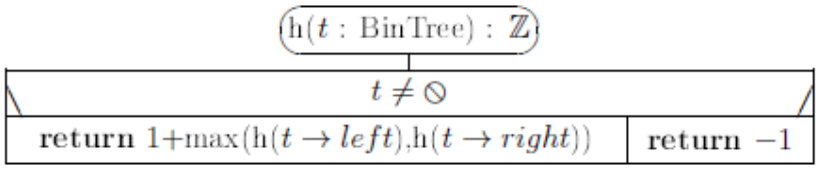


PRE: a b a b
POST: b a b a

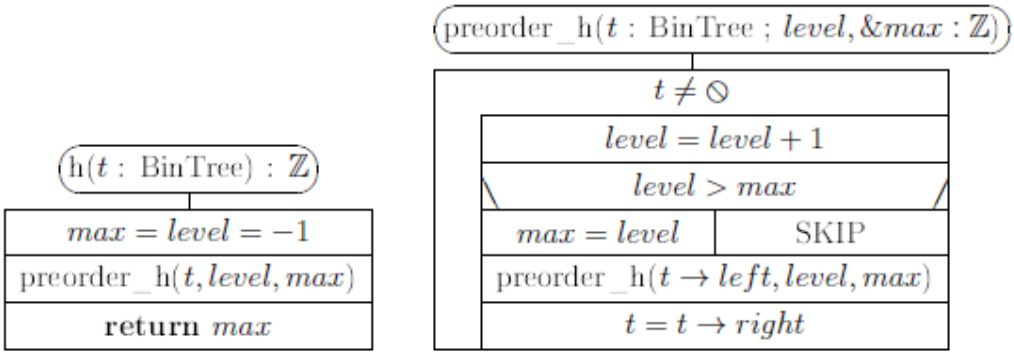
Rekurzív algoritmusok készítése bináris fákkal kapcsolatos feladatokhoz:

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 meghívásról és az abból kapott eredmény visszaadásáról. Így néznek ki (ezt nem kell felírni az órán, de ezekre támaszkodhatunk a feladatok megoldásánál):

(a)



(b)



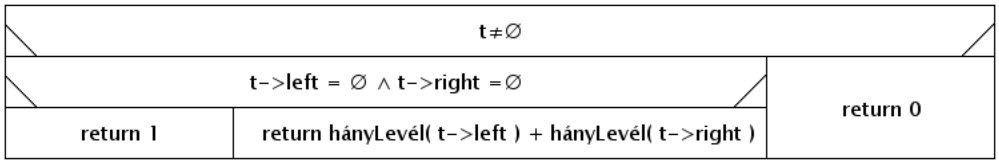
Megjegyzés: az itt közölt megoldásokban előnyben részesítjük az (a) szerinti megoldásokat, de ZH-n, vizsgán (b) is elfogadható. Érdekesség: a (b) alakú algoritmusokat össze lehet vetni a programozásból tanult felsorolós tételekkel: a kiválasztott bejáró algoritmust a fa csúcsainak felsorolására használjuk, a referencia szerint vitt paraméter pedig a progtétel 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 -∞ 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 ∧ t->right = 0			
	t->left ≠ 0 ∧ t->right ≠ 0	t->left ≠ 0 ∧ t->right = 0	t->left = 0 ∧ t->right ≠ 0
return t->key	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:

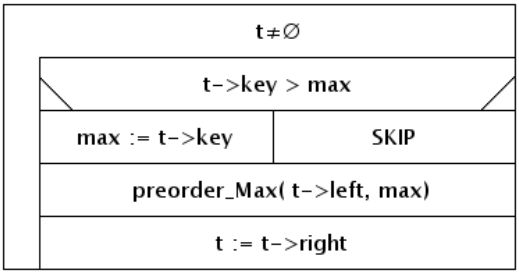
- *t lehet levél, vagy belső pont*
- *belső pont esetei: két gyerekes, csak bal-, illetve csak jobb gyerekkel rendelkező.*

(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

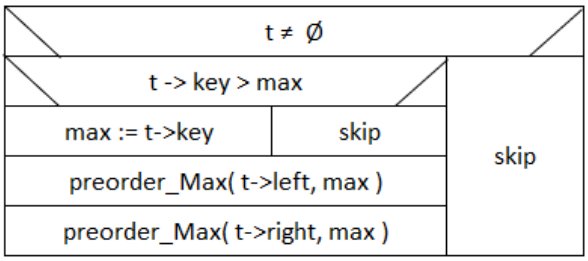
t≠∅	
max := t->key	EmptyTreeError
preorder_Max(t, max)	
return max	

preorder_Max(t: BinTree, &max : T)



Végrekurzió helyett ciklus (jegyzet ajánlása)

preorder_Max(t: BinTree, &max : T)



„Klasszikus preorder bejárással"

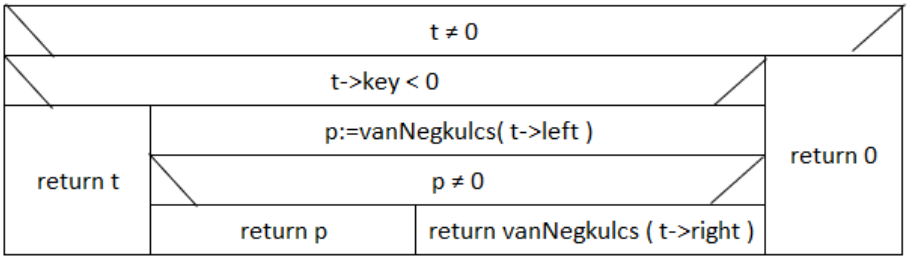
A bejáró algoritmus viszi magával a max változót, ha kell, módosítja. FONTOS, hogy max cím szerint átadott paraméter legyen!

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*



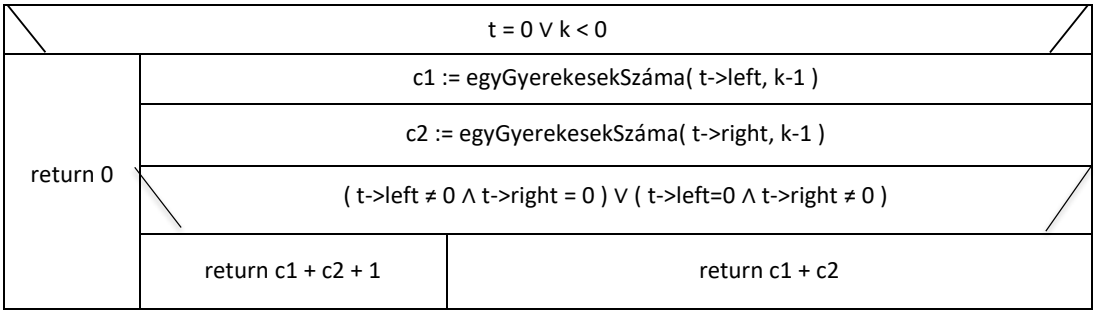
6. feladat

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:

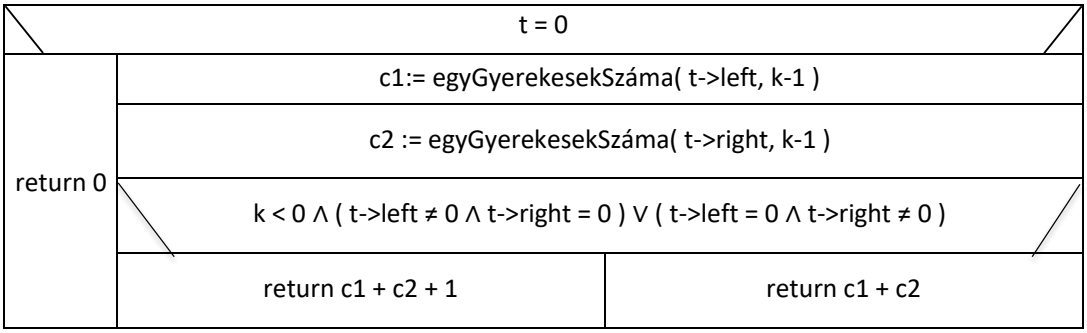
(a)

egyGyerekesekSzáma(t:BinTree, k: Z):N



(b)

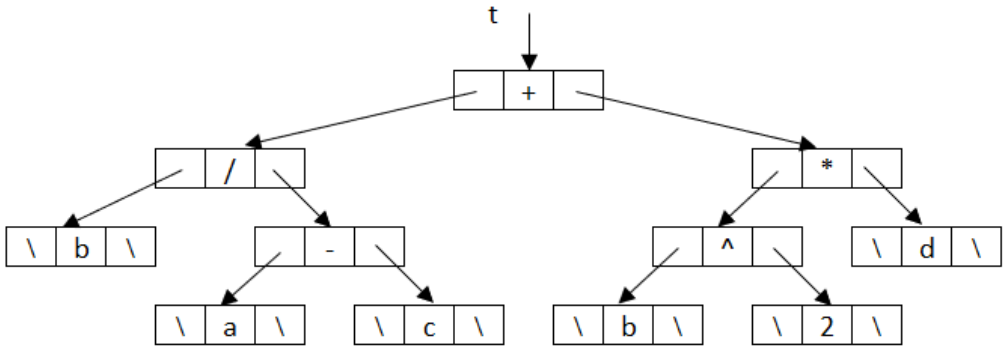
egyGyerekesekSzáma(t:BinTree, k: Z):N



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.)

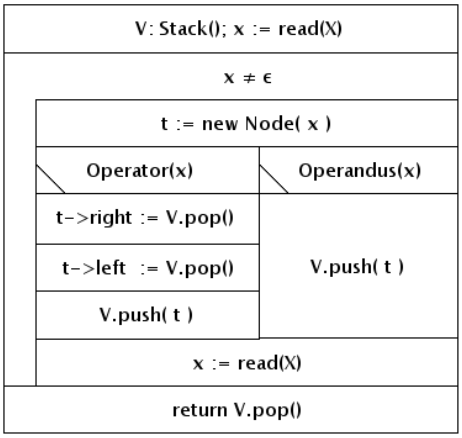
Feladat: adott egy kifejezés lengyel formája, építsük fel a kifejezés fát. Ez egy iteratív algoritmus, vermet használ, a lengyel forma kiértékelés képezheti az alapját a megoldásnak.

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*



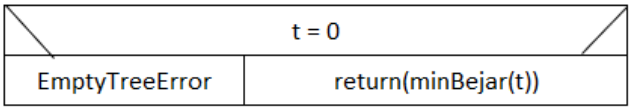
Házi feladatok:

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 paramérek nélküli megoldás készíthető a maximális kulcsot meghatározó algoritmus ötletének felhasználásával:

minLevelKulcs(t:BinTree): T



minBejar(t:BinTree): T