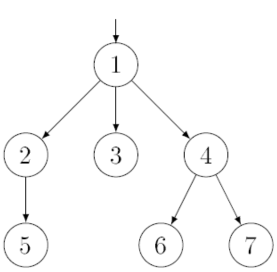
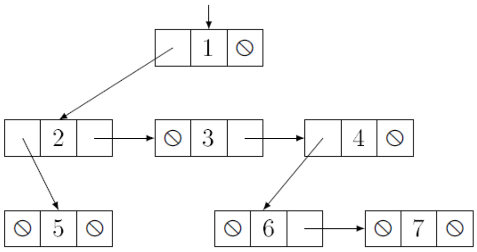
Általános fák gyakorlati anyag[1]

Jegyzetbeli anyag:





A fa jellemzői:

van egy kitüntetett csúcsa, a gyökér, a csúcsoknak tetszőleges sok leszármazottja lehet.

Ábrázolás:

két pointerrel, egyik az első leszármazottra mutat, a másik a testvérre. Esetleg kiegészíthetjük szülő pointerrel is: a testvérek mindegyike a szülőjére mutat vissza.

Másféle ábrázolás is létezhet, például a gráfok ábrázolásához használatos éllistás módszert is használhatjuk.

A fa egy csúcsának típusa:

Node
+ $child1, sibling$: Node* // $child1$: első gyerek; $sibling$: következő testvér + key : T // T ismert típus
+ Node() { $child1 := sibling := \emptyset$ } // egycsúcsú fát képez belőle + Node(x:T) { $child1 := sibling := \emptyset$; $key := x$ }

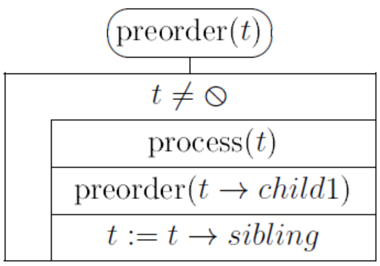
A fa zárójelezett alakja

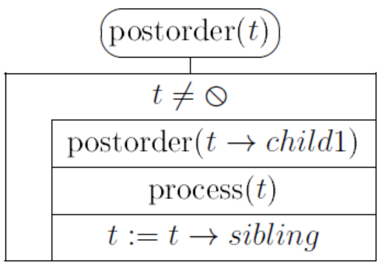
A fa szöveges leírása. egy nemüres fa általános alakja (G t_1 … t_n), ahol G a gyökércsúcs tartalma, t_1 … t_n pedig a részfák. Így pl. a fenti fa zárójelezett leírása a következő: { 1 [2 (5)] (3) [4 (6) (7)] }

Bejárások

Itt a tavalyi bejáró algoritmusok ciklusos alakját érdemes használni, hatékonyság miatt (testvéreken érdemesebb ciklussal végig iterálni).

A postorder látszólag a tavalyi „inorder”-re hajaz, hogy miért ezt definiáljuk postorderként, azt később egy példán bemutatom.





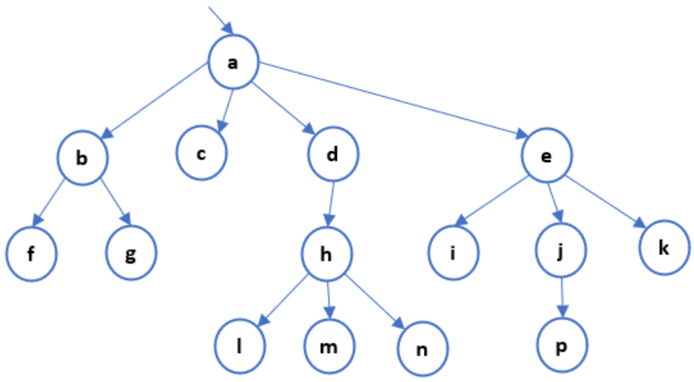
Példák:

Könyvtár rendszer, függvény kifejezések.

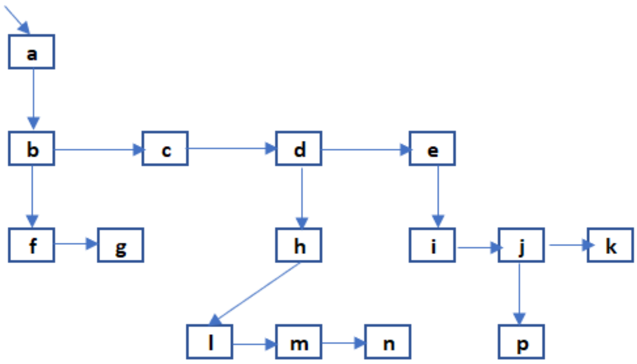
[1] Készítette: Veszprémi Anna, felhasználva Ásványi Tibor jegyzete

Feladatok:

Adott az alábbi általános fa:



Rajzoljuk le két pointeres ábrázolásban:

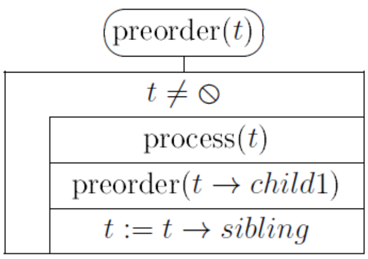


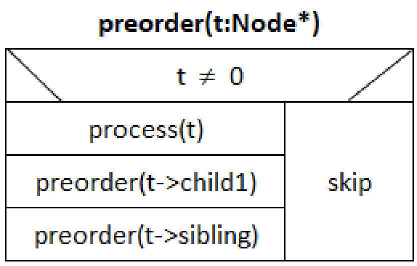
Adjuk meg zárójeles alakban, váltogassuk szisztematikusan a háromféle zárójel típust: { } [ ] ( ) :

{ a [b (f) (g)] [c] [d (h { l } { m } { n })] [e (i) (j { p }) (k)] }

Adjuk meg a preorder bejárás algoritmusát teljesen rekurzívan:

A gyakorlatban nem ezt használjuk, hatékonyság miatt, de itt jobban látszik a rokonság a bináris fák preorder bejáró algoritmusával.



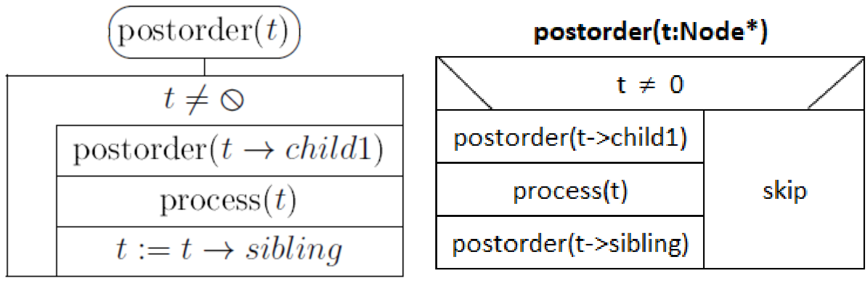


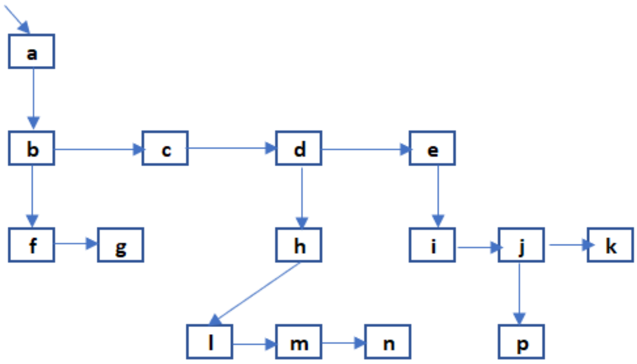
Járjuk be a fenti fát preorder bejárással, írjuk ki a kulcsokat:

a b f g c d h l m n e i j p k

Adjuk meg a postorder bejárás algoritmusát teljesen rekurzívan:

Azzal, hogy a feldolgozás a t->child1 után történik, ez alakját tekintve inkább a bináris fáknál definiált inorder bejárás rokona. Viszont a függvény kifejezések bejárásánál ezzel fogjuk a lengyel formát megkapni, ahogy azt kicsit alább egy példa szemlélteti.





Járjuk be a fenti fát postorder bejárással, írjuk ki a kulcsokat:

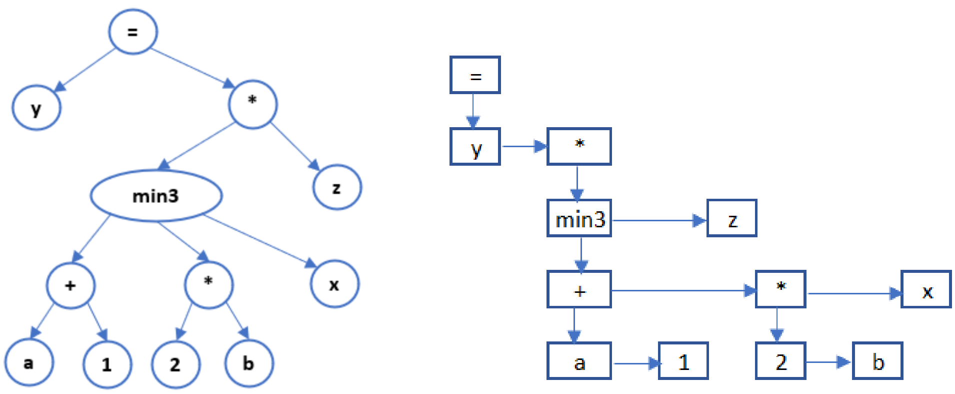
f g b c l m n h d i p j k e a

A következő példával szemléltethetjük, miért ezt hívjuk postorder bejárásnak?

Ez adja a ugyanis függvény kifejezések lengyel formáját.

Példa: legyen min3 egy három paraméteres függvény, y = min3(a+1, 2*b, x) * z egy függvény kifejezés.

A kifejezés általános fa alakban:



A kifejezés lengyel formáját a fent megadott postorder bejárással kapjuk meg.

Postorder: y a 1 + 2 b * x min3 z * =

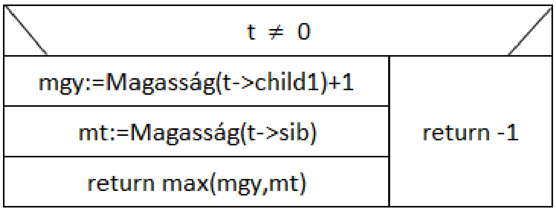
Algoritmus készítős feladatok

Az algoritmusokat készíthetjük teljesen rekurzívan, vagy rekurzívan a child1 irányban, és iteratívan a testvérek listájának irányában. (OEP-ből tanulták a felsorolós programozási tételeket, itt látható az alkalmazásuk: a bejáró algoritmusokat, mint a fa csúcsait felsoroló algoritmust használjuk.)

Készítsük el a fa magasságát megadó algoritmust.

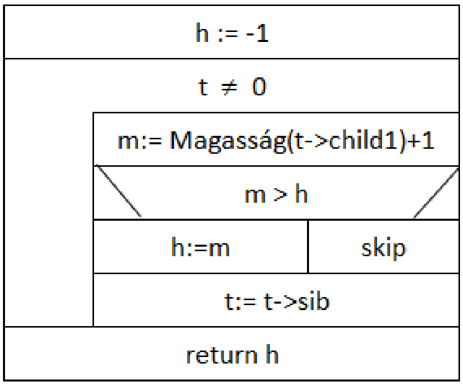
Teljesen rekurzív megoldás

Magasság(t:Node*) : Z



Testvér irányban ciklust használó megoldás

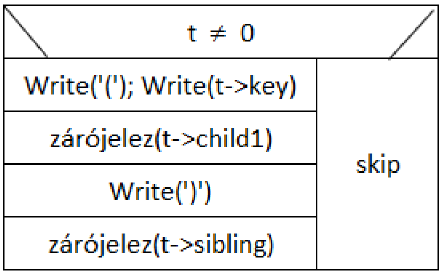
Magasság(t:Node*) : Z



Készítsünk algoritmust, mely kiírja a fa zárójelezett alakját

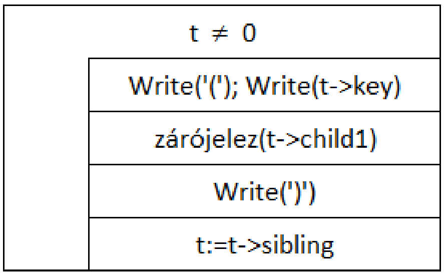
Teljesen rekurzív megoldás

zárójelez(t:Node*)



Testvér irányban ciklust használó megoldás

zárójelez(t:Node*)

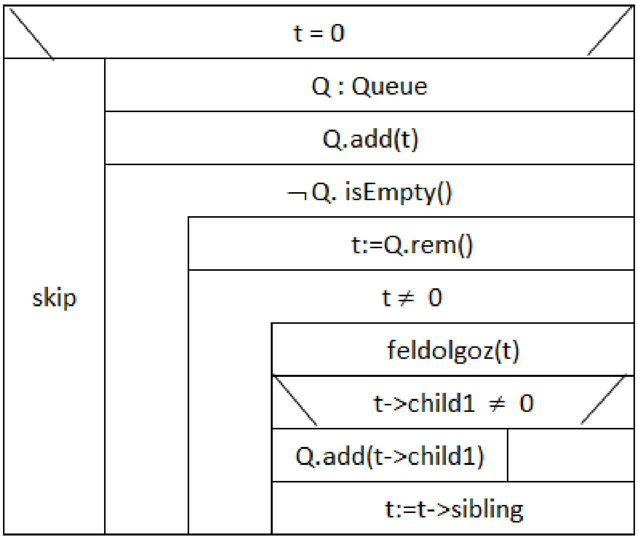


Érdemes a hallgatókkal közösen azt a változatot elkészíteni, mely három féle zárójelt használ :{}, [], () Ötlet: egy paramétert beteszünk még az algoritmusba, mely azt adja meg, milyen szinten járunk a fában, és a szinttől függően választjuk a megfelelő zárójelt.

Ez feladható esetleg szorgalmi házi feladatnak is.

Készítsük el a szintfolytonos bejárást a két pointerrel ábrázolt általános fára.

Szintfolytonos(t_Node*)



Szorgalmi házi feladatok, amiből válogatni lehet:

1. Adott egy két pointerrel, láncoltan ábrázolt általános fa. Egy csúcsban az első gyerekre és a testvérre mutató pointerek vannak. A kulcsok egész számok. Készítsen rekurzív algoritmust, mely kiírja minden testvér csoportból az egyik legnagyobb kulcsot. (Ennek megoldása a következő oldalon megtalálható.)

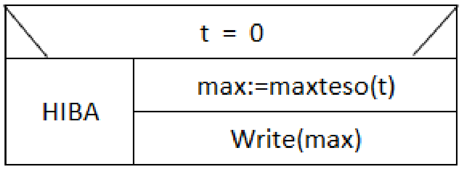
2. Adott egy két pointerrel, láncoltan ábrázolt általános fa. Egy csúcsban az első gyerekre és a testvérre mutató pointerek vannak. Készítse el a következő algoritmust: megadja annak a szülőnek a címét, amelynek a legtöbb gyereke van. Adja meg azt is, hogy hány gyereke van. Felteheti, hogy a fa nem üres, elegendő egy ilyen szülőt megadni, ha a maximum nem egyértelmű.

3. Adott egy két pointerrel, láncoltan ábrázolt általános fa. Egy csúcsban az első gyerekre és a testvérre mutató pointerek vannak. Készítse el a következő kereső algoritmust: megadunk egy kulcsot, ha a fában van ilyen kulcs, akkor kiírja a csúcsig vezető úton a szülők kulcsait. Elképzelhető, hogy az adott csúcs több helyen is szerepel a fában, akkor mindegyik előforduláshoz írja ki az útvonalat! Az utat a gyökértől indulva, a csúcsig kell kiírni. (Megjegyzés: feladható úgyis, hogy van egy harmadik pointer is a Node-okban, mely a szülőre mutat, az is érdekes.)
A feladat pl a DOS dir fájlnév /s parancsával szemléltethető: egy alkönyvtár rendszerben megkeressük azokat az alkönyvtárakat, amelyekben az adott fájlnév előfordul, kiírjuk a fájlhoz vezető elérési útvonalat.

Az 1-es feladat megoldása:

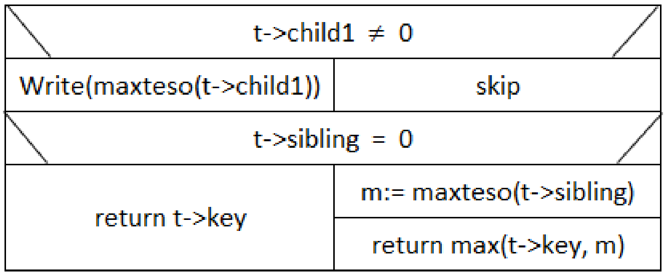
A bejáró algoritmusok előfeltétele lesz, hogy t nem az üres fa. Így szükség lesz egy felső szintre, amelyik ellenőrzi, hogy a paraméterben kapott pointer nem null értékű. A bejáró algoritmus függvény lesz, így az indító eljárás is visszakap egy maximum értéket (a gyökérben található értéket). Ha a gyökeret, is mint testvércsoport-ot dolgozzuk fel, akkor ezt is kiírhatjuk.

Maxtestvér(t:Node*)



maxteso teljesen rekurzívan (előfeltétel: t nem üres fa)

maxteso(t:Node*) : Z



maxteso testvér irányban iteratív (előfeltétel: t nem üres fa)

maxteso(t:Node*) : Z