Adatszerkezetek és algoritmusok

Horváth Géza

hatodik előadás

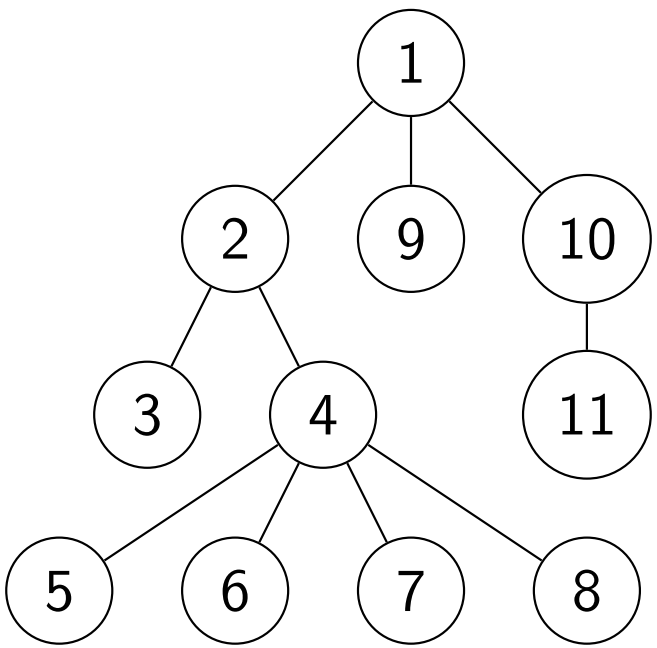
Előadások témái

1. Az algoritmusokkal kapcsolatos alapfogalmak bevezetése egyszerű példákon keresztül.
2. Az algoritmusok futási idejének aszimptotikus korlátai.
3. Az adatszerkezetekkel kapcsolatos alapfogalmak. A halmaz, a multihalmaz és a tömb adatszerkezet bemutatása.
4. Az adatszerkezetek folytonos és szétszórt reprezentációja. A verem, a sor és a lista.
5. Táblázatok, önátrendező táblázatok, hash függvények és hash táblák, ütközéskezelés.
6. Fák, bináris fák, bináris keresőfák, bejárás, keresés, beszúrás, törlés.
7. Kiegyensúlyozott bináris keresőfák: AVL fák.
8. Piros-fekete fák.
9. B-fák.
10. Gráfok, bejárás, legrövidebb út megkeresése.
11. Párhuzamos algoritmusok.
12. Eldönthetőség és bonyolultság, a P és az NP problémaosztályok.
13. Lineáris idejű rendezés. Összefoglalás.

A fa, mint absztrakt adattípus

A fa adattípust egyrészt a csúcsok halmaza, másrészt pedig a csúcsokat összekötő élek halmaza alkotja.
A fa adattípus a következő tulajdonságokkal rendelkezik:

- A fa bármely két csúcsa között pontosan egy út vezet.
- Van egy kijelölt csúcs, melyet gyökérnek hívunk.



A fa – fontosabb fogalmak

- csúcs
  - gyökér
  - közbenső elem
  - levél
- él
- szülő
- ős (szülő, nagyszülő, dédi, stb.)
- a fa magassága (az ősök számának maximuma)
- gyerek
- leszármazott (gyerek, unoka, dédunoka, stb.)
- testvér (két olyan csúcs, melyeknek közös a szülőjük)
- út
- részfa

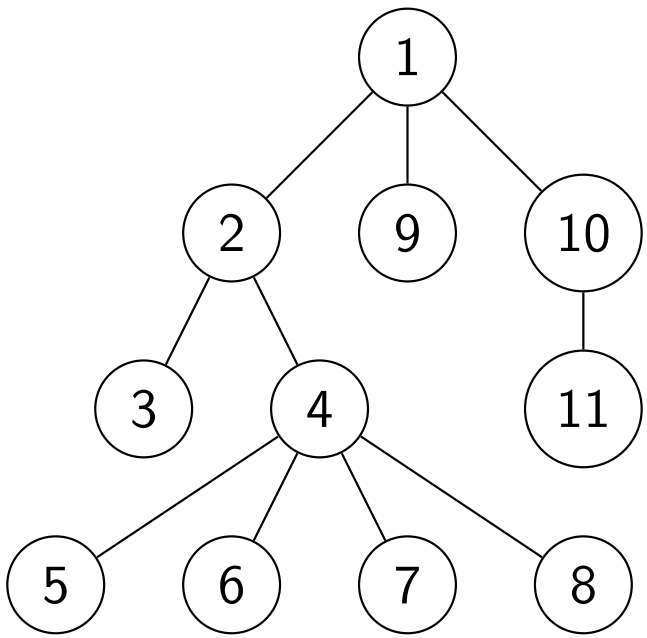
A fa, mint adatszerkezet

A fa tulajdonságai:

- homogén
- dinamikus
- hierarchikus
- néha folytonos reprezentációval tároljuk, míg máskor szétszórt (láncolt) reprezentációval

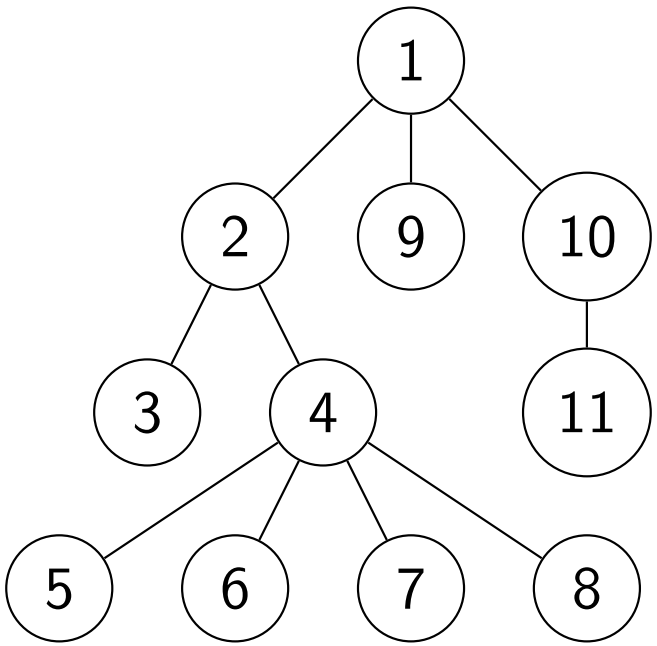
A fa láncolt reprezentációja

Sajnos a fát általános esetben nem tudjuk úgy tárolni, hogy nyilvántartjuk minden csúcshoz az értékét, és minden gyerekéhez egy rá mutató mutatót, mivel az adaszerkezet létrehozásakor nem tudjuk előre, hogy később, miután bővitjuk a fát, mennyi lesz a gyerekek maximális száma egy adott csúcs esetén. Ezért minden csúcs esetén az értéke mellett pontosan 2 mutatót használunk, az egyik a legbaloldalibb gyerekére mutat, a másik pedig a jobboldali testvérére. (Lássuk a rajzokat a táblán!)



A fa folytonos reprezentációja – hierarchikus lista

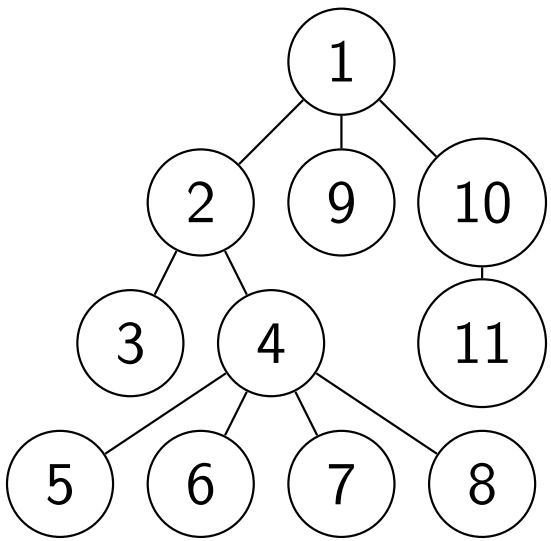
A hierarchikus lista a fa folytonos reprezentációjának az egyik lehetséges módja, mely nem rendelkezik közvetlen hozzáféréssel.



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

A fa bejárása

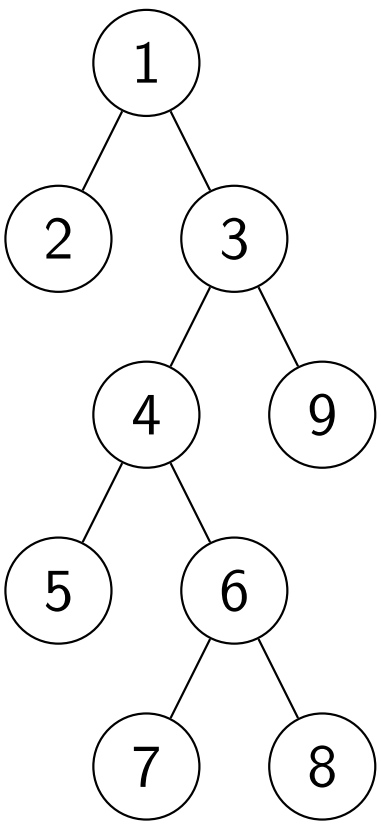
A fa bejárása olyan eljárás, mely esetén a fa minden csúcsát pontosan egyszer látogatjuk meg. A fa bejáró algoritmusokat az alapján különböztetjük meg, hogy azok milyen sorrendben látogatják meg a csúcsokat.



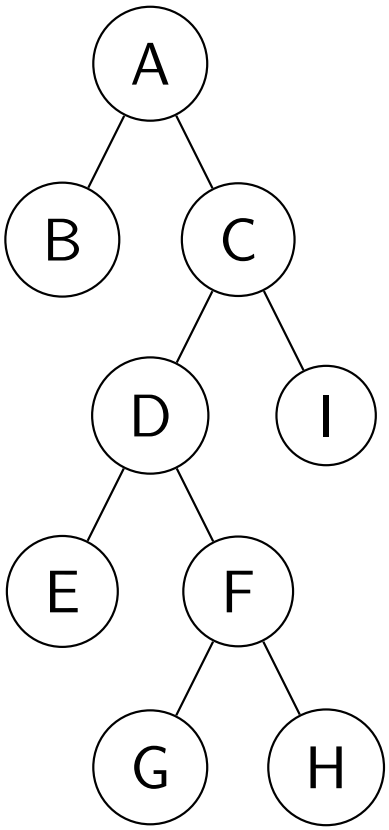
Preorder: 1,2,3,4,5,6,7,8,9,10,11
Postorder: 3,5,6,7,8,4,2,9,11,10,1

A bináris fa, mint absztrakt adattípus

A bináris fa olyan fa, ahol minden csúcsnak legfeljebb két gyereke lehet. Minden gyerek vagy baloldali, vagy jobboldali.

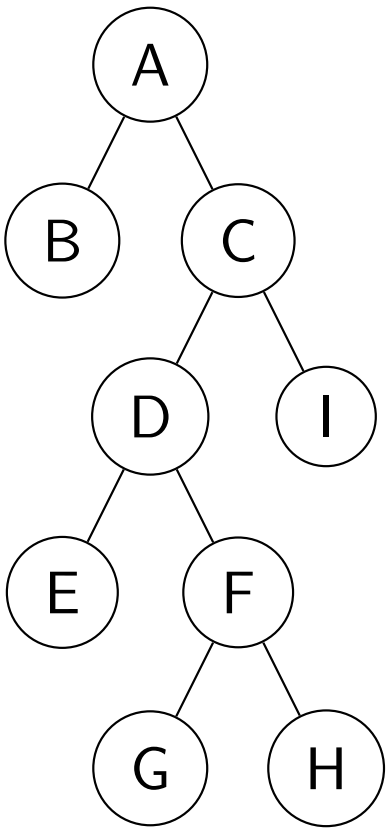


A bináris fa láncolt reprezentációja



A bináris fa esetében már az adaszerkezet létrehozásakor tudjuk előre, hogy egy csúcsnak maximum 2 gyereke lehet, ezért tudjuk úgy tárolni, hogy nyilvántartjuk minden csúcshoz az értékét, és minkét gyerekéhez egy rá mutató mutatót. Ezért minden csúcs esetén az értéke mellett pontosan 2 mutatót használunk, az egyik a baloldali gyerekére mutat, a másik pedig a jobboldali gyerekére. (Lássuk a rajzokat a táblán!)

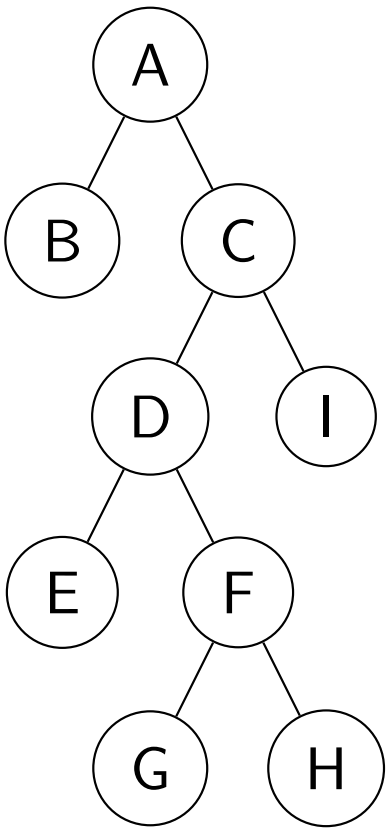
A bináris fa folytonos reprezentációja



	kulcs	bal	jobb
1	A	2	3
2	B	0	0
3	C	4	9
4	D	5	6
5	E	0	0
6	F	7	8
7	G	0	0
8	H	0	0
9	I	0	0

Gyökér: 1

A bináris fa folytonos reprezentációja

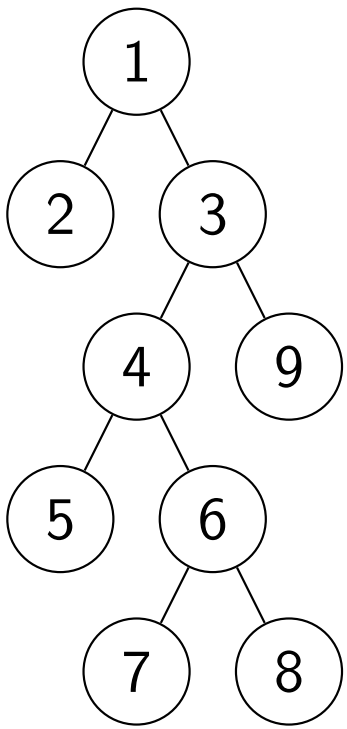


	kulcs	bal	jobb	szülő
1	A	2	3	0
2	B	0	0	1
3	C	4	9	1
4	D	5	6	3
5	E	0	0	4
6	F	7	8	4
7	G	0	0	6
8	H	0	0	6
9	I	0	0	3

Root: 1

A bináris fa bejárása

A fa bejárása olyan eljárás, mely esetén a fa minden csúcsát pontosan egyszer látogatjuk meg. A fa bejáró algoritmusokat az alapján különböztetjük meg, hogy azok milyen sorrendben látogatják meg a csúcsokat.



Preorder: 1,2,3,4,5,6,7,8,9

Inorder: 2,1,5,4,7,6,8,3,9

Postorder: 2,5,7,8,6,4,9,3,1

A bejárás nem tekinthető folytonos reprezentációnak, mivel nem egyértelmű.

A bináris fa bejárása

```
PREORDER-TREE-TRAVERSAL(x)
1 if x≠NIL
2    print x.key
3    PREORDER-TREE-TRAVERSAL(x.left)
4    PREORDER-TREE-TRAVERSAL(x.right)
```

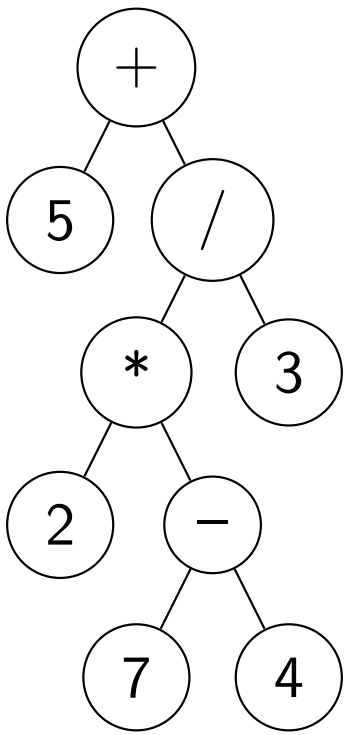
```
INORDER-TREE-TRAVERSAL(x)
1 if x≠NIL
2    INORDER-TREE-TRAVERSAL(x.left)
3    print x.key
4    INORDER-TREE-TRAVERSAL(x.right)
```

```
POSTORDER-TREE-TRAVERSAL(x)
1 if x≠NIL
2    POSTORDER-TREE-TRAVERSAL(x.left)
3    POSTORDER-TREE-TRAVERSAL(x.right)
4    print x.key
```

A kifejezésfa, mint absztrakt adattípus

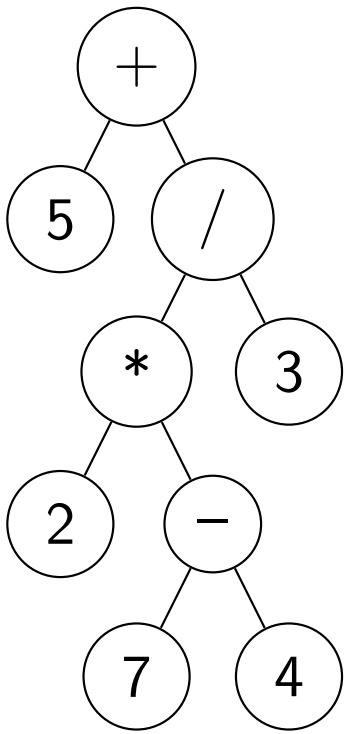
Az olyan bináris fát, mely minden közbenső eleme egy műveleti jelet tartalmaz, és minden levél eleme egy operandus, kifejezésfának hívjuk.

Példa: 5+2*(7–4)/3



A kifejezésfa bejárása

Példa: 5+2*(7–4)/3



Prefix: +5/*2–743

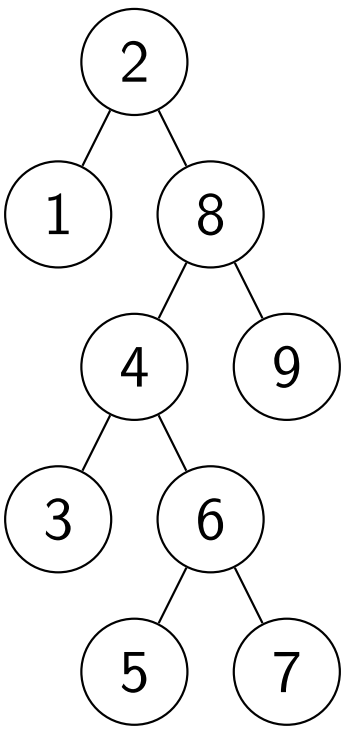
Infix: 5+2*7–4/3

Postfix: 5274–*3/+

Kifejezésfa esetén mind a prefix mind pedig a postfix bejárás egyértelmű.

A bináris keresőfa, mint absztrakt adattípus

A bináris keresőfa olyan bináris fa, ami kielégíti az alábbi feltételt: Legyen x a bináris keresőfa egy tetszőleges csúcsa. Ha az y csúcs az x csúcs baloldali részfájában található, akkor y.key$<$x.key. Ha az y csúcs az x csúcs jobboldali részfájában található, akkor y.key$>$x.key.

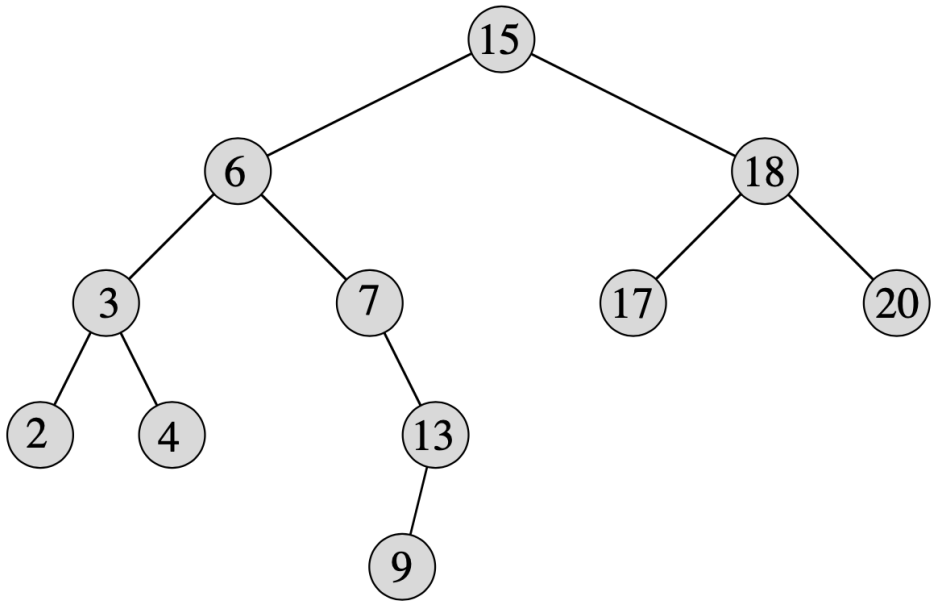


A bináris keresőfa műveletei

- **Keresés**
- **Beszúrás**
- **Törlés**
- Minimum
- Maximum
- Megelőző elem
- Rákövetkező elem

A bináris keresőfa műveleteinek időigénye alapvetően a fa magasságával egyenesen arányos. Egy n csúcsot tartalmazó teljes bináris keresőfa esetén egy művelet végrehajtásához szükséges idő a legrosszabb esetben $\Theta(\log_2 n)$. Ugyanakkor abban az esetben, ha a keresőfa elemei egy egyenest alkotnak, akkor egy művelet végrehajtásához szükséges idő a legrosszabb esetben $\Theta(n)$.

A bináris keresőfa műveletei



12.2. ábra. Keresés bináris keresőfában. A fában lévő 13-as kulcs keresése során a gyökérből induló 15 → 6 → 7 → 13 utat követjük. A fa minimális kulcsa 2, melyet a gyökérből a *bal* mutatókat követve érhetünk el. A fa maximális kulcsa 20, melyet szintén a gyökérből indulva, a *jobb* mutatókat követve találhatunk meg. A 15-ös kulcsú csúcs rákövetkezője a 17-es kulcsú csúcs, mivel az a 15 bal oldali részfájának minimális kulcsa. A 13-as kulcsú csúcsnak nincs jobb oldali részfája, ezért annak rákövetkezője az a legalacsonyabban lévő őse, amelynek bal oldali gyereke is őse a 13-nak. Esetünkben a 15-ös kulcsú csúcs a rákövetkező.

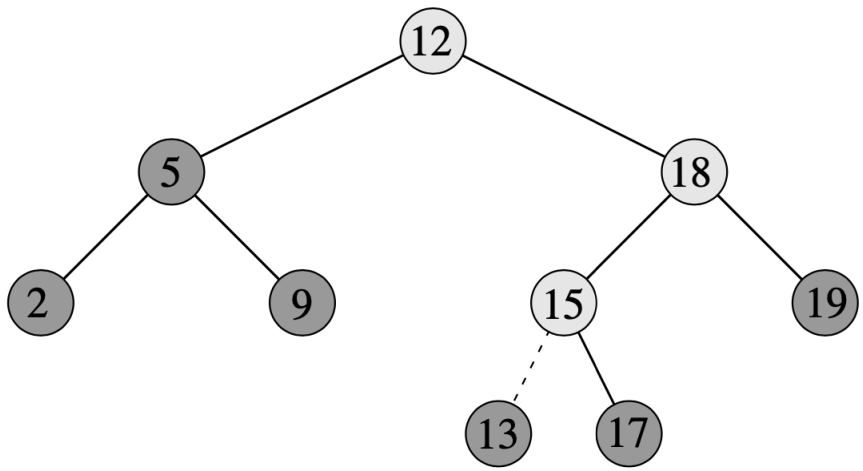
A bináris keresőfa műveletei

```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2       return x
3  if k < x.key
4       return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

A fentebbi művelet - csakúgy, mint minden rekurzív művelet, - megvalósítható ciklussal is. (Ez utóbbi változat a hatékonyabb.)

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2       if k < x.key
3            x = x.left
4       else x = x.right
5  return x
```

A bináris keresőfa műveletei



12.3. ábra. A 13 kulcsértékű elem beillesztése egy bináris keresőfába. A halványan árnyékolt csúcsok a gyökérből a beszúrás helyéhez vezető utat mutatják. A szaggatott vonal azt az új mutatót szemlélteti, amely az új elemet a fába illeszti.

A bináris keresőfa műveletei

```
TREE-INSERT(T, z)
 1   y = NIL
 2   x = T.root
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z          // tree T was empty
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

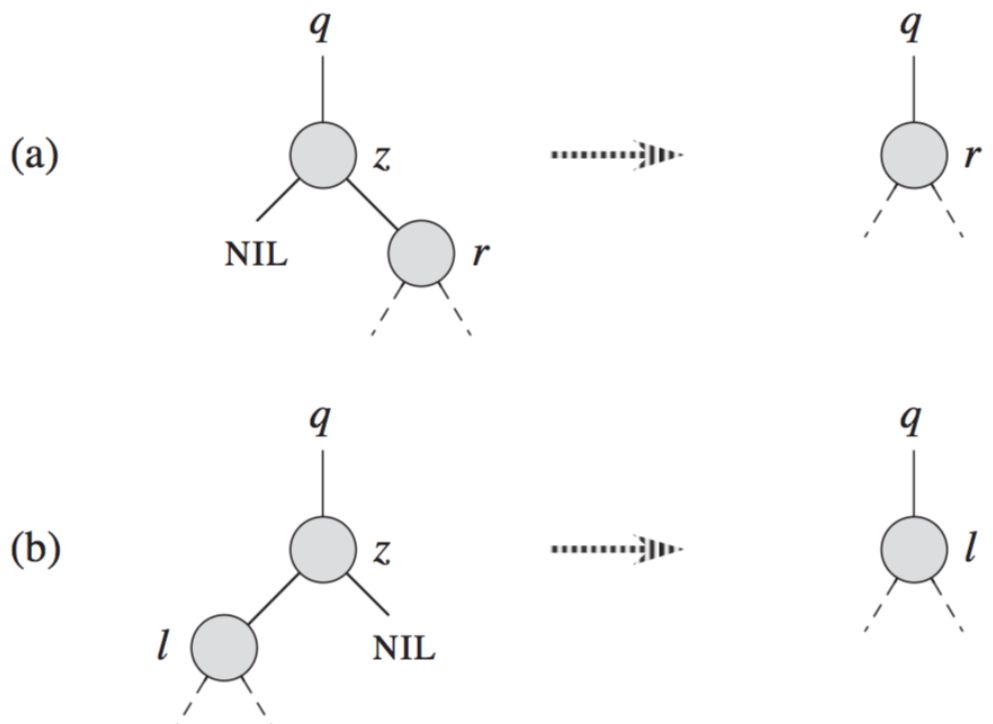
A bináris keresőfa műveletei

Egy bináris keresőfa valamely z csúcsának törlése elég trükkös feladat. Alapvetően 3 esetet különböztetünk meg:

- Ha z-nek nincs gyereke, akkor egyszerűen törölhető.
- Ha z-nek egy gyereke van, akkor z-t töröljük, és létrehozunk egy kapcsolatot a szülője és a gyereke között.
- Ha pedig két gyereke van, akkor megkeressük a legkisebb rákövetkező elemet, nevezzük ezt az elemet y-nak, beillesztjük y-t a z helyére, és y-t a régi helyéről töröljük. Ez megtehető, mivel y-nak legfeljebb egy gyereke lehet.

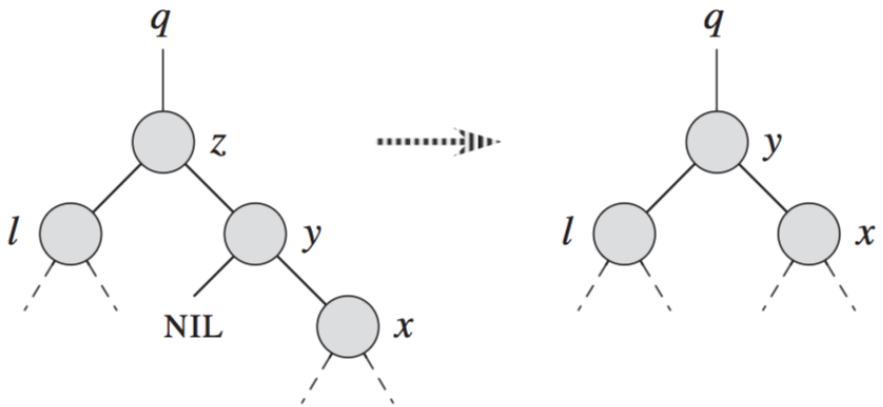
A bináris keresőfa műveletei

Figure 12.4 Deleting a node z from a binary search tree. Node z may be the root, a left child of node q, or a right child of q. **(a)** Node z has no left child. We replace z by its right child r, which may or may not be NIL. **(b)** Node z has a left child l but no right child. We replace z by l. **(c)** Node z has two children; its left child is node l, its right child is its successor y, and y's right child is node x. We replace z by y, updating y's left child to become l, but leaving x as y's right child. **(d)** Node z has two children (left child l and right child r), and its successor $y \neq r$ lies within the subtree rooted at r. We replace y by its own right child x, and we set y to be r's parent. Then, we set y to be q's child and the parent of l.

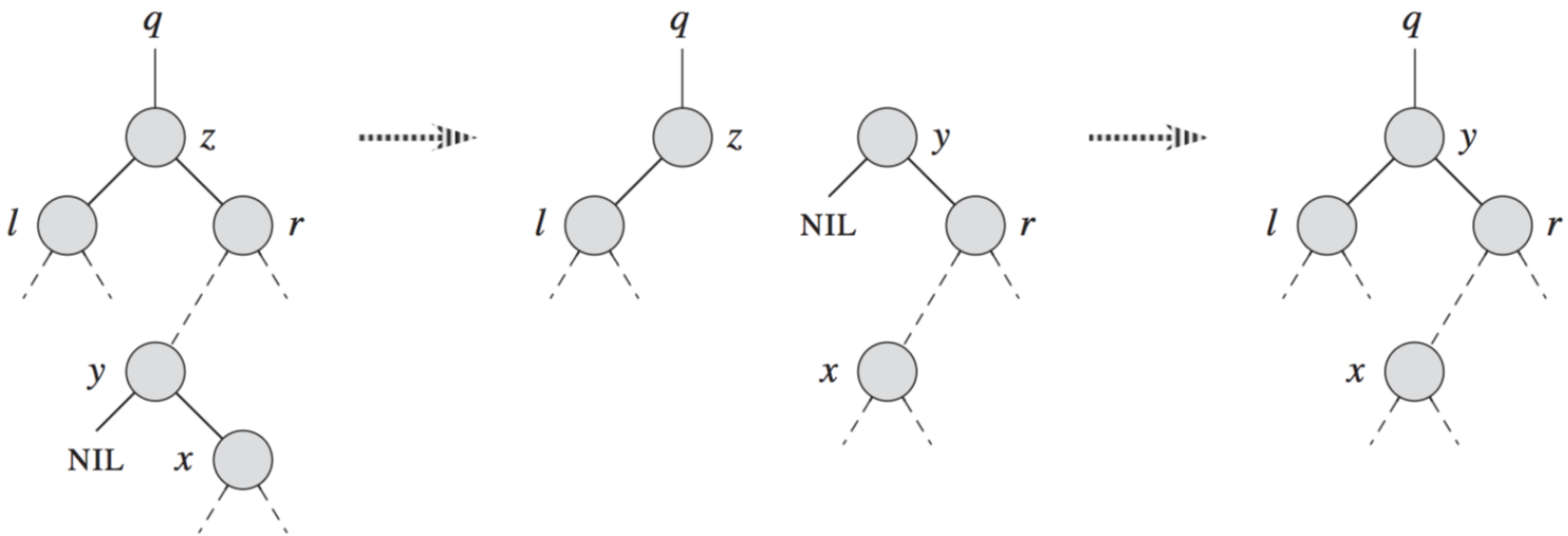


A bináris keresőfa műveletei

(c)



(d)



Műveletek bináris keresőfákkal, mint adatszerkezetekkel

Műveletek:

- adatszerkezetek **létrehozása**: folytonos vagy láncolt reprezentációval
- adatszerkezetek **módosítása**
  - elem hozáadása: TREE-INSERT
  - elem törlése: TREE-DELETE
  - elem cseréje: **nincs**
- elem **elérése**: ITERATIVE-TREE-SEARCH

Irodalomjegyzék