Kőrösi Gábor

Algoritmusok és adatszerkezetek a gyakorlatban

Jelen tananyag a Szegedi Tudományegyetemen készült az Európai Unió támogatásával.

Projekt azonosító: EFOP-3.4.3-16-2016-00014

Szegedi Tudományegyetem
Cím: 6720 Szeged, Dugonics tér 13.
www.u-szeged.hu
www.szechenyi2020.hu

Keresőfák

Összefoglaló

Az eddigiekben már számtalan adattárolási megoldást vagy algoritmust láthattunk, amelyeknek célja minden esetben az volt, hogy az O(n) időkomplexitást polinom vagy O(log n)-ra csökkentsük. A jelen fejezetben egy újabb adatelérési módszerrel ismerkedünk meg, ami egy tárolt adathalmaz elemét képes O(log n) idő alatt elérni. Az listáknak túl sok előnyét nem vehetjük, ha keresünk egy elemet, hiszen akkor az egész listát fel kell túrnunk. A keresőfa ehhez képest annyi előnyt jelent, hogy minden keresett elemről meg tudjuk mondani, hogy merre lehet. A keresőfára jellemző, hogy a bal oldali részfájában a gyökérnél kisebb elemek találhatóak, míg a jobb oldali részfájában a gyökérnél nagyobb elemek. Ez a tulajdonság igaz a részfáira is, mely gyorsítja az elem elérésének idejét.

Lecke fejezetei:

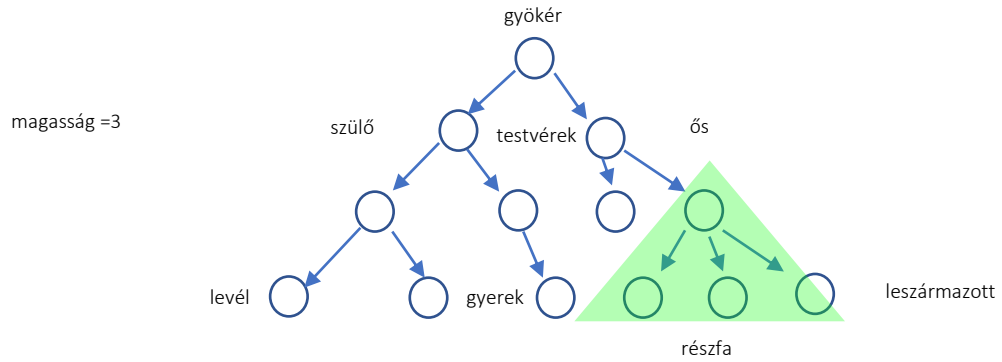
- A keresőfák felépítése – Olvasó (7 perc)
- Bináris keresőfák – Olvasó (3 perc)
- Műveletek a bináris keresőfákkal – Olvasó (5 perc)
- Legkisebb és legnagyobb elem – Olvasó (10 perc)
- Fa vizualizációja – Olvasó (5 perc)
- Elem törlése a fából – Olvasó (10 perc)
- Python implementáció – Olvasó (5 perc)
- Gyakorló feladatok – Gyakorlati (45 perc)

Téma típusa: Gyakorlati
Olvasási és gyakorlási idő: 90 perc

A keresőfák felépítése *(7 perc)*

Mielőtt rátérnénk a keresőfákra, nézzük meg azt, hogy mi is valójában az a fa. A fa egy összefüggő, körmentes gráf, melyre igaz, hogy:
- (Általában) egy gyökér csúcsa van, melynek 0 vagy több részfája van
- Pontosan egy út vezet bármely két csúcsa között
- A gyökéren kívül minden csúcsnak pontosan egy szülője van

A fa felépítése a következő:

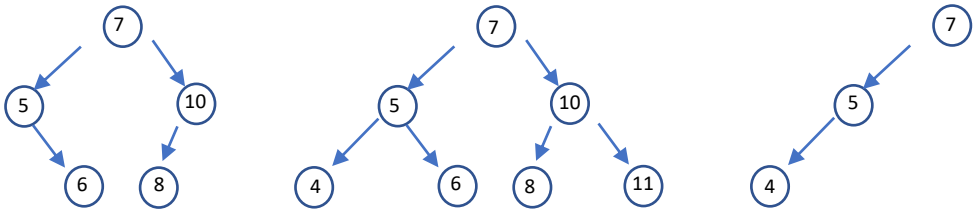


Szülő(i): az a csúcs, amely közvetlenül az i felett van
Gyerek(i): az i csúcs alatt közvetlen lévő csúcsok
Testvérek: ugyanannak a csúcsnak a gyerekei
Gyökér: az egyetlen csúcs, aminek nincsen szülője
Levél: olyan csúcs, amelynek nincs gyereke

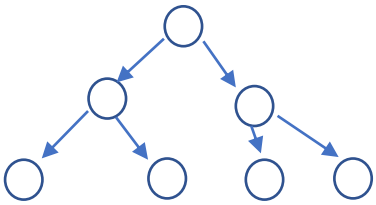
Magasság: a gyökértől bármely levélbe vezető leghosszabb út
Részfa(n): az a fa, amelynek a gyökere az n
Ős(n): minden csúcs az n-től a gyökérig vezető úton
Leszármazott(n): minden csúcs, amely az n gyökerű részfában van

Bináris keresőfák (3 perc)

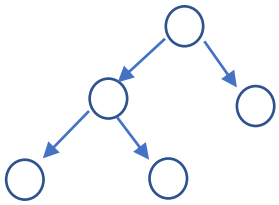
Bináris kereső fa minden csúcsnak legfeljebb két gyereke van. Minden gyökérelemnél kisebb balra, míg a nagyobb elemek jobbra helyezkednek el. A bináris kereső fa ágai további „kicsi" bináris kereső fára bonthatók.
Ezt addig tesszük, amíg a levelekhez nem érünk.



Teljes (full) bináris fáról akkor beszélünk, ha minden szint teljesen ki van töltve, míg majdnem teljes (complete) bináris fa az, amelynél a legalsó szint nincs teljesen kitöltve, csak balról jobbra haladva kitöltött néhány elemig.

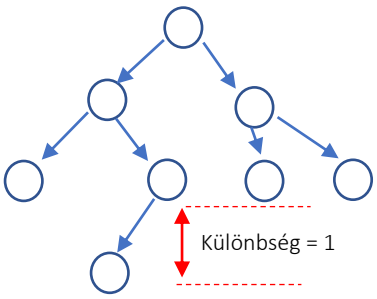


Teljes bináris fa

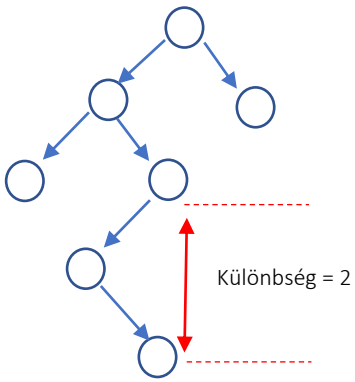


Majdnem teljes bináris

A bináris fa további tulajdonsága a kiegyensúlyozottság. A kiegyensúlyozott bináris fa, ha a fa bármely elemének bal és jobboldali részfájában az elemek száma legfeljebb eggyel tér el.



Kiegyensúlyozott fa



Nem kiegyensúlyozott fa

A bináris fák megalkotásának elsődleges célja a KERES művelet idejének gyorsítása volt, mely O(log n) időben megvalósítható. Hasonlítsuk össze a bináris keresőfa időkomplexitását más adatszerkezettel.

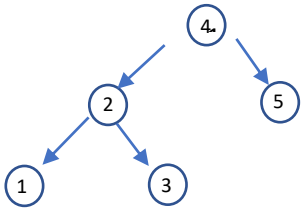
	Tömb	Láncolt lista	Tömb	BKF

	(rendezetlen)		(rendezett)	
Keresés	O(n)	O(n)	O(log n)	O(log n)
Beszúrás	O(1)	O(1)	O(n)	O(log n)
Törlés	O(n)	O(n)	O(n)	O(log n)

Műveletek a bináris keresőfákkal *(5 perc)*

A bináris kererső fában leggyakrabban ezeket a műveleteket szoktuk végezni:
Keres – egy elem megkeresése a fában
Min/max – a legkisebb vagy a legnagyobb elem megkeresése a fában
Következő (nagyság szerint) – elemek sorba rakása
Beszúr – egy új elem beszúrása a fába
Töröl (levél, egy gyerek, két gyerek) – egy elem törlése a fából
Fa magassága – levél és a gyökérelem közötti távolság

A fent leírt feladatok végrehajtásához azonban valamilyen módon be kell járnunk a fa egyes elemit, melyet a következőtekéképpen tehetünk meg:



- Inorder (bal, Root, jobb) : 1 2 3 4 5
- Preorder (Root, bal, jobb) : 4 2 1 3 5
- Postorder (bal, jobb, Root) : 1 3 2 5 4

Az Inorder bejárás algoritmusa:
1. A baloldali fa bejárása.
2. A root meglátogatása.
3. A jobboldali részfa bejárása.

A Preorder bejárás algoritmusa:
1. A root meglátogatása.
2. A baloldali fa bejárása.
3. A jobboldali részfa bejárása

A Postorder bejárás algoritmusa:
1. A baloldali fa bejárása.
2. A jobboldali részfa bejárása
3. A root meglátogatása.

A bejárások megvalósítása a következőféleképpen néz ki kód formájában:

```
class Node:
    def __init__(self,key):
        self.left = None
        self.right = None
        self.val = key

def printInorder(root):
    if root:
        # eloszor haladjunk rekurzivan a bal gyerek fele
        printInorder(root.left)
        # majd irjuk ki az aktualis elemet
        print(root.val),
        # most haladjunk rekurzivan a jobb gyerek fele
        printInorder(root.right)

def printPostorder(root):
    if root:

        # eloszor haladjunk rekurzivan a bal gyerek fele
        printPostorder(root.left)
```

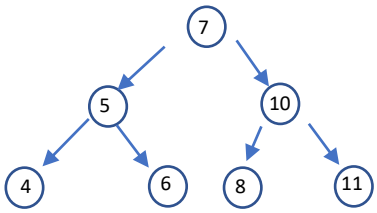
```
        # majd haladjunk rekurzivan a jobb gyerek fele
        printPostorder(root.right)
        # most irjuk ki az aktualis elemet
        print(root.val),

def printPreorder(root):
    if root:
        # eloszor irjuk ki az aktualis elemet
        print(root.val),
        # majd haladjunk rekurzivan a bal gyerek fele
        printPreorder(root.left)
        # majd haladjunk rekurzivan a jobb gyerek fele
        printPreorder(root.right)

root = Node(1)
root.left= Node(2)
root.right= Node(3)
root.left.left= Node(4)
root.left.right= Node(5)

print ("Preorder bejaras")
printPreorder(root)
print ("\nInorder bejaras")
printInorder(root)
print ("\nPostorder bejaras ")
printPostorder(root)
```

```
Preorder bejaras
1 2 4 5 3
Inorder bejaras
4 2 5 1 3
Postorder bejaras
4 5 2 3 1
```

.

Legkisebb és legnagyobb elem (10 perc)

Melyik a fa minimális illetve maximális eleme? A BKF legnagyobb előnye, hogy eleve rendezettek, így ha például a legkisebb vagy a legnagyobb elemet szeretnénk megtalálni, akkor ez nem jár túl sok költséggel. A legkisebb elemek mindig (leg)balról, még a legnagyobbak (leg)jobbról találhatóak.




```
class Node:
  def __init__(self,key):
    self.left = None
    self.right = None
    self.val = key

def min_(root):       # haladjunk balra, amig csak tudunk, a legbaloldalibb elem a legkissebb
    if root:
        min_(root.left)
        if not (root.left):
            print(root.val)

def max_(root):             # haladjunk jobbra, amig csak tudunk, a legjobboldalibb elem a legnagyobb
    if root:
        max_(root.right)
        if not (root.right):
```

```
            print(root.val)

# toltsuk fel elemekkel a fat
root = Node(7)
root.left= Node(5)
root.right= Node(10)
root.left.left= Node(4)
root.left.right= Node(6)
root.right.left= Node(8)
root.right.right= Node(11)
# keressuk meg a min/max elemet
print("Min")
min_(root)
print("Max")
max_(root)
```

Fa vizualizációja (5 perc)

A tananyag elsajátítása érdekében célszerű, hogy a keresőfát lépésről lépésre vizualizáljuk. Ebben nyújt segítséget a *binarytree* csomag, mellyel sorról sorra, lépésről lépésre tudjuk megjeleníteni a fa aktuális állapotát.

```
from binarytree import Node

root = Node(7)
print(root)
root.left= Node(5)
print(root)
root.right= Node(10)
print(root)
root.left.left= Node(4)
print(root)
root.left.right= Node(6)
print(root)
root.right.left= Node(8)
print(root)
```

```
7

  7
 /
5

  7
 / \
5   10

    __7
   /   \
  5     10
 / \
4   6

  7
 / \
5   10
/
4

    __7__
   /     \
  5       10
 / \     /
4   6   8
```

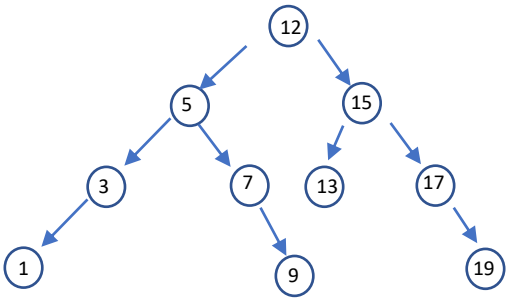
Egy másik példa a MAX elem keresésének vizualizációjára:

```
from binarytree import Node
def max_(root):            # haladjunk jobbra, amig csak tudunk, a legjobboldalibb elem a legnagyobb
    if root:
        print(root)
        max_(root.right)

# toltsuk fel elemekkel a fat
root = Node(7)
root.left= Node(5)
root.right= Node(10)
root.left.left= Node(4)
root.left.right= Node(6)
root.right.left= Node(8)
root.right.right= Node(11)
# keressuk meg a min/max elemet
max_(root)
```

```
    __7__
   /     \
  5       10
 / \     /  \
4   6   8    11

  10
 /  \
8    11

11
```

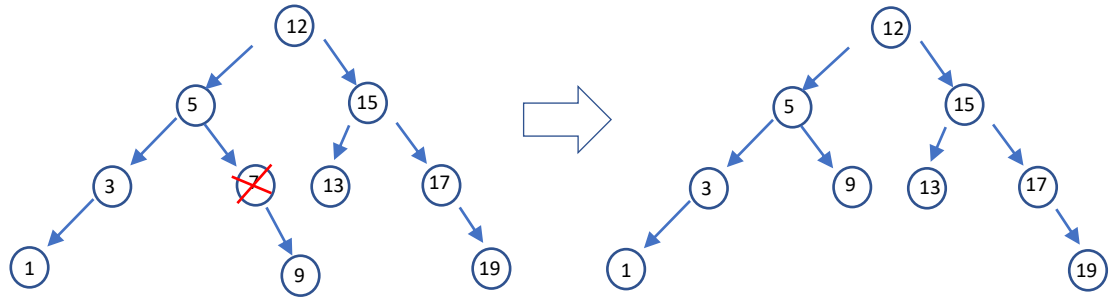
Elem törlése a fából (10 perc)

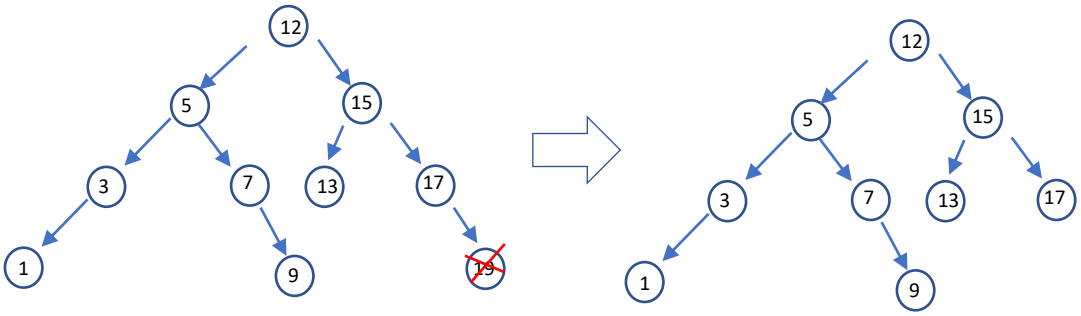
Adott egy bináris kereső fa, hogyan törölnénk belőle egy elemet?



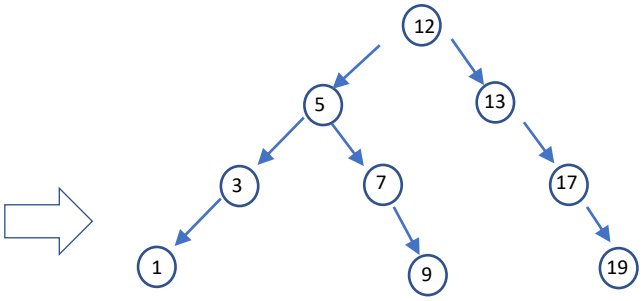
Hogyan törölnénk belőle a 19-es, 7-es vagy a 15-ös elemet?

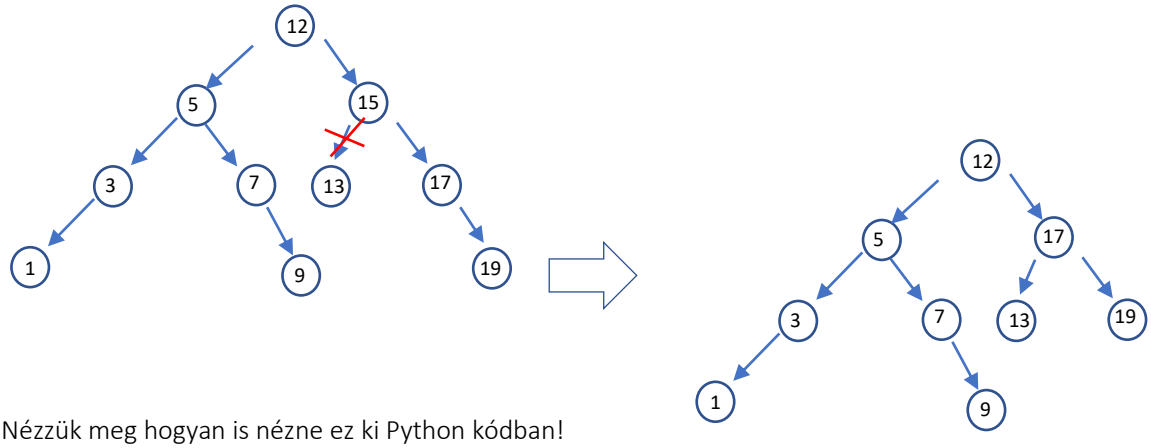
Könnyű ezt megtennünk egy „levéllel” vagy abban az esetben, ha csak egy leszármazottja van a törlendő elemünknek.





Viszont az okoz nehézséget, ha a törlendő elemnek két gyereke van, mivel ilyenkor az egyetlen törölt elem helyét nem foglalhatja el a két gyereke. Illetve azt is figyelembe kell vennünk, hogy törölni a BKF-ból csak úgy lehet, hogy a kiesett elem után megmaradjon a bináris fa megfelelő szerkezete (jobbra kisebbek, balra nagyobbak). Éppen ezért, ennél a műveletnél óvatosan kell eljárnunk, mely szerint törléskor megkeressük a jobboldali részfa legkisebb vagy a baloldali részfa legnagyobb elemét, és annak értékével helyettesítjük a törlendő elemet.
Más szóval, az Inorder bejárás szerint előtte vagy utána lévő elemet.





Nézzük meg hogyan is nézne ez ki Python kódban!

```
# forras https://www.geeksforgeeks.org/binary-search-tree-set-2-delete/
from binarytree import Node

def insert( node, val):
    if node is None: # amennyiben a fa ures
        return Node(val)
    # egyebkent keressuk meg a megfelelo helyet
    if val < node.val:
        node.left = insert(node.left, val)
    else:
        node.right = insert(node.right, val)
    return node

def minValueNode( node):
    current = node
    # keressuk meg a legbaloldalibb elemet
    while(current.left is not None):
        current = current.left

    return current
def deleteNode(root, val):

    if root is None: #ures a fa
        return root
    # keressuk meg az elemunket a faban
    if val < root.val:
        root.left = deleteNode(root.left, val)
    elif(val > root.val):
        root.right = deleteNode(root.right, val)
    else:
        # Node torlese egy gyerekkel, vagy gyerek nelkul
        if root.left is None:
            temp = root.right
            root = None
            return temp

        elif root.right is None:
            temp = root.left
            root = None
            return temp

        # Node torlese 2 gyerek eseten:
        # Keressuk meg az Inorder bejaras szerinti kovetkezo elemet
        temp = minValueNode(root.right)
        # masoljuk at az ertekeket
        root.val = temp.val
        # toroljuk az elemet
        root.right = deleteNode(root.right , temp.val)
```

```
      ____50___
     /         \
   _30         _70
  /   \       /   \
20    40    60    80
```

Delete 20

```
    ____50___
   /         \
 30          _70
   \        /   \
    40     60    80
```

Delete 30

```
   _50___
  /      \
 40      _70
        /   \
       60    80
```

Delete 50

```
  _60
 /   \
40    70
        \
         80
```

```
    return root

root = None
root = insert(root, 50)
root = insert(root, 30)
root = insert(root, 20)
root = insert(root, 40)
root = insert(root, 70)
root = insert(root, 60)
root = (root, 80)

print(root)
print ("\nDelete 20")
root = deleteNode(root, 20)
print(root)
print ("\nDelete 30")
root = deleteNode(root, 30)
print(root)
print ("\nDelete 50")
root = deleteNode(root, 50)
print(root)
```

Python implementáció (5 perc)

Amennyiben megértettük a BKF működését, úgy szükségtelen a saját implementációnkat használni, hiszen ehhez már léteznek előre megírt könyvtárak. Erről bővebben itt találhatunk leírást: Link

Gyakorló feladatok (45 perc)

1. Módosítsd úgy a törlésre szolgáló programot, hogy a minden egyes lépés után újra rajzoljuk a fát!
2. Készíts egy programot, amely a BKF időkomplexitását ábrázolja, beszúrás, törlés és keresés függvényekkel. Hasonlítsuk össze a kapott eredményt az előző órán tanult láncolt listával.
3. Készíts programot egy leszállópálya nyilvántartási feladat megoldására.
   Segítség/magyarázat: Videó link magyar nyelven

Ajánlott kitekintő anyag (187 perc)

Leszállópálya nyilvántartási feladat **(19 perc)** – Videó link magyar nyelven

Fák számítógépes reprezentációja **(7 perc)** – Videó link magyar nyelven

Bináris keresőfák **(27 perc)** – Videó link magyar nyelven

Leszállópálya nyilvántartási algoritmus **(6 perc)** – Videó link magyar nyelven

Kiegyensúlyozott bináris keresőfák **(12 perc)** – Videó link magyar nyelven

Implementációs kérdések **(5 perc)** – Videó link magyar nyelven

Keresőfák **(20 perc)** – Gelle Kitti - link

Bináris keresőfák **(20 perc)** – Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein - ÚJ ALGORITMUSOK link 232-241. oldal

Bináris keresőfák **(52 perc)** – MIT videó link angol nyelven

Bináris keresőfák **(19 perc)** – Youtube link angol nyelven