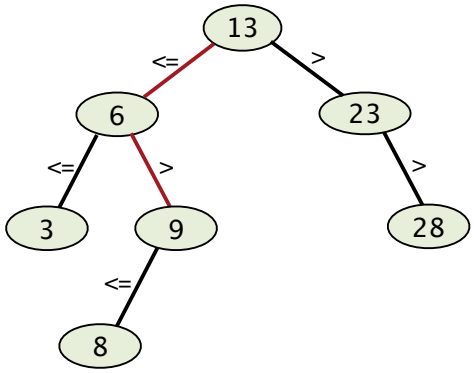
Bináris keresőfák

Az F = (M,R,Adat) absztrakt adatszerkezetet bináris keresőfának nevezzük, ha

- F bináris fa, $R = \{bal, jobb, apa\}$, $bal, jobb, apa : M \to M$,
- $Adat : M \to Elemtip$ és $Elemtip$-on értelmezett egy $\leq$ lineáris rendezési reláció,
- $(\forall x \in M)(\forall p \in F_{bal(x)})(\forall q \in F_{jobb(x)})(kulcs(p) \leq kulcs(x) \leq kulcs(q))$

```
InorderBejaras(F,M)
 if F!=Nil
 then {InorderBejaras(bal(F),M)
      M(F)
      InorderBejaras(jobb(F),M)}
```

Az InorderBejaras algoritmus a bináris keresőfa elemeit a kulcsok rendezés szerinti sorrendjében látogatja meg.



1. ábra.

Adott kulcsú elem keresése

```
KERES(F,k)
 if F=Nil or k=kulcs(F)
    then return F
 If k< kulcs(F)
    then return KERES(bal(F),k)
    else return KERES(jobb(F),k)
```

```
KERES2(F,k)
 while(F!=Nil and k!=kulcs(F))
   {if k<kulcs(F)
       then F:=bal(F)
       else F:=jobb(F)}
 return F
```

Futási idő: A fa magasságával arányos.

Minimális maximális elem keresése

```
FabanMinimum(F)
 while(bal(F)!=Nil)
      F:=bal(F)
 return F



FabanMaximum(F)
 while(jobb(F)!=Nil)
      F:=jobb(F)
 return F
```

Futási idő: A fa magasságával arányos.

Rákövetkező, megelőző elem keresése

```
FabanKovetkezo(p)
 if jobb(p)!=Nil
    then return FabanMinimum(jobb(p))
 q:=apa(p)
 while(q!=Nil and p=jobb(q))
    {p:=q
     q:=Apa(q)}
 return q



FabanMegelozo(p)
 if bal(p)!=Nil
    then return FabanMaximum(bal(p))
 q:=apa(p)
 while(q!=Nil and p=bal(q))
    {p:=q
     q:=Apa(q)}
 return q
```

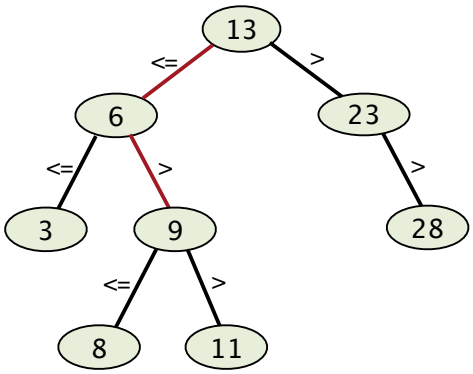
Futási idő: A fa magasságával arányos.

Beszúrás bináris keresőfába

A fát a gyökérpontja által adtuk meg.

```
Beszur(F,z)
  y:=Nil
  x:=F
  while(x!=Nil)
     {y:=x
     if kulcs(z)<kulcs(x)
        then x:=bal(x)
        else x:=jobb(x)}
  apa(z):=y
  if y=Nil
     then F:=z    //Üres volt a fa
     else {if kulcs(z)<kulcs(y)
          then bal(y):=z
          else jobb(y):=z}
```

Futási idő: A fa magasságával arányos.

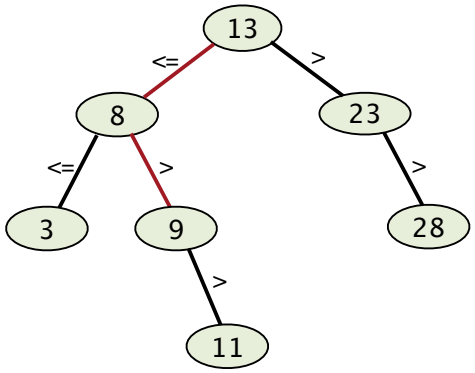


2. ábra.

Törlés bináris keresőfából

```
FabolTorol(F,z)
 If bal(z)=Nil or jobb(z)=Nil
    then y:=z
    else y:=FabanKovetkezo(z)
```

```
If bal(y)!=Nil
    then x:=bal(y)
    else x:=jobb(y)
If x!=Nil
    then apa(x):=apa(y)
If apa(y)=Nil
   then F:=x
   else {if y=bal(apa(y))
        then bal(apa(y)):=x
        else jobb(apa(y)):=x}
If y!=z
    then kulcs(z):=kulcs(y)
```



3. ábra.

Keresőfák magassága

Példa: Ha az $1,2,\ldots,n$ sorrendben szúrunk pontokat egy üres fába a magasság n lesz.

Tétel Az n csúcsból álló véletlen építésű bináris fák magassága $O(\log n)$.

Kiegyensúlyozott bináris keresőfák magassága $O(\log n)$

- piros fekete fa
- AVL fa

- önszervező bináris keresőfák (splay fák)

Optimális bináris keresőfa építése

Tegyük fel, hogy ismerjük minden a fában szereplő k_i kulcsra a keresésének gyakoriságát, ami p_i $(i = 1,\ldots,n)$. Továbbá adott minden i-re a k_i és k_{i+1} közés eső sikertelen keresések gyakorisága $q_0, q_1, \ldots, q_n$. Értelemszerűen q_0 a k_1-nél kisebb, q_n a k_n-nél nagyobb kulcsok gyakorisága.

Ha felépítünk egy F bináris keresőfát $x_1,\ldots,x_n$ pontokkal, akkor egy sikeres keresés költsége a gyakoriság szorozva a pont mélységével. A sikertelen keresésekhez hozzárendelhetünk új y_i pontokat, ahol a keresés kilép a fából.

Ekkor az F fára a várható keresési költség

$$C(F) = \sum_{i=1}^{n} p_i d_F(x_i) + \sum_{j=0}^{n} q_j d_F(y_j).$$

a cél azon fa megkonstruálása, amelyre ez az összeg minimális.

Rekurzív összefüggés

Tegyük fel, hogy van egy optimális F keresőfa a $k_1,\ldots,k_n$ kulcsokból, aminek x_r a gyökere. Ekkor a baloldali részfa F_1 a $k_1,\ldots,k_{r-1}$ kulcsokból álló bináris keresőfa, a jobboldali részfa F_2 pedig a $k_{r+1},\ldots,k_n$ csúcsokból áll. Nyilván mindkettő optimális kell legyen, különben lecserélve őket F helyett is jobbat kapnánk.

Könnyen igazolható, hogy

$$C(F) = C(F_1) + C(F_2) + \sum_{i=1}^{n} p_i + \sum_{j=0}^{n} q_j.$$

Részproblémák: Minden $0 \le i \le j \le n$-re legyen $OPT(i,j)$ az optimális bináris keresőfa költsége a $p_{i+1},\ldots,p_j$ sikeres és a $q_i,\ldots,q_j$ sikertelen keresési gyakoriságokkal.

$$OPT(i,i) = q_i$$

$$OPT(i,j) = \sum_{u=i+1}^{j} p_u + \sum_{v=i}^{j} q_v + \min_{i<r\le j}\{OPT(i,r-1) + OPT(r,j)\}.$$

A rekurzív összefüggések alapján a feladat megoldható átlós táblázatkitöltéssel.

A Verem absztrakt adattípus

Értékhalmaz: E $Verem = [a_1,\ldots,a_n : a_i \in E, i = 1,\ldots,n,]$ Műveletek: V : Verem, x : E

- $\{Igaz\}$ Letesit(V) $\{V = []\}$
- $\{V = V\}$ Megszuntet(V) $\{Igaz\}$
- $\{V = V\}$ Uresit(V) $\{V = []\}$
- $\{V = [a_1,\ldots,a_n]\}$ VeremBe(V,x) $\{V = [a_1,\ldots,a_n,x]\}$
- $\{V = [a_1,\ldots,a_n], n > 0\}$ VeremBol(V,x) $\{V = [a_1,\ldots,a_{n-1}], x = a_n\}$
- $\{V = V\}$ NemUres(V) $\{NemUres = (Pre(V) \neq [])\}$

- $\{V = [a_1, \ldots, a_n], n > 0\}$ Teteje(V,x) $\{x = a_n, V = Pre(V)\}$
- $\{V = [a_1, \ldots, a_n], n > 0\}$ Torol(V) $\{V = [a_1, \ldots, a_{n-1}]\}$

Megvalósítás: tömb, lánc, kombinált.

Megvalósítás tömbbel

A tömb az első elemétől kezdve tartalmazza a veremben szereplő értékeket, adott egy teto változó, amely megadja a verem tetején levő tömbelem indexét, továbbá egy *meret* érték, ami megadja a tömb méretét. A verem műveletek hatékonyan megvalósíthatóak.

Hátránya: előre rögzített méretű memóriát kell foglalni. Tegyük fel, hogy a vermet egy T tömbbel valósítjuk meg. Ekkor

```
Verembol(V,x)
  If teto=0
     then {print "Ures verem"
          Return False}
     Else
         {x:=T[teto]
         teto:=teto-1
         return True}


Verembe(V,x)
  If teto=meret
     Then {Print "Tele a verem"
           Return T
     Else
         {teto:=teto+1
         T[teto]:=x
         return T}
```

Ha nem akarjuk korlátozni a méret változóval a tömb méretét, akkor lehet dinamikusan növelni ezt az értéket.

```
Verembe(V,x)
  If teto=meret
      Then {Letesit(T2:Tomb)
           for i=1 to meret
               T2[i]:=T[i]
           teto:=teto+1
           T2[teto]:=x
           return T2}
      Else
           {teto:=teto+1
           T[teto]:=x
           return T}
```

Megvalósítás lánccal

A láncban az elemek egy elem.csat mutatót tartalmaznak a következő elemre és elem.adat tartalmazza a verembe rakott adatot. A láncot a legelső cellára mutató fej értékkel adjuk meg. A lánccal történő megvalósítás esetén nem kell előre lefoglalni az egész veremnek a memóriát.

```
Verembe(V,x)
   Letesit(E: lancelem)
   E.adat:=x
   E.csat:=fej
   fej:=E


Verembol(V,x)
   If fej=Nil
       Then Print "Ures Verem"
       Else
         {x:=fej.adat
         fej:=fej.csat}
```

Megvalósítás Kombinált adatszerkezettel

A kombinált adatszerkezetben az elemek tárolására adott méretű tömböket használunk, és ezen tömböket egy láncba fűzve tároljuk el.

Sor

Értékhalmaz: E $Sor = [a_1, \ldots, a_n : a_i \in E, i = 1, \ldots, n,]$ Műveletek: S: Sor, x : E

- $\{Igaz\}$ Letesit(S) $\{S = []\}$
- $\{S = S\}$ Megszuntet(S) $\{Igaz\}$
- $\{S = S\}$ Uresit(S) $\{S = []\}$
- $\{S = [a_1, \ldots, a_n\}$ SorBa(S,x) $\{S = [a_1, \ldots, a_n, x]\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ SorBol(S,x) $\{S = [a_2, \ldots, a_n], x = a_1\}$
- $\{S = [a_1, \ldots, a_n]\}$ Elemszam(S) $\{Elemszam = n\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ Elso(S,x) $\{x = a_1, S = Pre(S)\}$
- $\{S = [a_1, \ldots, a_n], n > 0\}$ Torol(S) $\{S = [a_2, \ldots, a_n]\}$

Megvalósítás: tömb, lánc kombinált.

Megvalósítás cirkuláris tömbbel

Az elemeket egy *meret* méretű tömbben tároljuk, a sor első elemének és utolsó elemének indexét megjegyezve. A létrehozott sor esetén az eleje érték 0, a vége érték 0. Az elemek a két érték között helyezkednek el (a tömb vége a tömb elejénél folytatódik).

```
Elemszam(S)
  if (elso<=utolso)
    then return utolso-elso;
   else
    return meret-(elso-utolso)
```

```
SorBa(S,x)
   if (Elemszam(S)=meret)
      Then Print "Megtelt"
      Else {utolso:=utolso+1
           If utolso>meret
              Then utolso:=utolso-meret
           T[utolso]:=x}
```

Megvalósítás lánccal

A láncban az elemek egy elem.csat mutatót tartalmaznak a következő elemre és elem.adat tartalmazza a verembe rakott adatot. A láncot a legelső cellára mutató fej értékkel és az utolsó cellára mutató vege értékkel adjuk meg. Továbbá számon kell tartani egy változóban az elmszám értéket. A lánccal történő megvalósítás esetén nem kell előre lefoglalni az egész veremnek a memóriát.

```
Sorba(S,x)
   Letesit(E: lancelem)
   E.adat:=x
   E.csat:=Nil
   vege.csat:=E
   vege:=E
   eszam:=eszam+1
```

Megvalósítás Kombinált adatszerkezettel

A kombinált adatszerkezetben az elemek tárolására adott méretű tömböket használunk, és ezen tömböket egy láncba fűzve tároljuk el. Az első tömbhöz tároljuk a sor első elemének indexét, az utolsóhoz a tömb utolsó elemének indexét.

Sorozat adattípus

Értékhalmaz: E $Sor = [a_1, \ldots, a_n : a_i \in E, i = 1, \ldots, n,]$ Műveletek: S: Sor, x : E

- $\{Igaz\}$ Letesit(S) $\{S = []\}$
- $\{S = S\}$ Megszuntet(S) $\{Igaz\}$
- $\{S = S\}$ Uresit(S) $\{S = []\}$
- $\{S = [a_1, \ldots, a_n]\}$ Elemszam(S) $\{Elemszam = n\}$
- $\{S = [a_1, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Kiolvas(S, i,x) $\{x = a_i \wedge S = Pre(S)\}$
- $\{S = [a_1, \ldots, a_i, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Modosit(S, i,x) $\{S = [a_1, \ldots, x, \ldots, a_n]\}$
- $\{S = [a_1, \ldots, a_i, a_{i+1}, \ldots, a_n] \wedge 0 \leq i \leq n\}$ Bovit(S, i,x) $\{S = [a_1, \ldots, a_i, x, a_{i+1}, \ldots, a_n]\}$
- $\{S = [a_1, \ldots, a_{i-1}, a_i, a_{i+1}, \ldots, a_n] \wedge 1 \leq i \leq n\}$ Torol(S, i) $\{S = [a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n]\}$
- $\{S = S\}$ IterKezd(S, I) $\{\}$
- $\{I = I\}$ IterAd(I,x) $\{\}$
- $\{I = I\}$ IterVege(I) $\{\}$

Megvalósítás: kiegyensúlyozott bináris keresőfával.

Tároljuk az $S = \{a_1, \ldots, a_n\}$ sorozatot egy F bináris fában úgy, hogy F inorder bejárása az S sorozatot adja. Ekkor F egy bináris keresőfa lesz a sorozat indexeire nézve.

A fa minden p pontjában tároljuk még extra információként a bal-részfájában levő pontok számánál 1-el nagyobb értéket $s(p)$. Ekkor $s(p)$ a p pontra végrehajtott inorder bejárás során p sorszáma. Ezen $s(p)$ értékek alapján a fa gyökeréből indulva megtalálható minden i-re az i-edik elem.

Bővítés esetén ha a keresőút balra halad egy p ponttól, akkor $s(p)$ értékéhez egyet kell adni. Törlés esetén, ha a keresőút balra halad egy p ponttól, akkor $s(p)$ értékéből egyet le kell vonni.

Vizsgára engedő dolgozat

Azok a hallgatók, akik csak a nagy ZH-ra vonatkozó feltételt nem teljesítik (tehát megvan a kiskérdésekből a 20 pont) az utolsó előadás időpontjában és helyén írhatnak a gyakorlat teljesítésért egy extra dolgozatot.

A dolgozatban a következő típusú kérdések közül lesznek feladatok:

- rekurzív algoritmus fejlesztése adott feladat megoldására (pl partíció feladat valamely változata)
- dinamikus programozási algoritmus kifejlesztése adott szöveges feladat megoldására
- rekurzív algoritmus fejlesztése egy elsőfiú testvér reprezentációval adott fára vonatkozó feladatra
- szöveges feladat megoldása gráfok segítségével (pl szélességi keresés)
- adott gráfalgoritmus végrehajtása

Felkészüléshez a pub-ban található gyak könyvtár slide-jait és a feladatgyújteményt javasoljuk.

Vizsgaleírás

A vizsgán elérhető maximális pontszám: 100. A vizsgán 10 kérdés lesz a honlapon található kérdéssorból. Lesz olyan kérdés, ami algoritmus végrehajtásáról szól, és olyan kérdés is lesz, amiben bizonyítás van (pl. futási idő vagy helyesség igazolása).