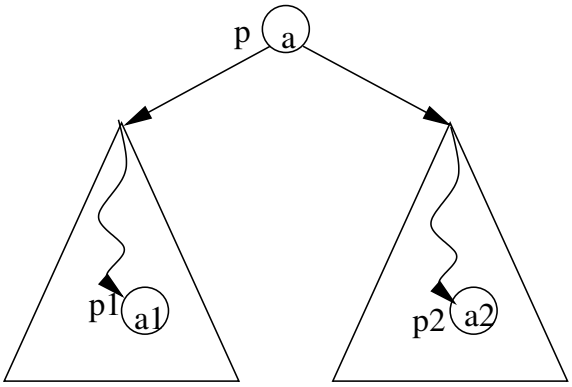
# Kiegyensúlyozott keresőfák
# AVL-fák

Szerző: Horváth Gyula

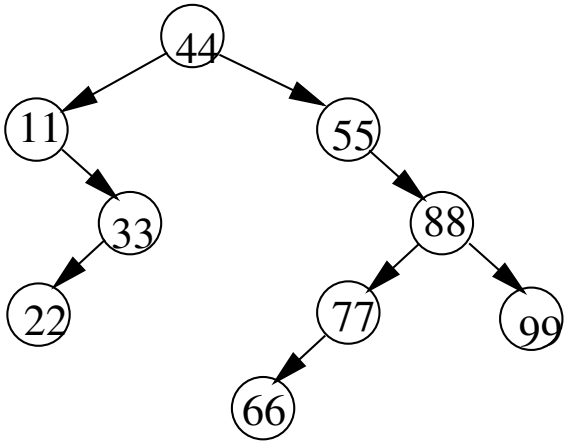
2025. október 30.

# 1. Definíció: Bináris keresőfa

Az F = block(M, R, Adat) (absztrakt) adatszerkezetet *bináris keresőfának* nevezzük, ha
1. F bináris fa, R = {bal, jobb}, bal, jobb : M → M
2. Adat : M → E és E-on értelmezett egy ≤ lineáris rendezési reláció,
3. (∀p ∈ M) (∀p1 ∈ $F_{bal(p)}$) (∀q ∈ $F_{jobb(p)}$) (Adat(p1) < Adat(p) < Adat(p2))




1. ábra. Keresőfa tulajdonság: Adat(p1) = a1 < Adat(p) = a < a2 = Adat(p2)




2. ábra. Bináris keresőfa példa

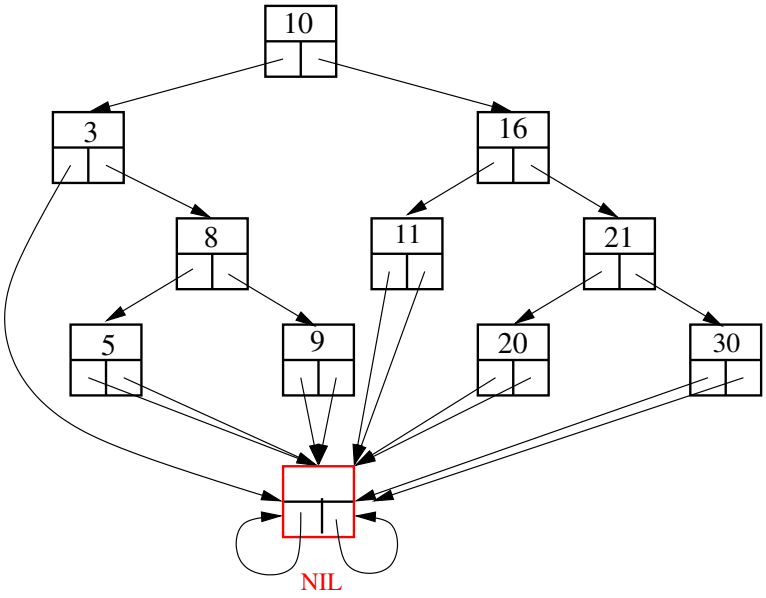
Bináris fa Inorder bejárása a fa pontjaiban lévő adatoknak azt a sorozatát állítja elő, amelyet úgy kapunk, hogy bejárjuk a p gyökerű fa bal-részfát, leírjuk a p-beli adatot, majd bejárjuk a jobb-részfát.

## 1.1. Állítás

Egy bináris fa akkor és csak akkor keresőfa, ha az Inorder bejárása rendezett sorozatot ad.

### 1.2. Bináris keresőfa adatszerkezete

Technikai okok miatt célszerű egy létező `NIL` cellával azonosítani a nemlétező szerkezeti kapcsolatot.




3. ábra. Hiányzó kapcsolat ábrázolása a `NIL` strázsa cellával.

### 1.3. Bináris keresőfa adatszerkezete

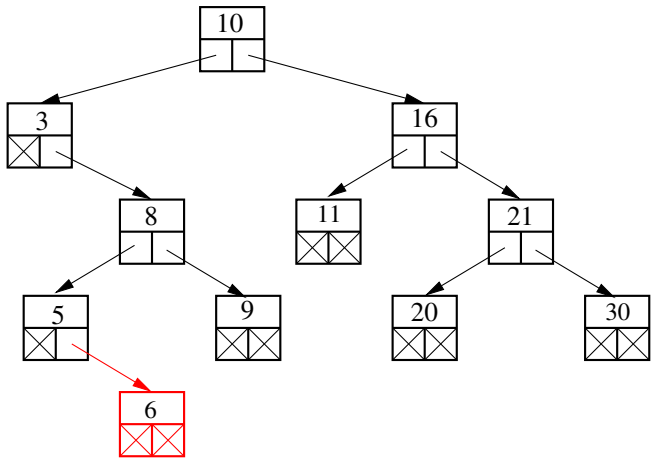
```
1  struct BinFaPont;//előre deklarálás NIL miatt
2  typedef BinFaPont* BinKerFa;
3  BinKerFa NIL;  //a strázsa cella
4
5  struct BinFaPont{
6     E adat;  // az E elemtípuson értelmezett a < rend. rel.
7     BinKerFa bal, jobb;
8     BinFaPont(){}; //ures konstrultor
9     BinFaPont(E x){//újpont konstruktor
10       adat=x;
11       bal=NIL,jobb=NIL;
12    }
13 };
```

### 1.4. A Keres művelet

```
1  BinKerFa Keres(BinKerFa p, E x){
2      while(p!=NIL){
3          if(x<p->adat)
4              p=p->bal;
5          else if(p->adat<x)
6              p=p->jobb;
7          else //x==p->adat
8              return p;
9      }
10     return p;
11 }
```

### 1.5. A Bővít művelet

Alapelv: bővítés a keresőút végére



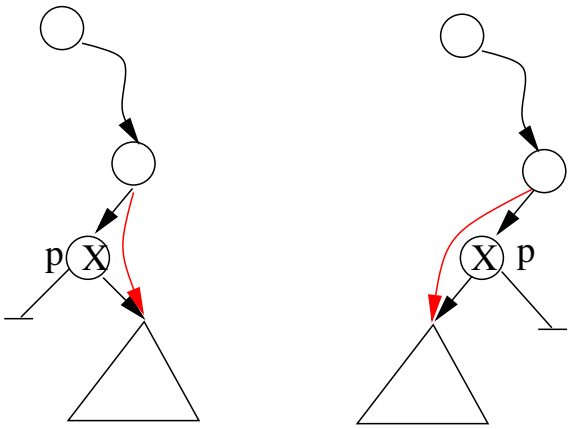

4. ábra. Bővítés a 6 adattal

### 1.6. A Bővít művelet megvalósítása

```
1  BinKerFa Bovit(BinKerFa p, E x){
2      if (p==NIL)
3          return new BinFaPont(x);
4      if (x<p->adat)
5          p->bal= Bovit(p->bal,x);
6      else if(p->adat<x)
7          p->jobb= Bovit(p->jobb,x);
8      else
9          return p;//ha nem lehet többszörös elem
10     return p;
11 }
```

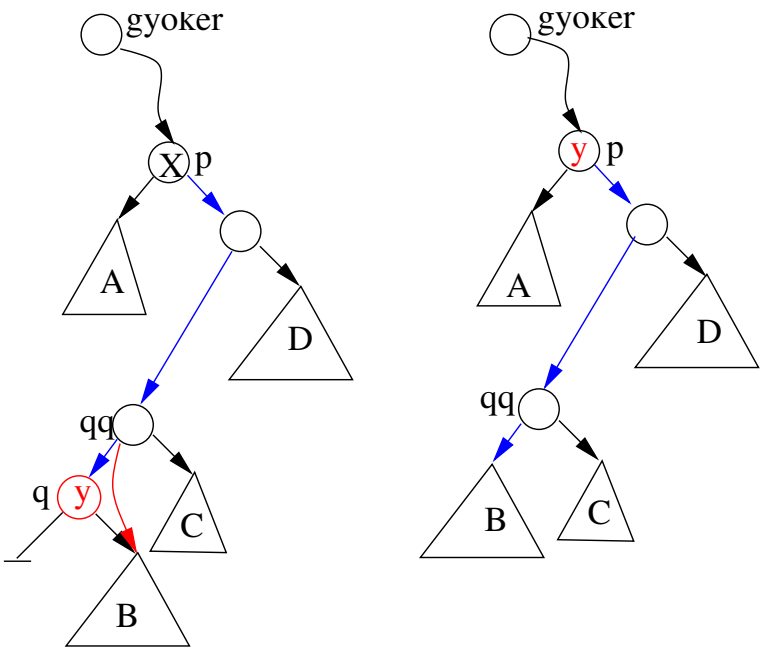
## 1.7. A Töröl művelet

1. eset: a törlendő pontnak legfeljebb egy fia van. 2. eset: a törlendő pontnak két fia van, ekkor a törlendő pontban




5. ábra. A törlendő pontnak nincs bal, vagy nincs jobb fia.

lévő adatot helyettesítsük a követő adattal, amelynek biztosan nincs bal fia.




A 2. eset

## 1.8. A Töről művelet algoritmusa

```
1  BinKerFa Torol(BinKerFa p, E x){
2      if(p==NIL) return NIL;
3      if(x<p->adat)
4          p->bal=Torol(p->bal, x);
5      else if(x>p->adat)
6          p->jobb=Torol(p->jobb, x);
7      else{//p->adat==x
8          if(p->bal==NIL) return p->jobb;
9          else if(p->jobb==NIL) return p->bal;
10         else{//p-nek 2 fia van, helyettesítés x követőjével
11             BinKerFa t=p->jobb;
12             while(t->bal!=NIL) t=t->bal;
13             p->adat=t->adat;
14             E xkovet=t->adat;
15             p->jobb=Torol(p->jobb, xkovet);
16         }
17     }
18     return p;
19 }
```

Megjegyezzük, hogy a Bővít és a Töröl műveletnek számos más algoritmusa is lehetséges (és ismert).
Itt most azért alkalmaztuk ezt a rekurzív változatot, mert a továbbiakban kihasználjuk ezen megvalósítás tulajdonságait.
Látható, hogy a három alapművelet mindegyikének a futási ideje legrosszabb (és átlagos) esetben a fa magasságával arányos.
Több olyan módszert, adatszerkezetet is kifejlesztték, amelyek alkalmazásával biztosítható, hogy a három alapművelet futási ideje a legrosszabb esetben is $\log_2(n)$-el legyen arányos, ha a fa $n$ pontot tartalmaz.
Több módszer kihasználja és alkalmazza azt, hogy a lokális forgatások megőrzik a keresőfa tulajdonságot.

# 2. AVL-fák (Adelson-Velskij, Landis, 1962)

## 2.1. Definíció

A $p \in F$ pont (magasság-)egyensúlya:
$Egys(p) = h(jobb(p)) - h(bal(p))$
Ahol $h(p)$ a $p$ pont magassága.

## 2.2. Definíció: AVL-tulajdonság

Az F bináris fa AVL-fa, ha $(\forall p \in F)(-1 \leq Egys(p) \leq 1)$

## 2.3. Tétel

Ha F AVL-fa, akkor $h(F) \leq 1.44 \cdot \lg(n+1)$; $|F| = n$

## 2.4. Bizonyítás

Legyen $N_m$ az $m$ magasságú, legkevesebb pontot tartalmazó AVL-fa pontjainak száma, azaz, $N_m \leq |F|$, ha $F$ AVL-fa és $h(F) = m$

- $N_0 = 0$, $N_1 = 1$, $N_2 = 2$
- $N_m = 1 + N_{m-2} + N_{m-1}$, ha $m > 1$

$\Rightarrow N_m + 1 = (N_{m-2} + 1) + (N_{m-1} + 1)$

Jelölje $B_i := N_i + 1$ értéket. Tehát

$B_0 := 1, B_1 := 2, \cdots, B_m = B_{m-2} + B_{m-1}$ $(m > 1)$

? $\varphi^m \leq B_m$ alsó korlátot keresünk.

Tfh. $1 = \varphi^0 \leq B_0, \varphi^1 \leq B_1$

és ha $2, \cdots, m-1$-ig áll az $\leq$

$B_m = B_{m-2} + B_{m-1} \geq \varphi^{m-2} + \varphi^{m-1} = \varphi^{m-2}(1+\varphi)$

Ha $(1+\varphi) \geq \varphi^2$, akkor $B_m \geq \varphi^{m-2}(1+\varphi) \geq \varphi^{m-2}\varphi^2 = \varphi^m$

De a $\varphi^2 = (1+\varphi)$ azaz $(\varphi^2 - \varphi - 1 = 0)$ egyenlet megoldása:

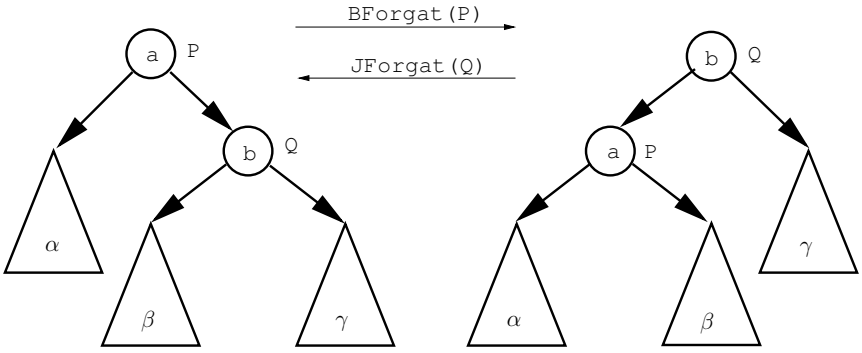
$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$

$\varphi^m \leq B_m = N_m + 1 \leq n + 1$

$m \cdot \lg\varphi \leq \lg(n+1)$

$h(F) = m \leq \frac{1}{\lg\varphi} \lg(n+1) = 1.44 \cdot \lg(n+1)$ □

Az a kérdés, hogy hogyan tartható fenn a fa AVL-egyensúlya bővítés és törlés esetén.
Az alapelv az, hogy az ismertetett standard algoritmussal elvégezzük a bővítést, illetve a törlést, majd a kereső úton visszafelé haladva helyreállítjuk lokális forgatással az egyensúlyt.




6. ábra. Az egyszerű balra/jobbra forgatás megőrzi a keresőfa tulajdonságot.

Az AVL-egyensúly betartása végett kiegészítő információként minden $p$ fapontban tároljuk a $p$-gyökerű részfa magasságának az értékét. Lokális forgatás esetén ezt aktualizálni kell!

### 2.5. AVLfa adatszerkezet

```
struct BinFaPont;//előre deklarálás NIL miatt
typedef BinFaPont* AVLFa;
AVLFa NIL;  //a nem létező szerkezeti kapcsolat ábrázolása
struct BinFaPont{
   E adat;  //az E típuson értelmezett a < rend. rel.
   int h;   //a fa magassága
   AVLFa bal, jobb;
   BinFaPont(){}; //ures konstrultor
   BinFaPont(E x){//újpont konstruktor
      adat=x;
      h=1;
      bal=NIL,jobb=NIL;
   }
};
```

### 2.6. Egyszerű balra forgatás

```
AVLFa BForgat(AVLFa p){
   AVLFa q = p->jobb;//   p          q
                     // / \        / \
   p->jobb = q->bal; //A   q =>   p   C
   q->bal=p;         //    / \    / \
                     //   B   C  A   B
   p->h = 1 + max(p->bal->h, p->jobb->h);
   q->h = 1 + max(q->bal->h, q->jobb->h);
   return q;
}
```

### 2.7. Egyszerű jobbra forgatás

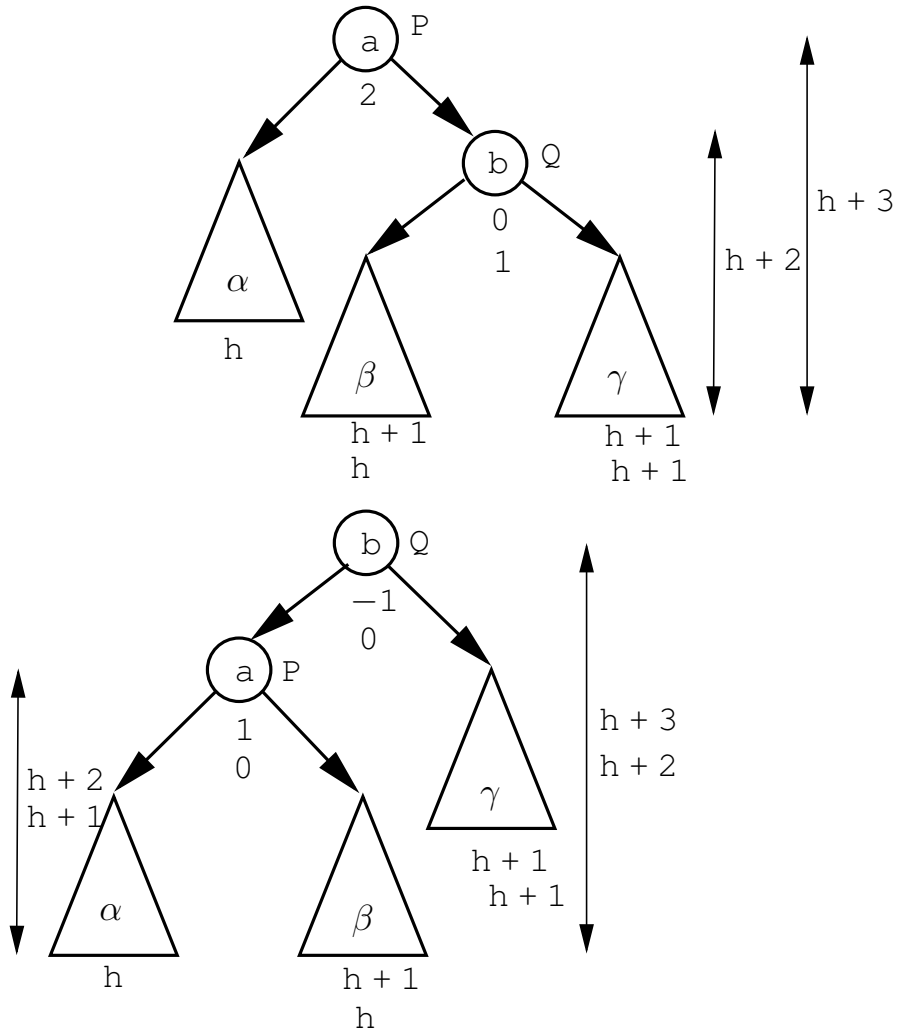
```
AVLFa JForgat(AVLFa p){
   AVLFa q = p->bal; //     p          q
                     //    / \        / \
   p->bal = q->jobb; //   q   C =>   A   p
   q->jobb = p;      //  / \            / \
                     //A   B          B   C
   p->h = 1 + max(p->bal->h, p->jobb->h);
   q->h = 1 + max(q->bal->h, q->jobb->h);
   return q;
}
```

Tegyük fel, hogy van olyan `AVLFa Egyenget(AVLFa p)` eljárásunk, amely helyreállítja a `p` pont AVL-egyensúlyát feltéve, hogy a `p`-gyökerű részfában `p`-től különböző minden pont teljesíti az AVL-egyensúly feltételt.
Ekkor a Bővítés és Törlés algoritmusa a következőképpen valósítható meg.

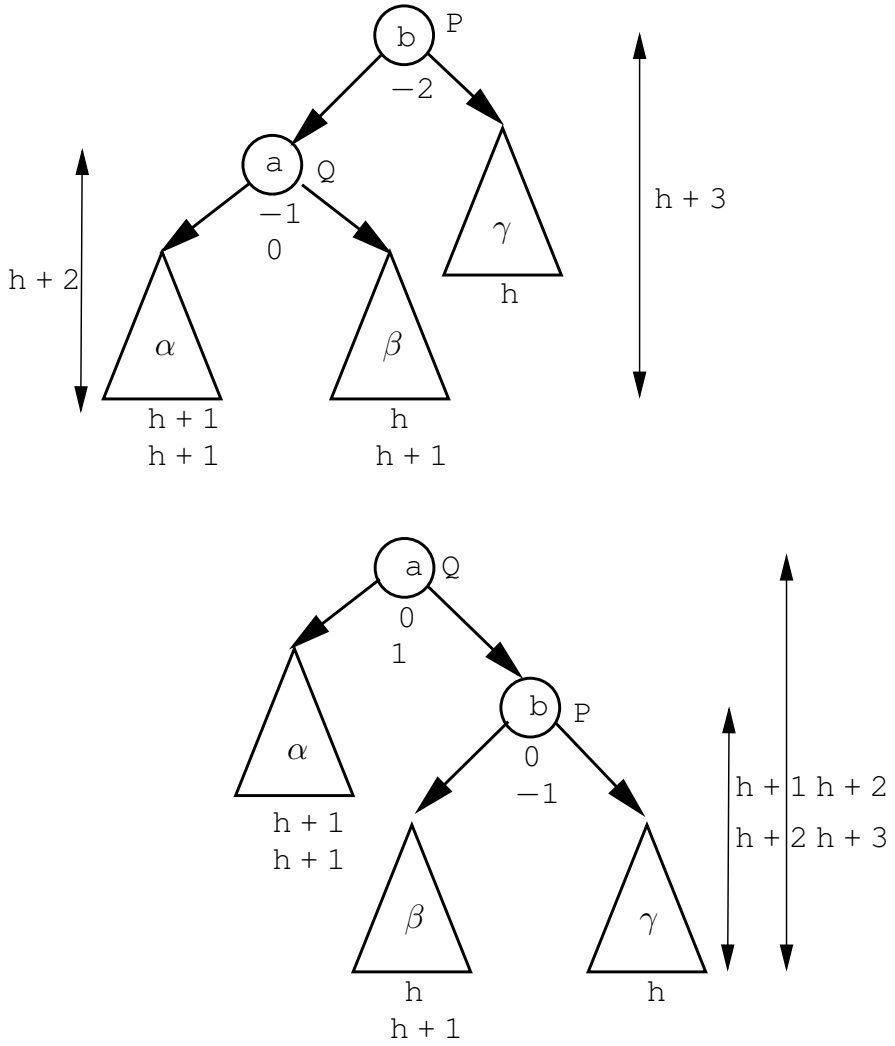
```
1  AVLFa Bovit(AVLFa p, E x){
2      if (p==NIL)
3          return new BinFaPont(x);
4      if (x<p->adat)
5          p->bal= Bovit(p->bal,x);
6      else if(p->adat<x)
7          p->jobb= Bovit(p->jobb,x);
8      else
9          return p;//ha nem lehet többszörös elem
10     p->h=1+max(p->bal->h,p->jobb->h); //h aktualizálása
11     if(abs(p->jobb->h - p->bal->h)>1) p=Egyenget(p);
12      //egyensúly helyreállítás
13     return p;
14 }
```

```
1  AVLFa Torol(AVLFa p, E x){
2      if(p==NIL) return NIL;
3      if(x<p->adat)
4          p->bal=Torol(p->bal, x);
5      else if(x>p->adat)
6          p->jobb=Torol(p->jobb, x);
7      else{//p->adat==x
8          if(p->bal==NIL) return p->jobb;
9          else if(p->jobb==NIL) return p->bal;
10         else{//2 fia van, helyettesítés x követőjével
11             AVLFa t=p->jobb;
12             while(t->bal!=NIL) t=t->bal;
13             p->adat=t->adat;
14             E xkovet=t->adat;
15             p->jobb=Torol(p->jobb, xkovet);
16         }
17     }
18     p->h=1+max(p->bal->h, p->jobb->h);
19     if(abs(p->jobb->h-p->bal->h)>1) p=Egyenget(p);
20     return p;
21 }
```

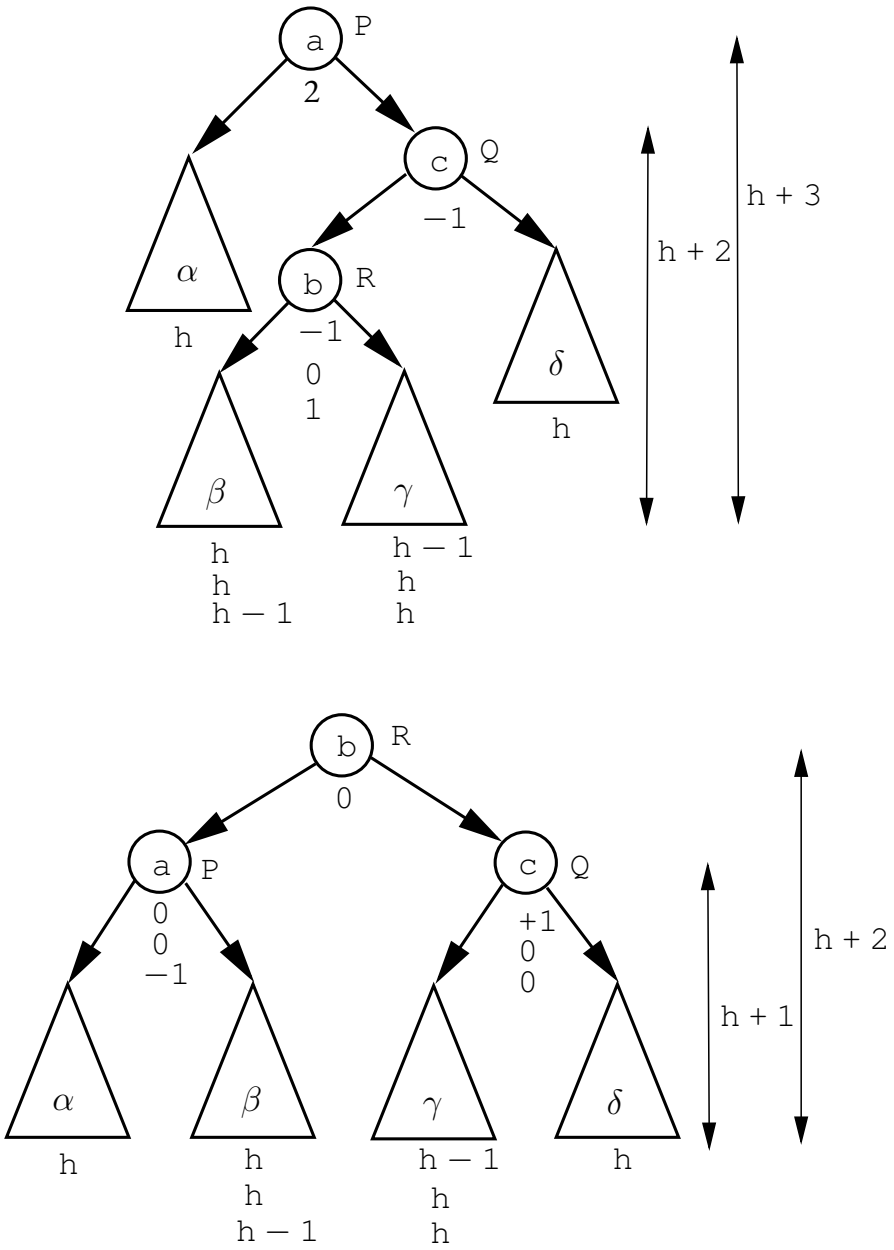
Akkor kell az AVL-egyensúlyt helyreállítani a `p` pontban, ha a művelet hatására az egyensúly értéke `Egys(p)` $< -1$ vagy `Egys(p)` $> 1$. Látható majd, hogy az aktuális részfa magassága legfeljebb 1-el nő, vagy 1-el csökken. Tehát akkor kell helyreállítani, ha `Egys(p)` $= -2$, vagy `Egys(p)` $= 2$. Mindkét esetben két alesetet kell megkülönböztetni.



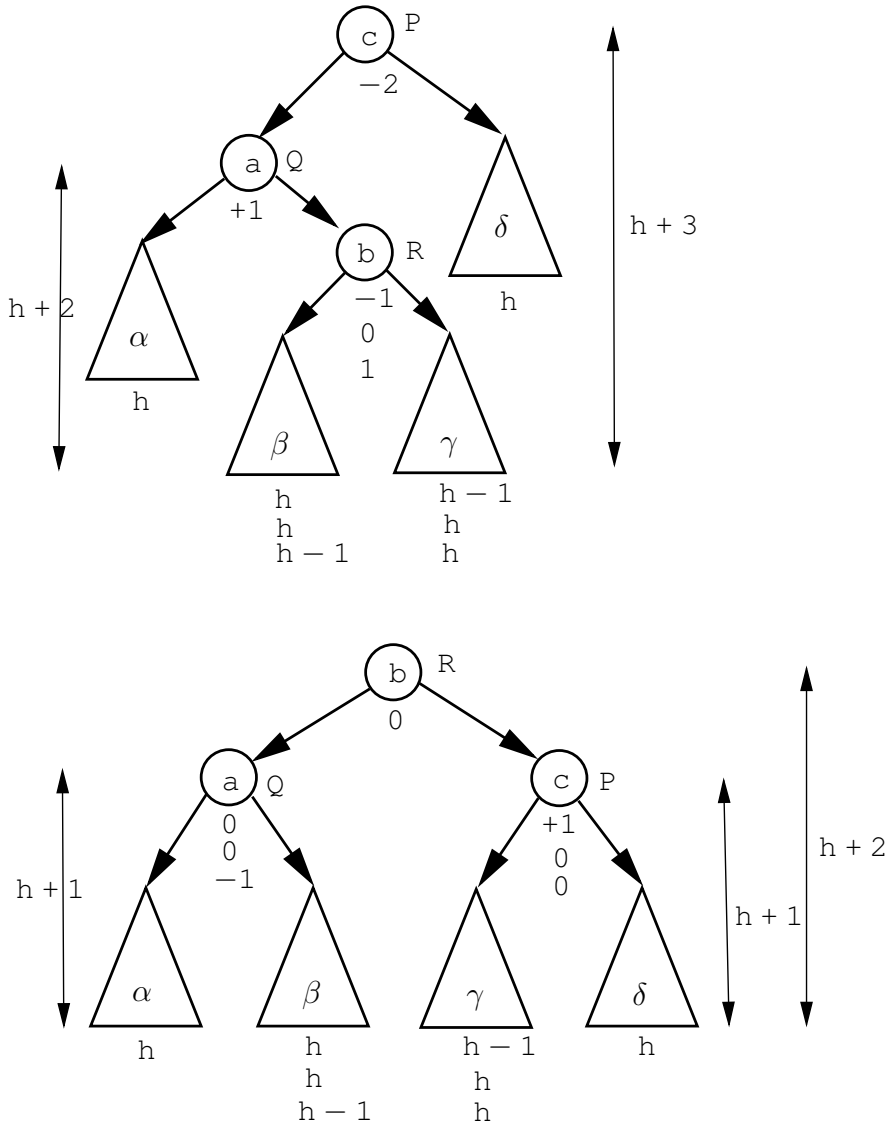

7. ábra. 1.a eset: `Egys(Q)` $\leq 0$, egyszerű balra forgatás. Az ábrán a pontok alatt a lehetséges egyensúlyi értékek, a fák alatt pedig a lehetséges magassági értékek. Az alsó fa mutatja a helyreállítás utáni helyzetet.




8. ábra. 2.a eset: `Egys(Q)` $\leq 0$, egyszerű jobbra forgatás. Az ábrán a pontok alatt a lehetséges egyensúlyi értékek, a fák alatt pedig a lehetséges magassági értékek. Az alsó fa mutatja a helyreállítás utáni helyzetet.




9. ábra. 1.b eset: `Egys(Q) = −1`, AVL egyensúly helyreállítása kettős balra forgatás. Az alsó fa mutatja a helyreállítás utáni helyzetet.




10. ábra. 2.b eset: `Egys(Q) = +1`, AVL egyensúly helyreállítása kettős jobbra forgatás. Az alsó fa mutatja a helyreállítás utáni helyzetet.

```
1  AVLFa Egyenget(AVLFa p){
2      int p_egys=p->jobb->h - p->bal->h;
3      if(p_egys==2){
4       int q_egys=(p->jobb)->jobb->h - p->jobb->bal->h;
5          if(q_egys>=0){ return BForgat(p);
6          }else{//q_egys=-1;
7              p->jobb=JForgat(p->jobb);
8              return BForgat(p);
9          }
10     }else if(p_egys==-2){
11         int q_egys=p->bal->jobb->h - p->bal->bal->h;
12         if(q_egys<=0){ return JForgat(p);
13         }else{//q_egys=+1;
14             p->bal=BForgat(p->bal);
15             return JForgat(p);
16         }
17     }
18     p->h=1+max(p->bal->h , p->jobb->h);
19     return p;
20 }
```

# 3. A Sorozat adattípus

A sorozat adattípus értékhalmaza `A = (a1, ..., an)` sorozatok halmaza, ahol az elemek mindegyike adott `E` típus, ahol `E` tetszőleges lehet.
A műveletek index szerint műveletek, az elemeket `1`-től indexelve.

## 3.1. Műveletek

**Elemszam:** A sorozat elemszámát adja.

**AdatElem(i):** A sorozat i-edik elemét adja.

**Bovit(i,x):** Az i-edik eleme után szúrja be az `x` adatelemet.

**Torol(i):** Törli a sorozat i-edik elemét.

**Modosit(i, x):** A sorozat i-edik elemét `x`-re változtatja.

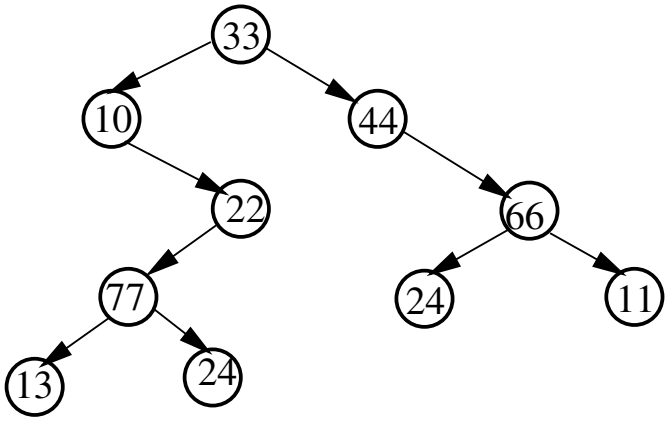
**Elejere(x):** A sorozat elejére szúrja be `x`-et.

**Vegere(x):** A sorozat végére szúrja be `x`-et.

**Bejar(Muvel()):** A sorozat minden elemére sorrendben végrehajtja a `Muvel` műveletet.

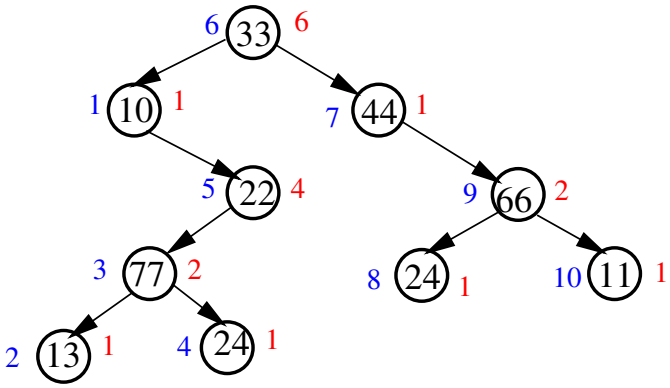
# 4. A Sorozat adattípus megvalósítása AVL fával

Reprezentáljuk (tároljuk) a sorozatot bináris fában: tetszőleges `n` pontú bináris fa azt a sorozatot ábrázolja, amely a fa Inorder bejárásával kapható.




A bináris fa az `A = (10, 13, 77, 24, 22, 33, 44, 24, 66, 11)` sorozatot reprezentálja.

# 5. A Sorozat adattípus megvalósítása AVL fával

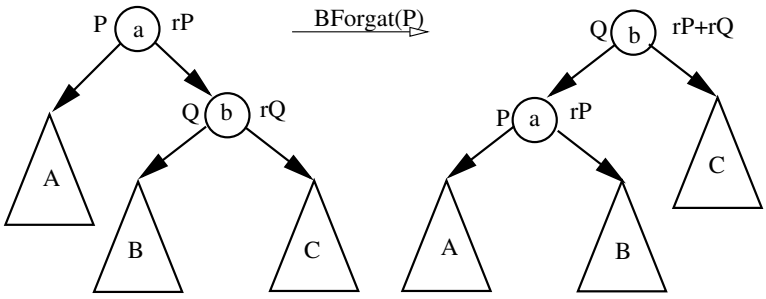



A pontok mellett bal oldalon: a sorozatbeli sorszám, jobb oldalon a bal-részfa pontjainak száma +1.

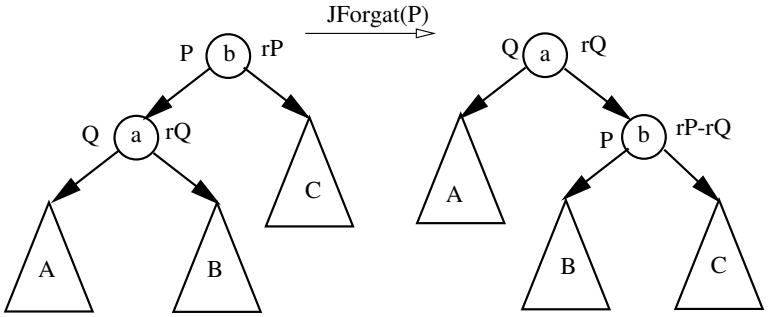
A fa minden `p` pontjában tároljuk kiegészítő adatként a `p` bal-részfájának a pontjainak száma +1 értéket. Ekkor a sorozat `i`-edik elemét tartalmazó fa pontját a következő algoritmussal tudjuk meghatározni:

```
1  AVLFa Fa_Keres(AVLFa fa, int i){
2      while(fa !=NIL && i!=fa->balresz){
3          if(i<fa->balresz)
4              fa=fa->bal;
5          else{
6              i-=(fa->balresz);
7              fa=fa->jobb;
8          }
9      }
10     return fa;
11 }
```

Tehát legrosszabb esetben is a fa magasságával arányos időben elérhető a sorozat bármelyik elemét tartalmazó fapont. Tehát, ha a fa AVL-fa, akkor `n` elemű sorozat esetén a futási idő legrosszabb esetben is `O(1.44 log(n))`
Azt kell megoldani, hogy a *balresz* kiegészítő adat aktualizálható a műveletek során a fa magasságával arányos időben. Az nyilvánvaló, hogy bővítés/törlés során aktualizálható a növelésével/csökkentésével, ha a kereső út balra megy a ponttól. Tehát azt kell megmutatni, hogy az AVL-feltétel helyreállításakor végzendő forgatások során konstans időben aktualizálható a *balresz* kiegészítő adat.




Az `rP` és `rQ` *balresz* aktualizálása balra forgatáskor.




Az `rP` és `rQ` *balresz* aktualizálása jobbra forgatáskor.

## 5.1. A Sorozat adattípus megvalósítása AVL-fával

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 typedef int E;//az adatelemek típusa: értelmezett rajta a < rendez
4
5 struct BinFaPont;//előre deklarálás NIL miatt
6 typedef BinFaPont* AVLFa;
7 AVLFa NIL;  //a nem létező szerkezeti kapcsolat ábrázolása
```

```
1  struct BinFaPont{
2     E adat;  //az elemtípus, tetszőleges
3  //kiegészítő adatok:
4     int h;   //a fa magassága
5     int balresz;//a fa pontjainak száma
6  //
7     AVLFa bal, jobb;
8     BinFaPont(){}; //ures konstrultor
9     BinFaPont(E x){//újpont konstruktor
10       adat=x;
11       h=1;
12       balresz=1;
13       bal=NIL,jobb=NIL;
14    }
15 };
16 struct Sorozat{
17    AVLFa fa;
18    int n;
19    int Elemszam(){return n;};
20    E AdatElem(int i);
21    void Bovit(int i, E x);
22    void Torol(int i);
23    void Modosit(int i, E x);
24    void Elejere(E x);
25    void Vegere(E x);
26    void Bejar( void Muvel(E));
27    Sorozat(){
28       n=0; fa=NIL;
29    }
30 };
```

```
31 //AVL-fa műveletek
32 AVLFa Fa_Keres(AVLFa fa, int i){
33     while(fa !=NIL && i!=fa->balresz){
34         if(i<fa->balresz)
35             fa=fa->bal;
36         else{
37             i-=(fa->balresz);
38             fa=fa->jobb;
39         }
40     }
41     return fa;
42 }

43 AVLFa BForgat(AVLFa p){
44     AVLFa q = p->jobb;
45
46     q->balresz+=p->balresz;
47
48     p->jobb = q->bal;
49     q->bal=p;
50
51     p->h = 1 + max(p->bal->h, p->jobb->h);
52     q->h = 1 + max(q->bal->h, q->jobb->h);
53     return q;
54 }

55 AVLFa JForgat(AVLFa p){
56     AVLFa q = p->bal;
57
58     p->balresz-=q->balresz;
59
60     p->bal = q->jobb;
61     q->jobb = p;
62
63     p->h = 1 + max(p->bal->h, p->jobb->h);
64     q->h = 1 + max(q->bal->h, q->jobb->h);
65
66     return q;
67 }
```

```
68 AVLFa Egyenget(AVLFa p){
69     int p_egys=p->jobb->h - p->bal->h;
70     if(p_egys==2){
71      int q_egys=(p->jobb)->jobb->h - p->jobb->bal->h;
72        if(q_egys>=0){ return BForgat(p);
73        }else{//q_egys=-1;
74           p->jobb=JForgat(p->jobb);
75           return BForgat(p);
76        }
77     }else if(p_egys==-2){
78        int q_egys=p->bal->jobb->h - p->bal->bal->h;
79        if(q_egys<=0){
80           return JForgat(p);
81        }else{//q_egys=+1;
82           p->bal=BForgat(p->bal);
83           return JForgat(p);
84        }
85     }
86     p->h=1+max(p->bal->h , p->jobb->h);
87     return p;
88 }

89 AVLFa Fa_Bovit(AVLFa fa, int i, E x){
90 //a sorozat i. eleme után szúrja be
91     if (fa==NIL)
92        return new BinFaPont(x);
93     if (i < fa->balresz){
94        fa->bal = Fa_Bovit(fa->bal, i, x);
95        fa->balresz++;
96     }else
97        fa->jobb = Fa_Bovit(fa->jobb, i-(fa->balresz), x);
98 //magasság aktualizálás, egyensúly helyreállítás
99     fa->h=1+max(fa->bal->h, fa->jobb->h);
100    if(abs(fa->jobb->h - fa->bal->h)>1) fa=Egyenget(fa);
101    fa->h=1+max(fa->bal->h, fa->jobb->h);
102    return fa;
103 }
```

```
104
105 AVLFa Fa_Torol(AVLFa fa, int i){
106     if(fa==NIL) return NIL;
107     if(i<fa->balresz){
108         fa->bal=Fa_Torol(fa->bal, i);
109         fa->balresz--;
110     }else if(i>fa->balresz)
111         fa->jobb=Fa_Torol(fa->jobb, i-(fa->balresz));
112     else{//fa a törlendő fafaont
113         if(fa->bal==NIL)
114             return fa->jobb;
115         else if(fa->jobb==NIL){
116             return fa->bal;
117         }else{//helyettesítés a követőjével
118             AVLFa t=fa->jobb;
119             while(t->bal!=NIL) t=t->bal;
120             fa->adat=t->adat;
121             fa->jobb=Fa_Torol(fa->jobb, 1);
122         }
123     }
124     if(fa==NIL) return NIL;
125     fa->h=1+max(fa->bal->h, fa->jobb->h);
126     if(abs(fa->jobb->h - fa->bal->h)>1) fa=Egyenget(fa);
127     fa->h=1+max(fa->bal->h, fa->jobb->h);   //magasság aktualizálás
128     return fa;
129 }

130     void Inorder(AVLFa f, void Muvel(E)){
131         if(f==NIL)  return;
132         if(f->bal!=NIL)  Inorder(f->bal, Muvel);
133         Muvel(f->adat);
134         if(f->jobb!=NIL)  Inorder(f->jobb, Muvel);
135     }
```

```
136 //A Sorozat műveletek megvalósítása
137     E Sorozat:: AdatElem(int i){
138         if(0<=i &&  i<=n){
139             AVLFa fip=Fa_Keres(fa, i);
140             return fip->adat;
141         }
142     }
143     void Sorozat:: Bovit(int i, E x){
144         if(0<=i &&  i<=n){
145             fa=Fa_Bovit(fa, i, x);
146             n++;
147         }
148     }
149     void Sorozat:: Torol(int i){
150         if(1<=i &&  i<=n){
151             fa=Fa_Torol(fa, i);
152             n--;
153         }
154     }

155     void Sorozat:: Modosit(int i, E x){
156         if(0<i &&  i<=n){
157             AVLFa fip=Fa_Keres(fa, i);
158             fip->adat=x;
159         }
160     }
161     void Sorozat:: Vegere(E x){
162         fa=Fa_Bovit(fa, n, x);
163         n++;
164     }
165     void Sorozat:: Elejere(E x){
166         fa=Fa_Bovit(fa, 0, x);
167         n++;
168     }
169     void Sorozat::Bejar( void Muvel(E)){
170         Inorder(fa, Muvel);
171         return;
172     }
```

# 6. Adatszerkezetek kibővítése

## 6.1. A stratégia lépései

1. Az alap adatszerkezet meghatározása.
2. Az alap adatszerkezetben fenntartandó kiegészítő adatok meghatározása.
3. Annak igazolása, hogy a kiegészítő adatok (hatékonyan) fenntarthatók az alap adatszerkezetet módosító műveletek során.
4. Új műveletek kifejlesztése.

# 7. Piros-fekete fák

Manapság a legtöbb programozási nyelvhez készített fejlesztői környezet piros-fekete fát alkalmaz rendezett halmaz adattípus megvalósítására.
Egy bináris fa Piros-fekete fa, ha teljesül rá a következő piros-fekete tulajdonság:

1. Minden pont színe Piros vagy Fekete.
2. A gyökér pont színe fekete.
3. Minden levél (NIL) színe fekete.
4. Minden Piros pontnak mindkét gyereke Fekete.
5. Bármely pontból bármely levélpontba vezető úton ugyanannyi Fekete pont van.

Bármely `n` pontú piros-fekete fa magassága legfeljebb `2 lg(n + 1)`.

# 8. Feladat 1: Sorrendi statisztika

Tekintsük azt az adattípust, amelynek értékhalmaza nemnegatív egész számokat tartalmazó (véges) halmazok. Az alábbi műveletek vannak értelmezve:

### Műveletek

1. S.Bovit(x): Beteszi az `S` halmazba az x számot. Ha már eleme volt, akkor a művelet hatástalan.
2. S.Torol(x): Törli az x elemet a halmazból. Ha nem volt eleme, akkor hatástalan.
3. S.Hanyadik(x): Az `x` elem hanyadik `S`-ben a rendezés szerinti sorrendben. Ha `x` nem eleme `S`-nek, akkor a $-1$ értéket adja.
4. S.Kadik(k): Az `S` halmaz rendezés szerinti `k`-adik elemét adja. Ha `k > |S|`, akkor a -1 értéket adja.
5. S.Osszeg(k): Az `S` halmaz rendezés szerinti első `k` elemének az összegét adja. Ha `k < |S|`, akkor `k` helyett `|S|` értendő.

## Bemenet/Kimenet

A standard bemenet soronként egy-egy m x számpárt tartalmaz, amely egy végrehajtandó műveletet ad meg. Az első szám a művelet sorszáma, a második pedig a művelet argumentuma. Ha a művelet `Hanyadik`, `Kadik` vagy `Osszeg`, akkor a művelet eredményét külön sorban a standard kimenetre kell írni!
A bemenetet a 0 0 számpár zárja, amit nem kell végrehajtani.

## Példa

| Bemenet | Kimenet |
|---|---|
| 1 22<br>1 12<br>1 44<br>1 33<br>2 22<br>3 33<br>4 2<br>5 2<br>0 0 | 2<br>33<br>45 |

## Korlátok

A bemenetben legfeljebb $500\,000$ művelet lehet. Minden x argumentumra teljesül, hogy $0 \leq x \leq 1\,000\,000$.
**Időlimit**: $0.2$ másodperc
**Memórialimit**: $64$ MB.

## Pontozás

A pontok 20%-át lehet szerezni olyan bemenetekre, ahol nincs sem Törlés, sem Összeg művelet.
A pontok további 40%-át lehet szerezni olyan bemenetekre, ahol nincs Összeg művelet.
A pontok további 40%-át lehet szerezni olyan bemenetekre, ahol nincs egyéb korlátozás.

Mester:Rekurzív adatszerkezetek: Sorrendi statisztika.

# 9. Feladat 2: Intervallumok

Intervallumok halmazán az alábbi műveleteket végezzük.

1. *Bovit(a,b)*: az [a,b] intervallumot hozzáveszi a halmazhoz. Feltétel: $0 < a \leq b$.
2. *Torol(a,b)*: az [a,b] intervallumot törli a halmazból, ha van ilyen eleme a halmaznak. Feltétel: az [a,b] intervallum eleme a halmaznak.
3. *MetszKeres(xa, xb, a, b)*: ha van olyan eleme a halmaznak, amelynek van közös része az [xa,xb] intevallummal, akkor a halmaznak egy ilyen [a,b] elemét adja eredményül, egyébként a a=0 és b=0 legyen a kimeneti a és b értéke.

## Könyvtári műveletek

```
1 #include "interval.h"
2 void Bovit(int a,int b)
3 void Torol(int a,int b)
4 void MetszKeres(int xa,int xb,int&a,int&b)
```

## Gyakorlás

A minta.zip fájlban letölthető egy üres minta.

## Korlátok

A számok értéke legfeljebb 2000 000 000. A függvényeket legfeljebb 100000-szer hívják.
**Időlimit**: 0.1 mp.
**Memórialimit**: 64 MiB
A programod nem írhat és nem olvashat semmilyen állományt, a standard outputra sem írhat!

Nem nehéz belátni, hogy az AVL-fák biztosítani tudják, hogy sorozatokon érték szerint és pozíció szerint is végezhessünk műveleteket hatékonyan, azaz legrosszabb esetben is $\mathcal{O}(\log n)$ lesz a futási idő.

## Gyakorló feladatok

https://open.kattis.com/contests/nma66n/problems/gcpc
https://cses.fi/problemset/task/2073
https://cses.fi/problemset/task/2072
https://cses.fi/problemset/task/1648
https://cses.fi/problemset/task/1651
https://cses.fi/problemset/task/1649
https://dmoj.ca/problem/noi05p2
https://dmoj.ca/problem/noi04p1
https://codeforces.com/contest/455/problem/D
https://codeforces.com/gym/100488/problem/L
https://codeforces.com/contest/702/problem/F