Kiegyensúlyozott keresőfák

AVL-fák

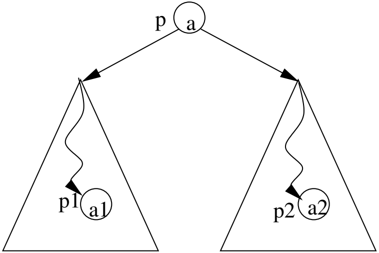
Szerző: Horváth Gyula
Előadó: Leitereg András

2025. október 19.

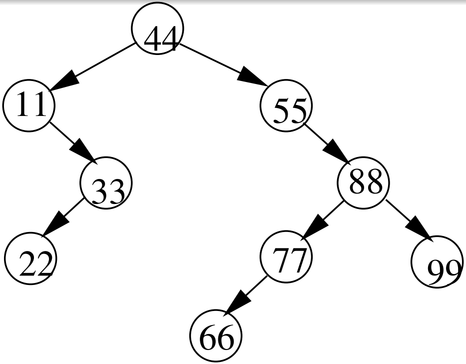
Definíció: Bináris keresőfa

Az $F = block\,(M, R, Adat)$ (absztrakt) adatszerkezetet *bináris keresőfának* nevezzük, ha

1. F bináris fa, $R = \{bal, jobb\}$, $bal, jobb : M \to M$
2. $Adat : M \to E$ és E-on értelmezett egy $\leq$ lineáris rendezési reláció,
3. $(\forall p \in M)\,(\forall p1 \in F_{bal\,(p)})\,(\forall q \in F_{jobb\,(p)})\,(Adat\,(p1) < Adat\,(p) < Adat\,(p2))$



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



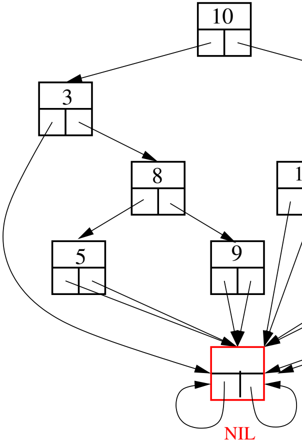
2. ábra. Bináris keresőfa példa

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.

Állítás

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

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



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

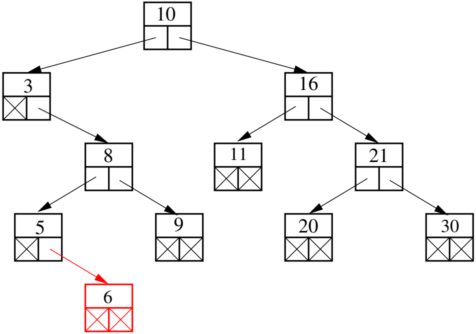
Bináris keresőfa adatszerkezete

```
struct BinFaPont;//előre deklarálás NIL miatt
typedef BinFaPont* BinKerFa;
BinKerFa NIL;  //a strázsa cella

struct BinFaPont{
   E adat;//az E elemtípuson értelmezett a < rend.
   BinKerFa bal, jobb;
   BinFaPont(){}; //ures konstrultor
   BinFaPont(E x){//újpont konstruktor
      adat=x;
      bal=NIL,jobb=NIL;
   }
};
```

```
BinKerFa Keres(BinKerFa p, E x){
    while(p!=NIL){
        if(x<p->adat)
            p=p->bal;
        else if(p->adat<x)
            p=p->jobb;
        else//x==p->adat
            return p;
    }
    return p;
}
```

A Bővít művelet

Alapelv: bővítés a keresőút végére

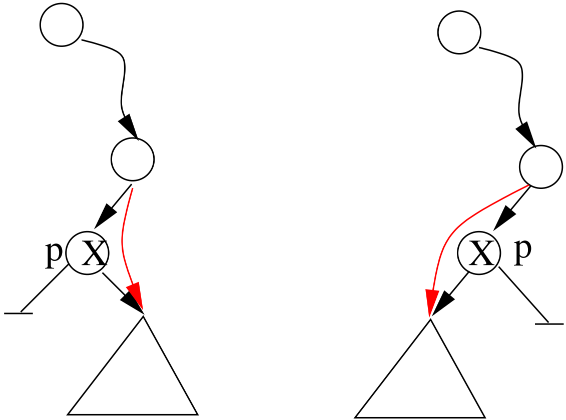


4. ábra. Bővítés a 6 adattal

A Bővít művelet megvalósítása

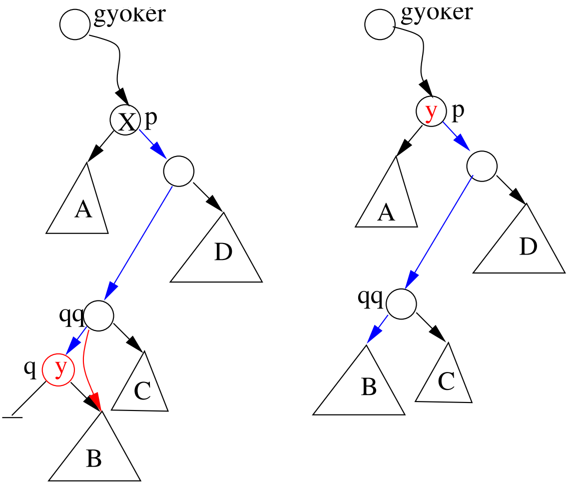
```
BinKerFa Bovit(BinKerFa p, E x){
    if (p==NIL)
        return new BinFaPont(x);
    if (x<p->adat)
        p->bal= Bovit(p->bal,x);
    else if(p->adat<x)
        p->jobb= Bovit(p->jobb,x);
    else
        return p;//ha nem lehet többszörös elem
    return p;
}
```

1. eset: a törlendő pontnak legfeljebb egy fia van.



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

2. eset: a törlendő pontnak két fia van, ekkor a törlendő pontban lévő adatot helyettesítsük a követő adattal, amelynek biztosan nincs bal fia.



A 2. eset

A Töről művelet algoritmusa

```
BinKerFa Torol(BinKerFa p, E x){
    if(p==NIL) return NIL;
    if(x<p->adat)
        p->bal=Torol(p->bal, x);
    else if(x>p->adat)
        p->jobb=Torol(p->jobb, x);
    else{//p->adat==x
        if(p->bal==NIL) return p->jobb;
        else if(p->jobb==NIL) return p->bal;
        else{//2 fia van, helyettesítés x követővel
            BinKerFa t=p->jobb;
            while(t->bal!=NIL) t=t->bal;
            p->adat=t->adat;
            E xkovet=t->adat;
            p->jobb=Torol(p->jobb, xkovet);
        }
    }
    return p;
}
```

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 kifejlesztttek, 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.

AVL-fák (Adelson-Velskij, Landis, 1962)

Definíció

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

Definíció: AVL-tulajdonság

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

Tétel

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

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:

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

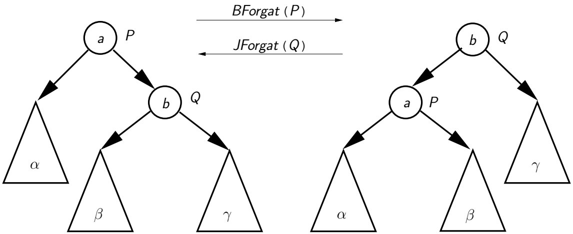
$\varphi^m \leq B_m = N_m + 1 \leq n + 1$

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

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

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!

```
struct BinFaPont;//előre deklarálás NIL miatt
typedef BinFaPont* AVLFa;
AVLFa NIL;//a nem létező 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;
    }
};
```

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;
}
```

Egyszerű jobbra forgatás

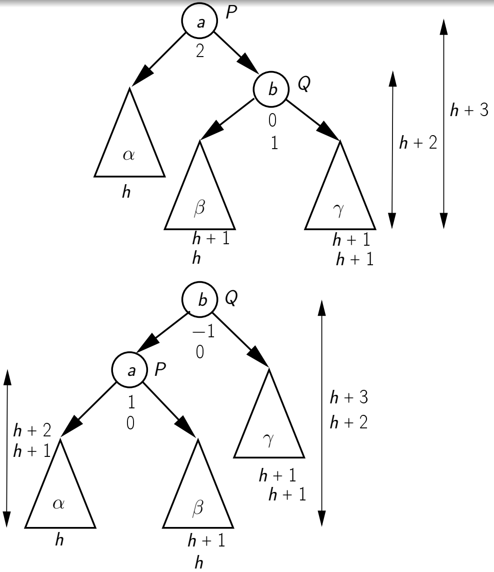
```
AVLFa JForgat(AVLFa p){
    AVLFa q = p->bal;  //       p            q
                       //      / \          / \
    p->bal = q->jobb;  //     q   C  =>    A   p
    q->jobb = p;       //    / \              / \
                       //   A   B            B   C
    p->h = 1 + max(p->bal->h,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

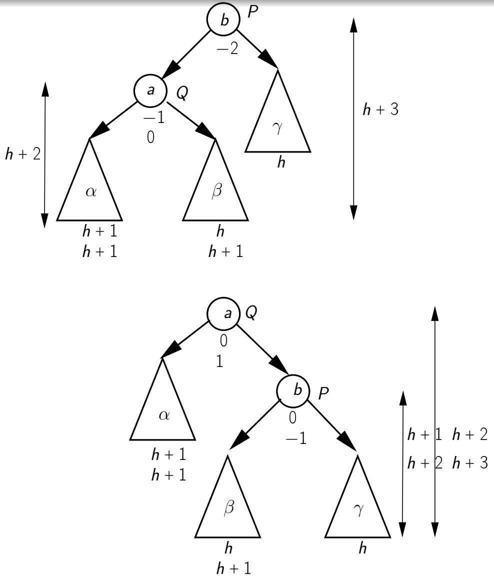
```
AVLFa Bovit(AVLFa p, E x){
    if (p==NIL)
        return new BinFaPont(x);
    if (x<p->adat)
        p->bal= Bovit(p->bal,x);
    else if(p->adat<x)
        p->jobb= Bovit(p->jobb,x);
    else
        return p;//ha nem lehet többszörös elem
    p->h=1+max(p->bal->h,p->jobb->h);//aktualizálás
    if(abs(p->jobb->h - p->bal->h)>1) p=Egyenget(p);
     //egyensúly helyreállítás
    return p;
}
```

```
AVLFa Torol(AVLFa p, E x){
    if(p==NIL) return NIL;
    if(x<p->adat)
        p->bal=Torol(p->bal, x);
    else if(x>p->adat)
        p->jobb=Torol(p->jobb, x);
    else{//p->adat==x
        if(p->bal==NIL) return p->jobb;
        else if(p->jobb==NIL) return p->bal;
        else{//2 fia van, helyettesítés x követőjével
            AVLFa t=p->jobb;
            while(t->bal!=NIL) t=t->bal;
            p->adat=t->adat;
            E xkovet=t->adat;
            p->jobb=Torol(p->jobb, xkovet);
        }
    }
    p->h=1+max(p->bal->h, p->jobb->h);
    if(abs(p->jobb->h-p->bal->h)>1) p=Egyenget(p);
    return p;
}
```

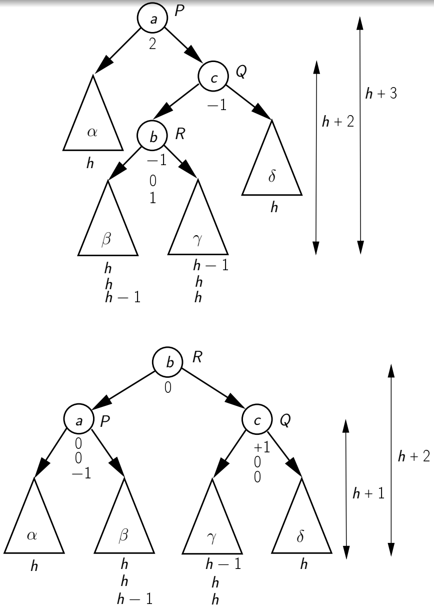
Akkor kell az AVL-egyensúlyt helyreállítani a p pontban, ha a művelet hatására az egyensúly értéke $Egys(p) < -1$ vagy $Egys(p) > 1$. Látható majd, hogy az aktuális részfa magassága legfeljebb 1-el nő, vagy 1-el csökken. Tehát akkor kell helyreállítani, ha $Egys(p) = -2$, vagy $Egys(p) = 2$. Mindkét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



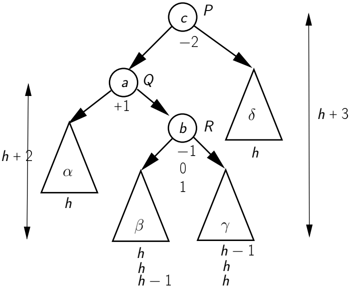
7. ábra. 1.a eset: $Egys\,(Q) \leq 0$, egyszerű balra forgatás. Az ábrán a pontok alatt a lehetséges egyensúlyi értékek, a fák alatt pedig a lehetséges magassági értékek. Az alsó fa mutatja a helyreállítás utáni helyzetet.

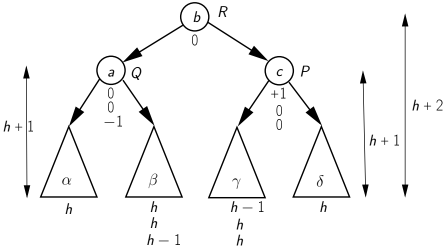


8. ábra. 2.a eset: *Egys* (*Q*) ≤ 0, egyszerű jobbra forgatás. Az ábrán a pontok alatt a lehetséges egyensúlyi értékek, a fák alatt pedig a lehetséges magassági értékek. Az alsó fa mutatja a helyreállítás utáni helyzetet.



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.

```
AVLFa Egyenget(AVLFa p){
    int p_egys=p->jobb->h - p->bal->h;
    if(p_egys==2){
     int q_egys=(p->jobb)->jobb->h - p->jobb->bal->h
       if(q_egys>=0){ return BForgat(p);
       }else{//q_egys=-1;
          p->jobb=JForgat(p->jobb);
          return BForgat(p);
       }
    }else if(p_egys==-2){
       int q_egys=p->bal->jobb->h - p->bal->bal->h;
       if(q_egys<=0){ return JForgat(p);
       }else{//q_egys=+1;
          p->bal=BForgat(p->bal);
          return JForgat(p);
       }
    }
    p->h=1+max(p->bal->h , p->jobb->h);
    return p;
}
```

A Sorozat adattípus

A sorozat adattípus értékhalmaza $A = (a_1, \ldots, a_n)$ sorozatok halmaza, ahol az elemek mindegyike adott E típus, ahol E tetszőleges lehet.
A műveletek index szerint műveletek, az elemeket 1-től indexelve.

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.

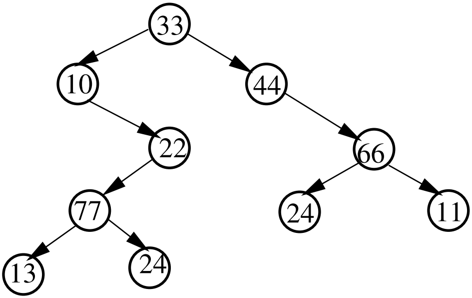
Elejere(x): A sorozat elejére szúrja be x-et.

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

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

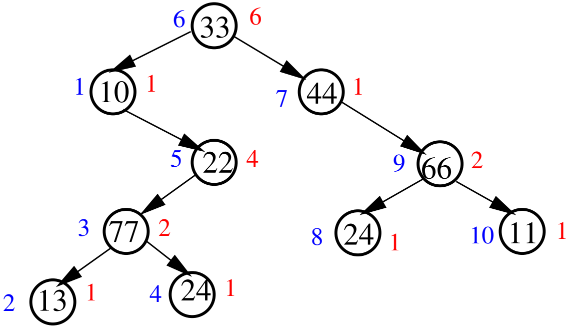
A Sorozat adattípus megvalósítása AVL fával

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 $A = \langle 10, 13, 77, 24, 22, 33, 44, 24, 66, 11 \rangle$ sorozatot reprezentálja.

A Sorozat adattípus megvalósítása AVL fával



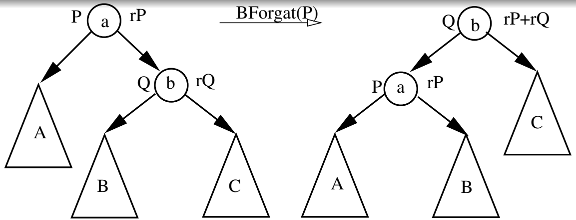
A pontok mellett bal oldalon: a sorozatbeli sorszám, jobb oldalon a bal-részfa pontjainak száma +1.

A Sorozat adattípus megvalósítása AVL fával

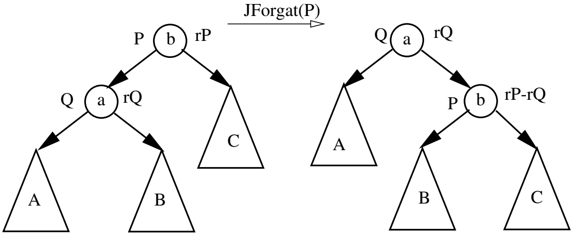
A fa minden *p* pontjában tároljuk kiegészítő adatként a *p* bal-részfájának a pontjainak száma +1 értéket. Ekkor a sorozat *i*-edik elemét tartalmazó fa pontját a következő algoritmussal tudjuk meghatározni:

```
AVLFa Fa_Keres(AVLFa fa, int i){
    while(fa !=NIL && i!=fa->balresz){
        if(i<fa->balresz)
            fa=fa->bal;
        else{
            i-=(fa->balresz);
            fa=fa->jobb;
        }
    }
    return fa;
}
```

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.

A Sorozat adattípus megvalósítása AVL-fával

```
#include <bits/stdc++.h>
using namespace std;
typedef int E;
//az adatelemek típusa: értelmezett rajta a
//  < rendezési reláció

struct BinFaPont;//előre deklarálás NIL miatt
typedef BinFaPont* AVLFa;
AVLFa NIL;//a nem létező kapcsolat ábrázolása
```

```
struct BinFaPont{
    E adat;  //az elemtípus, tetszőleges
//kiegészítő adatok:
    int h;   //a fa magassága
    int balresz;//a fa pontjainak száma
//
    AVLFa bal, jobb;
    BinFaPont(){}; //ures konstrultor
    BinFaPont(E x){//újpont konstruktor
        adat=x;
        h=1;
        balresz=1;
        bal=NIL,jobb=NIL;
    }
};
```

```
struct Sorozat{
    AVLFa fa;
    int n;
    int Elemszam(){return n;};
    E AdatElem(int i);
    void Bovit(int i, E x);
    void Torol(int i);
    void Modosit(int i, E x);
    void Elejere(E x);
    void Vegere(E x);
    void Bejar( void Muvel(E));
    Sorozat(){
        n=0; fa=NIL;
    }
};
```

```
//AVL-fa műveletek
AVLFa Fa_Keres(AVLFa fa, int i){
    while(fa !=NIL && i!=fa->balresz){
        if(i<fa->balresz)
            fa=fa->bal;
        else{
            i-=(fa->balresz);
            fa=fa->jobb;
        }
    }
    return fa;
}
```

```
AVLFa BForgat(AVLFa p){
    AVLFa q = p->jobb;

    q->balresz+=p->balresz;

    p->jobb = q->bal;
    q->bal=p;

    p->h = 1 + max(p->bal->h, p->jobb->h);
    q->h = 1 + max(q->bal->h, q->jobb->h);
    return q;
}
```

```
AVLFa JForgat(AVLFa p){
    AVLFa q = p->bal;

    p->balresz -=q->balresz;

    p->bal = q->jobb;
    q->jobb = p;

    p->h = 1 + max(p->bal->h, p->jobb->h);
    q->h = 1 + max(q->bal->h, q->jobb->h);

    return q;
}
```

```
AVLFa Egyenget(AVLFa p){
    int p_egys=p->jobb->h - p->bal->h;
    if(p_egys==2){
     int q_egys=(p->jobb)->jobb->h - p->jobb->bal->h
        if(q_egys>=0){ return BForgat(p);
        }else{//q_egys=-1;
            p->jobb=JForgat(p->jobb);
            return BForgat(p);
        }
    }else if(p_egys==-2){
        int q_egys=p->bal->jobb->h - p->bal->bal->h;
        if(q_egys<=0){
            return JForgat(p);
        }else{//q_egys=+1;
            p->bal=BForgat(p->bal);
            return JForgat(p);
        }
    }
    p->h=1+max(p->bal->h , p->jobb->h);
    return p;
}
```

```
AVLFa Fa_Bovit(AVLFa fa, int i, E x){
//a sorozat i. eleme után szúrja be
    if (fa==NIL)
        return new BinFaPont(x);
    if (i < fa->balresz){
        fa->bal = Fa_Bovit(fa->bal, i, x);
        fa->balresz++;
    }else
        fa->jobb = Fa_Bovit(fa->jobb, i-(fa->balresz)
//magasság aktualizálás, egyensúly helyreállítás
    fa->h=1+max(fa->bal->h, fa->jobb->h);
    if(abs(fa->jobb->h - fa->bal->h)>1) fa=Egyenget(
    fa->h=1+max(fa->bal->h, fa->jobb->h);
    return fa;
}
```

```
AVLFa Fa_Torol(AVLFa fa, int i){
    if(fa==NIL) return NIL;
    if(i<fa->balresz){
        fa->bal=Fa_Torol(fa->bal, i);
        fa->balresz--;
    }else if(i>fa->balresz)
        fa->jobb=Fa_Torol(fa->jobb, i-(fa->balresz));
    else{//fa a törlendő fafaont
        if(fa->bal==NIL)
            return fa->jobb;
        else if(fa->jobb==NIL){
            return fa->bal;
        }else{//helyettesítés a követőjével
            AVLFa t=fa->jobb;
            while(t->bal!=NIL) t=t->bal;
            fa->adat=t->adat;
            fa->jobb=Fa_Torol(fa->jobb, 1);
        }
    }
    if(fa==NIL) return NIL;
```

```
    fa->h=1+max(fa->bal->h, fa->jobb->h);
    if(abs(fa->jobb->h - fa->bal->h)>1)
        fa=Egyenget(fa);
    fa->h=1+max(fa->bal->h, fa->jobb->h);
    //magasság aktualizálás
    return fa;
}
```

```
void Inorder(AVLFa f, void Muvel(E)){
    if(f==NIL) return;
    if(f->bal!=NIL) Inorder(f->bal, Muvel);
    Muvel(f->adat);
    if(f->jobb!=NIL) Inorder(f->jobb, Muvel);
}
```

```
//A Sorozat műveletek megvalósítása
    E Sorozat:: AdatElem(int i){
        if(0<=i && i<=n){
            AVLFa fip=Fa_Keres(fa, i);
            return fip->adat;
        }
    }
    void Sorozat:: Bovit(int i, E x){
        if(0<=i && i<=n){
            fa=Fa_Bovit(fa, i, x);
            n++;
        }
    }
    void Sorozat:: Torol(int i){
        if(1<=i && i<=n){
            fa=Fa_Torol(fa, i);
            n--;
        }
    }
```

```
void Sorozat:: Modosit(int i, E x){
    if(0<i && i<=n){
        AVLFa fip=Fa_Keres(fa, i);
        fip->adat=x;
    }
}
void Sorozat:: Vegere(E x){
    fa=Fa_Bovit(fa, n, x);
    n++;
}
void Sorozat:: Elejere(E x){
    fa=Fa_Bovit(fa, 0, x);
    n++;
}
void Sorozat::Bejar( void Muvel(E)){
    Inorder(fa, Muvel);
    return;
}
```

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.

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 $2\,\mathtt{lg}(n+1)$.

Feladat 1:Mester:Rekurzív adatszerkezetek: 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	2
1 12	33
1 44	45
1 33	
2 22	
3 33	
4 2	
5 2	
0 0	

Korlátok

A bemenetben legfeljebb 100 000 művelet lehet.
Minden x argumentumra teljesül, hogy $0 \leq x \leq 1\,000\,000$.
Időlimit: 0.05 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.

Zárt intervallumok halmazán az alábbi műveleteket végezzük. Kezdetben a halmaz üres.

1. Bovit(a,b): az [a,b] intervallumot hozzáveszi a halmazhoz. Feltétel: $0 < a \leq b$. Ha már eleme a halmaznak az [a,b] intervallum, akkor a művelet hatástalan.
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

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

Gyakorlás

A minta.zip fájlban letölthető egy üres minta.

Korlátok

A számok értéke legfeljebb 2 000 000 000. A függvényeket legfeljebb 100 000-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.

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