Algoritmusok és adatszerkezetek II.

Horváth Gyula
Szegedi Tudományegyetem
Természettudományi és Informatikai Kar
horvath@inf.u-szeged.hu

5. Vágható-egyesíthető Halmaz adattípus megvalósítása önszervező bináris keresőfával

Értékhalmaz: $VEHalmaz = \{\{a_1,\ldots,a_n\} : a_i \in E\}$
Műveletek: RHalmaz műveletek +

$$H, H_1, H_2 : VEHalmaz, k : E$$

$$\begin{array}{rll} \{H = H\} & Vag(H,k,H_1,H_2) & \{H = \emptyset, H_1 = \{x \in Pre(H) : x < k\} \\ & & H_2 = \{x \in Pre(H) : x \geq k\}\} \\ \{\max(H_1) \leq \min(H_2)\} & Egyesit(H_1,H_2,H) & \{H_1 = \emptyset, H_2 = \emptyset, H = Pre(H_1) \cup Pre(H_1)\} \end{array}$$

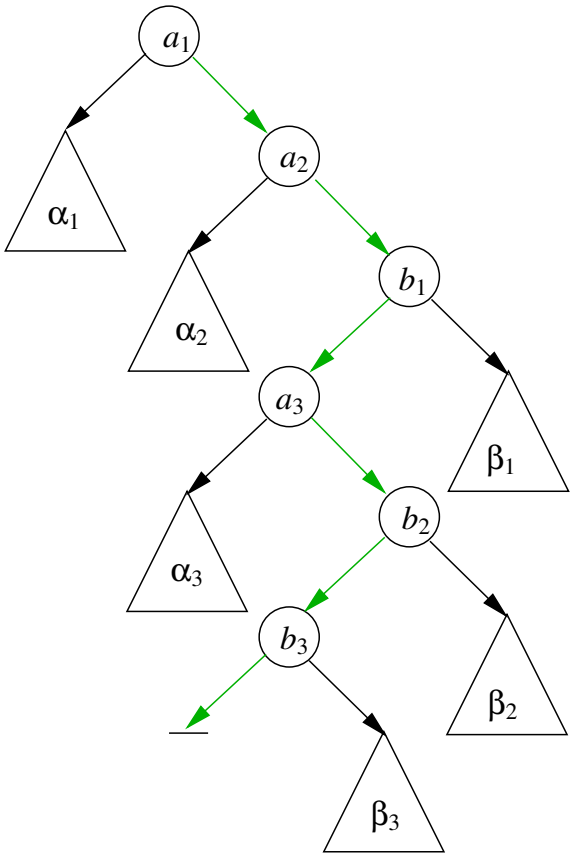
```
public interface VEHalmaz<E extends Comparable<E>>
   extends RHalmaz<E>{
      public VEHalmaz<E> Vag(E k);
      public void Egyesit(VEHalmaz<E> H2);
}
```

A vágás művelet megvalósítható egy menetben felülről lefelé haladva a következő transzformációs lépésekkel. Minden lépésben a bemeneti fa egy részfáját átkapcsoljuk vagy az F_1 fa jobb sarkához, vagy az F_2 fa bal sarkához. A lépéseket addig kell ismételni, amíg a bemeneti fa elfogy. Transzformációs invariáns.
A VAG művelet megvalósítása transzformációs lépések sorozatából áll. Minden lépésben három fa vesz részt (amelyek közül egy lépésben csak kettő változik), $\langle F_1, F, F_2\rangle$, ahol F a még vágandó fa, F_1 a kiindulási fából eddig kivágott és a K kulcsnál $<$ pontokat tartalmazó keresőfa, F_2 pedig a kiindulási fából eddig kivágott és a K kulcsnál $\geq$ pontokat tartalmazó keresőfa. A kezdő hármas: $\langle \perp, F, \perp\rangle$.
Az algoritmus helyességének bizonyítását transzformációs invariáns alkalmazásával végezzük. A transzformációs invariáns olyan tulajdonság (logikai kifejezés), amely ha teljesül egy $\langle F_1, F, F_2\rangle \mapsto \langle \overline{F_1}, \overline{F}, \overline{F_2}\rangle$ transzformációs lépés előtt az $\langle F_1, F, F_2\rangle$ hármasra, akkor a keletkező $\langle \overline{F_1}, \overline{F}, \overline{F_2}\rangle$ fa-hármasra ismét teljesülni fog.
Transzformációs invariáns a $T1$ és $T2$ vágási transzformációkra:

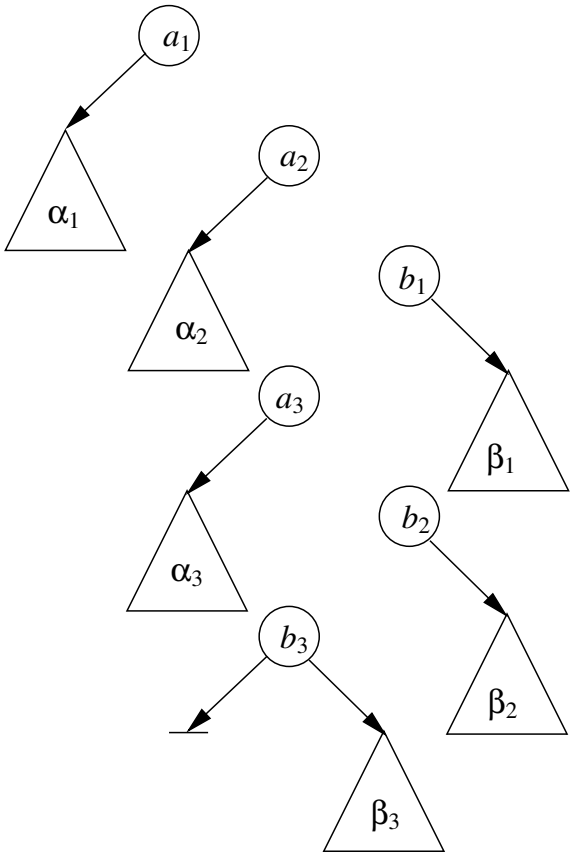
1. Az F, F_1 és F_2 fa bináris keresőfa
2. $\max(F_1) < K \leq \min(F_2)$ ha $F_1 \neq \perp$ és $F_2 \neq \perp$
3. $\max(F_1) \leq \min(F)$ ha $F_1 \neq \perp$
4. $\max(F) \leq \min(F_2)$ ha $F_2 \neq \perp$

Megvalósítás

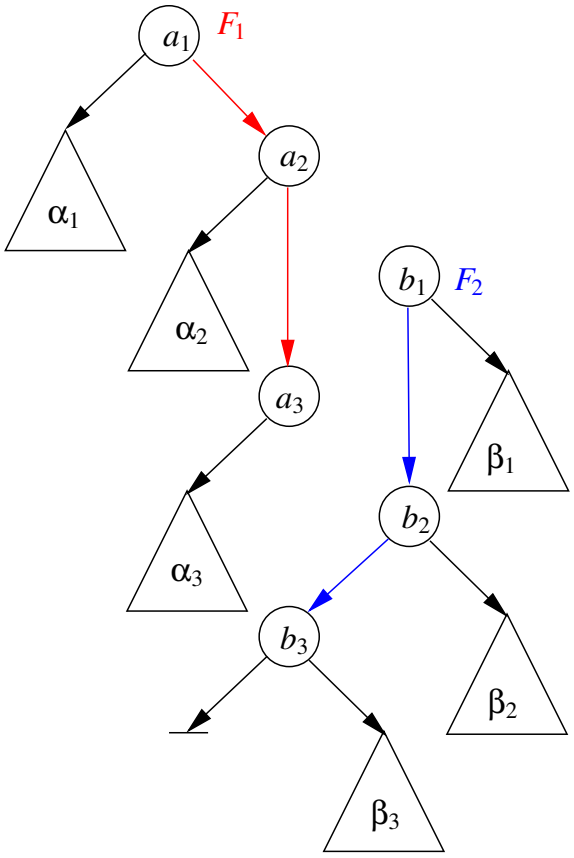
```
public BinKerFa<E> BKFaVag(E k){
   BinKerFaPont<E> Fej;
   BinFaPont<E> BJSarok, JBSarok, p;
```



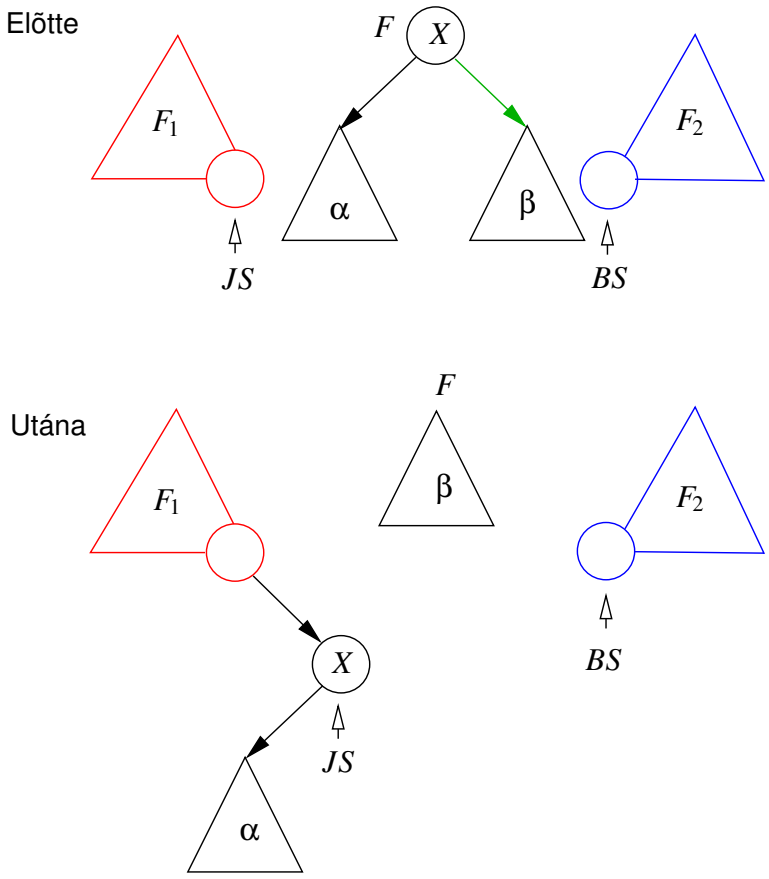
1. ábra. A K kulcshoz tartozó bővítőút.



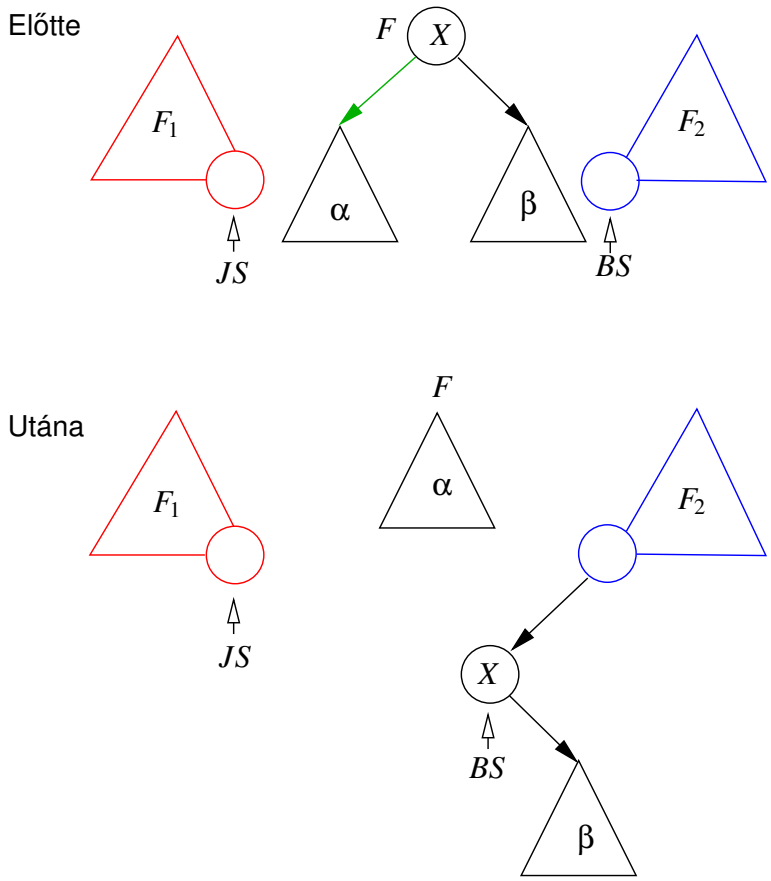
2. ábra. Az F fa K-nál kisebb gyökerű és K-nál nem kisebb gyökerű részfákra bontása.



3. ábra. A részfák összekapcsolása F_1 és F_2 fává.



4. ábra. A $T1$ vágási transzformáció; feltétel: $X < K$



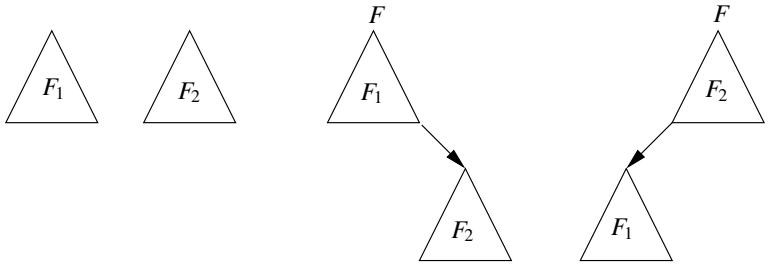
5. ábra. A $T2$ vágási transzformáció; feltétel: $K \leq X$

```
   Fej=new BinKerFaPont<E>();
   p=gyoker; BJSarok=JBSarok=Fej;
   while (p!=null){
      if (k.compareTo(p.elem)<=0 ){
         JBSarok.bal=p;    //(B, X<-P, J) --> (B, a, J)
         p.apa=JBSarok;    //   / \                /
         JBSarok=p;        //  a   b             X
         p=p.bal;          //                     \
      }else{               //                      b
         BJSarok.jobb=p;   //(B, X<-P, J) --> (B, b, J)
         p.apa=BJSarok;    //   / \                \
         BJSarok=p;        //  a   b               X
         p=p.jobb;         //                      /
      }                    //                     a
   }//while
   BJSarok.jobb=null; JBSarok.bal=null;
   gyoker=Fej.jobb;
   if (gyoker!=null) gyoker.apa=null;
   if (Fej.bal!=null) Fej.bal.apa=null;
   return new BinKerFaA<E>(Fej.bal);
}
```

A BKFAVAG eljárás futási ideje legrosszabb esetben $O(h(F))$. Azonban a vágás során nem tartható fenn sem az AVL, sem a piros-fekete kiegyensúlyozottság a fa magasságával arányos időben.

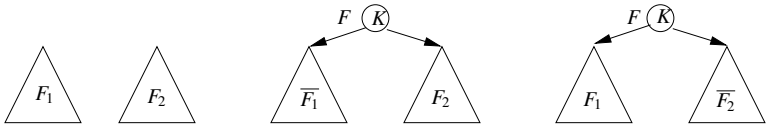
5.1. Egyesítés

A $\max(F_1) \leq \min(F_2)$ feltétel miatt a két fa egyszerűen összekapcsolható, akár F_2-t F_1 jobb-sarkához, akár F_1-et F_2 bal-sarkához kapcsolva. Kevésbé elfajuló fát kapunk, ha először eltávolítjuk F_1-ből a legnagyobb elemet tartalmazó pontot, majd az így kapott



6. ábra. Két bináris keresőfa egyszerű egyesítése

$\overline{F_1}$ fát az eltávolított pont bal, és az F_2 fát jobb fiaként kapcsoljuk be (vagy fordítva).



7. ábra. Két bináris keresőfa egyesítése

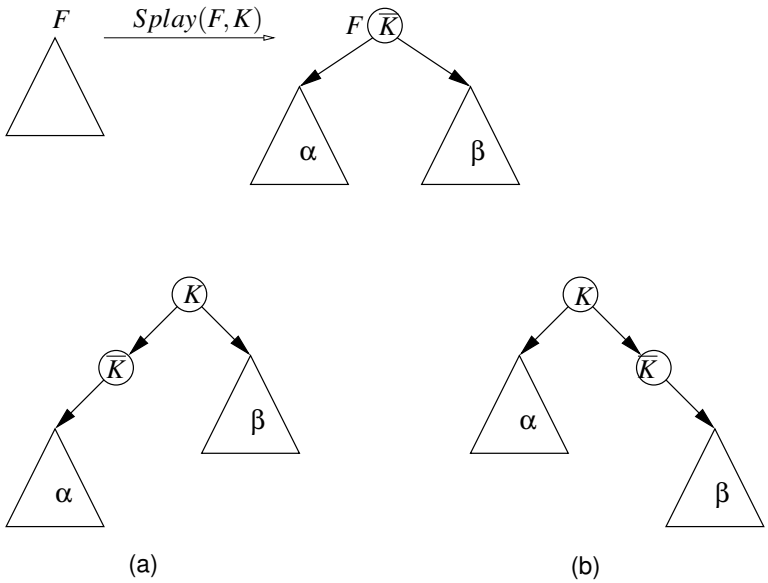
5.2. Önszervező bináris keresőfák

Tegyük fel, hogy van olyan $Splay(F,K)$ műveletünk, amely átalakítja az F bináris keresőfát úgy, hogy a K kulcsú elem kerül a gyökérbe, ha van F-ben K kulcsú elem, egyébként olyan $\overline{K}$ kulcsú elem kerül a gyökérbe, amely vagy követője, vagy előzője

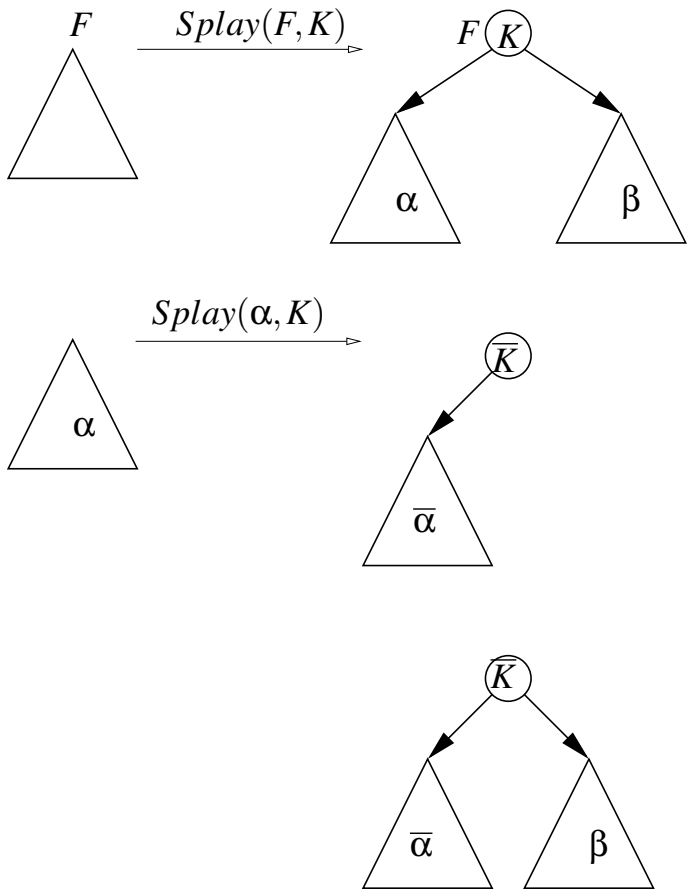
K-nak. (Tehát SPLAY(F,K) hatására a K-keresőút végén lévő pont kerül a gyökérbe.)
Tehát $Splay(F,K)$ végrehajtása után teljesül a **Splay** tulajdonság:

1. F bináris keresőfa
2. - $Adat(F) = K$, ha $K \in Pre(F)$ vagy
   - $Adat(F) = Max\{Adat(x) : x \in Pre(F) \wedge Adat(x) < K\}$, vagy
   - $Adat(F) = Min\{Adat(x) : x \in Pre(F) \wedge Adat(x) > K\}$ ha $K \notin Pre(F)$

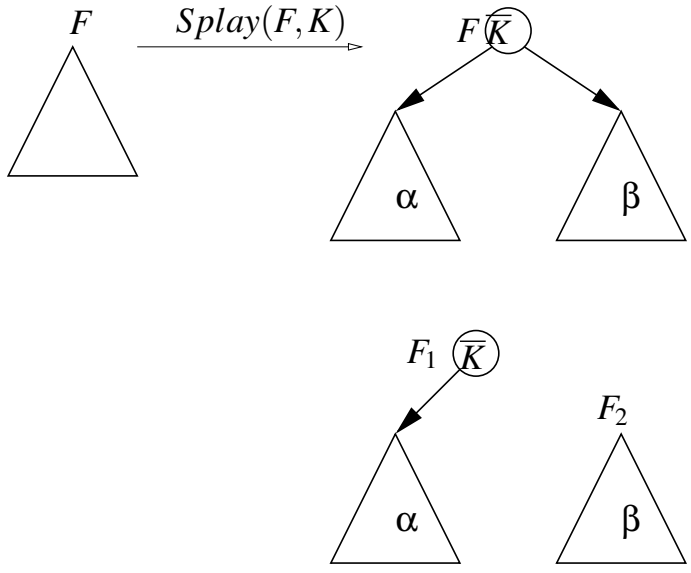
Ilyen SPLAY művelettel megvalósítható a KERES, BOVIT, TOROL, VAG és EGYESIT műveletek mindegyike.
A keresés nyilvánvaló, SPLAY(F,K) után ellenőrizni kell, hogy K kulcsú elem került-e a gyökérbe.



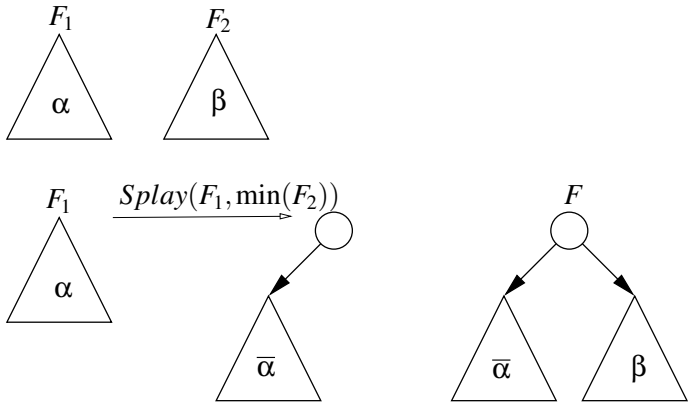
8. ábra. A BOVIT(F,K) művelet megvalósítása SPLAY művelettel. (a) eset: $\overline{K} \leq K$, (b)eset: $\overline{K} > K$.



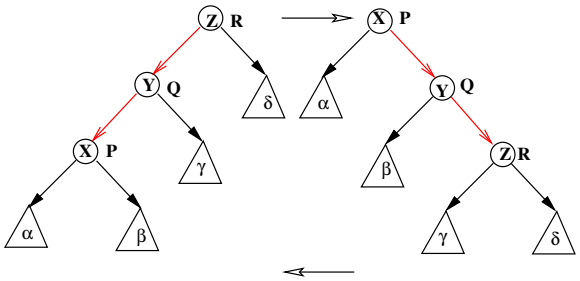
9. ábra. A TOROL(F,K) művelet megvalósítása SPLAY művelettel.



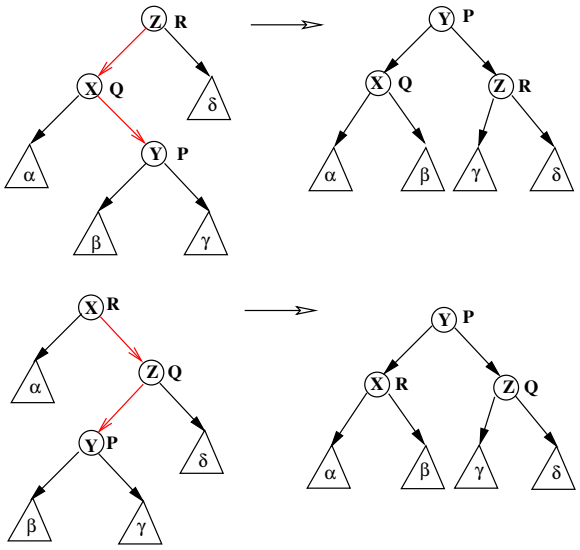
10. ábra. A VAG(F,K,F_1,F_2) művelet megvalósítása SPLAY művelettel.



11. ábra. A EGYESIT(F_1, F_2, F) művelet megvalósítása SPLAY művelettel.



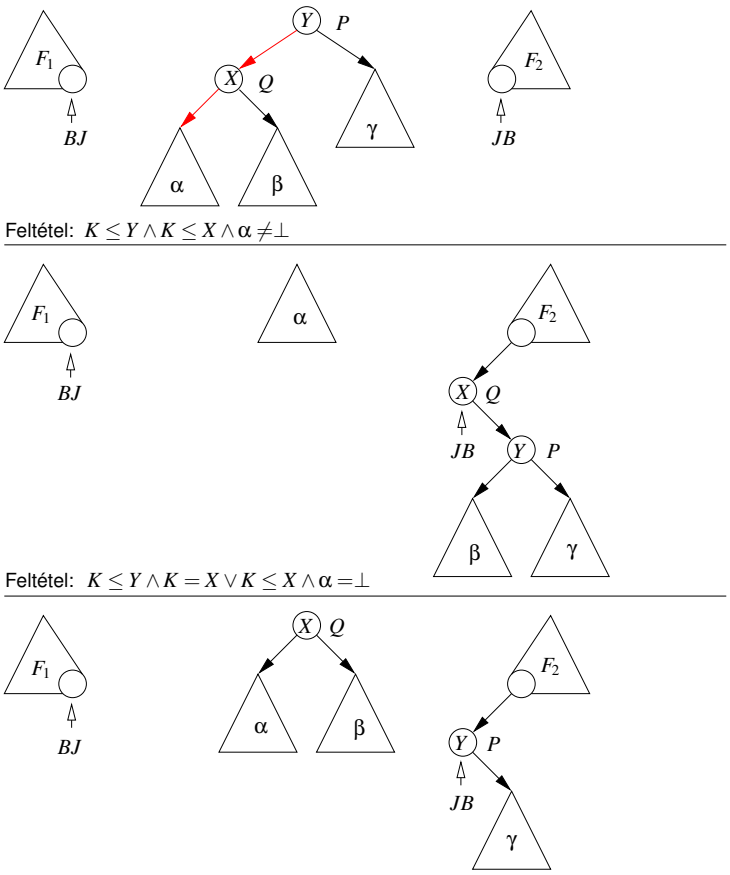
12. ábra. Alulról felfelé haladó transzformáció.



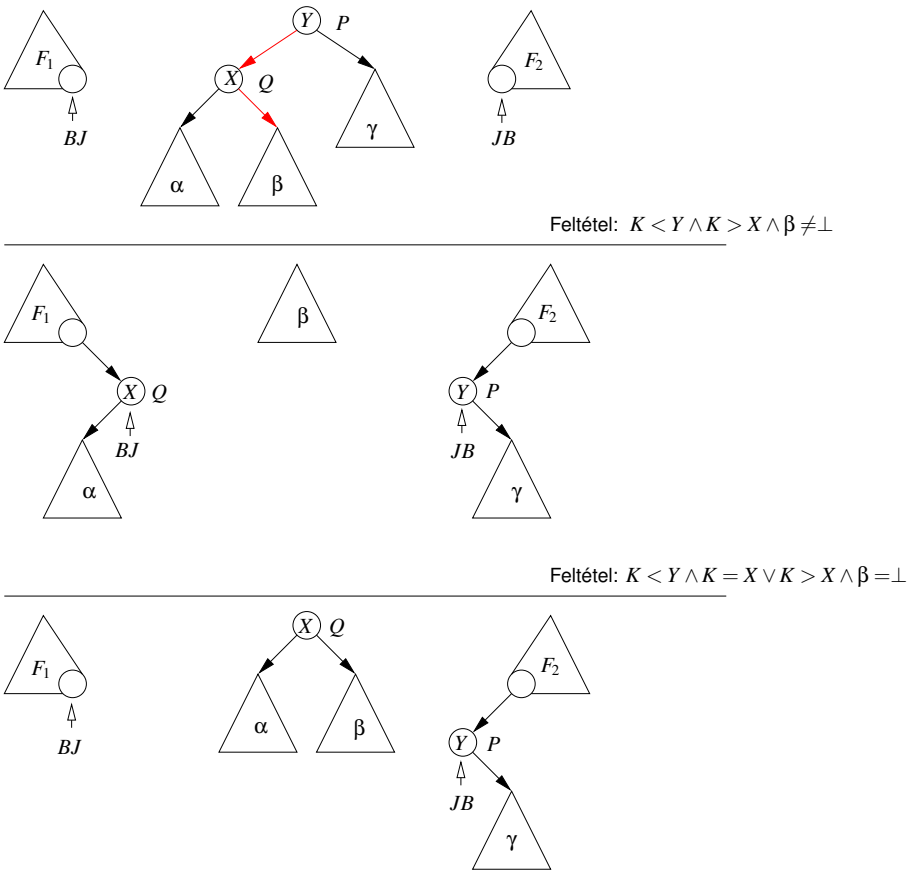
13. ábra. Alulról felfelé haladó transzformáció.

A SPLAY transzformáció megvalósítása felülről lefelé haladva.
A BKFAVAG művelethez hasonlóan, a SPLAY műveletet is transzformációs lépések sorozatával valósítjuk meg, amely az $S5$ összeépítő lépés végrehajtásával ér véget. Minden lépésben három fa vesz részt, a kezdő hármas: $\langle \perp, F, \perp \rangle$.
A transzformációs lépések mindegyikére a következő tulajdonság invariáns lesz.

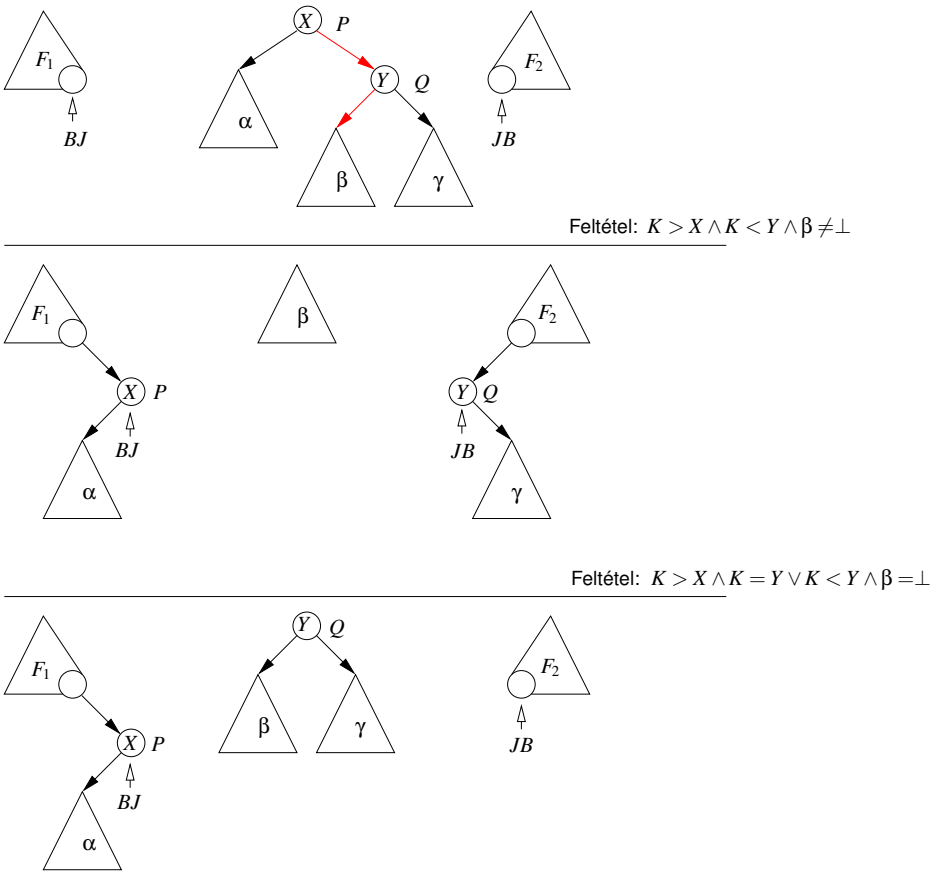
1. Az F, F_1 és F_2 fa bináris keresőfa
2. $\max(F_1) \leq K \leq \min(F_2)$ ha $F_1 \neq \perp$ és $F_2 \neq \perp$
3. $\max(F_1) \leq \min(F)$ ha $F_1 \neq \perp$
4. $\max(F) \leq \min(F_2)$ ha $F_2 \neq \perp$



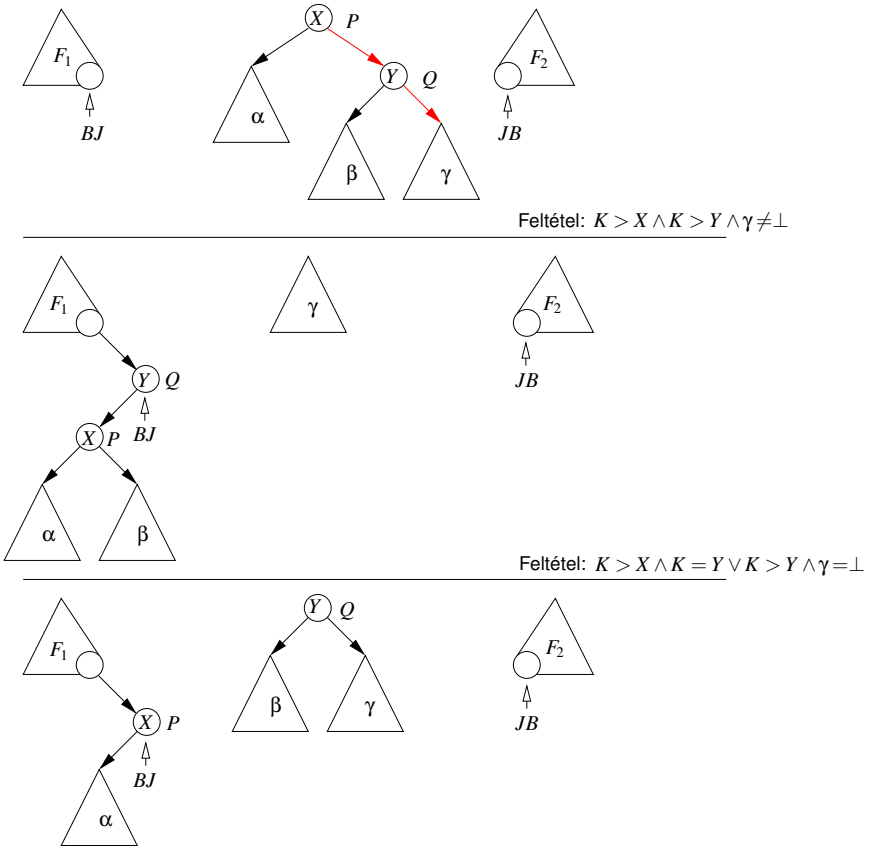
14. ábra. $S1$: Cikk-cikk transzformáció



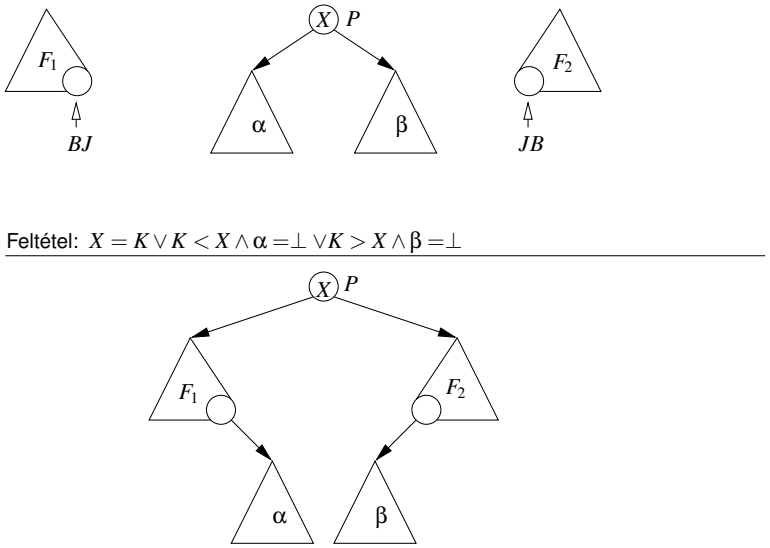
15. ábra. $S2$: Cikk-cakk transzformáció



16. ábra. $S3$: Cakk-cikk transzformáció



17. ábra. $S4$: Cakk-cakk transzformáció



18. ábra. $S5$: Összeépítő transzformáció

Mivel mindegyik transzformációs lépés teljesíti az invariánst, ezért a keletkező fára teljesül a Splay tulajdonság.

5.1. tétel. *Ha a halmazok ábrázolására önszervező bináris keresőfát használunk, akkor minden $\alpha_1, \ldots, \alpha_m$ műveletsor, ahol $\alpha_i \in \{$*KERES, BOVIT, TOROL, VAG, EGYESIT*$\}$ összesített futási ideje $O(m \lg n)$, ahol n a* BOVIT *műveletek száma és a műveletsor előtt üres halmazzal indulunk.*

```
private void Splay(E x){
   BinKerFaPont<E> BJSarok, JBSarok, p, q;
   BJSarok = Fej; JBSarok = Fej;
   Fej.bal = Fej.jobb = null;
   p = (BinKerFaPont<E>)gyoker;
   for (;;) {
      if (x.compareTo(p.elem) < 0) {
         q=(BinKerFaPont<E>)p.bal;
         if (q!=null && x.compareTo(q.elem) < 0) {//a keresőút balra halad
            p.bal = q.jobb;                // jobbra forgat
            if (q.jobb!=null) q.jobb.apa=p;
            q.jobb = p;
            p.apa=q;
            p = q;
         }
         if (p.bal == null) break;
         JBSarok.bal = p;                  // jobbra kapcsol
         p.apa=JBSarok;
         JBSarok = p;
         p = (BinKerFaPont<E>)p.bal;

      } else if (x.compareTo(p.elem) > 0) {a keresőút jobbra halad
         q=(BinKerFaPont<E>)p.jobb;
         if (q!=null && x.compareTo(q.elem) > 0) {
            p.jobb = q.bal;                // balra forgat
            if (q.bal!=null) q.bal.apa=p;
            q.bal = p;
            p.apa=q;
            p = q;
         }
```

```
         if (p.jobb == null) break;
         BJSarok.jobb = p;                 // balra kapcsol
         p.apa=BJSarok;
         BJSarok = p;
         p = (BinKerFaPont<E>)p.jobb;
      } else {
         break;
      }
   }

   BinFaPont<E> f1=p.bal;
   BinFaPont<E> f2=p.jobb;
   BJSarok.jobb = f1;                      // összeépítés
   JBSarok.bal = f2;
   p.bal = Fej.jobb;
   p.jobb = Fej.bal;
   if (f1!=null && BJSarok!=Fej) f1.apa=BJSarok;
   if (f2!=null && JBSarok!=Fej) f2.apa=JBSarok;
   if (p.bal!=null) p.bal.apa=p;
   if (p.jobb!=null) p.jobb.apa=p;
   p.apa=null;
   gyoker = p;
}

   public boolean Bovit(E x, boolean tobb){
      int ken;
      if (gyoker == null) {
          gyoker = new BinKerFaPont(x,null,null);
          return true;
      }
      Splay(x);
      if (( ken = x.compareTo(gyoker.elem)) == 0 && !tobb) return false;
      BinKerFaPont<E> ujpont = new BinKerFaPont<E>(x,null,null);
      if (ken < 0) {
          ujpont.bal = gyoker.bal;
          if (gyoker.bal!=null) gyoker.bal.apa=ujpont;
          ujpont.jobb = gyoker;
          gyoker.bal = null;
      } else {
          ujpont.jobb = gyoker.jobb;
          if (gyoker.jobb!=null) gyoker.jobb.apa=ujpont;
          ujpont.bal = gyoker;
          gyoker.jobb = null;
      }
      gyoker.apa=ujpont; gyoker = ujpont;
      gyoker.apa=null;
      return true;
   }//Bovit

public boolean Torol(E x){
   if (gyoker==null) return false;
   BinKerFaPont<E> f2;
   Splay(x);
   if (x.compareTo(gyoker.elem) != 0)
       return false;
   // a gyöker törlése
```

```
   f2 = (BinKerFaPont<E>)gyoker.jobb;
   gyoker = gyoker.bal;
   if (gyoker!=null){
      gyoker.apa=null;
      Splay(x);
      gyoker.jobb = f2;
      if (f2!=null) f2.apa=gyoker;
   }else
      gyoker=f2;
   if (gyoker!=null) gyoker.apa=null;
   return true;
}//Torol

   public BinKerFa<E> Vag(E x){
      Splay(x);
      BinKerFaPont<E> p2 = (BinKerFaPont<E>)gyoker;
      if (x.compareTo(gyoker.elem) <=0){
         gyoker=p2.bal;
         if (p2.bal!=null) gyoker.apa=null;
         p2.bal=null;
      }else{
         p2=(BinKerFaPont<E>)gyoker.jobb;
         if (p2.bal!=null) gyoker.jobb=null;
         gyoker.apa=null;
      }
      return new BinKerFaS<E>(p2);
   }

   public void Egyesit(BinKerFaS<E> f2){
      if (gyoker==null){
         gyoker=f2.gyoker;
         return;
      }
      BinKerFaPont<E> p2 = (BinKerFaPont<E>)f2.gyoker;
      if (p2==null) return;
      BinKerFaPont<E> p2m=(BinKerFaPont<E>)f2.Min();
      Splay(p2.elem);
      if (gyoker.elem.compareTo(p2m.elem)>0 )
         throw new RuntimeException("A két Halmaz nem erősen diszjunkt");
      gyoker.jobb=p2;
      p2.apa=gyoker;
      f2.gyoker=null;
   }
```

5.3. A VHHalmaz adattípus megvalósítása önszervező bináris keresőfával

```
public class VEHalmazSFa<E extends Comparable<E>>
   extends RHalmazFa<E> implements VEHalmaz<E>{
// örökölt adattagok RHalmazFa<E>-ból:
// protected BinKerFa<E><E> Fa;
// protected int eszam;
// private boolean multi=true;

   VEHalmazSFa(boolean multi){
      super("Splay", multi);
      Fa=new BinKerFaS<E>();
```

```
      eszam=0;
   }
   VEHalmazSFa(){
      super("Splay");
      Fa=new BinKerFaS<E>();
      eszam=0;
   }

   public VEHalmaz<E> Vag(E x){
      BinKerFaS<E> F2=(BinKerFaS<E>)((BinKerFaS<E>)Fa).Vag(x);
      VEHalmazSFa<E> H2=new VEHalmazSFa<E>();
      H2.Fa=F2;
      H2.eszam=Integer.MIN_VALUE;
      eszam=Integer.MIN_VALUE;
      return H2;
   }
   public void Egyesit(VEHalmaz<E> H2){
      if (H2 instanceof VEHalmazSFa){
         ((BinKerFaS<E>)Fa).Egyesit( (BinKerFaS<E>)((VEHalmazSFa<E>)H2).Fa );
//       eszam=Elemszam()+H2.Elemszam();
         eszam=Integer.MIN_VALUE;
      }else
         throw new RuntimeException("A két Halmaz nem azonos ábrázolású");
   }

   private int Bejar(BinFaPont<E> p){
      if (p==null) return 0;
      return 1+Bejar(p.bal)+Bejar(p.jobb);
   }
   public int Elemszam(){
      if (Fa.gyoker==null)
         eszam=0;
      else if (eszam<0){
         eszam=Bejar(Fa.gyoker);
      }
      return eszam;
   }
}
```