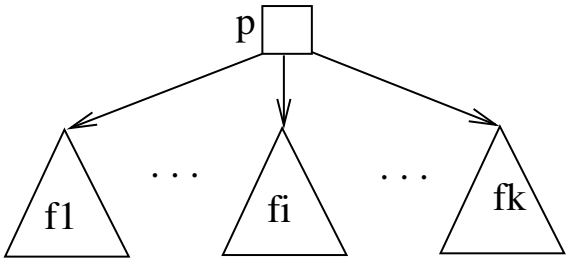
6. Fabejáró algoritmusok

Fa bejárásán olyan algoritmust értünk, amelynek bemenete egy F fa és egy M művelet, és az algoritmus adott sorrendben pontosan egyszer végrehajtja az M műveletet a fa pontjaiban lévő adatokra. A **preorder** bejárási sorrend azt jelenti, hogy F minden p és q pontjára p megelőzi q-t a bejárási sorrendben, ha

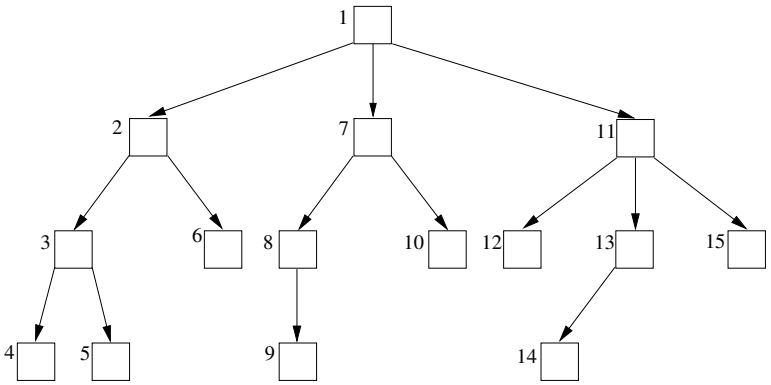
1. q fia p-nek,
2. p bal-testvére q-nak.

Tehát ha a p gyökerű fa fiai sorrendben az $f_1, \ldots, f_k$ fák, akkor

$$preorder(p) = \begin{cases} \langle p \rangle & \text{ha } k = 0 \\ \langle p, preorder(f_1), \ldots, preorder(f_k) \rangle & \text{ha } k > 0 \end{cases}$$



1. ábra. Fa preorder bejárása.



2. ábra. Preorder bejárási sorrend.

6.1. Rekurzív preorder bejárás (elsőfiú-testvér ábrázolásra)

```
public static <E> void Preorder(FaPont<E> p, Muvelet<E> M){
   if (p==null) return;
   M.muvelet(p.elem);
   FaPont<E> q=p.elsofiu;
   while (q!=null){
      Preorder(q, M);
      q=q.testver;
   }
}
```

Rekurzív preorder bejárás változata

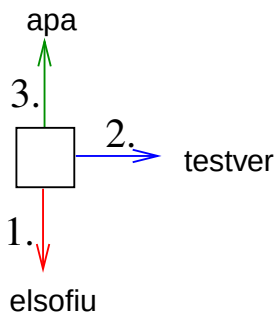
```
public static <E> void Preorder2(FaPont<E> p, Muvelet<E> M){
   if (p==null) return;
   M.muvelet(p.elem);
   if (p.elsofiu!=null)
      Preorder2(p.elsofiu, M);
   if (p.testver!=null)
      Preorder2(p.testver, M);
}
```

A Preorder2 algoritmus helyessége.

1. Alaplépés. Akkor és csak akkor nincs rekurzív hívás, ha a p pontnak se fia, se testvére nincs. Ekkor helyesen működik, mert végrehajtja az M műveletet a p pont adatára és terminál.
2. Rekurzív lépés. Bizonyítandó, hogy ha mindkét rekurzív hívás helyes, akkor helyes lesz a p gyökerű fára is. Az első rekurzív hívás az f_1 fa (helyes) preorder bejárását végzi el, majd a második rekurzív hívás pedig a $p.testver$ pontból az $elsofiu$ és $testver$ kapcsolatok szerint elérhető pontokra végez helyes preorder bejárást, tehát a p gyökerű fa preorder bejárását kapjuk.

6.2. Nem-rekurzív preorder bejárás

Feltesszük, hogy minden pont tartalmaz apa kapcsolatot.

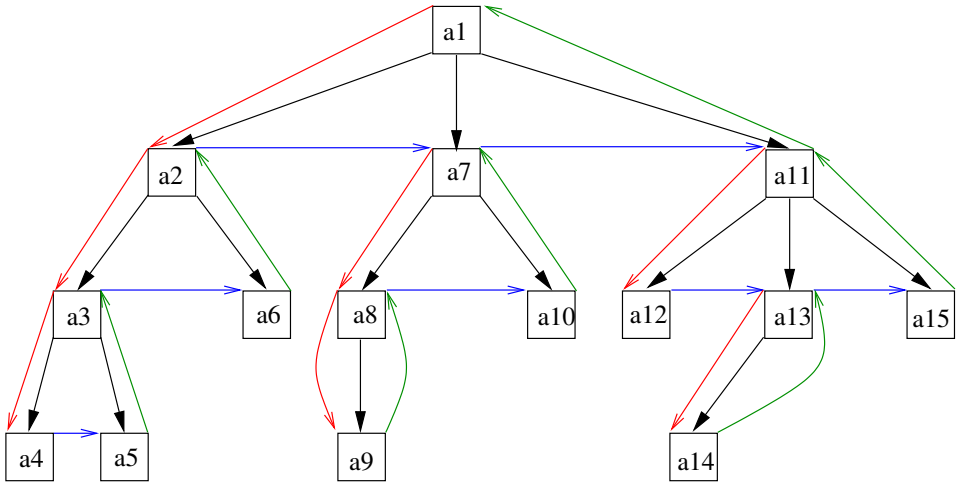


3. ábra. Továbblépési stratégia.

A bejárás során a továbblépési sorrendben:
1. Első-fiúra, ha van
2. Testvérre, ha van
3. Apára (visszalépés), ha van.

A műveletet a pont első érintésekor hajtjuk végre.

```
public static <E> void PreorderN(FaPont<E> p, Muvelet<E> M){
   while (p!=null){
      M.muvelet(p.elem);
      while (p.elsofiu!=null){
         p=p.elsofiu;
         M.muvelet(p.elem);
      }
      while ((p!=null) && (p.testver==null))
         p=p.apa;
      if (p!=null)
         p=p.testver;
   }
}
public static <E> void Postorder(FaPont<E> p, Muvelet<E> M){
   if (p==null) return;
```



4. ábra. Fa nem-rekurzív preorder bejárása.

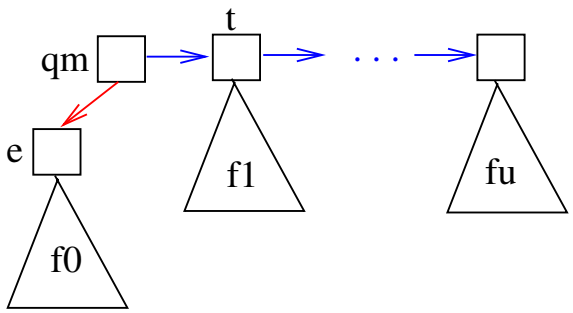
```
   FaPont<E> aktpont=p;
   p=p.elsofiu;
   while (p!=null){
      Postorder(p, M);
      p=p.testver;
   }
   M.muvelet(aktpont.elem);
}
```

6.3. Nem-rekurzív bejárás veremmel

```
public void PreorderV(FaPont<E> F, Muvelet<E> M){
// Preorder bejárás veremmel.
   Verem<FaPont<E>> V=new VeremL<FaPont<E>>();
   V.VeremBe(F);
   FaPont<E> p;
   while (V.NemUres()){
      p=V.VeremBol();
      M.muvelet(p.elem);
      if (p.testver!=null)
         V.VeremBe(p.testver);
      if (p.elsofiu!=null)
         V.VeremBe(p.elsofiu)
   }
}
```

A PREORDERV algoritmus helyességének bizonyítása.
Tekintsük a while ciklus végrehajtásának egy adott pillanatát.
Legyen $B = \langle p_1, \ldots, p_k \rangle$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a veremből kikerültek.
Legyen $V = \langle q_1, \ldots, q_m \rangle$ a V verem tartalma, ezek az aktív pontok.
A következő négy állítás konjunkciója ciklusinvariáns lesz.
1. Az B sorozatban a pontok helyes preorder sorrendben vannak.
2. A preorder bejárásban p_k-t közvetlenül q_m követi.
3. A preorder sorrendben q_i megelőzi q_{i-1}-et ($i = 2, \ldots, m$).
4. $B \cap V = \emptyset$ és a fa bármely $p \notin B$ pontjára, pontosan egy olyan $q \in V$ pont van, hogy p leszármazottja q-nak az elsőfiú-testvér fában.

Megmutatjuk, hogy ha a feltételek teljesülnek a ciklusmag végrehajtása előtt, és az ismétlési feltétel igaz, azaz a verem nem üres, akkor a ciklusmag végrehajtása után is teljesülni fognak.
Először kivesszük a q_m pontot a veremből az p változóba és végrehajtjuk rá az M műveletet, majd betesszük a verembe q_m t testvérét, ha létezik, aztán betesszük a verembe q_m e első fiát, ha létezik. Tehát $B = \langle p_1, \ldots, p_k, q_m \rangle$ lesz és a verem tartalmára



5. ábra. q_m a veremből éppen kivett pont.

az alábbi négy eset lehetséges.

a. $V = \langle q_1, \ldots, q_{m-1} \rangle$

b. $V = \langle q_1, \ldots, q_{m-1}, e \rangle$

c. $V = \langle q_1, \ldots, q_{m-1}, t \rangle$

d. $V = \langle q_1, \ldots, q_{m-1}, t, e \rangle$

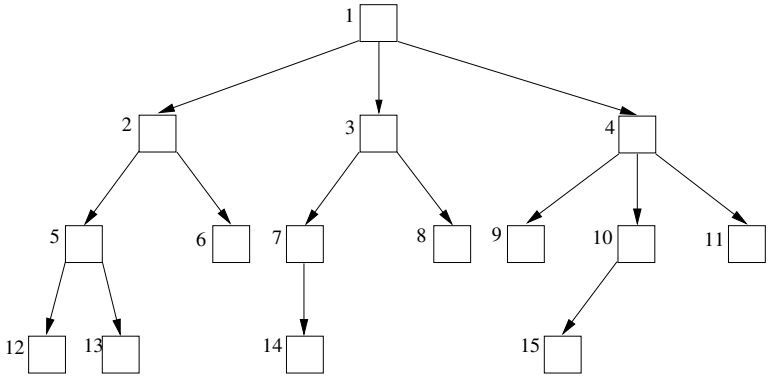
Az 1. feltétel teljesül, mert a 2. feltétel teljesült a ciklusmag előtt.
A 2. feltétel azért teljesül, mert q_m preorder követője az **a.** esetben q_{m-1}, a **b.** és **d.** esetben e, a **c.** esetben pedig t.
Mivel q_m megelőzi a preorder sorrendben q_{m-1}-et, ezért q_m minden leszármazottja, így e és t is megelőzi, továbbá e megelőzi t-t, tehát a 3. feltétel is teljesül.
A 4. feltétel nyilvánvalóan teljesül, hiszen q_m átkerült az B sorba, és q_m bármely leszármazottja vagy e-nek, vagy t-nek leszármazottja az Elsőfiú-testvér fában.
Az 1-4. feltételek mindegyike teljesül a ciklus végrehajtása előtt, mert $S = \langle\rangle$ és V csak a fa F gyökerét tartalmazza ($V = \langle F \rangle$).
Tehát a while ciklus bizonyítási szabálya szerint a while ciklus után $V = \langle\rangle$ és teljesül az 1-4. feltételek mindegyike. Ez azt jelenti, hogy B a fa minden pontját tartalmazza helyes preorder sorrendben.

6.4. Szintszerinti bejárás

A szintszerinti bejárási sorrend. Egy F fa bármely két, p és q pontjára p akkor és csak akkor előzi meg q-t a szintszerinti bejárásban, ha

$$d(p) < d(q) \vee d(p) = d(q) \wedge (\exists \bar{p}, \bar{q})(\bar{p} \textit{ bal-testvére } \bar{q} - \textit{nak} \wedge p \trianglelefteq \bar{p} \wedge q \trianglelefteq \bar{q})$$

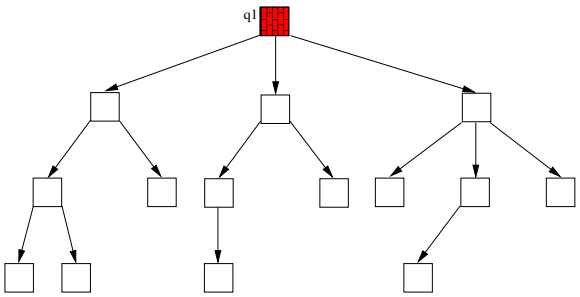
```
public static <E> void SzintBejar(FaPont<E> F, Muvelet<E> M){
   if (F==null) return;
   Sor<FaPont<E>> S = new SorL<FaPont<E>>();
   FaPont<E> p;
   S.SorBa(F);

   while (S.Elemszam()!=0){
      p=S.SorBol();
      M.muvelet(p.elem);
      p=p.elsofiu;
      while (p!=null){
         S.SorBa(p);
         p=p.testver;
```



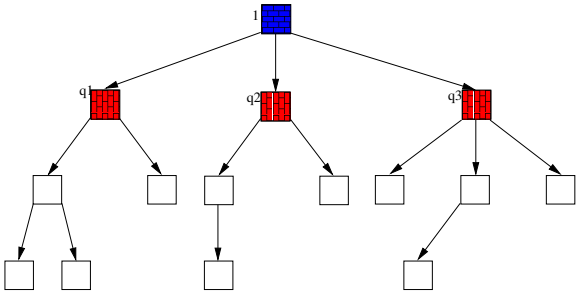
6. ábra. Fa szintszerinti bejárási sorrendje.

```
        }
    }
}
```

A SZINTFABEJAR algoritmus működésének szemléltetése.



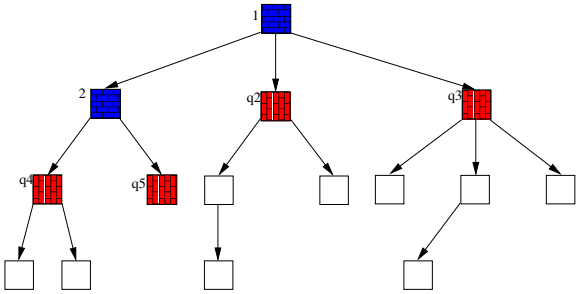
7. ábra. Az ciklus első végrehajtása előtt.



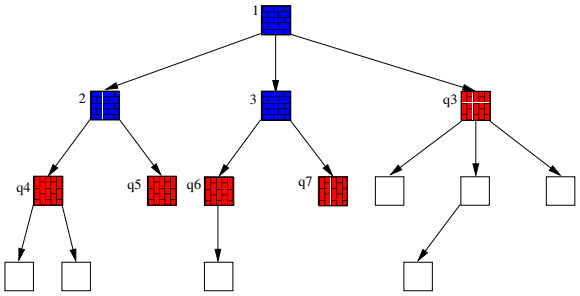
8. ábra. Az ciklus első végrehajtása után.

A SZINTFABEJAR algoritmus helyességének bizonyítása.

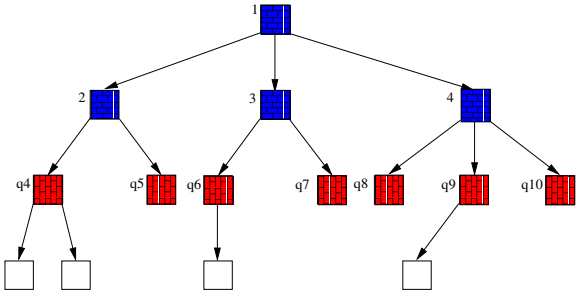
Tekintsük a külső while ciklus végrehajtásának egy adott pillanatát.
Legyen $B = \langle p_1, \ldots, p_k \rangle$ a fa azon pontjainak halmaza, amelyeket már bejártunk, abban a sorrendben, ahogy a sorból kikerültek.
Legyen $S = \langle q_1, \ldots, q_m \rangle$ a S sor aktuális tartalma, ezek az aktív pontok.
A következő öt állítás konjunkciója ciklusinvariáns lesz.
1. Az B sorozatban a pontok szintszerinti sorrendben vannak.
2. Az S sorban a pontok szintszerinti sorrendben vannak.
3. $d(q_m) \leq d(q_1) + 1$



9. ábra. Az ciklus második végrehajtása után.



10. ábra. Az ciklus harmadik végrehajtása után.



11. ábra. Az ciklus negyedik végrehajtása után.

4. A szintszerinti bejárásban p_k megelőzi q_1-et.
5. $B \cap S = \emptyset$ és $B \cup S = F$.

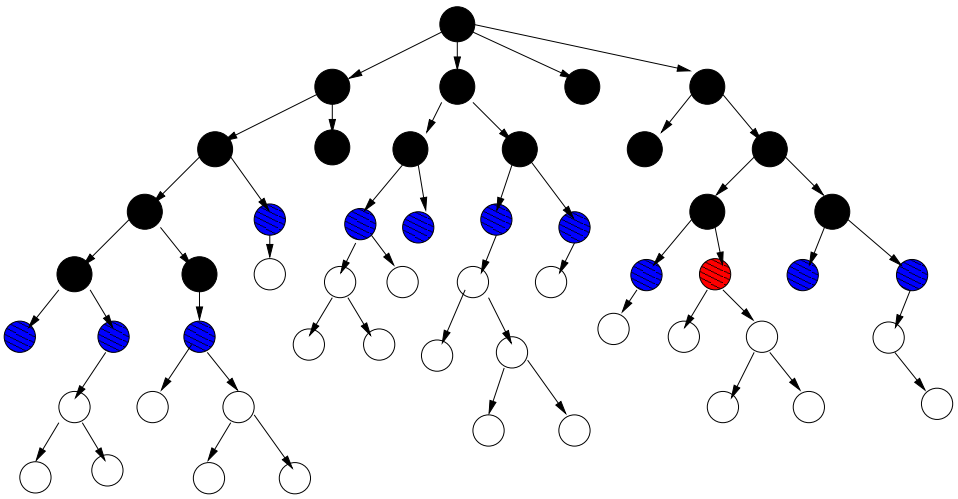
Megmutatjuk, hogy ha a feltételek teljesülnek a ciklusmag végrehajtása előtt, és az ismétlési feltétel igaz, azaz az S sor nem üres, akkor a ciklusmag végrehajtása után is teljesülni fognak.
Először kivesszük a sorból a q_1 pontot az F változóba és végrehajtjuk rá az M műveletet, majd betesszük a sorba q_1 fiait a $\langle f_1, \ldots, f_u \rangle$ fabeli sorrendjükben.
Az 1. feltétel teljesül, mert a 2. és 4. feltétel teljesült a ciklusmag előtt.
Mivel q_1 megelőzi q_2-t, ezért $d(q_1) \leq d(q_2)$, tehát $d(f_u) = d(q_1) + 1 \geq d(q_2) + 1$, tehát a 3. feltétel is teljesül. A 4. feltétel következik abból, hogy előtte teljesült a 2. feltétel.
Az 5. feltétel nyilvánvalóan teljesül, hiszen q_1 átkerült az B halmazba, és q_1 bármely leszármazottja valamelyik f_i leszármazottja.
A 2. feltétel teljesülésének bizonyítása maradt hátra. Ha $m = 1$, akkor az S sor új tartalma éppen a q_1 pont fiai lesznek. Ha $m > 1$, akkor elég azt bizonyítani, hogy q_m megelőzi f_1-et a szintszerinti sorrendben. A 3. feltétel miatt teljesül a $d(f_1) = d(q_1) + 1 \geq d(q_m)$. Ha $d(f_1) > d(q_m)$, akkor q_m a definíció szerint megelőzi f_1-et. Ha $d(f_1) = d(q_m)$, akkor legyen $\overline{q_m} := Apa(q_m)$, tehát $d(q_1) = d(\overline{q_m})$. Mivel q_m csak úgy kerülhetett be az S sorba, hogy kivettük apját, tehát $\overline{q_m}$ már B-ben van, tehát $\overline{q_m}$ előbb van a sorrendben, mint q_1. Mivel $d(q_1) = d(\overline{q_m})$, ezért a definíció szerint van olyan r_1 és r_2 pont, hogy r_1-nek jobb-testvére r_2 és q_1 leszármazottja r_1-nek, $\overline{q_m}$ pedig leszármazottja r_2-nek. De így f_1 leszármazottja r_1-nek és q_m is leszármazottja r_2-nek, továbbá $d(f_1) = d(q_m)$, tehát f_1 megelőzi q_m-et.

Az 1-5. feltételek mindegyike teljesül a ciklus végrehajtása előtt, mert $S = \langle\rangle$ és V csak a fa F gyökerét tartalmazza ($V = \langle F \rangle$). Tehát a while ciklus bizonyítási szabálya szerint a while ciklus után $V = \langle\rangle$ és teljesül az 1-5. feltételek mindegyike. Ez azt jelenti, hogy B a fa minden pontját tartalmazza helyes preorder sorrendben. Vegyük észre, hogy nem fontos a szintszerinti bejárási sorrend, a fenti algoritmusban az aktív pontok tárolására minden olyan absztrakt adattípus használható lenne, amely biztosítaná az alábbi specifikációban adott műveleteket.
Értékhalmaz: $Adagolo = \{A : A \subseteq E\}$
Műveletek:

$$A : Adagolo, x : E$$

$\{Igaz\}$	$Letesit(A)$	$\{A = \emptyset\}$
$\{A = A\}$	$Megszuntet(A)$	$\{Igaz\}$
$\{A = A\}$	$Uresit(A)$	$\{A = \emptyset\}$
$\{A = A\}$	$Betesz(A,x)$	$\{A = Pre(A) \cup \{x\}\}$
$\{A \neq \emptyset\}$	$Kivesz(A,x)$	$\{x \in Pre(A) \wedge Pre(A) = A \cup \{x\}\}$
$\{A = A\}$	$Elemszam(A)$	$\{Elemszam = \lvert A \rvert\}$

```
public static <E> void AdBejar(FaPont<E> F, Muvelet<E> M){
   if (F==null) return;
   Adagolo<FaPont<E>> A = new Adagolo<FaPont<E>>();
   FaPont<E> p;
   A.Betesz(F);

   while (A.Elemszam()!=0){
      p=A.Kivesz();
      M.muvelet(p.elem);
      p=p.elsofiu;
      while (p!=null){
         A.Betesz(p);
         p=p.testver;
      }
   }
}
```



12. ábra. A fa pontjai adagolós bejárás során.