1. **Tétel**

A diszkrét valószínűség-eloszlás valószínűségi tömegfüggvénnyel jellemzett valószínűség-eloszlás. Így az *X* valószínűségi változó eloszlása diszkrét, és *X*-et diszkrét valószínűségi változónak nevezzük, ha



mivel *u* az összes lehetséges *X* értéken értelmezhető. Ebből következik, hogy az ilyen változó csak véges vagy megszámlálhatóan végtelen számértékeket vehet fel.

A legismertebb diszkrét valószínűség-eloszlás, melyet statisztikai modellezésre is használnak, a [Poisson-eloszlás](https://hu.wikipedia.org/wiki/Poisson-eloszl%C3%A1s), a [Bernoulli-eloszlás](https://hu.wikipedia.org/wiki/Bernoulli-eloszl%C3%A1s), a [binomiális eloszlás](https://hu.wikipedia.org/wiki/Binomi%C3%A1lis_eloszl%C3%A1s), a [geometriai eloszlás](https://hu.wikipedia.org/wiki/Geometriai_eloszl%C3%A1s) és a [negatív binomiális eloszlás](https://hu.wikipedia.org/wiki/Negat%C3%ADv_binomi%C3%A1lis_eloszl%C3%A1s).

Ezenfelül a [diszkrét egyenletes eloszlást](https://hu.wikipedia.org/wiki/Diszkr%C3%A9t_egyenletes_eloszl%C3%A1s) általánosan alkalmazzák a számítógépes programozásban az egyenletesen kiválasztott véletlenszerű számoknál.

**Kumulatív sűrűség**

A fentieknek megfelelően egy diszkrét valószínűségi változót úgy határozhatunk meg, mint egy valószínűségi változót, melynek kumulatív eloszlásfüggvénye csak diszkontinuitásokkal, ugrásokkal nőhet, vagyis akkor nő, ha magasabb értékre „ugrik”, és konstans az ugrások között. Azok a pontok, ahol az ugrás történik, azok az értékek, melyeket a valószínűségi változó felvehet. Az ilyen pontok száma lehet véges vagy megszámolhatóan végtelen. Az ugrások helyének nem kell topológiailag diszkrétnek lennie; például a kumulatív eloszlásfüggvény ugorhat minden [racionális számnál](https://hu.wikipedia.org/wiki/Racion%C3%A1lis_sz%C3%A1m).

**Delta-függvény**

A diszkrét valószínűség-eloszlás gyakran a valószínűségi sűrűségfüggvény általánosított formájában jelenik meg, beleértve a [Dirac-delta](https://hu.wikipedia.org/wiki/Dirac-delta) függvényt, mely lényegében egységesíti a folytonos és diszkrét eloszlás kezelését. Ez akkor hasznos, amikor olyan valószínűség-eloszlásokkal foglalkozunk, melyek folytonos és diszkrét részeket is tartalmaznak.

**Indikátorfüggvény (**[**karakterisztikus függvény**](https://hu.wikipedia.org/wiki/Karakterisztikus_f%C3%BCggv%C3%A9ny)**)**

Legyen *X* egy diszkrét valószínűségi változó és *u*0, *u*1... azok értékek, melyeket felvehet nem zéró valószínűséggel. Jelöljük:



Ezek [diszjunkt halmazok](https://hu.wikipedia.org/wiki/Diszjunkt_halmazok) és képlettel (1):



Ebből következik, hogy *X* bármely értéket felvehet, kivéve az *u*0, *u*1, ... = 0 eseteket, és így írhatjuk:

{\displaystyle X=\sum \_{i}u\_{i}1\_{\Omega \_{i}}}

kivéve a zéró valószínűségű halmazra, ahol {\displaystyle 1\_{A}}az *A* indikátorfüggvénye.

**Binomiális elosztás**

Az *X* [valószínűségi változó](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gi_v%C3%A1ltoz%C3%B3) *n* és *p* paraméterű **binomiális eloszlást követ** – vagy rövidebben **binomiális eloszlású** – pontosan akkor, ha



ahol {\displaystyle 0\leq p\leq 1}. Azt, hogy az *X* valószínűségi változó *n* és *p* paraméterű binomiális eloszlást követ, a következő módon szoktuk jelölni: *X* ∼ *B*(*n*,*p*). Speciálisan, ha *X* ∼ *B*(1,*p*), akkor *X*-et [Bernoulli-eloszlásúnak](https://hu.wikipedia.org/wiki/Bernoulli-eloszl%C3%A1s) nevezzük.

**Poisson def**

Az *X* [valószínűségi változó](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gi_v%C3%A1ltoz%C3%B3) λ paraméterű Poisson-eloszlást követ – vagy rövidebben: Poisson-eloszlású – pontosan akkor, ha

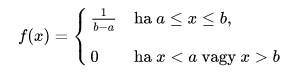


ahol λ > 0 konstans.

* Poisson-eloszlású [független](https://hu.wikipedia.org/w/index.php?title=F%C3%BCggetlens%C3%A9g_(matematika)&action=edit&redlink=1) valószínűségi változók összege is Poisson-eloszlású. Pontosabban ha *X*1 és *X*2 független Poisson-eloszlású valószínűségi változók λ1 és λ2 paraméterekkel, akkor *X*1 + *X*2 is Poisson-eloszlású λ1 + λ2 paraméterrel. Ugyanekkor *X*1 [feltételes eloszlása](https://hu.wikipedia.org/wiki/Felt%C3%A9teles_eloszl%C3%A1s) *X*1 + *X*2 = *n* -re vonatkozóan *n* és λ1/(λ1 + λ2) paraméterű [binomiális eloszlást](https://hu.wikipedia.org/wiki/Binomi%C3%A1lis_eloszl%C3%A1s) követ.
* Az összegzésre vonatkozó összefüggés fordítottja is igaz. Pontosabban ha *X*1 + *X*2 is Poisson-eloszlású valamint tudjuk, hogy *X*1 és *X*2 független valószínűségi változók, akkor *X*1 és *X*2 is Poisson-eloszlású.
* Ha binomiális eloszlások olyan [sorozatát](https://hu.wikipedia.org/wiki/Sorozat_(matematika)) vesszük, melyben az eloszlások *n* paramétere úgy tart a végtelenbe, hogy közben az *np* szorzat konstans marad (*p* így nyilván a 0-hoz tart), akkor határeloszlásként Poisson-eloszlást kapunk.

**Egyenletes elosztás**

A [valószínűségszámításban](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gsz%C3%A1m%C3%ADt%C3%A1s) egy X folytonos [valószínűségi változót](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gi_v%C3%A1ltoz%C3%B3) az [a,b] [intervallumon](https://hu.wikipedia.org/wiki/Intervallum) **egyenletes eloszlásúnak** nevezünk, ha [sűrűségfüggvénye](https://hu.wikipedia.org/wiki/S%C5%B1r%C5%B1s%C3%A9gf%C3%BCggv%C3%A9ny):



A véletlengenerátorokat úgy tervezik, hogy egy adott intervallumon minél inkább megközelítsék az egyenletes eloszlást.

Beszélnek [diszkrét egyenletes eloszlásról](https://hu.wikipedia.org/wiki/Diszkr%C3%A9t_egyenletes_eloszl%C3%A1s) is. Ilyen például a szabályos dobókockával dobott számok eloszlása.

**Exponenciális elosztás**

Az *X* [valószínűségi változó](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gi_v%C3%A1ltoz%C3%B3) λ paraméterű **exponenciális eloszlást követ** – vagy rövidebben **exponenciális eloszlású** – pontosan akkor, ha [sűrűségfüggvénye](https://hu.wikipedia.org/wiki/S%C5%B1r%C5%B1s%C3%A9gf%C3%BCggv%C3%A9ny)



ahol λ > 0.

**Normális elosztás**

Az *X* [valószínűségi változó](https://hu.wikipedia.org/wiki/Val%C3%B3sz%C3%ADn%C5%B1s%C3%A9gi_v%C3%A1ltoz%C3%B3) **normális eloszlás**t követ – vagy rövidebben: normális eloszlású – pontosan akkor, ha [sűrűségfüggvénye](https://hu.wikipedia.org/wiki/S%C5%B1r%C5%B1s%C3%A9gf%C3%BCggv%C3%A9ny)



ahol a két paraméter, *m* és [*σ*](https://hu.wikipedia.org/wiki/Szigma) ∈ [**R**](https://hu.wikipedia.org/wiki/Val%C3%B3s_sz%C3%A1mok), valamint [*σ*](https://hu.wikipedia.org/wiki/Szigma) > 0. A normális eloszlást szokták **Gauss-eloszlás**nak vagy néha **normál eloszlás**nak is nevezni.

Azt, hogy az *X* valószínűségi változó normális eloszlást követ, a következő módon szoktuk jelölni:



Speciálisan, ha *X* ~ *N*(0, 1), akkor *X*-et [standard normális eloszlásúnak](https://hu.wikipedia.org/wiki/Standard_norm%C3%A1lis_eloszl%C3%A1s) (vagy sztenderd normális eloszlásúnak) nevezzük.

A fenti sűrűségfüggvény grafikonját alakja miatt szokás **haranggörbé**nek nevezni.

**Absztrakció**

**Fizikai szint:** Az absztrakció legalacsonyabb szintje leírja, hogyan tárolja a rendszer az adatokat. A fizikai szint részletesen leírja az összetett alacsony szintű adatszerkezeteket. Absztrakció (informatika) - <https://hu.qaz.wiki/wiki/Abstraction_(computer_science)>

**Logikai szint:** Az absztrakció következő magasabb szintje leírja, hogy az adatbázis milyen adatokat tárol, és milyen összefüggések vannak az adatok között. A logikai szint tehát egy teljes adatbázist ír le kis számú, viszonylag egyszerű struktúra alapján. Bár az egyszerű struktúrák logikai szintű megvalósítása összetett fizikai szintű struktúrákat is magában foglalhat, a logikai szint felhasználójának nem kell tudatában lennie ennek a komplexitásnak. Ezt fizikai adatok függetlenségének nevezzük . Az adatbázis-rendszergazdák , akiknek el kell dönteniük, hogy milyen információkat tartsanak az adatbázisban, az absztrakció logikai szintjét használják. Absztrakció (informatika) - <https://hu.qaz.wiki/wiki/Abstraction_(computer_science)>

**Adatszerkezet**

**Homogén adatszerkezet:** Vizsgálnunk kell, hogy az adott adatszerkezetet alkotó elemi adatok típusa megegyező-e, vagy az adatszerkezetben különböző típusú elemi adatok találhatók. Ha az adatszerkezetet alkotó elemi adatok azonos típusúak, akkor homogén adatszerkezetről beszélünk

**Heterogén adatszerkezet:** ha az elemek nem azonos típusúak.

**statikus:**

adatszerkezet elemszáma állandó. Az általa elfoglalt tárterület nagysága nem változik a műveletek során.

**dinamikus:**

adatszerkezetek elemszáma is véges, hiszen a rendelkezésre álló tár nagysága behatárolja a bővítés lehetőségét. Ugyanakkor az elemszáma a műveletek során változhat. Értelmezve van az adatszerkezet üres állapota, illetve amikor elfogyott az erre a célra fenntartott tárterület, akkor azt mondjuk, hogy megtelt az adatszerkezet. Az ilyen adatszerkezetek sajátossága, hogy új adatelemmel történő bővítése során az elem számára le kell foglalni a memória egy megfelelő területét, illetve akkor, amikor egy elem feleslegessé válik, akkor gondoskodni kell arról, hogy az így felszabaduló tárterület később ismét lefoglalható legyen.

**strukúra nélküli:**

adatszerkezet esetében az adatelemek között semmiféle kapcsolat nincs (pl.: halmaz)

**asszociatív:**

adatszerkezetek elemei között nincs lényegi kapcsolat, az elemei egyedileg címezhetőek (pl.: tömb).

**szekvenciális:**

adatszerkezetben minden elem – két kitüntetett elem kivételével (első és utolsó) – pontosan egy másik elemtől érhető el, és minden elemtől pontosan egy elem érhető el. Ez alól csak a két kitüntetett elem kivétel, mert az adatszerkezetnek nincs olyan eleme, amelyből az első elem elérhető volna, és nincs olyan eleme sem, amely az utolsóból volna elérhető (pl.: egyszerű lista).

**hierarchikus:**

adatszerkezet esetében egy olyan kitüntetett elem van (a gyökérelem), amelynek megadása egyenértékű az adatszerkezet megadásával. A gyökérelem kivételével minden elem pontosan egy másik elemből érhető el, de egy elemből tetszőleges (véges) számú további elemet lehet elérni. Ez utóbbi alól a csak az adatszerkezet végpontjai kivételek (pl.: bináris fa).

**hálós:**

adatszerkezet elemei több elemből is elérhetőek és egy adott elemből is több további elemet tudunk elérni (pl.: gráf).

**Elemi adatszerkezet:**

**Lista:**

A legegyszerűbb lista esetében minden listaelem a tárolandó adaton kívül egy mutatóval van kiegészítve. Ennek az értékéből tudhatjuk meg, hogy van-e egyáltalán további eleme az adatszerkezetnek, vagy ha van, akkor a memória mely részében található.

Ebből az következik, hogy ennek a homogén adatstruktúrának az egyes elemei a memóriában szétszórtan helyezkednek el, elérésük csak szekvenciálisan lehetséges, de az adatszerkezet tárigénye dinamikusan változhat.

A továbbiakban szeretnénk modellezni magát a dinamikus tárkezelést azzal együtt, hogy konkrét adatszerkezetek műveleteit is bemutatnánk. A modellhez alapul szolgál a mutató memória és a tömbindex tömb között vont párhuzam. Modellünkben a dinamikus adatok számára felhasználható tárterületet egy vektor fogja jelenteni, a mutatók szerepét pedig az indexek töltik be.

# **Verem:**

Mind a verem, mind a sor adatszerkezet ábrázolható tömbökkel. Tekintsük az S tömböt! Jelölje az S.tető a legutoljára berakott elem indexét, így az S[1..S.tető] tömbrész tartalmazza a verem aktuális értékét. Ha S.tető= 0, akkor veremben nincs elem, azaz a verem üres. Ha üres veremből próbálunk törölni, az alulcsordulásnak nevezzük, míg ha egy S.hossz +1-dik elemet próbálunk beszúrni, akkor túlcsordulásról beszélünk. Az egyetlen lekérdező művelettel azt állapíthatjuk meg, hogy üres-e a verem, vagy sem. Az angol szakirodalomban Push és Pop néven nevezett beszúró és törlő műveletekre mi a VEREMBE és VEREMBŐ L nevekkel hivatkozunk. Mind a lekérdezés, mind beszúrás és törlés is O(1) bonyolultságú.

Function ÜRES(S)

Input: S verem

Eredmény: IGAZ, ha üres a verem, és HAMIS, ha nem

1 if S.tető == 0 then

2 return IGAZ

3 else

4 return HAMIS

5 endif

Procedure VEREMBE(S,x)

Input: S verem, x beszúrandó elem

Eredmény: Az adott elemet beszúrja verem tetejére

1 if S.tető ≤ S.hossz then

2 S.tető = S.tető + 1

3 S[S.tető] x

4 else

5 hiba "túlcsordulás"

6 endif

Function VEREMBőL(S)

Input: S verem

Output: A verem felső eleme, melyet egyben töröl is a veremből

1 if ÜRES(S) then

2 hiba "alulcsordulás"

3 endif

4 S.tető S.tető − 1

5 return S[S.tető + 1]

**Sor:**

Míg a veremnél egy adat meghatározza a verem méretét, a sornál kettőre van szükségünk. A sor feje a sor első elemét jelenti, míg a sor vége a sor utolsó elemét. Ha egy új elemet veszünk fel a sorba, akkor azt a sor végére írjuk, ahogy a pénztárnál is a sor végére állunk. Miután az első vevő fizetett a pénztárnál, elhagyja a sort. Hasonlóan a sor adattípusból is az első elemet, a sor fejét töröljük. Az alábbi programokban a Q listához tartozó Q.fej elem a sor első elemének indexe a Q[1..Q.hossz] tömbben. A Q.vége a sor utolsó eleme mögötti index, erre a helyre kerül a soron következő beszúrandó elem. A sor esetén is van egy lekérdező művelet, ez is arra ad választ, hogy az adatszerkezet üres-e vagy sem. A lista módosító műveletei hasonlóan a beszúrás (SORBA) és törlés (SORBÓL). Ha a lista üres, és így akarunk belőle törölni, akkor alulcsordul a sor. Ha pedig a megtelt listába akarunk még újabb elemeket szúrni, túlcsordul a sor. Ha hagyományos módon kezelnénk a tömböt, akkor a beszúrás és a törlés során a után mind a sor feje, mind a sor vége egyre hátrább és hátrább kerülne, s egy idő után bármilyen nagy tömb végét eléri mindkét index. Épp ezért a következő trükköt használjuk: kapcsoljuk össze a tömb végét és elejét, azaz készítsünk belőle egy gyűrűt.

Function ÜRES(Q)

Input: Q sor

Output: IGAZ, ha üres a sor, és HAMIS, ha nem

1 if Q.fej == Q.vége then

2 return IGAZ

3 else

4 return HAMIS

5 endif

Procedure SORBA(Q,x)

Input: Q sor, x beszúrandó elem

Eredmény: A sor végére beszúrja az adott elemet

1 if Q.vége == Q.hossz then

2 y = 1

3 else

4 y = Q.vége + 1

5 endif

6 if Q.fej == y then

7 hiba "Megtelt a sor!"

8 else

9 Q[Q.vége] = x

10 Q.vége = y

11 endif

Function SORBÓL(Q)

Input: Q sor

Output: A sor első eleme, amelyet egyben töröl is a sorból

1 x = Q[Q.fej]

2 if Q.fej == Q.hossz then

3 Q.fej = 1

4 else

5 Q.fej = Q.fej + 1

6 return x

**A halmaz adatszerkezet**

A halmaz adatszerkezet a matematikai halmaz fogalom

megjelenése az adatszerkezetek szintjén. Mindig véges –

ennyiben nem felel meg teljesen a matematikai halmaz

fogalmának.

**A halmaz alapműveletei**

eleme E: megmondja, hogy egy adatelem benne van-e a

halmazban vagy sem

unió U: két halmaz unióját adja

metszet : két halmaz metszetét adja

különbség \: két halmaz különbségét adja

**Az adatszerkezetekkel végezhető hagyományos műveletek**

**megvalósítása halmazok esetén:**

Létrehozás kétféleképpen:

explicit módon, a halmaz elemeinek felsorolásával (esetleg

üresen)

egy predikátum segítségével

Bővítés unióképzéssel

Törlés csak fizikai, különbségképzéssel

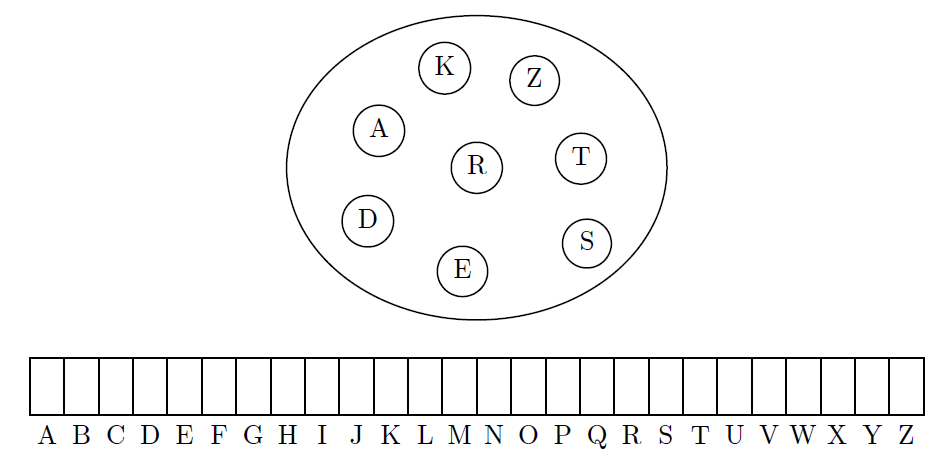
Csere nincs

Rendezés, keresés, elérés, bejárás nem értelmezettek

Feldolgozás a halmaz alapműveleteinek a segítségével

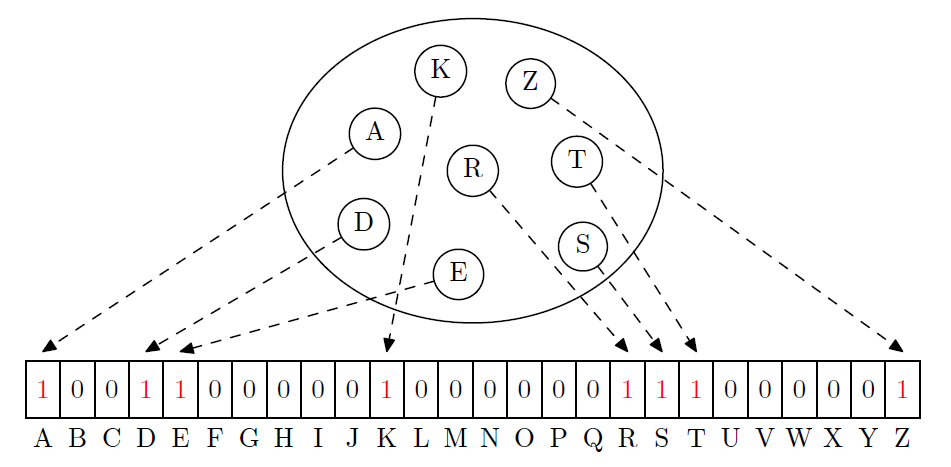
Klasszikus reprezentációja folytonosan, karakterisztikus

függvény segítségével történik.



A halmaz lehetséges elemeit sorba rendezzük, s mindegyikhez

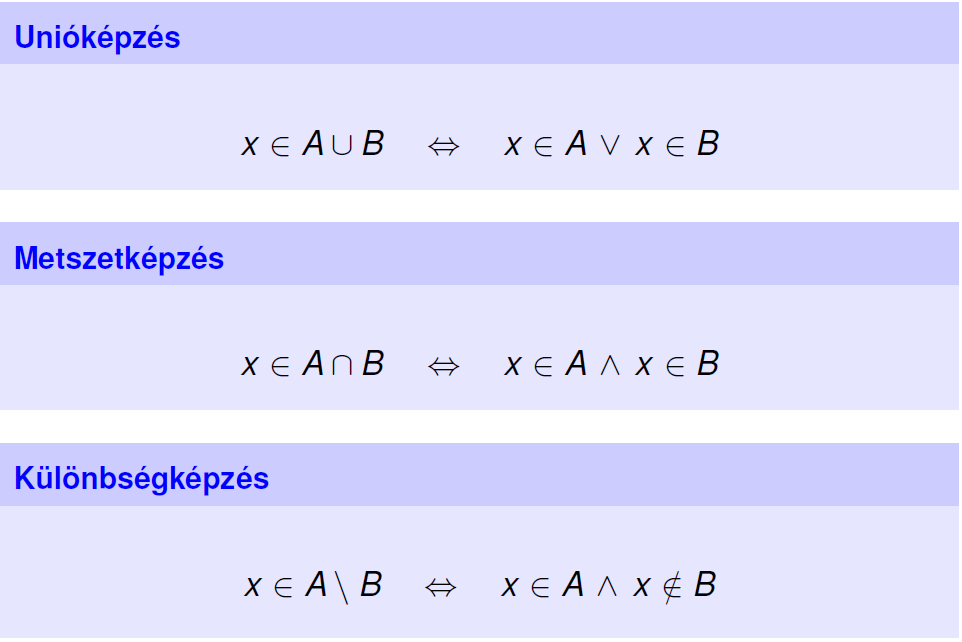
hozzárendelünk egy egy bit méretű tárterületet.



A halmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy egy bit méretű tárterületet. Az adott értékű adatelemhez tartozó bit fogja jelezni, hogy az adatelem benne van-e a halmazban (1) vagy sem (0).

**Halmaz adatszerkezet implementációja**

Folytonos reprezentáció esetén a halmaz alapműveleteinek megvalósítása visszavezethető egyszerű bitműveletekre:



**A multihalmaz adatszerkezet**

A multihalmaz abban különbözik a halmaztól, hogy megengedi az adatelemek ismétlődését, benne több azonos értékű elem is előfordulhat.

**A multihalmaz alapműveletei**

eleme E: megmondja, hogy egy adatelem benne van-e a

multihalmazban vagy sem

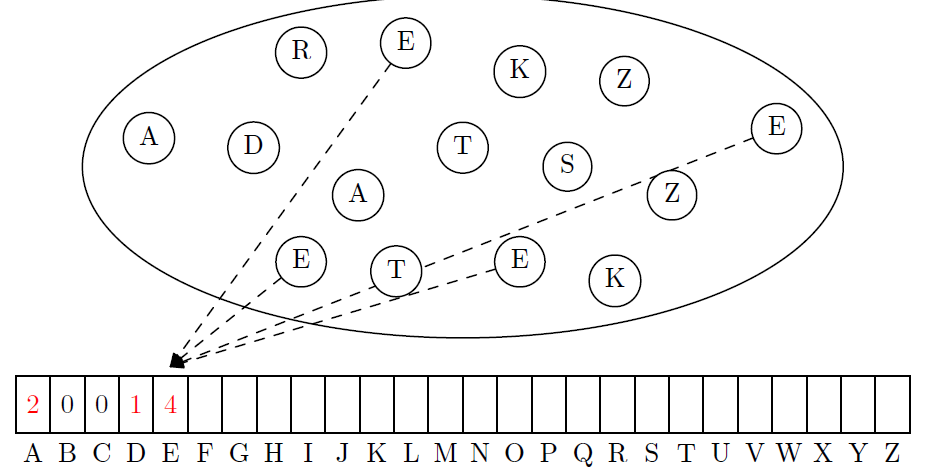
unió U: két multihalmaz unióját adja

metszet : két multihalmaz metszetét adja

különbség \: két multihalmaz különbségét adja

Multihalmazoknál az adatszerkezetekkel végezhető hagyományos műveletek megvalósítása hasonló a halmazokéhoz (lásd ott). A multihalmaz feldolgozása a multihalmaz alapműveleteinek a segítségével történik.

Klasszikus reprezentációja folytonosan, karakterisztikus függvény segítségével történik

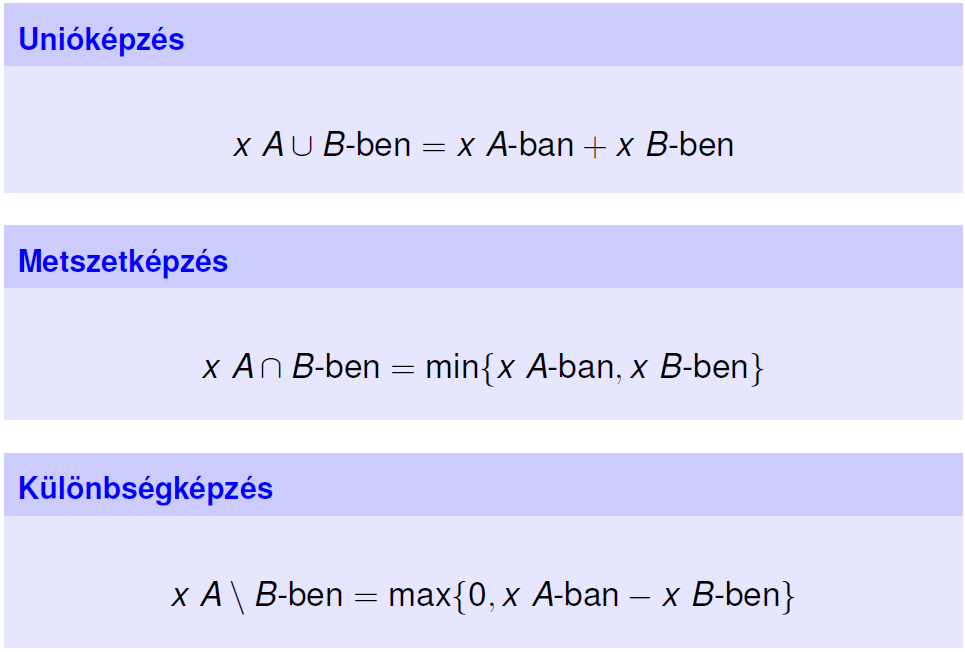


A multihalmaz lehetséges elemeit sorba rendezzük, s mindegyikhez hozzárendelünk egy tárterületet.

általában 1 bájtot

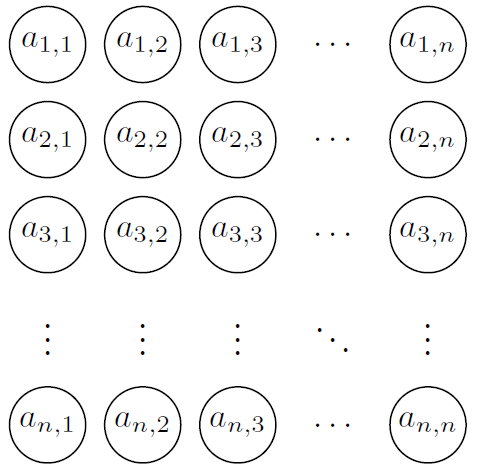
A tárhelyeken az adott értékű elemek előfordulásainak számát tároljuk.

Folytonos reprezentáció esetén a multihalmaz alapműveleteinek megvalósítása visszavezethető egyszerű aritmetikai (számtani) műveletekre:



**Háromszögmátrixok**

A háromszögmátrixok négyzetes (kvadratikus) mátrixok.

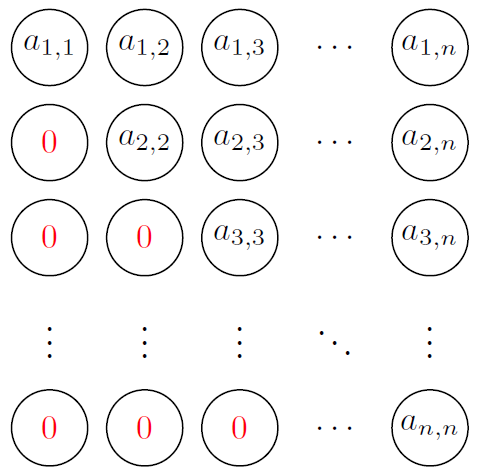


Kétfajta háromszögmátrixot szoktunk megkülönböztetni:

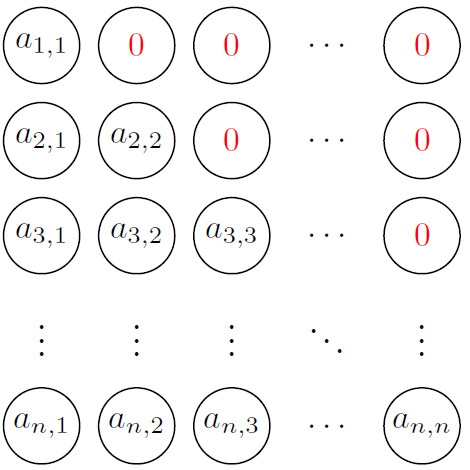
a felsőháromszög-mátrixot és

az alsóháromszög-mátrixot.

Az olyan négyzetes mátrixot, amelynek főátlója alatt csupa 0 elem található, felsőháromszög-mátrixnak nevezzük.

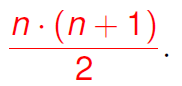


Ha a négyzetes mátrix főátlója fölött lévő elemek mindegyikének értéke 0, akkor alsóháromszög-mátrixról beszélünk.



A négyzetes mátrixokkal szemben, ahol az értékes elemek

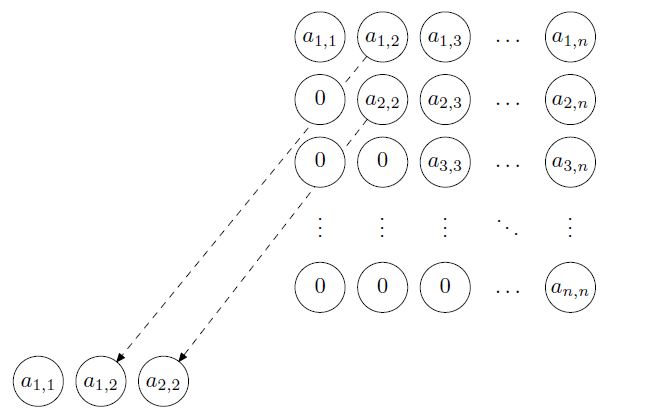
száma n^2, a háromszögmátrixoknál az értékes elemek száma

csupán 

Az értékes elemeket emiatt – sor- vagy oszlopfolytonosan – egy elemű V vektorra szoktuk leképezni.

**Felsőháromszög-mátrixok folytonos reprezentációja**

A felsőháromszög-mátrix értékes elemeit (a főátló elemeit és a fölötte elhelyezkedő elemeket) oszlopfolytonosan célszerű leképezni egy  elemű V vektorra:



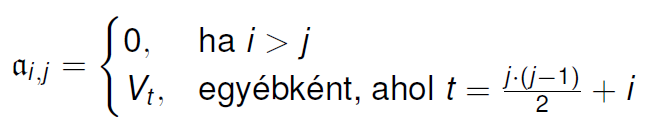
**Háromszög-mátrixok folytonos reprezentációja**

Felsőháromszög mátrixok esetén ennek megfelelően

A V vektorból a következő képlet segítségével kaphatjuk

vissza az eredeti felsőháromszög-mátrix (i; j) indexű

elemének az értékét:

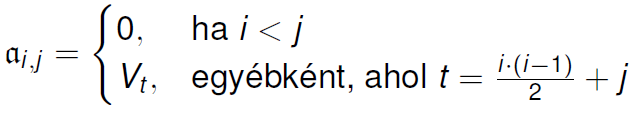


Alsóháromszög-mátrixok esetén viszont sorfolytonos leképezést célszerű használni.

Ennek megfelelően, a V vektorból a következő képlet

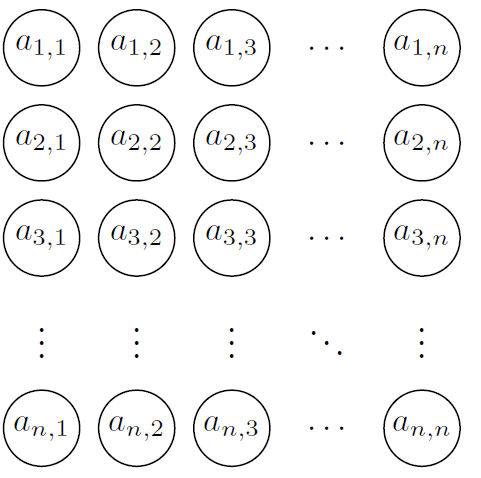
segítségével kaphatjuk vissza az eredeti

alsóháromszög-mátrix (i; j) indexű elemének az értékét:



**Szimmetrikus mátrixok**

A szimmetrikus mátrixok négyzetes (kvadratikus) mátrixok.



* Melyekre teljesül az



egyenlőség bármely i-re és j-re 

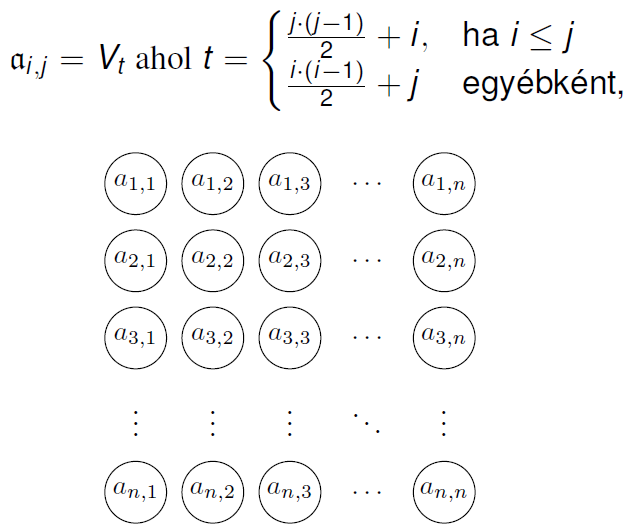
**Szimmetrikus mátrixok folytonos reprezentációja**

Szimmetrikus mátrixok esetén

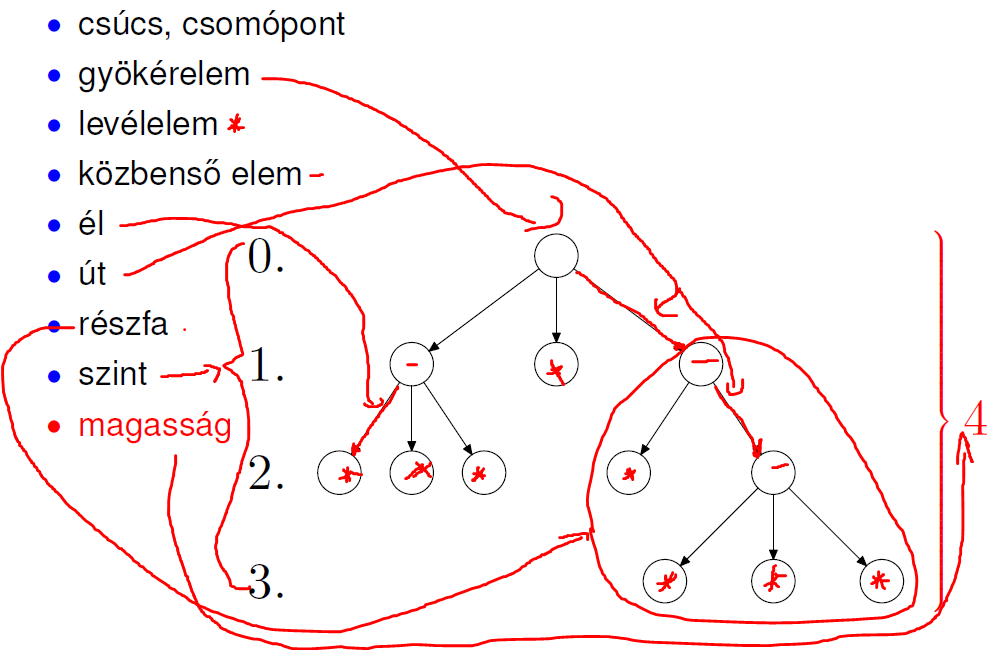
vagy a felső háromszöget képezzük le oszlopfolytonosan,

vagy pedig az alsóháromszöget sorfolytonosan

A V vektorból a következő képlet segítségével kaphatjuk vissza az eredeti szimmetrikus mátrix (i; j) indexű elemének az értékét:



**Fák ábrázolása**



Rendezetlen fáknál nem lényeges az ugyanazon csúcsból kiinduló élek sorrendje, rendezett fáknál viszont igen.

**Bináris fa**

**Bináris fa**

Olyan fa, melyben minden adatelemnek legfeljebb két

rákövetkezője van.

**Szigorú értelemben vett bináris fa**

Szigorú értelemben vett bináris fáról beszélünk, ha a bináris

fában minden adatelemnek 0 vagy 2 rákövetkezője van.

**Rendezett bináris fa**

Rendezett bináris fa elemeire értelmezhetők a következő

fogalmak:

* bal/jobb oldali rákövetkező
* bal/jobb oldali részfa

**Bináris fa bejárási algoritmusai (bejárások)**

**Preorder bejárás algoritmusa**

* Ha a bejárandó fa üres, az algoritmus véget ér.
* Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük

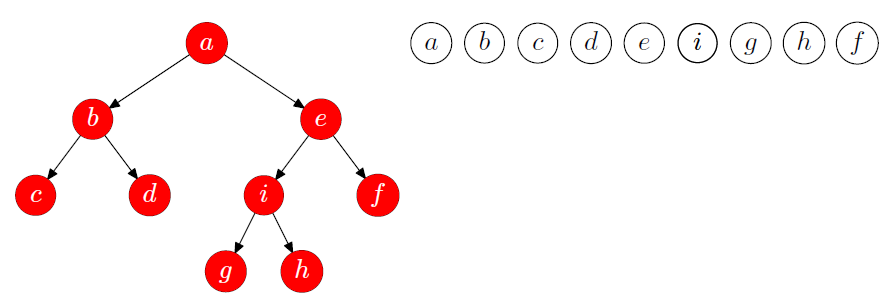
a gyökérelemet a sor végére).

* Járjuk be a gyökérelem bal oldali részfáját preorder

módon.

* Járjuk be a gyökérelem jobb oldali részfáját preorder

módon.



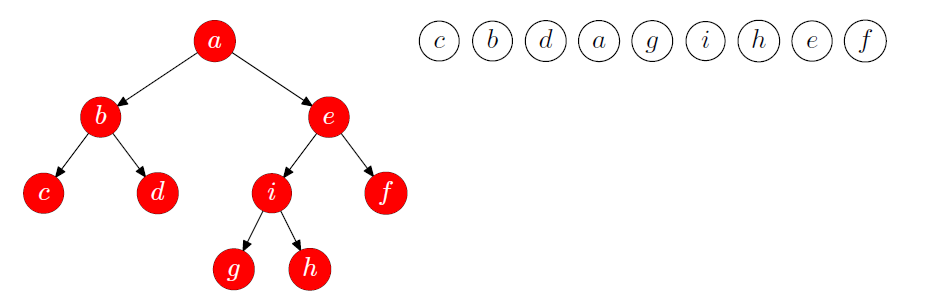
**Inorder bejárás algoritmusa**

* Ha a bejárandó fa üres, az algoritmus véget ér.
* Járjuk be a gyökérelem bal oldali részfáját inorder módon.
* Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük

a gyökérelemet a sor végére).

* Járjuk be a gyökérelem jobb oldali részfáját inorder

módon.



**Postorder bejárás algoritmusa**

* Ha a bejárandó fa üres, az algoritmus véget ér.
* Járjuk be a gyökérelem bal oldali részfáját postorder

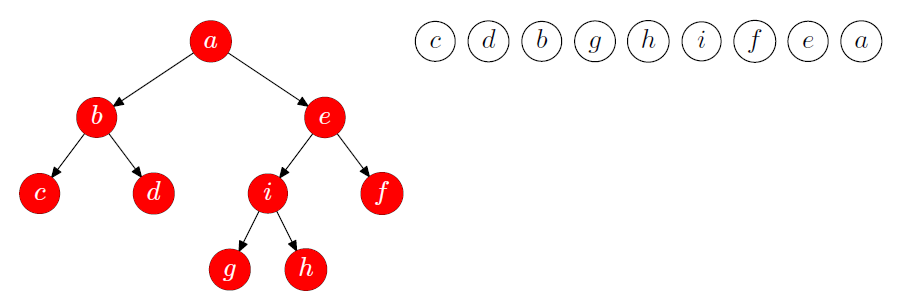
módon.

* Járjuk be a gyökérelem jobb oldali részfáját postorder

módon.

* Dolgozzuk fel a gyökérelemet (más szavakkal: helyezzük

a gyökérelemet a sor végére).



**Kupacosítás**

Legyen i az A tömb egy olyan elmének az indexe, amelyre:

– Az A[i] elem baloldali és jobboldali részfái kupacok.

– Elképzelhető viszont, hogy A[i] kisebb mint a gyerekei,

vagyis sérül a kupac tulajdonság.

A kupacosítás során az i. pozíción lévő bináris részfából kupacot csinálunk úgy, hogy az A[i] elemet lefelé mozgatjuk a kupacban.

A kupac tulajdonság sérül az 1-es indexű elem esetében. A 2-es és 3-as indexű részfák viszont kupacok.

kupacosít(1)

**Maximális elem törlése (#1)**

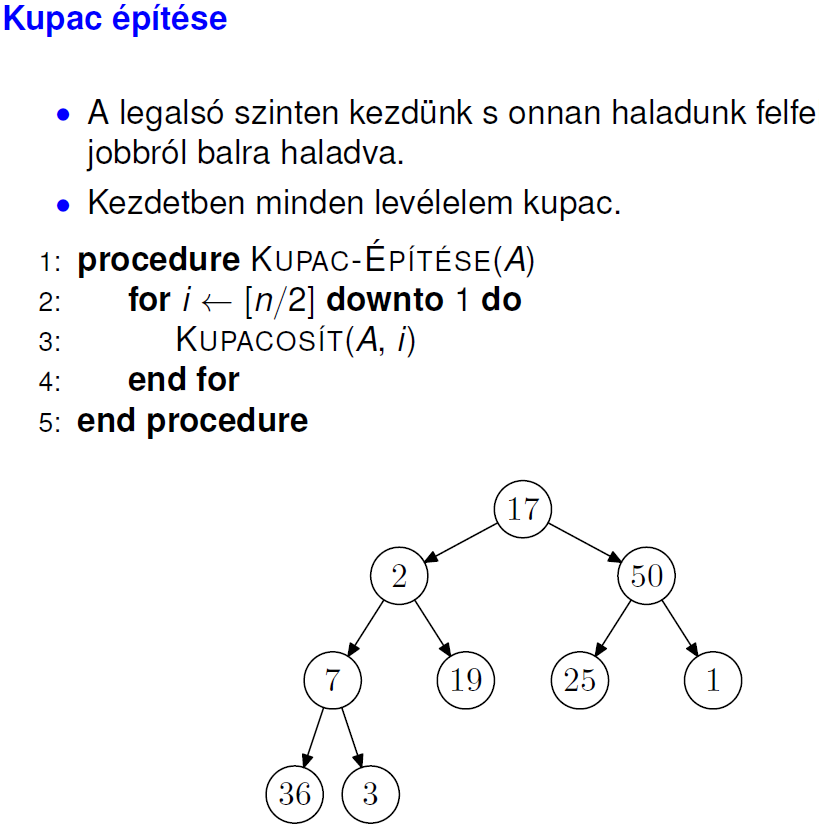
A legnagyobb elem a kupac tetején található. Ez a kupac gyökere.

Ezt kitörölhetjük, s felhozzuk a helyére az egyik gyerekét.

Az üres hely lefelé mozog a fában.

Az üres hely bárhová kerülhet az utolsó szinten.

Az így létrejövő fa utolsó szintjén az elemek jó eséllyel nem lesznek balra rendezettek, vagyis sérül az alak tulajdonság.



**Beszúrás piros-fekete fába (Okasaki-módszer)**

Egy piros-fekete fát úgy bővítünk, mint egy keresőfát: mindig levélelemmel, amelyet pirosra színezünk. A levélelemmel történő bővítést követően a következő esetek fordulhatnak elő:

1. A fa továbbra is rendelkezik a piros-fekete

tulajdonságokkal. Ekkor nincs teendőnk, készen vagyunk.

1. Nem teljesül a 2-es tulajdonság, miszerint a gyökérelem

fekete. Ez csak akkor fordulhat elő, ha éppen a gyökeret

szúrtuk be, azaz a fa előzőleg üres volt. Ekkor átszínezzük

a beszúrt (gyökér)elemet feketére, és készen vagyunk.

1. Nem teljesül a 4-es tulajdonság, miszerint nincs a fában

két egymást követő piros csomópont. Ez csak akkor

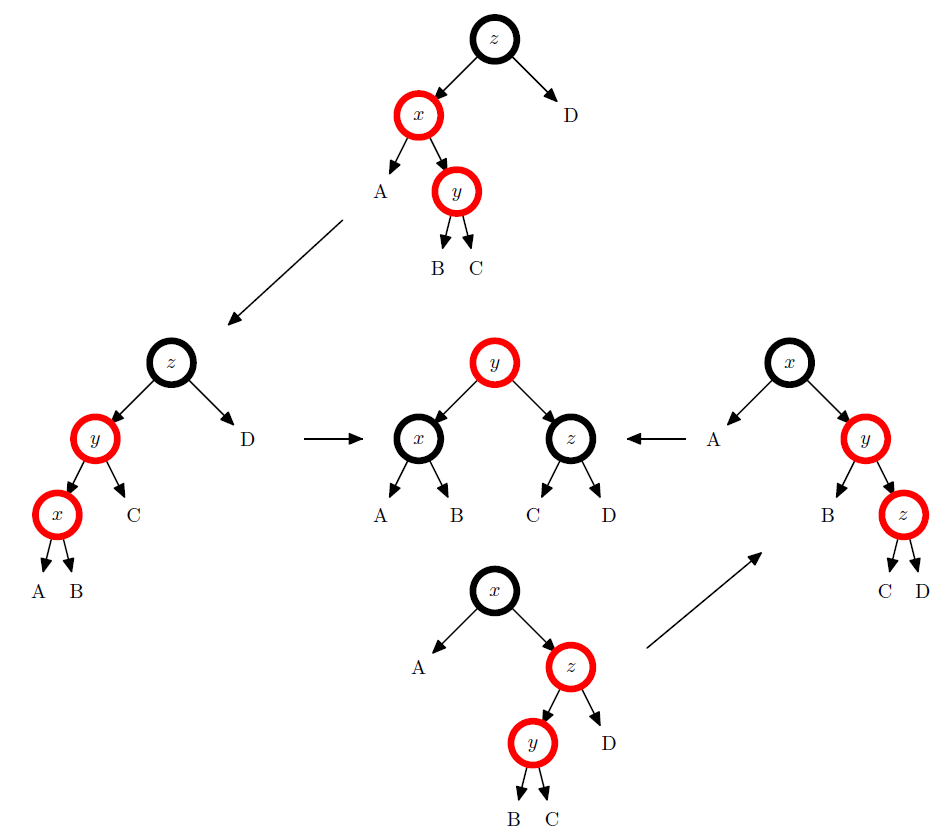
fordulhat elő, ha a beszúrt elem szülője is piros. Mivel a

gyökér fekete, a beszúrt elemnek biztosan létezik

nagyszülője, amelynek a 4-es tulajdonság miatt feketének

kell lennie. Ekkor forgatásokat és átszínezéseket kell

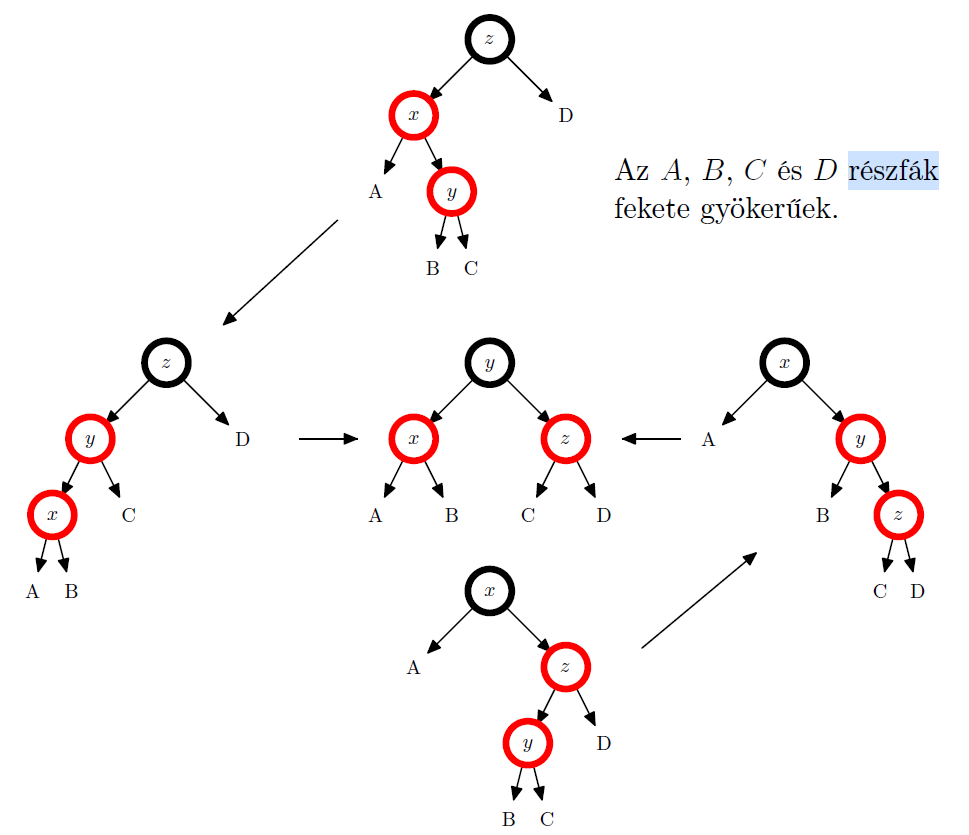
végrehajtanunk.



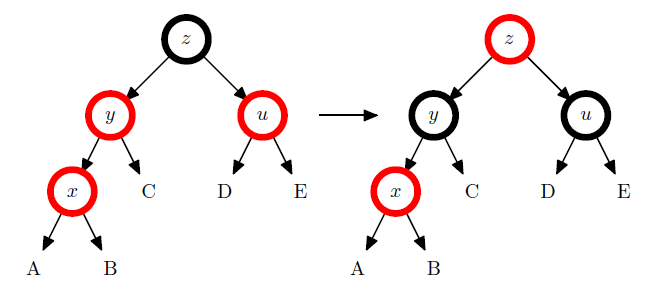
A fenti transzformáció (egy vagy két forgatás, valamint egy átszínezés) után az y szülőjéből (ha létezik) bármelyik levélbe vezető úton ugyanannyi fekete elem lesz, mint amennyi a transzformáció előtt volt. Az így kapott fa már vagy piros-fekete fa, vagy nem teljesül a 2-es tulajdonság (ha y a gyökérelem), vagy nem teljesül a 4-es tulajdonság (ha y szülője piros). A fenti transzformációt tehát addig kell ismételnünk, amíg y szülője fekete nem lesz (ekkor nincs további teendőnk), vagy y a gyökér nem lesz. Utóbbi esetben átszínezzük y-t feketére, és készen vagyunk. (A gyökérelem feketére színezésével a gyökérből az egyes levelekbe vezető utak mindegyikén ugyanannyival nő a fekete csomópontok száma, tehát az 5-ös tulajdonság továbbra is fennáll.)

**Beszúrás piros-fekete fába (CLRS-módszer)**

A CLRS-módszer abban különbözik az Okasaki-módszertől, hogy hogyan kezeli azt az esetet, amikor a beszúrás után nem teljesül a 4-es tulajdonság. Ekkor két esetet különböztetünk meg attól függően, hogy a beszúrt elem nagybácsija (a szülőjének a testvére) piros-e vagy fekete. Tételezzük fel először, hogy fekete! Ekkor hasonló forgatásokat hajtunk végre, mint az Okasaki-módszer esetén, viszont utána az y csomópont lesz fekete, míg a két gyermeke (x és z) piros. Ezáltal biztosan teljesülni fog a 4-es tulajdonság, így nincs szükség további forgatásokra és átszínezésekre (természetesen a 2-es tulajdonság is teljesül).



Mi történik akkor, ha a beszúrt elem nagybácsija piros? Ebben az esetben a beszúrt elem szülőjét és annak testvérét (a nagybácsit) feketére színezzük, a szülőjüket pedig pirosra. Forgatást ilyenkor nem kell végrehajtani. Az egyik lehetséges esetet szemlélteti a következő ábra:



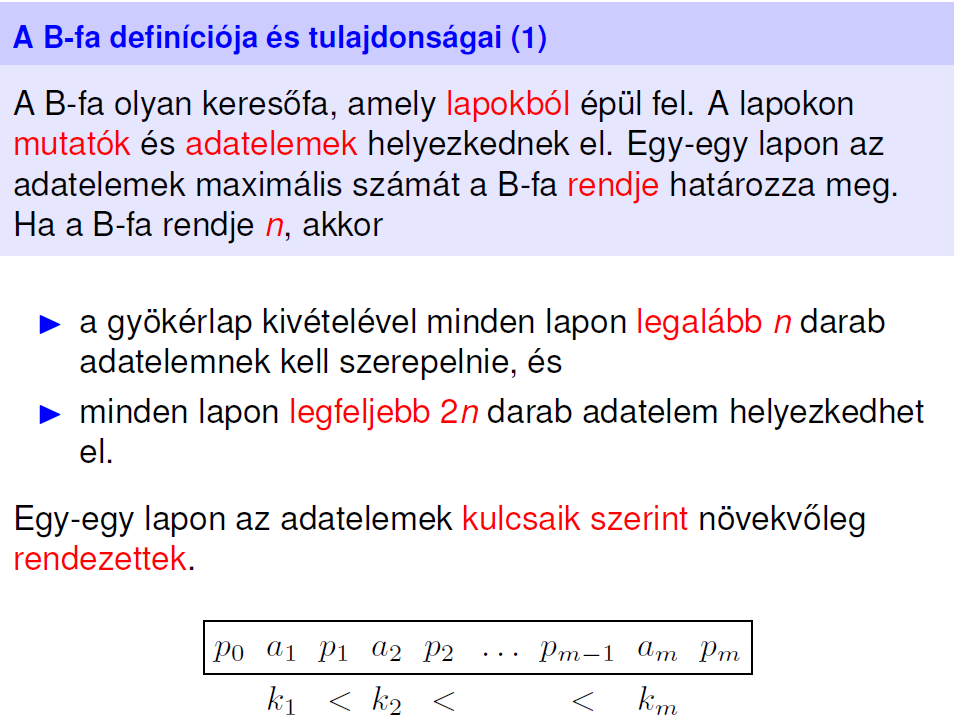
Könnyen látható, hogy az átszínezés után nem változik a gyökérből a levelekbe vezető utakon a fekete elemek száma. Előfordulhat viszont, hogy nem teljesül a 2-es vagy a 4-es tulajdonság. Az eljárást tehát mindaddig kell ismételnünk, amíg (i) z szülője fekete nem lesz (ekkor készen vagyunk), (ii) z a gyökér nem lesz (amit átszínezünk feketére, és készen vagyunk), vagy (iii) z nagybácsija fekete nem lesz (ekkor végrehajtunk egy vagy két forgatást, és készen vagyunk).

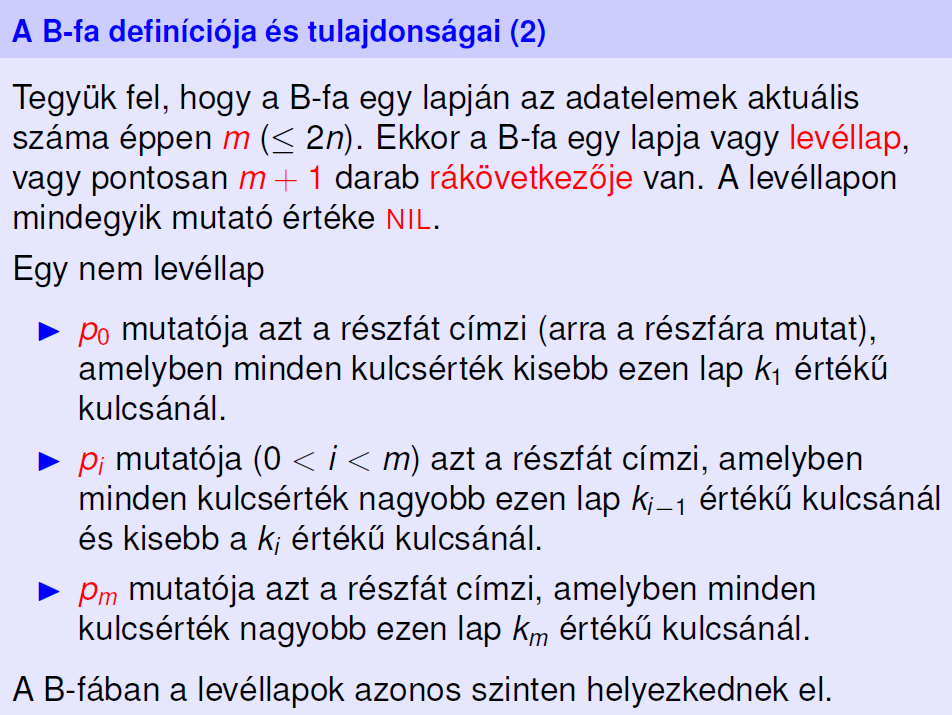
**Törlés piros-fekete fából**

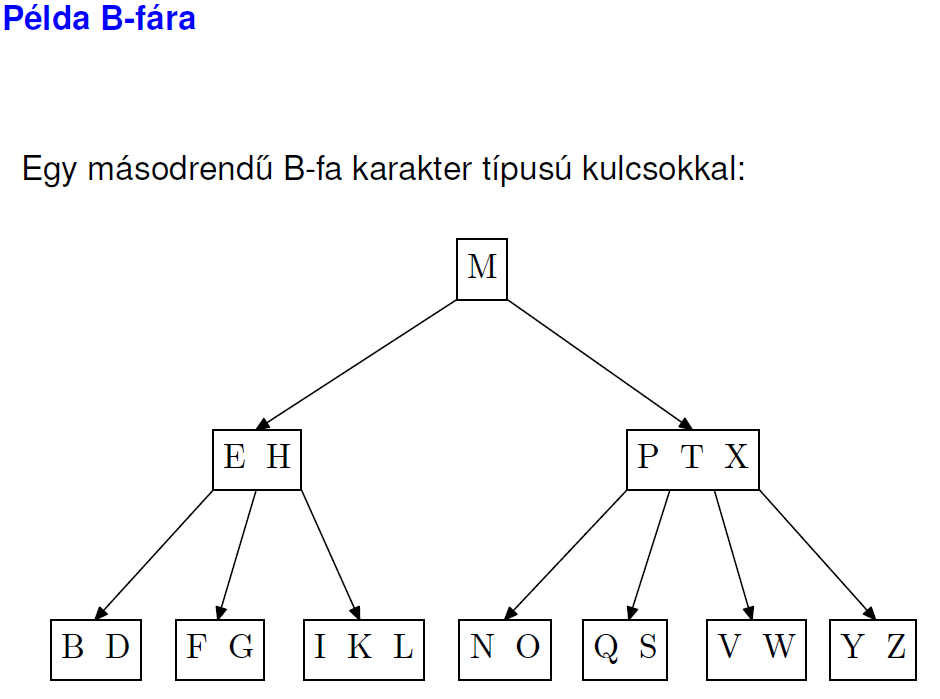
Egy piros-fekete fából ugyanúgy törlünk, mint egy keresőfából. Az a csomópont, amelyet eltávolítottunk a fából, nem feltétlenül az a csomópont, amely a törölt adatelemet tartalmazta. A piros-fekete tulajdonságok helyreállításához az eltávolított csomópontot kell figyelembe vennünk. Legyen ez a csomópont v, a szülője pedig p(v)! Az eltávolított csomópont (v) legalább egyik gyermekének levélnek kell lennie. Ha v-nek van egy nem levél gyermeke, akkor a helyét az a bizonyos gyermek, különben pedig egy levélelem veszi át. Legyen u az a gyermek, amelyik v helyére kerül a törlés után! Ha u levél, akkor tudjuk, hogy fekete. Ha v piros, akkor készen vagyunk, mivel egyetlenegy piros-fekete tulajdonságot sem sértettünk. Tehát tegyük fel, hogy v fekete!

A gyökérből a levelekbe vezető azon utak, amelyek keresztülmennek v-n, eggyel kevesebb fekete csomópontot fognak tartalmazni, mint a többi gyökér-levél út a fában, és ez megsérti az 5-ös tulajdonságot. Ha p(v) és u is piros, akkor a 4-es tulajdonságot is megsértjük, de látni fogjuk, hogy az 5-ös tulajdonság helyreállítása a 4-es tulajdonságot is helyreállítja további teendők nélkül, ezért mi most az 5-ös tulajdonság helyreállítására koncentrálunk. Képzeljük el, hogy egy fekete tokent rendelünk u-hoz! Ez a token azt jelzi, hogy az ezen a csomóponton átmenő, levélig vezető utak eggyel kevesebb fekete csomópontot tartalmaznak, mint kellene. (Kezdetben ez azért van, mert v-t kitöröltük.) A tokent a fában egyre feljebb visszük, amíg ki nem alakul egy olyan helyzet, amelyben az 5-ös tulajdonságot helyreállíthatjuk. Ezt a tokent egy kis fekete négyzettel jelöljük az ábrákon. Ha a tokennel rendelkező csomópont fekete, akkor azt duplán fekete csomópontnak nevezzük. (A token csak egy fogalmi eszköz, fizikailag nem jelenik meg az adatszerkezetben.)

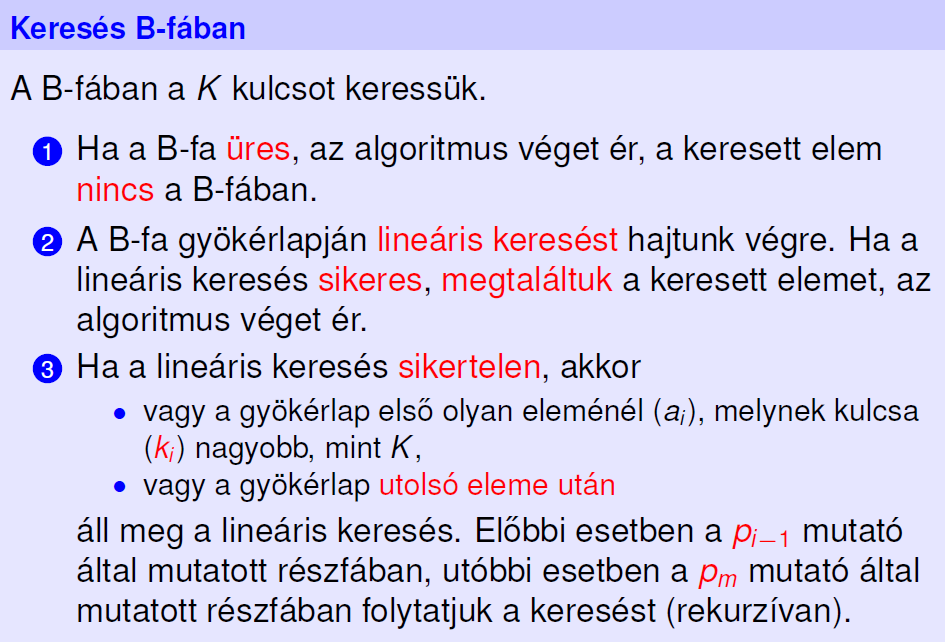
**B-fa definíciója és tulajdonságai / B fa definíciója és tulajdonságai**



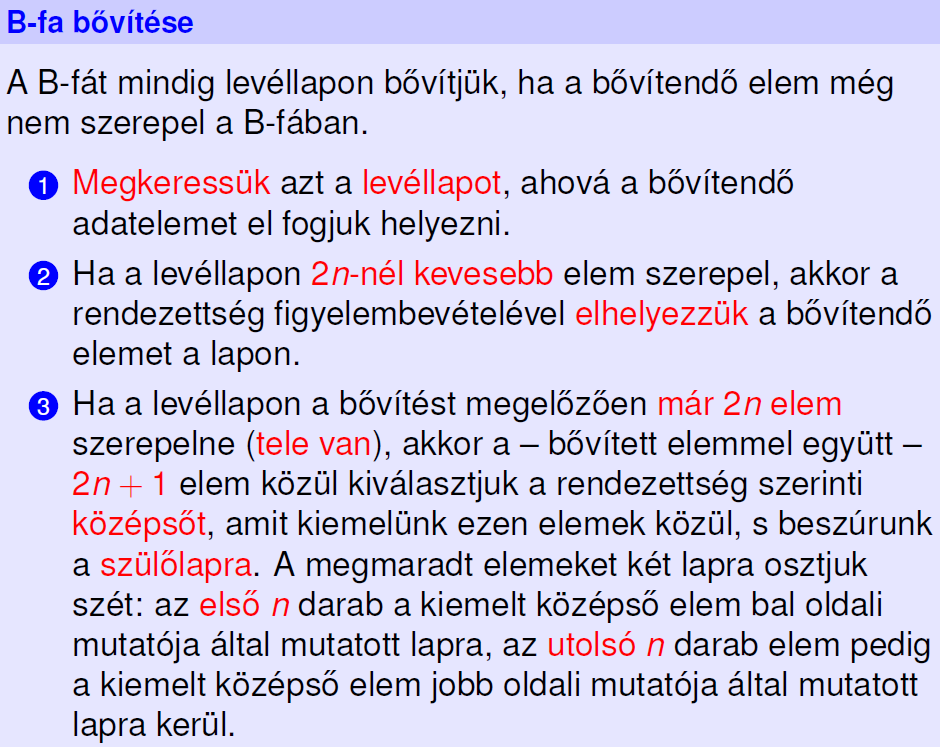




**Keresés B-fában / Keresés B fában**



**B-fa bővítése / B fa bővítése**



**Törlés B-fából / Törlés B fából**

