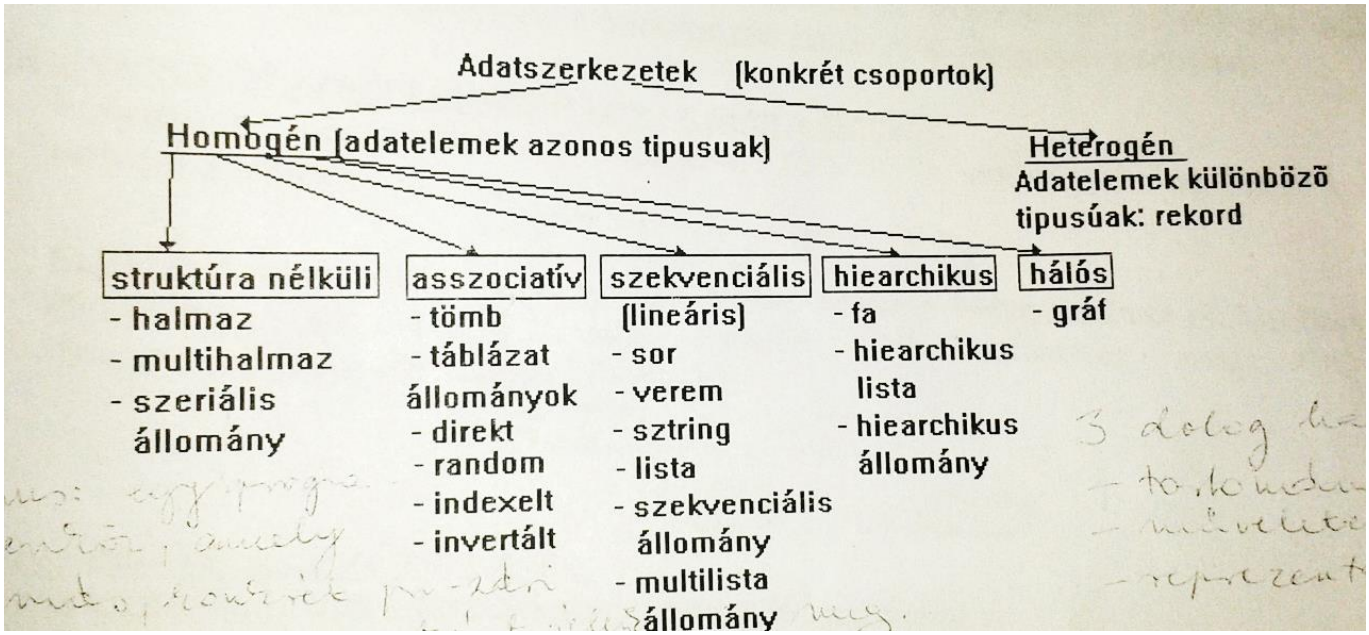
Adatszerkezetek

Adatelem: legkisebb adategység, amelyre hivatkozni lehet.

Adatszerkezet: adatelemek véges halmaza, amelyben az adatelemek között szerkezeti összefüggések vannak.

- Az összekapcsolás módja határozza meg az elemek egymáshoz való viszonyát, illetve a műveleteket.
- <adatelemek halmaza, reláció>
- homogén: azonos típusú elemek.
- heterogén: különböző típusú elemek.



Struktúra nélkül: az egyes adatok között nincs kapcsolat. nincsen sorrendje az elemeknek.

Asszociatív: az egyes adatok között nincsen kapcsolat. Valamilyen közös tulajdonság alapján összeállított halmaz, melyekből valamilyen ismeretek alapján részhalmazokat választunk ki.

Szekvenciális: az egyes adatelemek egymás mellett helyezkednek el.

> **Az adatok között egy-egy jellegű kapcsolat van:** minden adatelem csak egy helyről érhető el, és az adott elemről csak egy másik elem látható. Az első és utolsó elem kitüntetett szerepű.

Hierarchikus: az adatelemek között egy a többhöz kapcsolat van. Minden adatelem csak egy helyről érhető el, de egy adott elemből bármennyi adatelem látható. Egy kitüntetett eleme van, amelynek nincs megelőzője és több olyan elem van, amelynek nincs rákövetkezője.

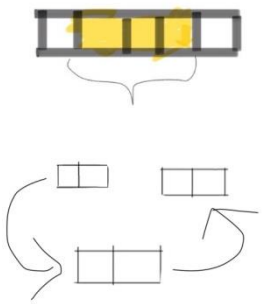
Hálós: a hálós adatszerkezet <Adat, Reláció> rendezett pár (az adatok között a kapcsolat több a többhöz.

Adatelemek száma szerint:

Statikus: rögzített számú adatelem van benne. Az adatelemek futás közben csak az értéket változtathatják meg, és a szerkezet és az elemek száma változatlan.

Dinamikus: az adatelemek száma véges, de a futás során változhat.

Reprezentáció szerint (hogyan jelenik meg a memóriában):



- **Folytonos:** egymás után helyezkednek el.
- **Szétszórt:** az elemek véletlenszerűen helyezkednek el, közöttük a kapcsolatot az teremti, hogy minden elem tartalmaz más elemek elhelyezkedésére információt.

Műveletek:

1. **Létrehozás**
2. **Módosítás:**
3. **Bővítés, beszúrás:** új adatelemek kerülnek az adatszerkezetbe.
4. **Törlés:**
   - **fizikai:** az adatelemek száma csökken.
   - **logikai:** nem csökken az adatelemek száma, hanem a törlendő elemeket megjelöljük, és utána nem tudjuk használni.
5. **Csere:** az adatelemek száma nem változik csak valamely elem vagy elemek értéke.
6. **Rendezés**
7. **Keresés**
8. **Bejárás:** az adatszerkezet minden elemét elérjük valamilyen sorrendben. A sorrend lehet pl. fizikai sorrend, szekvenciális, vagy közvetlen.

Homogén adatszerkezet

Struktúra nélküli: az adatelemek között nincs kapcsolat. Az egyes adatelemekből lehet megmondani, hogy elemei-e az adatszerkezetnek.

Halmaz: a matematikai halmaz megjelenése adatszerkezet szinten.

Multihalmaz: olyan halmaz, amelyben lehet ismétlődés.

Megjegyzés: üres halmazt is tekinthetünk, de végtelen halmazt nem.

Asszociatív adatszerkezet:

Tömb: az egy dimenziós tömböt vektornak nevezzük. A két dimenziós tömb a mátrix.

Műveletek:

1. **Létrehozás:** a tömb szerkezete létrejön.
2. **Módosítás:** nincs.
3. **Bővítés:** nincs.
4. **Törlés:** csak logikai törlés.
5. **Csere:** a tömb egy adott elemének értékét lehet cserélni másikra (pl. értékadással)
6. **Rendezés:** valamilyen rendezési algoritmussal.
7. **Keresés:** lineáris vagy bináris keresés.
8. **Bejárás:** az elemek indexének segítségével.

A kétdimenziós tömb (mátrix) elemeinek bejárása és tárolása kétféleképpen történhet: sorfolytonos, oszlopfolytonos. Példa mátrixra:

15	19	23
20	40	44
13	30	57

Ha a mátrix sorfolytonosan van tárolva, akkor az i, j elem címét a következőképpen érhetjük el.

k+l*(i-s)*(m-t+l)+l*(j-t) , ahol **'k'**, a tömb kezdő eleme
'l', egy elem hossza
's', a sorok száma
'm', az oszlopok száma

Ha a sorok és oszlopok száma megegyezik, akkor négyzetes mátrixot kapunk.

Főátló: A bal felső saroktól jobb alsó sarokig terjedő elemek.

15	19	23
20	**40**	44
13	30	**57**

Mellékátló: A jobb felső saroktól a bal alsó sarokig terjedő elemek.

15	19	**23**
20	**40**	44
13	30	57

Egységmátrix: A főátlóban csak 1-esek vannak, a többi helyen pedig 0.

1	0	0
0	1	0
0	0	1

Felső háromszög mátrix: A főátló alatt csak 0-k vannak.

10	17	14
0	15	30
0	0	23

Alsó háromszög mátrix: A főátló felett csak 0-k vannak.

10	0	0
43	15	0
25	16	23

Táblázat: Minden elem két részből áll: Érték, kulcs. A kulcs és az érték típusú bármilyen lehet, de a táblázaton belül a kulcsok és értékek típusa meg kell, hogy egyezzen. A kulcs értékének egyedinek kell lenni. A táblázat tulajdonképpen a tömb általánosításának tekinthető, ahol bármely elem a kulcson keresztül érhető el. A tárolási módja lehet szétszórt és folytonos is, de általában az utóbbit alkalmazzuk.

Gyakran úgy van megoldva, hogy a kulcsot és az értéket külön tároljuk egy vektorban.

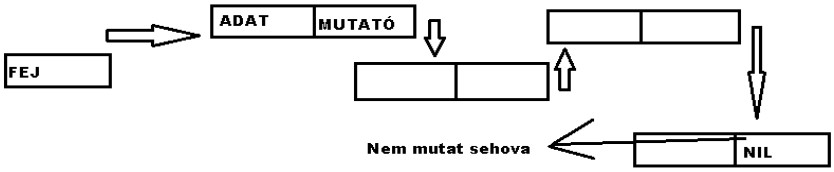
Műveletek:

1. **Létrehozás:** A táblázat szerkezete létrejön.
2. **Bővítés:** Az elemek az érkezés sorrendjében kerülnek a táblázatba, az utolsó elem után.
3. **Törlés:** Lehet fizikai és logikai törlés is.
4. **Módosítás:** Bővítés/törlés
5. **Csere:** Bármely elem felülírható.
6. **Rendezés:** Nem rendezett a táblázat.
7. **Keresés:** Kulcs alapján.
8. **Bejárás:** Sorban lehetséges.

Szekvenciális (lineáris) **adatszerkezet:** az adatelemek mindig két másik elemmel vannak kapcsolatban, kivéve a két szélső elemet.

Lista (list)**:** olyan dinamikus adatszerkezet, amelynek létezik egy első és utolsó eleme (kivéve az üres lista), és minden elemnek van rákövetkezője (kivétel az utolsó elem), továbbá minden elemnek van megelőzője (kivétel az első elem). A lista elemeinek van rendezettsége, amelyet az elemek elhelyezkedése biztosít.

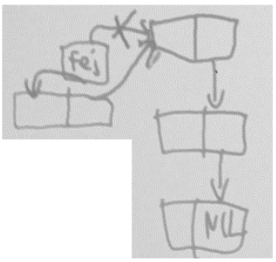
Egy irányban láncolt lista (linked list)**:**



Minden elemnél a mutatórész a következő elem tárolási címét, az adatrész pedig az adatelem értékét tartalmazza. A lista első eleme kitüntetett szerepű. Ennek a címét a fej tartalmazza. A NIL (vagy NULL) az üres mutatót jelenti.

Műveletek:

1. **Létrehozás:** Az üres lista létrehozása
2. **Bővítés:**

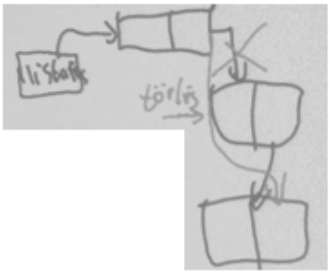



```
Eljárás bővítés
    Be: elem
    elem->következő=listafej->következő
    listafej->következő=elem
Eljárás vége
```

Megjegyzés: Ez a lista első helyére illeszt be elemet

3. **Törlés:**

```
Eljárás törlés
    Be: elem
    elozo=NIL
    i=listafej
    Ciklus amíg i!=NIL és i->adat!=elem
        elozo=i
        i=i->kovetkezo
    Ciklus vége
    Ha i!=NIL akkor
        elozo->kovetkezo=i->kovetkezo
    Elágazás vége
Eljárás vége
```



4. **Keresés:**

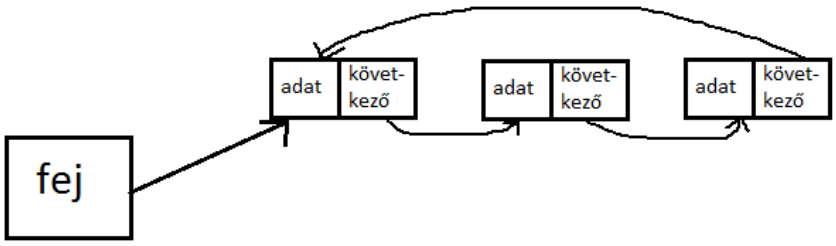
```
Eljárás keresés
   Be: elem
   i=listafej
   Ciklus amíg (i!=NIL és i->adat!=elem)
        i=i->kovetkezo
  Ciklus vége
  Ha i!=NIL
        Ki: i
  Különben
        Ki: nem létezik az elem a listában
  Elágazás vége
  Eljárás vége
```

5. **Bejárás:**

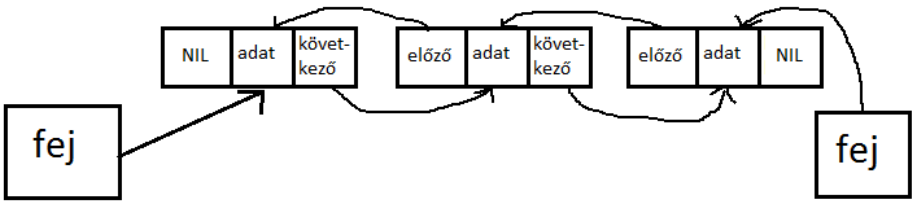
```
Eljárás bejárás
    i=listafej
    Ciklus amíg i!=NIL
         Ki: i->adat
         i=i->kovetkezo
    Ciklus vége
Eljárás vége
```

6. **Csere:** A lista adatrésze felülírható, miután megkerestük

Körkörös lista (cirkuláris lista):
Az utolsó elem mutatója az első elem címére mutat. Bármely elemből körbe lehet menni a listán, amely a bejárást egyszerűsíti.



Kétirányban láncolt lista:
Minden elemnek két mutatója van, amely a következő és az előző elemről tartalmaz információt. Általában két fej van, az egyik a lista elejére, a másik a lista végére mutat, ezért két irányban is lehet haladni.



Absztrakt adatszerkezet: Az adatszerkezet megvalósításától (implementációjától) eltekintünk. Azért szoktunk eltekinteni ettől, mert a megvalósítás mindig rendszerfüggő. Itt csak a tulajdonságokat adjuk meg.

Verem (stack): A tömb után a leggyakrabban alkalmazott adatszerkezet a tömb után. A verem egy speciális lista. Két műveletet tekintünk:

PUSH: az adatelemet betesszük a verem „tetejére”
POP: az adatelem kivétele a verem „tetjéről”
Üres-e a verem?
Hány elemet tartalmaz?

- Létezik üres verem is (nincs egy elem sem), és létezik olyan verem, amely tele van.
- Az üres verem onnan ismerhető fel, hogy a fej NIL.
- Csak az utolsónak behelyezett elemhez férhetünk hozzá.
- Ha az alatta lévőkhöz akarunk hozzáférni, akkor először a POP művelettel ezeket el kell távolítani.
- Last in first out (LIFO) adatszerkezet.
- Használata: rekurzív függvény, fordított sorrend előállítása
- Rekurzív függvény: olyan függvény, amely önmagát hívja meg.

Példa rekurzív függvényekre:

Faktoriális értékét kiszámító függvény mondatszerű leírásban leírva:

```
Függvény faktoriális(n)
Ha n==0 vagy n==1
  faktoriális=1
Különben
  faktoriális=n*faktoriális(n-1)
Elágazás vége
Függvény vége
```

Faktoriális értékét kiszámító függvény C#-ban leírva:

```
public static long fakt(int n)
{
  if (n==0 || n==1)
       return 1;
  else
       return n*fakt(n-1);
}
```

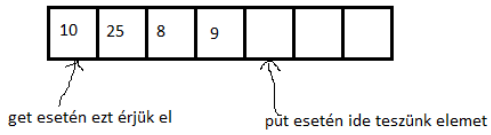
Fibonacci sorozat értékét kiszámító függvény mondatszerű leírásban leírva:

```
Függvény fib(n)
  Ha n==0 || n==1
       fib=1
  Különben
       fib=fib(n-1)+fib(n-2)
  Elágazás vége
Függvény vége
```

Fibonacci sorozat értékét kiszámító függvény C#-ban leírva:

```
public static int fib(int n)
{
  if (n==0 || n==1)
       return 1;
  else
       return fib(n-1)+fib(n-2);
}
```

Sor: Egy speciális lista, amelynek két művelete van:
PUT: adatelem betevése a sorba
GET: adatelem kivétele



A sor FIFO (First in first out): azt az elemet lehet kivenni, amelyik először került bele.
Létezik üres sor és tele sor is. A sor **reprezentációja** kétféle lehet:
Folytonos: vektorra képezzük le a sort.
Szétszórt: egyirányban láncolt listára képezzük le.
Egy lehetséges megvalósítás: Az elemeket beletesszük egy tömbbe, és mellé két mutatót rendelünk, az egyik a sor elejére mutat, a másik a sor végére. (Vándorló sor)

String: Egy olyan lista, amelynek elemei betűkből állnak. Műveletek, fogalmak string-ekkel kapcsolatban:
Összefűzés (konkatenáció)
Hossz (length): A string karaktereinek száma
Rész string: A stringből kiveszünk bizonyos karaktereket
Mintaillesztés: Rész string-et keresünk a string-ben.
Reprezentáció lehet szétszórt és folytonos is.

Hierarchikus adatszerkezetek

Fa (tree): Dinamikus, homogén adatszerkezet, amelyben minden elem megmondja a rákövetkezőjét.

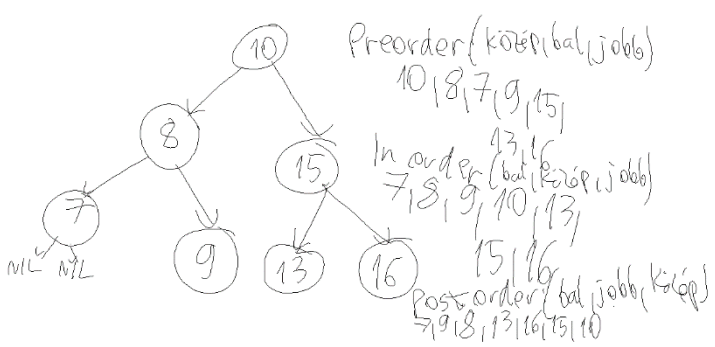
Alapfogalmak:
Gyökérelem: A fa azon eleme, amelynek nincs megelőzője. A legegyszerűbb fa csak egy gyökérből áll, és semmi más elem nincs benne. Az üres fában nincsen gyökérelem sem.
Levélelem: A fa azon eleme, amelynek nincs rákövetkezője. Ebből bármennyi lehet.
Közbenső elem: A fa azon eleme, amely se nem gyökérelem, se nem levélelem.
Út: A gyökérelemtől kiinduló, különböző szinteken átmenő és levélelemben véget érő egymáshoz kapcsolódó élsorozat. Az út hosszán az útban szereplő élek számát értjük. Minden levélelem a gyökértől pontosan egy úton érhető el. A fában annyi út van, amennyi levélelem.
Szint: A 0. szinten a gyökérelem van, az 1. szinten a gyökérelem rákövetkezői, a 2. szinten ennek rákövetkezői… A maximális szint számot a fa **magasságának** vagy **mélységének** nevezzük.

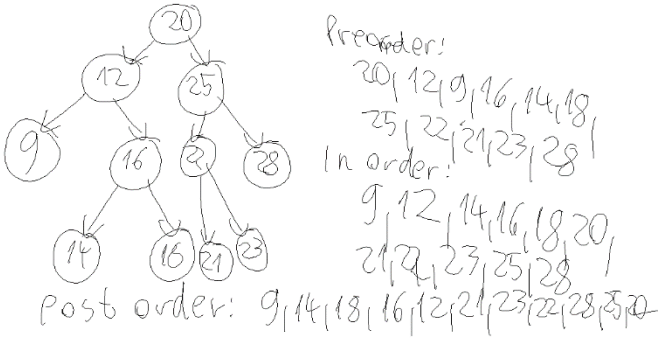
Műveletek:
Létrehozás: Az üres fát létrehozzuk
Bővítés: levélelemmel, vagy részfával
Törlés: levélelem vagy részfa
Bejárás: preorder, in order, post order.
Csere: Az elemet megkeressük, és utána írjuk felül az értékét

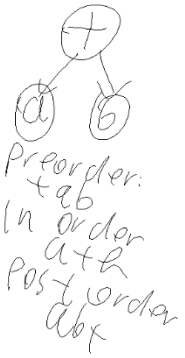
Bináris fa: Olyan fa, amelynél minden elemnek vagy 0 vagy 2 rákövetkezője van.
Van olyan bináris fa, amely rendezett. (rendezett bináris fa)
Minden fa reprezentálható bináris fa segítségével.

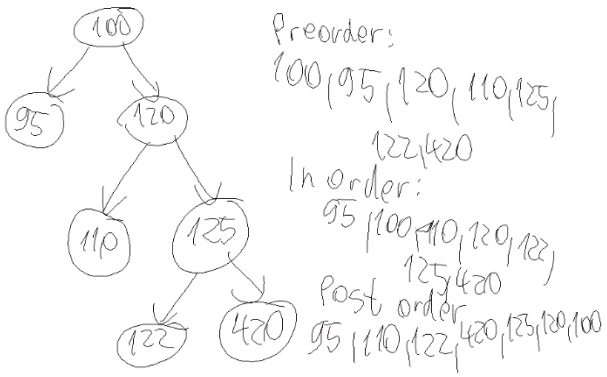
Bináris keresőfa: Olyan bináris fa, amelynek bal oldali részfájában csak az elemtől kisebb, a jobb oldali részfájában csak az elemnél nagyobb elemek vannak.

Példák bináris keresőfára és azoknak a preorder, in order és post order bejárására:


10
8
15
7
9
13
16
NIL
NIL
Preorder(közép|bal|jobb)
10|8|7|9|15|13|16
In order(bal|közép|jobb)
7|8|9|10|13|15|16
Post order(bal|jobb|közép)
=>9|8|13|16|15|10


20
12
25
9
16
21
28
14
16
21
23
Preorder:
20|12|9|16|14|18|25|22|21|23|28|
In order:
9|12|14|16|18|20|21|22|23|25|28
post order: 9|14|18|16|12|21|23|22|28|25|20


+
a
b
Preorder:
+ab
In order
a+b
Post order
ab+


100
95
120
110
125
122
420
Preorder:
100|95|120|110|125|122|420
In order:
95|100|110|120|122|125|420
Post order
95|110|122|420|125|120|100