1. **Rekurzió**

* egy olyan művelet, mely végrehajtásakor a saját maga által definiált műveletet, vagy műveletsort hajtja végre, ezáltal önmagát ismétli
* leggyakrabban a backtrack típusú algoritmusokban szokták használni
* minden rekurzió ciklussá alakítható, illetve minden ciklus rekurzióvá
* előfordulthat, hogy a hívás túl sokszor fordul elő, ez végtelen rekurziót eredményez, ennek jellegzetessége a verem túlcsordulása, másnéven *stack overflow*

1. **Vermek, sorok**

* a **verem** egy LIFO típusú struktúra, mely véges számú és azonos típusú adatot tárol
* a veremre értelmezett műveletek: **push, pop, isFull, isEmpty**
* szokásos megvalósítása egy véges méretű, összefüggű memóriaterület és egy veremmutató segítségével történik
* mindig az egyik végéről töltjük fel, míg a veremmutató a legfelső elemre mutat
* a **sor** egy FIFO típusú struktúra, melynél mindig a legelsőként elhelyezett elemet vehetjük ki legelőször, így nem érhető el minden eleme közvetlenül
* a soron értelmezett műveletek: **put, top, get, isEmpty**

1. **Láncolt listák, körkörösen láncolt listák, fák**

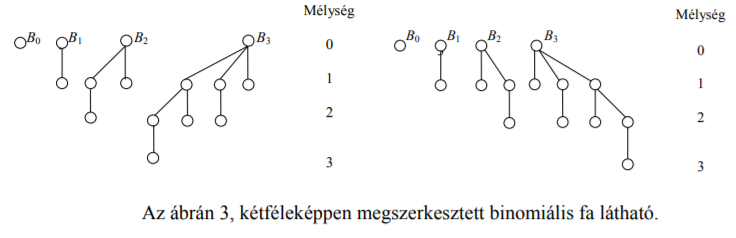
* a **láncolt lista** egy gyakran használt adatszerkezet
* cellák sorozatából épül fel, amelyek tetszőleges számú adatmezőt és egy vagy két hivatkozást tárolnak
* a hivatkozások a lista következő és előző elemére mutatnak
* a láncolt szerkezet lehetővé teszi tetszőleges helyre elemek beszúrását konstans időben
* változatok: egyszeresen láncolt, kétszeresen láncolt, körkörösen láncolt lista
* a **körkörösen láncolt** lista az egyszerűbb listák (egyszeresen és kétszeresen láncoltak) kibővítése
* eltérése, hogy az első és utolsó elem össze van kötve egymássl
* a bejárást bárhol kezdhetjük, viszont vigyáznunk kell a körbeérésre
* a **fák** körmentes, összefüggő gráfok
* a *gyökérelem* a fa azon eleme, melynek nincs őse
* a *levél* a fa azon eleme, melynek nincs utóda
* a *közbenső elemek* a fa nem gyökér- illetve levélelemei
* az *út* a gyökérből kiinduló bejárás

1. **Bináris fák és bináris keresőfák**

* a **bináris fa** olyan speciális fa, mely esetén minden csúcs legfeljebb két gyerekkel rendelkezik
* egy speciális esete a **kiegyensúlyozott bináris fa**, mely esetén minden csúcsnak pontosan 0 vagy 2 gyereke lehet, illetve bármely csúcspont két részfájának magassága közti különbség legfeljebb egy
* a **keresőfa** olyan fa, melynek bármely elemére igaz, hogy az elem baloldali részfájában az összes kulcs kissebb, a jobboldali részfájában az összes elem kulcsa nagyobb mint az adott csúcs kulcsa

1. **Kupacok, binomiális fák**

* a **kupac** egy speciális fa alapú adatszerkezet, mely eleget tesz a kupac tulajdonságnak, azaz ha **B** csúcs fia az **A** csúcsnak, akkor kulcs(A) >= kulcs(B), ebben az esetben a kupacot *max-kupacnak* nevezzük
* az összehasonlítás megfordításával *min-kupacot* kapunk
* a kupac adatszerkezet különböző fajtáit több algoritus hatékony implementációja során alkalmazhatjuk:
  + heap sort
  + kiválasztó algoritmusokba a k-adik legkissebb elem megkeresése lineáris időben
  + súlyozott gráfok bejárási algoritmusának gyorsítása
* a **binomiális** fát egy rekurzív definícióval adhatjuk meg:
  + a B0 fa egyetlen pontot tartalmaz
  + a Bk (k>0) pedig két összekapcsolt Bk-1 fából áll, ahol a fa gyökércsúcsa a másik fa gyökércsúcsának legbaloldalibb gyereke lesz

****

1. **Rendezési algoritmusok**

* egy olyan algoritmus, mely valamilyen szempont alapján sorba állítja elemek egy listáját
* rendezett adathalmazzal több és hatékonyabb műveletek végezhetőek, mint rendezetlennel
* megkülönböztetünk belső és külső rendezést
* belső rendezések: *Bubble sort, Selection sort (kivalaszto), Insertion sort (beszurasos), Merge sort (osszefesulo), Heapsort (kupacrendezes), Quicksort (gyorsrendezes)*
* belső rendezések esetén a rendezés az elemek összehasonlítására alapszik, míg küldő rendezés esetén az algoritmusok nem használják az elemek között fennálló hasonlítást

1. **Hasító táblák és hasító algoritmusok**

* a hash-table egy olyan adatszerkezet, amely egy hash-függvény segtségével rendel minden értékhez egy kulcsot, mellyel későb (ideális esetben) konstans időben azonosítja az értéket, ezáltal implementálva egy asszociatív tömböt
* egy leképzést végez két halmaz között
* ideális esetben minden kulcs egyedi, valóságban történhetnek ütközések, így számolni kell a fellépő ütközéssel is
* sok esetben hatékonyabbak mint a keresőfák, ezért gyakran használják asszociatív tömbök implementációjában, adatbázisok indexelésében, illetve cache memória felépítésében
* a **hasítófüggvény** egy speciális eljárás a hasítótáblák felépítésére
* nagy méretű adatállományok adatelemeinek gyors, hatékony megkeresésére használják
* adott K érték esetén kiszámolunk egy f(K) függvényt, amely meghatározza K érték heléyt az adatállományban
* szükséges továbbá az azonos hasítóértékekkel rendelkező elemek megkülönböztetése