Adatstuktúrák

Rekurzió

A **rekurzió** a matematikában, valamint a számítógép-tudományban egy olyan művelet, mely végrehajtáskor a saját maga által definiált műveletet, vagy műveletsort hajtja végre, ezáltal önmagát ismétli; a rekurzió ezáltal egy adott absztrakt *objektum* sokszorozása önhasonló módon.

```
int faktorialis (int n)
{
    if (n<=1) {return 1;}
    return n*faktorialis(n-1);
}
```

Vermek, sorok

Verem

A számítástechnikában a **verem** (angolul *stack*) egy LIFO adatszerkezet, amelyben általában véges számú azonos típusú (méretű) adatot lehet tárolni.

Hagyományosan két alapműveletet értelmezünk rá:

- ***push*** (rárak): A verem tetejére helyez egy új adatot. Ha a verem betelt, akkor túlcsordulásos állapotba kerül.
- ***pop*** (levesz): A verem legfelső elemét leveszi és visszaadja. Ha a verem már üres, akkor alulcsordulásos állapotba kerül.

A verem szokásos megvalósítása egy véges méretű összefüggő memóriaterület és egy veremmutató segítségével történik. A memóriaterületet egyik vége felől töltjük föl, és a veremmutató mindig a legfölső elemre mutat (a két művelet során ezt egyszerűen növelni vagy csökkenteni kell).

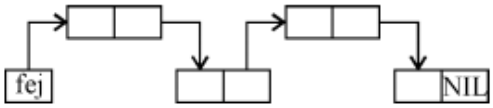
Sor

A sor adatszerkezet egy speciális szekvenciális tároló, amelyből mindig a legelsőként betett elemet vehetjük ki legelőször. Emiatt szokás a sort FIFO (First In – First Out) szerkezetnek nevezni.

Put + Get

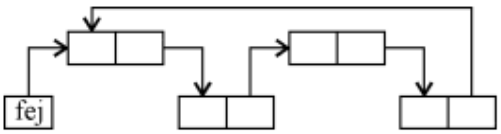
Láncolt listák, körkörösen láncolt listák, fák

Láncolt lista



Minden elemnél a mutatórész a következő elem tárolási címét, az adatrész pedig az adatelem értékét tartalmazza. Tudni kell, hogy melyik a lista első eleme. Ennek a címét a fej tartalmazza, ez listán kívüli elem, nem tartozik az adatszerkezethez. Az utolsó elem mutatórésze nem mutat sehova sem, a lista végét jelzi.
Üreslista: Csak a fej létezik a speciális, mutató értékkel. NIL

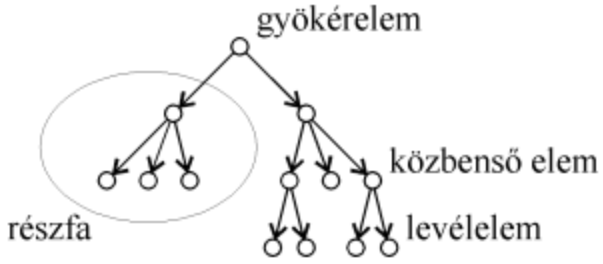
Körkörösen láncolt lista



Az utolsó elem mutatója az első elem címére mutat. Bármelyik elemből körbe tudok menni, a bejárást egyszerűsíti.

Fa

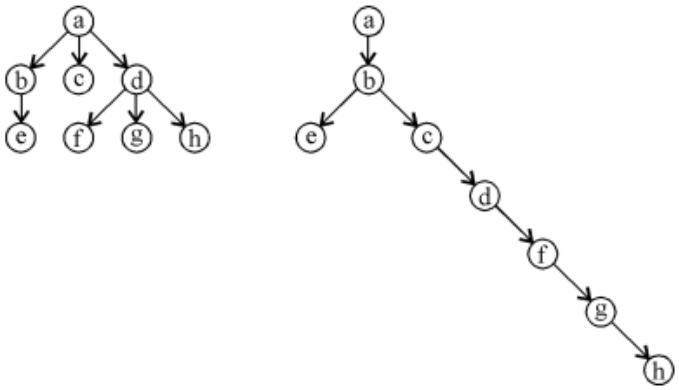
Dinamikus, homogén adatszerkezet, amelyben minden elem megmondja a rákövetkezőjét.
Alapfogalmak:
A *gyökérelem* a fa azon eleme, amelynek nincs megelőzője. A legegyszerűbb fa egyetlen gyökérből áll. Mindig csak egy gyökérelem van, üres fában egy sem.
A *levélelemek* a fa azon elemei, amelyeknek nincs rákövetkezőjük. Bármennyi lehet belőlük. Úgy érhetőek el, hogy a gyökérelemből indulva veszem a gyökérelem rákövetkezőjét, majd annak a rákövetkezőjét stb.
A fa *közbenső elemei* a fa nem gyökér- ill. levélelemei, hanem az összes többi eleme.
Az *út* a gyökérelemtől kiinduló, különböző szinteken átmenő, és levélelemben véget érő egymáshoz kapcsolódó élsorozat (lista). Az *út hosszán* az adott útban szereplő élek számát értjük. Minden levélelem a gyökértől pontosan egy úton érhető el, út helyett szokás beszélni a fa ágairól is. Egy fában az utak száma megegyezik a levélelemek számával.



Bináris fák és bináris keresőfák

Bináris fa

Bináris fa: A számítástechnikában kitüntetett szerepe van a bináris fának. A bináris fa olyan fa, amelyben minden elemnek legfeljebb két rákövetkezője lehet. Szigorú értelemben vett bináris fáról akkor beszélünk, amikor minden elemnek 0 vagy pontosan 2 rákövetkező eleme van. Rendezett bináris fáknál a rendezettség miatt az egyértelműség kedvéért beszélhetünk baloldali és jobboldali részfákról. Tetszőleges nem bináris fa reprezentálható bináris fa segítségével a következő módon:



A binárisan ábrázolandó fa gyökéreleme a bináris fában is gyökérelem lesz. Ezek után a bináris fa baloldali részfájának gyökéreleme legyen a következő szinten lévő legbaloldalibb elem. Ehhez láncoljuk hozzá az azonos szinten lévő, közös gyökerű elemeket egymás jobboldali részfáiként. Ezt a folyamatot ismételni kell az egész fára, minden szinten. A bináris fában az eredeti fa levélelemei nem feltétlenül maradnak levélelemek, viszont felismerhetőek arról, hogy nincs baloldali részfájuk. Bármely fa kezelhető bináris faként,

Bináris keresőfa

Ha adott elemszám mellett a fát úgy építem fel, hogy bármely elemére igaz, hogy az elem baloldali részfájában az összes eleme kulcsa kisebb, a jobboldali részfájában az összes eleme kulcsa pedig nagyobb az adott elem kulcsánál, akkor **keresőfáról** (vagy rendezőfáról) beszélünk.

Kupacok, binomiális fák.

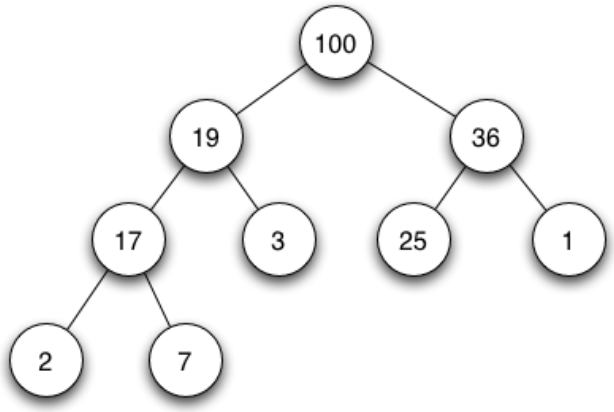
Kupac

A **kupac** (más néven **halom**) egy speciális fa alapú adatszerkezet, amely eleget tesz a *kupac tulajdonságnak*, azaz ha a **B** csúcs fia az **A** csúcsnak, akkor kulcs(A) ≥ kulcs(B) - és ebben az esetben a kupacot *max-kupacnak* (vagy *maximum-kupacnak*) nevezzük. Az összehasonlítás megfordításával *min-kupacot* (azaz *minimum-kupacot*) kapunk, melyben minden **A** csúcsból leszármazó **B** csúcshoz kulcs(B) ≥ kulcs(A). A kupac egy maximálisan hatékony implementációja a prioritási sor adatszerkezetnek.

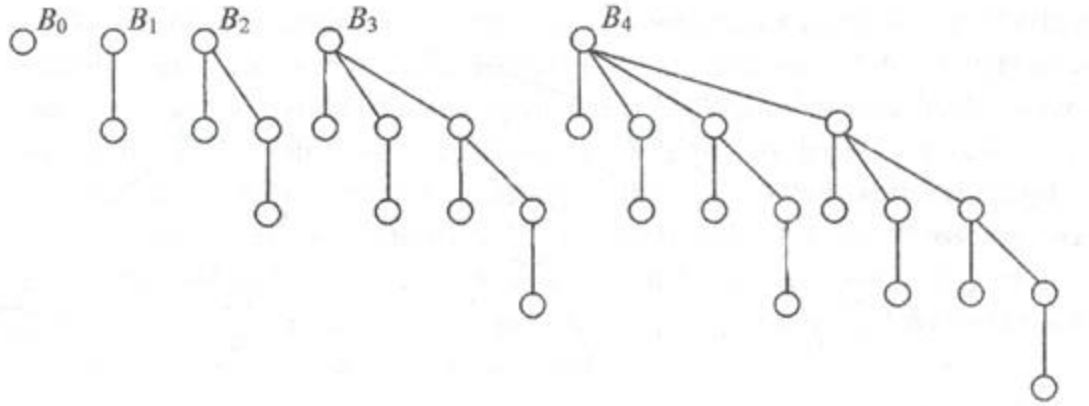
A kupac adatszerkezet különböző fajtáit több algoritmus hatékony implementációja során alkalmazhatjuk:

- Tömbök kupacos rendezése során, mivel a bináris kupacok tömb formájában is felírhatóak.
- Kiválasztó algoritmusokban a *k*-adik legkisebb vagy legnagyobb elem megkeresése lineáris időben elvégezhető kupaccal.

- Súlyozott gráfokat bejáró algoritmusok gyorsíthatóak kupacok alkalmazásával (pl. Dijkstra-algoritmus)



Binomiális fa



A binomiális fákat a következo rekurzív definícióval adhatjuk meg. A ˝ B0 fa egyetlen pontot tartalmaz. A Bk fa (k ≥ 1) pedig két összekapcsolt Bk−1 fából áll, az egyik fa gyökércsúcsa a másik fa gyökércsúcsának a legbaloldalibb gyereke lesz.

Rendezési algoritmusok

Rendezésnek nevezünk egy algoritmust, ha az valamilyen szempont alapján sorba állítja elemek egy listáját

- belso rendezés
- külso rendezés

Belso rendezési algoritmusok

- buborékrendezés (Bubble sort)
- beszúró rendezés (Insertion sort)
- Shell-rendezés (Shell sort)
- összefésülo rendezés (Merge sort)
- kupacrendezés (Heapsort)
- gyorsrendezés (Quicksort)

Hasító (hash) táblák és hasító algoritmusok

Hasító tábla

Adott egy nagyméret ˝u U univerzum amelyhez a kulcsuk által azonosított elemeket tartoznak. Ezek közül szeretnénk elemeket tárolni egy m méret ˝u T tömbben úgy, hogy az elemek a kulcs alapján hatékonyan megtalálhatóak legyenek. A továbbiakban feltesszük, hogy az elemek maguk a kulcsok, de a valódi alkalmazásokban a kulcsok cask azonosításra szolgálnak, és az elemek további adatokat tartalmaznak.

Választunk egy $h : U \rightarrow \{0,...,m-1\}$ hasítófüggvényt, amely minden a halmazelemre megadja azt a tömbindexet, ahol az elemet tárolni szeretnénk T[h(a)] := a.

Abban az esetben van probléma, ha két különbözo elemet ugyanott szeretnénk tárolni azaz, ha ˝ $a \neq b$ és $h(a) = h(b)$. Az ilyen esetekben ütközésrol beszélünk. Az ütközések feloldására két elterjedt módszer a láncolás és a nyílt címzés.