Kupac, prioritásos sor, kupacrendezés

9. GYAKORLAT

Kupac adatszerkezet

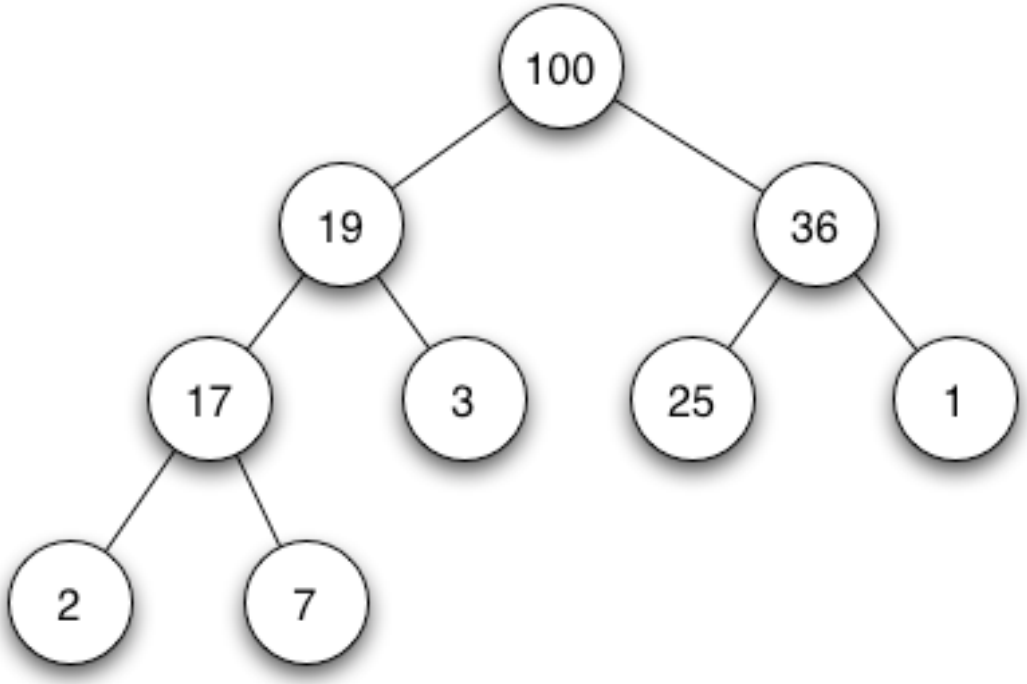
HEAP

Kupac

Mi is az a kupac?

Egy majdnem teljes, balra tömörített, bináris fa

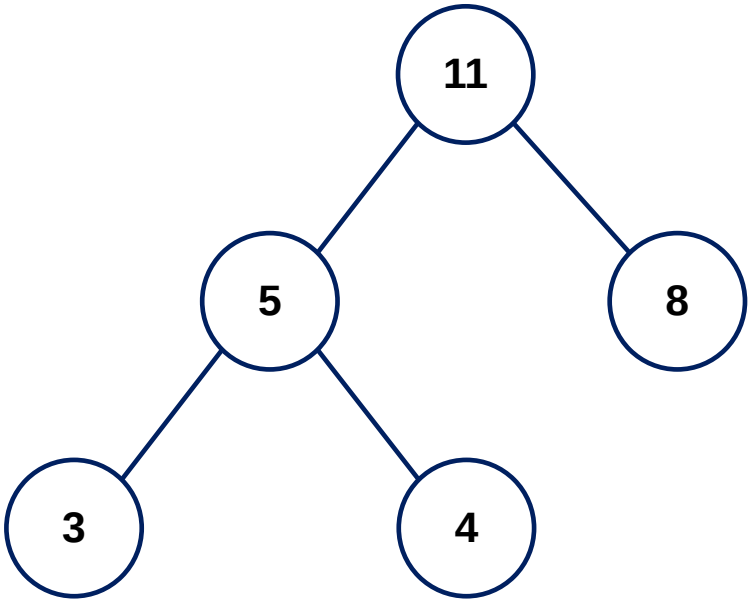
- Majdnem teljes: a fa *h* magasságú, a felső h-1 szintje teljes
- Balra tömörített: az utolsó szint balról kezdve van feltöltve, azaz csak a jobb oldaláról hiányozhatnak esetleg elemek (tehát speciális esetben a fa minden szintje teljes)

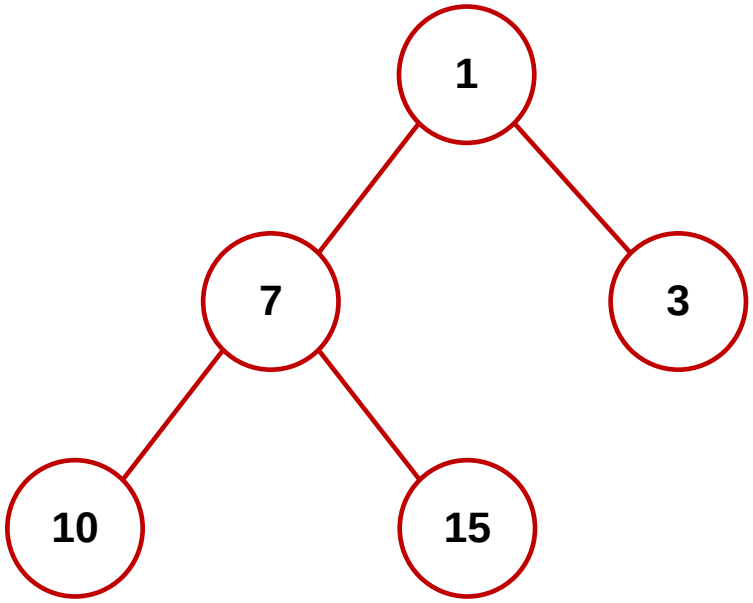


Kupac

Kupactulajdonság:

- minden szülő kulcsa nagyobb (maximum-kupac) / kisebb (minimum-kupac), mint a gyerekei kulcsa – így a legnagyobb / legkisebb kulcsú elem a gyökér





Kupac műveletei

Beszúrás:

- Az újonnan beszúrt elemet a balra tömörítettség szabálya szerint a következő szabad helyre tesszük
- Ezután ezt az elemet a megfelelő helyére juttatjuk a kupacban, így helyreállítjuk a kupactulajdonságot

Törlés:

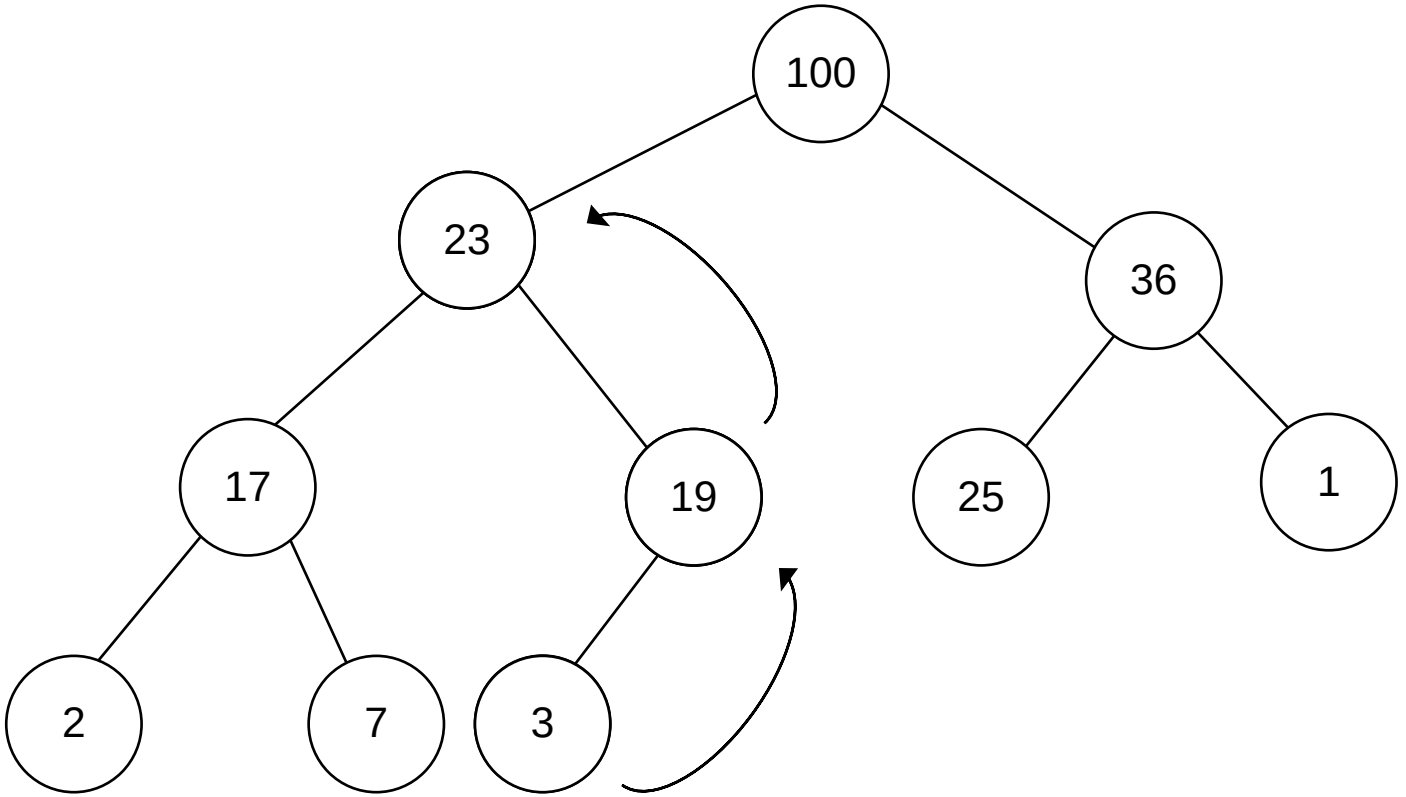
- A gyökérelemet kivesszük a helyéről
- A leendő gyökérelemet a helyére juttatjuk, így szintén helyreállítjuk a kupactulajdonságot

Beszúrás közelebbről

Miután az elemet beszúrtuk az első szabad helyre, biztosítanunk kell, hogy a fa továbbra is megfelel a kupactulajdonságnak, ezért a frissen beszúrt elemet a helyére juttatjuk, „felbuborékoltatjuk” a fában.

Buborékoltatás: az elemet a fában felfelé / lefelé mozgatjuk.

Beszúrás



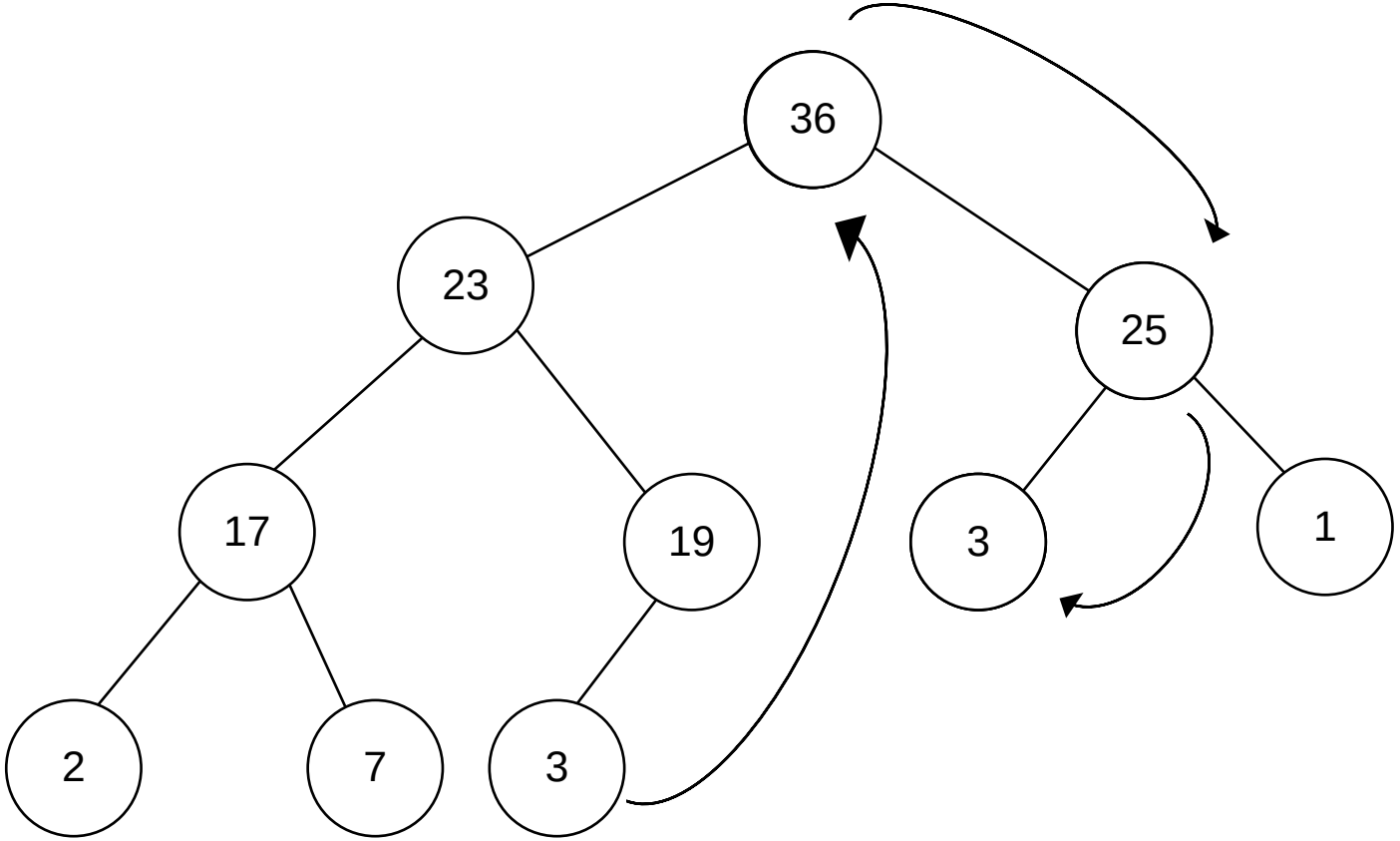
Törlés közelebbről

Elmentjük a gyökérelemet, majd az utolsó elemet kivesszük a helyéről, és a gyökércsúcsba tesszük.

A kupactulajdonság teljesülését most is biztosítanunk kell, ezért a gyökérbe került elemet „lefelé buborékoltatjuk”, mindig a **nagyobb** gyerek felé, amíg a helyére nem kerül.

Az elmentett gyökérelemből csak ezután lesz visszatérési érték.

Törlés



Fák ábrázolása tömbben

Mivel a kupac majdnem teljes, valamint balra tömörített, ezért tömbben is lehet ábrázolni.

A tömbben a kupac szintjei a gyökértől kezdve, egymás után jönnek, az egyes szintekben az elemek balról jobbra következnek: az első elem a gyökér, ezután jön a bal gyerek, majd a jobb gyerek, és így tovább…

Így a reprezentáció nem lesz lyukas.

A kupac ezen reprezentációja egyszerűbb, kisebb, ezáltal gyorsabb a gráfos ábrázolásnál, ezért ezt használják.

Leképezés a tömbbe

A fa elemeit le kell képezni egy tömbbe.

- Egyértelmű leképezésnek kell lennie, azaz oda-vissza működnie kell

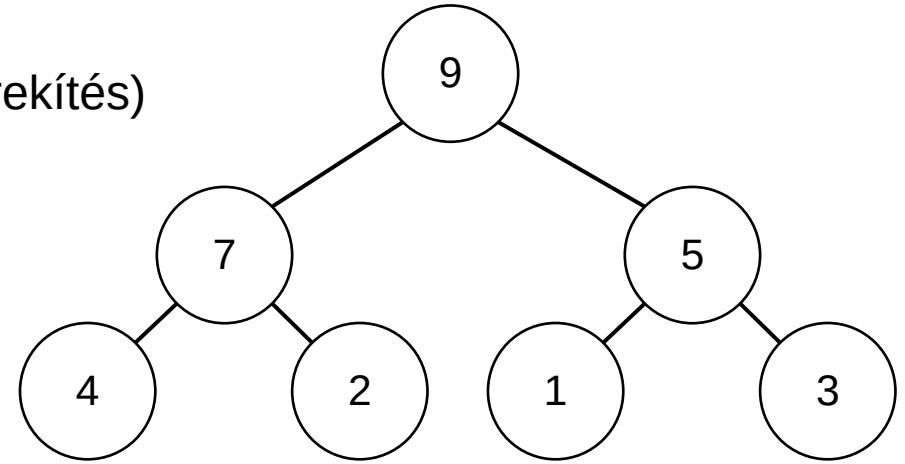
Egy indexfüggvényre van szükségünk.

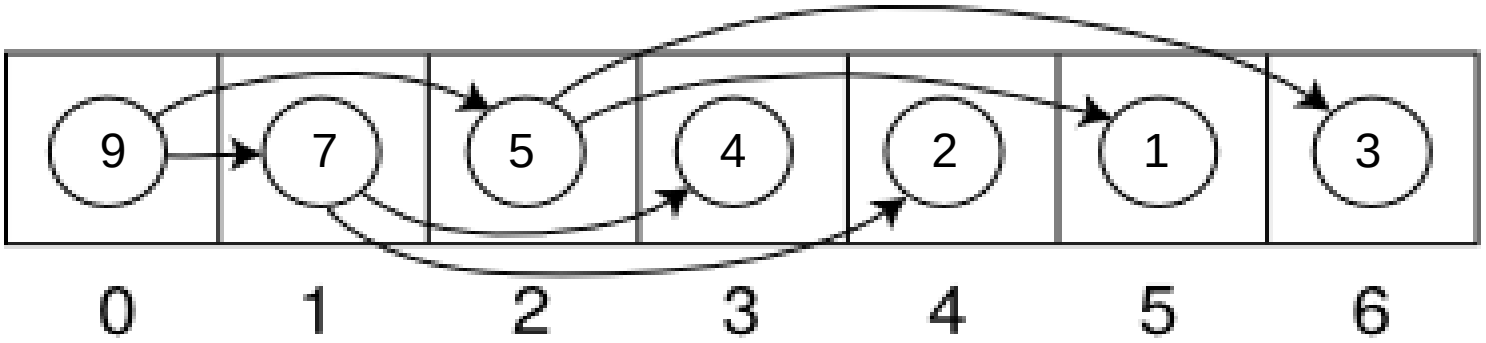
- A mátrixok sor- / oszlopfolytonos ábrázolásánál hasonló függvényeket használunk

Indexfüggvények

Hasonlóan a tömb indexeléshez – 0-tól induló indexelés esetén

- *i* csúcs bal gyereke: 2*i* + 1
- *i* csúcs jobb gyereke: 2*i* + 2
- *i* csúcs szülője: ⌊(*i* – 1) / 2 ⌋ (lefele kerekítés)





Órai kódolás

Programozzátok le a kupacot!

A kupac legfőbb alkalmazásai

PRIORITÁSOS SOR ÉS A KUPACRENDEZÉS

Prioritásos sor

PRIORITY QUEUE

Prioritásos sor

Egy olyan hagyományos sor, melyben minden elem ki van egészítve egy prioritást jelző adattaggal.

A sor „out” művelete mindig a legnagyobb prioritású elemet adja vissza.

Megvalósítható még kétirányú láncolt listával is.

- Ilyenkor a beszúrásnál a prioritás szerinti helyére szúrjuk be az elemet
- Ez lassabb az átlagos esetben

Művelet	Kupac	Láncolt lista
Beszúrás	O(log(*n*))	O(*n*)
Törlés	O(log(*n*))	O(1)

Prioritásos sor megvalósítása kupaccal

A prioritásos sor belsejében egy kupac található, melyben egy konténer osztály elemeit tároljuk.

- Ez a konténer osztály a prioritásos sor belső osztálya
- Az osztály rendelkezik egy prioritás adattaggal, valamint a sorba betett értéket is benne tároljuk
- A konténer osztály összehasonlító operátorai úgy vannak megírva, hogy a kupacon belül az elemek a prioritás szerint vannak összehasonlítva
- Ekkor a legmagasabb prioritású elem a sorban éppen a kupac maximális eleme lesz – csak ezt tudjuk elérni / eltávolítani.

Órai kódolás

Használjuk fel a kupacot a prioritásos sor megvalósításához!

Kupacrendezés

HEAPSORT

Kupacrendezés

Egyszerűen betesszük az elemeket egy kupacba, majd kivesszük belőle őket.

- Az elemek rendezett sorban fognak kijönni

Komplexitása mindig ~O(n log(n)).

Ez lényegesen jobb, mint a lassú rendezések komplexitása.

- Nincs legrosszabb eset, nincs legjobb eset
- Ez előnye és hátránya egyben

Nem stabil rendező, azaz az egyenlő kulcsú elemek sorrendje megváltozhat a rendezés közben.

A gyorsrendezés általában némileg gyorsabb nála, azonban annak időigénye legrosszabb esetben $O(n^2)$, így egyes esetekben időigényének kiszámíthatósága miatt a kupacrendezőt célszerű használni.

Órai kódolás

Használjuk fel a kupacot rendezésre!

Hasonlítsuk össze a kupacrendezés futásidejét a gyorsrendezőével!

- Vizsgáljuk meg, hogy a gyakorlatban mi történik a legrosszabb esetben

Rendezés beépített függvénnyel

template <class RandomAccessIterator>
void **sort** (RandomAccessIterator first, RandomAccessIterator last);

template <class RandomAccessIterator>
void **stable_sort** (RandomAccessIterator first, RandomAccessIterator last);

Mindkét függvény növekvő sorrendbe rendezi az iterátorok által megadott tartományon [first, last) szereplő elemeket.

Használatukhoz szükség van a **#include <algorithm>** könyvtárra, valamint a szokásos **std** névtérben találhatóak.

A **sort** nem stabil rendező, nem őrzi meg az azonos elemek eredeti sorrendjét.

A **stable_sort** megőrzi az azonos elemek eredeti sorrendjét.

További információk: *http://www.cplusplus.com/reference/algorithm/sort/*

Órai kódolás

A vektorunk kupaccal történő rendezése mellett készítsük el a main-ben a std::sort függvénnyel történő rendezését is.

(Ehhez készítsünk másolatot ez eredeti vektorról!)

Gyakorló Feladat

Írjunk egy interaktív konzolos programot, amellyel fontossági sorrendbe tudjuk állítani teendőinket.

A programnak a következő menüpontjai legyenek:

- Feladat hozzáadása prioritással
  - A prioritást egy 1-től 100-ig növekvő skálán lehessen megadni
- Legfontosabb feladat elvégzése
  - Ekkor írassuk ki az elvégzendő feladatot
- Feladatok hozzáadása fájlból
  - Itt egy fájl nevét kell megadni, melyben sorokban tároljuk a feladatok nevét és prioritását
  - A fájl sorainak formátuma legyen: **string;int**
  - Ezeket a már meglévő feladatokhoz kell hozzáadni
- (Kilépés a programból)

Házi feladat

1) Módosítsd a megírt kupacot úgy, hogy egy a konstruktorban átadott bool paraméter segítségével lehessen meghatározni, hogy maximum-kupaccal vagy minimum-kupaccal szeretnénk dolgozni (a kupac tetején a legkisebb elem foglal helyet, a csomópontok gyerekei nagyobb értékűek a szülőjüknél.)

Törekedj effektív kódolásra, lehetőség szerint kerüld a kódismétlést!

2) Alakítsd át a bináris kupacot, ternáris kupaccá, azaz minden csúcsnak maximum 3 gyereke lehet az eddigi kettő helyett! (pl: bal, középső, jobb gyerek) Őrizd meg a balra tömörített, majdnem teljes tulajdonságot!