

RI301

Strukture podataka

dr.sc. Edin Pjanić

Pregled predavanja

- Prioritetni redovi
 - elementarne implementacije
- Gomila – heap
 - definicija i osnovne operacije
 - implementacija operacija

Prioritetni redovi

- Prioritetni red (PR) je struktura podataka koja je generalizacija stoga i reda.
 - stog (stack) – zadnji ubačeni je prvi izbačeni
 - red (queue) – prvi ubačeni je prvi izbačeni
- Izlaz iz prioritetnog reda se određuje na osnovu jednog atributa (osobine) podatka koji se ubacuje u red – a kojeg nazivamo **prioritet**
- Primjeri:
 - planiranje izvršavanja procesa ili poslova u računarskim sistemima (job scheduling)
 - simulatori (ključevi mogu biti vremena događaja koji se moraju obraditi po redoslijedu)

Prioritetni redovi - primjer

- U PR proizvoljnim redoslijedom ubacujemo zadatke (obaveze) sa prioritetima. Kad izbacujemo zadatke iz liste, tj. uzimamo na obradu, prvo izlaze zadaci sa najvećim prioritetom.
 - redoslijed ubacivanja nije bitan

ULAZ

↓
1 – kupiti novine
5 – platiti ratu kredita
4 – nasuti gorivo – u crvenom je
3 – nazvati kolegu u vezi sa ispitom
4 – platiti račun – došla opomena
2 – otići na kafu
5 – predavanja iz SP
5 – kupiti sijalicu

IZLAZ

↓
5 – predavanja iz SP
5 – kupiti sijalicu
5 – platiti ratu kredita
4 – nasuti gorivo – u crvenom je
4 – platiti račun – došla opomena
3 – nazvati kolegu u vezi sa ispitom
2 – otići na kafu
1 – kupiti novine

Osnovne operacije PR ATP

- Osnovne operacije koje treba da podrži prioritetni red (PR) kao apstraktni tip podataka (ATP) su:
 - kreiranje (konstrukcija) PR
 - ubacivanje novog elementa
 - uklanjanje elementa sa najvećim prioritetom
 - mijenjanje prioriteta nekog elementa
 - spajanje dva PR u jedan veći
 - itd.
- Određena implementacija ne mora podržati sve nabrojane ali prve 3 su obavezne.

Elementarne implementacije PR

- Jedan od načina implementacije PR je korištenjem neuređene (nesortirane) liste, npr. implementirane pomoću niza
- Ono što treba obezbijediti je da se pri operaciji uklanjanja iz reda uklanja element sa najvećim prioritetom.
- Ostale operacije su iste kao kod obične liste.

```
template <typename Elem>
class PR
{
    private:
        // zavisi od implementacije
    public:
        PR(int);
        void ubaci(Elem const &);
        Elem izbaci();
        ...
};
```

Implementacije PR – nesortirana lista

```
template <typename Elem>
```

```
class PR
```

```
{
```

```
    private:
```

```
        Elem *pr;
```

```
        int N;
```

```
    public:
```

```
        PR(int maxN)
```

```
        { pr = new Elem[maxN]; N = 0; }
```

```
        void ubaci(Elem const v)
```

```
        { pr[N++] = v; }
```

pr



maxN

$O(1)$

$O(n)$

$O(1)$

Implementacije PR – sortirana lista

- Ako bismo PR implementirali sortiranom listom tako da elemente sortiramo po prioritetu onda bismo za operaciju izbacivanja imali $O(1)$ jer bismo tačno znali gdje je najveći element (na kraju niza)
- Problem je što bismo pri svakom ubacivanju novog elementa morali vršiti ubacivanje na odgovarajuću poziciju, što je $O(n)$
- Slično je i kod implementacije povezanim čvorovima (linkana lista). Uvijek imamo $O(n)$ za neku operaciju.
- Možemo li sve ove operacije raditi brže od linearnog vremena, tj. brže od $O(n)$?
 - možemo – struktura podataka **heap** (gomila)

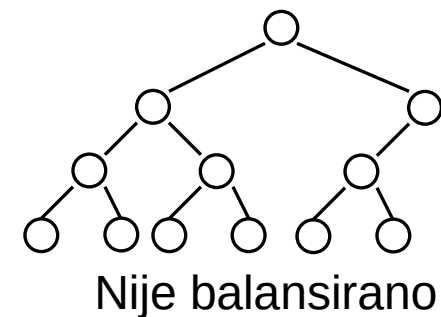
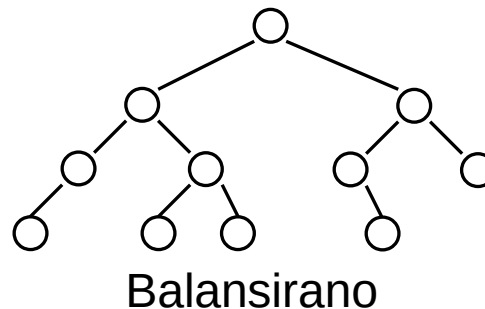
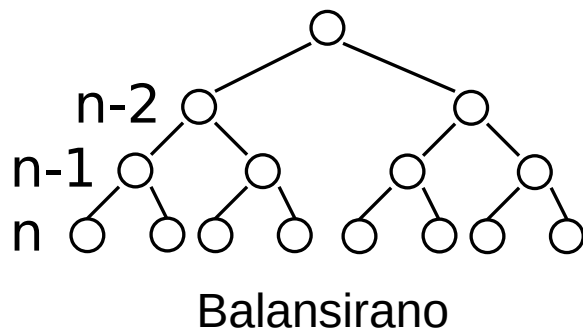
Heap – gomila

Definicija za heap (gomilu) kao strukturu podataka:

- **Kao struktura podataka, to je balansirano, potpuno binarno stablo u kojem svaki čvor ima vrijednost (ključa) koja nije veća od njegovog roditelja.**
- Primjena gomile - heap:
 - prioritetni red
 - zanimljiv pristup za sortiranje (heapsort)
 - složenost je $O(n \log n)$ – uvijek

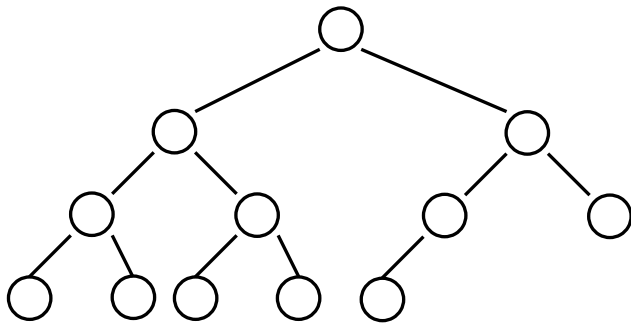
Definicija – balansirano binarno stablo

- Binarno stablo je **balansirano** ako svi čvorovi na dubinama od 0 do $n-2$ (do predzadnjeg) imaju dvoje djece.

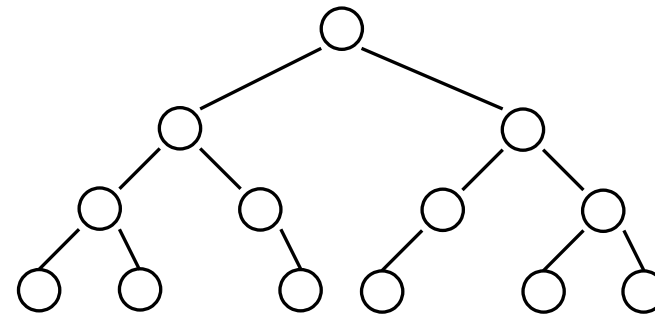


Definicija – potpuno binarno stablo

- Binarno stablo je **potpuno** ako je balansirano i svi čvorovi na dubini n su smješteni krajnje lijevo.



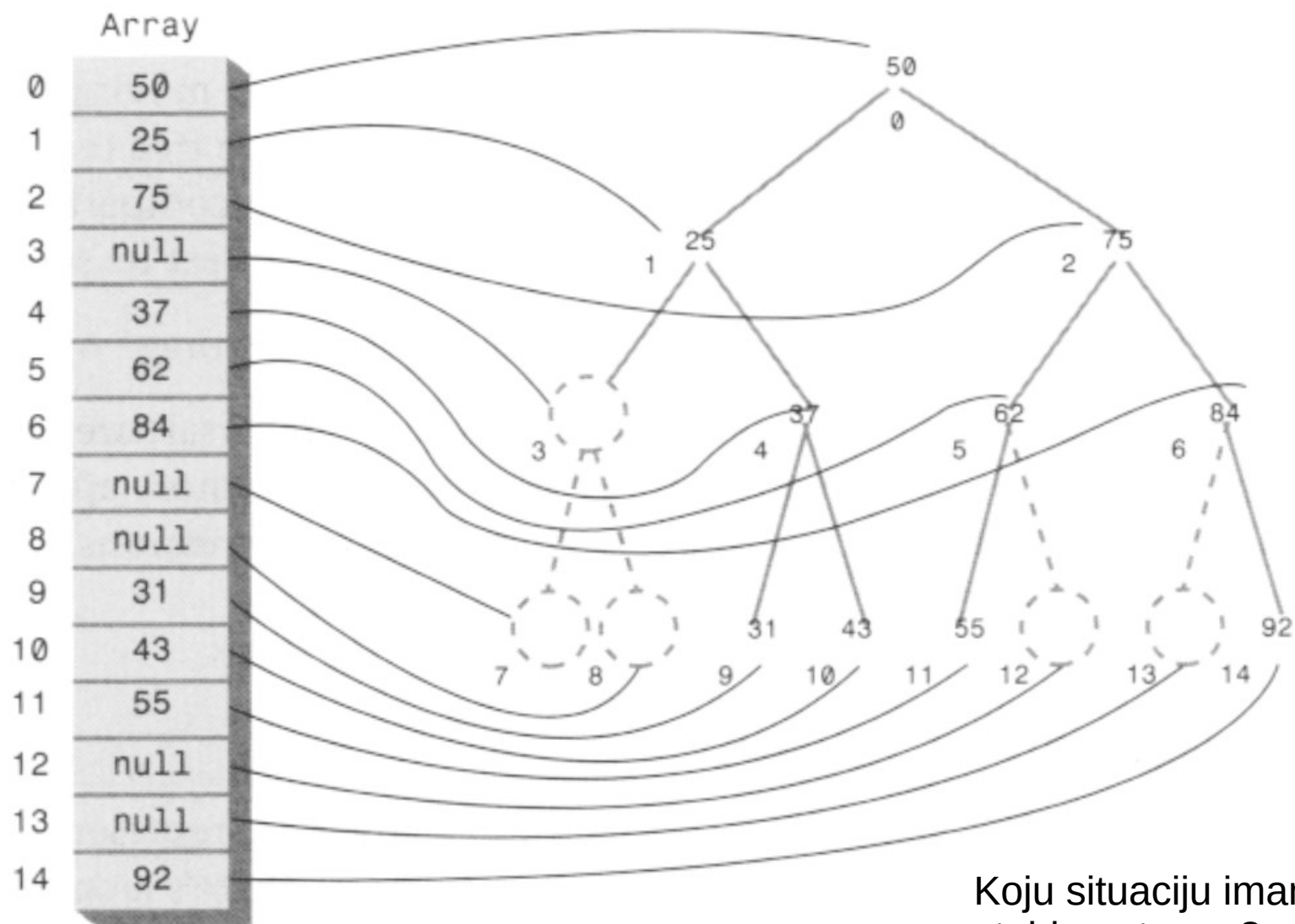
Potpuno



Nije potpuno

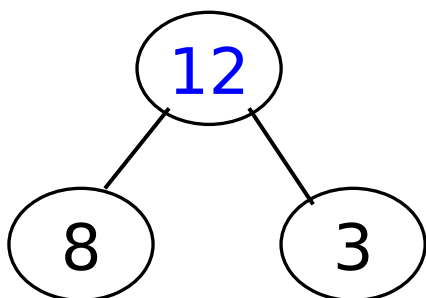
Prisjetimo se

- Binarno stablo je moguće implementirati pomoću niza:

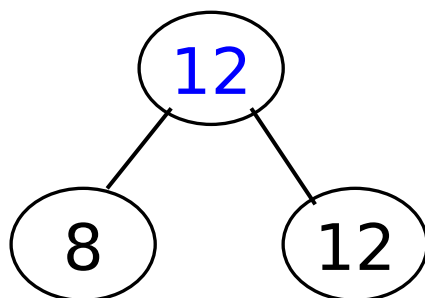


Osobina gomile

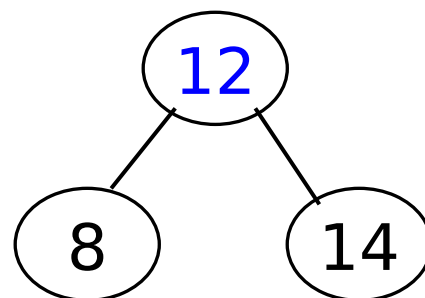
- Čvor ima osobinu gomile ako mu vrijednost ključa nije manja od vrijednosti ključa njegove djece.



Plavi čvor
ima osobinu
gomile



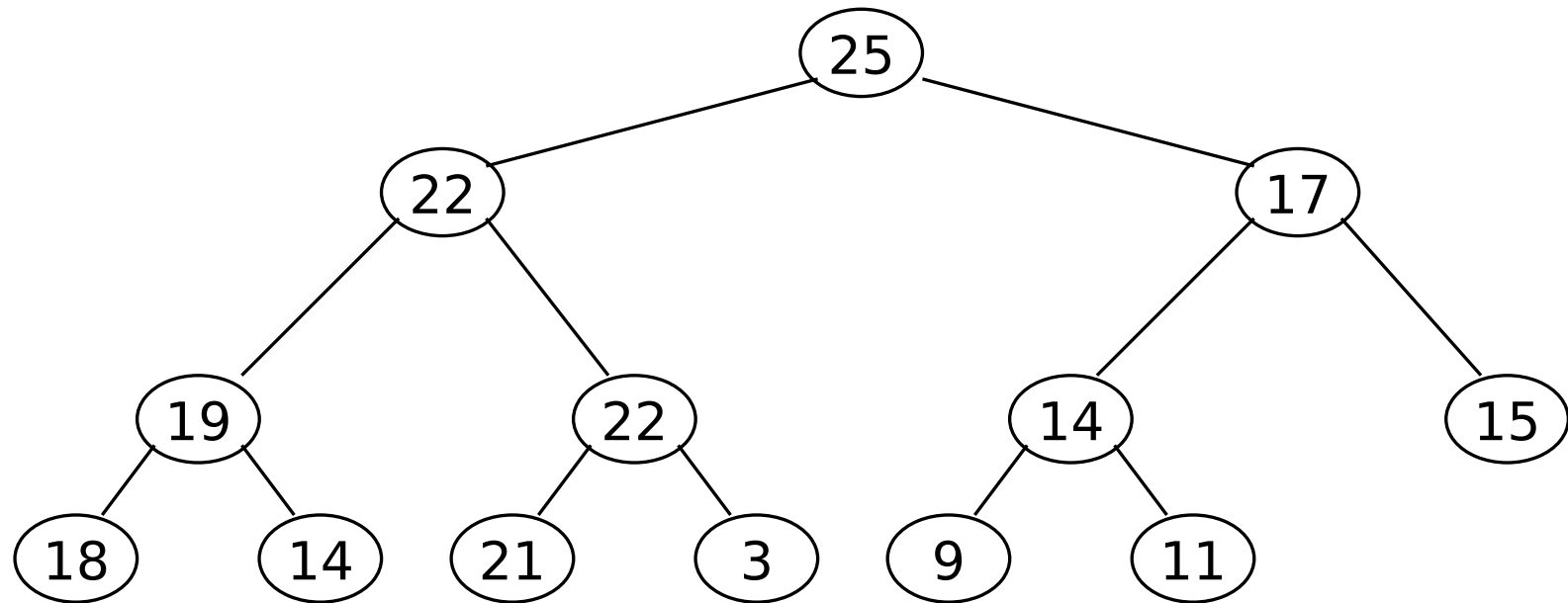
Plavi čvor
ima osobinu
gomile



Plavi čvor nema
osobinu gomile

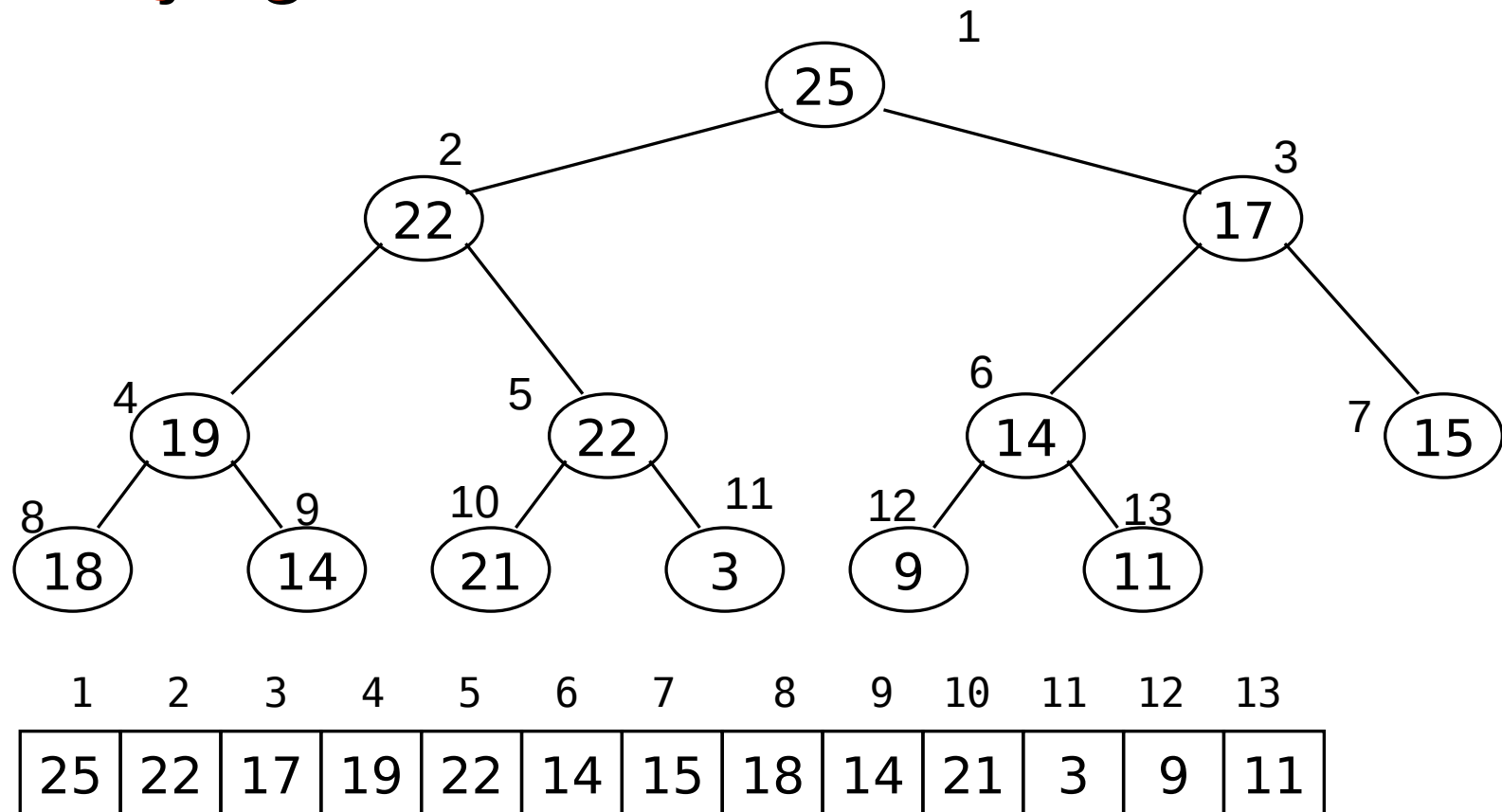
- Potpuno binarno stablo je gomila (heap) ako svi čvorovi imaju osobinu gomile

Neke karakteristike gomile



- Najviše $\lg(N)$ generacija (nivoa) u stablu
- Sve putanje: najviše $\lg(N)$ čvorova
- Svaka generacija ima 1/2 čvorova naredne generacije (osim predzadnje)
- => sve operacije nad gomilom imaju max. logaritamsko vrijeme.

Mapiranje gomile u niz



- Primijetimo:

- Lijevo dijete čvora na indeksu i je na poziciji $2*i$
- Desno dijete čvora na indeksu i je na poziciji $2*i+1$
- Primjer: djeca čvora 4 (19) su čvorovi 8 (18) i 9 (14)

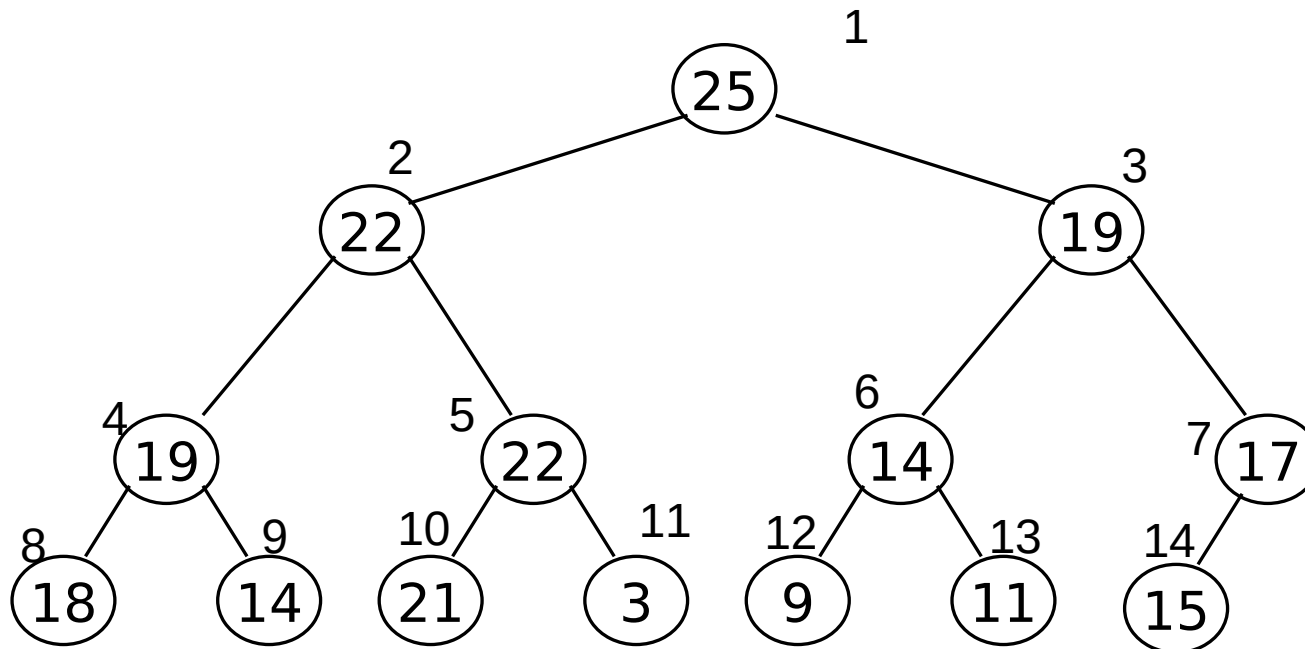
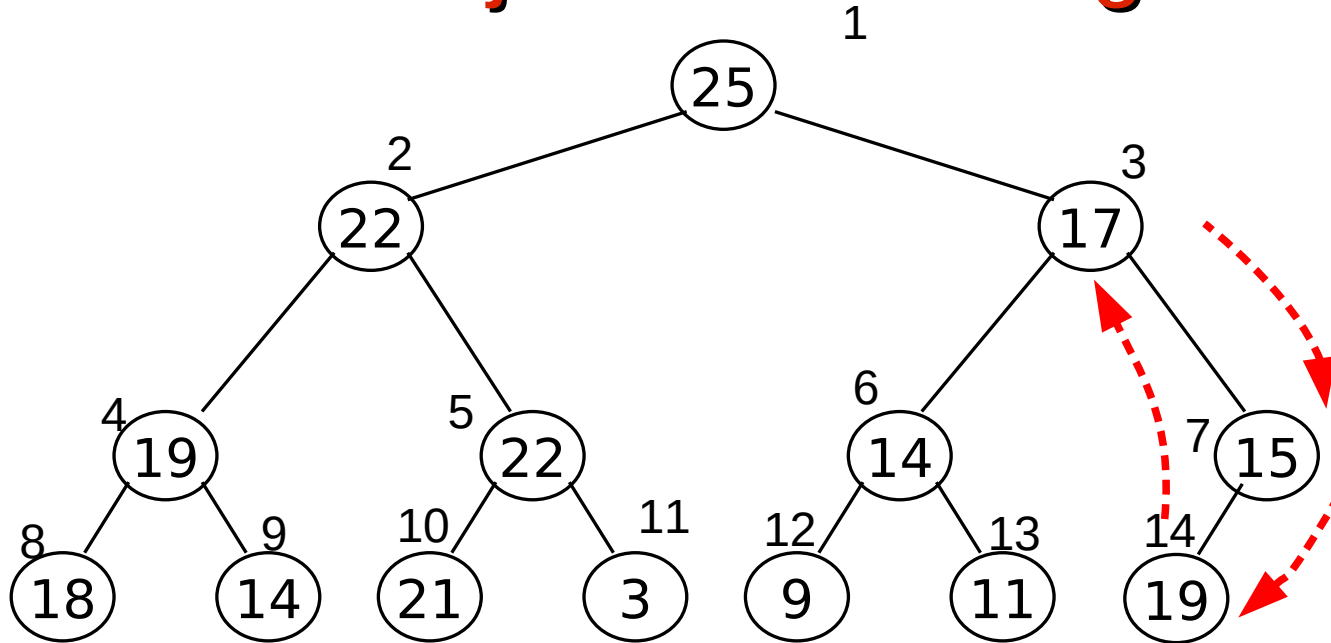
Operacije i algoritmi nad gomilom

- Svi algoritmi sa gomilom (dodavanje, uklanjanje, i sl.) rade tako da se najprije napravi jednostavna strukturalna izmjena, koja će narušiti osobinu gomile, a onda se prođe kroz gomilu da bi se uspostavio uslov gomile koji je narušen.
- Dva pristupa implementaciji algoritama:
 - Top-down (s vrha nadole)
 - Bottom-up (od dna nagore)
- U našem primjeru, gomilu ćemo implementirati pomoću niza (bazirano na klasi PR, sa slajda 7)

Ubacivanje elementa u gomilu

- Novi element ubacujemo uvijek na kraj niza jer je to $O(1)$
 - moramo povećati veličinu niza za 1 ($N=N+1$)
- Ako je narušena osobina gomile za roditeljski čvor novog čvora (ubačeni element je veći od svog roditelja), uspostavićemo tu osobinu tako što zamijenimo taj čvor sa njegovim roditeljem.
- Ovo može ponovo narušiti osobinu gomile za roditeljski čvor ovog čvora pa na isti način i to korigujemo.
- Ako ima potrebe, nastavimo isti postupak do vrha stabla.

Ubacivanje elementa u gomilu (sa slajda 15)

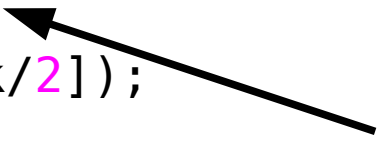


Implementacija ubacivanja novog elementa

- Novi element ubacujemo uvijek na kraj niza nakon povećanja veličine gomile.
- Nakon toga uspostavimo narušenu osobinu gomile.

```
template <typename Elem>
void PR<Elem>::insert(Elem const & v)
{
    pr[++N] = v;
    sredi_nagore(N);
}

template <typename Elem>
void PR<Elem>::sredi_nagore(int k)
{
    while (k > 1 && pr[k/2] < pr[k])
    {
        std::swap(pr[k], pr[k/2]);
        k = k/2;
    }
}
```



roditelj

Izbacivanje elementa iz gomile

- Iz gomile (prioritetni red) uvijek izbacujemo element sa vrha, tj. korijena (root).
- Ova operacija bi stablo podijelila na dva dijela.
- Da ne bismo razdvojili stablo te da bismo ponovo uspostavili balansirano, potpuno binarno stablo, najjednostavnije je:
 - uzeti zadnji element iz gomile i postaviti ga na vrh stabla, umjesto izbačenog elementa
 - ovo uvijek narušava osobinu gomile
 - krenuti s vrha i uspostaviti narušenu osobinu tako što se roditelj koji je manji od bar jednog djeteta zamijeni sa najvećim djetetom
 - pri tome moramo **najveće** dijete postaviti na mjesto roditelja da bi novi roditelj bio najveći
 - ponavljamo dok se ne stigne na dno gomile
 - ne zaboraviti smanjiti veličinu gomile.

Implementacija izbacivanja najvećeg elementa

```
template <typename Elem>
Elem PR<Elem>::izbaci()
{
    swap(pr[1], pr[N]);
    sredi_nadole(1, N-1);
    return pr[N--];
}
```

```
template <typename Elem>
void PR<Elem>::sredi_nadole(int k, int n)
{
    while (2*k <= n)
    {
        int j = 2*k;
        if (j < n && pr[j] < pr[j+1]) ++j;
        if (!(pr[k] < pr[j])) break;
        swap(pr[k], pr[j]);
        k = j;
    }
}
```

j treba biti indeks najvećeg (po prioritetu) djeteta: za početak - lijevog djeteta (indeks $2*k$).
Ako **j** nije indeks zadnjeg djeteta provjeri i naredno (desno) dijete pa ako je naredno veće onda postavi **j** na indeks tog djeteta.

Ako najveće dijete nije veće od roditelja – prekini, jer uslov gomile nije narušen. U protivnom, zamijeni roditelja sa najvećim djetetom i nastavi dalje.

PR kontejner – STL implementacija

- Header `<queue>`, klasa `std::priority_queue`
- Neki metodi:
 - ubacivanje elementa `x`: `push(x)`
 - konst. referenca do najvećeg elementa: `top()`
 - uklanjanje najvećeg elementa: `pop()`
 - `size()` – broj elemenata u kontejneru
 - `empty()` – vraća `true` ako je prazan, `false` ako nije