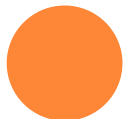
Prioritetna vrsta (angl. *priority queue*)



ADT Prioritetna vrsta

- v prioritetni ali prednostni vrsti ima vsak element oznako prioritete, ki določa vrstni red brisanja elementov iz vrste
- ne velja FIFO (first-in-first-out), kot za navadne vrste
- uporaba: npr. dodeljevanje virov računalniškega sistema
- dogovor: nižja je prioriteta, prej bo element prišel iz vrste

ADT Prioritetna vrsta

Osnovne operacije za ADT PRIORITY QUEUE:

- MAKENULL(Q) : napravi prazno prioritetno vrsto Q
- INSERT(x, Q) : vstavi element x v prioritetno vrsto Q
- DELETEMIN(Q) : vrne element z najmanjšo prioriteto iz prioritetne vrste Q in ga zbriše iz Q
- EMPTY(Q) : ali je prioritetna vrsta Q prazna

```
public interface PriorityQueue {
    public void makenull() ;
    public void insert(Comparable x) ;
    public Comparable deleteMin() ;
    public boolean empty() ;
} // PriorityQueue
```

Implementacije prioritetne vrsta

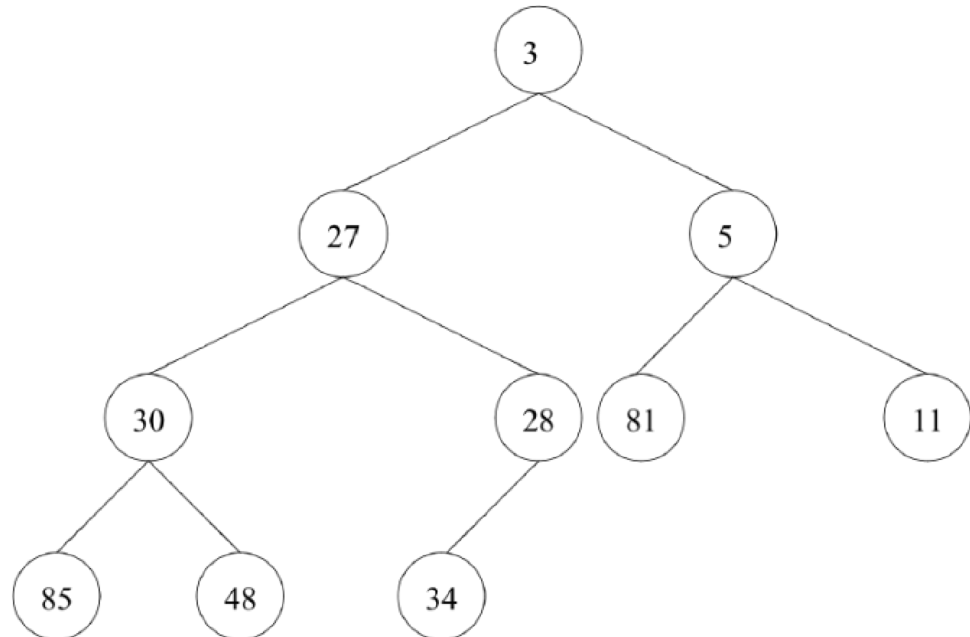
Učinkovitost posameznih implementacij prioritetne vrste:

	MAKENULL	EMPTY	INSERT	DELETEMIN
neurejeni seznam	$O(1)$	$O(1)$	$O(1)$	$O(n)$
urejeni seznam	$O(1)$	$O(1)$	$\leq O(n)$	$O(1)$
BST	$O(1)$	$O(1)$	$\leq O(n)$	$\leq O(\log n)$
AVL, RB- drevo	$O(1)$	$O(1)$	$= O(\log n)$	$= O(\log n)$
kopica	$O(1)$	$O(1)$	$\leq O(\log n)$	$\leq O(\log n)$

KOPICA (HEAP)

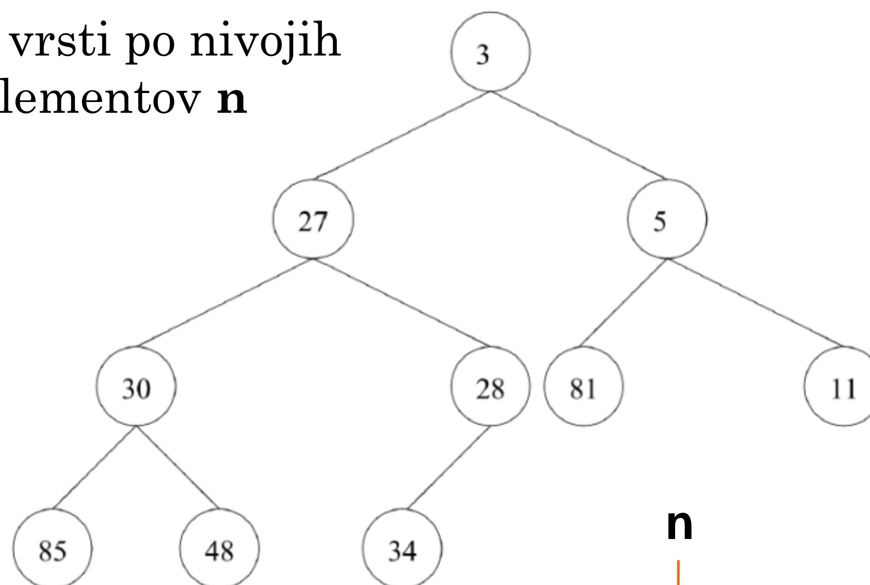
Kopica je binarno drevo z lastnostmi:

1. je levo poravnano
   - na najglobljem nivoju drevesa eventuelno manjkajo elementi samo z desne strani
2. je delno urejeno
   - za vsako poddrevo velja, da je v korenu najmanjši element tega poddrevesa



Implementacija kopice s poljem

- vozlišča hranimo po vrsti po nivojih
- hranimo še število elementov **n**



1	2	3	4	5	6	7	8	9	10	11	12	...
3	27	5	30	28	81	11	85	48	34			

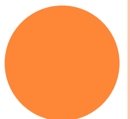
- v vozliščih ne potrebujemo dodatnih indeksov, saj jih lahko sproti izračunamo:

  če z *i* označimo indeks vozlišča, potem velja:

  - $2 * i$ je indeks levega sina
  - $2 * i + 1$ je indeks desnega sina
  - $i \ / \ 2$ je indeks očeta

Implementacija kopice s poljem

```
public class Heap implements PriorityQueue {
    static final int DEFAULT_SIZE = 100 ;
    static final int DEGREE = 2 ;
    Comparable nodes[] ;
    int noNodes, size ;
} // class Heap
```



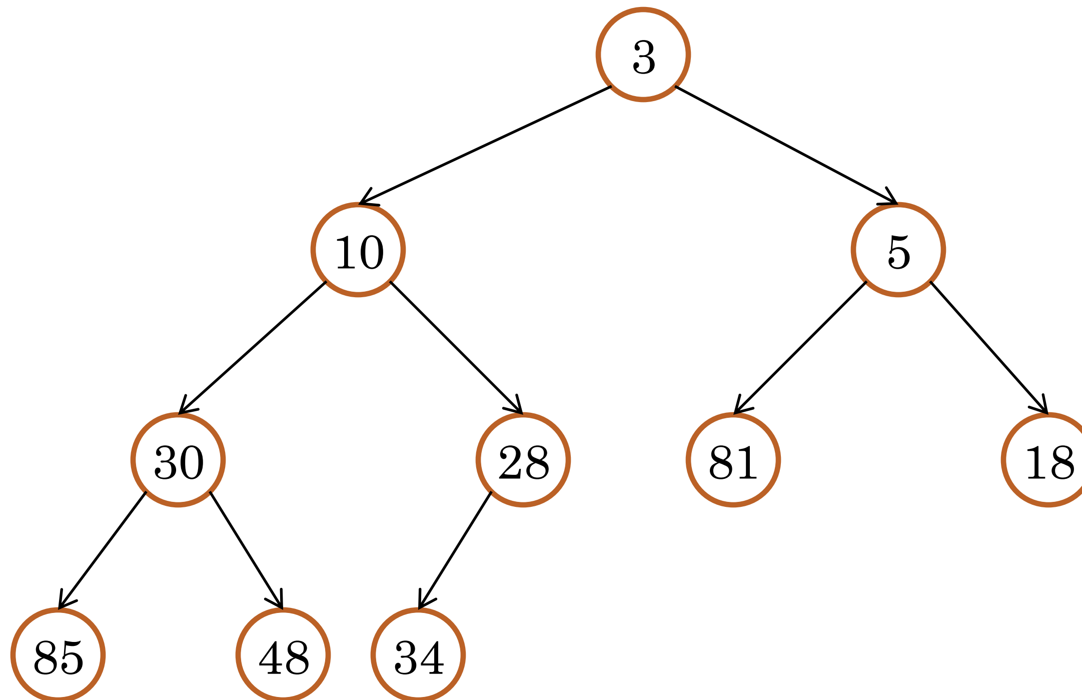
Realizacija operacij na kopici

INSERT:

- element x najprej dodamo na prvo prazno mesto z leve na zadnjem nivoju drevesa
- če je zadnji nivo zapolnjen, ga dodamo kot prvega z leve na naslednjem nivoju
- zamenjujemo x z očetom, dokler ni:
  - oče manjši od x ali
  - x v korenu drevesa
- časovna zahtevnost reda $O(\log n)$

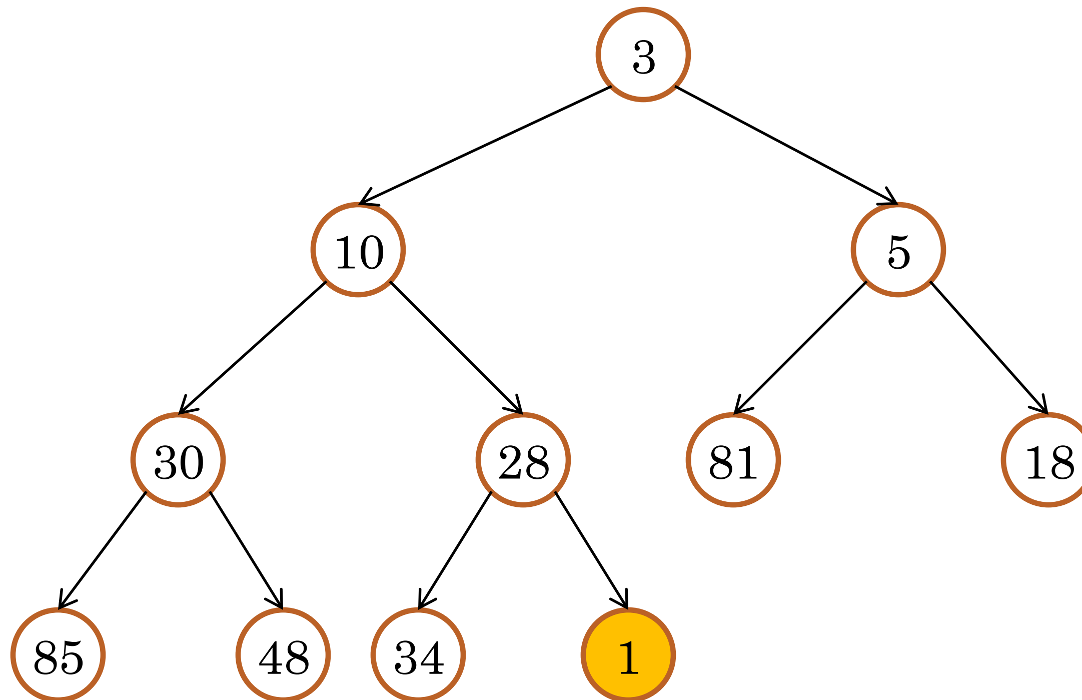
Primer – dodajanje elementa v kopico (1/5)

V kopico na sliki dodaj element 1.



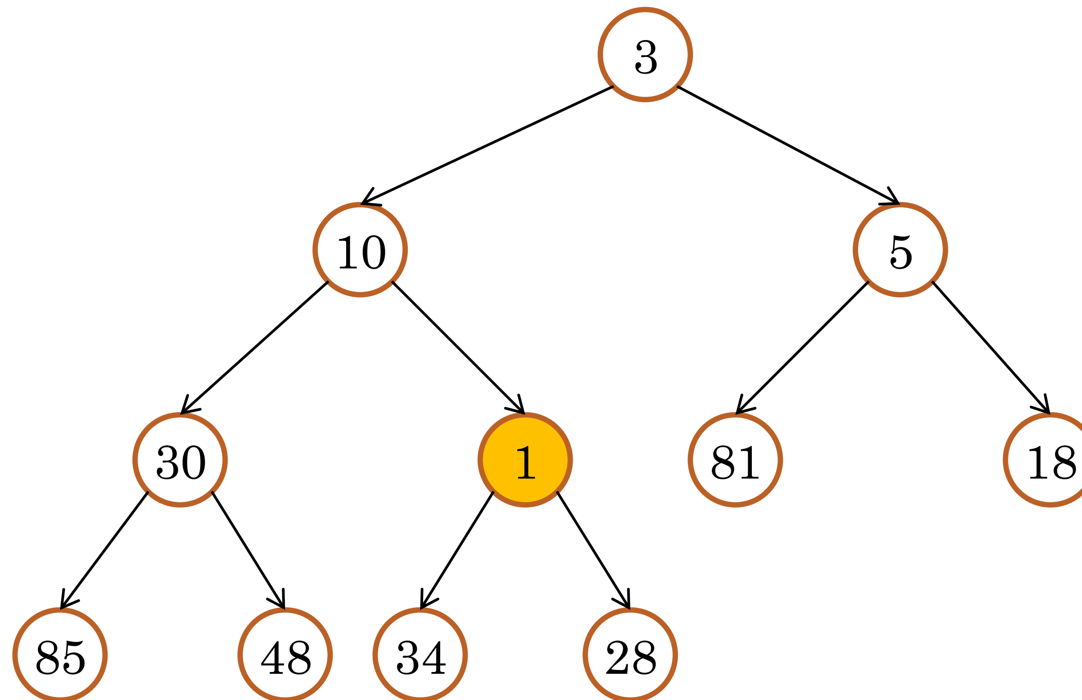
Primer – dodajanje elementa v kopico (2/5)

Element dodamo na prvo prazno mesto z leve na zadnjem nivoju…



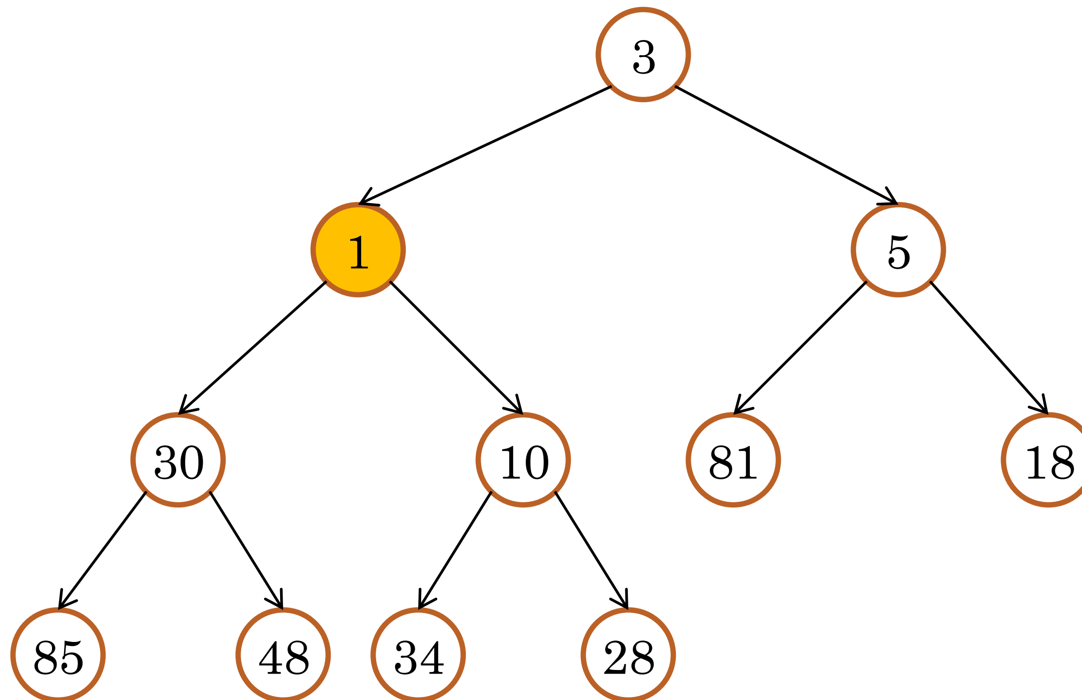
Primer – dodajanje elementa v kopico (3/5)

Zamenjujemo z očetom…



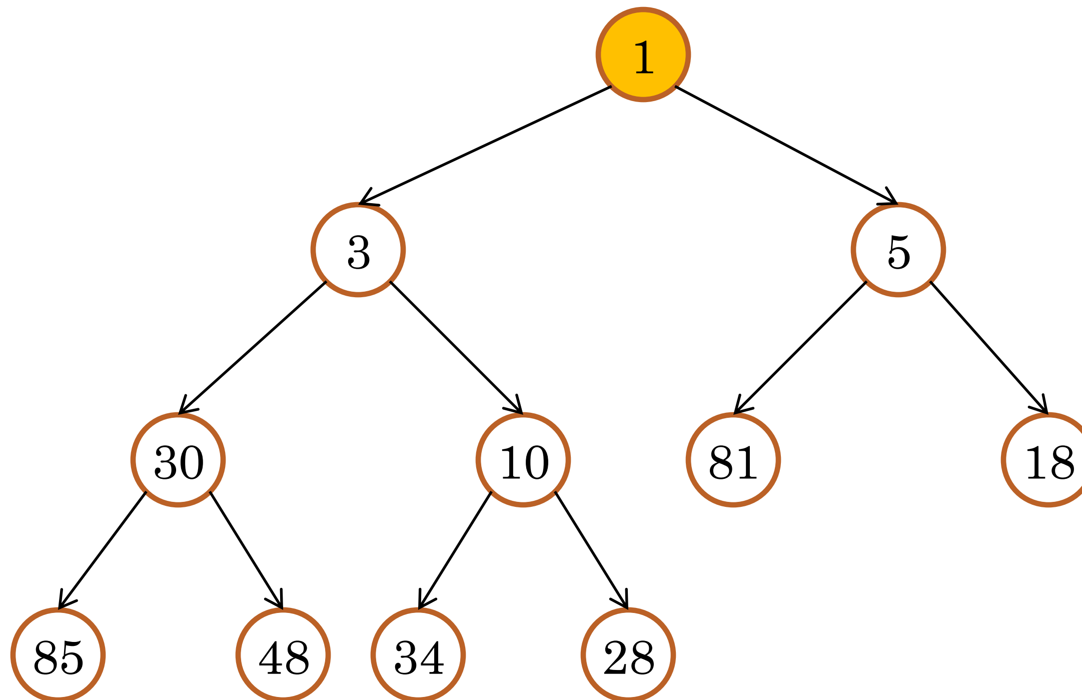
Primer – dodajanje elementa v kopico (4/5)

Zamenjujemo z očetom...



PRIMER – DODAJANJE ELEMENTA V KOPICO (5/5)

…dokler ni oče manjši ali ne pridemo do korena



Implementacija operacije insert

```
public void insert(Comparable x) {
   int newNode, parent; // indeks novega vozlisca in oceta

   noNodes = noNodes + 1;
   newNode = noNodes; // dodamo element na prvo prazno mesto
   parent = newNode / 2; // i−ti element je oce j−tega elementa
   while (parent > 0 && nodes[parent].compareTo(x) > 0) {
      nodes[newNode] = nodes[parent];
      newNode = parent;
      parent = parent / 2;
   }
   // element prepisemo sele, ko poznamo koncen polozaj
   nodes[newNode] = x;
} // insert
```

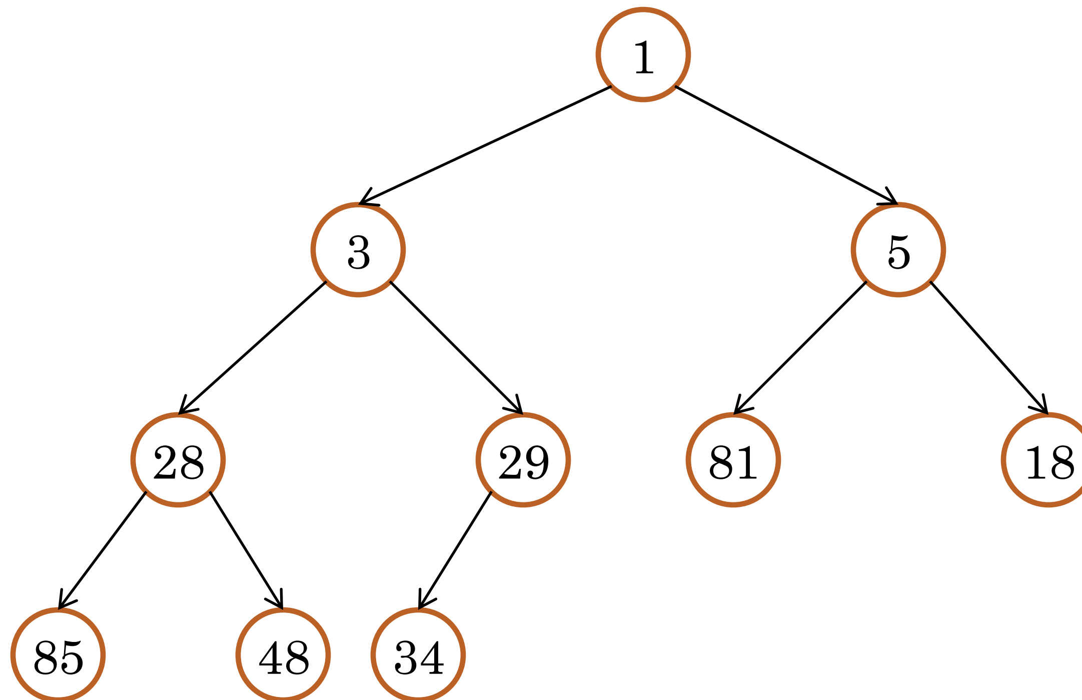
Realizacija operacij na kopici

DELETEMIN:

- najmanjši element se nahaja v korenu
- nadomestimo ga z najbolj desnim elementom x na zadnjem nivoju kopice
- zaporedno zamenjujemo x z manjšim od obeh sinov, dokler ni:
    - x manjši od obeh sinov
    - x list drevesa
- časovna zahtevnost reda $O(\log n)$

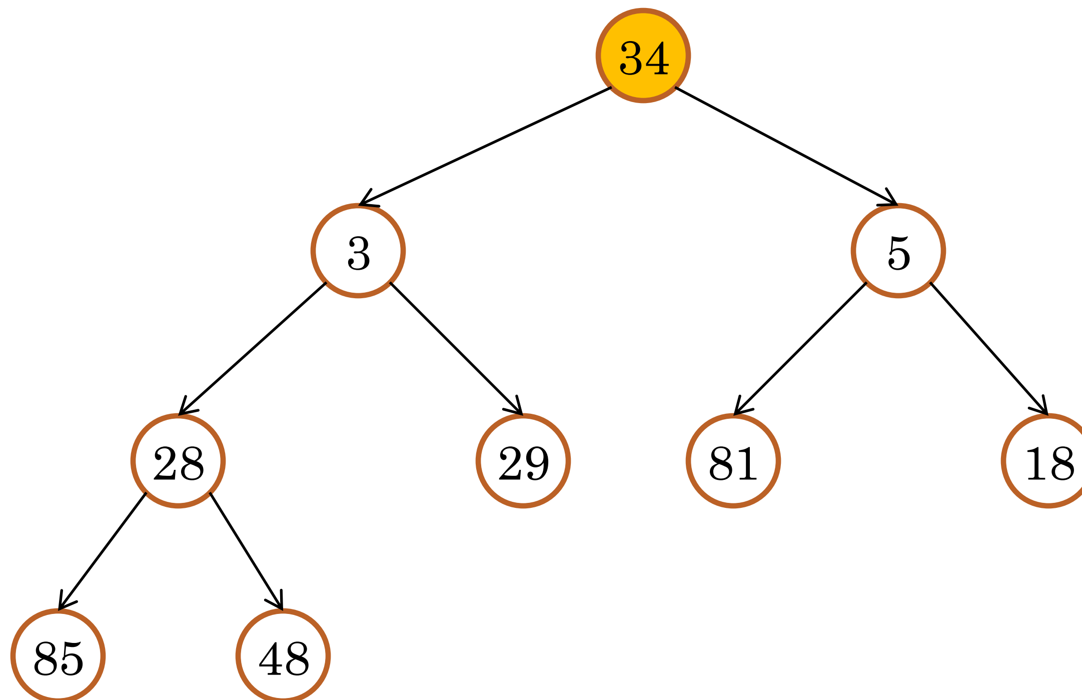
Primer – Brisanje elementa iz kopice (1/4)

Iz kopice na sliki izbriši najmanjši element



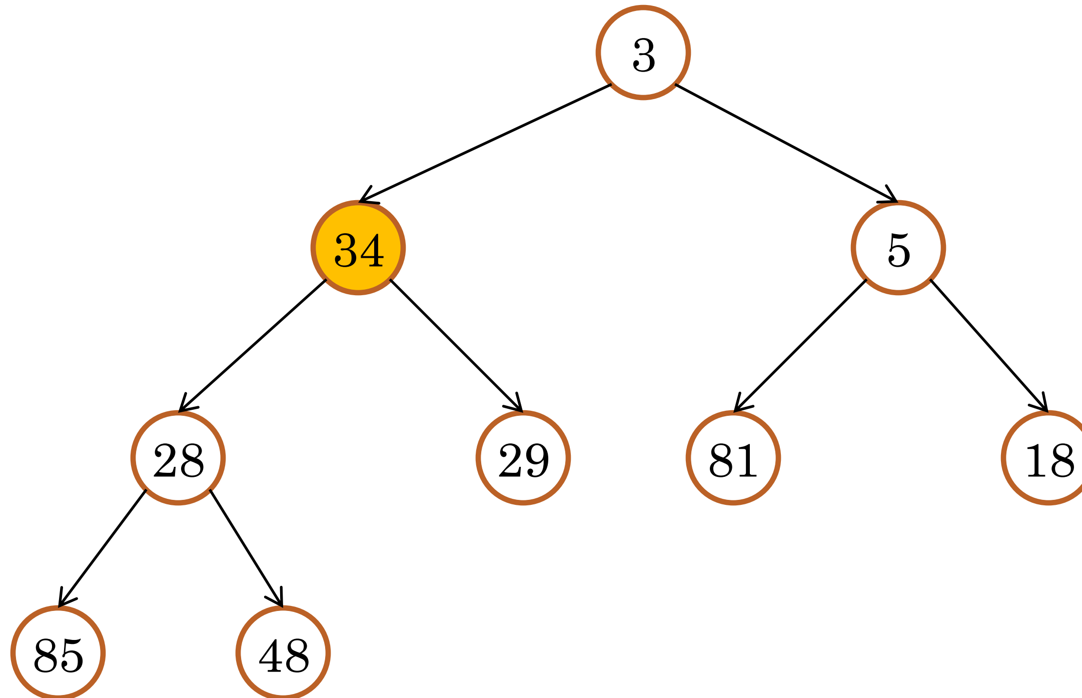
Primer – Brisanje elementa iz kopice (2/4)

Element v korenu nadomestimo z najbolj desnim elementom na zadnjem nivoju…



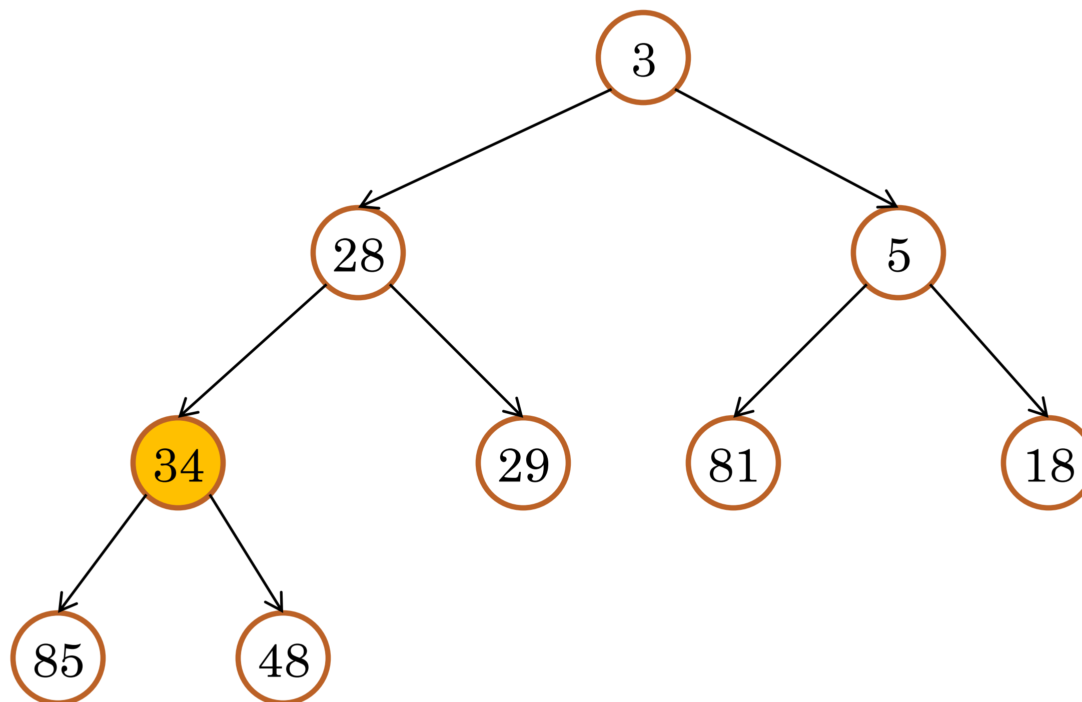
Primer – Brisanje elementa iz kopice (3/4)

…zaporedoma ga zamenjujemo z manjšim od obeh sinov…



Primer – Brisanje elementa iz kopice (4/4)

…dokler ni večji od obeh sinov ali pride v list kopice.



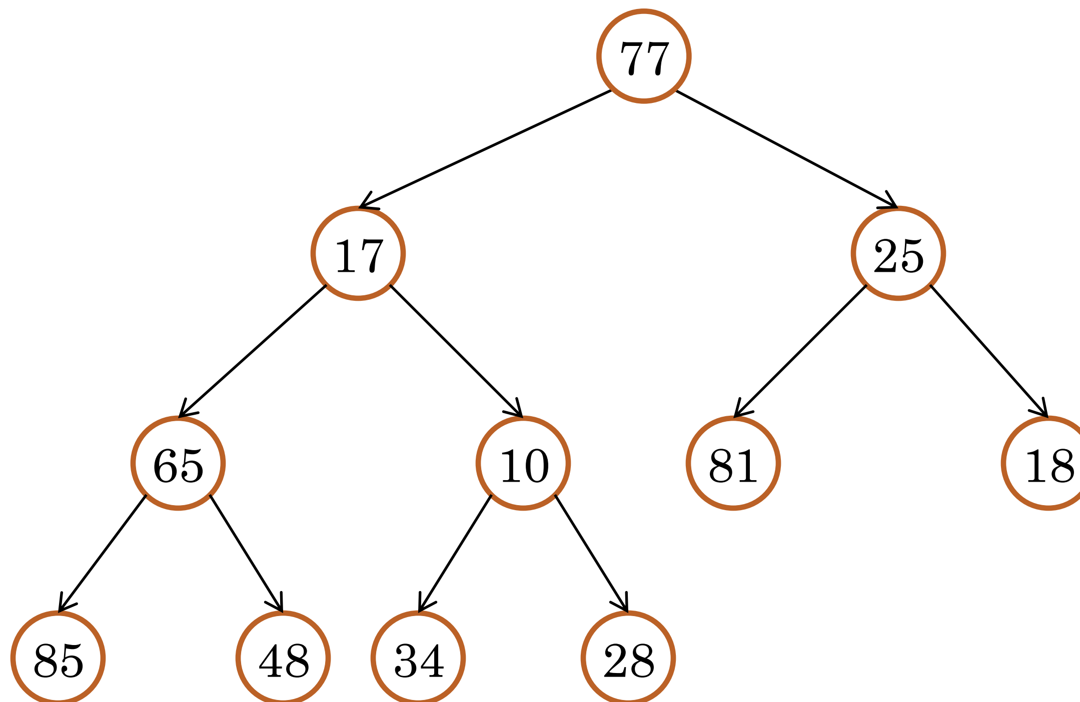
Izgradnja kopice

- kopico z n elementi zgradimo v času reda
  - $O(n\log n)$, če n krat uporabimo INSERT
  - $O(n)$, če so vsi elementi podani na začetku:

    1) elemente najprej kar v poljubnem vrstnem redu postavimo v kopico, ki je tako levo poravnana;
    2) kopico urejamo po nivojih od spodaj navzgor;

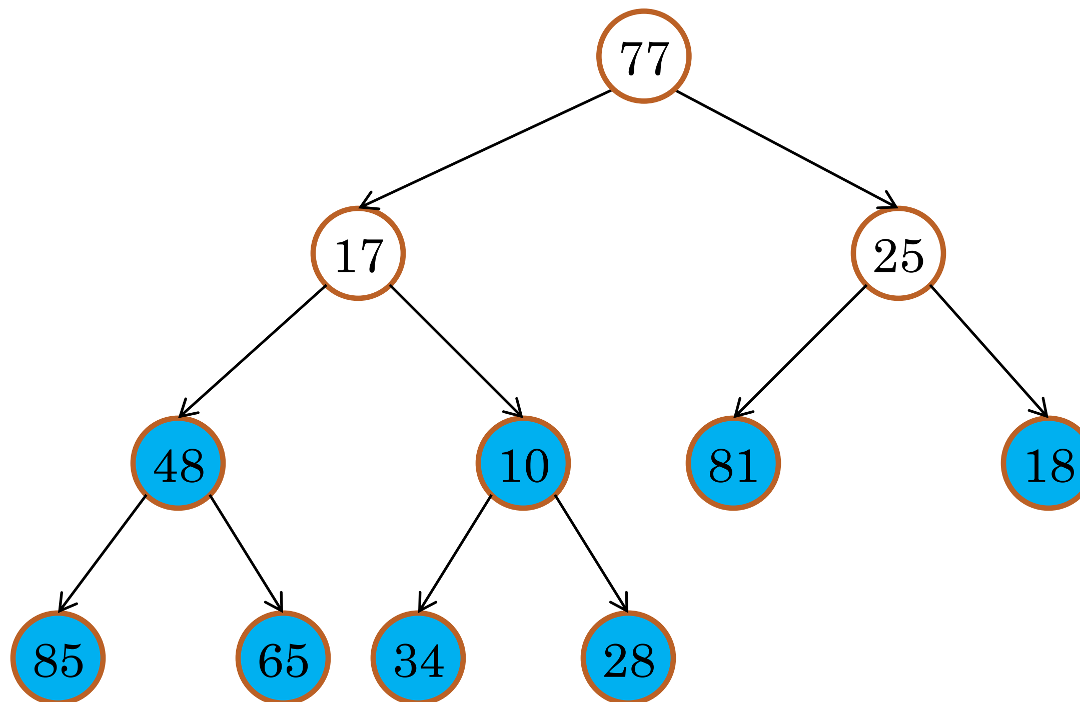
Primer – Izgradnja kopice

Elemente v poljubnem vrstnem redu postavimo v kopico…



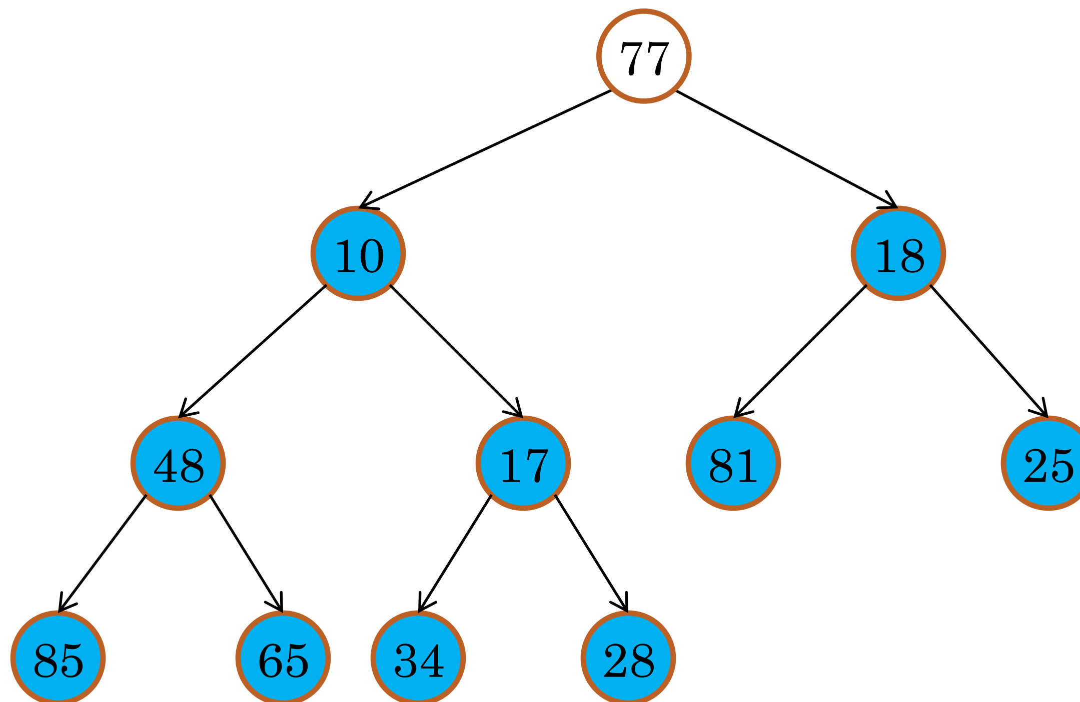
Primer – Izgradnja kopice

Uredimo najnižji nivo…



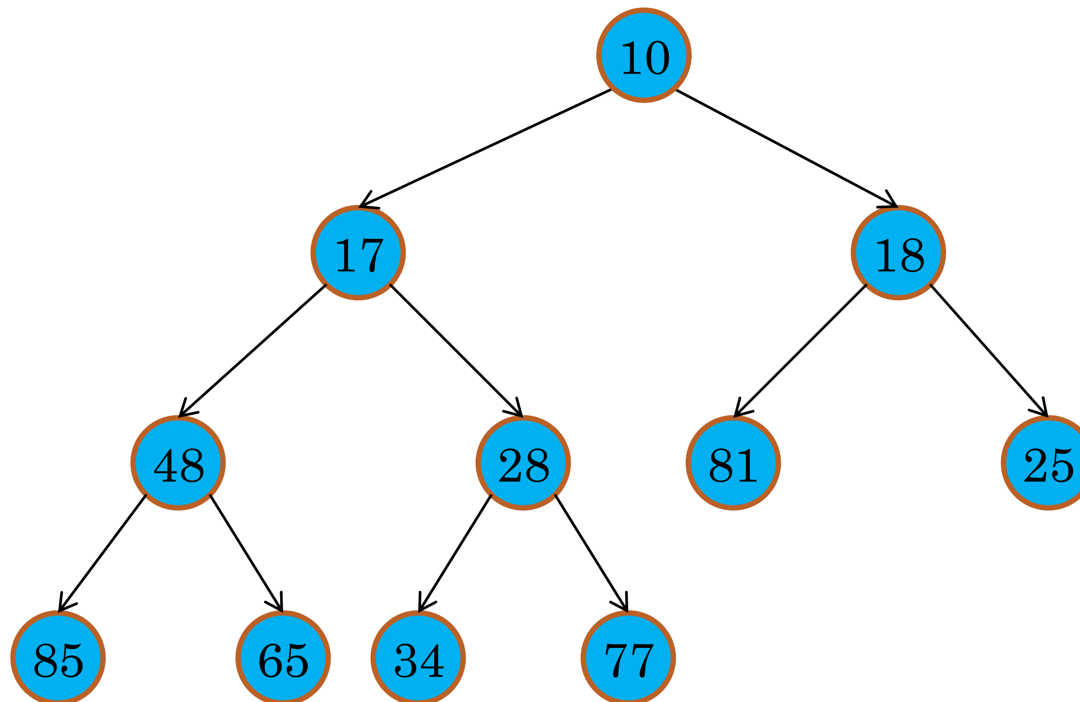
Primer – Izgradnja kopice

Uredimo še naslednji nivo…



Primer – Izgradnja kopice

Uredimo še zadnji nivo…

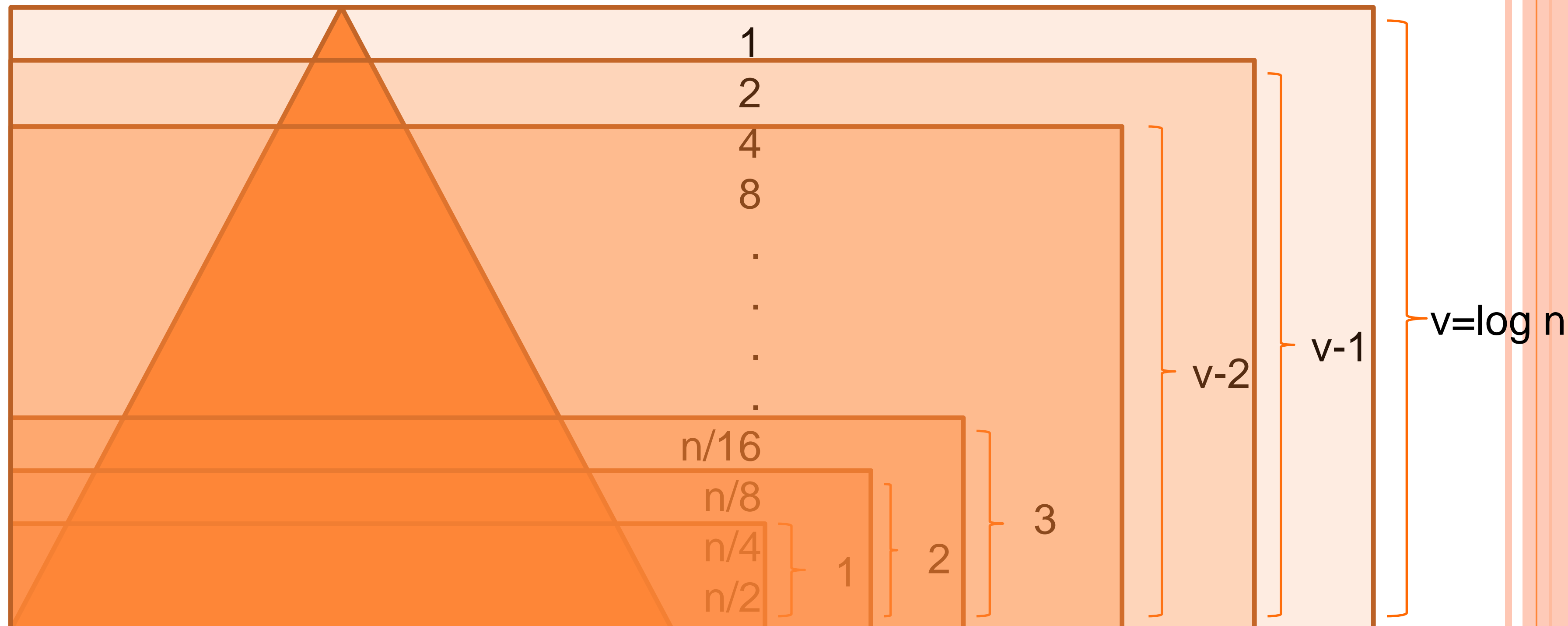


Izgradnja kopice: zahtevnost

- kopico z n elementi zgradimo v času reda
  - $O(n\log n)$, če n krat uporabimo INSERT
  - $O(n)$, če so vsi elementi podani na začetku:

  1) elemente najprej kar v poljubnem vrstnem redu postavimo v kopico, ki je tako levo poravnana;

  2) kopico urejamo po nivojih od spodaj navzgor;



Izgradnja kopice: zahtevnost

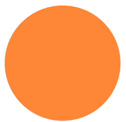
Število korakov: $$\sum_{i=1}^{\log n} i \times \frac{n}{2^i} = n \sum_{v=1}^{\log n} \sum_{i=v}^{\log n} \frac{1}{2^i}$$

Uporabimo neenakost: $$\sum_{i=v}^{\log n} \frac{1}{2^i} < \frac{1}{2^{v-1}}$$

Dobimo: $$n \sum_{v=1}^{\log n} \sum_{i=v}^{\log n} \frac{1}{2^i} \quad < \quad n \sum_{v=1}^{\log n} \frac{1}{2^{v-1}}$$

$$< \quad n \sum_{i=0}^{(\log n)-1} \frac{1}{2^i}$$

$$< \quad 2n$$



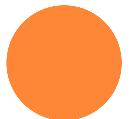
Uporaba kopice

- Urejanje (sortiranje) množice elementov:
  1) zgradi kopico – $O(n)$
  2) po vrsti jemlji elemente od najmanjšega do največjega – $O(n \log n)$

  Heapsort zahteva $O(n) + O(n \log n) = O(n \log n)$ časa

- Algoritem Dijkstra za gradnjo drevesa najkrajših poti
- Primov algoritem za gradnjo minimalnega vpetega drevesa
- Kruskalov algoritem za gradnjo minimalnega vpetega drevesa

} DECREASEKEY



ADT Prioritetna vrsta

ADT PRIORITY QUEUE:

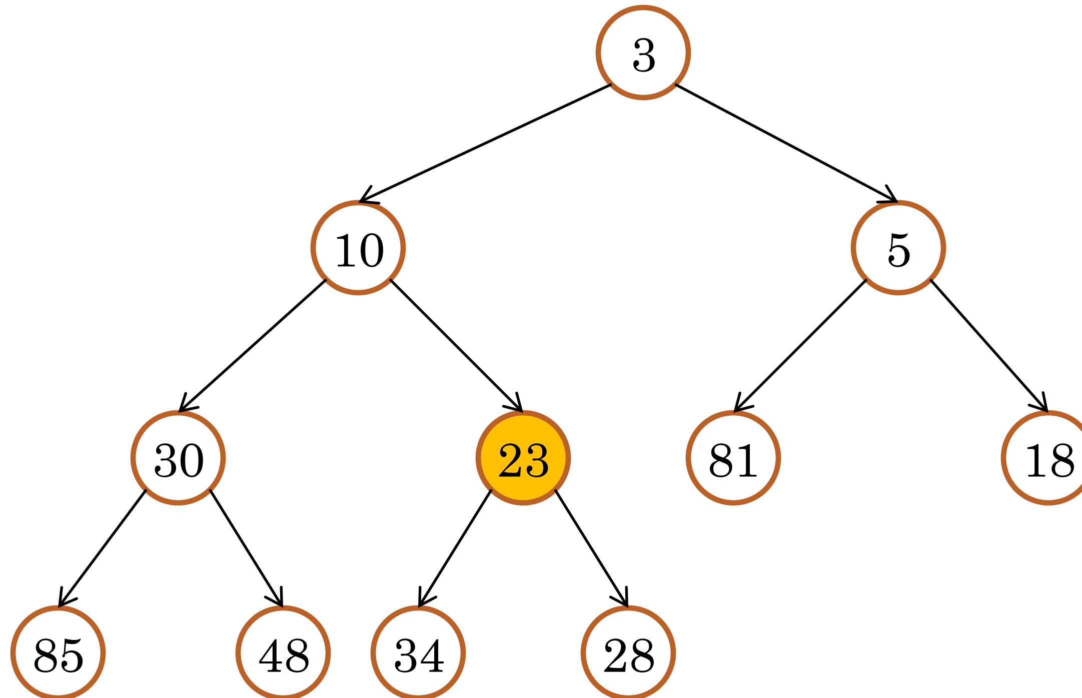
- MAKENULL(Q) : napravi prazno prioritetno vrsto Q
- INSERT(x, Q) : vstavi element x v prioritetno vrsto Q
- DELETEMIN(Q) : vrne element z najmanjšo prioriteto iz prioritetne vrste Q in ga zbriše iz Q
- EMPTY(Q) : ali je prioritetna vrsta Q prazna

Za algoritme na grafih potrebujemo še operacijo:

- DECREASEKEY(x,k,Q) : elementu *x* v kopici zmanjša ključ na *k*

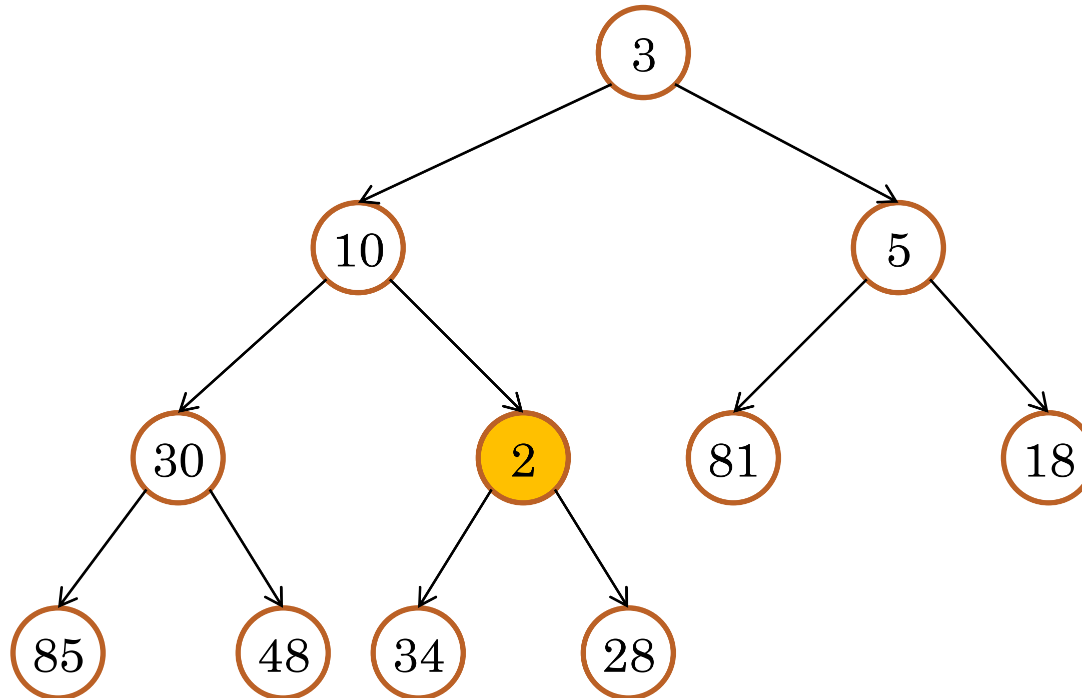
Primer – DECREASEKEY (1/3)

DECREASEKEY: 23→2



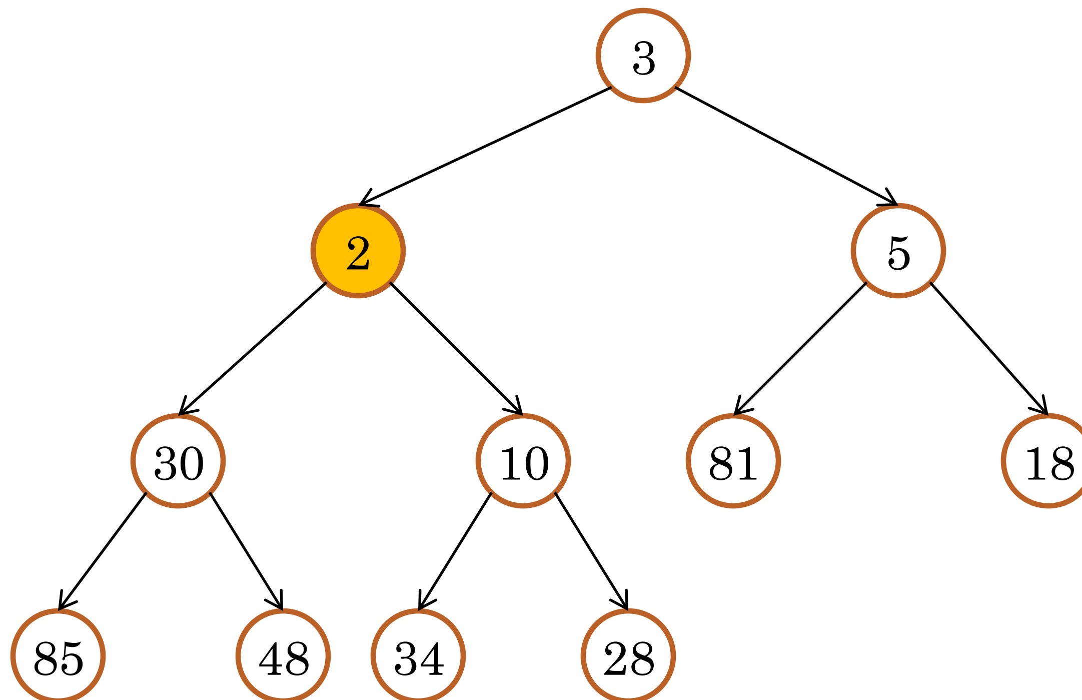
PRIMER – DECREASEKEY (1/3)

DECREASEKEY: 23→2



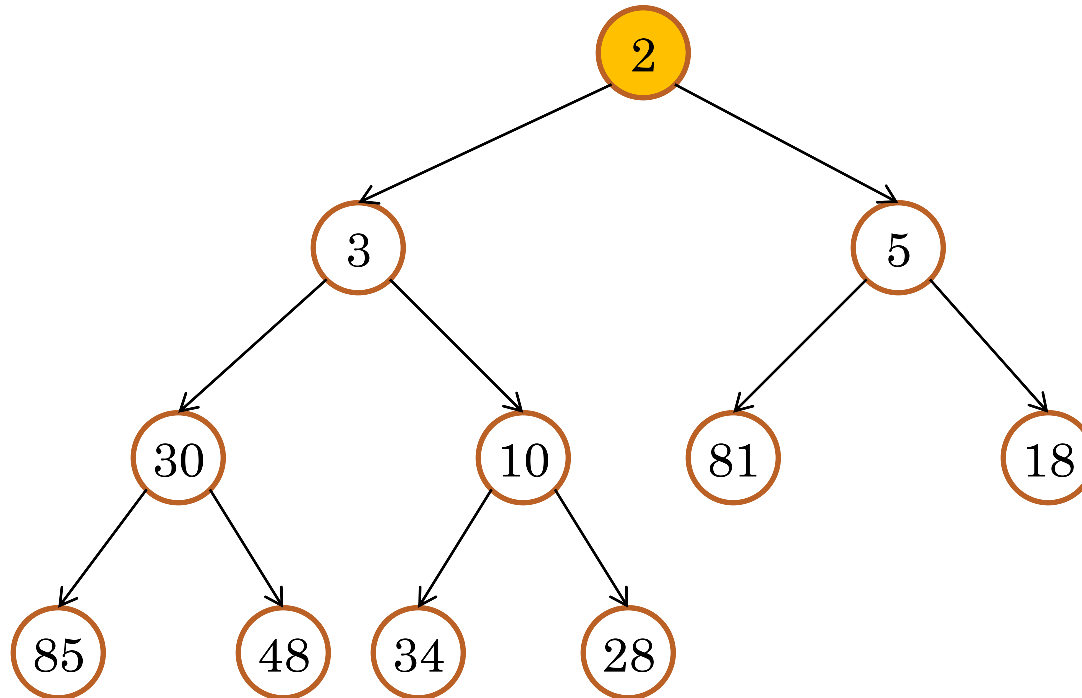
PRIMER – DECREASEKEY (2/3)

Zamenjujemo z očetom…



Primer – DECREASEKEY (3/3)

…dokler ni oče manjši ali ne pridemo do korena



Implementacija DECREASEKEY

DECREASEKEY:

1)Najdi element $O(n)$ → $O(1)$

2)Spremeni ključ $O(1)$

3)Popravi položaj $O(\log n)$

Vozlišče grafa

Za učinkovito implementacijo potrebujem direkten dostop do elementa

vsak element mora hraniti kazalec na svoj položaj v kopici