

Algoritmi i strukture podataka

07 Red sa prioritetom, Heap

Katedra za informatiku, Fakultet tehničkih nauka, Novi Sad

2024

Red sa prioritetom

- Element je sačinjen od **vrednosti** i **prioriteta**
- Prioritet (ključ) je vrednost zavisna od problema koji se rešava
- Ključevi moraju biti **uporedivi**
- Operacije:
 - `PQ.add(key, value)`
 - `PQ.min()`
 - `PQ.remove_min()`
 - `PQ.is_empty()`
 - `len(PQ)`

Red sa prioritetom

- Možemo sortirati proizvoljnu kolekciju elemenata tako što jedan po jedan element ubacimo u red sa prioritetom.
- Zatim uklanjamo jedan po jedan element.
- Elementi se uklanjaju u sortiranom redosledu.

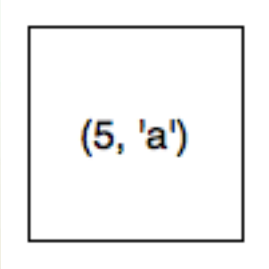
Sortirani red sa prioritetom

- Elementi reda se održavaju sortirani po ključu
- Novi element se ubacuje na poziciju koja odgovara prioritetu ključa
- Pronalaženje i uklanjanje minimalnog elementa je jednostavno s obzirom da se uvek nalazi na početku
- Sortiranje elemenata na način koji simulira funkcionisanje sortiranog reda sa prioritetom čini algoritam koji se zove insertion sort.

Primer upotrebe sortiranog PQ

- Kreiramo red sa prioritetom

```
pq = SortedPriorityQueue()  
pq.add(5, 'a')
```



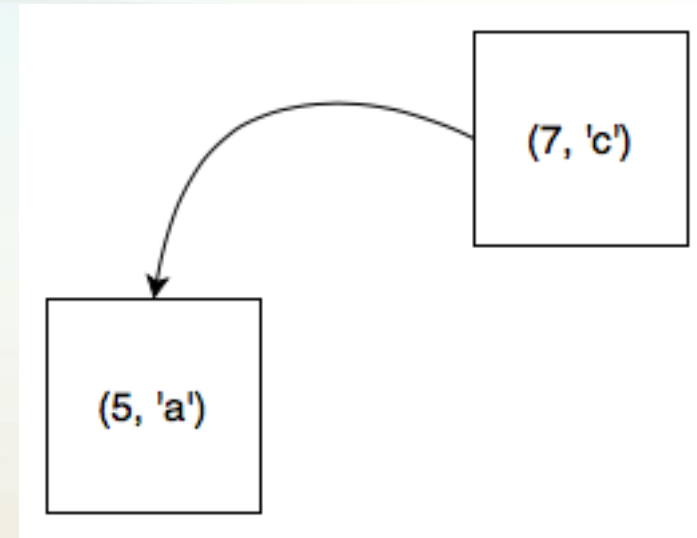
(5, 'a')

- Pošto je red sa prioritetom prazan, novi element se dodaje na kraj.

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(7, 'c')`



- Novi element se poredi sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

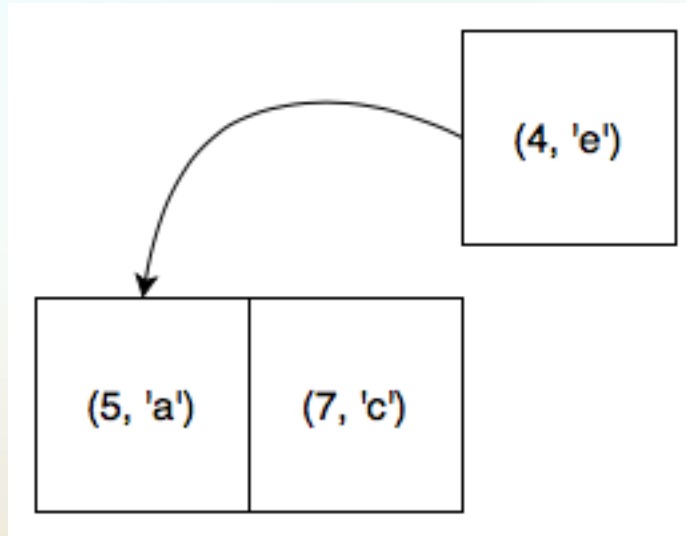
- Stanje posle dodavanja

(5, 'a')	(7, 'c')
----------	----------

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(4, 'e')`



- Novi element se poredi redom sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

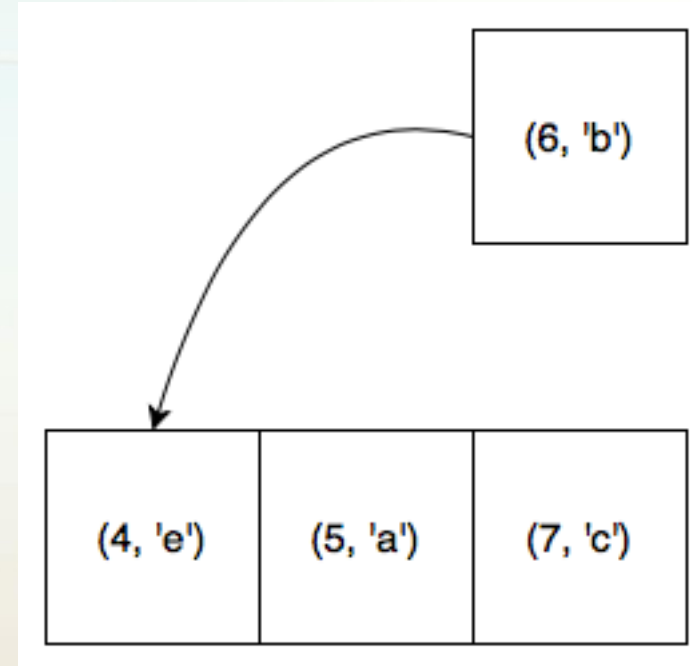
- Novi element je ubačen

(4, 'e')	(5, 'a')	(7, 'c')
----------	----------	----------

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(6, 'b')`

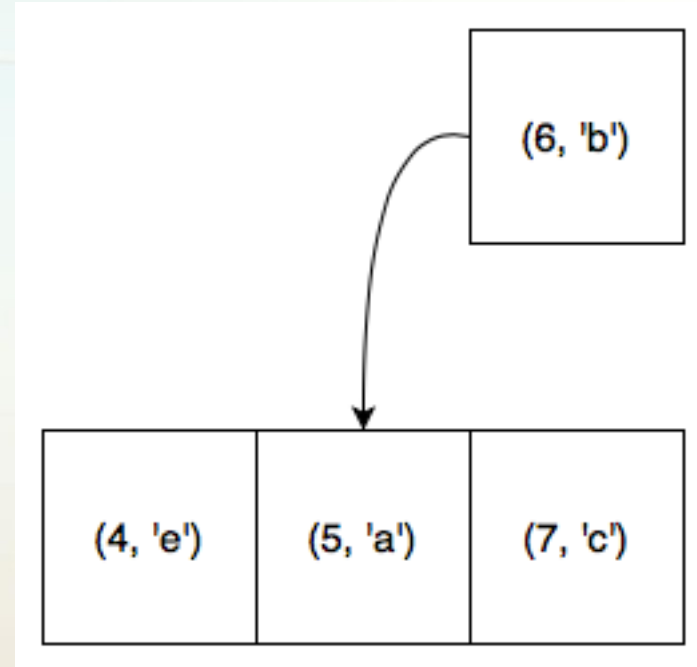


- Novi element se poredi redom sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(6, 'b')`

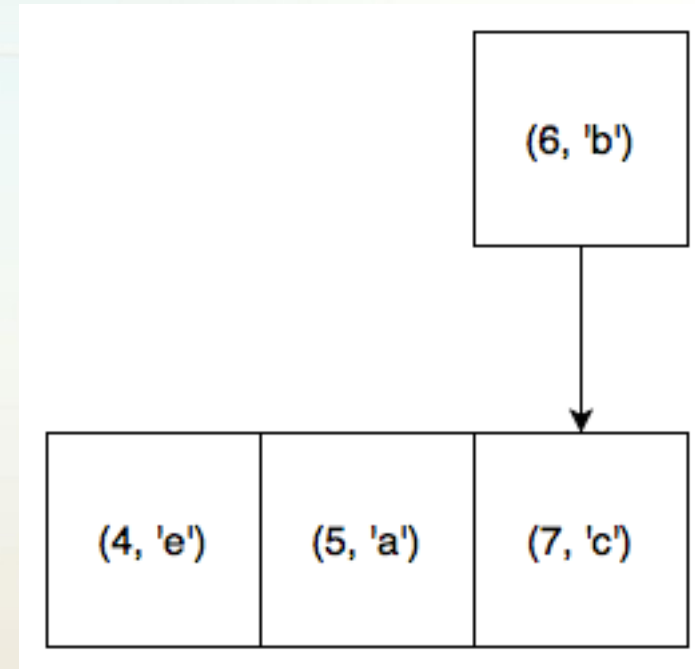


- Novi element se poredi redom sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(6, 'b')`

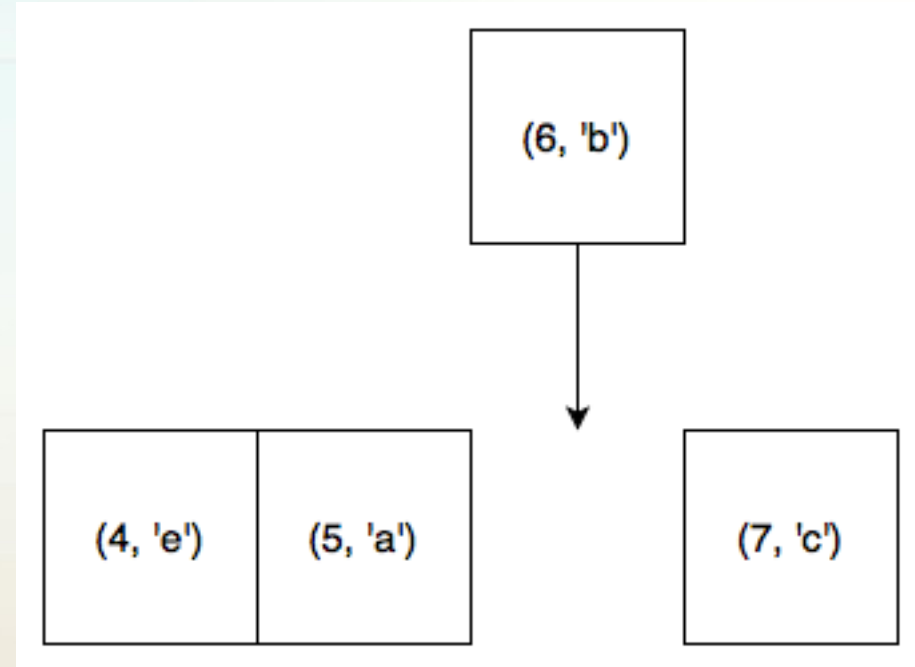


- Novi element se poredi redom sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

- Dodajemo novi element

`pq.add(6, 'b')`



- Novi element se poredi redom sa postojećim. Smešta se pre elementa sa većim ključem ili na kraj.

Primer upotrebe sortiranog PQ

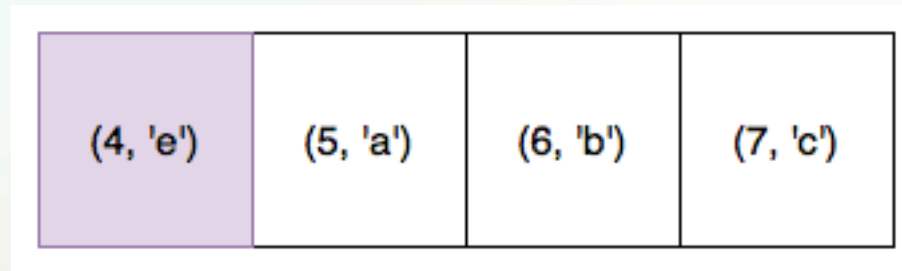
- Stanje reda sa prioritetom posle dodavanja

(4, 'e')	(5, 'a')	(6, 'b')	(7, 'c')
----------	----------	----------	----------

Primer upotrebe sortiranog PQ

- Pristupamo minimalnom elementu

`pq.min()`

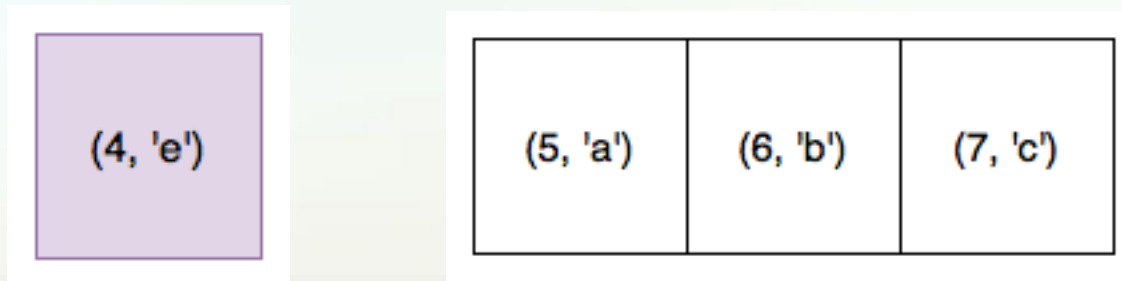


- Kod sortiranog reda sa prioritetom najmanji element se nalazi na početku.
- Složenost?

Primer upotrebe sortiranog PQ

- Uklanjamo minimalni element

`pq.remove_min()`



- Kod sortiranog reda sa prioritetom najmanji element se nalazi na početku.
- Složenost?

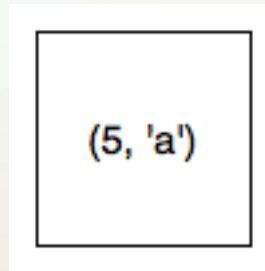
Nesortirani PQ

- Novi element se dodaje na kraj
- Pronalaženje i uklanjanje minimalnog je kompleksnije jer moramo pronaći element koji uklanjamo
- Sortiranje elemenata na način koji simulira funkcionisanje nesortiranog reda sa prioritetom čini algoritam koji se zove selection sort.

Primer upotrebe nesortiranog PQ

- Kreiramo red sa prioritetom

```
pq = UnsortedPriorityQueue()  
pq.add(5, 'a')
```

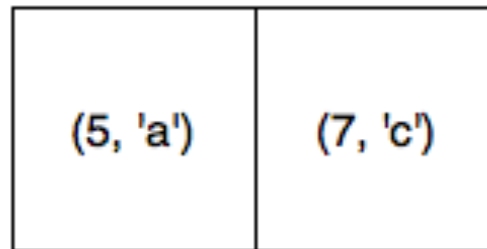


- Novi element se uvek dodaje na kraj.

Primer upotrebe nesortiranog PQ

- Dodajemo novi element

`pq.add(7, 'c')`



- Novi element se uvek dodaje na kraj.

Primer upotrebe nesortiranog PQ

- Dodajemo novi element

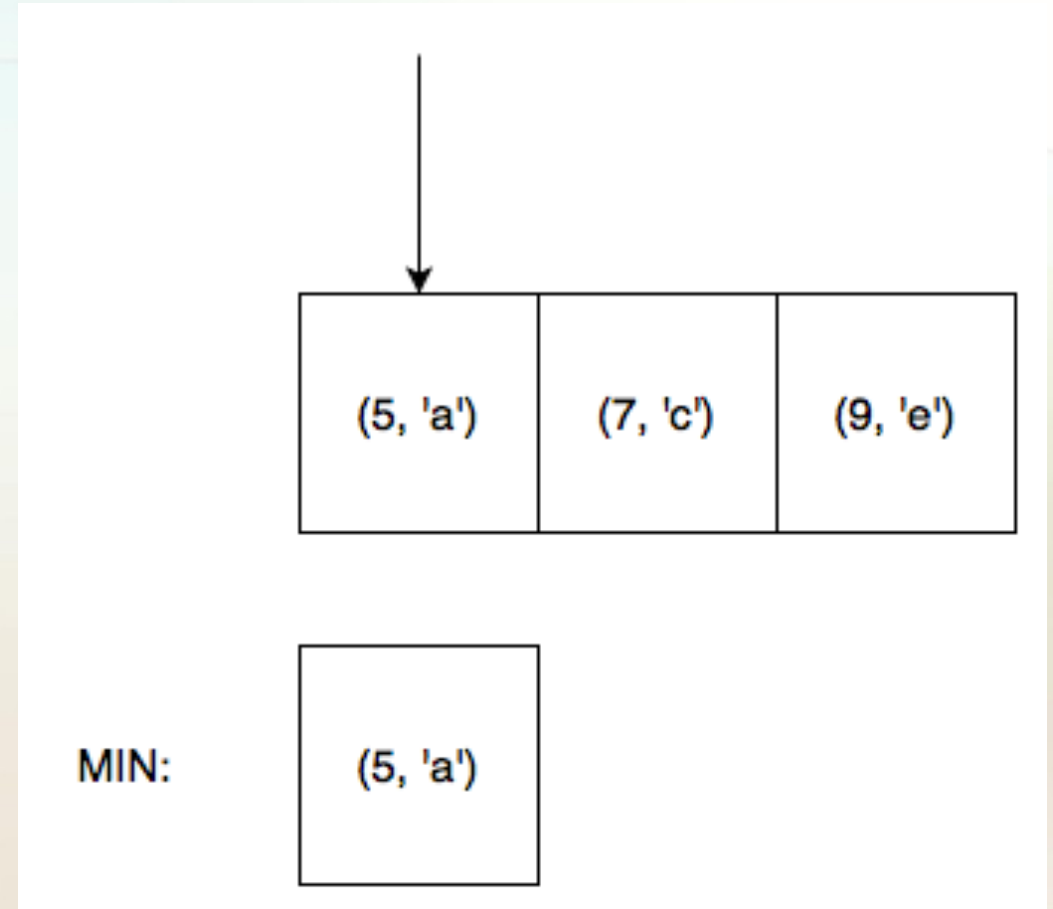
`pq.add(9, 'e')`

(5, 'a')	(7, 'c')	(9, 'e')
----------	----------	----------

- Novi element se uvek dodaje na kraj.

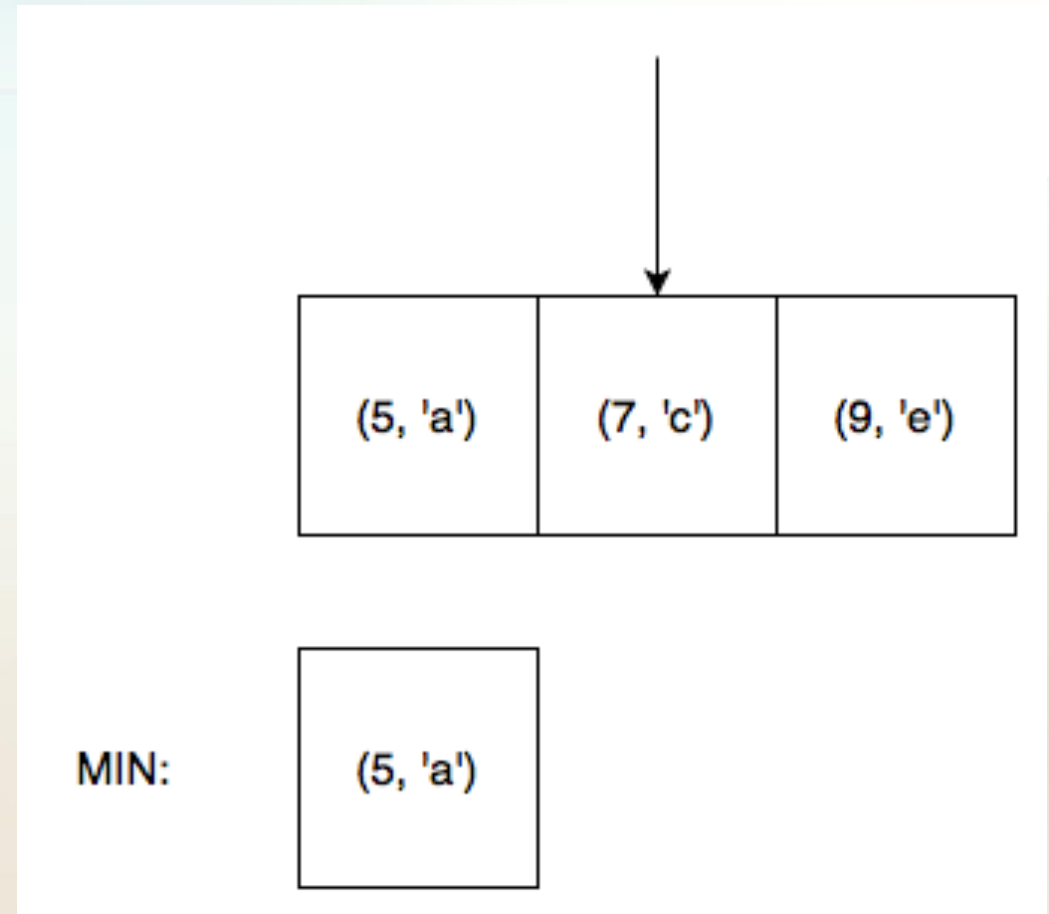
Primer upotrebe nesortiranog PQ

- Tražimo najmanji element
`pq.min()`
- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji



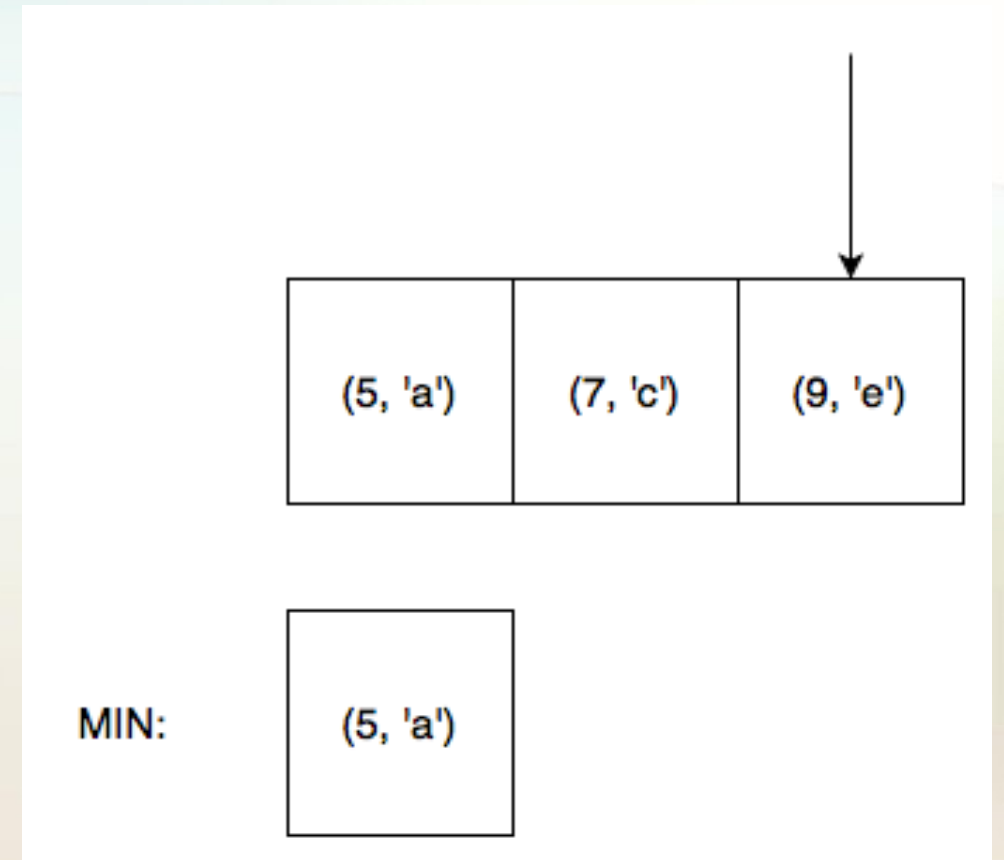
Primer upotrebe nesortiranog PQ

- Tražimo najmanji element
`pq.min()`
- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji



Primer upotrebe nesortiranog PQ

- Tražimo najmanji element
`pq.min()`
- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji
- Došli smo do kraja kolekcije, element sa ključem 5 je najmanji



Primer upotrebe nesortiranog PQ

- Dodajemo novi element

`pq.add(4, 'b')`

(5, 'a')	(7, 'c')	(9, 'e')	(4, 'b')
----------	----------	----------	----------

- Novi element se uvek dodaje na kraj.

Primer upotrebe nesortiranog PQ

- Dodajemo novi element

`pq.add(6, 'd')`

(5, 'a')	(7, 'c')	(9, 'e')	(4, 'b')	(6, 'd')
----------	----------	----------	----------	----------

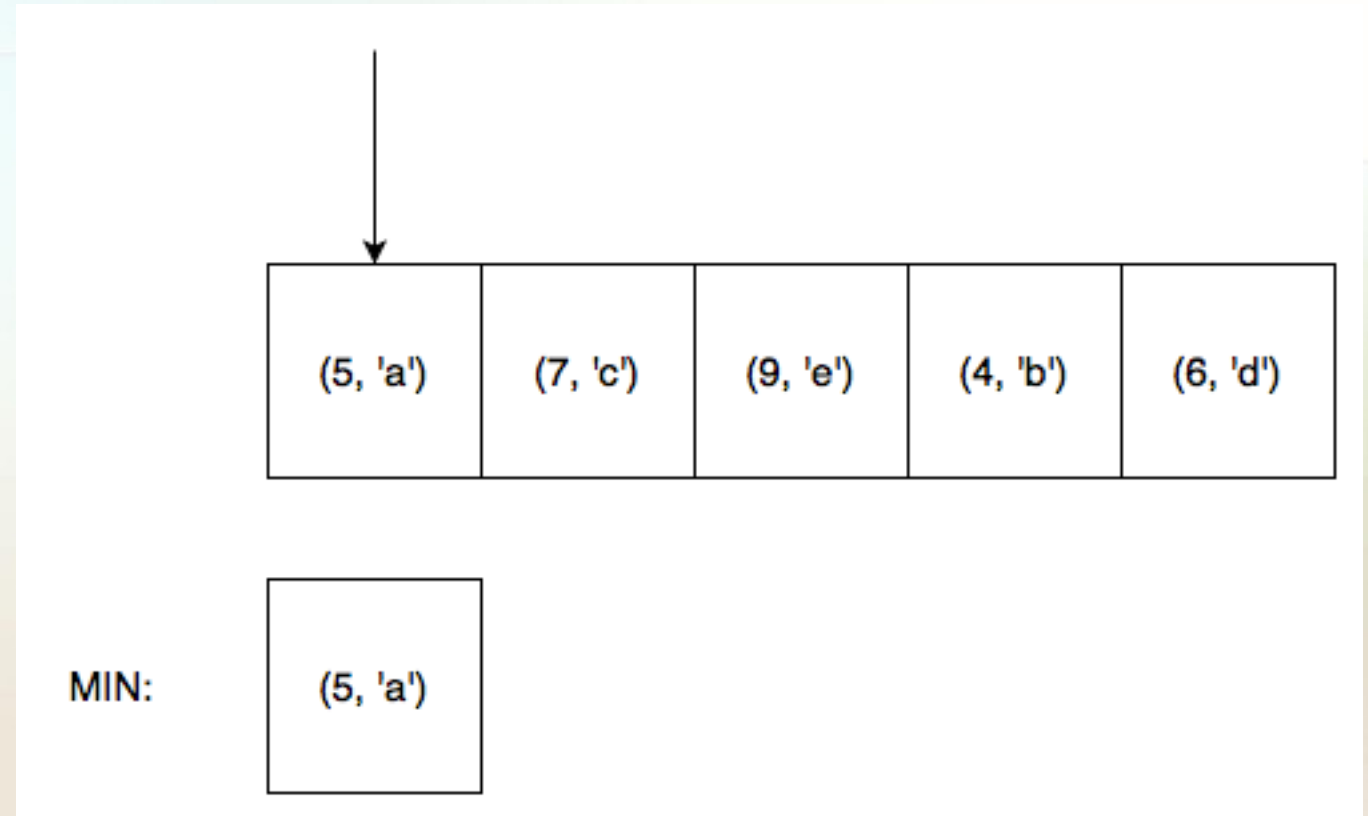
- Novi element se uvek dodaje na kraj.

Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji

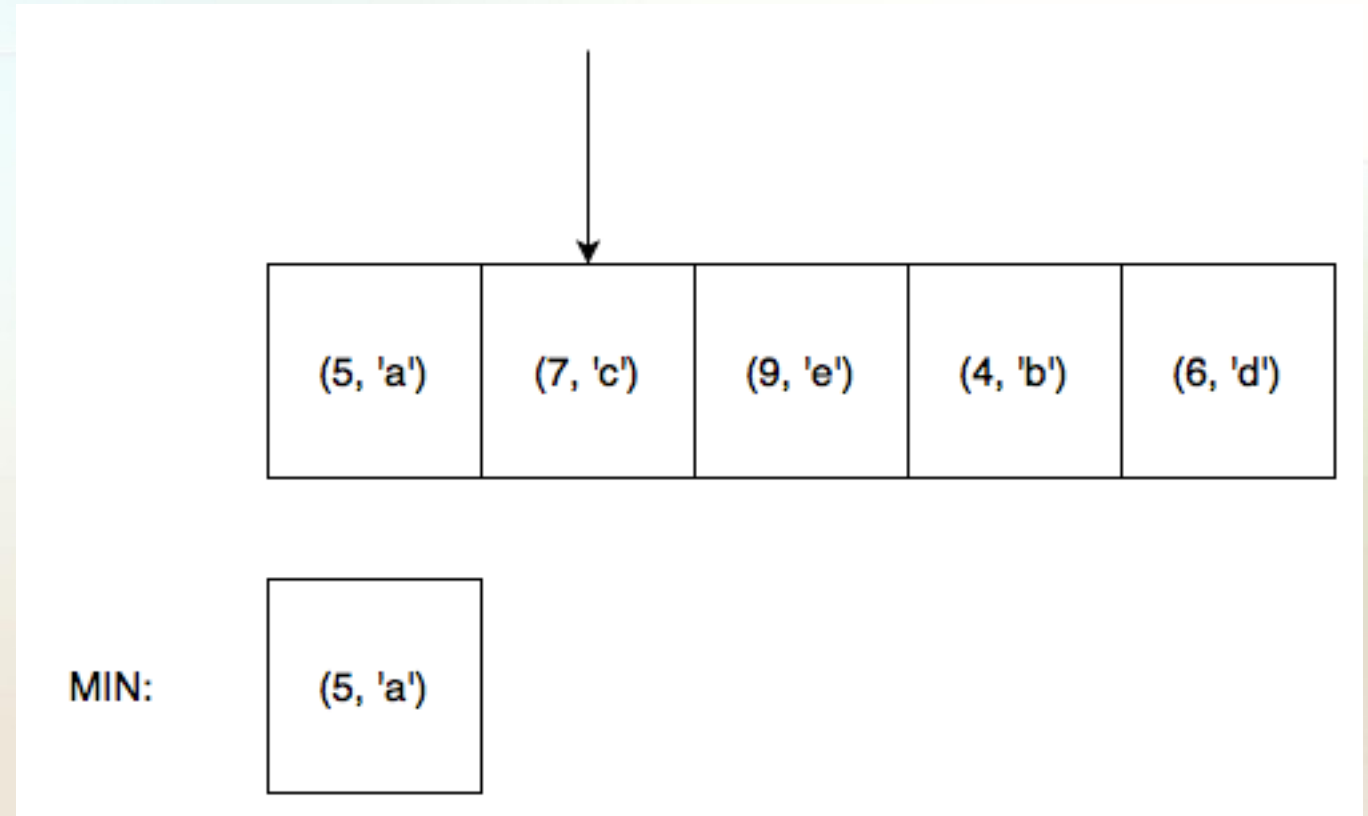


Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji

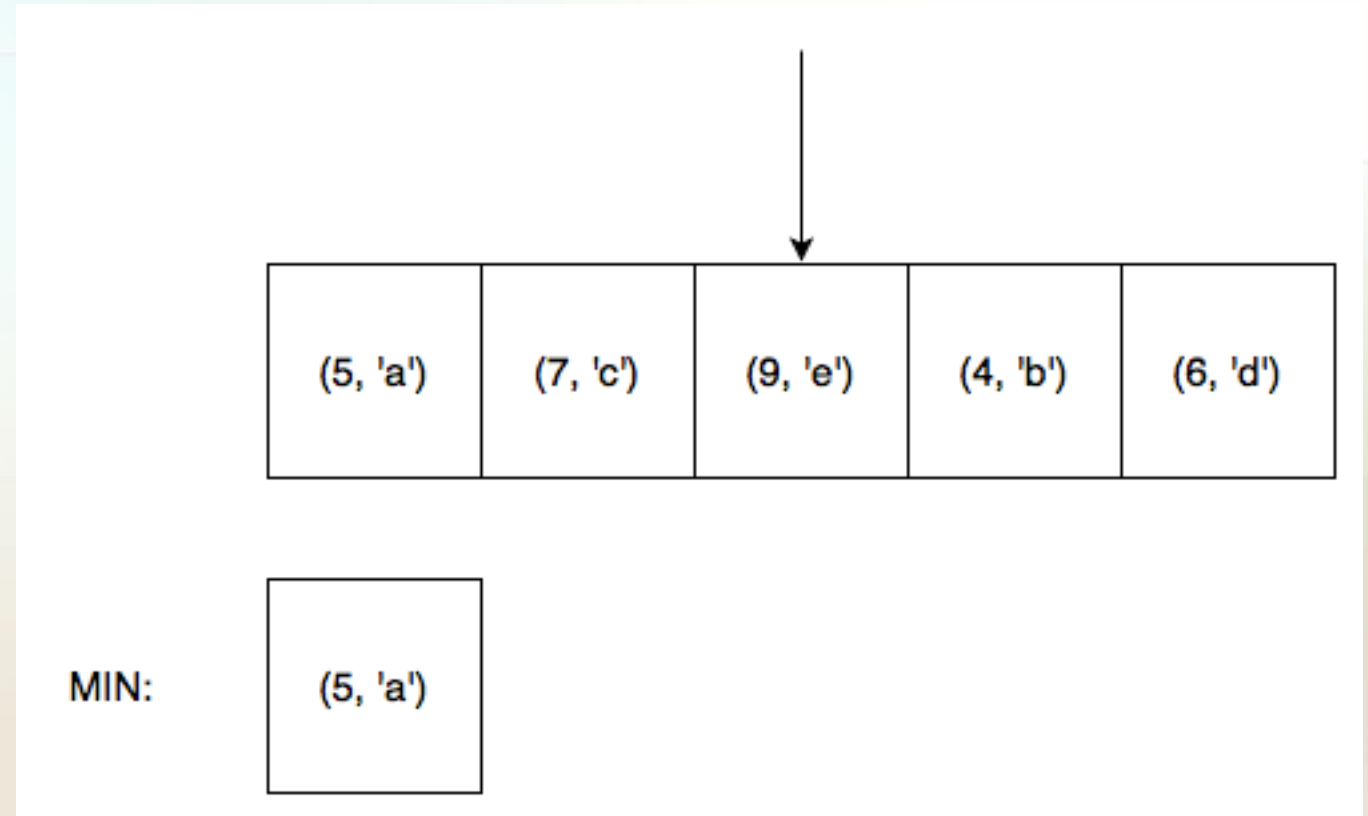


Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji

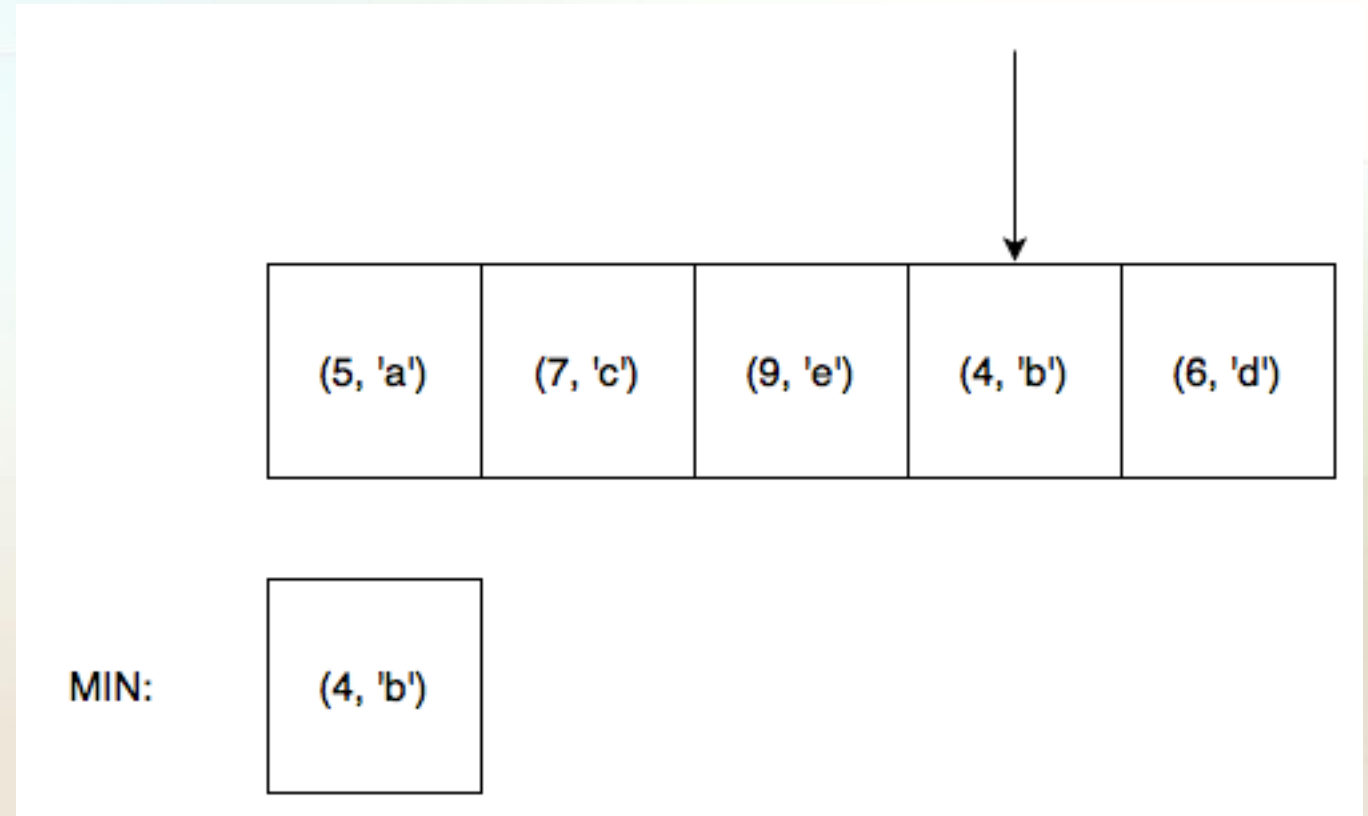


Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji

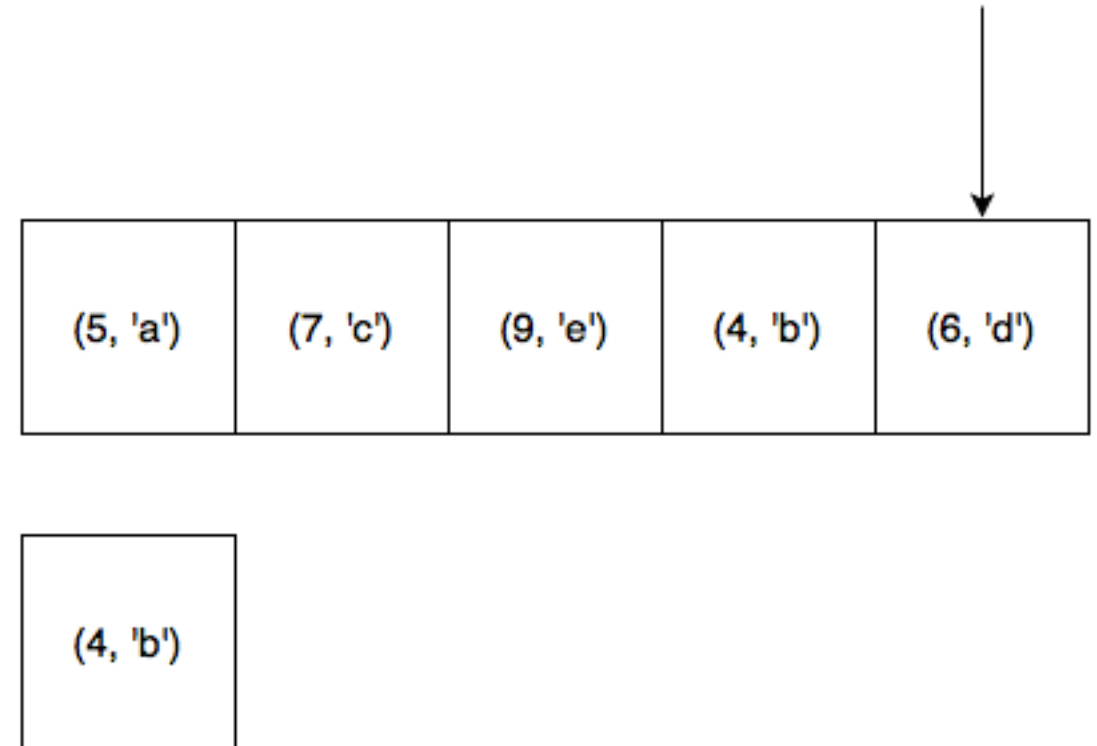


Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Pristupamo redom elementima, poredimo njihove ključeve, tražimo najmanji

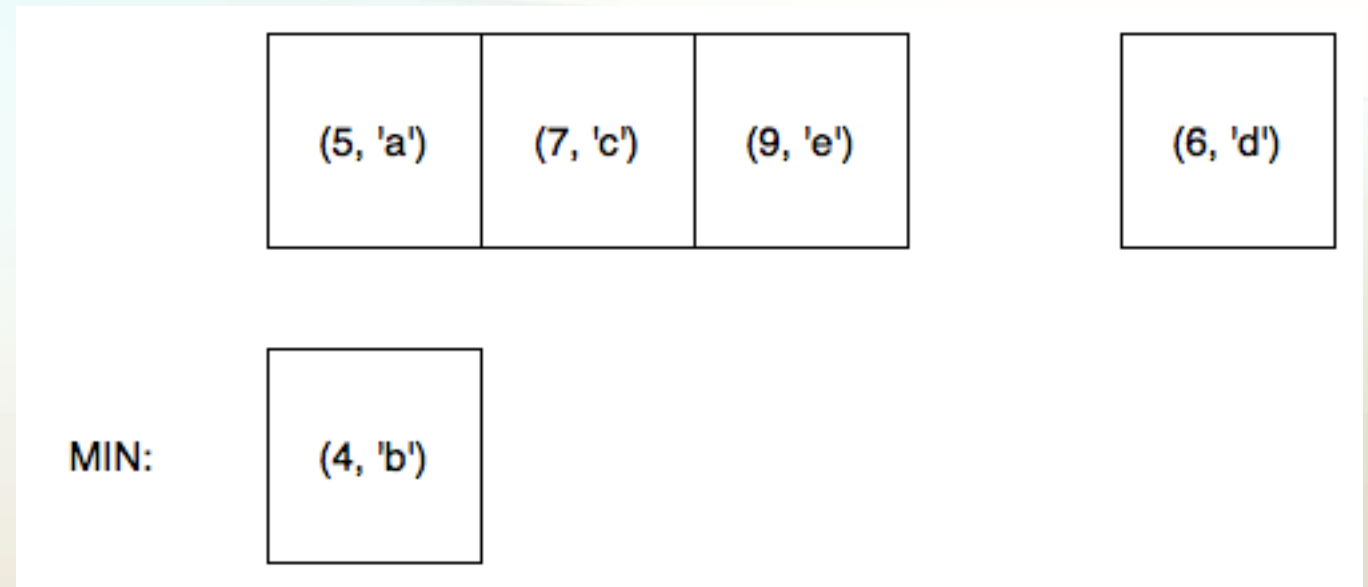


Primer upotrebe nesortiranog PQ

- Uklanjamo minimalni element

```
pq.remove_min()
```

- Uklanjamo element



Primer upotrebe nesortiranog PQ

- Stanje posle brisanja

(5, 'a')	(7, 'c')	(9, 'e')	(6, 'd')
----------	----------	----------	----------

Heap

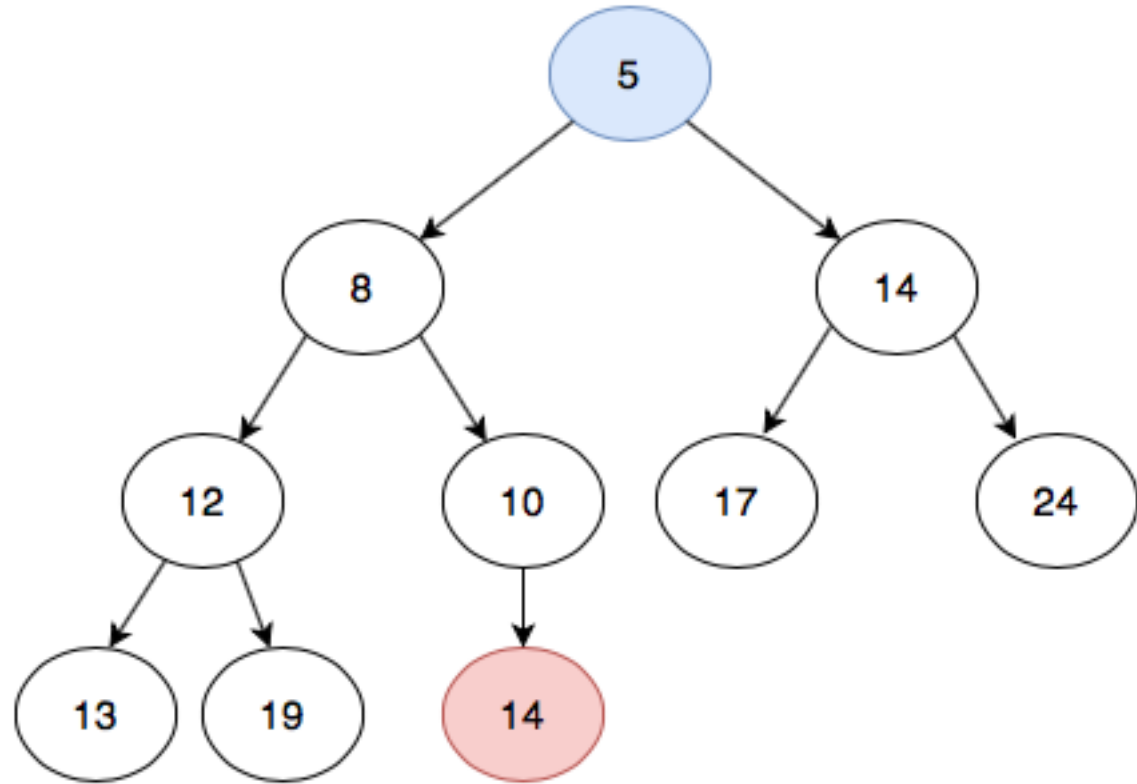
- „Skoro“ kompletno binarno stablo
- Implementacija bazirana na nizu
- Mogući načini organizovanja su **max-heap** i **min-heap**
- Min-heap operacije:
 - `H._left(i)`
 - `H._right(i)`
 - `H._parent(i)`
 - `H.build_min_heap()`
 - `H._upheap(i)`
 - `H._downheap(i)`

Heap

- Moguće je sortiranje kolekcije pomoću heap-a odnosno na način koji simulira funkcionisanje heap-a.
- Heap sort
- Razmisliti o složenosti:
 - Sortiramo kolekciju od n elemenata
 - Prva mogućnost:
 - Ubacujemo jedan po jedan element u heap
 - Zatim uklanjamo jedan po jedan.
 - Druga mogućnost:
 - Kreiramo heap od n elemenata - postupak heapify (min-heapify za min-heap i max-heapify za max-heap)

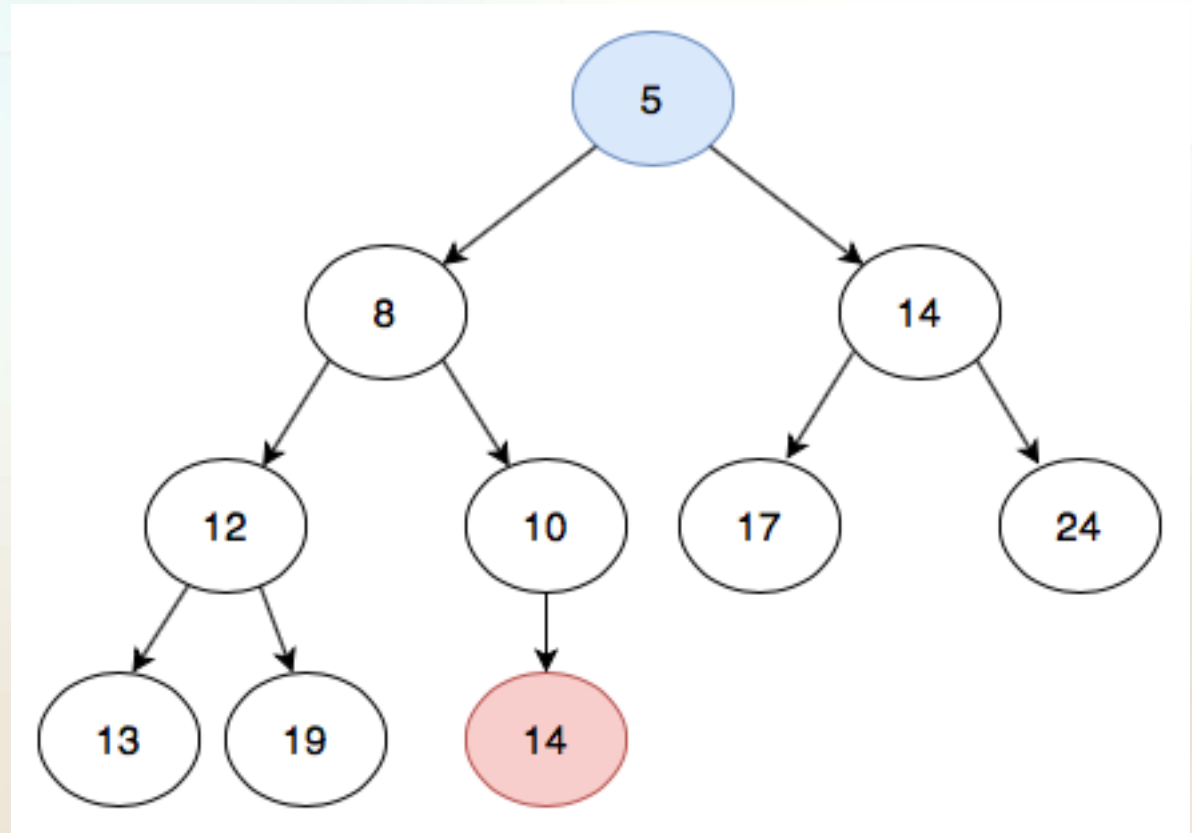
Primer Heap

- Svaki čvor heap-a je označen ključem (vrednosti nisu navedene)
- Plavom bojom označen je koren heap-a – najmanji element
- Crvenom bojom označen je poslednji element heap-a



Primer Heap

- Min-heap – Ključ roditeljskog elementa je uvek manji od ključeva dece
- (Kod max-heap-a bi ključ roditeljskog elementa bio uvek veći od ključeva dece)
- Moramo da očuvamo poredak prilikom dodavanja i brisanja elemenata



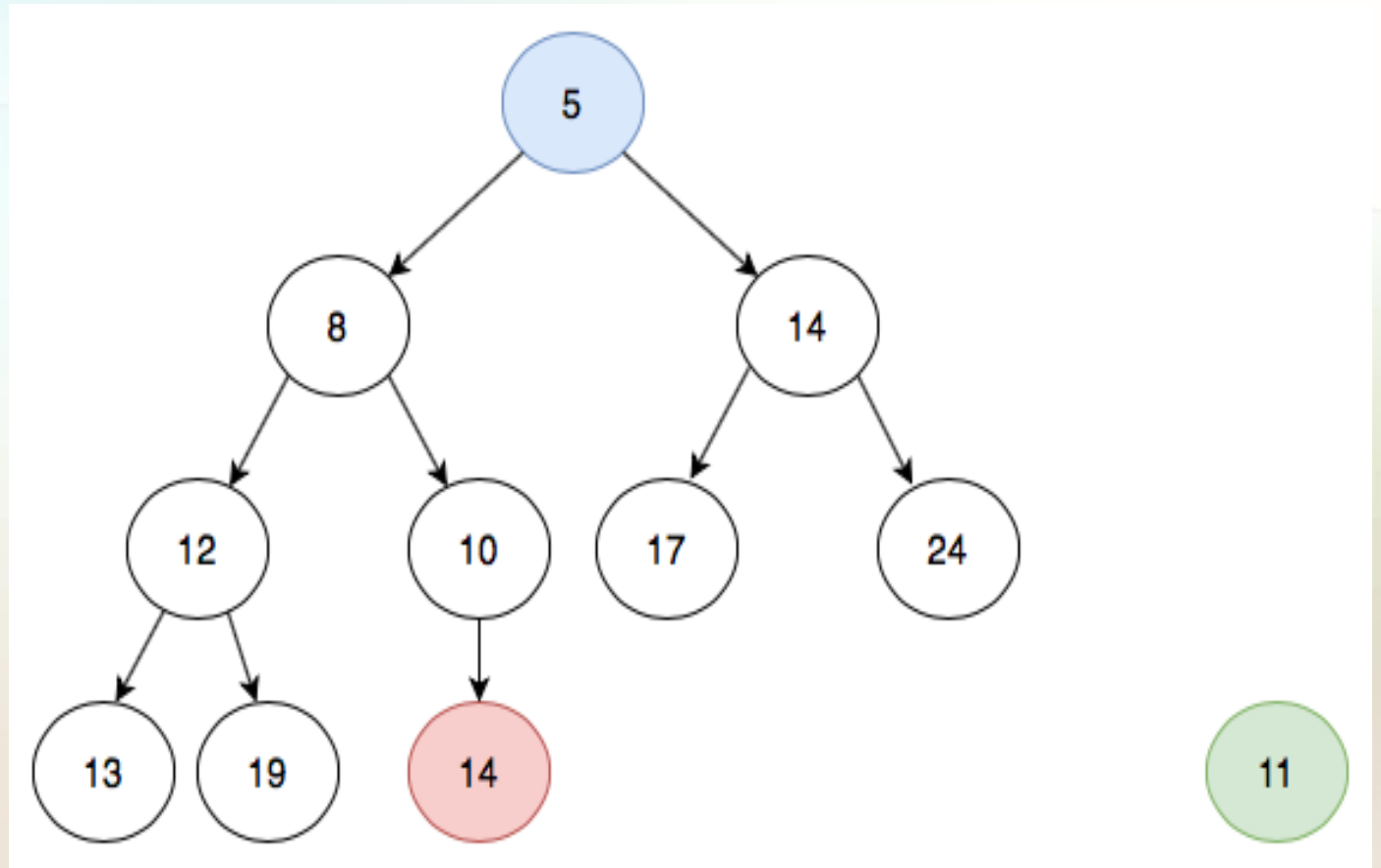
Primer Heap

- Dodajemo novi element sa ključem 11

```
heap = Heap()
```

```
heap.add(11, 'a')
```

- Novi element se dodaje posle poslednjeg



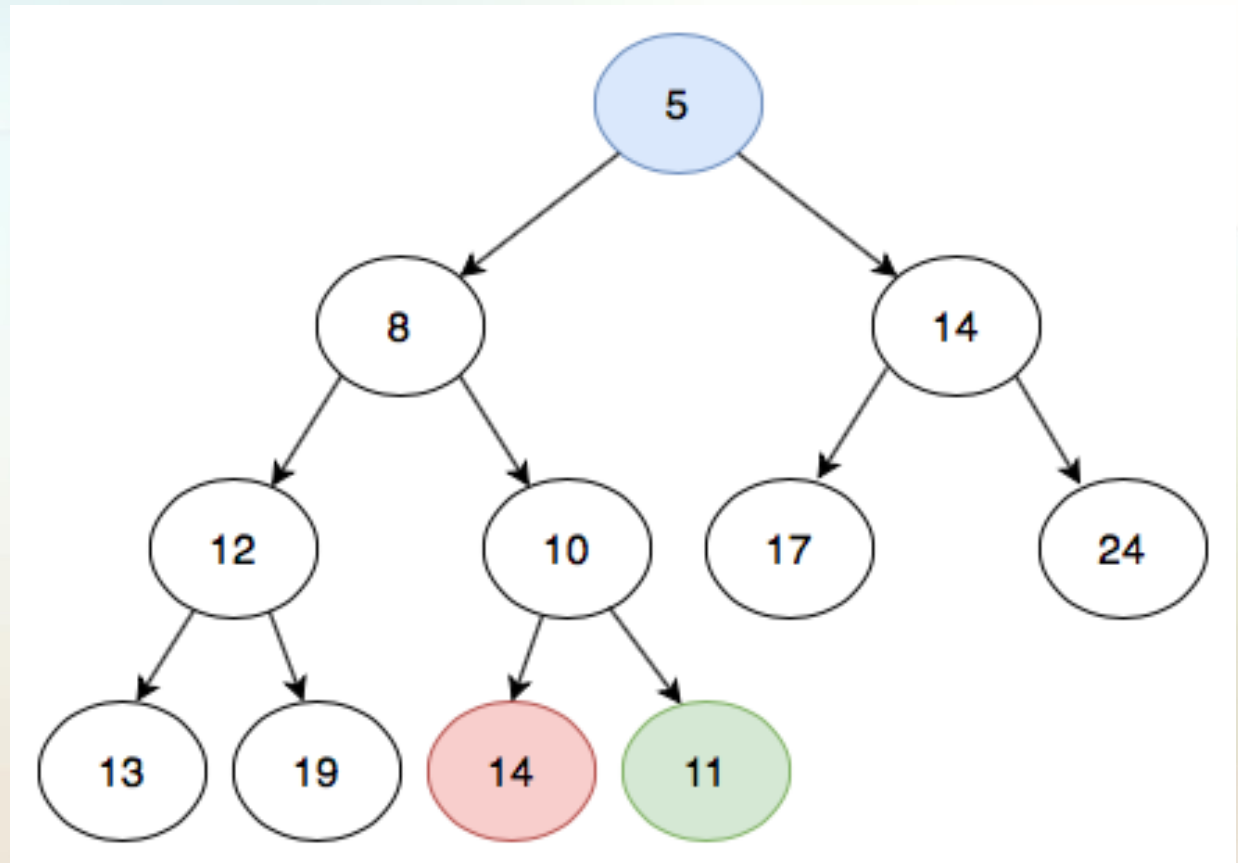
Primer Heap

- Dodajemo novi element sa ključem 11

```
heap = Heap()
```

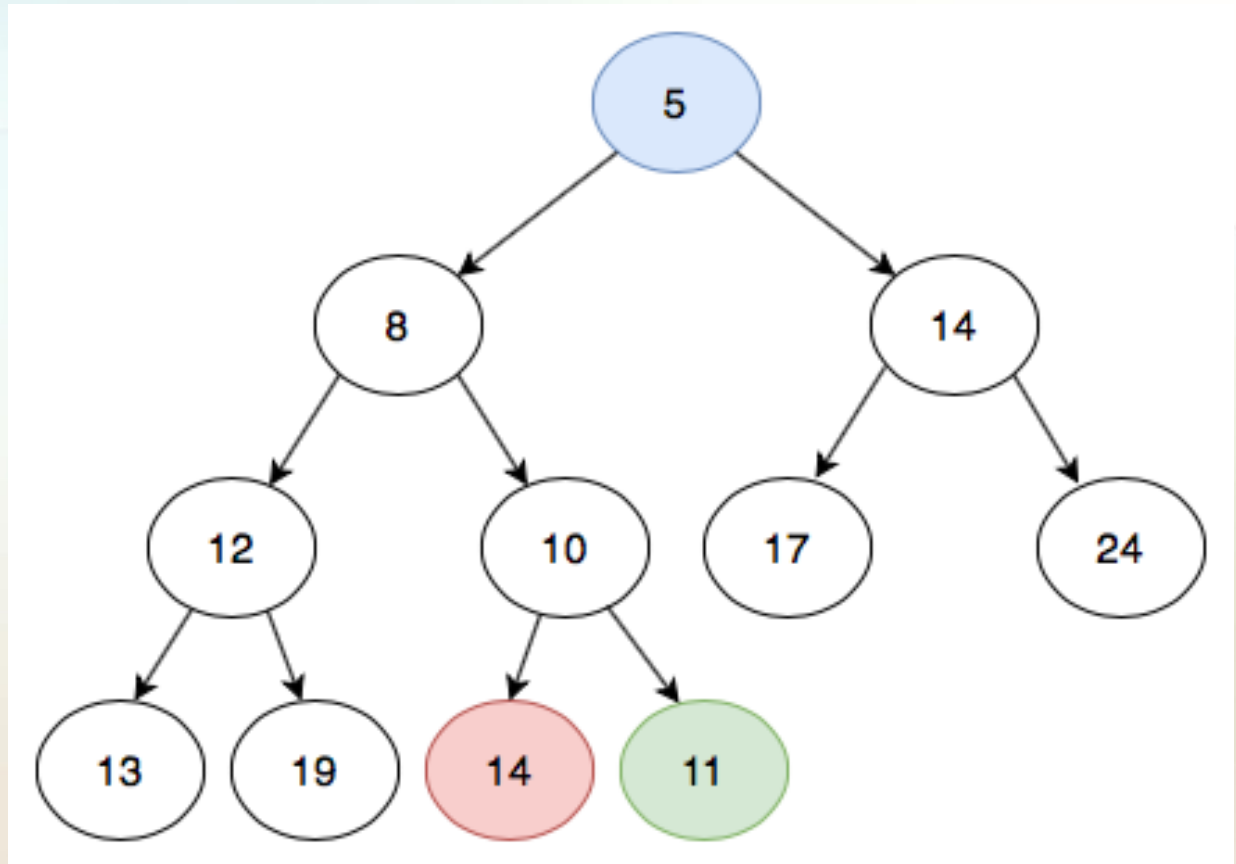
```
heap.add(11, 'a')
```

- Novi element se dodaje posle poslednjeg



Primer Heap

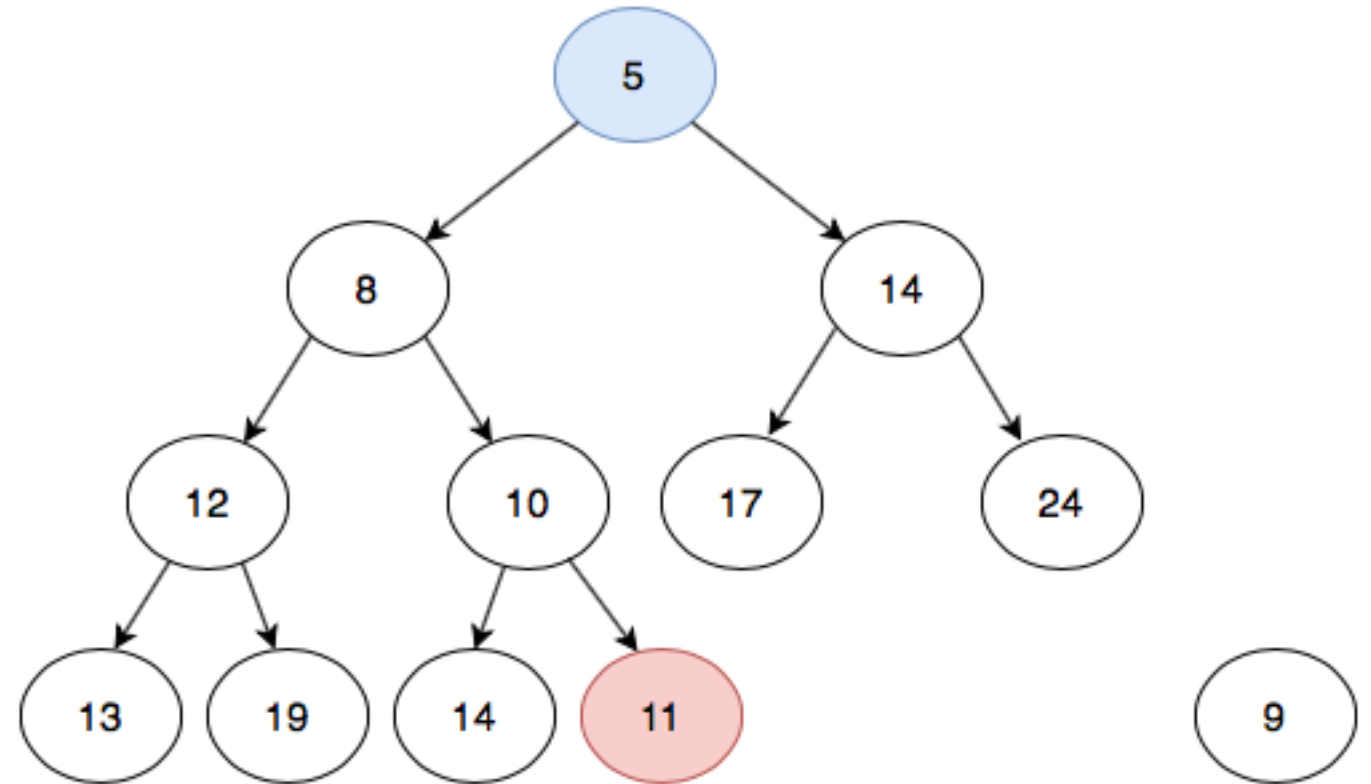
- Proveravamo da li je novi element na pravoj poziciji
- Poredimo ključ roditeljskog elementa sa ključem novog (10 i 11)
- U dobrom su poretku



Primer Heap

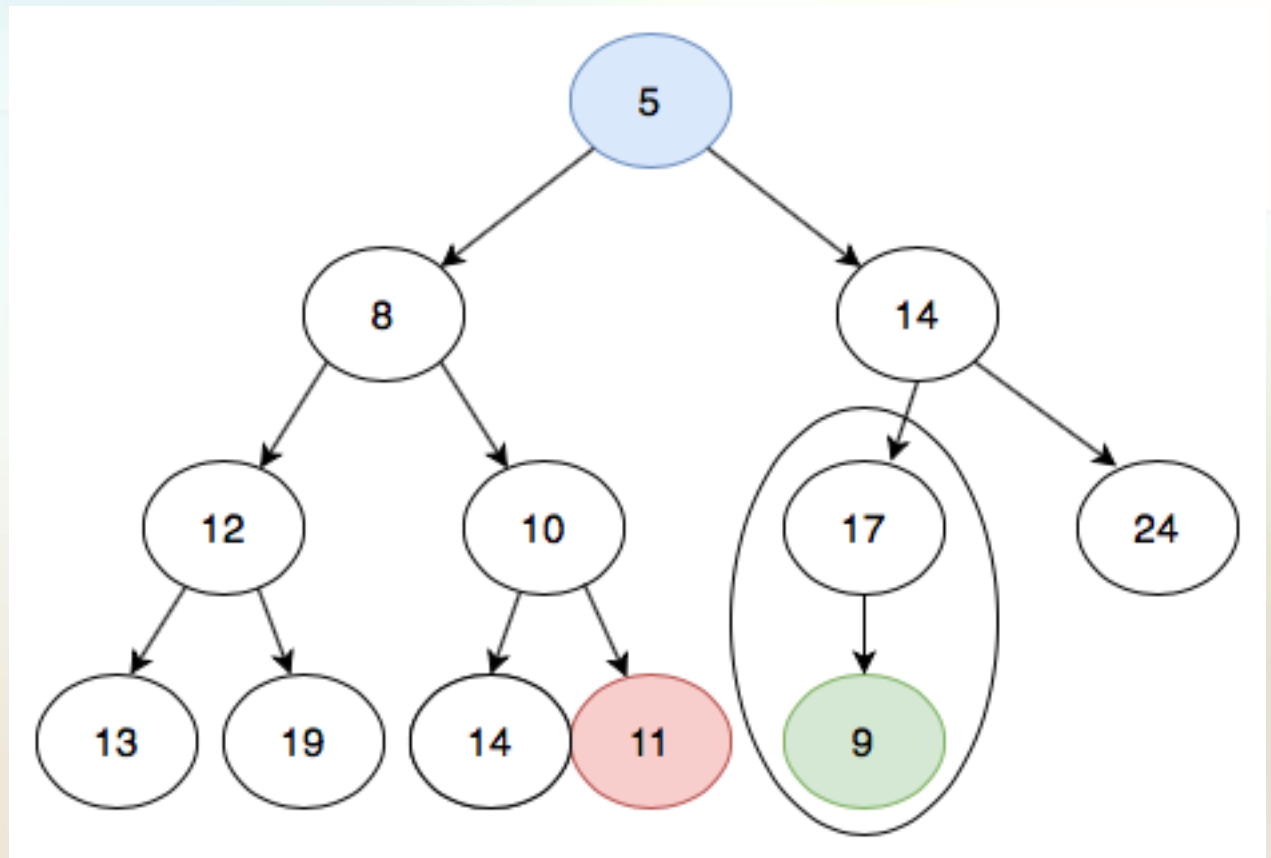
- Dodajemo novi element na heap

`heap.add(9, 'e')`



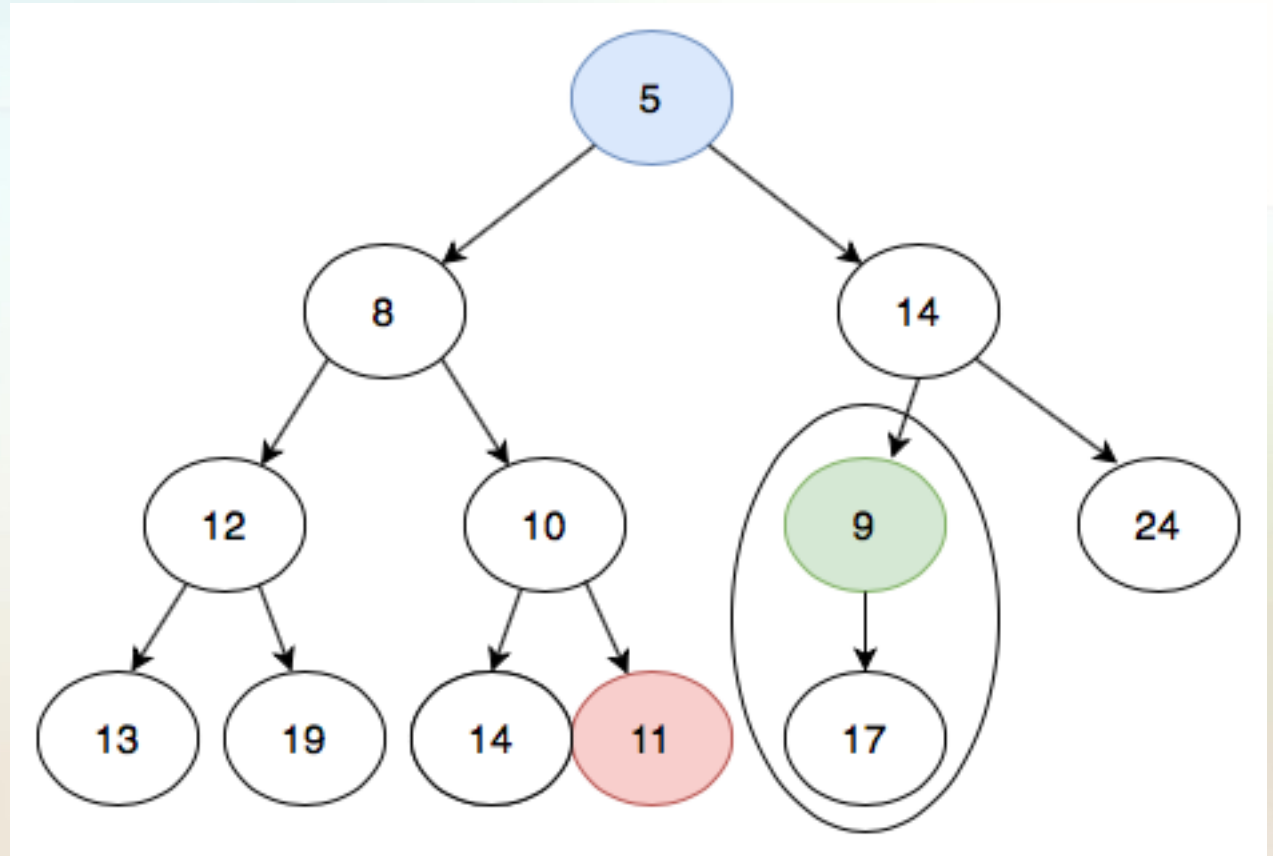
Primer Heap

- Novi element se dodaje na kraj heap-a (iza poslednjeg elementa)
- Započinjemo postupak uspostavljanja ispravnog redosleda pomeranjem novog elementa nagore kroz heap – upheap
- Poredimo novi element sa roditeljem - ukoliko je vrednost ključa roditelja veća, zamenimo im mesta



Primer Heap

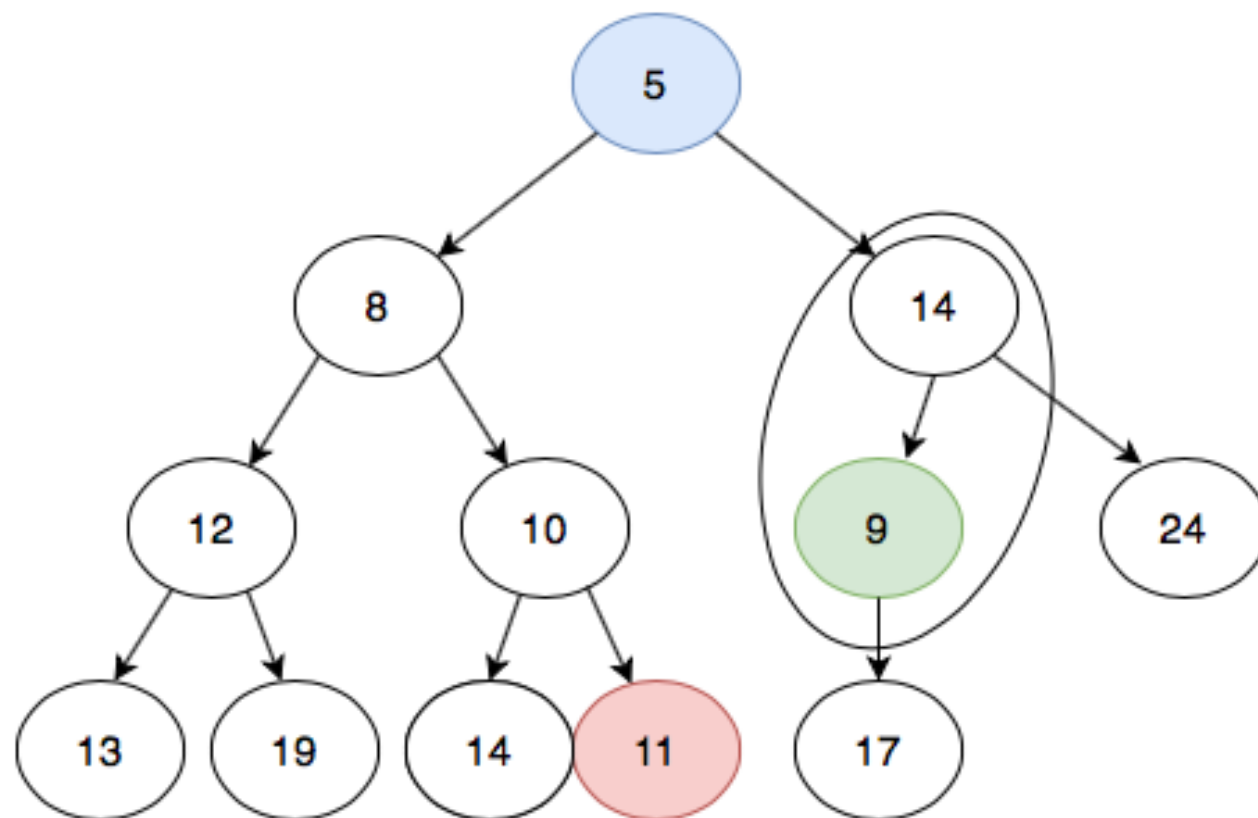
- Elementi su zamenili mesta



- *Ispravka: Poslednji element je 17, a ne označeni 11.*

Primer Heap

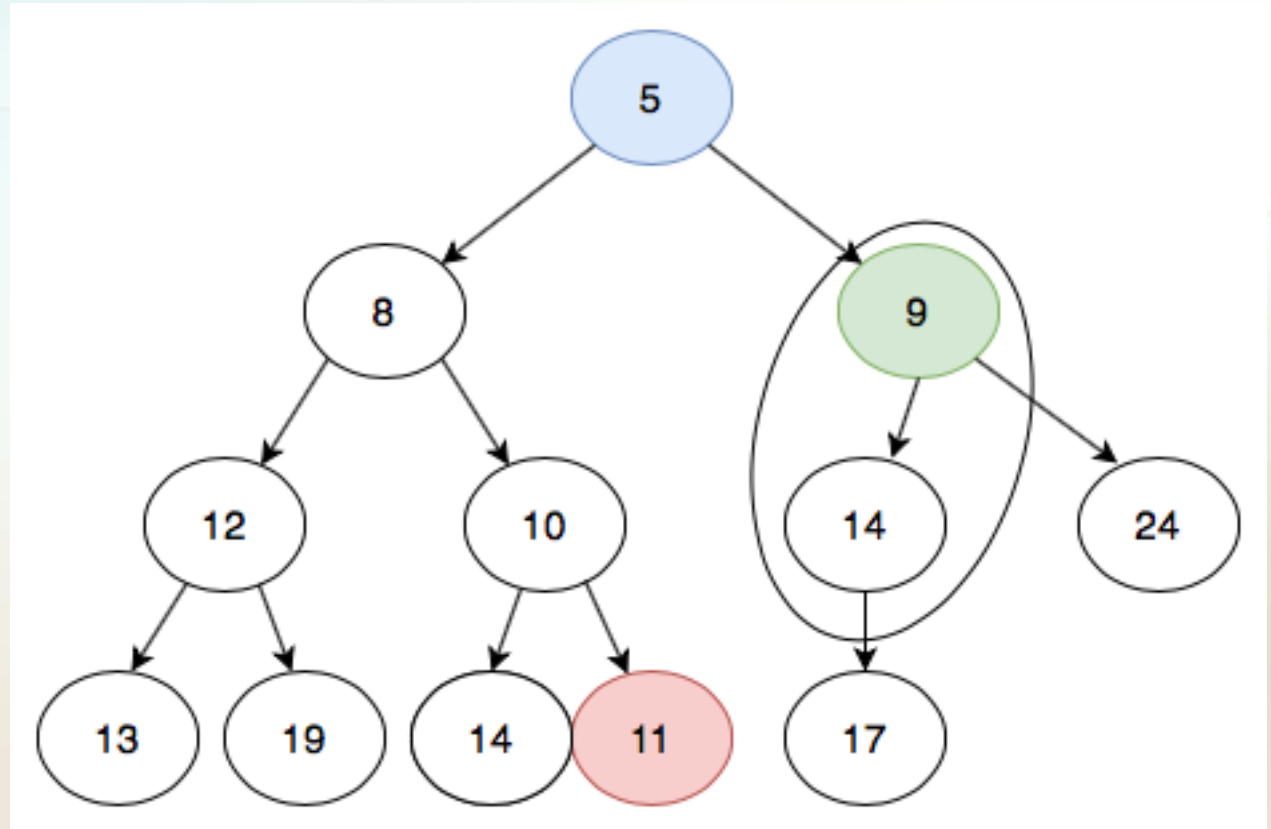
- Posle zamene, poredimo ključ novog elementa sa ključem sada novog roditelja
- Uočavamo da je potrebna zamena



- *Ispravka: Poslednji element je 17, a ne označeni 11.*

Primer Heap

- Posle zamene, poredimo ključ novog elementa sa ključem, sada novog, roditelja
- Uočavamo da je potrebna zamena

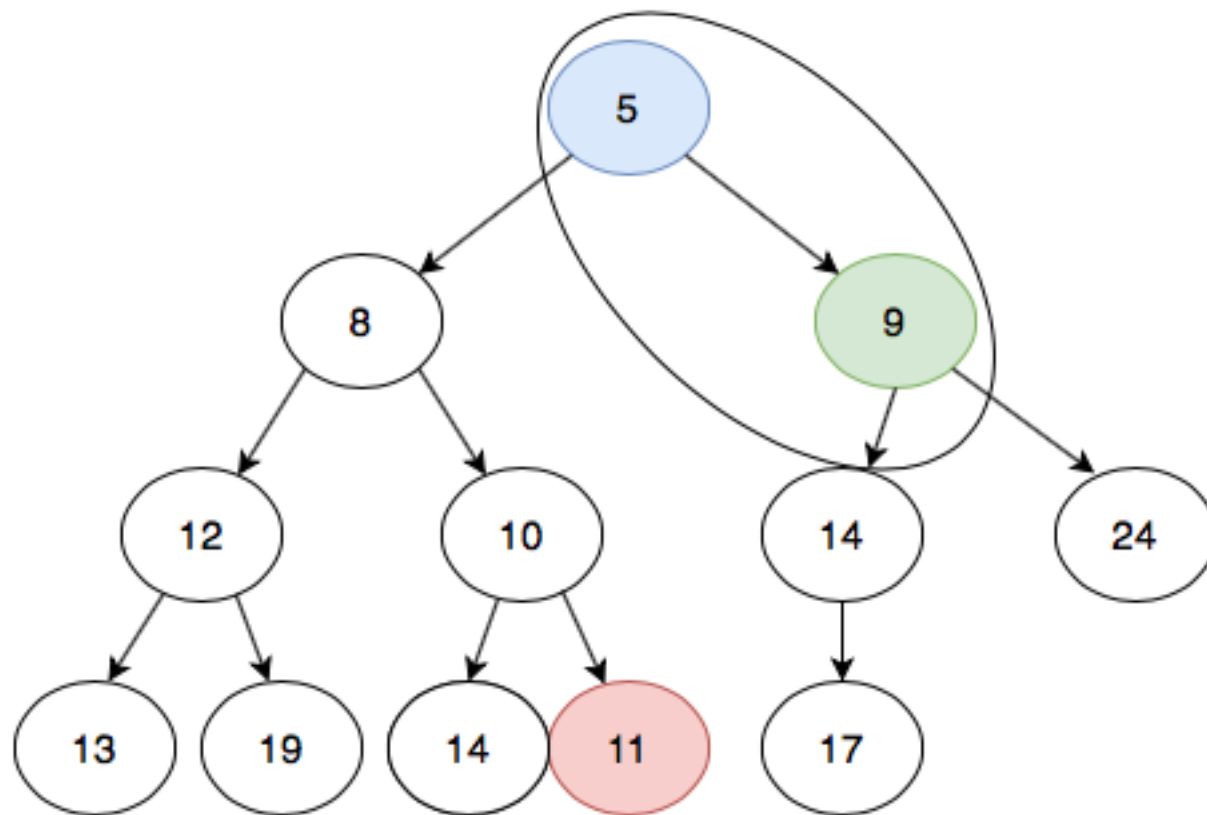


- *Ispravka: Poslednji element je 17, a ne označeni 11.*

Primer Heap

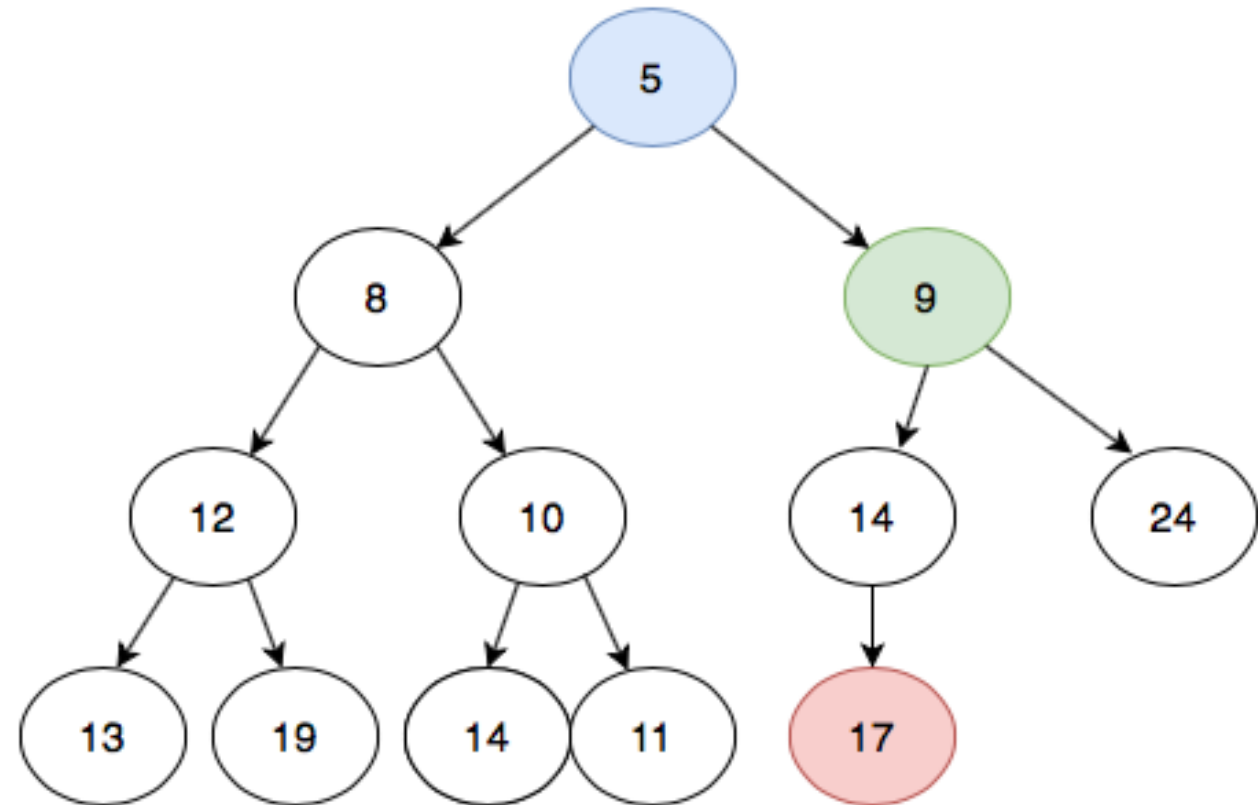
- Posle zamene, poredimo ključ novog elementa sa ključem, sada novog, roditelja, u ovom slučaju, korenskog elementa
- Uočavamo da zamena nije potrebna – postupak je završen

- *Ispravka: Poslednji element je 17, a ne označeni 11.*



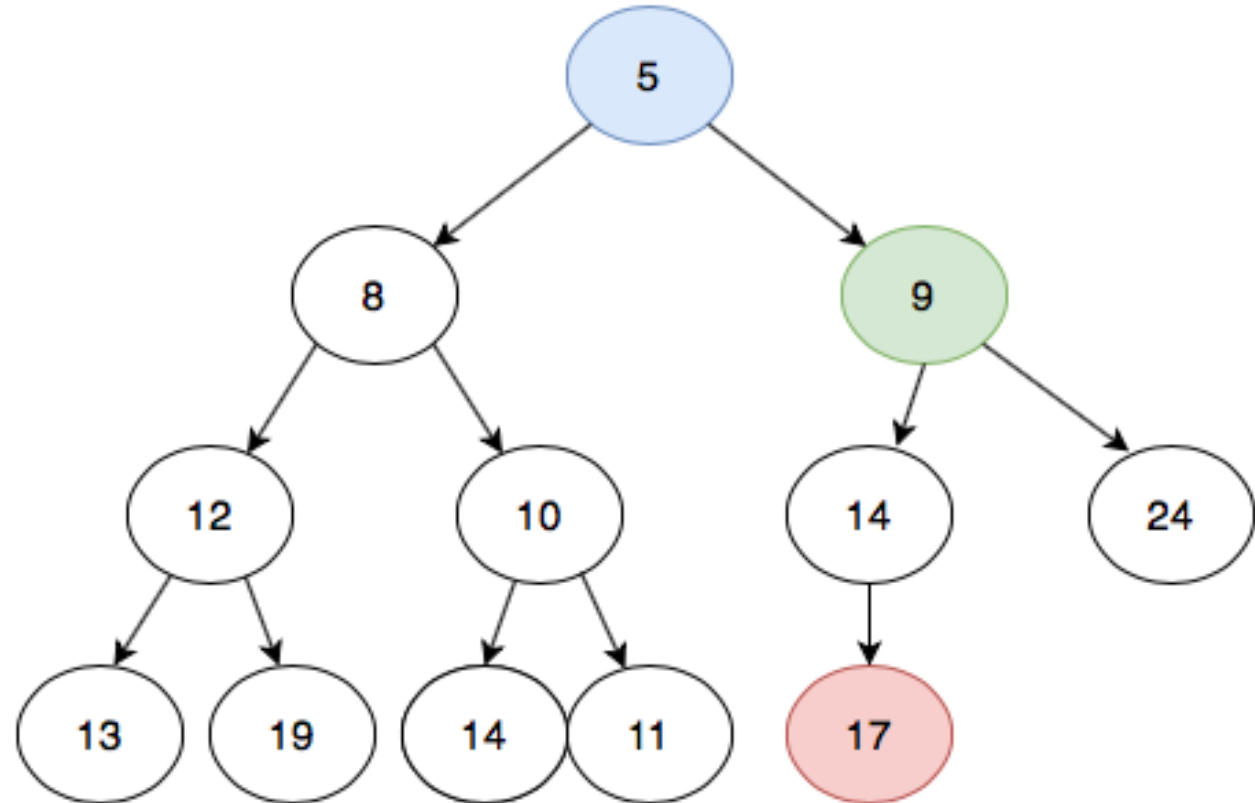
Primer Heap

- Novo stanje heap-a.



Primer Heap

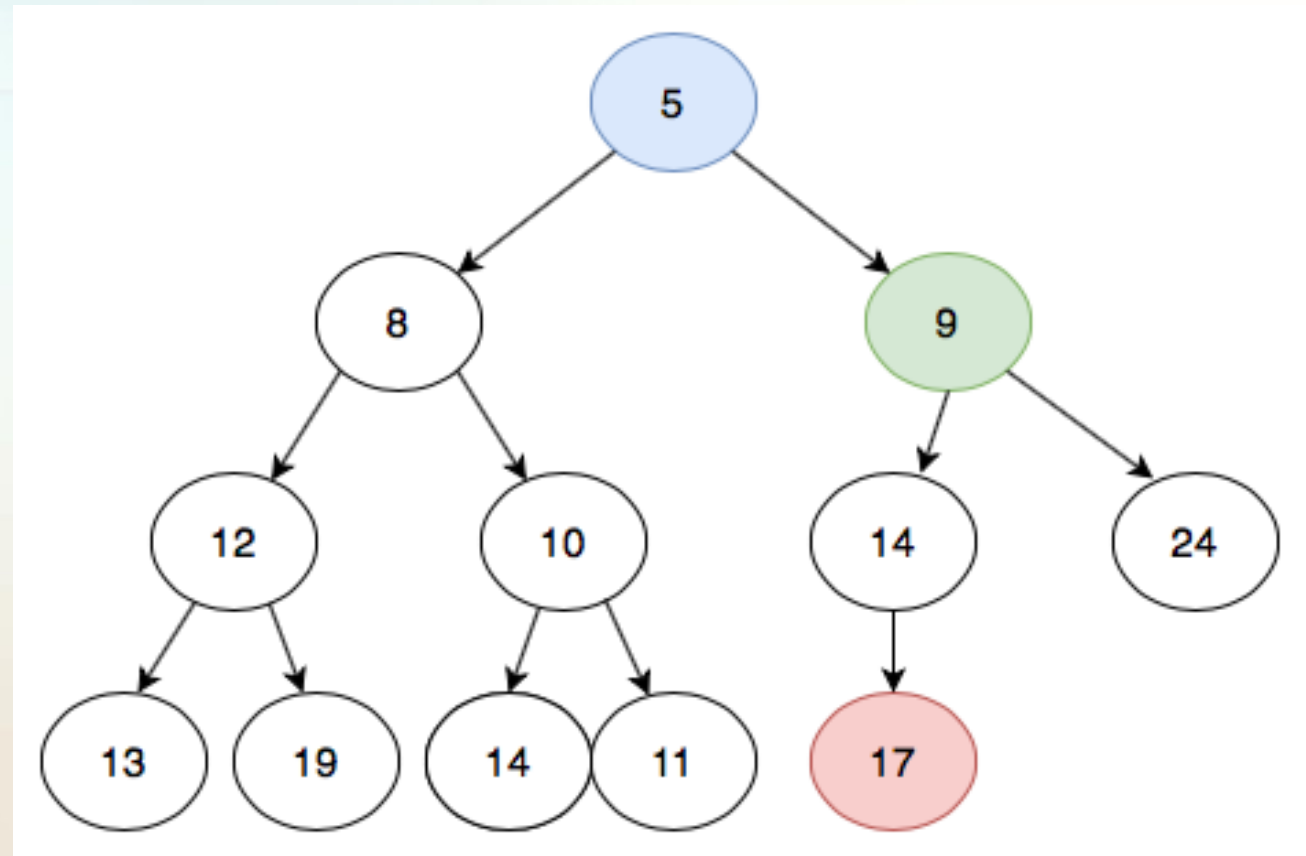
- Novo stanje heap-a.
- Ako bismo pozvali:
`heap.min()`
- Rezultat bi bio element sa ključem 5.



Primer Heap

- Uklanjamo najmanji element heap-a

`heap.remove_min()`

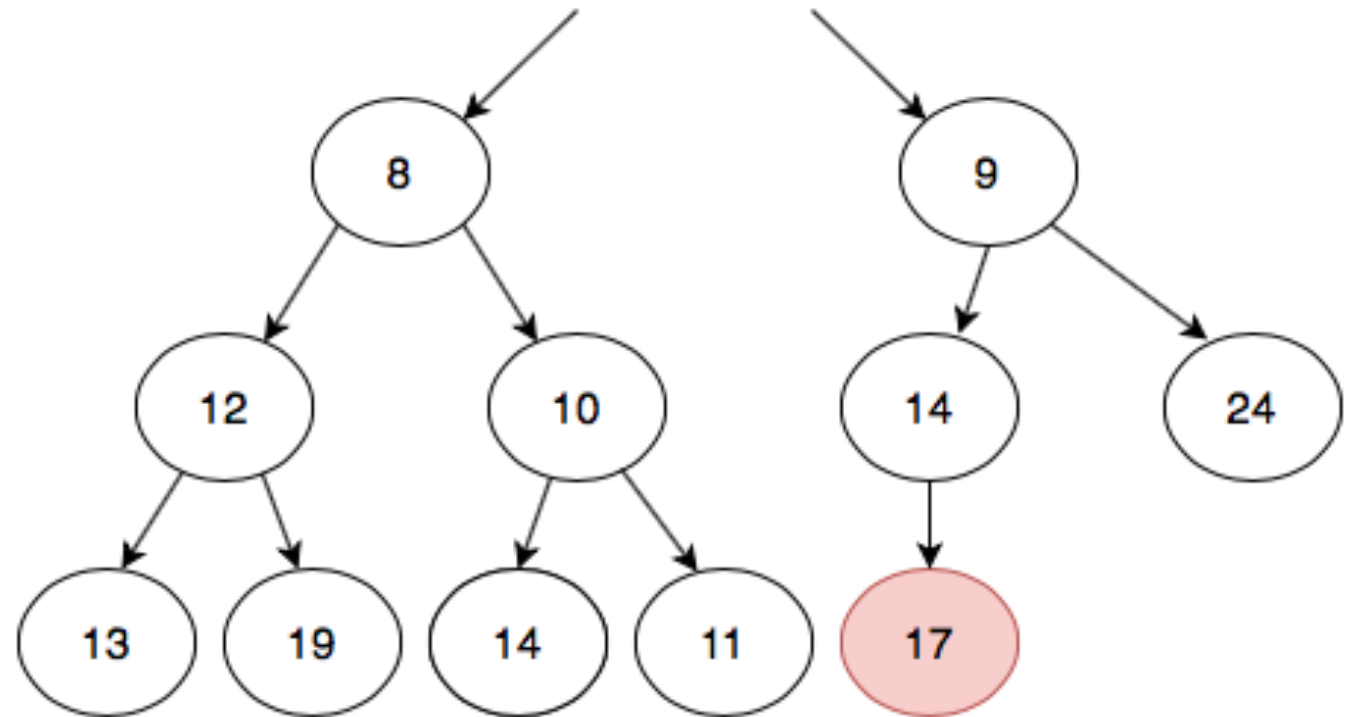


Primer Heap

- Uklanjamo najmanji element heap-a

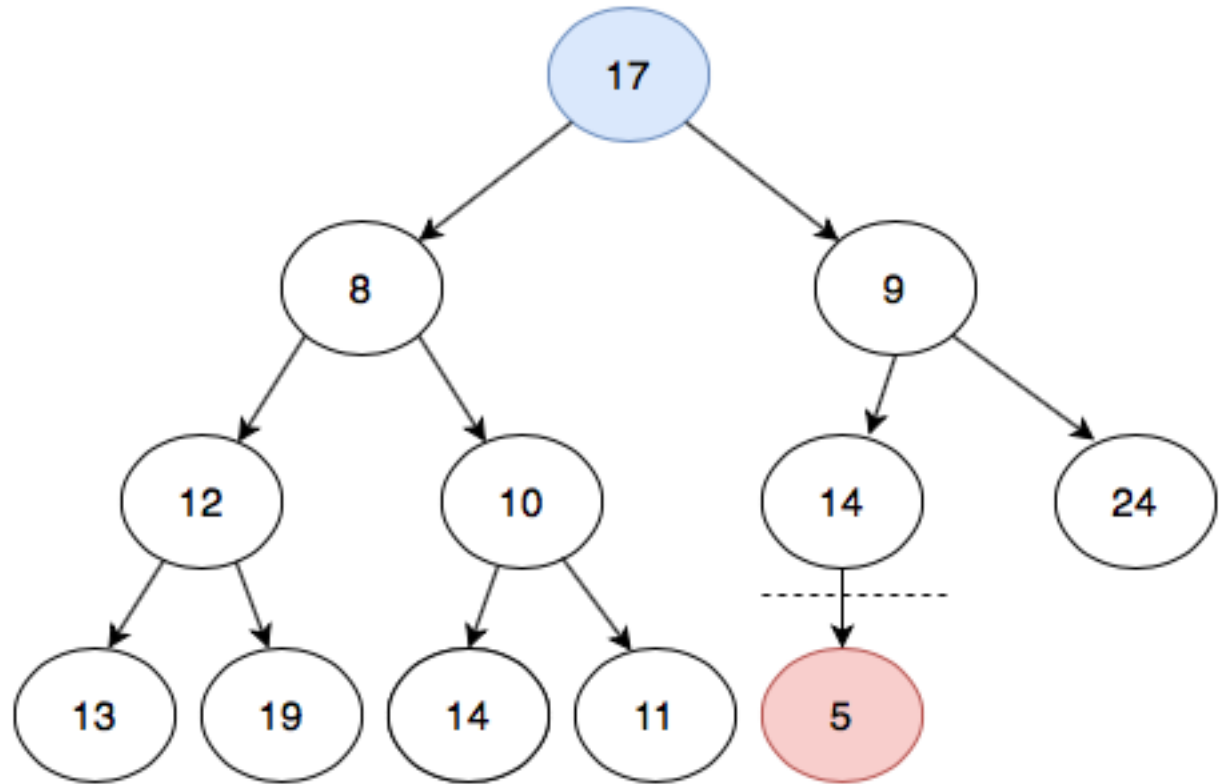
`heap.remove_min()`

- Ukoliko bi se element samo uklonio iz heap-a nastao bi problem spajanja dva novoformirana heap-a.
- Zbog toga, pristup je sledeći:



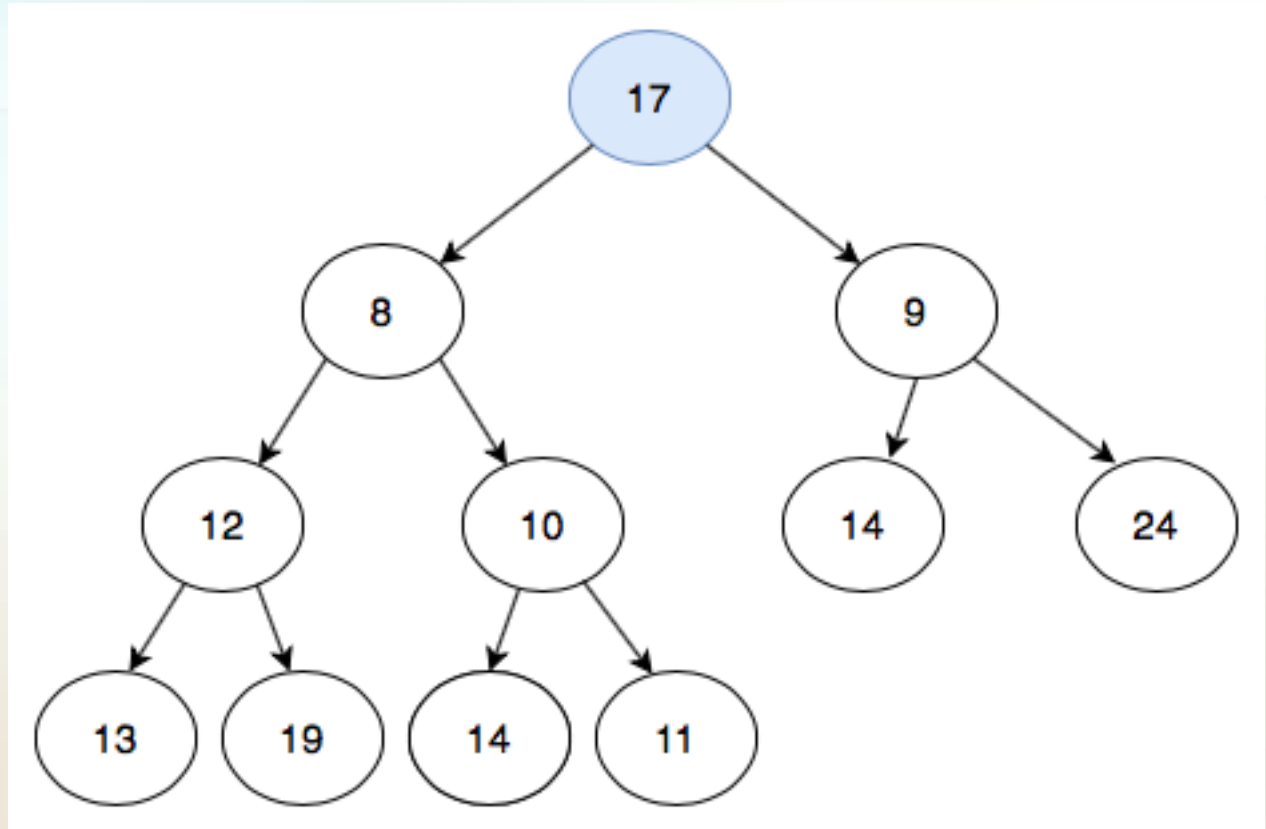
Primer Heap

- Prebacujemo najmanji element na poziciju sa koje je uklanjanje trivijalno – to je pozicija poslednjeg elementa
- Zamenjujemo poslednji i korenski element
- Stari koren uklanjamo
- Ostaje nam da uspostavimo ponovo ispravan redosled u heap-u u postupku koji zovemo downheap.



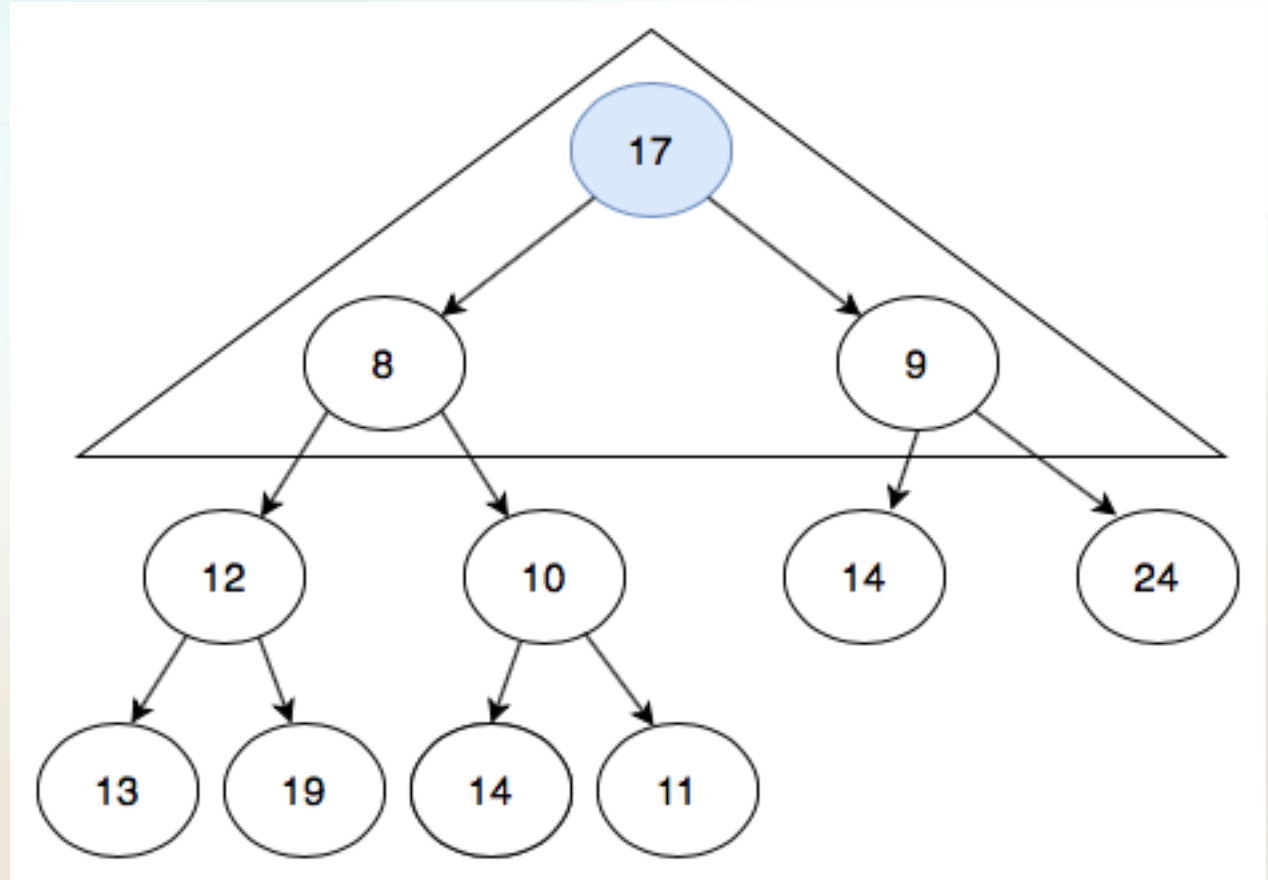
Primer Heap

- Stanje posle uklanjanja



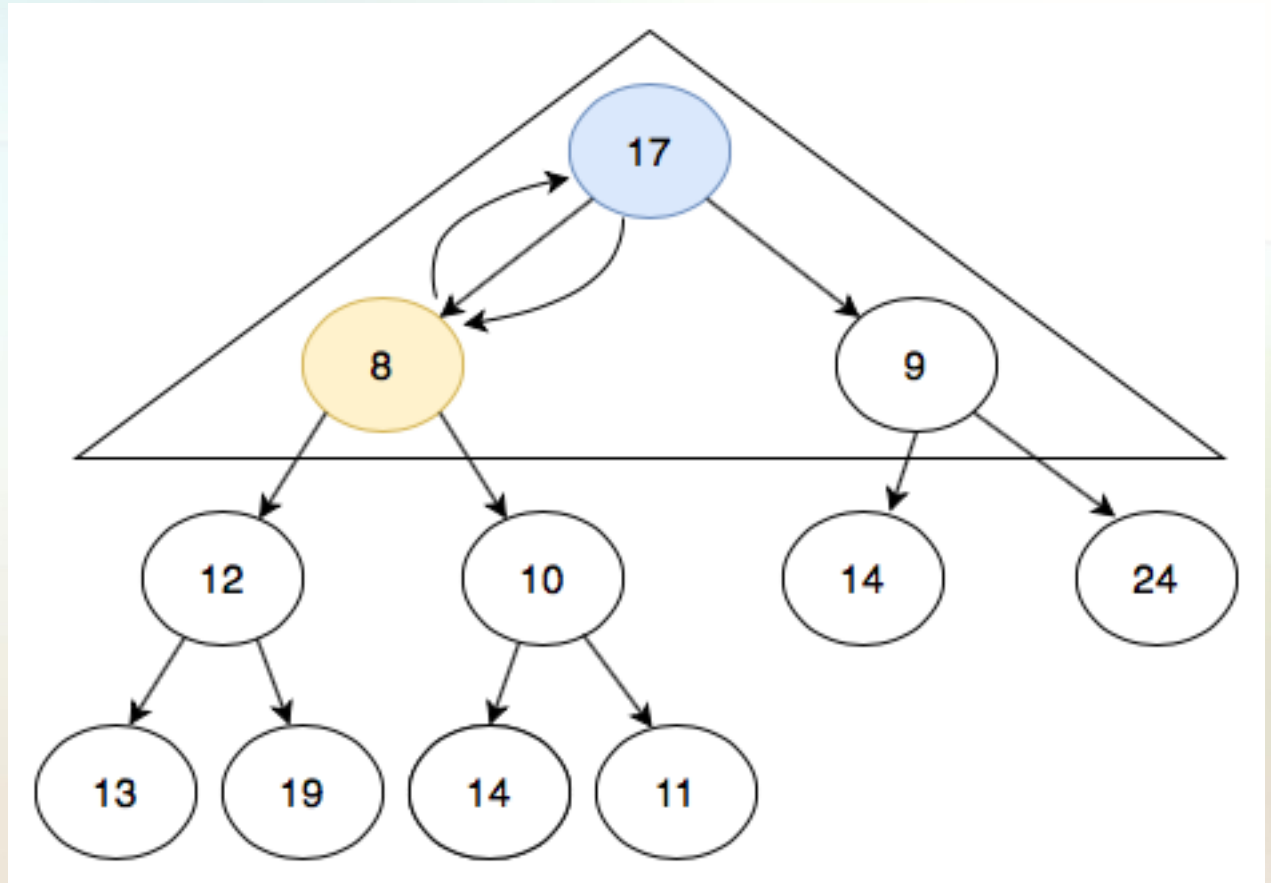
Primer Heap

- Poredimo element sa ključem 17 sa ključevima dece. Ukoliko neko od dece ima manji ključ od trenutnog elementa, zamenjujemo element sa tim čvorom deteta. Ponavljamo postupa rekurzivno.



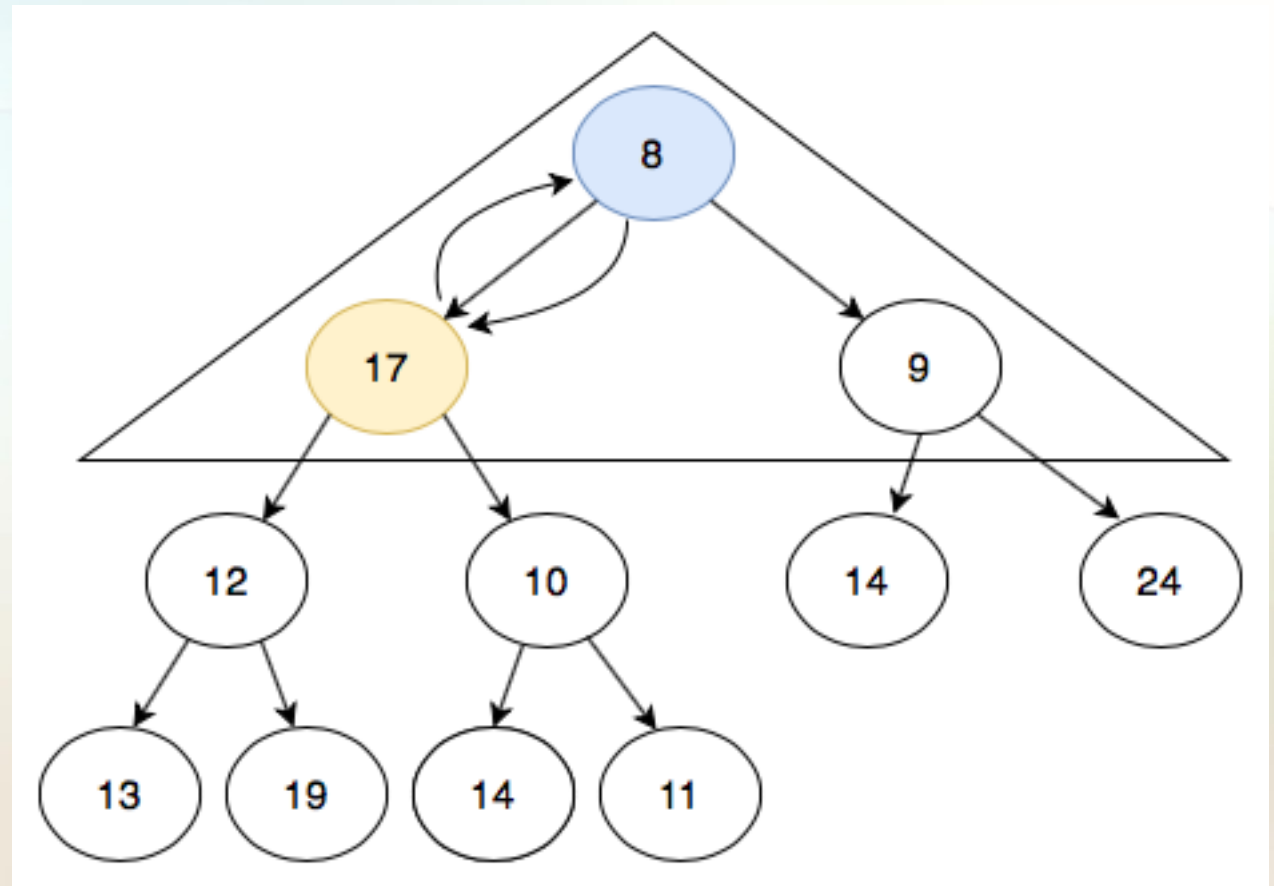
Primer Heap

- Poredimo trenutni element (sa ključem 17) sa ključevima dece. Pronalazimo koji od tri čvora ima najmanji ključ. Ukoliko je to roditeljski čvor, postupak se završava. Ukoliko je neki od čvorova dece, zamenjujemo trenutni element sa tim čvorom. Ponavljamo postupak rekurzivno.
- Žutom bojom označavamo čvor manjeg deteta.



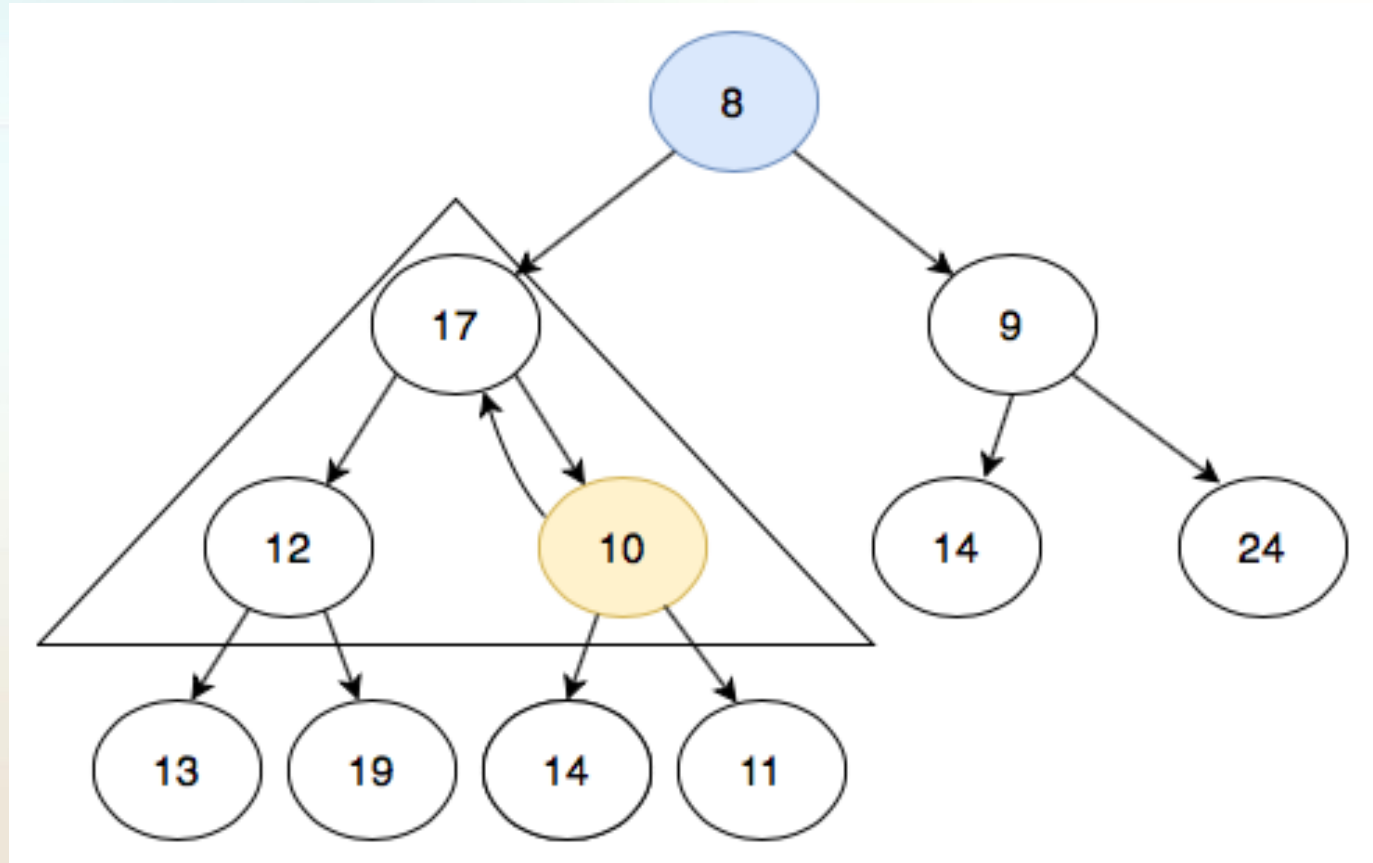
Primer Heap

- Zamena čvorova.



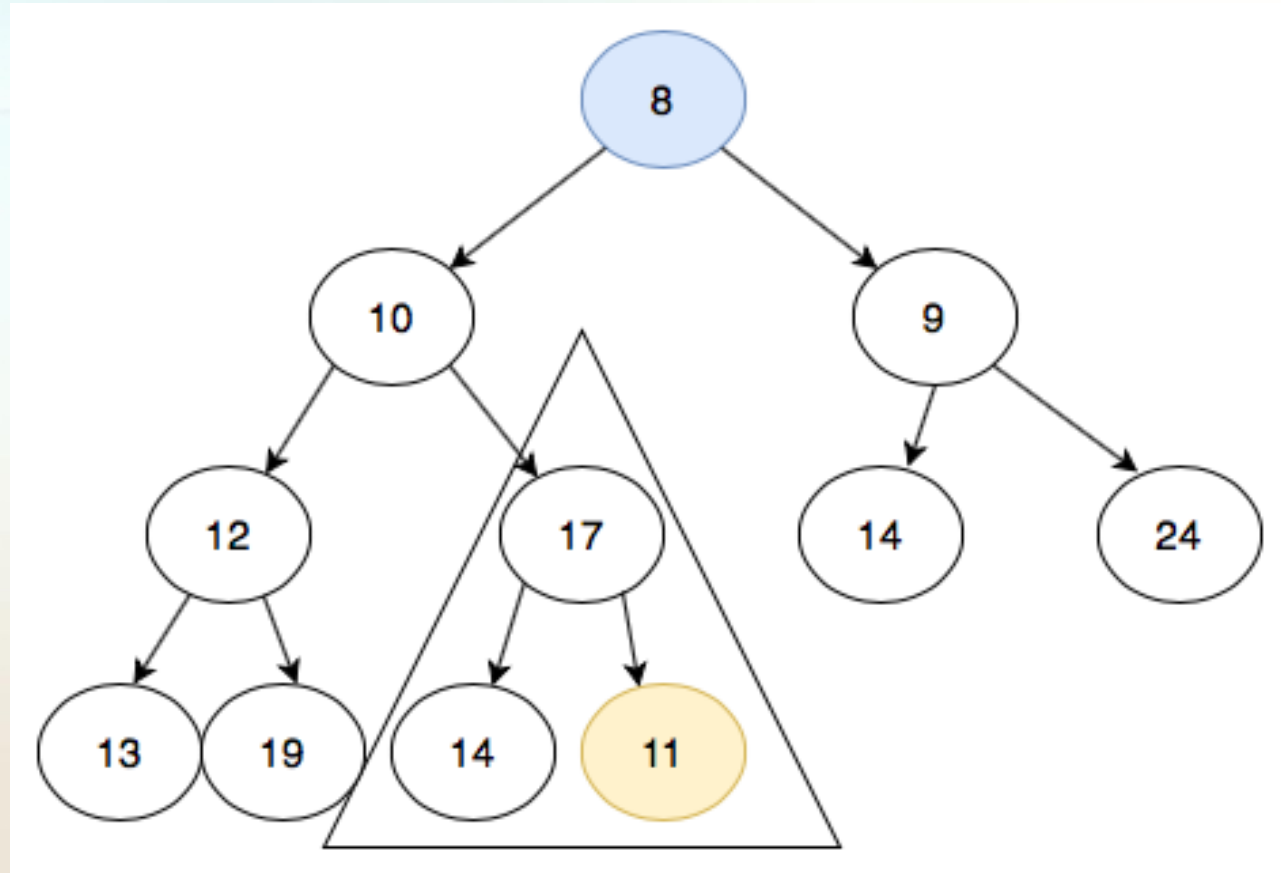
Primer Heap

- Pratimo čvor 17 i ponavljamo postupak.



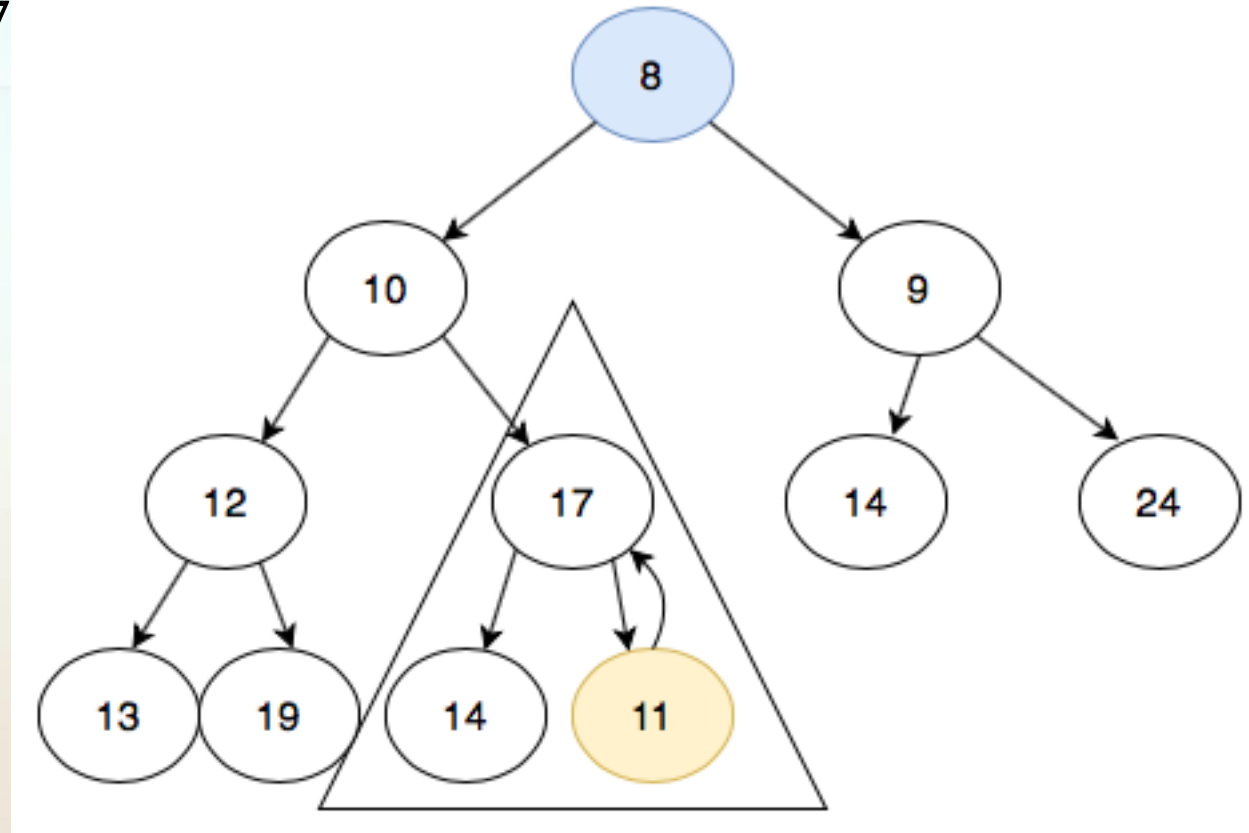
Primer Heap

- Pratimo čvor 17 i ponavljamo postupak.



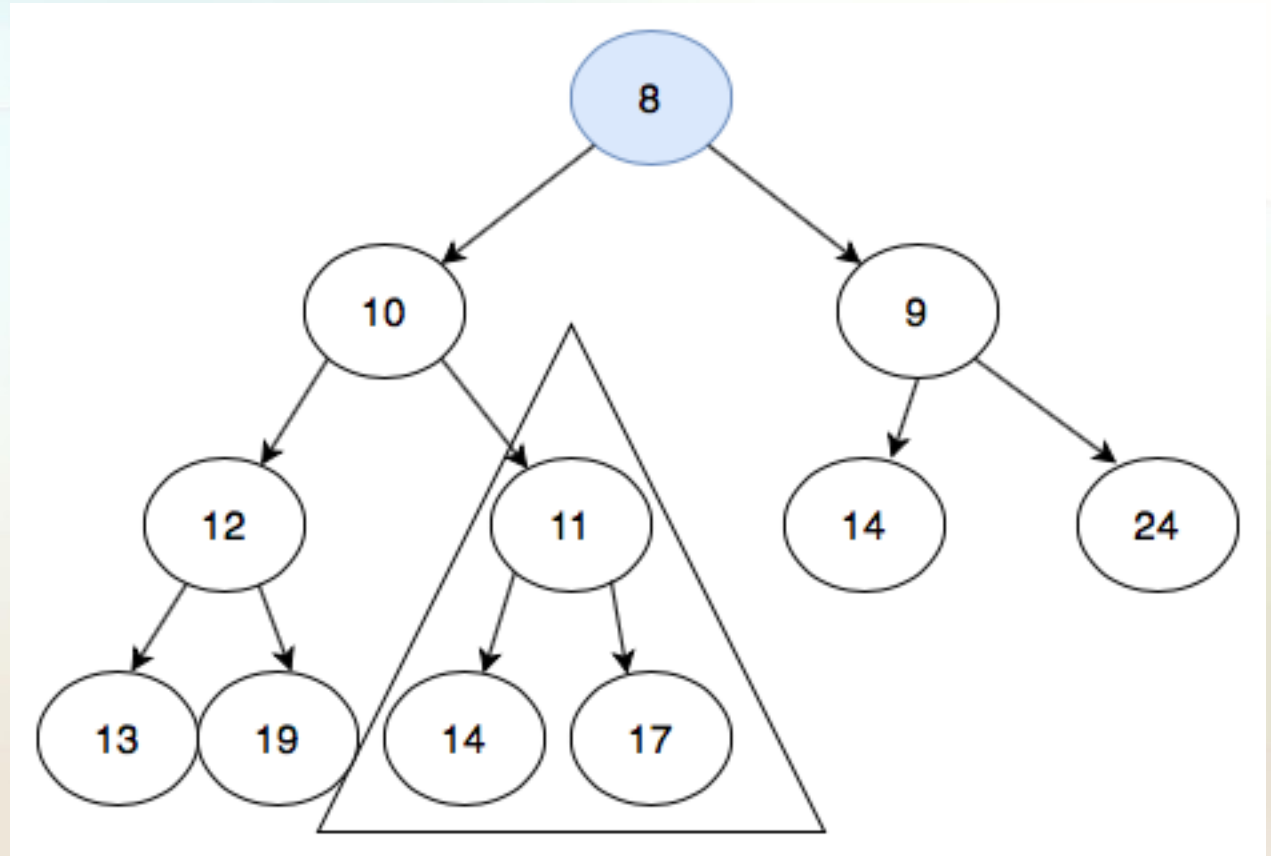
Primer Heap

- Zamenjujemo čvor sa ključem 17 sa čvorom deteta sa manjim ključem



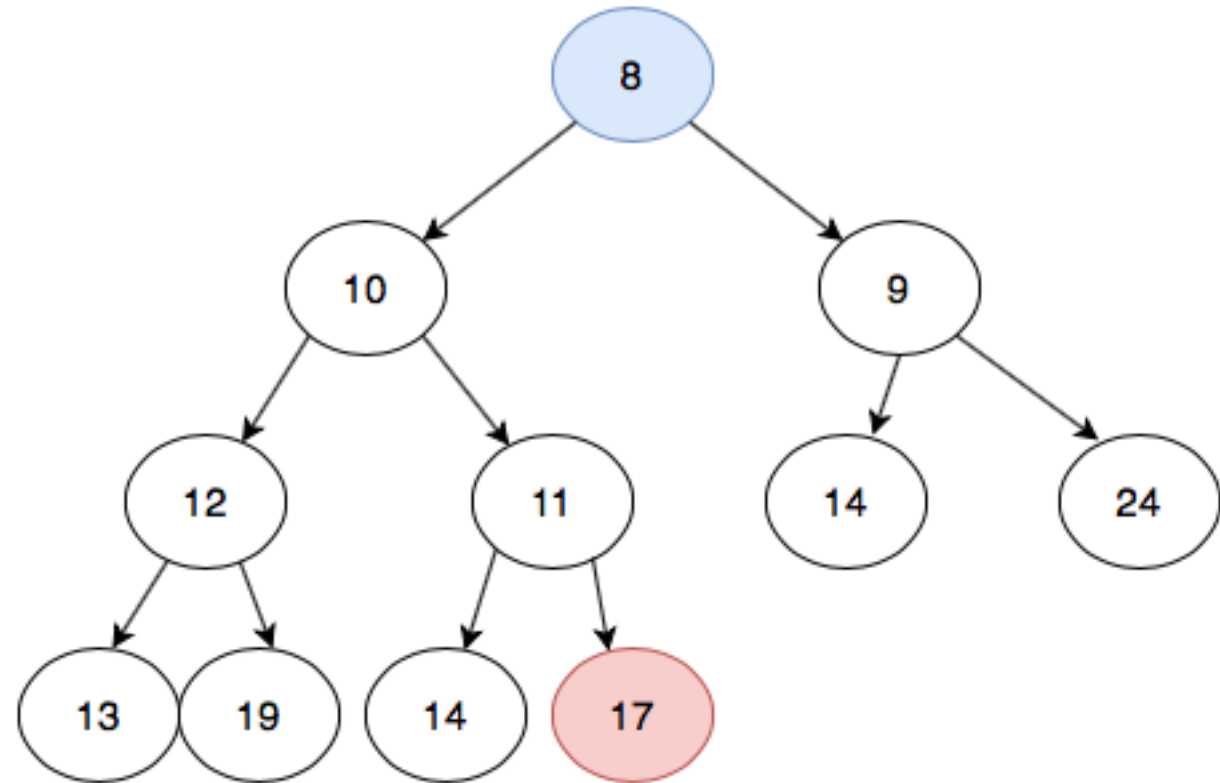
Primer Heap

- Čvor 17 je postao lisni (nema potomke) pa se postupak prekida



Primer Heap

- Stanje stabla posle završetka downheap postupka.

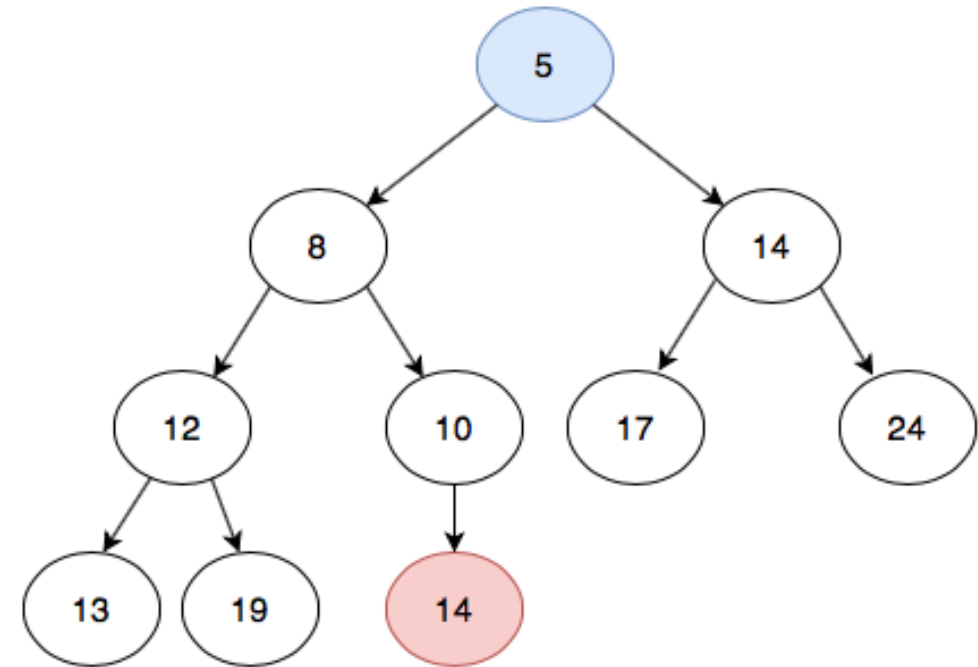


Implementacija heap-a pomoću niza

- Kao i svako drugo binarno stablo, i heap se može implementirati uz pomoć niza
- Heap sa slike se može predstaviti pomoću niza (crvenom bojom su označeni indeksi)

0 1 2 3 4 5 6 7 8 9
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14]

- Na koji način se utvrđuje pozicija roditelja čvora? A dece?



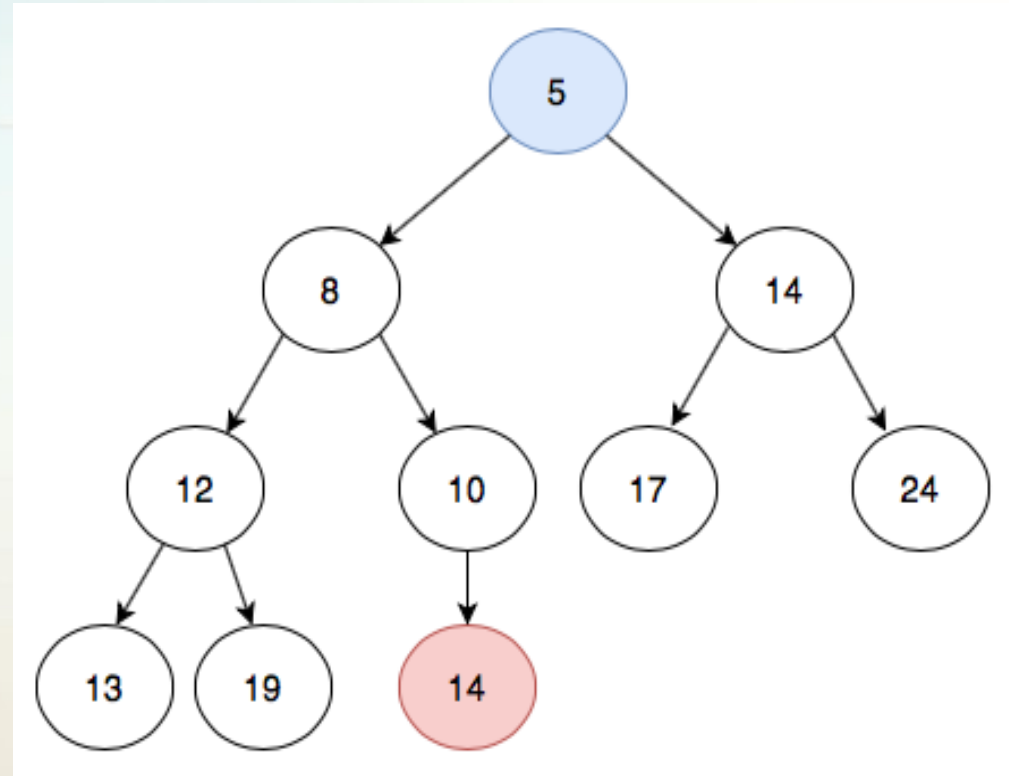
Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14]

Npr. Deca čvora sa ključem 8 su čvorovi sa ključevima 12 i 10. Čvor 8 se nalazi na indeksu 1 a deca na indeksima 3 i 4.

Deca čvora sa ključem 12 su čvorovi sa ključevima 13 i 19. Čvor 12 se nalazi na indeksu 3 a deca na indeksima 7 i 8.

Važi formula:

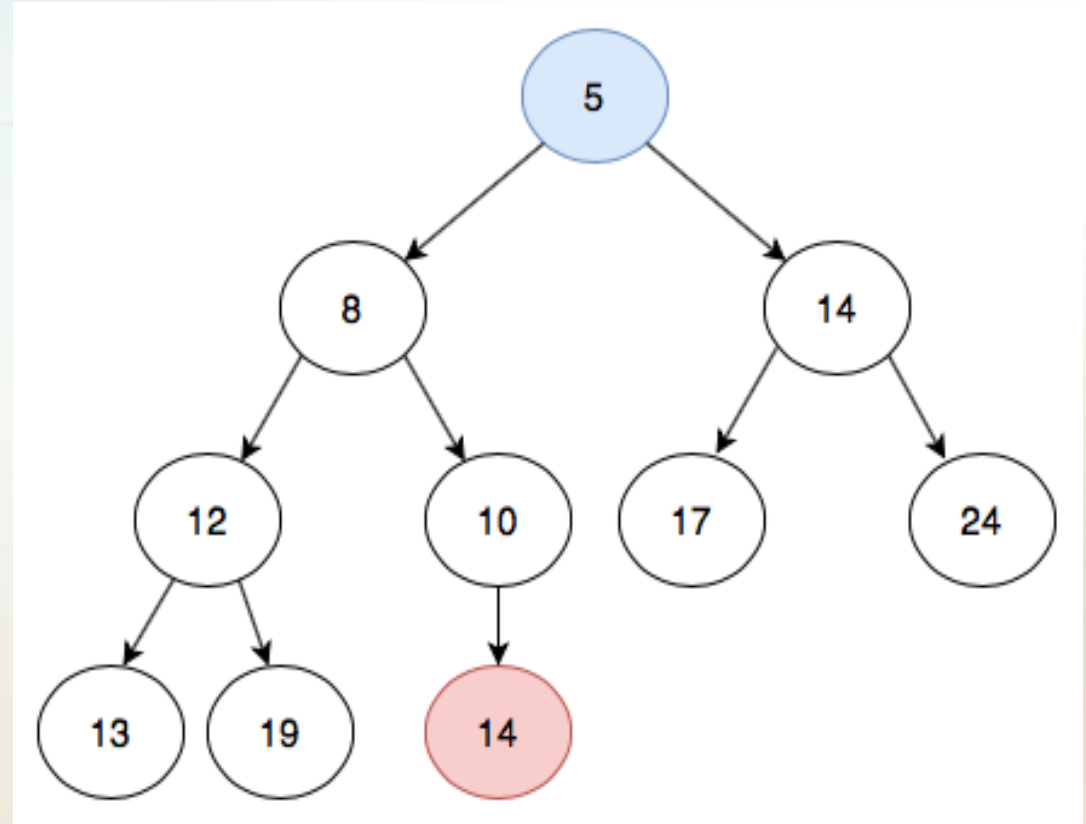
$$\text{left_child_index} = \text{parent_index} * 2 + 1$$
$$\text{right_child_index} = \text{parent_index} * 2 + 2$$


Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14]

Po uspostavljenoj formuli, može se utvrditi i broj dece koje neki čvor ima.

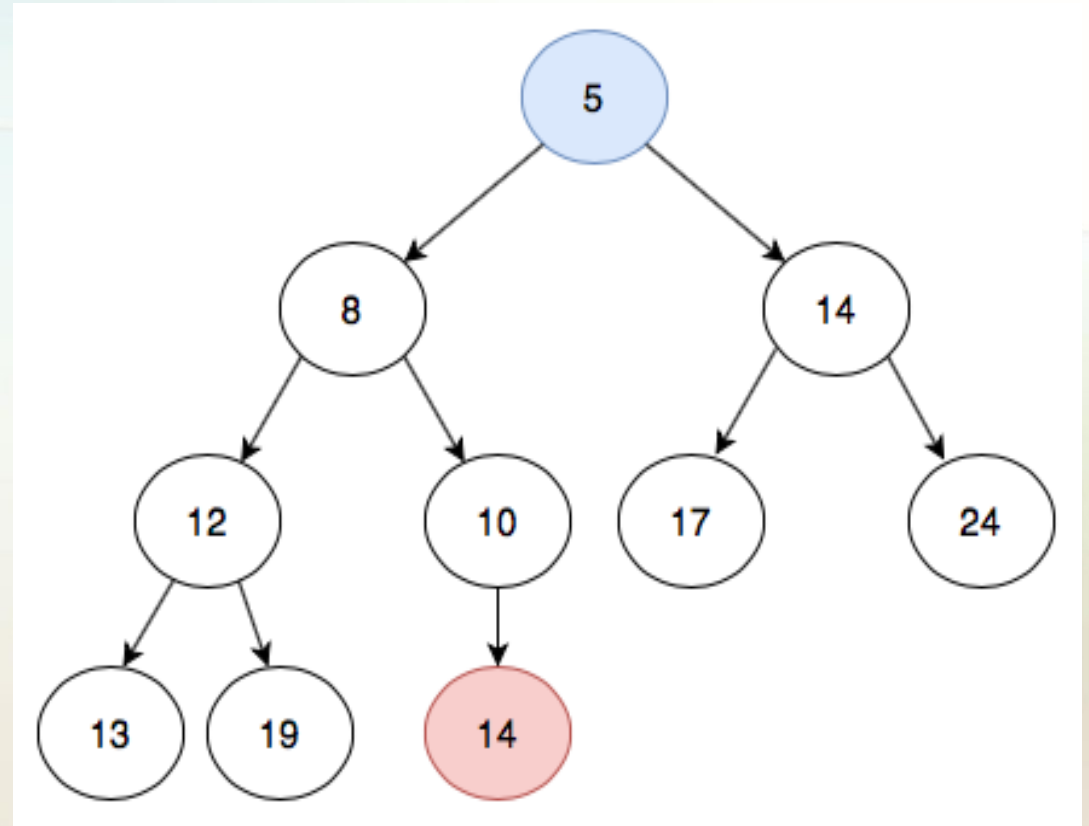
Npr. Za čvor sa ključem 10 na indeksu 4 očekujemo decu na indeksima $4*2+1 = 9$ i $4*2+2 = 10$. S obzirom da element na indeksu 10 ne postoji, zaključujemo da čvor sa ključem 10 ima jedno dete.



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14]

- Formula za računanje indeksa roditeljskog čvora proizilazi iz prethodne.

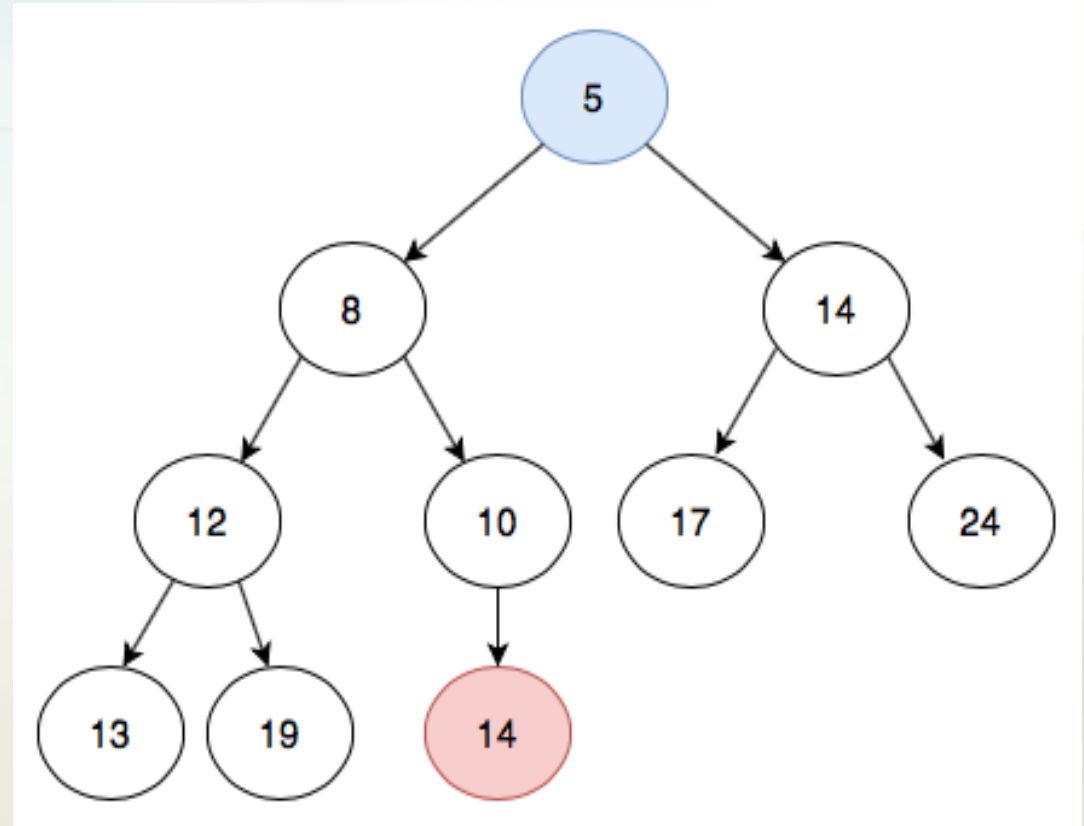


`parent_index = (child_index-1) // 2`

Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14]

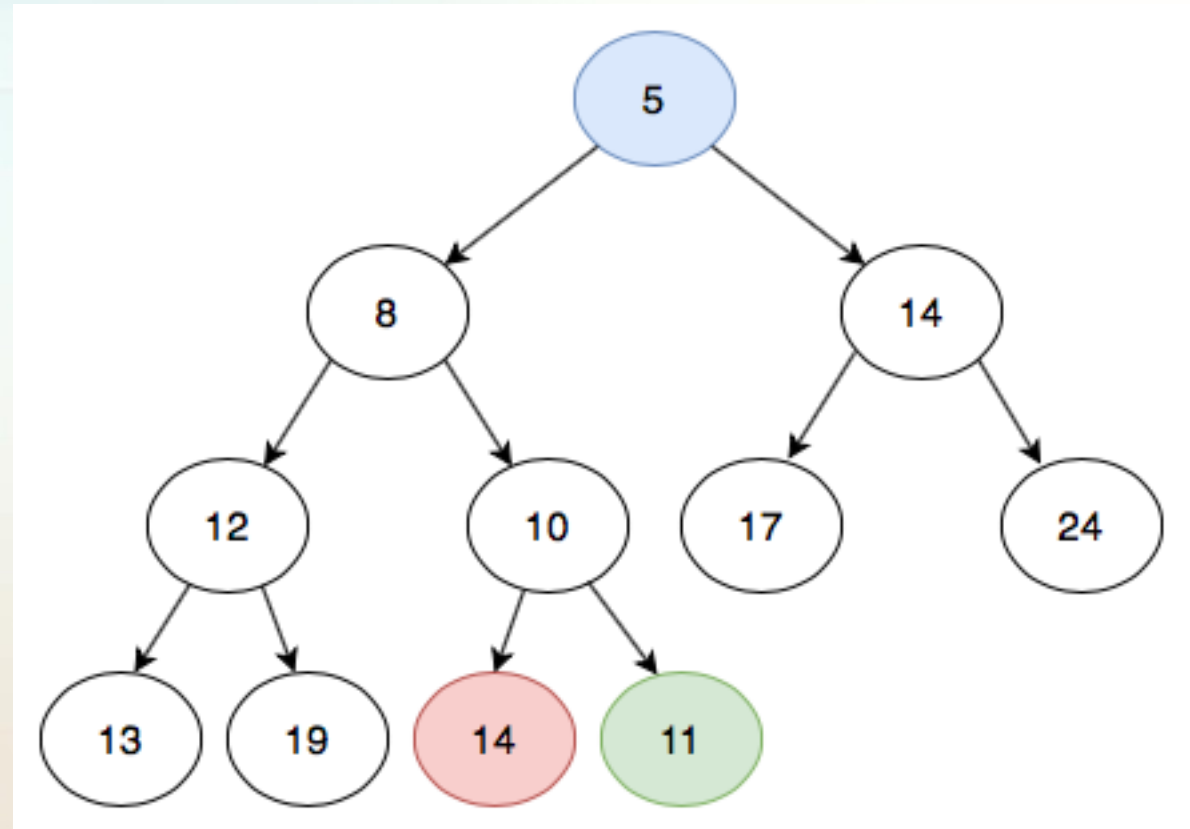
- Kako implementirati upheap i downheap postupke nad nizom?



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14, 11]

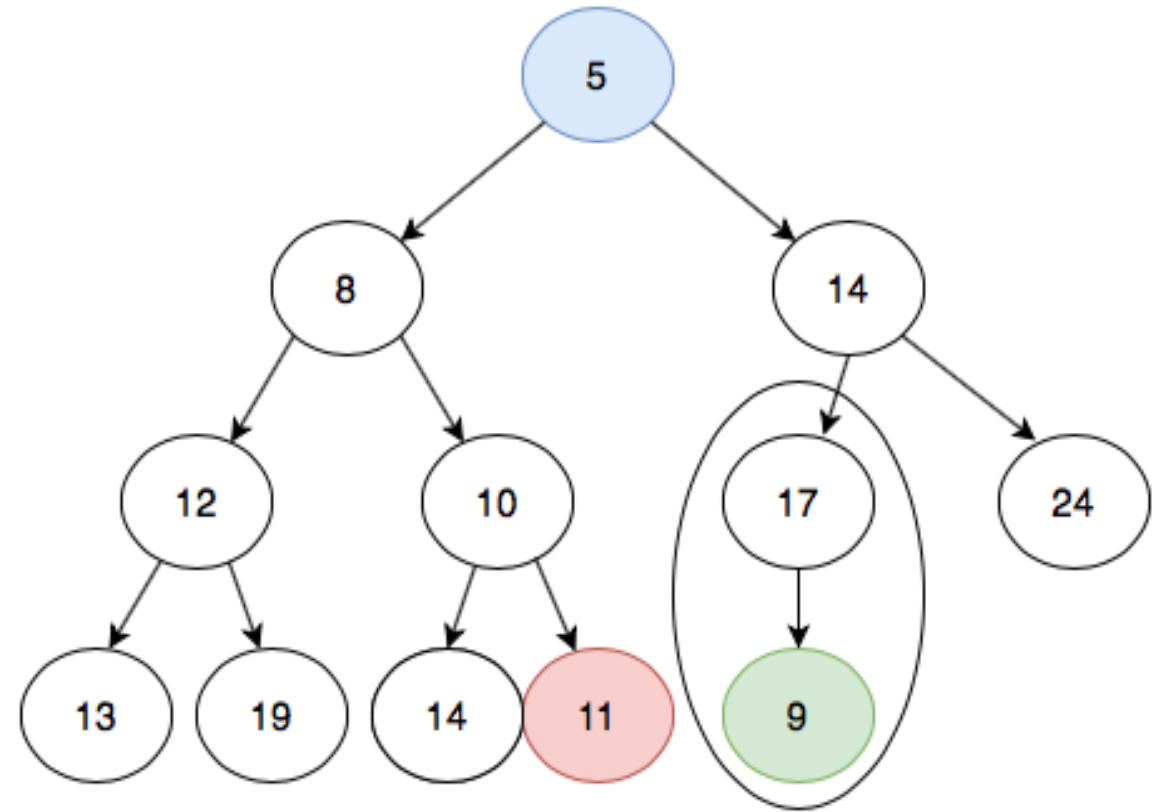
- Dodajemo 11. Dodaje se na kraj niza.



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14, 11, 9]

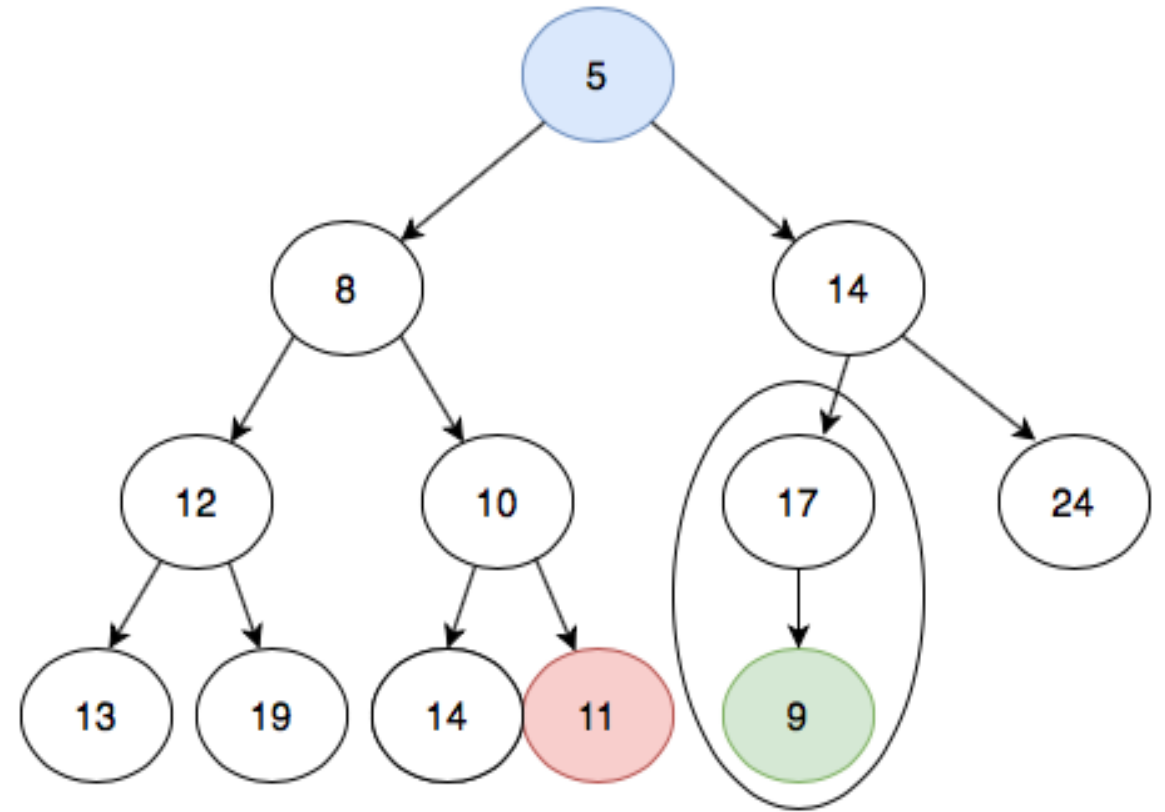
- Dodajemo 9. Dodaje se na kraj niza.
- Započinjemo upheap.
- Pronalazimo indeks roditelja i poredimo vrednosti ključeva.
- Zamenjujemo elemente u nizu.



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 14, 12, 10, 17, 24, 13, 19, 14, 11, 9]

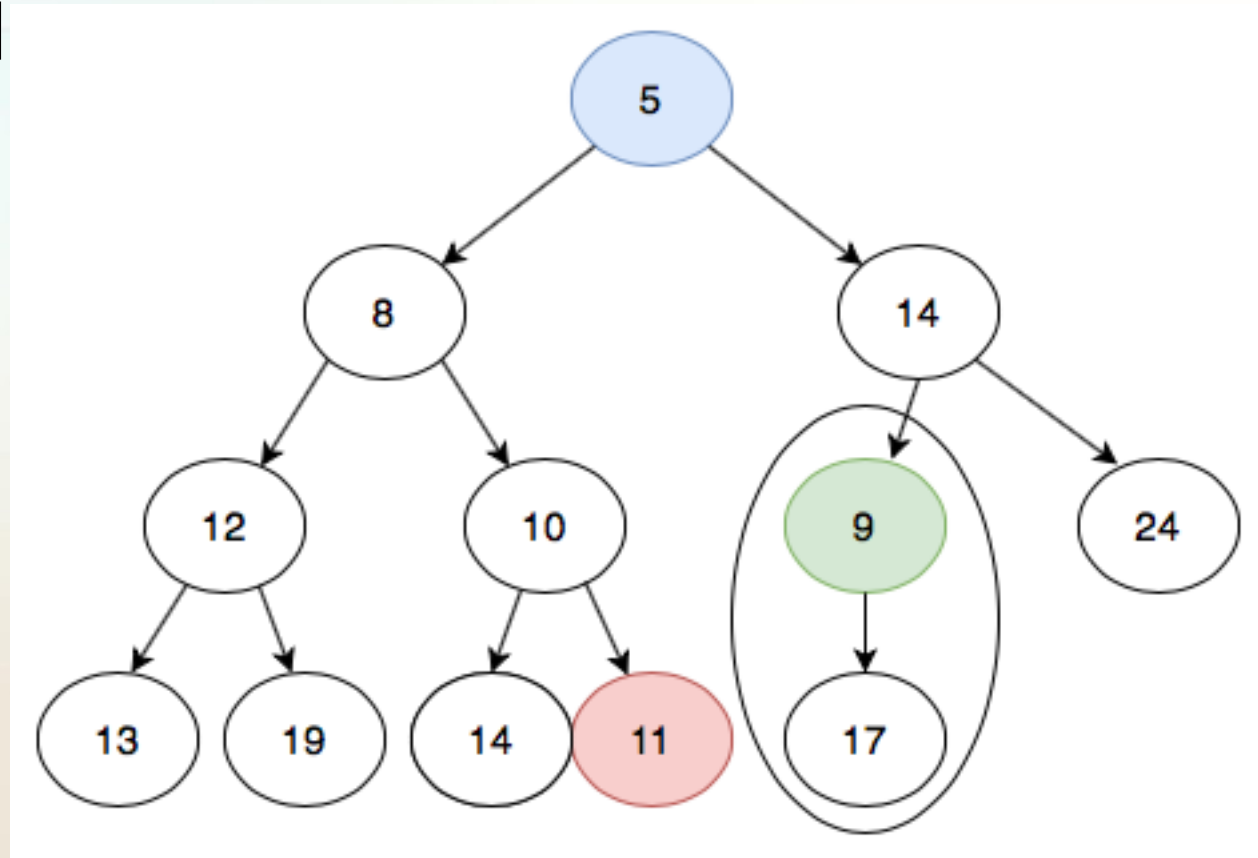
- Dodajemo 9. Dodaje se na kraj niza.
- Započinjemo upheap.
- Pronalazimo indeks roditelja i poredimo vrednosti ključeva.
- Zamenjujemo elemente u nizu.



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 14, 12, 10, 9, 24, 13, 19, 14, 11, 17]

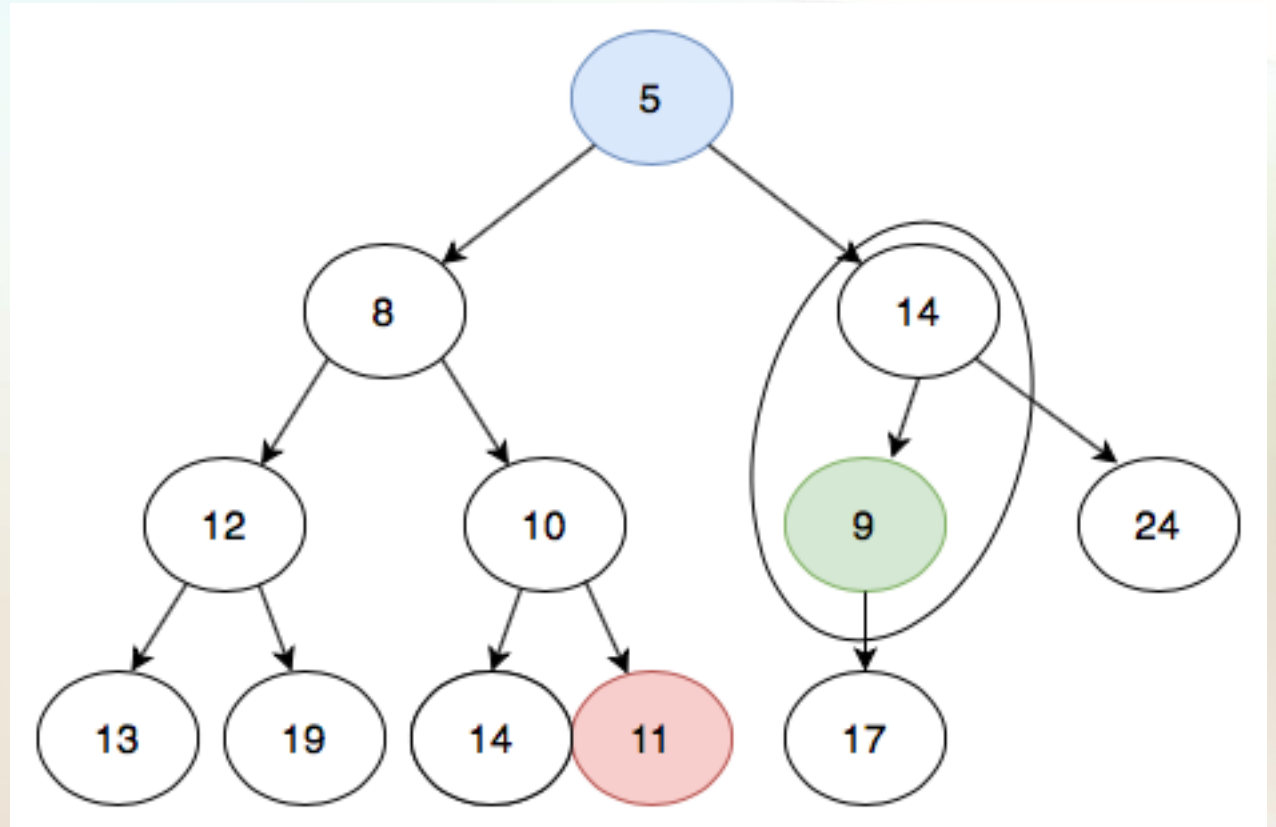
•



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 14, 12, 10, 9, 24, 13, 19, 14, 11, 17]

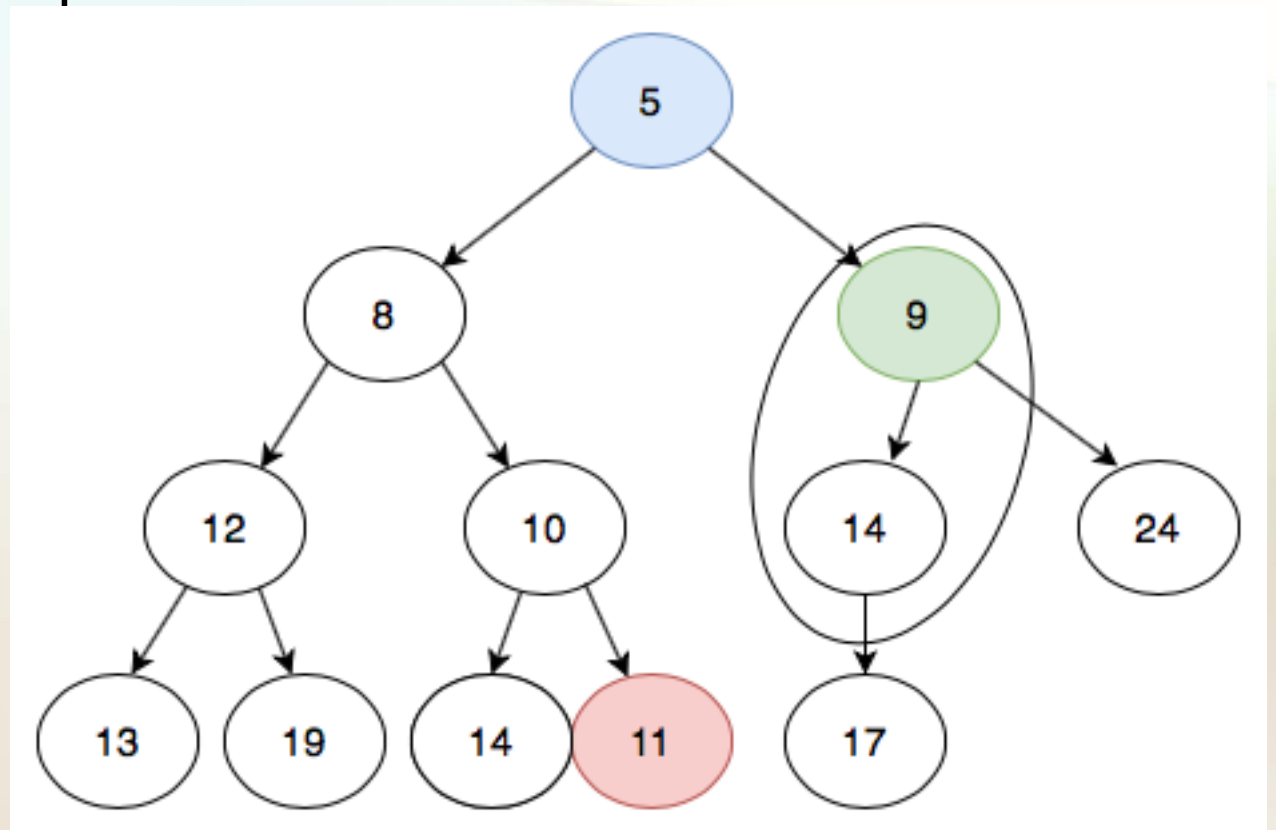
- Nastavljamo sa upheap postupkom



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11, 17]

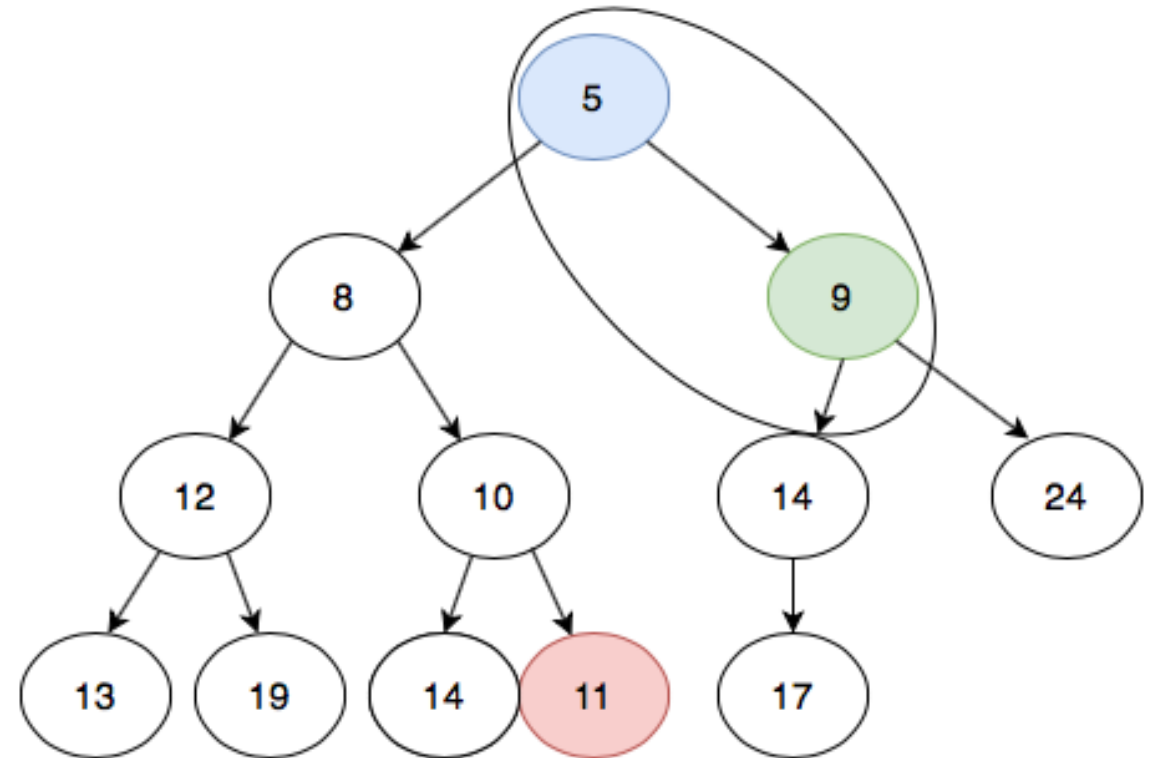
- Zamenjujemo elemente



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11, 17]

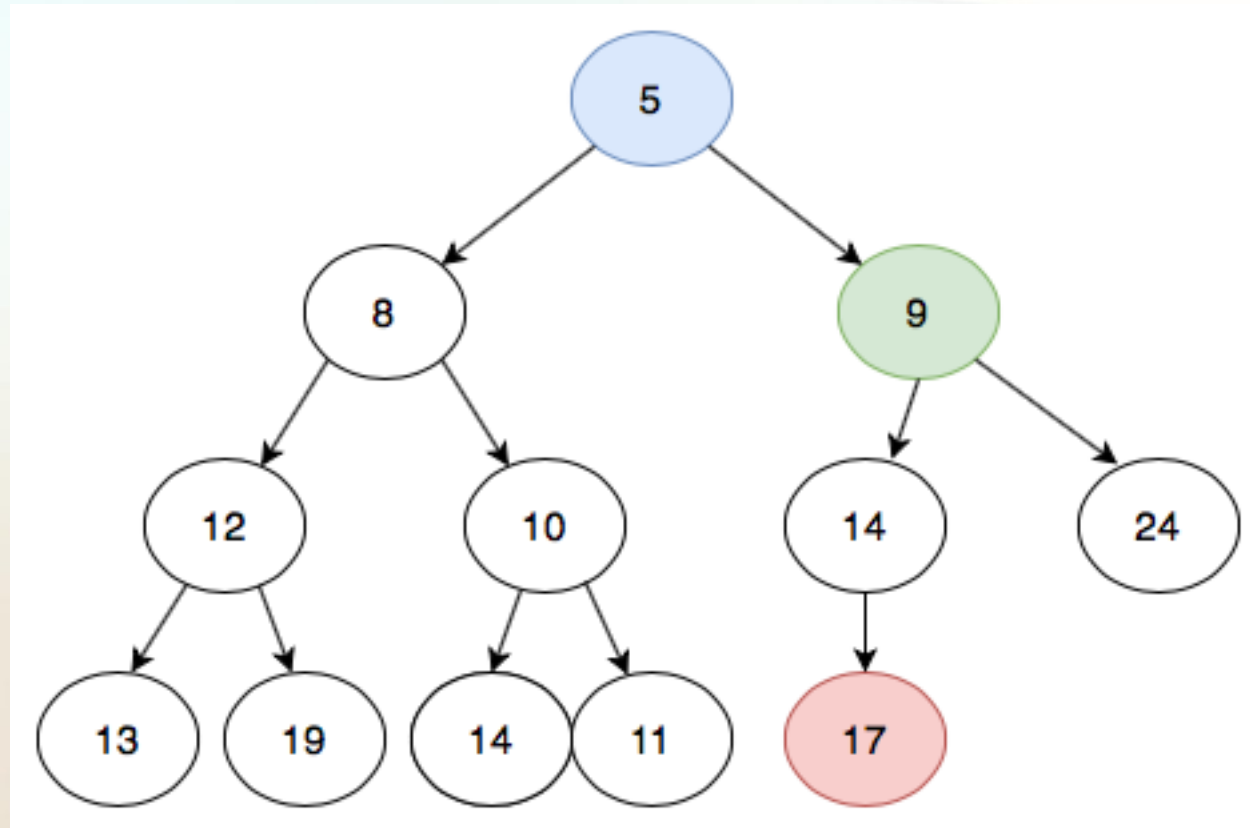
- Poredimo elemente
- Zamena nije potrebna - kraj postupka



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11, 17]

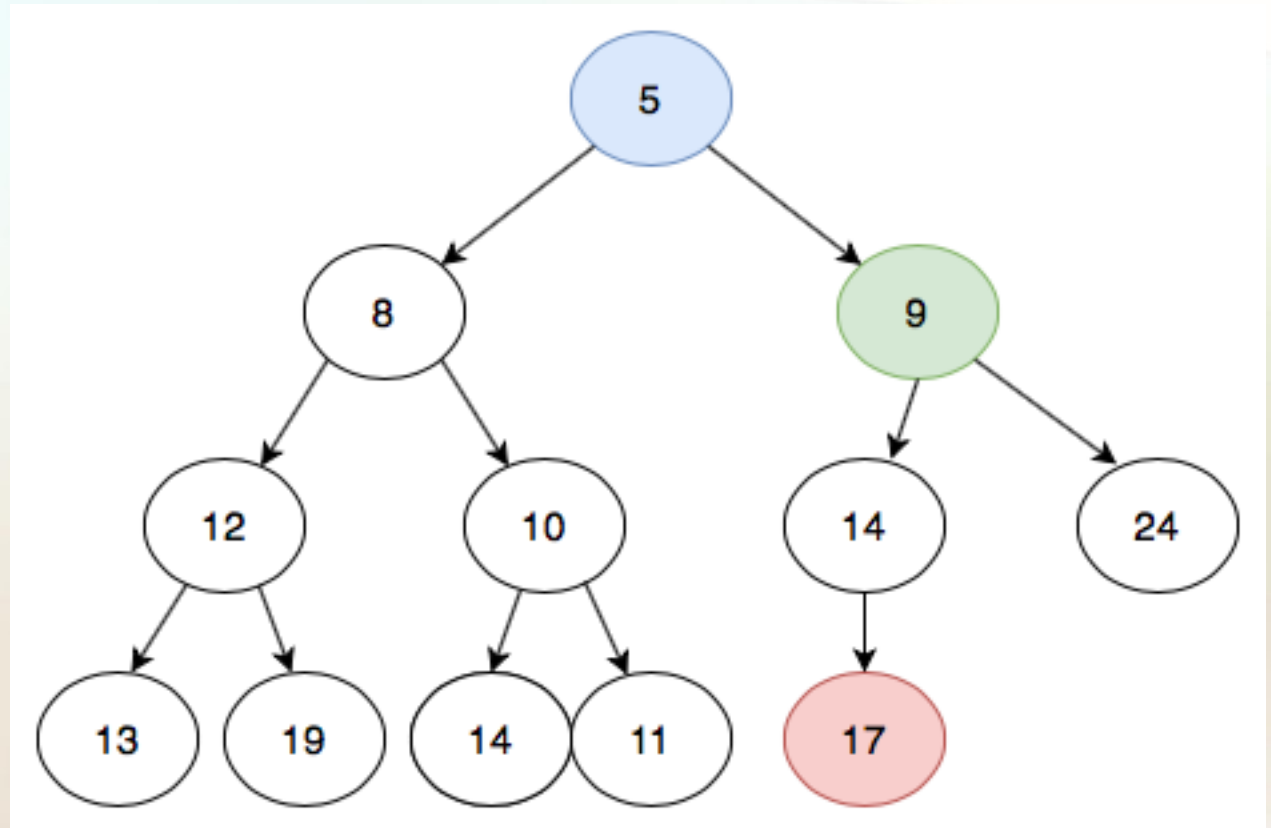
- Stanje heap-a



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[5, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11, 17]

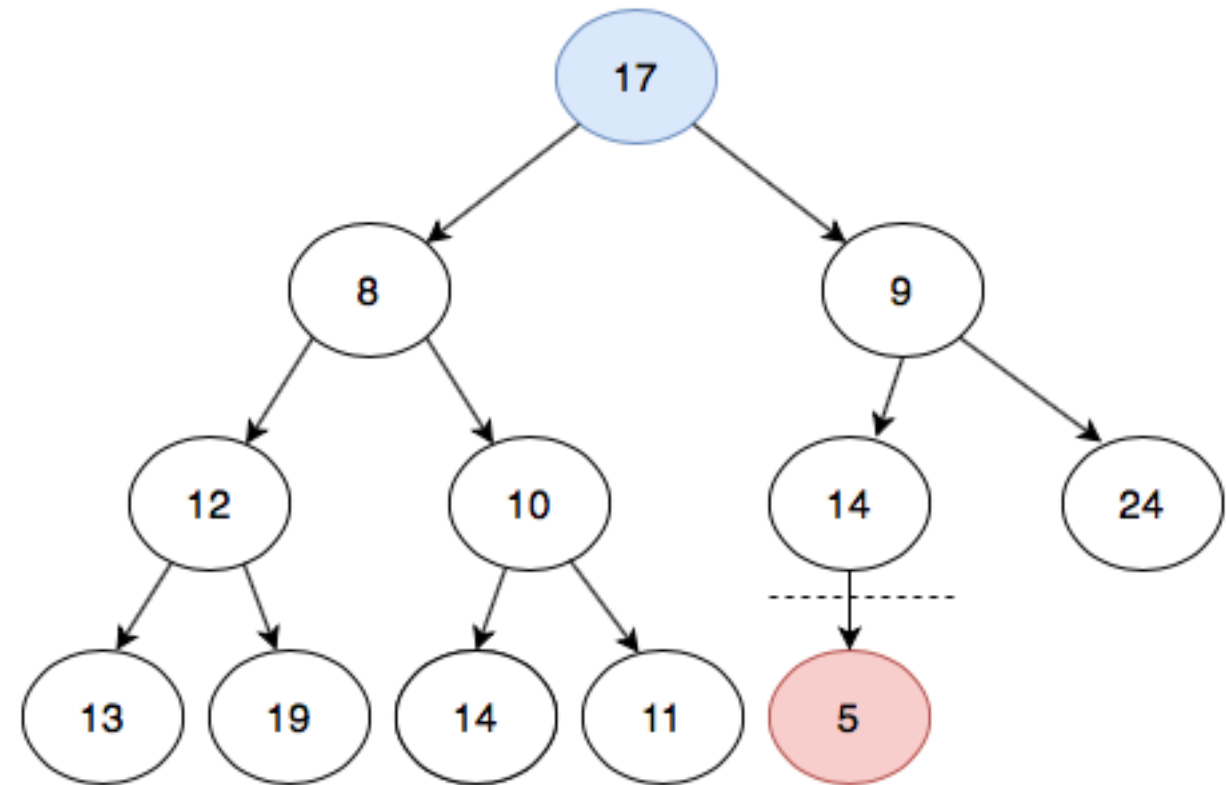
- Uklanjamo minimalni element
- Zamenjujemo prvi i poslednji element



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10 11
[17, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11, 5]

- Uklanjamo najmanji



Implementacija heap-a pomoću niza

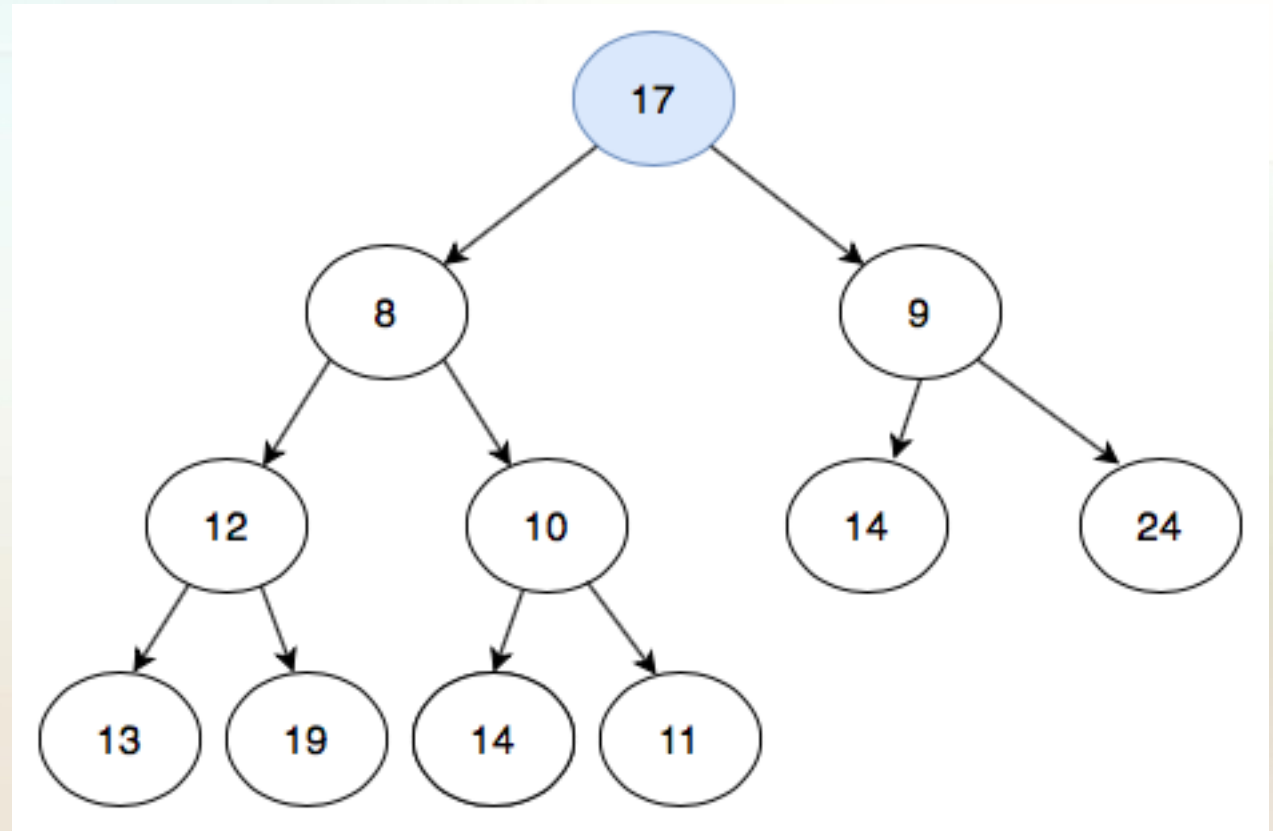
0 1 2 3 4 5 6 7 8 9 10
[17, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11]

- Pronalazimo decu čvora 17

$\text{left_child_index} = 0 * 2 + 1 = 1$

$\text{right_child_index} = 0 * 2 + 2 = 2$

- To su čvorovi sa ključevima 8 i 9.



Implementacija heap-a pomoću niza

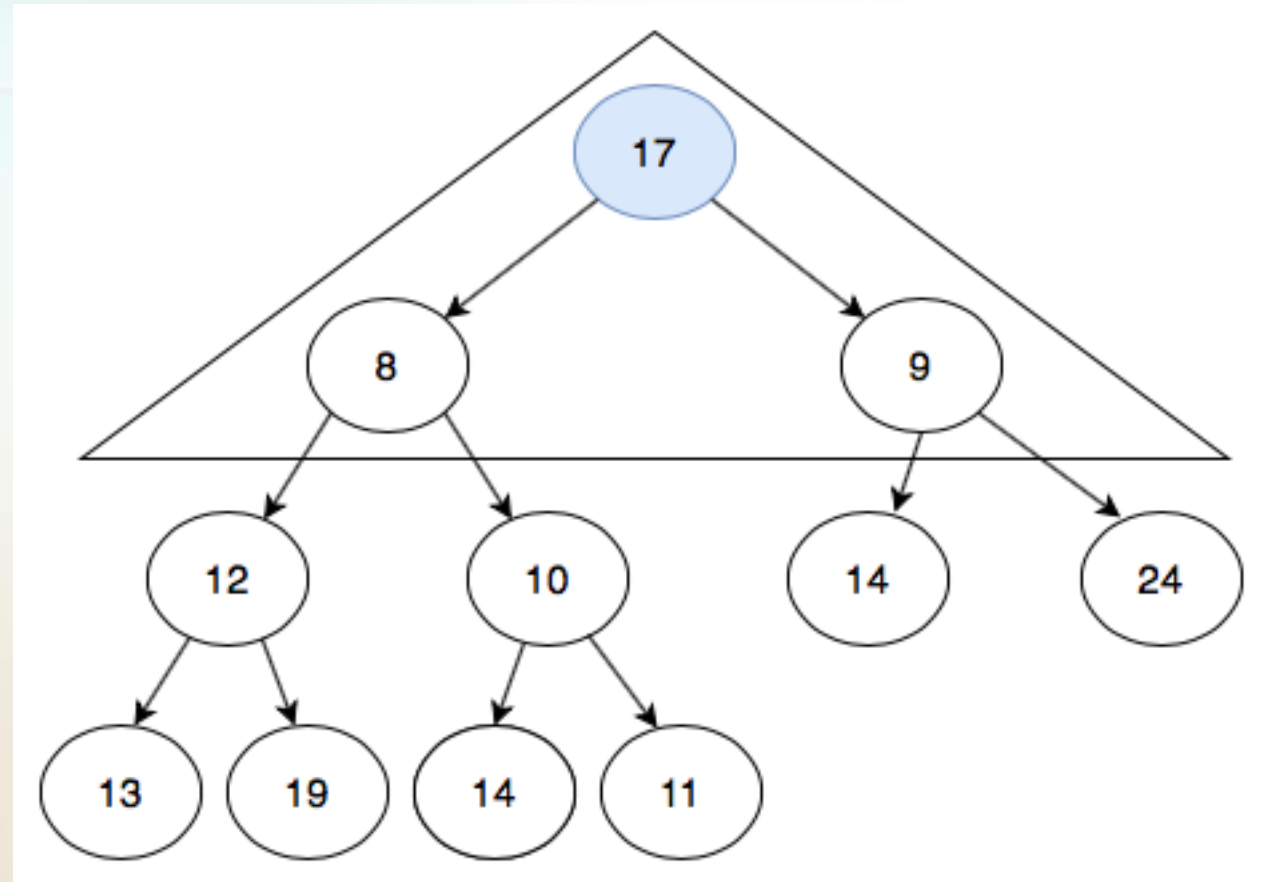
0 1 2 3 4 5 6 7 8 9 10
[17, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11]

- Pronalazimo decu čvora 17

$\text{left_child_index} = 0 * 2 + 1 = 1$

$\text{right_child_index} = 0 * 2 + 2 = 2$

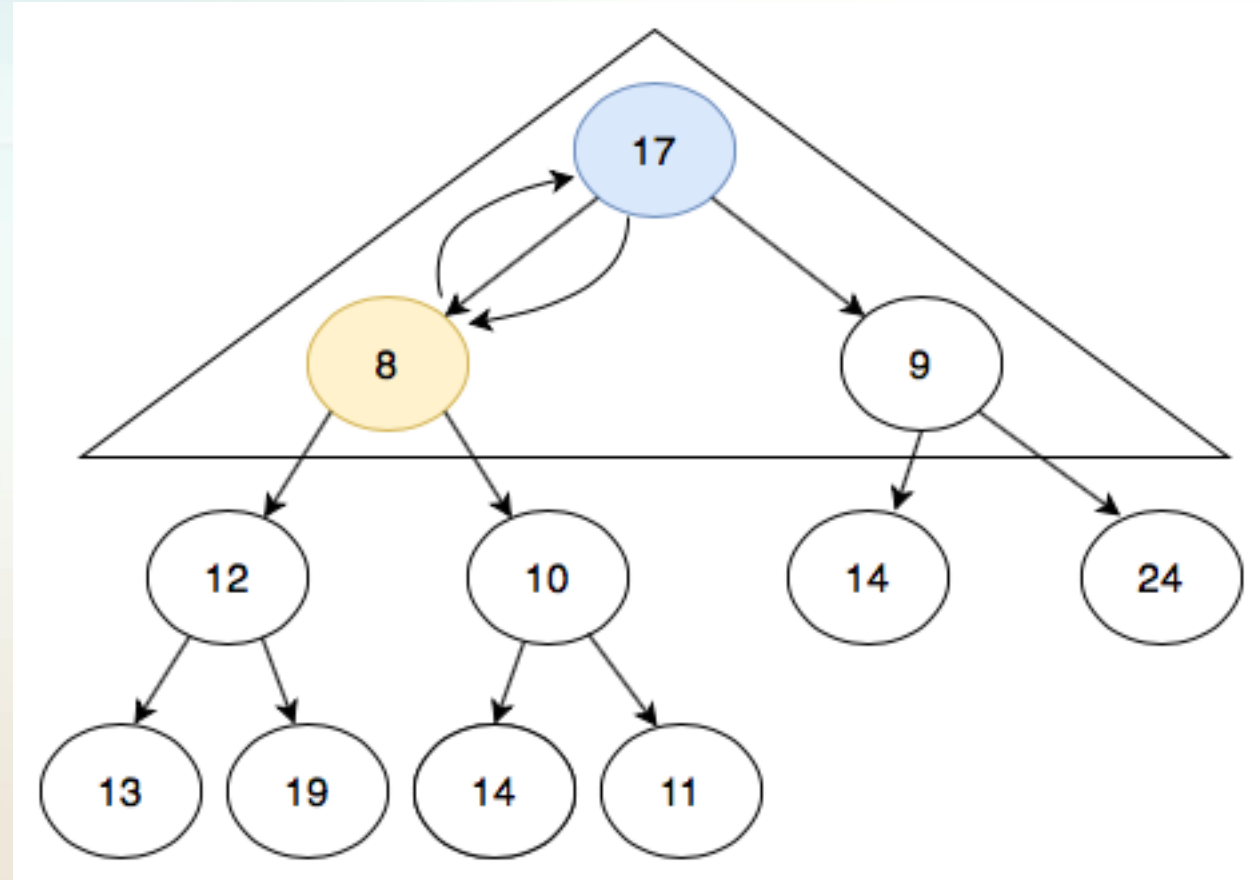
- To su čvorovi sa ključevima 8 i 9.



Implementacija heap-a pomoću niza

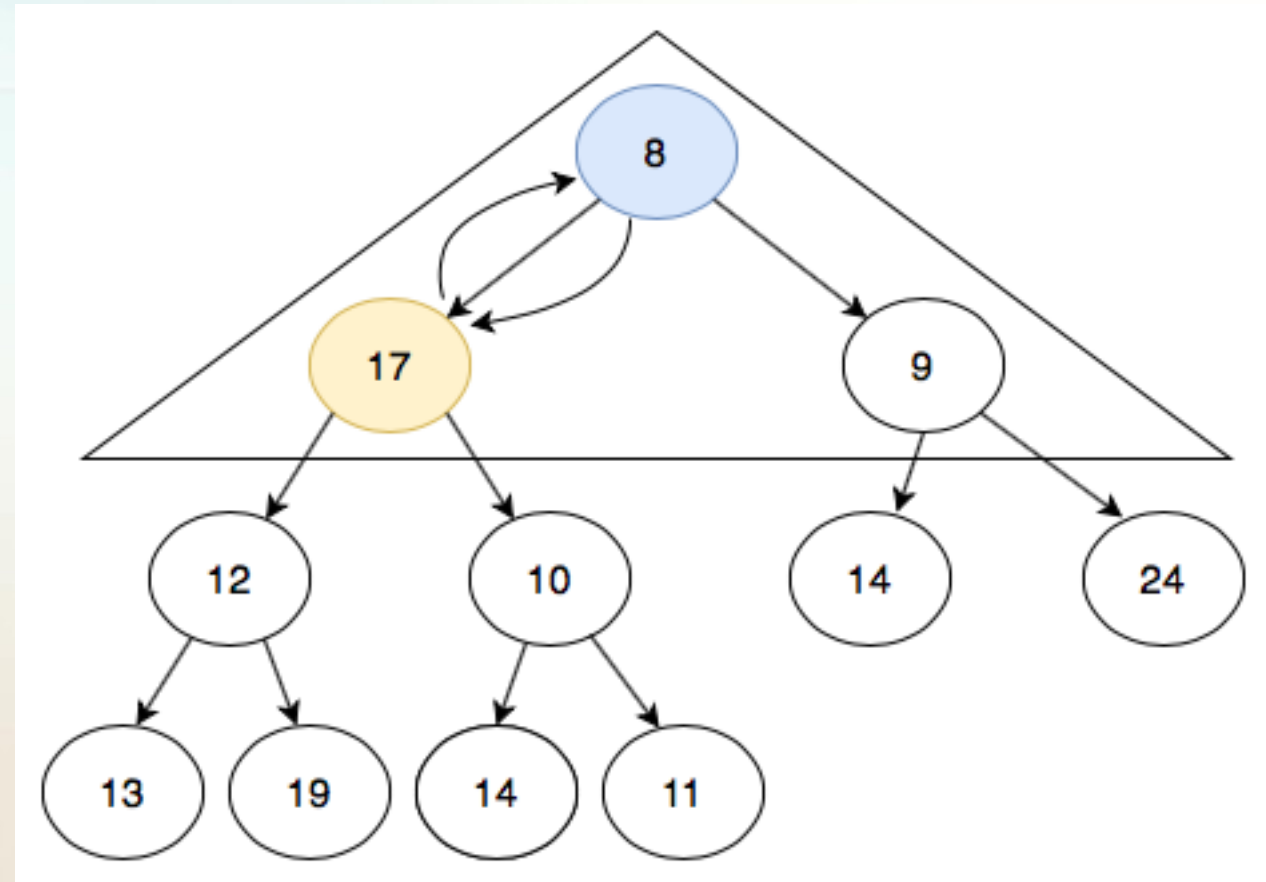
0 1 2 3 4 5 6 7 8 9 10
[17, 8, 9, 12, 10, 14, 24, 13, 19, 14, 11]

- Najmanji od njih je 8.
- Zamenjujemo čvorove.



Implementacija heap-a pomoću niza

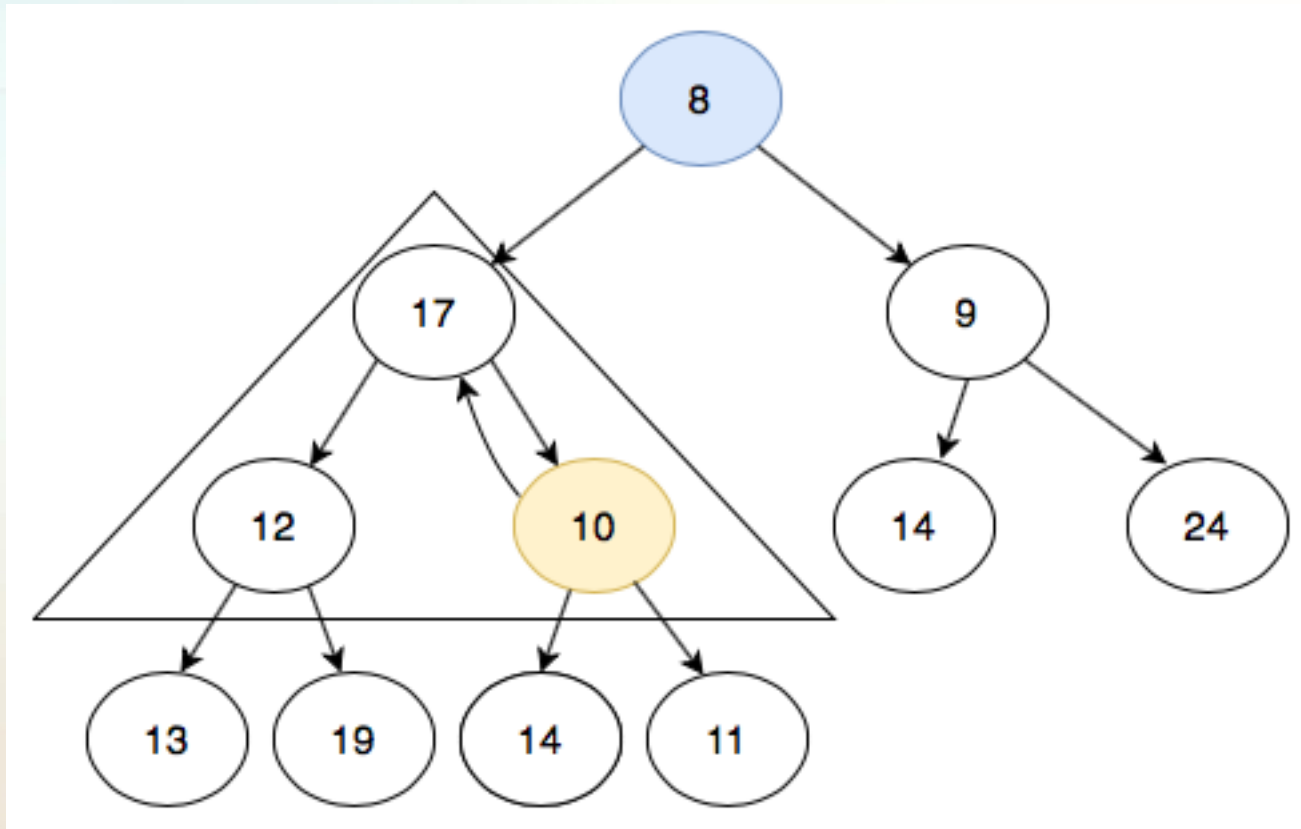
0 1 2 3 4 5 6 7 8 9 10
[8, 17, 9, 12, 10, 14, 24, 13, 19, 14, 11]



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[8, 17, 9, 12, 10, 14, 24, 13, 19, 14, 11]

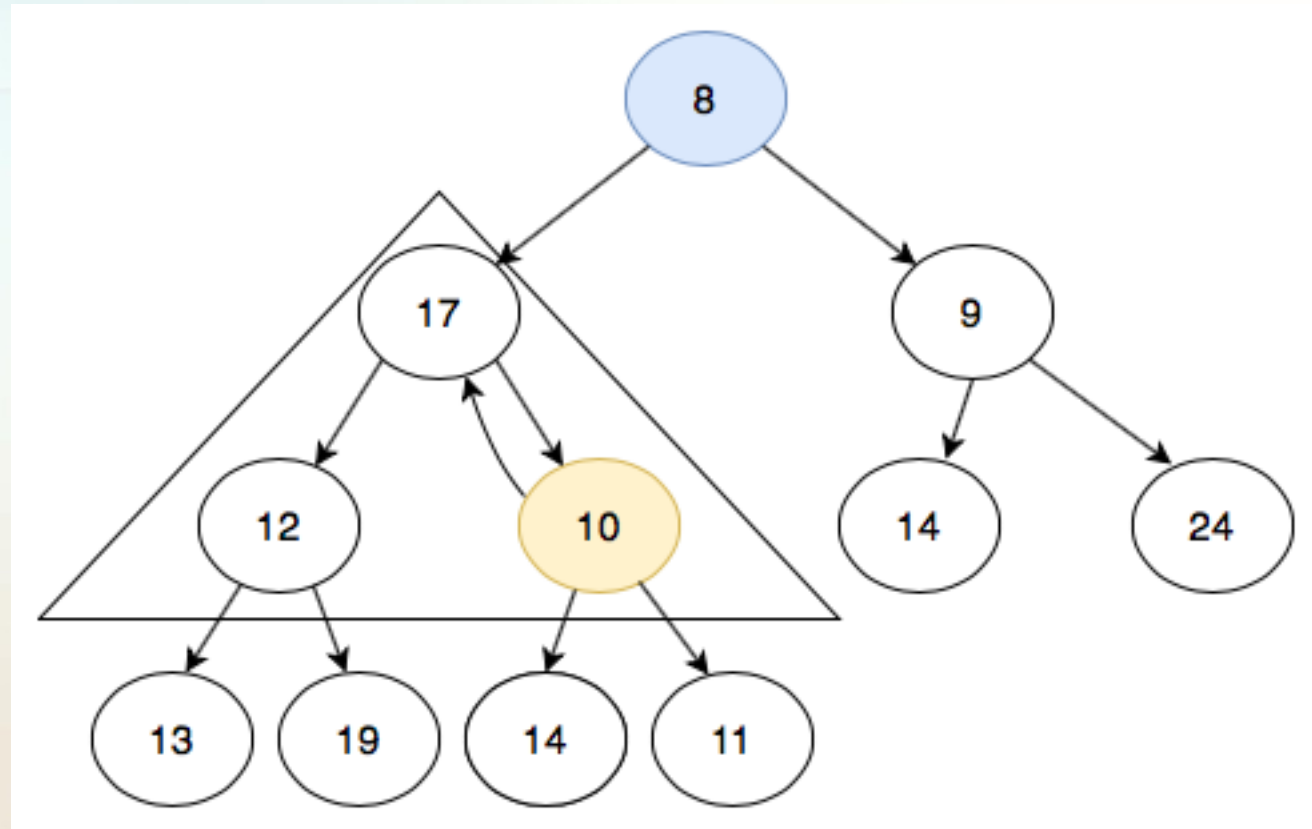
- Postupak se nastavlja



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[8, 17, 9, 12, 10, 14, 24, 13, 19, 14, 11]

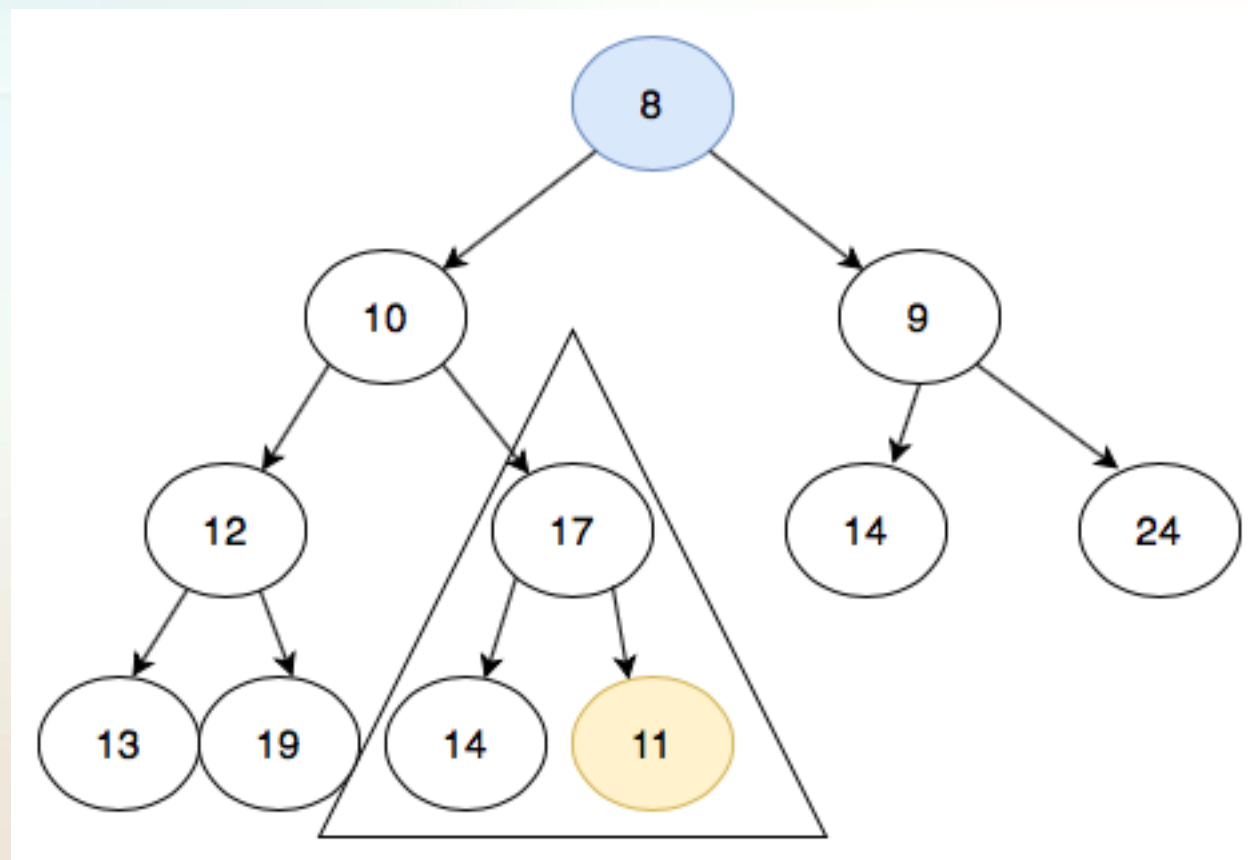
- Postupak se nastavlja
- Čvor 10 je najmanji.



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[8, 10, 9, 12, 17, 14, 24, 13, 19, 14, 11]

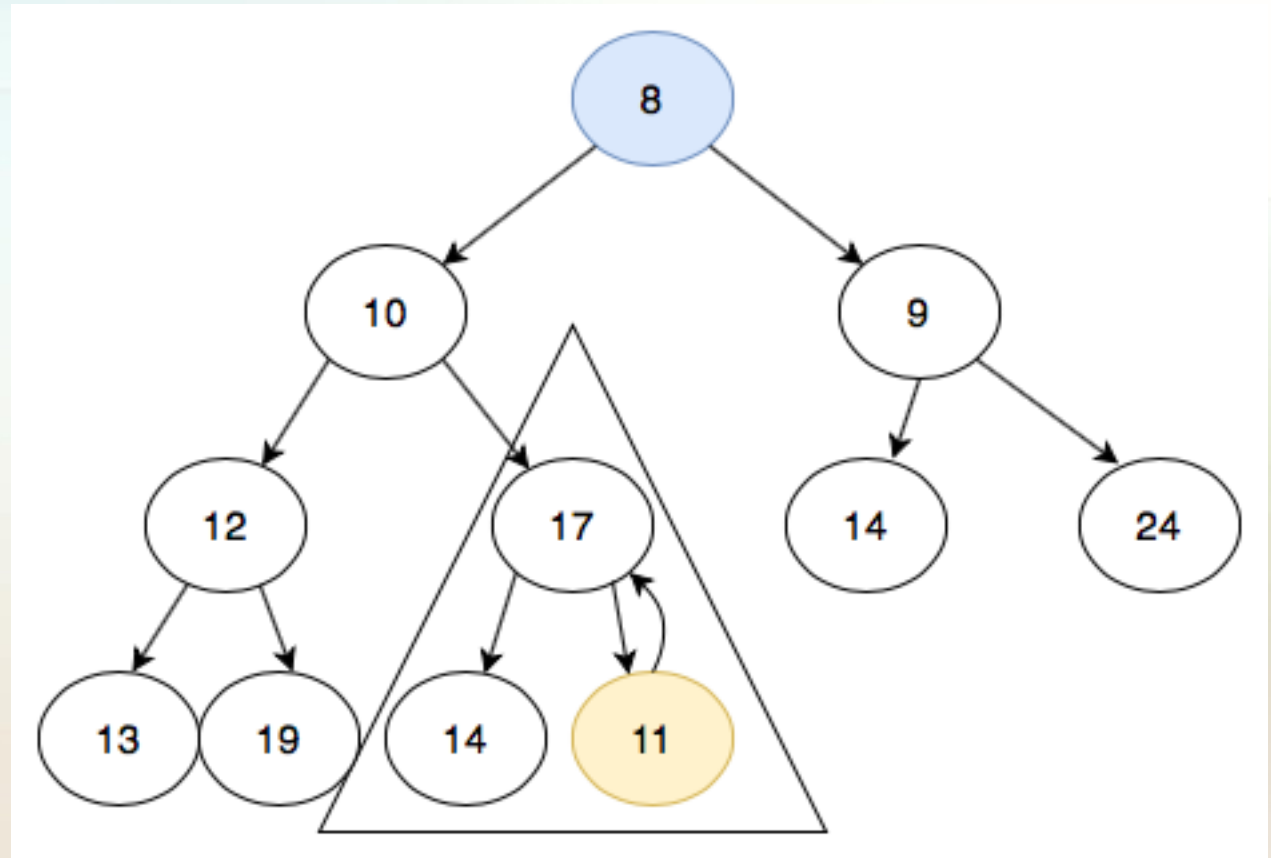
- Postupak se nastavlja



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[8, 10, 9, 12, 17, 14, 24, 13, 19, 14, 11]

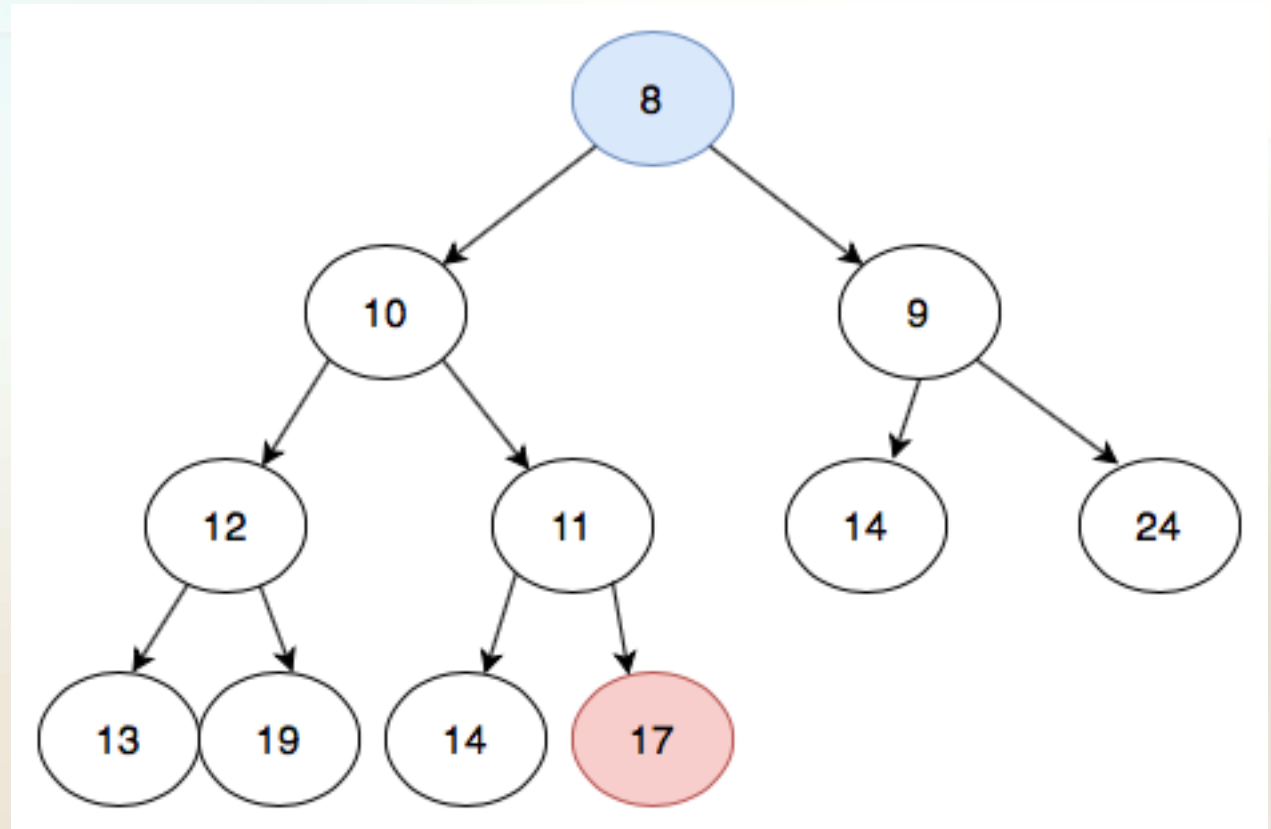
- Zamenjujemo mesta čvorovima



Implementacija heap-a pomoću niza

0 1 2 3 4 5 6 7 8 9 10
[8, 10, 9, 12, 11, 14, 24, 13, 19, 14, 17]

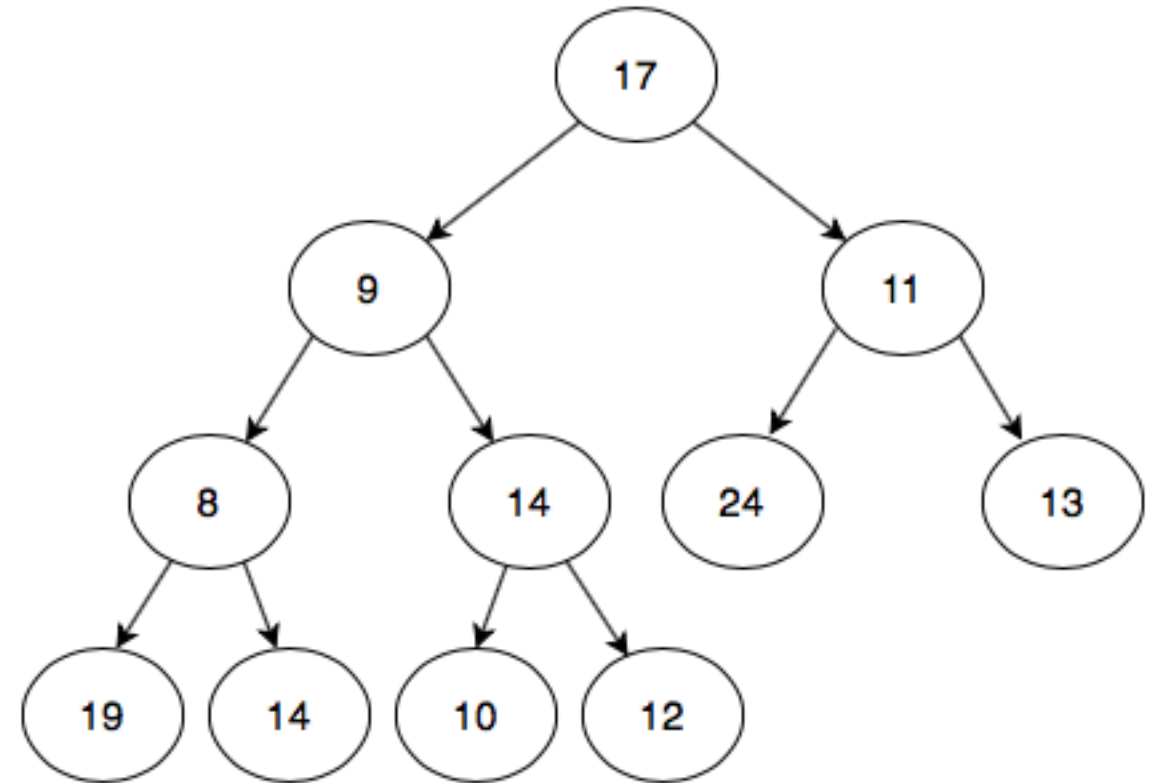
- Postupak se završava



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 14, 24, 13, 19, 14, 10, 12]

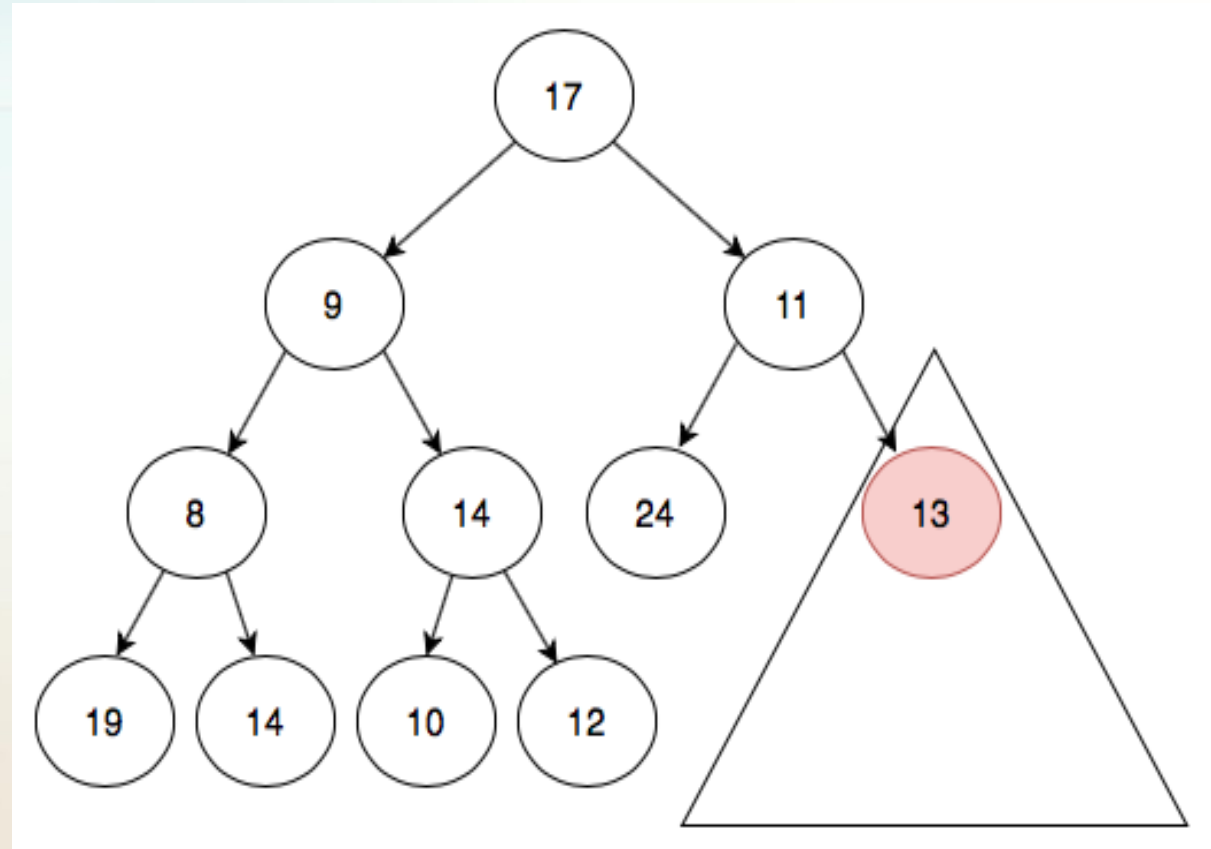
- Na početku, uzimamo da se heap sastoji redom od elemenata kolekcije koju želimo da sortiramo.
- Struktura koju dobijamo ne ispunjava uslove za heap.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 14, 24, 13, 19, 14, 10, 12]

- Počnemo od čvorova na nižim nivoima i uspostavljamo ispravan redosled u podheap-ovima (heap-ovima čiji je koren zadati čvor).
- Ako poželimo da uspostavimo ispravan redosled u podheap-u sa korenom u čvoru sa vrednošću 13, postupak bi bio trivijalan s obzirom da je u pitanju lisni čvor.



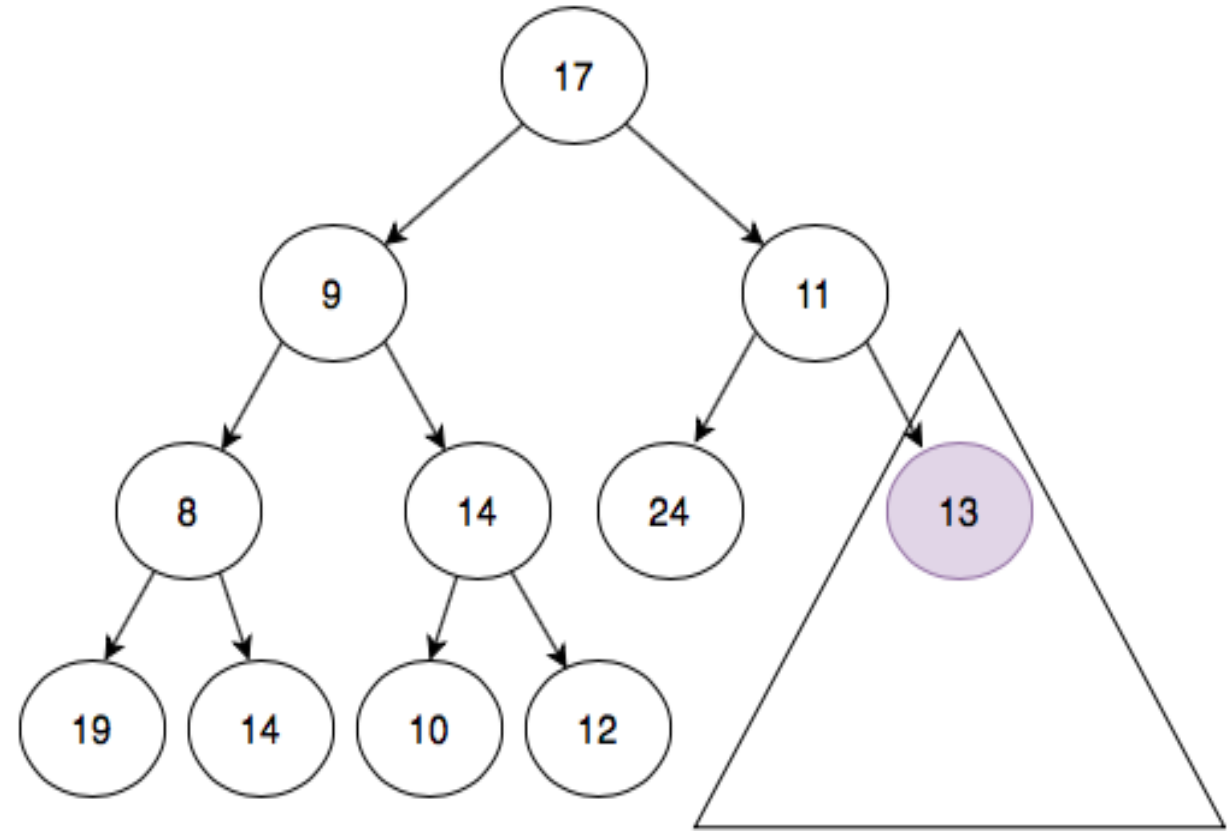
Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 14, 24, 13, 19, 14, 10, 12]

- Isto važi za sve lisne čvorove
- Za niz dužine n (u našem slučaju 11) bar polovina čvorova su listovi.

`no_leaves = ceil(n/2)`

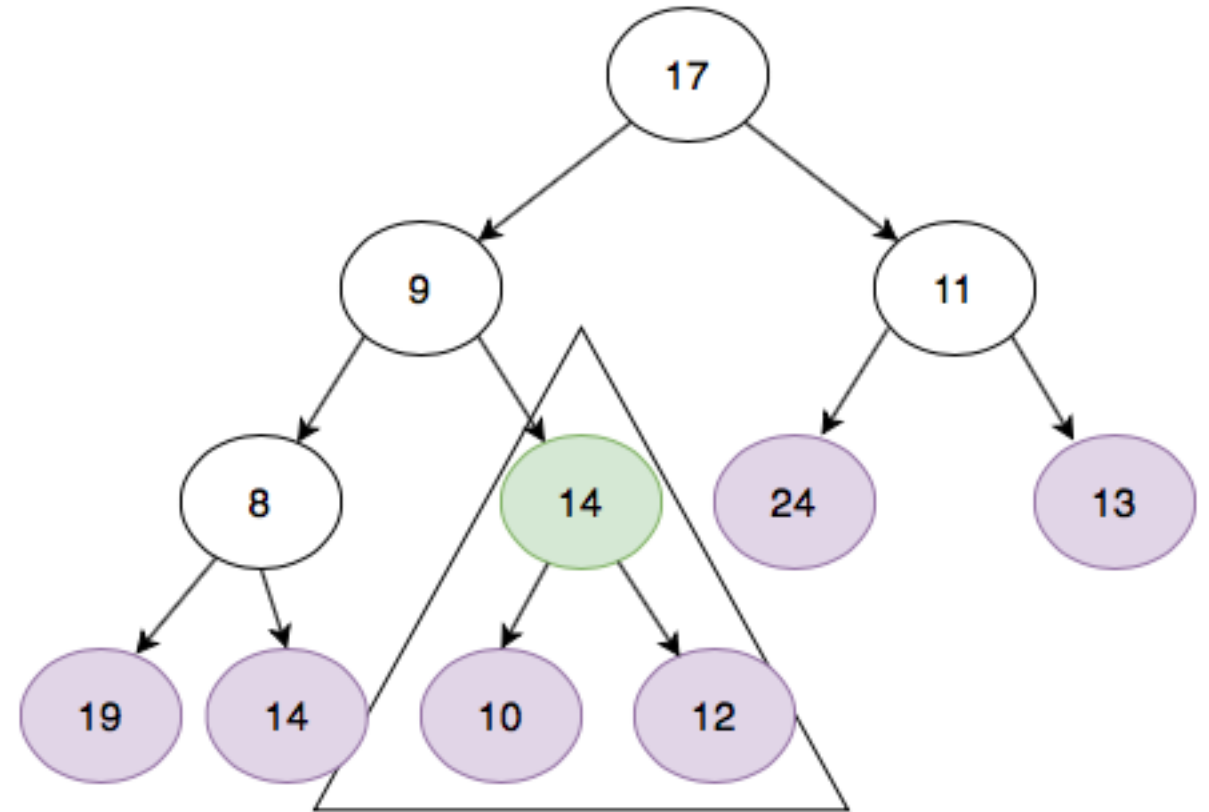
- Ukupno imamo 6 listova. Čvorovi na indeksima od 5 do 10 su listovi.
- Pozivamo restauriranje redosleda odnosno downheap za čvorove redom od indeksa 4 ka 0.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 14, 24, 13, 19, 14, 10, 12]

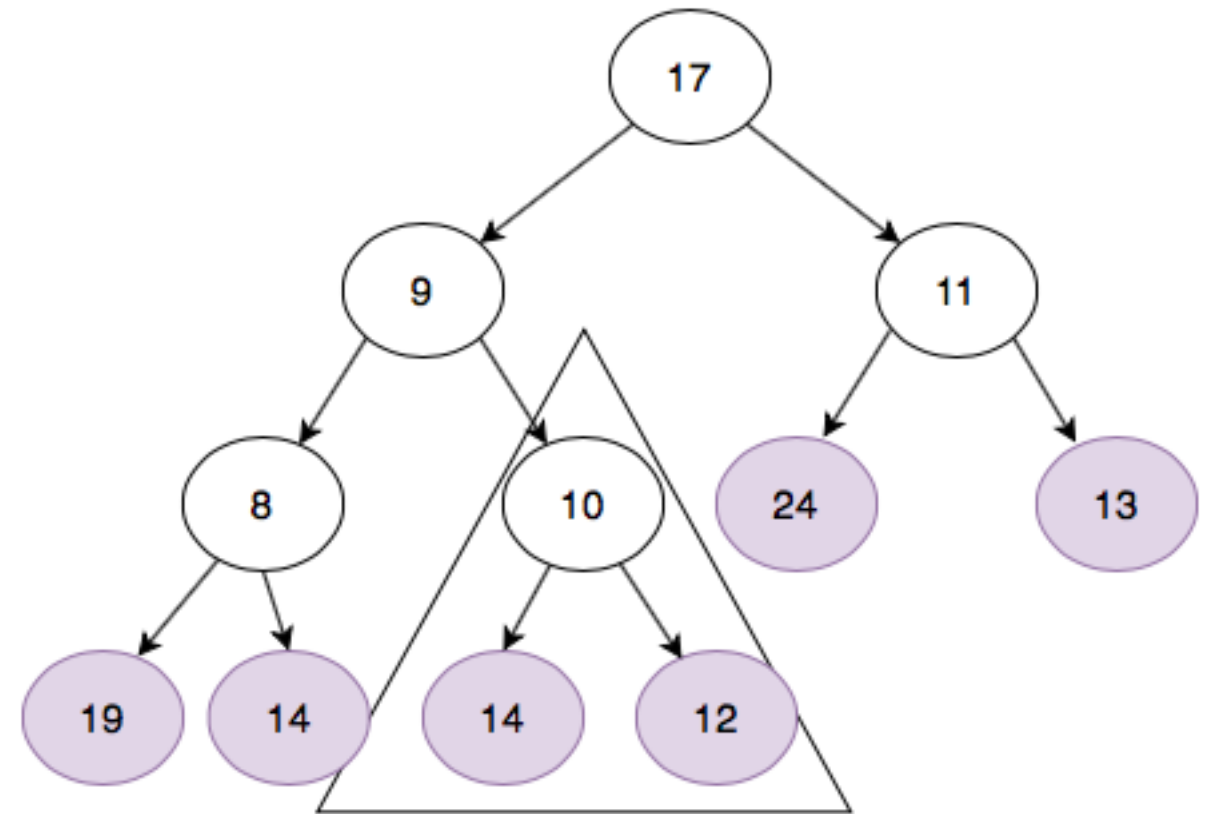
- Downheap počinje od elementa 14 na indeksu 4.
- Najmanji element je 10, on postaje roditelj.
- Zamenjujemo elemente 10 i 14.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 10, 24, 13, 19, 14, 14, 12]

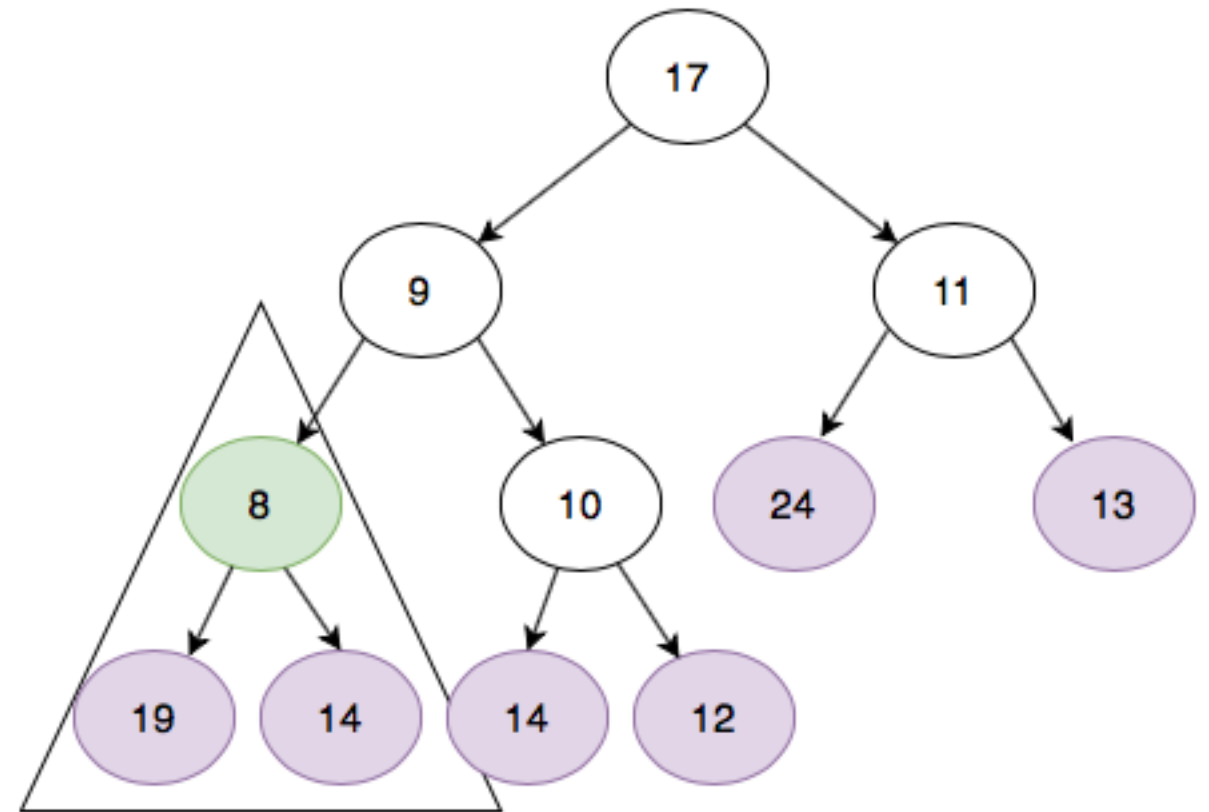
- Downheap nastavljamo sa elementom sa ključem 8 na indeksu 3.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 10, 24, 13, 19, 14, 14, 12]

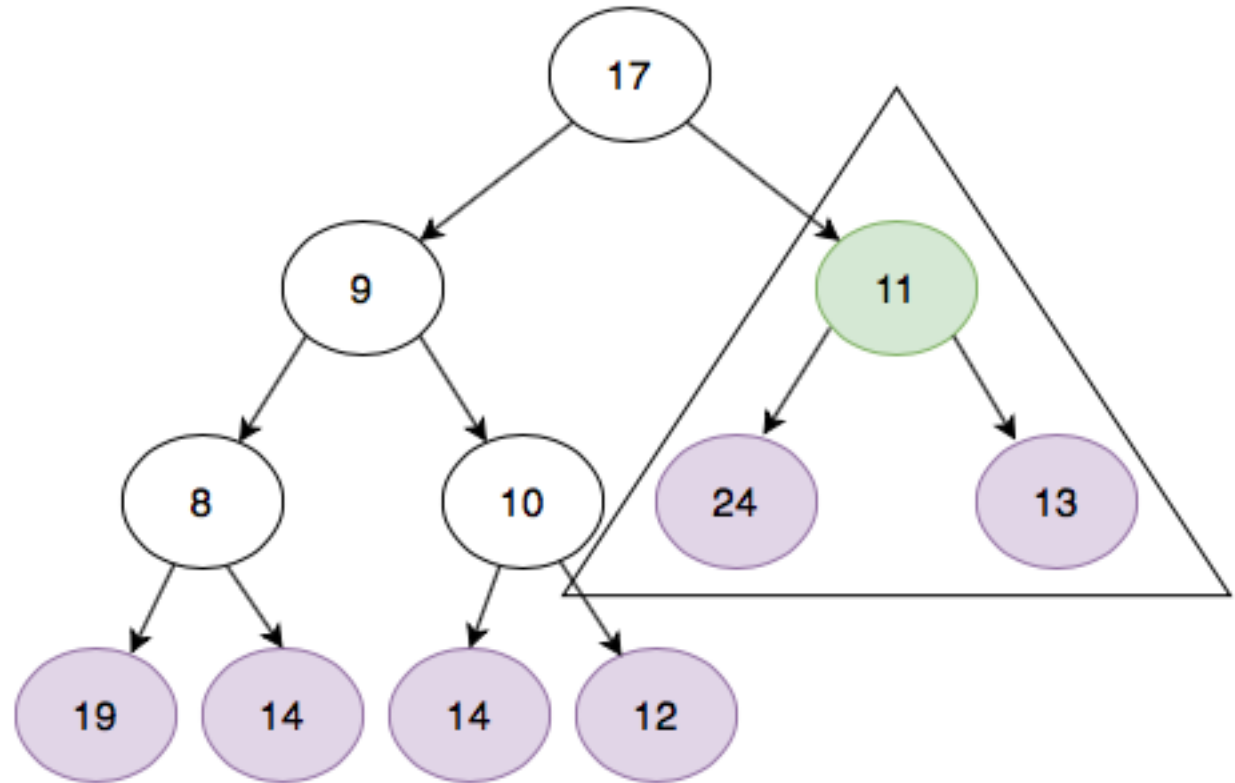
- Downheap nastavljamo sa elementom sa ključem 8 na indeksu 3.
- Poredimo ga sa decom.
- Najmanji element je 8, on ostaje roditelj.
- Postupak se završava bez zamene.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 10, 24, 13, 19, 14, 14, 12]

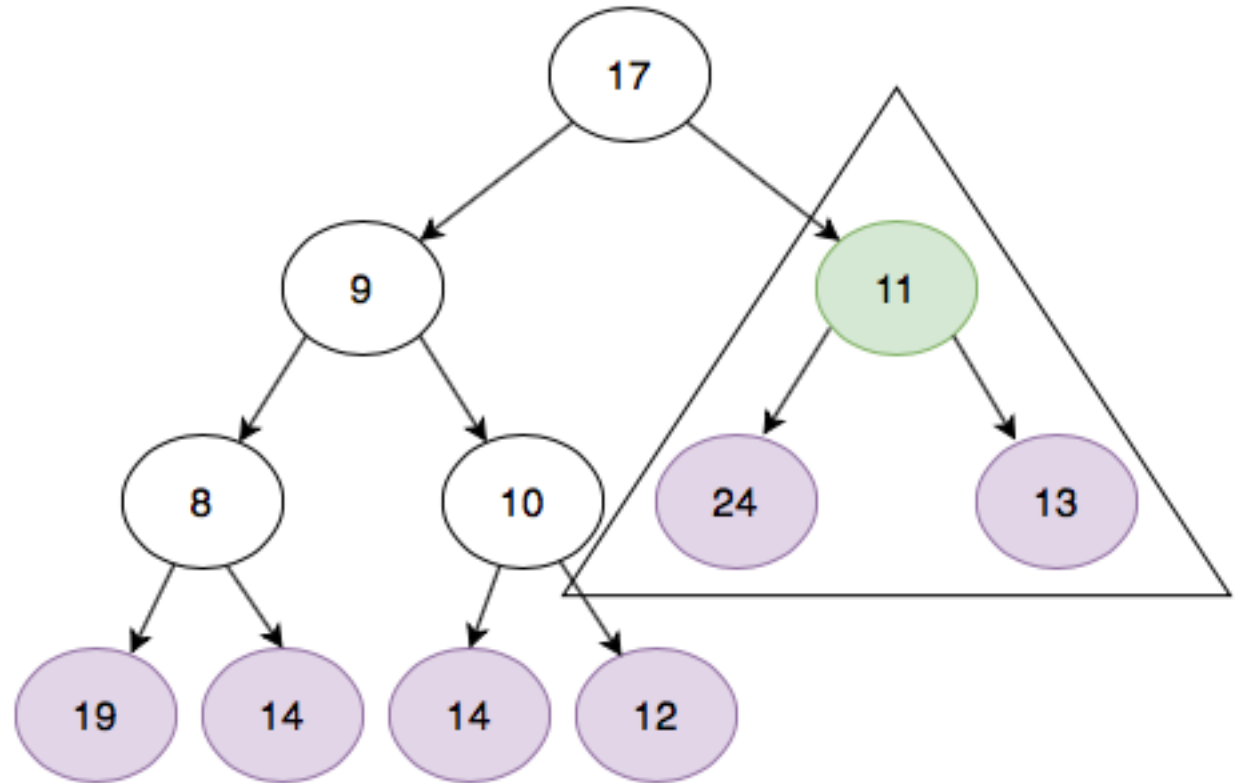
- Downheap nastavljamo sa elementom sa ključem 11 na indeksu 2.
- Poredimo ga sa decom.
- Najmanji element je 11, on ostaje roditelj.
- Postupak se završava bez zamene.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 10, 24, 13, 19, 14, 14, 12]

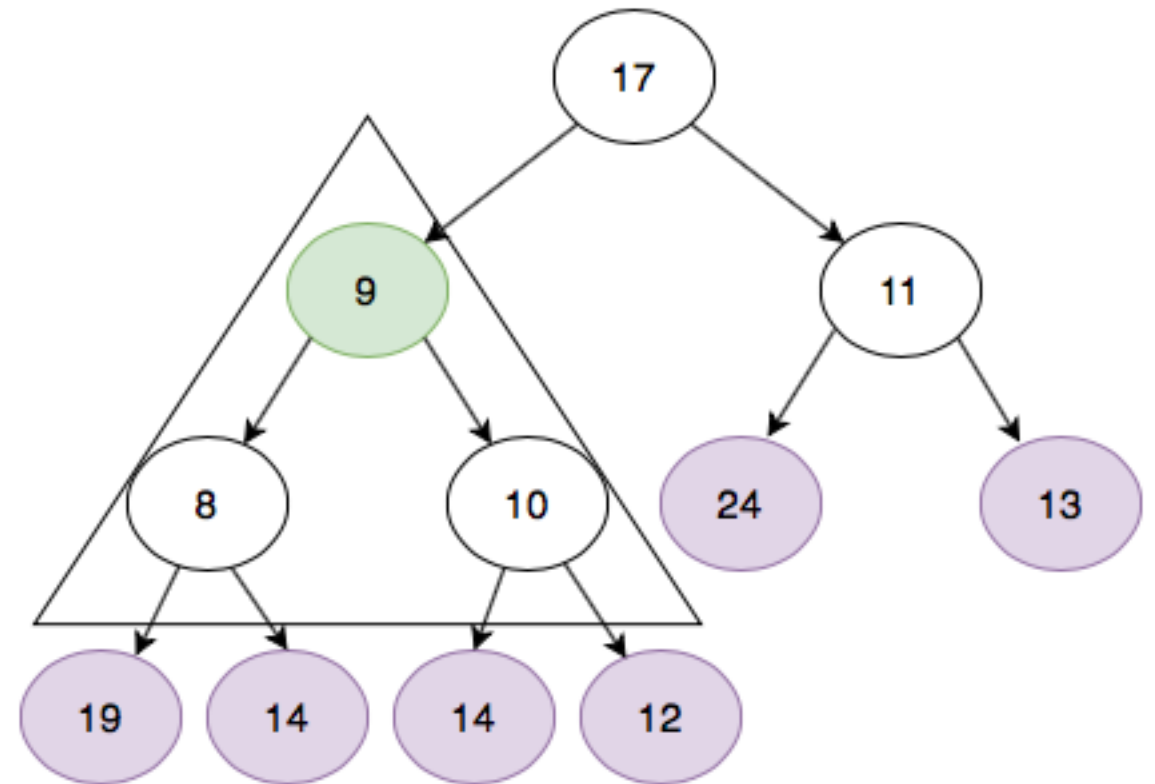
- Downheap nastavljamo sa elementom sa ključem 11 na indeksu 2.
- Poredimo ga sa decom.
- Najmanji element je 11, on ostaje roditelj.
- Postupak se završava bez zamene.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 9, 11, 8, 10, 24, 13, 19, 14, 14, 12]

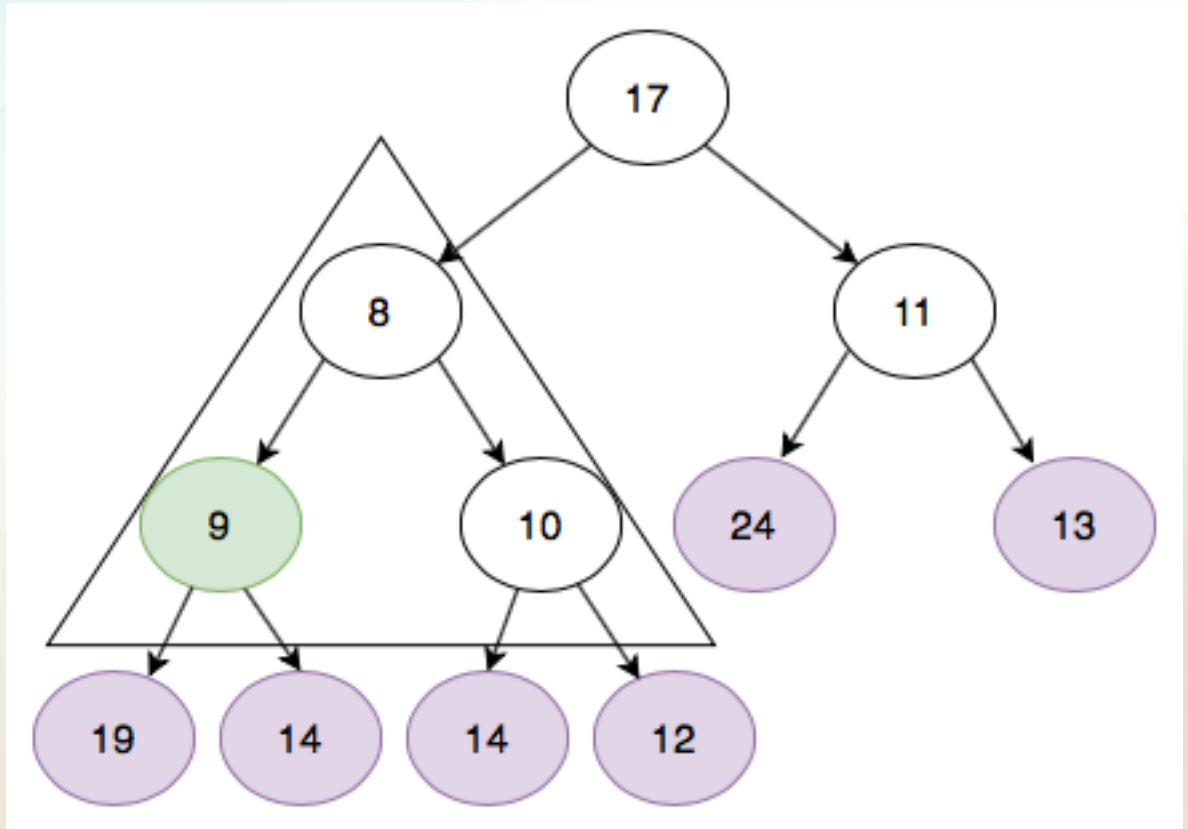
- Downheap nastavljamo sa elementom sa ključem 9 na indeksu 1.
- Poredimo ga sa decom.
- Najmanji element je 8, on postaje roditelj.
- Zamenjujemo 9 i 8.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 8, 11, 9, 10, 24, 13, 19, 14, 14, 12]

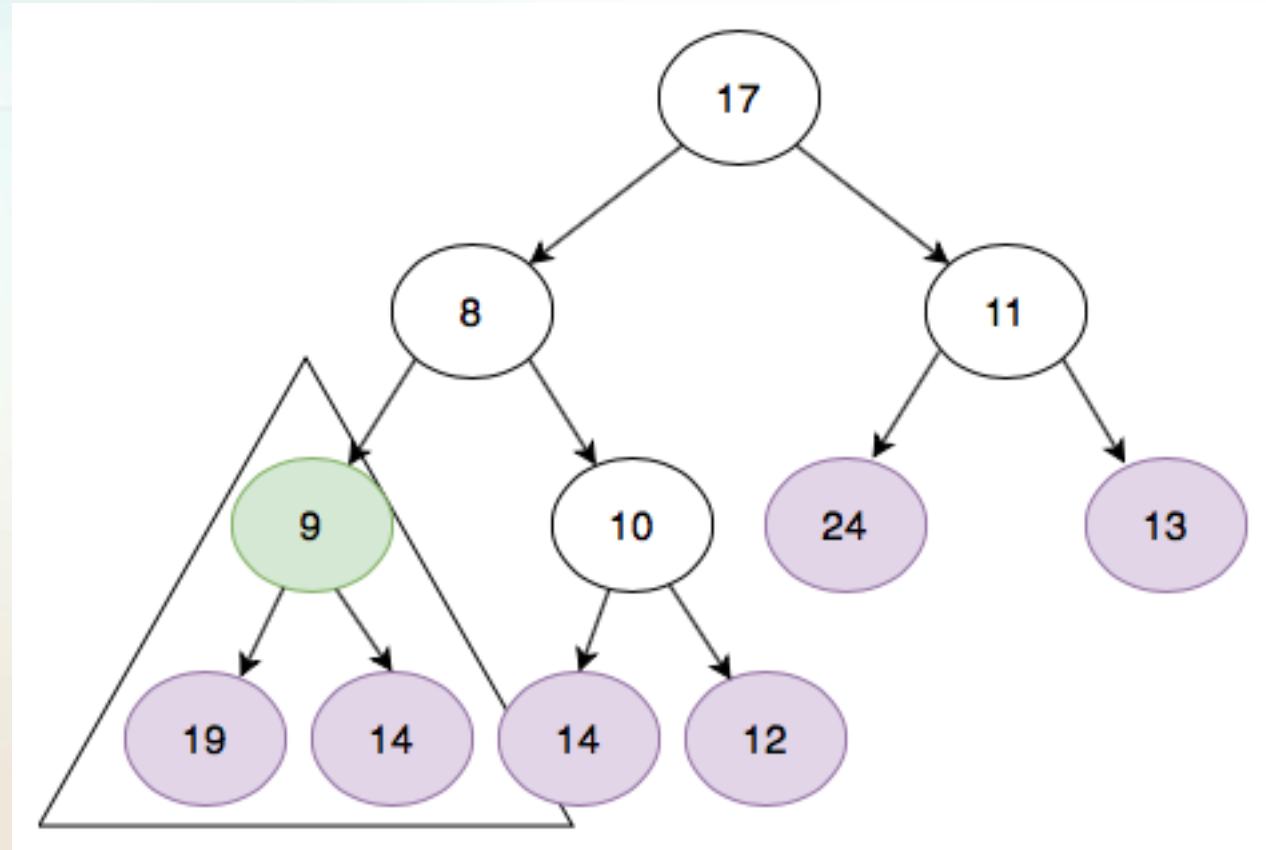
- Stanje posle zamene.
- Nastavljamo downheap za naš element 9.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 8, 11, 9, 10, 24, 13, 19, 14, 14, 12]

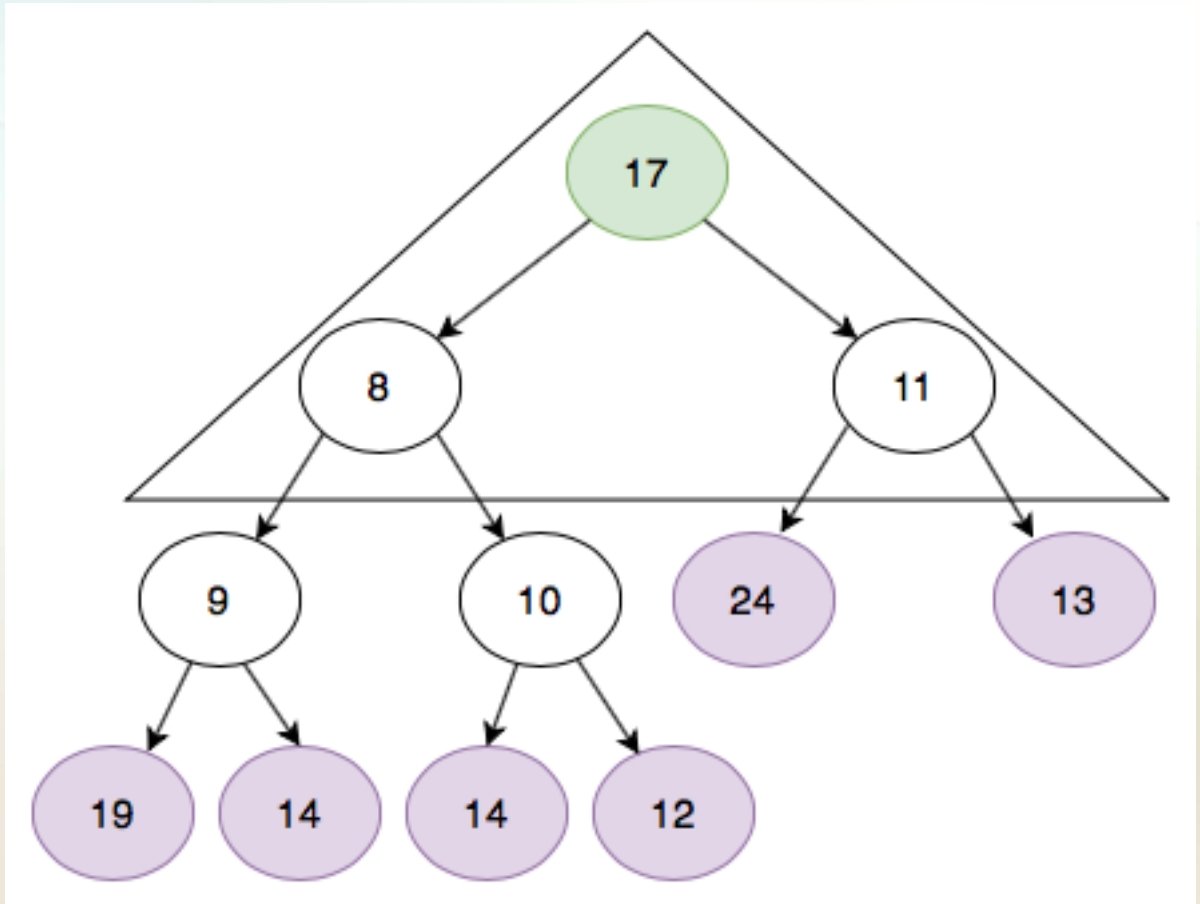
- Nastavljamo downheap za naš element 9.
- Poredimo element sa ključem 9 sa decom.
- Redosled je ispravan, zamene nisu potrebne.
- Postupak je završen za zadati element.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[17, 8, 11, 9, 10, 24, 13, 19, 14, 14, 12]

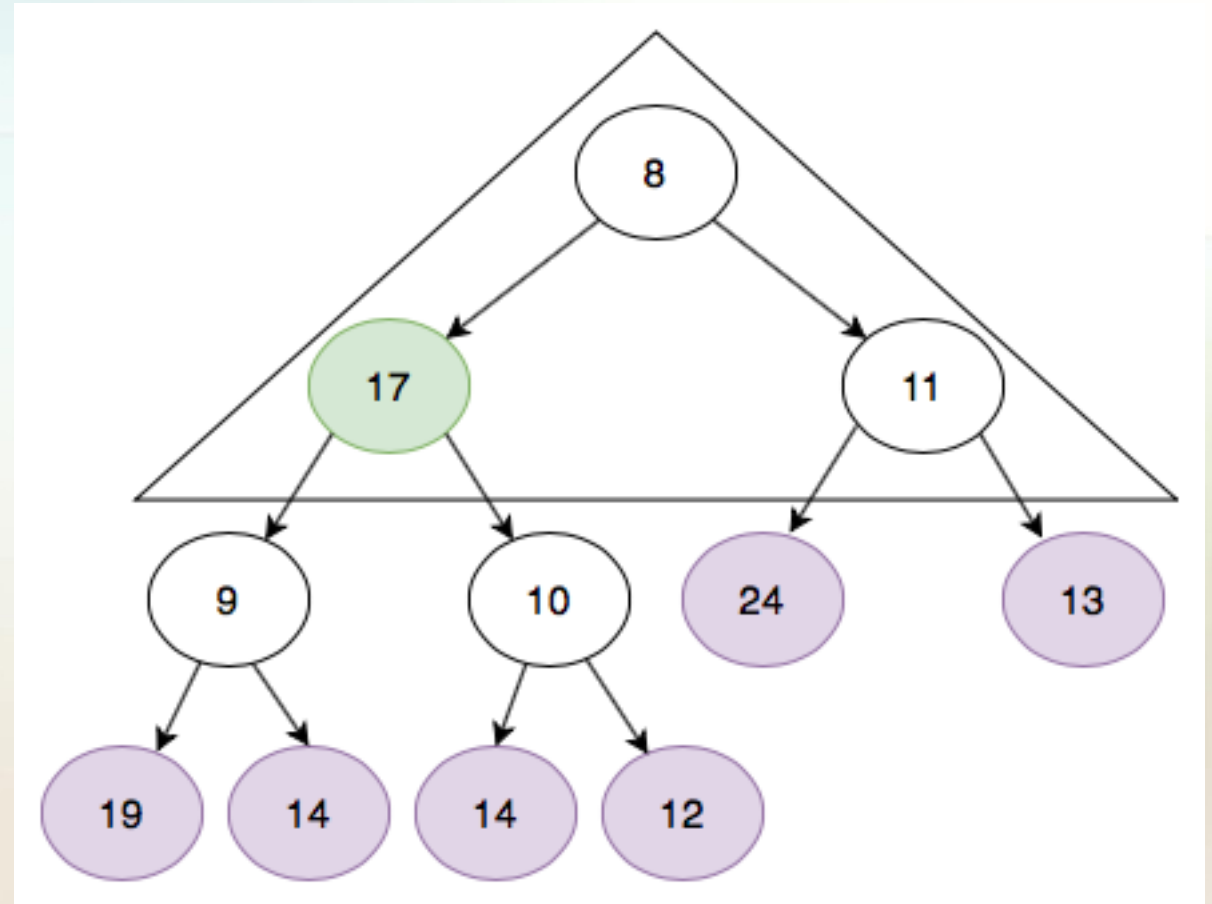
- Nastavljamo downheap za element 17.
- Poredimo element sa ključem 17 sa decom.
- Dete 8 je najmanje pa postaje roditelj.
- Zamenjujemo 17 i 8.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 17, 11, 9, 10, 24, 13, 19, 14, 14, 12]

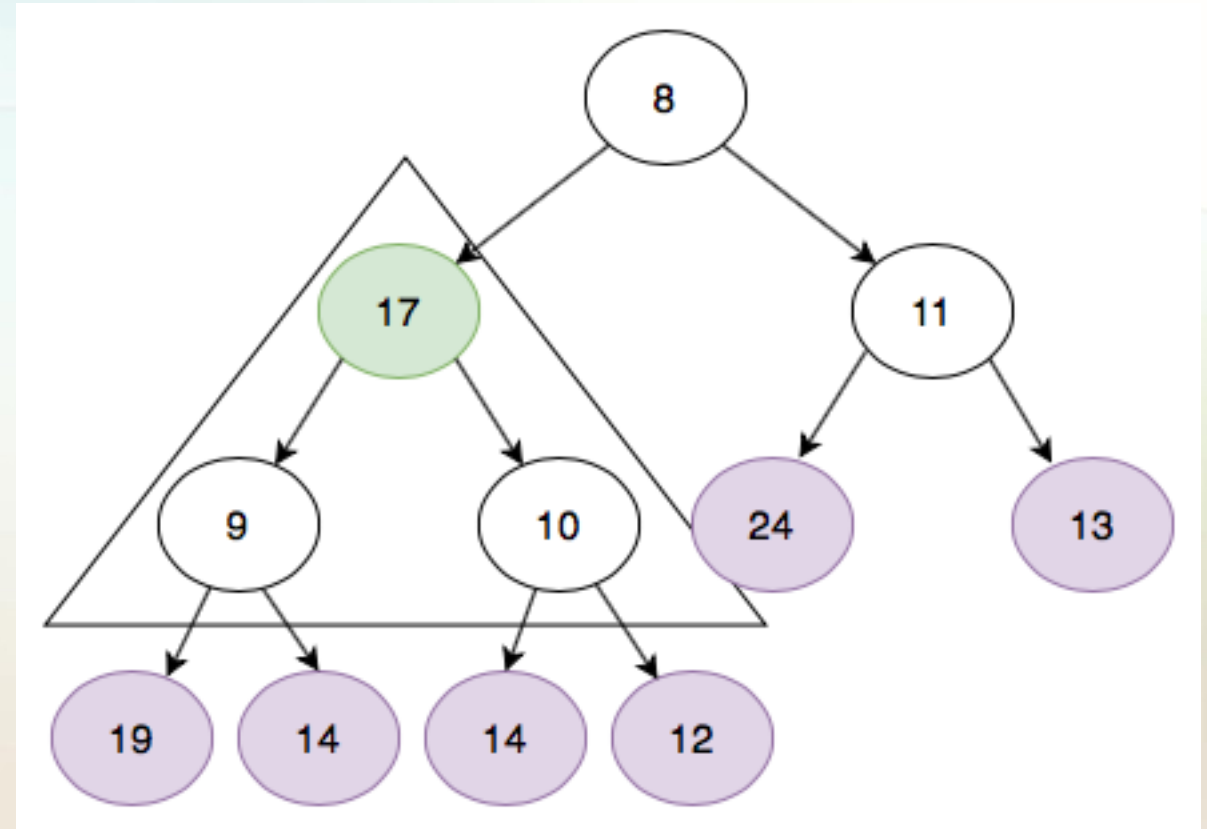
- Nastavljamo downheap za element 17.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 17, 11, 9, 10, 24, 13, 19, 14, 14, 12]

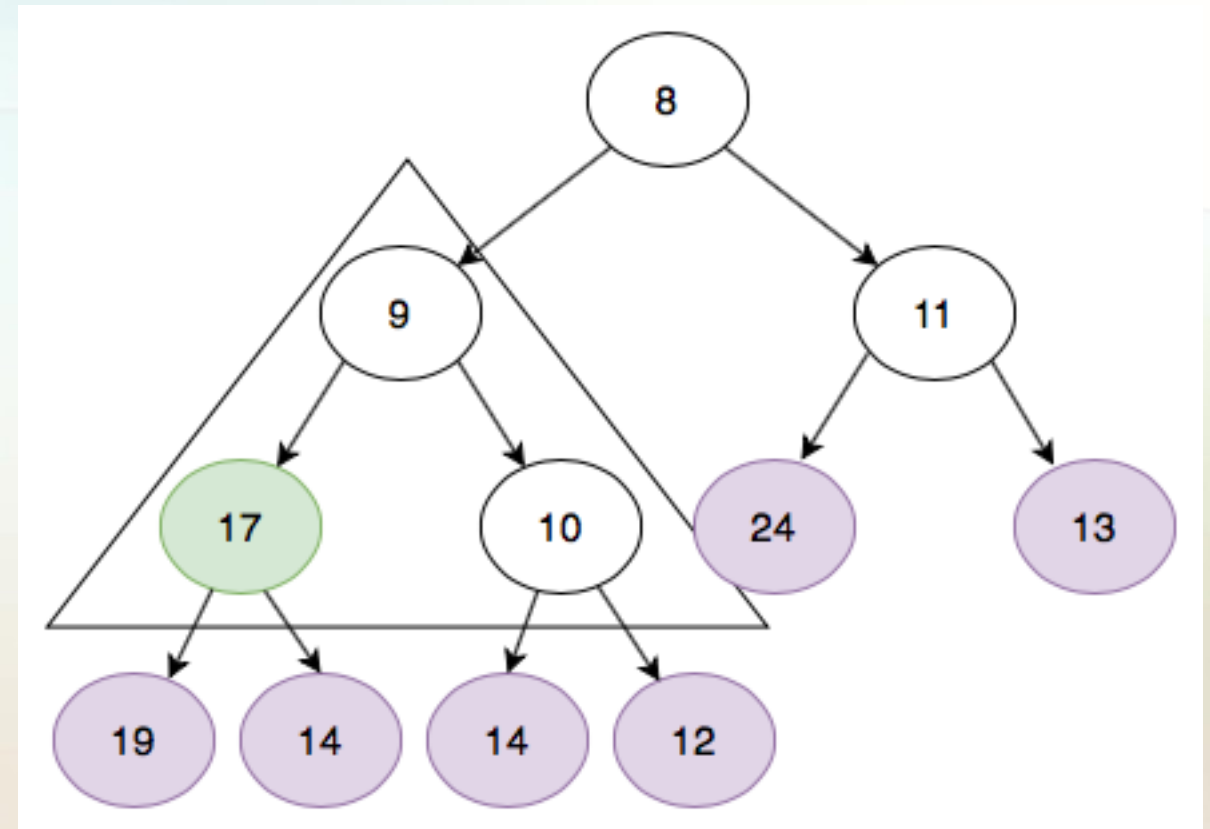
- Nastavljamo downheap za element 17.
- Poredimo element sa ključem 17 sa novom decom.
- Dete 9 je najmanje pa postaje roditelj.
- Zamenjujemo 17 i 9.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 9, 11, 17, 10, 24, 13, 19, 14, 14, 12]

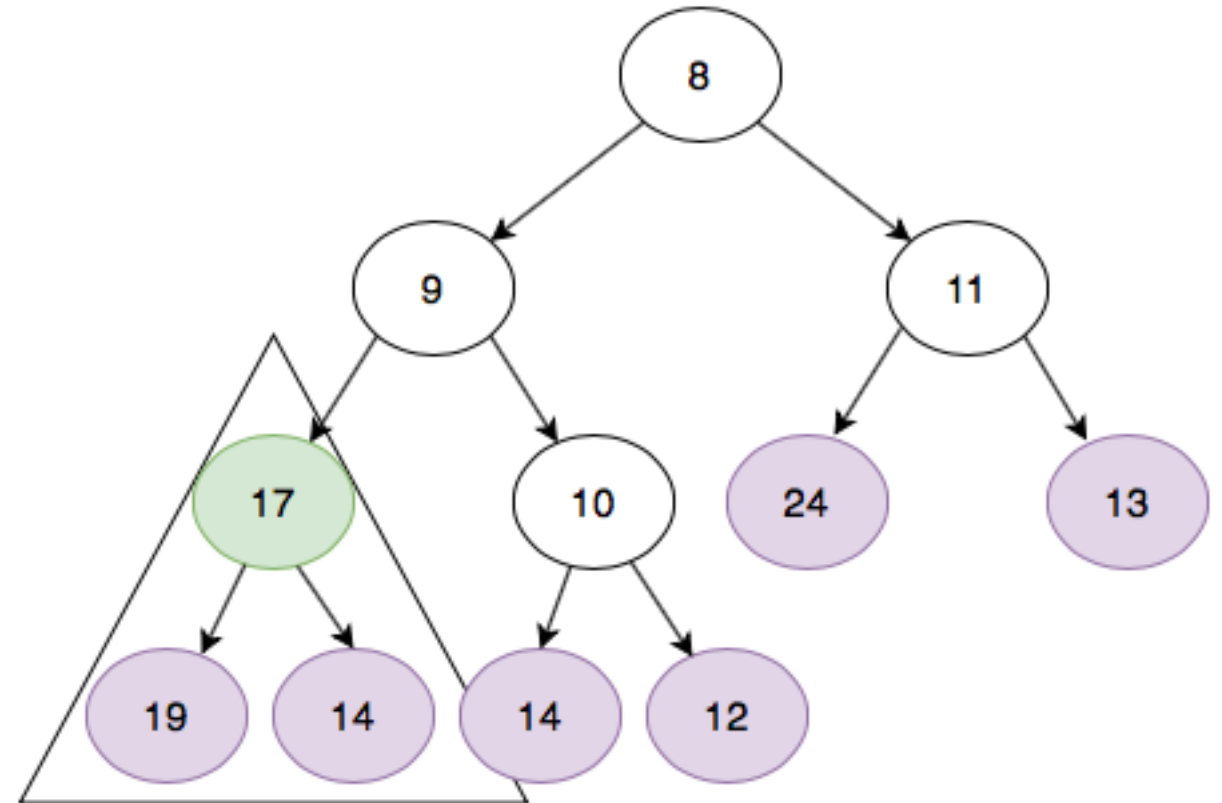
- Stanje posle zamene



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 9, 11, 17, 10, 24, 13, 19, 14, 14, 12]

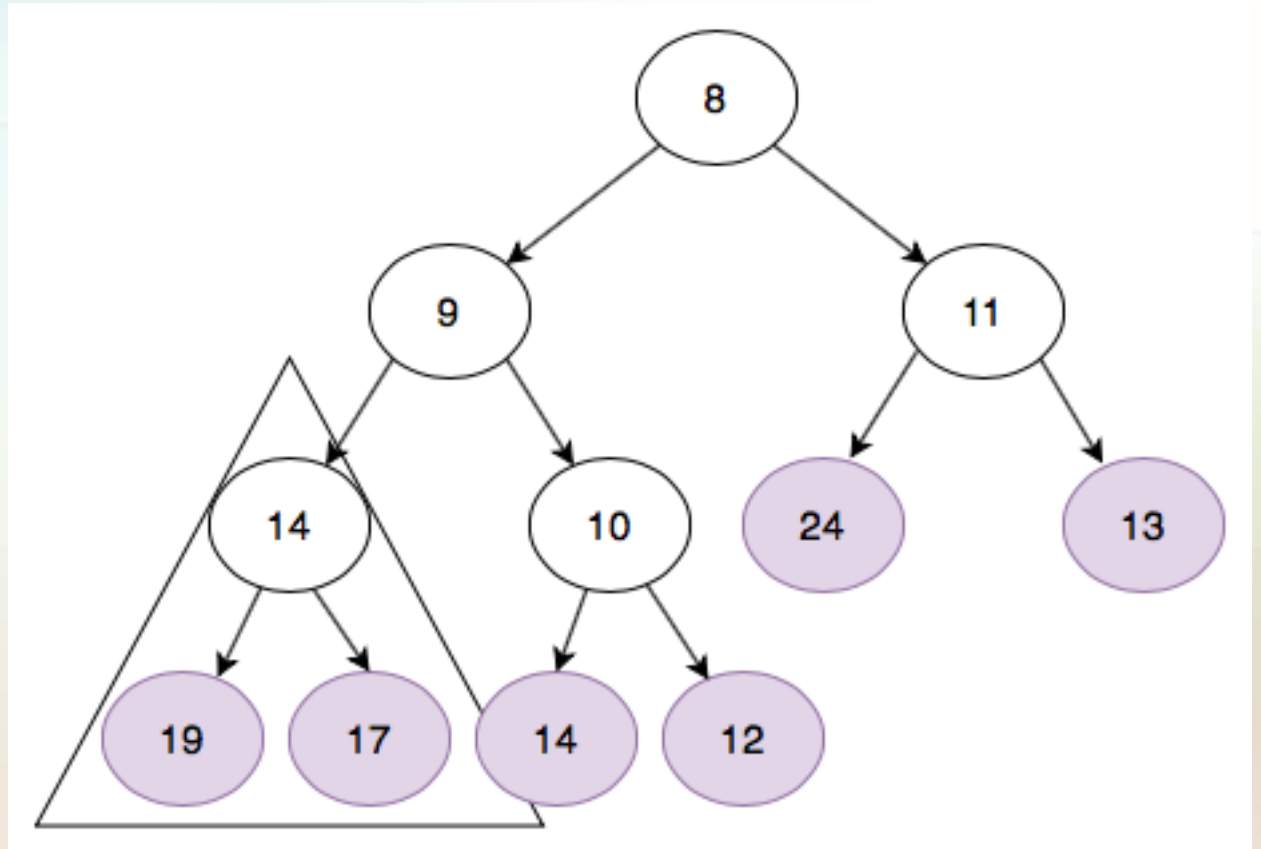
- Nastavljamo downheap za element 17.
- Poredimo element sa ključem 17 sa novom decom.
- Dete 14 je najmanje pa postaje roditelj.
- Zamenjujemo 17 i 14.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 9, 11, 14, 10, 24, 13, 19, 17, 14, 12]

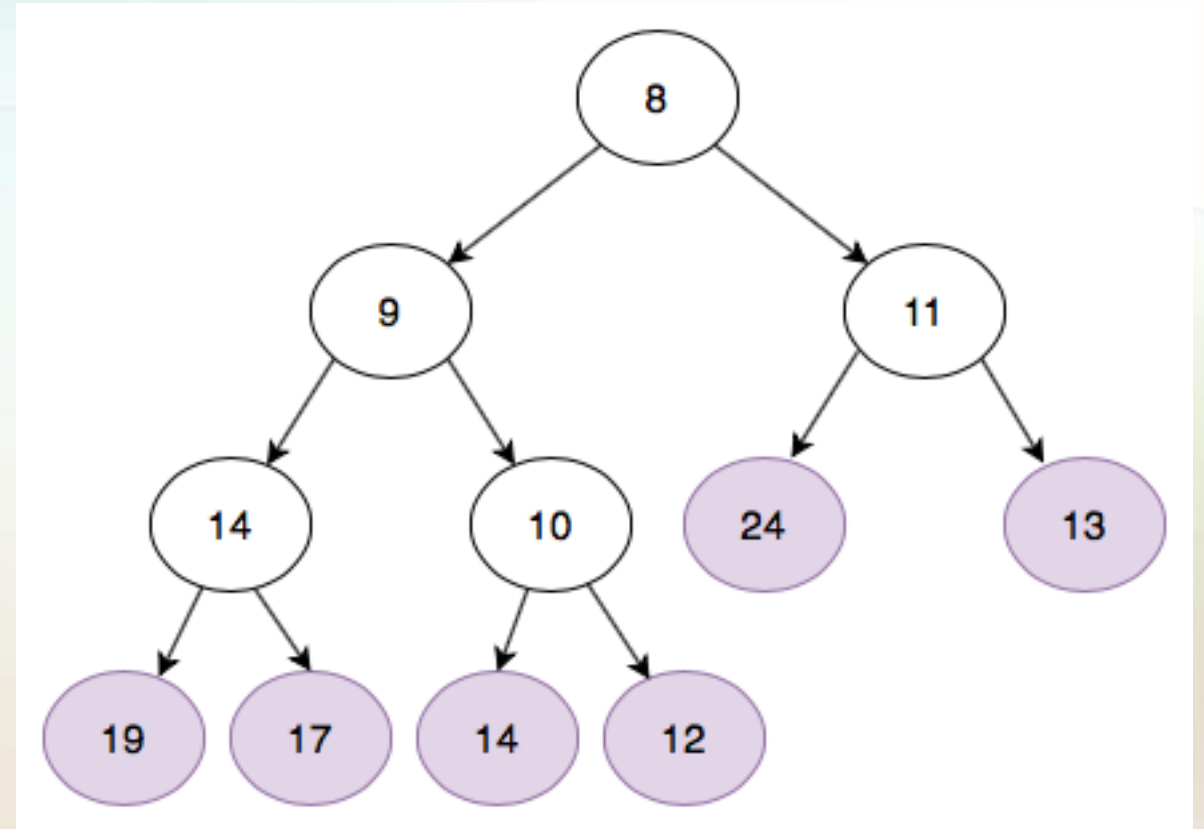
- Čvor 17 je postao list pa se postupak završava.
- Postupak heapify je završen.



Heapify

0 1 2 3 4 5 6 7 8 9 10
[8, 9, 11, 14, 10, 24, 13, 19, 17, 14, 12]

- Kao što se iz primera može videti, dobijena struktura podataka je heap



Napomene

- Šta još važi za heap?
 - Ne moraju svi ključevi heap-a da budu različiti.
 - Heap može da sadrži elemente sa istim prioritetom.
 - Ovo važi i za redove sa prioritetom.
 - Ne mora postojati utvrđen poredak između dece jednog čvora
 - Ne važi pravilo “Levo dete je uvek veće od desnog” ili “Levo dete je uvek manje od desnog”

Zadatak 1

- Napisati klase **UnsortedPriorityQueue** i **SortedPriorityQueue**.

Zadatak 2

- Implementirati **Selection sort** algoritam.
- Složenost?

Zadatak 3

- Implementirati **Insertion sort** algoritam.
- Napomene:
 - Efikasan za sortiranje malih kolekcija
 - Odgovara sortiranju karata
- Složenost?
- U kojim slučajevima je selection sort efikasniji od insertion sort algoritma i obrnuto?

Zadatak 4

- Napisati klasu **Heap**.

Zadatak 5

- Implementirati **Heap sort** algoritam.
- Napomene:
 - Koristi min-heap
 - Složenost?