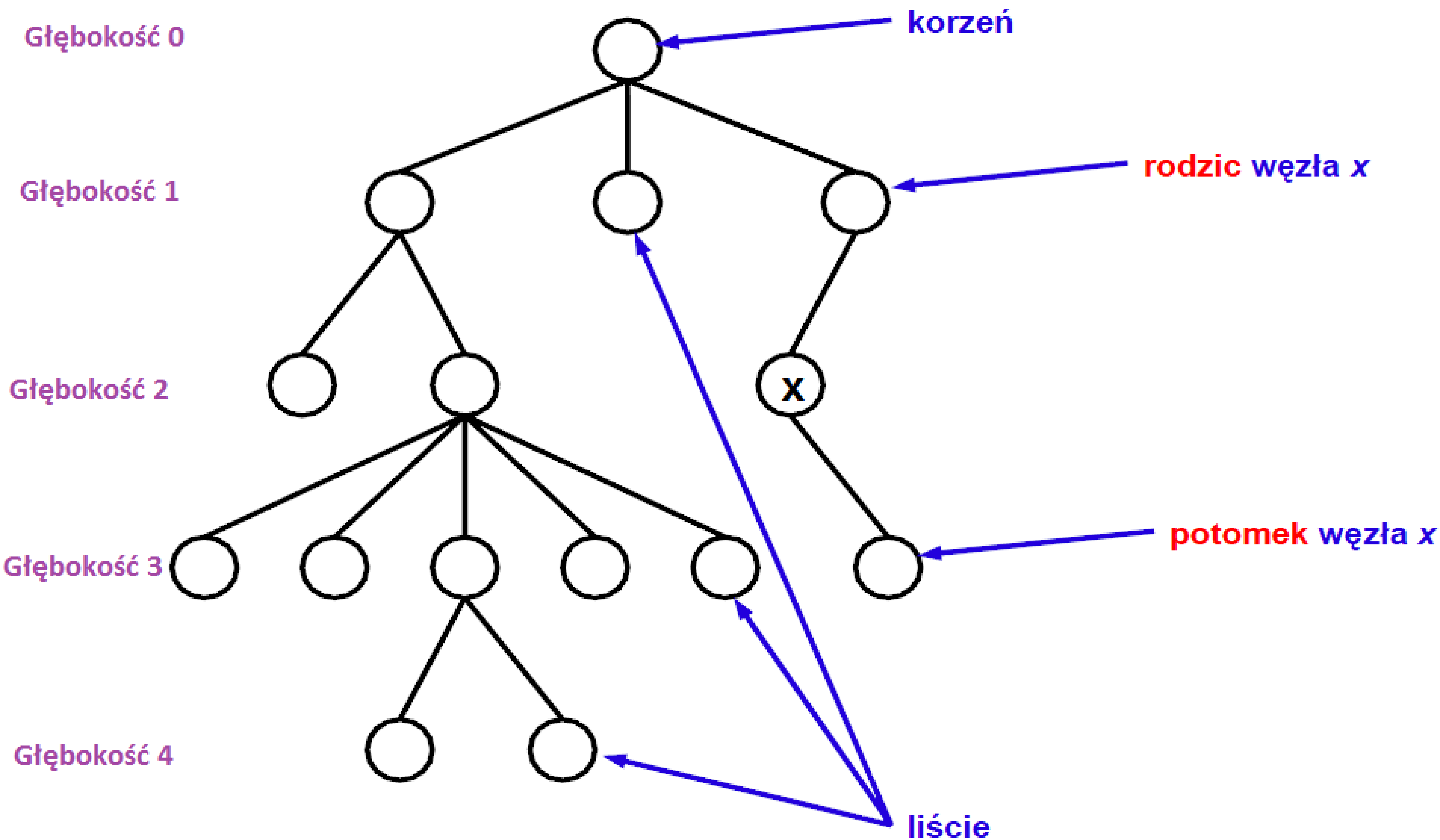
SDIZO-KOPIEC

Drzewo – spójny, acykliczny graf nieskierowany
Drzewo ukorzenione jest strukturą hierarchiczną, tzn. każdy element (węzeł) posiada element(y) podrzędny(e) i/lub element nadrzędny
Element nadrzędny nazywa się **rodzicem (poprzednikiem)**, a podrzędny **potomkiem (nastepnikiem)**. Wymogiem drzewa jest to, że każdy element może posiadać co najwyżej jednego rodzica i (w ogólności) dowolną liczbę potomków (0). Ilość potomków danego węzła nazywa się stopniem węzła
W drzewie wyróżniony jest element zwany **korzeniem**, który nie posiada rodzica (istnieje dokładnie jeden taki element). Z kolei elementy nie posiadające potomków zwane są **liśćmi lub węzłami zewnętrznymi**.

SDIZO-KOPIEC

Drzewo



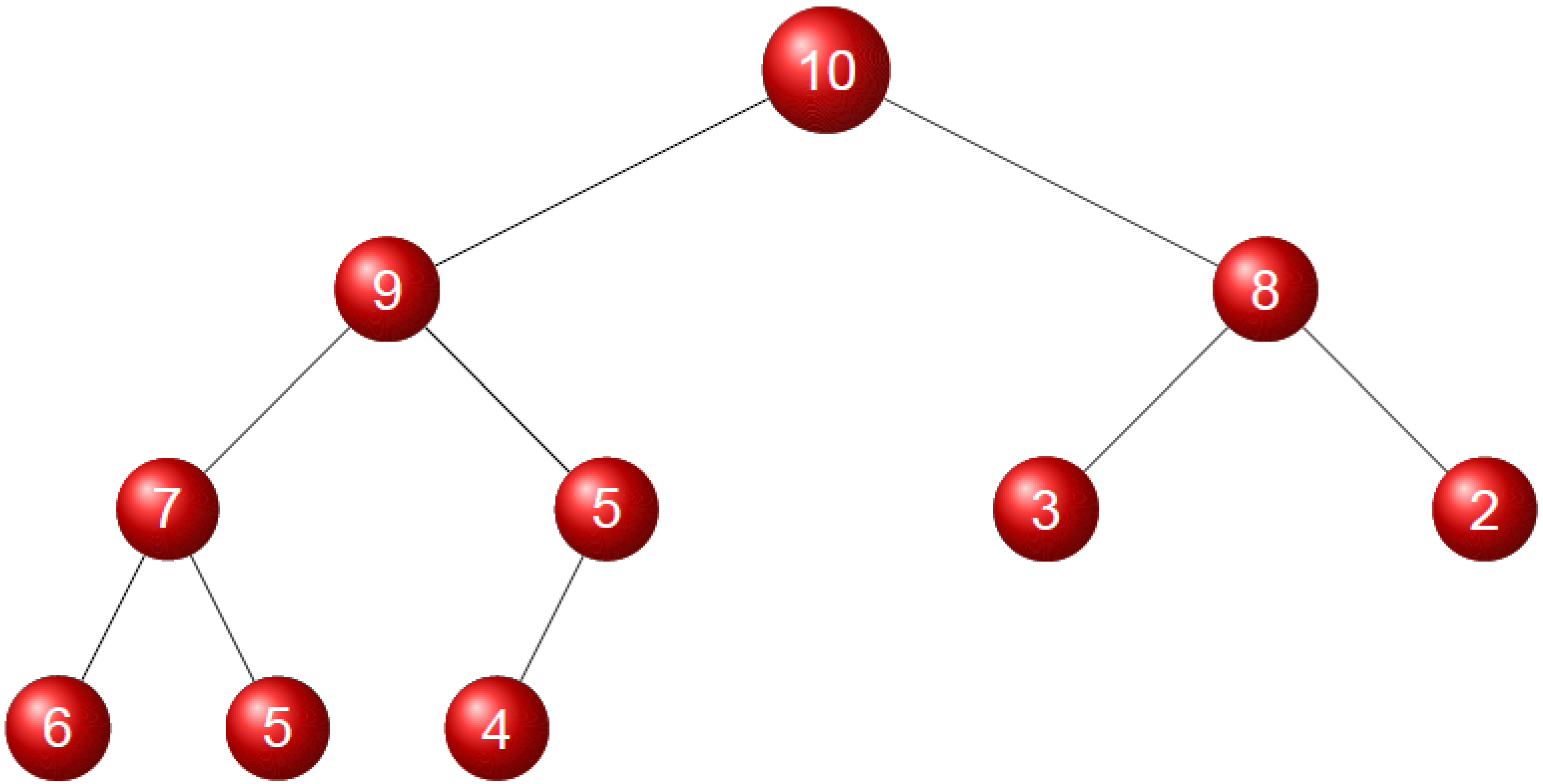
SDIZO-KOPIEC

Drzewo uporządkowane, to takie drzewo w którym następniki każdego z węzłów są uporządkowane
Liczba krawędzi w ścieżce od korzenia do węzła jest nazywana długością – liczba ta określa **poziom (głębokość) węzła**.
Wysokością drzewa jest największy poziom istniejący w drzewie.
Np. korzeń znajduje się na poziomie 0 (głębokości 0)

Szczególnym przypadkiem drzewa ukorzenionego jest drzewo binarne, które charakteryzuje się m.in. tym, że każdy element (nie będący liściem) posiada co najwyżej dwóch potomków.

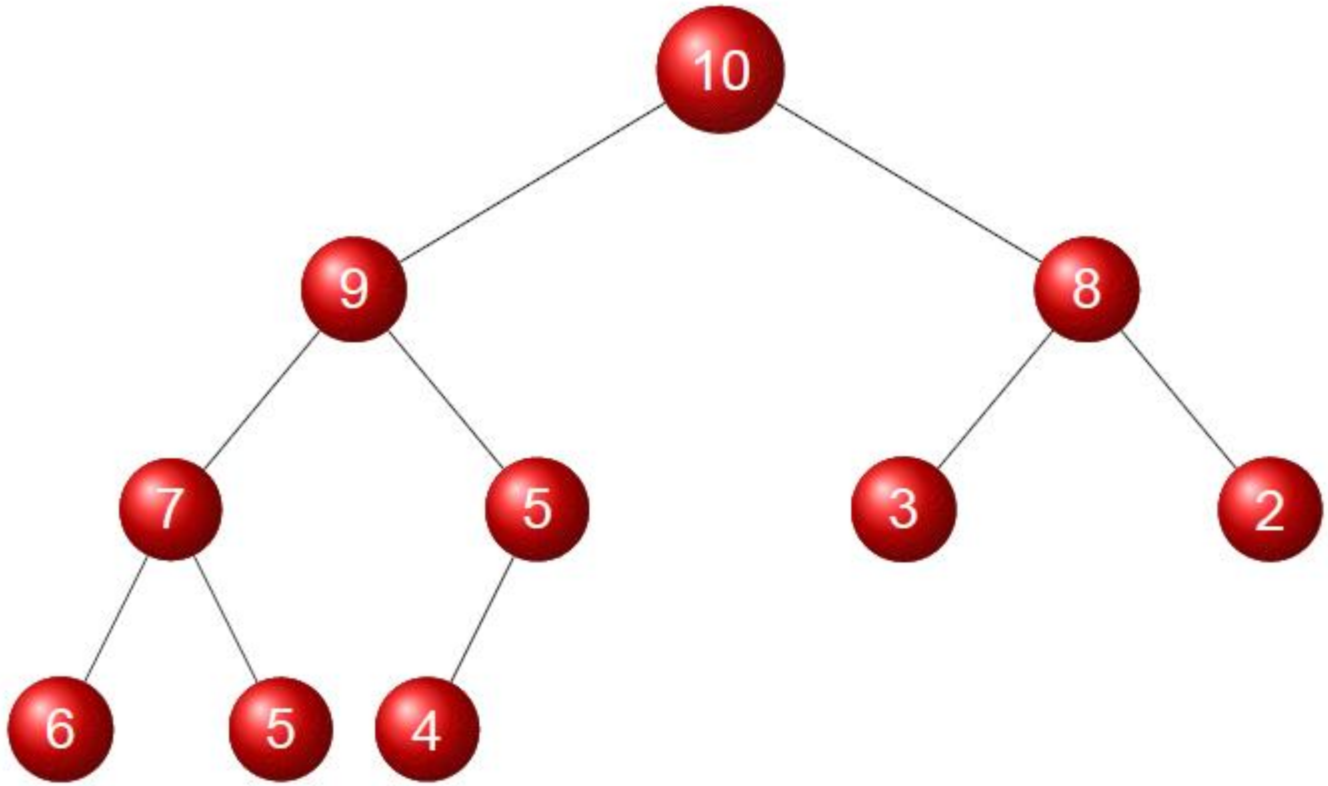
Kopiec - takie drzewo binarne (prawie) pełne , w którym wartość rodzica jest zawsze nie mniejsza (nie większa) od obu potomków (w przypadku jedynego – tylko od niego).
Dzięki temu w korzeniu mamy zawsze element maksymalny (minimalny).

Przykład kopca



Operacje na kopcu – dodawanie (1)

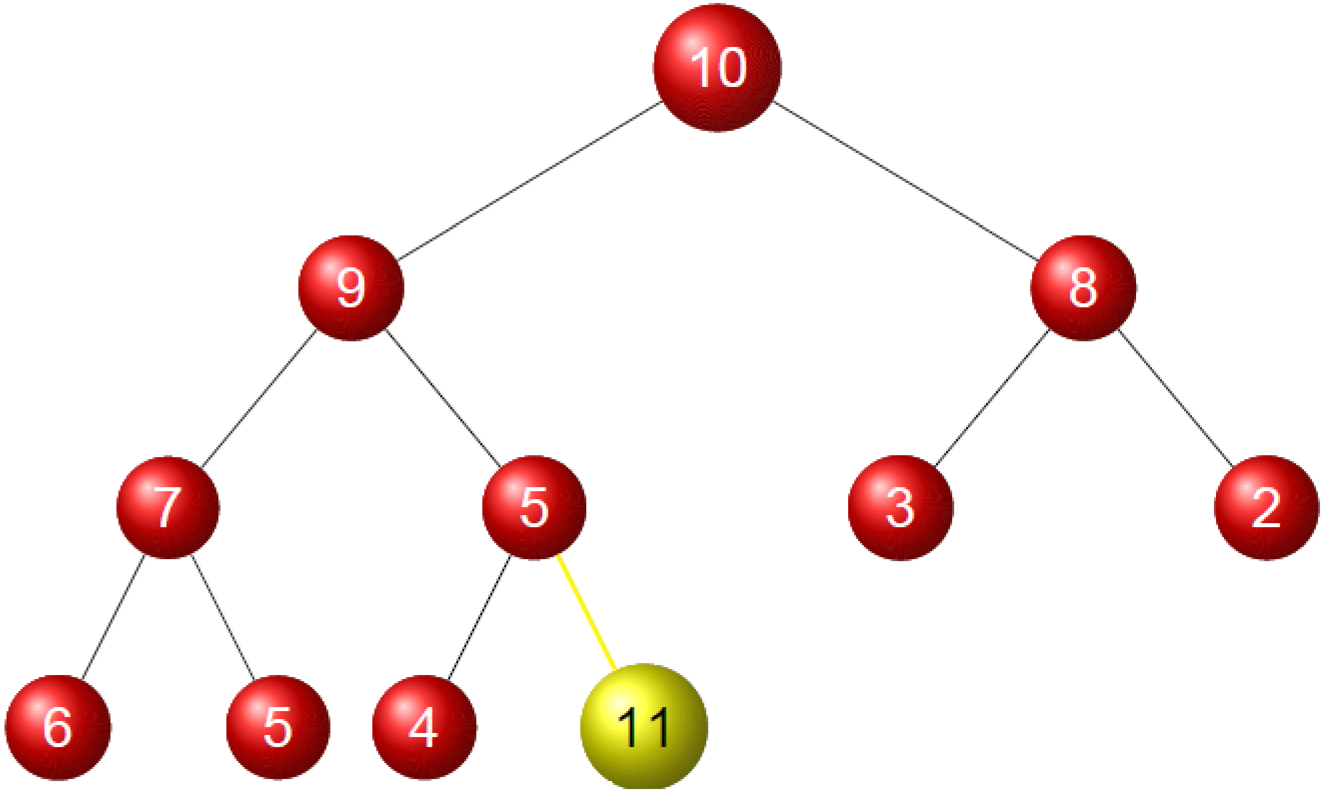
Krok 1: Kopiec początkowy.



Operacje na kopcu – dodawanie (2)

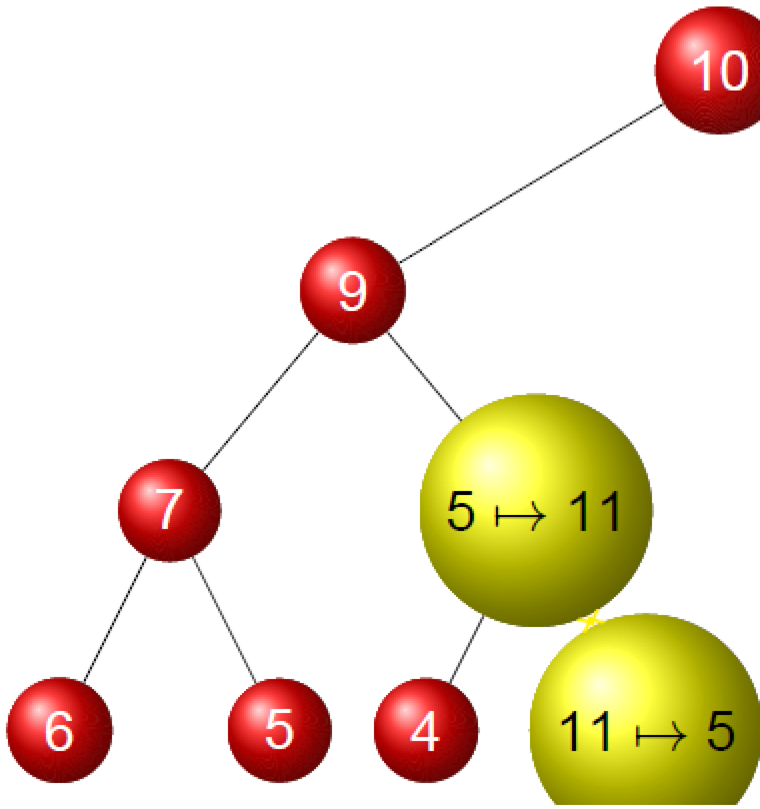
Krok 2:

- Utwórz nowy element x (np. 11) i ustal dane elementarne;
- Dowiąż nowy wierzchołek x do pierwszego z lewej wierzchołka na przedostatnim poziomie drzewa, którego rząd jest < 2.

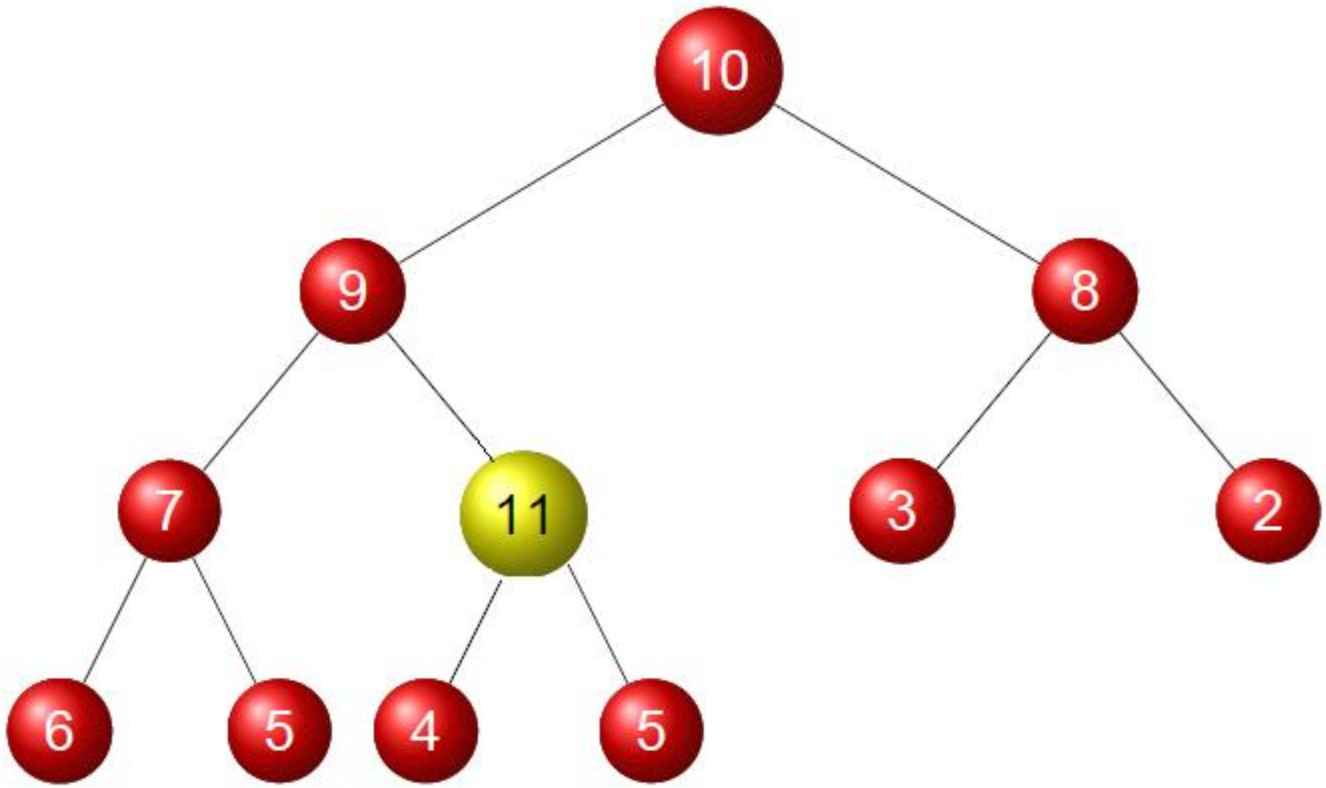


Operacje na kopcu – dodawanie (3)

Krok 3a: Jeżeli tak otrzymane drzewo nie jest częściowo uporządkowane, to przechodząc wzdłuż drogi od liścia *x* do korzenia, poprawić etykiety zamieniając etykietę ojca z etykietą syna, jeśli etykieta ojca jest **większa** niż etykieta syna.

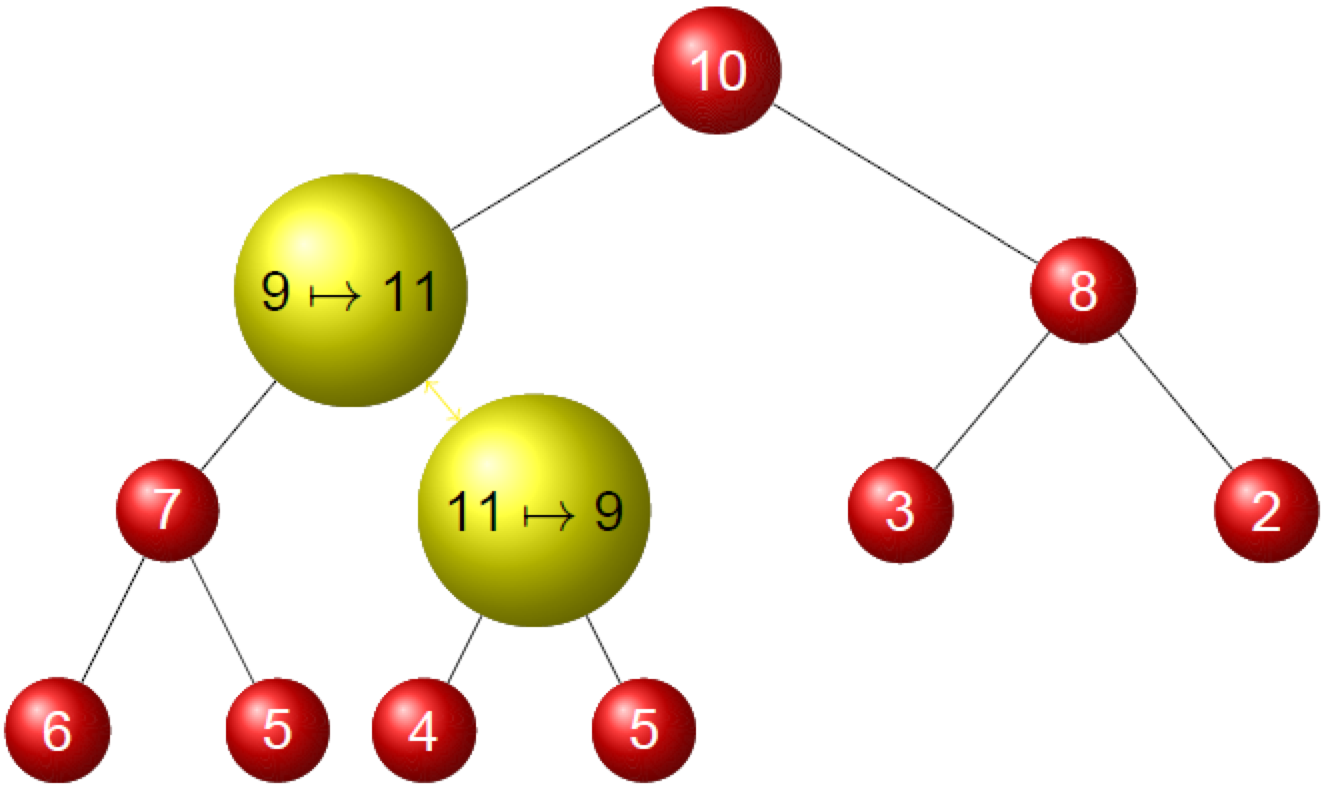


Operacje na kopcu – dodawanie (4)

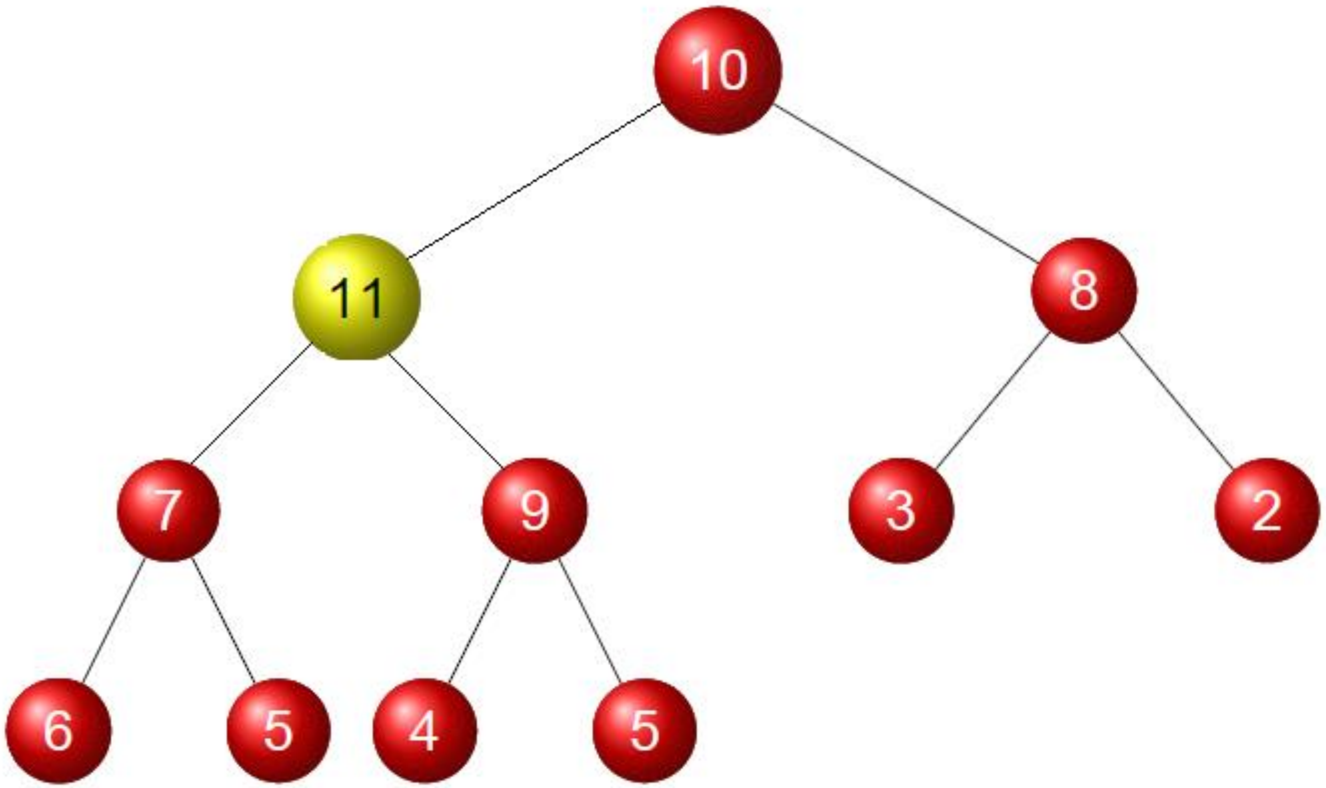


Operacje na kopcu – dodawanie (5)

Krok 3b: Jeżeli tak otrzymane drzewo nie jest częściowo uporządkowane, to przechodząc wzdłuż drogi od liścia *x* do korzenia, poprawić etykiety zamieniając etykietę ojca z etykietą syna, jeśli etykieta ojca jest **większa** niż etykieta syna.

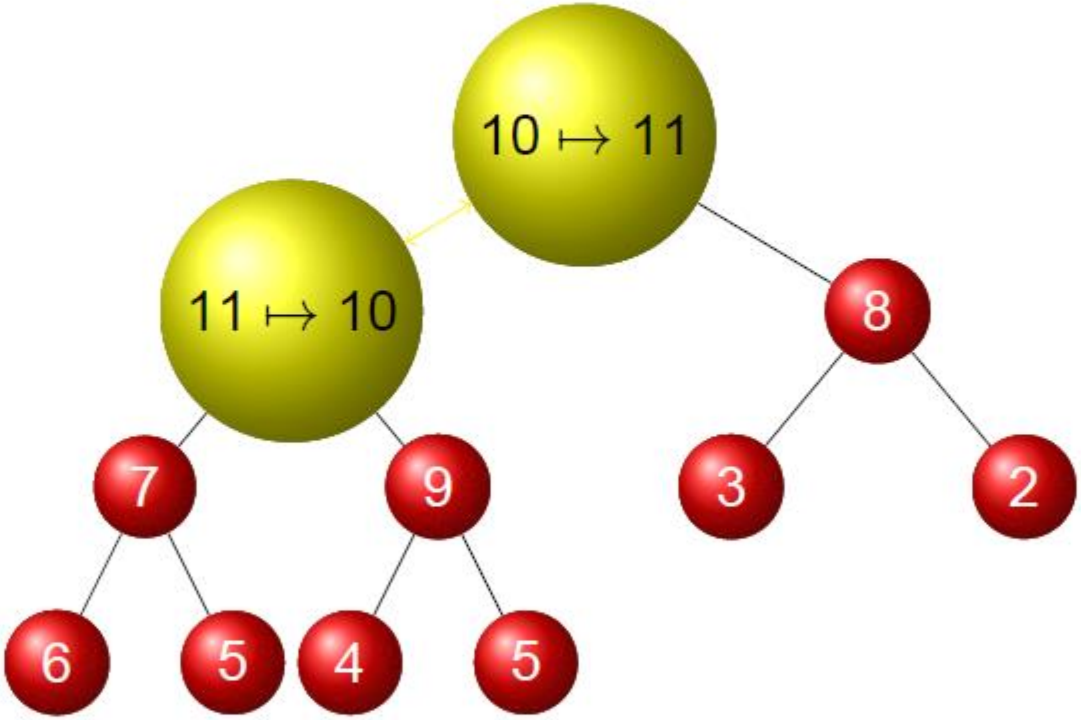


Operacje na kopcu – dodawanie (6)



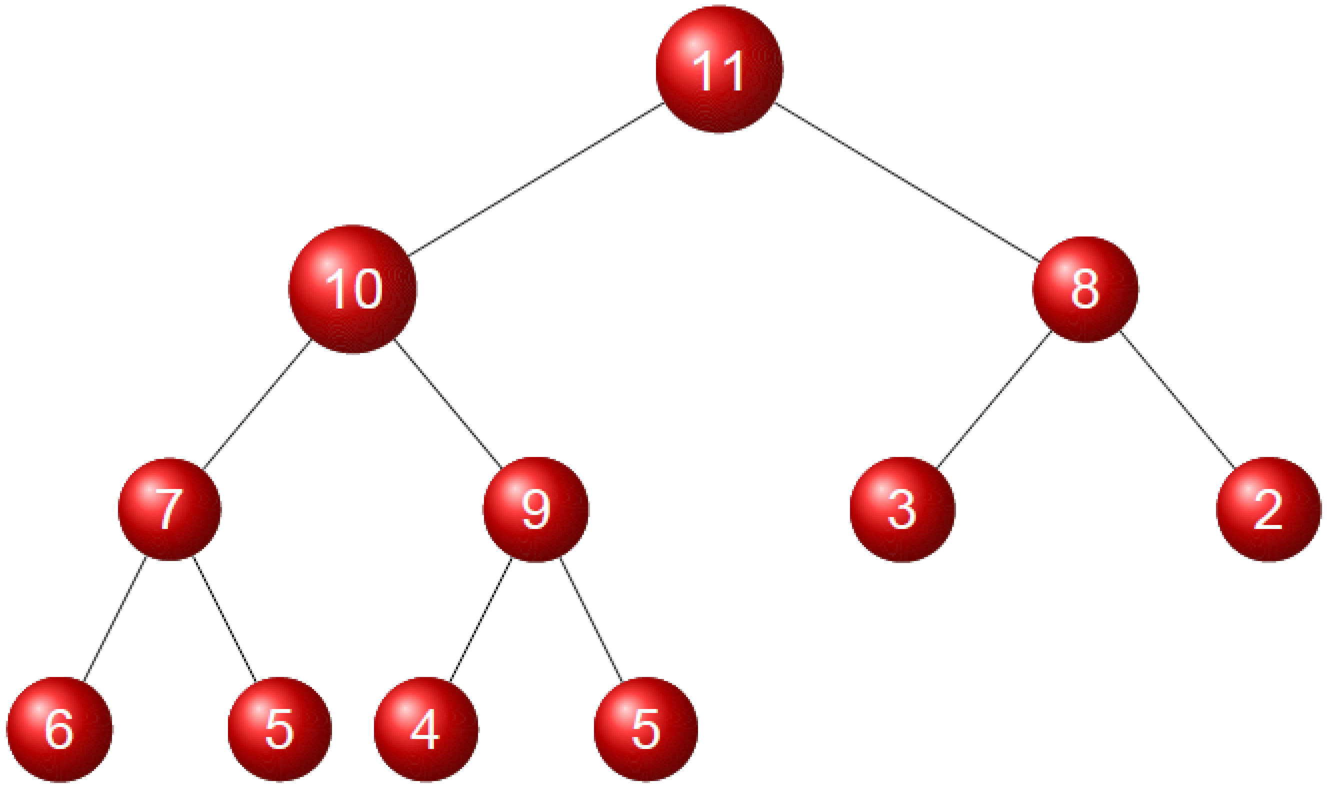
Operacje na kopcu – dodawanie (7)

Krok 3c: Jeżeli tak otrzymane drzewo nie jest częściowo uporządkowane, to przechodząc wzdłuż drogi od liścia *x* do korzenia, poprawić etykiety zamieniając etykietę ojca z etykietą syna, jeśli etykieta ojca jest **większa** niż etykieta syna.



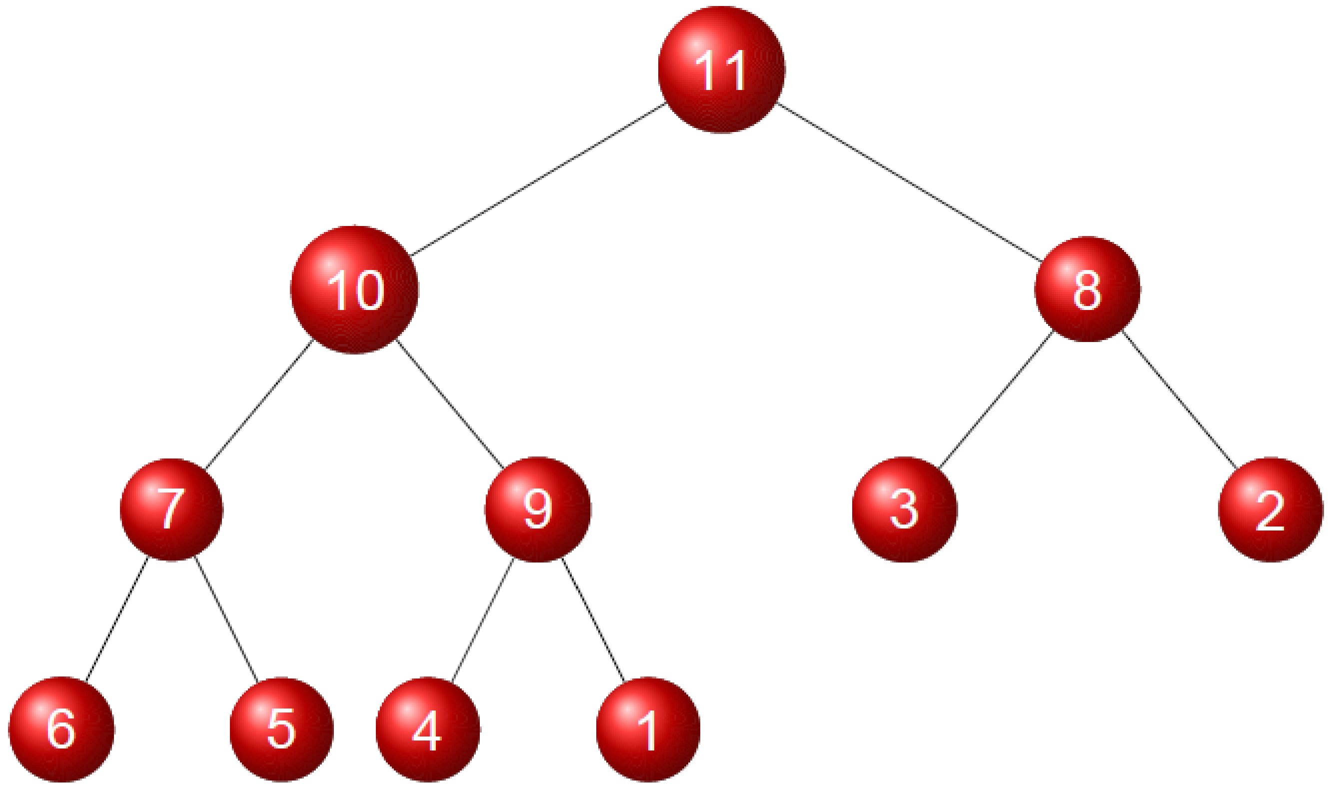
Operacje na kopcu – dodawanie (6)

Efekt końcowy.



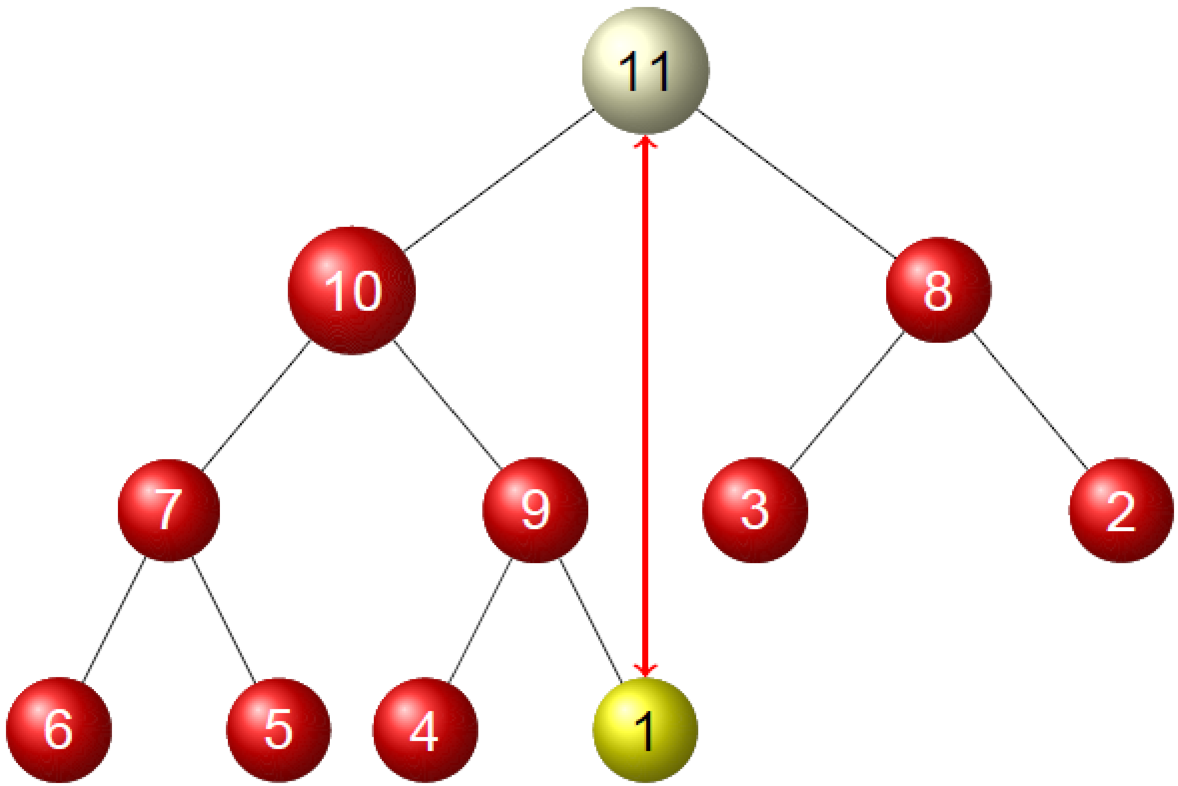
Operacje na kopcu – usuwanie(1)

Kopiec początkowy.



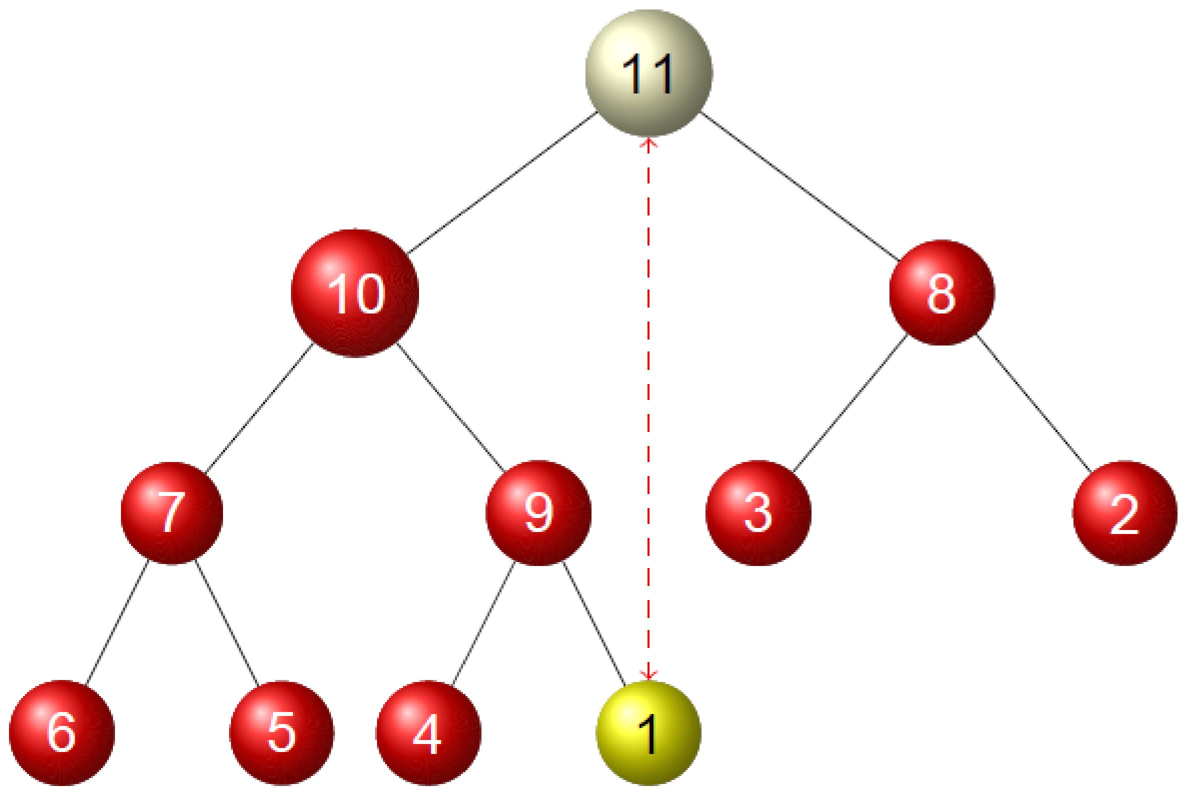
Operacje na kopcu – usuwanie(2)

Krok 1a: Zastąpić etykietę w korzeniu drzewa przez e = 1.



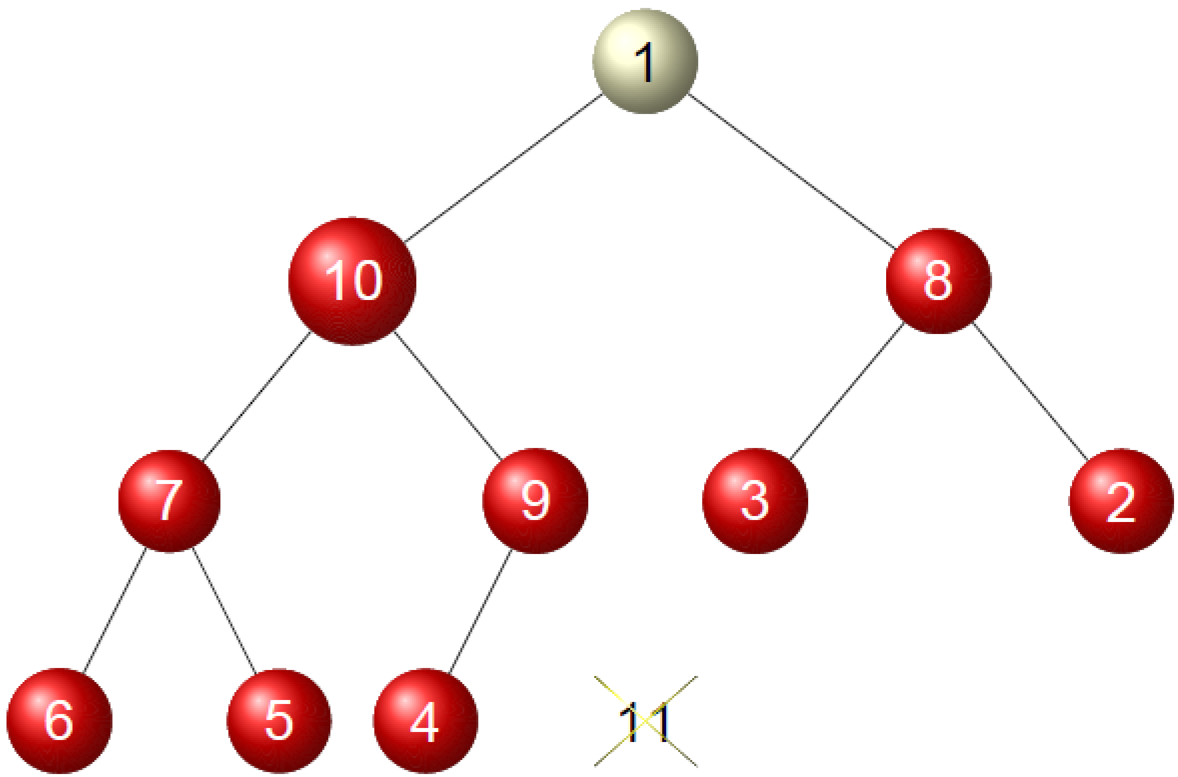
Operacje na kopcu – usuwanie(3)

Krok 1b: Zastąpić etykietę w korzeniu drzewa przez e = 1.



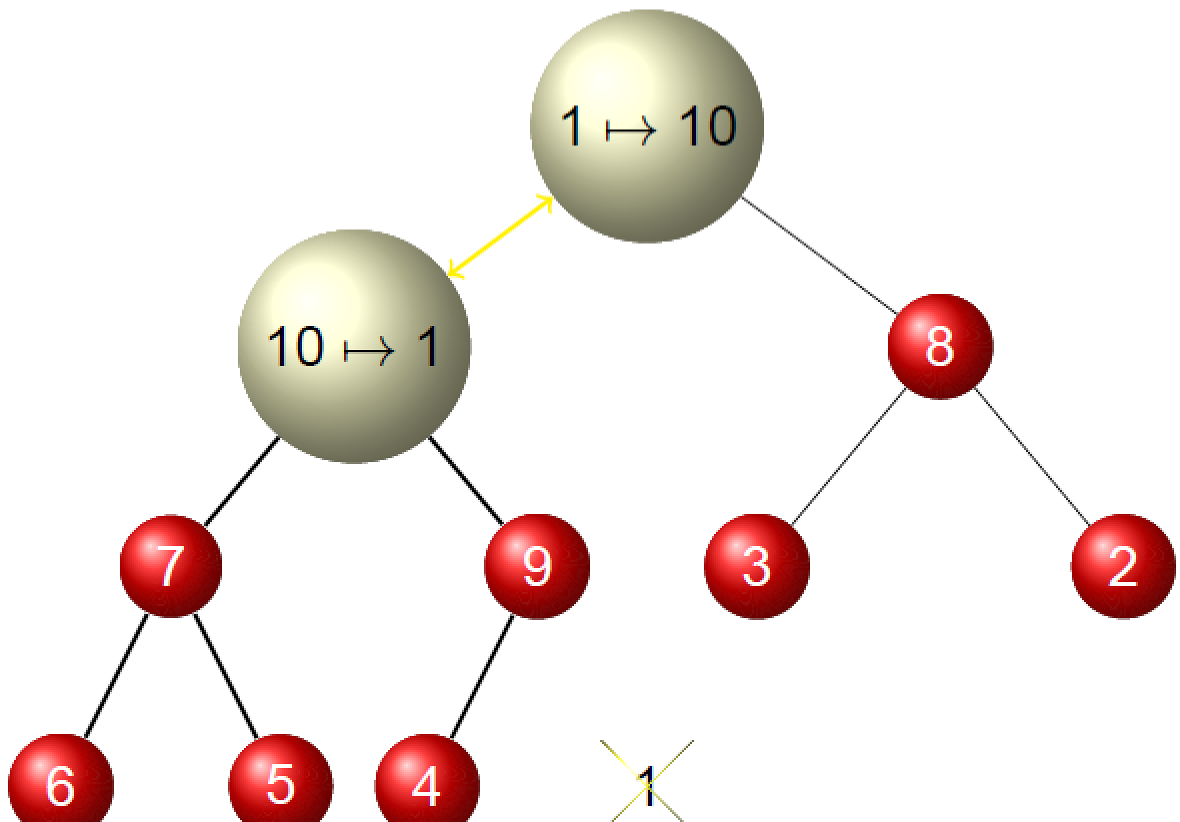
Operacje na kopcu – usuwanie(4)

Krok 2: Usunąć wierzchołek *x* (u nas zawiera on stary korzeń-11) z drzewa.

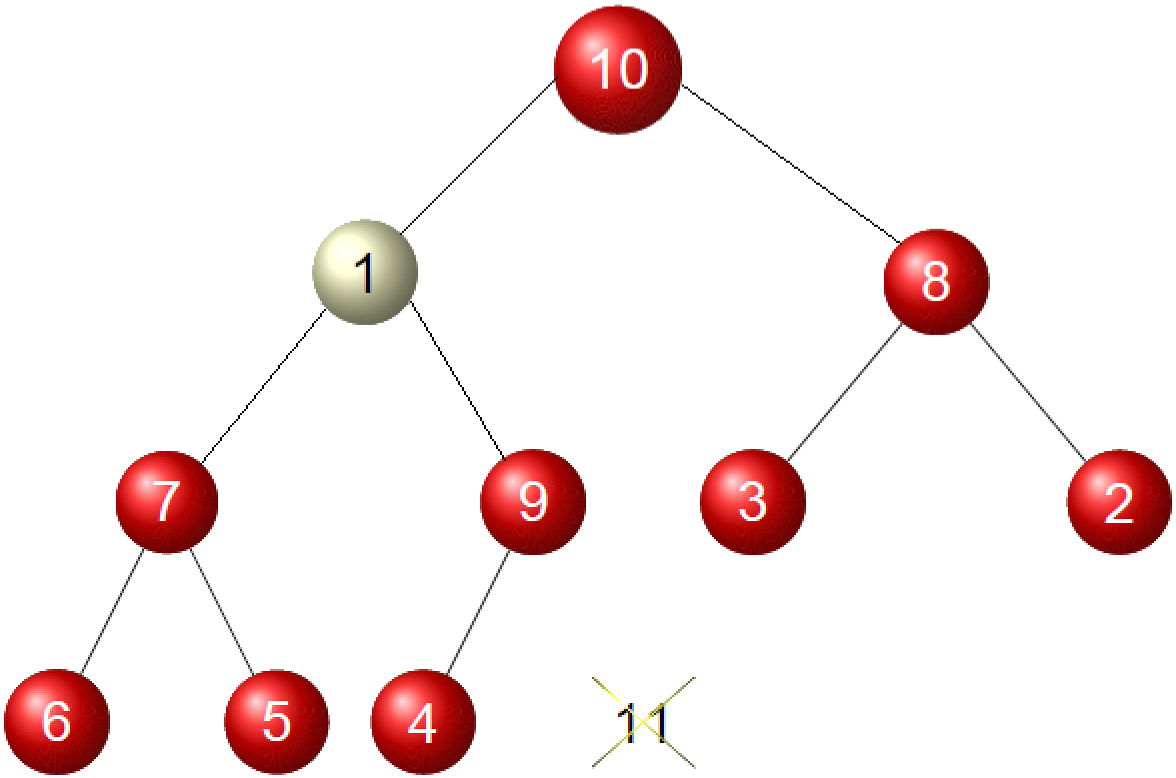


Operacje na kopcu – usuwanie(5)

Krok 3a: Jeśli tak otrzymane drzewo nie jest kopcem, to zaczynając od korzenia i idąc w kierunku liścia, zamieniać etykietę ojca z etykietą tego z jego synów, którego etykieta ma większą wartość, tak długo aż zostanie otrzymane drzewo częściowo uporządkowane.

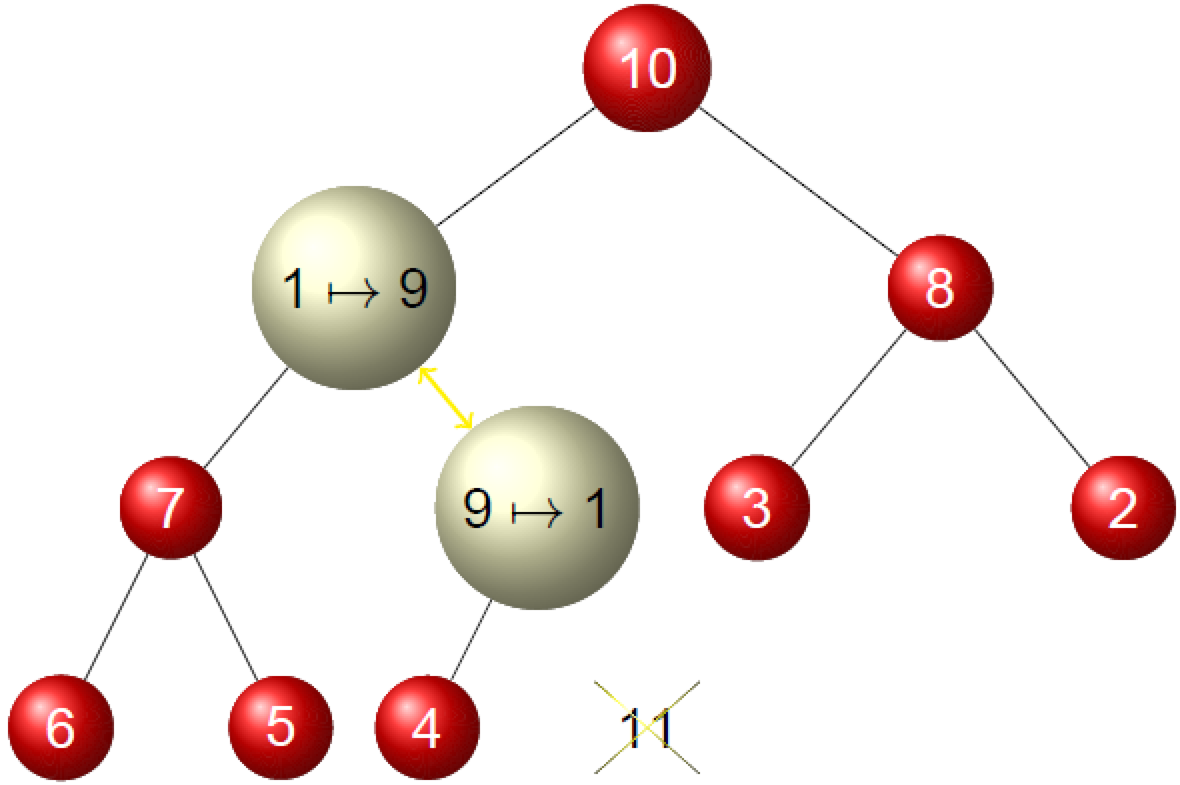


Operacje na kopcu – usuwanie(6)

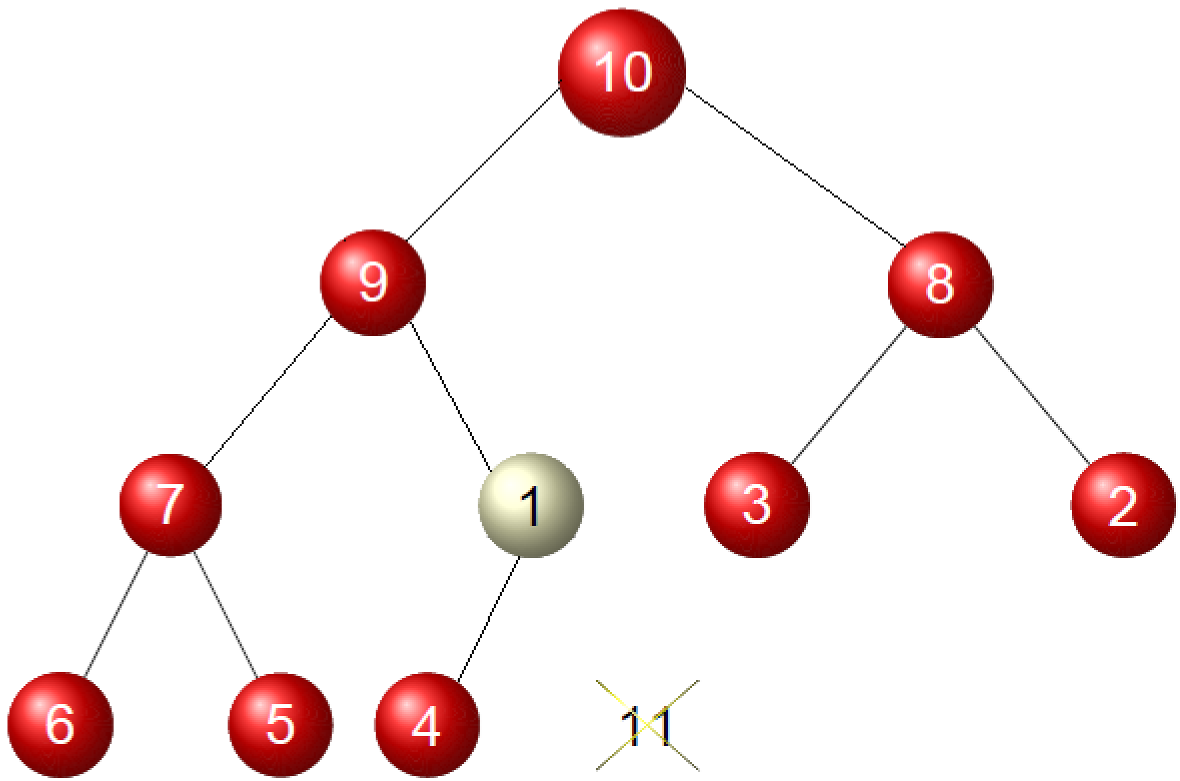


Operacje na kopcu – usuwanie(7)

Krok 3b: Jeśli tak otrzymane drzewo nie jest kopcem, to zaczynając od korzenia i idąc w kierunku liścia, zamieniać etykietę ojca z etykietą tego z jego synów, którego etykieta ma większą wartość, tak długo aż zostanie otrzymane drzewo częściowo uporządkowane.

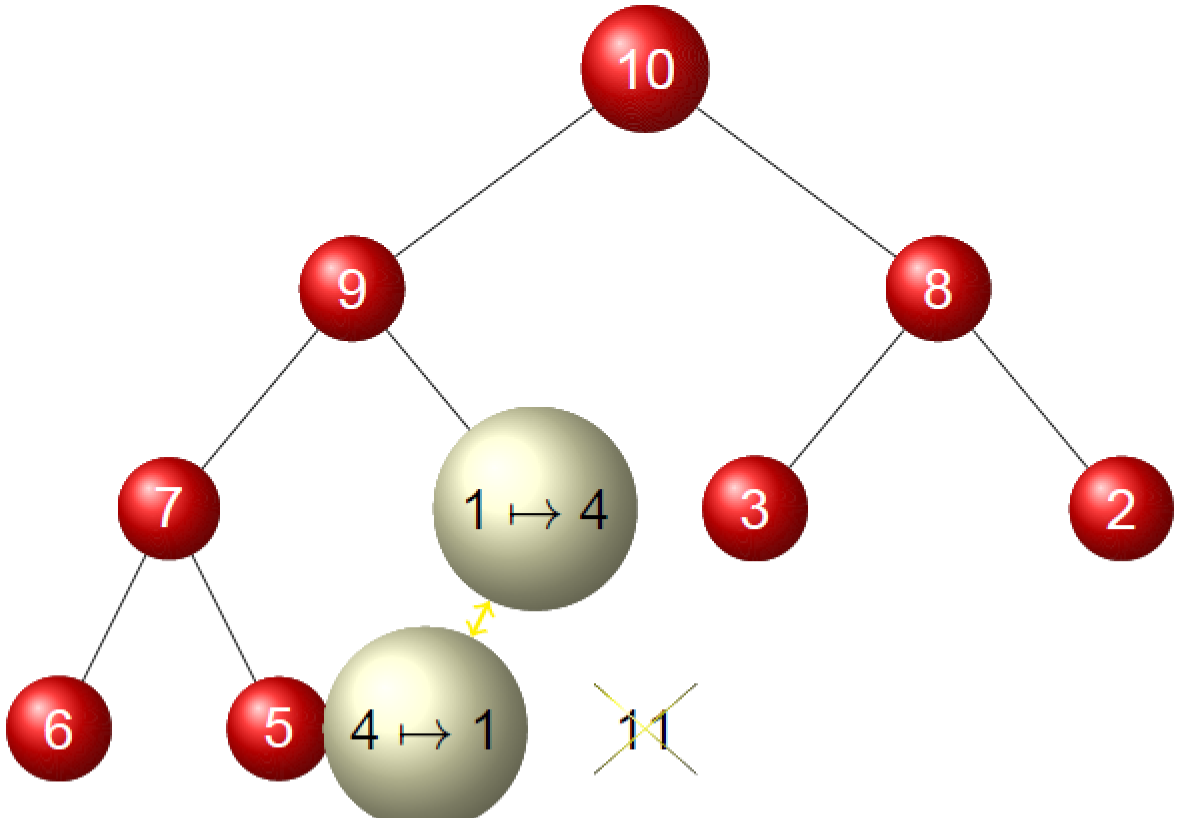


Operacje na kopcu – usuwanie(8)

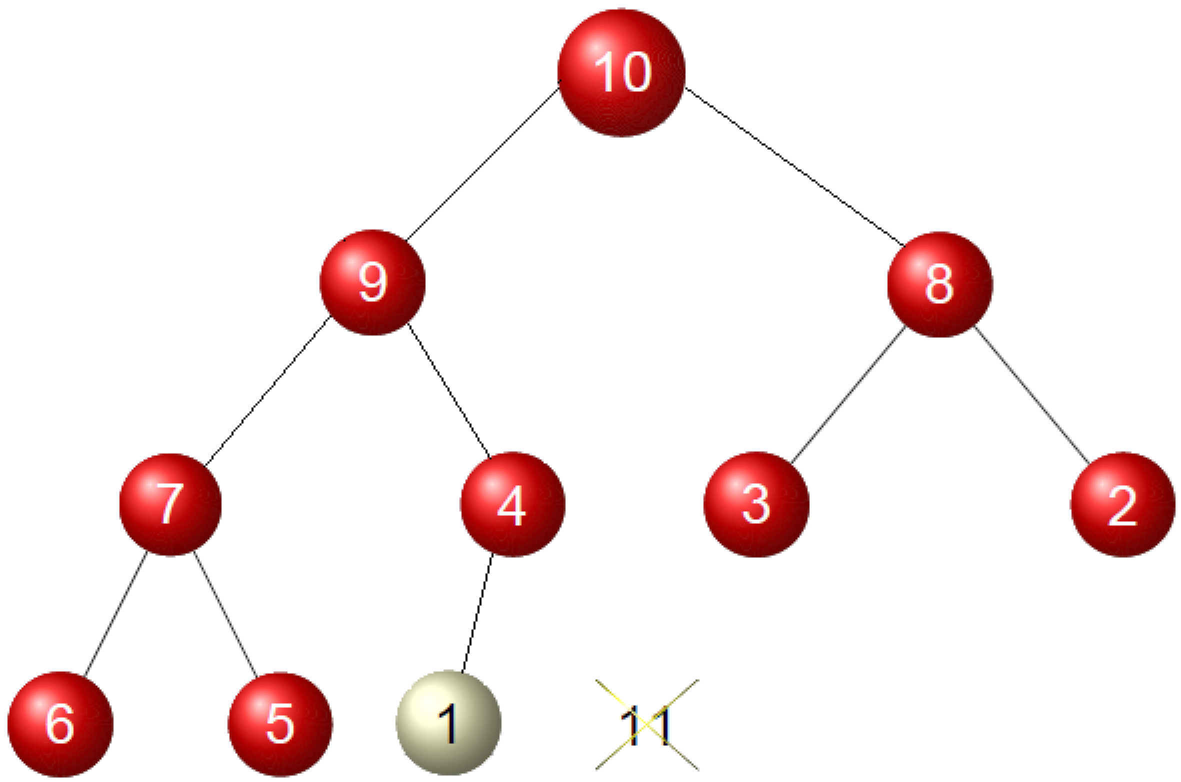


Operacje na kopcu – usuwanie(9)

Krok 3c: Jeśli tak otrzymane drzewo nie jest kopcem, to zaczynając od korzenia i idąc w kierunku liścia, zamieniać etykietę ojca z etykietą tego z jego synów, którego etykieta ma większą wartość, tak długo aż zostanie otrzymane drzewo częściowo uporządkowane.

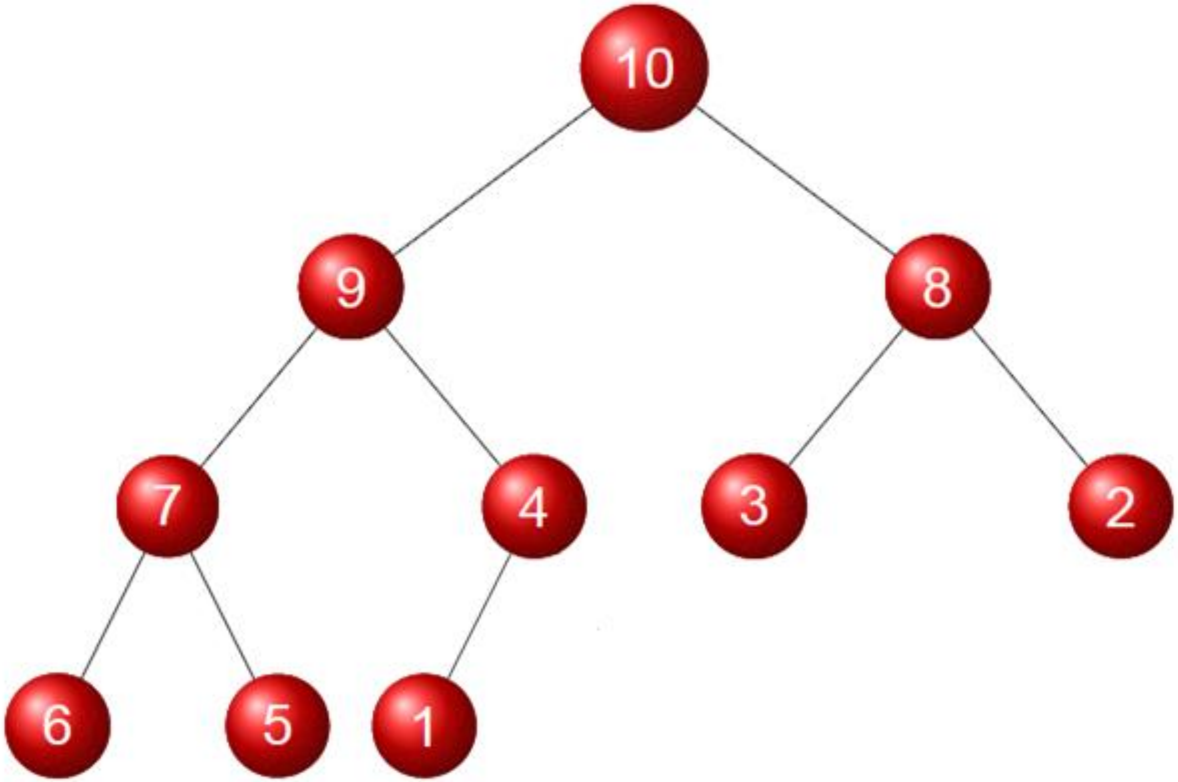


Operacje na kopcu – usuwanie(10)



Operacje na kopcu – usuwanie(11)

Efekt końcowy.



Operacje na kopcu – złożoność

Ilość elementów w pełnym drzewie binarnym:

$n = 2^0 + 2^1+2^2+..+2^h$ <-ciąg geometryczny

$n= 2^{h+1} – 1 \Rightarrow 2^{h+1} = n+1 \Rightarrow h=\log_2((n+1)/2)$

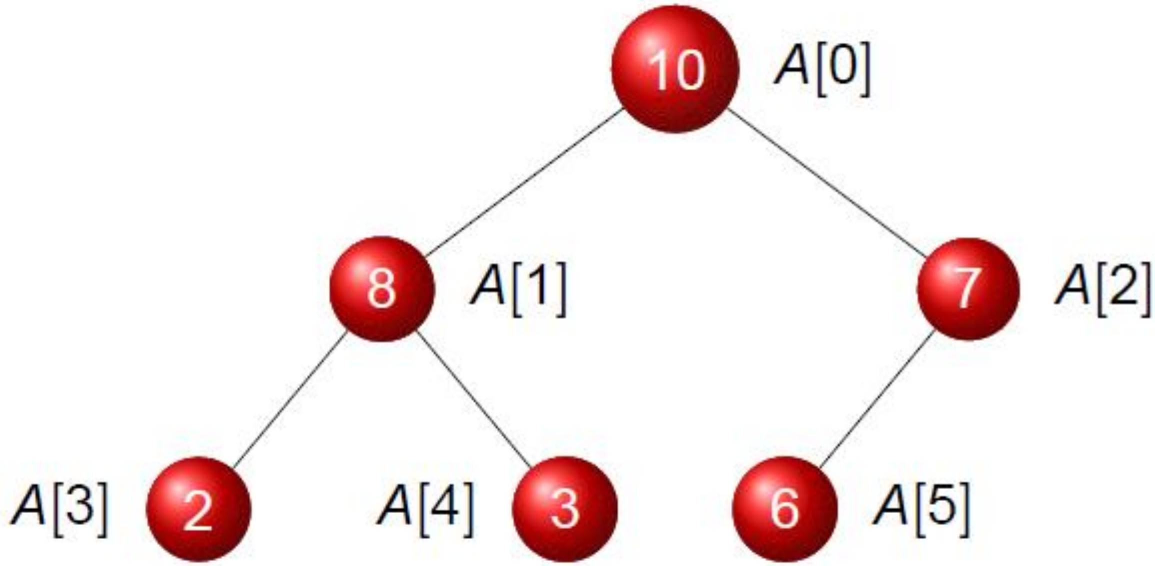
Złożoność operacji dodawania i usuwania (korzenia) wynosi

$$O(\log n)$$

Tworzenie drzewa $\Rightarrow$ O(*n*log*n*), a po dokładniejszej analizie O(n).

Wyszukiwanie $\Rightarrow$ O(*n*log*n*) – bo musimy usunąć wszystkie elementy aż do znalezienia (lub do końca), a później odbudować (po dokładniejszej analizie O(n))

Operacje na kopcu – tablicowa implementacja



$\lfloor x \rfloor$ - część całkowita z x

Tablica A:

0	1	2	3	4	5
10	8	7	2	3	6

Indeks rodzica: $\lfloor (p-1)/2 \rfloor$

Indeks potomka lewego: 2r+1

Indeks potomka prawego: 2r+2

Zachodzi $A[i] \geq A[2i+1]$ oraz $A[i] \geq A[2i+2]$

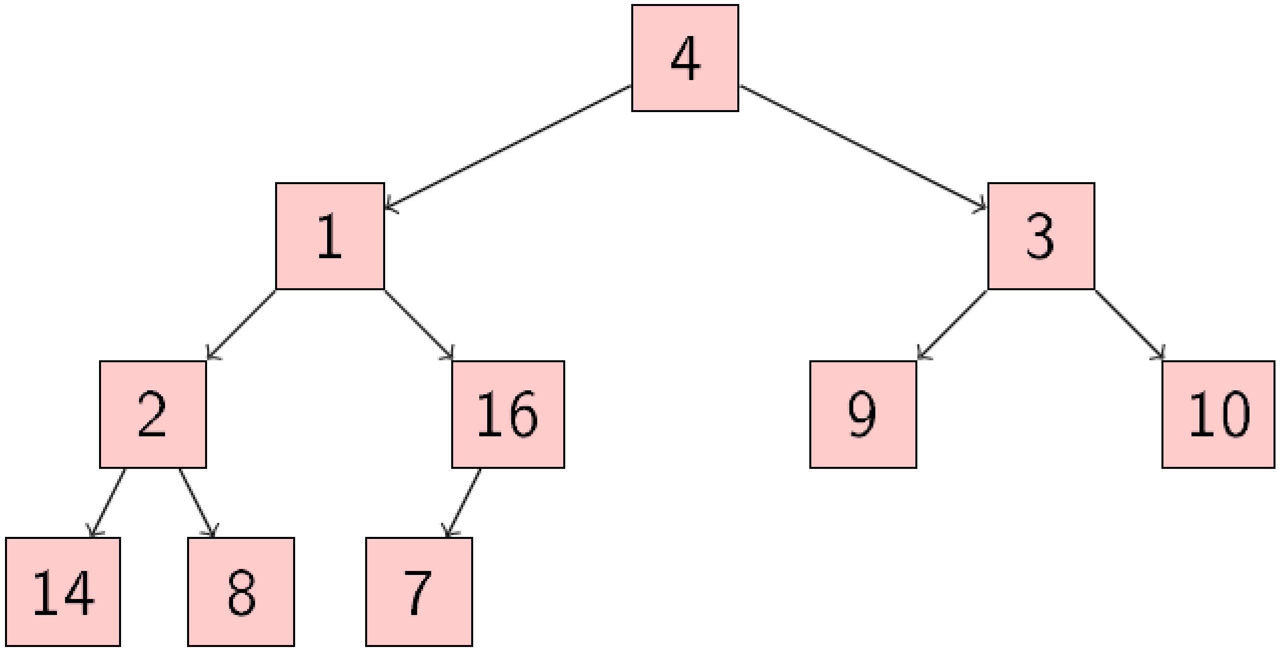
Operacje na kopcu – funkcje

tab – tablica, gdzie przechowany jest kopiec
len_tab – długość tablicy (ilość elem. kopca)
index – indeks elementu od którego zacznie się operacja naprawa kopca

funkcja naprawy kopca w dół:
void heap_fix_down(int tab[], int index, int len_tab);

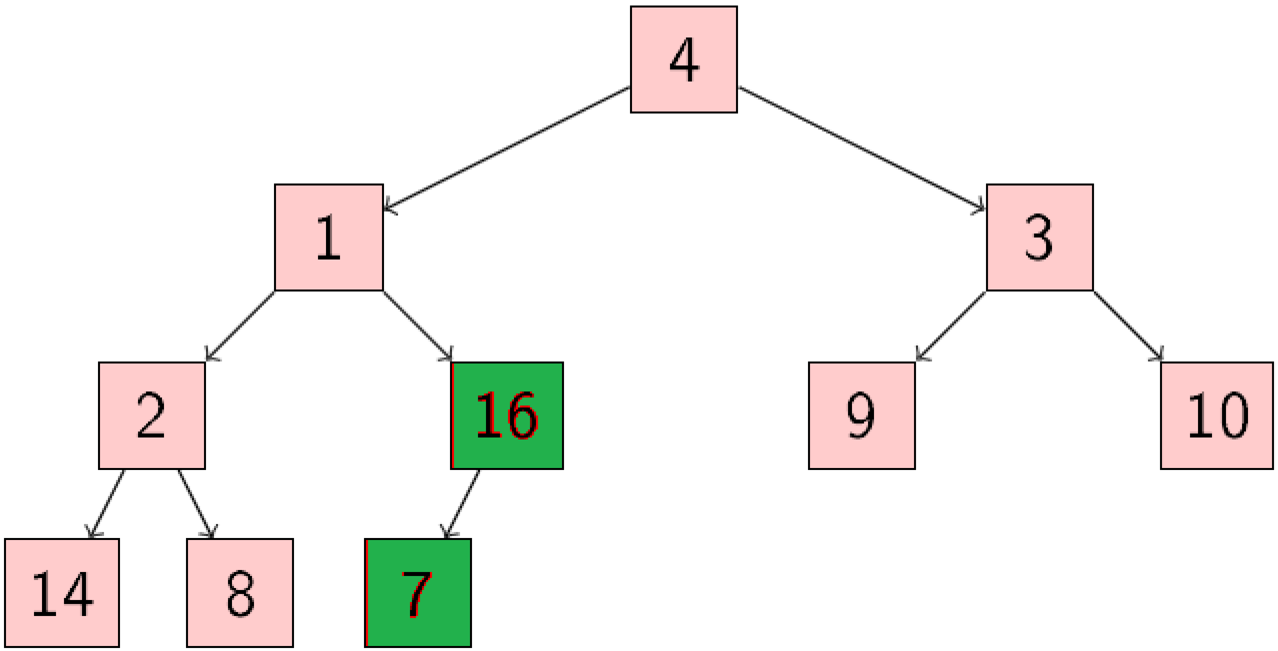
funkcja naprawy kopca w górę:
void heap_fix_up(int tab[], int index);

Operacje na kopcu – tworzenie kopca (1)
Algorytm Floyda



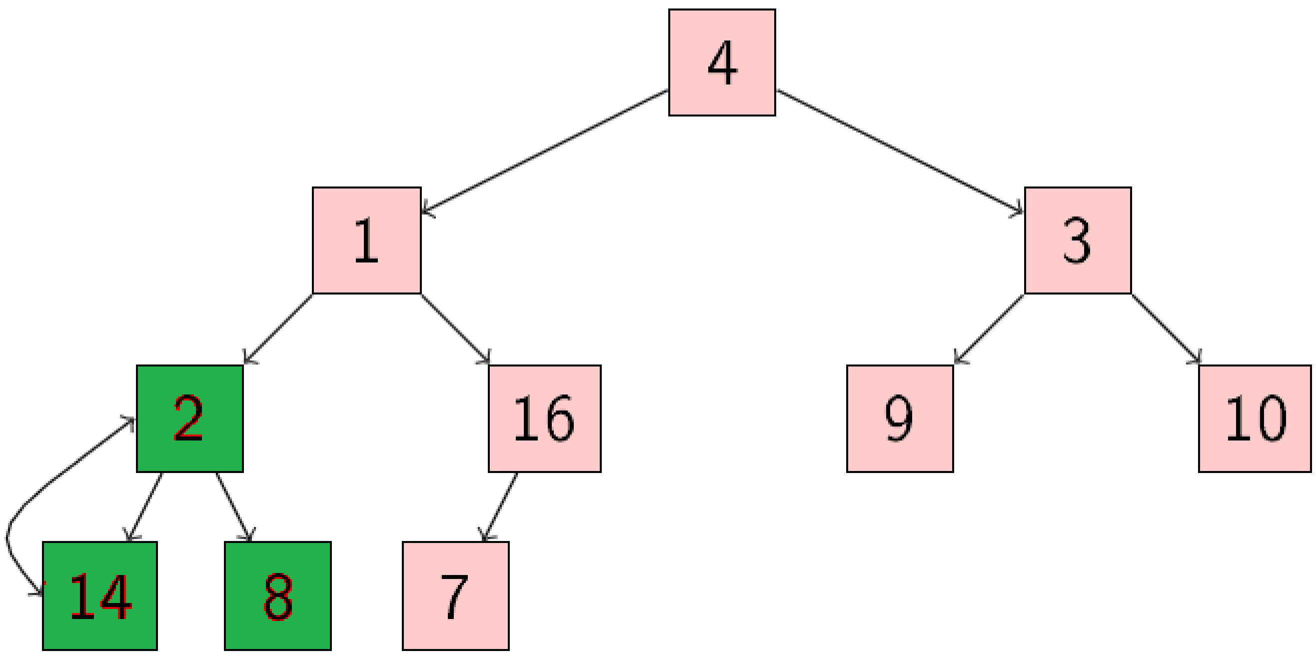
4	1	3	2	16	9	10	14	8	7
0	1	2	3	4	5	6	7	8	9

Operacje na kopcu – tworzenie kopca (2)



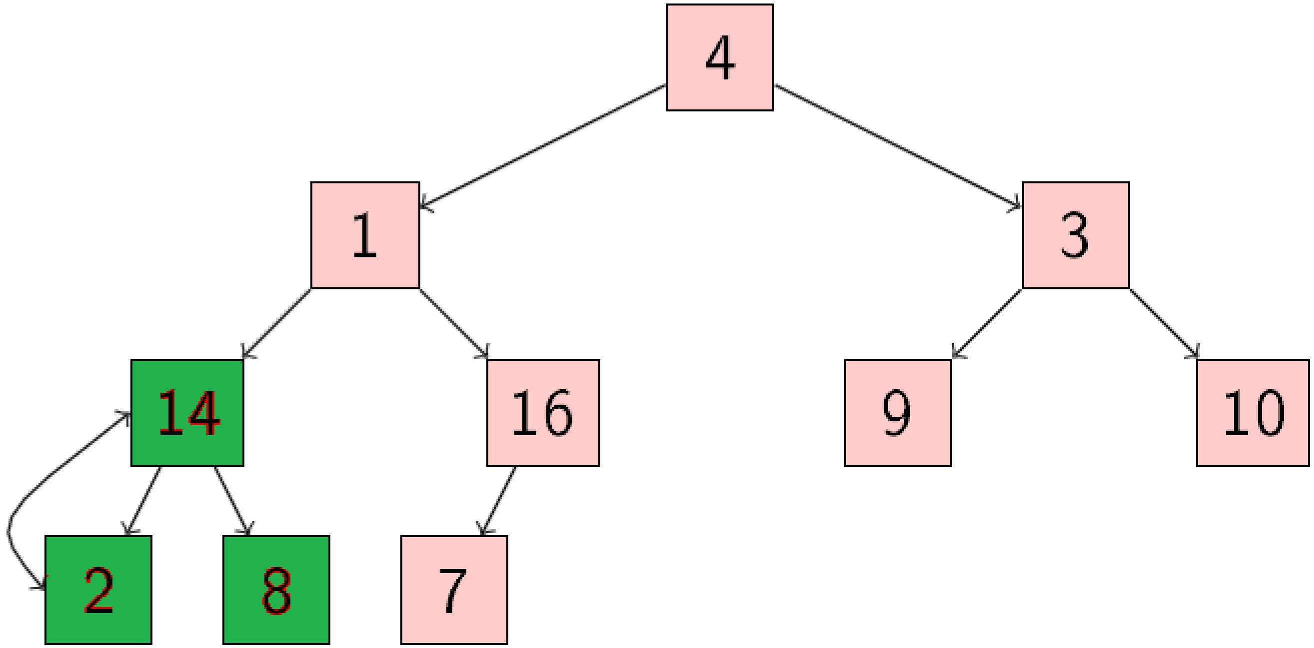
4	1	3	2	16	9	10	14	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (3)



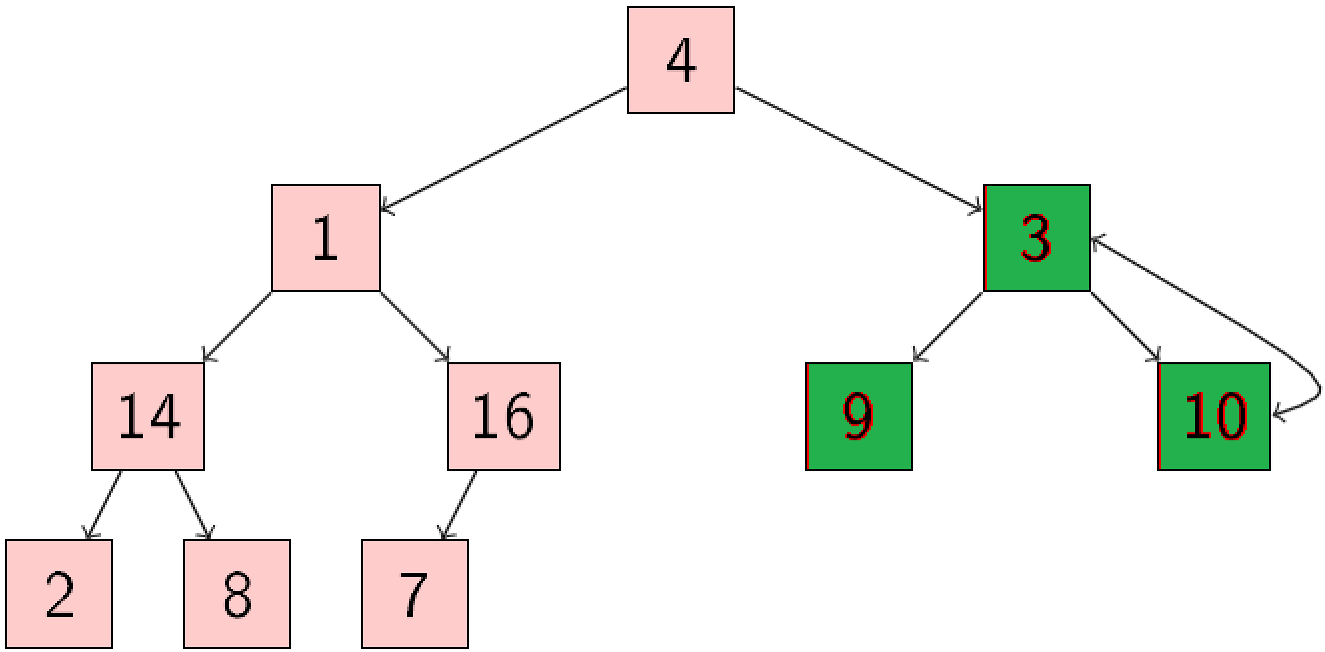
4	1	3	2	16	9	10	14	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (4)



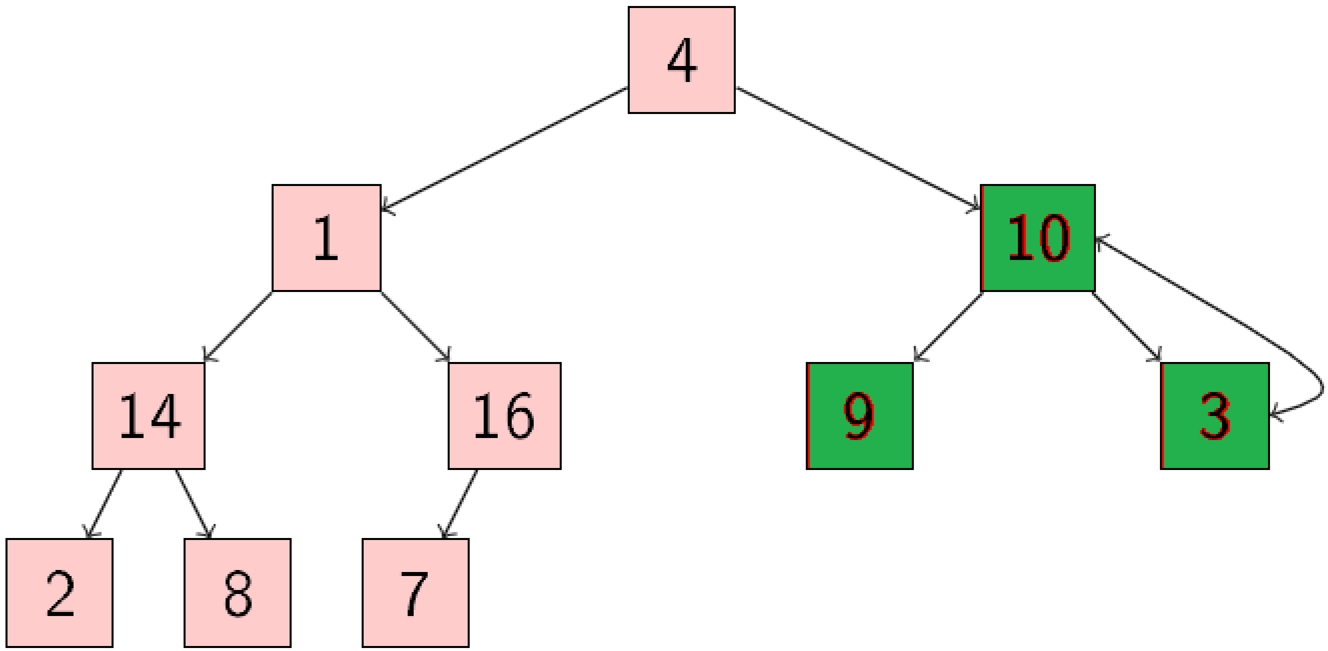
4	1	3	14	16	9	10	2	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (5)



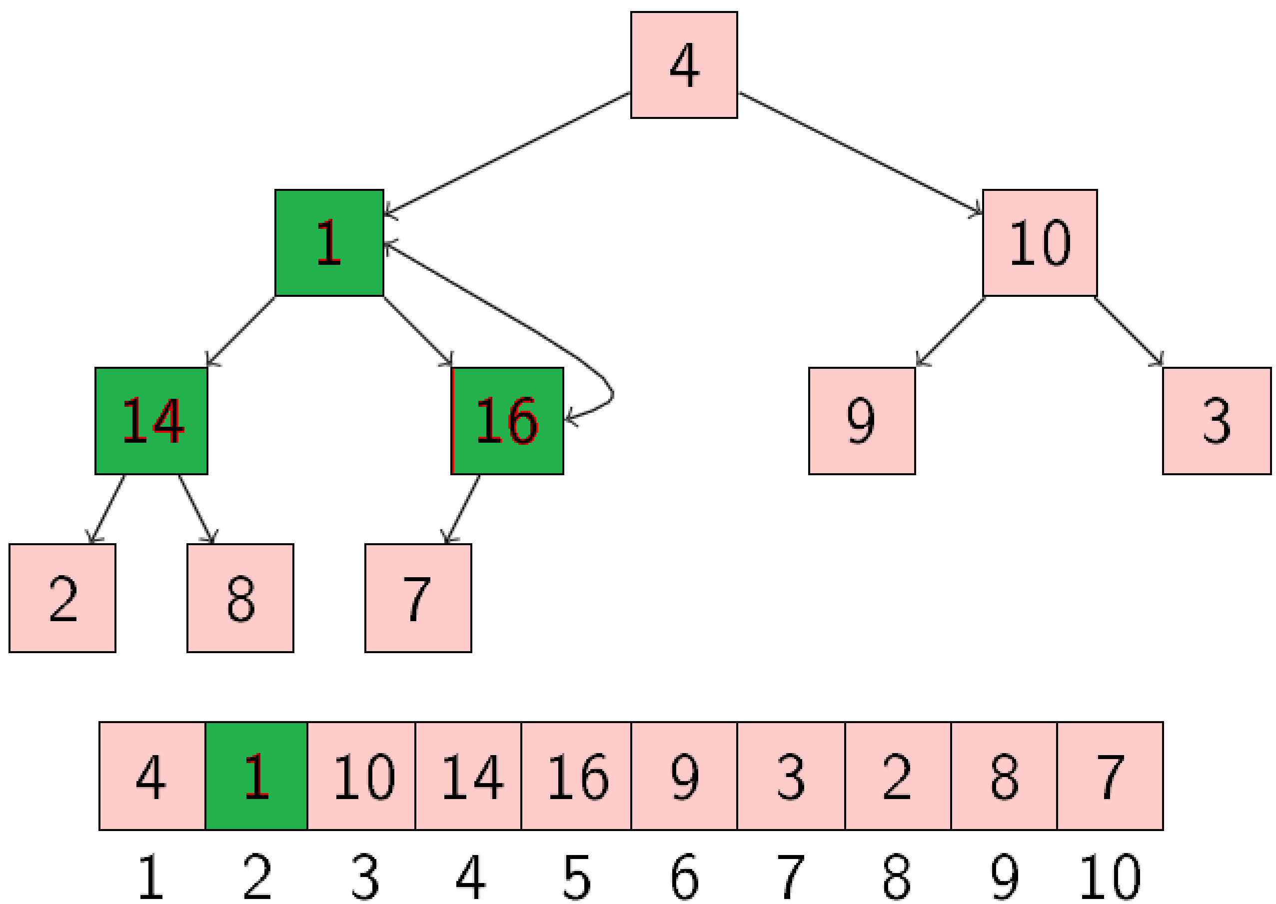
4	1	3	14	16	9	10	2	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (6)

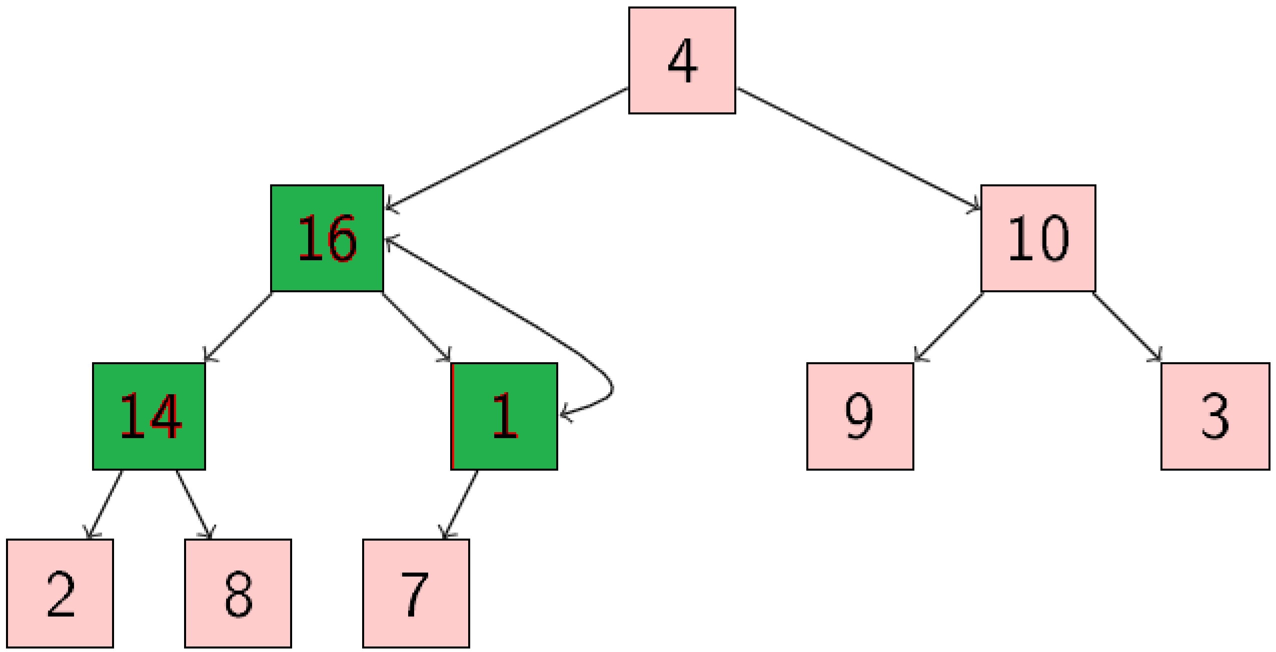


4	1	10	14	16	9	3	2	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (7)

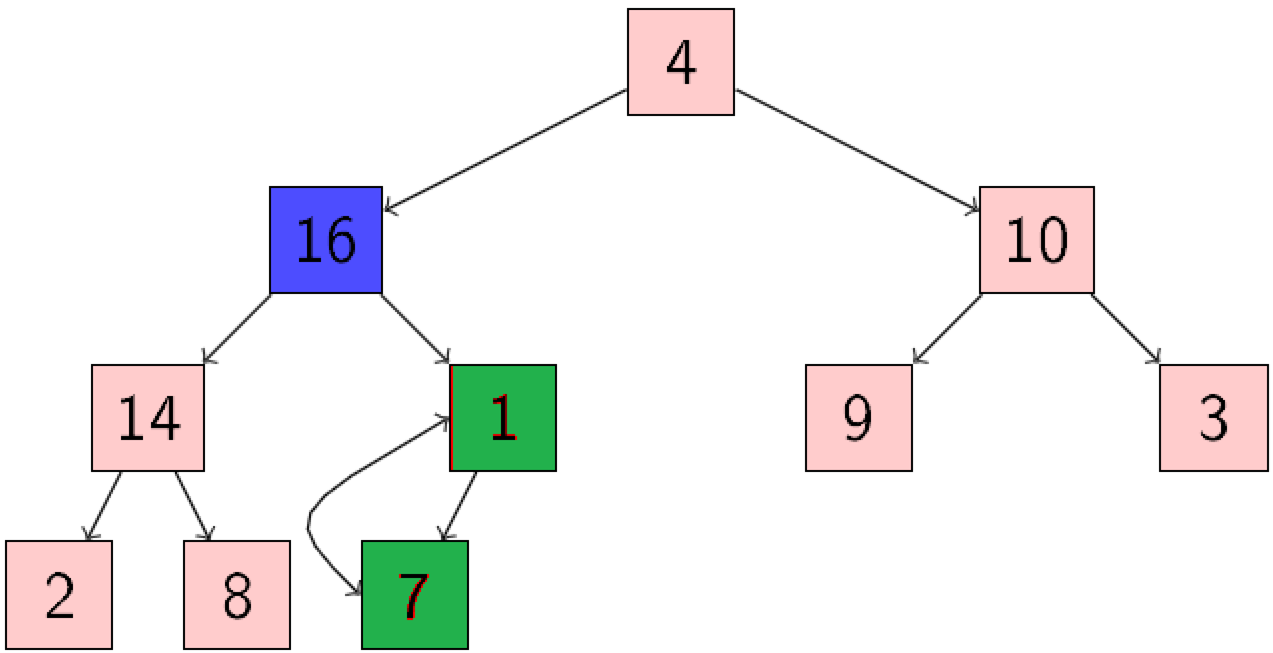


Operacje na kopcu – tworzenie kopca (8)



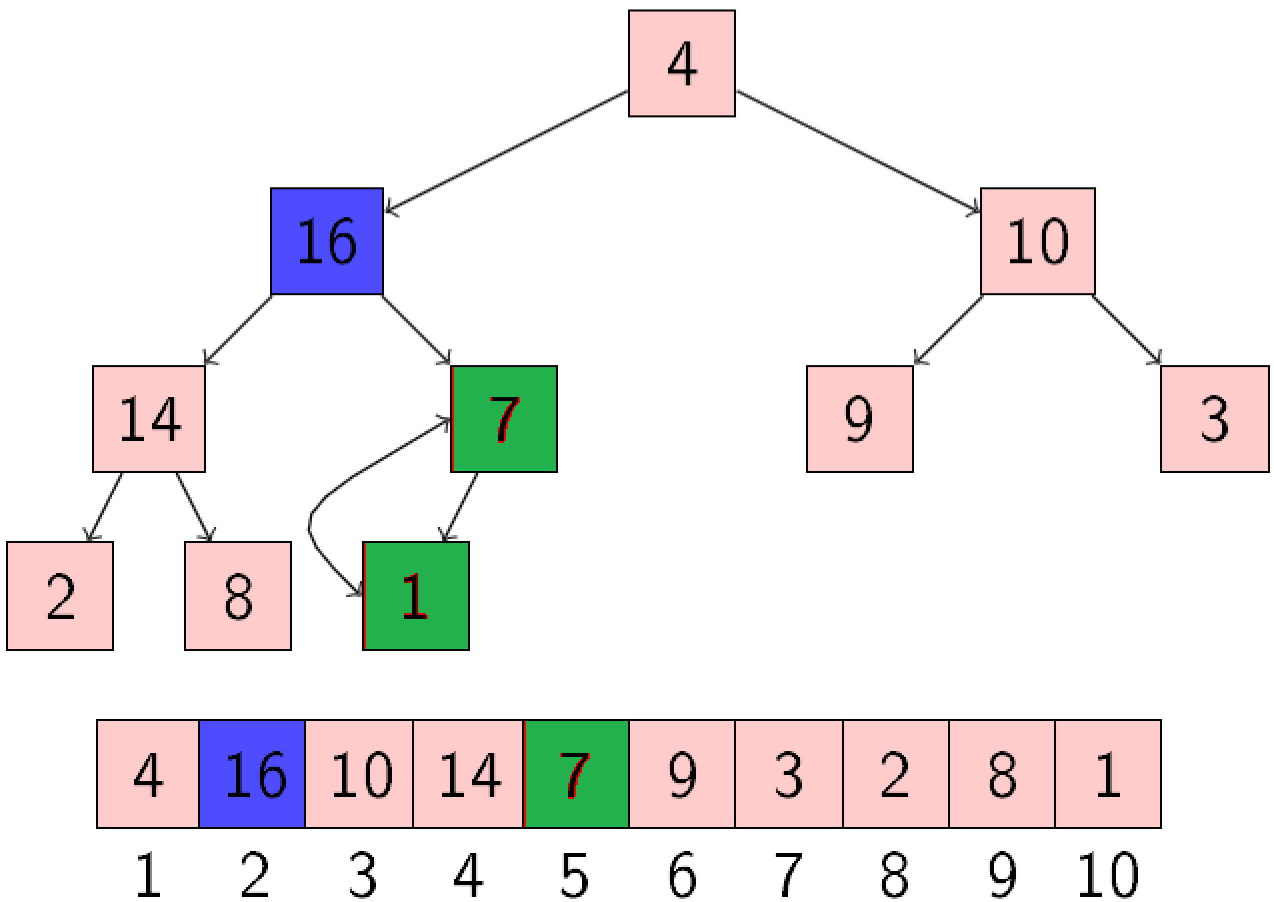
4	16	10	14	1	9	3	2	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (9)

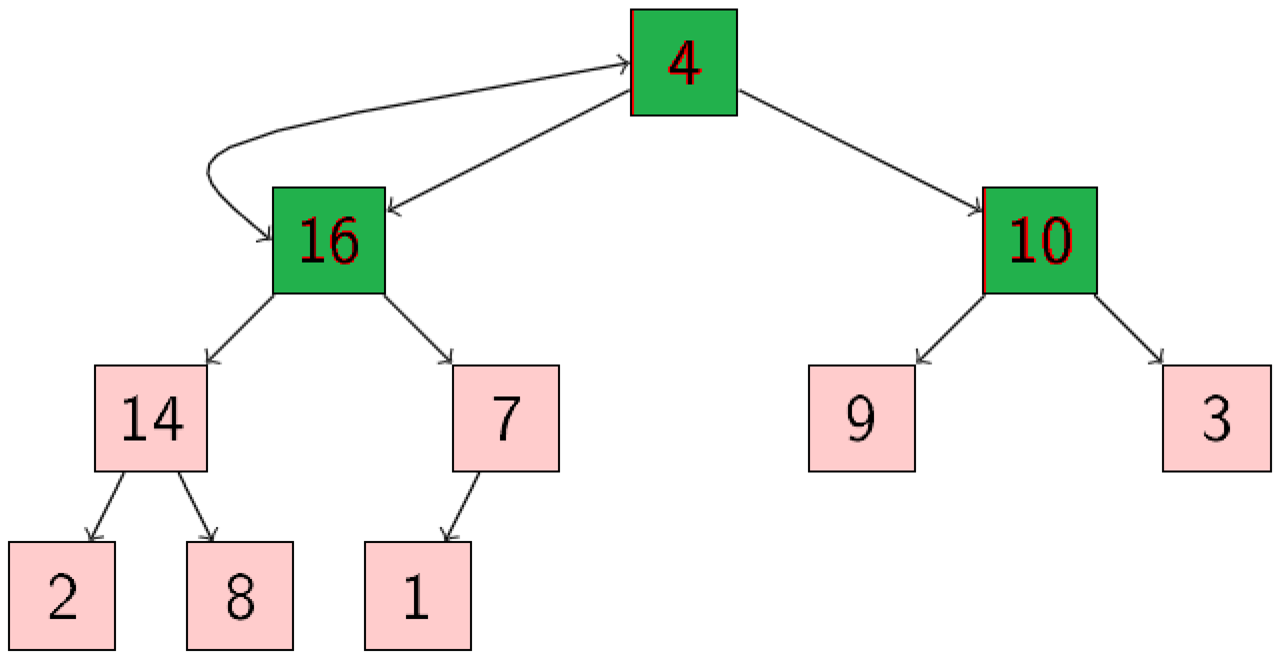


4	16	10	14	1	9	3	2	8	7
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (11)

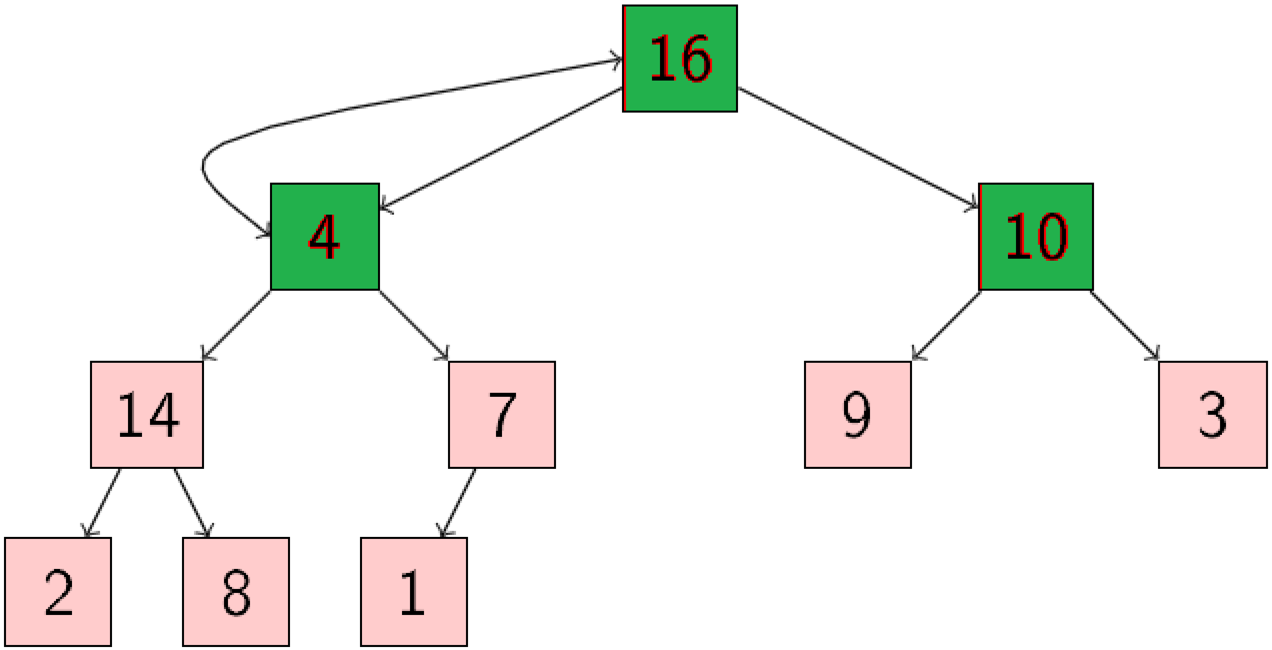


Operacje na kopcu – tworzenie kopca (12)



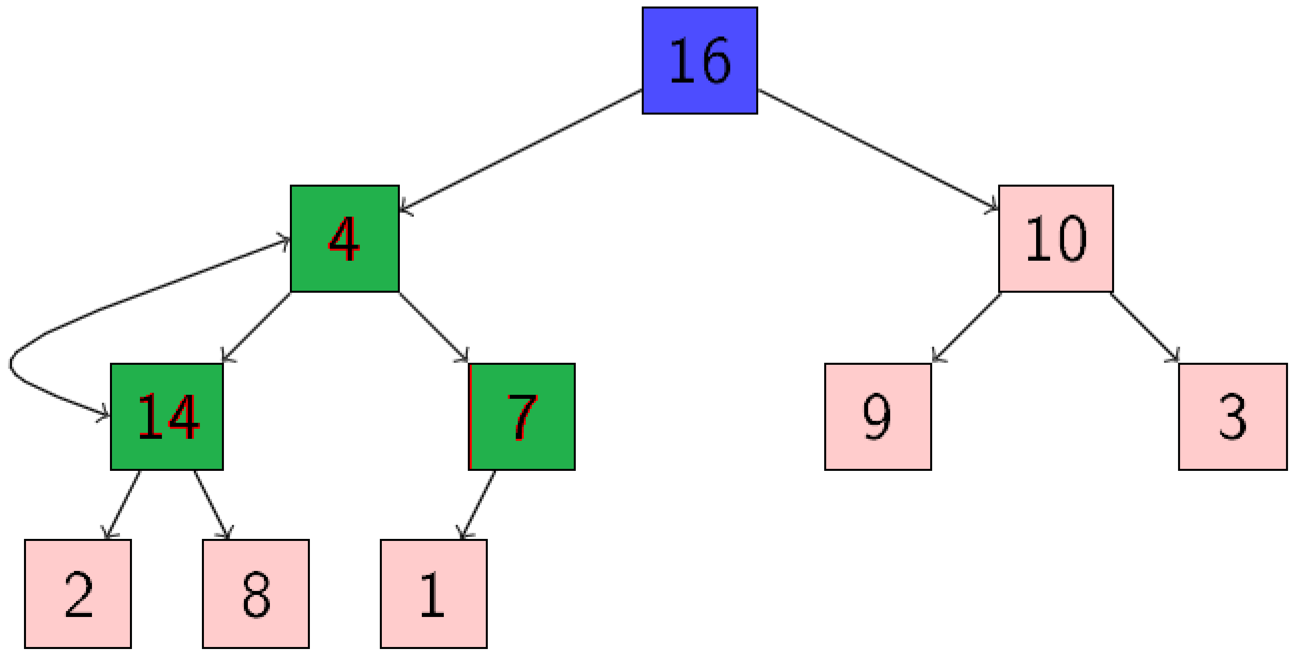
4	16	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (14)



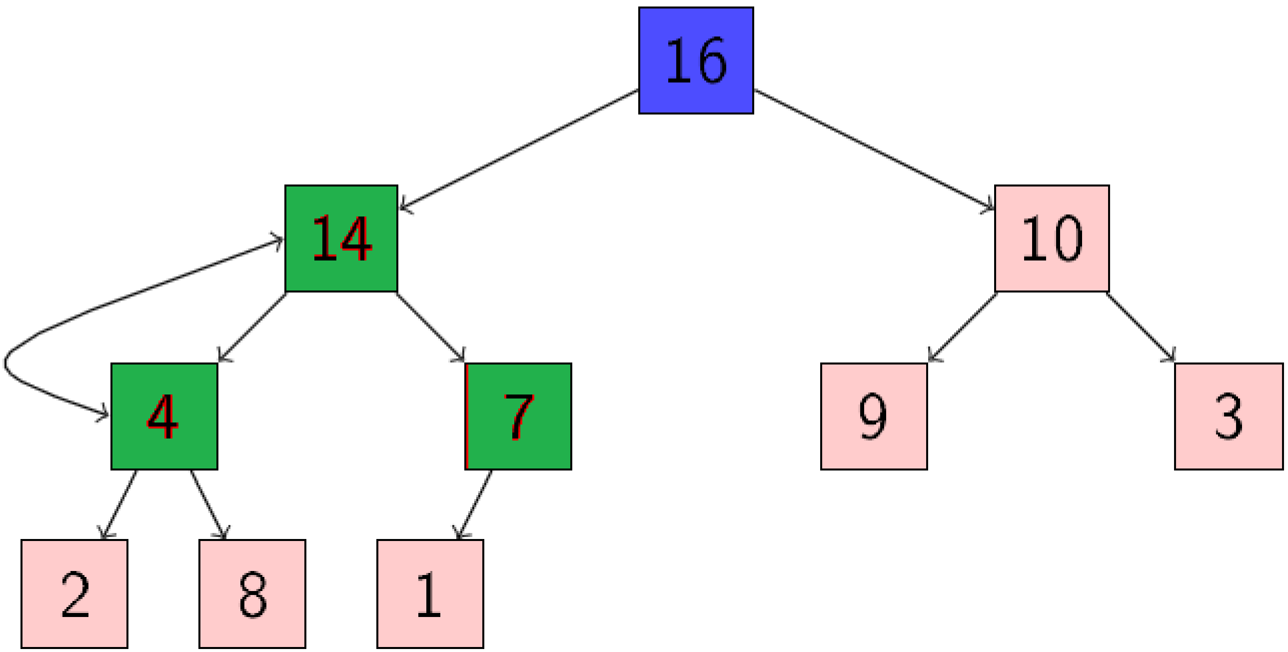
16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (15)



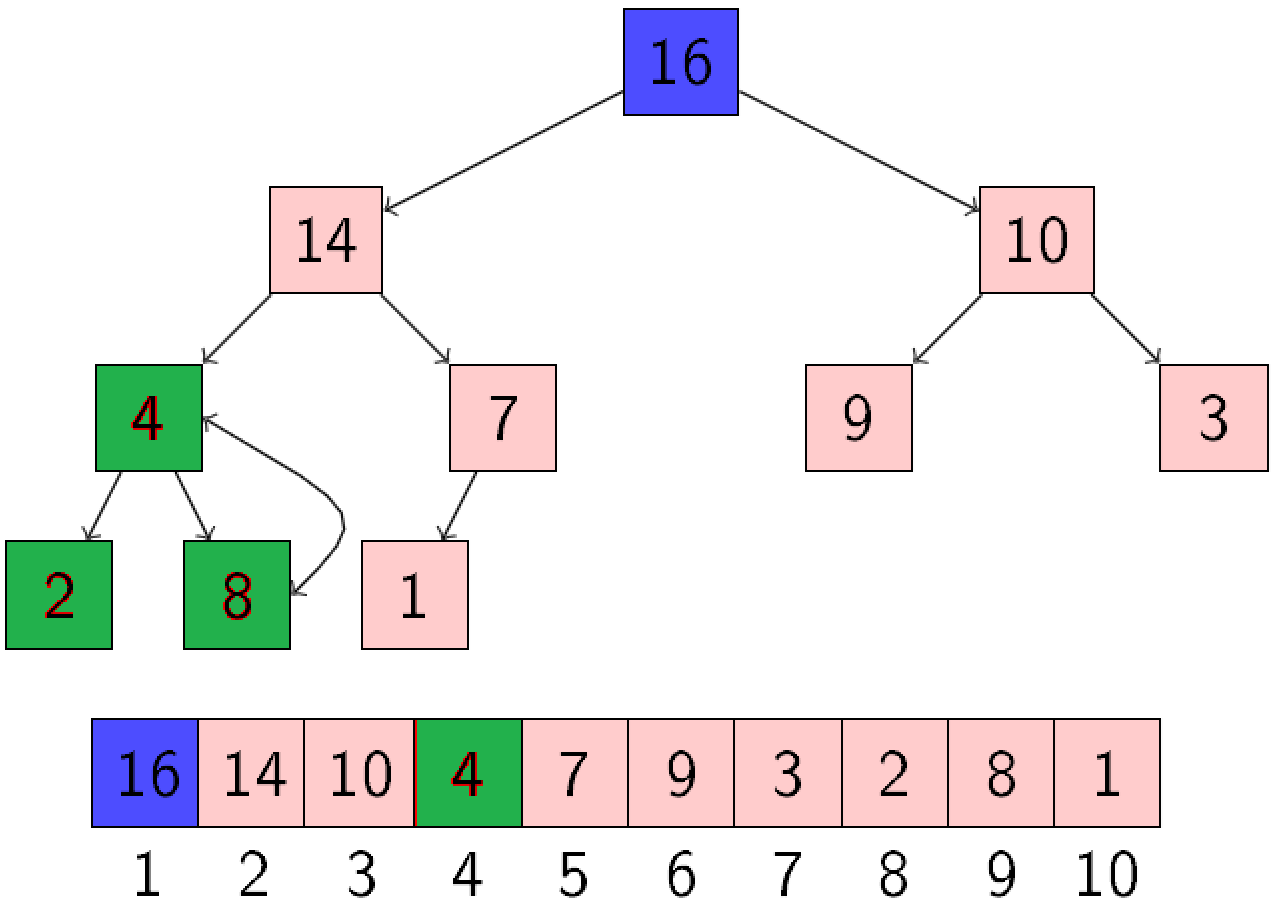
16	4	10	14	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (16)

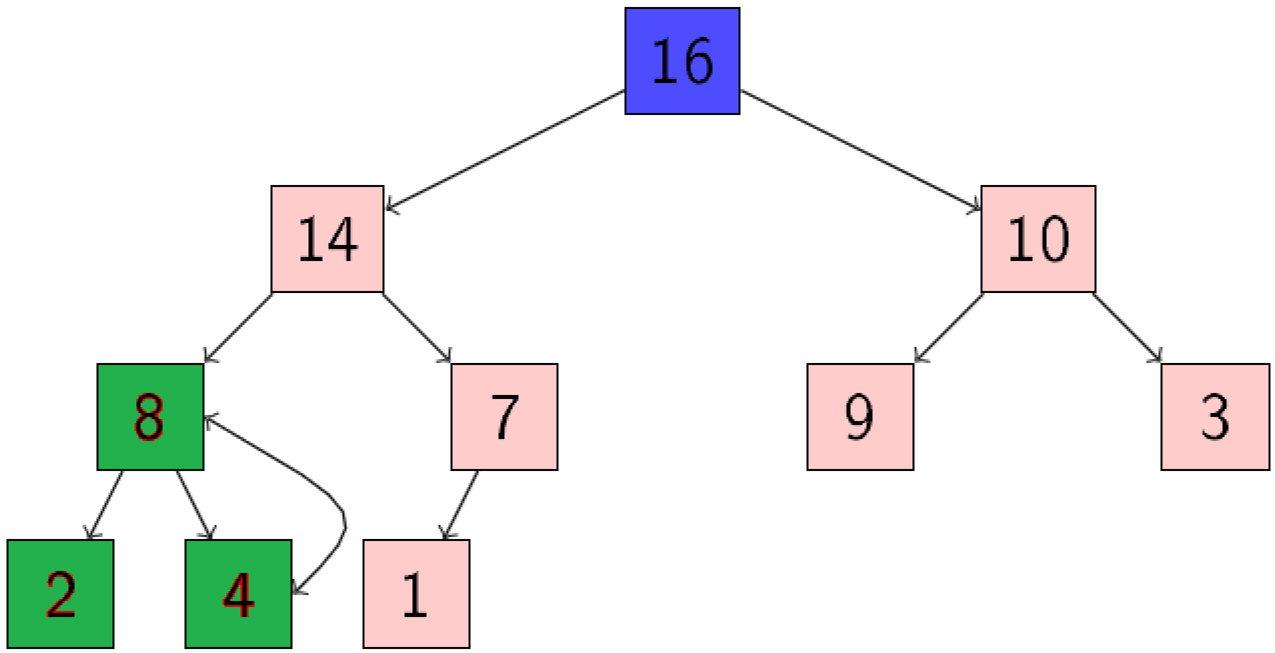


16	14	10	4	7	9	3	2	8	1
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (17)

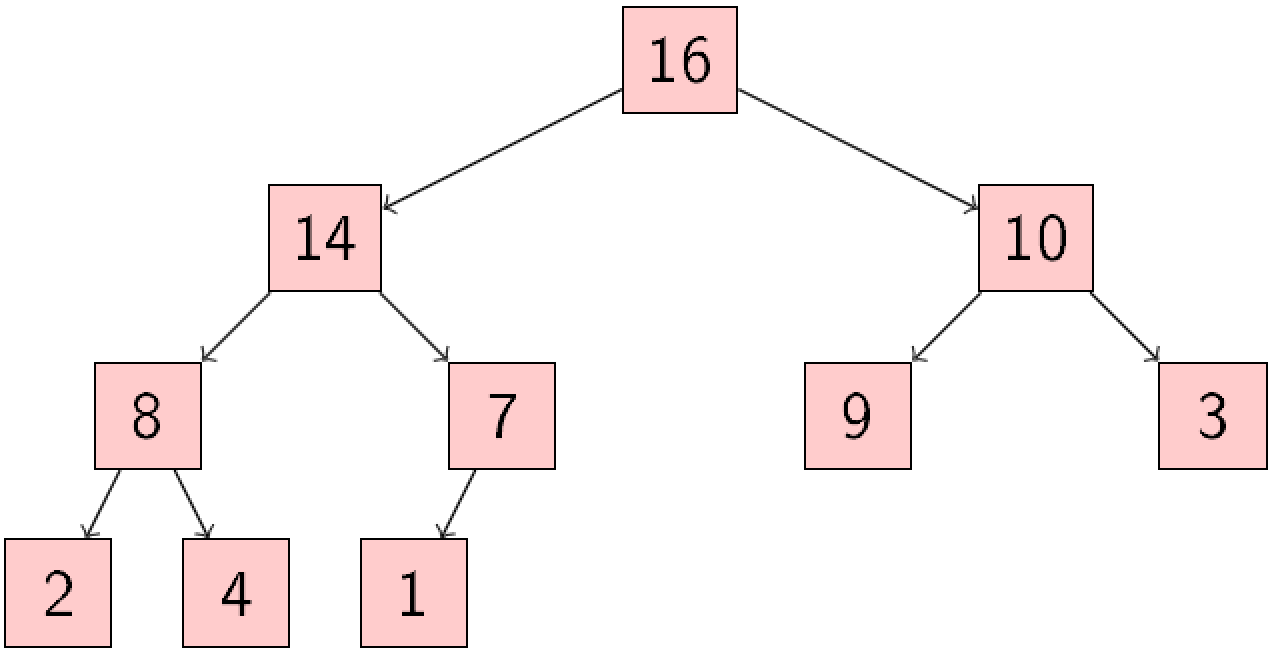


Operacje na kopcu – tworzenie kopca (18)



16	14	10	8	7	9	3	2	4	1
1	2	3	4	5	6	7	8	9	10

Operacje na kopcu – tworzenie kopca (19)



16	14	10	8	7	9	3	2	4	1
0	1	2	3	4	5	6	7	8	9

Operacje na kopcu – tworzenie kopca - pseudokod

```
//wersja szybsza – algorytm Floyda
void heap_create_dn(int tab[], int len_tab)
{
  for(int i = (len_tab-2)/2; i>=0; --i)
    heap_fix_down_rec(tab, i, len_tab);
}


//wersja wolniejsza
void heap_create_up(int tab[], int len_tab)
{
   for(int i=1; i<len_tab; ++i)
      heap_fix_up(tab, i);
}
```