

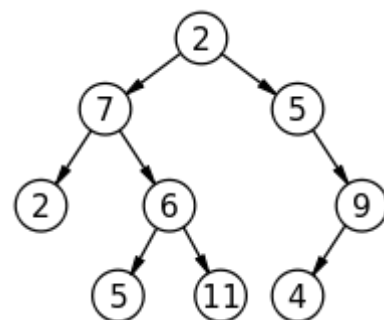
Algorytmy i złożoność obliczeniowa

Przygotowanie do Laboratorium 9: Drzewa binarne, drzewa BST.

Drzewa binarne – wprowadzenie.

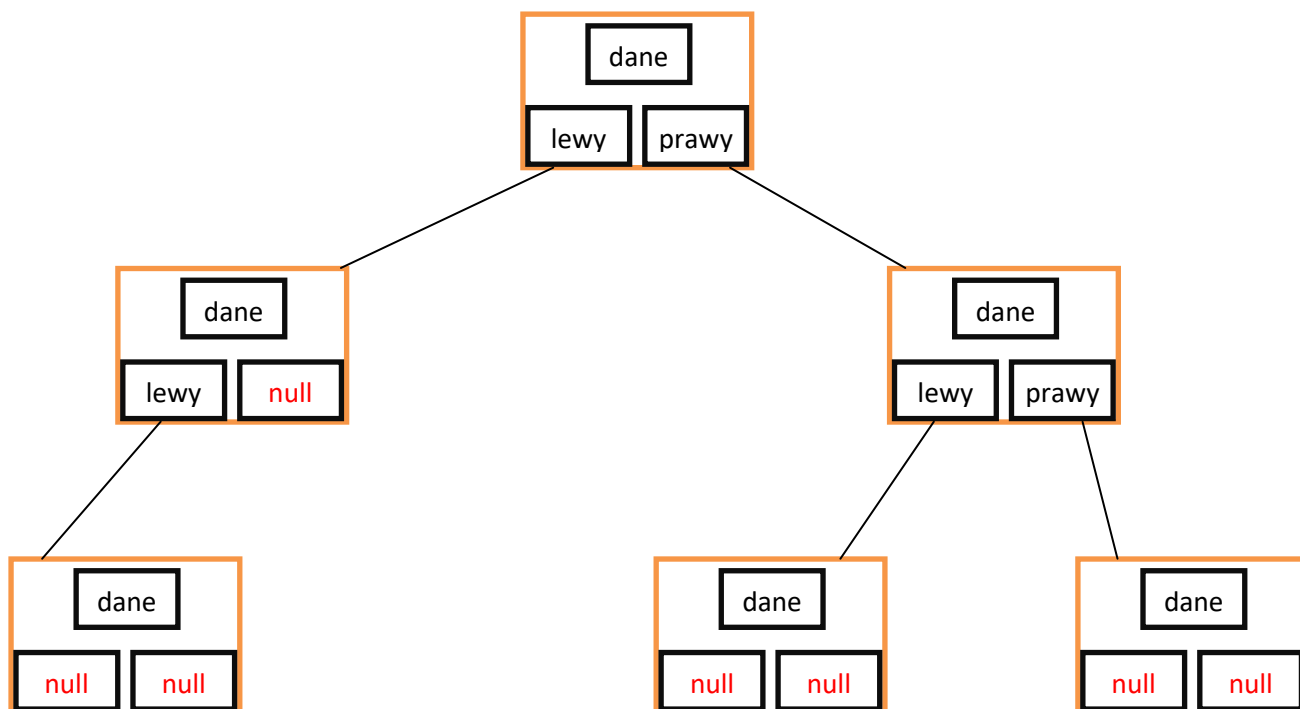
W informatyce drzewo binarne jest strukturą danych, w której każdy węzeł ma co najwyżej dwa węzły podrzędne, zwykle rozpoznawane jako "lewy" i "prawy".

Każdy węzeł w strukturze danych jest osiągalny z korzenia – wystarczy podążać za referencjami (lewy lub prawy) aby przechodzić coraz niżej w strukturze drzewa. Drzewa binarne są podstawą implementacji drzew BST (Binary Search Tree) oraz kopców binarnych.



1. Implementacja drzewa binarnego.

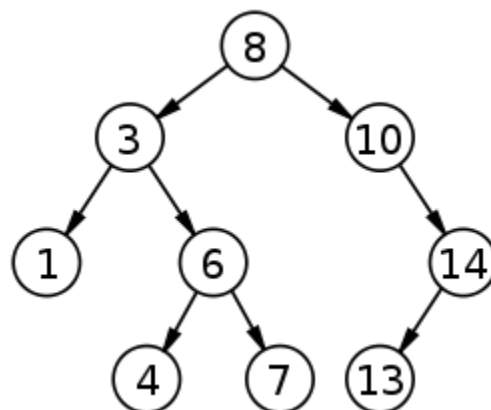
Węzły drzewa są obiektami składającymi się z trzech komponentów: danych przechowywanych w węźle oraz dwóch referencji do węzłów potomnych: *prawy* i *lewy*. Na zewnątrz całe drzewo jest reprezentowane jako referencja do korzenia drzewa.



2. Porządek w drzewach BST.

W drzewach BST węzły są uporządkowane.

Wartości znajdujące się w lewym poddrzewie są nie większe niż wartość w węźle nadrzędnym, a z kolei wartości w prawym poddrzewie są większe niż wartość w węźle nadrzędnym.



3. Iteracyjne wstawianie i wyszukiwanie węzła w drzewie

W celu budowania lub przeszukiwania drzewa BST wykorzystywane są często algorytmy rekurencyjne. Można jednak algorytmy te zapisać iteracyjnie przez „rozwiniecie” rekursji w pętlę `while`. Często ta druga wersja jest nawet efektywniejsza.

Pseudokod: Iteracyjne wyszukiwanie węzła w drzewie. Funkcja zwraca parę węzłów: element szukany i jego poprzednika(rodzica).

```
function search(v : T; r: node, var y: node) : node;
  {T jest dowolnym typem liniowo uporządkowanym; r jest korzeniem drzewa
  BST; y jest poprzednikiem wierzchołka wyszukiwanego przez search}
  var x : node;
begin
  x := r; y:=nil;
  while (x < > nil) and (key(x) < > v) do begin
    y:=x;
    if v < key(x) then x := left(x) else x := right (x);
  end;
  search := x
end search;
```

Procedura `insert` najpierw sprawdza czy element nie istnieje w drzewie. Następnie tworzy nowy wierzchołek i wstawia tam element `v`. Następnie przeszukuje drzewo w celu znalezienia „odpowiedniego” miejsca do dowiązania wierzchołka `x`.

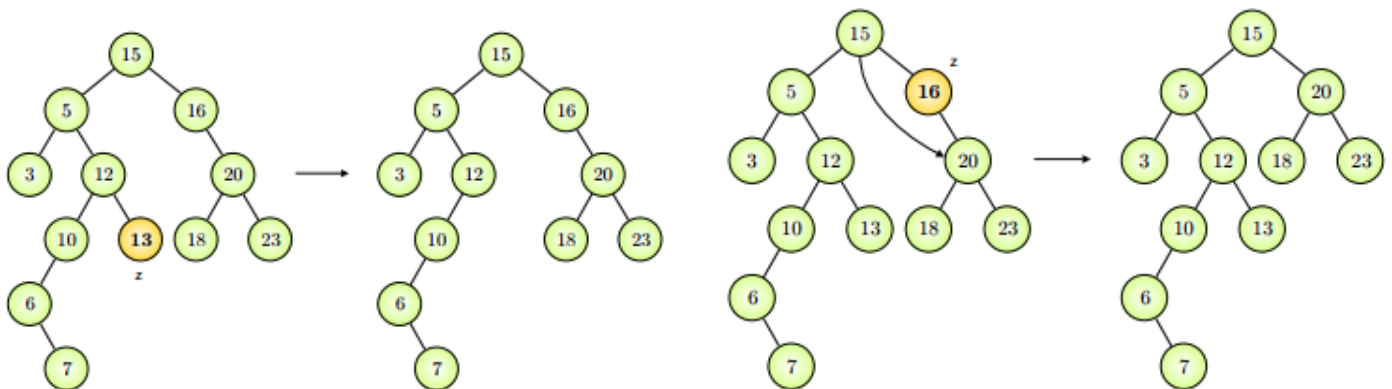
Pseudokod: Iteracyjne wstawianie węzła do drzewa BST.

```
procedure insert (v : T; var r : node);
var x, y: node;
begin
  if search(v, r, y) = nil then
  begin
    new(x) ;
    left(x) := nil; key(x) := v; right(x) := nil;
    if y= nil then r := x else
      if v< key(y) then left(y) := x
      else right(y) := x
    end
  end
end insert;
```

4. Usuwanie węzła z drzewa BST

W algorytmie tym rozpatrywane są trzy przypadki.

1. Jeśli z nie ma synów, to w jego ojcu rodzic[z] zastępujemy wskaźnik do z wartością NULL, zobacz Rys. 1.
2. Jeśli węzeł ma tylko jednego syna, to usuwamy z przez ustalenie wskaźnika między jego ojcem a jedynym synem, zobacz Rys. 2.

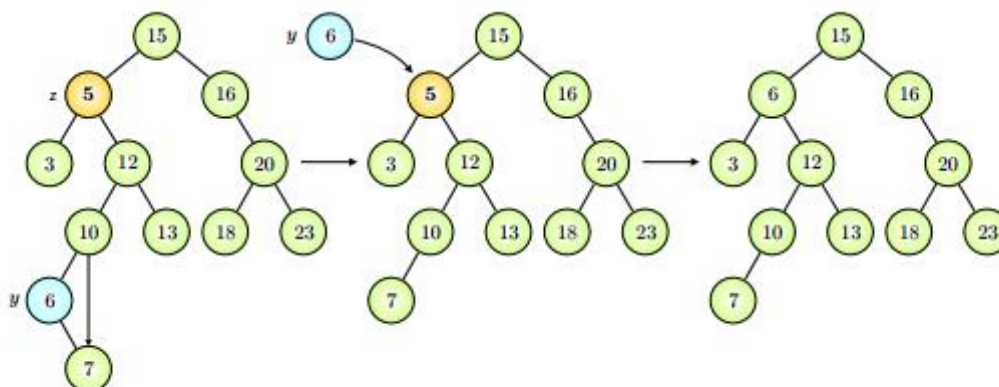


Rys. 1 Usuwanie węzła z z drzewa BST – przypadek 1. Rys.2 : Usuwanie węzła z z drzewa BST – przypadek 2.

3. Jeśli węzeł ma dwóch synów, to w miejscu usuwanego węzła z wstawiamy:
 - a. następnik z w zbiorze wierzchołków (najmniejszy z elementów większych od z) patrz . Rys. 3
 - b. poprzednik z w zbiorze wierzchołków(czyli największy spośród wszystkich elementów mniejszych od z).

Czyli

- (a) idziemy raz w prawo i do końca w lewo - Rys. 3 – wstawiamy 6
- (b) idziemy raz w lewo i do końca w prawo – wstawiamy 3



Rys. 3 : Usuwanie węzła z z drzewa BST – przypadek 3.

Poniższy pseudokod przedstawia pewną wersję algorytmu USUN realizującą wszystkie trzy wyżej przedstawione przypadki.

Pseudokod : Usuwanie węzła z drzewa

```
procedure delete(v ; T; var r : node);
var x, y, z, t : node;
b: 0 .. 1;
begin
  x := search(v, r , y);
  if x < > nil then
    begin
      if (left(x) =nil) or (right(x) = nil) then
        begin
          if (left(x) =nil) and (right(x) =nil) then z :=nil
          else if left(x) = nil then z := right (x)
              else z := left(x);
          if y = nil then r := z else
            if x= left(y) then left(y) := z else right(y) := z
        end
      else
        begin {left(x) < > nil i right(x) < > nil}
          b := random(2);
          //{jeśli b=0, to w miejsce v wstawiamy element bezpośrednio //poprzedzający
          v w zbiorze S. Jeśli b = 1, to w miejsce v wstawiamy //element bezpośrednio
          następujący po v w zbiorze S}

          if b = 0 then
            begin
              z := left(x) ;
              if right(z) = nil then left(x) := left(z)
            else
              begin
                repeat
                  t := z;
                  z := right(z)
                until right (z) = nil;
                right (t) := left (z)
              end
            end else
            begin
              z := right(x);
              if left(z) = nil then right(x) := right(z)
            else
              begin
                repeat
                  t := z;
                  z := left(z)
                until left (z) = nil;
                left(t) := right(z)
              end
            end
          end;
          key(x) := key(z)
        end end end delete;
```

6. Zadania.

1. Zaimplementować klasę reprezentującą drzewo binarne (węzeł drzewa). Uwzględnić metody dodające węzły potomne, przeszukujące drzewo, oraz wypisujące drzewo.
2. Zaimplementować klasę reprezentującą drzewo BST. Zaimplementować metody realizujące rekurencyjne i iteracyjne: dodawanie, wyszukiwanie i usuwanie węzłów.