Wstęp do informatyki

Wykład 11

Słownik; drzewa binarne; drzewa binarnych poszukiwań

Instytut Informatyki UWr

Temat wykładu

- Słownik – abstrakcyjny typ danych
- Drzewa – struktura danych
- Drzewa binarnych poszukiwań – implementacja słownika:
  - Wstawianie
  - Usuwanie
  - Wyszukiwanie
  - Wypisanie w porządku

Słownik

Słownik – struktura danych przechowująca elementy ze zbioru uporządkowanego, umożliwiająca wykonywanie operacji:

- Dodaj nowy element
- Usuń element
- Znajdź element (sprawdź czy występuje w zbiorze; jeśli tak, znajdź informacje o nim)
- Wypisz elementy w porządku

Uwaga: słownik jako abstrakcyjna struktura danych definiowany jest różnie; powyższa definicja dostosowana jest do potrzeb niniejszego wykładu!

Słownik

Słownik – implementacja:

- tablica?
- lista wiązana?
- przechowywanie dużego słownika w pamięci zewnętrznej (plik)?

Drzewo binarne – struktura danych

(ukorzenione) drzewo binarne:

Struktura danych złożona z węzłów powiązanych wskaźnikami:

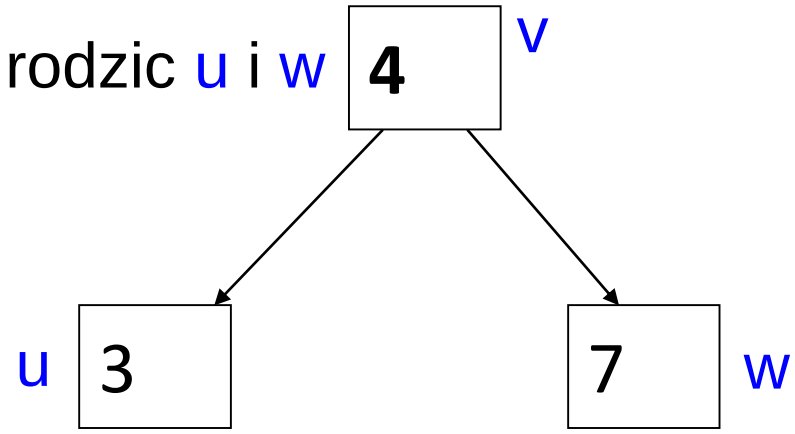
- Każdy węzeł ma co najwyżej dwoje dzieci (lewe dziecko i prawe dziecko)
- Każdy węzeł poza korzeniem ma dokładnie jednego rodzica
- Korzeń nie ma rodzica.
- W drzewie niepustym **dokładnie** jeden węzeł jest korzeniem.

Drzewo binarne – rodzic

Def. Rodzic

Węzeł v jest rodzicem x gdy:

- x jest lewym dzieckiem v LUB
- x jest prawym dzieckiem v.



lewe dziecko v

prawe dziecko v

Drzewo binarne w Ansi C

Definicje typów danych:

```
typedef struct node *pnode;
typedef struct node{
        int val;
        pnode left;
        pnode right;} snode;
```

Uwaga: `snode` to nazwa zastępująca „`struct node`”, a `pnode` zastępuje „`struct node *`”.

Drzewo binarne w Python

Obiekt definiujący węzeł drzewa:

```
class TreeItem:
    def __init__(self,value):
        self.val = value
        self.left = None
        self.right = None
```

Drzewo bin. – potomek, poddrzewo

Def**. Potomek**

Węzeł u jest potomkiem v gdy:

- u jest dzieckiem v LUB
- u jest potomkiem dziecka v

(uwaga – rekurencyjna definicja!)

Def. **Poddrzewo**

Lewe (prawe) poddrzewo v to drzewo składające się z lewego (prawego) dziecka v i jego potomków.

Drzewo binarnych poszukiwań

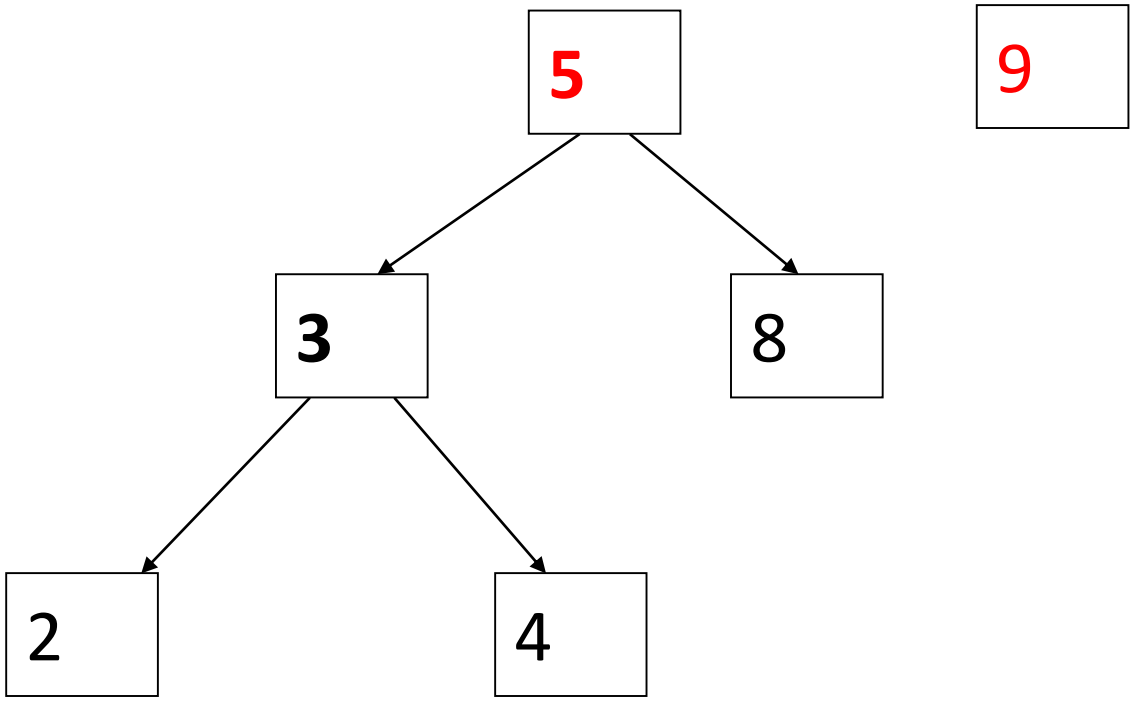
Drzewo binarnych poszukiwań

(Binary Search Tree (BST)):

- Drzewo binarne z kluczami w węzłach; klucze należą do zbioru uporządkowanego;
- Dla **każdego** węzła *v*, spełnione są następujące warunki :
  - klucze znajdujące się w lewym poddrzewie v są mniejsze (lub równe) od klucza w węźle v,
  - klucze znajdujące się w prawym poddrzewie v są większe (lub równe) od klucza w węźle v.

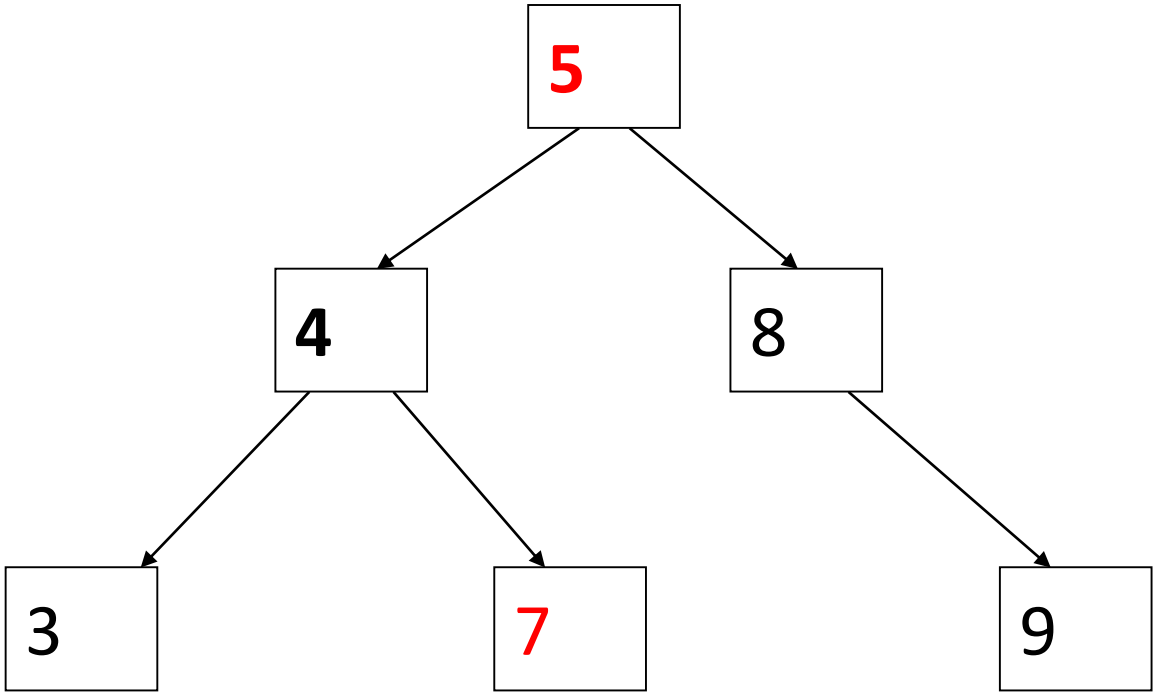
Drzewa – przykład

- Nie jest drzewem binarnym



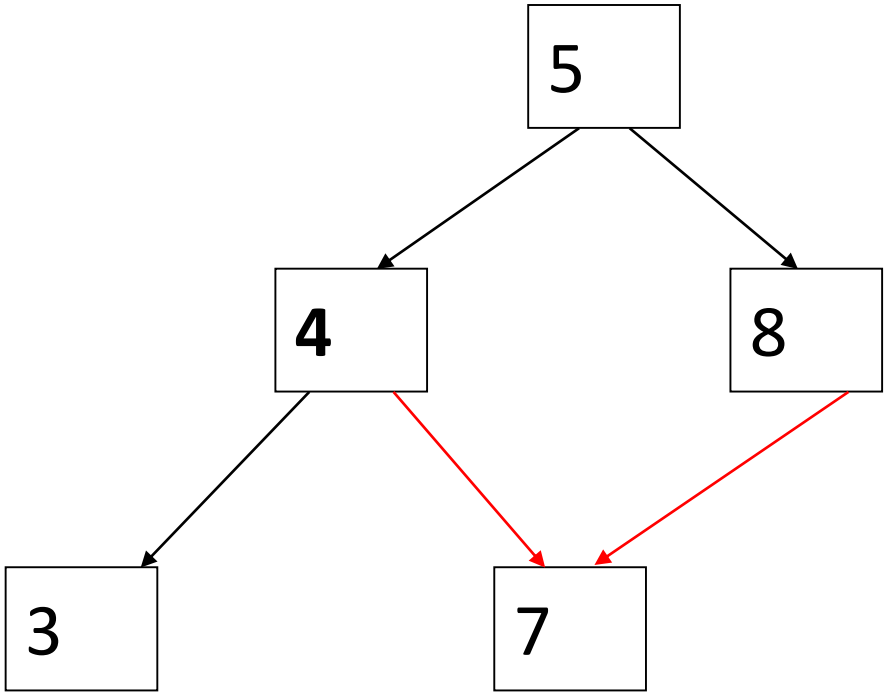
Drzewa – przykład

- Drzewo binarne
- Nie jest drzewem binarnych poszukiwań



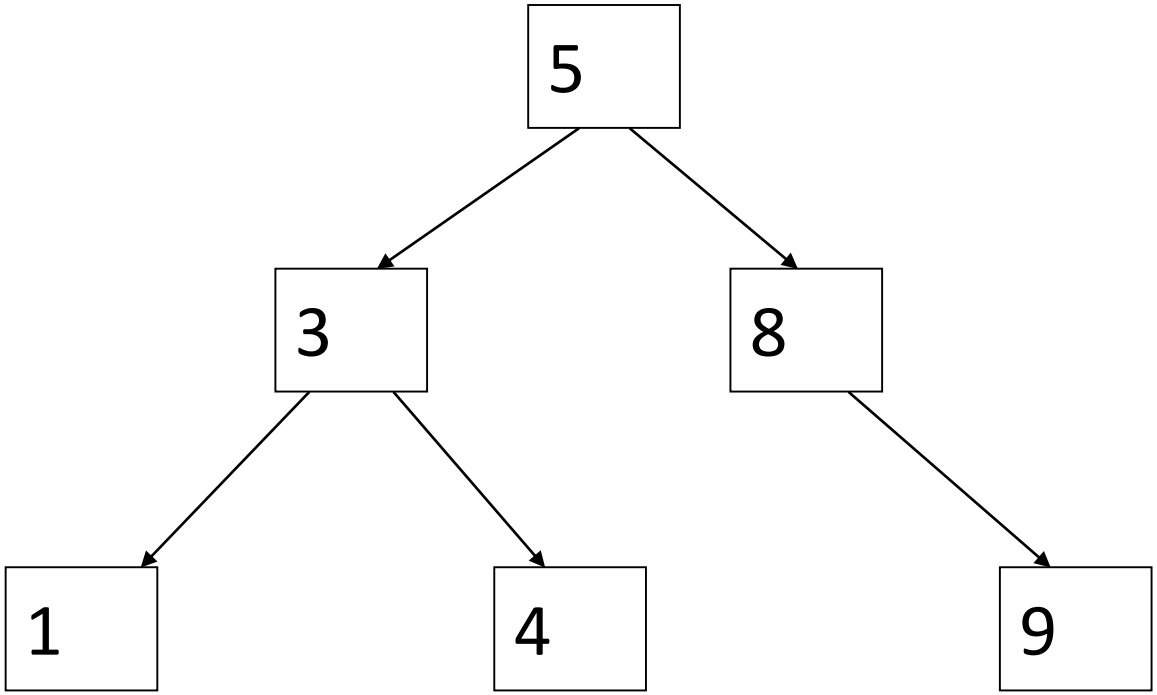
Drzewa – przykład

- Nie jest drzewem binarnym



Drzewa – przykład

- Drzewo binarne
- Drzewo binarnych poszukiwań



Drzewa – pojęcia

Def. Liść

Węzeł v jest liściem w drzewie gdy:

- v nie ma lewego dziecka, ORAZ
- v nie ma prawego dziecka.

Def. Ścieżka

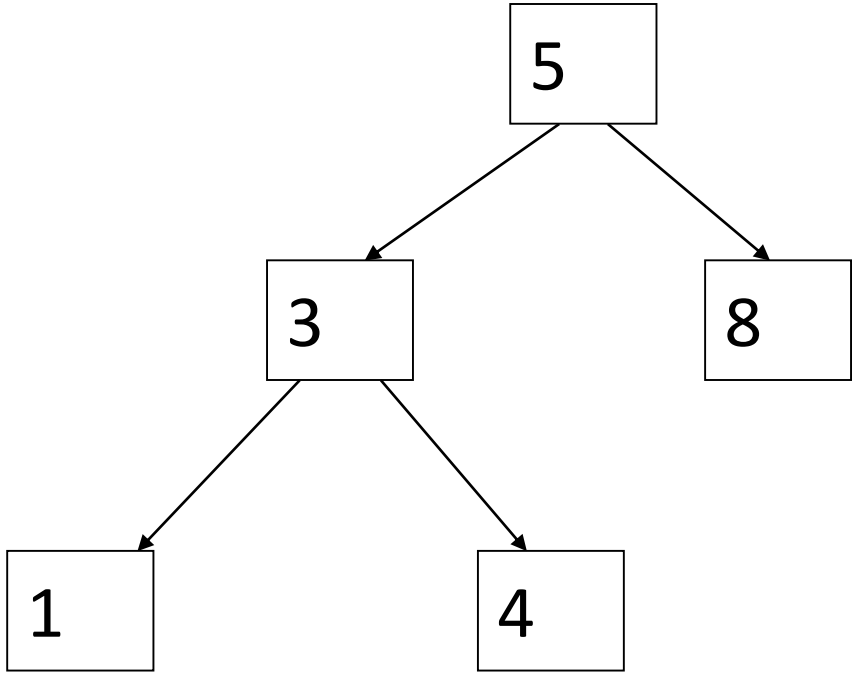
Ścieżka to ciąg węzłów v_1, v_2, …, v_k taki, że v_i jest dzieckiem v_{i-1} dla i=2,3,…,k

Def. Wysokość drzewa

Wysokość drzewa to największa długość (liczba węzłów) ścieżki od korzenia do liścia.

Drzewa – przykład

- Korzeń: 5
- Liście: 1, 4, 8
- Wysokość drzewa: 3 (najdłuższe ścieżki od korzenia do liścia mają 3 węzły, np. ścieżka z wartościami 5, 3, 4)



BST – wyszukiwanie

Specyfikacja

Wejście: root – korzeń drzewa,
skey – szukana wartość

Wyjście:

Jeśli skey występuje w drzewie o korzeniu root: wskaźnik na węzeł zawierający skey.
W przeciwnym przypadku: wskaźnik pusty NULL.

Idea rozwiązania:

- Jeśli drzewo puste: zwróć NULL
- Jeśli skey **równy** wartości w korzeniu drzewa: zwróć wskaźnik na korzeń.
- Jeśli skey **większy** od wartości w korzeniu: wyszukaj skey w **prawym** poddrzewie.
- Jeśli skey **mniejszy** od wartości w korzeniu: wyszukaj skey w **lewym** poddrzewie.

BST – wyszukiwanie

Specyfikacja

Wejście: root – korzeń drzewa,
skey – szukana wartość

Wyjście:

Jeśli skey występuje w drzewie o korzeniu root: wskaźnik na węzeł zawierający skey.

W przeciwnym przypadku: wskaźnik pusty NULL.

```
pnode search( pnode root, int skey)
{
     if (root==NULL || root->val == skey)
          return root;
     if (root->val > skey)
          return search(root->left, skey);
     else return search(root->right, skey);
}
```

BST – wyszukiwanie

Specyfikacja

Wejście: root – korzeń drzewa,
skey – szukana wartość

Wyjście:

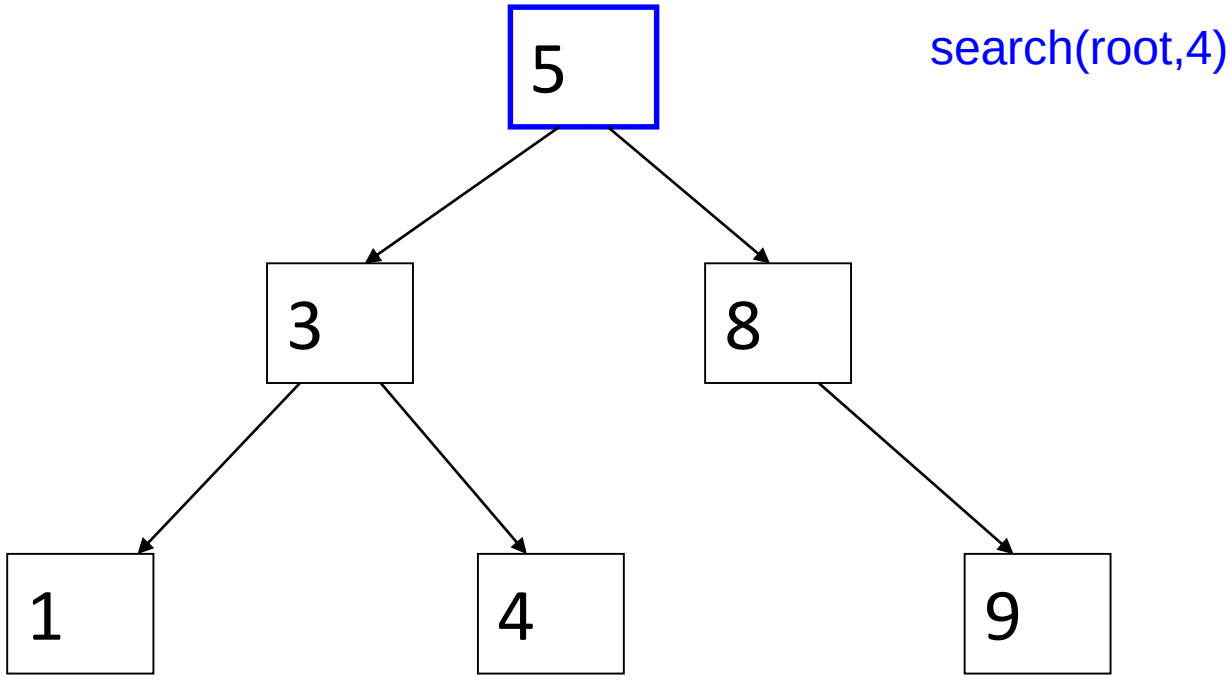
Jeśli skey występuje w drzewie o korzeniu root: wskaźnik na węzeł zawierający skey.

W przeciwnym przypadku: wskaźnik pusty NULL.

```
def search( root, skey):
  if (root==None or root.val == skey):
    return root
  if (root.val > skey):
    return search(root.left, skey)
  else:
    return search(root.right, skey)
```

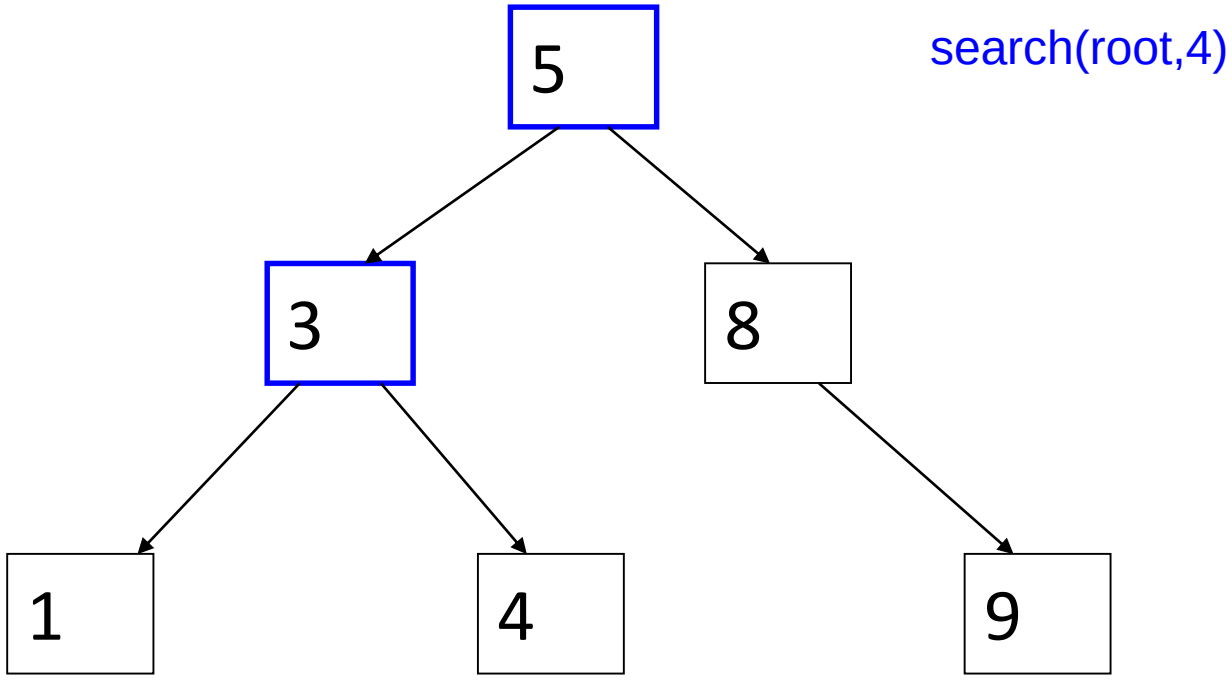
Wyszukiwanie - przykład

```
pnode search( pnode root, int skey)
{
        if (root==NULL || root->val == skey)
             return root;
        if (root->val > skey)
             return search(root->left, skey);
        else return search(root->right, skey);

}
```



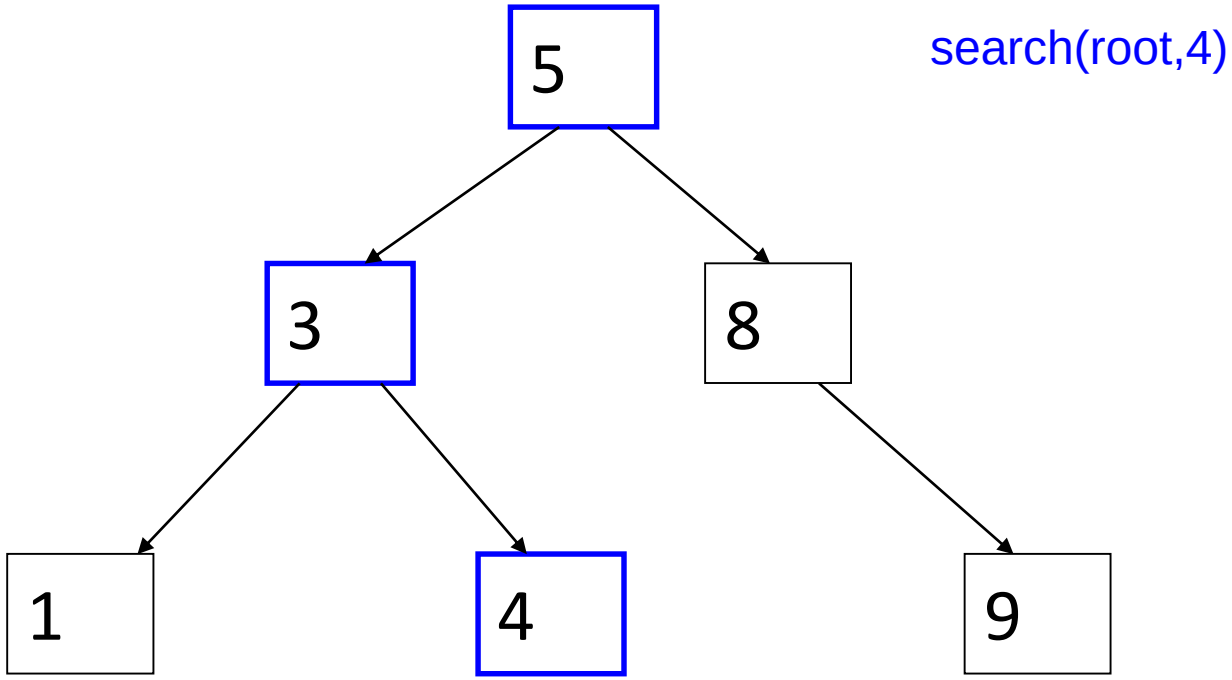
Wyszukiwanie - przykład

```
pnode search( pnode root, int skey)
{
        if (root==NULL || root->val == skey)
             return root;
        if (root->val > skey)
             return search(root->left, skey);
        else return search(root->right, skey);

}
```



Wyszukiwanie - przykład

```
pnode search( pnode root, int skey)
{
        if (root==NULL || root->val == skey)
             return root;
        if (root->val > skey)
             return search(root->left, skey);
        else return search(root->right, skey);

}
```



BST – wstawianie

Specyfikacja

Wejście: root – korzeń drzewa,
nkey – wstawiana wartość

Wyjście:

•Jeśli w drzewie **nie ma** węzła z kluczem nkey: wskaźnik na korzeń drzewa uzyskanego z root po wstawieniu węzła z kluczem nkey.

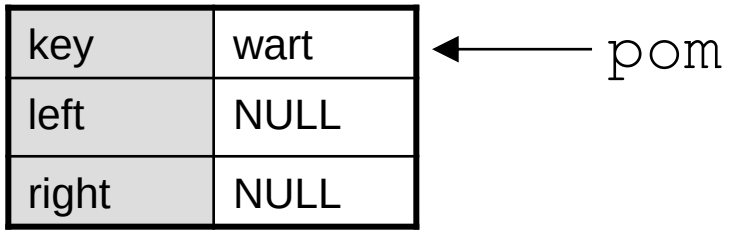
•W przeciwnym przypadku : oryginalna wartość root (drzewo się nie zmienia).

Idea rozwiązania:

- Jeśli drzewo puste: utwórz węzeł z kluczem nkey, bez dzieci, zwróć go jako wynik.
- Jeśli nkey **równy** wartości w korzeniu drzewa: zwróć wskaźnik na korzeń.
- Jeśli nkey **większy** od wartości w korzeniu: wstaw nkey do **prawego** poddrzewa, zwróć wskaźnik na korzeń.
- Jeśli nkey **mniejszy** od wartości w korzeniu: wstaw nkey do **lewego** poddrzewa, zwróć wskaźnik na korzeń.

Drzewa – tworzenie węzła

```
pnode utworz(int wart)
{ //utworzenie nowego wezla
    pnode pom;
    pom = (pnode) malloc(sizeof(snode));
    pom->left = NULL;
    pom->right = NULL;
    pom->val = wart;
    return pom;
}
```



Odpowiednik w python:

```
TreeItem(wart)
```

BST – wstawianie

Specyfikacja

Wejście: root – korzeń drzewa, nkey – wstawiana wartość

Wyjście:

•Jeśli w root **nie ma** węzła z kluczem nkey: wskaźnik na korzeń drzewa uzyskanego z root po wstawieniu węzła z kluczem nkey.

•W przeciwnym przypadku : oryginalna wartość root (drzewo się nie zmienia).

```
pnode insert( pnode root, int nkey)
{
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```

BST – wstawianie

Specyfikacja

Wejście: root – korzeń drzewa, nkey – wstawiana wartość

Wyjście:

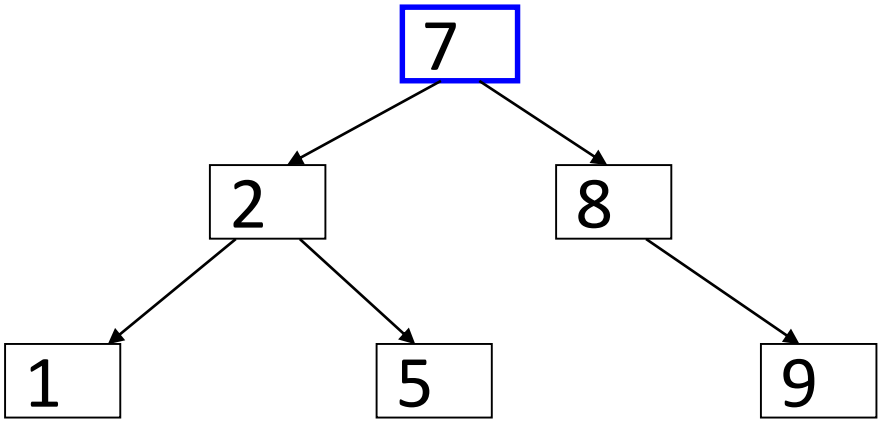
•Jeśli w root **nie ma** węzła z kluczem nkey: wskaźnik na korzeń drzewa uzyskanego z root po wstawieniu węzła z kluczem nkey.

•W przeciwnym przypadku : oryginalna wartość root (drzewo się nie zmienia).

```
def insert( root, nkey):
  if (root==None): return TreeItem(nkey)
  if (nkey < root.val):
    root.left = insert(root.left, nkey)
  else:
    if (nkey > root.val):
      root.right = insert(root.right, nkey)
  return root
```

Wstawianie - przykład

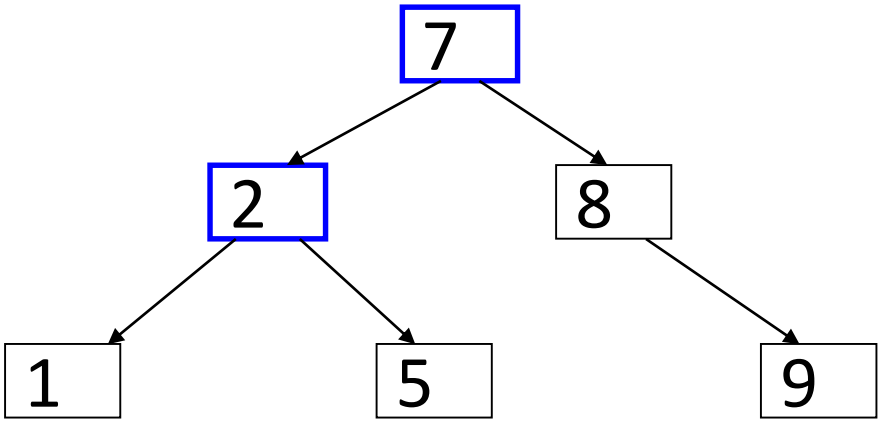
```
pnode insert( pnode root, int nkey)
{ pnode pom;
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```



insert(root,4)

Wstawianie - przykład

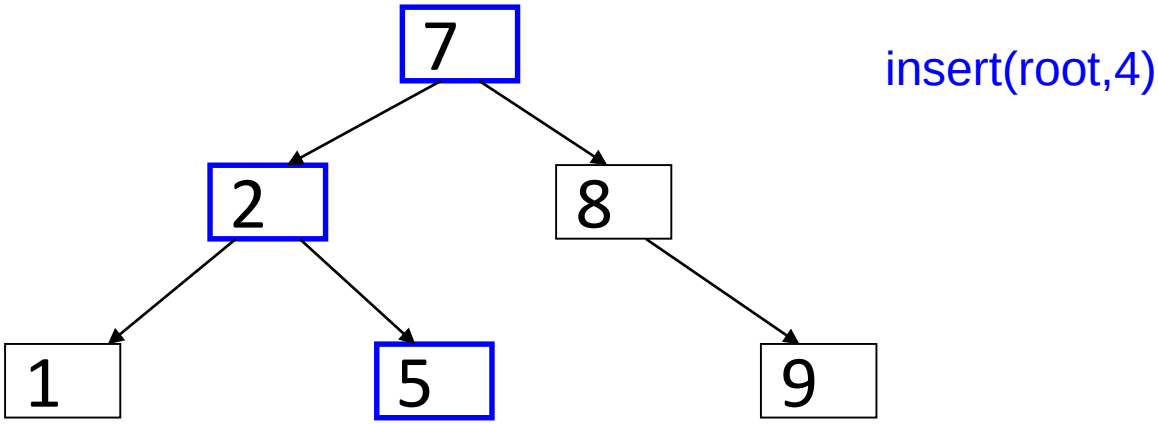
```
pnode insert( pnode root, int nkey)
{ pnode pom;
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```



insert(root,4)

Wstawianie - przykład

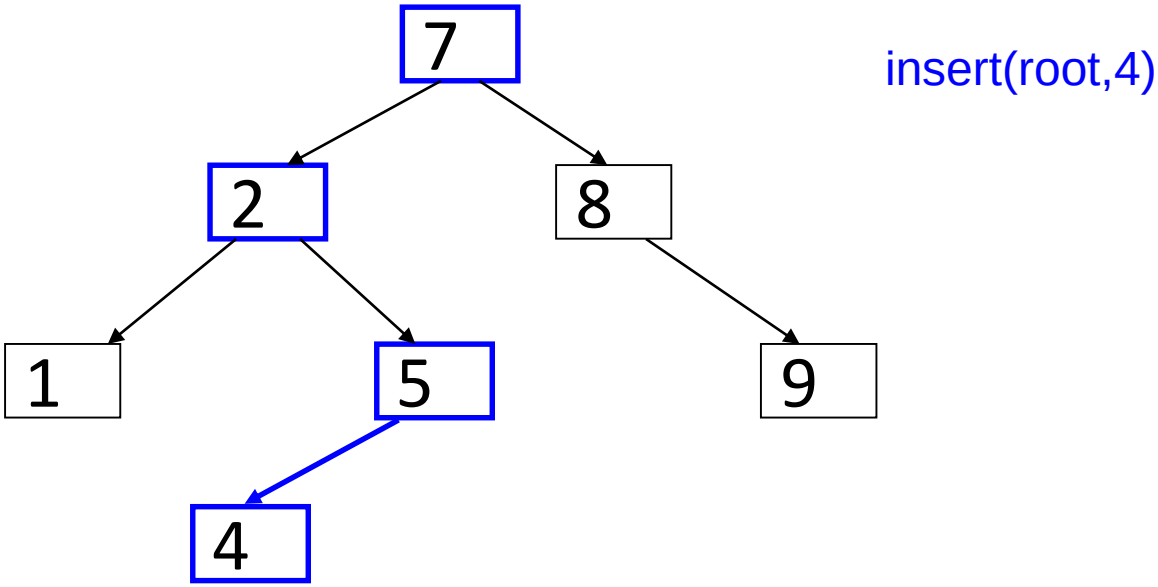
```
pnode insert( pnode root, int nkey)
{ pnode pom;
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```



insert(root,4)

Wstawianie - przykład

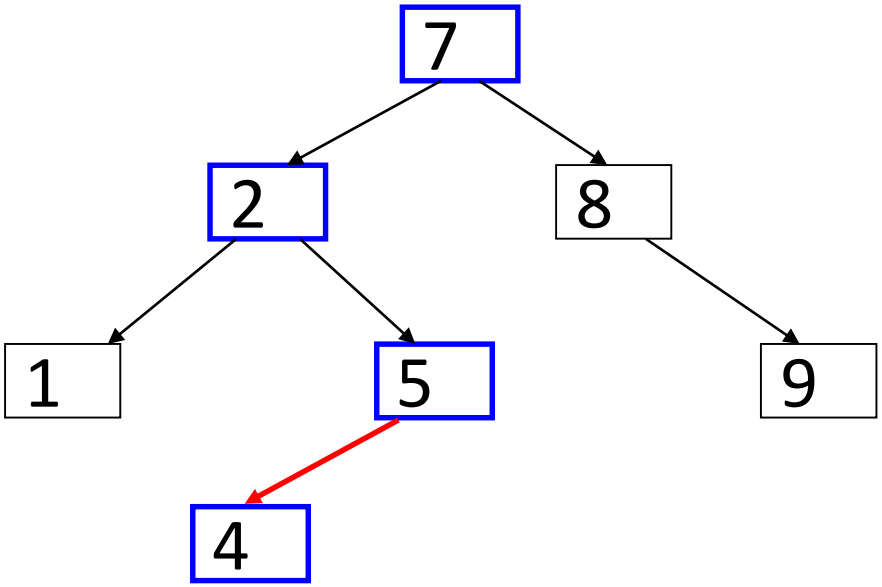
```
pnode insert( pnode root, int nkey)
{ pnode pom;
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```



insert(root,4)

Wstawianie - przykład

```
pnode insert( pnode root, int nkey)
{ pnode pom;
  if (root==NULL) return utworz(nkey);
  if (nkey < root->val)
    root->left = insert(root->left, nkey);
  else
    if (nkey > root->val)
      root->right = insert(root->right, nkey);
  return root;
}
```



insert(root,4)

BST – wypisanie w porządku

Specyfikacja

Wejście: root – korzeń drzewa BST

Wyjście:

wypisanie na wyjściu elementów z drzewa root w kolejności niemalejącej.

Idea rozwiązania:

- Jeśli drzewo niepuste:
  - Wypisz zawartość **lewego** poddrzewa korzenia.
  - Wypisz klucz znajdujący się **w korzeniu**.
  - Wypisz zawartość **prawego** poddrzewa korzenia.

BST – wypisanie w porządku

Specyfikacja

Wejście: root – korzeń drzewa BST

Wyjście:

wypisanie na wyjściu elementów z drzewa root w kolejności niemalejącej.

```
void write( pnode root )
{ if (root!=NULL){
    write( root->left);
    printf("%d\n", root->val);
    write( root->right);
  }
}
```

```
def write(root ):
  if (root!=None):
    write( root.left)
    print root.val
    write( root.right)
```

BST – usuwanie

Specyfikacja

Wejście: root – korzeń drzewa BST,
dkey – klucz do usunięcia

Wyjście:

Korzeń drzewa uzyskanego z drzewa o korzeniu root po usunięciu węzła zawierającego dkey.

Idea rozwiązania:

1.w ← wartość key w węźle root

2.Jeśli dkey **mniejszy** od *w*: usuń dkey z root->lewy

3.Jeśli dkey **większy** od *w*: usuń dkey z root->prawy

4.Jeśli dkey równy *w*:

a) Jeśli **lewe** poddrzewo root puste: wstaw **prawe** poddrzewo root w miejsce root

b) Jeśli **prawe** poddrzewo root puste: wstaw **lewe** poddrzewo root w miejsce root

c) Jeśli oba poddrzewa root niepuste: znajdź **węzeł** drzewa, który można („łatwo”) przenieść w miejsce root (nie naruszając porządku BST).

BST – usuwanie

Specyfikacja

Wejście: root – korzeń drzewa BST,
dkey – klucz do usunięcia

Wyjście:

Korzeń drzewa uzyskanego z drzewa o korzeniu root po usunięciu węzła zawierającego dkey.

Idea rozwiązania:

1. *w* ← wartość w węźle root

(…)

4. Jeśli dkey równy *w*:

(…)

c) Jeśli oba poddrzewa root niepuste: znajdź węzeł drzewa, który można wstawić w miejsce root (nie naruszając porządku BST):

i. znajdź *w'* – najmniejszy element prawego poddrzewa root (węzeł zawierający *w'* nie ma lewego poddrzewa!)

ii. wstaw wartość *w'* jako val w root

iii. usuń „stary" węzeł zawierający *w'*

BST – usuwanie

Specyfikacja

Wejście: root – korzeń drzewa BST,
dkey – klucz do usunięcia

Wyjście:

Korzeń drzewa uzyskanego z drzewa o korzeniu root po usunięciu węzła zawierającego dkey.

```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else // root wskazuje na element do usuniecia
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root);
           root=pom;
          } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root);
             root = pom;
           } else
            delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```

BST – usuwanie

Specyfikacja

Wejście: root – korzeń drzewa BST,
dkey – klucz do usunięcia

Wyjście:

Korzeń drzewa uzyskanego z drzewa o korzeniu root po usunięciu węzła zawierającego dkey.

```
def deletekey( root, dkey):
  if (root!=None):
    if (dkey < root.val):
      root.left = deletekey(root.left, dkey); # 2
    else:
      if (dkey > root.val):
        root.right = deletekey(root.right, dkey); #3
      else: # root wskazuje na element do usuniecia
        if (root.left == None): # 4a
          root=root.right
        else:
          if (root.right == None): #4b
            root = root.left
          else:
            delhlp(root) # 4c - oba poddrzewa niepuste
  return root
```

BST – usuwanie

Specyfikacja

Wejście: root – korzeń drzewa BST,
dkey – klucz do usunięcia

Wyjście:

Korzeń drzewa uzyskanego z drzewa o korzeniu root po usunięciu węzła zawierającego dkey.

Idea rozwiązania:

1. *w* ← wartość w węźle root
   (…)
4. Jeśli dkey równy *w*:
   (…)
   c) Jeśli oba poddrzewa root niepuste: znajdź węzeł drzewa, który można wstawić w miejsce root (nie naruszając porządku BST):
      i. znajdź *w'* – najmniejszy element prawego poddrzewa root (węzeł zawierający *w'* nie ma lewego poddrzewa!)
      ii. wstaw wartość *w'* jako val w root
      iii. usuń „stary" węzeł zawierający *w'*

`delhlp`

BST – usuwanie

Specyfikacja (funkcji pomocniczej)

Wejście: root – korzeń drzewa BST, z niepustym prawym poddrzewem

Wyjście:

root to drzewo BST uzyskane po usunięciu wartości przechowywanej w korzeniu i przeniesieniu do korzenia najmniejszej wartości z prawego poddrzewa węzła root

```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur, prev;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // szukanie min. w root->right
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) //korzeń root->right NIE jest jego minimum
       prev->left = cur->right;
  else             //korzeń root->right jest jego minimum
       root->right = cur->right;
  free(cur);
}
```

BST – usuwanie

Specyfikacja (funkcji pomocniczej)

Wejście: root – korzeń drzewa BST, z niepustym prawym poddrzewem

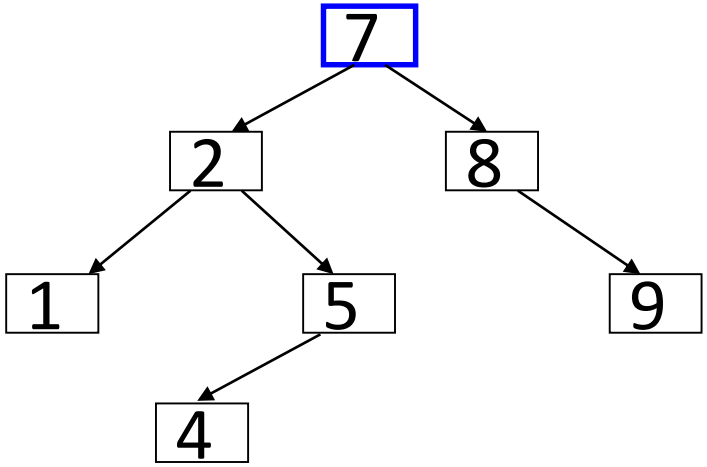
Wyjście:

root to drzewo uzyskane po usunięciu wartości przechowywanej w korzeniu i przeniesieniu do korzenia najmniejszej wartości z prawego poddrzewa węzła root

```
def delhlp(root): #FUNKCJA POMOCNICZA!
  prev = root
  cur = root.right
  while (cur.left!=None): # szukanie min. w root->right
    prev = cur; cur = cur.left;
  root.val = cur.val
  if (prev!=root): #korzeń root->right NIE jest jego minimum
    prev.left = cur.right;
  else:              #korzeń root->right jest jego minimum
    root.right = cur.right
```

Usuwanie - przykład

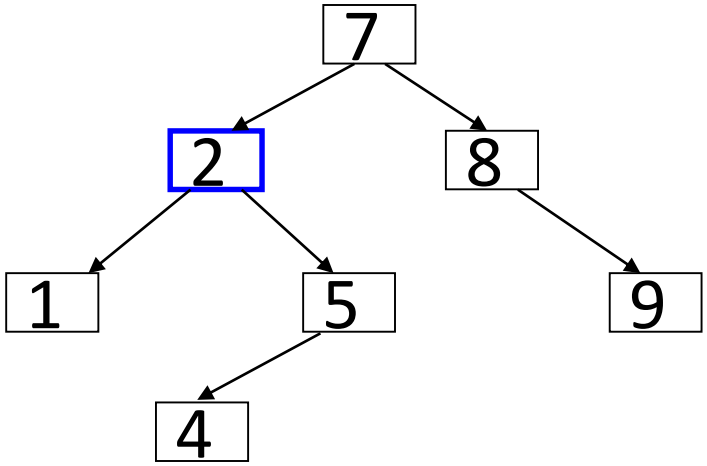
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
          } else
            if (root->right == NULL){ //4b
              pom=root->left; free(root); root = pom;
            } else
              delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,5)

Usuwanie - przykład

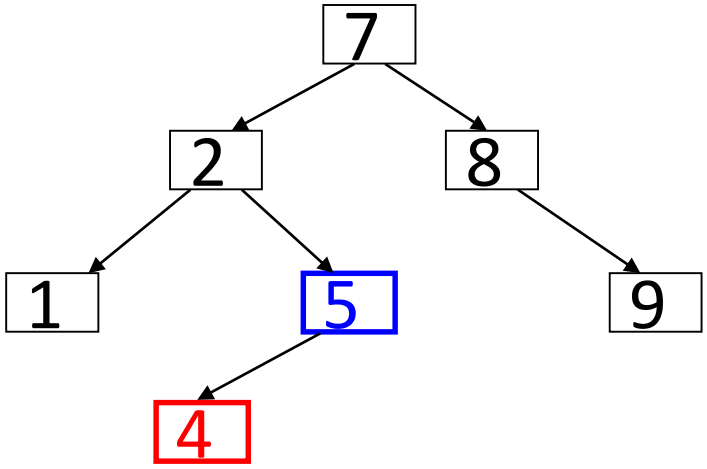
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
         } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root); root = pom;
           } else
             delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,5)

Usuwanie - przykład

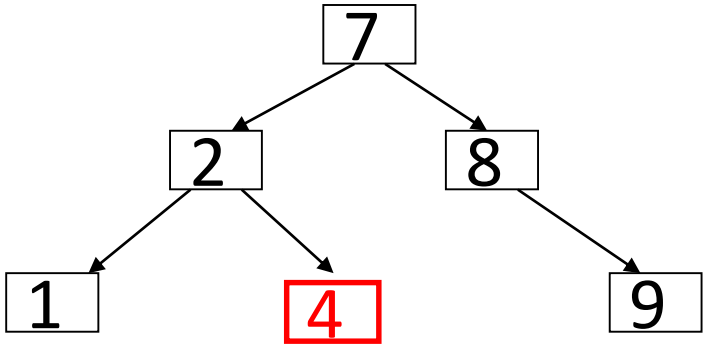
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
         } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root); root = pom;
           } else
             delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,5)

Usuwanie - przykład

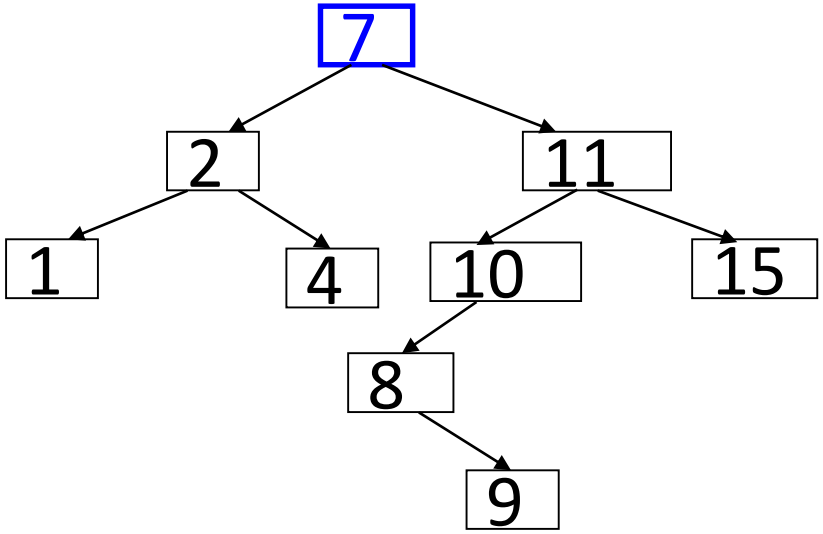
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
         } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root); root = pom;
           } else
             delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,5)

Usuwanie - przykład

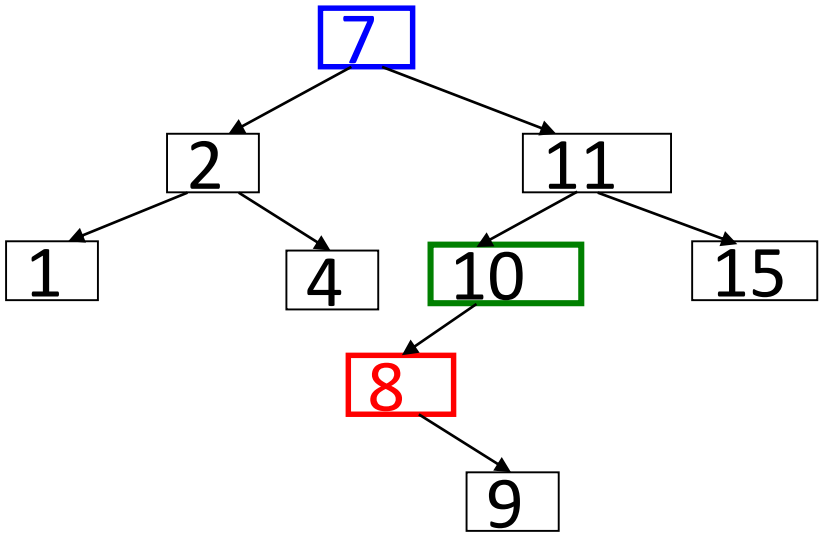
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
         } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root); root = pom;
           } else
             delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,7)

Usuwanie - przykład

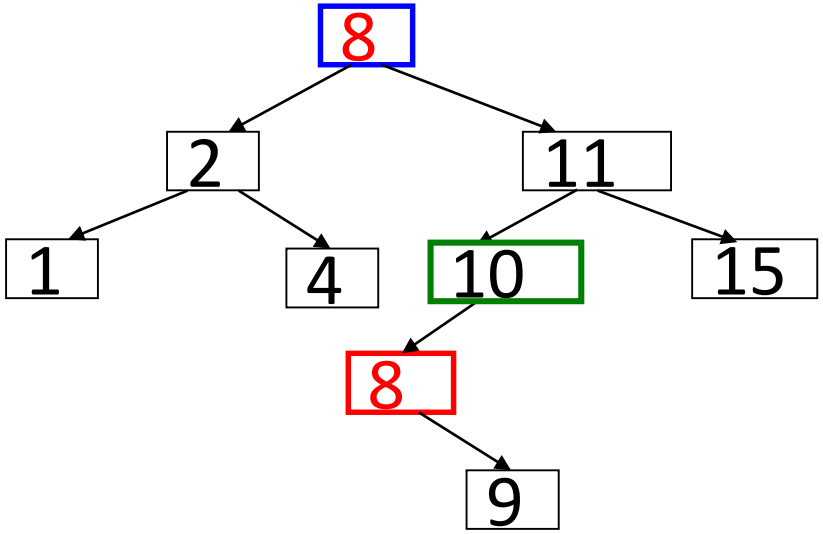
```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur;
  prev = cur;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

Usuwanie - przykład

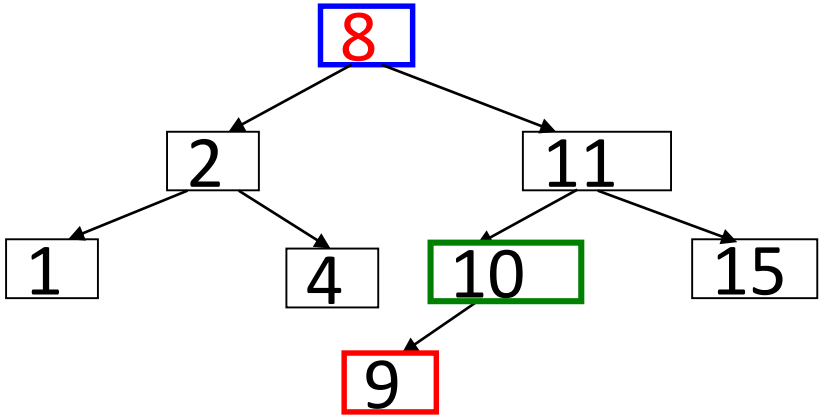
```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

Usuwanie - przykład

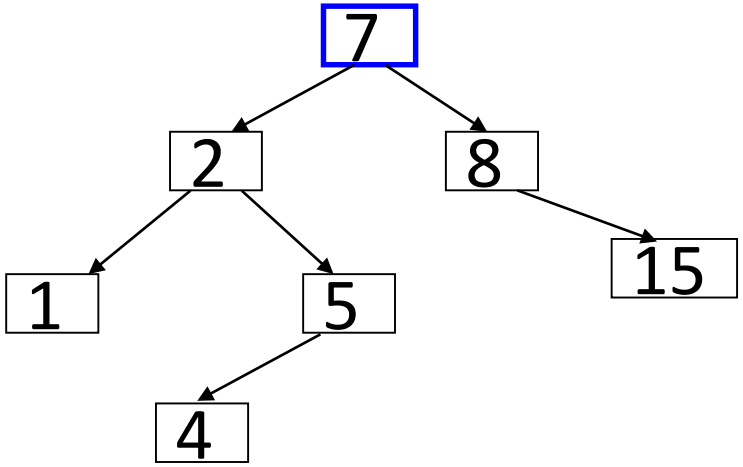
```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur,prev;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

Usuwanie - przykład

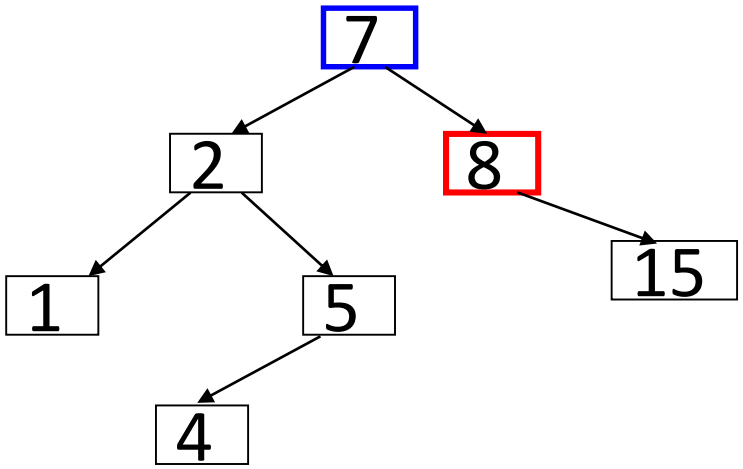
```
pnode deletekey( pnode root, int dkey)
{ pnode pom;
  if (root!=NULL)
    if (dkey < root->val) root->left = deletekey(root->left, dkey); // 2
    else
      if (dkey > root->val) root->right = deletekey(root->right, dkey); //3
      else
         if (root->left == NULL){ // 4a
           pom=root->right;  free(root); root=pom;
          } else
           if (root->right == NULL){ //4b
             pom=root->left; free(root); root = pom;
            } else
             delhlp(root); // 4c - oba poddrzewa niepuste
  return root;
}
```



deletekey(root,7)

Usuwanie - przykład

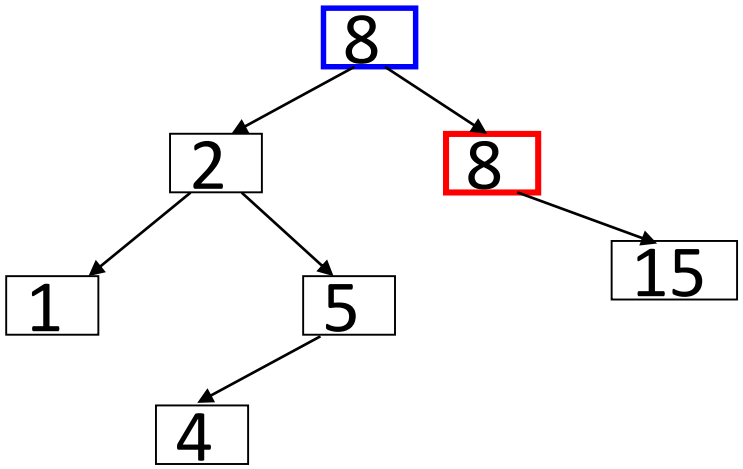
```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur, prev;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

Usuwanie - przykład

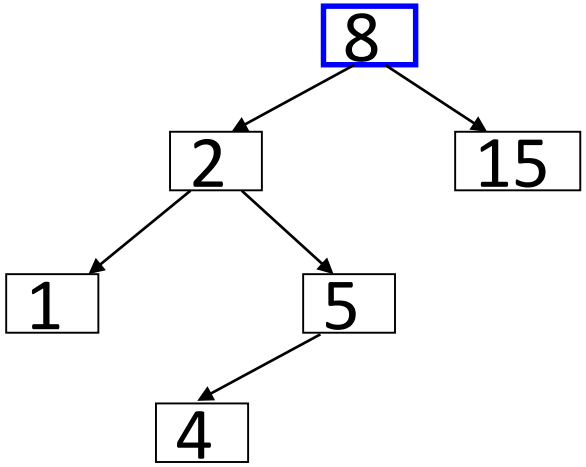
```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur, prev;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

Usuwanie - przykład

```
void delhlp(pnode root) //FUNKCJA POMOCNICZA!
{ pnode cur, prev;
  prev = root;
  cur = root->right;
  while (cur->left!=NULL) { // cur nie jest najmniejszy
    prev = cur; cur = cur->left;
  }
  root->val = cur->val;
  if (prev!=root) prev->left = cur->right;
  else root->right = cur->right;
  free(cur);
}
```



deletekey(root,7)

BST – złożoność operacji

Oznaczenie: n – liczba węzłów w drzewie

Fakt.
Złożoność czasowa (najgorszego przypadku) operacji:
- dodaj nowy element
- usuń element
- znajdź element

jest O(h), gdzie h to **wysokość drzewa**.

Fakt. Wypisanie elementów w porządku dla drzewa o n węzłach wymaga czasu O(*n*).

Fakt
Wysokość drzewa o *n* węzłach:
- najmniejsza: $\lceil \log (n+1) \rceil$
- największa: *n*

BST – złożoność operacji

Wniosek.
Złożoność czasowa (najgorszego przypadku) operacji dodaj nowy element, usuń element, znajdź element
jest O(n), gdzie n to **liczba węzłów w drzewie**.

Pytanie:
Dlaczego drzewa BST, skoro złożoność nie jest lepsza niż w przypadku list lub tablic?

Odp.:
1. Zazwyczaj (czyli przy „losowej” kolejności wstawiania/usuwania elementów) wysokość drzewa dużo mniejsza niż liczba węzłów.
2. Można zaimplementować operacje na BST tak, aby złożoność wyszukiwania, wstawiania i usuwania wynosiła O(log n) [algorytmy i struktury danych – II rok studiów]

Drzewa, BST, słowniki - podsumowanie

Słownik.
Abstrakcyjna struktura danych do (szybkiego) wyszukiwania, wstawiania, usuwania,…

Drzewo
Dynamiczna struktura danych, oparta na wskaźnikach

Drzewo BST
Struktura danych oparta na drzewach, operacje wstawiania, usuwania, wyszukiwania („szybkie” implementacje nie są omawiane na tym przedmiocie).

Usprawnienia i alternatywne implementacje
- bez rekurencji?
- użycie wartownika?
- rotacje… (AVL, drzewa czerwono-czarne, …)

Drzewa, BST, słowniki - podsumowanie

Usprawnienia i alternatywne implementacje

- bez rekurencji?
- użycie wartownika?
- **rotacje… (AVL, drzewa czerwono-czarne, …) – umożliwiają realizację**
  - **Wstawianie – O(log n)**
  - **Usuwanie – O(log n)**
  - **Wyszukiwanie – O(log n)**
  - **Wypisanie elementów w kolejności niemalejącej – O(n)**