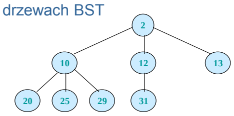
Algorytmy i struktury danych

Wykład 4: Struktury drzewiaste

- Drzewa binarne i wielokierunkowe
- Drzewa typu BST
- Podstawowe operacje na drzewach BST



Drzewiaste struktury danych

Drzewiastą strukturą danych (drzewem) nazywamy strukturę danych ***S=(D, R, e)***, w której relacja porządkująca ***N*** opisuje hierarchiczne powiązania pomiędzy węzłami drzewa, tworzącymi kolejne „poddrzewa”.

Uwagi:

- Drzewo ze swojej natury jest strukturą hierarchiczną (rekurencyjną).
- Niezwykle istotne jest tutaj odpowiednie przypisanie danych elementarnych ze zbioru ***D*** do kolejnych poziomów drzewa i zdefiniowanie relacji ***N*** (określającej porządek w drzewie)

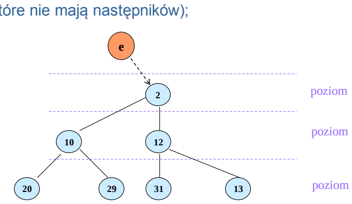
Drzewiaste struktury danych – pojęcia podstawowe

- **Korzeń drzewa** – element drzewa wskazywany przez element wejściowy ***e;*** założenie: jest tylko jeden korzeń drzewa.
- **Liść drzewa** – element drzewa, który nie posiada następnika w w sensie relacji ***N***,
- **Stopień drzewa** – maksymalna liczba możliwych następników dla dowolnego węzłami drzewa; często przyjmuje się, że stopień drzewa jest potęgą liczby 2 (drzewa dwójkowe, czwórkowe, ósemkowe, szesnastkowe),
- **Droga w drzewie** – ciąg kolejnych łuków pomiędzy dwoma węzłami drzewa, które trzeba pokonać, aby dojść od jednego elementu (węzła) drzewa do innego
- **Poziom drzewa** – węzły ułożone w tej samej odległości (długości drogi) od korzenia drzewa,
- **Drzewo zupełne** – takie drzewo, którego wszystkie węzły (oprócz liści) mają taką liczbę następników, ile wynosi stopień drzewa
- **Wysokość drzewa** – liczba poziomów drzewa (drzewo puste ma wysokość h=0; drzewo złożone z jednego węzła (korzenia) ma wysokość h=1 itd.)

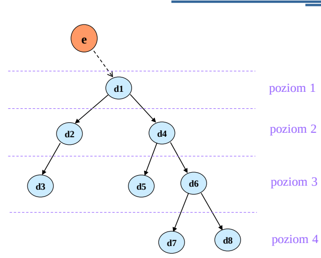
Struktury drzewiaste

Drzewo binarne (dwójkowe):

- drzewo o stopniu 2 (każdy węzeł ma co najwyżej dwa następniki);
- na ostatnim (najniższym) poziomie drzewa są ***liście*** (elementy, które nie mają następników);

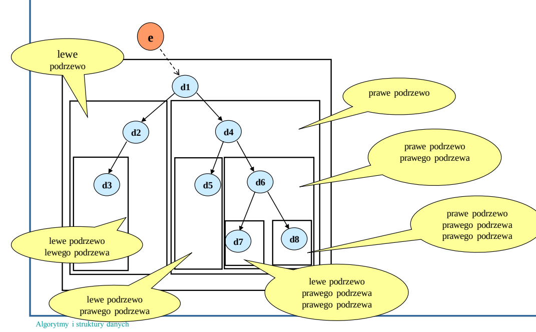


Przykład modelu grafowego drzewa binarnego



Dla powyższego drzewa wskaż: korzeń, liście, opisz zbiór ***D***, relacje r_{we} i *N*

Rekurencja w drzewie

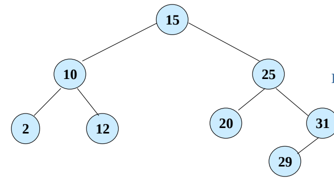


Drzewa poszukiwań binarnych (BST)

- Drzewo poszukiwań binarnych (BST – **B**inary **S**earch **T**ree):
  - dla każdego węzła (nie będącego liściem) wszystkie wartości przechowywane w jego lewym poddrzewie są **mniejsze**, natomiast wszystkie wartości przechowywane w prawym poddrzewie są **większe** od wartości w tym węźle (drzewo BST zbudowane w oparciu o relację **”<”**);

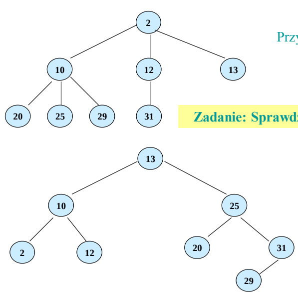
  lub
  - dla każdego węzła (nie będącego liściem) wszystkie wartości przechowywane w lewym poddrzewie są **większe**, natomiast wszystkie wartości przechowywane w prawym poddrzewie są **mniejsze** od wartości w tym węźle (drzewo BST zbudowane w oparciu o relację **”>”**);

Drzewa poszukiwań binarnych (BST)



Drzewo BST zbudowane w oparciu o relację ”<”

Drzewa poszukiwań binarnych (BST)



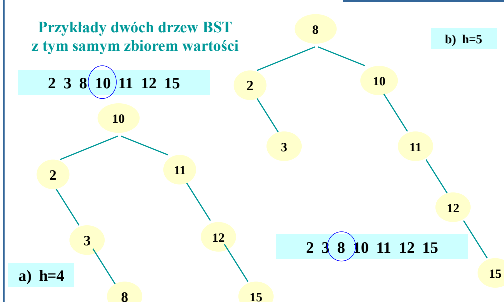
Przykładowe drzewo (trójkowe)

Zadanie: Sprawdzić, czy liczba *31* jest na liściu

Drzewo BST

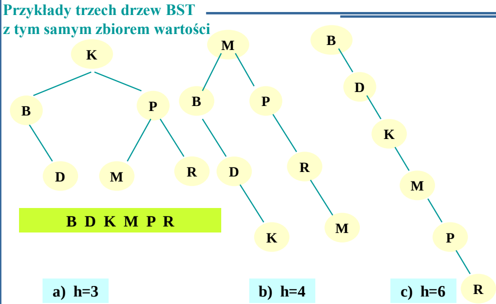
Drzewa poszukiwań binarnych (BST)

Przykłady dwóch drzew BST z tym samym zbiorem wartości



Drzewa poszukiwań binarnych (BST)

Przykłady trzech drzew BST z tym samym zbiorem wartości



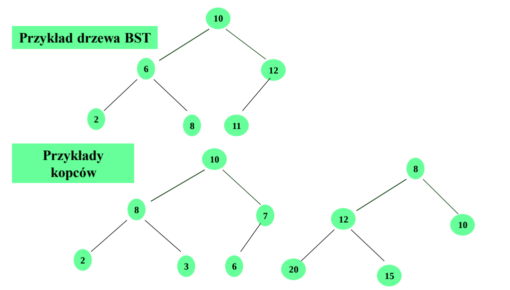
Drzewa poszukiwań binarnych

- Ograniczenia w drzewie poszukiwań binarnych (BST):
  - Podstawowe operacje realizują się szybko (O(log n)), jeśli drzewo jest zrównoważone
  - Jeśli będziemy do drzewa dodawać elementy, drzewo może rozrosnąć się w jedną ze stron (może w skrajnym przypadku zostać zdegenerowane do listy, czyli tzw. *winorośli*)
  - złożoność przeszukiwania takiego „zdegenerowanego” drzewa (a tym samym wszystkich innych operacji) jest liniowa O(n)
  - Aby równoważyć drzewa BST wymyślono ich różne odmiany, np. drzewa AVL lub tzw. drzewa kolorowe, np. drzewa czerwono-czarne
  - Dzięki zrównoważeniu nie tracimy podstawowej zalety struktury drzewiastej: *mniejszej niż liniowa złożoności obliczeniowej*

Rodzaje drzew binarnych

- **Drzewo binarne**
  Każdy węzeł, z wyjątkiem liści, ma dokładnie dwa następniki;
- **Drzewo poszukiwań binarnych (BST)**
  - drzewo binarne uporządkowane wg relacji < lub >
- **Drzewo AVL** (1962 – Adelson-Velskii, Landis)
  - drzewo BST jest drzewem AVL wtedy, kiedy dla każdego wierzchołka wysokości jego dwóch poddrzew (lewego i prawego) różnią się co najwyżej o 1;
- **Kopiec (sterta, stóg)**
  - wartości przechowywane w następnikach każdego węzła są **mniejsze** od wartości w danym węźle (tzw. **kopiec maksymalny**)
  - wartości przechowywane w następnikach każdego węzła są **większe** od wartości w danym węźle (tzw. **kopiec minimalny**)
  - drzewo jest szczelnie wypełniane od lewego poddrzewa;

Przykłady drzew binarnych



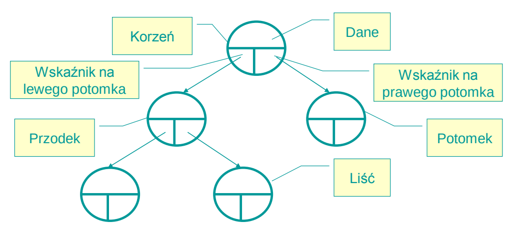
Kopiec binarny maksymalny   Kopiec binarny minimalny

Przetwarzanie drzew BST

- Podstawowe operacje na drzewach **binarnych**:
  - szukanie elementu w drzewie,
  - przechodzenie (linearyzacja) drzewa,
  - dodawanie elementu do drzewa,
  - usuwanie elementu z drzewa;
- Uwaga:
  - Operacje te często są realizowane rekurencyjnie

Element struktury drzewiastej

- Element drzewa zawiera:
  - Dane elementarne,
  - Realizację relacji następstwa – wskaźniki na następniki



Dynamiczne realizacje struktur drzewiastych

- Deklaracja elementu drzewa binarnego:

```
struct Node {
    int      data;
    struct Node *llink;
    struct Node *rlink;
};
```

dane elementarne

wskaźnik na lewy następnik

wskaźnik na prawy następnik

typedef struct Node *NodePtr;

- pomocniczy typ wskaźnikowy do struktury 'Node'
- np. zmienna tego typu będzie wskazywać korzeń

Algorytm szukania elementu w drzewie BST

- Cel:
  - uzyskanie adresu szukanego węzła;
- Dane wejściowe:
  - adres korzenia drzewa 'Root';
  - kryterium poszukiwania, np. wartość danej elementarnej;
- Uwagi:
  - kolejność przeszukiwania dowolna – w skrajnym przypadku należy przejrzeć wszystkie węzły w drzewie (złożoność *O(n)*);
  - stosowane rozwiązania: iteracja lub rekurencja;

Algorytm szukania elementu w drzewie BST

1. Ustaw aktualne dowiązanie na korzeń drzewa;
2. Dopóki aktualne dowiązanie jest różne od NULL:
   1) Jeżeli wartość szukana jest mniejsza od danej aktualnego węzła, to szukaj w jego lewym poddrzewie;
   2) Jeżeli wartość szukana jest większa od danej aktualnego węzła, to szukaj w jego prawym poddrzewie;
   3) Jeżeli wartość szukana jest równa danej aktualnego węzła, to koniec – zwróć dowiązanie do aktualnego węzła;

Algorytm szukania elementu w drzewie BST

- Wersja iteracyjna:

```
NodePtr find (int inValue, NodePtr node)
{
  while (node) {
    if (inValue == node -> data)
      return node;
    else if (inValue < node -> data)
      node = node -> llink;
    else if (inValue > node -> data)
      node = node -> rlink;
  }
  return NULL;
}
```

przejście do lewego poddrzewa

przejście do prawego poddrzewa

Algorytm szukania elementu w drzewie binarnym

- Wersja rekurencyjna:

```
NodePtr find (int inValue, NodePtr node)
{
  if (node) {
    if (inValue == node -> data)
      return node;
    else if (inValue < node -> data)
      return find (inValue, node -> llink);
    else if (inValue > node -> data)
      return find (inValue, node -> rlink);
  else return NULL;
}
```

wywołanie procedury dla lewego poddrzewa

wywołanie procedury dla prawego poddrzewa

Algorytm przechodzenia drzewa binarnego

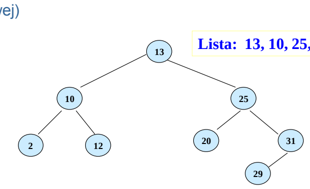
- Cel:
  - <u>jednokrotne</u> (nie licząc tzw. tranzytu) „odwiedzenie” każdego węzła drzewa;
  - można je interpretować jako umieszczenie wszystkich węzłów w jednej linii – linearyzacja drzewa;
- Dane wejściowe:
  - adres korzenia drzewa ‘Root’;
- Uwagi:
  - kolejność przejścia dowolna – liczba możliwych przejść w drzewie o *n* węzłach wynosi *n!* (liczba permutacji);
  - najczęściej stosowane sposoby przechodzenia: ***wszerz*** i ***w głąb***;

Sposoby przechodzenia drzewa binarnego

Przechodzenie *wszerz*

- Polega na odwiedzaniu kolejno każdego węzła od najwyższego (najniższego) poziomu i przechodzeniu kolejno po tych poziomach od góry w dół (od dołu do góry) i od lewej do prawej (od prawej do lewej)

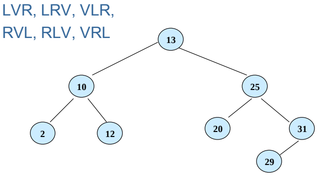
Lista: 13, 10, 25, 2, 12, 20, 31, 29



Sposoby przechodzenia drzewa binarnego

Przechodzenie *w głąb*

- Polega na przejściu jak najdalej w lewo (prawo), następnie powrocie do pierwszego rozwidlenia, przejściu jeden krok w prawo (lewo) itd.
- W zależności od momentu rejestrowania (wypisywania) wartości wyróżnia się 6 sposobów przechodzenia drzewa w głąb:
  - LVR, LRV, VLR,
  - RVL, RLV, VRL

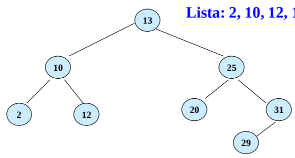


Sposoby przechodzenia drzewa binarnego

Przechodzenie *w głąb (LVR)*

- Polega na przejściu jak najdalej w lewo (prawo), następnie powrocie do pierwszego rozwidlenia, przejściu jeden krok w prawo (lewo) itd.

Lista: 2, 10, 12, 13, 20, 25, 29, 31



Algorytm przechodzenia drzewa binarnego *w głąb*

- Wersja „inorder” – LVR (porządek symetryczny)
  1. Przejście do lewego poddrzewa (L);
  2. Odwiedzenie węzła (V);
  3. Przejście do prawego poddrzewa (R);
- Wersja „preorder” – VLR (porządek prosty)
  1. Odwiedzenie węzła (V);
  2. Przejście do lewego poddrzewa (L);
  3. Przejście do prawego poddrzewa (R);
- Wersja „postorder” – LRV (porządek odwrotny)
  1. Przejście do lewego poddrzewa (L);
  2. Przejście do prawego poddrzewa (R);
  3. Odwiedzenie węzła (V);
- Inne możliwości: RVL, VRL, RLV

Algorytm przechodzenia drzewa binarnego *w głąb*

- Wersja procedury „inorder” – LVR
  - porządek symetryczny (lewe-korzeń-prawe)

```
void inorder (NodePtr node)
{
  if (node)
  {
    inorder (node -> llink);
    visit (node);
    inorder (node -> rlink);
  }
}
```


Algorytm przechodzenia drzewa binarnego *w głąb*

- Wersja procedury „preorder” – VLR
  - porządek prosty (korzeń-lewe-prawe)

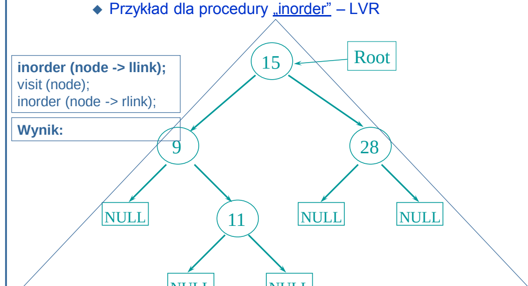
```
void preorder (NodePtr node)
{
  if (node)
  {
    visit (node);
    preorder (node -> llink);
    preorder (node -> rlink);
  }
}
```


- Wersja procedury „postorder” – LRV
  - porządek odwrotny (lewe-prawe-korzeń)

```
void postorder (NodePtr node)
{
  if (node)
  {
    postorder (node -> llink);
    postorder (node -> rlink);
    visit (node);
  }
}
```


Algorytm przechodzenia drzewa binarnego *w głąb*

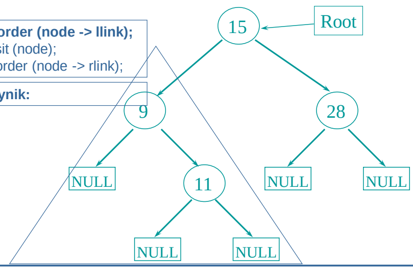
- Przykład dla procedury „inorder” – LVR



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

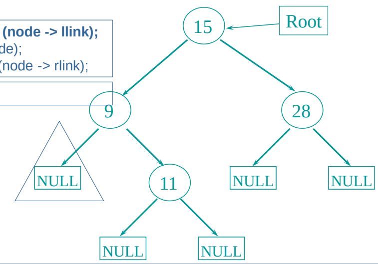
Wynik:



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

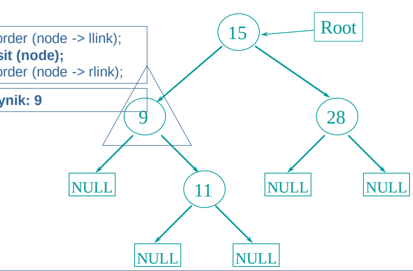
Wynik:



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

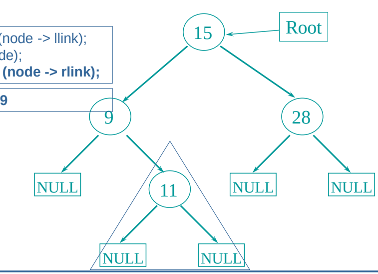
Wynik: 9



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

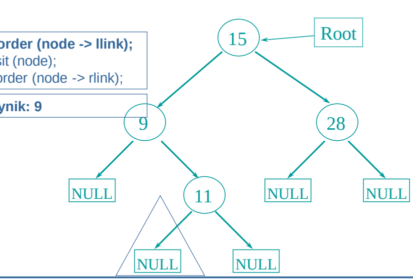
Wynik: 9



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

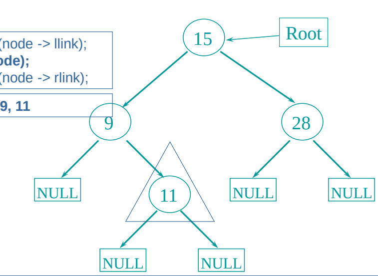
Wynik: 9



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

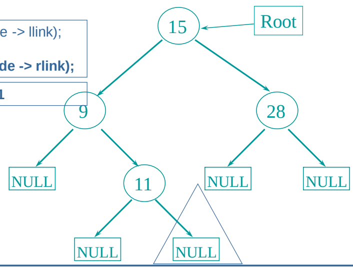
Wynik: 9, 11



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

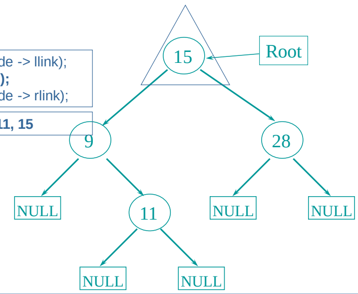
Wynik: 9, 11



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

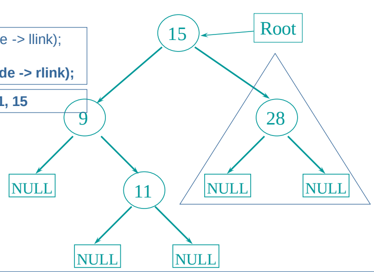
Wynik: 9, 11, 15



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

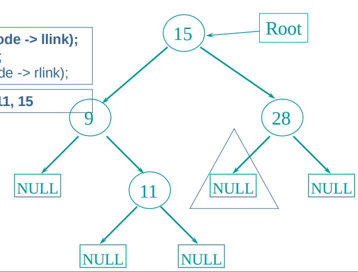
Wynik: 9, 11, 15



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

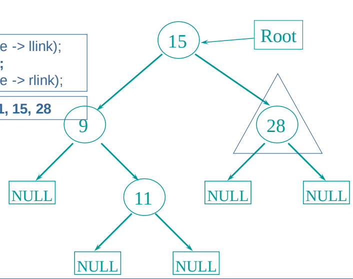
Wynik: 9, 11, 15



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

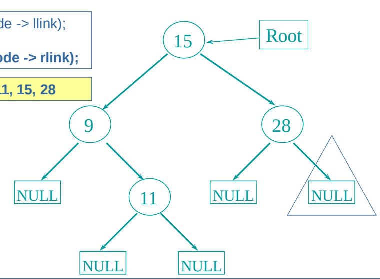
Wynik: 9, 11, 15, 28



Algorytm przechodzenia drzewa binarnego *w głąb*

inorder (node -> llink);
visit (node);
inorder (node -> rlink);

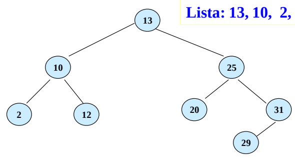
Wynik: 9, 11, 15, 28



Sposoby przechodzenia drzewa binarnego

- Przechodzenie *w głąb (VLR)*
- Polega na przejściu jak najdalej w lewo (prawo), następnie powrocie do pierwszego rozwidlenia, przejściu jeden krok w prawo (lewo) itd.

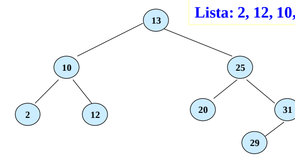
Lista: 13, 10, 2, 12, 25, 20, 31, 29



Sposoby przechodzenia drzewa binarnego

- Przechodzenie *w głąb (LRV)*
- Polega na przejściu jak najdalej w lewo (prawo), następnie powrocie do pierwszego rozwidlenia, przejściu jeden krok w prawo (lewo) itd.

Lista: 2, 12, 10, 20, 29, 31, 25, 13



Algorytm dodawania elementu do drzewa BST

- Cel:
  - dodanie nowego elementu do drzewa;
- Dane wejściowe:
  - adres korzenia drzewa ‘Root’;
  - nowe dane elementarne;
- Dane wyjściowe:
  - drzewo z dodanym elementem;
  - wskaźnik na nowy element;

Algorytm dołączania elementu do drzewa BST

1. Utwórz element i ustal dane elementarne;
2. Znajdź miejsce wstawienia elementu w drzewie;
3. Wstaw element do drzewa:
   - Wstaw element jako korzeń drzewa;

   lub
   - Wstaw element we wskazane miejsce w drzewie;

Algorytm dołączania elementu do drzewa BST

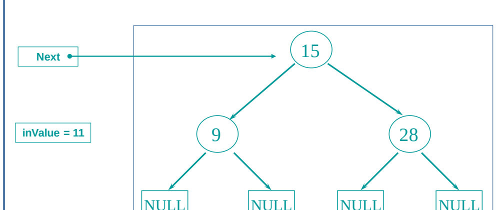
```
NodePtr Insert (int inValue, NodePtr next)
{
   if (next == NULL ) {
      next = (Node *) malloc(sizeof(Node));
      next -> llink = NULL;
      next -> rlink = NULL;
      next -> data = inValue;
   }
   else if ( inValue  <   next -> data )
     next -> llink = Insert( inValue, next -> llink ) ;
   else if ( inValue  >   next -> data )
     next -> rlink = Insert( inValue, next -> rlink );
   return next
}
```

rekurencja

Algorytm dołączania elementu do drzewa BST

- Przykład: Wstawienie do drzewa elementu z wartością *11*

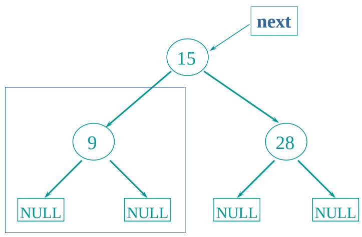
  Insert (11, Next);



Algorytm dołączania elementu do drzewa BST

- *inValue < next -> data*
  - Insert (11, next -> llink);

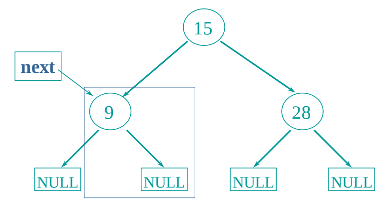
inValue = 11



Algorytm dołączania elementu do drzewa BST

- *inValue > next -> data*
  - Insert (11, next -> rlink);

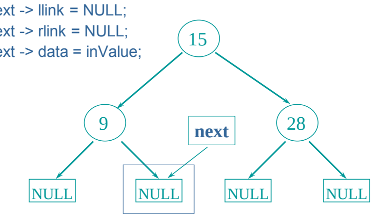
inValue = 11



Algorytm dołączania elementu do drzewa BST

```
if  (next == NULL )
{
  next = (Node *)malloc(sizeof(Node));
  next -> llink = NULL;
  next -> rlink = NULL;
  next -> data = inValue;
}
```

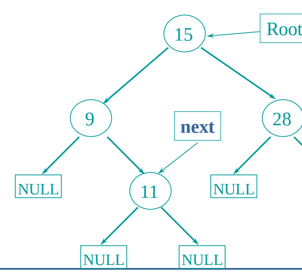
inValue = 11



Algorytm dołączania elementu do drzewa BST

- Drzewo po wstawieniu elementu z wartością '11';

inValue = 11

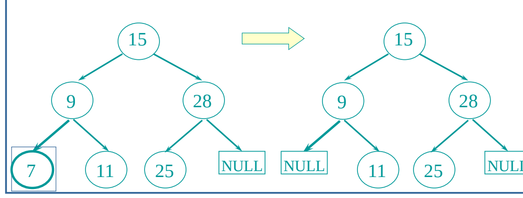


Algorytm usuwania elementu z drzewa BST

- Cel:
  - usunięcie węzła z drzewa;
- Dane wejściowe:
  - adres korzenia drzewa 'Root';
  - opis elementu usuwanego, np. wartość danej elementarnej;
- Uwagi:
  - Przypadek 1: węzeł jest liściem;
  - Przypadek 2: węzeł ma jednego potomka;
  - Przypadek 3: węzeł ma dwóch potomków;

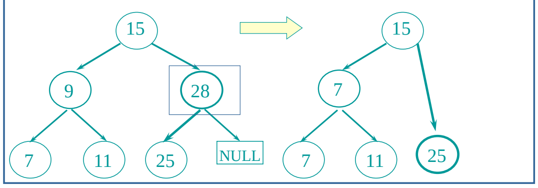
Algorytm usuwania elementu z drzewa BST

- **Przypadek 1**: węzeł jest liściem:
  1) Znajdź węzeł w drzewie;
  2) Usuń węzeł z drzewa;



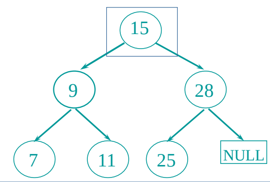
Algorytm usuwania elementu z drzewa BST

- **Przypadek 2**: węzeł ma jednego potomka:
  1) Znajdź węzeł w drzewie;
  2) Usuń węzeł z drzewa;
  3) Zastąp węzeł usunięty jego potomkiem (zmiana wskaźnika w poprzedniku węzła usuwanego)



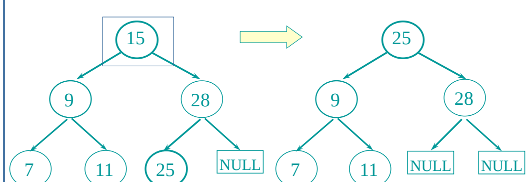
Algorytm usuwania elementu z drzewa BST

- **Przypadek 3**: węzeł ma dwóch potomków:
  1) Znajdź węzeł w drzewie;
  2) Usuń węzeł z drzewa;
  3) Zastąp węzeł usunięty: najmniejszym z prawego poddrzewa lub największym z lewego poddrzewa;



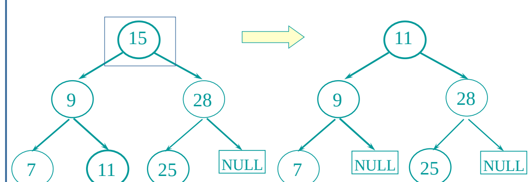
Algorytm usuwania elementu z drzewa BST

- Przypadek 3: węzeł ma dwóch potomków:
  - Wersja z przesunięciem najmniejszego elementu z prawego poddrzewa (skrajnie lewy wierzchołek tego poddrzewa);



Algorytm usuwania elementu z drzewa BST

- Przypadek 3: węzeł ma dwóch potomków:
  - Wersja z usunięciem największego elementu z lewego poddrzewa (skrajnie prawy wierzchołek);

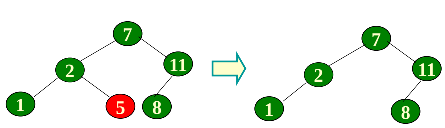


Algorytm usuwania elementu z drzewa BST

- **Podsumowanie**

Usunięcie węzła z drzewa BST prowadzi do jednego z trzech przypadków:

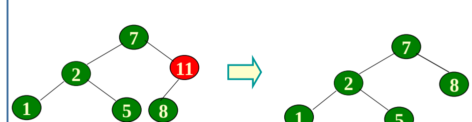
Przypadek 1



Uwaga:
Węzłem fizycznie usuniętym jest węzeł z wartością 5

Algorytm usuwania elementu z drzewa BST

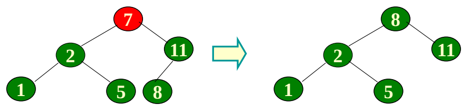
Przypadek 2



Uwaga:
Węzłem fizycznie usuniętym jest węzeł z wartością 11

Algorytm usuwania elementu z drzewa BST

Przypadek 3

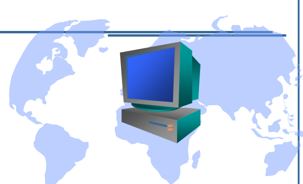


Uwaga:

Węzłem fizycznie usuniętym jest węzeł, który zawierał wartość 8

Prosty program do tworzenia i zobrazowania znakowego z drzewa BST

Program 4.1



Dziękuję za uwagę