Prezentowane materiały są przeznaczone dla uczniów szkół ponadgimnazjalnych.
Autor artykułu: mgr Jerzy Wałaszek, wersja 2.0

Drzewa poszukiwań binarnych – BST

Tematy pokrewne

Drzewa
Podstawowe pojęcia dotyczące drzew
Przechodzenie drzew binarnych – DFS: pre-order, in-order, post-order
Przechodzenie drzew binarnych – BFS
Badanie drzewa binarnego
Prezentacja drzew binarnych
Kopiec
Drzewa wyrażeń
Drzewa poszukiwań binarnych – BST
Tworzenie drzewa BST
Równoważenie drzewa BST – algorytm DSW
Proste zastosowania drzew BST
Drzewa AVL
Drzewa Splay
Drzewa Czerwono-Czarne
Kompresja Huffmana
Zbiory rozłączne – implementacja za pomocą drzew

Podrozdziały

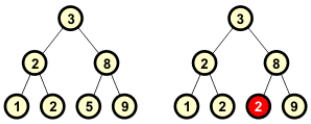
Budowa drzewa BST
Wyszukiwanie węzłów w drzewie BST
Następnik i poprzednik węzła w drzewie BST

Budowa drzewa BST

Drzewo poszukiwań binarnych (ang. Binary Search Tree) jest drzewem binarnym, w którym każdy węzeł spełnia reguły:

1. Jeśli węzeł posiada lewe poddrzewo (drzewo, którego korzeniem jest lewy syn), to wszystkie węzły w tym poddrzewie mają wartość niewiększą od wartości danego węzła.
2. Jeśli węzeł posiada prawe poddrzewo, to wszystkie węzły w tym poddrzewie są niemniejsze od wartości danego węzła.

Innymi słowy, przejście in-order tego drzewa daje ciąg wartości niemalejących.



Drzewo BST
in-order: 1 2 2 3 5 8 9

Drzewo nie BST
in-order: 1 2 2 3 **2** 8 9

Powyżej mamy przykład drzewa BST oraz drzewa nie będącego drzewem BST – zaznaczony liść 2 należy do prawego poddrzewa korzenia 2 i jest od niego mniejszy, a w prawym poddrzewie muszą być tylko węzły równe lub większe od korzenia.

Węzły w drzewie BST zawierają trzy wskaźniki, klucz oraz dane:

Lazarus

```
type
  PBSTNode = ^BSTNode;
  BSTNode  = record
    up    : PBSTNode;
    left  : PBSTNode;
    right : PBSTNode;
    key   : integer;
    data  : typ_danych;
  end;
```

Code::Blocks

```
struct BSTNode
{
  BSTNode * up;
  BSTNode * left;
  BSTNode * right;
  int key;
  typ_danych data;
};
```

Free Basic

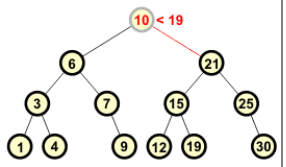
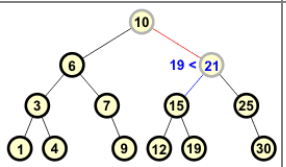
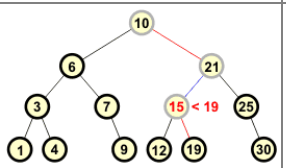
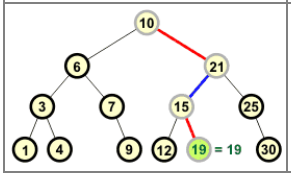
```
Type BSTNode
  up    As BSTNode Ptr
  Left  As BSTNode Ptr
  Right As BSTNode Ptr
  key   As Integer
  Data  As typ_danych
End Type
```

up – wskazanie ojca węzła
left – wskazanie lewego syna
right – wskazanie prawego syna
key – klucz, wg którego węzły są uporządkowane na drzewie BST
data – dowolne dane

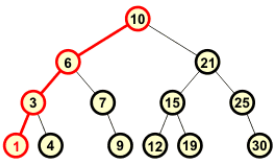
Dla niektórych drzew BST klucz może być daną, wtedy nie występuje pole data.

Wyszukiwanie węzłów w drzewie BST

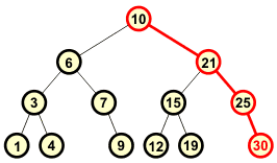
Drzewa BST pozwalają wyszukiwać zawarte w nich elementy z klasą złożoności obliczeniowej O(log n), gdzie *n* oznacza liczbę węzłów. Prześledźmy sposób wyszukiwania węzła o kluczu 19 w drzewie BST:

Stan	Opis
	Wyszukiwanie rozpoczynamy od korzenia drzewa. Porównujemy wartość węzła z wartością poszukiwaną. Ponieważ jest ona większa od wartości korzenia, idziemy wzdłuż prawej krawędzi do prawego syna (jeśli węzeł nie miałby prawego syna, to oznaczałoby to, że poszukiwanej wartości nie ma w drzewie BST.)
	W węźle 21 ponownie dokonujemy porównania. Ponieważ poszukiwany węzeł jest mniejszy od 21, wybieramy gałąź lewą i przechodzimy do lewego syna 15.
	Porównujemy węzeł 15 z poszukiwanym 19. Ponieważ 19 jest większe, idziemy prawą krawędzią do prawego syna 19.
	Porównujemy węzeł 19 z poszukiwanym. Są równe. Wyszukiwanie zakończone.

Z przedstawionego powyżej schematu wynikają dwa proste wnioski:



Element najmniejszy w drzewie znajdziemy, idąc lewymi gałęziami aż do elementu, który nie posiada już lewego syna



Element największy w drzewie znajdziemy, idąc prawymi gałęziami aż do elementu, który nie posiada już prawego syna.

W obu przypadkach nie musimy porównywać węzłów.

Algorytm wyszukiwania węzła w drzewie BST – findBST(root,k)

Wejście

k – klucz poszukiwanego węzła
p – wskazanie korzenia drzewa BST

Wyjście:

Wskazanie węzła na drzewie BST o kluczu *k* lub **nil**, jeśli drzewo nie zawiera takiego węzła

Lista kroków:

K01: **Dopóki** ($p \neq$ **nil**) $\wedge$ (($p \rightarrow key$) $\neq k$), **wykonuj** K02 *; węzła brak lub został znaleziony*
K02: **Jeśli** $k < (p \rightarrow key)$, **to** $p \leftarrow (p \rightarrow left)$ *; decydujemy, którą drogą pójść: w lewo czy w prawo*
Inaczej $p \leftarrow (p \rightarrow right)$
K03: **Zakończ z wynikiem** *p*

Algorytm wyszukiwania w drzewie BST węzła o najmniejszym kluczu – minBST(p)

Wejście

p – wskazanie korzenia drzewa BST

Wyjście:

Wskazanie węzła o najmniejszym kluczu lub **nil**, jeśli drzewo jest puste.

Lista kroków:

K01: **Jeśli** $p =$ **nil**, **to idź do** K03
K02: **Dopóki** ($p \rightarrow left$) $\neq$ **nil**, **wykonuj:** $p \leftarrow (p \rightarrow left)$ *; szukamy węzła bez lewego syna*
K03: **Zakończ z wynikiem** *p*

Algorytm wyszukiwania w drzewie BST węzła o największym kluczu – maxBST(p)

Wejście

p – wskazanie korzenia drzewa BST

Wyjście:

Wskazanie węzła o największym kluczu lub **nil**, jeśli drzewo jest puste.

Lista kroków:

K01: **Jeśli** *p* = **nil**, **to idź do** K03
K02: **Dopóki** (*p*→*right*) ≠ **nil**, **wykonuj:** *p* ← (*p*→*right*) *; szukamy węzła bez prawego syna*
K03: **Zakończ z wynikiem** *p*

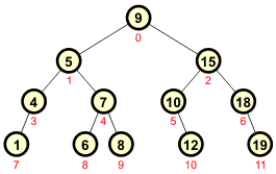
Program

Ważne:
Zanim uruchomisz program, przeczytaj wstęp do tego artykułu, w którym wyjaśniamy funkcje tych programów oraz sposób korzystania z nich.

Program wczytuje definicję drzewa binarnego, którego klucze są liczbami całkowitymi. Wyznacza klucz minimalny i maksymalny, a następnie idąc kolejno przez wartości kluczy od min do max wyświetla informacje o znalezionym węźle.

Sposób wprowadzania drzew binarnych opisaliśmy w rozdziale o badaniu drzew binarnych. Tutaj krótko przypomnijmy, że węzły drzewa są numerowane od strony lewej do prawej na kolejnych poziomach. Pierwszą liczbą jest ilość wierzchołków *n*. Kolejne *n* wierszy zawiera 3 liczby określające kolejne wierzchołki. Pierwsza liczba w trójce określa klucz węzła. Pozostałe dwie liczby określają kolejno numer węzła będącego lewym i prawym synem. Jeśli węzeł nie ma któregoś z synów, to numer syna przyjmuje wartość zero. W strukturach węzłów nie będziemy wykorzystywać pola up, które tutaj nie będzie nam potrzebne, ponieważ będziemy poruszać się tylko w dół drzewa.

Przykładowe dane dla programu:




```
12
9 1 2
5 3 4
15 5 6
4 7 0
7 8 9
10 0 10
18 0 11
1 0 0
6 0 0
8 0 0
12 0 0
19 0 0
```

Lazarus

```
// Wyszukiwanie w drzewie BST
// Data: 24.04.2013
// (C)2013 mgr Jerzy Wałaszek
//------------------------------

program bst;

// Typ węzłów drzewa BST

type
  PBSTNode = ^BSTNode;
  BSTNode  = record
    left  : PBSTNode;
    right : PBSTNode;
    key   : integer;
  end;

// Funkcja wczytuje drzewo BST ze standardowego
// wejścia i zwraca wskazanie korzenia.
//-----------------------------------------------
function readBST : PBSTNode;
var
  vp : array of PBSTNode;   // Tablica wskazań węzłów
  key,l,r,i,n : integer;

begin

  read(n);                  // Odczytujemy liczbę węzłów drzewa

  SetLength(vp,n);          // Tworzymy dynamiczną tablicę wskazań węzłów

  // Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  // które również tworzymy dynamicznie

  for i := 0 to n - 1 do new(vp[i]);

  // Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  // w pamięci wypełniając odpowiednie pola węzłów.

  for i := 0 to n - 1 do
  begin

    read(key,l,r);          // Czytamy klucz, numer lewego i prawego syna

    vp[i]^.key := key;      // Ustawiamy klucz

    if l > 0 then vp[i]^.left  := vp[l]  // Ustawiamy lewego syna
    else          vp[i]^.left  := nil;

    if r > 0 then vp[i]^.right := vp[r]  // Ustawiamy prawego syna
    else          vp[i]^.right := nil;

  end;

  readBST := vp[0];         // Zapamiętujemy korzeń

  SetLength(vp,0);          // Usuwamy tablicę dynamiczną

end;

// Funkcja szuka w drzewie BST węzła o zadanym kluczu.
// Jeśli go znajdzie, zwraca jego wskazanie. Jeżeli nie,
// to zwraca wskazanie puste.
```

```
// Parametrami są:
// p - wskazanie korzenia drzewa BST
// k - klucz poszukiwanego węzła
//------------------------------------------------------
function findBST(p : PBSTNode; k : integer) : PBSTNode;
begin

  while (p <> nil) and (p^.key <> k) do
    if k < p^.key then p := p^.left
    else               p := p^.right;

  findBST := p;
end;

// Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//------------------------------------------------------
function minBST(p : PBSTNode) : PBSTNode;
begin
  if p <> nil then
    while p^.left <> nil do
      p := p^.left;

  minBST := p;
end;

// Funkcja zwraca wskazanie węzła o największym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//------------------------------------------------------
function maxBST(p : PBSTNode) : PBSTNode;
begin
  if p <> nil then
    while p^.right <> nil do
      p := p^.right;

  maxBST := p;
end;

// Procedura DFS:postorder usuwająca drzewo
//-----------------------------------------
procedure DFSRelease(v : PBSTNode);
begin
  if v <> nil then
  begin
    DFSRelease(v^.left);   // usuwamy lewe poddrzewo
    DFSRelease(v^.right);  // usuwamy prawe poddrzewo
    dispose(v);            // usuwamy sam węzeł
  end;
end;

// **********************
// *** PROGRAM GŁÓWNY ***
// **********************

var
  root,p : PBSTNode;
  k, mink, maxk : integer;

begin
  root := readBST;     // Odczytujemy drzewo BST

  if root <> nil then
  begin
    mink := minBST(root)^.key; // Odczytujemy klucz minimalny
    maxk := maxBST(root)^.key; // Odczytujemy klucz maksymalny

    // Przechodzimy przez kolejne wartości kluczy

    for k := mink to maxk do
    begin
      p := findBST(root,k);    // szukamy węzła o kluczu k

      write('KEY = ',k:3,' : ');

      if p <> nil then
      begin
        if (p^.left = nil) and (p^.right = nil) then
          writeln('LEAF')
        else
          writeln('INNER NODE');
      end
      else writeln('NONE');

    end;
  end
  else writeln('BST is empty!!!');

  DFSRelease(root);    // usuwamy drzewo z pamięci

end.
```

Code::Blocks

```
// Wyszukiwanie w drzewie BST
// Data: 24.04.2013
// (C)2013 mgr Jerzy Wałaszek
//-----------------------------

#include <iostream>
#include <iomanip>

using namespace std;

// Typ węzłów drzewa BST

struct BSTNode
{
  BSTNode * left;
  BSTNode * right;
  int key;
};
```

```
// Funkcja wczytuje drzewo BST ze standardowego
// wejścia i zwraca wskazanie korzenia.
//-----------------------------------------------
BSTNode * readBST()
{
  BSTNode ** vp;    // Tablica wskazań węzłów
  int key,l,r,i,n;

  cin >> n;                 // Odczytujemy liczbę węzłów drzewa

  vp = new BSTNode * [n];   // Tworzymy dynamiczną tablicę wskazań węzłów

  // Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  // które również tworzymy dynamicznie

  for(i = 0; i < n; i++) vp[i] = new BSTNode;

  // Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  // w pamięci wypełniając odpowiednie pola węzłów.

  for(i = 0; i < n; i++)
  {
    cin >> key >> l >> r;   // Czytamy klucz, numer lewego i prawego syna

    vp[i]->key = key;       // Ustawiamy klucz

    vp[i]->left  = l ? vp[l]: NULL;  // Ustawiamy lewego syna

    vp[i]->right = r ? vp[r]: NULL;  // Ustawiamy prawego syna

  }

  BSTNode * p = vp[0];      // Zapamiętujemy korzeń

  delete [] vp;             // Usuwamy tablicę dynamiczną

  return p;
}

// Funkcja szuka w drzewie BST węzła o zadanym kluczu.
// Jeśli go znajdzie, zwraca jego wskazanie. Jeżeli nie,
// to zwraca wskazanie puste.
// Parametrami są:
// p - wskazanie korzenia drzewa BST
// k - klucz poszukiwanego węzła
//-----------------------------------------------------
BSTNode * findBST(BSTNode * p, int k)
{
  while(p && p->key != k)
    p = (k < p->key) ? p->left: p->right;

  return p;
}

// Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//-----------------------------------------------------
BSTNode * minBST(BSTNode * p)
{
  if(p) while(p->left) p = p->left;

  return p;
}

// Funkcja zwraca wskazanie węzła o największym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//-----------------------------------------------------
BSTNode * maxBST(BSTNode * p)
{
  if(p) while(p->right) p = p->right;

  return p;
}

// Procedura DFS:postorder usuwająca drzewo
//-----------------------------------------
void DFSRelease(BSTNode * v)
{
  if(v)
  {
    DFSRelease(v->left);   // usuwamy lewe poddrzewo
    DFSRelease(v->right);  // usuwamy prawe poddrzewo
    delete v;              // usuwamy sam węzeł
  }
}

// **********************
// *** PROGRAM GŁÓWNY ***
// **********************

int main()
{
  BSTNode * root, * p;
  int k, mink, maxk;

  root = readBST();     // Odczytujemy drzewo BST

  if(root)
  {
    mink = minBST(root)->key; // Odczytujemy klucz minimalny
    maxk = maxBST(root)->key; // Odczytujemy klucz maksymalny

    // Przechodzimy przez kolejne wartości kluczy

    for(k = mink; k <= maxk; k++)
    {
      p = findBST(root,k);    // szukamy węzła o kluczu k

      cout << "KEY = " << setw(3) <<  k << " : ";

      if(p)
      {
```

```
        if (!p->left && !p->right) cout << "LEAF";
        else                       cout << "INNER NODE";
      }
      else cout << "NONE";

      cout << endl;
    }
  }
  else cout << "BST is empty!!!" << endl;

  DFSRelease(root);    // usuwamy drzewo z pamięci

  return 0;
}
```

Free Basic

```
' Wyszukiwanie w drzewie BST
' Data: 24.04.2013
' (C)2013 mgr Jerzy Wałaszek
'-----------------------------

' Typ węzłów drzewa BST

Type BSTNode
  Left  As BSTNode Ptr
  Right As BSTNode Ptr
  key   As Integer
End Type

' Funkcja wczytuje drzewo BST ze standardowego
' wejścia i zwraca wskazanie korzenia.
'-----------------------------------------------
Function readBST() As BSTNode Ptr

  Dim As BSTNode Ptr Ptr vp    ' Tablica wskazań węzłów
  Dim As Integer key,l,r,i,n

  Open Cons For Input As #1

  Input #1,n                   ' Odczytujemy liczbę węzłów drzewa

  vp = New BSTNode Ptr [n]     ' Tworzymy dynamiczną tablicę wskazań węzłów

  ' Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  ' które również tworzymy dynamicznie

  For i = 0 To n - 1
        vp[i] = New BSTNode
  Next

  ' Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  ' w pamięci wypełniając odpowiednie pola węzłów.

  For i = 0 To n - 1
    Input #1,key,l,r       ' Czytamy klucz, numer lewego i prawego syna

    vp[i]->key = key       ' Ustawiamy klucz

    If l > 0 Then
        vp[i]->Left = vp[l]  ' Ustawiamy lewego syna
    Else
        vp[i]->Left = 0
    End If

    If r > 0 Then
        vp[i]->Right = vp[r] ' Ustawiamy prawego syna
    Else
        vp[i]->Right = 0
    End If

  Next

  Close #1

  readBST = vp[0]         ' Zapamiętujemy korzeń

  Delete [] vp            ' Usuwamy tablicę dynamiczną

End Function

' Funkcja szuka w drzewie BST węzła o zadanym kluczu.
' Jeśli go znajdzie, zwraca jego wskazanie. Jeżeli nie,
' to zwraca wskazanie puste.
' Parametrami są:
' p - wskazanie korzenia drzewa BST
' k - klucz poszukiwanego węzła
'-----------------------------------------------------
Function findBST(p As BSTNode Ptr, k As Integer) As BSTNode Ptr

  While (p <> 0) Andalso (p->key <> k)
    If k < p->key Then
        p = p->Left
    Else
        p = p->Right
    End If
  Wend

  Return p

End Function

' Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
' Parametrem jest wskazanie korzenia drzewa BST.
'-----------------------------------------------------
Function minBST(p As BSTNode Ptr) As BSTNode Ptr

  If p Then
    While p->Left
        p = p->Left
    Wend
```

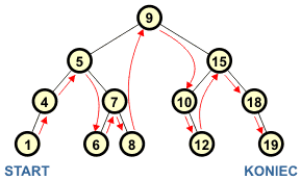
```
  End If

  Return p
End Function

' Funkcja zwraca wskazanie węzła o największym kluczu.
' Parametrem jest wskazanie korzenia drzewa BST.
'-----------------------------------------------------
Function maxBST(p As BSTNode Ptr) As BSTNode Ptr

  If p Then
    While p->Right
        p = p->Right
    Wend
  End If

  Return p
End Function

' Procedura DFS:postorder usuwająca drzewo
'-----------------------------------------
Sub DFSRelease(v As BSTNode Ptr)
  If v Then
    DFSRelease(v->Left)    ' usuwamy lewe poddrzewo
    DFSRelease(v->Right)   ' usuwamy prawe poddrzewo
    Delete v               ' usuwamy sam węzeł
  End If
End Sub

' **********************
' *** PROGRAM GŁÓWNY ***
' **********************

Dim As BSTNode Ptr root, p
Dim As Integer k, mink, maxk

root = readBST()     ' Odczytujemy drzewo BST

If root Then

  mink = minBST(root)->key ' Odczytujemy klucz minimalny
  maxk = maxBST(root)->key ' Odczytujemy klucz maksymalny

    ' Przechodzimy przez kolejne wartości kluczy

  For k = mink To maxk

    p = findBST(root,k)    ' szukamy węzła o kluczu k

    Print Using "KEY = ### : "; k;

    If p Then
      If (p->Left = 0) Andalso (p->Right = 0) Then
        Print "LEAF"
      Else
         Print "INNER NODE"
      End If

    Else
     Print "NONE"
    End If

  Next

Else

  Print "BST is empty!!!"

End If

DFSRelease(root)    ' usuwamy drzewo z pamięci

End
```

Wynik

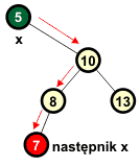
```
12
9 1 2
5 3 4
15 5 6
4 7 0
7 8 9
10 0 10
18 0 11
1 0 0
6 0 0
8 0 0
12 0 0
19 0 0
KEY =   1 : LEAF
KEY =   2 : NONE
KEY =   3 : NONE
KEY =   4 : INNER NODE
KEY =   5 : INNER NODE
KEY =   6 : LEAF
KEY =   7 : INNER NODE
KEY =   8 : LEAF
KEY =   9 : INNER NODE
KEY =  10 : INNER NODE
KEY =  11 : NONE
KEY =  12 : LEAF
KEY =  13 : NONE
KEY =  14 : NONE
KEY =  15 : INNER NODE
KEY =  16 : NONE
KEY =  17 : NONE
KEY =  18 : INNER NODE
KEY =  19 : LEAF
```

Następnik i poprzednik węzła w drzewie BST

Na początku rozdziału powiedzieliśmy, że kolejność węzłów w drzewie BST jest taka, iż w wyniku przejścia tego drzewa algorytmem in-order otrzymamy niemalejący ciąg kluczy. Przyjrzyjmy się dokładniej temu przejściu:

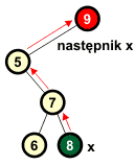


Zauważ, że znalezienie następnika wcale nie wymaga porównywania węzłów. Mogą wystąpić trzy przypadki:



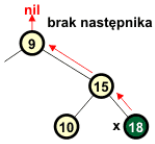
Przypadek 1:

Węzeł *x* posiada prawego syna – następnikiem jest wtedy węzeł o minimalnym kluczu w poddrzewie, którego korzeniem jest prawy syn. Wykorzystujemy tutaj algorytm wyszukiwania węzła o najmniejszym kluczu w prawym poddrzewie.



Przypadek 2:

Węzeł *x* nie posiada prawego syna. W takim przypadku, idąc w górę drzewa, musimy znaleźć pierwszego ojca, dla którego nasz węzeł leży w lewym poddrzewie. Tutaj również nie musimy porównywać węzłów. Po prostu idziemy w górę drzewa i w węźle nadrzędnym sprawdzamy, czy przyszliśmy od strony lewego syna. Jeśli tak, to węzeł ten jest następnikiem. Jeśli nie, to kontynuujemy marsz w górę drzewa. Wymaga to zapamiętywania adresów kolejno mijanych węzłów.



Przypadek 3:

Węzeł x nie posiada prawego syna. Idąc w górę drzewa, dochodzimy do korzenia, a następnie do adresu **nil**, który wskazuje pole *up* korzenia drzewa BST. W takim przypadku węzeł *x* jest węzłem o największym kluczu i nie posiada następnika.

Ponieważ będziemy musieli poruszać się w górę drzewa, węzły muszą posiadać pole *up* prowadzące do ojca.

Algorytm znajdowania następnika węzła w drzewie BST – succBST(p)

Wejście

p – wskazanie węzła na drzewie BST, dla którego poszukujemy następnika

Wyjście:

Wskazanie węzła będącego następnikiem węzła wejściowego lub nil, jeśli węzeł wejściowy nie ma następnika.

Zmienne pomocnicze:

r – wskazanie węzła
minBST(*w*) – znajduje element minimalny od węzła *w*

Lista kroków:

```
K01: Jeśli p = nil to zakończ z wynikiem p                    ; jeśli drzewo jest puste, kończymy
K02: Jeśli (p→right) ≠ nil, to zakończ z wynikiem minBST(p→right) ; Przypadek 1: zwracamy węzeł minimalny od prawego syna
K03: r ← (p→up)                                               ; r wskazuje ojca p
K04: Dopóki (r ≠ nil) ∧ (p = (r→right), wykonuj  K05...K06    ; Przypadki 2 i 3, brak prawego syna
K05:    p ← r                                                 ; przemieszczamy się w górę drzewa, aż trafimy na węzeł,
K06:    r ← (r→up)                                            ; dla którego p leży w lewej gałęzi
K07: Zakończ z wynikiem r                                     ; zwracamy znaleziony węzeł lub nil, jeśli następnika nie ma
```

Algorytm znajdowania poprzednika jest lustrzanym odbiciem algorytmu znajdowania następnika (dlaczego?):

Algorytm znajdowania poprzednika węzła w drzewie BST – predBST(p)

Wejście

p – wskazanie węzła na drzewie BST, dla którego poszukujemy poprzednika

Wyjście:

Wskazanie węzła będącego poprzednikiem węzła wejściowego lub nil, jeśli węzeł wejściowy nie ma następnika

Zmienne pomocnicze:

r – wskazanie węzła
maxBST(*w*) – znajduje element maksymalny od węzła *w*

Lista kroków:

```
K01: Jeśli p = nil to zakończ z wynikiem p                    ; jeśli drzewo jest puste, kończymy
K02: Jeśli (p→left) ≠ nil, to zakończ z wynikiem maxBST(p→left) ; Przypadek 1: zwracamy węzeł maksymalny od lewego syna
K03: r ← (p→up)                                               ; r wskazuje ojca p
K04: Dopóki (r ≠ nil) ∧ (p = (r→left), wykonuj  K05...K06     ; Przypadki 2 i 3, brak lewego syna
K05:    p ← r                                                 ; przemieszczamy się w górę drzewa, aż trafimy na węzeł,
K06:    r ← (r→up)                                            ; dla którego p leży w prawej gałęzi
K07: Zakończ z wynikiem r                                     ; zwracamy znaleziony węzeł lub nil, jeśli poprzednika nie ma
```

Program

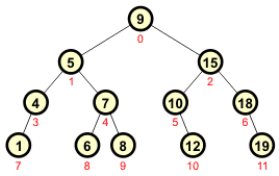
Ważne:

Zanim uruchomisz program, przeczytaj wstęp do tego artykułu, w którym wyjaśniamy funkcje tych programów oraz sposób korzystania z nich.

Program wczytuje definicję drzewa binarnego, którego klucze są liczbami całkowitymi. Definicja drzewa wygląda następująco:

Pierwsza liczba określa *n* – ilość węzłów w drzewie.
Kolejne *n* wierszy zawiera po cztery liczby: klucz, numer ojca (dla korzenia wartość jest nieistotna), numer lewego syna, numer prawego syna.

Program wyznacza kolejne następniki i poprzedniki korzenia. Przykładowe dane:




```
12
9 0 1 2
5 0 3 4
15 0 5 6
4 1 7 0
7 1 8 9
10 2 0 10
18 2 0 11
1 3 0 0
6 4 0 0
8 4 0 0
12 5 0 0
19 6 0 0
```

Lazarus

```
// Następnik i poprzednik w drzewie BST
// Data: 27.04.2013
// (C)2013 mgr Jerzy Wałaszek
//-------------------------------------

program sp_bst;

// Typ węzłów drzewa BST

type
  PBSTNode = ^BSTNode;
  BSTNode  = record
    up    : PBSTNode;
    left  : PBSTNode;
    right : PBSTNode;
    key   : integer;
  end;

// Funkcja wczytuje drzewo BST ze standardowego
// wejścia i zwraca wskazanie korzenia.
//-----------------------------------------------
function readBST : PBSTNode;
var
  vp : array of PBSTNode;   // Tablica wskazań węzłów
  key,u,l,r,i,n : integer;
begin

  read(n);                  // Odczytujemy liczbę węzłów drzewa

  SetLength(vp,n);          // Tworzymy dynamiczną tablicę wskazań węzłów

  // Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  // które również tworzymy dynamicznie

  for i := 0 to n - 1 do new(vp[i]);

  // Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  // w pamięci wypełniając odpowiednie pola węzłów.

  for i := 0 to n - 1 do
  begin

    read(key,u,l,r);        // Czytamy klucz, numery ojca, lewego i prawego syna

    vp[i]^.key := key;      // Ustawiamy klucz

    vp[i]^.up := vp[u];     // Ustawiamy ojca

    if l > 0 then vp[i]^.left  := vp[l]  // Ustawiamy lewego syna
    else          vp[i]^.left  := nil;

    if r > 0 then vp[i]^.right := vp[r]  // Ustawiamy prawego syna
    else          vp[i]^.right := nil;

  end;

  vp[0]^.up := nil;         // Korzeń nie posiada ojca

  readBST := vp[0];         // Zapamiętujemy korzeń

  SetLength(vp,0);          // Usuwamy tablicę dynamiczną

end;

// Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//-----------------------------------------------------
function minBST(p : PBSTNode) : PBSTNode;
begin
  if p <> nil then
    while p^.left <> nil do
      p := p^.left;

  minBST := p;
end;
```

```
// Funkcja zwraca wskazanie węzła o największym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//------------------------------------------------------
function maxBST(p : PBSTNode) : PBSTNode;
begin
  if p <> nil then
    while p^.right <> nil do
      p := p^.right;

  maxBST := p;
end;

// Funkcja znajduje następnik węzła p
//-----------------------------------
function succBST(p : PBSTNode) : PBSTNode;
var
  r : PBSTNode;
begin
  succBST := nil;
  if p <> nil then
  begin
    if p^.right <> nil then succBST := minBST(p^.right)
    else
    begin
      r := p^.up;
      while (r <> nil) and (p = r^.right) do
      begin
        p := r;
        r := r^.up;
      end;
      succBST := r;
    end;
  end;
end;

// Funkcja znajduje poprzednik węzła p
//------------------------------------
function predBST(p : PBSTNode) : PBSTNode;
var
  r : PBSTNode;
begin
  predBST := nil;
  if p <> nil then
  begin
    if p^.left <> nil then predBST := maxBST(p^.left)
    else
    begin
      r := p^.up;
      while (r <> nil) and (p = r^.left) do
      begin
        p := r;
        r := r^.up;
      end;
      predBST := r;
    end;
  end;
end;

// Procedura DFS:postorder usuwająca drzewo
//-----------------------------------------
procedure DFSRelease(v : PBSTNode);
begin
  if v <> nil then
  begin
    DFSRelease(v^.left);   // usuwamy lewe poddrzewo
    DFSRelease(v^.right);  // usuwamy prawe poddrzewo
    dispose(v);            // usuwamy sam węzeł
  end;
end;

// **********************
// *** PROGRAM GŁÓWNY ***
// **********************

var
  root,p : PBSTNode;

begin
  root := readBST;     // Odczytujemy drzewo BST

  if root <> nil then
  begin

    write('SUCCESORS    :');

    p := root;

    while p <> nil do
    begin
      write(p^.key:3);
      p := succBST(p);
    end;

    writeln;

    write('PREDECCESORS :');

    p := root;

    while p <> nil do
    begin
      write(p^.key:3);
      p := predBST(p);
    end;

    writeln;

  end
  else writeln('BST is empty!!!');

  DFSRelease(root);    // usuwamy drzewo z pamięci

end.
```

Code::Blocks

```
// Następnik i poprzednik w drzewie BST
// Data: 27.04.2013
// (C)2013 mgr Jerzy Wałaszek
//------------------------------------

#include <iostream>
#include <iomanip>

using namespace std;

// Typ węzłów drzewa BST

struct BSTNode
{
  BSTNode *up, *left, *right;
  int key;
};

// Funkcja wczytuje drzewo BST ze standardowego
// wejścia i zwraca wskazanie korzenia.
//-----------------------------------------------
BSTNode * readBST()
{
  BSTNode ** vp;   // Tablica wskazań węzłów
  int key,u,l,r,i,n;

  cin >> n;                 // Odczytujemy liczbę węzłów drzewa

  vp = new BSTNode * [n];   // Tworzymy dynamiczną tablicę wskazań węzłów

  // Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  // które również tworzymy dynamicznie

  for(i = 0; i < n; i++) vp[i] = new BSTNode;

  // Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  // w pamięci wypełniając odpowiednie pola węzłów.

  for(i = 0; i < n; i++)
  {
    cin >> key >> u >> l >> r; // Czytamy klucz, numery ojca, lewego i prawego syna

    vp[i]->key = key;       // Ustawiamy klucz

    vp[i]->up  = vp[u];     // Ustawiamy ojca

    vp[i]->left  = l ? vp[l]: NULL;  // Ustawiamy lewego syna

    vp[i]->right = r ? vp[r]: NULL;  // Ustawiamy prawego syna

  }

  vp[0]->up = NULL;         // Korzeń nie posiada ojca

  BSTNode * p = vp[0];      // Zapamiętujemy korzeń

  delete [] vp;             // Usuwamy tablicę dynamiczną

  return p;
}

// Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//-------------------------------------------------------
BSTNode * minBST(BSTNode * p)
{
  if(p) while(p->left) p = p->left;

  return p;
}

// Funkcja zwraca wskazanie węzła o największym kluczu.
// Parametrem jest wskazanie korzenia drzewa BST.
//-------------------------------------------------------
BSTNode * maxBST(BSTNode * p)
{
  if(p) while(p->right) p = p->right;

  return p;
}

// Funkcja znajduje następnik węzła p
//-----------------------------------
BSTNode * succBST(BSTNode * p)
{
  BSTNode * r;

  if(p)
  {
    if(p->right) return minBST(p->right);
    else
    {
      r = p->up;
      while(r && (p == r->right))
      {
        p = r;
        r = r->up;
      }
      return r;
    }
  }
  return p;
}

// Funkcja znajduje poprzednik węzła p
//-----------------------------------
BSTNode * predBST(BSTNode * p)
{
  BSTNode * r;
```

```
  if(p)
  {
    if(p->left) return maxBST(p->left);
    else
    {
      r = p->up;
      while(r && (p == r->left))
      {
        p = r;
        r = r->up;
      }
      return r;
    }
  }
  return p;
}

// Procedura DFS:postorder usuwająca drzewo
//-----------------------------------------
void DFSRelease(BSTNode * v)
{
  if(v)
  {
    DFSRelease(v->left);   // usuwamy lewe poddrzewo
    DFSRelease(v->right);  // usuwamy prawe poddrzewo
    delete v;              // usuwamy sam węzeł
  }
}

// **********************
// *** PROGRAM GŁÓWNY ***
// **********************

int main()
{
  BSTNode * root, * p;

  root = readBST();     // Odczytujemy drzewo BST

  if(root)
  {

    cout << "SUCCESORS    :";

    for(p = root; p; p = succBST(p))cout << setw(3) << p->key;

    cout << endl << "PREDECCESORS :";

    for(p = root; p; p = predBST(p)) cout << setw(3) << p->key;

    cout << endl;

  }
  else cout << "BST is empty!!!" << endl;

  DFSRelease(root);    // usuwamy drzewo z pamięci

  return 0;
}
```

Free Basic

```
' Następnik i poprzednik w drzewie BST
' Data: 27.04.2013
' (C)2013 mgr Jerzy Wałaszek
'------------------------------------

' Typ węzłów drzewa BST

Type BSTNode
  up As BSTNode Ptr
  Left As BSTNode Ptr
  Right As BSTNode Ptr
  key As Integer
End Type

' Funkcja wczytuje drzewo BST ze standardowego
' wejścia i zwraca wskazanie korzenia.
'---------------------------------------------
Function readBST() As BSTNode Ptr

  Dim As BSTNode Ptr Ptr vp   ' Tablica wskazań węzłów
  Dim As Integer key,u,l,r,i,n

  Open Cons For Input As #1

  Input #1,n               ' Odczytujemy liczbę węzłów drzewa

  vp = New BSTNode Ptr [n] ' Tworzymy dynamiczną tablicę wskazań węzłów

  ' Tablicę dynamiczną wypełniamy wskazaniami węzłów,
  ' które również tworzymy dynamicznie

  For i = 0 To n - 1
    vp[i] = New BSTNode
  Next

  ' Teraz wczytujemy definicję drzewa i tworzymy jego strukturę
  ' w pamięci wypełniając odpowiednie pola węzłów.

  For i = 0 To n - 1

    Input #1,key,u,l,r     ' Czytamy klucz, numery ojca, lewego i prawego syna

    vp[i]->key = key       ' Ustawiamy klucz

    vp[i]->up  = vp[u]     ' Ustawiamy ojca

    If l > 0 Then
       vp[i]->Left = vp[l]  ' Ustawiamy lewego syna
    Else
```

```
        vp[i]->Left = 0
    End If

    If r > 0 Then
        vp[i]->Right = vp[r] ' Ustawiamy prawego syna
    Else
        vp[i]->Right = 0
    End If

  Next

  Close #1

  vp[0]->up = 0             ' Korzeń nie posiada ojca

  readBST = vp[0]           ' Zapamiętujemy korzeń

  Delete [] vp              ' Usuwamy tablicę dynamiczną

End Function

' Funkcja zwraca wskazanie węzła o najmniejszym kluczu.
' Parametrem jest wskazanie korzenia drzewa BST.
'-----------------------------------------------------
Function minBST(p As BSTNode Ptr) As BSTNode Ptr

  If p Then
    While p->Left
        p = p->Left
    Wend
  End If

  Return p
End Function

' Funkcja zwraca wskazanie węzła o największym kluczu.
' Parametrem jest wskazanie korzenia drzewa BST.
'-----------------------------------------------------
Function maxBST(p As BSTNode Ptr) As BSTNode Ptr

  If p Then
    While p->Right
        p = p->Right
    Wend
  End If

  Return p
End Function

' Funkcja znajduje następnik węzła p
'-----------------------------------
Function succBST(Byval p As BSTNode Ptr) As BSTNode Ptr

  Dim As BSTNode Ptr r

  If p Then
    If p->Right Then
        Return minBST(p->Right)
    Else
      r = p->up
      While (r <> 0) Andalso (p = r->Right)
        p = r
        r = r->up
      Wend
      Return r
    End If
  End If
  Return p
End Function

' Funkcja znajduje poprzednik węzła p
'-----------------------------------
Function predBST(Byval p As BSTNode Ptr) As BSTNode Ptr

  Dim As BSTNode Ptr r

  If p Then
    If p->Left Then
        Return maxBST(p->Left)
    Else
      r = p->up
      While (r <> 0) Andalso (p = r->Left)
        p = r
        r = r->up
      Wend
      Return r
    End If
  End If
  Return p
End Function

' Procedura DFS:postorder usuwająca drzewo
'-----------------------------------------
Sub DFSRelease(v As BSTNode Ptr)
  If v Then
    DFSRelease(v->Left)   ' usuwamy lewe poddrzewo
    DFSRelease(v->Right)  ' usuwamy prawe poddrzewo
    Delete v              ' usuwamy sam węzeł
  End If
End Sub

' **********************
' *** PROGRAM GŁÓWNY ***
' **********************

Dim As BSTNode Ptr root, p

root = readBST()     ' Odczytujemy drzewo BST

If root Then

  Print "SUCCESORS    :";
```

```
  p = root

  While p
    Print Using "###";p->key;
    p = succBST(p)
  Wend

  Print
  Print "PREDECCESORS :";

  p = root

  While p
    Print Using "###";p->key;
    p = predBST(p)
  Wend

  Print

Else

  Print "BST is empty!!!"

End If

DFSRelease(root)    ' usuwamy drzewo z pamięci

End
```

Wynik

```
12
9 0 1 2
5 0 3 4
15 0 5 6
4 1 7 0
7 1 8 9
10 2 0 10
18 2 0 11
1 3 0 0
6 4 0 0
8 4 0 0
12 5 0 0
19 6 0 0
SUCCESORS    :  9 10 12 15 18 19
PREDECCESORS :  9  8  7  6  5  4  1
```