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

Tworzenie drzew 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

Dołączanie nowego węzła do drzewa BST
Usuwanie węzła z drzewa BST

Problem

Mając dany ciąg kluczy, zbudować na ich podstawie drzewo BST.

Dołączanie nowego węzła do drzewa BST

Jeśli będziemy chcieli utworzyć drzewo BST, to staniemy przed koniecznością dodawania do niego nowych węzłów. Zasada pracy algorytmu dołączającego węzeł jest następująca:

Jeśli drzewo jest puste, to nowy węzeł staje się jego korzeniem.

W przeciwnym razie porównujemy klucz wstawianego węzła z kluczami kolejnych węzłów, idąc w dół drzewa. Jeśli klucz nowego węzła jest mniejszy od klucza aktualnie odwiedzonego węzła w drzewie, to przechodzimy do lewego syna. Inaczej przechodzimy do prawego syna. Całą procedurę kontynuujemy do momentu, aż dany syn nie istnieje. Wtedy dołączamy nowy węzeł na jego miejsce i kończymy.

Algorytm wstawiania węzła do drzewa BST

Wejście

k – klucz dla wstawianego węzła
root – referencja do zmiennej wskazującej korzeń drzewa BST

Wyjście:

Drzewo BST z wstawionym nowym węzłem o kluczu *k*.

Zmienne pomocnicze:

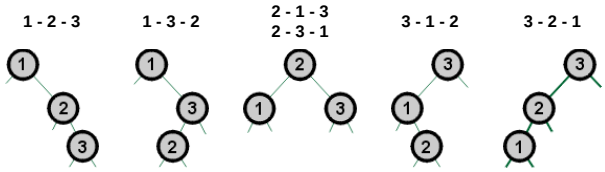
p,w – wskazanie węzłów

Lista kroków:

K01: Utwórz nowy węzeł ; *tworzymy nowy węzeł*
K02: *w* ← adres nowego węzła
K03: (*w*→*left*) ← **nil** ; *inicjujemy pola węzła*
K04: (*w*→*right*) ← **nil**
K05: (*w*→*key*) ← *k*
K06: *p* ← *root*
K07: **Jeśli** *p* ≠ **nil**, **to idź** do K11 ; *sprawdzamy, czy drzewo jest puste*
K08: *root* ← *w* ; *jeśli tak, to nowy węzeł staje się jego korzeniem*
K09: (*w*→*up*) ← *p* ; *uzupełniamy ojca węzła*
K10: **Zakończ**
K11: **Jeśli** *k* < (*p*→*key*), **to idź do** K15 ; *porównujemy klucze*
K12: **Jeśli** (*p*→*right*) = **nil**, **to**: ; *jeśli prawy syn nie istnieje, to*
(*p*→*right*) ← *w* ; *nowy węzeł staje się prawym synem*
Idź do K09 ; *i wychodzimy z pętli*
K13: *p* ← (*p*→*right*) ; *inaczej przechodzimy do prawego syna*

K14: **Idź do** K11 *; i kontynuujemy pętlę*
K15 **Jeśli** (*p*→*left*) = **nil**, **to**: *; to samo dla lewego syna*
(*p*→*left*) ← *w*
Idź do K09
K16: *p* ← (*p*→*left*)
K17: **Idź do** K11

W zależności od kolejności wprowadzania danych do drzewa BST mogą powstać różne konfiguracje węzłów. Dla 3 kluczy możemy otrzymać aż pięć różnych drzew BST:



Szczególnie niekorzystne jest wprowadzanie wartości uporządkowanych rosnąco lub malejąco - w takim przypadku otrzymujemy drzewo BST zdegenerowane do listy liniowej (pierwszy i ostatni przykład). W takim drzewie operacje wyszukiwania wymagają czasu rzędu O(*n*), a nie O(log *n*).

Program

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

Program generuje 20 losowych kluczy z zakresu od 1 do 10, tworzy z nich węzły i wstawia do drzewa BST. Na koniec drzewo zostaje zaprezentowane algorytmem prezentacji drzew, który omówiliśmy we wcześniejszym artykule. Procedura wstawiania węzła do drzewa BST jest nieco zmodyfikowana – przyjmuje jako parametry referencję do zmiennej przechowującej adres korzenia oraz wartość klucza k. Sam węzeł jest tworzony dynamicznie wewnątrz procedury.

Lazarus

```
// Budowanie drzewa BST
// Data: 1.05.2013
// (C)2013 mgr Jerzy Wałaszek
//-----------------------------

program bst;

// Typ węzłów drzewa BST

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

// Zmienne globalne

var
  cr,cl,cp : string;      // łańcuchy do znaków ramek

// Procedura wypisuje drzewo
//--------------------------
procedure printBT(sp,sn : string; v : PBSTNode);
var
  s : string;
begin
  if v <> nil then
  begin
    s := sp;
    if sn = cr then s[length(s) - 1] := ' ';
    printBT(s+cp,cr,v^.right);

    s := Copy(sp,1,length(sp)-2);
    writeln(s,sn,v^.key);

    s := sp;
    if sn = cl then s[length(s) - 1] := ' ';
    printBT(s+cp,cl,v^.left);
  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;

// Procedura wstawia do drzewa BST węzeł o kluczu k
//-------------------------------------------------
procedure insertBST(var root : PBSTNode; k : integer);
var
  w,p : PBSTNode;
begin

  new(w);            // Tworzymy dynamicznie nowy węzeł

  w^.left  := nil;   // Zerujemy wskazania synów
  w^.right := nil;
  w^.key := k;       // Wstawiamy klucz
```

```
  p := root;         // Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  if p = nil then    // Drzewo puste?
    root := w        // Jeśli tak, to w staje się korzeniem
  else
    while true do    // Pętla nieskończona
      if k < p^.key then // W zależności od klucza idziemy do lewego lub
      begin              // prawego syna, o ile takowy istnieje
        if p^.left = nil then // Jeśli lewego syna nie ma,
        begin
          p^.left := w;  // to w staje się lewym synem
          break;         // Przerywamy pętlę while
        end
        else p := p^.left;
      end
      else
      begin
        if p^.right = nil then // Jeśli prawego syna nie ma,
        begin
          p^.right := w;  // to w staje się prawym synem
          break;          // Przerywamy pętlę while
        end
        else p := p^.right;
      end;

  w^.up  := p;     // Ojcem w jest zawsze p

end;

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

var
  root : PBSTNode;
  i,k  : integer;

begin
  // ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
  // wstawiać znaki konsoli do tworzenia ramek.
  // cr = +--
  //      |

  // cl = |
  //      +--

  // cp = |
  //      |

  cr := #218#196;
  cl := #192#196;
  cp := #179#32;

  randomize;            // inicjujemy generator pseudolosowy

  root := nil;          // Tworzymy puste drzewo BST

  for i := 1 to 20 do   // Wypełniamy drzewo BST węzłami
  begin
    k := 1 + random(9); // Generujemy klucz 1...9
    write(k,' ');       // Wyświetlamy klucz
    insertBST(root,k);  // Tworzymy nowy węzeł o kluczu k i umieszczamy go w drzewie
  end;

  writeln; writeln;

  printBT('','',root); // Wyświetlamy drzewo
  DFSRelease(root);    // Usuwamy drzewo z pamięci
end.
```

Code::Blocks

```
// Budowanie drzewa BST
// Data: 1.05.2013
// (C)2013 mgr Jerzy Wałaszek
//-----------------------------

#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>

using namespace std;

// Typ węzłów drzewa BST

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

// Zmienne globalne

string cr,cl,cp;      // łańcuchy do znaków ramek

// Procedura wypisuje drzewo
//--------------------------
void printBT(string sp, string sn, BSTNode * v)
{
  string s;

  if(v)
  {
    s = sp;
    if(sn == cr) s[s.length() - 2] = ' ';
    printBT(s + cp, cr, v->right);

    s = s.substr(0,sp.length()-2);
    cout << s << sn << v->key << endl;
```

```
    s = sp;
    if(sn == cl) s[s.length() - 2] = ' ';
    printBT(s + cp, cl, v->left);
  }
}

// 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
  }
}

// Procedura wstawia do drzewa BST węzeł o kluczu k
//-------------------------------------------------
void insertBST(BSTNode * & root, int k)
{
  BSTNode * w, * p;

  w = new BSTNode;           // Tworzymy dynamicznie nowy węzeł

  w->left = w->right = NULL; // Zerujemy wskazania synów
  w->key = k;                // Wstawiamy klucz.

  p = root;                  // Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  if(!p)                     // Drzewo puste?
    root = w;                // Jeśli tak, to w staje się korzeniem
  else
    while(true)              // Pętla nieskończona
      if(k < p->key)         // W zależności od klucza idziemy do lewego lub
      {                      // prawego syna, o ile takowy istnieje
        if(!p->left)         // Jeśli lewego syna nie ma,
        {
          p->left = w;       // to węzeł w staje się lewym synem
          break;             // Przerywamy pętlę while
        }
        else p = p->left;
      }
      else
      {
        if(!p->right)        // Jeśli prawego syna nie ma,
        {
          p->right = w;      // to węzeł w staje się prawym synem
          break;             // Przerywamy pętlę while
        }
        else p = p->right;
      }

  w->up  = p;                // Ojcem węzła w jest zawsze węzeł wskazywany przez p

}

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

int main()
{
  BSTNode * root = NULL;
  int i, k;

  // ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
  // wstawiać znaki konsoli do tworzenia ramek.
  // cr = +--
  //      |

  // cl = |
  //      +--

  // cp = |
  //      |

  cr = cl = cp = "  ";
  cr[0] = 218; cr[1] = 196;
  cl[0] = 192; cl[1] = 196;
  cp[0] = 179;

  srand(time(NULL));         // inicjujemy generator pseudolosowy

  for(i = 0; i < 20; i++)    // Wypełniamy drzewo BST węzłami
  {
    k = 1 + rand() % 9;      // Generujemy klucz 1...9
    cout << k << " ";        // Wyświetlamy klucz
    insertBST(root,k);       // Tworzymy nowy węzeł o kluczu k i umieszczamy go w drzewie
  }

  cout << endl << endl;

  printBT("","",root);       // Wyświetlamy drzewo
  DFSRelease(root);          // Usuwamy drzewo z pamięci

  return 0;
}
```

Free Basic

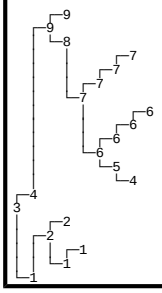
```
' Budowanie drzewa BST
' Data: 1.05.2013
' (C)2013 mgr Jerzy Wałaszek
'----------------------------

' Typ węzłów drzewa BST

Type BSTNode
```

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

' Zmienne globalne

Dim Shared As String * 2 cr,cl,cp ' łańcuchy do ramek

' Procedura wypisuje drzewo
'--------------------------
Sub printBT(sp As String, sn As String, v As BSTNode Ptr)

  Dim As String s

  If v Then
    s = sp
    If sn = cr Then Mid(s,Len(s) - 1, 1) = " "
    printBT(s + cp, cr, v->Right)

    s = Mid(s,1, Len(sp)-2)
    Print Using "&&&";s;sn;v->key

    s = sp
    If sn = cl Then Mid(s,Len(s) - 1, 1) = " "
    printBT(s + cp, cl, v->Left)
  End If
End Sub

' 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

' Procedura wstawia do drzewa BST węzeł o kluczu k
'-------------------------------------------------
Sub insertBST(Byref root As BSTNode Ptr, k As Integer)

  Dim As BSTNode Ptr w, p

  w = New BSTNode           ' Tworzymy dynamicznie nowy węzeł

  w->Left  = 0              ' Zerujemy wskazania synów
  w->Right = 0
  w->key = k                ' Wstawiamy klucz

  p = root                  ' Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  If p = 0 Then             ' Drzewo puste?
    root = w                ' Jeśli tak, to w staje się korzeniem
  Else
    Do                      ' Pętla nieskończona
      If k < p->key Then    ' W zależności od klucza idziemy do lewego lub
                            ' prawego syna, o ile takowy istnieje
        If p->Left = 0 Then ' Jeśli lewego syna nie ma,
          p->Left = w       ' to węzeł w staje się lewym synem
          Exit Do           ' Przerywamy pętlę
        Else
          p = p->Left
        End If
      Else
        If p->Right = 0 Then' Jeśli prawego syna nie ma,
          p->Right = w      ' to węzeł w staje się prawym synem
          Exit Do           ' Przerywamy pętlę
        Else
          p = p->Right
        End If
      End If
    Loop                    ' Koniec pętli
  End If

  w->up  = p                ' Ojcem węzła w jest zawsze węzeł wskazywany przez p

End Sub

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

Dim As BSTNode Ptr root = 0
Dim As Integer i, k

' ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
' wstawiać znaki konsoli do tworzenia ramek.
' cr = +--
'      |

' cl = |
'      +--

' cp = |
'      |

cr = Chr(218) + Chr(196)
cl = Chr(192) + Chr(196)
cp = Chr(179) + " "

Randomize                 ' inicjujemy generator pseudolosowy

For i = 1 To 20           ' Wypełniamy drzewo BST węzłami
  k = 1 + Int(Rnd() * 9) ' Generujemy klucz 1...9
  Print k;                ' Wyświetlamy klucz
  insertBST(root,k)       ' Tworzymy nowy węzeł o kluczu k i umieszczamy go w drzewie
Next

Print
Print
```

```
printBT("","",root)       ' Wyświetlamy drzewo
DFSRelease(root)          ' Usuwamy drzewo z pamięci

End
```

Wynik

```
3 4 9 8 9 7 6 1 6 7 6 5 2 7 6 1 1 7 2 4
```

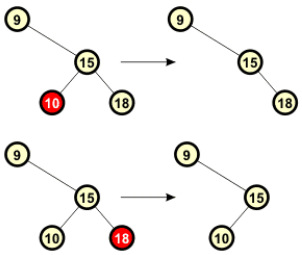


Problem

Usunąć z drzewa BST węzeł o zadanym kluczu.

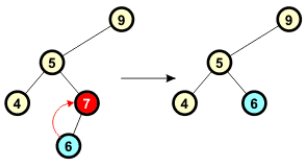
Usuwanie węzła z drzewa BST

Węzły usuwamy z drzewa BST tak, aby została zachowana hierarchia powiązań węzłów. Musimy rozpatrzyć kilka przypadków.



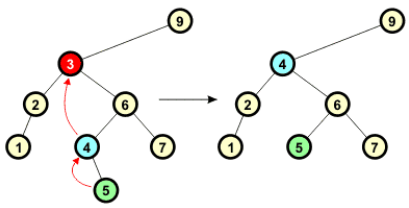
Przypadek 1:

Usuwany węzeł jest liściem, tzn. nie posiada synów. W takim przypadku po prostu odłączamy go od drzewa i usuwamy.



Przypadek 2:

Usuwany węzeł posiada tylko jednego syna. Węzeł zastępujemy jego synem, po czym węzeł usuwamy z pamięci.



Przypadek 3:

Najbardziej skomplikowany jest przypadek trzeci, gdy usuwany węzeł posiada dwóch synów. W takim przypadku znajdujemy węzeł będący następnikiem usuwanego węzła. Przenosimy dane i klucz z następnika do usuwanego węzła, po czym następnik usuwamy z drzewa – do tej operacji można rekurencyjnie wykorzystać tę samą procedurę lub zastąpić następnik przez jego prawego syna (następnik nigdy nie posiada lewego syna).

Jako wariant można również zastępować usuwany węzeł jego poprzednikiem.

Algorytm usuwania węzła z drzewa BST

Wejście

root – referencja do zmiennej wskazującej węzeł drzewa BST, który jest korzeniem poddrzewa z węzłem o kluczu *k*
X – wskazanie usuwanego węzła

Wyjście:

Drzewo BST z usuniętym węzłem *X*

Zmienne pomocnicze:

Y,Z – wskazania węzła
succBST(p) – zwraca wskazanie następnika węzła *p*

Lista kroków:

K01 **Jeśli** ((*X*→*left*) = **nil**) ∨ ((*X*→*right*) = **nil**), **to** *Y* ← *X* ; *Y wskazuje węzeł do usunięcia. Jeśli X nie ma synów lub ma tylko jednego, to Y jest X*
Inaczej Y ← succBST(*X*) ; *W przeciwnym razie Y jest bezpośrednim następnikiem X*
K02: **Jeśli** ((*Y*→*left*) ≠ **nil**), **to** *Z* ← (*Y*→*left*) ; *Z jest synem Y*
Inaczej *Z* ← (*Y*→*right*)
K03: **Jeśli** *Z* ≠ **nil**, **to** (*Z*→*up*) ← (*Y*→*up*) ; *Jeśli syn Z istnieje, to jego ojcem staje się ojciec Y*
K04: **Jeśli** (*Y*→*up*) ≠ **nil**, **to idź do** K07 ; *Sprawdzamy, czy usuwany węzeł jest korzeniem drzewa*
K05: *root* ← *Z* ; *Jeśli tak, to korzeniem stanie się syn Y*
K06: **Idź do** K08

K07: **Jeśli** $Y = (Y \rightarrow up \rightarrow left)$, **to** $(Y \rightarrow up \rightarrow left) \leftarrow Z$ *; Jeśli syn Y nie jest korzeniem, to zastępuje Y w drzewie*
Inaczej $(Y \rightarrow up \rightarrow right) \leftarrow Z$
K08: **Jeśli** $Y \neq X$, **to** przenieś dane z Y do X *; Jeśli Y nie jest pierwotnym węzłem, to jest jego następnikiem i zamieniamy dane*
K09: Usuń węzeł Y *; Teraz możemy usunąć węzeł Y z pamięci*
K10: **Zakończ**

Program

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

Program tworzy drzewo BST z 15 węzłów o kluczach od 1 do 15. Wyświetla je. Następnie usuwa z niego 5 losowych węzłów i wyświetla ponownie drzewo BST.

Lazarus

```
// Usuwanie węzłów w drzewie BST
// Data: 1.05.2013
// (C)2013 mgr Jerzy Wałaszek
//-------------------------------

program bst;

// Typ węzłów drzewa BST

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

// Zmienne globalne

var
  cr,cl,cp : string;      // łańcuchy do znaków ramek

// Procedura wypisuje drzewo
//--------------------------
procedure printBT(sp,sn : string; v : PBSTNode);
var
  s : string;
begin
  if v <> nil then
  begin
    s := sp;
    if sn = cr then s[length(s) - 1] := ' ';
    printBT(s+cp,cr,v^.right);

    s := Copy(sp,1,length(sp)-2);
    writeln(s,sn,v^.key);

    s := sp;
    if sn = cl then s[length(s) - 1] := ' ';
    printBT(s+cp,cl,v^.left);
  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;

// Procedura wstawia do drzewa BST węzeł o kluczu k
//-------------------------------------------------
procedure insertBST(var root : PBSTNode; k : integer);
var
  w,p : PBSTNode;
begin

  new(w);            // Tworzymy dynamicznie nowy węzeł

  w^.left  := nil;   // Zerujemy wskazania synów
  w^.right := nil;
  w^.key := k;       // Wstawiamy klucz

  p := root;         // Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  if p = nil then    // Drzewo puste?
    root := w        // Jeśli tak, to w staje się korzeniem
  else
    while true do    // Pętla nieskończona
      if k < p^.key then // W zależności od klucza idziemy do lewego lub
      begin              // prawego syna, o ile takowy istnieje
        if p^.left = nil then // Jeśli lewego syna nie ma,
        begin
          p^.left := w;  // to w staje się lewym synem
          break;         // Przerywamy pętlę while
        end
        else p := p^.left;
      end
      else
      begin
        if p^.right = nil then // Jeśli prawego syna nie ma,
        begin
          p^.right := w; // to w staje się prawym synem
          break;         // Przerywamy pętlę while
        end
        else p := p^.right;
```

```
      end;

  w^.up  := p;           // Ojcem w jest zawsze p

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 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;

// Procedura usuwa węzeł z drzewa BST
// root - referencja do zmiennej wskazującej węzeł
// X    - wskazanie węzła do usunięcia
//-------------------------------------------
procedure removeBST(var root : PBSTNode; X : PBSTNode);
var
  Y,Z : PBSTNode;
begin
  if X <> nil then
  begin

    // Jeśli X nie ma synów lub ma tylko jednego, to Y wskazuje X
    // Inaczej Y wskazuje następnik X

    if (X^.left = nil) or (X^.right = nil) then Y := X
    else                                        Y := succBST(X);

    // Z wskazuje syna Y lub nil

    if Y^.left <> nil then Z := Y^.left
    else                   Z := Y^.right;

    // Jeśli syn Y istnieje, to zastąpi Y w drzewie

    if Z <> nil then Z^.up := Y^.up;

    // Y zostaje zastąpione przez Z w drzewie

    if Y^.up = nil then root := Z
    else if Y = Y^.up^.left then Y^.up^.left  := Z
    else                         Y^.up^.right := Z;

    // Jeśli Y jest następnikiem X, to kopiujemy dane

    if Y <> X then X^.key := Y^.key;

    Dispose(Y);

  end;
end;

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

var
  Tk   : array[0..14] of integer;  // tablica kluczy węzłów
  i,x,i1,i2  : integer;
  root : PBSTNode;                 // korzeń drzewa BST

begin
  // ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
  // wstawiać znaki konsoli do tworzenia ramek.
  // cr = +--
  //      |
```

```
  // cl = |
  //      +--

  // cp = |
  //      |

  cr := #218#196;
  cl := #192#196;
  cp := #179#32;

  randomize;            // Inicjujemy generator pseudolosowy

  root := nil;          // Tworzymy puste drzewo BST

  for i := 0 to 14 do   // Tablicę wypełniamy wartościami kluczy
    Tk[i] := i + 1;

  for i := 1 to 100 do  // Mieszamy tablicę
  begin
    i1 := random(15);   // Losujemy 2 indeksy
    i2 := random(15);

    x := Tk[i1];        // Wymieniamy Tk[i1] <--> Tk[i2]
    Tk[i1] := Tk[i2];
    Tk[i2] := x;
  end;

  for i := 0 to 14 do   // Na podstawie tablicy tworzymy drzewo BST
  begin
    write(Tk[i]:3);
    insertBST(root,Tk[i]);
  end;

  writeln; writeln;

  printBT('','',root);  // Wyświetlamy zawartość drzewa BST

  writeln; writeln;

  for i := 1 to 100 do  // Ponownie mieszamy tablicę
  begin
    i1 := random(15); i2 := random(15);
    x := Tk[i1]; Tk[i1] := Tk[i2]; Tk[i2] := x;
  end;

  for i := 0 to 4 do    // Usuwamy 5 węzłów
  begin
    write(Tk[i]:3);
    removeBST(root,findBST(root,Tk[i]));
  end;

  writeln; writeln;

  printBT('','',root);  // Wyświetlamy zawartość drzewa BST

  DFSRelease(root);     // Usuwamy drzewo BST z pamięci

end.
```

Code::Blocks

```
// Usuwanie węzłów w drzewie BST
// Data: 1.05.2013
// (C)2013 mgr Jerzy Wałaszek
//-------------------------------

#include <iostream>
#include <iomanip>
#include <string>
#include <cstdlib>
#include <ctime>

using namespace std;

// Typ węzłów drzewa BST

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

// Zmienne globalne

string cr,cl,cp;      // łańcuchy do znaków ramek

// Procedura wypisuje drzewo
//--------------------------
void printBT(string sp, string sn, BSTNode * v)
{
  string s;

  if(v)
  {
    s = sp;
    if(sn == cr) s[s.length() - 2] = ' ';
    printBT(s + cp, cr, v->right);

    s = s.substr(0,sp.length()-2);
    cout << s << sn << v->key << endl;

    s = sp;
    if(sn == cl) s[s.length() - 2] = ' ';
    printBT(s + cp, cl, v->left);
  }
}

// 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
  }
}

// Procedura wstawia do drzewa BST węzeł o kluczu k
//-------------------------------------------------
void insertBST(BSTNode * & root, int k)
{
  BSTNode * w, * p;

  w = new BSTNode;           // Tworzymy dynamicznie nowy węzeł

  w->left = w->right = NULL; // Zerujemy wskazania synów
  w->key = k;                // Wstawiamy klucz

  p = root;                  // Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  if(!p)                     // Drzewo puste?
    root = w;                // Jeśli tak, to w staje się korzeniem
  else
    while(true)              // Pętla nieskończona
      if(k < p->key)         // W zależności od klucza idziemy do lewego lub
      {                      // prawego syna, o ile takowy istnieje
        if(!p->left)         // Jeśli lewego syna nie ma,
        {
          p->left = w;       // to węzeł w staje się lewym synem
          break;             // Przerywamy pętlę while
        }
        else p = p->left;
      }
      else
      {
        if(!p->right)        // Jeśli prawego syna nie ma,
        {
          p->right = w;      // to węzeł w staje się prawym synem
          break;             // Przerywamy pętlę while
        }
        else p = p->right;
      }

  w->up  = p;                // Ojcem węzła w jest zawsze węzeł wskazywany przez 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 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;
}

// Procedura usuwa węzeł z drzewa BST
// root - referencja do zmiennej wskazującej węzeł
// X    - wskazanie węzła do usunięcia
//-------------------------------------------------
void removeBST(BSTNode * & root, BSTNode * X)
{
  BSTNode * Y, * Z;

  if(X)
  {
    // Jeśli X nie ma synów lub ma tylko jednego, to Y wskazuje X
    // Inaczej Y wskazuje następnik X

    Y = !X->left || !X->right ? X : succBST(X);

    // Z wskazuje syna Y lub NULL
```

```
    Z = Y->left ? Y->left : Y->right;

    // Jeśli syn Y istnieje, to zastąpi Y w drzewie

    if(Z) Z->up = Y->up;

    // Y zostaje zastąpione przez Z w drzewie

    if(!Y->up) root = Z;
    else if(Y == Y->up->left) Y->up->left  = Z;
    else                      Y->up->right = Z;

    // Jeśli Y jest następnikiem X, to kopiujemy dane

    if(Y != X) X->key = Y->key;

    delete Y;

  }
}

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

int main()
{
  int Tk[15];   // tablica kluczy węzłów
  int i,x,i1,i2;
  BSTNode * root = NULL;

  // ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
  // wstawiać znaki konsoli do tworzenia ramek.
  // cr = +--
  //      |

  // cl = |
  //      +--

  // cp = |
  //      |

  cr = cl = cp = "  ";
  cr[0] = 218; cr[1] = 196;
  cl[0] = 192; cl[1] = 196;
  cp[0] = 179;

  srand(time(NULL));        // inicjujemy generator pseudolosowy

  for(i = 0; i < 15; i++)   // Tablicę wypełniamy wartościami kluczy
    Tk[i] = i + 1;

  for(i = 0; i < 100; i++)  // Mieszamy tablicę
  {
    i1 = rand() % 15;       // Losujemy 2 indeksy
    i2 = rand() % 15;

    x = Tk[i1];             // Wymieniamy Tk[i1] <--> Tk[i2]
    Tk[i1] = Tk[i2];
    Tk[i2] = x;
  }

  for(i = 0; i < 15; i++)   // Na podstawie tablicy tworzymy drzewo BST
  {
    cout << setw(3) << Tk[i];
    insertBST(root,Tk[i]);
  }

  cout << endl << endl;

  printBT("","",root);      // Wyświetlamy zawartość drzewa BST

  cout << endl << endl;

  for(i = 0; i < 100; i++)  // Ponownie mieszamy tablicę
  {
    i1 = rand() % 15; i2 = rand() % 15;
    x = Tk[i1]; Tk[i1] = Tk[i2]; Tk[i2] = x;
  }

  for(i = 0; i < 5; i++)    // Usuwamy 5 węzłów
  {
    cout << setw(3) << Tk[i];
    removeBST(root,findBST(root,Tk[i]));
  }

  cout << endl << endl;

  printBT("","",root);      // Wyświetlamy zawartość drzewa BST

  DFSRelease(root);         // Usuwamy drzewo BST z pamięci

  return 0;
}
```

Free Basic

```
' Usuwanie węzłów w drzewie BST
' Data: 1.05.2013
' (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
```

```
' Zmienne globalne

Dim Shared As String * 2 cr,cl,cp ' łańcuchy do ramek

' Procedura wypisuje drzewo
'--------------------------
Sub printBT(sp As String, sn As String, v As BSTNode Ptr)

  Dim As String s

  If v Then
    s = sp
    If sn = cr Then Mid(s,Len(s) - 1, 1) = " "
    printBT(s + cp, cr, v->Right)

    s = Mid(s,1, Len(sp)-2)
    Print Using "&&&";s;sn;v->key

    s = sp
    If sn = cl Then Mid(s,Len(s) - 1, 1) = " "
    printBT(s + cp, cl, v->Left)
  End If
End Sub

' 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

' Procedura wstawia do drzewa BST węzeł o kluczu k
'-------------------------------------------------
Sub insertBST(Byref root As BSTNode Ptr, k As Integer)

  Dim As BSTNode Ptr w, p

  w = New BSTNode           ' Tworzymy dynamicznie nowy węzeł

  w->Left  = 0              ' Zerujemy wskazania synów
  w->Right = 0
  w->key = k                ' Wstawiamy klucz

  p = root                  ' Wyszukujemy miejsce dla w, rozpoczynając od korzenia

  If p = 0 Then             ' Drzewo puste?
    root = w                ' Jeśli tak, to w staje się korzeniem
  Else
    Do                      ' Pętla nieskończona
      If k < p->key Then    ' W zależności od klucza idziemy do lewego lub
                            ' prawego syna, o ile takowy istnieje
        If p->Left = 0 Then ' Jeśli lewego syna nie ma,
          p->Left = w       ' to węzeł w staje się lewym synem
          Exit Do           ' Przerywamy pętlę
        Else
          p = p->Left
        End If
      Else
        If p->Right = 0 Then' Jeśli prawego syna nie ma,
          p->Right = w      ' to węzeł w staje się prawym synem
          Exit Do           ' Przerywamy pętlę
        Else
          p = p->Right
        End If
      End If
    Loop                    ' Koniec pętli
  End If

  w->up  = p                ' Ojcem węzła w jest zawsze węzeł wskazywany przez p

End Sub

' 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 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

' Procedura usuwa węzeł z drzewa BST
' root - referencja do zmiennej wskazującej węzeł
' X    - wskazanie węzła do usunięcia
'-----------------------------------------------
Sub removeBST(Byref root As BSTNode Ptr, Byval X As BSTNode Ptr)

  Dim As BSTNode Ptr Y, Z

  If X Then

    ' Jeśli X nie ma synów lub ma tylko jednego, to Y wskazuje X
    ' Inaczej Y wskazuje następnik X

    If (X->Left = 0) Orelse (X->Right = 0) Then Y = X: Else Y = succBST(X)

    ' Z wskazuje syna Y lub NULL

    If Y->Left Then Z = Y->Left: Else Z = Y->Right

    ' Jeśli syn Y istnieje, to zastąpi Y w drzewie

    If Z Then Z->up = Y->up

    ' Y zostaje zastąpione przez Z w drzewie

    If Y->up = 0 Then
        root = Z
    Elseif Y = Y->up->Left Then
        Y->up->Left  = Z
    Else
        Y->up->Right = Z
    End If

    ' Jeśli Y jest następnikiem X, to kopiujemy dane

    If Y <> X Then X->key = Y->key

    Delete Y

  End If

End Sub


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

Dim As Integer Tk(14)    ' tablica kluczy węzłów
Dim As Integer i,i1,i2
Dim As BSTNode Ptr root = 0

' ustawiamy łańcuchy znakowe, ponieważ nie wszystkie edytory pozwalają
' wstawiać znaki konsoli do tworzenia ramek.
' cr = +--
'      |

' cl = |
'      +--

' cp = |
'      |

cr = Chr(218) + Chr(196)
cl = Chr(192) + Chr(196)
cp = Chr(179) + " "

Randomize                ' Inicjujemy generator pseudolosowy

For i = 0 To 14          ' Tablicę wypełniamy wartościami kluczy
  Tk(i) = i + 1
Next

For i = 1 To 100         ' Mieszamy tablicę
  i1 = Int(Rnd() * 15)   ' Losujemy 2 indeksy
  i2 = Int(Rnd() * 15)
  Swap Tk(i1),Tk(i2)     ' Wymieniamy Tk[i1] <--> Tk[i2]
Next

For i = 0 To 14          ' Na podstawie tablicy tworzymy drzewo BST
  Print Using "###";Tk(i);
  insertBST(root,Tk(i))
Next

Print: Print

printBT("","",root)      ' Wyświetlamy zawartość drzewa BST

Print: Print

For i = 1 To 100         ' Ponownie ieszamy tablicę
  i1 = Int(Rnd() * 15): i2 = Int(Rnd() * 15)
  Swap Tk(i1),Tk(i2)
Next
```

```
For i = 0 To 4           ' Usuwamy 5 węzłów
  Print Using "###";Tk(i);
  removeBST(root,findBST(root,Tk(i)))
Next

Print: Print

printBT("","",root)      ' Wyświetlamy zawartość drzewa BST

DFSRelease(root)         ' Usuwamy drzewo BST z pamięci

End
```

Wynik

```
  5 10  9 12 11 15  3 14  2  1  6 13  4  8  7

    ┌─15
    │ └─14
    │   └─13
  ┌─12
  │ └─11
┌─10
│ └─9
│   │ ┌─8
│   │ └─7
│   └─6
5
│ ┌─4
└─3
  └─2
    └─1


  4  1  6  3 15

    ┌─14
    │ └─13
  ┌─12
  │ └─11
┌─10
│ └─9
│   └─8
│     └─7
5
└─2
```