



ALGORYTMY STRUKTURY DANYCH

©2014 mgr Jerzy Wałaszek
I LO w Tarnowie



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



Przechodzenie drzew binarnych – DFS

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 wyrażań
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

Algorytm DFS
pre-order
in-order
post-order

Podstawowe operacje na tablicach
Podstawowe pojęcia dotyczące stosów

Problem

Dokonać przejścia w głąb przez wszystkie węzły drzewa binarnego.

Drzewa odzwierciedlają różnego rodzaju struktury. W węzłach są przechowywane różne informacje. Często będziemy spotykać się z zadaniem, które wymaga odwiedzenia każdego węzła drzewa i przetworzenia informacji przechowywanych w węźle. Zadanie to realizują **algorytmy przechodzenia drzewa** (ang. *tree traversal algorithm*). Polegają one na poruszaniu się po drzewie od węzła do węzła wzdłuż krawędzi w pewnym porządku i przetwarzaniu danych przechowywanych w węzłach. Istnieje kilka takich algorytmów o różnych własnościach. Omówimy je w tym rozdziale.

Algorytm DFS

Odwiedzenie węzła (ang. *node visit*) polega na dojściu do danego węzła po strukturze drzewa oraz przetworzenie danych zawartych w węźle. **Przeszukiwanie w głąb** (ang. *Depth First Search - DFS*) jest bardzo popularnym algorytmem przechodzenia przez wszystkie węzły poddrzewa, którego korzeniem jest węzeł startowy (jeśli węzeł startowym będzie korzeń drzewa, to DFS przejdzie przez wszystkie węzły zawarte w drzewie). Zasada działania tego algorytmu opiera się na rekurencji lub stosie, którym zastępujemy wywołania rekurencyjne. Ze względu na kolejność odwiedzin węzłów wyróżniamy trzy odmiany DFS (istnieją jeszcze trzy dalsze będące "lustrzanymi odbiciami"):

DFS: pre-order – przejście wzdłużne

W **przejściu wzdłużnym** (ang. *pre-order traversal*) najpierw odwiedzamy korzeń poddrzewa, a następnie przechodzimy kolejno lewe i prawe poddrzewo. Prześledź poniższy przykład:

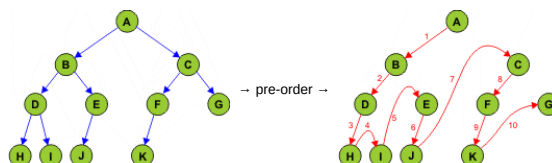
Dedykowane rozwiązania IT		
Poznaj nasz Software House i stwórz z nami aplikację albo oprogramowanie!		
Stan przejścia	Przetworzone węzły	Opis
	A	Odwiedzamy korzeń A i przechodzimy do lewego syna B, który jest korzeniem lewego poddrzewa dla węzła A.
	A B	Odwiedzamy korzeń B poddrzewa i przechodzimy do lewego syna D, który jest korzeniem poddrzewa dla węzła B.

	A B D	Odwiedzamy D i przechodzimy do H.
	A B D H	Odwiedzamy H. Węzeł nie ma dzieci, zatem lewe poddrzewo dla węzła D (ojca H) zostało przebyte. Wracamy do węzła D i przechodzimy do I, który jest korzeniem prawego poddrzewa dla węzła D.
	A B D H I	Odwiedzamy I. Węzeł jest liściem, zatem prawe drzewo węzła D zostało przebyte. Jednocześnie zostało przebyte całe lewe poddrzewo węzła B. Wracamy do B i przechodzimy do E, które jest korzeniem prawego poddrzewa węzła B.
	A B D H I E	Odwiedzamy węzeł E i przechodzimy do lewego poddrzewa, czyli do węzła J.
	A B D H I E J	Wracamy do korzenia A, ponieważ całe lewe poddrzewo zostało przebyte. Teraz w analogiczny sposób odwiedzamy prawe poddrzewo, rozpoczynając od jego korzenia C.
	A B D H I E J C	Odwiedzamy C i przechodzimy do lewego poddrzewa.
	A B D H I E J C F	Odwiedzamy F i przechodzimy do lewego poddrzewa.
	A B D H I E J C F K	Odwiedzamy K i wracamy do C (lewe poddrzewo węzła C jest przebyte), a następnie przechodzimy do prawego poddrzewa.
	A B D H I E J C F K G	Odwiedzamy G. Drzewo zostało przebyte.

Dedykowane rozwiązania IT

Poznaj nasz Software House i stwórz z nami aplikację albo oprogramowanie!

Na poniższym rysunku zaznaczyliśmy linię przejścia przez węzły drzewa.



Algorytm rekurencyjny DFS: preorder dla drzewa binarnego

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Lista kroków:

K01: Jeśli v = nil, to zakończ ; koniec rekurencji
 K02: Odwiedź węzeł wskazany przez v
 K03: preorder(v → left) ; przejdź rekurencyjnie przez lewe poddrzewo
 K04: preorder(v → right) ; przejdź rekurencyjnie przez prawe poddrzewo
 K05: Zakończ

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego jak w przykładzie powyżej. Danymi węzłów są znaki A. B. C. Po utworzeniu drzewa program przechodzi je za pomocą algorytmu preorder, wypisując odwiedzane kolejno węzły.

Lazarus
<pre>// Przechodzenie drzew binarnych DFS:preorder // Data: 17.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- program DFS_preorder; // Typ węzłów drzewa type PBTNode = ^BTNode; BTNode = record left : PBTNode; right : PBTNode; data : char; end; // Tworzenie struktury drzewa rozpoczynamy od liści var G : BTNode = (left:nil; right:nil; data:'G'); H : BTNode = (left:nil; right:nil; data:'H'); I : BTNode = (left:nil; right:nil; data:'I'); J : BTNode = (left:nil; right:nil; data:'J'); K : BTNode = (left:nil; right:nil; data:'K'); // Tworzymy kolejnych ojców D : BTNode = (left:@H; right:@I; data:'D'); E : BTNode = (left:@J; right:nil; data:'E'); F : BTNode = (left:@K; right:nil; data:'F'); B : BTNode = (left:@D; right:@E; data:'B'); C : BTNode = (left:@F; right:@G; data:'C'); // Tworzymy korzeń drzewa A : BTNode = (left:@B; right:@C; data:'A'); // Rekurencyjna procedura preorder procedure preorder(v : PBTNode); begin if v <> nil then begin write(v^.data, ' '); // odwiedzamy węzeł preorder(v^.left); // przechodzimy lewe poddrzewo preorder(v^.right); // przechodzimy prawe poddrzewo end; end; begin preorder(@A); // przejście rozpoczynamy od korzenia writeln; end.</pre>
Code::Blocks
<pre>// Przechodzenie drzew binarnych DFS:preorder // Data: 17.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- #include <iostream> using namespace std; // Typ węzłów drzewa struct BTNode { BTNode * left; BTNode * right; char data; }; // Tworzenie struktury drzewa rozpoczynamy od liści</pre>

```

BTNode G = {NULL, NULL, 'G'};
BTNode H = {NULL, NULL, 'H'};
BTNode I = {NULL, NULL, 'I'};
BTNode J = {NULL, NULL, 'J'};
BTNode K = {NULL, NULL, 'K'};

// Tworzymy kolejnych ojców

BTNode D = { &H, &I, 'D'};
BTNode E = { &J, NULL, 'E'};
BTNode F = { &K, NULL, 'F'};
BTNode B = { &D, &E, 'B'};
BTNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTNode A = { &B, &C, 'A'};

// Rekurencyjna funkcja preorder

void preorder(BTNode * v)
{
    if(v)
    {
        cout << v->data << " "; // odwiedzamy węzeł
        preorder(v->left);        // przechodzimy lewe poddrzewo
        preorder(v->right);       // przechodzimy prawe poddrzewo
    }
}

int main()
{
    preorder(&A); // przejście rozpoczynamy od korzenia
    cout << endl;
    return 0;
}

```

Free Basic

```

' Przechodzenie drzew binarnych DFS:preorder
' Data: 17.01.2013
' (C)2013 mgr Jerzy Wałaszek
'-----

' Typ węzłów drzewa

Type BTNode
    Left As BTNode Ptr
    Right As BTNode Ptr
    Data As String * 1
End Type

' Tworzenie struktury drzewa rozpoczynamy od liści

Dim G As BTNode => (0, 0, "G")
Dim H As BTNode => (0, 0, "H")
Dim I As BTNode => (0, 0, "I")
Dim J As BTNode => (0, 0, "J")
Dim K As BTNode => (0, 0, "K")

' Tworzymy kolejnych ojców

Dim D As BTNode => (@H, @I, "D")
Dim E As BTNode => (@J, 0, "E")
Dim F As BTNode => (@K, 0, "F")
Dim B As BTNode => (@D, @E, "B")
Dim C As BTNode => (@F, @G, "C")

' Tworzymy korzeń drzewa

Dim A As BTNode = (@B, @C, "A")

' Rekurencyjna procedura preorder

Sub preorder(Byval v As BTNode Ptr)
    If v Then
        Print v->Data;" "; ' odwiedzamy węzeł
        preorder(v->Left) ' przechodzimy lewe poddrzewo
        preorder(v->Right) ' przechodzimy prawe poddrzewo
    End If
End Sub

preorder(@A) ' przejście rozpoczynamy od korzenia
Print
End

```

Wynik

A B D H I E J C F K G

Algorytm stosowy DFS:preorder dla drzewa binarnego

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Elementy pomocnicze:

S – stos wskazań węzłów, rozmiar stosu nie przekracza podwójnej wysokości drzewa.

Lista kroków:

- K01: Utwórz pusty stos S
 K02: S.push(v) ; zapamiętujemy wskazanie węzła startowego na stosie
 K03: Dopóki S.empty() = false, wykonuj K04...K08 ; w pętli przetwarzamy kolejne węzły

```

K04:  v ← S.top()                ; pobieramy ze stosu wskazanie węzła
K05:  S.pop()                    ; pobrane wskazanie usuwamy ze stosu
K06:  Odwiedź węzeł wskazany przez v      ; przetwarzamy węzeł
K07:  Jeśli (v → right) ≠ nil, to S.push(v → right) ; umieszczamy na stosie wskazanie prawego syna, jeśli istnieje
K08:  Jeśli (v → left) ≠ nil, to S.push(v → left)  ; umieszczamy na stosie lewego syna, jeśli istnieje i kontynuujemy pętlę aż do wyczerpania stosu
K09:  Zakończ
    
```

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego. Danymi węzłów są znaki A, B, C, Po utworzeniu drzewa program przechodzi je za pomocą algorytmu preorder, wypisując odwiedzane kolejno węzły. Przy przechodzeniu drzewa korzysta z prostego stosu.

Lazarus
<pre> // Przechodzenie drzew binarnych DFS:preorder // Data: 17.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- program DFS_preorder; // Typ węzłów drzewa type PBTNode = ^BTNode; BTNode = record left : PBTNode; right : PBTNode; data : char; end; // Tworzenie struktury drzewa rozpoczynamy od liści var G : BTNode = (left:nil; right:nil; data:'G'); H : BTNode = (left:nil; right:nil; data:'H'); I : BTNode = (left:nil; right:nil; data:'I'); J : BTNode = (left:nil; right:nil; data:'J'); K : BTNode = (left:nil; right:nil; data:'K'); // Tworzymy kolejnych ojców D : BTNode = (left: @H; right: @I; data:'D'); E : BTNode = (left: @J; right:nil; data:'E'); F : BTNode = (left: @K; right:nil; data:'F'); B : BTNode = (left: @D; right: @E; data:'B'); C : BTNode = (left: @F; right: @G; data:'C'); // Korzeń drzewa A : BTNode = (left: @B; right: @C; data:'A'); S : array[0..6] of PBTNode; // stos sptr : Integer; // wskaźnik stosu v : PBTNode; begin S[0] := @A; // na stosie umieszczamy adres korzenia sptr := 1; // stos zawiera 1 element while sptr > 0 do begin dec(sptr); v := S[sptr]; // pobieramy ze stosu wskazanie węzła write(v^.data, ' '); // przetwarzamy węzeł // Na stosie umieszczamy wskazania dzieci, jeśli istnieją if v^.right <> nil then begin S[sptr] := v^.right; inc(sptr); end; if v^.left <> nil then begin S[sptr] := v^.left; inc(sptr); end; end; writeln; end. </pre>
Code::Blocks
<pre> // Przechodzenie drzew binarnych DFS:preorder // Data: 17.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- #include <iostream> using namespace std; // Typ węzłów drzewa </pre>

```

struct BTreeNode
{
    BTreeNode * left;
    BTreeNode * right;
    char data;
};

// Tworzenie struktury drzewa rozpoczynamy od liści

BTreeNode G = {NULL, NULL, 'G'};
BTreeNode H = {NULL, NULL, 'H'};
BTreeNode I = {NULL, NULL, 'I'};
BTreeNode J = {NULL, NULL, 'J'};
BTreeNode K = {NULL, NULL, 'K'};

// Tworzymy kolejnych ojców

BTreeNode D = { &H, &I, 'D'};
BTreeNode E = { &J, NULL, 'E'};
BTreeNode F = { &K, NULL, 'F'};
BTreeNode B = { &D, &E, 'B'};
BTreeNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTreeNode A = { &B, &C, 'A'};

int main()
{
    BTreeNode * v, * S[7]; // stos
    int sptr; // wskaźnik stosu

    S[0] = &A; // na stosie umieszczamy wskazanie korzenia
    sptr = 1; // stos zawiera 1 element

    while(sptr)
    {
        v = S[--sptr]; // pobieramy ze stosu wskazanie węzła

        cout << v->data << " "; // przetwarzamy węzeł

        // na stosie umieszczamy wskazania dzieci, jeśli istnieją

        if(v->right) S[sptr++] = v->right;

        if(v->left) S[sptr++] = v->left;
    }

    cout << endl;

    return 0;
}

```

Free Basic

```

' Przechodzenie drzew binarnych DFS:preorder
' Data: 17.01.2013
' (C)2013 mgr Jerzy Wałaszek
'-----

' Typ węzłów drzewa

Type BTreeNode
    Left As BTreeNode Ptr
    Right As BTreeNode Ptr
    Data As String * 1
End Type

' Tworzenie struktury drzewa rozpoczynamy od liści

Dim G As BTreeNode => (0, 0, "G")
Dim H As BTreeNode => (0, 0, "H")
Dim I As BTreeNode => (0, 0, "I")
Dim J As BTreeNode => (0, 0, "J")
Dim K As BTreeNode => (0, 0, "K")

' Tworzymy kolejnych ojców

Dim D As BTreeNode => (@H, @I, "D")
Dim E As BTreeNode => (@J, 0, "E")
Dim F As BTreeNode => (@K, 0, "F")
Dim B As BTreeNode => (@D, @E, "B")
Dim C As BTreeNode => (@F, @G, "C")

' Tworzymy korzeń drzewa

Dim A As BTreeNode => (@B, @C, "A")

Dim S(6) As BTreeNode Ptr ' stos
Dim sptr As Integer ' wskaźnik stosu
Dim v As BTreeNode Ptr ' wskaźnik węzła

S(0) = @A ' na stosie umieszczamy wskazanie korzenia
sptr = 1 ' stos zawiera 1 element

While sptr > 0
    sptr -= 1
    v = S(sptr) ' pobieramy ze stosu wskazanie węzła

    Print v->Data; " "; ' przetwarzamy węzeł

    ' na stosie umieszczamy wskazania dzieci, jeśli istnieją

    If v->Right Then
        S(sptr) = v->Right
        sptr += 1
    End If

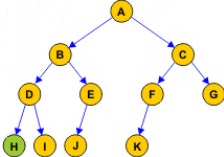
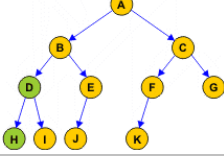
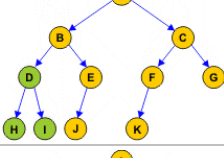
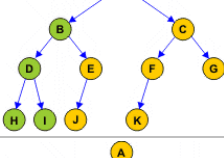
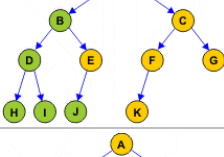
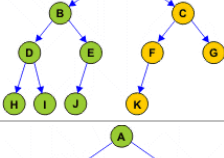
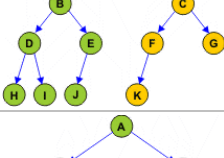
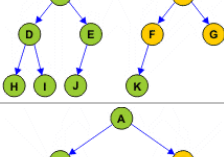
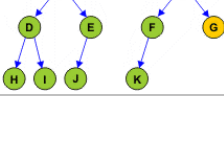
    If v->Left Then
        S(sptr) = v->Left
    End If
End While

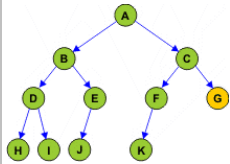
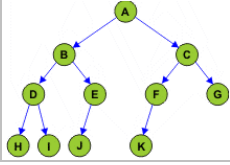
```

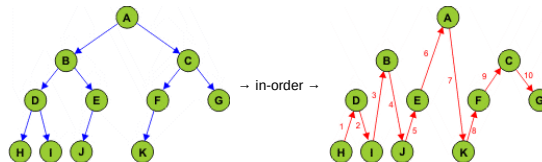
<pre> sptr += 1 End If Wend Print End </pre>	
	Wynik
A B D H I E J C F K G	

DFS: in-order – przejście poprzeczne

W przejściu poprzecznym ([ang. in-order traversal](#)) najpierw przechodzimy lewe poddrzewo danego węzła, następnie odwiedzamy węzeł i na koniec przechodzimy prawe poddrzewo.

Stan przejścia	Przetworzone węzły	Opis
		Rozpoczynamy w korzeniu A. Jednakże węzła A nie przetwarzamy, lecz przechodzimy do lewego syna B. Tutaj dalej przechodzimy do D i H. Węzeł H nie ma dzieci, więc przechodzenie do lewego poddrzewa się kończy i węzeł H zostaje przetworzony jako pierwszy.
	H D	Lewe poddrzewo węzła D zostało przebyte, zatem przetwarzamy węzeł D i przechodzimy do prawego poddrzewa, którego korzeniem jest węzeł I.
	H D I	Węzeł I jest liściem, zatem nie posiada lewego poddrzewa. Przetwarzamy węzeł I.
	H D I B	Lewe poddrzewo węzła B zostało przebyte. Przetwarzamy węzeł B i przechodzimy do jego prawego poddrzewa rozpoczynającego się w węźle E.
	H D I B J	Będąc w węźle E najpierw przechodzimy jego lewe poddrzewo, czyli idziemy do węzła J. Węzeł J jest liściem i nie posiada dalszych poddrzew. Przetwarzamy J i wracamy do E.
	H D I B J E	Lewe poddrzewo węzła E zostało przebyte. Przetwarzamy węzeł E. Ponieważ nie posiada on prawego poddrzewa, wracamy do A.
	H D I B J E A	Lewe poddrzewo węzła A zostało przebyte. Przetwarzamy A i przechodzimy do prawego poddrzewa rozpoczynającego się od węzła C.
	H D I B J E A K	Będąc w węźle C przechodzimy do lewego poddrzewa rozpoczynającego się od węzła F, a tam dalej przechodzimy do następnego lewego poddrzewa, czyli do węzła K. Ten węzeł jest liściem i nie posiada dalszych poddrzew. Przetwarzamy K i wracamy do F.
	H D I B J E A K F	Lewe poddrzewo węzła F zostało przebyte. Przetwarzamy F. Ponieważ brak prawego poddrzewa, wracamy do C.

	H D I B J E A K F C	Lewe poddrzewo węzła C zostało przebyte, przetwarzamy C i przechodzimy do prawego poddrzewa, czyli do węzła G.
	H D I B J E A K F C G	G jest liściem, przetwarzamy G i kończymy.



Algorytm rekurencyjny DFS:inorder dla drzewa binarnego

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Lista kroków:

- K01: Jeśli v = nil, **to zakończ** ; koniec rekurencji
 K02: inorder(v → left) ; przejdź rekurencyjnie przez lewe poddrzewo
 K03: Odwiedź węzeł wskazany przez v
 K04: inorder(v → right) ; przejdź rekurencyjnie przez prawe poddrzewo
 K05: **Zakończ**

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego jak w przykładzie powyżej. Danymi węzłów są znaki A, B, C, Po utworzeniu drzewa program przechodzi je za pomocą algorytmu inorder, wypisując odwiedzane kolejno węzły.

```

Lazarus

// Przechodzenie drzew binarnych DFS:inorder
// Data: 17.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

program DFS_inorder;

// Typ węzłów drzewa

type
  PBTNode = ^BTNode;
  BTNode = record
    left : PBTNode;
    right : PBTNode;
    data : char;
  end;

// Tworzenie struktury drzewa rozpoczynamy od liści

var
  G : BTNode = (left:nil; right:nil; data:'G');
  H : BTNode = (left:nil; right:nil; data:'H');
  I : BTNode = (left:nil; right:nil; data:'I');
  J : BTNode = (left:nil; right:nil; data:'J');
  K : BTNode = (left:nil; right:nil; data:'K');

// Tworzymy kolejnych ojców

D : BTNode = (left:@H; right:@I; data:'D');
E : BTNode = (left:@J; right:nil; data:'E');
F : BTNode = (left:@K; right:nil; data:'F');
B : BTNode = (left:@D; right:@E; data:'B');
C : BTNode = (left:@F; right:@G; data:'C');

// Tworzymy korzeń drzewa

A : BTNode = (left:@B; right:@C; data:'A');

// Rekurencyjna procedura inorder
    
```



```

procedure inorder(v : PBTNode);
begin
  if v <> nil then
    begin
      inorder(v^.left); // przechodzimy lewe poddrzewo
      write(v^.data, ' '); // odwiedzamy węzeł
      inorder(v^.right); // przechodzimy prawe poddrzewo
    end;
  end;

  begin
    inorder(@A); // przejście rozpoczynamy od korzenia
    writeln;
  end.

```

Code::Blocks

```

// Przechodzenie drzew binarnych DFS:inorder
// Data: 17.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

#include <iostream>

using namespace std;

// Typ węzłów drzewa

struct BTreeNode
{
  BTreeNode * left;
  BTreeNode * right;
  char data;
};

// Tworzenie struktury drzewa rozpoczynamy od liści

BTreeNode G = {NULL, NULL, 'G'};
BTreeNode H = {NULL, NULL, 'H'};
BTreeNode I = {NULL, NULL, 'I'};
BTreeNode J = {NULL, NULL, 'J'};
BTreeNode K = {NULL, NULL, 'K'};

// Tworzymy kolejnych ojców

BTreeNode D = { &H, &I, 'D'};
BTreeNode E = { &J, NULL, 'E'};
BTreeNode F = { &K, NULL, 'F'};
BTreeNode B = { &D, &E, 'B'};
BTreeNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTreeNode A = { &B, &C, 'A'};

// Rekurencyjna funkcja inorder

void inorder(BTreeNode * v)
{
  if(v)
  {
    inorder(v->left); // przechodzimy lewe poddrzewo
    cout << v->data << " "; // odwiedzamy węzeł
    inorder(v->right); // przechodzimy prawe poddrzewo
  }
}

int main()
{
  inorder(&A); // przejście rozpoczynamy od korzenia
  cout << endl;
  return 0;
}

```

Free Basic

```

' Przechodzenie drzew binarnych DFS:inorder
' Data: 17.01.2013
' (C)2013 mgr Jerzy Wałaszek
'-----

' Typ węzłów drzewa

Type BTreeNode
  Left As BTreeNode Ptr
  Right As BTreeNode Ptr
  Data As String * 1
End Type

' Tworzenie struktury drzewa rozpoczynamy od liści

Dim G As BTreeNode => (0, 0, "G")
Dim H As BTreeNode => (0, 0, "H")
Dim I As BTreeNode => (0, 0, "I")
Dim J As BTreeNode => (0, 0, "J")
Dim K As BTreeNode => (0, 0, "K")

' Tworzymy kolejnych ojców

Dim D As BTreeNode => (@H, @I, "D")
Dim E As BTreeNode => (@J, 0, "E")
Dim F As BTreeNode => (@K, 0, "F")
Dim B As BTreeNode => (@D, @E, "B")
Dim C As BTreeNode => (@F, @G, "C")

' Tworzymy korzeń drzewa

Dim A As BTreeNode => (@B, @C, "A")

' Rekurencyjna procedura inorder

```

<pre> Sub inorder(Byval v As BTreeNode Ptr) If v Then inorder(v->Left) ' przechodzimy lewe poddrzewo Print v->Data;" "; ' odwiedzamy węzeł inorder(v->Right) ' przechodzimy prawe poddrzewo End If End Sub inorder(@A) ' przejście rozpoczynamy od korzenia Print End </pre>
Wynik
H D I B J E A K F C G

Algorytm stosowy DFS:inorder dla drzewa binarnego

Algorytm wykorzystuje stos oraz dodatkowy wskaźnik cp, którym przemieszczamy się po drzewie.

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Elementy pomocnicze:

S – stos wskazań węzłów, rozmiar stosu nie przekracza podwójnej wysokości drzewa.

cp – wskaźnik bieżącego węzła

Lista kroków:

```

K01: Utwórz pusty stos S
K02: cp ← v ; bieżącym węzłem jest korzeń drzewa
K03: Dopóki (S.empty() = false) ∨ (cp ≠ nil) wykonuj K04...K11 ; pętla jest wykonywana, jeśli jest coś na stosie lub cp wskazuje węzeł drzewa
K04: Jeśli cp = nil, to idź do K07 ; sprawdzamy, czy wyszliśmy poza liść drzewa
K05: S.push(cp) ; jeśli nie, to umieszczamy wskazanie bieżącego węzła na stosie
K06: cp ← (cp->left) ; bieżącym węzłem staje się lewy syn
K07: Następnym obieg pętli K02 ; wracamy na początek pętli
K08: cp ← S.top() ; wyszliśmy poza liść, wracamy do węzła, pobierając jego wskazanie ze stosu
K09: S.pop() ; wskazanie usuwamy ze stosu
K10: Odwiedź węzeł wskazany przez cp ; przetwarzamy dane w węźle
K11: cp ← (cp->right) ; bieżącym węzłem staje się prawy syn
K12: Zakończ
        
```

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego. Danymi węzłów są znaki A, B, C, Po utworzeniu drzewa program przechodzi je za pomocą algorytmu inorder, wypisując odwiedzane kolejne węzły. Przy przechodzeniu drzewa korzysta z prostego stosu.

<pre> // Przechodzenie drzew binarnych DFS:inorder // Data: 18.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- program DFS_preorder; // Typ węzłów drzewa type PBTNode = ^BTreeNode; BTreeNode = record left : PBTNode; right : PBTNode; data : char; end; // Tworzenie struktury drzewa rozpoczynamy od liści var G : BTreeNode = (left:nil; right:nil; data:'G'); H : BTreeNode = (left:nil; right:nil; data:'H'); I : BTreeNode = (left:nil; right:nil; data:'I'); J : BTreeNode = (left:nil; right:nil; data:'J'); K : BTreeNode = (left:nil; right:nil; data:'K'); // Tworzymy kolejnych ojców D : BTreeNode = (left: @H; right: @I; data:'D'); E : BTreeNode = (left: @J; right:nil; data:'E'); F : BTreeNode = (left: @K; right:nil; data:'F'); B : BTreeNode = (left: @D; right: @E; data:'B'); C : BTreeNode = (left: @F; right: @G; data:'C'); // Korzeń drzewa A : BTreeNode = (left: @B; right: @C; data:'A'); S : array[0..6] of PBTNode; // stos sptr : Integer; // wskaźnik stosu </pre>

```

    cp : PBTNode;           // wskaźnik bieżącego węzła

begin

    sptr := 0;              // pusty stos

    cp := @A;               // cp ustawiamy na korzeń drzewa

    while (sptr > 0) or (cp <> nil) do
        if cp <> nil then
            begin
                S[sptr] := cp; // umieszczamy cp na stosie
                inc(sptr);
                cp := cp^.left; // cp staje się lewym synem
            end
        else
            begin
                dec(sptr);
                cp := S[sptr]; // pobieramy wskazanie ze stosu
                write(cp^.data, ' ');
                cp := cp^.right; // cp staje się prawym synem
            end;
        writeln;
    end.

```

Code::Blocks

```

// Przechodzenie drzew binarnych DFS:inorder
// Data: 18.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

#include <iostream>

using namespace std;

// Typ węzłów drzewa

struct BTreeNode
{
    BTreeNode * left;
    BTreeNode * right;
    char data;
};

// Tworzenie struktury drzewa rozpoczynamy od liści

BTreeNode G = {NULL, NULL, 'G'};
BTreeNode H = {NULL, NULL, 'H'};
BTreeNode I = {NULL, NULL, 'I'};
BTreeNode J = {NULL, NULL, 'J'};
BTreeNode K = {NULL, NULL, 'K'};

// Tworzymy kolejnych ojców

BTreeNode D = { &H, &I, 'D'};
BTreeNode E = { &J, NULL, 'E'};
BTreeNode F = { &K, NULL, 'F'};
BTreeNode B = { &D, &E, 'B'};
BTreeNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTreeNode A = { &B, &C, 'A'};

int main()
{
    BTreeNode * cp, * S[7]; // stos
    int sptr = 0;           // wskaźnik stosu

    cp = &A;                // cp ustawiamy na korzeń drzewa

    while(sptr || cp)
    {
        if(cp)
        {
            S[sptr++] = cp; // umieszczamy cp na stosie
            cp = cp->left;   // cp staje się lewym synem
        }
        else
        {
            cp = S[--sptr]; // pobieramy wskazanie ze stosu
            cout << cp->data << " ";
            cp = cp->right;  // cp staje się prawym synem
        }

        cout << endl;
    }

    return 0;
}

```

Free Basic

```

' Przechodzenie drzew binarnych DFS:inorder
' Data: 18.01.2013
' (C)2013 mgr Jerzy Wałaszek
'-----

' Typ węzłów drzewa

Type BTreeNode
Left As BTreeNode Ptr
Right As BTreeNode Ptr
Data As String * 1
End Type

' Tworzenie struktury drzewa rozpoczynamy od liści

Dim G As BTreeNode => (0, 0, "G")
Dim H As BTreeNode => (0, 0, "H")
Dim I As BTreeNode => (0, 0, "I")

```

```

Dim J As BTNode => (@H, @I, "J")
Dim K As BTNode => (@, @, "K")

' Tworzymy kolejnych ojców

Dim D As BTNode => (@H, @I, "D")
Dim E As BTNode => (@J, @, "E")
Dim F As BTNode => (@K, @, "F")
Dim B As BTNode => (@D, @E, "B")
Dim C As BTNode => (@F, @G, "C")

' Tworzymy korzeń drzewa

Dim A As BTNode => (@B, @C, "A")

Dim S(6) As BTNode Ptr ' stos
Dim sptr As Integer ' wskaźnik stosu
Dim cp As BTNode Ptr ' wskaźnik bieżącego węzła

sptr = 0 ' pusty stos
cp = @A ' cp ustawiamy na korzeń drzewa

While (sptr > 0) Or (cp <> 0)
  If cp Then
    S(sptr) = cp ' umieszczamy cp na stosie
    sptr += 1
    cp = cp->Left ' cp staje się lewym synem
  Else
    sptr -= 1
    cp = S(sptr) ' pobieramy wskazanie ze stosu
    Print cp->Data, " ";
    cp = cp->Right ' cp staje się prawym synem
  End If
Wend

Print

End

```

Wynik

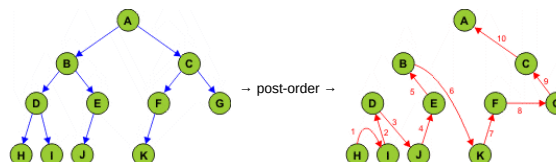
H D I B J E A K F C G

DFS: post-order – przejście wsteczne

W przejściu wstecznym (ang. [post-order traversal](#)) najpierw przechodzimy lewe poddrzewo, następnie prawe, a dopiero na końcu przetwarzamy węzeł.

Stan przejścia	Przetworzone węzły	Opis
	H	Rozpoczynamy w korzeniu A. Najpierw przechodzimy lewe poddrzewo. Idziemy do B. Tutaj również przechodzimy lewe poddrzewo i idziemy do D, a następnie do H. Węzeł H jest liściem i nie posiada poddrzew. Przetwarzamy H i wracamy do D.
	H I	Lewe poddrzewo węzła D jest przebyte, przechodzimy poddrzewo prawe. Idziemy do węzła I. Ponieważ jest on liściem, to nie posiada poddrzew. Przetwarzamy I. Wracamy do D.
	H I D	Oba poddrzewa węzła D zostały przebyte, przetwarzamy D i wracamy do B.
	H I D J	Dla węzła B przebyte zostało poddrzewo lewe. Teraz przechodzimy przez jego prawe poddrzewo. Idziemy do E i do J (lewe poddrzewo E). J jest liściem i nie posiada dalszych poddrzew, zatem przetwarzamy J i wracamy do E.
	H I D J E	Lewe poddrzewo węzła E zostało przebyte. Nie ma prawego poddrzewa dla E, zatem przetwarzamy E i wracamy do B.
	H I D J E B	Lewe i prawe poddrzewo węzła B jest przebyte. Przetwarzamy węzeł B i wracamy do A.

	H I D J E B K	Lewe poddrzewo węzła A zostało przebyte. Przechodzimy do prawego poddrzewa, czyli do węzła C, następnie do F (lewe poddrzewo C) i do K (lewe poddrzewo F). K jest liściem i nie posiada poddrzew. Przetwarzamy K i wracamy do F.
	H I D J E B K F	Lewe poddrzewo węzła F jest przebyte, a prawego poddrzewa brak. Przetwarzamy F i wracamy do C.
	H I D J E B K F G	Lewe poddrzewo węzła C jest przebyte, przechodzimy do poddrzewa prawego, czyli do węzła G. Węzeł G jest liściem i nie posiada poddrzew. Przetwarzamy G i wracamy do C.
	H I D J E B K F G C	Oba poddrzewa węzła C zostały przebyte. Przetwarzamy C i wracamy do A.
	H I D J E B K F G C A	Oba poddrzewa węzła A zostały przebyte. Przetwarzamy A i kończymy



Algorytm rekurencyjny DFS:postorder dla drzewa binarnego

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Lista kroków:

- K01: Jeśli $v = \text{nil}$, to zakończ ; koniec rekurencji
 K02: postorder($v \rightarrow \text{left}$) ; przejdź rekurencyjnie przez lewe poddrzewo
 K03: postorder($v \rightarrow \text{right}$) ; przejdź rekurencyjnie przez prawe poddrzewo
 K04: Odwiedź węzeł wskazany przez v
 K05: Zakończ

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego jak w przykładzie powyżej. Danymi węzłów są znaki A, B, C, Po utworzeniu drzewa program przechodzi je za pomocą algorytmu postorder, wypisując odwiedzane kolejno węzły.

Lazarus
<pre>// Przechodzenie drzew binarnych DFS:postorder // Data: 19.01.2013 // (C)2013 mgr Jerzy Wałaszek //----- program DFS_postorder; // Typ węzłów drzewa</pre>

```

type
  PBTNode = ^BTNode;
  BTNode = record
    left : PBTNode;
    right : PBTNode;
    data : char;
  end;

// Tworzenie struktury drzewa rozpoczynamy od liści

var
  G : BTNode = (left:nil; right:nil; data:'G');
  H : BTNode = (left:nil; right:nil; data:'H');
  I : BTNode = (left:nil; right:nil; data:'I');
  J : BTNode = (left:nil; right:nil; data:'J');
  K : BTNode = (left:nil; right:nil; data:'K');

// Tworzymy kolejnych ojców

  D : BTNode = (left: @H; right: @I; data:'D');
  E : BTNode = (left: @J; right:nil; data:'E');
  F : BTNode = (left: @K; right:nil; data:'F');
  B : BTNode = (left: @D; right: @E; data:'B');
  C : BTNode = (left: @F; right: @G; data:'C');

// Tworzymy korzeń drzewa

  A : BTNode = (left: @B; right: @C; data:'A');

// Rekurencyjna procedura postorder

procedure postorder(v : PBTNode);
begin
  if v <> nil then
    begin
      postorder(v^.left); // przechodzimy lewe poddrzewo
      postorder(v^.right); // przechodzimy prawe poddrzewo
      write(v^.data, ' '); // odwiedzamy węzeł
    end;
  end;

begin
  postorder(@A); // przejście rozpoczynamy od korzenia
  writeln;
end.

```

Code::Blocks

```

// Przechodzenie drzew binarnych DFS:postorder
// Data: 19.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

#include <iostream>

using namespace std;

// Typ węzłów drzewa

struct BTNode
{
  BTNode * left;
  BTNode * right;
  char data;
};

// Tworzenie struktury drzewa rozpoczynamy od liści

BTNode G = {NULL,NULL,'G'};
BTNode H = {NULL,NULL,'H'};
BTNode I = {NULL,NULL,'I'};
BTNode J = {NULL,NULL,'J'};
BTNode K = {NULL,NULL,'K'};

// Tworzymy kolejnych ojców

BTNode D = { &H, &I, 'D'};
BTNode E = { &J, NULL, 'E'};
BTNode F = { &K, NULL, 'F'};
BTNode B = { &D, &E, 'B'};
BTNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTNode A = { &B, &C, 'A'};

// Rekurencyjna funkcja postorder

void postorder(BTNode * v)
{
  if(v)
  {
    postorder(v->left); // przechodzimy lewe poddrzewo
    postorder(v->right); // przechodzimy prawe poddrzewo
    cout << v->data << " "; // odwiedzamy węzeł
  }
}

int main()
{
  postorder(&A); // przejście rozpoczynamy od korzenia
  cout << endl;
  return 0;
}

```

Free Basic

```
' Przechodzenie drzew binarnych DFS:postorder
```

<pre> ' Data: 19.01.2013 ' (C)2013 mgr Jerzy Wałaszek '----- ' Typ węzłów drzewa Type BTreeNode Left As BTreeNode Ptr Right As BTreeNode Ptr Data As String * 1 End Type ' Tworzenie struktury drzewa rozpoczynamy od liści Dim G As BTreeNode => (0, 0, "G") Dim H As BTreeNode => (0, 0, "H") Dim I As BTreeNode => (0, 0, "I") Dim J As BTreeNode => (0, 0, "J") Dim K As BTreeNode => (0, 0, "K") ' Tworzymy kolejnych ojców Dim D As BTreeNode => (@H, @I, "D") Dim E As BTreeNode => (@J, 0, "E") Dim F As BTreeNode => (@K, 0, "F") Dim B As BTreeNode => (@D, @E, "B") Dim C As BTreeNode => (@F, @G, "C") ' Tworzymy korzeń drzewa Dim A As BTreeNode => (@B, @C, "A") ' Rekurencyjna procedura postorder Sub postorder(Byval v As BTreeNode Ptr) If v Then postorder(v->Left) ' przechodzimy lewe poddrzewo postorder(v->Right) ' przechodzimy prawe poddrzewo Print v->Data; " "; ' odwiedzamy węzeł End If End Sub postorder(@A) ' przejście rozpoczynamy od korzenia Print End </pre>	
Wynik	
H I D J E B K F G C A	

Algorytm stosowy DFS:postorder dla drzewa binarnego

Algorytm wykorzystuje stos oraz dwa wskaźniki: *pp* – wskazanie poprzedniego węzła i *cp* – wskazanie bieżącego węzła. Są one używane do określenia, czy oba poddrzewa danego węzła zostały przebyte.

Wejście

v – wskazanie węzła startowego drzewa binarnego

Wyjście:

przetworzenie wszystkich węzłów drzewa.

Elementy pomocnicze:

S – stos wskazań węzłów, rozmiar stosu nie przekracza podwójnej wysokości drzewa.

pp – wskaźnik węzła poprzedniego

cp – wskaźnik węzła bieżącego.

Lista kroków:

```

K01: Utwórz pusty stos S
K02: S.push(v) ; węzeł startowy umieszczamy na stosie
K03: pp ← nil ; zerujemy wskaźnik węzła poprzedniego
K04: Dopóki S.empty() = false: wykonuj K05...K14
K05: cp ← S.top() ; ustawiamy cp na węzeł przechowywany na stosie
K06: Jeśli (pp = nil) ∨ ((pp → left) = cp) ∨ ((pp → right) = cp), to idź do K11 ; sprawdzamy, czy idziemy w głąb drzewa
K07: Jeśli (cp → left) = pp, to idź do K13 ; sprawdzamy, czy wróciliśmy z lewego poddrzewa
K08: Odwiedź węzeł wskazany przez cp ; oba poddrzewa przebyte, przetwarzamy węzeł
K09: S.pop() ; i usuwamy jego wskazanie ze stosu
K10: Idź do K14
K11: Jeśli (cp → left) ≠ nil, to S.push(cp → left) ; jeśli istnieje lewy syn cp, umieszczamy go na stosie
    inaczej jeśli (cp → right) ≠ nil, to S.push(cp → right) ; inaczej umieszczamy na stosie prawego syna, jeśli istnieje
K12: Idź do K14
K13: Jeśli (cp → right) ≠ nil, to S.push(cp → right) ; jeśli istnieje prawy syn, to umieszczamy go na stosie
K14: pp ← cp ; zapamiętujemy cp w pp i wykonujemy kolejny obieg pętli
K15: Zakończ

```

Program

Ważne:

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

Program tworzy strukturę drzewa binarnego. Danymi węzłów są znaki A. B. C, Po utworzeniu drzewa program przechodzi je za pomocą algorytmu postorder, wypisując odwiedzane kolejno węzły. Przy przechodzeniu drzewa korzysta z prostego stosu.

Lazarus

```

// Przechodzenie drzew binarnych DFS:postorder
// Data: 19.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

program DFS_postorder;

// Typ węzłów drzewa

type
  PBTNode = ^BTNode;
  BTNode = record
    left : PBTNode;
    right : PBTNode;
    data : char;
  end;

// Tworzenie struktury drzewa rozpoczynamy od liści

var
  G : BTNode = (left:nil; right:nil; data:'G');
  H : BTNode = (left:nil; right:nil; data:'H');
  I : BTNode = (left:nil; right:nil; data:'I');
  J : BTNode = (left:nil; right:nil; data:'J');
  K : BTNode = (left:nil; right:nil; data:'K');

// Tworzymy kolejnych ojców

  D : BTNode = (left:@H; right:@I; data:'D');
  E : BTNode = (left:@J; right:nil; data:'E');
  F : BTNode = (left:@K; right:nil; data:'F');
  B : BTNode = (left:@D; right:@E; data:'B');
  C : BTNode = (left:@F; right:@G; data:'C');

// Korzeń drzewa

  A : BTNode = (left:@B; right:@C; data:'A');

  S : array[0..6] of PBTNode; // stos
  sptr : Integer; // wskaźnik stosu
  pp,cp : PBTNode;

begin
  S[0] := @A; // węzeł startowy
  sptr := 1;

  pp := nil;

  while sptr > 0 do
  begin
    cp := S[sptr - 1];
    if (pp = nil) or (pp^.left = cp) or (pp^.right = cp) then
    begin
      if cp^.left <> nil then
      begin
        S[sptr] := cp^.left;
        inc(sptr);
      end
      else if cp^.right <> nil then
      begin
        S[sptr] := cp^.right;
        inc(sptr);
      end
      end
      else if cp^.left = pp then
      begin
        if cp^.right <> nil then
        begin
          S[sptr] := cp^.right;
          inc(sptr);
        end
        end
        else
        begin
          write(cp^.data, ' ');
          dec(sptr);
        end;
        pp := cp;
      end;

      writeln;
    end.
  end.

```

Code::Blocks

```

// Przechodzenie drzew binarnych DFS:postorder
// Data: 19.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----

#include <iostream>

using namespace std;

// Typ węzłów drzewa

struct BTNode
{
  BTNode * left;
  BTNode * right;
  char data;
};

// Tworzenie struktury drzewa rozpoczynamy od liści

BTNode G = {NULL, NULL, 'G'};
BTNode H = {NULL, NULL, 'H'};
BTNode I = {NULL, NULL, 'I'};

```



```

BTNode J = {NULL, NULL, 'J'};
BTNode K = {NULL, NULL, 'K'};

// Tworzymy kolejnych ojców

BTNode D = { &H, &I, 'D'};
BTNode E = { &J, NULL, 'E'};
BTNode F = { &K, NULL, 'F'};
BTNode B = { &D, &E, 'B'};
BTNode C = { &F, &G, 'C'};

// Tworzymy korzeń drzewa

BTNode A = { &B, &C, 'A'};

int main()
{
    BTNode * S[7]; // stos
    int sptr;      // wskaźnik stosu
    BTNode *pp, *cp;

    S[0] = &A;    // węzeł startowy
    sptr = 1;

    pp = NULL;

    while(sptr)
    {
        cp = S[sptr - 1];
        if(!pp || pp->left == cp || pp->right == cp)
        {
            if(cp->left) S[sptr++] = cp->left;
            else if(cp->right) S[sptr++] = cp->right;
        }
        else if(cp->left == pp)
        {
            if(cp->right) S[sptr++] = cp->right;
        }
        else
        {
            cout << cp->data << " ";
            sptr--;
        }
        pp = cp;
    }

    cout << endl;

    return 0;
}

```

Free Basic

```

' Przechodzenie drzew binarnych DFS:postorder
' Data: 19.01.2013
' (C)2013 mgr Jerzy Wałaszek
' -----

' Typ węzłów drzewa

Type BTNode
    Left As BTNode Ptr
    Right As BTNode Ptr
    Data As String * 1
End Type

' Tworzenie struktury drzewa rozpoczynamy od liści

Dim G As BTNode => (0, 0, "G")
Dim H As BTNode => (0, 0, "H")
Dim I As BTNode => (0, 0, "I")
Dim J As BTNode => (0, 0, "J")
Dim K As BTNode => (0, 0, "K")

' Tworzymy kolejnych ojców

Dim D As BTNode => (@H, @I, "D")
Dim E As BTNode => (@J, 0, "E")
Dim F As BTNode => (@K, 0, "F")
Dim B As BTNode => (@D, @E, "B")
Dim C As BTNode => (@F, @G, "C")

' Tworzymy korzeń drzewa

Dim A As BTNode => (@B, @C, "A")

Dim S(6) As BTNode Ptr ' stos
Dim sptr As Integer    ' wskaźnik stosu
Dim As BTNode Ptr pp, cp

S(0) = @A              ' węzeł startowy
sptr = 1

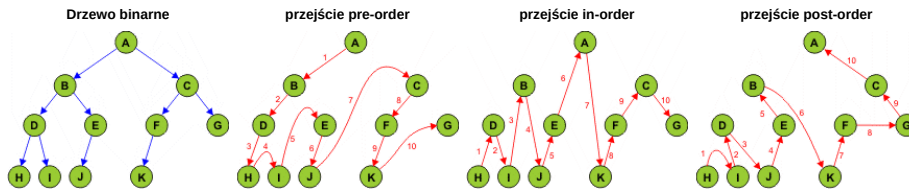
pp = 0

While sptr > 0
    cp = S(sptr - 1)
    If (pp = 0) OrElse (pp->Left = cp) OrElse (pp->Right = cp) Then
        If cp->Left Then
            S(sptr) = cp->Left
            sptr += 1
        ElseIf cp->Right Then
            S(sptr) = cp->Right
            sptr += 1
        End If
    ElseIf cp->Left = pp Then
        If cp->Right Then
            S(sptr) = cp->Right
            sptr += 1
        End If
    Else

```

<pre>Print cp->Data;" "; sptr -= 1 End If pp = cp Wend Print End</pre>
Wynik
H I D J E B K F G C A

Podsumowanie



Dokument ten rozpowszechniany jest zgodnie z zasadami licencji

GNU Free Documentation License.

Pytania proszę przysyłać na adres email: i-lo@eduinf.waw.pl

W artykułach serwisu są używane cookies. Jeśli nie chcesz ich otrzymywać, zablokuj je w swojej przeglądarce.

[Informacje dodatkowe](#)



I Liceum Ogólnokształcące
im. Kazimierza Brodzińskiego
w Warszawie
©2019 mgr Jerzy Walaszek