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

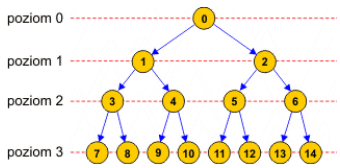
Przechodzenie drzew binarnych – BFS

Tematy pokrewne

Problem

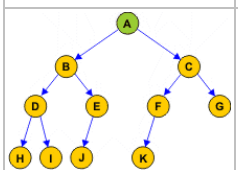
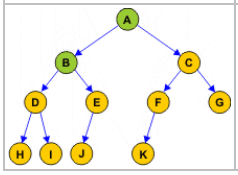
Dokonać przejścia wszerz przez wszystkie węzły drzewa binarnego.

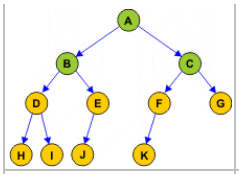
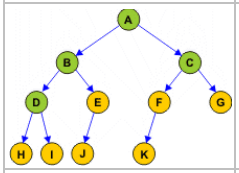
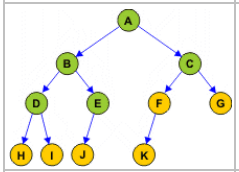
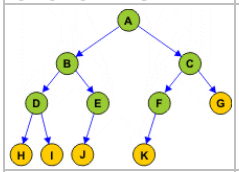
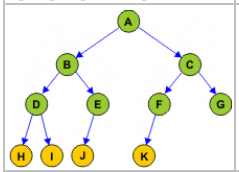
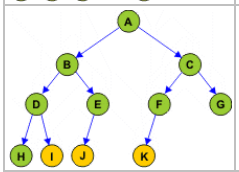
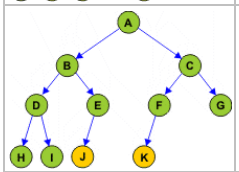
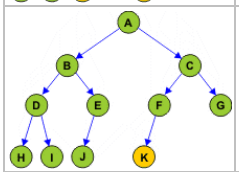
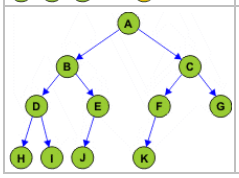
Przechodzenie drzewa binarnego wszerz (ang. breadth-first search, BFS) polega na odwiedzaniu kolejnych węzłów leżących na kolejnych poziomach.



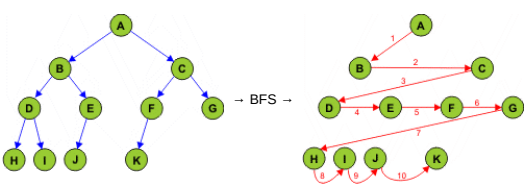
Najpierw odwiedzamy korzeń drzewa – węzeł nr 0, który znajduje się na poziomie 0. Dalej przechodzimy do jego synów i odwiedzamy węzły nr 1 i 2 na poziomie 1. W kolejnym kroku przechodzimy do poziomu 2 i odwiedzamy kolejno węzły nr 3, 4, 5 i 6. Operację tę kontynuujemy, aż do przejścia przez wszystkie węzły drzewa.

Tego typu przejście wymaga zapamiętywania wskazań węzłów w kolejce. Kolejka pozwala odczytywać umieszczone w niej elementy w tej samej kolejności, w jakiej zostały do niej wstawione, czyli jest jakby buforem opóźniającym. Prześledźmy przejście BFS przedstawione w poniższej tabelce (dane dopisujemy na koniec kolejki, czyli z prawej strony, a pobieramy z początku kolejki, czyli z lewej strony):

Stan przejścia	Kolejka	Przetworzone węzły	Opis
	[A]		Umieszczamy w kolejce węzeł startowy, czyli A. Przejście wykonywane jest w pętli dotąd, aż kolejka stanie się pusta.
	[B C]	A	Pobieramy z kolejki węzeł A. Odwiedzamy go. W kolejce umieszczamy dzieci węzła A, czyli węzły B i C
	[C D E]	A B	Pobieramy z kolejki węzeł B. Odwiedzamy go. W kolejce umieszczamy synów D i E.

	[D E **F G**]	A B **C**	Pobieramy z kolejki węzeł **C**. Odwiedzamy go. W kolejce umieszczamy synów **F** i **G**.
	[E F G **H I**]	A B C **D**	Pobieramy z kolejki węzeł **D**. Odwiedzamy go. W kolejce umieszczamy synów **H** i **I**.
	[F G H I **J**]	A B C D **E**	Pobieramy z kolejki węzeł **E**. Odwiedzamy go. W kolejce umieszczamy syna **J**
	[G H I J **K**]	A B C D E **F**	Pobieramy z kolejki węzeł **F**. Odwiedzamy go. W kolejce umieszczamy syna **K**
	[H I J K]	A B C D E F **G**	Pobieramy z kolejki węzeł **G**. Odwiedzamy go. Węzeł jest liściem, więc nic nie umieszczamy w kolejce.
	[I J K]	A B C D E F G **H**	Pobieramy z kolejki węzeł **H**. Odwiedzamy go. Węzeł jest liściem, więc nic nie umieszczamy w kolejce.
	[J K]	A B C D E F G H **I**	Pobieramy z kolejki węzeł **I**. Odwiedzamy go.
	[K]	A B C D E F G H I **J**	Pobieramy z kolejki węzeł **J**. Odwiedzamy go.
	[pusta]	A B C D E F G H I J **K**	Pobieramy z kolejki węzeł **K**. Odwiedzamy go. Kończymy przechodzenie drzewa, ponieważ kolejka jest już pusta.

Na poniższym rysunku zaznaczyliśmy linię przejścia przez kolejne węzły drzewa.



Rozmiar kolejki nie przekracza 2^h, gdzie h jest wysokością drzewa.

Algorytm kolejkowy BFS dla drzewa binarnego

Wejście

v – wskazanie węzła startowego drzewa binarnego
h – wysokość drzewa

Wyjście:

przetworzenie wszystkich węzłów drzewa kolejnymi poziomami od strony lewej do prawej.

Elementy pomocnicze:

Q – kolejka, której elementy są wskazaniami węzłów drzewa. Długość kolejki jest równa 2^h.

Lista kroków:

K01: **Utwórz** pustą kolejkę *Q*
K02: *Q*.push(*v*) *; zapamiętujemy wskazanie węzła startowego w kolejce*
K03: **Dopóki** *Q*.empty() = false: **wykonuj** K04...K08
K04: *v* ← *Q*.front() *; pobieramy wskazanie węzła z początku kolejki*
K05: *Q*.pop() *; usuwamy pobrany element z kolejki*
K06: Odwiedź węzeł wskazywany przez *v* *; przetwarzamy węzeł*
K07: **Jeśli** (*v*→*left*) ≠ nil, **to** *Q*.push(*v*→*left*) *; jeśli węzeł ma lewego syna, to umieszczamy jego wskazanie w kolejce*
K08: **Jeśli** (*v*→*right*) ≠ nil, **to** *Q*.push(*v*→*right*) *; jeśli węzeł ma prawego syna, to umieszczamy jego wskazanie w kolejce*
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 jak w przykładzie powyżej. Danymi węzłów są znaki A. B. C, ... Po utworzeniu drzewa program przechodzi je za pomocą algorytmu BFS. Wykorzystuje obiekt prostej kolejki.

Lazarus

```
// Przechodzenie drzew binarnych BFS
// Data: 23.01.2013
// (C)2013 mgr Jerzy Wałaszek
//-----------------------------

program BFS;

// Typ węzłów drzewa

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

// Typ tablicy wskazań węzłów

  TBTNode = array of PBTNode;

// Definicja typu obiektowego queue
//---------------------------------

  queue = object
    private
      n    : integer;    // rozmiar tablicy
      qptr : integer;    // wskaźnik początku kolejki
      qcnt : integer;    // licznik elementów
      Q    : TBTNode; // tablica dynamiczna wskazań węzłów drzewa

    public
      constructor init(x : integer);
      destructor destroy;
      function empty : boolean;
      function front : PBTNode;
      procedure push(v : PBTNode);
      procedure pop;
  end;

//---------------------
// Metody obiektu queue
//---------------------

// Konstruktor - tworzy tablicę dla kolejki
//-----------------------------------------
constructor queue.init(x : integer);
begin
  n := x;
  SetLength(Q,x);
  qptr := 0;  // początek kolejki na początku tablicy
  qcnt := 0;  // pusta kolejka
end;

// Destruktor - zwalnia tablicę dynamiczną
//----------------------------------------
destructor queue.destroy;
begin
  SetLength(Q,0);
end;

// Sprawdza, czy kolejka jest pusta
//---------------------------------
function queue.empty : boolean;
begin
  if qcnt = 0 then empty := true
  else empty := false;
```

```
end;

// Zwraca początek kolejki.
// Wartość specjalna to nil
//----------------------------
function queue.front : PBTNode;
begin
  if qcnt = 0 then front := nil
  else             front := Q[qptr];

end;

// Zapisuje do kolejki
//--------------------
procedure queue.push(v : PBTNode);
var
  i : integer;
begin
  if qcnt < n then
  begin
    i := qptr + qcnt;
    if i >= n then dec(i,n);
    Q[i] := v;
    inc(qcnt);
  end;
end;

// Usuwa z kolejki
//----------------
procedure queue.pop;
begin
  if qcnt > 0 then
  begin
    dec(qcnt);
    inc(qptr);
    if qptr = n then qptr := 0;
  end;
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');

  Q : queue;      // kolejka

  v : PBTNode; // wskazanie węzła

begin

  Q.init(8);   // rozmiar kolejki równy 2^3, gdzie 3 jest wysokością drzewa

  Q.push(@A);  // w kolejce umieszczamy wskazanie węzła A

  while not Q.empty do
  begin

    v := Q.front;  // pobieramy element z kolejki

    Q.pop;         // pobrany element usuwamy z kolejki

    write(v^.data,' '); // odwiedzamy węzeł

    // w kolejce umieszczamy synów węzła wskazywanego przez v

    if v^.left  <> nil then Q.push(v^.left);  // lewy syn

    if v^.right <> nil then Q.push(v^.right); // prawy syn

  end;

  writeln;

  Q.destroy;

end.
```

Code::Blocks

```
// Przechodzenie drzew binarnych BFS
// Data: 23.01.2013
// (C)2013 mgr Jerzy Wałaszek
//----------------------------

#include <iostream>

using namespace std;

// Typ węzłów drzewa

struct BTNode
{
  BTNode * left;
  BTNode * right;
```

```
  char data;
};

// Definicja typu obiektowego queue
//---------------------------------
class queue
{
  private:
    int n;           // rozmiar tablicy
    int qptr;        // wskaźnik początku kolejki
    int qcnt;        // licznik elementów
    BTNode * * Q; // tablica dynamiczna wskazań węzłów

  public:
    queue(int x); // konstruktor
    ~queue();     // destruktor
    bool empty(void);
    BTNode * front(void);
    void push(BTNode * v);
    void pop(void);
};

//---------------------
// Metody obiektu queue
//---------------------

// Konstruktor - tworzy tablicę dla kolejki
//-----------------------------------------
queue::queue(int x)
{
  n = x;
  Q = new BTNode * [x]; // tworzymy tablicę x wskazań węzłów
  qptr = qcnt = 0;
}

// Destruktor - zwalnia tablicę dynamiczną
//----------------------------------------
queue::~queue()
{
  delete [] Q;
}

// Sprawdza, czy kolejka jest pusta
//---------------------------------
bool queue::empty(void)
{
  return !qcnt;
}

// Zwraca początek kolejki.
// Wartość specjalna to NULL
//--------------------------
BTNode * queue::front(void)
{
  if(qcnt) return Q[qptr];
  return NULL;
}

// Zapisuje do kolejki
//--------------------
void queue::push(BTNode * v)
{
  int i;
  if(qcnt < n)
  {
    i = qptr + qcnt++;
    if(i >= n) i -= n;
    Q[i] = v;
  }
}

// Usuwa z kolejki
//----------------
void queue::pop(void)
{
  if(qcnt)
  {
    qcnt--;
    qptr++;
    if(qptr == n) qptr = 0;
  }
}

// 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()
{
  queue Q(8);       // rozmiar kolejki równy 2^3, gdzie 3 jest wysokością drzewa

  BTNode * v;    // wskazanie węzła

  Q.push(&A);       // w kolejce umieszczamy wskazanie węzła A

  while(!Q.empty())
  {
```

```
    v = Q.front();  // pobieramy element z kolejki

    Q.pop();        // pobrany element usuwamy z kolejki

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

    // w kolejce umieszczamy synów węzła wskazywanego przez v

    if(v->left)  Q.push(v->left);  // lewy syn

    if(v->right) Q.push(v->right); // prawy syn

  }

  cout << endl;

  return 0;
}
```

Free Basic

```
' Przechodzenie drzew binarnych BFS
' Data: 23.01.2013
' (C)2013 mgr Jerzy Wałaszek
'-------------------------------

' Typ węzłów drzewa

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

' Definicja typu obiektowego queue
'---------------------------------

Type queue
  Private:
    n     As Integer            ' rozmiar tablicy
    qptr As Integer             ' wskaźnik początku kolejki
    qcnt As Integer             ' licznik elementów
    Q     As BTNode Ptr Ptr   ' tablica dynamiczna wskazań węzłów

  Public:
    Declare Constructor(Byval x As Integer)
    Declare Destructor()
    Declare Function empty() As Integer
    Declare Function front As BTNode Ptr
    Declare Sub push(Byval v As BTNode Ptr)
    Declare Sub pop()
End Type

'---------------------
' Metody obiektu queue
'---------------------

' Konstruktor - tworzy tablicę dla kolejki
'-----------------------------------------
Constructor queue(Byval x As Integer)
  n = x
  Q = New BTNode Ptr [x]
  qptr = 0
  qcnt = 0
End Constructor

' Destruktor - zwalnia tablicę dynamiczną
'----------------------------------------
Destructor queue()
  Delete [] Q
End Destructor

' Sprawdza, czy kolejka jest pusta
'---------------------------------
Function queue.empty() As Integer
  If qcnt = 0 Then Return 1
  Return 0
End Function

' Zwraca początek kolejki.
' Wartość specjalna to 0
'---------------------------
Function queue.front() As BTNode Ptr
  If qcnt = 0 Then
    front = 0
  Else
    front = Q[qptr]
  End If
End Function

' Zapisuje do kolejki
'--------------------
Sub queue.push(Byval v As BTNode Ptr)
  Dim i As Integer
  If qcnt < n Then
    i = qptr + qcnt
    If i >= n Then i -= n
    Q[i] = v
    qcnt += 1
  End If
End Sub

' Usuwa z kolejki
'----------------
Sub queue.pop()
  If qcnt > 0 Then
    qcnt -= 1
    qptr += 1
    If qptr = n Then qptr = 0
  End If
End Sub
```

```
' 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 Q As queue = 8      ' rozmiar kolejki równy 2^3, gdzie 3 jest wysokością drzewa

Dim As BTNode Ptr v  ' wskazanie węzła

Q.push(@A)              ' w kolejce umieszczamy wskazanie węzła A

While Q.empty() = 0

  v = Q.front()         ' pobieramy element z kolejki

  Q.pop()               ' pobrany element usuwamy z kolejki

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

  ' w kolejce umieszczamy synów węzła wskazywanego przez v

  If v->Left  Then Q.push(v->Left)  ' lewy syn

  If v->Right Then Q.push(v->Right) ' prawy syn

Wend

Print

End
```

Wynik

```
A B C D E F G H I J K
```