Wstęp do programowania

Drzewa - podstawowe techniki

Drzewa wyszukiwań

- Drzewa często służą do przechowywania informacji. Jeśli uda sie nam stworzyć drzewo o niewielkiej wysokości i strukturze pozwalającej w każdym węźle określić, czy iść w lewo czy w prawo w poszukiwaniu jakiejś danej, to dostęp do niej będzie szybki.
- Takie warunki spełniają drzewa binarnych wyszukiwań (BST: Binary Search Trees)

Drzewa binarnych wyszukiwań

- W drzewach binarnych wyszukiwań przechowujemy dane ze zbioru liniowo uporządkowanego (A,<).
- Przestrzegamy następujących zasad:
  - Każdą wartość reprezentujemy w drzewie co najwyżej raz,
  - Na lewo od każdego węzła są podwieszone wyłącznie wartości mniejsze od niego, a na prawo większe.

Drzewa binarnych wyszukiwań

- Przy tak określonych zasadach wiemy, jak szukać danej wartości w drzewie:
  - Albo drzewo jest puste i wtedy jej nie ma
  - Albo drzewo jest niepuste i wtedy:
    - albo wartość jest w korzeniu
    - albo wartość jest mniejsza od tej w korzeniu, więc szukamy jej w lewym poddrzewie
    - albo wartość jest większa od tej w korzeniu, więc szukamy jej w prawym poddrzewie

Wyszukiwanie wartości w drzewie zwykłym

```
function jest(x:typ; d:drzewo):Boolean;
begin
 if d=nil then jest := false
 else if d^.w=x then jest := true
 else jest := jest(x,d^.lsyn) or
                jest(x,d^.psyn)
end;
{leniwie}
{Koszt O(n)}
```

Wyszukiwanie wartości w drzewie BST

```
function jest(x:typ; d:drzewo):Boolean;
begin
 if d=nil then jest := false
 else if d^.w=x then jest := true
 else if x<d^.w then
                jest := jest(x,d^.lsyn)
      else      jest := jest(x,d^.psyn)
end;
{Koszt O(h)}
```

Wstawianie wartości do drzewa BST

```
procedure wstaw(x:typ; var d:drzewo);
begin
 if d=nil then begin
   new(d);
   d^.w:=x;
   d^.lsyn:=nil; d^.psyn:=nil;
 end
 else if x<d^.w then wstaw(x,d^.lsyn)
   else if x>d^.w then wstaw(x,d^.psyn)
end; {Gdy x jest już w drzewie, to go
  ignorujemy! Koszt całości O(h)}
```

Usuwanie wartości z drzewa BST

- Usuwanie wartości z drzewa BST jest nieco trudniejsze. Brak po danej wartości trzeba uzupełnić jednym z dwóch elementów: albo największym z lewego poddrzewa albo najmnijeszym z prawego. Szczegóły na ASD.
- Ciekawe jest pytanie o to, jaka będzie średnia głębokość węzła (albo średnia wysokość drzewa) przy budowaniu drzewa BST poprzez losowe wkładanie (i ewentualne wyjmowanie) kolejnych wartości.

Twierdzenie o infiksie w BST

- Wartości drzewa BST wypisane w obiegu infiksowym sa posortowane rosnąco.
- Dowód: indukcja ze wzgledu na liczbę węzłów drzewa:
  - dla drzewa pustego OK, bo pusty ciąg jest posortowany
  - dla drzewa niepustego mamy korzeń z podwieszonymi dwoma poddrzewami o mniejszej liczbie węzłów – stosuje się więc do nich założenie indukcyjne. Zatem wartości lewego poddrzewa, wypisywane są jako pierwsze – rosnąco na mocy założenia indukcyjnego – potem idzie wartość z korzenia – jeszcze większa (def. BST), a na końcu rosnące wartości z prawego poddrzewa. ¤

Techniki przydatne przy przetwarzaniu drzew

- Pokażemy teraz na przykładzie obliczania wysokości drzewa dwie techniki specyficzne dla drzew.
- Wysokość drzewa możemy zdefiniować rekurencyjnie tak:
  - h(nil) = -1
  - h(d) = max(h(d^.lsyn),h(d^.psyn)) + 1
- Ta definicja przekłada się natychmiast na funkcję rekurencyjną:

Wysokość drzewa – wprost xz definicji

```
function h(d:drzewo):Integer;
begin
 if d=nil then h := -1
 else h:=max(h(d^.lsyn),h(d^.psyn))+1
end;
{Koszt O(n)}
```

- Mamy tu oczywiście ukryty obieg postfiksowy: nie możemy poznac wysokości ojca, zanim nie poznamy wysokości jego synów.

Metoda spaceru

- Jest to metoda, za pomocą której przechodzimy drzewo stosownym (dowolnym) obiegiem i zauważamy pewne fakty, które nas interesują zapamiętując ich efekty na zewnętrznej zmiennej.
- W przypadku wysokości interesuje nas największa głębokość, więc przechodząc drzewo dowolnym obiegiem po prostu za każdym razem, gdy wejdziemy do nowego węzła, aktualizujemy głębokośc, na której sie znajdujemy (aktg) i sprawdzamy, czy nie pobiliśmy rekordu (maxg).

Wysokość drzewa spacerem - prefiksowo

function wysokośćSpacerowa(d:drzewo):Integer;

```
var aktg,maxg:Integer;
 procedure spacer(d:drzewo);
  begin if d<>nil then begin {spacer}
     aktg:=aktg+1;
     if aktg>maxg then maxg:=aktg;
     spacer(d^.lsyn);
     spacer(d^.psyn);
     aktg:=aktg-1; {Bardzo ważne!!!}
  end; {spacer}
begin {WysokośćSpacerowa}
  aktg:=-1; maxg:=-1;
  spacer(d);
  WysokośćSpacerowa:=maxg
end;
```

Wysokość drzewa spacerem - infiksowo

```
function wysokośćSpacerowa(d:drzewo):Integer;
var aktg,maxg:Integer;
 procedure spacer(d:drzewo);
  begin if d<>nil then begin {spacer}
     aktg:=aktg+1;
     spacer(d^.lsyn);
     if aktg>maxg then maxg:=aktg; {“infiksowo”}
     spacer(d^.psyn);
     aktg:=aktg-1; {Bardzo ważne!!!}
  end; {spacer}
begin {WysokośćSpacerowa}
  aktg:=-1; maxg:=-1;
  spacer(d);
  WysokośćSpacerowa:=maxg
end;
```

Wysokość drzewa spacerem - postfiksowo

function wysokośćSpacerowa(d:drzewo):Integer;

```
var aktg,maxg:Integer;
 procedure spacer(d:drzewo);
  begin if d<>nil then begin {spacer}
     aktg:=aktg+1;
     spacer(d^.lsyn);
     spacer(d^.psyn);
     if aktg>maxg then maxg:=aktg; {"postfiksowo"}
     aktg:=aktg-1; {Bardzo ważne!!!}
  end; {spacer}
begin {WysokośćSpacerowa}
  aktg:=-1; maxg:=-1;
  spacer(d);
  WysokośćSpacerowa:=maxg
end;
```

Przekazywanie parametrów w obiegu prefiksowym

```
procedure Nadsumuj(d:drzewo);
  procedure DodajGore(d:drzewo; k:Integer);
  {Dosumowuje wartość parametru k do węzła d i nową wartość
   przekazuje rekurencyjnie obu synom.}
   begin
    if d<> nil then
      begin
       d^.w:=d^.w+k;
       DodajGore(d^.lsyn,d^.w);
       DodajGore(d^.psyn,d^.w);
      end;
    end;
begin {Nadsumuj}
 DodajGore(d,0)
end;
```

Przekazywanie parametrów w obiegu postfiksowym

```
procedure Podsumuj(d:drzewo);
var dowol:Integer;
  procedure DodajDół(d:drzewo; var k:Integer);
   var zdolu:Integer;
   begin
   if d=nil then k:=0 {To przypisanie jest konieczne!}
   else begin
        DodajDol(d^.lsyn, zdolu);
        d^.w:=d^.w+zdolu;
        DodajDol(d^.psyn, zdolu);
        d^.w:=d^.w+zdolu;
        k:=d^.w   end  {else};
    end; {DodajDół}
 begin {Podsumuj}
  DodajDol(d,dowol) {Można było uniknąć dowola wstawiając tu np.
   d^.w, ale to by zaciemniło kod}
 end;
```

Dobre rady dotyczące stylu programowania

- Procedura udostępniana użytkownikowi powinna mieć tyle parametrów, ile jest to konieczne z punktu widzenia specyfikacji zadania. Wszelkie nadmiarowe parametry ułatwiające zaprogramowanie są wyrazem niechlujstwa.
- Złą praktyką programistyczną jest nieumieszczanie *istotnych* parametrów w nagłówku i korzystanie ze zmiennych globalnych.
- Rekurencja i iteracja niezbyt się lubią nawzajem. Najczęściej dobrze jest zdecydować się na jedną z nich.

Dobre rady dotyczące stylu programowania

- W procedurach rekurencyjnych dotyczących drzew zawsze sprawdzamy na początku, czy argument nie jest nilem.
- Przy przekazywaniu informacji od korzenia w dół używamy najczęściej porządku prefiksowego, a informację przekazujmy przez wartość
- Przy przekazywaniu informacji z dołu drzewa do góry używamy najczęściej porządku postfiksowego i przekazujemy ją albo przez parametry wołane przez zmienną albo za pomocą wywołania funkcji.

Podsumuj w obiegu postfiksowym z funkcją

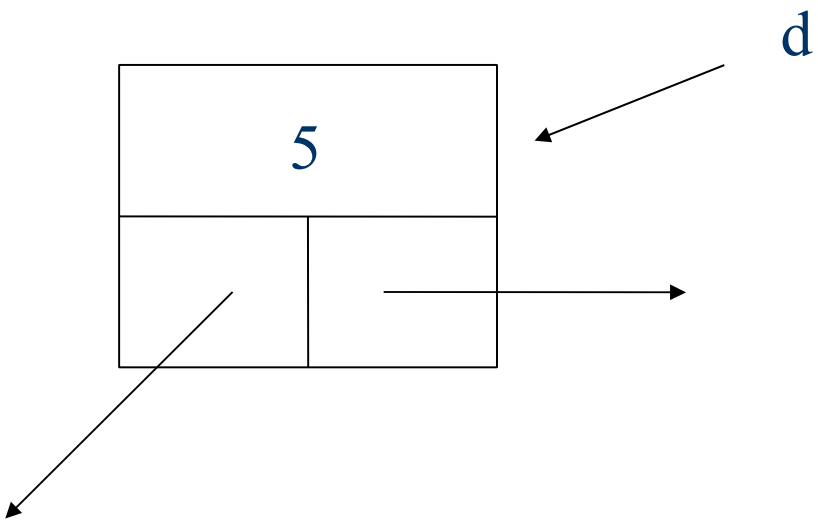
```
procedure Podsumuj(d:drzewo);
var dowol:Integer;
  function SumaPotomków(d:drzewo):Integer;
   begin
   if d=nil then SumaPotomków:=0 { koniecznie!}
   else
    begin
     d^.w := SumaPotomków(d^.lsyn)+SumaPotomków(d^.psyn)
             +d^.w;
     SumaPotomków:=d^.w
    end  {else};
    end; {DodajDół}
begin {Podsumuj}
 dowol:=SumaPotomków(d) {lub d^.w:=SumaPotomków(d)}
end;
```

Drzewa ogólne

- Jeśli drzewoma stopień większy niz dwa, to możemy je implementować w następujący sposób: w kazdym węźle przechowujemy poza wartością w:typ dwa wskaźniki: do pierwszego syna z lewej oraz do listy braci.

Drzewa ogólne

```
type drzewoOg = ^węzeł;
     węzeł = record
               w : typ;
      lsyn,brat : drzewoOg
     end;
var
```



Drzewa ogólne

- Jeśli drzewo ma stopień większy niz dwa, to możemy je implementować w następujący sposób: w kazdym węźle przechowujemy poza wartością w:typ dwa wskaźniki: do pierwszego syna z lewej oraz do listy braci.

Drzewa ogólne

- Widzimy, że różnica tak naprawdę polega na innej interpretacji takich samych danych.
- W rzeczywistości ta interpretacja oznacza, że w korzeniu pole `brat` jest zawsze puste.
- ... chyba że nie jest puste i wtedy w naturalny sposób mamy reprezentację lasu – ważnego uogólnienia drzew (drzewo wtedy, to las z jednym elementem).
- Procedury obiegu drzew ogólnych są analogiczne do takich procedur dla drzew binarnych

Drzewa ogólne - obiegi

- Jeśli chodzi o obiegi prefiksowy i postfiksowy, to odpowiedniość jest jednoznaczna
- Obiegów infiksowych można sobie wyobrazić więcej – tyle ile jest wynosi stopień drzewa minus 1. W zasadzie nie używa sie obiegów infiksowych w przypadku drzew ogólnych

Liczymy liczbę liści w drzewie ogólnym

```
function LiczbaLiści(d: drzewoOg): integer;
var lewo, prawo: integer;
begin
   if d = nil then LiczbaLiści:=0
   else if d^.lsyn=nil then
                        LiczbaLiści:=1
      else LiczbaLisci := LiczbaLiści(d^.lsyn)
                          + Liczbaliści(d^.brat);
      end
end;
```

Liczymy liczbę liści w drzewie binarnym

```
function LiczbaLiści(d: drzewo  ): integer;
var lewo, prawo: integer;
begin
   if d = nil then LiczbaLiści:=0
   else if d^.lsyn=d^.psyn then
                        LiczbaLiści:=1
      else LiczbaLiści := LiczbaLiści(d^.lsyn)
                          + Liczbaliści(d^.psyn);
      end
end;
```

Przykład – liczenie wysokości drzewa ogólnego

```
function h(d:drzewoOg):Integer;
begin
 if d=nil then h := -1
 else h:=max(h(d^.lsyn)+1,h(d^.brat))
end;
```

Przykład – liczenie wysokości drzewa bin. - przypomnienie

```
function h(d:drzewo  ):Integer;
begin
 if d=nil then h := -1
 else h:=max(h(d^.lsyn),h(d^.psyn))+1
end;
```