NOTATKI I UZUPEŁNIENIA DO WYKŁADU 5

27 listopada 2016

Drzewo z korzeniem

drzewo, w którym jeden z wierzchołków jest wyróżniony (korzeń drzewa); wierzchołki drzewa z korzeniem nazywane są węzłami

Przodek, potomek

Niech x będzie dowolnym węzełem drzewa T. Każdy węzeł y na ścieżce z korzenia r do x nazywamy przodkiem węzła x. Jeżeli węzeł y jest przodkiem x, to x jest potomkiem y

Poddrzewo

Poddrzewo o korzeniu x jest drzewem, utworzonym z potomków x, którego korzeniem jest x

Ojciec, syn

Jeżeli ostatnią krawędzią drzewa T na ścieżce od korzenia r do węzła x jest (y, x), to y jest poprzednikiem (ojcem) x, a x jest następnikiem (synem) y

Korzeń

Korzeń jest jedynym węzłem drzewa T, który nie ma ojca

Bracia

Jeżeli dwa węzły mają tego samego ojca, to nazywamy je braćmi

Typeset by $\mathcal{AMS}$-TEX

Węzeł wewnętrzny, zewnętrzny

Liść (węzł zewnętrzny) – węzeł, który nie ma następników (synów)

Węzł wewnętrzny – węzeł, który nie jest liściem

Stopień węzła

Stopień węzła x – liczba następników (synów) węzła x

Głębokość węzła

Głębokość węzła x w drzewie T – długość ścieżki od korzenia r do węzła x

Wysokość węzła

Wysokość węzła – liczba krawędzi na najdłuższej ścieżce w dół od tego węzła do liścia

Wysokość drzewa

Wysokość drzewa T – wysokość jego korzenia; jest równa największej głębokości węzła w drzewie T

Drzewa binarne

Drzewo binarne – struktura zdefiniowana na skończonym zbiorze węzłów, która

nie zawiera żadnych węzłów, lub składa się z trzech rozłącznych zbiorów węzłów:

– korzenia

– drzewa binarnego zwanego lewym poddrzewem

– drzewa binarnego zwanego prawym poddrzewem

Drzewo puste (drzewo zerowe) – drzewo binarne, które nie ma żadnych węzłów (oznaczane jako stała NIL)

Jeżeli w drzewie binarnym węzeł ma tylko jednego syna, to jego położenie ma istotne znaczenie, tj. czy jest on lewym synem czy też prawym synem

Regularne drzewo binarne

Drzewo, w którym każdy z węzłów jest bądź liściem, bądź ma stopień dwa; porządek synów węzła odpowiada ich położeniu

Pełne drzewo

Pełne drzewo rzędu k – drzewo, w którym wszystkie liście mają tę samą głębokość, a wszystkie węzły wewnętrzne mają stopień k

Własności pełnych drzew

– w pełnym drzewie rzędu k o wysokości h liczba liści wynosi k^h

– wysokość pełnego drzewa rzędu k o n liściach wynosi $\log_k n$

– liczba węzłów wewnętrznych pełnego drzewa rzędu k o wysokości h wynosi

$$\frac{k^h - 1}{k - 1}$$

– pełne drzewo binarne o wysokości h ma $2^h - 1$ węzłów wewnętrznych

Kopiec binarny

Tablicowa struktura danych, którą można rozpatrywać jako *prawie pełne* drzewo binarne (drzewo jest pełne na wszystkich poziomach z wyjątkiem być może najniższego, który jest wypełniony od strony lewej do pewnego miejsca)

Tablica A reprezentująca kopiec ma dwa atrybuty:

$A.length$ – liczba elementów tablicy

$A.heap\text{-}size$ – liczba elementów kopca przechowywanych w tablicy

Żaden element tablicy $A[1..A.length]$ występujący po $A[A.heap\text{-}size]$, gdzie $A.heap\text{-}size \leq A.length$ nie jest elementem kopca

Własności kopca

– korzeniem drzewa jest $A[1]$

– dla zadanego indeksu i węzła indeksy jego
  ojca, lewego syna i prawego syna wynoszą

PARENT(i)
  return $\lfloor i/2 \rfloor$

LEFT(i)
  return $2i$

RIGHT(i)
  return $2i + 1$

– w tablicowej reprezentacji kopca n-elementowego
  liście to węzły o indeksach

$$\left\lfloor \frac{n}{2} \right\rfloor + 1, \left\lfloor \frac{n}{2} \right\rfloor + 2, \ldots, n$$

– wysokość kopca mającego n-elementów jest rzędu $\lfloor \lg n \rfloor$

Własność kopca typu max

dla każdego węzła i, który nie jest korzeniem, zachodzi

$$A[PARENT(i)] \geq A[i]$$

tj. wartość w węźle jest nie większa niż wartość przechowywana w jego ojcu

Własność kopca typu min

dla każdego węzła i, który nie jest korzeniem, zachodzi

$$A[PARENT(i)] \leq A[i]$$

Przywracanie własności kopca typu max

Dane: Tablica A oraz indeks i w tej tablicy

Założenia: Drzewa binarne o korzeniach w LEFT(i) i RIGHT(i) są kopcami typu max, ale element $A[i]$ może być mniejszy od swoich synów (naruszenie własności kopca typu max)

Efekt: Poddrzewo zaczepione w węźle i staje się kopcem typu max

Algorytm

– wybieramy największy z elementów $A[i]$, $A[\mathrm{LEFT}(i)]$, $A[\mathrm{RIGHT}(i)]$; jego indeks zachowywany jest w zmiennej L

– jeżeli $A[i]$ jest największy, to koniec (poddrzewo zaczepione w i jest kopcem typu max)

– w przeciwnym razie jeden z synów jest największym elementem; zamieniamy $A[i] \leftrightarrow A[L]$ (węzeł i oraz jego synowie spełniają własność kopca typu max)

– poddrzewo zaczepione w L może nie spełniać własności kopca typu max; wywołaj rekurencyjnie procedurę przywracania włsności kopca na tym poddrzewie

MAX-HEAPIFY(A, i)
  $l =$ LEFT(i)
  $r =$ RIGHT(i)
  if $l \leq A.heap\text{-}size$ **and** $A[l] > A[i]$
    then $L = l$
    else $L = i$
  if $r \leq A.heap\text{-}size$ **and** $A[r] > A[L]$
    then $L = r$
  if $L \neq i$
    then $A[i] \leftrightarrow A[L]$
      MAX-HEAPIFY(A, L)

Czas działania MAX-HEAPIFY na węźle o wysokości h wynosi $O(h)$. W n-elementowym kopcu czas działania MAX-HEAPIFY wynosi $O(\lg n)$

Budowanie kopca

Obserwacja: W tablicowej reprezentacji kopca n-elementowego liście to węzły o indeksach

$$\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$$

Zatem każdy element podtablicy

$$A[\lfloor n/2 \rfloor + 1..n]$$

jest już 1-elementowym kopcem.

Algorytm: Przejdź przez pozostałe węzły drzewa i w każdym z nich wywołaj procedurę MAX-HEAPIFY

BUILD-MAX-HEAP(A)
   $A.heap\text{-}size = A.length$
   for $i = \lfloor A.length/2 \rfloor$ **downto** 1
      do MAX-HEAPIFY(A, i)

Ograniczenie górne na czas działania:

– koszt każdego wywołania MAX-HEAPIFY wynosi $O(\lg n)$

– takich wywołań jest jest $O(n)$

Zatem czas działania BUILD-MAX-HEAP wynosi

$$O(n \lg n)$$

Sortowanie przez kopcowanie

Algorytm:

– stosując procedurę BUILD-MAX-HEAP zbuduj kopiec typu max w tablicy wejściowej $A[1..n]$, gdzie $n = A.length$

– $A[1] \leftrightarrow A[n]$

– przekształć tablicę $A[1..n-1]$ w kopiec typu max; nowy korzeń może naruszać własność kopca typu max - MAX-HEAPIFY($A, 1$)

HEAPSORT(A)
   BUILD-MAX-HEAP(A)
   for $i = A.length$ **downto** 2
      do $A[1] \leftrightarrow A[i]$
      $A.heap\text{-}size = A.heap\text{-}size - 1$
      MAX-HEAPIFY($A, 1$)

Czas działania wynosi $O(n \lg n)$

BST

B_inary_ **S**_earch_ **T**_ree_

Drzewo wyszukiwań binarnych ma strukturę drzewa binarnego. Może być użyte zarówno jako słownik jak i kolejka priorytetowa

Podstawowe operacje na drzewach BST wymagają czasu proporcjonalnego do wysokości drzewa $\Theta(\lg n)$; w przypadku pesymistycznym dla pełnego drzewa binarnego na n węzłach $\Theta(n)$, gdy drzewo składa się z gałęzi o długości n

BST można zrealizować za pomocą struktury danych z dowiązaniami, w której każdy węzeł jest obiektem. Każdy węzeł w drzewie BST ma atrybut *key* oraz atrybuty służące do zapamiętywania wskaźników do *ojca* oraz *lewego* i *prawego syna*

$x.p = NIL$ – x jest korzeniem drzewa T

$x.left = NIL$ – węzeł x nie ma lewego syna

$x.right = NIL$ – węzeł x nie ma prawego syna

$T.root$ – atrybut zawierający wskaźnik do korzenia drzewa

Klucze są przechowywane w drzewie BST w taki sposób, aby spełniona była własność drzewa BST

Niech x będzie węzłem drzewa BST. Jeśli y jest węzłem znajdującym się w lewym poddrzewie węzła x, to

$$y.key \leq x.key$$

Jeśli y jest węzłem znajdującym się w prawym poddrzewie węzła x, to

$$y.key \geq x.key$$

Wyszukiwanie

Wyszukujemy w drzewie BST węzeł, który zawiera dany klucz

Dane: wskaźnik do korzenia oraz klucz k

Wynik: wskaźnik do węzła zawierającego klucz k lub NIL, gdy taki węzeł nie istnieje

Algorytm:

– rozpocznij wyszukiwanie w korzeniu i schodź po ścieżce w dół drzewa

– dla każdego węzła x porównaj klucz k z wartością $x.key$

– jeżeli wartości są równe, to wyszukiwanie zakończyło się sukcesem, jeżeli $x =$ NIL, to węzeł o takim kluczu nie istnieje

– jeżeli $k < x.key$, to przeszukuj lewe poddrzewo węzła x, jeżeli natomiast $k > x.key$, to przeszukuj prawe poddrzewo

TREE-SEARCH(x, k)
   if x ==NIL **or** $x.key == k$
      then return x
   if $k < x.key$
      then TREE-SEARCH($x.left, k$)
      else TREE-SEARCH($x.right, k$)

Czas działania procedury

$$\text{TREE-SEARCH}(T.root, k)$$

wynosi $O(h)$, gdzie h jest wysokością drzewa T

ITERATIVE-TREE-SEARCH(x, k)
   while $x \neq$NIL **and** $k \neq x.key$
         do if $k < x.key$
               then $x = x.left$
               else $x = x.right$
   return x

Minimum i maksimum

Procedura TREE-MINIMUM(x) wyznacza wskaźnik do węzła o najmniejszym kluczu w poddrzewie o korzeniu w węźle x

Procedura TREE-MAXIMUM(x) wyznacza wskaźnik do maksymalnego elementu poddrzewa o korzeniu w węźle x

TREE-MINIMUM(x)
  while $x.left \neq$NIL
    do $x = x.left$
  return x

TREE-MAXIMUM(x)
  while $x.right \neq$NIL
    do $x = x.right$
  return x

Przechodzenie drzewa metodą inorder

Klucz korzenia poddrzewa zostaje ”wypisany” **między** wartościami z jego lewego poddrzewa a wartościami z jego prawego poddrzewa

INORDRE-TREE-WALK(x)
  if $x \neq$NIL
    then INORDER-TREE-WALK($x.left$)
      wypisz $x.key$
      INORDRE-TREE-WALK($x.right$)

Własność: Jeśli x jest korzeniem poddrzewa o n węzłach, to wywołanie

INORDRE-TREE-WALK(x)

odbywa się w czasie $\Theta(n)$

Wywołanie

$$\text{INORDER-TREE-WALK}(T.root)$$

powoduje wypisanie wszystkich elementów drzewa BST

Przechodzenie drzewa metodą **preorder**: klucz korzenia "wypisywany" jest **przed** wypisaniem wartości znajdujących się w obu poddrzewach

Przechodzenie drzewa metodą **postorder**: klucz korzenia "wypisywany" jest **po** wypisaniu wartości znajdujących się w obu poddrzewach

Następnik

– procedura TREE-SUCCESSOR wyznacza następnika danego węzła w drzewie BST, tj. następny węzeł odwiedzany w czasie przechodzenia drzewa w porządku inorder

– jeżeli wszystkie klucze są **różne**, to następnikiem węzła x jest węzeł o najmniejszym kluczu większym niż $x.key$

– jeżeli x ma największy klucz w drzewie, to zostaje wyznaczona wartość NIL

Algorytm:

– jeżeli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest najbardziej na lewo położony węzeł w prawym poddrzewie (węzeł o najmniejszym kluczu w tym poddrzewie); zastosuj procedurę

$$\text{TREE-MINIMUM}(x.right)$$

– jeżeli węzeł x nie ma prawego poddrzewa, to wykorzystujemy własność drzewa BST

Własność: Niech T będzie drzewem BST, w którym wszystkie klucze są różne. Jeżeli prawe poddrzewo węzła x z T jest puste i y jest jego następnikiem, to y jest najniższym przodkiem x, którego lewy syn jest także przodkiem x

Algorytm: idź w górę drzewa aż do napotkania węzła, który jest lewym synem swego ojca; ten ojciec jest następnikiem węzła x

TREE-SUCCESSOR(x)
  if $x.right \neq$NIL
    then return TREE-MINIMUM($x.right$)
  $y = x.p$
  while $y \neq$NIL **and** $x == y.right$
      do $x = y$
        $y = y.p$
  return y

Czas działania procedury TREE-SUCCESSOR na drzewie o wysokości h wynosi $O(h)$, ponieważ przechodzimy albo po ścieżce w górę drzewa, albo po ścieżce w dół drzewa

Wstawianie

Nową wartość v wstawiamy do drzewa T za pomocą procedury TREE-INSERT. Do procedury przekazujemy jako argument węzeł z, w którym $z.key = v$, $z.left =$NIL i $z.right =$NIL

Algorytm:

– rozpocznij od korzenia przechodząc po ścieżce w dół drzewa; wskaźnik x przebiega po ścieżce, a zmienna y zawiera wskazanie na ojca x

– przesuwaj się w dół aż x przyjmie wartość NIL

– wstaw z do drzewa, przypisując właściwe wartości odpowiednim wskaźnikom

TREE-INSERT(T, z)
  $y =$ NIL
  $x = T.root$
  while $x \neq$ NIL
    do $y = x$
      if $z.key < x.key$
        then $x = x.left$
        else $x = x.right$
  $z.p = y$
  if $y ==$ NIL
   then $T.root = z$
   else if $z.key < y.key$
      then $y.left = z$
      else $y.right = z$

Konstruowanie drzewa BST

– jako korzeń drzewa weź pierwszy element listy

– **wstaw** kolejno każdy element listy jako nowy liść (procedura TREE-INSERT)

Własność: Pesymistyczny czas działania algorytmu konstruowania drzewa BST z dowolnej listy n elementów wynosi $\Omega(n \lg n)$