Dynamiczne struktury danych

Wstęp do Informatyki i Programowania

Maciek Gębala

5 grudnia 2024

Notatki

Słownik

Struktura danych umożliwiająca w miarę szybkie wykonanie operacji wstawienia elementu (bez powtórzeń), usunięcia elementu i sprawdzenie czy element jest w strukturze.

Notatki

Uporządkowane drzewo binarne

Drzewo binarne umożliwia szybkie wyszukiwanie, wstawianie i usuwanie elementów.

Struktura drzewa binarnego pozwala pomijać duże fragmenty drzewa podczas jego przeszukiwania.

Uporządkowane drzewo binarne składa się z węzłów przechowujących dane (`elem`) oraz wskaźniki na lewe (`left`) i prawe (`right`) poddrzewo (każde z poddrzew jest uporządkowanym drzewem binarnym).

Elementy są uporządkowane w ten sposób, że w lewym poddrzewie wskazanym z węzła o elemencie `elem` znajdują się węzły o elementach mniejszych od `elem` a w prawym większe od `elem`.

Notatki

Uporządkowane drzewo binarne

Jeśli drzewo jest zrównoważone (zawsze oba poddrzewa są podobnej wielkości) to operacje na takim drzewie można wykonywać w czasie logarytmicznym.

Jednak równoważenie drzew jest trudne i będzie dopiero na czwartym semestrze.

Na wykładzie zobaczymy implementację drzew niezrównoważonych z użyciem rekurencji.

Notatki

Implementacja w Adzie - treelib.ads

```
1  with Ada.Unchecked_Deallocation;
2  package TreeLib is
3     type Tree is private;
4     function isElement (t : Tree; e : Integer) return Boolean;
5     procedure Insert (t : in out Tree; e : Integer);
6     procedure Delete (t : in out Tree; e : Integer);
7     procedure Destroy (t : in out Tree);
8     procedure Print (t : Tree);
9     function Size (t : Tree) return Integer;
10 private
11    type Node;
12    type NodePtr is access Node;
13    type Node is record
14       elem  : Integer := 0;
15       left  : NodePtr := null;
16       right : NodePtr := null;
17    end record;
18    type Tree is record
19       root : NodePtr := null;
20       size : Integer := 0;
21    end record;
22    procedure Free is
23       new Standard.Ada.Unchecked_Deallocation (Node, NodePtr);
24 end TreeLib;
```


Notatki

Implementacja w Adzie - treelib.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2
3  package body TreeLib is
4     function isElement (t : Tree; e : Integer) return Boolean is
5        function isElem (n : NodePtr) return Boolean is
6        begin
7           if n = null then
8              return False;
9           elsif n.elem = e then
10             return True;
11          elsif e < n.elem then
12             return isElem (n.left);
13          else
14             return isElem (n.right);
15          end if;
16       end isElem;
17    begin
18       return isElem (t.root);
19    end isElement;
```


Notatki

Implementacja w Adzie - treelib.adb

```
21    procedure Insert (t : in out Tree; e : Integer) is
22       procedure ins (n : in out NodePtr) is
23       begin
24          if n = null then
25             n := new Node;
26             n.elem := e;
27             t.size := t.size + 1;
28          elsif e < n.elem then
29             ins (n.left);
30          elsif e > n.elem then
31             ins (n.right);
32          else    --  e = n.elem
33             null;
34          end if;
35       end ins;
36    begin
37       ins (t.root);
38    end Insert;
```


Notatki

Implementacja w Adzie - treelib.adb

```
40    procedure Delete (t : in out Tree; e : Integer) is
41       function extractMax (n : in out NodePtr) return NodePtr is
42          m : NodePtr;
43       begin
44          if n.right /= null then
45             return extractMax (n.right);
46          else
47             m := n;
48             n := m.left;
49             m.left := null;
50             return m;
51          end if;
52       end extractMax;
```


Notatki

Implementacja w Adzie - treelib.adb

```
53       procedure del (n : in out NodePtr) is
54          m : NodePtr;
55       begin
56          if n /= null then
57             if e < n.elem then
58                del (n.left);
59             elsif e > n.elem then
60                del (n.right);
61             else   --  e = n.elem
62                m := n;
63                if m.left = null then
64                   n := m.right;
65                elsif m.right = null then
66                   n := m.left;
67                else
68                   n := extractMax (m.left);
69                   n.left := m.left;
70                   n.right := m.right;
71                end if;
72                Free (m);
73                t.size := t.size - 1;
74             end if;
75          end if;
76       end del;
```


Notatki

Implementacja w Adzie - treelib.adb

```
77    begin
78       del (t.root);
79    end Delete;
80
81    procedure Destroy (t : in out Tree) is
82       procedure des (n : in out NodePtr) is
83       begin
84          if n /= null then
85             des (n.left);
86             des (n.right);
87             Free (n);
88          end if;
89       end des;
90    begin
91       des (t.root);
92       t.size := 0;
93    end Destroy;
```


Notatki

Implementacja w Adzie - treelib.adb

```
95    procedure Print (t : Tree) is
96       procedure prnt (n : NodePtr) is
97       begin
98          if n /= null then
99             Put ("[");
100            prnt (n.left);
101            Put (n.elem'Image & "␣");
102            prnt (n.right);
103            Put ("]");
104         else
105            Put ("^");
106         end if;
107      end prnt;
108   begin
109      prnt (t.root);
110      New_Line;
111   end Print;
112
113   function Size (t : Tree) return Integer is
114   begin
115      return t.size;
116   end Size;
117end TreeLib;
```


Notatki

Implementacja w Adzie - treetest.adb

```
1  with Ada.Text_IO; use Ada.Text_IO;
2  with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
3  with Ada.Strings.Unbounded; use  Ada.Strings.Unbounded;
4  with Ada.Strings.Unbounded.Text_IO; use Ada.Strings.Unbounded.Text_IO;
5
6  with TreeLib; use TreeLib;
7
8  procedure TreeTest is
9     t : Tree;
10    r : Integer;
11    b : Boolean;
12    command : Unbounded_String;
13    continue : Boolean := True;
14 begin
15    while continue loop
16       Put ("Command:␣");
17       Get_Line (command);
18       if command = "isElement" then
19          Put ("Value:␣");
20          Get (r);
21          Skip_Line;
22          b := isElement (t, r);
23          Put_Line ("Result:␣" & b'Image);
```


Notatki

Implementacja w Adzie - `treetest.adb`

```
24       elsif command = "Insert" then
25          Put ("Value: ");
26          Get (r);
27          Skip_Line;
28          Insert (t, r);
29          Put_Line ("Result: OK");
30       elsif command = "Delete" then
31          Put ("Value: ");
32          Get (r);
33          Skip_Line;
34          Delete (t, r);
35          Put_Line ("Result: OK");
36       elsif command = "Destroy" then
37          Destroy (t);
38          Put_Line ("Result: OK");
39       elsif command = "Print" then
40          Put ("Result: ");
41          Print (t);
42       elsif command = "Size" then
43          r := Size (t);
44          Put_Line ("Result: " & r'Image);
```


Notatki

Implementacja w Adzie - `treetest.adb`

```
45       elsif command = "Exit" then
46          continue := False;
47       else
48          Put_Line ("Unknown command!");
49       end if;
50    end loop;
51
52    if Size (t) > 0 then
53       Destroy (t);
54    end if;
55 end TreeTest;
```


Notatki

Tablice dynamiczne

Chcemy mieć tablicę zmiennego rozmiaru z operacjami `append` dodającymi nowy element na końcu (zwiększający rozmiar tablicy o 1) i `delete` usuwający ostatni element tablicy (zmniejszający rozmiar tablicy o 1).

Pomysł polega na tworzeniu co jakiś czas nowej tablicy i przepisywaniu do niej elementów starej.

Pytanie kiedy i jak często to robić?

Notatki

Tablice dynamiczne

Tablicę powiększamy dwukrotnie jeśli kończy się w niej miejsce.

Można pokazać, że przy takim działaniu każdy element tablicy jest przepisywany średnio nie więcej niż dwa razy (uzasadnienie na tablicy).

Tablicę zmniejszamy dwukrotnie jeśli liczba elementów jest nie większa niż 1/4 wielkości tablicy.

Można pokazać, że po dodaniu zmniejszania każdy element tablicy jest przepisywany średnio nie więcej niż trzy razy (uzasadnienie na tablicy).

Pełny dowód jest trochę bardziej skomplikowany i pojawia się na drugim stopniu studiów.

Notatki