Programming – Block C

http://www.win.tue.nl/~wstomv/2ip05/

Lecture 15

Tom Verhoeff

Technische Universiteit Eindhoven
Faculteit Wiskunde en Informatica
Software Engineering & Technology

Feedback to T.Verhoeff@TUE.NL

Today's Topics

- Branching dynamic data structures with pointers:
  - Trees
  - Binary Trees (BTs)
  - Binary Search Trees (BSTs)
- More recursion

Linear Structures: Linked Lists

- Storage overhead: 1 or 2 pointers per cell

  Single-linked versus double-linked (bi-directional)

  Double linking simplifies insertion and deletion of nodes.

- Access/modification time: worst-case proportional to list length

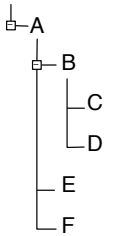
  Linear in the actual size: $\mathcal{O}(N)$

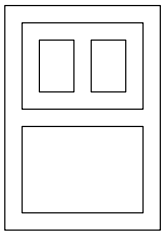
How to improve the worst-case time for operations?

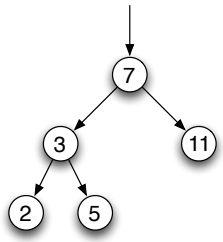
Branching Structures: (Binary (Search)) Trees

In computing, trees are everywhere

- Syntax trees
- Directory trees
- Widget trees
- Search trees
- ...

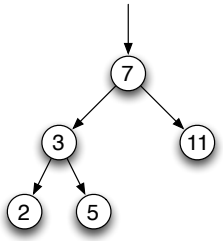






Tree Concepts and Terminology

- Graph: node, edge (connected, no cycles)
- Parent, child, subtree, order on children
- Root, leaf, front, root path
- Arity (fixed, variable)
- Height, balance
- Traversal: pre-order, in-order, post-order



Binary Trees

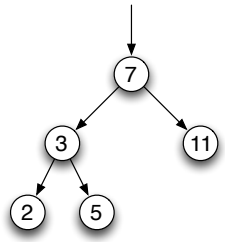
The set $BT(S)$ of Binary Trees over a set S is inductively defined by

(Basis) $\varepsilon \in BT(S)$ (the empty tree)

(Step) if $L \in BT(S), a \in S, R \in BT(S)$, then $\langle L, a, R\rangle \in BT(S)$

Some examples in $BT(\mathbb{N})$:

- ε
- $\langle \varepsilon, 2, \varepsilon\rangle$
- $\langle\langle \varepsilon, 2, \varepsilon\rangle, 3, \langle \varepsilon, 5, \varepsilon\rangle\rangle$
- $\langle\langle\langle \varepsilon, 2, \varepsilon\rangle, 3, \langle \varepsilon, 5, \varepsilon\rangle\rangle, 7, \langle \varepsilon, 11, \varepsilon\rangle\rangle$



Recursive Functions on Binary Trees

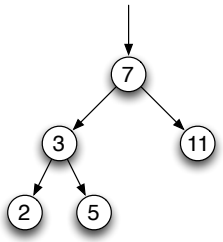
The height $h.t$ of a binary tree t is recursively defined by

(Basis) $h.\varepsilon = 0$

(Step) $h.\langle L, a, R\rangle = 1 + \max(h.L, h.R)$

Examples:

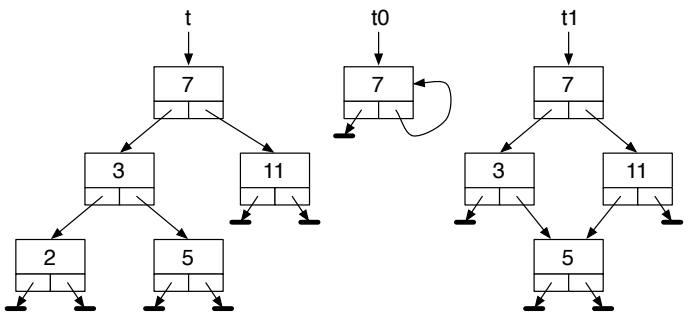
- $h.\langle \varepsilon, 11, \varepsilon\rangle = 1$
- $h.\langle \langle \varepsilon, 2, \varepsilon\rangle, 3, \langle \varepsilon, 5, \varepsilon\rangle\rangle = 2$
- $h.\langle\langle\langle \varepsilon, 2, \varepsilon\rangle, 3, \langle \varepsilon, 5, \varepsilon\rangle\rangle, 7, \langle \varepsilon, 11, \varepsilon\rangle\rangle = 3$



Binary Tree Representation in Pascal

```
1  type
2    TData = ...; // type for content of the binary trees
3
4    PNode = ^TNode;
5
6    TNode = // node of binary tree over TData
7    record
8      FData: TData; // value stored in this node
9      FLeft: PNode; // left subtree
10     FRight: PNode; // right subtree
11   end;
12
13   TBTData = PNode; // Binary Trees over TData
14     // data invariants
15     //   I0: all root paths via FLeft/FRight terminate in nil
16     //   I1: if two root paths both reach u then u = nil
17     // Abstraction function [[t]] for t: TBTData equals
18     // empty tree if t=nil else <[[t^.FLeft]], t^.FData, [[t^.FRight]]>
```

Binary Tree Type in Pascal: Examples



t is OK t0 violates I0 t1 violates I1

Binary Trees in Pascal: Observations

- Storage overhead: 2 pointers per node
- # **nil** pointers = # nodes + 1
- Maximum # nodes in tree of height $N = 2^N - 1$
- Minimum # nodes in tree of height $N = N$
- $\log_2(1 + \#\text{nodes}) \leq$ height $\leq \#\text{nodes}$
- In a balanced binary tree, the root path length to arbitrary node is *potentially* logarithmic in the total number of nodes.

Creating a Binary Tree in Pascal

The empty tree ε is represented by **nil** .

A nonempty tree is best created through an auxiliary function:

```
1  function MakeBTData(a: TData; L, R: TBTData): TBTData;
2    // pre: L and R do not share any nodes
3    // ret: <L, a, R>
4    begin
5      New(Result);
6      with Result^ do begin
7        FData := a;
8        FLeft := L;
9        FRight := R;
10     end; { with }
11     // Result satisfies I0 and I1
12   end;
```


Creating a Binary Tree in Pascal: Example

```
1  var
2    t: TBTData; // a binary tree over TData
3
4  begin
5    t := MakeBTData(7,
6           MakeBTData(3,
7             MakeBTData(2, nil, nil),
8             MakeBTData(5, nil, nil)),
9           MakeBTData(11, nil, nil));
```

Height of Binary Tree in Pascal

```
1  uses
2    Math; // for Max
3
4  function Height(t: TBTData): Integer;
5    // pre: true
6    // ret: h.t
7    begin
8      if t = nil then begin
9        Result := 0;
10     end
11     else { t <> nil } begin
12       Result := 1 + Max ( Height(t^.FLeft), Height(t^.FRight) );
13       // termination guaranteed because of I0
14     end;
15   end;
```


Binary Tree Traversals

Standard orders for "visiting" the nodes of a binary tree:

- Pre-order: first the node itself, then the subtrees
- In-order: first left subtree, next the node, then right subtree
- Post-order: first the subtrees, then the node

Coding techniques: recursively or with repetition and self-made stack

Pre-order Traversal in Pascal (recursive)

```
1  procedure PreOrder(t: TBTData);
2    // pre: true
3    // post: all nodes of t have been processed in pre-order
4    begin
5      if t = nil then begin
6        // skip
7      end
8      else { t <> nil } begin
9        with t^ do begin
10         ... process FData ...;
11         PreOrder(FLeft);
12         PreOrder(FRight);
13       end; { with }
14     end;
15   end;
```


In-order Traversal in Pascal (recursive)

Slightly "optimized" code

```
1  procedure InOrder(t: TBTData);
2    // pre: true
3    // post: all nodes of t have been processed in in-order
4    begin
5      if t <> nil then begin
6        with t^ do begin
7          InOrder(FLeft);
8          ... process FData ...;
9          InOrder(FRight);
10       end; { with }
11     end; { if }
12   end;
```

Parameterized Post-order Traversal in Pascal (recursive)

```
1  type TDataAction = procedure ( const AData: TData );
2
3  procedure PostOrder(t: TBTData; ADataAction: TDataAction);
4    // pre: true
5    // post: all nodes of t have been processed in post-order
6    //       by applying ADataAction
7    begin
8      if t <> nil then begin
9        with t^ do begin
10         PostOrder(FLeft, ADataAction);
11         PostOrder(FRight, ADataAction);
12         ADataAction(FData);
13       end; { with }
14     end; { if }
15   end;
```


Apply Parameterized Post-order Traversal in Pascal

```
1  procedure WriteData(const AData: TData);
2    begin
3      Writeln(... AData ...)
4    end;
5
6  var
7    t: TBTData; // a binary tree
8
9  // WriteData has type TDataAction
10 PostOrder(t, WriteData);
```


Apply Post-order Traversal in Pascal (2)

```
1  var
2    count: Integer; // global counter for CountData
3
4  procedure CountData(const AData: TData {; globvar count});
5    // effect: count := count + 1
6    begin
7      count := count + 1
8    end;
9
10 var
11   t: TBTData; // a binary tree over TData
12
13 count := 0;
14 PostOrder(t, CountData);
15 // count = # nodes in t
```


Binary Search Trees

Time to find data in a binary tree is still worst-case linear in the size

Can be improved by keeping data sorted

The list $\ell.t$ of a binary tree t is inductively defined by

(Basis) $\ell.\varepsilon = \varepsilon$ (the empty list)

(Step) $\ell.\langle L, a, R\rangle = \ell.L +\!\!+ [a] +\!\!+ \ell.R$ (cf. in-order traversal)

A Binary Search Tree is a binary tree t with additional invariant:

- $\ell.t$ is strictly increasing (w.r.t. a suitable order on S)

Binary Search Trees in Pascal

```
type
  TBSTData = TBTData; // Binary Search Tree over TData
    // data invariant
    //   I2: in-order list is strictly increasing
    // for every subtree <L, a, R>: data in L < a < data in R
  TCompareData = ( ls, eq, gt ); // less, equal, greater

function CompareData(X, Y: TData): TCompareData;
begin
  if ''X < Y'' then Result := ls
  else if ''X = Y'' then Result := eq
  else { ''X > Y'' } Result := gt
end;
```


Find Data in Binary Search Tree (Recursive)

```
function FindRec(const X: TData; t: TBSTData): PNode;
  // pre: true
  // ret: r with t -*-> r /\ r^.FData = X if X in t else nil
  begin
    if t = nil then Result := nil
    else { t <> nil } begin
      with t^ do begin
        case CompareData(X, FData) of
          ls: Result := FindRec(X, FLeft);
          eq: Result := t;
          gt: Result := FindRec(X, FRight);
        end; { case }
      end; { with }
    end; { else }
  end;
```


Find Data in Binary Search Tree (Iterative)

```
function FindIter(const X: TData; t: TBSTData): PNode;
  // pre: t = T
  // ret: r with t -*-> r /\ r^.FData = X if X in T else nil
  begin
    Result := nil;
    // inv: T -*-> t -*-> Result  /\  X in T == X in t /\
    //   (Result <> nil ==> Result^.FData = X)
    while t <> Result do { t <> nil } with t^ begin
      case CompareData(X, FData) of
        ls: t :=  FLeft;
        eq: Result := t;
        gt: t := FRight;
      end; { case }
    end; { while/with }
  end;
```


Insert Data into Binary Search Tree (Recursive)

```
procedure Insert(const X: TData; var t: TBSTData);
  // pre: true
  // post: t = initial t extended with X
  begin
    if t = nil then t := MakeBTData(X, nil, nil)
    else { t <> nil } begin
      with t^ do begin
        case CompareData(X, FData) of
          ls: Insert(X, FLeft);
          eq: { skip }; // already present, do not duplicate
          gt: Insert(X, FRight);
        end; { case }
      end; { with }
    end; { else }
  end;
```

Loose Ends

- Minimum, Maximum
- Iterative traversals, iterative insert
- Delete item, dispose entire tree
- Merge (union)
- Maintaining balance
- Sorting algorithms

What Lies Ahead

Course	Code	Year	Blocks
Programming Methods	2IP15	1	D-E-F
Data Structures	2IL05	1	D-E-F
Software Specification	2IW05	2	A-B-C
Algorithmics	2IL15	2	D-E-F
Distributed Algorithms	2IL25	2	D-E-F
Functional Programming	2IA05	3	D-E
Software Engineering	2IP25	3	D
Software Engineering Project	2IP35	3	D-E-F

Verhaal met moraal

Jorge Luis Borges

Argentijns schrijver en dichter

1899 – 1986

LA ROSA DE PARACELSO

—

De roos van Paracelsus