ADT_BST **Binary Search Tree specificatie** datastructuren college 4

Elementen:

Een pointer verwijst naar een instantiatie van een binaire boom;

De boom is opgebouwd uit knopen;

Elke knoop heeft een veld voor data (b.v. key + datarecord) en twee pointers naar gelijksoortige knopen

Structuur:

De binaire boom pointer verwijst naar een root knoop (naar nil indien het een lege boom betreft);

De pointers in de knopen verwijzen naar een geordend stel verschillende knopen, de linker en rechterknoop.

De binaire boom heeft een inherente hierarchie: elke knoop behalve de root heeft een unieke ouder; elke knoop heeft 0, 1 of 2 kinderen; een kind is linker of rechter knoop.

Een binaire knoop in de binaire zoek boom wordt op InOrder volgorde doorlopen.

Een key waarde komt hooguit 1 maal voor.

Domein:

Aantal knopen in een binaire zoek boom is eindig: lege boom heeft 0 knopen, boom met maximum aantal knopen is vol. Een niet lege boom heeft een CurrentKnoop met een CurrentElement (b.v. de key van de opgeslagen data in de knoop).

Type

```
BinarySearchTree;
Relatie      =   (LinkerKind,RechterKind, Root, Ouder);
Status       =   record
                     Root           :   KnoopPointer;
                     CurrentPointer :   KnoopPointer;
                     Rel            :   Relatie
                     end;
KnoopPointer     :   ^Knoop;
Knoop        =   record
                     KeyValue       :   KeyDataType;
                     LeftChild      :   KnoopPointer;
                     RightChild     :   KnoopPointer
                end;
```

Operaties:

Er zijn 14 operaties gedefinieerd

procedure **Create**(var BST:BinarySearchTree;var Created:Boolean);

postconditie: Created=Y als aanmaak BT gelukt, anders N

function **Empty**(var BST:BinarySearchTree):boolean;

postconditie: Empty=Y als BT leeg, anders N

function **Full**(var BST:BinarySearchTree):Boolean;

postconditie: Full=Y als BT vol, anders N

procedure **Delete**(var BST:BinarySearchTree);

postconditie: BT bestaat niet langer, geheugen vrijgegeven

prive procedure **DeleteSub**(var BST:BinarySearchTree);

preconditie: CurrentKnoop als root van subtree gegeven

postconditie: OuderKnoop van CurrentKnoop heeft Linker/Rechter Kind pointer op nil staan,

Current staat op Root; geheugen subtree vrijgegeven

Prive procedure **FreeKnots**(var BST:BinarySearchTree;Next:KnoopPointer)

Postconditie: geheugen van deze knoop en alle kinderen vrijgegeven

ADT_BST **BinarySearchTree** **specificatie** Datastructuren college 4 bladzijde 4

procedure **InsertKey**(var BST:BinarySearchTree;var Key:KeyDataType;Rel:Relatie;var Inserted:boolean);

preconditie: CurrentPointer wijst naar knoop in BST waar Key kan worden toegevoegd als rel.

postconditie: als Inserted=Y, Key toegevoegd aan BST en CurrentPointer wijst naar knoop met deze Key, anders deze Key bestond al of BST was vol

Procedure **FindKey**(var BST:BinarySearchTree;var key:KeyDataType;Found:Boolean);

Postconditie: Found=Y als key in boom voorkwam, interne CurrentPointer staat dan naar bijbehorende knoop; Found=N als key niet in boom BT voorkwam, CurrentPointer staat bij knoop waar key waarde kan worden toegevoegd aan de BST, relatie met CurrentPointer=[Root,Linker,Rechter]

Procedure **DeleteKey**(var BST:BinarySearchTree;key:KeyDataType;Deleted:boolean);

Postconditie: Deleted=Y als key in BT zat, N als key er al niet in zat; na afloop staat CurrentPointer op root.

Procedure **Doorloop**(var BST:BinarySearchTree);

Postconditie: elk element in de Binaire Zoek Boom wordt 1 maal bezocht en er wordt een door de gebruiker opgegeven functie mee uitgevoerd (b.v. printen naar een gesorteerde lijst).

Procedure **RetrieveKey**(var BST:BinarySearchTree;Key:KeyDataType);

Postconditie: waarde opgeslagen in CurrentKnoop wordt getoond.

Procedure **Update**(var BST:BinarySearchTree;Key:KeyDataType);

Preconditie: CurrentKnoop staat op Knoop positie waar wijziging key plaats moet vinden.

Postconditie: CurrentKnoop is gedelete, Key toegevoegd op plaats aangegeven door nieuwe CurrentKnoop waarde.

Prive Procedure **FindParent**(var BST:BinarySearchTree;Parent:);

Preconditie: ten opzichte van CurrentPointer, vindt OuderKnoop

Postconditie: CurrentPointer staat op die van gevraagde Ouder

Privé procedure **DeleteSub**(var BST:BinarySearchTree);

Preconditie: Subtree met CurrentPointer als root weghalen

Postconditie: in OuderKnoop van CurrentPointer is linker/rechter kindknoop naar nil gezet;

CurrentPointer staat op ouder van invoer knoop of op nu lege RootPointer

Type

```
Relatie          =   (LinkerKind,RechterKind, Root, Ouder);
KnoopPointer     =   ^Knoop;
Knoop            =   record
                         Links    :   KnoopPointer;
                         Key      :   KeyDataType;
                         Rechts   :   KnoopPointer
                     end;
BST              =   ^BinarySearchTree;
BinarySearchTree=    record
                         Root     :   KnoopPointer;
                         Current  :   KnoopPointer;
                         Rel      :   Relatie;
                     end;
```

```
procedure Create(var BST:BinarySearchPointer;Created:Boolean);
begin
        new(BST);
        BST^.Root:=nil;
        BST^.Current:=nil;
        BST^.Rel:=nil;
end;
procedure Delete(var BST:BinarySearchTree);
begin
        FreeKnots(BST,BST^.Root);
        Dispose(BST)
end;
function Empty(BST:BinarySearchTree):Boolean;
begin
        Empty:=(BST^.Root=nil)
end;
function Full(BST:BinarySearchTree):Boolean;
begin
        Full:=N
end;
```

```
procedure FindKey(var BST:BinarySearchTree;var Key:KeyDataType;Found:Boolean);
var Parent,Voorouder: KnoopPointer;
begin
        Found:= N;
        If not Empty(BST)
                then with BST^ do begin
                        Parent:=Root;
                        repeat
                                Voorouder:=Parent;
                                If (Key= Parent^.Key)
                                        Then begin
                                                Current:=Parent;
                                                Found:=Y;
                                        end
                                elseif (Key<Parent^.Key)
                                        then Parent:=Parent^.Links
                                        else Parent:=Parent^.Rechts
                        until ((Found) or (Parent:=nil))
                        if (not Found) then Current:=Voorouder;
                end
end;
```

```
procedure Insert(var BST:BinarySearchTree;var Key:KeyDataType;var Inserted:boolean);
var     Parent, ParentSaved: KnoopPointer;
        Found:Boolean;
begin
        Inserted:=N;
        With BST^ do begin
                ParentSaved:=Current;
                FindKey(BST,Key,Found);
                If (Found) then Current:=ParentSaved
                else begin
                        new(Parent);
                        Parent^.Links:=nil;
                        Parent^.Rechts:=nil;
                        Parent^.Key:=Key;
                        If (Empty(BST))
                        then    Root:=Parent;
                        else    if (Key<Current^.Key)
                                then    Current^.Links:=Parent
                                else    Current^.Rechts:=Parent;
                        Current:=Parent;
                        Inserted:=Y;
                end
        end
end;
```

```
procedure DeleteKey(var BST:BinarySearchTree;ZoekKey:KeyDataType;Deleted:boolean);
var Remove: KnoopPointer;
    Procedure SubDel(var Q:KnoopPointer);
    begin
        if (Q^.Rechts<>nil)
        then    SubDel(Q^.Rechts)
        else begin
                Remove^.Key:=Q^.Key;
                Remove:=Q;
                Q:=Q^.Links
        end
    end;
    procedure Del(ZoekKey:KeyDataType;var Parent:KnoopPointer;var Found:Boolean);
    begin
        if (Parent=nil) then Found:=N
        else with Parent^ do
                if (ZoekKey<Key)
                then Del(Zoekkey,Links,Found)
                else if (ZoekKey>Key)
                then Del(Zoekkey,Rechts,Found)
                else begin
                        Remove:=Parent;
                        If (Rechts=nil) then Parent:=Links;
                        else if (Links=nil) then Parent:=Rechts;
```

```
                                dispose(Remove);
                                Found:=Y
                        end
        end;
begin
        Del(ZoekKey,BST^.Root,Deleted);
        If  not (Empty(BST)) then BST^.Current:=BST^.Root;
end;
prive procedure FindParent(BST:BinarySearchTree;var Parent:KnoopPointer);
var     S:Stack;
        OK:Boolean;
begin
        StackCreate(S,OK);
        with BST^ do begin
                Parent:=Root;
                While ((Parent^.Links<>Current) and (Parent^.Rechts<>Current)) do begin
                        If (Parent^.Rechts<>nil) then Push(S,Parent^.Rechts);
                        If (Parent^.Links<>nil) then Push(S,Parent^.Links);
                        else Pop(S,Parent)
                end
        end;
        StackTerminate(S)
end;
```

```
procedure Update(var BST:BinarySearchTree;Key:KeyDataType);
var OK:Boolean;
begin
        Delete(BST,BST^.Current^.Key,OK);
        Insert(BST,Key,OK)
end;
Procedure ProcKey(Key:KeyDataType;Level:Integer);
begin
        Hier gebruiker gedefineerde bewerking op deze Key
end;
procedure Doorloop(var BST:BinarySearchTree);
        procedure InOrder(P:KnoopPointer;level:Integer);
        begin
                if (P<>nil)
                then begin
                        InOrder(P^.Links,Level+1);
                        Proc(P^.Key,level);
                        InOrder(P^.Rechts,Level+1)
                end
        end;
begin
        InOrder(BST^.Root,1)
end;
```

```
procedure InsertKey(var BST:BinarySearchTree;var Key:KeyDataType;Rel:Relatie;var Inserted:boolean);
var Child:KnoopPointer;
begin
        if (((Rel=LinkerKind) and (BST^.Current^.Links<>nil)) or
                ((Rel=RechterKind) and (BST^.Current^.Rechts<>nil)))
        then    Inserted:=N
        else begin
                new(Child);
                Child^.Links:=nil;
                Child^.Key:=Key;
                Child^.Rechts:=nil;
                Case Rel of
                        Root            :       BST^.Root:=Child;
                        LinkerKind      :       BST^.Current^.Links:=Child;
                        RechterKind     :       BST^.Current^.Rechts:=Child
                end;
                BST^.Current:=Child;
                Inserted:=Y;
        end
end;
```

```
prive procedure DeleteSub(var BST:BinarySearchTree);
var Start:KnoopPointer;
        OK:booelean;
begin
        with BST^ do begin
                if (Current=Root)
                then begin
                        FreeKnots(BST,Current);
                        Root:=nil;
                        Current:=nil;
                end
                else begin
                        Next:=Current;
                        FindParent(BST,Parent);
                        If (Current^.Links=Next) then Current^.Links:=nil
                        else Current^.Rechts:=nil;
                end;
                FreeKnots(BST,Start);
                Current:=Root;
        end
end;
```

```
prive procedure FreeKnots(var BST:BinarySearchTree;Next:KnoopPointer);
begin
        if (Next<>nil)
        then begin
                FreeKnots(BST,Next^.Links);
                FreeKnots(BST,Next^.Rechts);
                Dispose(Next)
        end
end;
```