Prüfungsvorbereitung

Sascha Riexinger

08.03.2007

Inhaltsverzeichnis

Listen

Was sollte man können?

Listen

Was sollte man können?

- Datenstruktur entwerfen.

Listen

Was sollte man können?

- Datenstruktur entwerfen.
- Neue Liste erstellen.

Listen

Was sollte man können?

- Datenstruktur entwerfen.
- Neue Liste erstellen.
- Element(e) vorne, hinten, sortiert in Liste einfügen.

Listen

Was sollte man können?

- Datenstruktur entwerfen.
- Neue Liste erstellen.
- Element(e) vorne, hinten, sortiert in Liste einfügen.
- Element(e) in Liste suchen.

Listen

Was sollte man können?

- Datenstruktur entwerfen.
- Neue Liste erstellen.
- Element(e) vorne, hinten, sortiert in Liste einfügen.
- Element(e) in Liste suchen.
- Element(e) aus Liste entfernen.

Listen - Beispiel - Bauer Gscheidle

Bauer Gscheidle ist es nach unzähligen und durchaus auch mühsamen Kreuzungsversuchen gelungen, das Traumtier eines jeden Landwirtes zu züchten - die eierlegende Woll-Milch-Sau. Um den Überblick über den Bestand und die Ergiebigkeit seiner neuen Tiere zu behalten, möchte er die Daten über seine Tiere gerne in seinem Computer halten. Und hat daher Ihre Firma LASB (landwirtschaftliche Anwendungs- und Systembeihilfe) gebeten ihm dafür ein entsprechendes Programm zu entwickeln.

Um für seine Säue einen aktuellen Wert zu ermitteln, möchte Bauer Gscheidle, dass die Milch-(l), Woll-(kg) und Eiermenge (Anzahl) der letzten 30 Tage (pro Sau) gespeichert wird. Ebenfalls möchte er monatlich das Gewicht (kg) einer jeden Sau aktualisieren können. Damit er die Säue auseinander halten kann, hat er ihnen Bändchen mit verschiedenen Farben an die Schwänzchen gebunden, diese Information sollte ebenfalls vom Programm zur Identifizierung der Sau gespeichert werden.

Listen - Beispiel - Bauer Gscheidle

Entwerfen Sie eine geeignete und möglichst einfache Datenstruktur zur Haltung der vorgegebenen Daten, dabei soll der Bestand aller Säue in einer Liste gehalten werden. Geben Sie die dazu nötige Datenstruktur ebenfalls an.

Listen - Beispiel - Bauer Gscheidle

Lösung

Listen - Beispiel - Bauer Gscheidle

Lösung

type Sau;

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band      : Unbounded_String;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band    : Unbounded_String;
        Milch   : TagesWerte;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band    : Unbounded_String;
        Milch   : TagesWerte;
        Wolle   : TagesWerte;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band    : Unbounded_String;
        Milch   : TagesWerte;
        Wolle   : TagesWerte;
        Eier    : TagesWerte;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band      : Unbounded_String;
        Milch     : TagesWerte;
        Wolle     : TagesWerte;
        Eier      : TagesWerte;
        Gewicht   : TagesWerte;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band     : Unbounded_String;
        Milch    : TagesWerte;
        Wolle    : TagesWerte;
        Eier     : TagesWerte;
        Gewicht  : TagesWerte;
        Index    : Integer range 1..30;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band     : Unbounded_String;
        Milch    : TagesWerte;
        Wolle    : TagesWerte;
        Eier     : TagesWerte;
        Gewicht  : TagesWerte;
        Index    : Integer range 1..30;
        Next     : Liste;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
type Sau;
type Liste is access Sau;
type TagesWerte is array (1 .. 30) of Float;
type Sau is record
        Band     : Unbounded_String;
        Milch    : TagesWerte;
        Wolle    : TagesWerte;
        Eier     : TagesWerte;
        Gewicht  : TagesWerte;
        Index    : Integer range 1..30;
        Next     : Liste;
end record;
```

Listen - Beispiel - Bauer Gscheidle

Um den Produktionswert einer Sau zu ermitteln, möchte Ihr Kunde, dass das Programm wie folgt vorgeht. Es bekommt vom Benutzer die aktuellen Marktpreise von Wolle, Eiern und Milch und ermittelt den durchschnittlichen Tageswert der Sau im Bezug auf die letzten 30 Tage. Schreiben Sie eine Funktion, die als Parameter die aktuellen Preise für Wolle (pro kg), Eier (pro Stück) und Milch (pro l) sowie einen Zeiger auf ein Datenobjekt der von Ihnen entworfenen Datenstruktur übergeben bekommt. Der Rückgabewert der von Ihnen zu schreibenden Funktion **SauProduktionsWert** soll der durchschnittliche Produktionswert der Sau innerhalb der letzten 30 Tage sein. (Programmieren sie auch die Durchschnittsberechnung aus.)

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
      SauWert := Durchschnitt (Sau.Milch) +
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
        SauWert := Durchschnitt (Sau.Milch) +
            Durchschnitt (Sau.Wolle) +
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
        SauWert := Durchschnitt (Sau.Milch) +
            Durchschnitt (Sau.Wolle) +
            Durchschnitt (Sau.Eier);
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
      SauWert := Durchschnitt (Sau.Milch) +
          Durchschnitt (Sau.Wolle) +
          Durchschnitt (Sau.Eier);
      return SauWert;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function SauProduktionsWert
                (PreisWolle:Float;
                 PreisEier:Float;
                 PreisMilch:Float;
                 Sau:Liste)
return Float is
                (1) ← Hierzu später ;)
                SauWert:Float;
begin
        SauWert := Durchschnitt (Sau.Milch) +
            Durchschnitt (Sau.Wolle) +
            Durchschnitt (Sau.Eier);
        return SauWert;
end;
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

function Durchschnitt

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

function Durchschnitt
(Tage:TagesWerte)

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
                Summe:Float:=0.0;
begin
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
                Summe:Float:=0.0;
begin
        for i in 1..30 loop
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
                Summe:Float:=0.0;
begin
        for i in 1..30 loop
            Summe := Summe + Tage(i);
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
                Summe:Float:=0.0;
begin
        for i in 1..30 loop
            Summe := Summe + Tage(i);
        end loop;
```

Listen - Beispiel - Bauer Gscheidle

Lösung - (1)

```
function Durchschnitt
                (Tage:TagesWerte)
return Float is
                Summe:Float:=0.0;
begin
        for i in 1..30 loop
            Summe := Summe + Tage(i);
        end loop;
        return Summe / 30.0;
end;
```

Listen - Beispiel - Bauer Gscheidle

Die Säue vermehren sich zur Freude des Bauern rasend schnell, innerhalb von 30 Tagen nimmt die Saupopulation um 10% zu. Der Bauer hat jedoch nur beschränkt Platz in seinen Stallungen und möchte daher, dass das Programm monatlich die schlechtesten 10% der Säue zum Verkauf auswählt, um Platz für den Nachwuchs zu schaffen. Die Gesamtheit der Säue des Bauern ist in einer einfach verketteten Liste gehalten. Schreiben sie eine Prozedur **WähleSauAus**, welche als Parameter die Liste mit allen (noch nicht ausgewählten) Säuen des Bauern und die Liste der bereits ausgewählten Säue übergeben bekommt und dann anhand des Produktivitätswertes die "schlechteste" Sau innerhalb der noch nicht ausgewählten Säue findet und diese ans Ende der Liste, der bereits ausgewählten Säue, anhängt.

Listen - Beispiel - Bauer Gscheidle

Lösung

procedure WähleSauAus

Listen - Beispiel - Bauer Gscheidle

Lösung

procedure WähleSauAus
(Unselektiert : **in out** Liste;

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
      Sau             : Liste := Unselektiert;
begin
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
      Sau             : Liste := Unselektiert;
begin
      if Sau = null then
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
      Sau             : Liste := Unselektiert;
begin
      if Sau = null then
          return;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
      Sau             : Liste := Unselektiert;
begin
      if Sau = null then
          return;
      end if;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
procedure WähleSauAus
   (Unselektiert : in out Liste;
   Selektiert : in out Liste)
is
      SchlechteSauVor : Liste := null;
      SchlechteSau    : Liste := Unselektiert;
      SauVor          : Liste := null;
      Sau             : Liste := Unselektiert;
begin
      if Sau = null then
          return;
      end if;
      Sau := Sau.Next;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

Listen - Beispiel - Bauer Gscheidle

Lösung

while Sau /= **null loop**

Listen - Beispiel - Bauer Gscheidle

Lösung

while Sau /= **null loop**
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡

Listen - Beispiel - Bauer Gscheidle

Lösung

while Sau /= **null loop**
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
      SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
        SchlechteSauVor := SauVor;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
        SchlechteSauVor := SauVor;
    end if;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
        SchlechteSauVor := SauVor;
    end if;
    SauVor := Sau;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
        SchlechteSauVor := SauVor;
    end if;
    SauVor := Sau;
    Sau := Sau.Next;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
while Sau /= null loop
    if SauProduktionsWert (Milch, Wolle, Eier, Sau) ¡
       SauProduktionsWert (Milch, Wolle, Eier, SchlechteSau) then
        SchlechteSau := Sau;
        SchlechteSauVor := SauVor;
    end if;
    SauVor := Sau;
    Sau := Sau.Next;
end loop;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

Listen - Beispiel - Bauer Gscheidle

Lösung

if SchlechteSauVor $/=$ **null then**

Listen - Beispiel - Bauer Gscheidle

Lösung

```
if SchlechteSauVor /= null then
    SchlechteSauVor.Next := SchlechteSau.Next;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
if SchlechteSauVor /= null then
    SchlechteSauVor.Next := SchlechteSau.Next;
else
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
if SchlechteSauVor /= null then
    SchlechteSauVor.Next := SchlechteSau.Next;
else
    Unselektiert := SchlechteSau.Next;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
if SchlechteSauVor /= null then
    SchlechteSauVor.Next := SchlechteSau.Next;
else
    Unselektiert := SchlechteSau.Next;
end if;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
    if SchlechteSauVor /= null then
        SchlechteSauVor.Next := SchlechteSau.Next;
    else
        Unselektiert := SchlechteSau.Next;
    end if;
SchlechteSau.Next := Selektiert;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
        if SchlechteSauVor /= null then
            SchlechteSauVor.Next := SchlechteSau.Next;
        else
            Unselektiert := SchlechteSau.Next;
        end if;
    SchlechteSau.Next := Selektiert;
    Selektiert := SchlechteSau;
end WähleSauAus;
```

Listen - Beispiel - Bauer Gscheidle

Schreiben Sie eine Funktion **TreffeSauAuswahl**, die die komplette Liste aller Säue übergeben bekommt, die Anzahl der Säue ermittelt und dann mittels der von Ihnen entworfenen Prozedur **WähleSauAus** die Liste der 10% auszuwählenden Säue erstellt. Diese Liste soll von der Funktion zurückgegeben werden.

Listen - Beispiel - Bauer Gscheidle

Lösung

Listen - Beispiel - Bauer Gscheidle

Lösung

function TreffeSauAuswahl

Listen - Beispiel - Bauer Gscheidle

Lösung

function TreffeSauAuswahl
(AlleSäue : Liste)

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
      Sau          : Liste := AlleSaue;
begin
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
      Sau          : Liste := AlleSaue;
begin
      while Sau /= null loop
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
      Sau          : Liste := AlleSaue;
begin
      while Sau /= null loop
            Sau := Sau.Next;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
      Sau          : Liste := AlleSaue;
begin
      while Sau /= null loop
            Sau := Sau.Next;
            Anzahl := Anzahl + 1;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
function TreffeSauAuswahl
      (AlleSäue : Liste)
   return Liste
is
      Unselektiert : Liste := null;
      Selektiert   : Liste := AlleSäue;
      Anzahl       : Integer := 0;
      Sau          : Liste := AlleSaue;
begin
      while Sau /= null loop
            Sau := Sau.Next;
            Anzahl := Anzahl + 1;
      end loop;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

Listen - Beispiel - Bauer Gscheidle

Lösung

```
for i in 1 .. Anzahl / 10 loop
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
for i in 1 .. Anzahl / 10 loop
    WähleSauAus (Selektiert, Unselektiert);
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
for i in 1 .. Anzahl / 10 loop
    WähleSauAus (Selektiert, Unselektiert);
end loop;
```

Listen - Beispiel - Bauer Gscheidle

Lösung

```
    for i in 1 .. Anzahl / 10 loop
        WähleSauAus (Selektiert, Unselektiert);
    end loop;
    return Unselektiert;
end TreffeSauAuswahl;
```

Listen - Beispiel - Bauer Gscheidle

Das war doch einfach, oder? :)

Baum

Wie sieht ein Baum aus?

Baum

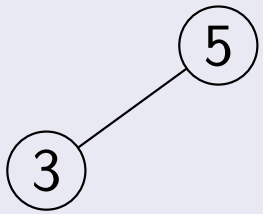
Wie sieht ein Baum aus?

Ein Baum

Baum

Wie sieht ein Baum aus?

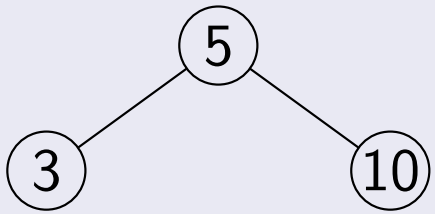
Ein Baum



Baum

Wie sieht ein Baum aus?

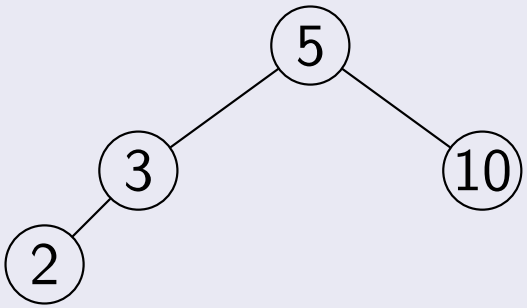
Ein Baum



Baum

Wie sieht ein Baum aus?

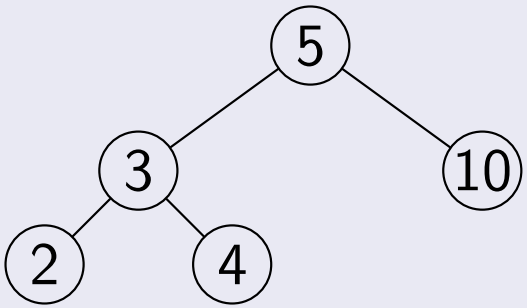
Ein Baum



Baum

Wie sieht ein Baum aus?

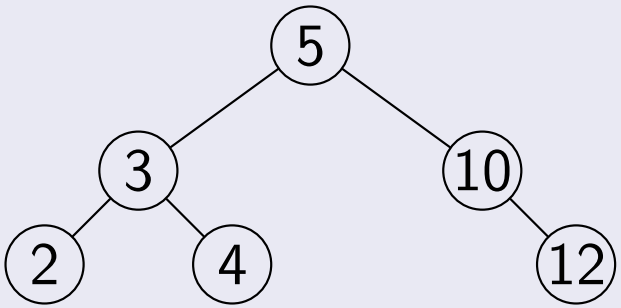
Ein Baum



Baum

Wie sieht ein Baum aus?

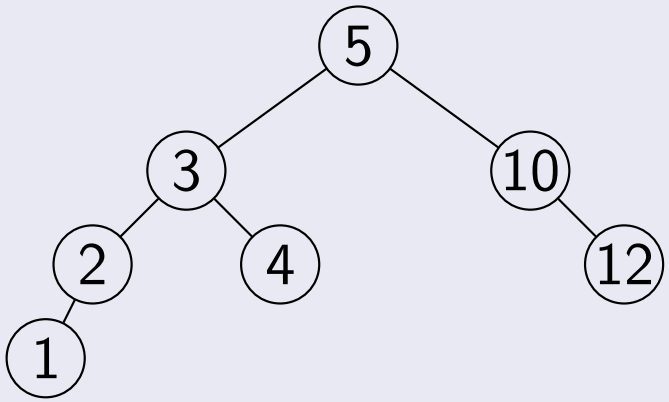
Ein Baum



Baum

Wie sieht ein Baum aus?

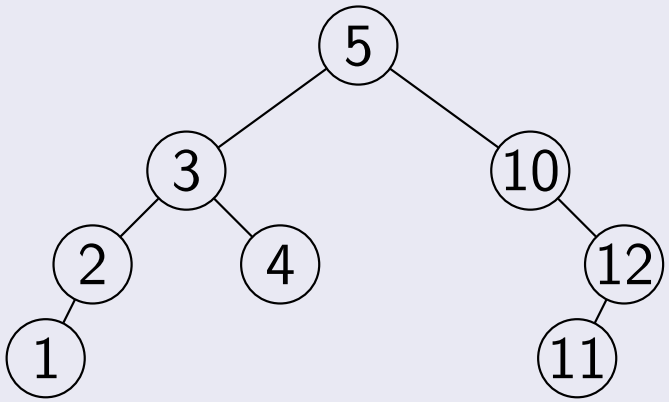
Ein Baum



Baum

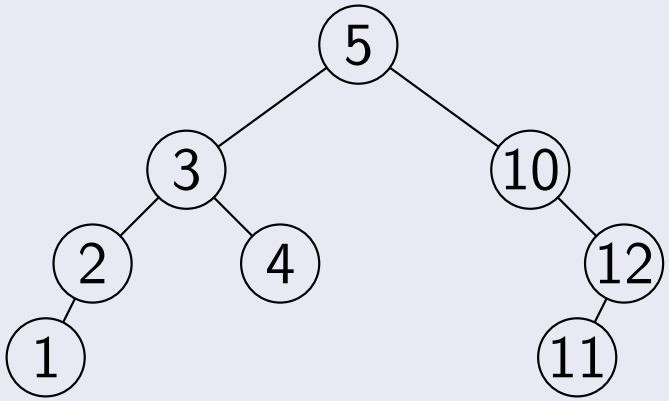
Wie sieht ein Baum aus?

Ein Baum



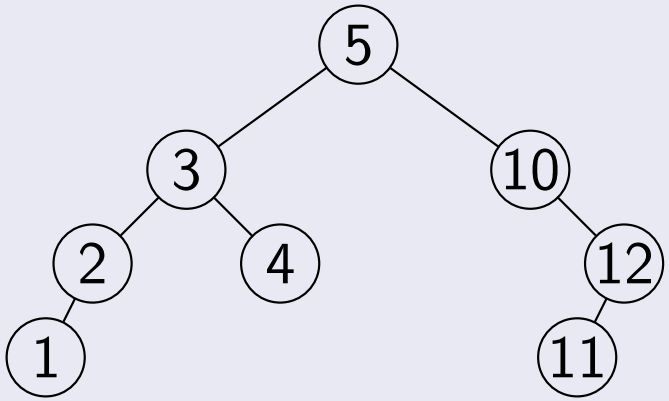
Baumtraversierung

Baumtraversierung



Traversierung

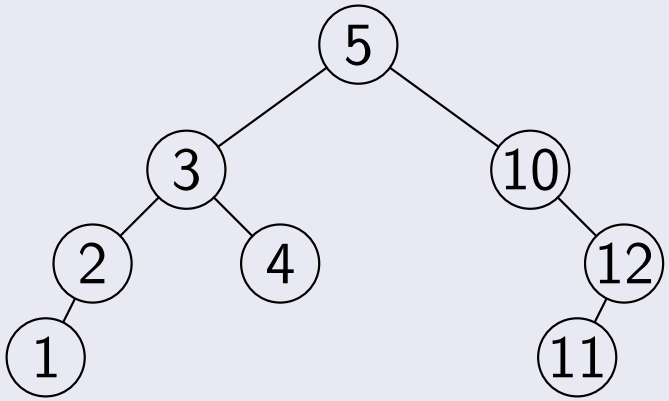
Baumtraversierung



Traversierung

Preorder :

Baumtraversierung

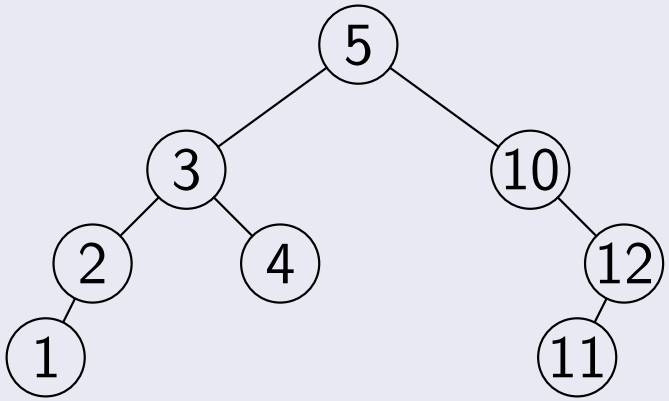


Traversierung

Preorder :

Inorder :

Baumtraversierung



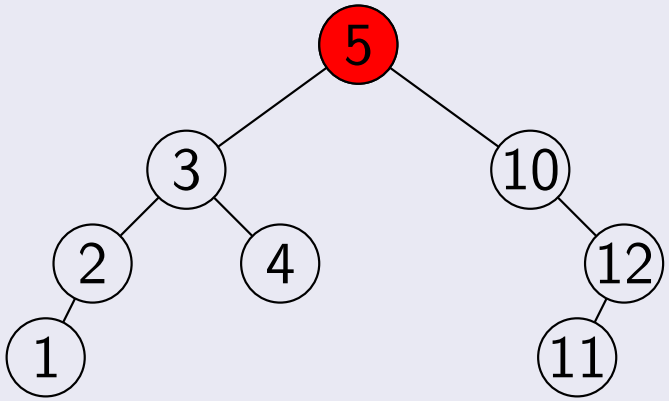
Traversierung

Preorder :

Inorder :

Postorder :

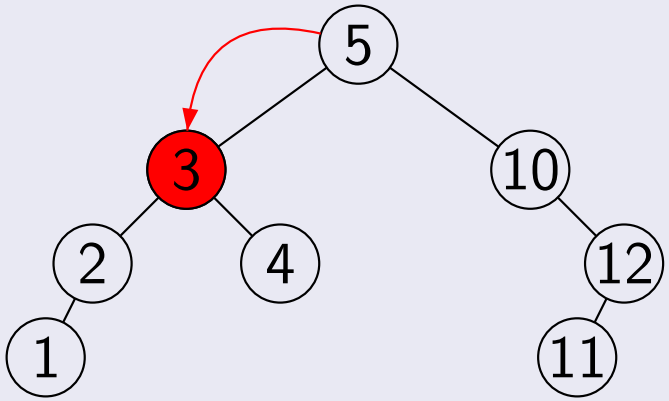
Baumtraversierung



Traversierung

Preorder : 5,
Inorder :
Postorder :

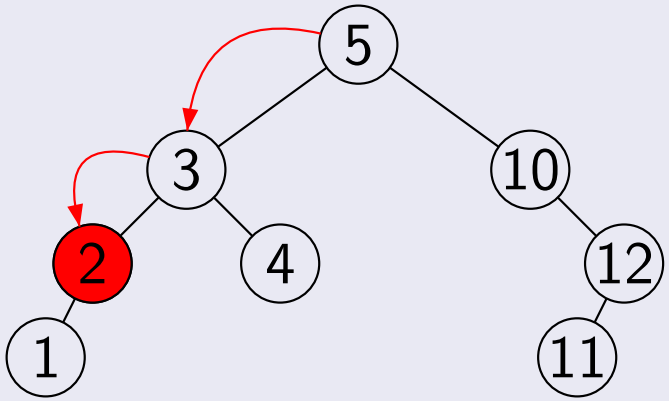
Baumtraversierung



Traversierung

Preorder : 5,3,
Inorder :
Postorder :

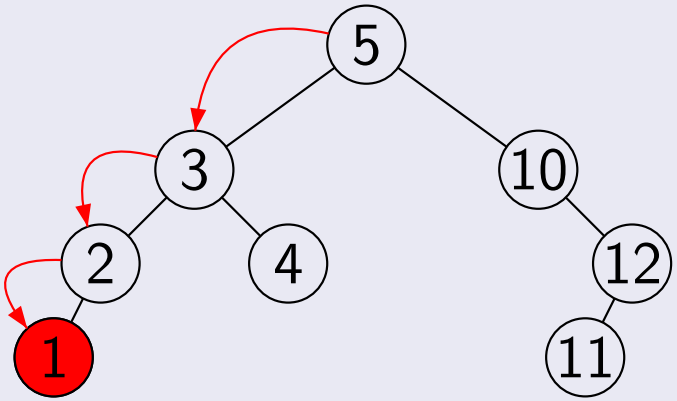
Baumtraversierung



Traversierung

Preorder : 5,3,2,
Inorder :
Postorder :

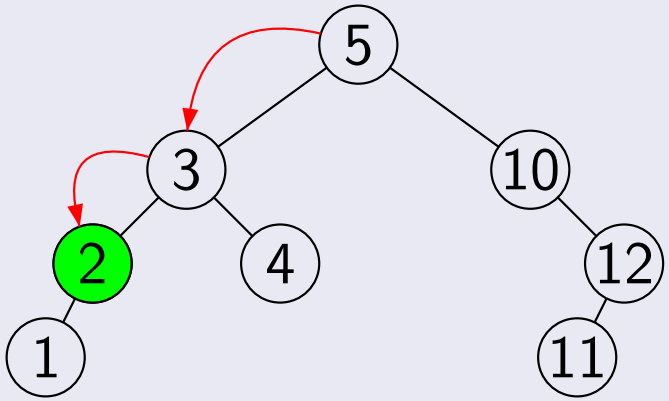
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,
Inorder : 1,
Postorder : 1,

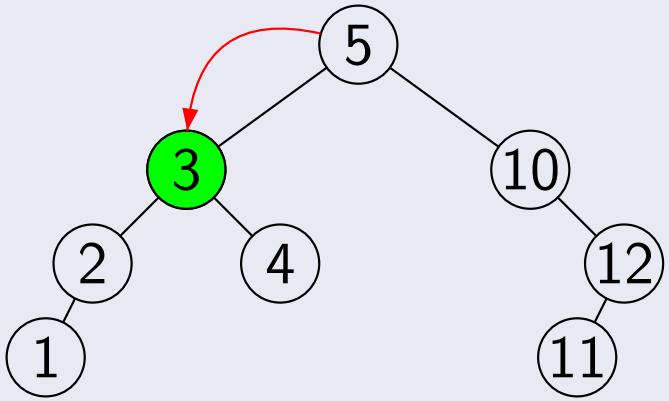
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,
Inorder : 1,2,
Postorder : 1,2,

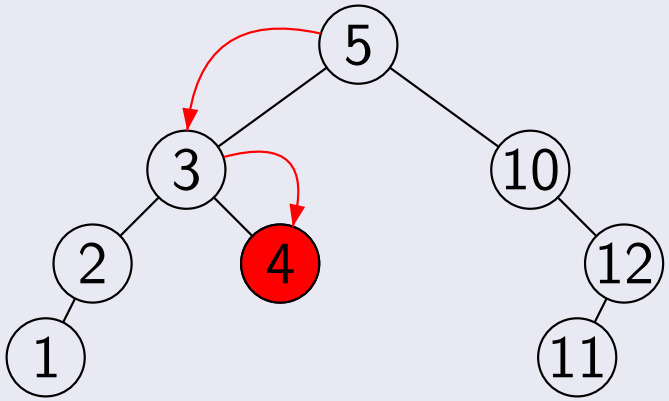
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,
Inorder : 1,2,3,
Postorder : 1,2,

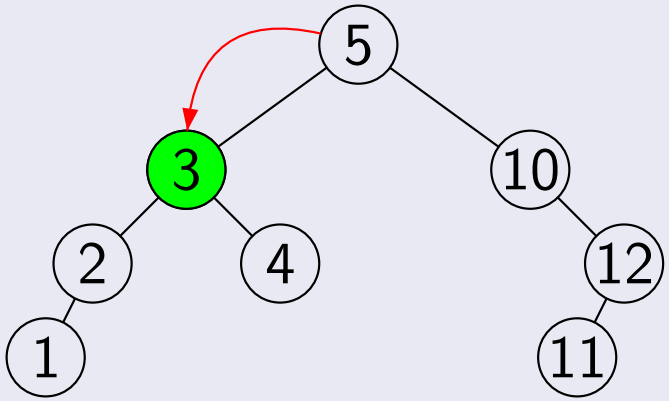
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,
Inorder : 1,2,3,4,
Postorder : 1,2,4,

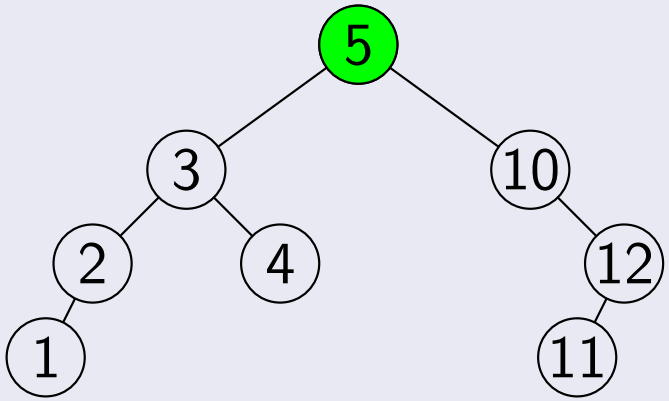
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,
Inorder : 1,2,3,4,
Postorder : 1,2,4,3,

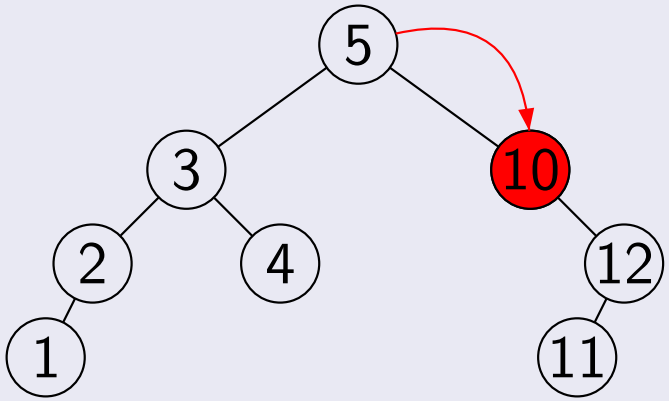
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,
Inorder : 1,2,3,4,5,
Postorder : 1,2,4,3,

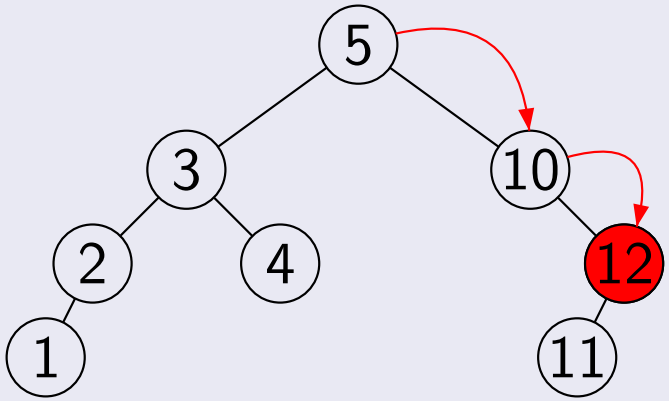
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,
Inorder : 1,2,3,4,5,10,
Postorder : 1,2,4,3,

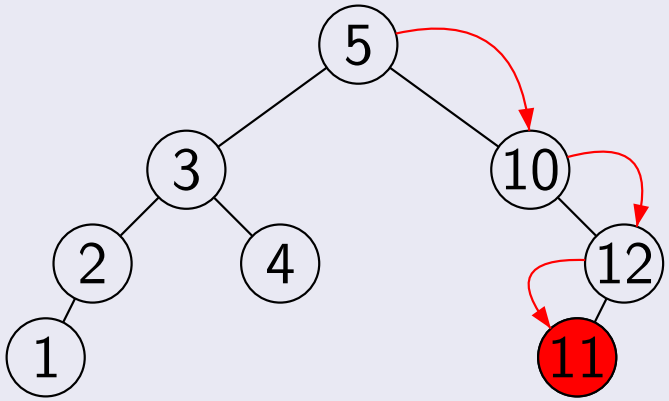
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,12,
Inorder : 1,2,3,4,5,10,
Postorder : 1,2,4,3,

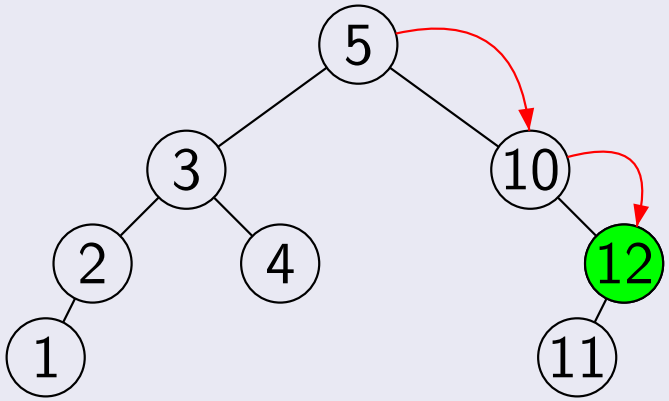
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,12,11
Inorder : 1,2,3,4,5,10,11,
Postorder : 1,2,4,3,11,

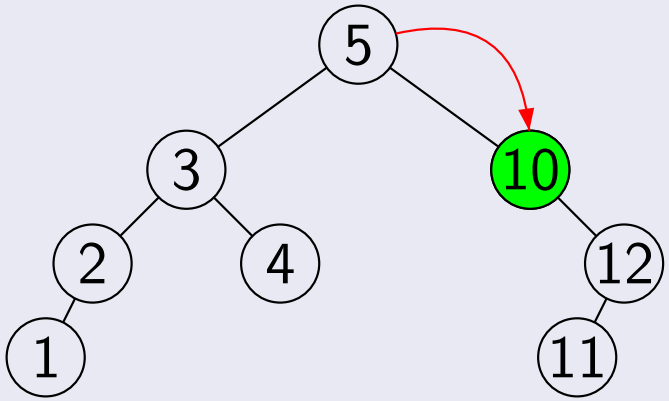
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,12,11
Inorder : 1,2,3,4,5,10,11,12
Postorder : 1,2,4,3,11,12,

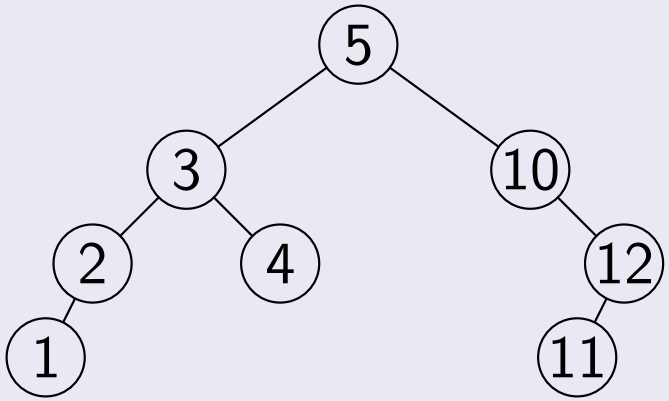
Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,12,11
Inorder : 1,2,3,4,5,10,11,12
Postorder : 1,2,4,3,11,12,10,

Baumtraversierung



Traversierung

Preorder : 5,3,2,1,4,10,12,11
Inorder : 1,2,3,4,5,10,11,12
Postorder : 1,2,4,3,11,12,10,5

Bäume - Der Datentyp

Datentyp

Bäume - Der Datentyp

Datentyp

type Knoten;

Bäume - Der Datentyp

Datentyp

type Knoten;
type Baum **is access** Knoten;

Bäume - Der Datentyp

Datentyp

type Knoten;
type Baum **is access** Knoten;
type Knoten **is record**

Bäume - Der Datentyp

Datentyp

```
type Knoten;
type Baum is access Knoten;
type Knoten is record
                Inhalt     : T;
```

Bäume - Der Datentyp

Datentyp

```
type Knoten;
type Baum is access Knoten;
type Knoten is record
                Inhalt     : T;
                Links      : Baum;
```

Bäume - Der Datentyp

Datentyp

```
type Knoten;
type Baum is access Knoten;
type Knoten is record
                Inhalt    : T;
                Links     : Baum;
                Rechts    : Baum;
```

Bäume - Der Datentyp

Datentyp

```
type Knoten;
type Baum is access Knoten;
type Knoten is record
                Inhalt    : T;
                Links     : Baum;
                Rechts    : Baum;
end record;
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

procedure TraversierungRekursiv

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
            return;
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
            return;
      end if;
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
            return;
      end if;
   PreOrder(Wurzel); – Preorder-Aufruf
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
            return;
      end if;
   PreOrder(Wurzel); – Preorder-Aufruf
      TraversierungRekursiv(Wurzel.Links);
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
   (Wurzel : Baum)
is
begin
      if Wurzel = null then
            return;
      end if;
   PreOrder(Wurzel); – Preorder-Aufruf
      TraversierungRekursiv(Wurzel.Links);
   InOrder(Wurzel); – Inorder-Aufruf
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
  (Wurzel : Baum)
is
begin
    if Wurzel = null then
        return;
    end if;
  PreOrder(Wurzel); – Preorder-Aufruf
    TraversierungRekursiv(Wurzel.Links);
  InOrder(Wurzel); – Inorder-Aufruf
    TraversierungRekursiv(Wurzel.Rechts);
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
  (Wurzel : Baum)
is
begin
    if Wurzel = null then
        return;
    end if;
  PreOrder(Wurzel); – Preorder-Aufruf
    TraversierungRekursiv(Wurzel.Links);
  InOrder(Wurzel); – Inorder-Aufruf
    TraversierungRekursiv(Wurzel.Rechts);
  PostOrder(Wurzel); – Postorder-Aufruf
```

Bäume - Die rekursive Traversierung

Traversierung, rekursiv

```
procedure TraversierungRekursiv
  (Wurzel : Baum)
is
begin
    if Wurzel = null then
        return;
    end if;
  PreOrder(Wurzel); – Preorder-Aufruf
    TraversierungRekursiv(Wurzel.Links);
  InOrder(Wurzel); – Inorder-Aufruf
    TraversierungRekursiv(Wurzel.Rechts);
  PostOrder(Wurzel); – Postorder-Aufruf
end;
```

Bäume - Erzeugung aus Traversierungsdaten

Bäume - Erzeugung aus Traversierungsdaten

Beispiele:

Preorder : 4,7,1,3,2

Inorder : 1,7,3,4,2

Bäume - Erzeugung aus Traversierungsdaten

Beispiele:

Preorder : 4,7,1,3,2
Inorder : 1,7,3,4,2

Preorder : 7,8,2,1,9,4,3,5,6
Inorder : 1,2,8,9,7,3,4,5,6

Bäume - Erzeugung aus Traversierungsdaten

Beispiele:

Preorder : 4,7,1,3,2
Inorder : 1,7,3,4,2

Preorder : 7,8,2,1,9,4,3,5,6
Inorder : 1,2,8,9,7,3,4,5,6

Inorder : 4,1,5,6,3,2
Postorder : 4,5,1,2,3,6

Bäume - Erzeugung aus Traversierungsdaten

Beispiele:

Preorder : 4,7,1,3,2
Inorder : 1,7,3,4,2

Preorder : 7,8,2,1,9,4,3,5,6
Inorder : 1,2,8,9,7,3,4,5,6

Inorder : 4,1,5,6,3,2
Postorder : 4,5,1,2,3,6

Anmerkung

Pre- und Postorder reichen (im allg.) nicht aus um einen Binärbaum eindeutig zu beschreiben!

Bäume - AVL-Bäume

AVL-Bäume an der Tafel! ;)

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)
- $(2, -1) \Rightarrow$ Rechts-Links-Roation (um v und dann um u)

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)
- $(2, -1) \Rightarrow$ Rechts-Links-Roation (um v und dann um u)
- $(-2, 1) \Rightarrow$ Links-Rechts-Roation (um v und dann um u)

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)
- $(2, -1) \Rightarrow$ Rechts-Links-Roation (um v und dann um u)
- $(-2, 1) \Rightarrow$ Links-Rechts-Roation (um v und dann um u)

Anmerkung

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)
- $(2, -1) \Rightarrow$ Rechts-Links-Roation (um v und dann um u)
- $(-2, 1) \Rightarrow$ Links-Rechts-Roation (um v und dann um u)

Anmerkung

- Beim Einfügen von neuen Knoten in einen AVL-Baum muss maximal 2 mal rotiert werden.

Bäume - AVL-Bäume - Rotationen

Grundlegend wird immer der dem eingefügten Knoten am nächsten liegende Knoten mit der Balance 2 als u und der Knoten direkt darunter auf dem Pfad zum neuen Knoten wird v genannt. Diese Kombination der Balancen von u und v stellen wir als Tuppel (b_u, b_v) dar. Anhand dieses Tupels kann entschieden werden, welche Rotation(en) durchgeführt werden müssen.

- $(2, 1) \Rightarrow$ Links-Roation (um u)
- $(-2, -1) \Rightarrow$ Rechts-Roation (um u)
- $(2, -1) \Rightarrow$ Rechts-Links-Roation (um v und dann um u)
- $(-2, 1) \Rightarrow$ Links-Rechts-Roation (um v und dann um u)

Anmerkung

- Beim Einfügen von neuen Knoten in einen AVL-Baum muss maximal 2 mal rotiert werden.
- Beim Löschen von Knoten innerhalb eines AVL-Baumes muss eventuell mehrfach rotiert werden.

Bäume - AVL-Bäume

Vielen Dank für Ihre Aufmerksamkeit und viel Erfolg bei Ihrer Prüfung!