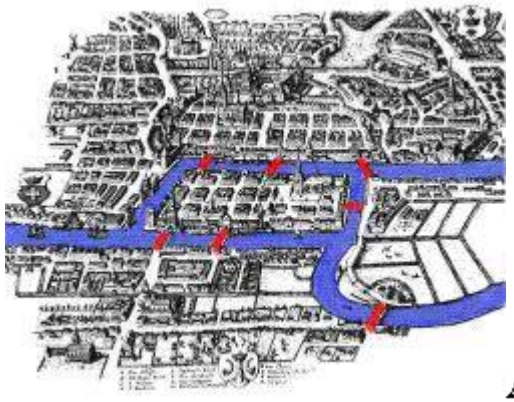
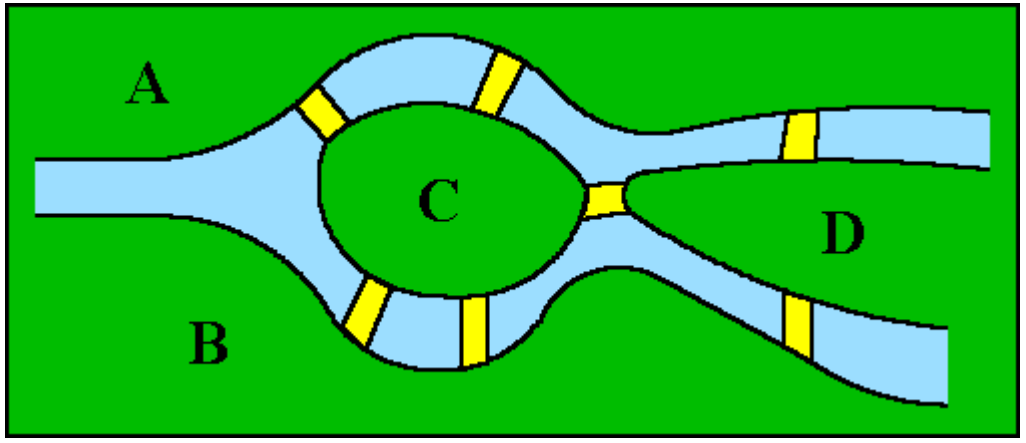
Grafen en bomen

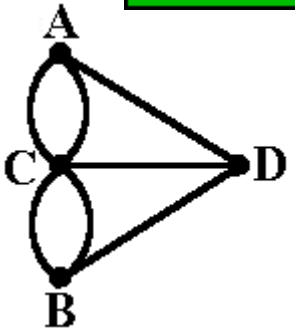
Grafen

- Koningsberger bruggen probleem (Euler, 1707-1783)

  is het mogelijk om een stadswandeling te maken die precies één keer over elk van de 7 bruggen gaat?





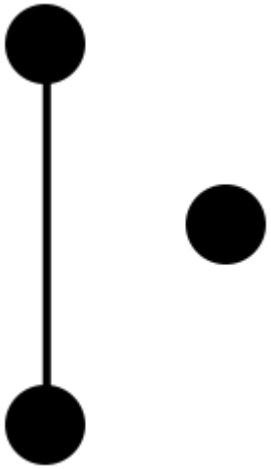


← *kun je dit figuurtje tekenen zonder de potlood van het papier te halen (en elke lijn maar één keer te volgen)?*

Grafen

Een *graaf* bestaat uit een verzameling punten, knopen genoemd, waarvan sommige verbonden zijn door lijnen, de zijden, kanten of takken.

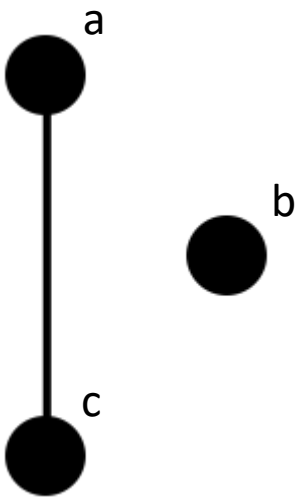
Bron: wikipedia



= één graaf

Grafen

Graaf 1



3 knopen: a, b en c

1 tak: {a,c}

- Altijd een set van knopen
- Takken/paden zijn niet verplicht

Hoe te definiëren?

Ongerichte grafen

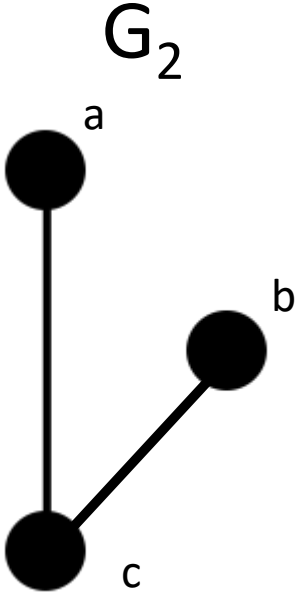
graaf = (knopen, paden)

G_2 = ({a, b, c} , { {a, c}, {c, b} })

K =

P =

Gebruik accolades om een set aan te geven waarvan de inhoud niet geordend is.



Gerichte grafen

Pijlen i.p.v. streepjes

$G_4 = (\{a,b\} , \{ \quad (a,b) \quad \})$

Gebruik ronde haakjes om een set aan te geven waarvan de inhoud wel geordend is.

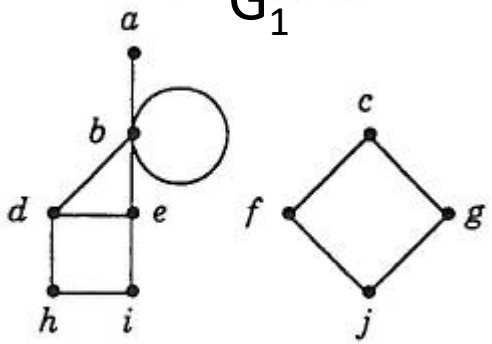
Grafen definities

- Pad_1 = a → b → e
- De padlengte van Pad_1 = 2
- Een graaf is *verbonden* als alle knopen (indirect) met elkaar verbonden zijn

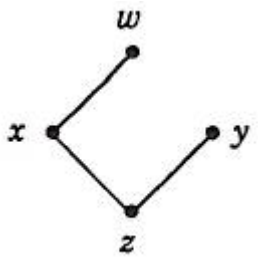
  G_2 dus wel en G_1 niet
- De graad van een knoop is het aantal takken dat uitmondt in die knoop
  (bij gerichte grafen: ingraad en uitgraad)

  graad(d) = 3, graad(b) = 5

G_1



G_2



Opdracht: teken graaf F_{1337}

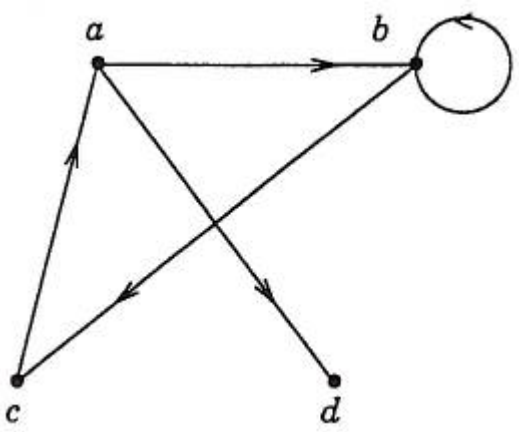
$F_{1337} = (X , Y)$

$X = \{a,b,c,d,e\}$

$Y = \{(a,b),(b,e),(e,c),(c,d),(d,e),(e,a),(a,c)\}$

Gerichte grafen definities

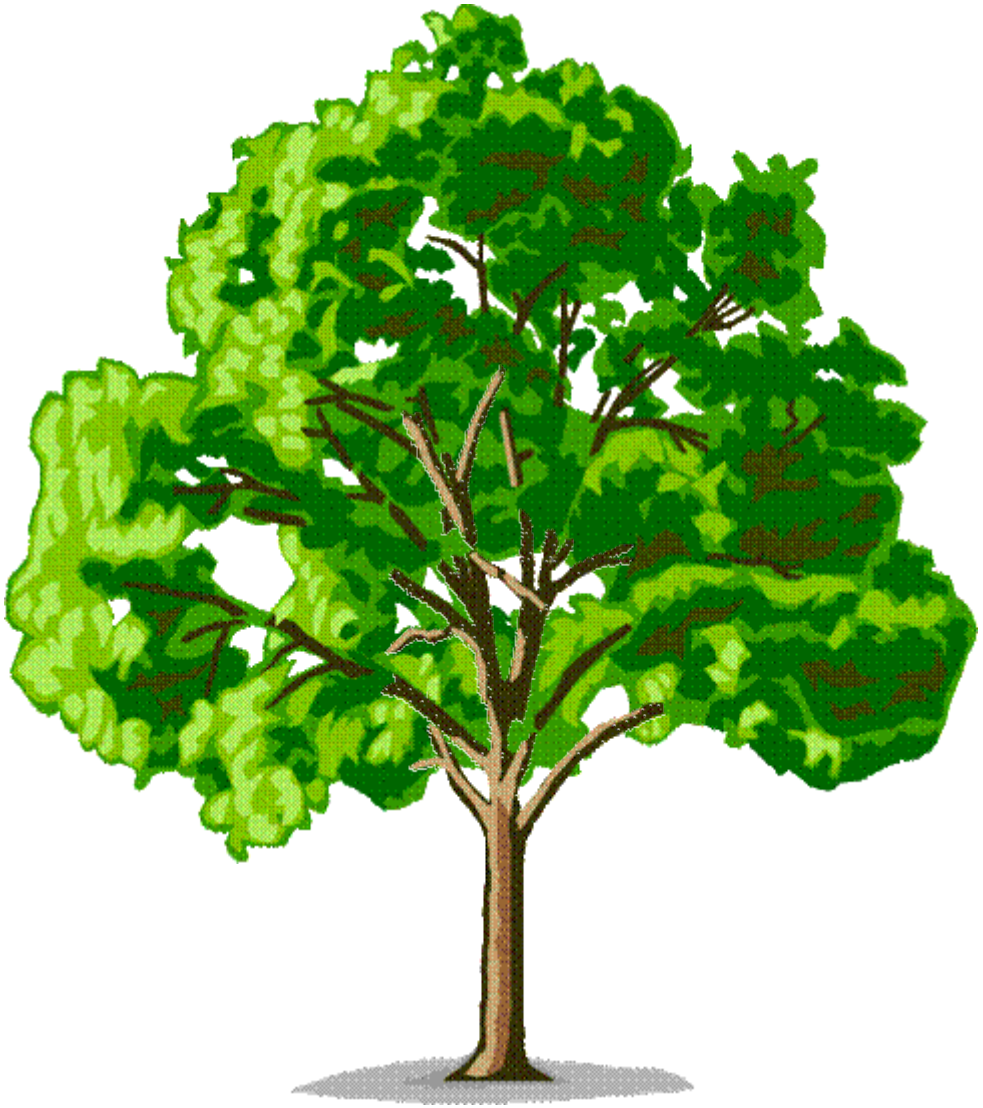
- Ingraad (a) = 1
- Uitgraad (a) = 2



- Pad_2 = b → b → c is gericht
- Pad_3 = a → b → b → a is ongericht

- *Volledig* verbonden als van elke knoop naar elke knoop een *gericht* pad is

Bomen

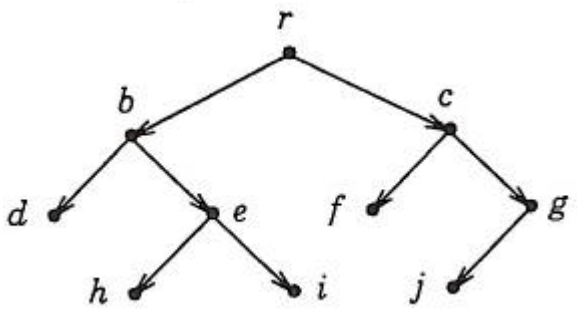


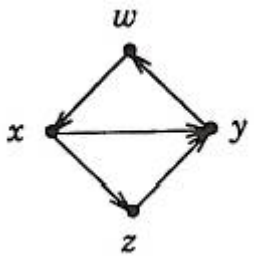
Bomen

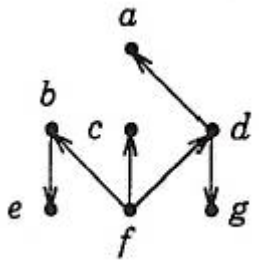
- Een boom is een *verbonden*, *gerichte* graaf

- Er is één knoop met ingraad 0.
  Deze knoop noemen we de *wortel*

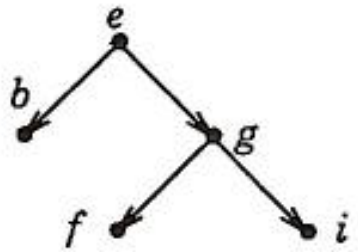
- Alle andere knopen hebben ingraad 1







Boom begrippen



Knoop e staat op niveau 0, b en g op niveau 1, f en i op niveau 2 (diepte is 2)

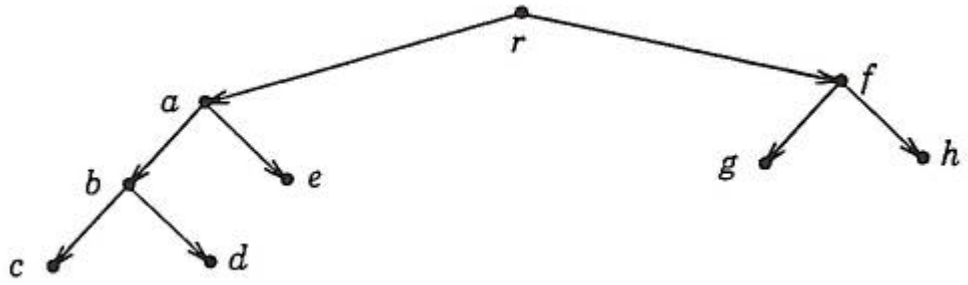
De diepte van de boom is het ‘hoogste’ niveau

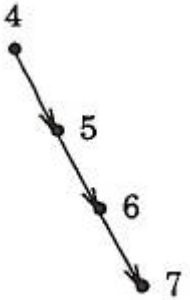
Een knoop zonder zonen/afstammelingen noemen we een ‘blad’; andere zijn ‘interne knopen’

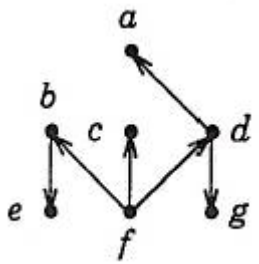
Binaire bomen

Extra eis: maximale uitgraad van 2

Een binaire boom is *compleet* als elke knoop precies 2 *of* 0 als uitgraad heeft.





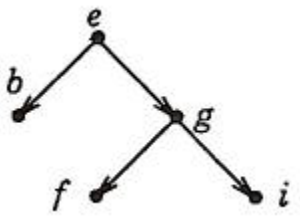


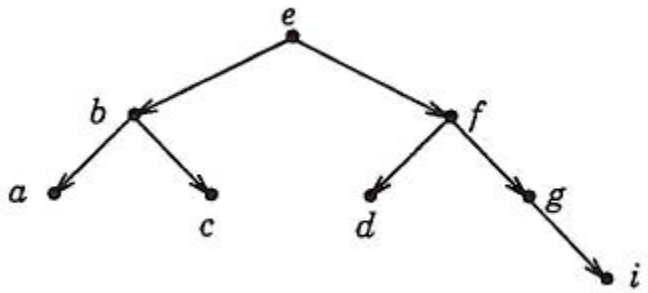
Geordende binaire bomen

om

We noemen een binaire boom *geordend* wanneer er een vaste ordening in alle knoop-waarden te herkennen is en wel zodanig dat voor elke knoop in de *linker-subboom* alle knoop-waarden kleiner zijn dan de onderhavige knoop-waarde en in de *rechter-subboom* alle knoop-waarden groter zijn dan de onderhavige knoop-waarde.

gebruikelijke'. In theorie echter

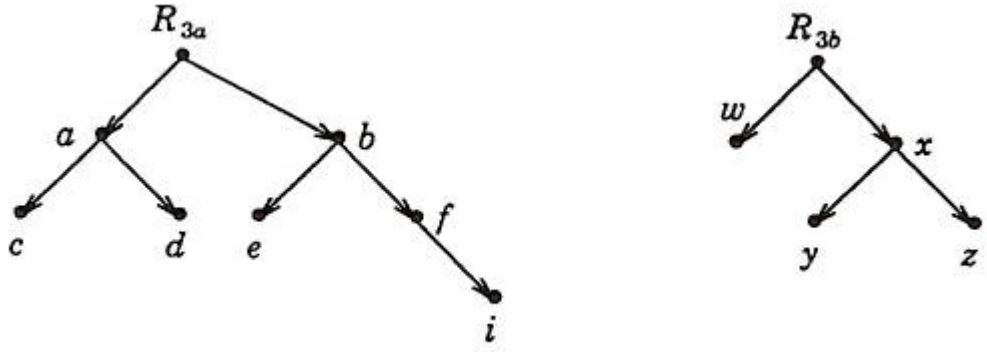




Knoop f opzoeken in de linker boom

Gebalanceerde binaire bomen

Een binaire boom is gebalanceerd als voor elke knoop geldt dat het aantal afstammelingen links en rechts maximaal één verschilt.

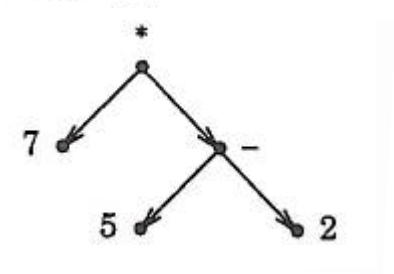


Perfect gebalanceerd
is zonder verschil in het aantal afstammelingen

Bomen praktijk voorbeeld

Rekenmachine gebruiken bomen om sommen in op te slaan

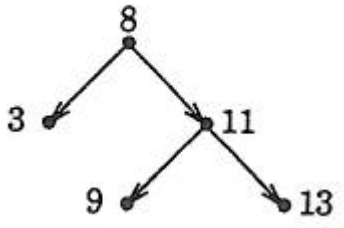
In de boom B_1 is de som 7 * (5 – 2) opgeslagen



Binaire bomen uitlezen: Preorder

Bekijk de boom en al zijn subbomen.

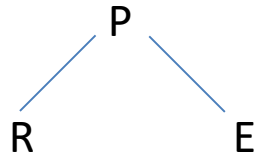
1. Pak eerst de wortel op
2. Pak daarna de linker zoon op
3. Pak daarna de rechter zoon op



Herhaal *eerst* de stappen als de zoon afstammelingen heeft voordat je de knoopwaarde opschrijft.

8 3 11 9 13

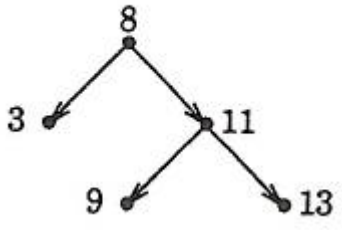
Ezelsbruggetje



Binaire bomen uitlezen: Inorder

Bekijk de boom en al zijn subbomen.

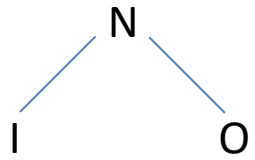
1. Pak eerst de linker zoon op
2. Pak daarna de wortel op
3. Pak daarna de rechter zoon op



Herhaal *eerst* de stappen als de zoon afstammelingen heeft voordat je de knoopwaarde opschrijft.

3 8 9 11 13

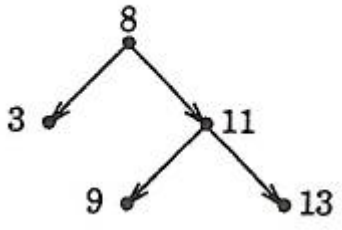
Ezelsbruggetje



Binaire bomen uitlezen: Postorder

Bekijk de boom en al zijn subbomen.

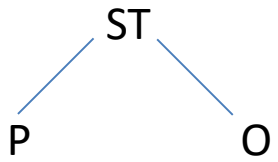
1. Pak eerst de linker zoon op
2. Pak daarna de rechter zoon op
3. Pak daarna de wortel op



Herhaal *eerst* de stappen als de zoon afstammelingen heeft voordat je de knoopwaarde opschrijft.

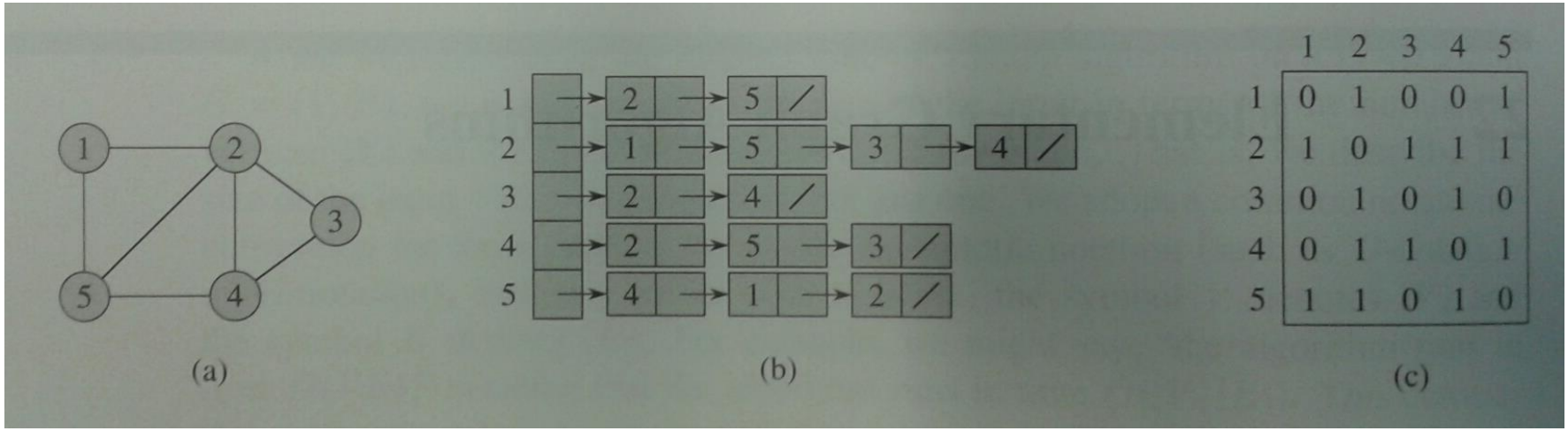
3 9 13 11 8

Ezelsbruggetje



Datastructuur voor grafen

- lijst van verbindingen
- (bool) matrix van verbindingen
- ???



Datastructuur voor (binaire) bomen

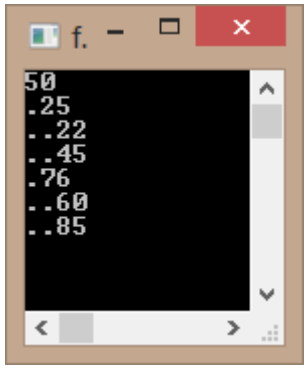
– class Node (met LeftNode, RightNode, ParentNode)

```
class Node
{
  public int Value { get; set; }
  public Node LeftNode { get; private set; }
  public Node RightNode { get; private set; }
  public Node Parent { get; private set; }

  public Node(int value, Node parent)
  {
    this.Value = value;
    this.Parent = parent;
    this.LeftNode = null;
    this.RightNode = null;
  }

  // ...
}
```

```
public Node SetLeftNode(int value)
{
  this.LeftNode = new Node(value, this);
  return this.LeftNode;
}

public Node SetRightNode(int value)
{
  this.RightNode = new Node(value, this);
  return this.RightNode;
}
```

Datastructuur voor (binaire) bomen

```
static void Main(string[] args)
{
  Node tree = new Node(50);
  Node left = tree.SetLeftNode(25);
  Node right = tree.SetRightNode(76);
  left.SetLeftNode(22);
  left.SetRightNode(45);
  right.SetLeftNode(60);
  right.SetRightNode(85);
}
```




```
static void PrintTree(Node tree)
{
  PrintNode(0, tree);
}

static void PrintNode(int level, Node node)
{
  if (node == null) return;
  for (int i = 0; i < level; i++)
    Console.Write(".");
  Console.WriteLine(node.Value);

  PrintNode(level + 1, node.LeftNode);
  PrintNode(level + 1, node.RightNode);
}
```

Datastructuur voor (binaire) bomen

```
static void Main(string[] args)
{
  Node tree = new Node(50);
  // ...

  Node resultNode = FindValueNode(tree, 86);
  if (resultNode == null)
    Console.WriteLine("value not found");
  else
    Console.WriteLine("value found");

  Console.ReadKey();
}
```

```
static Node FindValueNode(Node node, int value)
{
  if (node == null) return null;
  if (node.Value == value)
    return node;
  else if (value > node.Value)
    return FindValueNode(node.RightNode, value);
  else
    return FindValueNode(node.LeftNode, value);
}
```