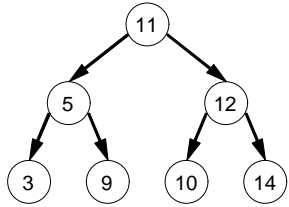


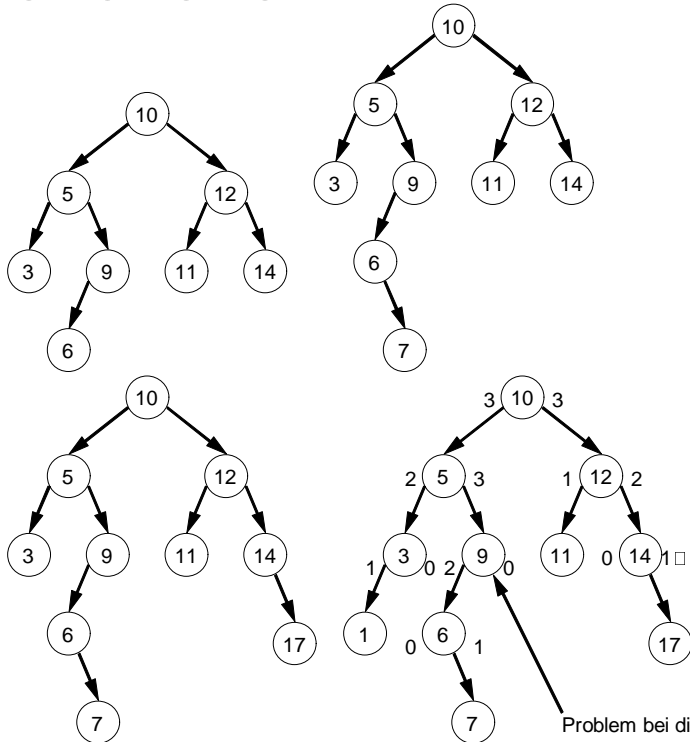
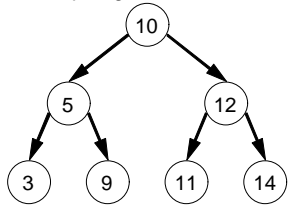
1) Binärbäume

Gegeben sei folgender Binärbaum:



Überprüfen Sie, ob ein binärer Suchbaum vorliegt (und korrigieren Sie den Baum bei Bedarf). Fügen Sie folgende Elemente in den natürlichen (= nicht balancierten), möglicherweise korrigierten, binären Suchbaum ein: 6, 7, 17, 1. Ist der resultierende Binärbaum ein AVL-Baum? Begründen Sie Ihre Antwort!

Der ursprüngliche Baum ist kein binärer Suchbaum, erst nach Vertauschen von 10 und 11:



Der entstandene binäre Suchbaum ist kein AVL-Baum!

2) Bäume

Gegeben sei der folgende Knotentyp.

```

TYPE
  Tree = POINTER TO NodeDesc;
  NodeDesc = RECORD
    left, right: Tree;
    val: INTEGER
  END;
    
```

Implementieren Sie einen Algorithmus der bestimmt wieviele Knoten ein Baum besitzt. Die Prozedur soll die folgende Schnittstelle haben.

```

PROCEDURE NumberOfNodes (t: Tree) : INTEGER;
    
```

Achten Sie auch auf den Sonderfall, dass ein leerer Baum (t=NIL) übergeben wird.

```

PROCEDURE NumberOfNodes (t: Tree) : INTEGER;
BEGIN
  IF t = NIL THEN RETURN 0
  ELSE RETURN 1 + NumberOfNodes(t.left) + NumberOfNodes(t.right)
  END
END NumberOfNodes;
    
```

3) Balancierung von Bäumen

Zeigen Sie, wie der folgende Baum schrittweise in eine Rebe umgewandelt wird. Verwenden Sie dazu den in der Vorlesung besprochenen Algorithmus *TreeToVine*. Geben Sie den Baum für jede Iteration der WHILE-Schleife an.

```

PROCEDURE TreeToVine (VAR root: BinTree; VAR n: INTEGER);
  VAR tail, rest, p: BinTree;
BEGIN
  tail := root; rest := root.right;
  n := 0;
  WHILE rest # NIL DO
    IF rest.left = NIL THEN (* move tail down *)
      tail := rest; rest := rest.right; INC(n)
    ELSE (* rotate *)
      p := rest.left; rest.left := p.right; p.right := rest; rest := p;
      tail.right := p
    END
  END
END TreeToVine;
    
```

