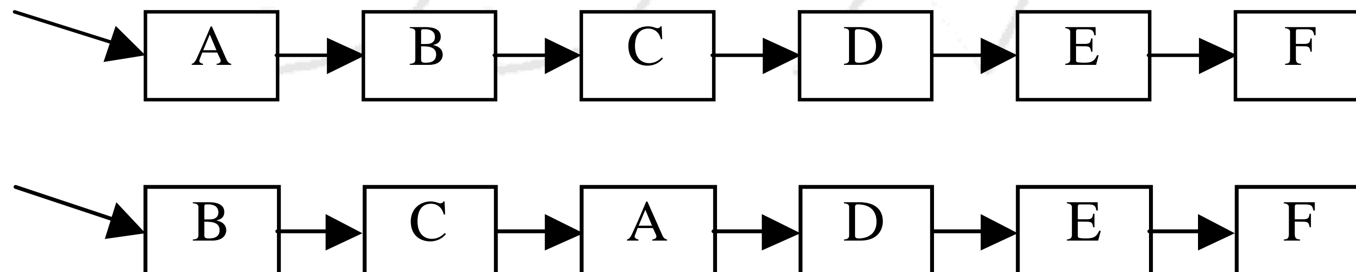
Lineare Liste

In einer linearen Liste soll das erste Element um k Stellen in der Liste nach hinten verschoben werden (`move`). Kann nicht um k Elemente verschoben werden, sollen das Element hinten angehängt werden.
Verwenden Sie dazu folgende Definition der Liste:

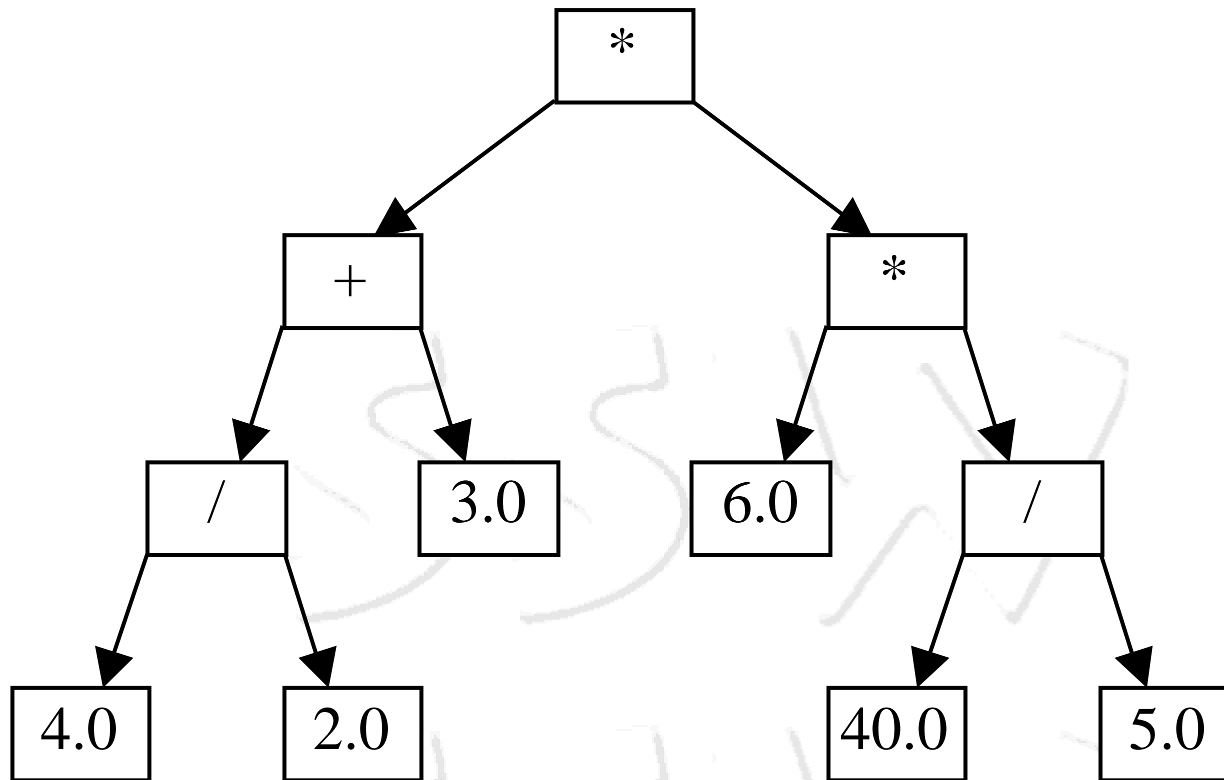
```
class List {
  Node head;
  void move(int k) {}
}
```

```
class Node {
    Node next;
    String value;
}
```

Beispiel:
`aList.move(2);`



Durchlauf durch Binärbaum



Geben Sie den Baum "in order", "pre order" und "post order" aus.

Binärer Suchbaum (1)

Ein Kaufhaus hat eine Vielzahl von Artikeln im Angebot. Diese Artikel haben einen bestimmten Preis und eine Beschreibung, die in folgender Datenstruktur verwaltet werden:

```
class ItemNode { // Ein Artikel
      ItemNode left, right;
      String description;
      double price;    //Preis des Artikels
      void printItem() //Ausgabe des Artikels
                       //(description + price)
}
```

Die Artikel werden in einem binären Suchbaum (sortiert nach dem Preis) verwaltet:

```
class InventoryTree {
      ItemNode root;
      void printCheapest(double maxPrice) {...}
}
```

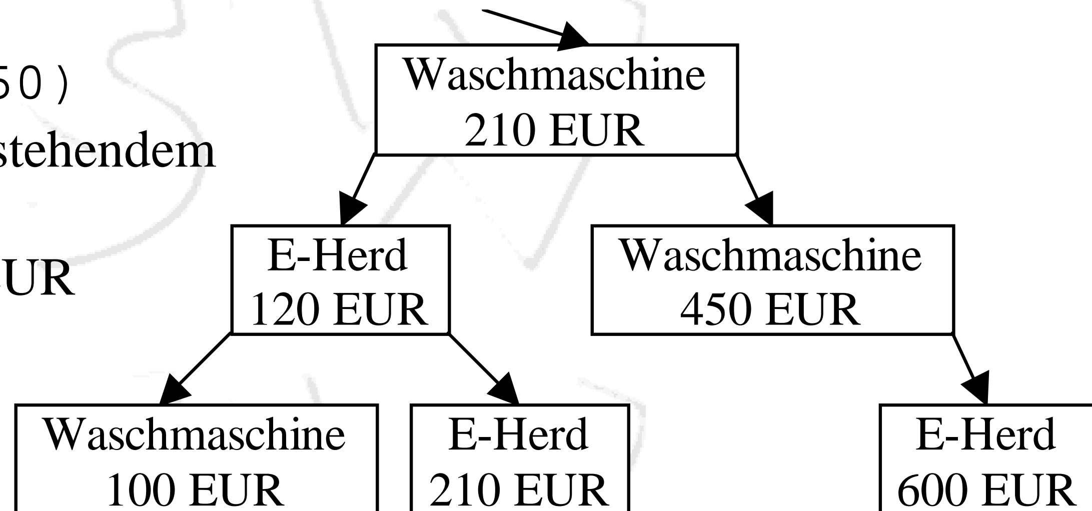
Binärer Suchbaum (2)

Das Kaufhaus möchte nun sein Image als Ramsch-Laden loswerden, weshalb alle billigen Artikel abtransportiert werden sollen. Gesucht ist die Implementierung für die angegebene Methode `printCheapest`, die die billigsten Artikel sortiert bis zu einem bestimmten Gesamtpreis ausgibt.

Beispiel:

`printCheapest(250)`

liefert bei nebenstehendem Baum als Ergebnis:
Waschmaschine, 100 EUR
E-Herd, 120 EUR



Anm.: Achten Sie auf eine effiziente Implementierung (es sollen möglichst wenige Knoten im Baum besucht werden). Nutzen Sie dazu die Sortierung des Suchbaums aus.

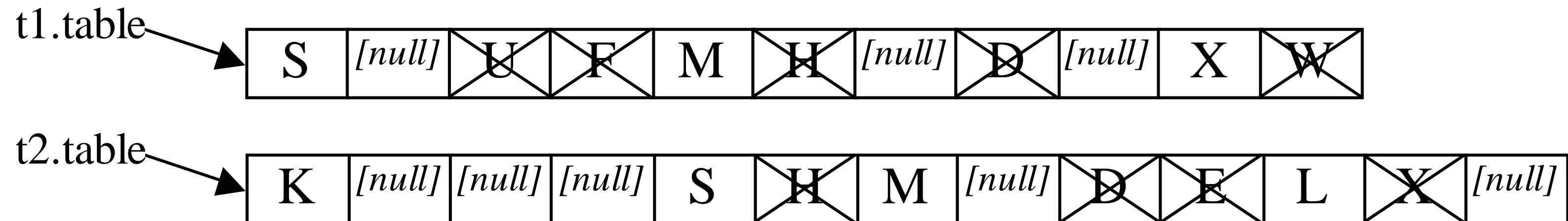
Hashtabelle (1)

```
public class Hashtable {
  Node[] table;
  int nOfElements, nOfDeleted;

  public Hashtable intersect (Hashtable t) {...}

  public boolean isSubsetOf(Hashtable t) {...}

  // Weitere Methoden (zur freien Verwendung)
  public Hashtable(int initSize) {...}
  public void insert(String val) {...}
  public void delete(String val) {...}
  public boolean contains(String val) {...}
}

class Node {
  boolean deleted;
  String value;

  Node(String val) {
    value = val;
  }
}
```

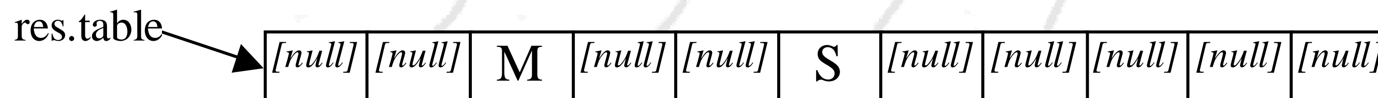
Gesucht:
intersect (Schnittmenge)
isSubsetOf (Ist Teilmenge von)

Hashtabelle (2)

Bsp:



res = t1.intersect(t2);



// res.isSubsetOf(t2) ➔ true

res.insert("K");

// res.isSubsetOf(t2) ➔ true

Graphen (1)

Gegeben sei ein zusammenhängender, gerichteter Graph mit Zyklen. Es sollen in diesem Graph alle Zyklen entfernt werden.

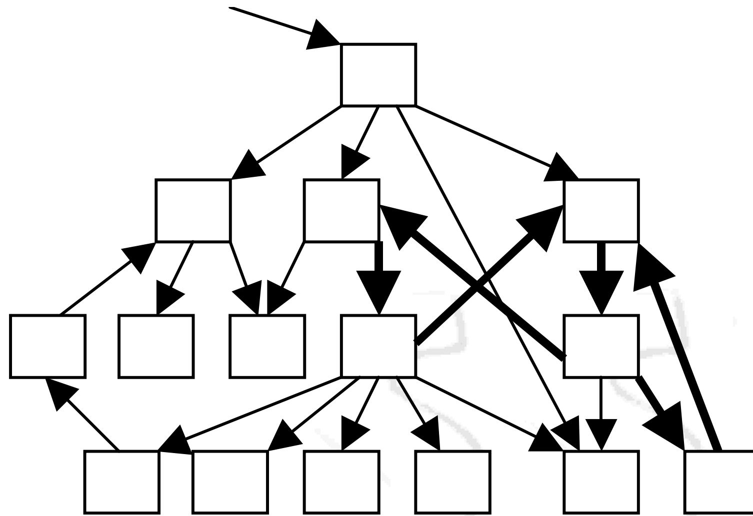
```
class Node {
  Node[] neighbours; // Direkte Nachbarn
  // Entfernt die Kante zu Knoten n
  public void removeNeighbour(Node n);
}
```

Gestartet wird von einem ausgezeichneten Knoten (root), bei dem sichergestellt ist, dass alle anderen Knoten erreicht werden können.
Gesucht ist eine Methode `breakCycles()`, der den Graphen nach Zyklen durchsucht und diese durch Entfernen der Kante, durch die der Zyklus gefunden wurde, auflöst.

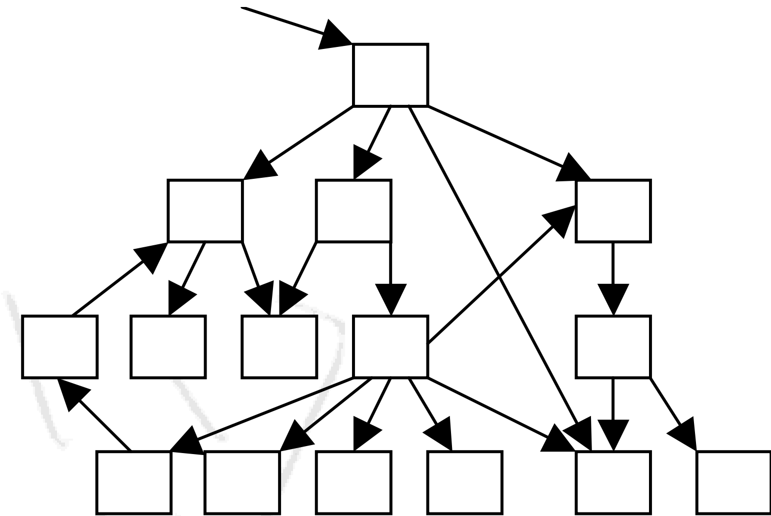
```
public class Graph {
  private Node root;
  public boolean breakCycles() {...}
}
```

Graphen (2)

Beispiele:



Mit Zyklen



nach Ausführung von breakCycles()

Hinweis: Es können den Knoten noch weitere Attribute hinzugefügt werden.