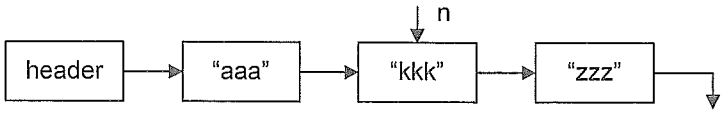
Tentamen in1605 Datastructuren en Algoritmen, 14-4-2010, 9.00-12.00
TU Delft, Faculteit EWI, Basiseenheid Software Engineering
U kunt gebruik maken van het studieboek.

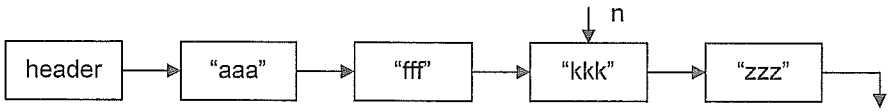
Opgave 1 (25 punten)

In Bijlage 1 is de implementatie gegeven van klasse LNode. Een LNode bevat een element (element:Object) en een verwijzing (next:LNode) naar een andere LNode. Met behulp van LNode's kan een enkelvoudige, lineaire lijst worden gemaakt. In Bijlage 1 vind je een deel van de implementatie van klasse List. Bestudeer beide klassen goed voordat je begint met het uitwerken van deze opgave. Hieronder is een voorbeeld van List gegeven.

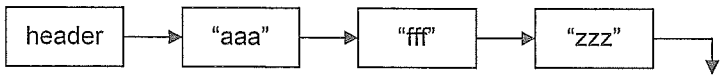


Gegeven de specificatie van een viertal methoden van klasse List:

```
insertBefore(n : LNode, o : Object)
pre: n is niet null en ongelijk aan de header-node
post: indien pre geldt, is er een nieuwe LNode die Object o bevat, aan
de List toegevoegd. De nieuwe LNode wijst naar LNode n. (zie hieronder)
```




```
remove(n : Lnode) : Object
pre: n is niet null
post: indien pre geldt, is de LNode die door n wordt aangewezen uit de
List verwijderd, en is het element van deze knoop geretourneerd; zoniet
dan is er null geretourneerd. (zie hieronder)
```




```
contains(o : Object) : boolean
post: retourneert true dsd er een LNode in de List voorkomt met een
Object dat gelijk is aan o

size() : int
post: retourneert het aantal LNodes in de List  (exclusief header).
```

Mocht je bij de implementatie van een van de methoden een hulpmethode nodig hebben, geef dan een volledige implementatie van die hulpmethode!

a. Geef een implementatie van methode insertBefore.
b. Geef een implementatie van methode remove.
c. Geef een implementatie van methode contains.
d. Geef een implementatie van methode size. **Deze methode moet gebruik maken van recursie!**
e. Wat is de tijdcomplexiteit van methode insertBefore? Licht je antwoord toe.
f. Wat is de tijdcomplexiteit van methode remove? Licht je antwoord toe.

Opgave 2 (18 punten)

a. Gegeven de volgende differentievergelijking voor T(n):

$T(n) = 5$,als $n = 0$
$T(n) = T(n-1) + 6n - 3$,voor $n > 0$

Geef een oplossing voor deze differentievergelijking. Laat zien hoe je deze oplossing hebt afgeleid.

b. Gegeven de volgende differentievergelijking:

$T(n) = c$, voor $n < d$
$T(n) = aT(n/b) + f(n)$, voor $n >= d$

Voor welke waarden van a, b en f(n) is T(n) van de orde O(nlog(n)) ? Licht je antwoord toe.

Opgave 3(17 punten)

Gegeven een verzameling van gehele getallen. Een voorbeeld hiervan is {20,4,15,16,11,5,10,2}. Uit de verzameling kunnen deelverzamelingen worden gevormd, zoals {20,5,15}, {11} of {}. De som van getallen uit de gegeven verzamelingen is 40, 11 of 0.

Gevraagd wordt om een methode die de deelverzameling bepaalt met de meeste elementen, waarvan de som kleiner of gelijk is aan een gegeven getal X. Bij de implementatie worden de verzamelingen gerepresenteerd door array's.

```
+grootsteDeelverzameling(x : int, lijst : int[]) : int[]
post: retourneert de grootste verzameling van de getallen uit lijst,
waarvan de som kleiner is of gelijk aan x.
```

a. Geef een implementatie van methode grootsteDeelverzameling.

b. Wat is de tijdcomplexiteit van de implementatie? Licht je antwoord toe.

c. Geef een redenering waaruit blijkt dat de implementatie correct is.

Opgave 4(30 punten)

In Bijlage 2 is de implementatie gegeven van klasses TNode. Een TNode bestaat uit een element(String), en twee verwijzingen(TNode) naar links en rechts. Er is geen verwijzing naar een parent!

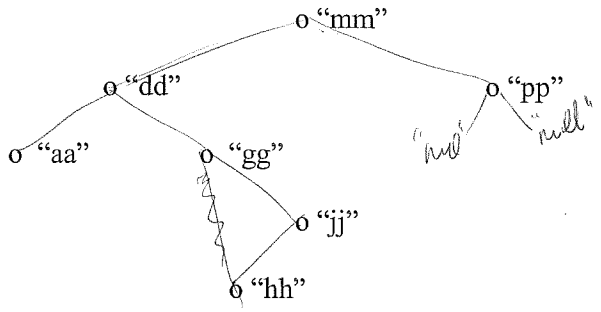
De klasse bevat een aantal- set- en get-methoden, en een methode isExternal.
Een TNode is een external als de linker- en rechterverwijzingen leeg zijn. Het **element** van een external TNode is altijd null.

Een BSTree is binaire zoekboom waarin Strings worden opgeslagen. Strings kunnen met methode compareTo() onderling worden vergeleken. Als een String eerder in een woordenboek voorkomt dan een andere String beschouwen we de 1e String als kleiner dan de 2e String.

Een lege BSTree bevat 1 externe knoop. De bijlage bevat een klein deel van de implementatie van klasse BSTree.

Bij het toevoegen worden elementen kleiner dan het element in de wortel van de (sub)boom links toegevoegd, elementen even groot of groter dan het element in de wortel rechts.

Hieronder is een voorbeeld gegeven van een BSTree.



In bijlage 2 is een deel van de implementatie van de methoden insert, treeToVector en vectorToTree gegeven. Deze methoden roepen **gelijknamige** methoden aan die recursief werken. Gevraagd wordt om de implementatie van de gelijknamige methoden en van de aangegeven methoden.

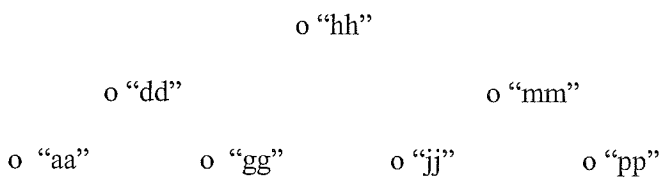
Methode insert voegt elementen aan een binaire zoekboom toe op zo'n manier dat de boom na toevoeging nog steeds een binaire zoekboom is.

Een binaire zoekboom is niet vanzelf in balans. In het voorbeeld is een zoekboom gegeven waarvan de hoogte van de linkerboom veel groter is dan de hoogte van de linkerboom. Om een niet-gebalanceerde boom in balans te brengen zijn de methoden treeToVector en vectorToTree ontworpen.

Methode treeToVector voegt de elementen uit de boom toe aan een Vector. Het kleinste element staat vooraan in de Vector, het grootste element achteraan. Bij het gegeven voorbeeld is de inhoud van de vector ["aa","dd","gg","hh","jj","mm","pp"].

Methode vectorToTree bouwt uit de Vector een nieuwe gebalanceerde boom op.

Bij het gegeven voorbeeld is de boom na reconstructie:



Gevraagd wordt om de implementatie van een aantal methoden. Mocht je bij de implementatie gebruik maken van een of meer hulpmethoden, geef dan van elke hulpmethode een volledige implementatie.

a. Geef de implementatie van methode insert(). **Geef daarna aan hoe deze methode wordt aangeroepen door methode insert() uit Bijlage 2.**

```
+insert(Tnode n, Object e)
pre: de boom is een binaire zoekboom
post: e is aan de boom met wortel n toegevoegd, de boom is een
binaire zoekboom
```

b. Geef de implementatie van methode treeToVector(). **Geef daarna aan hoe deze methode wordt aangeroepen door methode treeToVector() uit Bijlage 2.**

```
+treeToVector(TNode n, Vector v)
post: de elementen van de (sub)boom met wortel n, zijn in oplopende
grootte aan de vector toegevoegd
```

c. Geef de implementatie van methode vectorToTree(). **Geef daarna aan hoe de methode wordt aangeroepen door methode vectorToTree() uit Bijlage 2.**

```
+vectorToTree(Vector v, int og, int bg) : TNode
post: retourneert de wortel van een gebalanceerde zoekboom die de
elementen uit v bevat van ondergrens og t/m bovengrens bg
```

d. Geef de tijdcomplexiteit van methode vectorToTree. Licht je antwoord toe.

Bijlage 1.

```
public class LNode{

      private Object element;
      private LNode  next;

      public LNode(Object e, LNode n){
            setElement(e);
            setNext(n);
      }

      public void setElement(Object e)     {     element = e;      }
      public void setNext(LNode n)         {     next = n;         }
      public Object getElement()           {     return element;   }
      public LNode getNext()               {     return next;      }
}


public class List{

      LNode header;

      public List(){
            header = new LNode(null,null);
      }

      //niet alle methoden zoals bijv. addFirst of addAfter
      //zijn hier vermeld

      public void addBefore(LNode n, Object e)  {     ...     }
      public Object removeAfter(LNode n)        {  ...        }
      public boolean contains(Object e)         {  ...        }
      public int size()                         {  ...        }
}
```

Bijlage 2.

```
public class TNode{

      private String element;
      private TNode  left;
      private TNode  right;

      public TNode (String e, TNode l, TNode r){
            setElement(e);
            setLeft(l);
            setRight(r);
      }

      public boolean isExternal(){
            return (getLeft() == null && getRight() == null);
      }

      public String getElement()          {     return element;   }
      public TNode getLeft()              {     return left;      }
      public TNode getRight()             {     return right;     }

      public void setElement(String e)    {     element = e;      }
      public void setLeft(TNode l)        {     left = l;         }
      public void setRight(TNode r)       {     right = r;        }
}


public class BSTree{

      private TNode root;

      public BSTree(){
            root = new TNode(null,null,null);
      }

      public void insert(Object e)        { ...   }
      public Vector treeToVector()        { ...   }
      public void vectorToTree(v : Vector){ ...   }
}
```