DHBW Lörrach
Duale Hochschule
Baden-Württemberg

Algorithmen und Komplexität
TIF 21A/B
Dr. Bruno Becker

Übungsblatt 1: Einführung und ADT

Übungsblatt 1 –Aufgabe 1

Formulieren Sie einen rekursiven Algorithmus für n! (n Fakultät). Wie oft wird die Funktion für n >0 aufgerufen?

```
public static int fakultaet (int n)
{
        if n < 0 then return – 1; //Fehler Fakultät nicht definiert
        else  if n==0 return 1; // 0! = 1;
              else return n* fakultaet (n-1); // n! = n* (n-1)!
}
```

Für n=1 wird Funktion zweimal aufgerufen, für n=2 3x

➔ n+1 mal.

Übungsblatt 1 –Aufgabe 2

Gegeben sei ein Array a von n positiven ganzen Zahlen a[0],…,a[n-1] und eine Funktion g(x), die für eine positive ganze Zahl x den Wert 1 liefert, falls x gerade ist und 0 sonst. Analysieren Sie bitte die folgende Java Methode:

```
public static int gtest (int li, int re)
{
        if li > re return 0;
        else
        {
                int m = (li + re ) / 2;  // Ganzzahlige Division
                int gt = gtest (li,m-1) + g(a[m]) + gtest (m+1,re);
                return gt;
        }
}
```

a) Welches Resultat liefert die Methode beim Aufruf gtest (0, n-1)?

gtest (0,n-1) = $\sum_{i=0}^{n-1} g(a[i]) = \#\boldsymbol{Anzahl\ der\ geraden\ Zahlen\ in\ a}$

b) Wieviele Additionen werden beim Aufruf von gtest (li,re) benötigt? Geben Sie hierfür eine rekursive Formel in Abhängigkeit von re und li an.

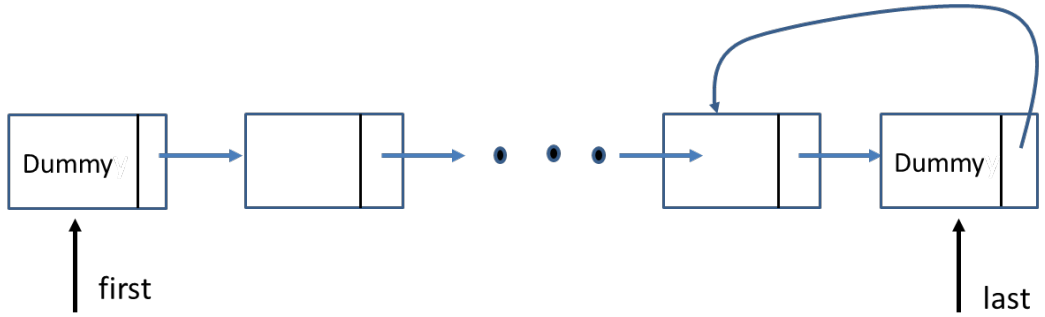
- re < li: 0 Additionen
- re = li: 3 Additionen
  re = li + 1: 3 + 3 Additionen
  Allgemein #Additionen = 3*(re-li+1)

c) Formulieren Sie ein alternatives, iteratives Verfahren zur Berechnung von gtest.

```
gtest = 0
for (int i=0; i < n; i++)
    gtest = gtest + g(a[i]);
```

Übungsblatt 1 –Aufgabe 3 a

Gegeben sei eine nicht leere verkettete lineare Liste *L* ganzer Zahlen mit ungerader Elementanzahl.



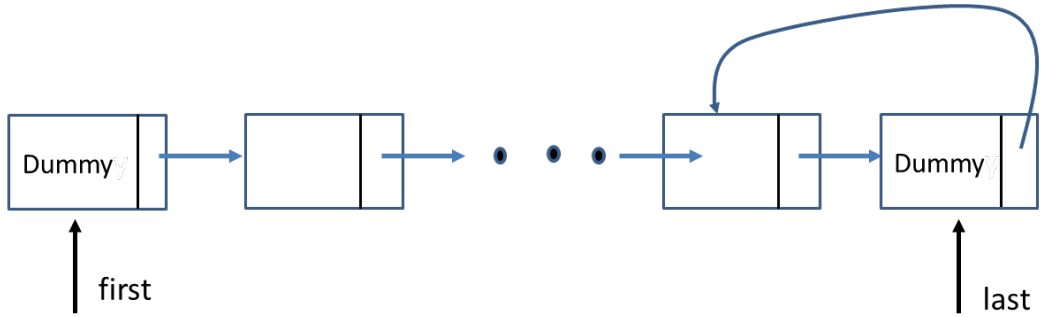
Schreiben Sie eine Methode *mitte* mit Inputparametern *first* und *last,* die das mittlere Element in der Liste entfernt.

Idee:

- Laufe mit 2 Zeigern durch Liste, einer in 2er- der andere in 1er-Schritten
- Wenn der 2er-Zeiger = Last ist, dann steht der 1er-Zeiger vor dem zu löschenden Element

```
Public void mitte (first, last node)
{  private node onestep = first;
   private node twostep = first.next.next;
   while ( twostep != last and twostep.next != last)
   {
       onestep = onestep.next;
       twostep = twostep.next.next;
   } //Jetzt steht onestep vor dem zu löschenden Element
    twostep = onestep.next; // twostep zeigt auf zu löschendes Element
    onestep.next = twostep.next; // löscht Verweis auf Element
   twostep.next = null;
   }
```

Übungsblatt 1 –Aufgabe 3 b

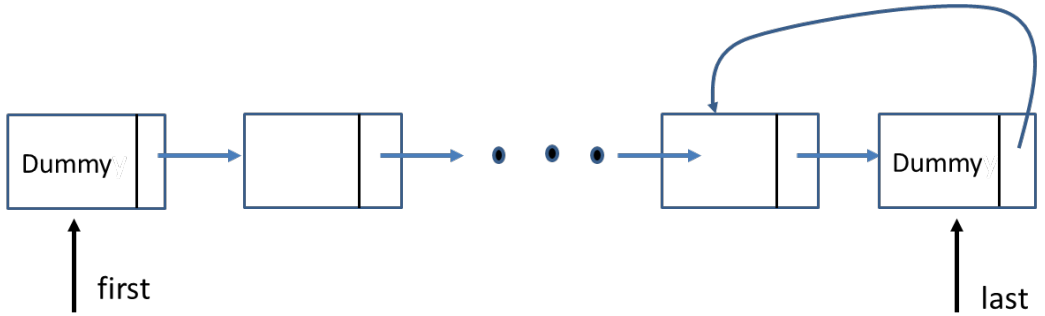


Schreiben Sie eine Methode *teilen* mit Inputparametern *first, firsteven, firstodd,* die die Liste L mit Anfangszeiger *first* aufteilt in zwei (anfangs leere, also durch zwei Dummy-Elemente gegebene) Listen mit Anfangszeiger *firsteven* bzw. *firstodd.* In die eine Liste sollen die Elemente aus L mit geradzahligen Elementen gehängt werden, in die andere Liste die Elemente aus L mit ungeradzahligen Elementen. Aufrufe von **new** sind **nicht** erlaubt.

Idee:

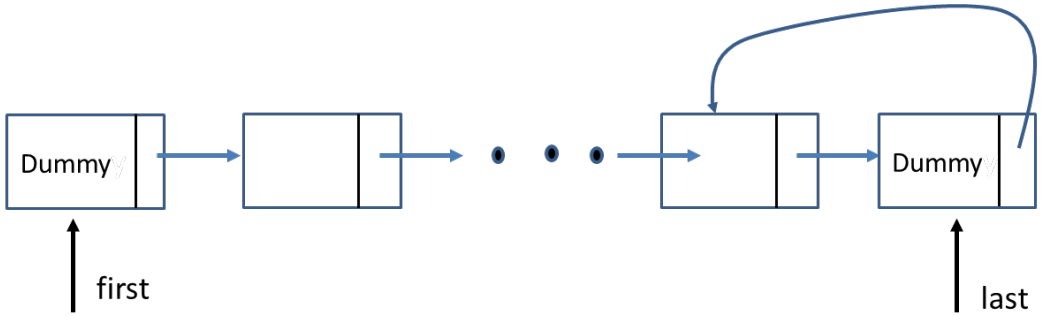
- Überprüfe immer 1. Element in L, bis Liste leer
- Falls Element.key gerade ist, füge es in Firsteven-Liste, sonst in Firstodd-Liste

Übungsblatt 1 –Aufgabe 3 b




```
Public void teilen (first, firsteven, firstodd node)
{
 private node help = first.next;  // help zeigt auf das zu löschende Element
  while (help != last)
  {
            first.next = help.next;  // Element aus L löschen
            if help.key % 2 == 0  // Element gerade
            {
                help.next = firsteven.next; // Element in gerade Liste
                firsteven.next = help;
            }
           else
           {
               help.next = firstodd.next; // Element in ungerade Liste
               firstodd.next = help;
           }
            help = first.next;  // help zeigt auf das nächste zu löschende Element
   }
}
```

Übungsblatt 1 –Aufgabe 3 c



Schreiben Sie eine Methode *umdrehen* mit Inputparametern *first* und *last,* die die Reihenfolge der Elemente, d.h. die Zeiger in der Liste, umdreht. Keine zweite Liste erstellen, d.h. Aufrufe von **new** sind **nicht** erlaubt.

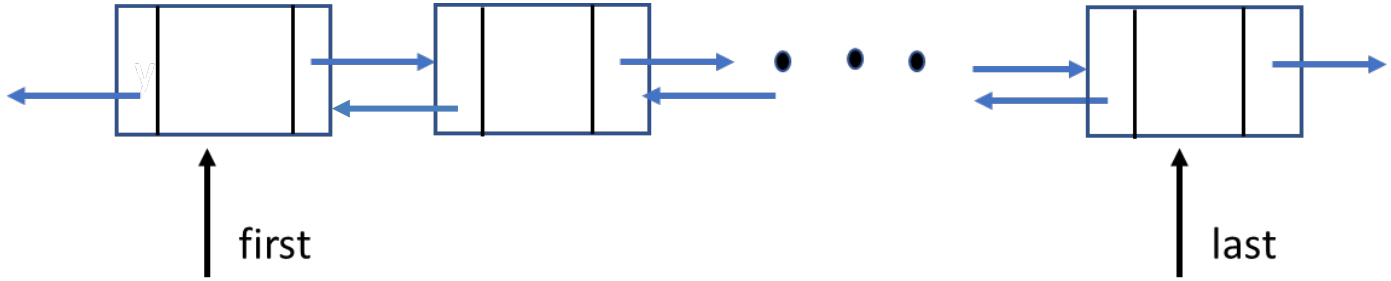
Idee:

- Lösche immer das 1. Element der Liste und füge es hinter dem (anfangs) letzten Element der Liste ein.
- Stoppen, sobald first.next auf das anfangs letzte Element der Liste ein

```
Public void umdrehen (first, last node)
{
        private node hz1 = first.next;  // hz1 zeigt auf das 1. umzuhängende Elem.
        private node hz2 = last.next;  // hz2 zeigt auf das anfangs letzte Element
       while (hz1.next != hz2) // Solange hz1.next nicht auf hz2 zeigt
       {
       first.next = hz1.next; // Lösche Verweis von first auf  Element
       hz1.next = hz2.next;  // Element.next zeigt nun hz2.next
              hz2.next = hz1;  // der next-Zeiger von hz2  zeigt auf Element hz1
              hz1 = first.next; // hz1 zeigt auf das nächste umzuhängende Element
       }
}
```

Übungsblatt 1 –Aufgabe 4

Gegeben sei eine doppelt verkettete nichtleere Liste. Schreiben Sie eine Methode, die das Maximum in der Liste (d.h. das Element mit dem größten Schlüsselwert) sucht, aus der Liste entfernt und den maximalen Schlüsselwert zurückgibt.

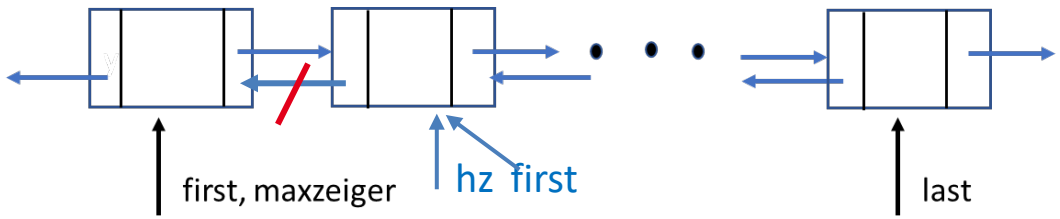


Idee:

- Suche in einem Durchlauf das maximale Element, *maxzeiger* zeigt auf dieses Element und *maxvalue* ist der maximale Wert
- Sonderfälle beachten
  - Einziges Element aus Liste löschen
  - Erstes Element löschen
  - Letztes Element löschen

```
Public void maxsuchenlöschen (first, last node)
{       private node hz = first;  // hz Hilfszeiger zum Durchlaufen
        private node maxzeiger = first;  // Zeigt nach Durchlauf auf Maximum
       int maxvalue = maxzeiger.value;
       while (hz != last) // Solange hz nicht auf letzes Element zeigt
       {  hz = hz.next;
          if  (hz.value > maxvalue) // Neues Maximum gefunden
              {maxzeiger= hz;
              maxvalue = maxzeiger.value}
       } // jetzt steht maxzeiger auf Maximalem Element
```

Übungsblatt 1 –Aufgabe 4




```
if (maxzeiger = first) // Erstes Element löschen
   {  if (maxzeiger = last) // Maximum ist das einzige Element -> Liste nach  Löschen leer
       {  first = null;
          last = null;
       }
      else
      { // Erstes Element löschen, Liste danach nicht leer
         hz = maxzeiger.next;
         hz.prev = null;
         first = first.next;
      }
    }
```

Übungsblatt 1 –Aufgabe 4

```
else if  (maxzeiger = last) // Letztes Element löschen
    {
       hz = maxzeiger.prev;
       hz.next = null;
       last = hz;
    }
else
    { // Maximum nicht am Anfang oder Ende
            hz = maxzeiger.prev;
            hz.next = maxzeiger.next;
            maxzeiger.next.prev = hz;
    }
return maxvalue
}
```