Script generated by TTT

Title: Seidl: Informatik_1 (26.11.2012)

Date: Mon Nov 26 18:14:15 CET 2012

Duration: 88:15 min

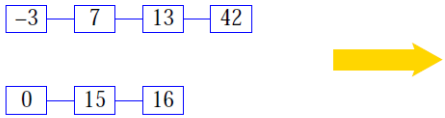
Pages: 41

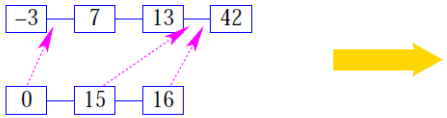
Anwendung: Mergesort – Sortieren durch Mischen

Mischen:

Eingabe: zwei sortierte Listen;

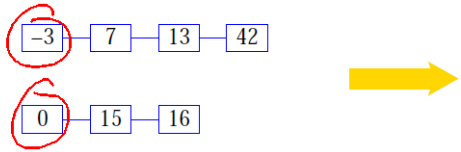
Ausgabe: eine gemeinsame sortierte Liste.

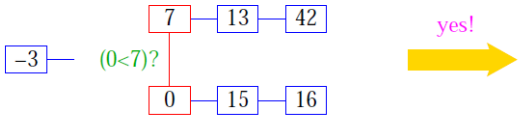


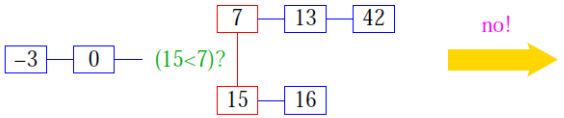


Idee:

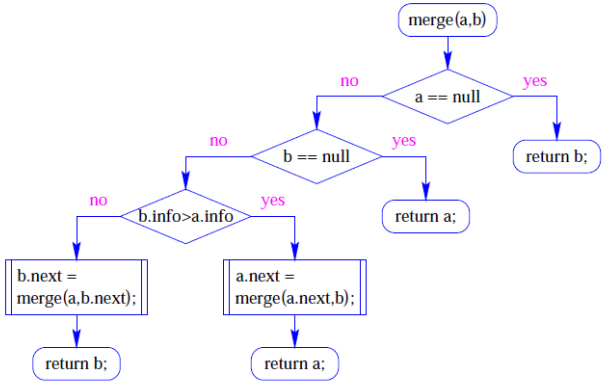
- Konstruiere sukzessive die Ausgabe-Liste aus den der Argument-Listen.
- Um das nächste Element für die Ausgabe zu finden, vergleichen wir die beiden kleinsten Elemente der noch verbliebenen Input-Listen.
- Falls die n die Länge der längeren Liste ist, sind offenbar maximal nur $n-1$ Vergleiche nötig.





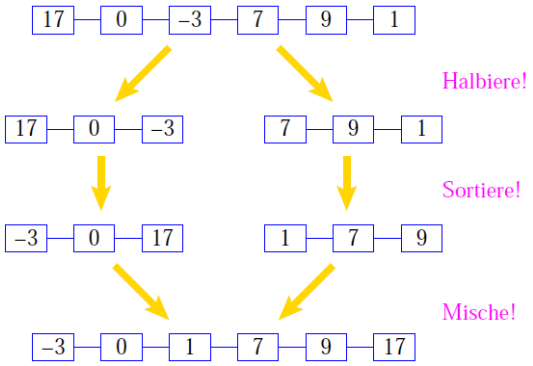


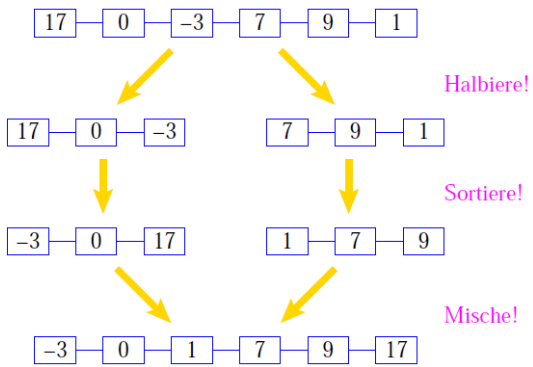
```
public static List merge(List a, List b) {
   if (b == null)
      return a;
   if (a == null)
      return b;
   if (b.info > a.info) {
      a.next = merge(a.next, b);
      return a;
   } else {
      b.next = merge(a, b.next);
      return b;
      }
   }
```

Sortieren durch Mischen:

- Teile zu sortierende Liste in zwei Teil-Listen;
- sortiere jede Hälfte für sich;
- mische die Ergebnisse!

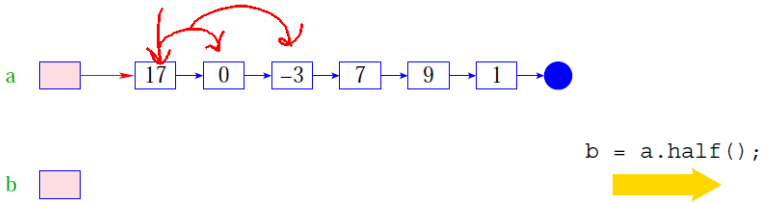


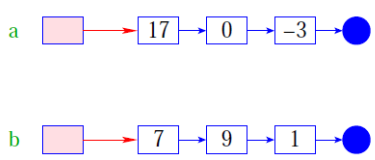



```
public static List sort(List a) {
   if (a == null || a.next == null)
      return a;
   List b = a.half(); // Halbiere!
   a = sort(a);
   b = sort(b);
   return merge(a,b);
   }
```



```
public List half() {
   int n = length();
   List t = this;
   for(int i=0; i<n/2-1; i++)
      t = t.next;
   List result = t.next;
   t.next = null;
   return result;
   }
```



Diskussion:

- Sei $V(n)$ die Anzahl der Vergleiche, die Mergesort maximal zum Sortieren einer Liste der Länge n benötigt.

  Dann gilt:

$$\begin{aligned} V(1) &= 0 \\ V(2n) &\leq 2 \cdot V(n) + 2 \cdot n \end{aligned}$$

- Für $n = 2^k$, sind das dann nur $k \cdot n$ Vergleiche !!!

Achtung:

- Unsere Funktion `sort()` zerstört ihr Argument ?!
- Alle Listen-Knoten der Eingabe werden weiterverwendet.
- Die Idee des Sortierens durch Mischen könnte auch mithilfe von Feldern realisiert werden. (wie ?)
- Sowohl das Mischen wie das Sortieren könnte man statt rekursiv auch iterativ impliementieren (wie ?)

Achtung:

- Unsere Funktion `sort()` zerstört ihr Argument ?!
- Alle Listen-Knoten der Eingabe werden weiterverwendet.
- Die Idee des Sortierens durch Mischen könnte auch mithilfe von Feldern realisiert werden. (wie ?)
- Sowohl das Mischen wie das Sortieren könnte man statt rekursiv auch iterativ impliementieren (wie ?)

399

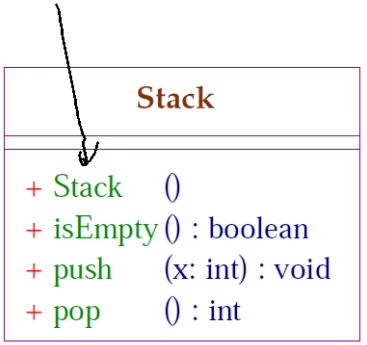
11.2 Keller (Stacks)

Operationen:

`boolean isEmpty()`	:	testet auf Leerheit;
`int pop()`	:	liefert oberstes Element;
`void push(int x)`	:	legt x oben auf dem Keller ab;
`String toString()`	:	liefert eine String-Darstellung.

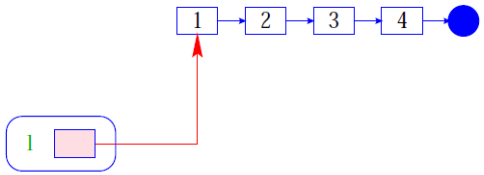
Weiterhin müssen wir einen leeren Keller anlegen können.

Modellierung:



Erste Idee:

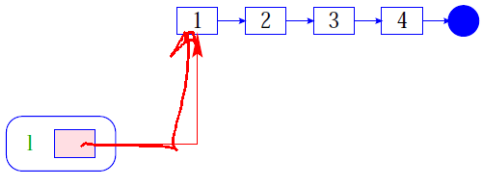
- Realisiere Keller mithilfe einer Liste!



- Das Attribut `l` zeigt auf das oberste Element.

Erste Idee:

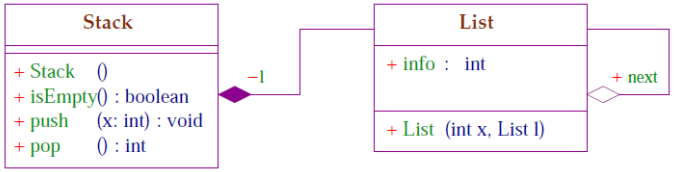
- Realisiere Keller mithilfe einer Liste!



- Das Attribut `l` zeigt auf das oberste Element.

402

Modellierung:



Die gefüllte Raute besagt, dass die Liste nur von `Stack` aus zugreifbar ist.

Implementierung:

```
public class Stack {
   private List l;
   // Konstruktor:
   public Stack() {
      l = null;
      }
   // Objekt-Methoden:
   public isEmpty() {
      return List.isEmpty(l);
      }
   ...
```



```
public int pop() {
   int result = l.info;
   l = l.next;
   return result;
   }
public void push(int a) {
   l = new List(a,l);
   }
public String toString() {
   return List.toString(l);
   }
} // end of class Stack
```


- Die Implementierung ist sehr einfach;
- ... nutzte gar nicht alle Features von `List` aus;
- ... die Listen-Elemente sind evt. über den gesamten Speicher verstreut;

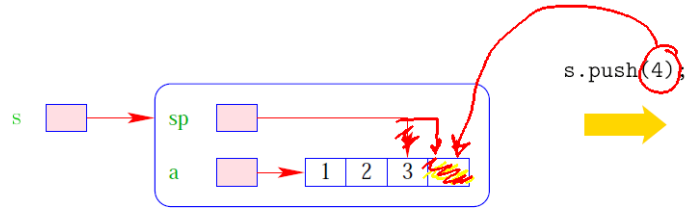
  $\Longrightarrow$ führt zu schlechtem ↑Cache-Verhalten des Programms !

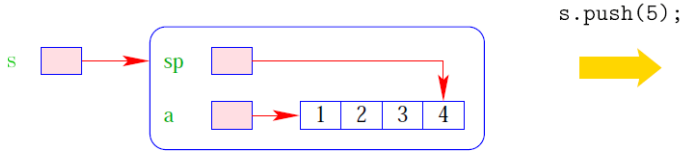
- Die Implementierung ist sehr einfach;
- ... nutzte gar nicht alle Features von `List` aus;
- ... die Listen-Elemente sind evt. über den gesamten Speicher verstreut;

  $\Longrightarrow$ führt zu schlechtem ↑Cache-Verhalten des Programms !

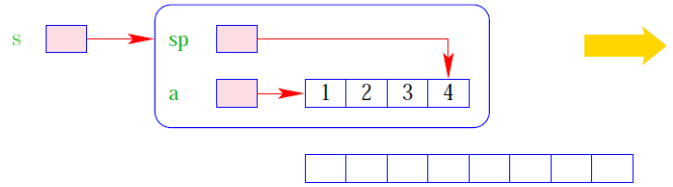
Zweite Idee:

- Realisiere den Keller mithilfe eines Felds und eines Stackpointers, der auf die oberste belegte Zelle zeigt.
- Läuft das Feld über, ersetzen wir es durch ein größeres.

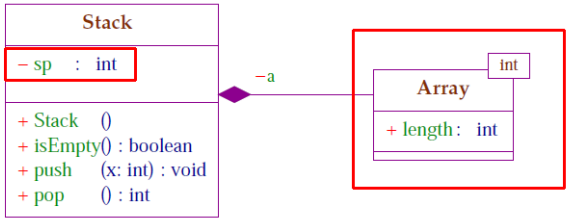




`s.push(5);`



Modellierung:



Implementierung:

```
public class Stack {
   private int sp;
   private int[] a;
// Konstruktoren:
   public Stack() {
      sp = -1; a = new int[4];
      }
// Objekt-Methoden:
   public boolean isEmpty() {
      return (sp<0);
      }
   ...
```



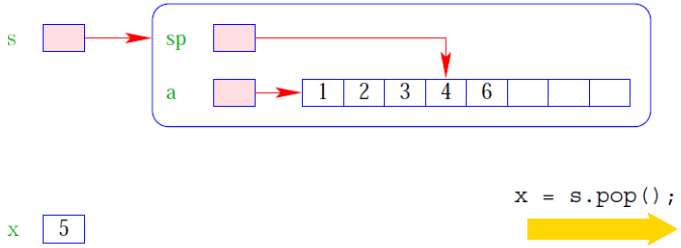
```
   public int pop() {
      return a[sp--];
      }
   public void push(int x) {
      ++sp;
      if (sp == a.length) {
        int[] b = new int[2*sp];
        for(int i=0; i<sp; ++i) b[i] = a[i];
        a = b;
        }
      a[sp] = x;
      }
   public toString() {...}
} // end of class Stack
```


Nachteil:

- Es wird zwar neuer Platz allokiert, aber nie welcher freigegeben.

Erste Idee:

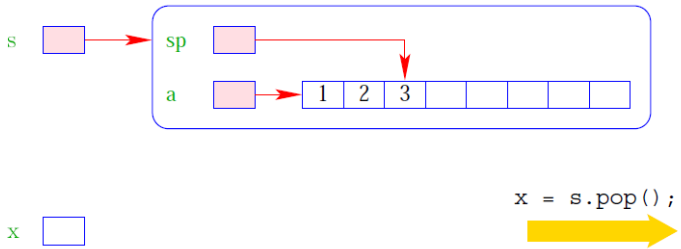
- Sinkt der Pegel wieder auf die Hälfte, geben wir diese frei ...



- Im schlimmsten Fall müssen bei jeder Operation sämtliche Elemente kopiert werden.

Zweite Idee:

- Wir geben erst frei, wenn der Pegel auf ein Viertel fällt – und dann auch nur die Hälfte !



- Vor jedem Kopieren werden mindestens halb so viele Operationen ausgeführt, wie Elemente kopiert werden.
- Gemittelt über die gesamte Folge von Operationen werden pro Operation maximal zwei Zahlen kopiert ↑amortisierte Aufwandsanalyse.

```
public int pop() {
   int result = a[sp];
   if (sp == a.length/4 && sp>=2) {
      int[] b = new int[2*sp];
      for(int i=0; i < sp; ++i)
         b[i] = a[i];
      a = b;
      }
   sp--;
   return result;
   }
```


11.3 Schlangen (Queues)

(Warte-) Schlangen verwalten ihre Elemente nach dem FIF0-Prinzip (First-In-First-Out).

Operationen:

`boolean isEmpty()`	:	testet auf Leerheit;
`int dequeue()`	:	liefert erstes Element;
`void enqueue(int x)`	:	reiht x in die Schlange ein;
`String toString()`	:	liefert eine String-Darstellung.

Weiterhin müssen wir eine leere Schlange anlegen können.