Datenstrukturen und Algorithmen

Übungsstunde 12.4.18

Nachbesprechung Serie 5

Allgemeines

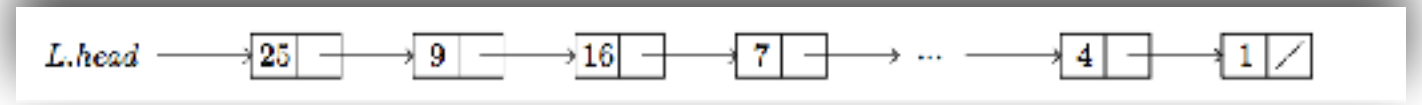
- Pseudocode = Pseudocode
  - Kein C, kein Java, kein sonstwas
  - Längeren Pseudocode (> 10 Zeilen) erklären, da es oft schwierig ist, eine Idee nachzuvollziehen
  - Arrays != List != Stack != Komische Selbsterfundene Sachen

- Schreibt keine „return“ in Pseudocode, ausser ihr wollt Programm wirklich abbrechen

Speziell zu Listen

- Linked Lists **sind keine Arrays**!
  - Liste[] / Liste[i] etc. ist **falsch**!
- An Notation von Buch/Vorlesung halten
  - Bei Queues **start / ende**, bei Linked Lists **head / tail**
    - L.head / L.tail
    - Head(L) / head[x] / head
- Randfälle auch betrachten (z.B. Liste leer; L.head = L.tail)
- Listen haben Zeiger —> müssen bei Veränderungen der Liste (Einfügen, löschen, neu verlinken etc.) betrachtet werden!

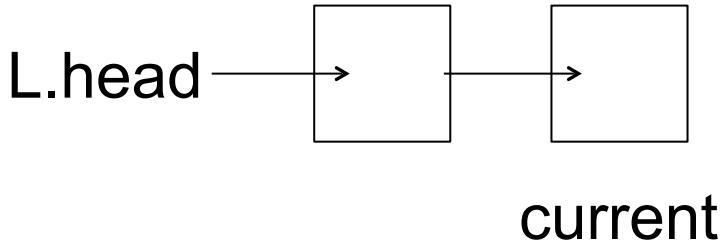
Aufgabe 1




```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

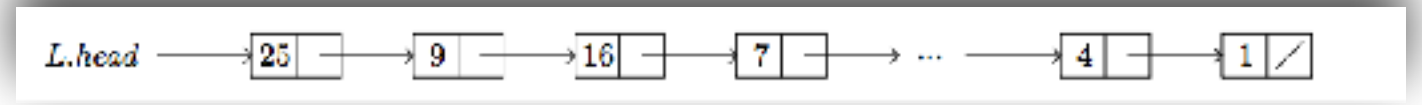
Aufgabe 1

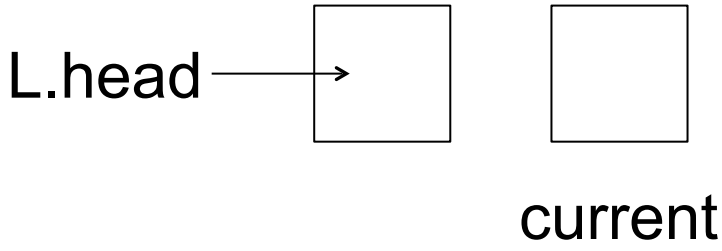
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```



Aufgabe 1

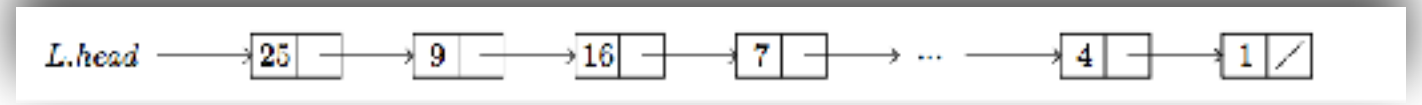
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

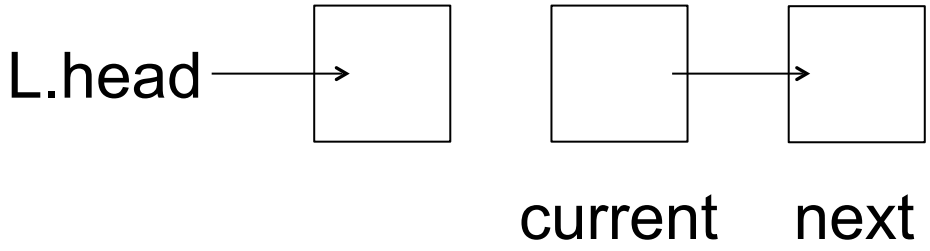




Aufgabe 1

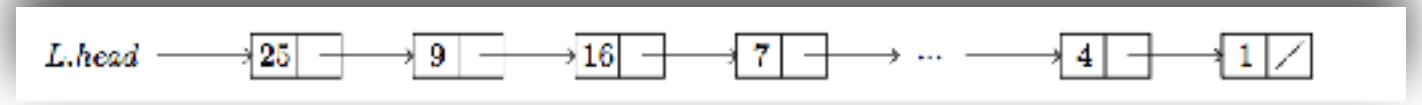
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

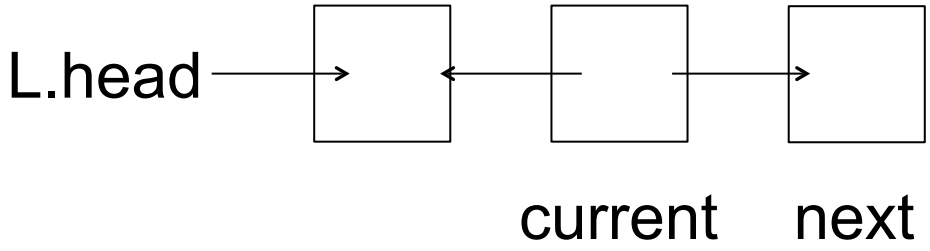




Aufgabe 1

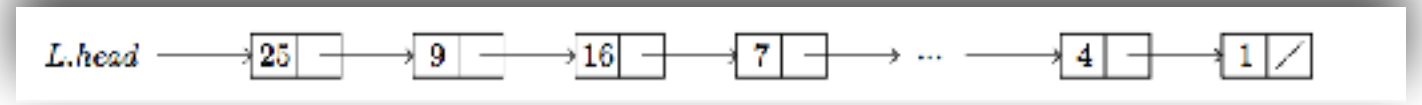
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

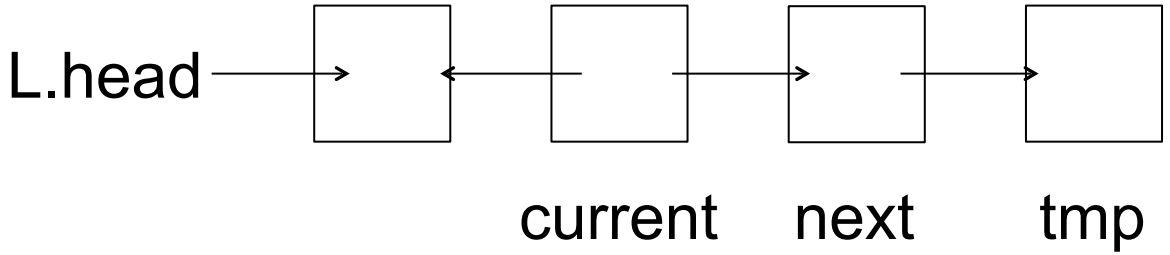




Aufgabe 1

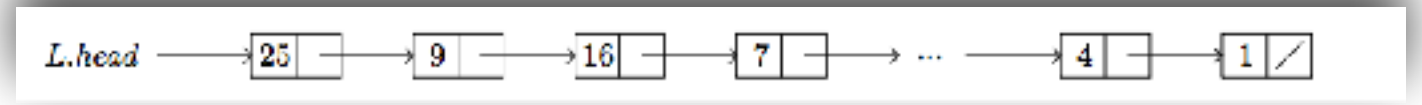
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

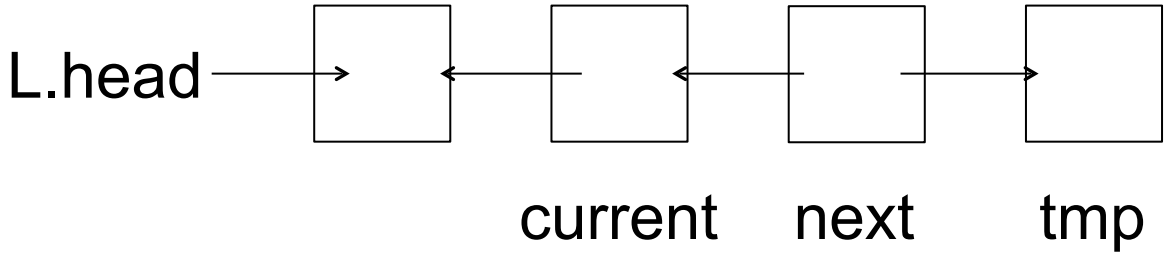




Aufgabe 1

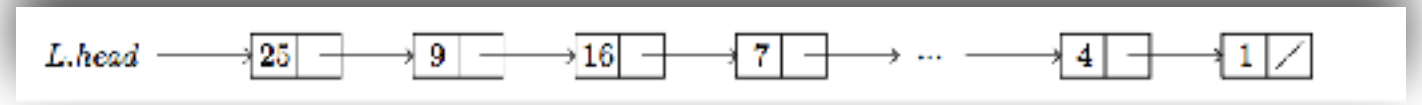
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

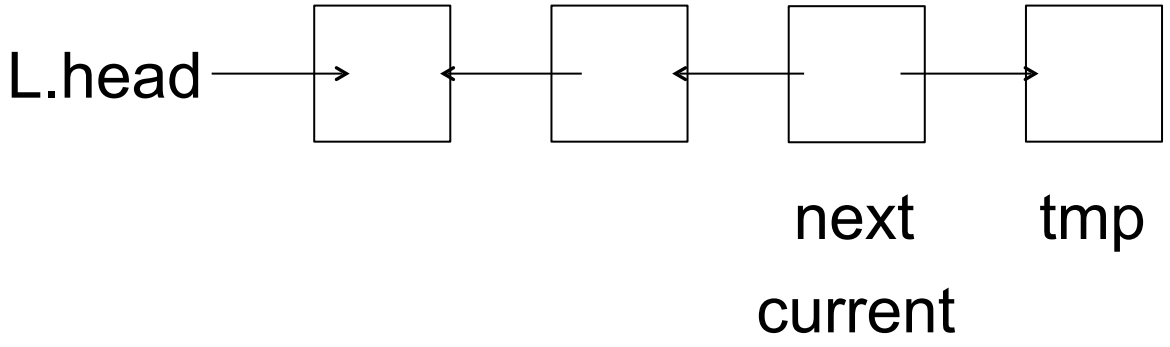




Aufgabe 1

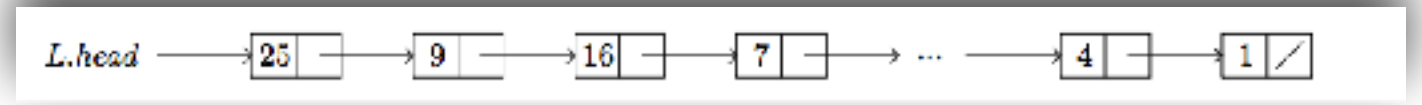
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```

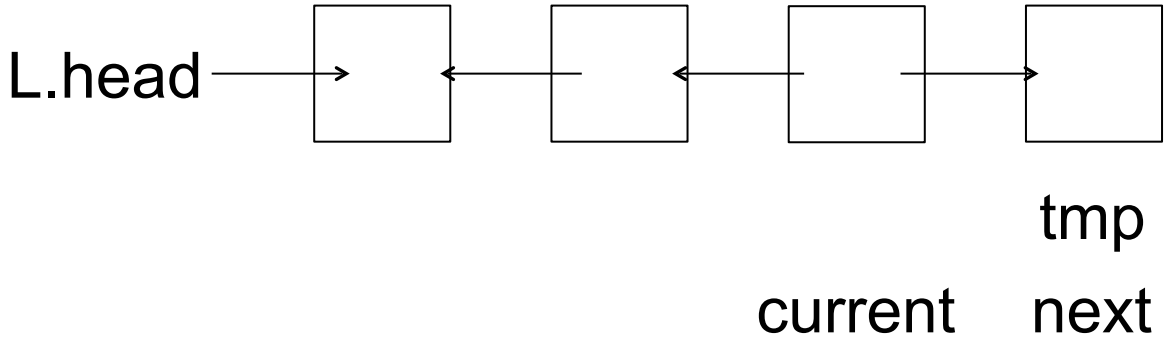




Aufgabe 1

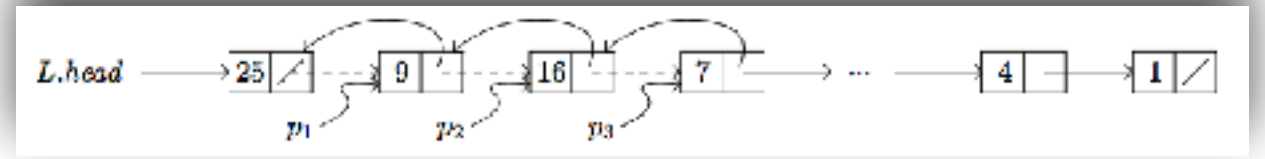
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```





Aufgabe 1

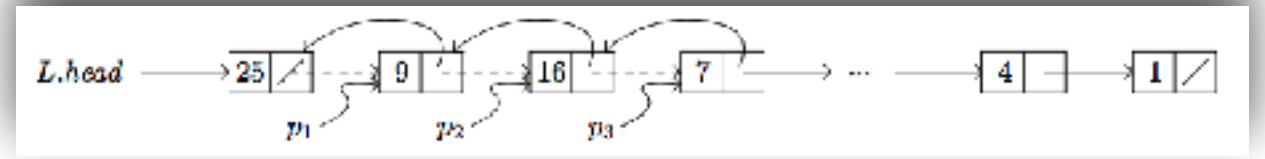
```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```



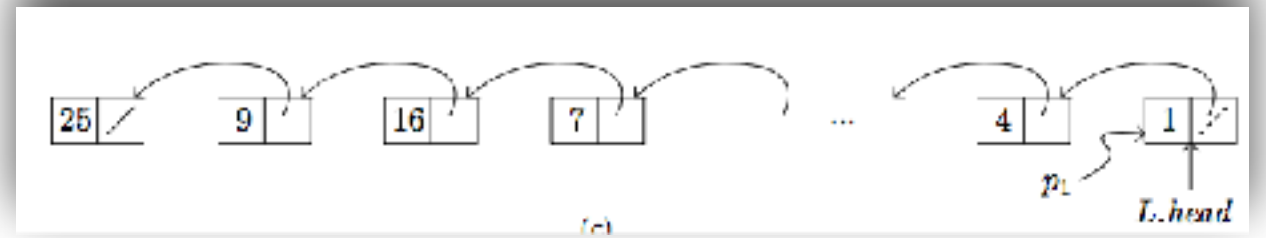
Nach 2. Iteration

Aufgabe 1

```
List-Reverse(L)
if L.head == NIL
    error
elseif L.head.next == NIL
    done
else
    current = L.head.next
    L.head.next = NIL
    next = current.next
    current.next = L.head
    while (next ≠ NIL)
        tmp = next.next
        next.next = current
        current = next
        next = tmp
    L.head = current
```



Nach 2. Iteration



Resultat

Aufgabe 2

```
ENQUEUE(Q, x)
  if QUEUE-EMPTY(Q)
    Q.head = x
  else
    Q.tail.next = x
    Q.tail = x
    x.next = NIL      // Zeiger muss auch aufgeschrieben werden!
```

Aufgabe 2

```
DEQUEUE(Q)
  if QUEUE-EMPTY(Q)
    error „underflow“
  else
    if Q.head == Q.tail
      Q.tail = NIL
    x = Q.head
    Q.head = Q.head.next
    return x
```

Aufgabe 3

- Schreiben Sie Pseudocode für eine **rekursive** Prozedur, die alle Knoten eines gerichteten Baumes mit unbeschränktem Grad besucht und jeweils den Schlüssel des Knotens ausgibt. Nehmen Sie an, die Knoten des Baumes hätten folgende Felder: key für den Schlüssel, left-child für den Zeiger auf das sich am weitesten links befindende Kind und right-sibling für den Zeiger auf das rechte Geschwister.

- Node:
  - Key
  - Left-child
  - Right-sibling

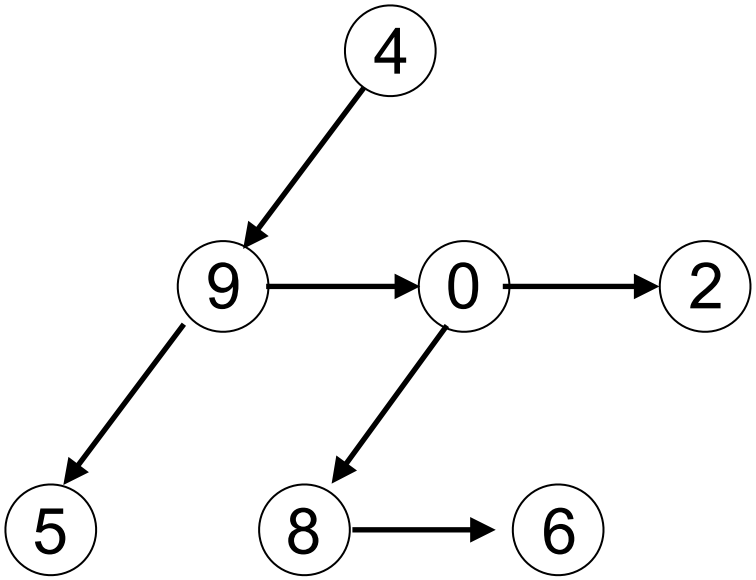
- Bitte haltet euch an die Notation!

Aufgabe 3

- Allgemeine Tipps
  - Beispiel aufschreiben und selber durchspielen
  - Algorithmus überlegen, der das Beispiel löst
  - Spezialfälle nicht vergessen!
  - Algorithmus testen

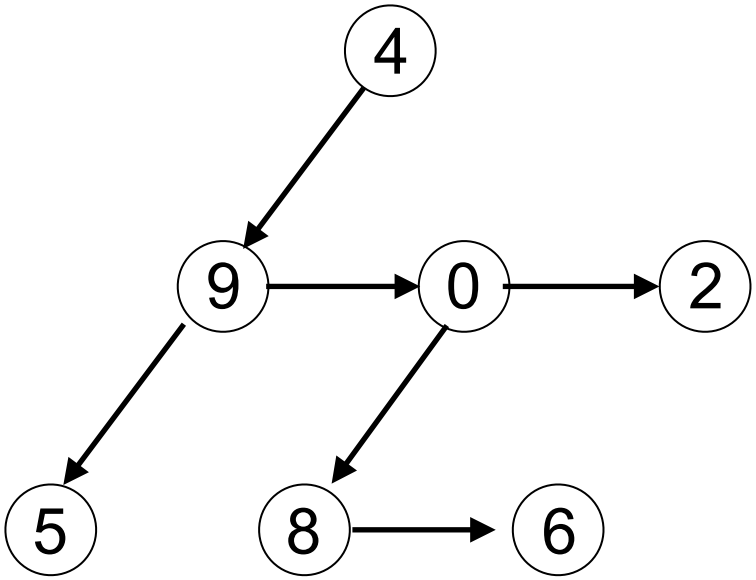
Aufgabe 3

- Vorgehen



Aufgabe 3

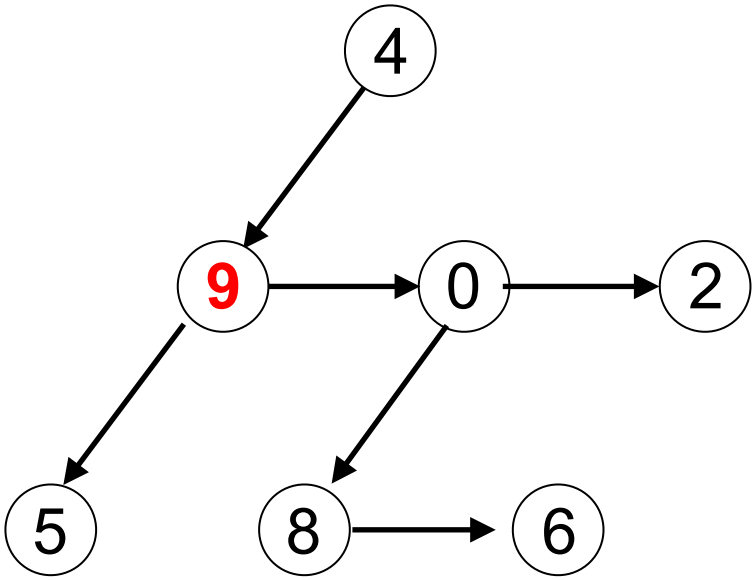
> Vorgehen



print(key)
  RECURSIVE-METHOD(left-child)
  RECURSIVE-METHOD(right-sibling)

Aufgabe 3

> Vorgehen



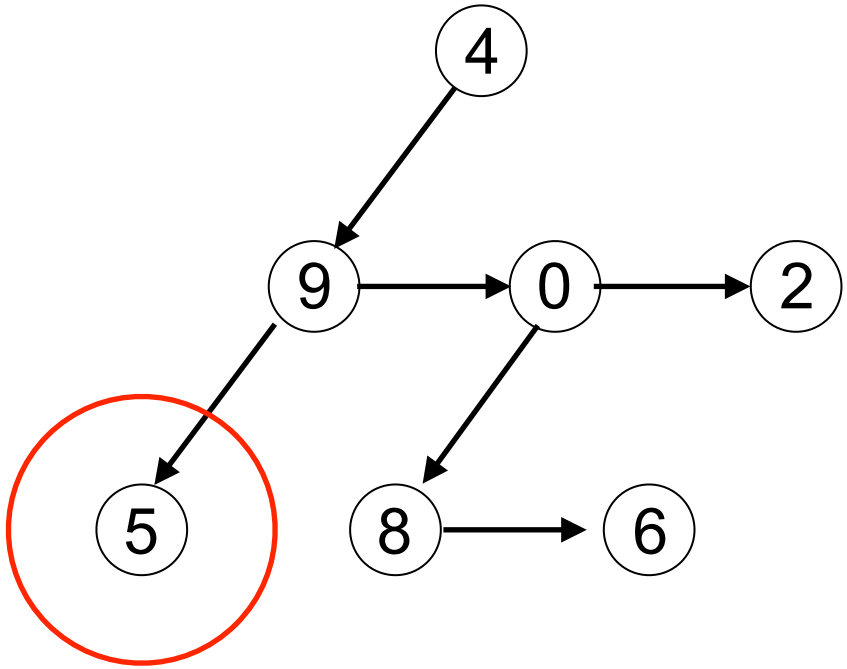
print(key)

RECURSIVE-METHOD(left-child)

RECURSIVE-METHOD(right-sibling)

Aufgabe 3

> Vorgehen



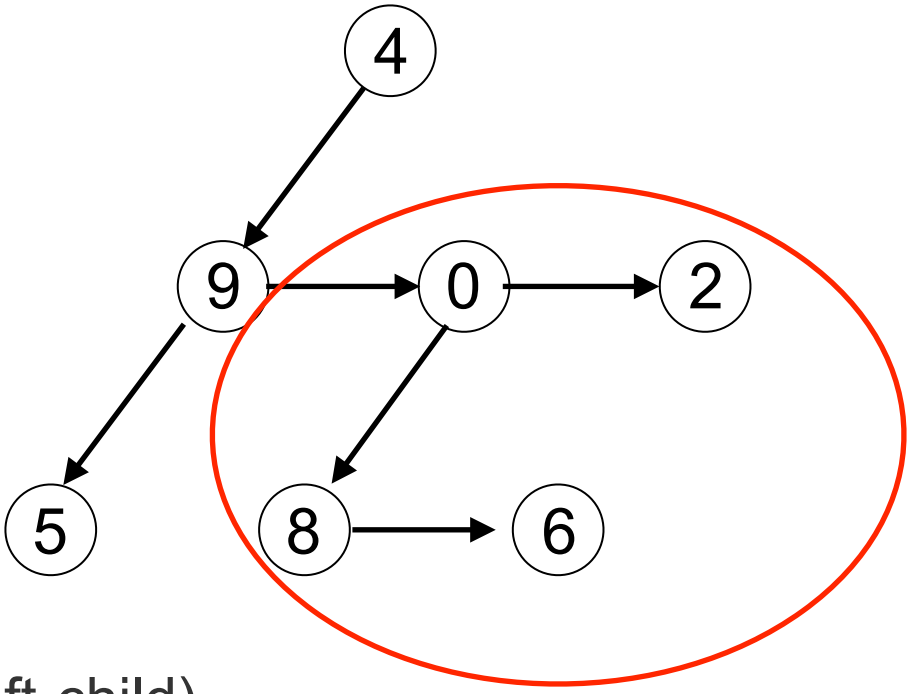
print(key)

RECURSIVE-METHOD(left-child)

RECURSIVE-METHOD(right-sibling)

Aufgabe 3

> Vorgehen



print(key)
  RECURSIVE-METHOD(left-child)
  RECURSIVE-METHOD(right-sibling)

Aufgabe 3

> Lösung

```
visit(T)
   if T ≠ NIL
      print T.key
      visit(T.left-child)
      visit(T.right-sibling)
```

Aufgabe 4

Selbes Vorgehen wie bei Aufgabe 3:

```
traverse(node)
   Stack s
   s.push node
   while (( current = s.pop) ≠ NIL )
      print current.key
      if (current.left-child ≠ NIL)
         s.push current.left-child
      if (current.right-sibling ≠ NIL
         s.push current.right-sibling
```

Aufgabe 5

- Geben Sie Pseudocode für eine Merge Methode an, die zwei sortierte einfach verkettete zyklische Listen als Parameter annimmt und diese in linearer Zeit zu einer einzelnen sortierten Liste zusammenfügt. Die ursprünglichen Listen dürfen dabei zerstört werden und es soll nur konstant viel zusätzlicher Speicher verwendet werden.
  Wieso ist die Zeitkomplexität quadratisch statt linear, wenn der Merge Pseudocode aus Kapitel 2, Seite 32 im Buch direkt verwendet wird, die Felder A,L,R aber durch verkettete Listen ersetzt werden?

- Idee: solange beide Listen Elemente haben immer das kleinere einfügen, wenn eine Liste leer ist den Rest rüber schaufeln

Aufgabe 5

> Detaillierte Lösung auf Ilias

Fragen?