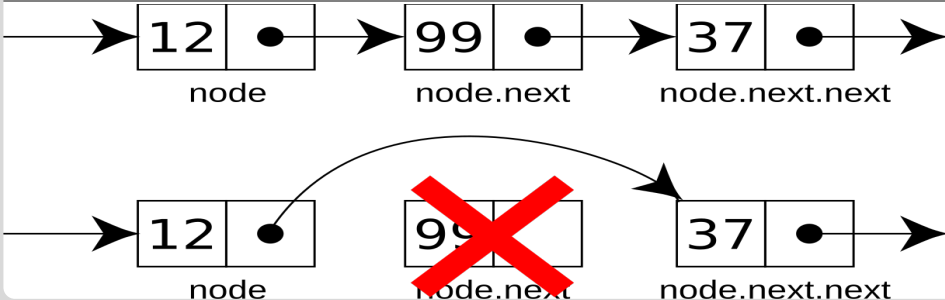


# Algorithmen I - Tutorium 3

Sebastian Schmidt – *isibboi@gmail.com*

Arbeitsgruppe Kryptographie und Sicherheit



Online unter: <https://github.com/ISibbol/Algol-Tut>

```

1: function merge( $A : \text{Array } [1..n_1] \text{ of } \mathbb{N}_{\geq 0}, B : \text{Array } [1..n_2] \text{ of } \mathbb{N}_{\geq 0}$ )
2: assert  $A[i] \leq A[j] \quad \forall i \leq j \text{ mit } i, j \in \{1, \dots, n_1\}$ 
3: assert  $B[i] \leq B[j] \quad \forall i \leq j \text{ mit } i, j \in \{1, \dots, n_2\}$ 
4:  $A[n_1 + 1] := \infty, B[n_2 + 1] := \infty$ 
5:  $n := n_1 + n_2$ 
6:  $j_A := 1, j_B := 1$ ;
7: for  $i := 1$  to  $n$  do
8:    $C[i] = \min(A[j_A], B[j_B])$ 
9:   if  $A[j_A] < B[j_B]$  then
10:      $j_A = j_A + 1$ 
11:   else
12:      $j_B = j_B + 1$ 
13:   invariant  $C[1..i]$  enthält genau  $A[1..j_A - 1], B[1..j_B - 1]$ 
14:   invariant  $B[k] \leq A[j_A] \quad \forall k \in \{1..j_B - 1\}, A[k] \leq B[j_B] \quad \forall k \in \{1..j_A - 1\}$ 
15:   invariant  $C[1..i]$  ist sortiert
16: assert  $j_A = n_1 + 1, j_B = n_2 + 1$ 
17: assert  $C[i] \leq C[j] \quad \forall i \leq j, i, j \in \{1, \dots, n\}$ 
18: assert  $C[1..n]$  enthält genau  $A[1..n_1], B[1..n_2]$ 
19: return  $C$ 

```

- Aufgabe 1.a.3) Formal korrekt  
 $n^2 \log n \in O(n^3)$

- Aufgabe 1.c) Hatte niemand richtig

$$\forall f, g : \mathbb{N} \rightarrow \mathbb{N} : f(n) \in O(g(n)) \Rightarrow 2^{f(n)} \in O(2^{g(n)})$$

- Aufgabe 1.a.3) Formal korrekt  
 $n^2 \log n \in O(n^3)$
- Aufgabe 1.c) Hatte niemand richtig  
 $\forall f, g : \mathbb{N} \rightarrow \mathbb{N} : f(n) \in O(g(n)) \Rightarrow 2^{f(n)} \in O(2^{g(n)})$

- Einfach verkettete Liste
- Doppelt verkettete Liste
- Unbounded Array
- Unbounded queue mit doppelt verketteter Liste
  - Direkt
  - Mit splice

- Einfach verkettete Liste
- Doppelt verkettete Liste
- Unbounded Array
- Unbounded queue mit doppelt verketteter Liste
  - Direkt
  - Mit splice

- Einfach verkettete Liste
- Doppelt verkettete Liste
- Unbounded Array
- Unbounded queue mit doppelt verketteter Liste
  - Direkt
  - Mit splice



- Einfach verkettete Liste
- Doppelt verkettete Liste
- Unbounded Array
- Unbounded queue mit doppelt verketteter Liste
  - Direkt
  - Mit splice

**Procedure** splice( $a, b, t : \text{Handle}$ ) // Cut out  $\langle a, \dots, b \rangle$  and insert after  $t$   
**assert**  $b$  is not before  $a \wedge t \notin \langle a, \dots, b \rangle$

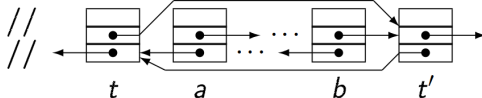
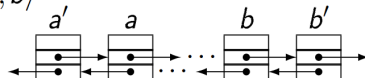
// Cut out  $\langle a, \dots, b \rangle$

$a' := a \rightarrow \text{prev}$

$b' := b \rightarrow \text{next}$

$a' \rightarrow \text{next} := b'$

$b' \rightarrow \text{prev} := a'$



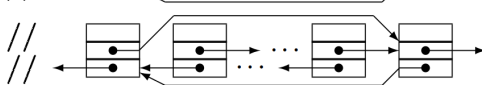
// insert  $\langle a, \dots, b \rangle$  after  $t$

$t' := t \rightarrow \text{next}$



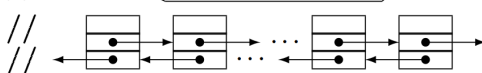
$b \rightarrow \text{next} := t'$

$a \rightarrow \text{prev} := t$



$t \rightarrow \text{next} := a$

$t' \rightarrow \text{prev} := b$



# Aufgabe 1.a

Entwickelt eine Datenstruktur, die folgendes kann:

- `pushBack` und `popBack` in  $O(1)$  Zeit im Worst-Case
- Wahlfreier Zugriff in  $O(\log n)$  Zeit im Worst-Case

Nicht amortisiert!

Speicherallokation geht hier in konstanter Zeit.

# Aufgabe 1.b

Entwickelt eine Datenstruktur, die folgendes kann:

- `pushBack` und `popBack` in  $O(\log n)$  Zeit im Worst-Case
- Wahlfreier Zugriff in  $O(1)$  Zeit im Worst-Case

Nicht amortisiert!

Speicherallokation geht hier in konstanter Zeit.

# Aufgabe 2

Was passiert, wenn man `splice(a, b, t : Handle)`  
fälschlicherweise mit  $t \in \langle a, \dots, b \rangle$  aufruft?

# splice

**Procedure** splice( $a, b, t : \text{Handle}$ ) // Cut out  $\langle a, \dots, b \rangle$  and insert after  $t$   
**assert**  $b$  is not before  $a \wedge t \notin \langle a, \dots, b \rangle$

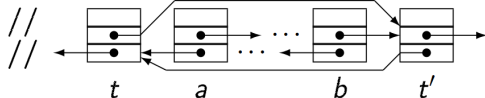
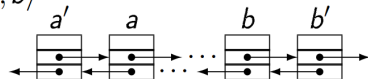
// Cut out  $\langle a, \dots, b \rangle$

$a' := a \rightarrow \text{prev}$

$b' := b \rightarrow \text{next}$

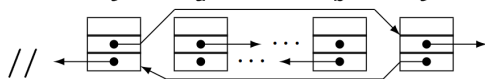
$a' \rightarrow \text{next} := b'$

$b' \rightarrow \text{prev} := a'$



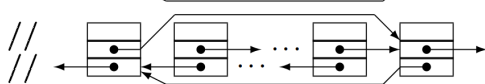
// insert  $\langle a, \dots, b \rangle$  after  $t$

$t' := t \rightarrow \text{next}$



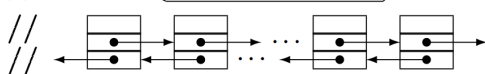
$b \rightarrow \text{next} := t'$

$a \rightarrow \text{prev} := t$



$t \rightarrow \text{next} := a$

$t' \rightarrow \text{prev} := b$



Gegeben ist ein Array  $A = A[1], \dots, A[n]$  mit  $n$  Zahlen in beliebiger Reihenfolge.

Suche für eine gegebene Zahl  $x$  ein Paar  $(A[i], A[j]), 1 \leq i, j, \leq n$  mit  $A[i] + A[j] = x$ .

- Beispiel: Gebt eine Lösung für  $x = 33$  und  $A = (7, 15, 21, 14, 18, 3, 9)$  an.
- Gebt einen Algorithmus an, der das Problem in erwarteter Zeit  $O(n)$  löst, und bei Erfolg ein Paar  $(A[i], A[j])$  ausgibt, ansonsten *NIL*.

Gegeben ist ein Array  $A = A[1], \dots, A[n]$  mit  $n$  Zahlen in beliebiger Reihenfolge.

Suche für eine gegebene Zahl  $x$  ein Paar  $(A[i], A[j]), 1 \leq i, j, \leq n$  mit  $A[i] + A[j] = x$ .

- Beispiel: Gebt eine Lösung für  $x = 33$  und  $A = (7, 15, 21, 14, 18, 3, 9)$  an.
- Gebt einen Algorithmus an, der das Problem in erwarteter Zeit  $O(n)$  löst, und bei Erfolg ein Paar  $(A[i], A[j])$  ausgibt, ansonsten *NIL*.



Gegeben ist ein Array  $A = A[1], \dots, A[n]$  mit  $n$  Zahlen in beliebiger Reihenfolge.

Suche für eine gegebene Zahl  $x$  ein Paar  $(A[i], A[j]), 1 \leq i, j, \leq n$  mit  $A[i] + A[j] = x$ .

- Sei  $c \in \mathbb{N}$  fest.
- Gebt einen Algorithmus an, der das Problem in deterministischer Zeit  $O(n)$  für natürliche Zahlen  $A[i] \leq c \cdot n$  löst, und bei Erfolg ein Paar  $(A[i], A[j])$  ausgibt, ansonsten *NIL*.

Gegeben ist ein Array  $A = A[1], \dots, A[n]$  mit  $n$  Zahlen in beliebiger Reihenfolge.

Suche für eine gegebene Zahl  $x$  ein Paar  $(A[i], A[j])$ ,  $1 \leq i, j \leq n$  mit  $A[i] + A[j] = x$ .

- Gebt einen Algorithmus an, der das Problem in deterministischer Zeit  $O(n)$  für Ganzzahlen löst, und bei Erfolg ein Paar  $(A[i], A[j])$  ausgibt, ansonsten *NIL*. Hinweise:
  - Wir befinden uns im RAM-Modell mit fixer Wortbreite. Darin kann man ganze Zahlen in  $O(n)$  sortieren.

