

Algorithmen I - Tutorium 3

Sebastian Schmidt – *isibboi@gmail.com*

Arbeitsgruppe Kryptographie und Sicherheit

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

- Aufgabe 1.c)

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

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

- Aufgabe 1.c)

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

- 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 $a \wedge t \notin \langle a, \dots, b \rangle$

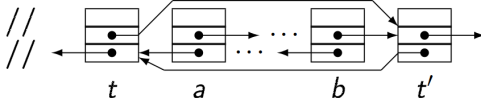
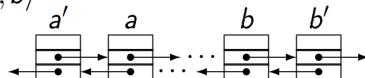
// Cut out $\langle a, \dots, b \rangle$

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

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

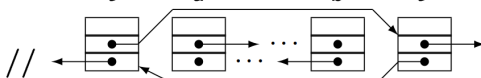
$a' \rightarrow \text{next} := b'$

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



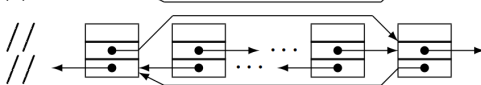
// insert $\langle a, \dots, b \rangle$ after t

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



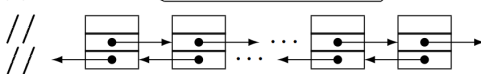
$b \rightarrow \text{next} := t'$

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



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

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



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?

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*.
- 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.

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*.
- 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.

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*.
- 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.