Implementierung von Datenstrukturen

Prof. Dr. Christian Becker

Universität Stuttgart, Institut für Parallele und Verteilte Systeme

15. Mai 2025

Problem

Angenommen, Sie haben ein Array von Integern:

```java
int[] data = new int[10];
```

Java

Wie würden Sie hier noch einen elften Integer anhängen?

```java
int[] append(int[] array, int value) {
  // ???
}
```

Java

Ganz oft ist die Anzahl der gespeicherten Elemente vorher nicht bekannt.

- Lesen einer Datei: wie viele Zeilen hat die Datei?
- Empfangen einer Nachricht aus dem Internet: Wie groß ist die Nachricht?
- Lesen eines User-Inputs: Wie viel wird die Userin schreiben?

Anhängen von Elementen an ein Array

```java
int[] append(int[] array, int value) {
    // Create a new array that is 1 larger
    var out = new int[array.length + 1];
    // Copy old array into the new one
    for(int i = 0; i < array.length; ++i) {
        out[i] = array[i];
    }
    // Put the new element at the last index
    out[out.length - 1] = value;
    return out;
}
```

Java

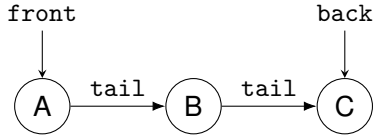
- Das sind $\mathcal{O}(n)$ Kopien um ein einziges Element einzufügen!
- Das muss besser gehen...

Ein Schritt zurück

- Wir suchen nach einer Möglichkeit, eine veränderliche Anzahl Elemente zu speichern, ohne beim Einfügen alle Elemente kopieren zu müssen
- Problem: Arrays haben eine feste Größe und können nicht wachsen (oder schrumpfen)
- Idee: Angenommen, wir haben schon ein paar Einträge in unserer Liste.. Beim Einfügen lassen wir das letzte Element auf das neue Element zeigen!

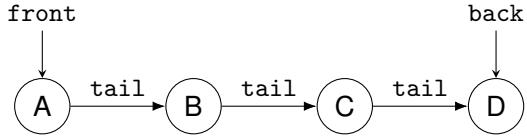
Verkettete Liste

- Idee: Wir hängen den neuen Datenpunkt an den letzten Eintrag an!
- Datenstruktur, die beliebig viele Elemente enthalten kann!



Verkettete Liste

- Idee: Wir hängen den neuen Datenpunkt an den letzten Eintrag an!
- Datenstruktur, die beliebig viele Elemente enthalten kann!



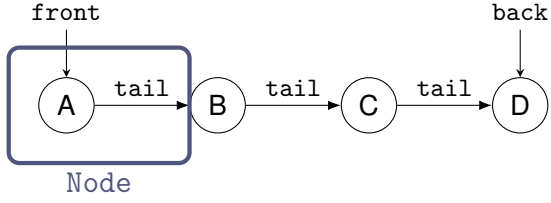
Wie implementiert man das?

Elemente der Verketteten Liste

```java
class Node {
  char value;
  Node tail = null;
}
```

Java

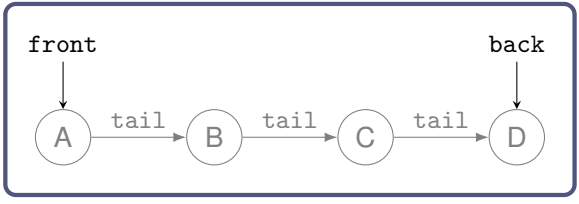
- Node enthält einen Pfeil auf eine weitere Node
- Wenn es kein weiteres Element gibt, dann ist `tail = null;`



Verkettete Liste

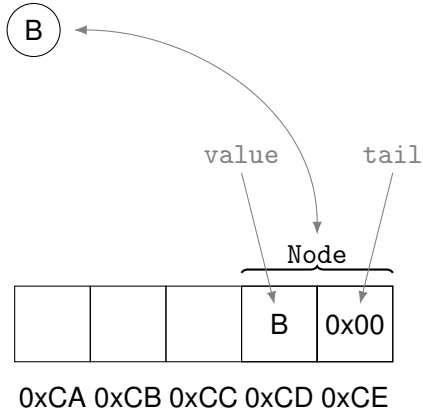
```java
class LinkedList {
  Node front = null;
  Node back = null;
}
```
Java

- Wir speichern das vordere und das hintere Ende der Liste
- Wenn es keine Elemente gibt, dann ist
  `front = back = null;`

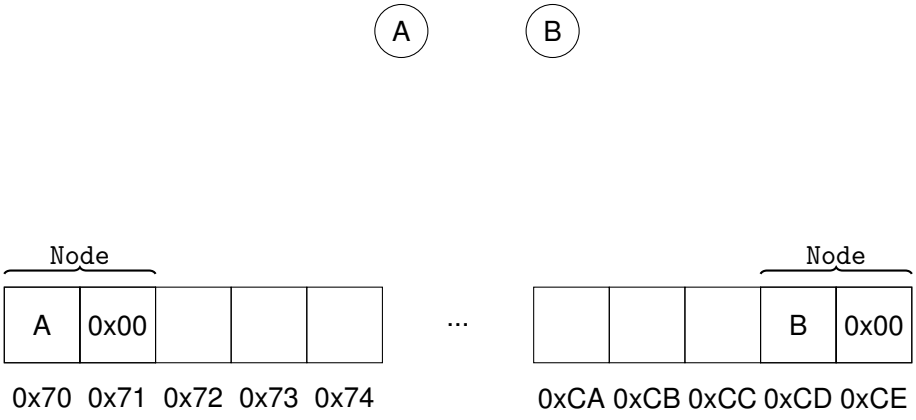


LinkedList

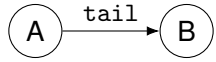
Wie sieht die Liste im Speicher aus?

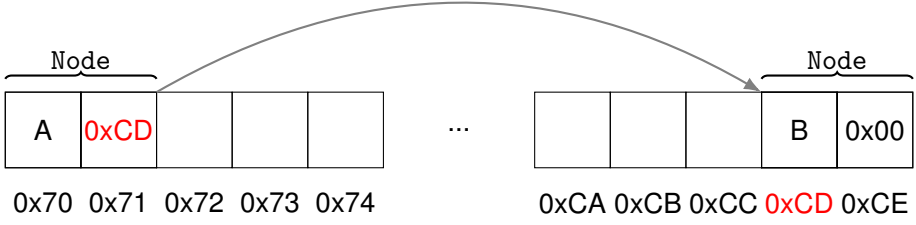


Wie sieht die Liste im Speicher aus?



Wie sieht die Liste im Speicher aus?





- Im Speicher werden Adressen abgespeichert, um die Elemente der Liste zu verketten

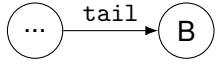
Anhängen von Elementen an eine Verkettete Liste

```java
// hängt das Zeichen value an die Liste list an
void append(LinkedList list, char value) {
  Node element = new Node();
  element.value = value;
  element.tail = null; // das neue Element hat keinen Nachfolger

  // Neues Element an die Liste anhängen
  list.back.tail = element;

  // Ende der Liste ist jetzt das neue Element
  list.back = element;
}
```

Java



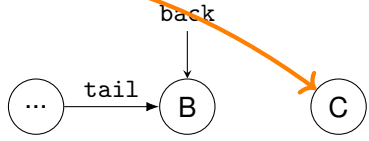
Anhängen von Elementen an eine Verkettete Liste

```java
// hängt das Zeichen value an die Liste list an
void append(LinkedList list, char value) {
  Node element = new Node();
  element.value = value;
  element.tail = null; // das neue Element hat
  ↪  keinen Nachfolger

  // Neues Element an die Liste anhängen
  list.back.tail = element;

  // Ende der Liste ist jetzt das neue Element
  list.back = element;
}
```

Java



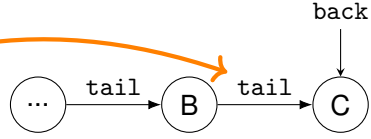
Anhängen von Elementen an eine Verkettete Liste

```java
// hängt das Zeichen value an die Liste list an
void append(LinkedList list, char value) {
  Node element = new Node();
  element.value = value;
  element.tail = null; // das neue Element hat
  ↪ keinen Nachfolger

  // Neues Element an die Liste anhängen
  list.back.tail = element;

  // Ende der Liste ist jetzt das neue Element
  list.back = element;
}
```

Java



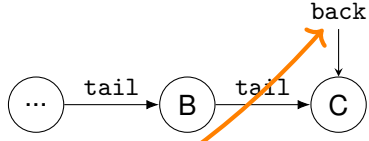
Anhängen von Elementen an eine Verkettete Liste

```java
// hängt das Zeichen value an die Liste list an
void append(LinkedList list, char value) {
  Node element = new Node();
  element.value = value;
  element.tail = null; // das neue Element hat
  ↪  keinen Nachfolger

  // Neues Element an die Liste anhängen
  list.back.tail = element;

  // Ende der Liste ist jetzt das neue Element
  list.back = element;
}
```

Java



Liste von Integern

Unsere verkettete Liste kann auch genau so gut Zahlen speichern indem wir `Node` anpassen:

```java
class Node {
  int value;
  Node tail = null;
}
```

Damit bauen wir jetzt ein paar Algorithmen!

Summe über die Verkettete Liste

```java
int sum(LinkedList list) {
  int result = 0;
  Node current = list.front;
  while(current != null) {
    result = result + current.value;
    current = current.tail;
  }
  return result;
}
```

Java

- Berechnet die Summe der „enthaltenen“ Werte
- Wenn die gegebene Liste leer ist, dann ist die Summe 0

Beispiel-Tikz

Lineare Suche in der Verketteten Liste

```java
boolean search(Node node, int target) {



}
```

- Betrachtet die Liste, die mit `node` beginnt
- Soll `true` zurück geben, wenn `target` in der Liste enthalten ist
- Wenn die Liste leer ist oder `target` nicht gefunden wurde, dann `false`

Lineare Suche in der Verketteten Liste (2)

```java
boolean search(Node node, int target) {
    return switch (node) {
        case null                          -> false;


    };
}
```

Java

- Betrachtet die Liste, die mit `node` beginnt
- Soll `true` zurück geben, wenn `target` in der Liste enthalten ist
- Wenn die **Liste leer ist oder** `target` **nicht gefunden wurde**, dann `false`

Lineare Suche in der Verketteten Liste (3)

```java
boolean search(Node node, int target) {
    return switch (node) {
        case null                          -> false;
        case Node n when n.value == target -> true;

    };
}
```

Java

- Betrachtet die Liste, die mit `node` beginnt
- Soll `true` zurück geben, **wenn** `target` **in der Liste enthalten ist**
- Wenn die Liste leer ist oder `target` nicht gefunden wurde, dann `false`

Lineare Suche in der Verketteten Liste (3)

```java
boolean search(Node node, int target) {
    return switch (node) {
        case null                          -> false;
        case Node n when n.value == target -> true;

    };
}
```

Java

- Betrachtet die Liste, die mit `node` beginnt
- Soll `true` zurück geben, **wenn** `target` **in der Liste enthalten ist**
- Wenn die Liste leer ist oder `target` nicht gefunden wurde, dann `false`

Reicht das schon?

Lineare Suche in der Verketteten Liste (3)

```java
boolean search(Node node, int target) {
    return switch (node) {
        case null                          -> false;
        case Node n when n.value == target -> true;
        case Node n                        -> search(n.tail, target);
    };
}
```

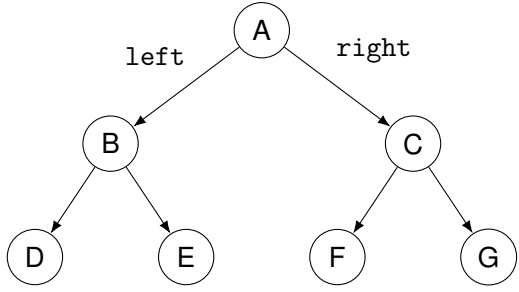
Java

- Rekursion, um die Liste zu durchsuchen, die mit `n.tail` beginnt.

Baum-Datenstrukturen

```java
class Node {
  int value;
  Node left;
  Node right;
}
```

- Jeder Knoten im Baum kann ein linkes und ein rechtes Kind haben



Bäume

Bäume sind wichtige Datenstrukturen die benutzt werden

- in Dateisystemen,
- für effzientes Suchen von Elementen,
- Zählen von eindeutigen Elementen,
- Maps bzw. Dictionaries (folgt gleich)
- Pfadsuche (z.B. in Computerspielen), ...

In Ihrem weiteren Studium werden Sie mehr über Bäume lernen
(Vorlesung Datenstrukturen und Algorithmen)

Sortiert Suchen

Übungsaufgabe: Sortiert Einfügen

Maps und Dictionaries

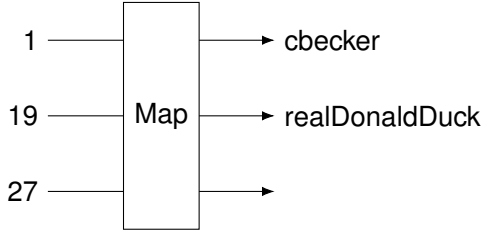
- Sie betreiben eine Chat-Applikation.
- Nutzer haben eine eindeutige UserID (`int`)
- Sie möchten aber den Nutzernamen anzeigen
- Die Zuordnung soll schnell (effizient) sein
- Das Einfügen neuer Nutzer soll bestehende Nutzer nicht beeinflussen

Sie brauchen also eine Zuordung

$$\texttt{int} \rightarrow \texttt{String}$$

Maps und Dictionaries (2)

- Eine Map ist eine Datenstruktur, die einen Lookup realisiert
- Sie erlaubt eine Zuordnung von Werten vom Typ T auf Werte vom Typ K
- Die Elemente T müssen eindeutig sein.



Maps und Dictionaries (3)

Die Zuordnung von UserID zum Nutzernamen soll möglichst effizient sein.

- Effizient bedeutet, möglichst wenige Zugriffe bei der Suche
- Sortieren erleichtert die Suche (vgl. binäre Suche in $\mathcal{O}(\log n)$)

Array

- Vorteil: Binäre Suche einfach zu implementieren
- Nachteil: Beim Einfügen müssen wir neu sortieren bzw. viele Elemente verschieben..

Maps und Dictionaries (3)

Die Zuordnung von UserID zum Nutzernamen soll möglichst effizient sein.

- Effizient bedeutet, möglichst wenige Zugriffe bei der Suche
- Sortieren erleichtert die Suche (vgl. binäre Suche in $\mathcal{O}(\log n)$)

Array

- Vorteil: Binäre Suche einfach zu implementieren
- Nachteil: Beim Einfügen müssen wir neu sortieren bzw. viele Elemente verschieben..

Baum

- Vorteil: Einfügen beeinflusst existierende Elemente nicht
- Was bedeutet „sortiert“ für einen Baum??

Suchen in einem Sortierten Binärbaum

- In Bäumen kann man sortiert einfügen ohne andere Elemente verschieben zu müssen
- Wir betrachten jetzt die Suche in einem sortierten Baum
- Was bedeutet „sortiert“ bei Bäumen?
- Sie kennen schon binäre Suche..

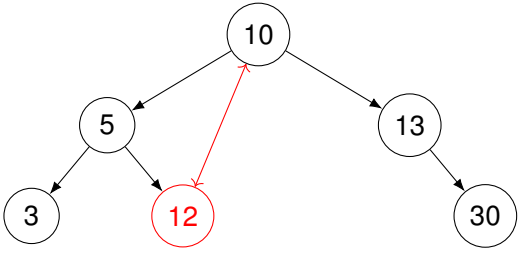
Suchen in einem Sortierten Binärbaum (2)

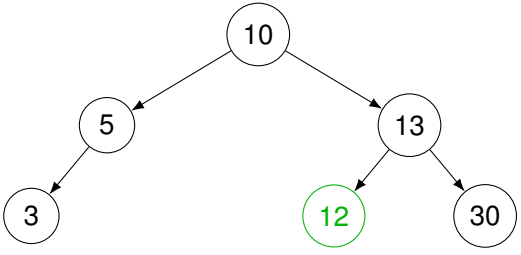
```java
class Node {
  int userid;
  String username;
  Node left;
  Node right;
}
```

Java

Wann ist ein Baum sortiert?

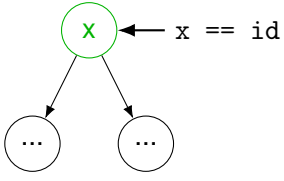
- Der Baum ist sortiert, wenn für alle `Nodes n` gilt
- Die `userid` von allen Nodes im linken Teilbaum ist kleiner als die von `n`
- Die `userid` von allen Nodes im rechten Teilbaum ist größer als die von `n`

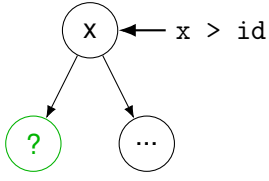


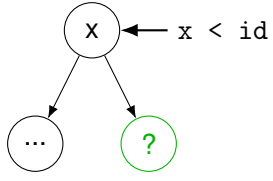


Suchen in einem Sortierten Binärbaum (3)

```java
String search(Node node, int id) {
    return switch(node) {
        case null                        -> null;
        case Node n when n.userid == id -> n.username;          // (1)
        case Node n when n.userid > id  -> search(n.left, id);  // (2)
        case Node n when n.userid < id  -> search(n.right, id); // (3)
    }
}
```







A