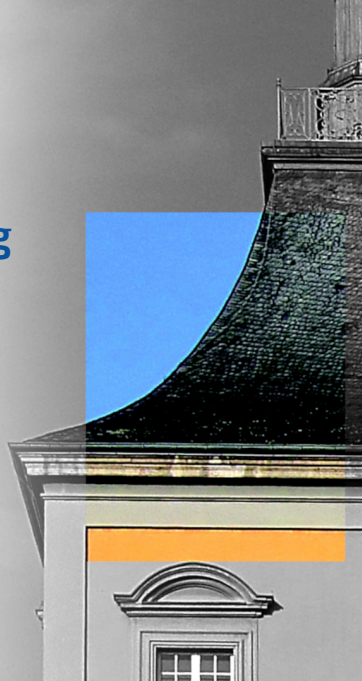
UNIVERSITÄT BONN

Algorithmen und Programmierung

Algorithmen II

Dr. Felix Jonathan Boes
boes@cs.uni-bonn.de
Institut für Informatik



KURZE WIEDERHOLUNG

Abstrakte Datentypen und Datenstrukturen

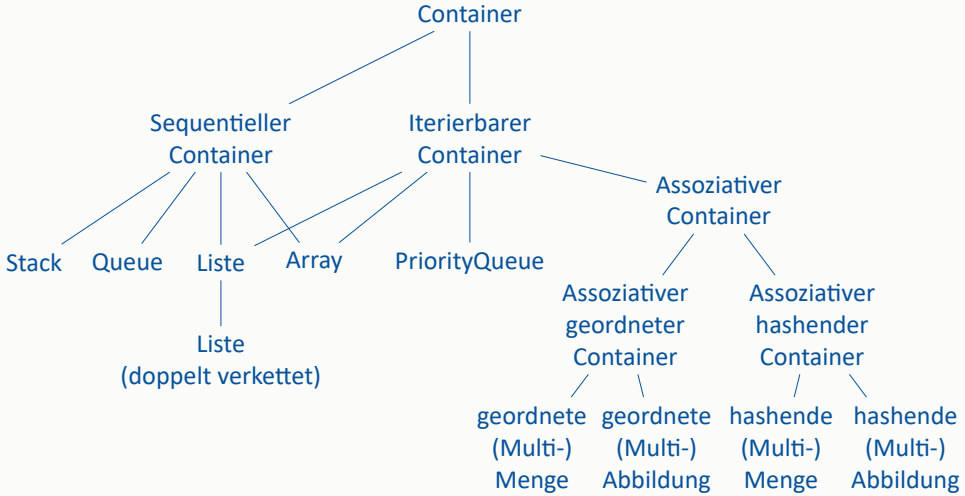
Offene Fragen

Wie sind die wichtigsten abstrakten Datentypen definiert?

Durch welche Datenstrukturen werden sie realisiert?

UNIVERSITÄT BONN

Die (wichtigsten) abstrakten Datentypen im Überblick



Abstrakte Datentypen und Datenstrukturen

Iterierbare Container

Offene Fragen

Was sind iterierbare Container?

Durch welche Datenstrukturen werden sie realisiert?

UNIVERSITÄT BONN

Iterierbare Container (moralisch)

Ein **iterierbarer Container** ist ein abstrakter Datentyp (zum Speichern von Objekten desselben Typs), der folgende Bedingungen erfüllt.

- Die Anzahl der gespeicherten Objekte wird in konstanter Zeit zurückgegeben.
- Die gespeicherten Objekte können nacheinander durchlaufen werden, sofern der Gesamtzustand des Containers beim Durchlauf unverändert bleibt.
- Das erste Element zu erhalten hat eine Laufzeit von $\mathcal{O}(1)$.
- Das jeweils nächste Element zu erhalten hat eine Laufzeit von $\mathcal{O}(\log(n))$.
- Die Prüfung, ob das aktuelle Element das letzte Element ist, hat eine Laufzeit von $\mathcal{O}(1)$.

Bemerkung: Bei fast allen iterierbaren Containern hat das Erhalten des nächsten Elements eine Laufzeit von $\mathcal{O}(1)$.

UNIVERSITÄT BONN

Iterierbare Container (formaler)

Ein **iterierbarer Container** ist ein abstrakter Datentyp (zum Speichern von Objekten desselben Typs) der folgende Bedingungen erfüllt.

- Die Anzahl der gespeicherten Objekte wird in konstanter Zeit zurückgegeben.
- Ein (zum aktuellen Zustand des Containers) zugehöriger **Iterator** kann mit einem Laufzeitaufwand von $\mathcal{O}(1)$ erstellt werden.

Ein **Iterator** referenziert entweder ein (zum Erstellungszeitpunkt des Iterators) enthaltenes Element Containers oder der Iterator enthält die Information, dass er auf kein Element des Containers zeigt. Der Iterator kann mit einer Laufzeit von $\mathcal{O}(\log(n))$ auf das nächste Element referenzieren oder, mit einer Laufzeit von $\mathcal{O}(1)$ die Information bereit stellen, dass alle Elemente einmal durchlaufen wurden.

Ausblick

Es ist sehr einfach die bisher besprochenen abstrakten Datentypen **Liste** und **Array** iterierbar zu machen.

Wie Iteratoren in C++ realisiert werden, besprechen wir später.

In C++ können alle iterierbaren Datenstruturen mit range-based for-loops durchlaufen werden.

Zusammenfassung

Container sind iterierbar falls Sie (mithilfe von Iteratoren) durchlaufen werden können

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Iterierbare Container und deren Realisierung - Priority Queue

Offene Fragen

Was ist eine Priority Queue?

Durch welche Datenstruktur wird sie realisiert?

UNIVERSITÄT BONN

Priority Queue als abstrakter Datentyp

Ein iterierbarer Container ist eine **Priority Queue** falls die folgenden Eigenschaften erfüllt sind.

- Die Anzahl der gespeicherten Objekte wird in konstanter Zeit zurückgegeben.
- Ein neues Element mit zugehöriger (numerischen) Priorität wird eingefügt.
- Das Element mit der höchsten Priorität wird entfernt und zurückgegeben.
- Beide Operationen haben eine Laufzeit von $\mathcal{O}(\log(n))$, wobei n die Anzahl der enthaltenen Elemente angibt.

Es gibt mehrere (ähnliche) Datenstrukturen um Priority Queues zu realisieren. Alle nutzen Varianten von (Binär)bäumen und (Binär)heaps.

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Binärbäume

Überblick

Binärbäume sind grundlegend in der Konstruktion von Datenstrukturen, die Priority Queues realiseren

Binärbäume sind außerdem grundlegend für Datenstrukturen, die geordnete Mengen realiseren

Wir führen hier Binärbäume und Binärheaps ein

Wiederholung: Ungerichteter Graph

(Ungerichteter Graph)

Ein **ungerichteter Graph** $G = (V, E)$ besteht aus einer endlichen Menge von **Knoten** V und einer (endlichen) Menge von **ungerichteten Kanten** $E \subseteq \{\{v, w\} \mid v, w \in V\}$.

Wir sagen dass eine ungerichtete Kante $e = \{v, w\} \in E$ die Knoten v und w verbindet.

Ein ungerichteter Graph ist **zusammenhängend**, wenn je zwei Knoten durch einen Weg miteinander verbunden werden können.

UNIVERSITÄT BONN

Wiederholung: Wege und Kreise

In einem ungerichteten Graph $G = (V, E)$ ist ein **Weg** definiert als eine alternierende Folge von Knoten und Kanten $x_0, e_1, x_1, e_2, x_2, \ldots, e_n, x_n$ sodass gilt

- $x_0, x_1, \ldots, x_n \in V$ und $e_1, e_2, \ldots, e_n \in E$
- $e_i = \{x_{i-1}, x_i\}$ für alle $i = 1, \ldots, n$ und
- jeder Knoten wird höchstens einmal besucht, also $x_i \neq x_j$ für $i \neq j$.

Die **Länge** eines Wegs $x_0, e_1, x_1, e_2, x_2, \ldots, e_n, x_n$ ist n.

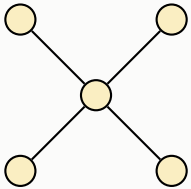
Vereinfacht gesagt ist ein **Kreis** ist ein Weg mit identischem Anfangs- und Endpunkt.

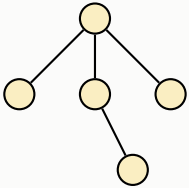
Baum

(Baum)

Ein ungerichteter Graph $G = (V, E)$ ist ein **Baum**, wenn je zwei Knoten durch genau einen Weg miteinander verbunden werden können.

Zwei Beispiele:





UNIVERSITÄT BONN

Äquivalente Baumdefinitionen

(Äquivalente Baumdefinitionen)

Sei $G = (V, E)$ ein ungerichteter Graph mit n Knoten. Dann ist äquivalent:

(1) Der Graph G ist ein Baum.

(2) Der Graph G ist zusammenhängend und kreisfrei.

(3) Der Graph G ist zusammenhängend und das Entfernen einer beliebigen Kante zerstört diese Eigenschaft.

(4) Der Graph G ist kreisfrei und das Hinzufügen einer beliebigen Kante zerstört diese Eigenschaft.

(5) Der Graph G ist zusammenhängend und hat genau $n - 1$ Kanten.

(6) Der Graph G ist kreisfrei und hat genau $n - 1$ Kanten.

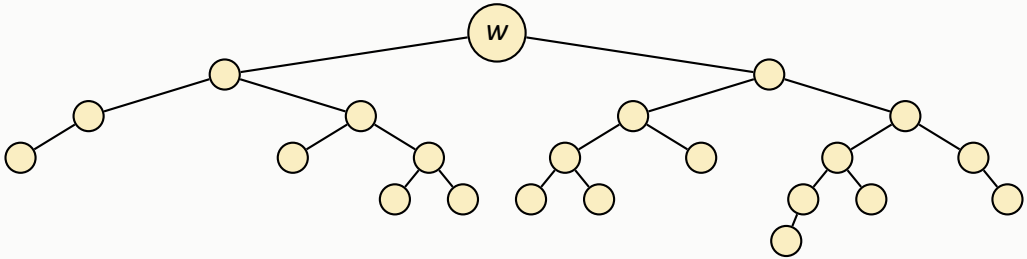
UNIVERSITÄT BONN

Baum mit Wurzel

(Baum mit Wurzel)

Ein **Baum mit ausgezeichneter Wurzel** (G, w) ist ein Baum $G = (V, E)$ zusammen mit der Wahl eines **Wurzelknoten** $w \in V$.

Bäume mit Wurzel werden typischerweise so dargestellt, dass Knoten die nah an der Wurzel liegen oben sind.



UNIVERSITÄT BONN

Knoteneigenschaften

Es sei (G, w) ein Baum mit ausgezeichneter Wurzel. Für jeden Knoten $v \in V$ gibt es genau einen Weg der von w nach v führt.

Die **Tiefe** des Baums ist die Länge des längsten Wegs der in w startet. Die Tiefe des leeren Graphen ist -1.

Gegeben ein Weg $w = v_0 \to \ldots \to v_{i-1} \to v_i \to v_{i+1} \to \ldots \to v_k = v$, dann ist v_{i+1} der **Kindknoten** von v_i und v_{i-1} ist der **Elternknoten** von v_i. Die Knoten v_j mit $j < i$ sind die **Vorfahren** von v_i und v_j mit $j > i$ sind die **Nachkommen** von v_i.

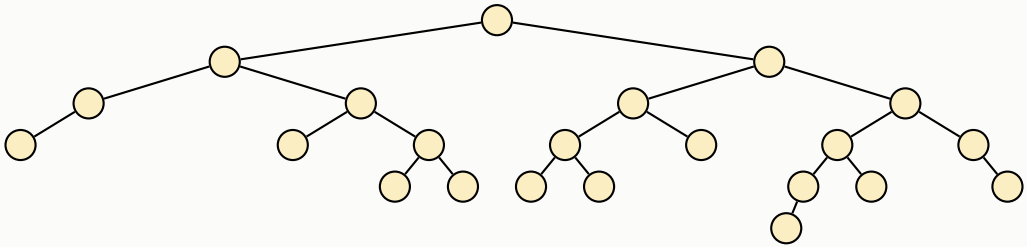
Ein Knoten heißt **Blatt** falls er keine Nachkommen hat. Ein Knoten heißt **innerer Knoten** falls er weder die Wurzel noch ein Blatt ist.

Binärbaum

(Binärbaum)

Ein Baum (G, w) heißt **Binärbaum**, wenn jeder Knoten höchstens zwei Kinder besitzt.

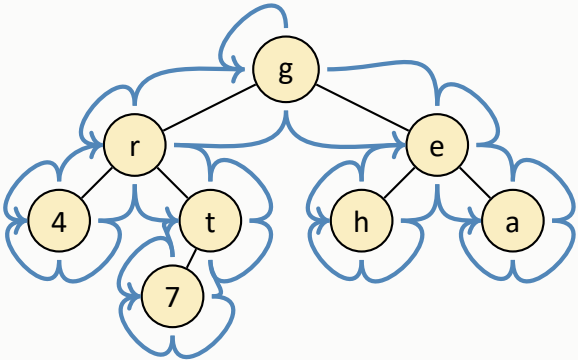
Bei Binärbäumen werden die Kinder meistens angeordnet, das heißt, es ist festgelegt, welches Kind rechts und welches Kind links liegt.



UNIVERSITÄT BONN

Typische Durchlaufstrategien

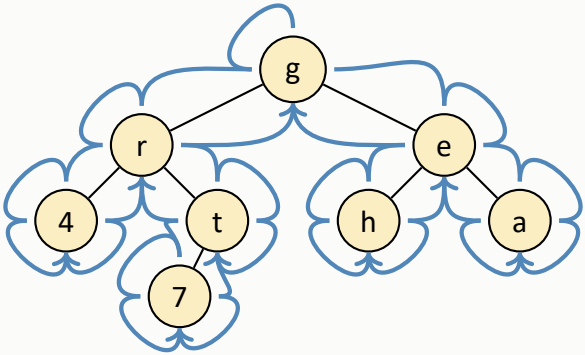
Wir durchlaufen einen Baum in der **Preorder**, wenn wir Tiefensuche verwenden und beim Durchlauf die Wurzel, anschließend (rekursiv) den linken Teilbaum und anschließend (rekursiv) den rechten Teilbaum markieren. Bildlich betrachtet, durchlaufen wir den Baum linksherum und notieren jeden Knoten dessen linke Seite wir passieren.



UNIVERSITÄT BONN

Typische Durchlaufstrategien

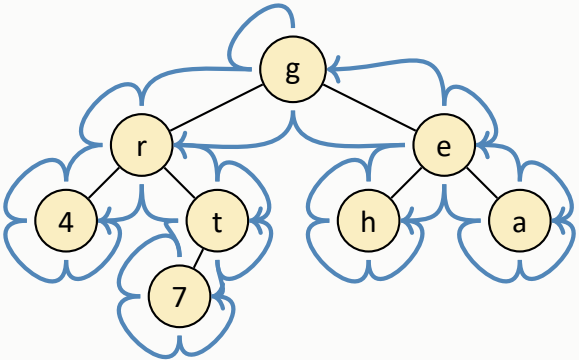
Wir durchlaufen einen Baum in der **Inorder**, wenn wir Tiefensuche verwenden und beim Durchlauf (rekursiv) den linken Teilbaum, anschließend die Wurzel und anschließend (rekursiv) den rechten Teilbaum markieren. Bildlich betrachtet, durchlaufen wir den Baum linksherum und notieren jeden Knoten dessen untere Seite wir passieren.



UNIVERSITÄT BONN

Typische Durchlaufstrategien

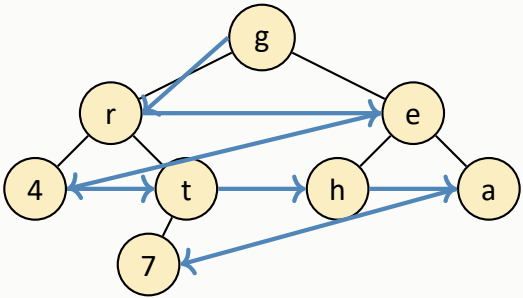
Wir durchlaufen einen Baum in der **Postorder**, wenn wir Tiefensuche verwenden und beim Durchlauf (rekursiv) den linken Teilbaum, anschließend (rekursiv) den rechten Teilbaum und anschließend die Wurzel markieren. Bildlich betrachtet, durchlaufen wir den Baum linksherum und notieren jeden Knoten dessen rechte Seite wir passieren.



UNIVERSITÄT BONN

Typische Durchlaufstrategien

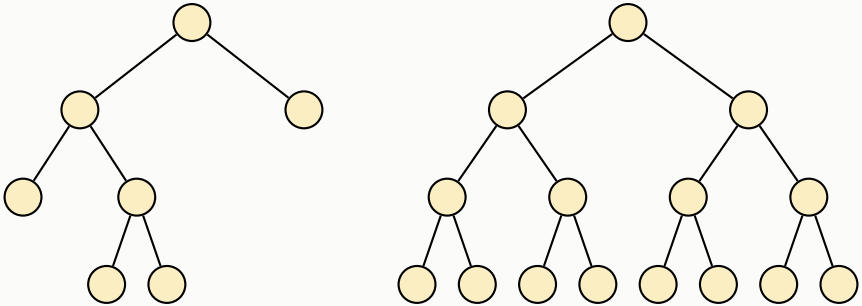
Wir durchlaufen einen Baum in der **Levelorder**, wenn wir Breitensuche verwenden und beim Durchlauf zuerst das linke und anschließend das rechte Kind markieren.



voller Binärbaum

(voller Binärbaum)

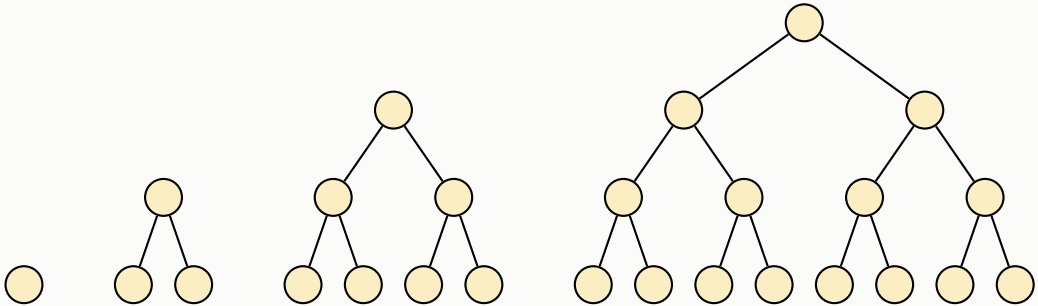
Ein Binärbaum heißt **voll**, falls die Wurzel und alle inneren Knoten genau zwei Kinder besitzen.



vollständiger Binärbaum

(vollständiger Binärbaum)

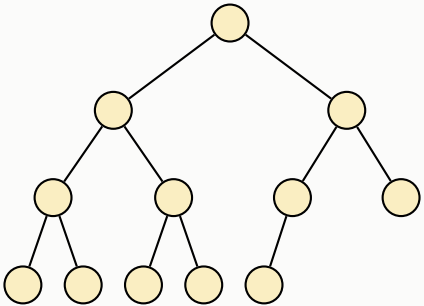
Ein Binärbaum heißt **vollständig**, falls sich alle Blätter auf demselben Level befinden.



Linksvollständiger Binärbaum

(linksvollständiger Binärbaum)

Ein Binärbaum heißt **linksvollständig**, falls alle bis auf das letzte Level voll besetzt sind und im letzten Level jeder mögliche linke Nachbar existiert.

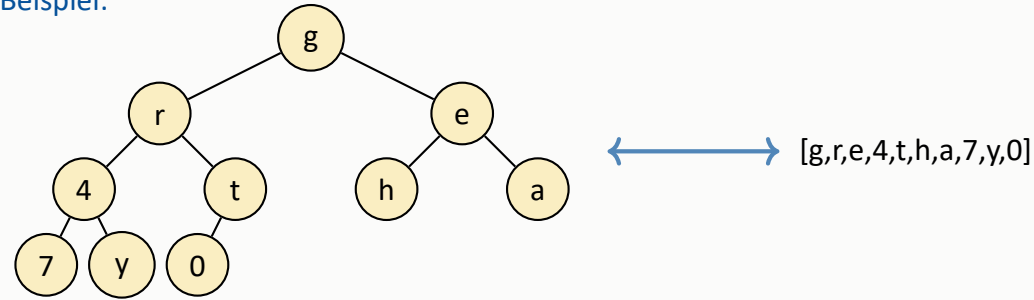


UNIVERSITÄT BONN

Linksvollständige Binärbäume sind Arrays

Die Levelorder bildet einen Binärbaum auf eine Folge von Knoten ab. Die Folge von Knoten kann im Allgemeinen und ohne zusätzliche Information nicht zur Rekonstruktion des Binärbaums genutzt werden. Allerdings funktioniert die Rekonstruktion immer bei linksvollständigen Binärbäumen.

Beispiel:



UNIVERSITÄT BONN

Ausblick

Binärbäume bilden die Grundlage für Datenstrukturen die Priority Queues und geordnete Mengen realisieren.

Im Folgenden verwenden wir linksvollständige Binärbäume um Priority Queues zu realisieren.

Abschließend realisieren wir geordnete Mengen mit speziellen, binären Suchbäumen.

Zusammenfassung

Sie haben Binärbäume und deren wichtigsten Eigenschaften kennen gelernt

Haben Sie Fragen?

Abstrakte Datentypen und Datenstrukturen

Iterierbare Container und deren Realisierung - Priority Queue

Offene Fragen

Durch welche Datenstruktur wird eine Priority Queue realisiert?

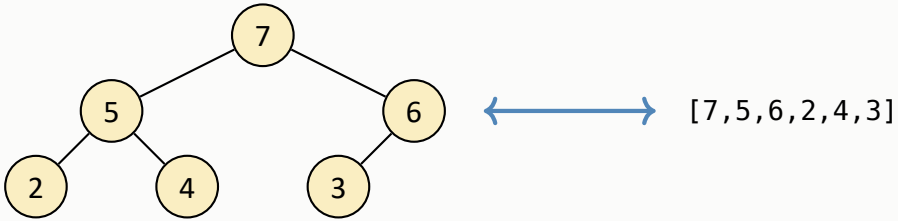
UNIVERSITÄT BONN

Binärer Maxheap

(Binärer Maximumsheap)

Ein **Binärer Maximumsheap**, kurz MaxHeap oder Heap, ist ein linksvollständiger Binärbaum, der die **Heapeigenschaft** erfüllt: Der Wert jedes Knoten ist größer als oder so groß wie die Werte seiner Kindknoten.

Beispiel:



UNIVERSITÄT BONN

Wesentliche Eigenschaften

Da Heaps linksvollständig Binärbäume sind, ist die **Tiefe eines Heaps** mit n Knoten **höchstens** $\lceil \log_2(n) \rceil$.

Auf jedem Weg von der Wurzel zu einem Blatt sind die **Werte absteigend sortiert**. Der Grund ist, dass der Wert jedes Knoten größer ist als die Werte seiner Kindknoten. Allerdings gibt ein Binärheap **keine vollständige Sortierung seiner Elemente** an.

UNIVERSITÄT BONN

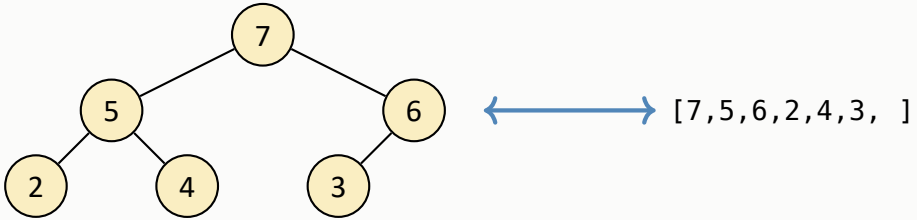
Einfügen eines Elements

Um dem Heap um ein neues Element zu erweitern, sind zwei Schritte notwendig. Zunächst wird das **neue Element** an die nächste, **freie Stelle** gesetzt. In der arrayschreibweise des Heaps entspricht das dem Hinzufügen am Ende des Arrays.

Wir bemerken, dass die Heapeigenschaft nur auf dem Weg von der Wurzel bis zu dem neuen Element wieder hergestellt werden muss. Dazu **tauschen wir iterativ die Position** des Knoten mit dem jeweiligen Elternknoten **bis der direkter Vorgänger größer ist**. Wir bemerken, dass das genügt.

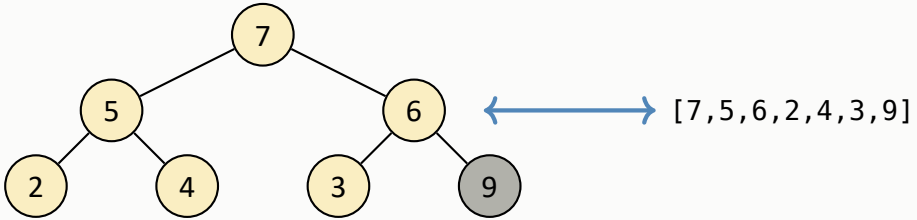
Beispiel

Wir fügen dem Heap das Element 9 hinzu.



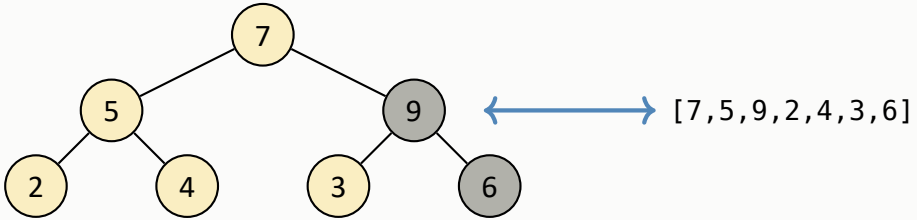
Beispiel

Wir fügen dem Heap das Element 9 hinzu.



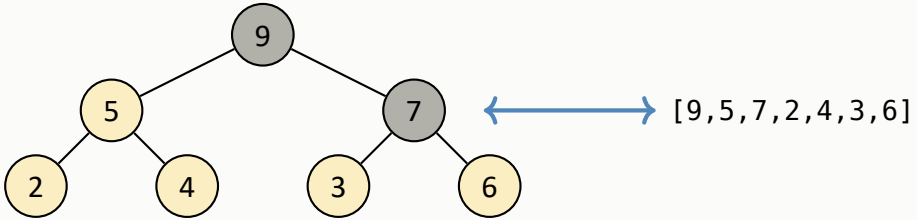
Beispiel

Wir fügen dem Heap das Element 9 hinzu.



Beispiel

Wir fügen dem Heap das Element 9 hinzu.



UNIVERSITÄT BONN

Entfernen des Maximums

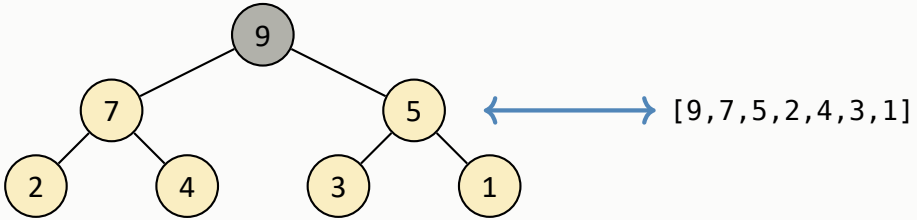
Um das größte Element aus dem Heap zu entfernen, sind drei Schritte notwendig. Zunächst wird das **größte Element** mit dem **letzten Element getauscht**. In der arrayschreibweise des Heaps entspricht das dem Element am Ende des Arrays.

Nun **entfernen** wir das **letzte Element**.

Wir bemerken, dass die Heapeigenschaft nur auf dem Weg von der Wurzel bis zu einem (zu bestimmenden) Blatt hergestellt werden muss. Dazu **tauschen wir iterativ die Posititon** des Knoten mit einem größeren Kindknoten **bis beide Kindknoten kleiner sind**. Wir bemerken, dass das genügt.

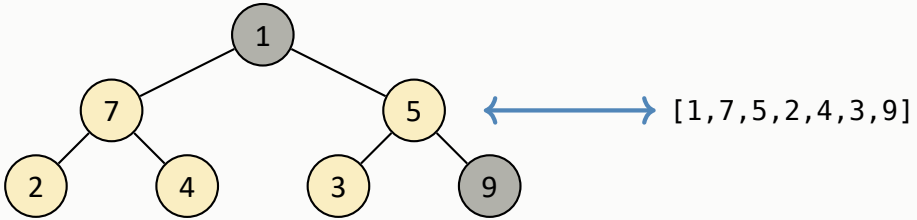
Beispiel

Wir entfernen das größte Element aus dem Heap.



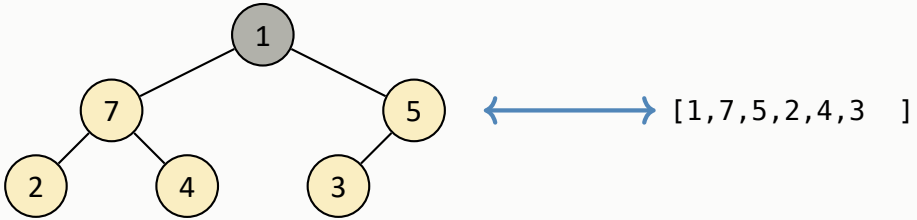
Beispiel

Wir entfernen das größte Element aus dem Heap.



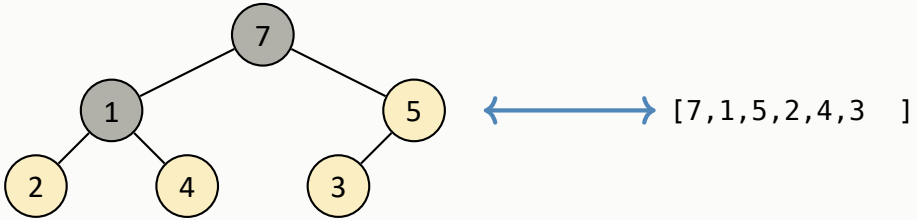
Beispiel

Wir entfernen das größte Element aus dem Heap.



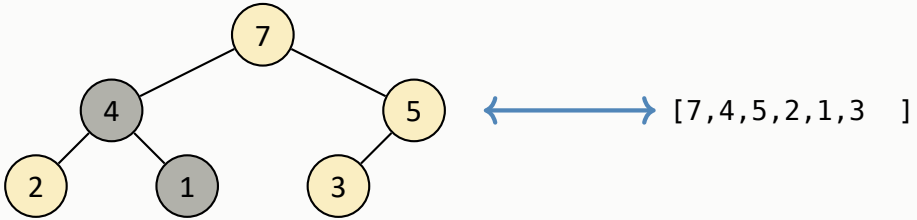
Beispiel

Wir entfernen das größte Element aus dem Heap.



Beispiel

Wir entfernen das größte Element aus dem Heap.



UNIVERSITÄT BONN

Priority Queues werden durch Binäre Maximalheaps realisiert

Um Priority Queues durch Binäre Maximalheaps zu realisieren, werden die Objekte zusammen mit ihrer Priorität entlang der Priorität in den Heap eingeordnet.

Ein Heap ist ein linksvollständiger Binärbaum, also ein Array, und somit iterierbar. Die Operationen des Heaps sind mit den angegebenen Laufzeiten der Priority Queue verträglich, da der Heap eine maximale Tiefe höchstens $\lceil \log_2(n) \rceil$ hat und jede Operation somit höchstens $\lceil \log_2(n) \rceil$ viele Vertauschungen beinhaltet.

Zusammenfassung

Priority Queue werden durch binäre Maximalheaps realisiert

Haben Sie Fragen?

Ausblick Heapsort

Heaps besitzen noch weitere Anwendungsgebiete. Beispielsweise wird der Sortieralgorithmus **Heapsort** mithilfe von Heaps implementiert. Heapsort ist so effizient wie Mergesort, verbraucht aber nur $\mathcal{O}(1)$ zusätzlichen Speicher.