Warteschlange (Priority Queue)

- Dynamische Menge an Elementen (Schlüssel, Wert)
- Elemente sollen nach Prioritäten sortiert aus der Warteschlange entnommen werden
- Priorität basiert auf Schlüssel
- Zwei Versionen: Min-Priorität (Element mit kleinstem Schlüssel hat höchste Priorität) oder Max-Priorität (Element mit größtem Schlüssel hat höchste Priorität, symmetrisch)
- Hier: Min-Priorität

Operationen

- MAKEQUEUE:

  erzeugt eine leere Warteschlange
- UNION(H_1, H_2):

  liefert neue Warteschlange mit allen Elementen in $H_1 \cup H_2$
- MIN(H):

  gibt Element x mit kleinstem Schlüssel in H zurück
- EXTRACT-MIN(H):

  gibt Element x mit kleinstem Schlüssel in H zurück und löscht es
- INSERT(x, H):

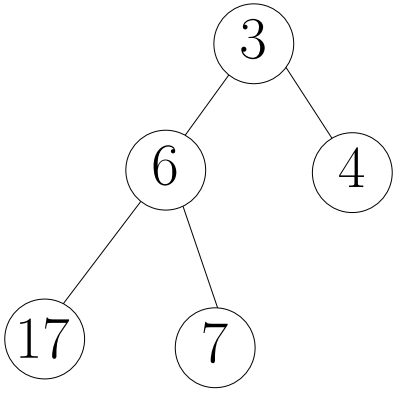
  fügt x in H ein
- DELETE(x, H):

  löscht x aus H
- DECREASE-KEY(x, H, k):

  weist x in H den kleineren Schlüssel k zu

Binary Heap

- Mögliche Implementierung anhand eines *perfekten* binären Baums, der die *Heapeigenschaft* erfüllt
- Binärer Baum ist *perfekt* wenn
  - sich alle Blätter auf benachbarten Ebenen befinden
  - in jeder Ebene alle Blätter möglichst weit links sind
  - es ≤ 1 Knoten mit einem Kind gibt
- Baum erfüllt (Min-)*Heapeigenschaft* falls für jeden Knoten ausser der Wurzel der Schlüsselwert größer ist als der Schlüsselwert des Vaterknotens



Binary Heap

- Wir kennen diese Datenstruktur von Heapsort
- Operationen können mit Heapify als Grundoperation implementiert werden
  - MAKEQUEUE - $O(1)$
  - UNION - $O(n)$
  - MIN - $O(1)$
  - EXTRACT-MIN - $O(\log n)$
  - INSERT - $O(\log n)$
  - DELETE - $O(\log n)$
  - DECREASE-KEY - $O(\log n)$

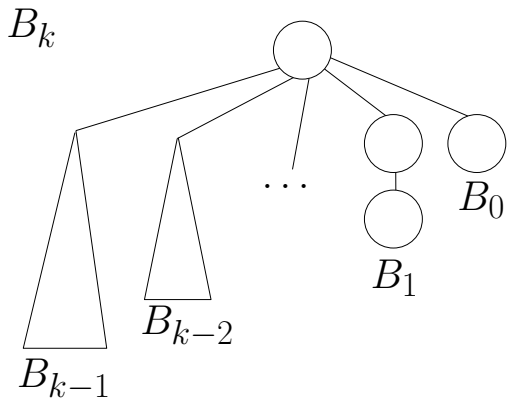
Alternative Datenstruktur

- Erlaubt schnellere Vereinigung (UNION) von Warteschlangen
- Nützlich für Mehrprozessor-Multitasking-Systeme

Binomial Baum

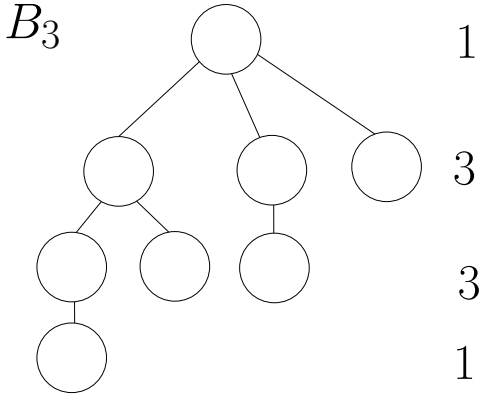
Definition

- B_0: Baum mit einem Knoten
- $B_i, i > 0$: Verkettung zweier Bäume B_{i-1}, wobei ein B_{i-1} an Wurzel des anderen B_{i-1} als linkstes Kind angehängt wird
- i nennen wir *Grad* von B_i



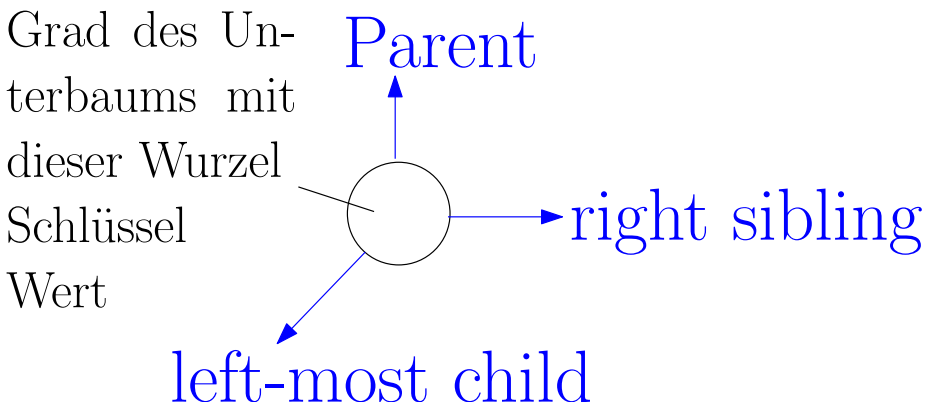
Name *Binomial* Baum

Anzahl der Knoten auf einer Tiefenstufe = Binomialkoeffizienten



Binomial Baum

- Speichern der Elemente als Knoten
- Knoten haben viele Kinder

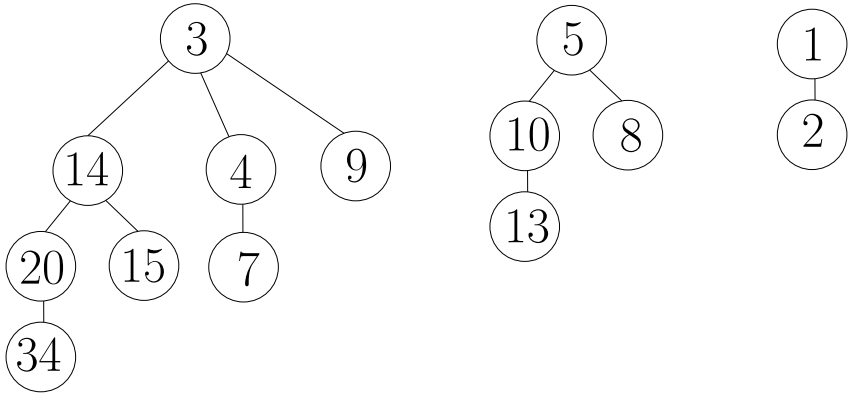


Binomial Baum

- Eigenschaften:
  - B_i hat 2^i Knoten
  - Wurzel von B_i hat i Kinder
  - B_i hat Höhe $i + 1$
- Binomial Baum mit Heapeigenschaft erlaubt schnelles Finden des Elements mit der höchsten Priorität (Wurzel)
- Vereinigung zweier Binomialbäume gleichen Grades mit Heapeigenschaft:
  - Baum mit größerem Schlüsselwert an der Wurzel wird als linkstes Kind an die Wurzel des anderen Baum angehängt

Binomial Heap

- Problem: Binomial Baum kann nur Mengen S von Elementen darstellen, die 2^i Knoten haben
- Binomial Heap
  - Menge von Binomial Bäumen mit einem Knoten pro Element
  - Jeder Baum hat die Heapeigenschaft
  - Keine zwei Bäume haben denselben Grad
- Für jede Menge S mit $|S| = n$ kann eine solche Menge von Binomial Bäumen gefunden werden



Operationen

- MAKEQUEUE: $O(1)$
- MIN(H): $O(\log n)$

  Finde kleinste aller Wurzeln
- UNION(H_1, H_2): $O(\log n)$

  Vereinigung zweier Binomial Heaps verwendet als Grundoperation Vereinigung zweier Binomial Bäume gleichen Grades und ist analog zur Addition von Binärzahlen
- EXTRACT-MIN(H): $O(\log n)$
  - Entferne Binomial Baum B_i mit kleinstem Schlüsselwert in Wurzel aus H
  - Vereinige Kinder von Wurzel von B_i mit dem geänderten H mit UNION

Operationen

- INSERT(x, H): $O(\log n)$
  - Produziere B_0 mit Wert x
  - Vereinige B_0 und H durch UNION
- DECREASE-KEY(x, H, k): $O(\log n)$
  - Verkleinere Schlüssel von x
  - Tausche x so lange mit Vaterknoten bis Heapeigenschaft wieder hergestellt ist
- DELETE(x): $O(\log n)$
  - DECREASE-KEY($x, H, -\infty$)
  - EXTRACT-MIN(H)