5. Vorrangwarteschlangen (priority queues)

Definition 200

Eine Vorrangwarteschlange (priority queue) ist eine Datenstruktur, die die folgenden Operationen effizient unterstützt:

1. Insert
2. ExtractMin Extrahieren und Löschen des Elements mit dem kleinsten Schlüssel
3. DecreaseKey Verkleinern eines Schlüssels
4. Union (meld) Vereinigung zweier (disjunkter) Priority Queues

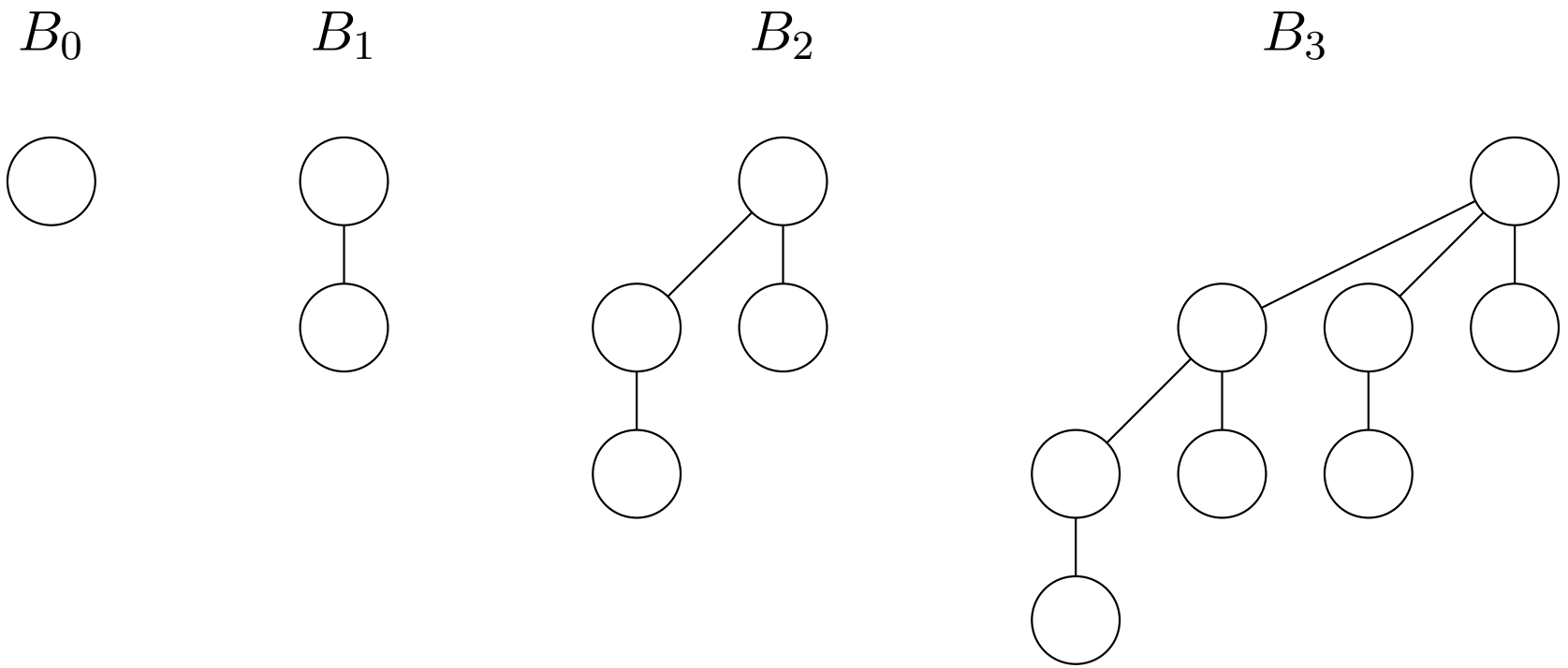
Wir besprechen die Implementierung einer Vorrangwarteschlange als Binomial Heap. Diese Implementierung ist (relativ) einfach, aber asymptotisch bei weitem nicht so gut wie modernere Datenstrukturen für Priority Queues, z.B. Fibonacci Heaps (die in der weiterführenden Vorlesung EA1 besprochen werden).

Hier ist ein kurzer Vergleich der Zeitkomplexitäten:

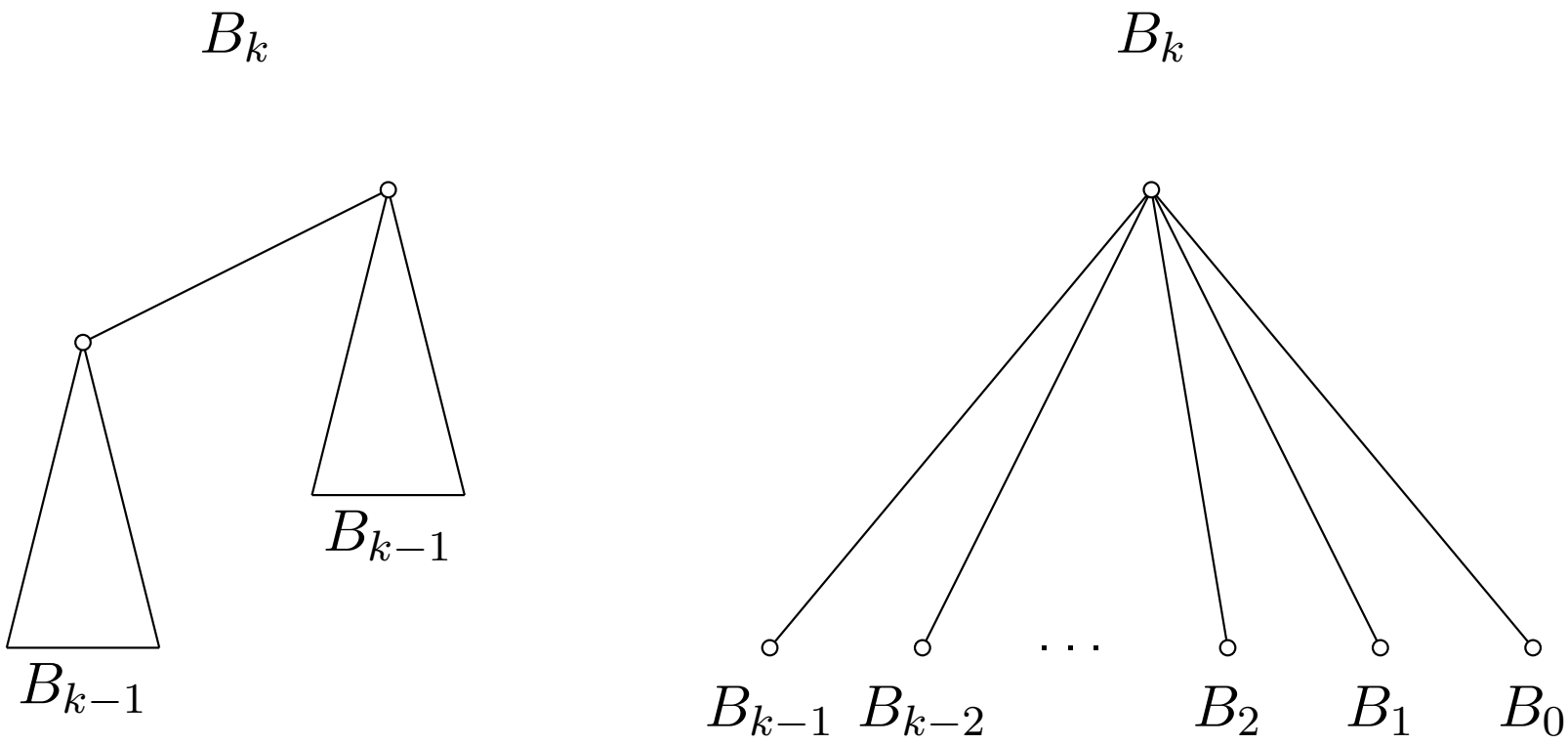
	BinHeap	FibHeap
Insert	$O(\log n)$	$O(1)$
ExtractMin	$O(\log n)$	$O(\log n)$
DecreaseKey	$O(\log n)$	$O(1)$
Union	$O(\log n)$	$O(1)$
	worst case	amortisiert

Definition 201

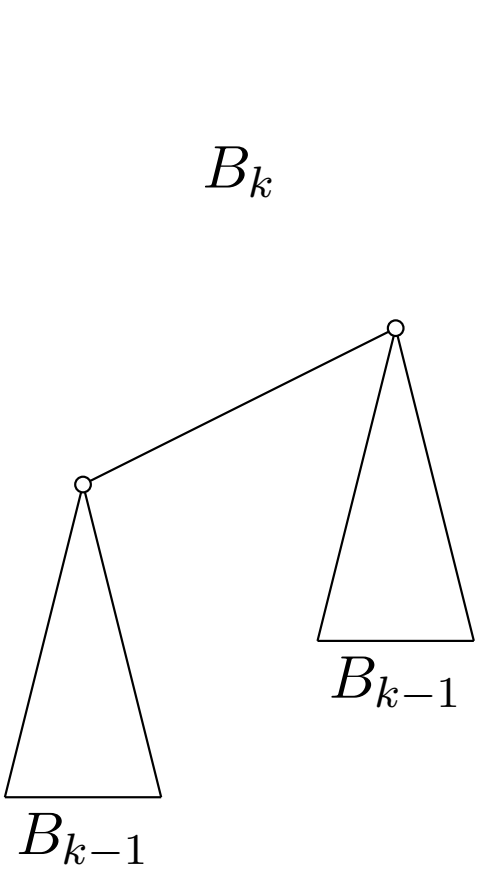
- Der Binomialbaum B_0 besteht aus genau einem Knoten.
- Den Binomialbaum B_k, $k \geq 1$, erhält man, indem man die Wurzel eines B_{k-1} zu einem zusätzlichen Kind der Wurzel eines zweiten B_{k-1} macht.

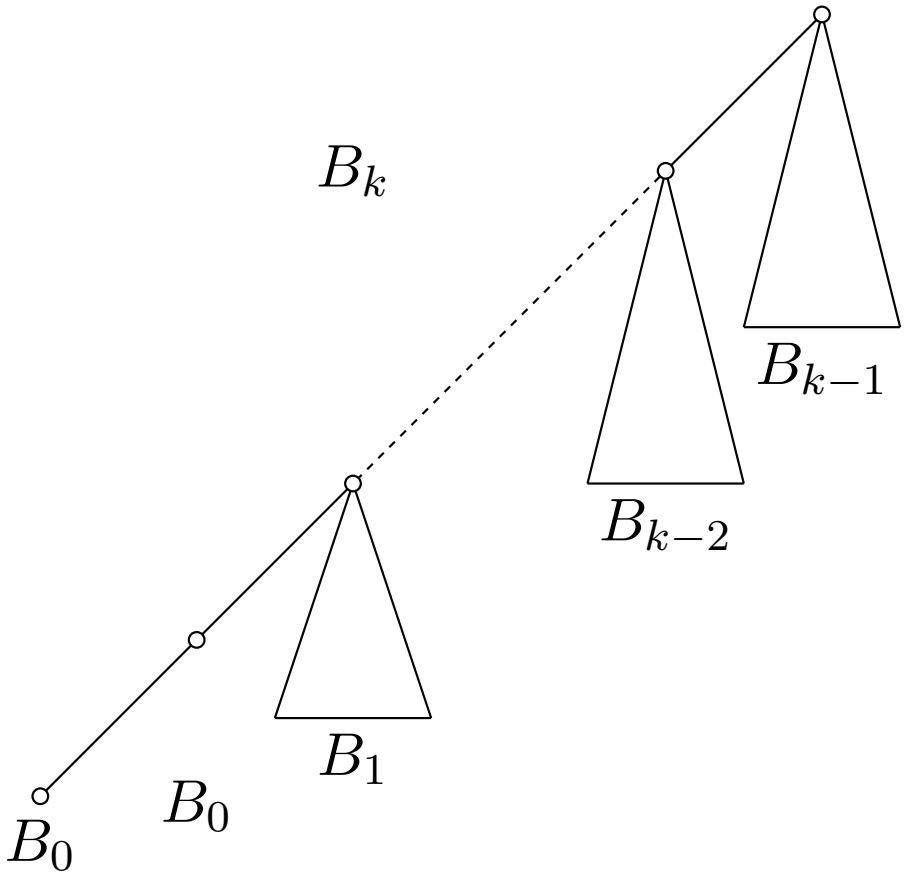


Rekursiver Aufbau des Binomialbaums B_k (rekursive Verfolgung des rechten Zweigs)



Rekursiver Aufbau des Binomialbaums B_k (rekursive Verfolgung des linken Zweigs)





Satz 202

Für den Binomialbaum B_k, $k \geq 0$, gilt:

1. *er hat Höhe k und enthält 2^k Knoten;*
2. *er enthält genau $\binom{k}{i}$ Knoten der Tiefe i, für alle $i \in \{0, \ldots, k\}$;*
3. *seine Wurzel hat k Kinder, alle anderen Knoten haben $< k$ Kinder.*

Beweis:

Der Beweis ergibt sich sofort aus dem rekursiven Aufbau des B_k und der Rekursionsformel

$$\binom{k}{i} = \binom{k-1}{i} + \binom{k-1}{i-1}$$

für Binomialkoeffizienten. □

Definition 203

Ein Binomial Heap ist eine *Menge* $\mathcal{H}$ von Binomialbäumen, wobei jedem Knoten v ein Schlüssel $key(v)$ zugeordnet ist, so dass folgende Eigenschaften gelten:

1. jeder Binomialbaum $\in \mathcal{H}$ erfüllt die Heap-Bedingung und ist ein min-Heap:

$$(\forall \text{ Knoten } v, w)[v \text{ Vater von } w \Rightarrow key(v) \leq key(w)]$$

2. $\mathcal{H}$ enthält für jedes $k \in \mathbb{N}_0$ höchstens einen B_k

Für jeden Binomial Heap $\mathcal{H}$ gilt also

1. Enthält $\mathcal{H}$ n Schlüssel, $n \in \mathbb{N}$, so besteht $\mathcal{H}$ höchstens aus $\max\{1, \lceil \mathrm{ld}\, n \rceil\}$ Binomialbäumen.
2. In jedem von $\mathcal{H}$'s Binomialbäumen ist ein kleinster Schlüssel an der Wurzel gespeichert; verlinkt man daher die Wurzeln aller Binomialbäume von $\mathcal{H}$ in einer zirkulären Liste, kann ein minimaler Schlüssel in $\mathcal{H}$ durch einfaches Durchlaufen dieser Liste gefunden werden.

Korollar 204

In einem Binomial Heap mit n Schlüsseln benötigt FindMin *Zeit* $O(\log n)$.

Wir betrachten nun die Realisierung der Union-Operation. Hierbei wird vorausgesetzt, dass die Objektmengen, zu denen die Schlüssel in den beiden zu verschmelzenden Binomial Heaps $\mathcal{H}_1$ und $\mathcal{H}_2$ gehören, disjunkt sind. Es ist jedoch durchaus möglich, dass der gleiche Schlüssel für mehrere Objekte vorkommt.

1. Fall $\mathcal{H}_1$ und $\mathcal{H}_2$ enthalten jeweils nur einen Binomialbaum B_k (mit dem gleichen Index k):

In diesem Fall fügen wir den B_k, dessen Wurzel den größeren Schlüssel hat, als neuen Unterbaum der Wurzel des anderen B_k ein, gemäß der rekursiven Struktur der Binomialbäume. Es entsteht ein B_{k+1}, für den per Konstruktion weiterhin die Heap-Bedingung erfüllt ist.

2. Fall Sonst. Sei $\mathcal{H}_1 = \{B_i^1;\ i \in I_1 \subset \mathbb{N}_0\}$ und sei $\mathcal{H}_2 = \{B_i^2;\ i \in I_2 \subset \mathbb{N}_0\}$.

Wir durchlaufen dann alle Indizes $k \in [0, \max\{I_1 \cup I_2\}]$ in aufsteigender Reihenfolge und führen folgende Schritte durch:

1. falls kein Binomialbaum mit Index k vorhanden ist: **noop**
2. falls genau ein Binomialbaum mit Index k vorhanden ist, wird dieser in den verschmolzenen Binomial Heap übernommen
3. falls genau zwei Binomialbäume mit Index k vorhanden sind, werden diese gemäß dem 1. Fall verschmolzen; der entstehende B_{k+1} wird im nächsten Schleifendurchlauf wieder betrachtet. Dadurch kann auch der folgende Fall eintreten!
4. falls genau drei Binomialbäume mit Index k vorhanden sind, wird einer davon in den verschmolzenen Binomial Heap übernommen; die beiden anderen werden gemäß dem 1. Fall verschmolzen; der entstehende B_{k+1} wird im nächsten Schleifendurchlauf wieder betrachtet.

Man beachte, dass nie mehr als 3 Binomialbäume mit gleichem Index auftreten können!

Bemerkung:
Es besteht eine einfache Analogie zur Addition zweier Binärzahlen, wobei das Verschmelzen zweier gleich großer Binomialbäume dem Auftreten eines Übertrags entspricht!

Lemma 205
Die Union-*Operation zweier Binomial Heaps mit zusammen* n *Schlüsseln kann in Zeit* $O(\log n)$ *durchgeführt werden.*

Die Operation Insert$(\mathcal{H}, k)$:

1. erzeuge einen neuen Binomial Heap $\mathcal{H}' = \{B_0\}$ mit k als einzigem Schlüssel
2. $\mathcal{H} :=$ Union$(\mathcal{H}, \mathcal{H}')$

Falls $\mathcal{H}$ n Schlüssel enthält, beträgt die Laufzeit der Insert-Operation offensichtlich $O(\log n)$.

Die Operation ExtractMin($\mathcal{H}$):

1. durchlaufe die Liste der Wurzeln der Binomialbäume in $\mathcal{H}$ und finde den/einen Baum mit minimaler Wurzel
2. gib den Schlüssel dieser Wurzel zurück
3. entferne diesen Binomialbaum aus $\mathcal{H}$
4. erzeuge einen neuen Binomial Heap $\mathcal{H}'$ aus den Unterbäumen der Wurzel dieses Baums
5. $\mathcal{H} := \text{Union}(\mathcal{H}, \mathcal{H}')$

Falls $\mathcal{H}$ n Schlüssel enthält, beträgt die Laufzeit der ExtractMin-Operation offensichtlich $O(\log n)$.