# Assignment 9
# Advanced Algorithms & Data Structures PS

Christian Müller 1123410
Daniel Kocher, 0926293

June 8, 2016

**Aufgabe 18**

Sei $Q$ eine Binomial Queue, die anfangs genau einen Binomialbaum $B_1$ mit den Schlüsseln 13 und 21 enthält. Fügen Sie die Schlüssel 3, 7, 15, 18, 8, 14 und 27 in die Queue ein. Löschen Sie anschließend die Elemente 15 und 27 und wenden Sie `decreasekey(18, 4)` an. Geben Sie nach jedem Schritt die resultierende Queue an.

Im Folgenden werden die einzelnen Schritte dargestellt, wobei für die Child-Sibling-Parent Darstellung die folgenden Pfeile verwendet werden:

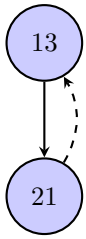




Figure 1: Ausgangs-Queue $Q$

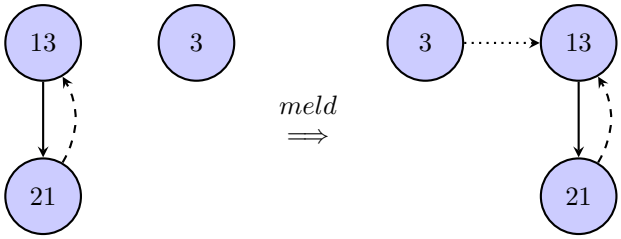



Figure 2: Einfügen von 3 (*meld*: $B_0$ vor $B_1$)

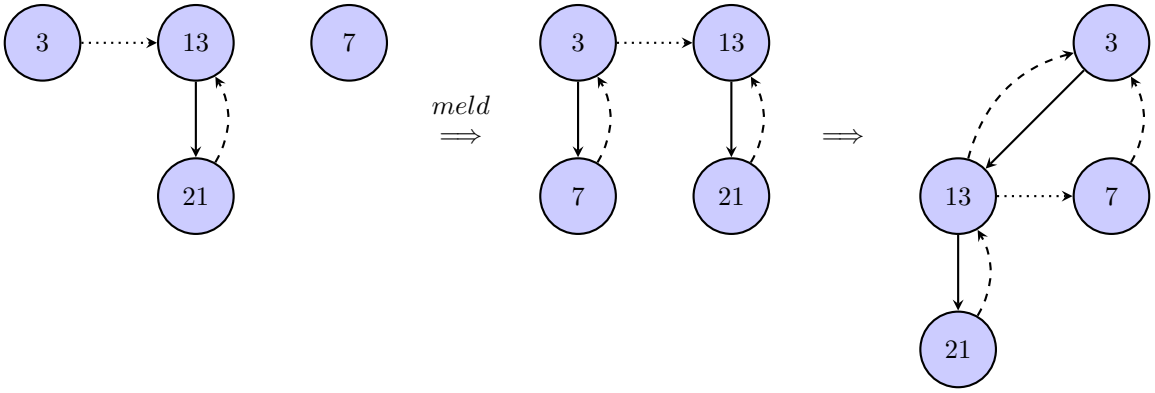



Figure 3: Einfügen von 7 (*meld* vereinigt zweimal: $B_0$ und $B_1$)

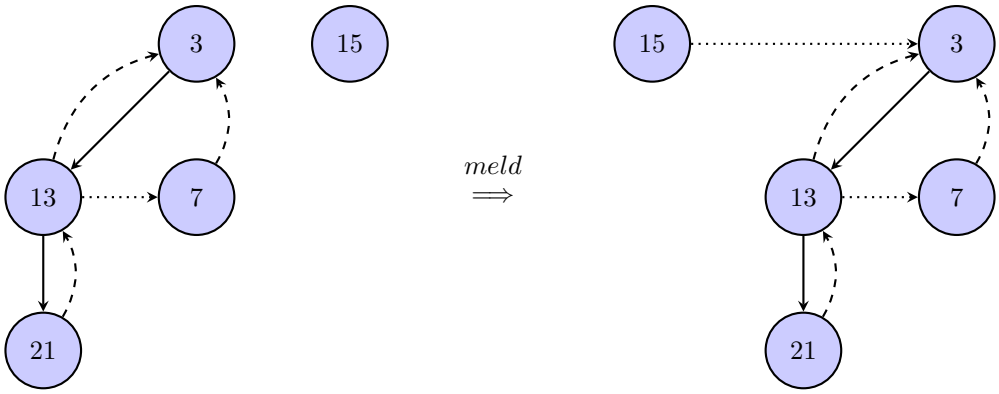



Figure 4: Einfügen von 15 (*meld*: $B_0$ vor $B_2$)

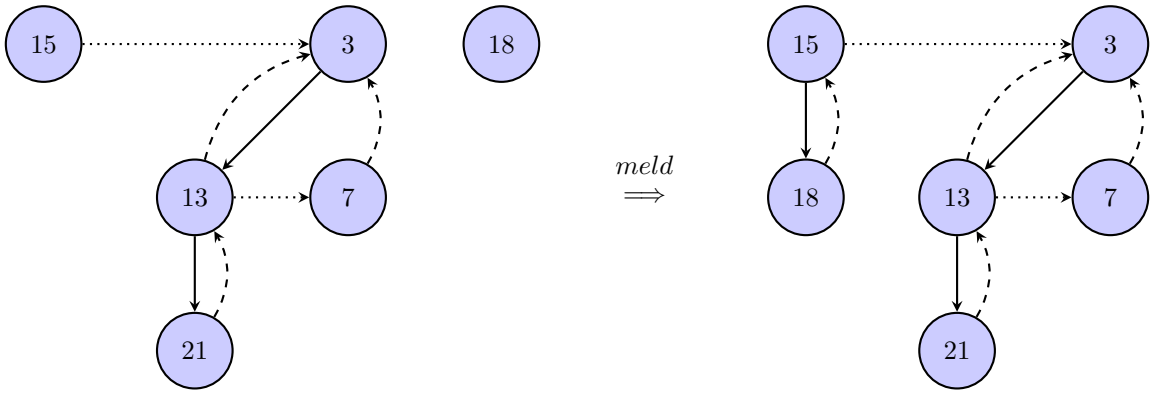



Figure 5: Einfügen von 18 (*meld* vereinigt einmal: $B_0$)

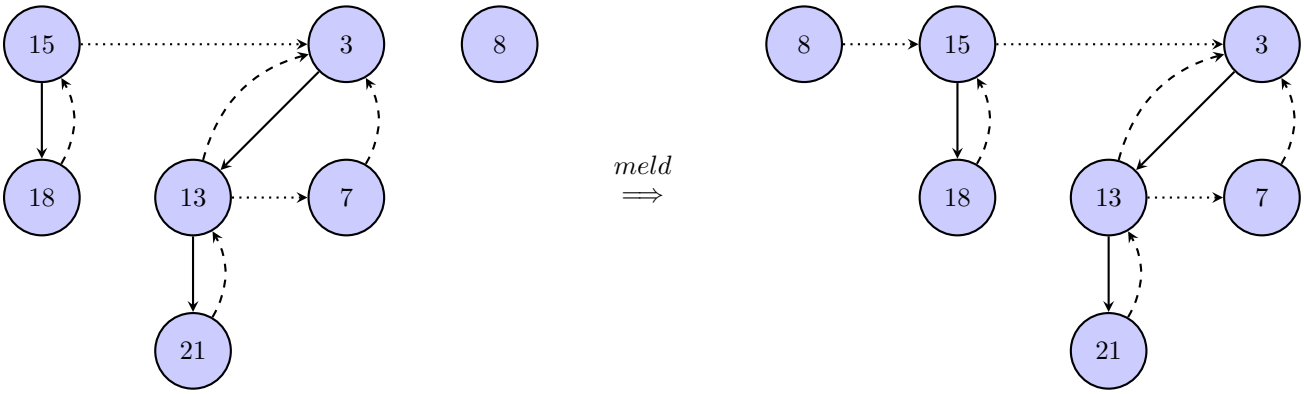



Figure 6: Einfügen von 8 (*meld*: $B_0$ vor $B_1$ vor $B_2$)

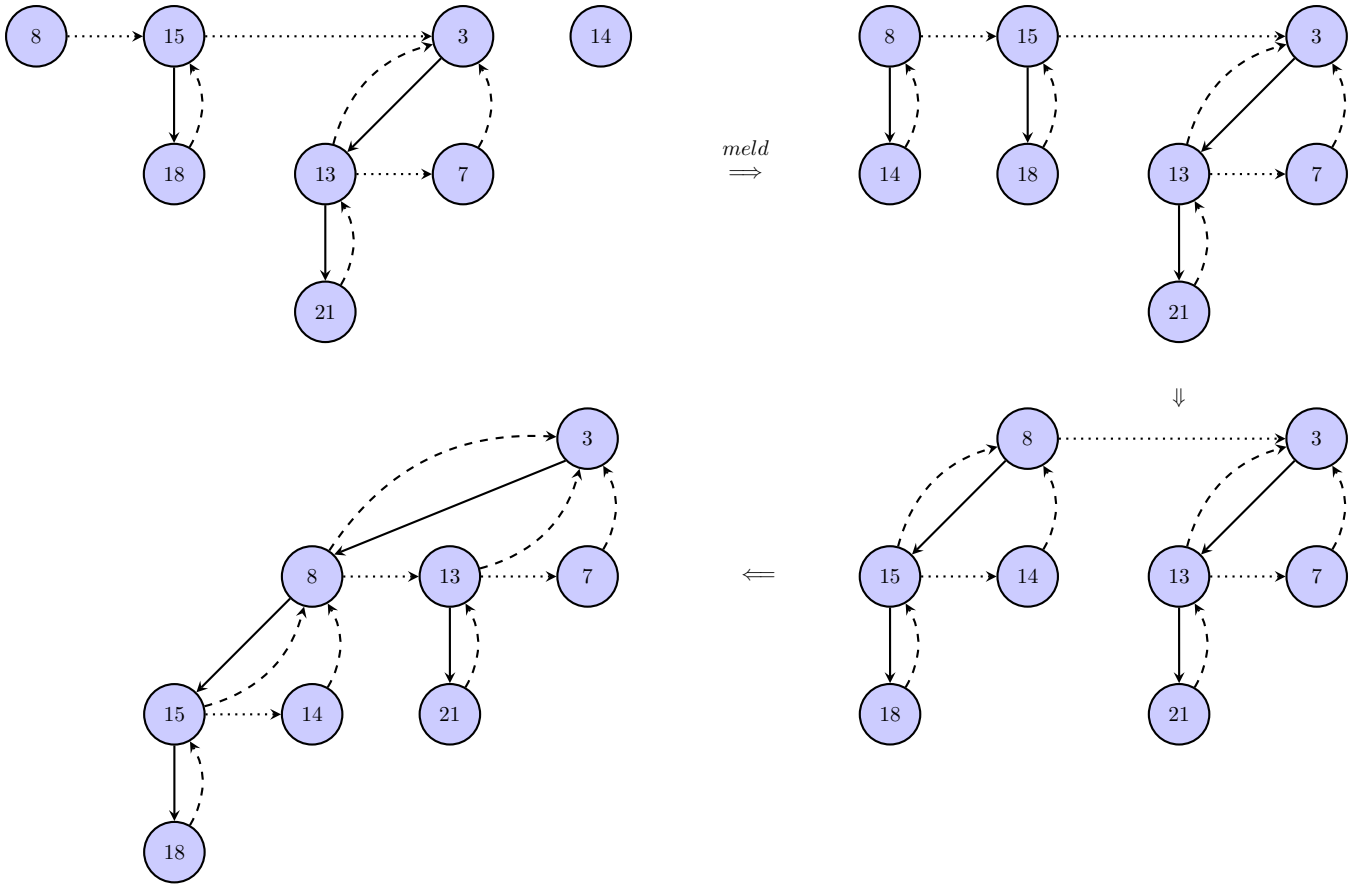



Figure 7: Einfügen von 14 (*meld* vereinigt dreimal: $B_0$, $B_1$ und $B_2$)

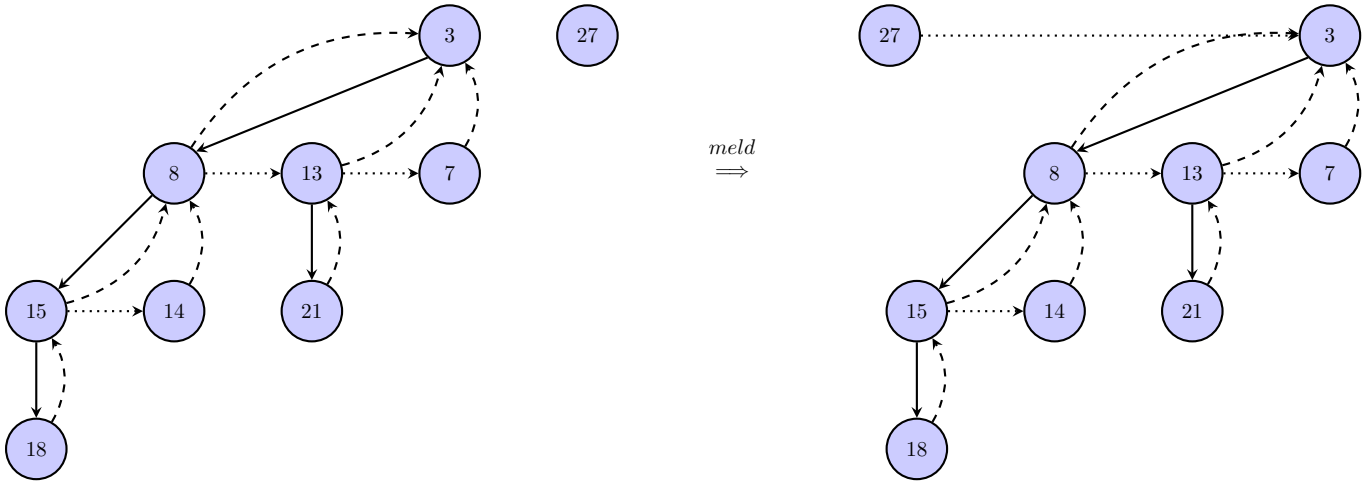



Figure 8: Einfügen von 27 (*meld*: $B_0$ vor $B_3$)

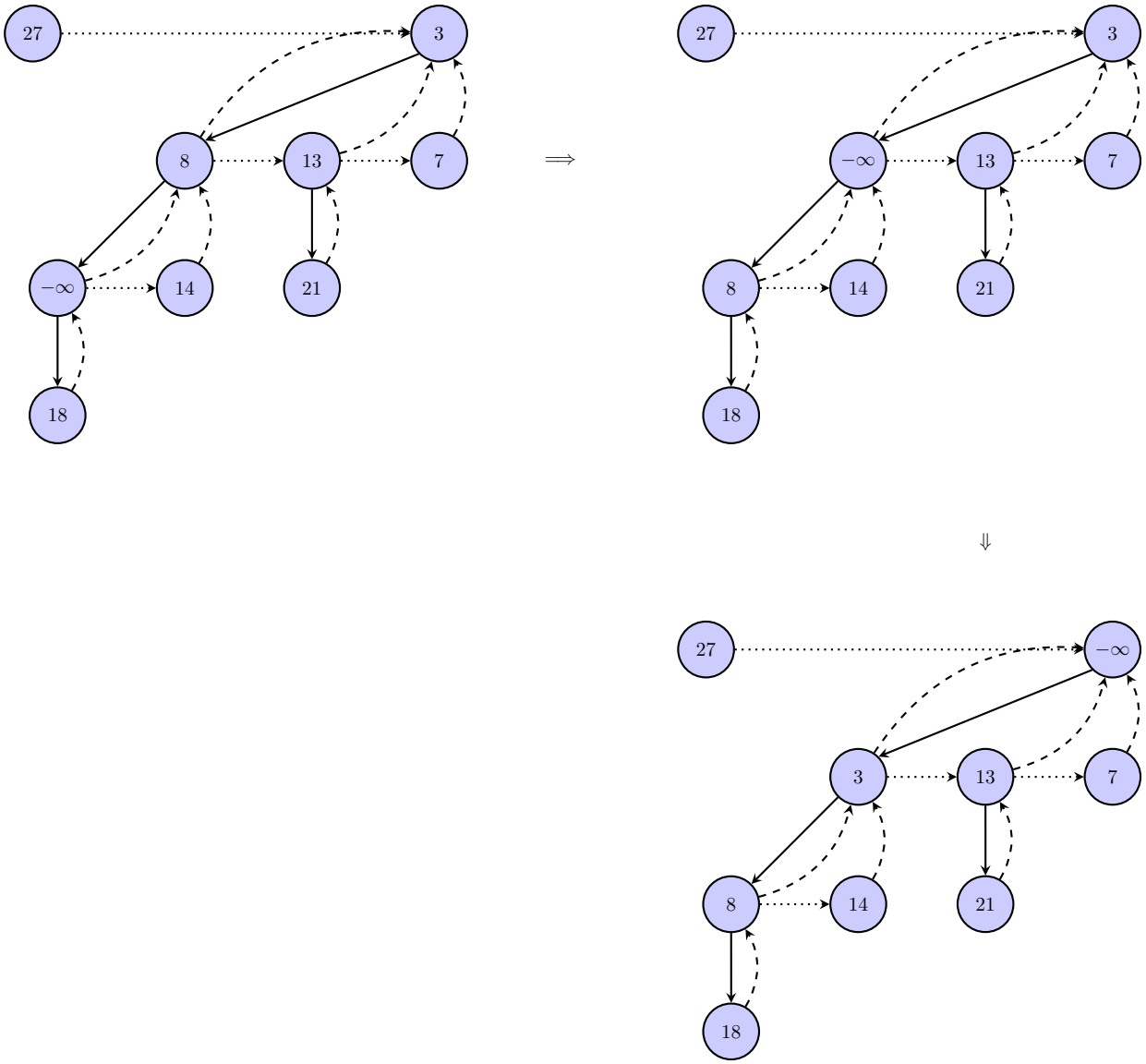



Figure 9: Löschen von 15: Ersetze 15 mit $-\infty$ und lasse diesen Knoten nach oben wandern.

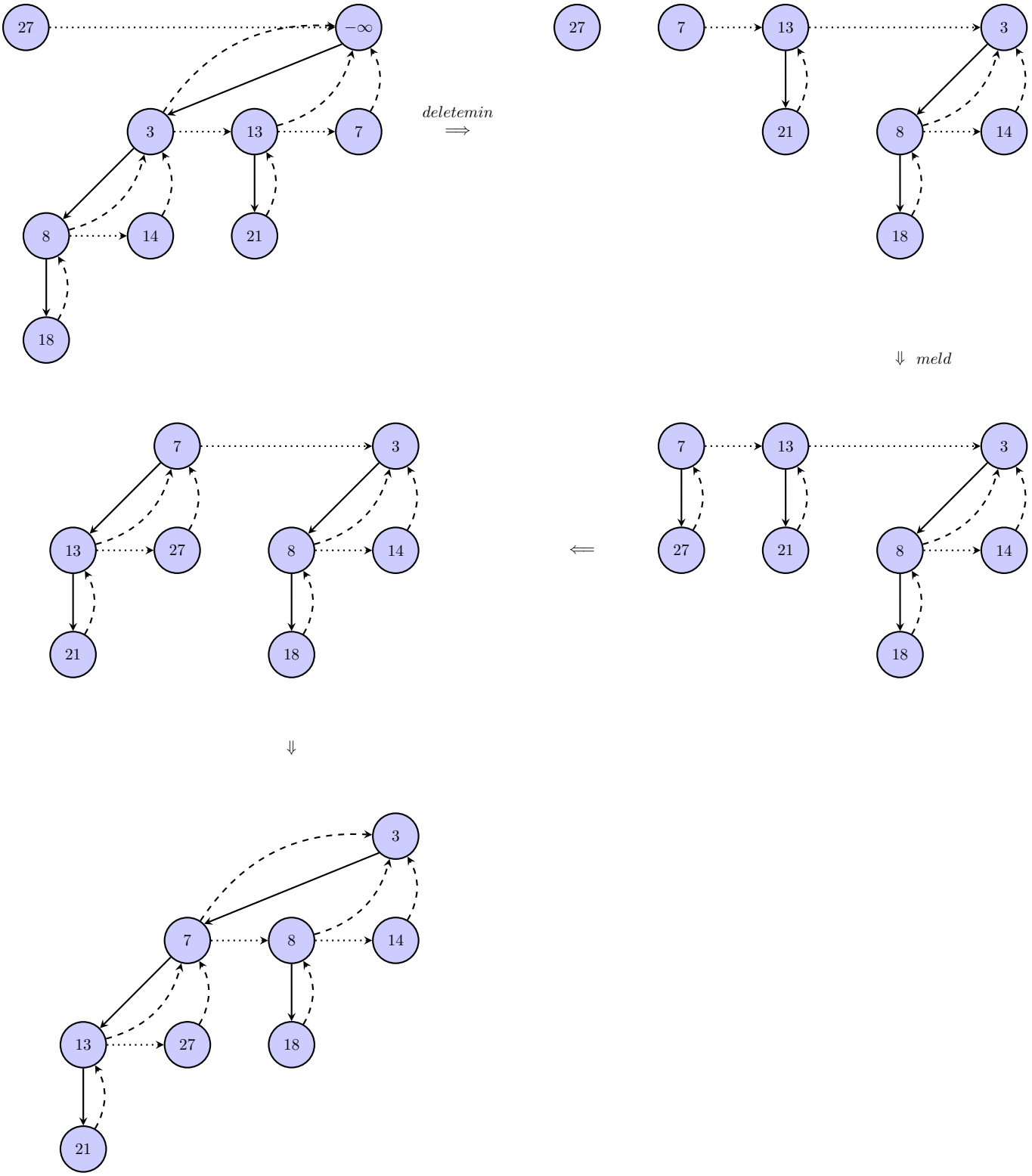



Figure 10: `deletemin()` ($-\infty$ hat minimalen Schlüssel in der Wurzelliste $\Rightarrow$ entferne $-\infty$ aus $Q$ (liefert $Q'$); Drehe Reihenfolge der Söhne von $-\infty$ um (liefert $Q''$); $Q'.meld(Q'')$)

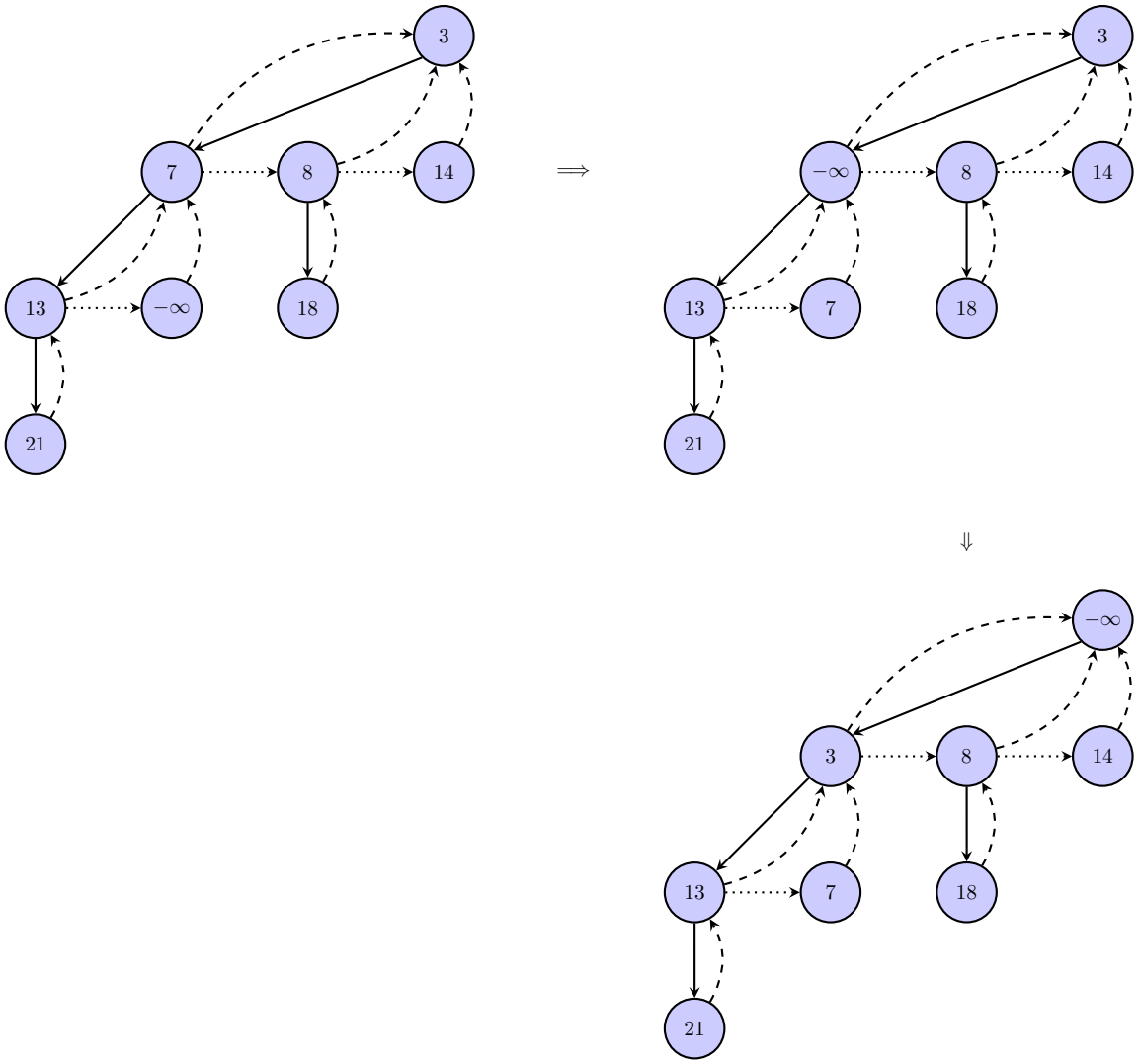



Figure 11: Löschen von 27: Ersetze 27 mit $-\infty$ und lasse diesen Knoten nach oben wandern.

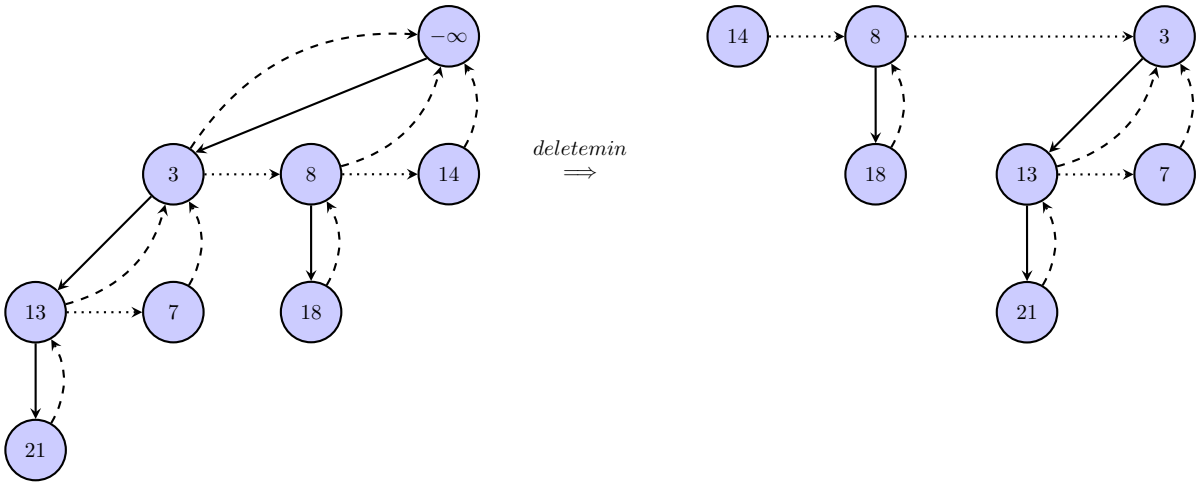



Figure 12: `deletemin()` ($-\infty$ hat minimalen Schlüssel in der Wurzelliste $\Rightarrow$ entferne $-\infty$ aus $Q$ (liefert $Q'$); Drehe Reihenfolge der Söhne von $-\infty$ um (liefert $Q''$); $Q'.meld(Q'')$). In diesem Fall ist $Q'$ leer, wodurch nurmehr $Q''$ bleibt.

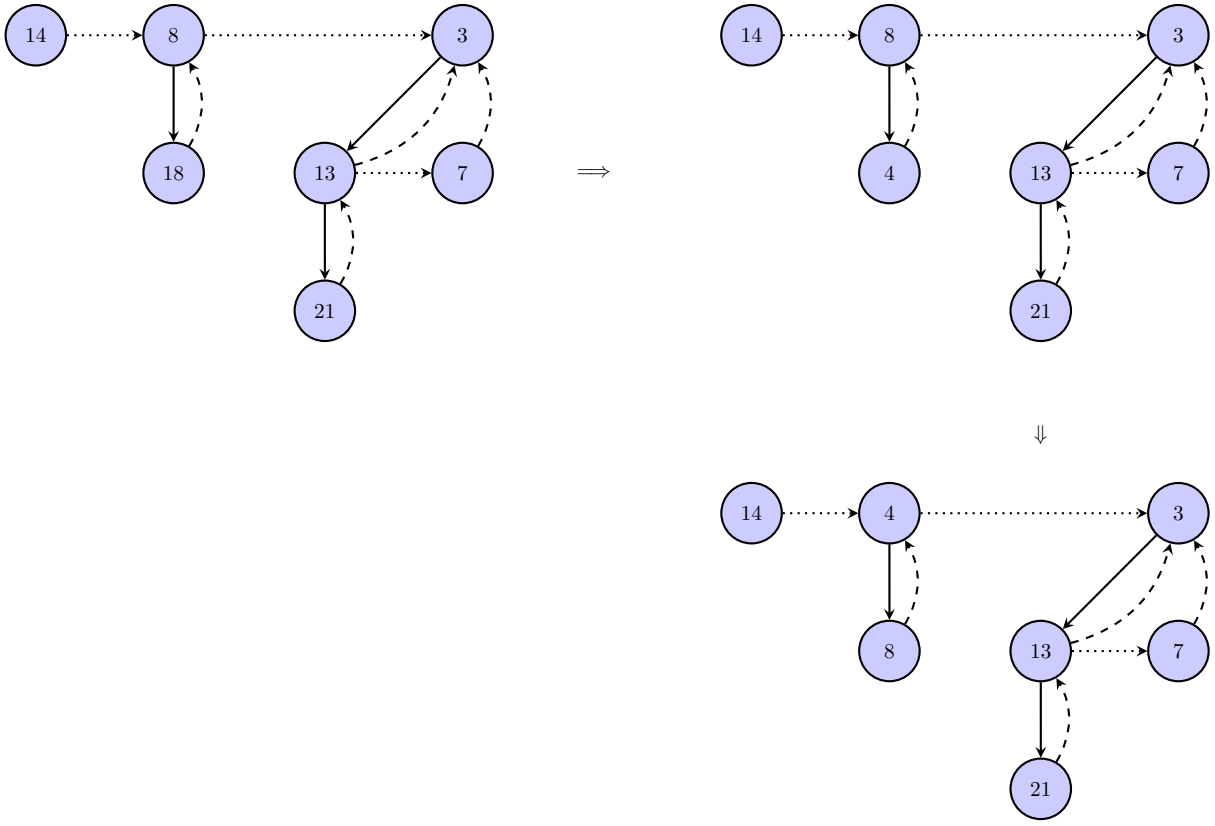
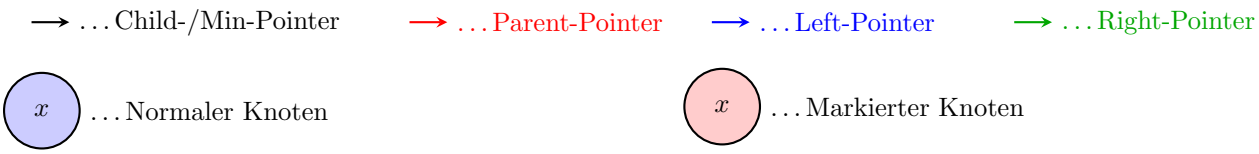



Figure 13: `decreasekey(18, 4)` (1. $v.entry.key = k$; 2. $v.entry$ nach oben steigen lassen in dem geg. Baum, bis die Heapbedingung erfüllt ist). In diesem Fall ersetzen wir 18 durch 4 und lassen 4 um eine Ebene nach oben steigen.

**Aufgabe 18**

Fügen Sie die Elemente 13, 21, 3, 7, 15, 18, 8, 14 und 27 in einen (anfangs leeren) Fibonacci-Heap ein. Entfernen Sie anschließend das Minimum und geben Sie den resultierenden Fibonacci-Heap an. Dekrementieren Sie zum Schluss den Wert eines jeden Schlüssels (Schritt für Schritt) um 7 und geben Sie die resultierende Datenstruktur an.

Im Folgenden werden die einzelnen Schritte dargestellt, wobei für die Child-Parent-Left-Right Darstellung die folgenden Pfeile und Knoten verwendet werden:




Beim Einfügen wurde immer rechts vom aktuellen Minimum eingefügt, da ansonsten dies nicht in $O(1)$ möglich wäre (es ist in den Folien zur VO nicht genau spezifiziert, an welcher Position eingefügt wird; daher diese Annahme).

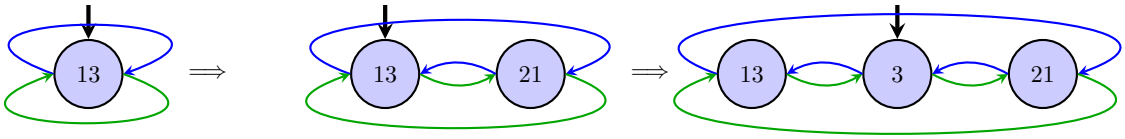



Figure 14: Einfügen von 13, 21, 3.

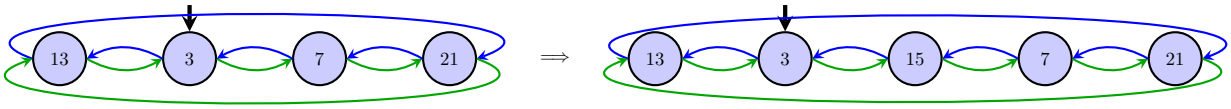



Figure 15: Einfügen von 7 und 15.

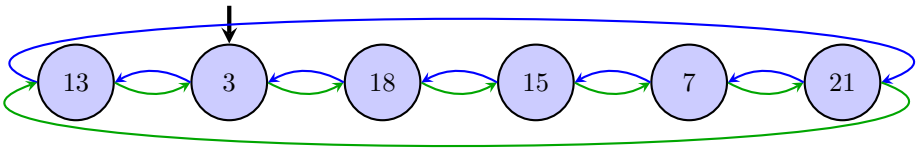



Figure 16: Nach Einfügen von 18.

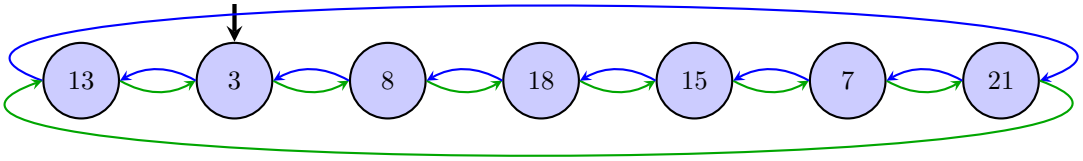



Figure 17: Nach Einfügen von 8.

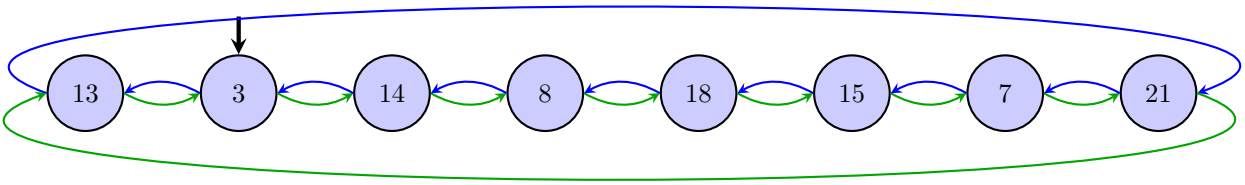



Figure 18: Nach Einfügen von 14.

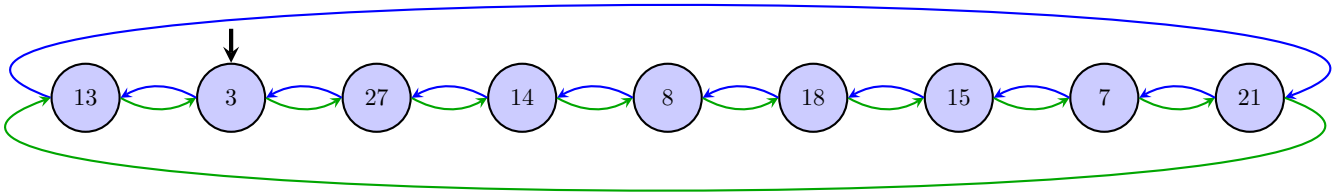



Figure 19: Nach Einfügen von 27.

Im Zuge von `deletemin()` wird `consolidate()` aufgerufen. Für `consolidate()` benötigt man ein Array der Größe $\lceil 2 \cdot \log_2 n \rceil$. Entfernt man nun das Minimum (3), dann verbleiben $n = 8$ Knoten. Wir benötigen daher ein Array der Größe $\lceil 2 \cdot \log_2 8 \rceil = \lceil 2 \cdot 3 \rceil = 6$.

Im Folgenden werden die einzelnen `consolidate()`-Schritte dargestellt, die nach dem Entfernen von 3 notwendig sind. Punktierte Pointer vom Array zu Knoten stehen dabei dafür, dass die betroffene Stelle im Array bereits besetzt ist. In diesem Fall werden die beiden Heaps, auf die die Pointer verweisen, verschmolzen.

`consolidate()` startet dabei wiederum beim rechten Knoten des zuvor entfernten Minimums - in diesem Fall also bei Knoten 27.

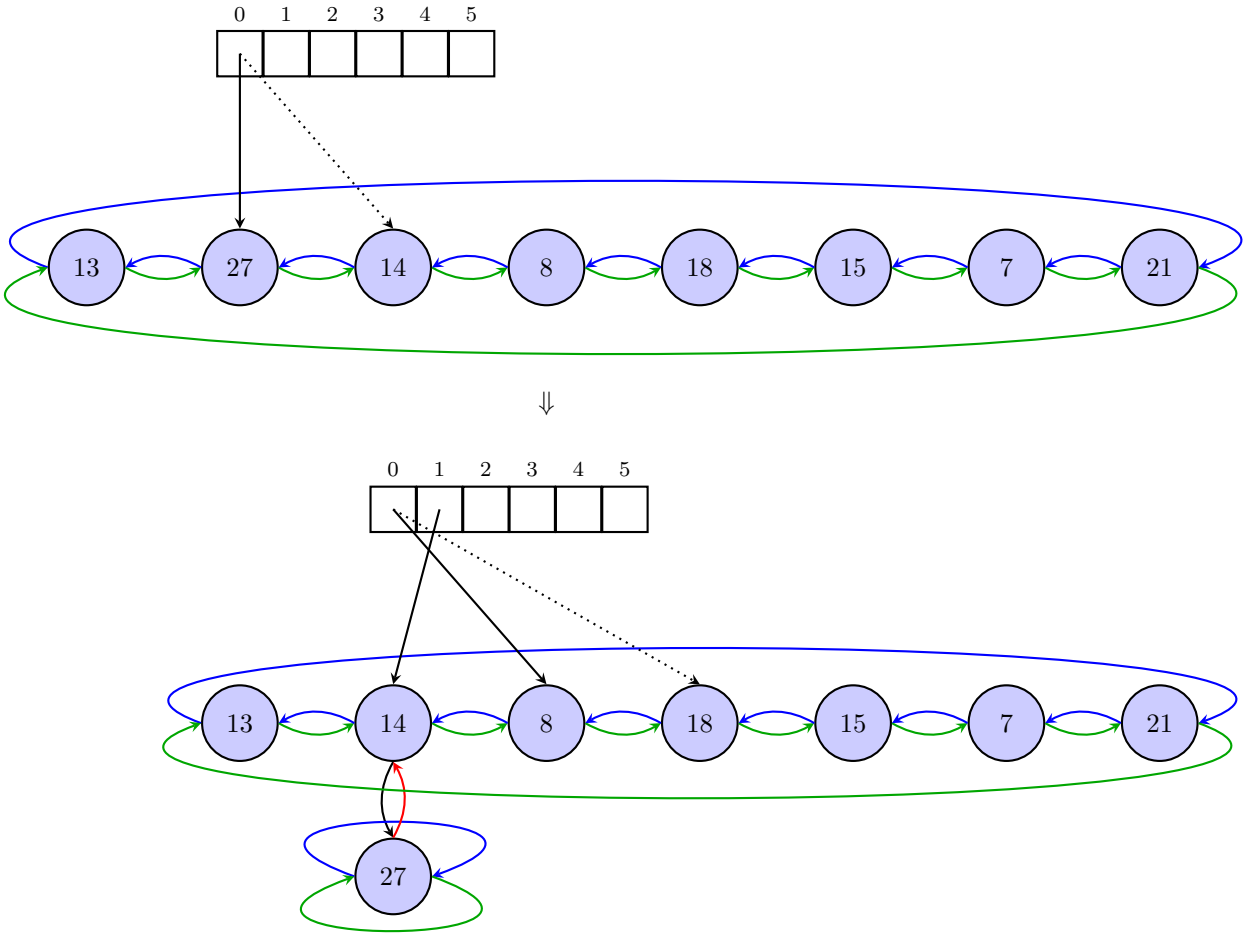



Figure 20: Nach Entfernen von 3.

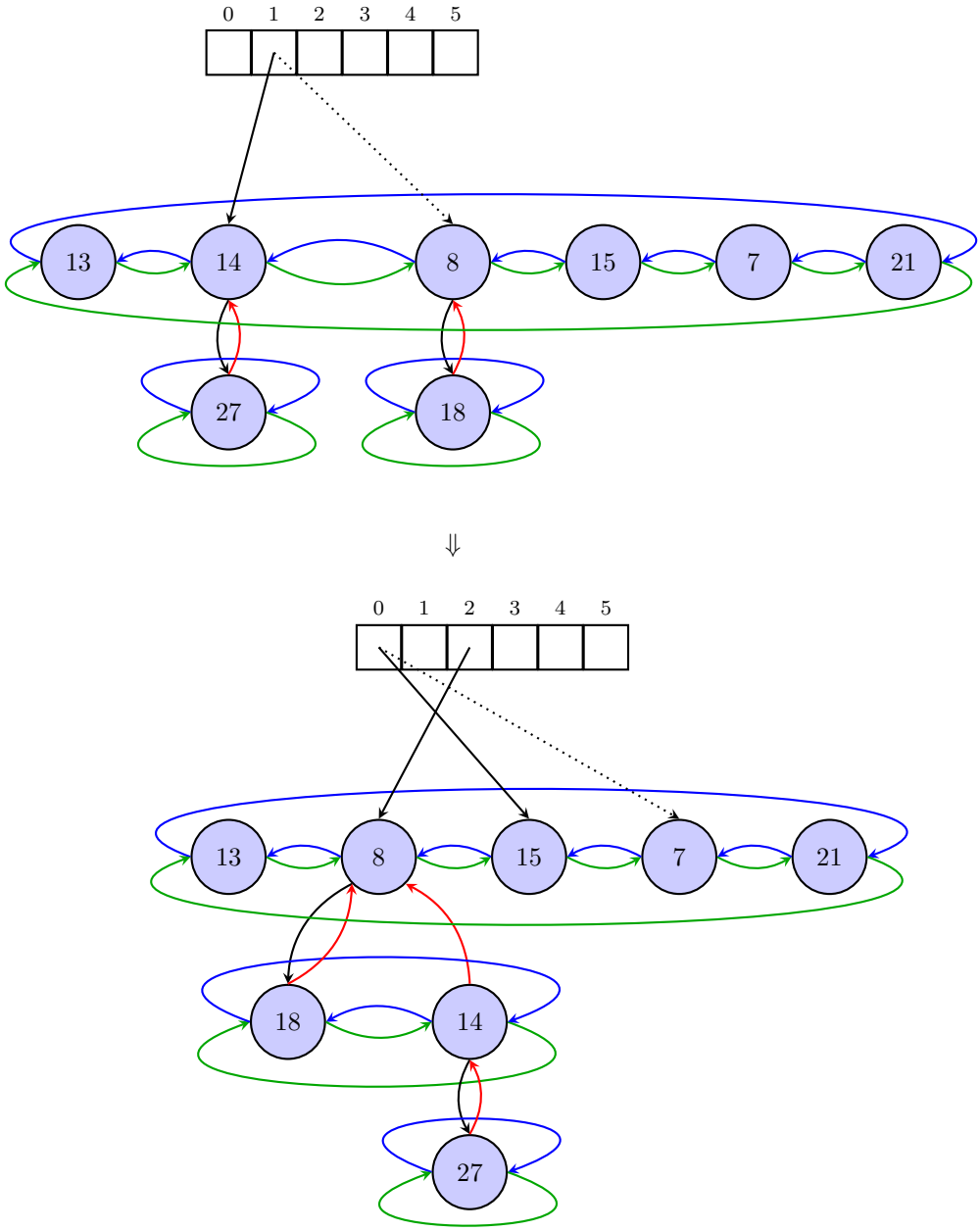



Figure 21: Nach Entfernen von 3 (Cont'd [1]).

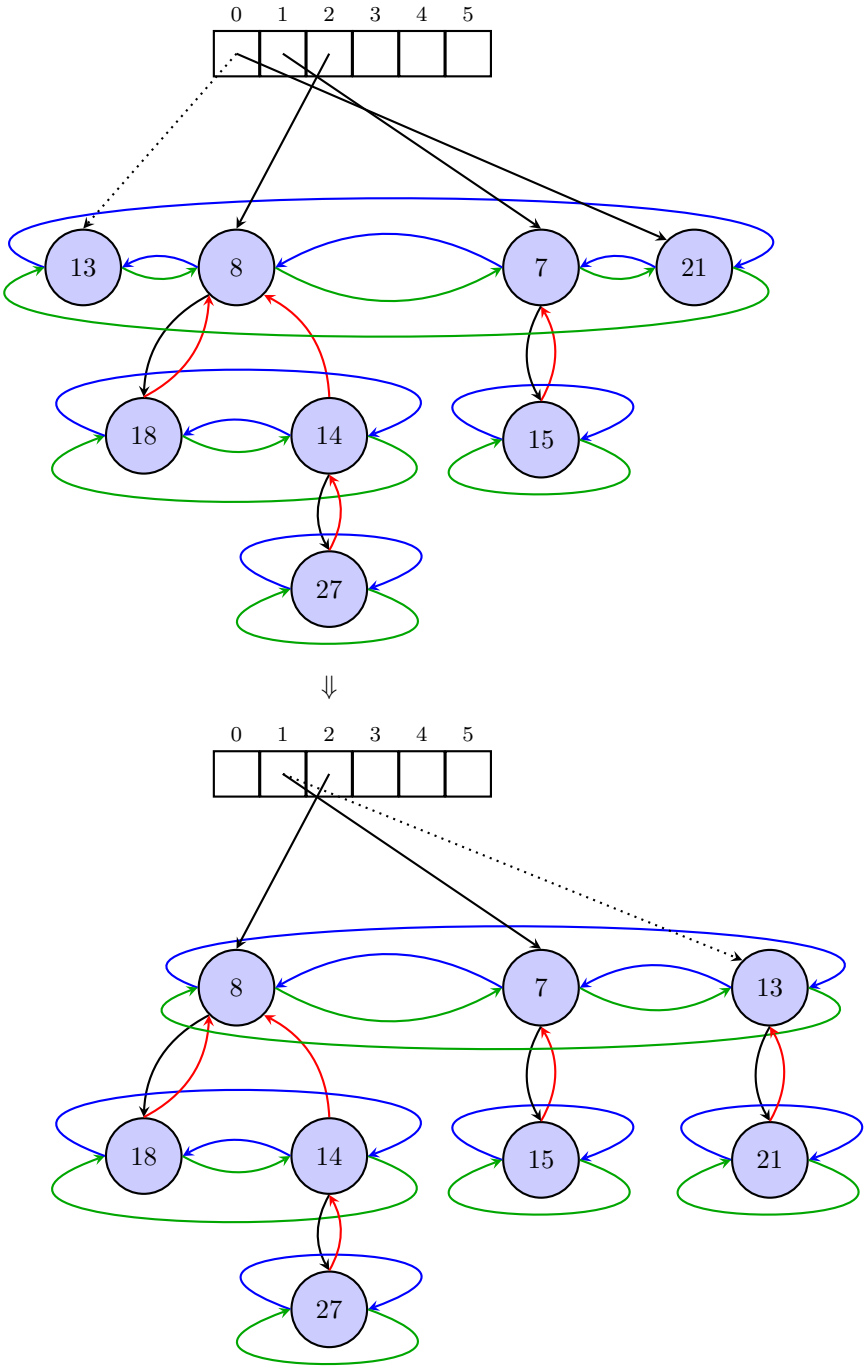



Figure 22: Nach Entfernen von 3 (Cont'd [2]).

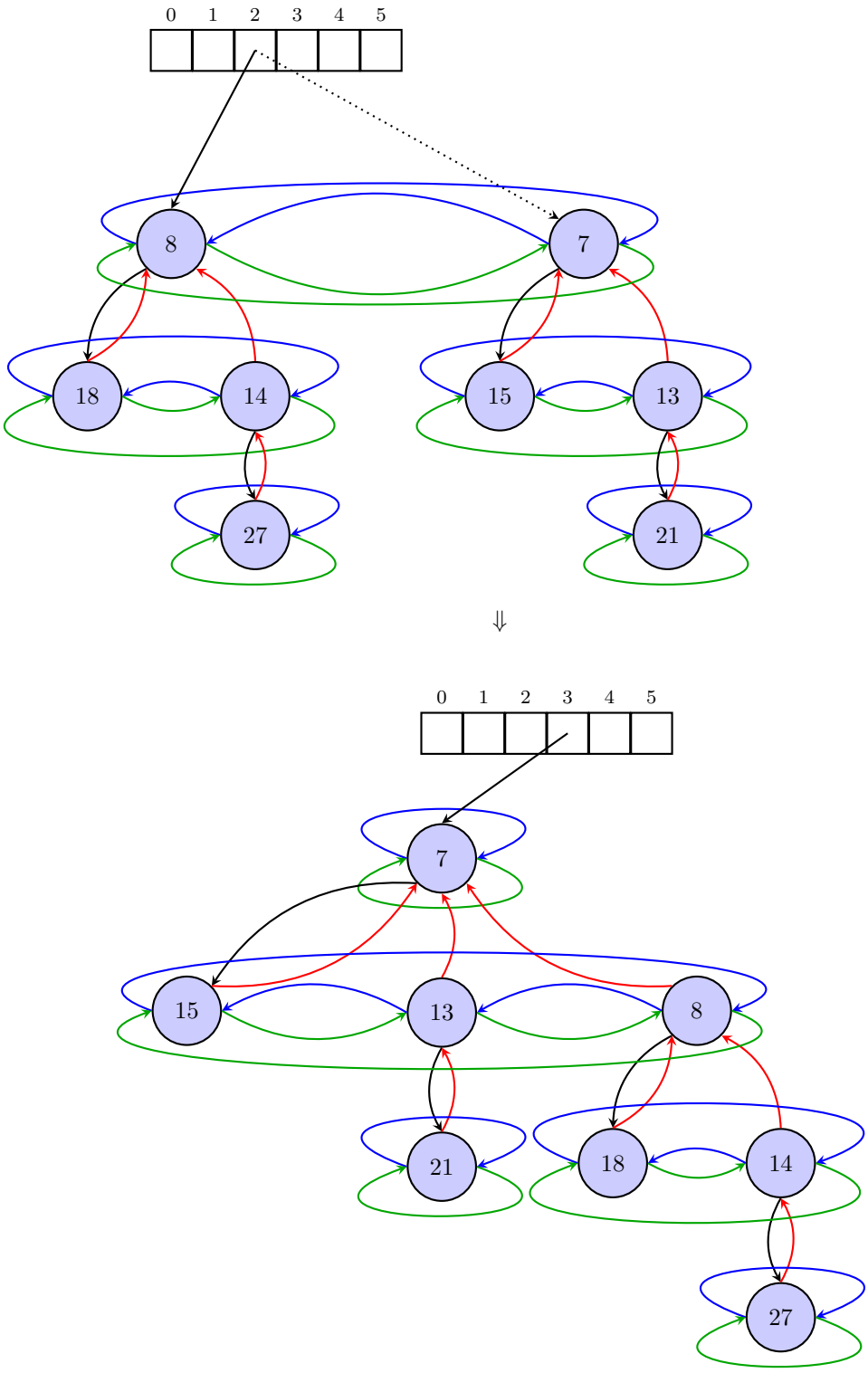



Figure 23: Nach Entfernen von 3 (Cont'd [3]).

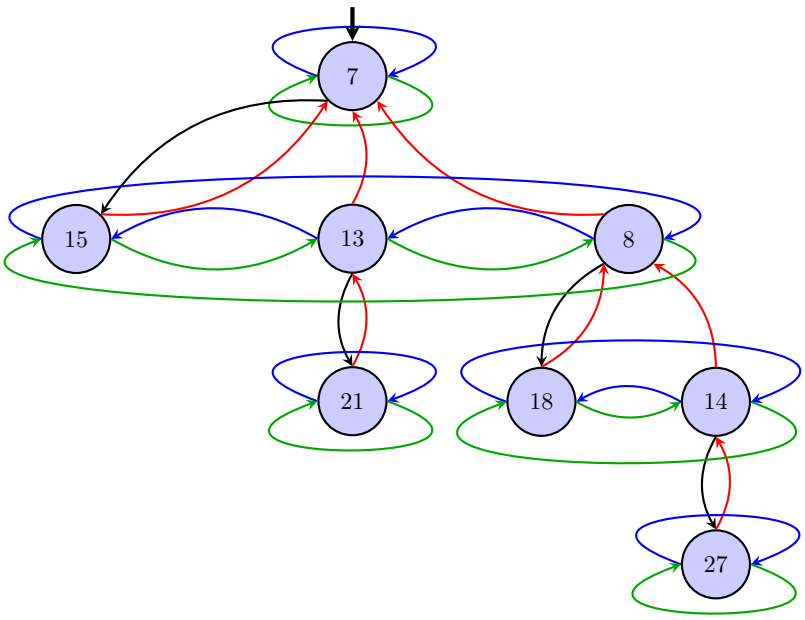



Figure 24: Finaler Fibonacci-Heap nach Entfernen von 3.

Nun wird jeder Schlüssel um 7 dekrementiert (in der Reihenfolge, in der sie eingefügt wurden, d.h. 13, 21, 7, 15, 18, 8, 14 und 27).

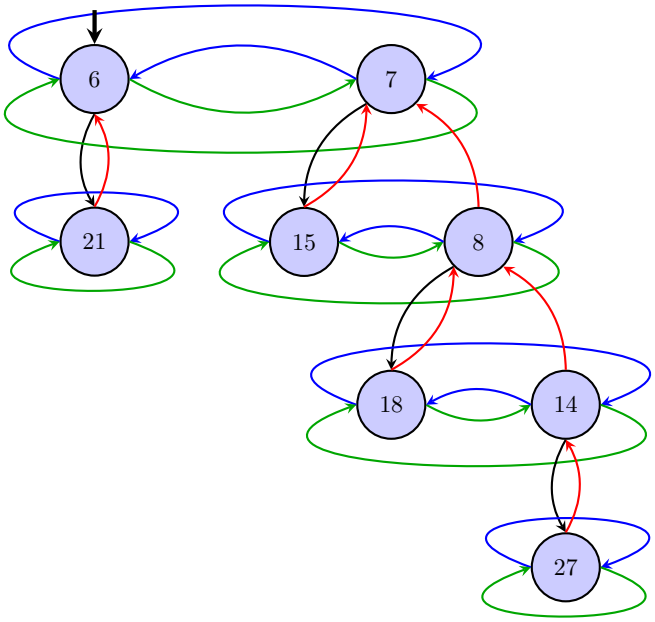



Figure 25: Nach `decreasekey(13, 6)` ($Q.min = 6$; Subtree mit Wurzel 6 wird abgetrennt; Vater von 6 wird nicht gefärbt, da 7 ein Root-Knoten ist).

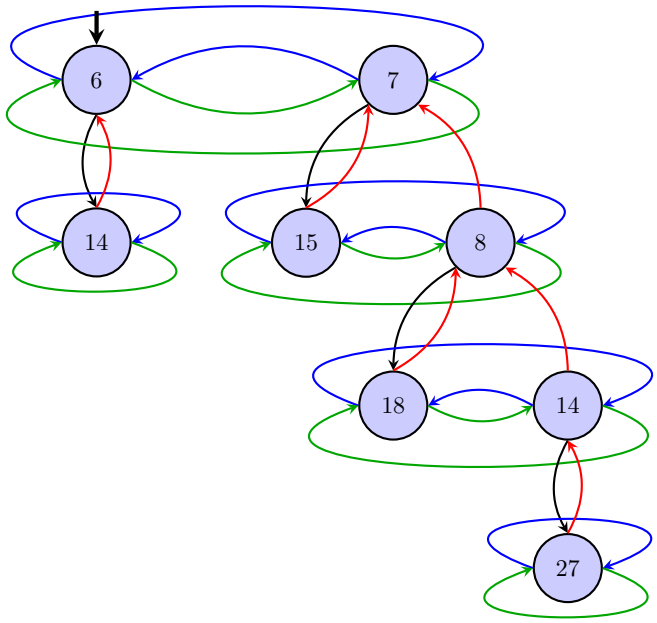



Figure 26: Nach `decreasekey(21, 14)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).

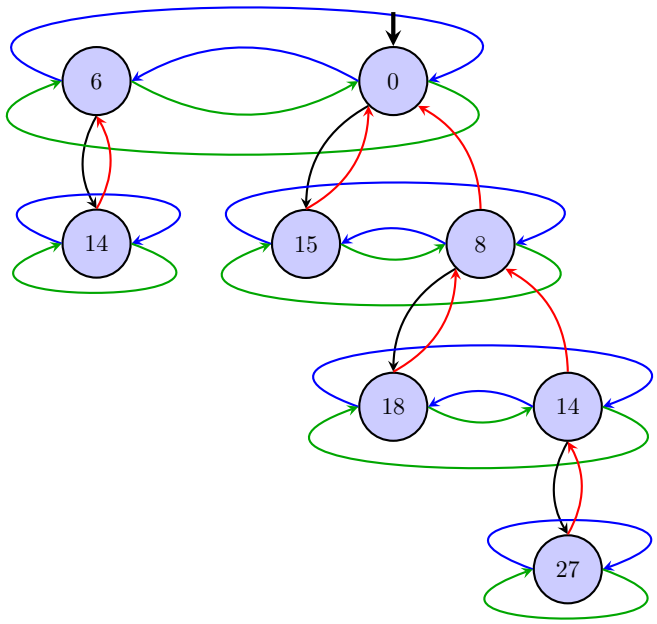



Figure 27: Nach `decreasekey(7, 0)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden; $Q.min$ wird $= 0$ gesetzt).

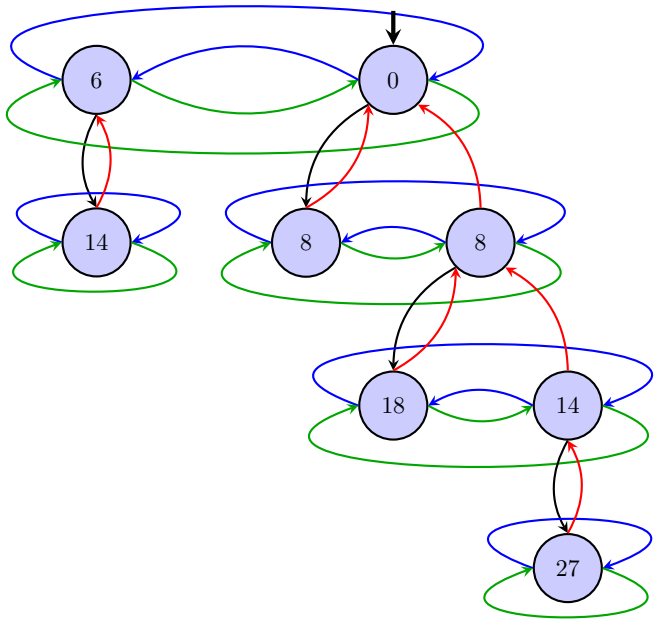



Figure 28: Nach `decreasekey(15, 8)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).

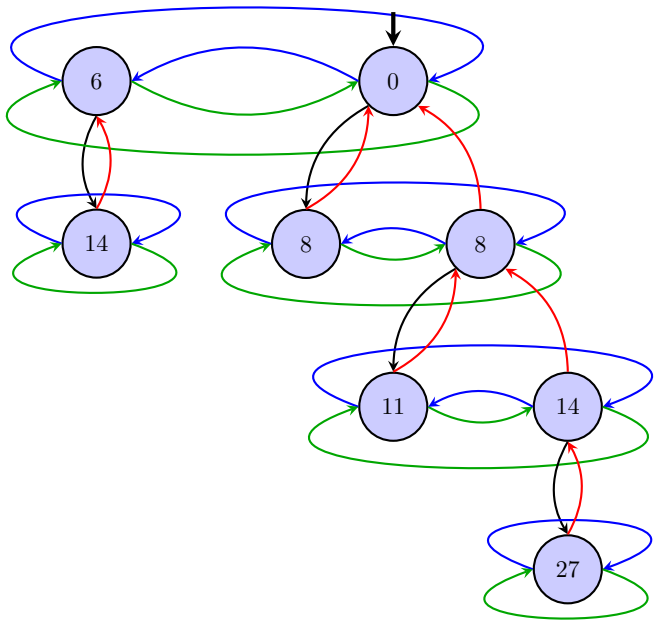



Figure 29: Nach `decreasekey(18, 11)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).

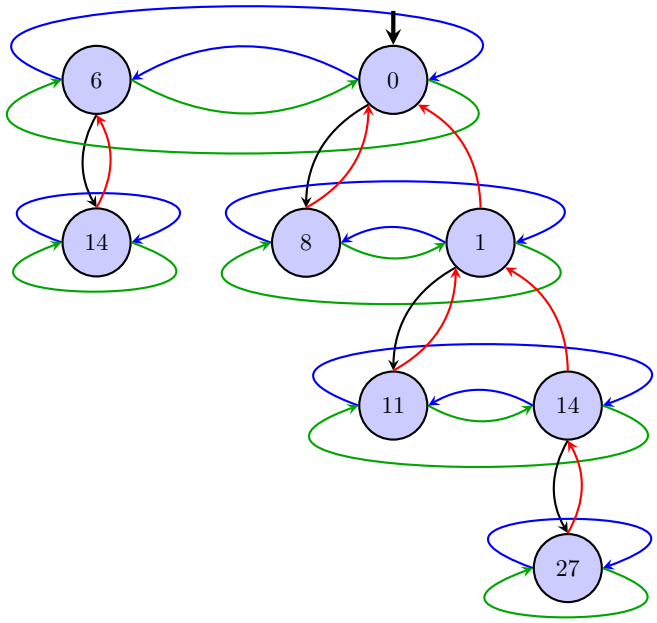



Figure 30: Nach `decreasekey(8, 1)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).

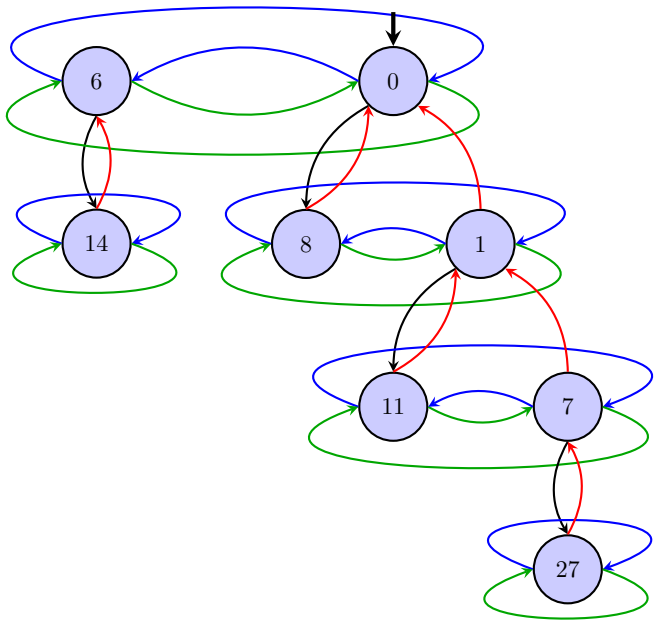



Figure 31: Nach `decreasekey(14, 7)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).

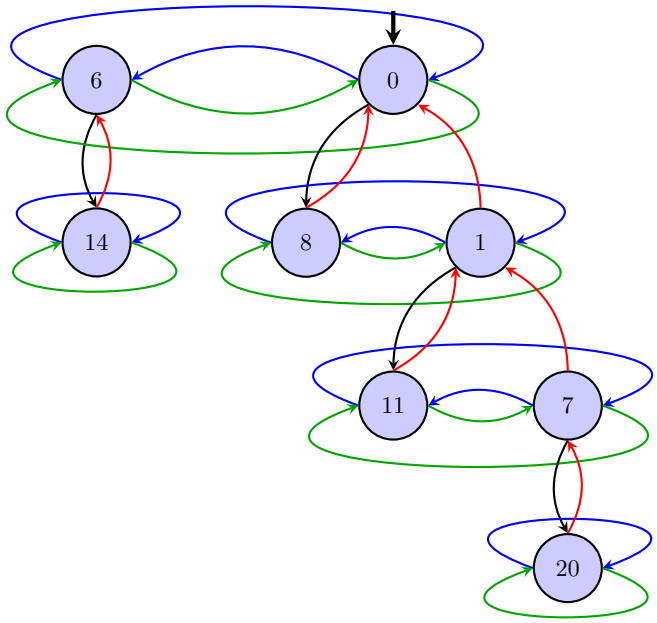



Figure 32: Nach `decreasekey(27, 20)` (Heap-Bedingung wird nicht verletzt, daher kann der Key einfach aktualisiert werden).