Assignment 11
Advanced Algorithms & Data Structures PS

Christian Müller 1123410
Daniel Kocher, 0926293

June 14, 2016

Aufgabe 20

Sei Q ein anfangs leerer Fibonacci-Heap. Geben Sie eine Folge von Schlüsseln sowie eine Folge von Operationen auf die von Ihnen gewählte Folge von Schlüsseln an, sodass der resultierende Fibonacci-Heap aus zwei Bäumen B_1 und B_2 mit jeweils $\frac{n}{2}$ Knoten besteht, wobei

- die Wurzel von B_1 mindestens Grad 4 hat und
- B_2 ein entarteter Baum ist, in dem jeder innere Knoten genau einen Sohn besitzt.

Sie dürfen davon ausgehen, dass $\frac{n}{2}$ die Form 2^k mit $k \geq 3$ hat.

Hinweis: Konstruieren Sie zunächst den Baum B_1 und anschließend den Baum B_2, sodass während der Konstruktion von B_2 der Baum B_1 sich nicht mehr ändert.

Zur Konstruktion von Baum B_1:
Es reicht hier eine Folge von 17 Aufrufen von *insert* und ein anschließender Aufruf von *deletemin*. Die Schlüssel werden dabei groß genug gewählt, sodass wir noch genug Spielraum für die Konstruktion von Baum B_2 haben.

Zur Konstruktion von Baum B_2:
Um einen entarteten Baum zu bekommen, benötigen wir die Operationen *insert*, *deletemin* sowie *decreasekey*. Zu Beginn fügen wir 3 Schlüssel ein mit anschließender *deletemin* Operation. Damit erhalten wir einen Baum mit zwei linear verketteten Elementen. Per *insert* fügen wir jeweils drei Knoten ein, wobei diese drei Knoten kleiner als das bisherige Minimum sind. Danach führen wir eine *deletemin* Operation durch. Dies hat zur Folge, dass im Zuge der *consolidate* Operation, die verbleibenden Knoten (ohne B_1) zusammengeführt werden, da beide Rang 1 besitzen. Im Anschluss wird die Operation $decreasekey(i, j)$ ausgeführt, wobei $i =$ der größte der zuvor eingefügten Schlüssel, und $j =$ der kleinste der zuvor eingefügten Schlüssel ist. Dadurch wird der Knoten von B_2 abgeschnitten und als neues Minimum in den Fibonacci-Heap eingefügt. Das anschließende *deletemin* entfernt diesen Knoten dann und es verbleiben B_1 (unverändert) und B_2 (entarteter Baum). Diese Vorgehensweise kann nun so oft wiederholt werden bis sich $\frac{n}{2}$ linear verkettete Elemente in B_2 befinden. Alle eingefügten Schlüssel müssen kleiner sein, als das Minimum des aktuellen Fibonacci-Heaps. Damit wird (1) verhindert, dass wir während der Konstruktion von B_2 den Baum B_1 verändern (wie im Hinweis erwähnt), und (2) erzwungen, dass B_2 in eine linear verkettete Liste entartet.

Daraus ergibt sich folgende Operationsfolge (Schlüssel werden in den jeweiligen Operationen angegeben).

Konstruktion von B_1:
$insert(50)$, $insert(51)$, $insert(52)$, $insert(53)$, $insert(54)$,
$insert(55)$, $insert(56)$, $insert(57)$, $insert(58)$, $insert(59)$,
$insert(60)$, $insert(61)$, $insert(62)$, $insert(63)$, $insert(64)$,
$insert(65)$, $insert(66)$, $deletemin()$.

Konstruktion von B_2:
$insert(40)$, $insert(39)$, $insert(38)$, $deletemin()$,
$insert(36)$, $insert(37)$, $insert(38)$, $deletemin()$, $decreasekey(38, 36)$, $deletemin()$,
$insert(34)$, $insert(35)$, $insert(36)$, $deletemin()$, $decreasekey(36, 34)$, $deletemin()$,
$insert(32)$, $insert(33)$, $insert(34)$, $deletemin()$, $decreasekey(34, 32)$, $deletemin()$,
$insert(30)$, $insert(31)$, $insert(32)$, $deletemin()$, $decreasekey(32, 30)$, $deletemin()$,
$insert(28)$, $insert(29)$, $insert(30)$, $deletemin()$, $decreasekey(30, 28)$, $deletemin()$,
$insert(26)$, $insert(27)$, $insert(28)$, $deletemin()$, $decreasekey(28, 26)$, $deletemin()$,
$insert(24)$, $insert(25)$, $insert(26)$, $deletemin()$, $decreasekey(26, 24)$, $deletemin()$,
$insert(22)$, $insert(23)$, $insert(24)$, $deletemin()$, $decreasekey(24, 22)$, $deletemin()$,
$insert(20)$, $insert(21)$, $insert(22)$, $deletemin()$, $decreasekey(22, 20)$, $deletemin()$,
$insert(18)$, $insert(19)$, $insert(20)$, $deletemin()$, $decreasekey(20, 18)$, $deletemin()$,
$insert(16)$, $insert(17)$, $insert(18)$, $deletemin()$, $decreasekey(18, 16)$, $deletemin()$,
$insert(14)$, $insert(15)$, $insert(16)$, $deletemin()$, $decreasekey(16, 14)$, $deletemin()$,
$insert(12)$, $insert(13)$, $insert(14)$, $deletemin()$, $decreasekey(14, 12)$, $deletemin()$,
$insert(10)$, $insert(11)$, $insert(12)$, $deletemin()$, $decreasekey(12, 10)$, $deletemin()$.

Für das Einfügen wird angenommen, dass wir immer rechts vom aktuellen Minimum einfügen. Dieselbe Annahme wurde bei `consolidate` getroffen, d.h. es wird beim rechten Zwilling der zuvor entfernten Minimums begonnen.

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

$\longrightarrow$ ... Child-/Min-Pointer $\quad$ $\longrightarrow$ ... Parent-Pointer $\quad$ $\longrightarrow$ ... Left-Pointer $\quad$ $\longrightarrow$ ... Right-Pointer

x ... Normaler Knoten $\quad$ x ... Markierter Knoten

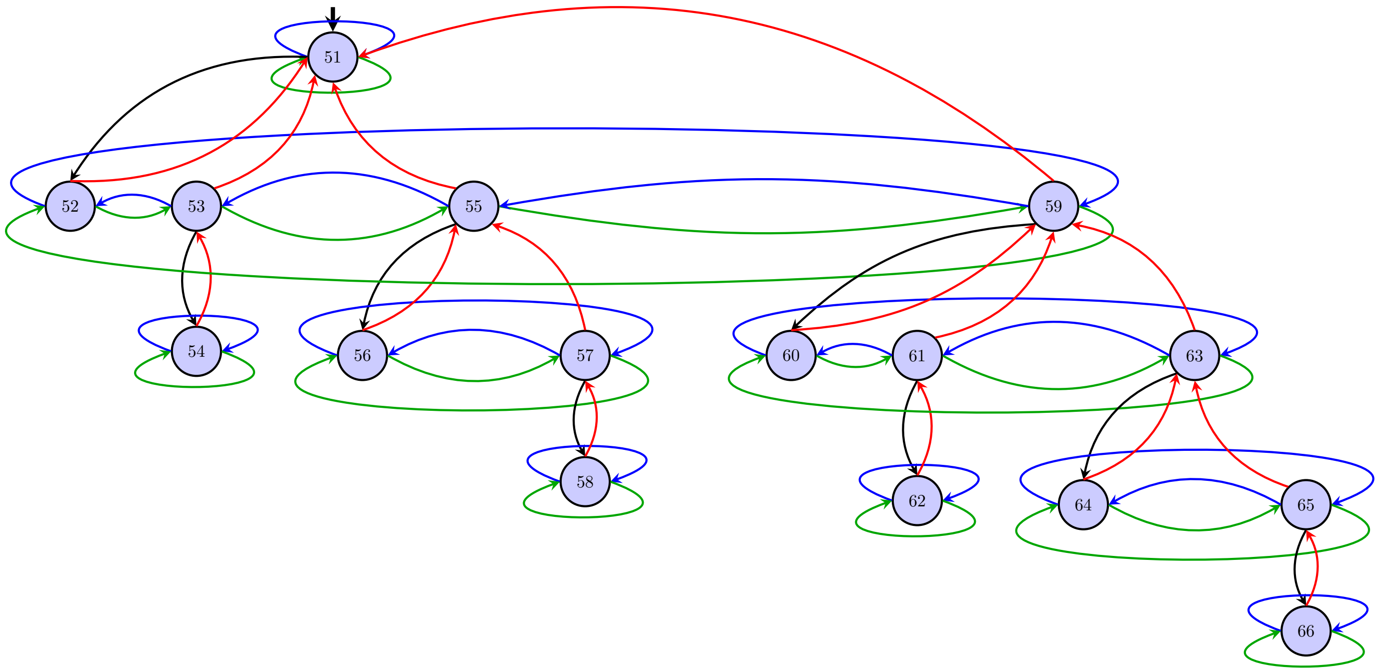

Figure 1: Baum B_1 mit $\frac{n}{2} = 16$ Knoten, wobei die Wurzel Grad 4 hat.

Konstruktion von B_1 ist abgeschlossen und die Bedingung aus der Angabe ist erfüllt. $\Longrightarrow \frac{n}{2} = 16$. Das bedeutet, wird benötigen für B_2 einen entarteten Baum mit 16 Elementen (d.h. eine linear verkettete Liste mit 16 Elementen).

Beispielhaft seien hier die ersten 3 Zeilen aus der Konstruktion von B_2 dargestellt. Der Baum B_1 wird nicht mehr explizit dargestellt, da er sich im Zuge der Konstruktion von B_2 nicht verändert.

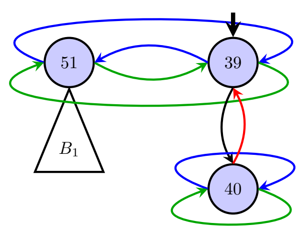

Figure 2: Nach *insert*(40), *insert*(39), *insert*(38), *deletemin*().

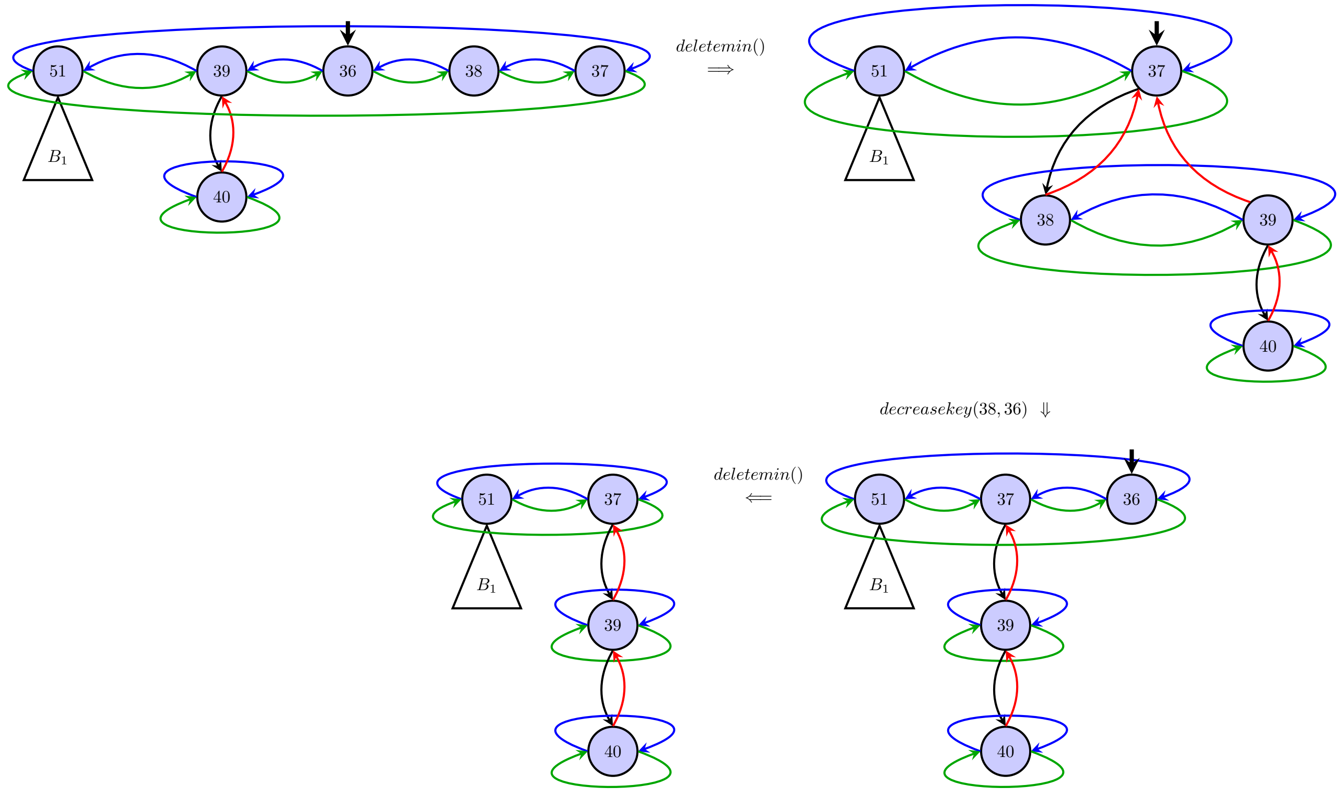

Figure 3: $insert(36)$, $insert(37)$, $insert(38)$, $deletemin()$, $decreasekey(38, 36)$, $deletemin()$.

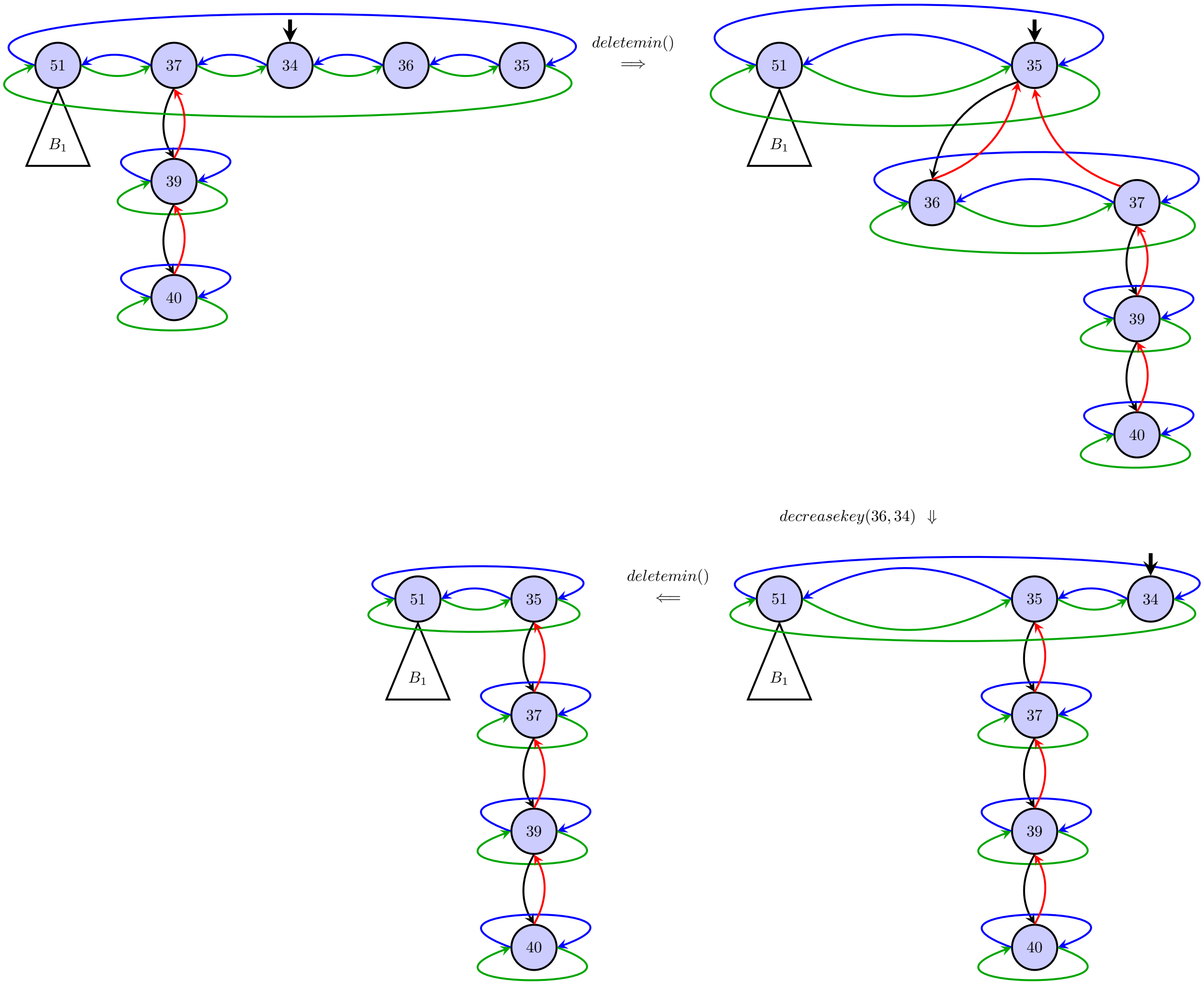

Figure 4: *insert*(34), *insert*(35), *insert*(36), *deletemin*(), *decreasekey*(36, 34), *deletemin*().

Setzt man dies nun in dieser Weise fort, dann erhält man am Ende das gewünschte Ergebnis (mit $\frac{n}{2} = 16 = 2^4$):

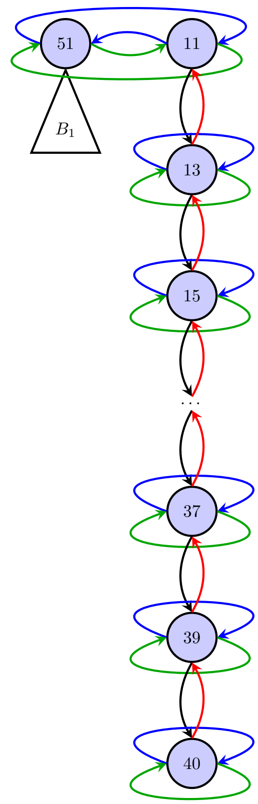

Figure 5: Gewünschtes Ergebnis: B_1 dessen Wurzel min. 4 Söhne hat und ein entarteter Baum B_2. Beide Bäume haben jeweils $\frac{n}{2} = 16$ Knoten.