Priority Queues – Zusammenfassung

$\longrightarrow$ unterstützt die Operationen `Insert`, `DecreaseKey` und `ExtractMin`

1. Fibonacci Heaps

Satz: Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit k `Insert`-/`DecreaseKey`- und ℓ `ExtractMin`–Operationen in einem Fibonacci Heap $\mathcal{O}(k + \ell \log n)$, wobei n die maximale Anzahl ELemente bezeichne, die gleichzeitig in dem Heap gespeichert sind.

2. 1–level Buckets

<u>Zusätzliche Annahmen:</u>

a) Die Schlüssel sind ganze Zahlen.

b) Zu jedem Zeitpunkt gilt für die gespeicherten Schlüssel:

$$\text{maximaler Schlüssel} - \text{minimaler Schlüssel} \leq C$$

<u>Satz:</u> Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit k `Insert`-/`DecreaseKey`- und ℓ `ExtractMin`–Operationen in einem 1–level Bucket $\mathcal{O}(k + \ell \cdot C)$.

3. 2–level Buckets

<u>Zusätzliche Annahmen:</u>

a) Die Schlüssel sind ganze Zahlen.

b) Zu jedem Zeitpunkt gilt für die gespeicherten Schlüssel:

$$\text{maximaler Schlüssel} - \text{minimaler Schlüssel} \leq C$$

c) Wenn durch `ExtractMin` ein Element mit Schlüssel k ausgegeben wurde, so werden danach nur noch Elemente mit Schlüssel $\geq k$ eingefügt.

<u>Satz:</u> Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit k `Insert`-/`DecreaseKey`- und ℓ `ExtractMin`–Operationen in einem 2–level Bucket $\mathcal{O}(k + \ell \cdot \sqrt{C})$.

4. Radix Heaps

Zusätzliche Annahmen:

a) Die Schlüssel sind nicht-negative ganze Zahlen.

b) Zu jedem Zeitpunkt gilt für die gespeicherten Schlüssel:

$$\text{maximaler Schlüssel} - \text{minimaler Schlüssel} \leq C$$

c) Wenn durch `ExtractMin` ein Element mit Schlüssel k ausgegeben wurde, so werden danach nur noch Elemente mit Schlüssel $\geq k$ eingefügt.

Satz: Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit k `Insert`- und ℓ/`DecreaseKey`-/`ExtractMin`–Operationen in einem Radix Heap $\mathcal{O}(k \cdot \log C + \ell)$.

Ausblick: Durch Kombination von Radix Heap/ 2–level Bucket und Fibinacci Heap kann man zeigen:

Satz: Ausgehend von einem leeren Heap beträgt die *worst–case*–Laufzeit k `Insert`- und ℓ `DecreaseKey`-/`ExtractMin`–Operationen in einem Radix Heap $\mathcal{O}(k \cdot \sqrt{\log C} + \ell)$.

Priority Queues – Anwendungen

Zwei Standard-Anwendungen sind:

1. Das kürzeste Wege Problem in einem Graphen: Algorithmus von Dijkstra

Mit Fibonacci Heaps: $\mathcal{O}(m + n \log n)$

Falls Kantengewichte nicht-negative ganze Zahlen und C das größte Gewicht einer Kante:

Mit 2-level Buckets: $\mathcal{O}(m + n\sqrt{C})$

Mit Radix Heaps: $\mathcal{O}(m + n \log C)$

2. Minimal spannenden Bäume in einem Graphen: Algorithmus von Prim

Mit Fibonacci Heaps: $\mathcal{O}(m + n \log n)$