Prioritetskøer

Prioritetskøer?

Datastrukturer

$$\boxed{\text{Datastruktur} = \text{data} + \text{operationer herpå}}$$

Data:

- Ofte en ID (nøgle) + associeret data.
- ID ofte fra et ordnet univers (dvs. ID'er har en ordning), f.eks. `int`, `float`, `String`.
- Vi nævner som regel ikke den associerede data—elementer er reelt (ID,data) eller (ID,reference til data), men benævnes blot som ID.

Operationer:

- Datastrukturens egenskaber udgøres af de tilbudte operationer, samt deres køretider.
- Målene er fleksibilitet og effektivitet (som regel konkurrende mål).

Tænk på en datastruktur som et API for adgang til en samling data, eller som en klasse i OO-programmering (indkapsler data, og giver adgang via metoder).

DM507: katalog af datastrukturer med bred anvendelse samt effektive implementationer heraf.

Prioritetskøer

Data:

- Element = nøgle (ID) fra et ordnet univers + associeret data.

Definerende operationer (max-version af prioritetskø):

- Q.Extract-Max: Returnerer elementet med den største nøgle i prioritetskøen Q (et vilkårligt sådant element, hvis der er flere lige store). Elementet fjernes fra Q.
- Q.Insert(e: element). Tilføjer elementet e til prioritetskøen Q.

Bemærk: vi kan sortere med disse operationer:

$$n \times \textsc{Insert}$$
$$n \times \textsc{Extract-Max}$$

Så mindst een af operationerne Insert og Extract-Max må tage $\Omega(\log n)$ for enhver implementation af prioritetskøer, som kan beskrives i den sammenligningsbaserede model.

Prioritetskøer

Ekstra operationer:

- Q.IncreaseKey(r: reference til et element i Q, k nøgle). Ændrer nøglen til $\max\{k, \text{gamle nøgle}\}$ for elementet refereret til af r.
- Q.Build(L: liste af elementer). Bygger en prioritetskø indeholdende elementerne.

Trivielle operationer for alle datastrukturer:

- Q.IsEmpty(), Q.CreateNew(), Q.RemoveEmpty().

Vil ikke blive nævnt fremover.

Implementation via Heaps

En mulig implementation: brug heapstrukturen fra Heapsort. (Flere implementationer mødes f.eks. i DM508).

(NB: Arrayversionen af heaps kræver et kendt maximum for størrelsen n af køen. Alternativt kan array erstattes af et extendible array (se kurset DM508), f.eks. `java.util.ArrayList` i Java, eller man kan implementere heaptræet pointerbaseret.)

Vi har allerede:

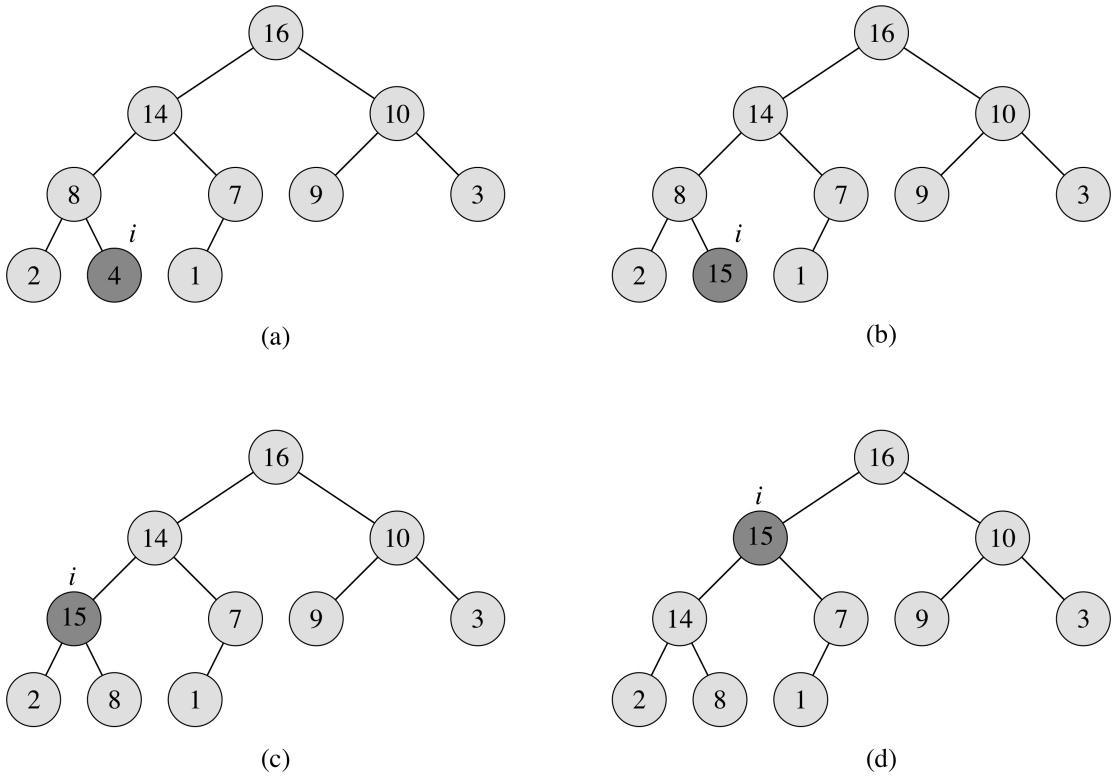
- Extract-Max: Er essentielt hvad der bruges under anden del af Heapsort – fjern rod, flyt sidste blad op som rod, kald Heapify. Køretid: $O(\log n)$.
- Build: Brug Heapify gentagne gange bottom-up. Køretid: $O(n)$.

Mangler:

- Insert
- Increase-Key

Increase-Key

1. Ændre nøgle for element.
2. Genopret heaporden.



Køretid: $O(\log n)$.

Insert

1. Indsæt nye element sidst ($\Rightarrow$ heapfacon i orden).
2. Genopret heaporden som i Increase-Key.

Køretid: $O(\log n)$.