Übersicht

Heute relativ viel Stoff:

- Bäume
- Sortierte Folgen
- Graphen

Bäume

Binäre Suchbäume

- Suchen, Einfügen, Entfernen in $O(\log n)$
- Problem: Balancierung
  - Rot-Schwarz-Baum
- Problem: $O(\log n)$ Pointer-Indirektionen (Caches!)
- Vorteil: Referentielle Integrität

a,b-Bäume

- Knoten mit a bis b Knoten
- Cache-effizienter
- Binärsuche in Knoten: Laufzeiten bleiben erhalten
- Balancierung nach wie vor nötig
  - Teilen und verbinden von inneren Knoten

Sortierte Folgen

Übersicht

- Suchbaum + doppelt-verlinkter Liste
- Viele Operationen die nicht toll, aber oft gut genug sind
  - Suchen: $O(\log n)$
  - Iteration durch n Elemente: $O(n)$
  - Einfügen/Entfernen: $O(\log n)$
  - Konkatenation und Aufteilen in $O(\log n)$

Graphen

Übersicht

- Prinzipien bekannt aus GBI
- gerichtet vs. ungerichtet (hier: primär gerichtet)
- gewichtet vs. ungewichtet (zunächst ungewichtet)

Adjazenzmatrix

- Bekannt aus GBI

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Viel Speicherverschwendung bei dünnen Graphen

Adjazenzliste

- Dynamisches Array von verlinkten Listen
- Array: Knoten
- Liste: Indexe von verbundenen Knoten (=Kanten)

```
[
    [1,2,3], // Node 0
    [0],     // Node 1
    [1,3],   // Node 2
    [0,1]    // Node 3
]
```

Adjazenzfeld

- Knotenarray + Kantenarray
- Knotenarray speichert Start/End-index im Kanten-Array
- KantenArray speichert verbundene Knoten

```
Nodes: [0,    2, 3,    5,   (7)]
Edges: [1, 2, 2, 0, 1, 1, 2]
```

Kreativaufgabe

Aufgabe

Entwerft ein *dynamisiertes Adjazenzarray*:

- Eindeutige und stabile KnotenIDs aus $\mathbb{N}_0$
- Eindeutige KantenIDs (nicht unbedingt aus $\mathbb{N}_0$)
- Zugriff auf Knoten und Kanten in $O(1)$ (ID $\rightarrow$ Handle)
- `firstEdge(NodeID)` und `nextEdge(EdgeID)` in $O(1)$
- Einfügen von Knoten und Kanten in amortisiert $O(1)$
- Entfernen von Knoten und Kanten in amortisiert $O(1)$

Wie hoch ist der Speicherverbrauch?

Lösung

- dynamisches Array von dynamischen Arrays
- gelöschte Knoten als Löcher (zwecks Wiederverwendung in einfach verlinkter Liste zu speichern)

Ähnlich zu dem, was in der Praxis etwa von NetworKit (Graphenbibliothek des ITI Meyerhenke) verwendet wird.