Heaps, Binäre Suchbäume, AVL-Bäume, a, b-Bäume

Tolga Tel

27.06.2019

Heaps

- Binärbaum mit Heapstruktur
  - jeder Knoten der Tiefe höchstens t-2 hat genau zwei Kinder
  - wenn ein Knoten **v** der Tiefe t-1 weniger als zwei Kinder hat, haben alle Knoten rechts von v bei Tiefe t-1 keine Kinder
  - wenn ein Knoten v Tiefe t-1 genau ein Knid hat, ist dies ein linkes Kind
- speichert Prioritäten gemäß der Heapordnung ab
  - Für jeden Knoten v und für jedes Kind w von v gilt: $p(v) \geq p(w)$

Arraydarstellung

Array H ist ein Heap für T, wenn

- H[1] = p(r) für Wurzel r von T und
- wenn H[i] die Priorität des Knotens v von T speichert:
  - linkes Kind: $H[2 \cdot i] = P(v_L)$
  - rechtes Kind: $H[2 \cdot i + 1] = P(v_R)$

Insert + Repair_up

Insert: Neue Priorität p wird am Ende des Arrays eingefügt → Heap-Struktur wird eingehalten, aber Heap-Ordnung könnte verletzt sein.

Repair_up: Wir verschieben die Priorität so lange nach oben, bis:

- Prioriät des Elternknotens mindestens genauso groß ist
- die Wurzel erreicht ist

Delete_max + Repair_down

Delete_max: Heap mit n Prioritäten:

1. Überschreibe die Wurzel mit H[n]
2. verringere n um 1

Repair_down:

- Priorität p wird mit der Priorität des größten Kindes verglichen und gegebenenfalls vertauscht
- endet, wenn die Position passt oder das Blatt erreicht wurde

Change_Priority

Wenn die Priorität ansteigt $\rightarrow$ repair_up
Wenn die Priorität fällt $\rightarrow$ repair_down

Binäre Suchbäume

Daten(v) = (Schlüssel(v), Info(v))
Eigenschaften:

- Für jeden Schlüsselwert x gibt es höchstens einen Knoten v mit Schlüssel(v) = x
- Für jeden Knoten v, jeden Knoten V_{LINKS} im linken Teilbaum von v und jeden Knoten v_{RECHTS} im rechten Teilbaum von v gilt:

  $$\text{Schlüssel}(v_{LINKS}) \leq \text{Schlüssel}(v) \leq \text{Schlüssel}(v_{RECHTS})$$

lookup + insert

lookup(x):

1. Beginn der Suche an der Wurzel
2. Vergleich x und Schlüssel(v)
   - x = Schlüssel(v) $\rightarrow$ Schlüssel gefunden
   - x < Schlüssel(v) $\rightarrow$ linker Teilbaum
   - x > Schlüssel(v) $\rightarrow$ rechter Teilbaum

insert(x): Suche nach Schlüssel x

- x gefunden $\rightarrow$ überschreibe Infoteil
- x nicht gefunden $\rightarrow$ füge Schlüssel an der Stelle ein, an der die Suche abbricht

remove

Suche nach Schlüssel x. Suche endet in Knoten v.

- v ist ein Blatt $\rightarrow$ entferne Blatt
- v hat genau ein Kind w $\rightarrow$ entferne v, der Elternknoten von v ist jetzt der Elternknoten von w
- v hat zwei Kinder $\rightarrow$ Ersetze v durch den kleinsten Schlüssel s im rechten Teilbaum von v

AVL-Bäume

- ein binärer Suchbaum
- für jeden Knoten v mit linkem Teilbaum $T_L(v)$ und rechtem Teilbaum $T_R(v)$:

$$| \text{Tiefe}(T_L(v)) - \text{Tiefe}(T_R(v)) | \leq 1$$

  - **Balance-Grad:** $b(v) = \text{Tiefe}(T_L(v)) - \text{Tiefe}(T_R(v))$

(a, b)-Eigenschaft

- a $\geq$ 2, b $\geq$ 2a-1
- alle Blätter von T haben die gleiche Tiefe
- alle Knoten haben höchstens b Kinder
- Wurzel hat mindestens 2 Kinder, alle anderen Knoten haben mindestens a Kinder

(a, b)-Bäume

- jeder Schlüssel wird in genau einem Knoten von T gespeichert
- ein Knoten speichert Schlüssel in aufsteigender Reihenfolge
- jeder Knoten mit k Kindern speichert genau k-1 Schlüssel
- ein Blatt speichert a-1 bis b-1 Schlüssel

Insert(x)

Suche nach x:

- x gefunden $\rightarrow$ überschreibe Infoteil
- x nicht gefunden $\rightarrow$ Suche endet in Blatt v $\rightarrow$ füge x in v ein
  - Fall 1: v hat maximal b-1 Schlüssel $\rightarrow$ (a, b)-Eigenschaft erfüllt
  - Fall 2: v hat b Schlüssel $\rightarrow$ (a, b)-Eigenschaft verletzt
    $\rightarrow$ ersetze v durch zwei Knoten:
    - v_{LINKS}(Schlüssel: $x_1, ..., x_{\lceil b/2 \rceil - 1}$)
    - v_{RECHTS}(Schlüssel: $x_{\lceil b/2 \rceil + 1}, ..., x_b$)
  - Schlüssel $x_{\lceil b/2 \rceil}$ unterscheidet v_{LINKS} und v_{RECHTS} und ist im Elternknoten enthalten

Remove(x)

- Suche nach x
  - x in innerem Knoten → ersetze x durch den kleinsten Schlüssel y mit x≤y
  - x in Blatt → entferne x
- Fall 1: v hat min. a-1 Schlüssel → (a, b)-Eigenschaft erfüllt
- Fall 2: v hat a-2 Schlüssel
  - Schlüsselklau von Elternknoten, wenn möglich
    Fall 2.1: Der linke oder rechte Geschwisterknoten hat mindestens a Schlüssel
    - linker Geschwisterknoten v' hat Schlüssel $y_1 < ... < y_{k'}$
    - Schlüssel z im Elternknoten trennt v' und v:
      v klaut z und hat a-1 Schlüssel
      Schlüssel z wird durch Schlüssel $y_{k'}$ ersetzt
  - Sonst: Fusion mit Geschwisterknoten
    Fall 2.2: beide Geschwisterknoten besitzen a-1 Schlüssel
    - linker Geschwisterknoten v' hat Schlüssel $y_1 < ... < y_{a-1}$
    - verschmelze v und v'
      Fusionierter Knoten hat (a-1)+(a-2)+1 = 2a-2 ≤ b-1
      Schlüssel z wird aus Elternknoten genommen