UA RUHR

Algorithmen auf Sequenzen

Eine Rank-Datenstruktur für Bitsequenzen

Sven Rahmann

Genominformatik
Universitätsklinikum Essen
Universität Duisburg-Essen
Universitätsallianz Ruhr

Rank-Anfragen

UA RUHR

- Sei s eine Bitsequenz.
- Die Anzahl der Eins-Bits in $s[:i]$ wird $rank_s(i)$ genannt.
- (Varianten: Bit i wird mitgezählt oder nicht)
- **Ziel:** Effiziente Berechnung von $rank_s(i)$ für alle i
- Anwendung: Abbilden von großen dünnbesiedelten Arrays auf kleine komplett befüllte Arrays, wichtige Beispiele später

Einfache Rank-Algorithmen

UA RUHR

- $rank_s(i)$: Anzahl der Einsen in $s[:i]$
- Langsam, aber leichtgewichtig: Schleife
  $\mathcal{O}(n)$ Zeit, $\mathcal{O}(1)$ zusätzlicher Speicher
- Etwas schneller: Schleife mit popcount
  $\mathcal{O}(n/W)$ Zeit, $\mathcal{O}(1)$ zusätzlicher Speicher
- Schnell, aber schwergewichtig: Tabelle
  $\mathcal{O}(1)$ Zeit, aber $\mathcal{O}(n \log n)$ Bits zusätzlich

Einfache Rank-Algorithmen

UA RUHR

- $rank_s(i)$: Anzahl der Einsen in $s[:i]$
- Langsam, aber leichtgewichtig: Schleife
  $\mathcal{O}(n)$ Zeit, $\mathcal{O}(1)$ zusätzlicher Speicher
- Etwas schneller: Schleife mit popcount
  $\mathcal{O}(n/W)$ Zeit, $\mathcal{O}(1)$ zusätzlicher Speicher
- Schnell, aber schwergewichtig: Tabelle
  $\mathcal{O}(1)$ Zeit, aber $\mathcal{O}(n \log n)$ Bits zusätzlich
- Gut wäre $\mathcal{O}(1)$ Zeit, $o(n)$ Bits zusätzlich, d.h.
  ist $x(n)$ der zusätzliche Platzbedarf neben n Bits für s selbst,
  dann soll $x(n)/n \to 0$ für $n \to \infty$ gelten.

Eine effiziente Rank-Datenstruktur

- Grundidee: Rank-Tabelle für jeden S-ten Eintrag;
  je S Bits bilden einen "Superblock":
  $\mathcal{O}(\log n \cdot n/S)$ Bits für Suberblock-Tabelle

Eine effiziente Rank-Datenstruktur

UA RUHR

- Grundidee: Rank-Tabelle für jeden S-ten Eintrag;
  je S Bits bilden einen "Superblock":
  $\mathcal{O}(\log n \cdot n/S)$ Bits für Suberblock-Tabelle
- Wähle $S = \Theta((\log n)^2)$;
  somit $\mathcal{O}(n/\log n) = o(n)$ Bits für Superblock-Tabelle
- Verbleibendes Problem:
  Rank auf "Superblöcken" der Größe S zählen

Eine effiziente Rank-Datenstruktur

- Grundidee: Rank-Tabelle für jeden S-ten Eintrag;
  je S Bits bilden einen "Superblock":
  $\mathcal{O}(\log n \cdot n/S)$ Bits für Suberblock-Tabelle
- Wähle $S = \Theta((\log n)^2)$;
  somit $\mathcal{O}(n/\log n) = o(n)$ Bits für Superblock-Tabelle
- Verbleibendes Problem:
  Rank auf "Superblöcken" der Größe S zählen
- Damit: Laufzeit $O(\log^2 n)$, Speicher $o(n)$

Eine effiziente Rank-Datenstruktur

- Verfeinerung: Jeder Superblock wird in $\Theta(\log n)$ Blöcke der Größe $\Theta(\log n)$ unterteilt.
- Jeder Superblock hat Tabelle mit Ranks für jeden Block-Beginn:
  Werte bis $\Theta(\log^2 n)$ benötigen $\mathcal{O}(\log \log n)$ Bits.
  Es gibt $\Theta(\log n \cdot n/S) = \Theta(n/\log n)$ viele Blöcke.
  Insgesamt also $\mathcal{O}(n \log \log n/\log n) = o(n)$ Bits.

Eine effiziente Rank-Datenstruktur

UA RUHR

- Verfeinerung: Jeder Superblock wird in $\Theta(\log n)$ Blöcke der Größe $\Theta(\log n)$ unterteilt.
- Jeder Superblock hat Tabelle mit Ranks für jeden Block-Beginn:
  Werte bis $\Theta(\log^2 n)$ benötigen $\mathcal{O}(\log \log n)$ Bits.
  Es gibt $\Theta(\log n \cdot n/S) = \Theta(n/\log n)$ viele Blöcke.
  Insgesamt also $\mathcal{O}(n \log \log n/\log n) = o(n)$ Bits.
- popcount innerhalb eines Blocks mit $\Theta(\log n)$ Bits kann in konstanter Zeit berechnet werden (RAM-Modell)

Eine effiziente Rank-Datenstruktur

UA RUHR

- Verfeinerung: Jeder Superblock wird in $\Theta(\log n)$ Blöcke der Größe $\Theta(\log n)$ unterteilt.
- Jeder Superblock hat Tabelle mit Ranks für jeden Block-Beginn:
  Werte bis $\Theta(\log^2 n)$ benötigen $\mathcal{O}(\log \log n)$ Bits.
  Es gibt $\Theta(\log n \cdot n/S) = \Theta(n/\log n)$ viele Blöcke.
  Insgesamt also $\mathcal{O}(n \log \log n/\log n) = o(n)$ Bits.
- popcount innerhalb eines Blocks mit $\Theta(\log n)$ Bits kann in konstanter Zeit berechnet werden (RAM-Modell)
- Berechnung benötigt konstante Zeit:
  Superblock-Rank + Block-Rank + Block-popcount

Praxis

- RAM-Modell: popcount von $O(\log n)$ Bits in konstanter Zeit
- Praxis: popcount von 64 Bits in konstanter Zeit, $n \leq 2^{64}$
- Wähle $S := 16 \cdot (64)^2 = 65536 = 2^{16}$
- 64-bit-ints für Superblock-Ranks, 16-bit-ints für Block-Ranks

Praxis

- RAM-Modell: popcount von $O(\log n)$ Bits in konstanter Zeit
- Praxis: popcount von 64 Bits in konstanter Zeit, $n \leq 2^{64}$
- Wähle $S := 16 \cdot (64)^2 = 65536 = 2^{16}$
- 64-bit-ints für Superblock-Ranks, 16-bit-ints für Block-Ranks
- $n/2^{16}$ Superblöcke mit 64-Bit-Werten
- Jeder Superblock hat 1024 Blöcke mit 16-Bit-Werten
- Insgesamt: $n/65536 \cdot (64 + 1024 \cdot 16) \approx 0.25 \cdot n$ Bits

Praxis

UA RUHR

- RAM-Modell: popcount von $O(\log n)$ Bits in konstanter Zeit
- Praxis: popcount von 64 Bits in konstanter Zeit, $n \leq 2^{64}$
- Wähle $S := 16 \cdot (64)^2 = 65536 = 2^{16}$
- 64-bit-ints für Superblock-Ranks, 16-bit-ints für Block-Ranks
- $n/2^{16}$ Superblöcke mit 64-Bit-Werten
- Jeder Superblock hat 1024 Blöcke mit 16-Bit-Werten
- Insgesamt: $n/65536 \cdot (64 + 1024 \cdot 16) \approx 0.25 \cdot n$ Bits
- für verschiedene Werte von n optimierbar