

Roaring bitmaps

Justin Schartner

15. Juni 2022

Struktur

- 1** Allgemein
- 2 Datenstruktur
- 3 Operationen
- 4 Logische Operatoren
- 5 Ergebnis

Roaring bitmaps

- Datenstruktur für Mengen
- $M = \{1, 2, 1000, 65\,000\}$
- Speichern von unsigned 32-Bit Integeren
- Komprimierung
- Alternative zu WAH

Roaring bitmaps

- Datenstruktur für Mengen
- $M = \{1, 2, 1000, 65\,000\}$
- Speichern von unsigned 32-Bit Integeren
- Komprimierung
- Alternative zu WAH

Ziele

- Möglichst kleiner Speicherverbrauch
- Operationen sollen schnell verarbeitet werden können
⇒ insert, remove, and, or

Idee

```
1010 1010 1111 1111 1111 0101 1010 0010
1010 1010 1111 1111 1011 1000 0101 0111
1010 1010 1111 1111 1010 0011 0010 1011
1010 1010 1111 1111 0111 1001 1011 0111
1010 1010 1111 1111 1001 0000 0101 1101
1010 1010 1111 1111 1001 1000 1111 0011
1010 1010 1111 1111 0001 1000 1101 0011
```

Idee

1010 1010 1111 1111 1111 0101 1010 0010 = 2 868 901 282

1010 1010 1111 1111 1011 1000 0101 0111 = 2 868 885 591

1010 1010 1111 1111 1010 0011 0010 1011 = 2 868 880 171

1010 1010 1111 1111 0111 1001 1011 0111 = 2 868 869 559

1010 1010 1111 1111 1001 0000 0101 1101 = 2 868 875 357

1010 1010 1111 1111 1001 1000 1111 0011 = 2 868 877 555

1010 1010 1111 1111 0001 1000 1101 0011 = 2 868 844 755

Idee

1010 1010 1111 1111 $\hat{=}$ 2 868 838 400

1010 1010 1111 1111 1111 0101 1010 0010

1010 1010 1111 1111 1011 1000 0101 0111

1010 1010 1111 1111 1010 0011 0010 1011

1010 1010 1111 1111 0111 1001 1011 0111

1010 1010 1111 1111 1001 0000 0101 1101

1010 1010 1111 1111 1001 1000 1111 0011

1010 1010 1111 1111 0001 1000 1101 0011

Idee

$$1010\ 1010\ 1111\ 1111 \hat{=} 2\ 868\ 838\ 400$$

$$1010\ 1010\ 1111\ 1111\ 1111\ 0101\ 1010\ 0010 = 2\ 868\ 838\ 400 + 62\ 882$$

$$1010\ 1010\ 1111\ 1111\ 1011\ 1000\ 0101\ 0111 = 2\ 868\ 838\ 400 + 47\ 191$$

$$1010\ 1010\ 1111\ 1111\ 1010\ 0011\ 0010\ 1011 = 2\ 868\ 838\ 400 + 41\ 771$$

$$1010\ 1010\ 1111\ 1111\ 0111\ 1001\ 1011\ 0111 = 2\ 868\ 838\ 400 + 31\ 159$$

$$1010\ 1010\ 1111\ 1111\ 1001\ 0000\ 0101\ 1101 = 2\ 868\ 838\ 400 + 36\ 957$$

$$1010\ 1010\ 1111\ 1111\ 1001\ 1000\ 1111\ 0011 = 2\ 868\ 838\ 400 + 39\ 155$$

$$1010\ 1010\ 1111\ 1111\ 0001\ 1000\ 1101\ 0011 = 2\ 868\ 838\ 400 + 6\ 355$$

Struktur

1 Allgemein

2 Datenstruktur

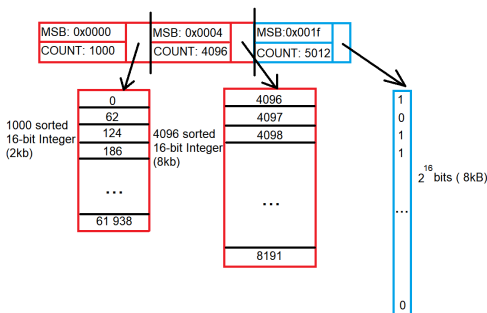
3 Operationen

4 Logische Operatoren

5 Ergebnis

Roaring bitmap

Datenstruktur



- Array von sortierten Chunks
- Jeder Chunk hält entweder einen sortierten Array oder eine Bitmap
- Integer mit den gleichen MSB werden in einem Chunk gespeichert

Roaring bitmap

Datenstruktur

Roaring bitmap

- Chunk chunks[n]

Chunk

- Short msb (16 bit)
- Short count (16 bit)
- Container container

Container

- **ENTWEDER:** Short array[4096]
- **ODER:** Long bitmap[1024]

Roaring bitmap

Datenstruktur

Roaring bitmap - (0 - $n \cdot 8\text{kB}$)

- Chunk chunks[n] (0 - $n \cdot 8\text{kB}$)

Chunk - (0 - 8kB)

- Short msb (16 bit)
- Short count (16 bit)
- Container container (0 - 8kB)

Container - (0 - 8kB)

- **ENTWEDER:** Short array[4096] ($0 \cdot (16 \text{ bit}) - 4096 \cdot (16 \text{ bit})$) (0 - 8kB)
- **ODER:** Long bitmap[1024] ($1024 \cdot 64 \text{ bit}$) (8kB)

Beispiel

$$\begin{aligned} S &= \{0, 62, 124, \dots, 61938\} && \cup \\ &\{65\,536, 65\,537, \dots, 65\,635\} && \cup \\ &\{x \mid x \in [2x2^{16}, 3x2^{16}) \wedge x \bmod 2 = 0\} \end{aligned}$$

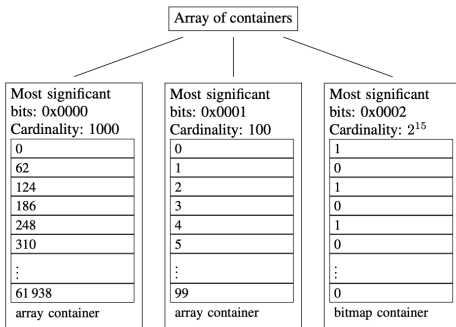
Beispiel

Lösung

$$S = \{0, 62, 124, \dots, 61938\} \quad \cup$$

$$\{65\,536, 65\,537, \dots, 65\,635\} \quad \cup$$

$$\{x \mid x \in [2 \times 2^{16}, 3 \times 2^{16}) \wedge x \bmod 2 = 0\}$$



Beispiel

Speicher

$$S = \{0, 62, 124, \dots, 61938\}$$

U

...

$$size_{bitmap} = (32 + 1\,000 * 16) + (32 + 100 * 16) + (32 + 65\,536)$$

$$size_{bitmap} = 83\,232 \text{ bits} \Rightarrow 10.4 \text{ kB}$$

$$size_{array} = (1\,000 + 100 + 32\,768) * 32$$

$$size_{array} = 1\,083\,776 \text{ bits} \Rightarrow 135.5 \text{ kB}$$

⇒ Roaring bitmap braucht nur 7,59%

⇒ Roaring bitmap hat noch Platz für weitere Integer

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

4 Logische Operatoren

5 Ergebnis

Kardinalität

- Summieren von den counts
⇒ $\mathcal{O}(n)$, n = Anzahl von Chunks

Existenz einer Zahl überprüfen

- Binary Search auf Chunks

⇒ $\mathcal{O}(\log n)$, n = Anzahl von Chunks

- Binary Search auf Container

⇒ $\mathcal{O}(\log m)$ oder $\mathcal{O}(1)$, m = Anzahl von Zahlen im Container

Insert & Remove

- Binary Search auf Chunks

⇒ $\mathcal{O}(\log n)$ n = Anzahl von Chunks

- Binary Search auf Container

⇒ $\mathcal{O}(\log m)$ oder $\mathcal{O}(1)$ m = Anzahl von Zahlen im Container

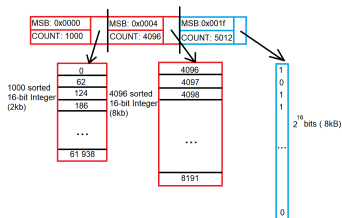
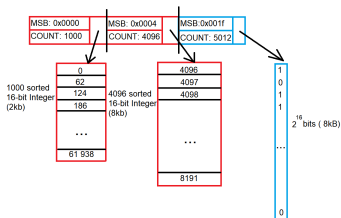
⇒ Wenn der Container sich ändern sollte, wird der alte gelöscht

Struktur

- 1 Allgemein
- 2 Datenstruktur
- 3 Operationen
- 4 Logische Operatoren**
- 5 Ergebnis

Vereinigung & Schnitt

OR & AND



OR

Bitmap vs Bitmap

- MSB müssen übereinstimmen
- Die Bitmaps werden Wörter-weise verundet
- Der neue Container ist auch eine Bitmap

OR

Bitmap vs Array

- MSB müssen übereinstimmen
- Der Array wird iteriert
- Die entsprechenden Bits werden in der Bitmap gesetzt
- Der neue Container ist auch eine Bitmap

OR

Array vs Array

- MSB müssen übereinstimmen
- Wenn die Summe der Kardinalitäten > 4096
 - ⇒ Bitmap wird erstellt und altes Verfahren angewendet
- Ansonsten
 - ⇒ Merge-Algorithmus merget die beiden Arrays
 - ⇒ Bei einer resultierenden Kardinalität von > 4096 , folgt Konvertierung in Bitmap

AND

Bitmap vs Bitmap

- MSB müssen übereinstimmen
- Die Bitmaps werden mit Wörter-weise verundet
- Der neue Container ist auch eine Bitmap

⇒ Bei einer Kardinlität von ≤ 4096 , folgt Konvertierung in Array

AND

Bitmap vs Array

- MSB müssen übereinstimmen
- Der Array wird iteriert
- Die Existenz der Zahl wird in der Bitmap geprüft
⇒ Zahlen aus dem Array entfernt
- Das Resultat ist ein neuer Array mit kleiner Kardinalität

AND

Array vs Array

- MSB müssen übereinstimmen
- Arrays werden mit optimierten Algorithmen miteinander verglichen
- Das Resultat ist ein neuer Array mit kleinen Kardinalität

Struktur

- 1 Allgemein
- 2 Datenstruktur
- 3 Operationen
- 4 Logische Operatoren
- 5 Ergebnis**

Ergebnis

Vergleich mit WAH

Vorteile

- Muss nicht dekomprimiert werden
- Verbrauchen weniger Platz
- Operationen werden schneller berechnet

Vielen Dank für ihre Aufmerksamkeit!

Quellen

- S. Chambi, D. Lemire, O. Kaser, R:Godin | Better bitmap performance with Roaring bitmaps | 15.06.2022 | <https://arxiv.org/pdf/1402.6407.pdf>