technische universität dortmund

fakultät für informatik

Roaring bitmaps

Justin Schartner

7. Juli 2022

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

4 Logische Operatoren

5 Beispiel

6 Ergebnis

Roaring bitmaps

- Datenstruktur für Mengen
- M = {1, 2, 1000, 65 000}

- Speichern von unsigned 32-Bit Integern
- Komprimierung
- Alternative zu WAH

Roaring bitmaps

- Datenstruktur für Mengen
- M = {1, 2, 1000, 65 000}

- Speichern von unsigned 32-Bit Integern
- Komprimierung
- Alternative zu WAH

Ziele

- Möglichst kleiner Speicherverbrauch
- Operationen sollen schnell verarbeitet werden können

  $\Rightarrow$ 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

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

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

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

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

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

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

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

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

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

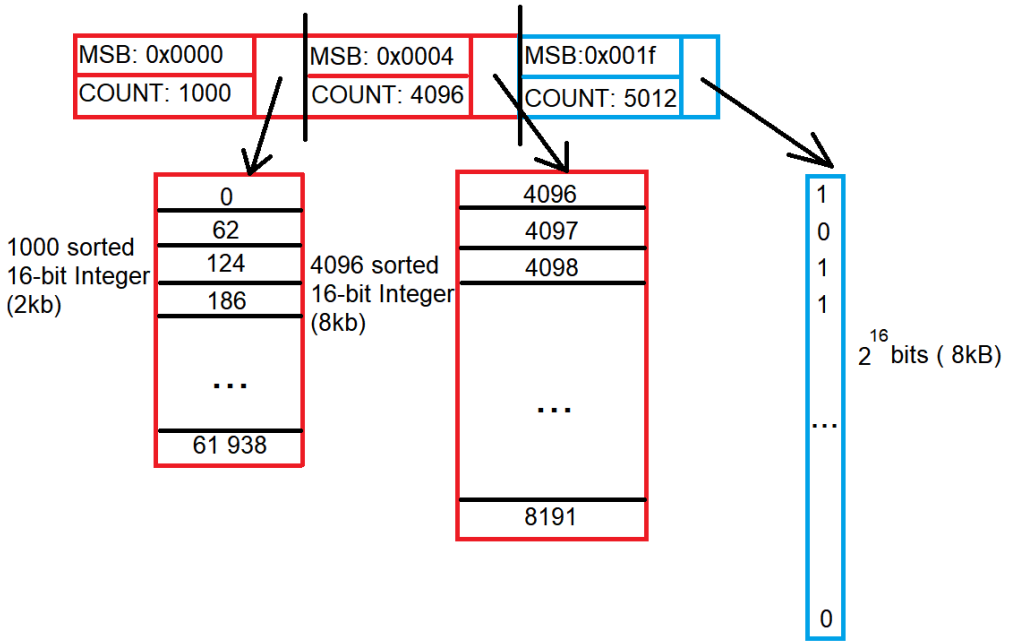
4 Logische Operatoren

5 Beispiel

6 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***8kB)

- Chunk chunks[**n**] (0 - **n***8kB)

Chunk - (0 - 8kB)

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

Container - (0 - 8kB)

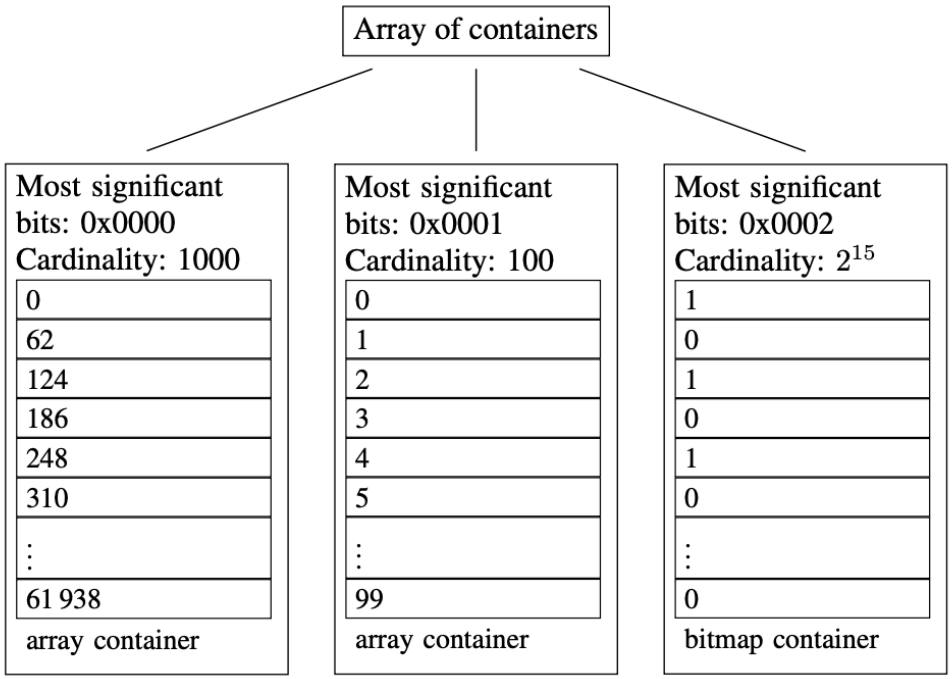
- **ENTWEDER:** Short array[4096] (0 * (16 bit) - 4096 * (16 bit)) (0 - 8kB)
- **ODER:** Long bitmap[1024] (1024 * 64 bit) (8kB)

Beispiel

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

Beispiel
Lösung

$$S = \{0, 62, 124, ..., 61938\} \cup$$
$$\{65\,536, 65\,537, ..., 65\,635\} \cup$$
$$\{x | x \in [2x2^{16}, 3x2^{16}) \wedge x \; mod \; 2 = 0\}$$



Beispiel
Speicher

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

$$...$$

$$size_{bitmap} = (32 + 1\,000 * 16) + (32 + 100 * 16) + (32 + 65\,536)$$
$$size_{bitmap} = 83\,232\ bits \Rightarrow 10.4kB$$

$$size_{array} = (1\,000 + 100 + 32\,768) * 32$$
$$size_{array} = 1\,083\,776\ bits \Rightarrow 135.5kB$$

$\Rightarrow$ Roaring bitmap braucht nur 7,59%

$\Rightarrow$ Roaring bitmap hat noch Platz für weitere Integer

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

4 Logische Operatoren

5 Beispiel

6 Ergebnis

Kardinalität

- Summieren von den counts

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

Existenz einer Zahl überprüfen

- Binary Search auf Chunks

  $\Rightarrow \mathcal{O}(logn)$, n = Anzahl von Chunks

- Binary Search auf Container

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

Insert & Remove

- Binary Search auf Chunks

  $\Rightarrow \mathcal{O}(logn)$ n = Anzahl von Chunks

- Binary Search auf Container

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

  $\Rightarrow$ Wenn der Container sich ändern sollte, wird der alte gelöscht

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

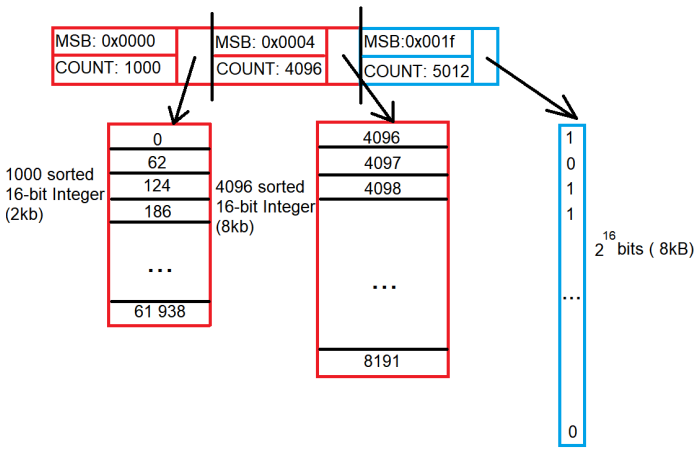
4 Logische Operatoren

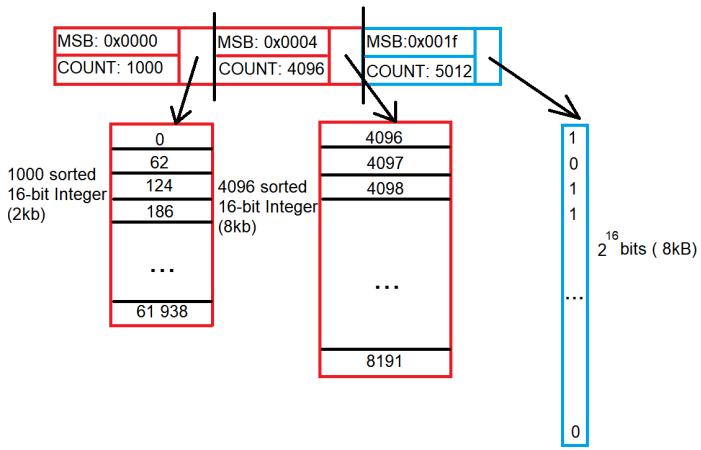
5 Beispiel

6 Ergebnis

Vereinigung & Schnitt

OR & AND





OR
Bitmap vs Bitmap

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

OR
Bitmap vs Array

- Der Array wird iteriert
- Die entsprechenden Bits werden in der Bitmap gesetzt

- Der neue Container ist auch eine Bitmap

OR
Array vs Array

- Wenn die Summe der Kardinalitäten > 4096

  $\Rightarrow$ Bitmap wird erstellt und altes Verfahren angewendet

- Ansonsten

  $\Rightarrow$ Merge-Algorithmus merget die beiden Arrays

  $\Rightarrow$ Bei einer resulitierenden Kardinalität von > 4096, folgt Konvertierung in Bitmap

AND
Bitmap vs Bitmap

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

$\Rightarrow$ Bei einer Kardinlität von <= 4096, folgt Konvertierung in Array

AND
Bitmap vs Array

- Der Array wird iteriert
- Die Existenz der Zahl wird in der Bitmap geprüft

  $\Rightarrow$ Zahlen aus dem Array entfernt

- Das Resultat ist ein neuer Array mit kleinen Kardinalität

AND
Array vs Array

- 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 Beispiel

6 Ergebnis

Beispiel

```
select sum(f.SalesAmount) as Betrag
from   FactResellerSales f inner join DimSalesTerritory dt
       on f.SalesTerritoryKey = dt.SalesTerritoryKey
       inner join DimDate dd on f.OrderDateKey=dd.DateKey
where  dt.SalesTerritoryCountry = 'United States'
  and  dd.CalendarYear between 2006 and 2008
```

$\Rightarrow$ United States $\wedge$ (2006 $\vee$ 2007 $\vee$ 2008)

BeispielDaten

FactResellerSales				
RID	OrderDateKey	SalesTerritoyKey	ProductKey	SalesAmount
f1	1	1	2da	12
f2	1	2	3fs	45
f3	1	2	1hf	32
f4	2	1	1hf	13
f5	2	1	1hf	26
f6	2	1	2da	57
f7	2	2	3fs	32
f8	3	1	3fs	11
f9	3	1	4q	5
f10	3	2	2da	16
f11	4	1	1hf	64
f12	4	2	1hf	31
f13	5	1	3fs	73
f14	5	2	3fs	31
f15	5	2	3fs	9

DimDate		
RID	OrderDateKey	CalendarYear
d1	1	2005
d2	2	2006
d3	3	2007
d4	4	2008
d5	5	2009

DimSalesTerritory		
RID	SalesTerritoryKey	SalesTerritoryCountry
c1	1	United States
c2	2	Germany

Beispieldaten-Indexing

FactResellerSales		
RID	OrderDateKey	SalesTerritoyKey
f1	1	1
f2	1	2
f3	1	2
f4	2	1
f5	2	1
f6	2	1
f7	2	2
f8	3	1
f9	3	1
f10	3	2
f11	4	1
f12	4	2
f13	5	1
f14	5	2
f15	5	2

IdxTerritory	
United States	Germany
1	0
0	1
0	1
1	0
1	0
1	0
0	1
1	0
1	0
0	1
1	0
0	1
1	0
0	1
0	1

IdxDate				
2005	2006	2007	2008	2009
1	0	0	0	0
1	0	0	0	0
1	0	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	1	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	1	0
0	0	0	0	1
0	0	0	0	1
0	0	0	0	1

Beispiel

UnitedStates :0000 0000 0000 0000 0001 0101 1011 1001

2006 :0000 0000 0000 0000 0000 0000 0111 1000

2007 :0000 0000 0000 0000 0000 0011 1000 0000

2008 :0000 0000 0000 0000 0000 1100 0000 0000

UnitedStates : 0000 0000 0000 0000

— 0001 0101 1011 1001

2006 : 0000 0000 0000 0000

— 0000 0000 0111 1000

2007 : 0000 0000 0000 0000

— 0000 0011 1000 0000

2008 : 0000 0000 0000 0000

— 0000 1100 0000 0000

Beispiel
Ergebnis

2006 :	0000 0000 0111 1000
2007 :	0000 0011 1000 0000
2008 :	0000 1100 0000 0000
	0000 1111 1111 1000
UnitedStates :	0001 0101 1011 1001
	0000 0101 1011 1000

$\Rightarrow$ Zeilen: 4, 5, 6, 8, 9 und 11

Struktur

1 Allgemein

2 Datenstruktur

3 Operationen

4 Logische Operatoren

5 Beispiel

6 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