PD Dr. rer. nat. habil. Sven Groppe

Übungen zur Vorlesung

# Semantic Web

WS 2012/2013

Übung 8 – Anfrageverarbeitung

## *Lösung*

### Aufgabe 1:

Beweis durch Gegenbeispiel. Betrachte dazu die folgenden Relationen R1 und R2 mit jeweils einem Tupel (die Relationen sind strukturell äquivalent, d. h., sie enthalten die gleichen Attribute):

R1

| m | k |
|---|---|
| 0 | 0 |

R2

| m | k |
|---|---|
| 0 | 1 |

$\Pi_m(R1 \cap R2) = \Pi_m($ | m | k | $) =$ | m |

$\Pi_m(R1) \cap \Pi_m(R2) =$ | m | 0 | $\cap$ | m | 0 | $=$ | m | 0 |

$\Pi_m(R1 - R2) = \Pi_m($ | m | k | $) =$ | m | 0 |

$\Pi_m(R1) - \Pi_m(R2) =$ | m | 0 | $-$ | m | 0 | $=$ | m |

### Aufgabe 2:

a) Die gesamte äußere Relation R wird einmal gelesen: $b_R$

In der inneren Schleife wird in einer Runde (d.h. für den jetzigen „Hauptspeicherkontext“ der Relation R) die gesamte Relation S einmal gelesen: $b_S$

Es werden so viele Runden benötigt, wie viel mal der „Hauptspeicherkontext“ der Relation R gewechselt werden muss: $\lceil b_R/(m-k) \rceil$

Also, die gesamte Anzahl der Platten(lese)zugriffe zu Seiten der Relation R wird $b_R$ und die gesamte Anzahl der Plattenzugriffe zu Seiten der Relation S wird $\lceil b_R/(m-k) \rceil b_S$

Insgesamt werden $\mathbf{b_R + \lceil b_R/(m-k) \rceil b_S}$ benötigt.

b) **Die Anzahl der Platten(lese)zugriffe wird minimiert, wenn die äußere Relation *R* die kleinere ist und wenn k=1 gewählt wird.**

In diesem Beispiel passen in eine Seite $\lfloor 8192 / 100 \rfloor = 81$ Tupel der Relation *A*, d.h.
$b_A = \lceil 800.000 / 81 \rceil$

Für die Relation *B* passen in eine Seite $\lfloor 8192 / 50 \rfloor = 163$ Tupel, d. h.
$b_B = \lceil 3.000.000 / 163 \rceil = 18.405$

Für *m* gilt
$m = \lfloor 2048 / 8 \rfloor = 256$

Die Anzahl der Platten(lese)zugriffe wird (Relation *A* wird die äußere)
$9877 + \lceil 9877 / 255 \rceil * 18.405 =$ **727.672**

## Aufgabe 3:

Im optimalen Fall, **wenn es keine doppelten Werte gibt**, sind maximal $\mathbf{b_R + b_S}$ Platten(lese)zugriffe notwendig, weil beide Relationen einmal gelesen werden (bei Equi-Join). Die Anzahl erhöht sich normalerweise nur gering, wenn es doppelte Werte gibt. Kommt es allerdings zu dem Extremfall, dass alle Werte gleich sind, ist der Aufwand gleich dem Aufwand des Nested-Loop-Joins. Der Grund des zusätzlichen Aufwands bei Duplikaten ist, dass der Zeiger (Cursor) einer Relation mehrmals zurückgesetzt werden muss, so dass er auf den ersten duplizierten Wert zeigt.

## Aufgabe 4:

a) Vor der ersten Runde ist die ganze Relation in einer Partition gespeichert, also $x_0 = b_R$. In jeder Runde verkleinert sich die Größe jeder Partition um einen Faktor $m - 1$. Also

$$x_i = b_R / (m-1)^i$$

(Dieser Wert müsste eigentlich immer wieder aufgerundet werden, wodurch sich die Anzahl der Seiten von Runde zu Runde etwas erhöht. Diesen kleinen Fehler werden wir aber ignorieren.)

b) Es muss solange partitioniert werden, bis $x_i \leq m - 1$. Daraus folgt

$$i \geq \log_{m-1}(b_R) - 1$$

Es sind also insgesamt $\lceil \log_{m-1}(b_R) - 1 \rceil$ Runden nötig.

c) Pro Runde muss jede Partition einmal gelesen und einmal geschrieben werden. Da alle Partitionen zusammen die gesamte Relation darstellen, wird pro Runde jede Seite der Relation einmal gelesen und einmal geschrieben. Das bedeutet $2\ b_R$ Plattenzugriffe pro Runde. Insgesamt sind für das Partitionieren der Relation *R*

$$2\ b_R \lceil \log_{m-1}(b_R) - 1 \rceil$$

Plattenzugriffe notwendig.

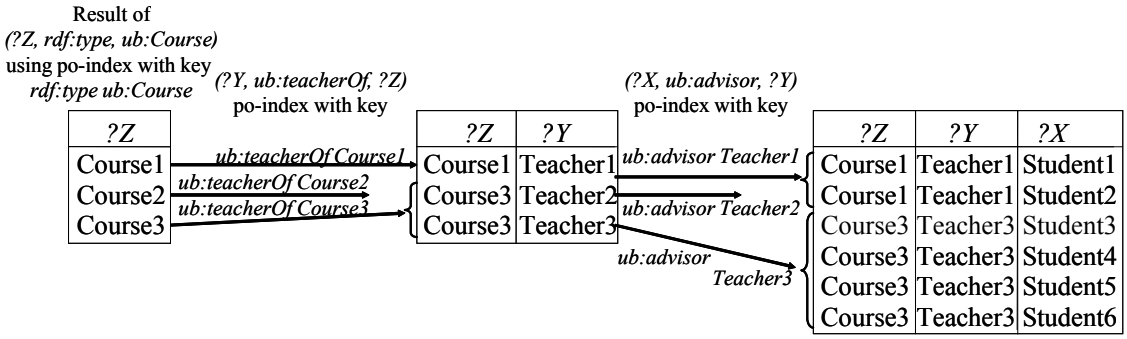
d) Insgesamt müssen beide Relationen partitioniert werden (die Anzahl von Runden ist für beide Partitionen gleich, siehe Aufgaben a) und b)) und anschließend muss der Join ausgeführt werden. Das bedeutet

$$2\ b_R \lceil \log_{m-1}(b_R) - 1 \rceil + 2\ b_S \lceil \log_{m-1}(b_R) - 1 \rceil + b_R + b_S = (b_R + b_S)\ (2 \lceil \log_{m-1}(b_R) \rceil - 1)$$

Plattenzugriffe.

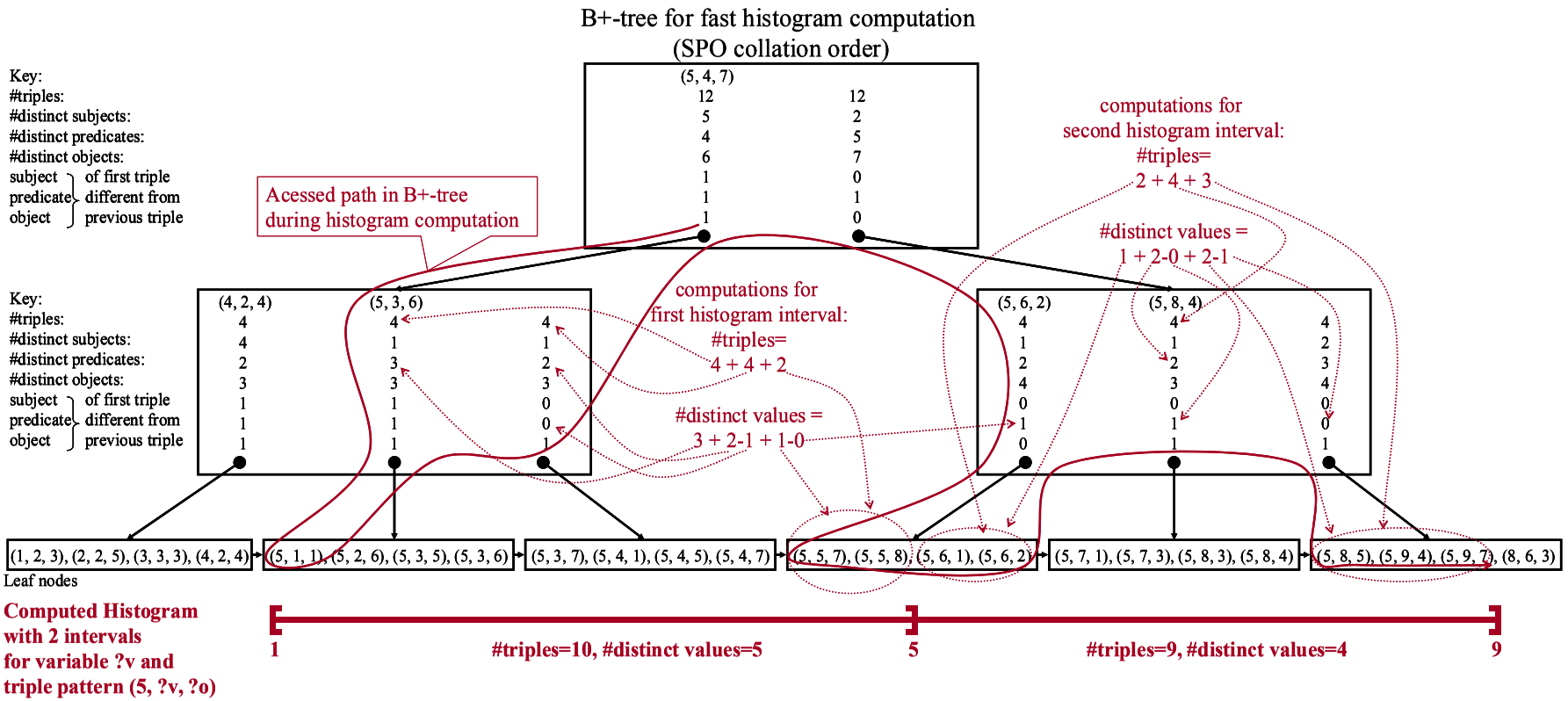
## Aufgabe 5:

Die Joinberechnungen können folgendermaßen visualisiert werden:




## Aufgabe 6:

Der Histogramm-Index mit berechnetem Histogramm:


B+-tree for fast histogram computation
(SPO collation order)
Key:
#triples:
#distinct subjects:
#distinct predicates:
#distinct objects:
subject } of first triple
predicate } different from
object } previous triple
(5, 4, 7)
12 12
5 2
4 5
6 7
1 0
1 1
1 0
Acessed path in B+-tree
during histogram computation
computations for
second histogram interval:
#triples=
2 + 4 + 3
#distinct values =
1 + 2-0 + 2-1
Key:
#triples:
#distinct subjects:
#distinct predicates:
#distinct objects:
subject } of first triple
predicate } different from
object } previous triple
(4, 2, 4)
4
4
2
3
1
1
1
(5, 3, 6)
4 4
1 1
3 2
3 3
1 0
1 0
1 1
computations for
first histogram interval:
#triples=
4 + 4 + 2
#distinct values =
3 + 2-1 + 1-0
(5, 6, 2)
4
1
2
4
0
1
0
(5, 8, 4)
4 4
1 2
2 3
3 4
0 0
1 0
1 1
(1, 2, 3), (2, 2, 5), (3, 3, 3), (4, 2, 4)
(5, 1, 1), (5, 2, 6), (5, 3, 5), (5, 3, 6)
(5, 3, 7), (5, 4, 1), (5, 4, 5), (5, 4, 7)
(5, 5, 7), (5, 5, 8), (5, 6, 1), (5, 6, 2)
(5, 7, 1), (5, 7, 3), (5, 8, 3), (5, 8, 4)
(5, 8, 5), (5, 9, 4), (5, 9, 7), (8, 6, 3)
Leaf nodes
Computed Histogram
with 2 intervals
for variable ?v and
triple pattern (5, ?v, ?o)
1
#triples=10, #distinct values=5
5
#triples=9, #distinct values=4
9