Datenbanksysteme WS 2016/17
Prof. Dr.-Ing. Sebastian Michel
BSc. Peter Brucker
TU Kaiserslautern, FB Informatik – Lehrgebiet Informationssysteme
http://dbis.informatik.uni-kl.de
Übungsblatt 7: Ausgabe 13.12.2016, Abgabe 19.12.2016

Aufgabe 1: Indexstrukturen im metrischen Raum (1 P.)

(a) Beweisen Sie Lemma 1 und Lemma 3 aus der Vorlesung.

(b) Bereichsanfragen im M-Baum

Eine Bereichsanfrage $Q = (q, \varepsilon)$ wählt alle Punkte aus, die nicht weiter als ε um Anfragepunkt q herum liegen. Der Algorithmus unten geht davon aus, dass der gerade besuchte Knoten N ein innerer Knoten ist, also kein Blatt und nicht die Wurzel. N besitzt also eine Zahl von Einträgen der Form (o, r_o, D, T). o bezeichnet das Pivot-Element, r_o den Radius um das Pivot-Element herum, d.h. den Raum, der durch diesen Knoteneintrag abgedeckt wird. D bezeichne den Abstand des Elternknotens p zu o. T ist ein Zeiger auf den entsprechenden Teilbaum.

```
1  for each (o, r_o, D, T) in N
2      if |d(p,q) − D| ≤ ε + r_o then
3          calculate d(o,q)
4          if d(o,q) ≤ ε + r_o then
5              call search procedure for subtree T
```

Wie wir sehen, beschreibt Zeile 2 eine Überprüfung, bevor in Zeile 3 die Distanz von o zu q berechnet und schließlich in Zeile 4 evaluiert wird.

(i) Erläutern/beweisen Sie die Korrektheit der Bedingung in Zeile 2. Veranschaulichen Sie dazu das "Problem" graphisch, für beide Fälle, d.h. $|d(p,q) - D| \leq \varepsilon + r_o$ sowie $|d(p,q) - D| > \varepsilon + r_o$

(ii) Ist die Bedingung in Zeile 2 überhaupt notwendig? Was passiert, wenn Zeile 2 wegfallen würde, ist der Algorithmus dann noch korrekt oder können Sie ein Gegenbeispiel angeben?

Aufgabe 2: Hashing (1 P.)

(a) Erwartungswerte für Statisches Hashing

Gegeben eine Hashtabelle, die als Überlaufbehandlung Lineares Sondieren benutzt, und eine Hashfunktion $\tilde{h}$, die in 4 verschiedene Buckets der Größe 15 abbildet. Für uniform verteilte Eingaben von $\tilde{h}$ gelten folgende Wahrscheinlichkeiten:

$$\Pr[\tilde{h}(x) = 0] = 0.4$$
$$\Pr[\tilde{h}(x) = 1] = 0.4$$
$$\Pr[\tilde{h}(x) = 2] = 0.1$$
$$\Pr[\tilde{h}(x) = 3] = 0.1$$

(i) Nachdem 50 zufällige Datensätze eingefügt wurden, wie ist der erwartete Füllstand der Hashtabelle?

(ii) Wenn ein weiterer zufälliger Datensatz eingefügt wird, wie viele Seitenzugriffe müssen im Erwartungswert durchgeführt werden?

(iii) Wie ändert sich die Zahl der erwarteten Seitenzugriffe beim Einfügen, wenn statt Linearem Sondieren zufällig, unabhängig und gleichverteilt sondiert wird?

(iv) Was ist der Unterschied zu einer Hashfunktion, die gleichverteilt in vier Buckets abbildet?

(b) Lineares Hashing

Gegeben sei eine Datei, die 3 Buckets mit jeweils einer Kapazität von 3 Datensätzen enhält. Es wird folgende Sequenz von Hash-Funktionen verwendet:

$$H_i(K) = K \mod (3 \cdot 2^i) \text{ mit } i \in \{0, 1, 2, \ldots, n\}$$

Verteilen Sie unter Verwendung von linearem Hashing die Datensätze mit den folgenden Schlüsseln in die zu Beginn leeren Buckets:

$$7, 3, 2, 6, 10, 12, 11, 8, 14, 9, 23, 19, 24, 20, 25, 30$$

Führen Sie bei Überschreitung der Belegung $\beta > 0.7$ kontrolliertes Splitting aus.

Aufgabe 3: Raumfüllende Kurven für NN-Anfragen (1 P.)

Gegeben eine Tabelle namens `mydata`, die Datenpunkte enthält, und eine Tabelle namens `anfragen`, die Anfragen enthält. Die Tabellen sind wie folgt definiert:

```
CREATE TABLE mydata (id serial, d1 integer, d2 integer, d3 integer, zcurve_value integer);
CREATE TABLE anfragen (id serial, d1 integer, d2 integer, d3 integer, zcurve_value integer);
```

Auf der Vorlesungswebseite finden Sie einen Link zu dem Postgresql-Dump, der Beispieldaten zu beiden Tabellen enthält.

(a) In den vorgegebenen Dumps fehlen die `zcurve_value` Einträge. Implementieren Sie eine User-Defined-Function in PostgresSQL:

```
CREATE or REPLACE FUNCTION zcurve(d1 int, d2 int, d3 int)
RETURNS int
AS $$
...
$$ LANGUAGE plpgsql;
```

welche für einen die Koordinaten eines Punktes in drei Dimensionen den entsprechenden Wert der Z-Kurve berechnet. Gehen Sie davon aus, dass jede Koordinate aus einem vorzeichenlosen Integer-Wert $< 2^{10}$ besteht Füllen Sie mittels dieser Funktion die o.g. Tabellen.

Mit folgender materialisierten Sicht werden nun für jede Anfrage die Distanzen bzgl. echten Werten und Werten der Z-Kurve berechnet.

```
CREATE MATERIALIZED VIEW all_distances AS (
SELECT a.id as aid, m.id as did, sqrt((m.d1-a.d1)^2+
       (m.d2-a.d2)^2+(m.d3-a.d3)^2) as rdistance,
 sqrt((m.zcurve_value-a.zcurve_value)^2) as zdistance
FROM mydata m, anfragen a)
```

(b) Beantworten Sie anhand der oben erstellten Daten folgende Fragen:

- Betrachten Sie zu jeder Anfrage die 3 nächsten Punkte (3-NN) anhand der echten Werte und der Z-Kurven-Werte. Bei welchen Anfragen gibt es Übereinstimmungen bei den 3-NN? Geben Sie neben der Lösung auch die SQL-Anfrage(n) an, mit der die Lösung berechnet wurde.
- Berechnen Sie für jede Anfrage den Mean-Squared-Error (MSE) über die 3-NN bzgl. der Z-Kurven Werte im Vergleich zur echten Distanz. Geben Sie ferner den maximalen und minimalen MSE über alle Anfragen an. Geben Sie auch hier neben der Lösung die SQL-Anfrage(n) an, mit der die Lösung berechnet wurde.

Hinweise: Folgender Java-Code berechnet den Z-Wert für einen zweidimensionalen Punkt mit Koordinaten (d1,d2), für Koordinaten je in $[0, 2^5]$.

```
static int zcurve(int d1, int d2) {
  int ret = 0;
  for (int loop=0; loop<6; loop++) {
    if ((d1 & (1<<loop)) == (1<<loop))
      ret = ret | (1<<2*loop);
    if ((d2 & (1<<loop)) == (1<<loop))
      ret = ret | (1<<(2*loop+1));
  }
  return ret;
}
```

Um in Postgres Zahlen als Bitstring der Länge n angezeigt zu bekommen, können Sie die Zahl zu `bit(n)` casten. Zum Beispiel:

```
select 5::bit(8); -- liefert: 00000101
```