TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einsatz und Realisierung von Datenbanken* im SoSe20

Maximilian {Bandle, Schüle}, Josef Schmeißer (i3erdb@in.tum.de)
http://db.in.tum.de/teaching/ss20/impldb/

Blatt Nr. 10

Hausaufgabe 1

Schätzen Sie die Anzahl der Cache-Misses, die entstehen, wenn man 1001 32-Bit-Integer-Werte (0-1000) in aufeinanderfolgender Reihenfolge in einen ART Baum einfügt. Wäre ein B+ Baum besser oder schlechter? Bei den Baumknoten müssen die Header nicht berücksichtigt werden, Pointer haben eine Größe von 64 Bit.

Größe der einzelnen ART Knoten (mit 64-Bit Pointern und ohne Header):

Node4 $4 + 4 * 8 = 36$ Byte

Node48 $256 + 48 * 8 = 640$ Byte

Node256 $256 * 8 = 2048$ Byte

Die Höhe eines ART-Baums ist durch die Schlüssellänge beschränkt (in unserem Fall maximal Höhe 4), da in jedem Knoten ein Byte des Schlüssels gespeichert wird. Da die Integerzahlen aufeinanderfolgend sind, unterscheiden sie sich maximal in den letzten zwei Bytes (die Werte zwischen 0 und 1000 haben immer 0x00 0x00 als Präfix). Für die ersten zwei Bytes reicht es, einen Node4 zu nehmen, da hier alle Einträge den selben Wert besitzen. Auf dem letzten Level reichen 4 Node256, um die letzten Bytes der 1000 Integer Werte einzufügen. Da es nur vier Kindnoten gibt, reicht auf Level drei auch ein Node4. Die Gesamtgröße des Baums ist somit $4 * 2048 + 3 * 36 = 8300$ Byte. Dies passt locker in den L1 Cache heutiger CPUs, der typischerweiße 64 KB groß ist. Somit gibt es keine Cache Misses. Während der Baum gebaut wird, sind auf der untersten Eben ursprünglich auch Node 4, die aber über Node 16, zu Node 48 und zu Node 256 wachsen.
Ein B+ Baum ist schlechter, da bei sequentiellem Einfügen die Knoten nur halb gefüllt sind. Außerdem werden in den Knoten jeweils pro Pointer auch noch ein kompletter 32-Bit Rangeschlüssel gespeichert, was den Speicherbedarf zusätzlich erhöht.

Hausaufgabe 2

In Abbildung 1 sehen Sie die Knoten eines ART Baums. Der Wurzelknoten liegt an Adresse `A`. Zeiger die mit `d` anfangen (z.B. da, db, ...) zeigen auf Daten. Suchschlüssel sind in den Aufgaben jeweils sowohl als Zahl z.B. 99, als auch hexadezimal codiert angegeben, z.B. der Wert 99 als 32 Bit Integer (0x00 0x00 0x00 0x63).

1) Beschreiben Sie kurz den Pfad durch den Baum für den 32-bit Suchschlüssel 2856344642 (0xAA 0x40 0x5C 0x42).

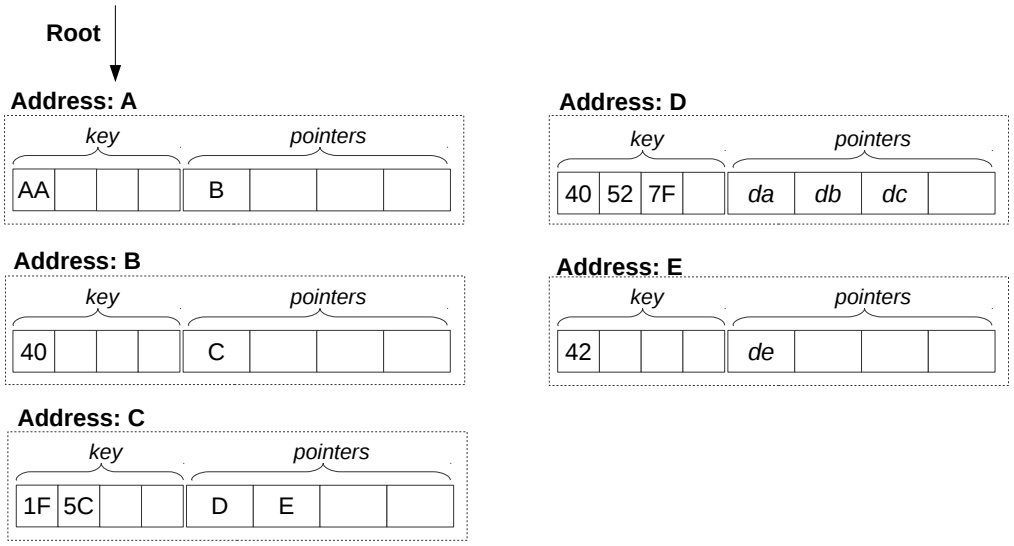

Abbildung 1: Knoten des ART (jeweils Node4)

- Suche 1. Byte in Wurzel A. Gefunden, gehe zu Knoten bei B
- Suche 2. Byte in Knoten B. Gefunden, gehe zu Knoten bei C
- Suche 3. Byte in Knoten C. Gefunden, gehe zu Knoten bei E
- Suche 4. Byte in Knoten E. Pointer zu Daten ve

Ergebnis: Schlüssel ist in Daten enthalten

2) Welche dieser Suchschlüssel sind im Baum enthalten? 291 (0x00 0x00 0x01 0x23), 2856329024 (0xAA 0x40 0x1F 0x40), 2856329026 (0xAA 0x40 0x1F 0x42)
   - 291: Nicht enthalten
   - 2856329024: Enthalten
   - 2856329026: Nicht enthalten
3) Beschreiben Sie kurz wie sich der Baum beim Einfügen des Schlüssels 2856352578 (0xAA 0x40 0x7B 0x42) verändert. Der Schlüssel soll auf den Wert an der Adresse `df` zeigen.
   - Die ersten beiden Bytes sind schon im Baum enthalten. Dafür keine Änderung notwendig.
   - Im Knoten C wird im Schlüsselfeld an der dritten Stelle der Wert 0x7B eingetragen und an der dritten Pointer Stelle dann X.
   - Das letzte Schlüsselbyte muss ein neuer Node4 Knoten an der Stelle F erstellt werden. Dieser Knoten enthält das Suchbyte 0x42 und den Pointer df jeweils an der ersten Stelle.

Hinweise Die Aufgaben können auf `http://xquery.db.in.tum.de/` getestet werden. Die Daten für das Unischema können mit doc('uni2') geladen werden. Zur Lösung der Aufgaben

können Sie die folgenden XQuery-Funktionen verwenden:

max(NUM), count(X), tokenize(STR,SEP), sum(NUM), contains(HAY,NEEDLE)

1. max(NUMBERS) - Returns largest number from list
2. count(LIST) - Return the number of elements in the list
3. tokenize(STR,SEP) - Splits up the string at the seperator
4. sum(NUMBERS) - Returns sum of all numbers in list
5. contains(HAY,NEEDLE) - Checks if the search string (NEEDLE) is contained in the string (HAY)
6. distinct-values(LIST) - Returns the distinct values from the list

Hausaufgabe 3

Lösen Sie in **reinem XPath** folgende Aufgaben und testen Sie diese auf `xquery.db.in.tum.de`.

1. Lassen Sie sich das gesamte Schema anzeigen.

```
doc('uni')
```

2. Finden Sie die Namen aller Fakultäten.

```
doc('uni')//FakName
```

3. Finden Sie die Namen aller Studenten, die Vorlesungen hören.

```
doc('uni')//Student[./hoert]/Name
```

Hausaufgabe 4

Formulieren Sie die zuvor in SQL bearbeiteten Anfragen zur Universitätsdatenbank in XQuery. Erstellen Sie insbesondere XQuery-Anfragen, um folgende Fragestellungen zu beantworten [1]:

a) Suchen Sie die Professoren, die Vorlesungen halten.

```
doc('uni2')//ProfessorIn[.//Vorlesung]/Name
```

b) Finden Sie die Studenten, die alle Vorlesungen gehört haben.

```
doc('uni2')//Student[count(tokenize(hoert/@Vorlesungen," "))=
count(//Vorlesung)]/Name
```

c) Finden Sie die Studenten mit der größten Semesterzahl unter Verwendung von Aggregatfunktionen.

```
let $maxsws:=max(data(doc('uni2')//Student/Semester))
return doc('uni2')//Student[Semester=$maxsws]
(: alternativ als Einzeiler :)
return doc('uni2')//Student[Semester=max(//Student/Semester))]
```

d) Berechnen Sie die Gesamtzahl der Semesterwochenstunden, die die einzelnen Professoren erbringen. Dabei sollen auch die Professoren berücksichtigt werden, die keine Vorlesungen halten.

```
for $p in doc('uni2')//ProfessorIn
return <Prof>{$p/Name}<Summe>{sum(data($p//SWS))}</Summe></Prof>
```

[1] Sie können die Aufgabe unter `http://xquery.db.in.tum.de` mit dem doc('uni2') Datensatz testen.

e) Finden Sie die Studenten, die alle vierstündigen Vorlesungen gehört haben.

```
let $fourcount:=count(doc('uni2')//Vorlesung[SWS=4])
for $s in doc('uni2')//Student
where count(
  for $h in tokenize($s/hoert/@Vorlesungen," ")
  where doc('uni2')//Vorlesung[@VorlNr=$h and SWS=4]
  return $h
) = $fourcount
return $s/Name
```

f) Finden Sie die Namen der Studenten, die in keiner Prüfung eine bessere Note als 3.0 hatten.

```
for $s in doc('uni')//Student
where count(
  for $p in $s//Pruefung
  where $p/@Note < 3
  return $p
) = 0
return $s
```

g) Berechnen Sie den Umfang des Prüfungsstoffes jedes Studenten. Es sollen der Name des Studenten und die Summe der Semesterwochenstunden der Prüfungsvorlesungen ausgegeben werden.

```
for $s in doc('uni')//Student
return <Student>{$s/Name}<sum>{
  sum(for $p in $s//Pruefung
      return doc('uni')//Vorlesung[@VorlNr=$p/@Vorlesung]/SWS)}
</sum></Student>
```

h) Finden Sie Studenten, deren Namen den eines Professors enthalten.

```
for $s in doc('uni')//Student
where doc('uni')//ProfessorIn[contains($s/Name,Name)]
return $s/Name
```

i) Ermitteln Sie den Bekanntheitsgrad der Professoren unter den Studenten, wobei wir annehmen, dass Studenten die Professoren nur durch Vorlesungen oder Prüfungen kennen lernen.

```
for $p in doc('uni')//ProfessorIn
return
  <Professor>
    {$p/Name}
    <Bekanntheit>
    {
      count(
        doc('uni')//Student[.//Pruefung[@Pruefer=$p/@PersNr]]/Name
        union
        ( for $v in $p//Vorlesung/@VorlNr
          return doc('uni')//Student[contains(hoert/@Vorlesungen,$v)]/Name)
      )
    }
    </Bekanntheit>
  </Professor>
```

Gruppenaufgabe 5

Überlegen Sie sich, wie Ihre Visitenkarte im JSON-Format aussähe und stellen Sie diese in der Übung vor.

```
{
  "vorname": "Maximilian",
  "nachname": "Schuele",
  "e-mail": "i3erdb@in.tum.de",
  "adresse": {
    "raum": "2.11.060",
    "strasse": "Boltzmannstr.␣3",
    "stadt": "Garching",
    "bundesland": "Bayern",
    "postleitzahl": 85748
  },
  "Telenummern": [
    {
      "typ": "fon",
      "nummer": "089␣289␣17250"
    },
    {
      "typ": "fax",
      "nummer": "089␣289␣17263"
    }
  ]
}
```