CS 377: Database Systems

Homework #4

Due: Monday, April 11, 2016 IN CLASS

1. **Extensible Hashing** ($5 + 5 \times 3$ points)

   Assume we have the following records where we indicate the hashed key (binary representation) in brackets. We want to use an Extendible Hashing structure where each bucket can hold up to 3 records and the structure initially starts out empty (one empty bucket).

   - 'a' [00001]
   - 'b' [01000]
   - 'c' [00000]
   - 'd' [10100]
   - 'e' [10101]
   - 'f' [00010]
   - 'g' [11000]
   - 'h' [10001]

   Consider the result after the records above have been inserted in the order shown, using the highest-bits for the hash function. In other words, records in a bucket of local depth d, agree on their *leftmost* d bits.

   (a) What does the hash structure look like after inserting all the records?

   (b) What is the global depth of the resulting directory?

   (c) What record causes the first split?

   (d) What record causes the second split?

   (e) After inserting these records, suppose that we insert 'i' [10000]. How many buckets will we have now?

   (f) What other keys are in the same bucket with record 'i'?

2. **B^+-Tree** (20 points)

   Consider the B^+-Tree of order d = 2, height h = 3 levels, and number of values per node n =4 shown in Figure 1. For each subpart, draw the tree after the specified operation starting from the one (Figure 1) provided below (i.e., the subparts are independent and do not affect one another). Please follow the guidelines listed below:

   - The left pointer denotes $<$ while the right pointer denotes $\geq$.
   - In the case of underflow, if you can borrow from both siblings, choose the one on the *left*.
   - In the case of overflow, promote the key in the middle from the left sibling.

   (a) Insert tuple with search key value 10.

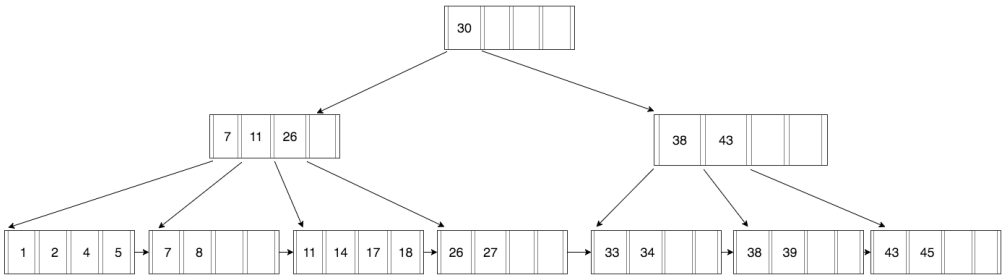

Figure 1: B^+-Tree for problem 2

(b) Insert tuple with search key value 20.

(c) Delete tuple with search key value 7.

(d) Delete tuples with search key values 17, 18, 26.

3. **Using Indexes on Yelp Reviews** (10 points) Consider the following Yelp review database:

```
Business(bid, name, city, state)
BusinessCategory(bid, category)
User(uid, name, fans)
Review(bid, uid, stars, date, votes_useful, votes_funny, votes_cool)
```

In addition to the primary keys that have been specified, we've also created an index on the attribute city on the relation business using the following command:

```
CREATE INDEX bcity_idx ON business(city);
```

For the following queries, answer if the index we created was used or not.

(a)
```
SELECT * FROM business
WHERE city = 'Atlanta';
```

(b)
```
SELECT * FROM business
WHERE city > 'Atlanta';
```

(c)
```
SELECT * FROM business
WHERE city > 'B' AND city < 'G' AND name = 'A';
```

(d)
```
SELECT * FROM business
WHERE city > 'B' AND city < 'G' OR state = 'GA';
```

(e)
```
SELECT * FROM business
WHERE city != 'Atlanta';
```

4. **Selection Search via B^+-Tree** (15 points)

Consider the following bank database (it was used as an example during the midterm review):

```
Branch(bname, bcity, assets)
Customer(cname, cstreet, ccity)
Loan(lID, bname, amount)
Borrower(cname, lID)
Account(aID, bname, balance)
Depositor(cname, aID)
```

Suppose a B^+-Tree index was created on bcity for relation Branch, and there are no other indexes in the database. Describe the optimal selection algorithm, specifying whether the index on bcity was used or not:

(a) $\sigma_{\text{NOT(bcity<'Brooklyn')}}(\text{Branch})$

(b) $\sigma_{\text{NOT(bcity='Brooklyn')}}(\text{Branch})$

(c) $\sigma_{\text{NOT(bcity<'Brooklyn' OR assets<5000)}}(\text{Branch})$

5. **Estimating Cost of Joins** (5 + 5 + 10 points)

Let relations $R_1(A, B, C)$ and $R_2(C, D, E)$ have the following properties: R_1 has 20,000 tuples, R_2 has 45,000 tuples. 25 tuples of R_1 fit on one block and 30 tuples of R_2 fit on one block. Assume that the join can not be all done in main memory, so it requires disk access. Estimate the number of block transfers and seeks required using each of the following join strategies for $R_1 \bowtie R_2$, assuming that there are M blocks of memory:

(a) Nested-loop join

(b) Merge join (assume neither are sorted on the join key but the tuples with the same bucket fit in memory)

(c) Hash join (assume no overflow occurs and calculate for both recursive partitioning and without recursive partitioning)

6. **Efficient Join Strategy** (15 points)

Consider the relations $R_1(A, B, C)$, $R_2(C, D, E)$, and $R_3(E, F)$ with primary keys A, C, and E, respectively. Assume that R_1 has 1000 tuples, R_2 has 1500 tuples, and R_3 has 750 tuples. Estimate the size of $R_1 * R_2 * R_3$ and give an efficient strategy (e.g., are there other indexes we should create in the relations, and what are the steps) for computing the join.