TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Dr. Thomas Neumann

Exercises for *Foundations in Data Engineering*, WiSe 23/24

Alexander Beischl, Maximilian Reif (i3fde@in.tum.de)
http://db.in.tum.de/teaching/ws2324/foundationsde

Sheet Nr. 10

Exercise 1 Neojoin

Relation X and Y are distributed over three compute nodes as shown in Table 1 and Table 2. They shall be joined on $X.a = Y.b$. The compute nodes are connected by a star-shaped network. That means, there is one central switch which every node is connected to. All nodes have the same bandwidth towards the switch.

In preparation of executing a Neojoin, create one *partition to node* assignment that

1. minimizes network traffic
2. one that balances the amount of tuples on all nodes

Use $h(x) = x \; mod \; 3$ to partition the elements using relation X's attribute a and relation Y's b. Table 1 and Table 2 already contain the results of $h(x)$ for each tuple.

Then, create a minimal schedule for the balanced variant.

Table 1: Relation X, distributed over three nodes.

Node	a	$h(a) = a \; mod \; 3$
A	3	0
A	4	1
A	7	1
A	8	2
A	6	0
A	9	0
B	2	2
B	5	2
B	3	0
B	5	2
B	4	1
C	3	0
C	2	2
C	7	1
C	9	0

Table 2: Relation Y

Node	b	$h(b) = b \; mod \; 3$	Label
A	3	0	Star
B	4	1	Square
C	2	2	Clique

Exercise 2 Demonstrate the insertion of the keys 5, 28, 19, 15, 20, 33, 12, 17, 10 into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \; mod \; 9$.

Exercise 3 We have a hashtable with m buckets and insert $m * n$ elements. For this task we assume m is a constant and the universe of keys is U.

Show that if $|U| > n * m$, there is a subset of U of size $\geq n$ consisting of keys that all hash to the same bucket, so that the worst-case lookup time for hashing with chaining is $\Omega(n)$.

Exercise 4 **Hashing and hash tables**

This exercise is about hash tables and properties of hash functions used for them. In order to get a feeling for implementation differences, we ask you to create your own implementations for this task.

1. Measure the performance of modulo with prime. Create some random integers and hash them. How many cycles does it take to compute one hash?
2. What is the performance when using modulo of power of two?
3. Using non-prime modulo as a hash function has a significant drawback. When all hashed values and the modulus have a factor in common, the domain of the hash is underutilized. In order to measure the impact of this, create a simple hashtable. Use hashing with chaining and implement the operations `insert` and `find`. Create some random integer data and insert it into the hashtable. Then, perform key lookups for all elements. How fast is the lookup when a prime is used? How fast is the lookup with powers of two? How fast is the lookup without maliciously constructed data?
4. Extra (not discussed in the tutorial): How fast is *Fibonacci Hashing*? Does it withstand the attack?

You can use the following `C++` function to measure cycles. Call the function before and after the code snippet you want to measure and calculate the difference, which is the number of cycles spent in the code snippet.

```cpp
/// Read cycle counter for x86 (Intel & AMD)
uint64_t rdtsc() {
  unsigned int lo, hi;
  __asm__ __volatile__("rdtsc" : "=a"(lo), "=d"(hi));
  return ((uint64_t)hi << 32) | lo;
}

/// Read cycle counter for ARM
uint64_t rdtsc() {
  unsigned int val;
  __asm__ __volatile__("mrs %0, cntvct_el0" : "=r" (val));
  return (uint64_t) val;
}
```