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

Exercises for *Foundations in Data Engineering*, WiSe 18/19

Timo Kersten (kersten@in.tum.de)
http://db.in.tum.de/teaching/

Sheet Nr. 03

Exercise 1 Show off how well you can handle bash, pipes and gnu tools by now!

Build a persistent key-value store using only these tools. It should support two actions: **set** and **get**.

Set takes a key and a value and stores it.

Get retrieves the value stored for a given key.

You may assume that keys do not contain a , (comma). Also, it is not necessary to achieve $O(1)$ runtime for set and get as this task is about finding a very short solution. The functions can each be implemented in one line.

Once the implementation is done, do simple performance measurements:

- How long does it take to insert 1000 keys?
- How long does it take to retrieve 1000 keys afterwards?

Exercise 2

1. For the toy example, illustrate how data moves through the machine. Consider two cases:
   a) The input file is read from disk.
   b) The input file is present in the operating system's file cache.
2. With the illustration of the previous exercise, explain the maximum performance for the toy example that was estimated in the lecture.
3. Three implementations of the toy example were shown in the lecture: `awk`, `python` and a first attempt in C++. Why did they not reach the maximum estimated performance? How can the difference in runtimes between the solutions be explained?
4. How does the toy example implementation in C++ (as shown in the lecture) process text?
5. In the lecture, we have seen a variant of the toy example that used a binary file as input instead of csv format. Why is this format more efficient for computing the sum of all quantities?

Exercise 3 We are about to optimize the toy example for better performance. Before we start, let's take a step back and have a look at some experiences with optimization.

"Programmers waste enormous amounts of time thinking about, or worrying about, the speed of noncritical parts of their programs, and these attempts at efficiency actually have a strong negative impact when debugging and maintenance are considered. We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%." – Donald Knuth in Structured Programming with go to Statements

"The First Rule of Program Optimization: Don't do it. The Second Rule of Program Optimization (for experts only!): Don't do it yet." – attributed to Michael A. Jackson

With this in mind, answer the following questions:

1. What is the goal of performance optimization?
2. What are the downsides of manual optimization?
3. What is a good strategy for optimization?
4. Which tools are there to support optimization?

Exercise 4 In the lecture, a programming trick was shown that allows to search for a newline character in an array of characters. The purpose of the following series of questions is to demonstrate how this technique works.

1. What is the general idea of the programming trick? Why do we employ the trick and how does it help?
2. What is a bitwise operation and which operations are available on modern Intel CPUs?
3. Addition is not a bitwise operation. How is it possible to perform 8 byte-wise additions with one (non-SIMD) add instruction?
4. Given 8 consecutive bytes in the variable `block` of type `uint64_t`, which bitwise operation on block sets all bits in a byte to 0 if the byte equals `'\n'` (or 1010 in binary representation or $0x0A$ in hexadecimal)?

   ```
   all0iffNl = <your answer here>;
   ```

5. Given the variable `searchResult` of type `uint64_t`, assume that the highest bit in each byte is 0 (or in other words, each byte is $\leq$ 127). Which operation on `searchResult` will set the highest bit to one in each byte iff the byte is not 0?

   ```
   highestBitSetIffByteNot0 = <your answer here>;
   ```

6. Given the previous result, which operations need to be performed so that the highest bit is 1 and all others are 0 if the highest bit is not set and all 0 if the highest bit is set?

   ```
   highestSetIf0 = <your answer here>;
   ```

7. Assuming all byte values in `block` are $\leq$ 127, i.e. their highest bit is 0, combine the results of the three previous questions to a chain of operations that takes `block` as input and produces a variable of type `uint64_t` in which every byte is 0 if the corresponding byte in `block` is something else than `'\n'` and has only the highest bit set if the corresponding byte equals `'\n'`.
8. Previously, we assumed that all byte values are smaller than 128. Adapt your solution so that it can also handle values greater 128.

Exercise 5 In this exercise, we want to optimize the C++ based approach on the TPC-H dataset corresponding to the following query as seen on the previous exercise sheet:

```
select sum(l_extendedprice) from lineitem
```

First, you should perform simple performance measurements with `time` and `perf` in order to identify the main performance issue. To achive this investigate for each of the following components of your program how much time it requires:

- reading a row of the input file
- locating delimiters
- conversion of a price string to an integer

In the lecture you have learnend about the `mmap` system call which may lead to a signifcant improvement in this situation. Clone the project at https://gitlab.db.in.tum.de/Josef/tpchjoinoptimized and fill in the missing code fragements in all sections marked with "TODO". Your implementation should exploit the aforementioned system call.

Analyze your current solution again with `perf`. What conclusion can you draw now? Which part of your implementation still offers room for improvement? Apply the blockwise programming trick as shown in the lecture in order to improve the execution performance further.

Hint: You may want to add some padding bytes right after the mapped file's region in order to simplify your implementation. It is suifficent to increase the `len` parameter accordingly (bytes after the mapped file's region will be set to zero).

Hint 2: Using the `__builtin_ctzll` built-in function is an efficient way to count the number trailing zero bits in a 64-bit integer. This built-in function might be useful in the implementation of your `find_first` function.

Exercise 6 With memory mapped IO, how is data transferred from the OS file cache to the reading process?