Chair of Applied Computer Science III
Prof. Dr. Guido Moerkotte
Email: moerkotte@uni-mannheim.de
Daniel Flachs
Email: daniel.flachs@uni-mannheim.de
University of Mannheim

Database Systems II
Spring Semester 2019

Solution to Exercise Sheet 3
Created March 12, 2019

Exercise 1

Download the `zip` archive for this exercise sheet from the website. The examples can be found in the folder `experiments`.

Exercise 1 a)

Compile `experiment2.cc` with optimization level 0, i.e., `g++ -O0 experiment2.cc -o experiment2.out`.
Analyze the code and try to explain the output for each of the following runs:

- `./experiment2.out -s -l 990 -u 1000`
- `./experiment2.out -l 990 -u 1000`
- `./experiment2.out -s -l 0 -u 10`
- `./experiment2.out -l 0 -u 10`

Exercise 1 b)

Compare the assembler codes of the functions `singleAmp` and `doubleAmp` given in the following:

```
1  // function with conjunctive predicate connected by single ampersand
2  bool singleAmp (int a, int b) {
3      if ((a == 3) & (b == 5)) {
4        return true;
5      }
6      return false;
7  }
8
9  // function with conjunctive predicate connected by double ampersand
10 bool doubleAmp(int a, int b) {
11     if ((a == 3) && (b == 5)) {
12       return true;
13     }
14     return false;
15 }
```

Recall that you can halt the compiler at every compilation level. As an alternative, you can use the "Compiler Explorer" (`https://godbolt.org/`).

Solution

See slides.

Exercise 1 c)

Compile `experiment3.cc` with optimization level 0, i.e., `g++ -O0 experiment3.cc -o experiment3.out`.
Analyze the code and try to explain the output for each of the following runs:

- `./experiment3.out -i 100`
- `./experiment3.out -i 100 -s1`

Solution

- Both functions sum up the elements in an integer array.
- `sum0` uses a single summation variable, iterates all elements using a for-loop, and always adds the current element: `lSum += aArray[i];`
- `sum1` uses a for-loop iterating the indices of the lower half of the array, and two summation variables, each of the summing up the elements in one half of the array:
  `lSum1 += aArray[i];`
  `lSum2 += aArray[i+lHalf];`
- Observation: `sum1` runs faster than `sum0`.
- Reason: μ-ops parallelism (cf. script)

Exercise 2

Exercise 2 a)

What does the term *universal hashing* mean?

Solution

Universal hashing captures the desired property that distinct keys do not collide too often. The idea is to choose a hash function from a class/family of hash functions *randomly* and *independent* of the values that are stored in it.

Let H be a family of hash functions that map values in U to values in $M := \{0, 1, \ldots, m-1\}$. H is said to be *universal* iff for each pair of distinct keys $k, l \in U$, the number of hash functions $h \in H$ for which $h(k) = h(l)$ is at most

$$\frac{|H|}{|M|}.$$

Equivalently, we can say that, for a hash function h, randomly chosen from the family of hash function H, we have that

$$\Pr[h(k) = h(l)] \leq \frac{1}{|M|},$$

i.e., the chance of a collision between k and l is no more than $\frac{1}{|M|}$. This equals the probability that $h(k) = h(l)$ for randomly chosen values of $h(k)$ and $h(l)$ from the interval $\{0, 1, \ldots, m-1\}$.

Exercise 2 b)

Consider the following four hash functions h_1, h_2, h_3, h_4 that map values from the universe $U := \{0, 1, ..., 5\}$ to the set $D := \{0, 1\}$.

$x \in U$	0	1	2	3	4	5
$h_1(x) \in \{0,1\}$	0	1	0	1	0	1
$h_2(x) \in \{0,1\}$	0	0	0	1	1	1
$h_3(x) \in \{0,1\}$	0	0	1	0	1	1
$h_4(x) \in \{0,1\}$	1	0	0	1	1	0

i. Let $H := \{h_1, h_2\}$. Is H universal?

ii. Let $H' := \{h_1, h_2, h_3\}$. Is H' universal?

iii. Let $H'' := \{h_1, h_2, h_3, h_4\}$. Is H'' universal?

Solution

We check for universality by testing some pairs of values and counting the number of collisions. If there is at least one pair that violates the universality bound, the respective hash family is not universal.

i. $H := \{h_1, h_2\}$ is not universal. The relevant bound is $\frac{|H|}{|D|} = \frac{2}{2} = 1$.

- $\delta_H(1, 4) = 0 \leq 1$ ✓, and $\delta_H(1, 3) = 1 \leq 1$ ✓, but
- $\delta_H(0, 2) = 2 \not\leq 1$ ✗, and $\delta_H(3, 5) = 2 \not\leq 1$ ✗.

ii. $H' := \{h_1, h_2, h_3\}$ is not universal. The relevant bound is $\frac{|H'|}{|D|} = \frac{3}{2} = 1.5$.

- $\delta_H(1,4) = 0 \leq 1.5$ ✓, but
- $\delta_H(0,2) = 2 \not\leq 1.5$ ✗, $\delta_H(1,3) = 2 \not\leq 1$ ✗, and $\delta_H(4,5) = 2 \not\leq 1$ ✗.

iii. $H'' := \{h_1, h_2, h_3, h_4\}$ is universal. The relevant bound is $\frac{|H''|}{|D|} = \frac{4}{2} = 2$.

- $\delta_H(1,4) = 0 \leq 2$ ✓, $\delta_H(0,2) = 2 \leq 2$ ✓, $\delta_H(1,3) = 2 \leq 2$ ✓, $\delta_H(4,5) = 2 \leq 2$ ✓, ...

There are no cases in which any pair $x, y \in U$ produces more than $\frac{|H''|}{|D|} = 2$ collisions. Therefore, H'' is universal.

Exercise 3

Implement a hash table that resolves collisions using chaining. Your hash table must be generic: Make the hash function a parameter. Find some data and insert it into your hash table under different hash functions. Output the average length and the maximum length of the buckets/chains in your hash table. What do you observe?

If you cannot find data, use the `words.txt` file provided in the `zip` archive. `words.txt` contains the 1000 most common words in the English language.

The `zip` archive also contains a hash table implementation which you can use as a framework for your own implementation. You can therefore choose which parts you take from the provided code and which parts you would like to implement yourself.

Solution

See code.