# Exercise Sheet 1: Hashing and Bloom filters

COMS31900 Advanced Algorithms 2019/2020

Please feel free to discuss these problems on the unit discussion board. If you would like to have your answers marked, please either hand them in in person at the lecture or email them to me with the email subject "Problem sheet 1" by the deadline stated.

## 1 Weakly-universal Hashing

A hash function family $H = \{h_1, h_2, \dots\}$ is weakly-universal iff for randomly and uniformly chosen $h \in H$, we have $\Pr(h[x] = h[y]) \leq 1/m$ for any distinct $x, y \in U$. Consider the following hash function families. For each one, prove that it is weakly universal or give a counter-example.

1. Let $p$ be a prime number and $m$ be an integer, $p \geq m$. Consider the hash function family where you pick at random $a \in \{1, \dots, p-1\}$ and then define $h_a : \{0, \dots, p-1\} \to \{0, \dots, m-1\}$ as $h_a(x) = (ax \mod p) \mod m$.

   **Solution.** Let us consider what we have to do to show a counterexample. The claim is that for any prime $p \geq m$ and for all $x \neq y$, $\Pr(h(x) = h(y)) \leq \frac{1}{m}$. So to prove the claim is not true we only need to show *one* prime $p \geq m$, one value for $m$, and one $x \neq y$ where the probability of a collision is greater than $1/m$.

   Consider the case $m = 3$ and $p = 5$. Then we obtain the following table:

   | $h_a(x)$ | $a = 1$ | $a = 2$ | $a = 3$ | $a = 4$ |
   |---|---|---|---|---|
   | $x = 0$ | 0 | 0 | 0 | 0 |
   | $x = 1$ | 1 | 2 | 0 | 1 |
   | $x = 2$ | 2 | **1** | **1** | 0 |
   | $x = 3$ | 0 | **1** | **1** | 2 |
   | $x = 4$ | 1 | 0 | 2 | 1 |

   We see, for example, that when $a \in \{2, 3\}$ then $h_a(2) = h_a(3) = 1$. Observe that $a \in \{2, 3\}$ happens with probability $\frac{1}{2}$. Hence, $\Pr[h_a(2) = h_a(3)] = \frac{1}{2} > \frac{1}{3}$. This family of hash functions is therefore not weakly universal. A similar argument can be made with values $x = 1$ and $x = 4$.

   ✓

2. Let $p$ be a prime and $m$ be an integer such that $p \geq m$. Consider the hash function family where you pick at random $b \in \{0, \dots, p-1\}$ and then define $h_b : \{0, \dots, p-1\} \to \{0, \dots, m-1\}$ as $h_b(x) = ((x + b) \bmod p) \bmod m$.

**Solution.** Again, we construct a counterexample using the values $p = 5$ and $m = 3$. We obtain the following table:

| $h_b(x)$ | $b = 0$ | $b = 1$ | $b = 2$ | $b = 3$ | $b = 4$ |
|---|---|---|---|---|---|
| $x = 0$ | 0 | 1 | 2 | 0 | 1 |
| $x = 1$ | 1 | 2 | 0 | 1 | 0 |
| $x = 2$ | 2 | 0 | 1 | 0 | 1 |
| $x = 3$ | 0 | 1 | 0 | 1 | 2 |
| $x = 4$ | 1 | 0 | 1 | 2 | 0 |

We see that $\Pr[h_b(0) = h_b(3)] = \frac{2}{5} > \frac{1}{3}$. This family of hash functions is therefore not weakly universal. ✓

3. Let $p$ be a multiple of $m$. Consider the hash function family where you pick at random $a \in \{1, \dots, m-1\}$ and $b \in \{0, \dots, m-1\}$. Define $h_{a,b} : \{0, \dots, p-1\} \to \{0, \dots, m-1\}$ as $h_{a,b}(x) = ((ax + b) \bmod p) \bmod m)$.

**Solution.** First, observe that when $p$ is a multiple of $m$ then

$$h_{a,b}(x) = ((ax + b) \bmod p) \bmod m) = (ax + b) \bmod m) \ .$$

Suppose that $p \neq m$ (for example $p = 2m$). Then, consider the values $x = 1$ and $x = m+1$. We have:

$$\begin{aligned} h_{a,b}(1) &= (a + b) \bmod m \ , \text{ and} \\ h_{a,b}(m+1) &= (a(m+1) + b) \bmod m = (a + b + am) \bmod m = (a + b) \bmod m \ , \end{aligned}$$

since $am$ is a multiple of $m$. We thus have $h_{a,b}(1) = h_{a,b}(m+1)$ and thus $\Pr[h_{a,b}(1) = h_{a,b}(m+1)] = 1 \geq \frac{1}{m}$. ✓

## 2 Cuckoo Hashing

1. This question is about cuckoo hashing. Consider a small variant of cuckoo hashing where we use two tables $T_1$ and $T_2$ of the same size and hash function $h_1$ and $h_2$. When inserting a new key $x$, we first try to put $x$ at position $h_1(x)$ in $T_1$. If this leads to a collision, then the previously stored key $y$ is moved to position $h_2(y)$ in $T_2$. If this leads to another collision, then the next key is again inserted at the appropriate position in $T_1$, and so on. In some cases, this procedure continues forever, i.e. the same configuration appears after some steps of moving the keys around to dissolve collisions.

   (a) Consider two tables of size 5 each and two hash functions $h_1(k) = k \bmod 5$ and $h_2(k) = \lfloor \frac{k}{5} \rfloor \bmod 5$. Insert the keys 27, 2, 32 in this order into initially empty hash tables, and show the result.

   **Solution.**

   - Insertion of 27:

| Table 1 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | | | 27 | | |

| Table 2 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | | | | | |

- Insertion of 7:

| Table 1 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | | | 2 | | |

| Table 2 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 27 | | | | |

  2 replaces 27.

- Insertion of 32:

| Table 1 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | | | 27 | | |

| Table 2 | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| | 2 | 32 | | | |

  32 replaces 2. Then 2 replaces 27. Then 27 replaces 32.

✓

(b) Find another key such that its insertion leads to an infinite sequence of key displacements.

**Solution.** Observe that $h_1(2) = h_1(27) = 2$ and $h_2(2) = h_2(27) = 0$. Any number $x$ different to 2 and 27 with $h_1(x) = 2$ and $h_2(x) = 0$ therefore works. The numbers $\{2 + c \cdot 25 \mid c \geq 2\}$ fulfill these conditions (e.g. 52). ✓

2. In order to use cuckoo hashing under an unbounded number of key insertions, we cannot have a hash table of fixed size. The size of the hash table has to scale with the number of keys inserted. Suppose that we never delete a key that has been inserted. Consider the following approach with Cuckoo hashing. When the current hash table fills up to its capacity, a new hash table of doubled size is created. All keys are then rehashed to the new table. Argue that the average time it takes to resize and rebuild the hash table, if spread out over all insertions, is constant in expectation. That is, the expected amortised cost of rebuilding is constant.

**Solution.** Suppose that the algorithm uses $k$ tables. Let $m_1, m_2, \dots, m_k$ with $m_{i+1} = 2 \cdot m_i$ be the sizes of the tables used. As discussed in the lecture, we can insert up to $n_i = \frac{m_i}{c}$ elements into table $i$ with amortized runtime $O(1)$ per insertion, for some large enough constant $c$ (in the lecture we discussed that any value $c \geq 3$ works). The *total runtime* for filling table $i$ is therefore $\frac{n_i}{c} \cdot O(1) = O(\frac{n_i}{c}) = O(n_i)$ (assuming that $c$ is a constant). Observe that $n_{i+1} = 2n_i$ holds, for every $i$.

Next, throughout this process every table (except possibly the last) will be entirely filled. Given $n$ insertions, we thus have $2n > n_k \geq n$. The total runtime is therefore:

$$\begin{aligned}\sum_{i=1}^{k} O(n_i) &= O\left(\sum_{i=1}^{k} \frac{n_k}{2^{i-1}}\right) = O\left(n_k \cdot \sum_{i=1}^{k} \frac{1}{2^{i-1}}\right) = O\left(n_k \cdot \sum_{i=0}^{\infty} \frac{1}{2^i}\right) = O\left(n_k \cdot 2\right) \\ &= O(n_k) = O(n) \ ,\end{aligned}$$

which yields an amortized runtime of $O(1)$ per insertion, since there are overall $n$ insertions. ✓

## 3 Bloom Filters

1. Answer the following three questions about Bloom filters:

   (a) What operations do we perform on Bloom filters?

**Solution.** Bloom filters support INSERT() and MEMBER(). ✓

(b) What is the difference between hash tables and Bloom filters in terms of which data we can access?

**Solution.** Hash tables allow the recovery of the inserted elements. Bloom filters do not allow this. ✓

(c) Why is there is a problem when deleting elements from a Bloom filter?

**Solution.** When deleting an element $x$ we cannot simply set the bits $h_1(x), \ldots, h_r(x)$ to zero since there may be other elements $y$ inserted into the Bloom filter so that $\{h_1(x), \ldots, h_r(x)\}$ and $\{h_1(y), \ldots, h_r(y)\}$ intersect. If this is the case then setting $h_1(x), \ldots, h_r(x)$ to zero will make MEMBER($y$) return 0 instead of 1. ✓

2. Suppose you have two Bloom filters $A$ and $B$ (each having the same number of cells and the same hash functions) representing the two sets $A$ and $B$. Let $C = A\&B$ be the Bloom filter formed by computing the bitwise Boolean *and* of $A$ and $B$.

(a) $C$ may not always be the same as the Bloom filter that would be constructed by adding the elements of the set ($A$ intersect $B$) one at a time. Explain why not.

**Solution.** Suppose that an element $x$ is inserted into $A$ and an element $y \neq x$ is inserted into $B$. Suppose further that $0 < |\{h_1(x), \ldots, h_r(x)\} \cap \{h_1(y), \ldots, h_r(y)\}| < r$. The Bloom filter constructed by adding the elements of the set $A$ intersect $B$ is empty, i.e., all bits are zero. The bits at positions $\{h_1(x), \ldots, h_r(x)\} \cap \{h_1(y), \ldots, h_r(y)\}$ in Bloom Fliter $C$ however are all 1. ✓

(b) Does $C$ correctly represent the set ($A$ intersect $B$), in the sense that it gives a positive answer for membership queries of all elements in this set? Explain why or why not.

**Solution.** Yes. If an element $x$ is contained in both $A$ and $B$ then the bits at positions $\{h_1(x), \ldots, h_r(x)\}$ in both $A$ and $B$ equal 1. The same thus holds for $C$ since $C$ is obtained by computing the logical 'and' between $A$ and $B$. ✓

(c) Suppose that we want to store a set $S$ of $n = 20$ elements, drawn from a universe of $U = 10000$ possible keys, in a Bloom filter of exactly $N = 100$ cells, and that we care only about the accuracy of the Bloom filter and not its speed. For this problem size, what is the best choice of the number of hash functions (the parameter $r$ in the lecture)? (That is what value of $r$ gives the smallest possible probability that a key not in $S$ is a false positive?) What is the probability of a false positive for this choice of $r$?

**Solution.** According to the lecture slides, the probability that $r$ randomly chosen positions are all 1 is

$$(\frac{20r}{100})^r = (\frac{r}{5})^r \ . \tag{1}$$

Again, according to the lecture slides, this expression is minimized for $r = 100/(20e) = \frac{5}{e} \approx 1.839$. We test the two closest integers 1 and 2 in Inequality 1. This shows that a false positive is obtained with probability $\frac{1}{5}$ for $r = 1$ and with probability $\frac{4}{25}$ for $r = 2$. The optimal choice thus is $r = 2$. ✓

## 4 Perfect Hashing

This question is about perfect hashing:

1. Our perfect hashing scheme assumed the set of keys stored in the table is static. Suppose instead that we want to add a few new items to our table after the initial construction. Suggest a way to modify our intitial construction so that we can insert these new items using no new space and without making significant changes to our existing table (in particular, we don't want to change our initial hash function). Your scheme should still do lookups of all items in $O(1)$ time, but you may use a bit more initial space.

2. Suppose now that we want to delete some of our initial items. Describe a simple way to support deletions in our perfect hashing scheme.