Advanced Algorithms - Problem Sheet 1

Dom Hutchinson

February 14, 2020

Question - 1.

A hash function family, $H = \{h1, h_2, \dots\}$ is weakly-universal iff for randomly and uniformly chosen $h \in H$, we have $\mathbb{P}[h(x) = h(y)] \leq 1/m$ for $x, y \in S$ with $x \neq y$.
Consider the following hash function families. For each one, prove that it is weakly universal or given a counter-example.

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

Answer 1 a)
Consider the case where $m = 3$ & $p = 5$.
We have that $a \in [1, 4]$, $h_a : [0, 4] \to [0, 2]$ and $h_a(x) = (ax \mod 5) \mod 3$.
By evaluating every x at every $h_a(\cdot)$ we get

$a\backslash x$	0	1	2	3	4
$h_1(x)$	0	1	2	0	1
$h_2(x)$	0	2	1	1	0
$h_3(x)$	0	0	1	1	2
$h_4(x)$	0	1	0	2	1

By considering this table we see that $\mathbb{P}(h(1) = h(4)) = \frac{2}{4} = \frac{1}{2} > \frac{1}{3} = \frac{1}{m}$.
Thus this is not a *Weakly Universal Hashing Family.*

Question 1 b)
Let p be a prime number and $m \in \mathbb{N}$ with $p \geq m$.
Consider the hash function family where you pick at random $b \in [0, p-1]$ and define $h_b : [0, p-1] \to [0, m-1]$ as $h_b(x) = (x + b \mod p) \mod m$.

Answer 1 b)
Consider the case where $m = 3$ & $p = 5$.
We have that $b \in [0, 4]$, $h_b : [0, 4] \to [0, 2]$ with $h_b(x) = (x + b \mod 5) \mod 3$.
By evaulating every x at every $h_b(\cdot)$ we get

$b\backslash x$	0	1	2	3	4
$h_0(x)$	0	1	2	0	1
$h_1(x)$	1	2	0	1	0
$h_2(x)$	2	0	1	0	1
$h_3(x)$	0	1	0	1	2
$h_4(x)$	1	0	1	2	0

By considering this table we see that $\mathbb{P}(h(0) = h(3)) = \frac{2}{5} > \frac{1}{3} = \frac{1}{m}$.
Thus this <u>is not</u> a *Weakly Universal Hashing Family.*

Question 1 c)
Let p be a prime number and $m \in \mathbb{N}$ with $p \geq m$.
Consider the hash function family where you pick at random $a \in [1, p-1]$ & $b \in [0, m-1]$ and define $h_{a,b} : [0, p-1] \to [0, m-1]$ as $h_{a,b}(x) = ((ax + b) \mod p) \mod m$.

Answer 1 c)
Let $x, y \in [0, p-1]$ with $x \neq y$.
Define $c := ax + b$ & $d := ay + b$.
Note that in order for $(c \mod p) \mod m \not\equiv (d \mod p) \mod m$ we require $c \mod p \not\equiv d \mod p$, this shall be proved first (by contradiction).

Suppose $c \mod p \equiv d \mod p$.
Then $\exists\ e \in \mathbb{Z}$ st $c = d + ep$.
Further

$$\begin{aligned} \implies \quad ax + b &= ay + ep \\ \implies \quad ax = ay + ep & \\ \implies \quad x = y + \tfrac{e}{a}p & \end{aligned}$$

We have a contradiction here. Either $x > p$ (out of bounds), or $x = y$ (contradiction of definitions).
Thus $c \mod p \not\equiv d \mod p$.

Define $f := (c \mod p) \mod m$ & $g := (d \mod p) \mod m$.
We have that $f, g \in [0, m-1]$.
If we fix the value of f and allow g to take any of the m values then $\mathbb{P}(f = g) \leq \frac{1}{m}$ since we know $c \mod p \not\equiv d \mod p$.
Thus $\mathbb{P}(h(x) = h(y)) \leq \frac{1}{m}$ for $x \neq y$.
Thus this <u>is</u> a *Weakly Universal Hashing Family.* □

Question - 2.

Consider a small variant of cuckoo hashing where we use two tables T_1 & T_2 of the same size and hash functions h_1 & 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 leasds 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 hte keys around to dissolve collisions.

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

Answer 2 b)

1) Insert 27 to $h_1(27) = 27 \mod 5 = 2$.
   There are no collisions.

	0	1	2	3	4
T_1	-	-	27	-	-
T_2	-	-	-	-	-

2) Insert 2 to $h_1(2) = 2 \mod 5 = 2$.
Collision with 27.
Move 27 to $h_2(27) = \lfloor \frac{27}{5} \rfloor \mod 5 = 5 \mod 5 = 0$.

	0	1	2	3	4
T_1	-	-	2	-	-
T_2	27	-	-	-	-

3) Insert 32 to $h_1(32) = 32 \mod 5 = 2$.
Collision with 2.
Move 2 to $h_2(2) = \lfloor \frac{2}{5} \rfloor \mod 5 = 0 \mod 5 = 0$.
Collision with 27.
Move 27 to $h_1(27) = 2$.
Collision with 32. Move 32 to $h_2(32) = \lfloor \frac{32}{5} \rfloor \mod 5 = 6 \mod 5 = 1$.

	0	1	2	3	4
T_1	-	-	27	-	-
T_2	2	32	-	-	-

Question 2 c)
Find another key such that its insertion leads to an infinite sequence of key displacements.

Answer 2 c)
Inserting a value x where $h_1(x) = 2$ and $h_2(x) = 0$ will create a value as we will have three values $\{x, 27, 2\}$ trying to fill two places.
The value <u>52</u> would be valid.
If we try inserting 52 we find the following occurs

Insert 52 to $h_1(52) = 52 \mod 5 = 2$.
Collision with 27.
Move 27 to $h_2(27) = 0$.
Collision with 2.
Move 2 to $h_1(2) = 2$.
Collision with 52.
Move 52 to $h_2(52) = \lfloor \frac{52}{5} \rfloor \mod 5 = 10 \mod 5 = 0$.
Collision with 27.
Move 27 to $h_1(27) = 2$.
Collision with 2.
Move 2 to $h_2(2) = 0$.
Collision with 52.
Move 52 to $h_1(52) = 2$.
$\vdots$

Question - 3.

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 teh new table. Argue that the average time it takes to resize and rebuild the has table, if spread out over all insertions is constant in expectation. That is, the expected amortised cost of rebuilding is constant.

Answer 3
We want to find $\mathbb{E}$(time to reize then rehash) a full table of n items.
This is equivalent to $\mathbb{E}$(time to reize) $+$ $\mathbb{E}$(time to rehash) as these are independent.
We have that $\mathbb{E}$(time to resize) $\in O(2n) = O(n)$ and $\mathbb{E}$(time to rehash) $\in O(n)$.
Thus $\mathbb{E}$(time to reize) $+$ $\mathbb{E}$(time to rehash) $\in O(n) + O(n) = O(n)$.
As we are acting on n items this is equivalent to expected amortised cost of $O(1)$.

Question - 4.

Answer the following three questions on Bloom filters.

Question 4 a) - What operations do we perform on Bloom filters?

Answer 4 a)
`insert(k)` which inserts key `k` into the Bloom filter and `member(k)` which returns a boolean stating whether or not key `k` is in the Bloom filter.

Question 4 b) - What is the difference between hash tables & Bloom filters in terms of which data can be accessed?

Answer 4 b)
In Hash tables you can store a value along with a key. When you perform a `lookup(k)` on a hash table you get returned the value `v` which is associated with key `k`.

Question 4 c) - Why is there a problem when deleting elements from a Bloom filter?

Answer 4 c)
Assume that `member(k)` would set $B[h_i(k)] = 0\ \forall\ i \in [1, r]$ where B is the bit string being maintained by the Bloom Filter and $\{h_1(\cdot), \dots, h_r(\cdot)\}$ are the hash functions being used.
When `insert(k)` is performed we set $B[h_i(k)] = 1\ \forall\ i \in [1, r]$. When `member(k)` is performed `true` is returned iff $B[h_i(k)] \equiv 1\ \forall\ i \in [1, r]$. So if at least one of these indexs is not equal to one then `false` is returned. There is a non-zero that there exists $i, j \in [1, m]$ st $h_i(x) = h_j(y)$ for distinct keys x & y. If both these were inserted and then `delete(x)` was performed then $B[h_j(y)]$ would be set to zero. Meaning that `member(y)` would return `false` even though the y has not been removed. Thus B no longer represents all the values stored in the Bloom filter.

Question - 5.

Suppose you have two Bloom Filters A and B (each having the same number of cells and same hash function) representing two sets A and B.
Let $C := A \wedge B$ be the Bloom filter formed by computing the bitwise boolean `and` of A and B.

Question 5 a)
C may not always be the same as the Bloom filter that would be constructed by adding the elements of the set $A \cap B$ one at a time. Explain why.

Answer 5 a)
Let x & y be two distinct keys & suppose we use a hash function, $h(\cdot)$, where $h(x) = h(y)$.
If we insert x into A & y into B only then A & B will have identical bit strings.

Thus $A \wedge B = A = B \implies C = A$.
Note that $A \cup B = \emptyset$ and thus the bloom filter which would be constructed by adding the elements of $A \cup B$ one at a time is just the empty string.
This is not the same as C, which is not empty.

Question 5 b)
Does C correctly represent the set $A \cap B$ in the sense that it gives a positive answer for membership queries of all elements in this set? Explain why, or why not.

Answer 5 b)
Yes.
Suppose we are inserting a key k into both A and B. Since we are using the same hash functions for both, the same indicies in A and B will be set to 1 by this insertion. Since indicies can never be changed by to 0 they will still have a value of 1 after all insertions. Thus we have the indicies k is hashed to will all have a value of 1 after all insertions. These values will still be 1 after bitwise `and` operation and thus will all have a value of 1 in $C := A \wedge B$. `lookup(k)` will hold for all values in $A \cup B$.

Question 5 c)
Suppose that you want to store a set S of $n = 20$ elements, drawn from a universe of $U = 10,000$ possible keys, in a Bloom filter of exactly $N = 100$ cells, and that we care 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, r? (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 value of r?

Answer 5
We expected $\frac{nr}{N} = \frac{20r}{100} = \frac{r}{5}$ bits to be set to 1 in the bit string maintained by this bloom filter.
Thus the probability of `member(k)` producing a false positive is $\left(\frac{r}{5}\right)^r$.
We want to find the value of r which minimises $\left(\frac{r}{5}\right)^r$.
Set $y = \left(\frac{r}{5}\right)^r$ and take the derivative wrt r

$$\begin{array}{rrcl}
\text{Let} & y & = & \left(\frac{r}{5}\right)^r \\
\implies & \ln y & = & r[\ln r - \ln 5] \\
\implies & \frac{d}{dr}(\ln y) & = & \frac{d}{dr}(r[\ln r - \ln 5]) \\
\implies & \frac{1}{y}\frac{dy}{dr} & = & [\ln r - \ln 5] + r[\frac{1}{r} - 0] \\
\implies & \frac{dy}{dr} & = & y(\ln r - \ln 5 + 1) \\
 & & = & \left(\frac{r}{5}\right)^r(\ln r - \ln 5 + 1) \\
\text{Set} & \frac{dy}{dr} & = & 0 \\
\implies & 0 & = & \left(\frac{r}{5}\right)^r(\ln r - \ln 5 + 1) \\
\implies & 0 & = & \left(\frac{r}{5}\right)^r(\ln r - \ln 5 + 1) \\
 & & & \text{No solutions for } r \\
\text{or} & 0 & = & \ln r - \ln 5 + 1 \\
\implies & \ln r & = & \ln 5 - 1 \\
\implies & r & = & e^{\ln 5 - 1} \\
 & & = & \frac{5}{e} \\
 & & \approx & 1.839
\end{array}$$

We require that r is an integer so test $r = 1$ and $r = 2$ to find the lowest value.
$r = 1 \implies y = \frac{1}{5}$, $r = 2 \implies y = \left(\frac{2}{5}\right)^2 = \frac{4}{25} < \frac{1}{5}$.
Thus using $r = 2$ hash functions produces the most accurate bloom filter in this scenario.

Question - 6.

This question is about perfect hashing.

Question 6 a)
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.

Answer 6 a)
When inserting a new item, x, use the initial hashing function to find which T_i to insert x into. Then use h_i to find where to place x in T_i. If $T_i[h_i(x)]$ is already occupied then rehash T_i with a new hashing function. Note that eventually this will eventually fill up but as T_i has n_i^2 spaces & initially only had n_i elements in it, there are $n_i(n_i - 1)$ free spaces so this will take a while for reasonably large n.

Question 6 b)
Suppose now that we want to delete some of our initial terms. Describe a simple way to support deletions in our perfect hashing scheme.

Answer 6 b)
Suppose we want to delete an entry with key x from the table produced by the FKS hashing scheme. Use $h(\cdot)$ to find which T_i x is in, then set $T_i[h_i(x)] = \texttt{NULL}$. If we are unsure whether there is an entry with key x in the table then we should check that the key of the element at position $T_i[h_i(x)]$ is equal to x.