28.05.14 no lecture

last week of semester:

test exam at home

2.4. Randomized Hashing

Problem: given large universe of elements U (finite)
subset $S \subseteq U$

Goal: store S as efficient as possible / while allowing quick look-ups

2.4.1 Universal Hashing Principle

- array of size m to store elements
- at each position a linked list
- function $h: S \to \{0, \dots, m-1\}$ to map / hash the elements in S

$|S| = n$

↳ look-up time for an element:
time to evaluate hash function
plus time to scan through the linked list at the respective position

⇒ no cond. hash function possible, because we could redetect elements

Lemma For any hash function h, if $|U| \geq (n-1)(m+n)$, there exists a subset $S \subseteq U$ of size m such that all items in S are mapped to the same position.

Proof. by contradiction

Assume every location in the hash table contains at most $n-1$ elements from U; then the total number of elements in U is $\leq (n-1) \cdot m < |U|$ ↯

□

Example: Bad Hashing

Let U be the set of integers $\leq 2^m$ and $h(u) = u \bmod m$ the used hash function.

If we choose S as $s_1 = m+c$, $s_2 = 2m+c$, ... $s_i = im+c$ with c some constant $< m$, then all the elements in S will be hashed to position c

$m = 5$ $\quad h(u) = u \bmod 5$

$S = 11, 16, 21, 26, \ldots$

So as soon as we fix a det. hash function, someone evil can construct an input which leads to search times of $O(n)$.

Idea: Create a family of hash functions H and pick $h \in H$ randomly on demand

$\Rightarrow$ we will show that for any kind of input S we will have expectedly a constant-time look-up operation

<u>Definition</u> (universal)

A family of hash functions H is called universal if $\forall u, v \in U \quad u \neq v$:

$$P_{h \in H}(h(u) = h(v)) \leq \frac{1}{m}$$

or in other words

$$|\{h \in H : h(u) = h(v)\}| \leq \frac{|H|}{m}.$$

c-universal:

$$P_{h \in H}(h(u) = h(v)) \leq \frac{c}{m}$$

$c = 2$ near universal

Lemma 2.13 (Expected Number of Collisions)

If H is universal, then for any set $S \subseteq U$ with $|S| = m$ and any $u \in U$ the expected number of collisions of u and elements in S is at most $\frac{n}{m}$.

Proof: Let X denotes the number of collisions with u, $E(X)$ the exp. value.

$$X_i = \begin{cases} 1 & \text{if } u \text{ collides with elem } S_i \in S \\ 0 & \text{otherwise} \end{cases}$$

$i = 1, \dots, n$

$$E(X) = \sum_{i=1}^{n} E(X_i)$$

$$E(X_i) = P(X_i = 1)$$

H is universal, hence

$$E(X_i) \leq \frac{1}{m}$$

$$\Rightarrow E(X) = \sum_{i=1}^{n} E(X_i) \leq n \cdot \frac{1}{m} = \frac{n}{m} \quad //$$

if we 100 elements and 5 positions, at least one position holds $\geq \frac{100}{5}$ elems

2) in general at least one position

is filled with $\frac{n}{m}$ elements $\Rightarrow$ in expectation a universal hash family works despite concerning the load factor

If we have $m \in O(n)$, we expect a constant number of collisions for a single element and therefore constant query times.

We are fine, except we have no idea how H looks like.

2.4.2 Designing a Universal Hash Family

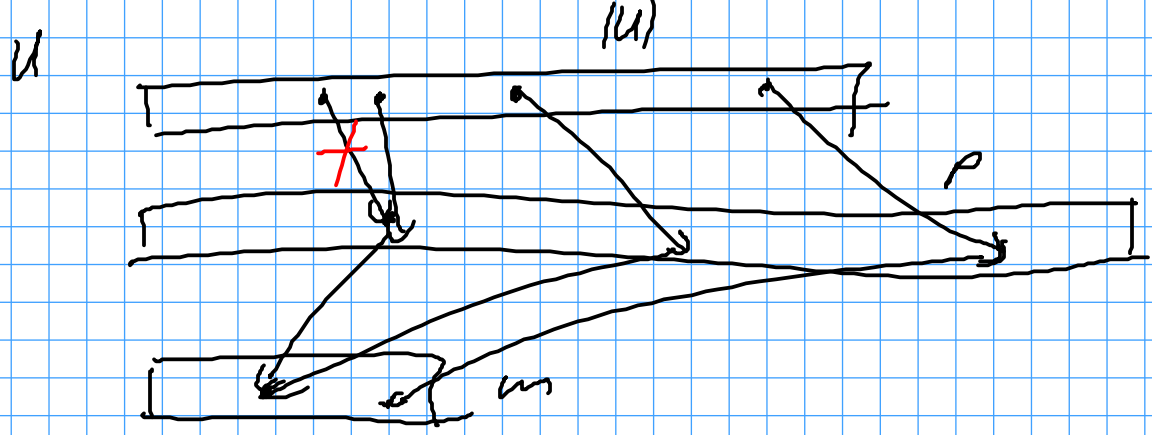
We now consider the keys to hash as integers and w.l.o.g. to be the set $\{1, \dots, |U|\}$.
There always exists a universal hash family H, namely the set of all functions

$$f: U \to \{0, \dots, m-1\}$$

Exercise: Prove that this family is universal.

No practical use :-.

Lemma Let p be a prime with $p \geq |U|$. Let H be the hash family that contains $h_a(u) = (a \cdot u \bmod p) \bmod m$ for each $a \in \{1, \dots, p-1\}$. Then H is nearly universal.



Proof. We have to show that the collision probability is bounded by $\frac{2}{m}$.

First, we show that for two elements $u, v \in U$, $u \neq v$ it yields

$(au \bmod p) \neq (av \bmod p)$, so no collisions are introduced in the first mapping. Assume otherwise for contradiction. Then we have:

$$(au \bmod p) = (av \bmod p)$$

$$au - av = k \cdot p \qquad k \in \mathbb{N}$$

$$a(u-v) = k \cdot p$$

$$|u-v| \leq |U| < p \quad \text{(by choice of } p\text{)}$$

Furthermore $a < p$ (by construction).

We have two terms smaller than p, hence their product can not be a multiple of p. ~~Z~~ to the assumption of $(au \bmod p) = (av \bmod p)$

$\Rightarrow$ no collision after the first mapping ($\triangleq$ if we would have a

$\sum + 3 = 8$

$3 \cdot \sum + 3 = 18$

$18 - 8 = 10$

hash table of size p no collisions would occur)

$\Rightarrow$ but. hash table size is much less p

So the second part of the mapping might introduce collisions, though.

$(au \bmod p) = q$

$(av \bmod p) = q'$

if $q \bmod m = q' \bmod m$

we have

$$q - q' = km \quad \text{for } k \in \mathbb{Z}$$

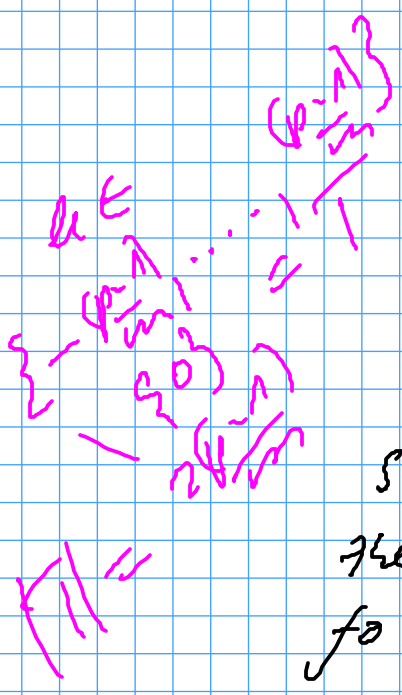
or in other words

$$q' = q - km$$

We know that $k \neq 0$ because otherwise $(au \bmod p) = (av \bmod p)$ which we disproved.

We would like to have an upper bound on k. In fact

$|k| \leq \frac{(p-1)}{m}$ because $q \leq p-1$ and $q' \geq 0$.



so the probability of a collision is the prob. of selecting the specific value for a that led to q, q', multiplied

with the number of possible values for h. Hence we get

$$\frac{1}{p-1} \cdot 2 \cdot \frac{(p-1)}{m} = \frac{2}{m} \; . \; //$$

prob. for certain choice of a (as h_a is selected u.a.r)

Why does this solve our problem?

If $|U| \leq 2^w$ we can choose p between 2^w and 2^{w+1}. (Bertrand's postulate: for $n > 1$ there always exists a prime p with $n < p < 2n$. Proven correct by Chebyshev). So for $n = 2^w$ we find a prime for sure, if we check the next 2^w many elements. Moreover we can store H now very compact, indeed we only need to store p and a. Also the hash function can be evaluated in constant time. Plugging in our Universality in Lemma 2.3. we get: The expected number of collisions is $\frac{2n}{m}$. Therefore

for a hash table of size cn with c being a small constant, we only expect $\frac{2}{c}$ collisions per entry in the hash table. Hence the total space consumption is linear in the set size of S and a look-up is expectedly a constant time operation.

2.4.3. Fingerprinting

Basic problem: Two complex, large objects should be tested for equality.

Example: Alice and Bob have a (long log) script and want to know if the version is the same. Alice could send Bob her script, Bob could check everything and report back to Alice yes or no.

⇒ Drawback:
- → lot of communication necessary
- → large evaluation time for Bob

⇒ Advantage:
- Result is always correct.

↝ Idea: reduce communication and evaluation time by applying hashing

⇒ Alice fixes a hash function h which reduces her script to u bits and then sends h and h(Alice's script) to Bob

⇒ Bob computes h(Bob's script) and compares to h(Alice script) and reports back to Alice

Advantage:

⇒ fixed communication costs

⇒ fixed evaluation time (time to hash the complex objects + comp.)

Disadvantage:

Bob might report the scripts to be equal, but they are not.

⇒ one-sided error

Exercise: Alice uses a universal hash family H to choose her h.f. to hash her

script. What is the probability of error if the scripts of Alice and B have a length of n bits and the hash function reduces it to 6 bits.

How does the probability change if repeating the process c times?