

Problem Set 11

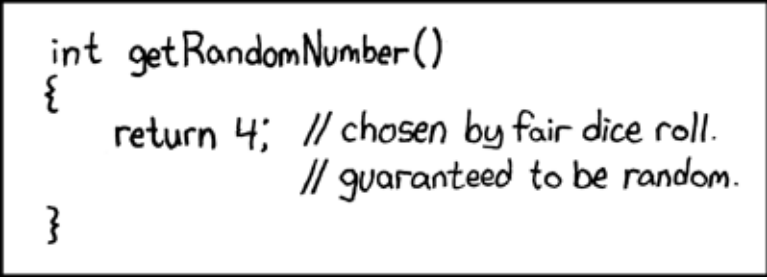
CS 331H

Due Monday, May 6

1. Give an algorithm that takes a function $h : [U] \rightarrow [m]$ and finds pairs of inputs that collide. If h is perfectly random, your algorithm should use $O(1)$ space and $O(\sqrt{m})$ expected time. **Hint:** Relate this problem to the classic problem of finding a loop in a linked list:

https://en.wikipedia.org/wiki/Cycle_detection

2. The Python programming language uses hash tables for “dictionaries” internally in many places. Until 2012, however, the hash function was not randomized: the same one was used every time, so keys that collided in one Python program would do so for every other program.



```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

Figure 1: How Python’s hash function was chosen pre-2012. (Credit: XKCD)

To avoid denial of service attacks, Python implemented hash randomization—but there was an issue with the initial implementation. Also, in Python 2, hash randomization is still not the default: one must enable it with the `-R` flag.

Find a 64-bit machine with both Python 2.7 and Python 3 (≥ 3.4); one is available at `linux.cs.utexas.edu`.

- (a) First, let’s look at the behavior of `hash(“a”) – hash(“b”)` over $n = 2000$ different initializations. If `hash` were pairwise independent over the range (64-bit integers, on a 64-bit machine), how many different values should we see appear?

- (b) How many different values *do* appear, for three different instantiations of Python: (I) no randomization (`python2`), (II) Python 2's hash randomization (`python2 -R`), and (III) Python 3's hash randomization (`python3`)?

Hint: You may find this tiny shell script, which runs a command 2000 times, useful:

```
for i in `seq 1 2000`; do
    python2 -R -c 'print(hash("a")-hash("b"))'
done
```

- (c) What might be going on here? Roughly how many “different” hash functions does this suggest that each version has?
- (d) The above suggests that Python 2's hash randomization is broken, but does not yet demonstrate a practical issue. Let's show that large collision probabilities happen. Observe that the strings “8177111679642921702” and “6826764379386829346” hash to the same value in non-randomized Python 2.

Check how often those two keys hash to the same value under `python2 -R`. What fraction of runs do they collide? To estimate this, run it until you've seen it collide at least 10 times.

How might an attacker use this sort of behavior to denial of service attack a website (i.e., to make the website slow)?

- (e) Using your algorithm from problem 1, find two other strings that collide under Python 2's non-randomized hash function.

Note: Your code may take about an hour to run. Try not to wait until the last minute.