Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Department of Computer Science
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger
Tomáš Gavenčiak

1st December 2016

Algorithms & Data Structures Solutions to Sheet P10 AS 16

Solution P10.1 *Finding a subarray.*

The solution, also known as Rabin-Karp algorithm, is mostly sketched in the task text, but we describe it in detail.

We first need to compute `cToK` $= c^k \bmod m$ by applying `cToK = (cToK * c) % m` from `cToK = 1` k times, taking $O(k)$ time. Note the modulo on every step is important to avoid overflow. We also can not afford to recompute $c^k \bmod m$ for every index of A we look at.

Then compute the hash `HB` $= H(B)$ by, for example, applying the hash update formula[1] k times and compute the hash `HA` $= H(a_0 \dots a_{k-1})$ in the same way. Then for all possible starting indices of B in A, $i = 0, \dots, n-k$, compare the current hash `HA` to `HB`, if equal, check `A[i+j]==B[j]` for $j = 0, \dots, k-1$ and if yes, output the index i. Then increase i and move the hash window by one to the right with the formula from the task text.

We can expect only very few hash collisions (the proposed parameters generate cca 20 in the judge), so checking the extra "false" matches costs very little time.

Note that this could be also solved with a Knuth-Morris-Pratt (originally for text searching) when the values are reasonably small (up to 1000 in our case) but we leave that as bonus material to the interested student – you can find a good description on Wikipedia.

Solution programs

On the lecture website, you can find a solution running in time $O(n + kp)$ when p is the number of matches found. The solution source contains further comments on the implementation.

Data

The test `judge1` contained several smaller sets with special cases, the test `judge2` added long repetitive sequences like $A = 0^{100000}10^{100000}$ and $B = 0^{10000}10^{10000}$ where a $O(nk)$ solution would take a very long time to realize the mismatch even if it tried to go backwards. `judge3` then added a long random sequence to generate some hash value collisions (probably even for different c and m, if m is not much larger).

Notes on submitted solutions. A very common problem with the solutions were mistakes in computing the sliding window hash function. As noted in the task text, you can do the modulo m operation as often as you want, and often enough to never see Java integer overflow, for example use modulo on every step of computing $c^k \bmod m = (\dots(((cc \bmod m)c \bmod m)c \bmod m)\dots)c \bmod m$.

[1] Using 0 as a neutral element "leaving" the hash window.

Also, when subtracting on the hash function update, you need to make sure the result is positive modulo m.

Unfortunately hash functions are generally harder to debug, since the hash values have no easily visible meaning.