Rabin & Karp Algorithm

Rabin-Karp – the idea

- Compare a string's hash values, rather than the strings themselves.
- For efficiency, the hash value of the next position in the text is easily computed from the hash value of the current position.

How Rabin-Karp works

- Let characters in both arrays T and P be digits in radix-Σ notation. ($\Sigma = (0,1,...,9)$)
- Let p be the value of the characters in P
- Choose a prime number *q* such that fits within a computer word to speed computations.
- Compute (p mod q)
  - The value of p mod q is what we will be using to find all matches of the pattern P in T.

How Rabin-Karp works (continued)

- Compute (T[s+1, .., s+m] mod q) for s = 0 .. n-m
- Test against P only those sequences in T having the same (mod q) value
- (T[s+1, .., s+m] mod q) can be incrementally computed by subtracting the high-order digit, shifting, adding the low-order bit, all in modulo q arithmetic.

A Rabin-Karp example

- Given T = 31415926535 and P = 26
- We choose q = 11
- P mod q = 26 mod 11 = 4

3	1	4	1	5	9	2	6	5	3	5

31 mod 11 = 9 not equal to 4

3	1	4	1	5	9	2	6	5	3	5

14 mod 11 = 3 not equal to 4

3	1	4	1	5	9	2	6	5	3	5

41 mod 11 = 8 not equal to 4

Rabin-Karp example continued

3	1	4	1	5	9	2	6	5	3	5

15 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

59 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

92 mod 11 = 4 equal to 4 -> spurious hit

3	1	4	1	5	9	2	6	5	3	5

26 mod 11 = 4 equal to 4 -> an exact match!!

3	1	4	1	5	9	2	6	5	3	5

65 mod 11 = 10 not equal to 4

Rabin-Karp example continued

3	1	4	1	5	9	2	6	5	3	5

53 mod 11 = 9 not equal to 4

3	1	4	1	5	9	2	6	5	3	5

35 mod 11 = 2 not equal to 4

As we can see, when a match is found, further testing is done to insure that a match has indeed been found.

Analysis

- The running time of the algorithm in the **<u>worst-case</u>** scenario is bad.. But it has a good **<u>average-case</u>** running time.

- O(mn) in worst case
- O(n) if we're more optimistic…
  - Why?
  - How many hits do we expect? (board)

Multiple pattern matching

- Given a text $T=T_1 \ldots T_n$ and a set of patterns $P_1 \ldots P_k$ over the alphabet Σ, such that each pattern is of length m, find all the indices in T in which there is a match for **one** of the patterns.
- We can run KMP for each pattern separately.
- O(kn)
- Can we do better?

Bloom Filters

- We'll hold a hash table of size O(k) (the number of patterns)
- For each offset in the text we'll check whether it's hash value matches that of **any** of the patterns.

Analysis

- Expected: $O(\max(mk, n))$