String Matching Using The **Rabin-Karp** Algorithm

Outline

- Definition of the Rabin-Karp algorithm
- How Rabin-Karp works
- A Rabin-Karp example
- Complexity
- Real Life applications

String Matching Problem

- The idea of the string matching problem is that we want to find all occurrences of the pattern P in the given text T.
- We could use the brute force method for string matching, which utilizes iteration over T. At each letter, we compare the sequence against P until all letters match of until the end of the alphabet is reached.
- The worst case scenario can reach O(N*M)

Definition of Rabin-Karp

- A string search algorithm which compares 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)

- The Rabin-Karp string searching algorithm calculates a **hash value** for the pattern, and for each M-character subsequence of text to be compared.
- If the hash values are unequal, the algorithm will calculate the hash value for next M-character sequence.
- If the hash values are equal, the algorithm will do a Brute Force comparison between the pattern and the M-character sequence.
- In this way, there is only one comparison per text subsequence, and Brute Force is only needed when hash values match.

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.

Rabin-Karp example (Alphabets)

- Let’s say that our alphabet consists of 10 letters.
- our alphabet = **a, b, c, d, e, f, g, h, i, j**
- Let’s say that “a” corresponds to 1, “b” corresponds to **2** and so on.

The hash value for string “cah” would be ...

$$\mathbf{3*100 + 1*10 + 8*1 = 318}$$

Complexity

- The running time of the Rabin-Karp algorithm in the worst-case scenario is O(n-m+1)m but it has a good average-case running time.
- If a sufficiently large prime number is used for the *hash function*, the hashed values of two different patterns will usually be distinct.
- If the expected number of valid shifts is small O(1) then the Rabin-Karp algorithm can be expected to run in time O(n+m) plus the time to required to process spurious hits.

Applications

- Bioinformatics
  - Used in looking for similarities of two or more proteins; i.e. high sequence similarity usually implies significant structural or functional similarity.