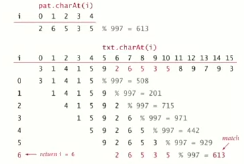
Rabin-Karp

Basic idea = modular hashing.

* Compute a hash of pattern characters – to M – 1
* For each i, compute a hash of text characters i to M + i – 1
* If pattern hash == text substring hash, check for a match



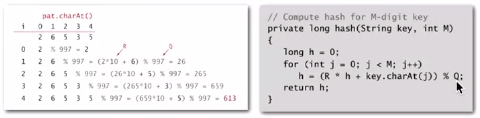
Efficiently computing the modular hash function

Using the notation ti for txt.charAt(i), we wish to compute:



Intuition is: M-digit, base-R integer, modulo Q

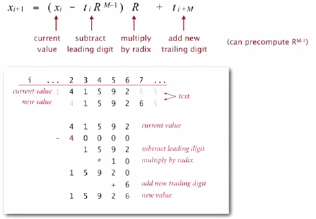
We can use **Horner’s method**: this is a linear-time method to evaluate degree-M polynomial.



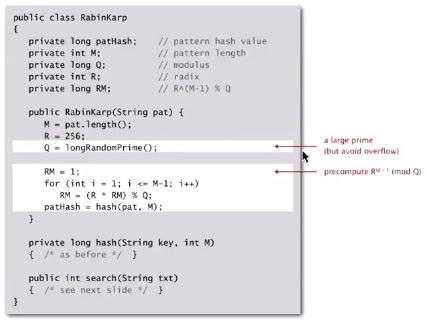
Challenge: How do we efficiently compute xi+1 given that we know xi ?



Key property: update hash function in constant time!!

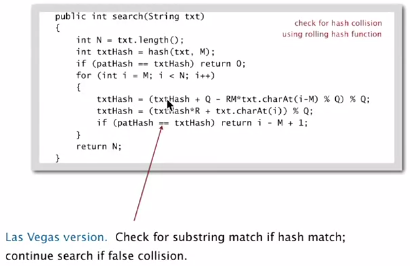


Rabin-Karb Java Implementation



Search implementation

Monte Carlo version: return match if hash match



Performance analysis

Theory: If Q is a sufficiently large random prime (about M N2),  
then the probability of a false collision is about 1 / N

Practice: Choose Q to be a large prime (but not so large to cause overflow). Under reasonable assumptions, probability is about 1/Q

Monte Carlo version:

* Always runs in linear time
* Extremely likely to return correct answer (but not always !)

Las Vegas version:

* Always returns correct answer
* Extremely likely to run in linear time (but worst case is M N)

Rabin-Karp can be extended to other, more general situations

Advantages:

* Extends to 2d patterns
* Extends to finding multiple patterns

Disadvantages:

* Arithmetic ops slower than char compares
* Las Vegas method version requires backup
* Poor worst-case guarantee