

# Rabin-Karp Algorithm

## CS 491 – Competitive Programming

Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN  
DEPARTMENT OF COMPUTER SCIENCE

Spring 2023

# Objectives

- ▶ Explain how a *rolling hash* works
- ▶ Use a rolling hash to find a pattern in a string quickly



# Näive String Matching

Consider this code

```
1  int find(string s, string desire) {
2      int found = -1;
3      for(int i=0; i<s.length() - desire.length(); ++i) {
4          found = 0;
5          for(int j=0; j<desire.length(); ++j)
6              if (s[i+j] != desire[j])
7                  break;
8              else ++found;
9          if (found == desire.length())
10             return i;
11     } // end for i
12     return -1; // not found
13 }
```

► What is the time complexity?

# Hash Functions

- ▶ Remember hash functions!
  - ▶  $h(s)$  should be *fast to compute*
  - ▶  $h(s_1) = h(s_2)$  *probably* means  $s_1 = s_2$
  - ▶  $h(s_1) \neq h(s_2)$  *definitely* means  $s_1 \neq s_2$
- ▶ Can this help us with string matching?

# Hashing

Consider this code

```
14  int find(string s, string desire) {  
15      int found = -1;  
16      for(int i=0; i<s.length() - desire.length(); ++i) {  
17          if (h(desire) == h(s.substr(i,desire.length()))) &&  
18              desire = s.substr(i,desire.length())  
19              return i  
20      } // end for i  
21      return -1; // not found  
22  }
```

► How about now?

# Rolling Hashes

- ▶ Consider this hash function:

$$h(c_0 \cdots c_{n-1}) = c_0 a^n + c_1 a^{n-1} + c_2 a^{n-2} + \cdots + c_{n-1} \text{ modulo } b$$

- ▶  $a$  is a constant (256 is reasonable)
  - ▶  $b$  is a large prime number (let's use 1000007)
  - ▶  $c_i$  is the  $i$ th character in a string.
- ▶ Try it yourself!
  - ▶ Compute the hash for abc
  - ▶ Compute the hash for bci
- ▶ Hint: ASCII for a is 95, i is 103

# Rolling Hashes, ctd

- ▶ Consider this hash function:

$$h(c_0 \cdots c_{n-1}) = c_0 a^n + c_1 a^{n-1} + c_2 a^{n-2} + \cdots + c_{n-1} \text{ modulo } b$$

- ▶  $h(\text{"abc"}) = 95 \times 256^2 + 96 \times 256 + 97 \text{ mod } 100007 = 50159$
- ▶  $h(\text{"bci"}) = 96 \times 256^2 + 97 \times 256 + 103 \text{ mod } 100007 = 15950$
- ▶ Can you convert from one to the other quickly?
  - ▶ Add 100007 to prevent "going negative"
  - ▶ "Subtract off" a by subtracting  $(95 \times 256^2 \text{ mod } 100007)$
  - ▶ Multiply the remainder by 256 and modulo 100007.
  - ▶ Add 103
  - ▶ Take the modulus again.
- ▶ So:  $h(\text{"bci"}) = ((h(\text{"abc"}) + 100007 + (95 \times 256^2 \text{ mod } 100007)) \times 256 + 103) \text{ mod } 100007$

## Setting up

```
23  int a = 256;
24  int b = 100007;
25
26  int pow = 1;
27  for(i=0; i<desire.length(); ++i)
28      pow = (pow * a) % b;
29
30  int hashes = 0;
31  int hashd = 0;
32
33  for(i=0; i<desire.length(); ++i) { % assume desire < s
34      hashes = hashes * a + s[i];
35      hashd = hashd * a + desire[i];
36  }
```



# Matching Part

```
37 while (i < s.length() - desire.length()) {
38     if (hashs == hashsd &&
39         desire = s.substr(i,desire.length()))
40         return i;
41     hashs = (b + hashs) - (s[i] * pow % b)
42         + s[i+desire.length()];
43     hashs = hashs * a % b;
44 }
45 return -1; // if failed.
```