1 Overview

- Signatures
- Hash table resizing
- Review of rolling hash

2 Hash Table Resizing

Question: How do we pick hash table size? Solution: Pick small and grow if necessary.

But this looks expensive!

Ahhh... but think of it as a small cost per element.

Each element inserted into small table, each element rehashed, inserted into new table: constant time per element!

Another way: Small table size \( m \). Time to insert \( O(m) \) elements: \( O(m) \), time to rehash: \( O(m) \). Total time for \( m \) elements: \( O(m) \Rightarrow O(1) \) per element!

3 Signatures

Problem: Sometimes comparing two values takes a long time. Example: Strings have length \( L \), comparison takes \( O(L) \). If we hash \( n \) strings to size-\( n \) table, we get \( O(n) \) collisions. Each collision requires work \( O(L) \), gives \( O(nL) = O(n^2) \). That’s really bad!

Solution: Table of size \( n^2 \). Now \( O(1) \) collisions, so total work \( O(n) \).

But! We don’t need a table of size \( n^2 \), just a way of comparing strings that isn’t \( O(L) \). So store \( n^2 \) hash with string in size-\( n \) table! Then compare \( n^2 \)-hashes (called signatures) = \( Pr(1/n^2) \) that two strings have same signature. So now \( O(1) \) comparison work on average.

4 Rolling Hash

Idea: Hash functions can be related!

Example: Hashing strings “the” and “her”

Converting to numbers:
\( \text{“the”} = (t \cdot (26)^2 + h \cdot (26) + e) \)

\( \text{“her”} = (h \cdot (26)^2 + e \cdot (26) + r) = 26(\text{“the”}-t) + r \)

In general: Converting to base-\( b \) numbers using:

\[
N(S) = S_0 b^L + S_1 b^{L-1} + S_2 b^{L-2} + \ldots + S_{L-1} b + S_L
\]

Given \( S \) and \( S' = S_{0:L} \) and \( S'' = S_{n:L+M+n} \)

\[
N(S'') = b^{M+n} (N(S') - b^{L-n} N(S'_{0:n})) + N(S'_{L+1:L+n+M})
\]

Mod properties:

\[
ab \mod m = ((a \mod m)(b \mod m)) \mod m
\]

\[
(a + b) \mod m = ((a \mod m) + (b \mod m)) \mod m
\]

\[
h_m(S) = N(S) \mod m = (((S_0 \mod m)(b^L \mod m)) \mod m)+\ldots+S_L \mod m) \mod m
\]

\[
h_m(S'') = N(S'') \mod m
= (b^{M+n} (h_m(S') - b^{L-n} h_m(S'_{0:n})) + h_m(S'_{L+1:L+M})) \mod m
\]

Just store division hash!

One character move:

\[
(b(h_m(S') - b^{L-1} h_m(S'_0)) + h_m(S''_{L+1})) \mod m
\]

Constant time hash calculation!

Can be used for string matching (Rabin-Karp):

Given string \( S \) and text \( T \)

- Compute \( h_m(S) \)
- Compute hash for each string of length \( L \) in \( T \)
- If hash = \( h_m(S) \), compare strings character-by-character \( O(L) \)

Time: \( O(|S| + |T| - |S| + |S|c) = O(|T| + |S|c) \)

Using signatures, \( c \) is \( 1/|T| \).