6.006 Recitation

Build 2008.12
PSI Solutions

- Posted on homework page
- Password-protected
  - Please write down username/password
Coming Up Next...

- More hashing!
- Rabin-Karp (String Matching)
  - vs the dumb naive algorithm
- Rolling Hashes
- Black Magic: why it works
Hashing without tables

- Fancy names: fingerprint, message digest
- Idea (hashing repeated):
  - given an object, compute a summary that’s easier to work with
- Very versatile concept! Don’t forget it!!
Hashing human beings
Hashing human beings

- Want something easy to handle
Hashing human beings

- Want something easy to handle
  - fingerprints (doh)
  - DNA samples
  - iris scans
  - face picture
Naive String Matching

- Want to find pattern in text
- Slide pattern over text one by one character
- If pattern matches overlapping characters of text, report match
Rabin-Karp

- Want to find pattern in text
- Slide pattern over text one by one character
  - If hash(pattern) matches hash(overlapping characters of text)
  - If pattern matches overlapping characters of text report match
Making Rabin-Karp fast

- Good hash function
- If many false positives, then many useless full-string comparisons
- Fast hash update when “sliding” pattern across text
- If we rehash every time, might as well use naive string comparison
Introducing Rolling Hashes

• Data Structure (just like hash table)
• start with empty list
• append(val): appends val at the end of list
• skip(): removes the first list element
• hash(): computes a hash of the list
But we have strings

- Characters are numbers (ASCII, Unicode)
- ‘A’ = 65, ‘B’ = 66
- Then strings are lists of numbers
  - “Boom! Headshot” = [66, 111, 111, 109, 33, 32, 72, 101, 97, 100, 115, 104, 111, 116]
- So we can work with lists of numbers
Building Rolling Hashes

• Key Idea: use division method for hashing
• “concatenate” list items into big number
• hash value: big number mod prime
• reason: skip() is doable (not true for most other hashing methods)
class AmnesiacRollingHash:
    def __init__(self, base = 256, prime = 1009):
        self.hash_value = 0
        self.base = base
        self.prime = prime
        # inv_base is computed s.t. (base * inv_base) % prime == 1
        self.inv_base = pow(base, prime - 2, prime)
        self.skip_multiplier = 1

    def append(self, value):
        self.hash_value = (self.hash_value * self.base + value) % self.prime
        self.skip_multiplier = (self.skip_multiplier * self.base) % self.prime

    def skip(self, value):
        self.skip_multiplier = (self.skip_multiplier * self.inv_base) % self.prime
        self.hash_value = (self.hash_value + self.prime - (value * self.skip_multiplier) % self.prime) % self.prime
Hashing Intuition
Hashing Intuition

- Base 100, modulo 23
- Hash [61, 8, 19, 91, 37]
Hashing Intuition

- Base 100, modulo 23
- Hash $[61, 8, 19, 91, 37]$  
  - $(6108199137 \mod 23) = 12$
- Hash $[a_3, a_2, a_1, a_0]$
Hashing Intuition

• Base 100, modulo 23
• Hash [61, 8, 19, 91, 37]
  • \((6108199137 \mod 23) = 12\)
• Hash \([a_3, a_2, a_1, a_0]\)
  • \((a_3 \cdot 100^3 + a_2 \cdot 100^2 + a_1 \cdot 100^1 + a_0 \cdot 100^0) \mod 23\)
Sliding Intuition

- Base 100, mod 23
- List: [3, 14, 15, 92, 65, 35, 89, 79, 31]
- [3, 14, 15, 92, 65] to [14, 15, 92, 65, 35]
  - get from 11 to 6
- [14, 15, 92, 65, 35] to [15, 92, 65, 35, 89]
  - get from 6 to 5
Simple Rolling Hashes

- formulas for updating the hash value on append and skip
Fast Rolling Hashes

- need to avoid exponentiation in skip
- cache the result \( (\text{base}^{\times \text{length}} \mod p) \)
- append: multiply by base
- skip: divide by base
  - can’t divide, use multiplicative inverse
Python design

• Step 1: Amnesiac Hash -- forgets list items
  • need to remind skip() what’s the front element of the list

• Step 2: Easy Hash -- keeps track of items
  • builds upon Amnesiac Hash
  • keeps track of list items
class AmnesiacRollingHash:
    def __init__(self, base = 256, prime = 1009):
        self.hash_value = 0
        self.base = base
        self.prime = prime
        # inv_base is computed s.t. (base * inv_base) % prime == 1
        self.inv_base = pow(base, prime - 2, prime)
        self.skip_multiplier = 1
    
    def append(self, value):
        self.hash_value = (self.hash_value * self.base + value) % self.prime
        self.skip_multiplier = (self.skip_multiplier * self.base) % self.prime
    
    def skip(self, value):
        self.skip_multiplier = (self.skip_multiplier * self.inv_base) % self.prime
        self.hash_value = (self.hash_value + self.prime - (value * self.skip_multiplier) % self.prime) % self.prime
```
from collections import deque

class RollingHash(AmnesiacRollingHash):
    def __init__(self, *args):
        AmnesiacRollingHash.__init__(self, *args)
        self.data = deque()

    def append(self, value):
        AmnesiacRollingHash.append(self, value)
        self.data.append(value)

    def skip(self):
        AmnesiacRollingHash.skip(self, self.data.popleft())
```