ProgTeam Spring Week 4

Strings I: Hashing

Strings

- List of characters, usually a-z, A-Z, 0-9
  - Different from normal array as “alphabet” is fairly small
  - Allows for different approaches to problems than just a normal array

Hashing

- Pro: “Jackknife” for string problems; probably 90% of problems can be solved by throwing hashing at it
- Con: Slower than many other algorithms; has large constant factor
  - Can require some manipulation to figure out which strings you’re comparing

Hashing

- “Unique” integer returned for a string
- Common hash: for primes p and m ($p < m$)
  - $(s[0] * p^0 + s[1] * p^1 + \ldots + s[n] * p^n) \bmod m$
  - Important!- $p >$ size of alphabet (otherwise we have collisions)
    - With large enough p, doesn’t require small alphabet

Hashing

- With m = $10^9 + 7$, odds of collision are ~= 10^{-9}
- With 100,000 strings, odds of *any* two colliding are 99.95%
  - (see also: Birthday paradox)
- Solution: hash with two different powers
  - Odds of collision are ~= 10^{-18}
  - With 100,000 strings, about 1 in 100 million
- Example: $p_1 = 131$, $p_2 = 499$
  - (Both greater than char max; fine for strings)

Rolling Hash

- (s[0] * p^0 + s[1] * p^1 + … + s[n] * p^n) mod m
- Notice:
  - hash("abcba") = (hash("bcba") - 'a') / p
  - With a prime 'm', we can calculate the "modulo inverse" p^{-1}
- Now we can get a hash of any substring using prefix sums!
- Hash(L,R) = (Sum[R] - Sum[L - 1]) * modinv(p^L)
- Examples of this on the Github