String Matching: Rabin-Karp algorithm

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

Programming, Data Structures and Algorithms using Python

Week 10

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \dots, 9\}$

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern p is an m-digit number n_p

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number
- Scan `t` and compare the number n_b generated by each block of m letters with the pattern number n_p

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number
- Scan `t` and compare the number n_b generated by each block of m letters with the pattern number n_p

Converting a string to a number

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number
- Scan `t` and compare the number n_b generated by each block of m letters with the pattern number n_p

Converting a string to a number

- Can convert a block `t[i:i+m]` to an integer n_i in one scan

```
num = 0
for j in range(m):
  num = 10*num + t[i+j]
```

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number
- Scan `t` and compare the number n_b generated by each block of m letters with the pattern number n_p

Converting a string to a number

- Can convert a block `t[i:i+m]` to an integer n_i in one scan

```
num = 0
for j in range(m):
  num = 10*num + t[i+j]
```

- Computing n_i from `t[i:i+m]` for each block from scratch will take time $O(nm)$

Reducing string matching to arithmetic

- Suppose $\Sigma = \{0, 1, \ldots, 9\}$
- Any string over Σ can be thought of as a number in base 10
- Pattern `p` is an m-digit number n_p
- Each substring of length m in the text `t` is again an m-digit number
- Scan `t` and compare the number n_b generated by each block of m letters with the pattern number n_p

Converting a string to a number

- Can convert a block `t[i:i+m]` to an integer n_i in one scan

```
num = 0
for j in range(m):
  num = 10*num + t[i+j]
```

- Computing n_i from `t[i:i+m]` for each block from scratch will take time $O(nm)$
- Instead
  - Subtract $10^{m-1} \cdot t[i-1]$ from n_{i-1} — drop leading digit
  - Multiply by 10 and add $t[i+m-1]$ to get n_i

Rabin-Karp algorithm

```
def rabinkarp(t,p):
  poslist = []

  numt,nump = 0,0
  for i in range(len(p)):
    numt = 10*numt + int(t[i])
    nump = 10*nump + int(p[i])

  if numt == nump:
    poslist.append(0)

  for i in range(1,len(t)-len(p)+1):
    numt = numt - int(t[i-1])*(10**(len(p)-1))
    numt = 10*numt + int(t[i+len(p)-1])
    if numt == nump:
      poslist.append(i)
  return(poslist)
```

Rabin-Karp algorithm

```
def rabinkarp(t,p):
  poslist = []

  numt,nump = 0,0
  for i in range(len(p)):
    numt = 10*numt + int(t[i])
    nump = 10*nump + int(p[i])

  if numt == nump:
    poslist.append(0)

  for i in range(1,len(t)-len(p)+1):
    numt = numt - int(t[i-1])*(10**(len(p)-1))
    numt = 10*numt + int(t[i+len(p)-1])
    if numt == nump:
      poslist.append(i)
  return(poslist)
```

- First convert `t[0:m]` to n_0 and `p` to n_p

Rabin-Karp algorithm

```
def rabinkarp(t,p):
  poslist = []

  numt,nump = 0,0
  for i in range(len(p)):
    numt = 10*numt + int(t[i])
    nump = 10*nump + int(p[i])

  if numt == nump:
    poslist.append(0)

  for i in range(1,len(t)-len(p)+1):
    numt = numt - int(t[i-1])*(10**(len(p)-1))
    numt = 10*numt + int(t[i+len(p)-1])
    if numt == nump:
      poslist.append(i)
  return(poslist)
```

- First convert `t[0:m]` to n_0 and `p` to n_p
- In the loop, incrementally convert n_{i-1} to n_i

Rabin-Karp algorithm

```
def rabinkarp(t,p):
  poslist = []

  numt,nump = 0,0
  for i in range(len(p)):
    numt = 10*numt + int(t[i])
    nump = 10*nump + int(p[i])

  if numt == nump:
    poslist.append(0)

  for i in range(1,len(t)-len(p)+1):
    numt = numt - int(t[i-1])*(10**(len(p)-1))
    numt = 10*numt + int(t[i+len(p)-1])
    if numt == nump:
      poslist.append(i)
  return(poslist)
```

- First convert `t[0:m]` to n_0 and `p` to n_p
- In the loop, incrementally convert n_{i-1} to n_i
- Whenever $n_i = n_p$ report a match

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly
- Instead, do all computations and comparisons modulo a suitable prime q

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly
- Instead, do all computations and comparisons modulo a suitable prime q

- `p = 31415`, `t = 2359023141526739921`
- $q = 13$

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly
- Instead, do all computations and comparisons modulo a suitable prime q

- `p = 31415`, `t = 2359023141526739921`
- $q = 13$
- $31415 \bmod 13 = 7$

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly
- Instead, do all computations and comparisons modulo a suitable prime q

- `p = 31415`, `t = 2359023141526739921`
- $q = 13$
- $31415 \bmod 13 = 7$
- $23590 \bmod 13 = 8$
  $35902 \bmod 13 = 9$
  $\ldots$
  $31415 \bmod 13 = 7$
  $\ldots$
  $67399 \bmod 13 = 7$
  $\ldots$

Rabin-Karp algorithm for general alphabets

- For $\Sigma = \{a_1, a_2, \ldots, a_k\}$, treat each letter as a digit in base k
- Naively, repeat the previous algorithm in base k
  - Multiply/divide by k rather than 10
- In practice, for realistic k, the numbers are too large to work with directly
- Instead, do all computations and comparisons modulo a suitable prime q

- `p = 31415`, `t = 2359023141526739921`
- $q = 13$
- $31415 \bmod 13 = 7$
- $23590 \bmod 13 = 8$
  $35902 \bmod 13 = 9$
  $\ldots$
  $31415 \bmod 13 = 7$
  $\ldots$
  $67399 \bmod 13 = 7$
  $\ldots$
- False positives — must scan and validate each block that appears to match

Summary

- Preprocessing time is $O(m)$
  - To convert `t[0:m]`, `p` to numbers

Summary

- Preprocessing time is $O(m)$
  - To convert `t[0:m]`, `p` to numbers
- Worst case for general alphabets is $O(nm)$
  - Every block `t[i:i+m]` may have same remainder modulo q as the pattern `p`
  - Must validate each block explicitly, like brute force

Summary

- Preprocessing time is $O(m)$
  - To convert `t[0:m]`, `p` to numbers
- Worst case for general alphabets is $O(nm)$
  - Every block `t[i:i+m]` may have same remainder modulo q as the pattern `p`
  - Must validate each block explicitly, like brute force
- In practice number of spurious matches will be small

Summary

- Preprocessing time is $O(m)$
  - To convert `t[0:m]`, `p` to numbers
- Worst case for general alphabets is $O(nm)$
  - Every block `t[i:i+m]` may have same remainder modulo q as the pattern `p`
  - Must validate each block explicitly, like brute force
- In practice number of spurious matches will be small
- If $|\Sigma|$ is small enough to not require modulo arithmetic, overall time is $O(n+m)$, or $O(n)$, since $m \ll n$
  - Also if we can choose q carefully to ensure $O(1)$ spurious matches