

String Matching

Thanks to
Prof. Piotr Indyk and Carola Wenk

String Matching

- **Input:** Two strings $T[1..n]$ (text) and $P[1..m]$ (pattern), containing symbols from alphabet Σ
- **Goal:** find all “shifts” $1 \leq s \leq n-m$ such that $T[s+1..s+m]=P$
 - In other words: Finds all **shifts** of a window of length m inside the text T , so the context of the window is identical to the pattern P .
- **Example:**
 - $\Sigma = \{a, b, \dots, z\}$
 - $T[1..18] = \text{“to be or not to be”}$
 - $P[1..2] = \text{“be”}$
 - Shifts: 3, 16

Simple Algorithm

```
for  $s \leftarrow 0$  to  $n-m$ 
  Match  $\leftarrow 1$ 
  for  $j \leftarrow 1$  to  $m$ 
    if  $T[s+j] \neq P[j]$  then
      Match  $\leftarrow 0$ 
  exit loop
if Match=1 then output  $s$ 
```

Results

- Running time of the simple algorithm:
 - Worst-case: $O(nm)$
 - Average-case (random text): $O(n)$
- Is it possible to achieve $O(n)$ for any input ?
 - Knuth-Morris-Pratt’77: deterministic
 - **Karp-Rabin’81: randomized**

Karp-Rabin Algorithm

- A very elegant use of an idea that we have encountered before, namely...
HASHING !
- **Idea:**
 - Hash all substrings $T[1..m]$, $T[2..m+1]$, $T[3..m+2]$, etc.
 - Hash (details later) the pattern $P[1..m]$
 - Report the substrings that hash to the same value as P
- **Problem:** how to hash $n-m$ substrings, each of length m , in $O(n)$ time ?

Implementation

- **Attempt I:**
 - Assume $\Sigma = \{0, 1\}$
 - Think about each $T^s = T[s+1..s+m]$ as a number in binary representation, i.e.,
$$t_s = T[s+1]2^0 + T[s+2]2^1 + \dots + T[s+m]2^{m-1}$$
 - Find a fast way of computing t_{s+1} given t_s
 - Output all s such that t_s is equal to the number p represented by P

The great formula

- How to transform

$$t_s = T[s+1]2^0 + T[s+2]2^1 + T[s+3]2^2 + \dots + T[s+m]2^{m-1}$$
 into

$$t_{s+1} = T[s+2]2^0 + T[s+3]2^1 + T[s+4]2^2 + \dots + T[s+m]2^{m-1} + T[s+m+1]2^{m-1} ?$$
- Three steps:
 - Subtract $T[s+1]2^0$
 - Divide by 2 (i.e., shift the bits by one position)
 - Add $T[s+m+1]2^{m-1}$
- Therefore: $t_{s+1} = (t_s - T[s+1]2^0) / 2 + T[s+m+1]2^{m-1}$

Algorithm

- Can compute t_{s+1} from t_s using 3 arithmetic operations
- Therefore, we can compute all t_0, t_1, \dots, t_{n-m} using $O(n)$ arithmetic operations
- We can compute a number corresponding to P using $O(m)$ arithmetic operations
- Are we done ?

Problem

- To get $O(n)$ time, we would need to perform each arithmetic operation in $O(1)$ time
- However, the arguments are m -bit long (and we have 32/64 bits machine) !
- It is unreasonable to assume that operations on such big numbers can be done in $O(1)$ time
- We need to reduce the number range to something more manageable

Warm-up

- $((x \bmod q) + (y \bmod q)) \bmod q = (x+y) \bmod q$
- $((x \bmod q) (y \bmod q)) \bmod q = (xy) \bmod q$
- $(ax+b \bmod q) = ((a \bmod q) (x \bmod q) + (b \bmod q)) \bmod q$
- Every integer x can be uniquely represented as $x = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$ where
 - p_i is a prime, and
 - e_i is an integer
 - $k \leq \log_2 x$ since each $p_i \geq 2$

Hashing

- We will instead compute

$$t'_s = T[s+1]2^0 + T[s+2]2^1 + \dots + T[s+m]2^{m-1} \bmod q$$
 where q is an "appropriate" prime number
- One can still compute t'_{s+1} from t'_s :

$$t'_{s+1} = (t'_s - T[s+1]2^0) * 2^{-1} + T[s+m+1]2^{m-1} \bmod q$$
- If q is not large, i.e., has $O(\log n)$ bits, we can compute all t'_s (and p') in $O(n)$ time

Problem

- Unfortunately, we can have **false positives**, i.e., $T \neq P$ but $t'_s = p'$
 - (to discover a single false positive, we spend $O(m)$ time)
- Need to use a random q
- We will show that the probability of a false positive is small \rightarrow randomized algorithm

False positives

- Consider any $t_s \neq p$. We know that both numbers are in the range $\{0 \dots 2^m - 1\}$
- How many primes q are there such that $t_s \bmod q = p \bmod q \equiv (t_s - p) \equiv 0 \pmod q$?
- Such prime has to divide $x = (t_s - p) \leq 2^m$
- Represent $x = p_1^{e_1} p_2^{e_2} \dots p_k^{e_k}$, p_i prime, $e_i \geq 1$
- Since $2 \leq p_i$, we have $2^k \leq x \leq 2^m \rightarrow k \leq m$
- There are $\leq m$ primes dividing x

Algorithm

- Let Π be a set of $2nm$ primes, each having $O(\log n)$ bits (not generated explicitly)
- Choose q uniformly at random from Π
- Compute t'_0, t'_1, \dots , and p'
- For each shift s , the probability that $t'_s = p'$ while $T^s \neq P$ is at most $\log t_s / |\Pi| = m/2nm = 1/2n$
- If $t'_s = p'$, we check if $t_s = p$ by checking each char. Takes time $O(m)$. Altogether $O(n)$
- The probability of *any* false positive is at most $(n-m)/2n \leq 1/2$

Geometric Hashing and other problems of shape matching

- This algorithm is an example of general idea:
 - Given a library of (many) shapes T_1, T_2, \dots, T_r . Preprocess such that given a query pattern P , find the most similar shape.
 - Checking for **given** T_i if it is similar to P is expensive.
 - **Idea:** Using hashing for **filtering** the shapes that need to be checked:
 - Compute hash values $h(T_1), \dots, h(T_r)$, and $h(P)$, and check if T_i matches P only if $h(P) = h(T_i)$.