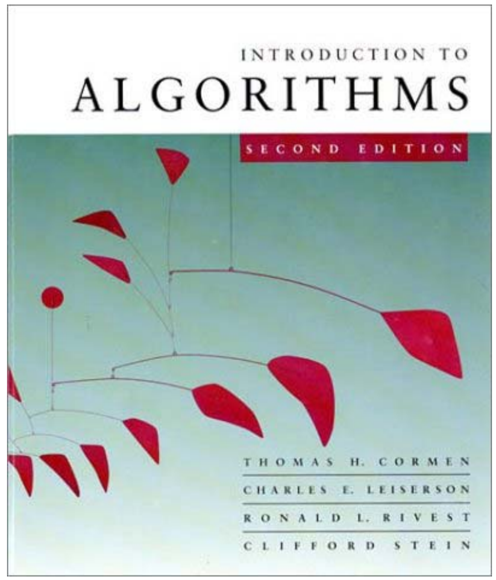
# *Introduction to Algorithms*

## 6.046J/18.401J




**Prof. Piotr Indyk**

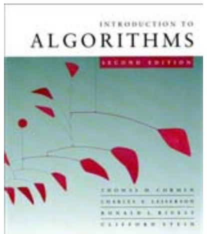
# String Matching

- Input: Two strings T[1…n] and P[1…m], containing symbols from alphabet Σ.

  E.g. :
  - Σ={a,b,…,z}
  - T[1…18]="to be or not to be"
  - P[1..2]="be"

- Goal: find all "shifts" $0 \le s \le n-m$ such that T[s+1…s+m]=P

  E.g. 3, 16




# Simple Algorithm

**for** $s \leftarrow 0$ **to** $n\text{-}m$

    $Match \leftarrow 1$

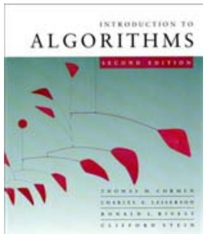
    **for** $j \leftarrow 1$ **to** $m$

        **if** $\mathrm{T}[s+j] \neq \mathrm{P}[j]$ **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): $O(n)$
    - $T_s$ = time spent on checking shift $s$
    - $E[T_s] \leq 2$
    - $E\,[\sum_s T_s] = \sum_s E[T_s] = O(n)$

# Worst-case

- Is it possible to achieve O(n) for any input ?
  – Knuth-Morris-Pratt'77: deterministic
  – Karp-Rabin'81: randomized

# Karp-Rabin Algorithm

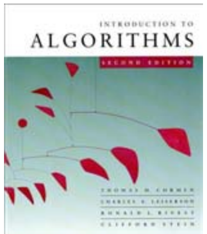
- A very elegant use of an idea that we have encountered before, namely…

  HASHING !

- Idea:
  - Hash all substrings

    T[1…m], T[2…m+1], …, T[m-n+1…n]
  - Hash 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 ?




# Attempt 0

- In Lecture 7, we have seen

$$h_a(x)=\sum_i a_i x_i \bmod q$$

where $a=(a_1,\ldots,a_r)$ , $x=(x_1,\ldots,x_r)$

- To implement it, we would need to compute

$$h_a(\,T[s\ldots s+m-1]\,)=\sum_{i=0,\ldots,m-1} a_i\, T[s+i] \bmod q$$

for $s=0\ldots n-m$

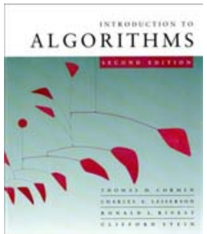
- How to compute it in $O(n)$ time ?
- A big open problem!

# Attempt 1

- Assume $\Sigma=\{0,1\}$
- Think about each $T^s=T[s+1\ldots s+m]$ as a number in binary representation, i.e.,

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

- 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

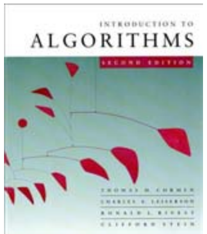
- How to transform

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

  into

  $$t_{s+1} = T[s+2]2^{m-1} + T[s+3]2^{m-2} + \ldots + T[s+m+1]2^0 \text{ ?}$$

- Three steps:
  - Subtract $T[s+1]2^{m-1}$
  - Multiply by $2$ (i.e., shift the bits by one position)
  - Add $T[s+m+1]2^0$
- Therefore: $t_{s+1} = (t_s - T[s+1]2^{m-1})*2 + T[s+m+1]2^0$




# Algorithm

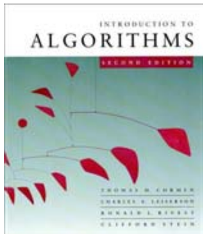
$$t_{s+1} = (t_s - T[s+1]2^{m-1})*2 + T[s+m+1]2^0$$

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

© Piotr Indyk

# 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 !
- If m large, 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 managable




# Attempt 2: Hashing

- We will instead compute

  $t'_s = T[s+1]2^{m-1} + T[s+2]2^{m-2} + \ldots + T[s+m]2^0 \bmod q$

  where $q$ is an "appropriate" prime number
- One can still compute $t'_{s+1}$ from $t'_s$ :

  $t'_{s+1} = (t'_s - T[s+1]2^{m-1}) * 2 + T[s+m+1]2^0 \bmod q$
- 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., $T^s \neq P$ but $t_s \bmod q = p \bmod q$
- Need to use a random $q$
- We will show that the probability of a false positive is small → randomized algorithm

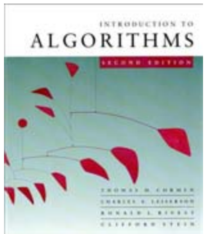
# False positives

- Consider any $t_s \neq p$. We know that both numbers are in the range $\{0 \ldots 2^m-1\}$
- How many primes $q$ are there such that

$$t_s \bmod q = p \bmod q \quad \equiv \quad (t_s - p) = 0 \mod q \ ?$$

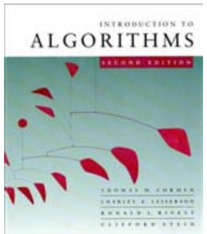
- Such prime has to divide $x = (t_s - p) \leq 2^m$
- Represent $x = p_1^{e1} p_2^{e2} \ldots p_k^{ek}$, $p_i$ prime, $e_i \geq 1$

  What is the largest possible value of $k$ ?
  - Since $2 \leq p_i$, we have $x \geq 2^k$
  - But $x \leq 2^m$
  - $k \leq m$
- There are $\leq m$ primes dividing $x$




# Algorithm

- Algorithm:
  - Let $\prod$ be a set of $2nm$ primes, each having $O(\log n)$ bits
  - Choose $q$ uniformly at random from $\prod$
  - Compute $t_0 \bmod q$, $t_1 \bmod q$, …., and $p \bmod q$
  - Report $s$ such that $t_s \bmod q = p \bmod q$
- Analysis:
  - For each $s$, the probability that $T^s \neq P$ but
    $$t_s \bmod q = p \bmod q$$
    is at most $m/2nm = 1/2n$
  - The probability of *any* false positive is at most $(n-m)/2n \leq 1/2$




# “Details”

- How do we know that such $\prod$ exists ?

  (That is, a set of $2nm$ primes, each having $O(\log n)$ bits)

- How do we choose a random prime from $\prod$ in $O(n)$ time ?

# Prime density

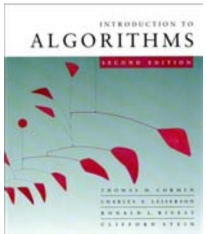
- Primes are "dense". I.e., if PRIMES(N) is the set of primes smaller than N, then asymptotically

$$|\mathrm{PRIMES}(N)|/N \sim 1/\ln N$$

- If N large enough, then

$$|\mathrm{PRIMES}(N)| \geq N/(2\ln N)$$

- Proof: Trust me.




# Prime density continued

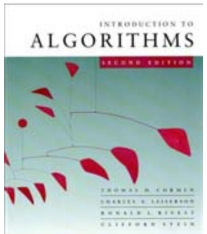
- Set $N = C\ mn \ln(mn)$
- There exists $C = O(1)$ such that

$$N/(2 \ln N) \geq 2mn$$

(Note: for such $N$ we have $\text{PRIMES}(N) \geq 2mn$ )
- Proof:

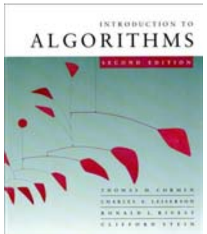
$$C\ mn \ln(mn) / [2 \ln(C\ mn \ln(mn))\ ]$$
$$\geq C\ mn \ln(mn) / [2 \ln(C\ (mn)^2)\ ]$$
$$= C\ mn \ln(mn) / 4[\ \ln(C) + \ln(mn)]$$

- All elements of $\text{PRIMES}(N)$ are $\log N = O(\log n)$ bits long



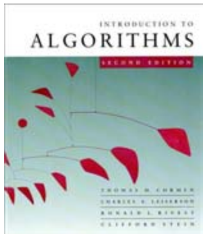

# Prime selection

- Still need to find a random element of PRIMES(N)
- Solution:
  - Choose a random element from {1 … N}
  - Check if it is prime
  - If not, repeat




# Prime selection analysis

- A random element $q$ from $\{1\ldots N\}$ is prime with probability $\sim 1/\ln N$
- We can check if $q$ is prime in time polynomial in $\log N$ :
  - Randomized: Rabin, Solovay-Strassen in 1976
  - Deterministic: Agrawal et al in 2002
- Therefore, we can generate random prime $q$ in $o(n)$ time




# Final Algorithm

- Set $N = C\ mn \ln(mn)$
- Repeat
  - Choose $q$ uniformly at random from $\{1 \ldots N\}$
- Until $q$ is prime
- Compute $t_0 \bmod q$, $t_1 \bmod q$, …., and $p \bmod q$
- Report $s$ such that $t_s \bmod q = p \bmod q$

Optional Final m Steps: Double check match for s is correct