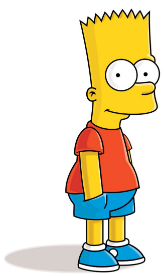
String Matching

String Matching

Problem: Given an alphabet Σ, a *text* $T \in \Sigma^*$ and a *pattern* $P \in \Sigma^*$, find some occurrence/all occurrences of P in T.



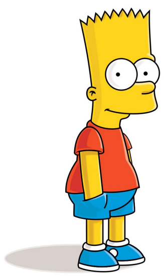
$$\Sigma = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}, \textvisiblespace\}$$

$$T = \texttt{Bart\textvisiblespace played\textvisiblespace darts\textvisiblespace at\textvisiblespace the\textvisiblespace party}$$

$$P = \texttt{art}$$

String Matching

Problem: Given an alphabet Σ, a *text* $T \in \Sigma^*$ and a *pattern* $P \in \Sigma^*$, find some occurrence/all occurrences of P in T.



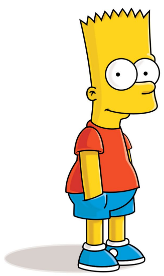
$$\Sigma = \{\texttt{A}, \texttt{B}, \ldots, \texttt{Z}, \texttt{a}, \texttt{b}, \ldots, \texttt{z}, \textvisiblespace\}$$

$$T = \texttt{B}\underline{\texttt{art}}\texttt{\textvisiblespace played\textvisiblespace d}\underline{\texttt{art}}\texttt{s\textvisiblespace at\textvisiblespace the\textvisiblespace p}\underline{\texttt{art}}\texttt{y}$$

$$P = \texttt{art}$$

String Matching

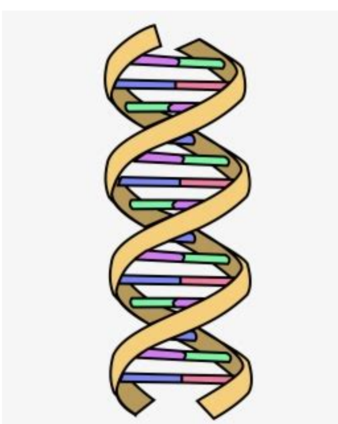
Problem: Given an alphabet Σ, a *text* $T \in \Sigma^*$ and a *pattern* $P \in \Sigma^*$, find some occurrence/all occurrences of P in T.



$$\Sigma = \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}, \textvisiblespace\}$$

$$T = \mathtt{B\underline{art}\textvisiblespace played\textvisiblespace d\underline{art}s\textvisiblespace at\textvisiblespace the\textvisiblespace p\underline{art}y}$$

$$P = \mathtt{art}$$



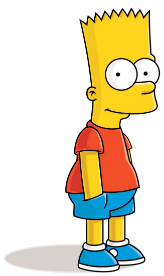
$$\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$$

$$T = \mathtt{ACGTGCTTGCAGTGTGCATTACCTGAGTGC}\ldots$$

$$P = \mathtt{GTG}$$

String Matching

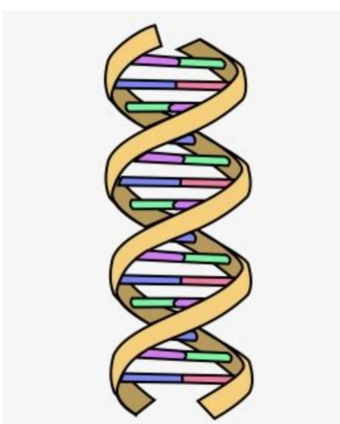
Problem: Given an alphabet Σ, a *text* $T \in \Sigma^*$ and a *pattern* $P \in \Sigma^*$, find some occurrence/all occurrences of P in T.



$$\Sigma = \{\mathtt{A}, \mathtt{B}, \ldots, \mathtt{Z}, \mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}, \textvisiblespace\}$$

$$T = \texttt{Bart␣played␣darts␣at␣the␣party}$$

$$P = \texttt{art}$$



$$\Sigma = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$$

$$T = \texttt{ACGTGCTTGCAGTGTGCATTACCTGAGTGC}\ldots$$

$$P = \texttt{GTG}$$

String Matching

One-shot:

- Both the text and the pattern are **part of the input**
- Algorithm design problem

String Matching

One-shot:

- Both the text and the pattern are **part of the input**
- Algorithm design problem

Repeated:

- The text is **static** and known beforehand
  (can be preprocessed)
- Patterns are revealed on-demand
- We want to answer each *query* as **quickly** as possible
- Data structure design problem

String Matching

One-shot:

- Both the text and the pattern are **part of the input**
- Algorithm design problem

Repeated:

- The text is **static** and known beforehand
  (can be preprocessed)
- Patterns are revealed on-demand
- We want to answer each *query* as **quickly** as possible
- Data structure design problem

String Matching

Input:

- A text $T = t_1 t_2 \ldots t_n \in \Sigma^*$, i.e., a word over an alphabet Σ.
- A pattern $P = p_1 p_2 \ldots p_m \in \Sigma^*$.
- For simplicity, assume $m = |P| < |T| = n$ and $|\Sigma| \leq n$.

Output:

- Does P occur in T with some shift s?
- P occurs in T with shift $s = 0, \ldots, n - m$ if $p_1 p_2 \ldots p_m = t_{s+1} t_{s+2} \ldots t_{s+m}$.

Model: Word-RAM with word size $w = \Theta(\log n)$.

Example

$\Sigma = \{a, b, c\}, n = 15, m = 5$

$T =$ `abbacbaabcbaaac`

$P =$ `acbaa`

`abbacbaabcbaaac`

Example

$\Sigma = \{a, b, c\}, n = 15, m = 5$

$T =$ `abbacbaabcbaaac`

$P =$ `acbaa`

$$\texttt{abb}\boxed{\texttt{acbaa}}\texttt{bcbaaac}$$

$$\underbrace{\quad\quad}_{s=3}$$

Output: yes

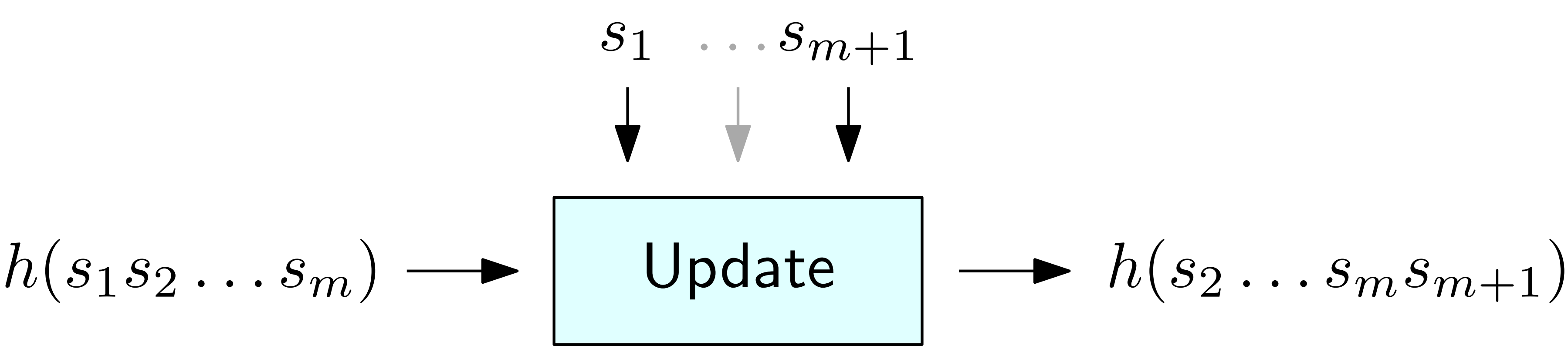
A Naive Algorithm

- For each $s = 0, 1, \ldots, n - m$: $O(n-m)$
  - If P occurs in T with shift s $O(m)$
    - return true
- return false

Time: $\Theta(m \cdot (n - m))$

Rolling Hash Functions

- A function $h : \Sigma^m \to \{0, \ldots, q\}$
- If $h(s_1 s_2 \ldots s_m)$ is known, then $h(s_2 \ldots s_m s_{m+1})$ can be computed *quickly*.



$h(abbac)$

$\boxed{\texttt{abbac}}\texttt{baabcbaaac}$

Rolling Hash Functions

- A function $h : \Sigma^m \to \{0, \ldots, q\}$
- If $h(s_1 s_2 \ldots s_m)$ is known, then $h(s_2 \ldots s_m s_{m+1})$ can be computed *quickly*.

$$s_1 \quad \ldots \quad s_{m+1}$$

$$h(s_1 s_2 \ldots s_m) \longrightarrow \boxed{\text{Update}} \longrightarrow h(s_2 \ldots s_m s_{m+1})$$

$$h(bbacb)$$

$$\texttt{a}\boxed{\texttt{bbacb}}\texttt{aabcbaaac}$$

Rolling Hash Functions

- A function $h : \Sigma^m \to \{0, \ldots, q\}$
- If $h(s_1 s_2 \ldots s_m)$ is known, then $h(s_2 \ldots s_m s_{m+1})$ can be computed *quickly*.

$$s_1 \quad \ldots s_{m+1}$$

$$h(s_1 s_2 \ldots s_m) \longrightarrow \boxed{\text{Update}} \longrightarrow h(s_2 \ldots s_m s_{m+1})$$

$h(bacba)$

ab$\boxed{\texttt{bacba}}$abcbaaac

A First Rolling “Hash” Function

- Interpret the elements of Σ as distinct digits in base $|\Sigma|$

A First Rolling "Hash" Function

- Interpret the elements of Σ as distinct digits in base $|\Sigma|$

- Interpret $s_1 \ldots s_m \in \Sigma^m$ as a number $\eta(s_1 \ldots s_m)$ in base $|\Sigma|$

$$\eta(s_1 \ldots s_m) = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i}$$

A First Rolling "Hash" Function

- Interpret the elements of Σ as distinct digits in base $|\Sigma|$

- Interpret $s_1 \ldots s_m \in \Sigma^m$ as a number $\eta(s_1 \ldots s_m)$ in base $|\Sigma|$

$$\eta(s_1 \ldots s_m) = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i}$$

$\Sigma = \{a, b, c\}$

$\mathtt{a} = 0,\ \mathtt{b} = 1,\ \mathtt{c} = 2$

. . . `bbacb` . . .

A First Rolling "Hash" Function

- Interpret the elements of Σ as distinct digits in base $|\Sigma|$

- Interpret $s_1 \dots s_m \in \Sigma^m$ as a number $\eta(s_1 \dots s_m)$ in base $|\Sigma|$

$$\eta(s_1 \dots s_m) = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i}$$

$\Sigma = \{a, b, c\}$

$\mathtt{a} = 0,\ \mathtt{b} = 1,\ \mathtt{c} = 2$

. . . `bbacb` . . .

$$\begin{array}{ccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array}$$

$$(\ 1\ 1\ 0\ 2\ 1\ )_3$$

A First Rolling "Hash" Function

- Interpret the elements of Σ as distinct digits in base $|\Sigma|$

- Interpret $s_1 \ldots s_m \in \Sigma^m$ as a number $\eta(s_1 \ldots s_m)$ in base $|\Sigma|$

$$\eta(s_1 \ldots s_m) = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i}$$

$\Sigma = \{a, b, c\}$

$\texttt{a} = 0,\ \texttt{b} = 1,\ \texttt{c} = 2$

. . . bbacb . . .

$(\ 1\ 1\ 0\ 2\ 1\ )_3$

$$\eta(\texttt{bbacb}) = 1 \cdot 3^4 + 1 \cdot 3^3 + 0 \cdot 3^2 + 2 \cdot 3^1 + 1 \cdot 3^0 = 115$$

Updating η

$\sigma = |\Sigma|^{m-1}$

$\eta(s_2 \ldots s_{m+1}) = \quad \eta(s_1 \ldots s_m)$

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \ldots\ \texttt{b} & \texttt{b} & \texttt{a} & \texttt{c} & \texttt{b} & \texttt{c}\ \ldots \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ (\ 1 & 1 & 0 & 2 & 1 &)_3 \end{array}$$

$\eta(\texttt{bacbc}) = \quad \eta(\texttt{bbacb})$

Updating η

$\sigma = |\Sigma|^{m-1}$

$\eta(s_2 \ldots s_{m+1}) = \quad \eta(s_1 \ldots s_m) - s_1 \cdot \sigma$

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \\ \texttt{b} & \texttt{b} & \texttt{a} & \texttt{c} & \texttt{b} & \texttt{c} \\ & \downarrow & \downarrow & \downarrow & \downarrow & \\ (& 1 & 0 & 2 & 1 &)_3 \end{array}$$

$\ldots\texttt{bbacbc}\ldots$

$\eta(\texttt{bacbc}) = \quad \eta(\texttt{bbacb}) \; -1 \cdot 3^4$

Updating η

$\sigma = |\Sigma|^{m-1}$

$\eta(s_2 \ldots s_{m+1}) = \ (\eta(s_1 \ldots s_m) - s_1 \cdot \sigma \ ) |\Sigma| + s_{m+1}$

$$
\begin{array}{c}
s_1\ s_2\ s_3\ s_4\ s_5\ s_6 \\
\ldots \texttt{bbacbc} \ldots \\
\downarrow\ \downarrow\ \downarrow\ \downarrow\ \downarrow \\
(\ 1\ \ 0\ \ 2\ \ 1\ \ 2\ )_3
\end{array}
$$

$\eta(\texttt{bacbc}) = \ (\ \ \eta(\texttt{bbacb}) \ \ -1 \cdot 3^4 \ \ ) \cdot 3 + 2$

Updating η

$$\sigma = |\Sigma|^{m-1}$$

$$\eta(s_2 \ldots s_{m+1}) = \left(\eta(s_1 \ldots s_m) - s_1 \cdot \sigma\right)|\Sigma| + s_{m+1}$$

$$\begin{array}{cccccc} s_1 & s_2 & s_3 & s_4 & s_5 & s_6 \end{array}$$

$$\ldots \texttt{b}\,\texttt{bacbc} \ldots$$

$$\begin{array}{ccccc} \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \end{array}$$

$$(\ 1\ 0\ 2\ 1\ 2\ )_3$$

$$\eta(\texttt{bacbc}) = \left(\ \eta(\texttt{bbacb})\ -1 \cdot 3^4\ \right) \cdot 3 + 2 \ = 104$$

Updating η

$$\sigma = |\Sigma|^{m-1}$$

$$\eta(s_2 \ldots s_{m+1}) = \left(\eta(s_1 \ldots s_m) - s_1 \cdot \sigma\right)|\Sigma| + s_{m+1}$$

$$\begin{array}{ccccccccc} & s_1 & s_2 & s_3 & s_4 & s_5 & s_6 & \\ \ldots & \mathtt{b} & \mathtt{b} & \mathtt{a} & \mathtt{c} & \mathtt{b} & \mathtt{c} & \ldots \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ & (& 1 & 0 & 2 & 1 & 2 &)_3 \end{array}$$

$$\eta(\mathtt{bacbc}) = \left(\eta(\mathtt{bbacb}) - 1 \cdot 3^4\right) \cdot 3 + 2 = 104$$

Notice that $\eta(\cdot)$ is **injective**!

A String Matching Algorithm using η

- $h^* \leftarrow \eta(p_1 \ldots p_m)$
- $h_0 \leftarrow \eta(t_1 \ldots t_m)$
- $\sigma = |\Sigma|^{m-1}$
- If $h_0 = h^*$: return true
- For $s = 1, \ldots, n-m$:
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma)|\Sigma| + t_{s+m}$
  - If $h_s = h^*$: return true
- return false

A String Matching Algorithm using η

- $h^* \leftarrow \eta(p_1 \dots p_m)$
- $h_0 \leftarrow \eta(t_1 \dots t_m)$
- $\sigma = |\Sigma|^{m-1}$
- If $h_0 = h^*$: return true
- For $s = 1, \dots, n-m$:
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma)|\Sigma| + t_{s+m}$
  - If $h_s = h^*$: return true
- return false
- **Problem**: working with integers up to Σ^m requires $\Omega(m \log |\Sigma|)$ bits, i.e., $\Omega(m \frac{\log |\Sigma|}{\log n})$ words.

A String Matching Algorithm using η

- $h^* \leftarrow \eta(p_1 \ldots p_m)$
- $h_0 \leftarrow \eta(t_1 \ldots t_m)$ $\widetilde{\Omega}(m)$
- $\sigma = |\Sigma|^{m-1}$
- If $h_0 = h^*$: return true
- For $s = 1, \ldots, n-m$: $\Omega(n-m)$
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma)|\Sigma| + t_{s+m}$ $\widetilde{\Omega}(m)$
  - If $h_s = h^*$: return true
- return false
- **Problem**: working with integers up to Σ^m requires $\Omega(m \log |\Sigma|)$ bits, i.e., $\Omega(m\frac{\log |\Sigma|}{\log n})$ words.

A Better Rolling Hash Function

Let q be a random prime number in $\{2, \ldots, M \log M\}$, for some $M \leq n^{O(1)}$

Define:

$$h(s_1 \ldots s_m) = \eta(s_1 \ldots s_m) \bmod q = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i} \pmod q$$

A Better Rolling Hash Function

Let q be a random prime number in $\{2, \ldots, M \log M\}$, for some $M \leq n^{O(1)}$

Define:

$$h(s_1 \ldots s_m) = \eta(s_1 \ldots s_m) \bmod q = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i} \pmod q$$

$h(s_1 \ldots s_m)$ can be stored in $\log(M \log M) = O(1)$ words.

A Better Rolling Hash Function

Let q be a random prime number in $\{2, \ldots, M \log M\}$, for some $M \leq n^{O(1)}$

Define:

$$h(s_1 \ldots s_m) = \eta(s_1 \ldots s_m) \bmod q = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i} \pmod q$$

$h(s_1 \ldots s_m)$ can be stored in $\log(M \log M) = O(1)$ words.

Computing $h(s_1 \ldots s_m)$

- $h(s_1) = s_1 \mod q$
- $h(s_1 s_2) = (h(s_1)|\Sigma| + s_2) \mod q$

  $\ldots$
- $h(s_1 s_2 \ldots s_m) = (h(s_1 s_2 \ldots s_{m-1})|\Sigma| + s_m) \mod q$

A Better Rolling Hash Function

Let q be a random prime number in $\{2, \ldots, M \log M\}$, for some $M \leq n^{O(1)}$

Define:

$$h(s_1 \ldots s_m) = \eta(s_1 \ldots s_m) \bmod q = \sum_{i=1}^{m} s_i \cdot |\Sigma|^{m-i} \pmod q$$

$h(s_1 \ldots s_m)$ can be stored in $\log(M \log M) = O(1)$ words.

Computing $h(s_1 \ldots s_m)$

- $h(s_1) = s_1 \mod q$
- $h(s_1 s_2) = (h(s_1)|\Sigma| + s_2) \mod q$

  $\ldots$
- $h(s_1 s_2 \ldots s_m) = (h(s_1 s_2 \ldots s_{m-1})|\Sigma| + s_m) \mod q$

Time: $O(m)$

A Better Rolling Hash Function

Updating $h(s_1 \ldots s_m)$

$$\eta(s_2 \ldots s_{m+1}) = \left(\eta(s_1 \ldots s_m) - s_1 \cdot \sigma\right) |\Sigma| + s_{m+1}$$

where $\sigma = |\Sigma|^{m-1}$

$$\Downarrow$$

$$h(s_2 \ldots s_{m+1}) = \left(h(s_1 \ldots s_m) - s_1 \cdot \sigma'\right) |\Sigma| + s_{m+1}$$

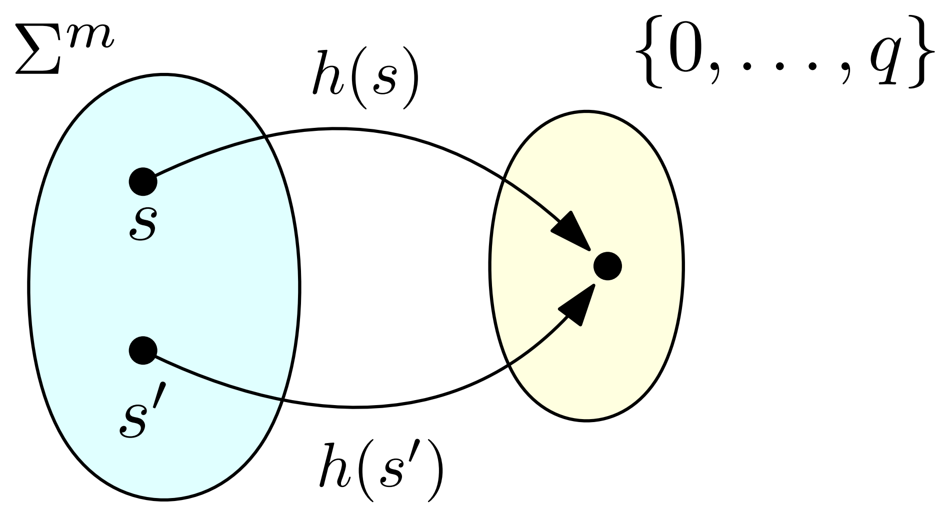
where $\sigma' = \sigma \bmod q = |\Sigma|^{m-1} \bmod q$ and all operations are performed modulo q.

Time: $O(1)$

Hash Collisions

Let $s, s' \in \Sigma^m$

- If $s = s'$ then $h(s) = h(s')$
- If $s \neq s'$ then it is not guaranteed that $h(s) \neq h(s')$
- If $s \neq s'$ and $h(s) = h(s')$ then we have a *hash collision*.



Collision Probability

Consider $s \neq s'$. If $h(s) = h(s')$ then $\eta(s) \equiv \eta(s') \pmod q$

$$\implies \eta(s) - \eta(s') \equiv 0 \pmod q$$

$\implies$ $\eta(s) - \eta(s')$ is a multiple of q

Collision Probability

Consider $s \neq s'$. If $h(s) = h(s')$ then $\eta(s) \equiv \eta(s') \pmod q$

$$\implies \eta(s) - \eta(s') \equiv 0 \pmod q$$

$$\implies \eta(s) - \eta(s') \text{ is a multiple of } q$$

Let $p_1 \cdot p_2 \cdot \cdots \cdot p_k$ be a factorization of $\Delta = |\eta(s) - \eta(s')|$

Since each p_i is at least 2, $k \leq \log_2 \Delta \leq \log |\Sigma|^m = m \log |\Sigma|$

Collision Probability

Consider $s \neq s'$. If $h(s) = h(s')$ then $\eta(s) \equiv \eta(s') \pmod q$

$$\implies \eta(s) - \eta(s') \equiv 0 \pmod q$$

$$\implies \eta(s) - \eta(s') \text{ is a multiple of } q$$

Let $p_1 \cdot p_2 \cdot \cdots \cdot p_k$ be a factorization of $\Delta = |\eta(s) - \eta(s')|$

Since each p_i is at least 2, $k \leq \log_2 \Delta \leq \log |\Sigma|^m = m \log |\Sigma|$

Let $\pi(x)$ be the number of prime numbers in $\{1, \ldots, x\}$.

Prime number theorem: $\pi(x) = \Theta(\frac{x}{\log x})$

Collision Probability

Consider $s \neq s'$. If $h(s) = h(s')$ then $\eta(s) \equiv \eta(s') \pmod q$

$$\implies \eta(s) - \eta(s') \equiv 0 \pmod q$$

$$\implies \eta(s) - \eta(s') \text{ is a multiple of } q$$

Let $p_1 \cdot p_2 \cdot \dots \cdot p_k$ be a factorization of $\Delta = |\eta(s) - \eta(s')|$

Since each p_i is at least 2, $k \leq \log_2 \Delta \leq \log |\Sigma|^m = m \log |\Sigma|$

Let $\pi(x)$ be the number of prime numbers in $\{1, \dots, x\}$.

Prime number theorem: $\pi(x) = \Theta(\frac{x}{\log x})$

$$P(q \in \{p_1, \dots, p_k\}) \leq \frac{k}{\pi(M \log M)} = O(\frac{m \log |\Sigma|}{M})$$

Collision Probability

Consider $s \neq s'$. If $h(s) = h(s')$ then $\eta(s) \equiv \eta(s') \pmod{q}$

$$\implies \eta(s) - \eta(s') \equiv 0 \pmod{q}$$

$$\implies \eta(s) - \eta(s') \text{ is a multiple of } q$$

Let $p_1 \cdot p_2 \cdot \dots \cdot p_k$ be a factorization of $\Delta = |\eta(s) - \eta(s')|$

Since each p_i is at least 2, $k \leq \log_2 \Delta \leq \log |\Sigma|^m = m \log |\Sigma|$

Let $\pi(x)$ be the number of prime numbers in $\{1, \dots, x\}$.

Prime number theorem: $\pi(x) = \Theta(\frac{x}{\log x})$

$$P(q \in \{p_1, \dots, p_k\}) \leq \frac{k}{\pi(M \log M)} = O(\frac{m \log |\Sigma|}{M}) = O(\frac{1}{m})$$

Pick $M = m^2 \log |\Sigma|$

The Rabin-Karp Algorithm

- $q \leftarrow$ A prime number chosen u.a.r. in $\{2, \ldots, M \log M\}$.
- $\sigma' \leftarrow |\Sigma|^{m-1} \mod q$
- $h^* \leftarrow h(p_1 \ldots p_m)$
- $h_0 \leftarrow h(t_1 \ldots t_m)$
- If $h_0 = h^*$ and $p_1 p_2 \ldots p_m = t_1 t_2 \ldots t_m$: return true
- For $s = 1, \ldots, n - m$:
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma')|\Sigma| + t_{s+m} \pmod q$
  - If $h_s = h^*$:
    - If $p_1 p_2 \ldots p_m = t_{s+1} t_{s+2} \ldots t_{s+m}$: return true
- return false

The Rabin-Karp Algorithm

- $q \leftarrow$ A prime number chosen u.a.r. in $\{2, \ldots, M \log M\}$.
- $\sigma' \leftarrow |\Sigma|^{m-1} \mod q$
- $h^* \leftarrow h(p_1 \ldots p_m)$
- $h_0 \leftarrow h(t_1 \ldots t_m)$
- If $h_0 = h^*$ and $p_1 p_2 \ldots p_m = t_1 t_2 \ldots t_m$: return true
- For $s = 1, \ldots, n-m$:
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma')|\Sigma| + t_{s+m} \pmod q$
  - If $h_s = h^*$:
    - If $p_1 p_2 \ldots p_m = t_{s+1} t_{s+2} \ldots t_{s+m}$: return true

Handle hash collisions

- return false

The Rabin-Karp Algorithm

- $q \leftarrow$ A prime number chosen u.a.r. in $\{2, \ldots, M \log M\}$.
- $\sigma' \leftarrow |\Sigma|^{m-1} \mod q$
- $h^* \leftarrow h(p_1 \ldots p_m)$
- $h_0 \leftarrow h(t_1 \ldots t_m)$
- If $h_0 = h^*$ and $p_1 p_2 \ldots p_m = t_1 t_2 \ldots t_m$: return true
- For $s = 1, \ldots, n - m$:
  - $h_s \leftarrow (h_{s-1} - t_{s-1} \cdot \sigma')|\Sigma| + t_{s+m} \pmod q$
  - If $h_s = h^*$:
    - If $p_1 p_2 \ldots p_m = t_{s+1} t_{s+2} \ldots t_{s+m}$: return true

  Handle hash collisions
- return false

Worst case time: $O(m \cdot (n - m))$.

Expected Time Complexity

$$\left.\begin{array}{l}\bullet\ \text{Computing } h^* \text{ takes time } O(m) \\ \bullet\ \text{Iterating over } T \text{ takes time } O(n-m)\end{array}\right\} O(n)$$

Expected Time Complexity

$$\left.\begin{array}{l}\bullet\ \text{Computing } h^* \text{ takes time } O(m)\\[1em] \bullet\ \text{Iterating over } T \text{ takes time } O(n-m)\end{array}\right\} O(n)$$

- If P appears in T, verifying the match takes time $O(m)$

Expected Time Complexity

$$\left.\begin{array}{l}\bullet\ \text{Computing } h^* \text{ takes time } O(m)\\[1em] \bullet\ \text{Iterating over } T \text{ takes time } O(n-m)\end{array}\right\} O(n)$$

- If P appears in T, verifying the match takes time $O(m)$

- The expected number of hash collisions is $\leq (n-m) \cdot O(\frac{1}{m})$
  - Each match collision needs to be verified in time $O(m)$

Expected Time Complexity

- Computing h^* takes time $O(m)$
- Iterating over T takes time $O(n-m)$

$$\left.\begin{array}{l} \\ \\ \end{array}\right\} O(n)$$

- If P appears in T, verifying the match takes time $O(m)$
- The expected number of hash collisions is $\leq (n-m) \cdot O(\frac{1}{m})$
  - Each match collision needs to be verified in time $O(m)$

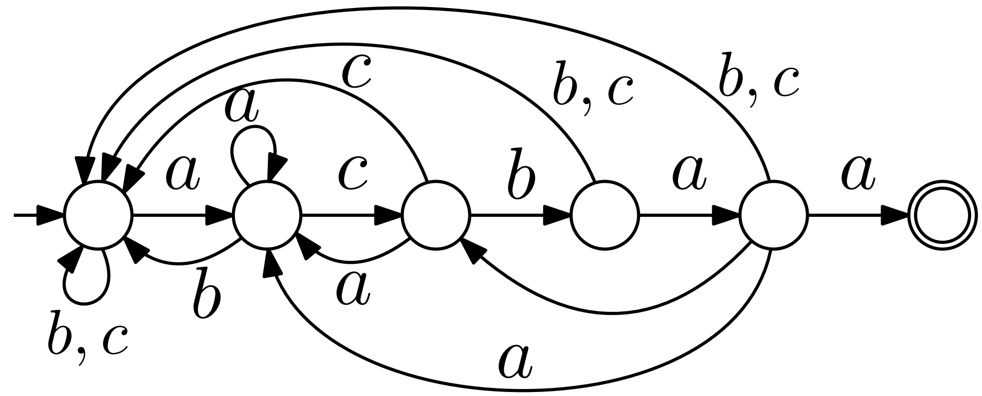
Total expected time: $O(n + m + \frac{n-m}{m} \cdot m) = O(n)$

Other Solutions

- Use a finite state automaton that matches P

$$\Sigma = \{a, b, c\}$$

$$P = \texttt{acbaa}$$

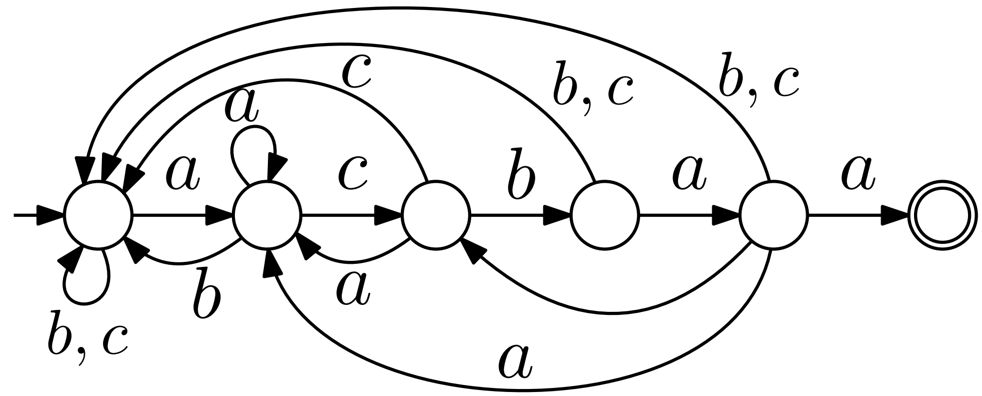


Other Solutions

- Use a finite state automaton that matches P

$\Sigma = \{a, b, c\}$

$P = \texttt{acbaa}$



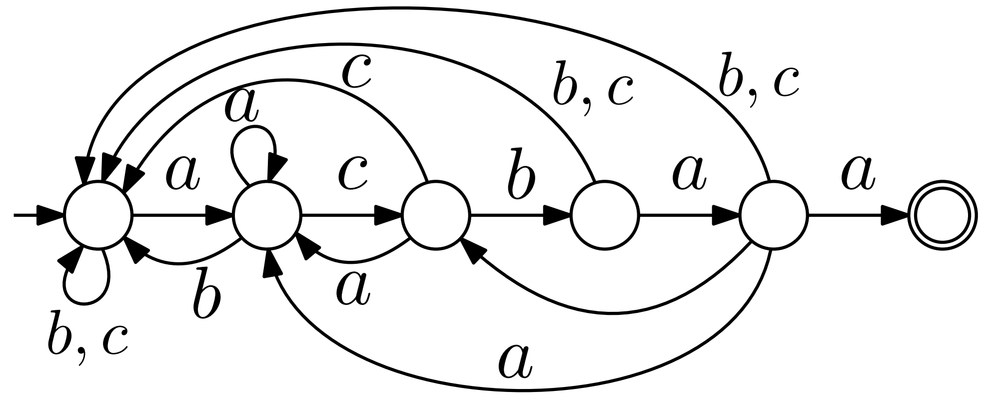
Time: $O(\underbrace{m|\Sigma|}_{\text{construction}} + \underbrace{n}_{\text{time to process } T})$

Other Solutions

- Use a finite state automaton that matches P

$\Sigma = \{a, b, c\}$

$P = \texttt{acbaa}$



Time: $O(\underbrace{m|\Sigma|}_{\text{construction}} + \underbrace{n}_{\text{time to process } T})$

- Knuth-Morris-Pratt: similar to finite state automaton but avoids the explicit computation of the transition function

Time: $O(n)$

Why Use Rabin-Karp?

Finite State Automaton: $O(m|\Sigma| + n)$

Knuth-Morris-Pratt: $O(n)$

Rabin-Karp: $O(n)$ in *expectation*

- Easy to implement
- Easy to extend to multiple patterns
- Can be extended to work in higher dimensions

Find $P = \begin{array}{|c|c|c|}\hline a & b & b \\ \hline b & a & c \\ \hline \end{array}$ in $T = \begin{array}{|c|c|c|c|c|}\hline b & c & a & b & a \\ \hline c & a & b & b & c \\ \hline a & b & a & c & c \\ \hline c & b & c & c & b \\ \hline b & a & a & c & a \\ \hline \end{array}$

How to select q? $M = m^2 \log |\Sigma|$

Efficient option: There are algorithms that select a random prime $q \in \{1, \ldots, M \log M\}$ in expected $O(\text{polylog}\, M) = o(m)$ time.

How to select q? $M = m^2 \log |\Sigma|$

Efficient option: There are algorithms that select a random prime $q \in \{1, \ldots, M \log M\}$ in expected $O(\text{polylog}\, M) = o(m)$ time.

Easy option: Reapeat until a prime is found:

- Pick a random number $q \in \{2, \ldots, M \log M\}$
- Check q for primality: For each $x = 2, \ldots, \lfloor \sqrt{q} \rfloor$
    - Check whether x divides q

How to select q? $M = m^2 \log|\Sigma|$

Efficient option: There are algorithms that select a random prime $q \in \{1, \ldots, M \log M\}$ in expected $O(\text{polylog}\, M) = o(m)$ time.

Easy option: Reapeat until a prime is found:

- Pick a random number $q \in \{2, \ldots, M \log M\}$
- Check q for primality: For each $x = 2, \ldots, \lfloor \sqrt{q} \rfloor$
  - Check whether x divides q

Expected time: if $\log m = \Omega(\log \log |\Sigma|)$

$$O(\underbrace{\log M}_{\#\text{ attempts}} \cdot \underbrace{\sqrt{M \log M}}_{\text{time per attempt}}) = O(m \cdot \log^{3/2} m \cdot \sqrt{\log |\Sigma|})$$

How to select q? $M = m^2 \log |\Sigma|$

Efficient option: There are algorithms that select a random prime $q \in \{1, \ldots, M \log M\}$ in expected $O(\text{polylog}\, M) = o(m)$ time.

Easy option: Reapeat until a prime is found:

- Pick a random number $q \in \{2, \ldots, M \log M\}$
- Check q for primality: For each $x = 2, \ldots, \lfloor \sqrt{q} \rfloor$
  - Check whether x divides q

Expected time: if $\log m = \Omega(\log \log |\Sigma|)$

$$O(\log M \cdot \sqrt{M \log M}) = O(m \cdot \log^{3/2} m \cdot \sqrt{\log |\Sigma|})$$

Lazy option: Hardcode a large prime q.

- Works if $m \lessapprox \sqrt{q}$ and P, T do not depend on q