Knuth Morris and Pratt Algorithm

It exploits the idea of matching prefix with suffix in a pattern itself.

Key observation

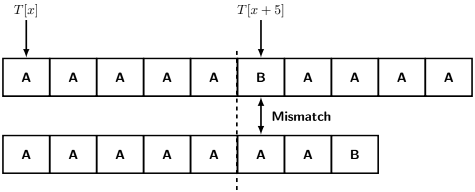
Suppose P has matched k characters with text $T[x, x+1, \ldots, x+k-1]$ and a mismatch occurs at $k+1$, i.e.,

$$P[1..k] = T[x..x+k-1], \text{ and } P[k+1] \neq T[x+k].$$

Then for any $0 < \ell < k$, if $T[x+\ell, \ldots, x+k-1]$ is not a prefix of P, P cannot occur in T at position $x+\ell$.

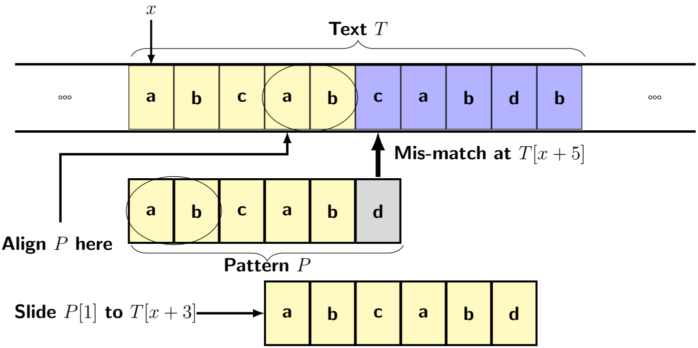
Example

- Consider following situation: P is matched with first five characters of T, and $T[x+5] \neq$ A.



- Shifting P to align $P[1]$ with any of the positions $T[x+1], T[x+2], T[x+3]$, or $T[x+4]$ will not obviously work.

Implication of the Observation



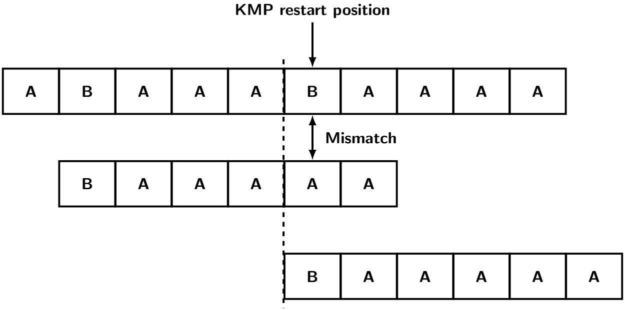
$k = 5$, and $T[x+3, x+4]$ is a prefix of $P[1, \ldots, 6]$. Matching can restart by aligning $P[1..2]$ with $T[x+3, x+4]$

Implication of the Observation

- In general, if first mismatch occurs after match k characters, matching restarts at the leftmost position $x + \ell$ such that $T[x + \ell, \ldots, x + k - 1]$ is a prefix of P
- Equivalently, if T is replaced by P, it also implies m is the smallest index such that:

$$P[\ell + 1, \ldots, k] \text{ is a prefix of } P[1..k].$$

Summary of Observation So Far



- In brute force: every position from $T[2]$ is a restart position.
- Since, none of the proper suffixes: $T[2..5]$, $T[3..5]$, $T[4..5]$, and $T[5..5]$ is a prefix of $P[1..5]$.
- So, matching can only restart at $T[6]$, i.e., after the border.

Knuth Morris Pratt

- The restart position is determined only with respect to already matched positions of T and P.
- This implies that the suffixes of matched portion of T (before the border) are also suffixes of matched part of P.
- Hence, the restart position in a text can be viewed with respect to P itself.
- The underlying idea is: whether any proper suffix of current position of P is a proper prefix of P.

Some Definitions

Definition (*Prefix*)

A prefix of x is a substring u such that $u = x_0 x_1 \cdots x_k$ where $k \in \{0, \ldots, m-1\}$.

Definition (*Suffix*)

A suffix of x is a substring u such that $u = x_{m-k-1} \cdots P_{m-1}$ where $k \in \{0, \ldots, m-1\}$.

Definition (*Proper prefix/suffix*)

A proper prefix (suffix) u of x is called a proper prefix (suffix) respectively, if $u \neq x$, i.e., length of u is less than the length of x.

Examples of Prefix and Suffix

For example, consider the string "**ababa**".

- Its proper prefixes are: "ϵ", "**a**", "**ab**", "**aba**", and "**abab**".
- Its proper suffixes are: "ϵ", "**a**", "**ba**", "**aba**" and "**baba**".
- Only "**a**" and "**aba**" are prefixes that are also suffixes, "**aba**" being the longest.

More on Prefix and Suffix

Definition (*Border*)

A border of x is a substring u is both a proper prefix and a proper suffix of x.

- In other words, u is a border if $u = x_0 x_1 \cdots x_{b-1}$ and $u = x_{k-b} x_{k-b-1} \cdots x_{k-1}$, where $b \in \{0, \cdots, k-1\}$
- E.g., proper prefixes of string **abacab** are: ϵ, **a**, **ab**, **aba**, **abac**, **abaca**
- Proper suffixes are: ϵ, **b**, **ab**, **cab**, **acab**, **bacab**
- Borders are: ϵ, **ab** of widths 0 and 2 respectively.

Prefix Function

Definition

The prefix function is a mapping

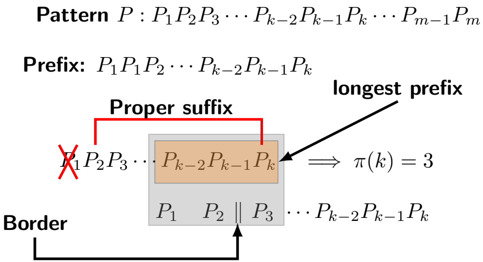
$$\pi : \{1, \cdots, m\} \to \{0, \cdots m-1\}.$$

such that

$$\pi(k) = \max\{i \,|P_i \text{ is a proper suffix of } P_k\},$$

where $P_i = P_1 P_2 \cdots P_i$ and $P_k = P_1 P_2 \cdots P_k$

Prefix Function



The border of the current prefix is P_1, P_2 and P_3 is the next symbol after the border. If $P_k = P_3$, then $\pi(k) = 3$.

Prefix Function Example

- Let P: **AACAAADACAAC**.
- To define $\pi[6]$, we consider P_6 = **AACAAA**, and all its proper prefixes $P_5, P_4, \cdots, \epsilon$.
- Then find out P_2 = **AA** is the longest proper suffix of P_6.
- Hence, $\pi(6) = 2$.

k	1	2	3	4	5	6	7	8	9	10	11	12
$\pi(k)$	0	1	0	1	2	2	0	1	0	1	2	3

Observation From Example

- For example in π table of P: AACAAADACAAC, $\pi(12) = 3$, implying $P_1P_2P_3$: AAC.
  - AAC is a suffix of $P_1 \cdots P_{12}$.
  - As $\pi(\pi(12)) = \pi(3) = 0$, corresponding substring is ϵ also a suffix of the string.
- Similarly, $\pi(11) = 2$, and implying $P_1 \cdots P_{\pi(11)}$ = P_1P_2: AA
  - AA is a suffix of $P_1 \cdots P_{11}$.
  - As $\pi(\pi(11)) = \pi(2) = 1$, corresponding substring $P_1 \cdots P_1$: A is also a suffix of $P_1 \cdots P_{11}$.

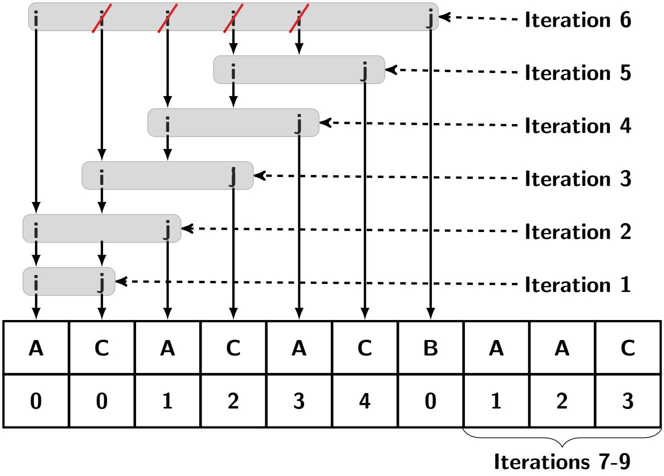
Computing Π

- Prefix function $\pi()$ can be computed incrementally.
- Initialize $\pi(1) = 0$, compute $\pi(2)$ first, then $\pi(3)$, and so on.
- That is if $\pi(1), \pi(2), \ldots, \pi(k)$ are known, then $\pi(k+1)$ can be computed.
- Observation 1: If $P_1 \cdots P_{\pi(i)}$ is a suffix of $P_1 \cdots P_i$ then $P_1 \cdots P_{\pi(i)-1}$ is a suffix of $P_1 \cdots P_{i-1}$.
- Observation 2: All prefixes of P that are suffixes of $P_1 \cdots P_i$ can be obtained by recursively applying π to i
  - E.g., $P_1 \cdots P_{\pi(i)}$, $P_1 \cdots P_{\pi(\pi(i))}$, $P_1 \cdots P_{\pi(\pi(\pi(i)))}$, are all suffixes of $P_1 \cdots P_i$.

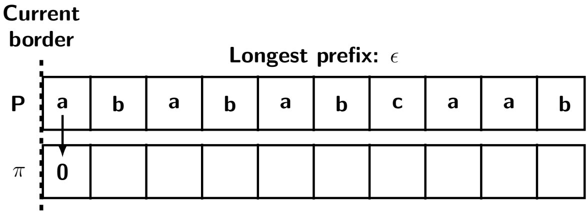
Computing Π

- Denote $\pi^k(i)$ as $\pi()$ applied k times to i
- For example, $\pi^2(i) = \pi(\pi(i))$.
- $\pi(i)$ is equal to $\pi^k(i-1)+1$, where k is the smallest integer that satisfies $P_{\pi^k(i-1)+1} = P_i$
  - If there is no such k then $\pi(i) = 0$.
- Intuition behind this is to look at all prefixes of P that are suffixes of $P_1 \cdots P_i$ and pick the longest one whose next symbol matches P_i.

Example for Computation of Π

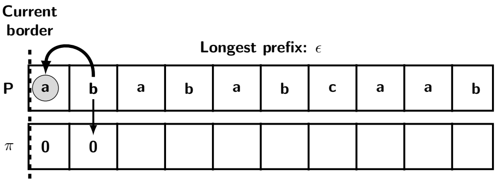


Another Example for Computation of Π



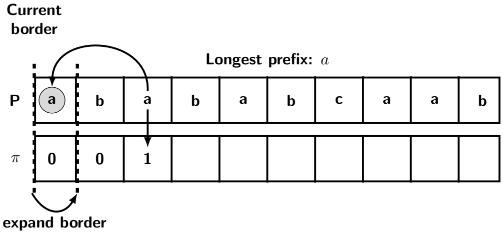
- For $P[1]$ = a, there is no border $\implies \pi(1) = 0$.

Computation of Π (contd.)



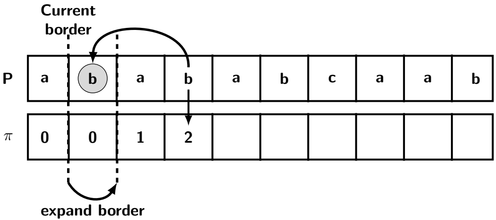
- Similarly for prefix $P[1..2]$ = ab, there is no border, because a $\neq$ b.

Computation of Π (contd.)



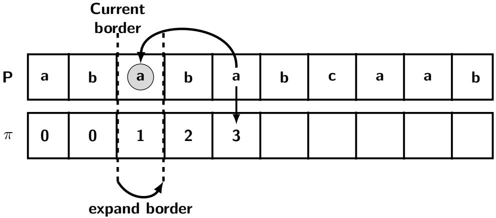
- The border is extended as the current symbol ”a” is equal to the symbol (”a”) after the current border.

Computation of Π (contd.)



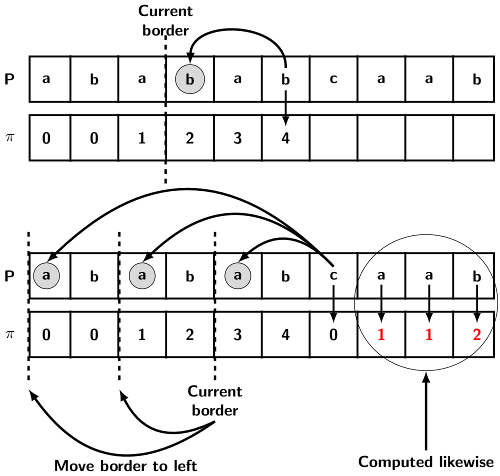
- Extend the border, since current symbol is equal to the symbol after the current border.

Computation of Π (contd.)



- Extend the border, since the current symbol is equal to the symbol after the current border.

Computation of Π (contd.)



Pseudo-code for failure function

```
m = strlen(P);
π[1] = 0;
k = 0;  // Initialization
for (i = 2; i < m; i++) {
    while (k > 0 ∧ P[k+1] ≠ P[i])
        k = π[k];  // Prefix border shrinks
    if (P[k+1] = P[i])
        k++;// Prefix border expands
    π[i] = k;
}
```

Searching Text

- Text search can be done simply by computing prefix function.
- First append an end marker after P call it P'.
- Now append T to P'.
- Computer prefix function for the entire string $P' + T$.
- Now to get the positions of T where P matches just get i such that $\pi(i) = |P|$.
- Then $i - 2|P|$ is the position where P would match.
  - The whole string length is $|T| + |P| + 1$.
  - $i - |P| + 1$ is where occurrence begins.
  - So in text it should occur from position $i - |P| + 1 - (|P| + 1) = i - 2|P|$.

Summary of String Compression & Matching

- Two compression algorithms were discussed, namely, Huffman coding and Lempel-Ziv algorithms.
- Huffman coding is another example of greedy algorithm.
- Lempel Ziv algorithm is based on the idea of incremental parsing.
- We also studied two matching algorithms, namely, Boyer Moore and Knuth Morris and Pratt.
- The basic idea in both algorithm is to avoid certain matching positions based on result of partial matching
- Boyer Moore used bad character and good suffix properties to avoid matching at certain positions.
- KMP algorithm ensure just a single scan by precomputing a failure function for each position of pattern.