Pattern Matching

- Let Σ be a set of alphabets.

- We are given a text $T \in \Sigma^n$ & a pattern $P \in \Sigma^m$; $m \leq n$.

  Qn: Does P appear in T?

- Eg.

$$\begin{array}{lcccccccccccc}
 & 1 & & & & & & & & & i & & n \\
T: & a & a & b & a & c & a & a & b & a & a & a & b \\
P: & & & & & & a & a & b & a & a & & \\
 & & & & & & 1 & & & & m & &
\end{array}$$

- Defn: Pattern matches Text at a location i if:

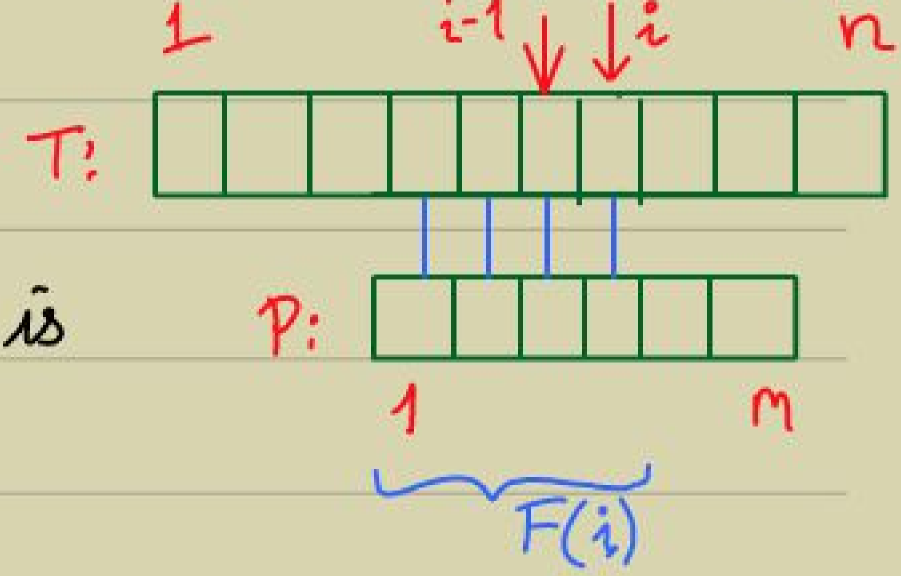
  for $k = 1$ to m, $P[k] = T[i-m+k]$;

Easy: All occurrences of the pattern can be computed in $O(mn)$ time.

- Qn: Is there a sub-quadratic time algorithm for pattern matching?

– What if we remember "some pattern" as we move along in T?

Collaboration: a bit like dynamic prog.



– $F(i)$:= longest prefix of P that is matched at i.

– $f(i) := |F(i)|$.

▷ $f(i) = m \Leftrightarrow$ P matches T at i.

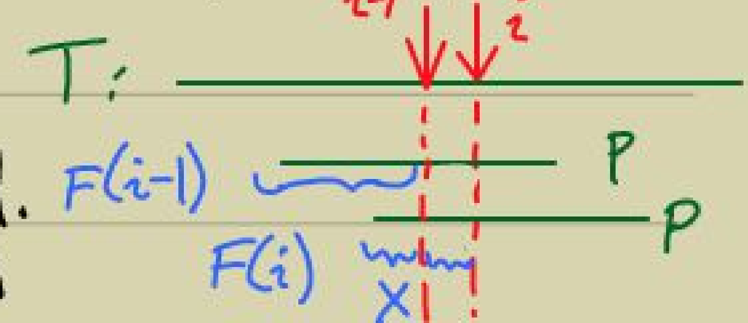
Claim 1: $f(i) \leq f(i-1) + 1$.

Pf: Prefix of $F(i)$ lower bounds $f(i-1)$. □

Claim 2: Nonempty $F(i)$ looks like $X \circ T[i]$ s.t. X is both a prefix & suffix of $F(i-1)$.

Pf: • X lower bounds $F(i-1)$; thus, X is a prefix of $F(i-1)$.

• X matches $F(i-1)$ at end.



□

- Idea to build $F(i)$ from $F(i-1)$:
  - For a string s, let $\underline{\pi(s)}$ be the longest proper prefix of s that is also a suffix of s.
  - Check whether: $F(i-1)$ extends to $F(i)$? by $T[i]$

Else 1) $\pi(F(i-1))$ extends to $F(i)$?

Else 2) $\pi(\pi(F(i-1)))$ " " " ?

Else 3) $\pi^3(F(i-1))$ " " " ?

Else ......

- We need π for $\underline{P_k} := P[1..k]$, for every $k \in [m]$. (Call the length, $\underline{\pi(k)}$)
- Eg. $P = ababab ca$

$\pi(1) = 0$, $\pi(2) = 0$, $\pi(3) = 1$, $\pi(4) = 2$,
$\pi(5) = 3$, $\pi(6) = 4$, $\pi(7) = 0$, $\pi(8) = 1$.

▷ $\pi(\cdot)$ increases by at most one. But, it may decrease by an arbitrary amount.

Pf: $\pi(k+1) \setminus P[k+1]$ lower bounds $\pi(k)$. □

as a lookup array

⇒ Given $F(i-1)$ & Π-function, one can easily compute $F(i)$.

- This is the idea of KMP-algorithm: (Knuth-Morris-Pratt '77)

```
Compute-f(i) {   // given f(i-1) & π(·)
    k ← f(i-1);
    while ( P[k+1] ≠ T[i] & k>0 )
        k ← π(k);   // π(·) is given
    if ( P[k+1] = T[i] )
        f(i) ← k+1;
    else  f(i) ← 0;
    return f(i);
}
```

```
PatternMatch( P[1...m], T[1...n] ) {
    f(0) ← 0;
    for ( i = 1 to n )
        if ( m = Compute-f(i) ) OUTPUT i;
    OUTPUT EOF;   // list of i's ends
}
```

- Analyzing KMP-algorithm:

<u>Claim</u>: Cost of i-th iteration is $O(2+f(i-1)-f(i))$.

<u>Pf</u>: • <u>Case 1</u>: $[f(i)=f(i-1)+1]$ In this case Compute-$f(i)$ takes $O(1)$ time.

• <u>Case 2</u>: $[f(i)\leq f(i-1)]$ Compute-$f(i)$ takes $\leq f(i-1)-f(i)+1$ calls to $\pi(\cdot)$ to return $f(i)$.

lookup array

□

$$\Rightarrow \sum_{i=1}^{n} (\text{cost in } i\text{-th iteration}) \leq O(n).$$

- <u>Qn</u>: How do we compute the fn. π?

<u>Claim</u>: $\pi(\cdot)$ on $P[1\ldots m]$ is $O(m)$-time computable.

<u>Pf</u>: • Note that given $\pi(1),\ldots,\pi(k-1)$, we can compute $\pi(k)$ by a process similar to Compute-$f(i)$.

• Exercise: Analyze like in the claim above.

□

<u>Theorem</u> [KMP'77]: Pattern Matching in linear-time.