Advanced Algorithm

String Matching

Topics To be Covered

- ✓ String Matching Terminology
- ✓ String Matching Applications
- ✓ Naïve String Matching (Brute-Force Algorithm)
- ✓ Horspool's Algorithm
- ✓ String Matching Using Finite Automata
- ✓ Rabin-Karp Algorithm ……and others

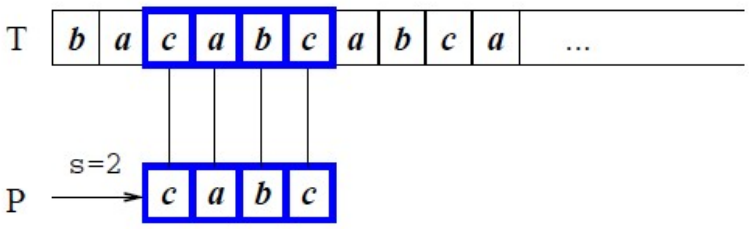
Terminology

Given a text array $T[1 \ldots n]$ and a pattern array $P[1 \ldots m]$ such that the elements of T and P are characters taken from alphabet Σ . e.g., $\Sigma = \{0, 1\}$ or $\Sigma = \{a, b, \ldots, z\}$.

The **String Matching Problem** is to find *all* the occurrence of P in T .

Terminology

A pattern P occurs with **shift** s in T , if $P[1 \ldots m]$ = $T[s + 1 \ldots s + m]$. The String Matching Problem is to find all values of s. Obviously, we must have $0 \leq s \leq n - m$.



Applications

- Password Verification
- Find a Pattern(word) in the PDF or WORD file
- Searching a text in the web page
- Search a computer virus pattern given in virus database into a newly installed software

Brute-Force Algorithm

Initially, P is aligned with T at the first index position. P is then compared with T from **left-to-right**. If a mismatch occurs, "slide" P to *right* by 1 position, and start the comparison again.

Brute-Force Algorithm

```
BF_StringMatcher(T, P) {
 n = length(T);
 m = length(P);

 // s increments by 1 in each iteration
 // => slide P to right by 1
 for (s=0; s<=n-m; s++) {
  // starts the comparison of P and T again
  i=1; j=1;
  while (j<=m && T[s+i]==P[j]) {
   // corresponds to compare P and T from
   // left-to-right
   i++; j++;
  }
  if (j==m+1)
   print "Pattern occurs with shift=", s
 }
}
```

Brute-Force Algorithm

```
BF_StringMatcher(T, P) {
 n = length(T);
 m = length(P);

 // s increments by 1 in each iteration
 // => slide P to right by 1
 for (s=0; s<=n-m; s++) {
  // starts the comparison of P and T again
  i=1; j=1;
  while (j<=m && T[s+i]==P[j]) {
   // corresponds to compare P and T from
   // left-to-right
   i++; j++;
  }
  if (j==m+1)
   print "Pattern occurs with shift=", s
 }
}
```

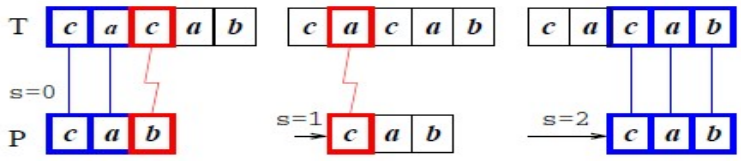
Brute-Force Algorithm

```
BF_StringMatcher(T, P) {
 n = length(T);
 m = length(P);

 // s increments by 1 in each iteration
 // => slide P to right by 1
 for (s=0; s<=n-m; s++) {
  // starts the comparison of P and T again
  i=1; j=1;
  while (j<=m && T[s+i]==P[j]) {
   // corresponds to compare P and T from
   // left-to-right
   i++; j++;
  }
  if (j==m+1)
   print "Pattern occurs with shift=", s
 }
}
```

Brute-Force Algorithm

Initially, P is aligned with T at the first index position. P is then compared with T from **left-to-right**. If a mismatch occurs, "slide" P to *right* by 1 position, and start the comparison again.



Complexity : Θ (m(n-m+1)) is equivalent to Θ(mn)