# So sánh xâu nhanh

String matching

# Nội dung

- String Matching Problem
  - Concept
  - brute force algorithm
  - complexity
- Knuth-Morris-Pratt (KMP) Algorithm
  - Pre-processing
  - complexity

***Tham khảo bài giảng 15-211 Fundamental Data Structures and Algorithms, CMU***

# Pattern Matching Algorithms

# String Matching

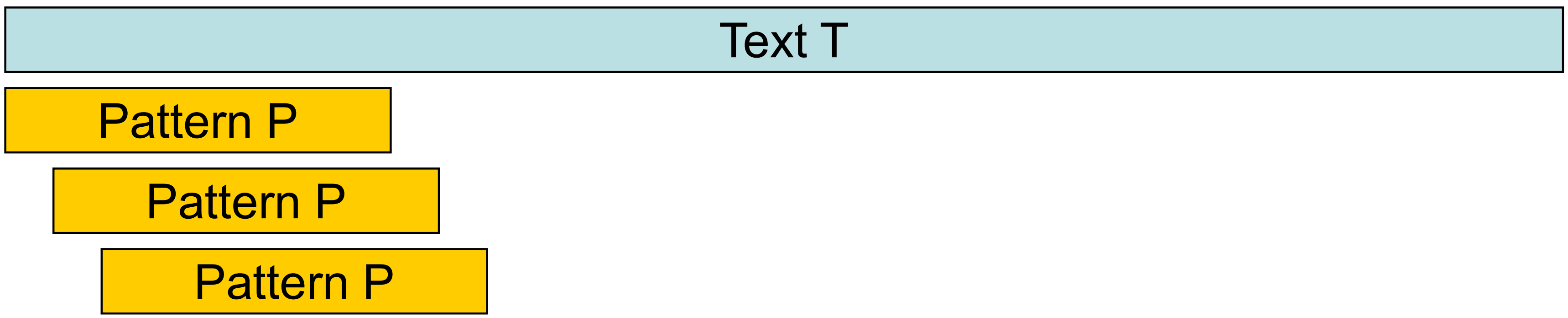
- Text string T[0..N-1]
  T = "**abacaabaccabacabaabb**"
- Pattern string P[0..M-1]
  P = "**abacab**"
- Where is the *first* instance of P in T?
  T[**10..15**] = P[0..5]
- Typically N >> M

# Why String Matching?

- **Applications in Computational Biology**
  - DNA sequence is a long word (or text) over a 4-letter alphabet
  - GTTTGAGTGGTCAGTCTTTTCGTTTCGACGGAGCCCCCAATTA ATAAACTCATAAGCAGACCTCAGTTCGCTTAGAGCAGCCGAA A…..
  - Find a Specific pattern W
- **Finding patterns in documents formed using a large alphabet**
  - Word processing
  - Web searching
  - Desktop search (Google, MSN)
- **Matching strings of bytes containing**
  - Graphical data
  - Machine code
- **grep in Unix/Linux**
  - grep searches for lines matching a pattern.

# Naïve Algorithm
## (or Brute Force)

- Assume |T| = n and |P| = m




Compare until a match is found. If so return the index where match occurs

else return -1

# String Matching

```
abacaabaccabacabaabb
abacab
 abacab
  abacab
   abacab
    abacab
     abacab
      abacab
       abacab
        abacab
         abacab
          abacab
```

- **The brute force algorithm**
- **22+6=28** comparisons.

# A bad case

```
000000000000000001

0000-
 0000-
  0000-
   0000-
    0000-
     0000-
      0000-
       0000-
        0000-
         0000-
          0000-
           0000-
            00001
```

- 60+5 = **65** comparisons are needed
- **How many of them could be avoided?**

# String Matching

- Brute force worst case
  - ***O*(MN)**
  - Expensive for long patterns in repetitive text
- How to improve on this?
- Intuition:
  - *Remember what is learned from previous matches*

# Knuth Morris Pratt (KMP) Algorithm

# KMP – The Big Idea

- Retain information from prior attempts.
- **Compute in advance how far to jump in P when a match fails.**
  - Suppose the match fails at **P[j] $\neq$ T[i+j]**.
  - Then we know **P[0 .. j-1] = T[i .. i+j-1]**.
- We must next try **P[0] ? T[i+1]**.
  - But we know **T[i+1]=P[1]**
  - What if we compare: **P[1] ? P[0]**
    - If so, increment **j** by **1**. No need to look at **T**.
  - What if **P[1] = P[0]** and **P[2] = P[1]** ?
    - Then increment **j** by **2**. Again, no need to look at **T**.
- In general, we can determine how far to jump without any knowledge of **T**!

# Ví dụ: mẫu không lặp

```
P = “abcd”

T = “111abcabcd00…”
        abcd
           abcd
```

# Ví dụ: mẫu có lặp

```
P = “ababc”

T = “111abababc00…”
        ababc
          ababc
```

# Implementing KMP

- Never decrement **i**, ever.
  - Comparing

    **T[i]** with **P[j]**.
- Compute a table **f** of how far to jump **j** forward when a match fails.
  - The next match will compare

    **T[i]** with **P[ f[j-1] ]**
- Do this by matching **P** against itself in all positions.

# Building the Table for f

- P = 1010011
- Find self-overlaps

| Prefix | Overlap | j | f |
|---|---|---|---|
| 1 | . | 1 | 0 |
| 10 | . | 2 | 0 |
| 10**1** | **1** | 3 | 1 |
| 10**10** | **10** | 4 | 2 |
| 10100 | . | 5 | 0 |
| 10100**1** | **1** | 6 | 1 |
| 101001**1** | **1** | 7 | 1 |

# What **f** means

| Prefix | Overlap | j | f |
|---|---|---|---|
| 1 | . | 1 | 0 |
| 10 | . | 2 | 0 |
| 10**1** | **1** | 3 | 1 |
| 10**10** | **10** | 4 | 2 |
| 10100 | . | 5 | 0 |
| 10100**1** | **1** | 6 | 1 |
| 101001**1** | **1** | 7 | 1 |

- If **f** is zero, there is no self-match
  - Set **j=0**
  - Do not change **i**

    The next match is

    **T[i] ? P[0]**

- **f** non-zero implies there is a self-match

  *E.g.,* **f=2** means **P**[0..1] **= P**[j-2..j-1]
  - Hence must start new comparison at **j-2**, since we know **T**[i-2..i-1] = **P**[0..1]
- In general:
  - Set **j = f[j-1]**
  - Do not change **i**.
    - The next match is

      **T[i] ? P[f[j-1]]**

```
T = 10101010011
        10100        j = 5
          1010011    f = 2
```

# Favorable conditions

- P = 1234567
- Find self-overlaps

| Prefix | Overlap | j | f |
|---|---|---|---|
| 1 | . | 1 | 0 |
| 12 | . | 2 | 0 |
| 123 | . | 3 | 0 |
| 1234 | . | 4 | 0 |
| 12345 | . | 5 | 0 |
| 123456 | . | 6 | 0 |
| 1234567 | . | 7 | 0 |

# Mixed conditions

- P = 1231234
- Find self-overlaps

| Prefix | Overlap | j | f |
|---|---|---|---|
| 1 | . | 1 | 0 |
| 12 | . | 2 | 0 |
| 123 | . | 3 | 0 |
| 123**1** | **1** | 4 | 1 |
| 123**12** | **12** | 5 | 2 |
| 123**123** | **123** | 6 | 3 |
| 1231234 | . | 7 | 0 |

# Poor conditions

- P = 1111110
- Find self-overlaps

| Prefix | Overlap | **j** | **f** |
|---|---|---|---|
| 1 | . | 1 | 0 |
| 1**1** | **1** | 2 | 1 |
| 1**11** | **11** | 3 | 2 |
| 1**111** | **111** | 4 | 3 |
| 1**1111** | **1111** | 5 | 4 |
| 1**11111** | **11111** | 6 | 5 |
| 1111110 | . | 7 | 0 |

# KMP pre-process Algorithm

```
m = |P|;
Define a table F of size m
F[0] = 0;
i = 1; j = 0;
while(i<m) {
   compare P[i] and P[j];
   if(P[j]==P[i])
      { F[i] = j+1;
        i++; j++; }
   else if (j>0) j=F[j-1];
   else {F[i] = 0; i++;}
}
```

Use previous values of **f**

# KMP Algorithm

```
input: Text T and Pattern P
|T| = n
|P| = m
Compute Table F for Pattern P
i=j=0

while(i<n) {
   if(P[j]==T[i])
      { if (j==m-1) return i-m+1;
        i++; j++; }
   else if (j>0) j=F[j-1];
   else i++;
}
output: first occurrence of P in T
```

Use **F** to determine next value for **j.**

# Brute Force

```
0000000000000000000000000001
0000000000000-
 0000000000000-
  0000000000000-
   0000000000000-
    0000000000000-
     0000000000000-
      0000000000000-
       0000000000000-
        0000000000000-
         0000000000000-
          0000000000000-
           0000000000000-
            0000000000000-
             0000000000000-
```

- A worse case example:
  **196 + 14 = 210** comparisons

# KMP

```
0000000000000000000000000001

0000000000000-
             0-
              0-
               0-
                0-
                 0-
                  0-
                   0-
                    0-
                     0-
                      0-
                       0-
                        0-
                         0-
                          01
```

**28+14 = 42** comparisons

# KMP Performance

- Pre-processing needs O(M) operations.
- At each iteration, one of three cases:
  - T[i] = P[j]
    - i increases
  - T[i] <> P[j] and j>0
    - i-j increases
  - T[I] <> P[j] and j=0
    - i increases *and* i-j increases
- Hence, maximum of 2N iterations.
- Thus worst case performance is O(N+M).

# Exercises

- E1
  - Construct the KMP table for P = 10010001
  - Trace the KMP algorithm with T = 00010010010001011 1
- E2
  - Construct the KMP table for pattern P = ababaca
  - Trace the KMP algorithm with T = bacbababaabcbab