

Applications:
Pattern Matching algorithms
CS240: Data Structures and Data Management
Slide Set 16

Jérémy Barbay

March 21-23 2006

Outline

Introduction to Pattern Matching

Rabin-Karp

Main Idea

Improvements

Knuth-Morris-Pratt

Main Idea

Failure Function

Boyer-Moore

Character Jump Heuristic

Partial Match Heuristic

Summary of Pattern Matching

Pattern Matching

- ▶ Search for a string in a large body of text
- ▶ T – The text being searched within
- ▶ P – The pattern being searched for
- ▶ Applications:
 - ▶ Information Retrieval
 - ▶ Compression
 - ▶ Data-Mining, classification

Definitions

- ▶ Σ – The alphabet
- ▶ Often T is written as $T[0..n-1]$
- ▶ Often P is written as $P[0..m-1]$
- ▶ Return first i such that

$$P[j] = T[i+j] \quad \text{for } 0 \leq j \leq m-1$$

- ▶ Return -1 if no such i exists
 - ▶ Define T_i as $T[i..(i+m-1)]$
 - ▶ Trying to find a $T_i = P$
- ▶ Example:
 - ▶ $P = \text{Waldo}$
 - ▶ $T = \text{Where's Waldo in the Land of Giants?}$

Naive Algorithm

- ▶ Brute-Force

$Naive(P[0..m-1], T[0..n-1])$

```
for  $i \leftarrow 0$  to  $n - m$  do
  for  $j \leftarrow 0$  to  $m - 1$  do
    if  $T[i + j] = P[j]$  then
      if  $j = m - 1$  then
        return  $i$ 
      end if
    else
      break out of inner ( $j$ ) loop
    end if
  end for
end for
return  $-1$ 
```

Example

- ▶ Example: $P = abba$
- ▶ We will only ever show the explicit character comparisons

	a	b	b	b	a	b	a	b	b	a	b
a	a	b	b	a							
		a									
			a								
				a							
					a	b	b				
						a					
							a	b	b	a	b

Match at position $i = 6$.

- ▶ What is the worst possible input?
 $P = a^{m-1}b$, $T = a^n$
- ▶ Worst case performance
 $(n - m + 1)m \in O(nm)$,
so $O(n^2)$ if $m = \frac{n}{2}$.

Outline

Introduction to Pattern Matching

Rabin-Karp

Main Idea

Improvements

Knuth-Morris-Pratt

Main Idea

Failure Function

Boyer-Moore

Character Jump Heuristic

Partial Match Heuristic

Summary of Pattern Matching

Rabin-Karp

Less iterations of inner loop in the naive algorithm, through quick **hashing test** to eliminate some candidates.

- ▶ Compute pattern's fingerprint $h(P)$
- ▶ For each i compute $h(T[i, \dots, i-1+m])$:
 1. if $h(T[i, \dots, i-1+m]) = h(P)$ then check each position
 2. $h(T[i, \dots, i-1+m]) \neq h(P)$ then move to next position
- ▶ $h(T[i, \dots, i-1+m])$ need to be computed quickly.

Example

$\Sigma = \{0, \dots, 9\}$ and $h(S) = \text{sum of the digits in } S$.

$$\begin{aligned} h(T[i, \dots, i-1+m]) &= T[i] + T[i+1] + \dots + T[i+m-1] \\ &= h(T[i-1, \dots, i-2+m]) \\ &\quad - T[i-1] + T[i-1+m] \end{aligned}$$

Example

$P = 1991$

$h(P) = 20$

	3	8	5	6	8	1	1	9	9	2	1	9	9	1
h	22	27	20	16	19	20	21	21	21	21	20			
			1											
						1	9							
											1	9	9	1

- ▶ Time spent computing hash values: $O(m + n)$
- ▶ Time spent comparing characters, assuming k collisions:
- ▶ Worst case performance: $O(n + km)$
- ▶ Typical results: $O(n)$ with $O(\frac{n}{m})$ hits

Better Signatures

- ▶ Use **polynomial hashing function** (slight variation)
- ▶ c_i is the numeric value of the i 'th character

$$h(P) = \sum_{i=0}^{m-1} c_i \cdot r^{m-1-i}$$

- ▶ Note that r should be greater than the maximum c_i
- ▶ Computing $h(T_i)$ given $h(T_{i-1})$:

$$h(T_i) = (h(T_{i-1}) \pmod{r^{m-1}}) \cdot r + T[i + m - 1]$$

- ▶ Generally work modulo a large prime

Improvements

- ▶ Do not just work entirely with character arrays
- ▶ Preprocess either P or T , and build a new data structure for P or T and then search
- ▶ Example: Multiple searches over a fixed body of text:
Preprocess T once, and use for all future searches

Outline

Introduction to Pattern Matching

Rabin-Karp

Main Idea

Improvements

Knuth-Morris-Pratt

Main Idea

Failure Function

Boyer-Moore

Character Jump Heuristic

Partial Match Heuristic

Summary of Pattern Matching

Knuth-Morris-Pratt

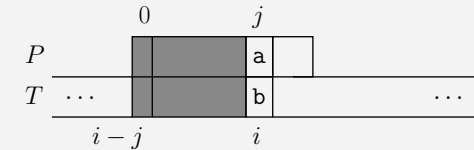
- ▶ When naive mismatches, we advance the string by one: can we skip ahead more than one character?
- ▶ Suppose $P = \text{abcdcaba}$ and mismatch on the last character of P :

$$T = \begin{array}{ccccccccccc} a & b & c & d & c & a & b & c & ? & ? & ? \\ \hline a & b & c & d & c & a & b & a & & & \\ & & & & & a & b & c & d & c & a \end{array}$$

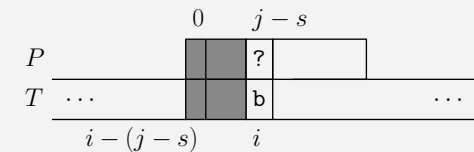
- ▶ How far can we safely move ahead, reusing knowledge from previous matches.

Matching Algorithm

- ▶ Keep an index into each string
 - ▶ i is an index in T while j is an index in P
 - ▶ The first character of T 's substring is always at $i - j$
- ▶ If we mismatch:



- ▶ Shift P ahead s places, and retest at the same spot



- ▶ **Note:** decreasing j has the effect of shifting P

KMP Failure Function

- ▶ What is the correct shift value?
- ▶ For $0 \leq x \leq m - 1$ we define failure function as:

$$f(x) = \begin{cases} 0 & \text{if } x = 0 \\ \text{length of longest prefix of } P \\ \text{that is a suffix of } P[1..x] & \text{if } x > 0 \end{cases}$$

- ▶ Consider $P = \text{abacaba}$

x	$P[1..x]$	P	$f(x)$
0	—	abacaba	0
1	b	abacaba	0
2	ba	abacaba	1
3	bac	abacaba	0
4	baca	abacaba	1
5	bacab	abacaba	2
6	bacaba	abacaba	3

Main Algorithm

```

KMP( $P[0..m-1], T[0..n-1]$ )
   $f \leftarrow \text{KMPPFAILURE}(P)$ 
   $i \leftarrow 0$ 
   $j \leftarrow 0$ 
  while ( $i < n$ ) do
    if  $T[i] = P[j]$  then
      if ( $j = m - 1$ ) then
        return  $i - j$ 
      end if
       $i \leftarrow i + 1$ 
       $j \leftarrow j + 1$ 
    else if ( $j > 0$ ) then
       $j \leftarrow f[j - 1]$ 
    else
       $i \leftarrow i + 1$ 
    end if
  end while
  return  $-1$ 
    
```

Example

$P = \text{abacaba}$

$T = \text{abaxyabacabbaababacaba}$

0	1	2	3	4	5	6	7	8	9	10	11
a	b	a	x	y	a	b	a	c	a	b	b
a	b	a	c								
		(a)	b								
			a								
				a							
					a	b	a	c	a	b	a
									(a)	(b)	a

Exercise: continue with $T = \text{abaxyabacabbaababacaba}$

Computing Failure Function

```

KMPFailure( $P[0..m-1]$ )
 $f[0] \leftarrow 0$ 
 $i \leftarrow 1$ 
 $j \leftarrow 0$ 
while ( $i < m$ ) do
  if ( $P[i] = P[j]$ ) then
     $f[i] \leftarrow j + 1$ 
     $i \leftarrow i + 1$ 
     $j \leftarrow j + 1$ 
  else if ( $j > 0$ ) then
     $j \leftarrow f[j - 1]$ 
  else
     $f[i] \leftarrow 0$ 
     $i \leftarrow i + 1$ 
  end if
end while
return  $f$ 

```

Analysis

- What is the running time to compute the failing function?
 $O(m)$

- What is the running time of KMP?

Each iteration we either:

1. increase i and j by one
2. decrease j by at least one
3. increase only i by one

(1) and (3) execute in time at most n ,
and (2) executes no more often than (1),
hence a total of $2n$ comparison

Hence a total running time of $O(m + n)$.

Example

$P = \text{ababbababa}$

0	1	2	3	4	5	6	7	8	9
a	b	a	b	b	a	b	a	b	a
		a	b	a					
				a					
					a	b	a	b	b
									a

$f(x)$:

0	1	2	3	4	5	6	7	8	9
0	0	1	2	0	1	2	3	4	3

Outline

Main Idea

Main Idea

Failure Function

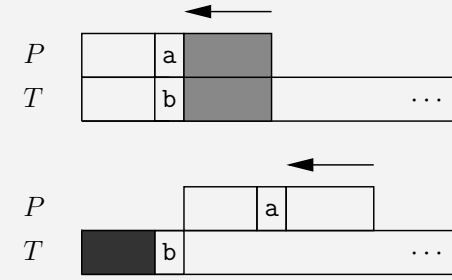
Boyer-Moore

Character Jump Heuristic

Partial Match Heuristic

Boyer-Moore

- ▶ Try matching P **backwards**!
- ▶ Still shift to the right
- ▶ Can skip large parts of T entirely
- ▶ Example:



- ▶ Use two heuristics to decide how far to shift.
- ▶ Pick whichever gives the furthest shift.

Character Jump Heuristic

- ▶ Look at character c we mismatched with in T
- ▶ If c is not in P shift all the way past
- ▶ Otherwise shift P to line up the **last** occurrence of c in P with the one in T
- ▶ **Note:** We should always shift at least one
- ▶ Use an array which for each character of the alphabet indicates the position where to jump.

Examples

P = a l d o
T = w h e r e i s w a l d o
o
o
a l d o

6 comparisons

$$\begin{array}{ccccccc} P & = & m & o & o & r & e \\ T & = & b & o & y & e & r \end{array}$$

$\begin{matrix} e \\ (r) \end{matrix}$

$\begin{matrix} e \\ (m) \end{matrix}$

$m \quad o \quad o \quad r \quad e$

6 comparisons

Partial Match Heuristic

- ▶ Similarly to KMP shift function, line up characters already matched in P with an occurrence further left.
- ▶ Use an array which for each position of the pattern indicates the position where to jump.

Examples:

```
P = b a n a n a
T = b o b a n a n a
      n a n a
      b a n a n a

P = b o b o
T = r o b o t i c s
      b o b o
          o
```

Results

- ▶ Worst-case running time $\in O(n + m + |\Sigma|)$
- ▶ Works very well when m is large
- ▶ Alphabet should not be too small
- ▶ On typical English text B-M probes approximately 25% of the characters in T

Outline

Introduction to Pattern Matching

Rabin-Karp

Main Idea

Improvements

Knuth-Morris-Pratt

Main Idea

Failure Function

Boyer-Moore

Character Jump Heuristic

Partial Match Heuristic

Summary of Pattern Matching

Summary

- ▶ Naive: $O(nm)$
- ▶ Rabin-Karp: Hashing, still $O(nm)$
- ▶ Knuth-Morris-Pratt Large/Intelligent Skip-Ahead, $2n + O(m)$
- ▶ Boyer-Moore Backward Matching + Intelligent Skip-Ahead, $O(n + m + |\Sigma|)$ in worst case, sublinear in practice
- ▶ If more time, preprocess Pattern to produce automata $n + O(m|\Sigma|)$.

Reading Materials

	Topic	GT	CLRS
	Rabin-Karp	418–421	906–922
	KMP+BM	422–428	923–930

- ▶ GT = Algorithm Design, by Goodrich & Tamassia
- ▶ CLRS = Introduction to Algorithms, by Cormen, Leiserson, Rivest & Stein