



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

ACM-ICPC Thailand Central Group B Training Program

Faculty of ICT, Mahidol University

Search

Keerati Jittrawong

August 27, 2016

Slides from Sedgewick & Wayne



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE



ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก



5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGIEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Substring search

Goal. Find pattern of length M in a text of length N

→ typically $N \gg M$

pattern → N E E D L

ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

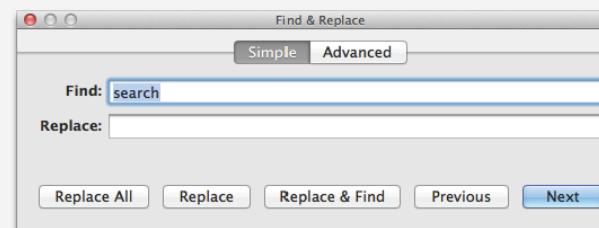
Substring search applications

Goal. Find pattern of length M in a text of length N

→ typically $N \gg 1$

pattern → N E E D L |

The diagram illustrates the step where the character 'E' at index 4 of the pattern 'NEEDLE' is compared with the character 'A' at index 4 of the string 'HAYSTACK'. The string is shown as 'I N A H A Y S T A C K' above a red underline 'N E E D L E'. An arrow points from the 'E' in 'NEEDLE' to the 'A' in 'HAYSTACK', with the word 'match' written below the arrow.



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

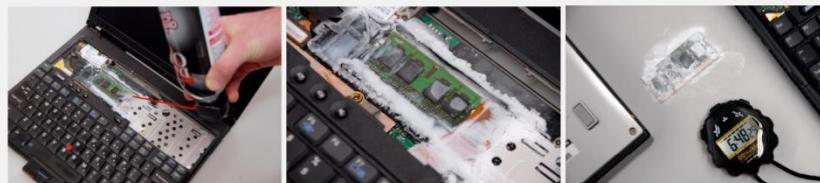
Substring search applications

Goal. Find pattern of length M in a text of length N

typically $N \gg M$

pattern → N E E D L E

Computer forensics. Search memory or disk for signatures e.g., all URLs or RSA keys that the user has entered.



<http://citp.princeton.edu/memory>

ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Substring search applications

Goal. Find pattern of length M in a text of length N

→ typically $N \gg M$

pattern → N E E D L I

Identify patterns indicative of spam

- PROFITS
 - LOSE WEIGHT
 - herbal Viagra
 - There is no catch.
 - This is a one-time mailing.
 - This message is sent in compliance with spam regulations

SpamAssassin





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Substring search applications

Electronic surveillance.



Need to monitor all
internet traffic.
(security)



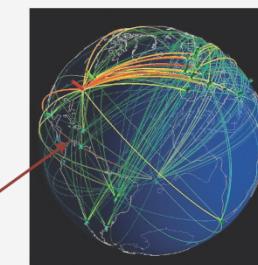
Well, we're mainly
interested in
“ATTACK AT DAWN”



No way!
(privacy)



OK. Build a
machine that just
looks for that.



“ATTACK AT DAWN”
substring search
machine

found



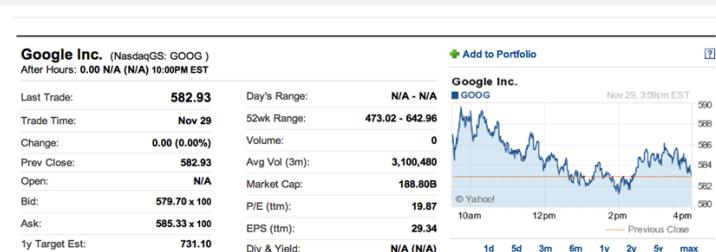
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Substring search applications

Screen scraping. Extract relevant data from web page.

Ex. Find string delimited by and after first occurrence of pattern Last Trade:.



<http://finance.yahoo.com/q?s=goog>

```
...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledata1">
<big><b>452.92</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledata1">
...
```



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Screen scraping: Java implementation

Java library. The `indexOf()` method in Java's string library returns the index of the first occurrence of a given string, starting at a given offset.

```
public class StockQuote
{
    public static void main(String[] args)
    {
        String name = "http://finance.yahoo.com/q?s=";
        In in = new In(name + args[0]);
        String text = in.readAll();
        int start    = text.indexOf("Last Trade:", 0);
        int from     = text.indexOf("<b>", start);
        int to       = text.indexOf("</b>", from);
        String price = text.substring(from + 3, to);
        StdOut.println(price);
    }
}
```

```
% java StockQuote goog
582.93
```

```
% java StockQuote msft
24.84
```



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก



5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGIEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Brute-force substring search

Check for pattern starting at each text position.

i	j	i+j	0	1	2	3	4	5	6	7	8	9	10
			txt → A	B	A	C	A	D	A	B	R	A	C
0	2	2	A	B	R	A							
1	0	1		A	B	R	A						
2	1	3			A	B	R	A					
3	0	3				A	B	R	A				
4	1	5					A	B	R	A			
5	0	5						A	B	R	A		
6	4	10							A	B	R	A	

entries in red are mismatches

entries in gray are for reference only

entries in black match the text

return i when j is M

match



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Brute-force substring search: Java implementation

Check for pattern starting at each text position.

i	j	i+j	0	1	2	3	4	5	6	7	8	9	10
			A	B	A	C	A	D	A	B	R	A	C
4	3	7					A	D	A	C	R		
5	0	5						A	D	A	C	R	

```
public static int search(String pat, String txt)
{
    int M = pat.length();
    int N = txt.length();
    for (int i = 0; i <= N - M; i++)
    {
        int j;
        for (j = 0; j < M; j++)
            if (txt.charAt(i+j) != pat.charAt(j))
                break;
        if (j == M) return i; ← index in text where
                           pattern starts
    }
    return N; ← not found
}
```



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Brute-force substring search: worst case

Brute-force algorithm can be slow if text and pattern are repetitive.

i	j	i+j	0	1	2	3	4	5	6	7	8	9
			txt → A	A	A	A	A	A	A	A	A	B
0	4	4	A	A	A	A	B ← pat					
1	4	5		A	A	A	A	B				
2	4	6			A	A	A	A	B			
3	4	7				A	A	A	A	B		
4	4	8					A	A	A	A	B	
5	5	10						A	A	A	A	B

↑
match

Worst case. $\sim MN$ char compares.



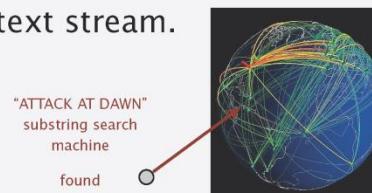
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

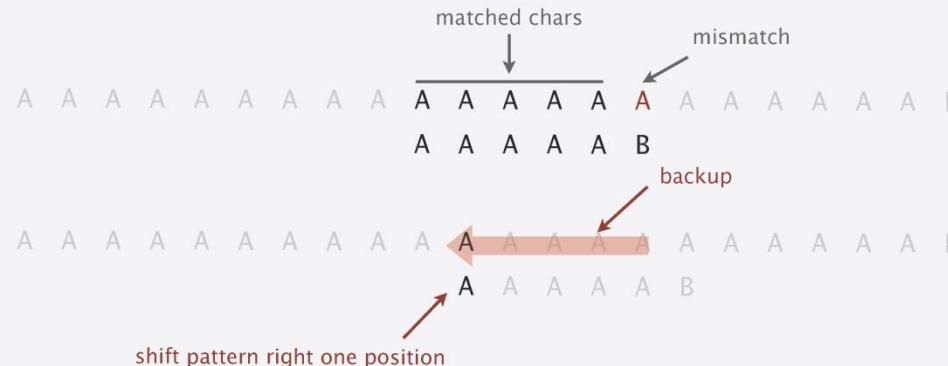
Backup

In many applications, we want to avoid **backup** in text stream.

- Treat input as stream of data.
- Abstract model: standard input.



Brute-force algorithm needs backup for every mismatch.



Approach 1. Maintain buffer of last M characters.

Approach 2. Stay tuned.



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Brute-force substring search: alternate implementation

Same sequence of char compares as previous implementation.

- *i* points to end of sequence of already-matched chars in text.
- *j* stores # of already-matched chars (end of sequence in pattern).

<i>i</i>	<i>j</i>	0	1	2	3	4	5	6	7	8	9	10
		A	B	A	C	A	D	A	B	R	A	C
7	3					A	D	A	C	R		
5	0					A	D	A	C	R		

```
public static int search(String pat, String txt)
{
    int i, N = txt.length();
    int j, M = pat.length();
    for (i = 0, j = 0; i < N && j < M; i++)
    {
        if (txt.charAt(i) == pat.charAt(j)) j++;
        else { i -= j; j = 0; } ← explicit backup
    }
    if (j == M) return i - M;
    else return N;
}
```



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Algorithmic challenges in substring search

Brute-force is not always good enough.

Theoretical challenge. Linear-time guarantee. ← fundamental algorithmic problem

Practical challenge. Avoid backup in text stream. ← often no room or time to save text

Now is the time for all people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for a lot of good people to come to the aid of their party. Now is the time for all of the good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for each good person to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many or all good people to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party. Now is the time for all people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for a lot of good people to come to the aid of their party. Now is the time for all of the good people to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for each person to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for all good Republicans to come to the aid of their party. Now is the time for all good people to come to the aid of their party. Now is the time for many or all good people to come to the aid of their party. Now is the time for all good Democrats to come to the aid of their party.



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก



5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ ***Knuth-Morris-Pratt***
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*

Algorithms

ROBERT SEDGIEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>

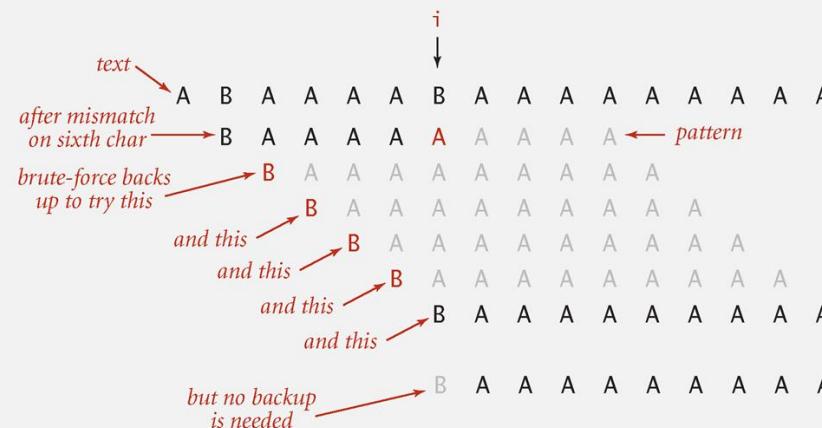
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt substring search

Intuition. Suppose we are searching in text for pattern BAAAAAAAAB

- Suppose we match 5 chars in pattern, with mismatch on 6th char
 - We know previous 6 chars in text are BAAAAAB.
 - Don't need to back up text pointer! 



Knuth-Morris-Pratt algorithm. Clever method to always avoid backup. (!)

ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Deterministic finite state automaton (DFA)

DFA is abstract string-searching machine.

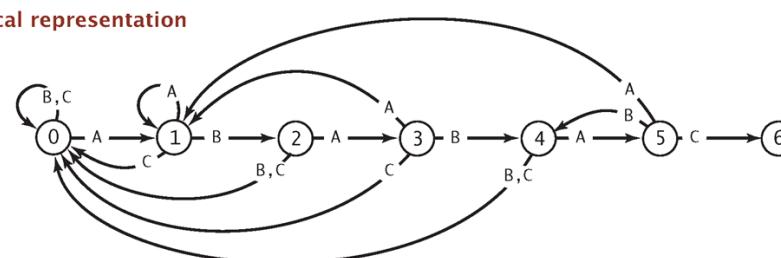
- Finite number of states (including start and halt).
- Exactly one transition for each char in alphabet.
- Accept if sequence of transitions leads to halt state.

internal representation

j	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
A	1	1	3	1	5	1
dfa[][][j]	B	0	2	0	4	0

If in state j reading char C:
if j is 6 halt and accept
else move to state dfa[c][j]

graphical representation





ACM-ICPC

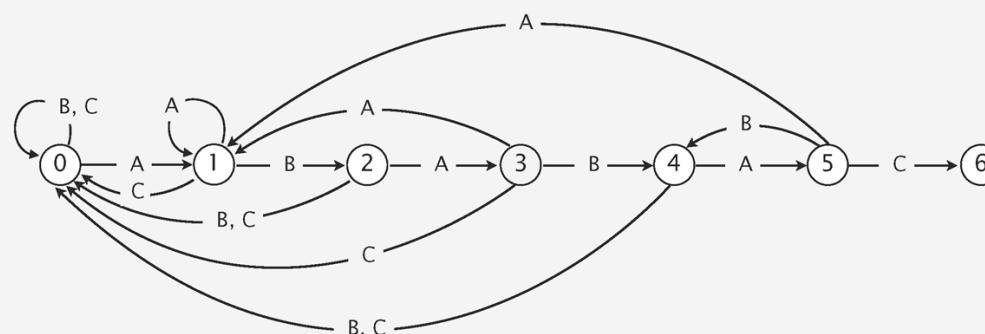
ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A



	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
A	1	1	3	1	5	1
dfa[][][j]	B	0	2	0	4	0
C	0	0	0	0	0	6



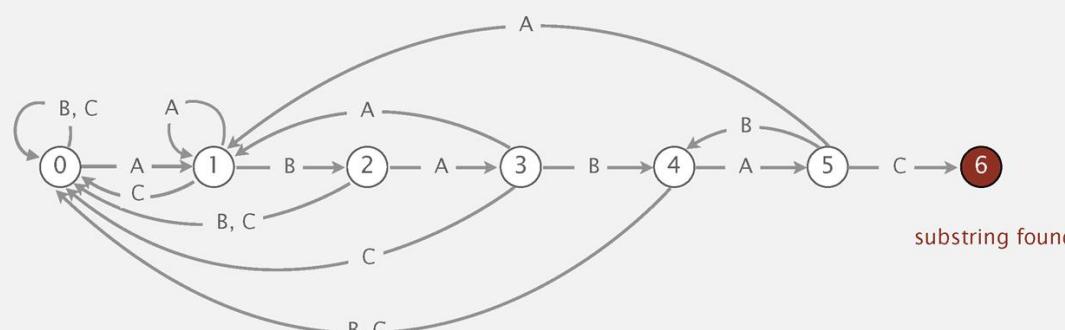
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA simulation

A A B A C A A B A B A C A A

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	A	1	1	3	1	5
	B	0	2	0	4	0
	C	0	0	0	0	0





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

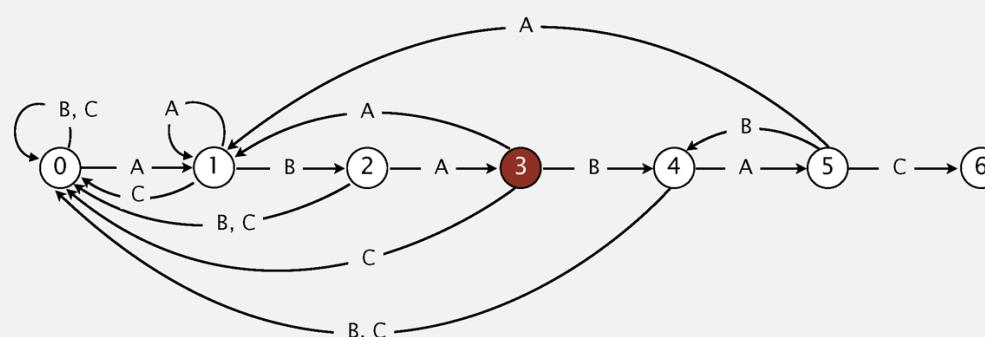
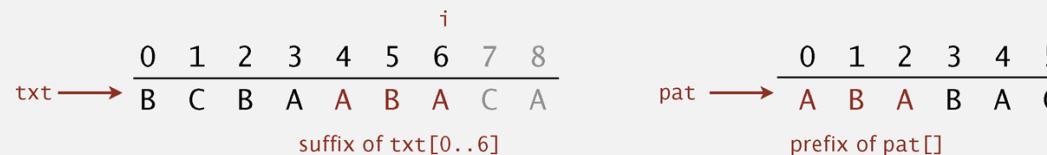
Interpretation of Knuth-Morris-Pratt DFA

Q. What is interpretation of DFA state after reading in $\text{txt}[i]$?

A. State = number of characters in pattern that have been matched.

length of longest prefix of $\text{pat}[]$
that is a suffix of $\text{txt}[0..i]$

Ex. DFA is in state 3 after reading in $\text{txt}[0..6]$.





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt substring search: Java implementation

Key differences from brute-force implementation.

- Need to precompute `dfa[][]` from pattern.
- Text pointer `i` never decrements.

```
public int search(String txt)
{
    int i, j, N = txt.length();
    for (i = 0, j = 0; i < N && j < M; i++)
        j = dfa[txt.charAt(i)][j];           ← no backup
    if (j == M) return i - M;
    else         return N;
}
```

Running time.

- Simulate DFA on text: at most N character accesses.
- Build DFA: how to do efficiently? [warning: tricky algorithm ahead]



ACM-ICPC

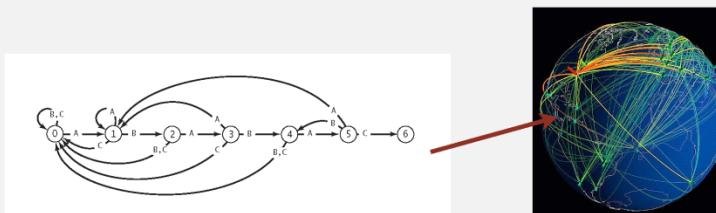
ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt substring search: Java implementation

Key differences from brute-force implementation.

- Need to precompute `dfa[][]` from pattern.
- Text pointer `i` never decrements.
- Could use `input stream`.

```
public int search(In in)
{
    int i, j;
    for (i = 0, j = 0; !in.isEmpty() && j < M; i++)
        j = dfa[in.readChar()][j]; ← no backup
    if (j == M) return i - M;
    else         return NOT_FOUND;
}
```





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA construction



	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	A	B				
	C					

Constructing the DFA for KMP substring search for A B A B A C

①

②

③

④

⑤

⑥



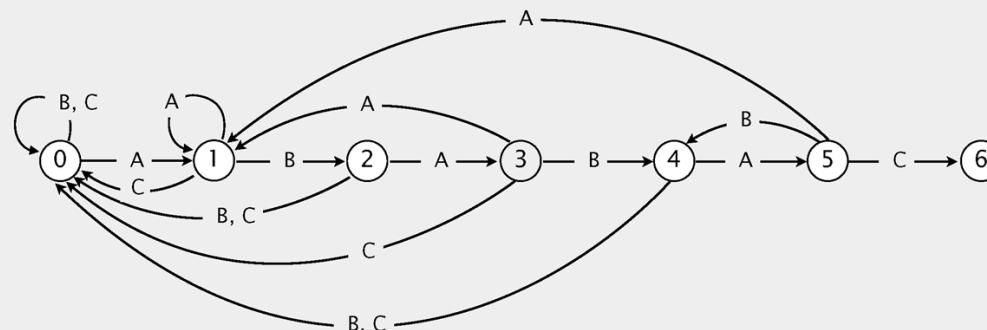
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA construction

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	A	1	1	3	1	5
	B	0	2	0	4	0
	C	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

How to build DFA from pattern?

Include one state for each character in pattern (plus accept state).

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[] [j]	A					
	B					
	C					

①

②

③

④

⑤

⑥



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

How to build DFA from pattern?

Match transition. If in state j and next char $c == \text{pat.charAt}(j)$, go to $j+1$.



first j characters of pattern
have already been matched



next char matches



now first $j+1$ characters of
pattern have been matched

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
A	1		3		5	
dfa[][][j]	B		2		4	
C						6





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

How to build DFA from pattern?

Mismatch transition. If in state j and next char $c \neq \text{pat.charAt}(j)$,
then the last $j-1$ characters of input are $\text{pat}[1..j-1]$, followed by c .

To compute $\text{dfa}[c][j]$: Simulate $\text{pat}[1..j-1]$ on DFA and take transition c .

Running time. Seems to require j steps.

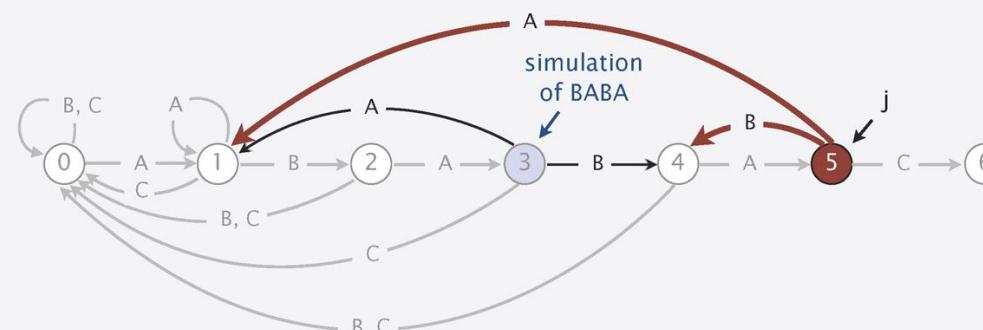
still under construction (!)

Ex. $\text{dfa}['A'][5] = 1; \quad \text{dfa}['B'][5] = 4$

simulate BABA;
take transition 'A'
 $= \text{dfa}['A'][3]$

simulate BABA;
take transition 'B'
 $= \text{dfa}['B'][3]$

pat.charAt(j)	0	1	2	3	4	5
	A	B	A	B	A	C



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

How to build DFA from pattern?

Mismatch transition. If in state j and next char $c \neq \text{pat.charAt}(j)$, then the last $j-1$ characters of input are $\text{pat}[1..j-1]$, followed by c .

state X

To compute $\text{dfa}[c][j]$: Simulate $\text{pat}[1..j-1]$ on DFA and take transition c .

Running time. Takes only constant time if we maintain state X.

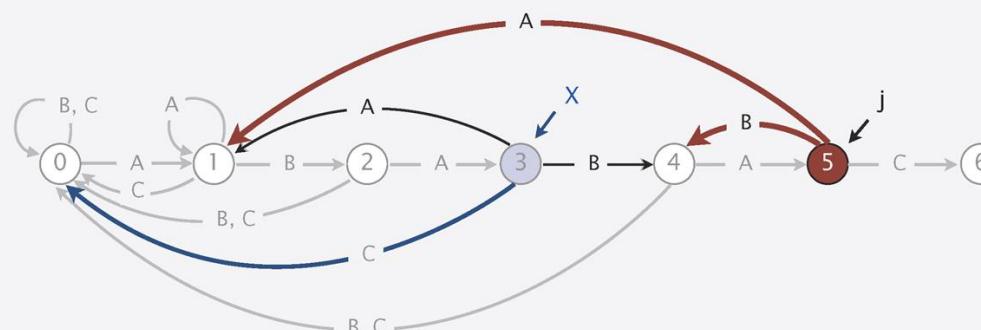
Ex. $\text{dfa}['A'][5] = 1$; $\text{dfa}['B'][5] = 4$ $X' = 0$

from state X,
take transition 'A'
 $= \text{dfa}['A'][X]$

from state X,
take transition 'B'
 $= \text{dfa}['B'][X]$

from state X,
take transition 'C'
 $= \text{dfa}['C'][X]$

0	1	2	3	4	5
A	B	A	B	A	C





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA construction in linear time

Include one state for each character in pattern (plus accept state).



	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	A					
	B					
	C					

Constructing the DFA for KMP substring search for A B A B A C

①

②

③

④

⑤

⑥



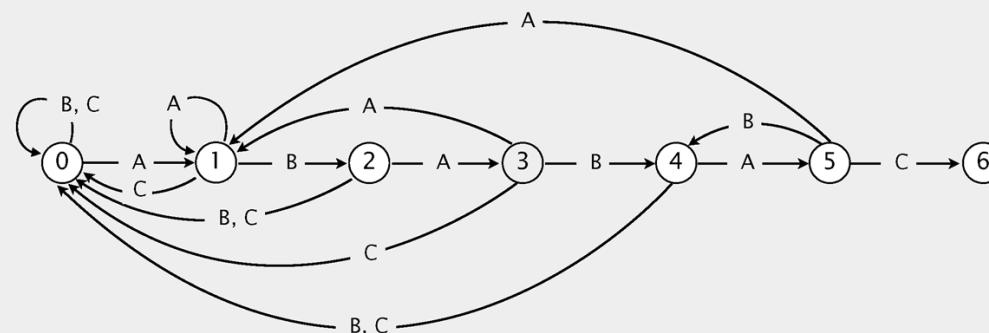
ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt demo: DFA construction in linear time

	0	1	2	3	4	5
pat.charAt(j)	A	B	A	B	A	C
dfa[][][j]	A	1	1	3	1	5
	B	0	2	0	4	0
	C	0	0	0	0	6

Constructing the DFA for KMP substring search for A B A B A C





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Constructing the DFA for KMP substring search: Java implementation

For each state j :

- Copy $dfa[] [X]$ to $dfa[] [j]$ for mismatch case.
- Set $dfa[pat.charAt(j)] [j]$ to $j+1$ for match case.
- Update X .

```
public KMP(String pat)
{
    this.pat = pat;
    M = pat.length();
    dfa = new int[R][M];
    dfa[pat.charAt(0)][0] = 1;
    for (int X = 0, j = 1; j < M; j++)
    {
        for (int c = 0; c < R; c++)
            dfa[c][j] = dfa[c][X]; ← copy mismatch cases
        dfa[pat.charAt(j)][j] = j+1; ← set match case
        X = dfa[pat.charAt(j)][X]; ← update restart state
    }
}
```

Running time. M character accesses (but space/time proportional to $R M$).



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

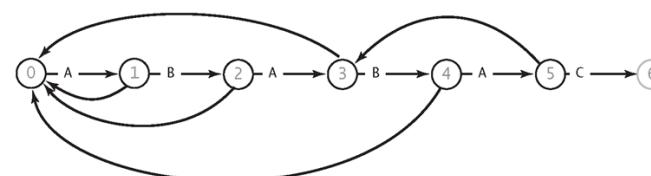
KMP substring search analysis

Proposition. KMP substring search accesses no more than $M + N$ chars to search for a pattern of length M in a text of length N .

Pf. Each pattern char accessed once when constructing the DFA; each text char accessed once (in the worst case) when simulating the DFA.

Proposition. KMP constructs $\text{dfa}[][]$ in time and space proportional to $R M$.

Larger alphabets. Improved version of KMP constructs $\text{nfa}[]$ in time and space proportional to M .



KMP NFA for ABABAC



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Knuth-Morris-Pratt: brief history

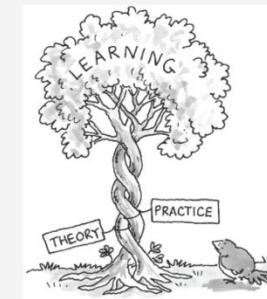
- Independently discovered by two theoreticians and a hacker.
 - Knuth: inspired by esoteric theorem, discovered linear algorithm
 - Pratt: made running time independent of alphabet size
 - Morris: built a text editor for the CDC 6400 computer
- Theory meets practice.

SIAM J. COMPUT.
Vol. 6, No. 2, June 1977

FAST PATTERN MATCHING IN STRINGS*

DONALD E. KNUTH†, JAMES H. MORRIS, JR.‡ AND VAUGHAN R. PRATT§

Abstract. An algorithm is presented which finds all occurrences of one given string within another, in running time proportional to the sum of the lengths of the strings. The constant of proportionality is low enough to make this algorithm of practical use, and the procedure can also be extended to deal with some more general pattern-matching problems. A theoretical application of the algorithm shows that the set of concatenations of even palindromes, i.e., the language $\{\alpha\alpha^R\}^*$, can be recognized in linear time. Other algorithms which run even faster on the average are also considered.



Don Knuth



Jim Morris



Vaughan Pratt



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก



5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*



Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>





Robert Boyer J. Strother Moore



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Intuition.

- Scan characters in pattern from right to left.
- Can skip as many as M text chars when finding one not in the pattern.

i	j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	
		<i>text</i> →	F	I	N	D	I	N	A	H	A	Y	S	T	A	C	K	N	E	E	D	L	E	I	N	A
0	5		N	E	E	D	L	E	←	<i>pattern</i>																
5	5								N	E	E	D	L	E												
11	4													N	E	E	D	L	E							
15	0														N	E	E	D	L	E						

return i = 15



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 1. Mismatch character not in pattern.



mismatch character 'T' not in pattern: increment i one character beyond 'T'



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2a. Mismatch character in pattern.



mismatch character 'N' in pattern: align text 'N' with rightmost pattern 'N'



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2b. Mismatch character in pattern (but heuristic no help).

before	i	
txt	.	E L E . . .
pat	N E E D L E	

aligned with rightmost E?	i	
txt	.	E L E . . .
pat	N E E D L E	

mismatch character 'E' in pattern: align text 'E' with rightmost pattern 'E' ?

ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
 รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 2b. Mismatch character in pattern (but heuristic no help)

mismatch character 'E' in pattern: increment i by



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

A. Precompute index of rightmost occurrence of character c in pattern.
(-1 if character not in pattern)

```
right = new int[R];
for (int c = 0; c < R; c++)
    right[c] = -1;
for (int j = 0; j < M; j++)
    right[pat.charAt(j)] = j;
```

c	N	E	E	D	L	E	right[c]
	0	1	2	3	4	5	
A	-1	-1	-1	-1	-1	-1	-1
B	-1	-1	-1	-1	-1	-1	-1
C	-1	-1	-1	-1	-1	-1	-1
D	-1	-1	-1	-1	③	3	3
E	-1	-1	①	②	2	2	5
...							-1
L	-1	-1	-1	-1	-1	④	4
M	-1	-1	-1	-1	-1	-1	-1
N	-1	0	0	0	0	0	0
...							-1

Boyer-Moore skip table computation



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: Java implementation

```
public int search(String txt)
{
    int N = txt.length();
    int M = pat.length();
    int skip;
    for (int i = 0; i <= N-M; i += skip)
    {
        skip = 0;
        for (int j = M-1; j >= 0; j--)
        {
            if (pat.charAt(j) != txt.charAt(i+j))
            {
                skip = Math.max(1, j - right[txt.charAt(i+j)]);
                break;
            }
        }
        if (skip == 0) return i; ← match
    }
    return N;
}
```

compute skip value

in case other term is nonpositive



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Boyer-Moore: analysis

Property. Substring search with the Boyer-Moore mismatched character heuristic takes about $\sim N / M$ character compares to search for a pattern of length M in a text of length N . sublinear!

Worst-case. Can be as bad as $\sim M N$.

i	skip	0	1	2	3	4	5	6	7	8	9
	txt →	B	B	B	B	B	B	B	B	B	B
0	0	A	B	B	B	B	← pat				
1	1		A	B	B	B	B				
2	1			A	B	B	B	B			
3	1				A	B	B	B	B		
4	1					A	B	B	B	B	
5	1						A	B	B	B	B

Boyer-Moore variant. Can improve worst case to $\sim 3 N$ character compares by adding a KMP-like rule to guard against repetitive patterns.



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก



5.3 SUBSTRING SEARCH

- ▶ *introduction*
- ▶ *brute force*
- ▶ *Knuth-Morris-Pratt*
- ▶ *Boyer-Moore*
- ▶ *Rabin-Karp*



Algorithms

ROBERT SEDGWICK | KEVIN WAYNE

<http://algs4.cs.princeton.edu>



Michael Rabin
Dick Karp



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp fingerprint search

Basic idea = modular hashing.

- Compute a hash of $\text{pat}[0..M-1]$.
- For each i , compute a hash of $\text{txt}[i..M+i-1]$.
- If pattern hash = text substring hash, check for a match.

pat.charAt(i)																					
i	0	1	2	3	4																
	2	6	5	3	5																
	2	6	5	3	5	% 997 = 613															
txt.charAt(i)																					
i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15					
	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3					
0	3	1	4	1	5	% 997 = 508															
1		1	4	1	5	9	% 997 = 201														
2			4	1	5	9	2	% 997 = 715													
3				1	5	9	2	6	% 997 = 971												
4					5	9	2	6	5	% 997 = 442											
5						9	2	6	5	3	% 997 = 929										
6	←	return i = 6					2	6	5	3	5	% 997 = 613									

modular hashing with R = 10 and $\text{hash}(s) = s \pmod{997}$



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Modular arithmetic

Math trick. To keep numbers small, take intermediate results modulo Q .

Ex.

$$\begin{aligned}(10000 + 535) * 1000 &\pmod{997} \\&= (30 + 535) * 3 \pmod{997} \\&= 1695 \pmod{997} \\&= 698 \pmod{997}\end{aligned}$$

$$(a + b) \pmod{Q} = ((a \pmod{Q}) + (b \pmod{Q})) \pmod{Q}$$

$$(a * b) \pmod{Q} = ((a \pmod{Q}) * (b \pmod{Q})) \pmod{Q}$$

two useful modular arithmetic identities



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Efficiently computing the hash function

Modular hash function. Using the notation t_i for `txt.charAt(i)`, we wish to compute

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0 \pmod{Q}$$

Intuition. M -digit, base- R integer, modulo Q .

Horner's method. Linear-time method to evaluate degree- M polynomial.

	pat.charAt()				
i	0	1	2	3	4
2	6	5	3	5	
0	2	% 997 = 2			
1	2	6 % 997 = (2*10 + 6) % 997 = 26	R	Q	
2	2	6 % 997 = (26*10 + 5) % 997 = 265			
3	2	6 % 997 = (265*10 + 3) % 997 = 659			
4	2	6 % 997 = (659*10 + 5) % 997 = 613			

```
// Compute hash for M-digit key
private long hash(String key, int M)
{
    long h = 0;
    for (int j = 0; j < M; j++)
        h = (h * R + key.charAt(j)) % Q;
    return h;
}
```

$$\begin{aligned} 26535 &= 2*10000 + 6*1000 + 5*100 + 3*10 + 5 \\ &= (((2 * 10 + 6) * 10 + 5) * 10 + 3) * 10 + 5 \end{aligned}$$



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Efficiently computing the hash function

Challenge. How to efficiently compute x_{i+1} given that we know x_i .

$$x_i = t_i R^{M-1} + t_{i+1} R^{M-2} + \dots + t_{i+M-1} R^0$$

$$x_{i+1} = t_{i+1} R^{M-1} + t_{i+2} R^{M-2} + \dots + t_{i+M} R^0$$

Key property. Can update "rolling" hash function in constant time!

$$x_{i+1} = (x_i - t_i R^{M-1}) R + t_{i+M}$$

↑ current value ↑ subtract leading digit ↑ multiply by radix ↑ add new trailing digit (can precompute R^{M-1})

i	...	2	3	4	5	6	7	...
current value		1	4	1	5	9	2	6 5 > text
new value		4	1	5	9	2	6	5
		4	1	5	9	2		<i>current value</i>
		- 4	0	0	0	0		
		1	5	9	2			<i>subtract leading digit</i>
		*	1	0				<i>multiply by radix</i>
		1	5	9	2	0		
					+ 6			<i>add new trailing digit</i>
		1	5	9	2	6		<i>new value</i>



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp substring search example

First R entries: Use Horner's rule.

Remaining entries: Use rolling hash (and % to avoid overflow).

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
	3	1	4	1	5	9	2	6	5	3	5	8	9	7	9	3	
0	3	%	997	=	3												
1	3	1	%	997	=	(3*10 + 1)	%	997	=	31							
2	3	1	4	%	997	=	(31*10 + 4)	%	997	=	314						
3	3	1	4	1	%	997	=	(314*10 + 1)	%	997	=	150					
4	3	1	4	1	5	%	997	=	(150*10 + 5)	%	997	=	508	RM	R		
5	1	4	1	5	9	%	997	=	((508 + 3*(997 - 30)) * 10 + 9)	%	997	=	201				
6	4	1	5	9	2	%	997	=	((201 + 1*(997 - 30)) * 10 + 2)	%	997	=	715				
7	1	5	9	2	6	%	997	=	((715 + 4*(997 - 30)) * 10 + 6)	%	997	=	971				
8	5	9	2	6	5	%	997	=	((971 + 1*(997 - 30)) * 10 + 5)	%	997	=	442				
9	9	2	6	5	3	%	997	=	((442 + 5*(997 - 30)) * 10 + 3)	%	997	=	929		match		
10	← return i-M+1 = 6	2	6	5	3	5	%	997	=	((929 + 9*(997 - 30)) * 10 + 5)	%	997	=	613			

-30 (mod 997) = 997 - 30 10000 (mod 997) = 30



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp: Java implementation

```
public class RabinKarp
{
    private long patHash;      // pattern hash value
    private int M;              // pattern length
    private long Q;              // modulus
    private int R;              // radix
    private long RM1;            //  $R^{M-1} \% Q$ 

    public RabinKarp(String pat) {
        M = pat.length();
        R = 256;
        Q = LongRandomPrime();           ← a large prime (but avoid overflow)

        RM1 = 1;
        for (int i = 1; i <= M-1; i++)
            RM1 = (R * RM1) % Q;
        patHash = hash(pat, M);
    }

    private long hash(String key, int M)
    { /* as before */ }

    public int search(String txt)
    { /* see next slide */ }
}
```

a large prime
(but avoid overflow)

← precompute $R^{M-1} \pmod{Q}$



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp: Java implementation (continued)

Monte Carlo version. Return match if hash match.

```
public int search(String txt)
{
    int N = txt.length();
    int txtHash = hash(txt, M);
    if (patHash == txtHash) return 0;
    for (int i = M; i < N; i++)
    {
        txtHash = (txtHash + Q - RM*txt.charAt(i-M) % Q) % Q;
        txtHash = (txtHash*R + txt.charAt(i)) % Q;
        if (patHash == txtHash) return i - M + 1;
    }
    return N;
}
```

check for hash collision
using rolling hash function

Las Vegas version. Check for substring match if hash match;
continue search if false collision.



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp analysis

Theory. If Q is a sufficiently large random prime (about $M N^2$), then the probability of a false collision is about $1 / N$.

Practice. Choose Q to be a large prime (but not so large to cause overflow). Under reasonable assumptions, probability of a collision is about $1 / Q$.

Monte Carlo version.

- Always runs in linear time.
- Extremely likely to return correct answer (but not always!).



Las Vegas version.

- Always returns correct answer.
- Extremely likely to run in linear time (but worst case is $M N$).



ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Rabin-Karp fingerprint search

Advantages.

- Extends to 2d patterns.
- Extends to finding multiple patterns.

Disadvantages.

- Arithmetic ops slower than char compares.
- Las Vegas version requires backup.
- Poor worst-case guarantee.

Q. How would you extend Rabin-Karp to efficiently search for any one of P possible patterns in a text of length N ?





ACM-ICPC

ACM-ICPC Thailand Central Group B Programming Contest 2016
รอบภาคกลางเขต 2 ครอบคลุมมหาวิทยาลัยในภาคกลาง (ไม่รวมกรุงเทพฯ) และภาคตะวันตก

Substring search cost summary

Cost of searching for an M -character pattern in an N -character text.

algorithm	version	operation count		backup in input?	correct?	extra space
		guarantee	typical			
brute force	—	MN	$1.1 N$	yes	yes	1
Knuth-Morris-Pratt	<i>full DFA</i> (Algorithm 5.6)	$2N$	$1.1 N$	no	yes	MR
	<i>mismatch transitions only</i>	$3N$	$1.1 N$	no	yes	M
Boyer-Moore	<i>full algorithm</i>	$3N$	N/M	yes	yes	R
	<i>mismatched char heuristic only</i> (Algorithm 5.7)	MN	N/M	yes	yes	R
Rabin-Karp [†]	<i>Monte Carlo</i> (Algorithm 5.8)	$7N$	$7N$	no	yes^{\dagger}	1
	<i>Las Vegas</i>	$7N^{\dagger}$	$7N$	yes	yes	1

[†] probabilistic guarantee, with uniform hash function