

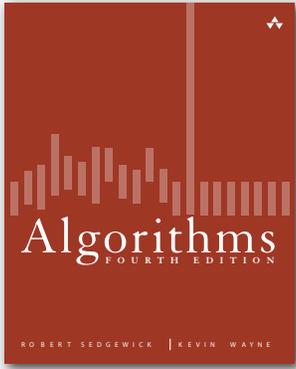
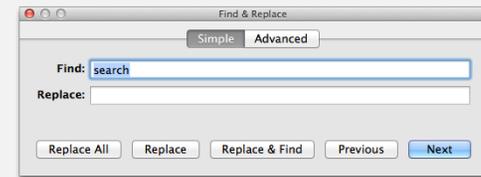
5.3 SUBSTRING SEARCH

Substring search

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

typically $N \gg M$

pattern → N E E D L E
text → I N A H A Y S T A C K N E E D L E I N A
↑
match



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

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
text → I N A H A Y S T A C K N E E D L E I N A
↑
match

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



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
text → I N A H A Y S T A C K N E E D L E I N A
↑
match

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.



Electronic surveillance.



Need to monitor all internet traffic. (security)

No way! (privacy)



Well, we're mainly interested in "ATTACK AT DAWN"

OK. Build a machine that just looks for that.



"ATTACK AT DAWN" substring search machine found



Screen scraping. Extract relevant data from web page.

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



```

...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledatal1">
<big><b>452.92</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledatal1">
...
    
```

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
    
```

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

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							← pat
1	0	1	A	B	R	A							← mismatches
2	1	3	A	B	R	A							← mismatches
3	0	3	A	B	R	A							← mismatches
4	1	5	A	B	R	A							← mismatches
5	0	5	A	B	R	A							← mismatches
6	4	10	A	B	R	A							← match

entries in black match the text
entries in red are mismatches
entries in gray are for reference only
 return i when j is M

9

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;
    }
    return N;
}
  
```

← index in text where pattern starts
 ← not found

10

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 A B											
0	4	4	A	A	A	A	B							← pat
1	4	5	A	A	A	A	B							← mismatches
2	4	6	A	A	A	A	B							← mismatches
3	4	7	A	A	A	A	B							← mismatches
4	4	8	A	A	A	A	B							← mismatches
5	5	10	A	A	A	A	B							← match

Worst case. $\sim MN$ char compares.

11

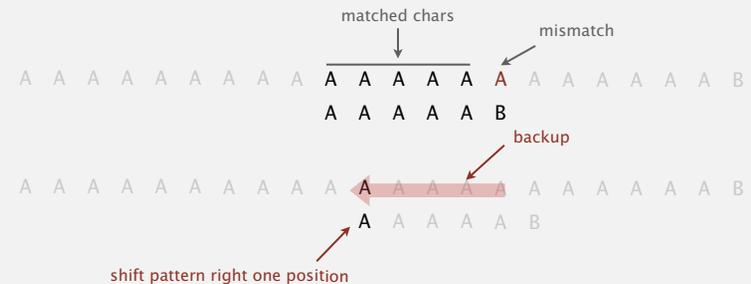
Backup

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.

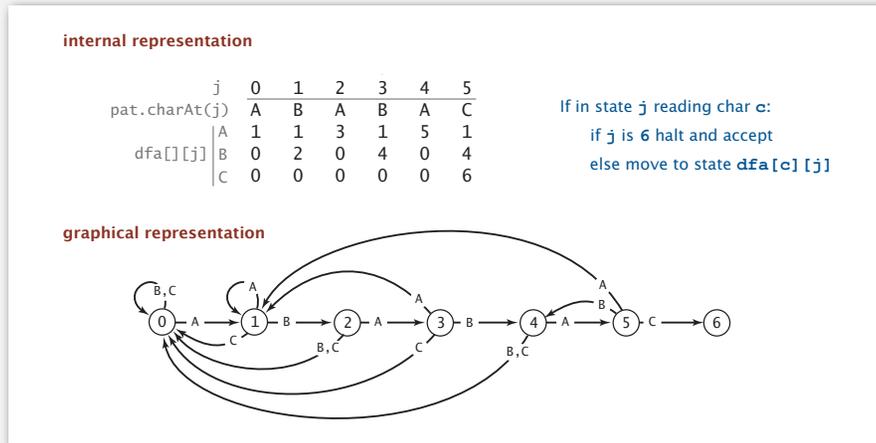
Approach 2. Stay tuned.

12

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.



17

DFA simulation demo

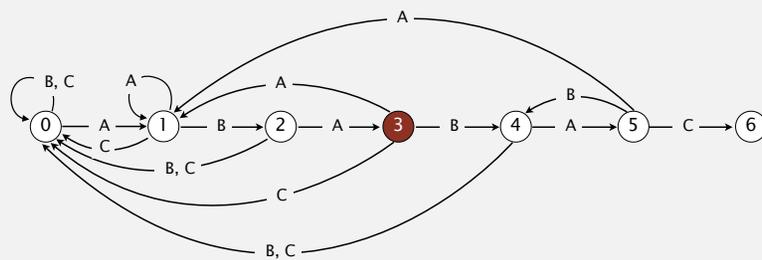
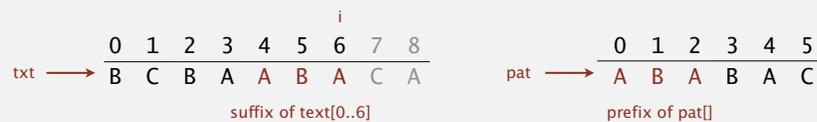
Interpretation of Knuth-Morris-Pratt DFA

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

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

length of longest prefix of pat[]
that is a suffix of txt[0..i]

Ex. DFA is in state 3 after reading in $txt[0..6]$.



19

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];
    if (j == M) return i - M;
    else return N;
}
```

no backup

Running time.

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

18

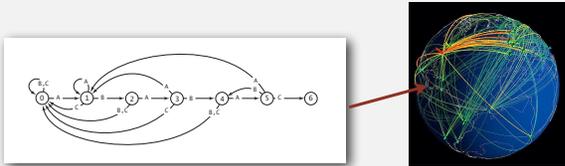
20

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];
    if (j == M) return i - M;
    else return NOT_FOUND;
}
```

no backup



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					



How to build DFA from pattern?

Match transition. If in state j and next char $c == pat.charAt(j)$, 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
dfa[][j]	A	1	3		5	
	B		2	4		
	C					6



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.

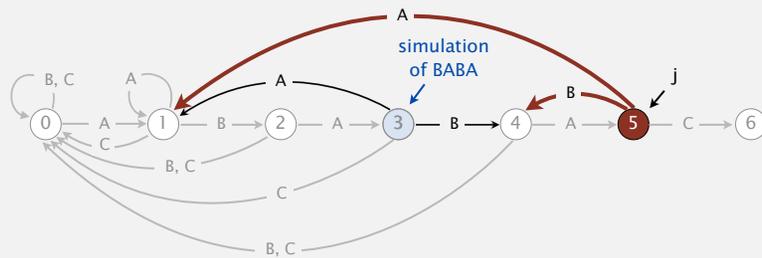
still under construction (!)

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

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

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

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



25

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. Takes only constant time if we maintain state X .

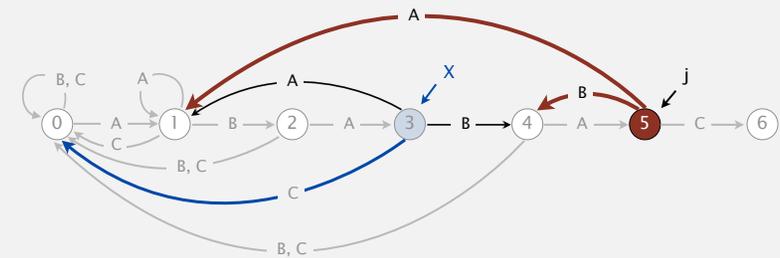
Ex. $\text{dfa}['A'][5] = 1$; $\text{dfa}['B'][5] = 4$; $X = 0$

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



26

Knuth-Morris-Pratt DFA construction (in linear time) demo

Constructing the DFA for KMP substring search: Java implementation

For each state j :

- Copy $\text{dfa}[c][X]$ to $\text{dfa}[c][j]$ for mismatch case.
- Set $\text{dfa}[\text{pat.charAt}(j)][j]$ 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];
            dfa[pat.charAt(j)][j] = j+1;
            X = dfa[pat.charAt(j)][X];
        }
    }
}
```

← copy mismatch cases
← set match case
← update restart state

Running time. M character accesses (but space proportional to RM).

27

28

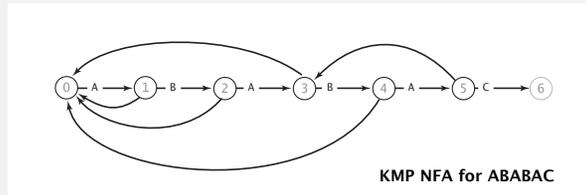
KMP substring search analysis

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 RM .

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



29

Knuth-Morris-Pratt: brief history

- Independently discovered by two theoreticians and a hacker.
 - Knuth: inspired by esoteric theorem, discovered linear-time 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

30

Knuth-Morris-Pratt application

A string s is a **cyclic rotation** of t if s and t have the same length and s is a suffix of t followed by a prefix of t .

yes	yes	no
ROTATEDSTRING	ABABABBABBABA	ROTATEDSTRING
STRINGROTATED	BABBABBABAABA	GNIRTSDETATOR

Problem. Given two strings s and t , design a linear-time algorithm that determines if s is a cyclic rotation of t .

Solution.

- Check that s and t are the same length.
- Search for s in $t + t$ using KMP.

$t + t \rightarrow$ STRINGROTATEDSTRINGROTATED
 $s \rightarrow$ ROTATEDSTRING

31

- brute force
- Knuth-Morris-Pratt
- Boyer-Moore
- Rabin-Karp



Robert Boyer



J. Strother Moore

32

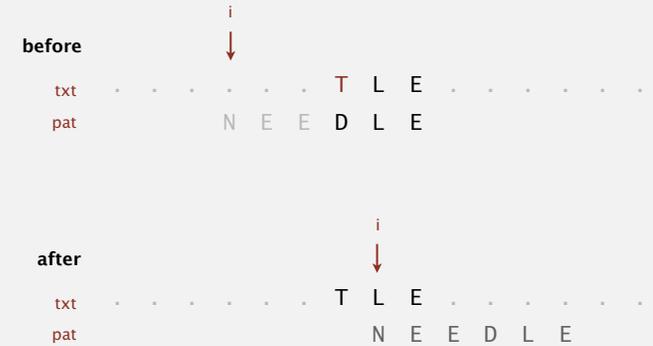
Intuition.

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



Q. How much to skip?

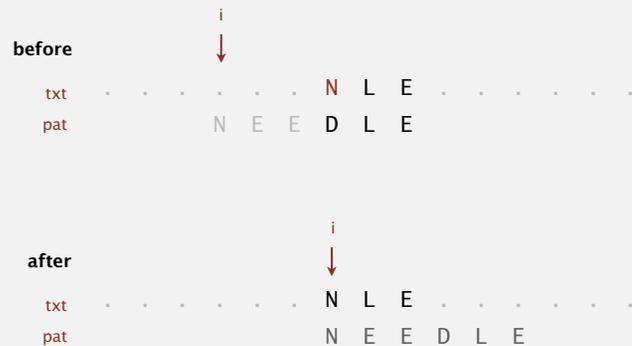
Case 1. Mismatch character not in pattern.



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

Q. How much to skip?

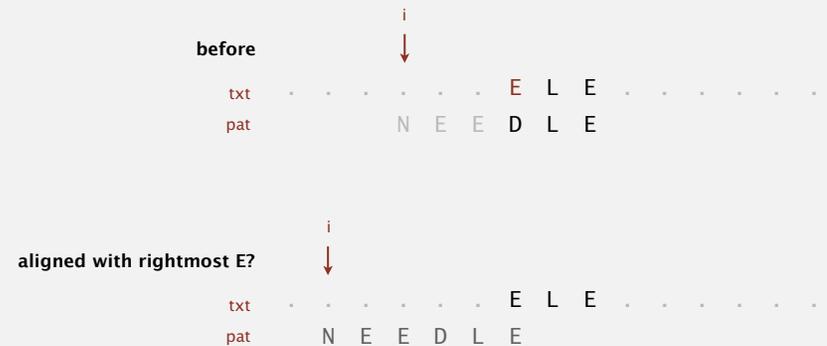
Case 2a. Mismatch character in pattern.



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

Q. How much to skip?

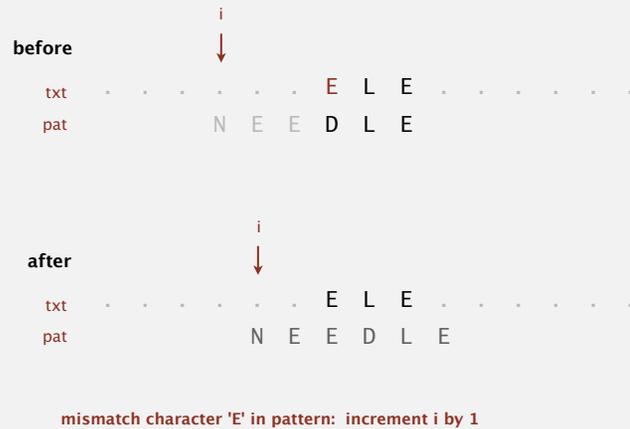
Case 2b. Mismatch character in pattern (but heuristic no help).



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

Q. How much to skip?

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



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	5	5
...							-1
L	-1	-1	-1	-1	4	4	4
M	-1	-1	-1	-1	-1	-1	-1
N	-1	0	0	0	0	0	0
...							-1

Boyer-Moore skip table computation

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;
    }
    return N;
}
```

Annotations:
 - Arrow to `right[txt.charAt(i+j)]`: compute skip value
 - Arrow to `Math.max(1, j - right[txt.charAt(i+j)])`: in case other term is nonpositive
 - Arrow to `if (skip == 0) return i;`: match

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 $\sim MN$.

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						
2	1			A	B	B	B					
3	1				A	B	B	B				
4	1					A	B	B	B			
5	1						A	B	B	B		

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

Rabin-Karp substrng search example

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															
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															
10	← return i-M+1 = 6 2 6 5 3 5 % 997 = ((929 + 9*(997 - 30))*10 + 5) % 997 = 613															

45

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 RM; // R^(M-1) % Q

    public RabinKarp(String pat) {
        M = pat.length();
        R = 256;
        Q = longRandomPrime();

        RM = 1;
        for (int i = 1; i <= M-1; i++)
            RM = (R * RM) % 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 $R^{M-1} \pmod{Q}$

46

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.

47

Rabin-Karp analysis

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

Practice. Choose Q to be a large prime (but not so large as 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 MN).



48

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 ?



49

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	$1.1N$	yes	yes	1
Knuth-Morris-Pratt	full DFA (Algorithm 5.6)	$2N$	$1.1N$	no	yes	MR
	mismatch transitions only	$3N$	$1.1N$	no	yes	M
Boyer-Moore	full algorithm	$3N$	N/M	yes	yes	R
	mismatched char heuristic only (Algorithm 5.7)	MN	N/M	yes	yes	R
Rabin-Karp [†]	Monte Carlo (Algorithm 5.8)	$7N$	$7N$	no	yes [†]	1
	Las Vegas	$7N$ [†]	$7N$	yes	yes	1

[†] probabilistic guarantee, with uniform hash function

50