http://algs4.cs.princeton.edu

5.3 SUBSTRING SEARCH

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

http://algs4.cs.princeton.edu

5.3 Substring Search

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

Substring search

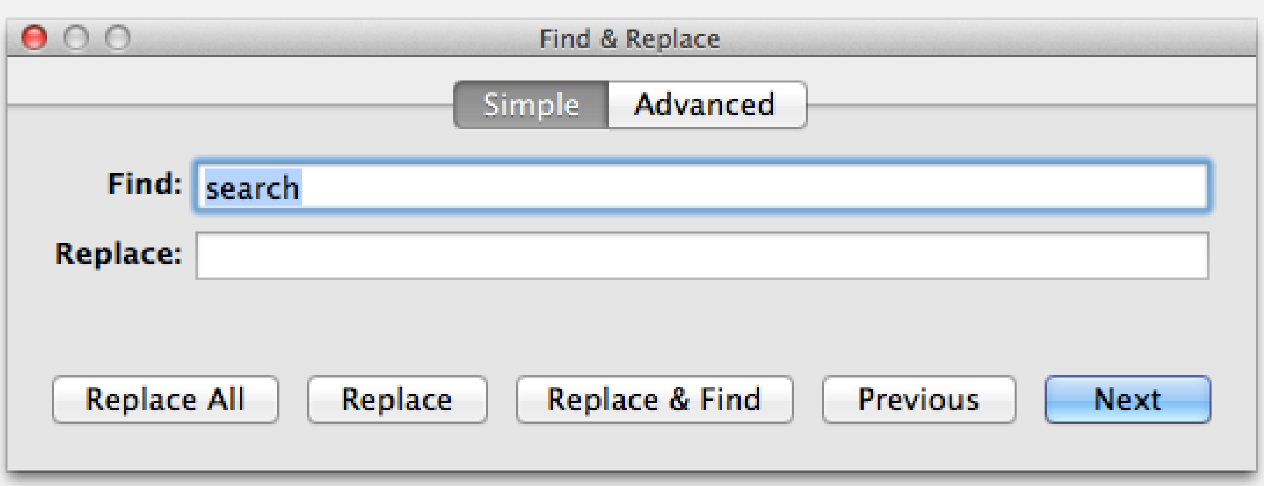
Goal. Find pattern of length M in a text of length N.

typically N >> M

Substring search applications

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

typically N >> M



Substring search applications

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

typically N >> M

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

Substring search applications

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

typically N >> M

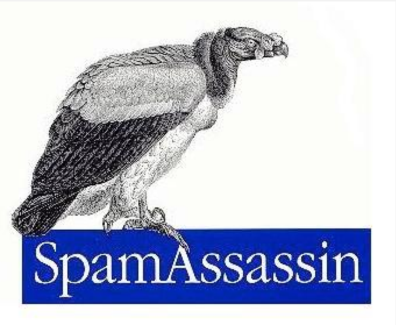
pattern → N E E D L E

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
- L0SE WE1GHT
- herbal Viagra
- There is no catch.
- This is a one-time mailing.
- This message is sent in compliance with spam regulations.



Substring search applications

Electronic surveillance.

Need to monitor all internet traffic. (security)

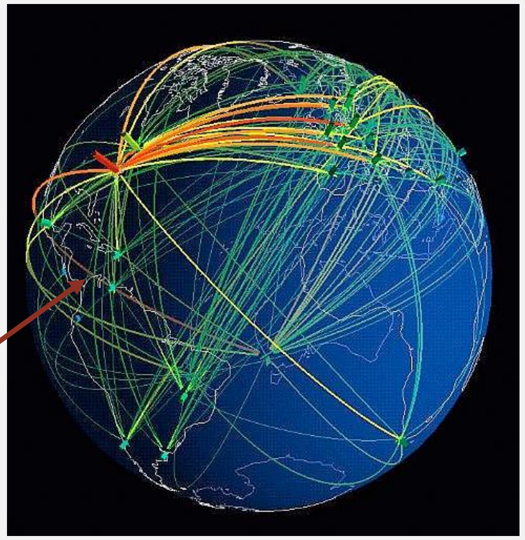
No way! (privacy)

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



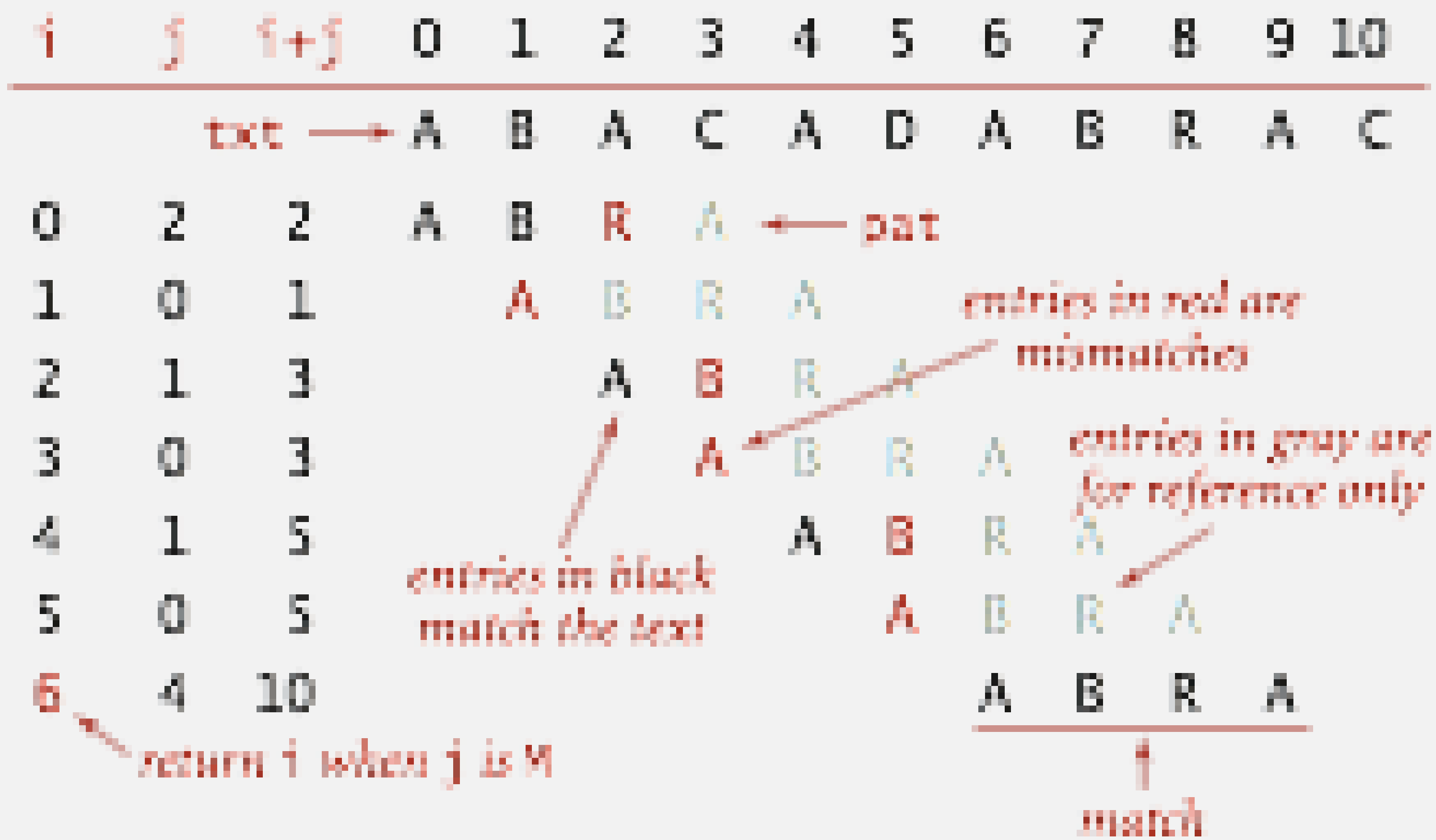
http://algs4.cs.princeton.edu

5.3 Substring Search

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

Brute-force substring search

Check for pattern starting at each text position.



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

Brute-force substring search: worst case

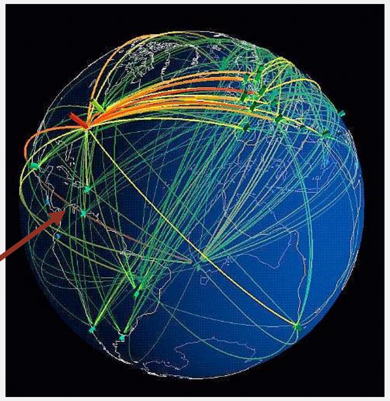
Brute-force algorithm can be slow if text and pattern are repetitive.

Worst case. $\sim M N$ char compares.

Backup

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

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



"ATTACK AT DAWN" substring search machine

found

Brute-force algorithm needs backup for every mismatch.

matched chars

mismatch

A A A A A A A A A A A A A A A A A A A A A B

A A A A A B

backup

A A A A A A A A A A A A A A A A A A A A A B

A A A A A B

shift pattern right one position

Approach 1. Maintain buffer of last M characters.

Approach 2. Stay tuned.

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 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 attack at dawn 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 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.

http://algs4.cs.princeton.edu

5.3 Substring Search

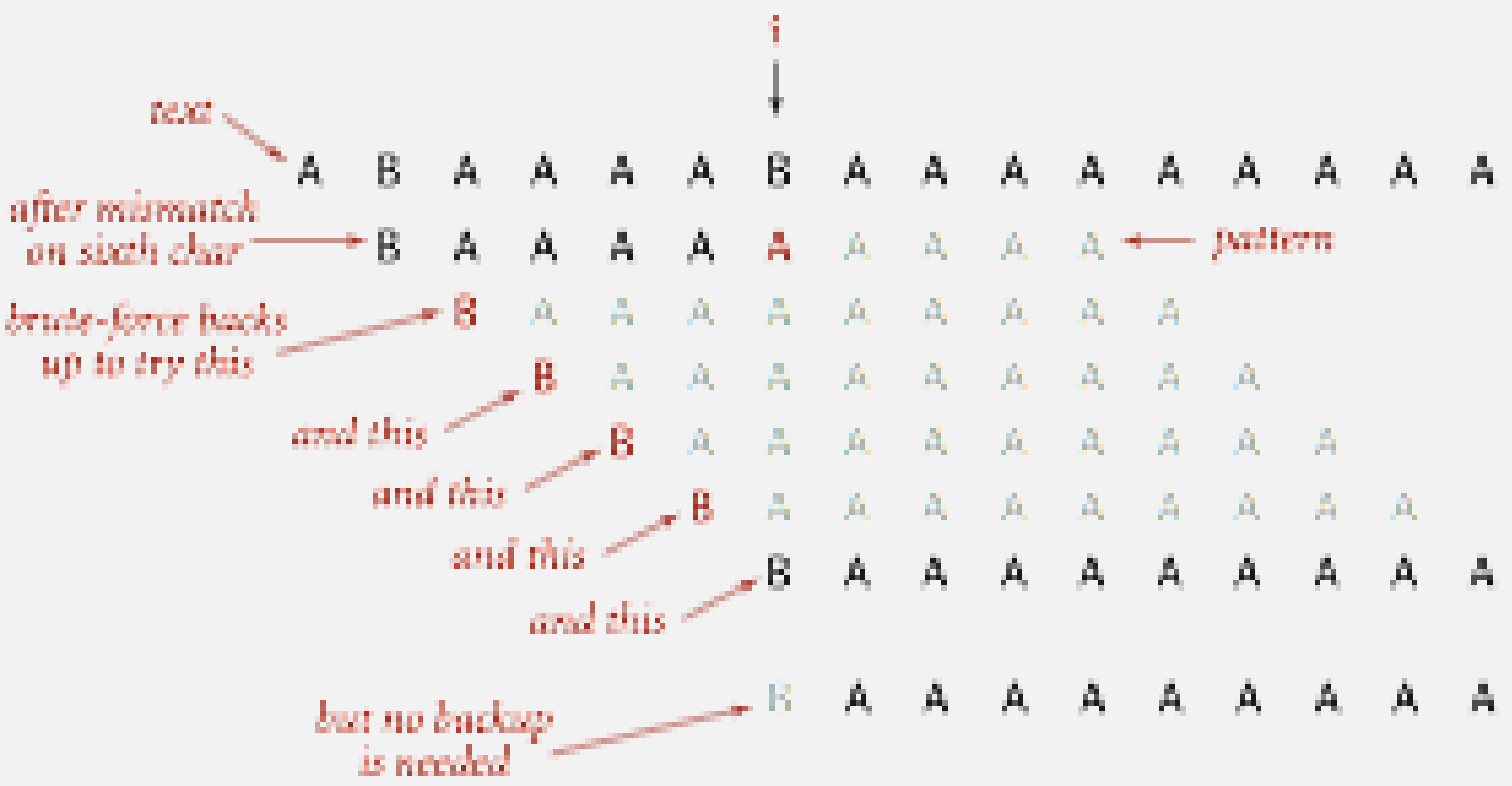
- *introduction*
- *brute force*
- *Knuth-Morris-Pratt*
- *Boyer-Moore*
- *Rabin-Karp*

Knuth-Morris-Pratt substring search

Intuition. Suppose we are searching in text for pattern BAAAAAAAAA.

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

assuming { A, B } alphabet



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

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

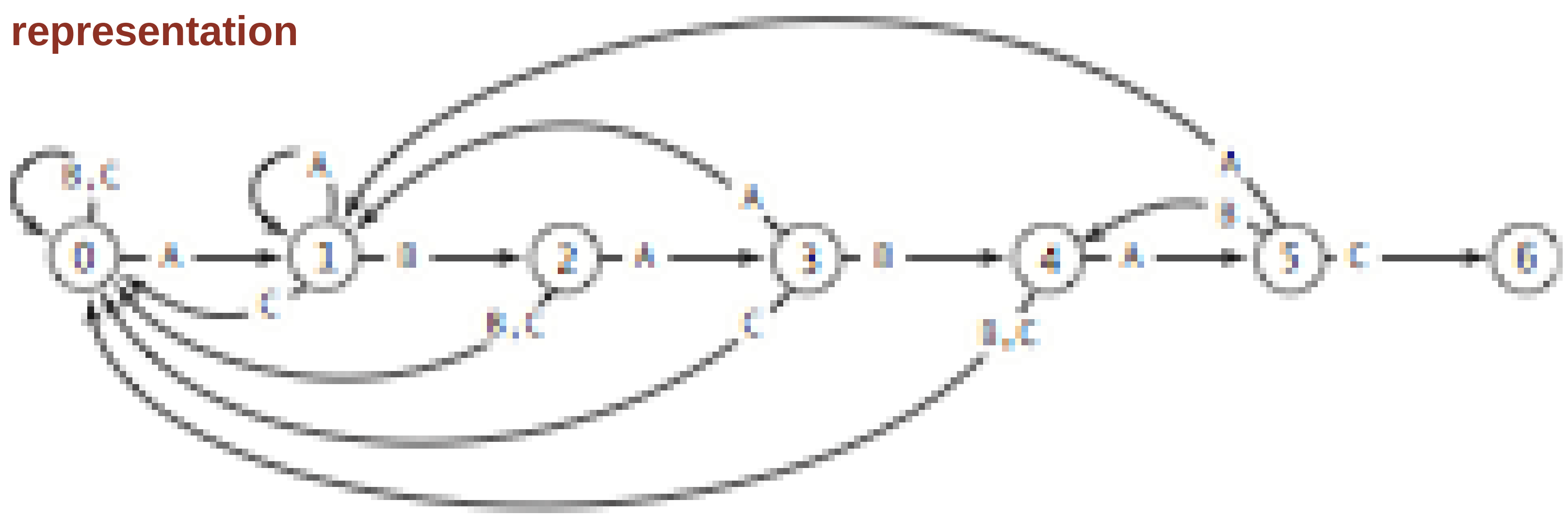
		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

If in state j reading char c:

if j is 6 halt and accept

else move to state dfa[c][j]

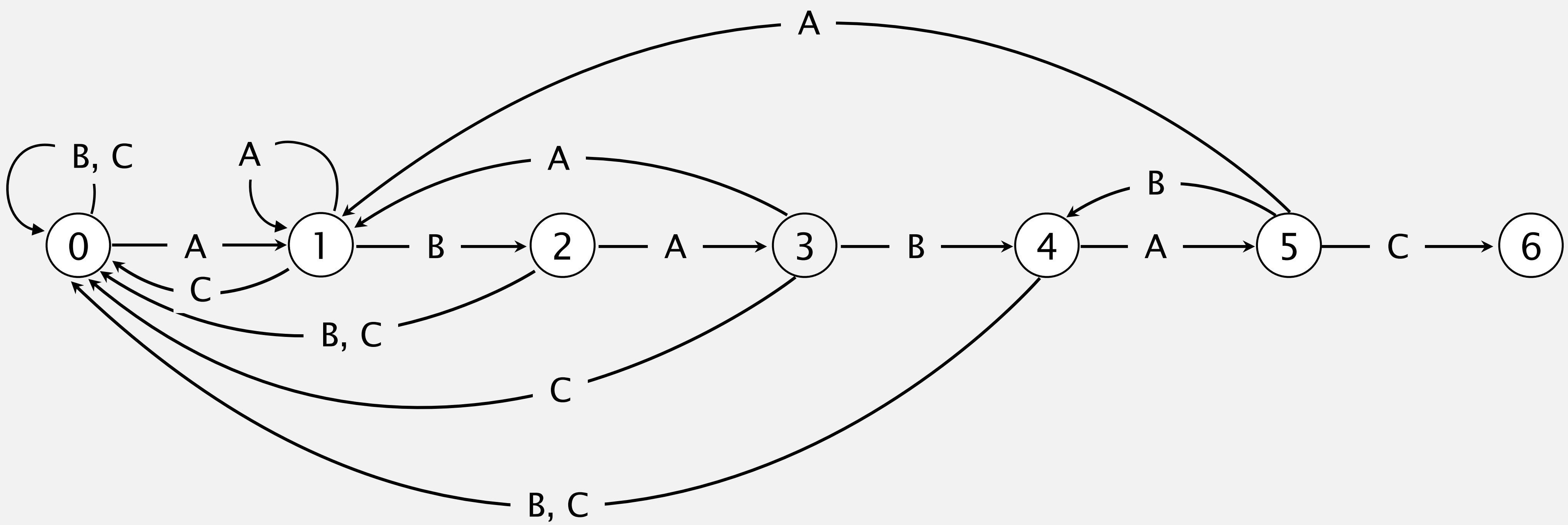
graphical representation



Knuth-Morris-Pratt demo: DFA simulation

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

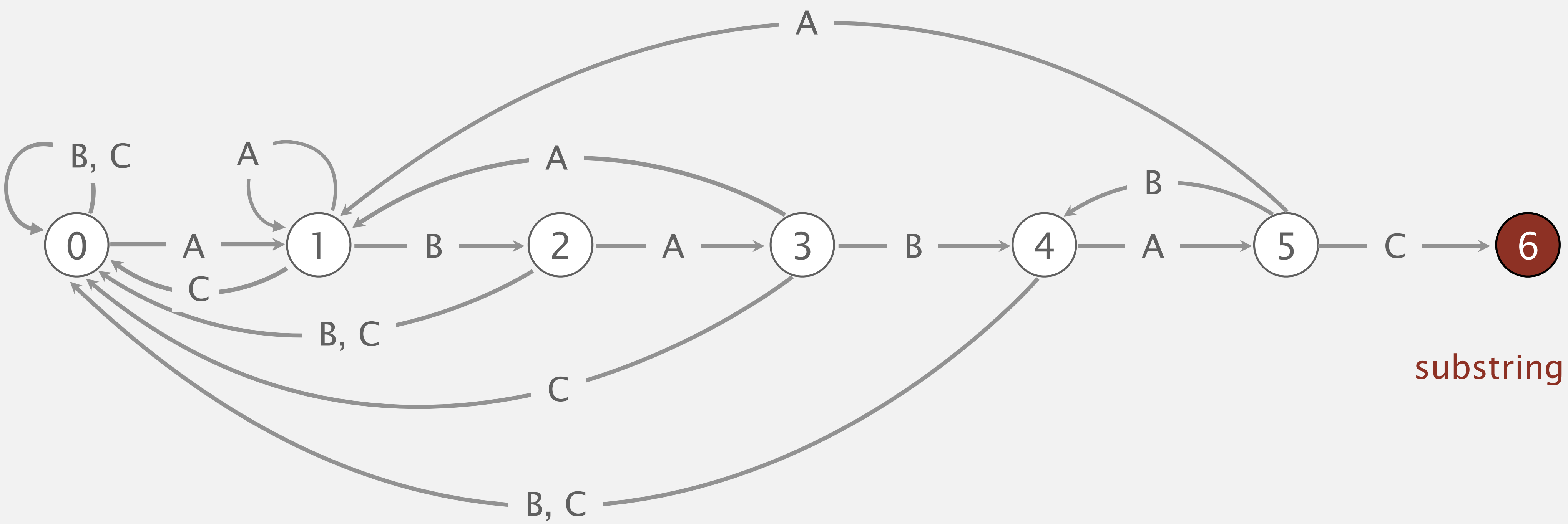
		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



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	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



substring found

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].

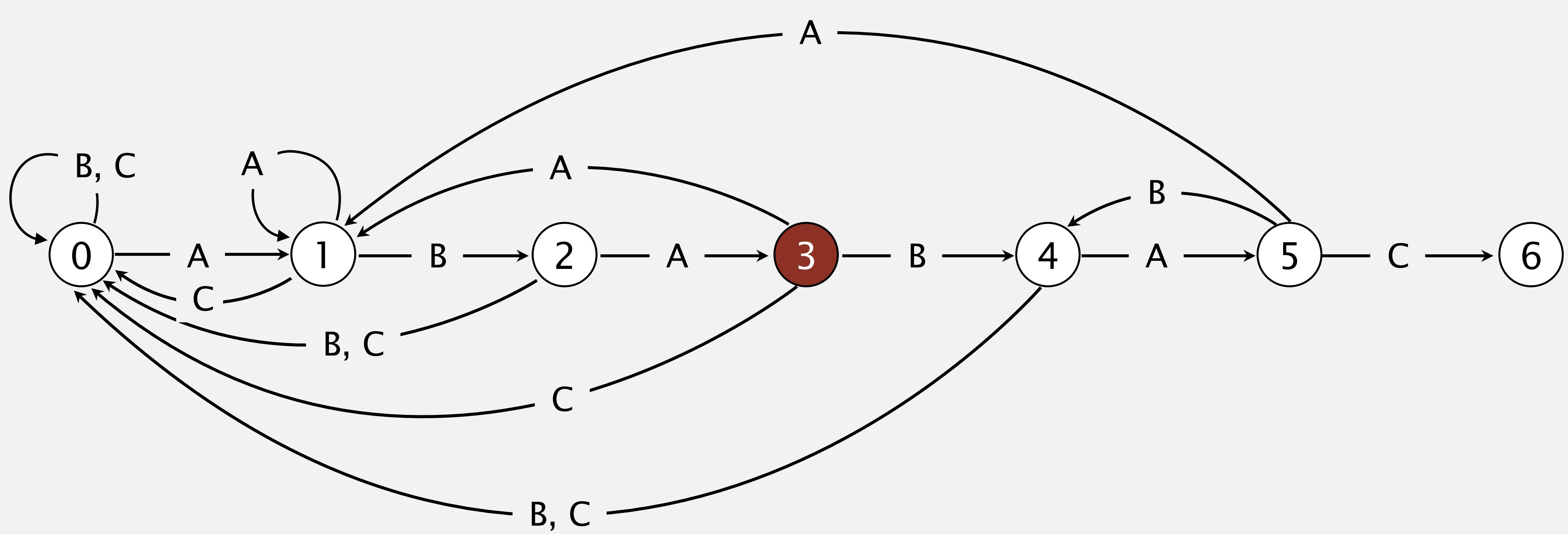
i

	0	1	2	3	4	5	6	7	8
txt	B	C	B	A	A	B	A	C	A

suffix of txt[0..6]

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

prefix of pat[]



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]

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 != pat.charAt(j), then the last j-1 characters of input are pat[1..j-1], followed by c.

To compute dfa[c][j]: Simulate pat[1..j-1] on DFA and take transition c.

Running time. Seems to require j steps.

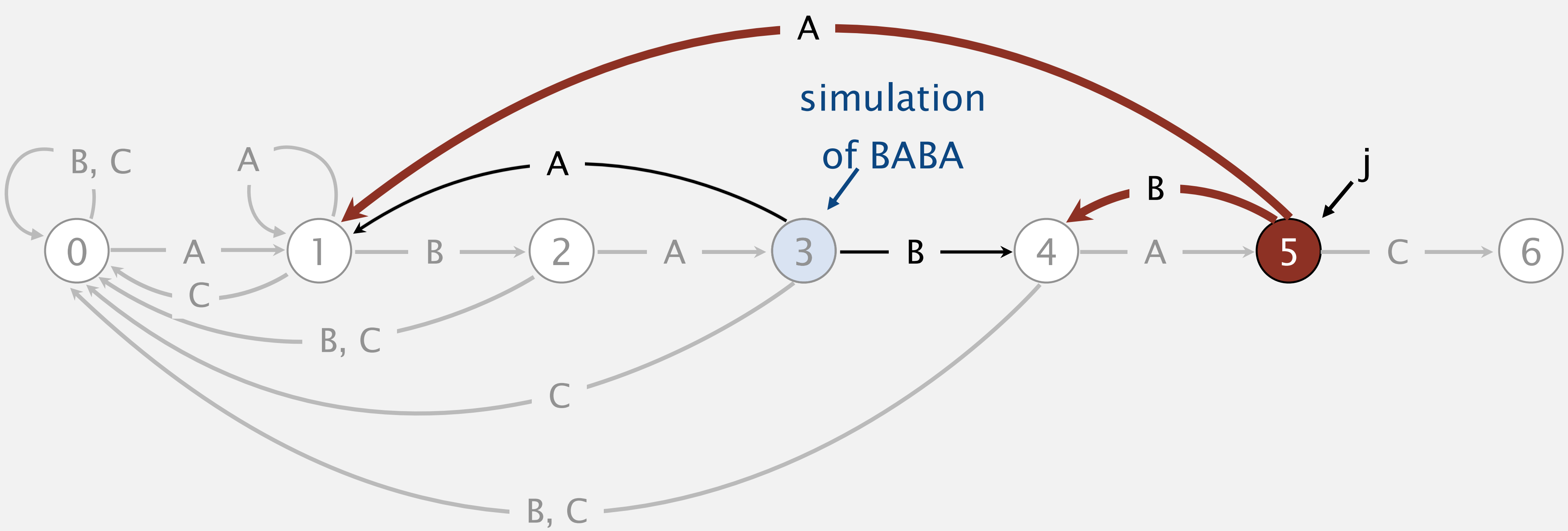
still under construction (!)

Ex. dfa['A'][5] = 1; dfa['B'][5] = 4

simulate BABA;
take transition 'A'
= dfa['A'][3]

simulate BABA;
take transition 'B'
= dfa['B'][3]

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



How to build DFA from pattern?

Mismatch transition. If in state j and next char c != pat.charAt(j), then the last j-1 characters of input are pat[1..j-1], followed by c.

state X

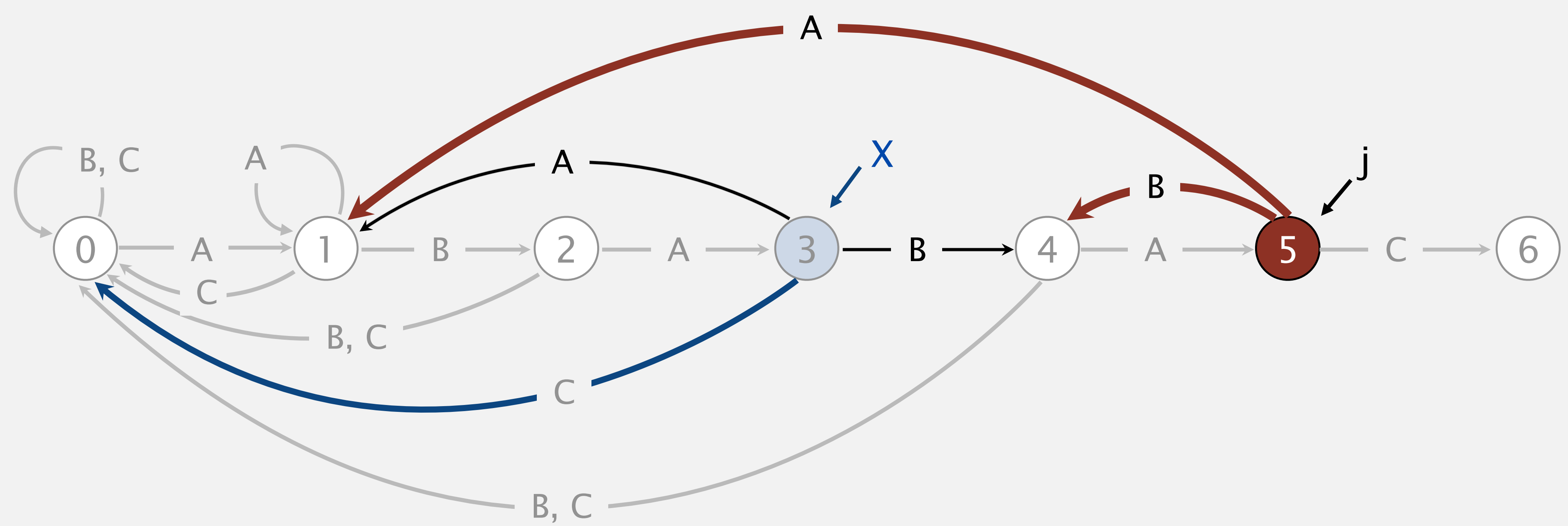
To compute dfa[c][j]: Simulate pat[1..j-1] on DFA and take transition c.

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

Ex. dfa['A'][5] = 1; dfa['B'][5] = 4 X' = 0

dfa['A'][5] = 1;	dfa['B'][5] = 4	X' = 0
from state X, take transition 'A' = dfa['A'][X]	from state X, take transition 'B' = dfa['B'][X]	from state X, take transition 'C' = dfa['C'][X]

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



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 $R\,M$).

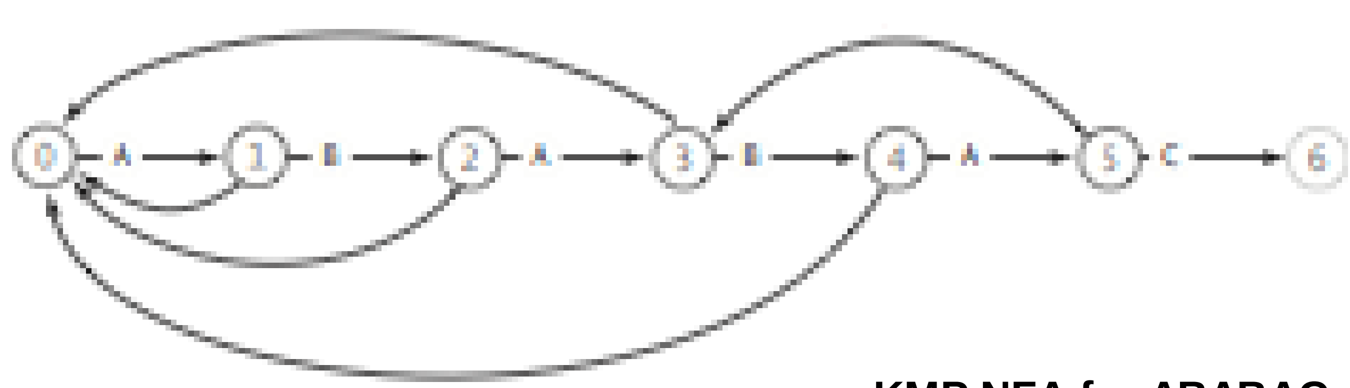
KMP substring search analysis

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 dfa[][] in time and space proportional to $R M$.

Larger alphabets. Improved version of KMP constructs nfa[] in time and space proportional to M.



KMP NFA for ABABAC

http://algs4.cs.princeton.edu

5.3 Substring Search

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

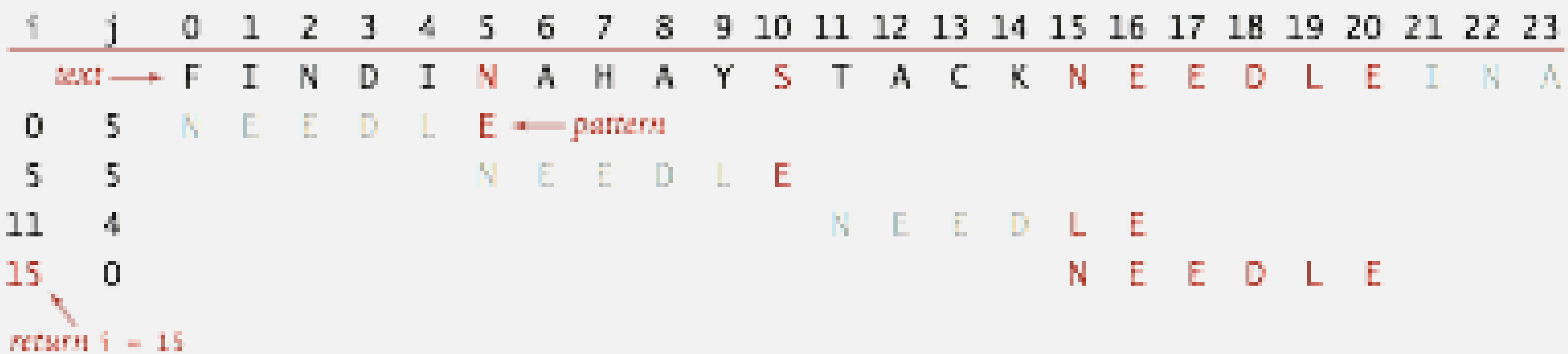
Robert Boyer

J. Strother Moore

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.



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;
```

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

Boyer-Moore skip table computation

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 1. Mismatch character not in pattern.

i   i + j

before

txt . . . . . . T L E . . . . . .

pat N E E D L E

j

i = i + j + 1

after

txt . . . . . . T L E . . . . . .

pat N E E D L E

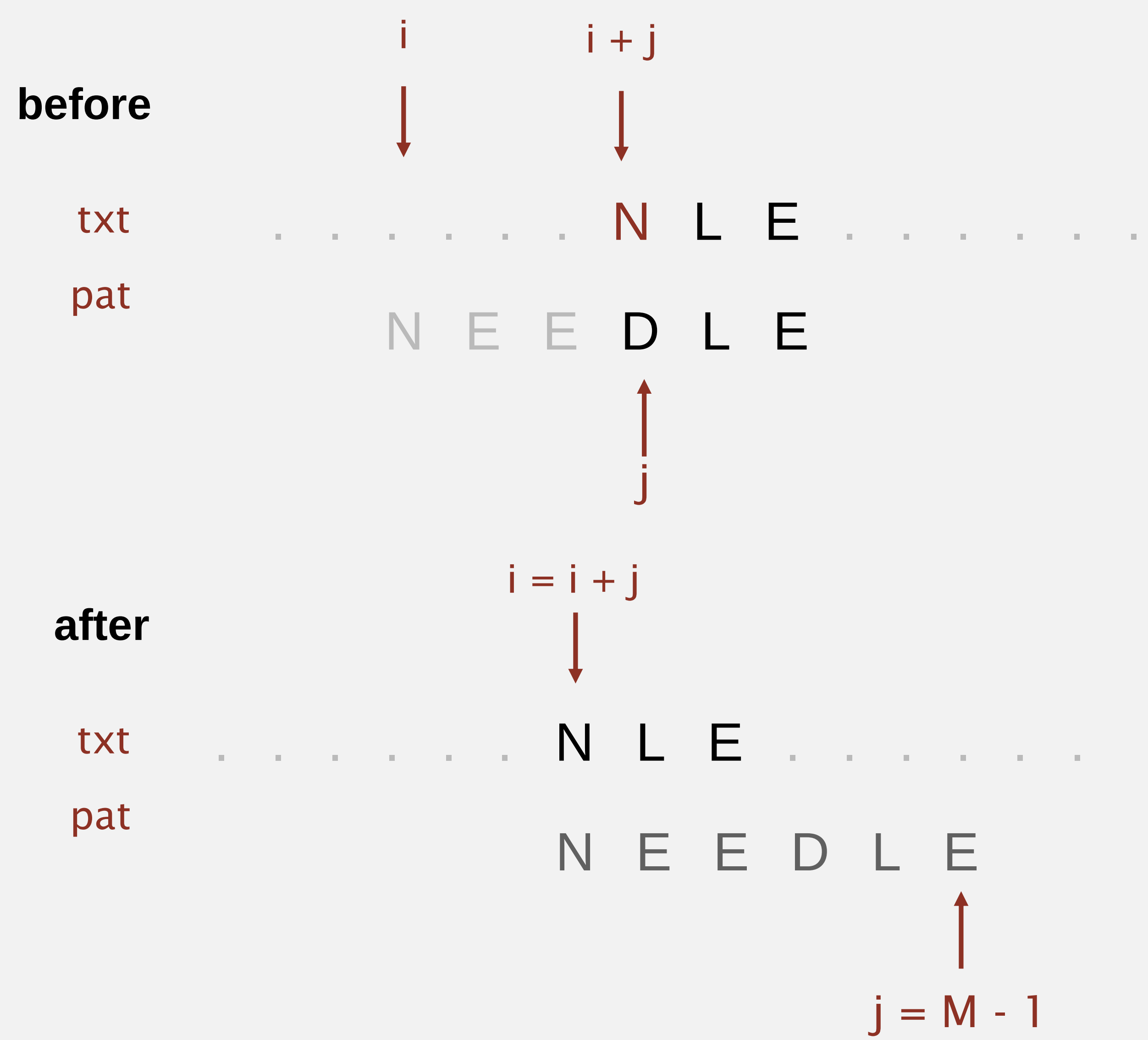
j = M - 1

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

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'

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

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

before

i   i + j

txt . . . . . . E L E . . . . . .

pat N E E D L E

j

aligned with rightmost E?

i = i + j - 5

txt . . . . . . E L E . . . . . .

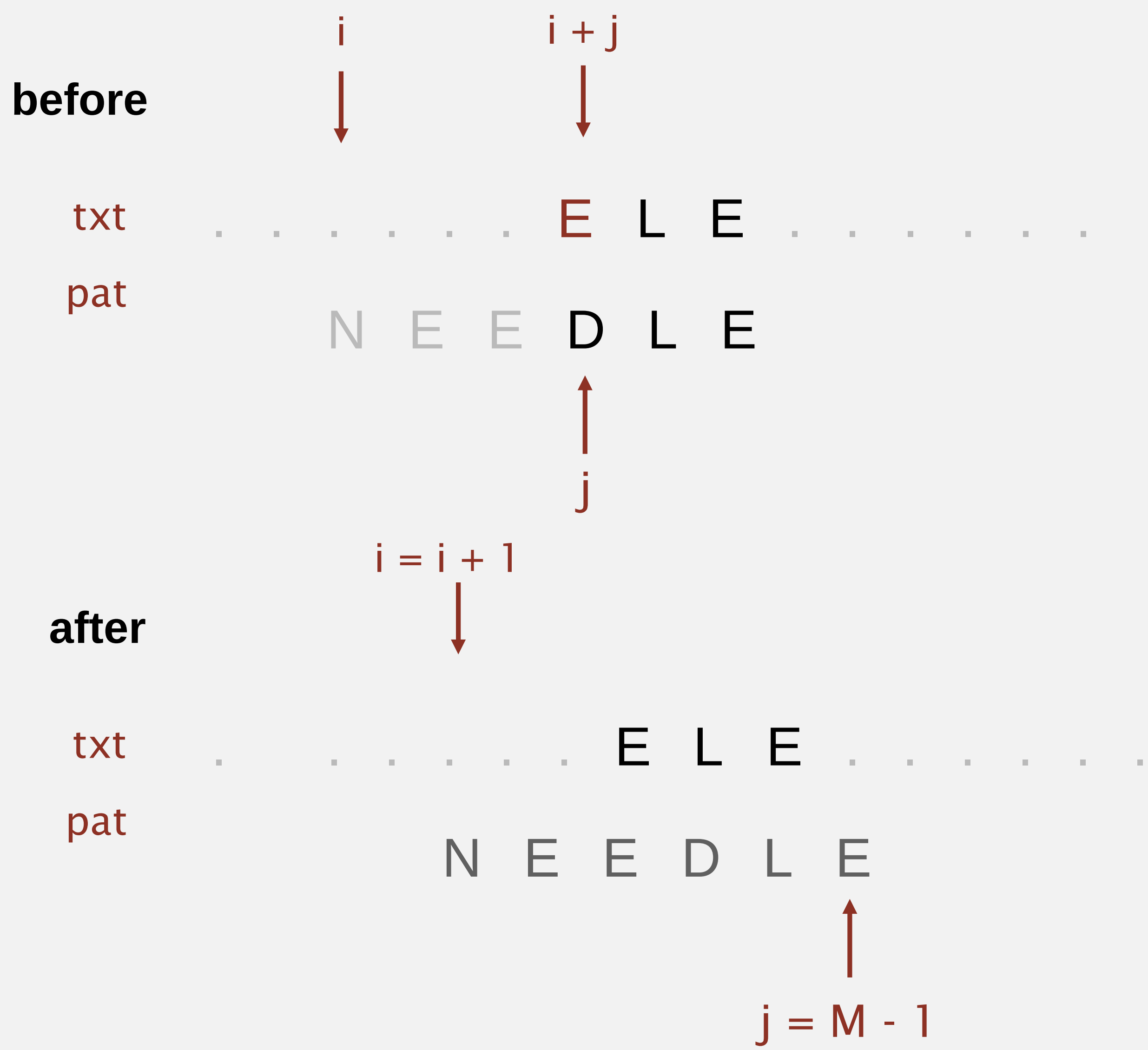
pat N E E D L E

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

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 1

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;
}
```

compute skip value

in case other term is nonpositive

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	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

http://algs4.cs.princeton.edu

5.3 Substring Search

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

Michael Rabin
Dick Karp

Rabin-Karp fingerprint search

Basic idea = modular hashing.

- Compute a hash of pat[0..M-1].
- For each i, compute a hash of 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	% 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 match				

modular hashing with R = 10 and hash(s) = s (mod 997)

Modular arithmetic

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

Ex.

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

10000 mod 997 = 30

1000 mod 997 = 3

$$(a + b) \bmod Q = ((a \bmod Q) + (b \bmod Q)) \bmod Q$$

$$(a * b) \bmod Q = ((a \bmod Q) * (b \bmod Q)) \bmod Q$$

two useful modular arithmetic identities

Efficiently computing the hash function

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

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

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			
2	2	6	5	% 997 = (26*10 + 5) % 997 = 265		
3	2	6	5	3	% 997 = (265*10 + 3) % 997 = 659	
4	2	6	5	3	5	% 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;
}
```

26535 = 2*10000 + 6*1000 + 5*100 + 3*10 + 5

= ((((2) *10 + 6) * 10 + 5) * 10 + 3) * 10 + 5

Efficiently computing the hash function

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

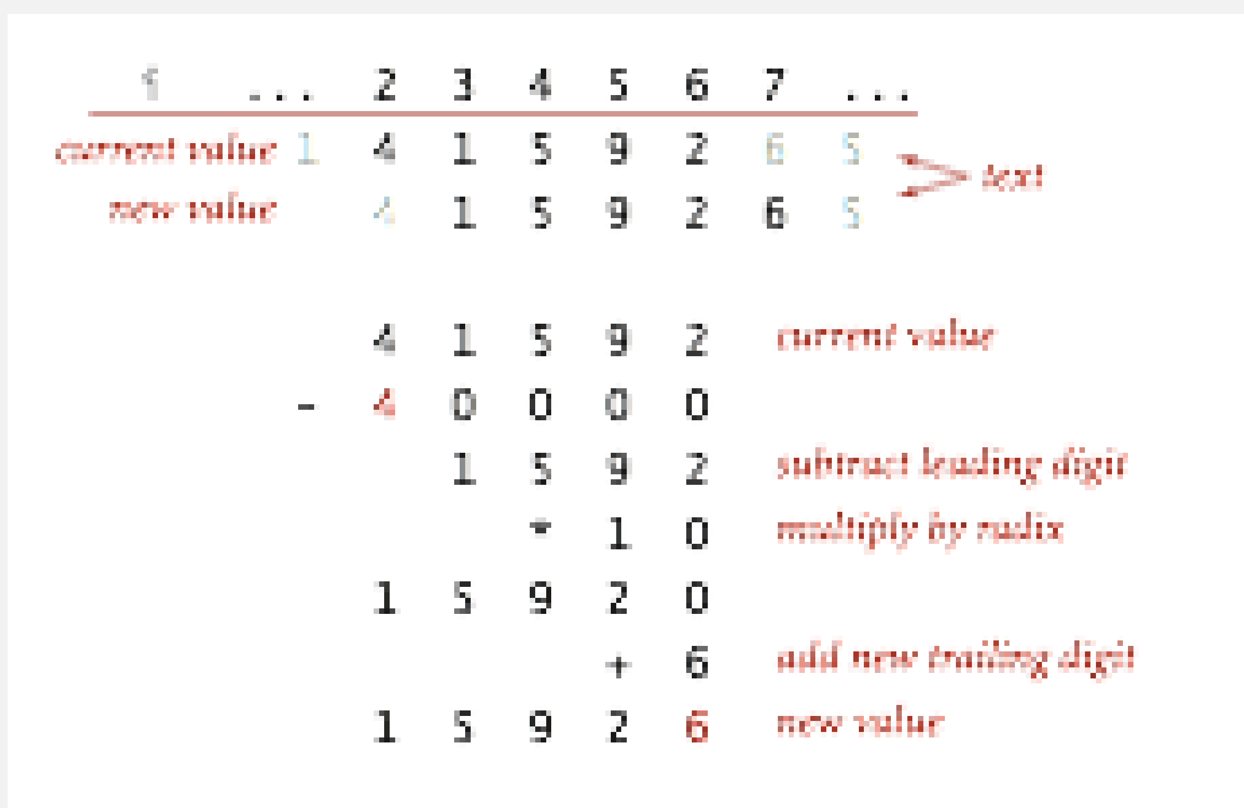
$$x_i \;= t_i\, R^{\,M-1} + t_{i+1}\, R^{\,M-2} + \ldots + t_{i+M-1}\, R^0$$

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

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

$$x_{i+1} \;=\; (\,\underbrace{x_i}_{\text{current value}} \;-\; \underbrace{t_i\, R^{\,M-1}}_{\text{subtract leading digit}}\,)\; \underbrace{R}_{\text{multiply by radix}} \;+\; \underbrace{t_{i+M}}_{\text{add new trailing digit}}$$

(can precompute R^{M-1})



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
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

Q

RM

R

match

Horner's rule

rolling hash

-30 (mod 997) = 997 – 30

10000 (mod 997) = 30

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();

      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 */  }
```

a large prime
(but avoid overflow)

precompute R^{M-1} (mod Q)

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.

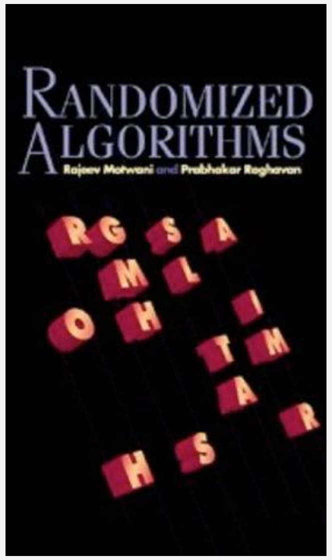
Rabin-Karp analysis

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 $M N$).

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.

Substring search cost summary

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

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

† probabilisitic guarantee, with uniform hash function