Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

5.3 SUBSTRING SEARCH

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

http://algs4.cs.princeton.edu

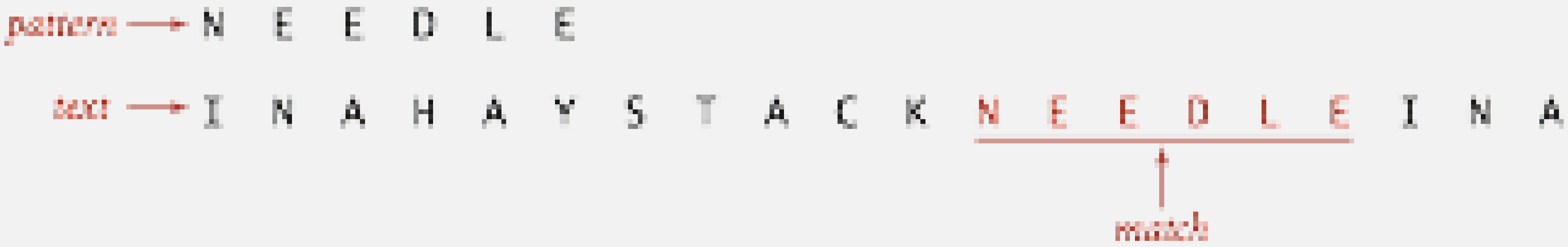
5.3 Substring Search

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

Substring search

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

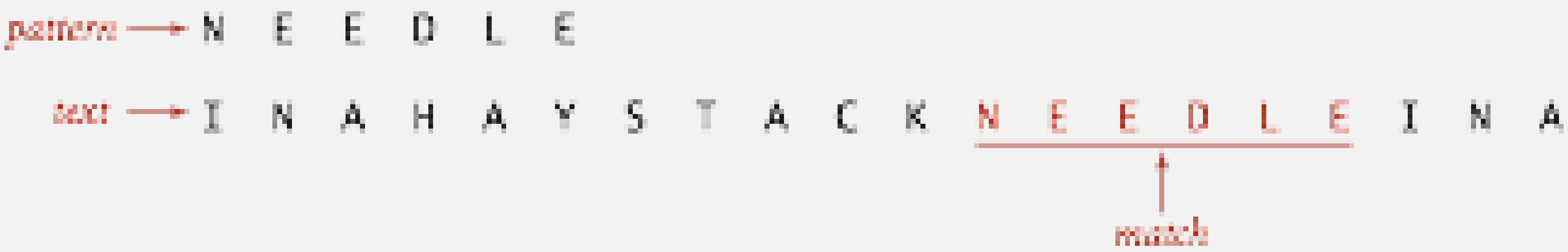
typically N >> M



Substring search applications

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

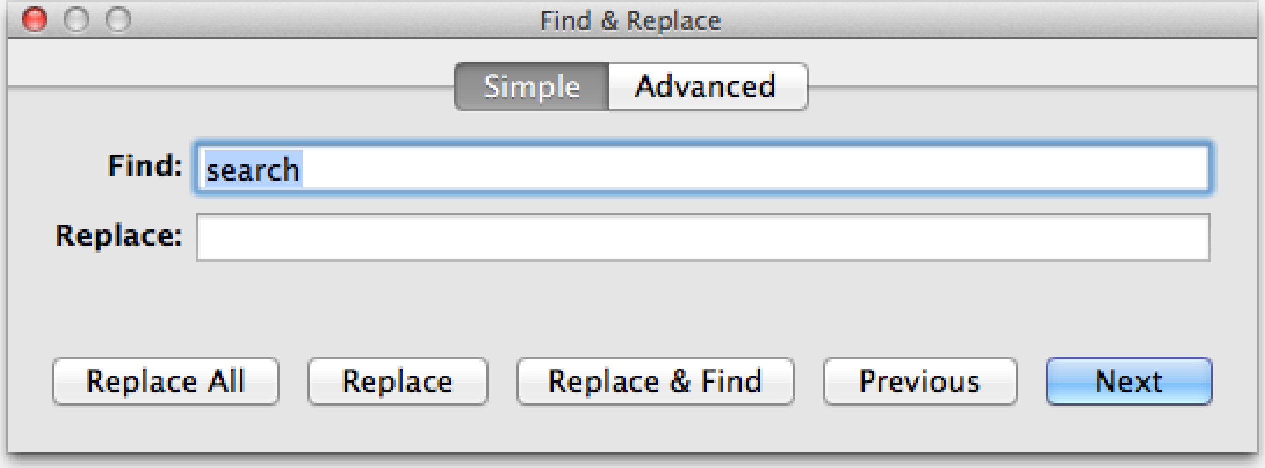
typically N >> M



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



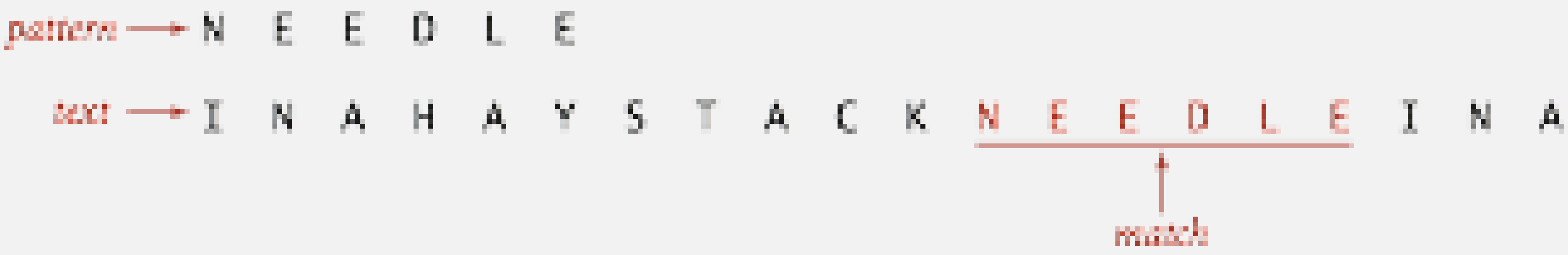
Find: search

Replace:

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

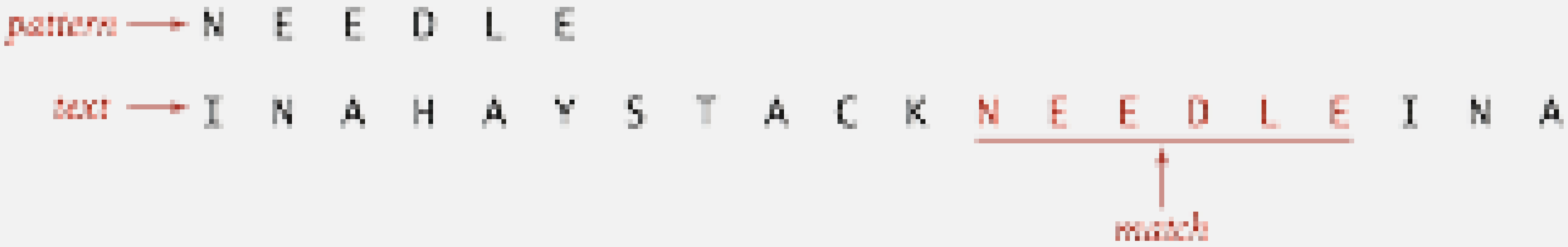
Computer forensics. Search memory or disk for signatures, e.g., all URLs or RSA keys that the user has entered.

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)

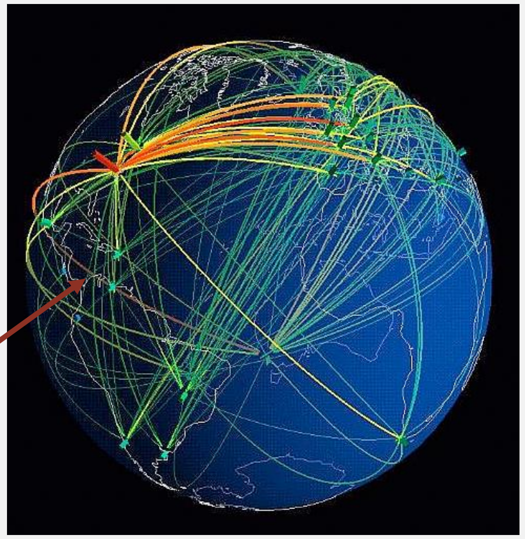
No way! (privacy)

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



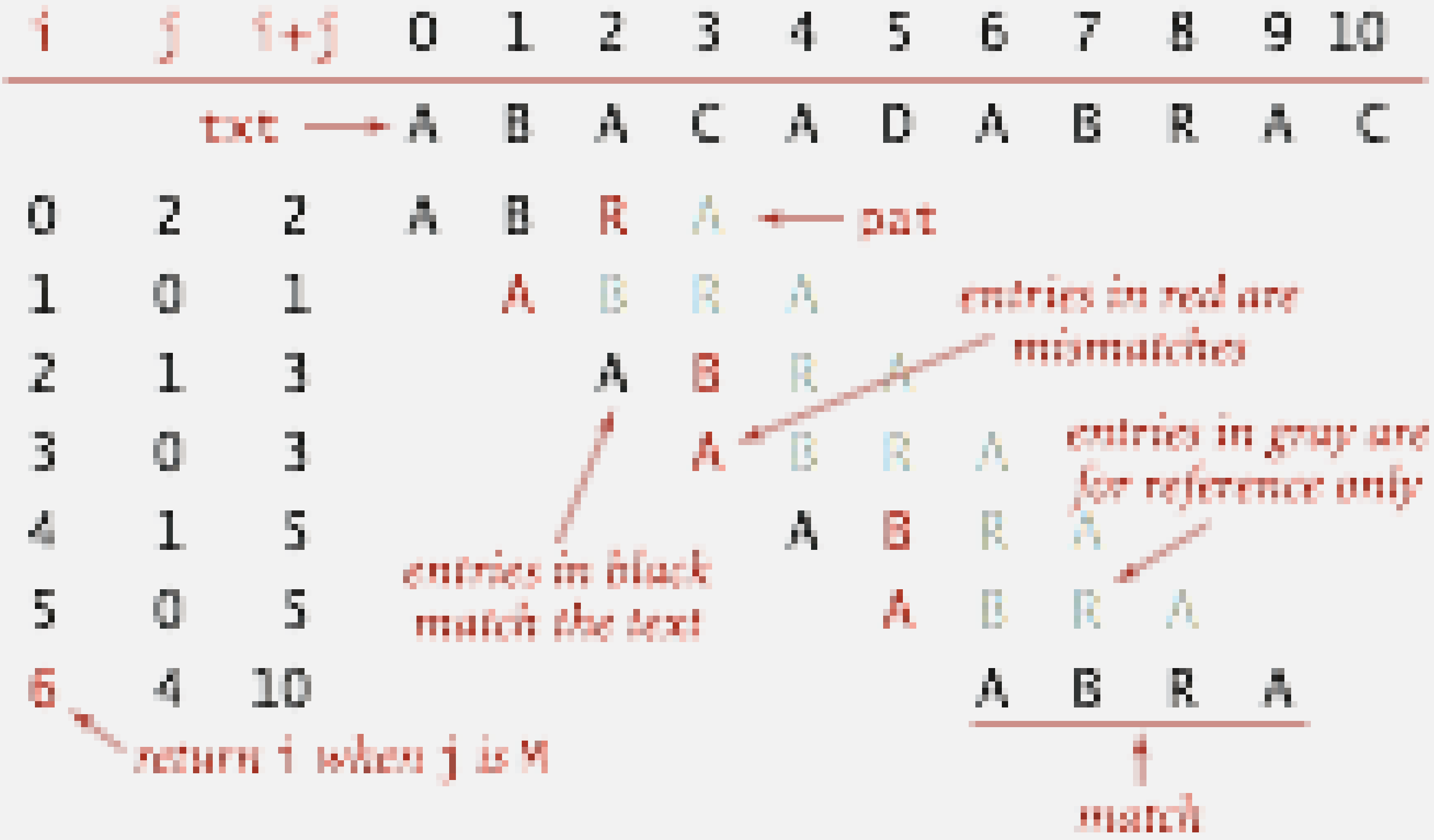
http://algs4.cs.princeton.edu

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.



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	

pat

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

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	

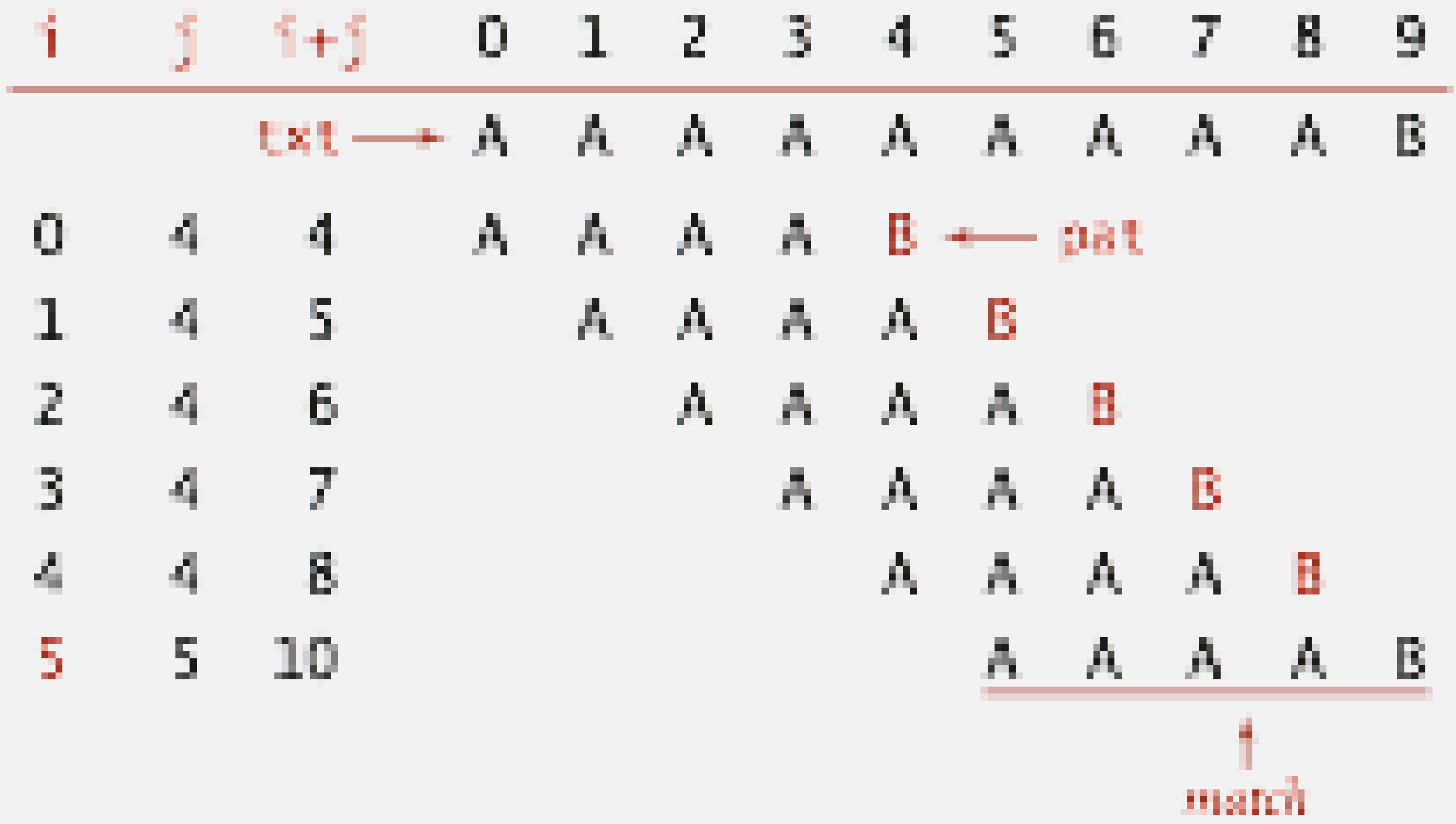
```
public static int search(String pat, String txt)
{
   int M = pat.length();
   int N = txt.length();
   for (int i = 0; i <= N - M; i++)
   {
      int j;
      for (j = 0; j < M; j++)
         if (txt.charAt(i+j) != pat.charAt(j))
            break;
      if (j == M) return i;
   }
   return N;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

← 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.



i	j	i+j	0	1	2	3	4	5	6	7	8	9
			A	A	A	A	A	A	A	A	A	B
0	4	4	A	A	A	A	B					
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

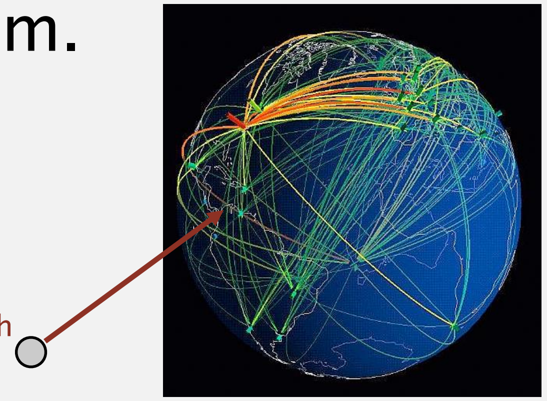
text
pat
match

Worst case. ~ $M\,N$ char compares.

Backup

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

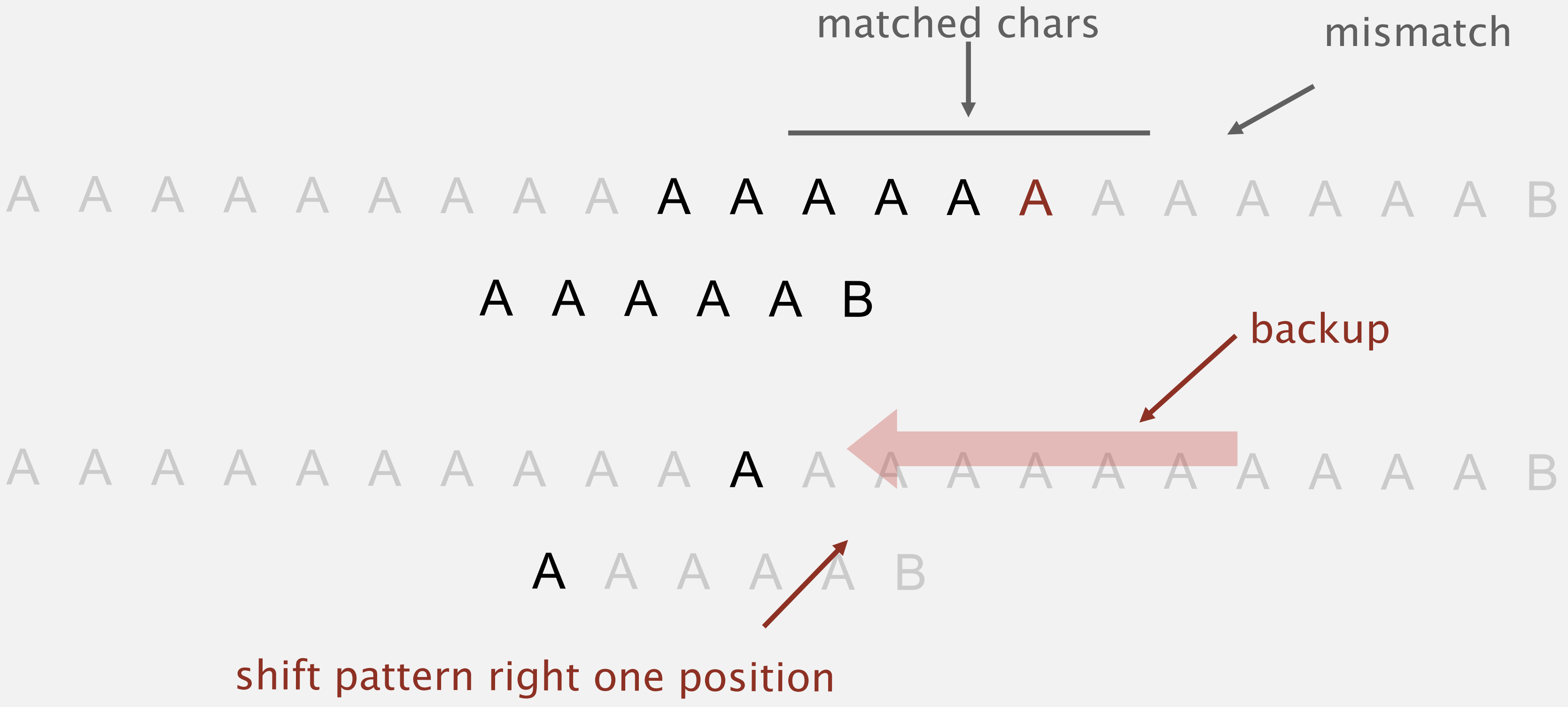
- Treat input as stream of data.
- Abstract model: standard input.



"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.

Algorithms

Robert Sedgewick | Kevin Wayne

http://algs4.cs.princeton.edu

5.3 Substring Search

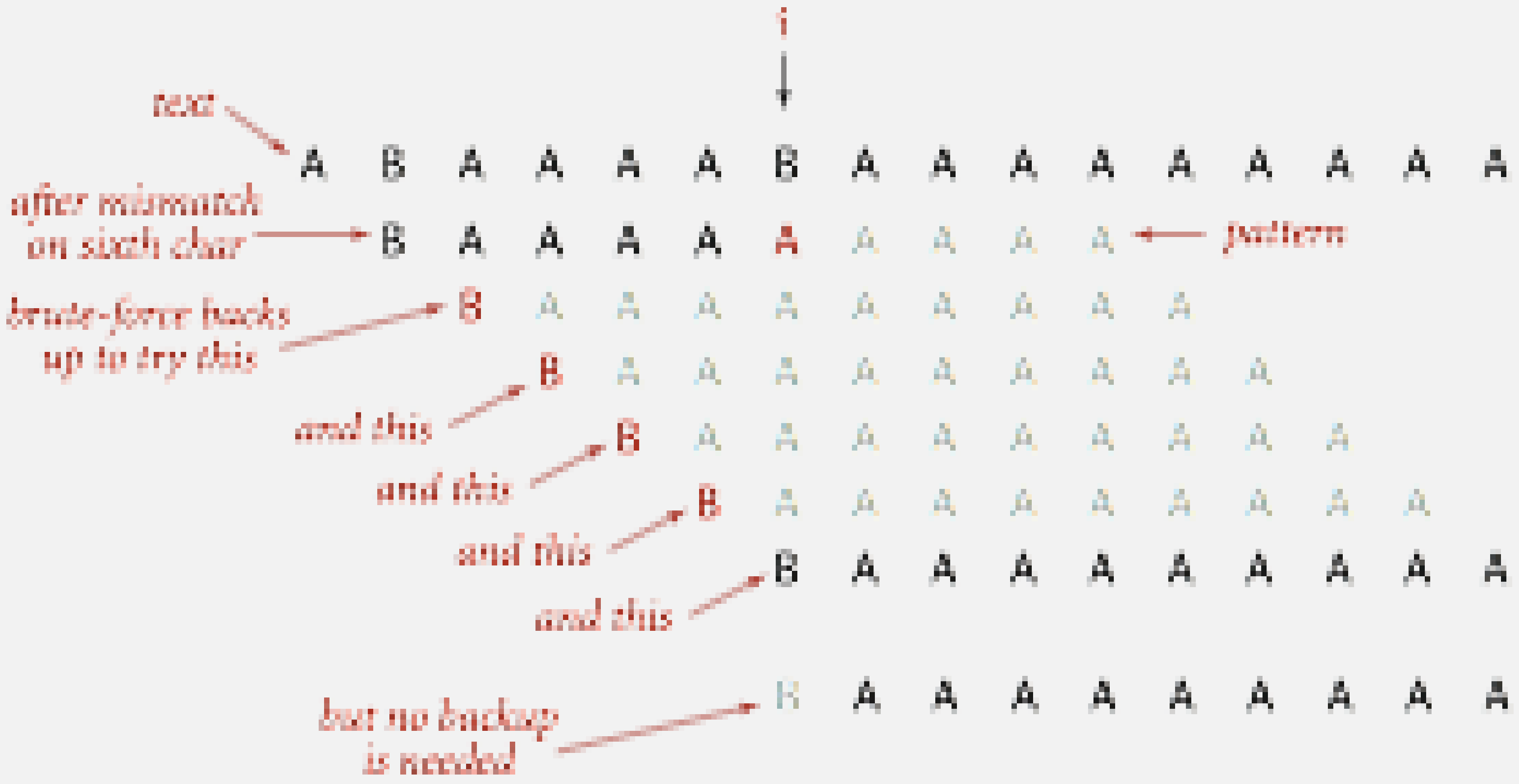
- ▸ *introduction*
- ▸ *brute force*
- ▸ *Knuth-Morris-Pratt*
- ▸ *Boyer-Moore*
- ▸ *Rabin-Karp*

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 6th 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)

A deterministic finite automaton is a 5-tuple $(Q, \square, \delta, q_0, F)$, where

- Q is a finite set of *states*
- Σ is a finite input *alphabet*
- $\delta : Q \times \square \to Q$ is a *transition function* of the form $\delta(q, a) \in Q$ for each $q \in Q$ and $a \in \Sigma$
- $q_0 \in Q$ is the *initial state*
- $F \subseteq Q$ is a set of *final (accepting) states*

Example 1: Consider the DFA $M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_0\}\,)$ where δ is given by the following table and transition diagram:

Table:

Diagram:

δ	0	1
q_0	q_2	q_1
q_0	q_3	q_0
q_0	q_0	q_3
q_0	q_1	q_2

Example for Substring Search

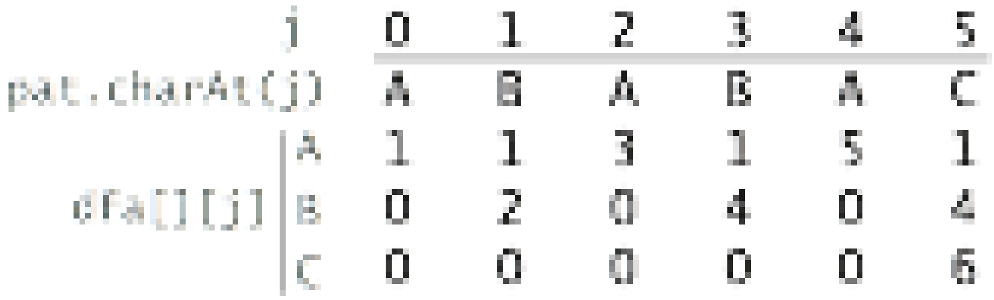
Example 2: Create a DFA for the pattern substring ABABAC from alphabet {A,B,C}.

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



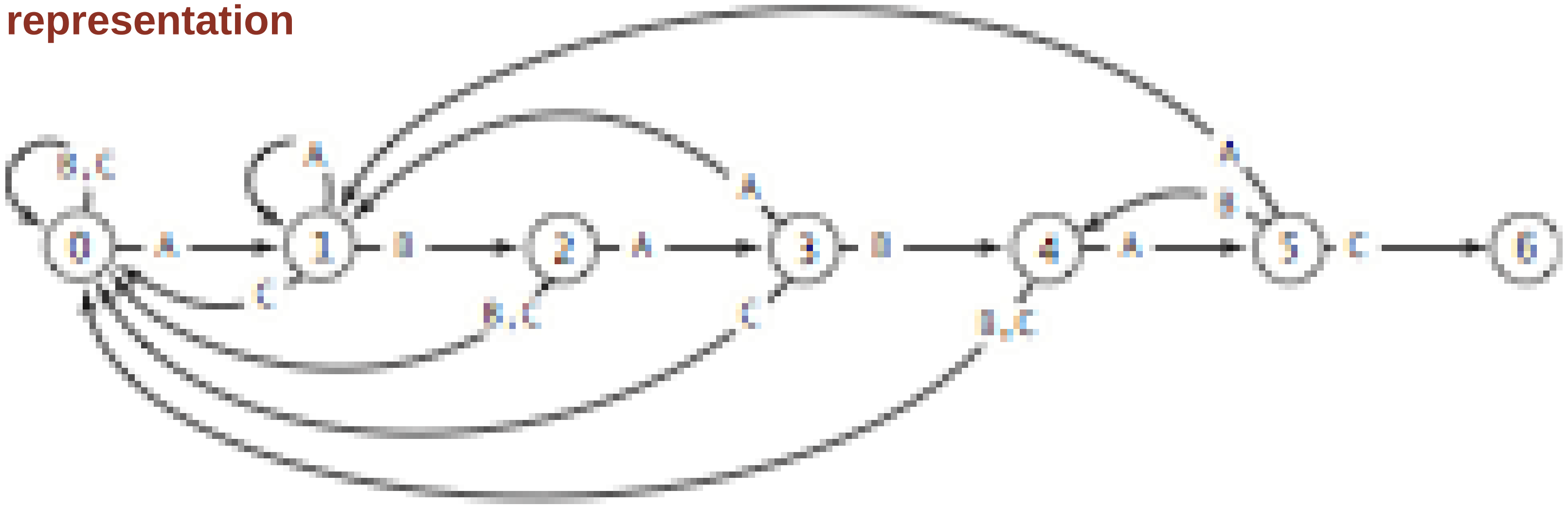
		0	1	2	3	4	5
j		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6

If in state j reading char c:

if j is 6 halt and accept

else move to state dfa[c][j]

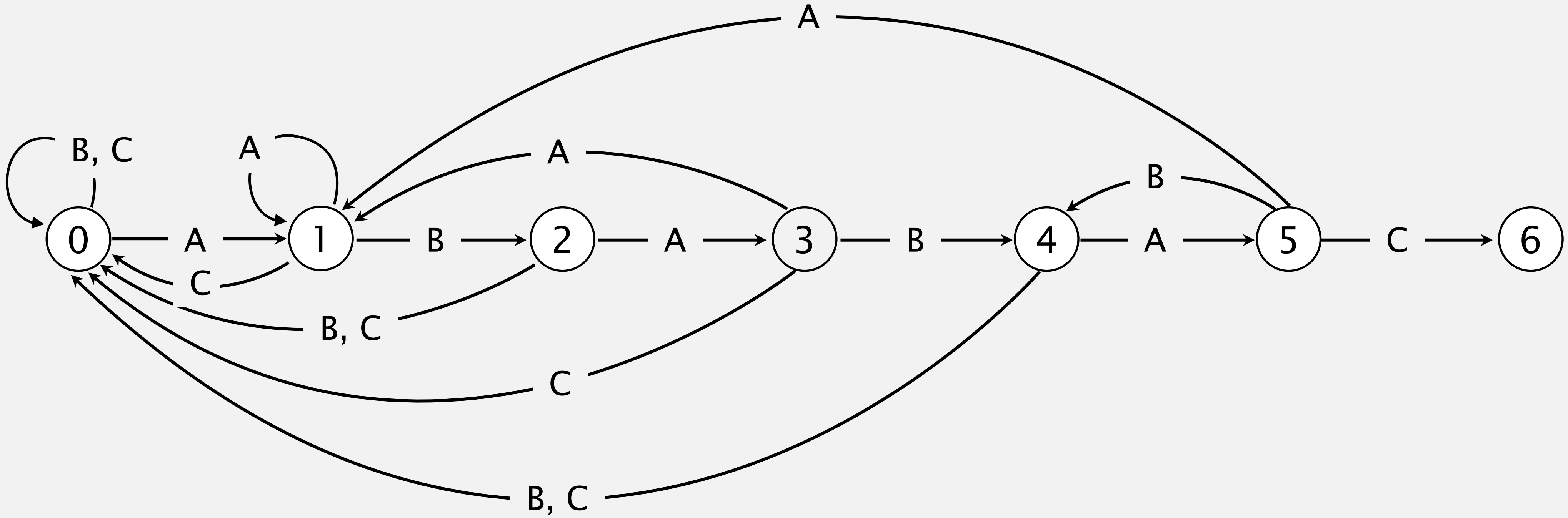
graphical representation



Knuth-Morris-Pratt demo: DFA simulation

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

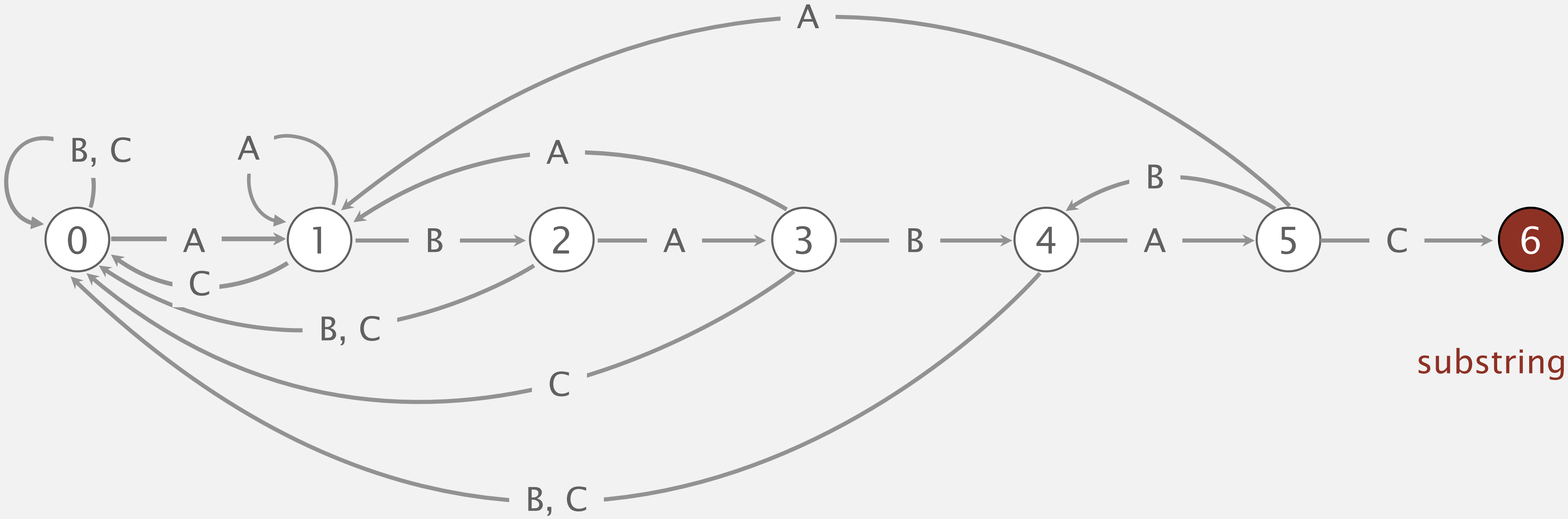
		0	1	2	3	4	5
pat.charAt(j)		A	B	A	B	A	C
dfa[][j]	A	1	1	3	1	5	1
	B	0	2	0	4	0	4
	C	0	0	0	0	0	6



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].

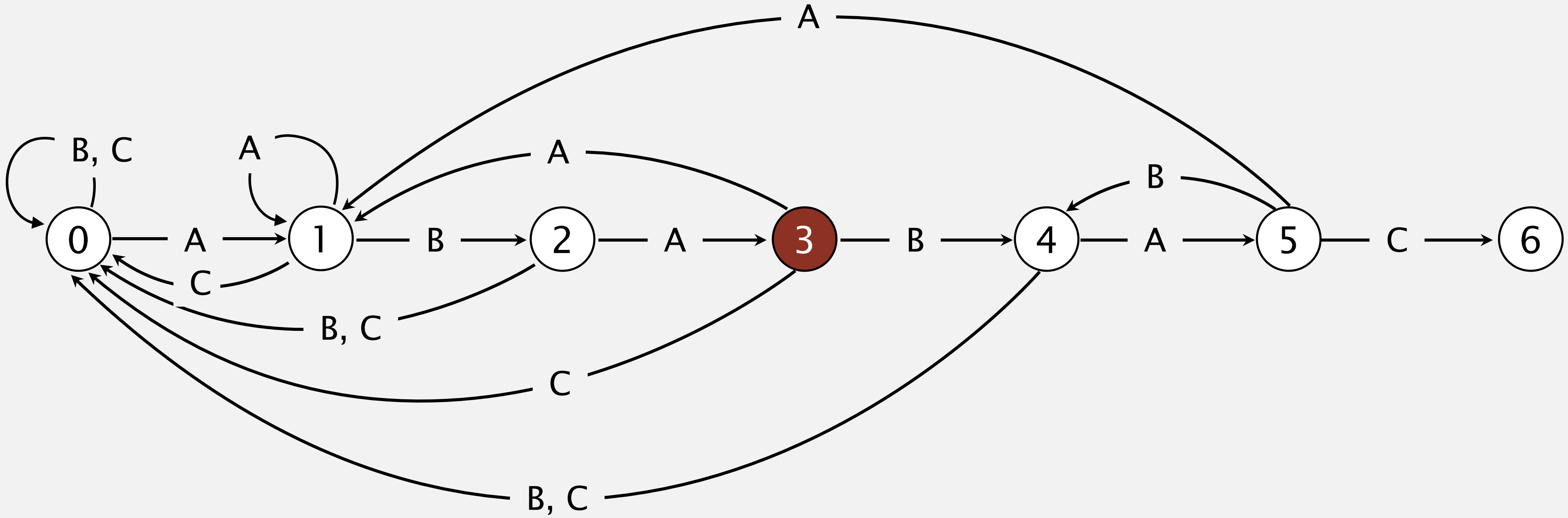
i

	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

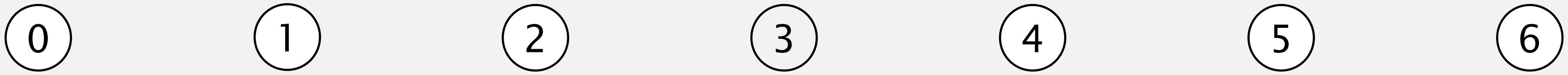
Running time.

- 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?

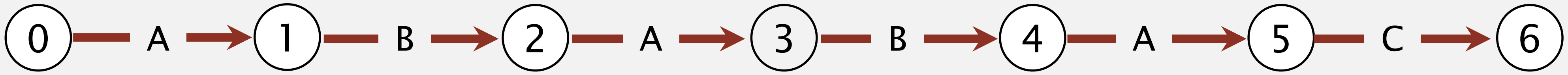
Match transition. If in state j and next char c == pat.charAt(j), go to j+1.

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.

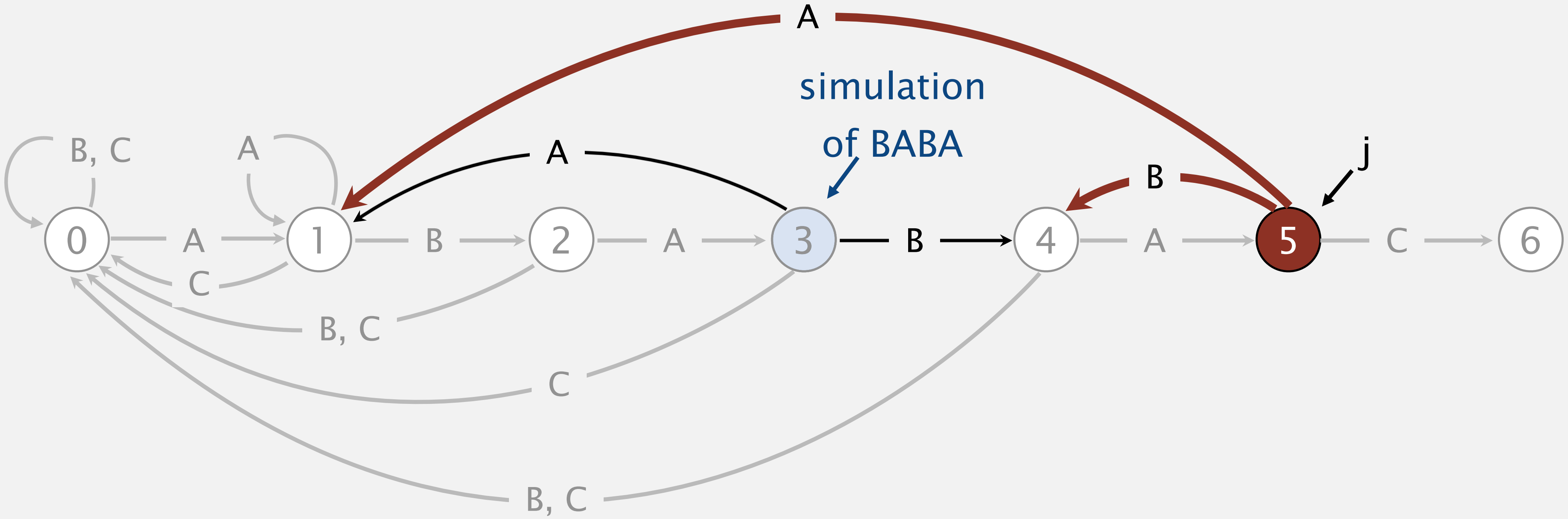
still under construction (!)

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


simulation of BABA

j

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;

from state X,
take transition 'A'
= dfa['A'][X]

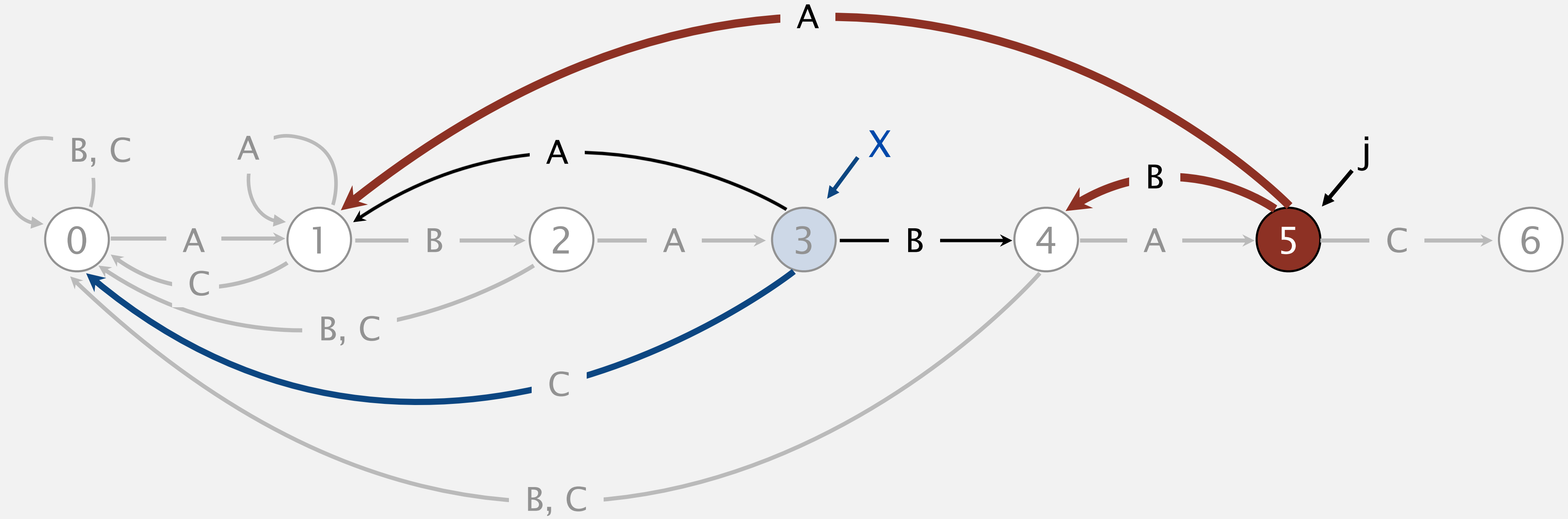
dfa['B'][5] = 4

from state X,
take transition 'B'
= dfa['B'][X]

X' = 0

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];
      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/time proportional to $R\,M$).

Example of DFA Construction: ABABAC

```
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];
  }
}
```

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

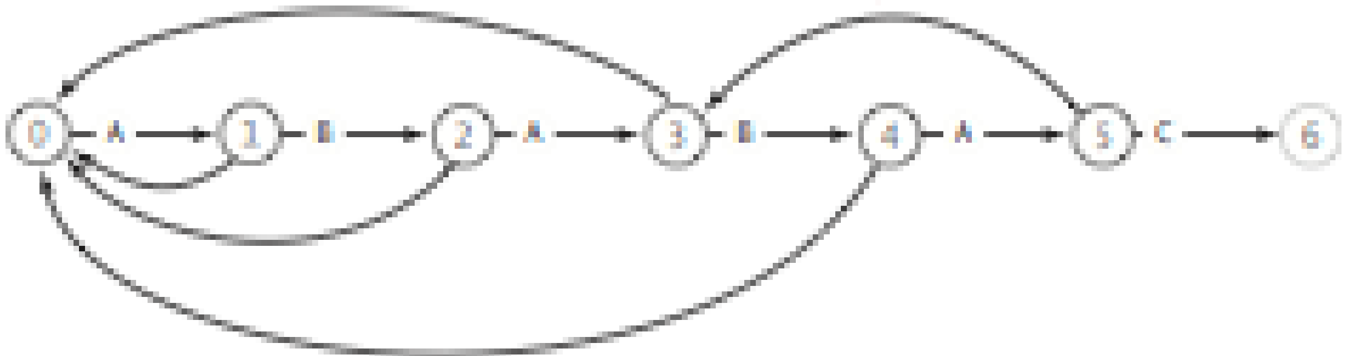
KMP substring search analysis

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