http://algs4.cs.princeton.edu

5.3 SUBSTRING SEARCH

- *introduction*
- *brute force*
- *Knuth–Morris–Pratt*
- *Boyer–Moore*

http://algs4.cs.princeton.edu

5.3 Substring Search

- *introduction*
- *brute force*
- *Knuth–Morris–Pratt*
- *Boyer–Moore*

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

Substring search applications

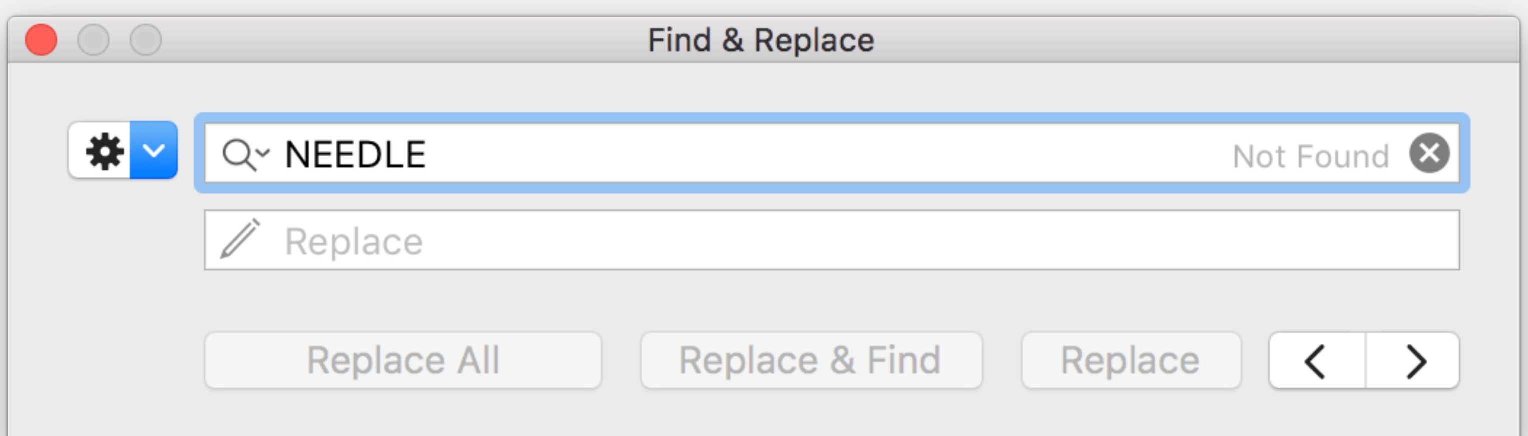
Goal. Find pattern of length m in a text of length n.

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



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.

http://citp.princeton.edu/memory

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 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)

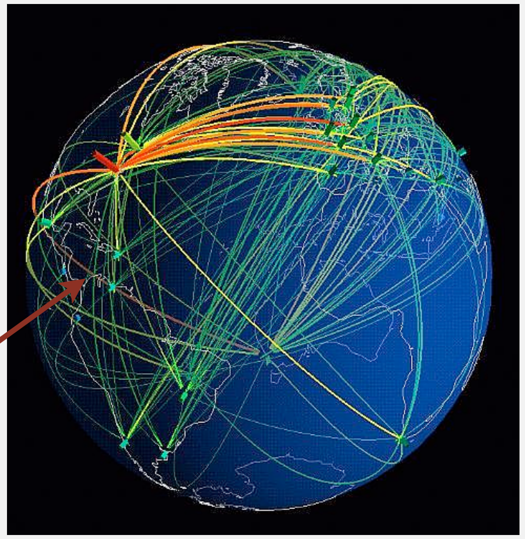
No way! (privacy)

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



Substring search applications

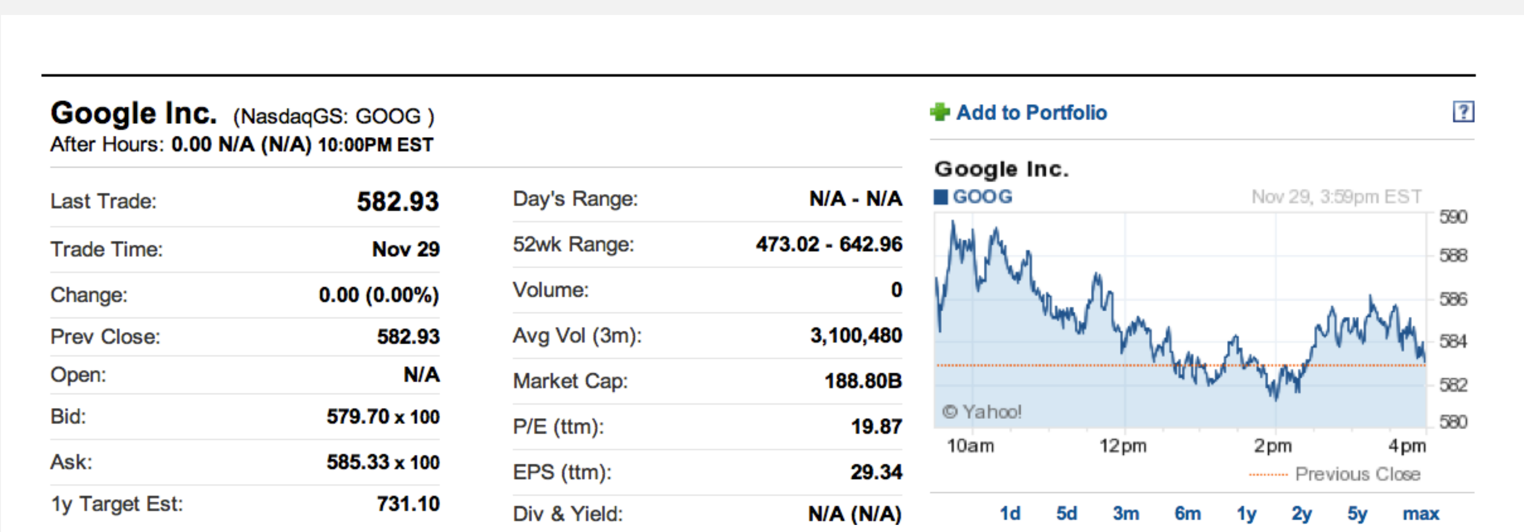
Web scraping. Extract relevant data from web page.

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

raw HTML

```
...
<tr>
<td class= "yfnc_tablehead1"
width= "48%">
Last Trade:
</td>
<td class= "yfnc_tabledata1">
<big><b>582.93</b></big>
</td></tr>
<td class= "yfnc_tablehead1"
width= "48%">
Trade Time:
</td>
<td class= "yfnc_tabledata1">
...
```

as rendered by browser



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

Web scraping: Java implementation

Java library. The `indexOf()` method in Java's `String` data type 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
```

Caveat. Must update program whenever Yahoo format changes.

http://algs4.cs.princeton.edu

5.3 Substring Search

- introduction
- **brute force**
- Knuth–Morris–Pratt
- Boyer–Moore

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						
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

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++)        for each possible offset
   {
      int j;                               number of characters that match
      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
}
```

Substring search: quiz 1

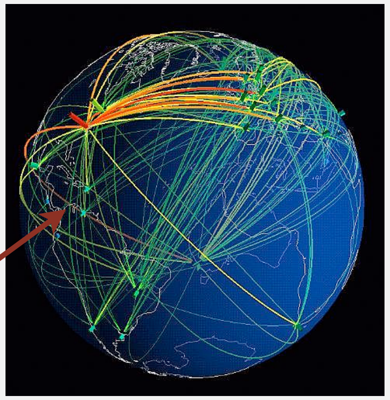
What is the worst-case running time of brute-force substring search as a function of both the pattern length m and text length n ?

- **A.** $m + n$
- **B.** m^2
- **C.** $m\,n$
- **D.** n^2

Backup

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

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



found

“ATTACK AT DAWN”
substring search machine

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.

Brute-force substring search: alternate implementation

Same sequence of character compares as previous implementation.

- `i` points to end of sequence of already-matched characters in text.
- `j` stores # of already-matched characters.

i	j	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;  }
   }
   if (j == m) return i - m;
   else            return n;
}
```

explicit backup

Algorithmic challenges in substring search

Brute-force substring search is not always good enough.

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

Practical challenge. Avoid backup in text stream. ⟵ avoid extra buffer

```
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

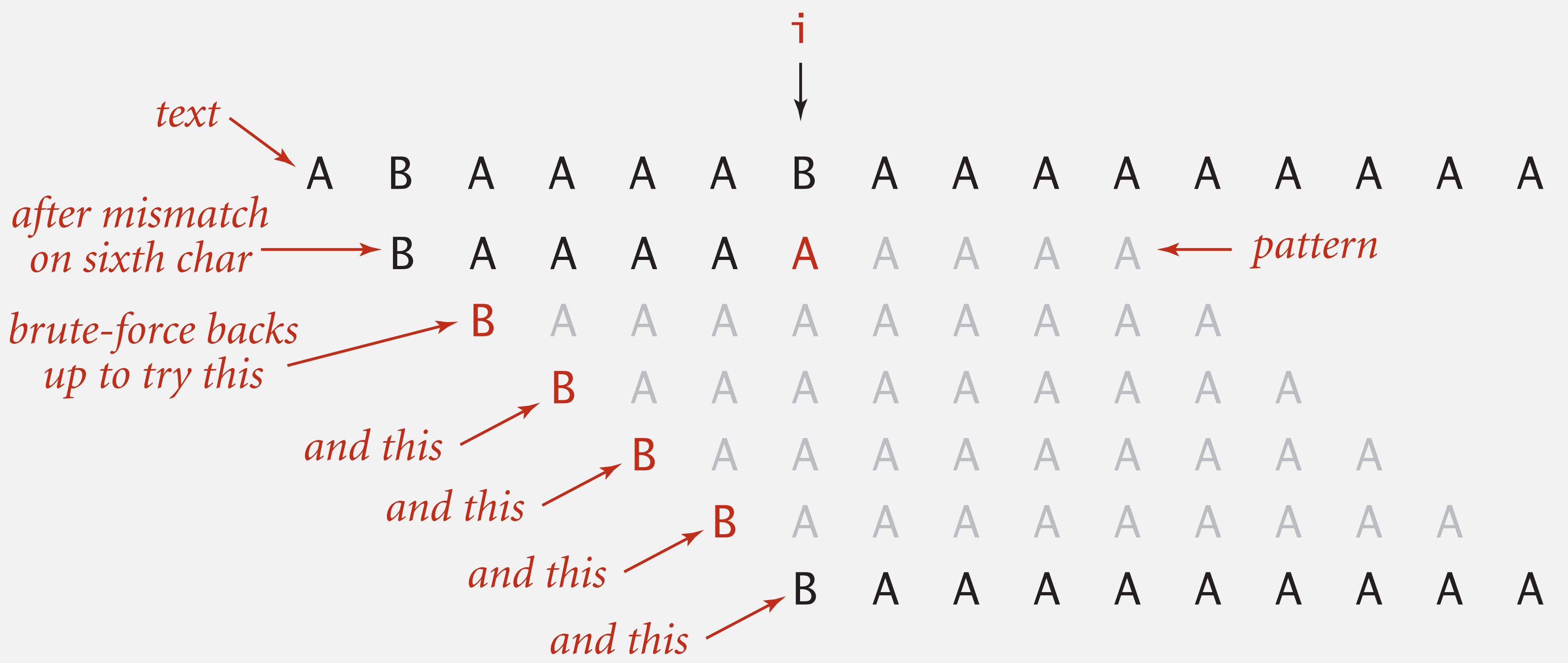
5.3 Substring Search

- introduction
- brute force
- **Knuth–Morris–Pratt**
- Boyer–Moore

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.

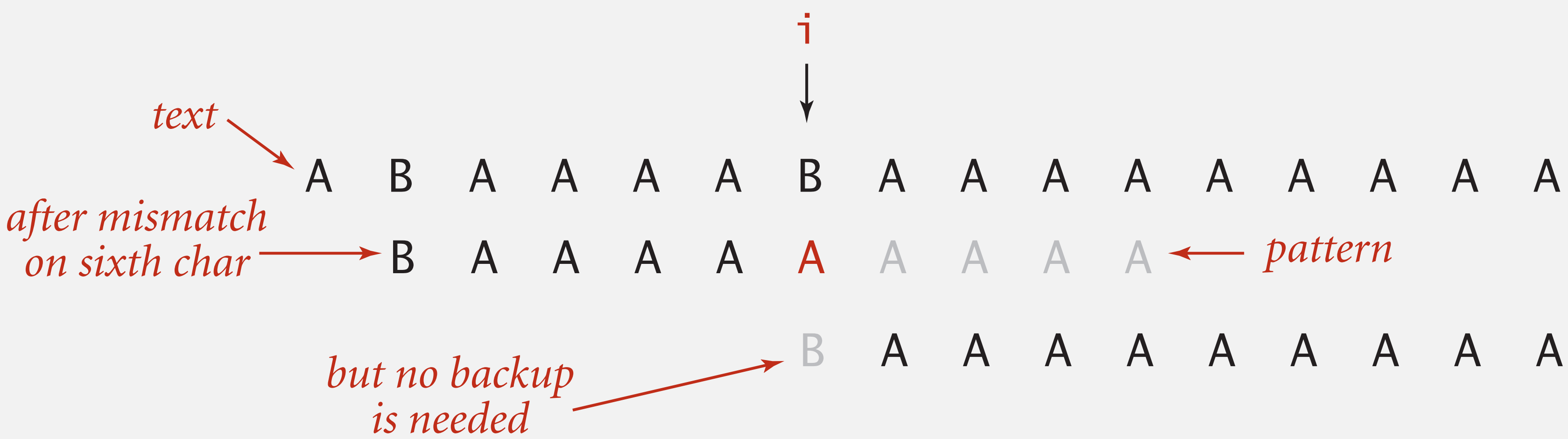


Knuth–Morris–Pratt substring search

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 must be 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.

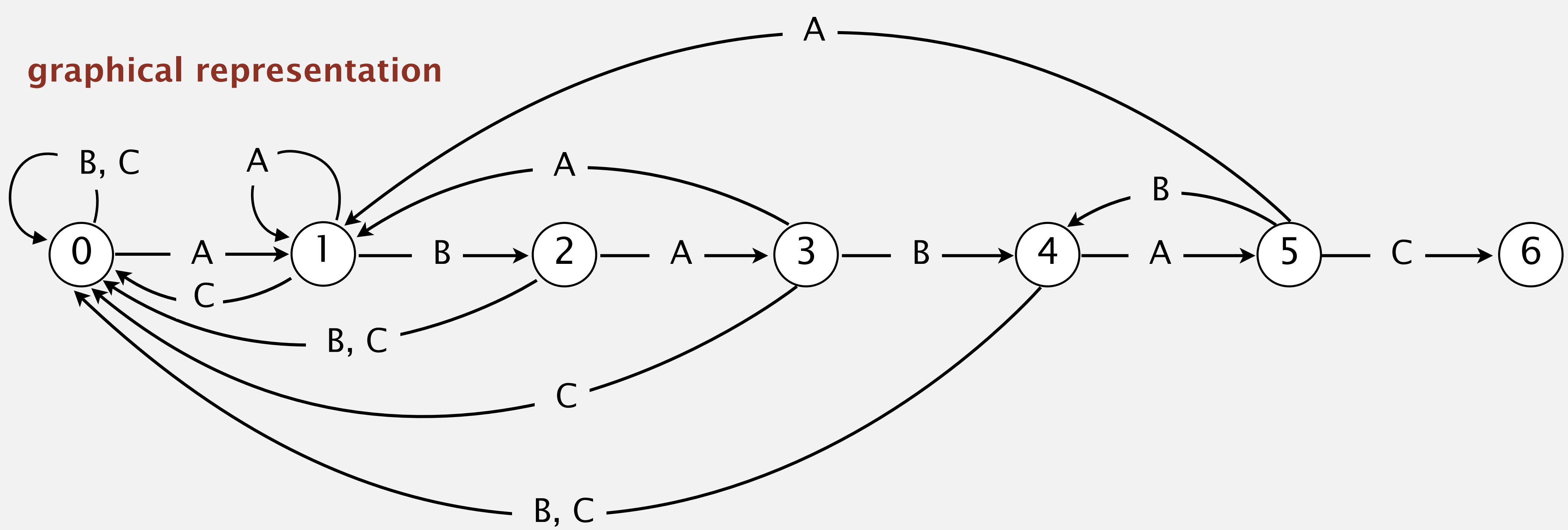
- Finite number of states (including start and halt).
- Exactly one state transition for each char in alphabet.
- Accept if sequence of state transitions leads to halt state.

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]`

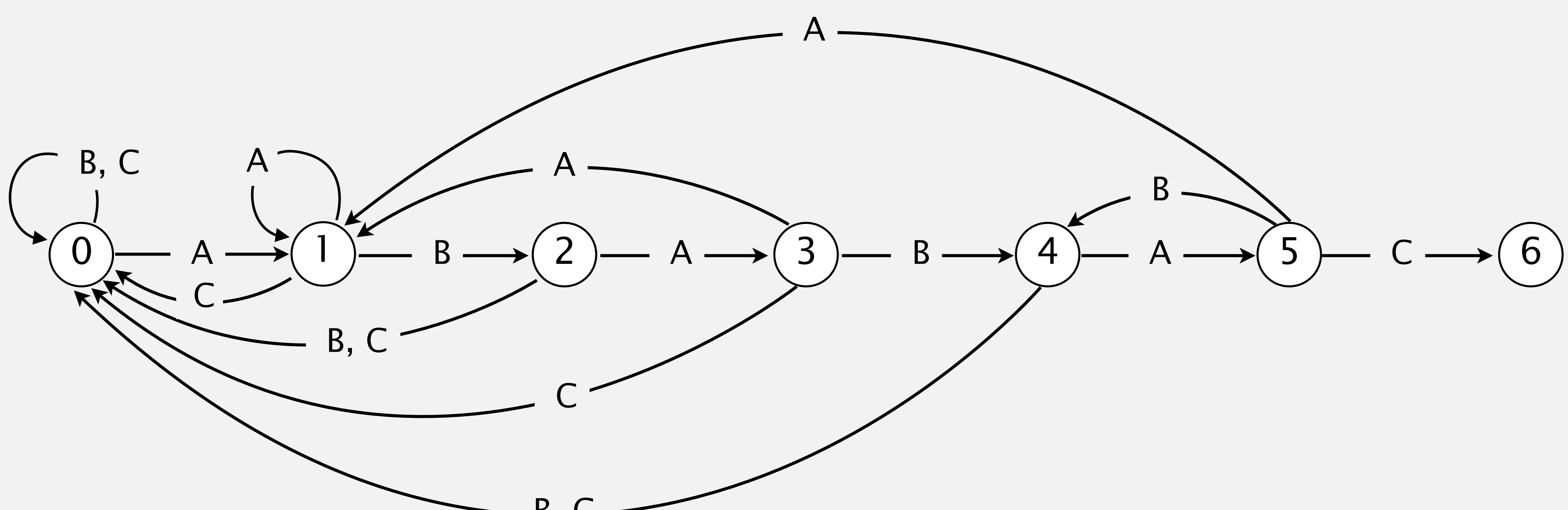
graphical representation



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



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]`.

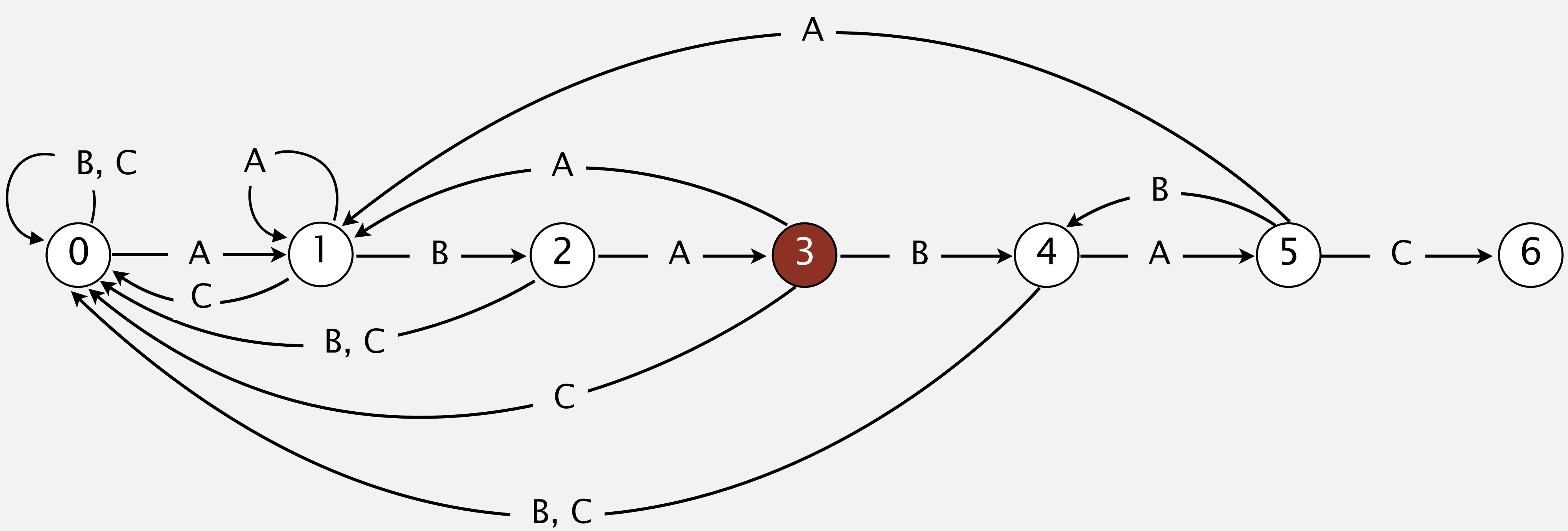
i

	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[]`



Substring search: quiz 2

Which state is the DFA in after processing the following input?

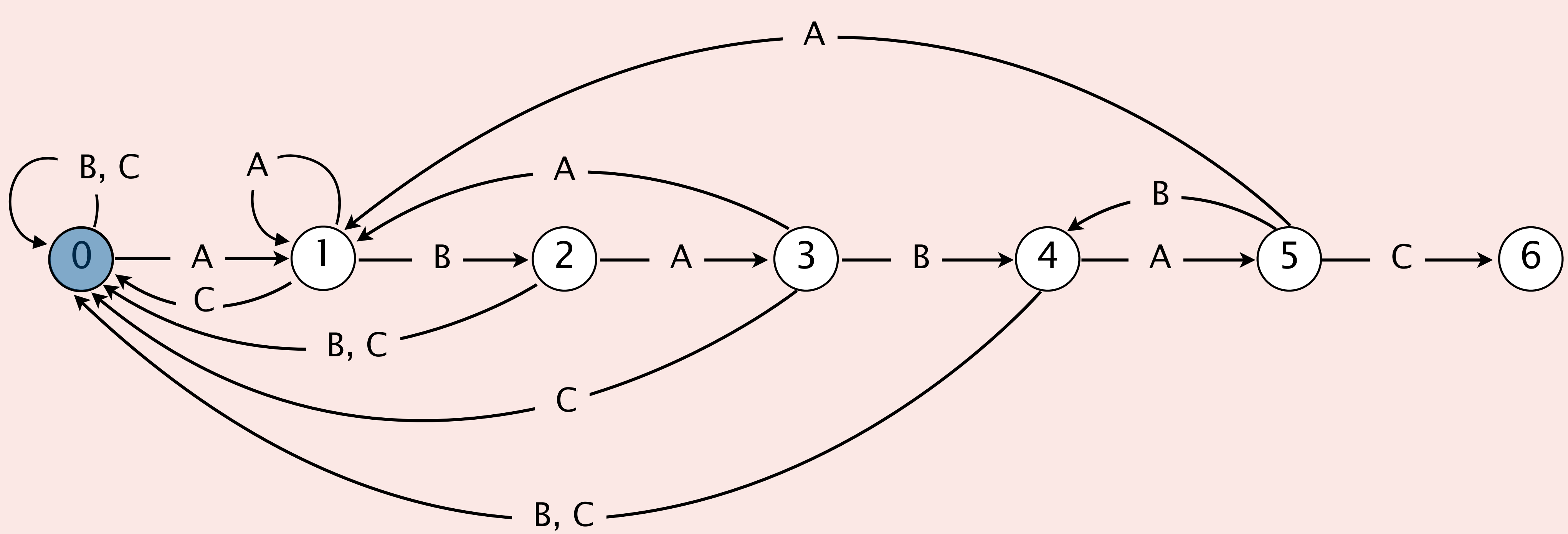
B A A B A B A B

A. 0

B. 1

C. 3

D. 4



Substring search: quiz 3

Which state is the DFA in after processing the following input?

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

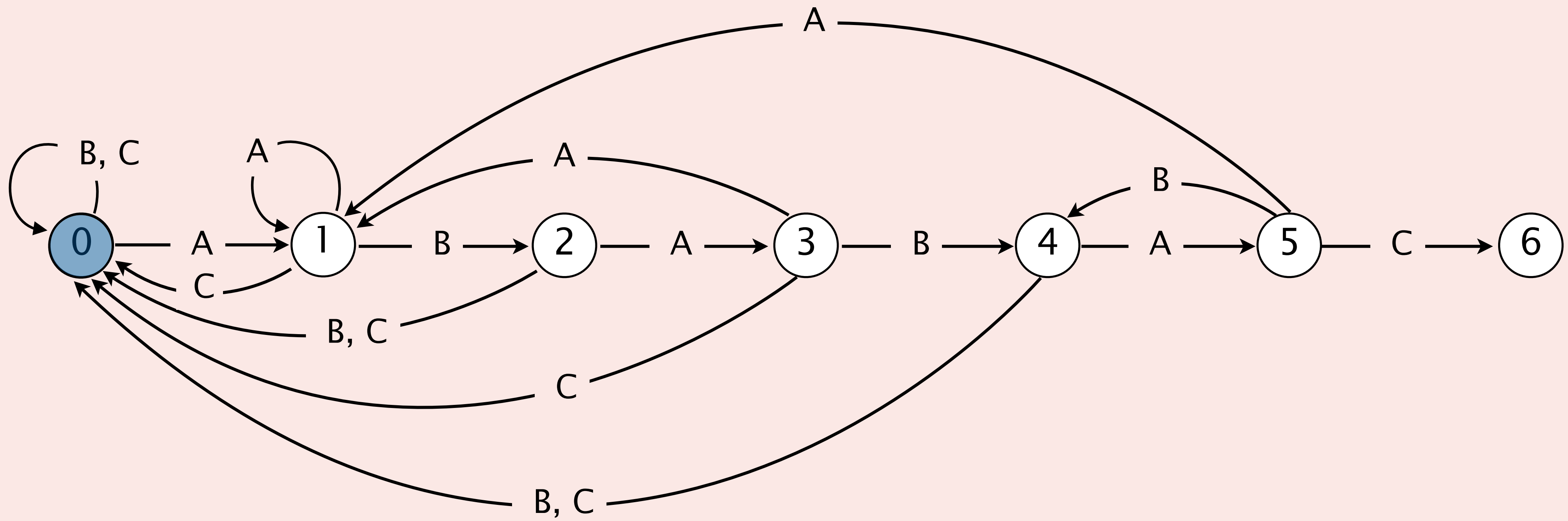
A. 0

B. 1

C. 3

D. 4

E. 5



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]

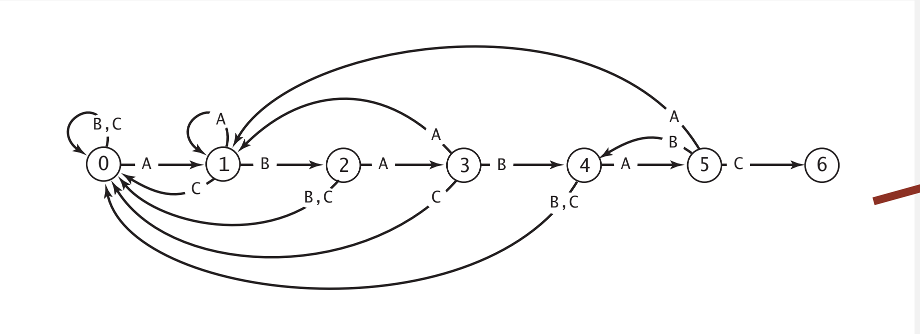
Knuth–Morris–Pratt substring search: Java implementation

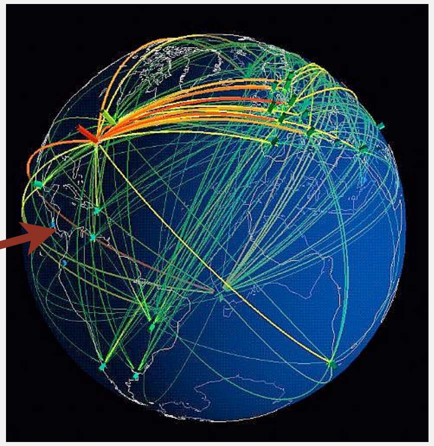
Key differences from brute-force implementation.

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

no backup

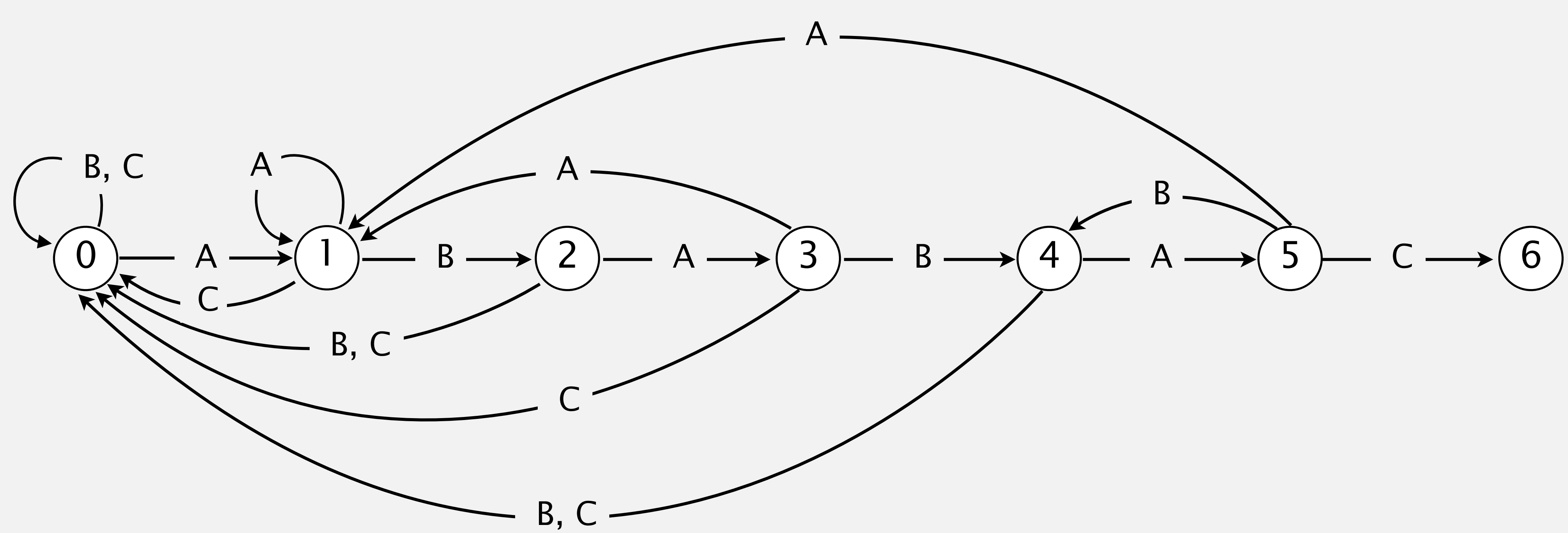




Knuth-Morris-Pratt demo: DFA construction

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

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



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?

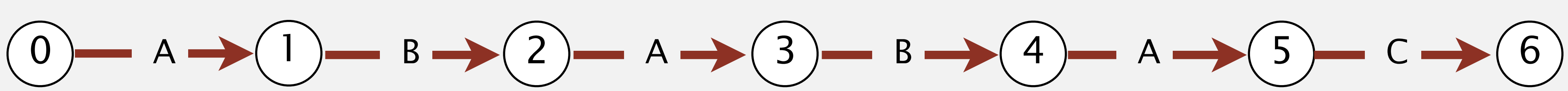
Match transition. If in state `j` and next char `c == pat.charAt(j)`, go to `j+1`.

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`.

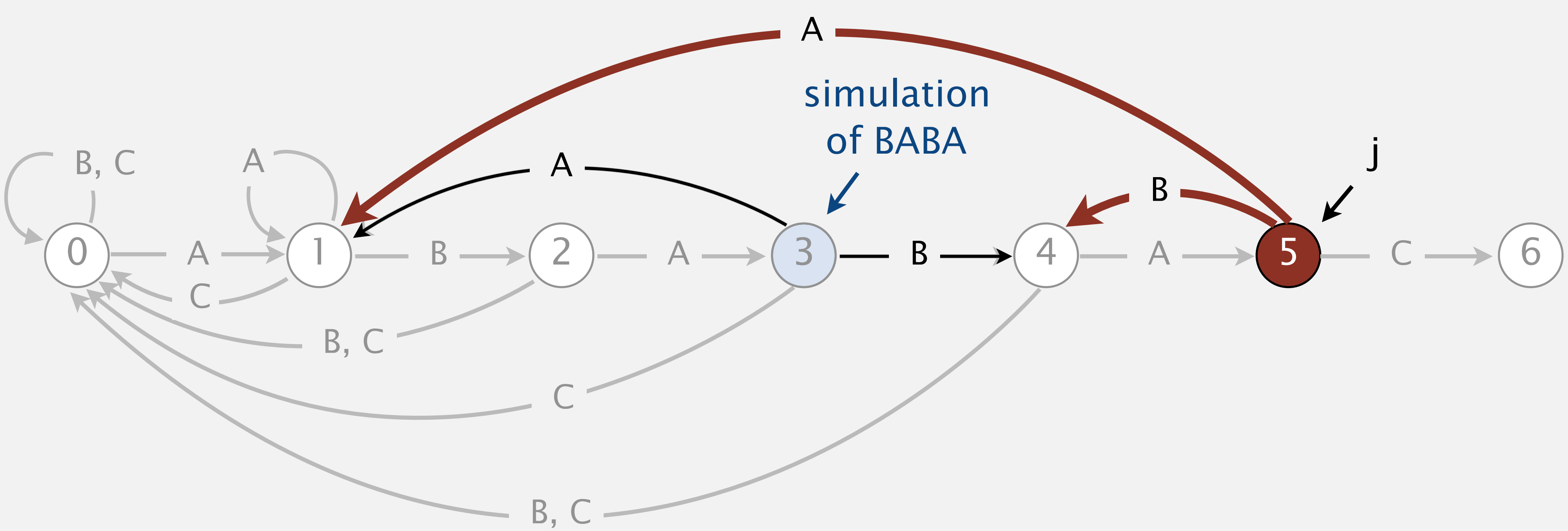
To compute `dfa[c][j]`: Simulate `pat[1..j-1]` on DFA and take transition `c`.
Running time. Seems to require `j` steps.

still under construction (!)

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

simulate BABAA   simulate BABAB

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.

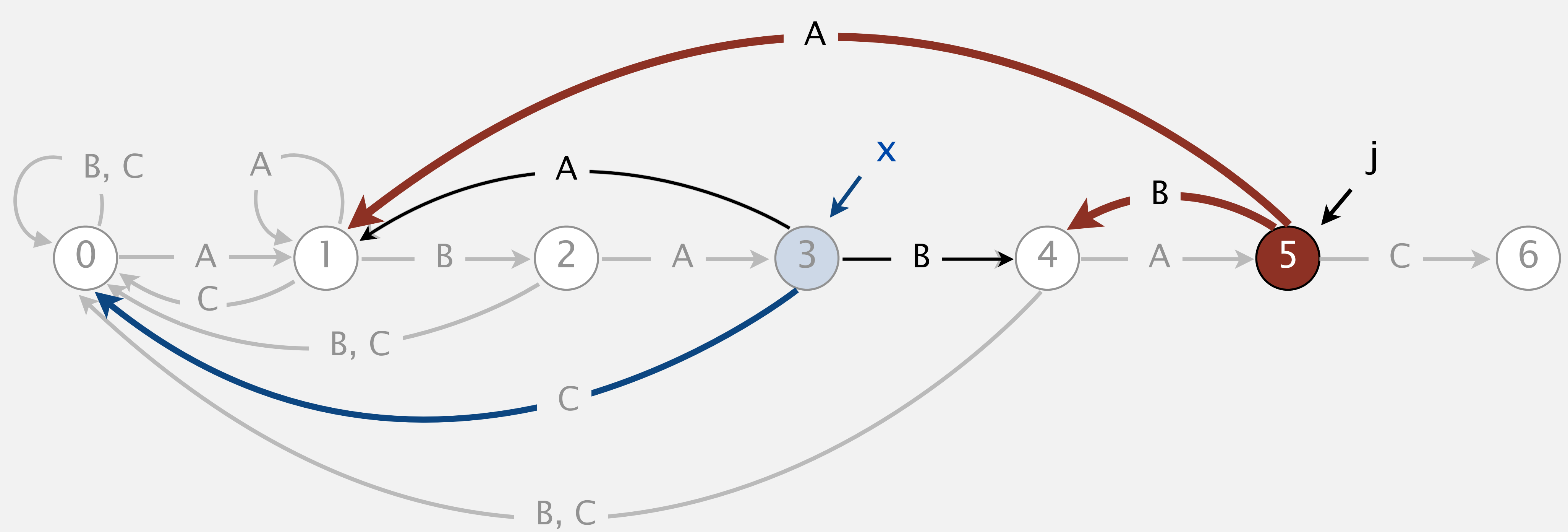
Ex. `dfa['A'][5] = 1` `dfa['B'][5] = 4` `x' = 0`

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]

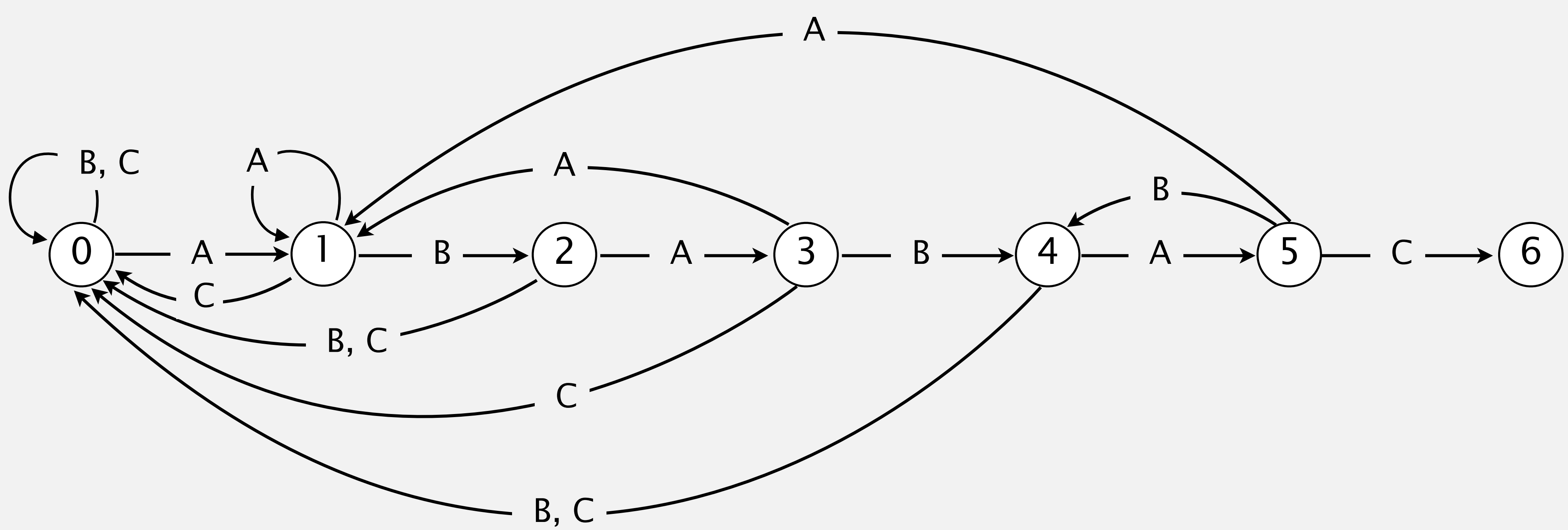
0	1	2	3	4	5
A	B	A	B	A	C



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

Constructing the DFA for KMP substring search for 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 $R\,m$).

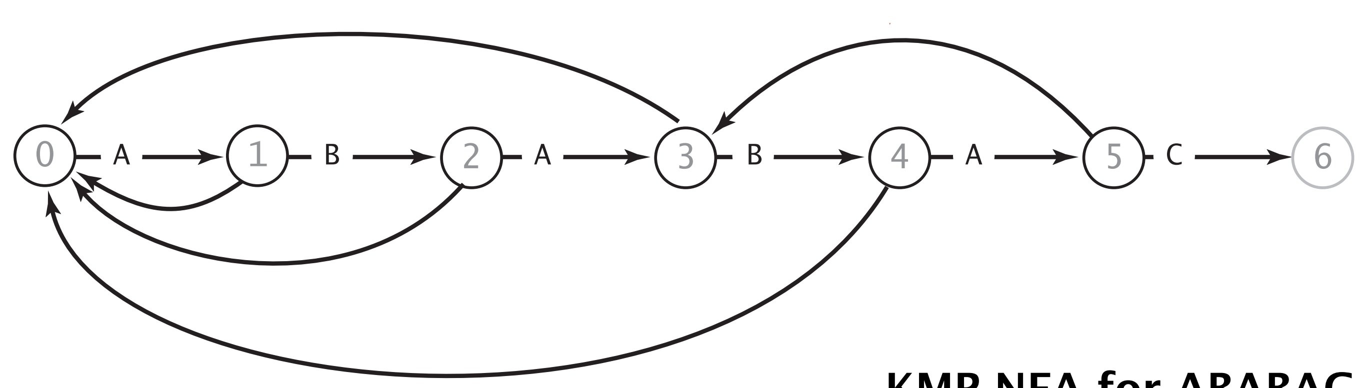
KMP substring search analysis

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 character accessed once when constructing the DFA; each text character accessed once (in the worst case) when simulating the DFA.

Proposition. KMP constructs `dfa[][]` in time and space proportional to $R\,m$.

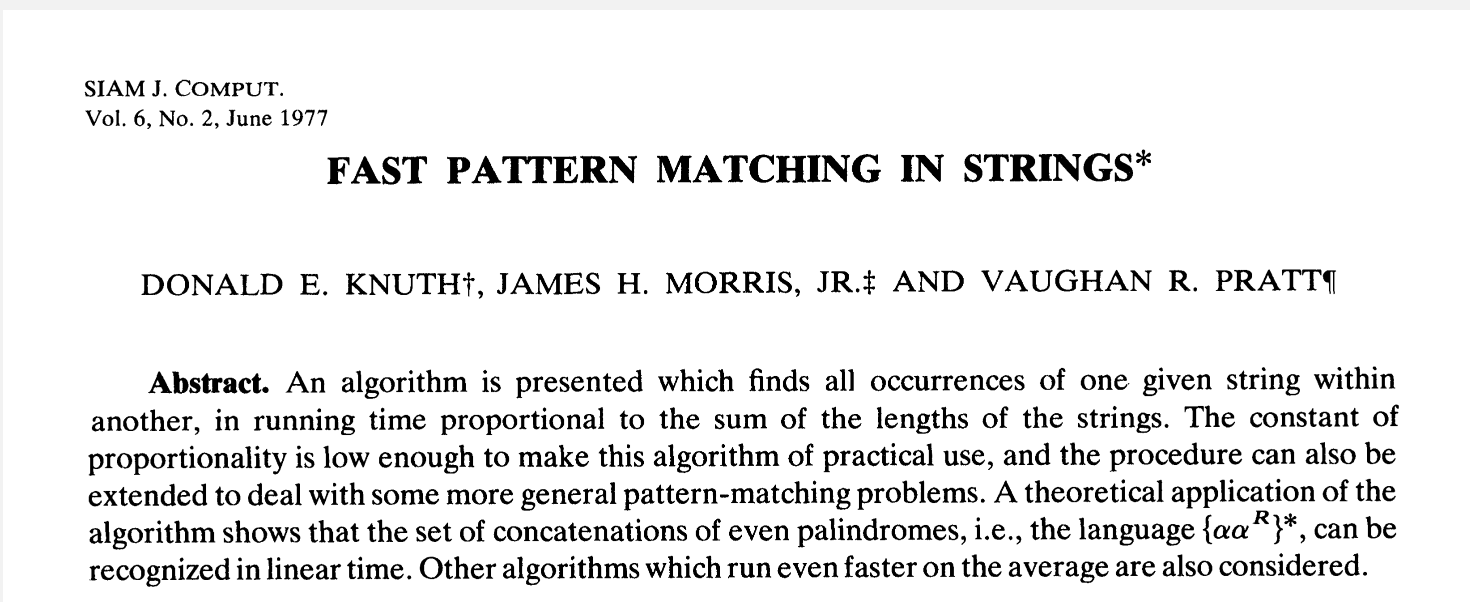
Larger alphabets. Improved version of KMP constructs `nfa[]` in time and space proportional to m.



KMP NFA for ABABAC

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

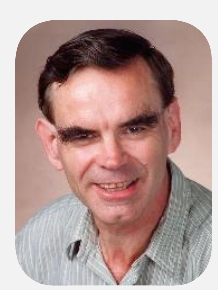
FAST PATTERN MATCHING IN STRINGS*

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

Cyclic Rotation

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 binary strings s and t, design a linear-time algorithm to determine if s is a cyclic rotation of t.

Cyclic Rotation

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 binary strings s and t, design a linear-time algorithm to determine 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 Knuth–Morris–Pratt.

```
t + t  ⟶  STRINGROTATEDSTRINGROTATED
s      ⟶        ROTATEDSTRING
```

http://algs4.cs.princeton.edu

5.3 Substring Search

- *introduction*
- *brute force*
- *Knuth–Morris–Pratt*
- *Boyer–Moore*

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.

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
	text →	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 ← *pattern*																		
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

no S in pattern

align N in text with N in pattern

align N in text with N in pattern

Boyer-Moore: mismatched character heuristic

Q. How much to skip?

Case 1. Mismatch character not in pattern.

before

i ↓

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

pat N E E D L E

after

i ↓

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

pat N E E D L E

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.

before

```
                i
                ↓
txt  . . . . . . N L E . . . . . .
pat        N E E D L E
```

after

```
                   i
                   ↓
txt  . . . . . . N L E . . . . . .
pat              N E E D L E
```

mismatch character N in pattern: align text N with rightmost (why?) pattern N

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 ?

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
```

after

```
                i
                ↓
txt  . . . . . . E L E . . . . . .
pat          N E E D L E
```

mismatch character E in pattern: increment i by 1

Substring search: quiz 4

Which text character is compared with the E next in Boyer–Moore?

A. R (index 5)

B. O (index 6)

C. O (index 12)

D. O (index 13)

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
text →	B	O	O	Y	E	R	O	B	E	R	T	M	O	O	R	E	J	S
pattern →	M	O	O	R	E													
					M	O	O	R	E									

Substring search: quiz 5

Which text character is compared with the E next in Boyer–Moore?

A. O

B. R

C. E

D. J

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
text →	B	O	O	Y	E	R	O	B	E	R	T	M	O	O	R	E	J	S
pattern →	M	O	O	R	E													
					M	O	O	R	E									
									M	O	O	R	E					

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: 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 zero or negative

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 $\sim m\, n$.

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.