Knuth-Morris-Pratt Algorithm

for Pattern Matching

Pattern Matching

The act of checking a given sequence of tokens (**text**) for the presence of the constituents of some **pattern**.

Brute Force Approach

- Pattern of length **M**
- Text of length **N**
- Complexity?

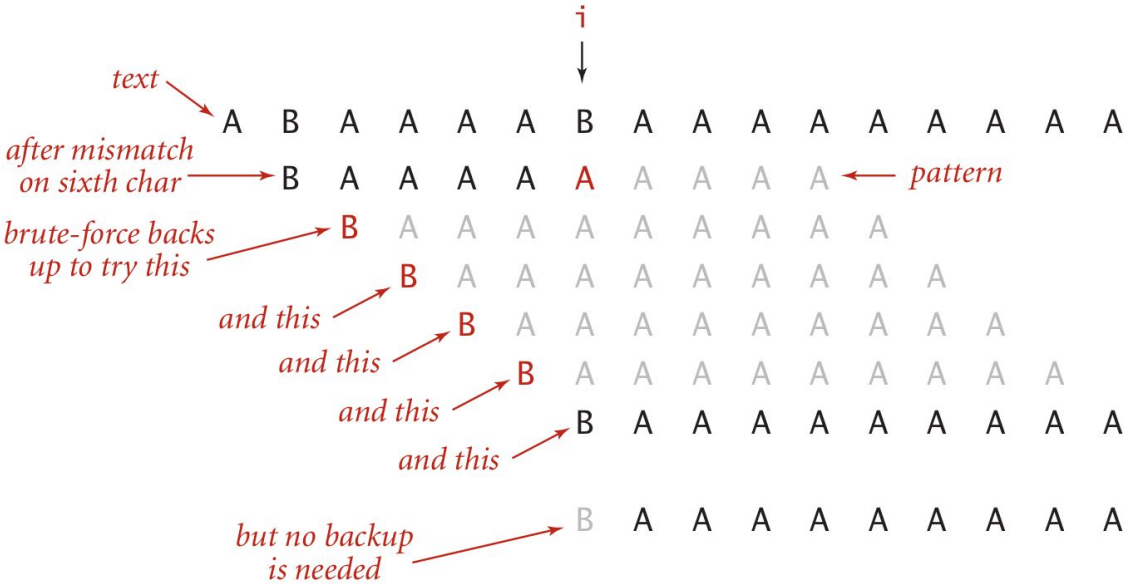
  O(N*M)

- Worst case?

  Text = XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXY
  Pattern = XXXXY

Knuth-Morris-Pratt Algorithm

- Takes advantage of the information about already matched characters to reduce the number of comparisons.
- Avoids backing up in the text (only moves forward).



KMP

To keep track of available shifts during each mismatched character we build a **DFA** (deterministic finite-state automata).

DFA is constructed just from the pattern and before the execution.

KMP DFA

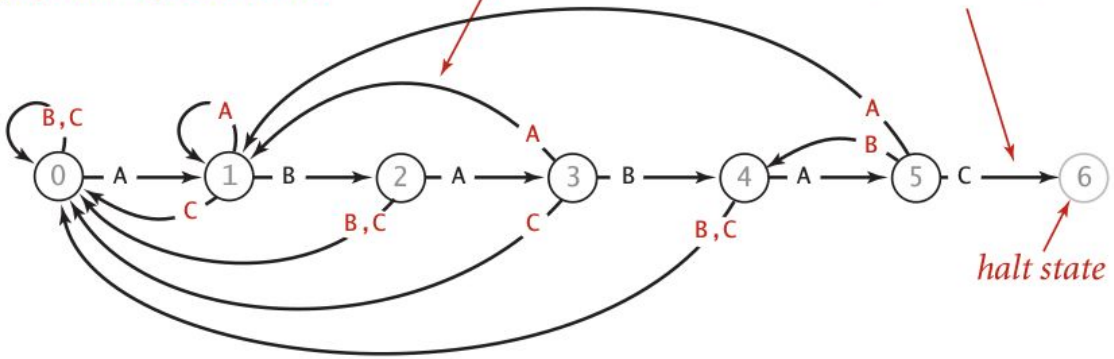
internal representation

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
dfa[][j] B	0	2	0	4	0	4
dfa[][j] C	0	0	0	0	0	6

mismatch transition (back up)

match transition (increment)

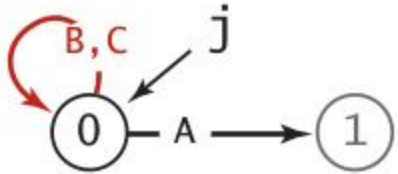
graphical representation



DFA corresponding to the string A B A B A C

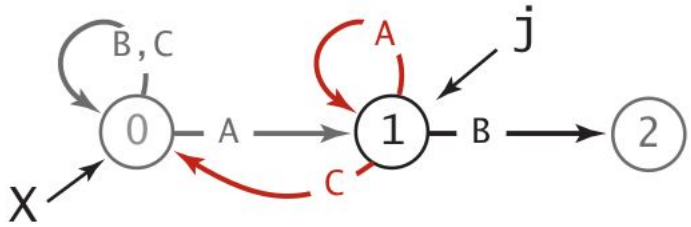
DFA Construction

j		0
pat.charAt(j)		A
dfa[][j]	A	1
	B	0
	C	0



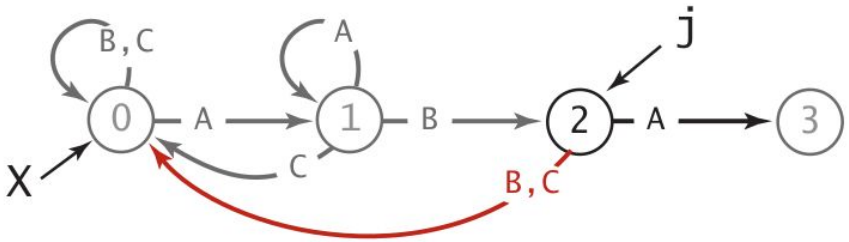
DFA Construction

		X	
	j	0	1
pat.charAt(j)		A	B
dfa[][j]	A	1	1
	B	0	2
	C	0	0



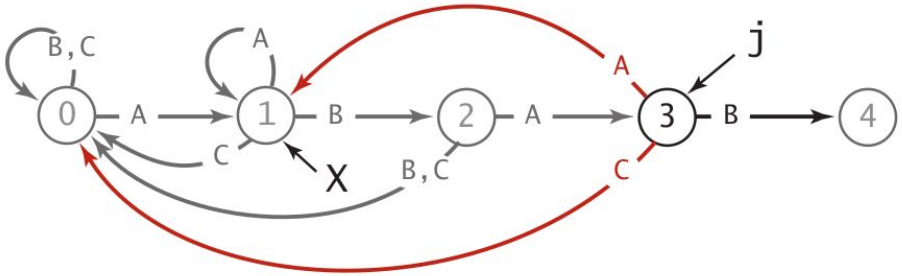
DFA Construction

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



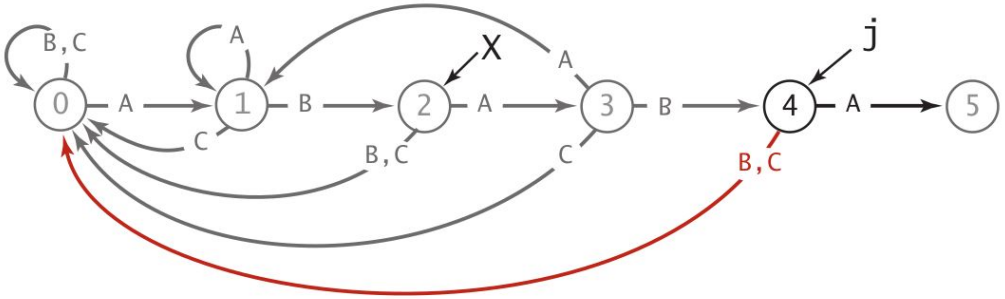
DFA Construction

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



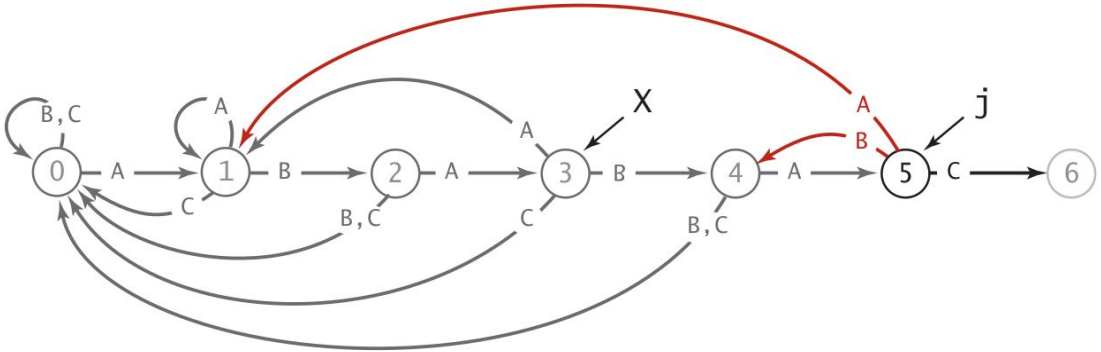
DFA Construction

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



DFA Construction

	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



KMP Code

```
public class KMP
{
   private String pat;
   private int[][] dfa;

   public KMP(String pat)
   {  // Build DFA from pattern.
      this.pat = pat;
      int M = pat.length();
      int R = 256;
      dfa = new int[R][M];
      dfa[pat.charAt(0)][0] = 1;
      for (int X = 0, j = 1; j < M; j++)
      {  // Compute dfa[][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.
      }
   }

   public int search(String txt)
   {  // Simulate operation of DFA on txt.
      int i, j, N = txt.length(), M = pat.length();
      for (i = 0, j = 0; i < N && j < M; i++)
         j = dfa[txt.charAt(i)][j];
      if (j == M) return i - M;  // found (hit end of pattern)
      else        return N;      // not found (hit end of text)
   }

   public static void main(String[] args)
   // See page 769.
}
```

KMP

In Class Practice

Build a KMP DFA for the following pattern:

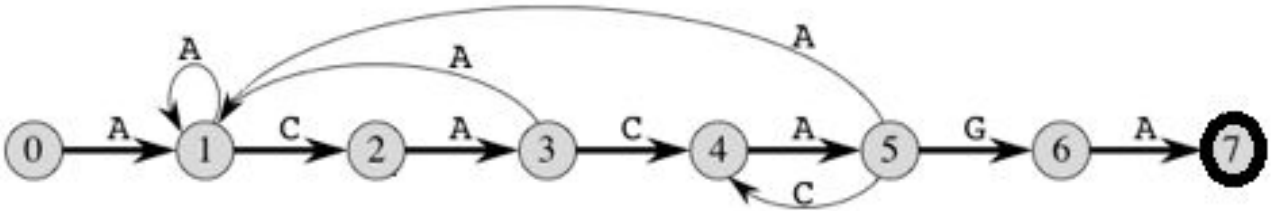
A C A C A G A

Produce both **table** and **graphical** representations of the DFA

KMP

Answer

	A	C	A	C	A	G	A
	0	**1**	**2**	**3**	**4**	**5**	**6**
A	1	1	3	1	5	1	7
C	0	2	0	4	0	4	0
G	0	0	0	0	0	6	0



Knuth-Morris-Pratt Algorithm

Questions?