

PSC'05

Reordering Finite Automata States for Fast String Recognition

E. Ketcha Ngassam

Bruce W. Watson

Derrick G. Kourie

FASTAR Research Group www.fastar.org

Agenda

- Background and Motivations
- The Table-driven algorithm
- The state Reordering Algorithm
- Illustrative Example
- Theoretical Assessment
- Experimental Results
- Conclusion and Future Work

Background and Motivations

- Finite Automata Implementations
 - Traditionally uses table-driven approach
 - Memory load problems
 - Limited capacity of data cache
 - Alternative approach: Hardcoding
 - Suggested by:
 - Thompson (Regular expressions, 1968)
 - Knuth (Pattern Matching, 1977)
 - Instructions load problems
 - Limited capacity of instruction cache

Background and Motivations

- Need to improve processing performance by
 - Exploiting hardware's capabilities with emphasis on
- Cache memory
 - Faster than main memory
 - Data movements from cache to CPU are inexpensive
 - Miss penalties avoidance for data already present in cache through:
 - Temporal locality of reference
 - Spatial locality of reference
- Our algorithm exploits spatial locality

The Table-driven algorithm

```
Proc tdRecognizer(table,inString)
;state,index := 0,0
do (index<inString.length())^(state ≥ 0) →
    state := table[state][inString[index]]
    index := index+1
od
```

- Time complexity $O(|inString|)$
- The table is unordered
 - Spatial locality problem:
 - High probability of cache misses

The State Reordering algorithm

- Parameters:
 - `table`, `inString`: transition table, input string
 - `srTable` is a dynamic two-dimensional array
 - Begins at address `start`
 - Each block holds `size` bytes
 - Auxiliary array `srMap` holds the position (`pos`) of each state in `srTable`
 - `nextB` holds the next memory address to be allocated to `srTable[pos]`.
 - `n`, a total number of states, alphabet size
 - `m = srTable[i][j]`
 - $m < n \Leftrightarrow$ the state `m` is not reordered
 - $m \geq n \Leftrightarrow$ the state at `srTable[m-n]` is reordered

The State Reordering algorithm

```
{assume n is the number of states and a is alphabet size}  
Proc srRecognizer(table, inString, start, size)  
;srMap[0..n-1] := -1  
;nextB := start  
;state := 0  
;index := 0  
;pos := 0
```

```

do (index < inString.length() ^ state ≥ 0 ) →
  if state < n →
    ;srMap[state],srTable[pos] := pos,malloc(nextB,size)
    ;srTable[pos][0..a-1],k,j := table[state][0..a-1],0,0
    do k ≤ pos →
      do j ≤ a →
        ;m := srTable[k][j]
        if m < n ^ srMap[m] < 0 → skip {m ∉ visited(index)}
        | m < n ^ srMap[m] ≥ 0 → srTable[k][j] := srMap[m]+n
        | m ≥ n → skip {m already updated}
        fi
        ; j := j+1
      od
      ; k := k+1
    od
    state,pos,nextB := srTable[pos][inString[index]],pos+1,nextB+size
  | state ≥ n → state := srTable[state-n][inString[index]]
  fi
;index := index+1

od

```

Illustrative example

table

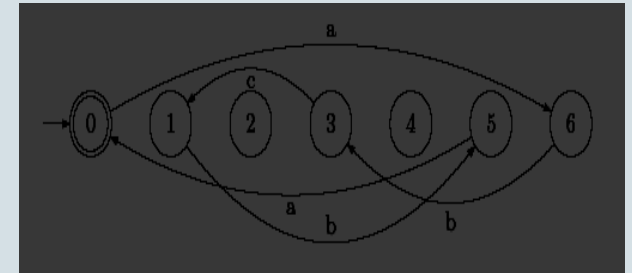
	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- String: abcbaabcba



Illustrative example

table

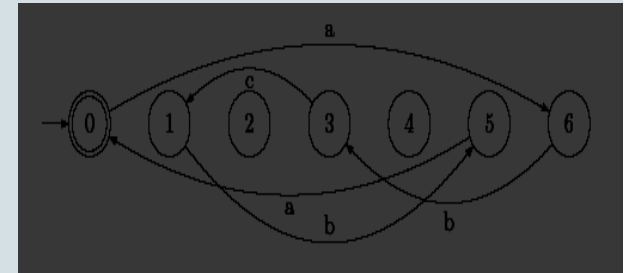
	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- String: abcbaabcba
- Initial phase:

srMap:

-1	-1	-1	-1	-1	-1	-1
----	----	----	----	----	----	----

– srTable does not exist



state

0

Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 1:
 - abcbaabcba

srMap:

0	-1	-1	-1	-1	-1	-1
---	----	----	----	----	----	----

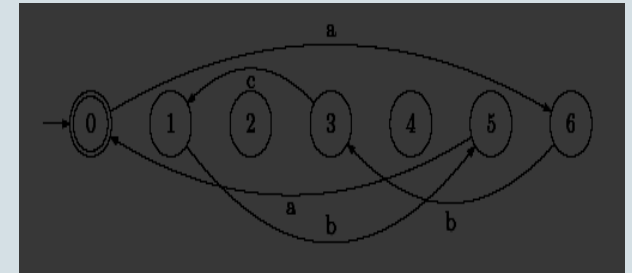
srTable:

6	3	1
---	---	---

 No change by update

state

0



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

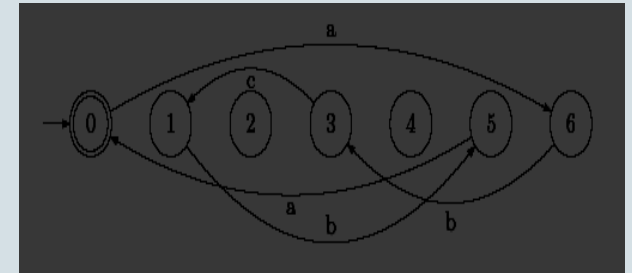
- Iteration 1:
 - abcbaabcba

srMap:

0	-1	-1	-1	-1	-1	-1
---	----	----	----	----	----	----

srTable:

6	3	1
---	---	---



state

6

Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 2:
 - abcbaabcba

srMap:

0	-1	-1	-1	-1	-1	1
---	----	----	----	----	----	---

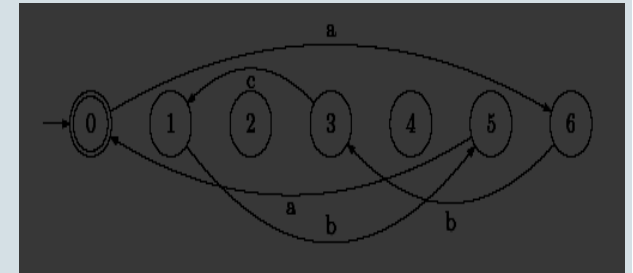
srTable:

6	3	1
1	3	5

State 6 to be updated to $1+7 = 8$

state

6



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 2:
 - **a**bcbaabcba

srMap:

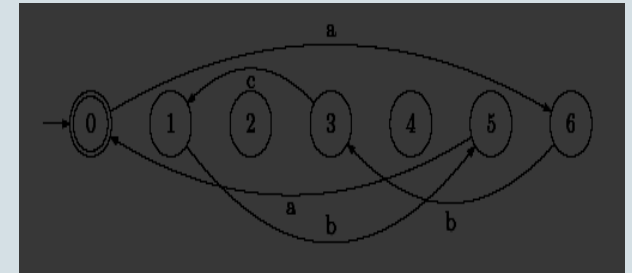
0	-1	-1	-1	-1	-1	1
----------	----	----	----	----	----	----------

srTable:

8	3	1
1	3	5

state

6



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 2:
 - abcbaabcba

srMap:

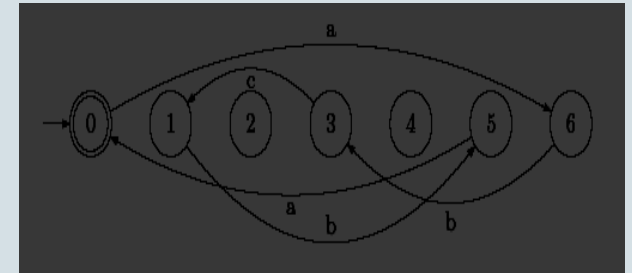
0	-1	-1	-1	-1	-1	1
---	----	----	----	----	----	---

srTable:

8	3	1
1	3	5

state

3



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 3:
– **abcbaabcba**

srMap:

0	-1	-1	2	-1	-1	1
----------	----	----	----------	----	----	----------

srTable:

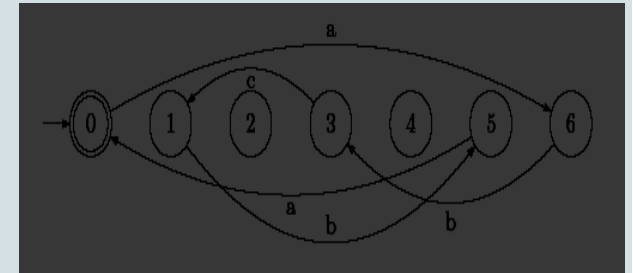
8	3	1
1	3	5
3	2	1

Already visited states must be updated

State 3 must be updated to $2+7=9$

state

3



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 3:
 - **a**bcbaabcba

srMap:

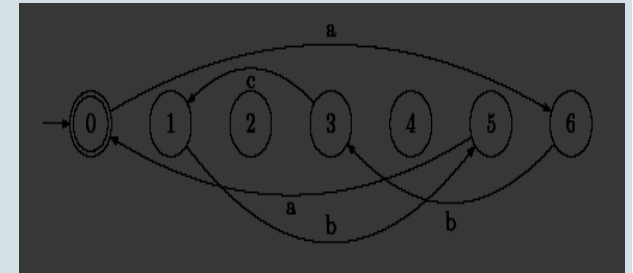
0	-1	-1	2	-1	-1	1
----------	----	----	----------	----	----	----------

srTable:

8	9	1
1	9	5
9	2	1

state

3



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

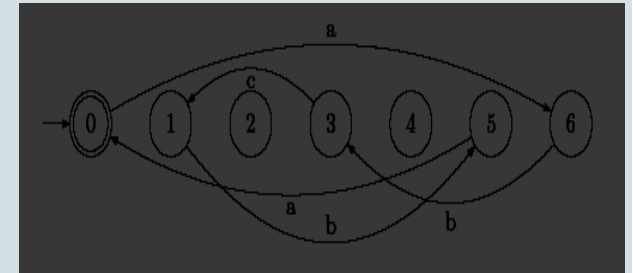
- Iteration 3:
 - **abcbaabcba**

srMap:

0	-1	-1	2	-1	-1	1
----------	----	----	----------	----	----	----------

srTable:

8	9	1
1	9	5
9	2	1



state

1

Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 4:
 - abcbaabcba

srMap:

0	3	-1	2	-1	-1	1
---	---	----	---	----	----	---

srTable:

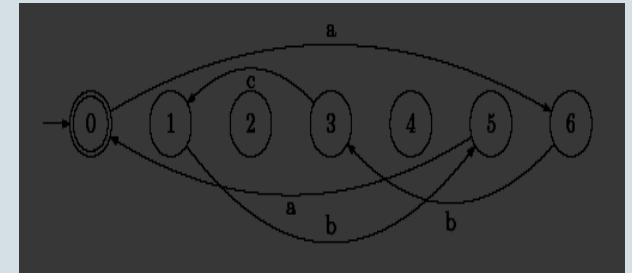
8	9	1
1	9	5
9	2	1
2	5	4

Already visited states must be updated

State 1 must be update to $3+7 = 10$

state

1



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 4:
 - **abcbaabcba**

srMap:

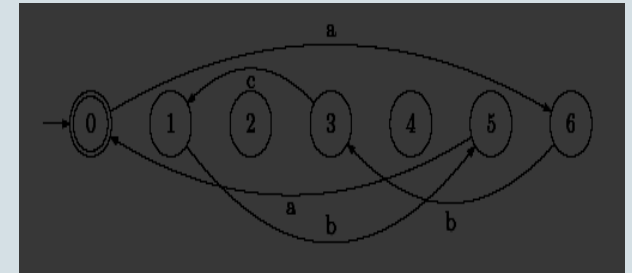
0	3	-1	2	-1	-1	1
----------	----------	----	----------	----	----	----------

srTable:

8	9	10
10	9	5
9	2	10
2	5	4

state

1



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 4:
 - **abcbaabcba**

srMap:

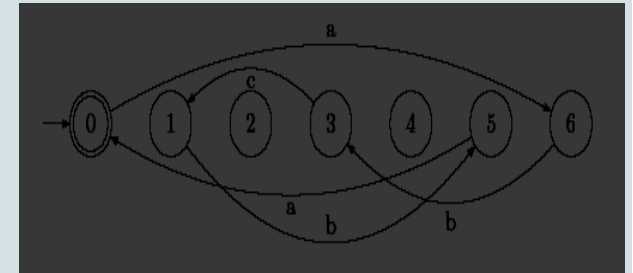
0	3	-1	2	-1	-1	1
---	---	----	---	----	----	---

srTable:

8	9	10
10	9	5
9	2	10
2	5	4

state

5



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 5:
 - **abcbaabcba**

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

srTable:

8	9	10
10	9	5
9	2	10
2	5	4
0	1	3

Already visited states must be updated

State 5 update to $4+7 = 11$

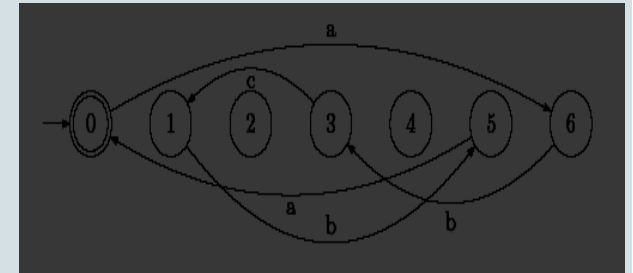
State 1 updated to $3+7 = 10$

State 3 updated to $2+7 = 9$

State 0 updated to $0+7 = 7$

state

5



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 5:
 - **abcbaabcba**

srMap:

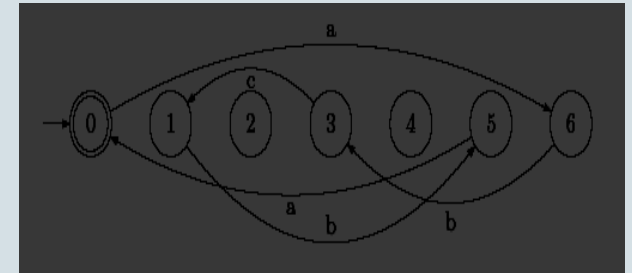
0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state

5



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 5:
 - abcbaabcba

srMap:

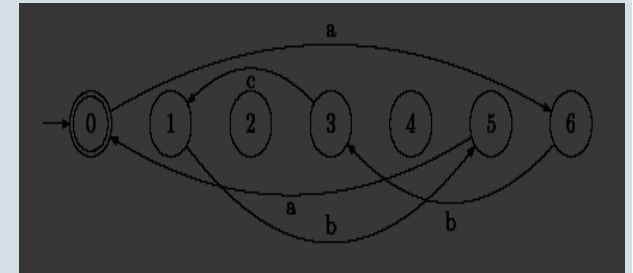
0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state

7



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 6:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

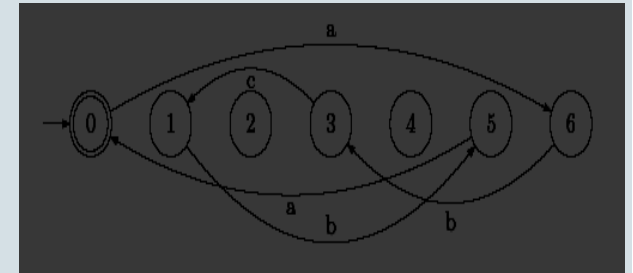
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state = 7 @ srTable[7-7] ie pos = 0

state

7



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 6:
 - abcbaabcba

srMap:

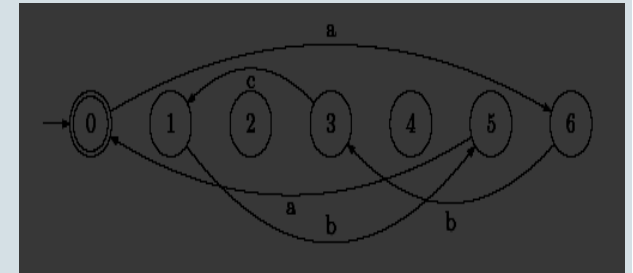
0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state

8



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 7:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

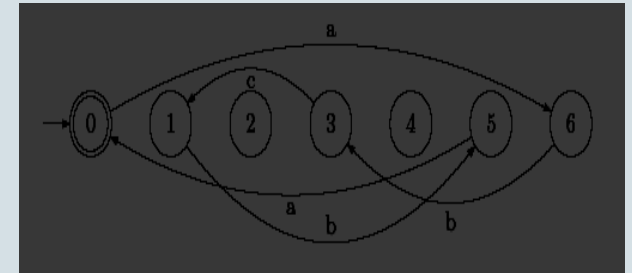
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state = 8 @ srTable[8-7] ie pos = 1

state

8



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 7:
 - abcbaabcba

srMap:

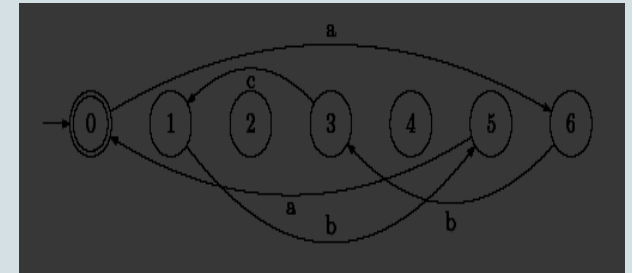
0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state

9



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 8:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

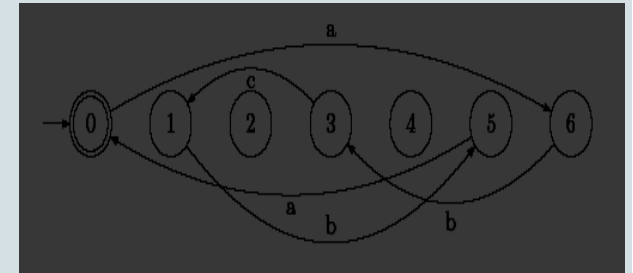
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state = 9 @ srTable[9-7] ie pos = 2

state

9



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 8:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

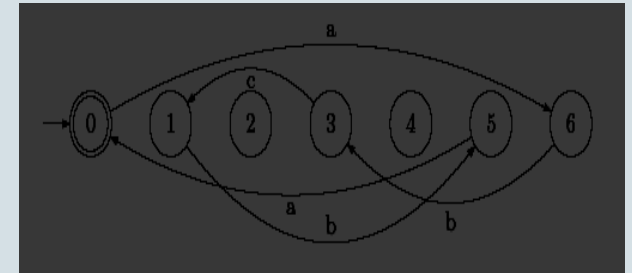
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9



state

10



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 9:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

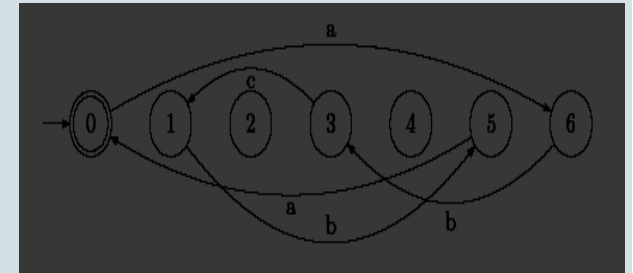
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state = 10 @ srTable[10-7] ie pos = 3

state

10



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 9:
 - abcbaabcba

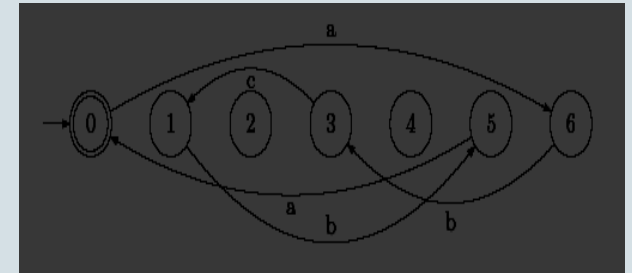
srMap: 0 3 -1 2 -1 4 1

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9



state 11



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 10:
– **abcbaabcba**

srMap:

0	3	-1	2	-1	4	1
----------	----------	----	----------	----	----------	----------

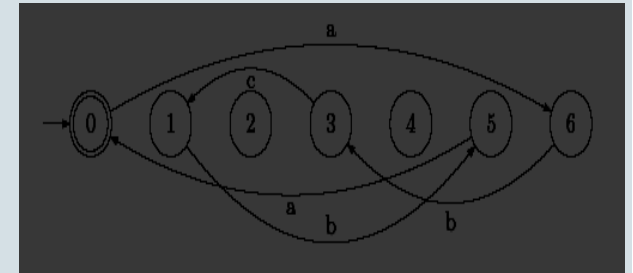
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state = 11 @ srTable[**11**-7] ie pos = 4

state

11



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

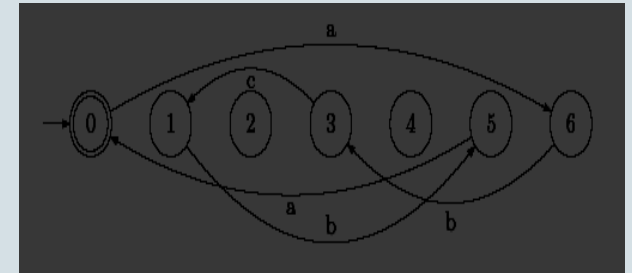
- Iteration 10:
 - abcbaabcba

srMap: 0 3 -1 2 -1 4 1

srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

state 7



Illustrative example

table

	a	b	c
0	6	3	1
1	2	5	4
2	1	1	2
3	3	2	1
4	4	6	0
5	0	1	3
6	1	3	5

- Iteration 11:
 - abcbaabcba

srMap:

0	3	-1	2	-1	4	1
---	---	----	---	----	---	---

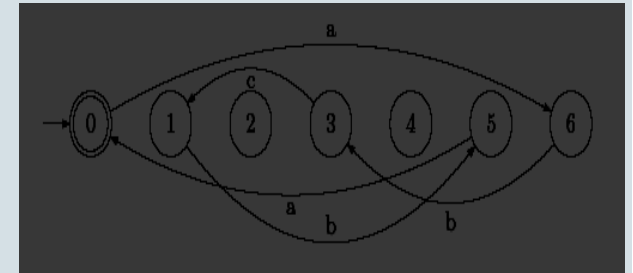
srTable:

8	9	10
10	9	11
9	2	10
2	11	4
7	10	9

Final State, no more symbols

state

7

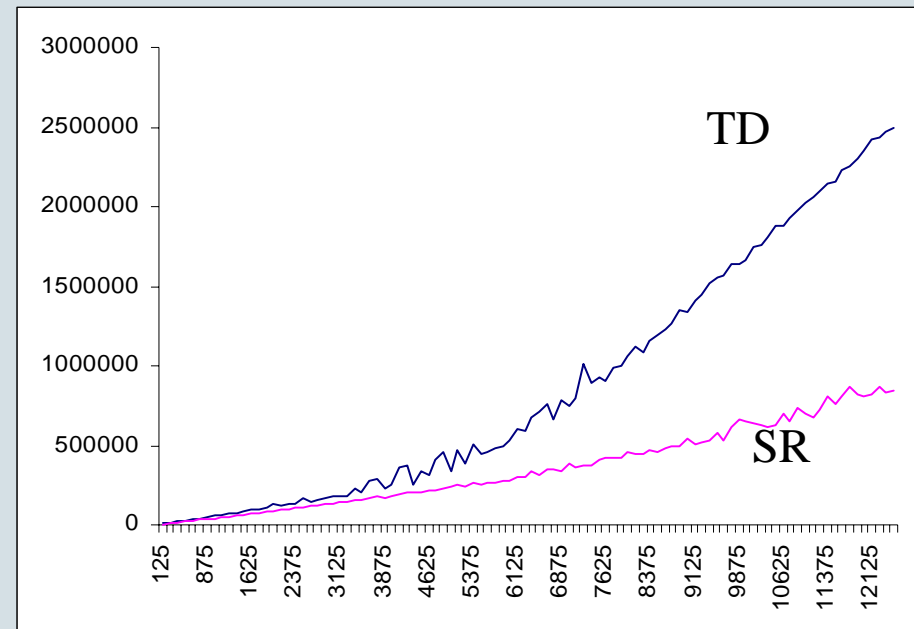


Theoretical assessment

- Time complexity: $O(|inString| \times n \times a)$
 - Cost for reordering: $O(n \times a)$
 - At hot-spot, time complexity: $O(|inString|)$
- Suitable for strings made of long sequences
 - Visiting a limited number of states
 - At hot-spot:
 - Enjoys Spatial locality of reference
 - Low probability of cache misses

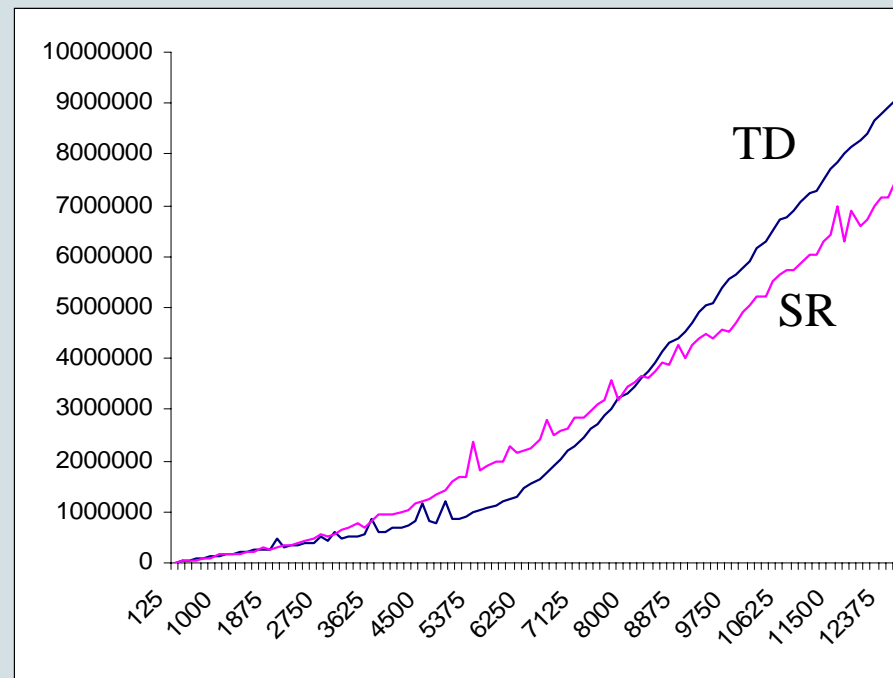
Experimental Results

- Strings of the form $s = S_0 k S_0 k$, $S_0 = s_1 s_2 \dots s_m$
 - k triggers a transition back to the final state 0
 - $|S| = 2n$
 - Without reordering cost



Experimental Results

- Strings of the form $s = S_0kS_0kS_0kS_0k$, $S_0 = s_1s_2\dots s_m$,
 k triggers a transition back to the final state 0
 - $|S| = 4n$
 - With reordering cost
 - Optimized TD and SR



Conclusion & Future Work

- SR outperforms TD for strings made of long sequences
 - Due to spatial locality of reference
- Application on real life data:
 - Network Intrusion Detection Systems
 - Tandem Repeat / DNA Analyzers
- Many variations of string recognizers to explore
 - Framework for the dynamic implementation of FAs