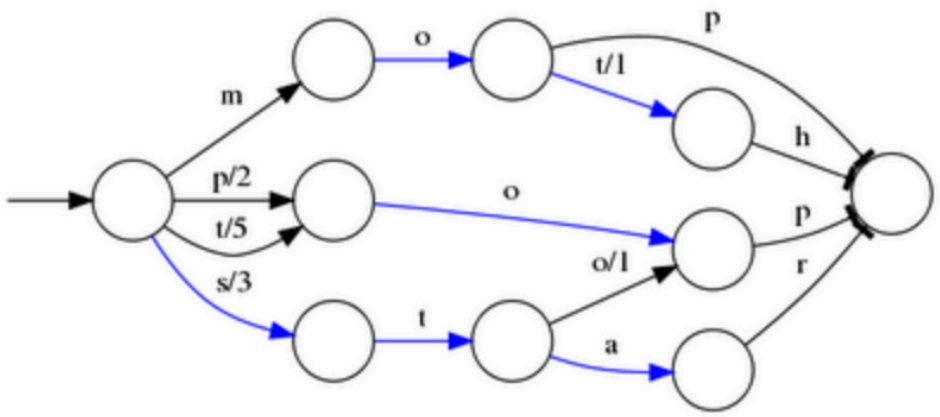
Friday, December 3, 2010

Using Finite State Transducers in Lucene

FSTs are finite-state machines that map a term (byte sequence) to an arbitrary output. They also look cool:

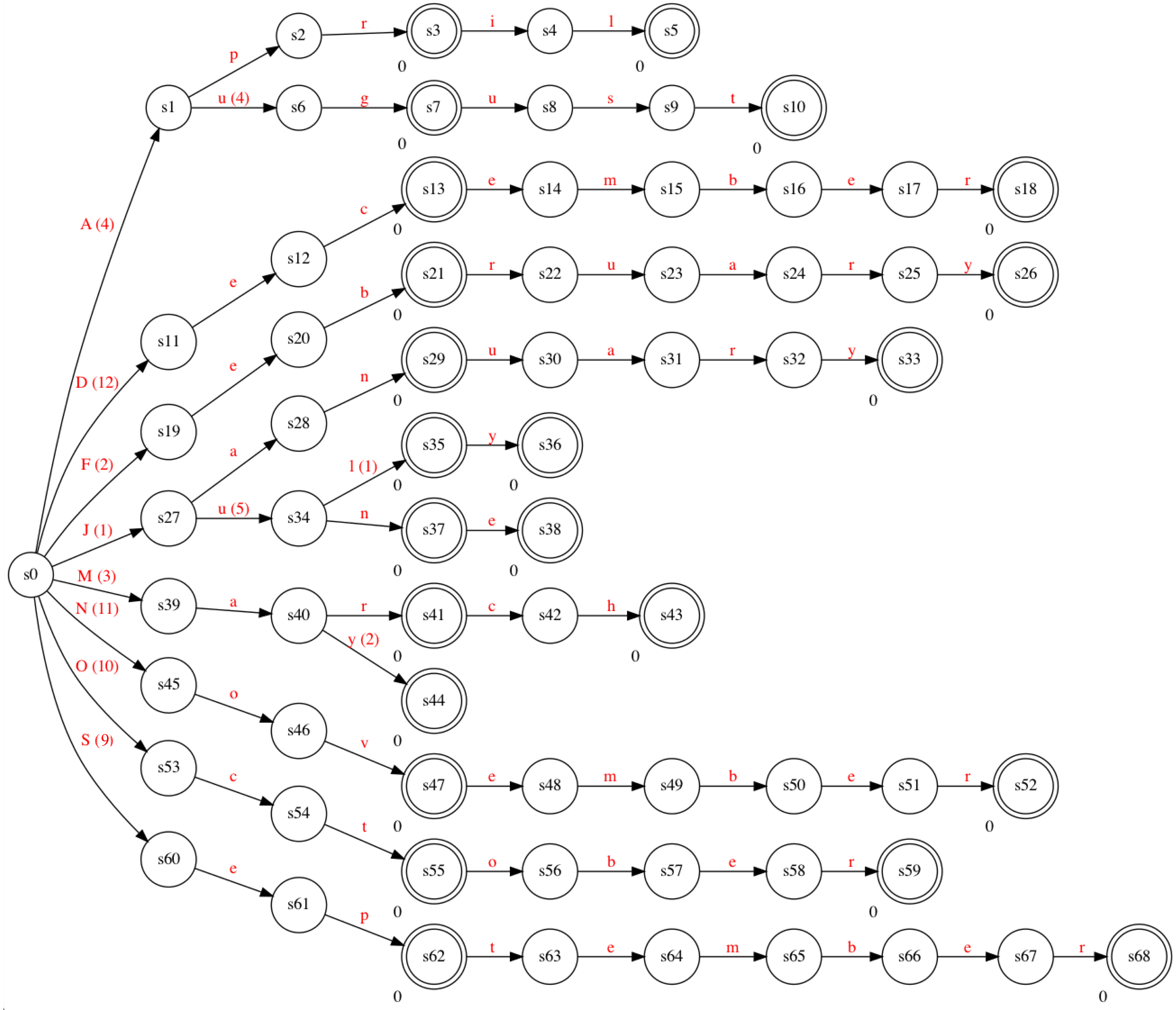


That FST maps the sorted words *mop*, *moth*, *pop*, *star*, *stop* and *top* to their ordinal number (0, 1, 2, ...). As you traverse the arcs, you sum up the outputs, so *stop* hits 3 on the **s** and 1 on the **o**, so its output ordinal is 4. The outputs can be arbitrary numbers or byte sequences, or combinations, etc. -- it's pluggable.

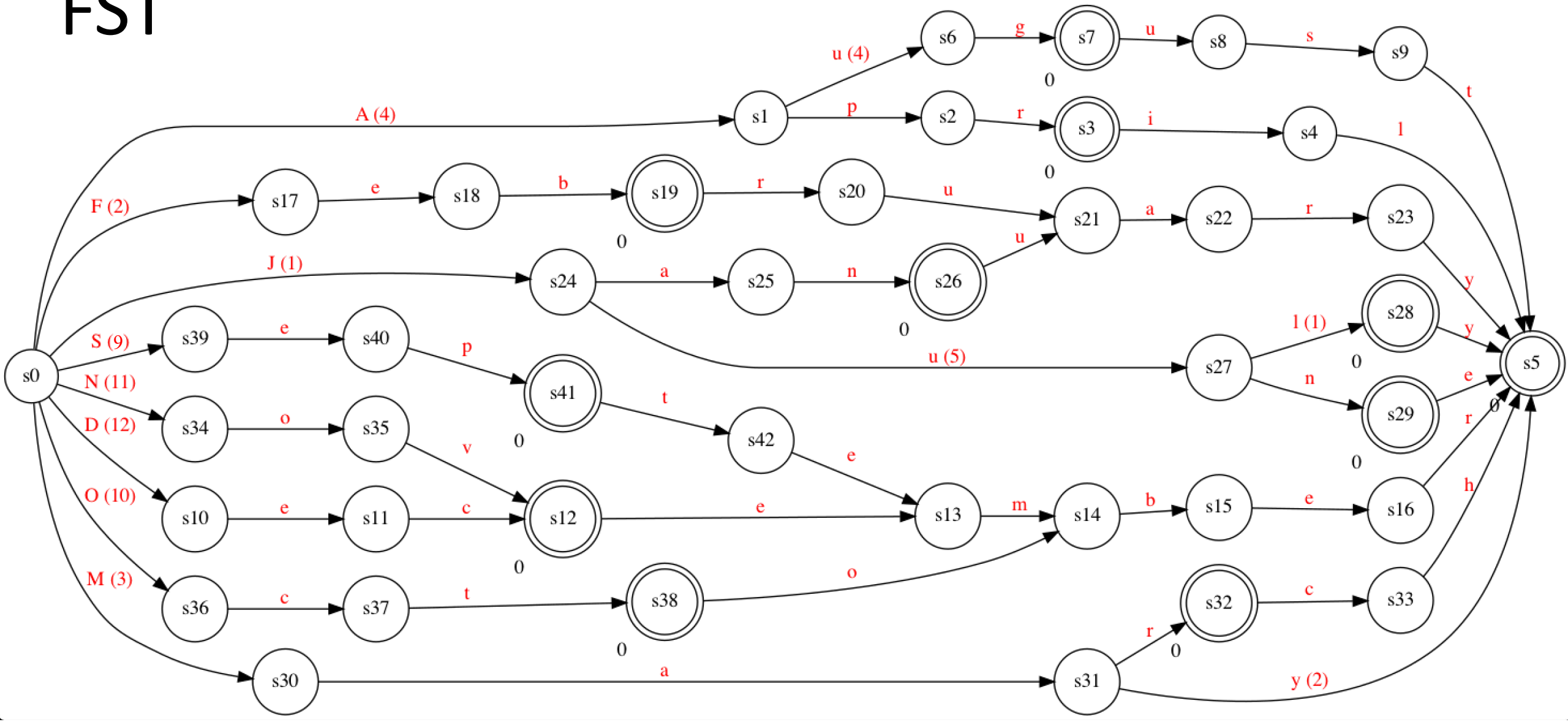
Essentially, an FST is a SortedMap<ByteSequence,SomeOutput>, if the arcs are in sorted order. With the right representation, it requires far less RAM than other SortedMap implementations, but has a higher CPU cost during lookup. The low memory footprint is vital for Lucene since an index can easily have many millions (sometimes, billions!) of unique terms.

January, 1
Jan, 1
February, 2
Feb, 2
March, 3
Mar, 3
April, 4
Apr, 4
May, 5
June, 6
Jun, 6
July, 7
Jul, 7
August, 8
Aug, 8
September, 8
Sep, 8
October, 10
Oct, 10
November, 11
Nov, 11
December, 12
Dec,12

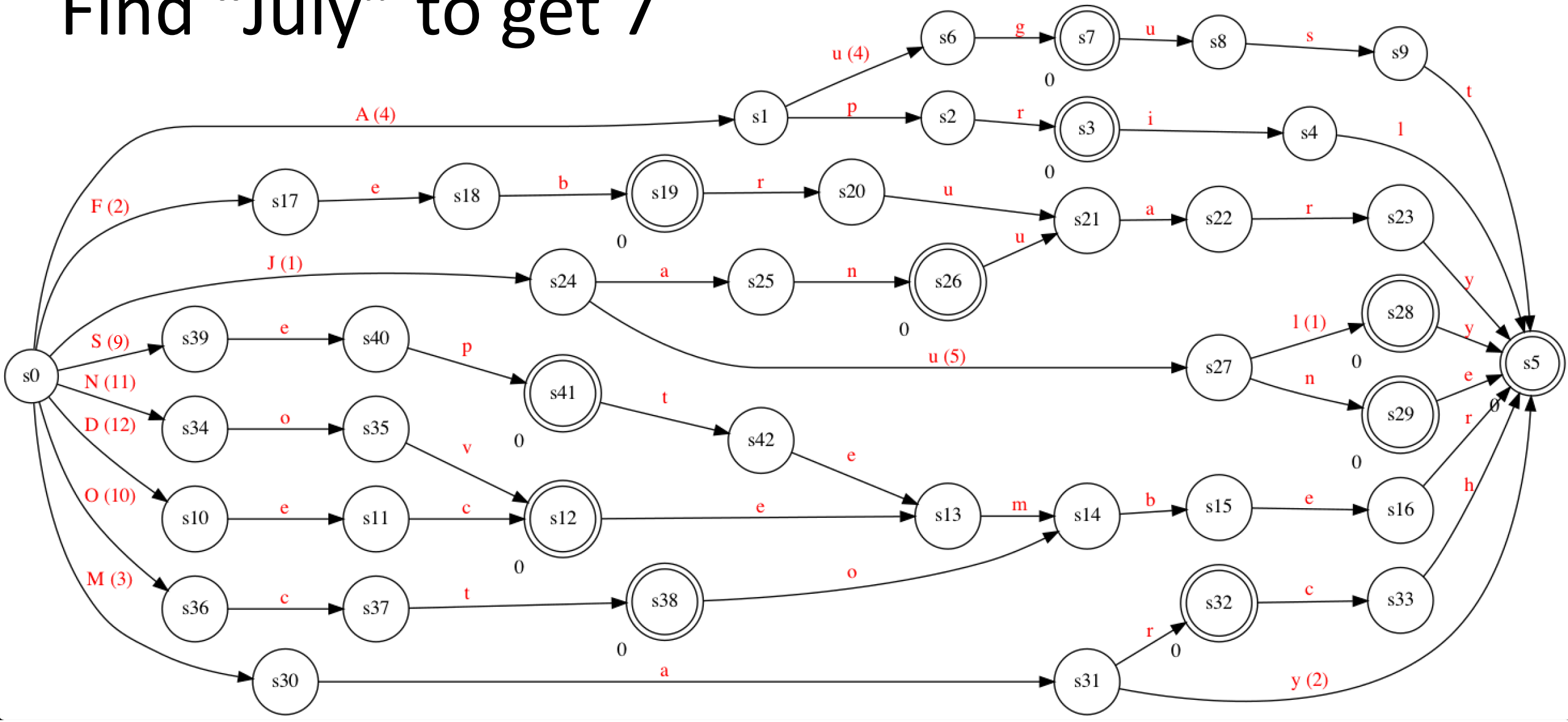
Trie


s0
A (4)
s1
p
s2
r
s3
i
s4
l
s5
0
0
u (4)
s6
g
s7
u
s8
s
s9
t
s10
0
0
D (12)
s11
e
s12
c
s13
e
s14
m
s15
b
s16
e
s17
r
s18
0
0
F (2)
s19
e
s20
b
s21
r
s22
u
s23
a
s24
r
s25
y
s26
0
0
J (1)
s27
a
s28
n
s29
u
s30
a
s31
r
s32
y
s33
0
0
u (5)
s34
l (1)
s35
y
s36
0
0
n
s37
e
s38
0
0
M (3)
s39
a
s40
r
s41
c
s42
h
s43
0
0
y (2)
s44
0
N (11)
s45
o
s46
v
s47
e
s48
m
s49
b
s50
e
s51
r
s52
0
0
O (10)
s53
c
s54
t
s55
o
s56
b
s57
e
s58
r
s59
0
0
S (9)
s60
e
s61
p
s62
t
s63
e
s64
m
s65
b
s66
e
s67
r
s68
0
0

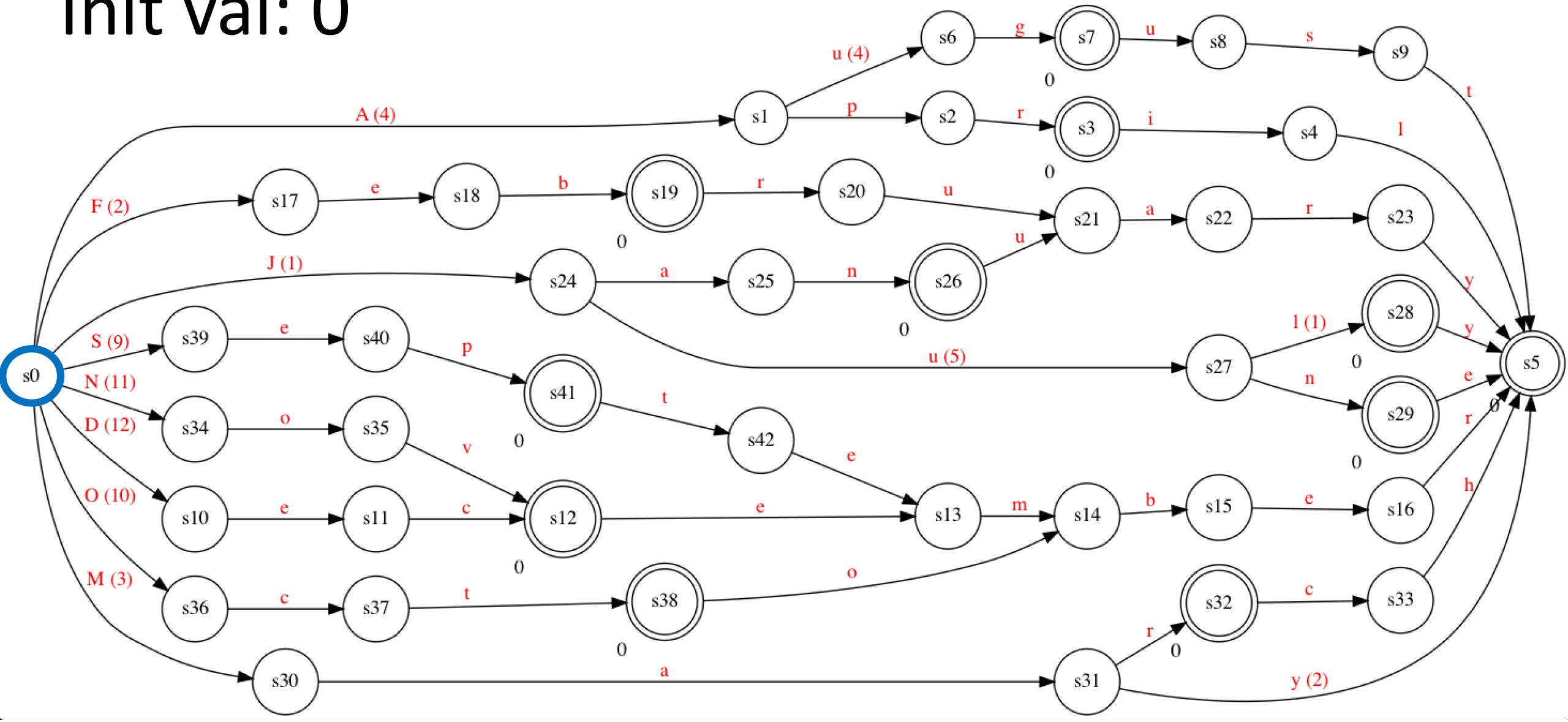
FST


s0
A (4)
s1
u (4)
s6
g
s7
u
s8
s
s9
t
p
s2
r
s3
i
s4
l
F (2)
s17
e
s18
b
s19
r
s20
u
s21
a
s22
r
s23
y
J (1)
s24
a
s25
n
s26
u
S (9)
s39
e
s40
p
s41
t
s42
e
s27
l (1)
s28
y
u (5)
s5
N (11)
s34
o
s35
v
n
s29
e
D (12)
r
O (10)
s10
e
s11
c
s12
e
s13
m
s14
b
s15
e
s16
h
M (3)
s36
c
s37
t
s38
o
s32
c
s33
s30
a
s31
r
y (2)

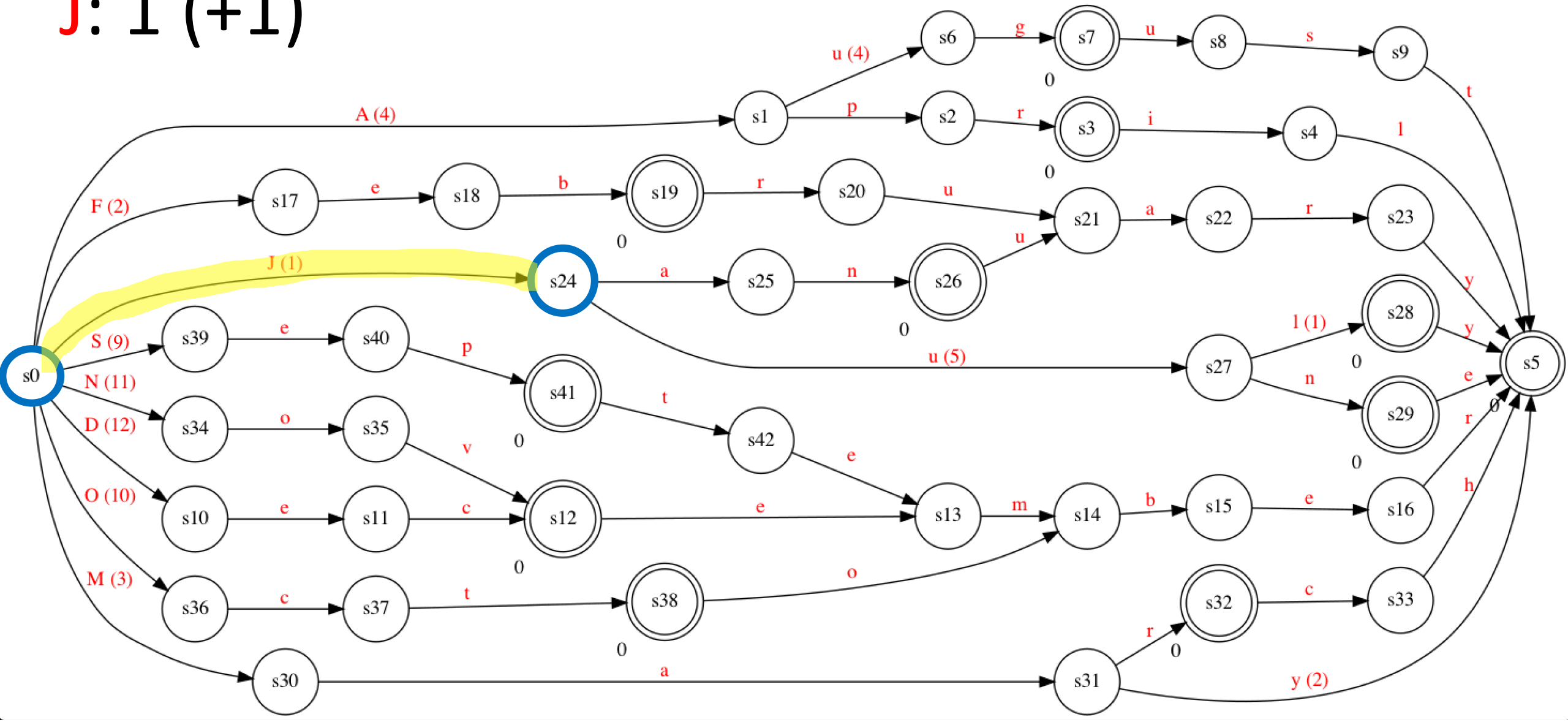
Find “July” to get 7



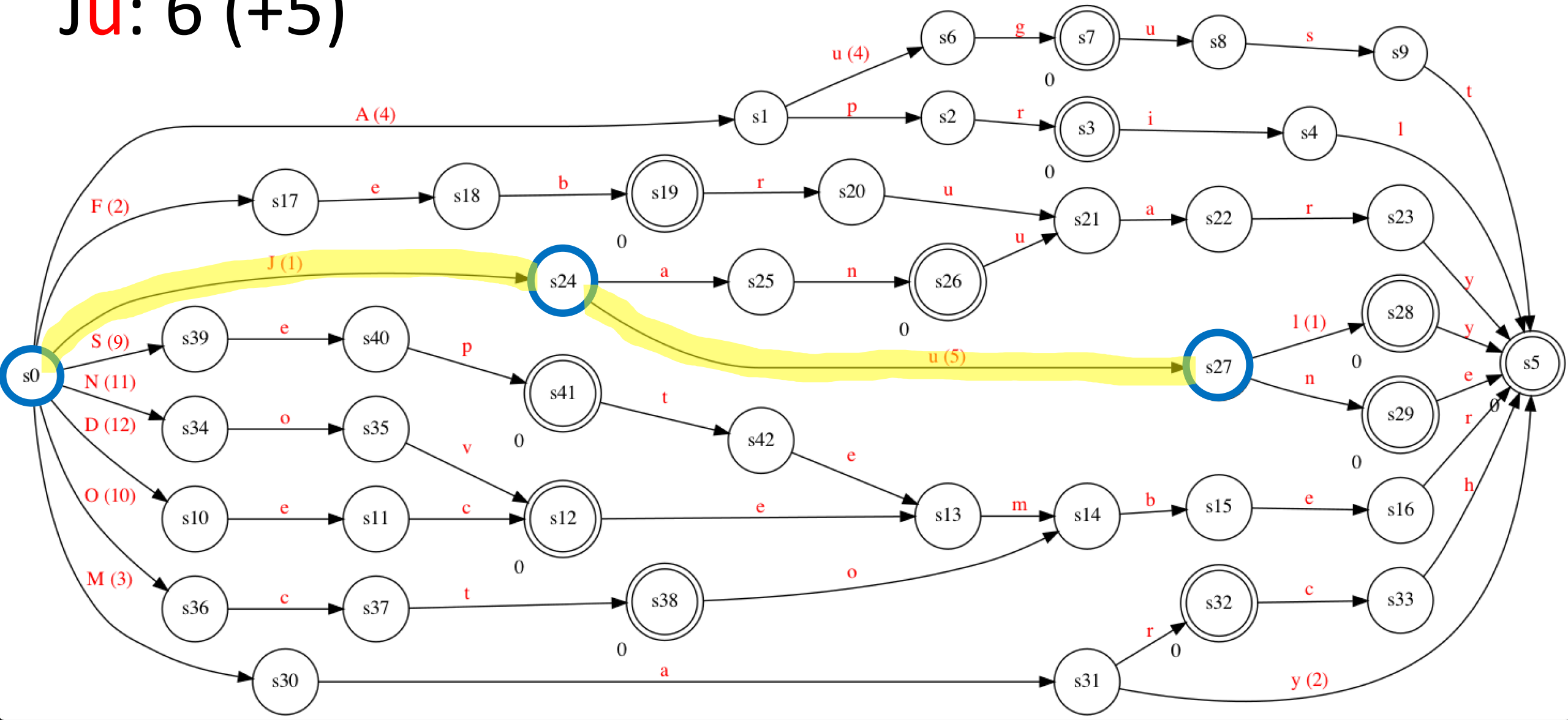
Init val: 0


s0
A (4)
s1
u (4)
s6
g
s7
u
s8
s
s9
t
p
s2
r
s3
i
s4
l
F (2)
s17
e
s18
b
s19
r
s20
u
s21
a
s22
r
s23
y
J (1)
s24
a
s25
n
s26
u
S (9)
s39
e
s40
p
s41
t
s42
e
s27
l (1)
s28
y
u (5)
s5
N (11)
s34
o
s35
v
n
s29
e
D (12)
r
O (10)
s10
e
s11
c
s12
e
s13
m
s14
b
s15
e
s16
h
M (3)
s36
c
s37
t
s38
o
s32
c
s33
s30
a
s31
r
y (2)

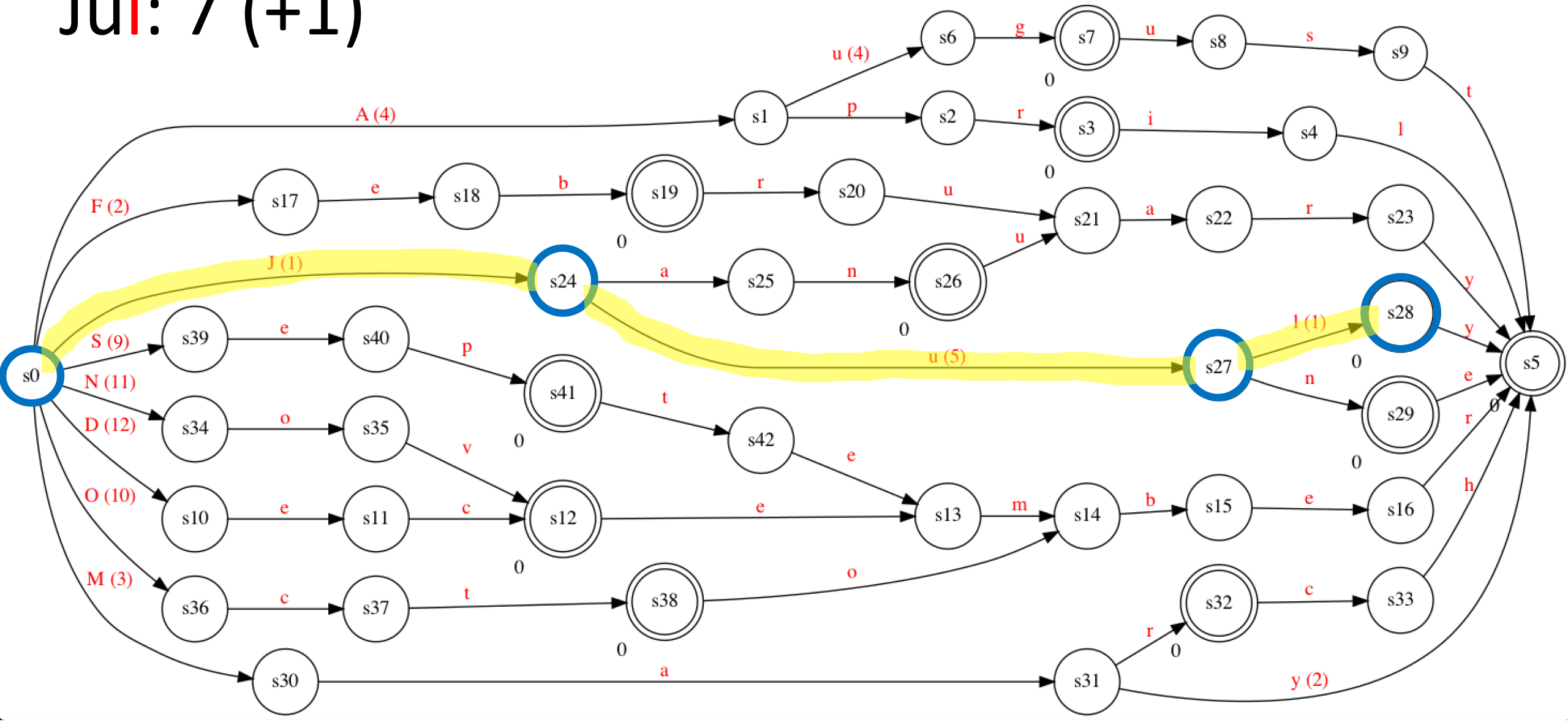
J: 1 (+1)


s0
A (4)
F (2)
J (1)
S (9)
N (11)
D (12)
O (10)
M (3)
s1
s2
s3
s4
s5
s6
s7
s8
s9
u (4)
p
r
i
t
l
g
u
s
0
s17
s18
s19
s20
s21
s22
s23
e
b
r
u
a
r
y
s24
s25
s26
a
n
u
u (5)
s27
s28
s29
l (1)
n
y
e
r
h
s39
s40
s41
s42
s34
s35
s10
s11
s12
s13
s14
s15
s16
s36
s37
s38
s30
s31
s32
s33
o
v
t
e
m
b
e
c
o
a
r
y (2)

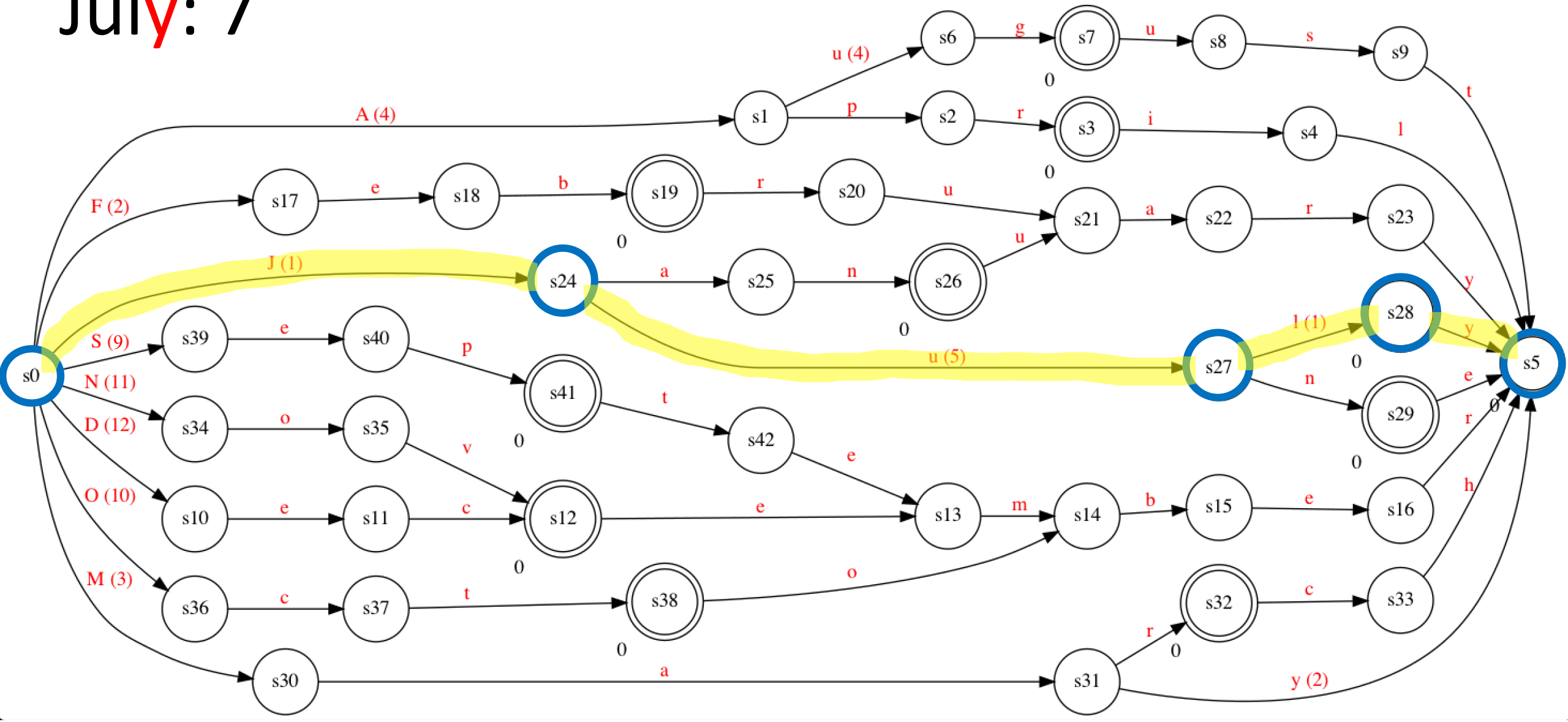
Ju: 6 (+5)


s0
A (4)
F (2)
J (1)
S (9)
N (11)
D (12)
O (10)
M (3)
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13
s14
s15
s16
s17
s18
s19
s20
s21
s22
s23
s24
s25
s26
s27
s28
s29
s30
s31
s32
s33
s34
s35
s36
s37
s38
s39
s40
s41
s42
u (4)
p
r
i
l
g
u
s
t
e
b
r
u
a
r
y
J (1)
a
n
u
S (9)
e
p
N (11)
o
D (12)
v
O (10)
e
c
M (3)
c
t
t
e
e
m
b
e
r
h
o
a
y (2)
l (1)
n
0

Jul: 7 (+1)


s0
s1
s2
s3
s4
s5
s6
s7
s8
s9
s10
s11
s12
s13
s14
s15
s16
s17
s18
s19
s20
s21
s22
s23
s24
s25
s26
s27
s28
s29
s30
s31
s32
s33
s34
s35
s36
s37
s38
s39
s40
s41
s42
A (4)
F (2)
J (1)
S (9)
N (11)
D (12)
O (10)
M (3)
u (4)
u (5)
l (1)
y (2)
0

July: 7


s0
A (4)
s1
u (4)
s6
g
s7
u
s8
s
s9
t
p
s2
r
s3
i
s4
l
F (2)
s17
e
s18
b
s19
r
s20
u
s21
a
s22
r
s23
y
J (1)
s24
a
s25
n
s26
u
S (9)
s39
e
s40
p
s41
t
s42
e
s27
l (1)
s28
y
s5
u (5)
N (11)
s34
o
s35
v
n
s29
e
D (12)
s10
e
s11
c
s12
e
s13
m
s14
b
s15
e
s16
r
h
O (10)
M (3)
s36
c
s37
t
s38
o
s32
c
s33
r
s30
a
s31
y (2)

Tree Structure

Contains pointers and containers

Can’t serialize it easily to a file…

```
// Tree Structure
struct State {
  struct Transition {
    char arc;
    State* state;
    uint32_t output;
  }

  bool final;
  std::vector<Transition> transitions;
}

State* root;

// Byte Code
struct Ope {
  char arc;
  bool last_transition;
  bool final;
  uint32_t next_address;
  uint32_t output;
};

std::vector<Ope> byte_code;
```

Byte Code

Just primitive values

Can directly save it to file
Zero loading time!

Byte code

```
[master] ~/Projects/cpp-fstlib/demo$ xxd months.fst
00000000: 806c 0669 0347 8074 0673 0363 6707 846b  .l.i.G.t.s.cg..k
00000010: 8b70 0080 4623 c36d 0323 e723 8079 0643  .p..F#.m.#.#.y.C
00000020: 8363 43c7 2385 626e 0780 7906 8026 6e07  .cC.#.bn..y..&n.
00000030: 8185 6c0c 8c82 8583 6880 6806 e382 8079  ..l.....h.h....y
00000040: 0e85 4483 ab76 066f 03b3 6f02 7407 e3b7  ..D..v.o..o.t...
00000050: 2274 0370 0723 8953 0b8a 8b4f 088b 954e  "t.p.#.S...O...N
00000060: 0883 9e4d 0881 ada8 82c4 4608 8cd1 4408  ...M......F...D.
00000070: 84de 4108 0004 080c 0f13 171b 88fe 0000  ..A.............
00000080: 0100 7d00 0000 0065 7275 614a 6263 0000  ..}....eruaJbc..
00000090: 0000 0000 0000                           ......
```

Byte code dump

Byte code records are stored in backward order!

N: No next address. Just move to next record
F: Final state (or ‘accepted’)
L: Last transition in a state

```
[master] ~/Projects/cpp-fstlib/demo$ ./fst dump months -f csv
Address Arc     N F L   NxtAddr Output  StOuts  Size
------- ---     ----- - ------- ------  ------  ----
2       l         * -   x                       3
4       i       ↑   -                           2
5       r       ↑ * -                           1
8       t         * -   x                       3
10      s       ↑   -                           2
11      u       ↑   -                           1
13      g       ↑ * -                           2
15      u       ↑   -           4               2
18      p                 5                     3
20      r         * -   x                       2
21      e       ↑   -                           1
22      b       ↑   -                           1
24      m       ↑   -                           2
25      e       ↑   -                           1
26      c       ↑ * -                           1
27      e       ↑   -                           1
30      y         * -   x                       3
31      r       ↑   -                           1
32      a       ↑   -                           1
33      u       ↑   -                           1
34      r       ↑   -                           1
35      b       ↑ * -                           1
36      e       ↑   -                           1
38      u                 32                    2
40      n       ↑ *                             2
43      y         *   x                         3
45      e         *   x                         2
47      n       ↑ *                             2
51      l         *   43      1                 4
53      a                 40                    2
56      u                 51      5             3
59      h         * -   x                       3
60      c       ↑   -                           1
64      y         * -   x       2               4
66      e         *             60              2
67      a       ↑   -                           1
70      v         * -   25                      3
72      o       ↑   -                           2
75      o                 22                    3
77      t       ↑ * -                           2
78      c       ↑   -                           1
80      e                 24                    2
82      t       ↑   -                           2
84      p       ↑ * -                           2
85      e       ↑   -                           1
88      S       ↑   -           9               3
92      O                 78      10            4
96      N                 72      11            4
100     M                 67      3             4
103     J                 56      1             3
107     F                 36      2             4
111     D                 27      12            4
125     A                 18      4             14
```

FST (Finite State Transducer) library

Trie-like data structure

O(M) access like std::unordered_map
Common prefix match (‘computer programming is fun.’ returns ‘computer’, ‘computer programming’)

Read-only binary database

Zero loading time when lookup
Memory-mapped file friendly
Compact (‘/usr/shared/dict/words’: 2.4M → 1.1M)

Examples that fstlib can do, but std::unordered_map cannot effectively

Keyword extraction (‘Longest common prefix search’)
Word/phrase list dictionary for full-text search library (‘Zero loading time’, ‘Compact’)

cpp-fstlib internals

Compile a source dictionary to byte code, then interpret it when searching
Incremental direct database construction without having a complete set of states on memory
Implemented based on https://blog.burntsushi.net/transducers/. Thanks for the valuable information!