Fighting Adversarial Regular Expressions in Apache Lucene™

Patrick Zhai

About me

- Lucene committer since 2021
- Work for LinkedIn search infra
- Email: zhai7631@gmail.com

Overview

- Introductions
- How does Lucene search with regular expressions (regex)
- Some problems we had with adversarial regex
- Improvements on protection of adversarial regex
- A new way of regex search

Apache Lucene™

Apache Lucene is a free and open-source search engine software library, originally written in Java by Doug Cutting.[0]

Supports multiple types of queries: term query, phrase query, boolean query, geo distance query, knn query, regexp query etc.

[0]: https://en.wikipedia.org/wiki/Apache_Lucene

Regular Expression (Regex)

- What is regex?
- How do we search using regex?

What is regex?

A regular expression (shortened as regex or regexp) is a sequence of characters that specifies a match pattern in text.[0]

^[\w!#$%&'*+/=?`{|}~^-]+(?:\.[\w!#$%&'*+/=?`{|}~^-]+)*@(?:[A-Z0-9-]+\.)+[A-Z]{2,6}$ [1]

Lucene's syntax: RegExp javadoc

[0]: https://en.wikipedia.org/wiki/Regular_expression

[1]: https://www.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ch04s01.html

How to search

Regex: **.*ware**

Doc 0: Patrick is a software engineer

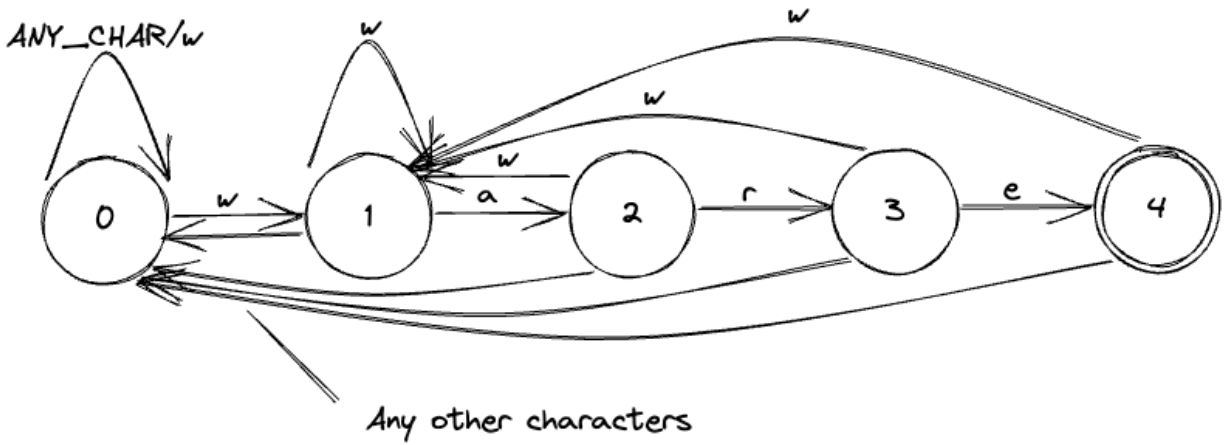
Doc 1: Sam is a hardware engineer

engineer	0,1
hardware	1
patrick	0
sam	1
software	0

How to match

How do we know whether a word, like “software”, matches our regex “.*ware”?

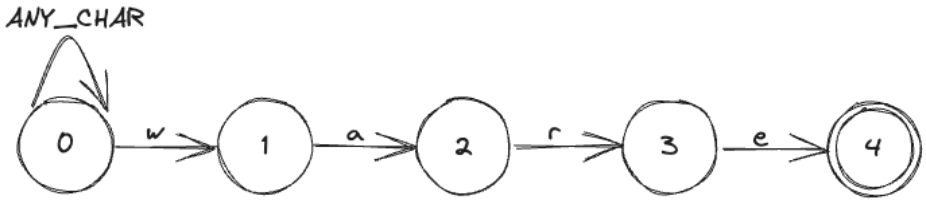
Automaton -



Recap of Regex to Automaton

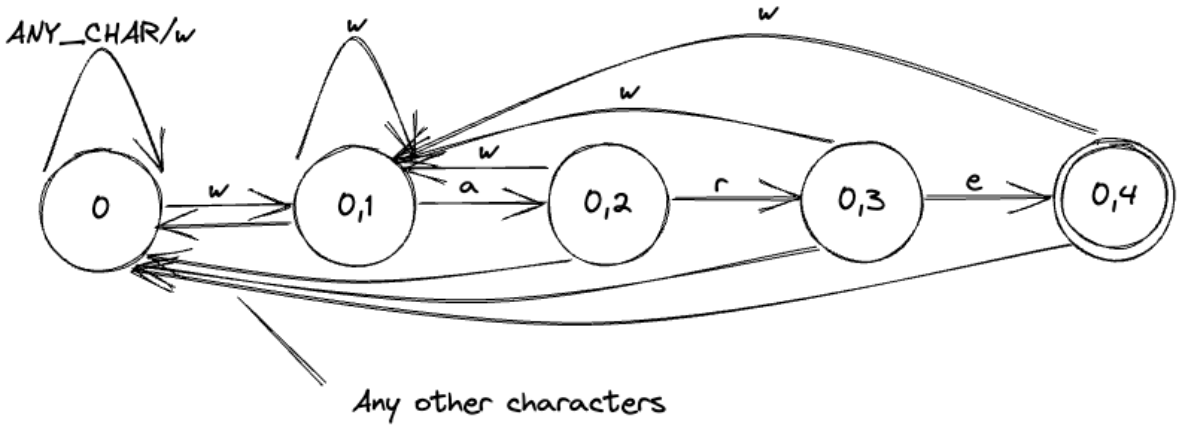
Non-deterministic

Finite Automaton (NFA):

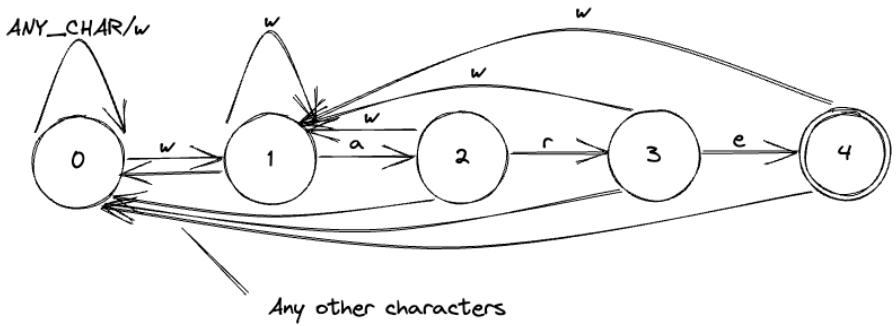


Powerset Construction

To Deterministic Finite Automaton (DFA):



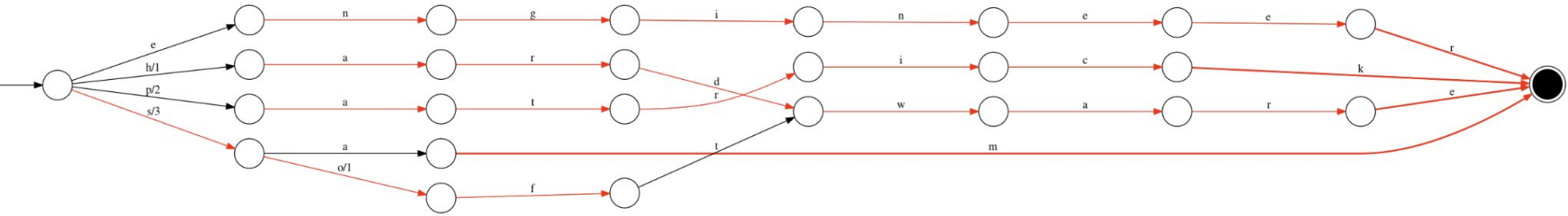
How to intersect with automaton – FST!



engineer	0,1
hardware	1
patrick	0
sam	1
software	0

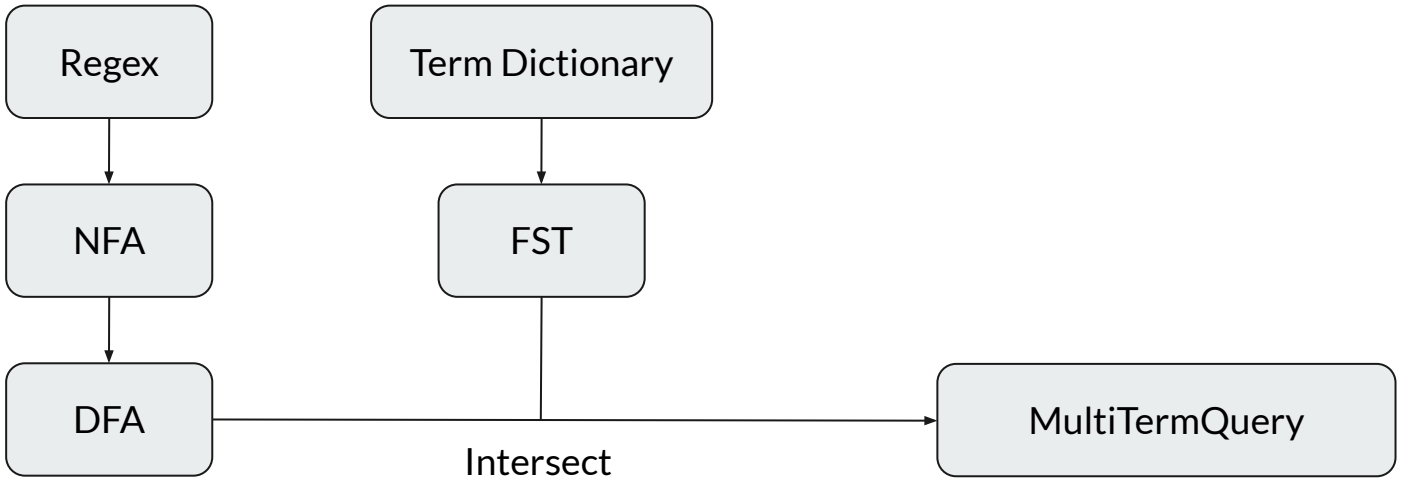
Finite State Transducer (FST)

FST = automaton + output

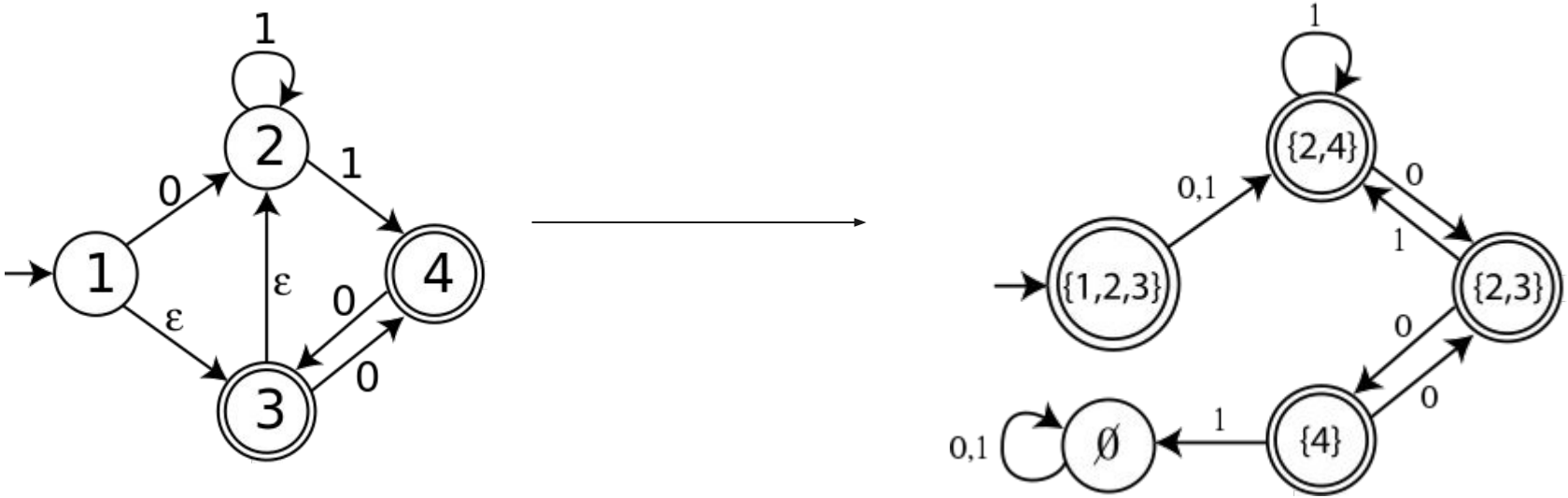


Note: Lucene's default Term Dictionary is not exactly one FST

So far...


Regex
Term Dictionary
NFA
FST
DFA
Intersect
MultiTermQuery

Determinization



Ref: https://en.wikipedia.org/wiki/Powerset_construction

Problem

[BUG] OpenSearch is exposed to ReDoS attack #687

An adversarial regex ("**(.*a){2000}**") took 268s to hit the TooComplexToDeterminizeException, which is because of:

1. One of the optimization for leading wildcard query – get common suffix, is determinizing the automaton, and the process – as shown previously is costing exponentially.
2. When mapping {NFA Nodes} -> DFA Node, we sort all NFA nodes before calculate the hash
3. We only guard on max number of final states created

Improvements on detecting adversarial regexps

1. Only perform get common suffix optimization when the automaton is sufficiently small, also improved the process such that no determinize is needed in this optimization anymore.
2. Lazily sort the NFA node set such that sorting overhead is minimized
3. Guard on the net work the automaton is costing (total length of NFA node sets) rather than the max number of states generated.

We now can throw an exception in 200ms for regex (.*a){2000} (tested on my local laptop)

But Still...

Why do we need to convert NFA to DFA to run the automaton?

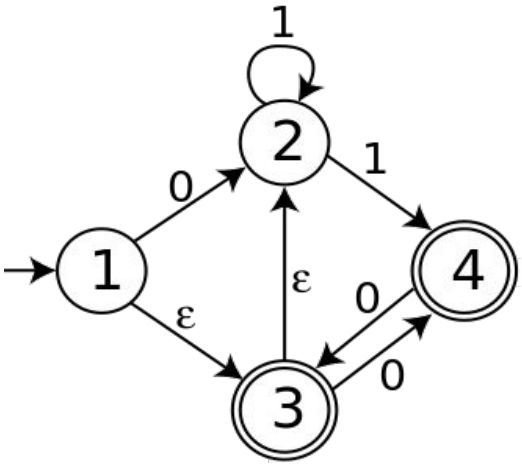
Can we possibly run the adversarial regex?

Regular Expression Matching Can be Simple And Fast (Russ Cox)[1]

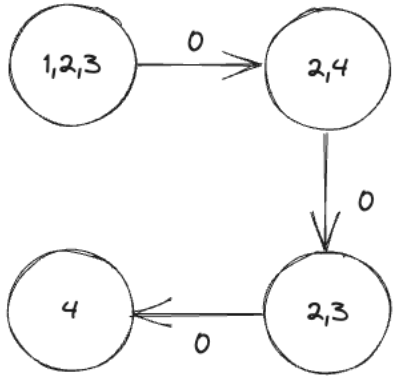
[1]: https://swtch.com/~rsc/regexp/regexp1.html

Run directly on NFA

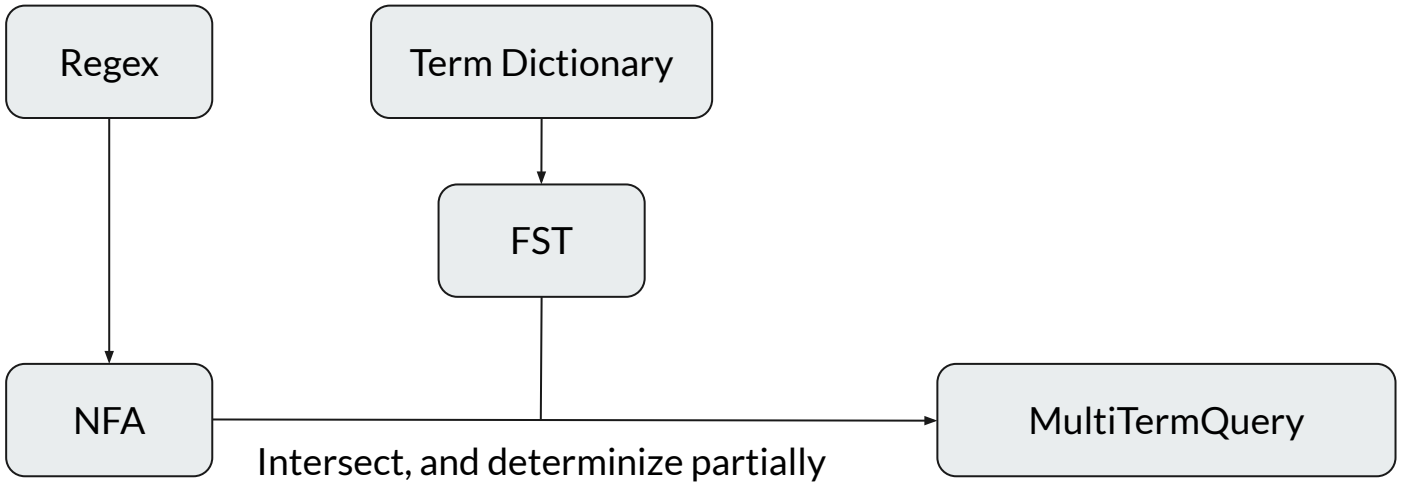
- Keep track of all possible states while running the automaton
- Remember all the states that are visited to avoid recomputation
- Essentially doing powerset construction lazily!
- Class: NFARunAutomaton



Run with “000”



If we run with NFA...



Pros & Cons

Pros:

- You pay the price when you need — Imagine checking “abc(.*a){2000}” against an index where the index contains no terms starting with ‘a’
- No more TooComplexToDeterminizeException are thrown!

Cons:

- Spend more memory because we need to keep the mapping of {NFA_nodes} -> DFA node
- Hard to be executed by multiple threads at the same time.

Future Works

- Release Lucene 10
- Benchmark RegexpQuery better
- Multithreading

...

And more! Contributions are welcomed!

Thank you

Patrick Zhai (zhai7631@gmail.com)