Mirror mirror...

what am I typing next?

A practical introduction into auto suggest

Alexander Reelsen

alr@spinscale.de | @spinscale

Today's goal

> Understand how auto suggest works
> Follow evolution until today

Ask questions... all the time

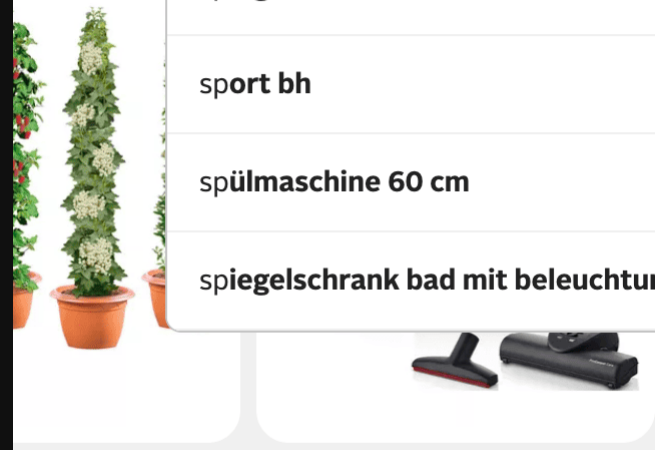
otto.de

sp

- spülmaschine
- spiegel
- sportschuhe damen
- spannbettlaken
- sportschuhe herren
- sporthose damen
- spiegelschrank
- sport bh
- spülmaschine 60 cm
- spiegelschrank bad mit beleuchtung

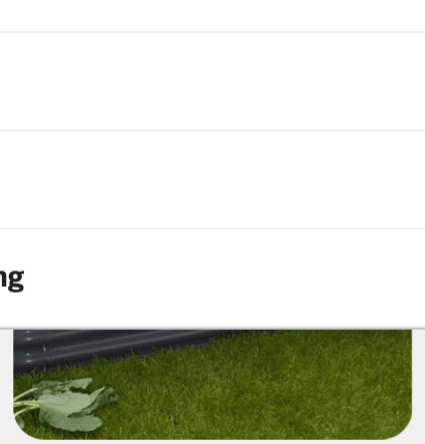
20% auf
nur noch 1

ungen fü

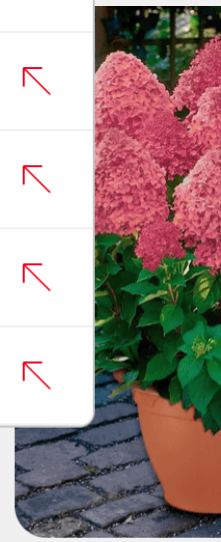


Säulenobst-Set ...
€ 24,99

SIEMENS
Bodenstaubsauger VSP3T21...
UVP ~~€ 229,99~~ € 99,99



KONIFERA
Hochbeet, BxTxH: 240x80x8...
€ 149,99



BCM
Gehölze Hortens
UVP ~~€ 34,99~~ €

Discuss: A good auto suggest?

- Speed
- Relevance
- Order
- Fuzziness
- Individualization
- Navigation & Selection
- Highlighting
- Infix vs. Prefix suggestion

Sample dataset

- wakeboard
- washing machine
- washington wizards basketball
- water glass
- wax crayon
- werewolf mask
- wool socks

Most naive implementation?

- Walk through dataset
- For each term: check if starts with input
- Collect until enough matches found

```
List<String> suggest = new ArrayList<>();

@Test
public void testSimpleSuggestions() {
    suggest.addAll(List.of("wakeboard",
            "washing machine",
            "washington wizards basketball",
            "water glass",
            "wax crayon",
            "werewolf mask",
            "wool socks"));
    suggest.sort(String.CASE_INSENSITIVE_ORDER);

    List<String> results = suggest("wa", 10);
    assertThat(results).containsExactly("wakeboard", "washing machine",
            "washington wizards basketball", "water glass", "wax crayon");

    results = suggest("was", 1);
    assertThat(results).containsExactly("washing machine");
}
```

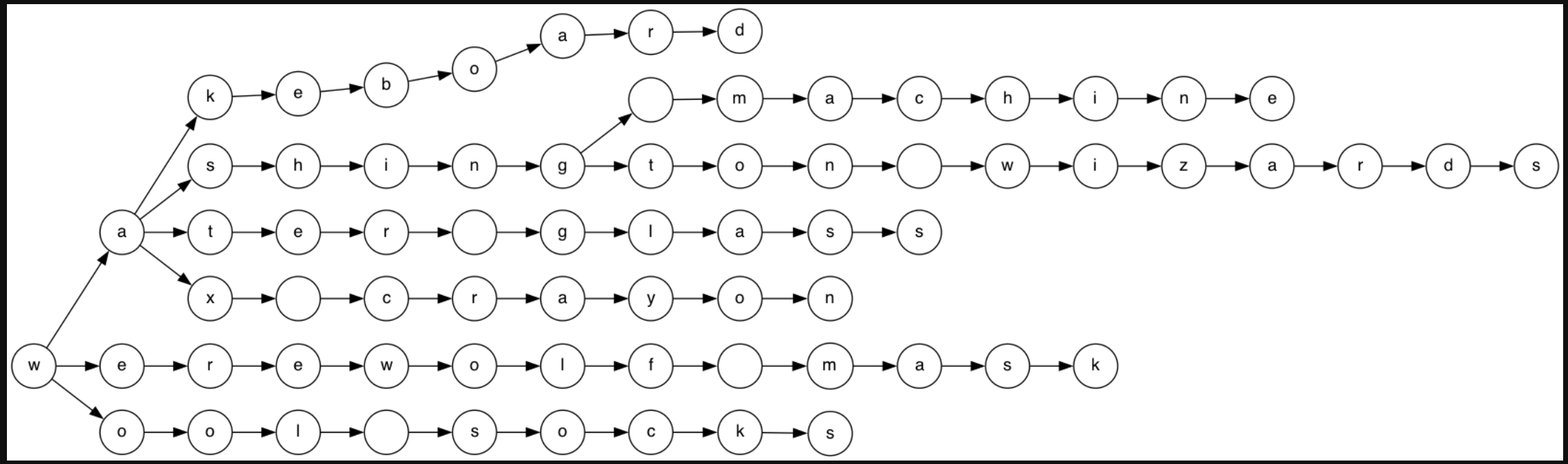
Naive implementation

```
private List<String> suggest(String input, int count) {
    return suggest.stream()
            .filter(s -> s.startsWith(input))
            .limit(count)
            .toList();
}
```

- Scales with size of dataset
- Speed changes based on sorting
- Custom sorting possible
- Live updates!

Better implementation ideas?

Radix tree

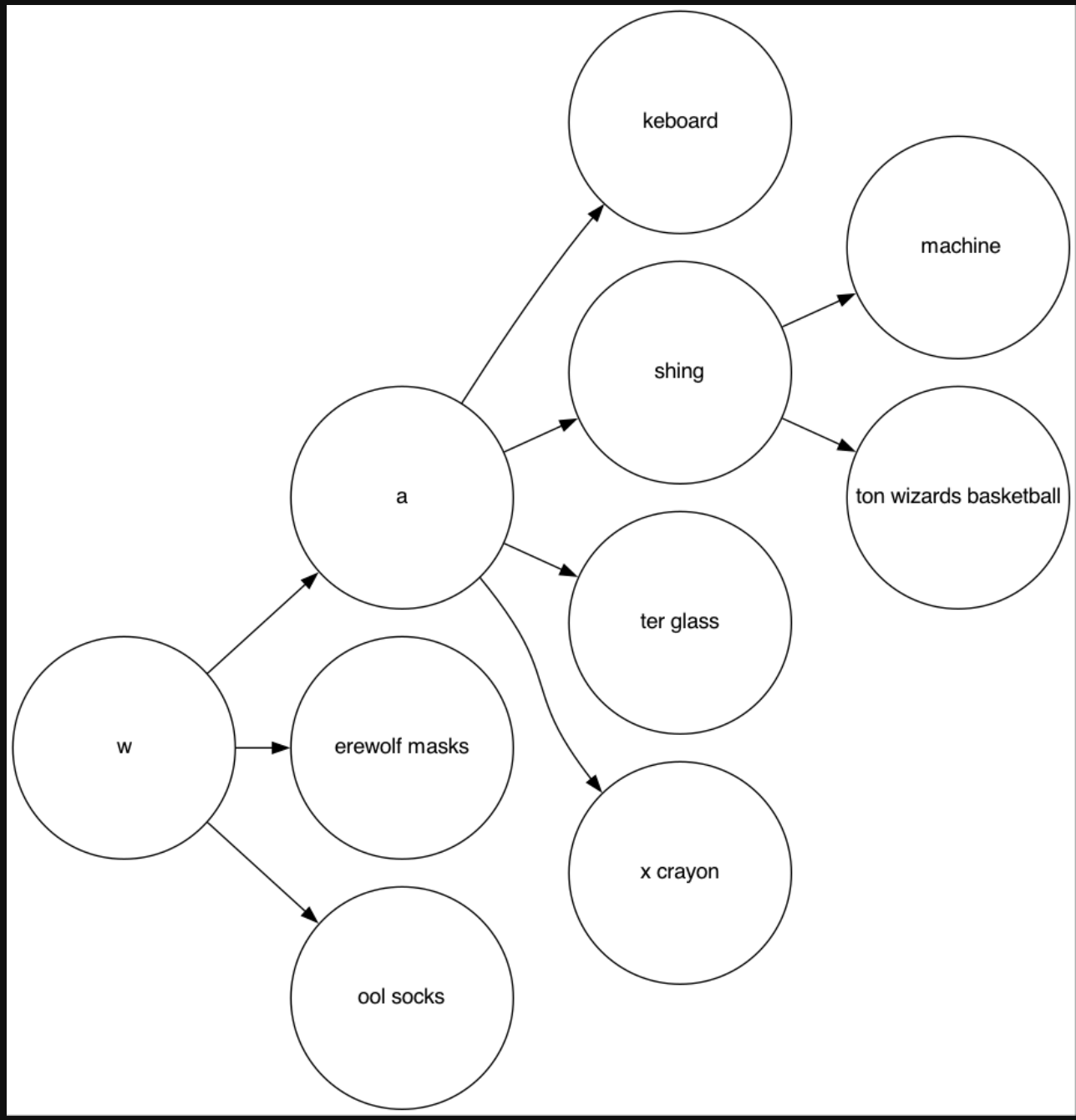

w
a
k
e
b
o
a
r
d
s
h
i
n
g
m
a
c
h
i
n
e
t
o
n
w
i
z
a
r
d
s
t
e
r
g
l
a
s
s
x
c
r
a
y
o
n
e
r
e
w
o
l
f
m
a
s
k
o
o
l
s
o
c
k
s

Radix tree

- Find matches by walking the tree
- Fast
- Easy to figure out if no matches left
- Updateable
- No weighting?
- Do you spot the obvious optimization?

Adaptive Radix tree

- Much smaller
- Much faster
- Complex updates
- Build time vs run time



Java implementation

```java
@Test
public void testRadixTree() {
  ConcurrentRadixTree<Integer> radixTree = new ConcurrentRadixTree<>(new DefaultCharArrayNodeFactory());
  radixTree.put("toast", 123);
  radixTree.put("test", 10);

  Iterable<KeyValuePair<Integer>> iterable = radixTree.getKeyValuePairsForClosestKeys("t");
  // wrong order, requires resorting of all results...
  assertThat(iterable).map(KeyValuePair::getKey).containsExactly("test", "toast");

  // this is inefficient
  Comparator<KeyValuePair<Integer>> cmp = (o1, o2) -> o2.getValue().compareTo(o1.getValue());
  SortedSet<KeyValuePair<Integer>> response = new TreeSet<>(cmp);
  iterable.forEach(response::add);
}
```

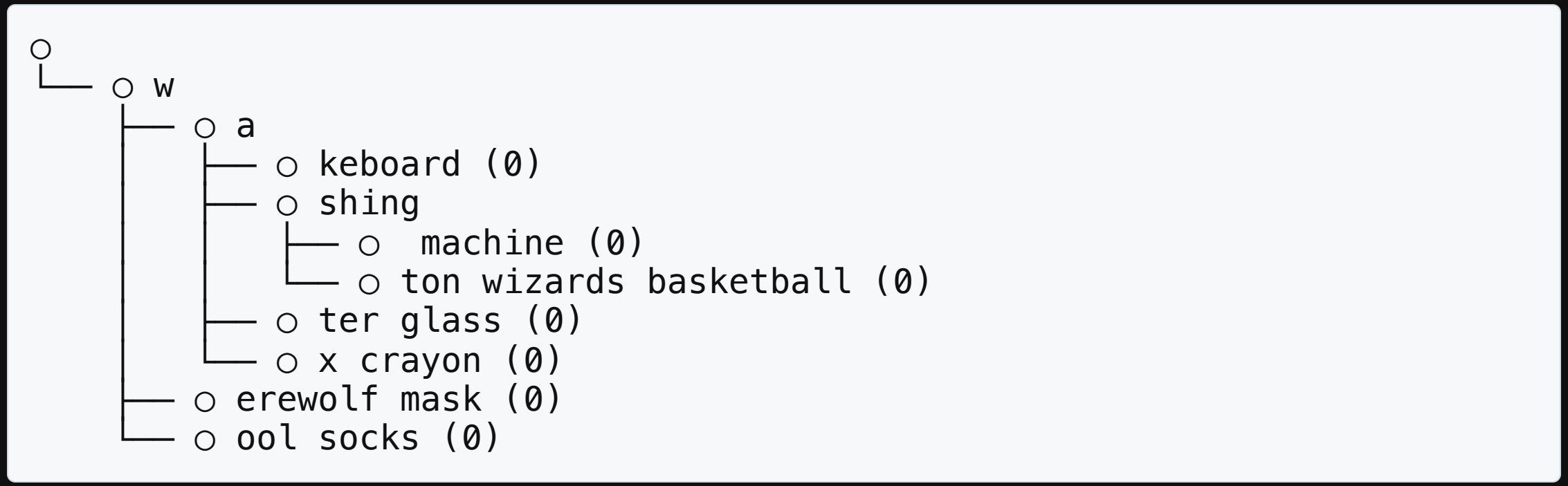
Concurrent radix tree

- Check out the concurrent-trees library, unfortunately unmaintained
- Contains `RadixTree`, `ReversedRadixTree`, `InvertedRadixTree`, `SuffixTree` implementations
- Lock free reads, concurrent writes, atomic updates

PrettyPrinter

```
@Test
public void testSampleDataset() {
    ConcurrentRadixTree<Integer> radixTree = new ConcurrentRadixTree<>(new DefaultCharArrayNodeFactory());
    radixTree.put("wakeboard", 0);
    radixTree.put("washing machine", 0);
    radixTree.put("washington wizards basketball", 0);
    radixTree.put("water glass", 0);
    radixTree.put("wax crayon", 0);
    radixTree.put("werewolf mask", 0);
    radixTree.put("wool socks", 0);

    System.out.println(PrettyPrinter.prettyPrint(radixTree));
}
```

PrettyPrinter



Relevancy

Relevancy

- `RadixTree` was not built for this!
- No early termination

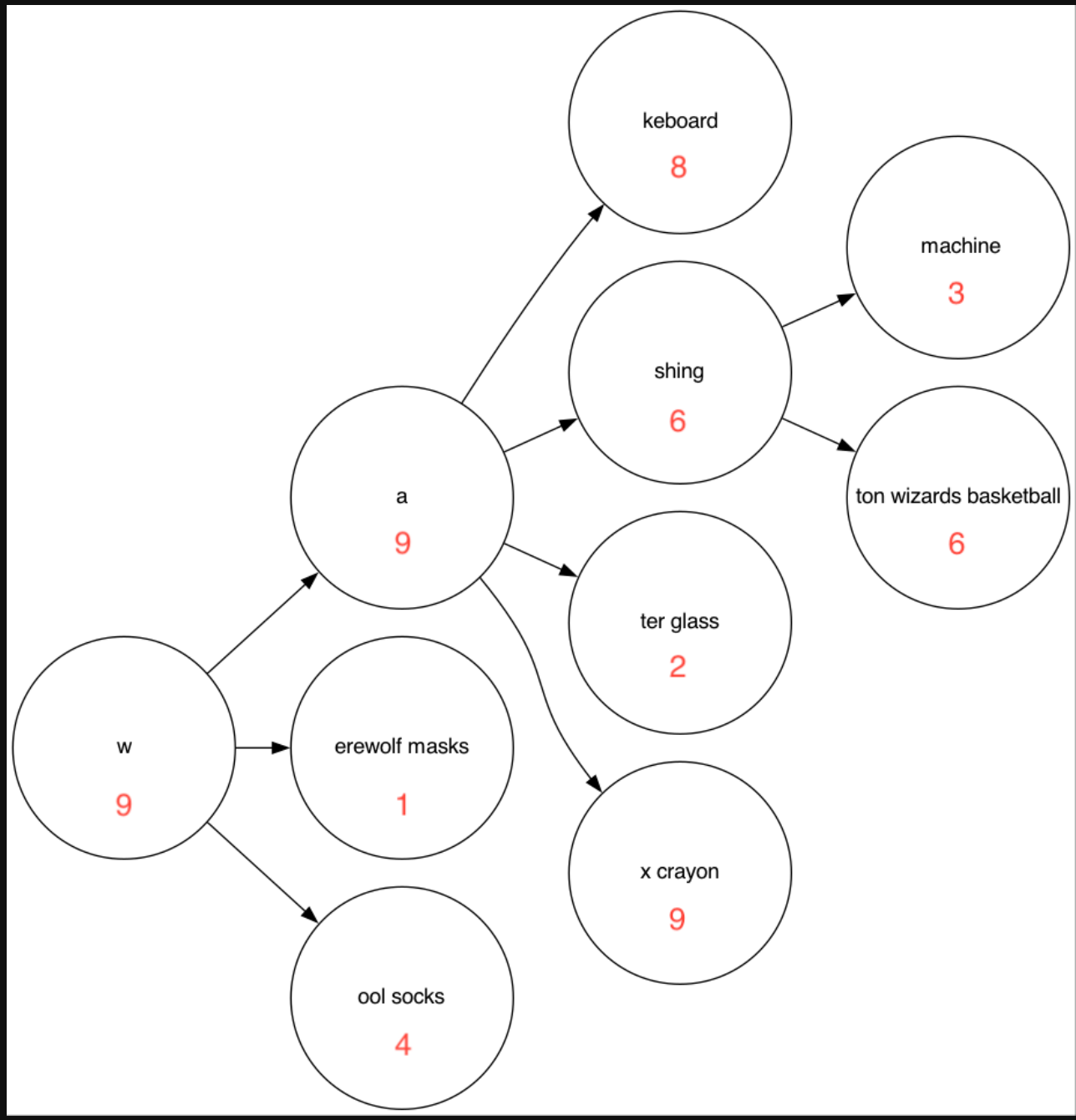
Pruning Radix Trie!

- RadixTree, but with scoring!
- ... and early termination
- Idea comes from Wolf Garbe, see this blog post
- Java implementation: JPruningRadixTrie

```java
@Test
public void testPruningRadixTree() {
  PruningRadixTrie prt = new PruningRadixTrie();
  AtomicInteger counter = new AtomicInteger(1);
  for (String input : List.of("wakeboard", "washing machine",
          "washington wizards basketball", "water glass",
          "wax crayon", "werewolf mask", "wool socks")) {
      prt.addTerm(input, counter.getAndIncrement());
  }

  List<TermAndFrequency> results = prt.getTopkTermsForPrefix("wa", 2);

  assertThat(results).map(t -> t.term() + "/" + t.termFrequencyCount())
    .containsExactly(
      "wax crayon/6",
      "water glass/5"
    );
}
```

Pruning Radix Tree

- Each node contains max score of all children
- Example: Input `wa`, size 2



Lucene?

Lucene!

- De-facto standard for open source full text search
- Clones in many different programming languages
- Just turned 21!

```
String data = """
              wakeboard\t1
              washing machine\t2
              washington wizards basketball\t3
              water glass\t4
              wax crayon\t5
              werewolf mask\t6
              wool socks\t7
              """;
```

WeightedFST

```
@Test
public void testLuceneWFST() throws Exception {
    Directory directory = new NIOFSDirectory(Paths.get("/tmp/"));
    FileDictionary fileDictionary = new FileDictionary(new StringReader(data));
    WFSTCompletionLookup lookup = new WFSTCompletionLookup(directory, "wfst", true);
    lookup.build(fileDictionary);

    List<Lookup.LookupResult> results = lookup.lookup("wa", null, false, 10);
    assertThat(results).hasSize(5);
    assertThat(results).map(Lookup.LookupResult::toString)
            .containsExactly("wax crayon/5",
                    "water glass/4",
                    "washington wizards basketball/3",
                    "washing machine/2",
                    "wakeboard/1");

}
```

FSTs

- Extremely fast
- Build-once
- No updates
- Can be serialized to disk, loaded with small deserialization overhead
- 6 million terms require 42 MB of disk space
- FTS power: `FuzzySuggester`, phonetic suggestions, infix suggestions, synonyms

FuzzySuggester

```
@Test
public void testFuzzySuggester() throws Exception {
    Directory directory = new NIOFSDirectory(Paths.get("/tmp/"));
    FuzzySuggester analyzingSuggester = new FuzzySuggester(directory, "suggest",
            new StandardAnalyzer());

    FileDictionary fileDictionary = new FileDictionary(new StringReader(data));
    analyzingSuggester.build(fileDictionary);

    List<Lookup.LookupResult> results =
        analyzingSuggester.lookup("wasch", false, 5);
    assertThat(results).hasSize(2);
    assertThat(results).map(Lookup.LookupResult::toString)
            .containsExactly("washing machine/1",
                             "washington wizards basketball/1");
}
```

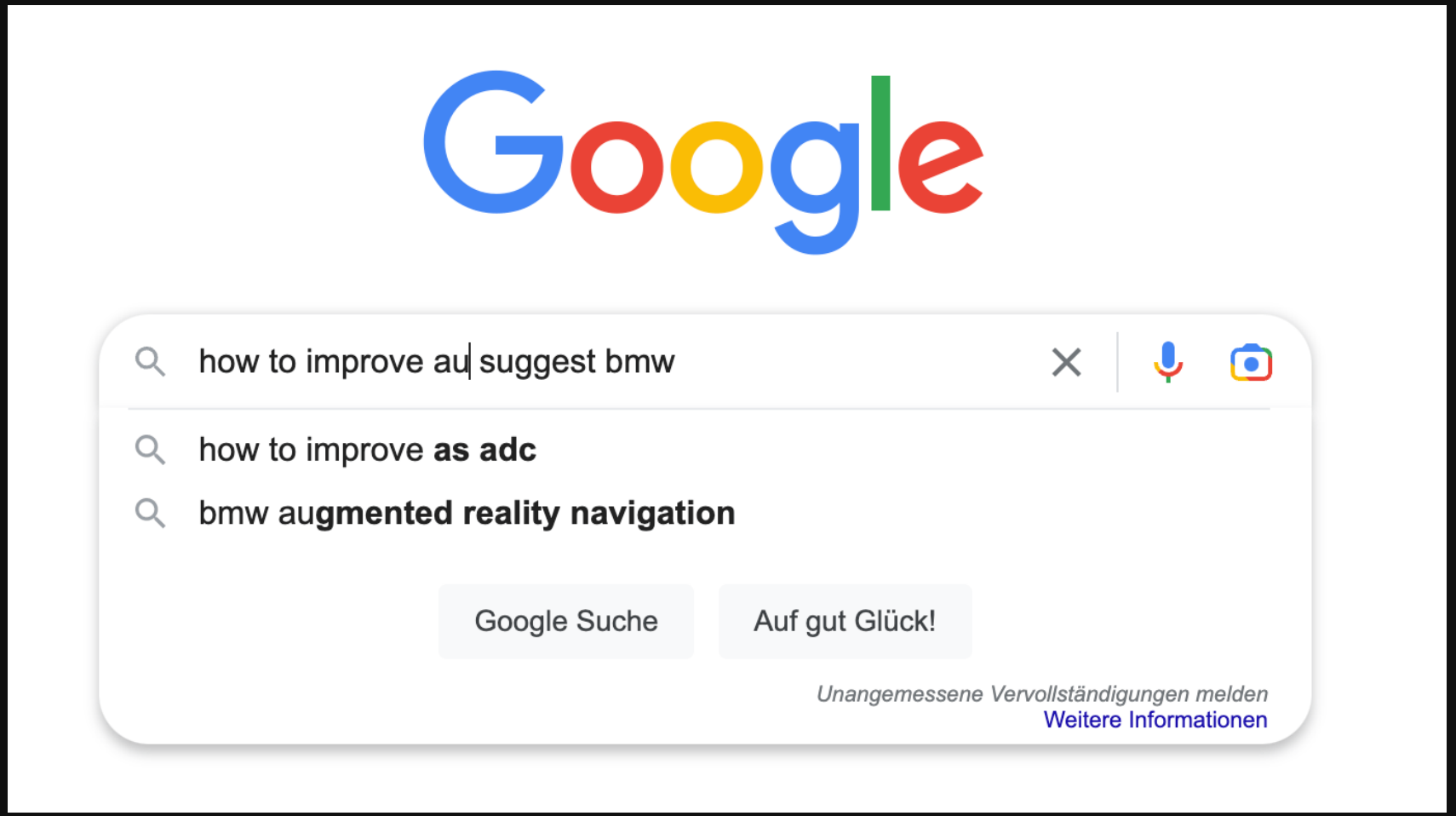
```java
@Test
public void testPhoneticSuggest() throws Exception {
    Map<String, String> args = new HashMap<>();
    args.put("encoder", "ColognePhonetic");
    CustomAnalyzer analyzer = CustomAnalyzer.builder()
            .addTokenFilter(PhoneticFilterFactory.class, args)
            .withTokenizer("standard")
            .build();

    Directory directory = new NIOFSDirectory(Paths.get("/tmp/"));
    AnalyzingSuggester suggester =
        new AnalyzingSuggester(directory, "lucene-tmp", analyzer);

    FileDictionary dictionary = new FileDictionary(new StringReader(input));
    suggester.build(dictionary);

    List<Lookup.LookupResult> results = suggester.lookup("vaschink", false, 5);
    assertThat(results).map(Lookup.LookupResult::toString)
            .containsExactly("washington wizards basketball/3",
                             "washing machine/2");

}
```

InfixSuggester

```
@Test
public void testInfixSuggester() throws Exception {
    Directory directory = new NIOFSDirectory(Paths.get("/tmp/"));
    AnalyzingInfixSuggester suggester =
        new AnalyzingInfixSuggester(directory, new StandardAnalyzer());

    FileDictionary dictionary = new FileDictionary(new StringReader(input));
    suggester.build(dictionary);

    List<Lookup.LookupResult> results = suggester.lookup("wiz", false, 5);
    assertThat(results).map(Lookup.LookupResult::toString)
            .containsExactly("washington wizards basketball/3");

    results = suggester.lookup("ma", false, 5);
    assertThat(results).map(Lookup.LookupResult::toString)
            .containsExactly("werewolf mask/6", "washing machine/2");
}
```

Basics

Dude, where's my cursor?



Typo tolerance

- Levenshtein
- Phonetic
- Keyboard
- Frequency dictionary (Symspell)

Ranking

- What is weight?
- Popularity?
- Recency?
- Score current category higher
- Is this the same for every user?
- Include previous queries or purchases
- Multidimensional

Learning-to-rank

- Let your data scientists build a model
- Reuse that model

Implementations

- Elasticsearch
  - `search-as-you-type` field type
  - with `rank_feature` fields
  - Returns full documents
- Vespa:
  - Regular query, then rescoring against ML model
  - Support for XGBoost, ONNX, LightGBM, Tensorflow

Whiteboard implementation

- Offline creation
- Incremental updates optional
- Synchronization with search engine
- No deserialization overhead
- Scalable readers
- Rescoring
- Are suggestions really worth all this work?

Summary

- Auto-suggest is powerful
- Fix your search first before playing with auto-suggest
- Never point suggestions into no results
- ML/LTR: Zero-shot models (soon: model marketplaces?)
- Search moves to the edge!
- Change of search changes requirements:
  - `voice` search
  - `chat gpt` like search
- LLMs/Generative search up and coming (i.e. Vectara)

Thanks for listening

Q & A

Alexander Reelsen

alr@spinscale.de | @spinscale

Resources

https://spinscale.de/posts/2023-01-18-mirror-mirror-what-am-i-typing-next.html

Discussion

- What technologies would you use?
- What algorithms would you use?
- Where did I go wrong?

Thanks for listening

Q & A

Alexander Reelsen

alr@spinscale.de | @spinscale