Efficient and Effective Query Auto-Completion

Giulio Ermanno Pibiri

ISTITUTO DI SCIENZA E TECNOLOGIE DELL'INFORMAZIONE "A. FAEDO"

Simon Gog

ebay

Rossano Venturini

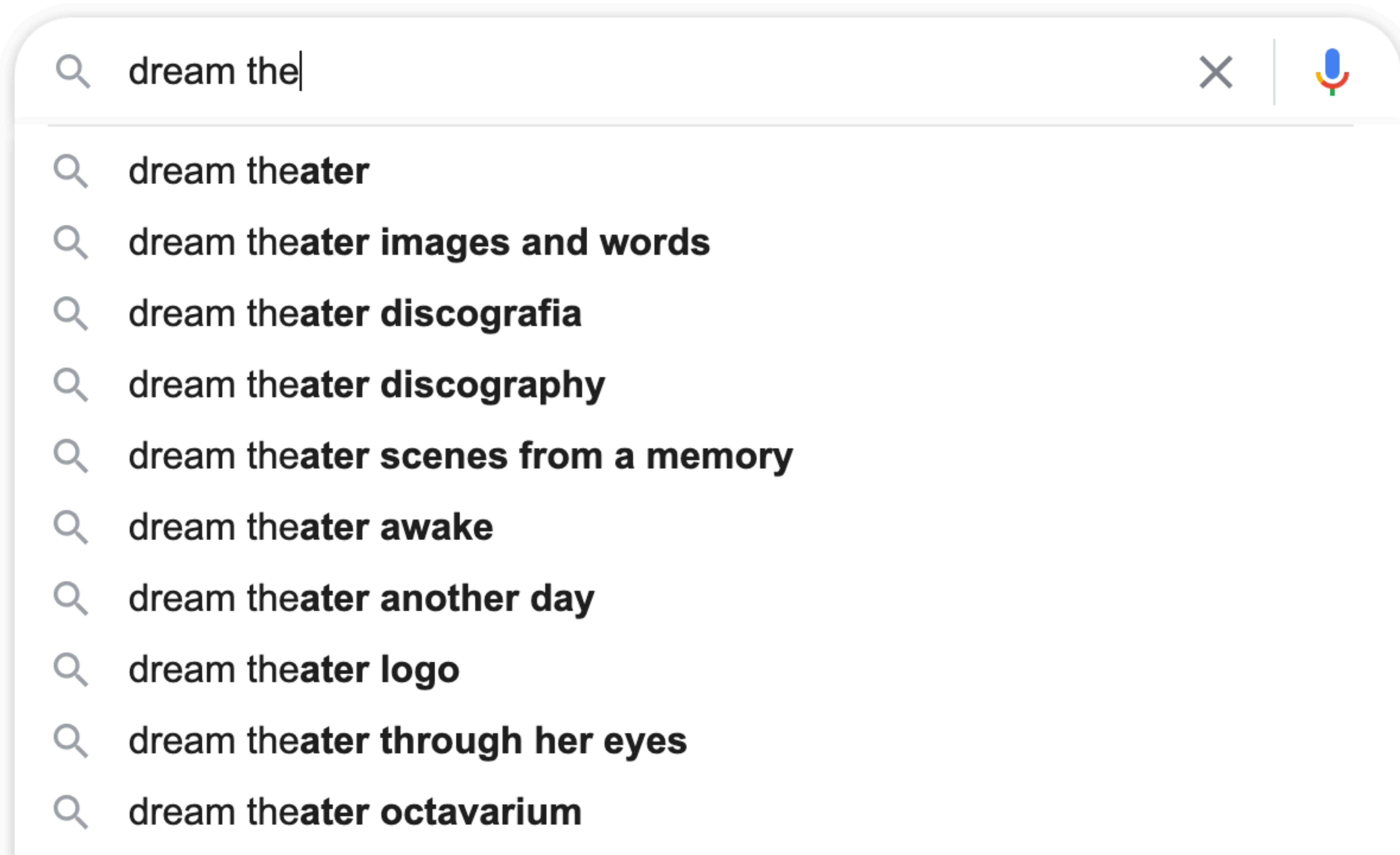
IN SUPREMÆ DIGNITATIS 1343

ACM Conference on Research and Development in Information Retrieval (SIGIR), 2020

27/07/2020

Query Auto-Completion

Given a collection S of **scored** strings and a *partially completed* user query Q, find the **top-*k*** strings that “match” Q in S.



Setting

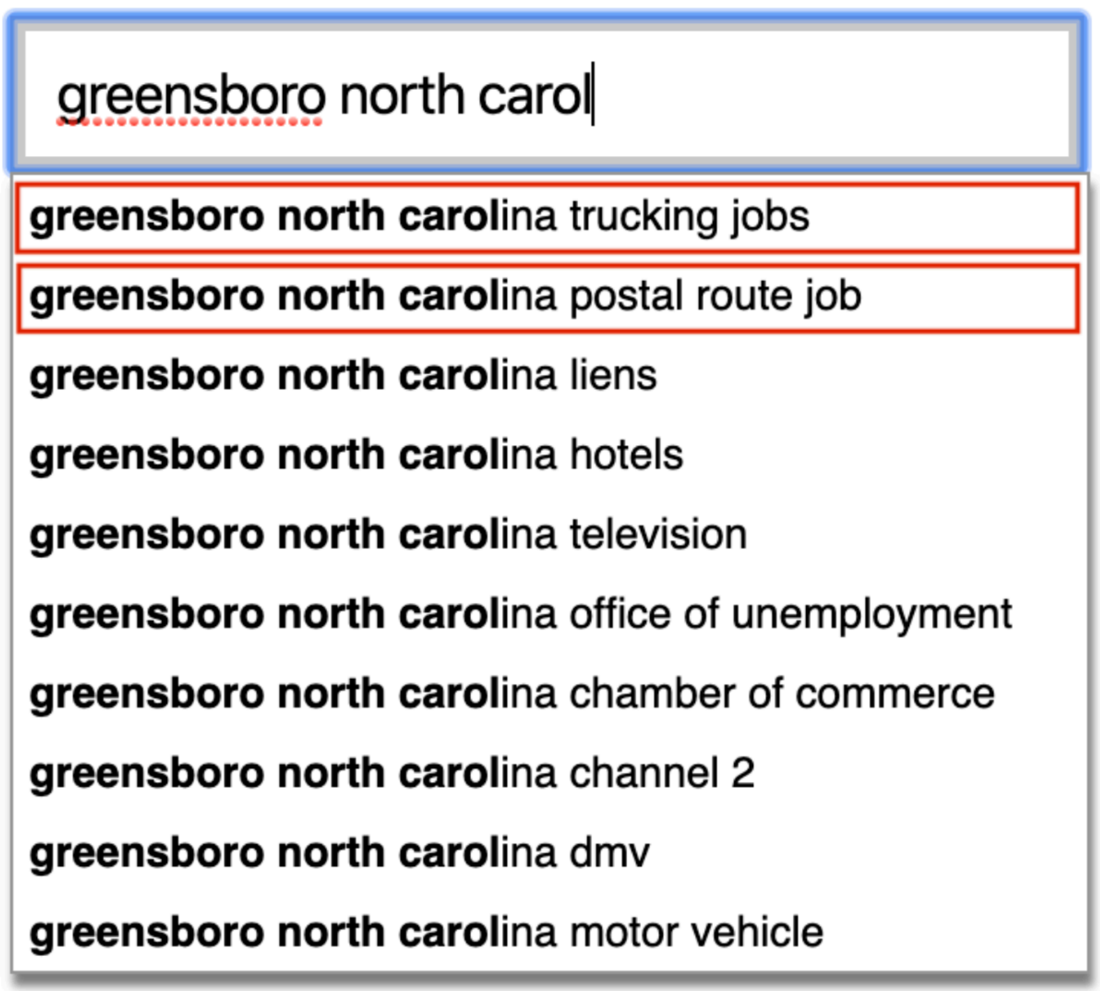
We focus on **matching algorithms**, not ranking mechanisms:
we return the "most popular" results from a query log.

Many matching algorithms are possible, such as:
exact, *prefix*, *pattern (substring), edit-distance…*

Setting

We focus on **matching algorithms**, not ranking mechanisms:
we return the "most popular" results from a query log.

Many matching algorithms are possible, such as:
exact, *prefix*, *pattern (substring)*, *edit-distance…*

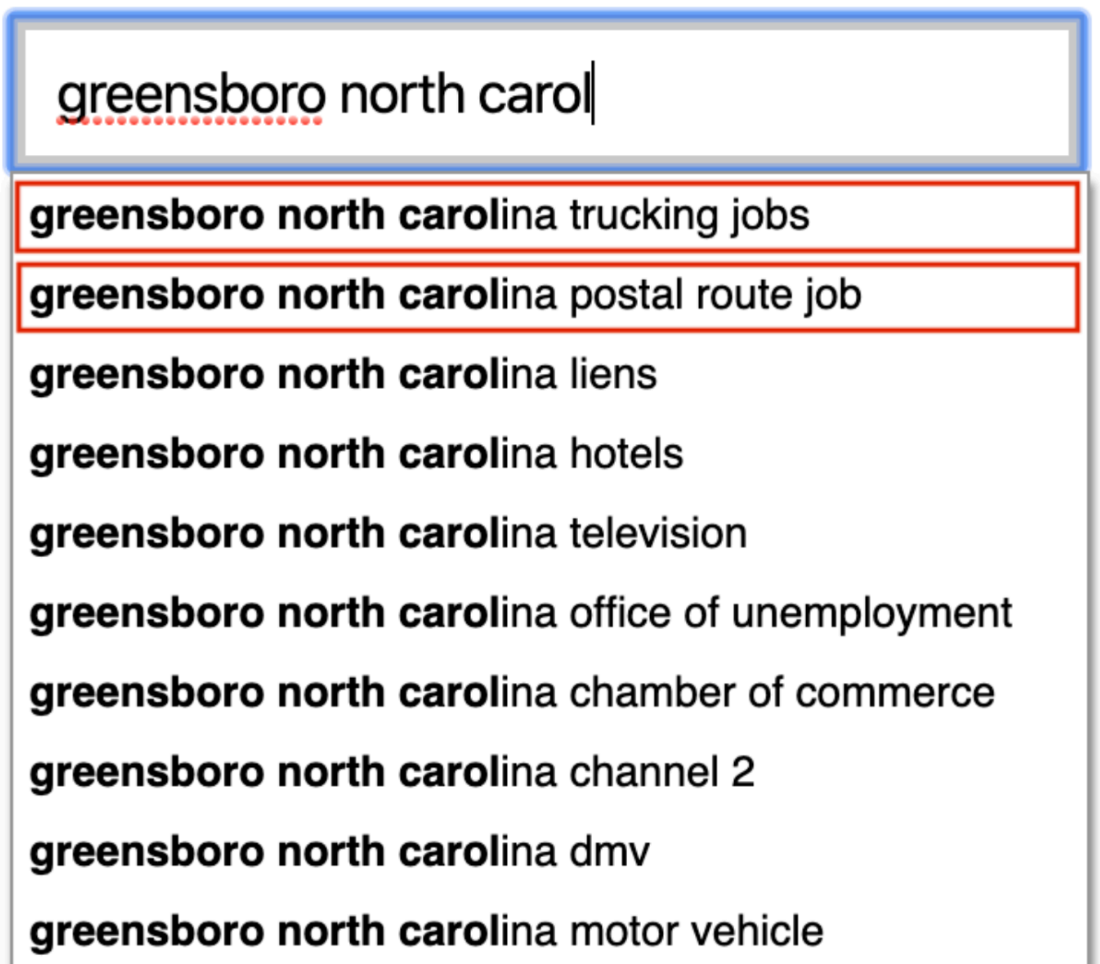


prefix

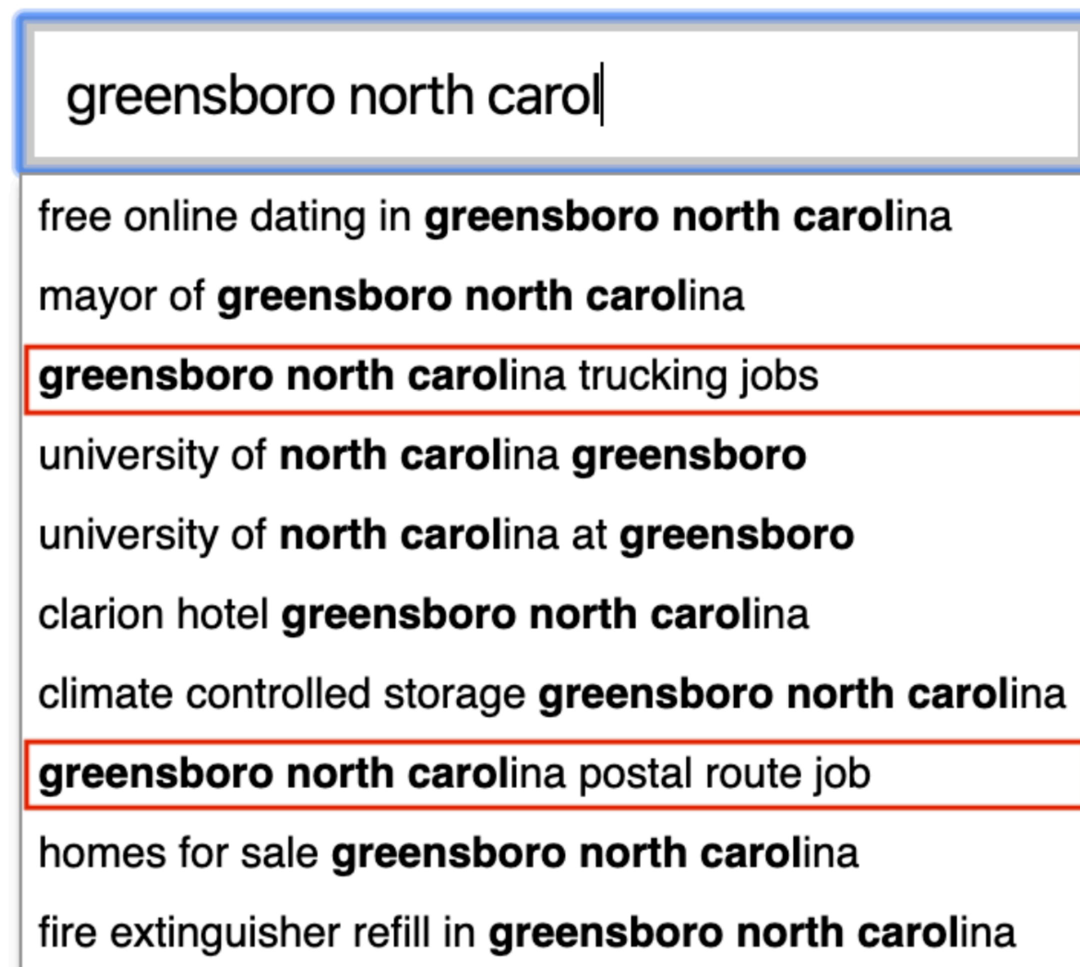
Setting

We focus on **matching algorithms**, not ranking mechanisms:
we return the "most popular" results from a query log.

Many matching algorithms are possible, such as:
exact, *prefix*, *pattern (substring), edit-distance…*



prefix



conjunctive

Conjunctive-Search

Return strings containing ***all*** the tokens in the prefix and ***any*** token prefixed by the suffix.

Build an inverted index where docids are assigned in decreasing score order: smaller docids are better.

Conjunctive-Search

Return strings containing ***all*** the tokens in the prefix
and ***any*** token prefixed by the suffix.

Build an inverted index where docids are assigned in decreasing score order: smaller docids are better.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms	inverted lists
1	a3	$\langle 6 \rangle$
2	audi	$\langle 3, 6, 9 \rangle$
3	bmw	$\langle 1, 2, 4, 5, 7, 8 \rangle$
4	i3	$\langle 1, 2, 4 \rangle$
5	i8	$\langle 7 \rangle$
6	q8	$\langle 3 \rangle$
7	sedan	$\langle 1, 3 \rangle$
8	sport	$\langle 4, 6, 7 \rangle$
9	sportback	$\langle 2 \rangle$
10	x1	$\langle 5 \rangle$

bmw s

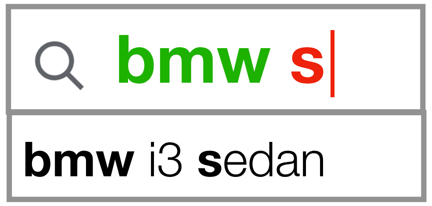
Conjunctive-Search

Return strings containing ***all*** the tokens in the prefix
and ***any*** token prefixed by the suffix.

Build an inverted index where
docids are assigned in decreasing score order:
smaller docids are better.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms	inverted lists
1	a3	⟨6⟩
2	audi	⟨3, 6, 9⟩
3	bmw	⟨1, 2, 4, 5, 7, 8⟩
4	i3	⟨1, 2, 4⟩
5	i8	⟨7⟩
6	q8	⟨3⟩
7	sedan	⟨1, 3⟩
8	sport	⟨4, 6, 7⟩
9	sportback	⟨2⟩
10	x1	⟨5⟩



Conjunctive-Search

Return strings containing ***all*** the tokens in the prefix
and ***any*** token prefixed by the suffix.

Build an inverted index where
docids are assigned in decreasing score order:
smaller docids are better.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms	inverted lists
1	a3	$\langle 6 \rangle$
2	audi	$\langle 3, 6, 9 \rangle$
3	bmw	$\langle 1, 2, 4, 5, 7, 8 \rangle$
4	i3	$\langle 1, 2, 4 \rangle$
5	i8	$\langle 7 \rangle$
6	q8	$\langle 3 \rangle$
7	sedan	$\langle 1, 3 \rangle$
8	sport	$\langle 4, 6, 7 \rangle$
9	sportback	$\langle 2 \rangle$
10	x1	$\langle 5 \rangle$

bmw s

bmw i3 **s**edan

bmw i3 **s**portback

Conjunctive-Search

Return strings containing ***all*** the tokens in the prefix and ***any*** token prefixed by the suffix.

Build an inverted index where docids are assigned in decreasing score order: smaller docids are better.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms	inverted lists
1	a3	$\langle 6 \rangle$
2	audi	$\langle 3, 6, 9 \rangle$
3	bmw	$\langle 1, 2, 4, 5, 7, 8 \rangle$
4	i3	$\langle 1, 2, 4 \rangle$
5	i8	$\langle 7 \rangle$
6	q8	$\langle 3 \rangle$
7	sedan	$\langle 1, 3 \rangle$
8	sport	$\langle 4, 6, 7 \rangle$
9	sportback	$\langle 2 \rangle$
10	x1	$\langle 5 \rangle$

bmw s
bmw i3 **s**edan
bmw i3 **s**portback
bmw i3 **s**port

Conjunctive-Search

bmw [7,9]

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    intersection = index.IntersectionIterator(prefix)
    results = [ ], heap = [ ]
    for i = ℓ; i ≤ r; i = i + 1 :
        heap.Append(index.Iterator(i))
    heap.MakeHeap()
    while intersection.HasNext() and !heap.Empty() :
        x = intersection.Next()
        while !heap.Empty() :
            top = heap.Top()
            if top.docid > x : break
            if top.docid < x :
                if top.NextGeq(x) < ∞ : heap.Heapify()
                else : heap.Pop()
            else :
                results.Append(x)
                if |results| == k : return results
                break
    return results
```

Conjunctive-Search

bmw [7,9]

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    intersection = index.IntersectionIterator(prefix)
    results = [ ], heap = [ ]
    for i = ℓ; i ≤ r; i = i + 1 :
        heap.Append(index.Iterator(i))
    heap.MakeHeap()
    while intersection.HasNext() and !heap.Empty() :
        x = intersection.Next()
        while !heap.Empty() :
            top = heap.Top()
            if top.docid > x : break
            if top.docid < x :
                if top.NextGeq(x) < ∞ : heap.Heapify()
                else : heap.Pop()
            else :
                results.Append(x)
                if |results| == k : return results
                break
    return results
```

Heap-based approach:
(1) Much better than explicitly computing the union.
(2) Terms involved in union may be too many!

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7, 6, 4, 1
2	audi q 8 2017	7, 10, 5, 3
4	bmw x 1	8, 11, 0
0	bmw i 3 2015	8, 9, 4, 1
3	bmw i 3 2016	8, 9, 4, 2
1	bmw i 3 2017	8, 9, 4, 3
6	bmw i 8 2015	8, 9, 5, 1

Conjunctive-Search

ConjunctiveSearch(*prefix*, $[\ell, r]$, k) :
  results = [ ]
  intersection = *index*.IntersectionIterator(*prefix*)
  while *intersection*.HasNext() :
    x = *intersection*.Next()
    completion = Extract(x)
    if *completion* intersects $[\ell, r]$:
      results.Append(x)
      if $|results| == k$: **break**
  return *results*

Forward search

bmw 2

bmw i 3 2015

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 2015

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 2015

bmw i 3 2017

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 2015

bmw i 3 2017

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 **2**015

bmw i 3 **2**017

bmw i 3 **2**016

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7, 6, 4, 1
2	audi q 8 2017	7, 10, 5, 3
4	bmw x 1	8, 11, 0
0	bmw i 3 2015	8, 9, 4, 1
3	bmw i 3 2016	8, 9, 4, 2
1	bmw i 3 2017	8, 9, 4, 3
6	bmw i 8 2015	8, 9, 5, 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 **2**015

bmw i 3 **2**017

bmw i 3 **2**016

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 **2**015

bmw i 3 **2**017

bmw i 3 **2**016

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

bmw 2

bmw i 3 **2**015

bmw i 3 **2**017

bmw i 3 **2**016

bmw i 8 **2**015

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Conjunctive-Search

```
ConjunctiveSearch(prefix, [ℓ, r], k) :
    results = [ ]
    intersection = index.IntersectionIterator(prefix)
    while intersection.HasNext() :
        x = intersection.Next()
        completion = Extract(x)
        if completion intersects [ℓ, r] :
            results.Append(x)
            if |results| == k : break
    return results
```

Forward search

Forward-Search approach:
(1) No heap management.
(2) Need "direct" access to completions: Fwd or FC.

bmw 2

- **bmw** i 3 **2**015
- **bmw** i 3 **2**017
- **bmw** i 3 **2**016
- **bmw** i 8 **2**015

terms	inverted lists	termids
1	4	0
2015	0, 5, 6	1
2016	3	2
2017	1, 2	3
3	0, 1, 3, 5	4
8	2, 6	5
a	5	6
audi	2, 5	7
bmw	0, 1, 3, 4, 6	8
i	0, 1, 3, 6	9
q	2	10
x	4	11

docids	completions	sets
5	audi a 3 2015	7 , 6 , 4 , 1
2	audi q 8 2017	7 , 10 , 5 , 3
4	bmw x 1	8 , 11 , 0
0	bmw i 3 2015	8 , 9 , 4 , 1
3	bmw i 3 2016	8 , 9 , 4 , 2
1	bmw i 3 2017	8 , 9 , 4 , 3
6	bmw i 8 2015	8 , 9 , 5 , 1

Experiments

Machine equipped with Intel i9-9900K cores (@3.60 GHz),
64 GB of RAM, and running Linux 5 (64 bits).

C++ code available at

https://github.com/jermp/autocomplete

Statistic	AOL	MSN	EBAY
Queries	10,142,395	7,083,363	7,295,104
Uncompressed size in MiB	299	208	189
Unique query terms	3,825,848	2,590,937	323,180
Avg. num. of chars per term	14.58	14.18	7.32
Avg. num. of queries per term	7.87	8.15	73.02
Avg. num. of terms per query	2.99	2.99	3.24

Datasets

Experiments — Efficiency

(a) AOL

	%	Query terms 1	2	3	4	5	6	7+
Fwd	0	4	5	22	30	24	24	16
	25	2	97	70	41	30	25	16
	50	0	149	77	48	30	25	16
	75	0	150	76	48	30	25	16
FC	0	5	15	27	30	24	24	16
	25	3	251	110	45	31	25	16
	50	1	370	121	56	31	25	16
	75	0	375	121	57	32	25	16
Heap	0	55,537	29,189	30,498	22,431	17,713	16,474	13,312
	25	474	623	957	485	376	378	299
	50	1	251	178	251	229	123	178
	75	0	226	162	240	219	116	173
Hyb	0	286	2,718	1,673	965	634	503	413
	25	11	184	223	276	258	221	192
	50	10	126	185	270	250	217	186
	75	6	116	178	268	248	216	184

(b) MSN

	%	Query terms 1	2	3	4	5	6	7+
Fwd	0	4	5	14	15	11	10	7
	25	1	39	34	18	13	10	7
	50	0	56	38	19	13	10	8
	75	0	57	37	19	12	10	7
FC	0	5	15	17	15	11	10	7
	25	2	101	51	19	13	10	8
	50	1	137	58	21	13	10	7
	75	0	137	57	21	13	10	7
Heap	0	7,626	12,459	11,964	8,921	6,164	5,749	5,686
	25	353	252	256	282	170	192	125
	50	10	73	70	109	84	66	54
	75	1	61	62	83	80	63	51
Hyb	0	53	1,626	915	477	307	270	237
	25	10	90	109	127	111	111	90
	50	7	53	97	122	107	108	87
	75	4	46	95	121	106	106	85

Top-**10** conjunctive-search query timings in µsec per query, by varying query length and percentage of the last query token.

Experiments — Effectiveness

(a) AOL

	Query terms						
%	1	2	3	4	5	6	7+
0	17	107	207	327	295	270	270
25	19	178	246	373	298	155	356
50	23	227	302	440	364	213	524
75	41	282	362	504	424	257	882

(b) MSN

Query terms						
1	2	3	4	5	6	7+
27	139	252	283	325	206	248
23	231	310	297	333	190	200
27	243	313	320	359	251	208
44	284	364	357	407	319	236

Percentage of better scored results returned by conjunctive-search with *respect to those returned by prefix-search* for top-**10** queries.

Experiments — Space

	AOL		MSN		EBAY	
	MiB	bpc	MiB	bpc	MiB	bpc
Fwd	312	32.28	218	32.32	168	24.14
FC	266	27.51	185	27.42	140	20.13
Heap	254	26.25	177	26.25	139	19.99
Hyb	275	28.48	191	28.26	157	22.50

Space usage in total MiB and bytes per completion (bpc).

Experiments — Space

Statistic	AOL	MSN	EBAY
Queries	10,142,395	7,083,363	7,295,104
Uncompressed size in MiB	299	208	189
Unique query terms	3,825,848	2,590,937	323,180
Avg. nun			
Avg. nun			
Avg. nun			

	AOL		MSN		EBAY	
	MiB	bpc	MiB	bpc	MiB	bpc
Fwd	312	32.28	218	32.32	168	24.14
FC	266	27.51	185	27.42	140	20.13
Heap	254	26.25	177	26.25	139	19.99
Hyb	275	28.48	191	28.26	157	22.50

Space usage in total MiB and bytes per completion (bpc).

Experiments — Space

Statistic	AOL	MSN	EBAY
Queries	10,142,395	7,083,363	7,295,104
Uncompressed size in MiB	299	208	189
Unique query terms	3,825,848	2,590,937	323,180
Avg. nun			
Avg. nun			
Avg. nun			

	AOL		MSN		EBAY	
	MiB	bpc	MiB	bpc	MiB	bpc
Fwd	312	32.28	218	32.32	168	24.14
FC	266	27.51	185	27.42	140	20.13
Heap	254	26.25	177	26.25	139	19.99
Hyb	275	28.48	191	28.26	157	22.50

Space usage in total MiB and bytes per completion (bpc).

Take-away Messages

- Conjunctive-search overcomes the limited effectiveness of prefix-search by returning more and better scored results.

- While prefix-search is very fast (less then 3 µsec per query on average), conjunctive-search is more expensive and costs between 4 and 500 µsec per query depending on the size of the query.

- Our optimized implementation of conjunctive-search substantially outperforms the use of a classical as well as blocked inverted index with small extra, or even less, space.

Thanks for your attention!

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)
    strings = ExtractStrings(topk_ids)
    return strings
```

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)
    strings = ExtractStrings(topk_ids)
    return strings
```

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

bmw i3 s

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)   1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)
    strings = ExtractStrings(topk_ids)
    return strings
```

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

1

bmw i3 s

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)                 1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])        2
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)
    strings = ExtractStrings(topk_ids)
    return strings
```

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

2

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

1

bmw i3 s

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)          1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])  2
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)                          3
    strings = ExtractStrings(topk_ids)
    return strings
```

Docids are assigned in decreasing score order: top-*k* algorithm reduces to RMQ.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

bmw i3 s

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)                1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])       2
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)                               3
    strings = ExtractStrings(topk_ids)
    return strings
```

Docids are assigned in decreasing score order: top-*k* algorithm reduces to RMQ.

2

3

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

1

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

bmw i3 s

bmw i3 sedan

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)            1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])   2
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)                           3
    strings = ExtractStrings(topk_ids)
    return strings
```

Docids are assigned in decreasing score order: top-*k* algorithm reduces to RMQ.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

2

3

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

1

bmw i3 s

bmw i3 sedan

bmw i3 sportback

Prefix-Search

```
Complete(query, k) :
    prefix, suffix = Parse(dictionary, query)
    if prefix was not found : return [ ]
    [ℓ, r] = dictionary.LocatePrefix(suffix)              1
    if [ℓ, r] is invalid : return [ ]
    [p, q] = completions.LocatePrefix(prefix, [ℓ, r])     2
    if [p, q] is invalid : return [ ]
    topk_ids = RMQ([p, q], k)                             3
    strings = ExtractStrings(topk_ids)
    return strings
```

Docids are assigned in decreasing score order: top-*k* algorithm reduces to RMQ.

docids	completions
9	audi
6	audi a3 sport
3	audi q8 sedan
8	bmw
5	bmw x1
1	bmw i3 sedan
4	bmw i3 sport
2	bmw i3 sportback
7	bmw i8 sport

termids	terms
1	a3
2	audi
3	bmw
4	i3
5	i8
6	q8
7	sedan
8	sport
9	sportback
10	x1

bmw i3 s

bmw i3 sedan

bmw i3 sportback

bmw i3 sport