Chapter IR:IV

IV. Indexes

- Inverted Indexes
- Query Processing
- Index Construction
- Index Compression

Query Processing

Query Types

The primary goal of indexing is to speed up query processing. Dependent on the query type, different index configurations in terms of postlist ordering are best-suited.

Ordering by document identifier:

- Conjunctive query, e.g.: Boolean AND query, phrasal query, proximity query
- Disjunctive query, i.e.: multi-term query where complete rankings are required

Ordering by term weight:

- Disjunctive query, e.g., multi term query where approximate ranking sufficient
- Single term query

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮											
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮											

Given a Boolean query $q = t_i \wedge t_j$, what needs to be done to answer it?

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮											
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	...
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	...
⋮											

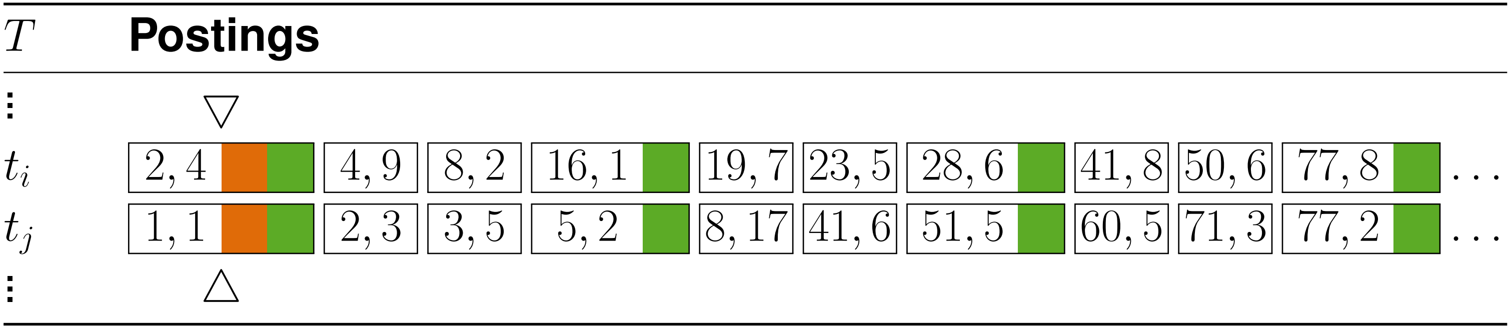
Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:



T	**Postings**										
⋮	▽										
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮	△										

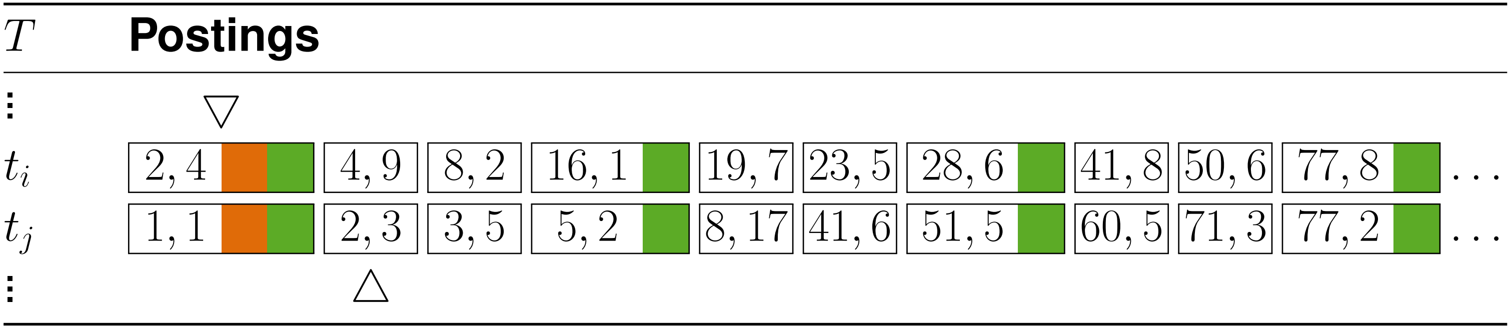
Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:



T	**Postings**										
⋮	▽										
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	…
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	…
⋮		△									

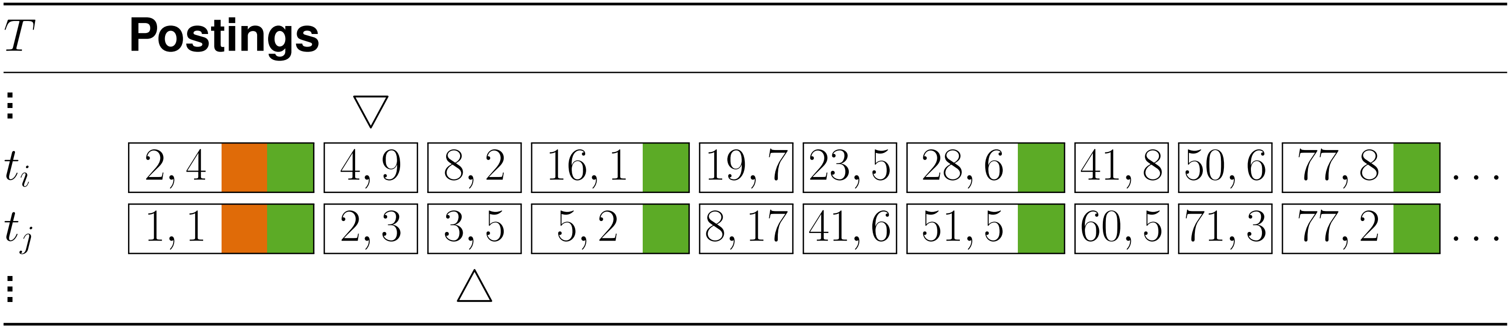
Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),   next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:



T	**Postings**										
⋮		▽									
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	…
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	…
⋮			△								

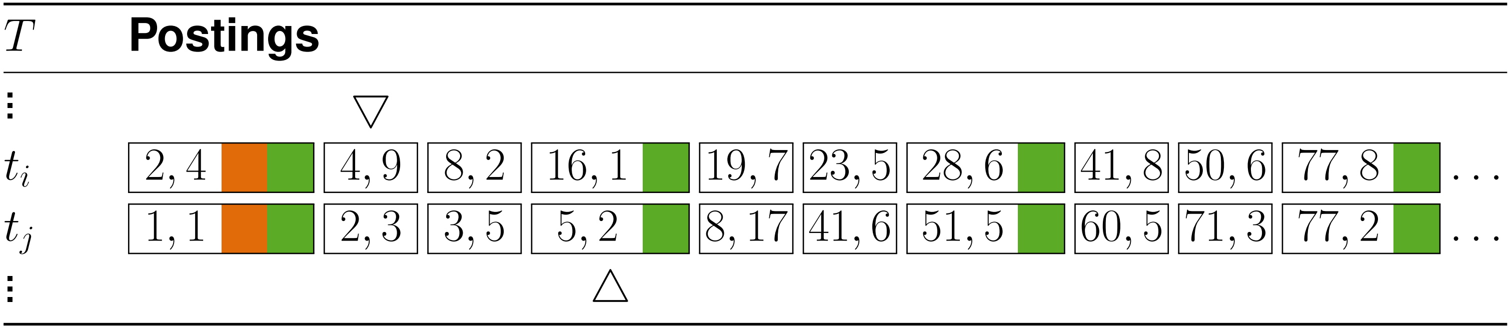
Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) = document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) < document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) < document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:



T	Postings										
⋮		▽									
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	$\ldots$
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	$\ldots$
⋮				△							

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮			▽								
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮				△							

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮			▽								
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	...
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	...
⋮					△						

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮				▽							
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮						△					

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮							▽				
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	$\ldots$
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	$\ldots$
⋮						△					

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮								▽			
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	…
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	…
⋮						△					

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8, 41\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮											
									▽		
t_i	$2,4$ 🟧🟩	$4,9$	$8,2$	$16,1$ 🟩	$19,7$	$23,5$	$28,6$ 🟩	$41,8$	$50,6$	$77,8$ 🟩	...
t_j	$1,1$ 🟧🟩	$2,3$	$3,5$	$5,2$ 🟩	$8,17$	$41,6$	$51,5$ 🟩	$60,5$	$71,3$	$77,2$ 🟩	...
							△				
⋮											

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8, 41\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮										▽	
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	…
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	…
⋮							△				

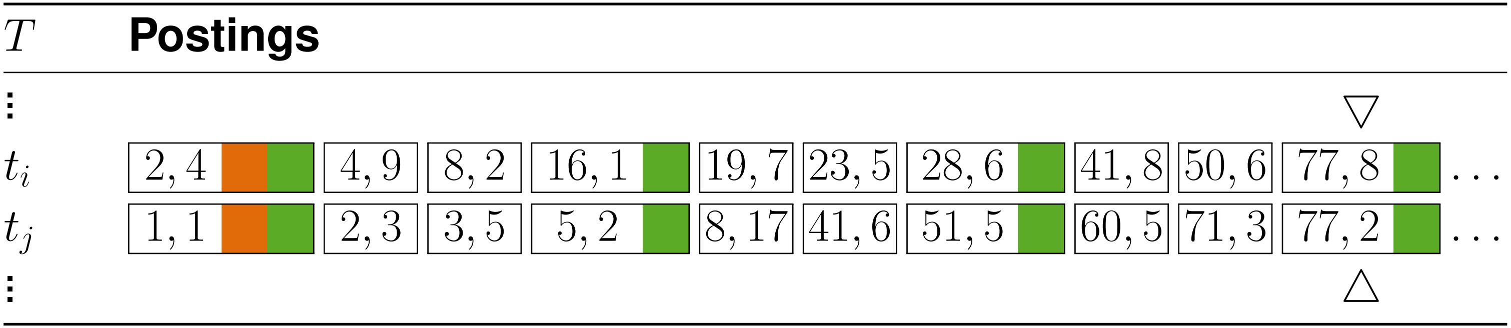
Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8, 41\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),   next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:



T	Postings										
⋮										▽	
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮										△	

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8, 41, 77\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Query Processing

Postlist Intersection

Let π_i denote the postlist of term t_i, ordered by document identifier:

T	**Postings**										
⋮											▽
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮											△

Given a Boolean query $q = t_i \wedge t_j$, the postlist intersection is computed as follows:

```
procedure INTERSECT($\pi_i$, $\pi_j$):
result $\leftarrow \{2, 8, 41, 77\}$
while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL do
    if document($\pi_i$) $=$ document($\pi_j$) then
        result $\leftarrow$ result $\cup$ document($\pi_i$)
        next($\pi_i$),    next($\pi_j$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_i$) $<$ document($\pi_j$) do
        if hasSkipTarget($\pi_i$, document($\pi_j$)) then skip($\pi_i$, document($\pi_j$)) else next($\pi_i$)
    while $\pi_i \neq$ NIL and $\pi_j \neq$ NIL and document($\pi_j$) $<$ document($\pi_i$) do
        if hasSkipTarget($\pi_j$, document($\pi_i$)) then skip($\pi_j$, document($\pi_i$)) else next($\pi_j$)
return result
```

Remarks:

- Postlists are typically too large to fit into main memory so that they cannot be accessed like an array, but have to be iterated.
- The *document* function returns the document identifier stored in a posting.
- The *next* function advances the postlist iterator to the next posting.
- The *hasSkipTarget* function checks whether the current posting has skip information, and whether a target with a document identifier smaller than the one passed is available.
- The *skip* function advances the postlist iterator to the posting nearest the passed skip target.
- For simplicity, the result of the Intersect procedure consists of only a set of document identifiers. In practice, it would return postings. Here, the question arises which postings to pick, those of π_i or those of π_j: if no other information than the document identifier is present or required for later processing, it does not matter which one is picked. Otherwise, a new posting must be constructed by merging the ones found having a matching document identifier.

Query Processing

Postlist Intersection

Given a Boolean AND query $q = t_1, \ldots, t_n$ comprising more than two terms, the required postlist intersections are conducted in increasing order of postlist length:

```
procedure INTERSECT($t_1, \ldots, t_n$):
terms ← sortByIncreasingPostlistLength($t_1, \ldots, t_n$)
$t_i$ ← first(terms)
terms ← rest(terms)
result ← $\pi_i$
while terms ≠ NIL and result ≠ NIL do
    $t_j$ ← first(terms)
    terms ← rest(terms)
    result ← INTERSECT(result, $\pi_j$)
return result
```

Observations:

- The amount of memory required to store the result postlist is bounded by the shortest postlist of the terms $t_1, \ldots, t_n$.
- Hard disk seeking is minimized since every postlist is read sequentially.
- The smaller the result, the more effective are the skip pointers.

Query Processing

Positional Indexing

Given a phrasal query $q =$"$t_1 \ \ldots \ t_m$", retrieving documents that contain the query's terms in that specific order requires positional indexing.

Example:

T	**Postings**		
to	. . .	$\boxed{4, 250, (..., 133, 137, ...)}$	. . .
be	. . .	$\boxed{4, 125, (..., 134, 138, ...)}$	. . .
or	. . .	$\boxed{4, 40, \ (..., 135, ...)}$	. . .
not	. . .	$\boxed{4, 15, \ (..., 136, ...)}$	. . .

What phrase does document 4 contain?

Query Processing

Positional Indexing

Given a phrasal query $q =$“$t_1 \ \ldots \ t_m$”, retrieving documents that contain the query’s terms in that specific order requires positional indexing.

Example:

T	**Postings**
`to`	$\ldots$ $\boxed{4, 250, (..., 133, 137, ...)}$ $\ldots$
`be`	$\ldots$ $\boxed{4, 125, (..., 134, 138, ...)}$ $\ldots$
`or`	$\ldots$ $\boxed{4, 40, \ (..., 135, ...)}$ $\ldots$
`not`	$\ldots$ $\boxed{4, 15, \ (..., 136, ...)}$ $\ldots$

Document 4 contains the phrase
`to be or not to be`
at term positions 133-138.

During postlist intersection, for each pair of matching postings, their position lists are compared to ensure that the terms in question occupy positions relative to each other that correspond to their relative positions in the query.

The algorithm for position list comparison works like that for postlist intersection.
The runtime for query processing is in $O(\sum_{d \in D} |d|)$.
The space requirements are 2-4 times that of a nonpositional index.

Query Processing

Positional Indexing

Given a phrasal query $q =$“$t_1 \ \ldots \ t_m$”, retrieving documents that contain the query’s terms in that specific order requires positional indexing.

Example:

T	**Postings**
`to be`	$\ldots$ $4, 250, (..., 133, 137, ...)$ $\ldots$
`be or`	$\ldots$ $4, 125, (..., 134, 138, ...)$ $\ldots$
`or not`	$\ldots$ $4, 40, \ (..., 135, ...)$ $\ldots$
`not to`	$\ldots$ $4, 15, \ (..., 136, ...)$ $\ldots$

Document 4 contains the phrase
`to be or not to be`
at term positions 133-138.

To speed up phrasal search, n-grams can be used as index terms:
How much faster can phrasal queries be processed?
What about space requirements?

Query Processing

Positional Indexing

Given a phrasal query $q =$"$t_1 \ \ldots \ t_m$", retrieving documents that contain the query's terms in that specific order requires positional indexing.

Example:

T	**Postings**
`to be`	$\ldots$ $\boxed{4, 250, (\ldots, 133, 137, \ldots)}$ $\ldots$
`be or`	$\ldots$ $\boxed{4, 125, (\ldots, 134, 138, \ldots)}$ $\ldots$
`or not`	$\ldots$ $\boxed{4, 40, (\ldots, 135, \ldots)}$ $\ldots$
`not to`	$\ldots$ $\boxed{4, 15, (\ldots, 136, \ldots)}$ $\ldots$

Document 4 contains the phrase
`to be or not to be`
at term positions 133-138.

To speed up phrasal search, n-grams can be used as index terms:
The time to process phrasal queries of length at least n is divided by n.

n-gram indexes for $n > 1$ are no substitute for 1-gram indexes; space requirements are k times the space of a 1-gram index, where k are the values of n indexed.

Relaxation: index n-grams without positional information; intersect postlists of overlapping n-grams from the phrasal query: some likelihood of wrong results.

Query Processing

Positional Indexing

Given a keyword query $q = \ldots\ t_i\ \ldots\ t_j\ \ldots$, documents that contain t_i and t_j in close proximity are likely more relevant than documents where the terms are far apart.

Let p_i, p_j denote the term position lists of t_i, t_j in a given document d. Whether t_i is in an ϵ-environment of t_j, where $\epsilon > 1$, can be determined as follows:

```
procedure ProximityIntersect($p_i, p_j, \epsilon$):
result $\leftarrow$ {}
while $p_i \neq$ NIL do
    matches $\leftarrow$ {}
    while $p_j \neq$ NIL and position($p_j$) $\leq$ position($p_i$) do
        if position($p_i$) $-$ position($p_j$) $\leq \epsilon$ then
            add(matches, position($p_j$))
        next($p_j$)
    for each match $\in$ matches do
        add(result, position($p_i$),match)
    next($p_i$)
return result
```

Remarks:

- Most retrieval models do not encode positional information, so that keyword proximity is used as an additional relevance signal or as prior probability for a document.

Query Processing

Document Scoring

In general, a query q is processed as a disjunctive query, where each term $t_i \in q$ may or may not occur in a relevant document d, as long as at least one t_i occurs.

The two most salient strategies for index-based document scoring are as follows:

- ❑ Document-at-a-time scoring
  - Precondition: a total order of documents in the index's postings lists is enforced (e.g., ordering criterion document ID or rather document quality).
  - Postlists of a query's terms are traversed in parallel to score one document at a time.
  - Each document's score is instantly complete, but the ranking only at the end.
  - Concurrent disk IO overhead increases with query length.
- ❑ Term-at-a-time scoring
  - Traverse postlists one a time (e.g., term ordering criterion frequency or importance).
  - Maintain temporary query postlist, containing candidate documents.
  - As each document's score accumulates, an approximate ranking becomes available.
  - More main memory required for maintaining temporary postlist.
- ❑ Safe and unsafe optimizations exist, e.g., to stop the search early.

Query Processing

Document Scoring

Given a query $q = t_1, \ldots, t_{|q|}$, its representation $\mathbf{q}$, and a relevance function ρ, in document-at-a-time scoring the postlists of $\mathbf{q}$'s terms are traversed concurrently:

```
procedure DAATScoring($\mathbf{q}$)
results $\leftarrow$ *priorityqueue*()
continue $\leftarrow$ TRUE
while continue do
    current $\leftarrow \infty$
    for all terms $t_i \in \mathbf{q}$ do
        if $\pi_i \neq$ NIL and *document*$(\pi_i) <$ current then current $\leftarrow$ *document*$(\pi_i)$
    $\mathbf{d} \leftarrow$ *representation*()
    for all terms $t_i \in \mathbf{q}$ do
        if $\pi_i \neq$ NIL and *document*$(\pi_i) =$ current then *add*$(\mathbf{d}, t_i,$ *weight*$(\pi_i))$
    *add*(results, current, $\rho(\mathbf{q}, \mathbf{d})$)
    continue $\leftarrow$ FALSE
    for all terms $t_i \in \mathbf{q}$ do
        if $\pi_i \neq$ NIL and *document*$(\pi_i) =$ current then
            *next*$(\pi_i)$
        if $\pi_i \neq$ NIL then continue $\leftarrow$ TRUE
return results
```

Remarks:

- DAAT = Document at a time
- Postlist skip pointers are not needed because of the disjunctive query semantics.
- Lines 7-9 reconstruct the representation $\mathbf{d}$ for d from the inverted index.
- Document-at-a-time scoring makes heavy use of disk seeks. With increasing query length $|q|$, dependent on the disks used and the distribution of the index across disks, the runtime of this approach is poor.
- Document-at-a-time scoring has a very small memory footprint on the order of the number of documents. This footprint can easily be bounded within top-k retrieval by limiting the size of the priority queue to k entries.
- Document-at-a-time scoring presumes a postlist ordering by document identifier.

Query Processing

Document Scoring

Given a query $q = t_1, \; \ldots, \; t_{|q|}$, its representation $\mathbf{q}$, and a relevance function ρ, in term-at-a-time scoring the postlists of $\mathbf{q}$'s terms are traversed iteratively:

```
procedure TAATScoring($\mathbf{q}$)
accumulators $\leftarrow$ hashmap()
for all terms $t_i \in \mathbf{q}$ do
    while $\pi_i \neq$ NIL do
        update(accumulators, document($\pi_i$), weight($\pi_i$))
        next($\pi_i$)
results $\leftarrow$ priorityqueue(accumulators)
return results
```

Term at a time scoring has a comparably high main memory load but reads every postlist consecutively. It can be implemented so that every postlist is read in parallel (e.g., from different disks).

Remarks:

- TAAT = Term at a time
- Postlist skip pointers are not needed because of the disjunctive query semantics.
- The *update* function updates the accumulated document score for the document posting currently read based on the term weight stored in the posting. This presumes that the relevance function ρ of the underlying retrieval model is additive.
- The order in which terms are processed (Line 2) affect how quick the accumulators accurately approximate the final document scores.
- Term at a time scoring makes no assumptions about postlist ordering.

Query Processing

Document Scoring

Given a single-term query $q = t$, the optimal postlist ordering is by term weight.

Example:

T	**Postings** (ordered by document identifier)										
⋮											
t_i	$2, 4$ 🟧🟩	$4, 9$	$8, 2$	$16, 1$ 🟩	$19, 7$	$23, 5$	$28, 6$ 🟩	$41, 8$	$50, 6$	$77, 8$ 🟩	$\ldots$
t_j	$1, 1$ 🟧🟩	$2, 3$	$3, 5$	$5, 2$ 🟩	$8, 17$	$41, 6$	$51, 5$ 🟩	$60, 5$	$71, 3$	$77, 2$ 🟩	$\ldots$
⋮											

In the worst case, the last document of the postlist is the most relevant one. Hence, the whole postlist must be examined.

Query Processing

Document Scoring

Given a single-term query $q = t$, the optimal postlist ordering is by term weight.

Example:

T	**Postings** (ordered by term weight)										
$\vdots$											
t_i	$4, 9$ 🟧🟩	$41, 8$	$77, 8$	$19, 7$ 🟩	$28, 6$	$50, 6$	$23, 5$ 🟩	$2, 4$	$8, 2$	$16, 1$ 🟩	$\ldots$
t_j	$8, 17$ 🟧🟩	$41, 6$	$3, 5$	$51, 5$ 🟩	$60, 5$	$2, 3$	$71, 3$ 🟩	$5, 2$	$77, 2$	$1, 1$ 🟩	$\ldots$
$\vdots$											

By definition of term weighting schemes, the document to which a term t is most important is the one with the highest term weight.

Query Processing

Document Scoring

Given a single-term query $q = t$, the optimal postlist ordering is by term weight.

Example:

T	**Postings** (ordered by term weight)										
⋮											
t_i	$4, 9$ 🟧	$41, 8$	$77, 8$	$19, 7$	$28, 6$	$50, 6$	$23, 5$	$2, 4$	$8, 2$	$16, 1$	$\ldots$
t_j	$8, 17$ 🟧	$41, 6$	$3, 5$	$51, 5$	$60, 5$	$2, 3$	$71, 3$	$5, 2$	$77, 2$	$1, 1$	$\ldots$
⋮											

By definition of term weighting schemes, the document to which a term t is most important is the one with the highest term weight.

Including a skip list in a postlist ordered by term weights is not useful.

Query Processing

Top-k Retrieval

The user of a search engine is only interested in the top-ranked k documents and will only look at those. All other documents retrieved and ranked will likely never be shown to the user.

Ideas for optimizing query processing:

- ❑ Term score threshold

  Disregard terms from a query that have a significantly lower inverse document frequency than other terms from the same query (e.g., in term-at-a-time scoring).

- ❑ Document score threshold

  In document-at-a-time scoring, once k documents have been found, determine which terms co-occcur in documents exceeding the k-th most relevant document, and skip documents where the terms in question do not co-occur.

- ❑ Early termination

  In indexes with postlists ordered by term weight, stop postlist traversal early, disregarding the rest of the postlist.

- ❑ Tiered indexes

  Divide documents into index tiers by quality or term frequency. If an insufficient amount of documents is found in the top tier, resort to the next one.

Query Processing

Index Distribution

The larger the size of the document collection D to be indexed, the more query processing time can be improved by scaling up and scaling out.

- ❑ Term distribution

  Distributing postlists across local storage devices allows for parallelization, which pertains particularly to spinning hard disks.

- ❑ Document distribution (also: sharding)

  Dividing the document collection into subsets by some criteria (so-called shards), and indexing each shard on a different index server adds a level of indirection: a query broker dispatches a query to index servers, which process a query as explained above. The broker fuses the results returned by index servers.

- ❑ Tiered indexes

  Index tiers are distributed across index servers, and optionally across device types within a server: for example, the index of important shards (tier 1) may be kept in RAM at all times, whereas tier 2 shards are kept in flash memory, and tier 3 shards on spinning hard disks.

Query Processing

Caching

Queries obey Zipf's law: roughly half the queries a day are unique, the other half has occurred before. Some queries are extremely frequent.

Caching can be applied at various points:

- ❑ Result caching
- ❑ Caching of postlist intersections
- ❑ Exploit cache hierarchies of hardware and operating system

Individual cache refresh strategies must be employed to avoid stale data.