DBLP Searcher

Stefano Dottore

Gestione dell’Informazione – UNIMORE – 2018/2019

Obbiettivi del progetto (traccia)

- Realizzazione di un sistema di ricerca full-text che consenta di effettuare ricerche avanzate nella bibliografia di DBLP mostrando i risultati ordinati secondo un modello di **ranking**.

- Il sistema realizzato è quindi responsabile di due aspetti
  - Creazione e gestione degli **indici** a partire dal file XML di DBLP
  - Supporto a **ricerche** full-text in base al particolare linguaggio proposto

Sintassi del linguaggio di ricerca

f-t-s: ([element-field:] search-pattern)+

search-pattern: term | "phrasal terms"

element-field: publication-search | venue-search

publication-search: publication-element[.publication-field]
publication-element: publication | article | incollection |
inproc | phThesis | masterThesis
publication-field: author | title | year

venue-search: venue[.venue-field]
venue-field: title | publisher

Approccio di sviluppo

- Lo sviluppo del software richiesto è stato diviso nelle seguenti macro fasi:
  - Analisi e parsing del **file XML**
  - Modellazione degli **indici**
  - Parsing della **query**
  - **Algoritmo di ricerca** basato sugli indici creati

Analisi XML

Analisi file XML

- La prima fase si è focalizzata sull'**analisi del file XML** e sul contenuto degli elementi coinvolti nel linguaggio di ricerca proposto.

- Gli obbiettivi perseguiti per la fase di analisi sono stati i seguenti:
  - Individuazione e distinzione di elementi *publication* e *venue*
  - Individuazione di tratti comuni/non comuni fra gli elementi
  - Individuazione di caratteristiche rilevanti/non rilevanti ai fini dell'applicazione (sulla base della sintassi del linguaggio di ricerca)

Analisi file XML: article

```
<article mdate="2017-05-28" key="journals/acta/Simon83">
    <author>Hans Ulrich Simon</author>
    <title>Pattern Matching in Trees and Nets.</title>
    <pages>227-248</pages>
    <year>1983</year>
    <volume>20</volume>
    <journal>Acta Inf.</journal>*
    <url>db/journals/acta/acta20.html#Simon83</url>
    <ee>https://doi.org/10.1007/BF01257084</ee>
</article>
```

ELEMENTO XML:	article
ELEMENTO DBLP SEARCHER:	article
TIPOLOGIA:	publication
CAMPI RILEVANTI:	author, title, year
CAMPO RIFERIMENTO ESTERNO:	journal

Analisi file XML: incollection

```
<incollection mdate="2017-05-20" key="journals/lncs/FormicaM93">
    <author>Anna Formica</author>
    <author>Michele Missikoff</author>
    <title>Modeling Semantic Interity Contraints in Database</title>
    <pages>129-147</pages>
    <year>1993</year>
    <crossref>books/sp/Atzeni93</crossref>
    <booktitle>Deductive Databases with Complex Objects</booktitle>
    <url>db/journals/lncs/lncs701.html#FormicaM93</url>
    <ee></ee>
</incollection>
```

ELEMENTO XML:	incollection
ELEMENTO DBLP SEARCHER:	incollection
TIPOLOGIA:	publication
CAMPI RILEVANTI:	author, title, year
CAMPO RIFERIMENTO ESTERNO:	crossref (→ *book*)

Analisi file XML: inproceedings

```
<inproceedings mdate="2017-05-20" key="journals/lncs/Smolka94">
    <author>Gert Smolka</author>
    <title>The Definition of Kernel Oz</title>
    <pages>251-292</pages>
    <booktitle>Constraint Programming</booktitle>
    <crossref>journals/lncs/1994-910</crossref>
    <year>1994</year>
    <url>db/journals/lncs/lncs910.html#Smolka94</url>
    <ee>https://doi.org/10.1007/3-540-59155-9_14</ee>
</inproceedings>
```

ELEMENTO XML:	inproceedings
ELEMENTO DBLP SEARCHER:	inproc
TIPOLOGIA:	publication
CAMPI RILEVANTI:	author, title, year
CAMPO RIFERIMENTO ESTERNO:	crossref (→ *proceedings*)

Analisi file XML: phdthesis

```
<phdthesis mdate="2016-10-06" key="phd/ethos/Diakonikolaou92">
    <author>George Diakonikolaou</author>
    <title>Methodological support for developing complex information
systems</title>
    <year>1992</year>
    <school>University of Manchester, UK</school>
    <ee>http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.335120</ee>
    <note type="source">British Library, EthOS</note>
</phdthesis>
```

ELEMENTO XML:	phdthesis
ELEMENTO DBLP SEARCHER:	phThesis
TIPOLOGIA:	publication
CAMPI RILEVANTI:	author, title, year
CAMPO RIFERIMENTO ESTERNO:	*Nessuno*

Analisi file XML: masterthesis

```
<mastersthesis mdate="2002-01-03" key="phd/VanRoy84">
    <author>Peter Van Roy</author>
    <title>A Prolog Compiler for the PLM.</title>
    <year>1984</year>
    <school>University of California at Berkeley</school>
</mastersthesis>
```

ELEMENTO XML:	mastersthesis
ELEMENTO DBLP SEARCHER:	masterThesis
TIPOLOGIA:	publication
CAMPI RILEVANTI:	author, title, year
CAMPO RIFERIMENTO ESTERNO:	*Nessuno*

Analisi file XML: proceedings

```
<proceedings mdate="2018-12-21" key="conf/sohoma/2018">
    <editor>Theodor Borangiu</editor>
    <editor>Damien Trentesaux</editor>
    <editor>Sergio Cavalieri</editor>
    <title>Service Orientation in Holonic and Multi-Agent Manufacturing
- Proceedings of SOHOMA 2018, Bergamo, Italy, June 11-12, 2018</title>
    <booktitle>SOHOMA</booktitle>
    <publisher>Springer</publisher>
    <year>2019</year>
    <series href="db/series/sci/index.html">Studies in Computational
Intelligence</series>
    <volume>803</volume>
    <isbn>978-3-030-03003-2</isbn>
    <ee>https://doi.org/10.1007/978-3-030-03003-2</ee>
    <url>db/conf/sohoma/sohoma2018.html</url>
</proceedings>
```

ELEMENTO XML:	proceedings
ELEMENTO DBLP SEARCHER:	proceedings
TIPOLOGIA:	venue
CAMPI RILEVANTI:	title, publisher, year

Analisi file XML: book

```
<book mdate="2018-07-28" key="phd/ethos/Dunsmore03">
    <author>Alastair Peter Dunsmore</author>
    <title>Reading techniques for object-oriented code of
production.</title>
    <year>2003</year>
    <publisher>University of Strathclyde, Glasgow, UK</publisher>
    <ee>http://ethos.bl.uk/OrderDetails.do?uin=uk.bl.ethos.269999</ee>
    <note type="source">British Library, EthOS</note>
</book>
```

ELEMENTO XML:	book
ELEMENTO DBLP SEARCHER:	book
TIPOLOGIA:	venue
CAMPI RILEVANTI:	author, title, publisher, year

Analisi file XML: journal !?

- Gli elementi *article* possiedono il campo *journal* che stabilisce quindi un riferimento esterno tra la publicazione e la rispettiva venue, come per gli elementi *inproceedings* e *incollection*.

  A differenza di questi ultimi, per la quale esiste effettivamente un elemento all'interno del file XML che ha come *key* il valore specificato in *crossref* della pubblicazione, gli elementi *journal* di fatto non sono presenti nel documento.

  Inoltre, il valore del campo journal corrisponde al nome effettivo del journal, non ad una *key* identificativa.

- La gestione di questo caso particolare sarà spiegata nella sezione di modellazione degli indici.

Analisi file XML: statistiche

- Propedeuticamente alla modellazione degli indici è stata svolta come prima operazione sul file XML una scansione ed un ottenimento delle principali **caratteristiche e statistiche del file**.

  In particolare è stato osservato quanto segue (dump *dblp.xml* del 1 aprile 2019):

- Numero di elementi (rilevanti all'applicazione): 4.544.480
- Numero di termini diversi: 1.482.266
- Numero di occorrenze di termini (all'interno di campi rilevanti): 79.025.763
- Massimo numero di campi uguali per elemento (*author*): 286
- Massima posizione di un termine all'interno di un campo: 142
- Massimo numero di occorrenze dello stesso termine (*of*): 1.777.439

Modellazione indici

Modellazione indici: obbiettivi

- Le **operazioni minime** che gli indici modellati dovranno soddisfare sono le seguenti:
  - Poter ottenere, a partire da un termine, l'elenco degli elementi che contengono tale termine all'interno di un campo (fra quelli rilevanti ai fini dell'applicazione).

    In aggiunta, è necessario poter risalire anche alla posizione del termine all'interno del campo (per poter soddisfare query di tipo frasale).
  - Poter ottenere, a partire da una pubblicazione, la venue associata a cui questa appartiene.
- Seppur **non essenziali** al puro soddisfacimento delle query, gli indici modellati permettono inoltre le seguenti operazioni:
  - Poter ottenere la posizione di un elemento all'interno del dump XML indicizzato (per mostrare il contenuto dell'elemento, oltre che alla *key*)

Modellazione indici: sommario

- Gli **indici** proposti sono i seguenti:
  - Identificatori (`.idix`)
  - Posting list (`.plix`)
  - Vocabolario (`.vix`)
  - Crossrefs (`.cix`)
  - [Posizioni XML (`.xpix`)]

- Di seguito verranno illustrate le caratteristiche di ognuno di essi.

Modellazione indici: identificatori

- Per evitare di trattare ogni elemento con l'identificatore associato (generalmente l'attributo *key*), è stato assegnato ad ognuno di essi un numero seriale (`elem_serial`).

- Il ruolo dell'indice degli identificatori è poter risalire, a partire dal seriale di un elemento, all'identificatore corrispondente.

- L'indice per gli identificatori (`.idix`) è un **file testuale** contenente per ogni riga l'identificatore corrispondente all'elemento che ha come numero seriale il numero di riga.

Modellazione indici: identificatori

- **Esempio** di file `.idix`:

```
journals/acta/Saxena96(\n)
journals/fgcs/CasasTRWZ17(\n)
journals/ieeesp/MatyasR03(\n)
journals/isci/KonecnyK17(\n)
Multimedia Tools Appl.(\n) *
conf/oopsla/GallagherGL14(\n)
conf/icassp/XiaLCGM11(\n)
phd/ethos/Alazemi14(\n)
```

- Ad esempio, l'elemento con seriale `0` avrà come identificatore `journals/acta/Saxena96`, mentre quello con seriale 1 avrà come identificatore `journals/fgcs/CasasTRWZ17`.

Modellazione indici: identificatori

- Si osserva che il file `.idix` contiene principalmente gli attributi *key* degli elementi indicizzati, ad eccezione dei *journal* per cui è stato utilizzato il contenuto del campo *journal* degli elementi *article* come identificatori.

- I *journal* sono stati infatti trattati come veri e proprio elementi dotati di un numero seriale, il cui identificatore è il loro titolo.

Modellazione indici: posting list

- La memorizzazione della locazione di un termine all'interno del documento è attuata mediante il concetto dei *post.*
- Un post è definito dalla tupla:
  - Numero seriale dell'elemento (`elem_serial`)
  - Numero progressivo del campo all'interno dell'elemento (`field_num`) (Necessario per gestire, ad esempio, multipli campi *author*)
  - Posizione del termine all'interno del campo (`term_pos`)
- In accordo con l'analisi del file XML iniziale, le tre componenti dei post sono state così dimensionate:
  - `elem_serial` 23 bit (2^{23}-1 = 8.388.607 > 4.544.480)
  - `field_num` 9 bit (2^{9}-1 = 511 > 286)
  - `term_pos` 8 bit (2^{8}-1 = 255 > 142)

  Ogni post occupa perciò (al più) 40 bit, ossia 5 byte.

Modellazione indici: posting list

- Un esempio di post per la rappresentazione dell'elemento con seriale 50635, numero progressivo del campo uguale a 2 e posizione del termine all'interno del campo uguale a 6 sarebbe:
- 000 0000 **1100 0101 1100 1011** 0 0000 00**10** 0000 0**110**
  <elem_serial> | <field_num> | <term_pos>

- Per ottimizzare il consumo di memoria si è inoltre scelto di ridurre la dimensione dei post riguardanti campi che dalla struttura del file XML si è osservato non essere presenti più di una volta all'interno dello stesso elemento (tutti tranne *author*).
  È stato possibile utilizzare così 4 byte anziché 5 nella maggior parte dei casi.
- Un post a 4 byte presenta la seguente struttura:

* = padding

- *0 000 0000 **1100 0101 1100 1011** 0000 0**110**
  | <elem_serial> | <term_pos>

Modellazione indici: posting list

- L’indice per i post (`.plix`) è un **file binario** contenente una sequenza di post.
- Per agevolare la creazione dell’indice dei termini e la conseguente ricerca nell’indice, si è scelto di porre i post relativi allo stesso termine adiacenti fra loro, così come i post relativi allo stesso tipo di campo (all’interno dei post dello stesso termine).
- Si osserva che l’indice per i post è l’unico a non essere caricato in memoria a runtime ma risiede su memoria permanente (a causa della dimensione dell’indice prodotto, che seppur gestibile non è esigua).
- Il corretto utilizzo e funzionamento della posting list è associato alla correttezza di un altro indice affrontato in seguito: *il vocabolario*.

Modellazione indici: posting list

- **L'ordine** con cui i post si susseguono all'interno dello stesso termine indicizzato segue i seguenti tipi di campo (in ordine):

Publications

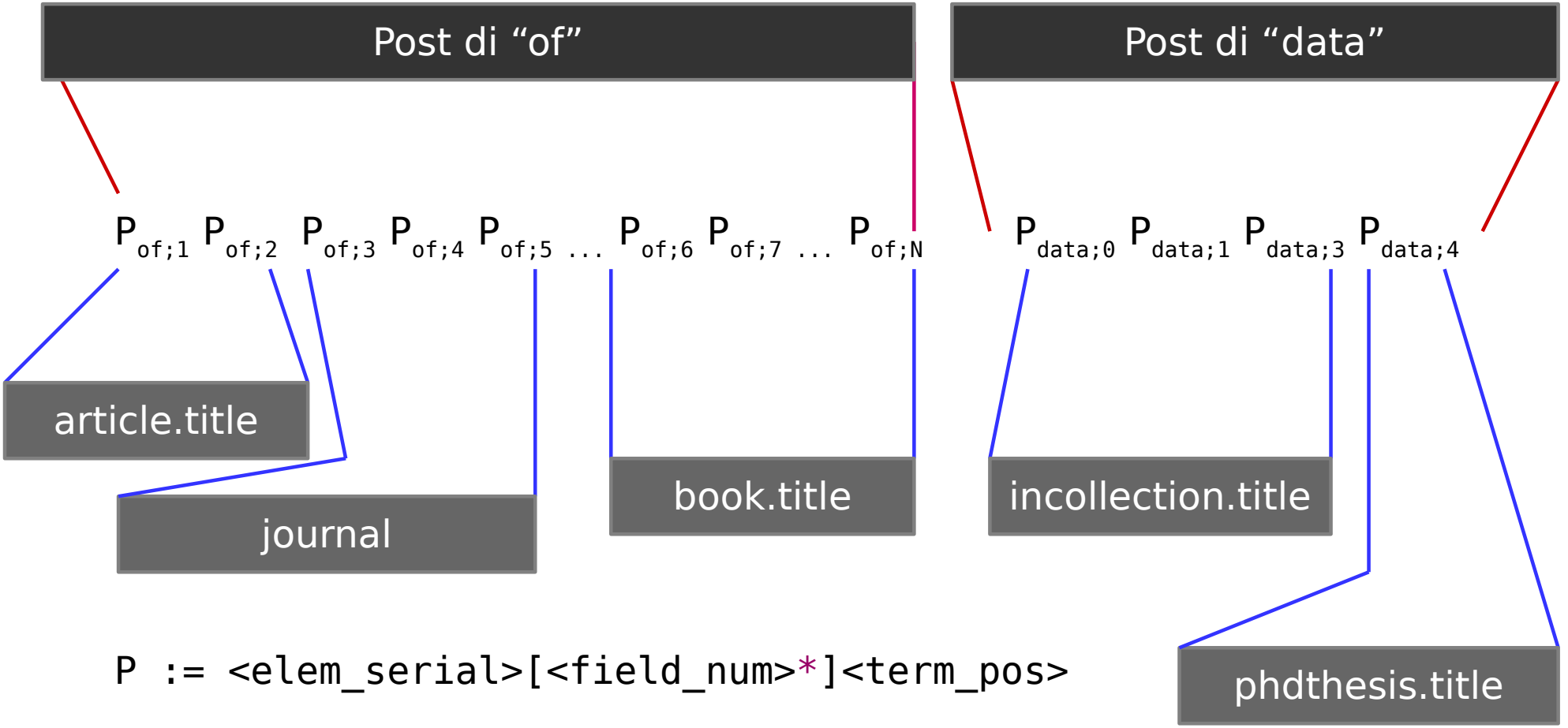
```
article.author, article.title, article.year
incollection.author, incollection.title, incollection.year
inproceedings.author, inproceedings.title, inproceedings.year
phdthesis.author, phdthesis.title, phdthesis.year
mastersthesis.author, mastersthesis.title, mastersthesis.year
```

Venues

```
journal
book.author, book.title, book.year, book.publisher
proceedings.title, proceedings.year, proceedings.publisher
```

Modellazione indici: posting list

- **Esempio** di file `.idix`:



`P := <elem_serial>[<field_num>*]<term_pos>`

* Presente solo per i campi di tipo author

Modellazione indici: vocabolario

- Per poter risalire al termine e al campo di appartenenza dei post indicizzati nella posting list è stato previsto un tipo di indice definito *vocabolario*.
- L'indice in questione (.vix) è un **file binario** contenente una sequenza di *riferimenti a termine della posting list*.
- Ogni *riferimento a termine della posting list* ha il seguente pattern:
  - Termine della posting list riferito (lunghezza variabile, UTF8)
  - Posizione iniziale del termine riferito nella posting list (64 bit)
  - Per ogni tipo di campo (precedentemente elencati per la modellazione della posting list) il numero di post presenti nella posting list appartenenti a tale campo.

Modellazione indici: vocabolario

- Per la memorizzazione del numero di post della posting list appartenenti ad un campo è stata utilizzata una codifica a lunghezza variabile.

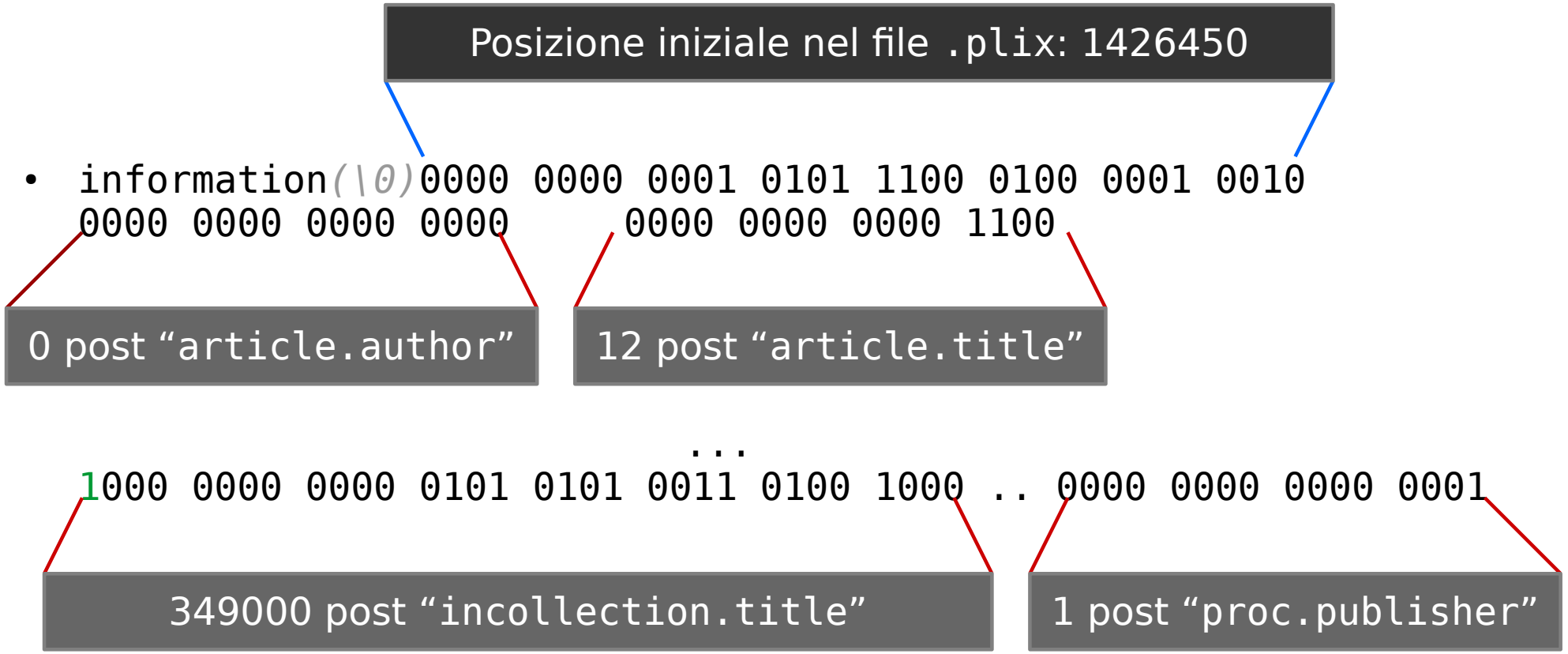
  Se il primo bit è uguale a `0`, il numero di post è dato dai primi 16 bit (quindi fino ad un massimo di $2^{15} - 1 = 32767$).

  Se il primo bit è uguale a 1, il numero dei post è dato dai successivi 31 bit, ossia dai primi 32 bit ad eccezione del primo bit.

- Per ogni tipo di campo deve comunque essere presente, nello stesso ordine con cui sono indicizzati nel file `.plix`, il numero di post della posting list facenti riferimento al campo (per lo stesso termine), sia esso anche 0.

Modellazione indici: vocabolario

- **Esempio** di file `.vix`:



Modellazione indici: crossrefs

- Per poter valutare se un match di una ricerca abbia riscontro sia su una pubblicazione che sulla relativa venue è stato modellato un indice definito *crossrefs*.

- L'indice crossrefs (`.cix`) è un **file binario** che contiene una sequenza di associazione fra i numeri seriali delle publicazioni e i numeri seriali delle venue associate ad essi (in base ai campi *crossref* o *journal*).

- L'ordine con cui le sequenze si susseguono è del tutto arbitrario.

Modellazione indici: crossrefs

- Ogni associazione è composta dai due seriali (23 bit):

- Per poter rappresentare i 46 bit richiesti è stato possibile 6 byte, ossia 48 bit, di cui i primi due bit sempre uguali a `0`.

- Un **esempio** di associazione fra la pubblicazione con seriale 7 e la venue con seriale 12 sarebbe:

- `00  00 0000 0000 0000 0000 0011 1  000 0000 0000 0000 0000 1100`
  `|  <pub_serial>  |  <venue_serial>  |`

Modellazione indici: posizioni XML

- Gli indici presentati fino ad ora (*identificatori*, *posting list*, *vocabolario* e *crossrefs*) sono indispensabili per l'effettivo soddisfacimento delle ricerche full-text.

- Per aumentare l'usabilità dell'applicazione è stato comunque introdotto un nuovo tipo di indice non indispensabile, quello delle *posizioni XML*.

- Il ruolo dell'indice delle posizioni XML è quello di poter risalire, a partire dal seriale di un elemento, alla posizione che questo occupava nel file XML originale su cui sono stati generati gli indici.

Modellazione indici: posizioni XML

- L’indice delle posizioni XML (`.xpix`) è un **file binario** che contiene una sequenza di posizioni di file, ognuna delle quali occupa 32 bit.

- L’*i*-esima posizione all’interno dell’indice indica la posizione dell’elemento con numero seriale uguale ad *i* all’interno del file XML originale.

- Per il modo con cui il file XML originale è stato letto (mediante letture bufferizzate), non è stato possibile risalire all’esatta posizione degli elementi nell’XML durante il parsing.

  La posizione salvata è perciò un upper bound della posizione esatta, che comunque si discosta dalla posizione esatta al più della dimensione del buffer usato per la lettura dell’XML.

Parsing della query

Query: chiarimenti

- Il parsing della query è stato svolto in accordo con il linguaggio proposto dalla traccia; di seguito si espongono alcune **considerazioni sulla semantica del linguaggio**.
- Qualsiasi stringa all'interno di doppi apici, ossia frasale, è considerata in modo trasparente come un token di cui si ignora il contenuto.

  Ad esempio, *"article.author:"* sarebbe trattato esattamente come una qualsiasi altra frase, non come l'inizio di un search pattern.
- Ogni token è considerato appartenente all'ultimo inizio di search pattern riconosciuto. Nel caso la query inizi con una serie di token non appartenenti ad un search pattern, essi sono considerati token "globali", cioè che non impongono nessun filtro.
- I risultati della query sono ottenuti dall'unione (non dall'intersezione) degli insiemi dei risultati che ciascun search pattern produrrebbe.

Query: chiarimenti

- Ogni search pattern può potenzialmente accogliere sia token normali che token frasali; ad esempio, è ammessa la query:
  article.title: “janus webrtc gateway” data science
- I risultati prodotti da una query possono essere di tre tipologie (esclusive):
  - Pubblicazione: un article o incollection o inproceedings o phdthesis o masterthesis soddisfa almeno un search pattern della query
  - Venue: un journal o book o proceedings soddisfa almeno un search pattern della query
  - Pubblicazione + Venue: una pubblicazione soddisfa almeno un search pattern della query e la venue associata soddisfa almeno un search pattern della query (non necessariamente lo stesso)
- Il search pattern *venue.publisher* non contribuisce alla produzione di un match pubblicazione + venue in cui la pubblicazione che matcha è un *article*: questo perché la venue associata (*journal*) non possiede il campo *publisher.*
  Un search pattern *venue.title* potrebbe invece contribuire alla produzione di un match pub + venue per un articolo in quanto è stato trattato come se il *title* del journal fosse il nome stesso di questo (nonostante tecnicamente non esista un campo *title* per i *journal*, in quanto questi non sono elementi)

Parsing della query: idea

- L'**algoritmo** adottato per il parsing della query è basato sulla seguente strategia:
  - Suddividere la query in input in modo da ottenere un'unica lista di token semplici e/o frasali
  - Analizzare la lista dei token ed inizializzare una nuova *query part* quando un token (semplice) corrisponde ad un inizio di search pattern noto (e.g. *author.title:*)
  - Aggiungere tutti i token successivamente analizzati alla *query part* inizializzata (fino all'inizializzazione della prossima *query part*)
  - Il modello della query ottenuto sarà quindi l'insieme delle *query part* ottenute dal processo di parsing

Parsing della query: pseudo codice

```
function parse_query(query_string): query

    query_tokens = [], query_parts = []
    macro_tokens = query_string.split_keep_empty_parts('"')

    // Distinguish phrasal from simple tokens
    for macro_token in macro_tokens
        if macro_token.empty()
            continue
        if is_odd_iteration()
            query_tokens.append(macro_token, PHRASAL)
        else // even iteration
            query_tokens.append(macro_token.split(' '), SIMPLE)

    // Handle search patterns among tokens for create query parts
    for token in query_tokens
        if token.is_phrasal() OR !token.is_search_pattern_begin()
            current_query_part.append(token)
        else
            parts.append(current_query_part)
            current_query_part = token.make_part_for_search_pattern_begin()

    // Add the last query part
    parts.append(current_query_part)

    return new_query(parts)
```

Algoritmo di risoluzione

Risoluzione query: idea

- L'**algoritmo** adottato per la risoluzione della query è basato sulla seguente strategia:
  - Per ogni token (frasale o semplice) della query, reperire i match dalla posting list (eventualmente applicando l'algoritmo di verifica di continuità dei post per i token frasali) ed inserirli opportunamente fra i match delle pubblicazioni o delle venue.
  - Calcolare lo score di ogni elemento fra quelli appartenenti ai match ottenuti, in base allo score associato ai termini per cui si è verificato il match. (Ossia calcolo della *tf.idf*).
  - Per ogni elemento fra le pubblicazioni che contengono match, verificare se la crossref di tale elemento è presente fra le venue che contengono match.

    Se sì: inserire un match di tipo publication+venue

    Se no: inserire un match di tipo publication
  - Aggiungere le venue non già aggiunte fra i match publication+venue
  - Ordinare i match in base allo score precedentemente calcolato

Risoluzione query: pseudo codice

```
function compute_score(matches): scored_matches
    scored_matches = {} // (serial => scored_element)
    for match in matches
        scored_element = scored_matches.find(match.elem_serial)
        scored_element.append_match(match)
        for token in match.matched_tokens()
            scored_element.score += ief(token)
    return scored_matches

function resolve_query(query): matches
    pub_matches = [], venue_matches = []

    // Find the matches for each token inside each query part
    for part in query.parts()
        for macro_token in part.tokens() // either phrasal or simple
            if part.is_publication()
                pub_matches.append(index.find_publication_matches(
                    macro_token, part.search_filter_fields()))
            else // is venue
                venue_matches.append(index.find_venue_matches(
                    macro_token, part.search_filter_fields()))

    // Compute the publications and venues score
    scored_pubs = compute_score(pub_matches)
    scored_venues = compute_score(pub_venues)
                                    ...
```

Risoluzione query: pseudo codice

```
                                                ...
crossreffed_venues = set()

// Distinguish publication from publication+venue matches (crossref check)
for scored_pub in scored_pubs
    crossref_serial = index.crossref(scored_pub.elem_serial)
    crossref_match = false
    if crossref_serial
        scored_venue = scored_venues.find(crossref_serial)
        if scored_venue
            matches.append(make_publication_venue_match(
                scored_pub, scored_venue))
            crossreffed_venues.put(scored_venue)
            crossref_match = true
    if !crossref_match
        matches.append(make_publication_match(scored_pub))

// Add the remaining venues
for scored_venue in scored_venues
    if !crossreffed_venues.contains(scored_venue.elem_serial)
        matches.append(make_venue_match(scored_venue))

return matches.sort() // sort by score
```

Risoluzione query: token frasali

- Nell'algoritmo di risoluzione della query proposto sono state utilizzate le funzioni `find_publication_matches` e `find_venue_matches`; l'implementazione di esse nel caso in cui il token di cui cercare i match fosse un token *semplice*, ossia composto da un solo termine, è immediata e consiste nell'effettuare una visita dell'indice delle posting list (in base alla posizione mantenuta dal dizionario per il termine richiesto).
- Nel caso in cui il token fosse invece *frasale*, ossia composto da più termini, è stata utilizzata la seguente strategia:
  - Reperire dalla posting list i post relativi ad ogni termine della frase ed inserirli in una hash table contenente le posizioni dei termini, indicizzata secondo lo specifico campo all'interno dell'elemento
  - Per ogni specifico elemento/campo fra i post ottenuti, verificare che:
    - Tutti i termini della frase siano presenti all'interno dell'elemento/campo
    - Le posizioni dei termini dell'elemento/campo rispettino le posizioni dei termini all'interno della frase.

Risoluzione query: token frasali

```
function find_phrase_matches(terms, field_type): matches

    terms_pos_by_elem_field = {} // (serial, field_num) => ((term) => positions)

    // For each term retrieve the associated posts and put those in
    // the hash table, mapped by (serial + field number)
    for term in terms
        term_posts = find_posts(term, field_type)

        if !term_posts
            // There won't be phrasal match, a term doesn't have posts
            return null

        for post in term_posts
            term_positions = terms_pos_by_elem_field.get(
                (post.elem_serial, post.field_num))
            term_positions.put(post.in_field_pos)
                                        ...
```

Risoluzione query: token frasali

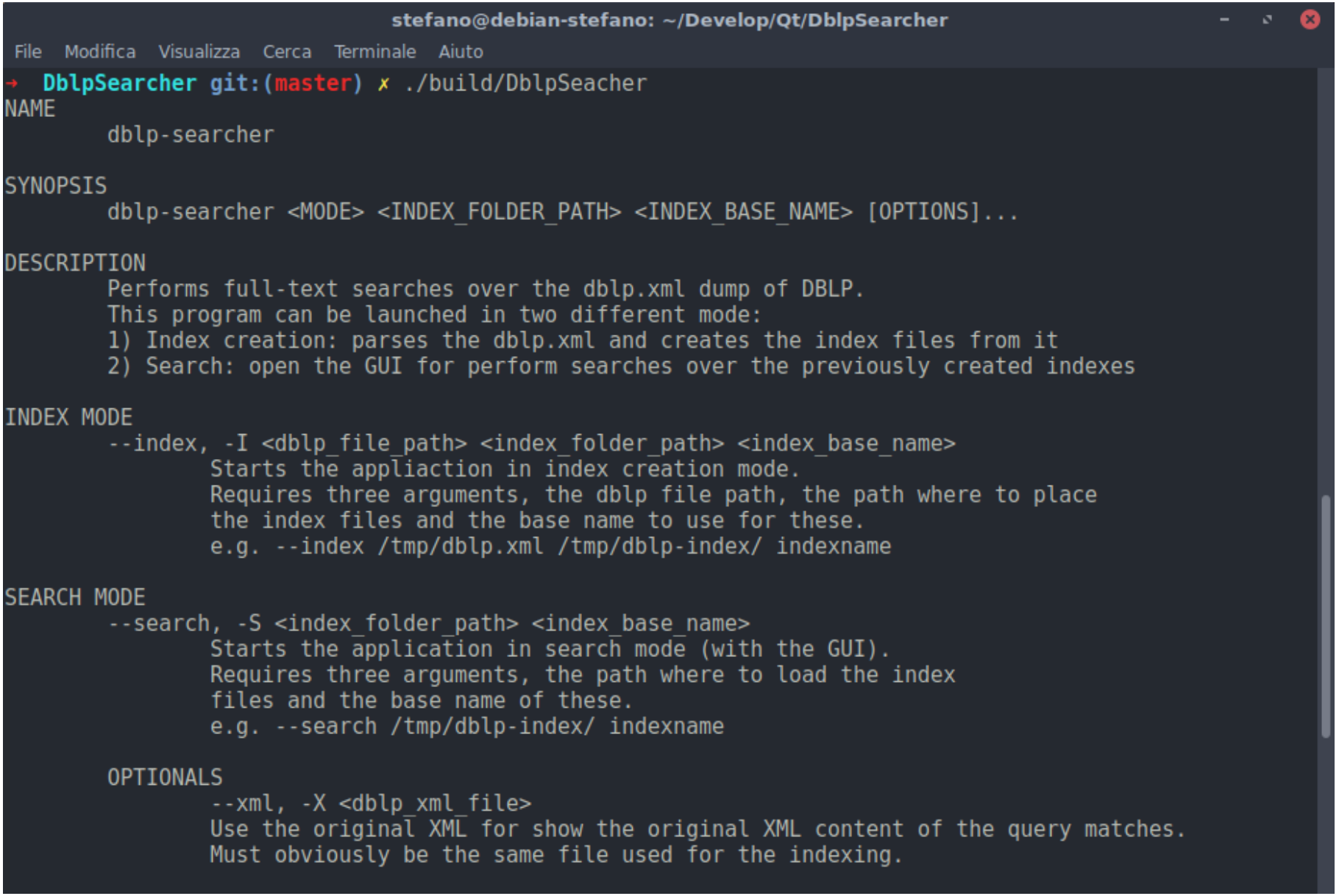
```
// Check consecutiveness within the same element + field of the terms
// accordingly to the required query terms
for (serial, field_num) , (term_positions) in terms_pos_by_elem_field
    pos_list_0 = term_positions.find(terms[0]))

    if !pos_list_0
        // This (arbitrary) term doesn't exists within this element + field
        continue

    for pos_0 in pos_list_0
        for (i, term) in terms; i != 0
            pos_list_i = term_positions.find(term)
            if !pos_list_i
                // element + field doesn't contain this term
                break twice
            if !pos_list_i.contains(pos_0 + i)
                // The term exists but not at this position
                break

        // Phrasal matches found: all the terms exist at the right positions
        matches.append(make_match(serial, field_type, field_num))

return matches
```

Applicazione

Applicazione: tecnologie utilizzate

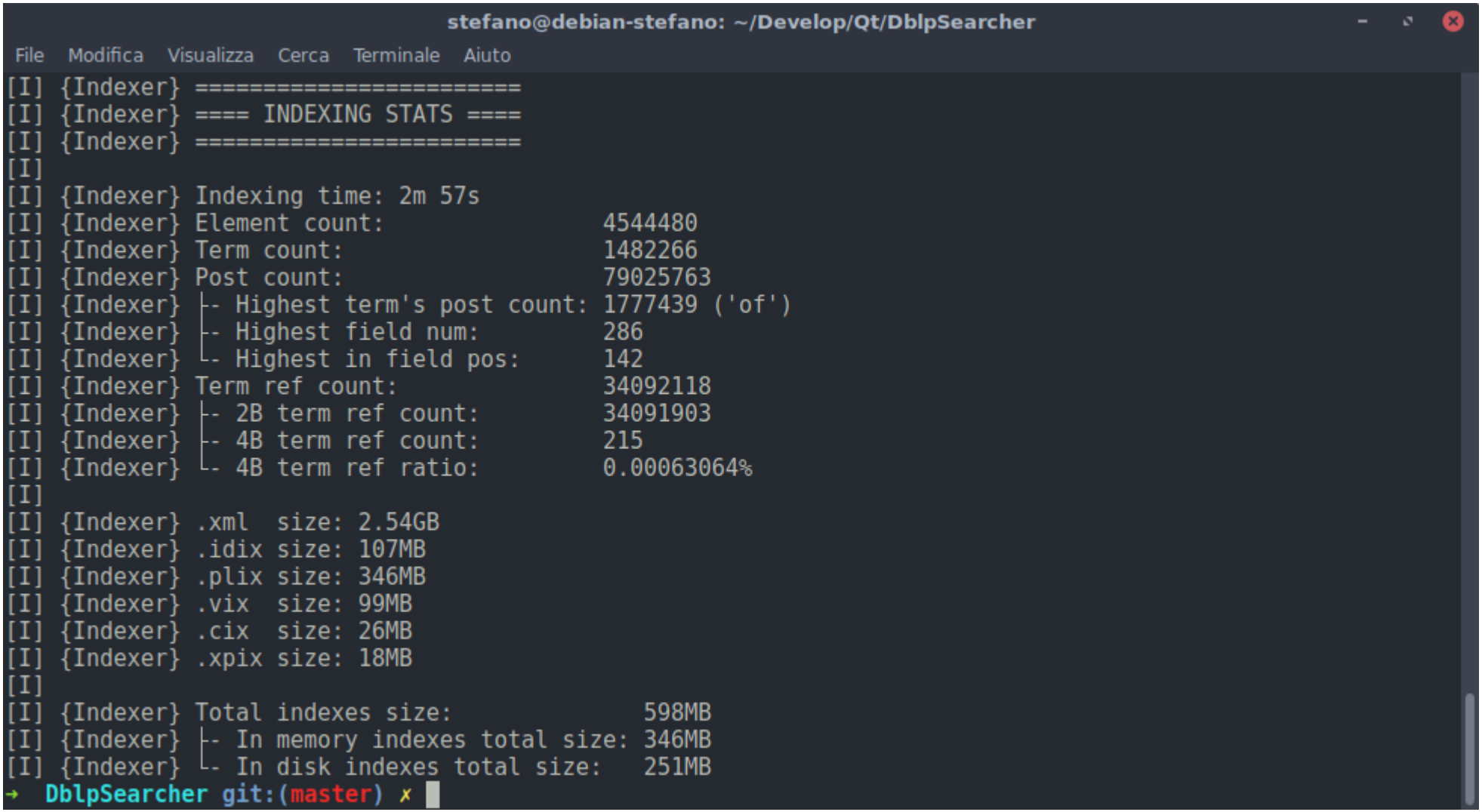
- L'applicazione *DBLP Searcher* è stata scritta in C++11 e Qt5.
- Per l'ottimizzazione di alcune parti di codice (la risoluzione delle query in particolare) è stato utilizzata una libreria dedicata al multithreading: OpenMP.
- Per la creazione degli indici e la loro gestione, così come per il resto dall'applicazione, **non** è stata alcuna libreria esterna di supporto (e.g. *CLucene*)
- Per l'interfaccia grafica (per la parte di ricerca) è stato utilizzato QML (linguaggio di markup per la scrittura di interfacce grafiche, integrato con Qt)

Applicazione: help?

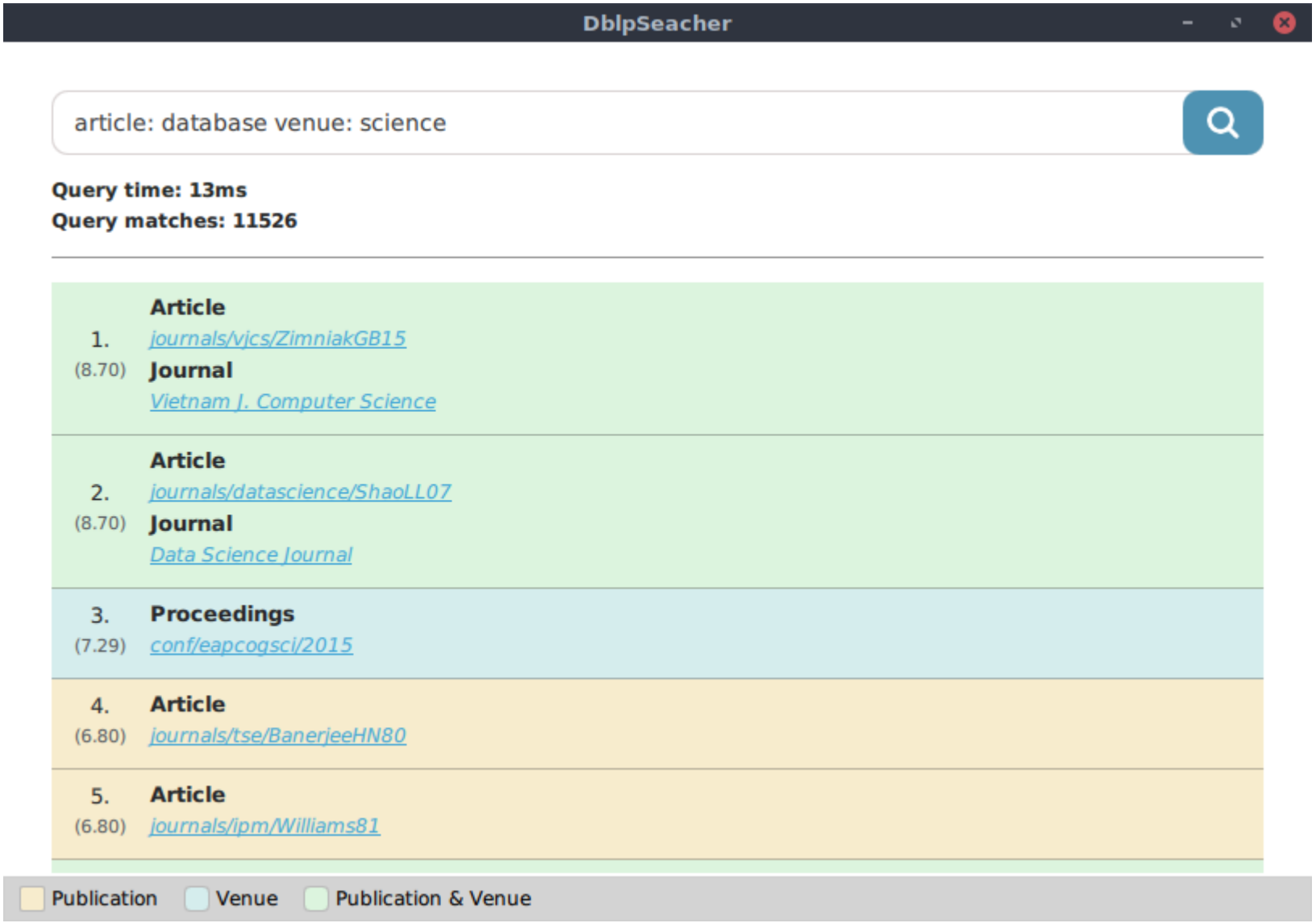


Applicazione: indicizzazione

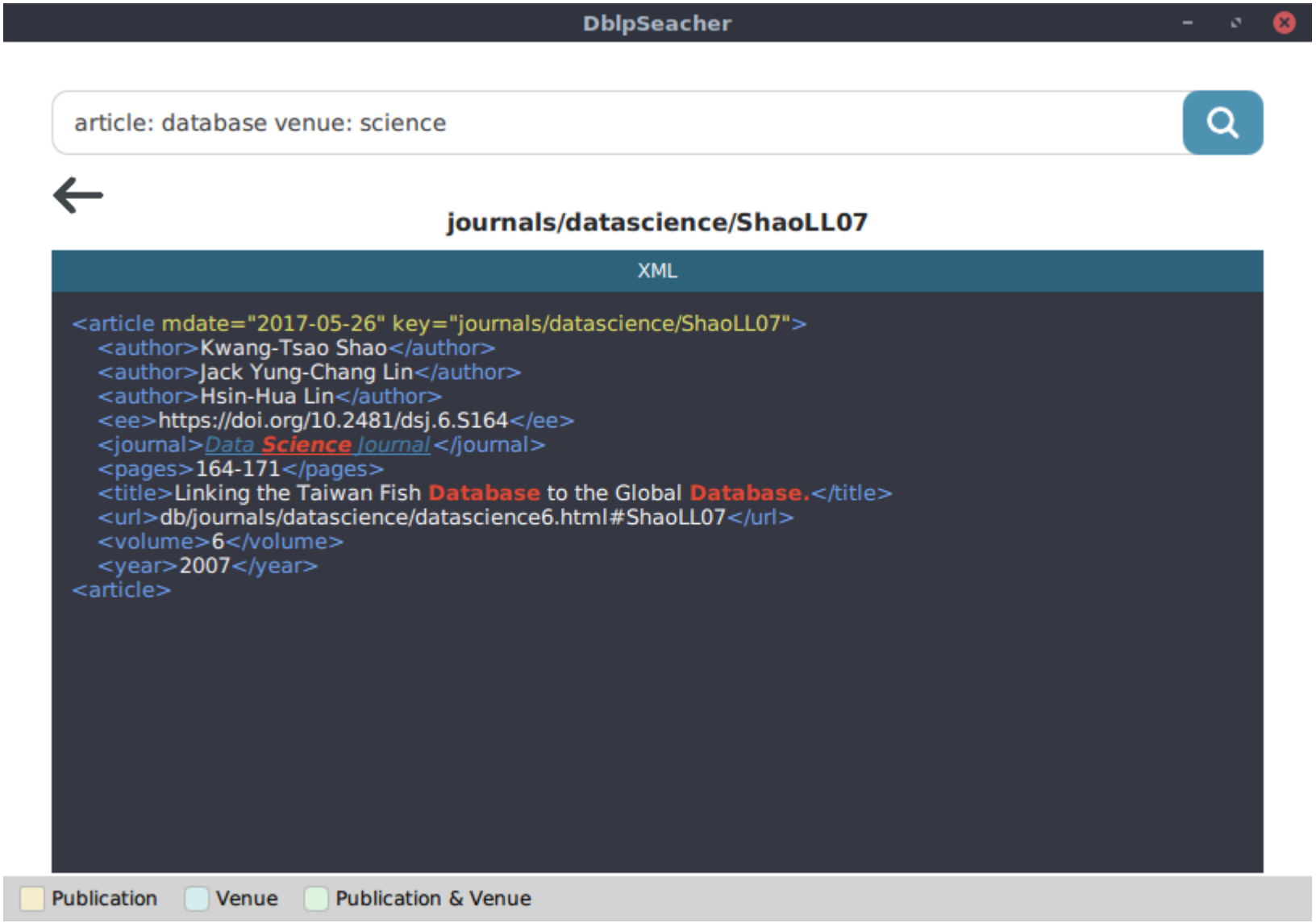
Esempio di indicizzazione di un file di 2.5GB (processore i7-4790K @ 4GHz)



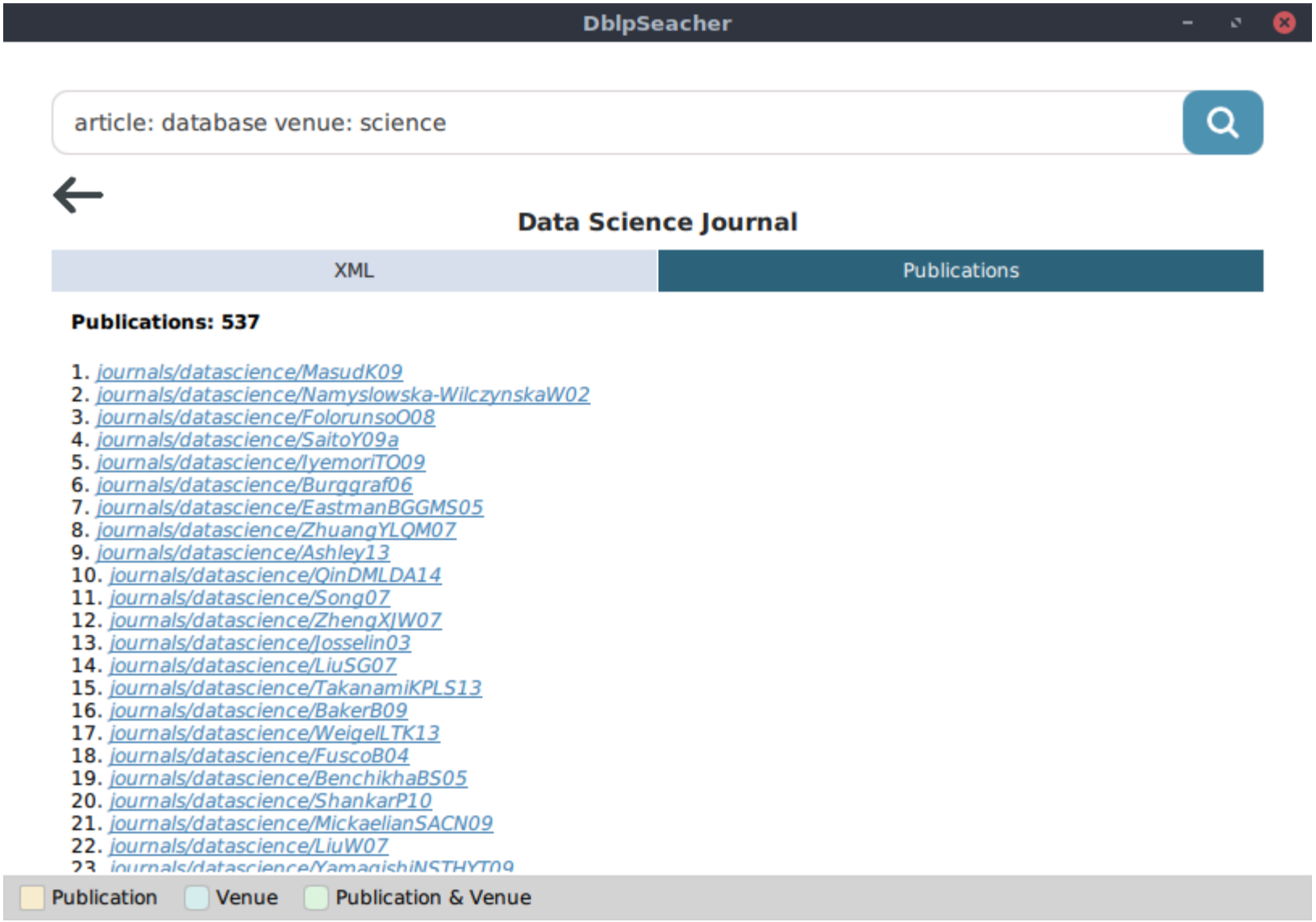
Applicazione: ricerca



Applicazione: dettagli elemento



Applicazione: lista pubblicazioni



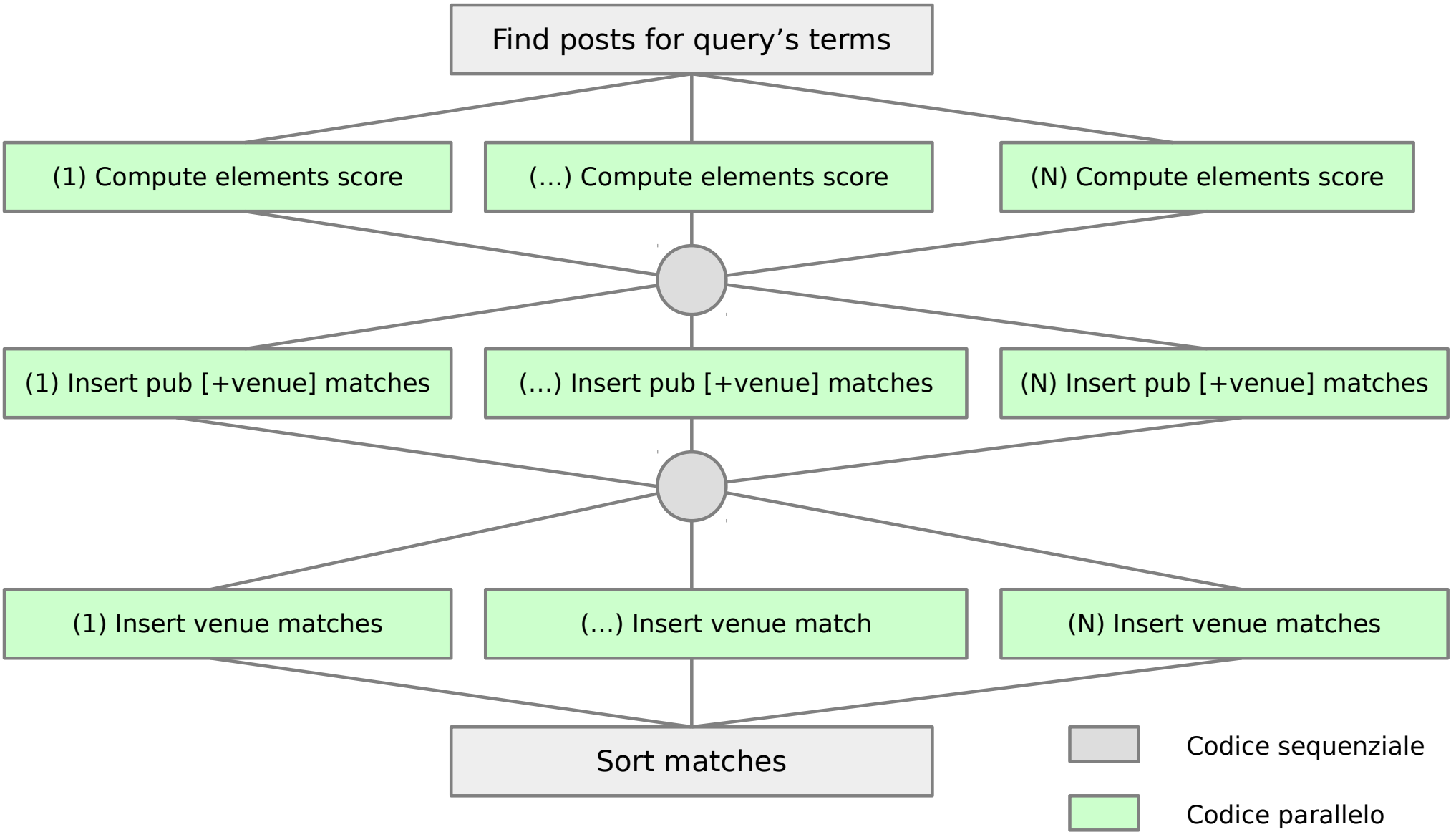
Ottimizzazioni

Ottimizzazioni: multithreading

- Al fine di incrementare le prestazioni di alcune parti di codice si è scelto di adottare la libreria OpenMP e di sfruttarne i paradigmi messi a disposizione.
- In particolare è stata parallelizzata la risoluzione della query poiché si è osservato essere una porzione di codice prevalentemente CPU-bound e ben disposta ad essere parallelizzata.
- Non è stata invece parallelizzata l'indicizzazione poiché ciò si è rivelato, al seguito di alcuni test effettuati, essere controproducente. Questo perché l'indicizzazione è strettamente vincolata alla lettura del file XML al punto da rendere un eventuale codice parallelo responsabile di accedere in modo sicuro al file XML e agli indici meno performante del corrispettivo codice sequenziale (a meno di utilizzare "trucchetti" come caricare in memoria tutto il file XML prima di effettuarne il parsing).

- Di seguito si propone uno schema di parallelizzazione dell'algoritmo di risoluzione precedentemente descritto.

Ottimizzazioni: risoluzione query



Fine

Riferimenti

- DBLP: https://dblp.org/
- Dump XML di DBLP: https://dblp.uni-trier.de/xml/
- Codice sorgente: https://github.com/Docheinstein/dblp-searcher
- Qt5: https://doc.qt.io/
- OpenMP: https://www.openmp.org/