



UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in  
Informatica

ELABORATO FINALE

DANDELION

Supervisore

Alberto Montresor

Laureando

Edoardo Lenzi

Anno accademico 2018

# Ringraziamenti

*Prima di addentrarmi nella descrizione del progetto di tesi voglio ricordare coloro che mi sono stati vicini.*

*Ringrazio la mia famiglia per essermi sempre stata accanto ed avermi supportato (e sopportato) in tutti questi anni, permettendomi di studiare fino a laurearmi; ringrazio in particolare mio fratello Massimo e mia sorella Maria Vittoria che più di chiunque altro mi hanno aiutato nello studio.*

*Ringrazio i miei zii Chiara e Christian, persone uniche e speciali che da sempre mi sono vicine e mi sostengono.*

*Ringrazio i miei amici, in particolare Michele, dispensatore dei migliori consigli e delle peggiori critiche, senza il quale non avrei nemmeno scelto questa via.*

*Ringrazio i miei colleghi di lavoro che mi hanno insegnato moltissimo, facendomi appassionare, di giorno in giorno, sempre più al mio lavoro.*

*Devo, infine, un ringraziamento particolare ad Alberto Montresor, Davide Setti, Alessio Guerrieri, Ugo Scaiella e Marco Fossati che mi hanno seguito ed aiutato moltissimo in questo percorso di tesi.*

*A tutti loro va la mia riconoscenza ed i miei più sentiti ringraziamenti.*

# Contents

<b>Sommario</b>	<b>2</b>
<b>Introduzione</b>	<b>2</b>
<b>1 Introduzione</b>	<b>2</b>
1.1 Dandelion . . . . .	2
1.2 GitHub e Travis . . . . .	3
1.3 Docker . . . . .	4
1.4 Prometheus e Grafana . . . . .	4
1.5 Wikidata e StrepHit . . . . .	4
1.5.1 SPARQL . . . . .	5
<b>Client C</b>	<b>5</b>
<b>2 Client C</b>	<b>5</b>
2.1 Dandelion . . . . .	5
2.2 Struttura Generale . . . . .	5
2.3 Il Client . . . . .	5
2.4 Testing . . . . .	6
2.5 Documentazione . . . . .	6
2.6 Nuget . . . . .	6
<b>3 Calcolo della Relatedness</b>	<b>7</b>
3.1 Implementazione Iniziale . . . . .	7
3.1.1 Il Dump . . . . .	7
3.1.2 Il Codice Nativo . . . . .	8
3.1.3 Osservazioni . . . . .	11
3.2 Implementazione con Hash Map . . . . .	12
3.2.1 Codice . . . . .	12
<b>4 Conclusioni</b>	<b>14</b>
<b>Bibliografia</b>	<b>14</b>
<b>A Titolo primo allegato</b>	<b>15</b>
A.1 Titolo . . . . .	15
A.1.1 Sottotitolo . . . . .	15
<b>B Titolo secondo allegato</b>	<b>16</b>
B.1 Titolo . . . . .	16
B.1.1 Sottotitolo . . . . .	16

# Sommario

Il progetto di tesi è il frutto di un tirocinio presso SpazioDati ed una successiva collaborazione al progetto Wikidata StrepHit. La tesi consiste principalmente in tre parti separate, ognuna delle quali è legata al servizio online Dandelion (creato da SpazioDati), che verrà brevemente introdotto nelle prossime pagine.

Il primo capitolo ha lo scopo di introdurre, al lettore, gli applicativi, le tecnologie ed i servizi usati durante il progetto di tesi, descrivendoli sommariamente; nei tre capitoli successivi, invece, si entrerà maggiormente nel dettaglio del progetto descrivendo i tre ambiti fondamentali su cui si è focalizzata la tesi.

La prima parte del progetto consiste nell'implementazione di un client in C#, finalizzato allo scopo di creare una libreria di metodi “plug&play” per chiamare automaticamente le RESTful-API del servizio Dandelion.

Durante lo sviluppo il codice è stato regolarmente caricato online, sulla piattaforma GitHub, per agevolare la periodica revisione del codice effettuata dal team di SpazioDati; per rendere lo sviluppo più efficiente si è cercato di seguire i principi del TDD (Test Driven Development), affidandosi al servizio di testing automatico Travis per validare automaticamente ogni pull-request su GitHub.

Una volta completata la libreria e l'annessa documentazione è stata compilata la dll e resa pubblica su Nuget; pertanto la libreria è ora facilmente integrabile in qualsiasi progetto C#.

La seconda parte del progetto riguarda l'analisi e il test di un algoritmo presente nel backend di Dandelion al fine di ottimizzarlo. Il task nasce dalla necessità pratica di ridurre la memoria necessaria alle macchine di produzione di SpazioDati per eseguire l'algoritmo senza subire cali prestazionali. Partendo dall'implementazione attualmente in uso si sono studiate delle implementazioni alternative per massimizzare la velocità dell'algoritmo e minimizzare la memoria occupata dalle strutture dati di appoggio dell'algoritmo.

L'ultima parte gravita attorno al progetto StrepHit di Wikidata, un progetto nato un anno fa in FBK ed approvato dalla community di Wikidata. Il progetto nasce per arricchire i database di Wikidata con riferimenti a fonti esterne (tendenzialmente altri siti web) in modo da dare all'utente, fruitore dell'enciclopedia, informazioni sempre più corrette, validate anche da un sito esterno e non più solo dalla community di Wikipedia/Wikidata.

Il progetto è fortemente legato a Dandelion perchè nelle logiche interne dello script di StrepHit si fa largo uso dei servizi offerti da Dandelion per l'analisi testuale delle pagine dei siti esterni. L'ultima attività consiste nell'implementazione di uno script in python (versione 2.7) utile ad arricchire un dataset di quickstatements aggiungendo riferimenti a proprietà di Wikidata. Per fare ciò è stato necessario studiare il linguaggio python ed il funzionamento generale di Wikidata per poi realizzare uno script capace di interfacciarsi con l'endpoint SPARQL di Wikidata.

## 1 Introduzione

### 1.1 Dandelion

Dandelion è un servizio online fornito da SpazioDati che mette a disposizione dell'utente servizi di analisi semantica testuale; grazie ad esso è possibile, dato un testo, estrarne le entità semantiche principali (*Entity Extraction*), trovare la lingua in cui è stato scritto (*Language Detection*), classificarlo secondo modelli definiti dall'utente stesso (*Text Classification*) e analizzarne la semantica per capire i sentimenti che l'autore ci vuole trasmettere (*Sentiment Analysis*).

Dandelion ha anche altre due RESTful-API che permettono di confrontare due testi generando un indice di similitudine fra i due (*Text Similarity*) e un motore di ricerca di entità di Wikipedia (*Wikisearch*), nel caso si voglia trovare il titolo di un contenuto senza conoscerlo a priori.

L'enciclopedia alla base di Dandelion è Wikipedia, anche se a volte fra i due si colloca come mediatore dbpedia, un progetto italiano per l'estrazione di informazioni semi-strutturate da Wikipedia.

Per poter usare gli end-point di Dandelion basta registrarsi sul portale dedicato e generare un token che andrà inserito come query parameter nelle chiamate https alle API; la documentazione delle API è disponibile all'indirizzo: <https://dandelion.eu/docs/>.

[TODO descrizione più dettagliata di Dandelion]

## 1.2 GitHub e Travis

Per il versioning del codice si è scelto di usare il software git, appoggiandosi alla piattaforma GitHub per creare repository pubbliche.

Le repository riguardanti il progetto di tesi sono:

- Repository contenente il testo della tesi scritto in  $\text{\LaTeX}$  (<https://github.com/EdoardoLenzi9/Dandelion.LaTeX>)
- Repository contenente lo script in Python per il progetto StrepHit (<https://github.com/EdoardoLenzi9/Wikipedia.StrepHit>)
- Repository contenente il codice Java con le varie implementazioni dell'algoritmo per il calcolo della relatedness (la seconda parte del progetto) (<https://github.com/EdoardoLenzi9/Dandelion.Relatedness>)
- Repository contenente il codice sorgente del client C# (<https://github.com/EdoardoLenzi9/SpazioDati.Dandelion-eu>)

Per quanto riguarda il testing automatico è stata scelta la piattaforma Travis che permette di agganciare una repository GitHub su cui eseguire automaticamente i test ad ogni commit.

Per fare ciò è necessario inserire nella repository un file di configurazione chiamato `.travis.yml` con i comandi necessari ad eseguire i test.

Per esempio nel caso del client C# il file yml contiene le seguenti istruzioni:

---

```
1 language: csharp
2 dist: trusty
3 mono: none
4 dotnet: 2.0.3
5
6 install:
7 - dotnet restore
8
9 script:
10 - dotnet build
11 - dotnet test SpazioDati.Dandelion.Test/SpazioDati.Dandelion.Test.csproj
```

---

Mentre per lo script in python la sintassi è la seguente:

---

```
1 language: python
2 python:
3 - "2.7"
4 # command to install dependencies
5 install:
```

---

```
6     - pip install -r requirements.txt
7     # command to run tests
8     script:
9     - python test.py
```

---

Come si nota dalle righe sopra, basta specificare il linguaggio, la versione ed i comandi (Travis esegue i test in ambiente Linux, quindi i comandi devono essere eseguibili in una bash-shell Linux) per installare e lanciare i test.

## 1.3 Docker

Per la seconda parte del progetto è stato usato Docker per creare un ambiente "chiuso", ad hoc, dove poter testare al meglio gli algoritmi senza doversi preoccupare di fattori esterni che potrebbero falsare il risultato del test.

Docker permette di creare un'immagine a partire da un file di testo chiamato "Dockerfile" il quale contiene tutte le informazioni per eseguire la build dell'immagine; in pratica il Docker file è una sorta di descrittore dello stato di una macchina virtuale che genera una macchina virtuale persistente chiamata immagine. L'immagine può essere istanziata, divenendo un container il quale però non è più persistente. Una delle caratteristiche più utili di Docker è la semplicità con cui è possibile interfacciare container diversi condividendo aree di memoria sul disco e porte TCP.

Per esempio l'idea di base per monitorare le prestazioni dell'algoritmo e l'allocazione di memoria nel Docker è quella di eseguire il file jar compilato, dentro un container, con un particolare javaagent chiamato jmx-exporter; l'exporter si occupa di monitorare l'applicativo e di esportare in tempo reale delle metrics che vengono esposte in un server locale su una porta a scelta. Parallelamente al primo container vengono istanziate altre due immagini, scaricate da hub.docker, chiamate prom/prometheus e grafana/grafana le quali permettono di filtrare le metrics generate dall'exporter e di visualizzarle direttamente su dei grafici.

Per il setup dell'ambiente è stato creato un'immagine a partire dal seguente Dockerfile:

---

```
1 FROM openjdk:latest
2 COPY . /usr/src/dandelion
3 WORKDIR /usr/src/dandelion
4 CMD ["java", "-Xmx6144m",
      "-javaagent:./lib/jmx_prometheus_javaagent-0.3.1.jar=8080:./lib/configs.yaml",
      "-jar", "/usr/src/dandelion/dist/Dandelion.jar"]
```

---

Sostanzialmente l'immagine viene edificata sopra un'altra immagine chiamata openjdk, scaricata da hub.docker, contenente un'implementazione open source di Java SE (Java Platform, Standard Edition) a partire dalla versione 7. La direttiva COPY serve per copiare, al momento della build dell'immagine, tutto il contenuto della cartella contenente dentro l'ambiente virtuale alla path specificata (/usr/src/dandelion); mentre WORKDIR specifica la path della *working directory* da cui partirà il container docker. Infine la direttiva CMD lancia il .jar generato dalla compilazione con il tool ant e setta come javaagent jmx-exporter (l'opzione -Xmx6144m serve solamente a specificare la massima allocazione di memoria heap, fino ad un massimo di 6GB).

## 1.4 Prometheus e Grafana

[TODO]

## 1.5 Wikidata e StrepHit

[TODO]

### 1.5.1 SPARQL

[TODO]

## 2 Client C

### 2.1 Dandelion

### 2.2 Struttura Generale

Per l'implementazione si è cercato di seguire le best practices, dettate dalle linee guida Microsoft, per la stesura del codice. Sono stati adottati, come pattern di programmazione, dependency injection, MVC (per quanto possibile) e TDD, appoggiandosi a librerie esterne Nuget quali Newtonsoft.Json e SimpleInjector. I metodi sono asincroni e le property thread-safe.

La solution è stata partizionata in vari progetti; il progetto Business contiene la business-logic (comprendente servizi, metodi estensione, l'implementazione del client e un wrapper del Container di SimpleInjector) e utilizza i modelli di Domain.

In Test sono contenute le classi di testing e le fixture XUnit, mentre in Documentation sono presenti i file generati dal tool Wyam per la documentazione.

### 2.3 Il Client

Dalla documentazione ufficiale delle API di Dandelion si evince che ogni end-point richiede uno o più testi ed una serie di parametri alcuni obbligatori ed altri opzionali; pertanto sono stati creati dei modelli, coerenti con tali definizioni, che, passati in argomento a dei servizi, ritornano i DTO delle risposte degli end-point di Dandelion.

Ogni servizio controlla che i parametri inseriti dall'utente rispettino i constraints definiti nella documentazione e costruisce, tramite i metodi *ContentBuilder* e *UriBuilder*, un dizionario di parametri e l'URI che identifica l'end-point.

Infine il servizio chiama il metodo generico *CallApiAsync*, passandogli il content, la URI ed il metodo HTTP; nel metodo di chiamata si è scelto di gestire separatamente i metodi HTTP GET e DELETE dato che essi non permettono il passaggio parametri nel body della chiamata HTTP. Pertanto è stato necessario inviare il content come query parameter facendone il percent-encoding.

Dandelion ammette sia chiamate in GET che in POST ai servizi, tuttavia nel caso del metodo GET non è garantito il corretto funzionamento del servizio per testi che superano i 2000 caratteri circa; pertanto nel caso in cui l'utente non specifichi il metodo HTTP, di default verrà usato il metodo POST.

Se la chiamata è andata a buon fine il JSON di ritorno viene mappato in un DTO specifico, a seconda del servizio scelto, e restituito direttamente all'utente.

---

```
1 public Task<T> CallApiAsync<T>(string uri, List<KeyValuePair<string, string>> content,
2   HttpMethod method = null)
3 {
4     var result = new HttpResponseMessage();
5     if (method == null)
6     {
7         method = HttpMethod.Post;
8     }
9     if (_client.BaseAddress == null)
10    {
11        _client.BaseAddress = new Uri(Localizations.BaseUrl);
12    }
13
14    return Task.Run(async () =>
```

```

15 {
16     if (method == HttpMethod.Get)
17     {
18         string query;
19         using (var encodedContent = new FormUrlEncodedContent(content))
20         {
21             query = encodedContent.ReadAsStringAsync().Result;
22         }
23         result = await _client.GetAsync($"{uri}/{?query}");
24     }
25     else if (method == HttpMethod.Delete)
26     {
27         string query;
28         using (var encodedContent = new FormUrlEncodedContent(content))
29         {
30             query = encodedContent.ReadAsStringAsync().Result;
31         }
32         result = await _client.DeleteAsync($"{uri}/{?query}");
33     }
34     else
35     {
36         var httpContent = new HttpRequestMessage(method, uri);
37         if (content != null)
38         {
39             httpContent.Content = new FormUrlEncodedContent(content.ToArray());
40         }
41         result = await _client.SendAsync(httpContent);
42     }
43     string resultContent = await result.Content.ReadAsStringAsync();
44     if (result.StatusCode == System.Net.HttpStatusCode.RequestUriTooLong)
45     {
46         throw new ArgumentException(ErrorMessages.UriTooLong);
47     }
48     if (!result.IsSuccessStatusCode)
49     {
50         throw new Exception(resultContent);
51     }
52     return JsonConvert.DeserializeObject<T>(resultContent);
53 });
54 }

```

---

## 2.4 Testing

La maggiorparte dei servizi è stata testata tramite il tool XUnit; principalmente sono stati eseguiti test di validazione, data la natura della libreria, appoggiandosi ad una fixture comune a tutti i test per l'inizializzazione dei servizi.

è stato usato fin da subito il servizio online di testing automatico Travis, tramite il quale è stato possibile validare ogni rilascio facendo partire automaticamente i test con l'evento di push di Git.

## 2.5 Documentazione

Infine le classi e i metodi più rilevanti sono stati commentati tramite i tag XML specificati nella documentazione Microsoft e la documentazione in formato HTML è stata generata automaticamente tramite il tool Wyam.

## 2.6 Nuget

La dll generata dalla compilazione della libreria è stata infine documentata sotto il profilo delle dipendenze, generando il file .nuspec, ed inclusa nel file .nupkg tramite l'apposito tool fornito da Nuget. Il



tutto è stato caricato sul portale online Nuget ed è ora possibile includerlo in un progetto tramite il comando cli:

---

```
1 $ dotnet add package SpazioDati.Dandelion
```

---

## 3 Calcolo della Relatedness

La seconda parte del progetto riguarda l'analisi di due metodi critici (*readDump()* e *rel(int a, int b)*) che servono a calcolare, dati gli identificativi di due entità semantiche, il loro valore di correlazione (relatedness).

Per fare ciò si ha a disposizione un dump dove sono salvati tutti i valori di correlazione (sopra una certa soglia minima) per ogni coppia di entità.

Il metodo *readDump()* serve a caricare in memoria i valori del dump sottoforma di “matrice” mentre il metodo *rel(int a, int b)* serve a fare una ricerca nella struttura dati per poi ritornare il valore di correlazione fra le entità con identificativi *a* e *b*.

Questa scelta implementativa, attualmente adottata, è sicuramente molto efficiente in termini di prestazioni ma sicuramente molto onerosa in termini di memoria; pertanto il fine dell'analisi sarebbe quello di studiare/testare implementazioni alternative per ottimizzare le prestazioni e/o diminuire lo spreco di memoria.

### 3.1 Implementazione Iniziale

#### 3.1.1 Il Dump

---

```
1 53676192
2 4922289
3 0.01
4 2
5 1.0
6 53676158 7 null
7 53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH
8 53676016 13 RDsFRUkyASRLf3T
9 53675811 16 null
10 ...
```

---

Leggendo le prime righe di un dump si nota che le prime cinque righe sono valori di inizializzazione mentre dalla sesta riga in poi troviamo le righe della matrice. Il primo numero intero, denominato *max\_id*, indica il limite massimo che gli id delle entità possono assumere; dato che gli id delle entità non sono necessariamente sequenziali (fra due di essi potrei avere dei “buchi”, degli intervalli in cui gli identificativi non sono associati ad alcuna entità) esiste una funzione *map* che mappa (“compatta”) gli id delle entità su altri id univoci e sequenziali.

Il secondo intero, denominato *nodesSize* è il valore massimo che la funzione *map* può assumere (limita il codominio della funzione).

$$\begin{aligned} \text{map}() : \mathbb{N} &\rightarrow \mathbb{N} \\ [0, \text{max\_id}] &\rightarrow [0, \text{nodeSize}], \text{max\_id} \geq \text{nodeSize} \end{aligned} \tag{3.1}$$

Continuando a leggere il dump troviamo la relatedness minima considerata *minRel* (sotto la quale i valori di correlazione non vengono salvati) e altri valori di configurazione quali *minIntersection* e *threshold* che tuttavia non ci interessano particolarmente.

La funzione *map* in un certo senso mappa “al contrario” gli id; infatti l'id massimo verrà mappato su 0, il penultimo id verrà mappato su 1 e così via. Nel caso del dump in questione abbiamo:

---

```

1 MaxId = 53676192
2 NodeSize = 4922289
3
4 ID          MAP(ID)  POSTINGLIST
5 53676158 ->    7      null
6 53676159 ->    6      FW/I...
7 53676164 ->    5      NnAw...
8 53676174 ->    4      null
9 53676176 ->    3      null
10 53676177 ->    2      null
11 53676180 ->    1      null
12 53676190 ->    0      null

```

Gli id presenti nel dump non sono ordinati ma le *postingList* ad essi associate lo sono.

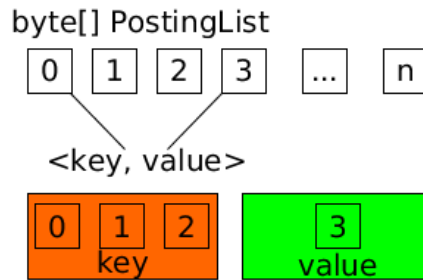
Dalla sesta riga del dump in poi inizia la definizione della matrice, ogni riga è composta da tre valori separati da uno spazio. Ad esempio se consideriamo la riga:

```

1 53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH

```

Il primo valore  $a = 53676164$  è l'id di un entità, il secondo valore  $b = 5 = \text{map}(a)$  è il risultato della funzione map, infine il terzo valore  $c = \text{NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH}$  è una stringa di dimensione variabile oppure *null*; quest'ultima stringa, denominata *postingList*, è la codifica in *Base64* di un array di byte che rappresenta un dizionario chiave valore.



In pratica una tupla  $\langle \text{chiave}, \text{valore} \rangle$  è codificata sull'array da una sequenza di quattro byte; quindi partizionando l'array in gruppi da 4 byte ottengo, per ogni gruppo, la chiave (è un id “mappato”) codificata sui primi tre byte ed il valore (la relatedness) definito sul quarto byte (il byte più a destra).

Questa codifica, pertanto, permette di gestire set di entità con id “mappato” massimo  $2^{3 \cdot 8} \simeq 1,7 \cdot 10^7$  ed una scala di valori di correlazione appartenenti all'intervallo  $[0, 255]$ .

Bisogna precisare che l'array di byte, denominato *postingList*, è ordinato per valore crescente delle chiavi e, dato che  $\text{rel}(a, b) = \text{rel}(b, a)$ , si è deciso di salvare una sola volta il valore di correlazione imponendo la relazione:  $a > b$ .

### Esempio

Nell'ipotesi in cui abbia una riga nel dump costituita dalla tupla  $(a, b, c)$  e volessi trovare  $\text{relatedness}(a, d)$  con  $\text{map}(a) > \text{map}(d)$ .

Devo cercare nella *postingList*  $c$ , un gruppo di quattro byte in cui i primi tre byte corrispondano a  $\text{map}(d)$  ed il quarto byte sarà il valore di correlazione voluto.

Ovviamente è possibile che  $c$  sia *null* (l'entità con id  $a$  non ha nessuna correlazione con entità i cui id “mappati” siano inferiori a  $b$ ) oppure che nella *posting* non sia presente nessuna chiave corrispondente a  $\text{map}(d)$ , in questi casi  $\text{relatedness}(a, d) = 0$ .

### 3.1.2 Il Codice Nativo

```

1 // STRUTTURA DATI
2 public class RelatednessMatrix {

```

```

3     public int[] map;
4     public byte[][] matrix;
5     public float configuredMinRel;
6     public int configuredMinIntersection;
7     public float threshold;
8     public static int EL_SIZE = 4;
9 }
10
11 // CODICE UTILIZZO
12 public float rel(int a, int b){
13     if(a==b) return 1f;
14
15     int nodeA = data.map[a];
16     int nodeB = data.map[b];
17     if (nodeA < 0 || nodeB < 0) return 0f;
18
19     int key = a>b ? nodeB : nodeA;
20     byte[] array = a>b ? data.matrix[nodeA] : data.matrix[nodeB];
21
22     if (array == null) return 0f;
23
24     int start = 0;
25     int end = array.length - RelatednessMatrix.EL_SIZE;
26     int pos = -1;
27     while(pos == -1 && start <= end)
28     {
29         int idx = ((start+end)/RelatednessMatrix.EL_SIZE)/2;
30         int idx_value = ((array[idx*RelatednessMatrix.EL_SIZE] & 0xFF) << 16)
31             + ((array[idx*RelatednessMatrix.EL_SIZE+1] & 0xFF) << 8)
32             + ( array[idx*RelatednessMatrix.EL_SIZE+2] & 0xFF);
33
34         if(idx_value == key) pos = idx;
35         else{
36             if(key > idx_value)
37                 start = (idx + 1) * RelatednessMatrix.EL_SIZE;
38             else
39                 end = (idx - 1) * RelatednessMatrix.EL_SIZE;
40         }
41     }
42
43     if(pos == -1) return 0f;
44     else
45     {
46         byte by = array[pos * RelatednessMatrix.EL_SIZE + 3];
47         int byint = by + 128;
48         float byteRel = byint / 255f;
49         return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
50     }
51 }
52
53 // CODICE LETTURA
54 public String END_OF_FILE = "" + '\0';
55
56 public RelatednessMatrix readDump() throws Exception {
57     URL url = getClass().getResource("dump.txt");
58     File file = new File(url.getPath());
59     BufferedReader fbr = new BufferedReader(new FileReader(file));
60     String line = new String();
61
62     try {

```

```

63     int max_id = Integer.parseInt(fbr.readLine().trim()); // Trims all leading and
        trailing whitespace from this string
64     int nodesSize = Integer.parseInt(fbr.readLine().trim());
65     float minRel = Float.parseFloat(fbr.readLine().trim());
66     int minIntersection = Integer.parseInt(fbr.readLine().trim());
67     float threshold = Float.parseFloat(fbr.readLine().trim());
68
69     RelatednessMatrix data = new RelatednessMatrix();
70     data.map = new int[max_id + 1];
71     data.matrix = new byte[nodesSize] [];
72     data.configuredMinRel = minRel;
73     data.configuredMinIntersection = minIntersection;
74     data.threshold = threshold;
75
76     int idx = 0;
77
78     while ((line = fbr.readLine()) != null) {
79         if (line.trim().equals(END_OF_FILE.trim())) {
80             break;
81         }
82         System.out.println(line);
83         String[] splittedLine = line.split("\\s+"); // split(" ")
84         if (splittedLine.length != 3) {
85             throw new Exception("Wrong format relatedness file for the line: " + line);
86         }
87
88         int wid = Integer.parseInt(splittedLine[0]);
89         int node = Integer.parseInt(splittedLine[1]);
90         data.map[wid] = node;
91
92         if (splittedLine[2].equals("null")) {
93             data.matrix[node] = null;
94         } else {
95             byte[] a = Base64.getDecoder().decode(splittedLine[2].toString());
96             data.matrix[node] = Base64.getDecoder().decode(splittedLine[2].toString());
97         }
98         idx++;
99     }
100
101     if (idx != nodesSize) {
102         throw new Exception("Wrong format relatedness file the number of line do not
            match with size of matrix");
103     }
104
105     return data;
106 } catch (Exception e) {
107     System.out.println(e.getMessage());
108     return null;
109 }
110 }

```

---

Il metodo *readDump()* legge riga per riga il dump e restituisce un'istanza di *RelatednessMatrix* valorizzata. La funzione *map* viene salvata in un array di interi (*data.map*) mentre la matrice con le correlazioni viene salvata in un array bidimensionale di byte (*data.matrix*).

Il metodo *rel(int a, int b)* invece calcola la funzione *map* sugli id *a* e *b* dopodichè fa una binary search sulla *postingList* ottenuta da *matrix[map(a)]* (assumendo *map(a) > map(b)*). Infine se la ricerca binaria ha successo il valore di *relatedness* ottenuto va riscalato su una scala di valori [0, 255], convertito in float e riscalato nuovamente in base al valore di *relatedness* minima considerata (per escludere i valori al di sotto della *relatedness* minima, che non verranno mai usati).

---

```

1 byte by = array[pos * RelatednessMatrix.EL_SIZE + 3]; //considero il quarto byte
2 int byint = by + 128; //scalo di 128 valori per ottenere una relatedness in [0, 255]
3
4 //cast in float, escludendo i valori sotto la relatedness minima (configuredMinRel)
5 float byteRel = byint / 255f;
6 return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);

```

---

### 3.1.3 Osservazioni

Considerato che il metodo `readDump()` viene chiamato una sola volta per allocare in memoria la struttura dati nel server, non ci sono particolari vincoli temporali per la sua esecuzione; per quanto riguarda il calcolo della *relatedness* invece l'algoritmo dev'essere il più performante possibile.

Il codice nativo per il calcolo della *relatedness* costa  $O(\lg_2(n))$  (con  $n = \text{nodeSize}$ ) per la ricerca binaria e  $O(1)$  per quanto riguarda le istruzioni precedenti e successive a quest'ultima.

Tolto un piccolo refactor del codice che portebbe ad evitare tre moltiplicazione per ciclo nella binary search (non è necessario dividere e poi moltiplicare per `RelatednessMatrix.EL_SIZE`), non sembra si possono migliorare di molto le prestazioni dell'algoritmo senza cambiare struttura dati.

Una via percorribile per non "pagare"  $O(\lg_2(n))$  per la binary search potrebbe essere quella di allocare direttamente in memoria la matrice completa (in questo caso avrei un algoritmo di tempo costante  $O(1)$ ). Questa soluzione però andrebbe a peggiorare drasticamente lo spreco di memoria. Infatti nell'implementazione attuale la matrice ha dimensione massima  $\text{nodeSize} \otimes \text{nodeSize}$  ma le righe hanno dimensione variabile e non occupano quindi sempre  $(\text{nodeSize} - 1) \cdot 4 \text{ Byte}$ .

### Stima della Memoria Allocata

Se consideriamo le prime righe del dump in questione possiamo stimare che mediamente il 57% degli id hanno `postingList` vuota (null) ed il restante 43% ha un numero medio di *relatedness* per id pari allo 0,000101% di `nodeSize`.

L'implementazione attuale alloca in memoria per l'array di interi map circa:

$$\text{nodeSize} \cdot 4 \text{ Byte} = 4^2 \cdot 10^6 \text{ Byte} = 16 \text{ MB} \quad (3.2)$$

(arrotondando e senza tenere conto di fattori secondari come i 32 byte occupati da overhead e puntatore).

Per la matrice invece possiamo assumere che, su una macchina a 64 bit, avrò il 57% di righe a null ( $0.57 \cdot 4 \cdot 10^6 \cdot 8 \text{ Byte} = 18 \text{ MB}$ ) ed il restante 43% di righe con mediamente 404 valori di *relatedness* codificati su 4 Byte ( $0.43 \cdot 4 \cdot 10^6 \cdot 404 \cdot 4 \text{ Byte} = 2.8 \text{ GB}$ ).

Quindi in totale potremmo stimare che solo questo dump occupa quasi 3 GB (notare che esiste un dump per ogni lingua supportata da Dandelion e che questo è uno dei dump più piccoli).

### Stima della Matrice Completa

Se allocassimo in memoria una matrice di Byte completa, di dimensioni  $\text{nodeSize} \otimes \text{nodeSize}$ , essa occuperebbe  $(4 \cdot 10^6)^2 = 1.6 \cdot 10^{13} \text{ Byte} = 16 \text{ TB}$ . Pur considerando che esistono strutture dati particolarmente ottimizzate per gestire array di grandi dimensioni con pochi valori al loro interno (teniamo presente che più della metà dei valori nella matrice non saranno valorizzati), come `SparseArray` e `SuperArrayList`, bisogna tener presente che spesso tali strutture dati si basano su liste; in questo caso però difficilmente si otterrebbero prestazioni migliori di quelle dell'implementazione attuale dato che il tempo di lookup sarebbe sempre  $O(\lg_2(\text{nodeSize}))$ .

In conclusione, a meno di uno spreco di memoria ulteriore, difficilmente otterremo buoni risultati mantenendo come struttura dati di riferimento la matrice.

## Stima del problema inverso

Una via praticabile potrebbe essere quella di vedere il problema al contrario, considerando che esistono solo 256 valori possibili di relatedness, per minimizzare lo spreco di memoria potrei pensare di allocare 256 array in cui storicizzo tutte le coppie di id che hanno la stessa relatedness. In questo modo eliminerei la ridondanza costituita da valori uguali di relatedness salvati nel dump migliaia di volte.

Ci si accorge subito che anche questa via non è praticabile dato che per risparmiare 1 Byte devo storicizzare 8 Byte per le coppie di id di tipo intero. Andrei ad occupare  $0.43 \cdot 4 \cdot 10^6 \cdot 404 \cdot 2 \cdot 4 = 5.6GB$ , il doppio rispetto all'implementazione nativa, senza contare che le performance peggiorerebbero.

## Stima implementazione di un grafo

Si potrebbe pensare di implementare un grafo orientato, salvando per ogni nodo l'id e la lista di archi uscenti e su ogni arco la relatedness fra nodo di partenza e nodo di arrivo. Dovrei quindi usare una struttura dati del tipo:

---

```
1 public class Node{
2     public int Id;
3     public List<Arrow> Arrows;
4 }
5
6 public class Arrow{
7     public Node Node;
8     public byte Relatedness;
9 }
```

---

Tuttavia in questo modo sprecherei memoria perchè il puntatore al nodo successivo (ipotizzando di lavorare su una macchina a 64 bit) peserebbe più dell'identificativo del nodo stesso. Al che potrei sostituire il puntatore con un intero ma anche in questo caso avrei il dizionario  $\langle Id, Relatedness \rangle$  che pesa 5 Byte mentre nella postingList pesa 4 Byte perchè chiave e valore sono accorpati. Infine se seguissi la stessa politica di codifica del dizionario della postingList di fatto sarei tornato all'implementazione iniziale senza trarre alcun giovamento, se non lo svantaggio aggiuntivo di non poter accedere in  $O(1)$  ad un nodo arbitrario.

## 3.2 Implementazione con Hash Map

Sembra che l'unica via praticabile per mantenere tutti i dati in memoria, diminuendo la memoria occupata e massimizzando le prestazioni sia una funzione di Hash Map, che mappi gli id concatenati sul valore della relatedness; se la funzione fosse ben ottimizzata otterrei in  $O(1)$  la relatedness (guadagno prestazionale) e potenzialmente potrebbe occupare meno memoria della matrice iniziale.

Purtroppo è molto difficile stimare a priori lo spazio occupato da un Hash Map, il modo più rapido è confrontare con dei benchmark l'implementazione nativa contro quella basata su Hash Map.

### 3.2.1 Codice

---

```
1 public float Relatedness(int a, int b) throws Exception {
2     if (a == b) {
3         return 1f;
4     }
5
6     int nodeA = data.map[a];
7     int nodeB = data.map[b];
8
9     if (nodeA < 0 || nodeB < 0) {
10        return 0f;
11    }
12
13    String key = (nodeA > nodeB) ? (nodeA + "." + nodeB) : (nodeB + "." + nodeA);
14    try {
```

```

15     byte value = (byte) data.hm.get(key);
16
17     if (value == 0) {
18         return 0f;
19     } else {
20         int byint = value + 128;
21         float byteRel = byint / 255f;
22         return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
23     }
24 } catch (Exception e) {
25     return 0f;
26 }
27 }
28
29 public RelatednessHashMap ReadDump(String dump) throws Exception {
30     URL url = getClass().getResource(dump);
31     File file = new File(url.getPath());
32     BufferedReader fbr = new BufferedReader(new FileReader(file));
33     String line = new String();
34
35     try {
36         RelatednessHashMap data = new RelatednessHashMap();
37         int max_id = Integer.parseInt(fbr.readLine().trim());
38         int nodesSize = Integer.parseInt(fbr.readLine().trim());
39         data.configuredMinRel = Float.parseFloat(fbr.readLine().trim());
40         data.configuredMinIntersection = Integer.parseInt(fbr.readLine().trim());
41         data.threshold = Float.parseFloat(fbr.readLine().trim());
42
43         data.map = new int[max_id + 1];
44         data.hm = new HashMap();
45
46         String END_OF_FILE = "" + '\0';
47
48         while ((line = fbr.readLine()) != null) {
49             if (line.trim().equals(END_OF_FILE.trim())) {
50                 break;
51             }
52
53             String[] splittedLine = line.split("\\s+");
54             if (splittedLine.length != 3) {
55                 throw new Exception("Wrong format relatedness file for the line: " + line);
56             }
57
58             int wid = Integer.parseInt(splittedLine[0]);
59             int node = Integer.parseInt(splittedLine[1]);
60             data.map[wid] = node;
61
62             if (!splittedLine[2].equals("null")) {
63                 byte[] postingList = Base64.getDecoder().decode(splittedLine[2].toString());
64
65                 int idx_value = ((postingList[0] & 0xFF) << 16)
66                     + ((postingList[1] & 0xFF) << 8)
67                     + ((postingList[2] & 0xFF) << 0);
68
69                 String key = idx_value + "." + node;
70                 data.hm.put(key, postingList[3]);
71             }
72         }
73
74         return data;
75     } catch (Exception e) {

```

```
76     System.out.println(e.getMessage());
77     return null;
78 }
79 }
```

---

## 4 Conclusioni

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

Al termine dell'esperienza si è concluso che ...



# Allegato A    Titolo primo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

## A.1    Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

### A.1.1    Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

# Allegato B      Titolo secondo allegato

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

## B.1      Titolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.

### B.1.1      Sottotitolo

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec sed nunc orci. Aliquam nec nisl vitae sapien pulvinar dictum quis non urna. Suspendisse at dui a erat aliquam vestibulum. Quisque ultrices pellentesque pellentesque. Pellentesque egestas quam sed blandit tempus. Sed congue nec risus posuere euismod. Maecenas ut lacus id mauris sagittis egestas a eu dui. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Pellentesque at ultrices tellus. Ut eu purus eget sem iaculis ultricies sed non lorem. Curabitur gravida dui eget ex vestibulum venenatis. Phasellus gravida tellus velit, non eleifend justo lobortis eget.