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

Contents

Sommario **2**

1 Client C# **2**
1.1 Dandelion . . . 2
1.2 Struttura Generale . . . 2
1.3 Il Client . . . 2
1.4 Testing . . . 4
1.5 Documentazione . . . 4
1.6 Nuget . . . 4

2 Calcolo della Relatedness **4**
2.1 Implementazione Iniziale . . . 5
2.1.1 Il Dump . . . 5
2.1.2 Il Codice Nativo . . . 6
2.1.3 Osservazioni . . . 8
2.2 Implementazione con Hash Map . . . 10

Bibliografia **10**

A Titolo primo allegato **11**
A.1 Titolo . . . 11
A.1.1 Sottotitolo . . . 11

B Titolo secondo allegato **12**
B.1 Titolo . . . 12
B.1.1 Sottotitolo . . . 12

Sommario

Il progetto di tesi consiste in una prima parte riguardante l'implementazione di un client in C#, finalizzato allo scopo di includere in una libreria metodi plug&play per chiamare automaticamente le RESTful-API del servizio online Dandelion.

Durante lo sviluppo il codice é stato regolarmente caricato su GitHub per una periodica revisione da parte del team di SpazioDati; per rendere lo sviluppo piú efficiente si é cercato di seguire i principi del TDD, affidandosi al servizio online Travian per validare automaticamente ogni rilascio.

Una volta completata la libreria, é stata stilata la documentazione (esportata in html tramite il tool Wyam) e caricata la dll su Nuget; pertanto la libreria é ora facilmente integrabile in qualsiasi progetto C#.

La seconda parte del progetto di tesi riguarda sempre il servizio Dandelion di SpazioDati ma, questa volta, una parte del codice back-end. Partendo dall'implementazione attualmente in uso si sono studiate delle implementazioni alternative per massimizzare la velocitá dell'algoritmo e minimizzare la memoria occupata dalle strutture dati di appoggio dell'algoritmo.

...

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

1 Client C#

1.1 Dandelion

Dandelion é un servizio online fornito da SpazioDati che mette a disposizione dell'utente servizi di analisi semantica testuale; grazie ad esso é possibile, dato un testo, estrarne le entitá semantiche principali (*Entity Extraction*), trovare la lingua in cui é 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 in oltre delle RESTful-API che permettono di confrontare due testi generando un indice di similitudine fra i due (*Text Similarity*) e un motore di ricerca di entitá di Wikipedia (*Wikisearch*), nel caso si voglia trovare il titolo di un contenuto senza conoscerlo a priori.

1.2 Struttura Generale

Per l'implementazione si é 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 é 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.

1.3 Il Client

Dalla documentazione ufficale delle API di Dandelion si evince che ogni end-point richiede uno o piú 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 é scelto di gestire separatamente i metodi HTTP GET e DELETE dato che essi non permettono il passaggio parametri nel boby della chiamata HTTP. Pertanto é 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 é 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 verrá usato il metodo POST.

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

```
1  public Task<T> CallApiAsync<T>(string uri, List<KeyValuePair<string, string>> content,
       HttpMethod method = null)
2  {
3      var result = new HttpResponseMessage();
4      if (method == null)
5      {
6          method = HttpMethod.Post;
7      }
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         }
```

```
        string resultContent = await result.Content.ReadAsStringAsync();
        if (result.StatusCode == System.Net.HttpStatusCode.RequestUriTooLong)
        {
            throw new ArgumentException(ErrorMessages.UriTooLong);
        }
        if (!result.IsSuccessStatusCode)
        {
            throw new Exception(resultContent);
        }
        return JsonConvert.DeserializeObject<T>(resultContent);
    });
}
```

1.4 Testing

La maggiorparte dei servizi é 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.

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

1.5 Documentazione

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

1.6 Nuget

La dll generata dalla compilazione della libreria é 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 é stato caricato sul portale online Nuget ed é ora possibile includerlo in un progetto tramite il comando cli:

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

2 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 entitá semantiche, il loro valore di correlazione (relatedness).

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

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 entitá con identificativi a e b.

Questa scelta implementativa, attualmente adottata, é 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.

2.1 Implementazione Iniziale

2.1.1 Il Dump

```
53676192
4922289
0.01
2
1.0
53676158 7 null
53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH
53676016 13 RDsFRUkyASRLFf3T
53675811 16 null
...
```

Leggendo le prime righe di un dump si nota che le prime cinque righe sono valori di inizializzazione mentre dalla sesta righa 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 entitá non sono necessariamente sequenziali (fra due di essi potrei avere dei "buchi", degli intervalli in cui gli identificativi non sono associati ad alcuna entitá) esiste una funzione *map* che mappa ("compatta") gli id delle entitá su altri id univoci e sequenziali.

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

$$\begin{aligned} &map() : \ \mathbb{N} \to \mathbb{N} \\ &[0, max\_id] \to [0, nodeSize], max\_id \geq nodeSize \end{aligned} \tag{2.1}$$

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 verrá mappato su 0, il penultimo id verrá mappato su 1 e cosí via. Nel caso del dump in questione abbiamo:

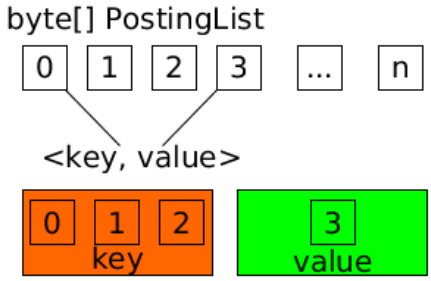
```
MaxId = 53676192
NodeSize = 4922289

   ID           MAP(ID)   POSTINGLIST
53676158   ->     7          null
53676159   ->     6          FW/I...
53676164   ->     5          NnAw...
53676174   ->     4          null
53676176   ->     3          null
53676177   ->     2          null
53676180   ->     1          null
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 é composta da tre valori separati da uno spazio. Ad esempio se consideriamo la riga:

```
53676164 5 NnAwQT/ZvTVJ05MeSfcdI0rMrg1LBmUH
```

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



In pratica una tupla $< chiave,\ valore >$ é codificata sull'array da una sequenza di quattro byte; quindi partizionando l'array in gruppi da 4 byte ottengo, per ogni gruppo, la chiave (é un id "mappato") codificata sui primi tre byte ed il valore (la relatedness) definito sul quarto byte (il byte piú a destra).

Questa codifica, pertanto, permette di gestire set di entitá 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*, é ordinato per valore crescente delle chiavi e, dato che rel(a, b) = rel(b, a), si é 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 $relatedness(a, d)$ con $map(a) > map(d)$.

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

Ovviamente é possibile che c sia *null* (l'entitá con id a non ha nessuna correlazione con entitá i cui id "mappati" siano inferiori a b) oppure che nella posting non sia presente nessuna chiave corrispondente a *map(d)*, in questi casi $relatedness(a, d) = 0$.

2.1.2 Il Codice Nativo

```
// STRUTTURA DATI
public class RelatednessMatrix {
    public int[] map;
    public byte[][] matrix;
    public float configuredMinRel;
    public int configuredMinIntersection;
    public float threshold;
    public static int EL_SIZE = 4;
}

// CODICE UTILIZZO
public float rel(int a, int b){
    if(a==b) return 1f;

    int nodeA = data.map[a];
    int nodeB = data.map[b];
    if (nodeA < 0 || nodeB < 0) return 0f;

    int key = a>b ? nodeB : nodeA;
    byte[] array = a>b ? data.matrix[nodeA] : data.matrix[nodeB];

    if (array == null) return 0f;

    int start = 0;
    int end = array.length - RelatednessMatrix.EL_SIZE;
    int pos = -1;
    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
```

```
            int wid = Integer.parseInt(splittedLine[0]);
            int node = Integer.parseInt(splittedLine[1]);
            data.map[wid] = node;

            if (splittedLine[2].equals("null")) {
                data.matrix[node] = null;
            } else {
                byte[] a = Base64.getDecoder().decode(splittedLine[2].toString());
                data.matrix[node] = Base64.getDecoder().decode(splittedLine[2].toString());
            }
            idx++;
        }

        if (idx != nodesSize) {
            throw new Exception("Wrong format relatedness file the number of line do not
                match with size of matrix");
        }

        return data;
    } catch (Exception e) {
        System.out.println(e.getMessage());
        return null;
    }
}
```

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 dopodiché 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).

```
byte by = array[pos * RelatednessMatrix.EL_SIZE + 3]; //considero il quarto byte
int byint = by + 128; //scalo di 128 valori per ottenere una relatedness in [0, 255]

//cast in float, escludendo i valori sotto la relatedness minima (configuredMinRel)
float byteRel = byint / 255f;
return data.configuredMinRel + byteRel * (1 - data.configuredMinRel);
```

2.1.3 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 piú performane possibile.

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

Tolto un piccolo refact del codice che portebbe ad evitare tre moltiplicazione per ciclo nella binary search (non é 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 peró andrebbe a peggiorare drasticamente lo spreco di memoria. Infatti nell'implementazione attuale la matrice ha dimensione massima $nodeSize \otimes nodeSize$ ma le righe hanno dimensione variabile e non occupano quindi sempre $(nodeSize - 1) \cdot 4\ Byte$.

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:

$$nodeSize \cdot 4\ Byte = 4^2 \cdot 10^6 Byte = 16MB \tag{2.2}$$

(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, avró il 57% di righe a null ($0.57 \cdot 4 \cdot 10^6 \cdot 8\ Byte = 18\ MB$) ed il restante 43% di righe con mediamente 404 valori di relatedness codificati su 4 Byte ($0.43 \cdot 4 \cdot 10^6 \cdot 404 \cdot 4\ Byte = 2.8\ GB$).

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 é uno dei dump piú piccoli).

Stima della Matrice Completa

Se allocassimo in memoria una matrice di Byte completa, di dimensioni $nodeSize \otimes nodeSize$, essa occuperebbe $(4 \cdot 10^6)^2 = 1.6 \cdot 10^{13}\ Byte = 16\ TB$. Pur considerando che esistono strutture dati particolarmente ottimizzate per gestire array di grandi dimensioni con pochi valori al loro interno (teniamo presente che piú della metá 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 peró difficilmente si otterrebbero prestazioni migliori di quelle dell'implementazione attuale dato che il tempo di lookup sarebbe sempre $O(lg_2(noseSize))$.

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 255 valori possibili di relatedness, per minimizzare lo spreco di memoria potrei pensare di allocare 255 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 é 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 archi 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 perché il puntatore al nodo successivo (ipotizzando di lavorare su una macchina a 64 bit) peserebbe piú dell'identificativo del nodo stesso. Al che potrei sostituire il puntatore con un intero ma anche in questo caso avrei il dizionario $< Id,\ Relatedness >$ che pesa 5 Byte mentre nella postingList pesa 4 Byte perché 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 agguintivo di non poter accedere in O(1) ad un nodo arbitrario.

2.2 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 é molto difficile stimare a priori lo spazio occupato da un Hash Map, il modo piú rapido é confrontare con dei banchmark l'implementazione nativa contro quella basata su Hash Map.

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.