



Università degli Studi di Firenze

Correzione parole - Edit distance vs Edit distance con
intersezione di N-Gram

Mattia Pardeo

10/12/2022

Contents

1	Introduzione	2
1.1	Descrizione del problema	2
1.2	Teoria Edit distance	2
1.2.1	Edit distance	3
1.2.2	Edit distance con intersezione di N-Gram	4
2	Documentazione codice	5
2.1	Modulo NGramJaccard	5
2.2	Classe EditDistanceAlg	5
2.3	File main	6
3	Eperimenti	8
4	Risultati sperimentalni	10
4.1	Test correzione parole con e senza indice	10
4.2	Test indice N crescente	11
5	Conclusioni	16

Chapter 1

Introduzione

1.1 Descrizione del problema

In questo esercizio vogliamo analizzare quali sono i tempi di esecuzione dell'algoritmo Edit distance nel caso in cui sia usato singolarmente o combinato con gli indici N-Gram nel contesto di correzione di parole ricevute in ingresso a partire da un lessico di parole corrette.

1.2 Teoria Edit distance

L'Edit distance è un algoritmo appartenente al gruppo di algoritmi di **programmazione dinamica**, ovvero tutti quegli algoritmi che risolvono spesso problemi di ottimizzazione in cui i sotto-problemi non sono indipendenti ma ne condividono i sotto-problemi, questo fa sì che si può salvare e all'occorrenza richiamarne la soluzione già trovata in precedenza. In particolare questi tipi di problemi posseggono una struttura sub-ottima della soluzione ottima, ovvero è combinazione degli ottimi dei sotto-problemi, infatti per questo in genere il problema si risolve con un algoritmo ricorsivo, in genere bottom-up ma anche top-down. I passi per scrivere un algoritmo di programmazione dinamica sono:

- Caratterizzare la struttura di una soluzione ottima
- Definire ricorsivamente il valore di una soluzione ottima del problema
- Calcolo dell'ottimo con un approccio bottom-up
- Costruisco la soluzione ottima

Nel nostro caso vedremo che l'algoritmo non sarà però di tipo ricorsivo in quanto porterebbe ad un tempo esponenziale, invece vedremo che sarà possibile definirlo con un tempo polinomiale.

1.2.1 Edit distance

Il compito dell'algoritmo è quello di andare a calcolare qual è la distanza che c'è tra due parole, in particolare quanti sono i passi che servono per trasformare la prima parola nella seconda cercando di mantenere il costo delle operazioni al minimo, questo perché, in base al contesto in cui è applicato, le operazioni che vengono eseguite hanno un costo. In particolare le operazioni ammesse sono quelle di inserimento, cancellazione, copia, sostituzione e scambio di caratteri, in genere il costo della copia è posto a zero e quello dell'inserimento, cancellazione e sostituzione a uno. L'algoritmo andrà a risolvere il problema partendo da problemi più piccoli, in questo caso corrisponde ad andare a risolvere il problema dell'edit distance di sottosequenze delle parole di partenza, dove una sotto-seguenza è una sequenza di caratteri appartenente alla parola di partenza disposti nello stesso ordine della parola di partenza, ad esempio: da "algoritmo", "algori" è una sotto-seguenza, "agri" lo è anche, "algrio" pur avendo gli stessi caratteri non lo è in quanto non sono in ordine. Se X è una stringa, allora una sua sotto-seguenza si indica come $X_i = \langle x_1, \dots, x_i \rangle$. L'idea generale è: conosciamo la tabella dei costi $c[i, j]$ della soluzione ottima al sotto-problema $X_i \rightarrow Y_j$ e supponiamo di conoscere l'operazione fatta in precedenza così da poter calcolare la soluzione successiva conoscendo quella precedente. A questo punto ci troviamo con 6 possibilità da analizzare:

- Se l'ultima operazione era una **copia** significa che era $x_i = y_j$, rimane quindi il sotto-problema $X_{i-1} \rightarrow Y_{j-1}$. Dovendo avere una sottostruttura ottima allora deve essere inclusa la soluzione del problema $X_{i-1} \rightarrow Y_{j-1}$, si avrà quindi $c[i, j] = c[i-1, j-1] + \text{costo(copia)}$.
- Se l'ultima operazione era una **sostituzione** allora era $x_i \neq y_j$, come sotto-struttura ottima avremo ancora il sotto-problema $X_{i-1} \rightarrow Y_{j-1}$ e quindi avremo $c[i, j] = c[i-1, j-1] + \text{costo(sostituzione)}$.
- Se l'ultima operazione era una **cancellazione** non abbiamo restrizioni su X e Y e viene rimosso l'ultimo carattere da X_i lasciando Y_j invariato, rimane quindi il sotto-problema $X_{i-1} \rightarrow Y_j$ e $c[i, j] = c[i-1, j] + \text{costo(cancellazione)}$.

- Se l'ultima operazione era un **inserimento** abbiamo l'opposto della cancellazione, in questo caso rimuovo l'ultimo carattere di Y_j e ritrovo il sotto-problema $X_i \rightarrow Y_{j-1}$ con $c[i, j] = c[i, j-1] + \text{costo}(inserimento)$.
- Se l'ultima operazione era uno **scambio** avremo avuto $x_i = y_{j-1}$ e $x_{i-1} = y_j$, nel caso però di i e j maggiori uguali a 2, il sotto-problema rimanente sarà quindi $X_{i-2} \rightarrow Y_{j-2}$ e $c[i, j] = c[i-2, j-2] + \text{costo}(scambio)$.
- Casi base:
 - Se $i = 0$ allora X_0 è stringa vuota, si deve quindi convertire la stringa vuota in Y_j , si avranno quindi j inserimenti e quindi $c[i, j] = j * \text{costo}(inserimento)$.
 - Se $j = 0$ allora Y_0 è stringa vuota, si deve quindi convertire in stringa vuota, avremo allora i cancellazioni e quindi $c[i, j] = i * \text{costo}(cancellazione)$.
 - Se $i = j = 0$ allora non si deve fare nulla e si hanno costi zero.

Questi casi ci portano alla formulazione ricorsiva del problema applicata in modo bottom-up:

$$c[i, j] = \begin{cases} c[i - 1, j - 1] + \text{costo}(copia) & \text{se } x_i = y_j \\ c[i - 1, j - 1] + \text{costo}(sostituzione) & \text{se } x_i \neq y_j \\ c[i - 2, j - 2] + \text{costo}(scambio) & \text{se } x_i = y_{j-1} \text{ e } x_{i-1} = y_j \\ c[i - 1, j] + \text{costo}(cancellazione) & \text{sempre} \\ c[i, j - 1] + \text{costo}(cancellazione) & \text{sempre} \end{cases}$$

1.2.2 Edit distance con intersezione di N-Gram

Un N-Gram è una sequenza di n caratteri presi in sequenza da una parola utile per creare un insieme di sequenze facendoli a partire dal primo carattere della parola e continuare in sequenza finché possibile, ad esempio: il bi-gram di canarino è ca, an, na, ar, ri, in, no. Quando abbiamo un lessico contenente un numero molto grande di parole corrette andare ad eseguire Edit distance su tutte queste parole per correggerne una potrebbe richiedere molto tempo, un modo per andare a diminuire le parole su cui eseguire l'algoritmo è fare l'intersezione con gli N-Gram, ovvero andare a calcolare un N-Gram della parola da correggere e confrontarlo con gli N-Gram delle altre parole, in particolare potremmo andare a prendere le parole che hanno più sequenze di caratteri in comune con la parola di partenza. Una misura normalizzata che si può utilizzare è il **coefficiente di Jaccard** definito come $J = \frac{|A \cap B|}{|A \cup B|}$ tramite il quale impostare una soglia di similarità tra parole.

Chapter 2

Documentazione codice

2.1 Modulo NGramJaccard

All'interno di questo modulo troviamo la funzione **ngram(n, string)** per calcolare l'N-Gram di una parola che verrà restituito come un oggetto Multiset, questo si è reso necessario in quanto alcune sequenze potrebbero ripetersi ed utilizzare un semplice Set avrebbe eliminato la ricorrenza di questi duplicati, invece il Multiset permette di avere al suo interno anche elementi uguali. Al suo interno è presente anche la funzione **jaccard_coefficient(set1, set2)** che prende in ingresso due Set o Multiset e ne va a calcolare il coefficiente di Jaccard. La funzione **create_ngrams_from_file()** è necessaria per creare un file che, a partire da un file .txt che fungerà da lessico, conterrà gli N-Grams delle parole del lessico. La funzione **insert_in_list(lista, index, value, caller)** è utile per inserire l'elemento value nella lista all'indice index, necessaria in quanto la lista, essendo quella di Python, potrebbe non presentare quella posizione se è più grande della dimensione attuale, in questo modo possiamo modificarla in modo tale da averla, il caller è necessario in quanto l'inserimento è gestito in modo leggermente diverso tra la funzione che trova le correzioni e **create_ngrams_from_file()**.

2.2 Classe EditDistanceAlg

In questa classe sono presenti le variabili che rappresentano i costi delle varie operazioni che vengono effettuate all'interno dell'algoritmo, in più è presente una lista, **ngrams_list**, che verrà usata dalla correzione di parole con N-Grams al cui interno sono presenti, in modo ordinato in base all'N utilizzato per creare il rispettivo N-Gram con cui sono state suddivise, le varie parole del lessico col rispettivo N-Gram, ovvero se una parola è stata sal-

vata con un 1-gram allora si troverà nella posizione 0, se è stato usato un 4-gram allora sarà nella posizione 3, in particolare ogni posizione contiene una ulteriore lista con tutte le coppie parola:N-Gram che sono state divise con lo stesso N. La lista verrà inizializzata la momento che viene creato l'oggetto. Al suo interno si trovano come metodi l'algoritmo di edit distance (**edit_distance(x, y)**) e il metodo per andare a stampare la sequenza di operazioni salvate (**op_sequence(op, i, j)**). Troviamo anche i due metodi che permettono di trovare le correzioni sia usando gli N-Grams che senza: **correction_word(word)** prende in ingresso la parola da correggere e va ad eseguire edit distance su tutte le parole del lessico (file "1160 parole italiane.txt") e ritorna vero se la parola è corretta, falso se non sono state trovare correzioni oppure una lista di liste contenente le possibili correzioni e l'indice (che corrisponde anche al costo minimo) in cui si trovano le possibili correzioni; **correction_word_ngram(word)** prende anche lei la parola da correggere ma stavolta va a sfruttare gli N-Grams precedentemente creati, andrà a calcolare l'N da usare per calcolare il suo N-Gram e grazie al fatto che la lista `ngrams.list` è ordinata si può usare come indice per andare a prendere gli N-Grams con cui confrontarla, successivamente andrà a confrontare gli N-Grams e tramite il coefficiente di Jaccard, presente nell'altro modulo, va a selezionare le parole da confrontare e ridurne così il numero possibile rispetto a tutto il lessico, alla fine ritorna gli stessi valori della funzione precedente. Esiste poi il metodo **correction_word_with_fixed_n(n, word_to_correct, correct_words)** che prende un numero n fissato con cui fare l'N-Gram, una parola da correggere ed una lista di parole corrette con cui confrontare la parola.

2.3 File main

Al suo interno troviamo le funzioni che vanno ad eseguire i test, in particolare **test_ed(to_return, edit_d, word_to_test, edit_distance_times)** e **test_ed_ngram(to_return, edit_d, word_to_test, edit_distance_ngrams_times)** che ricevono in ingresso entrambe una lista (**to_return**) in cui salvare il numero di correzioni trovare in quella iterazione che sarà usata per calcolare il numero di correzioni medie trovate, vanno a richiamare le correzioni senza e con N-Gram, **edit_d** che è l'oggetto che possiede i metodi da richiamare, **word_to_test** è la parola di cui trovare delle correzioni e l'ultimo parametro è una lista al cui interno salvare il tempo trascorso ad eseguire l'edit distance in quella iterazione così da calcolarne il tempo medio. Al suo interno si trovano le chiamate quindi alla correzione di parole e il calcolo del tempo trascorso e salvato il numero di correzioni trovate. La funzione **print_result(result,**

index) prende in ingresso un valore che può essere vero, falso o una lista e un indice, in particolare sono i valori restituiti dai metodi delle correzioni da utilizzare per stampare e video i risultati e se presenti le parole con cui correggere. Sono poi presenti le 3 funzioni che effettivamente vanno ad eseguire i test, ovvero **correct_words_test()**, **uncorrect_words_test()** e **not_in_words_test()**, la prima va a testare quanto tempo ci voglia a capire che la parola passata è in realtà corretta e non ha bisogno di correzioni. La seconda invece va a testare 4 tipi di errori, l'ultima invece va a testare quali correzioni vengono presentate quando si immettono parole non appartenenti al lessico. In ognuna di queste funzioni vengono calcolate le medie dei tempi trascorsi e la media delle correzioni trovate. Le ultime funzioni che sono presenti sono **multiple_ngrams_test**, **textbf{test}_ed_multiple_ngrams**, e **draw_plot(results, test_name, word)**, in particolare queste ultime due sono richiamate nella prima e servono per fare il test di correzione di parole ogni volta cambiando l'indice N fino ad un certo massimo e vedere con i vari N quali sono i tempi di esecuzione e se sono state trovate correzioni e successivamente disegnarne il grafico.

Chapter 3

Esperimenti

Gli esperimenti sono stati condotti su un pc dotato di processore a 6 core e 12 thread con frequenza base di 3.2GHz e 3.6GHz in boost, la memoria RAM a disposizione è di 16GB ed il sistema operativo è Windows 10. La versione di Python utilizzata è la 3.10.

In ognuno di questi test vengono create delle liste di parole che verranno utilizzate nei test e saranno prese dai due file .txt:

- Nel test delle parole corrette vengono estratte a caso dal file "1160 parole italiane.txt" ed inserite nella lista `correct_words_to_test`, successivamente verrà creato l'oggetto `Edit` col quale poi eseguire i test e le liste in cui salvare i tempi nelle varie iterazioni, `edit_distance_times` e `edit_distance_ngrams_times`, le variabili `ed_correct` e `ed_ngrams_correct` erano usate per controllare che effettivamente venisse riconosciuta la parola come corretta, non effettivamente usate nel test. Sarà quindi iterata la lista delle parole da testare e successivamente saranno calcolati i tempi medi nel caso di edit distance classico e con N-Grams. Il numero di parole testate sono la metà di quelle presenti nel file, ovvero 580.
- Nel test delle parole errate invece andiamo come anticipato andiamo a testare 4 tipologie di errori, in particolare:
 - Carattere errato alla fine della parola.
 - Carattere errato in cima alla parola.
 - Carattere errato in mezzo alla parola.
 - Carattere mancante.

Questi caratteri vengono scelti casualmente come anche la posizione del carattere mancante. Anche in questo caso vengono create delle liste al

cui interno verranno messe delle parole corrette come fatto precedentemente ma viene stavolta inserito l'errore. Lo stesso verranno iterate le liste da testare e successivamente calcolati i tempi medi. Stavolta vengono create anche delle liste in cui salvare ad ogni iterazioni il numero di correzioni che sono state trovate per calcolare in media quante correzioni vengono trovate. Il numero di parole di testate per ogni tipo di errore è 580.

- Il penultimo test è quello delle parole non presenti nel lessico. Anche qua vengono testati tempi e correzioni trovate. Le vengono estratte delle parole dal file "9000 nomi propri.txt" che sicuramente non ha nessuna parola del file utilizzato come lessico. Il numero di parole testate sono la metà di quelle nel file, quindi 4500.
- L'ultimo test esegue va a selezionare solo 4 parole e crea gli errori visti in precedenza, però questa volta andiamo a testare quanto tempo ci mette a trovare una correzione, insieme al fatto che ne trovi qualcuna, a partire da nessun N-Gram, quindi $N = 0$ come fosse Edit distance classico, fino ad arrivare all'indice N calcolato come negli altri test.

Quando vengono usate le intersezioni N-Grams l'indice viene calcolato come la lunghezza della parola divisa 3 e parte intera superiore.

Chapter 4

Risultati sperimentali

4.1 Test correzione parole con e senza indice

I risultati trovati, espressi in secondi, sono i seguenti:

Versione Python: 3.10.0

TEST PAROLE CORRETTE

Tempo medio per parole corrette: 3.30e-01
Tempo medio per parole corrette con N-Grams: 1.63e-02

TEST PAROLE NON CORRETTE

Tempo medio correzione con errato ultimo carattere: 3.86e-01
Tempo medio correzione con errato ultimo carattere con N-Grams: 1.48e-02

Tempo medio correzione con errato primo carattere: 3.66e-01
Tempo medio correzione con errato primo carattere con N-Grams: 1.48e-02

Tempo medio correzione con errato carattere centrale: 3.67e-01
Tempo medio correzione con errato carattere centrale con N-Grams: 1.46e-02

Tempo medio correzione senza un carattere: 3.67e-01
Tempo medio correzione senza un carattere con N-Grams: 1.55e-02

Correzioni trovate in media per errore ultimo carattere: 1.02e+00
Correzioni trovate in media per errore ultimo carattere con N-Grams: 1.03e+00

Correzioni trovate in media per errore primo carattere: 1.05e+00

Correzioni trovate in media per errore primo carattere con N-Grams: 1.04e+00

Correzioni trovate in media per errore carattere centrale: 1.04e+00

Correzioni trovate in media per errore carattere centrale con N-Grams: 1.06e+00

Correzioni trovate in media per senza carattere: 1.00e+00

Correzioni trovate in media per senza carattere con N-Grams: 1.00e+00

TEST PAROLE NON IN LESSICO

Tempo medio correzione: 3.48e-01

Tempo medio correzione con N-Grams: 1.66e-02

Correzioni trovate in media: 4.51e+00

Correzioni trovate in media con N-Grams: 1.29e+00

Possiamo vedere come per tutti i test i tempi medi per la correzione usando Edit distance e intersezione di N-Gram sono di un ordine inferiore rispetto all'utilizzo di Edit distance standard e all'incirca qualunque tipo di test facciamo i tempi rimangono molto simili. Per quanto riguarda invece il numero medio di correzioni che vengono trovate sono più o meno le stesse per entrambe le metodologie, se non per il test delle parole non nel lessico dove la metodologia senza N-Gram trova più parole con cui correggere, mentre l'altra versione si attesta sempre su circa una parola trovata.

4.2 Test indice N crescente

In questo caso possiamo vedere nelle figure i tempi necessari a correggere le 4 parole con i 4 errori. I quadrati verdi significano che sono state trovate delle correzioni, i quadrati rossi no.

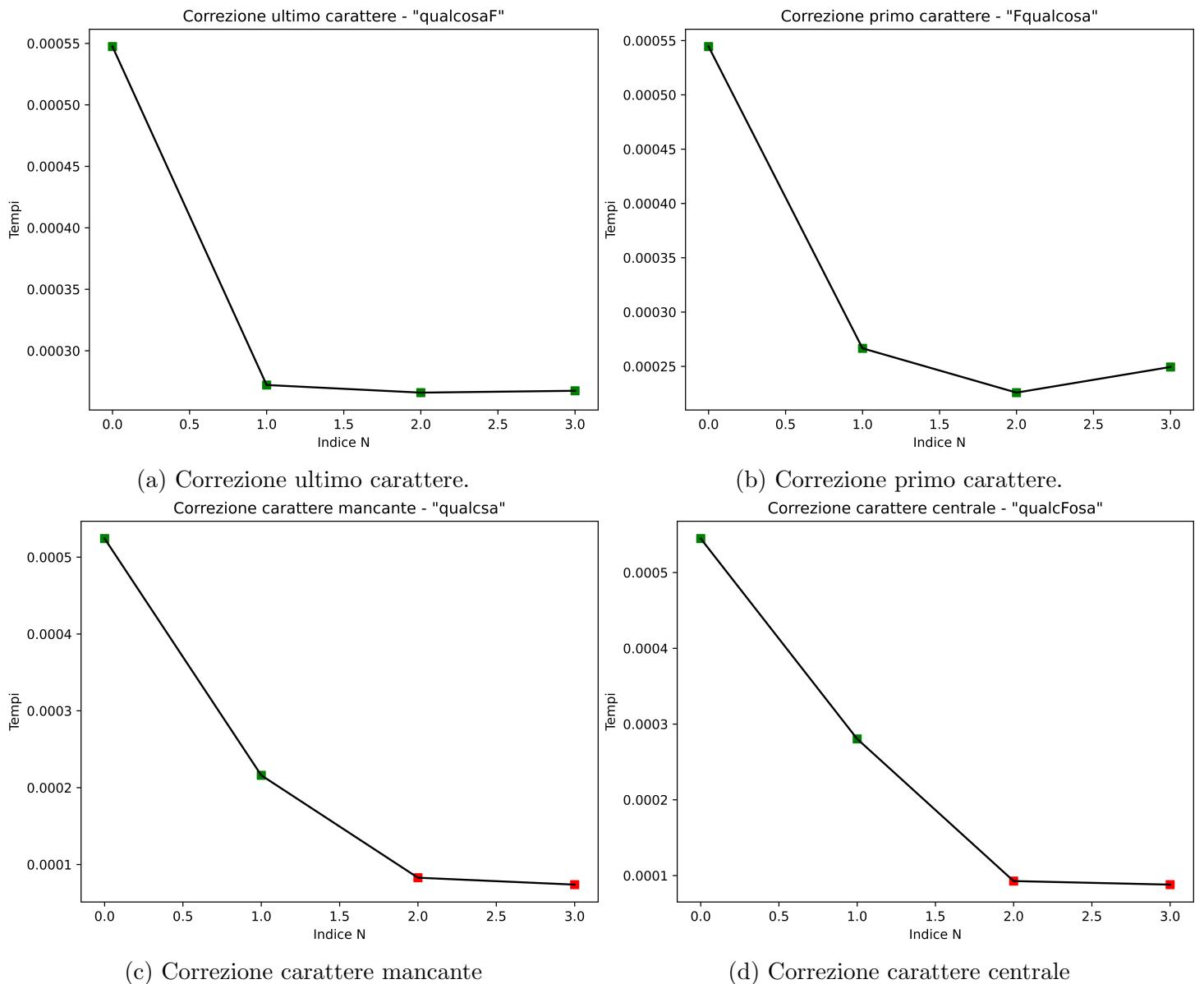


Figure 4.1: Correzione parola "qualcosa" con i 4 errori.

Come possiamo vedere in figura 4.1 non appena si inizia ad usare l'intersezione con gli N-Gram il tempo cala di molto fino a rialzarsi leggermente all'aumentare dell'indice utilizzato, il che è normale in quanto le intersezioni utilizzate nel coefficiente di Jaccard diventano più piccole. Si può notare come nella correzione di carattere mancante e carattere centrale con indice più grande di 1 inizia a non trovare delle correzioni. Queste osservazioni sono riscontrabili anche nelle altre 3 parole testate.

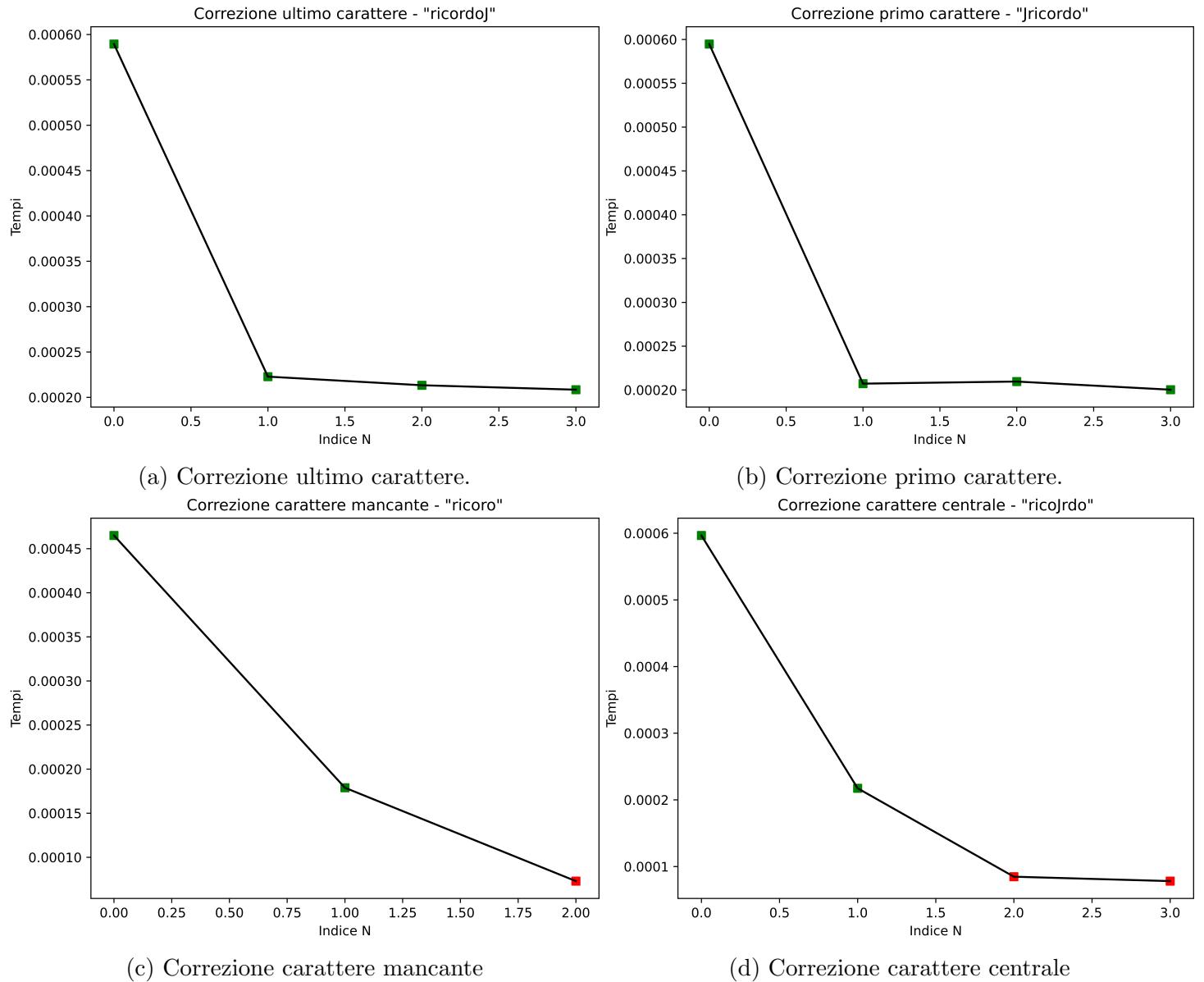


Figure 4.2: Correzione parola "ricordo" con i 4 errori.

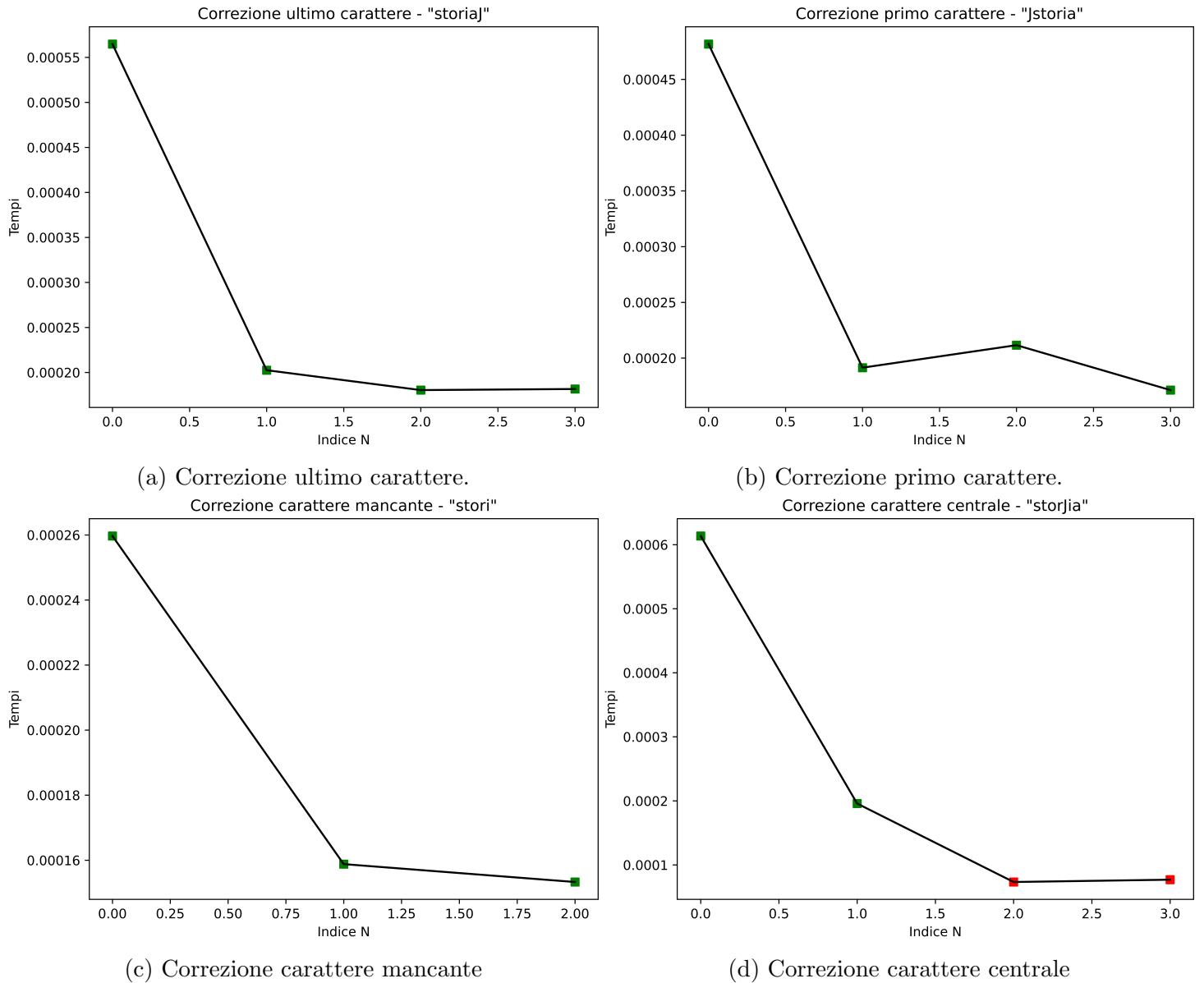


Figure 4.3: Correzione parola "storia" con i 4 errori.

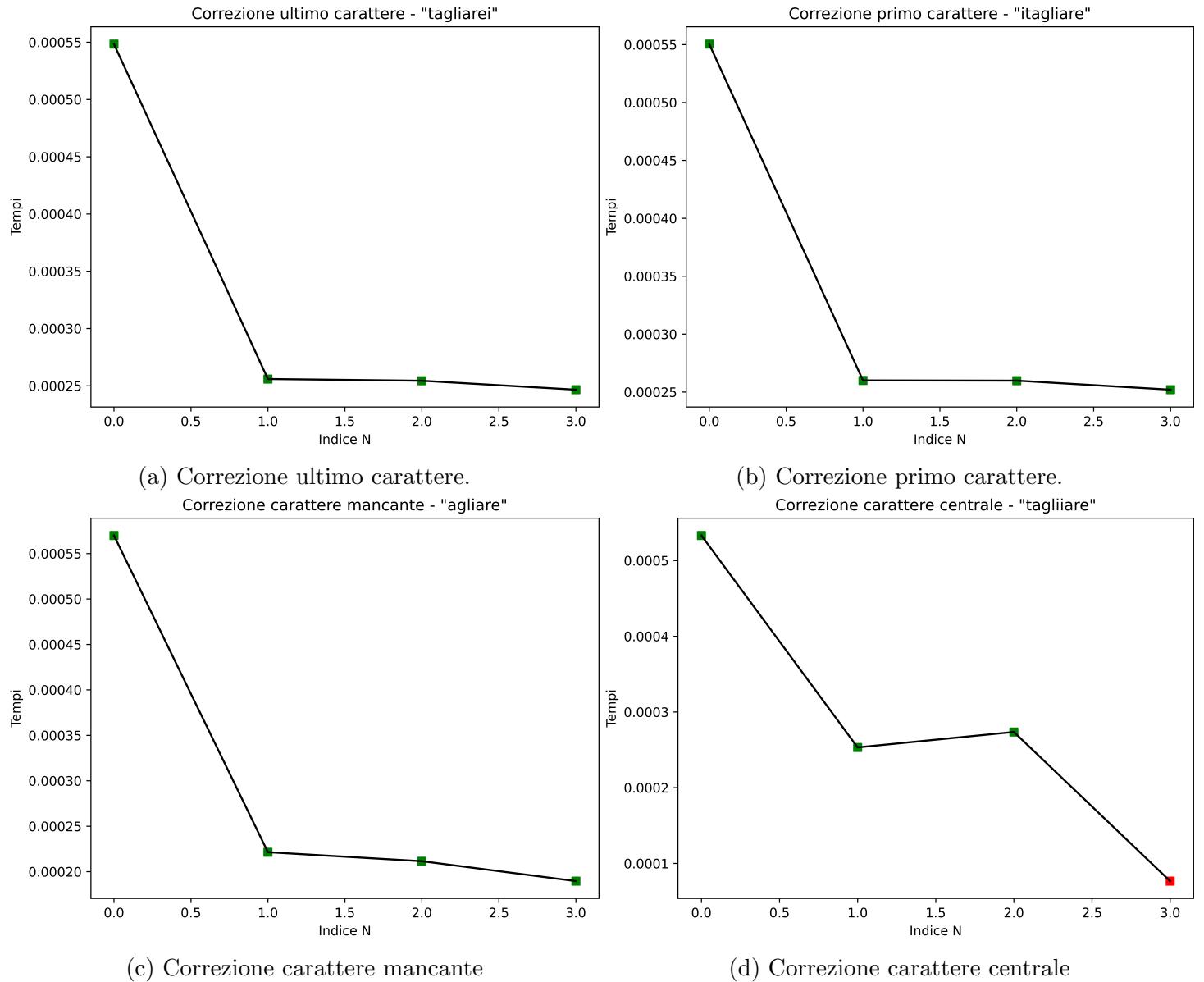


Figure 4.4: Correzione parola "tagliare" con i 4 errori.

Chapter 5

Conclusioni

In conclusione possiamo affermare che l'utilizzo di Edit distance con le intersezioni N-Gram per trovare la distanza da delle parole fornite in ingresso con una query dal lessico presente in quel contesto è più veloce rispetto alla metodologia che utilizza che va ad eseguire l>Edit distance su tutte le parole del lessico.