UNIVERSITÀ DEGLI STUDI DI FIRENZE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea in Ingegneria Informatica

Edit Distance

Professore:
Ch.mo Prof.
Simone Marinai

Candidato:
Hamidreza Hashemi
Mat. 5996260

Corso:
Algoritmi e Strutture Dati

ANNO ACCADEMICO 2020/2021

Dedicata a coloro che
ci provano ogni giorno.

INDICE

INTRODUZIONE

In questo eserizio vediamo insieme l'algoritmo edit distance e l'implementazione nel linguaggio python. Questo algoritmo serve per trovare la parola più simile ad una query in un lessico di parole.

CAPITOLO 1

EDIT DISTANCE

L'algoritmo Edit Distance trova la distanza tra una parola e un'altra. La distanza qui vuoldire il numero di volte di fare certe operazioi per arrivare alla parola di lessico. Questa distanza si chiama la distanza di Levenshtein. Queste operazioni sono :

- Inserimento
- Cancellazione
- Sostituzione

```python
def editDistDP(str1, str2, m, n):
    cost = np.zeros((m + 1, n + 1))
    for i in range(m + 1):
        for j in range(n + 1):
            if i == 0:
                cost[i, j] = j
            elif j == 0:
                cost[i, j] = i
            elif str1[i - 1] == str2[j - 1]:
                cost[i, j] = cost[i - 1, j - 1]
            else:
                cost[i, j] = 1 + min(cost[i, j - 1], cost[i - 1, j], cost[i - 1, j - 1])
    return cost[m, n]
```

Figura 1.1: Edit distance in python

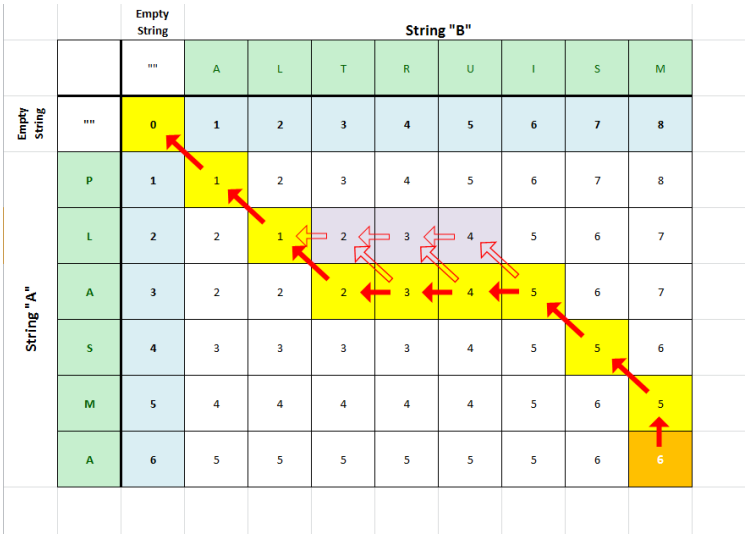

Figura 1.2: Esempio di Edit Distance

Il costo temporale di edit distance è $\theta(m \cdot n)$ dove m e n sono le lunghezze di stringhe. Questo algoritmo può essere applicato in Biologia ed elaborazione del linguaggio naturale.

CAPITOLO 2

IMPLEMENTAZIONE

Ci sono diversi modi per implementare algoritmo edit distance. Uno di questi è l'implementazione con N-Grams. N-Grams divide una stringa e crea una sottosequenza di n elementi. Questo implementazione normalmente serve per migliorare il costo temporale dell'algoritmo perchè prima di trovare la distanza tra le due parole lunghe, va a vedere se esiste una sottosequenza di N-Gram nella parola di lessico e poi va a calcolare la distanza. Vediamo un'esempio qui sotto :

..AGCTTCGA.. $\longrightarrow$ .., AGC, GCT, CTT, TTC, TCG, CGA,..

Questo è un esempio di **3-Gram**.

Si può trovare la distanza tra le due parole calcolando il **Coefficiente di Jaccard** tra gli N-Grams. la formula del coefficiente di Jaccard è :

$$JC = \frac{A \cap B}{A \cup B}$$, dove A e B sono i set di N-Grams

```
def run(self):
    ngramstrings = []
    for element in self.query:
        ngramstrings.append(letters(element))
    dojaccardcalculations = False
    print("Starting " + self.name)
    for line in self.linesarray:
        # Vede se esiste un ngram nella parola di lessico. Altrimenti non fa i calcoli.
        if any(word in line for word in ngramstrings):
            line_set = set(ngrams(line, self.ngram))
            # Calcola il coefficiente jaccard tra query e una parola
            jc = jaccard_coef(self.query, line_set)
            if jc > self.top3[0]:
                self.top3[0] = jc
                self.similar_words[0] = line
            elif jc > self.top3[1]:
                self.top3[1] = jc
                self.similar_words[1] = line
            elif jc > self.top3[2]:
                self.top3[2] = jc
                self.similar_words[2] = line
    print("Exiting " + self.name)
```

Figura 2.1: ngrams in thread

In questo esercizio è stato implementato Edit distance semplice e con gli N-Grams. Il programma trova le parole che hanno almeno una sottosequenza di N-Grams nella parola di query. Poi va a calcolare il coefficiente di Jaccard tra gli N-Grams e da come output i primi 3 valori più simili rispetto alla parola query. Dopo calcola la distanza tra queste 3 parole con la parola query. Questo serve per ottimizzare il costo temporale. Come vediamo nella figura di sopra, il calcolatore di jaccard è stato implementato come un Thread di python. Ogni thread può cercare la parola più simile in un lessico.

CAPITOLO 3

CONCLUSIONE E CONFRONTO

Abbiamo provato diversi algoritmi per trovare la somiglianza tra 2 parole. Come un esempio è stato provato la somiglianza tra la parola "tigr" e un lessico di parole italiane con l'algoritmo di 3grams. vedete sotto l'output generato :

```
Starting th1
Exiting th1
3Grams !!
Word : tigra
with Jaccard Coefficient : --- 0.6666666666666666 ---
has the minimum distance : 1.0
Word : tigre
with Jaccard Coefficient : --- 0.6666666666666666 ---
has the distance : 1.0
Word : tigri
with Jaccard Coefficient : --- 0.6666666666666666 ---
has the distance : 1.0
Spent time : 0.6595729000000006
```

provando da 1-Gram e aumentando la divisione della query si capisce che il tempo diminuisce e la precisione aumenta. Vediamo dalla tabella di sotto, il risultato di una serie di esperimentazioni :

Tempo trascorso in sec.					
Query	1-Gram	2-Grams	3-Grams	4-Grams	Edit Distance
can	5.3722	2.1627	0.6163	err	67.3821
tigr	5.5277	2.0518	0.6595	0.5442	77.3288
conigl	5.5500	2.1150	1.0300	0.6344	112.7757

Come vediamo per la query "can" abbiamo un errore in 4-grams perchè non è divisibile a 4 caratteri. Dalla tabella di sotto notiamo che aumentando la divisione abbiamo le parole più precise :

Parole trovate					
Query	1-Gram	2-Grams	3-Grams	4-Grams	Edit Distance
can	accana anca can	can cana cancan	can cana cane	err	can
tigr	tigri aggirati argati	tigra tigre tigri	tigra tigre tigri	tigra tigre tigri	tigi
conigl	ciglioni cigolino cinciglio	conigli coniglia coniglie	conigli coniglia coniglie	conigli coniglia coniglie	conigli

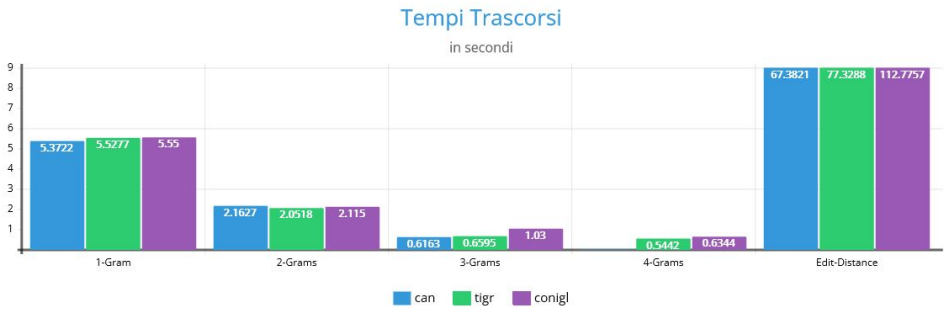

Figura 3.1: tempi trascorsi in sec.

Possiamo notare che L'algoritmo Edit Distance è notevolmente più costoso rispetto ai N-Grams