

# DIPARTIMENTO DI INFORMATICA

## EDIT DISTANCE WITH DYNAMIC-PROGRAMMING

$$\begin{aligned} d_{i0} &= \sum_{k=1}^i w_{\text{del}}(b_k), & \text{for } 1 \leq i \leq m \\ d_{0j} &= \sum_{k=1}^j w_{\text{ins}}(a_k), & \text{for } 1 \leq j \leq n \\ d_{ij} &= \begin{cases} d_{i-1,j-1} & \text{for } a_j = b_i \\ \min \begin{cases} d_{i-1,j} + w_{\text{del}}(b_i) \\ d_{i,j-1} + w_{\text{ins}}(a_j) \\ d_{i-1,j-1} + w_{\text{sub}}(a_j, b_i) \end{cases} & \text{for } a_j \neq b_i \end{cases} & \text{for } 1 \leq i \leq m, 1 \leq j \leq n. \end{aligned}$$

		String "B"									
		Empty String	A	L	T	R	U	I	S	M	
String "A"	Empty String	--	1	2	3	4	5	6	7	8	
	P	1	1	2	3	4	5	6	7	8	
	L	2	2	2	3	4	5	6	7		
	A	3	2	2	2	3	4	5	6	7	
	S	4	3	3	3	3	4	5	5	6	
	M	5	4	4	4	4	4	5	6		
	A	6	5	5	5	5	5	5	6	6	

$$dp[i][j] = \begin{cases} dp[i-1][j-1], \text{ if } A[i] == B[j] \\ 1 + \min \begin{pmatrix} dp[i-1][j], \\ dp[i][j-1], \\ dp[i-1][j-1] \end{pmatrix} \text{ if } A[i] \neq B[j] \end{cases}$$

## Relazione Esercitazione 2 : EditDistance con l'uso della Programmazione Dinamica.

**Nome : Andrea**

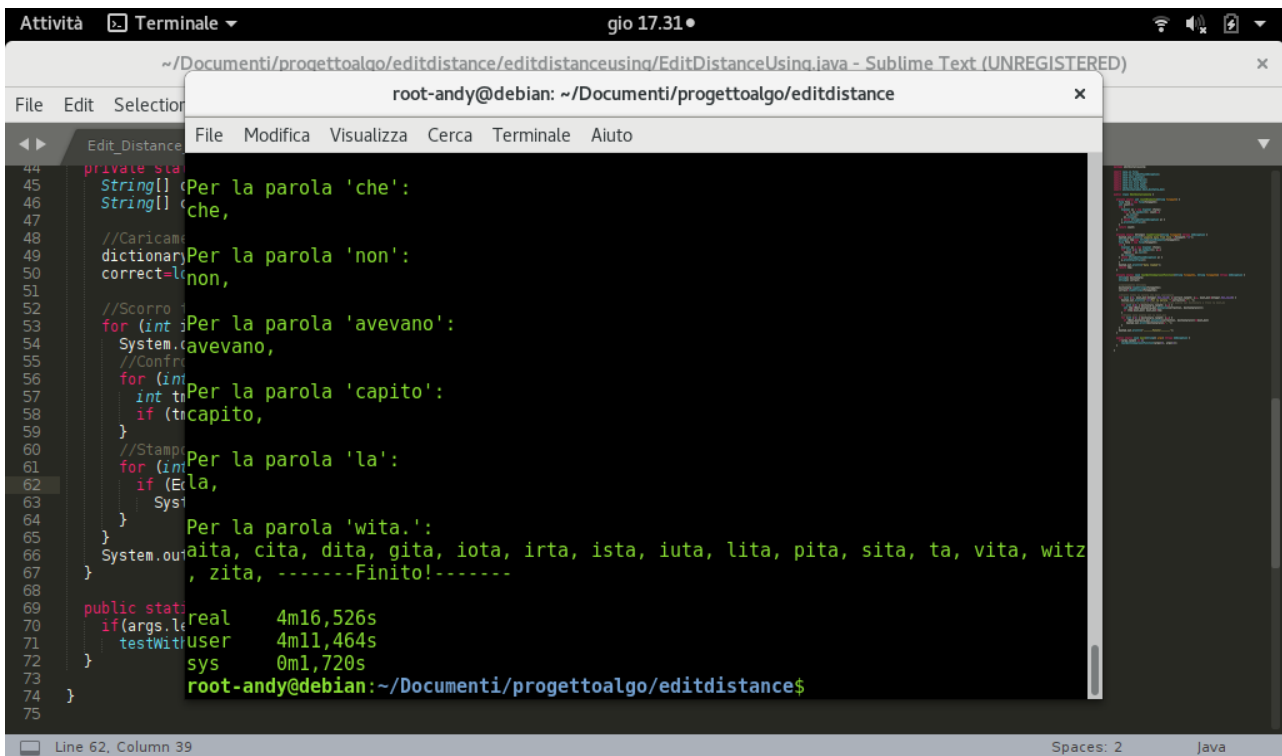
**Cognome : Senese**

**Matricola : 811634**

**Data la ripetizione di casi già risolti nell'esecuzione dell'algoritmo edit\_distance ricorsivo tradizionale tramite la programmazione dinamica si è verificato che questi sottocasi vengono evitati di essere risolti se già incontrati in passato in modo da evitare la ripetizione dei singoli problemi più volte. La complessità in termini di tempo dell'algoritmo è  $O(mn)$  per il riempimento della matrice di calcolo della distanza di edit, ed  $O(n+m)$  per la ricostruzione della trasformazione da S1 a S2 quindi:**

$$**O(nm) + O(n+m) = O(nm)**$$

**vale anche per la complessità spaziale è  $O(mn)$  ma è possibile ridurla a  $O(\min(n,m))$ . Si può osservare così che in qualsiasi istante l'algoritmo richiede solo due righe(o due colonne) in memoria. Tuttavia questa ottimizzazione dell'algoritmo rende impossibile leggere la serie minima di operazioni di modifica. Il tempo di esecuzione dell'algoritmo sui file "Dictionary.txt" && "correctme.txt" è riportato qui sotto :**



```
44 private static String[] Per la parola 'che':
45 String[] che,
46
47 //Caricamento del dizionario
48 dictionary Per la parola 'non':
49 correct=non,
50
51 //Scorrendo il dizionario
52 for (int i=0; i<dictionary.length; i++)
53     System.out.println("Per la parola 'avevano':");
54     //Controllo se la parola è presente nel dizionario
55     for (int i=0; i<dictionary[i].length; i++)
56         if (dictionary[i].equals("avevano"))
57             System.out.println("Per la parola 'capito':");
58             if (dictionary[i].equals("capito"))
59                 System.out.println("Per la parola 'la':");
60                 for (int i=0; i<dictionary[i].length; i++)
61                     if (dictionary[i].equals("la"))
62                         System.out.println("Per la parola 'wita.':");
63                         for (int i=0; i<dictionary[i].length; i++)
64                             if (dictionary[i].equals("wita."))
65                                 System.out.println("aita, cita, dita, gita, iota, irta, ista, iuta, lita, pita, sita, ta, vita, witz");
66                                 System.out.println("zita, -----Finito!-----");
67 }
68
69 public static void main(String[] args) {
70     real 4m16,526s
71     if (args.length > 0) {
72         testWitiz(args[0]);
73     }
74 }
75 }
```

Si è deciso di usare una matrice , in quanto l'algoritmo riportato tramite la programmazione dinamica tradizionalmente è risolto con una matrice e non si è voluto adottare una struttura dati differente. Per Concludere si ricapitolano i requisiti per risolvere tale problema:

- 1) Si è identificato che il problema originale può essere risolto suddividendolo in sotto procedure/problemi di grandezza minore.
- 2) Tramite relazione di ricorrenza data dall'esercitazione si è testato che l'algoritmo ricorsivo tradizionale è completo ma non è ottimo proprio per il fatto che va a risolvere eventuali sotto procedure generate dall'approccio divide et impera e le risolve nuovamente quindi è una soluzione abbastanza ingenua.
- 3) Si è proceduto a modificare la soluzione ingenua utilizzando un approccio iterativo memoized(bottom-up) dato che si è riscontrato che le chiamate ricorsive sono un problema per stringhe di cardinalità molto grandi. In questa versione si è utilizzato una matrice per contenere i risultati e si è calcolato in che modo bisognerebbe riempirla utilizzando la stessa formula di ricorrenza identica a quella di partenza. Quindi la soluzione al problema è semplicemente un adattamento dell'algoritmo di partenza tenendo una struttura dati per far in modo che si evitino di ricalcolare ogni volta sotto procedure già incontrate in precedenza.

