Esercizio 2. Utilizzo delle funzioni per il calcolo della "edit distance"

Si riportano i risultati ottenuti in alcuni approcci che sono stati utilizzati. L'obiettivo cercato è la minimizzazione del tempo di calcolo, fortemente dipendente dalla dimensione elevata del file "dictionary.txt".

Approccio n° 1

- I file *correctme.txt* e *dictionary.txt* sono memorizzati rispettivamente in due array di stringhe
- Per salvare le parole con edit distance minima, relativa ad una parola in *correctme.txt* si utilizza un **ArrayList**
- Fissata una parola di *correctme.txt*, per ogni parola in *dictionary.txt*:
  - si tiene traccia del minimo tra tutte le edit distance
  - si calcola l'edit distance tra le due parole
  - se l'edit distance è minore del minimo allora svuota la lista e aggiungi la parola corrente
  - se l'edit distance è uguale al minimo la si aggiunge alla lista
  - altrimenti non la si aggiunge
  - se si è trovata una edit distance uguale a 0 allora si interrompe il ciclo (perché ogni parola nel dizionario compare una e una sola volta)
  - fatto scorrere tutto *dictionary.txt* si stampa la lista.

→ Svantaggi:

- il programma impiega molto tempo ad eseguire, data la dimensione dei file
- la sua complessità spaziale dipende dalla somma delle dimensioni dei due file: dal momento che *dictionary.txt* è molto grande, il programma usa molta memoria

Approccio n° 2

Quasi del tutto equivalente all'approccio n° 1, l'unica differenza è stata la scelta di scandire tutto il file *dictionary.txt* usando un **BufferedReader** per ogni parola nella frase, memorizzando soltanto la stringa appena letta e non tutto il file.

→ Vantaggio: la complessità spaziale dipende solo dalla lunghezza della frase in *correctme.txt,* che rappresentando una frase in generale è molto più piccolo di *dictionary.txt*

→ Svantaggi:

- il programma impiega molto tempo come nell'approccio 1
- il programma deve leggere molte volte il file *dictionary.txt*

Approccio n°3 (quello scelto)

Uguale al n°1, ma con la differenza che l'array che contiene le parole di "*dictionary.txt"* è ordinato per lunghezza delle parole (tra parole della stessa lunghezza per ordine alfabetico).

→ Vantaggio: sebbene non sia variato il funzionamento rispetto all'approccio 1 il tempo di esecuzione è quasi dimezzato

Approccio n° 4

Eseguendo degli esperimenti eseguendo l'algoritmo su singole parole abbiamo notato che le parole inserite nella lista con edit distance minima avevano tutte lunghezza molto simile alla parola esaminata. Si ordina il dizionario come indicato nell'approccio 3. Data la proprietà (*) si fa interrompere il ciclo quando è verificata l'ipotesi.

→ Vantaggi:

- il programma esegue in un tempo che è minimo tra gli esperimenti fatti

Nota: esperimenti ulteriori hanno denotato che, in alcuni casi, la lista delle parole restituita da questo approccio non era la stessa di quella dei primi 3, che è trivialmente corretta. Questa scarsa stabilità di questo approccio ha fatto sì che fosse mantenuto l'approccio n°3.

Proprietà:

Data una stringa X non vuota e un array di stringhe Y[] con le seguenti caratteristiche:

- *length(Y) = m finito e > 0*
- *ogni stringa in Y compare una e una sola volta*
- *le stringhe in Y sono ordinate per lunghezza crescente, poi alfabetico*

Vale la seguente proprietà: