Relazione 2 ASD

Mana Alessio, Perlo Giacomo, Sanino Fabrizio

Aprile 2021

1 Scelte implementative per l'ottimizzazione

Per implementare la strategia di Programmazione Dinamica come descritta nel paragrafo 15[1], e di **Memoization**, si è scelto di implementare una **Hash Table**.

L'**Hash Table** mantiene al suo interno coppie *chiave, valore* dove la *chiave* $\in \mathbb{N}$ è ottenuta concatenando le due stringhe di cui si vuol calcolare l'*edit distance*, mentre il *valore* è rappresentato dalla distanza stessa (calcolata con l'edit distance) fra le due.

L'**Hash Table** utilizzata è quella che sfrutta l'indirizzamento aperto con funzione di hashing a due livelli come descritto nel paragrafo 11.4[1].

In questo modo è possibile reperire un elemento all'interno della stessa con complessità $O(\frac{1}{\alpha}\ln\frac{1}{1-\alpha})$ se l'elemento è presente, $O(\frac{1}{1-\alpha})$ altrimenti,

dove α è il fattore di carico pari a $\alpha = \frac{n}{m} \leq 1$

(n=numero di elementi effettivamente presenti nell'Hash Table, m=capacità totale dell'Hash Table).

Abbiamo dunque deciso di mantenere un *fattore di carico* basso (0.15) per ottimizzare le operazioni di ricerca nel caso dell'ispezione senza successo, infatti mantenendo $\alpha \leq 0.15$ otteniamo $O(1,1765)$ (ovvero $O(1)$ ispezioni).

Per ottimizzare ulteriormente la velocità della correzione di una parola abbiamo adottato altre tre scelte implementative:

- In una ulteriore HashTable vengono salvate tutte le parole del dizionario iniziale, permettendo così di verificare se una parola è già corretta, in una sola operazione (essendo presente nel dizionario). Per ogni parola da correggere, quindi, verifichiamo se già presente nel dizionario ed in caso negativo la correggiamo prima di passare alla successiva;

- Se la parola da controllare non è stata trovata nel dizionario (dunque non è corretta) sarà impossibile trovare un edit distance pari a 0. Viene, dunque, calcolato l'edit distance: se si ottiene 1 significa che quell'edit distance è il valore migliore che poteva essere trovato. La ricerca viene interrotta senza calcolare ulteriori edit distance inutili;

- Un ulteriore controllo evita i confronti fra parole troppo distanti fra di loro. Il parametro controlla se la distanza tra la parola da correggere e la

parola del dizionario considerata risulta minore o uguale della metà della lunghezza della parola da correggere.
Ovvero:

$$\begin{cases} \text{Controllo la parola} & \text{Se} |p_{\text{dizionario}}| \leq \frac{|p_{\text{attuale}}|}{2} \\ \text{Salto la parola} & \text{Altrimenti} \end{cases}$$

Questo perché non avrebbe senso, ad esempio, cercare di controllare una parola molto lunga (10 lettere) con una parola composta da una sola lettera perchè ci saranno sicuramente parole più corte e quindi con un edit distance minore.

1.1 Aggiornamento del 29/06/2021

Abbiamo deciso di sistemare alcuni problemi sulla gestione della HashTable:

- La **HashTable** ora non deve più essere creata dall'utente e passata alla *edit_distance*, ma è direttamente la libreria stessa a crearla prima della esecuzione.
- La **HashTable** è generata di dimensioni proporzionali alle stringhe passate in input alla *edit_distance* (in precedenza era costante), permettendo all'esecuzione di essere più veloce ed efficiente.
- Generando una HashTable proporzionale ad ogni parola da confrontare viene rimosso il fattore di carico in quanto la HashTable non potrà mai essere riempita totalmente.

References

[1] Thomas H. Cormen. *Introduzione agli algoritmi e strutture dati.* McGraw-Hill, 2010.