Relazione esercizio 2

Componenti del gruppo
Carlino Mattia (matricola: 913397)
Ferreri Federico (matricola: 929655)
Fontana Anna (matricola: 912716)

Si consideri il problema di determinare la distanza di edit tra due stringhe (Edit distance): date due stringhe s1 e s2, non necessariamente della stessa lunghezza, determinare il minimo numero di operazioni necessarie per trasformare la stringa s2 in s1. Si assuma che le operazioni disponibili siano: cancellazione e inserimento.
Si implementi un'applicazione che usa la funzione edit_distance_dyn per determinare, per ogni parola w in correctme.txt, la lista di parole in dictionary.txt con edit distance minima da w.

Descrizione

Sono state implementate due versioni diverse per il calcolo dell'edit_distance tra due parole: nella prima è stata utilizzata la ricorsione mentre nella seconda si è utilizzata la ricorsione con la programmazione dinamica. Entrambi i metodi si basano sulla scomposizione ricorsiva di un problema in sottoproblemi e la successiva combinazione di essi.

Scelte implementative di edit_distance_dyn

Per quanto riguarda la versione dinamica dell'edit_distance sfruttiamo una matrice m * n, dove m rappresenta la lunghezza della prima stringa da confrontare, mentre n la lunghezza della seconda stringa da confrontare. Inizialmente tutte le celle della matrice vengono inizializzate a -1 per indicare che nessun risultato è ancora stato salvato. Ogni volta che si calcola il valore di un'edit_distance tra due sottostringhe questo viene inserito nella matrice. Se la funzione memo dovesse essere richiamata con degli stessi parametri con cui è stata chiamata in passato, è possibile ottenere il risultato con tempo costante O(1) andando semplicemente a leggere il valore presente nella matrice.

Scelte implementative del main

Per evitare di calcolare tutte le edit_distance tra tutte le parole nel file correct_me.txt e le parole del dictionary.txt si è scelto di controllare se la parola è contenuta nel dizionario (in questo caso la edit_distance ritornerebbe 0). Si utilizza una funzione binary_search_word di complessità$O(log\ (n))$, per controllare se una parola è presente nel dizionario: se lo è si ritorna 1 e non si calcola l'edit_distance, altrimenti ritorna 0 ed è necessario calcolare l'edit_distance.

Per aumentare ulteriormente l’efficienza, viene calcolata la differenza tra la parola da correggere e le parole del dizionario. Si confronta questa differenza con il minimo edit_distance calcolato fino a quel momento: se la differenza è maggiore l’edit_distance non verrà calcolato poiché ritornerebbe sicuramente un edit_distance maggiore.

Osservazioni

Quando i problemi non sono indipendenti tra loro, come in questo caso, la versione ricorsiva richiede tempi più lunghi per la risoluzione del problema. Durante l'esecuzione della funzione edit_distance senza l'utilizzo della programmazione dinamica, si è potuto osservare che l’esecuzione impiega un tempo eccessivamente elevato. Questo è dovuto al fatto che gli stessi sottoproblemi vengono eseguiti più volte e il numero di chiamate ricorsive è molto elevato. Infatti, senza l'utilizzo della programmazione dinamica, la complessità temporale è esponenziale.

Si vedano i seguenti esempi a dimostrazione delle precedenti osservazioni:

Edit_distance_classic vs edit_distance_dyn

Number of recursive calls (dynamic programming) between casa casa == 5
    EDIT_DYN == 0
Number of recursive calls (classic programming) between casa casa == 297
    EDIT_CLASS == 0

Number of recursive calls (dynamic programming) between matto mattia == 9
    EDIT_DYN == 3
Number of recursive calls (classic programming) between matto mattia == 2100
    EDIT_CLASS == 3

Number of recursive calls (dynamic programming) between metro sarta == 47
    EDIT_DYN == 8
Number of recursive calls (classic programming) between metro sarta == 603
    EDIT_CLASS == 8