Relazione esercizio 2

La programmazione dinamica – una dimostrazione emblematica delle sue potenzialità

Questa esercitazione ci ha insegnato quanto, in determinate circostanze, la programmazione dinamica influenzi sull'efficienza di un algoritmo, in fattispecie quelli ricorsivi. L'utilizzo di una sorta di "memoria" dove l'algoritmo memorizza il risultato ottenuto da una chiamata di se stesso in funzione di uno specifico input, influenza in modo sostanziale sull'efficienza dell'algoritmo stesso. Nello specifico caso di questo esercizio, la funzione edit_distance che non sfrutta la programmazione dinamica invoca un'esplosione di chiamate ricorsive, anche per stringhe relativamente corte. Notiamo però che le chiamate ricorsive spesso hanno come input gli stessi dati che altre chiamate ricorsive precedenti hanno trattato. Analizzandole, ci siamo accorti che in realtà i dati che edit_distance necessita per portare a termine la sua computazione date due stringhe s1 e s2 sono pari a (lunghezza della stringa s1 + 1) * (lunghezza della stringa s2 + 1), decisamente inferiore alle chiamate ricorsive che vengono invocate dalla funzione.

Stringa s1 = "ciao"

Stringa s2 = "bah"

	"ciao"	"iao"	"ao"	"o"	/
"bah"	<"bah","ciao">	<"bah","iao">	<"bah","ao">	<"bah","o">	<"bah", / >
"ah"	<"ah","ciao">	<"ah","iao">	<"ah","ao">	<"ah","o">	<"ah", / >
"h"	<"h","ciao">	<"h","iao">	<"h","ao">	<"h","o">	<"h", / >
/	< / ,"ciao">	< / ,"iao">	< / ,"ao">	< / ,"o">	< / , / >

Rappresentazione di tutte le possibili configurazioni di input su cui edit_distance("ciao", "bah") richiama se stessa per calcolare il risultato finale.

Da questa rappresentazione notiamo che l'input delle varie chiamate ricorsive differisce sostanzialmente dalla lunghezza rimanente delle stringhe s1 e s2. Da qui deriva la scelta di utilizzare una matrice di editDistanceMemory***** come memoria, ove nella coordinata ***[x][y]*** verranno memorizzati i valori delle tre variabili (rispettivamente: dNoOp, dInsertion, dDelete) ottenuti dalla chiamata ricorsiva su s1 con lunghezza ***x*** (quindi considerando solo le ultime ***x*** lettere della stringa iniziale), in egual modo con ***y*** intendiamo le ultime ***y*** lettere di s2. Facendo in questo modo la complessità dell'algoritmo si riduce a O(s1.length * s2.length), ottimale per il tipo di problema che stiamo trattando.

Applicazione - correggere un testo con edit_distance

Per individuare le parole errate nel file "correctme.txt" abbiamo utilizzato il dizionario insieme alla funzione edit_distance_dyn. L'individuazione della parola errata avviene nel seguente modo:

- Si sceglie la parola candidata dallo stream di "correctme.txt"
- Si utilizza edit_dinstance_dyn (s1,s2) dove s1 è la parola candidata a essere errata, s2 è ogni parola del dizionario
- Si memorizzano i risultati di edit_distance_dyn in un vettore dinamico denominato editDistance
- Si cerca il valore minimo memorizzato in editDistance: se il minimo è 0 allora significa che la parola analizzata è presente nel vocabolario, altrimenti vengono stampate a schermo tutte le parole con edit_distance pari al valore minimo ricavato precedentemente.

Dal test effettuato è emerso che, bensì le parole risultanti siano effettivamente simili a quella corretta, non sempre lo erano anche sintatticamente. Per esempio, nel caso di "cinqve" il risultato è stato "cinque" e "cive", cinque risulta quindi la parola effettivamente ricercata. Anche nel caso di "domandrono" il risultato "domandarono" risulta corretto all'interno del contesto. Stesso discorso per "vuolessi" (risultato = "volessi"), e per altre tante parole lessicalmente scorrette. In tutti questi casi, l'algoritmo ha dato esito positivo per risolvere questo problema, però in altri casi, no. L'algoritmo, analizzando parola per parola senza tener conto del contesto in cui queste sono situate, ha trovato parole che effettivamente rispettavano la definizione di edit_distance minimo, ma questo non implica che nel contesto in cui queste sono situate possono effettivamente essere corrette. Per riportare un esempio lampante di questa problematica, la parola "squola" all'interno della frase "Quando andai a squola mi domandarono come volessi essere da grande." da come risultato "suola", che corrisponde ad una cancellazione (ovvero edit_distance = 1), mentre la parola ricercata è "scuola", che corrisponde ad una cancellazione più un inserimento (ovvero edit_distance = 2), facendo cadere quindi il presupposto iniziale dove le parole con edit_distance minore erano le parole effettivamente corrette. I risultati ottenuti hanno dimostrato che per risolvere un problema come quello proposto non è condizione sufficiente analizzare le parole

solo a livello lessicale, ma è necessario tener conto del contesto in cui queste sono situate, dovendo quindi fare un'analisi ulteriore, quella sintattica.

*****Inizialmente abbiamo utilizzato una matrice di interi per memorizzare il risultato ottenuto dalle chiamate ricorsive (Nella cartella esercizio2/"esercizio2 - versione con matrice di interi" c'è questa implementazione a solo scopo dimostrativo). Questa sembrava la soluzione più semplice ed ottimale da implementare, ma ci siamo accorti che un test degli utest che abbiamo scritto appositamente falliva in quanto dava un risultato sbagliato.

In particolare, ci stiamo riferendo al test su sottostringhe interne (la funzione che testa questo caso si chiama editDistanceDynamicSubstringTest) dove nel caso di "TE" e "aTEa" da come risultato 3 invece di 2, nel caso di "TESTA" e "stringTESTAstring" da come risultato 16 invece di 12.

Quindi abbiamo deciso di implementare una matrice di oggetti "editDistanceMemory" che usiamo per memorizzare le singole variabili e non il risultato finale della funzione; in questo modo abbiamo ottenuto i risultati aspettati.