Commento all'es. 1

Anche se non richiesto dalle specifiche, per una maggior modularizzazione sono state realizzate 2 funzioni `leggiMatrice` e `scriviMatrice`. La funzione `leggiMatrice` rispetta le specifiche date nell'esercizio 2 del Lab. 3 (ritorna il numero di righe e di colonne effettivamente lette della matrice come parametri "by reference").

L'input viene fatto dal `main` leggendo una riga e, al suo interno, utilizzando esclusivamente formati `%s` per semplificare la gestione del formato e del carattere a-capo.

La funzione `ruotaMatrice` utilizza direttamente la `ruotaVettore` se si sta ruotando una riga, mentre nel caso si ruoti una colonna, prima la salva in un vettore temporaneo, poi la ruota con `ruotaVettore` ed infine salva il risultato della rotazione nella matrice. Questo a causa dell'organizzazione row-major (le caselle di una riga sono contigue in memoria, quindi utilizzabili come vettore, mentre le caselle di una colonna non lo sono).

Nella funzione `ruotaVettore` si osservi l'uso dell'operatore % per ridurre il numero di rotazioni `P` a un valore non superiore alla dimensione del vettore.

Si propongono 2 versioni della funzione `ruotaVettore`:

1. la rotazione di `P` posizioni è realizzata mediante `P` iterazioni ciascuna delle quali esegue uno scalamento opportuno (costo in tempo O(P*N). Lo scalamento di una posizione viene fatto (nella direzione opportuna) salvando la casella ad una estremità, scalando tutte le celle di una posizione e ripristinando il dato salvato all'altra estremità.
2. la variante `ruotaVettore2` è più efficiente in tempo (O(N)) al costo di un vettore in più (sovradimensionato). Lo spostamento viene fatto direttamente di `P` posizioni, ma questo richiede di salvare in un vettore temporaneo `P` valori (anziché uno solo come nel caso precedente).

Commento all'es. 2

La soluzione si basa su due fasi distinte:

- caricamento del dizionario in un vettore di `struct` (tipo `entry`)
- lettura del testo riga per riga, con ricerca, sostituzione e scrittura immediata sul file di uscita.

Si evita quindi di immagazzinare il testo in una struttura dati interna, limitandosi quindi a un vettore di caratteri (`riga`).

La generica riga viene elaborata percorrendo ognuno dei suoi caratteri e verificando se si tratta dell'inizio di una delle parole nel dizionario (le si provano tutte finché non sono terminate oppure si trova una corrispondenza). In caso affermativo, si scrive su file il codice sostitutivo (e si avanza in riga al termine della parola, altrimenti si scrive direttamente il carattere).
Il confronto tra una parola del dizionario e la sottostringa di riga iniziante all'i-esimo carattere viene proposto in due varianti:

- in una si realizza direttamente (nella funzione `confronta`, che non richiede uso di puntatori) il confronto carattere per carattere tra le stringhe
- in una seconda variante (`confrontaPunt`) si utilizza la funzione di libreria `strncmp`, con la quale è tuttavia necessario identificare la sottostringa di riga mediante aritmetica dei puntatori (`riga+inizio`).

Si noti che, con utilizzo leggermente più avanzato di aritmetica dei puntatori, si sarebbe potuto utilizzare la funzione di libreria `strstr`, che svolge direttamente la ricerca di una sottostringa in una stringa.

Si noti infine che, come scelta alternativa, si sarebbe potuto organizzare la ricerca con un criterio alternativo: per ogni parola nel dizionario, cercane tutte le occorrenze in riga, effettuando le opportune sostituzioni. Pur trattandosi di soluzione equivalente alla precedente, questa richiede di manipolare la stringa in riga, effettuandovi le sostituzioni, in quanto, per ogni parola in dizionario, occorre ricominciare a cercare dall'inizio di riga.

Commento all'es. 3

Il problema viene risolto utilizzando funzioni di input e di output di matrice (la seconda non è richiesta ma può essere utile per debug) e una funzione che calcola e scrive direttamente su file di uscita la matrice risultato. Come alternativa si sarebbe potuta generare internamente una seconda matrice (risultato): non è possibile infatti trasformare direttamente la matrice originale, in quanto servono tutti i dati iniziali per calcolare i valori (le somme) all'interno della matrice risultato.

Si propongono tre versioni della funzione che calcola le somme dei dati a distanza R: la `risolvi1` e la `risolvi2` sono due varianti, rispettivamente con costrutti `for` e `while`, dello stesso algoritmo, che percorre iterativamente i 4 lati di un quadrato (caselle a distanza R dalla caselle (r,c)).
La funzione `risolvi3`, invece, percorre con una doppia iterazione tutte le caselle (anche quelle interne) del quadrato, filtrando quelle a distanza R.