Esercizi di programmazione

Di Giuseppe Martinelli

Matr: 7093926

Esercizio N 1:

"*Scrivere una funzione che calcoli le seguenti caratteristiche di un testo (sequenza di caratteri). Si faccia riferimento all'Esercizio 2.1 per la definizione dei concetti teorici.*

1. Istogramma della frequenza delle 26 lettere;

2. Dato m ≥ 1 in input, distribuzione empirica degli m-grammi (blocchi di m lettere);

3. Dato m ≥ 1, indice di coincidenza ed entropia della distribuzione degli m-grammi.

Illustrare i risultati ottenuti impiegando la funzione sul primo capitolo di Moby Dick (H. Melville, 1851), per m = 1, 2, 3, 4."

Nota breve:

Questo esercizio viene risolto nel codice Python freqAnalysis.py. All'interno della cartella del codice si trovano i seguenti file:

1. text.txt

   Corrisponde al testo analizzato dall'esercizio, in questo caso è il primo capitolo di Moby Dick in lingua inglese. La presenza di questo file è obbligatoria

2. result.txt

   File di testo contente i risultati dell'analisi dell'entropia e della coincidenza del file di testo a seconda della dimensione di *m*

Durante l'esecuzione del codice verranno create diverse immagini in formato .png che corrispondono agli istogrammi delle distribuzioni dei m-grammi. La costruzione degli istogrammi con m = 3,4,5 **ci mette un po' di tempo**; quindi bisogna avere un po' di pazienza

Svolgimento:

Il punto 1) viene risolto dalla funzione *createHistogram(lines)* la quale prende in input una stringa contenete il testo letto dal file **text.txt.** Ripulisce il testo letto chiamando la funzione *preprocessText(lines)* e per ogni lettera dell'alfabeto conta il numero di occorrenze della lettera stessa all'interno del testo

Il punto 2) invece viene risolto dalla funzione *distributionMblocks(lenght,lines)* la quale prende in input un numero m che indica la grandezza dei m-grammi da ricercare ed il testo stesso. All'interno della funzione si divide il testo in m che sono tutti i potenziali m-grammi possibili (Nel caso in cui la lunghezza del testo diviso m dia resto diverso da 0 allora si aggiungono caratteri "." alla fine) e si studia la frequenza degli m-grammi unici *rispetto* al numero totale di tutti gli m-grammi possibili.

Alla fine, la funzione ritorna: un dizionario contenente la coppia m-grammo – frequenza, il numero dei m-grammi possibili ed infine il testo (ripulito sempre nella funzione con la procedura *preprocessText(lines)*

Il punto 3) viene risolto dalla funzione *coincidenzaEntropiaMblocchi(block_dic,block_num,text,lenght)* dove viene passato in input: il dizionario, il numero dei blocchi, il testo (tutti ottenuti nel punto 2.) ed infine la lunghezza di m. La coincidenza per un blocco X viene calcolata come:

$$F(X) * (F(X) - 1) / \text{Numero blocchi} * (\text{Numero blocchi} - 1)$$

Dove F(X) rappresenta il numero di occorrenze del blocco X nel testo. Nello stesso metodo viene calcolata l'entropia secondo questa formula:

$$(-1) * (Freq(X) * \log_2(Freq(X)))$$

Dove Freq(X) rappresenta la frequenza di X nel testo (cioè il numero di occorrenze di X fratto la lunghezza del testo, questo valore è contenuto nel dizionario **block_dic**)

Esercizio N 2:

"Programmare una funzione che permetta all'utente di cifrare e decifrare con il cifrario di Hill, e di forzare un ciphertext tramite l'attacco known plaintext."

Svolgimento:

Questo esercizio viene svolto dal codice python main.py contenuto nella cartella Hill, il codice legge da un file testuale **text.txt** il messaggio da cifrare con una chiave K già fornita a priori (di cui comunque se ne controlla l'invertibilità modulo 26 grazie al metodo *generateKey()*). Successivamente, il testo viene diviso in trigrammi con la funzione *divideTextInBlocks* e viene cifrato con la funzione *encryption* la quale moltiplica ogni blocco del testo con la chiave ed effettua il modulo 26.

La decifratura del ciphertext viene effettuata attraverso la funzione *decryption* la quale:

1. Calcola l'inverso della chiave K
2. Verifica che la chiave inversa della chiave sia corretta, cioè moltiplica K con K**-1 e verifica se si ottiene la matrice identità (funzione *isIdentity*)
3. Moltiplica un blocco di ciphertext con la chiave inversa e applica il modulo 26

Infine l'attacco viene effettuato attraverso la funzione *attacckHill(m,key)* si noti che l'attacco al cifrario di Hill è di tipo *known plaintext* dove un attaccante si costruisce delle coppie < Plaintext – E[Plaintext]>, cioè conosce delle coppie plaintext ed il loro corrispettivo cifrato. Per simulare questo caso, passiamo alla funzione anche la chiave per permettere di poter cifrare queste coppie che ci creiamo. Nonostante ciò, **l'attaccante non conosce la chiave, il passaggio della chiave è stato fatto per permettere la cifratura di quei blocchi**; un procedimento alternativo sarebbe stato di prendere dal ciphertext il blocco i scelto casualmente dal testo (dal momento in cui i blocchi hanno lo stesso ordine sia nel ciphertext che nel plaintext). Nella funzione *createRandomMcoppie* vengono scelte m coppie di testo random che vengono cifrate (appunto per simulare quella condizione precedentemente descritta), si verifica che le coppie di plaintext scelte siano invertibili modulo 26 (cioè il MCD tra il determinante della matrice di plaintext e 26 sia uguale 1, le funzioni che effettuano l'MCD ed il determinante sono *GCD* e *det*) e se lo sono allora vengono moltiplicate con le coppie cifrate corrispondenti modulo 26 ottenendo infine così la chiave.