1

ESERCIZI DI PROGRAMMAZIONE

1.1 3.1 ANALISI DELLE FREQUENZE DI UN TESTO

La soluzione di questo esercizio è presente nel file *analisi_frequenze.py*.
Ho implementato il tutto in un'unica classe, **AnalisiFrequenze**, la quale, nel costruttore, recupera il file inerente al testo *Moby Dick* e si occupa di trasformare il testo secondo la filosofia *lowercase* e di rimuovere i caratteri superflui.
Dopodichè ho implementato i vari punti dell'esercizio:

- 1: nel metodo *histogram*, in primo luogo, ho creato un vettore contenente tutte le lettere dell'alfabeto, dopodichè, tramite il metodo *_get_occurrences*, ho contato tutte le occorrenze delle 26 lettere nel testo di *Moby Dick*. Infine, ho passato il vettore e le occorrenze al metodo *_plot* che, tramite la libreria *matplotlib* crea l'istogramma desiderato.

- 2: nel metodo *m_histograms*, innanzitutto, ho creato, tramite il metodo privato *_create_dictionary_for_occurrences*, un dizionario contenente come chiavi gli m-grammi, mentre come valori il loro numero di occorrenze nel testo. Successivamente, imposto un numero massimo (n) di elementi da mostrare in un unico plot, altrimenti ci sarebbero stati elementi sovrapposti e poi creo un nuovo dizionario con i primi n valori ordinati in maniera descrescente, calcolando successivamente la distribuzione empirica degli m-grammi. Infine, passo al metodo *_plot* le chiavi e i valori del nuovo dizionario per creare il plot.

- 3: tramite il metodo *coincidence_index* calcolo l'indice di coincidenza degli m-grammi, per un m specificato. Il metodo fa questo: cicla tutti gli m-grammi salvati nel dizionario e per ogni m-gramma trovato, prende il suo valore, cioè il suo numero di occorrenze nel testo, e

lo usa per calcolare un valore. Una volta ciclati tutti gli elementi, vengono sommate tutte le occorrenze di ogni trigramma tra loro, tramite il metodo *_calculate_all_occurrences*, per poi applicare la formula degli indici di coincidenza, la quale è:

$$IC = \sum_{j} \frac{f_j(f_j - 1)}{N(N-1)}$$

.
Mentre per l'entropia di Shannon, ho creato il metodo *shannon_entropy*, il quale, in prima battuta, calcola le frequenze totali degli m-grammi, tramite il metodo *_calculate_all_occurrences*, per poi ciclare tutti gli m-grammi trovati e salvare in una variabile il valore ottenuto dal prodotto di $p(x) \log p(x)$. Una volta finito il ciclo, moltiplico il valore ottenuto per -1, completando la formula:

$$H(X) = -\sum_{x \in X} p(x) \log p(x).$$

.

1.2 CIFRARIO DI HILL

La soluzione di questo esercizio è presente nel file *main.py*, nel quale vengono riportati i passi svolti da Hill, per quanto riguarda la cifratura e la decifratura, e dall'attaccante, per quanto riguarda l'attacco.
Ho deciso di implementare i due principali soggetti tramite la programmazione orientata agli oggetti, come segue:

- **Hill**: come primo step, quando viene creato l'oggetto, vengono richiesti 2 input da parte dell'utente, i quali sono il plaintext e la dimensione dei blocchi; tale richiesta viene gestita dal metodo *_check_input*. Dopodichè viene richiamato il metodo *encryption*, il quale si occupa, in prima battuta, di rimuovere i caratteri superflui dal PT e successivamente di rimpiazzare i caratteri con la relativa codifica in Z_{26}. In seguito, tramite il metodo *_generate_k_matrix*, viene generata randomicamente una matrice, contenente valori in Z_{26} e restituita solo se invertibile.
  Infine, viene generato il CT a partire dall'equazione:

  $$C = KxP$$

  Segue la fase di decryption mediante il medesimo metodo, il quale calcola, in primo luogo, la matrice K inversa tramite la formula di

Cramer:
$A^{-1}[i][j] = (-1)^{i+j} det(A_{ji})/det(A)$, dove $1/det(A)$ è da intendersi come $det(A)^{-1} mod 26$. Mentre, in secondo luogo, calcola il PT utilizzando la formula:

$$P = K^{-1} x C$$

Ovviamente, tutte le operazioni di cui ho parlato, si intendono modulo 26, dato che ho lavorato sull'alfabeto completo che comprende le 26 lettere.

- **Attaccante**: l'oggetto viene creato sapendo il PT, CT e la dimensione dei blocchi. Successivamente, viene eseguito il metodo *attack* che si occupa di sferrare l'attacco. Questo metodo, inizialmente, richiama il metodo *_create_block_matrix*, il quale trasforma sia il PT, che il CT, in una matrice a blocchi di dimensione fissata (quella saputa), questo perchè, col proseguo del metodo *attack*, tramite la libreria *itertools*, vengono create tutte le possibili combinazioni di PT/CT. Successivamente, vengono provate, una ad una, le possibili combinazioni per ottenere, in primo luogo, la matrice P^{*-1}, la quale viene creata solo se il suo determinante è coprimo con 26. Dopodichè, viene calcolata la matrice K, tamite la formula $K = C^* x P^{*-1}$, e viene generato il CT tramite la matrice K ottenuta ed il PT 'rubato' ad Hill, questo perchè, se il CT generato è uguale al CT che ha ottenuto da Hill, allora è riuscito a trovare la chiave K per decifrare i messaggi. Infatti, se i due CT sono uguali, vengono stampati a video tutti i valori che hanno permesso di ottenere la matrice K.

N.B.: un'ulteriore possibile implementazione sarebbe quella della creazione di una superclasse (Person, per esempio) per Hill e l'attaccante, dato che condividono metodi comuni.

2

ESERCIZI DI APPROFONDIMENTO

2.1 2.2 INDICI DI COINCIDENZA

- **A**: noto che $f_i = \sum_{j=1}^{n} Y_j$ dove Y_j è la variabile Bernoulli che assume il valore 1 se $x_j = i$, 0 altrimenti, sfruttando la linearità del valore atteso, è possibile dimostrare che:
  $E[f_i] = E[\sum_{j=1}^{n} Y_j] = E[Y_1] + E[Y_2] + ... + E[Y_n] = 1 \cdot p_1 + 0 \cdot (1 - p_1) + 1 \cdot p_2 + 0 \cdot (1 - p_2) + ... + 1 \cdot p_n + 0 \cdot (1 - p_n) = \sum_{j=1}^{n} p_i = p_i \sum_{j=1}^{n} 1 = np_i = E[f_i]$

- **B**: sapendo che la definizione della varianza è la seguente: $var(f_i) = E[(X - E[X])^2]$, allora, svolgendo i calcoli, è possibile dimostrare che $E[f_i^2] = var(f_i) + E[f_i]^2$.
  Infatti:
  $var(f_i) = E[(f_i - E[f_i])^2] = E[f_i^2 - 2f_iE[f_i] + E[f_i]^2] = E[f_i^2] - 2E[f_i]E[f_i] + E[f_i]^2 = E[f_i^2] - 2E[f_i]^2 - E[f_i] = E[f_i^2] - E[f_i]^2 \rightarrow$
  $E[f_i^2] = var(f_i) + E[f_i]^2$

- **C**: supponendo che i caratteri x_j siano estratti in maniera indipendente ed identicamente distribuita, è possibile notare che f_i è una distribuzione Binomiale, il suo valore atteso è $E[f_i] = np_i$ e che la sua varianza è $var(f_i) = np_i(1 - p_i)$. Allora, sfruttando queste accortezze e andando a sostituire i valori nel punto precedente, si ottiene che:
  $E[f_i^2] = var(f_i) + E[f_i]^2 = np_i(1 - p_i) + (np_i)^2 = np_i - np_i^2 + n^2p_i^2$.
  Perciò:
  $E[f_i(f_i - 1)] = E[f_i^2] - E[f_i] = np_i - np_i^2 + n^2p_i^2 - np_i = n(n - 1)p_i^2$

- **D**: ricordando che $I_c(x) = \sum_{i \in Z_{26}} \frac{f_i}{n} \frac{f_i - 1}{n - 1}$ e sfruttando il risultato ottenuto al passo precedente, è possibile dare una formula esatta a $E[I_c(x)]$.
  Infatti:

$E[I_c(x)] = E[\sum \frac{f_i}{n}\frac{f_i-1}{n-1}] = \sum E[\frac{f_i}{n}\frac{f_i-1}{n-1}] = \frac{\sum E[\frac{f_i}{n}\frac{f_i-1}{n-1}]}{n(n-1)} = \frac{\sum np_i^2(n-1)}{n(n-1)} = \sum p_i^2$

- **E**: supponendo che **q** sia lo shift circolare di *k* posizioni ($0 \leqslant k \leqslant 25$) che meglio approssima il vettore **p** delle probabilità, caratteristico della lingua Inglese, si vuole dimostrare che:
  $\forall j \in Z_{26} < p, q > \geqslant < p, q_j >$
  quindi, in sostanza, si vuole dimostrare che il prodotto scalare tra **p** e **q** è maggiore o uguale al prodotto scalare che si otterrebbe moltiplicando **p** con qualunque altro shift *j* del vettore caratteristico delle probabilità.
  Supponendo che i due vettori, **p** e **q**, abbiano la stessa lunghezza *m*, si ricorda che:
  - $- : < p, q >= \sum_{j=0}^{m-1} p_j q_j$
  - $- : \|p\|_2 = \sqrt{\sum_{j=0}^{m-1} p_j^2}$

  Inoltre, dato che per semplicità si suppone che la relazione $\mathbf{p} \approx \mathbf{q}$ sia una relazione di uguaglianza, allora si ha che:
  $< p, q >=< p, p >= \sum_{j=0}^{m-1} p_j^2 = \|p\|_2 \cdot \|p\|_2$
  Servendosi della disuguaglianza di Cauchy-Schwarz, definito lo shift di *j* posizioni q_j del vettore delle probabilità empiriche, si ha che:
  $< p, q_j > \geqslant \|q_j\|_2 \cdot \|p\|_2$
  Ricordando che la norma euclidea di un vettore è costante per ogni shift, questo significa che è costante sia per ogni q_j, che per **q**, inoltre, queste norme sono tutte uguali fra loro. Sicchè, applicando queste considerazioni a quanto richiesto, si ottiene che:
  $< p, q >= \|q\|_2 \cdot \|p\|_2 = \|q_j\|_2 \cdot \|p\|_2 \geqslant < p, q_j >$

2.2 UN CRITTOGRAMMA VIGENÈRE

Per risolvere questo problema, ho creato un file Python *vigenere.py*, il quale contiene la classe Vigenere, che rappresenta tutto quello che bisogna fare per decifrare il crittogramma e il metodo *main*, in cui sono richiamati i passi dell'esercizio. Inoltre, ho utilizzato sia gli indici di coincidenza, che il test di Kasiski.
Inizialmente, quando l'oggetto viene creato, viene salvato come attributo il CT, questo perchè mi è utile richiamarlo in altre parti del codice.

- 1: successivamente, viene richiamato il metodo *repetitions*, il quale si occupa di creare in primis un dizionario che ha come chiavi tutti i possibili trigrammi, con sovrapposizione, nel testo, e come valori tutte le loro ripetizioni. Inoltre, viene creato un altro dizionario, il quale ha tutte le chiavi del primo, solo se tali chiavi hanno più di un valore, ovvero se tali trigrammi hanno più di una ripetizione nel testo. Alla fine, stampa a schermo ogni trigramma ed il suo numero di ripetizioni e ogni trigramma con il suo numero di ripetizioni solo nel caso se le ripetizioni sono maggiori di 1.
  In seguito viene richiamato il metodo *distances*, il quale crea un dizionario contenente, per ogni trigramma che ha più di una ripetizione, tutte le posizioni delle successive ripetizioni e un altro dizionario contenente le medesime chiavi, ma come valori ha le distanze di ogni trigramma dal primo trigramma trovato. Infine, stampa a schermo le distanze di ogni trigramma.
  Dopodichè, viene chiamato il metodo *m_values*, il quale, in prima battuta, crea un dizionario contenente i trigrammi e il loro rispettivo MCD ottenuto dalle distanze. Successivamente, vengono salvati in una lista tutti gli MCD, senza ripetizioni, e stampati a schermo, questo perchè possono indicare il valore possibile della chiave.

- 2: nel *main* viene richiamato il metodo *correct_m_and_coincidences_indexes*, nel quale vengono provate le chiavi, definite in un range (2,16), dove 2 è incluso e 16 no, su possibili divisioni del testo a seconda della chiave[1], calcolati i rispettivi indici di coincidenza, calcolata la devizione d'errore, ovvero lo scarto quadratico medio, tramite la frequenza delle lettere media, e salvati, se la deviazione trovata è inferiore alla deviazione definita in precedenza, i migliori valori degli indici, la chiave utilizzata per trovare questi valori e sostituito il valore massimo della deviazione con quello trovato attualmente, affinchè si ottengo lo scarto minimo.
  Infine, viene stampata la miglior chiave trovata con gli indici di coincidenza e i rispettivi valori degli indici.

- 3: viene richiamato il metodo *calculate_key*, il quale, per ogni testo suddiviso secondo la chiave, calcola le frequenze delle lettere presenti in quella riga e ricerca lo shift migliore che massimizza il

1 Esempio: se la chiave è n = 2, allora viene creato un vettore di 2 celle, dove nella prima sono presenti i caratteri che partono dalla posizione 0 e vengono incrementati di n, mentre nella seconda partono da 1 e vengono incrementati di n, come se fosse una matrice n x (lunghezza_CT/n).

prodotto scalare tra le frequenze inglesi e le frequenze delle lettere presenti nel testo. Stampa a video la lettera della chiave, il relativo prodotto scalare e la chiave generale ottenuta.
In seguito, viene richiamato, nel *main*, il metodo *decrypt*, il quale si occupa di decifrare il CT utilizzando la chiave ottenuto al passo precedente e stampare a video il PT.