Claudia Landi
Matricola: 7149573
Anno Accademico: 2023-2024

Sicurezza dei Dati e Privacy - Relazione primo set di esercizi

1 Esercizi di programmazione

1.1 Esercizio 3.1

Il codice relativo all'esercizio si trova nel file *AnalisiFrequenze.py*. Per utilizzarlo è sufficiente eseguire il main, i risultati dei punti dell'esercizio verranno mostrati in sequenza.
Eseguendo il codice, innanzitutto viene aperto il file *moby.txt* in modalità di lettura ("r") con l'encoding UTF-8, leggendo tutto il suo contenuto, che verrà memorizzato nella variabile *text*.

1. Per ottenere l'istogramma delle frequenze delle lettere è stata creato il metodo *frequency_histogram*, che prende in input il testo da analizzare. Come prima cosa viene chiamata la funzione *clean_text*, che rimuove dal testo tutti gli spazi e gli elementi non alfabetici (selezionati tramite il comando *isalpha*) e trasforma i caratteri in minuscolo con il comando *lower()*. Dopodiché viene utilizzata la classe Counter dal modulo collections per contare quante volte ciascuna lettera appare nel testo pulito. Questo crea un dizionario in cui le chiavi sono le lettere e i valori sono le rispettive frequenze. Le lettere e le loro frequenze vengono poi estratte dal dizionario e ordinate in ordine alfabetico, per assicurare che le lettere siano disposte in ordine corretto sull'istogramma. Per creare il grafico è stata utilizzata la libreria *matplotlib*, disponendo le lettere saranno sull'asse x e le frequenze sull'asse y. Ogni barra rappresenta quante volte una lettera specifica appare nel testo (figura 1). Il comando *plt.show()*, infine, mostra l'istogramma a schermo.

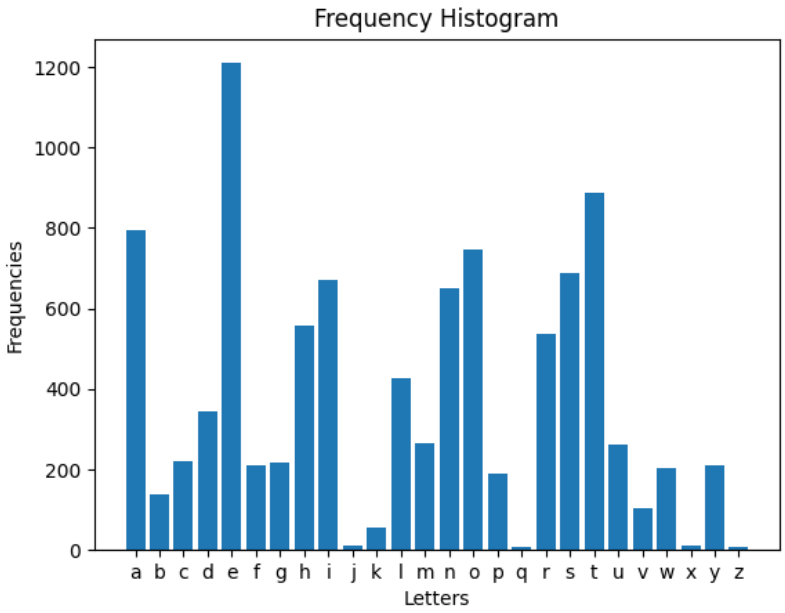

Figura 1: Istogramma delle frequenze delle 26 lettere.

2. Per determinare le distribuzioni empiriche degli m-grammi, per $m \geq 1$, è stato definito un ciclo for all'interno del main, così da iterare per ogni valore di m. Nel ciclo vene chiamato il metodo *mgram_distribution*, che prende in input il testo e il valore corrente di m. Dopo aver richiamato la funzione di pulizia del testo, viene nuovamente utilizzata la classe Counter, che stavolta restituisce un dizionario con gli m-grammi come chiavi e le frequenze come rispettivi valori. Il comando "*for i in range(0, len(cleaned_text) - m + 1, m)*" crea un loop che itera attraverso il testo pulito in passi di lunghezza m. Successivamente è stata calcolata la frequenza totale degli m-grammi, sommando tutte le frequenze nel dizionario. Questo ha permesso di ottenere un nuovo dizionario, *relative_frequencies* con le frequenze relative, calcolate dividendo la frequenza di ciascun m-gramma per la frequenza totale. Infine, in modo analogo al caso precedente i valori vengono estratti dal dizionario e mostrati in un istogramma, in cui vengono riportati solo i primi 50 elementi in ordine alfabetico, per migliorarne la leggibilità (esempio in figura 2).

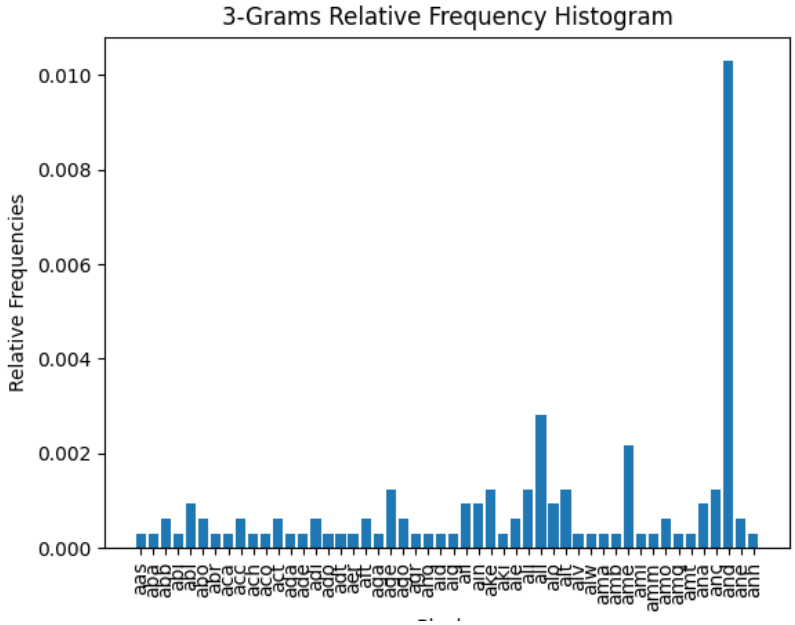

Figura 2: Istogramma delle frequenze relative dei trigrammi.

3. Infine per determinare gli indici di coincidenza e l'entropia della distribuzione degli m-grammi, sono stati sviluppati rispettivaente i metodi *coincidence_ index* e *shannon_ entropy*, che vengono richiamati dal main all'interno del ciclo for che itera per tutti i valori di m, per $m \geq 1$. Il primo prende in input il dizionario con le distribuzioni degli m-grammi, ne calcola il numero totale e infine applica la formula:

$$IC = \frac{\sum_i (f_i \cdot (f_i - 1))}{n \cdot (n-1)} \tag{1}$$

Dove:

$IC =$ Indice di Coincidenza
$f_i =$ Frequenza dell'i-esimo m-gramma
$n =$ Numero totale di m-grammi.

Per ricavare l'entropia, sempre all'interno del ciclo for nel main, viene richiamato il metodo *shannon_ entropy*, che prende in input il dizionario delle distribuzioni degli m-grammi, ne calcola il numero totale, e applica la formula dell'entropia di Shannon:

$$H = -\sum_i \left(\frac{f_i}{n}\right) \cdot \log_2 \left(\frac{f_i}{n}\right) \tag{2}$$

Dove:

$H =$ Entropia di Shannon
$f_i =$ Frequenza dell'i-esimo m-gramma
$n =$ Numero totale di m-grammi.

Blocco di Lunghezza	Indice di Coincidenza	Entropia di Shannon
1	0.06595143929297416	4.1631232707071
2	0.00777074960358351	7.6555226919834
3	0.00138496925644816	10.0516434364616
4	0.00031782235599842	10.7587829716046

Tabella 1: Indici di Coincidenza ed Entropia di Shannon per blocchi di diverse lunghezze.

I risultati ottenuti vengono stampati a video eseguendo il codice e sono riportati nella tabella 1.

1.2 Esercizio 3.2

La soluzione di questo esercizio è contenuta nel file *Hill.py*, di cui è sufficiente eseguire il main per ottenere i risultati richiesti.
Innanzitutto per permetere la cifratura e decifratura di un messaggio tramite il cifrario di Hill, è stato inserito nel main un frammento di testo tratto dalla poesia "Il corvo" di Edgar Allan Poe (1845). Inoltre è stato implementato il metodo *generate_square_matrix* che, dato in input un intero *matrix_size*, genera randomicamente una matrice quadrata ed invertibile modulo 26, di tale dimensione. Tale matrice verrà utilizzata come chiave k per il cifrario.

Prima della cifratura viene chiamato il metodo *prepare_message*, che rimuove dal testo gli spazi e gli elementi non alfabetici e trasforma le lettere in minuscolo, tramite la funzione *clean_text*. Inoltre questo metodo prende in input la dimensione della matrice chiave e suddivide il testo in blocchi di tale dimensione, restituendo una lista che li contiene, sotto forma di array Numpy. Eventuali porzioni di testo rimanenti vengono scartate. Si nota che grazie al comando *ascii_num = ord(char) - 97*, ogni lettera del testo viene converita nel corrispondente valore ASCII, per permettere in seguito di cifrare il messaggio utilizzando le operazioni in aritmetica modulare, come previsto nel cifrario di Hill.
Il metodo *encrypt* si occupa di cifrare il plaintext restituito da *prepare_message*, con la chiave generata da *generate_square_matrix*, che viene passata come argomento. Ciascun blocco cifrato viene ottenuto dalla moltiplicazione modulo 26 del blocco in chiaro con la matrice k, eseguita tramite il metodo *dot()* della libreria Numpy. Ciascun blocco cifrato è stato poi trasformato nella stringa corrispondente, riconvertendo in caratteri i valori ASCII ottenuti dopo la cifratura. La funzione *encrypt* restituisce quindi il testo cifrato sotto forma di una stringa, ottenuta dalla concatenazione dei vari blocchi cifrati.

Per permettere la decifratura, il testo criptato viene nuovamente passato a *prepare_message*, che lo restituisce sotto forma di blocchi di valori ASCII di dimensione della chiave. I blocchi di testo criptati vengono passati alla funzione *decrypt*, che inverte la matrice chiave k modulo 26, dopodiché richiama la funzione *encrypt* passando i blocchi cifrati e la matrice inversa, in modo da applicare le stesse operazioni in aritmetica modulare usate

per la cifratura, ma con la matrice inversa. Per invertire la matrice viene richiamata la funzione *matrix_mod_inverse*, che come prima cosa controlla che la matrice sia invertibile modulo 26. Se il controllo ha successo, converte la matrice Numpy in un oggetto di matrice Sympy. Questo è necessario perché Sympy offre una funzione *inv_mod* per calcolare l'inversa modulo n. Successivamente al calcolo, l'inversa della matrice è rappresentata come un oggetto Sympy: per ottenere nuovamente una matrice Numpy, il codice converte l'oggetto Sympy in una matrice Python utilizzando *.tolist()* e quindi utilizza *.astype(int)* per assicurarsi che tutti i valori siano interi anziché numeri razionali. Infine la matrice inversa viene restituita al chiamante. Eseguendo il codice si può notare come la stringa ottenuta sia equivalente al messaggio originale, privato dei caratteri non alfabetici e riportato in minuscolo, segno che le funzioni di cifratura e decifratura funzionano correttamente.

Per portare a termine l'attacco known-plaintext, si suppone che l'attaccante disponga del messaggio in chiaro e di quello cifrato e che debba usarli per trovare la chiave. Tale attacco si basa sul fatto che, se si riesce ad ottenere una matrice quadrata composta da vettori di blocchi di testo in chiaro P, e tale matrice risulta invertibile, conoscendo la matrice dei relativi blocchi cifrati, C, è possibile ricavare la chiave k come prodotto matriciale tra l'inversa modulo 26 della matrice del testo in chiaro e quella del testo cifrato, ossia $K = P' \cdot C$. Per eseguire l'attacco, alla funzione *attack* responsabile di determinare la chiave, vengono quindi passati il messaggio originale e quello cifrato, insieme alla dimensione dei blocchi. Il metodo innanzitutto richiama *prepare_message* per ottenere i blocchi in chiaro e quelli cifrati, dopodiché li usa per creare delle matrici della stessa dimensione della matrice chiave, una formata da blocchi consecutivi di plaintext, l'altra da blocchi consecutivi di ciphertext, chiamate rispettivamente *p1* e *c1*. A questo punto la funzione tenta di richiamare *matrix_mod_inverse* per invertire *p1*, se il metodo restituisce un errore (ovvero *p1* non è invertibile modulo 26), tenta con un'altra combinazione di blocchi di plaintext e ciphertext. Se nessuna matrice *p1* è invertibile, non è possibile determinare la chiave e quindi il metodo restituisce il messaggio di errore "*Chiave non trovata.*". Se invece una delle matrici *p1* viene invertita correttamente, il metodo ricava la chiave k dalla molttiplicazione modulo 26 dell'inversa di *p1* con *c1*. A questo punto viene fatto un controllo per verificare che la decifratura funzioni correttamente con la chiave trovata, chiamando la funzione *decrypt* e passandogli il messaggio cifrato e la chiave trovata. Se il messaggio decifrato restituito da *decrypt*, *decripted_message*, è uguale al messaggio in chiaro di cui disponeva l'attaccante (nel codice: *message*), la funzione *attack* termina e restituisce la chiave trovata. Se la chiave trovata non è corretta, continua la ricerca con nuove matrici *p1* e *c1*. Eseguendo il codice si può notare come la matrice restituita da *attack*, quando questo ha successo, corrisponde con quella utilizzata per cifrare il messaggio, questo dimostra la correttezza del metodo sviluppato (Figura 3).

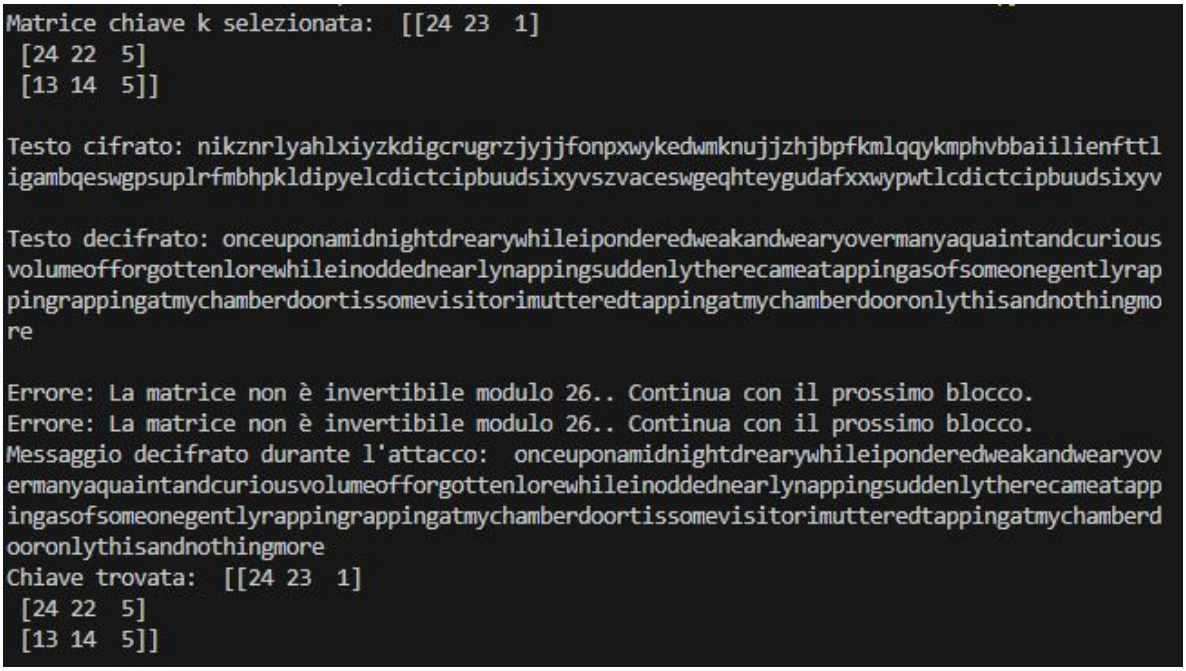

Figura 3: Esempio di risultati ottenuti dall'esecuzione del codice *Hill.py*.

2 Esercizi di approfondimento

2.1 Esercizio 2.2

a) Sia $i \in Z_{26}$ qualsiasi fissato. Notiamo che $f_i = \sum_{j=1}^{n} Y_j$, dove Y_j è la variabile Bernoulli che assume il valore 1 se $x_j = i$, altrimenti assume il valore 0. Sfruttando la linearità del valore atteso, possiamo dimostrare che $E[f_i] = np_i$.
Iniziamo considerando:

$$E[f_i] = E[\sum_{j=1}^{n} Y_j]$$

per la linearità del valore atteso possiamo scrivere:

$$E[f_i] = E[Y_1] + E[Y_2] + ... + E[Y_n] = \sum_{j=1}^{n} E[Y_j]$$

sostituendo con i valori della variabile Bernoulli si ottiene:

$$E[Y_j] = 1 \cdot p_i + 0 \cdot (1 - p_i) = p_i$$

per ciascuno degli n Y_j, ovvero, sostituendo nella sommatoria:

$$E[f_i] = \sum_{j=1}^{n} p_i = p_i \sum_{j=1}^{n} 1 = np_i$$

b) Dalla definizione di varianza si ha:

$$\begin{aligned} v(f_i) &= E[(f_i - E[f_i])^2] \\ &= E[f_i^2 + E[f_i]^2 - 2E[f_i] \cdot f_i] \end{aligned}$$

per la proprietà di linearità del valore atteso:

$$\begin{aligned} v(f_i) &= E[f_i^2] - 2E[f_i] \cdot E[f_i] + E[f_i]^2 \\ &= E[f_i^2] - 2E[f_i]^2 + E[f_i]^2 \\ &= E[f_i^2] - E[f_i]^2 \end{aligned}$$

abbiamo quindi ottenuto:

$$v(f_i) = E[f_i^2] - E[f_i]^2$$

isolando $E[f_i^2]$ si ottiene:

$$E[f_i^2] = v(f_i) + E[f_i]^2$$

c) Supponendo che i caratteri x_j siano estratti in maniera indipendente ed identicamente distribuita, possiamo notare che f_i segue una distribuzione binomiale. Sappiamo che il suo valore atteso è $E[f_i] = np_i$, mentre la varianza è $var(f_i) = np_i(1 - p_i)$.
Per la dimostrazione precedente:

$$E[f_i] = v(f_i) + E[f_i]^2$$

sostituendo i valori di varianza e valore atteso si ottiene:

$$E[f_i] = np_i(1 - p_i) + (np_i)^2 = np_i - np_i^2 + n^2p_i^2$$

questo termina la prima parte di dimostrazione.
Vogliamo ora dimostrare che:

$$E[f_i(f_i - 1)] = E[f_i^2] - E[f_i] = n(n - 1)p_i^2$$

eseguendo i calcoli e applicando la linearità del valore atteso si ottiene:

$$E[f_i(f_i - 1)] = E[f_i^2 - f_i] = E[f_i^2] - E[f_i]$$

sostituendo il risultato appena ottenuto e quello del punto (a):

$$\begin{aligned} E[f_i(f_i - 1)] &= np_i - np_i^2 - n^2p_i^2 - np_i = \\ &= n^2p_i^2 - nP - i^2 = n(n - 1)p_i^2 \end{aligned}$$

terminando la dimostrazione.

d) Ricordando che $I_c(x) = \sum_{i \in Z_{26}} \frac{f_i}{n} \frac{f_{i-1}}{n-1}$ è possibile dimostrare che

$$E[I_c(x)] = \sum_{i=0}^{25} p_i^2$$

sfruttando i risultati del punto precedente. Si ha infatti:

$$E[I_c(x)] = E\left[\sum_{i \in Z_{26}} \frac{f_i}{n} \cdot \frac{f_{i-1}}{n-1}\right]$$

per la linearità del valore atteso si ottiene:

$$\begin{aligned} E[I_c(x)] &= \sum_{i \in Z_{26}} E\left[\frac{f_i}{n} \cdot \frac{f_{i-1}}{n-1}\right] \\ &= \frac{1}{n(n-1)} \sum_{i \in Z_{26}} E[f_i(f_i - 1)] \\ &= \frac{1}{n(n-1)} \sum_{i=0}^{25} E[f_i(f_i - 1)] \end{aligned}$$

per il risultato in (c):

$$E[I_c(x)] = \frac{1}{n(n-1)} \sum_{i=0}^{25} n(n-1)p_i^2 = \sum_{i=0}^{25} p_i^2$$

e) Considerando la seconda fase dell'attacco al cifrario di Vigenére, chiamando $\mathbf{q}$ lo shift circolare di k posizioni $0 \leq n \leq 25$ che meglio approssima il vettore delle probabilità caratteristico della lingua inglese $\mathbf{p}$, vogliamo dimostrare che il prodotto scalare tra $\mathbf{p}$ e $\mathbf{q}$ è maggiore o uguale del prodotto scalare che otterrei tra $\mathbf{p}$ e qualsiasi altro shift i del vettore caratteristico delle probabilit'a del testo, ovvero che:

$$\langle \mathbf{p}, \mathbf{q} \rangle \geq \langle \mathbf{p}, \mathbf{qi} \rangle \quad \forall i \in [0, 25]$$

servendosi della disuguaglianza di Cauchy-Schwarz, che afferma che, dati due vettori $\mathbf{v}$ e $\mathbf{u}$:

$$|\langle \mathbf{v}, \mathbf{u} \rangle| \leq \|\mathbf{u}\|_2 \cdot \|\mathbf{v}\|_2$$

e considerando $\mathbf{p} \approx \mathbf{q}$ come $\mathbf{p} = \mathbf{q}$.
Dalla definizione di prodotto scalare sappiamo che:

$$\langle \mathbf{v}, \mathbf{u} \rangle := \sum_{i=0}^{n-1} v_i u_i$$

dove n è la lunghezza dei vettori. Tuttavia in questo caso si ha $\mathbf{p} = \mathbf{q}$, quindi si ottiene:

$$\langle \mathbf{p}, \mathbf{q} \rangle = \sum_{i=0}^{n-1} p_i q_i = \langle \mathbf{p}, \mathbf{p} \rangle = \sum_{i=0}^{n-1} p_i p_i = \|\mathbf{p}\|_2^2$$

dato che il prodotto scalare di un vettore per sé stesso è pari al quadrato della norma 2 del vettore. Riscrivendolo in termini di $\mathbf{p}$ e $\mathbf{q}$ si ottiene:

$$\langle \mathbf{p}, \mathbf{q} \rangle = \|\mathbf{p}\|_2 \cdot \|\mathbf{q}\|_2$$

Applichiamo adesso la disuguaglianza di Cauchy-Schwarz sui vettori $\mathbf{p}$ e $\mathbf{q}$, ottenendo:

$$|\langle \mathbf{p}, \mathbf{q} \rangle| \leq \|\mathbf{p}\|_2 \cdot \|\mathbf{q}\|_2$$

e facciamo lo stesso per i vettori $\mathbf{p}$ e $\mathbf{qi}$, ottenendo:

$$|\langle \mathbf{p}, \mathbf{qi} \rangle| \leq \|\mathbf{p}\|_2 \cdot \|\mathbf{qi}\|_2$$

Per mettere in relazione questi risultati consideriamo la definizione di norma 2 di un vettore:

$$\|\mathbf{v}\|_2 := \sum_{i=0}^{n-1} v_i^2$$

dove n è la lunghezza del vettore. Si nota che il suo valore non dipende dall'ordine degli elementi v_i del vettore, e quindi possiamo concludere che per qualsiasi shift circolare dello stesso vettore la sua norma 2 non cambia. Pertanto si ha:

$$\|\mathbf{q}\|_2 = \|\mathbf{qi}\|_2 \quad \forall i \in [0, 25]$$

dato che $\mathbf{q}$ è uno shift di k posizioni di $\mathbf{q0}$.
Possono ora essere messe in relazione le due disuguaglianze di Cauchy-Schwarz:

$$|\langle \mathbf{p}, \mathbf{q} \rangle| \leq \|\mathbf{p}\|_2 \cdot \|\mathbf{q}\|_2 = \|\mathbf{p}\|_2 \cdot \|\mathbf{qi}\|_2 \geq |\langle \mathbf{p}, \mathbf{qi} \rangle|$$

si conclude quindi che:

$$\langle \mathbf{p}, \mathbf{q} \rangle \geq \langle \mathbf{p}, \mathbf{qi} \rangle \quad \forall i \in [0, 25]$$

2.2 Esercizio 2.3

Per lo svolgimento dell'esercizio si è scelto di implementare un codice Python, al fine di semplificare le operazioni da effettuare. Tale codice è contenuto nel file *Vigenere.py*, e per ottenere i risultati richiesti è sufficiente eseguire il main. Come prima cosa il testo da decifrare tramite il cifrario di Vigenére è stato assegnato ad una stringa nel main, trasformando le lettere in minuscolo con il comando *.lower()*.

1. Il primo punto richiedeva di trovare le ripetizioni nel testo, le loro distanze e i valori di m (lunghezza della chiave). Per determinare le ripetizioni nel testo è stato implementato il metodo *get_repeated_ngrams_frequencies* che prende in input il ciphertext e un intero n e restituisce un dizionario contenente le frequenze degli n-grammi ripetuti. Questo è stato ottenuto scorrendo il ciphertext a gruppi di n lettere e incrementando la frequenza nel dizionario *ngrams_frequencies*. In seguito è stato creato un nuovo dizionario, *repeated_ngrams_frequencies*

contenente solo gli n-grammi ripetuti nel testo. Per trovare le distanze tra gli n-grammi ripetuti è stato implementato il metodo *get_ngram_distances*, che richiama *equals_ngrams_positions* per determinare le posizioni dei vari n-grammi, dopodiché effettua le differenze tra le posizioni e le riporta in un dizionario associate ai rispettivi n-grammi. Le ripetizioni nel testo sono molte, e si possono osservare fino al caso di n=5. Si riportano i risultati ottenuti nel caso di n=3 (figura 4) , ma eseguendo il codice si possono visualizzare stampati a video i risultati ottenuti per n che va da 2 a 5.

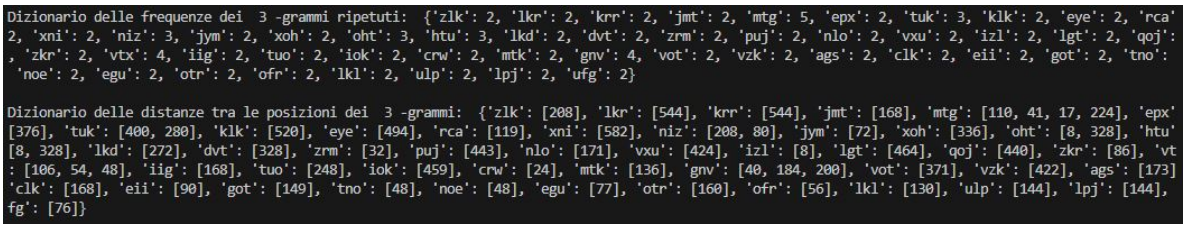

Figura 4: Ripetizioni e distanze dei trigrammi.

Per determinare la lunghezza della chiave, come prima cosa è stato tentato l'approccio del test di Kasiski, a tale fine sono state eseguite le funzioni *get_repeated_ngrams_frequencies* e *get_ngram_distances* con n=3. Una volta ottentute le distanze tra i trigrammi ripetuti, è stato implementato il metodo *get_trigram_gcd* che calcola gli MCD tra le distanze per ciascun trigramma, restituendoli un dizionario. Infine la funzione *get_top_3_frequent_mcd* restituisce gli MCD più frequenti tra quelli trovati, che suggeriscono possibili lunghezze m della chiave. Eseguendo il codice, come tre MCD più frequenti si ottengono 8, 168 e 544. La lunghezza della chiave suggerita dal test di Kasiski risulta quindi essere 8, poiché gli altri valori risultano troppo elevati e probabilmente sono il risultato di distanze spurie.
A seguito del test di Kasiski si è scelto di applicare anche il metodo degli indici di coincidenza, che sfrutta quest'ultimi per determinare sia la lunghezza della chiave, sia per determinare la chiave stessa. Nel metodo *find_vigenere_key_length* vengono fatti più tentativi con diversi valori di m (lunghezza della chiave), ricercando quello che permette di minimizzare lo scarto quadratico medio dell'indice di coincidenza rispetto a quello della lingua inglese, che corrisponde a circa 0.067. Per ogni valore della chiave, il testo viene suddiviso in sottostringhe nel metodo *get_subtexts*, che restituisce un array di sottostringhe suddivise in base alla lettera della chiave utilizzata per la cifratura. Per ciascun *subtext* viene calcolato l'indice di coincidenza tramite la funzione *calculate_coincidence_index*, infine viene determinato lo scarto quadratico medio rispetto all'indice di coincidenza della lingua inglese. La lunghezza della chiave restituita sarà quella che minimizza tale valore. Anche dal metodo degli indici di coincidenza si ottiene come lunghezza della chiave suggerita m=8.

2. Il secondo punto richiedeva i valori degli indici di coincidenza che si ottengono per il valore corretto di m. Per m=8 si ottengono i seguenti

valori degli indici di coincidenza:

- 0.06248685041026719
- 0.07027140753208501
- 0.05680622764569744
- 0.058699766722069
- 0.052177572059751716
- 0.05954134231011993
- 0.06636597938144329
- 0.05562714776632303

Il valore medio di questi indici di coincidenza risulta essere 0.060, un valore che, come ci si aspetterebbe, si avvicina molto a quello della lingua inglese.

3. Una volta determinata la lunghezza della chiave, è possibile utilizzare gli indici di coincidenza anche per determinare il valore stesso della chiave. A tale fine è stata implementata la funzione *find_vigenere_key*. Innanzitutto viene assegnata ad una variabile *eng_frequencies* il valore delle distribuzioni di probabilità di ogni lettera della lingua inglese. Dopodiché viene nuovamente richiamato il metodo *get_subtexts* per ottenere il testo raggruppato in sottostringhe cifrate con lo stesso valore della chiave. Per ciascuna delle sottostringhe, *subtext*, viene ricercata la lettera utilizzata per la cifratura. Per farlo è necessario ricercare lo shift del vettore delle distribuzioni di probabilità di ogni lettera in *subtext* che massimizza il prodotto scalare di tale vettore con *eng_frequencies*. Per otenere tale vettore, per ogni *subtext* viene utilizzato un oggetto Counter per determinare le frequenze di ogni lettera, poi viene creato un vettore con le frequenze relative, *vector_frequencies*, a cui vengono aggiunte le lettere non presenti in *subtext* con frequenza nulla. Per ogni shift verso sinistra di *vector_frequencies*, viene eseguito il prodotto scalare con *eng_frequencies* (figura 5). Lo shift che permette di ottenere il risultato maggiore viene selezionato e trasformato in lettera tramite il comando *chr(shift + 97)*. Eseguendo questo metodo si ottiene la chiave "***gregegan***".

```
Lettera della chiave trovata:  g  con prodotto scalare 0.06312897959183675
Lettera della chiave trovata:  r  con prodotto scalare 0.06873469387755103
Lettera della chiave trovata:  e  con prodotto scalare 0.06196122448979593
Lettera della chiave trovata:  g  con prodotto scalare 0.06239591836734695
Lettera della chiave trovata:  e  con prodotto scalare 0.059942244897959185
Lettera della chiave trovata:  g  con prodotto scalare 0.06548469387755101
Lettera della chiave trovata:  a  con prodotto scalare 0.06612577319587629
Lettera della chiave trovata:  n  con prodotto scalare 0.06412061855670104
```

Figura 5: Valori dei prodotti scalari per ogni lettera della chiave trovata.

Trovata la chiave è possibile eseguire la decifratura del testo. Questo è possibile grazie alla funzione *decrypt_vigenere*, che per ogni lettera del

ciphertext, la trasforma nel corrispondente valore ASCII, vi sottrae in modulo 26 il valore ASCII della lettera della chiave corrispondente e trasforma nuovamente il risultato in stringa. Dalla decifratura con la chiave trovata si ottiene il testo in figura 6.

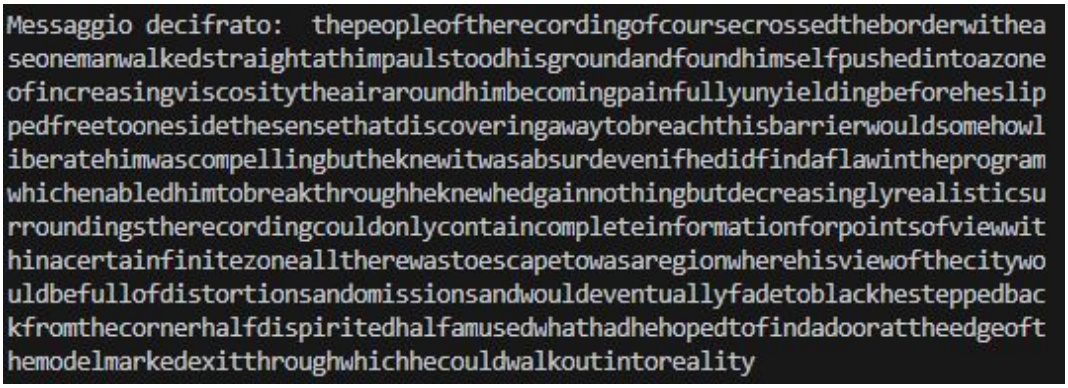

Figura 6: Testo decifrato.

Pertanto la decifratura si considera riuscita.