program that can perform a letter frequency attack on any monoalphabetic substitution cipher without human intervention. Your software should produce possible plaintexts in rough order of likelihood. It would be good if your user interface allowed the user to specify “give me the top 10 possible plaintexts.”

import random

from collections import Counter

EN\_ORDER="ETAOINSHRDLCUMWFGYPBVKJXQZ"

COMMON\_WORDS=["the","be","to","of","and","a","in","that","have","i","it","for","not","on","with","he","as","you","do","at","this","but","his","by","from","they","we","say","her","she"]

def clean\_text(s):

return ''.join(c for c in s.lower() if c.isalpha() or c==' ')

def apply\_key(cipher,key\_map):

out=[]

for ch in cipher:

if ch.isalpha():

up=ch.upper()

dec=key\_map.get(up,'?')

out.append(dec if ch.isupper() else dec.lower())

else:

out.append(ch)

return ''.join(out)

def freq\_initial(cipher):

letters=[c for c in cipher.upper() if c.isalpha()]

cnt=Counter(letters)

items=sorted(cnt.items(), key=lambda x:(-x[1], random.random()))

order=''.join(c for c,\_ in items)

rem=[c for c in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" if c not in order]

random.shuffle(rem)

order=order+''.join(rem)

key\_map={order[i]:EN\_ORDER[i] for i in range(26)}

return key\_map

def random\_key():

letters=list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

random.shuffle(letters)

return {c:letters[i] for i,c in enumerate("ABCDEFGHIJKLMNOPQRSTUVWXYZ")}

EN\_FREQ={'A':8.167,'B':1.492,'C':2.782,'D':4.253,'E':12.702,'F':2.228,'G':2.015,'H':6.094,'I':6.966,'J':0.153,'K':0.772,'L':4.025,'M':2.406,'N':6.749,'O':7.507,'P':1.929,'Q':0.095,'R':5.987,'S':6.327,'T':9.056,'U':2.758,'V':0.978,'W':2.361,'X':0.150,'Y':1.974,'Z':0.074}

def chi\_score(text):

letters=[c.upper() for c in text if c.isalpha()]

N=len(letters)

if N==0: return -1e9

obs=Counter(letters)

chi=0.0

for L,pct in EN\_FREQ.items():

expected=pct/100.0\*N

observed=obs.get(L,0)

exp=expected if expected>1e-8 else 1e-8

chi += (observed-exp)\*\*2/exp

return -chi

def word\_score(text):

s=text.lower()

score=0.0

for w in COMMON\_WORDS:

score += s.count(" "+w+" ")

if s.startswith(w+" "): score+=1

if s.endswith(" "+w): score+=1

score += s.count(' ')

return score

def score\_plain(text):

return word\_score(text)\*3.0 + chi\_score(text)

def neighbor\_swap(key\_map):

letters=list("ABCDEFGHIJKLMNOPQRSTUVWXYZ")

mapping=[key\_map[c] for c in letters]

i,j=random.sample(range(26),2)

mapping[i],mapping[j]=mapping[j],mapping[i]

return {letters[k]:mapping[k] for k in range(26)}

def hill\_climb(cipher, init\_key, max\_iters=2000):

best\_key=init\_key

best\_plain=apply\_key(cipher,best\_key)

best\_score=score\_plain(clean\_text(best\_plain))

current\_key=best\_key

current\_score=best\_score

for \_ in range(max\_iters):

cand\_key=neighbor\_swap(current\_key)

cand\_plain=apply\_key(cipher,cand\_key)

cand\_score=score\_plain(clean\_text(cand\_plain))

if cand\_score>current\_score:

current\_key=cand\_key

current\_score=cand\_score

if current\_score>best\_score:

best\_key,best\_score=current\_key,current\_score

best\_plain=apply\_key(cipher,best\_key)

if random.random()<0.01:

current\_key=random\_key()

current\_score=score\_plain(clean\_text(apply\_key(cipher,current\_key)))

return best\_key,best\_plain,best\_score

def attack(cipher, top\_n=10, restarts=30, iters=1500):

candidates=[]

seen=set()

for \_ in range(restarts):

if random.random()<0.7:

init=freq\_initial(cipher)

for \_\_ in range(5): init=neighbor\_swap(init)

else:

init=random\_key()

\_,plain,sc=hill\_climb(cipher,init,max\_iters=iters)

norm=clean\_text(plain).strip()

if norm and norm not in seen:

seen.add(norm)

candidates.append((sc,plain))

candidates.sort(key=lambda x:-x[0])

return candidates[:top\_n]

if \_\_name\_\_=="\_\_main\_\_":

ct=input("Enter ciphertext:\n")

tn=input("Top how many candidates? (default 10): ").strip()

try:

top\_n=int(tn) if tn else 10

except:

top\_n=10

results=attack(ct,top\_n,restarts=40,iters=2000)

print(f"{'Rank':>4} {'Score':>9} Plaintext")

print("-"\*60)

for i,(sc,pt) in enumerate(results,1):

print(f"{i:>4} {sc:9.3f} {pt[:200]}")

