**Text Encryption and Decryption using 2s Complement based on Collatz Conjecture**

import tkinter as tk

from tkinter import simpledialog, messagebox

import time

import random

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import numpy as np

encryption\_table = {}

last\_encryption\_key = None

first\_encryption\_done = False

encryption\_times = []

decryption\_times = []

repeated\_encryption\_times = []

def textuni(text):

return [ord(char) for char in text]

def calculate\_value(unicode\_val, key):

if key < unicode\_val:

return key - unicode\_val

elif key > unicode\_val:

return unicode\_val - key

else:

return -key

def twos\_comp(num):

if num < 0:

num\_bits = num.bit\_length() + 1

twos\_complement = (1 << num\_bits) + num

return twos\_complement

return num

def collatz\_transform(value, iterations):

for \_ in range(iterations):

if value % 2 == 0:

value //= 2

else:

value = 3 \* value + 1

return value

def encrypt\_text(text, key, iterations=2):

global first\_encryption\_done

unicode\_values = textuni(text)

encrypted\_values = []

random\_offset = 0 if not first\_encryption\_done else random.randint(1, 100)

for unicode\_val in unicode\_values:

value = calculate\_value(unicode\_val, key) + random\_offset

value = twos\_comp(value)

transformed\_value = collatz\_transform(value, iterations)

encrypted\_char = chr((transformed\_value % 94) + 33)

encrypted\_values.append(encrypted\_char)

encrypted\_text = ''.join(encrypted\_values)

encryption\_table[encrypted\_text] = (text, random\_offset)

first\_encryption\_done = True

return encrypted\_text

def decrypt\_text(encrypted\_text, key):

if encrypted\_text in encryption\_table:

original\_text, random\_offset = encryption\_table[encrypted\_text]

if key == last\_encryption\_key:

del encryption\_table[encrypted\_text]

return original\_text

else:

return "Invalid Key"

else:

return None

def count\_characters(text):

letters = sum(1 for char in text if char.isalpha())

digits = sum(1 for char in text if char.isdigit())

special = len(text) - letters - digits

return letters, digits, special

def update\_counts():

# Update character count for original text

original\_text = text\_entry.get("1.0", tk.END).strip()

char\_count\_label.config(text=f"Character Count (Original): {len(original\_text)}")

# Update character count for encrypted text

encrypted\_text = encrypted\_text\_box.get("1.0", tk.END).strip()

letters, digits, special = count\_characters(encrypted\_text)

char\_count\_encrypt\_label.config(text=f"Letters: {letters}, Digits: {digits}, Special: {special}")

# Update character count for decrypted text

decrypted\_text = decrypted\_text\_box.get("1.0", tk.END).strip()

letters, digits, special = count\_characters(decrypted\_text)

char\_count\_decrypt\_label.config(text=f"Letters: {letters}, Digits: {digits}, Special: {special}")

def handle\_encrypt():

text = text\_entry.get("1.0", tk.END).strip()

key = key\_entry.get()

iterations = iterations\_entry.get()

if not text or not key or not iterations:

messagebox.showerror("Error", "Please fill in all fields.")

return

try:

key = int(key)

iterations = int(iterations)

start\_time = time.perf\_counter()

encrypted\_text = encrypt\_text(text, key, iterations)

end\_time = time.perf\_counter()

encryption\_time = end\_time - start\_time

encryption\_times.append(encryption\_time) # Store encryption time

# Store repeated encryption times for the same input

repeated\_encryption\_times.append(encryption\_time)

encrypted\_text\_box.delete("1.0", tk.END)

encrypted\_text\_box.insert("1.0", encrypted\_text)

global last\_encryption\_key

last\_encryption\_key = key

digit\_count\_label.config(text=f"Total Count (Encrypted): {len(encrypted\_text)}")

encryption\_time\_label.config(text=f"Encryption Time: {encryption\_time:.6f} seconds")

# Update counts after encryption

update\_counts()

except ValueError:

messagebox.showerror("Error", "Key and Iterations must be numbers.")

def handle\_decrypt():

encrypted\_text = encrypted\_text\_box.get("1.0", tk.END).strip()

if not encrypted\_text:

messagebox.showerror("Error", "Please provide encrypted text for decryption.")

return

key = simpledialog.askinteger("Decryption Key", "Enter the decryption key:")

if key is None:

messagebox.showerror("Error", "Please enter a valid key.")

return

start\_time = time.perf\_counter()

decrypted\_text = decrypt\_text(encrypted\_text, key)

end\_time = time.perf\_counter()

decryption\_time = end\_time - start\_time

decryption\_times.append(decryption\_time) # Store decryption time

if decrypted\_text:

decrypted\_text\_box.delete("1.0", tk.END)

decrypted\_text\_box.insert("1.0", decrypted\_text)

decryption\_time\_label.config(text=f"Decryption Time: {decryption\_time:.6f} seconds")

# Update counts after decryption

update\_counts()

else:

messagebox.showerror("Error", "Invalid encrypted text or decryption key.")

def show\_time\_graph():

if not encryption\_times or not decryption\_times:

messagebox.showerror("Error", "No data available to plot. Perform encryption and decryption first.")

return

graph\_window = tk.Toplevel()

graph\_window.title("Encryption and Decryption Time Graph")

fig, ax = plt.subplots(figsize=(8, 4))

# Plot encryption and decryption times as a bar graph

processes = ['Encryption', 'Decryption']

times = [encryption\_times[-1], decryption\_times[-1]] # Use the most recent times

ax.bar(processes, times, color=['blue', 'red'])

ax.set\_xlabel('Process')

ax.set\_ylabel('Time (seconds)')

ax.set\_title('Time taken for Encryption and Decryption')

canvas = FigureCanvasTkAgg(fig, master=graph\_window)

canvas.draw()

canvas.get\_tk\_widget().pack()

def show\_repeated\_encryption\_graph():

if not repeated\_encryption\_times:

messagebox.showerror("Error", "No repeated encryption data available. Encrypt the same text multiple times.")

return

avg\_time = np.mean(repeated\_encryption\_times) # Calculate the average time

graph\_window = tk.Toplevel()

graph\_window.title("Repeated Encryption Time Graph")

fig, ax = plt.subplots(figsize=(8, 4))

attempts = range(1, len(repeated\_encryption\_times) + 1)

ax.bar(attempts, repeated\_encryption\_times, color='purple', label="Encryption Time")

# Plot the average as a horizontal line

ax.axhline(avg\_time, color='red', linestyle='dashed', linewidth=2, label=f'Avg Time: {avg\_time:.6f}s')

ax.set\_xlabel('Encryption Attempt')

ax.set\_ylabel('Time (seconds)')

ax.set\_title('Encryption Time for Same Input, Key, and Iterations')

ax.legend() # Show legend

canvas = FigureCanvasTkAgg(fig, master=graph\_window)

canvas.draw()

canvas.get\_tk\_widget().pack()

def create\_gui():

global text\_entry, key\_entry, iterations\_entry, encrypted\_text\_box, decrypted\_text\_box

global char\_count\_label, digit\_count\_label, encryption\_time\_label, decryption\_time\_label, char\_count\_decrypt\_label, char\_count\_encrypt\_label

root = tk.Tk()

root.title("Encrypt/Decrypt Tool")

tk.Label(root, text="Text:").grid(row=0, column=0, padx=10, pady=10, sticky="nw")

text\_entry = tk.Text(root, width=70, height=10)

text\_entry.grid(row=0, column=1, padx=10, pady=10)

tk.Label(root, text="Key:").grid(row=1, column=0, padx=10, pady=10)

key\_entry = tk.Entry(root, width=30)

key\_entry.grid(row=1, column=1, padx=10, pady=10)

tk.Label(root, text="Iterations:").grid(row=2, column=0, padx=10, pady=10)

iterations\_entry = tk.Entry(root, width=30)

iterations\_entry.grid(row=2, column=1, padx=10, pady=10)

encrypt\_button = tk.Button(root, text="Encrypt", command=handle\_encrypt, width=15, bg="lightblue")

encrypt\_button.grid(row=3, column=0, padx=10, pady=10)

decrypt\_button = tk.Button(root, text="Decrypt", command=handle\_decrypt, width=15, bg="lightgreen")

decrypt\_button.grid(row=3, column=1, padx=10, pady=10)

tk.Label(root, text="Encrypted Text:").grid(row=4, column=0, padx=10, pady=10, sticky="nw")

encrypted\_text\_box = tk.Text(root, width=70, height=10)

encrypted\_text\_box.grid(row=4, column=1, padx=10, pady=10)

tk.Label(root, text="Decrypted Text:").grid(row=5, column=0, padx=10, pady=10, sticky="nw")

decrypted\_text\_box = tk.Text(root, width=70, height=10)

decrypted\_text\_box.grid(row=5, column=1, padx=10, pady=10)

graph\_button = tk.Button(root, text="Show Graph", command=show\_time\_graph, bg="lightyellow")

graph\_button.grid(row=3, column=4, padx=50, pady=10) # Moved right

char\_count\_label = tk.Label(root, text="Character Count (Original): 0")

char\_count\_label.grid(row=0, column=2, padx=10, pady=5, sticky="nw")

digit\_count\_label = tk.Label(root, text="Total Count (Encrypted): 0")

digit\_count\_label.grid(row=1, column=2, padx=10, pady=5, sticky="nw")

encryption\_time\_label = tk.Label(root, text="Encryption Time: 0.000000 seconds")

encryption\_time\_label.grid(row=2, column=2, padx=10, pady=5, sticky="nw")

decryption\_time\_label = tk.Label(root, text="Decryption Time: 0.000000 seconds")

decryption\_time\_label.grid(row=3, column=2, padx=10, pady=5, sticky="nw")

char\_count\_decrypt\_label = tk.Label(root, text="Letters: 0, Digits: 0, Special: 0")

char\_count\_decrypt\_label.grid(row=5, column=2, padx=10, pady=5, sticky="nw")

# New label for encrypted text character count

char\_count\_encrypt\_label = tk.Label(root, text="Letters: 0, Digits: 0, Special: 0")

char\_count\_encrypt\_label.grid(row=4, column=2, padx=10, pady=5, sticky="nw")

repeated\_graph\_button = tk.Button(root, text="Repeated Encryption Graph", command=show\_repeated\_encryption\_graph, bg="lightpink")

repeated\_graph\_button.grid(row=4, column=4, padx=10, pady=10)

root.mainloop()

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

create\_gui()