

算法与数据结构设计报告

（ 2020 / 2021学年 第 一 学期）

题 目： A. 二叉树的宽度求解问题

B. 凯撒密码加解密实现

|  |  |
| --- | --- |
| **专 业** | **计算机科学与技术** |
| **学 生 姓 名** | **殷祥** |
| **班 级 学 号** | **B18030825** |
| **指 导 教 师** | **徐 鹤** |
| **指 导 单 位** | **计算机科学与技术系** |
| **日 期** | **2020.10.26-2020.11.06** |

|  |  |  |
| --- | --- | --- |
| **支撑指标点** | **评价准则** | **计分（每项10分）** |
| 3.1掌握设计/开发复杂工程问题解决方案所需要的专业知识和开发工具（40分） | 1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，进行合理的分析与设计 |  |
| 2、能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现 |  |
| 3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述 |  |
| 4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便 |  |
| 4.1能基于专业理论，针对计算机及应用领域复杂工程问题设计可行的实验方案（40分） | 5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标 |  |
| 6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解 |  |
| 7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能 |  |
| 8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试 |  |
| 4.2能够根据实验方案，配置实验环境、开展实验，使用定性或定量分析方法进行数据分析与处理，综合实验结果以获得合理有效的结论（20分） | 9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握 |  |
| 10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范 |  |
| 算法与数据结构设计能力测评总分 | |  |
| **指导教师： 2020 年 11 月 10 日** | | |
| **备注：** | | |

**二叉树的宽度求解问题**

**一、课题内容和要求**

设二叉树的宽度定义为具有结点数最多的那一层上的结点个数。试设计算法求二叉树的宽度，并输出各结点的高度。

扩展：如何设计算法求解指定二叉树的层次上的结点个数并输出该层的所有结点？

**二、课题需求分析**

本课题需要实现二叉树这个数据结构，且要设计算法来实现相应的功能。

我们需要实现的功能主要有以下几个：

1. 根据给定的数据，构建成一个二叉树；
2. 设计算法来求解二叉树的宽度；
3. 输出二叉树每一个结点的高度，即其所在的层数；
4. 给定一个层数，输出该层上所有结点以及结点的总个数

本课题所需的功能框架图如图1所示。

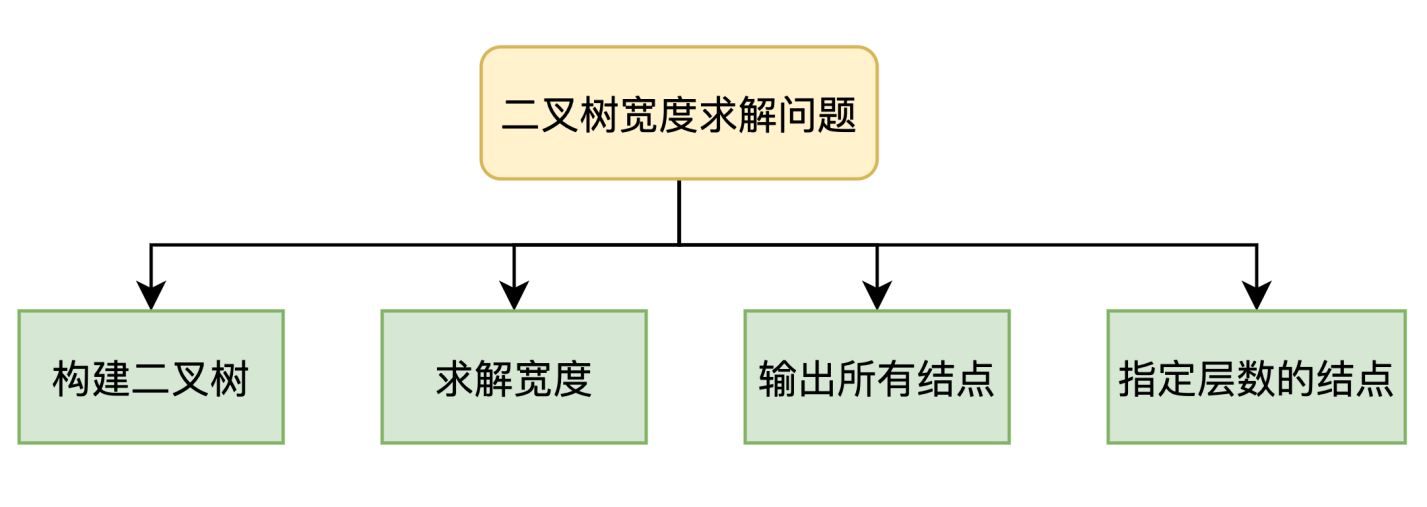


图1 功能框架图

根据课题需求，选择了Golang作为编程语言，对上述功能进行了具体的代码实现。

**三、课题相关数据结构及算法设计**

1 主要数据结构

type DataField interface{}

type BtNode struct {

data DataField

lChild \*BtNode

rChild \*BtNode

level uint

}

type BinaryTree struct {

root \*BtNode

}

2 主要算法流程

1. 构建二叉树算法：将两个子树与一个根结点合并在一起，形成一个新的二叉树，合并后需要将子树所在的层数+1。
2. 层次遍历算法：基于队列这个数据结构，实现了对树中所有结点的遍历。先将根节点入队列，然后只要队列不空，就不断的将队列中的元素出列，并将出列元素的左孩子和右孩子（如果有的话）入队列。通过这个算法，可以对整个二叉树按照层数进行遍历。该算法流程图如图2所示。

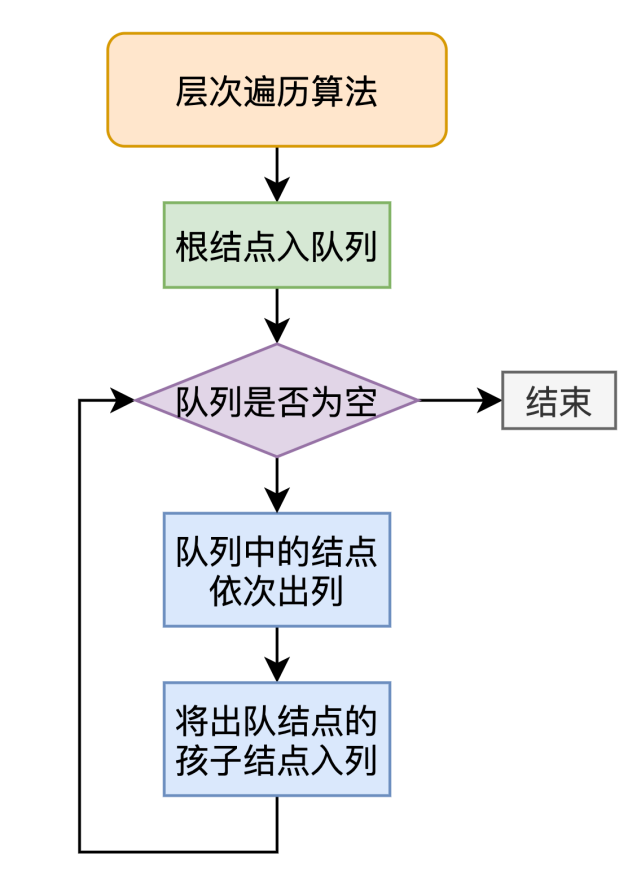


图2 层次遍历图

1. 宽度求解算法：先采用层次遍历算法，得到所有的结点，然后遍历所有结点，计算出每一层上的结点数，找出最大的那一个结点数。
2. 指定层数结点个数求解算法：先采用层次便利算法，得到所有的结点，然后遍历所有结点，判断该结点的层数是否为指定的层数，如果是就输出该结点。

**四、源程序代码**

// binarytree.go

package main

import (

"fmt"

)

type DataField interface{}

type BtNode struct {

data DataField

lChild \*BtNode

rChild \*BtNode

level uint

}

type BinaryTree struct {

root \*BtNode

}

// 合并两棵树

func MakeTree(x DataField, left \*BinaryTree, right \*BinaryTree) \*BinaryTree {

if left == right {

panic("Left tree and right tree cannot be the same.")

}

if left == nil || right == nil {

panic("left tree or right tree cannot be nil.")

}

AddLevel(left.root)

AddLevel(right.root)

newTree := &BinaryTree{

root: &BtNode{x, left.root, right.root, 0},

}

left.root, right.root = nil, nil

return newTree

}

func AddLevel(node \*BtNode) {

if node == nil {

return

}

node.level += 1

AddLevel(node.lChild)

AddLevel(node.rChild)

}

// 层次遍历

func (bt \*BinaryTree) LevelOrderTree() []\*BtNode {

nodes := make([]\*BtNode, 0)

queue := make([]\*BtNode, 0)

queue = append(queue, bt.root)

for len(queue) != 0 {

// pop 1 node from queue

node := queue[0]

queue = queue[1:]

// push, if any, child node of this node to queue

if node.lChild != nil {

queue = append(queue, node.lChild)

}

if node.rChild != nil {

queue = append(queue, node.rChild)

}

// add this node to the slice of all nodes

nodes = append(nodes, node)

}

return nodes

}

// 结点个数

func (bt \*BinaryTree) SizeOfNodes() uint {

return bt.root.SizeOfNodes()

}

func (node \*BtNode) SizeOfNodes() uint {

if node == nil {

return 0

}

return 1 + node.lChild.SizeOfNodes() + node.rChild.SizeOfNodes()

}

// 二叉树高度

func (bt \*BinaryTree) Height() uint {

return bt.root.Height()

}

func (node \*BtNode) Height() uint {

if node == nil {

return 0

}

if node.lChild == nil && node.rChild == nil {

return 1

}

lHeight, rHeight := node.lChild.Height(), node.rChild.Height()

if lHeight > rHeight {

return 1 + lHeight

}

return 1 + rHeight

}

// 二叉树宽度

func (bt \*BinaryTree) Width() uint {

// 层次遍历出所有结点

nodes := bt.LevelOrderTree()

// 计算每一层的结点数

widths := make([]uint, bt.Height())

for \_, node := range nodes {

widths[node.level] += 1

}

// 找最大的结点数

maxWidth := uint(0)

for \_, width := range widths {

if width > maxWidth {

maxWidth = width

}

}

return maxWidth

}

// 输出各结点的高度

func (bt \*BinaryTree) HeightOfAllNodes() {

// 层次遍历出所有结点

nodes := bt.LevelOrderTree()

// 输出每个结点的高度

for \_, node := range nodes {

fmt.Printf("The level of node %v is: %d\n", node.data, node.level)

}

}

// 输出指定层数的所有结点及其总个数

func (bt \*BinaryTree) AllNodesOfLevel(level uint) {

// 判断高度

if level > bt.Height() {

fmt.Printf("level %d is beyond level of the tree %d\n", level, bt.Height())

return

}

// 层次遍历出所有结点

nodes := bt.LevelOrderTree()

// 输出指定高度的所有结点

sum := 0

for \_, node := range nodes {

if node.level == level {

sum += 1

fmt.Printf("node: %v\n", node.data)

}

}

fmt.Printf("Level %d contains %d nodes in total\n", level, sum)

}

func main() {

y := &BinaryTree{

root: &BtNode{"E", nil, nil, 0},

}

z := &BinaryTree{

root: nil,

}

x := MakeTree("C", y, z)

y = &BinaryTree{

root: &BtNode{"D", nil, nil, 0},

}

tree := MakeTree("B", y, x)

/\*

B

/ \

D C

/

E

\*/

println(`树：

B

/ \

D C

/

E

`)

fmt.Printf("Width of the tree is %d\n", tree.Width())

tree.HeightOfAllNodes()

println("\nAll nodes of level 1 is: ")

tree.AllNodesOfLevel(1)

}

**五、测试数据及其结果分析**

出于对代码测试的目的，构造了一个3层的具有4个结点的二叉树；并分别调用了相应的函数，对该树的宽度进行了求解、输出该树所有结点及其对应的层数和输出指定某一层次（测试数据为第2层）的所有结点进行了测试。

测试结果如图3所示。

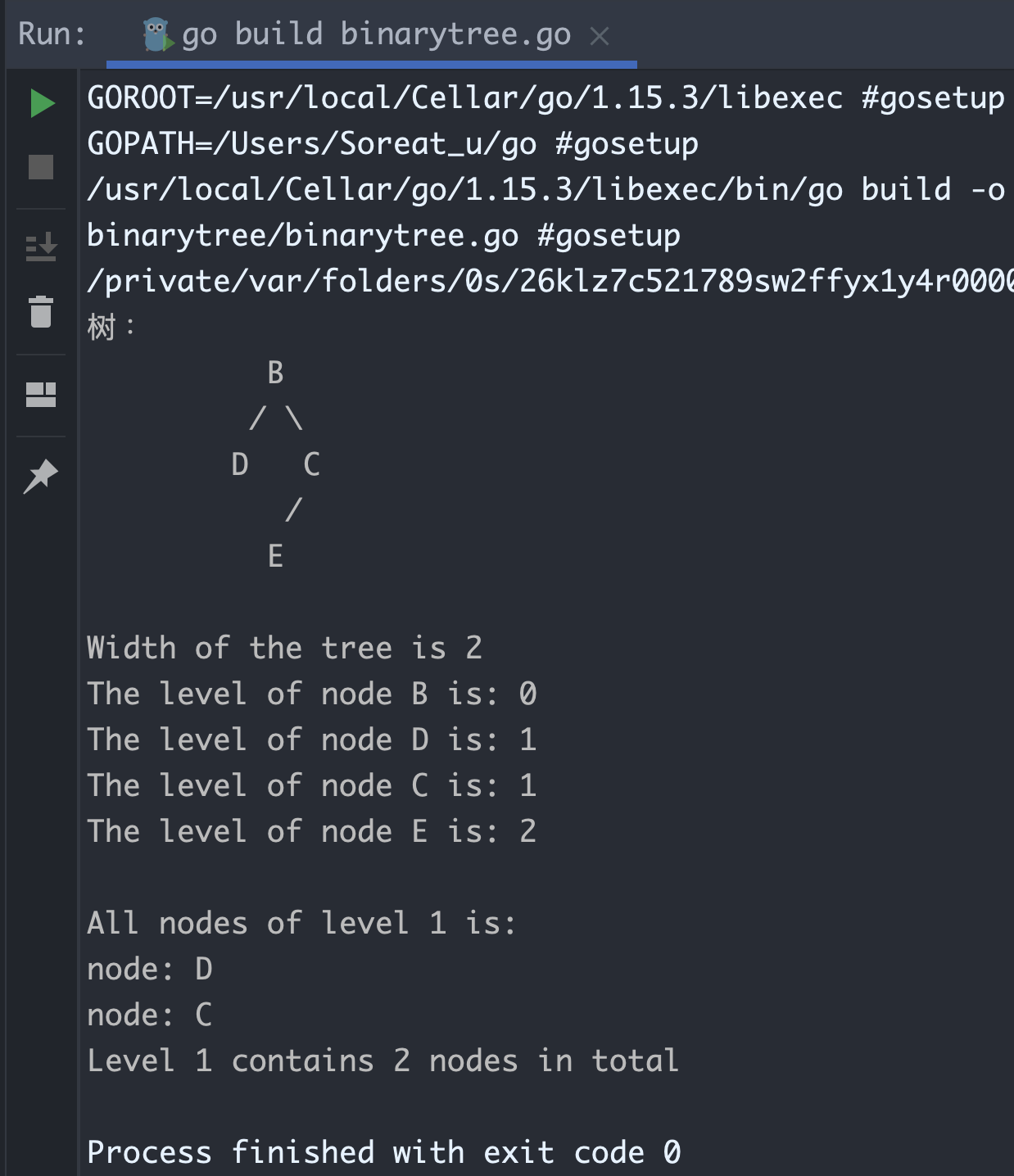


图3 程序测试图

程序能够正确地求解出树的宽度、输出每一个结点所在的层数以及找到指定层数的所有结点。

**六、课题完成过程中遇到的问题及解决方法**

问题1：在GoLand中对程序进行编译过程中，报错“undefined: xxx”。

解决方法：将IDE的run\_kind从file模式修改为以package模式。

问题2：程序运行过程中，报错“nil deference”。

解决方法：调试代码后，发现没有对nil进行检测，添加nil检测后，可以正常运行。

问题3：在实现层次遍历算法的过程中，给make函数生成的slice指定了默认值，导致会有nil相关的报错。

解决方法：将make函数生成的slice的默认值设置为0，就不会报错了。

**凯撒密码加解密实现**

**一、课题内容和要求**

凯撒密码（英语：Caesar Cipher），是一种替换加密的技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是3的时候，所有的字母A将被替换成D，B变成E，以此类推。凯撒密码通常被作为其他更复杂的加密方法中的一个步骤，例如维吉尼亚密码，凯撒密码还应用在现代的ROT13系统中。但是和所有的利用字母表进行替换的加密技术一样，凯撒密码容易被破解。请设计并实现凯撒密码的加密解密算法。

基本要求：

1. 设计一个凯撒密码的加密和解密的程序，要求输入一段字符和密码，输出相应的密文，完成加密过程；若输入被加密的密文及解密密钥，能还原出原文，完成解密。
2. 支持动态的密钥参数设置。
3. 设计合适的界面展示结果；界面操作友好、健壮；界面排版整洁美观。

提高要求：

1. 设计暴力破解算法，实现对密文进行自动破解。
2. 改进算法以提高凯撒密码的安全性。
3. 图形可视化界面，支持字符输入或文件导入，以及结果保存。

**二、课题需求分析**

本课题需要实现一个凯撒密码的加解密系统，并且要以图形化界面来展示。

我们需要实现的功能主要有以下几个：

1. 实现凯撒密码的加密与解密算法；
2. 设计并实现针对于凯撒密码的暴力破解算法，对密文进行自动破解；
3. 设计并实现改进的凯撒密码，以提高密码系统的安全性；
4. 设计图形化界面，将上述功能展示给用户。
5. 提供数据的导入和导出功能，以方便存储加密结果。

本课题所需的功能框架图如图4所示。

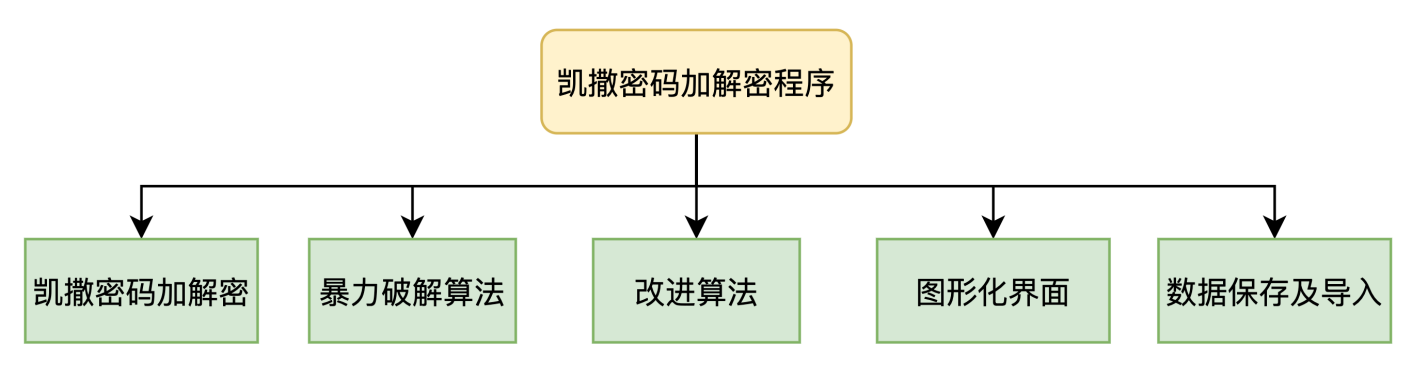


图4 凯撒密码系统功能框架图

根据课题需求，选择了Python作为编程语言，先实现了一遍凯撒密码加解密系统，然后借助PySimpleGUI库来完成图形化界面的实现。

**三、课题相关数据结构及算法设计**

1 主要数据结构

from string import ascii\_uppercase as \_alphabet

2 图形化界面设计

图形化界面设计如图5所示。

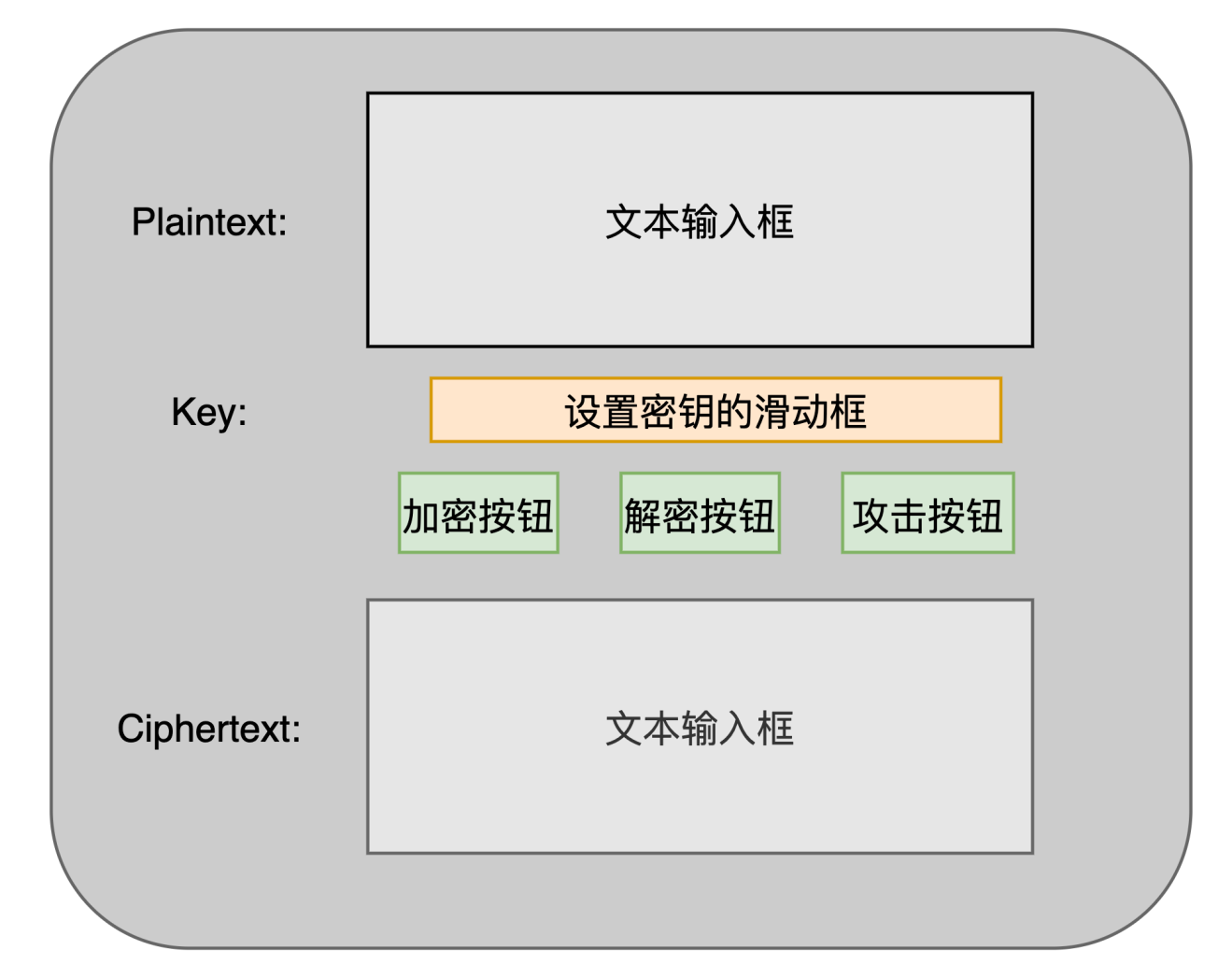


图5 凯撒密码图形化界面设计图

3 主要算法流程

（1）凯撒密码加解密算法：对于明文中的每一个字符，先判断是否在字母表中，如果 在字母表中，则根据密钥对其进行一定的偏移，得到密文字符，然后添加到密文 字符串中；如果字符不在字母表中，则直接将该字符添加到密文字符串中。

（2）改进版的凯撒密码加解密算法：同样也是对明文字符加上一定的偏移量得到密文 字符，不过与标准版的凯撒密码的区别是，每加密完一个字符后，偏移量就会增 加1。这样攻击者如果只能拿到密文，而不知道加密算法的话，是很难破解的。

（3）图形化界面算法：通过事件来对具体功能进行实现。每当有一个事件放生，就 会被路由至相应的处理语句，对该事件进行处理。

**四、源程序代码**

程序分成了两个文件：Caesar.py和main.py

// Caesar.py

from string import ascii\_uppercase as \_alphabet

\_\_all\_\_ = ['Caesar', 'CaesarPlus']

\_ctoi = lambda c: \_alphabet.index(c)

\_itoc = lambda i: \_alphabet[i]

class Caesar:

def \_\_init\_\_(self, key=3):

self.\_key = key

@property

def key(self):

return self.\_key

@key.setter

def key(self, new\_key):

self.\_key = new\_key

def encrypt(self, msg):

cipher = ""

for m in msg.upper():

if m in \_alphabet:

cipher += \_itoc((\_ctoi(m) + self.\_key) % 26)

else:

cipher += m

return cipher

def decrypt(self, cipher):

msg = ""

for c in cipher.upper():

if c in \_alphabet:

msg += \_itoc((\_ctoi(c) - self.\_key) % 26)

else:

msg += c

return msg

def attack(self, cipher):

res = []

inital\_key = self.key

for k in range(26):

self.key = k

res.append((k, self.decrypt(cipher)))

self.key = inital\_key

return res

def \_\_repr\_\_(self):

return "Caesar"

class CaesarPlus:

def \_\_init\_\_(self, key=3):

self.\_key = key

@property

def key(self):

return self.\_key

@key.setter

def key(self, new\_key):

self.\_key = new\_key

def encrypt(self, msg):

cipher = ""

i = self.\_key

for m in msg.upper():

if m in \_alphabet:

cipher += \_itoc((\_ctoi(m) + i) % 26)

i += 1

else:

cipher += m

return cipher

def decrypt(self, cipher):

msg = ""

i = self.\_key

for c in cipher.upper():

if c in \_alphabet:

msg += \_itoc((\_ctoi(c) - i) % 26)

i += 1

else:

msg += c

return msg

def attack(self, cipher):

res = []

inital\_key = self.key

for k in range(26):

self.key = k

res.append((k, self.decrypt(cipher)))

self.key = inital\_key

return res

def \_\_repr\_\_(self):

return "CaesarPlus"

# -------------------------------------------------

def test1():

crypto = Caesar(13)

msg = "the fox jumps over the lazy dog".upper()

cipher = crypto.encrypt(msg)

decrypted = crypto.decrypt(cipher)

print(f"msg: {msg}")

print(f"cipher: {cipher}")

print(f"decrypted: {decrypted}")

def test2():

crypto = CaesarPlus(13)

msg = "the fox jumps over the lazy dog".upper()

cipher = crypto.encrypt(msg)

decrypted = crypto.decrypt(cipher)

print(f"msg: {msg}")

print(f"cipher: {cipher}")

print(f"decrypted: {decrypted}")

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

test1()

test2()

// main.py

import logging

import json

import PySimpleGUI as sg

import Caesar

# ------ Logger Setup ------ #

logging.basicConfig(format='%(asctime)s %(message)s')

logger = logging.getLogger('simple\_example')

logger.setLevel(logging.DEBUG)

# ------ Menu Definition ------ #

menu\_def = [['&File', ['&Open', '&Save', 'E&xit']],

['&Cipher', ['Caesar', 'CaesarPlus']],

['&Help', '&About'], ]

# ------ Layout Definition ------ #

layout = [

[sg.Menu(menu\_def, tearoff=True)],

[sg.Text("Plaintext: ", size=(8, 1)), sg.Multiline("Please input your plaintext here!", key="-PLAINTEXT-")],

[sg.Text("Key: ", size=(8, 1)),

sg.Slider(range=(0, 25), size=(41, 15), orientation='h', default\_value=3, key="-KEY-", enable\_events=True,)],

[sg.Text("", size=(8, 1)), sg.Button("encrypt", size=(10, 1), border\_width=2, pad=(15,15)), sg.Button("decrypt", size=(10, 1), border\_width=2, pad=(15,15)), sg.Button("attack", size=(10, 1), border\_width=2, pad=(15,15))],

[sg.Text("Ciphertext: ", size=(8, 1)), sg.Multiline(key="-CIPHERTEXT-")]

]

window = sg.Window(title="Caesar Cipher Tool", layout=layout, margins=(30, 50))

crypto = Caesar.Caesar()

while True:

event, values = window.read()

if event == None or event == 'Exit':

logger.warning("EXIT...")

break

elif event == "encrypt":

plain = values["-PLAINTEXT-"]

cipher = crypto.encrypt(plain)

window["-CIPHERTEXT-"].update(cipher)

logger.info(f"ENCRYPT: {plain.strip()} -> {cipher.strip()}")

elif event == "decrypt":

cipher = values["-CIPHERTEXT-"]

plain = crypto.decrypt(cipher)

window["-PLAINTEXT-"].update(plain)

logger.info(f"DECRYPT: {cipher.strip()} -> {plain.strip()}")

elif event == "attack":

cipher = values["-CIPHERTEXT-"]

guesses = crypto.attack(cipher)

plains = "\n".join(str(k) + " " + guess.strip() for k, guess in guesses)

window["-PLAINTEXT-"].update(plains)

logger.info(f"ATTACK: {cipher.strip()} -> {plains}")

elif event == "-KEY-":

crypto.key = int(values["-KEY-"])

logger.info(f"Key has been set as {crypto.key}")

elif event == "Open":

filename = sg.popup\_get\_file('Please enter a file name to open')

with open(filename, "r") as f:

data = json.loads(f.read())

window["-PLAINTEXT-"].update(data["Plaintext"])

window["-CIPHERTEXT-"].update(data["Ciphertext"])

window["-KEY-"].update(data["Key"])

if data["Crypto"] == "Caesar":

crypto = Caesar.Caesar(data["Key"])

elif data["Crypto"] == "CaesarPlus":

crypto = Caesar.CaesarPlus(data["Key"])

logger.info(f"OPEN: File {filename} has been loaded, with data={data}")

elif event == "Save":

data = dict()

data["Plaintext"] = values["-PLAINTEXT-"].strip()

data["Ciphertext"] = values["-CIPHERTEXT-"].strip()

data["Key"] = int(values["-KEY-"])

data["Crypto"] = str(crypto)

data = json.dumps(data)

filename = sg.popup\_get\_file('Please enter a file name to save')

with open(filename, "w") as f:

f.write(data)

logger.info(f"SAVE: File has been saved to {filename} with data={data}")

elif event == "Caesar":

key = int(values["-KEY-"])

crypto = Caesar.Caesar(key)

logger.warning(f"CRYPTO: Crypto has been switched to Caesar with key={key}")

elif event == "CaesarPlus":

key = int(values["-KEY-"])

crypto = Caesar.CaesarPlus(key)

logger.warning(f"CRYPTO: Crypto has been switched to CaesarPlus with key={key}")

elif event == "About":

sg.Popup("Written by Soreat\_u")

else:

logger.error(f"Unhandled event: {event}")

window.close()

**五、测试数据及其结果分析**

首先对实现的标准版的凯撒密码系统进行了测试。

输入明文：THE FOX JUMPS OVER THE LAZY DOG

输入密钥key：13

程序会输出密文：GUR SBK WHZCF BIRE GUR YNML QBT

对该密文进行解密，可以得到解密结果：THE FOX JUMPS OVER THE LAZY DOG

然后对改进版的凯撒密码系统进行了测试。

输入明文：THE FOX JUMPS OVER THE LAZY DOG

输入密钥key：13

程序会输出密文：GVT VFP COHLP MUES VKI QGGG MYR

对该密文进行解密，可以得到解密结果：THE FOX JUMPS OVER THE LAZY DOG

测试结果如图6所示。

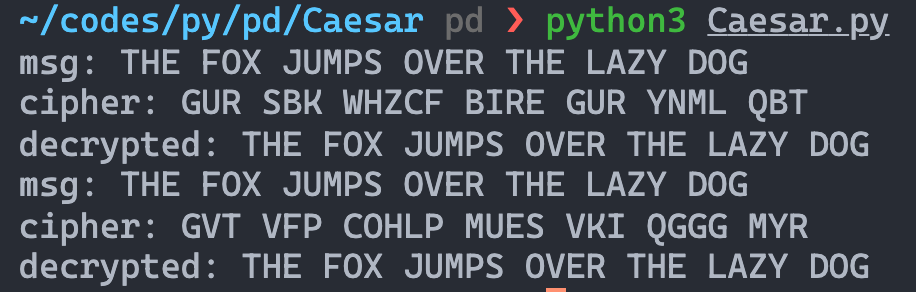


图6 凯撒密码测试图

最后对用PySimpleGUI实现的图形化界面进行了测试。

输入明文ATTACK AT DAWN，选择密钥为8，点击“encrypt”按钮，得到密文IBBIKS IB LIEV，如图7所示。

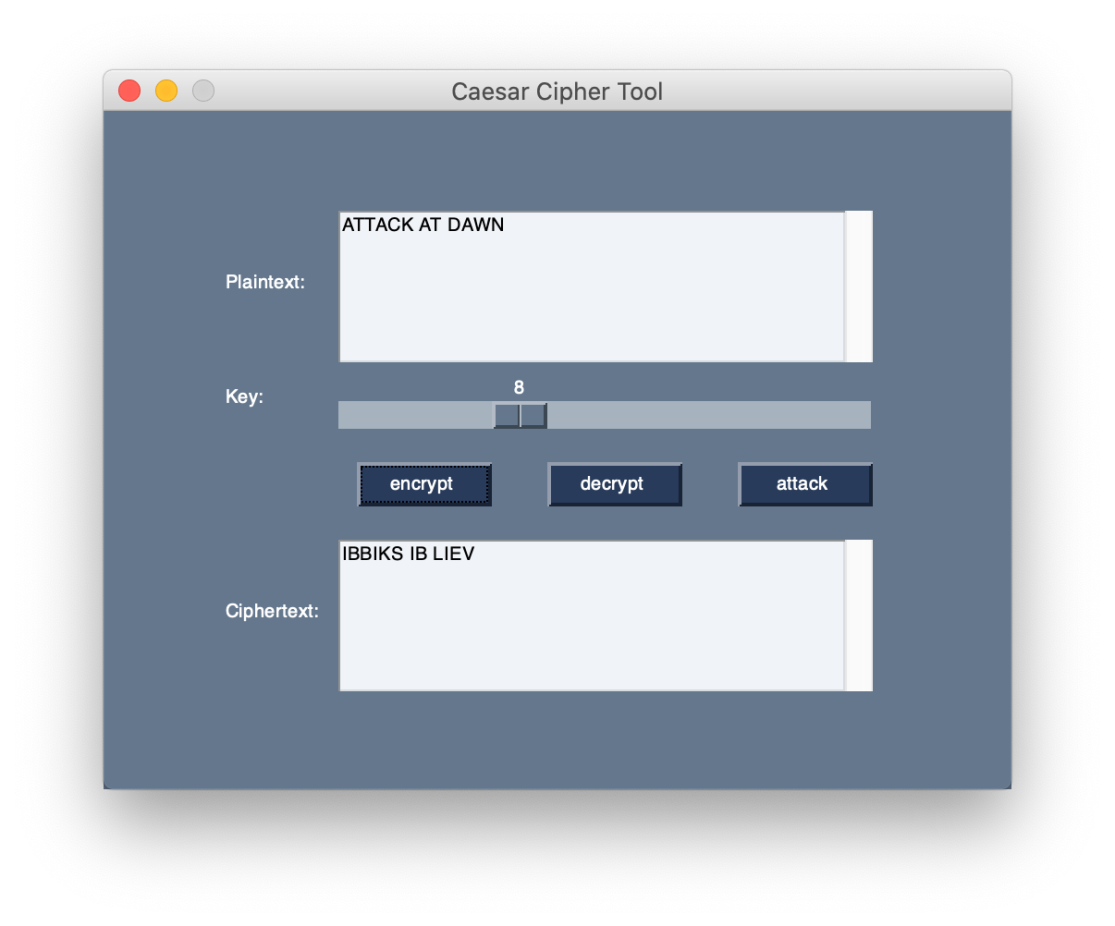


图7 图形化界面加密功能测试图

清除明文输入框中的内容，再点击“decrypt”按钮，可以从明文输入框中得到解密的结果，如图8所示。

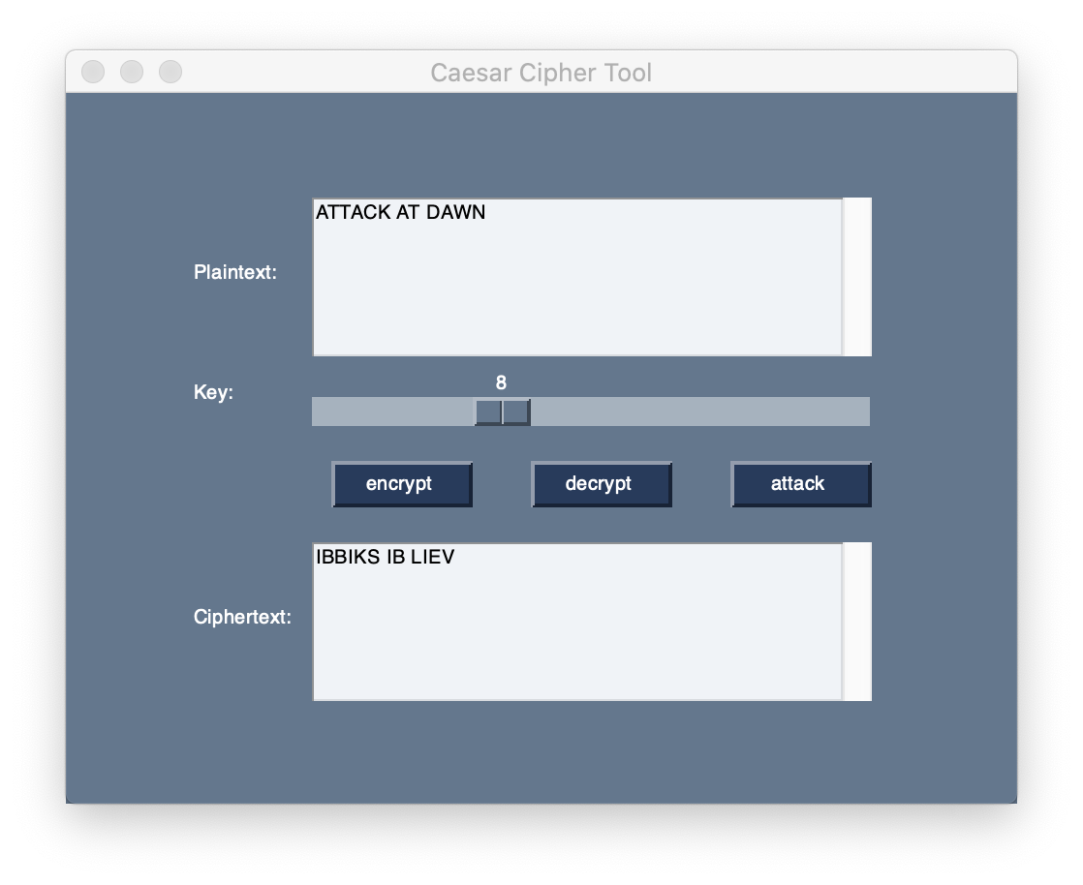


图8 图形化界面解密功能测试图

输入明文ATTACK AT DAWN，选择密钥为13，点击“encrypt”按钮，得到密文NGGNPX NG QNJA，改变密钥的内容为17，再解密，只能得到错误的明文WPPWYG WP ZWSJ。如图9所示。

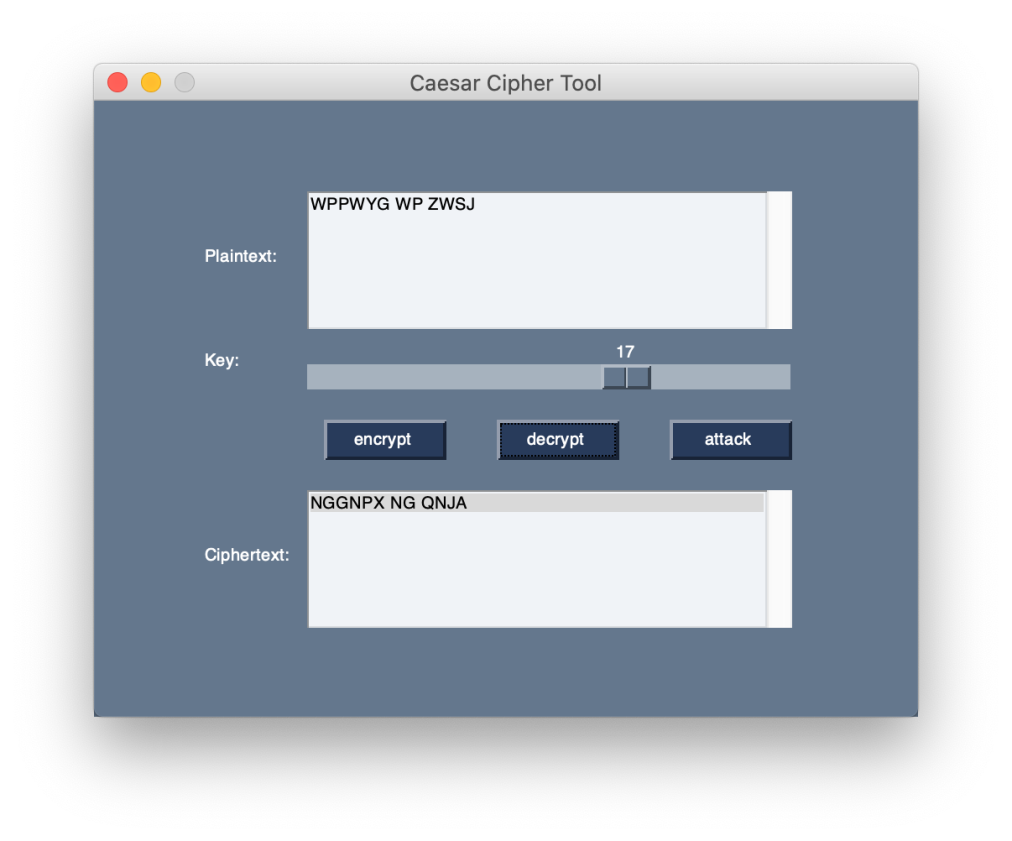


图9 图形化界面解密功能测试图

输入明文ATTACK AT DAWN，选择密钥为21，点击“encrypt”按钮，得到密文VOOVXF VO YVRI。点击“attack”按钮，将会采用暴力方式尝试所有的解密密钥，并输出对应的解密结果，在倒数第5行能够找到正确解密明文。

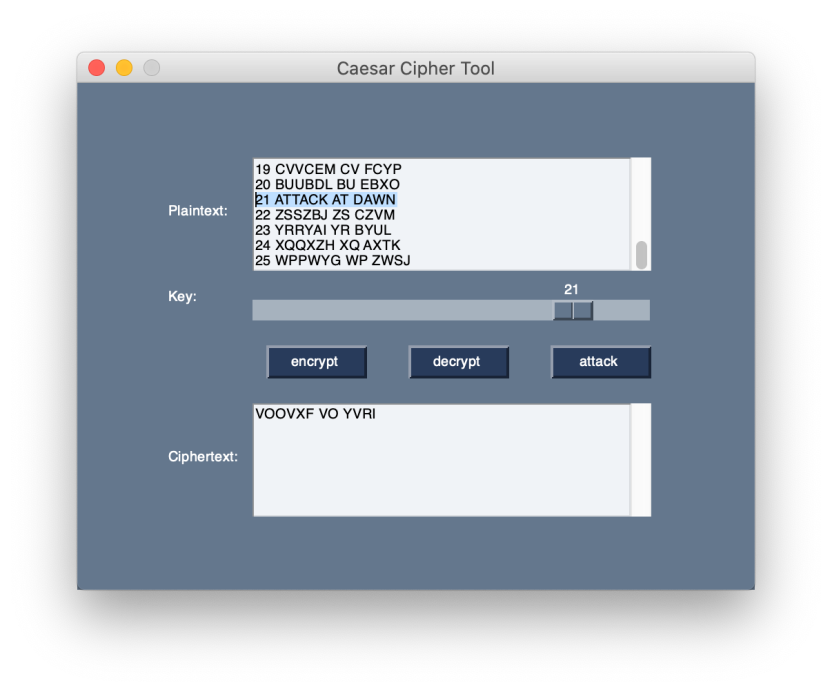


图10 图形化界面暴力破解功能测试图

**六、课题完成过程中遇到的问题及解决方法**

问题1：实现了一个标准版的凯撒密码和改进版的凯撒密码，但是无法区分开这种密码。

解决方法：在每一个密码的class里加上\_\_repr\_\_方法，用以区别。

问题2：图形化界面的排版不整齐

解决方法：修改layout中每一个组件的size值，调整排版。

问题3：报错AttributeError: module 'PySimpleGUI' has no attribute 'WINDOW\_CLOSED'

解决方法：将event路由处的代码，改成if event == None or event == 'Exit':即可正常运行。

1. **总结**

在这两周的时间内，根据给定的题目，我完成了“二叉树的宽度求解问题”的算法设计和“凯撒密码加解密实现”的程序实现。

算法是计算机科学的核心，是程序设计的精髓。在很多问题的解决，程序的编写都要依赖它，在软件还是面向过程的阶段，就有程序=算法+数据结构这个公式。算法的学习对于培养一个人的逻辑思维能力是有极大帮助的，它可以培养我们养成思考分析问题，解决问题的能力。

所谓算法简单来说就是指解题方案的准确而完整的描述，是一系列解决问题的清晰指令，也就是说算法告诉计算机怎么做，以此来解决问题。同一个问题存在多种算法来解决它，但是这些算法存在着优劣之分，好的算法速度快，效率高，占用空间小，差的算法不仅复杂难懂，而且效率低，对机器要求还高，当然，有时候算法之间存在一种互补关系，有些算法效率高，节省时间，但浪费空间，另外一些算法可能速度上慢些，但是空间比较节约，这时候我们就应该根据实际要求，和具体情况来采取相应的算法来解决问题。

如果一个算法有缺陷，或不适合某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂性和时间复杂度来衡量。算法可以使用自然语言、伪代码、流程图等多种不同的方法来描述。计算机系统中的操作系统、语言编译系统、数据库管理系统以及各种各样的计算机应用系统中的软件，都必须使用具体的算法来实现。算法设计与分析是计算机科学与技术的一个核心问题。因此，学习算法无疑会增强自己的竞争力，提高自己的修为，为自己增彩。