Tugas Kecil 1 IF2211 Strategi Algoritma

Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force

Disusun oleh :

13522164 - Valentino Chryslie Triadi

INSTITUT TEKNOLOGI BANDUNG

2024

Daftar Isi

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Brute Force

Algoritma Brute Force adalah salah satu metode pendekatan yang lempang (*straightforward*) untuk memecahkan suatu permasalahan yang sudah terdefinisi. Algoritma Brute Force didasarkan pada pernyataan pada persoalan (*problem statement*) dan konsep yang dilibatkan pada persoalan yang sedang dibahas.

Algoritma Brute Force dapat memecahkan persoalan dengan sangat sederhana, langsung, dan jelas cara pemecahannya (*obvious way*). Algoritma Brute Force pada umumnya memiliki karakteristik tidak mangkus, karena volume komputasi yang diperlukan untuk menyelesaikan masalah yang besar dan waktu yang diperlukan untuk menyelesaikan masalah yang cukup lama. Sehingga, Algoritma Brute Force lebih tepat jika digunakan dalam memecahkan persoalan yang cenderung kecil. Namun, meskipun Algoritma Brute Force dinilai tidak mangkus dalam memecahkan masalah, tetapi Algoritma Brute Force dapat menyelesaikan hampir semua permasalahan yang ada.

1.2 Cyberpunk 2077 Breach Protocol

Cyberpunk 2077 Breach Protocol adalah minigame meretas pada permainan video Cyberpunk 2077. Minigame ini merupakan simulasi peretasan jaringan local dari ICE (Intrusion Countermeasures Electronics) pada permainan Cyberpunk 2077. Permainan ini memiliki beberapa komponen seperti, token (dua karakter alfanumerik), matriks (tersusun atas token-token yang disusun sebagai urutan kode), sekuens (rangkaian dua atau lebih token yang harus dicari atau dicocokan), dan buffer (jumlah maksimal token yang dapat disusun secara sekuensial).

Cyberpunk 2077 Breach Protocol memiliki aturan sebagai berikut pemain harus bergerak dengan pola vertikal → horizontal → vertikal → horizontal → dst (bergantian) hingga semua sekuens berhasil dicocokan atau buffer sudah penuh, pemain memulai dengan memilih satu token pada posisi baris paling atas dari matriks, sekuens dicocokan dengan token-token pada buffer, satu token pada buffer dapat digunakan pada lebih dari satu sekuens, setiap sekuens memiliki bobot yang variatif, setiap sekuens memiliki panjang minimal dua token, dan token pada matriks yang sudah berada di dalam buffer tidak dapat dimasukan kedalam buffer untuk kedua kalinya (posisi token yang masuk ke buffer tidak boleh berulang).

1.3 Algoritma Penyelesaian Cyberpunk 2077 Breach Protocol dengan Pendekatan Brute Force

Dalam menyelesaikan Cyberpunk 2077 Breach Protocol secara algoritmik, penulis menggunakan pendekatan brute force. Adapun langkah penyelesaian permasalahan dalam algoritma adalah sebagai berikut:

1. Pengguna memilih metode pembuatan matriks dan sekuens permainan

(masukan dari file atau dibuat secara acak).

2. Jika pengguna memilih metode masukan file, maka pengguna memasukan file teks (berekstensi .txt). Jika pengguna memilih metode acak, maka pengguna memasukan masukan yang diperlukan.
3. Program akan mencari semua kemungkinan buffer dengan metode *backtracking*. Misalnya, buffer E5-77-1C memiliki lebih dari satu token (BD dan E5) yang sesuai dengan sekuens pada baris atau kolom yang sama, maka pencarian akan dilakukan terhadap E5-77-1C-BD terlebih dahulu apabila tidak sesuai maka baru dilakukan pencarian terhadap E5-77-1C-E5.
4. Untuk mengetahui buffer yang sesuai, program akan menyesuaikan token terakhir dengan token yang harus didapat pada semua sekuens. Misal, terdapat dua sekuens, yaitu BD E5 77 dan 1C 77 E5 maka pada pencarian pertama akan dicari token yang bersesuaian dengan sekuens pertama atau sekuens kedua, yakni token BD atau token 1C. Jika ternyata yang sesuai adalah token 1C, maka pencarian token berikutnya akan disesuaikan dengan token selanjutnya pada sekuens 1C 77 E5, yaitu 77 dan token pertama pada sekuens lainnya.
5. Untuk mencegah terjadinya satu sekuens didapat pada buffer lebih dari satu kali, maka program akan menghapus sekuens yang sudah berhasil didapat sebelum melanjutkan pencarian token.
6. Kemudian, program akan mengeluarkan solusi dengan pendapatan bobot paling besar dan juga akan menyediakan tombol untuk melakukan penyimpanan ke dalam file berekstensi .txt.

BAB II

IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman python. Struktur dari program ini terbagi menjadi 5 file dan 2 folder, yaitu file **main.py, solve.py, GUI.py, data.py, CLI_GUI.py**, folder **input**, dan folder **Component.**

2.1 File main.py

File ini merupakan bagian utama dalam program ini, sehingga hanya memiliki fungsi utama untuk memulai program, yaitu ***startGUI***.

2.2 File solve.py

File ini berisi objek ***SOLVE*** yang memiliki fungsi-fungsi yang berkaitan dengan algoritma penyelesaian dengan pendekatan brute force.

Fungsi	**Deskripsi**
`__init__`	Menginisiasi sebuah objek kelas
`findAllToken`	Mencari semua token yang bersesuaian dengan token yang akan dicari pada baris yang sama jika arah pencarian horizontal atau pada kolom yang sama jika arah pencarian vertikal
`getCurrentBufferNumber`	Mengambil urutan buffer yang akan diisi
`changeDirection`	Mengubah arah pencarian
`getCurrentPositionToken`	Mengambil nilai token pada posisi saat ini
`changeTokentoSearch`	Mengubah list token yang akan dicari
`isDone`	Mengecek apakah pencarian sudah selesai atau belum
`copyList`	Menduplikat list
`findLast`	Melakukan pencarian terakhir
`findNext`	Melakukan pencarian token selanjutnya
`start`	Memulai proses pencarian buffer

2.3 File GUI.py

File ini berisi fungsi-fungsi yang berkaitan dengan tampilan program.

Fungsi / Objek	Deskripsi
`ImageLoader`	Objek untuk melakukan load terhadap aset gambar
`relative_to_assets`	Penghubung folder asset berdasarkan nama file
`closeAllRoots`	Menutup semua window
`resetMain`	Kembali ke menu utama
`main`	Memulai tampilan utama
`getLength`	Mendapatkan panjang buffer
`getParsedResult`	Mengembalikan string yang sudah di proses
`displayCoordinat`	Menampilkan list koordinat
`resetAll`	Me-*reset* semua komponen menjadi kondisi awal dijalankan
`saveResult`	Menyimpan hasil ke file berekstensi .txt
`solve`	Menyelesaikan persoalan
`displayMatrix`	Menampilkan matriks
`displaySequence`	Menampilkan list sekuens
`show`	Menampilkan informasi terkait matriks dan sekuens pada metode masukan file
`showInput`	Menampilkan informasi terkait matriks dan sekuens pada metode acak
`solveInput`	Melakukan pengacakan matriks dan sekuens
`inputFromKeyboard`	Tampilan untuk masukan pengguna

2.4 File data.py

File ini berisi objek kelas ***INFO*** yang memiliki beberapa fungsi sebagai berikut

Fungsi	Deskripsi
`__init__`	Melakukan inisiasi objek ***INFO***
`reset`	Melakukan *reset* terhadap semua variabel
`parse`	Membaca dan melakukan *parsing* pada file masukan

`random`	Melakukan pengacakan matriks dan sekuens (Versi CLI)
`randomGUI`	Melakukan pengacakan matriks dan sekuens (Versi GUI)
`print`	Menampilkan informasi objek (hanya untuk *debugging*)
`solve`	Memulai penyelesaian permainan

2.5 File CLI_GUI.py

File ini berisi tampilan pada versi ***CLI***. Namun, versi yang digunakan adalah versi ***GUI***, sehingga isi dari file ini tidak akan dibahas oleh penulis.

2.6 Folder input

Folder ini berisi file berekstensi .txt yang digunakan sebagai penyimpanan sementara program.

2.7 Folder Component

Folder ini berisi aset-aset gambar yang dibutuhkan oleh tampilan pada program

2.8 Library

Dalam mengembangkan program ini, digunakan beberapa library python sebagai berikut:

- tkinter
- os
- time
- Random

BAB III

SOURCE CODE PROGRAM

3.1 Repository Program

Repository Program dapat diakses melalui tautan *Github* berikut:
ValentinoTriadi/Cyberpunk-2077-Breach-Protocol_Tucil-IF2211-23-24

3.2 Source Code Program

3.2.1 main.py

```
# GUI version
def startGUI():
    import GUI as G
    G.main()

# CLI version
def start():
    import data
    import CLI_GUI
    obj = data.INFO()
    while (True):
        choice = CLI_GUI.main()
        if (choice == 1):
            file_name = CLI_GUI.solveFromFile()
            if (file_name):
                obj.parse(file_name)
                obj.print()
                input("Press Enter to continue...")
                res = obj.solve()
                CLI_GUI.printResult(res, obj.matrix,
obj.time_executed)
                input("Press Enter to continue...")
        elif (choice == 2):
            obj.random(*CLI_GUI.solveFromInput())
            obj.print()
            input("Press Enter to continue...")
            res = obj.solve()
            CLI_GUI.printResult(res, obj.matrix, obj.time_executed)
            input("Press Enter to continue...")
        obj.reset()


if __name__ == "__main__":
    startGUI()
```

3.2.2 solve.py

```
class SOLVE:
    def __init__(self, matrix, sequences, buffer_size) -> None:
        self.buffer_size = buffer_size
        self.matrix = matrix
        self.matrix_row = len(matrix)
        self.matrix_col = len(matrix[0])
        self.sequences = sequences
        self.number_of_sequences = len(self.sequences)
        self.buffer = [() for i in range(self.buffer_size)]
        self.current_position = ()
        self.direction = False # True for horizontal, False for
vertical
        self.token_to_search = [self.sequences[i][0][0] for i in
range(self.number_of_sequences)]
        self.token_to_search_index = [0 for i in
range(self.number_of_sequences)]
        self.acc_token = [[] for i in range(self.buffer_size)]
        self.curr_col = 0
        self.curr_row = 0
        self.result = []
        self.curr_score = 0

    def findAllToken(self):
        temp = []
        max_iterate = self.matrix_col if self.direction else
self.matrix_row
        for i in range(max_iterate):
            token = self.matrix[self.curr_row][i] if self.direction
else self.matrix[i][self.curr_col]
            if token in self.token_to_search:
                if self.direction:
                    if ((self.curr_row,i) not in self.buffer):
                        temp.append((self.curr_row, i))
                else:
                    if ((i,self.curr_col) not in self.buffer):
                        temp.append((i, self.curr_col))
        return temp

    def getCurrentBufferNumber(self):
        temp = 0
        while temp < self.buffer_size and self.buffer[temp] != ():
            temp += 1
        return temp

    def changeDirection(self):
        self.direction = not self.direction

    def getCurrentPositionToken(self):
        return self.matrix[self.curr_row][self.curr_col]

    def changeTokenToSearch(self):
        curr_token = self.getCurrentPositionToken()
        temp = []
        for i in range(self.number_of_sequences):
            if curr_token == self.token_to_search[i]:
                if (self.token_to_search_index[i] <
len(self.sequences[i][0])-1):
                    self.token_to_search_index[i] += 1
                else:
                    self.curr_score += self.sequences[i][1]
                    temp.append(i)
                    self.result.append([self.copyList(self.buffer),
self.curr_score])
            else:
                self.token_to_search_index[i] = 0
            self.token_to_search[i] =
self.sequences[i][0][self.token_to_search_index[i]]
        for i in range(len(temp)):
            for j in range(len(temp)):
                if (i!=j):
                    temp[j] -= 1
            self.token_to_search_index.pop(temp[i])
            self.token_to_search.pop(temp[i])
            self.sequences.pop(temp[i])
            self.number_of_sequences -= 1

    def isDone(self):
        return self.getCurrentBufferNumber() >= self.buffer_size

    def copyList(self, arr):
        temp = []
        for i in arr:
            temp.append(i)
        return temp

    def findLast(self, arr):
        self.changeDirection()
        buff_number = self.buffer_size-1
        temp_score = self.curr_score
        for coor in arr:
            self.acc_token[buff_number].remove(coor)
            self.curr_score = temp_score
            self.buffer[buff_number] = coor
            self.curr_row = coor[0]
            self.curr_col = coor[1]
            self.changeTokenToSearch()
        self.buffer[buff_number] = ()

    def findNext(self, arr):
        self.changeDirection()
        dir = self.direction
        buff_number = self.getCurrentBufferNumber()
        temp_score = self.curr_score
        temp_token_to_search = self.copyList(self.token_to_search)
        temp_token_to_search_index =
self.copyList(self.token_to_search_index)
        temp_sequence = self.copyList(self.sequences)
        temp_number_of_sequences = self.number_of_sequences
        for coor in arr:
            self.direction = dir
            self.acc_token[buff_number].remove(coor)
            self.token_to_search =
self.copyList(temp_token_to_search)
            self.token_to_search_index =
self.copyList(temp_token_to_search_index)
            self.number_of_sequences = temp_number_of_sequences
            self.sequences = self.copyList(temp_sequence)
            self.curr_score = temp_score
            self.buffer[buff_number] = coor
            self.curr_row = coor[0]
            self.curr_col = coor[1]
            self.changeTokenToSearch()
            temp = self.findAllToken() if not self.isDone() else []
            if temp != []:
                if (self.isDone()):
                    # self.result.append([self.buffer,
self.curr_score]) if self.curr_score > 0 else None
                    self.findLast(self.copyList(temp))
                else:
                    self.acc_token[buff_number+1] = temp
                    self.findNext(self.copyList(temp))
        self.buffer[buff_number] = ()



    def start(self):
        for i in range(self.matrix_col):
            self.direction = False
            self.buffer[0] = (0, i)
            self.curr_score = 0
            self.curr_col = i
            self.curr_row = 0
            self.token_to_search = [self.sequences[i][0][0] for i
in range(self.number_of_sequences)]
            self.token_to_search_index = [0 for i in
range(self.number_of_sequences)]
            temp = self.findAllToken()
            if (temp != []):
                self.acc_token[1] = temp
                self.findNext(self.copyList(temp))
```

3.2.3 GUI.py

```
from tkinter import Tk, Canvas, Label, Button, PhotoImage, Frame,
Text, Toplevel
from tkinter.filedialog import askopenfile, asksaveasfile
import data, os, sys
from pathlib import Path

allRoots = []
OUTPUT_PATH = Path(__file__).parent
ASSETS_PATH = OUTPUT_PATH /
Path(fr"{os.getcwd()}\Component\assets")

def relative_to_assets(path: str) -> Path:
    return ASSETS_PATH / Path(path)


class ImageLoader:
    def __init__(self, wd) -> None:
        self.window = wd
        # Load Image
        self.input_image_1 = PhotoImage(
                file=relative_to_assets("image_1.png"))
        self.input_image_2 = PhotoImage(
                file=relative_to_assets("image_3.png"))
        self.input_image_3 = PhotoImage(
                file=relative_to_assets("image_4.png"))
        self.image_image_3 = PhotoImage(
                file=relative_to_assets("image_3.png"))
        self.image_image_4 = PhotoImage(
                file=relative_to_assets("image_4.png"))
        self.button_image_1 = PhotoImage(
                file=relative_to_assets("button_1.png"))
        self.button_image_2 = PhotoImage(
                file=relative_to_assets("button_2.png"))
        self.entry_image_1 = PhotoImage(
                file=relative_to_assets("entry_1.png"))
        self.entry_image_2 = PhotoImage(
                file=relative_to_assets("entry_1.png"))
        self.entry_image_3 = PhotoImage(
                file=relative_to_assets("entry_1.png"))
        self.entry_image_4 = PhotoImage(
                file=relative_to_assets("entry_4.png"))
        self.button_image_solve = PhotoImage(
                file=relative_to_assets("button_solve.png"))
        self.entry_image_5 = PhotoImage(
                file=relative_to_assets("entry_4.png"))
        self.entry_image_6 = PhotoImage(
                file=relative_to_assets("entry_1.png"))
        self.entry_image_7 = PhotoImage(
                file=relative_to_assets("entry_1.png"))

def closeAllRoots():
    global allRoots
    print(allRoots)
    for i in allRoots:
        i.destroy()

def resetMain(window, img_loader, obj):
    canvas = Canvas(
        window,
        bg = "#FFFFFF",
        height = 600,
        width = 1000,
        bd = 0,
        highlightthickness = 0,
        relief = "ridge"
    )

    canvas.place(x = 0, y = 0)

    image_1 = canvas.create_image(
        500.0,
        300.0,
        image=img_loader.input_image_1
    )


    image_2 = canvas.create_image(
        500.0,
        300.0,
        image=img_loader.input_image_2
    )


    image_3 = canvas.create_image(
        523.0,
        325.0,
        image=img_loader.image_image_3
    )


    image_4 = canvas.create_image(
        500.0,
        82.0,
        image=img_loader.image_image_4
    )


    button_1 = Button(
        image=img_loader.button_image_1,
        borderwidth=0,
        highlightthickness=0,
        command=lambda: show(obj),
        relief="flat"
    )
    button_1.place(
        x=249.0,
        y=439.0,
        width=206.0,
        height=48.0
    )


    button_2 = Button(
        image=img_loader.button_image_2,
        borderwidth=0,
        highlightthickness=0,
        command=lambda: inputFromKeyboard(img_loader, window, obj),
        relief="flat"
    )
    button_2.place(
        x=545.0,
        y=439.0,
        width=206.0,
        height=48.0
    )

    canvas.create_text(
        260.0,
        194.0,
        anchor="nw",
        text="Tugas Kecil 1 IF2211 Strategi Algoritma",
        fill="#FFFFFF",
        font=("Poppins Bold", 25 * -1)
    )

    canvas.create_text(
        364.0,
        233.0,
        anchor="nw",
        text="Bruteforce Algorithm",
        fill="#FFFFFF",
        font=("Poppins Bold", 25 * -1)
    )

    canvas.create_text(
        370.0,
        357.0,
        anchor="nw",
        text="Valentino Chryslie Triadi / 13522164",
        fill="#FFFFFF",
        font=("Poppins Regular", 15 * -1)
    )

    canvas.create_text(
        483.0,
        334.0,
        anchor="nw",
        text="Oleh",
        fill="#FFFFFF",
        font=("Poppins Regular", 15 * -1)
    )

def main():
    global allRoots

    obj = data.INFO()

    window = Tk()
    allRoots.append(window)

    window.title("Cyberpunk 2077 Breach Protocol")

    window.geometry("1000x600")
    window.configure(bg = "#FFFFFF")

    img_loader = ImageLoader(window)

    resetMain(window, img_loader, obj)

    window.bind("<Escape>", lambda e: closeAllRoots())
    window.bind("<F11>", lambda e: window.attributes("-fullscreen",
                                                     not
window.attributes("-fullscreen")))
    window.protocol('WM_DELETE_WINDOW', closeAllRoots)  # window is
your window window

    window.resizable(False, False)
    window.mainloop()



# TODO: SHOW RESULT

def getLength(l):
    for i in range(len(l)):
        if l[i]  == ():
            return i
    return len(l)

def getParsedResult(arr, matrix, conn):
    max = arr[0][1]
    temp = []
    for i in arr:
        if (i[1] == max):
            temp.append(i)
    arr = sorted(temp, key=lambda x: getLength(x[0]),
reverse=False)
    res = ""
    for i in range(getLength(arr[0][0])):
        if (i != 0):
            res += (conn)
        res += (matrix[arr[0][0][i][0]][arr[0][0][i][1]])
    return res

def displayCoordinat(arr, ws):
    max = arr[0][1]
    temp = []
    for i in arr:
        if (i[1] == max):
            temp.append(i)
    arr = sorted(temp, key=lambda x: getLength(x[0]),
reverse=False)
    for i in range(getLength(arr[0][0])):
        Label(ws, text=str(arr[0][0][i][1] + 1) + ',' +
str(arr[0][0][i][0] + 1)).grid(row=i, column=1, padx=5, pady=5)
    return True

def resetAll(obj, ws, info_window):
    obj.reset()
    info_window.destroy()
    ws.destroy()
    return True

def saveResult(res, matrix, time_executed):
    f = asksaveasfile(defaultextension=".txt", filetypes=[("Text
Files", "*.txt")])
    f.write(str(res[0][1])+"\n")
    f.write(getParsedResult(res, matrix, " ") + "\n")
    max = res[0][1]
    temp = []
    for i in res:
        if (i[1] == max):
            temp.append(i)
    arr = sorted(temp, key=lambda x: getLength(x[0]),
reverse=False)
    for i in range(getLength(arr[0][0])):
        f.write(str(arr[0][0][i][1] + 1) + ',' +
str(arr[0][0][i][0] + 1) + "\n")
    f.write("\n" + str(round(time_executed*1000)) + " ms")
    f.close()
    return True
```

```
def solve(obj, info_window):
    global allRoots
    res = obj.solve()
    print(res)
    ws = Tk()
    allRoots.append(ws)
    ws.title('Solution Window')
    ws.geometry('1000x600')

    main_frame  =  Frame(ws,  width=1000,  height=600,  bg='grey')
    main_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    first_frame  =  Frame(main_frame,  width=400,  height=500,
bg='grey')
    first_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    second_frame  =  Frame(main_frame,  width=400,  height=500,
bg='grey')
    second_frame.grid(row=0,  column=1,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    third_frame  =  Frame(main_frame,  width=400,  height=500,
bg='grey')
    third_frame.grid(row=0,  column=2,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")


    Label(first_frame, text=f"Execution Time:
{round(obj.time_executed*1000)} ms").grid(row=0, column=0, padx=10,
pady=10, sticky="NW")
    if (res != []):

        Label(first_frame, text=f"Maximum Score:
{res[0][1]}").grid(row=1, column=0, padx=10, pady=10, sticky="NW")

        Label(first_frame, text="Buffer: " + getParsedResult(res,
obj.matrix, " → ")).grid(row=2, column=0, padx=10, pady=10,
sticky="NW")

        Label(second_frame, text="Coordinate:").grid(row=0,
column=0, padx=10, pady=10, sticky="NW")

        displayCoordinat(res, second_frame)

        Button(
            third_frame,
            borderwidth=0,
            highlightthickness=0,
            text= "Save Result",
            command=lambda: saveResult(res, obj.matrix,
obj.time_executed),
            width=30,
        ).grid(row=0, column=0, padx=10, pady=10, sticky="N" + "E"
+ "W" + "S")
    else:
        Label(first_frame, text="No solutions found!").grid(row=1,
column=0, padx=10, pady=10, sticky="NW")

    Button(
        third_frame,
        borderwidth=0,
        highlightthickness=0,
        text= "Back to Main Menu",
        command=lambda: resetAll(obj,ws,info_window),
        width=30,
    ).grid(row=1, column=0, padx=10, pady=10, sticky="N" + "E" +
"W" + "S")

    ws.mainloop()



# TODO: SHOW MATRIX AND SEQUENCE

def displayMatrix(matrix, ws):
    for i in range(len(matrix)):
        for j in range(len(matrix[i])):
            Label(ws, text=matrix[i][j]).grid(row=i, column=j+1,
padx=5, pady=5)
    return True

def displaySequence(sequence, ws):
    print(sequence)
    for i in range(len(sequence)):
        Label(ws, text=sequence[i][0]).grid(row=i, column=1,
padx=5, pady=5, sticky="W")
        Label(ws, text=sequence[i][1]).grid(row=i, column=2,
padx=5, pady=5)
    return True

def show(obj):
    global allRoots
    file_path = askopenfile(mode='r', filetypes=[("Text Files",
"*.txt")])
    text = file_path.read()
    f = open("input/input.txt", "w")
    f.write(text)
    f.close()
    obj.parse("input/input.txt")
    obj.print()

    ws = Tk()
    allRoots.append(ws)
    ws.title('Information Window')
    ws.geometry('1000x600')

    main_frame  =  Frame(ws,  width=1000,  height=600,  bg='grey')
    main_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    first_frame  =  Frame(main_frame,  width=300,  height=500,
bg='grey')
    first_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    second_frame  =  Frame(main_frame,  width=250,  height=500,
bg='grey')
    second_frame.grid(row=0,  column=1,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    third_frame  =  Frame(main_frame,  width=50,  height=500,
bg='grey')
    third_frame.grid(row=0,  column=2,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    Label(first_frame, text="Matrix:").grid(row=0, column=0,
padx=10, pady=10, sticky="NW")
    displayMatrix(obj.matrix, first_frame)

    Label(second_frame, text="Sequence:").grid(row=0, column=0,
padx=10, pady=10, sticky="NW")
    displaySequence(obj.sequences, second_frame)

    Button(third_frame,
           borderwidth=0,
           highlightthickness=0,
           text="START",
           command=lambda: solve(obj, ws),
           ).grid(row=0, column=0, padx=10, pady=10, sticky="N" +
"E" + "W" + "S")

    ws.mainloop()

def showInput(obj):
    global allRoots
    ws = Tk()
    allRoots.append(ws)
    ws.title('Information Window')
    ws.geometry('1000x600')
    main_frame  =  Frame(ws,  width=1000,  height=600,  bg='grey')
    main_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    first_frame  =  Frame(main_frame,  width=300,  height=500,
bg='grey')
    first_frame.grid(row=0,  column=0,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    second_frame  =  Frame(main_frame,  width=250,  height=500,
bg='grey')
    second_frame.grid(row=0,  column=1,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    third_frame  =  Frame(main_frame,  width=50,  height=500,
bg='grey')
    third_frame.grid(row=0,  column=2,  padx=25,  pady=5,
sticky="N"+"E"+"W"+"S")

    Label(first_frame, text="Matrix:").grid(row=0, column=0,
padx=10, pady=10, sticky="NW")
    displayMatrix(obj.matrix, first_frame)

    Label(second_frame, text="Sequence:").grid(row=0, column=0,
padx=10, pady=10, sticky="NW")
    displaySequence(obj.sequences, second_frame)

    Button(third_frame,
           borderwidth=0,
           highlightthickness=0,
           text="START",
           command=lambda: solve(obj, ws),
           ).grid(row=0, column=0, padx=10, pady=10, sticky="N" +
"E" + "W" + "S")

    ws.mainloop()



# TODO: INPUT USER WINDOW

def solveInput(buffer_size, sequence_count, max_sequence_length,
matrix_width, matrix_height, token_count, token):
    obj = data.INFO()
    obj.randomGUI(int(buffer_size.get("1.0", "end-1c")),
int(sequence_count.get("1.0", "end-1c")),
int(max_sequence_length.get("1.0", "end-1c")),
int(matrix_width.get("1.0", "end-1c")),
int(matrix_height.get("1.0", "end-1c")), int(token_count.get("1.0",
"end-1c")), token.get("1.0", "end-1c"))
    showInput(obj)
    return True

def inputFromKeyboard(img_loader, wd, obj):
    window = wd

    window.geometry("1000x600")
    window.configure(bg = "#FFFFFF")

    canvas = Canvas(
        window,
        bg = "#FFFFFF",
        height = 600,
        width = 1000,
        bd = 0,
        highlightthickness = 0,
        relief = "ridge"
    )

    canvas.place(x = 0, y = 0)

    image_1 = canvas.create_image(
        500.0,
        300.0,
        image=img_loader.input_image_1
    )


    image_2 = canvas.create_image(
        523.0,
        325.0,
        image=img_loader.input_image_2
    )



    image_3 = canvas.create_image(
        500.0,
        82.0,
        image=img_loader.input_image_3
    )

    canvas.create_text(
        364.0,
        150.0,
        anchor="nw",
        text="Input From Keyboard",
        fill="#FFFFFF",
        font=("Poppins Bold", 25 * -1)
    )
```

```
entry_bg_1 = canvas.create_image(
    312.5,
    262.0,
    image=img_loader.entry_image_1
)
buffer_size = Text(
    bd=0,
    bg="#F4F4F4",
    fg="#000716",
    highlightthickness=0
)
buffer_size.place(
    x=185.0,
    y=242.0 + 6,
    width=255.0,
    height=25.0
)


entry_bg_2 = canvas.create_image(
    687.5,
    262.0,
    image=img_loader.entry_image_2
)
sequence_count = Text(
    bd=0,
    bg="#F4F4F4",
    fg="#000716",
    highlightthickness=0
)
sequence_count.place(
    x=560.0,
    y=242.0 + 6,
    width=255.0,
    height=25.0
)


entry_bg_3 = canvas.create_image(
    687.5,
    350.0,
    image=img_loader.entry_image_3
)
max_sequence_length = Text(
    bd=0,
    bg="#F4F4F4",
    fg="#000716",
    highlightthickness=0
)
max_sequence_length.place(
    x=560.0,
        y=330.0 + 6,
        width=255.0,
        height=25.0
    )


    entry_bg_4 = canvas.create_image(
        615.0,
        439.0,
        image=img_loader.entry_image_4
    )
    matrix_width = Text(
        bd=0,
        bg="#F4F4F4",
        fg="#000716",
        highlightthickness=0
    )
    matrix_width.place(
        x=560.0,
        y=419.0 + 6,
        width=110.0,
        height=25.0
    )


    solve = Button(
        image=img_loader.button_image_solve,
        borderwidth=0,
        highlightthickness=0,
        command=lambda: solveInput(buffer_size, sequence_count,
max_sequence_length, matrix_width, matrix_height, token_count,
token),
        relief="flat"
    )
    solve.place(
        x=450.0,
        y=496.0,
        width=100.0,
        height=40.0
    )

    back = Button(
        borderwidth=0,
        highlightthickness=0,
        relief="flat",
        text="Back",
        command=lambda: resetMain(wd, img_loader, obj),
    )
    back.place(
        x=50.0,
        y=496.0,
        width=100.0,
        height=40.0
    )


    entry_bg_5 = canvas.create_image(
        760.0,
        439.0,
        image=img_loader.entry_image_5
    )
    matrix_height = Text(
        bd=0,
        bg="#F4F4F4",
        fg="#000716",
        highlightthickness=0
    )
    matrix_height.place(
        x=705.0,
        y=419.0 + 6,
        width=110.0,
        height=25.0
    )


    entry_bg_6 = canvas.create_image(
        312.5,
        350.0,
        image=img_loader.entry_image_6
    )
    token_count = Text(
        bd=0,
        bg="#F4F4F4",
        fg="#000716",
        highlightthickness=0
    )
    token_count.place(
        x=185.0,
        y=330.0 + 6,
        width=255.0,
        height=25.0
    )


    entry_bg_7 = canvas.create_image(
        312.5,
        439.0,
        image=img_loader.entry_image_7
    )
    token = Text(
        bd=0,
        bg="#F4F4F4",
        fg="#000716",
        highlightthickness=0
    )
    token.place(
        x=185.0,
        y=419.0 + 6,
        width=255.0,
        height=25.0
    )

    canvas.create_text(
        175.0,
        219.0,
        anchor="nw",
        text="Buffer Size",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        550.0,
        219.0,
        anchor="nw",
        text="Sequence Count",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        550.0,
        307.0,
        anchor="nw",
        text="Maximum Sequence Length",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        550.0,
        396.0,
        anchor="nw",
        text="Matrix Width",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        695.0,
        396.0,
        anchor="nw",
        text="Matrix Height",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        175.0,
        307.0,
        anchor="nw",
        text="Unique Token Count ",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        175.0,
        396.0,
        anchor="nw",
        text="Token",
        fill="#FFFFFF",
        font=("Poppins Bold", 15 * -1)
    )

    canvas.create_text(
        175.0,
        459.0,
        anchor="nw",
        text="Note: Separate each token with space (E5 1C 77 ...)",
        fill="#FFFFFF",
        font=("Poppins Regular", 10 * -1)
    )

    window.resizable(False, False)
    window.mainloop()
```

3.2.4 data.py

```
import random, os, time
import solve

class INFO:
    def __init__(self):
        self.reset()
        self.time_executed = 0

    def reset(self):
        self.buffer_size = 0
        self.matrix_col = 0
        self.matrix_row = 0
        self.number_of_sequences = 0
        self.sequences = []
        self.matrix = []

    def parse(self, file):
        with open(file, 'r') as f:
            # Read buffer size
            line = f.readline()
            self.buffer_size = int(line.rstrip())

            # Read matrix width and height
            line = f.readline()
            self.matrix_col, self.matrix_row = map(int,
line.rstrip().split(' '))

            # Read Matrix
            for i in range(self.matrix_row):
                line = f.readline()
                self.matrix.append(list(line.rstrip().split(' ')))

            # Read number of sequences
            line = f.readline()
            self.number_of_sequences = int(line.rstrip())

            # Read sequences and scores
            for i in range(self.number_of_sequences):
                line = f.readline()
                sequence = line.rstrip().split(' ')
                line = f.readline()
                score = int(line.rstrip())
                self.sequences.append([sequence, score])

    def random(self, jumlah_token_unik, token, buffer_size,
matrix_col, matrix_row, number_of_sequences,
max_number_of_sequence):
        self.buffer_size = buffer_size
        self.matrix_col = matrix_col
        self.matrix_row = matrix_row
        self.number_of_sequences = number_of_sequences
        for i in range(matrix_row):
            self.matrix.append([random.choice(token) for j in
range(matrix_col)])
        for i in range(number_of_sequences):
            sequence = [random.choice(token) for j in
range(random.randint(2, max_number_of_sequence))]
            score = random.randint(1, 100)
            self.sequences.append([sequence, score])

    def randomGUI(self, buffer_size, sequence_count,
max_sequence_length, matrix_width, matrix_height, token_count,
token):
        unique_token = ["" for i in range(token_count)]
        token = token.strip().split(' ')
        for i in range(token_count):
            unique_token[i] = token[i]
        self.buffer_size = buffer_size
        self.matrix_col = matrix_width
        self.matrix_row = matrix_height
        self.number_of_sequences = sequence_count
        for i in range(matrix_height):
            self.matrix.append([random.choice(unique_token) for j
in range(matrix_width)])
        for i in range(sequence_count):
            sequence = [random.choice(unique_token) for j in
range(random.randint(2, max_sequence_length))]
            score = random.randint(1, 100)
            self.sequences.append([sequence, score])


    def print(self):
        os.system("cls || clear")
        print("buffer size: ",self.buffer_size)
        print("matrix width: ",self.matrix_col)
        print("matrix height: ",self.matrix_row)
        print("matrix: ")
        for i in self.matrix:
            for j in i:
                print(j, end=' ')
            print()

        print("number of sequences:",self.number_of_sequences)
        print("sequences: ")
        for i in self.sequences:
            print(f"Score: {i[1]:3d}", end='  →  ')
            for j in i[0]:
                print(j, end=' ')
            print()

    def solve(self):
        # ide nya cari kebawah kalo directionnya horizontal, kalo
nemu gas turun kalo ga nemu lanjut ke baris sampingnya
        os.system("cls || clear")
        print("Solving the problem...")
        solveClass = solve.SOLVE(self.matrix, self.sequences,
self.buffer_size)
        start = time.time()
        solveClass.start()
        end = time.time()
        self.time_executed = end-start
        # print(res)
        # print(sorted(res, key = lambda x: x[1], reverse=True))
        return (sorted(solveClass.result, key = lambda x: x[1],
reverse=True))
```

BAB IV

MASUKAN DAN LUARAN PROGRAM

4.1 Input File

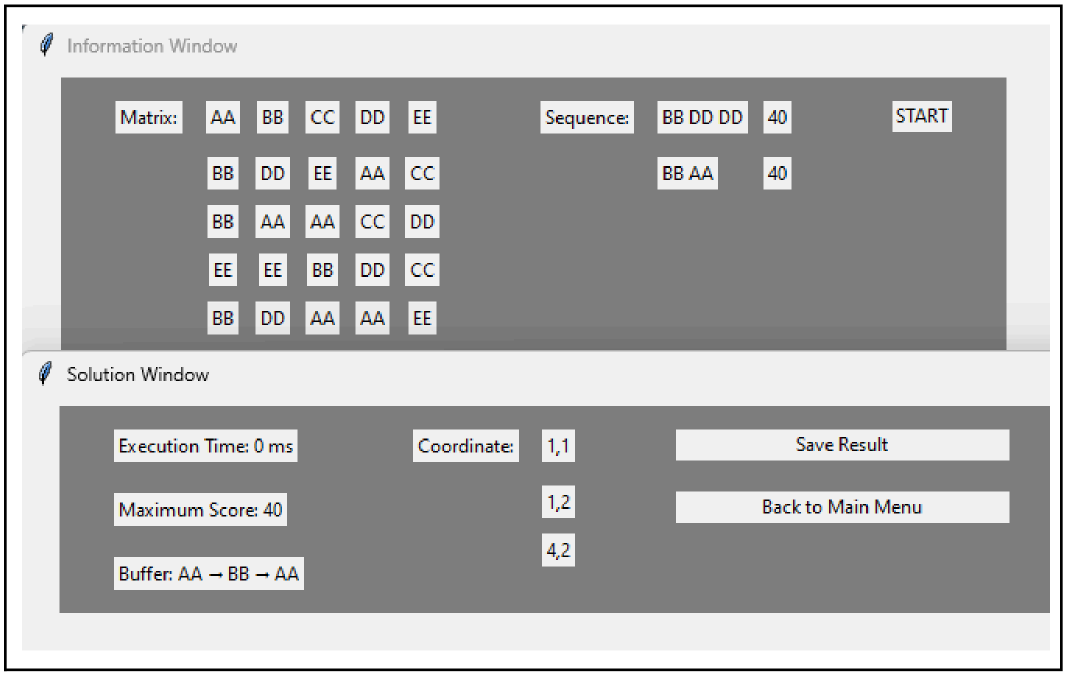
a.

Masukan:

```
4
5 5
AA BB CC DD EE
BB DD EE AA CC
BB AA AA CC DD
EE EE BB DD CC
BB DD AA AA EE
2
BB DD DD
40
BB AA
40
```

Keluaran:

```
40
AA BB AA
1,1
1,2
4,2

0 ms
```


Information Window

Matrix: AA BB CC DD EE
BB DD EE AA CC
BB AA AA CC DD
EE EE BB DD CC
BB DD AA AA EE

Sequence: BB DD DD 40
BB AA 40

START

Solution Window

Execution Time: 0 ms
Maximum Score: 40
Buffer: AA → BB → AA

Coordinate: 1,1
1,2
4,2

Save Result
Back to Main Menu

b.

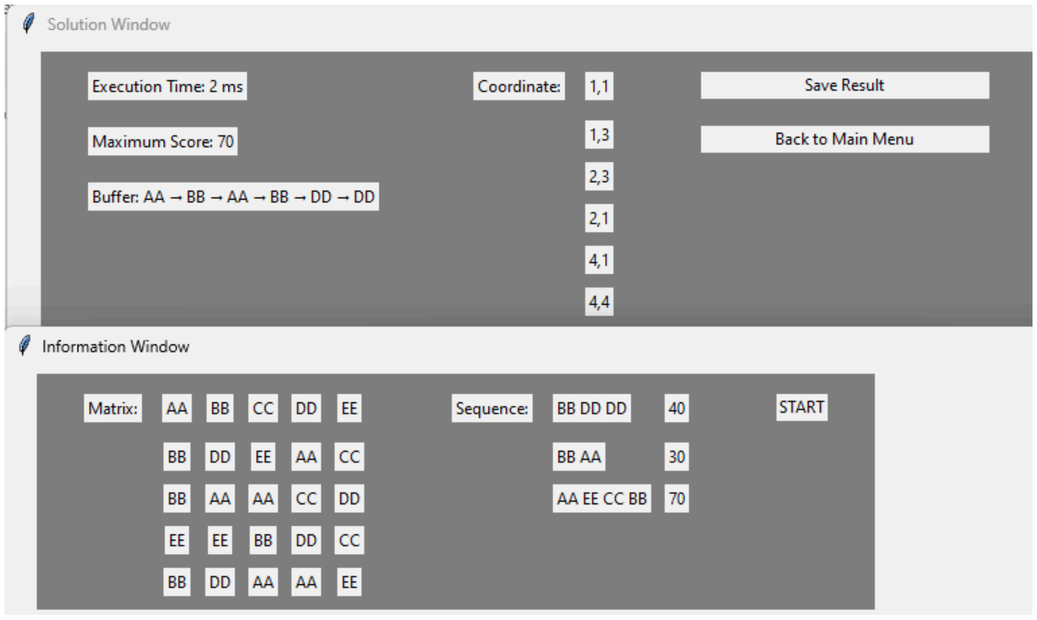
Masukan:

```
7
5 5
AA BB CC DD EE
BB DD EE AA CC
BB AA AA CC DD
EE EE BB DD CC
BB DD AA AA EE
3
BB DD DD
40
BB AA
30
AA EE CC BB
70
```

Keluaran:

```
70
AA BB AA BB DD DD
1,1
1,3
2,3
2,1
4,1
4,4
```

```
2 ms
```



Solution Window

Execution Time: 2 ms
Maximum Score: 70
Buffer: AA → BB → AA → BB → DD → DD
Coordinate: 1,1
1,3
2,3
2,1
4,1
4,4

Information Window

Matrix:
AA BB CC DD EE
BB DD EE AA CC
BB AA AA CC DD
EE EE BB DD CC
BB DD AA AA EE

Sequence:
BB DD DD 40
BB AA 30
AA EE CC BB 70

c.

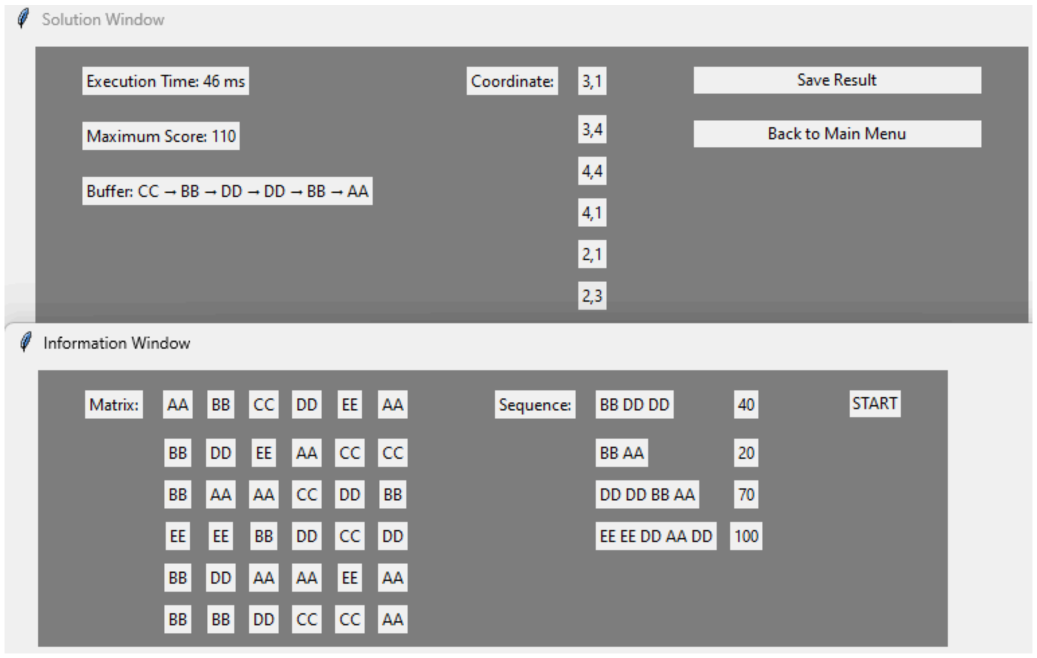
Masukan:

```
7
6 6
AA BB CC DD EE AA
BB DD EE AA CC CC
BB AA AA CC DD BB
EE EE BB DD CC DD
BB DD AA AA EE AA
BB BB DD CC CC AA
4
BB DD DD
40
BB AA
20
DD DD BB AA
70
EE EE DD AA DD
100
```

Keluaran:

```
110
CC BB DD DD BB AA
3,1
3,4
4,4
```

```
4,1
2,1
2,3

46 ms
```



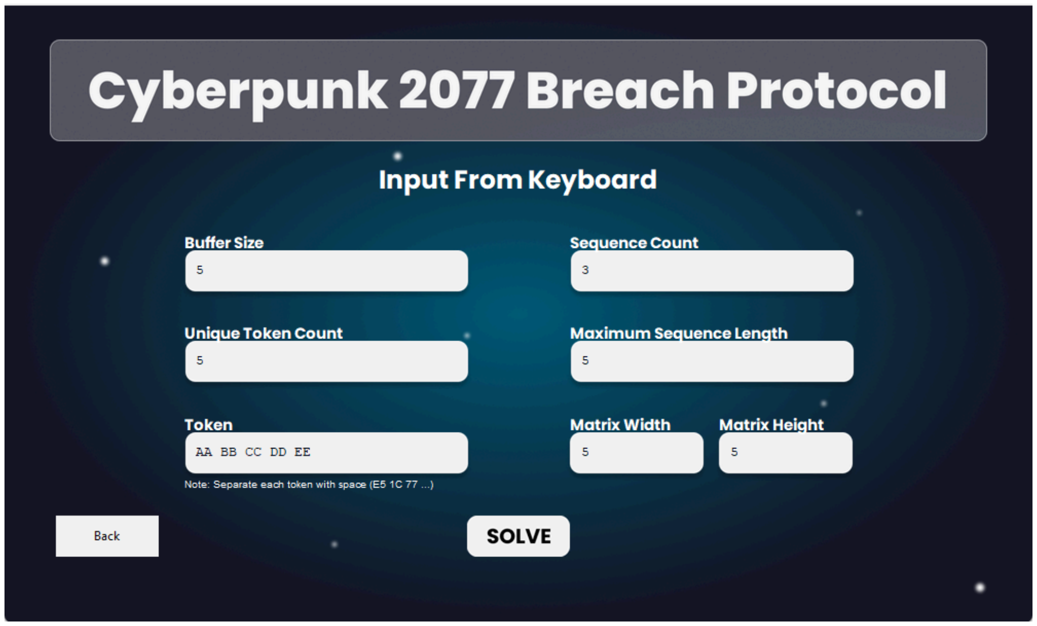
```
Execution Time: 46 ms        Coordinate: 3,1
Maximum Score: 110                       3,4
Buffer: CC → BB → DD → DD → BB → AA      4,4
                                         4,1
                                         2,1
                                         2,3

Matrix: AA BB CC DD EE AA    Sequence: BB DD DD        40
        BB DD EE AA CC CC              BB AA           20
        BB AA AA CC DD BB              DD DD BB AA     70
        EE EE BB DD CC DD              EE EE DD AA DD  100
        BB DD AA AA EE AA
        BB BB DD CC CC AA
```

4.2 Acak

a.

Masukan:

```
5
AA BB CC DD EE
5
5 5
3
5
```



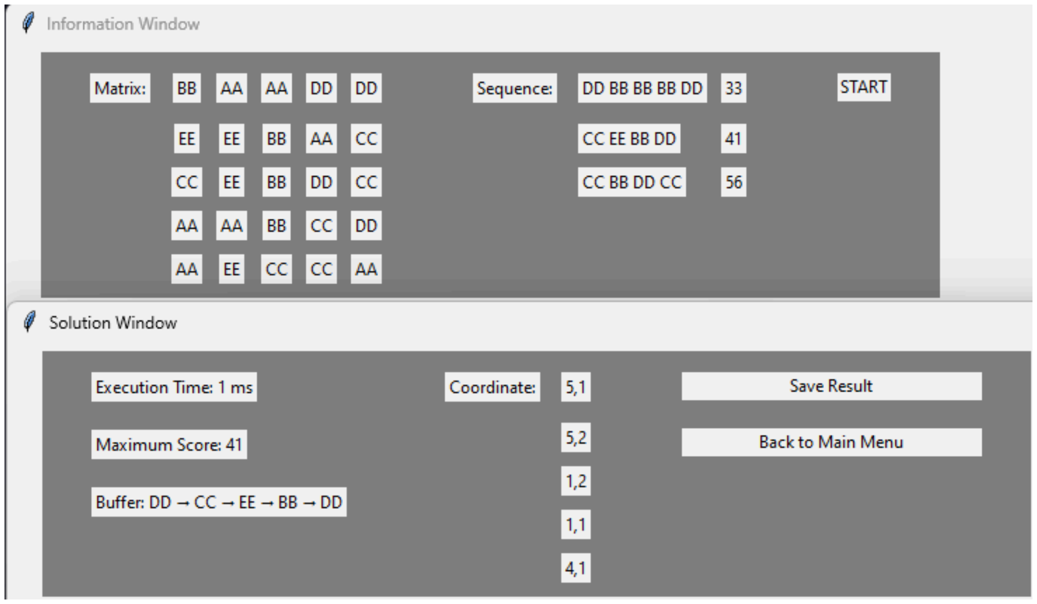
Cyberpunk 2077 Breach Protocol

Input From Keyboard

Buffer Size: 5
Sequence Count: 3
Unique Token Count: 5
Maximum Sequence Length: 5
Token: AA BB CC DD EE
Note: Separate each token with space (E5 1C 77 ...)
Matrix Width: 5
Matrix Height: 5

Keluaran:

```
41
DD CC EE BB DD
5,1
5,2
1,2
1,1
4,1

1 ms
```



Information Window

Matrix:

BB	AA	AA	DD	DD
EE	EE	BB	AA	CC
CC	EE	BB	DD	CC
AA	AA	BB	CC	DD
AA	EE	CC	CC	AA

Sequence:

DD BB BB BB DD	33
CC EE BB DD	41
CC BB DD CC	56

Solution Window

Execution Time: 1 ms

Maximum Score: 41

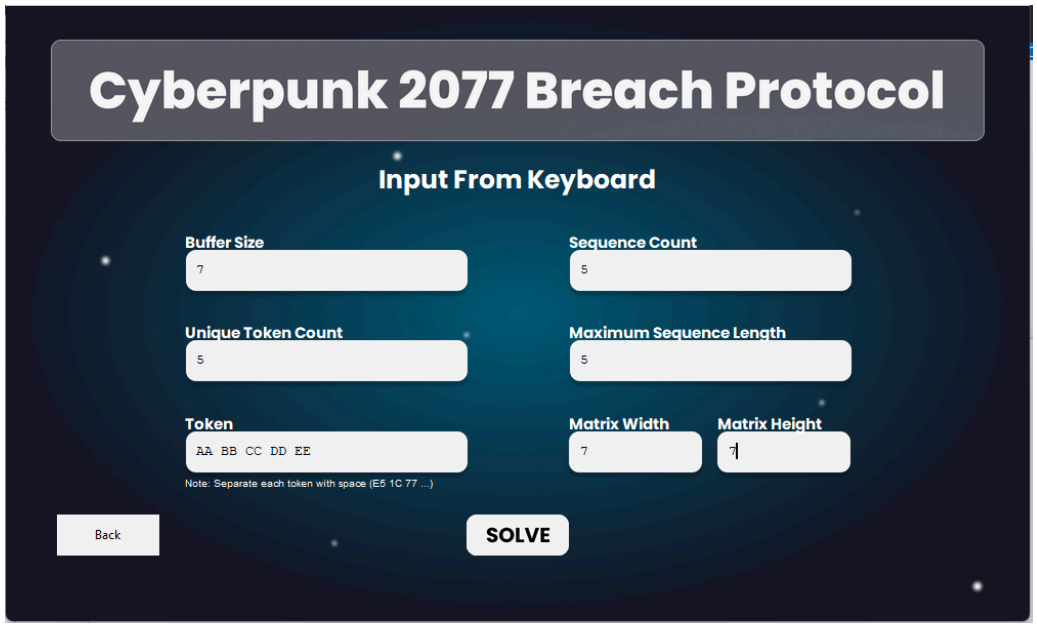
Buffer: DD → CC → EE → BB → DD

Coordinate: 5,1 5,2 1,2 1,1 4,1

b.

Masukan:

```
7
AA BB CC DD EE
7
7 7
5
5
```



Cyberpunk 2077 Breach Protocol

Input From Keyboard

Buffer Size: 7

Sequence Count: 5

Unique Token Count: 5

Maximum Sequence Length: 5

Token: AA BB CC DD EE

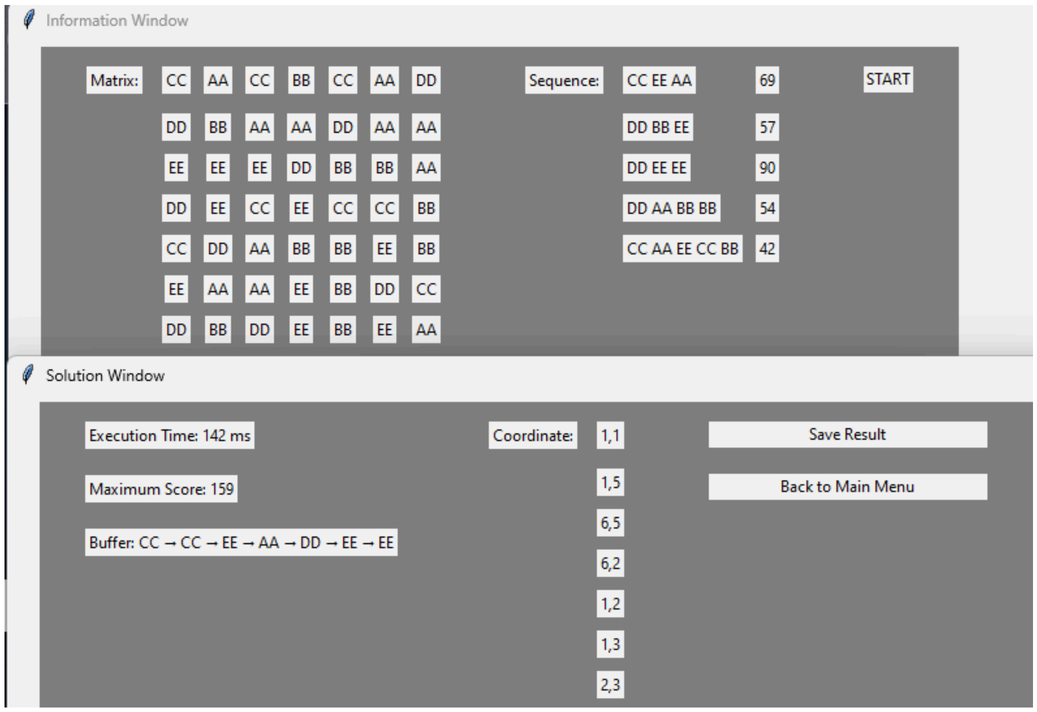
Note: Separate each token with space (E5 1C 77 ...)

Matrix Width: 7

Matrix Height: 7

Keluaran:

```
159
CC CC EE AA DD EE EE
1,1
1,5
6,5
6,2
1,2
1,3
2,3

142 ms
```

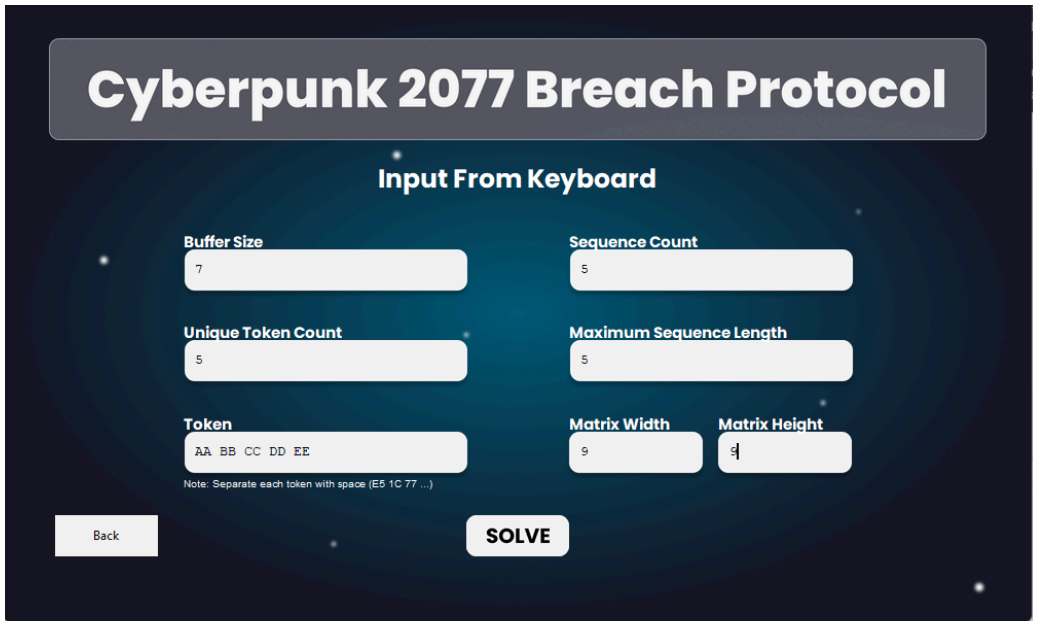


```
Information Window

Matrix:  CC AA CC BB CC AA DD        Sequence:  CC EE AA        69     START
         DD BB AA AA DD AA AA                   DD BB EE        57
         EE EE EE DD BB BB AA                   DD EE EE        90
         DD EE CC EE CC CC BB                   DD AA BB BB     54
         CC DD AA BB BB EE BB                   CC AA EE CC BB  42
         EE AA AA EE BB DD CC
         DD BB DD EE BB EE AA

Solution Window

Execution Time: 142 ms                     Coordinate:  1,1     Save Result

Maximum Score: 159                                      1,5     Back to Main Menu

Buffer: CC → CC → EE → AA → DD → EE → EE                6,5

                                                        6,2

                                                        1,2

                                                        1,3

                                                        2,3
```

c.

Masukan:

```
7
AA BB CC DD EE
7
9 9
5
5
```



Cyberpunk 2077 Breach Protocol

Input From Keyboard

Buffer Size: 7

Sequence Count: 5

Unique Token Count: 5

Maximum Sequence Length: 5

Token: AA BB CC DD EE

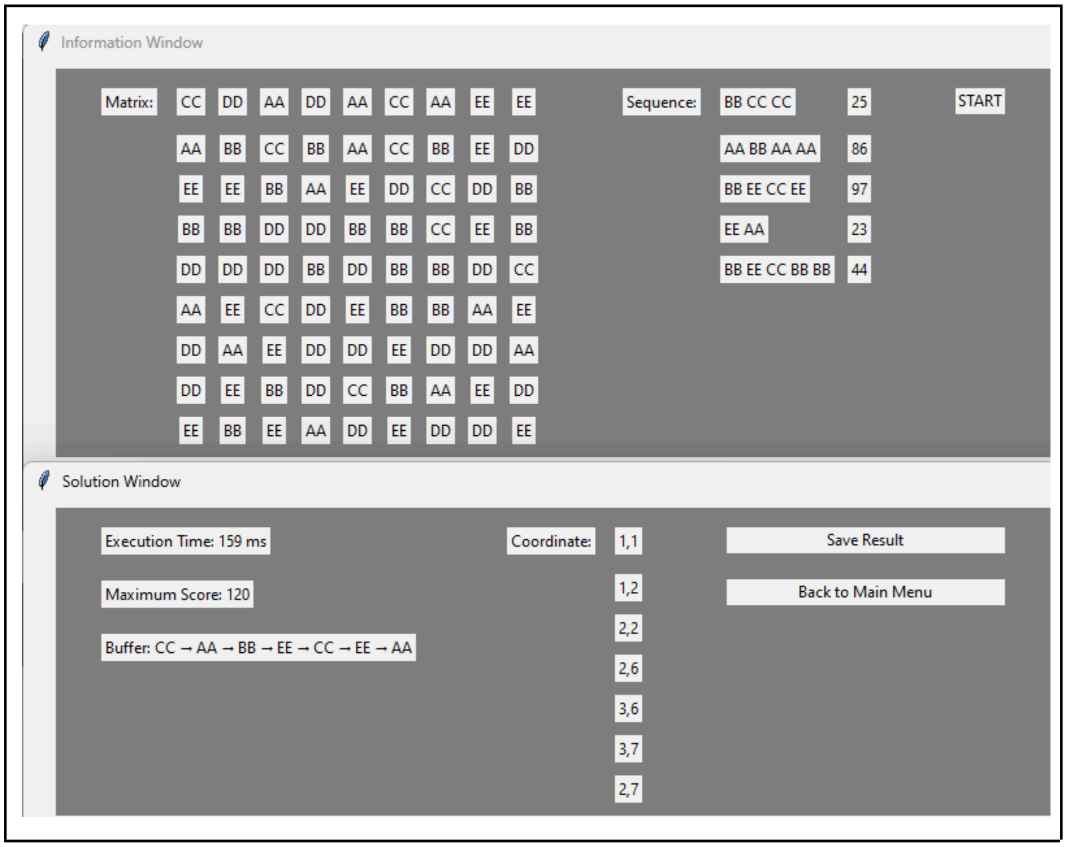
Note: Separate each token with space (E5 1C 77 ...)

Matrix Width: 9

Matrix Height: 9

Keluaran:

```
120
CC AA BB EE CC EE AA
1,1
1,2
2,2
2,6
3,6
3,7
2,7

159 ms
```



Information Window

Matrix:

CC	DD	AA	DD	AA	CC	AA	EE	EE
AA	BB	CC	BB	AA	CC	BB	EE	DD
EE	EE	BB	AA	EE	DD	CC	DD	BB
BB	BB	DD	DD	BB	BB	CC	EE	BB
DD	DD	DD	BB	DD	BB	BB	DD	CC
AA	EE	CC	DD	EE	BB	BB	AA	EE
DD	AA	EE	DD	DD	EE	DD	DD	AA
DD	EE	BB	DD	CC	BB	AA	EE	DD
EE	BB	EE	AA	DD	EE	DD	DD	EE

Sequence:

BB CC CC	25
AA BB AA AA	86
BB EE CC EE	97
EE AA	23
BB EE CC BB BB	44

START

Solution Window

Execution Time: 159 ms

Maximum Score: 120

Buffer: CC → AA → BB → EE → CC → EE → AA

Coordinate:

1,1
1,2
2,2
2,6
3,6
3,7
2,7

Save Result

Back to Main Menu

BAB V

LAMPIRAN

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program dapat membaca masukan berkas .txt	✓	
4. Program dapat menghasilkan masukan secara acak	✓	
5. Solusi yang diberikan program optimal	✓	
6. Progam dapat menyimpan solusi dalam berkas .txt	✓	
7. Program memiliki GUI	✓	

Referensi

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf