Tugas Kecil 1 - IF2211 Strategi dan Algoritma
Penyelesaian Cyberpunk 2077 Breach Protocol
dengan Algoritma Brute Force

Disusun oleh:
Muhammad Dava Fathurrahman (13522114)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024

Daftar Isi

Daftar Isi **2**

BAB 1 **3**

Algoritma Brute Force 3

1.1. Pendahuluan 3

1.2. Deskripsi Langkah 3

BAB 2 **4**

Kode Sumber 4

BAB 3 **18**

Testing 18

3.1. Masukan Manual dan Acak 18

3.2. Masukan File 19

3.3. Simpan File 19

3.4. Ukuran Matriks Tidak Sesuai 20

3.5. Token Tidak Alfanumerik 20

LAMPIRAN **21**

BAB 1

Algoritma Brute Force

1.1. Pendahuluan

Brute force adalah sebuah pendekatan yang lempang (straightforward) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (problem statement) dan definisi konsep yang dilibatkan. Algoritma Brute Force memecahkan suatu persoalan dengan metode pencarian solusi yang langsung dan dengan cara yang jelas. Algoritma ini secara eksplisit mencoba semua kemungkinan solusi.

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. Komponen pada permainan ini terdiri atas

1. Token, dua karakter alfanumerik seperti E9, BD, dan 55.
2. Matriks, terdiri atas token-token yang akan dipilih untuk menyusun urutan kode.
3. Sekuens, sebuah rangkaian token (dua atau lebih) yang harus dicocokkan.
4. Buffer, jumlah maksimal token yang dapat disusun secara sekuensial.

Pemain akan mengikuti aturan yang meliputi gerakan vertikal dan horizontal secara bergantian, dimulai dengan memilih satu token di posisi teratas matriks kemudian mencocokkan sekuens pada token dalam buffer. Satu token dapat digunakan untuk beberapa sekuens dan setiap sekuens memiliki bobot hadiah yang bervariasi, dengan panjang minimal sekuens adalah dua token.

1.2. Deskripsi Langkah

Dengan pendekatan brute force dan fungsi rekursif, program akan membentuk sekuens hingga sepanjang ukuran buffer jika memungkinkan. Pada sumber program, algoritma brute force terdapat pada fungsi move(). Token pertama (langkah ke-1) dipilih dari baris-baris teratas matriks. Kemudian, sel yang telah dipilih akan mengunjungi salah satu sel token di bawah nya sehingga membentuk rangkaian token (sekuens). Langkah di atas diulangi dengan gerakan secara horizontal (jika langkah pada kode sumber adalah ganjil) atau vertikal (jika langkah pada kode sumber adalah genap). Setiap rangkaian baru akan dilakukan pengecekan hadiah pada fungsi checkScore() berdasarkan bobot sekuens. Rekursif akan berhenti jika langkah telah sebanyak ukuran buffer atau seluruh token di sekitarnya telah dikunjungi. Penulis menandai token yang telah dikunjungi dengan membuat cermin matriks dan memberi sel matriks dengan nomor atau langkah kunjungan. Selain itu, penulis membatasi pembuatan sekuens secara acak dengan maksimum 10.000.000. Hal tersebut disebabkan pustaka Tkinter yang terganggu oleh senarai yang menampung begitu banyak produk katersian, serta kinerja yang mulai melambat.

BAB 2

Kode Sumber

```
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from math import ceil
from random import randrange, randint
from itertools import product
import time

uniqToken = []; mtx_input = []; sequences = {}
best_binMtx = []; best_sequence = []; entries_token = {}
entries_mtx = {}


def clearWidget():
    global buffer_size, N_uniqToken, uniqToken, mtx_width, mtx_height,
best_score, row_offset, num_rows
    mtx_width = 0; mtx_height = 0
    buffer_size = 0; N_uniqToken = 0
    uniqToken.clear(); mtx_input.clear(); sequences.clear()
    best_score = float('-inf')

    for entry in entries_mtx.values():
        entry.destroy()

    best_binMtx.clear(); best_sequence.clear(); entries_mtx.clear()
    row_offset = 3
    num_rows = 0
    resultSeq.delete("1.0", tk.END);
    resultPath.delete("1.0", tk.END)
    resultSeq.grid_remove()
    resultPath.grid_remove()
    save_button.grid_remove()


def writeBinMtx(matrix, file, step=1):
    row = 0; col = 0
    while step <= len(best_sequence):
        if step % 2 == 0:
            for i in range(mtx_height):
                if matrix[i][col] == step:
                    row = i
                    file.write(f"{col+1}, {row+1}\n")
        else:
            for j in range(mtx_width):
                if matrix[row][j] == step:
                    col = j
                    file.write(f"{col+1}, {row+1}\n")
        step += 1


def read_token():
    global N_uniqToken
    uniqToken.clear()

    N_uniqToken = int(box_NToken.get())
    width = 0 if box_NToken.get()=='' else int(box_NToken.get()[:2])
    valid = True

    current_cell = 0
    for row in range(num_rows + 1):
        for col in range(3, min(10, width - current_cell) + 3):
            token = entries_token[(row, col)].get()
            if token not in uniqToken and token.isalnum() and len(token)==2:
                uniqToken.append(token)
            else:
                entries_token[((row, col))].delete(0 ,'end')
                valid = False
            current_cell += 1
    if not valid:
        uniqToken.clear()

    return valid


def read_Matrix():
    global mtx_height, mtx_width, mtx_input
    mtx_input.clear()

    mtx_height = 0 if box_rows.get()=='' else int(box_rows.get())
    mtx_width = 0 if box_cols.get()=='' else int(box_cols.get())

    valid = True
    for i in range(mtx_height):
        row = []
        for j in range(mtx_width):
            try:
                value = entries_mtx[(i, j)].get()
                if value in uniqToken:
                    row.append(value)
                else:
                    entries_mtx[(i, j)].delete(0 ,'end')
                    valid = False
            except KeyError:
                messagebox.showinfo('Matrix Size', 'Matrix size not updated!')
                valid = False
        mtx_input.append(row)

    if not valid:
        mtx_input.clear()

    return valid

def updateBuffer(event):
    try:
        x = int(box_buffer.get())
        if x < 2:
            box_buffer.delete(0 ,'end')
    except:
        box_buffer.delete(0 ,'end')
```

```
def updateNSeq(event):
    try:
        x = int(box_NSeq.get())
        if x < 1:
            box_NSeq.delete(0 ,'end')
    except:
        box_NSeq.delete(0 ,'end')


def updateMaxSeq(event):
    try:
        x = int(box_MaxSeq.get())
        if x < 2:
            box_MaxSeq.delete(0 ,'end')
    except:
        box_MaxSeq.delete(0 ,'end')

def updateTokens(event):
    clearWidget()
    global row_offset, num_rows

    width = 0 if box_NToken.get()=='' else int(box_NToken.get()[:2])

    for entry in entries_token.values():
        entry.destroy()
    entries_token.clear()

    num_rows = ceil(width / 10)
    current_cell = 0

    for row in range(num_rows + 1):
        for col in range(3, min(10, width - current_cell) + 3):
            entry = tk.Entry(root, width=5, justify='center')
            entry.grid(row=row, column=col, sticky="nsew")
            entries_token[(row, col)] = entry
            current_cell += 1

    row_offset = num_rows + 1

    label_buffer.grid(row=row_offset, column=0)
    box_buffer.grid(row=row_offset, column=1)
    label_rows.grid(row=row_offset+1, column=0)
    box_rows.grid(row=row_offset+1, column=1)

    label_cols.grid(row=row_offset+1, column=2)
    box_cols.grid(row=row_offset+1, column=3)

    label_NSeq.grid(row=row_offset+2, column=0)
    box_NSeq.grid(row=row_offset+2, column=1)

    label_MaxSeq.grid(row=row_offset+2, column=2)
    box_MaxSeq.grid(row=row_offset+2, column=3)

    browse_button.grid(row=row_offset+3, column=1)
    generate_button.grid(row=row_offset+3, column=2)
    updateMtxSize()


def updateMtxSize():
    global mtx_height, mtx_width
    if read_token():
        mtx_input.clear()
        sequences.clear()
        resultSeq.delete("1.0", tk.END)
        resultPath.delete("1.0", tk.END)

        mtx_height = 0 if box_rows.get()=='' else int(box_rows.get())
        mtx_width = 0 if box_cols.get()=='' else int(box_cols.get())

        for entry in entries_mtx.values():
            entry.destroy()
        entries_mtx.clear()

        for i in range(mtx_height):
            for j in range(mtx_width):
                entry = tk.Entry(root, width=5, justify='center')
                entry.grid(row=i + row_offset+4, column=j+3, sticky="nsew")
                entries_mtx[(i, j)] = entry

        if mtx_height != 0:
            resultSeq.grid(row=mtx_height+ row_offset+4, column=3,
columnspan=7)
```

```
            save_button.grid(row=row_offset+3, column=13)
            resultPath.grid(row=row_offset+4, column=13, rowspan=10)

        return True
    else:
        mtx_input.clear()

        for entry in entries_mtx.values():
            entry.destroy()
        resultSeq.grid_remove()

        entries_mtx.clear()

        return False


def insertMtx():
    mtx_input.clear()
    valid = True

    for i in range(mtx_height):
        row = []
        for j in range(mtx_width):
            try:
                value = uniqToken[randint(0, N_uniqToken-1)]
                entries_mtx[(i, j)].insert(0, value)
                row.append(value)
            except KeyError:
                messagebox.showerror('Invalid Matrix Size', 'Invalid Matrix
Size')
                valid = False
        mtx_input.append(row)

    if not valid:
        mtx_input.clear()

    return valid


def generateProduct(L, r):
    est_len = sum(len(L) ** x for x in range(2, r + 1))
    prod = []
    if est_len <= 10000000:
        for r in range(2, r+1):
            prod.extend(product(L, repeat=r))

    return prod


def genSequences():
    sequences.clear()

    N_seq = int(box_NSeq.get())
    max_seq = int(box_MaxSeq.get())

    while True:
        tokenProduct = generateProduct(uniqToken, max_seq)

        if N_seq <= len(tokenProduct):
            for _ in range(N_seq):
                idx = randint(0, len(tokenProduct)-1)
                val = str(list(tokenProduct[idx]))

                sequences[val] = randrange(-200, 200, 10)
                resultSeq.insert(tk.END, ' '.join(eval(val)) + ' : ' +
str(sequences[val]) + '\n')
                tokenProduct.pop(idx)
            break
        else:
            messagebox.showinfo('Sequence Failed!', f'Cannot create {N_seq}
sequences')
            break


def checkScore(temp_seq, binMtx):
    global best_score, best_binMtx, best_sequence
    score = 0; flag_occur = False

    for key, value in sequences.items():
        key = eval(key)
        occur = len([key for idx in range(len(temp_seq)) if temp_seq[idx : idx
+ len(key)] == key])
```

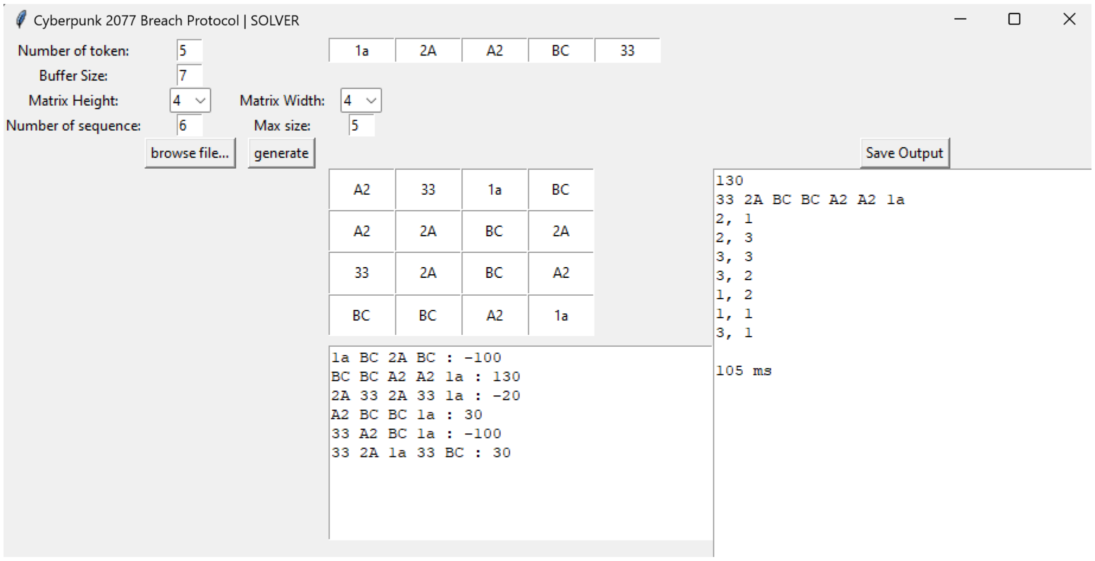
```
        if occur:
            flag_occur = True
        score += occur * value


    if flag_occur and score > 0:
        if score > best_score:
            best_score = score
            best_sequence = [_ for _ in temp_seq]
            best_binMtx = [row[:] for row in binMtx]
        elif score == best_score:
            if len(temp_seq) < len(best_sequence):
                best_score = score
                best_sequence = [_ for _ in temp_seq]
                best_binMtx = [row[:] for row in binMtx]


def move(step=1, row=0, col=0, temp_seq=[], binMtx=None):
    if step == (buffer_size+1):
        return

    # ganjil horizontal
    if step % 2 != 0:
        for i in range(col, mtx_width):
            if binMtx[row][i] == 0:
                temp_seq.append(mtx_input[row][i])
                binMtx[row][i] = step

                checkScore(temp_seq, binMtx)

                move(step + 1, row, i, temp_seq, binMtx)

                temp_seq.pop()
                binMtx[row][i] = 0

        for i in range(col-1, -1, -1):
            if binMtx[row][i] == 0:
                temp_seq.append(mtx_input[row][i])
                binMtx[row][i] = step

                checkScore(temp_seq, binMtx)
                move(step + 1, row, i, temp_seq, binMtx)

                temp_seq.pop()
                binMtx[row][i] = 0

    # genap vertikal
    else:
        for i in range(row, mtx_height):
            if binMtx[i][col] == 0:
                temp_seq.append(mtx_input[i][col])
                binMtx[i][col] = step

                checkScore(temp_seq, binMtx)

                move(step + 1, i, col, temp_seq, binMtx)

                temp_seq.pop()
                binMtx[i][col] = 0

        for i in range(row-1, -1, -1):
            if binMtx[i][col] == 0:
                temp_seq.append(mtx_input[i][col])
                binMtx[i][col] = step

                checkScore(temp_seq, binMtx)

                move(step + 1, i, col, temp_seq, binMtx)

                temp_seq.pop()
                binMtx[i][col] = 0


def startSolver():
    global end_time
    end_time = 0
    binMtx = [[0 for _ in range(mtx_width)] for __ in range(mtx_height)]

    start_time = time.time()
    move(binMtx=binMtx)

    if best_score == float('-inf'):
        resultPath.insert(tk.END, '0\n')
    else:
        resultPath.insert(tk.END, str(best_score) + '\n')
        resultPath.insert(tk.END, ' '.join(best_sequence) + '\n')


        step=1; row = 0 ; col = 0
        while step <= len(best_sequence):
            # vertikal
            if step % 2 == 0:
                for i in range(mtx_height):
                    if best_binMtx[i][col] == step:
                        row = i
                        resultPath.insert(tk.END, str(col+1) + ', ' +
str(row+1) + '\n')
            # horizontal
            else:
                for j in range(mtx_width):
                    if best_binMtx[row][j] == step:
                        col = j
                        resultPath.insert(tk.END, str(col+1) + ', ' +
str(row+1) + '\n')
            step += 1

    resultPath.insert(tk.END, '\n')
    end_time = int(round((time.time() - start_time) * 1000))
    resultPath.insert(tk.END, str(end_time) + ' ms\n')


def random_input():
    global buffer_size, best_score
    if updateMtxSize():
        genSequences()
        if len(sequences) > 0:
            insertMtx()

            resultSeq.grid(row=mtx_height+ row_offset+4, column=3,
columnspan=7)

            best_score = float('-inf')
            best_binMtx.clear()
            best_sequence.clear()
            if read_token():
                if read_Matrix():
                    buffer_size = int(box_buffer.get())
                    startSolver()


def file_input():
    clearWidget()

    box_NToken.delete(0 ,'end')
    for entry in entries_token.values():
        entry.destroy()
    entries_token.clear()
    box_buffer.delete(0 ,'end')
    box_cols.set('')
    box_rows.set('')
    box_NSeq.delete(0 ,'end')
    box_MaxSeq.delete(0 ,'end')

    global buffer_size, mtx_width, mtx_height, mtx_input, sequences

    finput_path = filedialog.askopenfilename(title='Open a file',
filetypes=(("Text Files", "*.txt"),))

    with open(finput_path, 'r') as finput:
        buffer_size = int(finput.readline())
        mtx_width, mtx_height = map(int, finput.readline().split())

        if mtx_height > 0 and mtx_height > 0:
            try:
                for _ in range(mtx_height):
                    row = finput.readline().split()
                    temp = []
                    for j in range(mtx_width):
                        token = row[j]
                        if token.isalnum() and len(token)==2:
                            temp.append(row[j])
                        else:
                            messagebox.showerror('Invalid token', 'Invalid
token')
                            root.destroy()
                    mtx_input.append(temp)
            except:
                messagebox.showerror('Invalid Matrix', 'Invalid Matrix')
                root.destroy()

            try:
                N_seq = int(finput.readline())

                for _ in range(N_seq):
                    key = finput.readline().split()
                    for token in key:
                        if not (token.isalnum() and len(token)==2):
                            messagebox.showerror('Invalid sequences', 'Invalid
sequences')
                            root.destroy()

                    val = int(finput.readline())
                    sequences[str(key)] = val
            except:
                messagebox.showerror('Invalid sequences', 'Invalid sequences')
                root.destroy()

    save_button.grid(row=row_offset+3, column=4)
    resultPath.grid(row=row_offset+4, column=4, rowspan=10)
    startSolver()


def saveFile():
    file_path = filedialog.asksaveasfilename(defaultextension=".txt",
filetypes=[("Text files", "*.txt"),])
    if file_path:
        try:
            with open(file_path, 'w') as file:
                if best_score == float('-inf'):
                    file.write('0\n')
                    file.write('\n' + str(end_time) + ' ms\n')
                else:
                    file.write(str(best_score) + '\n')
                    file.write(str(' '.join(best_sequence)) + '\n')
                    writeBinMtx(best_binMtx, file)
                    file.write('\n' + str(end_time) + ' ms\n')
            messagebox.showinfo("Success", "File saved successfully.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {str(e)}")


root = tk.Tk()
root.title("Cyberpunk 2077 Breach Protocol | SOLVER")

curr_mtx_width = tk.StringVar()
curr_mtx_height = tk.StringVar()

# Label and entry widgets for number of rows
label_NToken = tk.Label(root, text="Number of token:")
label_NToken.grid(row=0, column=0)
box_NToken = tk.Entry(root, width=3)
box_NToken.grid(row=0, column=1)

box_NToken.bind('<KeyRelease>', updateTokens)

# Label and entry widgets for number of rows
label_buffer = tk.Label(root, text="Buffer Size:")
label_buffer.grid(row=2, column=0)
box_buffer = tk.Entry(root, width=3)
box_buffer.grid(row=2, column=1)

box_buffer.bind('<KeyRelease>', updateBuffer)

# Label and entry widgets for number of rows
label_rows = tk.Label(root, text="Matrix Height:")
label_rows.grid(row=3, column=0)

box_rows = ttk.Combobox(root, width=2, textvariable=curr_mtx_height)
box_rows.grid(row=3, column=1)
box_rows['values'] = [_ for _ in range(2, 11)]
box_rows['state'] = 'readonly'


# Label and entry widgets for number of columns
label_cols = tk.Label(root, text="Matrix Width:")
label_cols.grid(row=3, column=2)

box_cols = ttk.Combobox(root, width=2, textvariable=curr_mtx_width)
box_cols.grid(row=3, column=3)
box_cols['values'] = [_ for _ in range(2, 11)]
```

```
box_cols['state'] = 'readonly'


# Label and entry widgets for Sequences
label_NSeq = tk.Label(root, text="Number of sequence:")
label_NSeq.grid(row=4, column=0)
box_NSeq = tk.Entry(root, width=3)
box_NSeq.grid(row=4, column=1)

# Label and entry widgets for Length Sequences
label_MaxSeq = tk.Label(root, text="Max size:")
label_MaxSeq.grid(row=4, column=2)

box_MaxSeq = tk.Entry(root, width=3)
box_MaxSeq.grid(row=4, column=3)

browse_button = tk.Button(root, text="browse file...", command=file_input)
browse_button.grid(row=5, column=1)

generate_button = tk.Button(root, text="generate", command=random_input)
generate_button.grid(row=5, column=2)


resultSeq = tk.Text(root, height = 10, width = 40)
resultPath = tk.Text(root, height = 25, width = 40)
save_button = tk.Button(root, text='Save Output', command=saveFile)

root.mainloop()
```

BAB 3

Testing

3.1. Masukan Manual dan Acak



Cyberpunk 2077 Breach Protocol | SOLVER

Number of token: 5

Buffer Size: 7

Matrix Height: 4 Matrix Width: 4

Number of sequence: 6 Max size: 5

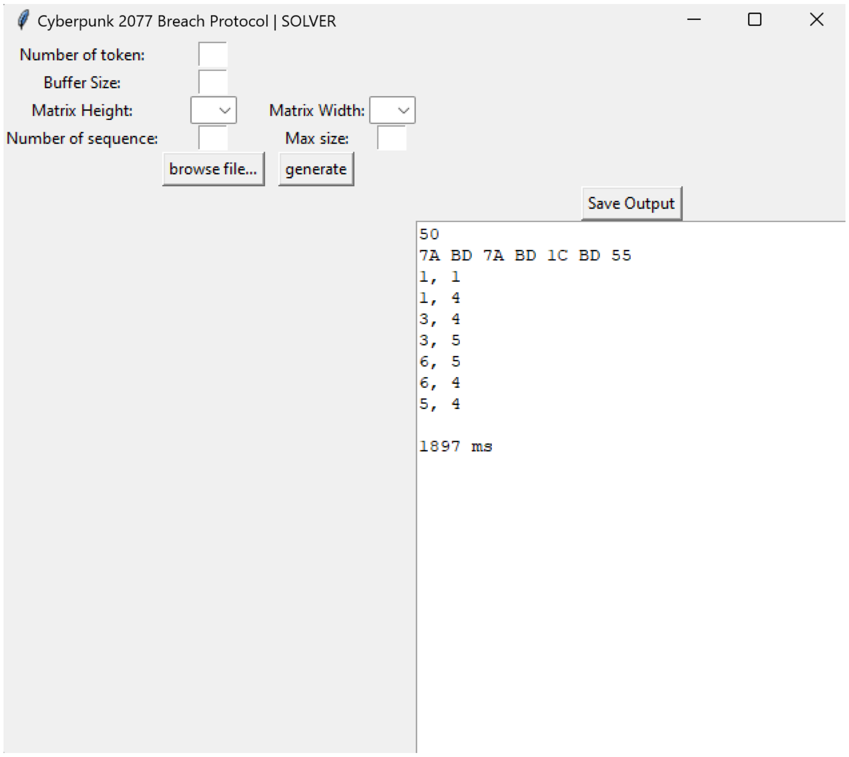
1a	2A	A2	BC	33

A2	33	1a	BC
A2	2A	BC	2A
33	2A	BC	A2
BC	BC	A2	1a

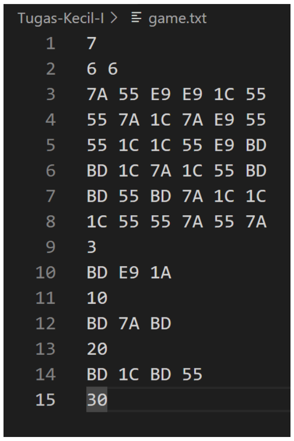
```
1a BC 2A BC : -100
BC BC A2 A2 1a : 130
2A 33 2A 33 1a : -20
A2 BC BC 1a : 30
33 A2 BC 1a : -100
33 2A 1a 33 BC : 30
```

```
130
33 2A BC BC A2 A2 1a
2, 1
2, 3
3, 3
3, 2
1, 2
1, 1
3, 1

105 ms
```

3.2. Masukan File

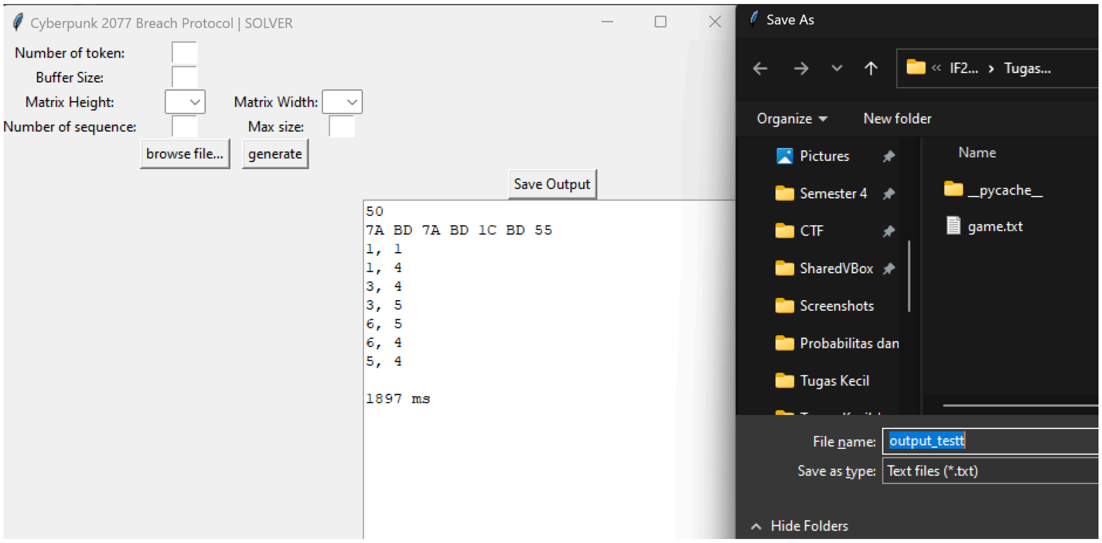


```
50
7A BD 7A BD 1C BD 55
1, 1
1, 4
3, 4
3, 5
6, 5
6, 4
5, 4

1897 ms
```



```
7
6 6
7A 55 E9 E9 1C 55
55 7A 1C 7A E9 55
55 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 55 BD 7A 1C 1C
1C 55 55 7A 55 7A
3
BD E9 1A
10
BD 7A BD
20
BD 1C BD 55
30
```

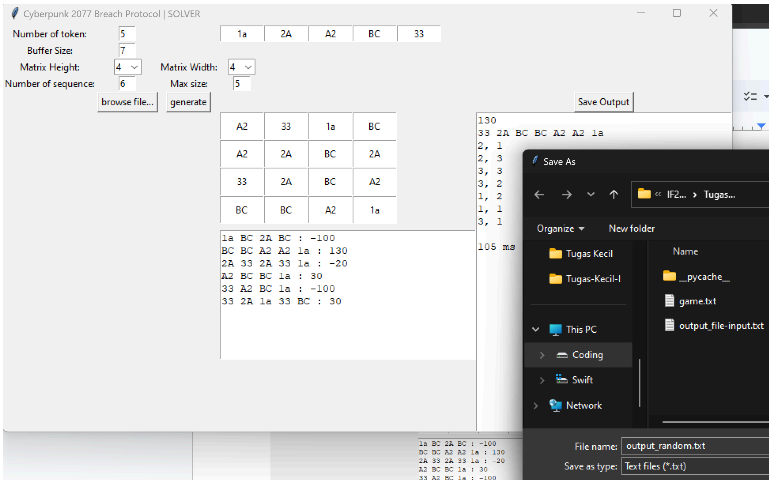
3.3. Simpan File



```
50
7A BD 7A BD 1C BD 55
1, 1
1, 4
3, 4
3, 5
6, 5
6, 4
5, 4

1897 ms
```

File name: output_testt

Save as type: Text files (*.txt)


Cyberpunk 2077 Breach Protocol | SOLVER

Number of token: 5
Buffer Size: 7
Matrix Height: 4
Matrix Width: 4
Number of sequence: 6
Max size: 5

1a	2A	A2	BC	33

A2	33	1a	BC
A2	2A	BC	2A
33	2A	BC	A2
BC	BC	A2	1a

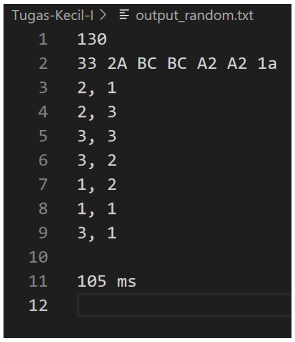
```
1a BC 2A BC : -100
BC BC A2 A2 1a : 130
2A 33 2A 33 1a : -20
A2 BC BC 1a : 30
33 A2 BC 1a : -100
33 2A 1a 33 BC : 30
```

```
130
33 2A BC BC A2 A2 1a
2, 1
2, 3
3, 3
3, 2
1, 2
1, 1
3, 1

105 ms
```

Save As

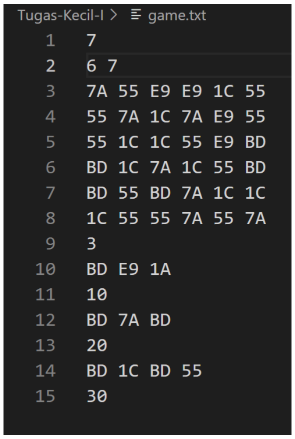
File name: output_random.txt
Save as type: Text files (*.txt)

1a BC 2A BC : -100
BC BC A2 A2 1a : 130
2A 33 2A 33 1a : -20
A2 BC BC 1a : 30
33 A2 BC 1a : -100

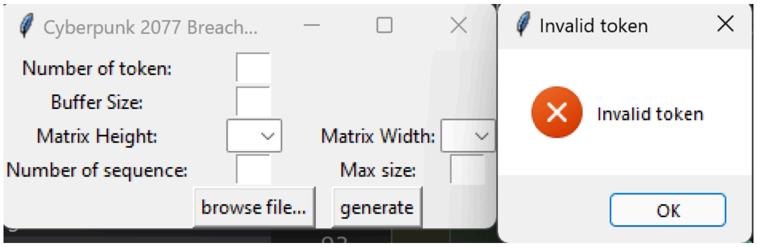

Tugas-Kecil-I > output_random.txt

```
130
33 2A BC BC A2 A2 1a
2, 1
2, 3
3, 3
3, 2
1, 2
1, 1
3, 1

105 ms

```

3.4. Ukuran Matriks Tidak Sesuai


Tugas-Kecil-I > game.txt

```
7
6 7
7A 55 E9 E9 1C 55
55 7A 1C 7A E9 55
55 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 55 BD 7A 1C 1C
1C 55 55 7A 55 7A
3
BD E9 1A
10
BD 7A BD
20
BD 1C BD 55
30
```

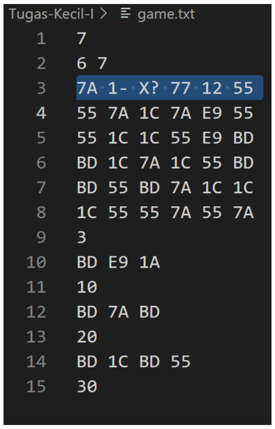

Cyberpunk 2077 Breach...

Number of token:
Buffer Size:
Matrix Height:
Matrix Width:
Number of sequence:
Max size:

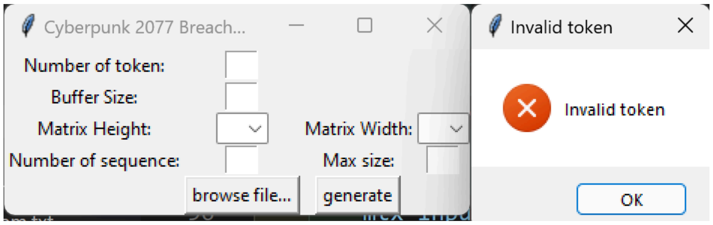
Invalid token

Invalid token

3.5. Token Tidak Alfanumerik


Tugas-Kecil-I > game.txt

```
7
6 7
7A 1- X? 77 12 55
55 7A 1C 7A E9 55
55 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 55 BD 7A 1C 1C
1C 55 55 7A 55 7A
3
BD E9 1A
10
BD 7A BD
20
BD 1C BD 55
30
```


Cyberpunk 2077 Breach...

Number of token:
Buffer Size:
Matrix Height:
Matrix Width:
Number of sequence:
Max size:

Invalid token

Invalid token

LAMPIRAN

Link repository: https://github.com/mdavaf17/Tucil-1-Stima

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. Program dapat menyimpan solusi dalam berkas .txt	✓	
7. Program memiliki GUI	✓	