LAPORAN TUGAS KECIL 1
IF2211 – STRATEGI ALGORITMA
CYBERPUNK 2077 BREACH PROTOCOL SOLVER

Oleh:

Salsabiila

13522062

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2023

A. Penjelasan Singkat

Gambar 1. Permainan *Mini Game Breach Protocol* pada Cyberpunk 2077
(Sumber: https://cyberpunk.fandom.com/wiki/Quickhacking)

Cyberpunk 2077 Breach Protocol merupakan sebuah *mini game* pada permainan Cyberpunk 2077 yang merupakan simulasi peretasan jaringan lokal dari ICE (Intrusion Countermeasures Electronics). Dalam permainan ini, pemain memiliki *goal* untuk mendapatkan sekuens yang memiliki bobot poin tertinggi. Adapun beberapa komponen pada permainan ini, antara lain:

1. Token – terdiri dari 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.

Dan peraturan permainan sebagai berikut:

1. Pemain bergerak dengan pola horizontal, vertikal, horizontal, vertikal (bergantian) hingga semua sekuens berhasil dicocokkan atau buffer penuh.
2. Pemain memulai dengan memilih satu token pada posisi baris paling atas dari matriks.
3. Sekuens dicocokkan pada token-token yang berada di buffer.
4. Satu token pada buffer dapat digunakan pada lebih dari satu sekuens.
5. Setiap sekuens memiliki bobot hadiah atau reward yang variatif.
6. Sekuens memiliki panjang minimal berupa dua token.

B. Algoritma *Brute Force*

Algoritma *Brute Force* merupakan sebuah algoritma dengan cara yang dianggap paling jelas (*obvious way*) dan biasanya memiliki pendekatan yang sederhana dan *straightforward*. Pada program Cyberpunk 2077 Breach Protocol Solver ini, alur berpikir pertamanya adalah dengan menelusuri setiap elemen pada matriks. Jika elemen yang sedang ditelusuri merupakan token pertama dari satu atau lebih sekuens, maka program akan mencari koordinat-koordinat token lain hingga terbentuk sekuens yang diinginkan. Setiap sekuens akan dicoba satu per satu hingga tidak terdapat sekuens dengan token awal yang sama dengan elemen matriks tersebut.

Karena pemain hanya dapat memulai sekuens dari baris pertama matriks dan arah pola selanjutnya harus vertikal, maka pengecekan tiap elemen matriks dibagi menjadi dua. Pada pengecekan pertama, arah polanya adalah horizontal dan pada pengecekan yang kedua arah polanya adalah vertikal. Di pengecekan pertama, apabila terdapat token yang sesuai dengan token yang muncul setelah elemen pada sekuens maka pencarian akan dilanjutkan hingga isi buffer bersisa satu atau tidak ada lagi elemen pada matriks yang memenuhi persyaratan. Isi buffer sengaja tidak dibiarkan penuh, hal ini dikarenakan pergerakan pola yang berarah horizontal berarti awal pola sekuens harus ditambahkan token pada baris pertama kolom tersebut. Karena hal ini jugalah pengecekan pertama ini hanya dilakukan dari baris kedua.

Pada pengecekan kedua, pergerakan pola pertama yang akan dilakukan adalah secara vertikal sehingga pengecekan dapat dilakukan hanya pada baris pertama matriks, meskipun dalam penulisan program masih dilakukan pengecekan hingga baris terakhir yang sebenarnya tidak diperlukan. Pengecekan ini akan dilakukan hingga isi buffer penuh atau saat tidak ada lagi elemen pada matriks yang memenuhi persyaratan.

Untuk meng-*handle* sekuens akhir yang dapat merupakan gabungan dari beberapa sekuens, digunakan konsep looping untuk mendapatkan kombinasi-kombinasi yang mungkin. Saat suatu kombinasi sudah tercapai, jika semua sekuens termasuk dalam kombinasi tersebut, maka looping akan berhenti dan langsung memberikan sekuens tersebut. Namun jika hal tersebut belum tercapai, maka akan dilakukan pengecekan apakah menambah sekuens baru akan melebihi kapasitas buffer, dan jika jawabannya ya maka pengecekan elemen matriks akan berlanjut untuk mencari sekuens dengan bobot poin paling besar.

C. Kode Program

main.py

```
import time
from txt_input_output import *
from movement import *
from util import *

print("Cyberpunk 2077 Breach Protocol Solver")
print("====================================")
print()

print("Jenis Input")
print("1. File .txt")
print("2. CLI")
user_input = input("Masukkan jenis input: ")

while user_input not in ["1", "2"]:
    print("Input tidak valid")
    user_input = input("Masukkan jenis input: ")

print()

if user_input == "1":
    buffer_size, matrix_size, matrix, num_sequences, sequences, points = read_file()

elif user_input == "2":
    num_unique_tokens = int(input("Masukkan jumlah token unik: "))
    unique_tokens = input("Masukkan token unik: ").split()
    buffer_size = int(input("Masukkan panjang buffer: "))
    matrix_size = tuple(map(int, input("Masukkan ukuran matriks: ").split()))
    num_sequences = int(input("Masukkan jumlah sequence: "))
    max_sequence_length = int(input("Masukkan panjang maksimal sequence: "))

    matrix = generate_matrix(unique_tokens, matrix_size[0], matrix_size[1])
    sequences = generate_sequences(unique_tokens, num_sequences, max_sequence_length)
    points = point_generator(num_sequences)
```

```
    print("MATRIKS:")
    for row in range(matrix_size[0]):
        for col in range(matrix_size[1]):
            print(matrix[row][col], end=" ")
        print()

    for i in range(num_sequences):
        print(f"Sekuen {i+1}: {sequences[i]}")
        print(f"Poin: {points[i]}")

start = time.time()

hor_res, hor_p = horizontal_first(sequences, num_sequences, matrix_size, matrix,
points, buffer_size)
ver_res, ver_p = vertical_first(sequences, num_sequences, matrix_size, matrix,
points, buffer_size)

end = time.time()

print()

if ver_res != [] and hor_res != []:
    if ver_p > hor_p and ver_res[0][0] == 0:
        result = ver_res
        point = ver_p
    else:
        result = [(0, hor_res[1], -1, matrix[0][hor_res[1]])]
        result.extend(hor_res)
        point = hor_p

elif ver_res != [] and ver_res[0][0] == 0:
    result = ver_res
    point = ver_p

elif hor_res != []:
    result = [(0, hor_res[0][1], -1, matrix[0][hor_res[0][1]])]
    result.extend(hor_res)
```

```
    point = hor_p

else:
    point = 0
    print(f"Point: {point}")
    print("Tidak ada sequence yang ditemukan")

if point != 0:
    print_result(result, point)

print()
print("Waktu eksekusi:", (end - start)*1000, "ms")
print()
user_input2 = input("Apakah Anda ingin menyimpan solusi ke dalam file? (y/n):
").lower()

if user_input2 == "y":
    if user_input == "1":
        write_file1(result, point)
    elif user_input == "2":
        write_file2(result, point, matrix)
else:
    print("Terima kasih telah menggunakan program ini! :D")
```

movement.py

```
from util import *

def horizontal_first(sequences, num_sequences, matrix_size, matrix, points,
buffer_size):
    point = 0
    result = []

    for x in range(num_sequences):
        sequence = sequences[x]

        for i in range(1, matrix_size[0]):
            for j in range(matrix_size[1]):
```

```
                if matrix[i][j] == sequence[0]:
                    temp_result = []
                    current_seq_checked = x
                    temp_result.append((i, j, 1, matrix[i][j]))
                    num = 1
                    move = 1
                    temp_matrix = duplicate_matrix(matrix)
                    start_i = i
                    start_j = j
                    length = len(sequence)
                    found_sequence = []
                    indicator = 0

                    while temp_matrix[i][j] != None:
                        isFound = True
                        while len(temp_result) < length and isFound and num <=
len(sequences[current_seq_checked]) - 1:
                            move = temp_result[-1][2]

                            if move == 1:
                                isFound = False
                                hor = 0
                                while hor < matrix_size[1] and not isFound:
                                    if temp_matrix[start_i][hor] == sequence[num] and
hor != start_j:
                                        isFound = True
                                        if (start_i, hor, 0,
temp_matrix[start_i][hor]) not in temp_result:
                                            temp_result.append((start_i, hor, 0,
temp_matrix[start_i][hor]))
                                        num += 1
                                        start_j = hor
                                    hor += 1

                            elif move == 0:
                                isFound = False
                                ver = 0
```

```
                                while ver < matrix_size[0] and not isFound:
                                    if temp_matrix[ver][start_j] == sequence[num] and
ver != start_i:
                                        isFound = True
                                        if (ver, start_j, 1,
temp_matrix[ver][start_j]) not in temp_result:
                                            temp_result.append((ver, start_j, 1,
temp_matrix[ver][start_j]))
                                        num += 1
                                        start_i = ver
                                    ver += 1

                            if len(temp_result) == length:
                                if temp_result == result:
                                    temp_matrix[temp_result[-1][0]][temp_result[-1][1]] =
None
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                                else:
                                    if current_seq_checked not in found_sequence:
                                        found_sequence.append(current_seq_checked)
                                    temp_point = check_sequence(temp_result, sequences,
num_sequences, points)
                                    if temp_point > point:
                                        point = temp_point
                                        result = temp_result.copy()

                                    if len(found_sequence) == num_sequences:
                                        return result, point
                                    else:
                                        base = temp_result.copy()
                                        for y in range(num_sequences):
                                            if y not in found_sequence and temp_result[-
1][3] == sequences[y][0]:
                                                if temp_result[-1][2] == 1:
```

```
                                            if check_horizontal(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][0]) and length+len(sequences[y]) <=
buffer_size:
                                                current_seq_checked = y
                                                length += len(sequences[y])-1
                                                sequence = sequences[y]
                                                num = 1
                                                indicator = 1
                                            elif
check_horizontal(sequences[y][1], temp_matrix, matrix_size, temp_result[-1][0]) and
length+len(sequences[y]) > buffer_size:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j

                                        elif temp_result[-1][2] == 0:
                                            if check_vertical(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) <=
buffer_size:
                                                current_seq_checked = y
                                                length += len(sequences[y])-1
                                                sequence = sequences[y]
                                                num = 1
                                                indicator = 1
                                            elif check_vertical(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) >
buffer_size:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
```

```
                                                start_j = j



                                        elif y not in found_sequence and
temp_result[-1][3] != sequences[y][0]:
                                            if temp_result[-1][2] == 1:
                                                if check_horizontal(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][0]) and length+len(sequences[y]) <=
buffer_size-1:
                                                    current_seq_checked = y
                                                    length += len(sequences[y])
                                                    sequence = sequences[y]
                                                    num = 0
                                                    indicator = -1
                                                elif
check_horizontal(sequences[y][0], temp_matrix, matrix_size, temp_result[-1][0]) and
length+len(sequences[y]) > buffer_size-1:
                                                    temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                    temp_result = [(i, j, 1,
matrix[i][j])]
                                                    num = 1
                                                    start_i = i
                                                    start_j = j
                                                else:
                                                    temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                    temp_result = [(i, j, 1,
matrix[i][j])]
                                                    num = 1
                                                    start_i = i
                                                    start_j = j

                                            elif temp_result[-1][2] == 0:
                                                if check_vertical(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) <=
buffer_size-1:
```

```
                                                current_seq_checked = y
                                                length += len(sequences[y])
                                                sequence = sequences[y]
                                                num = 0
                                                indicator = -1
                                            elif check_vertical(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) >
buffer_size-1:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j
                                            else:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j



                        else:
                            temp_matrix[temp_result[-1][0]][temp_result[-1][1]] =
None

                            if indicator == 1:
                                if
check_horizontal(sequences[current_seq_checked][1], temp_matrix, matrix_size,
temp_result[-1][0]):
                                    temp_result = base
                                    start_i = temp_result[-1][0]
                                    start_j = temp_result[-1][1]
```

```
                                    num = 1
                                else:
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                            elif indicator == -1:
                                if check_vertical(sequences[current_seq_checked][1],
temp_matrix, matrix_size, temp_result[-1][1]):
                                    temp_result = base
                                    start_i = temp_result[-1][0]
                                    start_j = temp_result[-1][1]
                                    num = 1
                                else:
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                            else:
                                temp_result = [(i, j, 1, matrix[i][j])]
                                num = 1
                                start_i = i
                                start_j = j

    return result, point

def vertical_first(sequences, num_sequences, matrix_size, matrix, points,
buffer_size):
    point = 0
    result = []

    for x in range(num_sequences):
        sequence = sequences[x]

    for x in range(num_sequences):
        sequence = sequences[x]
        result = []
```

```
        for i in range(matrix_size[0]):
            for j in range(matrix_size[1]):
                if matrix[i][j] == sequence[0]:
                    temp_result = []
                    current_seq_checked = x
                    temp_result.append((i, j, 1, matrix[i][j]))
                    num = 1
                    move = 1
                    temp_matrix = duplicate_matrix(matrix)
                    start_i = i
                    start_j = j
                    length = len(sequence)
                    found_sequence = []
                    indicator = 0

                    while temp_matrix[i][j] != None:
                        isFound = True
                        while len(temp_result) < length and isFound and num <=
len(sequences[current_seq_checked]) - 1:
                            move = temp_result[-1][2]

                            if move == 1:
                                isFound = False
                                hor = 0
                                while hor < matrix_size[1] and not isFound:
                                    if temp_matrix[start_i][hor] == sequence[num] and
hor != start_j:
                                        isFound = True
                                        if (start_i, hor, 0,
temp_matrix[start_i][hor]) not in temp_result:
                                            temp_result.append((start_i, hor, 0,
temp_matrix[start_i][hor]))
                                        num += 1
                                        start_j = hor
                                    hor += 1
```

```
                            elif move == 0:
                                isFound = False
                                ver = 0
                                while ver < matrix_size[0] and not isFound:
                                    if temp_matrix[ver][start_j] == sequence[num] and
ver != start_i:
                                        isFound = True
                                        if (ver, start_j, 1,
temp_matrix[ver][start_j]) not in temp_result:
                                            temp_result.append((ver, start_j, 1,
temp_matrix[ver][start_j]))
                                        num += 1
                                        start_i = ver
                                    ver += 1

                            if len(temp_result) == length:
                                if temp_result == result:
                                    temp_matrix[temp_result[-1][0]][temp_result[-1][1]] =
None
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                                else:
                                    if current_seq_checked not in found_sequence:
                                        found_sequence.append(current_seq_checked)
                                    temp_point = check_sequence(temp_result, sequences,
num_sequences, points)
                                    if temp_point > point:
                                        point = temp_point
                                        result = temp_result.copy()

                                if len(found_sequence) == num_sequences:
                                    return result, point
                                else:
                                    base = temp_result.copy()
                                    for y in range(num_sequences):
```

```
                                    if y not in found_sequence and temp_result[-
1][3] == sequences[y][0]:
                                        if temp_result[-1][2] == 1:
                                            if check_horizontal(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][0]) and length+len(sequences[y]) <=
buffer_size:
                                                current_seq_checked = y
                                                length += len(sequences[y])-1
                                                sequence = sequences[y]
                                                num = 1
                                                indicator = 1
                                            elif
check_horizontal(sequences[y][1], temp_matrix, matrix_size, temp_result[-1][0]) and
length+len(sequences[y]) > buffer_size:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j

                                        elif temp_result[-1][2] == 0:
                                            if check_vertical(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) <=
buffer_size:
                                                current_seq_checked = y
                                                length += len(sequences[y])-1
                                                sequence = sequences[y]
                                                num = 1
                                                indicator = 1
                                            elif check_vertical(sequences[y][1],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) >
buffer_size:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
```

```
                                                    temp_result = [(i, j, 1,
matrix[i][j])]
                                                    num = 1
                                                    start_i = i
                                                    start_j = j


                                        elif y not in found_sequence and
temp_result[-1][3] != sequences[y][0]:
                                            if temp_result[-1][2] == 1:
                                                if check_horizontal(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][0]) and length+len(sequences[y]) <=
buffer_size-1:
                                                    current_seq_checked = y
                                                    length += len(sequences[y])
                                                    sequence = sequences[y]
                                                    num = 0
                                                    indicator = -1
                                                elif
check_horizontal(sequences[y][0], temp_matrix, matrix_size, temp_result[-1][0]) and
length+len(sequences[y]) > buffer_size-1:
                                                    temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                    temp_result = [(i, j, 1,
matrix[i][j])]
                                                    num = 1
                                                    start_i = i
                                                    start_j = j
                                                else:
                                                    temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                    temp_result = [(i, j, 1,
matrix[i][j])]
                                                    num = 1
                                                    start_i = i
                                                    start_j = j
```

```
                                        elif temp_result[-1][2] == 0:
                                            if check_vertical(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) <=
buffer_size-1:
                                                current_seq_checked = y
                                                length += len(sequences[y])
                                                sequence = sequences[y]
                                                num = 0
                                                indicator = -1
                                            elif check_vertical(sequences[y][0],
temp_matrix, matrix_size, temp_result[-1][1]) and length+len(sequences[y]) >
buffer_size-1:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j
                                            else:
                                                temp_matrix[temp_result[-
1][0]][temp_result[-1][1]] = None
                                                temp_result = [(i, j, 1,
matrix[i][j])]
                                                num = 1
                                                start_i = i
                                                start_j = j




                        else:
                            temp_matrix[temp_result[-1][0]][temp_result[-1][1]] =
None


                            if indicator == 1:
```

```
                                if
check_horizontal(sequences[current_seq_checked][1], temp_matrix, matrix_size,
temp_result[-1][0]):
                                    temp_result = base
                                    start_i = temp_result[-1][0]
                                    start_j = temp_result[-1][1]
                                    num = 1
                                else:
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                            elif indicator == -1:
                                if check_vertical(sequences[current_seq_checked][1],
temp_matrix, matrix_size, temp_result[-1][1]):
                                    temp_result = base
                                    start_i = temp_result[-1][0]
                                    start_j = temp_result[-1][1]
                                    num = 1
                                else:
                                    temp_result = [(i, j, 1, matrix[i][j])]
                                    num = 1
                                    start_i = i
                                    start_j = j
                            else:
                                temp_result = [(i, j, 1, matrix[i][j])]
                                num = 1
                                start_i = i
                                start_j = j


    return result, point
```

txt_input_output.py

```
import os
from util import *


def read_file():
```

```
current_directory = os.path.dirname(os.path.abspath(__file__))
input_file = input("Masukkan nama file: ")
input_file = os.path.join(current_directory, '..', 'test', f'{input_file}')

while not os.path.exists(input_file):
    print("File tidak ditemukan")
    input_file = input("Masukkan nama file: ")
    input_file = os.path.join(current_directory, '..', 'test', f'{input_file}')

buffer_size = 0
matrix_size = (0, 0)
matrix = []
num_sequences = 0
sequences = []
points = []

with open(input_file, "r") as file:
    lines = file.readlines()
    for line_index, line in enumerate(lines):
        line = line.strip()
        if not line:
            continue
        tokens = line.split()

        if line_index == 0:
            buffer_size = int(tokens[0])

        elif line_index == 1:
            matrix_size = (int(tokens[0]), int(tokens[1]))

        elif 1 < line_index <= matrix_size[0] + 1:
            matrix.append(tokens)

        elif line_index == matrix_size[0] + 2:
            num_sequences = int(tokens[0])
```

```
            elif matrix_size[0] + 3 <= line_index < matrix_size[0] + 3 +
num_sequences * 2:
                if (line_index - matrix_size[0] - 3) % 2 == 0:
                    sequences.append(tokens)
                else:
                    points.append(tokens[0])

    return buffer_size, matrix_size, matrix, num_sequences, sequences, points

def write_file1(result, point):
    folder_path = os.path.join(os.path.abspath('.'), 'test')
    file_name = input("Masukkan nama file (.txt): ")
    file_path = os.path.join(folder_path, f'{file_name}')

    while os.path.exists(file_path):
        overwrite = input("File dengan nama tersebut sudah ada. Apakah Anda ingin
melakukan overwrite? (y/n): ").lower()
        if overwrite != 'y':
            file_name = input("Masukkan nama file (.txt): ")
            file_path = os.path.join(folder_path, f'{file_name}')
        else:
            break

    sequence = sequence_string(result)
    coordinate = coordinate_string(result)

    with open(file_path, 'w') as file:
        file.write(f"Point: {point}\n")

        file.write(f"Sequence: {sequence}\n")

        file.write(f"{coordinate}")

    print(f"File berhasil disimpan di {file_path}")

def write_file2(result, point, matrix):
    folder_path = os.path.join(os.path.abspath('.'), 'test')
```

```
    file_name = input("Masukkan nama file (.txt): ")
    file_path = os.path.join(folder_path, f'{file_name}')

    if os.path.exists(file_path):
        overwrite = input("File dengan nama tersebut sudah ada. Apakah Anda ingin
melakukan overwrite? (y/n): ").lower()
        if overwrite != 'y':
            while os.path.exists(file_path):
                file_name = input("Masukkan nama file (.txt): ")
                file_path = os.path.join(folder_path, f'{file_name}')

    sequence = sequence_string(result)
    coordinate = coordinate_string(result)

    with open(file_path, 'w') as file:
        file.write("Matriks:\n")
        for row in matrix:
            file.write(' '.join(map(str, row)) + '\n')

        file.write(f"Point: {point}\n")

        file.write(f"Sequence: {sequence}\n")

        file.write(f"{coordinate}")

    print(f"File berhasil disimpan di {file_path}")
```

util.py

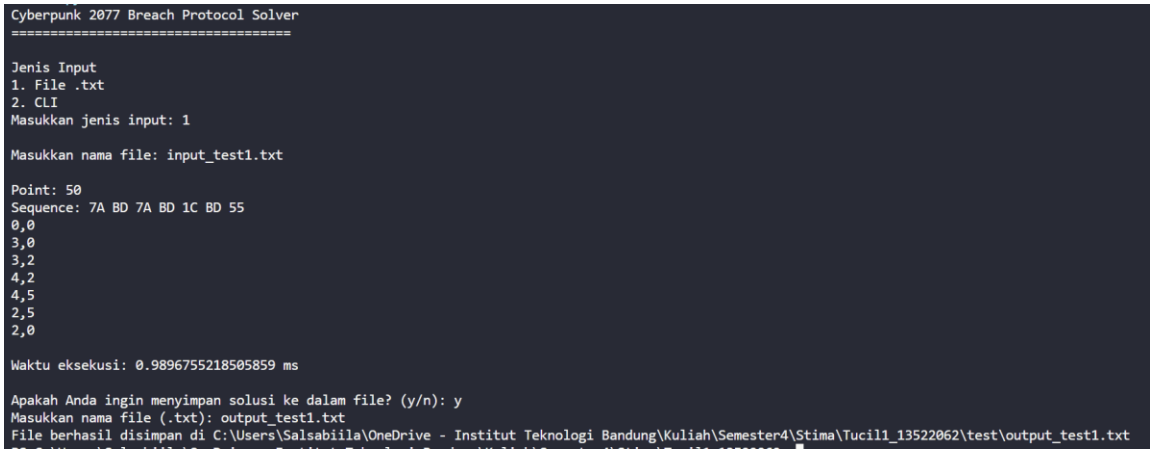
```
import random

def generate_matrix(tokens, n, m):
    matrix = []
    for i in range(n):
        row = [random.choice(tokens) for i in range(m)]
        matrix.append(row)
    return matrix
```

```python
def generate_sequences(tokens, num_sequences, max_sequence_length):
    sequences = []
    for i in range(num_sequences):
        sequence_length = random.randint(2, max_sequence_length)
        sequence = [random.choice(tokens) for i in range(sequence_length)]
        sequences.append(sequence)
    return sequences

def point_generator(num_sequences):
    points = []
    for i in range(num_sequences):
        points.append((random.randint(1, 20))*5)
    return points

def duplicate_matrix(matrix):
    return [row.copy() for row in matrix]

def check_horizontal(token, matrix, matrix_size, start_i):
    for j in range(matrix_size[1]):
        if matrix[start_i][j] == token:
            return True

def check_vertical(token, matrix, matrix_size, start_j):
    for i in range(matrix_size[0]):
        if matrix[i][start_j] == token:
            return True

def check_sequence(temp_result, sequences, num_sequences, points):
    string = ""
    point = 0

    for i in range(len(temp_result)):
        string += temp_result[i][3]

    for i in range(num_sequences):
        checker = ""
        for j in range(len(sequences[i])):
```

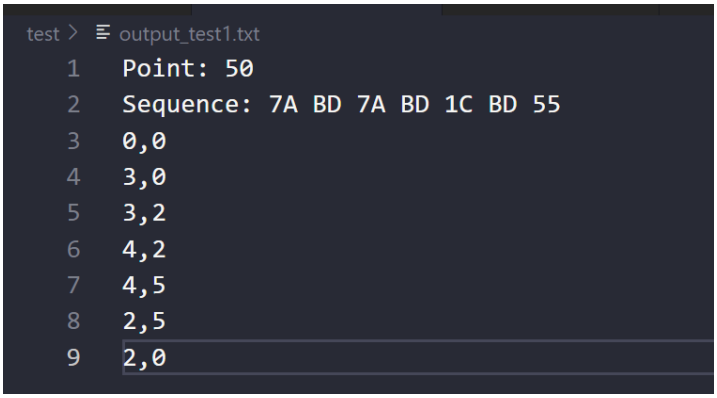
```
            checker += sequences[i][j]
        if checker in string:
            point += int(points[i])

    return point

def print_result(result, point):
    print("Point:", point)
    print("Sequence: ", end="")
    for i in range(len(result)):
        if i != len(result) - 1:
            print(result[i][3], end=" ")
        else:
            print(result[i][3])

    for i in range(len(result)):
        print(f"{result[i][0]},{result[i][1]}")

def sequence_string(result):
    string = ""
    for i in range(len(result)):
        if i != len(result) - 1:
            string += f"{result[i][3]} "
        else:
            string += f"{result[i][3]}"
    return string

def coordinate_string(result):
    string = ""
    for i in range(len(result)):
        if i != len(result) - 1:
            string += f"{result[i][0]},{result[i][1]}\n"
        else:
            string += f"{result[i][0]},{result[i][1]}"
      return string
```

D. Implementasi Program

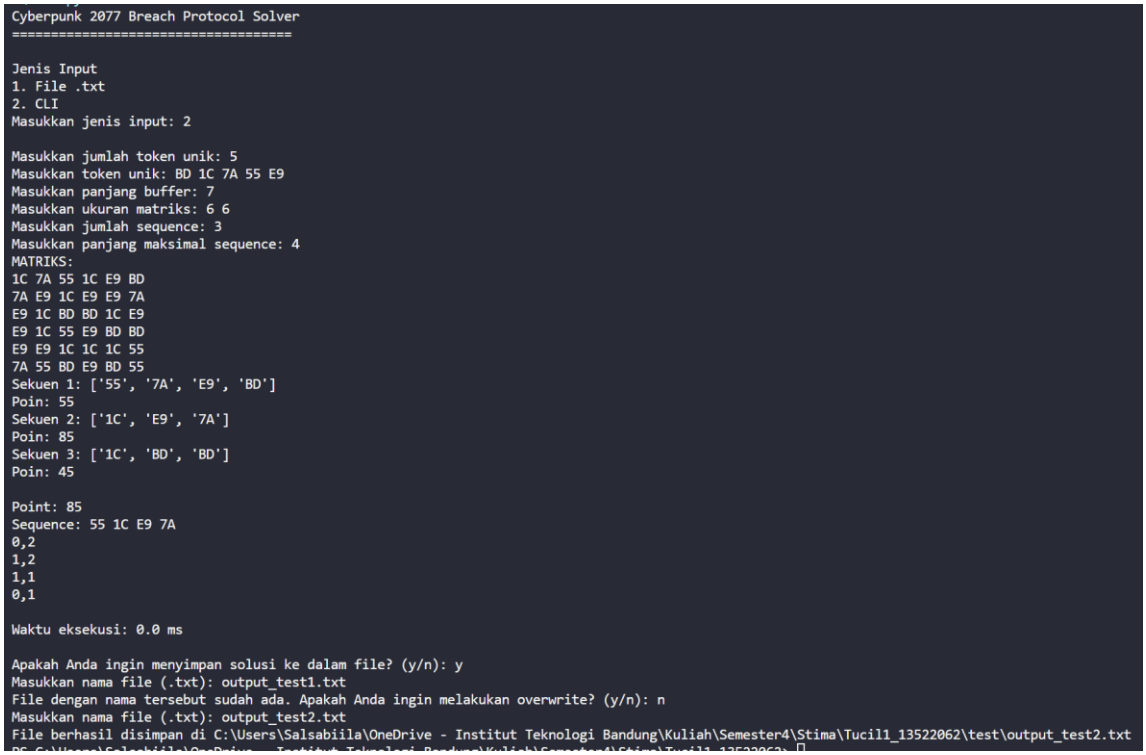
Berikut merupakan tangkapan layar untuk impelementasi program.



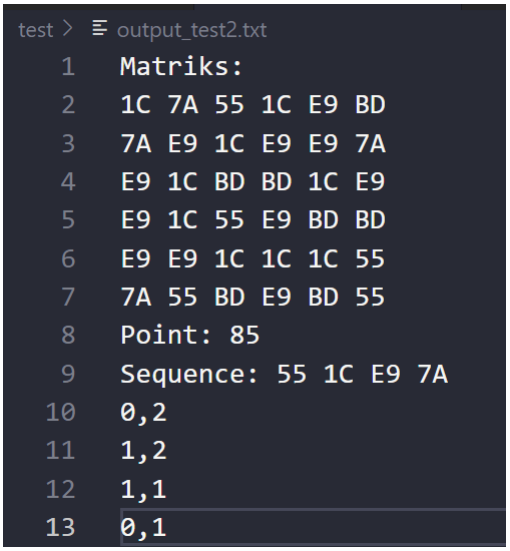
Gambar 2. Contoh Masukkan dan Keluaran 1



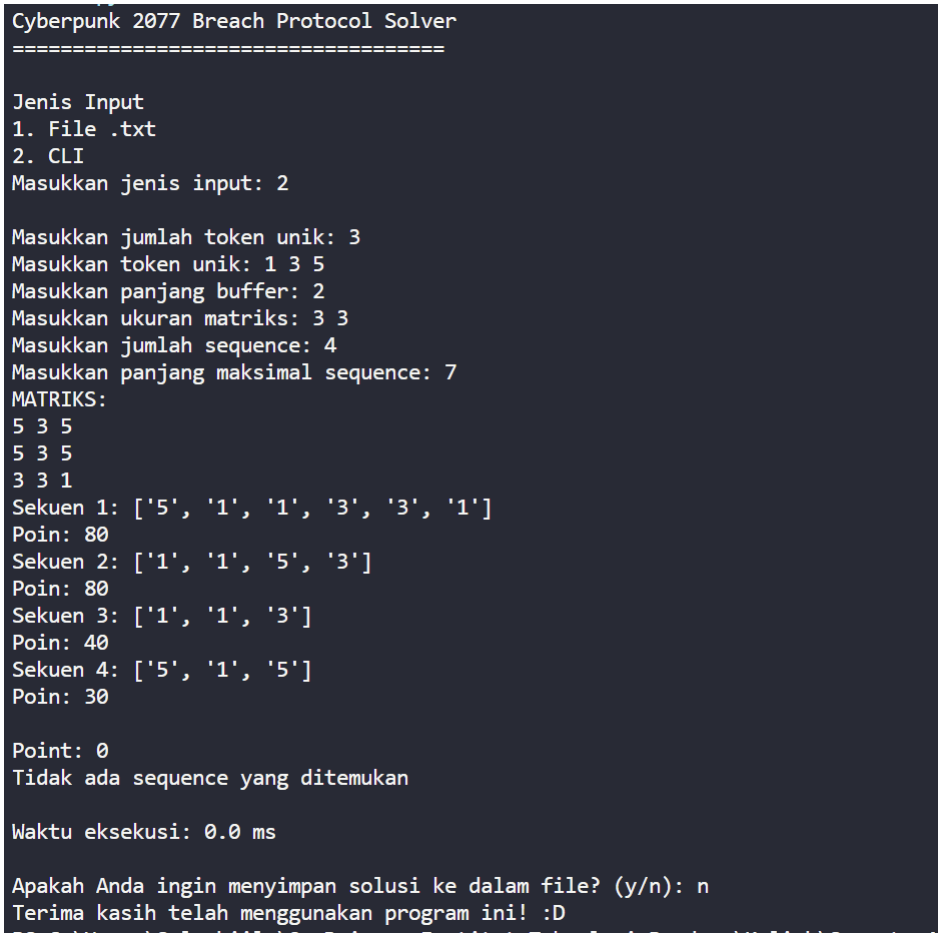
Gambar 3. Hasil *File* .txt Contoh Masukkan dan Keluaran 1



Gambar 4. Contoh Masukkan dan Keluaran 2



Gambar 5. Hasil *File* .txt Contoh Masukkan dan Keluaran 2



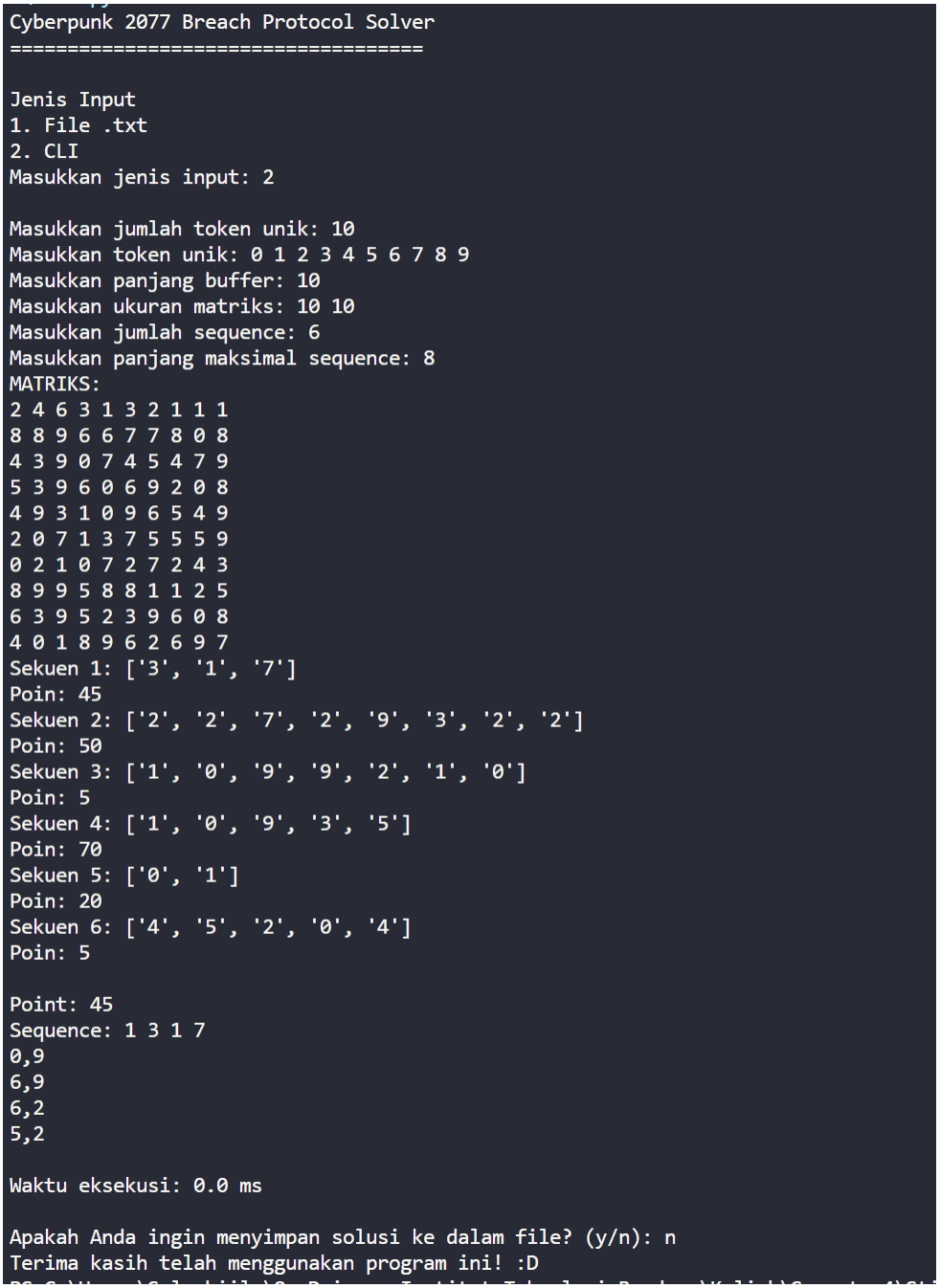
Gambar 6. Contoh Masukkan dan Keluaran 3

```
Cyberpunk 2077 Breach Protocol Solver
==================================

Jenis Input
1. File .txt
2. CLI
Masukkan jenis input: 2

Masukkan jumlah token unik: 2
Masukkan token unik: 1 2
Masukkan panjang buffer: 4
Masukkan ukuran matriks: 4 4
Masukkan jumlah sequence: 3
Masukkan panjang maksimal sequence: 3
MATRIKS:
1 1 1 2
2 2 1 1
2 1 1 1
2 2 2 1
Sekuen 1: ['1', '2']
Poin: 25
Sekuen 2: ['1', '2', '1']
Poin: 65
Sekuen 3: ['2', '2']
Poin: 70

Point: 90
Sequence: 1 1 2 1
0,2
1,2
1,0
0,0

Waktu eksekusi: 0.0 ms

Apakah Anda ingin menyimpan solusi ke dalam file? (y/n): n
Terima kasih telah menggunakan program ini! :D
```

Gambar 7. Contoh Masukkan dan Keluaran 4

```
Cyberpunk 2077 Breach Protocol Solver
==================================

Jenis Input
1. File .txt
2. CLI
Masukkan jenis input: 2

Masukkan jumlah token unik: 1
Masukkan token unik: 0
Masukkan panjang buffer: 4
Masukkan ukuran matriks: 5 5
Masukkan jumlah sequence: 4
Masukkan panjang maksimal sequence: 4
MATRIKS:
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
Sekuen 1: ['0', '0']
Poin: 55
Sekuen 2: ['0', '0']
Poin: 10
Sekuen 3: ['0', '0', '0', '0']
Poin: 90
Sekuen 4: ['0', '0', '0']
Poin: 75

Point: 230
Sequence: 0 0 0 0 0
0,0
1,0
1,1
0,1
0,0

Waktu eksekusi: 0.6878376007080078 ms

Apakah Anda ingin menyimpan solusi ke dalam file? (y/n): n
Terima kasih telah menggunakan program ini! :D
```

Gambar 8. Contoh Masukkan dan Keluaran 5



```
Cyberpunk 2077 Breach Protocol Solver
====================================

Jenis Input
1. File .txt
2. CLI
Masukkan jenis input: 2

Masukkan jumlah token unik: 10
Masukkan token unik: 0 1 2 3 4 5 6 7 8 9
Masukkan panjang buffer: 10
Masukkan ukuran matriks: 10 10
Masukkan jumlah sequence: 6
Masukkan panjang maksimal sequence: 8
MATRIKS:
2 4 6 3 1 3 2 1 1 1
8 8 9 6 6 7 7 8 0 8
4 3 9 0 7 4 5 4 7 9
5 3 9 6 0 6 9 2 0 8
4 9 3 1 0 9 6 5 4 9
2 0 7 1 3 7 5 5 5 9
0 2 1 0 7 2 7 2 4 3
8 9 9 5 8 8 1 1 2 5
6 3 9 5 2 3 9 6 0 8
4 0 1 8 9 6 2 6 9 7
Sekuen 1: ['3', '1', '7']
Poin: 45
Sekuen 2: ['2', '2', '7', '2', '9', '3', '2', '2']
Poin: 50
Sekuen 3: ['1', '0', '9', '9', '2', '1', '0']
Poin: 5
Sekuen 4: ['1', '0', '9', '3', '5']
Poin: 70
Sekuen 5: ['0', '1']
Poin: 20
Sekuen 6: ['4', '5', '2', '0', '4']
Poin: 5

Point: 45
Sequence: 1 3 1 7
0,9
6,9
6,2
5,2

Waktu eksekusi: 0.0 ms

Apakah Anda ingin menyimpan solusi ke dalam file? (y/n): n
Terima kasih telah menggunakan program ini! :D
```

Gambar 9. Contoh Masukkan dan Keluaran 6

E. Lampiran

Pranala repository: https://github.com/salsbiila/Tucil1_13522062

Tabel Implementasi Program

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