LAPORAN TUGAS KECIL I

IF2211 - STRATEGI ALGORITMA

Penyelesaian Cyberpunk 2077 *Breach Protocol* dengan Algoritma *Brute Force*

Disusun oleh :

RAFFAEL BOYMIAN SIAHAAN

13522046

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB 1
DESKRIPSI MASALAH

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 antara lain adalah:

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.

Aturan permainan *Breach Protocol* antara lain:

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.

Tujuan dari diberikan tugas kecil 1 adalah menemukan solusi dari permainan *Breach Protocol* yang paling optimal untuk setiap kombinasi matriks, sekuens, dan ukuran buffer dengan menggunakan algoritma brute force, dengan bahasa pemrograman yang dibebaskan dan aturan tertentu yang berada pada file Spesifikasi Tugas Kecil 1.

BAB 2
TEORI SINGKAT

2.1 Algoritma *Brute Force*

Algoritma *Brute force* adalah pendekatan yang lempeng (*straightforward*) untuk memecahkan suatu persoalan. Algoritma ini didasarkan pada pernyataan pada persoalan (problem statement) dan Definisi/konsep yang dilibatkan. Algoritma ini dapat memecahkan persoalan dengan sangat sederhana, langsung, jelas caranya, dan seringkali disebut sebagai "sapu jagad" karena dapat menyelesaikan segala jenis persoalan tanpa mempedulikan efisiensi dari alur berpikir program yang dibuat.

Contoh dari algoritma Brute force adalah sebagai berikut :

1. Mencari elemen terkecil atau terbesar pada suatu senarai (array)
2. Pencarian beruntun (*Sequential Search*)
3. Menghitung nilai eksponen, faktorial, perkalian matriks, uji bilangan prima, logika pengurutan (sorting), seperti bubble sort dan selection sort, pencocokan string (*String Matching/Pattern Matching*)

Karakteristik dari Algoritma Brute Force adalah :

1. Algoritma brute force umumnya tidak "cerdas" dan tidak mangkus, karena ia membutuhkan volume komputasi yang besar dan waktu yang lama dalam penyelesaiannya. Kata "*force*" mengindikasikan "tenaga" ketimbang "otak" Kadang-kadang algoritma *brute force* disebut juga algoritma naif *(naïve algorithm*).
2. Algoritma brute force lebih cocok untuk persoalan yang ukuran masukannya (n) kecil. Pertimbangannya adalah sederhana dan implementasinya mudah. Algoritma brute force sering digunakan sebagai basis pembanding dengan algoritma lain yang lebih mangkus.
3. Meskipun bukan metode *problem solving* yang mangkus, hampir semua persoalan dapat diselesaikan dengan algoritma *brute force*. Sangat sukar untuk menunjukkan persoalan yang tidak dapat diselesaikan dengan metode brute force. Bahkan, ada persoalan yang hanya dapat diselesaikan dengan brute force. Contoh: mencari elemen terbesar di dalam senarai.

Kekuatan dan kelemahan Algoritma Brute Force adalah sebagai berikut :

Kekuatan:

1. Algoritma brute force dapat diterapkan untuk memecahkan hampir sebagian besar masalah (*wide applicability*).
2. Algoritma brute force sederhana dan mudah dimengerti.
3. Algoritma brute force menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
4. Algoritma brute force menghasilkan algoritma baku (standard) untuk tugas-tugas komputasi seperti penjumlahan/perkalian n buah bilangan, menentukan elemen minimum atau maksimum di dalam senarai (larik).

Kelemahan:

1. Algoritma *brute force* jarang menghasilkan algoritma yang mangkus.
2. Algoritma *brute force* umumnya lambat untuk masukan berukuran besar sehingga tidak dapat diterima.
3. Tidak sekontruktif/sekreatif strategi pemecahan masalah lainnya.

2.2 *Exhaustive Search*

Exhaustive search adalah teknik pencarian solusi secara solusi *brute force* untuk persoalan-persoalan kombinatorik, yaitu persoalan di antara objek-objek kombinatorik seperti permutasi, kombinasi, atau himpunan bagian dari sebuah himpunan.

Langkah-langkah di dalam exhaustive search:

1. Enumerasi (list) setiap kemungkinan solusi dengan cara yang sistematis.
2. Evaluasi setiap kemungkinan solusi satu per satu, simpan solusi terbaik yang ditemukan sampai sejauh ini (*the best solution found so far*).
3. Bila pencarian berakhir, umumkan solusi terbaik (*the winner*).

Meskipun *exhaustive search* secara teoritis menghasilkan solusi, namun waktu atau sumber daya yang dibutuhkan dalam pencarian solusinya sangat besar. Contoh dari *exhaustive search* adalah *Travelling Salesperson Problem* (TSP) dan 1/0 *Knapsack Problem*.

BAB 3
ALGORITMA BRUTE FORCE

Kode yang saya buat menggunakan pendekatan *brute force* untuk mencari solusi yang optimal dari permainan *Breach Protocol* yang paling optimal untuk setiap kombinasi matriks, sekuens, dan ukuran buffer (dalam konteks Tucil ini adalah mencari kemungkinan buffer berdasarkan kecocokan sekuens dengan pertimbangan menghasilkan bobot yang paling besar). Penjelasan lebih prosedural dan sederhana dari implementasi algoritma *brute force* adalah sebagai berikut:

1. Program dimulai dengan menginisialisasi variabel-variabel yang dibutuhkan, termasuk waktu mulai untuk menghitung durasi eksekusi, matriks input, daftar sekuen dan bobotnya, serta sebuah list untuk menyimpan jalur unik (`unique_paths`) dan hasilnya (`results`).
2. Program akan mencoba membangun jalur yang mungkin mulai dari baris pertama (paling atas).
3. Dari setiap titik awal, program membangun jalur baru dengan menambahkan langkah-langkah berikutnya secara bertahap. Ini dilakukan dengan menambahkan koordinat baru ke jalur saat ini berdasarkan aturan tertentu (bergantian antara bergerak vertikal dan horizontal).
4. Jalur hanya diperluas jika panjangnya tidak melebihi ukuran buffer yang ditentukan, dan jalur yang telah dihasilkan sebelumnya tidak diulangi (untuk menghindari duplikasi).
5. Untuk setiap jalur yang dihasilkan, program mengecek apakah jalur tersebut mengandung sekuen yang diberikan sebagai subsekuen untuk memastikan bahwa semua elemen sekuen muncul secara berurutan dalam jalur.
6. Setelah menentukan jalur yang mengandung sekuen yang valid, program menghitung bobot total jalur tersebut dengan menjumlahkan bobot untuk setiap sekuen yang ditemukan dalam jalur itu.
7. Dari semua jalur yang dihasilkan, program menentukan jalur dengan bobot maksimum yang dilakukan dengan membandingkan bobot dari setiap jalur yang valid dan memilih jalur (atau jalur-jalur) dengan bobot tertinggi.

BAB 4
SOURCE CODE PROGRAM

Pseudocode sederhana versi saya untuk potongan program yang berfokus pada algoritma *brute force* adalah sebagai berikut:

```
brute_force_algorithm(matrix, buffer_size, sequences)
    INITIALIZE rows, cols from the dimensions of matrix
    INITIALIZE unique_paths as an empty list
    INITIALIZE results as an empty list

    FOR each start_col in range of cols
        INITIALIZE paths with starting point at (0, start_col)

        WHILE paths is not empty
            INITIALIZE new_paths as an empty list

            FOR each path in paths
                CONVERT path to path_tuple
                      IF path length <= buffer_size AND path_tuple not in
unique_paths THEN
                    ADD path_tuple to unique_paths
                    ADD path to results

                IF path length == buffer_size THEN
                    CONTINUE to next iteration

                GET last_r, last_c from the last element of path
                IF path length % 2 == 1 THEN
                    FOR r in range of rows
                        IF (r, last_c) not in path THEN
                            APPEND new path to new_paths
                ELSE
                    FOR c in range of cols
                        IF (last_r, c) not in path THEN
                            APPEND new path to new_paths

                paths = new_paths

     TRANSFORM results into final_results by mapping coordinates to matrix
values

          FIND  max_weight  and  corresponding  max_weight_solutions  from
final_results
```

Selanjutnya, berikut adalah potongan kode lengkap dari program yang telah dibuat :

utility.py

```
import random
import time
```

```
import os

# Fungsi untuk menghasilkan jalur valid dari matriks sesuai dengan
buffer size dan sekuen yang diberikan
def generate_valid_paths(matrix, buffer_size, sequences,
input_file_name=None):
    # Catat waktu mulai eksekusi
    start_time = time.time()

    # Inisialisasi ukuran matriks dan array
    rows = len(matrix)
    cols = len(matrix[0])
    unique_paths = []
    results = []

    # Iterasi melalui setiap kolom di baris pertama untuk menentukan
titik awal
    for start_col in range(cols):
        # Setiap titik awal dijadikan sebagai jalur awal
        paths = [[(0, start_col)]]

        # Ekspansi jalur
        while paths:
            new_paths = []
            for path in paths:
                path_tuple = tuple(path)
                # Jika jalur tidak melebihi buffer size dan belum ada
sebelumnya
                if len(path) <= buffer_size and path_tuple not in
unique_paths:
                    unique_paths.append(path_tuple)
                    results.append(path)

                # Jika jalur sudah mencapai buffer size, lanjutkan ke
jalur berikutnya
                if len(path) == buffer_size:
                    continue

                # Tambahkan langkah baru ke jalur berdasarkan aturan
pergerakan (Vertikal - Horizontal)
                last_r, last_c = path[len(path)-1]
                if len(path) % 2 == 1: # Pergerakan vertikal
```

```
                    for r in range(rows):
                        if (r, last_c) not in path:
                            new_path = path + [(r, last_c)]
                            new_paths.append(new_path)
                else: # Pergerakan horizontal
                    for c in range(cols):
                        if (last_r, c) not in path:
                            new_path = path + [(last_r, c)]
                            new_paths.append(new_path)
            paths = new_paths

    # Konversi jalur menjadi sekuen matriks
    final_results = [[matrix[r][c] for r, c in path] for path in
results]

    # Fungsi untuk mengecek apakah jalur merupakan subsekuen dari
sekuen yang diberikan
    def is_subsequence(path, sequence):
        seq_index = 0
        for item in path:
            if item == sequence[seq_index]:
                seq_index += 1
                if seq_index == len(sequence):
                    return True
            else:
                if seq_index > 0:
                    seq_index = 0
                    if item == sequence[seq_index]:
                        seq_index += 1
        return False

    # Fungsi untuk menghitung total bobot dari jalur berdasarkan
sekuen yang ditemukan
    def calculate_weight(path, sequences):
        total_weight = 0
        for i in range(len(sequences)):
            seq = sequences[i][0]
            weight = sequences[i][1]
            if is_subsequence(path, seq):
                total_weight += weight
        return total_weight
```

```
    # Menemukan jalur yang menghasilkan bobot maksimum
    max_weight = 0
    max_weight_solutions = []
    for i in range(len(final_results)):
        path = final_results[i]
        weight = calculate_weight(path, sequences)
        if weight > max_weight:
            max_weight = weight
            max_weight_solutions = [(path, weight)]
        elif weight == max_weight:
            max_weight_solutions.append((path, weight))

    # Hitung waktu eksekusi
    execution_time = (time.time() - start_time) * 1000  # Waktu
eksekusi

    # Siapkan output
    output = ""
    if not max_weight_solutions or (max_weight_solutions and
max_weight_solutions[0][1] == 0):
        output += "Maaf, tidak ada sekuen yang berhasil
didapatkan.\n\n"
    else:
        max_weight = max_weight_solutions[0][1]
        output += f"{max_weight}\n"
        for path, weight in max_weight_solutions:
            if weight == max_weight:
                output += ' '.join(path) + "\n"
                for r, c in results[final_results.index(path)]:
                    output += f"{c + 1}, {r + 1}\n"
                break

    output += f"\n{execution_time:.2f} ms\n"

    # Mencetak output ke terminal
    print(output)

    # Tawarkan pengguna untuk menyimpan solusi
    save_prompt = input("Apakah ingin menyimpan solusi? (y/n): ")
    if save_prompt.lower() == 'y':
        save_solution(output, input_file_name)
```

```
# Fungsi untuk menyimpan solusi ke dalam berkas
def save_solution(output, input_file_name=None):
    # Menentukan direktori tempat menyimpan file solusi
    base_dir = os.path.dirname(__file__)
    solution_dir = os.path.join(base_dir, '..', 'test')

    if input_file_name:
        # Jika nama file input diberikan, buat nama file solusi
berdasarkan format yang saya buat (_solution.txt)
        base_name =
os.path.splitext(os.path.basename(input_file_name))[0]
        file_name = os.path.join(solution_dir,
f"{base_name}_solution.txt")
    else:
        # Jika tidak, minta pengguna memasukkan nama file
        file_name_input = input("Masukkan nama berkas (tanpa ekstensi
.txt): ")
        file_name = os.path.join(solution_dir,
f"{file_name_input}.txt")

    with open(file_name, 'w') as file:
        file.write(output)
    print(f"\nSolusi telah disimpan dalam berkas '{file_name}'.")

# Fungsi untuk menghasilkan data permainan secara otomatis dari input
pengguna
def generate_game_data():
    num_unique_tokens = int(input("\nMasukkan Jumlah Token Unik: "))
    tokens_input = input("Masukkan Daftar Token yang Anda Inginkan
(Pisahkan dengan Spasi): ")
    tokens = tokens_input.split()
    buffer_size = int(input("Masukkan Ukuran Buffer: "))
    matrix_size = input("Masukkan Ukuran Matriks (format:
lebarxtinggi): ")
    num_sequences = int(input("Masukkan Jumlah Sekuens: "))
    max_sequence_size = int(input("Masukkan Ukuran Maksimal Sekuens:
"))
    matrix_width, matrix_height = map(int, matrix_size.split())

    matrix = [[random.choice(tokens) for _ in range(matrix_width)]
for _ in range(matrix_height)]
```

```
    sequences = []
    for _ in range(num_sequences):
        sequence_length = random.randint(1, max_sequence_size)
        sequence = [random.choice(tokens) for _ in
range(sequence_length)]
        reward = random.randint(1, 100)
        sequences.append((sequence, reward))

    return buffer_size, matrix, sequences

# Fungsi untuk mencetak data permainan yang dihasilkan
def print_game_data(buffer_size, matrix, sequences):
    print("\nMatriks dan Sekuens Permainan berhasil dibuat!")
    print(f"\nBuffer Size: {buffer_size}\n")
    for row in matrix:
        print(' '.join(row))
    print("\nSekuens beserta hadiah yang dapat diperoleh:")
    for i, (sequence, reward) in enumerate(sequences, start=1):
        print(f"{i}. {' '.join(sequence)} dengan Bobot Hadiah
{reward}")

# Fungsi untuk membaca data permainan dari berkas
def input_file(file_name):
    base_dir = os.path.dirname(__file__)
    file_path = os.path.join(base_dir, '..', 'test', file_name)

    with open(file_path, 'r') as file:
        buffer_size = int(file.readline().strip())
        matrix_width, matrix_height = map(int,
file.readline().strip().split())

        matrix = []
        for _ in range(matrix_height):
            row = file.readline().strip().split()
            matrix.append(row)

        number_of_sequences = int(file.readline().strip())
        sequences = []
        for _ in range(number_of_sequences):
            sequence = file.readline().strip().split()
            reward = int(file.readline().strip())
            sequences.append((sequence, reward))
```

```
    return buffer_size, matrix, sequences
 random
import time
import os

# Fungsi untuk menghasilkan jalur valid dari matriks sesuai dengan
buffer size dan sekuen yang diberikan
def generate_valid_paths(matrix, buffer_size, sequences,
input_file_name=None):
    # Catat waktu mulai eksekusi
    start_time = time.time()

    # Inisialisasi ukuran matriks dan array
    rows = len(matrix)
    cols = len(matrix[0])
    unique_paths = []
    results = []

    # Iterasi melalui setiap kolom di baris pertama untuk menentukan
titik awal
    for start_col in range(cols):
        # Setiap titik awal dijadikan sebagai jalur awal
        paths = [[(0, start_col)]]

        # Ekspansi jalur
        while paths:
            new_paths = []
            for path in paths:
                path_tuple = tuple(path)
                # Jika jalur tidak melebihi buffer size dan belum ada
sebelumnya
                if len(path) <= buffer_size and path_tuple not in
unique_paths:
                    unique_paths.append(path_tuple)
                    results.append(path)

                # Jika jalur sudah mencapai buffer size, lanjutkan ke
jalur berikutnya
                if len(path) == buffer_size:
                    continue
```

```
                # Tambahkan langkah baru ke jalur berdasarkan aturan
pergerakan (Vertikal - Horizontal)
                last_r, last_c = path[len(path)-1]
                if len(path) % 2 == 1: # Pergerakan vertikal
                    for r in range(rows):
                        if (r, last_c) not in path:
                            new_path = path + [(r, last_c)]
                            new_paths.append(new_path)
                else: # Pergerakan horizontal
                    for c in range(cols):
                        if (last_r, c) not in path:
                            new_path = path + [(last_r, c)]
                            new_paths.append(new_path)
            paths = new_paths

    # Konversi jalur menjadi sekuen matriks
    final_results = [[matrix[r][c] for r, c in path] for path in
results]

    # Fungsi untuk mengecek apakah jalur merupakan subsekuen dari
sekuen yang diberikan
    def is_subsequence(path, sequence):
        seq_index = 0
        for item in path:
            if item == sequence[seq_index]:
                seq_index += 1
                if seq_index == len(sequence):
                    return True
            else:
                if seq_index > 0:
                    seq_index = 0
                    if item == sequence[seq_index]:
                        seq_index += 1
        return False

    # Fungsi untuk menghitung total bobot dari jalur berdasarkan
sekuen yang ditemukan
    def calculate_weight(path, sequences):
        total_weight = 0
        for i in range(len(sequences)):
            seq = sequences[i][0]
            weight = sequences[i][1]
```

```
            if is_subsequence(path, seq):
                total_weight += weight
        return total_weight

    # Menemukan jalur yang menghasilkan bobot maksimum
    max_weight = 0
    max_weight_solutions = []
    for i in range(len(final_results)):
        path = final_results[i]
        weight = calculate_weight(path, sequences)
        if weight > max_weight:
            max_weight = weight
            max_weight_solutions = [(path, weight)]
        elif weight == max_weight:
            max_weight_solutions.append((path, weight))

    # Hitung waktu eksekusi
    execution_time = (time.time() - start_time) * 1000  # Waktu
eksekusi

    # Siapkan output
    output = ""
    if not max_weight_solutions or (max_weight_solutions and
max_weight_solutions[0][1] == 0):
        output += "Maaf, tidak ada sekuen yang berhasil
didapatkan.\n\n"
    else:
        max_weight = max_weight_solutions[0][1]
        output += f"{max_weight}\n"
        for path, weight in max_weight_solutions:
            if weight == max_weight:
                output += ' '.join(path) + "\n"
                for r, c in results[final_results.index(path)]:
                    output += f"{c + 1}, {r + 1}\n"
                break

    output += f"\n{execution_time:.2f} ms\n"

    # Mencetak output ke terminal
    print(output)

    # Tawarkan pengguna untuk menyimpan solusi
```

```
    save_prompt = input("Apakah ingin menyimpan solusi? (y/n): ")
    if save_prompt.lower() == 'y':
        save_solution(output, input_file_name)

# Fungsi untuk menyimpan solusi ke dalam berkas
def save_solution(output, input_file_name=None):
    # Menentukan direktori tempat menyimpan file solusi
    base_dir = os.path.dirname(__file__)
    solution_dir = os.path.join(base_dir, '..', 'test')

    if input_file_name:
        # Jika nama file input diberikan, buat nama file solusi
berdasarkan format yang saya buat (_solution.txt)
        base_name =
os.path.splitext(os.path.basename(input_file_name))[0]
        file_name = os.path.join(solution_dir,
f"{base_name}_solution.txt")
    else:
        # Jika tidak, minta pengguna memasukkan nama file
        file_name_input = input("Masukkan nama berkas (tanpa ekstensi
.txt): ")
        file_name = os.path.join(solution_dir,
f"{file_name_input}.txt")

    with open(file_name, 'w') as file:
        file.write(output)
    print(f"\nSolusi telah disimpan dalam berkas '{file_name}'.")

# Fungsi untuk menghasilkan data permainan secara otomatis dari input
pengguna
def generate_game_data():
    num_unique_tokens = int(input("\nMasukkan Jumlah Token Unik: "))
    tokens_input = input("Masukkan Daftar Token yang Anda Inginkan
(Pisahkan dengan Spasi): ")
    tokens = tokens_input.split()
    buffer_size = int(input("Masukkan Ukuran Buffer: "))
    matrix_size = input("Masukkan Ukuran Matriks (format:
lebarxtinggi): ")
    num_sequences = int(input("Masukkan Jumlah Sekuens: "))
    max_sequence_size = int(input("Masukkan Ukuran Maksimal Sekuens:
"))
    matrix_width, matrix_height = map(int, matrix_size.split())
```

```
    matrix = [[random.choice(tokens) for _ in range(matrix_width)]
for _ in range(matrix_height)]

    sequences = []
    for _ in range(num_sequences):
        sequence_length = random.randint(1, max_sequence_size)
        sequence = [random.choice(tokens) for _ in
range(sequence_length)]
        reward = random.randint(1, 100)
        sequences.append((sequence, reward))

    return buffer_size, matrix, sequences

# Fungsi untuk mencetak data permainan yang dihasilkan
def print_game_data(buffer_size, matrix, sequences):
    print("\nMatriks dan Sekuens Permainan berhasil dibuat!")
    print(f"\nBuffer Size: {buffer_size}\n")
    for row in matrix:
        print(' '.join(row))
    print("\nSekuens beserta hadiah yang dapat diperoleh:")
    for i, (sequence, reward) in enumerate(sequences, start=1):
        print(f"{i}. {' '.join(sequence)} dengan Bobot Hadiah
{reward}")

# Fungsi untuk membaca data permainan dari berkas
def input_file(file_name):
    base_dir = os.path.dirname(__file__)
    file_path = os.path.join(base_dir, '..', 'test', file_name)

    with open(file_path, 'r') as file:
        buffer_size = int(file.readline().strip())
        matrix_width, matrix_height = map(int,
file.readline().strip().split())

        matrix = []
        for _ in range(matrix_height):
            row = file.readline().strip().split()
            matrix.append(row)

        number_of_sequences = int(file.readline().strip())
        sequences = []
```

```
        for _ in range(number_of_sequences):
            sequence = file.readline().strip().split()
            reward = int(file.readline().strip())
            sequences.append((sequence, reward))

    return buffer_size, matrix, sequences
```

main.py

```
from utility import *
import sys, os

def art():
    print("""
╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗ ╔═══╗
╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝ ╚═══╝

Welcome to Breach Protocol Solver!
""")
def print_menu():
    print("""
Menu:
[1] berkas .txt
[2] CLI input (Auto Generate)

[E] Exit
    """)

def main():
    base_dir = os.path.dirname(__file__)
    test_dir = os.path.join(base_dir, '..', 'test')

    while True:
        art()
        print_menu()
        command = input("Masukkan pilihan Anda: ")

        if command == '1':
            file_name = input("\nMasukkan nama berkas/file
ber-ekstensi .txt: ")
```

```
            file_path = os.path.join(test_dir, file_name)

            if os.path.isfile(file_path):
                buffer_size, matrix, sequences =
input_file(file_path)
                print("\nTunggu sebentar, program sedang mencari
solusi ...\n")
                generate_valid_paths(matrix, buffer_size, sequences,
file_path)
            else:
                print(f"\nFile '{file_path}' tidak ditemukan. Silakan
coba lagi.")
        elif command == '2':
            buffer_size, matrix, sequences = generate_game_data()
            print_game_data(buffer_size, matrix, sequences)
            print("\nTunggu sebentar, program sedang mencari solusi
...\n")
            generate_valid_paths(matrix, buffer_size, sequences)
        elif command.lower() == 'e':
            print("\nKeluar dari program...\n")
            sys.exit()
        else:
            print("\nPilihan tidak valid, silakan coba lagi.\n")


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

BAB 5

TESTING PROGRAM

Test Case	Result
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 1C 15 BD 7A BD 20 BD 1C BD 55 30	50 7A BD 7A BD 1C BD 55 1, 1 1, 4 3, 4 3, 5 6, 5 6, 3 1, 3 315891.34 ms
7 6 6 BD 1C BD E9 7A 1C 55 1C 7A BD 7A 55 BD 7A E9 55 1C 7A E9 BD 1C 55 1C 7A BD E9 7A 7A 7A 55 55 E9 1C 1C 1C 1C 3 BD 55 E9 39 E9 55 50 55 E9 27	116 BD 55 E9 E9 55 1, 1 1, 6 2, 6 2, 5 6, 5 388197.25 ms
5 4 7 55 E9 BD 55 55 1C 7A 7A 1C 55 E9 1C 7A 7A 1C 1C 7A E9 E9 55 BD 1C 7A 55 1C BD 1C 1C 4	214 BD 7A 1C 1C 55 3, 1 3, 2 2, 2 2, 6 4, 6 83.05 ms

7A 1C 87 1C 1C 55 100 BD 27 7A 66 7A 10	
5 4 7 55 E9 BD 55 55 1C 7A 7A 1C 55 E9 1C 7A 7A 1C 1C 7A E9 E9 55 BD 1C 7A 55 1C BD 1C 1C 1 7A 7A 7A 7A 50	Masukkan nama berkas/file ber-ekstensi .txt: testcase4.txt Tunggu sebentar, program sedang mencari solusi ... Maaf, tidak ada sekuen yang berhasil didapatkan. 58.19 ms
6 5 6 1C 7A 1C 1C 55 55 1C E9 7A E9 55 7A BD BD 7A 1C 55 55 55 E9 1C E9 7A 1C BD BD BD 55 7A BD 2 BD E9 BD BD 7A 19 1C 55 3	19 55 BD E9 BD BD 7A 5, 1 5, 5 2, 5 2, 6 5, 6 5, 3 2057.03 ms
6 6 4 55 7A 1C E9 1C BD 7A 1C BD 1C BD 55 BD 1C 1C 7A E9 55 E9 55 E9 E9 E9 E9 3 7A E9 1C 80 1C 12 BD 1C 7A 7A BD 87	92 55 BD 7A E9 1C 1, 1 1, 3 4, 3 4, 1 3, 1 752.09 ms

BAB 6
KESIMPULAN

5.1 Kesimpulan

Sebagai algoritma "sapu jagad" yang ampuh menyelesaikan segala jenis permasalahan, Pendekatan brute force dalam kasus ini menunjukkan keefektifannya dalam mencari semua kemungkinan jalur yang mungkin dan mengevaluasi setiap jalur berdasarkan kriteria yang diberikan, yaitu menemukan solusi dari permainan *Breach Protocol* yang paling optimal untuk setiap kombinasi matriks, sekuens, dan ukuran buffer. Namun, pendekatan ini juga mungkin tidak efisien dalam hal waktu eksekusi, terutama untuk matriks berukuran besar dan/atau ukuran buffer yang besar, karena jumlah kombinasi jalur yang harus dievaluasi dapat menjadi sangat besar.

5.2 Saran

Saya merasa program yang diselesaikan masih jauh dari kata sempurna. Untuk kedepannya, masih banyak pengembangan yang dapat dilakukan untuk memaksimalkan program ini. Dengan memperhatikan fungsi, memanfaatkan kegunaan untuk hal lain, memperbaiki apa yang masih kurang, dan eksekusi yang lebih baik dalam tugas berikutnya.

5.3 Komentar

Saya mengucapkan terima kasih kepada asisten-asisten yang bertanggung jawab terhadap pelaksanaan tugas kecil ini serta sangat membantu dalam proses mengerjakan tugas kecil pertama di semester 4 Teknik Informatika ITB (bahkan di minggu pertama perkuliahan).

5.4 Refleksi

Tugas kecil pertama ini telah memberikan pengalaman berharga bagi saya atas situasi dan tantangan yang muncul. Salah satu kendala yang saya hadapi adalah alokasi waktu pengerjaan yang kurang efisien dan efektif karena bertabrakan dengan hari raya Imlek, ditambah kemampuan adaptasi untuk memutar otak kembali setelah berbulan-bulan libur kuliah ("*beku banget nih otak :)*"). Selain itu, target yang ingin dicapai juga menjadi tantangan tersendiri, "*meskipun GUI tidak jadi terealisasikan xixixi*". Setelah mengerjakan tugas kecil perdana ini, saya semakin *aware* dengan *time management* dan *planning* yang

cukup matang selama mengerjakan tugas-tugas kedepannya yang konon semakin banyak dan semakin gila. Saya percaya bahwa segala bentuk kerja keras dan *struggle* selama mengerjakan tugas ini dapat memotivasi diri saya untuk jauh lebih maksimal di tugas-tugas kecil atau besar berikutnya di Teknik Informatika ITB.

REFERENSI

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

LAMPIRAN

Link Repository GitHub :
https://github.com/slntkllr01/Cyberpunk-2077-Breach-Protocol

Progress Tracking

No	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		✔