LAPORAN TUGAS KECIL 01

IF2211 STRATEGI ALGORITMA

“Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force”

Disusun oleh:

Satriadhikara Panji Yudhistira K-03 13522125

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

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.

Dalam permainan ini, pemain harus menavigasi matriks token untuk mencocokkan sekuens tertentu dalam buffer dengan bergerak sesuai pola tertentu. Tantangan utama adalah mencari solusi paling optimal yang memaksimalkan bobot hadiah yang didapat, dengan mempertimbangkan kombinasi matriks, sekuens, dan ukuran buffer yang diberikan. Program yang dikembangkan diharapkan dapat menemukan solusi ini menggunakan pendekatan brute force, menghasilkan bobot hadiah maksimal, isi buffer, koordinat token, dan waktu eksekusi sebagai outputnya.

BAB 2
TEORI SINGKAT

2.1 Algoritma Brute Force

Pendekatan brute force adalah metode pemecahan masalah yang sistematis, di mana semua kemungkinan solusi untuk masalah tersebut dijelajahi hingga solusi optimal ditemukan. Dalam konteks Cyberpunk 2077 Breach Protocol, ini berarti mencoba setiap kombinasi mungkin dari gerakan dalam matriks token untuk mencocokkan sekuens dalam buffer sesuai aturan permainan. Pendekatan ini, meskipun tidak efisien dari segi waktu eksekusi dibandingkan dengan algoritma yang lebih canggih, menjamin penemuan solusi terbaik yang memaksimalkan bobot hadiah yang didapat, asalkan waktu dan sumber daya komputasi mencukupi.

BAB 3
IMPLEMENTASI PROGRAM

Program ini dibuat dalam bahasa *python*. Dikarenakan program ini harus memakai pendekatan *brute force,* maka pertama-tama program ini mencari semua kemungkinan kombinasi sesuai aturan yang berlaku di spesifikasi. Kemudian program akan melanjutkan dengan mengecek semua kemungkinan tersebut satu-satu dengan *sequences,* kalau ada *sequence* yang ada di dalam kemungkinan tersebut akan menambah nilai dari kemungkinan yang lagi di cek. Program ini mengecek semua kemungkinan dan hasil nilai nya kemudian dibandingkan satu-satu nilai yang terbesarnya (paling optimal). Setelah semua proses sudah selesai program akan mengeluarkan hasil kemungkinan yang paling optimal dan koordinatnya dan juga berapa lama waktunya.

Program ini memakai tipe data abstrak pada tipe *Sequence* yang berisi *token* yaitu *list of string* dan *reward* yaitu int dan *Data* yang berisi data-data yang diperlukan seperti *buffer_size, matrix*, dll.

Program ini awalnya menanyakan ke user tentang metode input yang dimau, kemudian program akan mengambil data sesuai pilihan user (file atau terminal). Kemudian timer langsung dimulai dan lanjut ke fungsi yang mencari semua kemungkinan dari matriks, Fungsi ini menggunakan rekursi untuk menjelajahi setiap kemungkinan path dalam matriks, bergantian antara gerakan vertikal dan horizontal. Setiap langkah, fungsi mencatat token dan posisinya, memastikan tidak mengunjungi posisi yang sama lebih dari satu kali. Kemudian semua kombinasi tersebut dicek lagi untuk mencari solusi optimalnya yaitu dengan mencari apakah dalam kombinasi yang sedang dicek ada *sequence* kemudian membandingkan semua nilainya.

Program ini langsung mengeluarkan hasil optimalnya setelah algoritmanya selesai, tetapi user bisa memilih untuk menyimpan hasilnya ke dalam fail txt. Berikut merupakan *source code* dari program ini (untuk lebih jelas ada di github):

```
from __future__ import print_function, unicode_literals
from PyInquirer import prompt
from typing import List, Tuple, Set, Dict, Optional
import time
from pyfiglet import Figlet
import random
class Sequence:
    def __init__(self) -> None:
        self.tokens: List[str] = []
        self.reward: int = 0
class Data:
    def __init__(self) -> None:
        self.buffer_size: int = 0
        self.matrix_width: int = 0
        self.matrix_height: int = 0
        self.matrix: List[List[str]] = []
        self.number_of_sequences: int = 0
        self.sequences: List[Sequence] = []
def main() -> None:
    input_method: Optional[str] = welcome()
    if input_method == "Load from file":
        data: Data = load_data_from_file()
    else:
        data: Data = load_data_from_input()
    # Start timer
    start_time: float = time.time()
    all_possibilities: List[List[Tuple[str, Tuple[int, int]]]] = (
        generate_all_posibilities(data.matrix, data.buffer_size - 1)
    )
    result: Optional[Tuple[List[Tuple[str, Tuple[int, int]]], int]] = calculate_optimal(
        all_possibilities, data.sequences
    )
    # End timer
    end_time: float = time.time()
    runtime: float = (end_time - start_time) * 1000
    if result:
        if input_method == "Load from input":
            output_result: str = output(result, runtime, data.matrix, data.sequences)
        else:
            output_result: str = output(result, runtime)
        print("\n" + output_result + "\n")
        output_to_file(output_result)
def output_to_file(result: str) -> None:
    main_question: List[Dict[str, str | bool]] = [
        {
            "type": "confirm",
            "message": "Do you want to save the solution?",
            "name": "save",
            "default": False,
        },
    ]
```

```
    if prompt.prompt(main_question)["save"]:
        sub_question: List[Dict[str, str]] = [
            {
                "type": "input",
                "name": "filename",
                "message": "What would you like to name the file? (.txt)",
            }
        ]
        with open(
            f"../test/{prompt.prompt(sub_question)['filename']}.txt", "w"
        ) as file:
            file.write(result)
def load_data_from_input() -> Data:
    data: Data = Data()
    questions: List[Dict[str, str | function]] = [
        {
            "type": "input",
            "name": "number_of_unique_token",
            "message": "Number of unique token?",
            "validate": lambda val: val.isdigit() and int(val) > 0,
        }
    ]
    answer: Dict[str, str] = prompt.prompt(questions)
    number_of_unique_token: int = int(answer["number_of_unique_token"])
    questions: List[Dict[str, str | function]] = [
        {
            "type": "input",
            "name": "tokens",
            "message": "Enter the tokens:",
            "validate": lambda val: len(val.split()) == number_of_unique_token,
        },
        {
            "type": "input",
            "name": "buffer_size",
            "message": "Enter the buffer size:",
            "validate": lambda val: val.isdigit() and int(val) > 0,
        },
        {
            "type": "input",
            "name": "matrix_size",
            "message": "Enter the matrix size (m x n):",
            "validate": lambda val: len(val.split()) == 2
            and all(i.isdigit() for i in val.split()),
        },
        {
            "type": "input",
            "name": "number_of_sequences",
            "message": "Enter the number of sequences:",
            "validate": lambda val: val.isdigit() and int(val) > 0,
        },
        {
```

```
            "type": "input",
            "name": "max_size_sequence",
            "message": "Enter the maximum size of the sequence:",
            "validate": lambda val: val.isdigit() and int(val) > 1,
        },
    ]
    answer.update(prompt.prompt(questions))
    data.buffer_size = int(answer["buffer_size"])
    data.matrix_width, data.matrix_height = map(int, answer["matrix_size"].split())
    data.number_of_sequences = int(answer["number_of_sequences"])
    tokens: List[str] = answer["tokens"].split()
    data.matrix = [
        [random.choice(tokens) for _ in range(data.matrix_width)]
        for _ in range(data.matrix_height)
    ]
    for _ in range(data.number_of_sequences):
        sequence = Sequence()
        sequence.tokens = [
            random.choice(tokens)
            for _ in range(random.randint(2, int(answer["max_size_sequence"])))
        ]
        sequence.reward = random.randint(10, 50)
        data.sequences.append(sequence)
    return data
def output(
    result: Tuple[List[Tuple[str, Tuple[int, int]]], int],
    runtime: float,
    matrix: List[List[str]] | None = None,
    sequences: List[Sequence] | None = None,
) -> str:
    outcome: str = ""
    if matrix and sequences:
        outcome += "Matrix\n"
        max_length = max(len(token) for row in matrix for token in row)
        outcome += "\n".join(
            " ".join(token.ljust(max_length) for token in row) for row in matrix
        )
        outcome += "\n\n"
        outcome += "Sequences\n"
        for sequence in sequences:
            outcome += (
                " ".join(token.ljust(max_length) for token in sequence.tokens) + "\n"
            )
            outcome += f"{sequence.reward}\n"
        outcome += "\n"
    outcome += f"{result[1]}\n"
    for token in result[0]:
        outcome += token[0]
        if token == result[0][-1]:
            outcome += "\n"
        else:
```

```
            outcome += " "
    for position in result[0]:
        outcome += f"{position[1][0] + 1},{position[1][1] + 1}\n"
    outcome += "\n"
    outcome += f"{runtime:.2f} ms"
    return outcome
def welcome() -> str | None:
    header = Figlet(font="cyberlarge")
    print(header.renderText("Welcome to\nCyberpunk Breach Protocol"))
    main_question: List[Dict[str, str | List[str]]] = [
        {
            "type": "list",
            "name": "input_method",
            "message": "What method would you like to use to start the game?",
            "choices": [
                "Load from file",
                "Load from input",
                "Quit",
            ],
        }
    ]
    answers: Dict[str, str] = prompt.prompt(main_question)
    if answers["input_method"] == "Quit":
        print("Goodbye!")
        exit()
    return answers["input_method"]
def is_sublist(list1: List[str], list2: List[str]) -> bool:
    len_sublist: int = len(list1)
    len_list: int = len(list2)
    for i in range(len_list - len_sublist + 1):
        if list2[i : i + len_sublist] == list1:
            return True
    return False
def calculate_optimal(
    all_possibilities: List[List[Tuple[str, Tuple[int, int]]]],
    sequences: List[Sequence],
) -> Tuple[List[Tuple[str, Tuple[int, int]]], int] | None:
    optimal: Tuple[List[Tuple[str, Tuple[int, int]]], int] | None = None
    for list in all_possibilities:
        temp: List[str] = []
        for token in list:
            temp.append(token[0])
        total_reward: int = 0
        for sequence in sequences:
            if is_sublist(sequence.tokens, temp):
                total_reward += sequence.reward
        if total_reward != 0:
            if optimal:
                if total_reward > optimal[1]:
                    optimal = (list, total_reward)
            else:
```

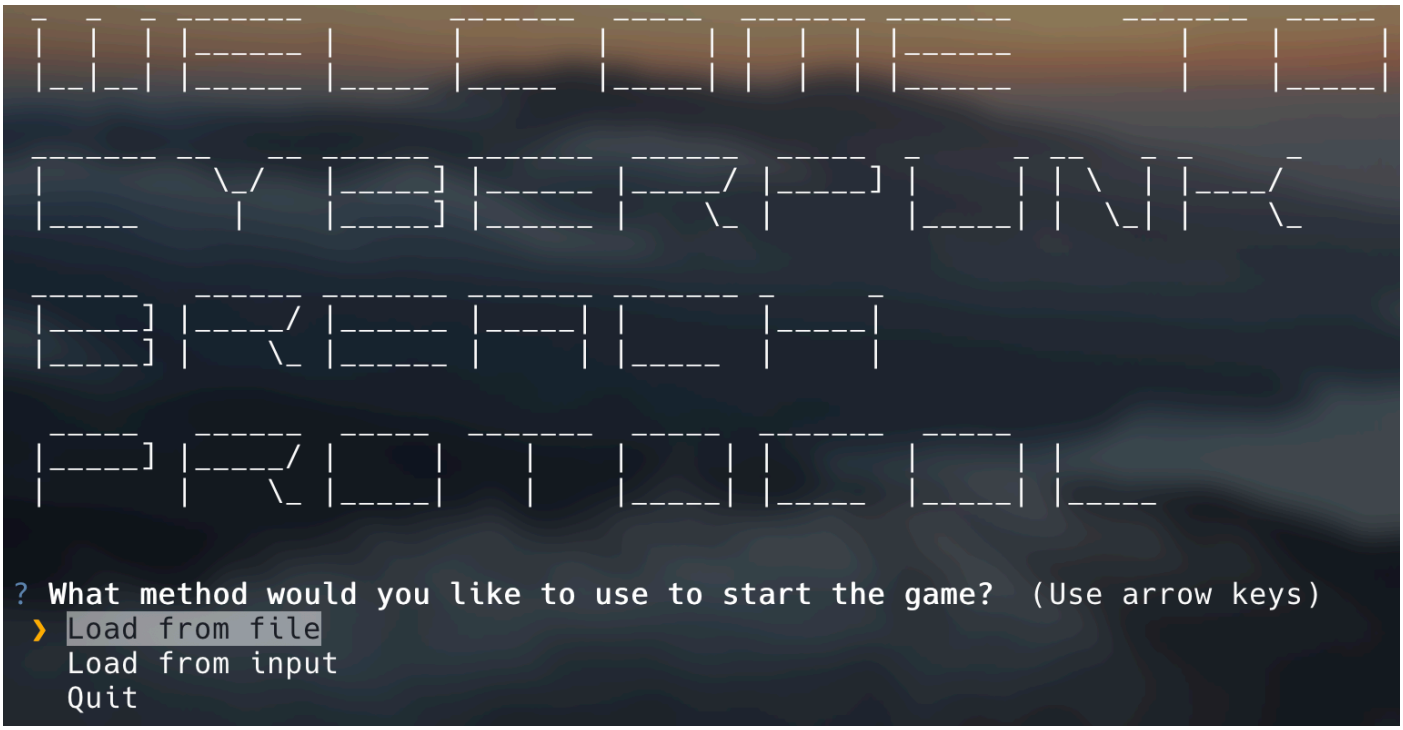
```
            optimal = (list, total_reward)
    return optimal
def load_data_from_file() -> Data:
    data: Data = Data()
    questions: List[Dict[str, str]] = [
        {
            "type": "input",
            "name": "filename",
            "message": "What is the name of the file you would like to load? (.txt)",
        }
    ]
    filename: str = prompt.prompt(questions)["filename"]
    with open(f"../test/{filename}.txt", "r") as file:
        data.buffer_size = int(file.readline())
        data.matrix_width, data.matrix_height = map(int, file.readline().split())
        for _ in range(data.matrix_height):
            data.matrix.append(list(file.readline().split()))
        data.number_of_sequences = int(file.readline())
        for _ in range(data.number_of_sequences):
            sequence: Sequence = Sequence()
            sequence.tokens = file.readline().split()
            sequence.reward = int(file.readline())
            data.sequences.append(sequence)
    return data
def generate_all_posibilities(
    matrix: List[List[str]], step: int
) -> List[List[Tuple[str, Tuple[int, int]]]]:
    rows: int = len(matrix)
    cols: int = len(matrix[0])
    all_paths: List[List[Tuple[str, Tuple[int, int]]]] = []
    def generate_paths(
        start_x: int,
        start_y: int,
        path: List[Tuple[str, Tuple[int, int]]] = [],
        visited: Set[Tuple[int, int]] = set(),
        direction: str = "vertical",
        steps: int = step,
    ) -> None:
        if steps == 0:
            all_paths.append(path.copy())
            return
        if direction == "vertical":
            for next_y in range(rows):
                if (start_x, next_y) not in visited:
                    generate_paths(
                        start_x,
                        next_y,
                        path + [(matrix[next_y][start_x], (start_x, next_y))],
                        visited | {(start_x, next_y)},
                        "horizontal",
                        steps - 1,
```

```
                )
        else:  # horizontal
            for next_x in range(cols):
                if (next_x, start_y) not in visited:
                    generate_paths(
                        next_x,
                        start_y,
                        path + [(matrix[start_y][next_x], (next_x, start_y))],
                        visited | {(next_x, start_y)},
                        "vertical",
                        steps - 1,
                    )
    for x in range(cols):
        generate_paths(
            x,
            0,
            path=[(matrix[0][x], (x, 0))],
            visited={(x, 0)},
            direction="vertical",
            steps=step,
        )
    return all_paths
if __name__ == "__main__":
    main()
```

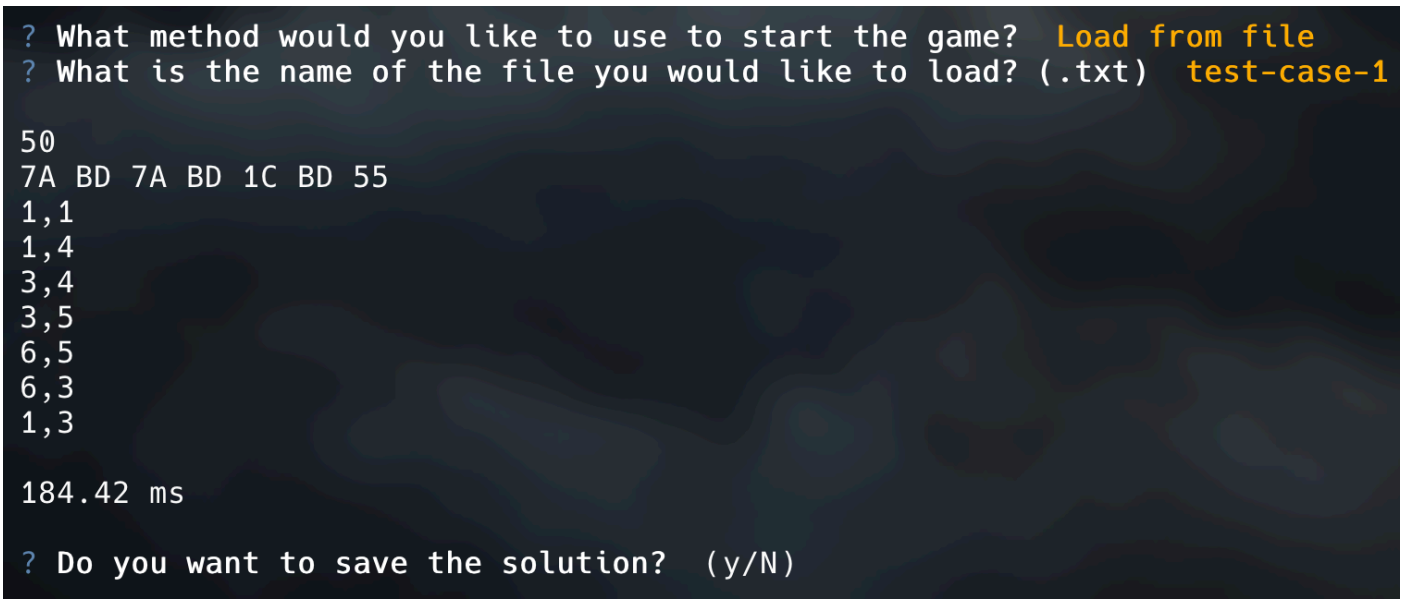
BAB 4

EKSPERIMEN

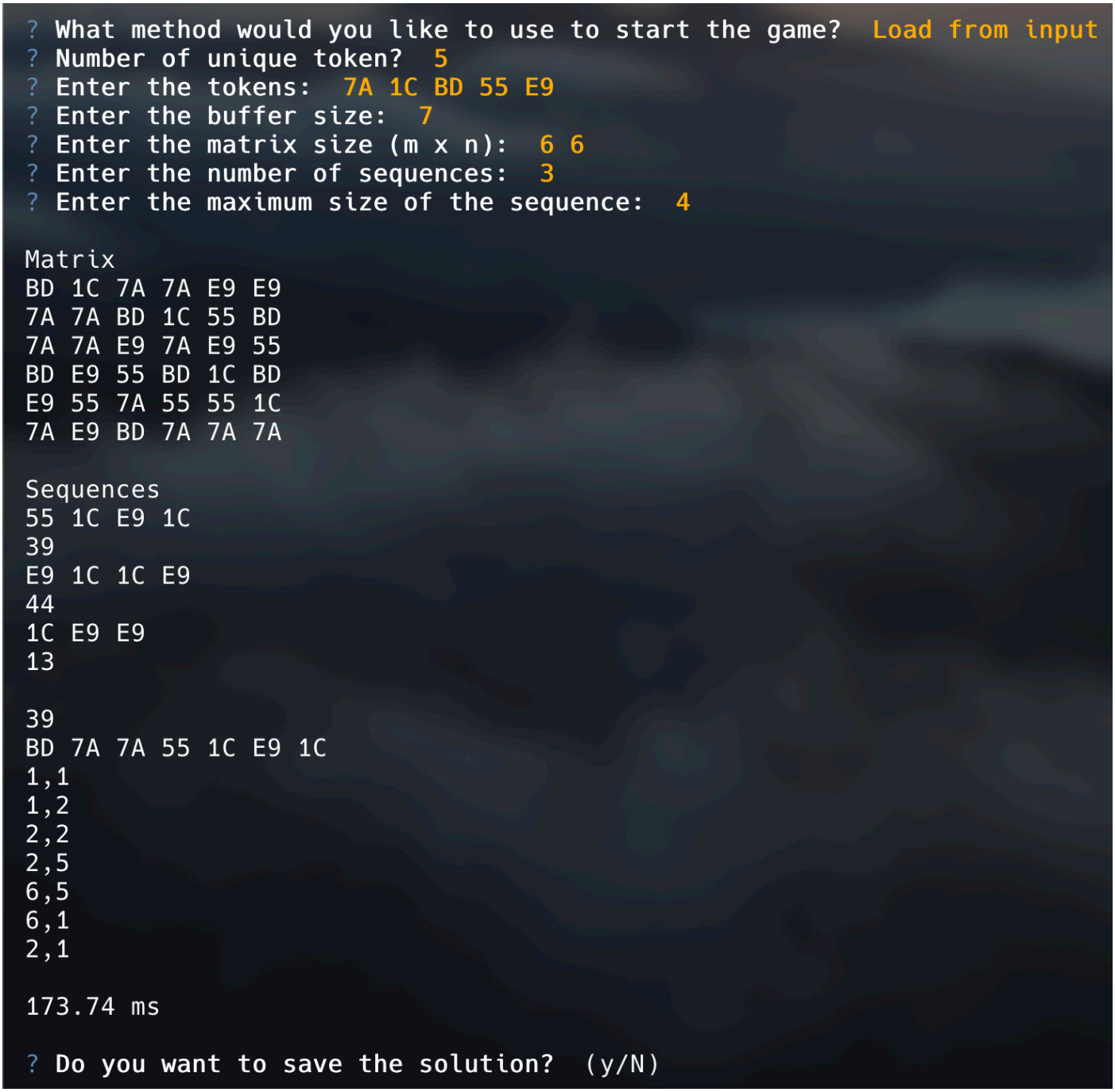
Inisiasi Program



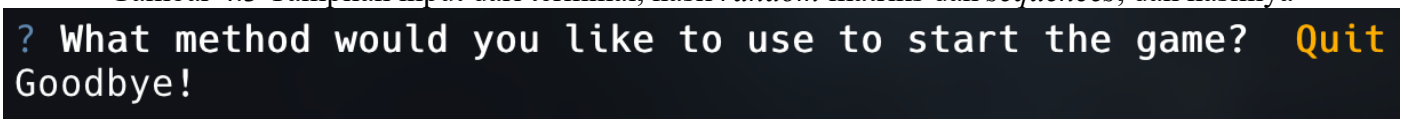
Gambar 4.1 Tampilan awal



Gambar 4.2 Tampilan input dari file dan hasilnya



Gambar 4.3 Tampilan input dari terminal, hasil *random* matriks dan *sequences*, dan hasilnya



Gambar 4.4 Tampilan keluar dari game

BAB 5
PENUTUP

5.1 Kesimpulan

Pendekatan brute force merupakan metode efektif untuk menjamin penemuan solusi optimal. Meskipun pendekatan ini mungkin tidak efisien dari segi waktu eksekusi, ia memungkinkan pemahaman mendalam tentang semua kemungkinan solusi dan pemilihan yang paling menguntungkan berdasarkan bobot hadiah. Hal ini menunjukkan pentingnya pemilihan strategi yang tepat dalam pemecahan masalah komputasi, khususnya dalam situasi yang membutuhkan optimasi hasil dalam kondisi terbatas.

5.2 Saran

Sebagai saran untuk pengembangan lebih lanjut, pertimbangkan penggunaan algoritma yang lebih efisien dari brute force untuk mengatasi masalah dengan kompleksitas tinggi atau batasan waktu eksekusi. Algoritma seperti Dynamic Programming, Greedy, atau teknik pemrograman lanjutan lainnya dapat meningkatkan efisiensi dan mengurangi waktu eksekusi sambil masih mempertahankan kemampuan untuk menemukan solusi optimal. Selain itu, eksplorasi pada pengoptimalan kode dan penggunaan sumber daya komputasi secara efektif juga dapat membantu dalam menangani masalah skala besar.

5.3 Link Repository

Link repository untuk tugas kecil 1 mata kuliah IF2211 Strategi Algoritma adalah sebagai berikut

Link : https://github.com/satriadhikara/Tucil1_13522125

5.4 Tabel Checkpoint 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 optimal	✓	
6. Program dapat menyimpan solusi dalam berkas .txt	✓	
7. Program memiliki GUI		

DAFTAR REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Tucil1-2024.pdf