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

Disusun Oleh:
13522070 - Marzuli Suhada M

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
TAHUN AJARAN 2023/2024

Daftar Isi

Bab I

Deskripsi Masalah dan Algoritma

1.1 Algoritma Brute Force

Algoritma brute force adalah pendekatan komputasional yang melibatkan pemeriksaan atau pencarian secara langsung melalui semua kemungkinan solusi dalam sebuah permasalahan. Metode ini secara sistematis mengevaluasi setiap kemungkinan solusi untuk mencapai solusi yang diinginkan tanpa memperhatikan kompleksitas atau keunikan struktur masalah. Meskipun dapat dianggap sebagai pendekatan yang sederhana dan sering kali tidak efisien, algoritma brute force umumnya dapat memberikan solusi yang akurat untuk masalah yang kompleks. Namun, kelemahannya adalah kinerja yang buruk dalam kasus di mana jumlah solusi yang mungkin sangat besar, karena algoritma tersebut harus memeriksa setiap kemungkinan secara terpisah. Hal ini dapat dilihat dari waktu eksekusinya dimana semakin besar ukuran yang ingin dicari solusinya maka semakin banyak kemungkinan-kemungkinan yang ditetapkan.

1.2 Cyberpunk 2077 Breach Protocol

Mini games Breach Protocol di Cyberpunk 2077 adalah simulasi hacking yang terdiri dari serangkaian tugas-tugas yang harus diselesaikan oleh pemain untuk mengakses sistem keamanan atau mendapatkan informasi rahasia. Dalam mini games ini, pemain diberi tugas untuk menentukan buffer yang berisikan sequence-sequence yang jika digabungkan akan memberikan reward paling maksimal namun dengan ukuran yang paling minimal. Breach Protocol menggabungkan elemen pemecahan teka-teki, kecepatan, dan taktik untuk memberikan pengalaman hacking yang menantang dan mendalam bagi pemain dalam dunia futuristic Cyberpunk 2077.

Gambar 1 Permainan Breach Protocol
(Sumber: https://cyberpunk.fandom.com/wiki/Quickhacking)

1.3 Algoritma Penyelesaian Cyberpunk 2077 Breach Protocol dengan Pendekatan Brute Force

Algoritma penyelesaian Cyberpunk 2077 Breach Protocol dengan pendekatan brute force akan mencoba semua kombinasi kemungkinan untuk mencapai reward dari target yang paling optimal dengan keterbatasan pada jumlah buffer tokennya. Berikut adalah langkah-langkah algoritma brute force untuk menyelesaikan Breach Protocol:

1. Pertama, identifikasi lintasan yang ingin diakses atau data yang ingin diambil dari sistem keamanan yang dimulai dari token-token pada baris utama matriks.
2. Perhatikan sequence yang disediakan dalam mini game. Sequence ini terdiri dari token-token yang memiliki jumlah masing-masing dan reward yang berbeda-beda.
3. Mulai looping melalui semua kemungkinan kombinasi perintah dalam setiap baris pertama dengan lintasan yang bergerak horizontal, vertikal secara bergantian. Kemungkinan lintasan ini diuji satu per satu.
4. Untuk setiap kemungkinan, periksa apakah itu memenuhi syarat validitas. Misalnya, apakah memenuhi syarat jumlah buffernya atau rewardnya yang terbesar.
5. Evaluasi setiap kemungkinan. Misalnya, apakah lintasan tersebut sudah sesuai dengan ketentuan-ketentuan yang diberikan.
6. Simpan kemungkinan lintasan yang berhasil mendapatkan reward yang optimal atau memberikan hasil yang diinginkan.
7. Keluarkan atau tampilkan kemungkinan lintasan yang paling optimal beserta titik posisinya juga.

Meskipun algoritma brute force ini dapat menyelesaikan Breach Protocol, perlu diingat bahwa ini mungkin bukan pendekatan yang paling efisien terutama jika ukurannya sangat besar.

Bab II

Implementasi Algoritma dalam Bahasa Python

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman python. Struktur dari program ini dibuat dalam 1 file yang utuh yaitu main.py.

2.1 File main.py

File ini berisikan fungsi-fungsi yang melakukan perhitungan pencarian kemungkinan solusi dan mengambil solusi yang paling optimal sekaligus sebagai *driver* utama dari program ini.

2.1.1 Fungsi-fungsi

Methods	Description
`main_menu()`	Fungsi ini mencetak menu utama aplikasi dan meminta pengguna untuk memilih opsi (input berupa angka). Kemudian, memastikan opsi yang dipilih valid dan mengembalikan opsi yang dipilih.
`startscreen()`	Fungsi ini mencetak tampilan awal program yang berupa teks ASCII.
`gun()`	Fungsi ini mencetak senjata ASCII art.
`generate_random_matrix(kolom_matriks, baris_matriks)`	Fungsi ini menghasilkan matriks acak dengan ukuran yang ditentukan (kolom dan baris) dan mencetaknya dalam bentuk yang diwarnai.
`generate_random_sequences(jumlah_sequence, ukuran_maksimal_sequence)`	Fungsi ini menghasilkan urutan karakter acak (sequence) sebanyak `jumlah_sequence`, masing-masing dengan panjang antara 2 hingga `ukuran_maksimal_sequence`, serta memberikan reward acak untuk setiap urutan tersebut. Fungsi mengembalikan daftar sequence dan daftar reward.
`sequenceInPath(sequences, currentPath)`	Fungsi ini memeriksa apakah urutan karakter (sequence) ada di dalam jalur yang diberikan (currentPath). Mengembalikan `True` jika ditemukan, `False` sebaliknya.
`possibilities(current_buffer, index, currentPath, currentPosition,`	Fungsi ini merupakan bagian dari algoritma rekursif untuk mengeksplorasi kemungkinan path optimal. Fungsi ini mengeksplorasi semua kemungkinan path yang mungkin dalam matriks dengan kriteria tertentu dan mengembalikan

Methods	Description
`seenPath, ukuran_buffer, paths, matrix_size, matrix, position, sequences)`	path-path yang valid dan posisi-path.
`getReward(paths, sequences, rewards, listOfReward)`	Fungsi ini menghitung total reward untuk setiap path yang ditemukan. Mengembalikan daftar reward.
`getOptimal(listOfReward, paths)`	Fungsi ini mencari path dengan total reward maksimum dan mengembalikan nilai reward maksimum dan indeks path yang sesuai.
`save_solution(sequences, rewards, maximal, outputBuffer, outputposition, execution_time, matrix, filename)`	Fungsi ini menyimpan solusi ke dalam file teks. Mencetak matriks, sequence, reward, path optimal, dan waktu eksekusi. Menyimpan solusi ke dalam file .txt.

2.1.2 Variabel-variabel

Variable Name	Description
`option`	Menyimpan opsi yang dipilih pengguna.
`token`	Menyimpan token unik yang dimasukkan user
`sequences`	List sequence yang digunakan.
`rewards`	List reward untuk setiap sequence.
`listOfReward`	List total reward untuk setiap path.
`matrix`	Matriks yang tersusun atas `kolom_matriks` dan `baris_matriks`
`paths`	List path yang mungkin.
`position`	List posisi dari path-path yang mungkin.

Variable Name	Description
`inputBuffer`	Buffer input yang terbentuk dari path optimal.
`outputBuffer`	Buffer output untuk ditampilkan ke pengguna.
`outputposition`	Posisi path optimal.
`execution_time`	Waktu eksekusi program.

2.2 Library

Terdapat juga beberapa library yang digunakan untuk program ini, antara lain :

- `Import random`
- `Import time`
- `Import os`
- `From termcolor import colored`

Bab III

Implementasi Algoritma dalam Bahasa Python

3.1 Repository Program

Repository program dapat diakses melalui tautan GitHub berikut :
https://github.com/zultopia/Tucil-1-Stima.git

3.2 Source Code Program

```
import random
import time
import os
from termcolor import colored

def main_menu():
   print(colored('>>>>>>>=====[ MENU UTAMA ]=====<<<<<<<', 'light_yellow', 'on_white',
['bold']))
   print(colored("1. Masukan via CLI", "red"))
   print(colored("2. Masukan via file (*.txt)", "green"))
   print(colored("3. Exit", "blue"))

   option = int(input("Pilihan (1/2/3): "))
   while option > 3 or option < 1:
       print(colored("Pilihan tersebut tidak ada, pastikan memilih nomor yang
tersedia", 'red'))
       option = int(input("Pilihan (1/2/3): "))
   return option

def startscreen():
   ascii_text = '''
   _____           __                                  __          ___    ___  ____ ____
  / ___/  __ __   / /  ___   ____   ___  __ __ ___    / /__       |_  |  / _ \/ _ \/ _ /
 / /__   / // / / _ \/ -_) / __/  / _ \/ // / / _ \ /  '_/      / __/ / // / / / /  / /
 \___/   \_, / /_.__/\__/ /_/    / .__/\_,_/ /_//_//_/\_\      /____/ \___/ /_/  /_/
    ___ /___/                   /__          ___                __                    __
   / _ )  ____ ___  ___ _ ____  / /         / _ \  ____ ___  / /_ ___  ____ ___    / /
  / _  | / __// -_)/ _ `// __/ / _ \       / ___/ / __// _ \/ __// _ \/ __// _ \ / /
 /____/ /_/   \__/ \_,_/ \__/ /_//_/      /_/    /_/   \___/\__/ \___/\__/ \___//_/


   '''
```

```
    colored_ascii_text = colored(ascii_text, color='yellow')
    print(colored_ascii_text)

def gun():
    ascii_text = '''
    +--^----------,--------,-----,--------^-,
     | |||||||||   `--------'     |          O
     `+---------------------------^----------|
       `\_,---------,---------,--------------'
         / XXXXXX /'|       /'
        / XXXXXX /  `\    /'
       / XXXXXX /`-------'
      / XXXXXX /
     / XXXXXX /
    (________(                by
     `------'      Marzuli Suhada M - 13522070
    '''

    colored_ascii_text = colored(ascii_text, color='yellow')
    print(colored_ascii_text)

def generate_random_matrix(kolom_matriks, baris_matriks):
    matrix = []
    for i in range(baris_matriks):
        row = [random.choice(token) for _ in range(kolom_matriks)]
        matrix.append(row)
        colored_row = [colored(symbol, 'yellow') for symbol in row]
        print(" ".join(colored_row))
    return matrix

def generate_random_sequences(jumlah_sequence, ukuran_maksimal_sequence):
    sequences = []
    rewards = []
    for i in range(jumlah_sequence):
        sequence_size = random.randint(2, ukuran_maksimal_sequence)
        reward = random.randint(-100, 100)
        sequence = "".join(random.choice(token) for _ in range(sequence_size))
        sequences.append(sequence)
        rewards.append(reward)
    return sequences, rewards

def sequenceInPath(sequences, currentPath):
```

```
    return currentPath in sequences

def possibilities(current_buffer, index, currentPath, currentPosition, seenPath,
ukuran_buffer, paths, matrix_size, matrix, position, sequences):
    if current_buffer == ukuran_buffer or sequenceInPath(sequences, currentPath):
        paths.append(currentPath)
        position.append(currentPosition)
        return None
    if current_buffer == 0:
        for i in range(matrix_size[0]):
            possibilities(1, i, matrix[0][i], currentPosition + ([i + 1, 1]), [[i, 0]],
ukuran_buffer, paths, matrix_size,
                          matrix, position, sequences)
    elif current_buffer % 2 == 1:
        for j in range(matrix_size[1]):
            seen = False
            for positions in seenPath:
                if positions[0] == index and positions[1] == j:
                    seen = True
                    break
            if seen:
                continue
            seenPath.append([index, j])
            possibilities(current_buffer + 1, j, currentPath + matrix[j][index],
currentPosition + ([index + 1, j + 1]),
                          seenPath, ukuran_buffer, paths, matrix_size, matrix,
position, sequences)
            seenPath.pop()
    else:
        for i in range(matrix_size[0]):
            seen = False
            for positions in seenPath:
                if positions[1] == index and positions[0] == i:
                    seen = True
                    break
            if seen:
                continue
            seenPath.append([i, index])
            possibilities(current_buffer + 1, i, currentPath + matrix[index][i],
currentPosition + ([i + 1, index + 1]),
                          seenPath, ukuran_buffer, paths, matrix_size, matrix,
position, sequences)
```

```
            seenPath.pop()
    return paths, position

def getReward(paths, sequences, rewards, listOfReward):
    for path in paths:
        reward = 0
        for i in range(len(sequences)):
            if (sequences[i] in path):
                reward += rewards[i]
        listOfReward.append(reward)
    return listOfReward

def getOptimal(listOfReward, paths):
    maximal = 0
    imax = 0
    for i in range(len(paths)):
        if (listOfReward[i] > maximal):
            maximal = listOfReward[i]
            imax = i
        elif listOfReward[i] == maximal:
            if len(paths[i]) < len(paths[imax]):
                maximal = listOfReward[i]
                imax = i
    return maximal, imax

def save_solution(sequences, rewards, maximal, outputBuffer, outputposition,
execution_time, matrix, filename):
    folderpath =
"/Users/azulsuhada/Documents/Semester4/Stima/Tucil-1-Stima/test/output"
    filepath = os.path.join(folderpath, filename + ".txt")
    # Mengecek apakah file sudah ada
    while (os.path.exists(filepath)):
        print(colored(f"File dengan nama '{filename}.txt' sudah ada. Silakan pilih nama
yang lain.", 'red'))
        filename = input("Masukkan nama file untuk menyimpan solusi: ")
        folderpath =
"/Users/azulsuhada/Documents/Semester4/Stima/Tucil-1-Stima/test/output"
        filepath = os.path.join(folderpath, filename + ".txt")
    with open(filepath, "w") as file:
        if (option == 1):
            for row in matrix:
                file.write(" ".join(row) + "\n")
```

```
            for i in range(len(sequences)):
                outputSequence = ""
                inputSequence = sequences[i]
                for j in range(0, len(sequences[i]), 2):
                    outputSequence += inputSequence[j:j+2] + " "
                outputSequence = outputSequence.rstrip()
                file.write(outputSequence + "\n")
                file.write(str(rewards[i]) + "\n")
            file.write("\n")
        file.write(str(maximal) + "\n")
        if (maximal > 0) :
            file.write(outputBuffer + "\n")
            for i in range(0, len(outputposition), 2):
                file.write("{}, {}\n".format(outputposition[i], outputposition[i + 1]))
            print("\n")
        file.write("\n{} ms".format(execution_time))

if __name__ == "__main__":
    run = True
    startscreen()
    gun()
    while run:
        option = main_menu()
        token = []
        sequences = []
        rewards = []
        listOfReward = []

        if option == 1:
            valid = False
            valid2 = False
            while not valid:
                valid = True
                jumlah_token_unik = int(input("Masukkan jumlah token unik: "))
                while not valid2:
                    inputTokens = input("Masukkan token (pisahkan dengan spasi):
").split()
                    token_set = set()
                    for token in inputTokens:
                        if len(token) == 2 and token.isalnum() and token not in
token_set:
                            token_set.add(token)
```

```
                        valid2 = True
                    else:
                        print("Token harus terdiri dari dua karakter alfanumerik
dan bersifat unik.")
                        valid2 = False
                        break
            token = list(token_set)
            ukuran_buffer = int(input("Masukkan ukuran buffer: "))
            ukuran_matriks = input("Masukkan ukuran matriks (kolom baris): ")
            kolom_matriks, baris_matriks = map(int, ukuran_matriks.split())
            jumlah_sequence = int(input("Masukkan jumlah sequence: "))
            ukuran_maksimal_sequence = int(input("Masukkan ukuran maksimal
sequence: "))

            print(colored("\nMatrix:", "light_red"))
            matrix = generate_random_matrix(kolom_matriks, baris_matriks)
            print(colored("\nSequences & Reward:", 'light_red'))
            sequences, rewards = generate_random_sequences(jumlah_sequence,
ukuran_maksimal_sequence)
            for i in range(0, jumlah_sequence):
                outputSequence = ""
                inputSequence = sequences[i]
                for j in range(0, len(sequences[i]), 2):
                    outputSequence += colored(inputSequence[j:j+2], 'green') + " "
                outputSequence = outputSequence.rstrip()
                print(outputSequence)
                colored_reward = colored(str(rewards[i]), 'blue')
                print(rewards[i])
            start_time = time.time()
            paths, position = possibilities(0, 0, '', [], [], ukuran_buffer, [],
[kolom_matriks, baris_matriks], matrix, [], sequences)
            listOfReward = getReward(paths, sequences, rewards, listOfReward)
            maximal, imax = getOptimal(listOfReward, paths)
            end_time = time.time()
            execution_time = (end_time - start_time) * 1000
            print(colored("\nMaximal Reward:", 'light_red'))
            print(maximal)
            inputBuffer = paths[imax]
            outputBuffer = ""
            outputposition = position[imax]
            if maximal == 0 :
```

```
                    print(colored("\nTidak ada optimal path yang memenuhi",
'light_red'))
                else :
                    print(colored("\nOptimal Path:", 'light_red'))
                    for i in range (0, len(paths[imax]), 2):
                        outputBuffer += inputBuffer[i:i+2] + " "
                    outputBuffer = outputBuffer.rstrip()
                    print(outputBuffer)
                    print(colored("\nOptimal Path position:", 'light_red'))
                    for i in range(0, len(position[imax]), 2):
                        print("{}, {}".format(outputposition[i], outputposition[i +
1]))
                print(colored("\nExecution Time:", 'light_red'))
                print(execution_time, "ms")

                # Pilihan apakah ingin menyimpan ke dalam file .txt
                validName = False
                while (validName == False):
                    answer = input("Apakah ingin menyimpan solusi? (y/n): ")
                    if answer.lower() == 'y':
                        validName = True
                        filename = input("Masukkan nama file untuk menyimpan solusi: ")
                        save_solution(sequences, rewards, maximal, outputBuffer,
outputposition, execution_time, matrix, filename)
                        print("Solusi telah disimpan dalam file", filename + ".txt\n")
                    elif answer.lower() == 'n':
                        validName = True
                        print("Kembali ke menu utama program.\n")
                    else:
                        validName = False
                        print("Masukan tidak valid. Silakan masukkan 'y' atau 'n'.")

        elif option == 2:
            valid = False
            validSequence = True
            while not valid:
                valid = True
                fileName = input("Masukkan nama file yang akan diproses: ")
                try:
                    with
open(f"/Users/azulsuhada/Documents/Semester4/Stima/Tucil-1-Stima/test/input/{fileName}
.txt", 'r') as file:
```

```
                        lines = file.readlines()
                        ukuran_buffer = int(lines[0])
                        kolom_matriks, baris_matriks = map(int, lines[1].split())
                        matrix = []
                        for i in range(baris_matriks):
                            matrix.append(lines[i+2].split())
                        jumlah_sequence = int(lines[2 + baris_matriks])
                        # Pengecekan apakah matriks yang dibaca dari file sesuai dengan
kolom dan baris yang diharapkan
                        if len(matrix) != baris_matriks or any(len(row) !=
kolom_matriks for row in matrix):
                            print(colored("Matriks yang dibaca tidak sesuai dengan
kolom dan baris yang tertulis pada file!\n", 'red'))
                            break

                        sequence_reward_map = {}
                        for i in range(3 + baris_matriks, len(lines), 2):
                            sequence = (lines[i].replace(' ', '')).rstrip()
                            reward = int(lines[i + 1])
                            if sequence in sequence_reward_map:
                            # Jika sequence sudah ada dalam map, periksa apakah reward
sama
                                if sequence_reward_map[sequence] != reward:
                                    print(colored("Ada sequence yang sama namun
rewardnya berbeda.\n", 'red'))
                                    validSequence = False
                            else:
                            # Jika sequence belum ada dalam map, kita tambahkan
sequence dan reward ke dalam map
                                sequence_reward_map[sequence] = reward

                            sequences.append(sequence)
                            rewards.append(reward)
                        # Pengecekan apakah jumlah sequence yang dibaca dari file
sesuai dengan yang tertulis di file
                        if validSequence == False:
                            break
                        if len(sequences) != jumlah_sequence:
                            print(colored("Jumlah sequence yang dibaca tidak sesuai
dengan yang tertulis di file!\n", 'red'))
                            break
```

```
            except FileNotFoundError:
                valid = False
                print("File tidak ditemukan. Silahkan coba kembali.")
                print("================================================")

            if valid:
                start_time = time.time()
                paths, position = possibilities(0, 0, '', [], [], ukuran_buffer,
[], [kolom_matriks, baris_matriks], matrix, [], sequences)
                listOfReward = getReward(paths, sequences, rewards, listOfReward)
                maximal, imax = getOptimal(listOfReward, paths)
                end_time = time.time()
                execution_time = (end_time - start_time) * 1000
                for i in range(0, jumlah_sequence):
                    outputSequence = ""
                    inputSequence = sequences[i]
                    for j in range(0, len(sequences[i]), 2):
                        outputSequence += colored(inputSequence[j:j+2], 'green') +
" "
                    outputSequence = outputSequence.rstrip()
                    print(outputSequence)
                    colored_reward = colored(str(rewards[i]), 'blue')
                    print(rewards[i])
                print(colored("\nMaximal Reward:", 'light_red'))
                print(maximal)
                inputBuffer = paths[imax]
                outputBuffer = ""
                outputposition = position[imax]
                if maximal == 0 :
                    print(colored("\nTidak ada optimal path yang memenuhi",
'light_red'))
                else :
                    print(colored("\nOptimal Path:", 'light_red'))
                    for i in range (0, len(paths[imax]), 2):
                        outputBuffer += inputBuffer[i:i+2] + " "
                    outputBuffer = outputBuffer.rstrip()
                    print(outputBuffer)
                    print(colored("\nOptimal Path position:", 'light_red'))
                    for i in range(0, len(position[imax]), 2):
                        print("{}, {}".format(outputposition[i], outputposition[i +
1]))
                print(colored("\nExecution Time:", 'light_red'))
```

```
                    print(execution_time, "ms")

                    # Pilihan apakah ingin menyimpan ke dalam file .txt
                    validName = False
                    while (validName == False):
                        answer = input("Apakah ingin menyimpan solusi? (y/n): ")
                        if answer.lower() == 'y':
                            validName = True
                            filename = input("Masukkan nama file untuk menyimpan
solusi: ")
                            save_solution(sequences, rewards, maximal, outputBuffer,
outputposition, execution_time, matrix, filename)
                            print("Solusi telah disimpan dalam file", filename +
".txt\n")
                        elif answer.lower() == 'n':
                            validName = True
                            print("Kembali ke menu utama program.\n")
                        else:
                            validName = False
                            print("Masukan tidak valid. Silakan masukkan 'y' atau
'n'.")

        elif option == 3:
            run = False
            print(colored("Thank you for using this program!", 'yellow'))
            print(colored("You're ready for the next Martinez!", 'yellow'))
            input(colored("Press enter to exit... and byeee 👋", 'yellow'))

        valid = False
```

3.3 Kompleksitas Waktu (Big-O Notation)

Kompleksitas waktu (notasi Big O) dari algoritma yang digunakan dapat ditentukan dengan menganalisis bagian-bagian utamanya berikut:

- **Generating Random Matrix and Sequences:** Untuk menghasilkan matriks dengan ukuran $n \times m$, kita harus membuat n baris dan di setiap baris, kita harus membuat m elemen. Oleh karena itu, kompleksitas waktu untuk menghasilkan matriks adalah O(n * m).
- **Finding All Possible Paths:** Pada setiap langkah, fungsi `possibilities()` secara rekursif mempertimbangkan setiap sel dalam matriks sebagai titik awal dan menjelajahi matriks menggunakan pendekatan DFS (Depth-First Search). Meskipun ini tampaknya seperti perulangan melalui seluruh matriks, namun fungsi `possibilities()` tidak

benar-benar menelusuri seluruh matriks pada setiap langkah rekursifnya. Ini mengikuti jalur yang mungkin dari setiap sel yang dipertimbangkan. Oleh karena itu, kompleksitas waktu untuk mencari semua jalur yang mungkin sebenarnya tidak pasti, tetapi dapat disimpulkan sebagai eksponensial dalam kasus terburuknya.

- **Calculating Rewards and Finding Optimal Path:** Setelah semua jalur yang mungkin ditemukan, perlu dilakukan iterasi melalui setiap jalur untuk menghitung total reward yang diperoleh. Meskipun iterasi ini dilakukan melalui setiap jalur, jumlah jalur yang ditemukan akan mempengaruhi kompleksitas waktu akhirnya. Kemudian, mencari jalur optimal melibatkan memeriksa semua hadiah dan memilih yang maksimal. Kompleksitas ini bergantung pada jumlah jalur yang ditemukan dan panjang jalur optimal.

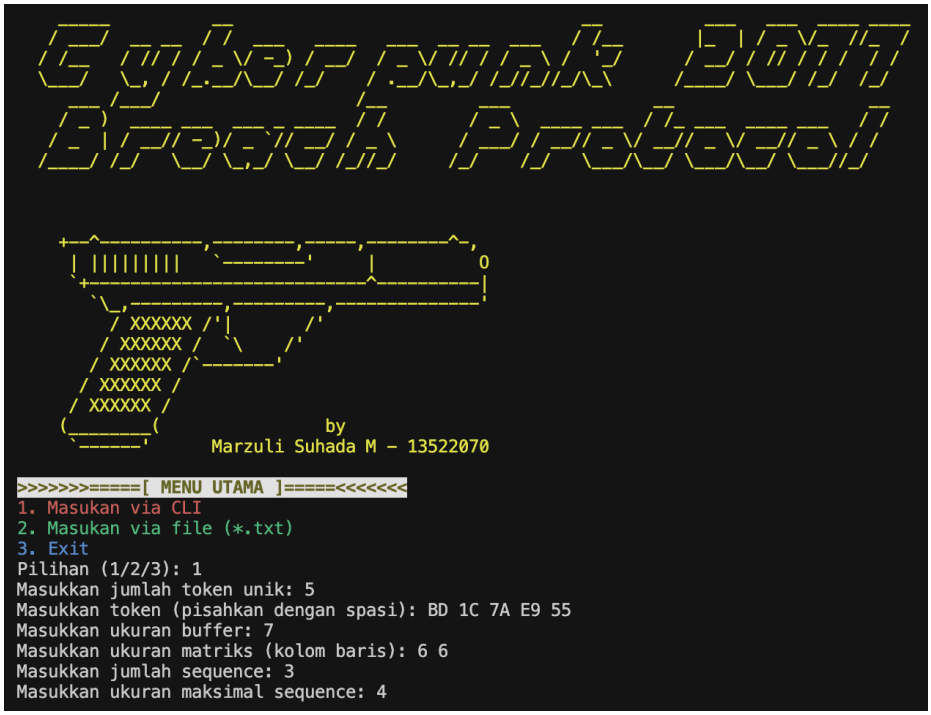
Jadi, sementara kompleksitas waktu untuk menghasilkan matriks adalah O(n * m), kompleksitas pencarian jalur dan perhitungan hadiah serta menemukan jalur optimal tidak sebanding dengan O(n * m). Namun satu hal yang pasti, kompleksitas pencarian jalur kemungkinan lebih dekat dengan eksponensial dalam kasus terburuk. Oleh karena itu, kompleksitas waktu keseluruhan bergantung pada kompleksitas pencarian jalur, yang mungkin jauh lebih tinggi daripada O(n * m).

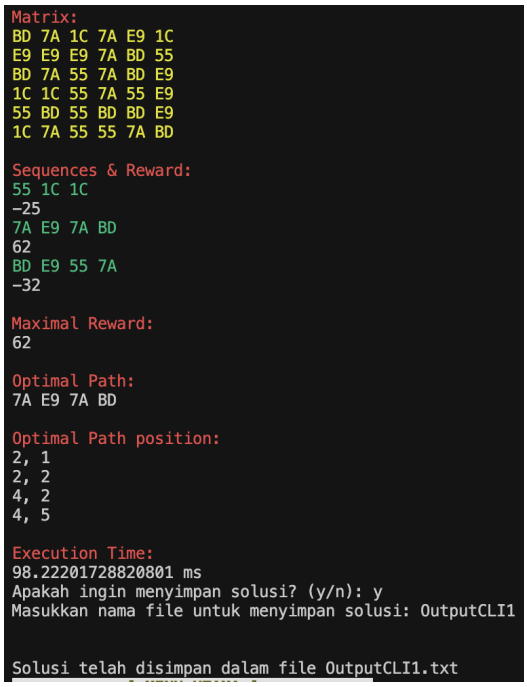
Bab IV

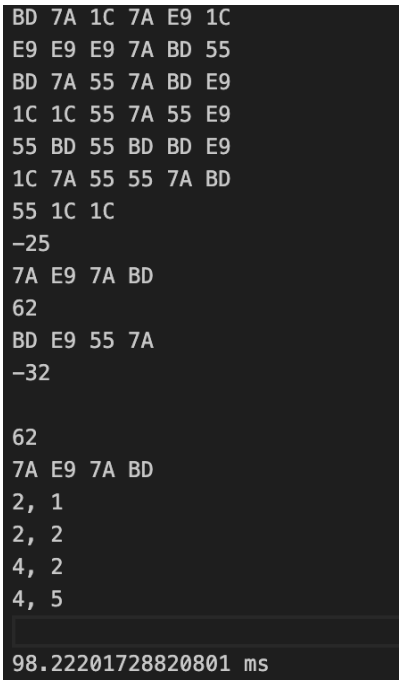
Masukan dan Luaran Program

4.1 Input melalui CLI

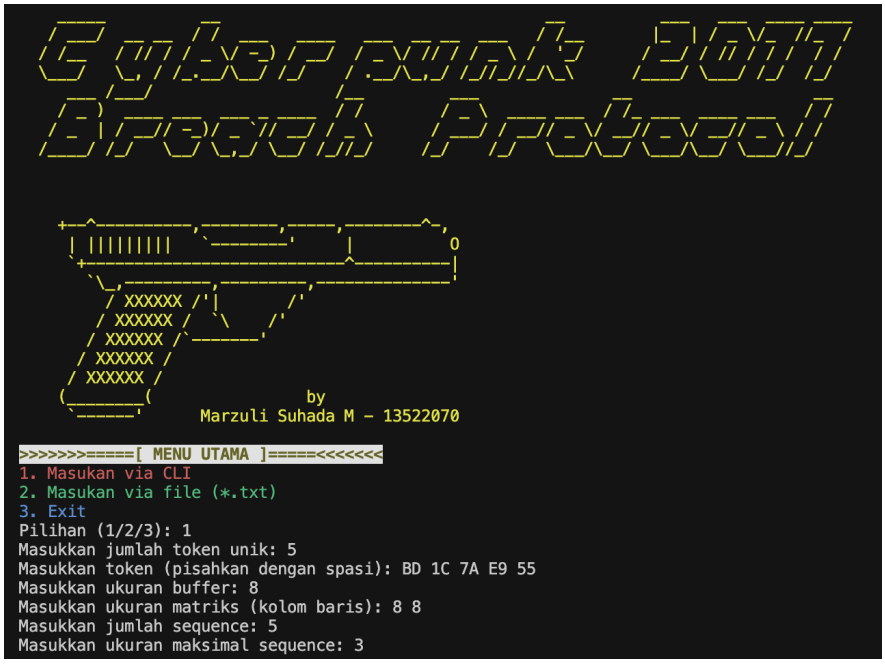
4.2.1 InputCLI1

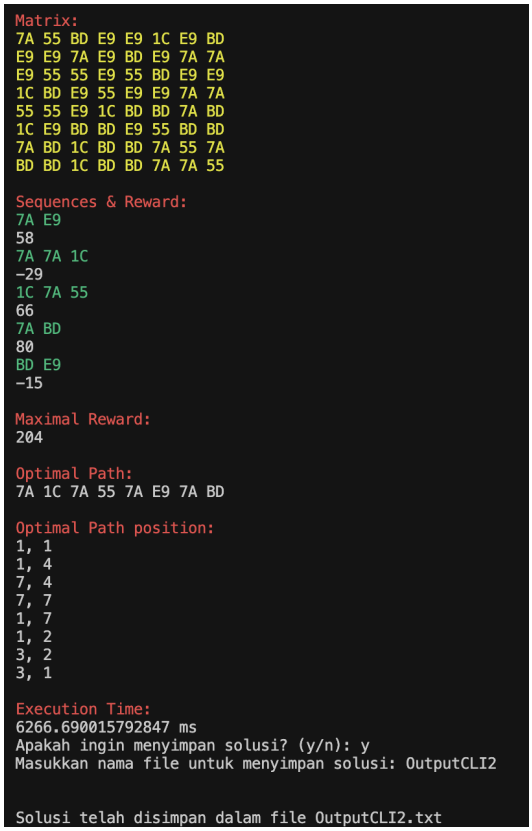


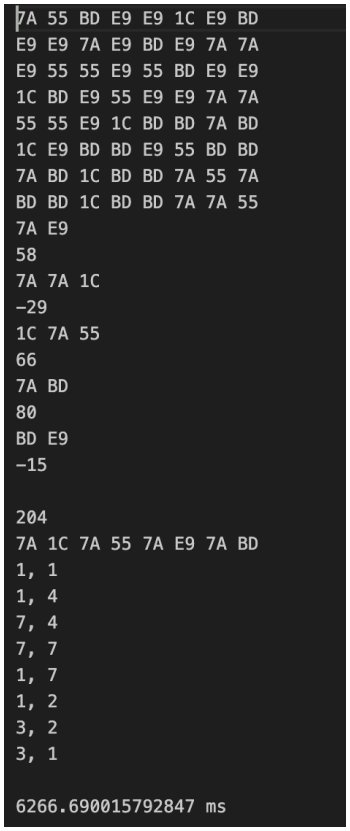




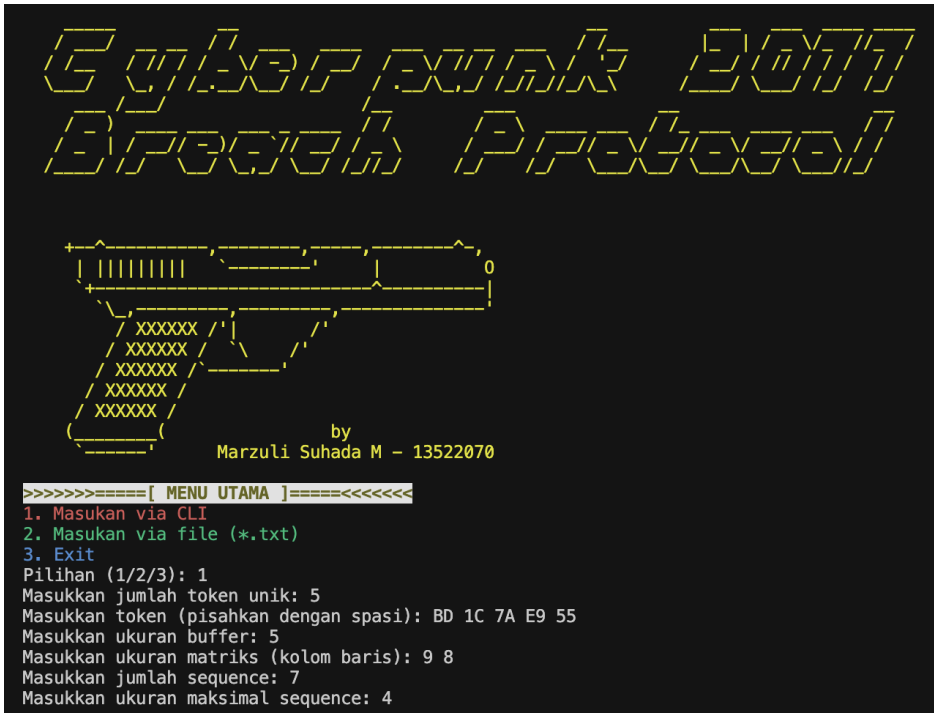
4.2.1 InputCLI2

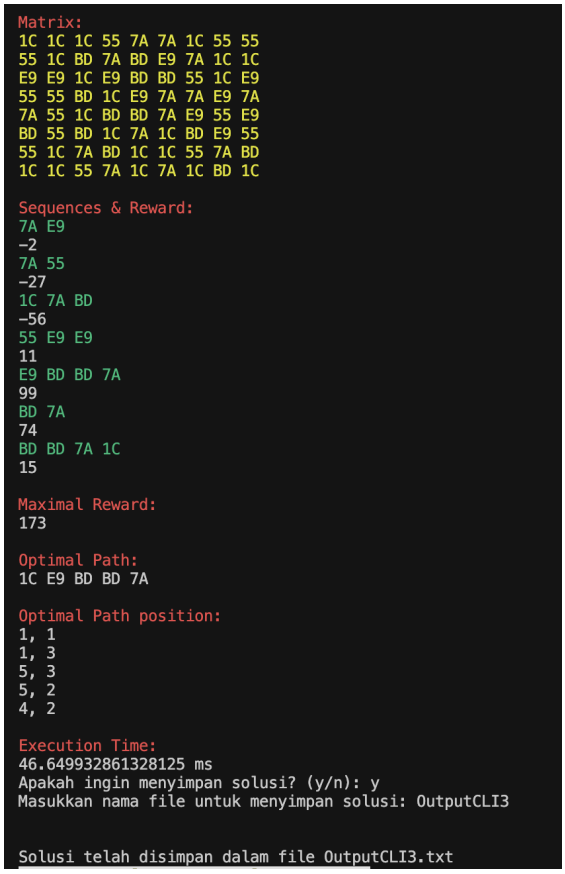


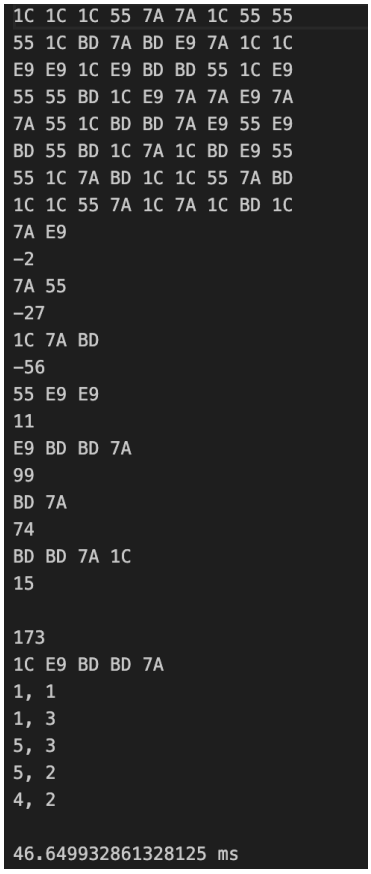




4.2.1 InputCLI3

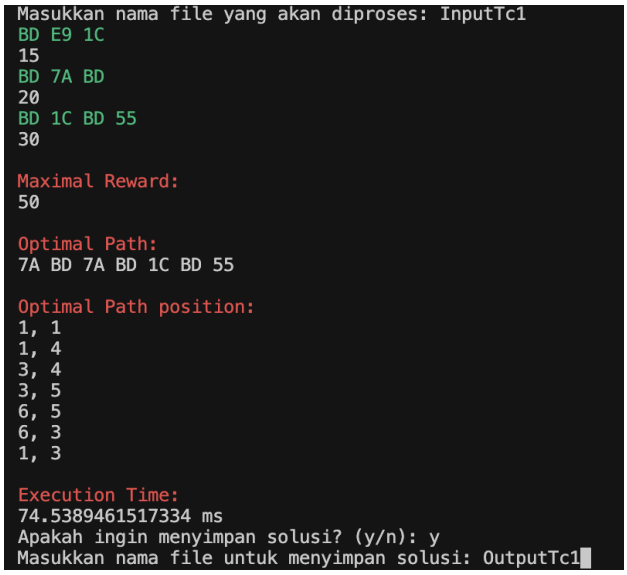


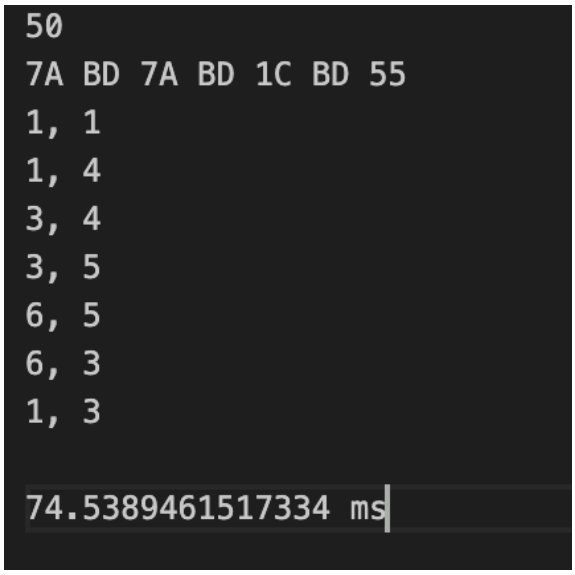




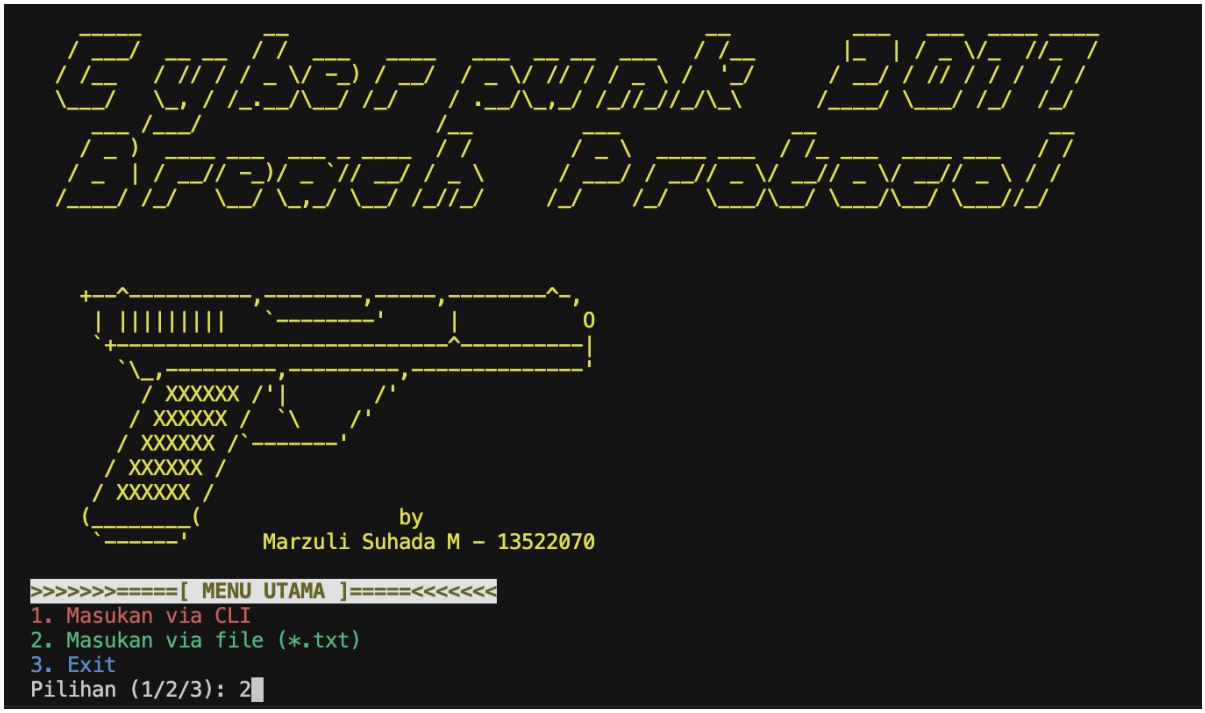
4.2 Input melalui .txt

4.2.1 InputTc1





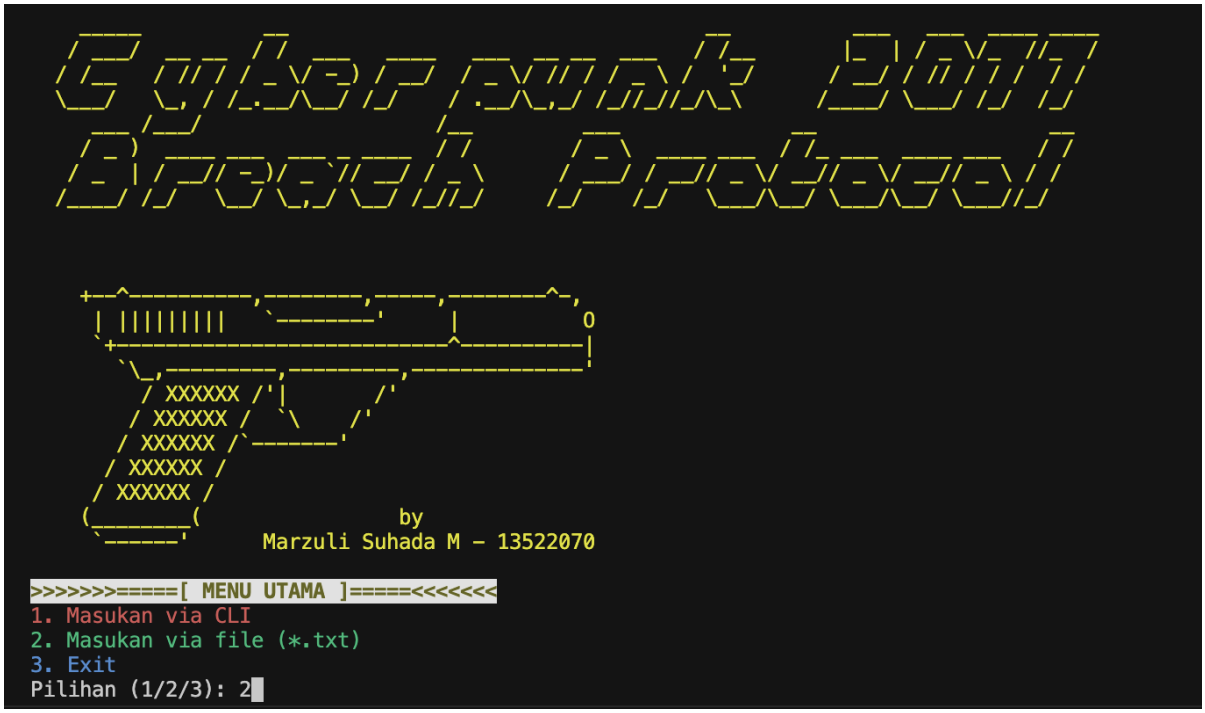
4.2.2 InputTc2

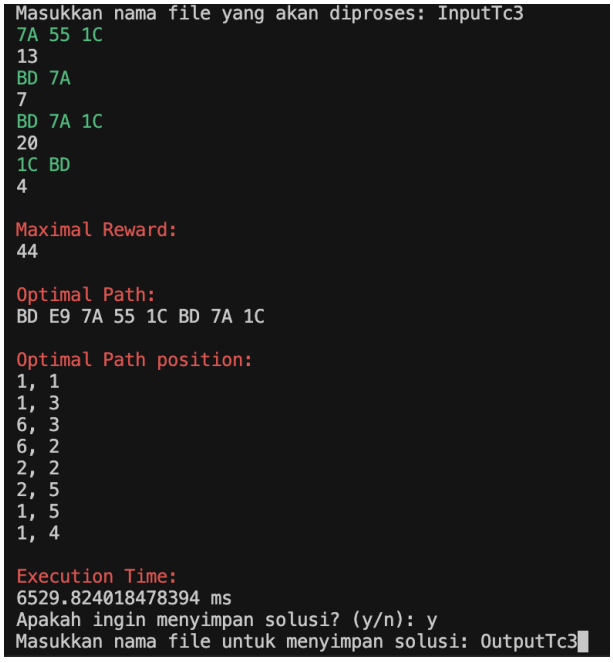


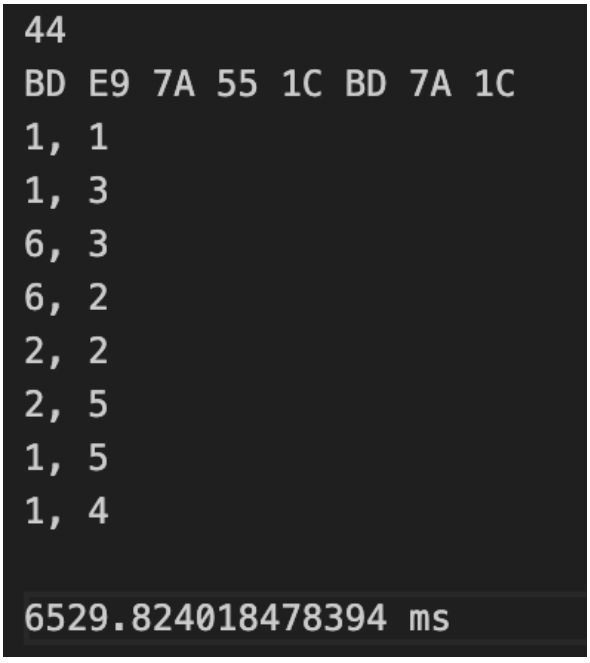

```
Masukkan nama file yang akan diproses: InputTc2
7A 55 1C
20
55 1C 7A
30
7A 1C
50

Maximal Reward:
50

Optimal Path:
7A 1C

Optimal Path position:
1, 1
1, 2

Execution Time:
0.2923011779785156 ms
Apakah ingin menyimpan solusi? (y/n): y
Masukkan nama file untuk menyimpan solusi: OutputTc2
```

```
50
7A 1C
1, 1
1, 2

0.2923011779785156 ms
```

4.2.3 InputTc3







Bab V

Lampiran

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

REFERENSI

Rinaldi Munir. "Aljabar dan Geometri untuk Informatika 2023/2024." https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

“Cyberpunk Quickhacking”, Cyberpunk Wiki https://cyberpunk.fandom.com/wiki/Quickhacking

All possible Paths To reach bottom of Matrix https://m.youtube.com/watch?v=2DKjP0Mjxyw