LAPORAN TUGAS KECIL

IF2211 STRATEGI ALGORITMA

SEMESTER II 2022-2023

PENYELESAIAN *CYBERPUNK 2077 BREACH PROTOCOL*

DENGAN **ALGORITMA *BRUTE FORCE***

Disusun oleh:

Attara Majesta Ayub 13522139

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

Algoritma Brute Force

Algoritma Brute Force diimplimentasikan melalui dua fungsi utama: `find_buffer` untuk inisialisasi variabel dan iterasi token pertama dalam buffer, serta `search_token` sebagai fungsi rekursif untuk menemukan token-token berikutnya. Algoritma pendukung yang diimplementasikan adalah **Knuth-Morris-Pratt Pattern Searching** untuk mencocokan buffer dengan sekuens.

Pertama, algoritma memulai dengan melakukan loop pada baris teratas matriks, terurut dari kiri ke kanan untuk menentukan token pertama dalam buffer. Setelah token pertama ditentukan, koordinatnya dinyatakan sebagai True dalam boolean visited untuk menghindari adanya *overlapping*. Dengan menggunakan operasi modulo, token diklasifikasikan menjadi ganjil dan genap. Token ganjil akan berarah vertikal dari token sebelumnya, sedangkan token genap berarah horizontal.

Langkah gerak token dibuat dengan loop, untuk menentukan ”berapa kali jalan” dari koordinat sebelumnya, memungkinkan algoritma untuk menjangkau seluruh token hingga yang terjauh. Fungsi kemudian melakukan pemanggilan rekursif untuk menemukan token berikutnya. Basis tercapai ketika jumlah token sama dengan panjang buffer yang diminta. Saat basis tercapai, algoritma menghitung reward melalui fungsi `calculate_reward`. Jika melebihi reward maksimum saat ini, nilai reward, koordinat, dan token diperbarui dalam variabel `max_reward` dan `max_path`.

Algoritma menggunakan beberapa library pendukung. Library Random digunakan untuk fitur pembuatan matriks dan sekuens secara acak. Kemudian, library NumPy berguna untuk mempercepat perhitungan dan pekerjaan teknikal seperti *handling matrix*. Algoritma menggunakan cache dalam beberapa fungsi, terutama KMP Searching (Knuth-Morris-Pratt Pattern Searching) untuk memoisasi hasil kombinasi dan juga dalam fungsi `calculate_reward` agar kombinasi yang berulang tidak dikalkulasi kembali.

Source Code

A. Program 'main.py':

Program berisi algoritma *brute-force* untuk Cyberpunk 777 – Beach Protocol Solver.

```
import time
import sys
import random
import numpy as np

def cli_input():
    while True:
        try:
```

```
            num_unique_tokens = int(input("\nJumlah token: "))
            tokens = input("Masukkan token (dipisahkan spasi): ").split()
            if len(tokens) < num_unique_tokens:
                raise ValueError("Tidak sesuai dengan jumlah token unik.")
            buffer_size = int(input("Masukkan ukuran buffer: "))
            matrix_size = tuple(map(int, input("Masukkan ukuran matriks (baris
kolom): ").split()))
            num_sequences = int(input("Masukkan jumlah sekuens: "))
            max_sequence_size = int(input("Masukkan ukuran maksimal sekuens:
"))

            matrix = np.random.choice(tokens, matrix_size)

            sequences = []
            for _ in range(num_sequences):
                sequence = ''.join(random.choices(tokens,
k=max_sequence_size))
                reward = random.choice([10, 20, 30])
                sequences.append((sequence, reward))

            return buffer_size, matrix, sequences

        except ValueError:
            print("Error: Input tidak sesuai. Proses input diulang... \n\n")

def file_input(filename):
    with open(filename, 'r') as file:
        buffer_size = int(file.readline())
        matrix_size = tuple(map(int, file.readline().split()))
        matrix = np.array([file.readline().split() for _ in
range(matrix_size[0])])  # Menggunakan np.array
        num_of_sequences = int(file.readline())
        sequences = []
        for _ in range(num_of_sequences):
            sequence = ''.join(file.readline().split())
            reward = int(file.readline())
            sequences.append((sequence, reward))
    return buffer_size, matrix, sequences

def KMPSearch(pat, txt, lps_cache):
    if (pat, txt) in lps_cache:
        return lps_cache[(pat, txt)]

    M = len(pat)
    N = len(txt)
    lps = [0] * M
    j = 0
    found = 0
```

```
    length = 0
    lps[0] = 0
    i = 1

    while i < M:
        if pat[i] == pat[length]:
            length += 1
            lps[i] = length
            i += 1
        else:
            if length != 0:
                length = lps[length - 1]
            else:
                lps[i] = 0
                i += 1

    i = 0
    while (N - i) >= (M - j):
        if pat[j] == txt[i]:
            i += 1
            j += 1

        if j == M:
            found += 1
            j = lps[j - 1]

        elif i < N and pat[j] != txt[i]:
            if j != 0:
                j = lps[j - 1]
            else:
                i += 1

    lps_cache[(pat, txt)] = found
    return found


def calculate_reward(path, sequences, lps_cache):
    reward = 0
    path_str = ''.join(path)
    for sequence, reward_value in sequences:
        found = KMPSearch(sequence, path_str, lps_cache)
        if found != 0:
            reward += found * reward_value
    return reward

def search_token(matrix, sequences, x, y, visited, path, max_path, max_reward,
lps_cache):
    rows, cols = matrix.shape
```

```
    if x < 0 or x >= rows or y < 0 or y >= cols or visited[x][y]:
        return

    visited[x][y] = True
    path.append((x, y, matrix[x, y]))

    if len(path) == buffer_size:
        reward = calculate_reward([id for _, _, id in path], sequences,
lps_cache)
        if reward > max_reward[0]:
            max_reward[0] = reward
            max_path.clear()
            max_path.extend(path)
    else:
        if len(path) % 2 == 0:
            for j in range(-(rows - 1), rows - 1):
                if j != 0:
                    search_token(matrix, sequences, x, y + j, visited, path,
max_path, max_reward, lps_cache)
        elif len(path) % 2 == 1:
            for i in range(-(cols - 1), cols - 1):
                if i != 0:
                    search_token(matrix, sequences, x + i, y, visited, path,
max_path, max_reward, lps_cache)

    visited[x][y] = False
    path.pop()

def find_buffer(buffer_size, matrix, sequences, lps_cache):
    rows, cols = matrix.shape
    max_reward = [0]
    max_path = []
    visited = np.zeros_like(matrix, dtype=bool)

    for j in range(cols):
        search_token(matrix, sequences, 0, j, visited, [], max_path,
max_reward, lps_cache)

    return max_path

def save_all_paths(filename, max_path, execution_time, reward):
    with open(filename, 'w') as file:
        file.write(f'Reward: {reward}\n')
        file.write('Path: ' + ' -> '.join([f'{y + 1},{x + 1}' for x, y, _ in
max_path]) + '\n')
        file.write('IDs: ' + ' '.join([id for _, _, id in max_path]) + '\n\n')
        file.write(execution_time + '\n')
```

```python
if __name__ == "__main__":
    print("\nWELCOME!\n\nApakah Anda ingin menggunakan input dari file atau
generate input?")
    print("1. File\n2. Generate\n")
    choice = input("Pilihan: ")
    if choice == "1":
        buffer_size, matrix, sequences = file_input('input.txt')
    elif choice == "2":
        buffer_size, matrix, sequences = cli_input()
    else:
        print("Error: Pilihan tidak valid. Keluar... \n\n")
        sys.exit(1)

    lps_cache = {}
    start_time = time.time()
    max_path = find_buffer(buffer_size, matrix, sequences, lps_cache)
    reward = calculate_reward([id for _, _, id in max_path], sequences,
lps_cache)
    execution_time = f'{(time.time() - start_time) * 1000:.2f} ms'

    if choice == "2":
        print("\nMATRIX:\n")
        for i in range(matrix.shape[0]):
            for j in range(matrix.shape[1]):
                print(matrix[i, j], end=' ')
            print()
        print("\nSEQUENCES:\n")
        for seq, reward in sequences:
            print(seq)
            print(reward)

    if reward == 0:
        print("\nTidak ada path yang memenuhi syarat.")
    else:
        print(f'\n{reward}')
        print(' '.join([id for _, _, id in max_path]))
        print('\n'.join([f'{y + 1},{x + 1}' for x, y, _ in max_path]) + '\n')

    print(execution_time + '\n')
    save = input("Apakah Anda ingin menyimpan hasil ke file? (y/n): ")
    if save == "y":
        save_all_paths('output.txt', max_path, execution_time, reward)
        print("Berhasil disimpan ke 'output.txt'.\n\n")
    else:
        print("Keluar... \n\n")
        sys.exit(1)
```

Gui.py:

Program berisi kode GUI dengan library tkinter.

```
import time
import random
import numpy as np
from tkinter import *
from tkinter import messagebox
from tkinter.ttk import *
import main

# Auto fill matrix (rand)
def fill_matrix_randomly():
    tokens = ['BD', '1C', '7A', '55', 'E9']
    for i in range(rows):
        for j in range(cols):
            token = np.random.choice(tokens)
            text_var[i][j].set(token)

# Auto fill sequences (rand)
def fill_sequences():
    num_sequences = 3
    max_sequence_size = 3
    tokens = ['BD', '1C', '7A', '55', 'E9']

    additional_input.delete("1.0", END)

    for _ in range(num_sequences):
        sequence = ' '.join(random.choices(tokens, k=max_sequence_size))
        reward = random.choice([10, 20, 30])
        additional_input.insert(END, f"{sequence}\n{reward}\n")

# Solve path
def solve_path():
    buffer_size = int(buffsize_choosen.get())
    matrix = np.array([[text_var[i][j].get() for j in range(cols)] for i in
range(rows)])
    sequences_text = additional_input.get("1.0", END).strip().split("\n")
    sequences = []

    for i in range(0, len(sequences_text), 2):
        sequence = ''.join(sequences_text[i].split())
        reward = int(sequences_text[i+1])
        sequences.append((sequence, reward))

    lps_cache = {}
    start_time = time.time()
    max_path = main.find_buffer(buffer_size, matrix, sequences, lps_cache)
```

```
    reward = main.calculate_reward([id for _, _, id in max_path], sequences,
lps_cache)
    execution_time = f'{(time.time() - start_time) * 1000:.2f} ms'

    label_result = LabelFrame(root, text=f"Result (Execution Time:
{execution_time})")
    label_result.pack(expand='yes', fill='both', side='left')

    if reward > 0:
        max_path_str = '\n'.join([f'{y + 1},{x + 1}' for x, y, _ in max_path])
        ids_str = ' '.join([id for _, _, id in max_path])

        reward_label = Label(label_result, text=f"REWARD: {reward}",
font=('arial', 10))
        reward_label.grid(row=2, column=0, padx=10, pady=1, sticky='w')

        ids_label = Label(label_result, text=f"TOKENS: {ids_str}",
font=('arial', 10))
        ids_label.grid(row=1, column=0, padx=10, pady=1, sticky='w')

        path_label = Label(label_result, text=f"\nPATH:\n{max_path_str}",
font=('arial', 10))
        path_label.grid(row=0, column=0, padx=10, pady=1, sticky='w')

    else:
        path_label = Label(label_result, text="No path found", font=('arial',
10))
        path_label.grid(row=0, column=0, padx=10, pady=1, sticky='w')

    # Save result
    def save_result():
        result = f"Reward: {reward}\nTokens: {ids_str}\nPath:\n{max_path_str}"
        with open("result.txt", "w") as file:
            file.write(result)
        messagebox.showinfo("Save", "Result saved to 'result.txt'")

    # Save button
    save_btn = Button(label_main, text='Save', width=10, command=save_result)
    save_btn.grid(row=3, column=1, padx=10, pady=10, sticky='n')

# Master window
root = Tk()
root.geometry('800x510')

# Scroll bar
scrollbar = Scrollbar(root, orient=VERTICAL)
scrollbar.pack(side=RIGHT, fill=Y)
```

```
# About
def display_about():
    about_info = "A program designed to find the optimal solution for
Cyberpunk 2077's Breach Protocol mini-game. To generate new solution, exit and
rerun the program.\n\n Developed by Attara Majesta A. (13522139)"
    messagebox.showinfo("About", about_info)

# Menu (exit)
menu = Menu(root)
root.config(menu=menu)
filemenu = Menu(menu)
menu.add_cascade(label='File', menu=filemenu)
filemenu.add_command(label='Exit', command=root.quit)
helpmenu = Menu(menu)
menu.add_cascade(label='Help', menu=helpmenu)
helpmenu.add_command(label='About', command=display_about)

# Title
title = 'CYBERPUNK 777\nBeach Protocol Solver'
root.title(title)
messageVar = Message(root, text=title, font=('@Kozuka Mincho Pr6N L', 15,
'bold'))
messageVar.config(bg='lightgreen', fg='black', width=250, relief='raised',
justify='center')
messageVar.pack(pady=10)

# Select buffer size
label_buffer = LabelFrame(root, text="Buffer Size")
label_buffer.pack(expand='yes', fill='both')
label_buff_title = Label(label_buffer, text="Select the buffer size :",
font=("Times New Roman", 10))
label_buff_title.grid(column=0, row=5, padx=10, pady=25)

n = StringVar()
buffsize_choosen = Combobox(label_buffer, width=27, textvariable=n)
buffsize_choosen['values'] = ('3', '4', '5', '6', '7')
buffsize_choosen.grid(column=1, row=5)
buffsize_choosen.current(0)

# Main label for matrix and sequences
label_main = LabelFrame(root, text="Main Path")
label_main.pack(expand='yes', fill='both', side='right')

label_matrix = Label(label_main, text="CODE MATRIX", font=('arial', 10,
'bold'))
label_matrix.grid(row=0, column=0, padx=10, pady=10, sticky='w')

frame_matrix = Frame(label_main)
```

```
frame_matrix.grid(row=1, column=0, padx=10, pady=10, sticky='w')

rows, cols = (6, 6)
text_var = []

for i in range(rows):
    text_var.append([])
    for j in range(cols):
        text_var[i].append(StringVar())
        entry = Entry(frame_matrix, width=5, font=('Arial', 10),
textvariable=text_var[i][j])
        entry.grid(row=i, column=j, padx=3, pady=3)

# Random button
random_btn = Button(label_main, text='Random', width=10,
command=fill_matrix_randomly)
random_btn.grid(row=2, column=0, padx=10, pady=10)

label_additional = Label(label_main, text="SEQUENCES", font=('arial', 10,
'bold'))
label_additional.grid(row=0, column=1, padx=10, pady=10, sticky='w')

frame_additional = Frame(label_main)
frame_additional.grid(row=1, column=1, padx=10, pady=10, sticky='w')

additional_input = Text(frame_additional, width=30, height=10, font=('Arial',
10))
additional_input.grid(row=0, column=0, padx=3, pady=3)

scrollbar.config(command=additional_input.yview)
additional_input.config(yscrollcommand=scrollbar.set)

random_seq_btn = Button(label_main, text='Random', width=10,
command=fill_sequences)
random_seq_btn.grid(row=2, column=1, padx=10, pady=10)

# Solve button
solve_btn = Button(label_main, text='Solve', width=10, command=solve_path)
solve_btn.grid(row=3, column=0, padx=10, pady=10, sticky='n')

root.mainloop()
```

TESTCASE

Main.py:

1. Input data dari file .txt

```
WELCOME!

Apakah Anda ingin menggunakan input dari file atau generate input?
1. File
2. Generate

Pilihan: 1

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

480.62 ms

Apakah Anda ingin menyimpan hasil ke file? (y/n): n
Keluar...
```

2. Input data dari CLI dengan token BD 1C 7A 55 E9

```
WELCOME!

Apakah Anda ingin menggunakan input dari file atau generate input?
1. File
2. Generate

Pilihan: 2

Jumlah token: 5
Masukkan token (dipisahkan spasi): BD 1C 7A 55 E9
Masukkan ukuran buffer: 7
Masukkan ukuran matriks (baris kolom): 6 6
Masukkan jumlah sekuens: 3
Masukkan ukuran maksimal sekuens: 4

MATRIX:

1C E9 BD 55 1C BD
7A BD 1C 55 BD 7A
7A 55 1C 7A E9 7A
1C 1C 55 55 BD 7A
E9 1C 55 E9 55 55
1C E9 E9 7A 7A 7A

SEQUENCES:

1C55E9E9
20
1C7A557A
20
1C551CE9
30

30
```

```
30
1C 7A BD 1C 55 1C E9
1,1
1,2
2,2
2,4
3,4
3,3
5,3

389.04 ms

Apakah Anda ingin menyimpan hasil ke file? (y/n): y
Berhasil disimpan ke 'output.txt'.
```

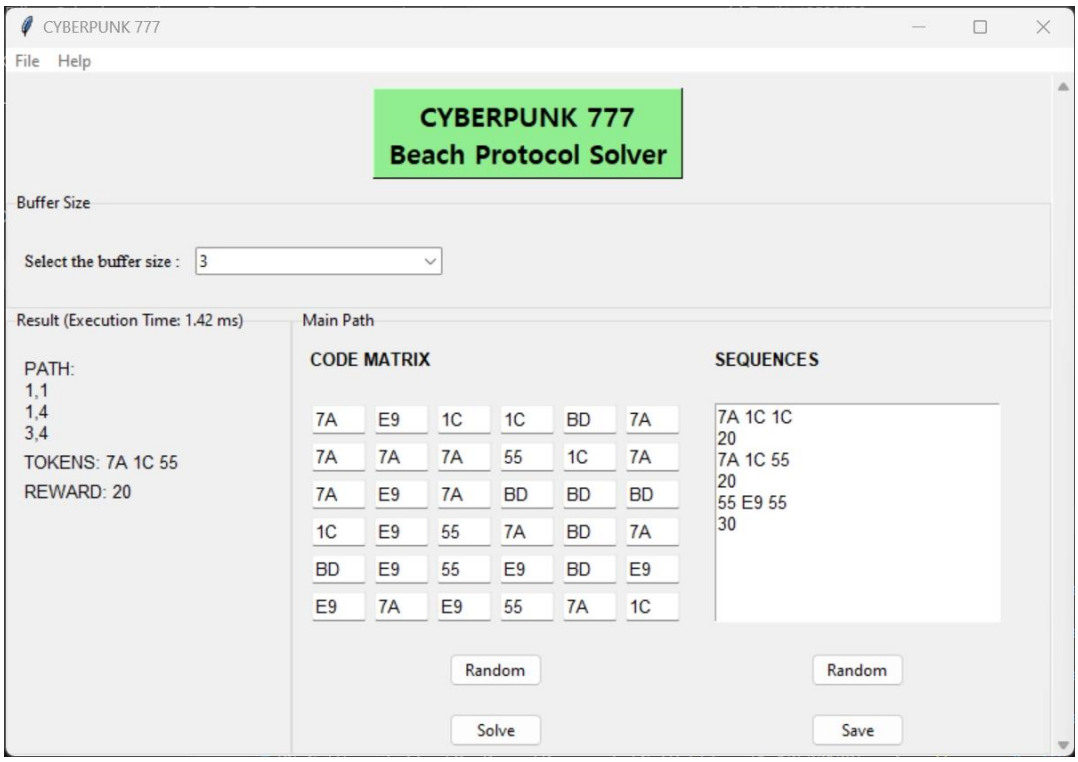
3. Input data dari CLI dengan token: A B C D E

```
WELCOME!

Apakah Anda ingin menggunakan input dari file atau generate input?
1. File
2. Generate

Pilihan: 2

Jumlah token: 5
Masukkan token (dipisahkan spasi): A B C D E
Masukkan ukuran buffer: 6
Masukkan ukuran matriks (baris kolom): 6 6
Masukkan jumlah sekuens: 3
Masukkan ukuran maksimal sekuens: 4

MATRIX:

C D B E B B
D C D A A E
E B B D B B
E C D C A C
D B A C D B
A C C D E C

SEQUENCES:

EAEC
20
AEEC
20
BBDD
20
```

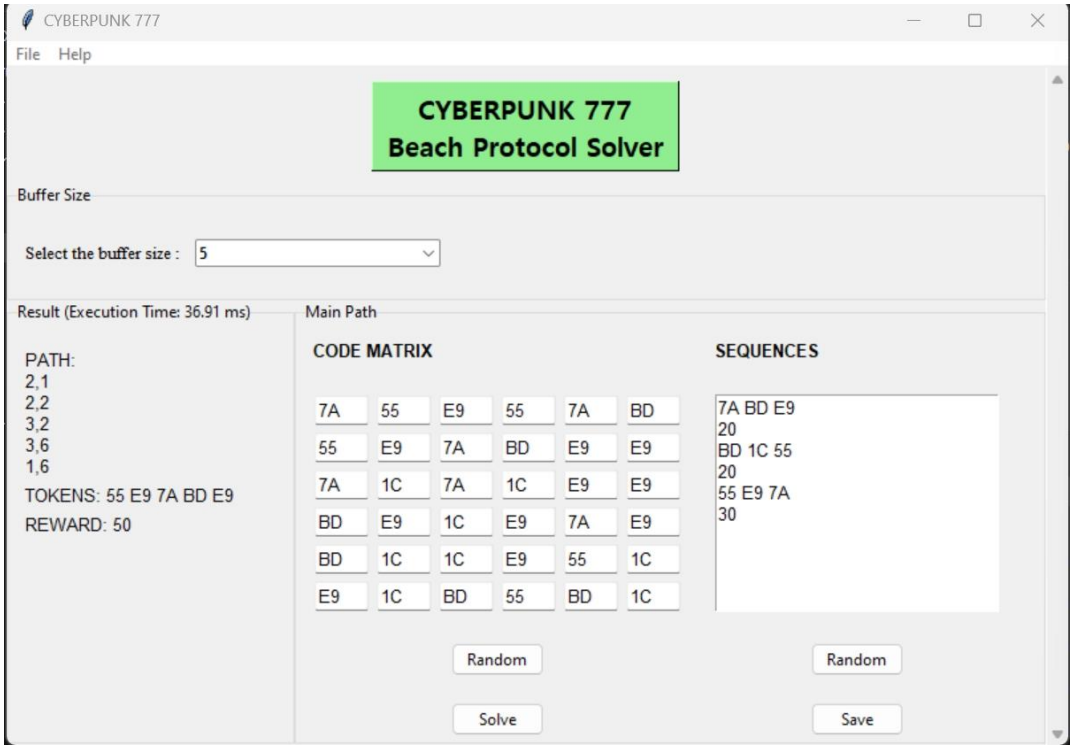
```
20
C E B B D D
1,1
1,3
2,3
2,5
1,5
1,2

128.09 ms

Apakah Anda ingin menyimpan hasil ke file? (y/n): n
Keluar...


PS C:\Users\attar\OneDrive\Documents\GitHub\Tucil1_13522139\src>
```

Gui.py:

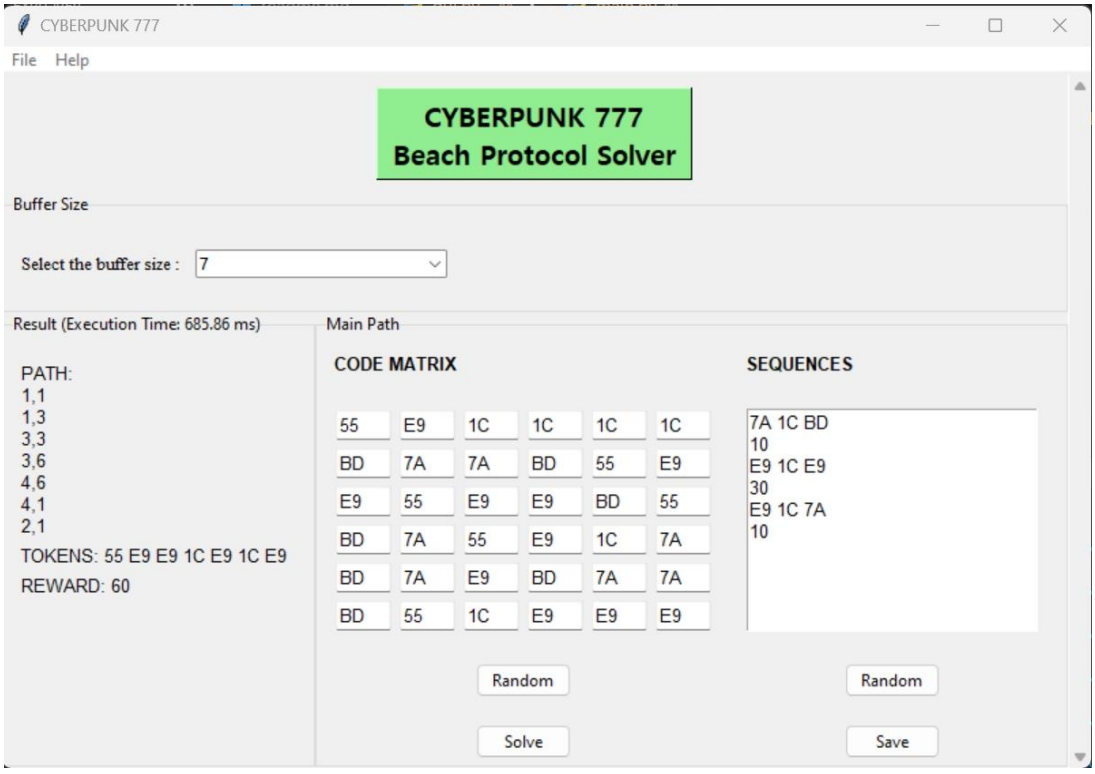
1. Input dengan buffer size = 3



2. Input dengan buffer size = 5



3. Input dengan buffer size = 7



REPOSITORY

https://github.com/attaramajesta/Tucil1_13522139.git

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	☺	