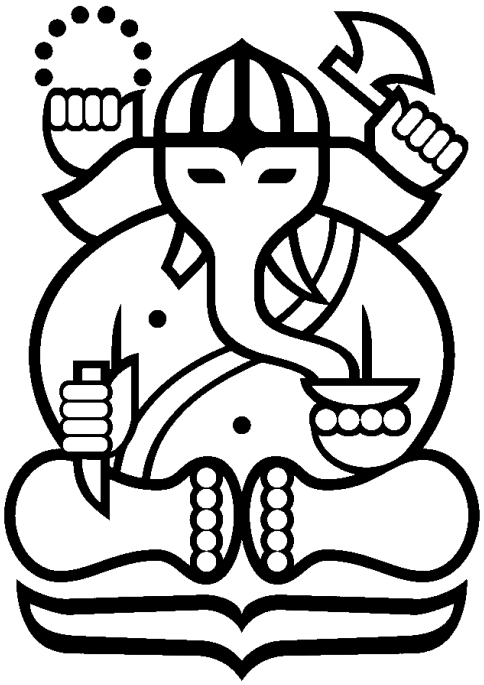
LAPORAN TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA

PENYELESAIAN CYBERPUNK 2077 BREACH PROTOCOL DENGAN ALGORITMA BRUTE FORCE



Oleh

CHELVADINDA

13522154

Sekolah Teknik Elektro dan Informatika – Institut Teknologi Bandung

Semester II tahun 2023/2024

BAB I

PENDAHULUAN

Cyberpunk 2077 Breach Protocol merupakan sebuah minigame meretas yang terdapat dalam permainan video Cyberpunk 2077. Minigame ini mensimulasikan proses peretasan jaringan lokal dari ICE (Intrusion Countermeasures Electronics) yang ada dalam dunia Cyberpunk 2077. Dalam permainan ini, pemain akan dihadapkan dengan beberapa komponen yang harus dipecahkan, antara lain token, matriks, sekuens, dan buffer. Token merupakan karakter alfanumerik yang terdiri dari dua karakter, contohnya E9, BD, dan 55. Matriks adalah kumpulan token yang disusun dalam urutan tertentu. Sekuens adalah rangkaian token, minimal dua token, yang harus cocok dengan urutan tertentu. Buffer merupakan jumlah maksimal token yang dapat disusun secara sekuensial.

Tujuan utama dari permainan ini adalah untuk menemukan solusi paling optimal untuk setiap kombinasi matriks, sekuens, dan ukuran buffer dengan menggunakan algoritma brute force. Algoritma brute force digunakan untuk mencari solusi dengan mencoba semua kemungkinan kombinasi secara sistematis. Dalam laporan ini, akan diimplementasikan sebuah program sederhana menggunakan bahasa pemrograman Python yang mengimplementasikan algoritma Brute Force untuk mencari solusi paling optimal dalam permainan Breach Protocol. Program ini akan mencoba semua kemungkinan kombinasi matriks, sekuens, dan ukuran buffer untuk menemukan solusi terbaik dalam waktu yang efisien.

BAB II

ALGORITMA BRUTE FORCE DAN IMPLEMENTASI

A. Algoritma Brute Force

Algoritma Brute Force adalah salah satu metode penyelesaian masalah yang sederhana namun kuat, yang bekerja dengan mencoba semua kemungkinan solusi secara berurutan dan memilih yang paling optimal. Metode ini tidak menggunakan strategi atau pengetahuan khusus tentang masalah yang sedang dihadapi, tetapi hanya mengandalkan kekuatan komputasi untuk mengecek setiap kemungkinan Solusi.

Langkah-langkah umum dalam algoritma Brute Force adalah sebagai berikut:

1. Generate Semua Kemungkinan Solusi: Algoritma Brute Force akan menghasilkan atau mencoba semua kemungkinan solusi yang mungkin untuk masalah yang diberikan. Ini bisa berupa kombinasi, permutasi, atau variasi dari elemen-elemen yang ada dalam masalah.
2. Evaluasi Setiap Solusi: Setelah solusi dihasilkan, algoritma akan mengevaluasi masing-masing dari mereka untuk melihat apakah memenuhi kriteria atau syarat yang ditetapkan dalam masalah.
3. Pilih Solusi Terbaik: Algoritma kemudian akan memilih solusi yang memenuhi syarat dan dianggap sebagai solusi terbaik sesuai dengan kriteria yang ditetapkan dalam masalah.
4. Optimasi: Terkadang, ada ruang untuk meningkatkan kinerja algoritma Brute Force dengan mengurangi jumlah solusi yang perlu diuji atau dengan meningkatkan efisiensi pengecekan solusi. Namun, pada dasarnya, algoritma ini akan tetap mencoba semua kemungkinan solusi.

Meskipun algoritma Brute Force sederhana dan dapat diterapkan pada berbagai jenis masalah, namun sering kali memiliki kompleksitas waktu yang tinggi, terutama jika ruang pencarian besar atau jumlah kemungkinan solusi sangat besar. Oleh karena itu, Brute Force seringkali tidak praktis untuk masalah yang skala atau kompleksitasnya tinggi, tetapi sering

digunakan untuk masalah yang ukurannya kecil atau ketika tidak ada pendekatan yang lebih efisien yang tersedia.

B. Implementasi

Source Code

Main.py

src > Breach_Protokol > ...

```python
import time
import random

########################################################################## READ ###################################################
def read_file_to_string(file_path):
    try:
        with open(file_path, 'r') as file:
            file_contents = file.read()
        return file_contents
    except FileNotFoundError:
        print(f"File '{file_path}' not found.")
        return None
    except Exception as e:
        print(f"Error:{e}")
        return None

def main_reader_txt(file_path):
    file_content = read_file_to_string(file_path)

    buffer_size = int(file_content[0])
    matrix_width = int(file_content[2])
    matrix_length = int(file_content[4])

    matrix_idx = 6
    matrix = []
    for i in range(matrix_length):
        a = []
        for j in range(matrix_width):
            a.append(file_content[matrix_idx] + file_content[matrix_idx + 1])
            matrix_idx += 3
        matrix.append(a)

    number_of_sequences = int(file_content[matrix_idx])
    sequences = []
    matrix_idx += 2

    for i in range(number_of_sequences):
        string = ''
        while file_content[matrix_idx] != '\n':
            string += file_content[matrix_idx]
            matrix_idx += 1

        matrix_idx += 1

        score = ''
        while (file_content[matrix_idx] != '\n') and (matrix_idx < len(file_content) - 1):
            score += file_content[matrix_idx]
            matrix_idx += 1

        if matrix_idx == len(file_content) - 1:
            score += file_content[matrix_idx]

        if i != number_of_sequences - 1:
            matrix_idx += 1

        sequences.append((string.split(), score))

    return matrix, buffer_size, sequences

```

```python

################################################################## SOLVER ##################################################
def is_valid_move(matrix, pos, row, col):
    rows = len(matrix)
    cols = len(matrix[0])
    return 0 <= row < rows and 0 <= col < cols and not pos[row][col]


def find_paths(matrix, pos, row, col, length, path, coor, all_paths, all_coor, direction):
    if length == 0:
        all_paths.append(path[:])
        all_coor.append(coor[:])
        return

    pos[row][col] = True

    if direction == 'horizontal':
        horizontal_dir = []

        for i in range(1, len(matrix[0])):
            horizontal_dir.append((0, i))
            horizontal_dir.append((0, i * -1))

        # Horizontal movements
        for dr, dc in horizontal_dir:
            new_row, new_col = row + dr, col + dc
            if is_valid_move(matrix, pos, new_row, new_col):
                path.append(matrix[new_row][new_col])
                coor.append((new_col + 1, new_row + 1))
                find_paths(matrix, pos, new_row, new_col, length - 1, path, coor, all_paths, all_coor, 'vertical')
                path.pop()
                coor.pop()
    else:
        vertical_dir = []

        for i in range(1, len(matrix[0])):
            vertical_dir.append((i, 0))
            vertical_dir.append((i * -1, 0))

        # Vertical movements
        for dr, dc in vertical_dir:
            new_row, new_col = row + dr, col + dc
            if is_valid_move(matrix, pos, new_row, new_col):
                path.append(matrix[new_row][new_col])
                coor.append((new_col + 1, new_row + 1))
                find_paths(matrix, pos, new_row, new_col, length - 1, path, coor, all_paths, all_coor, 'horizontal')
                path.pop()
                coor.pop()

    pos[row][col] = False

def find_all_paths(matrix, length):
    rows = len(matrix)
    cols = len(matrix[0])
    all_paths = []
    all_coor = []

```

```
    for j in range(cols):  # Iterate over cells in the first row only
        pos = [[False] * cols for _ in range(rows)]
        path = [matrix[0][j]]
        coor = [(j + 1, 1)]
        find_paths(matrix, pos, 0, j, length - 1, path, coor, all_paths, all_coor, 'vertical')

    return (all_paths, all_coor)

def isSublist(lst, sub):
    if not sub:
        return True
    if not lst:
        return False

    for i in range(len(lst)):
        if lst[i:i + len(sub)] == sub:
            return True

    return False

def max_index(lst):
    if not lst:
        return None  # Return None if the list is empty
    max_value = max(lst)
    max_index = lst.index(max_value)
    return max_index

def main_solver_txt(matrix, buffer_size, sequences):
    start = time.time()
    all_paths = find_all_paths(matrix, buffer_size)[0]
    all_coor = find_all_paths(matrix, buffer_size)[1]

    score_path = []
    for path in all_paths:
        temp_score = 0
        for sequence in sequences:
            if isSublist(path, sequence[0]):
                temp_score += int(sequence[1])
        score_path.append(temp_score)

    #Print the optimal solution
    str = f"{max(score_path)}\n"

    str += f"{' '.join(all_paths[max_index(score_path)])}\n"

    for coor in all_coor[max_index(score_path)]:
        str += f"{coor[0]}, {coor[1]}\n"

    end = time.time()
    res = end - start
    final_res = res * 1000

    str += f"\n{final_res} ms"

    print(f"\n{str}")
    return str

def random_matrix(rows, cols, elements):
```

```
    matrix = []
    for _ in range(rows):
        row = [random.choice(elements) for _ in range(cols)]
        matrix.append(row)

    return matrix

def random_sequence(token, seq_max_size, seq):
    a = [random.choice(token) for _ in range(random.randint(2, seq_max_size))]

    #Loop to ensure that every sequence is unique
    loop_stat = False
    while not loop_stat:
        if a in seq:
            a = [random.choice(token) for _ in range(random.randint(2, seq_max_size))]
        else:
            loop_stat = True

    return a

def save_string_to_file(string, filename):
    with open(filename, 'w') as file:
        file.write(string)

def main_solver_cli():
    numof_unique_tokens = int(input("Jumlah Token: "))

    token = []
    for i in range(numof_unique_tokens):
        token.append(input(f"Token {i + 1}: "))

    buffer_size = int(input("Ukuran Buffer: "))
    # Meminta masukan dari pengguna
    input_string = input("Masukkan jumlah baris dan kolom (dipisahkan spasi): ")
    # Memisahkan masukan menjadi baris dan kolom
    matrix_row, matrix_column = map(int, input_string.split())
    matrix = random_matrix(matrix_row, matrix_column, token)

    numof_sequence = int(input("Jumlah Sekuens: "))
    seq_max_size = int(input("Ukuran Sekuens: "))

    start = time.time()

    seq = []
    for _ in range(numof_sequence):
        a = random_sequence(token, seq_max_size, seq)
        seq.append(a)

    seq_value = [random.randint(10, 50) for _ in range(len(seq))]

    all_paths = find_all_paths(matrix, buffer_size)[0]
    all_coor = find_all_paths(matrix, buffer_size)[1]

    score_path = []
    for path in all_paths:
        temp_score = 0
        for sequence in seq:
            if isSublist(path, sequence):
```

```
                temp_score += int(seq_value[seq.index(sequence)])
            score_path.append(temp_score)

    # Print the matrix and sequence
    print("")
    str = "Matrix:\n"

    for row in matrix:
        str += f"{row}\n"

    str += f"\nSequence:\n"

    for i in range(len(seq)):
        str += f"{seq[i]}\n"
        str += f"{seq_value[i]}\n"

    str += f"\n{max(score_path)}\n{' '.join(all_paths[max_index(score_path)])}\n"
    save_str = f"{max(score_path)}\n{' '.join(all_paths[max_index(score_path)])}\n"

    for coor in all_coor[max_index(score_path)]:
        str += f"{coor[0]}, {coor[1]}\n"
        save_str += f"{coor[0]}, {coor[1]}\n"

    end = time.time()
    res = end - start
    final_res = res * 1000

    str += f"\n{final_res} ms"
    save_str += f"\n{final_res} ms"

    print(str)
    return save_str

################################################################## MAIN ##################################################

print("1. TXT File")
print("2. CLI")
input_mode = input("Pilih metode: ")

str = ""
if input_mode == "1":
    file_path = input("File path: ")
    dir_file_path = "D:\\Semester 4 - Teknik Informatika\\IF2211 - Strategi Algoritma\\Tucil1_13522154\\test" + file_path
    file_content = main_reader_txt(dir_file_path)
    str += main_solver_txt(file_content[0], file_content[1], file_content[2])
elif input_mode == "2":
    str += main_solver_cli()

save_stat = input("Apakah anda ingin menyimpan solusi ini? (Y/N) ")
if save_stat.upper() == "Y":
    file_name = input("Input nama file: ")
    dir_file = "D:\\Semester 4 - Teknik Informatika\\IF2211 - Strategi Algoritma\\Tucil1_13522154\\test" + file_name
    print("Saving...")
    save_string_to_file(str, dir_file)
    print("Saved!")

elif save_stat.upper() == "N":
    print("not saved")
    print("not saved")
    exit()
```

Input / Output

Terminal	File Output.txt
1. TXT File 2. CLI Pilih metode input: 1 Masukkan nama file: input.txt 50 7A BD 7A BD 1C BD 55 1, 1 1, 4 3, 4 3, 5 6, 5 6, 4 5, 4 907.3917865753174 ms Apakah anda ingin menyimpan solusi ini? (Y/N) Y Input nama file: output1 Saving... Saved!	testoutput1 50 7A BD 7A BD 1C BD 55 1, 1 1, 4 3, 4 3, 5 6, 5 6, 4 5, 4 907.3917865753174 ms
1. TXT File 2. CLI Pilih metode: 2 Jumlah Token: 3 Token 1: 1A Token 2: 2A Token 3: 3A Ukuran Buffer: 4 Masukkan jumlah baris dan kolom (dipisahkan spasi): 5 5 Jumlah Sekuens: 3 Ukuran Sekuens: 4 Matrix: ['3A', '3A', '2A', '1A', '1A'] ['1A', '1A', '2A', '3A', '2A'] ['1A', '3A', '1A', '2A', '3A'] ['1A', '1A', '1A', '3A', '1A'] ['2A', '3A', '2A', '3A', '2A'] Sequence: ['2A', '2A', '3A'] 30 ['3A', '1A'] 48 ['1A', '2A'] 22 78 2A 2A 3A 1A 3, 1 3, 2 4, 2 4, 1 2.4518966674804688 ms Apakah anda ingin menyimpan solusi ini? (Y/N) Y Input nama file: output2 Saving... Saved!	testoutput2 78 2A 2A 3A 1A 3, 1 3, 2 4, 2 4, 1 2.4518966674804688 ms

```
1. TXT File
2. CLI
Pilih metode: 2
Jumlah Token: 5
Token 1: 1A
Token 2: 2C
Token 3: 3D
Token 4: DD
Token 5: 3A
Ukuran Buffer: 7
Masukkan jumlah baris dan kolom (dipisahkan spasi): 4 4
Jumlah Sekuens: 2
Ukuran Sekuens: 3

Matrix:
['DD', '3A', 'DD', '3A']
['3A', '3A', '3D', '1A']
['3D', 'DD', '1A', 'DD']
['1A', '3A', '3D', 'DD']

Sequence:
['3D', '1A', '2C']
41
['DD', '2C', '3A']
31

0
DD 3A 3A DD 1A 3D DD
1, 1
1, 2
2, 2
2, 3
3, 3
3, 4
4, 4

13.925552368164062 ms
Apakah anda ingin menyimpan solusi ini? (Y/N) Y
Input nama file: output3
Saving...
Saved!
```

testoutput3

```
0
DD 3A 3A DD 1A 3D DD
1, 1
1, 2
2, 2
2, 3
3, 3
3, 4
4, 4

13.925552368164062 ms
```

```
1. TXT File
2. CLI
Pilih metode: 2
Jumlah Token: 2
Token 1: CD
Token 2: 3A
Ukuran Buffer: 4
Masukkan jumlah baris dan kolom (dipisahkan spasi):
Jumlah Sekuens: 2
Ukuran Sekuens: 2

Matrix:
['CD', 'CD', 'CD']
['CD', '3A', 'CD']
['CD', '3A', '3A']

Sequence:
['3A', 'CD']
26
['CD', '3A']
16

42
CD CD 3A CD
1, 1
1, 2
2, 2
2, 1

0.9548664093017578 ms
Apakah anda ingin menyimpan solusi ini? (Y/N) Y
Input nama file: output4
Saving...
Saved!
```

```
1  42
2  CD CD 3A CD
3  1, 1
4  1, 2
5  2, 2
6  2, 1
7
8  0.9548664093017578 ms
```

```
1. TXT File
2. CLI
Pilih metode: 2
Jumlah Token: 4
Token 1: 3D
Token 2: AZ
Token 3: 7E
Token 4: O9
Ukuran Buffer: 5
Masukkan jumlah baris dan kolom (dipisahkan spasi): 3 3
Jumlah Sekuens: 3
Ukuran Sekuens: 4

Matrix:
['O9', 'O9', '3D']
['3D', 'O9', 'AZ']
['O9', 'AZ', '7E']

Sequence:
['AZ', '7E', '7E', 'AZ']
49
['3D', '3D']
20
['7E', 'O9']
23

23
O9 3D AZ 7E O9
1, 1
1, 2
3, 2
3, 3
1, 3

0.0 ms
Apakah anda ingin menyimpan solusi ini? (Y/N) Y
Input nama file: output5
```

```
23
O9 3D AZ 7E O9
1, 1
1, 2
3, 2
3, 3
1, 3

0.0 ms
```

```
1. TXT File
2. CLI
Pilih metode: 2
Jumlah Token: 3
Token 1: 1L
Token 2: 2L
Token 3: 3L
Ukuran Buffer: 4
Masukkan jumlah baris dan kolom (dipisahkan spasi): 5 5
Jumlah Sekuens: 2
Ukuran Sekuens: 3

Matrix:
['1L', '2L', '3L', '1L', '3L']
['1L', '1L', '2L', '2L', '2L']
['1L', '3L', '2L', '1L', '3L']
['2L', '1L', '2L', '1L', '2L']
['3L', '2L', '3L', '2L', '2L']

Sequence:
['1L', '1L']
18
['3L', '2L']
41

59
1L 1L 3L 2L
1, 1
1, 3
2, 3
2, 5

4.174232482910156 ms
Apakah anda ingin menyimpan solusi ini? (Y/N) Y
Input nama file: output6
```

testoutput6

```
1  59
2  1L 1L 3L 2L
3  1, 1
4  1, 3
5  2, 3
6  2, 5
7
8  4.174232482910156 ms
```

BAB III

KESIMPULAN DAN SARAN

A. Kesimpulan

Dalam hal ini, meskipun algoritma Brute Force dapat memberikan solusi yang andal untuk permainan Breach Protocol, namun perlu diingat bahwa implementasinya mungkin tidak efisien untuk skala masalah yang besar. Pilihan alternatif atau peningkatan pada algoritma mungkin perlu dipertimbangkan untuk meningkatkan kinerja program solusi.

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

B. Saran

Semoga kedepannya lebih diperbanyak live coding agar lebih banyak belajar sebelum Tugas Kecil.