

LAPORAN TUGAS KECIL 1 IF2211 Strategi Algoritma

Penyelesaian Cyberpunk 2077 Breach Protocol

dengan Algoritma Brute Force



Disusun oleh
Elijah Darrellshane Suryanegara (13522097)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

Daftar Isi

BAB 1

Algoritma Brute Force

3

1.1. Pendahuluan

3

1.2. Aplikasi

3

BAB 2

Algoritma Brute Force

4

2.1. Reader

4

a) read_file_to_string

4

b) main_reader_txt

4

2.2. Solver

5

a) is_valid_move

5

b) find_paths

5

c) find_all_paths

6

d) isSublist

6

e) find_max_index

6

f) main_solver_txt

7

g) random_matrix

7

h) random_sequence

8

i) save_string_to_file

8

j) main_solver_cli

8

2.3. Main

9

BAB 3

Testing Program

11

3.1. Initial

11

3.2. Input File .txt

11

3.3. Input Command Line Interface (CLI)

13

LAMPIRAN

16

BAB 1

Algoritma Brute Force

1.1. Pendahuluan

Cyberpunk 2077 Breach Protocol adalah *minigame* meretas dari *video game* Cyberpunk 2077. Dalam *minigame* ini, pemain akan disimulasikan untuk meretas jaringan local dari ICE (Intrusion Countermeasures Electronics) dalam *video game* tersebut. Komponen utama dari *minigame* ini mencakup Token, Matriks, Sekuens, dan Buffer. Pemain akan mengikuti aturan yang meliputi gerakan horizontal dan vertikal secara bergantian, memulai dengan memilih satu token di posisi teratas matriks kemudian mencocokkan sekuens pada token dalam buffer. Satu token dapat digunakan untuk beberapa sekuens dan setiap sekuens memiliki bobot hadiah yang bervariasi, dengan panjang minimal sekuens adalah dua token.

Salah satu pendekatan yang dapat digunakan untuk menemukan solusi yang paling optimal dalam *minigame* Breach Protocol adalah *brute force*. Pendekatan *brute force* adalah sebuah metode dalam pemrograman yang mencari solusi dari sebuah masalah dengan mencoba semua kemungkinan secara sistematis. Ini seringkali digunakan ketika tidak ada struktur atau pola yang dapat dimanfaatkan untuk mengoptimalkan pencarian solusi. Dalam konteks algoritma ini, pendekatan *brute force* akan mencoba semua kemungkinan *sequence* dengan *buffer size* tertentu tanpa memperhatikan apakah *reward* yang dihasilkan *sequence* tersebut optimal atau tidak.

1.2. Aplikasi

Program ini menggunakan pendekatan *brute force* untuk menemukan terlebih dahulu semua kemungkinan kombinasi token buffer dari matriks yang diberikan. Algoritma yang digunakan dalam mencari semua kemungkinan token buffer adalah Depth-First Search (DFS) yang diterapkan secara rekursif. Pertama-tama, terdapat fungsi `is_valid_move()` yang memastikan langkah ke selanjutnya dalam matriks masih dalam batas matriks dan belum dikunjungi sebelumnya. Selanjutnya, terdapat fungsi `find_paths()` yang merupakan inti dari pencarian jalur. Fungsi ini melakukan rekursi dan mengeksplorasi semua kemungkinan langkah, menggabungkan pergerakan horizontal dan vertikal bergantian berdasarkan parameter arah (*direction*). Ketika suatu jalur telah mencapai panjang yang diinginkan, jalur tersebut akan ditambahkan ke dalam daftar `all_paths`. Selain itu, koordinat dari setiap sel yang dikunjungi juga akan ditambahkan ke dalam daftar `all_coor`. Fungsi utama, `find_all_paths()`, menginisialisasi pencarian dengan memulai iterasi dari setiap sel pada baris teratas matriks dan memanggil `find_paths()` untuk setiap sel tersebut. Hasil akhirnya adalah daftar semua jalur yang berhasil ditemukan beserta koordinat setiap sel yang dilewati dalam setiap jalur tersebut. Meskipun pendekatan ini sederhana dan dapat diterapkan pada berbagai masalah, tetapi mencoba semua kemungkinan memiliki konsekuensi berupa waktu eksekusi yang menjadi sangat lambat jika ukuran masalah cukup besar.

BAB 2

Algoritma Brute Force

2.1. Reader

a) read_file_to_string

```
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"An error occurred while reading the file: {e}")
        return None
```

b) main_reader_txt

```
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

```

2.2. Solver

a) is_valid_move

```

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

```

b) find_paths

```

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

    visited[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, visited, new_row, new_col):
                path.append(matrix[new_row][new_col])
                coor.append((new_col + 1, new_row + 1))
                find_paths(matrix, visited, 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, visited, new_row, new_col):
            path.append(matrix[new_row][new_col])
            coor.append((new_col + 1, new_row + 1))
            find_paths(matrix, visited, new_row, new_col, length - 1,
path, coor, all_paths, all_coor, 'horizontal')
            path.pop()
            coor.pop()

    visited[row][col] = False

```

c) find_all_paths

```

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
        visited = [[False] * cols for _ in range(rows)]
        path = [matrix[0][j]]
        coor = [(j + 1, 1)]
        find_paths(matrix, visited, 0, j, length - 1, path, coor,
all_paths, all_coor, 'vertical')

    return (all_paths, all_coor)

```

d) isSublist

```

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

```

e) find_max_index

```

def find_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

```

f) main_solver_txt

```

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[find_max_index(score_path)])}\n"

    for coor in all_coor[find_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

```

g) random_matrix

```

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

    return matrix

```

h) random_sequence

```

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

```

i) save_string_to_file

```

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

```

j) main_solver_cli

```

def main_solver_cli():
    numof_unique_tokens = int(input("Number of unique tokens: "))

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

    buffer_size = int(input("Buffer size: "))
    matrix_row = int(input("Number of matrix rows: "))
    matrix_column = int(input("Number of matrix columns: "))
    matrix = random_matrix(matrix_row, matrix_column, token)

    numof_sequence = int(input("Number of sequences: "))
    seq_max_size = int(input("Maximum size of sequence: "))

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

for coor in all_coor[find_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

```

2.3. Main

```

from solver import main_solver_txt, main_solver_cli, save_string_to_file
from reader import main_reader_txt

print("=====")
print("CYBERPUNK 2077 BREACH PROTOCOL BRUTE FORCE SOLVER")
print("=====")
print("Available input method")
print("1. TXT File")
print("2. Command Line Interface")
input_mode = input("Select your preferred input method: ")

str = ""
if input_mode == "1":
    file_path = input("File path: ")
    dir_file_path = "../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("Would you like to save the solution? (Y/N) ")
if save_stat.upper() == "Y":
    file_name = input("Input file name: ")
    dir_file = "../test/" + file_name
    print("Saving...")
    save_string_to_file(str, dir_file)
    print("Solution saved. Have fun!")

elif save_stat.upper() == "N":
    print("Solution not saved. Have fun!")
    exit()

```

BAB 3

Testing Program

3.1. Initial

```
=====
CYBERPUNK 2077 BREACH PROTOCOL BRUTE FORCE SOLVER
=====

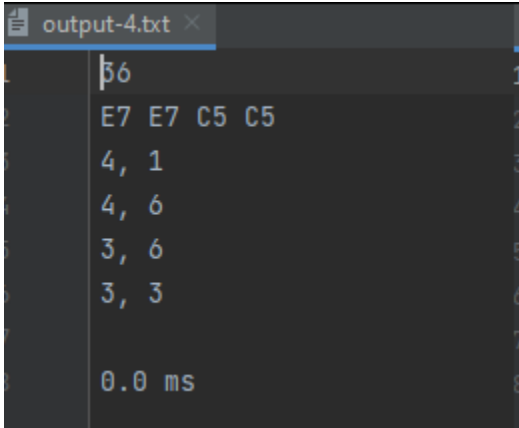
Available input method
1. TXT File
2. Command Line Interface
Select your preferred input method: 
```

3.2. Input File .txt

No.	File input .txt	Terminal	File output .txt
1	<pre>input.txt 1 7 2 6 6 3 7A 55 E9 E9 1C 55 4 55 7A 1C 7A E9 55 5 55 1C 1C 55 E9 BD 6 BD 1C 7A 1C 55 BD 7 BD 55 BD 7A 1C 1C 8 1C 55 55 7A 55 7A 9 3 10 BD E9 1C 11 15 12 BD 7A BD 13 20 14 BD 1C BD 55 15 30</pre>	<pre>Terminal: Local (6) × + v 50 7A BD 7A BD 1C BD 55 1, 1 1, 4 3, 4 3, 5 6, 5 6, 4 5, 4</pre>	<pre>output-1.txt × 1 50 2 7A BD 7A BD 1C BD 55 3 1, 1 4 1, 4 5 3, 4 6 3, 5 7 6, 5 8 6, 4 9 5, 4 10 11 2986.001968383789 ms</pre>

No.	File input .txt	Terminal	File output .txt
2	<pre> input.txt 1 7 2 6 6 3 7A 55 E9 E9 1C 55 4 55 7A 1C 7A E9 55 5 55 1C 1C 55 E9 BD 6 BD 1C 7A 1C 55 BD 7 BD 55 BD 7A 1C 1C 8 1C 55 55 7A 55 7A 9 2 10 7A 55 1C 11 30 12 BD 55 7A 13 20 </pre>	<pre> 50 7A 55 1C 1C BD 55 7A 1, 1 1, 2 1, 2 6, 2 4, 2 1457.4599266052246 ms </pre>	<pre> 50 7A 55 1C 1C BD 55 7A 1, 1 1, 2 3, 2 3, 3 6, 3 6, 2 4, 2 1457.4599266052246 ms </pre>
3	<pre> input.txt 1 7 2 5 4 3 7A 55 E9 E9 1C 4 55 7A 1C 7A E9 5 55 1C 1C 55 E9 6 BD 1C 7A 1C 55 7 3 8 E9 1C 7A 9 32 10 1C E9 7A 55 11 25 12 7A 55 1C 1C 13 40 </pre>	<pre> 72 E9 1C 7A 55 1C 1C 7A 3, 1 3, 2 4, 2 4, 3 2, 3 2, 4 3, 4 78.5226821899414 ms </pre>	<pre> output-3.txt 1 72 2 E9 1C 7A 55 1C 1C 7A 3 3, 1 4 3, 2 5 4, 2 6 4, 3 7 2, 3 8 2, 4 9 3, 4 10 78.5226821899414 ms </pre>

3.3. Input Command Line Interface (CLI)

No.	Terminal	File output .txt
1	<pre>===== CYBERPUNK 2077 BREACH PROTOCOL BRUTE FORCE SOLVER ===== Available input method 1. TXT File 2. Command Line Interface Select your preferred input method: 2 Number of unique tokens: 5 Token 1: A1 Token 2: B3 Token 3: C5 Token 4: E7 Token 5: F9 Buffer size: 4 Number of matrix rows: 6 Number of matrix columns: 6 Number of sequences: 3 Maximum size of sequence: 3 Matrix: ['C5', 'C5', 'F9', 'E7', 'B3', 'B3'] ['A1', 'F9', 'C5', 'B3', 'A1', 'C5'] ['B3', 'A1', 'C5', 'F9', 'B3', 'A1'] ['C5', 'F9', 'B3', 'F9', 'F9', 'A1'] ['E7', 'A1', 'F9', 'F9', 'E7', 'B3'] ['A1', 'C5', 'C5', 'E7', 'F9', 'F9'] Sequence: ['E7', 'C5', 'C5'] 36 ['A1', 'F9', 'B3'] 32 ['C5', 'A1', 'A1'] 21 36 E7 E7 C5 C5 4, 1 4, 6 3, 6 3, 3 0.0 ms</pre>	 <pre>output-4.txt x 36 E7 E7 C5 C5 4, 1 4, 6 3, 6 3, 3 0.0 ms</pre>

2

Select your preferred input method: 2

Number of unique tokens: 6

Token 1: A5

Token 2: B2

Token 3: C3

Token 4: D4

Token 5: E9

Token 6: F1

Buffer size: 5

Number of matrix rows: 6

Number of matrix columns: 6

Number of sequences: 4

Maximum size of sequence: 3

Matrix:

['B2', 'F1', 'B2', 'D4', 'C3', 'E9']

['F1', 'F1', 'B2', 'E9', 'E9', 'E9']

['F1', 'A5', 'B2', 'E9', 'F1', 'B2']

['D4', 'D4', 'A5', 'F1', 'B2', 'D4']

['F1', 'B2', 'E9', 'D4', 'C3', 'D4']

['B2', 'E9', 'D4', 'C3', 'E9', 'E9']

Sequence:

['F1', 'D4']

38

['E9', 'A5']

10

['B2', 'B2', 'D4']

47

['D4', 'E9', 'F1']

43

90

B2 B2 D4 E9 F1

1, 1

1, 6

3, 6

3, 5

1, 5

40.001869201660156 ms

Would you like to save the solution? (Y/N) Y

Input file name: output-5.txt

Saving...

Solution saved. Have fun!

output-5.txt ×

90

B2 B2 D4 E9 F1

1, 1

1, 6

3, 6

3, 5

1, 5

40.001869201660156 ms

3

```
=====
CYBERPUNK 2077 BREACH PROTOCOL BRUTE FORCE SOLVER
=====
```

Available input method

1. TXT File

2. Command Line Interface

Select your preferred input method: 2

Number of unique tokens: 6

Token 1: A1

Token 2: B2

Token 3: C3

Token 4: D4

Token 5: E5

Token 6: F6

Buffer size: 7

Number of matrix rows: 10

Number of matrix columns: 10

Number of sequences: 5

Maximum size of sequence: 5

Matrix:

```
[ 'A1', 'E5', 'C3', 'C3', 'E5', 'C3', 'B2', 'E5', 'F6', 'C3' ]
[ 'F6', 'B2', 'E5', 'F6', 'F6', 'D4', 'C3', 'D4', 'F6', 'B2' ]
[ 'B2', 'A1', 'A1', 'D4', 'D4', 'F6', 'E5', 'D4', 'A1', 'F6' ]
[ 'F6', 'E5', 'E5', 'B2', 'A1', 'F6', 'C3', 'D4', 'F6', 'F6' ]
[ 'A1', 'F6', 'D4', 'F6', 'D4', 'D4', 'E5', 'E5', 'A1', 'F6' ]
[ 'E5', 'F6', 'C3', 'C3', 'C3', 'E5', 'C3', 'E5', 'E5', 'D4' ]
[ 'D4', 'C3', 'D4', 'C3', 'A1', 'A1', 'C3', 'B2', 'B2', 'A1' ]
[ 'C3', 'E5', 'F6', 'A1', 'D4', 'A1', 'F6', 'E5', 'D4', 'E5' ]
[ 'A1', 'C3', 'F6', 'F6', 'E5', 'B2', 'E5', 'B2', 'F6', 'C3' ]
[ 'E5', 'A1', 'F6', 'B2', 'A1', 'A1', 'F6', 'A1', 'E5', 'A1' ]
```

Sequence:

```
[ 'E5', 'F6' ]
```

24

```
[ 'C3', 'C3', 'D4' ]
```

37

```
[ 'E5', 'D4', 'C3', 'B2' ]
```

45

```
[ 'D4', 'D4', 'D4', 'A1' ]
```

22

```
[ 'F6', 'A1', 'D4' ]
```

12

82

```
C3 C3 D4 E5 D4 C3 B2
```

```
4, 1
```

```
4, 7
```

```
3, 7
```

```
3, 2
```

```
6, 2
```

```
6, 1
```

```
7, 1
```

79813.7469291687 ms

Would you like to save the solution? (Y/N) Y

Input file name: output-6.txt

Saving...

Solution saved. Have fun!

output-6.txt

1 B2

2 C3 C3 D4 E5 D4 C3 B2

3 4, 1

4 4, 7

5 3, 7

6 3, 2

7 6, 2

8 6, 1

9 7, 1

10

1 79813.7469291687 ms

LAMPIRAN

Link repository: https://github.com/HenryofSkalitz1202/Tucil1_13522097

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program berhasil dijalankan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Program dapat membaca masukan berkas .txt	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Program dapat menghasilkan masukan secara acak	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. Solusi yang diberikan program optimal	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6. Program dapat menyimpan solusi dalam berkas .txt	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7. Program memiliki GUI	<input type="checkbox"/>	<input checked="" type="checkbox"/>