TUGAS KECIL 1

IF2211 STRATEGI ALGORITMA

SEMESTER II TAHUN 2023/2024

"Penyelesaian *Cyberpunk 2077 Breach Protocol* dengan Algoritma Brute Force"

OLEH:

Irfan Sidiq Permana 13522007

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

BAB I
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.

Komponen pada permainan ini antara lain adalah:

1. Token – terdiri dari dua karakter alfanumerik seperti E9, BD, dan 55.
2. Matriks – terdiri atas token-token yang akan dipilih untuk menyusun urutan kode.
3. Sekuens – sebuah rangkaian token (dua atau lebih) yang harus dicocokkan.
4. Buffer – jumlah maksimal token yang dapat disusun secara sekuensial.

Aturan permainan Breach Protocol antara lain:

1. Pemain bergerak dengan pola horizontal, vertikal, horizontal, vertikal (bergantian) hingga
1. semua sekuens berhasil dicocokkan atau buffer penuh.
2. Pemain memulai dengan memilih satu token pada posisi baris paling atas dari matriks.
3. Sekuens dicocokkan pada token-token yang berada di buffer.
4. Satu token pada buffer dapat digunakan pada lebih dari satu sekuens.
5. Setiap sekuens memiliki bobot hadiah atau reward yang variatif.
6. Sekuens memiliki panjang minimal berupa dua token.

Pada Tugas Kecil 1 ini, penulis merancang program untuk mencari solusi dari permainan Cyberpunk 2077 Breach Protocol menggunakan bahasa Python. Program menerima input berupa file *txt* atau input keyboard untuk memasukkan data permainan, lalu mengembalikan salah satu solusi yang mungkin dengan nilai *reward* yang terbesar.

BAB II
ALGORITMA BRUTE FORCE

Algoritma *brute force* adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Algoritma *brute force* umumnya didasarkan pada pernyataan pada persoalan (*problem statement*), dan definisi/konsep yang dilibatkan. Algoritma *brute force* memecahkan persoalan dengan sangat sederhana, langsung, dan jelas caranya (*obvious way*). Kelemahan utamanya dari algoritma *brute force* yaitu umumnya tidak mangkus dan membutuhkan volume komputasi yang besar.

Pada program ini, penulis memanfaatkan algoritma *brute force* untuk menyelesaikan permasalahan Cyberpunk 2077 Breach Protocol dengan implementasi sebagai berikut:

1. Algoritma *brute force* diimplementasikan secara rekursif, dengan parameternya yaitu posisi sel saat ini, isi buffer saat ini, koordinat dari tiap sel yang telah dipilih sejauh ini, serta boolean *horizontal* yang dapat bernilai *true* atau *false*.
2. Pencarian dimulai dari baris pertama, yaitu dengan posisi [0, 0], buffer dan koordinat kosong, serta boolean *horizontal* bernilai *true*. Karena *horizontal* bernilai *true*, maka program akan mengecek semua sel yang horizontal dengan posisi [0, 0], yaitu sel mulai dari [0, 0] hingga [0, *n-1*].
3. Untuk setiap sel yang sedang dicek, program akan mengecek apakah sel tersebut sudah dikunjungi sebelumnya.
    - Jika iya, maka program akan *skip* ke sel berikutnya.
    - Jika belum, maka program menandai bahwa sel tersebut sudah dikunjungi lalu memanggil secara rekursif fungsi *brute force* dengan menambahkan isi buffer dengan sel tersebut, menambahkan isi koordinat dengan koordinat sel tersebut, lalu men-*toggle* nilai *horizontal* berdasarkan nilai *horizontal* sebelumnya. Setelah memanggil secara rekursif, program menghilangkan tanda bahwa sel tersebut telah dikunjungi lalu lanjut ke sel berikutnya.
4. Jika buffer sudah penuh atau sudah tidak ada lagi sel yang horizontal/vertikal dengan sel saat ini yang belum dikunjungi, maka nilai buffer dan koordinat disimpan sebagai salah satu solusi yang mungkin agar nantinya dihitung nilai reward dan dipilih buffer yang memiliki nilai reward tertinggi.

BAB III
IMPLEMENTASI KODE ALGORITMA

Program terdiri dari tiga file source code, yaitu main.py, util.py, dan protocol.py. File main.py berfungsi sebagai file utama yang dijalankan, file util.py berisi fungsi-fungsi yang digunakan pada file main.py, dan file protocol.py berisi algoritma utama dari program.

main.py

```
import util

util.clear_screen()

# Input data
input_valid = False
while not input_valid:
    input_choice = input("Pilih metode input yang diinginkan (f: file, k:
keyboard): ")
    if input_choice == "f" or input_choice == "F":
        game = util.input_from_file()
        input_valid = True
    elif input_choice == "k" or input_choice == "K":
        game = util.input_from_keyboard()
        input_valid = True
    else:
        print("Mohon masukkan input yang valid (f/k).\n")

if input_choice == "f" or input_choice == "F":
    print("\nGame berhasil dimuat!")
else:
    print("\nGame berhasil dihasilkan secara random!")
    game.display_generated_game()

# Solve from given data
print("\nMohon tunggu...\n")
game.solve()
game.display_solution()

# Save result if prompted
confirm_save = input("\nApakah ingin menyimpan solusi? (y/n): ")
if (confirm_save == "y" or confirm_save == "Y"):
    game.export_file()
    print("File berhasil disimpan!")

print("\nTerima kasih telah menggunakan program ini!\n")
```

util.py

```
from protocol import BreachProtocol
import math
import os

def clear_screen():
    if os.name == 'nt':
        os.system('cls')
    else:
        os.system('clear')

def input_from_file():
    default_load_path = "../input/"
    path_valid = False
    while not path_valid:
        filename = input("\nMasukkan nama file yang ingin dimuat: ")
        if os.path.exists(default_load_path + filename):
            path_valid = True
        else:
            print(f"File {filename} tidak ditemukan!\n")

    return BreachProtocol.load_file(filename)

def input_from_keyboard():
    number_of_tokens = int(input("\nMasukkan jumlah token unik: "))
    token = input("Masukkan token: ").split()
    buffer_size = int(input("Masukkan ukuran buffer: "))
    matrix_width, matrix_height = map(int, input("Masukkan ukuran matriks
(Contoh: '6 6'): ").split())
    number_of_sequence = int(input("Masukkan jumlah sekuens: "))

    min_sequence_length = find_min_sequence_length(number_of_tokens,
number_of_sequence)
    sequence_size_valid = False

    while not sequence_size_valid:
        max_sequence_length = int(input(f"Masukkan panjang maksimal sekuens
(minimal ≥ {min_sequence_length}): "))
        if max_sequence_length >= min_sequence_length:
            sequence_size_valid = True
        else:
            print("Panjang tidak valid!\n")

    return BreachProtocol.random_generate(token, buffer_size, matrix_width,
matrix_height, number_of_sequence, max_sequence_length)

def find_min_sequence_length(a: int, s: int):
    return math.ceil(math.log((s+a)*(a-1)/a + 1, a))
```

protocol.py

```
from typing import List, Tuple
import random
import time
import datetime
import os
import sys

class BreachProtocol:
    possible_solution_buffers = []
    possible_solution_coordinates = []

    def __init__(self, matrix: List[List[str]], sequences: List[str],
sequence_rewards: List[int], buffer_size: int, path: str):
        self.filename = os.path.basename(path) if path != "" else path
        self.matrix = matrix
        self.sequences = sequences
        self.sequence_rewards = sequence_rewards
        self.buffer_size = buffer_size
        self.visited = [[False for _ in range(len(matrix[0]))] for _ in
range(len(matrix))]

    def load_file(path: str):
        default_load_path = "../test/input/"
        file = open(default_load_path + path, "r")
        f = file.readlines()

        input = []
        for line in f:
            input.append(line.strip())

        current_index = 0
        buffer_size = int(input[current_index])
        matrix_width, matrix_height = map(int, input[current_index +
1].split())
        matrix = [["" for _ in range(matrix_width)] for _ in
range(matrix_height)]

        current_index += 2
        for i in range(matrix_height):
            row = input[i + current_index].split()
            for j in range(matrix_width):
                matrix[i][j] = row[j]

        current_index += matrix_height
        number_of_sequences = int(input[current_index])
        sequences = ["" for _ in range(number_of_sequences)]
        sequence_rewards = [0 for _ in range(number_of_sequences)]
```

```
        idx = current_index + 1
        for i in range(number_of_sequences):
            sequences[i] = input[idx]
            sequence_rewards[i] = int(input[idx + 1])
            idx += 2

        file.close()
        return BreachProtocol(matrix, sequences, sequence_rewards,
buffer_size, path)

    def random_generate(tokens: List[str], buffer_size: int, matrix_width:
int, matrix_height: int, number_of_sequences: int, max_sequence_size: int):
        matrix = [["" for _ in range(matrix_width)] for _ in
range(matrix_height)]
        sequences = ["" for _ in range(number_of_sequences)]
        sequence_rewards = [0 for _ in range(number_of_sequences)]

        for i in range(matrix_height):
            for j in range(matrix_width):
                idx = random.randint(0, len(tokens) - 1)
                matrix[i][j] = tokens[idx]

        for i in range(number_of_sequences):
            sequence = ""
            sequence_length = random.randint(2, max_sequence_size)
            contains_duplicate_sequence = True

            while contains_duplicate_sequence:
                sequence = ""
                for _ in range(sequence_length):
                    idx = random.randint(0, len(tokens) - 1)
                    sequence += tokens[idx] + " "

                contains_duplicate_sequence = False
                for j in range(i):
                    if sequence == sequences[j]:
                        contains_duplicate_sequence = True
                        break

            sequences[i] = sequence
            sequence_rewards[i] = random.randint(1, 5) * 2 * sequence_length

        return BreachProtocol(matrix, sequences, sequence_rewards,
buffer_size, "")

    def display_generated_game(self):
        print("\nMatriks yang dihasilkan:")
```

```
        for i in range(len(self.matrix)):
            for j in range(len(self.matrix[0])):
                print(self.matrix[i][j] + " ", end="")
            print()

        print("\nSekuens yang dihasilkan:")
        for i in range(len(self.sequences)):
            print(f"{i+1}.", self.sequences[i] + "dengan reward berbobot",
self.sequence_rewards[i])

    def solve(self):
        start_time = time.time()

        self.generate_possible_solutions(0, 0, [], [], True)
        self.choose_best_solution()

        end_time = time.time()
        self.execution_time = end_time - start_time

    def generate_possible_solutions(self, row: int, col: int,
current_buffer: List[str], current_coordinates: List[Tuple[int, int]],
horizontal: bool):
        if len(current_buffer) == self.buffer_size:
            self.possible_solution_buffers.append(current_buffer)
            self.possible_solution_coordinates.append(current_coordinates)
            return

        continue_search = False
        if horizontal:
            for i in range(len(self.matrix[0])):
                if not self.visited[row][i]:
                    continue_search = True
                    self.visited[row][i] = True
                    self.generate_possible_solutions(
                        row, i,
                        current_buffer + [self.matrix[row][i]],
                        current_coordinates + [(i + 1, row + 1)],
                        False)
                    self.visited[row][i] = False
        else:
            for i in range(len(self.matrix)):
                if not self.visited[i][col]:
                    continue_search = True
                    self.visited[i][col] = True
                    self.generate_possible_solutions(
                        i, col,
                        current_buffer + [self.matrix[i][col]],
                        current_coordinates + [(col + 1, i + 1)],
```

```
                        True)
                    self.visited[i][col] = False


        if not continue_search:
            self.possible_solution_buffers.append(current_buffer)
            self.possible_solution_coordinates.append(current_coordinates)

    def choose_best_solution(self):
        best_solution_index = -999
        max_reward = 0

        for i in range(len(self.possible_solution_buffers)):
            total_reward = 0
            for j in range(len(self.sequences)):
                if self.contains_sequence(self.possible_solution_buffers[i],
self.sequences[j].split()):
                    total_reward += self.sequence_rewards[j]

            if total_reward > max_reward:
                max_reward = total_reward
                best_solution_index = i

        self.best_solution_index = best_solution_index
        self.max_reward = max_reward

    def contains_sequence(self, buffer: List[str], sequence: List[str]):
        if len(sequence) > len(buffer):
            return False

        i = 0
        found = False

        while not found and i < len(buffer) - len(sequence) + 1:
            if buffer[i] == sequence[0]:
                j, idx = 1, i + 1
                match = True

                while match and j < len(sequence):
                    if buffer[idx] != sequence[j]:
                        match = False
                    j, idx = j + 1, idx + 1

                if match:
                    found = True
            i += 1

        return found
```

```
    def display_solution(self):
        print(f"Reward maksimal: {self.max_reward}")

        if self.best_solution_index != -999:     # Solution found
            print(f"Isi buffer:
{self.possible_solution_buffers[self.best_solution_index]}")
            print(f"Koordinat tiap token:
{self.possible_solution_coordinates[self.best_solution_index]}")
        else:
            print("Tidak ada solusi")

        print(f"Waktu eksekusi: {round(self.execution_time * 1000)} ms")

    def export_file(self):
        default_save_path = "output/"
        file_name =
str(datetime.datetime.now().replace(microsecond=0)).replace(":", "-") +
".txt"

        with open(default_save_path + file_name, "w") as file:
            if self.filename != "":
                file.write(f"Nama file: {self.filename}\n\n")

            file.write(f"Reward maksimal: {self.max_reward}\n")

            if self.best_solution_index != -999:
                file.write(f"Isi buffer:
{self.possible_solution_buffers[self.best_solution_index]}\n")
                file.write(f"Koordinat tiap token:
{self.possible_solution_coordinates[self.best_solution_index]}\n")
            else:
                file.write("Tidak ada solusi\n")

            file.write(f"Waktu eksekusi: {round(self.execution_time * 1000)}
ms")
```

BAB IV

HASIL UJI COBA PROGRAM

Program dijalankan dengan memasukkan command “python main.py” pada folder src. Tampilan awal program adalah sebagai berikut:

Gambar 4.1 Tampilan awal program

Selanjutnya pengguna dapat memilih untuk menginput menggunakan file atau keyboard. Bila menggunakan keyboard, maka data akan dihasilkan secara random sesuai parameter yang diinput pengguna selanjutnya.

A. Input melalui file

1. Data Uji 1

```
7
6 6
7A 55 E9 E9 1C 55
55 7A 1C 7A E9 55
55 1C 1C 55 E9 BD
BD 1C 7A 1C 55 BD
BD 55 BD 7A 1C 1C
1C 55 55 7A 55 7A
3
BD E9 1C
15
BD 7A BD
20
BD 1C BD 55
30
```

```
Masukkan nama file yang ingin dimuat: percobaan1.txt

Game berhasil dimuat!

Mohon tunggu...

Reward maksimal: 50
Isi buffer: 7A BD 7A BD 1C BD 55
Koordinat tiap token:
1, 1
1, 4
3, 4
3, 5
6, 5
6, 3
1, 3

Waktu eksekusi: 198 ms
```

Gambar 4.2 Hasil Data Uji 1

2. Data Uji 2

```
10
3 3
55 1C 55
BD 7A BD
E9 BD 7A
2
1C 7A BD E9 7A 55
55
10
55 BD
5
```

```
Masukkan nama file yang ingin dimuat: percobaan2.txt

Game berhasil dimuat!

Mohon tunggu...

Reward maksimal: 0
Tidak ada solusi

Waktu eksekusi: 0 ms
```

Gambar 4.3 Hasil Data Uji 2

3. Data Uji 3

```
10
3 3
55 1C 55
BD 7A BD
E9 BD 7A
2
1C 7A BD E9 7A 55
55
10
55 BD
5
```

```
Masukkan nama file yang ingin dimuat: percobaan3.txt

Game berhasil dimuat!

Mohon tunggu...

Reward maksimal: 10
Isi buffer: 1C 7A BD E9 7A 55 55
Koordinat tiap token:
2, 1
2, 2
1, 2
1, 3
3, 3
3, 1
1, 1

Waktu eksekusi: 0 ms
```

Gambar 4.4 Hasil Data Uji 3

4. Data Uji 4

```
6
6 6
7A BD BD 55 55 BD
1C BD 55 1C 7A BD
7A 1C BD 55 55 1C
7A 55 1C 55 7A BD
55 55 55 7A 7A BD
1C 1C 55 BD 1C 1C
3
55 1C
20
BD 7A BD 55
24
7A 1C 55 1C
32
```

```
Masukkan nama file yang ingin dimuat: percobaan4.txt

Game berhasil dimuat!

Mohon tunggu...

Reward maksimal: 52
Isi buffer: 7A 1C 55 1C 7A 7A
Koordinat tiap token:
1, 1
1, 2
3, 2
3, 4
1, 4
1, 3

Waktu eksekusi: 39 ms
```

Gambar 4.5 Hasil Data Uji 4

B. Input melalui keyboard

1. Data Uji 1

```
Pilih metode input yang diinginkan (f: file, k: keyboard): k

Masukkan jumlah token unik: 5
Masukkan token: 7A BD 55 1C E9
Masukkan ukuran buffer: 6
Masukkan ukuran matriks (Contoh: '6 6'): 6 6
Masukkan jumlah sekuens: 4
Masukkan panjang maksimal sekuens (minimal ≥ 2): 5

Game berhasil dihasilkan secara random!

Matriks yang dihasilkan:
7A 7A 55 E9 E9 1C
55 1C 1C 55 55 7A
7A 1C 1C 7A 1C 55
E9 BD 1C E9 7A 1C
7A 55 7A E9 BD 1C
55 BD 7A 55 55 E9

Sekuens yang dihasilkan:
1. 7A E9 BD 1C 1C dengan reward berbobot 40
2. 55 7A BD BD dengan reward berbobot 16
3. 55 55 7A 55 E9 dengan reward berbobot 50
4. BD 7A BD 7A BD dengan reward berbobot 20

Mohon tunggu...

Reward maksimal: 50
Isi buffer: 7A 55 55 7A 55 E9
Koordinat tiap token:
1, 1
1, 2
4, 2
4, 3
6, 3
6, 6

Waktu eksekusi: 94 ms
```

Gambar 4.6 Hasil Data Uji 5

2. Data Uji 2

```
Pilih metode input yang diinginkan (f: file, k: keyboard): k

Masukkan jumlah token unik: 3
Masukkan token: 7A 55 E9
Masukkan ukuran buffer: 5
Masukkan ukuran matriks (Contoh: '6 6'): 4 5
Masukkan jumlah sekuens: 3
Masukkan panjang maksimal sekuens (minimal ≥ 2): 5

Game berhasil dihasilkan secara random!

Matriks yang dihasilkan:
55 7A E9 7A
55 7A E9 7A
E9 E9 E9 55
E9 E9 E9 7A
E9 7A 7A 7A
```

```
Sekuens yang dihasilkan:
1. 55 55 7A dengan reward berbobot 30
2. 7A 7A 55 55 55 dengan reward berbobot 40
3. 7A E9 dengan reward berbobot 20

Mohon tunggu...

Reward maksimal: 50
Isi buffer: 55 55 7A 7A E9
Koordinat tiap token:
1, 1
1, 2
2, 2
2, 1
3, 1

Waktu eksekusi: 0 ms
```

Gambar 4.7 Hasil Data Uji 6

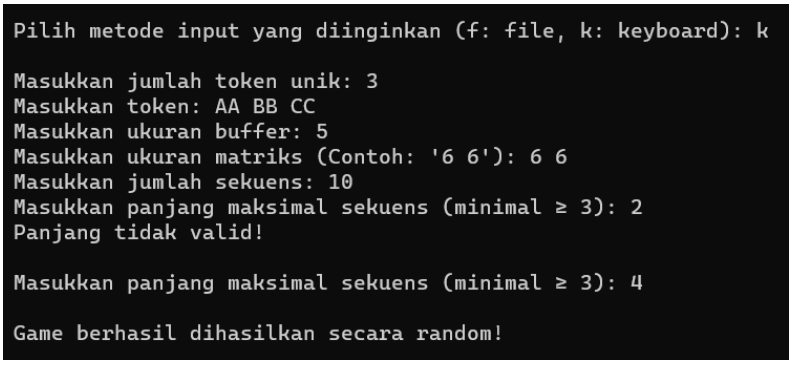
C. Contoh input yang tidak valid

```
Pilih metode input yang diinginkan (f: file, k: keyboard): f

Masukkan nama file yang ingin dimuat: percobaan4.txt
Jumlah sekuens tidak boleh negatif!
Isi file txt tidak valid!


Masukkan nama file yang ingin dimuat: percobaan3.txt
Ukuran matriks tidak boleh negatif!
Isi file txt tidak valid!
```

Gambar 4.8 Hasil Data Uji 7

```
Pilih metode input yang diinginkan (f: file, k: keyboard): k

Masukkan jumlah token unik: 0
Jumlah tidak valid! Pastikan jumlah token unik >= 2


Masukkan jumlah token unik: 3
Masukkan token: AA BB
Jumlah token terlalu sedikit!

Masukkan token: AA BB CC DD
Jumlah token terlalu banyak!

Masukkan token: AA BB CC
Masukkan ukuran buffer: -1
Ukuran buffer tidak boleh negatif!
```

Gambar 4.9 Hasil Data Uji 8



Gambar 4.10 Hasil Data Uji 9

DAFTAR REFERENSI

Algoritma Brute Force. (n.d.). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf.

LAMPIRAN

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

Repositori github dapat diakses melalui pranala berikut:

https://github.com/IrfanSidiq/Tucil1_13522007