TUGAS KECIL

Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force



OLEH

JIMLY NUR ARIF

13522123

Dalam rangka memenuhi Tugas Kecil mata kuliah Strategi Algoritma

Teknik Informatika

STEI ITB

# Table of Contents

[CheckList 2](#_Toc158799317)

[Algoritma brute force 2](#_Toc158799318)

[Source Program bf.py 2](#_Toc158799319)

[Tangkapan layar 15](#_Toc158799320)

[Pranala Repository 18](#_Toc158799321)

# CheckList

|  |  |  |
| --- | --- | --- |
| **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 |

# Algoritma brute force

1. Deteksi ukuran matriks
2. Untuk setiap kolom pada baris pertama digunakan sebagai pangkal path
3. Setiap pangkal path dihasilkan segala kemungkinan dengan fungsi generate\_pattern
4. Fungsi generate\_pattern akan secara masif mencari pattern, misal belok kanan, maka langkah berikutnya ke atas, ke kanan lagi, dan ke bawah
5. Bila sudah mengenai dinding matriks harus belok
6. Untuk meningkat efisiensi, maka pembuatan pola akan terhenti bila perubahan arah pada pattern sudah melebihi jumlah buffer

# Source Program bf.py

import time

import random

def generate\_patterns(n, m, max\_len, buffer\_size):

directions = [(0, 1), (0, -1), (1, 0), (-1, 0)] # Right, left, down, up directions

patterns = []

def search(x, y, path, change\_dir, preddx, preddy):

if len(path) > max\_len:

return

patterns.append(path[:])

for i in range(4):

dx, dy = directions[i]

nx, ny = x + dx, y + dy

if (len(path) > 1 or (dx, dy) == (1, 0)) and 0 <= nx < n and 0 <= ny < m and (nx, ny) not in path and change\_dir < buffer\_size:

if (preddx == dx or preddy == dy):

path.append((nx, ny))

search(nx, ny, path, change\_dir, dx, dy)

else:

path.append((nx, ny))

search(nx, ny, path, change\_dir + 1, dx, dy)

path.pop()

for i in range(m):

search(0, i, [(0, i)], 1, 0, 0)

return patterns

def deletePath(patterns):

for i in range(len(patterns)):

coordinates = patterns[i]

j = 1

while j < len(coordinates) - 1:

if coordinates[j][1] == coordinates[j - 1][1] and coordinates[j][1] == coordinates[j + 1][1]:

del coordinates[j]

elif coordinates[j][0] == coordinates[j - 1][0] and coordinates[j][0] == coordinates[j + 1][0]:

del coordinates[j]

else:

j += 1

return patterns

def developer\_sequences\_build():

sequences = [

['BD', 'E9', '1C'],

['BD', '7A', 'BD'],

['BD', '1C', 'BD', '55']

]

sequences\_rewards = [10, 20, 30]

return sequences, sequences\_rewards

def count\_point(all\_path, sequences, sequences\_rewards):

path\_reward = [0] \* len(all\_path)

for i, path in enumerate(all\_path):

used\_index = []

for j in range(len(path) - 1):

for k in range(j + 1, len(path)):

sequence = path[j:k + 1]

if sequence in sequences:

# Dapatkan indeks dari sequence dalam sequences

index = sequences.index(sequence)

# Jika indeks belum ada dalam used\_index, tambahkan nilai dari sequences\_rewards

if index not in used\_index:

path\_reward[i] += sequences\_rewards[index]

# Tambahkan indeks ke used\_index

used\_index.append(index)

return path\_reward

def path\_biggest\_point(path\_reward, all\_path):

biggest = max(path\_reward)

if biggest == 0:

return "none","none"

index = path\_reward.index(biggest)

path = all\_path[index]

return path, biggest, index

def sequences\_build():

number\_of\_sequences = int(input("Enter the number of sequences: "))

sequences = []

sequences\_rewards = []

for i in range(number\_of\_sequences):

sequence = input("Enter the sequence {}: ".format(i + 1)).strip().split()

while len(sequence) < 2:

print("Sequence harus terdiri dari minimal dua elemen. Coba lagi.")

sequence = input("Enter the sequence {}: ".format(i + 1)).strip().split()

sequences.append(sequence)

sequences\_reward = int(input("Enter the reward for sequence {}: ".format(i + 1)))

sequences\_rewards.append(sequences\_reward)

return sequences, sequences\_rewards

def extractPath(results, matrix):

all\_path = []

for i in range(len(results)):

coordinates = results[i]

values = [matrix[row][col] for row, col in coordinates]

all\_path.append(values)

return all\_path

def matrix\_build():

matrix\_dimensions = input("matrix\_width matrix\_height: ").split()

matrix\_width = int(matrix\_dimensions[1])

matrix\_height = int(matrix\_dimensions[0])

matrix = [[0 for i in range(matrix\_width)] for j in range(matrix\_height)]

matrix = []

print("Masukkan matriks (pisahkan elemen dengan spasi, baris baru untuk setiap baris):")

for \_ in range(matrix\_height):

row = input().strip().split()

while len(row) != matrix\_width:

print("Error: Jumlah elemen harus sama jumlah kolom!!!")

print("Ulangi input baris terkini: ")

row = input().strip().split()

matrix.append(row)

return matrix\_width, matrix\_height, matrix

def matrix\_build\_txt(matrix\_height, matrix\_width):

matrix = [[0 for i in range(matrix\_width)] for j in range(matrix\_height)]

matrix = []

print("Masukkan matriks (pisahkan elemen dengan spasi, baris baru untuk setiap baris):")

for \_ in range(matrix\_height):

row = input().strip().split()

while len(row) != matrix\_width:

print("Error: Jumlah elemen harus sama jumlah kolom!!!")

print("Ulangi input baris terkini: ")

row = input().strip().split()

matrix.append(row)

return matrix

def generate\_random\_sequences(token):

sequences = []

sequences\_rewards = []

jumlah\_sekuens = int(input("jumlah sekuens: "))

ukuran\_maksimal\_sekuens = int(input("ukuran maksimal sekuens: "))

for \_ in range(jumlah\_sekuens):

sequence\_length = random.randint(2, ukuran\_maksimal\_sekuens)

sequence = [random.choice(token) for \_ in range(sequence\_length)]

sequences.append(sequence)

sequences\_rewards.append(random.randint(1, 30))

return sequences, sequences\_rewards

def matrix\_generator(token, n, m):

matrix = []

for i in range(n):

row = []

for j in range(m):

row.append(random.choice(token))

matrix.append(row)

return matrix

def generate\_random\_sequence\_txt(token, jumlah\_sekuens, ukuran\_maksimal\_sekuens):

sequences = []

sequences\_rewards = []

for \_ in range(jumlah\_sekuens):

sequence\_length = random.randint(2, ukuran\_maksimal\_sekuens)

sequence = [random.choice(token) for \_ in range(sequence\_length)]

sequences.append(sequence)

sequences\_rewards.append(random.randint(1, 30))

return sequences, sequences\_rewards

def main():

file\_name = "input.txt"

try:

with open(file\_name, "r") as file:

# Inisialisasi counter baris

counter = 0

# Loop melalui setiap baris dalam file

for baris in file:

# Tambahkan 1 ke counter setiap kali loop

counter += 1

# Jika counter adalah 2, ini adalah baris kedua

if counter == 2:

# Split baris menjadi dua bagian

angka = baris.split()

# Pastikan terdapat dua angka

if len(angka) == 2:

# Periksa apakah keduanya bilangan angka

if angka[0].isdigit() and angka[1].isdigit():

method = "manual"

else:

method = "auto"

else:

method = "auto"

break

with open(file\_name, "r") as file:

# method = file.readline().strip()

if method == "manual":

buffer\_size = int(file.readline().strip())

matrix\_width, matrix\_height = map(int, file.readline().split())

matrix = [file.readline().split() for \_ in range(matrix\_height)]

max\_len = matrix\_width \* matrix\_height

sequences, sequences\_rewards = [], []

number\_of\_sequences = int(file.readline().strip())

for \_ in range(number\_of\_sequences):

sequence = file.readline().strip().split() # Membaca sequence

sequences.append(sequence)

sequence\_reward = int(file.readline().strip()) # Membaca reward untuk sequence

sequences\_rewards.append(sequence\_reward)

print("loading... too big buffer\_size can take a long time, please sabar \n")

start\_time = time.time()

patterns = generate\_patterns(matrix\_height, matrix\_width, max\_len, buffer\_size)

patterns = deletePath(patterns)

results = extractPath(patterns, matrix)

end\_time = time.time()

path\_reward = count\_point(results, sequences, sequences\_rewards)

path, biggest, path\_biggest\_index = path\_biggest\_point(path\_reward, results)

biggest\_path\_Windexes = patterns[path\_biggest\_index]

runtime = round((end\_time - start\_time) \* 1000)

print("biggest score = ", biggest)

# print(path)

print("path: ", end="")

for i in range(len(path)):

print(path[i], end=" ")

print("")

for point in biggest\_path\_Windexes:

print(f"{point[1]+1}, {point[0]+1}")

print("\n" + str(runtime) + " ms")

is\_save = input("Apakah ingin menyimpan hasil ke file? (y/n): ")

# simpan tanpa prompt

if is\_save == "y":

with open("../test/result.txt", "w") as file:

file.write(str(biggest) + "\n")

for i in range(len(path)):

file.write(str(path[i]) + " ")

file.write("\n")

for point in biggest\_path\_Windexes:

file.write(f"{point[1]+1}, {point[0]+1}\n")

file.write("\n" + str(runtime) + " ms")

print("Berhasil disimpan di result.txt")

elif method == "auto":

jumlah\_token\_unik = int(file.readline().strip())

token = file.readline().split() # misal bentuknya kek gini ['E9', 'B7', '8M']

while len(token) != jumlah\_token\_unik:

print("Error: Jumlah token harus", jumlah\_token\_unik, "!!!")

print("Ulangi input baris terkini: ")

token = input().strip().split()

buffer\_size = int(file.readline().strip())

matrix\_width, matrix\_height = map(int, file.readline().split())

print("loading... too big buffer\_size can take a long time, please sabar \n")

max\_len = matrix\_width \* matrix\_height

jumlah\_sekuens = int(file.readline().strip())

ukuran\_maksimal\_sekuens = int(file.readline().strip())

sequences, sequences\_rewards = generate\_random\_sequence\_txt(token, jumlah\_sekuens, ukuran\_maksimal\_sekuens)

matrix = matrix\_generator(token, matrix\_height, matrix\_width)

for \_ in range(len(matrix)):

print(matrix[\_])

print("sequences: ", sequences)

print("sequences\_rewards: ", sequences\_rewards)

start\_time = time.time()

patterns = generate\_patterns(matrix\_height, matrix\_width, max\_len, buffer\_size)

patterns = deletePath(patterns)

results = extractPath(patterns, matrix)

end\_time = time.time()

path\_reward = count\_point(results, sequences, sequences\_rewards)

# print("path\_reward:", path\_reward)

path, biggest, path\_biggest\_index = path\_biggest\_point(path\_reward, results)

biggest\_path\_Windexes = patterns[path\_biggest\_index]

runtime = round((end\_time - start\_time) \* 1000)

print("biggest score = ", biggest)

# print(path)

print("path: ", end="")

for i in range(len(path)):

print(path[i], end=" ")

print("")

for point in biggest\_path\_Windexes:

print(f"{point[1]+1}, {point[0]+1}")

print("\n" + str(runtime) + " ms")

is\_save = input("Apakah ingin menyimpan hasil ke file? (y/n): ")

if is\_save == "y":

with open("../test/result.txt", "w") as file:

file.write(str(biggest) + "\n")

for i in range(len(path)):

file.write(str(path[i]) + " ")

file.write("\n")

for point in biggest\_path\_Windexes:

file.write(f"{point[1]+1}, {point[0]+1}\n")

file.write("\n" + str(runtime) + " ms")

print("Berhasil disimpan di result.txt")

else:

print("Invalid method input")

main()

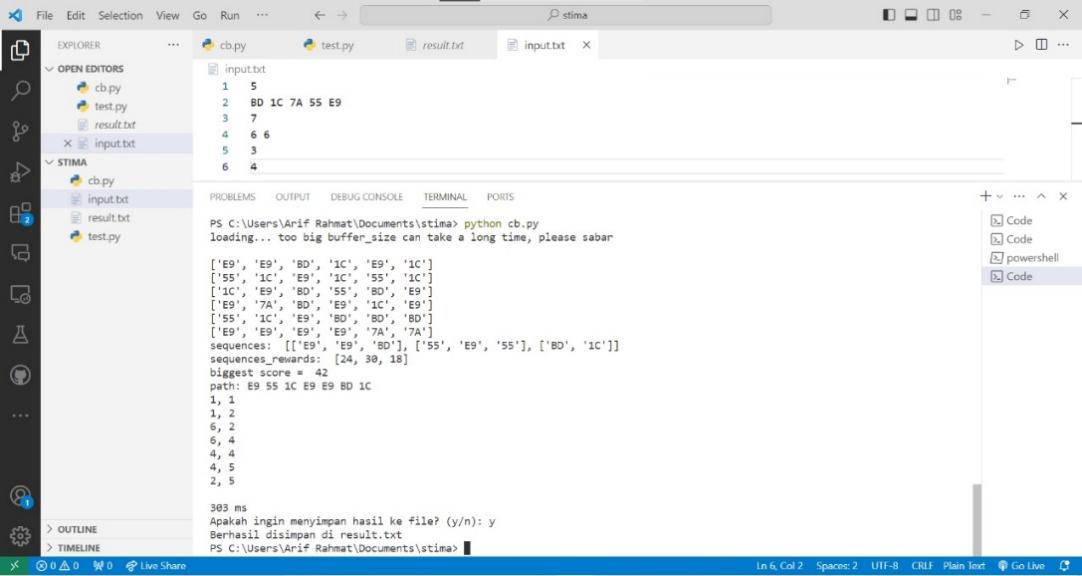
except FileNotFoundError:

print(f"File '{file\_name}' tidak ditemukan.")

if \_\_name\_\_ == "\_\_main\_\_":

main()

# Tangkapan layar

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A black and white image of a person

Description automatically generated

A screenshot of a computer

Description automatically generated

# Pranala Repository

<https://github.com/jimlynurarif/Tucil1_13522123>