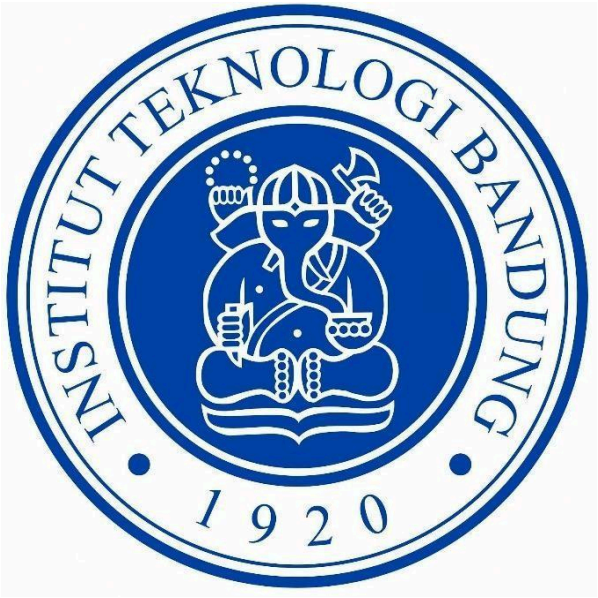
LAPORAN TUGAS KECIL

Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force

Disajikan untuk memenuhi salah satu tugas kecil Mata Kuliah IF2211 Strategi Algoritma yang diampu oleh:

Dr. Nur Ulfa Maulidevi, S.T., M.Sc.



Disusun oleh:

Nelsen Putra (13520130)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

A. Algoritma *Brute Force*

Algoritma *brute force* adalah penyelesaian suatu masalah dengan pendekatan yang sederhana, langsung, jelas, dan mudah dipahami. Dengan algoritma *brute force*, penyelesaian suatu masalah cenderung dilakukan dengan mencari dan meninjau semua kasus yang ada. Dengan demikian, algoritma *brute force* dapat dikatakan sebagai algoritma yang lempang (*straight-forward)*. Kelebihan dari algoritma ini adalah cukup mudah untuk dipahami dan dapat diterapkan untuk hampir semua permasalahan komputasi. Akan tetapi, kekurangan dari algoritma ini adalah tidak efisien karena membutuhkan langkah yang banyak dalam penyelesaiannya sehingga membutuhkan waktu yang relatif lama dibandingkan dengan kebanyakan algoritma lainnya.

Dalam Tugas Kecil 1 IF2211 Strategi Algoritma ini, mahasiswa diminta untuk mengimplementasikan algoritma *brute force* dalam menyelesaikan *Cyberpunk 2077 Breach Protocol*. Sebelum menerapkan algoritma *brute force*, program pertama – tama akan menerima dan membaca *input* dari *user* berupa sebuah *file* .txt. File tersebut berisi input data dalam bentuk format seperti berikut ini.

```
buffer_size
matrix_width matrix_height
matrix
number_of_sequences
sequences_1
sequences_1_reward
sequences_2
sequences_2_reward
…
sequences_n
sequences_n_reward
```

Program dimulai dengan mencari titik awal token. Pencarian dilakukan secara berurutan dari kolom pertama hingga kolom terakhir pada matriks dan dimulai dari baris paling atas hingga paling bawah. Titik awal ditentukan oleh token yang cocok dengan token pertama dari sequence yang memiliki reward tertinggi di antara sequence lainnya. Begitu titik awal ditemukan, program menetapkan titik start pada baris 0 dan kolom ke-j dari koordinat token yang cocok. Setelah menemukan titik awal, program akan mencari secara bergantian secara vertikal dan horizontal. Pencarian dimulai dari baris teratas ke baris terbawah, dengan kolom yang sama dengan titik awal.

Program mencari token pertama dari sequence yang memiliki reward tertinggi. Setelah menemukan token pertama, program mencari token berikutnya dalam sequence tersebut. Jika program tidak berhasil menemukan token berikutnya dalam pencarian vertikal dan horizontal bergantian, maka pencarian akan beralih ke sequence dengan reward tertinggi kedua, dan seterusnya. Program terus mencari hingga semua token dalam sequence ditemukan, dan mengupdate indeks i dan j ke posisi token terakhir dalam sequence. Pencarian dilakukan selama nilai init atau count masih lebih kecil dari buffer_size.

B. *Source Code* Program

Source code program *Cyberpunk 2077 Breach Protocol Solver* ditulis dalam bahasa Python dengan isi kode sebagai berikut.

1. File randomizer.py

```
import random

# Matrix Randomizer
def randomizeMatrix(m, n):
    matrix = []
    for _ in range(n):
        row = [random.choice(['7A', '55', 'E9', '1C', 'BD']) for _ in
range(m)]
        matrix.append(row)
    return matrix

# Sequences and Rewards Randomizer
def randomizeSAR(numOfSequences):
    sequencesAndRewards = {}
    for _ in range(numOfSequences):
        sequenceLenght = random.randint(1, 5)
        sequence = ' '.join(random.choices(['7A', '55', 'E9', '1C',
'BD'], k=sequenceLenght))
        reward = random.randint(10, 50)
        sequencesAndRewards[sequence] = reward
    return sequencesAndRewards

def randomize():
    bufferSize = random.randint(5, 10)
    m = random.randint(4, 8)
    n = random.randint(4, 8)
    numOfSequences = random.randint(1, 5)

    matrix = randomizeMatrix(m, n)

    sequencesAndRewards = randomizeSAR(numOfSequences)

    return bufferSize, m, n, matrix, sequencesAndRewards
```

2. File main.py

```
import time
```

```
import pyfiglet
from colorama import Fore, Style
from randomizer import randomize

def readFile(filename):
 with open(filename, 'r') as file:
   lines = file.readlines()
   bufferSize = int(lines[0])
   m, n = map(int, lines[1].split()) # Matrix weight, matrix height
   matrix = [line.split() for line in lines[2:(2 + n)]]
   numOfSequence = int(lines[2 + n])
   sequencesAndRewards = {}
   for i in range(numOfSequence):
     sequence = lines[3 + n + (i * 2)].split()
     reward = int(lines[4 + n + (i * 2)])
     sequencesAndRewards[" ".join(sequence)] = reward

   return bufferSize, m, n, matrix, sequencesAndRewards

def horizontalSearch(token, i, j, matrix, m):
   for col in range(m):
       if (matrix[i][col] == token):
           return True, (i, col)

   return False, (i, j)

def verticalSearch(token, i, j, matrix, n):
   for row in range(n):
       if (matrix[row][j] == token):
           return True, (row, j)

   return False, (i, j)

def sequenceSearch(init, i, j, sequence, matrix, n, m):
 tokenAdded = sequence[0].split()
 reward = sequence[1]
 coordinateAdded = []
 bufferAdded = len(sequence[0].split())
 foundSequence = True

 if (init < 2): # First search
```

```
    length = range(len(sequence[0].split()))
  else: #
    length = range(2, len(sequence[0].split()))

  if (init % 2 != 0): # Odd init, search horizontally
    for t in length:
      if (t % 2 != 0):

        if (init > 1):
          found, newRowIndex = verticalSearch(sequence[0].split()[t], i,
j, matrix, n)
          if found: i = newRowIndex[0]
        else:
          found, newRowIndex = horizontalSearch(sequence[0].split()[t],
i, j, matrix, m)
          if found: j = newRowIndex[1]

        if found:
          # i = newRowIndex[0]
          # j = newRowIndex[1]
          coordinateAdded.append((i, j))
          continue
        else:
          foundSequence = False
          break

      elif (t % 2 == 0):

        if (init > 1):
          found, newColIndex = horizontalSearch(sequence[0].split()[t],
i, j, matrix, m)
          if found:
            j = newColIndex[1]
        else:
          found, newColIndex = verticalSearch(sequence[0].split()[t], i,
j, matrix, n)
          if found:
            i = newColIndex[0]

        if found:
          # i = newColIndex[0]
```

```
        # j = newColIndex[1]
        coordinateAdded.append((i, j))
        continue
      else:
        foundSequence = False
        break

 elif (init % 2 == 0): # Even init, search vertically
   for t in length:
     if (t % 2 != 0):

       if (init > 1):
         found, newColIndex = horizontalSearch(sequence[0].split()[t],
i, j, matrix, m)
         if found: j = newColIndex[1]
       else:
         found, newColIndex = verticalSearch(sequence[0].split()[t], i,
j, matrix, n)
         if found: i = newColIndex[0]

       if found:
         # i = newColIndex[0]
         # j = newColIndex[1]
         coordinateAdded.append((i, j))
         continue
       else:
         foundSequence = False
         break

     elif (t % 2 == 0):

       if (init > 1):
         found, newRowIndex = verticalSearch(sequence[0].split()[t], i,
j, matrix, n)
         if found:
           i = newRowIndex[0]
       else:
         found, newRowIndex = horizontalSearch(sequence[0].split()[t],
i, j, matrix, m)
         if found:
           j = newRowIndex[1]
```

```
      if found:
        # i = newRowIndex[0]
        # j = newRowIndex[1]
        coordinateAdded.append((i, j))
        continue
      else:
        foundSequence = False
        break

 if foundSequence:
   found = True
   x = i # New i
   y = j # New j
   return found, tokenAdded, reward, coordinateAdded, bufferAdded, x, y
 else:
   found = False
   return found, [], 0, [], 0, i, j

def findMaxReward(bufferSize, m, n, matrix, sortedSequences):
 maxBuffer = ""
 coordinates = []
 maxReward = 0
 execTime = 0
 startTime = time.time()
 init = 0
 i, j = 0, 0

 while (init < bufferSize):
   print("\nInit:", init)
   found = False

   if (init == 0): # Find starting point by column and by row
     startSequence = sortedSequences[0][0].split()[0]
     startPoint = (0,0)
     for j in range(m):
       found = False
       for i in range(n):
         if matrix[i][j] == startSequence: # Find first token from
sequence with maximum reward
           startPoint = (0, j)
```

```
            maxBuffer += matrix[0][j]
            coordinates.append((0, j))
            found = True
            break
        if found:
          break

    print("Starting point: ", startPoint)
    print("First token: ", matrix[startPoint[0]][startPoint[1]])
    print("Starting reward: ", maxReward)

    i = startPoint[0]
    j = startPoint[1]

  else:
    if (init < 2): # First search

      # Odd init, search vertically
      if (init % 2 != 0):
        print("Search vertically")
        for row in range(0, n):
          if (row == i):
            continue
          else:
            print("\n>> Checked token :", matrix[row][j], "on
coordinate ", (row, j))
            found = False
            for sequenceInRow in range(len(sortedSequences)): #
Checking sequences one by one
              print("Checked sequence: ",
sortedSequences[sequenceInRow][0])

              if ((init < 2) and (matrix[row][j] ==
sortedSequences[sequenceInRow][0].split()[0])):
                found, tokenAdded, reward, coordinateAdded,
bufferAdded, x, y = sequenceSearch(init, row, j,
sortedSequences[sequenceInRow], matrix, n, m)

                if found:
                  coordinates.append((row, j))
                  i = x
```

```
                    j = y

                    for m in range(bufferAdded):
                      maxBuffer += " " + tokenAdded[m]
                    for n in range(len(coordinateAdded)):
                      coordinates.append(coordinateAdded[n])

                    maxReward += reward
                    init += (bufferAdded - 1)
                    break
                  else:
                    continue

            if found:
              print("Sequence has been found!")
              print("Reward received: ",
sortedSequences[sequenceInRow][1])
              print("Current token coordinate: ", (i, j))
              print("Current token: ", matrix[i][j])
              break
            else:
              print("Sequence does not match, searching for the next
token..")

        # Even init, search horizontally
        else:
          print("Search horizontally")
          for col in range(0, m):
            if (col == j):
              continue
            else:
              print("\n>> Checked token:", matrix[i][col], "on
coordinate ", (i, col))
              found = False
              for seq_in_col in range(len(sortedSequences)): # Checking
sequence one by one
                print("Checked sequence:",
sortedSequences[seq_in_col][0])

                if ((init < 2) and (matrix[i][col] ==
sortedSequences[seq_in_col][0].split()[0])):
```

```
                found, tokenAdded, reward, coordinateAdded,
bufferAdded, x, y = sequenceSearch(init, i, col,
sortedSequences[sequenceInRow], matrix, n, m)

                if found:
                  coordinates.append((i, col))
                  i = x
                  j = y

                  for m in range(tokenAdded):
                    maxBuffer += " " + tokenAdded[m]
                  for n in range(len(coordinateAdded)):
                    coordinates.append(coordinateAdded[n])

                  maxReward += reward
                  init += (bufferAdded - 1)
                  break
                else:
                  continue

          if found:
            print("Sequence has been found!")
            print("Reward received: ", sortedSequences[seq_in_col][1])
            print("Current token coordinate: ", (i, j))
            print("Current token: ", matrix[i][j])
            break
          else:
            print("Sequence does not match, searching for the next
token..\n")

    else: # Second search and etc.

      for seq in range(len(sortedSequences)):
        # Odd init, search vertically
        if (init % 2 != 0):
          print("Search horizontally")
          for row in range(0, n):
            if (row == i):
              continue
            elif matrix[row][j] == sortedSequences[seq][0].split()[0]:
              print("\n>> Checked token: ", matrix[row][j], "on
```

```
coordinate ", (row, j))
                print("Checked sequence: ", sortedSequences[seq][0])

                found, tokenAdded, reward, coordinateAdded, bufferAdded,
x, y = sequenceSearch(init, row, j, sortedSequences[seq], matrix, n, m)

                if found:
                  coordinates.append((row, j))
                  i = x
                  j = y

                  for m in range(1, bufferAdded):
                    maxBuffer += " " + tokenAdded[m]
                  for n in range(1, len(coordinateAdded)):
                    coordinates.append(coordinateAdded[n])

                  maxReward += reward
                  init += (bufferAdded - 1)
                  break
                else:
                  continue

            if found:
              print("Sequence has been found!")
              print("Reward received: ", sortedSequences[seq][1])
              print("Current token coordinate: ", (i, j))
              print("Current token: ", matrix[i][j])
              break
            else:
              print("Sequence does not match, searching for the next
token..\n")

          if found: break

          # Even init, search horizontally
          elif init % 2 == 0:
            print("Search horizontally")
            for col in range(0, m):
              if (col == j):
                continue
              elif matrix[i][col] == sortedSequences[seq][0].split()[1]:
```

```
                print("\n>> Checked token: ", matrix[i][col], "on
coordinate ", (i, col))
                print("Checked sequence: ", sortedSequences[seq][0])

                found, tokenAdded, reward, coordinateAdded, bufferAdded,
x, y = sequenceSearch(init, i, col, sortedSequences[seq], matrix, n, m)

                if found:
                  coordinates.append((i, col))
                  i = x
                  j = y

                  for m in range(1, bufferAdded):
                    maxBuffer += " " + tokenAdded[m]
                  for n in range(1, len(coordinateAdded)):
                    coordinates.append(coordinateAdded[n])

                  maxReward += reward
                  init += (bufferAdded - 1)
                  break
                else:
                  continue

            if found:
              print("Sequence has been found!")
              print("Reward received: ", sortedSequences[seq][1])
              print("Current token coordinate: ", (i, j))
              print("Current token: ", matrix[i][j])
              break
            else:
              print("Sequence does not match, searching for the next
token..\n")

          if found: break

    # Increment init
    init += 1

  end_time = time.time()
  execTime = (end_time - startTime) * 1000
  return maxBuffer, coordinates, maxReward, execTime
```

```
def displayGrid(matrix):
 for row in matrix:
   print(" ".join(row))

def saveToFile(filename, maxReward, maxBuffer, coordinates, matrix,
execTime, overload):
 writtenCoordinates = set()

 with open(filename, 'w') as file:
   file.write("Maximum reward: {}\n".format(maxReward))
   file.write("Buffer: {}\n".format(maxBuffer))
   file.write("Coordinates of each token: \n")
   for coordinate in coordinates:
     if coordinate not in writtenCoordinates:
         file.write("{} {}\n".format(coordinate,
matrix[coordinate[0]][coordinate[1]]))
         writtenCoordinates.add(coordinate)
   file.write("Execution time: {:.3f} ms\n".format(execTime))
   if overload:
     file.write("Buffer exceeds capacity!\n")

def main():
 print(Fore.CYAN +"\n<< ============ TUGAS KECIL 1 IF2211 STRATEGI
ALGORITMA ============ >>")
 print("\n")
 print("Nama       : Nelsen Putra")
 print("NIM        : 13520130")
 print("Kelas      : 02")
 print("\n")
 title = pyfiglet.figlet_format("Cyberpunk 2077 Breach Protocol",
font="big")
 print(title)
  print(Fore.YELLOW +"********** CHOOSING INPUT METHOD **********\n" +
Style.RESET_ALL)
 print("1. By inserting file name")
 print("2. Auto input by randomizer")
 option = input("\nChoose method (1/2): ")

 if (option == "1"):
   filename = input("\nInsert file name: ")
```

```
    bufferSize, m, n, matrix, sequencesAndRewards = readFile(filename)
    sortedSequences = sorted(sequencesAndRewards.items(), key=lambda x:
x[1], reverse=True)
  elif (option == "2"):
    bufferSize, m, n, matrix, sequencesAndRewards = randomize()
    sortedSequences = sorted(sequencesAndRewards.items(), key=lambda x:
x[1], reverse=True)

  print(Fore.GREEN + "\n\n>>>>>>>>>> INPUT <<<<<<<<<<\n" +
Style.RESET_ALL)
  print(Style.RESET_ALL + "Buffer Size:", bufferSize)
  print("Matrix width:", m)
  print("Matrix height:", n)
  print("\nMatrix:")
  displayGrid(matrix)
  print("\nSequences and rewards:")
  for sequence, reward in sequencesAndRewards.items():
    print(f"{sequence}: Reward {reward}")

  print(Fore.YELLOW + "\n***** BRUTE FORCE ALGORITHM *****" +
Style.RESET_ALL)
  maxBuffer, coordinates, maxReward, execTime =
findMaxReward(bufferSize, m, n, matrix, sortedSequences)

  print(Fore.GREEN + "\n>>>>>>>>>> OUTPUT <<<<<<<<<<\n" +
Style.RESET_ALL)
  overload = False
  print("Maximum reward: ", maxReward)
  print("Buffer: ", maxBuffer)
  print("Coordinates of each token: ")
  printed_coordinates = set()
  for coordinate in coordinates:
      if (coordinate not in printed_coordinates):
          print(coordinate, matrix[coordinate[0]][coordinate[1]])
          printed_coordinates.add(coordinate)
   if len(maxBuffer.split()) > bufferSize:
    print(Fore.RED + "\nBuffer exceeds capacity! (",
len(maxBuffer.split()), ">", bufferSize, ")" + Style.RESET_ALL)
    overload = True

  print(Fore.BLUE + "\nExecution time:", "{:.3f}".format(execTime),
```

```
"ms\n" + Style.RESET_ALL)


 print(">> Do you want to keep the solution? (Y/N)")
 save = input()
 if (save == "Y"):
   filename = input("\nInsert file name to save (.txt): ")
   print(Fore.GREEN + "\nThe solution has been successfully saved!\n" +
Style.RESET_ALL)
   saveToFile(filename, maxReward, maxBuffer, coordinates, matrix,
execTime, overload)
 else:
   print(Fore.CYAN + "\nThank you!\n" + Style.RESET_ALL)
 if __name__ == "__main__":
 main()
```

C. *Screenshot* Program (Input dan Output)

Berikut ini adalah hasil uji coba kode program berupa input yang digunakan dan output yang dihasilkan pada program.

1. Uji Coba 1

```
Input:
5
4 7
E9 BD 7A 55
1C 55 55 55
1C 1C 1C E9
E9 7A 1C BD
1C E9 1C 7A
55 E9 7A BD
55 1C 7A 7A
1
1C BD
40

Output:
Maximum reward:  40
Buffer:  E9 BD
Coordinates of each token:
(0, 0) E9
(0, 1) BD

Execution time: 0.294 ms
```

2. Uji Coba 2

```
Input:
8
5 4
1C BD 7A 7A E9
1C 55 BD E9 E9
BD BD 1C 7A E9
```

```
7A 55 E9 E9 7A
5
1C E9 1C BD BD
49
E9 55 BD
23
55 1C E9
49
E9 E9 BD E9
42
55 E9 7A 1C 55
21

Output:
Maximum reward:  49
Buffer:  1C E9 1C BD BD
Coordinates of each token:
(0, 0) 1C
(1, 0) 1C
(2, 0) BD

Execution time: 0.612 ms
```

3. Uji Coba 3

```
Input:
8
7 4
55 1C 55 55 1C 1C 1C
55 E9 1C 1C 1C 55 E9
55 55 7A 1C BD 7A 55
55 BD 1C E9 7A BD 55
5
7A BD 1C
23
BD 55
22
1C E9 1C
40
1C
35
7A 1C BD
27

Output:
Maximum reward:  22
Buffer:  1C BD 55
Coordinates of each token:
(0, 1) 1C
(3, 1) BD
(3, 0) 55

Execution time: 0.622 ms
```

4. Uji Coba 4

```
Input:
6
7 4
E9 BD E9 BD 1C 1C 1C
```

```
1C E9 7A E9 55 BD 7A
E9 BD 1C BD 55 7A E9
E9 BD E9 BD E9 E9 55
2
7A E9 E9
16
E9
46


Output:
Maximum reward:  46
Buffer:  E9 E9
Coordinates of each token:
(0, 0) E9
(2, 0) E9

Execution time: 0.514 ms
```

5. Uji Coba 5

```
Input:
5
4 7
1C 7A 1C 1C
1C E9 55 BD
E9 E9 E9 BD
BD 1C E9 1C
7A E9 7A 1C
7A 7A BD 1C
55 55 1C 1C
1
BD 1C
33

Output:
Maximum reward:  33
Buffer:  1C BD 1C
Coordinates of each token:
(0, 0) 1C
(3, 0) BD
(3, 1) 1C

Execution time: 0.251 ms
```

6. Uji Coba 6

```
Input:
10
7 5
55 7A 7A 1C 55 1C 55
BD 7A 55 BD E9 7A 1C
BD BD 55 1C 1C E9 BD
7A BD 1C 7A 1C E9 E9
7A 7A 7A 1C BD 7A E9
1
BD 55
12

Output:
```

```
Maximum reward:  12
Buffer:  55 BD 55
Coordinates of each token:
(0, 0) 55
(1, 0) BD
(1, 2) 55

Execution time: 0.323 ms
```

D. Pranala

Berikut ini adalah alamat *repository* yang berisi kode program Tugas Kecil 1 IF2211 Strategi Algoritma milik Nelsen Putra (13520130):
https://github.com/nelsenputra/Tucil1_13520130

E. *Checklist*

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

Program belum memiliki GUI