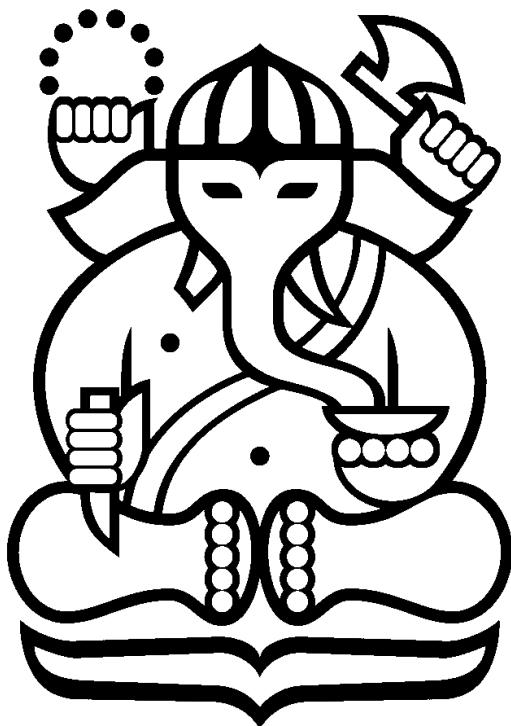


**TUGAS KECIL 1 IF2211 STRATEGI ALGORITMA  
PENYELESAIAN CYBERPUNK 2077 BREACH PROTOCOL DENGAN  
ALGORITMA BRUTE FORCE**



**Disusun oleh:  
Shazya Audrea Taufik  
13522063**

**Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2024**

## **BAB I**

### **ALGORITMA BRUTE FORCE**

Algoritma *brute force* adalah pendekatan yang lempang (*straightforward*) untuk memecahkan suatu persoalan. Biasanya algoritma *brute force* 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*).

Langkah penggunaan algoritma *brute force* pada program yang telah saya buat adalah dengan:

1. Pada awal, token diawali dengan token pada ujung kiri atas.
2. Kemudian dilakukan iterasi secara rekursif, apabila langkah terkini adalah langkah yang ganjil (`current_buffer % 2 = 1`), maka iterasi dilakukan secara vertikal. Apabila genap, maka iterasi dilakukan secara horizontal. Pembagian kasus ini (ganjil dan genap) dilakukan agar urutan gerakan terjadi secara vertikal-horizontal-vertikal-horizontal-...
3. Pencarian setiap token pada path dilakukan dari atas ke bawah (vertikal) dan dari kiri ke kanan (horizontal).
4. Iterasi rekursif terus diulang hingga path terkini telah mencapai limit buffer atau sudah memiliki seluruh sekuens (memperoleh sekuens optimal). Ketika sudah mencapai limit atau sudah memiliki seluruh sekuens, program akan menambahkan path ini kedalam suatu array.
5. Selanjutnya, iterasi akan dilakukan terus-menerus terhadap semua path yang bermula setiap dari semua token pada baris paling atas.
6. Setiap *path* kemudian akan dioptimasi dengan menghapus token akhir yang apabila dihapus, tidak akan mengubah nilai reward.

Langkah tersebut termasuk penggunaan algoritma *brute force* karena mencari seluruh kemungkinan *path* sehingga diperoleh *path* yang memperoleh nilai hadiah tertinggi dengan token paling sedikit (optimal). Semua inputan dari program ini dianggap benar dan sesuai.

## BAB II

### IMPLEMENTASI ALGORITMA DALAM BAHASA PYTHON

#### 2.1 Fungsi main.py

Fungsi	Deskripsi
def read_file(file_name)	Fungsi ini membaca file dan mengembalikan ukuran buffer, ukuran matriks, matriks, jumlah sekuens, sekuens, dan hadiah.
def generate()	Fungsi ini melakukan pengacakan terhadap matriks dan sekuens. Fungsi ini mengembalikan ukuran buffer, ukuran matriks, jumlah sekuens, sekuens, dan hadiah.
def generate_path(current_buffer, index, current_path, current_coor, passed, buffer_size, paths, matrix_size, matrix, coordinate)	Fungsi ini mencari semua kemungkinan jalan yang dimulai dari token-token pada baris pertama. Pencarian ini dilakukan dengan menggunakan algoritma brute force.
def all_sequence(sequences, path)	Fungsi ini mencari apakah suatu <i>path</i> atau kemungkinan sudah memiliki semua sekuens.
def find_sequence(sequences, paths)	Fungsi ini mencari nilai hadiah dari setiap kemungkinan jalan dan menambahkannya ke dalam suatu <i>array</i> .
def find_optimal(maxreward, paths, coors)	Fungsi ini mengembalikan nilai hadiah, jalan, dan koordinat yang memiliki nilai hadiah tertinggi.
def max_length_sequence(sequences)	Fungsi ini mencari panjang maksimum dari suatu sekuens.
def optimize(paths, sequences, initreward, coordinates)	Fungsi ini mencari jalan optimal dari setiap jalan dengan menghapus token akhir yang tidak mengubah nilai hadiah maksimum.
def save_solution(file_name, max_reward, max_sequence, max_coordinates, execution_time)	Fungsi ini menyimpan hasil output ke dalam file txt.

#### 2.2 Alur Program

1. Tampilan awal program.

2. Program menerima inputan berupa pilihan 1, 2, atau 3. Jika memilih 1, maka program akan menerima inputan file. Jika memilih 2, maka program akan membuat matriks dan sekvensi secara acak. Jika memilih 3, maka program akan berhenti. Program akan terus menerima pilihan hingga ada pilihan 3.
3. Jika memilih 2, program akan memunculkan matriks dan sekvensi beserta hadiahnya. Kemudian program akan memunculkan output baik ketika memilih 1 ataupun 2.
4. Akan muncul prompt untuk menyimpan file. Jika memilih y, maka akan muncul input nama file dan kemudian di save. Jika tidak, maka file tidak akan disimpan.
5. Akan muncul lagi tampilan awal. Hingga pengguna memilih 3.

```
Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 1
=====
Enter the file name with the game matrix: test3.txt
=====
20
7A BD 7A 1C
1, 1
1, 3
5, 3
5, 1
2, 1
2, 2

15.826940536499023 ms

Do you want to save this solution? (y/n): y
Input file name: save3.txt
Your file has been saved!
Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 3
=====
Thank You!
=====
Shazya Audrea Taufik
13522063
```

## BAB III

### SOURCE CODE

#### 3.1 Source Code

```
import random
import time
import os
def read_file(file_name):
    with open(file_name, 'r') as file:
        lines = file.readlines()
        buffer_size = int(lines[0])
        matrix_size = tuple(map(int, lines[1].split()))
        matrix = []
        for i in range(matrix_size[1]):
            matrix.append(lines[i+2].split())
        number_of_sequences = int(lines[2+matrix_size[1]])
        sequences = []
        rewards = []
        for i in range(3+matrix_size[1], len(lines), 2):
            sequences.append((lines[i].replace(" ", ""))
                .rstrip())
            rewards.append(int(lines[i+1]))
    return buffer_size, matrix_size, matrix,
number_of_sequences, sequences, rewards
def generate():
    unique_token_number = int(input("Input number of unique
token: "))
    tokens = list(map(str, input("Input unique token(s): ")
        .split()))
    buffer_size = int(input("Input buffer size: "))
    matrix_size = tuple(map(int, input("Input matrix size (col
row): ").split()))
    number_of_sequences = int(input("Input number of sequence:
"))
    max_sequence_size = int(input("Input maximum size of
sequence: "))
    matrix = [[None for j in range(matrix_size[0])] for i in
range(matrix_size[1])]
    for i in range(matrix_size[1]):
        for j in range(matrix_size[0]):
            matrix[i][j] = random.choice(tokens)
```

```

rewards = []
sequences = []
for i in range(number_of_sequences):
    length = random.randint(2, max_sequence_size)
    sequence = ''
    for j in range(length):
        sequence = sequence+random.choice(tokens)
    sequences.append(sequence)
    rewards.append(random.randint(0,100))
return buffer_size, matrix_size, matrix,
number_of_sequences, sequences, rewards
def generate_path(current_buffer, index, current_path,
current_coor, passed, buffer_size, paths, matrix_size, matrix,
coordinate):
    if(current_buffer == buffer_size) or
(all_sequence(sequences, current_path) == True):
        paths.append(current_path)
        coordinate.append(current_coor)
        return None
    if (current_buffer == 0):
        for j in range(matrix_size[0]):
            generate_path(1, j, matrix[0][j],
current_coor+([j+1, 1]), [[j, 0]], buffer_size, paths,
matrix_size, matrix, coordinate)
    elif(current_buffer % 2 == 1):
        for i in range(matrix_size[1]):
            seen = False
            for coordinates in passed:
                if coordinates[0] == index and coordinates[1]
== i:
                    seen = True
                    break
            if (seen == True):
                continue
            passed.append([index, i])
            generate_path(current_buffer+1, i,
current_path+matrix[i][index], current_coor+([index+1, i+1]),
passed, buffer_size, paths, matrix_size, matrix, coordinate)
            passed.pop()

```

```

else:
    for j in range(matrix_size[0]):
        seen = False
        for coordinates in passed:
            if coordinates[1] == index and coordinates[0]
== j:
                seen = True
                break
        if (seen == True):
            continue
        passed.append([j, index])
            generate_path(current_buffer+1, j,
current_path+matrix[index][j], current_coort([j+1, index+1]),
passed, buffer_size, paths, matrix_size, matrix, coordinate)
            passed.pop()
    return paths, coordinate
def all_sequence(sequences, path):
    foundAll = 0
    for sequence in sequences:
        if sequence in path:
            foundAll += 1
    if foundAll == len(sequences):
        return True
    else:
        return False
def find_sequence(sequences, paths):
    initreward = []
    for path in paths:
        reward = 0
        for i in range(len(sequences)):
            occur = path.count(sequences[i])
            if occur >= 1:
                reward += rewards[i]
        initreward.append(reward)
    return initreward
def find_optimal(maxreward, paths, coors):
    max_num = maxreward[0]
    max_index = 0
    for i in range(len(maxreward)):
        if maxreward[i] > max_num or (maxreward[i] == max_num

```

```

and len(paths[i]) < len(paths[max_index])):
    max_num = maxreward[i]
    max_index = i
if max_num == 0:
    paths[max_index] = ''
    coors[max_index] = ''
return max_num, paths[max_index], coors[max_index]
def max_length_sequence(sequences):
    maxlen = -1
    for sequence in sequences:
        if(len(sequence) > maxlen):
            maxlen = len(sequence)
    return maxlen
def optimize(paths, sequences, initreward, coordinates):
    op_path = []
    op_coor = []
    for k in range(len(paths)):
        path = paths[k]
        coor = coordinates[k]
        if (len(path) >= max_length_sequence(sequences)+2) and
init_reward[k] != 0:
            temp = path[:-2]
            tempcoor = coor[:-2]
            reward = 0
            for i in range(len(sequences)):
                occur = temp.count(sequences[i])
                if occur >= 1:
                    reward += rewards[i]
            if reward == initreward[k]:
                path = temp
                coor = tempcoor
            else:
                op_path.append(path)
                op_coor.append(coor)
        while(reward == initreward[k]):
            temp = path[:-2]
            tempcoor = coor[:-2]
            reward = 0
            for i in range(len(sequences)):
                occur = temp.count(sequences[i])

```

```

        if occur >= 1:
            reward += rewards[i]
        if reward == initreward[k]:
            path = temp
            coor = tempcoor
        else:
            op_path.append(path)
            op_coor.append(coor)
    else:
        op_path.append(path)
        op_coor.append(coor)
    return op_path, op_coor
def save_solution(file_name, max_reward, max_sequence,
max_coordinates, execution_time):
    with open(file_name, 'w') as file:
        if (max_num == 0):
            file.write("Tidak ada solusi."+'\n')
            file.write(str(max_reward) + '\n')
            file.write("\n")
            file.write(execution_time+" ms")
        else:
            file.write(str(max_reward))
            file.write("\n")
            for i in range(len(max_sequence)):
                if i == len(max_sequence) - 1:
                    file.write(max_sequence[i])
                else:
                    if i % 2 == 1:
                        file.write(max_sequence[i] + ' ')
                    else:
                        file.write(max_sequence[i])
            file.write("\n")
            for i in range(len(max_coordinates)):
                if i % 2 == 0:
                    file.write(str(max_coordinates[i]) + ', ')
                else:
                    file.write(str(max_coordinates[i]))
                    file.write("\n")
            file.write("\n")
            file.write(execution_time+" ms")

```

```

#main program
print("Cyberpunk 2077 Breach Protocol Solution")
print("=====")
print("1. Input Text File")
print("2. Generate Matrix and Sequence")
print("3. Exit")
print("=====")
choice = int(input("Choice: "))
print("=====")
stop_program = False
while(1 <= choice <= 3 and stop_program == False):
    if (1 <= choice <= 2):
        if (choice == 1):
            file_name = input("Enter the file name with the
game matrix: ")
            relative_path = "test/"+file_name
            while not os.path.isfile(relative_path):
                file_name = input("File not found, reenter your
file name: ")
                relative_path = "test/"+file_name
                buffer_size, matrix_size, matrix,
number_of_sequences, sequences, rewards =
read_file(relative_path)
            else:
                buffer_size, matrix_size, matrix,
number_of_sequences, sequences, rewards = generate()
            print("=====")
            for i in range(matrix_size[1]):
                for j in range(matrix_size[0]):
                    if (j == matrix_size[0]-1):
                        print(matrix[i][j], end='\n')
                    else:
                        print(matrix[i][j], end=' ')
            for j in range(len(sequences)):
                sequence = sequences[j]
                for i in range(len(sequence)):
                    if i == len(sequence) - 1:
                        print(sequence[i])
                        print(str(rewards[j])+"\n")
                    else:

```

```

        if i % 2 == 1:
            print(sequence[i], end=' ')
        else:
            print(sequence[i], end=' ')
    start_time = time.time()
    paths, coordinate = generate_path(0, 0, [], [], [], buffer_size, [], matrix_size, matrix, [])
    init_reward = find_sequence(sequences, paths)
    op_path, op_coor = optimize(paths, sequences, init_reward, coordinate)
    max_num, path, coor = find_optimal(init_reward, op_path, op_coor)
    print("====")
    if (max_num == 0):
        print("Tidak ada solusi.")
    print(max_num)
    for i in range(len(path)):
        if i == len(path) - 1:
            print(path[i])
        else:
            if i % 2 == 1:
                print(path[i] + ' ', end=' ')
            else:
                print(path[i], end=' ')
    for i in range(len(coor)):
        if i % 2 == 0:
            print(str(coor[i]) + ', ', end=' ')
        else:
            print(str(coor[i]), end='\n')
    end_time = time.time()
    print("\n")
    print((end_time - start_time) * 1000, "ms\n")
    save_file = input("Do you want to save this solution? (y/n): ")
    if (save_file == 'y'):
        file_name_save = input("Input file name: ")
        relative_path_save = "test/" + file_name_save
        while os.path.isfile(relative_path_save):
            file_name_save = input("File name is taken, reenter your file name: ")

```

```

        relative_path_save = "test/" + file_name_save
        save_solution(relative_path_save, max_num, path,
coor, str((end_time - start_time) * 1000))
        print("Your file has been saved!")
print("====")
print("Cyberpunk 2077 Breach Protocol Solution")
print("====")
print("1. Input Text File")
print("2. Generate Matrix and Sequence")
print("3. Exit")
print("====")
choice = int(input("Choice: "))
print("====")
else:
    print("====")
    print("                    Thank You!                ")
    print("====")
    print("                    Shazya Audrea Taufik      ")
    print("                    13522063                  ")
    print("====")
    stop_program = True

```

### 3.2 Source Code (Tampilan Layar)

```

src > main.py > @ read_file
1 import random
2 import time
3 import os
4
5 def read_file(file_name):
6     with open(file_name, 'r') as file:
7         lines = file.readlines()
8         buffer_size = int(lines[0])
9         matrix_size = tuple(map(int, lines[1].split()))
10        matrix = []
11        for i in range(matrix_size[1]):
12            matrix.append(lines[i+2].split())
13        number_of_sequences = int(lines[2+matrix_size[1]])
14        sequences = []
15        rewards = []
16        for i in range(3+matrix_size[1], len(lines), 2):
17            sequences.append((lines[i].replace(" ", "")).rstrip())
18            rewards.append(int(lines[i+1]))
19        return buffer_size, matrix_size, matrix, number_of_sequences, sequences, rewards
20
21 def generate():
22     unique_token_number = int(input("Input number of unique token: "))
23     tokens = list(map(str, input("Input unique tokens: ").split()))
24     buffer_size = int(input("Input buffer size: "))
25     matrix_size = tuple(map(int, input("Input matrix size (col row): ").split()))
26     number_of_sequences = int(input("Input number of sequence: "))
27     max_sequence_size = int(input("Input maximum size of sequence: "))
28
29     matrix = [None for j in range(matrix_size[0]) for i in range(matrix_size[1])]
30     for i in range(matrix_size[1]):
31         for j in range(matrix_size[0]):
32             matrix[i][j] = random.choice(tokens)
33
34     rewards = []
35     sequences = []
36
37     for i in range(number_of_sequences):
38         length = random.randint(2, max_sequence_size)
39         sequence = ''
40         for j in range(length):
41             sequence = sequence + random.choice(tokens)
42         sequences.append(sequence)
43         rewards.append(random.randint(0, 100))
44
45     return buffer_size, matrix_size, matrix, number_of_sequences, sequences, rewards

```

```

47     def generate_path(current_buffer, index, current_path, current_coor, passed, buffer_size, paths, matrix_size, matrix, coordinate):
48         if(current_buffer == buffer_size) or (all_sequence(sequences, current_path) == True):
49             paths.append(current_path)
50             coordinate.append(current_coor)
51             return None
52         if (current_buffer == 0):
53             for j in range(matrix_size[0]):
54                 generate_path(1, j, matrix[0][j], current_coor+[j+1, 1], [[j, 0]], buffer_size, paths, matrix_size, matrix, coordinate)
55         elif(current_buffer % 2 == 1):
56             for i in range(matrix_size[1]):
57                 seen = False
58                 for coordinates in passed:
59                     if coordinates[0] == index and coordinates[1] == i:
60                         seen = True
61                         break
62                 if (seen == True):
63                     continue
64                 passed.append([index, i])
65                 generate_path(current_buffer+1, i, current_path+matrix[i][index], current_coor+[index+1, i+1], passed, buffer_size, paths, matrix_size, matrix, coordinate)
66                 passed.pop()
67             else:
68                 for j in range(matrix_size[0]):
69                     seen = False
70                     for coordinates in passed:
71                         if coordinates[1] == index and coordinates[0] == j:
72                             seen = True
73                             break
74                         if (seen == True):
75                             continue
76                         passed.append([j, index])
77                         generate_path(current_buffer+1, j, current_path+matrix[index][j], current_coor+[j+1, index+1], passed, buffer_size, paths, matrix_size, matrix, coordinate)
78                         passed.pop()
79             return paths, coordinate
80
81     def all_sequence(sequences, path):
82         foundAll = 0
83         for sequence in sequences:
84             if sequence in path:
85                 foundAll += 1
86         if foundAll == len(sequences):
87             return True
88         else:
89             return False
90
91     def find_sequence(sequences, paths):
92         initreward = []
93         for path in paths:
94             reward = 0
95             for i in range(len(sequences)):
96                 occur = path.count(sequences[i])
97                 if occur >= 1:
98                     reward += rewards[i]
99             initreward.append(reward)
100        return initreward
101
102    def find_optimal(maxreward, paths, coors):
103        max_num = maxreward[0]
104        max_index = 0
105        for i in range(len(maxreward)):
106            if maxreward[i] > max_num or (maxreward[i] == max_num and len(paths[i]) < len(paths[max_index])):
107                max_num = maxreward[i]
108                max_index = i
109        if max_num == 0:
110            paths[max_index] = ''
111            coors[max_index] = ''
112        return max_num, paths[max_index], coors[max_index]
113
114    def max_length_sequence(sequences):
115        maxlen = -1
116        for sequence in sequences:
117            if len(sequence) > maxlen:
118                maxlen = len(sequence)
119        return maxlen
120
121    def optimize(paths, sequences, initreward, coordinates):
122        op_path = []
123        op_coor = []
124        for k in range(len(paths)):
125            path = paths[k]
126            coor = coordinates[k]
127            if (len(path) >= max_length_sequence(sequences)+2) and init_reward[k] != 0:
128                temp = path[-2]
129                tempcoor = coor[-2]
130                reward = 0
131                for i in range(len(sequences)):
132                    occur = temp.count(sequences[i])
133                    if occur >= 1:
134                        reward += rewards[i]
135                if reward == initreward[k]:
136                    Path = temp
137                    coor = tempcoor
138                else:
139                    op_path.append(path)
140                    op_coor.append(coor)
141            while(reward == initreward[k]):
142                temp = path[-2]
143                tempcoor = coor[-2]
144                reward = 0
145                for i in range(len(sequences)):
146                    occur = temp.count(sequences[i])
147                    if occur >= 1:
148                        reward += rewards[i]
149                if reward == initreward[k]:
150                    path = temp
151                    coor = tempcoor
152                else:
153                    op_path.append(path)
154                    op_coor.append(coor)
155            else:
156                op_path.append(path)
157                op_coor.append(coor)
158        return op_path, op_coor

```

```

160     def save_solution(file_name, max_reward, max_sequence, max_coordinates, execution_time):
161         with open(file_name, 'w') as file:
162             if (max_num == 0):
163                 file.write("Tidak ada solusi."+'\n')
164                 file.write(str(max_reward) + '\n')
165                 file.write("\n")
166                 file.write(execution_time+" ms")
167             else:
168                 file.write(str(max_reward))
169                 file.write("\n")
170                 for i in range(len(max_sequence)):
171                     if i == len(max_sequence) - 1:
172                         file.write(max_sequence[i])
173                     else:
174                         if i % 2 == 1:
175                             file.write(max_sequence[i] + ' ')
176                         else:
177                             file.write(max_sequence[i])
178                 file.write("\n")
179                 for i in range(len(max_coordinates)):
180                     if i % 2 == 0:
181                         file.write(str(max_coordinates[i]) + ', ')
182                     else:
183                         file.write(str(max_coordinates[i]))
184                         file.write("\n")
185                 file.write("\n")
186                 file.write(execution_time+" ms")
187
188     #main program
189     print("Cyberpunk 2077 Breach Protocol Solution")
190     print("=====")
191     print("1. Input Text File")
192     print("2. Generate Matrix and Sequence")
193     print("3. Exit")
194     print("=====")
195     choice = int(input("Choice: "))
196     print("=====")
197     stop_program = False
198     while(1 <= choice <= 3 and stop_program == False):
199         if (1 <= choice <= 2):
200             if (choice == 1):
201                 file_name = input("Enter the file name with the game matrix: ")
202                 relative_path = "test/" + file_name
203                 while not os.path.isfile(relative_path):
204                     file_name = input("File not found, reenter your file name: ")
205                     relative_path = "test/" + file_name
206                     buffer_size, matrix_size, matrix, number_of_sequences, sequences, rewards = read_file(relative_path)
207             else:
208                 buffer_size, matrix_size, matrix, number_of_sequences, sequences, rewards = generate()
209                 print("=====")
210                 for i in range(matrix_size[1]):
211                     for j in range(matrix_size[0]):
212                         if (j == matrix_size[0]-1):
213                             print(matrix[i][j], end='\n')
214                         else:
215                             print(matrix[i][j], end=' ')
216
217                 for j in range(len(sequences)):
218                     sequence = sequences[j]
219                     for i in range(len(sequence)):
220                         if i == len(sequence) - 1:
221                             print(sequence[i])
222                             print(str(rewards[j])) + "\n"
223                         else:
224                             if i % 2 == 1:
225                                 print(sequence[i], end=' ')
226                             else:
227                                 print(sequence[i], end=' ')
228
229     start_time = time.time()
230     paths, coordinate = generate_path(0, 0, [], [], [], buffer_size, [], matrix_size, matrix, [])
231     init_reward = find_sequences(sequences, paths)
232     op_path, op_coor = optimize(paths, sequences, init_reward, coordinate)
233     max_num, path, coor = find_optimal(init_reward, op_path, op_coor)
234     print("=====")
235     if (max_num == 0):
236         print("Tidak ada solusi.")
237     print(max_num)
238     for i in range(len(path)):
239         if i == len(path) - 1:
240             print(path[i])
241         else:
242             if i % 2 == 1:
243                 print(path[i] + ' ', end='')
244             else:
245                 print(path[i], end='')
246     for i in range(len(coor)):
247         if i % 2 == 0:
248             print(str(coor[i]) + ', ', end='')
249         else:
250             print(str(coor[i]), end='\n')
251     end_time = time.time()
252     print("\n")
253     print(end_time - start_time)*1000, "ms\n")
254     save_file = input("Do you want to save this solution? (y/n): ")
255     if(save_file == 'y'):
256         file_name_save = input("Input file name: ")
257         relative_path_save = "test/" + file_name_save
258         while os.path.isfile(relative_path_save):
259             file_name_save = input("File name is taken, reenter your file name: ")
260             relative_path_save = "test/" + file_name_save
261         save_solution(relative_path_save, max_num, path, coor, str((end_time - start_time)*1000))
262         print("Your file has been saved!")
263     print("=====")
264     print("Cyberpunk 2077 Breach Protocol Solution")
265     print("=====")
266     print("1. Input Text File")
267     print("2. Generate Matrix and Sequence")
268     print("3. Exit")
269     print("=====")
270     choice = int(input("Choice: "))
271     print("=====")
272     else:
273         print("=====")
274         print("Thank You!")
275         print("=====")
276         print("Shazy Audrea Taufik")
277         print("13522063")
278         print("=====")
279     stop_program = True

```

## BAB IV

### HASIL PENGUJIAN

#### 4.1 Test Case 1

Text File:

```
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
```

Hasil:

```
Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 1
=====
Enter the file name with the game matrix: test1.txt
=====
50
7A BD 7A BD 1C BD 55
1, 1
1, 4
3, 4
3, 5
6, 5
6, 3
1, 3

111.58490180969238 ms

Do you want to save this solution? (y/n): y
Input file name: save1.txt
Your file has been saved!
=====
Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 3
=====
Thank You!
=====
Shazya Audrea Taufik
13522063
=====
```

#### 4.2 Test Case 2

Text File:

```
2
3 2
7A 55 E9
55 7A BD
```

```
1  
55 BD  
15
```

Hasil:

```
Cyberpunk 2077 Breach Protocol Solution  
=====  
1. Input Text File  
2. Generate Matrix and Sequence  
3. Exit  
=====  
Choice: 1  
=====  
Enter the file name with the game matrix: test2.txt  
=====  
Tidak ada solusi.  
0  
  
0.11491775512695312 ms  
  
Do you want to save this solution? (y/n): y  
Input file name: save2.txt  
Your file has been saved!
```

### 4.3 Test Case 3

Text File:

```
7  
5 4  
7A BD 55 E9 1C  
BD 1C FF E9 E9  
BD 55 E9 E9 7A  
FF E9 1C E9 7A  
2  
7A BD 7A 1C  
20  
55 1C FF FF  
10
```

Hasil:

```
Cyberpunk 2077 Breach Protocol Solution  
=====  
1. Input Text File  
2. Generate Matrix and Sequence  
3. Exit  
=====  
Choice: 1  
=====  
Enter the file name with the game matrix: test3.txt  
=====  
20  
7A BD 7A 1C  
1, 1  
1, 3  
5, 3  
5, 1  
  
16.93272590637207 ms  
  
Do you want to save this solution? (y/n): y  
Input file name: save3.txt  
Your file has been saved!
```

### 4.4 Test Case 4

Hasil:

```

Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit

Choice: 2
=====
Input number of unique token: 4
Input unique token(s): AB BC CD DE
Input buffer size: 6
Input matrix size (col row): 5 6
Input number of sequence: 4
Input maximum size of sequence: 3
=====
CD AB BC BC BC
AB AB BC CD CD
BC DE DE CD BC
BC BC CD CD CD
DE DE BC DE DE
BC DE CD CD CD
BC CD
10
AB CD DE
38
AB DE
41
CD CD
52
=====
103
AB DE BC CD CD
2, 1
2, 3
5, 3
5, 2
4, 2
12360.860109329224 ms

Do you want to save this solution? (y/n): y
Input file name: save4.txt
Your file has been saved!

```

## 4.5 Test Case 5

Hasil:

```

Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit

Choice: 2
=====
Input number of unique token: 4
Input unique token(s): E9 1C 55 7A
Input buffer size: 5
Input matrix size (col row): 5 4
Input number of sequence: 3
Input maximum size of sequence: 5
=====
E9 55 7A 7A 55
55 7A E9 1C 7A
7A 55 7A E9 55
1C 7A E9 1C 7A
E9 E9
34
E9 E9 E9 7A
28
E9 55 1C E9 E9
25
=====
34
E9 55 E9 E9 1C
1, 1
1, 2
3, 2
3, 4
1, 4
123728332519531 ms

Do you want to save this solution? (y/n): y
Input file name: save5.txt
Your file has been saved!

```

## 4.6 Test Case 6

Hasil:

```

Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 2
=====
Input number of unique token: 3
Input unique token(s): AB AC AD
Input buffer size: 3
Input matrix size (col row): 2 3
Input number of sequence: 4
Input maximum size of sequence: 5
=====
AD AD
AD AC
AC AB
AB AB AC AC AC
54
AB AB
74
AC AD AB
36
AD AB AD AB
11
=====
Tidak ada solusi.
0

0.5428791046142578 ms
Do you want to save this solution? (y/n): y
Input file name: save6.txt
Your file has been saved!

```

#### 4.7 Test Case 7

Text file:

```

15
5 5
BD 1C 1C 55 E9
55 E9 1C 55 BD
1C BD 7A 7A 7A
E9 7A E9 7A 7A
55 BD E9 1C 7A
10
55 55
-53
1C E9 E9
83
7A 7A
-18
55 55
-96
55 7A
-49
BD BD
-59
7A E9
16
55 E9
-92
BD E9
-20

```

1C BD  
-67

Hasil:

```
Cyberpunk 2077 Breach Protocol Solution
=====
1. Input Text File
2. Generate Matrix and Sequence
3. Exit
=====
Choice: 1
=====
Enter the file name with the game matrix: test7.txt
=====
99
1C E9 E9 1C 7A E9
3, 1
3, 4
1, 4
1, 3
3, 3
3, 5

276572.0200538635 ms

Do you want to save this solution? (y/n): y
Input file name: save7.txt
Your file has been saved!
```

## **BAB V**

### **LAMPIRAN**

#### **5.1 Repository**

[https://github.com/zyaaa-aaa/Tucil1\\_13522063.git](https://github.com/zyaaa-aaa/Tucil1_13522063.git)

#### **5.2 Tabel Kelengkapan**

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"/>