Laporan Tugas Kecil 1:
Penerapan Algoritma Brute Force Pada Permainan Breach Protocol

Mata Kuliah IF2211 Strategi Algoritma

Disusun oleh :

Atqiya Haydar Luqman
13522163

Sekolah Teknik Elektro dan Informatika
Program Studi Teknik Informatika
Institut Teknologi Bandung
2024

Daftar Isi

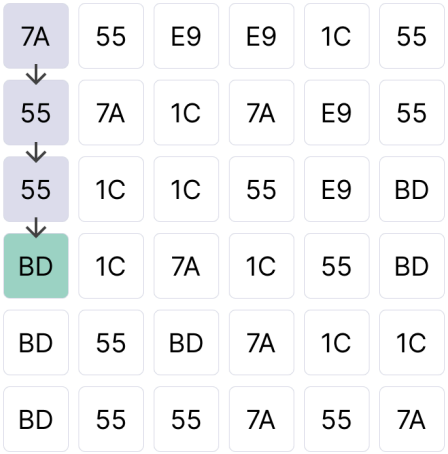
I. Algoritma Brute Force

Program dimulai dengan mencari titik awal token. Pencarian dilakukan secara berurutan dari kolom pertama hingga kolom terakhir pada matriks, 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/count masih lebih kecil dari buffer_size.

Berikut adalah contoh ilustrasi penerapan algoritma brute force dengan menggunakan masukkan Uji Coba 1:

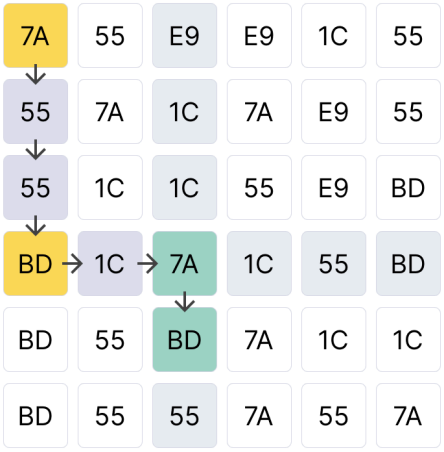
1. Program akan mencari titik start dengan mencari secara perbaris kemudian perkolom (for j in range(), for i in range()).



Program berhasil menemukan token pertama dari *sequence* yang paling besar nilainya. Titik start akan dimulai pada token yang berada di posisi indeks baris 0 dan indeks kolom token pertama dari *sequence* dengan nilai terbesar.

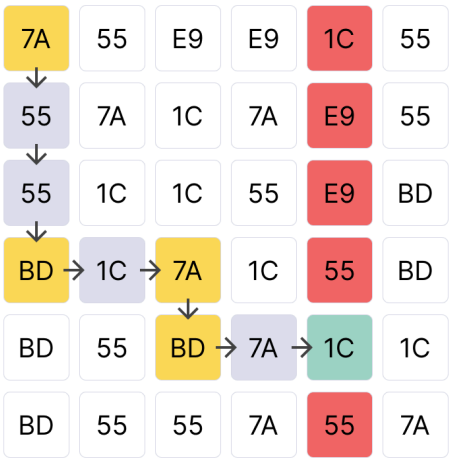
2. Program akan mencari semua token yang berada pada *sequence* tersebut secara horizontal dan vertikal bergantian. (BD 1C BD 55).

Karena tidak ada token ‘BD’ pada kolom yang sama dengan ‘1C’, program melanjutkan dengan pencarian *sequence* yang memiliki reward kedua paling tinggi. (BD 7A BD).

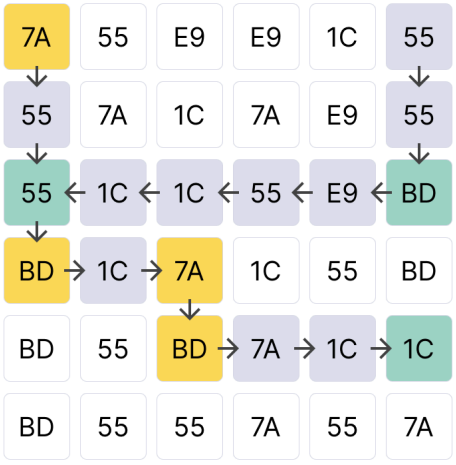


Program berhasil mencari semua token yang berada pada *sequence* BD 7A BD. Indeks posisi diperbarui menjadi koordinat token ‘BD’.

3. Program melanjutkan mencari *sequence* selanjutnya, dimulai dari *sequence* yang token pertamanya sama dengan token saat ini (‘BD’) dan memiliki reward terbesar, yaitu BD 1C BD 55.

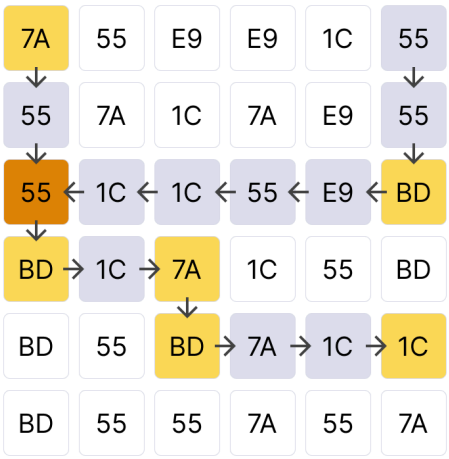


Karena pencarian token ‘BD’ tidak ditemukan pada kolom yang sama dengan token ‘1C’, program akan melanjutkan pencarian token ‘1C’ di kolom yang berbeda.



Program berhasil mendapatkan semua token dari *sequence* BD 1C BD 55.

4. Program berhasil mencari buffer dengan reward terbesar, dengan setiap pencarian horizontal dimulai dari token paling kiri dan pencarian vertikal dimulai dari token paling atas.



II. Source Program dalam Bahasa Python

```python
# Nama : Atqiya Haydar Luqman
# NIM : 13522163
# Kelas : K3

import time
import pyfiglet
from colorama import Fore, Style

from Randomize import randomize

def read_data_from_file(filename):
  with open(filename, 'r') as file:
    lines = file.readlines()
    buffer_size = int(lines[0])
    matrix_width, matrix_height = map(int, lines[1].split())
    matrix = [line.split() for line in lines[2:2+matrix_height]]
    number_of_sequence = int(lines[2+matrix_height])
    sequences_and_rewards = {}
    for i in range(number_of_sequence):
      sequence = lines[3+matrix_height+i*2].split()
      reward = int(lines[4+matrix_height+i*2])
      sequences_and_rewards[" ".join(sequence)] = reward

    return buffer_size, matrix_width, matrix_height, matrix, sequences_and_rewards
```

```python
def search_horizontal(token, i, j, matrix, matrix_width):
    for col in range(matrix_width):
        if matrix[i][col] == token:
            return True, (i, col)

    return False, (i, j)

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

    return False, (i, j)
```

```python
def search_sequence(init, i, j, sequence, matrix, matrix_height, matrix_width):
  token_added = sequence[0].split()
  reward = sequence[1]
  coordinates_added = []
  buffer_added = len(sequence[0].split())
  found_sequence = True

  if init < 2: # Pencarian pertama
    length = range(len(sequence[0].split()))
  else: # Pencarian kedua dan seterusnya
    length = range(2, len(sequence[0].split()))

```

```python
if init % 2 != 0: # Init ganjil mencari horizontal
    for t in length:
      if t % 2 != 0:

        if init > 1:
          found, new_row_index = search_vertical(sequence[0].split()[t], i, j, matrix, matrix_height)
          if found: i = new_row_index[0]
        else:
          found, new_row_index = search_horizontal(sequence[0].split()[t], i, j, matrix, matrix_width)
          if found: j = new_row_index[1]

        if found:
          # i = new_row_index[0]
          # j = new_row_index[1]
          coordinates_added.append((i, j))
          continue
        else:
          found_sequence = False
          break

      elif t % 2 == 0:

        if init > 1:
          found, new_col_index = search_horizontal(sequence[0].split()[t], i, j, matrix, matrix_width)
          if found: j = new_col_index[1]
        else:
          found, new_col_index = search_vertical(sequence[0].split()[t], i, j, matrix, matrix_height)
          if found: i = new_col_index[0]

        if found:
          # i = new_col_index[0]
          # j = new_col_index[1]
          coordinates_added.append((i, j))
          continue
        else:
          found_sequence = False
          break
```

```
elif init % 2 == 0: # Init genap mencari vertikal
    for t in length:
      if t % 2 != 0:

        if init > 1:
          found, new_col_index = search_horizontal(sequence[0].split()[t], i, j, matrix, matrix_width)
          if found: j = new_col_index[1]
        else:
          found, new_col_index = search_vertical(sequence[0].split()[t], i, j, matrix, matrix_height)
          if found: i = new_col_index[0]

        if found:
          # i = new_col_index[0]
          # j = new_col_index[1]
          coordinates_added.append((i, j))
          continue
        else:
          found_sequence = False
          break

      elif t % 2 == 0:

        if init > 1:
          found, new_row_index = search_vertical(sequence[0].split()[t], i, j, matrix, matrix_height)
          if found: i = new_row_index[0]
        else:
          found, new_row_index = search_horizontal(sequence[0].split()[t], i, j, matrix, matrix_width)
          if found: j = new_row_index[1]

        if found:
          # i = new_row_index[0]
          # j = new_row_index[1]
          coordinates_added.append((i, j))
          continue
        else:
          found_sequence = False
          break
```

```
if found_sequence:
    found = True
    new_i = i
    new_j = j
    return found, token_added, reward, coordinates_added, buffer_added, new_i, new_j
  else:
    found = False
    return found, [], 0, [], 0, i, j
```

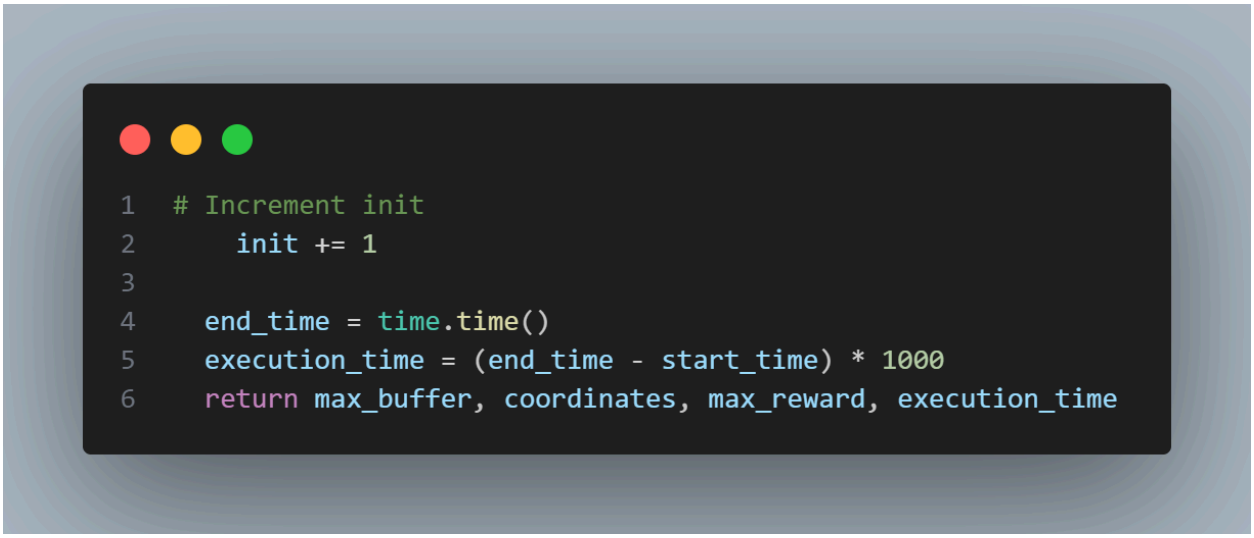
```python
def find_maximum_reward(buffer_size, matrix_width, matrix_height, matrix, sorted_sequences):
  max_buffer = ""
  coordinates = []
  max_reward = 0
  execution_time = 0
  start_time = time.time()
  init = 0
  i, j = 0, 0

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

    if init == 0: # Mencari titik mulai secara perkolom kemudian perbaris
      start_sequence = sorted_sequences[0][0].split()[0]
      start_point = (0,0)
      for j in range(matrix_width):
        found = False
        for i in range(matrix_height):
          if matrix[i][j] == start_sequence: # Mencari token pertama dari sequence dengan reward tertinggi
            start_point = (0, j)
            max_buffer += matrix[0][j]
            coordinates.append((0, j))
            found = True
            break
        if found:
          break

      print("Start Point:", start_point)
      print("Token pertama: ", matrix[start_point[0]][start_point[1]])
      print("Reward mula-mula: ", max_reward)

      i = start_point[0]
      j = start_point[1]
```

```
else:
      if init < 2: # Pencarian pertama

        # Init ganjil, mencari secara vertikal
        if init % 2 != 0:
          print("Mencari secara vertikal")
          for row in range(0, matrix_height):
            if row == i:
              continue
            else:
              print("\n>> Token yang dicheck:", matrix[row][j], "pada koordinat", (row, j))
              found = False
              for seq_in_row in range(len(sorted_sequences)): # Sequence dicheck secara bergiliran
                print("Sequence yang dicheck:", sorted_sequences[seq_in_row][0])

                if init < 2 and matrix[row][j] == sorted_sequences[seq_in_row][0].split()[0]:
                  found, token_added, reward, coordinates_added, buffer_added, new_i, new_j = search_sequence(init, row, j, sorted_sequences[seq_in_row], matrix, matrix_height, matrix_width)

                  if found:
                    coordinates.append((row, j))
                    i = new_i
                    j = new_j

                    for m in range(buffer_added):
                      max_buffer += " " + token_added[m]
                    for n in range(len(coordinates_added)):
                      coordinates.append(coordinates_added[n])

                    max_reward += reward
                    init += (buffer_added - 1)
                    break
                  else:
                    continue

            if found:
              print("Sequence berhasil ditemukan!")
              print("Mendapatkan reward:", sorted_sequences[seq_in_row][1])
              print("Koordinat token saat ini:", (i, j))
              print("Token saat ini:", matrix[i][j])
              break
            else:
              print("Sequence tidak ada yang cocok, pencarian token selanjutnya..")
```

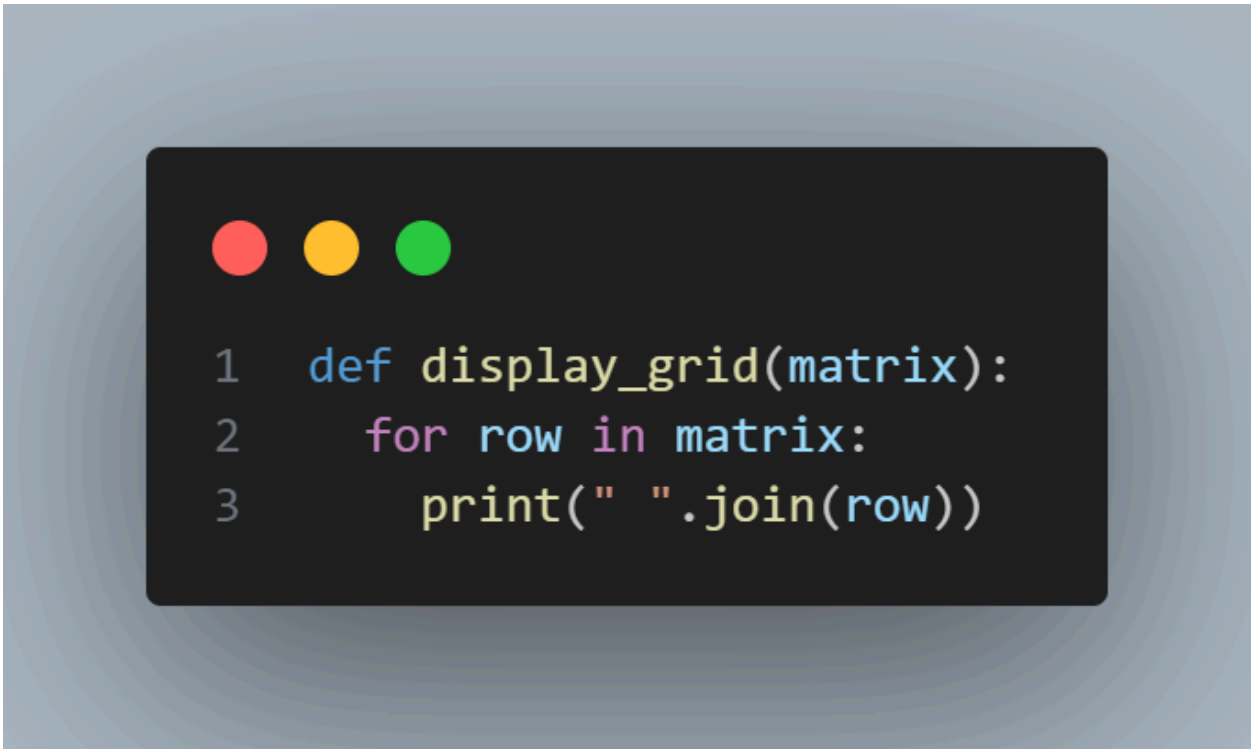
```
# Init genap, mencari secara horizontal
        else:
          print("Mencari secara horizontal")
          for col in range(0, matrix_width):
            if col == j:
              continue
            else:
              print("\n>> Token yang dicheck:", matrix[i][col], "pada koordinat", (i, col))
              found = False
              for seq_in_col in range(len(sorted_sequences)): # Sequence dicheck secara bergiliran
                print("Sequence yang dicheck:", sorted_sequences[seq_in_col][0])

                if init < 2 and matrix[i][col] == sorted_sequences[seq_in_col][0].split()[0]:
                  found, token_added, reward, coordinates_added, buffer_added, new_i, new_j = search_sequence(init, i, col, sorted_sequences[seq_in_row], matrix, matrix_height, matrix_width)

                  if found:
                    coordinates.append((i, col))
                    i = new_i
                    j = new_j

                    for m in range(token_added):
                      max_buffer += " " + token_added[m]
                    for n in range(len(coordinates_added)):
                      coordinates.append(coordinates_added[n])

                    max_reward += reward
                    init += (buffer_added - 1)
                    break
                  else:
                    continue

            if found:
              print("Sequence berhasil ditemukan!")
              print("Mendapatkan reward:", sorted_sequences[seq_in_col][1])
              print("Koordinat token saat ini:", (i, j))
              print("Token saat ini:", matrix[i][j])
              break
            else:
              print("Sequence tidak ada yang cocok, pencarian token selanjutnya..\n")
```

```
else: # Pencarian kedua dan seterusnya

        for seq in range(len(sorted_sequences)):
          # Init ganjil, mencari secara vertikal
          if init % 2 != 0:
            print("Mencari secara vertikal")
            for row in range(0, matrix_height):
              if row == i:
                continue
              elif matrix[row][j] == sorted_sequences[seq][0].split()[0]:
                print("\n>> Token yang dicheck:", matrix[row][j], "pada koordinat", (row, j))
                print("Sequence yang dicheck:", sorted_sequences[seq][0])

                found, token_added, reward, coordinates_added, buffer_added, new_i, new_j = search_sequence(init, row, j, sorted_sequences[seq], matrix, matrix_height, matrix_width)

                if found:
                  coordinates.append((row, j))
                  i = new_i
                  j = new_j

                  for m in range(1, buffer_added):
                    max_buffer += " " + token_added[m]
                  for n in range(1, len(coordinates_added)):
                    coordinates.append(coordinates_added[n])

                  max_reward += reward
                  init += (buffer_added - 1)
                  break
                else:
                  continue

            if found:
              print("Sequence berhasil ditemukan!")
              print("Mendapatkan reward:", sorted_sequences[seq][1])
              print("Koordinat token saat ini:", (i, j))
              print("Token saat ini:", matrix[i][j])
              break
            else:
              print("Sequence tidak ada yang cocok, pencarian token selanjutnya..\n")

          if found: break
```

```
# Init genap, mencari secara horizontal
          elif init % 2 == 0:
            print("Mencari secara horizontal")
            for col in range(0, matrix_width):
              if col == j:
                continue
              elif matrix[i][col] == sorted_sequences[seq][0].split()[1]:
                print("\n>> Token yang dicheck:", matrix[i][col], "pada koordinat", (i, col))
                print("Sequence yang dicheck:", sorted_sequences[seq][0])

                found, token_added, reward, coordinates_added, buffer_added, new_i, new_j = search_sequence(init, i, col, sorted_sequences[seq], matrix, matrix_height, matrix_width)

                if found:
                  coordinates.append((i, col))
                  i = new_i
                  j = new_j

                  for m in range(1, buffer_added):
                    max_buffer += " " + token_added[m]
                  for n in range(1, len(coordinates_added)):
                    coordinates.append(coordinates_added[n])

                  max_reward += reward
                  init += (buffer_added - 1)
                  break
                else:
                  continue

            if found:
              print("Sequence berhasil ditemukan!")
              print("Mendapatkan reward:", sorted_sequences[seq][1])
              print("Koordinat token saat ini:", (i, j))
              print("Token saat ini:", matrix[i][j])
              break
            else:
              print("Sequence tidak ada yang cocok, pencarian token selanjutnya..\n")

          if found: break
```



```
# Increment init
    init += 1

  end_time = time.time()
  execution_time = (end_time - start_time) * 1000
  return max_buffer, coordinates, max_reward, execution_time
```



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

```python
def save_output_to_file(filename, max_reward, max_buffer, coordinates, matrix, execution_time, over_capacity):
  written_coordinates = set()

  with open(filename, 'w') as file:
    file.write("Reward maksimal: {}\n".format(max_reward))
    file.write("Isi buffer: {}\n".format(max_buffer))
    file.write("Koordinat setiap token: \n")
    for coordinate in coordinates:
      if coordinate not in written_coordinates:
          file.write("{} {}\n".format(coordinate, matrix[coordinate[0]][coordinate[1]]))
          written_coordinates.add(coordinate)
    file.write("Waktu eksekusi: {:.3f} ms\n".format(execution_time))
    if over_capacity:
      file.write("Buffer melebihi kapasitas!\n")
```

```python
def main():
  print(Fore.BLUE +"\n<< ============ TUCIL 1 ============ >>")
  title = pyfiglet.figlet_format("Breach Protocol", font="slant")
  print(title)

  print("<< ========= PILIH METODE MASUKKAN ========= >>\n" + Style.RESET_ALL)
  print("1. Masukkan nama file")
  print("2. Masukkan secara otomatis")
  option = input("\nPilihan: ")

  if option == "1":
    filename = input("\nMasukkan nama file: ")
    buffer_size, matrix_width, matrix_height, matrix, sequences_and_rewards = read_data_from_file(filename)
    sorted_sequences = sorted(sequences_and_rewards.items(), key=lambda x: x[1], reverse=True)
  elif option == "2":
    buffer_size, matrix_width, matrix_height, matrix, sequences_and_rewards = randomize()
    sorted_sequences = sorted(sequences_and_rewards.items(), key=lambda x: x[1], reverse=True)

  print(Fore.CYAN + "\n<< ============ HASIL INPUT ============ >>\n" + Style.RESET_ALL)
  print(Style.RESET_ALL + "Buffer Size:", buffer_size)
  print("Matrix Width:", matrix_width)
  print("Matrix Height:", matrix_height)
  print("\nMatrix:")
  display_grid(matrix)
  print("\nSequences and Rewards:")
  for sequence, reward in sequences_and_rewards.items():
    print(f"{sequence}: Reward {reward}")

  print(Fore.MAGENTA + "\n<< ========= ALGORITMA BRUTEFORCE ========= >>" + Style.RESET_ALL)
  max_buffer, coordinates, max_reward, execution_time = find_maximum_reward(buffer_size, matrix_width, matrix_height, matrix, sorted_sequences)

  print(Fore.GREEN + "\n<< ============ HASIL OUTPUT ============ >>\n" + Style.RESET_ALL)
  over_capacity = False
  print("Reward maksimal: ", max_reward)
  print("Isi buffer: ", max_buffer)
  print("Koordinat setiap 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(max_buffer.split()) > buffer_size:
    print(Fore.RED + "\nBuffer melebihi kapasitas! (", len(max_buffer.split()), ">", buffer_size, ")" + Style.RESET_ALL)
    over_capacity = True

  print(Fore.BLUE + "\nWaktu eksekusi:", "{:.3f}".format(execution_time), "ms\n" + Style.RESET_ALL)

  print(">> Apakah ingin menyimpan solusi? (y/n)")
  save = input()
  if save == "y":
    filename = input("\nMasukkan nama file (dalam txt): ")
    print(Fore.GREEN + "\nBerhasil menyimpan solusi pada file output.txt!\n" + Style.RESET_ALL)
    save_output_to_file(filename, max_reward, max_buffer, coordinates, matrix, execution_time, over_capacity)
  else:
    print(Fore.GREEN + "\nTerima kasih telah menggunakan program ini!\n" + Style.RESET_ALL)

if __name__ == "__main__":
  main()
```

III. Tangkapan Layar Hasil Uji Coba Program

1. Uji Coba 1

Masukkan :

```
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
```

Keluaran :

```
<< ============ HASIL OUTPUT ============ >>

Reward maksimal:  50
Isi buffer:  7A BD 7A BD 1C BD 55
Koordinat setiap token:
(0, 0) 7A
(3, 0) BD
(3, 2) 7A
(4, 2) BD
(4, 5) 1C
(2, 0) 55

Waktu eksekusi: 6.555 ms
```

2. Uji Coba 2

Masukkan :

```
3
6 6
55 1C E9 55 7A E9
55 1C E9 1C 7A BD
```

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

Keluaran :

```
<< ============ HASIL OUTPUT ============ >>

Reward maksimal:  30
Isi buffer:  55 BD 1C BD 55
Koordinat setiap token:
(0, 0) 55
(5, 0) BD
(5, 1) 1C
(3, 1) BD
(3, 0) 55

Buffer melebihi kapasitas! ( 5 > 3 )

Waktu eksekusi: 6.529 ms
```

3. Uji Coba 3

Masukkan :

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

```
7A 55
15
```

Keluaran :

```
<< ============ HASIL OUTPUT ============ >>

Reward maksimal:  25
Isi buffer:  1C 7A 55 BD
Koordinat setiap token:
(0, 0) 1C
(2, 0) 7A
(2, 2) 55
(1, 2) 1C

Waktu eksekusi: 8.167 ms
```

4. Uji Coba 4

Masukkan :

```
5
5 5
1C BD 7A E9 55
55 1C BD 7A E9
E9 55 1C BD 7A
7A E9 55 1C BD
BD 7A E9 55 1C
3
1C BD
10
7A E9
15
BD 7A 1C
20
```

Keluaran :

```
<< ============ HASIL OUTPUT ============ >>

Reward maksimal:  35
Isi buffer:  1C 7A E9 7A 1C
Koordinat setiap token:
(0, 0) 1C
(3, 0) 7A
(3, 1) E9
(0, 1) BD

Waktu eksekusi: 0.000 ms
```

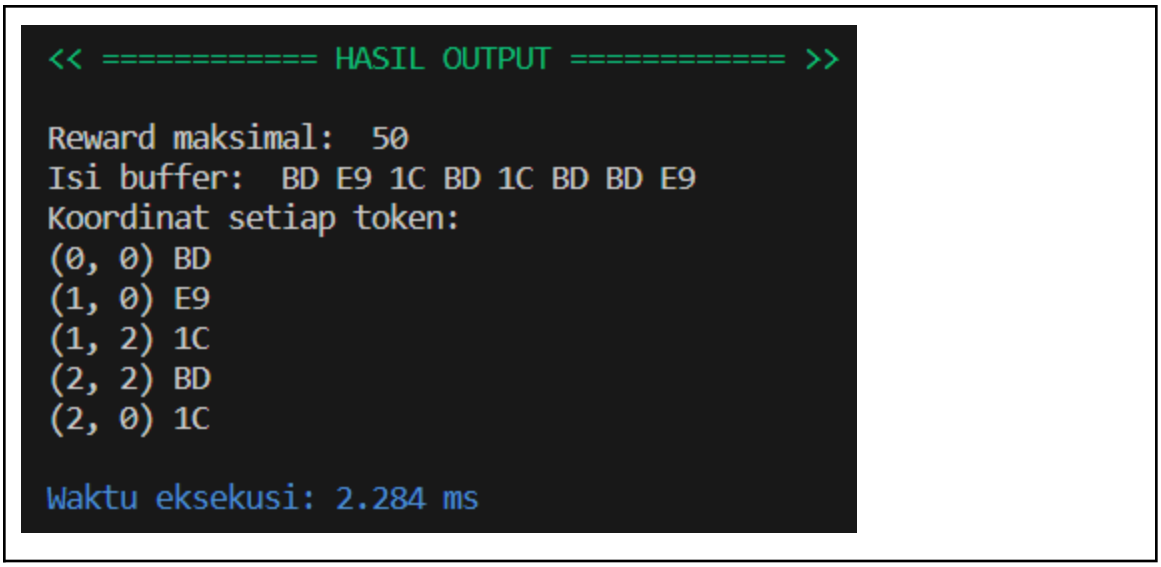
5. Uji Coba 5

```
Masukkan :
3
4 4
BD E9 1C 55
1C 55 BD E9
E9 1C 55 BD
55 BD E9 1C
3
BD 55
10
1C E9
15
E9 1C 55
20

Keluaran :
```

```
<< ============ HASIL OUTPUT ============ >>

Reward maksimal:  15
Isi buffer:  BD 1C E9
Koordinat setiap token:
(0, 0) BD
(1, 0) 1C
(1, 3) E9

Waktu eksekusi: 0.000 ms
```

6. Uji Coba 6

```
Masukkan :
8
3 3
BD 1C E9
E9 BD 1C
1C E9 BD
3
BD E9 1C
10
E9 1C BD
15
1C BD E9
20

Keluaran :
```



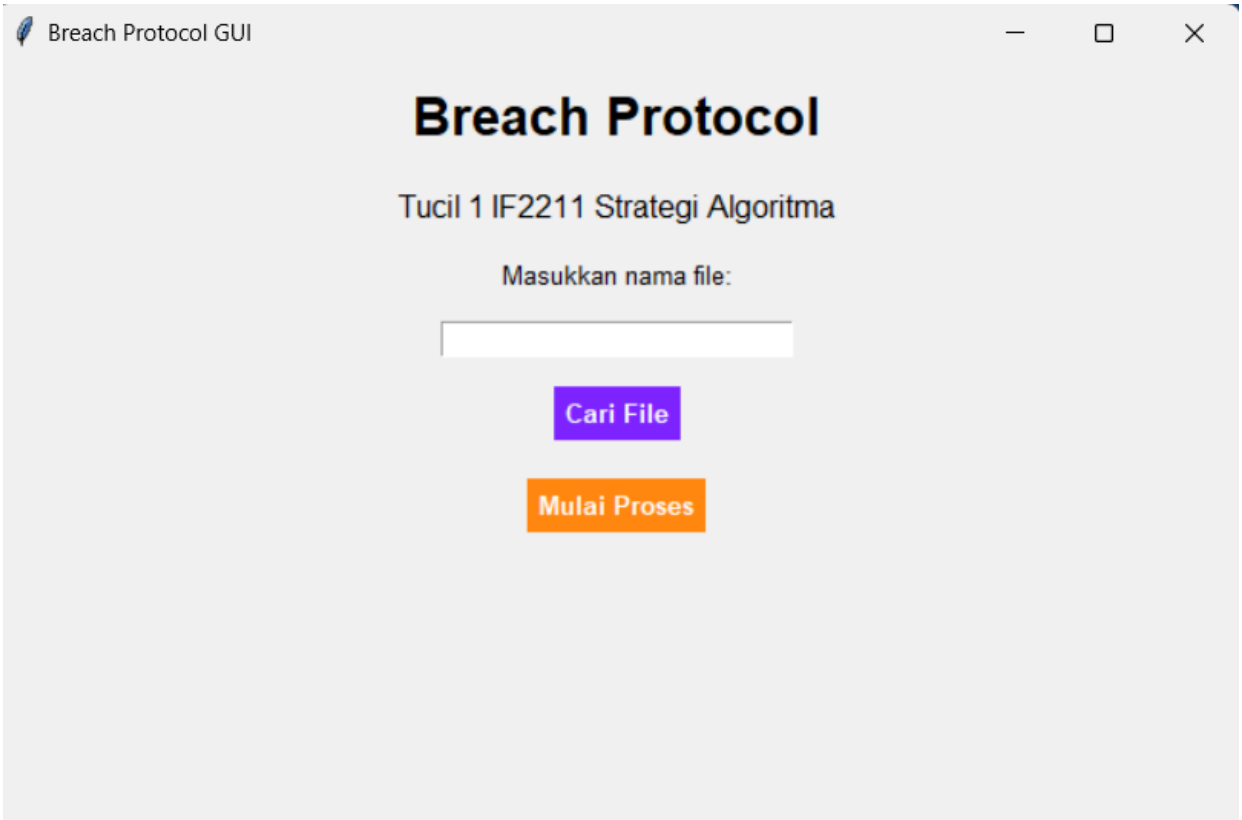
IV. Repository

[Github Repository]

V. Lampiran

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	✔	
7. Program memiliki GUI	✔	

Tabel 5.1 Spesifikasi Program



Gambar 5.1 GUI Program