**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
2. semua sekuens berhasil dicocokkan atau buffer penuh.
3. Pemain memulai dengan memilih satu token pada posisi baris paling atas dari matriks.
4. Sekuens dicocokkan pada token-token yang berada di buffer.
5. Satu token pada buffer dapat digunakan pada lebih dari satu sekuens.
6. Setiap sekuens memiliki bobot hadiah atau reward yang variatif.
7. 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.

1. 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**

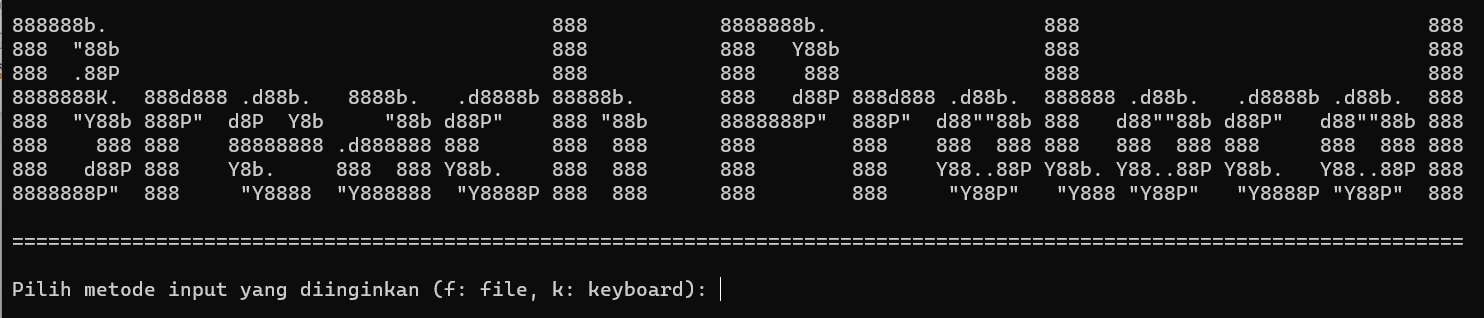
Program terdiri dari tiga file source code, yaitu main.py, util.py, dan protocol.py. File main.py berfungsi sebagai file utama yang dijalankan, file util.py berisi fungsi-fungsi yang digunakan pada file main.py, dan file protocol.py berisi algoritma utama dari program.

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

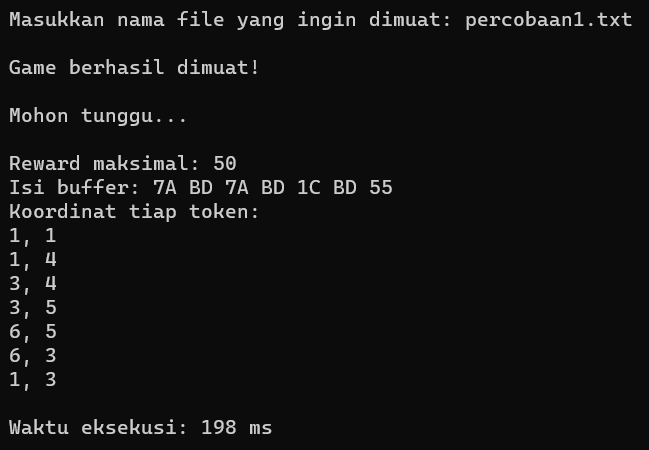
**HASIL UJI COBA PROGRAM**

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



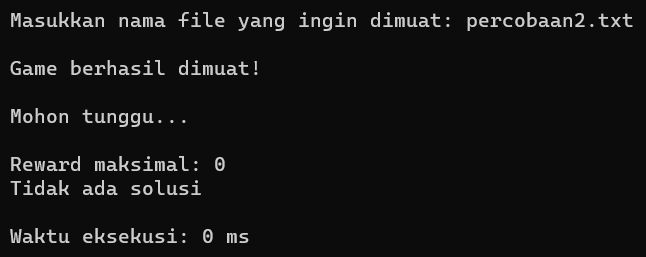
**Gambar 4.1** Tampilan awal program

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.

1. **Input melalui file**
2. Data Uji 1

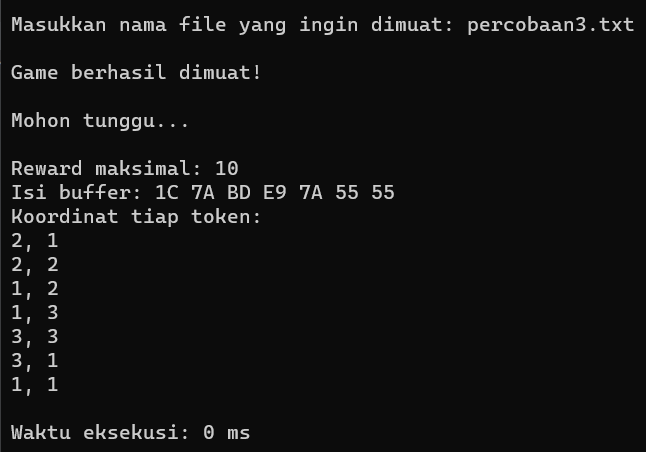
|  |
| --- |
| 7  6 6  7A 55 E9 E9 1C 55  55 7A 1C 7A E9 55  55 1C 1C 55 E9 BD  BD 1C 7A 1C 55 BD  BD 55 BD 7A 1C 1C  1C 55 55 7A 55 7A  3  BD E9 1C  15  BD 7A BD  20  BD 1C BD 55  30 |

**Gambar 4.2** Hasil Data Uji 1

1. Data Uji 2

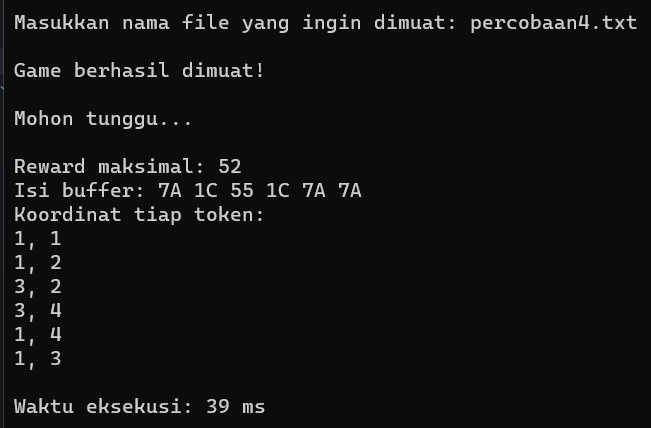
|  |
| --- |
| 10  3 3  55 1C 55  BD 7A BD  E9 BD 7A  2  1C 7A BD E9 7A 55 55  10  55 BD  5 |

**Gambar 4.3** Hasil Data Uji 2

1. 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 |

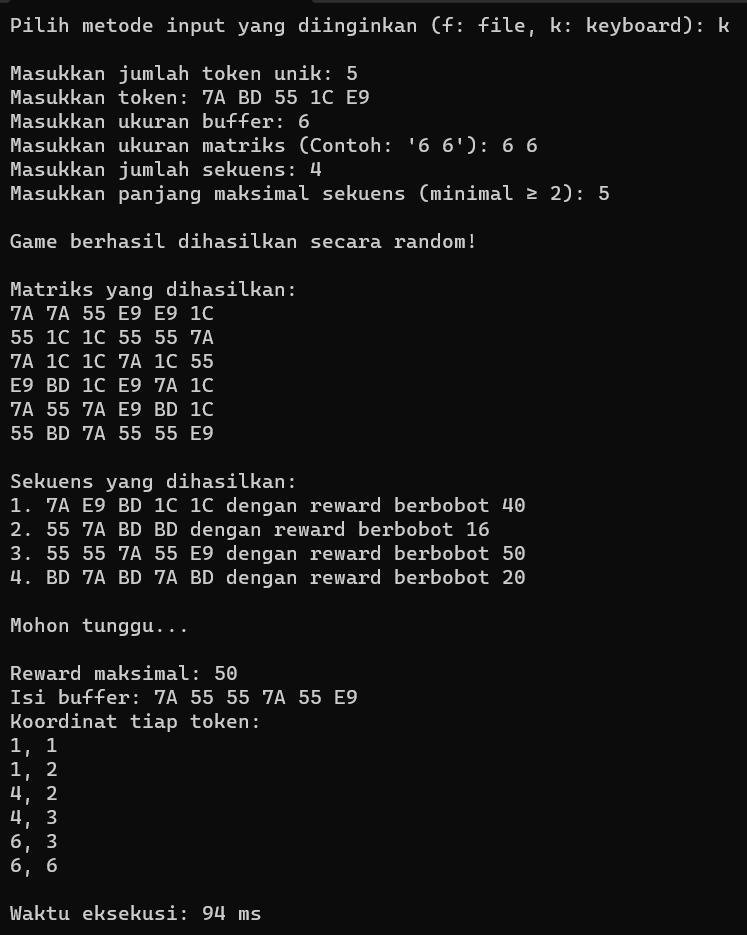
**Gambar 4.4** Hasil Data Uji 3

1. 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 |

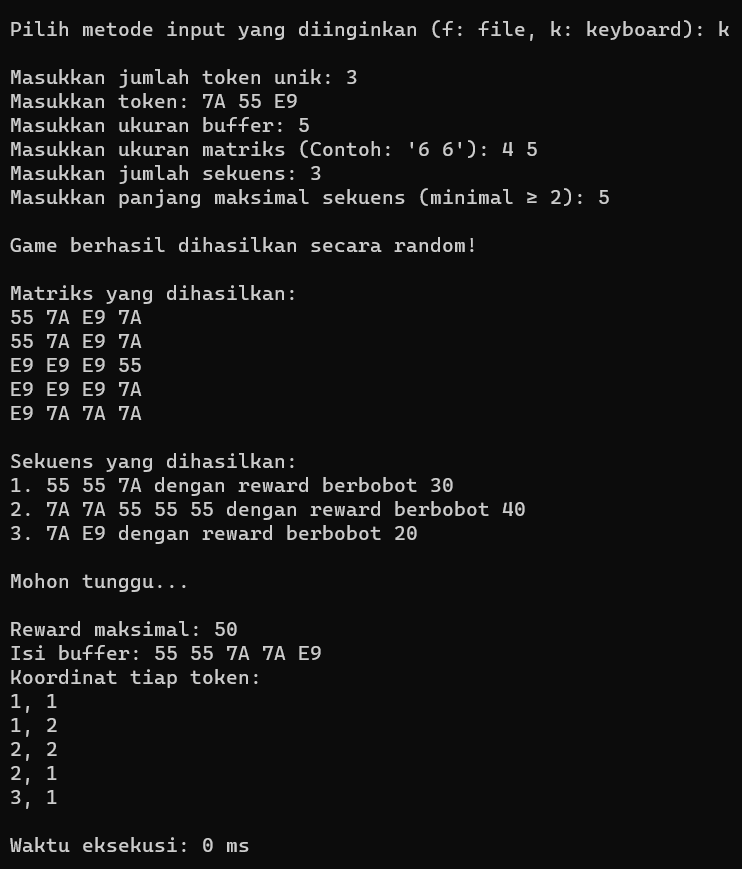
**Gambar 4.5** Hasil Data Uji 4

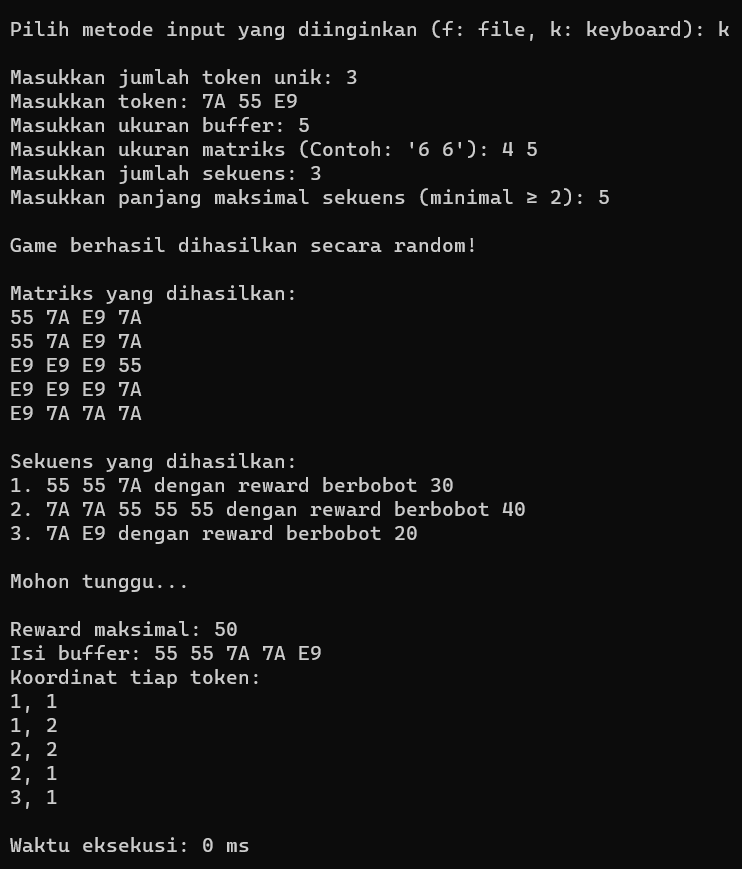
1. **Input melalui keyboard**
2. Data Uji 1



**Gambar 4.6** Hasil Data Uji 5

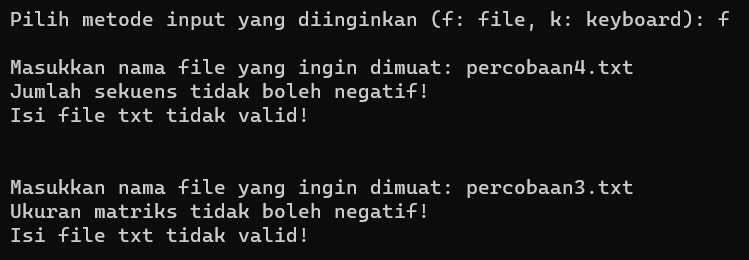
1. Data Uji 2



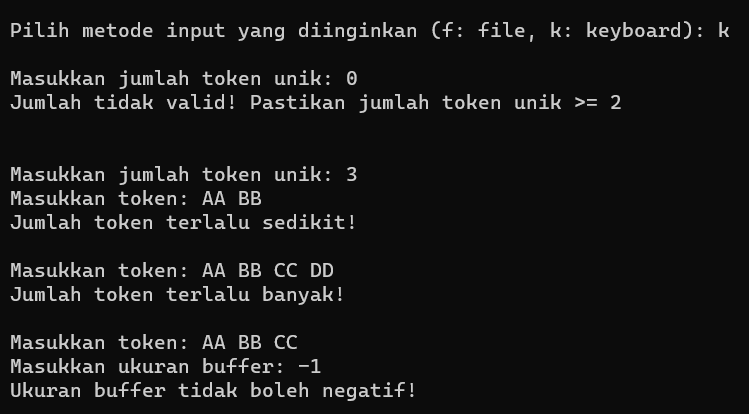


**Gambar 4.7** Hasil Data Uji 6

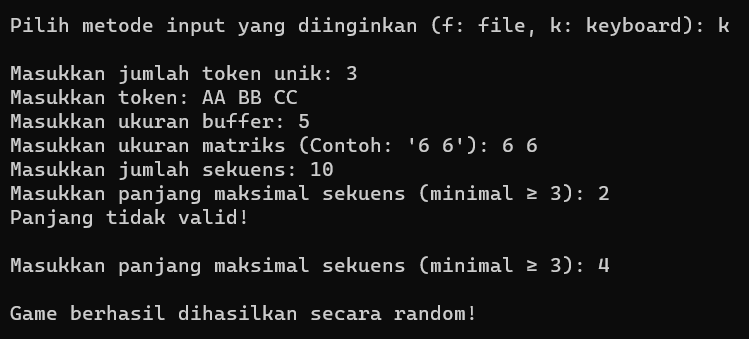
1. **Contoh input yang tidak valid**

****

**Gambar 4.8** Hasil Data Uji 7

****

**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**

Repositori github dapat diakses melalui pranala berikut:

<https://github.com/IrfanSidiq/Tucil1_13522007>

|  |  |  |
| --- | --- | --- |
| **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 |  | **√** |