Laporan Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II Tahun Akademik 2023/2024

Penyelesaian *Cyberpunk 2077 Breach Protocol* Dengan Algoritma Brute Force

Disusun Oleh:
Edbert Eddyson Gunawan - 13522039
K-01

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2024

Daftar Isi

1. Deskripsi Tugas

Gambar 1. Cyberpunk 2077 Breach Protocol
(Sumber: https://cyberpunk.fandom.com/wiki/Quickhacking)

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

2. Algoritma Bruteforce

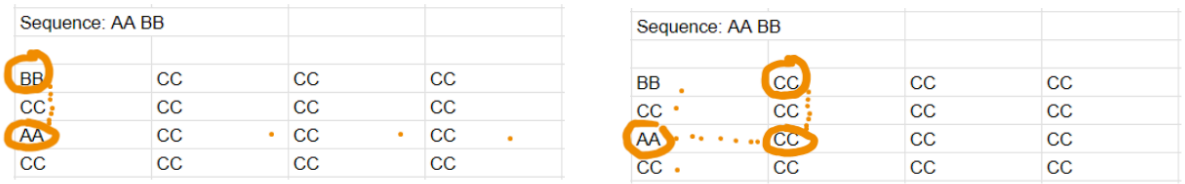
Pada laporan ini, algoritma yang digunakan oleh penulis adalah bruteforce dengan pendekatan *heuristic search*.

2.1 Pendekatan Heuristik

Pendekatan heuristik yang digunakan oleh penulis adalah, setiap *cell* matriks dapat dikunjungi dalam 3 langkah, sehingga setiap token pertama pada *sequence* (jawaban *sequence* potensial) dapat diidentifikasi dalam 3 langkah. Saat program *generate* token ke-3 untuk himpunan solusi, maka dilakukan pengecekkan dengan tahap sebagai berikut.

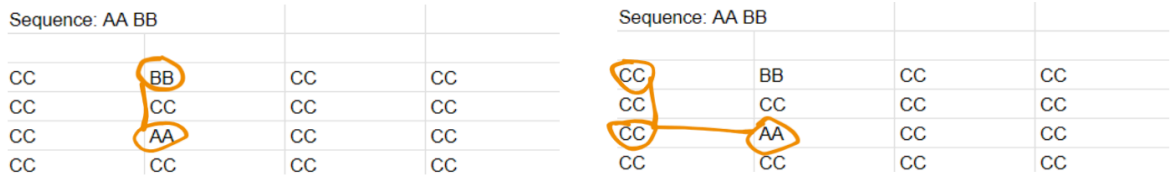
1. Jika token ke-3 dari himpunan solusi sama dengan token ke-1 salah satu *sequence*, maka dilanjutkan proses *generate* token ke-4, dst.
2. Jika token ke-2 dan ke-3 dari himpunan solusi sama dengan token ke-1 dan ke-2 salah satu *sequence* maka dilanjutkan proses *generate* token ke-4, dst.
3. Jika token ke-1 dan ke-2 dari himpunan solusi merupakan *sequence* dari salah satu *sequence* maka dilanjutkan proses *generate* token ke-4, dst.
4. Jika token ke-1, ke-2, dan ke-3 dari himpunan solusi merupakan awalan dari salah satu *sequence* maka dilanjutkan proses *generate* token ke-4, dst.
5. Jika ketiga token dari himpunan solusi tidak memenuhi salah satu dari keempat syarat diatas (syarat 1-4), maka proses *generate* di stop, dan dilanjutkan dengan token ke-3 yang lain.

Sebagai contoh Gambar 2, dimana pada awal pembentukan pola didapatkan “BB AA …” karena setelah 2 token, tidak memenuhi syarat-syarat diatas, maka tidak akan dilanjutkan. Namun saat pembentukan pola didapatkan “CC CC AA” maka pola dilanjutkan karena memenuhi salah satu syarat diatas, dimana token ke-3, AA, merupakan token ke-1 dari *sequence*.



Gambar 2. Contoh Visualisasi Pembentukan Pola
(Sumber: Arsip Pribadi)

Alasan 3 langkah dipilih karena, 4, 5, dst hanya *redundancy* sedangkan terdapat *counter test-case* pada 2 langkah, dan dan tidak mungkin mendapat kepastian pada 1 langkah.



Gambar 3. Kasus 2 langkah tidak mungkin, dan 3 langkah mungkin.
(Sumber: Arsip Pribadi)

Pada Gambar 3, kita dapat melihat bahwa pada kasus 2 langkah, meskipun bisa menjangkau setiap *cell* pada matrix namun pola AA BB tidak akan bisa dicapai. Berbeda pada kasus 3 langkah dimana *sequence* AA BB dapat dicapai.

2.2 Langkah-Langkah Bruteforce

Berikut langkah-langkah penyelesaian

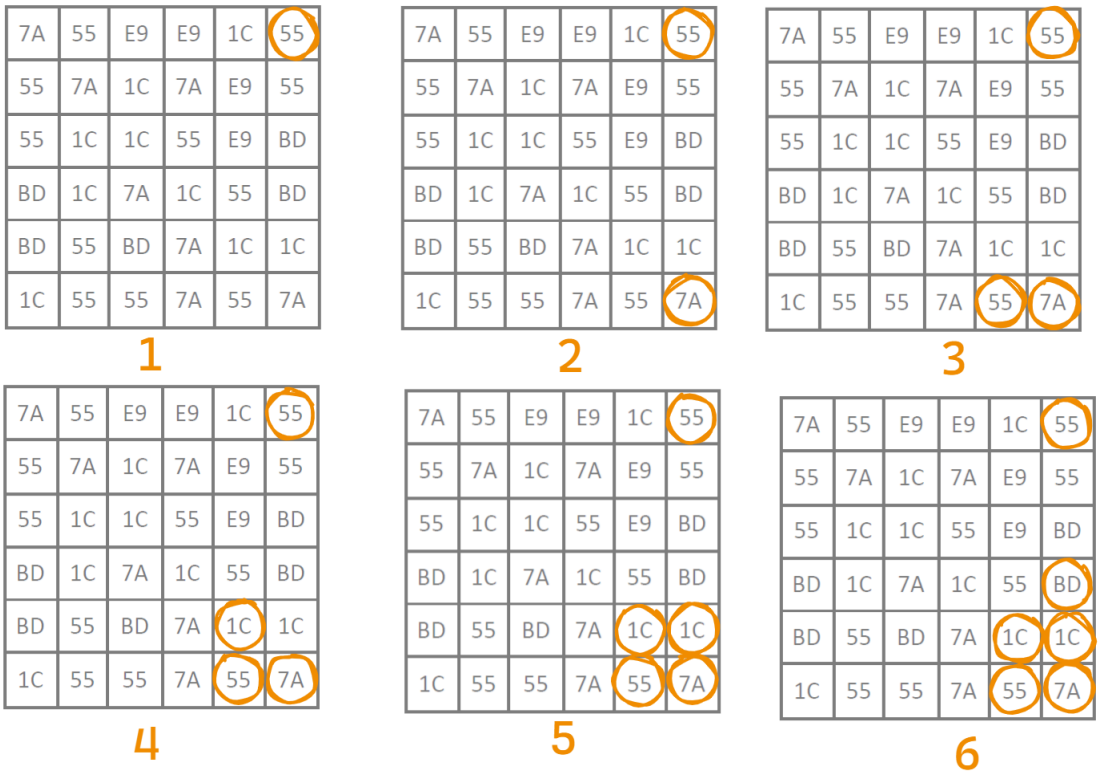
1. *Generate* semua kemungkinan pola token sesuai dengan panjang buffer. Metode yang digunakan adalah pemilihan secara selang-seling horizontal-vertikal. Saat *generate* maka dilakukan pengecekkan secara heuristik (lihat 2.1 Pendekatan Heuristik).
2. Pola selalu dimulai dari lokasi (1,1) kiri-atas. Jika *cell* belum pernah dikunjungi maka pilih *cell* tersebut. Setelah memilih sebuah *cell* pada koordinat (a, 1), dengan a adalah rentang dari 1 hingga lebar matriks, dilanjutkan dengan memilih *cell* pada koordinat (a, b), dengan nilai b adalah rentang dari 1 hingga tinggi matriks, syaratnya yakni *cell* belum pernah dikunjungi. Misal, buffer berukuran 6 maka akan dibuat pola sebagai berikut.

Gambar 4. Proses Pembentukan Pola “7A 55 7A 55 E9 1C”
(Sumber: Arsip Pribadi)

Menghasilkan “7A 55 7A 55 E9 1C” lalu

Gambar 5. Proses Pembentukan Pola “7A 55 7A 55 E9 1C”
(Sumber: Arsip Pribadi)

Menghasilkan pola “7A 55 7A 55 E9 1C”. hingga pola terakhir yaitu,



Gambar 6. Proses Pembentukan Pola “55 7A 55 1C 1C BD”
(Sumber: Arsip Pribadi)

Yang menghasilkan pola “55 7A 55 1C 1C BD” dengan syarat **sel matriks (token) yang sudah di pilih, tidak dapat dipilih kembali.**

Disini, disimpan titik koordinat setiap token yang dipilih, dan *string* hasilnya dalam *array*.

3. Kemudian, dilakukan pengecekkan *sequences* dengan himpunan solusi. Dengan cara mencari apakah *substring* dari solusi mengandung *sequences* yang ingin dicari.
4. Dengan meng-iterasi semua himpunan solusi, dilakukan pencatatan maksimum skor yang mungkin didapatkan. Jika maksimum skor ditemukan, maka catat juga *string* solusi dan titik koordinat setiap token yang dipilih.
5. Jika permasalahan tidak mengandung *sequence* (tidak memiliki solusi) maka output akan bernilai 0.
6. Jika Permasalahan mengandung *sequence* meskipun seluruh *sequence* bernilai negatif, maka output akan bernilai nilai negatif paling besar (solusi optimum, dengan terdapat *sequence*).

3. Source Code

Penulis membagi file menjadi 4 file yakni server.py, main.py, RF.py, Solution.py. Struktur folder sebagai berikut

```
/Tucil1_13522039
.
├── LICENSE
├── README.md
├── bin
├── doc
│   └── Tucil1_K1_13522039_Edbert Eddyson Gunawan.pdf
├── src
│   ├── RF.py
│   ├── Solution.py
│   ├── main.py
│   ├── server.py
│   ├── templates
│   │   ├── index.html
│   │   └── solution.html
│   └── test.py
└── test
    ├── solution-21.txt
    ├── solution-generated-empty.txt
    ├── solution-generated1.txt
    ├── solution-haha.txt
    ├── solution-test1.txt
    ├── solution-test2.txt
    ├── test1.txt
    └── test2.txt
```

3.1 server.py

```python
from flask import Flask, render_template, request, url_for, flash
import os
import numpy as np
import json

import main
import Solution

m_obj = None
sol = None

app = Flask(__name__)

abs_path = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

@app.route('/')
def hello():
    return render_template("index.html")

@app.route('/save-file', methods=['POST'])
def saveFile():
    result_file = sol.saveToFile(request.form.get('filename'))
    return json.dumps({'success':True, 'fname': result_file}), 200, {'ContentType':'application/json'}

@app.route('/solve', methods=['POST'])
def solve():
    global sol, m_obj

    m_obj = main.App()
    if "auto-generate" in request.form:
        print("go to auto generate")
        m_obj.rf.choice = 2
        arr = ['buffer-size', 'token-size', 'token-id', 'matrix-col', 'matrix-row', 'seq-id', 'seq-len', 'seq-min',
'seq-max']

        for x in arr:
            if x not in request.form:
                flash(f'{x} is required!')

        buffer_size = int(request.form['buffer-size'])
        token_size = int(request.form['token-size'])
        token = np.array(request.form['token-id'].split(' '))
        token = token[token != '']
        matrix_size = (int(request.form['matrix-row']), int(request.form['matrix-col']))
        seq_size = int(request.form['seq-id'])
        seq_len = int(request.form['seq-len'])
        min_score = int(request.form['seq-min'])
        max_score = int(request.form['seq-max'])

        print("buffer size:", buffer_size)
        m_obj.rf.generate_problem(buffer_size, matrix_size, token_size, token, seq_size, seq_len, min_score,
max_score)
        print("buffer size:", m_obj.rf.buffer_size)
        sol = Solution.Solution(m_obj.rf.matrix_size, m_obj.rf.sequence, m_obj.rf.matrix, m_obj.rf.buffer_size)
        print("maximum depth:", sol.max_depth)
        sol.main()
```

```
    elif "submit" in request.form:
        arr = ['buffer-size', 'matrix', 'seq-score']
        for x in arr:
            if x not in request.form:
                flash(f'{x} is required!')

        m_obj.rf.buffer_size = int(request.form['buffer-size'])
        matrix = request.form['matrix'].split('\n')
        m_obj.rf.matrix = list(map(lambda x: [y for y in x.strip('\r').split(' ') if  y], matrix))
        # matrix = matrix[matrix != '']
        m_obj.rf.sequence = []
        seq_score = request.form['seq-score'].split('\n')
        for i in range(len(seq_score) // 2):
            seq, val = seq_score[2*i].strip('\r'), int(seq_score[1 + i*2].strip('\r'))
            m_obj.rf.sequence.append(([y for y in seq.split(' ') if y], val))
        m_obj.rf.sequence_size = len(m_obj.rf.sequence)

        # print(m_obj.rf.matrix)
        # print(m_obj.rf.sequence)
        m_obj.rf.matrix_size = [len(m_obj.rf.matrix), len(m_obj.rf.matrix[0])]
        sol = Solution.Solution(m_obj.rf.matrix_size, m_obj.rf.sequence, m_obj.rf.matrix, m_obj.rf.buffer_size)
        # print(sol.ans_pool)
        sol.main()
    elif "file-solve" in request.form:
        if "file-name" not in request.form:
            flash("filename is required!")

        filename = request.form['file-name']
        m_obj.rf.read_file(filename)

        sol = Solution.Solution(m_obj.rf.matrix_size, m_obj.rf.sequence, m_obj.rf.matrix, m_obj.rf.buffer_size)
        sol.main()


    between = []

    for i in range(len(sol.max_coor)-1):
        a, b = sol.max_coor[i], sol.max_coor[i+1]

        if(a[0] == b[0]):
            start_col = min(a[1], b[1])
            end_col = max(a[1], b[1])

            for j in range(start_col, end_col+1):
                between.append((a[0], j))
        elif(a[1] == b[1]):
            start_r = min(a[0], b[0])
            end_r = max(a[0], b[0])

            for j in range(start_r, end_r):
                between.append((j, a[1]))

    return render_template('solution.html', matrix=m_obj.rf.matrix, sequence=m_obj.rf.sequence,
                           matrix_size=m_obj.rf.matrix_size, max_score=sol.max_score, coordinates=sol.max_coor,
                           exec_time=round(sol.time_elapsed * 1000, 2), len_coor=len(sol.max_coor), between=between)

if __name__ == "__main__":
    app.run(debug=True)
```

3.2 main.py

```python
# libraries
from colorama import Style, Fore

# external files
import RF
import Solution

class Util:
    def __init__(self):
        pass

    def showBanner(self):
        print("Make Sure Your Input File is in test folder!")

class App:
    def __init__(self):
        self.rf = RF.RF()
        self.ut = Util()
        self.sol = None


    def main(self):
        self.rf.input_prompt()
        if(self.rf.choice == 3):
            self.exit_program()
        elif(self.rf.choice == 1):
            self.rf.read_file()
        elif(self.rf.choice == 2):
            self.rf.generate_auto()

        self.sol = Solution.Solution(self.rf.matrix_size, self.rf.sequence, self.rf.matrix, self.rf.buffer_size)
        self.sol.main()

        choice = input("Apakah anda ingin menyimpan solusi? (y/n)\n> ")

        if(choice == 'y'):
            fname = input("Masukkan nama file\n> ")
            self.sol.saveToFile(fname)


    def exit_program(self):
        exit(0)

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

3.2 RF.py

```python
import random
import numpy as np
import os

class RF:
    def __init__(self):
        self.fname = ''
        self.buffer_size = 0
        self.matrix_size = None
        self.matrix = []
        self.sequence_size = 0
        self.sequence = []
        self.choice = 0

        self.abs_path = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

    def input_prompt(self):
        choice = input("Apakah Anda ingin generate otomatis? (y/n)\n> ")
        if(choice == 'n'):
            self.choice = 1
        elif(choice == 'y'):
            self.choice = 2
        else:
            self.choice = 3

    def showBanner(self):
        print("Make Sure Your Input File is in test folder!")

    def generate_auto(self):
        token_n = input("Masukkan jumlah token\n> ")
        token = np.array(input("Masukkan token\n> ").split(' '))
        token = token[token != '']
        buffer_size = int(input("Masukkan ukuran buffer\n> "))
        matrix_size = list(map(lambda x: int(x), input("Masukkan ukuran matriks (col row)\n> ").split(' ')))
        matrix_size = matrix_size[::-1]
        seq_size = int(input("Masukkan banyak sequence\n> "))
        seq_len = int(input("Masukkan maksimum panjang sequence\n> "))

        # self.generate_problem(buffer_size, matrix_size, seq_size, token_n, token, seq_len)

    def generate_problem(self, buffer_size, matrix_size, token_n, token, seq_size, seq_len, min_score, max_score):
        self.buffer_size = buffer_size
        self.matrix_size = matrix_size
        self.sequence_size = seq_size

        self.matrix = np.random.randint(0, token_n, size=(matrix_size[0], matrix_size[1]))
        self.matrix = token[self.matrix]
        print(self.matrix)

        rand_seq_size = np.random.randint(2, seq_len+1, size=(seq_size))

        self.sequence = []

        for x in rand_seq_size:
            self.sequence.append((list(token[(np.random.randint(0, token_n, size=(x)))]), random.randint(min_score,
max_score)))

        for x in self.sequence:
            print(" ".join(x[0]), "|", x[1])


    def read_file(self, fname):
        self.fname = self.abs_path + "/test/" + fname
        f = open(self.fname, 'r')
        f = f.read()
        f = f.split('\n')
        f[:] = [x for x in f if x.strip('\n')]

        # assign values
        self.buffer_size = int(f[0])
        self.matrix_size = list(map(lambda x: int(x), f[1].split(' ')))
        self.matrix_size = self.matrix_size[::-1]
        self.matrix = list(map(lambda x: x.split(' '), f[2:2+self.matrix_size[0]]))
        self.sequence_size = int(f[2+self.matrix_size[0]])
        for i in range(self.sequence_size):
            seq, val = f[2+self.matrix_size[0] + 1 + 2 *i], f[2+self.matrix_size[0] + 2 + 2 *i]
            self.sequence.append((seq.split(' '), int(val)))
```

3.3 Solution.py

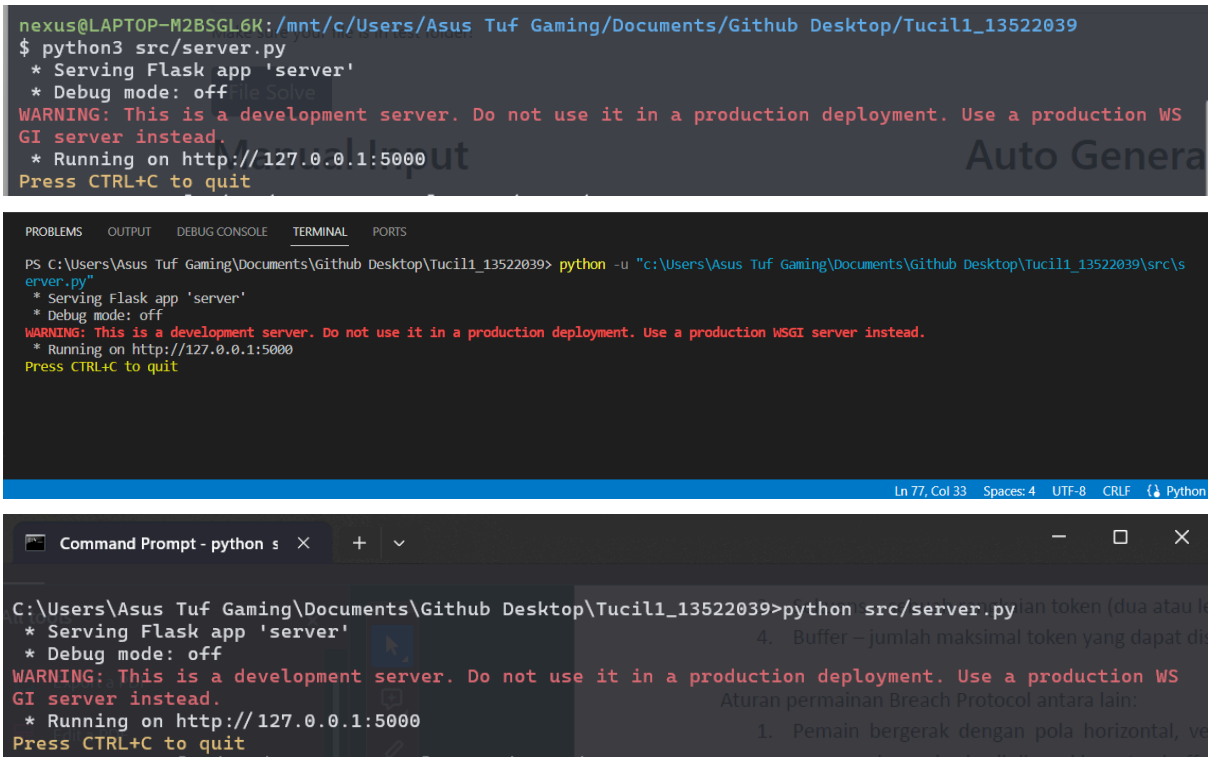
```python
import numpy as np
import time
import multiprocessing
import os

class Solution:
    def __init__(self, matrix_size, sequence, matrix, buffer_size):
        self.row = matrix_size[0]
        self.col = matrix_size[1]
        self.sequence = sequence
        self.matrix = matrix
        self.ans_pool = []
        self.max_depth = buffer_size
        self.max_score = 0
        self.max_coor = []
        self.time_elapsed = 0
        self.abs_path = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))

    def main(self):
        visited = np.zeros((self.row, self.col))

        start_t = time.time()
        self.brute_force(0, 0, True, "", 0, [], visited)
        self.check_ans()
        end_t = time.time()

        self.time_elapsed = end_t - start_t

        self.show_banner()
        self.show_answer()

    def brute_force(self, row, col, b, ans, depth, coor, visited):
        # print(len(self.ans_pool))
        if (depth == self.max_depth): # basis max depth
            # print("called")
            self.ans_pool.append((ans, coor))
            return
        elif (depth > self.max_depth):
            return

        # heuristic approach
        if (depth == 3):
            temp_ans = ans.split(' ')
            temp_ans = [x for x in temp_ans if x]

            verdict = False
            # check 3rd token of ans is 1st token of sequence
            for x in self.sequence:
                if(len(temp_ans) >= 3):
                    if x[0][0] == temp_ans[2]:
                        verdict = True

            if(len(temp_ans) >= 3 and len(x[0]) >= 2 and not verdict):
                for x in self.sequence:
                    if x[0][0] == temp_ans[1] and x[0][1] == temp_ans[2]:
                        verdict = True
            if(len(x[0]) >= 2 and not verdict):
                for x in self.sequence:
                    if x[0][0] == temp_ans[0] and x[0][1] == temp_ans[1] and len(x[0]) == 2:
                        verdict = True

            if(len(temp_ans) >= 3 and len(x[0]) >= 3 and not verdict):
                for x in self.sequence:
                    if x[0][0] == temp_ans[0] and x[0][1] == temp_ans[1] and x[0][2] == temp_ans[2]:
                        verdict = True


            if (not verdict):
                return
```

```
        if (b): # b = True -> Horizontal
            for j in range(self.col):
                visited2 = np.copy(visited)
                if(visited2[row][j] == 0):
                    visited2[row][j] = 1
                    ans2 = ans + self.matrix[row][j] + " "
                    self.brute_force(row, j, not b, ans2, depth+1, [*coor, (row, j)], visited2)
        else: # b = False -> Vertical
            for i in range(self.row):
                visited2 = np.copy(visited)
                if(visited2[i][col] == 0):
                    visited2[i][col] = 1
                    ans2 = ans + self.matrix[i][col] + " "
                    self.brute_force(i, col, not b, ans2, depth+1, [*coor, (i, col)], visited2)

    def printMatrix(self, visited):
        for x in list(map(lambda x: " ".join(str(x)), visited)):
            print(x)
        print("====================================")

    def check_ans(self):
        # print("checK_ans clalled")
        max_score = float('-inf')
        max_coor = []
        # print(self.ans_pool)
        for x in self.ans_pool:
            isExist = False
            temp_score = 0
            for i in range (len(self.sequence)):
                s = " ".join(self.sequence[i][0])
                # print(s, " | ", x)
                if s in x[0]:
                    # print(s, " | ", x)
                    temp_score += self.sequence[i][1]
                    isExist = True

            # print(temp_score)
            if(temp_score > max_score and isExist):
                max_score = temp_score
                max_coor = x[1]
        if(max_score == float('-inf')):
            self.max_score = 0
        else:
            self.max_score = max_score
        self.max_coor = max_coor

    def show_banner(self):
        print("====================================")
        print("|               SOLUTION             |")
        print("====================================")

    def show_answer(self):
        print("Max score:\t", self.max_score)
        print("Sequence:\t", " ".join(list(map(lambda x: self.matrix[x[0]][x[1]], self.max_coor))))
        print("Coordinates:\n", "\n".join(list(map(lambda x: "("+str(x[1]+1)+", "+str(x[0]+1)+")", self.max_coor))))
        print("Time elapsed:\t", round(self.time_elapsed * 1000, 2), "ms")

    def saveToFile(self, fname):
        fname = self.abs_path + "/test/solution-" + fname #"solution-test1.txt"
        f = open(fname, "w")

        f.write("====================================\n")
        f.write("|               SOLUTION             |\n")
        f.write("====================================\n")
        f.write("Max score:\t{}\n".format(self.max_score))
        f.write("Sequence:\t{}\n".format(" ".join(list(map(lambda x: self.matrix[x[0]][x[1]], self.max_coor)))))
        f.write("Coordinates:\n{}\n".format("\n".join(list(map(lambda x: "("+str(x[1]+1)+", "+str(x[0]+1)+")",
self.max_coor)))))
        f.write("Time elapsed:\t{} ms".format(round(self.time_elapsed*1000, 2)))

        print(f"File saved in {fname}")

        return fname
```

4. How To Use

Berikut merupakan langkah-langkah untuk menjalankan program.

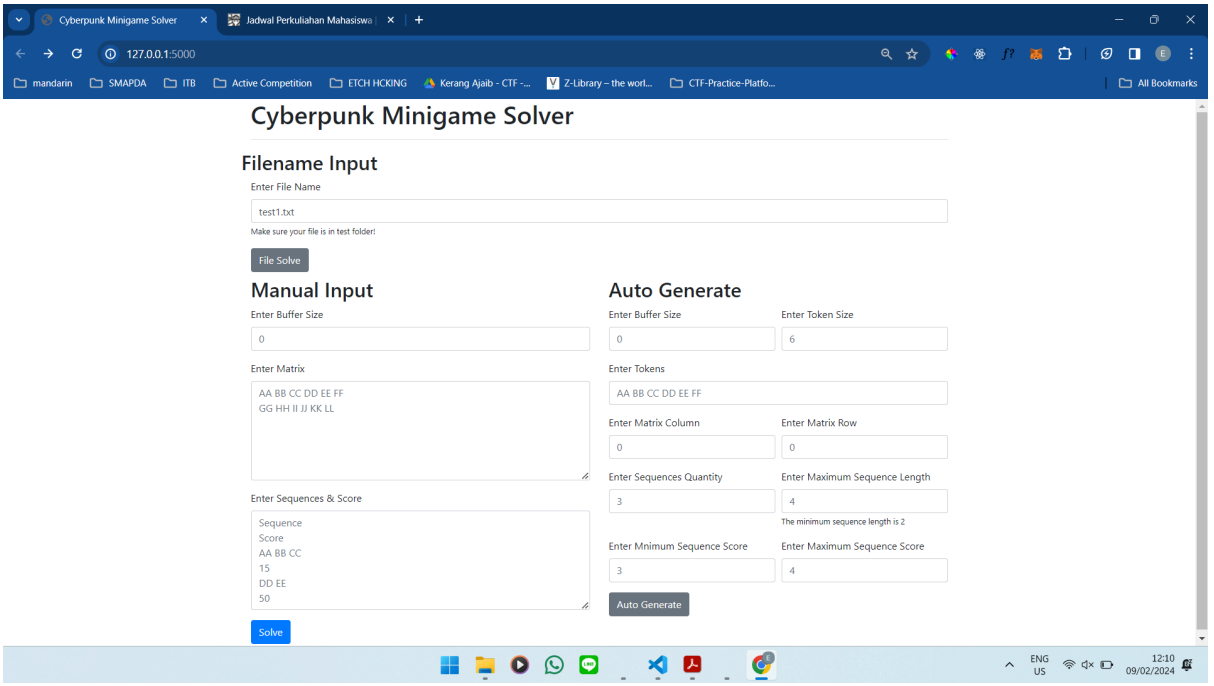
1. Clone repository dari github. Kemudian masuk ke dalam folder root.
2. Setelah itu jalankan perintah python3 src/server.py atau python src/server.py sesuaikan dengan komputer masing-masing. Berikut tampilan program berhasil di run pada komputer penulis berturut-turut WSL Ubuntu, VSCode Terminal, Command Prompt.



Gambar 7. Menjalankan Program Berturut-turut WSL Ubuntu, VSCode Terminal, Command Prompt

(Sumber: Arsip Pribadi)

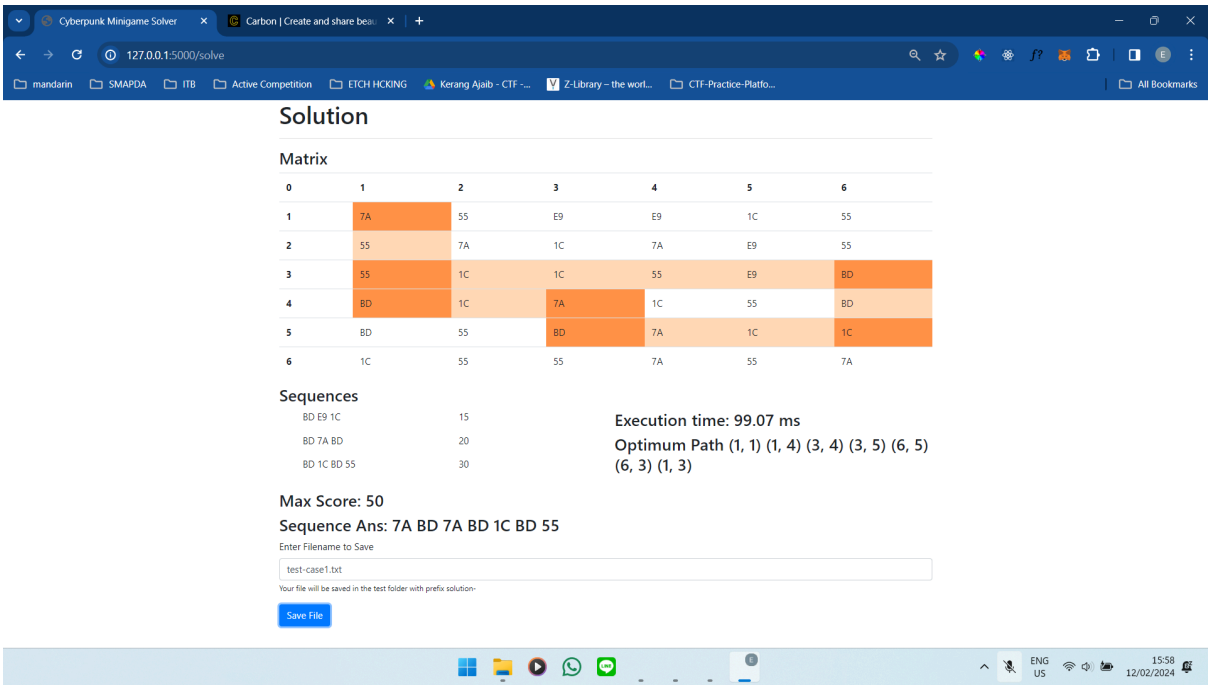
3. Setelah server dijalankan maka akan diberikan tampilan GUI dalam bentuk website sebagai berikut.



Gambar 8. Tampilan GUI Solver

(Sumber: Arsip Pribadi)

4. Terdapat 3 cara untuk menerima masukan, yakni dengan file.txt, ketik secara manual, dan otomatis inisiasi. **Untuk input dengan file, pastikan file.txt berada didalam folder test sebelum menekan tombol file solve.**
5. Setelah input dan tekan tombol, akan di redirect ke halaman baru dan ditampilkan solusi.



Gambar 9. Tampilan GUI Solusi

(Sumber: Arsip Pribadi)

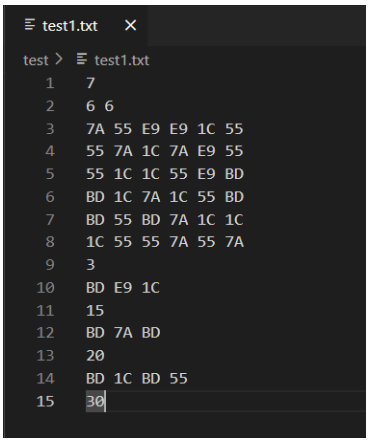
6. Setelah solusi didapatkan, maka kita dapat menyimpan solusi dengan input nama file lalu tekan tombol save.

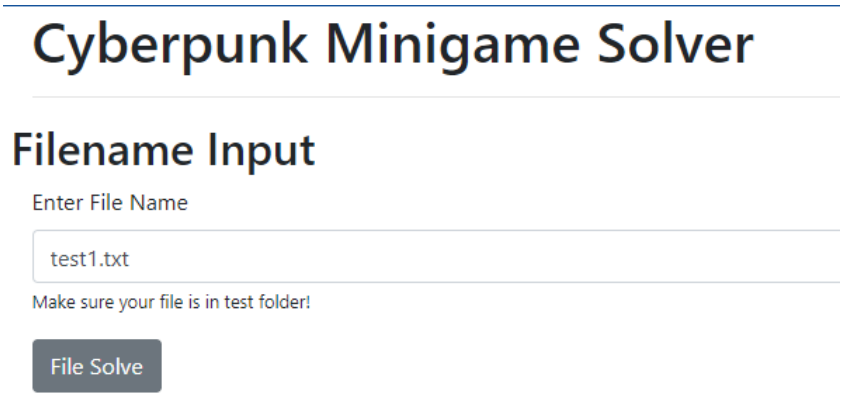
5. Test Case

5.1 Test Case 1

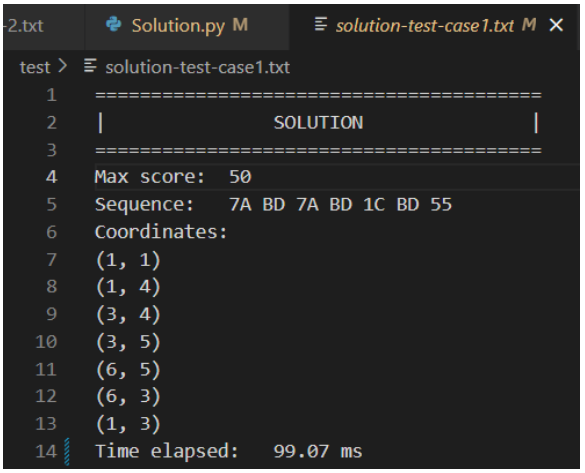
Input

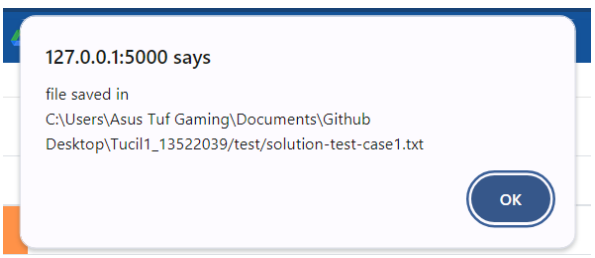
File berada didalam folder test.





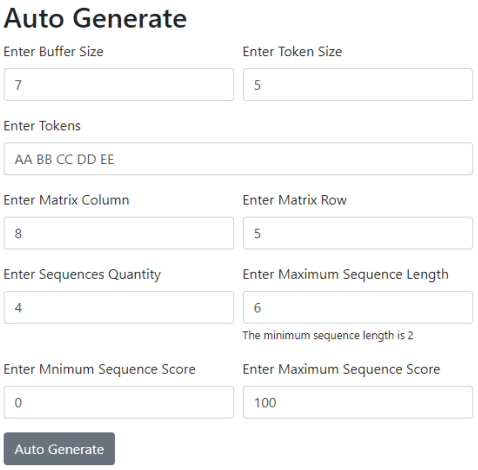
Result



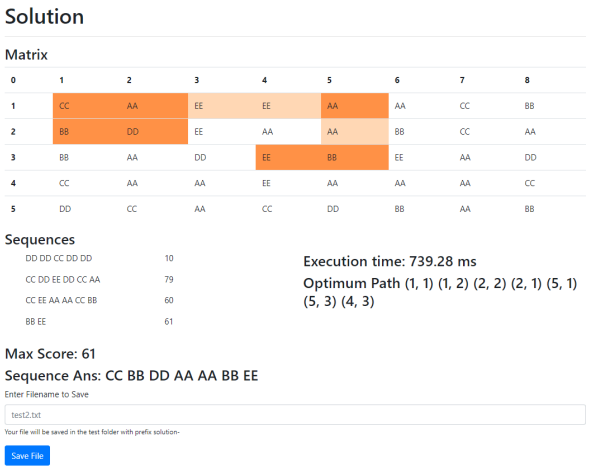


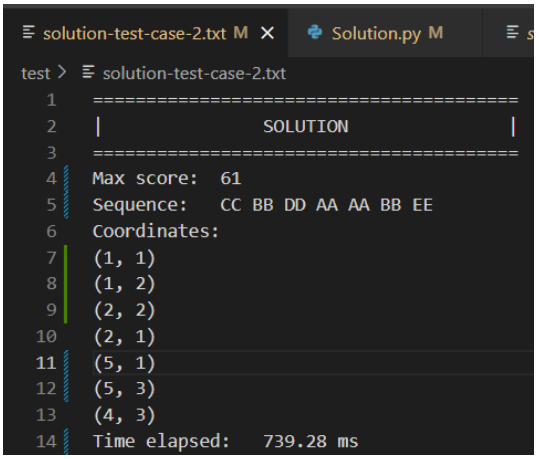
5.2 Test Case 2

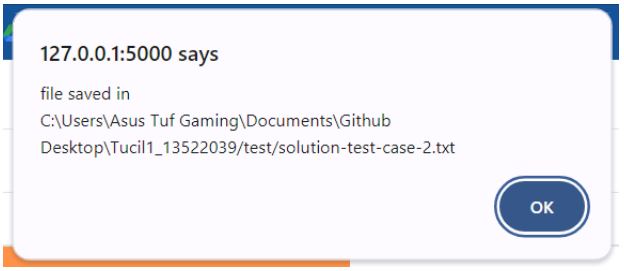
Input



Result



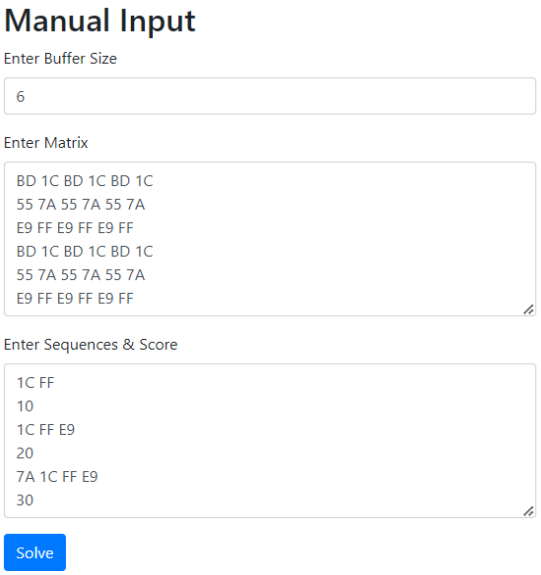




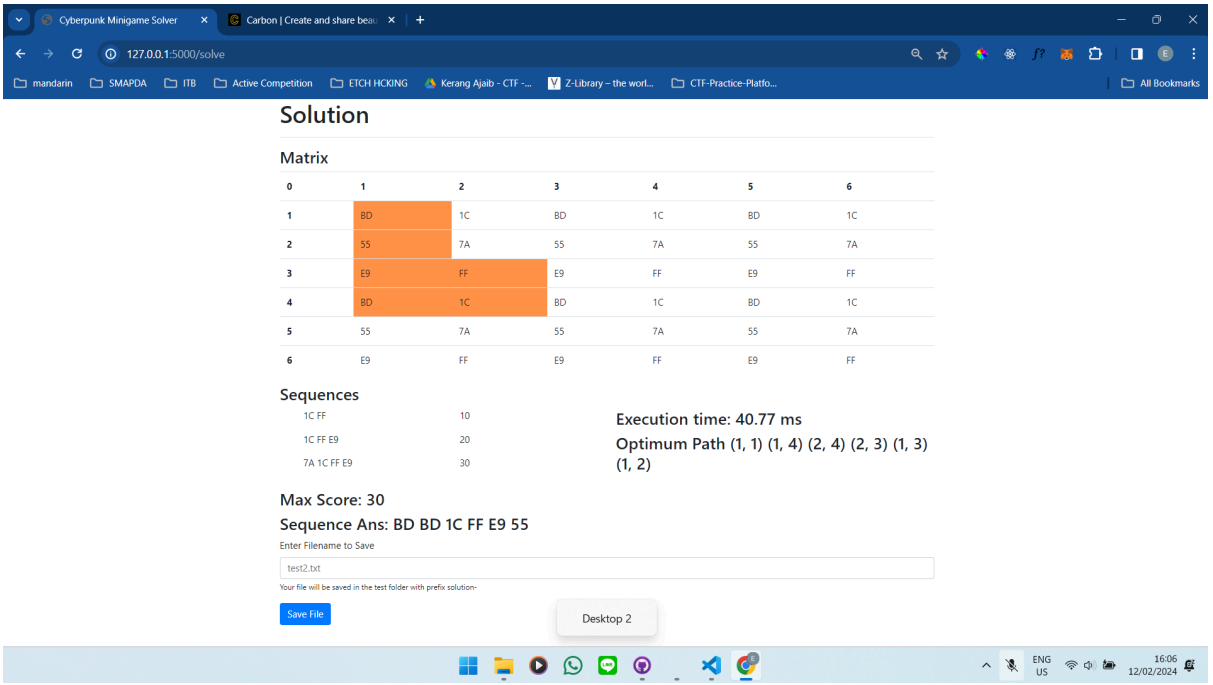
5.3 Test Case 3

Input

Ketik Manual

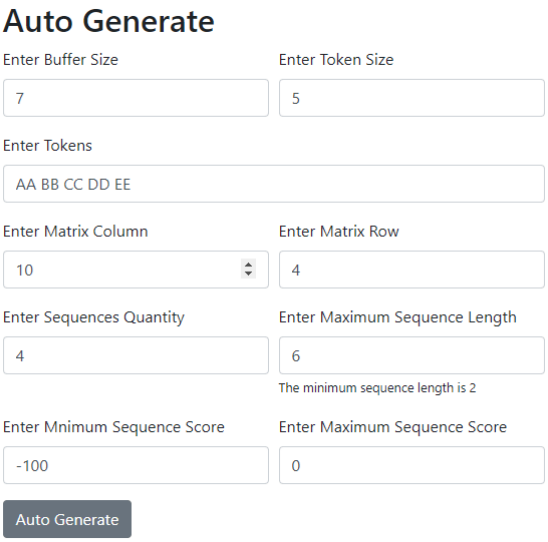


Result

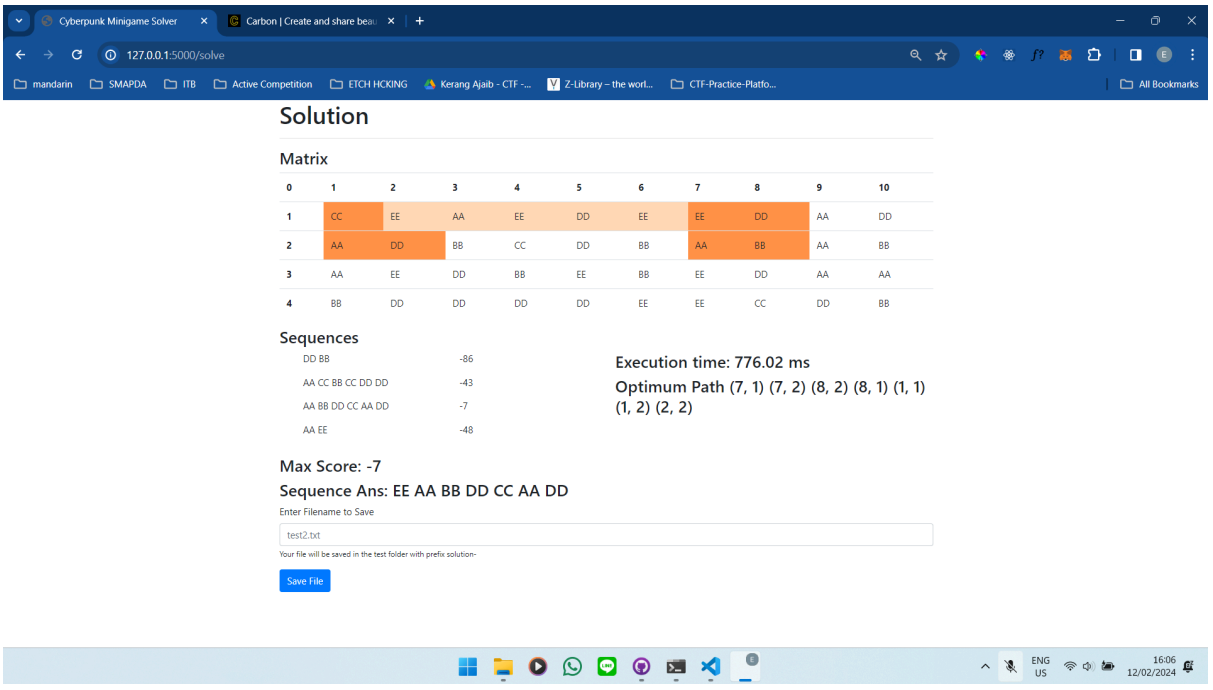


5.4 Test Case 4

Input

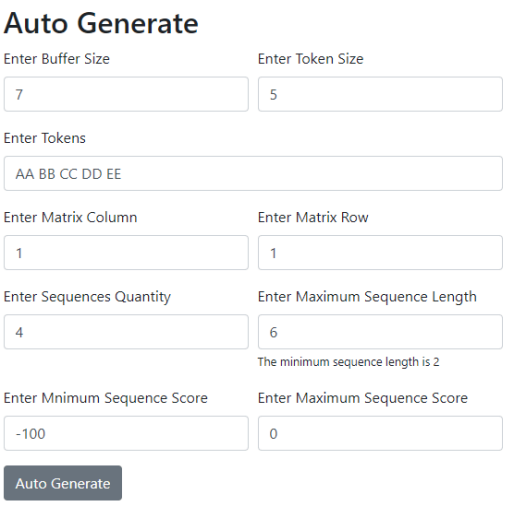


Result

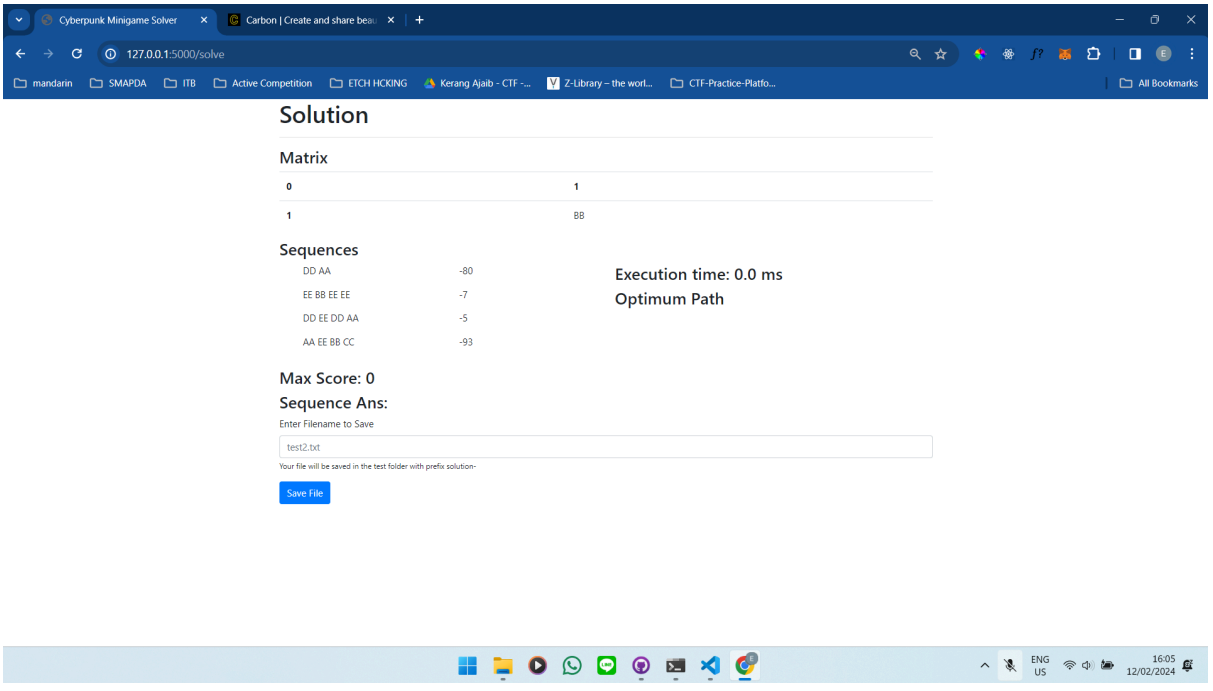


5.5 Test Case 5

Input

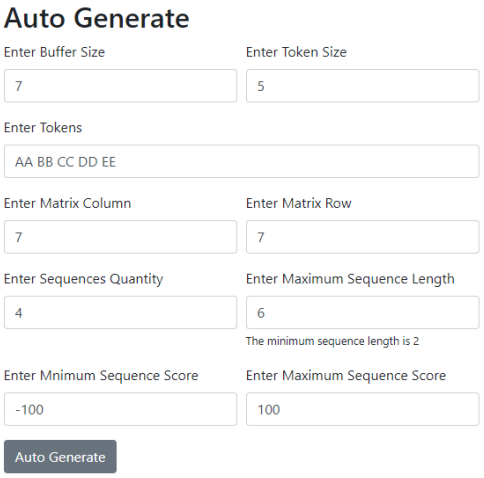


Result

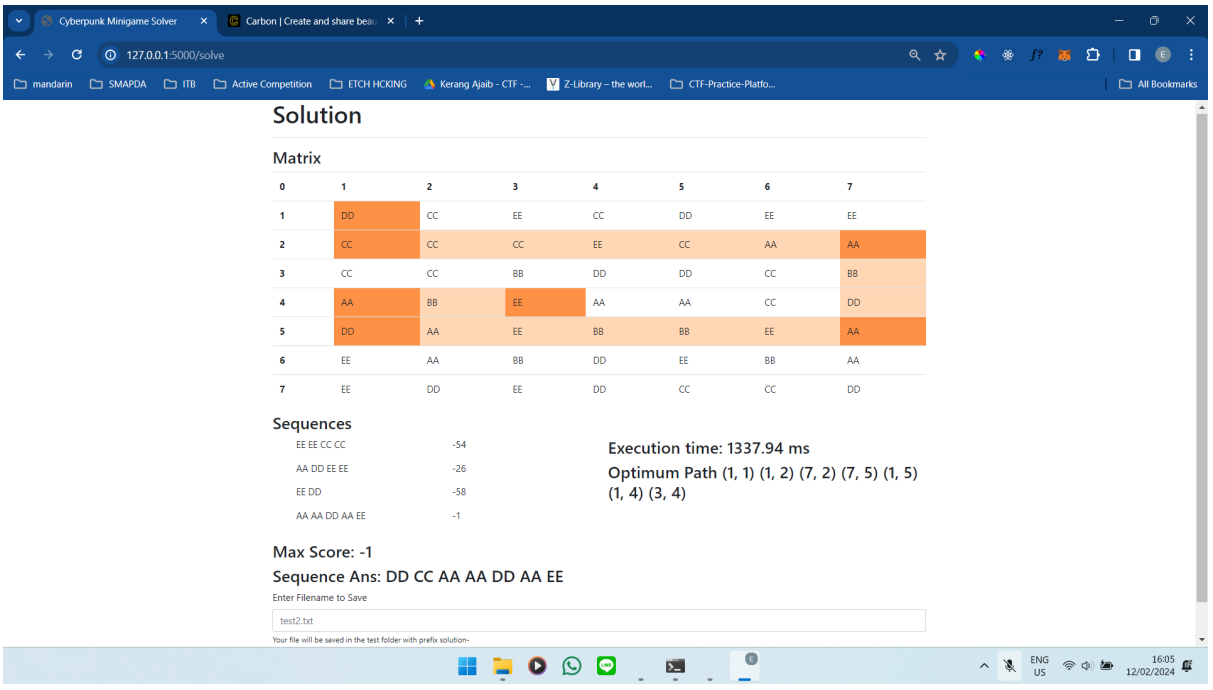


5.6 Test Case 6

Input

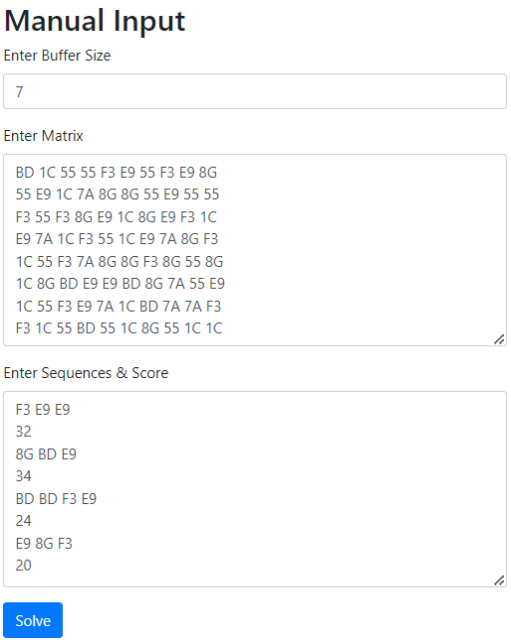


Result



5.7 Test Case 7

Input



Result

Matrix

0	1	2	3	4	5	6	7	8	9	10
1	BD	1C	55	55	F3	E9	55	F3	E9	8G
2	55	E9	1C	7A	8G	8G	55	E9	55	55
3	F3	55	F3	8G	E9	1C	8G	E9	F3	1C
4	E9	7A	1C	F3	55	1C	E9	7A	8G	F3
5	1C	55	F3	7A	8G	8G	F3	8G	55	8G
6	1C	8G	BD	E9	E9	BD	8G	7A	55	E9
7	1C	55	F3	E9	7A	1C	BD	7A	7A	F3
8	F3	1C	55	BD	55	1C	8G	55	1C	1C

Sequences

F3 E9 E9	32
8G BD E9	34
BD BD F3 E9	24
E9 8G F3	20

Execution time: 8106.76 ms

Optimum Path (1, 1) (1, 3) (5, 3) (5, 6) (7, 6) (7, 7) (4, 7)

Max Score: 66

Sequence Ans: BD F3 E9 E9 8G BD E9

Enter Filename to Save

test2.txt

GitHub

Kode dapat diakses pada repository GitHub https://github.com/WazeAzure/Tucil1_13522039

Repository akan di public pada Hari Senin 12 Februari, dan paling telat pada Selasa 13 Februari pagi.

Jika tidak dapat diakses, mohon kontak penulis di LINE yenyenhui atau TEAMS.

Poin-Poin Penting

No	Poin	Ya	Tidak
1.	Program berhasil dikompilasi tanpa kesalahan	**V**	
2.	Program berhasil dijalankan	**V**	
3.	Program dapat membaca masukan berkas .txt	**V**	
4.	Program dapat menghasilkan masukan secara acak	**V**	
5.	Solusi yang diberikan program optimal	**V**	
6.	Program dapat menyimpan solusi dalam berkas .txt	**V**	
7.	Program memiliki GUI	**V**	