LAPORAN TUGAS KECIL
STRATEGI ALGORITMA
IF 2211

Disusun oleh:

Dhafin Fawwaz Ikramullah

13522084

Daftar Isi

Daftar Isi **2**
BAB I Deskripsi Masalah **3**
BAB II Landasan Teori **5**
BAB III Pembahasan **6**
A. Source Code Program Utama 7
B. Source Code Program CLI 11
C. Source Code Program GUI 20
BAB IV Hasil Pengujian **34**
A. Berbagai Kasus 34
B. Contoh Auto Generated 43
C. Tampilan GUI 44
BAB V Kesimpulan **45**
BAB VI Lampiran **46**

BAB I Deskripsi Masalah

Gambar 1 Permainan 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.

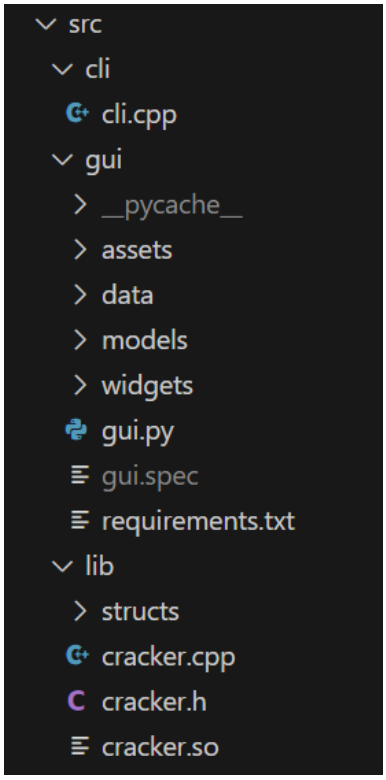
Dalam laporan ini, akan dibahas metode untuk menemukan solusi dari permainan Breach Protocol yang paling optimal untuk setiap kombinasi matriks, sekuens, dan ukuran buffer dengan menggunakan algoritma brute force.

BAB II Landasan Teori

Algoritma Brute Force merupakan sebuah algoritma dalam menyelesaikan sebuah persoalan dengan pemrograman dengan cara menguji setiap kemungkinan yang ada secara sistematis untuk mencari solusi yang diinginkan. Umumnya pendekatan ini merupakan pendekatan yang tidak efektif karena membutuhkan waktu dan memory yang cukup besar dalam algoritmanya. Biasanya ada cara yang lebih efektif dibanding algoritma brute force karena pada algoritma brute force semua kemungkinan solusi akan dilalui.

BAB III Pembahasan

Solusi program pada laporan ini dibuat menggunakan 2 bahasa pemrograman. Untuk pemrosesan berat yang berisi algoritma bruteforce menggunakan c++, untuk cli menggunakan c++, dan untuk GUI menggunakan python dengan library PyQt5. Struktur folder yang digunakan sebagai berikut



Gambar 2 Struktur folder

Pada struktur folder, folder lib akan berisi implementasi dari algoritma bruteforce. Di dalamnya terdapat cracker.cpp yang berisi fungsi utama dan folder structs yang berisi header dan implementasi header untuk struktur data yang digunakan selama proses bruteforce. Isi folder lib ini kemudian akan di compile ke dll dalam format .so untuk digunakan oleh GUI agar dapat menyampaikan data dari python ke c++. Folder cli akan berisi implementasi untuk program cli. Program cli ini akan menggunakan fungsi utama pada cracker.cpp untuk melakukan bruteforce. Lalu terdapat folder gui yang berisi implementasi untuk GUI. GUI ini akan menggunakan file cracker.so untuk melakukan bruteforce. Untuk source code lengkap dapat dilihat di pada repository: https://github.com/DhafinFawwaz/Tucil1_13522084.

Pada cracker.cpp terdapat fungsi getOptimalSolution yang merupakan fungsi utama yang akan dilakukan untuk proses bruteforce. Cara kerjanya diinisialisasi terlebih dahulu reward dan

TokenSlot maksimum yang nantinya akan diisi. Lalu dilakukan pencarian brute force menggunakan fungsi rekursif. Hasil dari rekursif tersebut akan di masukkan kedalam struktur data CrackData yang akan dikembalikan oleh getOptimalSolution. Implementasi dari fungsi rekursif tersebut cukup sederhana yaitu dengan melakukan searching dengan for loop yang di dalamnya memanggil fungsi rekursif ini lagi hingga tidak bisa ditemukan lagi jalur yang bisa dilalui. Jika sudah mentok, maka akan dihitung berapa poin yang dapat dihasilkan oleh TokenSlot tersebut, lalu dibandingkan dengan yang lebih besar saat ini. Setiap sequence dengan ukuran yang lebih pendek juga perlu dilakukan pengecekan total reward karena bisa saja TokenSlot yang lebih pendek menghasilkan total reward yang lebih besar misalnya jika ada yang rewardnya negatif. Di dalam fungsi ada juga boolean isHorizontal yang akan berubah-ubah setiap kali dipanggil. Kemudian ada pula matriks yang bisa diberikan penanda agar tidak dapat melalui token yang sudah dilalui sebelumnya.

A. Source Code Program Utama

Cracker.cpp

```
#include "cracker.h"
using namespace std;

void recursion(
    MarkableToken** matrix, int width, int height,
    TokenSlot slot, int currentSlotIdx,
    bool isHorizontal,
    int* maxReward,
    Sequence sequence[], int sequenceLength,
    int posX, int posY,
    TokenSlot* mostRewardingSlot)
{

    // base case
    // khusus for loop paling dalam, atau slot paling kanan
    if(currentSlotIdx == slot.bufferSize)
    {
        slot.filledSlot = currentSlotIdx;
        int currentTotalReward = slot.calculateReward(sequence, sequenceLength);
        if(currentTotalReward > *maxReward)
        {
```

```
            mostRewardingSlot->CopyFrom(slot);
            *maxReward = currentTotalReward;
        }

        return;
    }

    // isHorizontal udah direverse pas fungsi rekursinya dipanggil
    if(isHorizontal)
    {
        for(int j = 0; j < width; j++)
        {
            if(matrix[posY][j].isMarked) continue;

             // everytime a new mark is set, the previous needs to be unset. Like
swapping
            // indexHasBeenFilled just to make sure the value is initialized
            if(slot.indexHasBeenFilled(currentSlotIdx))

matrix[slot.slotList[currentSlotIdx].y][slot.slotList[currentSlotIdx].x].isMarked
= false;

            posX = j;
            slot.slotList[currentSlotIdx].token = matrix[posY][posX].token;
            slot.slotList[currentSlotIdx].x = posX;
            slot.slotList[currentSlotIdx].y = posY;
            matrix[posY][posX].isMarked = true;
            recursion(
                matrix, width, height,
                slot, currentSlotIdx+1,
                !isHorizontal,
                maxReward,
                sequence, sequenceLength,
                posX, posY,
                mostRewardingSlot
            );
        }
    }
    else
```

```
    {
        for(int i = 0; i < height; i++)
        {
            if(matrix[i][posX].isMarked) continue;

             // everytime a new mark is set, the previous needs to be unset. Like
swapping
            // happens when backtracking
            if(slot.indexHasBeenFilled(currentSlotIdx))

matrix[slot.slotList[currentSlotIdx].y][slot.slotList[currentSlotIdx].x].isMarked
= false;

            posY = i;
            slot.slotList[currentSlotIdx].token = matrix[posY][posX].token;
            slot.slotList[currentSlotIdx].x = posX;
            slot.slotList[currentSlotIdx].y = posY;
            matrix[posY][posX].isMarked = true;
            recursion(
                matrix, width, height,
                slot, currentSlotIdx+1,
                !isHorizontal,
                maxReward,
                sequence, sequenceLength,
                posX, posY,
                mostRewardingSlot
            );

        }
    }

    // base case khusus ga ada jalur lagi
    slot.filledSlot = currentSlotIdx;
    int currentTotalReward = slot.calculateReward(sequence, sequenceLength);
    if(currentTotalReward > *maxReward)
    {
        mostRewardingSlot->CopyFrom(slot);
        *maxReward = currentTotalReward;
    }
```

```
}

/// @brief Get the result packed inside SolveData for passing data outside dll
/// @param bufferSize
/// @param width width of matrix
/// @param height height of matrix
/// @param matrix token matrix that can be marken
/// @param sequenceLength length of sequence
/// @param sequence
/// @return
CrackData    getOptimalSolution(int    bufferSize,    int    width,    int    height,
MarkableToken** matrix, int sequenceLength, Sequence sequence[])
{

    clock_t startTime = clock();

    TokenSlot mostRewardingSlot(bufferSize);
    TokenSlot slot(bufferSize);
    int maxReward = INT_MIN;

    // Start from horizontal
    recursion(
        matrix, width, height,
        slot, 0,
        true,
        &maxReward,
        sequence, sequenceLength,
        0, 0,
        &mostRewardingSlot
    );

    // Output
    CrackData solveData = {mostRewardingSlot, maxReward, clock() - startTime};
    return solveData;
}
```

B. Source Code Program CLI

cli.cpp

```
#include <bits/stdc++.h>
#include <iostream>
#include <string>
#include <time.h>
#include <stdio.h>
#include <cstdlib>
#include "../lib/cracker.h"
using namespace std;

/// @brief Save data to path
/// @param data
/// @param path
void saveToPath(CrackData data, string path)
{
    // same as printSolveData but to file
    ofstream file(path);
    file << data.maxReward << endl;
    for(int i = 0; i < data.mostRewardingSlot.filledSlot; i++)
    {
        file << data.mostRewardingSlot.slotList[i].token << ' ';
    }
    file << endl;
    for(int i = 0; i < data.mostRewardingSlot.filledSlot; i++)
    {
        // plus 1 because it started from 1
                file  <<  data.mostRewardingSlot.slotList[i].x  +  1  <<  ",  "  <<
data.mostRewardingSlot.slotList[i].y + 1 << endl;
    }
    file << endl;
    file << data.executionDuration << " ms" << endl;
}

/// @brief ask whether user want to save data to text file or not
/// @param data
void askForSavingOutput(CrackData data)
{
```

```
    cout << endl << "Apakah ingin menyimpan solusi? (y/n): ";
    char c; cin >> c;
    if(c == 'y' || c == 'Y')
    {
        string savePath;
        cout << "Enter save path (Ex: test/output/test100.txt) " << endl;
        cin >> savePath;
        saveToPath(data, savePath);
        cout << endl;
    }
}

/// @brief Print result
/// @param data
void printSolveData(CrackData data)
{
    cout << data.maxReward << endl;
    for(int i = 0; i < data.mostRewardingSlot.filledSlot; i++)
    {
        cout << data.mostRewardingSlot.slotList[i].token << ' ';
    }
    cout << endl;
    for(int i = 0; i < data.mostRewardingSlot.filledSlot; i++)
    {
        // plus 1 because it started from 1
                cout  <<  data.mostRewardingSlot.slotList[i].x  +  1  <<  ",  "  <<
data.mostRewardingSlot.slotList[i].y + 1 << endl;
    }
    cout << endl;
    cout << data.executionDuration << " ms" << endl;
}

/// @brief start cli by typing the input manually
void startByTyping()
{
    int bufferSize;
    cin >> bufferSize;

    int width, height;
```

```
    cin >> width >> height;

    MarkableToken** matrix;
    matrix = new MarkableToken*[height];
    for(int i = 0; i < height; i++)
    {
        matrix[i] = new MarkableToken[width];
        for(int j = 0; j < width; j++)
        {
            char c[2];
            cin >> c;
            matrix[i][j].token = c;
            matrix[i][j].isMarked = false;
        }
    }

    int sequenceLength;
    cin >> sequenceLength;

    string line;
    Sequence sequence[sequenceLength];
    for(int i = 0; i < sequenceLength; i++)
    {
        cin.ignore();
        getline(cin, line);
        Token t = {line[0], line[1]};
        sequence[i].push_back(t);
        int lineLength = line.length();
        int j = 3;
        while(j < lineLength)
        {
            Token t = {line[j], line[j+1]};
            sequence[i].push_back(t);
            j += 3;
        }
        cin >> sequence[i].reward;
    }
    cout << endl << "Processing..." << endl << endl;
```

```
        CrackData data = getOptimalSolution(bufferSize, width, height, matrix,
sequenceLength, sequence);
    cout << "Result:" << endl;
    printSolveData(data);
    askForSavingOutput(data);
}

/// @brief start cli by reading the input from file
/// @param path
void startByPath(string path)
{
    ifstream file(path);
    string line;
    if(!file.is_open())
    {
        cout << "File not found" << endl;
        return;
    }
    getline(file, line);
    int bufferSize = stoi(line);

    getline(file, line);
    int pos = line.find(" ");
    string splitedLine1 = line.substr(0, pos);
    string splitedLine2 = line.substr(pos + 1, line.length());
    int width = stoi(splitedLine1);
    int height = stoi(splitedLine2);


    MarkableToken** matrix;
    matrix = new MarkableToken*[height];
    for(int i = 0; i < height; i++)
    {
        matrix[i] = new MarkableToken[width];
        getline(file, line);
        for(int j = 0; j < width; j++)
        {
            char c[2] = {line[j*3], line[j*3+1]};
            matrix[i][j].token = c;
```

```
            matrix[i][j].isMarked = false;
        }
    }

    getline(file, line);
    int sequenceLength = stoi(line);

    Sequence sequence[sequenceLength];
    for(int i = 0; i < sequenceLength; i++)
    {
        getline(file, line);
        Token t = {line[0], line[1]};
        sequence[i].push_back(t);
        int lineLength = line.length();
        int j = 3;
        while(j < lineLength)
        {
            Token t = {line[j], line[j+1]};
            sequence[i].push_back(t);
            j += 3;
        }

        getline(file, line);
        sequence[i].reward = stoi(line);
    }

    file.close();
    cout << endl << "Processing..." << endl << endl;

       CrackData data = getOptimalSolution(bufferSize, width, height, matrix,
sequenceLength, sequence);
    cout << "Result:" << endl;
    printSolveData(data);
    askForSavingOutput(data);
}

#define amountOfUniqueTokensStr "Amount of Unique Tokens: "
#define possibleTokensStr "Possible Tokens: "
#define bufferSizeStr "Buffer Size: "
```

```
#define matrixDimensionStr "Matrix Dimension (width height): "
#define sequenceAmountStr "Amount of Sequence: "
#define maximalSequenceLengthStr "Maximal Sequence Length: "
#define generatedMatrixStr "Generated Matrix: "

int randomRange(int min, int max)
{
    return rand() % (max - min + 1) + min;
}

bool isSequenceExistInListOfSequence(Sequence sequence, Sequence* sequenceList,
int sequenceListLength)
{
    for(int i = 0; i < sequenceListLength; i++)
    {
        if(!sequence.isEqual(sequenceList[i]))
        {
            return false;
        }
    }
    return true;
}

void startByAutoGenerateInput()
{
    // jumlah_token_unik
    cout << amountOfUniqueTokensStr << endl;
    int uniqueTokenCount;
    cin >> uniqueTokenCount;

    // token
    cout << possibleTokensStr << endl;
    Token possibleTokens[uniqueTokenCount];
    for(int i = 0; i < uniqueTokenCount; i++)
    {
        cin >> possibleTokens[i].value;
    }

    // ukuran_buffer
```

```
    cout << bufferSizeStr;
    int bufferSize;
    cin >> bufferSize;

    // ukuran_matriks
    cout << matrixDimensionStr << endl;
    int width, height;
    cin >> width >> height;

    // jumlah_sekuens
    cout << sequenceAmountStr;
    int sequenceAmount;
    cin >> sequenceAmount;

    // ukuran_maksimal_sekuens
    cout << maximalSequenceLengthStr;
    int maxSequenceLength;
    int minSequenceLength = 2;
    cin >> maxSequenceLength;

    // generate matrix
    MarkableToken** matrix;
    matrix = new MarkableToken*[height];

    // seed random
    srand(time(NULL));

    // generate matrix
    cout << endl << generatedMatrixStr << endl;
    for(int i = 0; i < height; i++)
    {
        matrix[i] = new MarkableToken[width];
        for(int j = 0; j < width; j++)
        {
            int randomIndex = rand() % uniqueTokenCount;
            matrix[i][j].token = possibleTokens[randomIndex];
            matrix[i][j].isMarked = false;

            cout << matrix[i][j].token << ' ';
```

```
        }
        cout << endl;
    }

    // generate sequence
    Sequence sequence[sequenceAmount];
    int i = 0;
    while(i < sequenceAmount)
    {
        int sequenceLength = randomRange(minSequenceLength, maxSequenceLength);
        for(int j = 0; j < sequenceLength; j++)
        {
            int randomIndex = rand() % uniqueTokenCount;
            sequence[i].push_back(possibleTokens[randomIndex]);
        }
        // unique check
                    if(isSequenceExistInListOfSequence(sequence[i],   sequence,
sequenceLength))
        {
            continue;
        }

        cout << endl << "Sequence " << i << ':' << endl;
        for(int j = 0; j < sequenceLength; j++)
        {
            cout << sequence[i].buffer[j] << ' ';
        }
        cout << endl;
        // range reward 10 - 50
        int reward = randomRange(10, 50);
        sequence[i].reward = reward;
        cout << "Reward: " << reward << endl;
        i++;
    }
    cout << endl << "Processing..." << endl << endl;

       CrackData  data  =  getOptimalSolution(bufferSize,  width,  height,  matrix,
sequenceAmount, sequence);
    cout << "Result:" << endl;
```

```
    printSolveData(data);
    askForSavingOutput(data);
}


#define titleStr "Cyberpunk 2077 Breach Protocol Cracker"
#define chooseOptionStr "Choose option (number): "
#define option1Str "1. Input by typing manually"
#define option2Str "2. Input from file"
#define option3Str "3. Auto generate input"
#define option4Str "4. Exit"

void cliScreen()
{
    cout << titleStr << endl << endl;
    cout << option1Str << endl;
    cout << option2Str << endl;
    cout << option3Str << endl;
    cout << option4Str << endl;
    cout << endl << chooseOptionStr;

    int option = 0;
    while(!(option >= 1 && option <= 3))
    {
        cin >> option;
        if(option == 1) startByTyping();
        else if(option == 2)
        {
            string path;
            cout << "File path: ";
            cin >> path;
            startByPath(path);
        }
        else if(option == 3) startByAutoGenerateInput();
        else if(option == 4) return;
        else if(!cin.good())
        {
            cin.clear();
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
```

```
            cout << "Please input a number!: ";
        }
        else cout << "Please choose between range 1 - 3!: ";
    }
}


int main(int argc, char* argv[])
{
    // from file
    if(argc > 1) startByPath(argv[1]);
    else cliScreen();
    return 0;
}
```

C. Source Code Program GUI

gui.py

```
import sys
from time import time

from PyQt5.QtGui import QResizeEvent
from models.token_types import Token, MarkableToken, Sequence, TokenSlot,
CrackData
from models.custom_c_types import C_Token, C_MarkableToken, C_Sequence,
C_TokenSlot, C_CrackData
from ctypes import cdll, CDLL, c_void_p, c_int, c_float, c_double, POINTER,
c_char_p, c_bool, c_char, Structure, _Pointer, Array

from models.cracker import Cracker
from models.token_types import TokenMatrix
from PyQt5.QtWidgets import QApplication, QWidget, QHBoxLayout, QVBoxLayout,
QMainWindow, QSizePolicy, QGridLayout, QLabel, QFileDialog, QScrollArea
from PyQt5 import QtWidgets

from PyQt5 import QtCore
from widgets.nicebutton import NiceButton
from widgets.normaltext import NormalText
from widgets.numberinput import NumberInput
```

```
from widgets.matrixinput import MatrixInput
from widgets.sequenceinput import SequenceInput
from widgets.vcontainer import VContainer
from widgets.sequenceinputlist import SequenceInputList
from data.data import Data
import os

class Window(QWidget):
    def resizeEvent(self, a0: QResizeEvent | None) -> None:
        self.scrollArea.setFixedWidth(self.width())
        self.scrollArea.setFixedHeight(self.height())
        return super().resizeEvent(a0)

    def on_buffer_size_changed(self, text: str):
        if text.isdigit():
            val = int(text)
            if val < 1:
                return
            self.buffer_size = val

    def on_width_changed(self, text: str):
        if text.isdigit():
            val = int(text)
            if val < 1:
                return
            self.matrix_input.set_width(val)

    def on_height_changed(self, text: str):
        if text.isdigit():
            val = int(text)
            if val < 1:
                return
            self.matrix_input.set_height(val)

    def on_sequence_amount_changed(self, text: str):
        if text.isdigit():
            val = int(text)
            if val < 1:
                return
```

```
            self.sequence_input_list.set_sequence_length(val)

    def __init__(self):
        super().__init__()
        self.buffer_size: int = 5
        self.sequence_amount: int = 2



        self.setStyleSheet("""
            background-color: rgb(2, 6, 23);
            color: white;
            font-size: 17px;
            font-weight: bold;
        """)

        self.setWindowTitle(Data.title)
        self.resize(Data.screen_width, Data.screen_height)

        grid_layout = QGridLayout()
        grid_layout.setSpacing(Data.padding_2)
        self.setLayout(grid_layout)



        self.v_layout_left = QVBoxLayout()
        self.v_layout_left.setSpacing(Data.padding_1)
        self.v_layout_right = QVBoxLayout()
        self.v_layout_right.setSpacing(Data.padding_1)
        grid_layout.addLayout(self.v_layout_left, 1, 0)
        grid_layout.addLayout(self.v_layout_right, 1, 1)
        grid_layout.setColumnStretch(0, 1)
        grid_layout.setRowStretch(1, 1)

        # Make the v_layout_left scrollable without shrinking
        self.scrollArea = QScrollArea(self)
        self.scrollArea.setVerticalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.scrollArea.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOn)
        self.scrollArea.setWidgetResizable(True)
        self.scrollArea.setMinimumHeight(Data.screen_height)
        self.scrollArea.setMinimumWidth(Data.screen_width)
```

```
        widget = QWidget()
        self.scrollArea.setWidget(widget)
        widget.setLayout(grid_layout)
        # make scroll area size always adjust according to window size
                            self.scrollArea.setSizePolicy(QSizePolicy.Expanding,
QSizePolicy.Expanding)
        self.scrollArea.setWidgetResizable(True)



        # title
        title = NormalText(None, Data.title)
        title.setStyleSheet("QLabel{font-size: 24px;}")
        title.setAlignment(QtCore.Qt.AlignCenter)
        grid_layout.addWidget(title, 0, 0, 1, 0)


        # input title
        input_title = NormalText(None, "Input:")
        self.v_layout_left.addWidget(input_title)
        input_title.setAlignment(QtCore.Qt.AlignLeft)
        input_title.setMinimumWidth(Data.screen_width//2)

        # result title
        result_title = NormalText(None, "Result:")
        self.v_layout_right.addWidget(result_title)
        result_title.setAlignment(QtCore.Qt.AlignLeft)
        result_title.setMinimumWidth(Data.screen_width//2)


        buffer_size_hcontainer = VContainer()

        # buffer_size_input
        buffer_size_layout = QHBoxLayout()
        buffer_size_label = NormalText(None, "Buffer Size: ")
        self.buffer_size_input = NumberInput(self, 5)
        self.buffer_size_input.set_allow_negative_or_zero(False)
        self.buffer_size_input.setText(str(self.buffer_size))
```

```
        self.buffer_size_input.setFixedWidth(Data.input_width_2)
        buffer_size_layout.addWidget(buffer_size_label)
        buffer_size_layout.addWidget(self.buffer_size_input)
        buffer_size_layout.addStretch()
        buffer_size_hcontainer.addLayout(buffer_size_layout)
        self.v_layout_left.addWidget(buffer_size_hcontainer)
        self.buffer_size_input.textChanged.connect(self.on_buffer_size_changed)


        dimension_hcontainer = VContainer()

        # width, height
        dimension_h_layout = QHBoxLayout()
        dimension_h_layout.setAlignment(QtCore.Qt.AlignLeft)
        width_label = NormalText(None, "Width: ")
        height_label = NormalText(None, "Height: ")
        self.width_input = NumberInput(self, 5)
        self.height_input = NumberInput(self, 5)
        self.width_input.set_allow_negative_or_zero(False)
        self.height_input.set_allow_negative_or_zero(False)
        self.width_input.setMaximumWidth(Data.input_width_1)
        self.height_input.setMaximumWidth(Data.input_width_1)
        dimension_h_layout.addWidget(width_label)
        dimension_h_layout.addWidget(self.width_input)
        dimension_h_layout.addWidget(height_label)
        dimension_h_layout.addWidget(self.height_input)
        dimension_h_layout.addStretch()
        dimension_hcontainer.addLayout(dimension_h_layout)

        # matrix
        matrix_label = NormalText(None, "Tokens: ")
        dimension_hcontainer.addWidget(matrix_label)
        matrix_h_layout = QHBoxLayout()

        self.matrix_input = MatrixInput(None)
        self.width_input.setText(str(self.matrix_input.matrix_width))
        self.height_input.setText(str(self.matrix_input.matrix_height))
        self.width_input.textChanged.connect(self.on_width_changed)
        self.height_input.textChanged.connect(self.on_height_changed)
```

```
        matrix_h_layout.addWidget(self.matrix_input)
        matrix_h_layout.addStretch()
        dimension_hcontainer.addLayout(matrix_h_layout)

        self.v_layout_left.addWidget(dimension_hcontainer)


        sequence_hcontainer = VContainer()

        # sequence_amount_input
        h_layout_sequence = QHBoxLayout()
        sequence_label = NormalText(None, "Amount of Sequence: ")
        self.sequence_amount_input = NumberInput(None, 0)
        self.sequence_amount_input.set_allow_negative_or_zero(False)
        self.sequence_amount_input.setFixedWidth(Data.input_width_1+7)
        self.sequence_amount_input.setText(str(self.sequence_amount))

self.sequence_amount_input.textChanged.connect(self.on_sequence_amount_changed)
        h_layout_sequence.addWidget(sequence_label)
        h_layout_sequence.addWidget(self.sequence_amount_input)
        h_layout_sequence.addStretch()
        sequence_hcontainer.addLayout(h_layout_sequence)
        self.v_layout_left.addWidget(sequence_hcontainer)

        self.h_layout_sequence_list = QHBoxLayout()

        self.sequence_input_list = SequenceInputList(None)
        self.sequence_input_list.set_sequence_length(self.sequence_amount)
        self.v_layout_left.addWidget(self.sequence_input_list)


        self.v_layout_left.addStretch()

        # button
        button_h_layout = QHBoxLayout()
        self.calculate_button = NiceButton(None)
        self.calculate_button.setText("Auto Generate Values")
        self.calculate_button.clicked.connect(self.on_auto_generate_clicked)
```

```
        self.v_layout_left.addWidget(self.calculate_button)
        button_h_layout.addWidget(self.calculate_button)
        self.calculate_button = NiceButton(None)
        self.calculate_button.setText("Import Values from File")
        self.calculate_button.clicked.connect(self.on_file_open_clicked)
        self.v_layout_left.addWidget(self.calculate_button)
        button_h_layout.addWidget(self.calculate_button)
        self.v_layout_left.addLayout(button_h_layout)
        button_h_layout.setSpacing(Data.padding_1)
        # button_h_layout.addStretch()


        button_h_layout = QHBoxLayout()
        self.calculate_button = NiceButton(None)
        self.calculate_button.setText("Calculate")
        button_h_layout.addWidget(self.calculate_button)
        self.v_layout_left.addLayout(button_h_layout)
        self.calculate_button.clicked.connect(self.on_calculate_clicked)

        # Result
        self.execution_time_label = NormalText(None, "")
        self.max_reward_label = NormalText(None, "")
        self.v_layout_right.addWidget(self.execution_time_label)
        self.v_layout_right.addWidget(self.max_reward_label)

        self.matrix_result_label: list[NormalText] = []

        self.v_layout_right.addStretch()

        self.crack_data: CrackData = None
        self.matrix_c_p: Array[_Pointer[C_MarkableToken]] = None

        self.button_h_layout: QHBoxLayout = None

    def validate_input(self) -> bool:
        # buffer size > 0
        # width > 0
        # height > 0
        # sequence amount > 0
```

```
        # count > 0

        # matrix is all filled with 2 character
        # token is all filled with 2 character

        self.execution_time_label.setText("Error: ") # Error message
        self.execution_time_label.set_error_style()
        return True

    def on_calculate_clicked(self):
        if not self.validate_input():
            return

        self.execution_time_label.set_default_style()
        self.calculate_button.setText("Processing...")
        self.calculate_button.setEnabled(False)


        buffer_size: int = self.buffer_size
        width: int = self.matrix_input.matrix_width
        height: int = self.matrix_input.matrix_height

        matrix_data_c_p = self.matrix_input.get_matrix_c()

        sequence_c_p = self.sequence_input_list.get_sequence_c()

        crack_data_c = Cracker.cracker.getOptimalSolution(
            buffer_size,
            width,
            height,
            matrix_data_c_p,
            self.sequence_amount,
            sequence_c_p
        )
        crack_data = CrackData(crack_data_c)
        self.draw_result(crack_data, matrix_data_c_p)

        self.calculate_button.setText("Calculate")
        self.calculate_button.setEnabled(True)
```

```
        self.crack_data = crack_data
        self.matrix_c_p = matrix_data_c_p

    def draw_result(self, crack_data: CrackData, matrix_data_c_p):
        # remove all label
        for i in self.matrix_result_label:
            i.setParent(None)
        self.matrix_result_label = []


                    self.execution_time_label.setText("Execution   Time:   "   +
str(crack_data.executionDuration) + "ms")
        self.max_reward_label.setText("Max Reward: " + str(crack_data.maxReward))

        initX = self.max_reward_label.x()
        initY = self.max_reward_label.y() + 25

        delt = 25
        marginX = 20
        marginY = 15


        for i in range(self.matrix_input.matrix_height):
            for j in range(self.matrix_input.matrix_width):
                                                label    =    NormalText(self,
Token.byteToStr(matrix_data_c_p[i][j].token.value))
                label.move(initX + j*(delt+marginX), initY + i*(delt+marginY))
                label.setStyleSheet("""
                    QLabel {
                        color: white;
                        font-size: 14px;
                        font-weight: bold;
                        background-color: rgb(30, 41, 59);
                        max-width: 25px;
                        border-radius: 10px;
                        padding: 5px;
                    }
                """)
```

```
                label.show()
                self.matrix_result_label.append(label)

        # draw lines between marks
        line_width = 10
        for i in range(crack_data.mostRewardingSlot.bufferSize-1):
            x1 = crack_data.mostRewardingSlot.slotList[i].x
            y1 = crack_data.mostRewardingSlot.slotList[i].y
            x2 = crack_data.mostRewardingSlot.slotList[i+1].x
            y2 = crack_data.mostRewardingSlot.slotList[i+1].y
            label = NormalText(self, "")

            # swap because negative line width is not supported
            if x1 > x2:
                x1, x2 = x2, x1
            if y1 > y2:
                y1, y2 = y2, y1

            line_draw_deltX = 12
            line_draw_deltY = 10
            if x1 == x2:
                  label.setGeometry(initX + x1*(delt+marginX) + line_draw_deltX,
initY + y1*(delt+marginY) + line_draw_deltY, line_width, (y2-y1)*(delt+marginY))
            elif y1 == y2:
                  label.setGeometry(initX + x1*(delt+marginX) + line_draw_deltX,
initY + y1*(delt+marginY) + line_draw_deltY, (x2-x1)*(delt+marginX), line_width)

            label.setStyleSheet("""
                QLabel{
                    background-color: rgb(203, 213, 225);
                }
            """)
            label.show()
            self.matrix_result_label.append(label)

        # draw marked token
        for i in range(crack_data.mostRewardingSlot.bufferSize):
            x = crack_data.mostRewardingSlot.slotList[i].x
            y = crack_data.mostRewardingSlot.slotList[i].y
```

```
                                                label    =    NormalText(self,
Token.byteToStr(matrix_data_c_p[y][x].token.value))
            label.move(initX + x*(delt+marginX), initY + y*(delt+marginY))
            label.setStyleSheet("""
                QLabel{
                    background-color: rgb(203, 213, 225);
                    color: rgb(2, 6, 23);
                    font-size: 14px;
                    font-weight: bold;
                    max-width: 25px;
                    border-radius: 10px;
                    padding: 5px;
                }
            """)
            label.show()
            self.matrix_result_label.append(label)

        initY = initY + self.matrix_input.matrix_height*(delt+marginY)
        for i in range(crack_data.mostRewardingSlot.bufferSize):
            x = crack_data.mostRewardingSlot.slotList[i].x
            y = crack_data.mostRewardingSlot.slotList[i].y
            label = NormalText(self, "(" + str(x+1) + ", " + str(y+1) + ")")
            label.move(initX, initY + i*25)
            label.setStyleSheet("""
                QLabel{
                    color: white;
                    font-size: 14px;
                    font-weight: bold;
                }
            """)
            label.show()
            self.matrix_result_label.append(label)

            self.draw_save_button()



    def draw_save_button(self):
        if self.button_h_layout is not None:
```

```
            for i in range(self.button_h_layout.count()):
                self.button_h_layout.itemAt(i).widget().setParent(None)
            self.v_layout_right.removeItem(self.button_h_layout)

        self.button_h_layout = QHBoxLayout()
        self.save_result_button = NiceButton(None)
        self.save_result_button.setText("Save Result to File")
        self.save_result_button.clicked.connect(self.on_save_result_clicked)
        self.button_h_layout.addWidget(self.save_result_button)
        self.v_layout_right.addLayout(self.button_h_layout)

    def on_file_open_clicked(self):
        current_path = os.path.dirname(os.path.abspath(__file__))
           fname = QFileDialog.getOpenFileName(self, 'Open file', current_path,
"Text files (*.txt)")

        if fname[0]:
            with open(fname[0], 'r') as f:
                lines = f.readlines()
                if len(lines) < 3:
                    return
                self.buffer_size = int(lines[0])
                self.buffer_size_input.setText(str(self.buffer_size))

                # 6 6
                # split by space
                dimension = lines[1].split(" ")
                width = int(dimension[0])
                height = int(dimension[1])
                self.width_input.setText(str(width))
                self.height_input.setText(str(height))
                self.matrix_input.set_width(width)
                self.matrix_input.set_height(height)

                for i in range(height): # height
                    splitted = lines[i+2].split(" ")
                    for j in range(width):
                        current_input = self.matrix_input.inputs[i*width + j]
                        current_input.setText(splitted[j][:2])
```

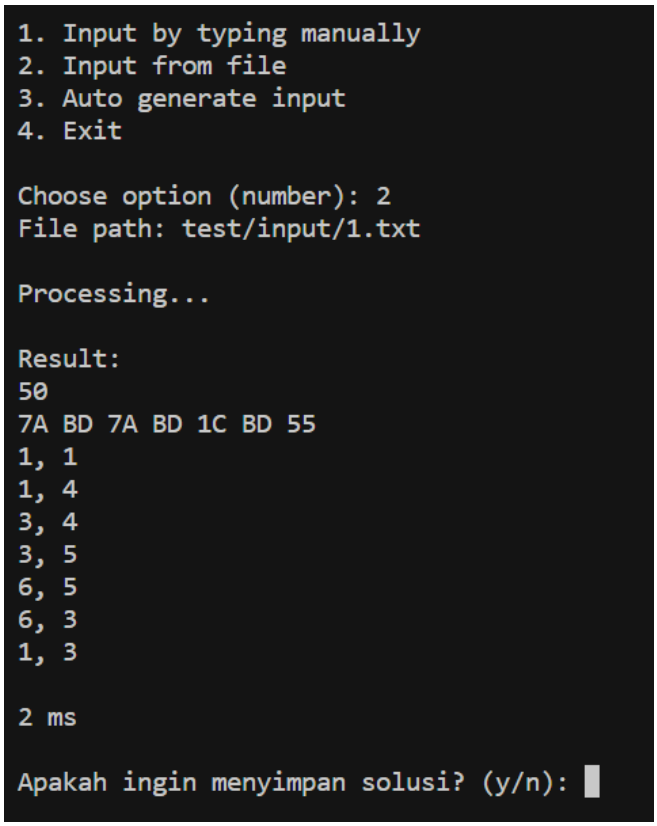
```
                self.sequence_amount = int(lines[2 + height])

self.sequence_input_list.set_sequence_length(self.sequence_amount)
                self.sequence_amount_input.setText(str(self.sequence_amount))

                for i in range(0, self.sequence_amount):
                    # input -> i
                    # lines -> i*2
                    splitted = lines[2 + height + 1 + i*2].split(" ")
                    length = len(splitted)
                    self.sequence_input_list.inputs[i].count = length

self.sequence_input_list.inputs[i].count_input.setText(str(length))
                    for j in range(0, length):
                                                                current_input   =
self.sequence_input_list.inputs[i].inputs[j]
                        current_input.setText(splitted[j][:2])
                    reward = int(lines[2 + height + 1 + i*2 + 1])

self.sequence_input_list.inputs[i].reward_input.setText(str(reward))
                    self.sequence_input_list.inputs[i].reward = reward

    def on_auto_generate_clicked(self):
        pass

    def on_save_result_clicked(self):
        current_path = os.path.dirname(os.path.abspath(__file__))
           fname = QFileDialog.getSaveFileName(self, 'Save file', current_path,
"Text files (*.txt)")


        if fname[0]:
            with open(fname[0], 'w') as f:
            # format
            # 50
            # 7A BD 7A BD 1C BD 55
            # 1, 1
            # 1, 4
```

```
            # 3, 4
            # 3, 5
            # 6, 5
            # 6, 3
            # 1, 3

            # 0 ms
                f.write(str(self.crack_data.maxReward) + "\n")
                for i in range(self.crack_data.mostRewardingSlot.bufferSize):
                    x = self.crack_data.mostRewardingSlot.slotList[i].x
                    y = self.crack_data.mostRewardingSlot.slotList[i].y

f.write(self.crack_data.mostRewardingSlot.slotList[i].token.value + " ")
                f.write("\n")
                for i in range(self.crack_data.mostRewardingSlot.bufferSize):
                    x = self.crack_data.mostRewardingSlot.slotList[i].x
                    y = self.crack_data.mostRewardingSlot.slotList[i].y
                    f.write(str(x+1) + ", " + str(y+1) + "\n")
                f.write("\n")
                f.write(str(self.crack_data.executionDuration) + "ms\n")


if __name__ == "__main__":
    Cracker.initialize()
    App = QApplication(sys.argv)
    window = Window()
    window.show()
    sys.exit(App.exec())
```

BAB IV Hasil Pengujian

A. Berbagai Kasus

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 3 Kasus 1

12
8 5
7A 55 E9 E9 1C 55 1C 7A
55 7A 1C 7A E9 55 55 7A
55 1C 1C 55 E9 BD 55 1C
BD 1C 7A 1C 55 BD 7A BD
BD 55 BD 7A 1C 1C 55 55
3
BD E9 1C
15
BD 7A BD
20
BD 1C BD 55
30

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/2.txt

Processing...

Result:
65
7A 55 55 BD E9 1C BD 7A BD 1C BD 55
1, 1
1, 2
6, 2
6, 3
5, 3
5, 5
3, 5
3, 4
6, 4
6, 5
1, 5
1, 3

3230 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 4 Kasus 2

6
3 4
7A 55 E9
55 7A 1C
55 1C 1C
BD 1C 7A
3
E9 1C 55
15
1C 55 BD
20
55 BD 1C 7A
30

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/3.txt

Processing...

Result:
65
E9 1C 55 BD 1C 7A
3, 1
3, 3
1, 3
1, 4
2, 4
2, 2

0 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 5 Kasus 3

11
3 4
7A 55 E9
55 7A 1C
55 1C 1C
BD 1C 7A
3
E9 1C 55
15
1C 55 BD
20
55 BD 1C 7A
30

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/4.txt

Processing...

Result:
35
7A 55 7A 55 E9 1C 55 BD 1C 1C
1, 1
1, 2
2, 2
2, 1
3, 1
3, 3
1, 3
1, 4
2, 4
2, 3

0 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 6 Kasus 4

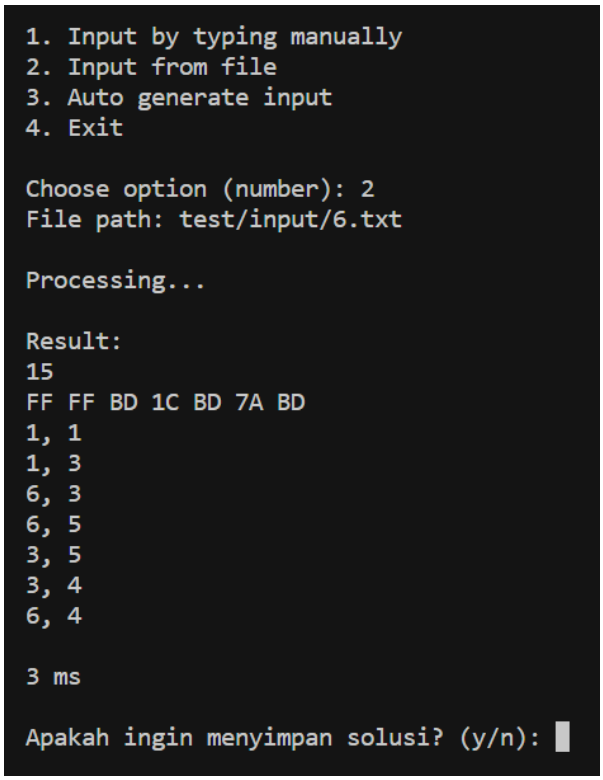
12
9 6
AA DD DD DD CC DD BB CC CC
EE BB DD CC AA BB EE AA AA
DD CC CC AA CC BB EE CC BB
AA DD BB CC EE AA EE CC AA
EE CC AA EE BB AA BB CC DD
CC DD EE AA EE AA AA DD DD
10
BB CC DD AA BB
37
AA BB CC
32
CC AA DD EE BB
33
EE EE BB CC
48
EE BB CC
32

```
179
AA EE EE BB CC DD AA BB CC BB EE CC
1, 1
1, 2
7, 2
7, 1
9, 1
9, 5
6, 5
6, 3
3, 3
3, 4
5, 4
5, 1

194562 ms
```

CC BB AA
14
BB CC BB EE
30
AA CC DD AA CC
38
AA AA CC BB
39
BB DD CC CC
31

Gambar 7 Kasus 5

7
6 6
FF FF FF FF FF 55
FF FF FF FF FF FF
FF FF FF 55 FF BD
FF FF 7A FF FF BD
FF FF BD FF FF 1C
FF FF FF FF FF FF
2
BD 7A BD
5
BD 1C BD
10



Gambar 8 Kasus 6

```
9
2 3
7A 55
55 7A
55 1C
3
55 1C 55
15
55 55 7A
20
55 7A 55
30
```

```
Cyberpunk 2077 Breach Protocol Cracker

1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/7.txt

Processing...

Result:
30
7A 55 7A 55
1, 1
1, 2
2, 2
2, 1

0 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 9 Kasus 7

```
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
```

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/8.txt

Processing...

Result:
50
7A BD 7A BD 1C BD 55
1, 1
1, 4
3, 4
3, 5
6, 5
6, 3
1, 3

3 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 10 Kasus 8

```
7
6 6
55 55 55 55 55 55
55 55 55 55 55 55
55 55 55 55 55 55
55 55 55 55 55 55
55 55 55 55 55 55
55 55 55 55 55 55
3
55 55 55 55
-100
55 55 55
15
55 55 55 55 55
30
```

```
Cyberpunk 2077 Breach Protocol Cracker

1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/9.txt

Processing...

Result:
15
55 55 55
1, 1
1, 2
2, 2

3 ms

Apakah ingin menyimpan solusi? (y/n):
```

Gambar 11 Kasus 9

```
7
7 7
PG D4 PG 7C 7C 7C 7C
PG PG D4 7C 7C 1A 7C
PG D4 1A 1A 1A D4 D4
7C 7C D4 7C D4 7C 7C
D4 7C PG PG 7C 1A 1A
D4 1A PG 1A D4 D4 1A
D4 7C 7C 1A 7C PG 7C
6
D4 7C 1A
211
PG D4 7C
125
7C PG D4 PG
220
1A PG
272
1A 7C 1A
282
PG 1A 7C
253
```

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/10.txt

Processing...

Result:
1018
D4 7C 1A PG 1A 7C 1A
2, 1
2, 7
4, 7
4, 5
7, 5
7, 2
6, 2

18 ms

Apakah ingin menyimpan solusi? (y/n):
```

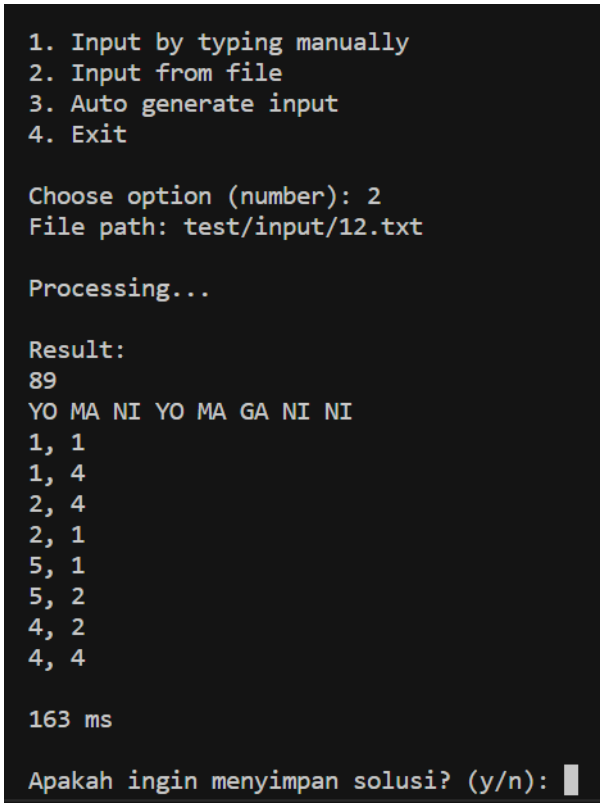
Gambar 12 Kasus 10

```
8
5 10
FF FF AA BB DD
BB FF EE EE DD
AA DD DD BB AA
EE FF CC FF EE
AA DD CC BB DD
EE AA CC CC DD
FF FF FF FF FF
EE CC EE FF EE
BB EE AA DD AA
AA DD DD DD EE
15
EE CC AA
48
EE CC
-20
AA BB FF AA BB
-87
DD CC DD
48
AA
-80
DD BB DD DD CC
85
BB AA CC
-86
AA CC BB BB DD AA
-27
CC
-28
FF BB CC
-16
EE FF CC
77
CC
72
AA
-49
AA FF DD
21
AA BB
96
```

```
1. Input by typing manually
2. Input from file
3. Auto generate input
4. Exit

Choose option (number): 2
File path: test/input/11.txt

Processing...

Result:
177
FF DD CC DD BB DD DD CC
2, 1
2, 5
3, 5
3, 3
4, 3
4, 10
2, 10
2, 8

205 ms

Apakah ingin menyimpan solusi? (y/n):
```

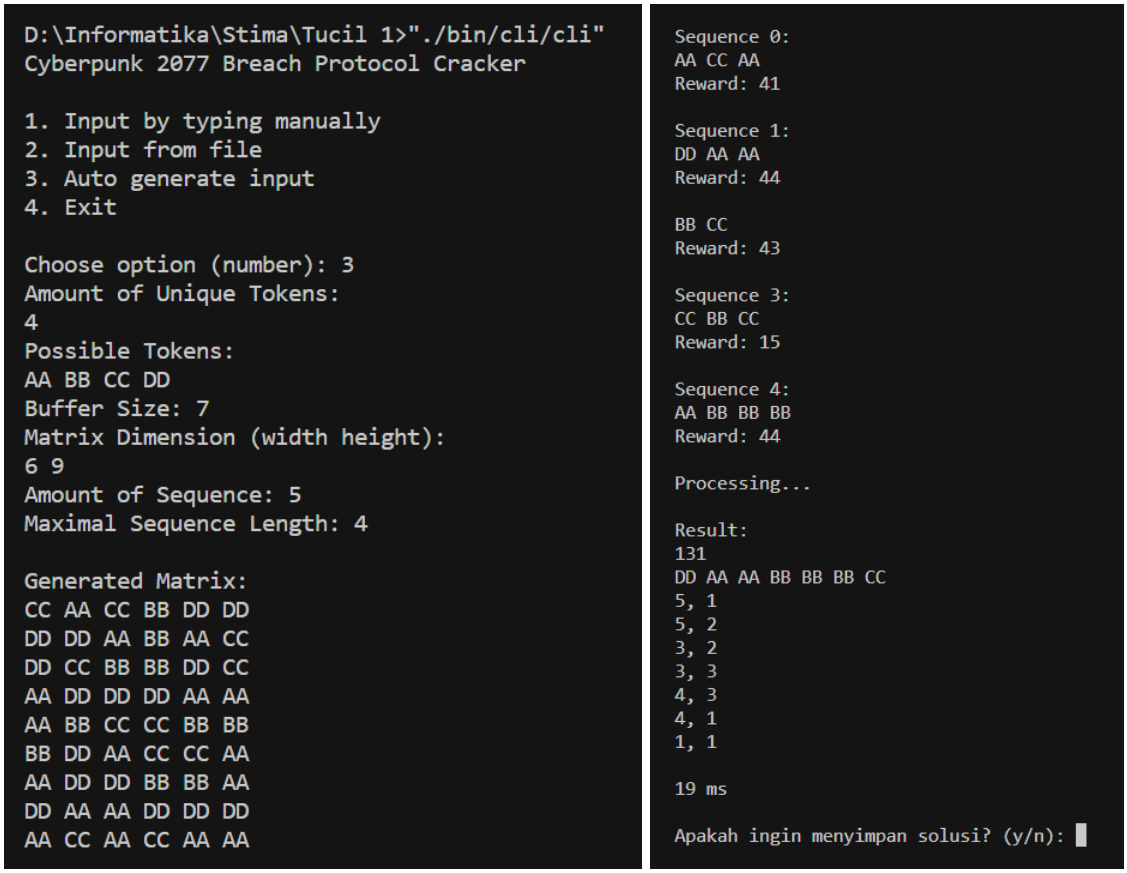
Gambar 13 Kasus 11

8
7 7
YO YO YO YO MA YO NI
YO YO YO NI GA MA GA
YO NI MA MA GA NI GA
MA NI YO NI YO MA MA
MA GA YO GA YO GA GA
YO NI NI NI YO YO MA
MA YO MA MA NI GA NI
10
NI YO YO
-64
YO YO MA
-85
MA NI
69
NI GA MA
-80
YO MA GA GA
-87
NI NI
20
MA YO MA
-28
MA MA NI
-4
YO YO
-87
MA MA YO
-29



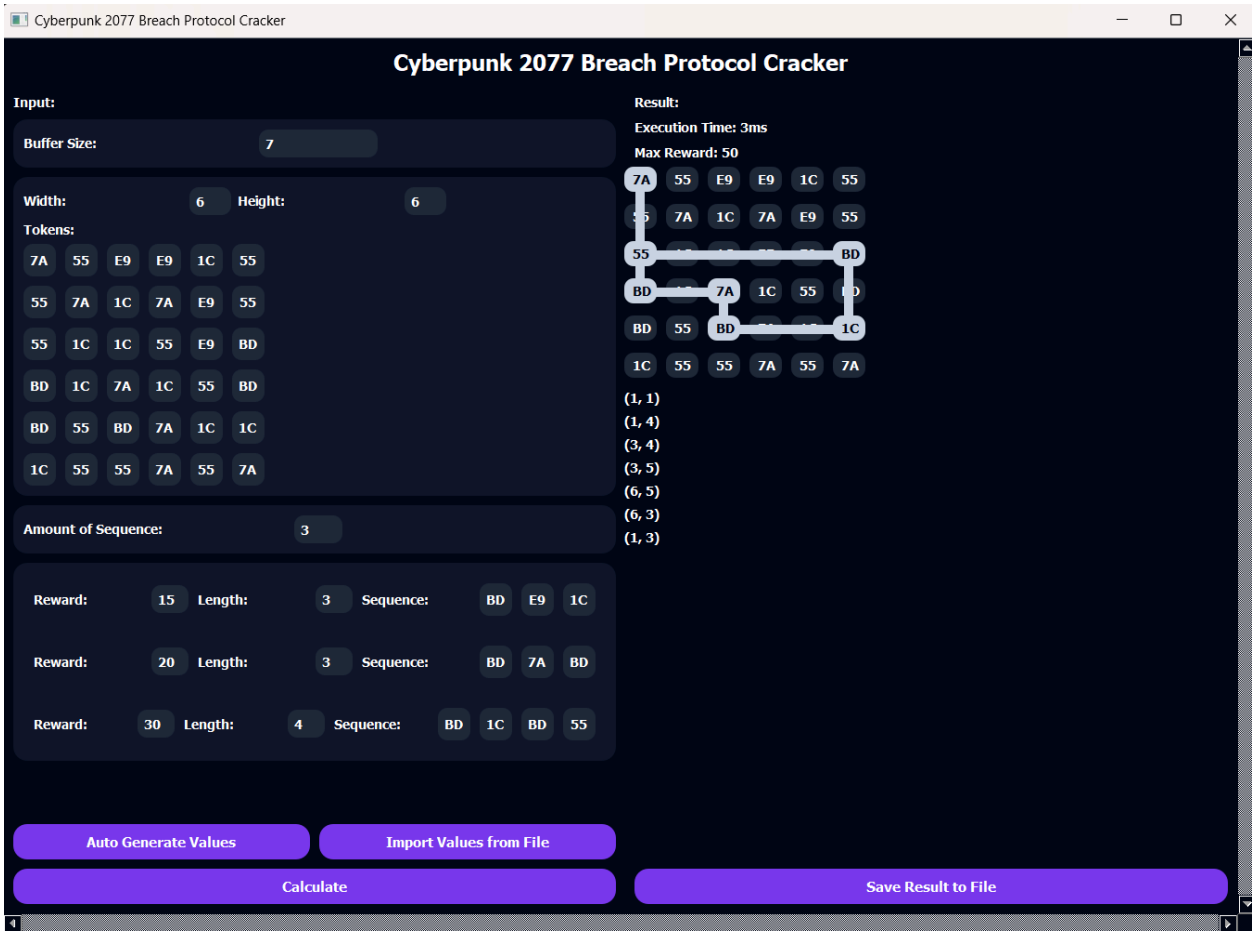
Gambar 14 Kasus 12

B. Contoh Auto Generated



Gambar 15 Contoh Auto generated

C. Tampilan GUI



Gambar 16 Tampilan GUI

BAB V Kesimpulan

Solusi untuk Minigame Cyberpunk 2077 Breach Protocol dapat diselesaikan dengan pemrograman dengan memanfaatkan algoritma bruteforce. Pada algoritma ini ditelusuri semua kemungkinan rangkaian token yang dapat dibuat, kemudian pada masing masing rangkaian token tersebut dihitung berapa total rewardnya. Hasil dari masing-masing reward ini juga akan dibandingkan mana yang maksimum.

Pendekatan Algoritma Brute force merupakan algoritma yang sering digunakan untuk menyelesaikan berbagai masalah. Tentunya terdapat algoritma lain yang lebih efektif untuk menyelesaikan berbagai persoalan selain bruteforce. Namun ada juga beberapa kasus yang membuat satu satunya solusi yang bisa digunakan adalah dengan cara bruteforce.

BAB VI Lampiran

Source code: https://github.com/DhafinFawwaz/Tucil1_13522084

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

Tabel 1 Checklist laporan