LAPORAN TUGAS KECIL I

IF2211 STRATEGI ALGORITMA

Penyelesaian *Cyberpunk 2077 Breach Protocol* dengan Algoritma Brute Force

Disusun oleh:

Raden Francisco Trianto Bratadiningrat

13522091

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2024

Daftar Isi

BAGIAN I
ALGORITMA BRUTE FORCE

Algoritma *Brute Force*, adalah algoritma dengan memecahkan suatu masalah dengan menggunakan solusi, konsep, atau cara yang paling mudah dimengerti. Algoritma ini bergantung pada kekuatan komputasi yang tinggi untuk mendapatkan semua solusi yang tepat daripada menggunakan teknik yang canggih yang jauh lebih efisien.

Pada permainan Cyberpunk 2077 Breach Protoco**l** terdapat komponen 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.

Dalam penyelesaian Cyberpunk 2077 Breach Protocol dengan pendekatan *brute force*, algoritma yang digunakan adalah sebagai berikut:

1. Buffer dapat dimisalkan adalah urutan Token yang digabungkan menjadi sebuah string.
2. Semua Buffer yang mungkin didapatkan secara rekursif dengan syarat: Langkah pertama memilih salah satu token pada baris pertama, Langkah selanjutnya bersifat vertical dilanjutkan dengan Langkah horizontal dan berulang.
3. Dengan menggunakan Pattern Matching sekuens terhadap Buffer, maka didapatkan hasil poin dari suatu Buffer.
4. Dari semua Buffer yang mungkin dilakukan pencarian yang menghasilkan poin terbanyak dengan langkah (Panjang Buffer) yang paling kecil.
5. Hasil Path ini adalah salah satu solusi optimal untuk permainan.

Algoritma yang digunakan mencoba semua kemungkinan solusi buffer yang mungkin. Semua solusi dibandingkan nilai reward total dengan mengambil hasil dengan reward paling tinggi serta Panjang buffer yang paling pendek sehingga didaptkan hasil yaitu solusi dari **Cyberpunk 2077 Breach Protocol** dengan pendekatan *brute force.*

BAB II
SOURCE PROGRAM

Program utama ditulis dalam Bahasa C++, menggunakan *library*:

5. iostream (c++)
6. fstream (c++)
7. sstream (c++)
8. filestream (c++)
9. string (c++)
10. vector (c++)
11. unordered_set (c++)
12. chrono (c++)
13. random (c++)
14. algorithm (c++)

Berikut is dari *source code* untuk main.cpp:

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <filesystem>

#include <string>
#include <vector>
#include <unordered_set>
#include <chrono>
#include <random>
#include <algorithm>

using namespace std;
using namespace chrono;

class Game{
   public:
      int bufferSize;
      int width;
      int height;
      int seqAmount;
      int maxPointsPossible;
      vector<vector<string>> matrix;
      vector<string> seq;
      vector<string> paths;
      vector<vector<vector<int>>> matrixPaths;
      vector<int> seqLen;
      vector<int> rewards;
```

```
Game() : bufferSize(0), width(0), height(0), seqAmount(0) {}

    bool solveGame(int *resPoint, string *resPath, vector<vector<int>> *resMatrixPath){
        generatePaths();


        if(paths.empty()){
            return 1;
        }

        int maxPoints = 0;
        string maxPath;
        vector<vector<int>> maxMatrixPath;

        for(int i = 0; i < paths.size(); i++){
            string path = paths[i];
            int point = pathToPoints(path);

            if(point >= maxPoints && point != 0){
                if(maxPoints == 0 || point > maxPoints){
                    maxPoints = point;
                    maxPath = cleanNonUsedBuffer(path);
                    maxMatrixPath = cleanedNonUsedMatrixPath(maxPath, matrixPaths[i]);
                } else{
                    string cleandePath = cleanNonUsedBuffer(path);

                    if(cleandePath.size() < maxPath.size()){
                        maxPoints = point;
                        maxPath = cleandePath;
                        maxMatrixPath = cleanedNonUsedMatrixPath(maxPath, matrixPaths[i]);
                    }
                }
            }
        }

        if(maxPoints == 0){
            maxPath = "No Solution";
            maxMatrixPath = {};
        }

        *resPoint = maxPoints;
        *resPath = maxPath;
        *resMatrixPath = maxMatrixPath;
        return 0;
    }
```

```
void solveGameIO(){
    auto start = high_resolution_clock::now();
    int resPoint;
    string resPath;
    vector<vector<int>> resMatrixPath;
    if(solveGame(&resPoint, &resPath, &resMatrixPath)){
        cout << "Solving failed" << endl;
        return;
    }

    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<microseconds>(stop - start);
    // duration is in microsecond

    cout << "\n\033[1;33m========== RESULT ==========\033[0m" << endl;
    cout << "Max Points: " << resPoint << endl;
    cout << "Buffer : ";
    if(resPath == "No Solution"){
        cout << resPath << endl;
    } else{
        for(int i = 0; i < resPath.size()-1; i += 2){
            cout << resPath[i] << resPath[i+1] << " ";
        } cout << endl;
    }

    cout << "paths : ";
    if(resMatrixPath.empty()){
        cout << "No Solution" << endl;
    } else{
        cout << endl;
        for(auto& coordinats: resMatrixPath){
            cout << "" << (coordinats[0] + 1) << ", "<< (coordinats[1] + 1) << endl;
        } cout << endl;
    }

    cout << "Time taken: " << int(duration.count()/1000) << " ms\n" << endl;

    while(1){
        string input;
        cout << "Save solution ? (y/n): \033[1;32m";
        cin >> input;
        cout << "\033[0m";
        if(input == "y" || input == "Y" || input == "yes" || input == "Yes"){
            stringstream outputStream;
```

```
streambuf* originalCoutBuffer = cout.rdbuf();
cout.rdbuf(outputStream.rdbuf());

cout << resPoint << endl;
if(resPath == "No Solution"){
    cout << resPath << endl;
} else{
    for(int i = 0; i < resPath.size()-1; i += 2){
        cout << resPath[i] << resPath[i+1] << " ";
    } cout << endl;
}

if(resMatrixPath.empty()){
    cout << "No Solution" << endl;
} else{
    for(auto& coordinats: resMatrixPath){
        cout << "" << (coordinats[0] + 1) << ", "<< (coordinats[1] + 1) << endl;
    }
}

cout << "Time taken: " << int(duration.count()/1000) << " ms";

cout.rdbuf(originalCoutBuffer);

saveOutput(outputStream.str());

// DATA FOR GUI
stringstream outputStreamGUI;
streambuf* originalCoutBufferGUI = cout.rdbuf();
cout.rdbuf(outputStreamGUI.rdbuf());

cout << width << " " << height << endl;
for (int i = 0; i < height; ++i) {
    for (int j = 0; j < width; ++j) {
        cout << matrix[i][j] << " ";
    }
    cout << endl;
}

for (int i = 0; i < seqAmount; ++i) {
    cout << rewards[i] << " ";
    for(int j = 0; j < seq[i].size()-1; j += 2){
        cout << seq[i][j] << seq[i][j+1] << " ";
    } cout << endl;
}
```

```
            cout.rdbuf(originalCoutBufferGUI);

            saveOutputGUI(outputStreamGUI.str());

            break;
        } else if(input == "n" || input == "N" || input == "no" || input == "No"){
            break;
        }
    }
}

void inputIO(){
    string input;
    int tokenAmount;
    int maxSeqSize;
    vector<string> uniqueToken;

    /* Unique Token Amount */
    while(1){
        cout << "\nAmount of Unique Token: \033[1;32m";
        cin >> input;
        cout << "\033[0m";

        if (stringstream(input) >> tokenAmount) {
            if (tokenAmount > 1) {
                break;
            } else {
                cout << "Invalid input. Token must be a number > 1.\n";
            }
        } else {
            cout << "Invalid input. Please enter a number.\n";
        }
    }

    /* Unique Token */
    while(1){
        cout << "Unique Token: \033[1;32m";
        for(int i = 0; i < tokenAmount; i++){
            string token;
            cin >> token;
            if(token.size() != 2){
                cout << "\033[0mInvalid Input. A token is made of 2 character.\033[1;32m" << endl;
            } else{
                uniqueToken.push_back(token);
```

```
        }
    }
    cout << "\033[0m";

    unordered_set<string> seen;
    for(int i = 0; i < tokenAmount; i++){
        seen.insert(uniqueToken[i]);
    }

    if (seen.size() < uniqueToken.size()) {
        cout << "Invalid Input. Tokens must all be unique\n";
    } else {
        break;
    }
}

/* Buffer Size*/
while(1){
    cout << "Buffer Size: \033[1;32m";
    cin >> input;
    cout << "\033[0m";

    if (stringstream(input) >> bufferSize) {
        if (bufferSize >= 0) {
            break;
        } else {
            cout << "Invalid input. Buffer size must be positif.\n";
        }
    } else {
        cout << "Invalid input. Please enter a number.\n";
    }
}

/* Matrix Size*/
while(1){
    string input2;
    cout << "Matrix size: \033[1;32m";
    cin >> input >> input2;
    cout << "\033[0m";

    if (stringstream(input) >> width && stringstream(input2) >> height) {
        if (width >= 1 && height >= 1) {
            break;
        } else {
            cout << "Invalid input. Matrix Width must be positif.\n";
```

```
            }
        } else {
            cout << "Invalid input. Please enter a number.\n";
        }
    }

    /* Sequence Amount*/
    while(1){
        cout << "Sequence Amount: \033[1;32m";
        cin >> input;
        cout << "\033[0m";

        if (stringstream(input) >> seqAmount) {
            if (seqAmount >= 1) {
                break;
            } else {
                cout << "Invalid input. Max Sequence Size must be positif.\n";
            }
        } else {
            cout << "Invalid input. Please enter a number.\n";
        }
    }

    /* Max Sequence Size*/
    while(1){
        cout << "Max Sequence Size: \033[1;32m";
        cin >> input;
        cout << "\033[0m";

        if (stringstream(input) >> maxSeqSize) {
            if (maxSeqSize >= 1) {
                break;
            } else {
                cout << "Invalid input. Max Sequence Size must be positif.\n";
            }
        } else {
            cout << "Invalid input. Please enter a number.\n";
        }
    }

    genMatrix(uniqueToken);
    genSeq(uniqueToken, maxSeqSize);
}

private:
```

```
void saveOutput(string outputStream) {
    string baseFilename = "output";
    string filename = "../test/" + baseFilename + ".txt";

    ofstream file(filename);
    file << outputStream;
    file.close();

    cout << "Solution saved to " << filename << endl;
}

void saveOutputGUI(string outputStream) {
    string filename = "../bin/temp";

    ofstream file(filename);
    file << outputStream;
    file.close();
}

void generatePaths(){
    if(paths.empty()){
        genPaths(0, 0, "", {});
    }
}

void genPaths(int currBuffer, int lastSignificantIndex, string currPath, vector<vector<int>> seenPath){
    if(currBuffer == bufferSize){
        paths.push_back(currPath);
        matrixPaths.push_back(seenPath);
        return;
    }

    if(currBuffer == 0){
        // first move
        for(int x = 0; x < width; x++){
            genPaths(1, x, matrix[0][x], {{x,0}});
        }

    } else if(currBuffer % 2 == 1){
        // move vertical
        for(int y = 0; y < height; y++){
            bool seen = false;
            for(auto& coordinats : seenPath){
                if(coordinats[0] == lastSignificantIndex && coordinats[1] == y){
                    seen = true;
```

```
                break;
            }
        }
        if(seen){
            continue;
        }
        seenPath.push_back({lastSignificantIndex, y});
        genPaths(currBuffer + 1, y, currPath + matrix[y][lastSignificantIndex], seenPath);
        seenPath.pop_back();
    }

} else{
    // move horizontal
    for(int x = 0; x < width; x++){
        bool seen = false;
        for(auto& coordinats : seenPath){
            if(coordinats[0] == x && coordinats[1] == lastSignificantIndex){
                seen = true;
                break;
            }
        }
        if(seen){
            continue;
        }
        seenPath.push_back({x, lastSignificantIndex});
        string temp = currPath + matrix[lastSignificantIndex][x];
        genPaths(currBuffer + 1, x, temp, seenPath);
        seenPath.pop_back();
    }
}
}

void genSeq(vector<string> uniqueToken, int maxSeqSize){
    for(int i = 0; i < seqAmount; i++){
        seqLen.push_back((rand() % (maxSeqSize)) + 1);
        rewards.push_back((rand() % (40)) + 10);

        string temp;
        for(int j = 0; j < seqLen[i]; j++){
            temp += uniqueToken[rand() % uniqueToken.size()];
        }
        seq.push_back(temp);
    }

    for(int i = 0; i < seqAmount; i++){
```

```
        seqLen[i] = seqLen[i]*2;
    }
}

void genMatrix(vector<string> uniqueToken){
    for(int i = 0; i < height; i++){
        vector<string> row;
        for(int j = 0; j < width; j++){
            row.push_back(uniqueToken[rand() % uniqueToken.size()]);
        }
        matrix.push_back(row);
    }
}

int pathToPoints(string path){
    int points = 0;
    int pathLen = path.size();
    bool seqUsed[seqAmount] {false};

    int i = 0;
    while(i < pathLen){
        for(int j = 0; j < seqAmount; j++){

            if(path[i] == seq[j][0] && !seqUsed[j] && pathLen-i+1 >= seqLen[j]){
                int k = 1;
                for(; k < seqLen[j]; k++){
                    if(path[i+k] != seq[j][k]){
                        break;
                    }
                }

                if(k == seqLen[j]){
                    points += rewards[j];
                    seqUsed[j] = true;
                }
            }
        }

        i++;
    }

    return points;
}

string cleanNonUsedBuffer(string path){
```

```
        int lastIdx = 0;
        int points = 0;
        int pathLen = path.size();
        bool seqUsed[seqAmount] {false};
        int i = 0;
        while(i < pathLen){
            for(int j = 0; j < seqAmount; j++){
                if(path[i] == seq[j][0] && !seqUsed[j] && pathLen-i+1 >= seqLen[j]){
                    int k = 1;
                    for(; k < seqLen[j]; k++){
                        if(path[i+k] != seq[j][k]){
                            break;
                        }
                    }

                    if(k == seqLen[j]){
                        lastIdx = i+k;
                        seqUsed[j] = true;
                    }
                }
            }

            i++;
        }

        return path.erase(lastIdx,path.size()-1);
    }

    vector<vector<int>> cleanedNonUsedMatrixPath(string cleanedBuffer, vector<vector<int>> matrixPath){
        while(matrixPath.size()*2 > cleanedBuffer.size()){
            matrixPath.pop_back();
        }
        return matrixPath;
    }
};

bool readFile(Game& game) {
    string fileName;

    while (true) {
        cout << "\nFile name: \033[1;32m";
        cin >> fileName;
        cout << "\033[0m";

        if (filesystem::exists("../test/" + fileName)) {
```

```
            break;
        } else {
            cerr << "Error: File not found. Please try again." << endl;
        }
    }

    ifstream MyReadFile("../test/" + fileName);

    if (MyReadFile.is_open()) {
        string line;
        int lineNumber = 1;

        while (getline(MyReadFile, line)) {
            stringstream ss(line);
            string token;

            // Parse buffer_size
            if (lineNumber == 1) {
                int bSize;

                if (!(ss >> bSize)) {
                    cerr << "Error parsing buffer size on line " << lineNumber << endl;
                    return 0;
                }

                char remainingChar;
                if (ss >> remainingChar) {
                    cerr << "Error parsing buffer size on line " << lineNumber << endl;
                    return 0;
                }

                if(bSize < 0){
                    cerr << "Error buffer size must be non-negatif" << endl;
                    return 0;
                }

                game.bufferSize = bSize;
            }

            // Parse matrix_width and matrix_height
            else if (lineNumber == 2) {
                if (!(ss >> game.width >> game.height)) {
                    cerr << "Error parsing matrix dimensions on line " << lineNumber << endl;
                    return 0;
                }
```

```
    char remainingChar;
    if (ss >> remainingChar) {
        cerr << "Error parsing matrix dimensions on line " << lineNumber << endl;
        return 0;
    }

    if(game.width <= 0 || game.height <= 0){
        cerr << "Error matrix size must be a positif number" << endl;
        return 0;
    }

    game.matrix.resize(game.height, vector<string>(game.width));
}

// Parse matrix elements
else if (lineNumber <= game.height + 2) {
    int row = lineNumber - 3;
    int col = 0;
    while (ss >> token) {
        if (col >= game.width) {
            cerr << "Error: Incorrect number of elements on line" << lineNumber << endl;
            return 0;
        }

        if(token.size() != 2){
            cerr << "Error parsing matrix on line " << lineNumber << endl;
            return 0;
        }
        game.matrix[row][col++] = token;
    }

}

// Parse number_of_sequences
else if (lineNumber == game.height + 3) {
    if (!(ss >> token) || !(stringstream(token) >> game.seqAmount)) {
        cerr << "Error parsing amount of sequance on line " << lineNumber << endl;
        return 0;
    }

    char remainingChar;
    if (ss >> remainingChar) {
        cerr << "Error parsing amount of sequance on line " << lineNumber << endl;
        return 0;
```

```
        }

        if(game.seqAmount <= 0){
            cerr << "Error amount of sequance must be at least 1" << endl;
            return 0;
        }
    }

    // Parse sequences and rewards
    else if (lineNumber >= game.height + 4 && lineNumber <= game.height + 4 + 2 * game.seqAmount) {
        int seqIndex = (lineNumber - game.height - 4) / 2;
        bool isSequence = (lineNumber - game.height - 4) % 2 == 0;

        if (isSequence) {
            string sequenceStr;
            while (ss >> token) {
                if(token.size() != 2){
                    cerr << "Error parsing sequance on line " << lineNumber << endl;
                    return 0;
                }
                sequenceStr += token;
            }
            game.seq.push_back(sequenceStr);
        } else {
            int reward;
            if (!(ss >> reward)){
                cerr << "Error parsing reward for sequence " << seqIndex << endl;
                return 0;
            }
            game.rewards.push_back(reward);

            char remainingChar;
            if (ss >> remainingChar) {
                cerr << "Error parsing sequance reward on line " << lineNumber << endl;
                return 0;
            }
        }

    } else {
        cerr << "Error: Unexpected line content on line " << lineNumber << endl;
        return 0;
    }

    lineNumber++;
}
```

```
        MyReadFile.close();
    } else {
        cerr << "Error opening file!" << endl;
        return 0;
    }

    // Get sequnce string len
    for(int i = 0; i < game.seqAmount; i++){
        game.seqLen.push_back(game.seq[i].size());
    }

    // Get max points possible
    game.maxPointsPossible = 0;
    for(int i = 0; i < game.seqAmount; i++){
        game.maxPointsPossible += game.rewards[i];
    }

    return 1;

}

bool readGameFromFile(const string& filename, Game& game) {
    // Check file extension
    if (filename.substr(filename.find_last_of('.') + 1) != "txt") {
        cerr << "Error: Only .txt files are supported." << endl;
        return false;
    }

    ifstream MyReadFile(filename);
    if (!MyReadFile.is_open()) {
        cerr << "Error opening file!" << endl;
        return false;
    }

    string line;
    int lineNumber = 1;

    while (getline(MyReadFile, line)) {
        stringstream ss(line);

        // Parse buffer_size
        if (lineNumber == 1) {
            int bSize;
            if (!(ss >> bSize)) {
```

```
            cerr << "Error parsing buffer_size on line " << lineNumber << endl;
            return false;
        }
        game.bufferSize = bSize;
    }

    // Parse matrix_width and matrix_height
    else if (lineNumber == 2) {
        int w, h;
        if (!(ss >> w >> h)) {
            cerr << "Error parsing matrix dimensions on line " << lineNumber << endl;
            return false;
        }
        game.width = w;
        game.height = h;
        game.matrix.resize(h, vector<string>(w));
    }

    // Parse matrix elements
    else if (lineNumber <= game.height + 2) {
        int row = lineNumber - 3;
        int col = 0;
        string token;
        while (ss >> token) {
            if (col >= game.width) {
                cerr << "Error: Too many elements in row " << row << endl;
                return false;
            }
            game.matrix[row][col++] = token;
        }
        if (col != game.width) {
            cerr << "Error: Missing elements in row " << row << endl;
            return false;
        }
    }

    // Parse number_of_sequences
    else if (lineNumber == game.height + 3) {
        int seqNum;
        if (!(ss >> seqNum)) {
            cerr << "Error parsing number_of_sequences on line " << lineNumber << endl;
            return false;
        }
        game.seqAmount = seqNum;
    }
```

```
        // Parse sequences and rewards
        else if (lineNumber >= game.height + 4 && lineNumber <= game.height + 4 + 2 * game.seqAmount) {
            int seqIndex = (lineNumber - game.height - 4) / 2;
            bool isSequence = (lineNumber - game.height - 4) % 2 == 0;

            if (isSequence) {
                game.seq.push_back(line); // Store entire sequence string
            } else {
                int reward;
                if (!(ss >> reward)) {
                    cerr << "Error parsing reward for sequence " << seqIndex << endl;
                    return false;
                }
                game.rewards.push_back(reward);
            }
        }

        else {
            cerr << "Error: Unexpected line content on line " << lineNumber << endl;
            return false;
        }

        lineNumber++;
    }

    MyReadFile.close();
    return true;
}

int main(){
    while(1){
        string inputMethod;
        Game game;

        cout << "\n====== Welcome to Breach Protocol Solver ======" << endl;
        cout << "Pick your method of input:\n1. Text File\n2. CLI\n3. Exit" << endl;
        cout << "\033[1;32m>>> \033[0m";
        cin >> inputMethod;

        if(inputMethod == "1"){
            if(!readFile(game)){
                cout << "File read failed.\n" << endl;
            } else{
                game.printGameVar();
```

```
            game.solveGameIO();
        }
    } else if (inputMethod == "2"){
        game.inputIO();
        game.printGameVar();
        game.solveGameIO();
    } else if (inputMethod == "3"){
        break;
    } else{
        cout << "Invalid input. Please input the number.\n" << endl;
        game.printGameVar();
    }
}

cout << "Program exited" << endl;
return 0;
}
```

BAGIAN III
SCREENSHOT HASIL TEST PROGRAM

Input file (test1.txt):

```
6
4 7
C4 AK 47 C4
47 47 09 09
AK C4 47 47
09 09 AK AK
AK 09 47 11
AK 47 AK AK
AK AK AK C4
5
09 C4
10
47 09 11
0
11 09
35
AK AK AK 09
15
C4 AK 47
5
```

Input file (test2.txt):

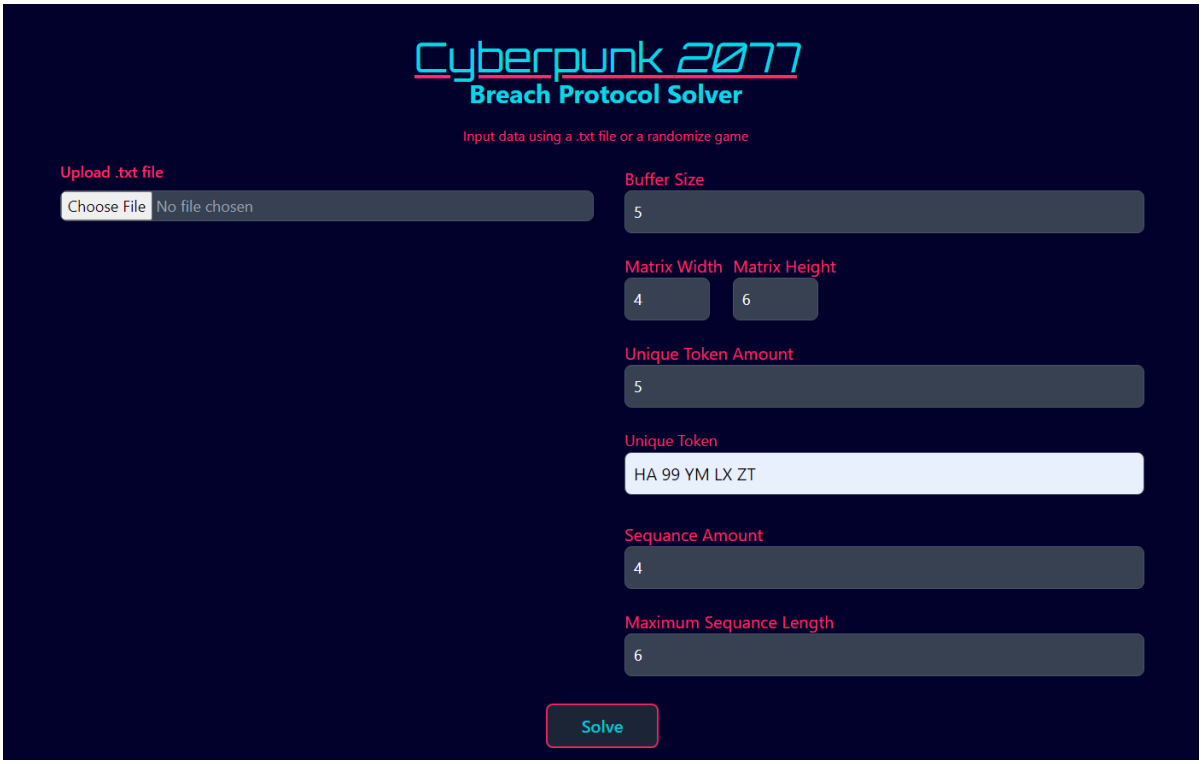
```
5
5 5
BB BB AA FF DD
CC DD DD CC DD
BB FF EE FF CC
EE DD DD BB FF
DD BB EE EE FF
4
AA EE EE
12
EE CC DD
34
EE CC AA
18
CC DD
43
```

Input file (test3.txt):

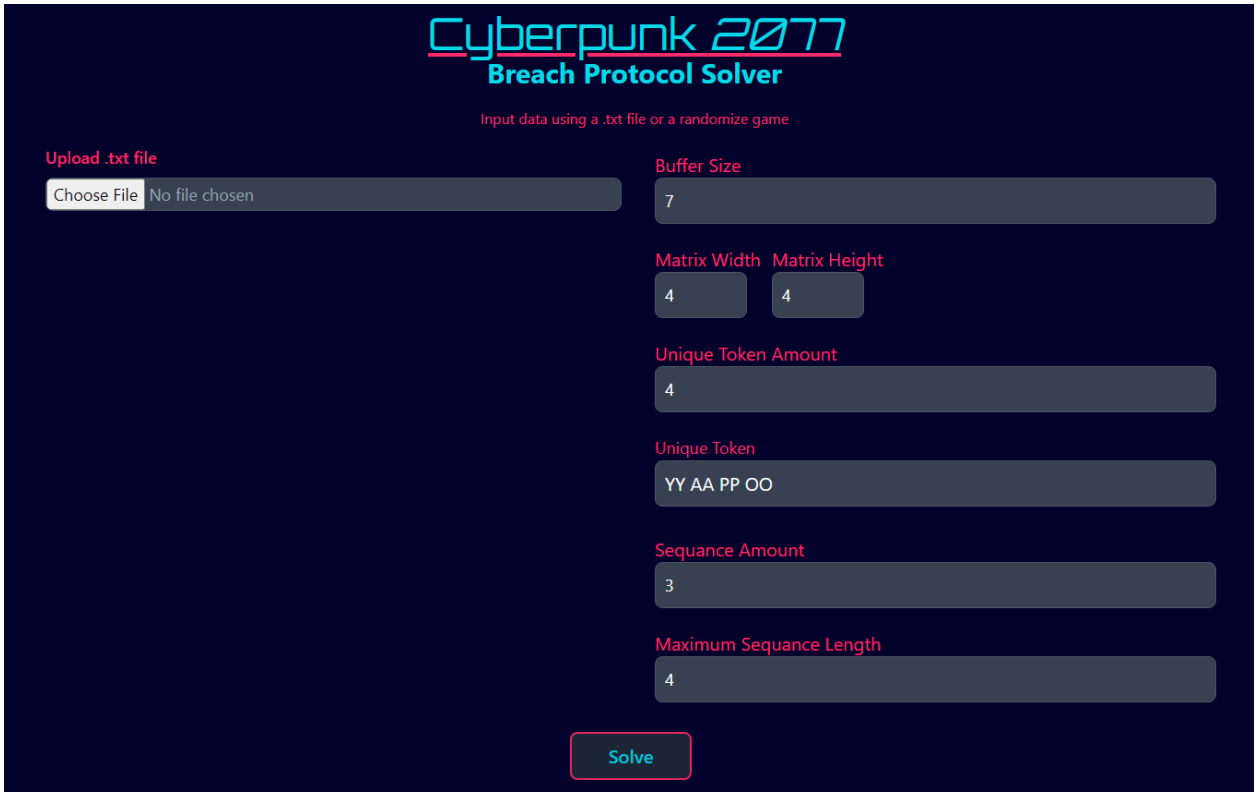
```
6
7 7
99      YM      ZT      HA      ZT      ZT      LX
LX      YM      ZT      HA      HA      99      YM
99      99      HA      YM      YM      99      99
ZT      YM      LX      YM      YM      99      99
LX      HA      YM      99      99      LX      ZT
YM      YM      ZT      HA      ZT      LX      99
YM      LX      LX      ZT      99      99      LX
5
HA      ZT      ZT      LX      LX      YM
24
HA      99      YM
35
HA      YM
21
99      ZT      YM      LX
46
```

99 99 LX HA YM
32

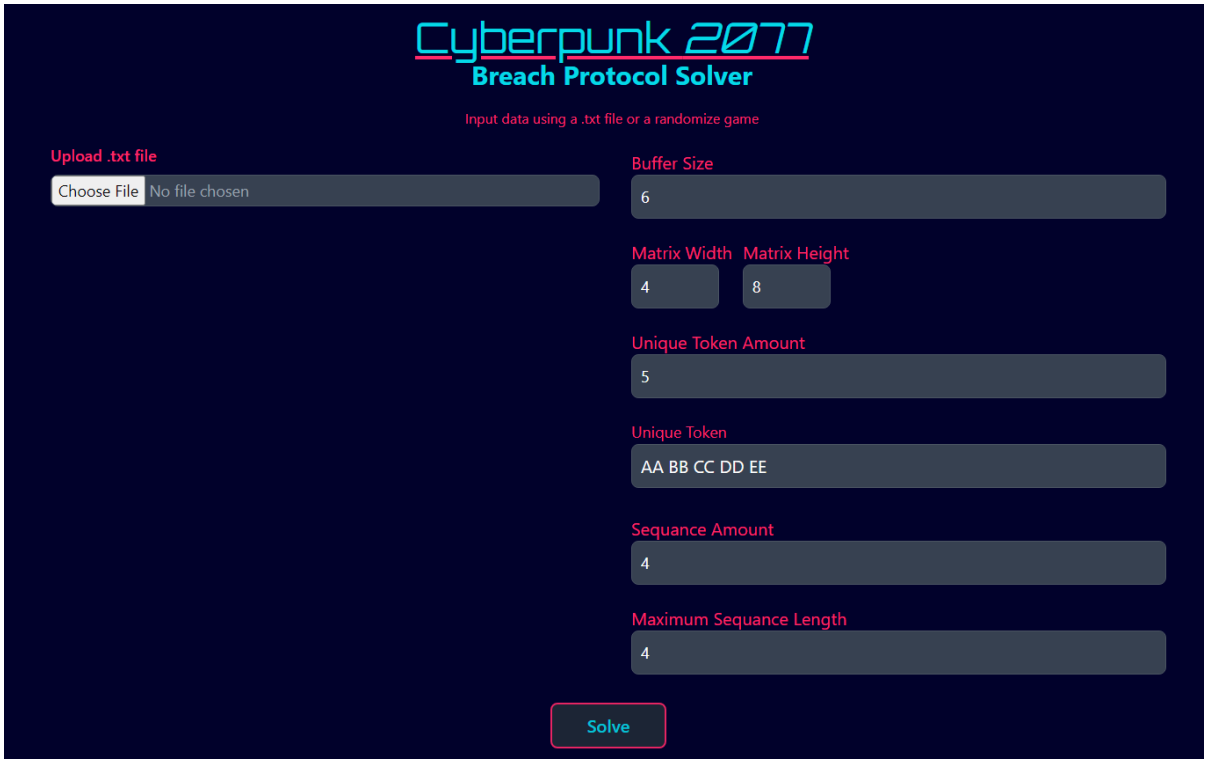
Input Ketik (input1.txt):



Input Ketik (input2.txt):



Input Ketik (input3.txt):



Input tanpa GUI dengan file:

```
====== Welcome to Breach Protocol Solver ======
Pick your method of input:
1. Text File
2. CLI
3. Exit
>>> 1

File name: test2.txt

========== Game Variables ==========
> bufferSize: 5
> width: 5
> height: 5
> seqAmount: 4
> matrix:
   BB BB AA FF DD
   CC DD DD CC DD
   BB FF EE FF CC
   EE DD DD BB FF
   DD BB EE EE FF
> sequences:
   reward: 12, seq: AA EE EE
   reward: 34, seq: EE CC DD
   reward: 18, seq: EE CC AA
   reward: 43, seq: CC DD


========== RESULT ==========
Max Points: 89
Buffer : AA EE EE CC DD
paths :
3, 1
3, 5
4, 5
4, 2
2, 2

Time taken: 6 ms

Save solution ? (y/n): y
Solution saved to ../test/output.txt
```

```
====== Welcome to Breach Protocol Solver ======
Pick your method of input:
1. Text File
2. CLI
3. Exit
>>> 1

File name: test1.txt

========== Game Variables ==========
> bufferSize: 10
> width: 4
> height: 7
> seqAmount: 5
> matrix:
   C4 AK 47 C4
   47 47 09 09
   AK C4 47 47
   09 09 AK AK
   AK 09 47 11
   AK 47 AK AK
   AK AK AK C4
> sequences:
   reward: 10, seq: 09 C4
   reward: 0, seq: 47 09 11
   reward: 35, seq: 11 09
   reward: 15, seq: AK AK AK 09
   reward: 5, seq: C4 AK 47


========== RESULT ==========
Max Points: 65
Buffer : C4 AK 47 11 09 AK AK AK 09 C4
paths :
1, 1
1, 3
4, 3
4, 5
2, 5
2, 7
3, 7
3, 4
2, 4
2, 3

Time taken: 6373 ms

Save solution ? (y/n):
```

Input tanpa GUI dengan ketik:

```
====== Welcome to Breach Protocol Solver ======
Pick your method of input:
1. Text File
2. CLI
3. Exit
>>> 2

Amount of Unique Token: 4
Unique Token: AK BB IO WX
Buffer Size: 6
Matrix size: 6 3
Sequence Amount: 4
Max Sequence Size: 4

========== Game Variables ==========
> bufferSize: 6
> width: 6
> height: 3
> seqAmount: 4
> matrix:
   BB WX IO AK BB AK
   IO IO IO AK BB BB
   BB WX BB WX WX IO
> sequences:
   reward: 46, seq: WX AK IO BB
   reward: 32, seq: BB
   reward: 48, seq: WX
   reward: 16, seq: WX IO BB AK


========== RESULT ==========
Max Points: 126
Buffer : BB BB WX
paths :
1, 1
1, 3
4, 3

Time taken: 4 ms

Save solution ? (y/n): y
Solution saved to ../test/output.txt
```

LINK REPOSITORY

https://github.com/NoHaitch/Tucil1_13522091

CHECKLIST

No.	Poin	Ya	Tidak
1.	Program dapat dikompilasi tanpa kesalahan	✓	
2.	Program berhasil dijalankan	✓	
3.	Program dapat membaca masukan berkas.txt	✓	
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	✓	