LAPORAN TUGAS KECIL I

IF2211 Strategi Algoritma

"Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force"

Dosen:

Ir. Rila Mandala, M. Eng, Ph. D.

Monterico Adrian, S. T., M. T.

Disusun Oleh:

13522140 Yasmin Farisah Salma

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER II TAHUN 2023/2024

KATA PENGANTAR

Puji syukur kepada Tuhan Yang Maha Esa penulis ucapkan atas kesempatan dan keberhasilan dalam menyelesaikan Tugas Kecil 1 IF2211 Strategi Algoritma, Semester II tahun 2023/2024, yang berjudul "Penyelesaian Cyberpunk 2077 Breach Protocol dengan Algoritma Brute Force". Laporan ini merupakan dokumentasi komprehensif dari proses pengembangan program yang dirancang untuk menemukan solusi paling optimal dalam permainan Breach Protocol dari https://cyberpunk-hacker.com/ menggunakan pendekatan algoritma *brute force*.

Tugas ini tidak hanya menantang penulis secara akademis, tetapi juga memberikan pengalaman belajar yang menarik dalam mengaplikasikan teori algoritma pada kasus nyata yang cukup kompleks. Penulis berusaha keras untuk mengembangkan solusi yang efisien dan efektif, mulai dari tahap *design* algoritma hingga implementasi *code* dan pengujian.

Dengan dukungan dari dosen, asisten kelompok, input dari rekan mahasiswa, dan berbagai sumber daya yang tersedia, penulis telah menyelesaikan program untuk tugas kecil ini yang diharapkan dapat bekerja dengan baik dan efisien. Akhir kata, penulis mengucapkan terima kasih kepada semua yang telah berpartisipasi dan mendukung. Penulis berharap agar tugas besar ini dapat memenuhi mata kuliah Strategi Algoritma dan dapat memberikan wawasan yang bermanfaat bagi pembaca serta menjadi sumber inspirasi untuk inovasi lebih lanjut.

Sumedang, 12 Februari 2024,

Yasmin Farisah Salma.

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

1.1. Algoritma Brute Force

Algoritma *brute force* adalah sebuah pendekatan yang langsung (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan. Algoritma brute force memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*).

Kelebihan Algoritma Brute Force:

1) Algoritma brute force dapat digunakan untuk memecahkan hampir sebagian besar masalah.
2) Sederhana dan mudah dimengerti.
3) Menghasilkan algoritma yang layak untuk beberapa masalah penting seperti pencarian, pengurutan, pencocokan string, perkalian matriks.
4) Menghasilkan algoritma baku (standar) untuk tugas-tugas komputasi seperti penjumlahan/perkalian N buah bilangan, menentukan elemen minimum atau maksimum ditabel.

Kelemahan Algoritma Brute Force:

1) Jarang menghasilkan algoritma yang mangkus/efektif.
2) Lambat sehingga tidak dapat diterima.
3) Tidak sekreatif teknik pemecahan masalah lainnya.

Cara Kerja Algoritma Brute Force:

1) Definisikan masalah yang ingin diselesaikan dan parameter-parameter yang terlibat.
2) *Generate* semua kemungkinan solusi tanpa mempertimbangkan keefisienan.

3) Evaluasi setiap solusi untuk memeriksa apakah memenuhi kriteria keberhasilan.
4) Pilih solusi yang paling optimal atau memenuhi kriteria terbaik dari semua solusi yang dihasilkan.
5) Implementasikan solusi yang dipilih dan analisis performa serta keefisiensinya.
6) Optimalkan algoritma jika diperlukan untuk meningkatkan performa atau mengurangi kompleksitasnya.
7) Uji coba algoritma pada berbagai kasus uji untuk memastikan hasil yang benar dan sesuai dengan ekspektasi.
8) Ulangi proses jika diperlukan untuk meningkatkan performa atau menangani masalah baru yang muncul.
9) Gunakan algoritma *brute force* secara efektif untuk menyelesaikan masalah, dengan memperhitungkan kebutuhan dan keterbatasan yang ada.

1.2. Cyberpunk 2077 Breach Protocol

Breach Protocol dalam Cyberpunk 2077 adalah sebuah minigame yang mensimulasikan proses meretas jaringan lokal dari ICE (Intrusion Countermeasures Electronics) dalam permainan tersebut. Komponen utama dalam minigame ini termasuk Token, Matriks, Sekuens, dan Buffer. Token adalah karakter alfanumerik seperti E9, BD, dan 55. Matriks adalah kumpulan token yang akan dipilih untuk menyusun urutan kode. Sekuens adalah rangkaian token yang harus dicocokkan, sedangkan Buffer adalah jumlah maksimal token yang dapat disusun secara sekuensial.

Aturan permainan Breach Protocol mencakup langkah-langkah seperti pemain bergerak secara bergantian horizontal dan vertikal hingga semua sekuens berhasil dicocokkan atau buffer mencapai kapasitas maksimum. Pemain memulai dengan memilih satu token dari baris teratas matriks, lalu

sekuens dicocokkan dengan token yang ada di buffer. Satu token dalam buffer dapat digunakan untuk lebih dari satu sekuens. Setiap sekuens memiliki bobot hadiah atau reward yang berbeda.

1.3. Langkah-Langkah Penyelesaian CyberPunk 2077 Breach Protocol dengan Pendekatan Brute Force

Langkah-langkah penyelesaian CyberPunk 2077 Breach Protocol dengan algoritma *brute force* yang digunakan pada program ini adalah sebagai berikut:

1) Mulai dengan meminta input dari pengguna, baik itu secara manual, secara *random*, atau dari *file* berekstenti .txt. Input yang diambil termasuk ukuran *buffer*, dimensi matriks, elemen-elemen matriks, jumlah sekuens, dan sekuens beserta hadiahnya.
2) Setelah input diterima, program mulai menginisialisasi variabel global yang diperlukan untuk melacak maksimum *reward*, *path* dari maksimum *reward*, dan semua *path* yang mungkin.
3) Setelah inisialisasi, program memulai iterasi melalui setiap token pada baris paling atas matriks dan melakukan gerakan vertikal dan horizontal secara berturut-turut sebanyak jumlah *buffer*.
4) Untuk setiap token, program memulai pencarian berbasis *brute force* dengan mencoba semua kemungkinan jalur yang dapat diambil dengan menggunakan fungsi **`findPaths`**.
5) Selama pencarian, program menyimpan setiap jalur yang ditemukan ke dalam **`allPaths`**.
6) Setiap kali ukuran *path* mencapai ukuran *buffer*, program memeriksa total *reward* yang diperoleh dari jalur tersebut dengan menggunakan fungsi **`calculateReward`**.

7) Jika total *reward* dari jalur tersebut lebih besar dari total *reward* maksimum yang ditemukan sebelumnya, program memperbarui total reward maksimum dan jalur maksimum reward.
8) Setelah semua *path* telah dieksplorasi, program mencetak jalur maksimum reward bersama dengan total rewardnya, serta koordinat dari setiap token dalam jalur.
9) Program juga mencatat waktu yang dibutuhkan untuk menemukan *path* maksimum reward.
10) *User* kemudian diberikan opsi untuk menyimpan solusi tersebut ke dalam *file*, termasuk *path*, total *reward*, koordinat token, dan waktu eksekusi.

BAB II
SOURCE CODE & IMPLEMENTASI

2.1 GitHub Repository

Source code dari program ini dapat diakses melalui GitHub https://github.com/caernations/Tucil_13522140

2.2 Library

Penggunaan berbagai *library* memiliki peran penting dalam memfasilitasi berbagai fungsi dan fitur yang diimplementasikan dalam program ini. Melalui penggunaan *library* seperti **`iostream`**, **`vector`**, **`string`**, dan *library* lain yang tertulis pada gambar di bawah ini, program dapat melakukan berbagai hal seperti manipulasi data, interaksi dengan *user* melalui input dan output, serta pemrosesan string. Demikian pula, *library-library* tersebut memungkinkan pengembangan program secara lebih efisien dengan menyediakan alat dan fungsionalitas yang telah dirancang sebelumnya.

```
#include <iostream> // I/O operations
#include <vector>   // dynamic arrays to store matrix &
                       paths
#include <string>   // string class
#include <sstream>  // sequence generator
#include <map>      // mapping sequence to its reward
#include <fstream>  // file I/O
#include <random>   // randomized inputs
#include <chrono>   // duration
```

```
#include <cstdlib>  // system()
#include <set>      // for storing unique sequences
using namespace std;
```

2.3 Global Variable

Penggunaan variabel global memiliki tujuan khusus dalam menyimpan informasi yang dibutuhkan secara luas di berbagai bagian dalam program ini. Variabel global digunakan untuk menyimpan nilai-nilai seperti ukuran matriks, ukuran *buffer*, jumlah sekuens, dan hasil maksimal yang diperoleh, dan kemudian dapat diakses dan dimanipulasi oleh fungsi-fungsi utama program. Berikut merupakan variabel global yang digunakan dalam program ini untuk mendukung fungsi-fungsi program secara keseluruhan:

```
int n, m, u, numOfSequences;       // n = rows, m = buffer
                                      size, u = column
int maxReward = INT_MIN;           // global variable to
                                      track the max reward
vector<int> maxRewardPath;         // store the path of
                                       maximum reward
vector<vector<int>> allPaths;      // store all the
                                      possible paths
vector<vector<string>> matrix;     // matrix to store
                                      strings
map<string, int> sequenceRewards; // mapping sequence to
                                      its reward
```

2.4 Main Program Functions

Fungsi-fungsi utama dalam program ini bertanggung jawab untuk mengatur alur utama program, mulai dari menerima input pengguna hingga menangani proses pencarian *path* yang paling optimum. Fungsi-fungsi utama

ini merupakan inti dari program yang langsung terlibat dalam menjalankan algoritma pencarian jalur optimal yang diimplementasikan.

manualInput()

Fungsi **manualInput()** berfungsi untuk meminta input dari pengguna secara manual. Pertama, pengguna diminta untuk memasukkan ukuran *buffer* yang akan digunakan dalam proses pencarian *path*. Selanjutnya, pengguna diminta untuk memasukkan dimensi matriks, yang terdiri dari lebar dan tinggi matriks. Nilai-nilai ini kemudian disesuaikan dengan variabel global **u** dan **n**. Setelah itu, pengguna diminta untuk memasukkan elemen-elemen matriks, satu per satu. Matriks yang telah dimasukkan akan disimpan dalam vektor **matrix**. Selanjutnya, pengguna diminta untuk memasukkan jumlah urutan (*sequences*) yang akan digunakan dalam pencarian jalur. Setiap urutan terdiri dari sebuah string dan nilai reward yang terkait. Pengguna diminta untuk memasukkan setiap urutan secara berurutan, dengan memastikan bahwa setiap urutan yang dimasukkan adalah unik dan sesuai dengan aturan yang ditetapkan. **Proses validasi** dilakukan untuk memastikan bahwa setiap token dalam urutan terdiri dari dua karakter alfanumerik yang dituliskan dalam huruf besar. Jika input tidak valid, pengguna diminta untuk memasukkan urutan kembali. Setelah semua urutan dimasukkan dengan benar, informasi urutan dan nilai *reward*-nya disimpan dalam peta **sequenceRewards**.

```
void manualInput()
{
    cout << "\n   === MANUAL INPUT ===\n\n"
         << ">> Buffer size\t\t\t: ";
    cin >> m;

    int width, height;
    cout << ">> Matrix dimension (w h)\t: ";
    cin >> width >> height;
```

```
    // adjusting to the global variables
    u = width;
    n = height;

    matrix.resize(n, vector<string>(u));
    vector<vector<bool>> visited(n, vector<bool>(u, false));
    cout << ">> Matrix elements\t\t:\n";
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < u; ++j)
        {
            cin >> matrix[i][j];
        }
    }

    cout << "\n>> Number of sequences\t\t: ";
    cin >> numOfSequences;
    cin.ignore();

    set<string> uniqueSequences; // store unique sequences
using set
    for (int i = 1; i <= numOfSequences; ++i)
    {
        string sequence;
        int reward;
        bool validInput = false;

        while (!validInput)
        {
            cout << ">> Sequence " << i << "\t\t\t: ";
            getline(cin, sequence);
            // make sure the sequence is unique
            if (uniqueSequences.find(sequence) !=
uniqueSequences.end())
            {
                cout << "!! This sequence already exist.\n!!
Please enter a different sequence." << endl;
                system("pause");
                continue;
            }

            istringstream iss(sequence);
            string token;
            validInput = true;

            // validate tokens (alphanumeric & capital
letters)
            while (iss >> token)
            {
                if (token.size() != 2 || !isalnum(token[0]) ||
!isalnum(token[1]) ||
```

```
                    (isalpha(token[0]) && !isupper(token[0]))
|| (isalpha(token[1]) && !isupper(token[1])))
                {
                    validInput = false;
                    cout << "!! Each token must consist of
exactly 2 alphanumeric & in uppercase.\n!! Please re-enter
sequence " << i << "! " << endl;
                    break;
                }
            }
        }

        if (validInput)
        {
            cout << ">> Sequence " << i << " reward\t\t: ";
            cin >> reward;
            cin.ignore();
            sequenceRewards[sequence] = reward;
            uniqueSequences.insert(sequence);
        }
    }
}
```

`randomizedInput()`

Fungsi **`randomizedInput`** berfungsi untuk menghasilkan data masukan secara acak untuk program. Fungsi ini pertama-tama meminta pengguna untuk memasukkan jumlah token unik dan melakukan validasi untuk memastikan jumlah tersebut positif (untuk menghindari *error*). Selanjutnya, pengguna diminta untuk memasukkan token-token tersebut yang harus terdiri dari dua karakter alfanumerik dalam huruf besar. Fungsi ini melakukan **validasi** dengan mengulang permintaan masukan token sampai pengguna memasukkan token yang valid. Setelah itu, pengguna diminta untuk memasukkan ukuran *buffer*, dimensi matriks (lebar dan tinggi), jumlah *sequences*, dan panjang maksimum token dalam setiap *sequence*. Fungsi ini juga memastikan bahwa panjang maksimum urutan tidak kurang dari dua. Menggunakan *library random generator*, fungsi ini mengisi matriks dengan token yang dipilih secara acak dari daftar token. Selanjutnya, fungsi ini juga

menghasilkan urutan token yang unik beserta nilai *reward* yang telah ditentukan, memastikan bahwa tidak ada urutan yang berulang. Hasil dari input acak ini kemudian ditampilkan ke pengguna, termasuk ukuran *buffer*, dimensi matriks, dan daftar urutan beserta *reward*-nya. Akhirnya, pengguna diberi pilihan untuk melanjutkan dengan masukan ini atau tidak. Jika pengguna memilih untuk tidak melanjutkan ('n' atau 'N'), program akan dihentikan; jika tidak, program akan melanjutkan eksekusi dengan data yang telah dihasilkan tadi.

```
void randomizedInput()
{
    int jumlah_token_unik, ukuran_buffer, ukuran_matriks_x,
ukuran_matriks_y, jumlah_sekuens, ukuran_maksimal_sekuens;

    cout << "\n   === RANDOMIZED INPUT ===\n\n"
         << ">> Number of unique tokens\t: ";
    cin >> jumlah_token_unik;

    if (jumlah_token_unik <= 0) // error handling
    {
        cerr << "!! Number of unique tokens must be positive."
<< endl;
        return;
    }

    vector<string> tokens;
    tokens.reserve(jumlah_token_unik);
    cout << ">> Tokens (must be unique)\t: ";
    // validate tokens (alphanumeric & capital letters)
    for (int i = 0; i < jumlah_token_unik; ++i)
    {
        string token;
        while (true)
        {
            cin >> token;
            if (token.size() != 2 || !isalnum(token[0]) ||
!isalnum(token[1]) ||
                (isalpha(token[0]) && !isupper(token[0])) ||
(isalpha(token[1]) && !isupper(token[1])))
            {
                cout << "!! Invalid token.\n>> Please re-enter
token\t: ";
                continue;
            }
```

```
            tokens.push_back(token);
            break;
        }
    }

    cout << ">> Buffer size\t\t\t: ";
    cin >> ukuran_buffer;
    cout << ">> Matrix dimension (w h)\t: ";
    cin >> ukuran_matriks_x >> ukuran_matriks_y;
    cout << ">> Number of sequences\t\t: ";
    cin >> jumlah_sekuens;
    cout << ">> Maximum tokens in sequences\t: ";
    cin >> ukuran_maksimal_sekuens;
    if (ukuran_maksimal_sekuens < 2)
        ukuran_maksimal_sekuens = 2;

    random_device rd;
    mt19937 gen(rd());

    n = ukuran_matriks_y;
    u = ukuran_matriks_x;
    m = ukuran_buffer;
    matrix.resize(n, vector<string>(u));
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < u; ++j)
        {
            matrix[i][j] = tokens[gen() % jumlah_token_unik];
        }
    }

    set<string> uniqueSequences;
    sequenceRewards.clear();
    for (int i = 0; i < jumlah_sekuens; ++i)
    {
        stringstream sequence;
        string seqStr;
        bool uniqueSequenceFound = false;

        while (!uniqueSequenceFound)
        {
            sequence.str("");
            int sequenceLength = gen() %
(ukuran_maksimal_sekuens - 1) + 2; // make sure that a
sequence at least contains 2 tokens

            for (int j = 0; j < sequenceLength; ++j)
            {
                if (j > 0)
                    sequence << " ";
                sequence << tokens[gen() % jumlah_token_unik];
```

```
            }
            seqStr = sequence.str();

            if (uniqueSequences.find(seqStr) ==
uniqueSequences.end())
            {
                uniqueSequenceFound = true;
                uniqueSequences.insert(seqStr);
                int reward = gen() % 100 + 1;
                sequenceRewards[seqStr] = reward;
            }
        }
    }

    cout << "\n\n   === RESULT: ===\n\n";
    cout << ukuran_buffer << "\n"
         << ukuran_matriks_x << " " << ukuran_matriks_y <<
"\n";
    for (const auto &row : matrix)
    {
        for (const auto &elem : row)
        {
            cout << elem << " ";
        }
        cout << "\n";
    }
    cout << sequenceRewards.size() << "\n";
    for (const auto &seq : sequenceRewards)
    {
        cout << seq.first << "\n"
             << seq.second << "\n";
    }

    char proceed;
    cout << "\n   === CONFIRM INPUT ===\n"
         << ">> Proceed with this input? (y/n)"
         << ">> ";
    cin >> proceed;
    system("pause");

    if (proceed == 'n' || proceed == 'N')
    {
        exit(0);
    }
}
```

fileInput()

Fungsi **fileInput** dalam program ini berfungsi untuk membaca data masukan dari sebuah file teks. Prosesnya dimulai dengan meminta pengguna untuk memasukkan nama file (tanpa ekstensi) yang berada di direktori "../inputs/". Setelah pengguna memasukkan nama *file*, program mencoba membuka *file* tersebut. Jika *file* tidak ditemukan, program akan memberikan pesan kesalahan dan meminta pengguna untuk memasukkan nama *file* kembali. Ini dilakukan dalam loop tak terbatas sampai *file* yang valid ditemukan. Setelah *file* berhasil dibuka, program mulai membaca data dari *file* tersebut. Pertama, ia membaca ukuran *buffer* (**m**), diikuti oleh dimensi matriks (**n** untuk baris dan **u** untuk kolom). Kemudian, matriks diinisialisasi sesuai dengan dimensi tersebut, dan elemen-elemen matriks dibaca satu per satu. Selanjutnya, program membaca jumlah urutan atau *sequence* (**numOfSequences**) dan memulai *loop* untuk membaca setiap *sequence* beserta nilai *reward* yang terkait. Dalam *loop* ini, untuk setiap *sequence*, program membaca *sequence* sebagai string dan *reward* sebagai integer. Setiap pasangan *sequence* dan reward ini disimpan dalam **sequenceRewards**, yang merupakan *map* yang menghubungkan *sequence* dengan *reward*-nya. Akhirnya, setelah semua data berhasil dibaca, *file* input ditutup. Fungsi ini memungkinkan program untuk memproses data dari *file* teks, memberikan fleksibilitas dan kemudahan dalam memasukkan data, terutama untuk jumlah data yang besar atau untuk pengujian berulang dengan set data yang sama. Ini sangat berguna dalam lingkungan pengembangan dan pengujian, di mana dapat menghemat waktu dan mengurangi kesalahan dalam memasukkan data secara manual.

```
void fileInput()
{
    ifstream inputFile;
    string filename;
```

```
    string filePath;

    while (true)
    {
        cout << "\n   === FILE INPUT ===\n\n"
             << ">> .txt filename (without extension)\t: ";
        cin >> filename;
        filePath = "../inputs/" + filename + ".txt"; // make
sure that it's a .txt file

        inputFile.open(filePath);
        if (inputFile.is_open())
        {
            break;
        }
        else
        {
            cerr << "File not found: " << filePath << ".
Please try again." << endl;
            system("pause");
            system("cls");
        }
    }

    inputFile >> m;      // buffer size
    inputFile >> n >> u; // matrix dimension
    matrix.resize(n, vector<string>(u));

    // matrix elements
    for (int i = 0; i < n; ++i)
    {
        for (int j = 0; j < u; ++j)
        {
            inputFile >> matrix[i][j];
        }
    }

    // number of sequences
    inputFile >> numOfSequences;
    inputFile.ignore();

    // sequences and & rewards
    for (int i = 0; i < numOfSequences; ++i)
    {
        string sequence;
        int reward;
        getline(inputFile, sequence); // sequence
        inputFile >> reward;          // reward
        inputFile.ignore();
        sequenceRewards[sequence] = reward;
    }
```

```
    inputFile.close();
}
```

calculateReward(const vector<int> &path)

Fungsi **calculateReward** berfungsi menghitung total *reward* sebuah *path* yang ditemukan. *Path* tersebut direpresentasikan sebagai vektor integer, di mana setiap elemen menunjukkan indeks pada matriks permainan. Fungsi ini mengonversi jalur tersebut menjadi *string* yang merepresentasikan urutan token pada jalur itu. Proses ini dilakukan dengan menggabungkan elemen-elemen matriks berdasarkan indeks yang diberikan dalam jalur. Setelah mendapatkan representasi string dari jalur tersebut, fungsi ini kemudian mencari *sequence* yang telah didefinisikan sebelumnya dan terdapat dalam jalur. Jika sebuah urutan ditemukan, *reward* yang terkait dengan urutan tersebut ditambahkan ke total *reward*. *Reward* total ini mengindikasikan nilai dari jalur tersebut berdasarkan seberapa banyak dan nilai *reward* dari urutan token yang ditemukan.

```
int calculateReward(const vector<int> &path)
{
    string pathStr;
    for (int num : path)
    {
        if (!pathStr.empty())
            pathStr += " ";
        pathStr += matrix[num / u][num % u];
    }

    int totalReward = 0;
    for (const auto &seqReward : sequenceRewards)
    {
        if (pathStr.find(seqReward.first) != string::npos)
        {
            totalReward += seqReward.second;
        }
    }
    return totalReward;
}
```

storePath(const vector<int> &path)

Fungsi **storePath** berfungsi untuk menyimpan setiap *path* yang ditemukan selama proses eksekusi fungsi **findPaths**. *Path* yang disimpan adalah representasi dari urutan token dalam matriks. Fungsi ini menerima sebuah vektor integer yang mewakili *path* dan menambahkannya ke dalam vektor **allPaths** yang merupakan kumpulan dari semua jalur yang telah ditemukan. Setiap jalur disimpan untuk tujuan analisis lebih lanjut atau untuk penggunaan dalam fungsi lain, seperti dalam proses pencarian jalur dengan reward tertinggi atau untuk keperluan debugging dan verifikasi. Dengan menyimpan setiap jalur yang ditemukan, program memungkinkan pengguna untuk melihat semua kemungkinan solusi yang ada dan bagaimana setiap solusi itu terbentuk.

```
void storePath(const vector<int> &path)
{
    allPaths.push_back(path);
}
```

printPath(int x, int y, vector<int> &path, vector<vector<bool>> &visited, bool horizontal)

Fungsi **printPath** dalam program ini dirancang untuk menampilkan informasi detail tentang jalur yang diambil dalam matriks. Fungsi ini menerima dua parameter: vektor **path** yang berisi *path* dalam bentuk indeks matriks dan *boolean* **printReward** yang menentukan apakah *reward* total harus dicetak atau tidak. Ketika dipanggil, fungsi ini pertama-tama menghitung *reward* total *path* dengan menggunakan fungsi **calculateReward**, yang menjadikan *path* sebagai argumen fungsi. *Reward* ini dihitung berdasarkan seberapa banyak urutan token yang telah ditentukan sebelumnya ditemukan dalam *path* tersebut dan nilai *reward* yang terkait dengan urutan-urutan tersebut. Jika **printReward** bernilai *true*,

fungsi ini akan mencetak reward total ke layar. Setelah itu, fungsi ini melanjutkan dengan mengiterasi melalui setiap elemen dalam *path*. Untuk setiap elemen, fungsi mengkonversi indeks matriks menjadi koordinat x dan y yang sesuai, dan kemudian mencetak elemen matriks di posisi tersebut. Hal ini memberikan visualisasi dari jalur yang diambil dalam bentuk token yang diakses. Selanjutnya, fungsi ini mencetak koordinat setiap elemen dalam jalur. Koordinat ini dihitung dengan membagi indeks dengan lebar matriks untuk mendapatkan nilai x (baris) dan menggunakan modulo lebar matriks untuk mendapatkan nilai y (kolom). Setiap pasangan koordinat ini menunjukkan lokasi spesifik dari setiap token dalam matriks.

```
void printPath(const vector<int> &path, bool printReward =
true)
{
    int reward = calculateReward(path);
    if (printReward)
    {
        cout << endl;
        cout << endl;
        cout << "   === RESULT: ===\n\n";
        cout << reward; // print total reward
    }
    cout << endl;
    for (int num : path)
    {
        int x = num / u;
        int y = num % u;
        cout << matrix[x][y] << " ";
    }

    // print tokens coordinates
    cout << endl;
    for (int num : path)
    {
        int x = num / u;
        int y = num % u;
        cout << y + 1 << ", " << x + 1 << endl;
    }
}
```

isValid(int x, int y, vector<int> &path, vector<vector<bool>> &visited, bool horizontal)

Fungsi **isValid** memiliki peran kunci dalam memvalidasi koordinat selama proses pencarian jalur di dalam matriks. Fungsi ini menerima tiga parameter: koordinat x dan y yang merepresentasikan posisi dalam matriks, serta sebuah matriks **visited** yang mencatat sel-sel matriks yang telah dikunjungi. Fungsi ini mengembalikan nilai *true* jika posisi (x, y) berada dalam *range* matriks dan belum pernah dikunjungi sebelumnya. Hal ini memastikan bahwa pencarian *paths* tidak keluar dari batas matriks dan tidak mengunjungi sel matriks yang sama lebih dari satu kali sehingga meningkatkan efisiensi dan menghindari *infinite loop*.

```
bool isValid(int x, int y, vector<vector<bool>> &visited)
{
    return x >= 0 && x < n && y >= 0 && y < u &&
!visited[x][y];
}
```

findPaths(int x, int y, vector<int> &path, vector<vector<bool>> &visited, bool horizontal)

Fungsi **findPaths** adalah inti dari algoritma pencarian *path* secara *brute force* dalam program ini. Fungsi ini menggunakan pendekatan rekursif untuk mengeksplorasi semua kemungkinan jalur dalam matriks berdasarkan ukuran *buffer* yang ditentukan. Fungsi ini menerima koordinat awal (x, y), jalur yang sedang dibangun, matriks **visited**, dan *boolean* horizontal yang menentukan arah gerak selanjutnya. Setiap kali fungsi dipanggil, ia menandai posisi saat ini sebagai dikunjungi dan menambahkannya ke *paths*. Jika ukuran jalur sama dengan *buffer size* (**m**), maka *path* tersebut dicek untuk *reward*nya, dan kemudian disimpan jika memiliki reward yang lebih tinggi dari yang sudah ada. Selanjutnya, fungsi ini melakukan pemanggilan rekursif untuk mengeksplorasi selanjutnya, baik secara vertikal maupun horizontal,

tergantung pada nilai horizontal. Dengan cara ini, semua kombinasi *path* yang mungkin dihasilkan dan dievaluasi. Kombinasi ini lah yang merupakan implementasi dari algoritma *brute force*.

```
void findPaths(int x, int y, vector<int> &path,
vector<vector<bool>> &visited, bool horizontal)
{
    visited[x][y] = true;
    path.push_back(x * u + y);

    if (path.size() == m)
    {
        int currentReward = calculateReward(path);
        if (currentReward > maxReward ||
maxRewardPath.empty())
        {
            maxReward = currentReward;
            maxRewardPath = path;
        }
        storePath(path);
    }
    else
    {
        if (!horizontal)
        {
            // move vertically down
            for (int i = x + 1; i < n; ++i)
            {
                if (isValid(i, y, visited))
                {
                    findPaths(i, y, path, visited,
!horizontal);
                }
            }
            // move vertically up
            for (int i = x - 1; i >= 0; --i)
            {
                if (isValid(i, y, visited))
                {
                    findPaths(i, y, path, visited,
!horizontal);
                }
            }
        }
        else
        {
            // move horizontally right & left
            for (int j = 0; j < u; ++j)
            {
```

```
                if (j != y && isValid(x, j, visited))
                {
                    findPaths(x, j, path, visited,
!horizontal);
                }
            }
        }
    }

    visited[x][y] = false;
    path.pop_back();
}
```

saveOptimumPath(const vector<int> &path, const long long executionTime)

Fungsi **saveOptimumPath** bertugas menyimpan jalur dengan reward tertinggi ke dalam *file* teks. Pengguna diminta untuk memasukkan nama *file* (tanpa ekstensi) tempat hasil akan disimpan. Fungsi ini kemudian menciptakan file output dan menulis reward dari jalur optimal, elemen-elemen dari jalur dalam bentuk token, koordinat masing-masing token, dan waktu eksekusi pencarian jalur dalam milisekon.

```
bool isValid(int x, int y, vector<vector<bool>> &visited)
{
    return x >= 0 && x < n && y >= 0 && y < u &&
!visited[x][y];
}
```

2.5 Debugging Functions

Pengembangan sebuah website secara umum merupakan proses yang kompleks dan detail, dimulai dengan fase perencanaan yang melibatkan pemahaman mendalam tentang kebutuhan dan tujuan.

printAllPaths()

Fungsi **printAllPaths** dalam program ini bertujuan untuk menampilkan semua *paths* yang mungkin, berdasarkan input yang diberikan. Fungsi ini tidak menerima parameter apa pun dan langsung mengakses

variabel global **allPaths** yang berisi daftar semua jalur yang telah dieksplorasi. Untuk setiap *path* dalam **allPaths**, fungsi ini melakukan iterasi melalui setiap elemen *path* (yang merupakan indeks dalam matriks) dan mengkonversinya menjadi token yang sesuai dari matriks. Setelah itu, fungsi ini mencetak token-token ini secara berurutan, memberikan visualisasi langsung dari jalur tersebut dalam matriks. Selain itu, fungsi ini juga menghitung dan mencetak reward untuk setiap jalur dengan menggunakan fungsi **calculateReward**.

```
void printAllPaths()
{
    cout << "All possible paths:\n";
    for (const auto &path : allPaths)
    {
        for (int num : path)
        {
            int x = num / u;
            int y = num % u;
            cout << matrix[x][y] << " ";
        }
        int reward = calculateReward(path);
        cout << "= " << reward << endl;
    }
}
```

saveAllPathsToFile()

Fungsi **saveAllPathsToFile** berfungsi untuk menyimpan semua *path* yang mungkin beserta rewardnya ke dalam sebuah *file* berekstensi .txt. Fungsi ini pertama kali meminta pengguna untuk memasukkan nama file (tanpa ekstensi), kemudian menciptakan atau membuka *file* dengan nama tersebut untuk operasi penulisan. Dalam fungsi ini, proses iterasi dilakukan pada setiap jalur yang tersimpan dalam **allPaths**. Untuk setiap *path*, fungsi ini menuliskan token-token yang terdapat dalam jalur tersebut ke dalam *file*, diikuti dengan *reward* yang dihitung untuk jalur tersebut. Hal ini memberikan dokumentasi lengkap dari semua *paths* yang telah dieksplorasi

selama eksekusi program, yang berguna untuk analisis lebih lanjut atau keperluan *debugging*.

```
void saveAllPathsToFile()
{
    string fileName;
    cout << ">> Filename (without extension): ";
    cin >> fileName;
    fileName = "../test/" + fileName + ".txt";
    ofstream file(fileName);
    if (!file.is_open())
    {
        cout << "Failed to open file for writing." << endl;
        return;
    }

    for (const auto &path : allPaths)
    {
        for (int num : path)
        {
            file << matrix[num / u][num % u] << " ";
        }
        int reward = calculateReward(path);
        file << "= " << reward << endl;
    }

    file.close();
    cout << "All paths saved to " << fileName << endl;
}
```

Dalam fase ini, tim pengembang menentukan fungsi-fungsi utama yang harus diintegrasikan, dan merencanakan arsitektur informasi yang akan menuntun pengalaman pengguna. Setelah perencanaan, tahap desain dijalankan dengan menciptakan prototipe visual yang merefleksikan identitas merek dan memastikan usability yang baik. Desainer UX/UI berkolaborasi erat untuk menghasilkan desain antarmuka yang intuitif dan menarik. Dengan desain sebagai panduan, pengembangan *website* dibagi menjadi dua area utama: pengembangan *frontend* dan *backend*.

2.6 Main Function

Fungsi **`main()`** dalam program ini merupakan titik awal dan inti dari eksekusi program. Fungsi ini memulai dengan menyajikan menu interaktif kepada pengguna untuk memilih jenis input: manual, acak, atau dari file. Berdasarkan pilihan pengguna, fungsi yang sesuai (**`manualInput`**, **`randomizedInput`**, atau **`fileInput`**) dipanggil untuk menerima input yang diperlukan, seperti ukuran *buffer*, dimensi matriks, elemen matriks, dan urutan token berserta rewardnya. Setelah input selesai, fungsi ini menggunakan **`chrono`** untuk memulai penghitungan waktu, yang penting untuk mengukur durasi eksekusi pencarian *paths*. Kemudian, algoritma pencarian jalur dijalankan dengan memanggil fungsi **`findPaths`**. Fungsi ini mencari semua *path* yang mungkin dalam matriks berdasarkan input yang diberikan, mengevaluasi dan menyimpan *path* dengan reward maksimum. Setelah proses pencarian jalur selesai, **`main()`** memanggil **`printPath`** untuk menampilkan *path* dengan *reward* terbesar beserta koordinatnya. Kemudian, fungsi ini menghentikan penghitungan waktu dan mencetak durasi eksekusi. Selanjutnya, program memberikan opsi kepada pengguna untuk menyimpan solusi ke dalam *file*. Jika pengguna memilih untuk menyimpan, fungsi **`saveOptimumPath`** dipanggil, yang menyimpan *path* optimal beserta detailnya ke dalam *file* yang namanya ditentukan oleh *user*.

```
int main()
{
    using namespace std::chrono;

    int inputType;
    bool validInput = false;

    while (!validInput)
    {
        cout << "\n   === CYBERPUNK 2077 BREACH PROTOCOL===\n"
             << "_____________________________________________\n"
             << "|                                           |\n"
             << "| Tugas Kecil 1 IF2211 Strategi Algoritma  |\n"
             << "| 2024 | 13522140, Yasmin Farisah Salma    |\n"
```

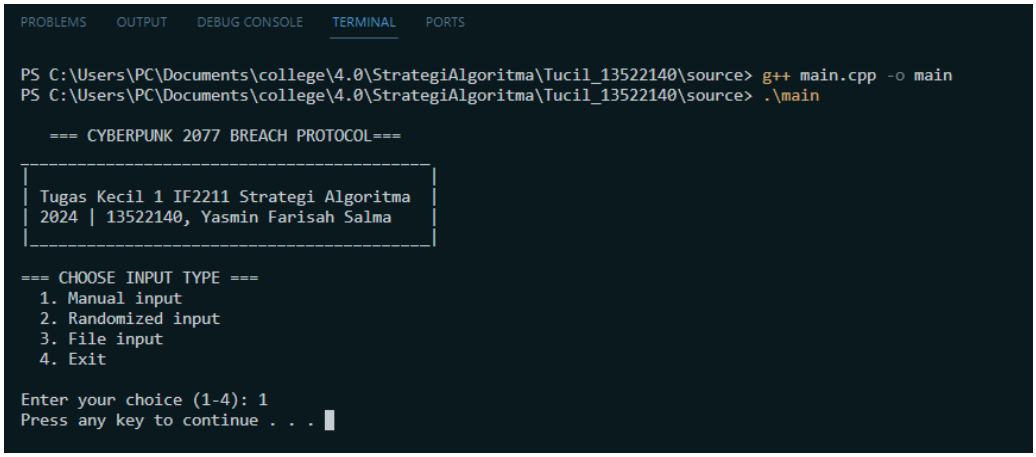
```
             << "|_____________________________________________|\n\n"
             << "=== CHOOSE INPUT TYPE ===\n"
             << "  1. Manual input\n"
             << "  2. Randomized input\n"
             << "  3. File input\n"
             << "  4. Exit\n\n"
             << "Enter your choice (1-4): ";
        cin >> inputType;
        system("pause");
        system("cls");

        if (cin.fail())
        {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "\n>> Invalid input.\n>>Please enter a number
between 1 and 4." << endl;
            continue;
        }

        cin.ignore();

        switch (inputType)
        {
        case 1:
            manualInput();
            validInput = true;
            break;
        case 2:
            randomizedInput();
            validInput = true;
            break;
        case 3:
            fileInput();
            validInput = true;
            break;
        case 4:
            cout << ">> Exiting program." << endl;
            return 0;
        default:
            cout << "\n>> Invalid choice.\n>>Please enter a number
between 1 and 4." << endl;
        }
    }

    // start timing after all inputs are done
    auto start = high_resolution_clock::now();

    vector<vector<bool>> visited(n, vector<bool>(u, false));
    for (int i = 0; i < u; ++i)
    {
        vector<int> path;
        findPaths(0, i, path, visited, false);
    }

    // printAllPaths();
```

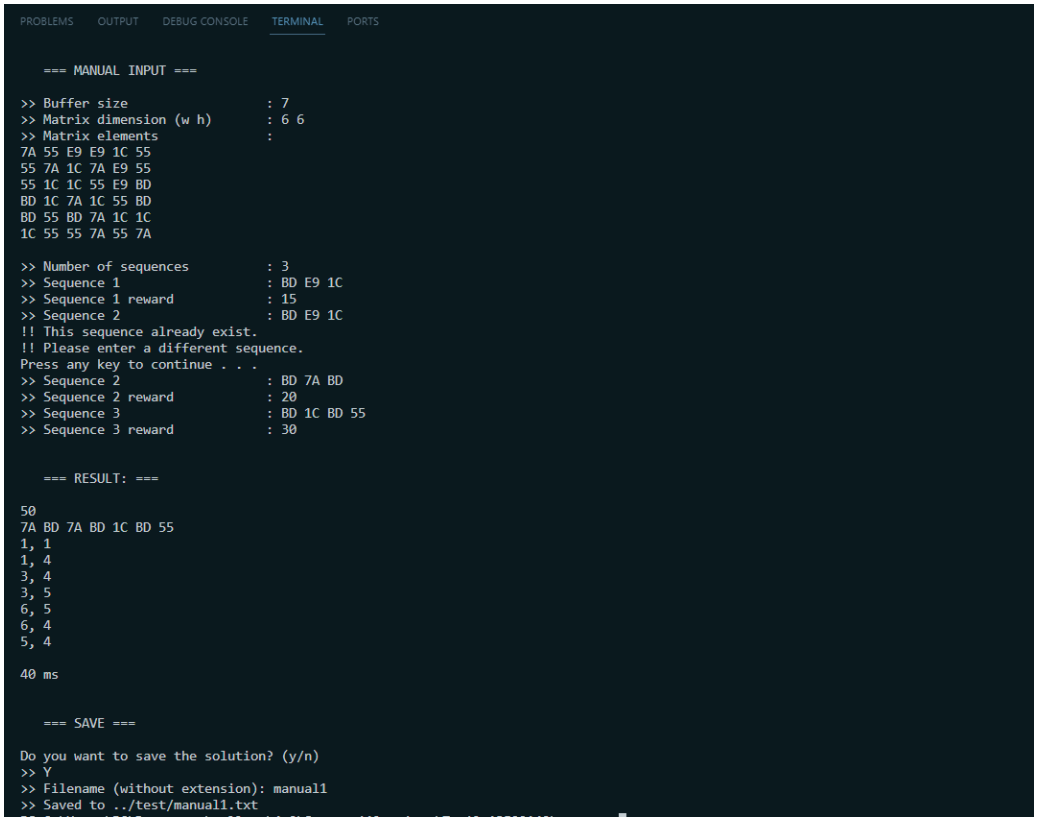
```
    // print and save the optimum path
    if (!maxRewardPath.empty())
    {
        printPath(maxRewardPath); // prints the path, reward, and
coordinates
    }
    else
    {
        cout << ">> No paths found.\n";
    }

    // stop timing after printing the coordinates
    auto stop = high_resolution_clock::now();
    auto duration = duration_cast<milliseconds>(stop - start);
    cout << endl;
    cout << duration.count() << " ms\n";

    char saveSolution;
    cout << endl;
    cout << endl;
    cout << "   === SAVE ===\n\n"
         << "Do you want to save the solution? (y/n)\n"
         //  << "Do you want to save all the paths? (y/n)\n"
         << ">> ";
    cin >> saveSolution;
    if (saveSolution == 'y' || saveSolution == 'Y')
    {
        saveOptimumPath(maxRewardPath, duration.count());
        // saveAllPathsToFile();
    }
    else
    {
        cout << ">> Solution's not saved." << endl;
    }

    return 0;
}
```

BAB III

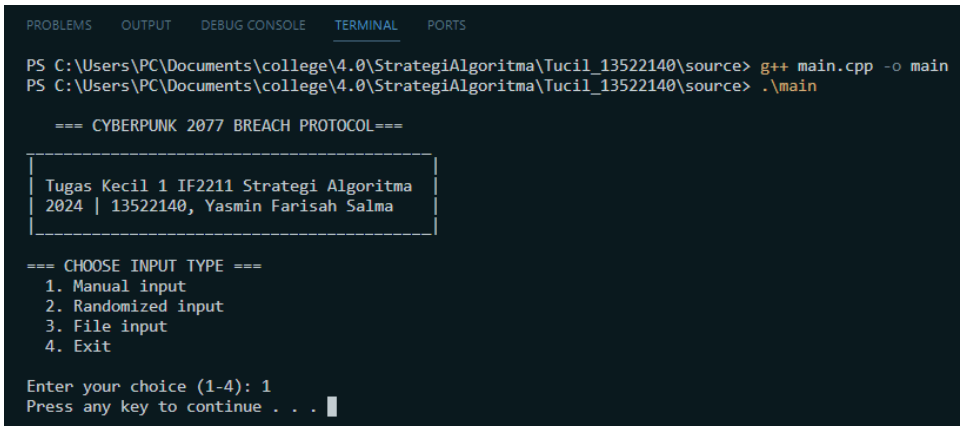
HASIL EKSEKUSI PROGRAM

3.1 Hasil Manual Input 1

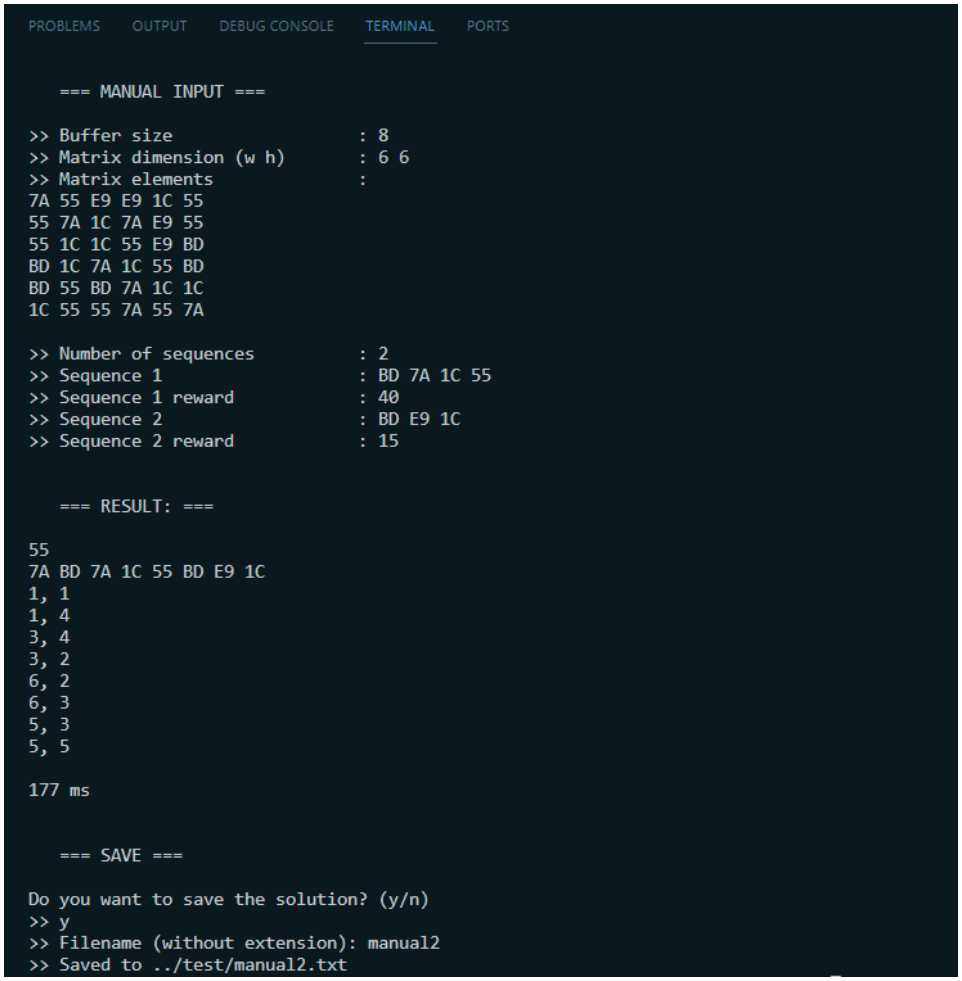




3.2 Hasil Manual Input 2

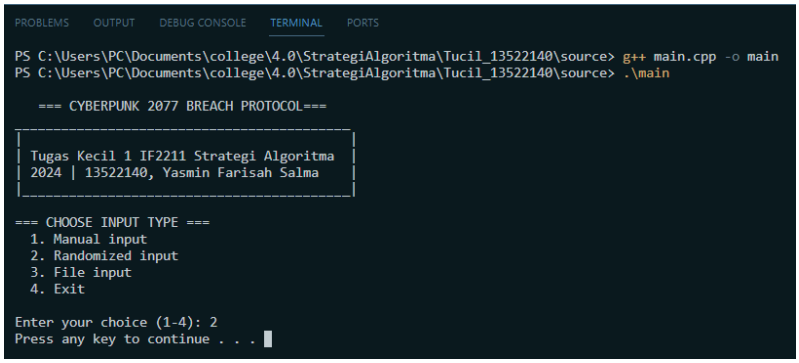


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> g++ main.cpp -o main
PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> .\main

   === CYBERPUNK 2077 BREACH PROTOCOL===
 _____________________________________
|                                     |
| Tugas Kecil 1 IF2211 Strategi Algoritma  |
| 2024 | 13522140, Yasmin Farisah Salma    |
|_____________________________________|

=== CHOOSE INPUT TYPE ===
  1. Manual input
  2. Randomized input
  3. File input
  4. Exit

Enter your choice (1-4): 1
Press any key to continue . . .
```

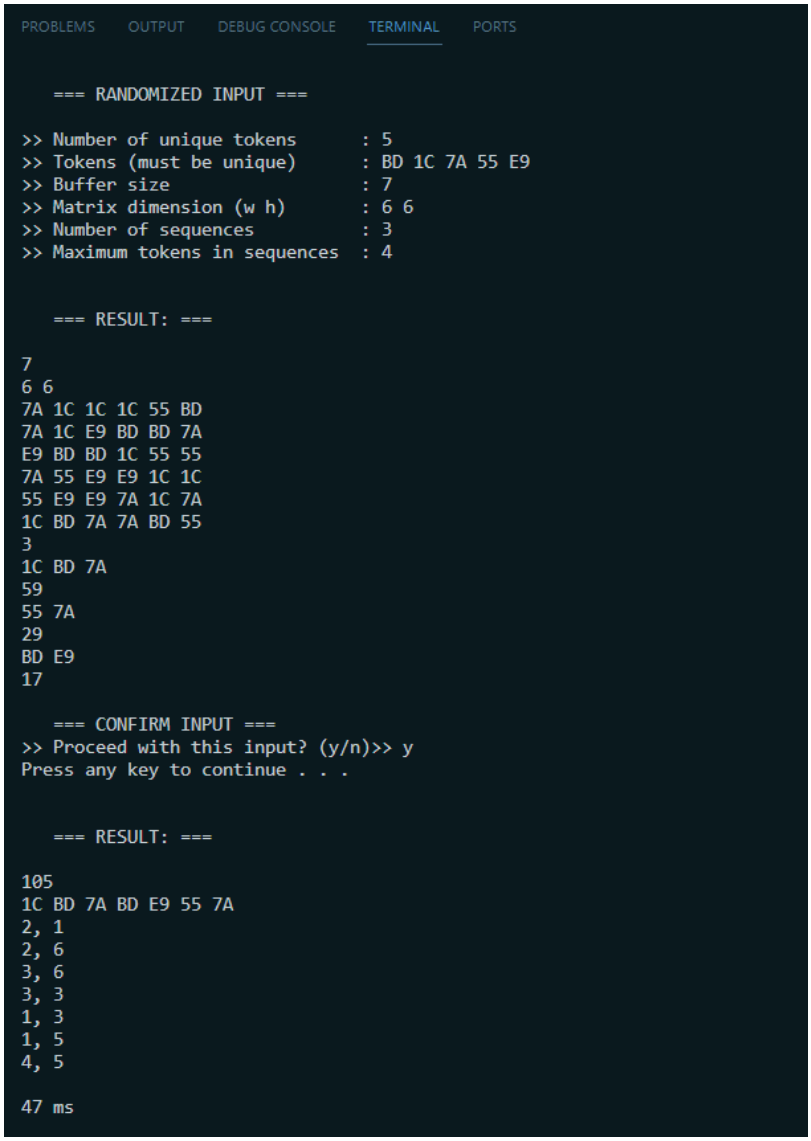


```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS


   === MANUAL INPUT ===

>> Buffer size                : 8
>> Matrix dimension (w h)     : 6 6
>> Matrix elements            :
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

>> Number of sequences        : 2
>> Sequence 1                 : BD 7A 1C 55
>> Sequence 1 reward          : 40
>> Sequence 2                 : BD E9 1C
>> Sequence 2 reward          : 15


   === RESULT: ===

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

177 ms


   === SAVE ===

Do you want to save the solution? (y/n)
>> y
>> Filename (without extension): manual2
>> Saved to ../test/manual2.txt
```

3.3 Hasil Random Input 1



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> g++ main.cpp -o main
PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> .\main

   === CYBERPUNK 2077 BREACH PROTOCOL===
 _____________________________________
|                                     |
| Tugas Kecil 1 IF2211 Strategi Algoritma  |
| 2024 | 13522140, Yasmin Farisah Salma    |
|_____________________________________|

=== CHOOSE INPUT TYPE ===
  1. Manual input
  2. Randomized input
  3. File input
  4. Exit

Enter your choice (1-4): 2
Press any key to continue . . .
```

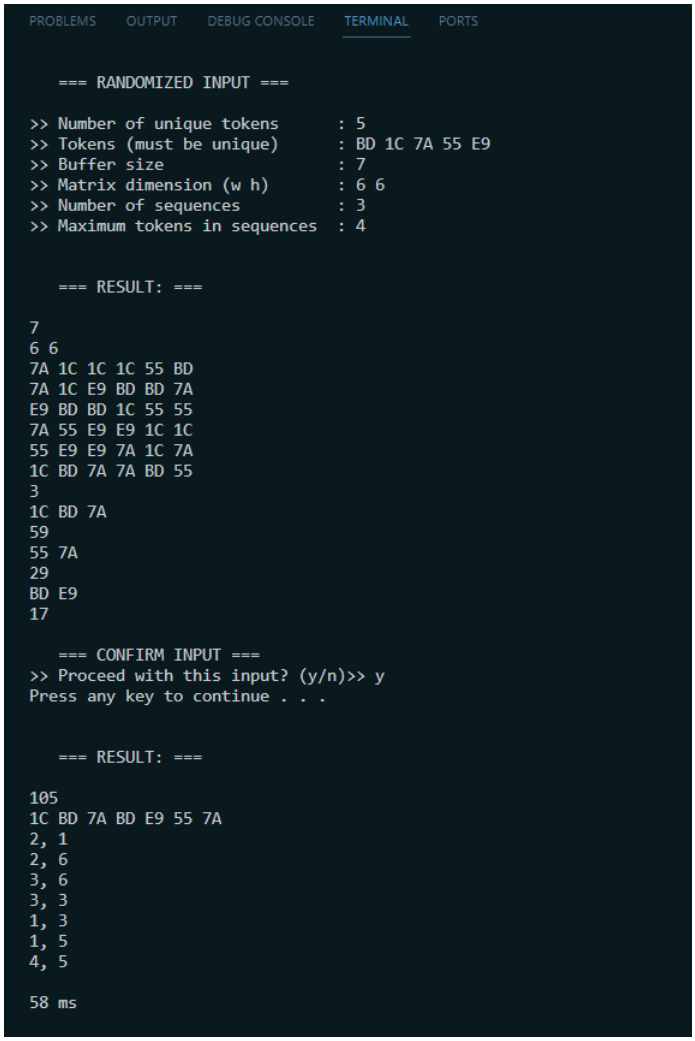


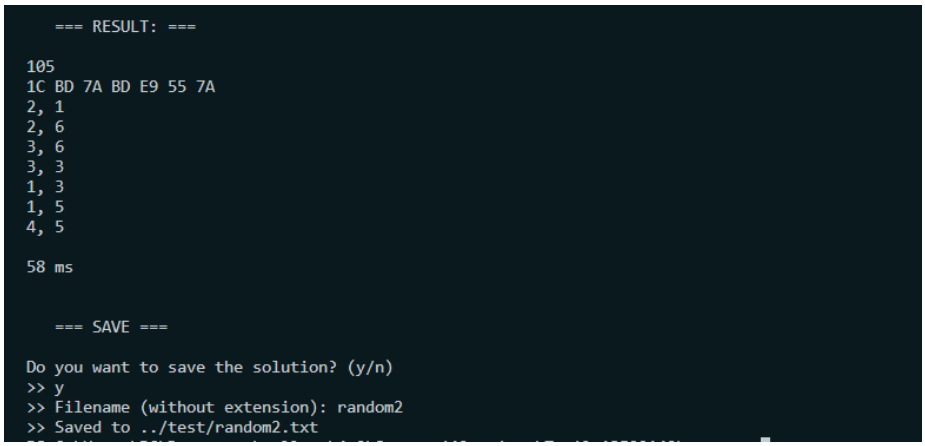
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS


   === RANDOMIZED INPUT ===

>> Number of unique tokens      : 5
>> Tokens (must be unique)      : BD 1C 7A 55 E9
>> Buffer size                  : 7
>> Matrix dimension (w h)       : 6 6
>> Number of sequences          : 3
>> Maximum tokens in sequences  : 4


   === RESULT: ===

7
6 6
7A 1C 1C 1C 55 BD
7A 1C E9 BD BD 7A
E9 BD BD 1C 55 55
7A 55 E9 E9 1C 1C
55 E9 E9 7A 1C 7A
1C BD 7A 7A BD 55
3
1C BD 7A
59
55 7A
29
BD E9
17

   === CONFIRM INPUT ===
>> Proceed with this input? (y/n)>> y
Press any key to continue . . .


   === RESULT: ===

105
1C BD 7A BD E9 55 7A
2, 1
2, 6
3, 6
3, 3
1, 3
1, 5
4, 5

47 ms
```

```
   === CONFIRM INPUT ===
>> Proceed with this input? (y/n)>> y
Press any key to continue . . .


   === RESULT: ===

105
1C BD 7A BD E9 55 7A
2, 1
2, 6
3, 6
3, 3
1, 3
1, 5
4, 5

47 ms


   === SAVE ===

Do you want to save the solution? (y/n)
>> y
>> Filename (without extension): random1
>> Saved to ../test/random1.txt
```

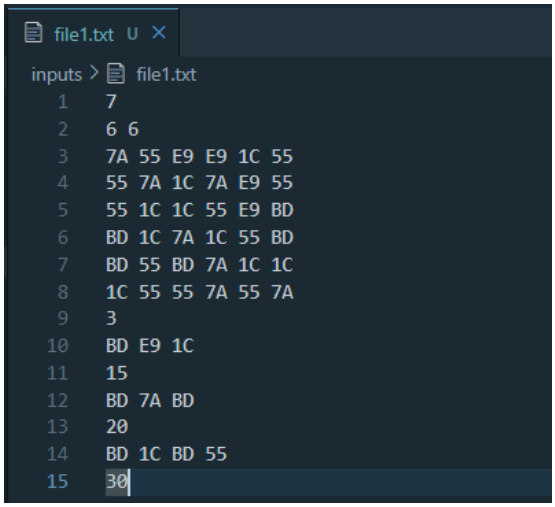
3.4 Hasil Random Input 2

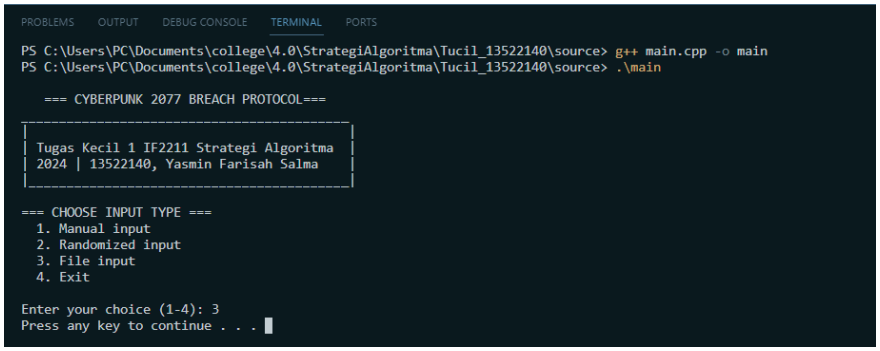
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> g++ main.cpp -o main
PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> .\main

    === CYBERPUNK 2077 BREACH PROTOCOL===
 _____________________________________
|                                     |
| Tugas Kecil 1 IF2211 Strategi Algoritma  |
| 2024 | 13522140, Yasmin Farisah Salma    |
|_____________________________________|

=== CHOOSE INPUT TYPE ===
  1. Manual input
  2. Randomized input
  3. File input
  4. Exit

Enter your choice (1-4): 2
Press any key to continue . . .
```

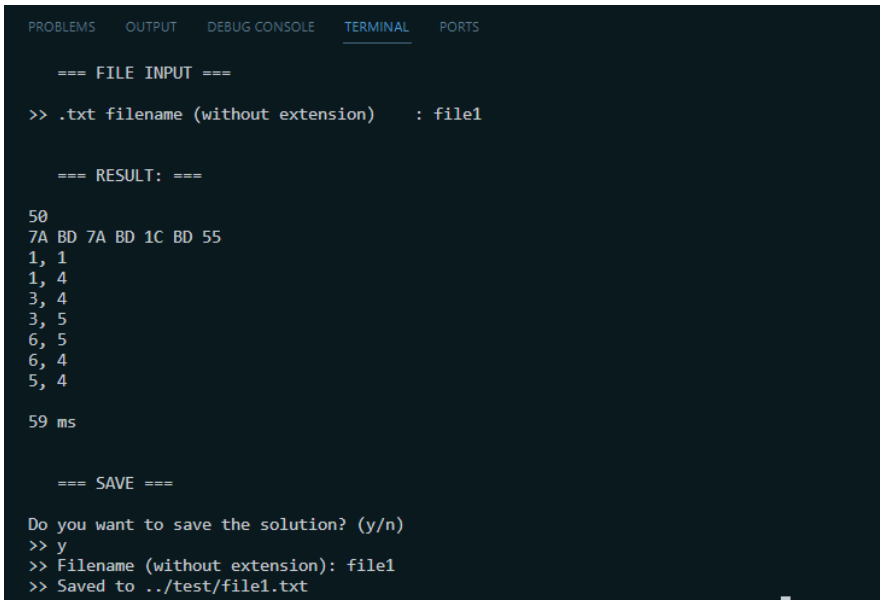

PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
=== RANDOMIZED INPUT ===
>> Number of unique tokens : 5
>> Tokens (must be unique) : BD 1C 7A 55 E9
>> Buffer size : 7
>> Matrix dimension (w h) : 6 6
>> Number of sequences : 3
>> Maximum tokens in sequences : 4
=== RESULT: ===
7
6 6
7A 1C 1C 1C 55 BD
7A 1C E9 BD BD 7A
E9 BD BD 1C 55 55
7A 55 E9 E9 1C 1C
55 E9 E9 7A 1C 7A
1C BD 7A 7A BD 55
3
1C BD 7A
59
55 7A
29
BD E9
17
=== CONFIRM INPUT ===
>> Proceed with this input? (y/n)>> y
Press any key to continue . . .
=== RESULT: ===
105
1C BD 7A BD E9 55 7A
2, 1
2, 6
3, 6
3, 3
1, 3
1, 5
4, 5
58 ms


=== RESULT: ===
105
1C BD 7A BD E9 55 7A
2, 1
2, 6
3, 6
3, 3
1, 3
1, 5
4, 5
58 ms
=== SAVE ===
Do you want to save the solution? (y/n)
>> y
>> Filename (without extension): random2
>> Saved to ../test/random2.txt

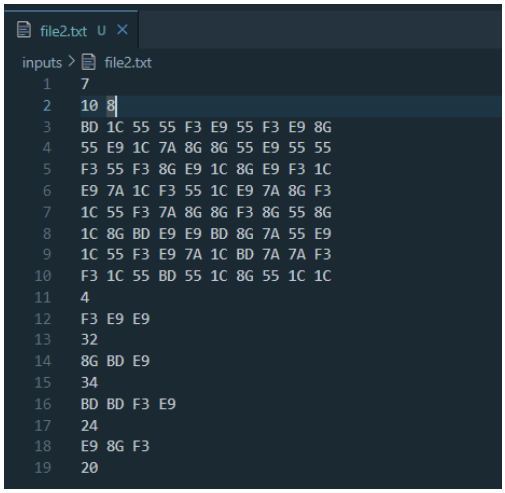
3.5 Hasil File Input 1



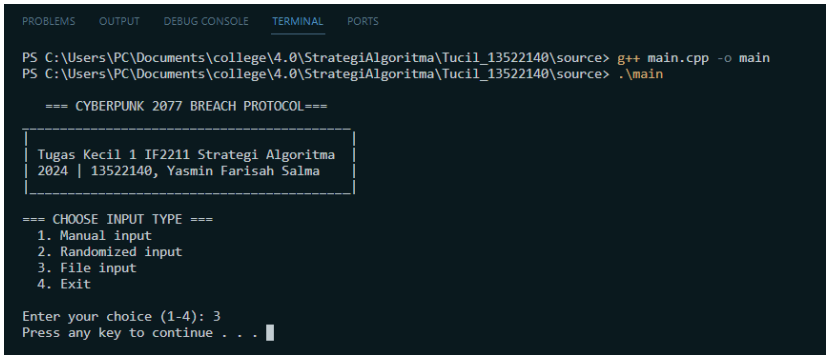




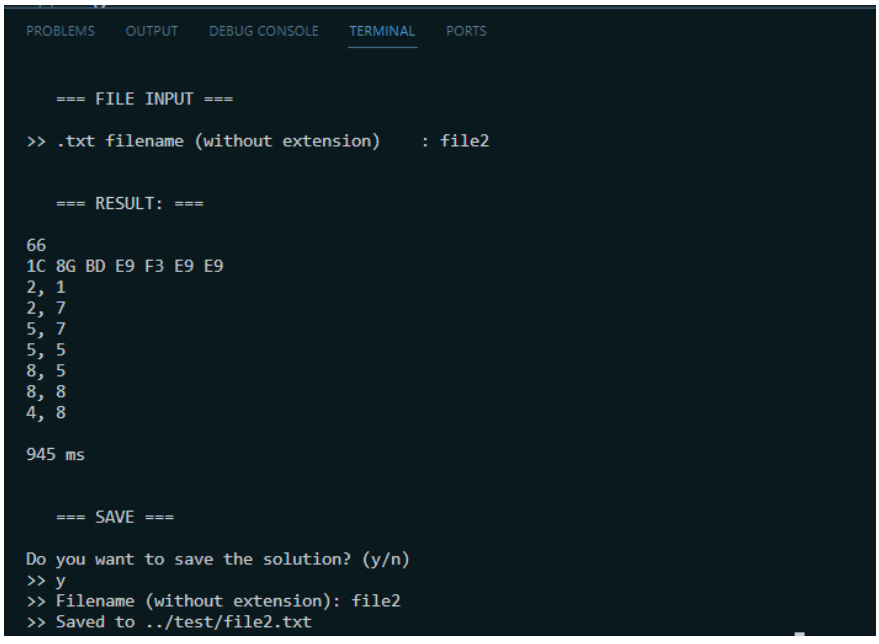
3.6 Hasil File Input 2



```
file2.txt U
inputs > file2.txt
1   7
2   10 8
3   BD 1C 55 55 F3 E9 55 F3 E9 8G
4   55 E9 1C 7A 8G 8G 55 E9 55 55
5   F3 55 F3 8G E9 1C 8G E9 F3 1C
6   E9 7A 1C F3 55 1C E9 7A 8G F3
7   1C 55 F3 7A 8G 8G F3 8G 55 8G
8   1C 8G BD E9 E9 BD 8G 7A 55 E9
9   1C 55 F3 E9 7A 1C BD 7A 7A F3
10  F3 1C 55 BD 55 1C 8G 55 1C 1C
11  4
12  F3 E9 E9
13  32
14  8G BD E9
15  34
16  BD BD F3 E9
17  24
18  E9 8G F3
19  20
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> g++ main.cpp -o main
PS C:\Users\PC\Documents\college\4.0\StrategiAlgoritma\Tucil_13522140\source> .\main

   === CYBERPUNK 2077 BREACH PROTOCOL===
 _____________________________________
|                                     |
| Tugas Kecil 1 IF2211 Strategi Algoritma |
| 2024 | 13522140, Yasmin Farisah Salma   |
|_____________________________________|

=== CHOOSE INPUT TYPE ===
  1. Manual input
  2. Randomized input
  3. File input
  4. Exit

Enter your choice (1-4): 3
Press any key to continue . . .
```



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

   === FILE INPUT ===

>> .txt filename (without extension)    : file2

   === RESULT: ===

66
1C 8G BD E9 F3 E9 E9
2, 1
2, 7
5, 7
5, 5
8, 5
8, 8
4, 8

945 ms

   === SAVE ===

Do you want to save the solution? (y/n)
>> y
>> Filename (without extension): file2
>> Saved to ../test/file2.txt
```

REFERENSI

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag2.pdf

https://core.ac.uk/download/pdf/288088999.pdf