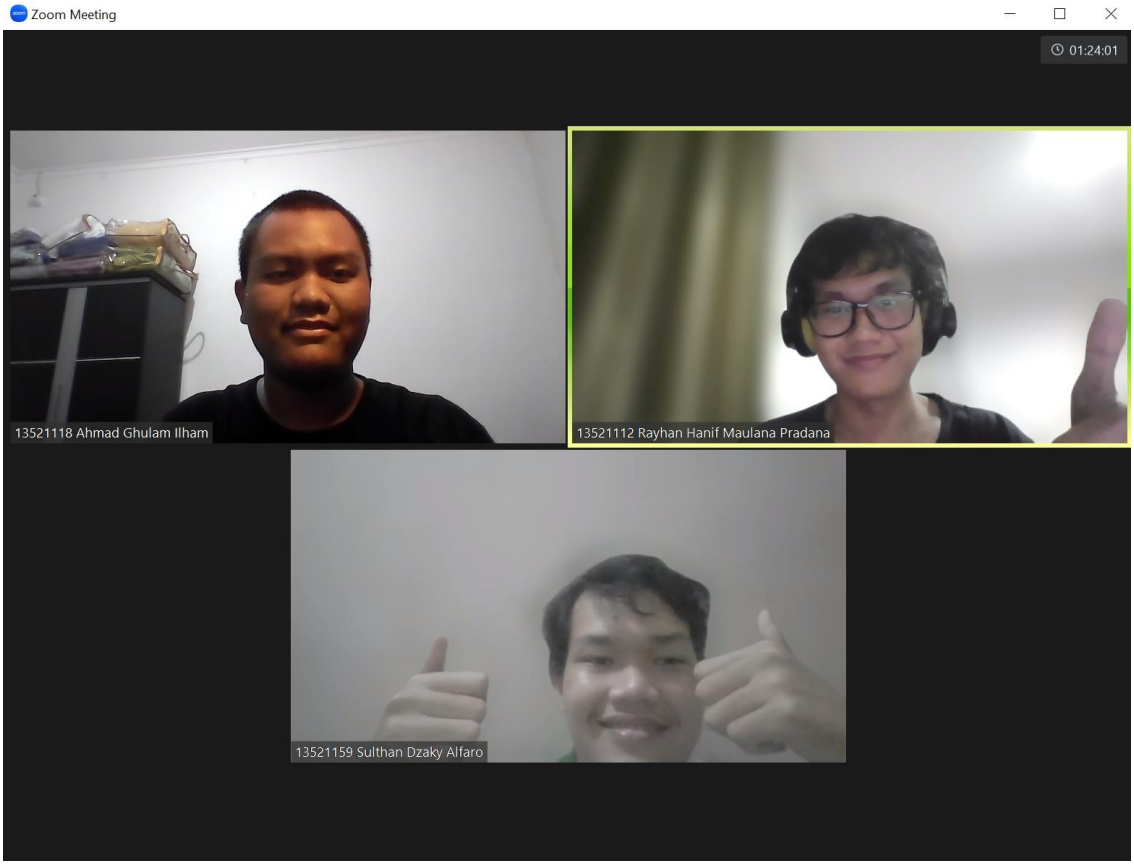
Tugas Besar 2 Strategi Algoritma IF2211

Pengaplikasian Algoritma BFS dan DFS dalam Menyelesaikan Persoalan Maze Treasure Hunt



Disusun oleh:

1. Muhammad Rayhan Hanif – 13521112
2. Ahmad Ghulam Ilham – 13521118
3. Sulthan Dzaky Alfaro – 13521159

Teknik Informatika

Sekolah Elektro dan Informatika

Tahun 2022/2023

Bab 1

Deskripsi Tugas

Tuan Krabs menemukan sebuah labirin distorsi terletak tepat di bawah Krusty Krab bernama El Doremi yang Ia yakini mempunyai sejumlah harta karun di dalamnya dan tentu saja Ia ingin mengambil harta karunnya. Dikarenakan labirinnya dapat mengalami distorsi, Tuan Krabs harus terus mengukur ukuran dari labirin tersebut. Oleh karena itu, Tuan Krabs banyak menghabiskan tenaga untuk melakukan hal tersebut sehingga Ia perlu memikirkan bagaimana caranya agar Ia dapat menelusuri labirin ini lalu memperoleh seluruh harta karun dengan mudah.

Gambar 1. Labirin di Bawah Krusty Krab

(Sumber:https://static.wikia.nocookie.net/theloudhouse/images/e/ec/Massive_Mustard_Pockt.png/revision/latest?cb=20180826170029)

Setelah berpikir cukup lama, Tuan Krabs tiba-tiba mengingat bahwa ketika Ia berada pada kelas Strategi Algoritma-nya dulu, Ia ingat bahwa Ia dulu mempelajari algoritma BFS dan DFS sehingga Tuan Krabs menjadi yakin bahwa persoalan ini dapat diselesaikan menggunakan kedua algoritma tersebut. Akan tetapi, dikarenakan sudah lama tidak menyentuh algoritma, Tuan Krabs telah lupa bagaimana cara untuk menyelesaikan persoalan ini dan Tuan Krabs pun kebingungan. Tidak butuh waktu lama, Ia terpikirkan sebuah solusi yang brilian. Solusi tersebut adalah meminta mahasiswa yang saat ini sedang berada pada kelas Strategi Algoritma untuk menyelesaikan permasalahan ini.

Dalam tugas besar ini, diminta untuk membangun sebuah aplikasi dengan GUI sederhana yang dapat mengimplementasikan BFS dan DFS untuk mendapatkan rute memperoleh seluruh treasure atau harta karun yang ada. Program dapat menerima dan membaca input sebuah file txt yang berisi maze yang akan ditemukan solusi rute mendapatkan treasure-nya.

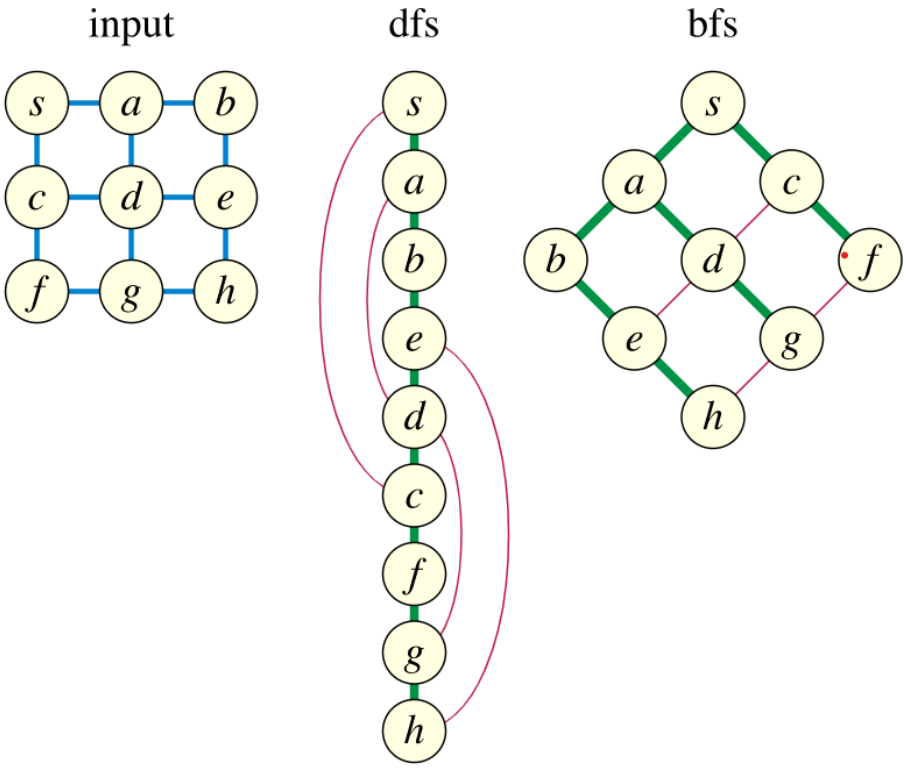
Bab 2

Landasan Teori

○ **Dasar teori**

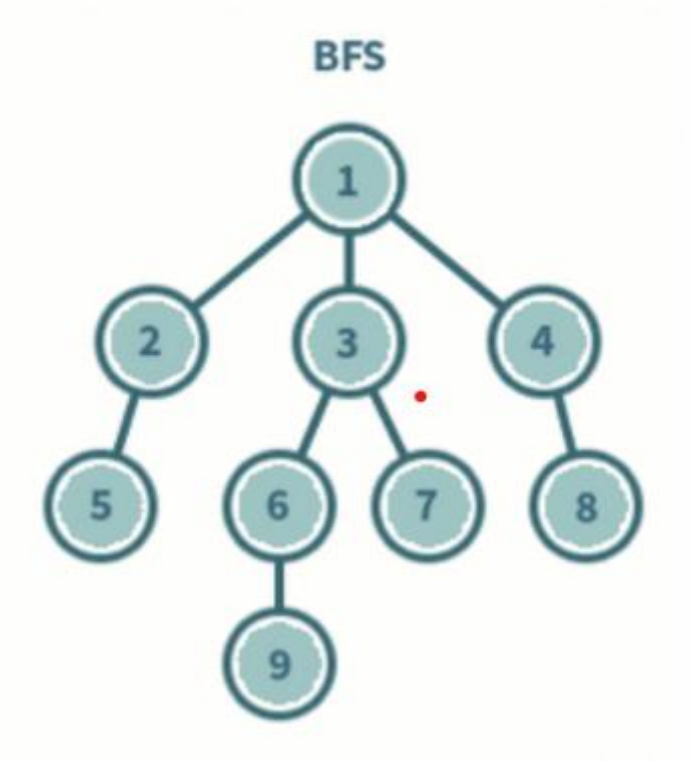
A. Graph Traversal

Graph traversal merupakan metode untuk menentukan posisi titik dalam sebuah graph. Sebelumnya graph merupakan kumpulan titik titik dan garis dimana pasangan-pasangan titik dihubungkan dengan garis. Titik-titik tersebut biasa disebut vertex dan garisnya biasa disebut edge. Graph treversal berarti mengunjungi semua titik/simpul dan garis yang ada di graph tepat 1 kali. Apabila ingin menggunakan graph traversal ini harus dipastikan setiap simpul harus dikunjungi tepat 1 kali. Urutan dimana simpul dikunjungi terlebih dahulu sangat penting dan bergantung pada algoritma yang ingin dipecahkan. Terdapat 2 algoritma dari graph traversal, yaitu Breadth-First Search dan Depth-First Search.



B. BFS

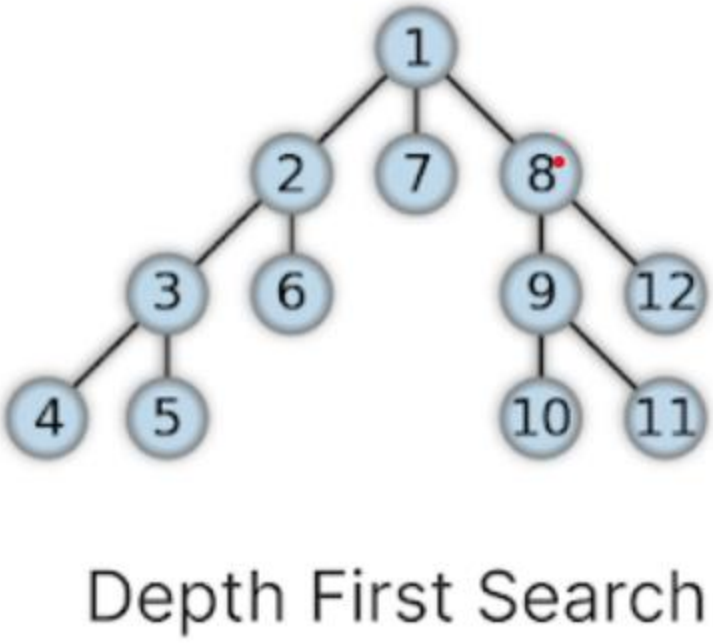
BFS atau Breadth-First Search merupakan algortima yang digunakan untuk membuat grafik data dan mencari atau melintasi struktur graph. Algoritma ini mengunjungi semua node/simpul dalam sebuah graph dengan cara yang akurat. Algoritma ini bekerja dengan cara mengunjungi salah satu simpul lalu mengunjungi semua simpul yang saling berhubungan dengan node yang dipilih lalu menandai simpul sudah dikunjungi hingga semua simpul pada graph berhasil dikunjungi dan ditandai. Untuk algoritma ini menggunakan struktur data queue.



C. DFS

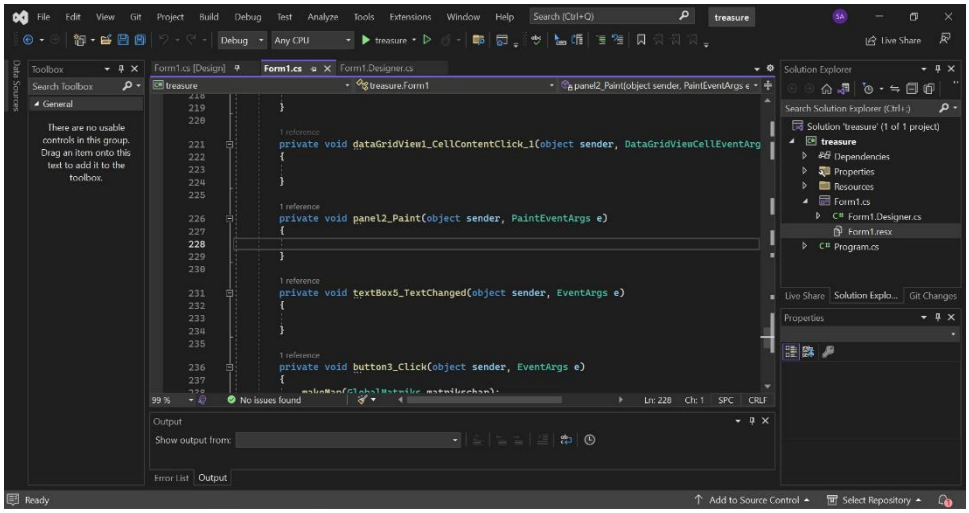
DFS atau Depth-First Search merupakan algoritma untuk menemukan atau melintasi sebuah graph maupun tree dengan arah lintasan secara mendalam. Cara kerja algoritma DFS ini adalah dengan cara dimulai dari simpul akar dan menjelajahi setiap cabang sebelum proses backtracking(mundur menjelajahi cabang lain). Algoritma ini menggunakan struktur data stack untuk mengingat, mendapatkan simpul berikutnya, dan untuk memulai pencarian setiap kali jalan buntu muncul di setiap iterasi. Di setiap iterasi, algoritma DFS menambahkan sebuah simpul anak dari simpul yang saat ini, lalu selanjutnya ambil salah satu anak dari anaknya(cucu) dari simpul yang dipilih dan seterusnya sampai menemukan yang ingin dituju.

Apabila ada salah satu anak yang buntu, dilakukan backtracking ke parent dari simpul anak dan mencari simpul anak yang lain dari parent apabila ada. Apabila tidak ada, lakukan backtracking sampai menemukan parent yang memiliki simpul anak yang lain.



○ **Penjelasan C# desktop application development**

Untuk C# Desktop Application Development, kami menggunakan Windows Form Visual Studio. Windows Forms adalah sebuah nama yang diberikan untuk elemen antarmuka dengan pengguna berbasis grafik yang dimasukkan ke dalam Microsoft .NET Framework, yang mengizinkan akses terhadap elemen-elemen antarmuka grafik sistem operasi Windows dengan memasukkan Windows API yang telah ada ke dalam managed code. Aplikasi ini menggunakan bahasa C# untuk membangun sebuah aplikasi yang nantinya dapat digunakan untuk kepentingan tertentu.


File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
treasure
Debug
Any CPU
Live Share
Toolbox
Search Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Form1.cs [Design]
Form1.cs
Form1.Designer.cs
treasure
treasure.Form1
panel2_Paint(object sender, PaintEventArgs e
1 reference
private void dataGridView1_CellContentClick_1(object sender, DataGridViewCellEventArg
{
}
1 reference
private void panel2_Paint(object sender, PaintEventArgs e)
{
}
1 reference
private void textBox5_TextChanged(object sender, EventArgs e)
{
}
1 reference
private void button3_Click(object sender, EventArgs e)
{
99 %
No issues found
Ln: 228
Ch: 1
SPC
CRLF
Output
Show output from:
Error List
Output
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'treasure' (1 of 1 project)
treasure
Dependencies
Properties
Resources
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
Live Share
Solution Explo...
Git Changes
Properties
Ready
Add to Source Control
Select Repository

Bab 3

Landasan Teori

○ Langkah-langkah pemecahan masalah

Pemecahan Masalah pada DFS:

Inisialisasi

1. Membuat struktur data Node dan Map of Nodes berdasarkan isi file .txt yang diberikan.
2. Menentukan start Node sebagai Node dengan karakter 'K', serta menentukan semua Treasure Node yang terdapat pada Map.
3. Melakukan pemanggilan pencarian rute secara rekursif dimulai dari start Node.

Rekursi

4. Urutan prioritas Node yang akan dikunjungi adalah sebagai berikut: Kanan, Bawah, Kiri, Atas. Node satu per satu dikunjungi secara rekursif hingga seluruh treasure berhasil ditemukan.
5. Catat urutan Node menggunakan struktur data Route. Jika ditemukan Treasure Node, tambahkan treasure pada Route. Selama jumlah treasure pada Route belum sesuai dengan jumlah treasure pada Map, DFS akan terus dipanggil secara rekursif.

Basis

6. Jika mencapai Dead End (Node tidak memiliki arah gerakan lagi), maka akan dilakukan backtrack sehingga Node akan bergerak menuju previous Node dan dilanjut melakukan pemanggilan DFS secara rekursif dari parent Node tersebut.
7. Jika jumlah treasure Route sudah sesuai dengan jumlah treasure Map, pemanggilan DFS dihentikan dan Route berhasil ditemukan.

Pemecahan Masalah pada BFS:

Inisialisasi

1. Membuat struktur data Node dan Graph of Nodes yang berbentuk adjaceny list berdasarkan isi file.txt yang diberikan.

2. Menentukan jumlah dari node treasure, serta menentukan start Node dengan karakter ‘K’.
3. Membuat sebuah queue dan enqueue dengan start Node. Start Node tersebut dimark (artinya telah dikunjungi).
4. Dequeue, dan kunjungi node hasil dequeue, kemudian enqueue setiap node tetangga dari node yang di dequeue tersebut yang belum dikunjungi.
5. Apabila ditemukan Node dengan tipe Node ‘T’, yang artinya treasure, maka simpan Node tersebut.
6. Selama queue belum kosong atau semua treasure belum ditemukan, ulangi langkah-langkah tersebut dimulai dengan langkah 4.

○ **Proses mapping persoalan menjadi elemen-elemen algoritma BFS dan DFS**

Mapping Elemen pada DFS:

1. Node (Problem State): Semua karakter pada isi file .txt yang menyimpan informasi arah gerakan DFS.
2. Start Node (Initial State): Node dengan karakter adalah 'K'.
3. Treasure Node (Solution State): Node dengan karakter 'T'.
4. Route (Solution Space): Catatan arah gerakan yang dilakukan oleh DFS selama pencarian Treasure Node.

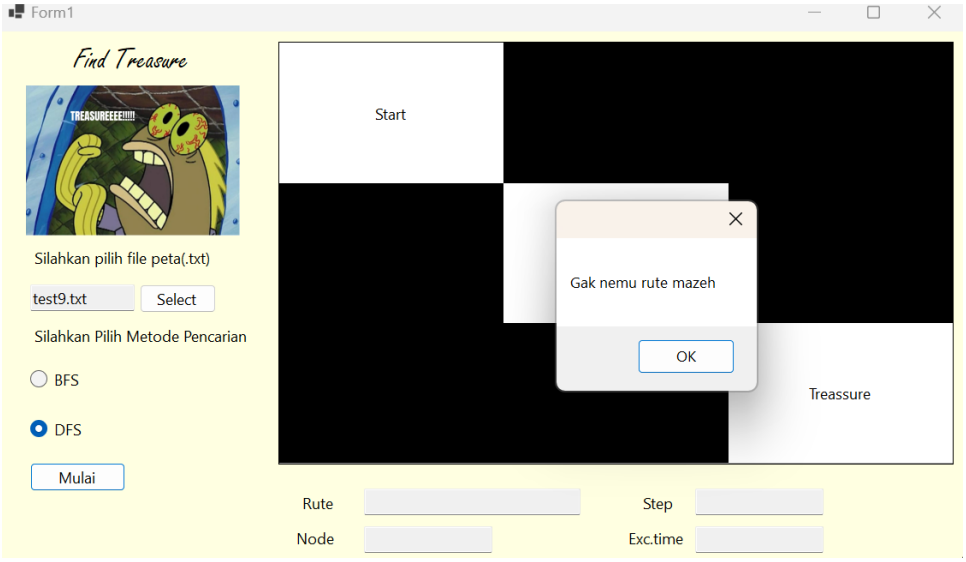
Mapping Elemen pada BFS:

1. Node (Problem State): Semua karakter pada isi file .txt yang menyimpan informasi arah gerakan BFS.
2. Start Node (Initial State): Node dengan karakter adalah 'K'.
3. Treasure Node (Solution State): Node dengan karakter 'T'.
4. Route (Solution Space): Rute yang melewati semua treasure, dimana rute tersebut hanya melewati node yang telah dikunjungi saat BFS.

○ **Contoh ilustrasi kasus lain yang berbeda dengan contoh pada spesifikasi tugas**

Kasus Lain 1: Terdapat Treasure Node yang sama sekali tidak dapat dicapai dari Start Node

Jika terdapat Treasure Node yang sama sekali tidak dapat dicapai dari Start Node maka Program akan menampilkan pesan tidak terdapat rute yang ditemukan. Contoh visualisasi kasus:



Bab 4

Analisis Pemecahan Masalah

○ **Implementasi program (pseudocode program utama).**

Implementasi program DFS:

Metode di bawah ini adalah implementasi dari penjelasan langkah pada Bab 3, yaitu DFS yang dipanggil secara rekursif untuk mencari Route yang mendapatkan semua Treasure Node.

```
public class DFS {
    // atribut
    15 references
    private int curRow;
    15 references
    private int curCol;
    12 references
    private Node curNode;
    3 references
    private bool tsp;

    // ctor
    0 references
    public DFS(Map map) {
        for (int i = 0; i < map.GetRowCount(); i++) {
            for (int j = 0; j < map.GetColCount(); j++) {
                if (map.GetElmt(i,j).GetSymbol() == 'K') {
                    curRow = i;
                    curCol = j;
                }
            }
        }
        curNode = map.GetElmt(curRow,curCol);

    }

    // setter
    0 references
    public void SetTSP(bool newTSP) {
        tsp = newTSP;
    }

    // getter
    0 references
    public int GetRow() {
        return curRow;
    }
    // getter
    0 references
    public int GetCol() {
        return curCol;
    }
    // getter
    0 references
    public bool GetTSP() {
        return tsp;
    }

    // method
    4 references
    public void FindRoute(int newRow, int newCol, Map newMap, Route newRoute) {
        Route Copy = new Route(newRoute);
        Console.Write("Current Route: ");
        while (Copy.GetElmt().Count != 0) {
            Console.Write(Copy.GetElmt().Pop());
            if (Copy.GetElmt().Count != 0) {
                Console.Write("->");
            }
        }
        Console.WriteLine();

        int prevRow = curRow;
        int prevCol = curCol;
        Node prevNode = curNode;
        curRow = newRow;
        curCol = newCol;
        curNode = newMap.GetElmt(curRow,curCol);

        curNode.SetStatus("Visiting");

        if (newRoute.GetTreasureCount() == newMap.GetTreasureCount()) {
            newRoute.SetStatus("Complete");
        }
        if ((tsp) && (newRoute.GetStatus() == "Complete")) {
            if (TSP.GetElmt().Count == 0 || (newRoute.GetElmt().Count < TSP.GetElmt().Count)) {
                Route route = new Route(newRoute);
                TSP.SetElmt(route);
            }
            newRoute.SetStatus("Incomplete");
        }

        // Console.WriteLine("Checking Right Position");
        if ((curNode.GetBool("right")) && (newRoute.GetStatus() != "Complete")) {
            int nextCol = curCol+1;
            Node nextNode = newMap.GetElmt(curRow, nextCol);
            if (nextNode.GetStatus() == "Unvisited") {
                if (nextNode.GetSymbol() == 'T') {
                    newRoute.AddTreasure();
                }
                newRoute.GetElmt().Push('R');
                int[] simpan = new int[2];
                simpan[0] = curRow;
                simpan[1] = nextCol;
                cekRute.simpanrute.Push(simpan);
                cekRute.semuarute.Add(simpan);
                // Console.WriteLine("Adding R to Unreversed Route");
                FindRoute(curRow, nextCol, newMap, newRoute);
            }
        }
        // Console.WriteLine("Checking Down Position");
        if ((curNode.GetBool("down")) && (newRoute.GetStatus() != "Complete")) {
            int nextRow = curRow+1;
            Node nextNode = newMap.GetElmt(nextRow, curCol);
            if (nextNode.GetStatus() == "Unvisited") {
                if (nextNode.GetSymbol() == 'T') {
                    newRoute.AddTreasure();
                }
                newRoute.GetElmt().Push('D');
                int[] simpan = new int[2];
                simpan[0] = nextRow;
                simpan[1] = curCol;
                cekRute.simpanrute.Push(simpan);
                cekRute.semuarute.Add(simpan);
                // Console.WriteLine("Adding D to Unreversed Route");
                FindRoute(nextRow, curCol, newMap, newRoute);
            }
        }
        // Console.WriteLine("Checking Left Position");
        if ((curNode.GetBool("left")) && (newRoute.GetStatus() != "Complete")) {
            int nextCol = curCol-1;
            Node nextNode = newMap.GetElmt(curRow, nextCol);
            if (nextNode.GetStatus() == "Unvisited") {
                if (nextNode.GetSymbol() == 'T') {
                    newRoute.AddTreasure();
                }
                newRoute.GetElmt().Push('L');
                int[] simpan = new int[2];
                simpan[0] = curRow;
                simpan[1] = nextCol;
                cekRute.simpanrute.Push(simpan);
                cekRute.semuarute.Add(simpan);
                // Console.WriteLine("Adding L to Unreversed Route");
                FindRoute(curRow, nextCol, newMap, newRoute);
            }
        }
        // Console.WriteLine("Checking Up Position");
        if ((curNode.GetBool("up")) && (newRoute.GetStatus() != "Complete")) {
            int nextRow = curRow-1;
            Node nextNode = newMap.GetElmt(nextRow, curCol);
            if (nextNode.GetStatus() == "Unvisited") {
                if (nextNode.GetSymbol() == 'T') {
                    newRoute.AddTreasure();
                }
                newRoute.GetElmt().Push('U');
                int[] simpan = new int[2];
                simpan[0] = nextRow;
                simpan[1] = curCol;
                cekRute.simpanrute.Push(simpan);
                cekRute.semuarute.Add(simpan);
                // Console.WriteLine("Adding U to Unreversed Route");
                FindRoute(nextRow, curCol, newMap, newRoute);
            }
        }
    }
```

```
    // Console.WriteLine("Reached Dead End...");

    if (newRoute.GetStatus() != "Complete") {
        if (curNode.GetSymbol() == 'T') {
            newRoute.RemoveTreasure();
        }
        if (curNode.GetSymbol() != 'K') {
            char curChar = newRoute.GetElmt().Pop();
            int[] buang = cekRute.simpanrute.Pop();


            cekRute.semuarute.Add(buang);
            // Console.WriteLine("Removing " + curChar + " from Unreversed Route...");
        }
        curNode.SetStatus("Unvisited");
        curRow = prevRow;
        curCol = prevCol;
        curNode = prevNode;
    }
}
```

Implementasi program BFS:

Metode di bawah ini adalah algoritma BFS yang mencari akan berjalan hingga semua treasure ditemukan.

```
static public List<Node> bfs(Graph g, int treasureCount)    // Algoritma BFS utama
{
    Node n;
    Node end;
    Node startNode = g.getStartNode();
    List<Node> treasures = new List<Node>();
    List<Node> neighbors;
    Queue<Node> queue = new Queue<Node>();

    queue.Enqueue(startNode);
    startNode.setMarked(true);

    while (queue.Count > 0 && treasureCount > 0)
    {
        n = queue.Dequeue();
        neighbors = g.getNeighbors(n);
        for (int i = 0; i < neighbors.Count; i++)
        {
            if (!neighbors[i].getMarked())
            {
                if (neighbors[i].getNodeType() == 'T')
                {

                    end = neighbors[i];
                    treasures.Add(end);
                    treasureCount--;
                }
                queue.Enqueue(neighbors[i]);
                neighbors[i].setMarked(true);
                neighbors[i].setPrevNode(n);
            }
        }
    }
    return treasures;   // Mengembalikan semua treasure yang ditemukan untuk dibangun path/rutenya
}
```

Metode di bawah ini akan membangun rute dari start ke semua treasure

```
5 references
static public List<List<Node>> constructPath(List<Node> treasures)  // Untuk membangun path/rute dari treasure ke starting node, yaitu 'K'
{

    List<List<Node>> paths = new List<List<Node>>();

    for (int i = 0; i < treasures.Count; i++)
    {
        Node current = treasures[i];
        paths.Add(new List<Node>());
        while (current.getPrevNode() != null)
        {
            paths[i].Add(current);
            current = current.getPrevNode();
        }
        paths[i].Add(current);

        paths[i].Reverse();
    }

    return paths;
}
```

○ **Penjelasan struktur data yang digunakan dalam program dan spesifikasi program**

Struktur data yang digunakan pada DFS:

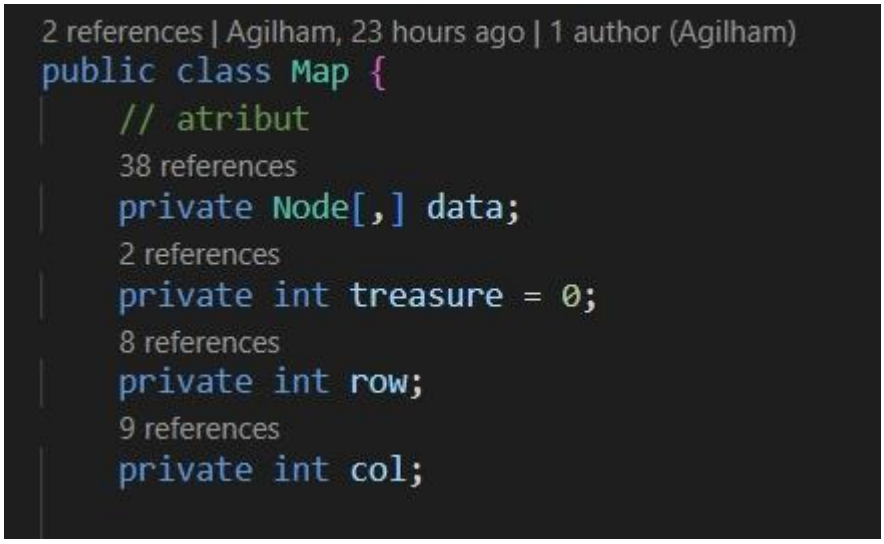
1. Node

```
11 references | Agilham, 23 hours ago | 1 author (Agilham)
public class Node {
    // atribut
    5 references
    private char symbol;
    3 references
    private string status;
    3 references
    private bool right;
    3 references
    private bool down;
    3 references
    private bool left;
    3 references
    private bool up;

    // ctor
    1 reference
    public Node(char elmt) {
        symbol = elmt;
        status = "";
        if (symbol == 'K') {
            SetStatus("Visiting");
        } else if (symbol == 'T' || symbol == 'R') {
            SetStatus("Unvisited");
        } else {
            SetStatus("Unvisitable");
        }
        right = false;
        down = false;
        left = false;
        up = false;
    }

    // setter
    5 references
    public void SetStatus(string newStatus) {
        status = newStatus;
    }
```

```
// setter
16 references
public void SetBool(string position, bool newBool) {
    if (position == "right") {
        right = newBool;
    } else if (position == "down") {
        down = newBool;
    } else if (position == "left") {
        left = newBool;      Agilham, 23 hours ago • fe
    } else {
        up = newBool;
    }
}

// getter
26 references
public char GetSymbol() {
    return symbol;
}
// getter
4 references
public string GetStatus() {
    return status;
}
// getter
4 references
public bool GetBool(string position) {
    if (position == "right") {
        return right;
    } else if (position == "down") {
        return down;
    } else if (position == "left") {
        return left;
    } else {
        return up;
    }
}

// overload output
public override string ToString() {
    return GetSymbol().ToString();
}
```

Node memiliki atribut symbol dan status yang merepresentasikan tipe karakter (K, R, T X) dan informasi apakah karakter tersebut sudah dikunjungi atau belum selama pencarian Treasure Node. Node juga memililiki atribut bool up, right, down, left yang merepresentasikan kemungkinan gerakan yang dapat dilakukan oleh Node tersebut.

2. Map


2 references | Agilham, 23 hours ago | 1 author (Agilham)
public class Map {
// atribut
38 references
private Node[,] data;
2 references
private int treasure = 0;
8 references
private int row;
9 references
private int col;

```
public Map(Matrix matrix) {
    row = matrix.GetRowCount();
    col = matrix.GetColCount();
    data = new Node[row,col];
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            Node node = new Node(matrix.GetElmt(i,j));
            data[i,j] = node;
        }
    }
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < col; j++) {
            if (data[i,j].GetSymbol() != 'X') {
                if (data[i,j].GetSymbol() == 'T') {
                    treasure++;
                }
                if (i == 0) {
                    if (j == 0) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                    } else if (j < col-1) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    } else {
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    }
                    if (data[i+1,j].GetSymbol() != 'X') {
                        data[i,j].SetBool("down",true);
                    }
                } else if (i < row-1) {
                    if (j == 0) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                    } else if (j < col-1) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    } else {
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    }
                    if (data[i-1,j].GetSymbol() != 'X') {
                        data[i,j].SetBool("up",true);
                    }
                    if (data[i+1,j].GetSymbol() != 'X') {
                        data[i,j].SetBool("down",true);
                    }
                } else {
                    if (j == 0) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                    } else if (j < col-1) {
                        if (data[i,j+1].GetSymbol() != 'X') {
                            data[i,j].SetBool("right",true);
                        }
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    } else {
                        if (data[i,j-1].GetSymbol() != 'X') {
                            data[i,j].SetBool("left",true);
                        }
                    }
                    if (data[i-1,j].GetSymbol() != 'X') {
                        data[i,j].SetBool("up",true);
                    }
                }
            }
        }
    }
```

Map yang digunakan dalam algoritma DFS berbentuk matrix 2D yang tersusun atas Node-Node berdasarkan isi file .txt. Map menyimpan jumlah Treasure Node, serta informasi ukuran Map, yaitu Row (jumlah baris) dan Col (jumlah kolom).

3. Route

```
7 references | Agilham, 21 hours ago | 1 author (Agilham)
public class Route {
    // atribut
    5 references
    private Stack<char> data;
    6 references
    private int treasure;
    5 references
    private string status;

    // ctor
    0 references
    public Route() {
        data = new Stack<char>();
        treasure = 0;
        status = "Incomplete";
    }
    // cctor
    2 references
    public Route(Route other) {
        data = new Stack<char>(other.data);
        treasure = other.treasure;
        status = other.status;
    }

    // setter
    1 reference
    public void SetElmt(Stack<char> newData) {
        data = newData;
    }
    // setter
    4 references
    public void AddTreasure() {
        treasure++;
    }
    // setter
    1 reference
    public void RemoveTreasure() {
        treasure--;
    }
    // setter
    2 references
    public void SetStatus(string newStatus) {
        status = newStatus;
    }

    // getter
    13 references
    public Stack<char> GetElmt() {
        return data;
    }
    // getter
    1 reference
    public int GetTreasureCount() {
        return treasure;
    }
    6 references
    public string GetStatus() {
        return status;
    }
```

Route memiliki atribut Stack<char> yang merepresentasikan urutan langkah gerakan yang dilakukan selama pencarian DFS secara rekursif. Setiap kali Node bergerak, akan di-push pada stack karakter sesuai gerakan yang baru saja dilakukan. Route juga memiliki atribut treasure, yaitu jumlah treasure yang ditemukan selama pencarian DFS. Terakhir, Route menyimpan atribut status, yang merepresentasikan apakah pencarian DFS sudah selesai atau belum. Status akan dibuah menjadi selesai jika dan hanya jika pencarian Route berhasil ditemukan dan valid, yaitu jumlah Treasure Node pada Route sama dengan jumlah Treasure Node pada Map.

Struktur data yang digunakan pada BFS:

1. Node

```
38 references
class Node
{
    2 references
    private int x;
    2 references
    private int y;
    2 references
    private char nodeType;
    3 references
    private bool marked;
    3 references
    private Node prevNode;

    1 reference
    public Node(int x, int y, char nodeType)
    {
        this.x = x;
        this.y = y;
        this.nodeType = nodeType;
        this.marked = false;
        this.prevNode = null;
    }

    13 references
    public int getX() { return this.x;}
    13 references
    public int getY() { return this.y;}
    3 references
    public char getNodeType() { return this.nodeType;}
    1 reference
    public bool getMarked() { return this.marked;}
    2 references
    public void setMarked(bool marked) { this.marked = marked;}
    2 references
    public Node getPrevNode() { return this.prevNode;}
    1 reference
    public void setPrevNode(Node prevNode) { this.prevNode = prevNode; }
}
```

Node memiliki atribut x dan y yang merepresentasikan posisi node dalam labirin, dimana node yang berada pada bagian ujung kiri atas memiliki nilai x dan y nol. Setiap bergerak ke kanan, nilai x bertambah satu dan sebaliknya. Setiap pergerakan ke bawah, nilai y bertambah satu dan sebaliknya. Tipe node nodeType merupakan character ‘R’, ‘K’, ‘X’, dan ‘T’. Marked digunakan untuk menandai apakah node sudah dikunjungi saat penelusuran BFS. Atribut prevNode adalah parent dari node tersebut.

2. Graf

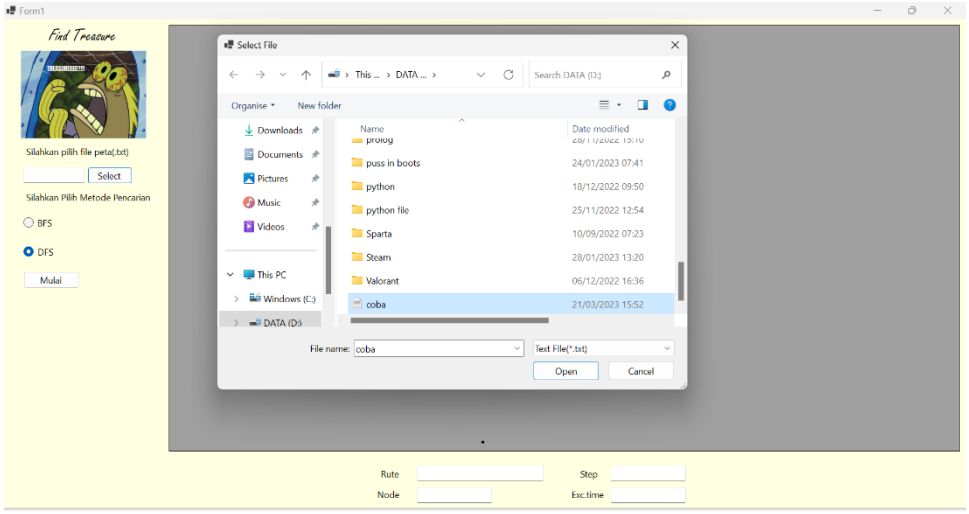
```
12 references
class Graph
{
    7 references
    private Dictionary<Node, List<Node>> graph;
    2 references
    private int noOfNodes;

    5 references
    public Graph()
    {
        this.graph = new Dictionary<Node, List<Node>>();
        this.noOfNodes = 0;
    }

    1 reference
    public List<Node> getNeighbors(Node n)
    {
        return this.graph[n];
    }
    1 reference
    public Node getStartNode()
    {
        foreach(Node node in this.graph.Keys)
        {
            if(node.getNodeType() == 'K')
            {
                return node;
            }
        }
        return this.graph.Keys.First();
    }
    1 reference
    public void addNode(Node n, List<Node> neighbors)
    {
        this.graph.Add(n, neighbors);
        this.noOfNodes++;
    }
    2 references
    public void addNeighbor(Node n, Node neighbor)
    {
        this.graph[n].Add(neighbor);
    }
    5 references
    public int getTreasureCount()
    {
        int treasureCount = 0;
        foreach(Node node in this.graph.Keys)
        {
            if(node.getNodeType() == 'T')
            {
                treasureCount++;
            }
        }
        return treasureCount;
    }
```

Graf yang digunakan dalam algoritma BFS berbentuk adjacency list, yang dalam implementasi C# menggunakan dictionary dengan key dianggap sebagai head node dan value merupakan list of node yang bertetanggaan dengan node key.

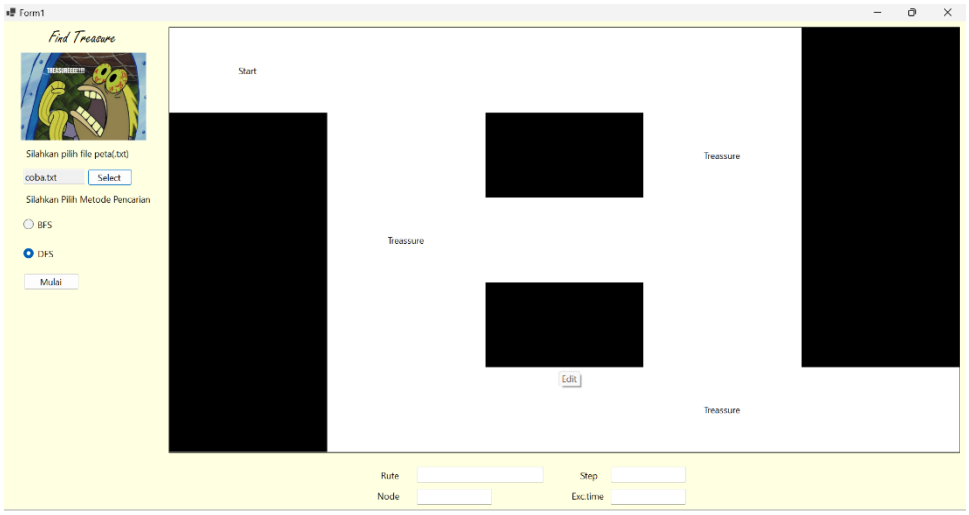
○ **Penjelasan tata cara penggunaan program**

Pertama-tama akan ditampilkan aplikasi sebagai berikut

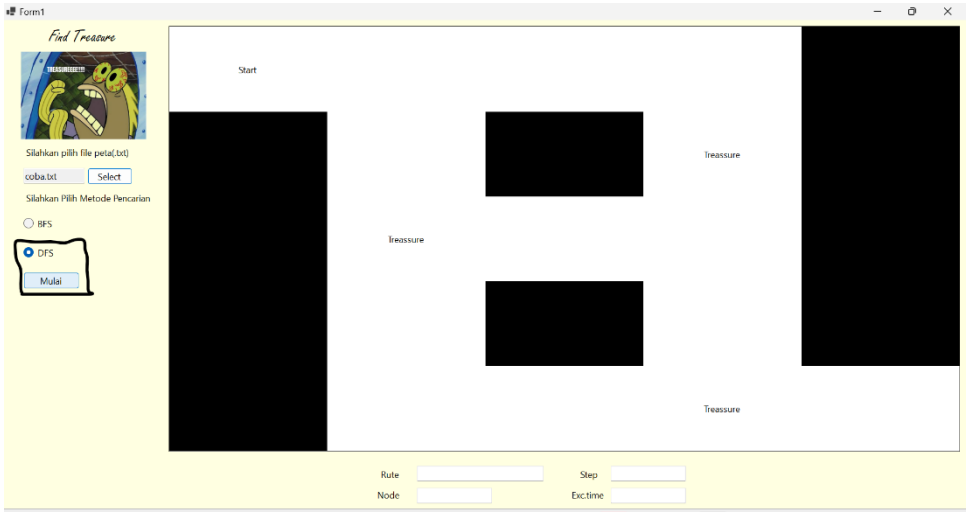
Kemudian pilih file map dengan menekan 'select' pada aplikasi. Pilih file txt untuk file map. Pastikan format sudah benar.



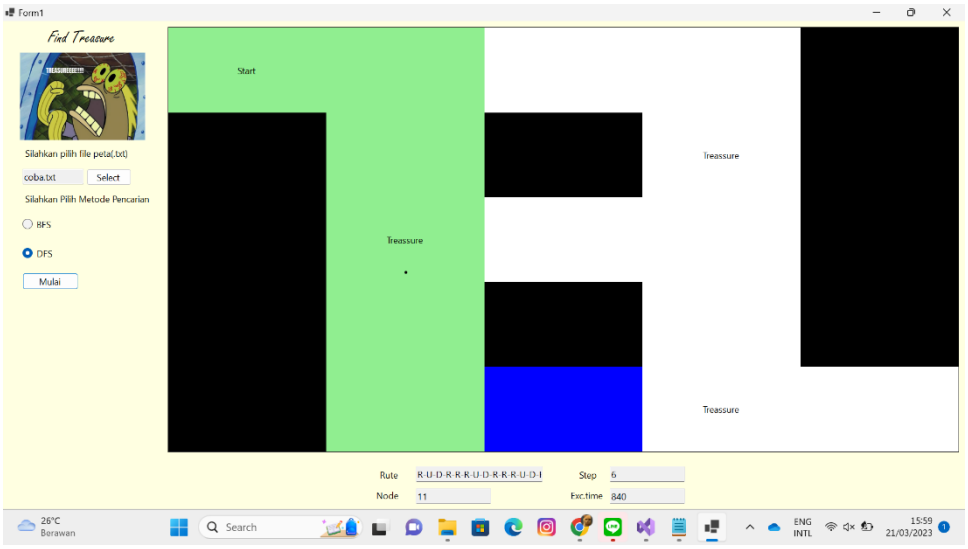
Lalu, aplikasi akan langsung menampilkan map dari file yang telah diinputkan.



Untuk memulai pencarian rute untuk mendapatkan harta karun, pilih salah 1 metode antara DFS dan BFS. Lalu setelah menentukan metode pencarian rute, klik 'Mulai' untuk memulai pencarian rute terbaik untuk algoritma yang telah dipilih.



Lalu program akan menelusuri rute berdasarkan algoritma yang dipilih. Jika sudah selesai, akan ditampilkan juga rute yang didapatkan beserta node dan step serta waktu eksekusi pada bagian bawah layar.

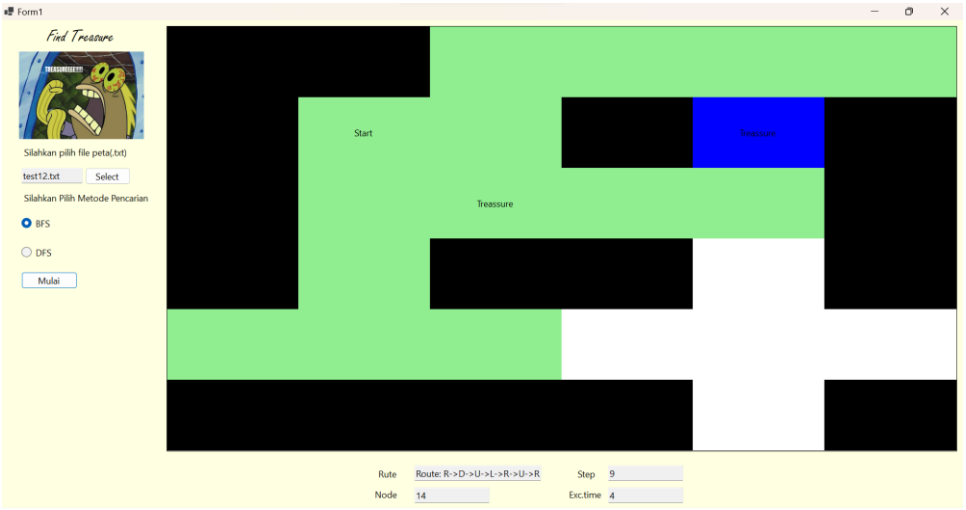


○ **Hasil pengujian (screenshot antarmuka program dan beberapa data uji beserta skenario pengujian). Diharapkan bahwa kelompok melakukan hasil pengujian untuk beberapa test case dengan ukuran input yang berbeda.**

Test Case 1

```
X X R R R R
X K R X T X
X R T R R X
X R X X R X
R R R R R R
X X X X R X
```

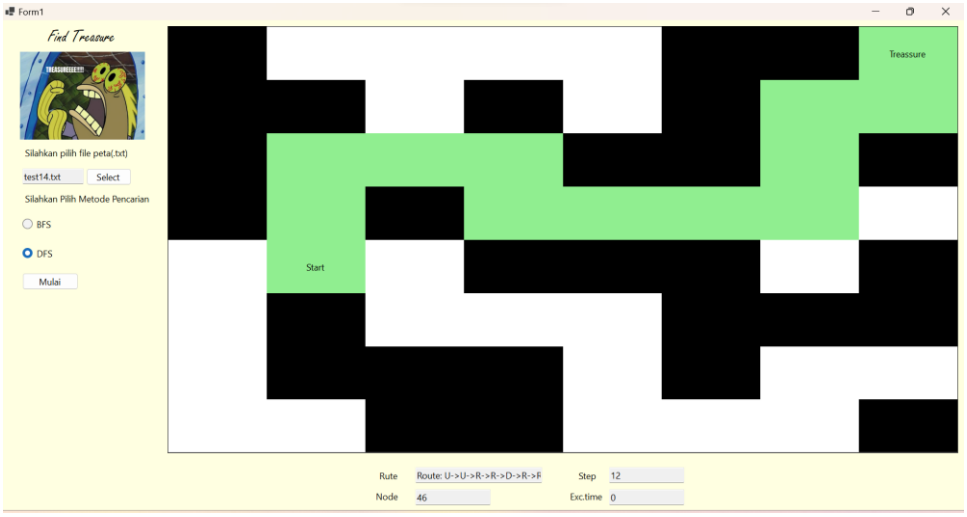
Route: R->D->R->R->U->Finished



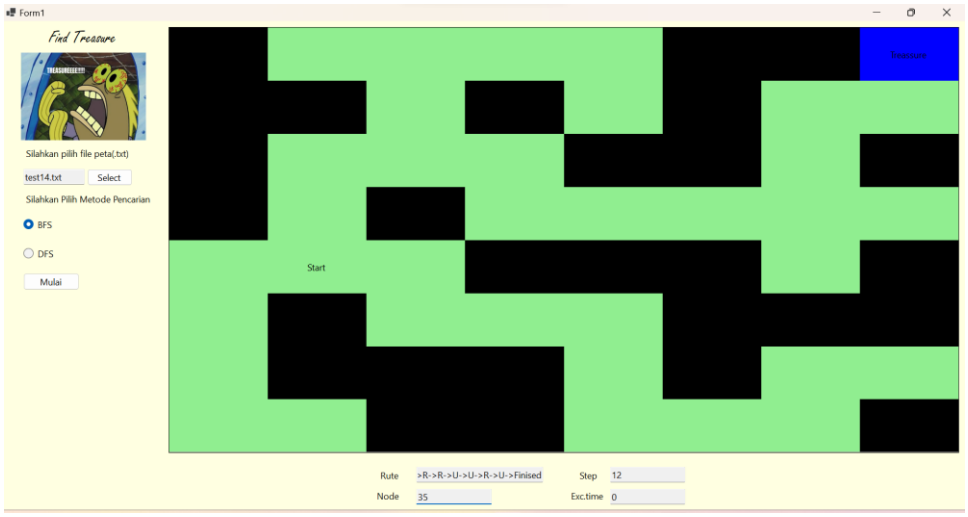
Route: R->D->U->L->R->U->R->R->D->Finised

Test Case 2

```
X R R R R X X T
X X R X R X R R
X R R R X X R X
X R X R R R R R
R K R X X X R X
R X R R R X X X
R X X X R X R R
R R X X R R R X
```



Route: U->U->R->R->D->R->R->R->U->U->R->U->Finished

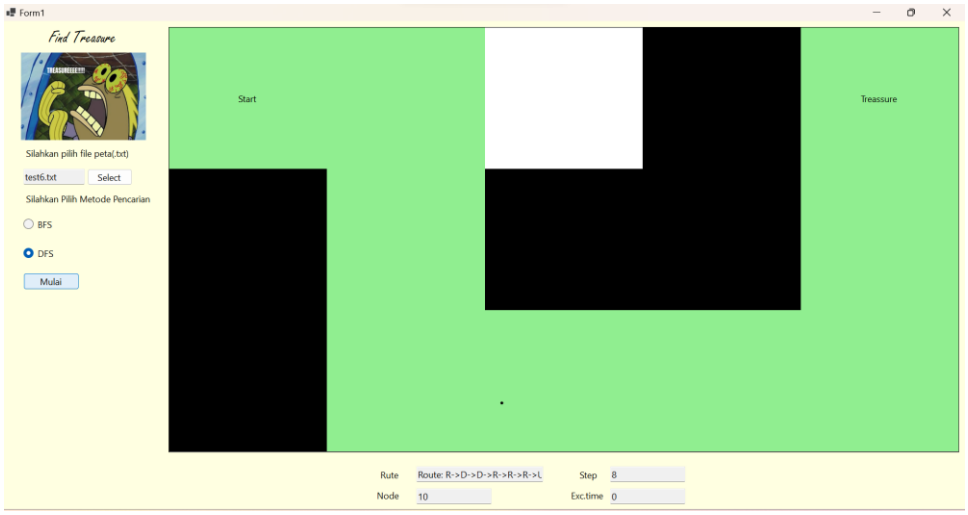


Route: U->U->R->R->D->R->R->R->U->U->R->U->Finished

Test Case 3

K R R X T
X R X X R
X R R R R

•



Route: R->D->D->R->R->R->U->U->Finished

Route: R->D->D->R->R->R->U->U->Finised

○ **Analisis dari desain solusi algoritma BFS dan DFS yang diimplementasikan pada setiap pengujian yang dilakukan. Misalnya adalah apakah strategi DFS lebih baik dari BFS pada kasus kasus tertentu, dan analisis kalian mengenai mengapa hal itu bisa terjadi**.

Analisis desain solusi DFS dan BFS:

Algoritma BFS, terutama untuk satu treasure atau end Node, lebih cenderung untuk mendapatkan shortest path. Algoritma BFS juga cenderung lebih baik ketika treasure atau end Node berada tersebar atau tidak jauh dari start Node. Hal tersebut dikarekan pada algoritma BFS, pencarian dilakukan secara melebar.

Sedangkan pada algoritma DFS, cenderung lebih baik ketika treasure berada jauh dengan start Node, yang artinya memiliki kedalaman tinggi terhadap start Node, dimana DFS membutuhkan waktu yang lebih singkat dibandingkan BFS. Tetapi, pada kasus tertentu algoritma DFS akan mengalami penurunan performa ketika memerlukan proses BackTracking, yaitu jika setelah mencapai kedalaman paling dalam tidak ditemukan solusi yang sesuai.

Bab 5

Kesimpulan dan Saran

○ Kesimpulan

Algortima DFS dan BFS merupakan algoritma yang digunakan untuk mencari rute pada sebuah graph traversal. Algoritma ini terbilang cukup efektif dalam pencarian rute. Walaupun hasil dari DFS dan BFS bisa saja berbeda, tetapi hasil yang diberikan cukup baik dibandingkan algoritma lain. Eksekusi program yang dihasilkan juga terbilang cepat daripada algoritma brute force.

○ Saran

Pada algoritma DFS, proses backtracking masih belum optimal, sehinga memungkinkan jumlah gerakan yang dilakukan sangat tinggi jika DFS perlu melakukan backtrack.

Pada algoritma BFS, rute yang ditemukan masih belum optimal, dimana ketika treasure lebih dari dua, rute yang dibentuk akan bercabang, yang menyebabkan perlunya kembali ke percabangan rute yang dimaksud atau bahkan kembali ke start node ketika telah mendapatkan satu node. Akan lebih baik jigk=

○ Refleksi

Tugas besar Strategi Algoritma ini sangat bermanfaat untuk kami. Manfaat yang telah kami dapat adalah bagaimana cara kerja DFS dan BFS dalam pencarian rute dalam sebuah peta/graph dan juga kami menjadi lebih mengetahui secara mendalam apa itu BFS dan DFS. Selain BFS dan DFS, kami juga dikenalkan bahasa pemrograman baru, yaitu C# sekaligus pembuatan aplikasi GUI dari bahasa tersebut.

○ Tanggapan anda terkait tugas besar ini.

Mantap, Sampai ketinggalan charger di kelas, di hari-H deadline pengumpulan :")

Daftar Pustaka

https://www.trivusi.web.id/2022/05/apa-itu-algoritma-breadth-first-search.html

https://11011110.github.io/blog/2013/12/17/stack-based-graph-traversal.html

https://www.trivusi.web.id/2023/01/perbedaan-dfs-dan-bfs.html

https://id.wikipedia.org/wiki/Windows_Forms

Link Github = https://github.com/SulthanDA28/Tubes2_COKLAAAAAT.git