Penggunaan Algoritma BFS dan DFS dalam Pencarian Solusi pada Permainan Puzzle "Unblock Me"

Muthmainnah / 13515059
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515059@std.stei.itb.ac.id

***Abstract*—Salah satu variasi permainan *puzzle* yang populer dimainkan sebagai *mobile application* adalah permainan "Unblock Me" atau bisa juga disebut *block sliding puzzle*. Pada permainan ini, tersedia sebuah papan dengan ukuran 6x6 satuan, dimana terdapat blok-blok yang dapat digeser-geser dan tujuan dari permainan ini adalah melakukan variasi gerakan pada blok-blok sehingga sebuah blok utama dapat dikeluarkan dari papan yang hanya memiliki satu jalan keluar. Variasi dalam meletakkan blok-blok pada papan dapat menghasilkan variasi tingkat kesulitan yang berbeda dalam permainan ini. Oleh karena itu, dapat digunakan algoritma BFS dan DFS dalam menemukan solusi permainan ini.**

Keywords—Unblock Me, Puzzle, BFS, DFS, Block Sliding.

I. PENDAHULUAN

Permainan berbasis *mobile application* dapat dikatakan sebagai permainan yang paling populer dimainkan saat ini, dimana hampir semua orang memiliki *smartphone* atau ponsel pintar. Salah satu jenis permainan berbasis *mobile application* tersebut adalah permainan *block sliding puzzle* dan salah satu contohnya adalah permainan bernama *Unblock Me*.

Gambar 1. Tampilan awal aplikasi *Unblock Me* saat dibuka
Sumber: http://assistant.9game.com/games-511564.html?template_id=607&p=and_ablums_pictxt_511564&text=and_ablums_pictxt_511564

Permainan ini dapat ditemukan di *playstore* pada *Android-based gadget* atau di *appstore* pada *iOS-based gadget*. Permainan ini biasanya dapat ditemukan pada bagian *arcade games*. Permainan ini banyak digemari karena aturannya yang cukup sederhana namun menantang karena terdapat berbagai variasi level dan tingkat kesulitan pada permainan ini. Tingkat kesulitan yang terdapat pada permainan ini adalah *beginner* (mudah), *intermediate* (sedang), *advanced* (sulit), *expert* (sangat sulit) dan *original free*. Sedangkan untuk level pada setiap tingkat kesulitan, yaitu terdapat 3000 level pada mode *beginner*, 1500 level pada mode *intermediate*, 400 level pada mode *intermediate*, 400 level pada mode *expert*, dan 1200 level pada mode *original free*. Maka dari itu, permainan ini memiliki total 6500 variasi level pada versi *free* atau gratis.

Pada permainan *Unblock Me* terdapat sebuah papan berukuran 6x6 satuan dengan penghalang/dinding di sekitarnya kecuali pada satu posisi di dinding bagian kanan pada baris ketiga, yang selanjutnya akan disebut sebagai jalan keluar. Hal ini disebabkan pada baris ketiga terdapat sebuah blok khusus atau selanjutnya disebut blok utama, yang memiliki warna berbeda dengan blok lainnya (biasanya berwarna merah). Blok utama ini berbentuk horizontal sehingga merupakan jenis blok yang dapat bergeser ke kanan atau ke kiri. Tujuan dari permainan *Unblock Me* ini adalah supaya blok utama dapat keluar dari papan permainan. Maka dari itu pemain harus dapat menyingkirkan blok-blok lain yang menghalangi gerakan blok utama untuk menuju jalan keluar. Blok-blok dapat berupa blok vertical atau blok horizontal, dimana blok vertical dapat bergerak ke atas atau ke bawah dan blok horizontal dapat bergerak ke kiri atau ke kanan.

Gambar 2. Tampilan permainan *Unblock Me* pada level 9 mode *intermediate*

Sumber: http://assistant.9game.com/games-511564.html?template_id=607&p=and_ablums_pictxt_511564&text=and_ablums_pictxt_511564

Pencarian solusi dari permainan *Unblock Me* ini dapat menghasilkan beberapa kemungkinan, yaitu dimana setiap blok dapat bergerak ke arah-arah tertentu sebanyak satuan tertentu pula. Kemudian dari gerakan blok tersebut dihasilkan "peta" permainan baru yang akan menghasilkan kemungkinan-kemungkinan gerakan blok selanjutnya, lalu begitu seterusnya sampai blok utama dapat menuju pintu keluar tanpa halangan.

Kemungkinan-kemungkinan yang muncul untuk menyelesaikan permainan ini bisa menjadi sangat banyak seiring bertambahnya tingkat kesulitan pada permainan.ini. Pencarian solusi ini dapat dibantu dengan membuat program yang menggunaka algoritma BFS atau DFS. Pada makalah ini, penulis akan menjelaskan penggunaan algoritma BFS dan DFS dalam mencari solusi permainan *Unblock Me* serta membandingkan kedua algoritma tersebut.

II. Dasar Teori

A. Aturan Permainan Unblock Me

Berikut ini merupakan detail dan aturan-aturan yang ada dalam permainan Unblock Me.

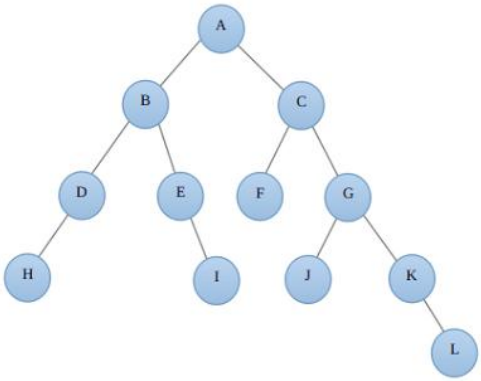
1. Papan permainan berukuran 6x6 satuan dan dikelilingi penghalang kecuali pada satu posisi di sebelah paling kanan pada baris ketiga yang berguna sebagai jalan keluar.
2. Terdapat blok-blok pada papan yang dapat berbentuk persegi panjang ke arah horizontal atau vertical dengan panjang 2 atau 3 satuan. Blok jenis horizontal hanya dapat bergerak ke kanan atau kiri, sedangkan blok vertical hanya dapat bergerak ke atas atau ke bawah.
3. Tujuan utama permainan ini adalah mengeluarkan blok utama dari papan permainan menuju pintu keluar. Blok utama berada pada baris ketiga dan berbentuk horizontal sepanjang 2 satuan dan berwarna berbeda dengan blok lain (biasanya merah).
4. Terdapat pilihan *undo* (batalkan gerakan) dan *restart* (ulang permainan) untuk setiap level serta pilihan *hints* (bantuan).

B. Algoritma BFS dan DFS

Algoritma traversal di dalam graf adalah mengunjungi simpul-simpul dengan cara sistematik. Algoritma traversal untuk graf terbagi menjadi dua, yaitu algoritma pencarian melebar atau *Breadth First Search* (BFS) dan algoritma pencarian mendalam atau *Depth First Search* (DFS). Gambaran umum untuk kedua algoritma ini adalah sebagai berikut.

1. *Breadth First Search* (BFS)
   - Traversal dimulai dari simpul v.
   - Algoritma: Kunjungi simpul v, lalu kunjungi semua simpul yang bertetangga dengan simpul v terlebih dahulu, kemudian kunjungi simpul yang belum dikunjungi dan bertetangga dengan simpul-simpul yang tadi dikunjungi, demikian seterusnya.
   - Jika graf berbentuk pohor berakar, maka semua simpul pada aras d dikunjungi lebih dahulu sebelum simpul-simpul pada aras d + 1.
2. *Depth First Search* (DFS)
   - Traversal dimulai dari simpul v.
   - Algoritma: Kunjungi simpul v, lalu kunjungi simpul w yang bertetangga dengan simpul v. Ulangi DFS mulai dari simpul w.
   - Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunutbalik ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
   - Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

Contoh penggunaan algoritma BFS dan DFS pada penelusuran sebuah graf ditunjukkan dengan mengacu pada gambar graf seperti berikut.



Gambar 3. Contoh graf

Sumber: http://www.postingankeren.com/2017/01/algoritma-bfs-dan-dfs.html

Penelusuran graf pada Gambar 3 menggunakan cara BFS menghasilkan urutan sebagai berikut.

A – B – C – D – E – F – G – H – I – J – K - L

Sedangkan penelusuran graf pada Gambar 3 menggunakan cara DFS menghasilkan urutan sebagai berikut.

A – B – D – H – E – I – C – F – G – J – K – L

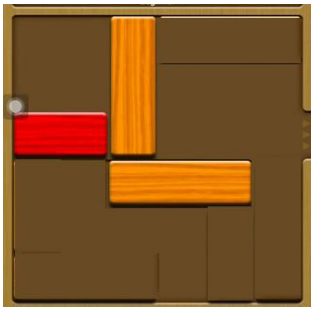
III. Analisis Permasalahan

Untuk menyelesaikan permainan *Unblock Me* menggunakan algoritma BFS dan DFS, penulis telah melakukan pengujian dengan program yang dibuat sendiri oleh penulis. Program ini diimplementasikan dalam bahasa pemrograman Java.

A. Penggunaan Program

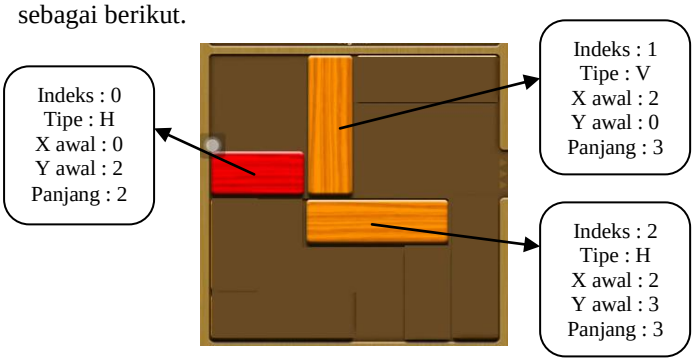
Program ini dapat dijalankan dengan memasukkan argument berupa "bfs" atau "dfs" sesuai algoritma yang ingin diterapkan pada program. Setelah itu, pengguna akan diminta memasukkan informasi yang berhubungan dengan permainan, yaitu jumlah blok pada papan dan posisi-posisi beserta ukuran blok tersebut. Informasi blok pertama yang dimasukkan haruslah informasi blok utama atau blok yang akan dikeluarkan dari papan pada akhir permainan. Informasi setiap blok dimasukkan satu per satu dan dipisahkan dengan *enter*, dengan setiap informasinya dipisahkan oleh spasi, dimana setiap blok memiliki informasi: jenis (vertical/horizontal), posisi x awal , posisi y awal, panjang blok (dalam satuan papan, biasanya 2 atau 3). Informasi tersebut dimasukkan secara berurutan. Posisi x awal dan y awal dari sebuah blok tidak boleh melampaui batas-batas papan, dalam hal ini batasnya adalah dari 0 sampai 5. Blok-blok akan memiliki indeks sesuai dengan urutan dimasukkannya blok dan indeks ini dimulai dari 0.

Output dari program ini adalah jumlah semua simpul yang dihidupkan, langkah-langkah penyelesaian permainan, serta waktu yang dibutuhkan untuk penyelesaian permainan tersebut. Berikut contoh ilustrasi penggunaan program.



Gambar 4. Contoh persoalan serupa permainan Unblock Me
Sumber: https://i.ytimg.com/vi/aeUQUSF_S_Q/maxresdefault.jpg (dengan perubahan oleh penulis)

Persoalan pada gambar 4 adalah persoalan pada permainan Unblock Me pada level 1 mode *beginner* yang telah disederhanakan agar menghasilkan simpul yang tidak terlalu banyak sehingga mudah untuk dijelaskan. Pada persoalan tersebut terdapat 3 buah blok yang memiliki keterangan sebagai berikut.



Gambar 5. Contoh persoalan dengan keterangan
Sumber: https://i.ytimg.com/vi/aeUQUSF_S_Q/maxresdefault.jpg (dengan perubahan oleh penulis)

Setelah didapatkan semua informasi mengenai blok-blok yang terdapat pada permainan, berikutnya adalah memasukkan informasi-informasi tersebut ke dalam program untuk dicari solusinya. Berikut hasil penggunaan program dengan persoalan seperti pada gambar 5 dengan menggunakan algoritma BFS dan DFS.

```
D:\FILE\IF\Semester 4\Stima makalah\unblockme\src>java UnblockMeSolver bfs
Jumlah block pada permainan (termasuk block utama) :
3
Masukkan informasi setiap block: <jenis(v/h)> <x_awal> <y_awal> <panjang>
(informasi block pertama yg dimasukkan haruslah informasi block utama)
h 0 2 2
v 2 0 3
h 2 3 3
Jumlah simpul yang dihidupkan (termasuk simpul 0) ada 14 simpul.
Solusi BFS :
Simpul 0 - posisi awal
Simpul 1 - block 2 ke arah d sejauh 1
Simpul 6 - block 1 ke arah s sejauh 3
Simpul 10 - block 0 ke arah d sejauh 4
Waktu pencarian solusi : 8 ms.
```

Gambar 6. Contoh penggunaan program dengan algoritma BFS

```
D:\FILE\IF\Semester 4\Stima makalah\unblockme\src>java UnblockMeSolver dfs
Jumlah block pada permainan (termasuk block utama) :
3
Masukkan informasi setiap block: <jenis(v/h)> <x_awal> <y_awal> <panjang>
(informasi block pertama yg dimasukkan haruslah informasi block utama)
h 0 2 2
v 2 0 3
h 2 3 3
Jumlah simpul yang dihidupkan (termasuk simpul 0) ada 11 simpul.
Solusi DFS :
Simpul 0 - posisi awal
Simpul 1 - block 2 ke arah d sejauh 1
Simpul 6 - block 1 ke arah s sejauh 3
Simpul 10 - block 0 ke arah d sejauh 4
Waktu pencarian solusi : 28 ms.
```

Gambar 7. Contoh penggunaan program dengan algoritma DFS

B. Penjelasan Singkat Mengenai Program

Program memiliki beberapa kelas, diantaranya:

- Step
- Block

- Simpul
- UnblockMeSolver

Kelas Step merupakan kelas yang merepresentasikan langkah yang dilakukan pada permainan ketika sebuah simpul dibentuk. Kelas Step memiliki atribut: idSimpul yang merupakan indeks simpul yang menyebabkan langkah tersebut terjadi, idBlock yang merupakan indeks blok yang berpindah pada langkah ini, direction yang merupakan arah perpindahan blok dengan indeks idBlock, dan terdapat atribut weight yang merupakan panjang perpindahan blok dengan idBlok ke arah yang sesuai dengan direction. Berikut prototype kelas Step.

```
public class Step {
    public int idBlock;
    public char direction; // w/a/s/d
    public int weight;
    public int idSimpul;

// ctor
}
```

Kelas Block merupakan kelas yang merepresentasikan setiap blok yang berada di papan permainan. Kelas Block memiliki atribut: idx yang merupakan indeks blok yang sesuai dengan urutan dimasukkannya informasi blok tersebut, length yang merupakan panjang dari blok, x yang merupakan posisi x awal dari blok, y yang merupakan posisi y awal dari blok, dan jenis yang merupakan jenis dari blok yaitu vertical atau horizontal. Berikut prototype kelas Block.

```
public class Block {
    private int idx; // idx = 0 untuk block utama
    private int length;
    private int x;
    private int y;
    private char jenis; // vertikal (v) atau
horizontal (h)

// ctor
// getter dan setter
}
```

Kelas Simpul merupakan kelas yang merepresentasikan setiap simpul penyusun graf yang digunakan untuk menyelesaikan persoalan menggunakan algoritma BFS atau DFS. Kelas Simpul memiliki atribut: idx yang merupakan indeks dari simpul yang ditentukan dari urutan penghidupannya (urutan penghidupan dan urutan pengecekan dapat berbeda), nodes yang merupakan list dari indeks simpul-simpul mana saja yang telah dikunjungi dari awal sampai dengan simpul ini, dan ada isSolusi yang merupakan keterangan apakah simpul tersebut adalah solusi dari persoalan. Berikut prototype kelas Simpul.

```
public class Simpul {
    private int idx;
    private ArrayList<Integer> nodes;
    private boolean isSolusi;

//ctor
//getter dan setter
}
```

Kelas UnblockMeSolver adalah kelas utama dari program ini dimana pada kelas ini terdapat prosedur main yang dapat dijalankan. Kelas ini memiliki atribut map sebagai array of Block sebagai peta posisi awal dari semua blok pada papan permainan. Pada kelas ini juga terdapat dua buah atribut static yaitu nSimpul yang digunakan untuk menentukan indeks pada saat pembuatan simpul serta menghitung jumlah simpul yang telah dibuat (nSimpul dimulai dari 0, maka jumlah simpul-simpul sebenarnya adalah nSimpul+1), dan ada steps yang merupakan list of Step yang merupakan kompilasi dari semua step yang terjadi akibat pembuatan simpul. Berikut prototype kelas UnblockMeSolver.

```
public class UnblockMeSolver {
    private static int nSimpul = 0;
    private Block[] map;
    private static ArrayList<Step> steps = new
ArrayList<Step>();

//ctor
//getter dan setter

// mengecek apakah step bisa dihasilkan dari sebuah
block ke suatu arah dengan jumlah langkah tertentu
public int countStepAvail(ArrayList<Integer> nodes,
int idBlock, char direction){…}

//asumsi direction sesuai dengan jenis block dan
perpindahan tidak melampaui batas layar
public Simpul createSimpul(ArrayList<Integer> nodes,
int idBlock, char direction, int weight){…}

//prosedur main
}
```

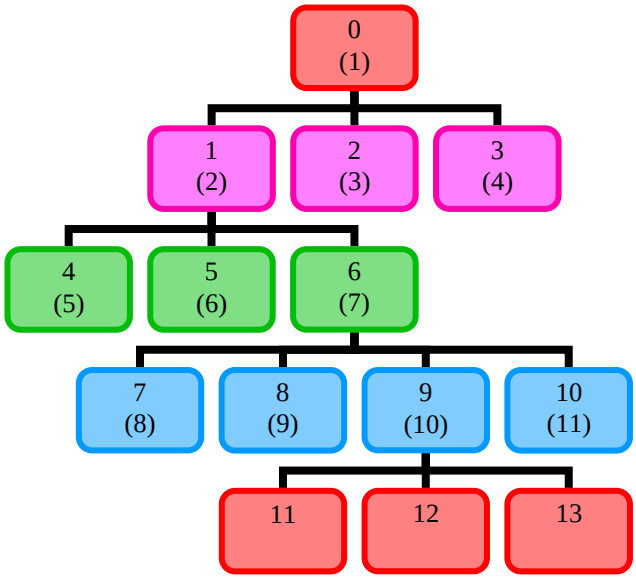
C. Cara Kerja Program

Sesuai dengan hasil yang didapat pada gambar 6 dan gambar 7 diatas, berikut akan dijelaskan cara kerja program mulai dari menerima input sampai mengeluarkan output seperti gambar-gambar hasil *screenshot* program tersebut.

Pertama-tama akan dijelaskan mengenai cara kerja program dengan algoritma BFS. Berikut langkah-langkah penyelesaian persoalan dengan algoritma BFS menggunakan program.

1. Membuat sebuah queue untuk menyimpan simpul-simpul.
2. Buat simpul dengan indeks ke-0 dimana semua blok pada papan belum bergerak dan list of Step masih kosong, lalu masukkan simpul ke queue dan mulai loop.
3. Cek apakah head dari queue adalah solusi, jika bukan, remove simpul dari queue dan jadikan sebagai simpul ekspan. Jika iya, maka loop selesai.
4. Ekspan simpul ekspan. Pengekspanan simpul dapat berakibat bertambahnya simpul-simpul baru pada tail queue.
5. Ulangi langkah 3 sampai ditemukan solusi.

Berikut ini adalah graf yang terbentuk dari simpul-simpul yang terbentuk dari penyelesaian persoalan pada gambar 5 dengan contoh output program seperti pada gambar 6.



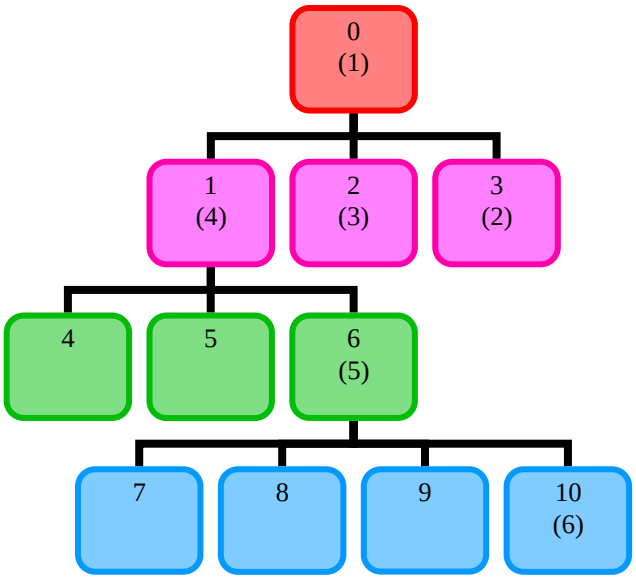
Gambar 8. Graf penyelesaian persoalan dengan BFS

Pada gambar diatas adalah graf yang terbentuk dari penyelesaian dengan algoritma BFS pada program. Angka pada setiap simpul adalah indeks dari simpul tersebut, sedangkan angka yang berada di dalam kurung adalah urutan pengecekan simpul tersebut. Dari graf tersebut terlihat bahwa pada akhir program masih tersisa simpul 11, 12, dan 13 pada queue yang tidak diperiksa karena simpul 10 sudah merupakan solusi.

Selanjutnya akan dijelaskan mengenai cara kerja program dengan algoritma DFS. Berikut langkah-langkah penyelesaian persoalan dengan algoritma DFS menggunakan program.

1. Membuat sebuah stack untuk menyimpan simpul-simpul.
2. Buat simpul dengan indeks ke-0 dimana semua blok pada papan belum bergerak dan list of Step masih kosong, lalu masukkan simpul ke stack dan mulai loop.
3. Cek apakah top dari stack adalah solusi, jika bukan, pop simpul dari stack dan jadikan sebagai simpul ekspan. Jika iya, maka loop selesai.
4. Ekspan simpul ekspan. Pengekspanan simpul dapat berakibat bertambahnya simpul-simpul baru pada top stack.
5. Ulangi langkah 3 sampai ditemukan solusi.

Berikut ini adalah graf yang terbentuk dari simpul-simpul yang terbentuk dari penyelesaian persoalan pada gambar 5 dengan contoh output program seperti pada gambar 7.



Gambar 9. Graf penyelesaian persoalan dengan DFS

Pada gambar diatas adalah graf yang terbentuk dari penyelesaian dengan algoritma DFS pada program. Seperti graf sebelumnya, angka pada setiap simpul melambangkan indeks dari simpul tersebut dan angka yang berada di dalam kurung melambangkan urutan pengecekannya. Di akhir program, di dalam stack masih tersisa simpul dengan indeks 4, 5, 7, 8, dan 9.

Pada program ini jumlah simpul yang dihidupkan dapat berbeda dengan jumlah simpul yang dicek. Hal ini terjadi karena pengekspanan simpul tidak dilakukan saat tepat sebelum simpul akan dicek, tapi saat simpul yang mengekspannya selesai dicek dan ternyata simpul tersebut bukan solusi, barulah dilakukan ekspan ke semua simpul yang mungkin.

IV. Kesimpulan

Dari hasil beberapa kali percobaan, rata-rata jumlah simpul yang dihidupkan dengan solusi DFS lebih sedikit daripada dengan menggunakan solusi BFS. Karena itu, rata-rata waktu yang dibutuhkan algoritma DFS juga lebih sedikit daripada algoritma BFS. Namun percobaan yang telah dilakukan masih terbatas dengan jumlah blok yang sedikit pada papan permainan. Hal ini disebabkan ketika jumlah blok semakin banyak dimasukkan, waktu yang dibutuhkan semakin lama, bahkan dapat mencapai lebih dari 1 jam. Untuk mendapatkan waktu penyelesaian yang lebih cepat masih harus dilakukan beberapa perubahan dalam program, sehingga program dapat digunakan untuk menyelesaikan persoalan sulit dalam waktu singkat.

V. Referensi

[1] Munir, Rinaldi. 2009. Diktat Strategi Algoritma. Bandung :Penerbit Institut Teknologi Bandung.

[2] Mu, Dimas. 2017. Algoritma BFS dan DFS. http://www.postingankeren.com/2017/01/algoritma-bfs-dan-dfs.html/ . Diakses pada tanggal 19 Mei 2017.

[3] Ttsiodras. 2012. Solving The "Unblock Me" Puzzle. https://www.thanassis.space/unblock.html/ . Diakses pada tanggal 16 Mei 2017.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Mei 2017

Muthmainnah

Lampiran – Source Code Program

File : Step.java

```
public class Step {
    public int idBlock;
    public char direction; // w/a/s/d
    public int weight;
    public int idSimpul;

    Step(int idBlock, char direction, int weight, int idSimpul) {
        this.idBlock = idBlock;
        this.direction = direction;
        this.weight = weight;
        this.idSimpul = idSimpul;
    }
}
```

File : Block.java

```
import java.util.ArrayList;

public class Step {
public class Block {
    private int idx; // idx = 0 untuk block utama
    private int length;
    private int x;
    private int y;
    private char jenis; // vertikal (v) atau horizontal (h)

    public Block(int idx, char jenis, int x, int y, int length) {
        this.idx = idx;
        this.length = length;
        this.x = x;
        this.y = y;
        this.jenis = jenis;
    }

    public int getIdx() {
        return idx;
    }

    public int getLength() {
        return length;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    public char getJenis() {
        return jenis;
    }
}
```

File : Simpul.java

```
import java.util.ArrayList;

public class Simpul {
    private int idx;
    private ArrayList<Integer> nodes;
```

```
    private boolean isSolusi;

    Simpul(int idx, ArrayList<Integer> nodes, boolean isSolusi) {
        this.idx = idx;
        this.nodes = nodes;
        this.isSolusi = isSolusi;
    }

    public int getIdx() {
        return idx;
    }

    public ArrayList<Integer> getNodes() {
        return nodes;
    }

    public void setNodes(ArrayList<Integer> nodes) {
        this.nodes = nodes;
    }

    public boolean getIsSolusi() {
        return isSolusi;
    }
}
```

File : UnblockMeSolver.java

```
import java.lang.reflect.Array;
import java.util.*;

public class UnblockMeSolver {
    private static int nSimpul = 0;
    private Block[] map;
    private static ArrayList<Step> steps = new ArrayList<Step>();

    public UnblockMeSolver(int nBlock) {
        Block[] map = new Block[nBlock];
    }

    public Block[] getMap() {
        return map;
    }

    public void setMap(Block[] map) {
        this.map = map;
    }

    // mengecek apakah step bisa dihasilkan dari sebuah block ke suatu arah dengan jumlah langkah tertentu
    public int countStepAvail(ArrayList<Integer> nodes, int idBlock, char direction) {
        // mengisi mapTemp
        Block[] mapTemp = new Block[map.length];
        for (int i = 0; i < map.length; i++) {
            mapTemp[i] = new Block(map[i].getIdx(), map[i].getJenis(), map[i].getX(), map[i].getY(),
map[i].getLength());
        }
        for (int i = 0; i < nodes.size(); i++) {
            int j = 0;
            boolean stop = false;
            while (!stop && j < steps.size()) {
                if (nodes.get(i) == steps.get(j).idSimpul) {
                    int id = steps.get(j).idBlock;
                    char dir = steps.get(j).direction;
                    int w = steps.get(j).weight;
                    if (dir == 'w') {
                        mapTemp[id].setY(mapTemp[id].getY() - w);
                    }
                    else if (dir == 's') {
                        mapTemp[id].setY(mapTemp[id].getY() + w);
                    }
                    else if (dir == 'a') {
                        mapTemp[id].setX(mapTemp[id].getX() - w);
                    }
```

```
                    else {
                        mapTemp[id].setX(mapTemp[id].getX() + w);
                    }
                }
                if (steps.get(j).idSimpul > nodes.get(i)) {
                    stop = true;
                }
                else {
                    j++;
                }
            }
        }
        int stepAvail = 0;
        boolean stop = false;
        int goalStep;
        int position;
        if (mapTemp[idBlock].getJenis() == 'h' && direction == 'd') {
            position = mapTemp[idBlock].getX() + mapTemp[idBlock].getLength() - 1;
            goalStep = 5 - position;
        }
        else if (mapTemp[idBlock].getJenis() == 'h' && direction == 'a') {
            goalStep = mapTemp[idBlock].getX();
            position = mapTemp[idBlock].getX();
        }
        else if (mapTemp[idBlock].getJenis() == 'v' && direction == 's') {
            position = mapTemp[idBlock].getY() + mapTemp[idBlock].getLength() - 1;
            goalStep = 5 - position;
        }
        else if (mapTemp[idBlock].getJenis() == 'v' && direction == 'w') {
            goalStep = mapTemp[idBlock].getY();
            position = mapTemp[idBlock].getY();
        }
        else {
            return 0;
        }
        while (!stop && stepAvail < goalStep && position >= 0 && position < 6) {
            if (direction == 'd' || direction == 's') {
                position++;
            }
            else {
                position--;
            }
            int i = 0;
            while (!stop && i < mapTemp.length) {
                // mengecek apa ada block yg menghalang pada position
                if (i != idBlock) {
                    if (mapTemp[idBlock].getJenis() == 'h') {
                        if (mapTemp[i].getX() == position && mapTemp[i].getJenis() == 'v' &&
mapTemp[i].getY() <= mapTemp[idBlock].getY() &&
                                mapTemp[i].getY()+mapTemp[i].getLength() > mapTemp[idBlock].getY()) {
                            stop = true;
                        }
                        else if (mapTemp[i].getY() == mapTemp[idBlock].getY() && mapTemp[i].getJenis() ==
'h' && mapTemp[i].getX() <=
                                position && mapTemp[i].getX()+mapTemp[i].getLength() > position) {
                            stop = true;
                        }
                    }
                    else if (mapTemp[idBlock].getJenis() == 'v') {
                        if (mapTemp[i].getY() == position && mapTemp[i].getJenis() == 'h' &&
mapTemp[i].getX() <= mapTemp[idBlock].getX() &&
                                mapTemp[i].getX()+mapTemp[i].getLength() > mapTemp[idBlock].getX()) {
                            stop = true;
                        }
                        else if (mapTemp[i].getX() == mapTemp[idBlock].getX() && mapTemp[i].getJenis() ==
'v' && mapTemp[i].getY() <=
                                position && mapTemp[i].getY()+mapTemp[i].getLength() > position) {
                            stop = true;
                        }
                    }
                }
                i++;
            }
```

```
            if (!stop) {
                stepAvail++;
            }
        }

        return stepAvail;
    }

    //asumsi direction sesuai dengan jenis block dan perpindahan tidak melampaui batas layar
    public Simpul createSimpul(ArrayList<Integer> nodes, int idBlock, char direction, int weight) {
        nSimpul++;
        // mengisi mapTemp
        Block[] mapTemp = new Block[map.length];
        for (int i = 0; i < map.length; i++) {
            mapTemp[i] = new Block(map[i].getIdx(), map[i].getJenis(), map[i].getX(), map[i].getY(),
map[i].getLength());
        }
        for (int i = 0; i < nodes.size(); i++) {
            int j = 0;
            boolean stop = false;
            while (!stop && j < steps.size()) {
                if (nodes.get(i) == steps.get(j).idSimpul) {
                    int id = steps.get(j).idBlock;
                    char dir = steps.get(j).direction;
                    int w = steps.get(j).weight;
                    if (dir == 'w') {
                        mapTemp[id].setY(mapTemp[id].getY() - w);
                    }
                    else if (dir == 's') {
                        mapTemp[id].setY(mapTemp[id].getY() + w);
                    }
                    else if (dir == 'a') {
                        mapTemp[id].setX(mapTemp[id].getX() - w);
                    }
                    else {
                        mapTemp[id].setX(mapTemp[id].getX() + w);
                    }
                }
                if (steps.get(j).idSimpul > nodes.get(i)) {
                    stop = true;
                }
                else {
                    j++;
                }
            }
        }
        //menggeser block uji
        if (direction == 'w') {
            mapTemp[idBlock].setY(mapTemp[idBlock].getY()-weight);
        }
        else if (direction == 's') {
            mapTemp[idBlock].setY(mapTemp[idBlock].getY()+weight);
        }
        else if (direction == 'a') {
            mapTemp[idBlock].setX(mapTemp[idBlock].getX()-weight);
        }
        else {
            mapTemp[idBlock].setX(mapTemp[idBlock].getX()+weight);
        }
        // menambah step
        Step currentStep = new Step(idBlock, direction, weight, nSimpul);
        steps.add(currentStep);
        boolean isSolusi;
        if (mapTemp[0].getX() >= 4) {
            isSolusi = true;
        }
        else {
            isSolusi = false;
        }
        // membuat simpul baru
        ArrayList<Integer> newNodes = new ArrayList<Integer>();
        newNodes.addAll(nodes);
        newNodes.add(nSimpul);
```

```
        Simpul newSimpul = new Simpul(nSimpul, newNodes, isSolusi);

        return newSimpul;
    }

    public static int getnSimpul() { return nSimpul; }

    public static void main(String[] args) {
        if (args.length != 1 || (!args[0].equals("bfs") && !args[0].equals("dfs"))) {
            System.out.println("Cara menggunakan program ini adalah dengan menambahkan argumen berupa bfs
atau dfs.");
        }
        else {
            Scanner sc = new Scanner(System.in);
            System.out.println("Jumlah block pada permainan (termasuk block utama) :");
            int nBlock = sc.nextInt();
            UnblockMeSolver main = new UnblockMeSolver(nBlock);
            System.out.println("Masukkan informasi setiap block: <jenis(v/h)> <x_awal> <y_awal> <panjang>");
            System.out.println("(informasi block pertama yg dimasukkan haruslah informasi block utama)");
            Scanner in = new Scanner(System.in);
            int x, y, panjang;
            char jenis;
            Block[] blocks = new Block[nBlock];
            for (int i = 0; i < nBlock; i++) {
                String[] input = in.nextLine().split(" ");
                jenis = input[0].charAt(0);
                x = Integer.parseInt(input[1]);
                y = Integer.parseInt(input[2]);
                panjang = Integer.parseInt(input[3]);
                blocks[i] = new Block(i,jenis,x,y,panjang);
            }
            main.setMap(blocks);
            if (args[0].equals("bfs")) {
                long startTime = System.currentTimeMillis();
                LinkedList<Simpul> queue = new LinkedList<Simpul>();
                ArrayList<Integer> nodes = new ArrayList<Integer>();
                nodes.add(0);
                Simpul simpulE = new Simpul(0, nodes, false);
                queue.add(simpulE);
                boolean stop = false;
                while (!queue.isEmpty() && !stop) {
                    // mengambil head queue sebagai simpul expand
                    simpulE = queue.remove();
                    // menambahkan simpul hidup yang mungkin ke dalam queue
                    for (int i = 0; i < main.getMap().length; i++) {
                        int prevSimpul = simpulE.getNodes().get(simpulE.getNodes().size()-1);
                        int iExc = 0;
                        int idx = steps.size()-1;
                        boolean stopwhile = false;
                        while (idx >= 0 && !stopwhile) {
                            if (steps.get(idx).idSimpul == prevSimpul) {
                                iExc = steps.get(idx).idBlock;
                                stopwhile = true;
                            }
                            else {
                                idx--;
                            }
                        }
                        if (i != iExc) {
                            if (main.getMap()[i].getJenis() == 'v') {
                                int stepUp = main.countStepAvail(simpulE.getNodes(), i, 'w');
                                int stepDown = main.countStepAvail(simpulE.getNodes(), i, 's');
                                if (stepUp > 0) {
                                    for (int j = 1; j <= stepUp; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'w', j);
                                        queue.add(simpulH);
                                    }
                                }
                                if (stepDown > 0) {
                                    for (int j = 1; j <= stepDown; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 's', j);
                                        queue.add(simpulH);
                                    }
```

```
                            }
                        }
                        else {
                            int stepRight = main.countStepAvail(simpulE.getNodes(), i, 'd');
                            int stepLeft = main.countStepAvail(simpulE.getNodes(), i, 'a');
                            if (stepRight > 0) {
                                for (int j = 1; j <= stepRight; j++) {
                                    Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'd', j);
                                    queue.add(simpulH);
                                }
                            }
                            if (stepLeft > 0) {
                                for (int j = 1; j <= stepLeft; j++) {
                                    Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'a', j);
                                    queue.add(simpulH);
                                }
                            }
                        }
                    }
                }
                if (queue.peek().getIsSolusi()) {
                    stop = true;
                }
            }
            long endTime = System.currentTimeMillis();
            long totalTime = endTime - startTime;
            if (queue.isEmpty()) {
                System.out.println("Solusi tidak ditemukan.");
            }
            else {
                System.out.println("Jumlah simpul yang dihidupkan (termasuk simpul 0) ada " +
(nSimpul+1) + " simpul.");
                Simpul solusi = queue.remove();
                System.out.println("Solusi BFS :");
                System.out.println("Simpul 0 - posisi awal");
                for (Integer i: solusi.getNodes()) {
                    int j = 0;
                    boolean stoploop = false;
                    while (!stoploop && j < steps.size()) {
                        if (i == steps.get(j).idSimpul) {
                            System.out.println("Simpul " + steps.get(j).idSimpul + " - block " +
steps.get(j).idBlock + " ke arah " + steps.get(j).direction + " sejauh " + steps.get(j).weight);
                        }
                        if (steps.get(j).idSimpul > i) {
                            stoploop = true;
                        }
                        else {
                            j++;
                        }
                    }
                }
                System.out.println("Waktu pencarian solusi : " + totalTime + " ms.");
            }
        }
        else if (args[0].equals("dfs")) {
            long startTime = System.currentTimeMillis();
            Stack<Simpul> stack = new Stack<Simpul>();
            ArrayList<Integer> nodes = new ArrayList<Integer>();
            nodes.add(0);
            Simpul simpulE = new Simpul(0, nodes, false);
            stack.push(simpulE);
            boolean stop = false;
            while (!stack.isEmpty() && !stop) {
                // mengambil head queue sebagai simpul expand
                simpulE = stack.pop();
                // menambahkan simpul hidup yang mungkin ke dalam queue
                for (int i = 0; i < main.getMap().length; i++) {
                    int prevSimpul = simpulE.getNodes().get(simpulE.getNodes().size()-1);
                    int iExc = 0;
                    int idx = steps.size()-1;
                    boolean stopwhile = false;
                    while (idx >= 0 && !stopwhile) {
                        if (steps.get(idx).idSimpul == prevSimpul) {
```

```
                                iExc = steps.get(idx).idBlock;
                                stopwhile = true;
                            }
                            else {
                                idx--;
                            }
                        }
                        if (i != iExc) {
                            if (main.getMap()[i].getJenis() == 'v') {
                                int stepUp = main.countStepAvail(simpulE.getNodes(), i, 'w');
                                int stepDown = main.countStepAvail(simpulE.getNodes(), i, 's');
                                if (stepUp > 0) {
                                    for (int j = 1; j <= stepUp; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'w', j);
                                        stack.push(simpulH);
                                    }
                                }
                                if (stepDown > 0) {
                                    for (int j = 1; j <= stepDown; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 's', j);
                                        stack.push(simpulH);
                                    }
                                }
                            }
                            else {
                                int stepRight = main.countStepAvail(simpulE.getNodes(), i, 'd');
                                int stepLeft = main.countStepAvail(simpulE.getNodes(), i, 'a');
                                if (stepRight > 0) {
                                    for (int j = 1; j <= stepRight; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'd', j);
                                        stack.push(simpulH);
                                    }
                                }
                                if (stepLeft > 0) {
                                    for (int j = 1; j <= stepLeft; j++) {
                                        Simpul simpulH = main.createSimpul(simpulE.getNodes(), i, 'a', j);
                                        stack.push(simpulH);
                                    }
                                }
                            }
                        }
                    }
                    if (stack.peek().getIsSolusi()) {
                        stop = true;
                    }
                }
                long endTime = System.currentTimeMillis();
                long totalTime = endTime - startTime;
                if (stack.isEmpty()) {
                    System.out.println("Solusi tidak ditemukan.");
                }
                else {
                    System.out.println("Jumlah simpul yang dihidupkan (termasuk simpul 0) ada " +
(nSimpul+1) + " simpul.");
                    Simpul solusi = stack.pop();
                    System.out.println("Solusi DFS :");
                    System.out.println("Simpul 0 - posisi awal");
                    for (Integer i: solusi.getNodes()) {
                        int j = 0;
                        boolean stoploop = false;
                        while (!stoploop && j < steps.size()) {
                            if (i == steps.get(j).idSimpul) {
                                System.out.println("Simpul " + steps.get(j).idSimpul + " - block " +
steps.get(j).idBlock + " ke arah " + steps.get(j).direction + " sejauh " + steps.get(j).weight);
                            }
                            if (steps.get(j).idSimpul > i) {
                                stoploop = true;
                            }
                            else {
                                j++;
                            }
                        }
                    }
```

```
                    System.out.println("Waktu pencarian solusi : " + totalTime + " ms.");
                }
            }
            else {
                System.out.println("Incorrect argument.");
            }
        }
    }
}
```