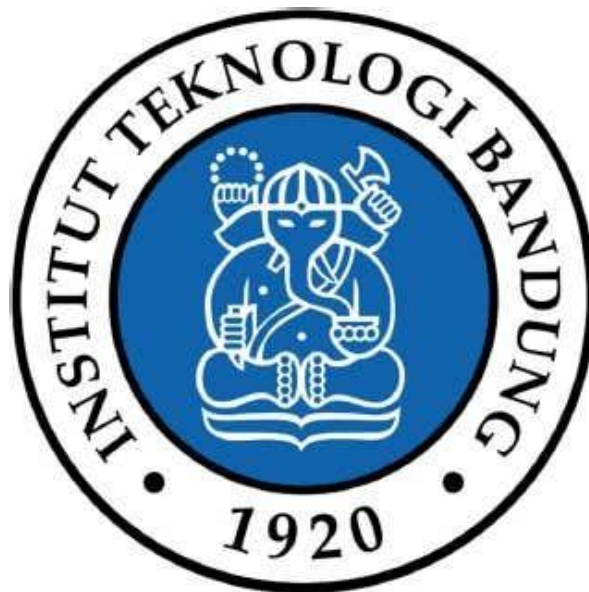


Laporan Tugas Kecil 3
IF2211 Strategi Algoritma

Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch and Bound*

Disusun oleh :

Ghazian Tsabit Alkamil
13520165



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
TAHUN AJARAN 2021/2022

BAB I

Algoritma *Branch and Bound*

A. Definisi Algoritma *Branch and Bound*

Algoritma branch and bound adalah algoritma yang biasa digunakan untuk menyelesaikan persoalan optimasi. Persoalan optimasi yang dimaksud adalah persoalan meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasan (*constraints*) persoalan. Secara singkat algoritma *branch and bound* adalah sebuah algoritma BFS (*breadth first search*) yang diberikan atau ditambah dengan *least cost search*.

Algoritma *branch and bound* memiliki beberapa karakteristik sebagai berikut :

1. Setiap simpul diberi sebuah nilai *cost*

$\hat{c}(i)$ = nilai taksiran lintasan termurah (*cost* terkecil) ke simpul status tujuan melalui simpul status i .

2. Berbeda dengan algoritma *breadth first search*, pada algoritma *branch and bound* simpul yang akan di expand berikutnya bukan berdasarkan urutan pembangkitannya, tetapi simpul yang memiliki *cost* yang paling kecil (*least cost search*) khususnya pada kasus minimasi.

B. Algoritma *Branch and Bound* vs Algoritma *Backtracking*

Persamaan algoritma *branch and bound* dengan algoritma *backtracking* :

1. Pencarian solusi dengan pembentukan pohon ruang status
2. ‘Membunuh’ simpul yang tidak ‘mengarah’ ke simpul solusi

Perbedaan algoritma *branch and bound* dengan algoritma *backtracking* :

1. ‘Nature’ persoalan yang bisa diselesaikan:
 - *Backtracking* : Tidak ada batasan (optimasi/non-optimasi), umumnya untuk persoalan non-optimasi.
 - *Branch and Bound* :
 - a. Persoalan optimasi
 - b. Untuk setiap simpul pada pohon ruang-status, diperlukan cara penentuan batas (*bound*) nilai terbaik fungsi objektif pada setiap simpul yang mungkin, dengan menambahkan komponen pada solusi sementara yang direpresentasikan oleh simpul
 - c. Nilai dari solusi terbaik sejauh ini

2. Pembangkitan simpul

- *Backtracking* : Umumnya DFS (*Depth First Search*)
- *Branch and Bound* : beberapa ‘aturan’ tertentu, yang paling umum adalah *best-first rule*

C. Fungsi Pembatas pada Algoritma Branch and Bound

Algoritma *branch and bound* juga menerapkan ‘pemangkasan’ pada jalur yang dianggap tidak lagi mengarah pada solusi. Terdapat beberapa kriteria pemangkasan pada algoritma *branch and bound* :

1. Nilai simpul yang tidak lebih baik dari nilai terbaik sejauh ini (*the best solution so far*)
2. Simpul tidak merepresentasikan solusi yang *feasible* karena ada batasan yang dilanggar
3. Solusi pada simpul tersebut hanya terdiri atas satu titik, tidak terdapat pilihan lain. Perlu dibandingkan nilai fungsi objektif dengan solusi terbaik saat ini, yang terbaik yang akan diambil

D. Implementasi Algoritma Branch and Bound pada Permainan 15-puzzle

Permainan 15 puzzle adalah permainan puzzle geser yang memiliki 15 ubin persegi bernomor 1–15 dalam bingkai dengan tinggi 4 ubin dan lebar 4 ubin, menyisakan satu posisi ubin kosong. Ubin di baris atau kolom yang sama dari posisi terbuka dapat dipindahkan dengan menggesernya secara horizontal atau vertikal. Tujuan dari puzzle ini adalah untuk menempatkan ubin dalam urutan numerik.

1	3	4	15
2		5	12
7	6	11	14
8	9	10	13

(a) Susunan awal

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

(b) Susunan akhir

State : berdasarkan ubin kosong

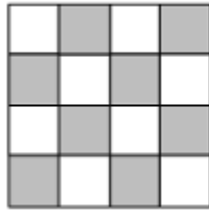
Aksi : Up, Down, Left, Right

Pada permainan 15 puzzle terdapat $16!$ (16 faktorial) kemungkinan susunan ubin yang berbeda, dan hanya setengah yang dapat dicapai dari state awal sembarang.

Teorema

Status tujuan hanya dapat dicapai dari status awal jika $\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap.

$X = 1$ jika ubin kosong pada posisi awal ada pada sel yang diarsir.

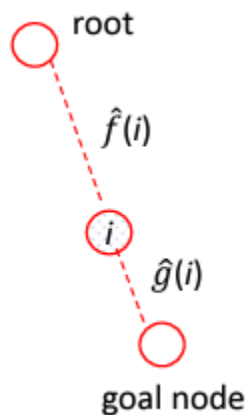


(c) Gambar sel yang diarsir

$\sum KURANG(i)$ = total banyaknya ubin bernomor j sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$. $POSISI(i)$ = posisi ubin bernomor i pada susunan yang diperiksa.

Cost dari Simpul Hidup

Pada umumnya, untuk kebanyakan persoalan letak simpul solusi (tujuan) tidak diketahui. Cost setiap simpul umumnya berupa taksiran.



(d) Ilustrasi Menghitung Cost

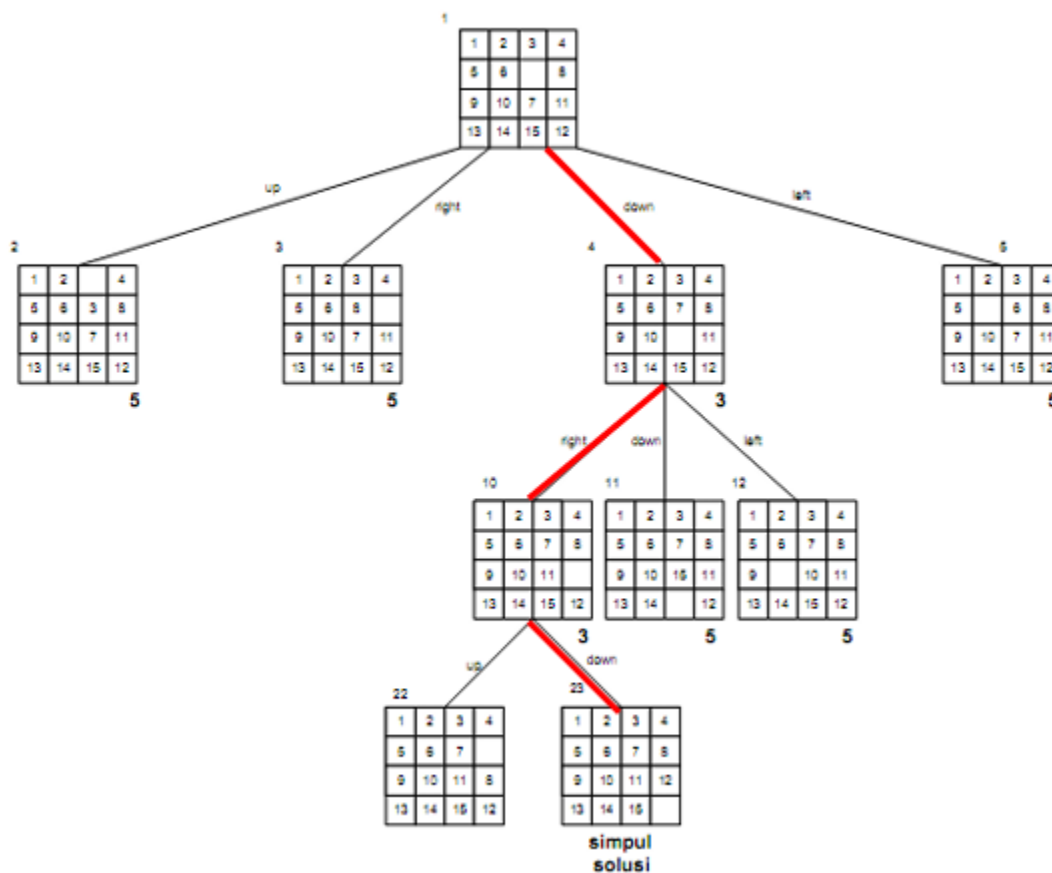
Cost simpul P pada 15-puzzle dirumuskan sebagai berikut :

$$\hat{c}(P) = f(P) + \hat{g}(P)$$

$f(P)$ = adalah panjang lintasan dari simpul akar ke P

$\hat{g}(P)$ = jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir

Pembentukan Pohon Ruang Status 15-puzzle dengan Algoritma *Branch and Bound*



(e) Pohon Ruang Status 15-puzzle

BAB II

Source Program

Program penyelesaian 15-puzzle dengan algoritma *branch and bound* diimplementasikan dengan menggunakan bahasa pemrograman Java. Pada implementasi dengan menggunakan bahasa Java dibuat beberapa kelas yang berguna untuk menyelesaikan persoalan 15-puzzle. Beberapa kelas yang dibuat pada implementasi kali ini adalah sebagai berikut :

1. Puzzle

Pada program ini kelas puzzle terdapat pada file Puzzle.java, kelas puzzle memiliki beberapa atribut yang dapat dilihat pada gambar 3.1.

```
public enum DIRECTION {UP, DOWN, LEFT, RIGHT}
private final int [][] puzzle;
private int [][] correctPuzzle;
private int cost;
private int blankX, blankY;
```

Gambar 3.1. Atribut Kelas Puzzle

Atribut *puzzle* adalah array dua dimensi yang bertipe integer yang akan berisi nilai puzzle yang akan dipecahkan. Atribut *Direction* adalah atribut yang bertipe kelas *enum*, yang berisi pergerakan dari ubin kosong, yaitu *up* (atas), *down* (bawah), *left* (kiri), dan *right* (kanan). Atribut *correctPuzzle* adalah array dua dimensi yang bertipe integer yang akan berisi nilai puzzle yang sudah terurut secara numerik yang memiliki ukuran yang sama dengan atribut *puzzle*. *Cost* adalah atribut bertipe integer yang nilai nya dihitung berdasarkan jumlah ubin tidak kosong pada *puzzle* yang posisinya tidak berada pada posisi akhir ditambah dengan jarak simpul puzzle tersebut ke simpul puzzle akar. Atribut *blankX* dan *blankY* adalah atribut bertipe integer yang nilai nya berisi indeks posisi dari ubin kosong pada *puzzle*.

Kelas *puzzle* memiliki beberapa *method* sebagai berikut :

a. *public Puzzle()*

Method ini adalah *default constructor* untuk kelas *puzzle* yang akan menginisialisasi atribut *puzzle* dengan sebuah array dua dimensi berukuran 4 × 4.

b. `public Puzzle(int [][] puzzle, int [][] correctPuzzle)`

Method ini adalah *user-defined constructor* untuk kelas *puzzle* yang memiliki argumen array dua dimensi bertipe integer bernama *puzzle* dan array dua dimensi bertipe integer bernama *correctPuzzle*. Kemudian program ini akan mengisi nilai atribut *puzzle* dan *correctPuzzle* sesuai masukan dari pengguna. Selain itu, pada *method* ini juga akan diisi nilai atribut *cost*, *blankX*, dan *blankY* sesuai dengan kondisi atribut *puzzle* pada saat itu.

c. `public Puzzle(Puzzle newPuzzle)`

Method ini adalah *user-defined constructor* untuk kelas *puzzle* yang memiliki argumen objek *puzzle* bernama *newPuzzle*. Kemudian program ini akan mengisi nilai atribut *puzzle* dan *correctPuzzle* sesuai dengan nilai yang ada pada objek *newPuzzle*. Selain itu, pada *method* ini juga akan diisi nilai atribut *cost*, *blankX*, dan *blankY* sesuai dengan nilai atribut terkait pada objek *newPuzzle*.

d. `public void printPuzzle()`

Method ini berfungsi untuk menampilkan atribut *puzzle* pada kelas *puzzle* ke layar. Contoh tampilan *puzzle* di layar dapat dilihat pada gambar 3.2.

[1]	[2]	[3]	[4]
[5]	[6]	[]	[8]
[9]	[10]	[7]	[11]
[13]	[14]	[15]	[12]

Gambar 3.2 Tampilan Puzzle di layar

e. `public void setTile(int x, int y, int tile)`

Method ini berfungsi sebagai *setter* yang akan mengisi nilai dari atribut *puzzle* pada baris ke-*x* dan kolom ke-*y* dengan nilai *tile*.

```
public void setTile(int x, int y, int tile){  
    this.puzzle[x][y] = tile;  
}
```

Gambar 3.3 setTile()

f. `public int getTile(int x, int y)`

Method ini berfungsi sebagai *getter* yang akan mengembalikan nilai dari atribut *puzzle* yang terletak pada baris ke-*x* dan kolom ke-*y*.

```
public int getTile(int x, int y){  
    return this.puzzle[x][y];  
}
```

Gambar 3.3 getTile()

g. `public void swap(int x1, int y1, int x2, int y2)`

Method ini berfungsi untuk menukar nilai dari atribut *puzzle* yang berada pada baris ke-*x1* dan kolom ke-*y1* dengan nilai dari atribut *puzzle* yang berada pada baris ke-*x2* dan kolom ke-*y2*, begitupun sebaliknya.

```
public void swap(int x1, int y1, int x2, int y2){  
  
    int previous = getTile(x1, y1);  
  
    setTile(x1, y1, getTile(x2, y2));  
  
    setTile(x2, y2, previous);  
    blankY = y2;  
    blankX = x2;  
}
```

Gambar 3.4 swap()

h. `public void move(DIRECTION direction)`

Method ini berfungsi untuk memindahkan posisi ubin kosong sesuai dengan arah yang tersedia, yaitu *up* (atas), *down* (bawah), *right* (kanan), dan *left* (kiri). Setelah itu, *method* ini akan memperbaharui nilai dari atribut *cost* sesuai dengan posisi *puzzle* yang baru.


```

public void move(DIRECTION direction) {
    switch (direction) {
        case UP:
            swap(blankX, blankY, (blankX - 1), blankY);

            break;
        case DOWN:
            swap(blankX, blankY, (blankX + 1), blankY);

            break;
        case LEFT:
            swap(blankX, blankY, blankX, (blankY - 1));

            break;
        case RIGHT:
            swap(blankX, blankY, blankX, (blankY + 1));

            break;
    }
    this.set_cost(hitung_cost(this.puzzle));
}

```

Gambar 3.5 move()

i. *public boolean canMove(DIRECTION direction)*

Method ini berfungsi untuk memeriksa apakah ubin kosong pada *puzzle* memungkinkan untuk melakukan pergerakan berdasarkan posisinya pada saat itu. Misal apabila ubin kosong berada pada baris pertama maka ubin kosong tersebut tidak dapat melakukan perpindahan ke atas dan lain sebagainya.

```

public boolean canMove(DIRECTION direction) {
    switch (direction) {
        case UP:
            if (blankX != 0) {
                return true;
            }
            break;
        case DOWN:
            if (blankX != puzzle.length - 1) {
                return true;
            }
            break;
        case LEFT:
            if (blankY != 0) {
                return true;
            }
            break;
        case RIGHT:
            if (blankY != puzzle[blankX].length - 1) {
                return true;
            }
            break;
    }
    return false;
}

```

Gambar 3.6 canMove()

j. *public boolean isSolved()*

Method ini berfungsi memeriksa apakah kondisi *puzzle* pada saat itu sudah sama dengan kondisi *correctPuzzle*, apabila kondisinya sudah sama maka *puzzle* tersebut telah dipecahkan atau sudah selesai.

```

public boolean isSolved() {
    for (int y = 0; y < puzzle.length; ++y) {
        for (int x = 0; x < puzzle[y].length; ++x) {
            if (puzzle[y][x] != correctPuzzle[y][x]) {
                return false;
            }
        }
    }
    return true;
}

```

Gambar 3.7 isSolved()

k. `public int kurangI()`

Method ini berfungsi menghitung nilai dari $\sum_{i=1}^{16} KURANG(i) + X$ tiap ubin

berdasarkan kondisi awal *puzzle* pada saat pertama kali di *construct*. Nilai yang dikembalikan dari fungsi ini nantinya digunakan untuk menentukan apakah suatu *puzzle* dapat dipecahkan atau tidak.

```
public int kurangI(){
    int kurangi = 0;
    int [] content = new int[16];
    int k = 0;

    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            content[k] = this.puzzle[i][j];
            k++;
        }
    }

    for(int i = 0; i < content.length; i++){
        for(int j = i; j < content.length; j++){
            if(content[j] < content[i]){
                kurangi++;
            }
        }
    }

    int X = 0;

    if(content[1] == 16 || content[3] == 16 || content[4] == 16 || content[6] == 16
    || content[9] == 16 || content[11] == 16 || content[12] == 16 || content[14] == 16){
        X = 1;
    }

    return kurangi + X;
}
```

Gambar 3.8 kurangI()

Algoritma dari method *kurangI()* ini mengikuti teorema yang sebelumnya sudah dibahas pada bab satu.

l. `public void kurang()`

Method ini berfungsi untuk menampilkan nilai dari *KURANG(i)* tiap ubin pada kondisi awal *puzzle* di *construct*.

i (ubin)	kurang(i)
1	0
2	0
3	0
4	0
5	0
6	0
16	9
8	1
9	1
10	1
7	0
11	0
13	1
14	1
15	1
12	0

Gambar 3.9 Tampilan Nilai KURANG(i) tiap ubin

```
public void kurang(){
    int kurangi = 0;
    int [] content = new int[16];
    int k = 0;
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            content[k] = this.puzzle[i][j];
            k++;
        }
    }
    System.out.println("i (ubin) " + " kurang(i)");
    for(int i = 0; i < content.length; i++){
        for(int j = i; j < content.length ; j++){
            if(content[j] < content[i]){
                kurangi++;
            }
        }
        if(content[i] < 10){
            System.out.println(" " + content[i] + " " + kurangi);
            kurangi = 0;
        }else{
            System.out.println(content[i] + " " + kurangi);
            kurangi = 0;
        }
    }
}
```

Gambar 3.10 Algoritma kurang()

m. *public boolean isReachable()*

Method ini berfungsi untuk memeriksa apakah sebuah *puzzle* dapat diselesaikan berdasarkan nilai yang dikembalikan oleh *method kurangI()*. Apabila nilai yang dikembalikan oleh *method kurangI()* bernilai genap maka *puzzle* dapat diselesaikan dan apabila nilai yang dikembalikan oleh *method kurangI()* bernilai ganjil maka *puzzle* tidak dapat diselesaikan.

```
public Boolean isReachable(){
    if(this.kurangI() % 2 == 0){
        return true;
    } else {
        return false;
    }
}
```

Gambar 3.11 isReachable()

n. *public int hitung_cost(int [][] puzzle)* dan *public int hitung_cost(Puzzle puzzle)*

Method ini berfungsi untuk mengembalikan nilai dari *cost* pada sebuah *puzzle*, nilai *cost* dihitung dengan menjumlahkan total ubin tidak kosong yang tidak berada pada posisi akhir.

```
public int hitung_cost(int [][] puzzle){
    int gP = 0;
    int [][] final_state = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16}};
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            if(puzzle[i][j] != 16){
                if(puzzle[i][j] != final_state[i][j]){
                    gP++;
                }
            }
        }
    }
    return gP;
}
```

Gambar 3.12 hitung_cost()

o. *public int get_cost()*

Method ini berfungsi untuk mengembalikan nilai dari atribut *cost* yang dimiliki oleh *puzzle*.

```
public int get_cost(){  
    return this.cost;  
}
```

Gambar 3.13 get_cost()

p. *public void add_cost(int c)*

Method ini berfungsi untuk menambahkan nilai dari atribut *cost* dengan *c*.

```
public void add_cost(int c){  
    this.cost = this.cost + c;  
}
```

Gambar 3.14 add_cost()

q. *public void set_cost(int _cost)*

Method ini berfungsi sebagai *setter* yang mengisi nilai dari atribut *cost* dengan *_cost*.

```
public void set_cost(int _cost){  
    this.cost = _cost;  
}
```

Gambar 3.15 set_cost()

r. *public void get_x_blank()* dan *public void get_y_blank()*

Method ini berfungsi untuk mengembalikan posisi dari ubin kosong pada *puzzle*.

```
public void get_x_blank(){  
  
    int x = -1;  
  
    for(int i = 0; i < 4; i++){  
        for(int j = 0; j < 4; j++){  
            if(this.puzzle[i][j] == 16){  
                x = i;  
            }  
        }  
    }  
    this.blankX = x;  
}
```

Gambar 3.16.1 *get_x_blank()*

```
public void get_y_blank(){  
  
    int y = -1;  
  
    for(int i = 0; i < 4; i++){  
        for(int j = 0; j < 4; j++){  
            if(this.puzzle[i][j] == 16){  
                y = j;  
            }  
        }  
    }  
    this.blankY = y;  
}
```

Gambar 3.16.2 *get_y_blank()*

2. PuzzleComparator

Pada program ini kelas PuzzleComparator berada di PuzzleComparator.java, kelas ini berfungsi untuk membandingkan dua buah *puzzle* berdasarkan nilai *cost* nya. Kelas ini juga akan dipakai pada kelas BnBSolver ketika akan menambahkan simpul pada Priority Queue, yang akan diutamakan adalah simpul atau *puzzle* yang memiliki *cost* yang lebih kecil.

```
public class PuzzleComparator implements Comparator<Puzzle>{

    public int compare(Puzzle p1, Puzzle p2) {
        if(p1.get_cost() < p2.get_cost()){
            return -1;
        } else if(p1.get_cost() > p2.get_cost()){
            return 1;
        } else {
            return 0;
        }
    }
}
```

Gambar 3.17 kelas PuzzleComparator

3. PuzzleLoader

Pada program ini kelas PuzzleLoader terdapat di file PuzzleLoader.java, kelas ini berfungsi untuk membaca dan menginput *puzzle* dari file text.


```

public class PuzzleLoader {

    public int[][] load(String pathFile) throws FileNotFoundException{

        int [][] puzzle;

        puzzle = new int[4][4];

        Scanner sc = new Scanner(new BufferedReader(new FileReader(pathFile)));

        while(sc.hasNextLine()) {
            for (int i=0; i<puzzle.length; i++) {
                String[] line = sc.nextLine().trim().split(" ");
                for (int j=0; j<line.length; j++) {
                    puzzle[i][j] = Integer.parseInt(line[j]);
                }
            }
        }

        return puzzle;
    }
}

```

Gambar 3.18 kelas PuzzleLoader

4. BnBSolver

Pada program ini kelas BnBSolver terdapat di file BnBSolver.java, kelas ini berfungsi untuk melakukan penyelesaian persoalan 15-puzzle dengan menggunakan algoritma *branch and bound*.

Kelas BnBSolver memiliki sebuah atribut bertipe Priority Queue yang bernama *frontiers*. Priority Queue yang berada pada kelas BnBSolver memiliki elemen yang bertipe *puzzle*. Selain itu, prioritas yang digunakan pada Priority Queue ini adalah *puzzle* yang memiliki *cost* yang terkecil.

```

private final PriorityQueue<Puzzle> frontiers =
    new PriorityQueue<Puzzle>(new PuzzleComparator());

```

Gambar 3.19 atribut frontiers

Kelas BnBSolver hanya memiliki satu *method* yang bernama *solve*. Method ini digunakan untuk mencari solusi penyelesaian dari 15-puzzle.

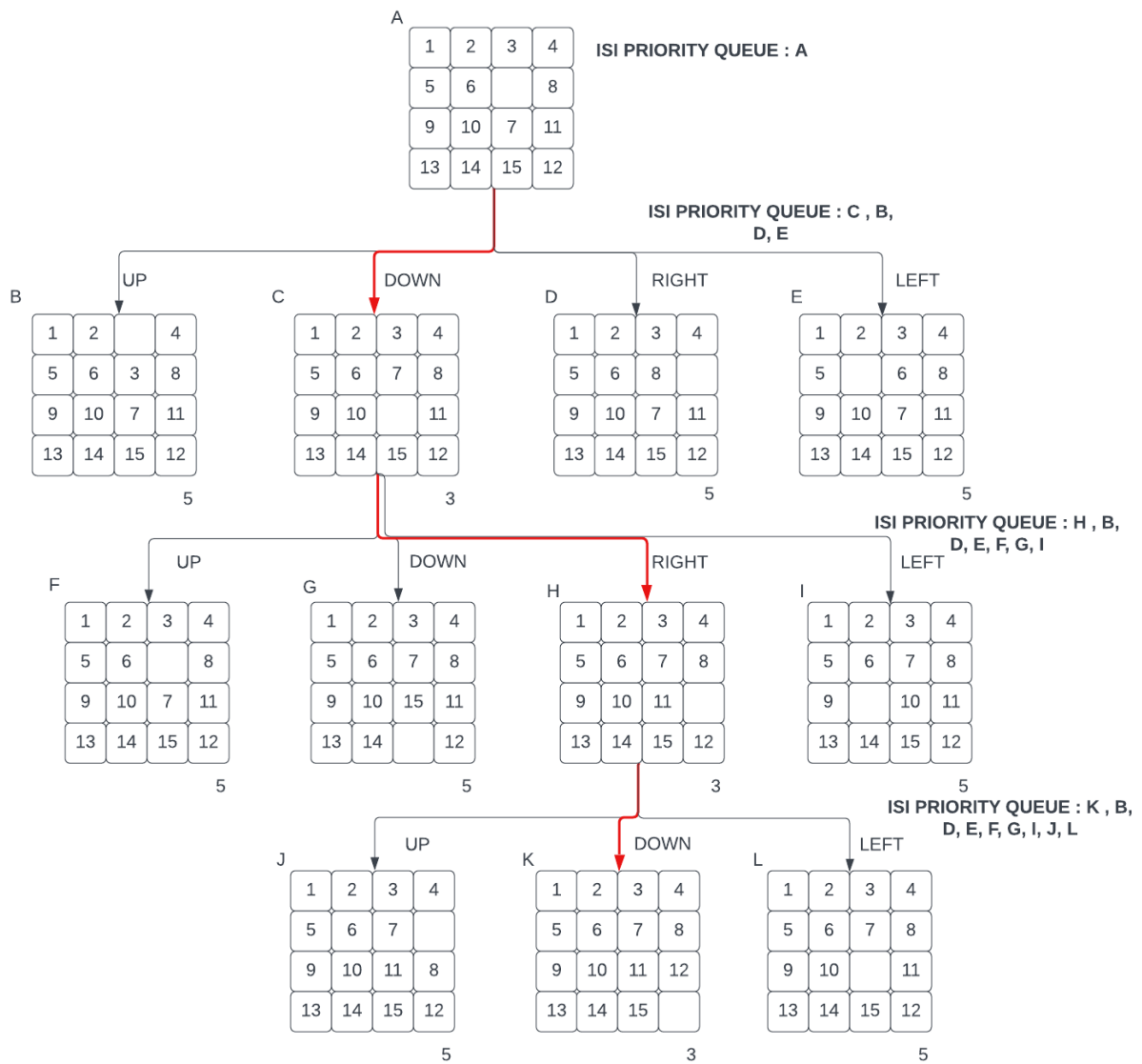
```
public Puzzle solve(Puzzle puzzleToSolve, Puzzle.DIRECTION[] gerakan){
    frontiers.add(puzzleToSolve);
    int r_to_root = 0;
    int size = 1;
    while(!frontiers.isEmpty()){
        Puzzle puzzle = frontiers.poll();
        r_to_root++;
        puzzle.printPuzzle();
        System.out.println("Cost simpul saat ini : " + puzzle.get_cost());
        System.out.println();
        if(puzzle.isSolved()){
            System.out.println("Jumlah simpul yang dibangkitkan : " + size);
            return puzzle;
        }
        for(int i = 0; i < gerakan.length; i++){
            if(puzzle.canMove(gerakan[i])) {
                Puzzle newPuzzle = new Puzzle(puzzle);
                newPuzzle.move(gerakan[i]);
                newPuzzle.add_cost(r_to_root);
                frontiers.add(newPuzzle);
                size++;
            }
        }
    }
    return null;
}
```

Gambar 3.20 solve()

Langkah-langkah penyelesaian dari 15 puzzle dengan algoritma *branch and bound* adalah sebagai berikut :

1. Tambahkan *puzzle* yang ingin diselesaikan sebagai elemen pertama dari priority queue, *puzzle* pertama ini juga akan menjadi root dari pohon ruang status persoalan 15 puzzle.
2. Keluarkan *puzzle* yang telah ditambahkan tersebut, kemudian tampilkan *puzzle* tersebut ke layar menggunakan method *printPuzzle()*.
3. Cek apakah *puzzle* merupakan *puzzle* sudah terselesaikan, apabila iya maka kembalikan *puzzle* tersebut, jika tidak maka lanjut ke langkah berikutnya.
4. Bangkitkan simpul baru yaitu simpul *puzzle* yang telah dilakukan perpindahan ubin kosong sesuai arah yang memungkinkan pada *puzzle* tersebut. Selain itu, tambahkan *cost* dari *puzzle* tersebut sebesar jarak simpul tersebut ke simpul akar.
5. Lakukan langkah-langkah diatas sampai ditemukan sebuah *puzzle* yang merupakan solusi atau sampai priority queue tidak memiliki elemen lagi (kosong).

Contoh ilustrasi pencarian solusi dengan menggunakan *method solve* :



Gambar 3.21 pohon ruang status persoalan 15 puzzle

Ketika simpul merupakan simpul solusi, maka simpul tersebut akan dikembalikan dan simpul lain akan dihapus atau “dibunuh”.

5. Main

Kelas main berfungsi untuk mengkombinasikan semua kelas yang sebelumnya sudah dibuat untuk menyelesaikan persoalan 15 puzzle dengan algoritma *branch and bound*. Pada kelas main dibuat sebuah fungsi statik tambahan yaitu fungsi *buatCorrectPuzzle()* dan fungsi *buatRandomPuzzle()*.

Fungsi *buatCorrectPuzzle()* berfungsi untuk membuat sebuah array dua dimensi yang memiliki ukuran yang sama dengan *puzzle* yang ingin diselesaikan dan elemen-elemennya sudah terurut secara numerik.

```
private static int [][] buatCorrectPuzzle(int xsize, int ysize){  
    int [][] correctPuzzle = new int [xsize][ysize];  
    int counter = 1;  
    for(int i = 0; i < xsize; i++){  
        for(int j = 0; j < ysize; j++){  
            correctPuzzle[i][j] = counter;  
            counter++;  
        }  
    }  
    return correctPuzzle;  
}
```

Gambar 3.22 fungsi buatCorrectPuzzle()

Fungsi *buatRandomPuzzle()* berfungsi untuk menghasilkan *puzzle* yang elemen-elemennya akan diacak atau di-*random* urutannya, yang akan digunakan ketika pengguna memilih mode input *puzzle* dengan *random puzzle*.

```

private static int [][] buatRandomPuzzle(int xsize, int ysize){

    int [][] randomPuzzle = new int [xsize][ysize];
    Random rand = new Random();
    List<Integer> list = new ArrayList<>();
    for(int i = 1; i <= 16; ++i){
        list.add(i);
    }
    Collections.shuffle(list, rand);
    int counter = 0;
    for(int i = 0; i < xsize; i++){
        for(int j = 0; j < ysize; j++){
            randomPuzzle[i][j] = list.get(counter);
            counter++;
        }
    }
    return randomPuzzle;
}

```

Gambar 3.23 fungsi buatRandomPuzzle()

Langkah-langkah program penyelesaian persoalan 15 puzzle pada kelas main:

1. Program menampilkan banner
2. Program meminta input mode kepada pengguna, input *puzzle* dari file text atau gunakan *random puzzle*.
3. Program menampilkan puzzle ke layar. Kemudian program menampilkan nilai dari $KURANG(i)$ tiap ubin tidak kosong pada *puzzle*. Kemudian program menampilkan nilai dari $\sum_{i=1}^{16} KURANG(i) + X$ *puzzle* tersebut.
4. Apabila *puzzle* tidak dapat diselesaikan maka program akan mengeluarkan pesan “Puzzle tidak dapat dipecahkan !!!”
5. Apabila *puzzle* dapat dipecahkan maka program akan menampilkan pergerakan ubin tidak kosong pada *puzzle*. Setelah proses pencarian solusi telah selesai, maka program akan menampilkan waktu eksekusi program dan jumlah simpul yang dibangkitkan pada di dalam pohon ruang status pencarian

BAB III

Hasil Program

1. Data Uji test_01.txt

Input :

```
1 2 3 4
5 6 16 8
9 10 7 11
13 14 15 12
```

Gambar 4.1.1 test_01.txt

```
#####
#
#
#  2  1  3  4  5  6  7  8  9  10  11  12  13  14  15  16
#  5  6  16  8
#  9  10  7  11
#  13  14  15  12
#
#
#  oleh : Ghazian Tsabit Alkamil / 13520165
#
#####

Selamat datang di 15 puzzle solver
Terdapat dua mode input puzzle :
1. Input puzzle dari file
2. Gunakan random puzzle

Input mode pilihan Anda (1/2): 1
Mode input puzzle dari file dipilih !!!
Input filename: test_01.txt]
```

Gambar 4.1.2 input file test_01.txt

Output :

```

----- Puzzle To Solve -----

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [  ] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]

i (ubin)  kurang(i)
1         0
2         0
3         0
4         0
5         0
6         0
16        9
8         1
9         1
10        1
7         0
11        0
13        1
14        1
15        1
12        0
Nilai dari KURANG(i) + X : 16

```

Gambar 4.1.3 output test_01.txt

```

----- Puzzle Movement -----

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [  ] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 3

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [  ] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 3

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [11] [  ]
[13] [14] [15] [12]
Cost simpul saat ini : 3

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [11] [12]
[13] [14] [15] [  ]
Cost simpul saat ini : 3

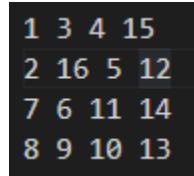
Jumlah simpul yang dibangkitkan : 12
Waktu eksekusi program : 11 ms

```

Gambar 4.1.4 output test_01

2. Data uji test_02.txt

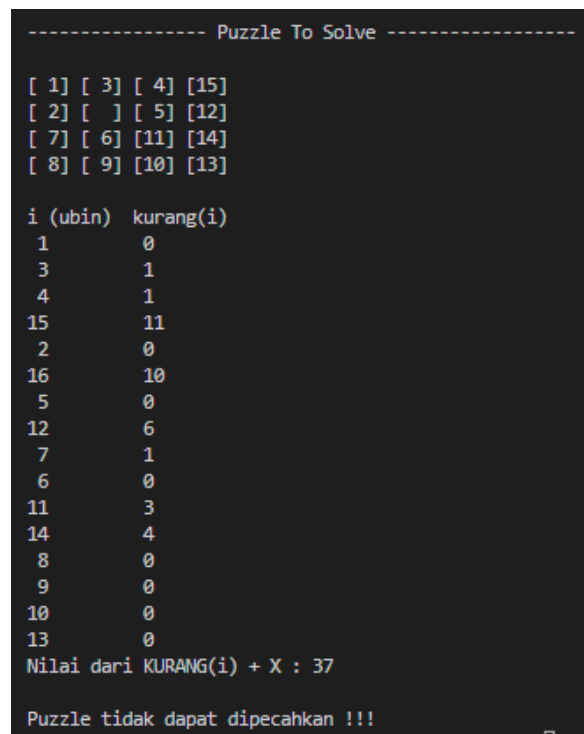
Input :



1	3	4	15
2	16	5	12
7	6	11	14
8	9	10	13

Gambar 4.2.1 test_02.txt

Output :



```
----- Puzzle To Solve -----  
  
[ 1] [ 3] [ 4] [15]  
[ 2] [  ] [ 5] [12]  
[ 7] [ 6] [11] [14]  
[ 8] [ 9] [10] [13]  
  
i (ubin)  kurang(i)  
1         0  
3         1  
4         1  
15        11  
2         0  
16        10  
5         0  
12         6  
7         1  
6         0  
11         3  
14         4  
8         0  
9         0  
10         0  
13         0  
Nilai dari KURANG(i) + X : 37  
  
Puzzle tidak dapat dipecahkan !!!
```

Gambar 4.2.2 output test_02.txt

3. Data uji test_03.txt

Input :

```
3 9 1 15
14 11 4 6
13 16 10 12
2 7 8 5
```

Gambar 4.3.1 test_03.txt

Output :

```
----- Puzzle To Solve -----

[ 3] [ 9] [ 1] [15]
[14] [11] [ 4] [ 6]
[13] [ ] [10] [12]
[ 2] [ 7] [ 8] [ 5]

i (ubin)  kurang(i)
3         2
9         7
1         0
15        11
14        10
11         7
4         1
6         2
13         6
16         6
10         4
12         4
2         0
7         1
8         1
5         0
Nilai dari KURANG(i) + X : 63

Puzzle tidak dapat dipecahkan !!!
```

Gambar 4.3.2 output test_03.txt

4. Data uji test_04.txt

Input :

```
1 2 4 16
5 6 3 8
9 10 7 11
13 14 15 12
```

Gambar 4.4.1 test_04.txt

Output :

```
----- Puzzle To Solve -----

[ 1] [ 2] [ 4] [  ]
[ 5] [ 6] [ 3] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]

i (ubin)  kurang(i)
1          0
2          0
4          1
16         12
5          1
6          1
3          0
8          1
9          1
10         1
7          0
11         0
13         1
14         1
15         1
12         0
Nilai dari KURANG(i) + X : 22
```

Gambar 4.4.2 output test_04.txt

```

----- Puzzle Movement -----

[ 1] [ 2] [ 4] [  ]
[ 5] [ 6] [ 3] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 5

[ 1] [ 2] [  ] [ 4]
[ 5] [ 6] [ 3] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 5

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [  ] [ 8]
[ 9] [10] [ 7] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 5

```

```

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [  ] [11]
[13] [14] [15] [12]
Cost simpul saat ini : 5

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [11] [  ]
[13] [14] [15] [12]
Cost simpul saat ini : 5

[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [11] [12]
[13] [14] [15] [  ]
Cost simpul saat ini : 5

Jumlah simpul yang dibangkitkan : 17
Waktu eksekusi program : 11 ms

```

Gambar 4.4.3 output test_04.txt

5. Data uji test_05.txt

Input :

```
5 1 3 4
2 16 7 8
9 6 10 12
13 14 11 15
```

Gambar 4.5.1 test_05.txt

Output :

```
----- Puzzle To Solve -----
[ 5] [ 1] [ 3] [ 4]
[ 2] [  ] [ 7] [ 8]
[ 9] [ 6] [10] [12]
[13] [14] [11] [15]

i (ubin)  kurang(i)
5          4
1          0
3          1
4          1
2          0
16         10
7          1
8          1
9          1
6          0
10         0
12         1
13         1
14         1
11         0
15         0
Nilai dari KURANG(i) + X : 22
```

Gambar 4.5.2 output test_05.txt

```
[ 1] [ 2] [ 3] [ 4]
[ 5] [ 6] [ 7] [ 8]
[ 9] [10] [11] [12]
[13] [14] [15] [  ]
Cost simpul saat ini : 7151

Jumlah simpul yang dibangkitkan : 74834
Waktu eksekusi program : 36107 ms
```

Gambar 4.5.3 output test_05.txt

6. Data uji test random_01

Input :

[illegible]

Gambar 4.6.1 input data uji random_01

Output :

```

input mode pilihan Anda (1/2): 2
Mode input random puzzle dipilih !!!
----- Puzzle To Solve -----

[15] [ 2] [ 9] [ 8]
[ 3] [ 5] [12] [11]
[ 4] [10] [14] [ 7]
[  ] [ 1] [13] [ 6]

i (ubin)   kurang(i)
15          14
2           1
9           7
8           6
3           1
5           2
12          6
11          5
4           1
10          3
14          4
7           2
16          3
1           0
13          1
6           0

Nilai dari KURANG(i) + X : 57
Puzzle tidak dapat dipecahkan !!!

```

Gambar 4.6.2 output data uji random 01

BAB IV

Kesimpulan

Kesimpulan yang didapat dari implementasi program penyelesaian persoalan 15-puzzle dengan algoritma *branch and bound* adalah algoritma *branch and bound* dapat dimanfaatkan dengan baik untuk penyelesaian persoalan 15-puzzle untuk kelima data uji yang tersedia.

Source code program implementasi program penyelesaian persoalan 15-puzzle dengan algoritma *branch and bound* : https://github.com/ZianTsabit/Tucil3_13520165

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil <i>running</i>	✓	
3. Program dapat menerima input dan menuliskan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	
5. Bonus dibuat		✓

DAFTAR PUSTAKA

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf>