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

Muhammad Alif Putra Yasa 13520135

Detail Algoritma *Branch and Bound*

Program terdiri dari 5 kelas:

1. *GUI*
2. *Parser*
3. *PuzzleGUI*
4. *PuzzleSolver*
5. *PrioQueue*

Kelas yang bertanggung jawab dalam menyelesaikan *puzzle* menggunakan algoritma *Branch and Bound* adalah kelas *PuzzleSolver* dan *PrioQueue*.

Kelas *PuzzleSolver* berisi algoritma yang akan digunakan untuk menyelesaikan *puzzle*.

Kelas *prioQueue* digunakan untuk menyimpan pohon dalam pencarian solusi *puzzle*. Atribut yang penting di kelas ini antara lain adalah atribut `prioQueue` dan `stepTree`. Atribut `prioQueue` merupakan representasi dari struktur data *Priority Queue*. Elemen dari atribut ini bertipe `tuple` yang menyimpan *layout puzzle*, kedalaman *node*, dan *index* dari `stepTree` yang menyatakan langkah yang diambil *node*. Atribut `stepTree` merupakan representasi dari struktur data *Tree*. Atribut ini menyimpan langkah langkah yang diambil oleh sebuah *node* di `prioQueue`.

Langkah-Langkah Algoritma *Branch and Bound*

1. Masukkan kondisi awal *puzzle* ke `prioQueue`.
2. *Dequeue* `prioQueue`. Simpan ke variabel `nObj`.
3. Selama `nObj` bukan simpul hasil dan waktu eksekusi dibawah 5 menit:
    1. Hasilkan langkah-langkah yang mungkin diambil dari kondisi `nObj`.
    2. Masukkan hasil-hasil dari langkah sebelumnya ke prioQueue.
    3. *Dequeue* `prioQueue`. Simpan ke variabel `nObj`.
4. Cek `nObj`:
    - Jika `nObj` merupakan solusi, *return* `nObj` dan waktu eksekusi.
    - Selain itu, *raise error*.

Source Code

`GUI.py`

```
import tkinter as tk
from tkinter import filedialog as fd
import random as rd
import PuzzleGUI as PGUI
import Parser

class GUI:
    def __init__(self, interactive=True):
        # Inisialisasi Window Utama GUI
        self.window = tk.Tk()
        self.window.title("15 Puzzle")
```

```
        self.window.geometry("600x400")
        self.window.resizable(False, False)

        # Tambahin Puzzle
        self.puzzle = PGUI.PuzzleGUI(self.window)

        # Tambah tombol open file
        openButton = tk.Button(
                self.window,
                text = "Open File",
                command = (self.from_file)
            )

        openButton.place(x = 450, y = 25, width = 100)

        # Tambah tombol solve
        solveButton = tk.Button(
                self.window,
                text = "Solve",
                command = (self.solve)
            )

        solveButton.place(x = 450, y = 60, width = 100)

        # Tambah tombol reset
        resetButton = tk.Button(
                self.window,
                text = "Reset",
                command = (self.reset)
            )

        resetButton.place(x = 450, y = 95, width = 100)

        if(interactive):
            self.bind_key()

    def solve(self):
        if self.puzzle.reachable():
            try:
                t, nAwakened = self.puzzle.solve()
                tk.messagebox.showinfo(
                    "Solve Completed",
                    "Selesai dalam {:.2f} detik dan \
                    membangkitkan {} simpul".format(t, nAwakened))
            except Exception as err:
                tk.messagebox.showerror("Error", err)

        else:
            tk.messagebox.showinfo("Information", "Puzzle cannot be Solved!")

    def from_file(self):
        filename = fd.askopenfilename()
        try:
            l = Parser.Parser.parse(filename)
            self.load_layout(l)
        except Exception as err:
            tk.messagebox.showerror("Error", err)
```

```
    def reset(self):
        self.load_layout([str(i) for i in range(1, 17)])

    def show(self):
        self.window.mainloop()

    def random_layout(self):
        normal = [str(i) for i in range(1, 17)]
        rd.shuffle(normal)
        self.load_layout(normal)

    def load_layout(self, layout):
        self.puzzle.load_layout(layout)

    def bind_key(self):
        # Key Gerak
        self.window.bind("<KeyPress-Left>", lambda _: self.puzzle.left())
        self.window.bind("<KeyPress-Right>", lambda _: self.puzzle.right())
        self.window.bind("<KeyPress-Up>", lambda _: self.puzzle.up())
        self.window.bind("<KeyPress-Down>", lambda _: self.puzzle.down())

        # Key Random Reset
        self.window.bind("<space>", lambda _: self.random_layout())

if __name__ == "__main__":
    G = GUI()
    G.show()
```

Parser.py

```
from os.path import exists

class Parser:
    def load_file(filename):
        f = open(filename, "r")
        res = f.read().split()
        print(res)
        return res

    def exist_check(filename):
        return exists(filename)

    def format_check(arr):
        # Pastikan semua file ada
        l = len(arr)
        if(l != 16):
            return False
        else:
            for i in range(1, 17):
                if (str(i) not in arr):
                    return False
            return True

    def parse(filename):
        if(not Parser.exist_check(filename)):
            raise Exception("Error: File not Exist")
```

```
        l = Parser.load_file(filename)

        if(not Parser.format_check(l)):
            raise Exception("Error: Wrong Format")

        return l
```

PuzzleGUI.py

```
import tkinter as tk
import time
import PuzzleSolver as PS

class PuzzleGUI:
    def __init__(self, master, layout=[str(i) for i in range(1,17)]):

        self.canvas = tk.Canvas(
                master,
                width=400,
                height=400
            )

        self.rect_size = 100

        self.load_layout(layout)

        self.canvas.pack(side="left")

        self.is_moving = False

    def load_layout(self, layout):
        # Ngebikin tiap bagian puzzle
        self.canvas.delete("all")
        self.tiles = layout
        for idx, tile in enumerate(layout):
            if(tile != "16"):
                tp = (idx //  4) * self.rect_size
                lt = (idx %  4) * self.rect_size
                self.canvas.create_rectangle(
                        lt,
                        tp,
                        lt + self.rect_size,
                        tp + self.rect_size,
                        fill="white",
                        tags="tile" + tile
                    )

                self.canvas.create_text(
                        lt + self.rect_size // 2,
                        tp + self.rect_size // 2,
                        text= tile,
                        font = ("Arial", 22),
                        tags="tile" + tile
                    )

                self.canvas.create_text(
                        lt + self.rect_size // 8,
```

```
                    tp + self.rect_size // 8,
                    text= 0,
                    font = ("Arial", 11),
                    tags=("tile" + tile, "kurangTile" + tile)
                )
            else:
                self.xIdx = idx

        self.refreshKurang()

    def refreshKurang(self):
        # Hitung ulang nilai KURANG di display
        for tile in self.tiles:
            if tile != '16':
                # print("kurangTile" + tile)
                self.canvas.itemconfigure(
                    self.canvas.find_withtag("kurangTile" + tile)[0],
                    text = self.KURANG(tile)
                )

    def solve(self):
        ps = PS.PuzzleSolver()
        sSteps, t, found, nAwakened = ps.solve(self.tiles)
        if not found:
            raise Exception("Time Limit Reached")
        self.arr_move(sSteps)

        return t, nAwakened

    def str_move(self, move):
        if (move == "Up"):
            self.up()
        elif (move ==  "Down"):
            self.down()
        elif (move == "Right"):
            self.right()
        elif (move == "Left"):
            self.left()

    def arr_move(self, arr):
        for el in arr:
            self.str_move(el)

    def moveRec(self, tag, x, y):
        # Bergerak sedikit-sedikit supaya terlihat seperti animasi
        newX = x
        if (x > 0):
            self.canvas.move(tag, 5, 0)
            newX -= 5
        elif (x < 0):
            self.canvas.move(tag, -5, 0)
            newX += 5

        newY = y
        if (y > 0):
            self.canvas.move(tag, 0, 5)
            newY -= 5
```

```
        elif (y < 0):
            self.canvas.move(tag, 0, -5)
            newY += 5

        if(x != 0 or y != 0):
            self.canvas.update()
            self.canvas.after(10)
            self.moveRec(tag, newX, newY)

    def move(self, tag, x, y):
        self.moveRec(tag, x, y)
        self.refreshKurang()

    def swap(self, idx):
        # Tukar Tile
        tmp = self.tiles[idx]
        self.tiles[idx] = "16"
        self.tiles[self.xIdx] = tmp
        self.xIdx = idx

        return tmp

    def down(self):
        # Yang kosong ke Bawah
        if(not self.is_moving and self.xIdx // 4 != 3):
            print("Down")
            self.is_moving = True
            nLoc = self.xIdx + 4
            nTag = self.swap(nLoc)
            self.move("tile" + nTag, 0, -self.rect_size)
            print(self.tiles)
            self.is_moving = False

    def up(self):
        if(not self.is_moving and self.xIdx // 4 != 0):
            print("Up")
            self.is_moving = True
            nLoc = self.xIdx - 4
            nTag = self.swap(nLoc)
            self.move("tile" + nTag, 0, self.rect_size)
            print(self.tiles)
            self.is_moving = False

    def right(self):
        if(not self.is_moving and self.xIdx % 4 != 3):
            print("Right")
            self.is_moving = True
            nLoc = self.xIdx + 1
            nTag = self.swap(nLoc)
            self.move("tile" + nTag, -self.rect_size, 0)
            print(self.tiles)
            self.is_moving = False

    def left(self):
        if(not self.is_moving and self.xIdx % 4 != 0):
            print("Left")
            self.is_moving = True
```

```
            nLoc = self.xIdx - 1
            nTag = self.swap(nLoc)
            self.move("tile" + nTag, self.rect_size, 0)
            print(self.tiles)
            self.is_moving = False

    def KURANG(self, tile):
        res = int(tile) - 1
        idx = 0
        while(self.tiles[idx] != tile):
            if(int(tile) > int(self.tiles[idx])):
                res -= 1
            idx += 1
        return res

    def ALL_KURANG(self):
        res = 0
        for i in range(1, 17):
            res += self.KURANG(str(i))
        return res

    def X_val(self):
        row = (self.xIdx % 4) % 2 == 0
        col = (self.xIdx // 4) % 2 == 0

        return row ^ col

    def reachable(self):
        return (self.ALL_KURANG() + self.X_val()) % 2 == 0
```

PuzzleSolver.py

```
import time

class PrioQueue:
    def __init__(self, layout):
        self.prioQueue = [(layout, 0, 0)]
        self.stepTree = [("None", 0)]
        self.treeLastIdx = 0
        self.lastIdx = 0
        self.entryDict = {}

    def enqueue(self, nObj, nStep):
        layout_str = self.layout_to_str(nObj[0])
        # Cek apakah kondisi puzzle sudah pernah muncul
        if (layout_str not in self.entryDict):

            # Tambahin step di tree
            self.stepTree.append(nStep)
            self.treeLastIdx += 1

            # Tambahin node
            idx = self.get_insert_index(nObj[0], nObj[1])
            self.prioQueue.insert(idx, (nObj[0], nObj[1], self.treeLastIdx))

            self.entryDict[layout_str] = True
```

```
            self.lastIdx += 1

    def layout_to_str(self, layout):
        return ";".join(layout)

    def dequeue(self):
        self.lastIdx -= 1
        return self.prioQueue.pop(0)

    def g(self, P):
        res = 0
        for i in range(1, 17):
            if (str(i) != P[i - 1]):
                res += 1
        return res

    def get_insert_index(self, layout, fP):
        # Menggunakan binary search
        lBound = 0
        uBound = self.lastIdx
        if uBound < 0:
            return 0
        elL = self.prioQueue[(lBound + uBound) // 2]
        elHR = self.h(elL[1], elL[0])
        crHR  = self.h(fP, layout)
        while(lBound != uBound and elHR != crHR):
            if(elHR < crHR):
                lBound = (lBound + uBound) // 2 + 1
            elif (elHR > crHR):
                uBound = (lBound + uBound) // 2

            elL = self.prioQueue[(lBound + uBound) // 2]
            elHR = self.h(elL[1], elL[0])

        return lBound + (lBound <= crHR)

    def h(self, fP, layout):
        return fP + self.g(layout)

    def get_step(self, nObj):
        return self.stepTree[nObj[2]]

    def get_full_step(self, nIdx):
        # Mengambil langkah-langkah
        if nIdx == 0:
            return []
        else:
            step, pIdx = self.stepTree[nIdx]
            return self.get_full_step(pIdx) + [step]

    def show(self):
        for el in self.prioQueue:
            print(el[0], el[1], self.g(el[0]))

class PuzzleSolver:

    def solve(self, layout):
        st = time.time()
```

```
        self.prioQueue = PrioQueue(layout)

        nObj = self.prioQueue.dequeue()
        checkNum = 1
        print(f"Check Number: {checkNum}, Height: {nObj[1]}") # Logging

        # Diulang selama bukan solusi dan dibawah 5 menit
        while(self.g(nObj[0]) != 0 and time.time() - st < 300):
            self.gen_branch(nObj)
            nObj = self.prioQueue.dequeue()

            checkNum += 1
            print(f"Check Number: {checkNum}, Height: {nObj[1]}") # Logging

        # Langkah, Waktu Eksekusi, Apakah berhasil, jumlah simpul yang dieksekusi
        return (
            self.prioQueue.get_full_step(nObj[2]),
            time.time() - st,
            self.g(nObj[0]) == 0,
            self.prioQueue.treeLastIdx + 1
        )

    def show(self):
        self.prioQueue.show();

    def gen_branch(self, nObj):
        layout, fP, sIdx = nObj
        child = self.gen_child(layout)
        for c in child:
            # Bukan lawan dari langkah sebelumnya
            # Contoh:
            #   Kalau sebelumnya Left, kali ini gak bisa Right
            if (c[1] != "None" and \
                self.prioQueue.get_step(nObj) != self.inv_mov(c[1])):
                self.prioQueue.enqueue((c[0], fP + 1, 0), (c[1], sIdx))

    def inv_mov(self, mov):
        if(mov == "Up"):
            return "Down"
        elif (mov == "Down"):
            return "Up"
        elif (mov == "Right"):
            return "Left"
        elif (mov == "Left"):
            return "Right"
        else:
            return mov

    def get_xIdx(self, layout):
        for idx, item in enumerate(layout):
            if item == "16":
                return idx
        return -1

    def g(self, P):
        res = 0
        for i in range(1, 17):
```

```
            if (str(i) != P[i - 1]):
                res += 1
        return res

    def swap(self, layout, xA, xB):
        tmp = layout[xA]
        layout[xA] = layout[xB]
        layout[xB] = tmp

    def gen_up(self, layout, xIdx):
        if(xIdx // 4 != 0):
            newLayout = list(layout)
            nLoc = xIdx - 4
            nTag = self.swap(newLayout, xIdx, nLoc)
            return (newLayout, "Up")
        return ([], "None")

    def gen_down(self, layout, xIdx):
        if(xIdx // 4 != 3):
            newLayout = list(layout)
            nLoc = xIdx + 4
            nTag = self.swap(newLayout, xIdx, nLoc)
            return (newLayout, "Down")
        return ([], "None")

    def gen_right(self, layout, xIdx):
        if(xIdx % 4 != 3):
            newLayout = list(layout)
            nLoc = xIdx + 1
            nTag = self.swap(newLayout, xIdx, nLoc)
            return (newLayout, "Right")
        return ([], "None")

    def gen_left(self, layout, xIdx):
        if(xIdx % 4 != 0):
            newLayout = list(layout)
            nLoc = xIdx - 1
            nTag = self.swap(newLayout, xIdx, nLoc)
            return (newLayout, "Left")
        return ([], "None")

    def gen_child(self, layout):
        # Membuat anak di segala arah
        xIdx = self.get_xIdx(layout)
        return  [self.gen_up(layout, xIdx)]    + \
                [self.gen_down(layout, xIdx)]  + \
                [self.gen_left(layout, xIdx)]  + \
                [self.gen_right(layout, xIdx)]
```

Testing Program

Test Case 1

File input *Test Case 1*:

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

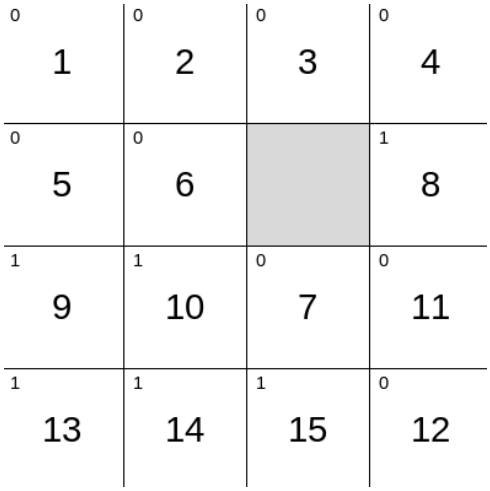

Figure 1: Input Test Case 1

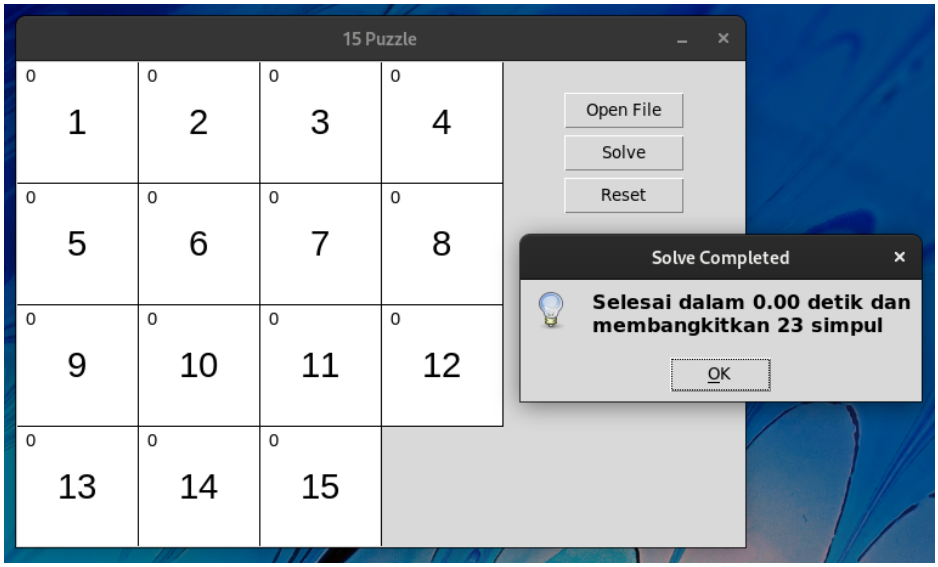

Figure 2: Output Test Case 1

Test Case 2

File input *Test Case 2*:

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

0 1	1 3	1 4	11 15
0 2		0 5	6 12
1 7	0 6	3 11	4 14
0 8	0 9	0 10	0 13

Figure 3: Input Test Case 2

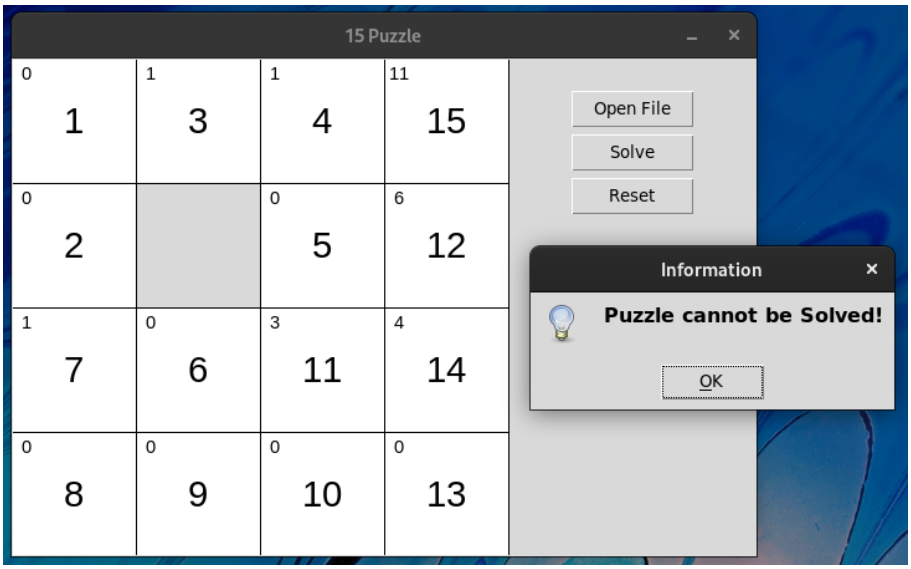

Figure 4: Output Test Case 2

Test Case 3

File input *Test Case 3*:

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

4 5	2 3	3 6	
5 9	0 1	2 7	1 4
5 13	0 2	5 15	0 8
3 14	0 10	1 12	0 11

Figure 5: Input Test Case 3

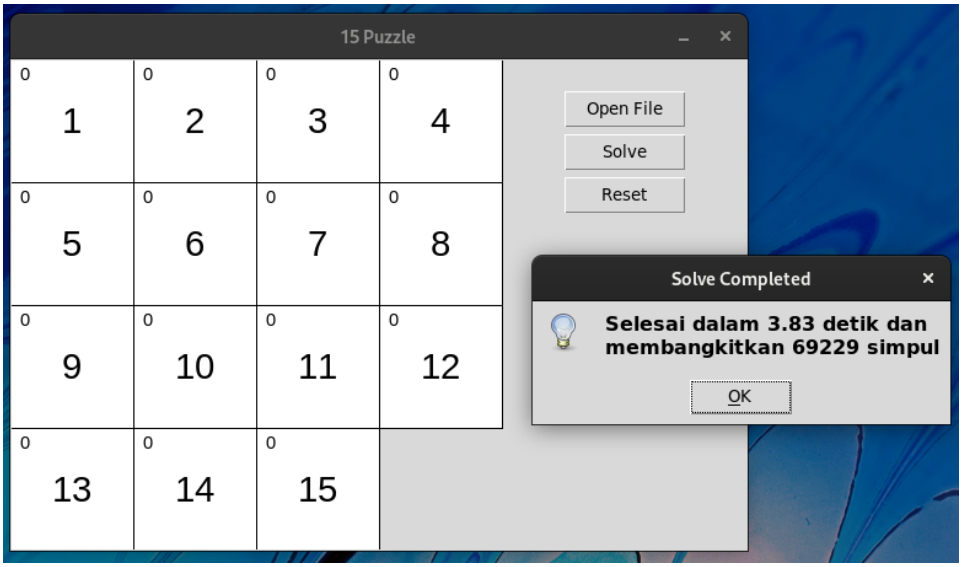

Figure 6: Output Test Case 3

Test Case 4

File input *Test Case 4*:

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

4 5	2 3	7 10	2 4
5 9		0 1	3 8
2 7	0 2	0 6	0 11
1 13	1 14	1 15	0 12

Figure 7: Input Test Case 4

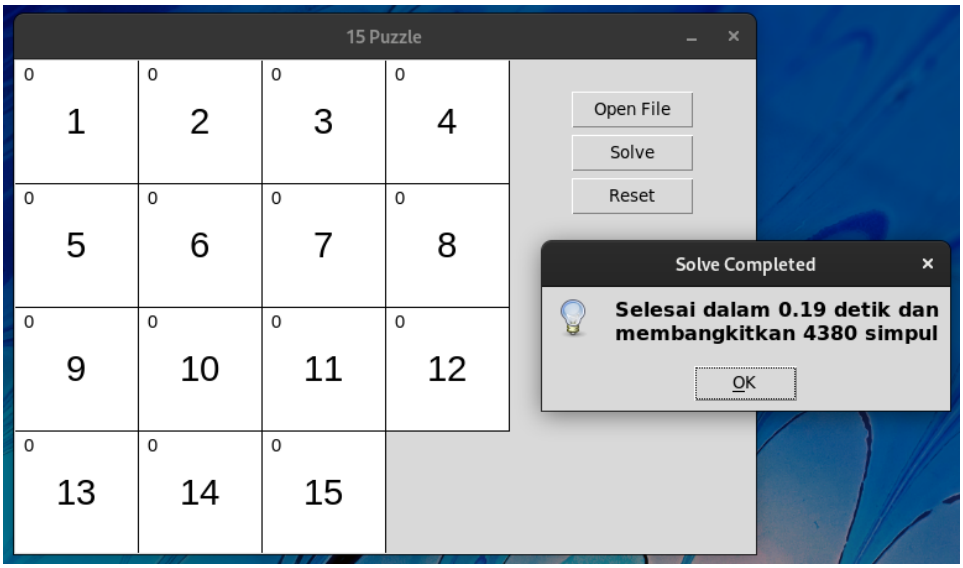

Figure 8: Output Test Case 4

Test Case 5

File input *Test Case 5*:

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

13 14	10 11	3 4	10 13
2 3		4 7	4 8
3 6	2 5	4 12	1 2
3 15	2 10	0 1	0 9

Figure 9: Input Test Case 5

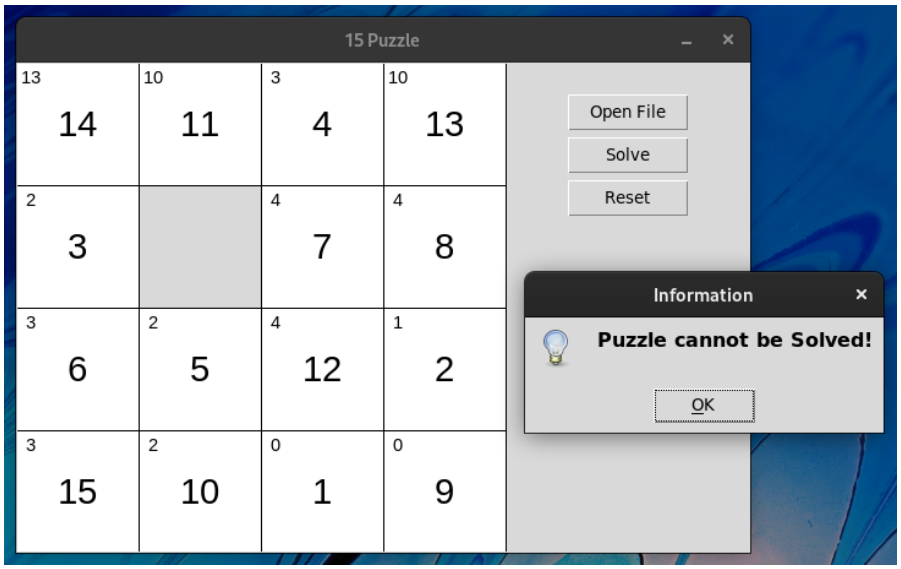

Figure 10: Output Test Case 5

Lampiran

Check Lists

Poin	Ya
Program berhasil dikompilasi	×
Program berhasil running	×
Program dapat menerima input dan menuliskan output.	×
Luaran sudah benar untuk semua data uji	×
Bonus dibuat	×

Link

GitHub Link

File Test Cases

`tc1.txt`

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

`tc2.txt`

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

`tc3.txt`

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

`tc4.txt`

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

`tc5.txt`

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