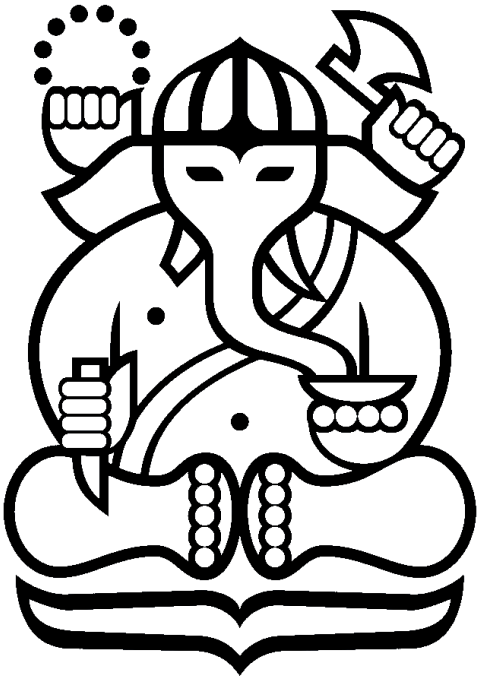
Laporan Tugas Kecil 3

IF2211 Strategi Algoritma

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

Disusun oleh:

Diky Restu Maulana 13520017



PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

1. Algoritma *Branch and Bound*

Branch and Bound adalah algoritma yang digunakan untuk persoalan optimisasi, yaitu meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar Batasan (*constraint*) persoalan. Pada algoritma ini, setiap simpul diberi sebuah nilai *cost*:

$\hat{c}(i)$ = nilai taksiran lintasan termurah ke simpul status tujuan yang melalui simpul status i

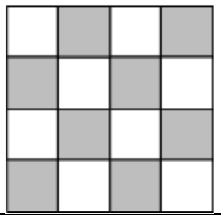
Simpul berikutnya yang akan di-*expand* tidak lagi berdasarkan urutan pembangkitannya, tetapi simpul yang memiliki *cost* yang paling kecil (*least cost search*) pada kasus minimasi.

Algoritma Branch and Bound menerapkan pemangkasan pada jalur yang dianggap tidak lagi mengarah kepada solusi. Pemangkasan ini dilakukan berdasarkan fungsi pembatas (*bounding function*). Kriteria pemangkasan secara umum adalah sebagai berikut.

- Nilai simpul tidak lebih baik dari nilai terbaik sejauh ini (*the best solution so far*)
- Simpul tidak merepresentasikan solusi yang *feasible* karena ada batasan yang dilanggar
- Solusi pada simpul tersebut hanya terdiri atas satu titik → tidak ada pilihan lain, bandingkan nilai fungsi objektif dengan solusi terbaik saat ini, ambil yang terbaik

Persoalan 15 puzzle dapat diselesaikan dengan algoritma *Branch and Bound*. Status dibangkitkan berdasarkan posisi setiap ubin dan aksi yang dapat dilakukan adalah *up, down, left,* dan *right*. Aksi dilakukan untuk menggeser ubin kosong ke arah tersebut.

Teorema: Status tujuan hanya dapat dicapai dari status awal jika $\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap.
$KURANG(i)$ = banyaknya ubin bernomor j sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$
$POSISI(i)$ = posisi ubin bernomor i pada susunan yang diperiksa
X = 1 jika ubin kosong pada posisi awal ada pada sel yang diarsir



Cost simpul P pada 15 puzzle dihitung sebagai

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

$f(P) =$ panjang lintasan dari simpul akar ke P
$\hat{g}(P) =$ taksiran panjang lintasan terpendek dari P ke simpul solusi pada upapohon berakar P

Implementasi elemen-elemen *Branch and Bound* dalam algoritma yang telah dibuat adalah sebagai berikut.

Kelas	Keterangan
Puzzle	Memiliki atribut board yang merupakan representasi posisi ubin dan n yang merupakan ukuran puzzle
PriorityQueue	Memiliki atribut queue yang menyimpan antrian node yang akan diperiksa selanjutnya. Antrian tersebut diurutkan berdasarkan sebuah fungsi yang disimpan sebagai atribut func
Node	Memiliki atribut puzzle sebagai posisi ubin, parent yang menyimpan parent node, move yang menyimpan aksi untuk mencapai state ini, dan depth yang menyimpan kedalaman node ini di dalam space state tree.

$f(P)$ dihitung dari atribut depth pada kelas Node dan $\hat{g}(P)$ dihitung menggunakan fungsi check_misplaced_tiles(puzzle) yang terdapat di dalam main.py sehingga $\hat{c}(P)$ untuk sebuah node dihitung dari hasil penjumlahan kedua fungsi tersebut.

Langkah-langkah penyelesaian 15 puzzle menggunakan algoritma *Branch and Bound* adalah sebagai berikut.

1. Masukkan simpul akar ke dalam antrian Q (sebuah priority queue yang terurut membesar berdasarkan *cost*. Jika *cost* sama, *node* terbaru diprioritaskan). Jika simpul akar adalah simpul solusi (*goal node*) maka solusi telah ditemukan.
2. Jika Q kosong, stop.
3. Jika Q tidak kosong, pilih simpul i yang merupakan elemen pertama pada Q.
4. Jika simpul i adalah simpul solusi, maka solusi telah ditemukan. Matikan semua simpul yang memiliki *cost* lebih besar daripada simpul i.
5. Jika simpul i bukan solusi, bangkitkan semua anaknya yang mungkin berdasarkan pilihan aksi (*up, down, left,* dan *right*). Jika i tidak memiliki anak, kembali ke langkah 2.
6. Untuk setiap anak j dari simpul i, hitung *cost*-nya dan masukkan ke dalam Q.
7. Kembali ke langkah 2.

Program ini juga dilengkapi dengan *Graphical User Interface* yang dibuat dengan memanfaatkan library tkinter.

2. Kode Program dalam Bahasa Python

2.1. puzzle.py

```python
# puzzle.py
# Containing Puzzle class to represent states

import copy
class Puzzle:
    # Constructor
    def __init__(self, path):
        # Define board and size
        self.board = []
        self.n = 0

        f = open(path, "r")
        for line in f:
            self.board.append(list(map(lambda x : int(x), line.split())))

        self.n = len(self.board)

    # Find empty cell position
    # Returns : (row, colunn)
    def find_empty(self):
        for i,row in enumerate(self.board):
            for j,value in enumerate(row):
                if (value == self.n**2):
                    return (i,j)

    # Move empty cell to (r+dr, c+dc)
    def move(self, dr, dc):
        (r, c) = self.find_empty()
        if(r+dr>=0 and r+dr<self.n and c+dc>=0 and c+dc<self.n):
            moved_puzzle = copy.deepcopy(self)
            moved_puzzle.board[r][c], moved_puzzle.board[r+dr][c+dc] = moved_puzzle.board[r+dr][c+dc],
moved_puzzle.board[r][c]
            return moved_puzzle
        else:
            return None

    # Test whether the puzzle is solvable or not, inversion, parity, total, and kurang(i) array
    def is_solveable(self):
        # Get empty cell location
        (r, c) = self.find_empty()

        # Flatten board
        tmp = self.flattened_board()

        # Find empty cell parity
        x = (r+c) % 2

        sum = 0
        kurang_i = [0 for i in range(self.n**2)]
        for i in range(0,self.n**2):
            sum_ubin = 0
            for j in range(i+1,self.n**2):
                if(tmp[i]>tmp[j]):
                    sum+=1
                    sum_ubin+=1
            kurang_i[tmp[i]-1] = sum_ubin

        return (sum + x) % 2 == 0, sum, x, sum+x, kurang_i

    # Return flattened board
    def flattened_board(self):
        return [val for arr in self.board for val in arr]
```

2.2. prioQueue.py

```python
# prioQueue.py
# Contains PriorityQueue class

class PriorityQueue:
    # Constructor
    def __init__(self, priority_function):
        self.queue = []
        self.func = priority_function

    # Check if PQ is empty
    def is_empty(self):
        return len(self.queue) == 0

    # Peek function
    def front(self):
        return self.queue[0]

    # Insert item with corresponding function; O(n)
    def push(self, item):
        pos = 0
        found = False

        while(not found and pos < len(self.queue)):
            if(self.func(item, self.queue[pos])):
                found = True
            else:
                pos+=1

        self.queue.insert(pos, item)

    # Remove front item; O(1)
    def pop(self):
        self.queue.pop(0)
```

2.3. *node.py*

```python
# Node.py
# Contains the class Node to represent the state space tree in BnB

class Node:
    def __init__(self, puzzle, parent=None, depth=0, move=""):
        self.puzzle = puzzle
        self.parent = parent
        self.move = move
        self.depth = depth
```

2.4. *main.py*

```python
# main.py
# Contains main program including GUI

import tkinter as tk
import time
from puzzle import Puzzle
from prioQueue import PriorityQueue
from node import Node

# g(i) function for checking misplaced tiles
def check_misplaced_tiles(puzzle):
    result = 0
    flat = puzzle.flattened_board()

    for i in range(1, puzzle.n**2+1):
        if(flat[i-1] != i):
            result += 1

    return result

# Check whether matrix is sorted or not
def check_goal(puzzle):
    flat = puzzle.flattened_board()

    for i in range(1, (puzzle.n**2)+1):
        if(flat[i-1] != i):
            return False

    return True
```

```python
# Command fot solve button
def solve_button_clicked():
    global xypos, solution_array, root, moves_names, moves_units

    # Clear all buttons in puzzle frame
    clear_puzzle_frame()

    # Variable to store the solution
    solution_array = None

    # Get filename from user input
    filename = filename_entry.get()
    root = Node( Puzzle("../test/" + filename) )
    flat = root.puzzle.flattened_board()

    # Create dict of button positions and display puzzle
    xypos = {}
    xypos["empty"] = root.puzzle.find_empty()
    for i in range(root.puzzle.n):
        for j in range(root.puzzle.n):
            num = flat[i*root.puzzle.n + j]
            xypos[num] = (i, j)
            if (num != root.puzzle.n**2):
                num_btn = tk.Button(master=puzzle_frm, text=num, font=font_puzzle, fg="blue", width=4,
height=2, bg="skyblue")
                num_btn["command"] = lambda num_btn=num_btn: puzzle_switch(num_btn)
                num_btn.place(relx=j/root.puzzle.n, rely=i/root.puzzle.n, relwidth=1/root.puzzle.n,
relheight=1/root.puzzle.n)

    # Check if puzzle is solvable
    is_solveable, inversion, parity, total, kurang_i = root.puzzle.is_solveable()
    if (not is_solveable):
        solveable_lbl["text"] = "Puzzle is unsolvable."
        solveable_lbl["bg"] = "light salmon"
        display_table(kurang_i)
        verdict_lbl["text"] = "Inversions: " + str(inversion) + "\nParity: " + str(parity) + "\nTotal: "
+ str(total)
        move_lbl["text"] = "---"
        details_lbl["text"] = "---"
        return
    solveable_lbl["text"] = "Puzzle is solvable."
    solveable_lbl["bg"] = "light green"
    display_table(kurang_i)
    verdict_lbl["text"] = "Inversions: " + str(inversion) + "\nParity: " + str(parity) + "\nTotal: " +
str(total)

    # Node generated count
    node_count = 1

    # Make priority queue for branching
    # On priority : lowest cost with last in first
    cost_function = check_misplaced_tiles
    pq = PriorityQueue(lambda x,y : x.depth + cost_function(x.puzzle) <= y.depth +
cost_function(y.puzzle))

    # Initiate priority queue
    pq.push(root)

    # Variable to store solution state
    solution_state = None

    # List possible moves for puzzle
    moves_units = [(-1,0), (0,-1), (1,0), (0,1)]
    moves_names = ["Up", "Left", "Down", "Right"]

    # Start timer
    time_start = time.process_time_ns()

    # Searching for solution using Branch and Bound
    while(not pq.is_empty()):
        # Get front item in queue
        current = pq.front()
        pq.pop()

        # If currently checking final state, save the current state
        if (check_goal(current.puzzle)):
            solution_state = current
            break
```

```
        # Append generate states to pq
        for i, (dr, dc) in enumerate(moves_units):
            # If moves are NOT opposite to previous move, generate new node
            if(moves_names[(i+2)%4] != current.move):
                # Generate node
                result = Node(current.puzzle.move(dr, dc), parent=current, depth=current.depth+1,
move=moves_names[i])

                # If move is possible..
                if(result != None and result.puzzle != None):
                    node_count += 1
                    pq.push( result )

    # Generate solution from result
    solution_array = generate_solution(solution_state)

    # Print the first move
    move = solution_array[0].move
    move_lbl["text"] = move

    # Stop timer
    time_stop = time.process_time_ns()

    time_taken = (time_stop - time_start) / 1000000
    details_lbl["text"] = "Total moves: " + str(len(solution_array)) + "\n" + str(node_count) + " nodes
generated\n" + str(time_taken) + " ms taken"

# -=-=-=-=- MAIN PROGRAM -=-=-=-=- #

# Main window
window = tk.Tk()
window.title("15 PUZZLE SOLVER")

# Font list
font_title = ('Monaco', 20, 'bold')
font_label = ('Times', 12)
font_puzzle = ('Comic Sans MS', 12)

# Entry frame contains title, filename input, and solve button
entry_frm = tk.Frame(master=window, height=100, bg="gray80")
entry_frm.pack(fill=tk.X)
title_lbl = tk.Label(master=entry_frm, text="15 PUZZLE SOLVER", font=font_title, bg="gray80")
title_lbl.pack(pady=5)
filename_lbl = tk.Label(master=entry_frm, text="Enter the file name:", font=font_label, bg="gray80")
filename_lbl.pack(side=tk.LEFT)
filename_entry = tk.Entry(master=entry_frm, font=font_label, bg="white")
filename_entry.pack(pady=5, padx=10, side=tk.LEFT)
solve_btn = tk.Button(master=entry_frm, text="solve", font=font_label, bg="gray80",
command=solve_button_clicked)
solve_btn.pack(pady=5, side=tk.LEFT)

# Center frame contains puzzle, move, and details
center_frm = tk.Frame(master=window, height=250, width=250, relief=tk.SUNKEN, borderwidth=4, bg="gray80")
center_frm.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
details_lbl = tk.Label(master=center_frm)
details_lbl.pack(side=tk.BOTTOM, fill=tk.X)
verdict_lbl = tk.Label(master=center_frm)
verdict_lbl.pack(side=tk.BOTTOM, fill=tk.X)
move_lbl = tk.Label(master=center_frm, font=font_label)
move_lbl.pack(side=tk.TOP, fill=tk.X)
puzzle_frm = tk.Frame(master=center_frm, height=250, width=250, bg="yellow")
puzzle_frm.pack()

# Table frame contains kurang(i) table
table_frm = tk.Frame(master=window, bg="gray80")
table_frm.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
solveable_lbl = tk.Label(master=table_frm, font=font_label)
solveable_lbl.pack(side=tk.TOP, fill=tk.X)
kurang_frm = tk.Frame(master=table_frm, bg="gray80")
kurang_frm.pack()

window.mainloop()
```

3. File Text

3.1. solvable1.txt

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

3.2. solvable2.txt

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

3.3. solvable3.txt

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

3.4. unsolvable1.txt

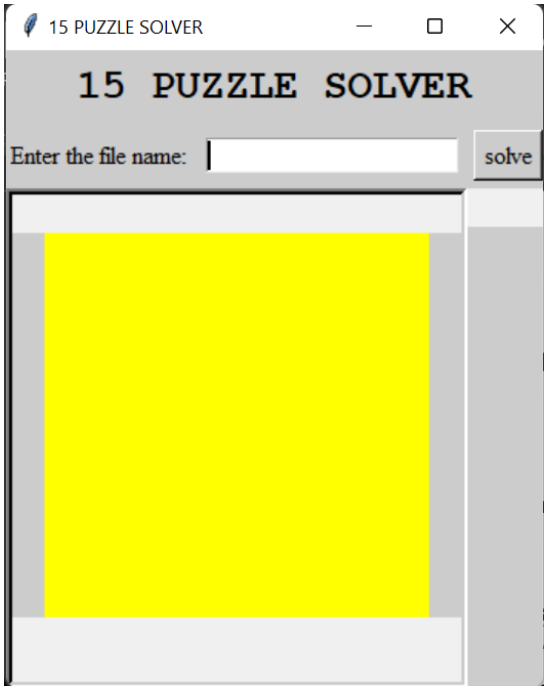
```
2 1 4 7
5 6 16 3
9 11 12 8
13 10 14 15
```

3.5. unsolvable2.txt

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

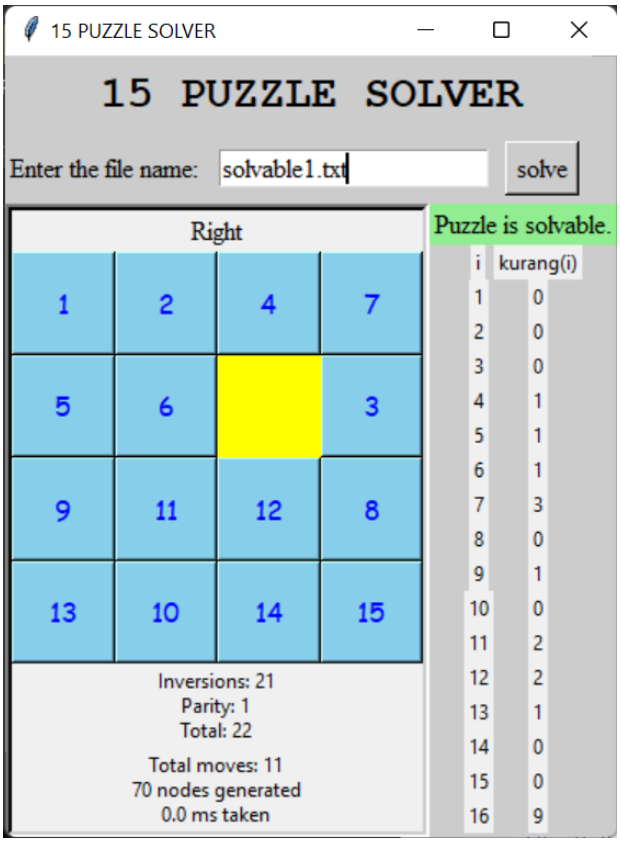
4. Input Output Program

4.1. Tampilan Awal

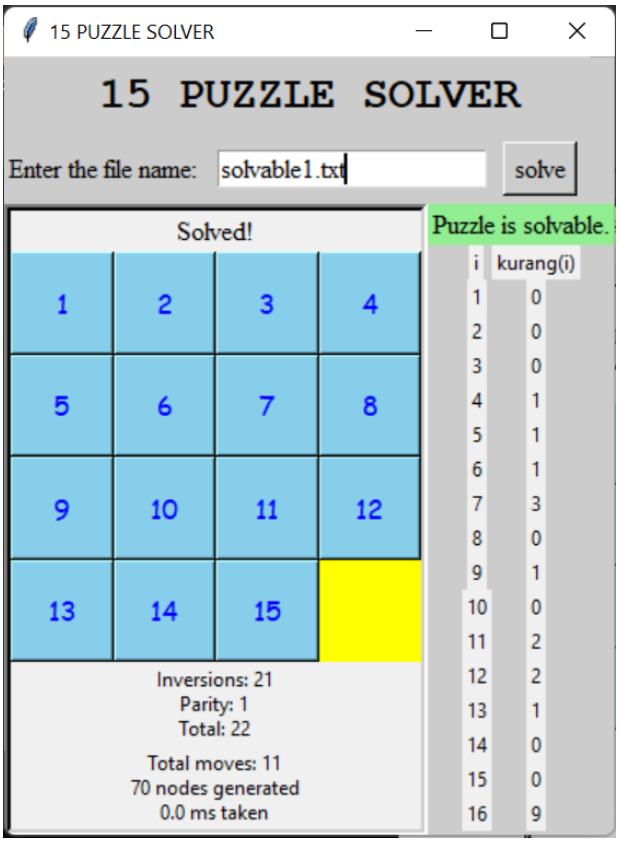


4.2. solvable1.txt

4.2.1. Sebelum dimainkan

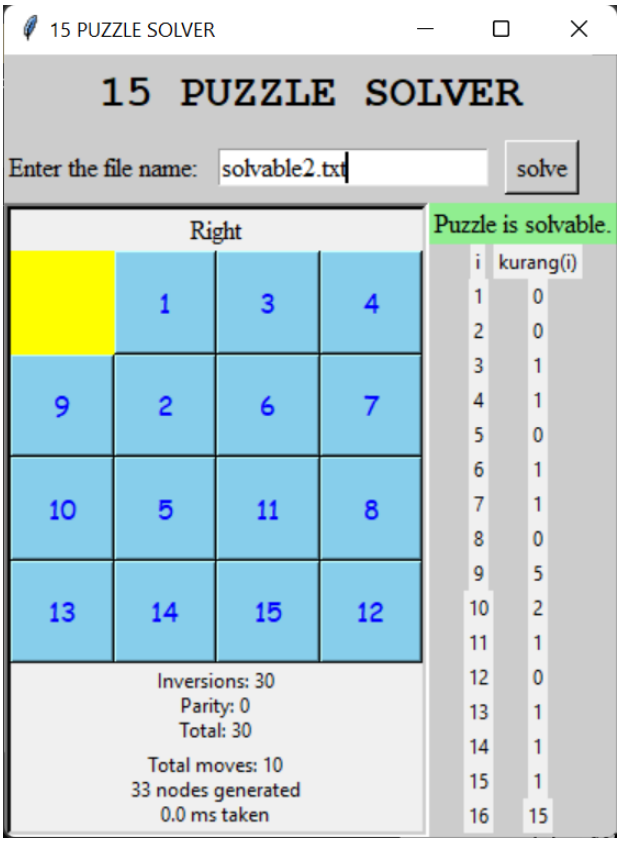


4.2.2. Setelah dimainkan

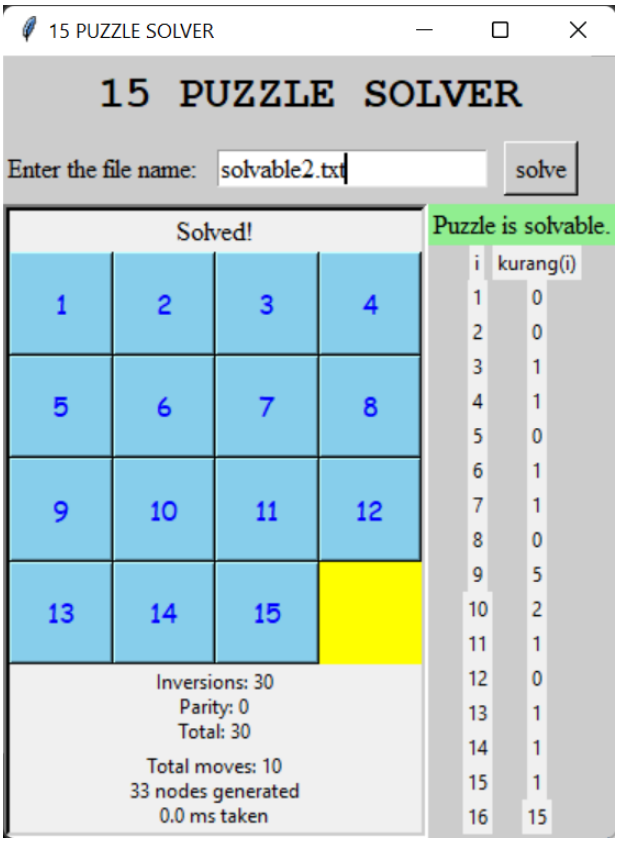


4.3. solvable2.txt

4.3.1. Sebelum dimainkan

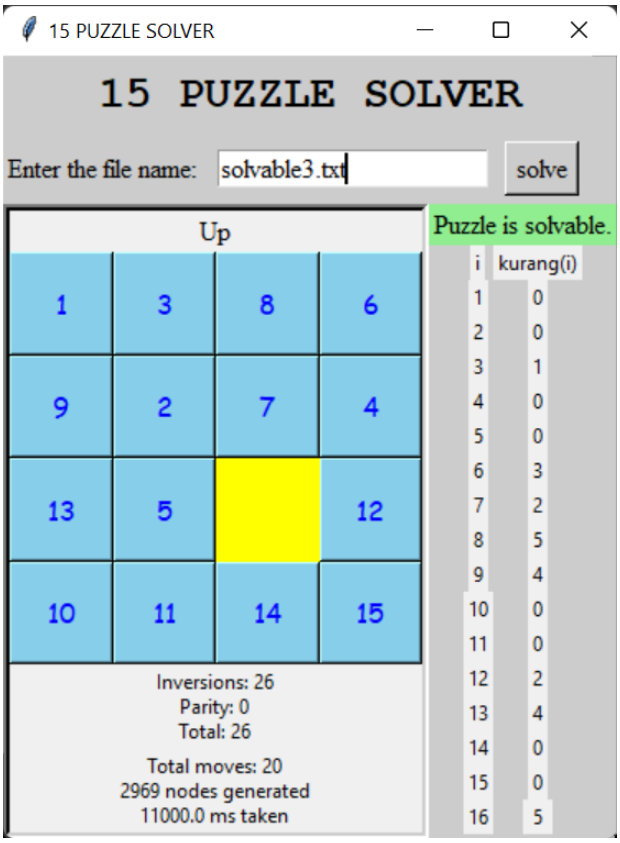


4.3.2. Setelah dimainkan

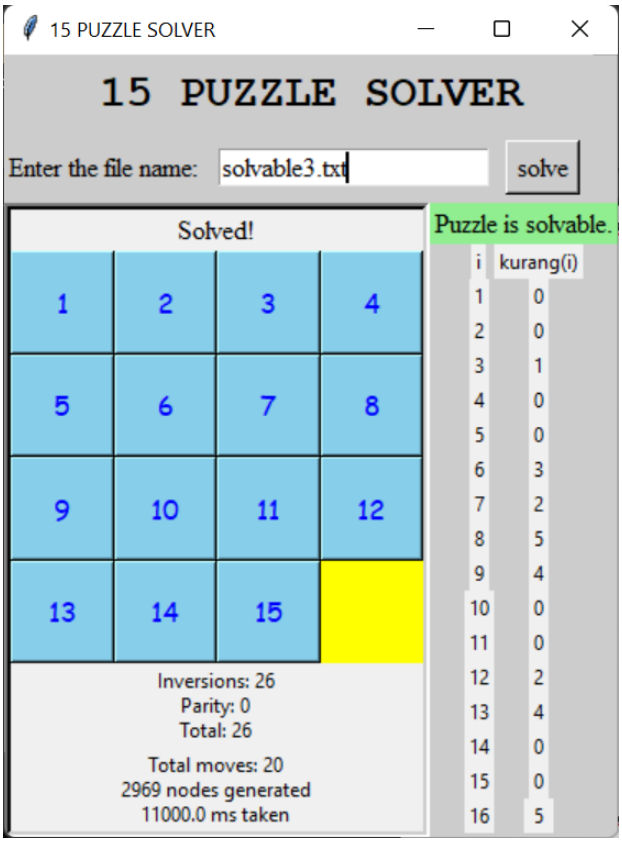


4.4. solvable3.txt

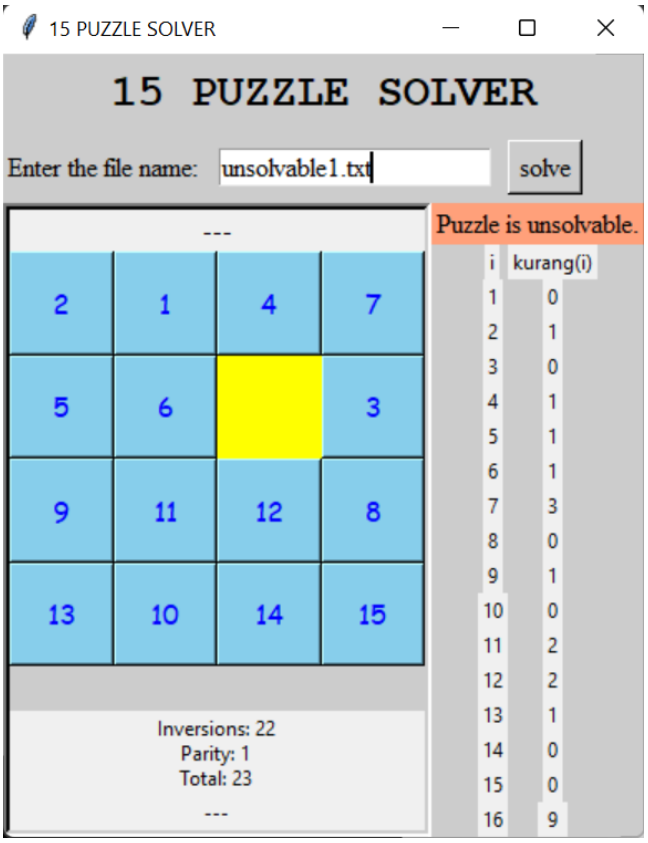
4.4.1. Sebelum dimainkan



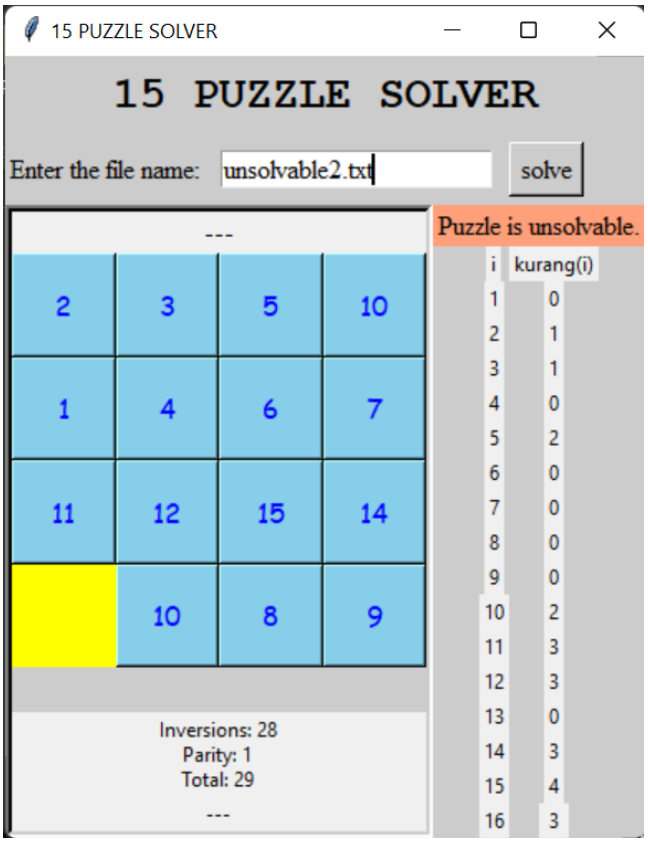
4.4.2. Setelah dimainkan



4.5. unsolvable1.txt



4.6. unsolvable1.txt



5. Tautan Github

https://github.com/dikyrest/15-puzzle

6. Lampiran

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