Laporan Tugas Kecil 3

IF2211 - Strategi Algoritma

Semester II Tahun Ajaran 2021/2022

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

Disusun oleh:

Amar Fadil

13520103

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

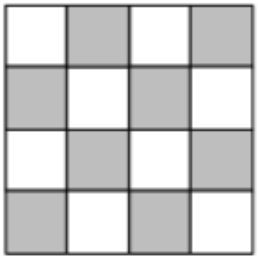
Jl. Ganesha 10, Bandung 4013

Daftar Isi

A. Algoritma Branch and Bound 15-Puzzle

Algoritma dalam mencari penyelesaian instansi permasalahan 15-puzzle disadur dari salindia Algoritma Branch and Bound (Bagian 1) untuk permainan 15 Puzzle. Terdapat dua tahap utama dalam mencari solusi 15-puzzle. Tahapan pertama (implementasi `can_solve` pada kelas `Solver` di modul `solver`) yang dilakukan program ini adalah menentukan apakah instansi persoalan dapat diselesaikan atau tidak. Terdapat fungsi `kurang(i)` yaitu banyaknya ubin bernomor *j* sedemikian rupa sehingga $j < i$ dan `posisi(j) > posisi(i)`, dimana `posisi(i)` merupakan posisi ubin bernomor *i* pada susunan yang diperiksa. Secara keseluruhan, tahapan pertama ini melakukan prosedur berikut:

1. Memanggil fungsi `bound` pada objek yang sama untuk menghitung nilai *reachable goal* total.
2. Pada fungsi `bound`, akan dihitung nilai *reachable goal* total dengan menghitung jumlah kurang(i) untuk $1 \leq i \leq 16$ ditambah dengan X (X = 1 jika sel kosong berada pada daerah yang diarsir di samping, X = 0 jika sebaliknya), atau secara matematis dituliskan:



$$\left(\sum_{i=1}^{16} kurang(i)\right) + X$$

3. Hasil nilai reachable goal total dari pemanggilan fungsi `bound` akan digunakan untuk menghitung apakah nilainya genap atau ganjil. Jika genap, maka instansi persoalan dapat diselesaikan. Jika ganjil, maka instansi persoalan tidak dapat diselesaikan.

Jika dari tahap pertama, instansi persoalan dapat diselesaikan, maka algoritma akan menjalankan tahap kedua, dimana instansi persoalan akan diselesaikan dengan *branch and bound*. Tahapan kedua (implementasi `solve` pada kelas `Solver` di modul `solver`) yang dilakukan program ini adalah sebagai berikut:

1. Masukkan simpul akar yang telah dibuat ke dalam antrian prioritas. Tandai simpul sebagai telah diperiksa.
2. Jika Q kosong, stop.
3. Jika Q tidak kosong, ambil simpul *i* dengan nilai cost yang paling kecil pada antrian. Jika ada lebih dari satu simpul *i* yang memenuhi, pilih yang paling pertama masuk antrian (menggunakan `queue.get` dan perbandingan dilakukan dengan fungsi `__lt__` pada objek node `Puzzle`).
4. Periksa apakah simpul *i* merupakan simpul solusi atau bukan (fungsi `is_solution` pada objek node `Puzzle`). Jika simpul *i* merupakan simpul solusi, maka solusi telah ditemukan, simpan simpul solusi dan stop.
5. Jika simpul *i* bukan merupakan simpul solusi, bangkitkan semua anak-anaknya dan hitung cost masing-masing anak (fungsi `calc_next` pada objek node `Puzzle`). *Cost* atau *bound* setiap anak dihitung dengan penjumlahan cost yang diperlukan untuk sampai ke suatu simpul x dari akar (atribut `depth` pada objek node `Puzzle`) dan taksiran heuristik berupa jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir atau *goal state* (fungsi `__h` pada objek node `Puzzle`).
6. Untuk setiap anak *j* dari simpul *i*, masukkan semua anak-anaknya ke dalam antrian prioritas Q jika ada sembari menambah penghitung simpul yang dibangkitkan.
7. Kembali ke langkah 2.

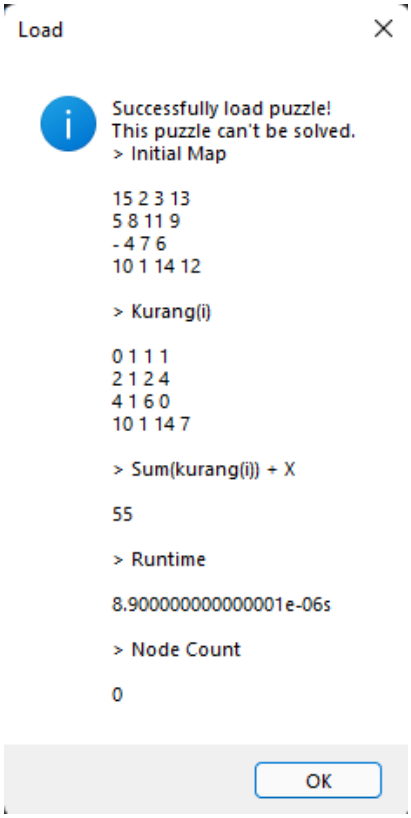
Untuk mengetahui node Puzzle mana yang telah dikunjungi (fungsi `visit` pada objek `Solve`), digunakan sebuah *dictionary* dengan key merupakan hashing dari penggabungan map menjadi

representasi string. Key ini dihitung setiap pengecekan anak yang valid saat dibangkitkan, sehingga jika terdapat key yang sama, maka simpul tersebut tidak akan menjadi anak.

Output yang ditampilkan berupa informasi apakah puzzle bisa diselesaikan, initial map, matriks kurang(i), sum(kurang(i)) + X, waktu yang dibutuhkan untuk menjalankan algoritma, dan jumlah node yang dibangkitkan. Matriks kurang(i) memiliki format sebagai berikut:

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

Dimana setiap angka i berisi nilai $kurang(i)$. Sebagai contoh, instansi puzzle ini:



Memiliki nilai kurang(i) sebagai berikut:

Kurang(1) = 0	Kurang(9) = 4
Kurang(2) = 1	Kurang(10) = 1
Kurang(3) = 1	Kurang(11) = 6
Kurang(4) = 1	Kurang(12) = 0
Kurang(5) = 2	Kurang(13) = 10
Kurang(6) = 1	Kurang(14) = 1
Kurang(7) = 2	Kurang(15) = 14
Kurang(8) = 4	Kurang(16) = 7

B. Source Code

1. puzzle.py

```
from enum import Enum
from typing import List, Tuple, Union


class MoveDirection(Enum):
    """Valid move direction

    Args:
        Enum (Enum): Enumerable type object.
    """
    UP = 1
    DOWN = 2
    RIGHT = 3
    LEFT = 4

class Puzzle:
    def __init__(self, context, last_map, action=None, last_center=None, depth=0, parent=None) ->
None:
        """Initialize new puzzle node.

        Args:
            context (Solver): The solver context for this puzzle.
            last_map (list[list]): The map before action applied.
            action (MoveDirection, optional): The direction of move action. Defaults to None.
            last_center (tuple[int,int], optional): The position of empty block. Defaults to None.
            depth (int, optional): The depth of this puzzle node. Defaults to 0.
            parent (Puzzle, optional): The puzzle parent of this node. Defaults to None.
        """
        self.context = context
        """The solver context for this puzzle."""
        self.depth = depth
        """The depth of this puzzle node."""
        self.parent:Puzzle = parent
        """The parent of this puzzle node"""
        self.children: List[Puzzle] = []
        """The childrens of this puzzle node."""
        self.map_ = [ [ last_map[i][j] for j in range(4) ] for i in range(4) ]
        """The map of this puzzle node."""
        self.action:MoveDirection = action
        """The action taken for this puzzle node."""
        self.__center:Tuple[int,int] = None
        """The position of empty block in this puzzle node."""
        if last_center is None:
            self.get_center() # If root, we find manually the empty block pos
```

```
        else: # if not root, we adjust from center point beforehand.
            self.__center = self.get_next_point(last_center, action)
            self.swap(last_center, self.__center)
        self.cost = self.depth + self.__h()
        """The cost of this puzzle node."""
        self.key = hash("|".join([
            "|".join([
                str(y)
                for y in x
            ])
            for x in self.map_
        ]))
        """The key unique id for this puzzle node."""


    def __h(self) -> int:
        """Get the heuristic cost approximation of this node.

        Cost approximation is based on the number of misplaced
        tiles in compare to the goal state (except empty tile).

        Returns:
            int: The heuristic cost approximation of this node.
        """
        cnt = 0
        for i in range(4):
            for j in range(4):
                # Exclude empty tile AND check if the tile[i][j] is misplaced
                if (i, j) != self.__center and self.map_[i][j] - 1 != i * 4 + j:
                    cnt += 1
        return cnt

    def swap(self, p1, p2):
        """Swap the number of two tiles in coord p1 and p2.

        Args:
            p1 (Tuple[int,int]): Tile 1 coordinate.
            p2 (Tuple[int,int]): Tile 2 coordinate.
        """
        temp = self.map_[p1[0]][p1[1]]
        self.map_[p1[0]][p1[1]] = self.map_[p2[0]][p2[1]]
        self.map_[p2[0]][p2[1]] = temp

    def calc_next(self) -> None:
        """Calculate the next puzzle node.
        """
        self.children.clear()
        for act in MoveDirection:
```

```
            # For every move, we check if it's in map boundary
            if self.is_valid_pos(act):
                # If it's in boundary, we create a new puzzle node
                new_puzzle = Puzzle(
                    self.context,
                    self.map_,
                    act,
                    self.__center,
                    self.depth + 1,
                    self
                )
                # Check if we already visit the puzzle.
                # If not, we append to the child.
                # If yes, we delete the new puzzle.
                if not self.context.visit(new_puzzle):
                    self.children.append(new_puzzle)
                else:
                    del new_puzzle

    def is_valid_pos(self, action:MoveDirection) -> bool:
        """Check if action returning to valid position.

        Valid position is 0 <= i <= 3 and 0 <= j <= 3.

        Args:
            action (MoveDirection): The direction move.

        Returns:
            bool: Validity of the action.
        """
        next_point = self.get_next_point(self.__center, action)
        return (
            next_point[0] >= 0 and next_point[0] < 4
            and next_point[1] >= 0 and next_point[1] < 4
        )

    def is_solution(self) -> bool:
        """Check if this node is a solution node.

        Returns:
            bool: True if node is a solution node.
        """
        for i in range(4):
            for j in range(4):
                if self.map_[i][j] - 1 != i * 4 + j:
                    return False
        return True
```

```
    @staticmethod
    def get_next_point(pfrom:Tuple[int,int], action:MoveDirection) -> Tuple[int,int]:
        """Get the next empty block position after move action.

        Will calculate the next empty block from pfrom
        based on the action that will be taken.

        Args:
            pfrom (tuple[int,int]): The last position of empty block.
            action (MoveDirection): The action that will be taken.

        Returns:
            Tuple[int,int]: The next point after action taken.
        """
        res = [pfrom[0], pfrom[1]]
        if action == MoveDirection.DOWN:
            res[0] += 1
        elif action == MoveDirection.UP:
            res[0] -= 1
        elif action == MoveDirection.LEFT:
            res[1] -= 1
        elif action == MoveDirection.RIGHT:
            res[1] += 1
        return (res[0], res[1])

    def get_center(self) -> Union[Tuple[int, int], None]:
        """Get the empty block position if any.

        Returns:
            Union[Tuple[int, int], None]: The empty block position.
        """
        if self.__center is None:
            for i in range(4):
                for j in range(4):
                    if self.map_[i][j] == 16:
                        self.__center = (i, j)
        return self.__center

    def __lt__(self, other) -> bool:
        """Lower than comparison for this object.

        Comparison is based on the node cost.

        Args:
            other (Puzzle): The other puzzle node to be compared with.

        Returns:
            bool: Is this puzzle cost lower than other puzzle cost.
```

```
        """
        return self.cost < other.cost

    def __str__(self) -> str:
        """Convert the puzzle map to string.

        Returns:
            str: String representation of the puzzle map.
        """
        return "".join([
            "\n".join([" ".join(
                map(lambda x: str(x) if x != 16 else "-", v)
            )
            for v in self.map_]),
            "\nStep {} | Action {}".format(
                self.depth,
                self.action.name
            ) if self.action else ""
        ])
```

2. solver.py

```
from queue import PriorityQueue
from time import perf_counter_ns
from FifteenPuzzleSolver.puzzle import Puzzle


class Solver:
    NONE_VALUE = 16

    def __init__(self, maps:str=None) -> None:
        """Create a new solver with maps.

        Args:
            maps (str, optional): Map to solve.
                Defaults to None.

        Raises:
            Exception: Invalid configuration map.
        """
        if maps is None: # We generate map from input
            self.map = [
                [
                    int(x) if x.isnumeric()
                    else Solver.NONE_VALUE
                    for x in input().split(' ')
                ]
                for _ in range(4)
```

```
            ]
            print("Searching for solution...")
        else: # Generate map from file
            self.map = [
                [
                    int(x) if x.isnumeric()
                    else Solver.NONE_VALUE
                    for x in s.split(' ')
                ]
                for s in maps.splitlines()
            ]
        # Check if map is valid (matrix 4x4)
        valid = len(self.map) == 4
        for i in self.map:
            if len(i) != 4:
                valid = True
        if not valid:
            raise Exception(
                "Invalid configuration map. Map is not a matrix 4 x 4."
            )
        # Reset, create root, and precalculate kurang(i), and solve
        self.reset()
        self.root = Puzzle(self, self.map)
        self.calc_kurang = [
            self.kurang(i+1)
            for i in range(8)
        ]
        self.solve()

    def kurang(self, num) -> int:
        """Kurang(i) implementation.

        Args:
            num (int): Number to calculate as i.

        Returns:
            int: Kurang(num).
        """
        cnt = 0
        for i in range(4):
            for j in range(4):
                if self.map[i][j] == num:
                    for a in range(i, 4):
                        for b in range(4):
                            if a == i and b < j:
                                continue
                            if self.map[a][b] < self.map[i][j]:
                                cnt += 1
```

```
                return cnt
        return cnt

    def visit(self, puzzle) -> bool:
        """Visit a puzzle.

        Args:
            puzzle (Puzzle): Puzzle to visit.

        Returns:
            bool: is puzzle already visited.
        """
        if puzzle.key in self.__visited:
            return True
        else:
            self.__visited[puzzle.key] = True
            return False

    def reset(self):
        """Reset solver."""
        self.runtime = 0
        self.final = None
        self.count_nodes = 0
        self.__visited = {}

    def solve(self):
        """Solve this puzzle instantiation.
        """
        # Start timer
        self.runtime = perf_counter_ns()
        # If we can solve, create prioqueue, add root
        # to the queue and visit it.
        if self.can_solve():
            queue = PriorityQueue()
            queue.put(self.root)
            self.visit(self.root)
            # We visit every node in the queue
            # until we found the solution.
            while not queue.empty():
                m:Puzzle = queue.get()
                if m.is_solution():
                    self.final = m
                    break
                m.calc_next()
                for c in m.children:
                    self.count_nodes += 1
                    queue.put(c)
        # Stop timer and calculate runtime
```

```
        self.runtime = (perf_counter_ns() - self.runtime) * 0.000000001 #in seconds

    def bound(self) -> int:
        """Get the required bound to solve the puzzle.

        Returns:
            int: sum(kurang(i)) + X.
        """
        t = self.root.get_center()
        return (
            sum(self.calc_kurang)
            + (1 if any([
                (t[i%2] % 2 == 0 and t[(i+1)%2] % 2 != 0)
                for i in range(2)
            ]) else 0)
        )

    def can_solve(self):
        """Check if we can solve this puzzle.

        The puzzle can be solved if it has even bound.

        Returns:
            bool: If this puzzle can be solved.
        """
        return (self.bound() % 2 == 0)

    @staticmethod
    def solve_path(final) -> str:
        """Traverse to the root and return the path.

        Args:
            final (Puzzle): Solution node puzzle.

        Returns:
            str: Path from root to final node.
        """
        res = ""
        if final.parent is not None:
            res += Solver.solve_path(final.parent)
        return res + str(final) + "\n\n"

    def describe(self, show_solution=False) -> str:
        """Return string representation of this solver.

        Args:
            show_solution (bool, optional): Include solution path.
                Defaults to False.
```

```
        Returns:
            str: String representation of this solver.
        """
        return "\n\n".join([
            "> Initial Map",
            str(self.root),
            "> Kurang(i)",
            "\n".join([
                " ".join([
                    str(self.calc_kurang[j+i*4])
                    for j in range(4)
                ])
                for i in range(4)
            ]),
            "> Sum(kurang(i)) + X",
            str(self.bound()),
            (
                "> Solution\n\n" + (
                    "No solution found" if not self.can_solve()
                    else "Solution found!\n" + self.solve_path(self.final)
                ) + "\n> Runtime"
            ) if show_solution else "> Runtime",
            str(self.runtime) + "s",
            "> Node Count",
            str(self.count_nodes)
        ])

    def __str__(self) -> str:
        """Return string representation of this solver.

        Also includes the solution path.

        Returns:
            str: String representation with solution path.
        """
        return self.describe(True)
```

3. visualizer.py [GUI]

```
import os
from tkinter import NORMAL, DISABLED, ttk, messagebox, Tk, StringVar, IntVar
from typing import List

from FifteenPuzzleSolver.solver import Solver
from src.FifteenPuzzleSolver.puzzle import Puzzle


class Visualizer(Tk):
```

```
    def __init__(self, solver=None) -> None:
        """Initialize a new visualizer.

        Args:
            solver (Solver, optional): Solver instantiation.
                Defaults to None.
        """
        super().__init__()
        self.resizable(0, 0)
        self.title("15-Puzzle Solver")

        # create input btn
        frame1 = ttk.Frame(self)
        frame1.rowconfigure(0, weight=4)
        # inp file
        self.filename = StringVar()
        inp_file = ttk.Entry(frame1, textvariable=self.filename)
        inp_file.grid(row=0, column=0, columnspan=3)
        # btn apply
        btn_apply = ttk.Button(frame1, text="Load", command=self.apply)
        btn_apply.grid(row=0, column=3)
        # btn solve
        self.btn_solve = ttk.Button(frame1, text="Solve", command=self.solve)
        self.btn_solve.grid(row=1, column=0, columnspan=3, sticky="ew")
        self.btn_solve.config(state=DISABLED)
        # btn reset
        self.btn_reset = ttk.Button(frame1, text="Reset", command=self.reset)
        self.btn_reset.grid(row=1, column=3)
        self.btn_reset.config(state=DISABLED)

        frame_slider = ttk.Frame(self)
        # slider label
        slider_label = ttk.Label(frame_slider, text="Speed")
        slider_label.grid(row=2, column=0, sticky="ew", ipadx=5)
        # slider value
        self.anim_speed = IntVar(value=50)
        self.slider_speed = ttk.Scale(frame_slider, from_=50, to=500,
            variable=self.anim_speed, orient="horizontal",
            command=lambda _: slider_val_lbl.config(text=str(self.anim_speed.get()) + "ms"))
        self.slider_speed.grid(row=2, column=1, columnspan=2, sticky="ew")
        # slider value label
        slider_val_lbl = ttk.Label(frame_slider, text="50ms")
        slider_val_lbl.grid(row=2, column=3, sticky="ew", ipadx=5)

        # create grid frame
        frame2 = ttk.Frame(self, relief="sunken", borderwidth=2)
        self.grid_puzzle:List[ttk.Label] = []
        for i in range(4):
```

```
            for j in range(4):
                g = ttk.Label(frame2, text=str(i*4+j+1), style="Puzzle.TLabel")
                g.grid(row=i, column=j)
                self.grid_puzzle.append(g)
        self.grid_puzzle[15].grid_remove()

        # style the grid
        grid_style = ttk.Style(frame2)
        grid_style.configure("Puzzle.TLabel",
            font=("Arial", 20),
            background="#aeb9c8",
            foreground="#000",
            width=3,
            borderwidth=5,
            relief="raised",
            anchor="center",
        )

        # show frame
        frame1.grid(padx=5, pady=5)
        frame_slider.grid()
        frame2.grid(padx=5, pady=5)

        self.after_list = []
        self.solver = None

        if solver:
            self.solver = solver
            self.load_solver()

        self.mainloop()

    def load_solver(self) -> None:
        """Load the solver instance.
        """
        messagebox.showinfo("Load","\n".join([
            "Successfully load puzzle!",
            "{}",
            "{}",
        ]).format(
            "This puzzle can be solved!"
            if self.solver.can_solve() else
            "This puzzle can't be solved.",
            self.solver.describe()
        ))
        if self.solver.can_solve():
            self.btn_solve["state"] = NORMAL
            self.btn_reset["state"] = NORMAL
```

```
        else:
            self.btn_solve["state"] = DISABLED
            self.btn_reset["state"] = DISABLED
        self.update(self.solver.root.map_)

    def apply(self) -> None:
        """Apply text in input file.
        """
        try:
            with open(self.filename.get(), "r") as f:
                self.solver = Solver(f.read())
                self.load_solver()
        except FileNotFoundError:
            messagebox.showerror(
                "Error",
                "File not found! Current working directory: {}"
                .format(os.getcwd())
            )

    def update(self, cur_map) -> None:
        """Update the grid with map.

        Args:
            cur_map (list[list[int]]): Map to update.
        """
        for i in range(4):
            for j in range(4):
                g = self.grid_puzzle[i*4+j]
                if g["text"] == "16": # bring back last tile 16
                    g.grid(row=i, column=j)
                if cur_map[i][j] == 16: # hide current tile 16
                    g.grid_remove()
                g["text"] = str(cur_map[i][j])

    def update_after(self, depth, map):
        """Update from after coroutine.

        Args:
            depth (int): The depth of this map.
            map (list[list[int]]): The map to update.
        """
        self.update(map)
        if depth == self.solver.final.depth:
            # We update gui to normal after reaching solution.
            self.btn_solve.config(state=NORMAL)
            self.slider_speed.config(state=NORMAL)

    def traverse(self, depth, cur_state:Puzzle):
```

```
        """Traverse the puzzle in a delay animation.

        It will continously traverse parent from child
        first, then update the grid from the parent to
        child by increasing the delay time as it
        reaches the final node.

        Args:
            depth (int): Current depth of the traverse.
            cur_state (Puzzle): Current puzzle to traverse.
        """
        if cur_state.parent is not None:
            self.traverse(depth-1, cur_state.parent)
        self.after_list.append(self.after(
            self.anim_speed.get()*(depth),
            lambda: self.update_after(depth, cur_state.map_)
        ))

    def solve(self) -> None:
        """Start the solve animation.
        """
        # Disable buttons
        self.btn_solve.config(state=DISABLED)
        self.slider_speed.config(state=DISABLED)
        # Stop any after coroutine if available and clear
        for i in self.after_list:
            self.after_cancel(i)
        self.after_list.clear()
        # Start traversing and animating the solution
        self.traverse(self.solver.final.depth, self.solver.final)

    def reset(self) -> None:
        """Reset the solve animation.
        """
        # Enable buttons
        self.btn_solve.config(state=NORMAL)
        self.slider_speed.config(state=NORMAL)
        # Stop any after coroutine if available and clear
        for i in self.after_list:
            self.after_cancel(i)
        self.after_list.clear()
        # Reset the grid to initial root
        self.update(self.solver.root.map_)

if __name__ == "__main__":
    v = Visualizer()
```

4. __main__.py [CLI Main Driver]

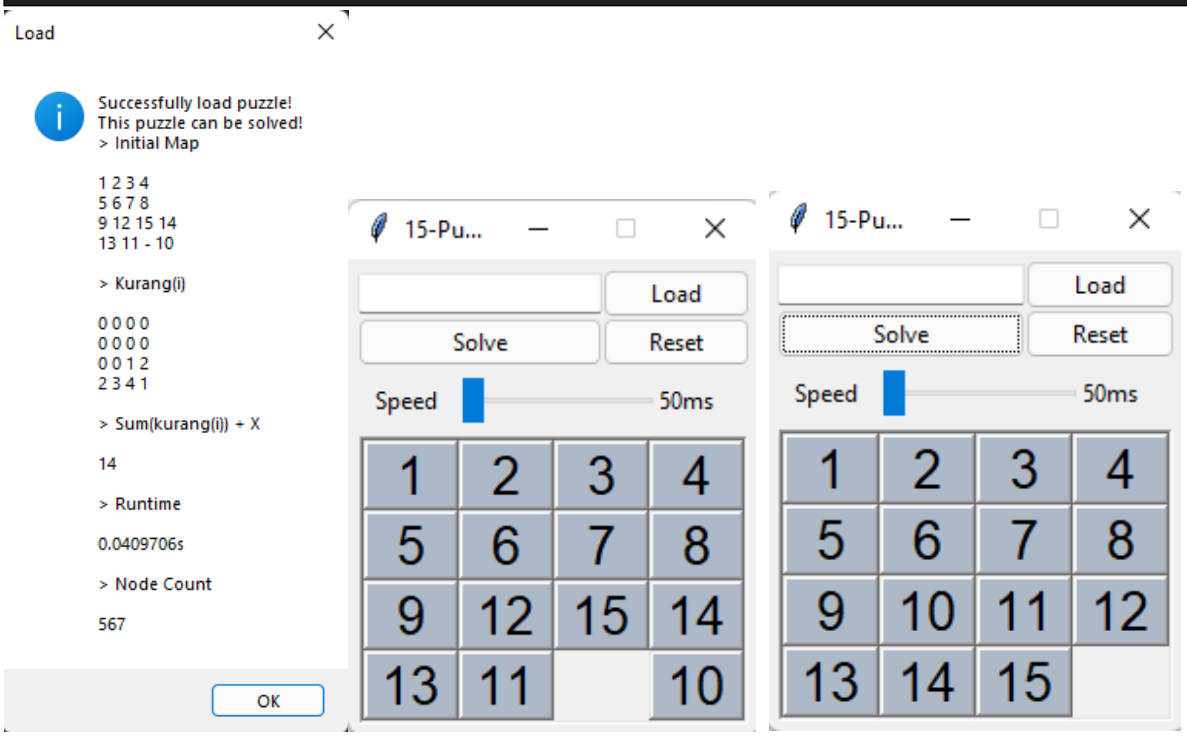
```
import argparse
import os

from FifteenPuzzleSolver.solver import Solver

# Argument Parser
parser = argparse.ArgumentParser(
    description=' '.join([
        'Main driver of Fifteen Puzzle Solver.',
        'It will generate a solution path for the problem instantiation.',
        'You can supply manually the initial state of the puzzle to show in GUI by specify -i/--input
and -g/--gui.',
    ]),
)
parser.add_argument('-f', '--file', help='input file path.')
parser.add_argument('-g', '--gui', help='show GUI visualizer.', action='store_true')
parser.add_argument('-i', '--input', help='get puzzle from input.', action='store_true')
args = parser.parse_args()

# Get solver
solver = None
if args.file: # if file is specified, load solver from it
    try:
        with open(args.file, 'r') as f:
            solver = Solver(f.read())
    except FileNotFoundError:
        parser.error('File not found! Current working directory: {}'.format(os.getcwd()))
elif not args.gui or args.input: # if it's not GUI or input stdin enabled, ask the input map.
    print('Please enter the matrix instantiation of the problem.')
    print('Format Example:')
    print()
    print('3 1 2 4')
    print('- 5 7 8')
    print('10 6 11 12')
    print('9 13 14 15')
    print()
    print('NOTE: You can use any non numeric string or number 16 as the blank space.')
    print()
    solver = Solver()

if args.gui: # If gui, show the gui
    # Lazy load the gui
    from FifteenPuzzleSolver.visualizer import Visualizer
    Visualizer(solver)
else: # if not gui, just print the solver state with the solution
    print(solver)
```

C. Screenshot

1. berhasil1.txt

Test case ini dijalankan dengan perintah berikut:

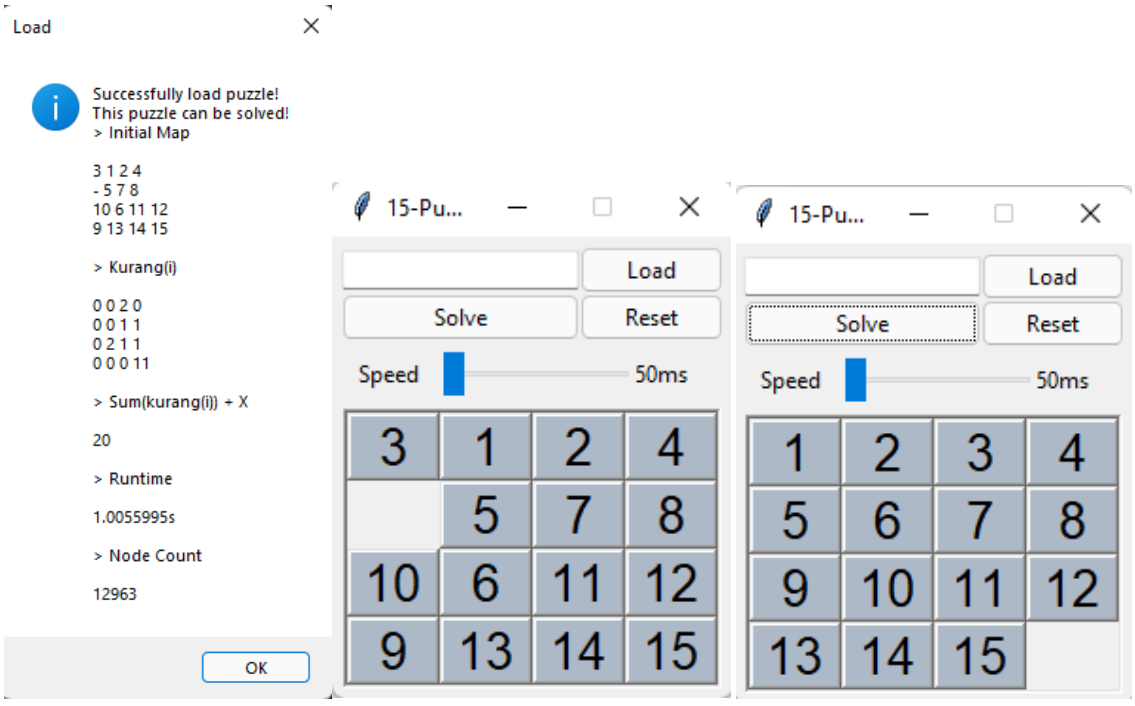
```
python -m FifteenPuzzleSolver -f test/berhasil1.txt -g
```



2. berhasil2.txt

Test case ini dijalankan dengan perintah berikut:

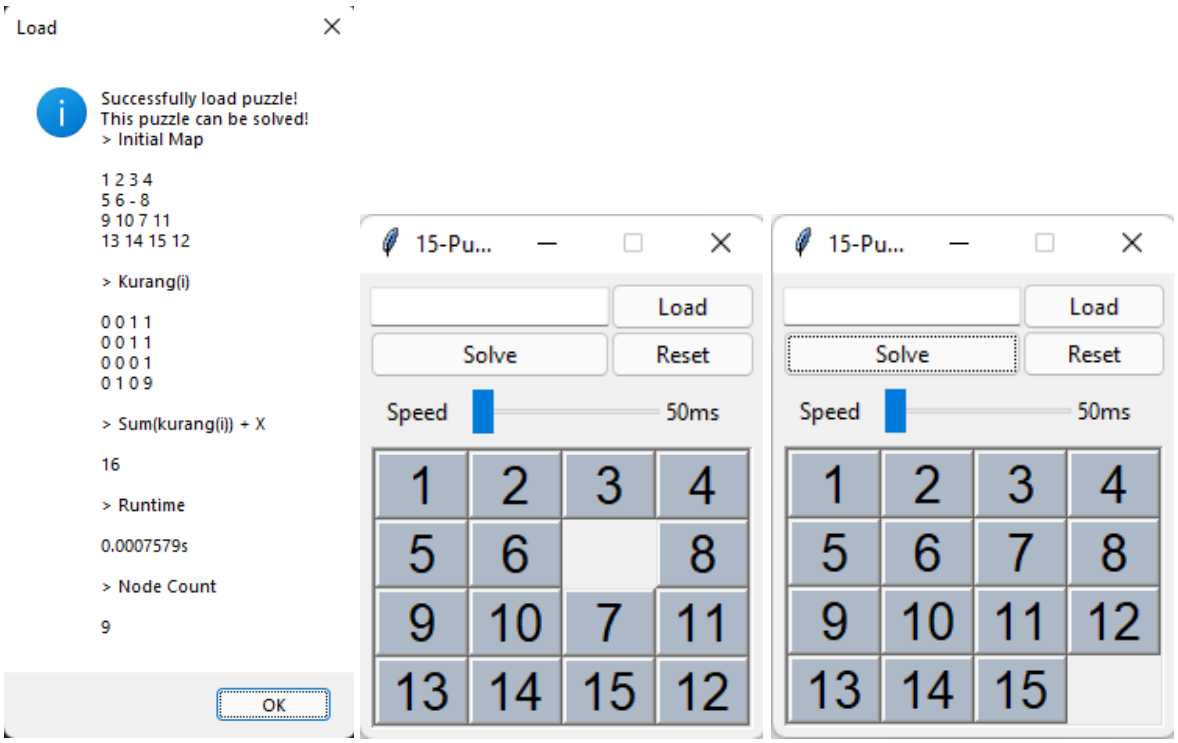
```
python -m FifteenPuzzleSolver -f test/berhasil2.txt -g
```



3. berhasil3.txt

Test case ini dijalankan dengan perintah berikut:

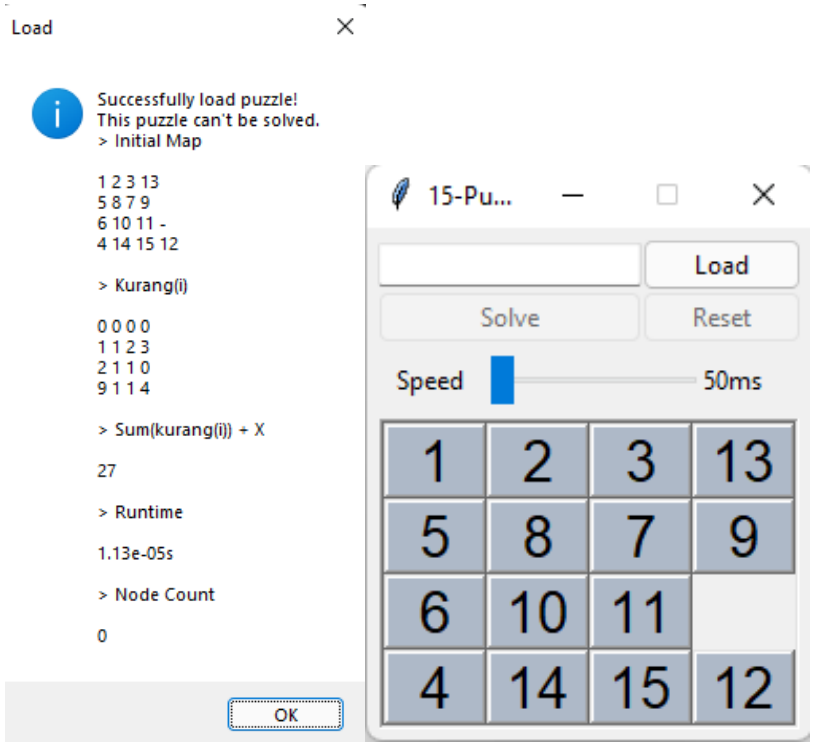
```
python -m FifteenPuzzleSolver -f test/berhasil3.txt -g
```



4. gagal1.txt

Test case ini dijalankan dengan perintah berikut:

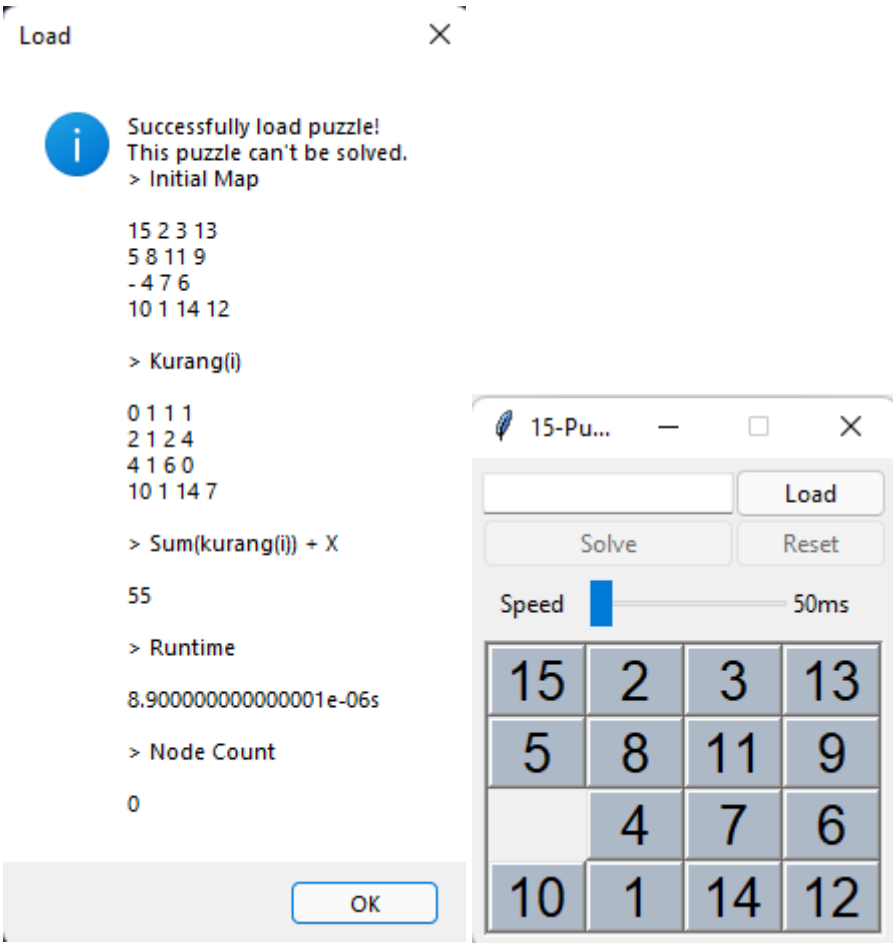
```
python -m FifteenPuzzleSolver -f test/gagal1.txt -g
```



5. gagal2.txt

Test case ini dijalankan dengan perintah berikut:

```
python -m FifteenPuzzleSolver -f test/gagal2.txt -g
```



Lampiran

Assistant Checklist

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.	√	

Link Repository
https://github.com/marfgold1/Puzzle15Solver

berhasil1.txt
1 2 3 4
5 6 7 8
9 12 15 14
13 11 - 10

berhasil2.txt
3 1 2 4
- 5 7 8
10 6 11 12
9 13 14 15

berhasil3.txt
1 2 3 4
5 6 - 8
9 10 7 11
13 14 15 12

gagal1.txt
1 2 3 13
5 8 7 9
6 10 11 -
4 14 15 12

gagal2.txt
15 2 3 13
5 8 11 9
- 4 7 6
10 1 14 12