Implementasi Algoritma *Branch and Bound* untuk Penyelesaian Permainan 15-Puzzle

LAPORAN TUGAS KECIL

Diajukan Untuk Memenuhi Tugas IF2211 Strategi Algoritma
Semester II Tahun 2021/2022

Disusun oleh

Andika Naufal Hilmy (13520098)

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG

2022

A. ALGORITMA DIVIDE AND CONQUER

Algoritma *branch and bound* adalah sebuah algoritma pencarian yang bersifat melebar dengan membangkitkan simpul baru dari simpul yang memiliki biaya terkecil. Secara teknis, algoritma ini mirip dengan algoritma BFS, namun pembangkitan simpul barunya diprioritaskan pada simpul dengan biaya terkecil. Biaya dari sebuah simpul didefinisikan sebagai perkiraan biaya termurah dalam lintasan. Akibatnya, nilai biaya dari sebuah simpul dihitung secara heuristik dengan rumus berikut:

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

dengan $\hat{c}(P)$ adalah biaya untuk simpul P, $f(P)$ adalah panjang lintasan dari simpul induk sampai P, dan $\hat{g}(P)$ adalah biaya heuristik simpul terpendek dari P sampai simpul solusi berupa jumlah dari banyak petak tak kosong yang terletak setelah petak tersebut dengan nilai yang lebih kecil dari petak yang ditunjuk.

Algoritma *Branch and bound* akan diterapkan untuk mencari solusi dari sebuah 15-puzzle dengan keadaan acak. Langkah penerapan ini dimulai dengan membuat sebuah simpul dasar yang memuat kondisi puzzle sementara dan panjang lintasan dari simpul akar. Sebagai kelas, puzzle juga memiliki metode untuk menggeser petak kosong, mendapatkan biaya dari konfigurasi puzzle, dan menentukan apakah sebuah puzzle bisa diselesaikan atau tidak. Di sini, petak kosong adalah petak dengan nomor 16. Setiap simpul yang dibangkitkan akan dimasukkan ke sebuah *priority queue* dengan prioritas biaya terkecil.

Untuk mencegah galat puzzle di luar batas, maka hanya langkah valid yang akan ditambahkan di simpul. Misalnya, pada program ini, jika posisi petak kosong berada di sisi paling kanan, maka langkah ke kanan tidak akan dibangkitkan.

Secara garis besar, langkah algoritma yang diimplementasikan adalah seperti berikut:

1. Masukkan simpul dasar ke antrean *priority queue*. Jika simpul akar adalah solusi, pencarian dihentikan.
2. Jika *priority queue* kosong, berhenti.
3. Jika *priority queue* tidak kosong, pilih simpul *i* dengan biaya terkecil.
4. Jika simpul *i* adalah solusi, berhenti.
5. Jika simpul *i* bukan solusi, bangkitkan semua simpul baru untuk langkah atas, bawah, kiri, dan kanan dengan syarat langkah yang dilakukan harus valid.
6. Untuk setiap simpul baru yang dibangkitkan, hitung biayanya dan masukan semua simpul baru tersebut dengan membandingkan biaya dari simpul tersebut untuk peletakan simpul di *priority queue*.

B. KODE PROGRAM

Secara garis besar, program dibuat secara modular dengan sepuluh modul. Semua modul dijadikan satu di folder *src*, termasuk modul *main.py* yang berfungsi sebagai program utama.

main.py

Modul program utama untuk antarmuka CLI.

```python
import menu
import process

if __name__ == "__main__":
    # repeat until input menu is correct
    while True:
        # select input menu
        input_menu = menu.select_input_menu()
        # if input menu is 1
        if input_menu == "1":
            # insert file name dialog
            filename = menu.insert_file_name_dialog()
            # load file
            np_array = menu.load_file(filename)
            # if file is OK
            if np_array is not None:
                # print numpy array
                print(np_array)
                # go to main process
                process.main(np_array)
        # if input menu is 2
        elif input_menu == "2":
            # insert input string from keyboard
            input_str = menu.insert_from_keyboard_dialog()
            # convert string to numpy array
            np_array = menu.convert_to_np_array(input_str)
            # print numpy array
            print(np_array)
            # go to main process if np_array not none
            if np_array is not None:
                process.main(np_array)
        # if input menu is 3
        elif input_menu == "3":
            # exit
            break
        # if input menu is not 1, 2 or 3
        else:
            # print error
            print("Error: invalid input")
```

menu.py

Modul berisi prosedur dan fungsi esensial untuk membantu proses pada antarmuka program secara garis besar.

```
import numpy as np
import os


def select_input_menu():
    """
    Select input menu
    """
    print("\nSelect input menu")
    print("1. Input from file")
    print("2. Input from keyboard")
    print("3. Exit")
    return input("Select: ")


def insert_file_name_dialog():
    """
    Insert file name dialog
    """
    print("\nInsert your file name")
    return input("File name: ")


def examine(filename):
    # Required: file must be a txt file
    # Required: file cannot be empty
    # Required: file exists

    # Check if file exists
    if not os.path.isfile(filename):
        print("File does not exist")
        return False

    # Check if file is a txt file
    if not filename.endswith(".txt"):
        print("File is not a txt file")
        return False

    # Check if file is empty
    if os.stat(filename).st_size == 0:
        print("File is empty")
        return False
```

```
    return True


def load_file(filename):
    # examine filename using function examine(filename)
    # if file is OK, load it
    if examine(filename):
        # open file
        file = open(filename, "r")
        # read file
        string = file.read()
        # close file
        file.close()
        # convert string to numpy array
        np_array = convert_to_np_array(string)
        # return numpy array
        return np_array
    else:
        return None


def insert_from_keyboard_dialog():
    """
    Insert from keyboard dialog
    """
    print("\nInsert your puzzle")
    print("Input format: int1 int2 int3 ... int16")
    print("Example: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16\n")
    return input("Puzzle: ")


def examine_array(string):
    # assume the given input string is a line of numbers separated by space
    # convert the given string to list of integers
    # and then check if list length is 16 and contains unique numbers
    # from 1 to 16, number order is not mandatory
    # return True if all requirements are met
    # return False otherwise

    # convert string to list of integers
    list_of_int = [int(i) for i in string.split()]

    # check if list length is 16
    if len(list_of_int) != 16:
        print("List length is not 16")
        return False

    # check if list contains unique numbers
    if len(list_of_int) != len(set(list_of_int)):
```

```
        print("List contains duplicate numbers")
        return False

    # check if list contains numbers from 1 to 16
    if not all(i in range(1, 17) for i in list_of_int):
        print("List contains numbers not from 1 to 16")
        return False

    return True


def convert_to_np_array(string):
    # the previous input string is to be converted
    # to a numpy array with size 4x4
    # return the numpy array
    # if the input string is not valid, return None

    # check the string using function examine_array
    if not examine_array(string):
        return None

    # convert string to list of integers
    list_of_int = [int(i) for i in string.split()]

    # convert list to numpy array
    np_array = np.array(list_of_int, dtype=np.int8)

    # reshape the numpy array
    np_array = np_array.reshape(4, 4)

    return np_array
```

moves.py

Modul enumerasi perpindahan petak kosong pada puzzle.

```
import enum


class Moves(enum.Enum):
    # DEFINITION
    "Base enumeration class for moves"

    # ENUMERATIONS
    UP = 1
    DOWN = 2
    LEFT = 3
```

```
    RIGHT = 4
```

position.py

Modul kelas Position yang berfungsi untuk menentukan dan mengecek posisi dari petak kosong.

```
class Position:
    # DEFINITION
    "Base class for position"

    # attributes
    row = 0
    col = 0

    # constructors
    def __init__(self, row=0, col=0):
        "Constructor"

        self.row = row
        self.col = col

    # methods
    def get_row(self):
        "Returns row"

        return self.row

    def get_col(self):
        "Returns column"

        return self.col

    def set_row(self, row):
        "Sets row to row (int)"

        self.row = row

    def set_col(self, col):
        "Sets col to col (int)"

        self.col = col

    def is_row_valid(self, row_size):
        "Checks if position is valid in row"

        return self.row < row_size and self.row >= 0
```

```
    def is_col_valid(self, col_size):
        "Checks if position is valid in column"

        return self.col < col_size and self.col >= 0

    def is_out_of_bounds(self, row_size, col_size):
        "Checks if position is out of boundary"

        return not (self.is_row_valid(row_size) and self.is_col_valid(col_size))

    def is_top_most(self):
        "Checks if position is topmost"

        return self.row == 0

    def is_bottom_most(self, row_size):
        "Checks if position is bottommost"

        return self.row == row_size - 1

    def is_left_most(self):
        "Checks if position is leftmost"

        return self.col == 0

    def is_right_most(self, col_size):
        "Checks if position is rightmost"

        return self.col == col_size - 1
```

process.py

Modul yang berisi fungsi dan prosedur untuk meringkas alur program utama yang berkaitan dengan algoritma *branch and bound*.

```
from puzzle import Puzzle
from puzzlenode import PuzzleNode
from collections import deque
import search
import time
import menu


def make_base_puzzle(pz_array):
    # return new Puzzle class from a numpy array
    return Puzzle(pz_array)
```

```
def make_base_tree(pz_array):
    # return new PuzzleNode class from a numpy array
    return PuzzleNode(make_base_puzzle(pz_array))


def check_solvable(puzzle):
    # check if a Puzzle class is solvable
    # return True if solvable
    # return False otherwise
    # assume the puzzle is a valid Puzzle class

    # using method is_solvable()
    return puzzle.is_solvable()


def print_solvability_msg(puzzle):
    # print solvability message
    # assume the puzzle is a valid Puzzle class

    # using method get_solvable_message
    print("This puzzle is " + puzzle.get_solvable_message())


def solvability_est_value(puzzle):
    # get solvable estimate from the puzzle
    return puzzle.get_solvable_estimate()


def print_count_smaller(puzzle):
    # print count of smaller values
    # assume the puzzle is a valid Puzzle class

    # do it traversally from tile number 1 to tile number 15
    for i in range(1, 16):
        # print the number of tiles whose number is smaller than i
        print("kurang(" + str(i) + "): "
              + str(puzzle.count_smaller(i)))


def get_solution(base_tree):
    # get solution from given puzzle node
    # return as stack of nodes

    # check if puzzle is already solved
    # if solved, generate solution with just the node
    # otherwise, generate solution with the node and its children
    if base_tree.puzzle.is_solved():
        return deque([base_tree])
```

```
    else:
        return search.get_solution_stack(base_tree)


def print_solution(solution):
    # assume given solution is a valid stack of nodes
    step = 0
    # print solution
    print("Solution:")
    # assume logging is enabled
    log = "Solution:\n"
    while solution:
        node = solution.pop()
        print("Step " + str(step))
        log += "Step " + str(step) + "\n"
        node.print()
        log += node.get_printed_text()
        print()
        step += 1
    print("** End of solution **")
    log += "\n** End of solution **"
    return log


def main(pz_array):
    # main process of pz_array solving
    # assume the pz_array is a valid numpy array

    # create base tree from pz_array
    base_tree = make_base_tree(pz_array)

    # print base tree
    print("Base tree:")
    base_tree.print()

    # print solvability message
    print_solvability_msg(base_tree.puzzle)

    # give space
    print()

    # for each tile number 1 to 15, print tile with smaller values
    print("Value of kurang(i) for each non-empty tile\n")
    print_count_smaller(base_tree.puzzle)

    # give space
    print()

    # print solvability estimate values
```

```
    print(
        "Estimate values (sum(kurang(i) + X)): "
        + str(solvability_est_value(base_tree.puzzle))
        )

    # print solvability message
    print_solvability_msg(base_tree.puzzle)

    # do searching process if puzzle is solvable
    # otherwise print termination message
    # then terminate the search process
    if check_solvable(base_tree.puzzle):
        # start calculating execution time
        start_time = time.time()
        solution, node_count = get_solution(base_tree)
        # end calculating execution time
        end_time = time.time()
        solution_log = print_solution(solution)
        print("Number of nodes (Simpul dibangkitkan): " + str(node_count))
        solution_log += "\nNumber of nodes (Simpul dibangkitkan): " +
str(node_count)
        # print execution time
        print("Execution time: " + str(end_time - start_time) + " seconds")
        solution_log += "\nExecution time: " + str(end_time - start_time) + "
seconds"
        # Enable logging prompt question
        print("Do you want to save the solution to a file? (Y/n)")
        while True:
            # get input
            input_str = input()
            # if input is Y or y
            if input_str.lower() == "y" or input_str.lower() == "":
                # insert file name dialog
                filename = menu.insert_file_name_dialog()
                # default file name is result.txt if filename is empty
                if filename == "":
                    filename = "result.txt"
                # write solution to file
                with open(filename, "w") as f:
                    f.write(solution_log)
                # print termination message
                print("Solution saved to " + filename)
                # break
                break
            # if input is n or N
            elif input_str.lower() == "n":
                # break
                break
            # if input is not Y or y or n or N
```

```
            else:
                # print error
                print("Error: invalid input")
        # reset static attribute node_count to zero
        PuzzleNode.node_count = 0
    else:
        print("Termination message:")
        print("This puzzle is not solvable")
```

puzzle.py

Modul yang berisi representasi dari kelas Puzzle sebagai dasar pembangun puzzle beserta keadaan puzzle tersebut

```
# ATTENTION
# numpy is required for this file

import numpy as np
import position as pos
from moves import Moves

# CONSTANTS
BASE_ARRAY = np.array(
    [
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
        [13, 14, 15, 16]
    ],
    dtype=np.int8
    )


class Puzzle:
    # DEFINITION
    "Base class for puzzle"

    # ATTRIBUTE
    # private puzzle: list of list of int
    # private steps: int
    # private last_move: enum Moves

    # CONSTRUCTORS
    def __init__(self, puzzle=BASE_ARRAY, steps=0, last_move=""):
        self.puzzle = puzzle
        self.steps = steps
        self.last_move = last_move
```

```
    # METHODS
    def get_puzzle(self):
        "Returns puzzle"

        return self.puzzle

    def get_steps(self):
        "Returns steps"

        return self.steps

    def get_last_move(self):
        "Returns last move"

        return self.last_move

    def clone(self):
        "Clones puzzle"

        return Puzzle(self.puzzle.copy(), self.steps, self.last_move)

    def to_array(self):
        "Converts puzzle to array"

        return self.puzzle.copy()

    def to_string(self):
        "Converts puzzle to string"

        return str(self.puzzle)

    def print(self):
        "Prints puzzle"

        result = ""
        for i in range(4):
            result += "+----+----+----+----+\n"
            for j in range(4):
                result += (
                    "|"
                    + (
                        "{:3d} ".format(self.puzzle[i][j])
                        if self.puzzle[i][j] != 16 else "    "
                        )
                    )
            result += "|\n"
        result += "+----+----+----+----+"
        print(result)
```

```
    def get_printed_text(self):
        "Returns printed text"

        result = ""
        for i in range(4):
            result += "+----+----+----+----+\n"
            for j in range(4):
                result += (
                    "|"
                    + (
                        "{:3d} ".format(self.puzzle[i][j])
                        if self.puzzle[i][j] != 16 else "    "
                        )
                    )
            result += "|\n"
        result += "+----+----+----+----+"
        return result

    def print_state(self):
        "Prints steps with last move"

        print("Steps: " + str(self.steps))
        print("Last move: " + str(self.last_move))

    def is_solved(self):
        "Checks if puzzle is solved"

        return np.array_equal(self.puzzle, BASE_ARRAY)

    def get_tile_position(self, number):
        "Returns position of tile"

        for i in range(4):
            for j in range(4):
                if self.puzzle[i][j] == number:
                    return pos.Position(i, j)
        raise ValueError("Tile not found")

    def get_tile_number(self, position):
        "Returns number of tile"

        return self.puzzle[position.get_row()][position.get_col()]

    def get_tile_position_16(self):
        "Returns position of tile 16"

        return self.get_tile_position(16)
```

```
    def swap_tile(self, pos1, pos2):
        "Swaps two tiles"

        temp = self.puzzle[pos1.get_row()][pos1.get_col()]
        self.puzzle[pos1.get_row()][pos1.get_col()] =
self.puzzle[pos2.get_row()][pos2.get_col()]
        self.puzzle[pos2.get_row()][pos2.get_col()] = temp

    def move_up(self):
        "Moves tile up"

        pos1 = self.get_tile_position(16)
        pos2 = pos.Position(pos1.get_row() - 1, pos1.get_col())
        self.swap_tile(pos1, pos2)
        self.steps += 1
        self.last_move = Moves.UP

    def move_down(self):
        "Moves tile down"

        pos1 = self.get_tile_position(16)
        pos2 = pos.Position(pos1.get_row() + 1, pos1.get_col())
        self.swap_tile(pos1, pos2)
        self.steps += 1
        self.last_move = Moves.DOWN

    def move_left(self):
        "Moves tile left"

        pos1 = self.get_tile_position(16)
        pos2 = pos.Position(pos1.get_row(), pos1.get_col() - 1)
        self.swap_tile(pos1, pos2)
        self.steps += 1
        self.last_move = Moves.LEFT

    def move_right(self):
        "Moves tile right"

        pos1 = self.get_tile_position(16)
        pos2 = pos.Position(pos1.get_row(), pos1.get_col() + 1)
        self.swap_tile(pos1, pos2)
        self.steps += 1
        self.last_move = Moves.RIGHT

    def count_smaller(self, number):
        """
        Returns number of tiles whose number is smaller than given number
        that placed after the tile with the given number
        """
```

```
        count = 0
        # flatten the whole puzzle
        flat_puzzle = self.puzzle.flatten()
        # get tile position that have the given number
        pos = 0
        while flat_puzzle[pos] != number:
            pos += 1
        # count the number of tiles whose number is smaller than given number
        # that placed after tile with the given number
        for i in range(pos + 1, len(flat_puzzle)):
            if flat_puzzle[i] < number:
                count += 1

        return count

    # function count_all_smaller
    # returns number of tiles whose number is smaller than given number
    # process is done traversally from first element to last element
    # complexity is O(n**2)
    def count_all_smaller(self):
        # flatten the whole puzzle
        flat_puzzle = self.puzzle.flatten()
        # for each pass, get tile number from current element
        # and count number of tiles whose number is smaller than it
        # that placed after the current element
        count = 0
        for i in range(len(flat_puzzle)):
            for j in range(i + 1, len(flat_puzzle)):
                if flat_puzzle[i] > flat_puzzle[j]:
                    count += 1
        return count

    # function tile_16_estimate
    # returns (row + col) mod 2 of tile numbered 16
    # just get the position of tile numbered 16
    def tile_16_estimate(self):
        pos_16 = self.get_tile_position(16)
        return (pos_16.get_row() + pos_16.get_col()) % 2

    def get_solvable_estimate(self):
        "Returns solvable estimate"

        return self.count_all_smaller() + self.tile_16_estimate()

    def is_solvable(self):
        "Checks if puzzle is solvable"

        return self.get_solvable_estimate() % 2 == 0
```

```
    def get_solvable_message(self):
        "Returns solvable message"

        if self.is_solvable():
            return "Solvable"
        else:
            return "Not solvable"

    # Heuristic cost is defined of how many tiles displaced from goal state
    def get_heuristic_cost(self):
        "Returns heuristic cost"

        count = 0
        flat_puzzle = self.puzzle.flatten()
        for i in range(len(flat_puzzle)):
            if flat_puzzle[i] != BASE_ARRAY.flatten()[i]:
                count += 1
        return count

    def get_total_cost(self):
        return self.steps + self.get_heuristic_cost()
```

puzzlecomparator.py

Modul pembanding puzzle untuk membantu peletakan puzzle pada *priority queue*.

```
def compare(pznode1, pznode2):
    "Compares two puzzles"

    if pznode1.puzzle.get_total_cost() < pznode2.puzzle.get_total_cost():
        return -1
    elif pznode1.puzzle.get_total_cost() > pznode2.puzzle.get_total_cost():
        return 1
    else:
        # compare steps
        if pznode1.puzzle.steps < pznode2.puzzle.steps:
            return -1
        elif pznode1.puzzle.steps > pznode2.puzzle.steps:
            return 1
        else:
            return 0
```

puzzlenode.py

Modul yang berisi representasi dari kelas PuzzleNode sebagai simpul yang dibangkitkan dari objek Puzzle. Kelas PuzzleNode berfungsi sebagai komponen utama dari pencarian solusi berdasarkan algoritma *branch and bound*.

```
from puzzle import Puzzle


class PuzzleNode:
    # DEFINITION
    "Tree data structure for puzzle"

    # ATTRIBUTES
    # private puzzle: Puzzle
    # private parent: PuzzleNode
    # private children: list of PuzzleNode
    # private id: int (unique)

    # STATIC VALUE
    node_count = 0

    # CONSTRUCTORS
    def __init__(self, puzzle=Puzzle(), parent=None, children=[]):
        self.puzzle = puzzle
        self.parent = parent
        self.children = children
        self.id = PuzzleNode.apply_id()

    # METHODS
    def apply_id():
        "Applies id"

        PuzzleNode.node_count += 1
        return PuzzleNode.node_count

    def get_puzzle(self):
        "Returns puzzle"

        return self.puzzle

    def get_parent(self):
        "Returns parent"

        return self.parent

    def get_children(self):
        "Returns children"
```

```
        return self.children

    def set_parent(self, parent):
        "Sets parent"

        self.parent = parent

    def add_child(self, child):
        "Adds child"

        self.children.append(child)
        child.set_parent(self)

    def print(self):
        "Prints puzzle"

        self.puzzle.print()
        print("Puzzle ID: " + str(self.id))
        print("Parent: " + str(self.parent))
        print("Children: " + str(self.children))
        print("Last move: " + str(self.puzzle.get_last_move()))

    def get_printed_text(self):
        "Returns printed text"

        return (
            self.puzzle.get_printed_text()
            + "\nPuzzle ID: "
            + str(self.id)
            + "\nParent: "
            + str(self.parent)
            + "\nChildren: "
            + str(self.children)
            + "\nLast move: "
            + str(self.puzzle.get_last_move())
            )

    def clone(self):
        "Clones puzzle"

        return PuzzleNode(self.puzzle.clone(), self.parent, self.children)
```

puzzleprioqueue.py

Modul yang merepresentasikan *priority queue* yang dibuat menyesuaikan dengan kebutuhan algoritma *branch and bound*.

```python
# from puzzle import Puzzle
from puzzlecomparator import compare


class PuzzlePrioQueue():
    # DEFINITION
    "Priority queue for puzzle"

    # ATTRIBUTES
    # private queue: list of PuzzleNode

    # CONSTRUCTORS
    def __init__(self):
        self.queue = []

    # METHODS

    # add puzzle with priority
    # puzzle with smaller cost goes in the 1st priority
    def add(self, puzzlenode):
        "Adds puzzle to queue"

        pz_idx = len(self.queue)
        self.queue.append(puzzlenode)
        while (
            pz_idx > 0
            and (
                compare(self.queue[pz_idx], self.queue[pz_idx - 1]) < 0
                )
                ):
            temp = self.queue[pz_idx - 1]
            self.queue[pz_idx - 1] = self.queue[pz_idx]
            self.queue[pz_idx] = temp
            pz_idx -= 1

    def clear(self):
        "Clears queue"

        self.queue = []

    def contains(self, puzzlenode):
        "Returns true if queue contains puzzle"

        return puzzlenode in self.queue

    def peek(self):
        "Returns first puzzle in queue"

        return self.queue[0]
```

```
    def poll(self):
        "Removes first puzzle in queue"

        return self.queue.pop(0)

    def remove(self, puzzlenode):
        "Removes puzzle from queue"

        self.queue.remove(puzzlenode)

    def size(self):
        "Returns size of queue"

        return len(self.queue)

    def to_array(self):
        "Converts queue to array"

        return self.queue.copy()

    def is_empty(self):
        "Returns true if queue is empty"

        return len(self.queue) == 0
```

search.py

Modul yang berisi algoritma utama dari pencarian solusi 15-puzzle.

```
from puzzlenode import PuzzleNode
from puzzleprioqueue import PuzzlePrioQueue
from collections import deque
from moves import Moves


def get_last_move(puzzlenode):
    return puzzlenode.puzzle.get_last_move()


def get_solution_stack(base_tree):
    # assume base_tree is valid PuzzleNode class
    # base_tree puzzle is also not solved
    # return solution stack

    # method:
    # 1. create a priority queue
```

```
    # 2. add base_tree to the queue
    # 3. while queue is not empty:
    # 4.   poll the first node from the queue
    # 5.   if the node is solved:
    # 6.       return the solution stack
    # 7.   else:
    # 8.       add the children of the node to the queue
    # 9. return None

    # 1. create a priority queue
    # 2. add base_tree to the queue
    queue = PuzzlePrioQueue()
    queue.add(base_tree)

    node_count = 1

    checked_puzzle = {}
    checked_puzzle[base_tree.puzzle.puzzle.tobytes()] = True

    # 3. while queue is not empty:
    while not queue.is_empty():
        # 4.   poll the first node from the queue
        node = queue.poll()

        # node is a valid PuzzleNode object

        print("node_count: " + str(node_count))

        # 5.   if the node is solved:
        if node.puzzle.is_solved():
            # 6a. create solution stack
            solution_stack = deque()

            # 6b. add node to solution stack
            solution_stack.append(node)

            # 6c. while node is not base_tree:
            while node.parent is not None:
                # 6d. add node's parent to solution stack
                solution_stack.append(node.parent)

                # 6e. set node to node's parent
                node = node.parent

            # 6f. return solution stack
            return solution_stack, node_count

        # 7.   else:
        else:
```

```
            # 8a. get tile 16 position
            tile_16_pos = node.puzzle.get_tile_position_16()

            last_move = get_last_move(node)

            # 8b. add only valid moves to children
            if last_move != Moves.RIGHT or not tile_16_pos.is_left_most():
                # 8c. create child puzzle
                child_puzzle = node.puzzle.clone()
                child_puzzle.move_left()

                if child_puzzle.puzzle.tobytes() not in checked_puzzle:
                    # 8d. create child node
                    child_node = PuzzleNode(child_puzzle, node, [])

                    # 8e. add child node to queue
                    queue.add(child_node)

                    checked_puzzle[child_puzzle.puzzle.tobytes()] = True
                    node_count += 1

            # 8g. repeat for right, up, and down
            if last_move != Moves.LEFT and not tile_16_pos.is_right_most(4):
                child_puzzle = node.puzzle.clone()
                child_puzzle.move_right()

                if child_puzzle.puzzle.tobytes() not in checked_puzzle:
                    child_node = PuzzleNode(child_puzzle, node, [])

                    queue.add(child_node)

                    checked_puzzle[child_puzzle.puzzle.tobytes()] = True
                    node_count += 1

            if last_move != Moves.DOWN and not tile_16_pos.is_top_most():
                child_puzzle = node.puzzle.clone()
                child_puzzle.move_up()

                if child_puzzle.puzzle.tobytes() not in checked_puzzle:
                    child_node = PuzzleNode(child_puzzle, node, [])

                    queue.add(child_node)

                    checked_puzzle[child_puzzle.puzzle.tobytes()] = True
                    node_count += 1

            if last_move != Moves.UP and not tile_16_pos.is_bottom_most(4):
                child_puzzle = node.puzzle.clone()
                child_puzzle.move_down()
```

```
                if child_puzzle.puzzle.tobytes() not in checked_puzzle:
                    child_node = PuzzleNode(child_puzzle, node, [])

                    queue.add(child_node)

                    checked_puzzle[child_puzzle.puzzle.tobytes()] = True
                    node_count += 1

        # endif
    # end while
```

C. INPUT-OUTPUT

Tes 1-3 merupakan puzzle yang bisa diselesaikan, tes 4-5 adalah puzzle yang itdak bisa diselesaikan.

Tiap tes berupa satu baris panjang.

test1.txt

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

```
D:\repos\Tucil3_13520098>python src/main.py

Select input menu
1. Input from file
2. Input from keyboard
3. Exit
Select: 1

Insert your file name
File name: test1.txt
Base tree:
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|    |  6 |  7 |  8 |
+----+----+----+----+
|  5 | 10 | 11 | 12 |
+----+----+----+----+
|  9 | 13 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:
This puzzle is Solvable
```

```
Value of kurang(i) for each non-empty tile

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

Estimate values (sum(kurang(i) + X)): 18
This puzzle is Solvable
Solution:
Step 0
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|    |  6 |  7 |  8 |
+----+----+----+----+
|  5 | 10 | 11 | 12 |
+----+----+----+----+
|  9 | 13 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:
```

```
Step 1
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|    | 10 | 11 | 12 |
+----+----+----+----+
|  9 | 13 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 5
Parent: <puzzlenode.PuzzleNode object at 0×00000207609D7FA0>
Children: []
Last move: Moves.DOWN

Step 2
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
|    | 13 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 8
Parent: <puzzlenode.PuzzleNode object at 0×000002077915EB90>
Children: []
Last move: Moves.DOWN
```

```
Step 3
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 |    | 14 | 15 |
+----+----+----+----+
Puzzle ID: 10
Parent: <puzzlenode.PuzzleNode object at 0×000002077915EDD0>
Children: []
Last move: Moves.RIGHT

Step 4
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 |    | 15 |
+----+----+----+----+
Puzzle ID: 13
Parent: <puzzlenode.PuzzleNode object at 0×000002077915EF50>
Children: []
Last move: Moves.RIGHT
```

```
Step 5
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 | 15 |    |
+----+----+----+----+
Puzzle ID: 15
Parent: <puzzlenode.PuzzleNode object at 0×000002077915F130>
Children: []
Last move: Moves.RIGHT

** End of solution **
Number of nodes (Simpul dibangkitkan): 16
Execution time: 0.0029997825622558594 seconds
Do you want to save the solution to a file? (Y/n)
y

Insert your file name
File name:
Solution saved to result.txt
```

test2.txt

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

```
Select input menu
1. Input from file
2. Input from keyboard
3. Exit
Select: 1

Insert your file name
File name: test2.txt
Base tree:
+----+----+----+----+
|  1 |  3 |  7 |  4 |
+----+----+----+----+
|  5 |  2 | 11 |    |
+----+----+----+----+
|  9 |  6 | 12 |  8 |
+----+----+----+----+
| 13 | 10 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:
This puzzle is Solvable

Value of kurang(i) for each non-empty tile

kurang(1): 0
kurang(2): 0
kurang(3): 1
kurang(4): 1
kurang(5): 1
kurang(6): 0
kurang(7): 4
kurang(8): 0
kurang(9): 2
kurang(10): 0
kurang(11): 4
kurang(12): 2
kurang(13): 1
```

```
Estimate values (sum(kurang(i) + X)): 24
This puzzle is Solvable
Solution:
Step 0
+----+----+----+----+
|  1 |  3 |  7 |  4 |
+----+----+----+----+
|  5 |  2 | 11 |    |
+----+----+----+----+
|  9 |  6 | 12 |  8 |
+----+----+----+----+
| 13 | 10 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:

Step 1
+----+----+----+----+
|  1 |  3 |  7 |  4 |
+----+----+----+----+
|  5 |  2 | 11 |  8 |
+----+----+----+----+
|  9 |  6 | 12 |    |
+----+----+----+----+
| 13 | 10 | 14 | 15 |
+----+----+----+----+
Puzzle ID: 4
Parent: <puzzlenode.PuzzleNode object at 0×00000207609D7F70>
Children: []
Last move: Moves.DOWN
```

(... step 2-8 diskip ...)

```
Step 9
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 |    | 15 |
+----+----+----+----+
Puzzle ID: 25
Parent: <puzzlenode.PuzzleNode object at 0×000002077915FC40>
Children: []
Last move: Moves.RIGHT

Step 10
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 | 15 |    |
+----+----+----+----+
Puzzle ID: 26
Parent: <puzzlenode.PuzzleNode object at 0×000002077915FAC0>
Children: []
Last move: Moves.RIGHT

** End of solution **
Number of nodes (Simpul dibangkitkan): 27
Execution time: 0.006016254425048828 seconds
Do you want to save the solution to a file? (Y/n)
y
```

test3.txt

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

```
D:\repos\Tucil3_13520098>python src/main.py

Select input menu
1. Input from file
2. Input from keyboard
3. Exit
Select: 1

Insert your file name
File name: test3.txt
Base tree:
+----+----+----+----+
|  1 |  3 |  7 |  4 |
+----+----+----+----+
|  5 |  2 |    |  8 |
+----+----+----+----+
|  9 |  6 | 11 | 15 |
+----+----+----+----+
| 13 | 10 | 12 | 14 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:
This puzzle is Solvable
```

```
Value of kurang(i) for each non-empty tile

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

Estimate values (sum(kurang(i) + X)): 26
This puzzle is Solvable
Solution:
Step 0
+----+----+----+----+
|  1 |  3 |  7 |  4 |
+----+----+----+----+
|  5 |  2 |    |  8 |
+----+----+----+----+
|  9 |  6 | 11 | 15 |
+----+----+----+----+
| 13 | 10 | 12 | 14 |
+----+----+----+----+
Puzzle ID: 1
Parent: None
Children: []
Last move:
```

```
Step 1
+----+----+----+----+
|  1 |  3 |    |  4 |
+----+----+----+----+
|  5 |  2 |  7 |  8 |
+----+----+----+----+
|  9 |  6 | 11 | 15 |
+----+----+----+----+
| 13 | 10 | 12 | 14 |
+----+----+----+----+
Puzzle ID: 4
Parent: <puzzlenode.PuzzleNode object at 0×000001A6BF057FA0>
Children: []
Last move: Moves.UP

Step 2
+----+----+----+----+
|  1 |    |  3 |  4 |
+----+----+----+----+
|  5 |  2 |  7 |  8 |
+----+----+----+----+
|  9 |  6 | 11 | 15 |
+----+----+----+----+
| 13 | 10 | 12 | 14 |
+----+----+----+----+
Puzzle ID: 6
Parent: <puzzlenode.PuzzleNode object at 0×000001A6D77A7790>
Children: []
Last move: Moves.LEFT
```

(... step 3-11 diskip ...)

```
Step 12
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 |    | 15 |
+----+----+----+----+
Puzzle ID: 392
Parent: <puzzlenode.PuzzleNode object at 0×000001A6D7859720>
Children: []
Last move: Moves.RIGHT

Step 13
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
| 13 | 14 | 15 |    |
+----+----+----+----+
Puzzle ID: 393
Parent: <puzzlenode.PuzzleNode object at 0×000001A6D7859C00>
Children: []
Last move: Moves.RIGHT

** End of solution **
Number of nodes (Simpul dibangkitkan): 394
Execution time: 0.466031551361084 seconds
Do you want to save the solution to a file? (Y/n)
```

test4.txt

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

```
Insert your file name
File name: test4.txt
Base tree:
+----+----+----+----+
|  1 |  2 |  3 |  4 |
+----+----+----+----+
|  5 |  6 |  7 |  8 |
+----+----+----+----+
|  9 | 10 | 11 | 12 |
+----+----+----+----+
|    | 15 | 14 | 13 |
+----+----+----+----+
Puzzle ID: 2
Parent: None
Children: []
Last move:
This puzzle is Not solvable

Value of kurang(i) for each non-empty tile

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

Estimate values (sum(kurang(i) + X)): 7
```

```
Estimate values (sum(kurang(i) + X)): 7
This puzzle is Not solvable
Termination message:
This puzzle is not solvable
```

test5.txt

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

```
Insert your file name
File name: test5.txt
Base tree:
+----+----+----+----+
|  4 |  3 |  2 |  1 |
+----+----+----+----+
|  8 |  7 |  6 |  5 |
+----+----+----+----+
|    | 15 | 13 | 14 |
+----+----+----+----+
| 10 | 12 | 11 |  9 |
+----+----+----+----+
Puzzle ID: 3
Parent: None
Children: []
Last move:
This puzzle is Not solvable

Value of kurang(i) for each non-empty tile

kurang(1): 0
kurang(2): 1
kurang(3): 2
kurang(4): 3
kurang(5): 0
kurang(6): 1
kurang(7): 2
kurang(8): 3
kurang(9): 0
kurang(10): 1
kurang(11): 1
kurang(12): 2
kurang(13): 4
kurang(14): 4
kurang(15): 6
```

```
Estimate values (sum(kurang(i) + X)): 37
This puzzle is Not solvable
Termination message:
This puzzle is not solvable
```

D. **ALAMAT GITHUB**

https://github.com/dawetmaster/Tucil3_13520098

E. **CHECK LIST**

Poin	Ya	Tidak
Program berhasil dikompilasi	V	
Program berhasil *running*	V	
Program dapat menerima input dan menuliskan output.	V	
Luaran sudah benar untuk semua data uji		

Bonus dibuat		V