LAPORAN TUGAS KECIL 3

IF2211 STRATEGI ALGORITMA

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound

NIM	: 13520072
Nama	: Jova Andres Riski Sirait
Kelas	: K03

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2022

A. Algoritma Branch and Bound

Algoritma Branch and Bound adalah suatu algoritma pencarian jalur menuju sebuah solusi dengan membentuk pohon ruang status. Pada algoritma Branch and Bound akan dihitung cost dari sebuah simpul yang ada dan memilih pembangkitan cabang yang memiliki cost paling kecil.

Pada tugas 15-Puzzle ini, penerapan algoritma Branch and Bound yang dilakukan adalah sebagai berikut:

1. Pilih puzzle yang ingin diselesaikan sebagai simpul awal. Puzzle awal dapat dimuat dari file eksternal dalam bentuk matriks ataupun dapat menggunakan generator yang disediakan oleh program. Untuk puzzle yang digenerate random, kemungkinan komputasi sangat besar karena simpul yang dibangkitkan cukup banyak dan mungkin juga tidak dapat diselesaikan.
2. Tentukan apakah solusi dari puzzle dapat ditemukan dengan algoritma Branch and Bound dengan implementasi dari fungsi Kurang(i) dan posisi ubin kosong di awal (X).
3. Tentukan gerakan dari ubin kosong yang mungkin. Untuk simpul awal, simpul diinisiasi untuk dapat bergerak ke arah atas, bawah, kiri, dan kanan. Kemudian berdasarkan posisi ubin kosong, akan ditentukan arah gerakan yang tidak memungkinkan. Contohnya jika ubin berada pada baris paling atas, maka gerakan ke arah atas akan dihapus.
4. Untuk setiap gerakan yang mungkin dari ubin kosong (UP, DOWN, LEFT, RIGHT), akan dilakukan iterasi dan generasi simpul anak yang baru. Setiap simpul yang dibangkitkan kemudian akan dimasukkan ke dalam PriorityQueue dengan prioritas pertama adalah cost dari setiap simpul yang akan dihitung terlebih dahulu dan prioritas kedua adalah angka acak.
5. Selama belum menemukan simpul solusi dan masih terdapat antrian pada PriorityQueue, maka pohon ruang status akan terus terbentuk (kembali ke langkah no 3).
6. Untuk simpul selain simpul awal, akan dicatat gerakan ubin sebelumnya dari setiap *instance* untuk mengeliminasi sebuah gerakan ubin yang tidak perlu. Contohnya jika gerakan sebelumnya adalah UP, maka DOWN akan dieliminasi karena kembali ke state sebelumnya (tidak menuju solusi).

B. Source Code Program

Terdapat 3 file utama pada program yaitu Puzzle.py sebagai representasi objek dari Puzzle, Solver.py sebagai kelas utama yang menyimpan alur utama program, Direction.py sebagai kelas representasi dari objek yang menunjukkan arah gerak ubin (UP, DOWN, RIGHT, LEFT) dan main.py sebagai titik awal berjalannya program.

1. Direction.py

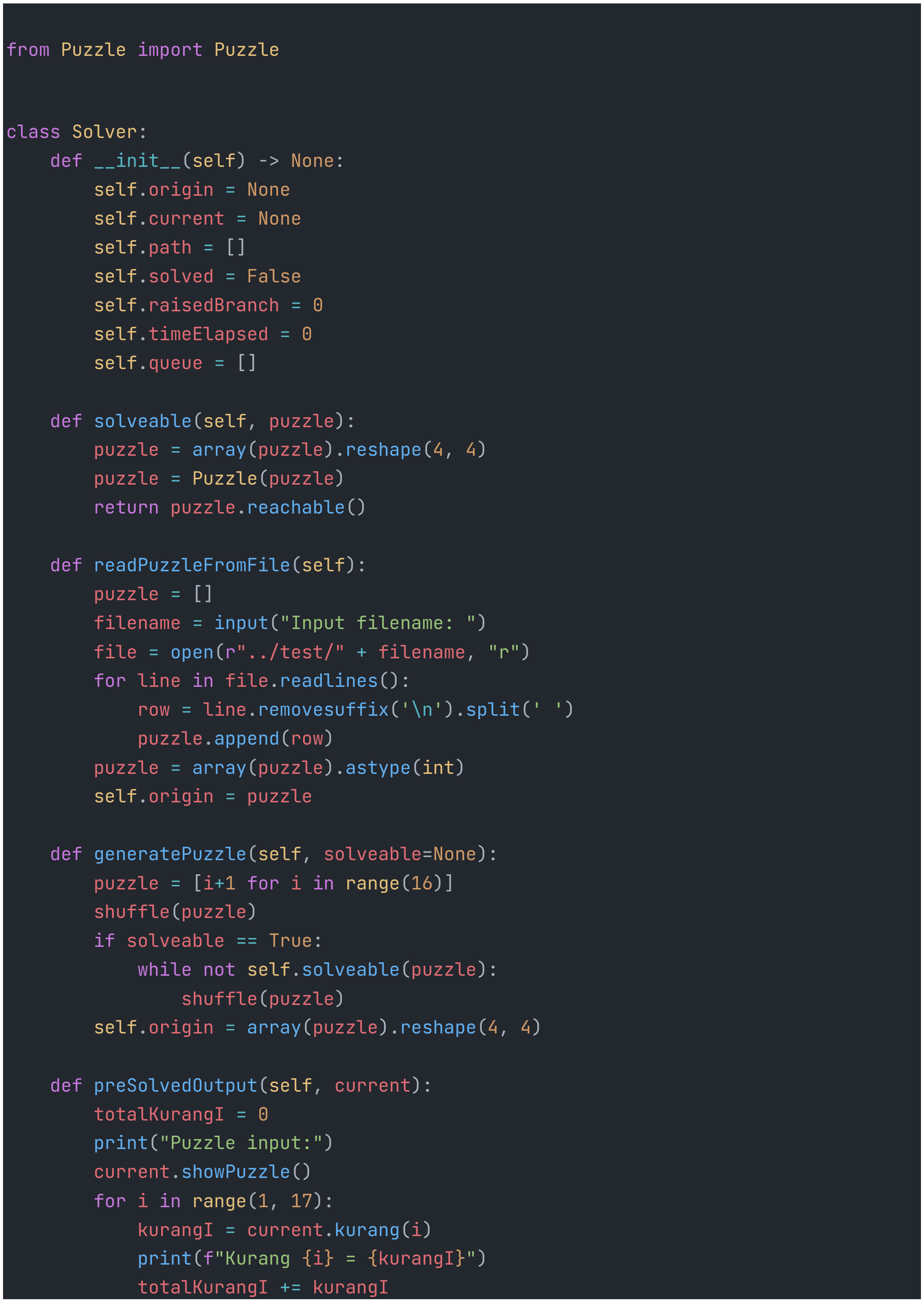
```python
from enum import Enum


class Direction(Enum):
    UP = "UP"
    DOWN = "DOWN"
    LEFT = "LEFT"
    RIGHT = "RIGHT"

    def opposite(direction):
        if direction == Direction.UP:
            return Direction.DOWN
        elif direction == Direction.DOWN:
            return Direction.UP
        elif direction == Direction.LEFT:
            return Direction.RIGHT
        elif direction == Direction.RIGHT:
            return Direction.LEFT
```

2. Puzzle.py

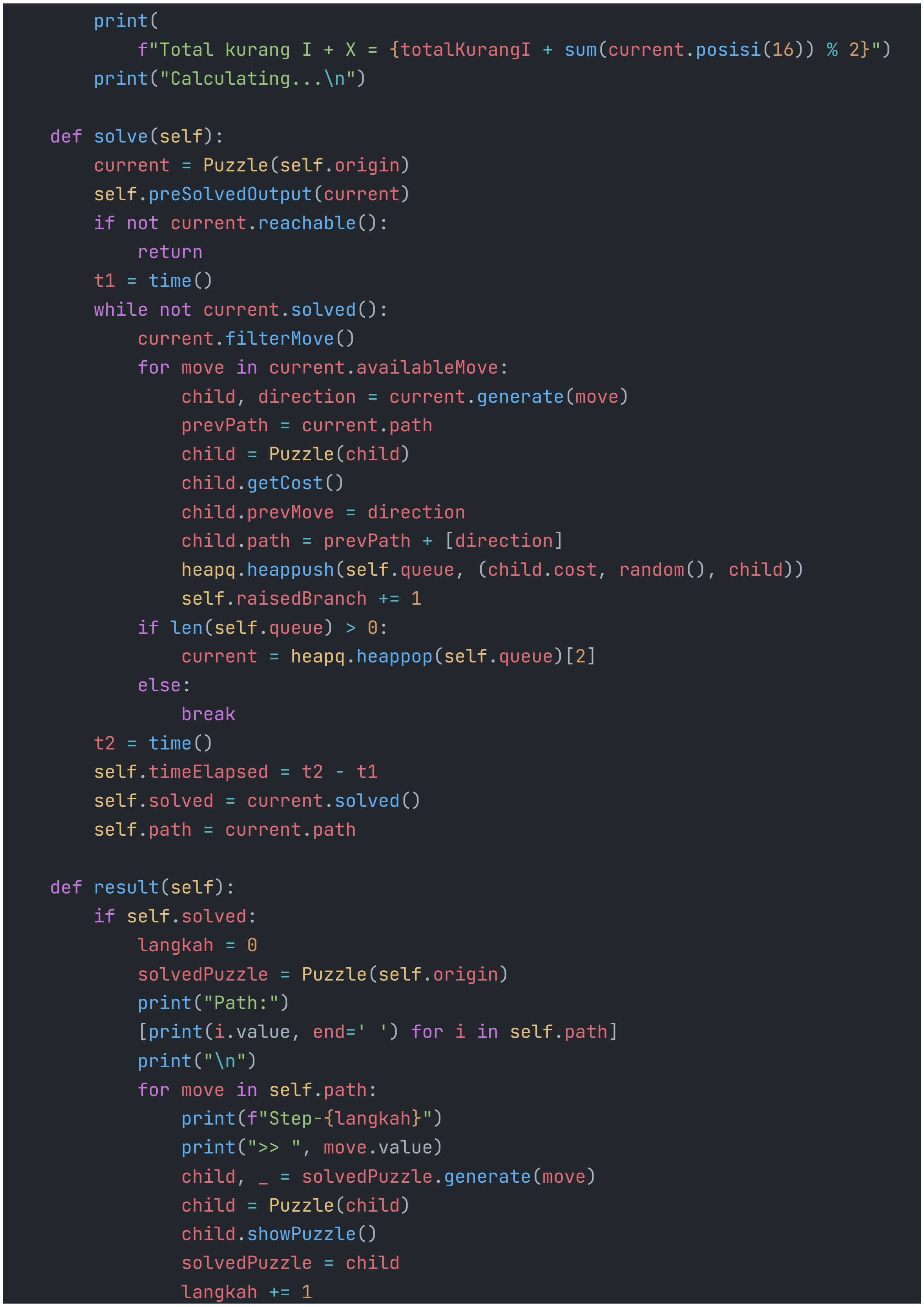
```python
from numpy import array, array_equal, copy, where
from Direction import Direction


class Puzzle:
    def __init__(self, puzzleData) -> None:
        self.puzzle = puzzleData
        self.availableMove = []
        self.cost = 0
        self.path = []
        self.prevMove = None
        self.solution = array([
            [1, 2, 3, 4],
            [5, 6, 7, 8],
            [9, 10, 11, 12],
            [13, 14, 15, 16]
        ])
    def filterMove(self):
        blankPosition = where(self.puzzle == 16)
        blankPosition = (blankPosition[0][0], blankPosition[1][0])
        self.availableMove = [Direction.UP,
                              Direction.DOWN, Direction.LEFT, Direction.RIGHT]
        if blankPosition[0] == 0:
            self.availableMove.remove(Direction.UP)
        if blankPosition[0] == 3:
            self.availableMove.remove(Direction.DOWN)
        if blankPosition[1] == 0:
            self.availableMove.remove(Direction.LEFT)
        if blankPosition[1] == 3:
            self.availableMove.remove(Direction.RIGHT)
        if self.prevMove != None and self.prevMove in self.availableMove:
            opposite = Direction.opposite(self.prevMove)
            self.availableMove.remove(opposite)

    def getCost(self):
        intersect = where((self.puzzle == self.solution), 0, 1)
        self.cost = intersect.sum()

    def posisi(self, i):
        x = where(self.puzzle == i)
        return x[0][0], x[1][0]

    def kurang(self, i):
        total = 0
        for j in range(4):
            for k in range(4):
                if self.puzzle[j][k] < i and self.posisi(self.puzzle[j][k]) >
self.posisi(i):
                    total += 1
        return total

    def generate(self, direction):
        if direction == Direction.UP:
            return self.generateUp()
        elif direction == Direction.DOWN:
            return self.generateDown()
        elif direction == Direction.LEFT:
            return self.generateLeft()
        elif direction == Direction.RIGHT:
            return self.generateRight()
    def generateLeft(self):
        child = copy(self.puzzle)
        x, y = self.posisi(16)
        child[x][y], child[x][y - 1] = child[x][y - 1], child[x][y]
        return child, Direction.LEFT

    def generateRight(self):
        child = copy(self.puzzle)
        x, y = self.posisi(16)
        child[x][y], child[x][y + 1] = child[x][y + 1], child[x][y]
        return child, Direction.RIGHT

    def generateUp(self):
        child = copy(self.puzzle)
        x, y = self.posisi(16)
        child[x][y], child[x - 1][y] = child[x - 1][y], child[x][y]
        return child, Direction.UP

    def generateDown(self):
        child = copy(self.puzzle)
        x, y = self.posisi(16)
        child[x][y], child[x + 1][y] = child[x + 1][y], child[x][y]
        return child, Direction.DOWN

    def reachable(self):
        totalKurangI = sum([self.kurang(i) for i in range(1, 17)])
        a, b = self.posisi(16)
        X = (a + b) % 2
        return (totalKurangI + X) % 2 == 0

    def solved(self):
        return array_equal(self.puzzle, self.solution)

    def showPuzzle(self):
        newPuzzle = copy(self.puzzle)
        newPuzzle[newPuzzle == 16] = 0
        print(newPuzzle)
        print()
```

3. Solver.py

```
import heapq
from random import random, shuffle
from time import time
from numpy import array
```



```python
from Puzzle import Puzzle


class Solver:
    def __init__(self) -> None:
        self.origin = None
        self.current = None
        self.path = []
        self.solved = False
        self.raisedBranch = 0
        self.timeElapsed = 0
        self.queue = []

    def solveable(self, puzzle):
        puzzle = array(puzzle).reshape(4, 4)
        puzzle = Puzzle(puzzle)
        return puzzle.reachable()

    def readPuzzleFromFile(self):
        puzzle = []
        filename = input("Input filename: ")
        file = open(r"../test/" + filename, "r")
        for line in file.readlines():
            row = line.removesuffix('\n').split(' ')
            puzzle.append(row)
        puzzle = array(puzzle).astype(int)
        self.origin = puzzle

    def generatePuzzle(self, solveable=None):
        puzzle = [i+1 for i in range(16)]
        shuffle(puzzle)
        if solveable == True:
            while not self.solveable(puzzle):
                shuffle(puzzle)
        self.origin = array(puzzle).reshape(4, 4)

    def preSolvedOutput(self, current):
        totalKurangI = 0
        print("Puzzle input:")
        current.showPuzzle()
        for i in range(1, 17):
            kurangI = current.kurang(i)
            print(f"Kurang {i} = {kurangI}")
            totalKurangI += kurangI
```



```python
        print(
            f"Total kurang I + X = {totalKurangI + sum(current.posisi(16)) % 2}")
        print("Calculating...\n")

    def solve(self):
        current = Puzzle(self.origin)
        self.preSolvedOutput(current)
        if not current.reachable():
            return
        t1 = time()
        while not current.solved():
            current.filterMove()
            for move in current.availableMove:
                child, direction = current.generate(move)
                prevPath = current.path
                child = Puzzle(child)
                child.getCost()
                child.prevMove = direction
                child.path = prevPath + [direction]
                heapq.heappush(self.queue, (child.cost, random(), child))
                self.raisedBranch += 1
            if len(self.queue) > 0:
                current = heapq.heappop(self.queue)[2]
            else:
                break
        t2 = time()
        self.timeElapsed = t2 - t1
        self.solved = current.solved()
        self.path = current.path

    def result(self):
        if self.solved:
            langkah = 0
            solvedPuzzle = Puzzle(self.origin)
            print("Path:")
            [print(i.value, end=' ') for i in self.path]
            print("\n")
            for move in self.path:
                print(f"Step-{langkah}")
                print(">> ", move.value)
                child, _ = solvedPuzzle.generate(move)
                child = Puzzle(child)
                child.showPuzzle()
                solvedPuzzle = child
                langkah += 1
            print("Total raised nodes: ", self.raisedBranch)
        else:
            print("Can't find a solution")
        print(f"Time elapsed: {round(self.timeElapsed, 6)} s")
        print("THANK YOU!!!")
```

4. main.py

```python
from Solver import Solver


print("15-Puzzle Solver\n")
solver = Solver()
print("Select options: \n1. Read puzzle from file \n2. Generate puzzle")
option = int(input(">> "))
if option == 1:
    solver.readPuzzleFromFile()
elif option == 2:
    solver.generatePuzzle()
solver.solve()
solver.result()
```

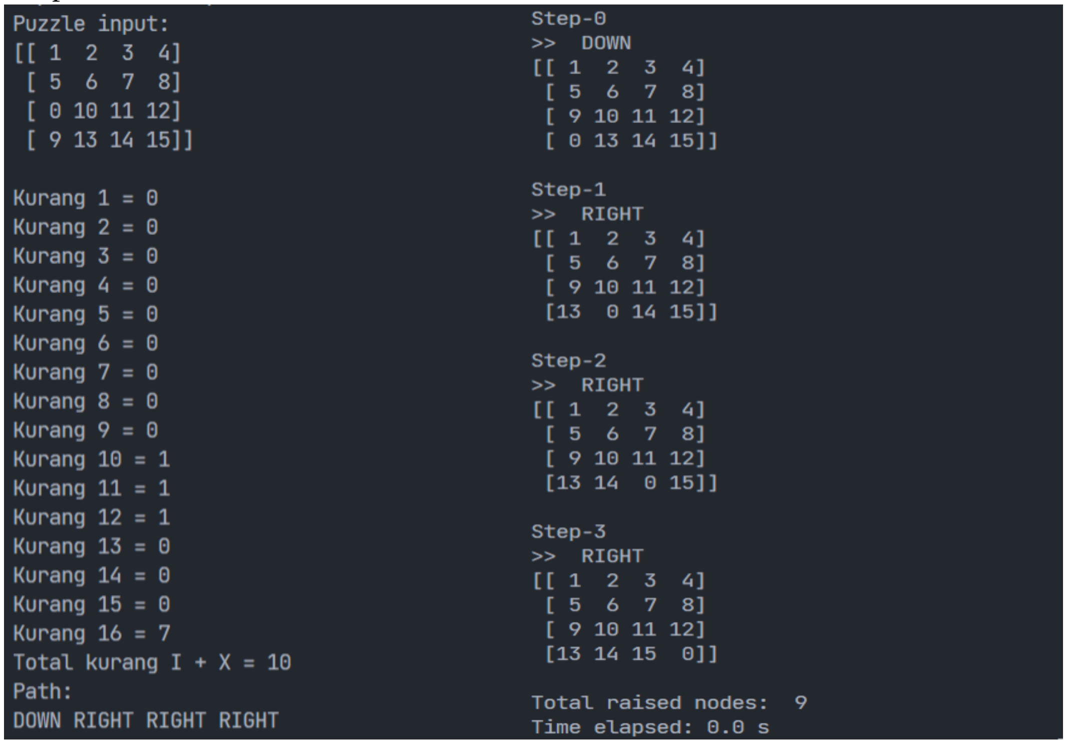
C. Screenshot Input dan Output

1. Puzzle 1 (Dapat diselesaikan)

   Input:

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

   Output:



```
Puzzle input:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 0 10 11 12]
 [ 9 13 14 15]]

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 = 1
Kurang 11 = 1
Kurang 12 = 1
Kurang 13 = 0
Kurang 14 = 0
Kurang 15 = 0
Kurang 16 = 7
Total kurang I + X = 10
Path:
DOWN RIGHT RIGHT RIGHT
```

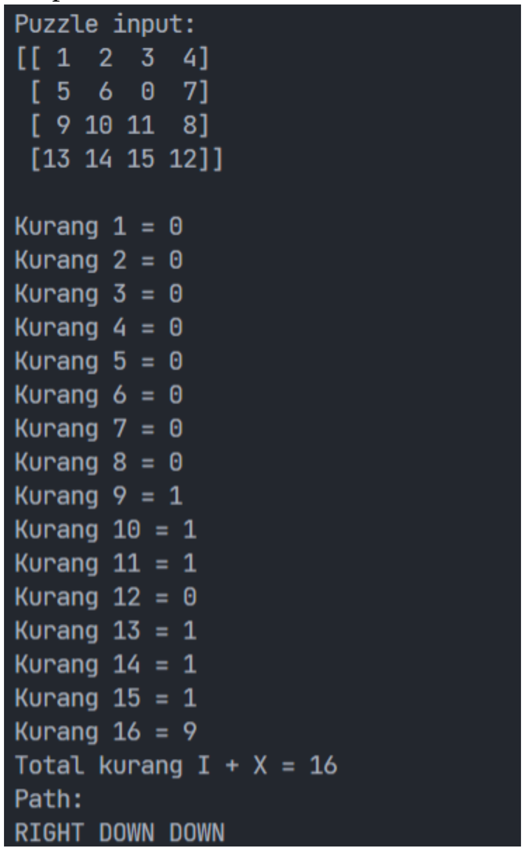
```
Step-0
>>  DOWN
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [ 0 13 14 15]]

Step-1
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13  0 14 15]]

Step-2
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14  0 15]]

Step-3
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15  0]]

Total raised nodes:  9
Time elapsed: 0.0 s
```

2. Puzzle 2 (Dapat diselesaikan)

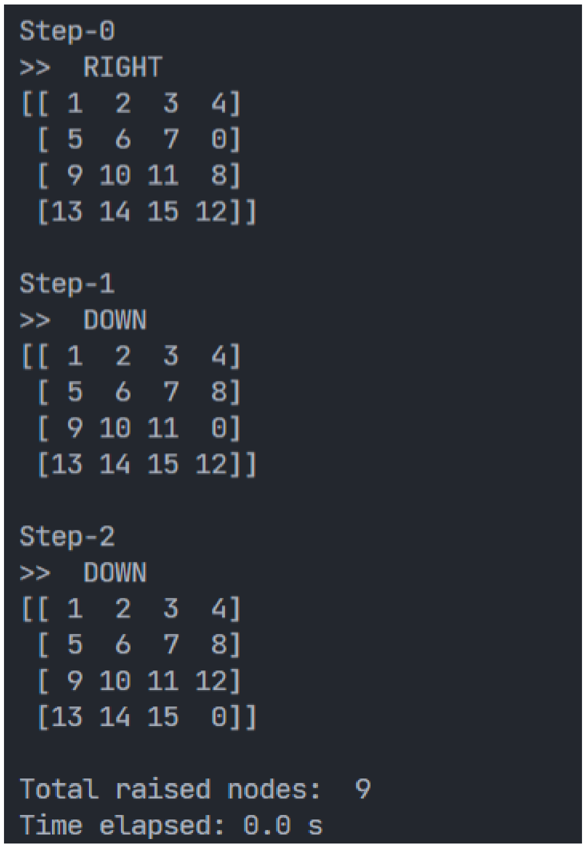
   Input:

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

Output:



```
Puzzle input:
[[ 1  2  3  4]
 [ 5  6  0  7]
 [ 9 10 11  8]
 [13 14 15 12]]

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 = 1
Kurang 10 = 1
Kurang 11 = 1
Kurang 12 = 0
Kurang 13 = 1
Kurang 14 = 1
Kurang 15 = 1
Kurang 16 = 9
Total kurang I + X = 16
Path:
RIGHT DOWN DOWN
```

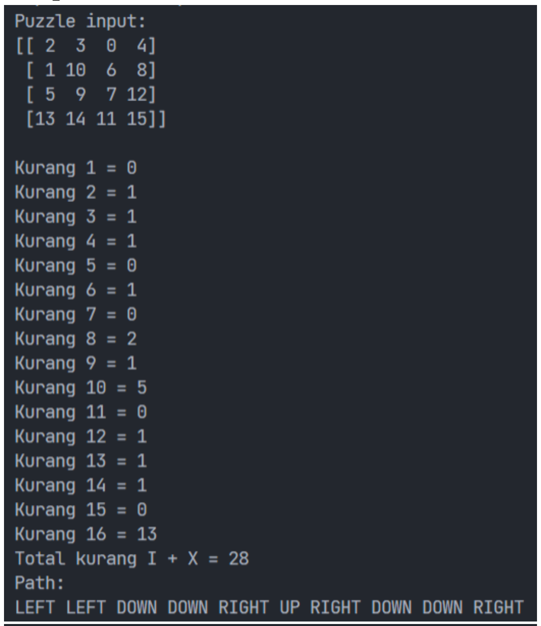


```
Step-0
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  7  0]
 [ 9 10 11  8]
 [13 14 15 12]]

Step-1
>>  DOWN
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11  0]
 [13 14 15 12]]

Step-2
>>  DOWN
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15  0]]

Total raised nodes:  9
Time elapsed: 0.0 s
```

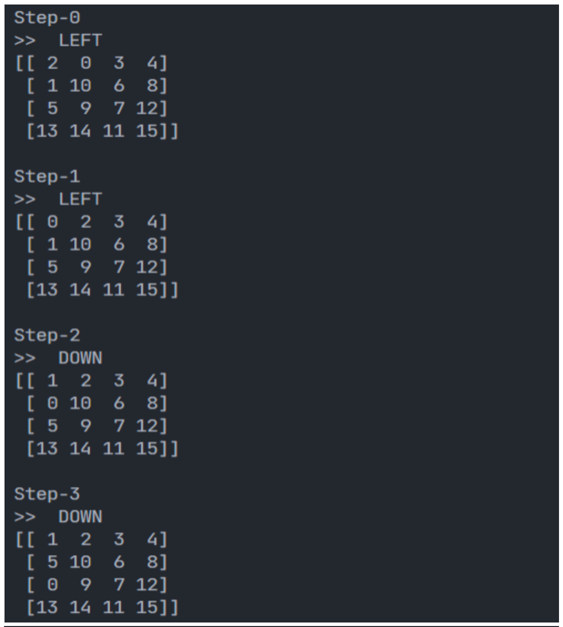
3. Puzzle 3 (Dapat diselesaikan)
   Input:

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

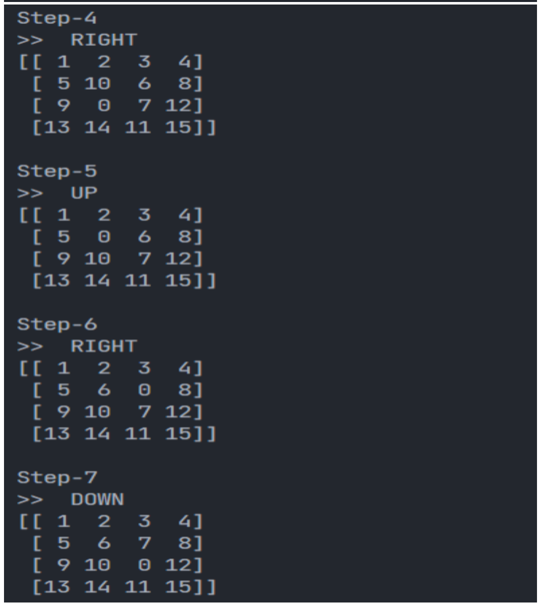
Output:



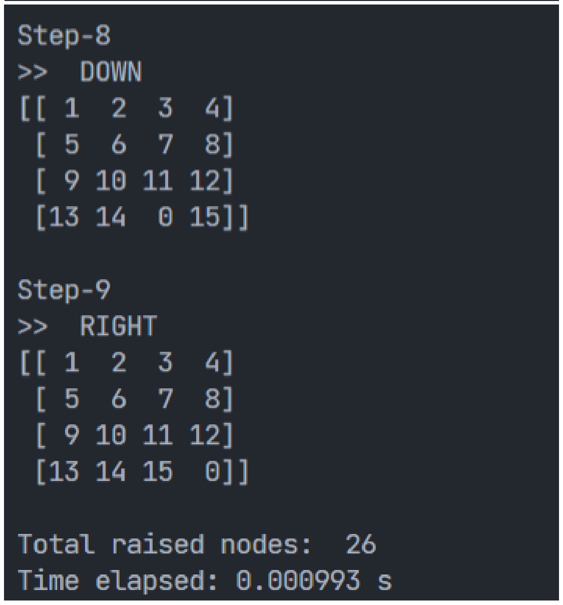
```
Puzzle input:
[[ 2  3  0  4]
 [ 1 10  6  8]
 [ 5  9  7 12]
 [13 14 11 15]]

Kurang 1 = 0
Kurang 2 = 1
Kurang 3 = 1
Kurang 4 = 1
Kurang 5 = 0
Kurang 6 = 1
Kurang 7 = 0
Kurang 8 = 2
Kurang 9 = 1
Kurang 10 = 5
Kurang 11 = 0
Kurang 12 = 1
Kurang 13 = 1
Kurang 14 = 1
Kurang 15 = 0
Kurang 16 = 13
Total kurang I + X = 28
Path:
LEFT LEFT DOWN DOWN RIGHT UP RIGHT DOWN DOWN RIGHT
```



```
Step-0
>>  LEFT
[[ 2  0  3  4]
 [ 1 10  6  8]
 [ 5  9  7 12]
 [13 14 11 15]]

Step-1
>>  LEFT
[[ 0  2  3  4]
 [ 1 10  6  8]
 [ 5  9  7 12]
 [13 14 11 15]]

Step-2
>>  DOWN
[[ 1  2  3  4]
 [ 0 10  6  8]
 [ 5  9  7 12]
 [13 14 11 15]]

Step-3
>>  DOWN
[[ 1  2  3  4]
 [ 5 10  6  8]
 [ 0  9  7 12]
 [13 14 11 15]]
```



```
Step-4
>>  RIGHT
[[ 1  2  3  4]
 [ 5 10  6  8]
 [ 9  0  7 12]
 [13 14 11 15]]

Step-5
>>  UP
[[ 1  2  3  4]
 [ 5  0  6  8]
 [ 9 10  7 12]
 [13 14 11 15]]

Step-6
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  0  8]
 [ 9 10  7 12]
 [13 14 11 15]]

Step-7
>>  DOWN
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10  0 12]
 [13 14 11 15]]
```

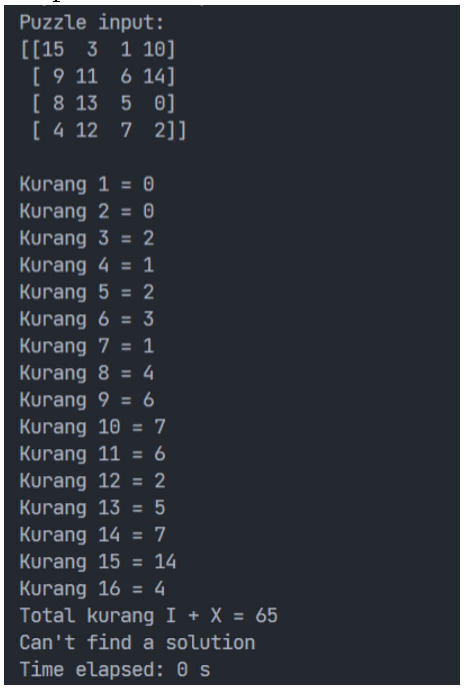


```
Step-8
>>  DOWN
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14  0 15]]

Step-9
>>  RIGHT
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15  0]]

Total raised nodes:  26
Time elapsed: 0.000993 s
```

4. Puzzle 4 (Tidak dapat diselesaikan)
   Input:

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

Output:

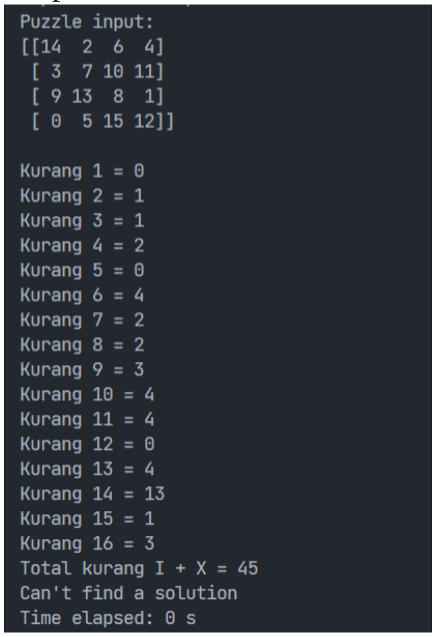


```
Puzzle input:
[[15  3  1 10]
 [ 9 11  6 14]
 [ 8 13  5  0]
 [ 4 12  7  2]]

Kurang 1 = 0
Kurang 2 = 0
Kurang 3 = 2
Kurang 4 = 1
Kurang 5 = 2
Kurang 6 = 3
Kurang 7 = 1
Kurang 8 = 4
Kurang 9 = 6
Kurang 10 = 7
Kurang 11 = 6
Kurang 12 = 2
Kurang 13 = 5
Kurang 14 = 7
Kurang 15 = 14
Kurang 16 = 4
Total kurang I + X = 65
Can't find a solution
Time elapsed: 0 s
```

5. Puzzle 5 (Tidak dapat diselesaikan)
   Input:

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

Output:



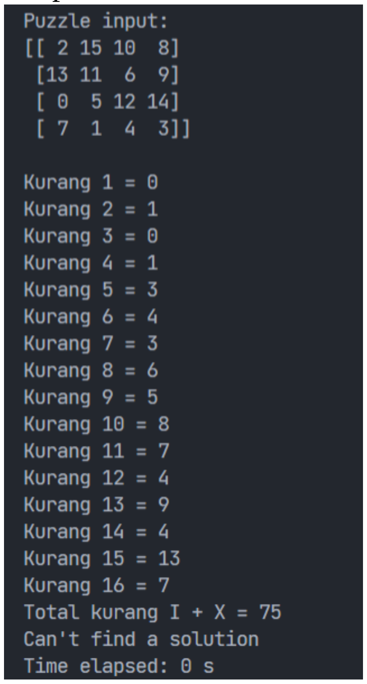
```
Puzzle input:
[[14  2  6  4]
 [ 3  7 10 11]
 [ 9 13  8  1]
 [ 0  5 15 12]]

Kurang 1 = 0
Kurang 2 = 1
Kurang 3 = 1
Kurang 4 = 2
Kurang 5 = 0
Kurang 6 = 4
Kurang 7 = 2
Kurang 8 = 2
Kurang 9 = 3
Kurang 10 = 4
Kurang 11 = 4
Kurang 12 = 0
Kurang 13 = 4
Kurang 14 = 13
Kurang 15 = 1
Kurang 16 = 3
Total kurang I + X = 45
Can't find a solution
Time elapsed: 0 s
```

6. Puzzle 6 (Random puzzle generator)

Input:

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

Output:



```
Puzzle input:
[[ 2 15 10  8]
 [13 11  6  9]
 [ 0  5 12 14]
 [ 7  1  4  3]]

Kurang 1 = 0
Kurang 2 = 1
Kurang 3 = 0
Kurang 4 = 1
Kurang 5 = 3
Kurang 6 = 4
Kurang 7 = 3
Kurang 8 = 6
Kurang 9 = 5
Kurang 10 = 8
Kurang 11 = 7
Kurang 12 = 4
Kurang 13 = 9
Kurang 14 = 4
Kurang 15 = 13
Kurang 16 = 7
Total kurang I + X = 75
Can't find a solution
Time elapsed: 0 s
```

D. Alamat Drive

https://github.com/jovaandres/Tucil3_13520072

E. Tabel 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		✓