LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

15-Puzzle Solver using Branch and Bound Algorithm

oleh

Nadia Mareta Putri Leiden (13520007)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

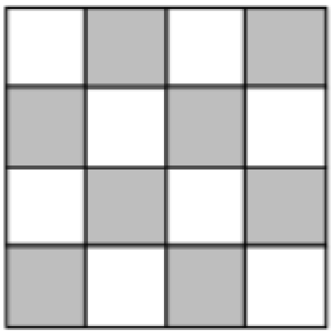
INSTITUT TEKNOLOGI BANDUNG

2022

1. Penjelasan Algoritma Branch and Bound untuk 15-Puzzle

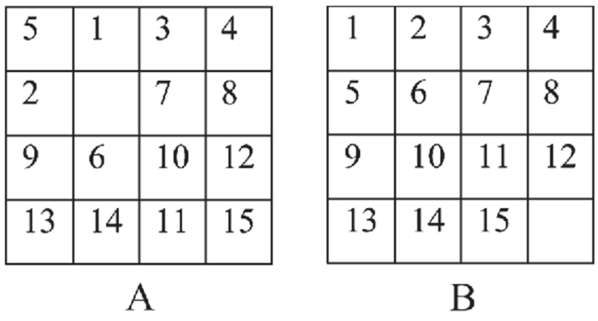
Sebelum masuk ke dalam Algoritma Branch and Bound itu sendiri, puzzle harus terlebih dahulu dicek apakah bisa diselesaikan atau tidak. Algoritma Kurang[i] digunakan untuk menghitung nilainya pada setiap elemen, jika dijumlahkan hasilnya genap, maka persoalan dapat diselesaikan. Sebaliknya, jika dijumlahkan hasilnya ganjil, maka persoalan tidak dapat diselesaikan. Algoritma ini terletak pada file BranchandBound.py dengan rangkaian fungsi: Kurangi(), countKurang(), dan isSolvable().

Ide algoritmanya adalah dengan mengiterasi seluruh elemen pada matriks misalnya matriks[i][j] kemudian mencari apakah ada elemen yang lebih kecil dari elemen saat ini pada posisi selanjutnya. Jika ada, maka jumlahnya disimpan dalam variable count. Nantinya ini akan digabung lagi apakah kotak kosong berada pada warna hitam atau putih, jika hitam (pada Gambar 1. Tile Black and White) maka nilai count tersebut akan ditambah satu, jika tidak maka tidak akan ditambahkan apa-apa.



Gambar 1. Tile Black and White

http://www.cs.umsl.edu/~sanjiv/classes/cs5130/lectures/bb.pdf



Gambar 2. Contoh Kasus 15-Puzzle

https://www.semanticscholar.org/paper/Solving-combinatorial-problems%3A-The-15-puzzle-Pizlo-Li/2c31956a61b5430292c667aa3e5c5f0f920278b5/figure/0

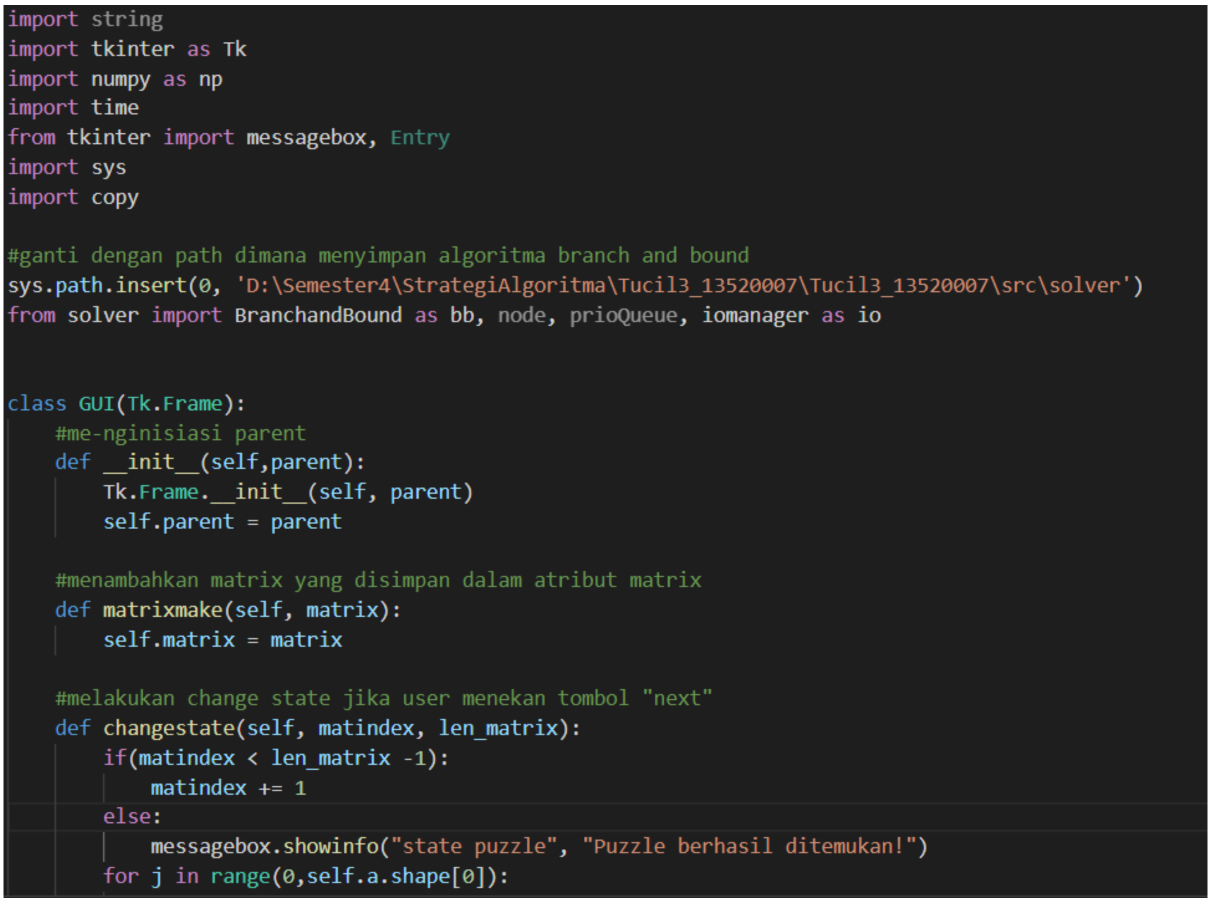
Untuk Algoritma Branch and Boundnya sendiri akan dieksekusi oleh fungsi utama yaitu fungsi solve_puzzle() pada BranchandBound.py.

Idenya adalah dengan terlebih dahulu membuat kelas node pada node.py, yang dapat menyimpan informasi berupa parent node, matrix parent, matrix saat ini, posisi tile kosong, cost yang dibutuhkan untuk mencapai node tersebut dan level node atau kedalaman node. Nantinya untuk membuat daftar node yang bisa dieksplor akan digunakan struktur data *Priority Queue* yang terdapat pada prioQueue.py yang urutan antrian didasarkan pada cost+level, sehingga eksekusi akan dilakukan dari node yang memiliki total cost+level paling sedikit. Node yang sudah dieksekusi akan di-*dequeue* dari antrian.

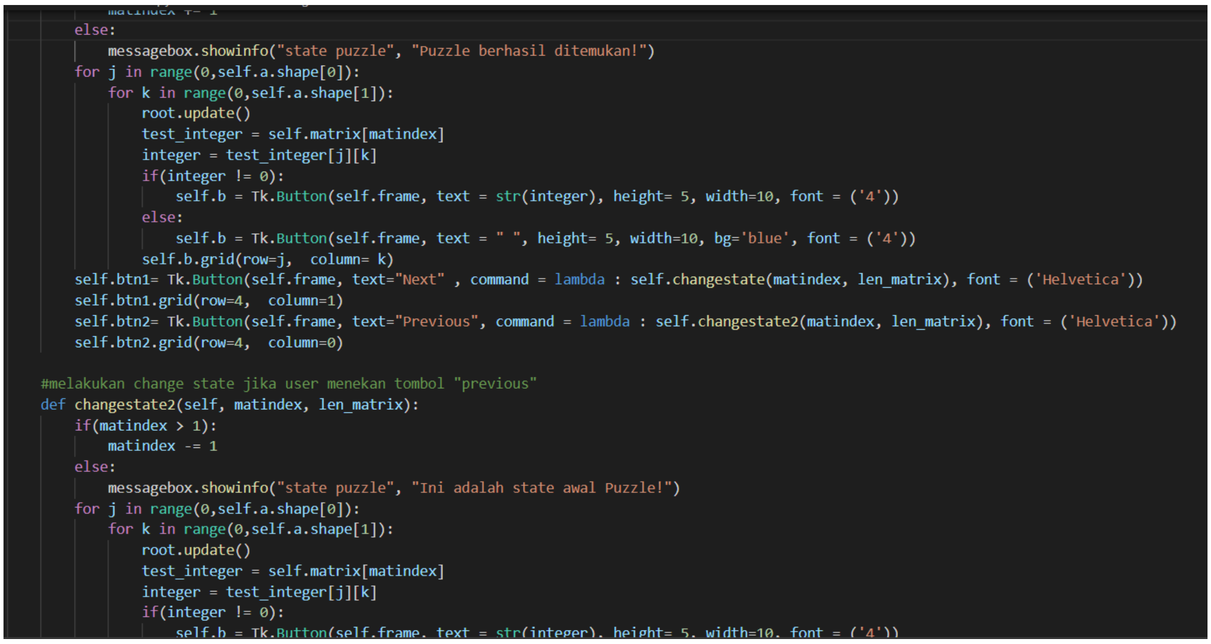
Setiap node akan di-*generate* simpul anaknya, selama simpul anaknya masih bisa di-*generate*, maka akan terus di-*generate* hingga cost nodenya sama dengan 0, yang artinya target matrix yang diinginkan sudah tercapai. Nantinya fungsi utama akan me-return dalam bentuk *array of matrix* yang akan digunakan untuk pengolahan data pada GUI yang telah dibuat.

2. Screenshot Seluruh Kode

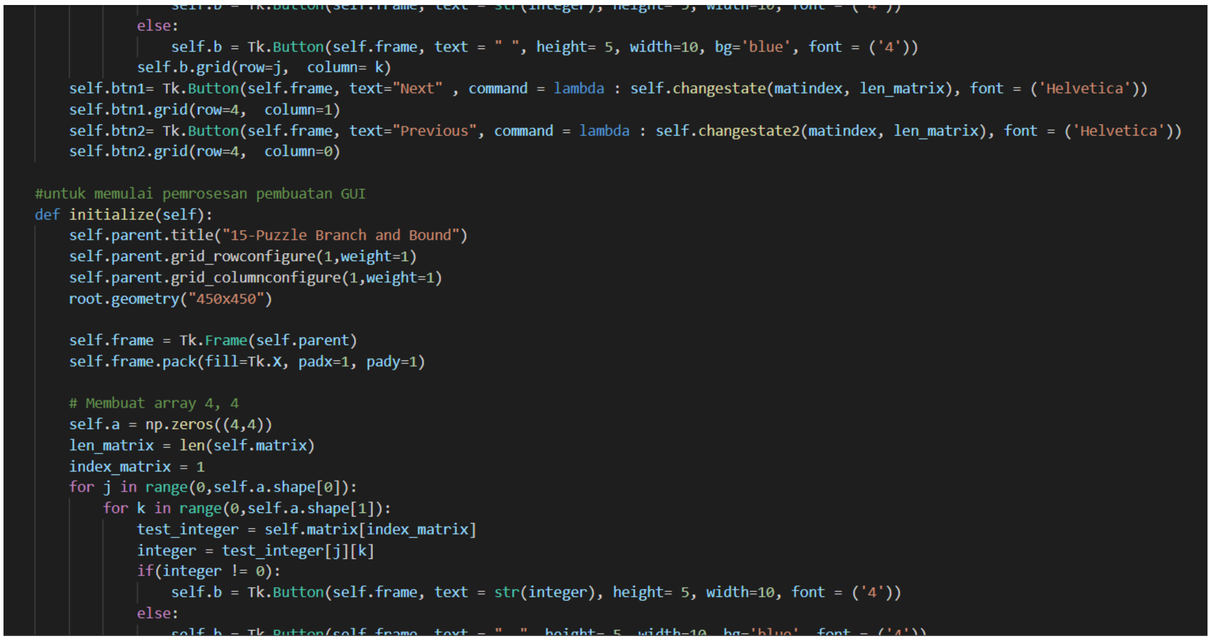
a. <u>GUI.py</u>



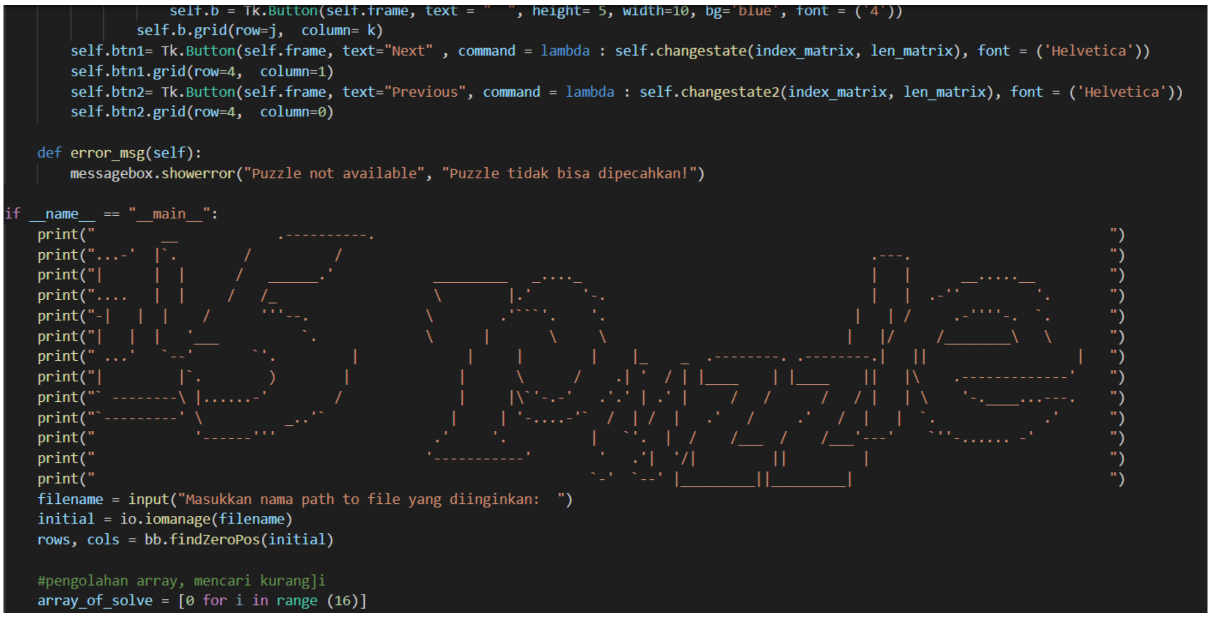
```python
import string
import tkinter as Tk
import numpy as np
import time
from tkinter import messagebox, Entry
import sys
import copy

#ganti dengan path dimana menyimpan algoritma branch and bound
sys.path.insert(0, 'D:\Semester4\StrategiAlgoritma\Tucil3_13520007\Tucil3_13520007\src\solver')
from solver import BranchandBound as bb, node, prioQueue, iomanager as io


class GUI(Tk.Frame):
    #me-nginisiasi parent
    def __init__(self,parent):
        Tk.Frame.__init__(self, parent)
        self.parent = parent

    #menambahkan matrix yang disimpan dalam atribut matrix
    def matrixmake(self, matrix):
        self.matrix = matrix

    #melakukan change state jika user menekan tombol "next"
    def changestate(self, matindex, len_matrix):
        if(matindex < len_matrix -1):
            matindex += 1
        else:
            messagebox.showinfo("state puzzle", "Puzzle berhasil ditemukan!")
        for j in range(0,self.a.shape[0]):
```



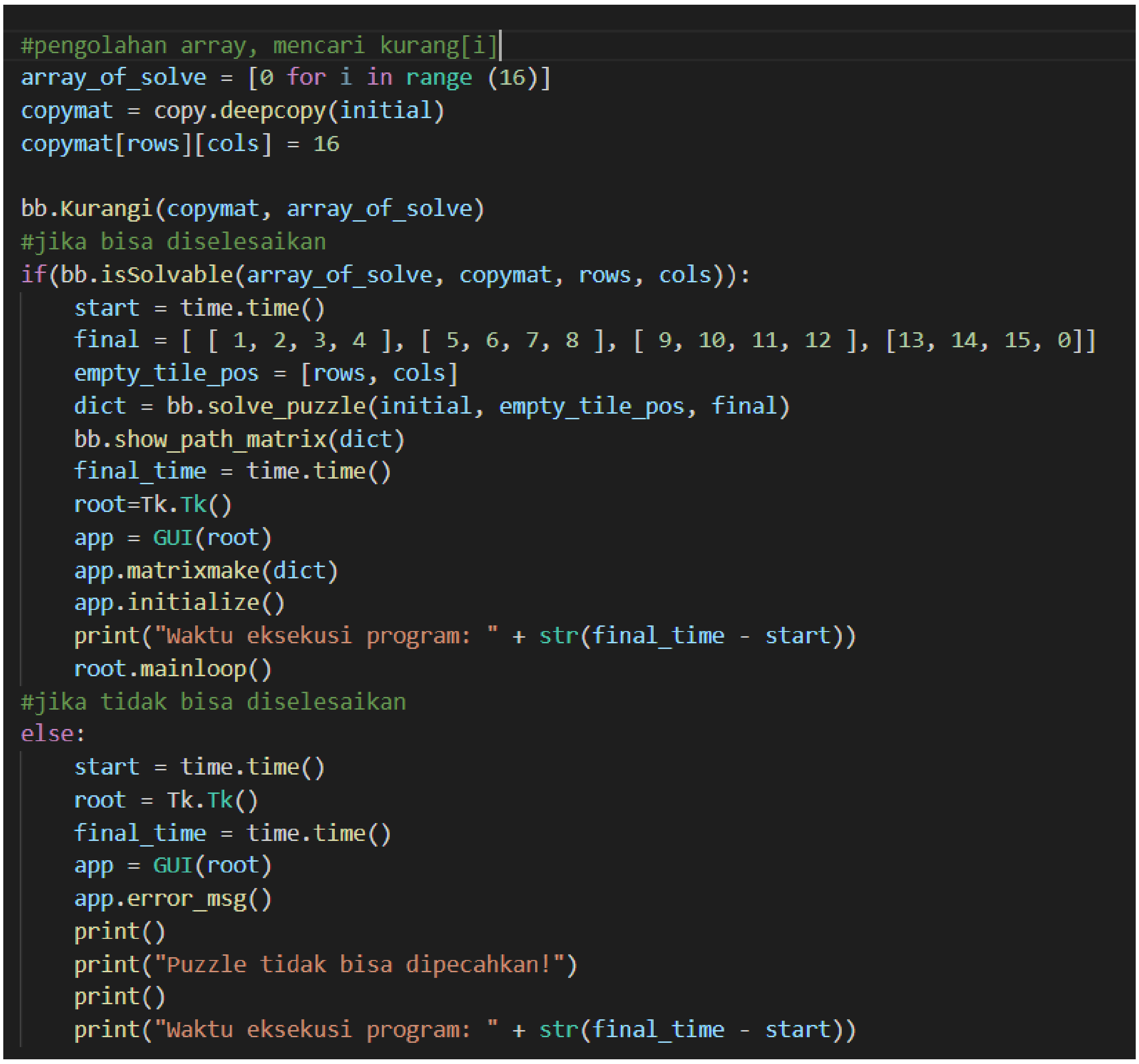
```
            matindex += 1
        else:
            messagebox.showinfo("state puzzle", "Puzzle berhasil ditemukan!")
        for j in range(0,self.a.shape[0]):
            for k in range(0,self.a.shape[1]):
                root.update()
                test_integer = self.matrix[matindex]
                integer = test_integer[j][k]
                if(integer != 0):
                    self.b = Tk.Button(self.frame, text = str(integer), height= 5, width=10, font = ('4'))
                else:
                    self.b = Tk.Button(self.frame, text = " ", height= 5, width=10, bg='blue', font = ('4'))
                self.b.grid(row=j,  column= k)
        self.btn1= Tk.Button(self.frame, text="Next" , command = lambda : self.changestate(matindex, len_matrix), font = ('Helvetica'))
        self.btn1.grid(row=4,  column=1)
        self.btn2= Tk.Button(self.frame, text="Previous", command = lambda : self.changestate2(matindex, len_matrix), font = ('Helvetica'))
        self.btn2.grid(row=4,  column=0)

    #melakukan change state jika user menekan tombol "previous"
    def changestate2(self, matindex, len_matrix):
        if(matindex > 1):
            matindex -= 1
        else:
            messagebox.showinfo("state puzzle", "Ini adalah state awal Puzzle!")
        for j in range(0,self.a.shape[0]):
            for k in range(0,self.a.shape[1]):
                root.update()
                test_integer = self.matrix[matindex]
                integer = test_integer[j][k]
                if(integer != 0):
                    self.b = Tk.Button(self.frame, text = str(integer), height= 5, width=10, font = ('4'))
```



```
                    self.b = Tk.Button(self.frame, text = str(integer), height= 5, width=10, font = ('4'))
                else:
                    self.b = Tk.Button(self.frame, text = " ", height= 5, width=10, bg='blue', font = ('4'))
                self.b.grid(row=j,  column= k)
        self.btn1= Tk.Button(self.frame, text="Next" , command = lambda : self.changestate(matindex, len_matrix), font = ('Helvetica'))
        self.btn1.grid(row=4,  column=1)
        self.btn2= Tk.Button(self.frame, text="Previous", command = lambda : self.changestate2(matindex, len_matrix), font = ('Helvetica'))
        self.btn2.grid(row=4,  column=0)

    #untuk memulai pemrosesan pembuatan GUI
    def initialize(self):
        self.parent.title("15-Puzzle Branch and Bound")
        self.parent.grid_rowconfigure(1,weight=1)
        self.parent.grid_columnconfigure(1,weight=1)
        root.geometry("450x450")

        self.frame = Tk.Frame(self.parent)
        self.frame.pack(fill=Tk.X, padx=1, pady=1)

        # Membuat array 4, 4
        self.a = np.zeros((4,4))
        len_matrix = len(self.matrix)
        index_matrix = 1
        for j in range(0,self.a.shape[0]):
            for k in range(0,self.a.shape[1]):
                test_integer = self.matrix[index_matrix]
                integer = test_integer[j][k]
                if(integer != 0):
                    self.b = Tk.Button(self.frame, text = str(integer), height= 5, width=10, font = ('4'))
                else:
                    self.b = Tk.Button(self.frame, text = " ", height= 5, width=10, bg='blue', font = ('4'))
```

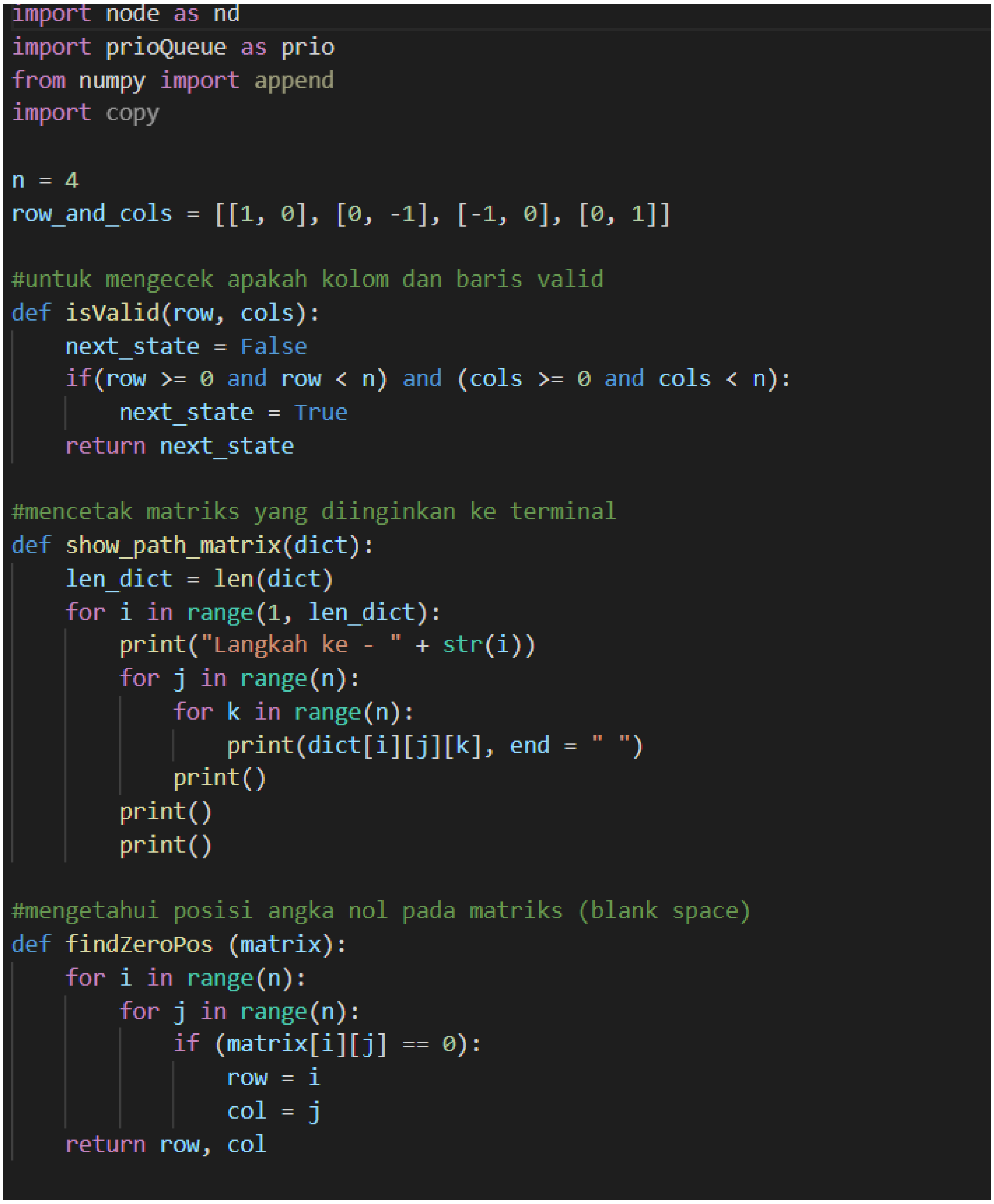


```python
                    self.b = Tk.Button(self.frame, text = "  ", height= 5, width=10, bg='blue', font = ('4'))
                self.b.grid(row=j,  column= k)
        self.btn1= Tk.Button(self.frame, text="Next" , command = lambda : self.changestate(index_matrix, len_matrix), font = ('Helvetica'))
        self.btn1.grid(row=4,  column=1)
        self.btn2= Tk.Button(self.frame, text="Previous", command = lambda : self.changestate2(index_matrix, len_matrix), font = ('Helvetica'))
        self.btn2.grid(row=4,  column=0)

    def error_msg(self):
        messagebox.showerror("Puzzle not available", "Puzzle tidak bisa dipecahkan!")

if __name__ == "__main__":
    print("        __             .----------.                                                                                          ")
    print("...-'  |`.        /           /                                                                 .---.                        ")
    print("|      |  |      /   ______.'             _________  _...._                                     |   |     __.....__          ")
    print("....   |  |     /   /_                    \        |.'       '-.                                |   |  .-''        '.        ")
    print("-|    | |    /       '''--.              \        .'```'.    '.                               |   | /     .-''''-. `.        ")
    print("|   |  |    '___          `.             \      |        \    \                              |   |/     /________\  \        ")
    print(" ...'   `--'        `'.        |              |     |         |    |_    _  .--------. .--------.|   ||                 |    ")
    print("|         |`.        )         |             |      \      /     .| '  / | |____    | |____    ||   |\    .-------------'    ")
    print("` -------\   |......-'        /              |     |\`'-.-'    .'.' | .' |     /   /      /   / |   | \    '-.____...---.    ")
    print("`---------' \          _..'`                |     | '-....-'`   /  |/  |    .'   /     .'   /  |   |  `.             .'      ")
    print("             '-----'''                    .'     '.           |   `'.  |  /    /___  /    /___'---'    `''-...... -'         ")
    print("                                         '----------'          '   .'|  '/|         ||         |                             ")
    print("                                                              `-'  `--' |_________||_________|                               ")
    filename = input("Masukkan nama path to file yang diinginkan:  ")
    initial = io.iomanage(filename)
    rows, cols = bb.findZeroPos(initial)

    #pengolahan array, mencari kurang]i
    array_of_solve = [0 for i in range (16)]
```

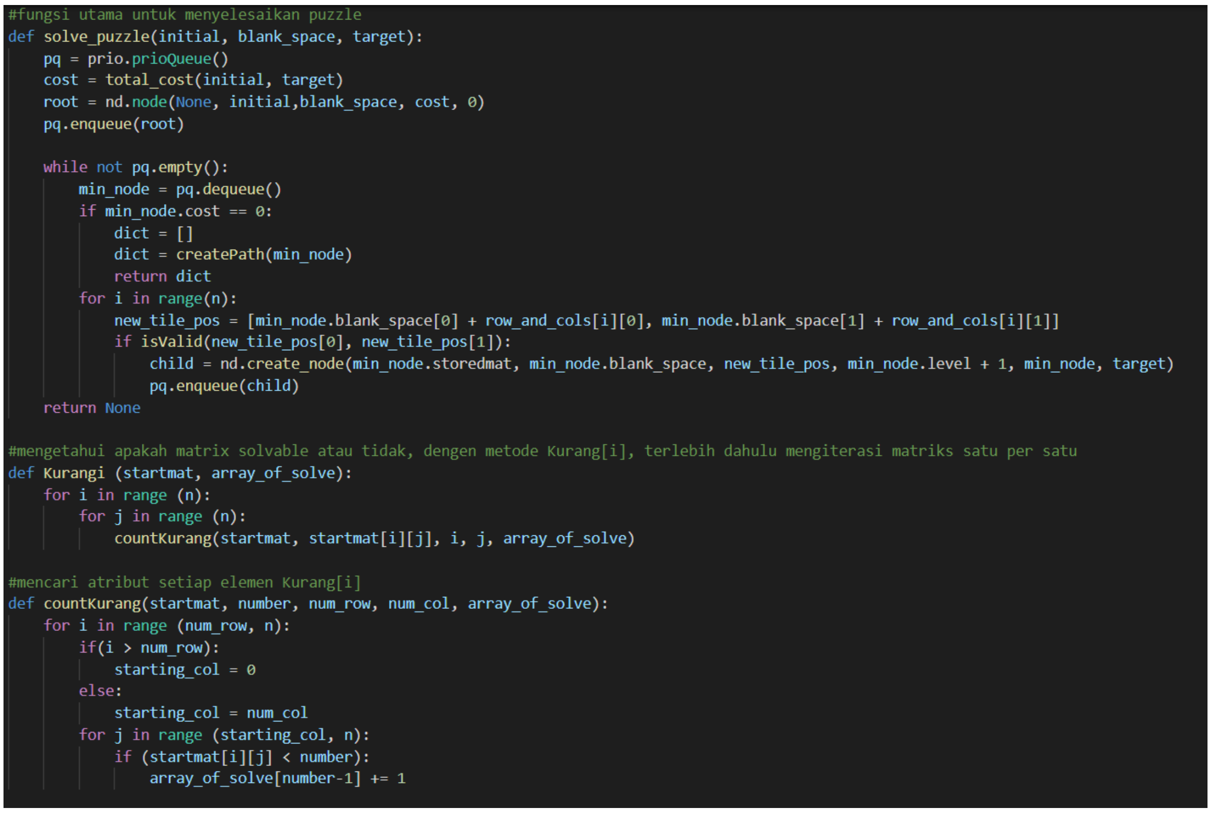


```python
#pengolahan array, mencari kurang[i]
array_of_solve = [0 for i in range (16)]
copymat = copy.deepcopy(initial)
copymat[rows][cols] = 16

bb.Kurangi(copymat, array_of_solve)
#jika bisa diselesaikan
if(bb.isSolvable(array_of_solve, copymat, rows, cols)):
    start = time.time()
    final = [ [ 1, 2, 3, 4 ], [ 5, 6, 7, 8 ], [ 9, 10, 11, 12 ], [13, 14, 15, 0]]
    empty_tile_pos = [rows, cols]
    dict = bb.solve_puzzle(initial, empty_tile_pos, final)
    bb.show_path_matrix(dict)
    final_time = time.time()
    root=Tk.Tk()
    app = GUI(root)
    app.matrixmake(dict)
    app.initialize()
    print("Waktu eksekusi program: " + str(final_time - start))
    root.mainloop()
#jika tidak bisa diselesaikan
else:
    start = time.time()
    root = Tk.Tk()
    final_time = time.time()
    app = GUI(root)
    app.error_msg()
    print()
    print("Puzzle tidak bisa dipecahkan!")
    print()
    print("Waktu eksekusi program: " + str(final_time - start))
```

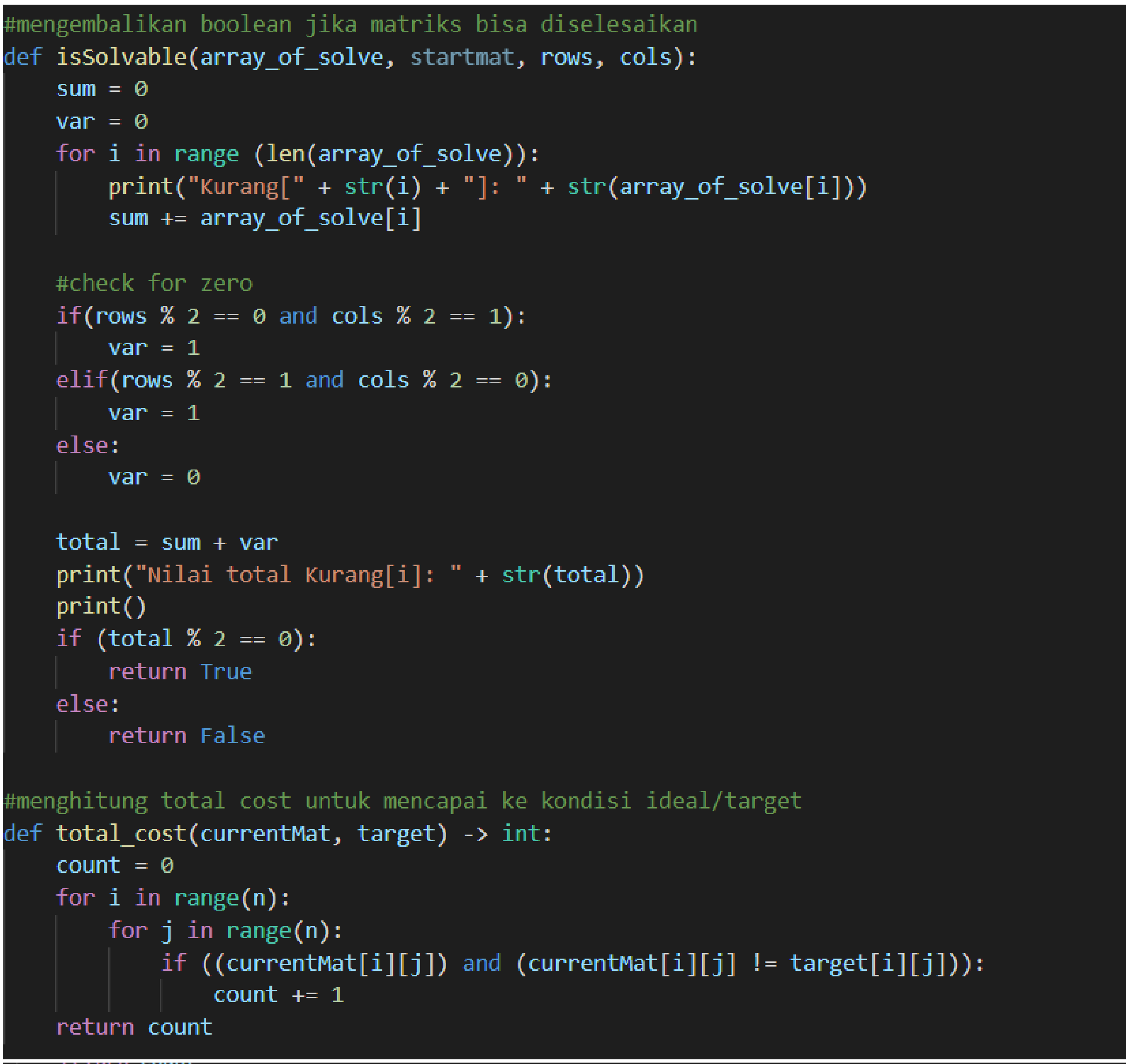
b. BranchandBound.py



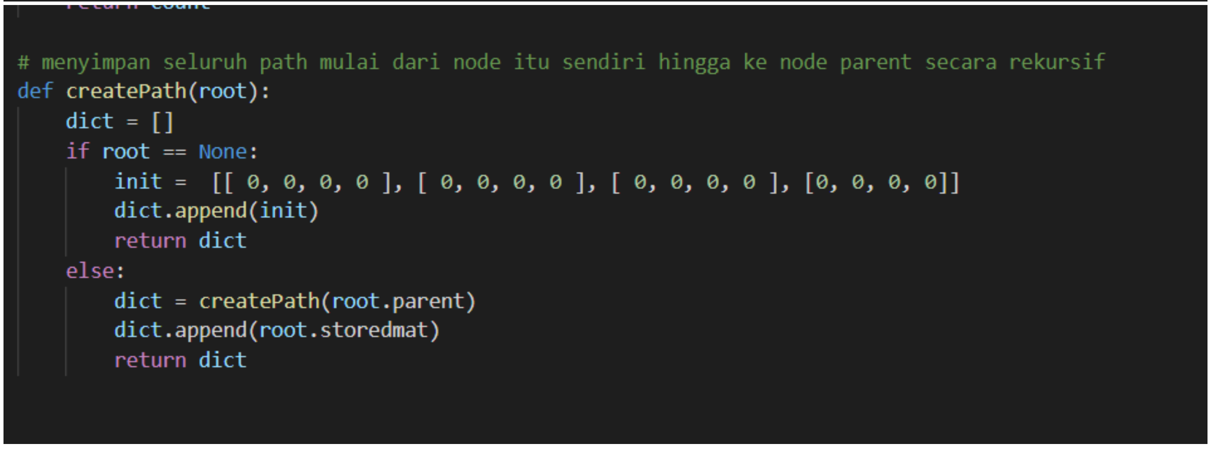
```python
import node as nd
import prioQueue as prio
from numpy import append
import copy

n = 4
row_and_cols = [[1, 0], [0, -1], [-1, 0], [0, 1]]

#untuk mengecek apakah kolom dan baris valid
def isValid(row, cols):
    next_state = False
    if(row >= 0 and row < n) and (cols >= 0 and cols < n):
        next_state = True
    return next_state

#mencetak matriks yang diinginkan ke terminal
def show_path_matrix(dict):
    len_dict = len(dict)
    for i in range(1, len_dict):
        print("Langkah ke - " + str(i))
        for j in range(n):
            for k in range(n):
                print(dict[i][j][k], end = " ")
            print()
        print()
        print()

#mengetahui posisi angka nol pada matriks (blank space)
def findZeroPos (matrix):
    for i in range(n):
        for j in range(n):
            if (matrix[i][j] == 0):
                row = i
                col = j
    return row, col
```



```
#fungsi utama untuk menyelesaikan puzzle
def solve_puzzle(initial, blank_space, target):
    pq = prio.prioQueue()
    cost = total_cost(initial, target)
    root = nd.node(None, initial,blank_space, cost, 0)
    pq.enqueue(root)

    while not pq.empty():
        min_node = pq.dequeue()
        if min_node.cost == 0:
            dict = []
            dict = createPath(min_node)
            return dict
        for i in range(n):
            new_tile_pos = [min_node.blank_space[0] + row_and_cols[i][0], min_node.blank_space[1] + row_and_cols[i][1]]
            if isValid(new_tile_pos[0], new_tile_pos[1]):
                child = nd.create_node(min_node.storedmat, min_node.blank_space, new_tile_pos, min_node.level + 1, min_node, target)
                pq.enqueue(child)
    return None

#mengetahui apakah matrix solvable atau tidak, dengen metode Kurang[i], terlebih dahulu mengiterasi matriks satu per satu
def Kurangi (startmat, array_of_solve):
    for i in range (n):
        for j in range (n):
            countKurang(startmat, startmat[i][j], i, j, array_of_solve)

#mencari atribut setiap elemen Kurang[i]
def countKurang(startmat, number, num_row, num_col, array_of_solve):
    for i in range (num_row, n):
        if(i > num_row):
            starting_col = 0
        else:
            starting_col = num_col
        for j in range (starting_col, n):
            if (startmat[i][j] < number):
                array_of_solve[number-1] += 1
```

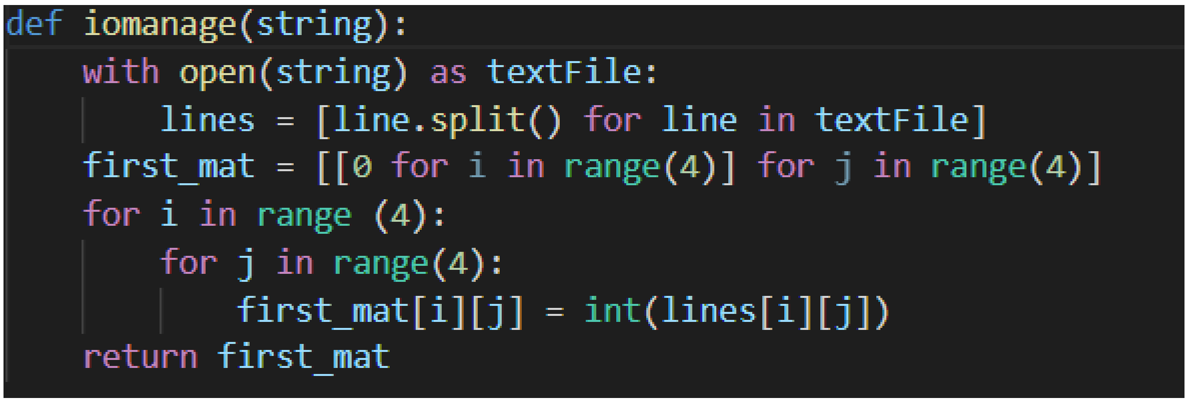


```python
#mengembalikan boolean jika matriks bisa diselesaikan
def isSolvable(array_of_solve, startmat, rows, cols):
    sum = 0
    var = 0
    for i in range (len(array_of_solve)):
        print("Kurang[" + str(i) + "]: " + str(array_of_solve[i]))
        sum += array_of_solve[i]

    #check for zero
    if(rows % 2 == 0 and cols % 2 == 1):
        var = 1
    elif(rows % 2 == 1 and cols % 2 == 0):
        var = 1
    else:
        var = 0

    total = sum + var
    print("Nilai total Kurang[i]: " + str(total))
    print()
    if (total % 2 == 0):
        return True
    else:
        return False

#menghitung total cost untuk mencapai ke kondisi ideal/target
def total_cost(currentMat, target) -> int:
    count = 0
    for i in range(n):
        for j in range(n):
            if ((currentMat[i][j]) and (currentMat[i][j] != target[i][j])):
                count += 1
    return count
```



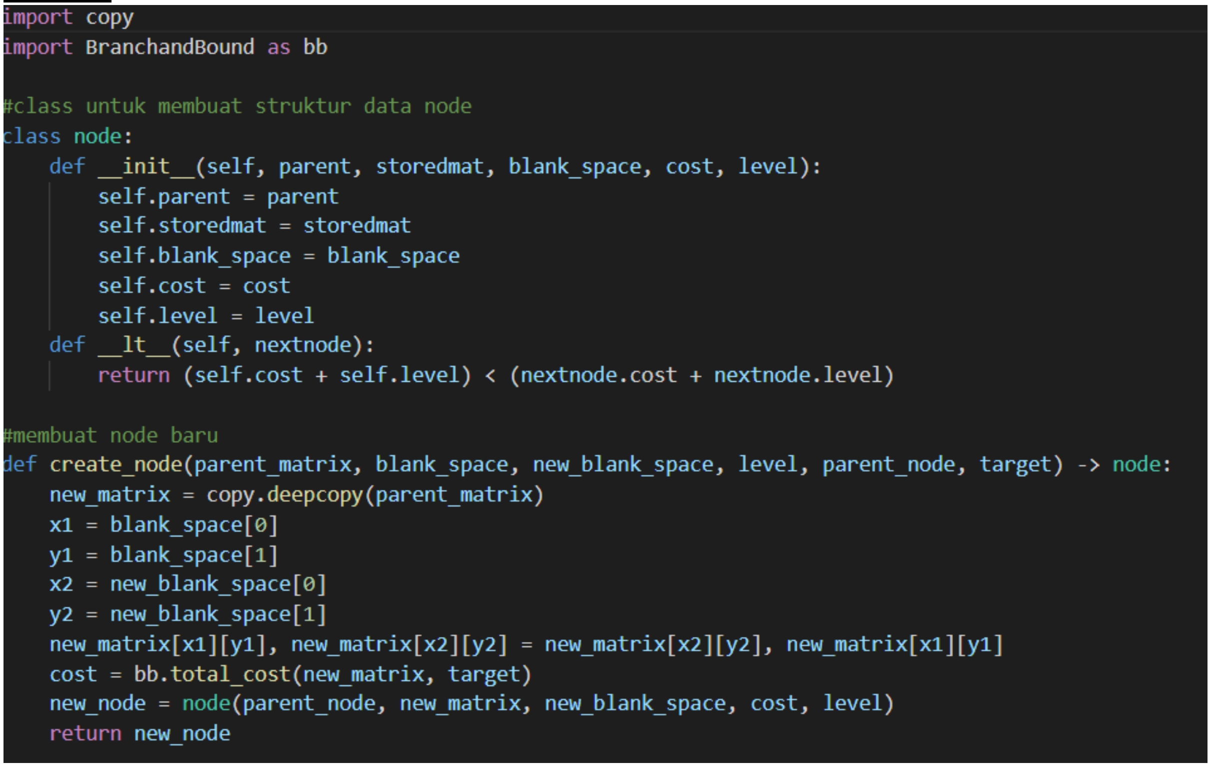
```python
# menyimpan seluruh path mulai dari node itu sendiri hingga ke node parent secara rekursif
def createPath(root):
    dict = []
    if root == None:
        init =  [[ 0, 0, 0, 0 ], [ 0, 0, 0, 0 ], [ 0, 0, 0, 0 ], [0, 0, 0, 0]]
        dict.append(init)
        return dict
    else:
        dict = createPath(root.parent)
        dict.append(root.storedmat)
        return dict
```

c. **iomanager.py**



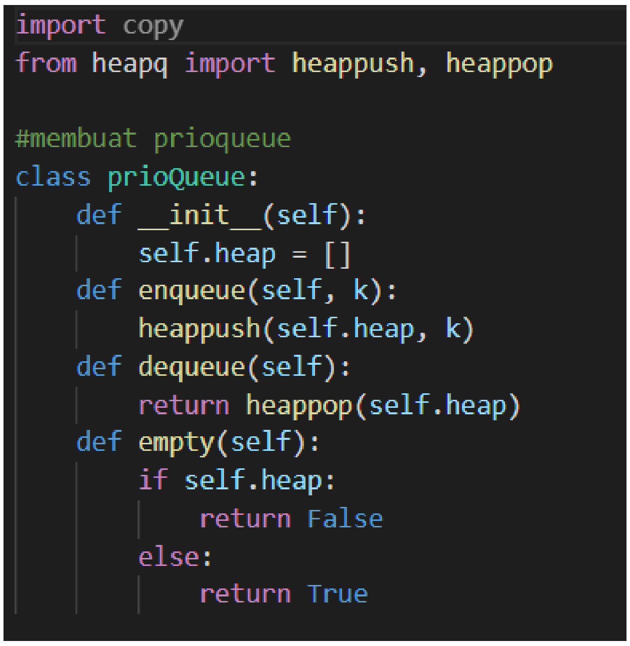
```python
def iomanage(string):
    with open(string) as textFile:
        lines = [line.split() for line in textFile]
    first_mat = [[0 for i in range(4)] for j in range(4)]
    for i in range (4):
        for j in range(4):
            first_mat[i][j] = int(lines[i][j])
    return first_mat
```

d. node.py



```python
import copy
import BranchandBound as bb

#class untuk membuat struktur data node
class node:
    def __init__(self, parent, storedmat, blank_space, cost, level):
        self.parent = parent
        self.storedmat = storedmat
        self.blank_space = blank_space
        self.cost = cost
        self.level = level
    def __lt__(self, nextnode):
        return (self.cost + self.level) < (nextnode.cost + nextnode.level)

#membuat node baru
def create_node(parent_matrix, blank_space, new_blank_space, level, parent_node, target) -> node:
    new_matrix = copy.deepcopy(parent_matrix)
    x1 = blank_space[0]
    y1 = blank_space[1]
    x2 = new_blank_space[0]
    y2 = new_blank_space[1]
    new_matrix[x1][y1], new_matrix[x2][y2] = new_matrix[x2][y2], new_matrix[x1][y1]
    cost = bb.total_cost(new_matrix, target)
    new_node = node(parent_node, new_matrix, new_blank_space, cost, level)
    return new_node
```

e. prioQueue.py



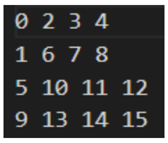
```
import copy
from heapq import heappush, heappop

#membuat prioqueue
class prioQueue:
    def __init__(self):
        self.heap = []
    def enqueue(self, k):
        heappush(self.heap, k)
    def dequeue(self):
        return heappop(self.heap)
    def empty(self):
        if self.heap:
            return False
        else:
            return True
```

3. Screenshot Pengujian Kode

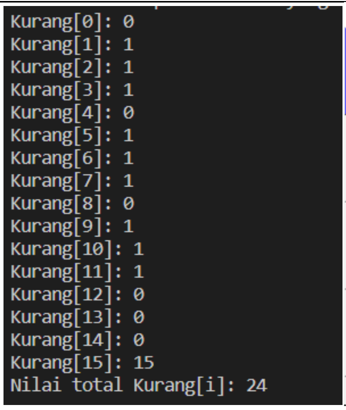
Link berkas testcase (menggunakan akun std):
https://drive.google.com/drive/folders/1_vXsZaTmIpu9cgofOm6oFgBCblazOc8G?usp=sharing

TestCase	Screenshot (GUI + Output)

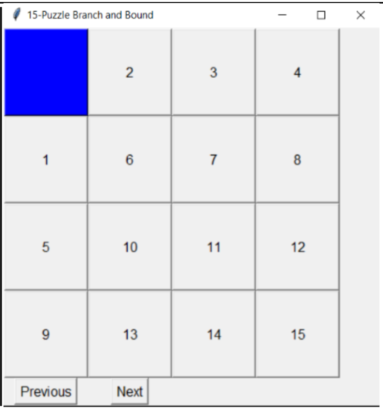
Solved1.txt

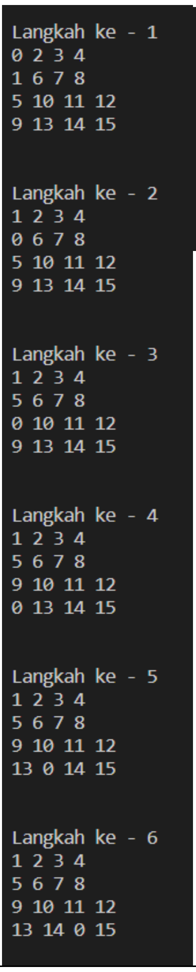


```
0 2 3 4
1 6 7 8
5 10 11 12
9 13 14 15
```

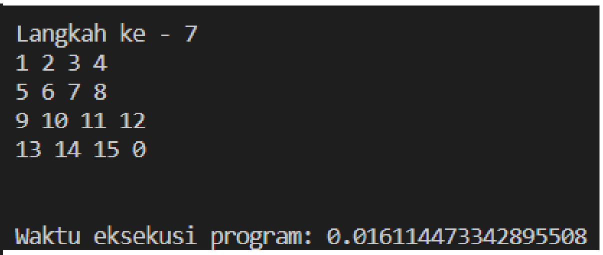


```
Kurang[0]: 0
Kurang[1]: 1
Kurang[2]: 1
Kurang[3]: 1
Kurang[4]: 0
Kurang[5]: 1
Kurang[6]: 1
Kurang[7]: 1
Kurang[8]: 0
Kurang[9]: 1
Kurang[10]: 1
Kurang[11]: 1
Kurang[12]: 0
Kurang[13]: 0
Kurang[14]: 0
Kurang[15]: 15
Nilai total Kurang[i]: 24
```



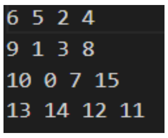


```
Langkah ke - 1
0 2 3 4
1 6 7 8
5 10 11 12
9 13 14 15


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


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


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


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


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

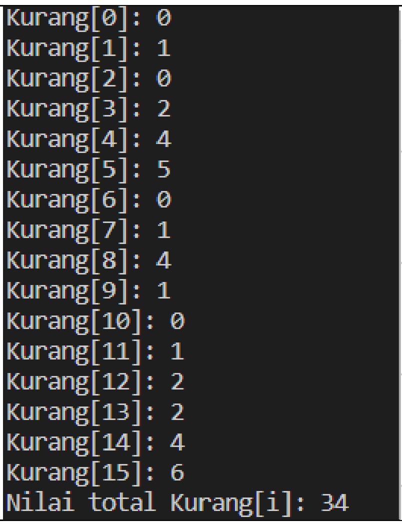


```
Langkah ke - 7
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0


Waktu eksekusi program: 0.016114473342895508
```

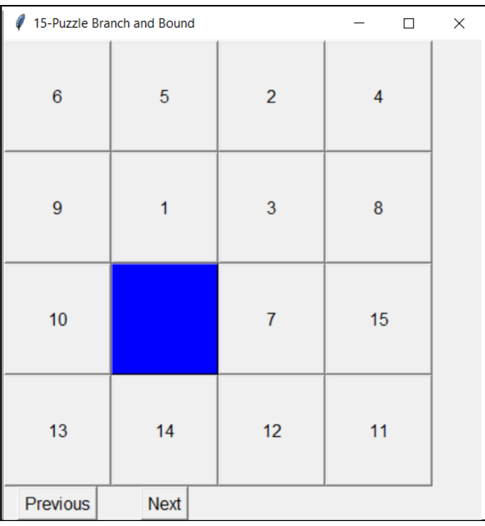
Solved2.txt

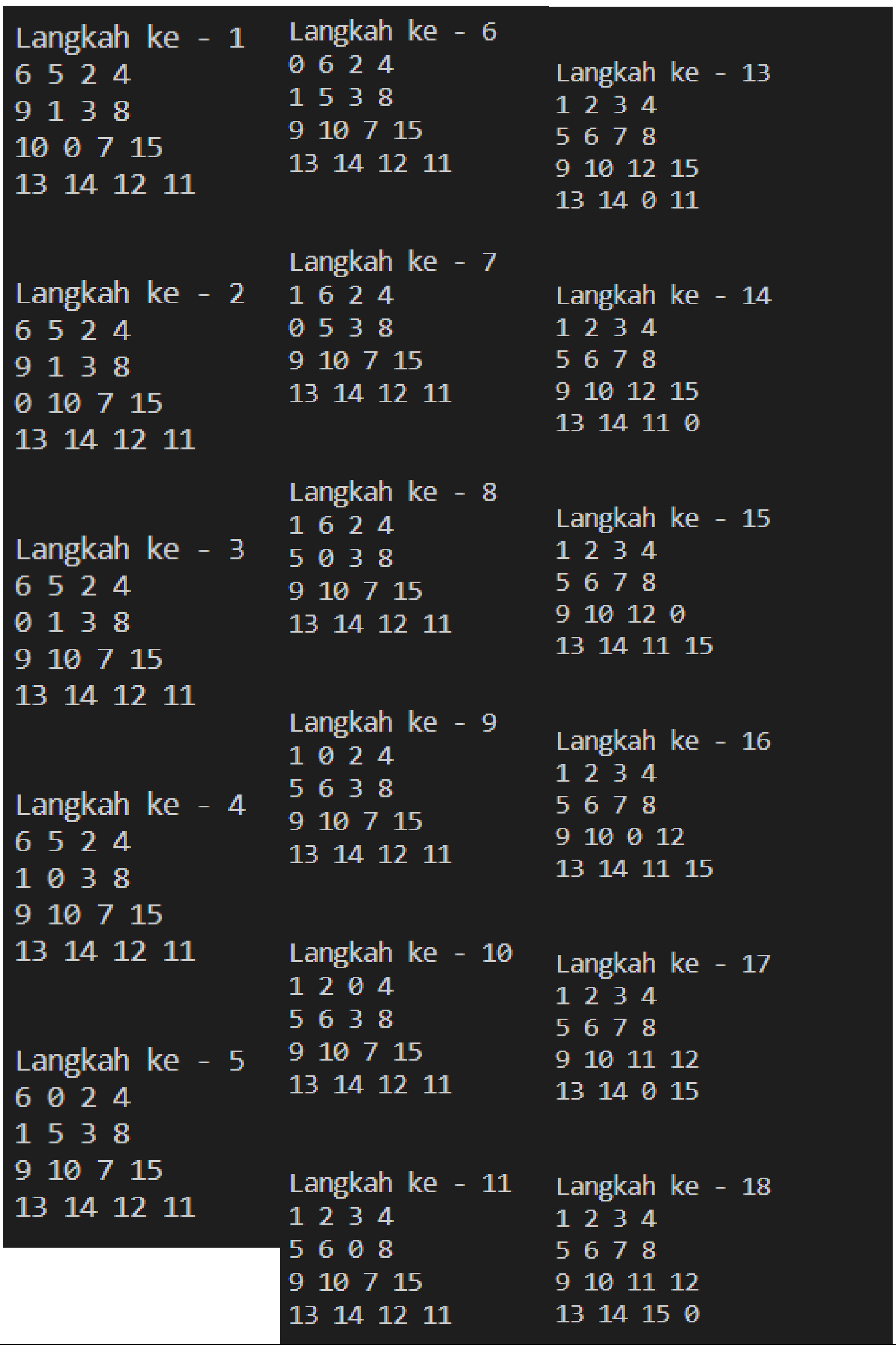


```
6 5 2 4
9 1 3 8
10 0 7 15
13 14 12 11
```



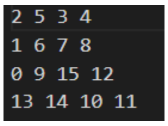
```
Kurang[0]: 0
Kurang[1]: 1
Kurang[2]: 0
Kurang[3]: 2
Kurang[4]: 4
Kurang[5]: 5
Kurang[6]: 0
Kurang[7]: 1
Kurang[8]: 4
Kurang[9]: 1
Kurang[10]: 0
Kurang[11]: 1
Kurang[12]: 2
Kurang[13]: 2
Kurang[14]: 4
Kurang[15]: 6
Nilai total Kurang[i]: 34
```



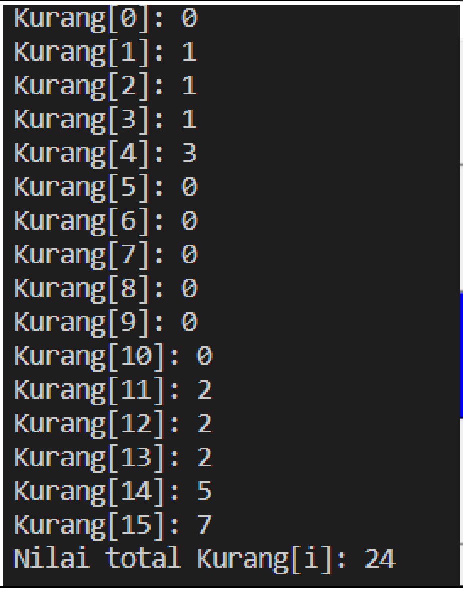


```
Langkah ke - 1
6 5 2 4
9 1 3 8
10 0 7 15
13 14 12 11

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

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

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

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

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

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

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

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

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

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

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

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

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

Langkah ke - 16
1 2 3 4
5 6 7 8
9 10 0 12
13 14 11 15

Langkah ke - 17
1 2 3 4
5 6 7 8
9 10 11 12
13 14 0 15

Langkah ke - 18
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0
```

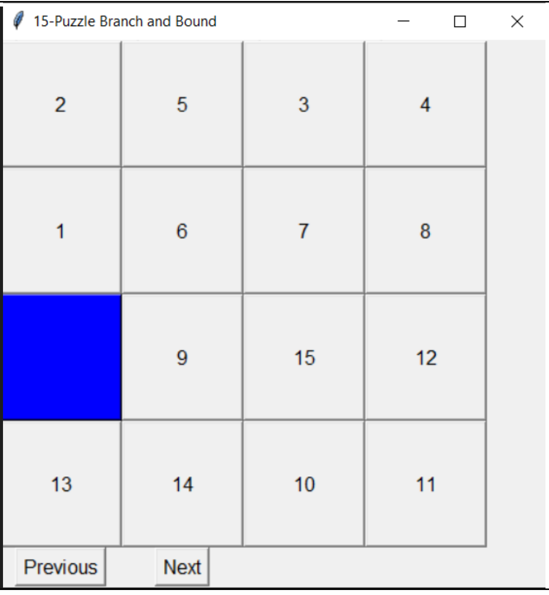
Solved3.txt



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

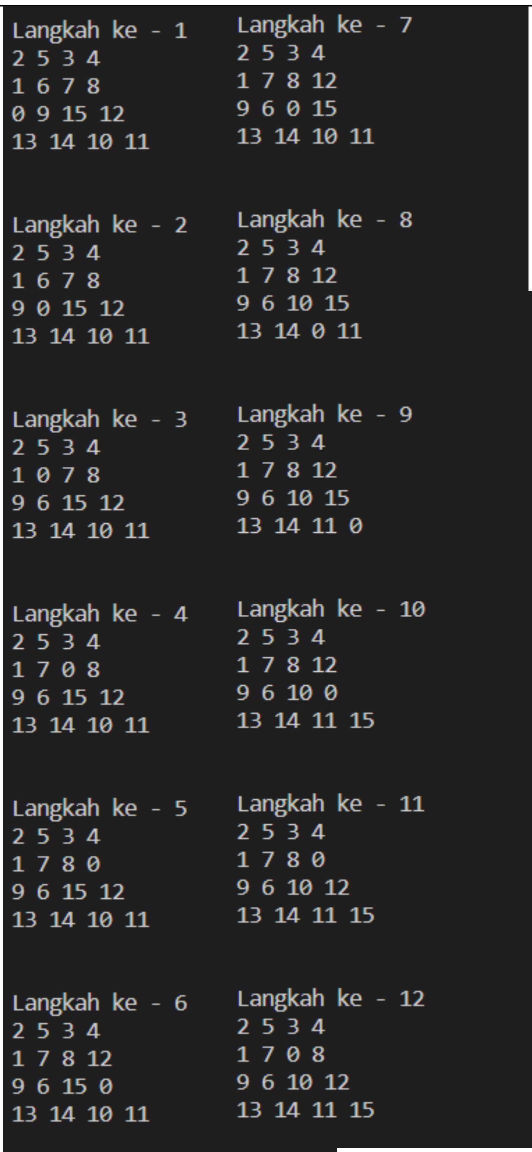
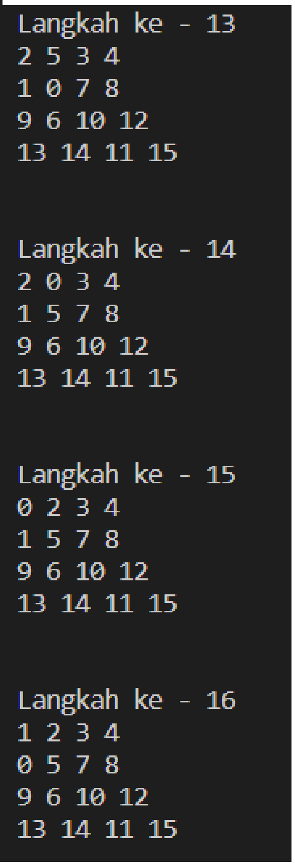
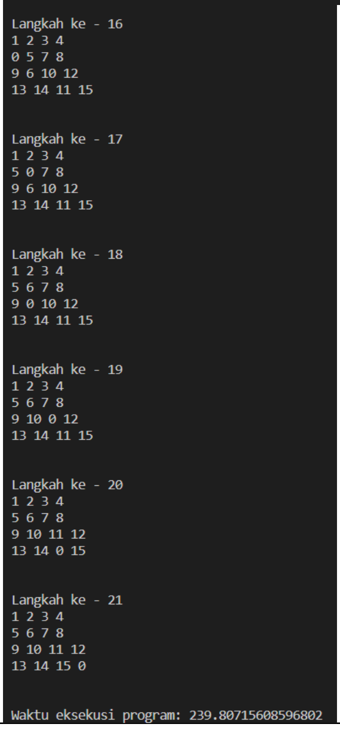


```
Kurang[0]: 0
Kurang[1]: 1
Kurang[2]: 1
Kurang[3]: 1
Kurang[4]: 3
Kurang[5]: 0
Kurang[6]: 0
Kurang[7]: 0
Kurang[8]: 0
Kurang[9]: 0
Kurang[10]: 0
Kurang[11]: 2
Kurang[12]: 2
Kurang[13]: 2
Kurang[14]: 5
Kurang[15]: 7
Nilai total Kurang[i]: 24
```



15-Puzzle Branch and Bound

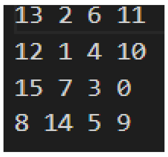
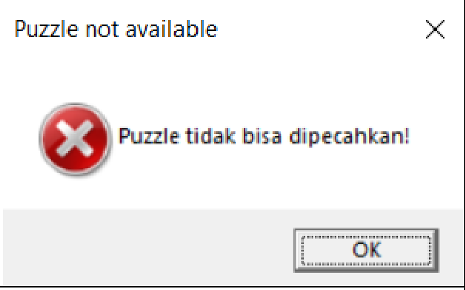
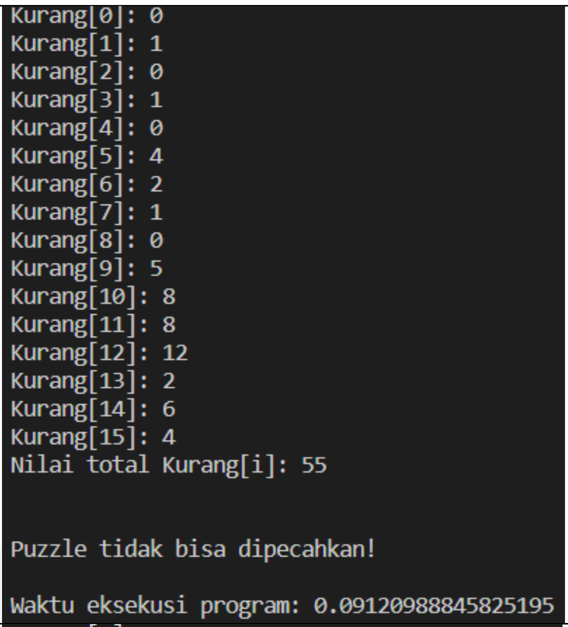
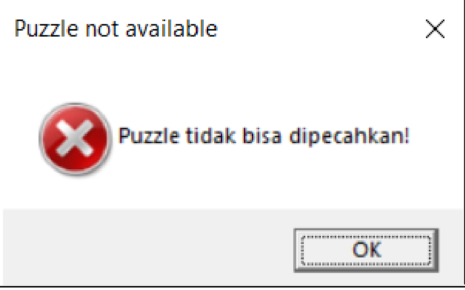
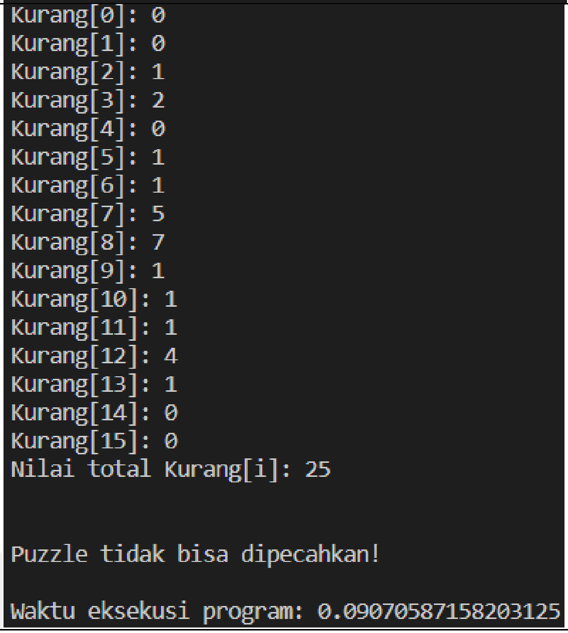
2	5	3	4
1	6	7	8
	9	15	12
13	14	10	11

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


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


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


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


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


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


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


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


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


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


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


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

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


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


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


Langkah ke - 16
1 2 3 4
0 5 7 8
9 6 10 12
13 14 11 15
```

```
Langkah ke - 16
1 2 3 4
0 5 7 8
9 6 10 12
13 14 11 15


Langkah ke - 17
1 2 3 4
5 0 7 8
9 6 10 12
13 14 11 15


Langkah ke - 18
1 2 3 4
5 6 7 8
9 0 10 12
13 14 11 15


Langkah ke - 19
1 2 3 4
5 6 7 8
9 10 0 12
13 14 11 15


Langkah ke - 20
1 2 3 4
5 6 7 8
9 10 11 12
13 14 0 15


Langkah ke - 21
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0


Waktu eksekusi program: 239.80715608596802
```

Unsolved1.txt 		
Unsolved2.txt		

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

Puzzle not available

Puzzle tidak bisa dipecahkan!

OK

```
Kurang[0]: 0
Kurang[1]: 1
Kurang[2]: 0
Kurang[3]: 1
Kurang[4]: 0
Kurang[5]: 4
Kurang[6]: 2
Kurang[7]: 1
Kurang[8]: 0
Kurang[9]: 5
Kurang[10]: 8
Kurang[11]: 8
Kurang[12]: 12
Kurang[13]: 2
Kurang[14]: 6
Kurang[15]: 4
Nilai total Kurang[i]: 55


Puzzle tidak bisa dipecahkan!

Waktu eksekusi program: 0.09120988845825195
```

Puzzle not available

Puzzle tidak bisa dipecahkan!

OK

```
Kurang[0]: 0
Kurang[1]: 0
Kurang[2]: 1
Kurang[3]: 2
Kurang[4]: 0
Kurang[5]: 1
Kurang[6]: 1
Kurang[7]: 5
Kurang[8]: 7
Kurang[9]: 1
Kurang[10]: 1
Kurang[11]: 1
Kurang[12]: 4
Kurang[13]: 1
Kurang[14]: 0
Kurang[15]: 0
Nilai total Kurang[i]: 25


Puzzle tidak bisa dipecahkan!

Waktu eksekusi program: 0.09070587158203125
```

4. LAMPIRAN

Link github: https://github.com/KorbanFidas2A/Tucil3_13520007.git

https://github.com/KorbanFidas2A/Tucil3_13520007

Poin	Yes	No
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	√	