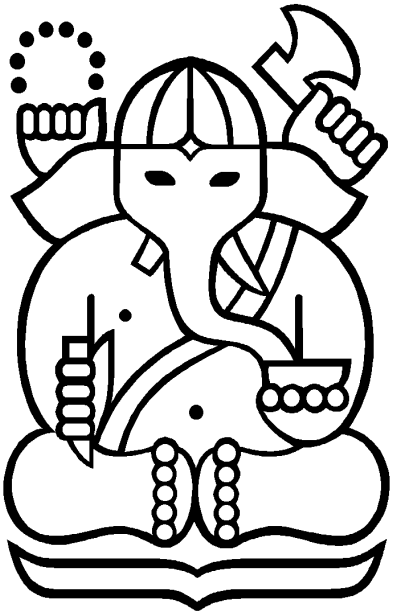
LAPORAN TUGAS KECIL 3

IF2211 Strategi Algoritma

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



Disusun Oleh:

Aira Thalca Avila Putra

13520101

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

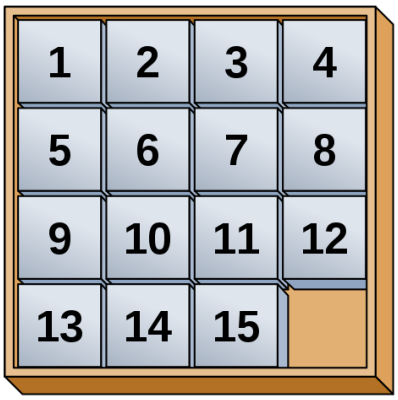
BAB I
PENYELESAIAN 15-PUZZLE ALGORITMA *BRANCH AND BOUND*

Algoritma Branch and Bound (B&B) merupakan Algoritma yang digunakan untuk persoalan optimasi, yakni meminimalkan/memaksimalkan suatu fungsi objektif, yang tidak melanggar batasan (constraint) persoalan. Cara kerja Algoritma B&B ini biasanya menggunakan ilustrasi sebuah tree, dengan setiap node memiliki sebuah nilai cost:

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

Kemudian, node berikutnya yang akan di-expand tidak lagi berdasarkan urutan pembangkitannya, tetapi simpul yang memiliki cost terkecil (least cost search) – pada kasus minimasi dan cost terkecil untuk kasus memaksimalkan suatu hasil. Pada algoritma Branch and Bound yang pencarian solusinya tidak hanya satu (mencari semua solusi), biasanya ada funsgi pembatas untuk menghentikan pencarian pada jalur yang tidak bisa mengarah pada solusi. Biasanya penghentian ini terjadi Ketika nilai simpul saat ini sudah tidak bisa lebih baik dari solusi terbaik yang sudah didapat maupun ada batasan-batasan tertentu sehingga simpul saat ini tidak bisa membentuk solusi yang sesuai dengan persoalan.

15-Puzzle adalah salah satu permainan dengan menggunakan papan yang berisikan angka 1 sampai 15 dalam 16 bagian ubin. Terdapat satu ubin kosong yang dapat digerakkan ke atas, bawah, kiri, dan kanan untuk menggeser ubin lainnya. Tujuan yang dicapai dari permainan ini adalah menyusun angka 1 sampai 15 terurut dari atas ke bawah dengan cara menggeser ubin kosong.



Gambar 1. *Contoh* 15 Puzzle

Untuk menyelsaikan permainan 15-Puzzle yang memiliki susunan acak, terlebih dahulu perlu dilakukan pengecekan apakah susunan yang menjadi persoalan dapat diselesaikan atau tidak. Pengecekan dilakukan dengan menghtiung nilai dari $\sum_{i=1}^{16} KURANG(i) + X$. Nilai dari KURANG(i) dihitung dengan cara menghtiung banyaknya $j \neq i$ dan memenuhi dua kondisi, yakni $A[i] > A[j]$ tetapi $j < i$. Nilai X adalah letak ubin kosong pada susunan awal. Misalkan

X berada pada kolom ke-i dan baris ke-j. X bernilai 1 jika i+j adalah ganjil dan bernilai 0 pada jika i+j adalah genap. Jika nilai dari $\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap, maka susunan ubin tersebut dapat diselesaikan dan jika nilainya ganjil maka tidak bisa diselesaikan. Apabila susunan ubin dapat diselesaikan, maka akan dilakukan proses pencarian simpul tujuan dari simpul utama. Contoh penerapan teorema di atas adalah sebagai berikut

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

$$\sum_{i=1}^{16} Kurang(i) + X = 37 + 0 = 37$$

Gambar 2. Kasus ubin yang *not solveable*

Persoalan 15-Puzzle diselesaikan dengan menggunakan algoritma *Branch and Bound* dengan cara sebagai berikut:

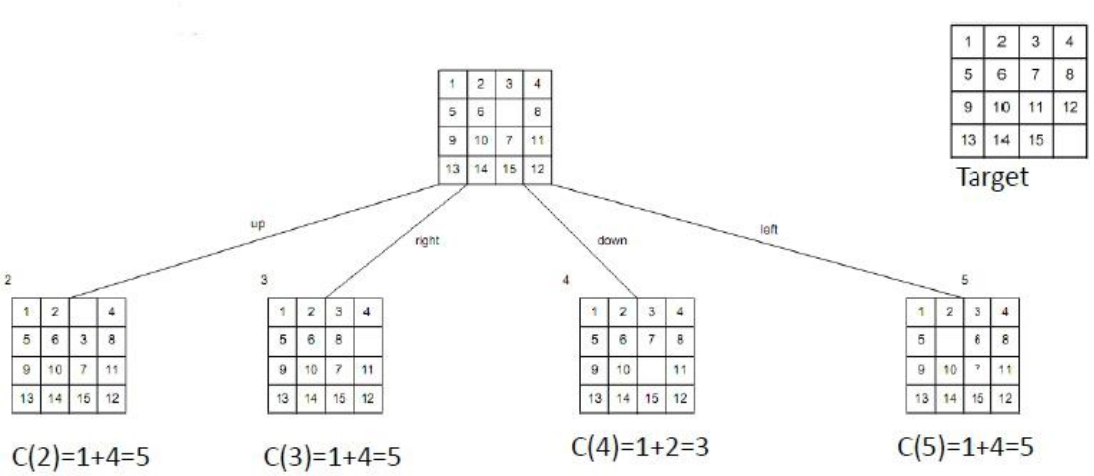
1. Masukkan *node root(*dalam hal ini susunan ubin awal) ke *Priority Queue* dan hashmap sebagai penanda bahwa node ini pernah dikunjungi. Priority Queue dalam persoalan ini memprioritaskan Node dengan cost terkecil (*least cost search).*
2. Jika *Priority Queue* kosong, hentikan pencarian.
3. Selama *Priority Queue* tidak kosong ambil node dari *Priority Queue* (dalam hal ini node dengan *cost* terkecil dan hapus dari *Priority Queue.*
4. Cek apakah node saat ini sama dengan goal, jika sama, hentikan pencarian dan kembalikan node ini sebagai solusi juga jumlah node yang dibangkitkan.
5. Jika node saat ini bukan goal, bangkitkan semua anak dari node saat ini yang belum pernah dibangkitkan (jika sudah ada di hashmap, maka tidak perlu dibangkitkan).
6. Hitung nilai *cost* untuk setiap *child* yang dibangkitkan, perhitungan menggunakan rumus sebagai berikut

   $$c(\mathrm{i}) = f(\mathrm{i}) + g(\mathrm{i})$$

   Dengan c(i) adalah nilai *cost* dari node tersebut (yang menjadi perbandingan prioritas masing-masing node pada *Priority Queue*). f(i) dan g(i) masing-masing adalah panjang lintasan dari *root* ke node saat ini dan taksiran panjang lintasan terpendek dari node saat ini ke goal. Taksiran ini dihitung dengan mencari banyaknya ubin tidak kosong yang belum berada pada tempat yang sesuai dengan goal.

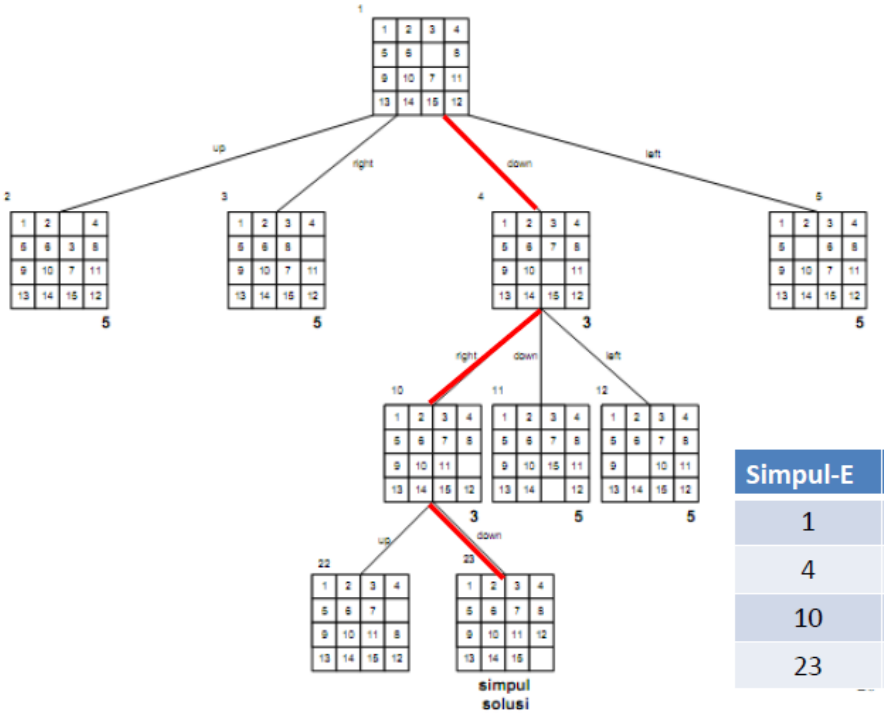
7. Masukkan semua *child* yang baru dibangkitkan tersebut ke *Priority Queue*.
8. Ulangi langkah ke 2 hingga node yang diambil dari *Priority Queue* adalah goal.

Perhitungan nilai cost dapat dilihat pada contoh berikut:



Gambar 3. Perhitungan nilai cost setiap node

Karena node 4 memiliki nilai cost terkecil, maka anak dari node 4 akan dibangkitkan seperti pada gambar dibawah ini.



Simpul-E	Simpul Hidup
1	4,2,3,5
4	10,2,3,5,11,12
10	23,2,3,5,11,12,22
23	Solusi ketemu

Gambar 4. Pohon ruang status

BAB II
SOURCE CODE PROGRAM

1. Library puzzle.py

```
from queue import PriorityQueue
import numpy as np

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

class Puzzle:
    def __init__(self, matrix = np.copy(final), empty_x = 3, empty_y = 3,
cost = 0, level = 0, parent = None, path = None):
        self.matrix = np.copy(matrix)
        self.empty_x = empty_x
        self.empty_y = empty_y
        self.cost = cost
        self.level = level
        self.parent = parent
        self.path = path

    def __lt__(self, other):
        if (self.cost + self.level == other.cost + other.level):
            return self.cost <= other.cost
        return self.cost + self.level < other.cost + other.level

    def Cost(self):
        result = 0
        for i in range(4):
            for j in range(4):
                current = self.matrix[i][j]
                if(current != 16 and current != final[i][j]):
                    result += 1
        return result

    def findNumber(self, number):
        ans = [-1,-1]
        for i in range(4):
            for j in range(4):
                if self.matrix[i][j] == number:
                    ans = [i,j]
        return ans

    #calculate Kurang(i)
    def Kurang(self, number):
        if(number == 16):
            location = [self.empty_x,self.empty_y]
        else:
            location = self.findNumber(number)
```

```
        res = 0
        for i in range(location[0]*4+location[1]+1,16):
            if self.matrix[i//4][i%4] <
self.matrix[location[0]][location[1]]:
                res += 1
        return res

    #calculate sigma Kurang(i) + X to
    def KurangTotal(self):
        res = 0
        for i in range(1,17):
            res += self.Kurang(i)
        if (self.empty_x + self.empty_y) % 2 != 0:
            res += 1
        return res

    #function to check if matrix solveable or not
    def Cek(self):
        return (self.KurangTotal() % 2 == 0)

    #check if matrix valid (contain all number from 1...16 exactly once)
    def Valid(self):
        countInput = [0 for i in range(17)]
        for i in range(4):
            for j in range(4):
                if (self.matrix[i][j] > 16 or self.matrix[i][j] <= 0):
                    return False
                else :
                    countInput[self.matrix[i][j]] += 1
        for i in range(1,17):
            if (countInput[i] != 1):
                return False
        return True

    #function to read the input from file and move it to attribute matrix
    def readFile(self, file):
        try:
            f = open(file, "r")
            for i in range(4):
                temp = f.readline()
                self.matrix[i] = temp.split()
                for j in range(4):
                    self.matrix[i][j] = int(self.matrix[i][j])
                    if (self.matrix[i][j] == 16):
                        self.empty_x = i
                        self.empty_y = j
            self.matrix = np.array(self.matrix)
            self.cost = self.Cost()
```

```
        except:
            pass

    #function to read the input from console and move it to attribute
matrix
    def inputCommand(self, input):
        try:
            inputmatrix = []
            inputmatrix = input.split()
            for i in range(16):
                self.matrix[i//4][i%4] = int(inputmatrix[i])
                if (self.matrix[i//4][i%4] == 16):
                        self.empty_x = i//4
                        self.empty_y = i%4
            self.matrix = np.array(self.matrix)
            self.cost = self.Cost()
        except:
            pass

    #function to move the puzzle
    def Move(self, x, y):
        self.matrix[self.empty_x][self.empty_y],
self.matrix[self.empty_x+x][self.empty_y+y] =
self.matrix[self.empty_x+x][self.empty_y+y],
self.matrix[self.empty_x][self.empty_y]
        self.empty_x += x
        self.empty_y += y

    #function to generate new Puzzle (child) and calculate the cost
    def MakeChild(self, move_x, move_y, path):
        child = Puzzle(self.matrix, self.empty_x, self.empty_y, self.cost,
self.level + 1, self, path)
        if(child.matrix[child.empty_x + move_x][child.empty_y + move_y] ==
(child.empty_x + move_x) * 4 + child.empty_y + move_y + 1):
            child.cost += 1
        elif(child.matrix[child.empty_x + move_x][child.empty_y + move_y]
== (child.empty_x) * 4 + child.empty_y + 1):
            child.cost -= 1
        child.Move(move_x, move_y)
        return child

    def Up(self):
        if(self.empty_x - 1 >= 0):
            return self.MakeChild(-1,0,"Up")
        return None

    def Down(self):
        if(self.empty_x + 1 < 4):
```

```
            return self.MakeChild(1,0,"Down")
        return None

    def Left(self):
        if(self.empty_y - 1 >= 0):
            return self.MakeChild(0,-1,"Left")
        return None

    def Right(self):
        if(self.empty_y + 1 < 4):
            return self.MakeChild(0,1,"Right")
        return None

def newNode(child, Visited, bangkit, Q):
    #check if child node already visited, if visited, dont enqueue to
PrioQueue
    matrix_hash = child.matrix.tobytes()
    if(not matrix_hash in Visited):
        bangkit += 1
        Visited[matrix_hash] = 1
        Q.put(child)
    return Visited, bangkit, Q

#function to solve puzzle using Branch and Bound
def SolvePuzzle(root, iterationnumber):
    Visited = {} #to check if node is already visited
    Q = PriorityQueue()
    iterationnumber = 1
    matrix_hash = root.matrix.tobytes()
    Visited[matrix_hash] = 1
    Q.put(root)
    while (not(Q.empty())): #while Queue still has node to check, dequeue
from the queue
        now = Q.get()
        if (np.array_equal(final, now.matrix)): #if node now equal goal
node, finish the algorithm
            return now, iterationnumber
            break
        else:
            childUp = now.Up() #generate all child possible from node now
            childDown = now.Down()
            childLeft = now.Left()
            childRight = now.Right()
            if (childUp != None):
                Visited, iterationnumber, Q = newNode(childUp, Visited,
iterationnumber, Q)
            if (childDown != None):
```

```
                Visited, iterationnumber, Q = newNode(childDown, Visited,
iterationnumber, Q)
            if (childLeft != None):
                Visited, iterationnumber, Q = newNode(childLeft, Visited,
iterationnumber, Q)
            if (childRight != None):
                Visited, iterationnumber, Q = newNode(childRight, Visited,
iterationnumber, Q)
```

2. Library grid4x4.py

```
from puzzle import *
from time import sleep
from tkinter import *
from tkinter import filedialog as fd

class gridFrame(Frame):
    def __init__(self, window, puzzle):
        super().__init__(window)
        self.puzzle = puzzle
        self.buttons = [None] * 16
        self.create_grid()
        self.pack(pady=5, padx=15, side=LEFT)

    def create_grid(self):
        for i in range(4):
            for j in range(4):
                self.create(4*i + j, i, j)

    def create(self, index, i, j):
        if (self.puzzle.matrix[i][j] == 16):
            txttoplace = ""
        else:
            txttoplace = str(self.puzzle.matrix[i][j])
        self.buttons[index] = Label(self,bg="dodger blue", borderwidth = 2,
relief  = 'solid', text = txttoplace, justify = LEFT, font=("Comic Sans
MS", "11"), width=7, height=3)
        self.buttons[index].grid(row=i, column=j)

    def change_button(self):
        x = 0
        for i in self.buttons:
            if (self.puzzle.matrix[x//4][x%4] == 16):
                txttoplace = ""
            else:
                txttoplace = str(self.puzzle.matrix[x//4][x%4])
            i.configure(text = txttoplace)
            x += 1
```

```
    def change(self, matrix):
        x = 0
        for i in self.buttons:
            if (matrix[x//4][x%4] == 16):
                txttoplace = ""
            else:
                txttoplace = str(matrix[x//4][x%4])
            i.configure(text = txttoplace)
            x += 1

    def path(self, solution, window):
        if(solution.parent is None):
            return
        else:
            self.path(solution.parent, window)
            self.change(solution.matrix)
            sleep(0.2)
            window.update()
```

3. main.py

```
from grid4x4 import *
from time import perf_counter

#function to solve matrix after button Solve pressed
def solve():
    global iterationnumber
    global Puz
    global window
    tStart = perf_counter()
    if (not Puz.Valid()):
        text.delete("1.0","end")
        text.insert("1.0", "Input Tidak Valid")
        text.tag_add("tag_name", "1.0", "end")
    else:
        text.delete("1.0","end")
        for i in range(5):
            check = 3*i + 1
            text.insert(str(float(i+1)),
f"Kurang({check})={Puz.Kurang(check)}  Kurang({check+1})={Puz.Kurang(check+
1)}  Kurang({check+2})={Puz.Kurang(check+2)}\n")
        text.insert("6.0", f"Nilai dari sum_{1}^{16} kurang(i)+X adalah=
{Puz.KurangTotal()}\n")
        if (Puz.Cek()):
            solution, iterationnumber = SolvePuzzle(Puz, iterationnumber)
            tStop = perf_counter()
```

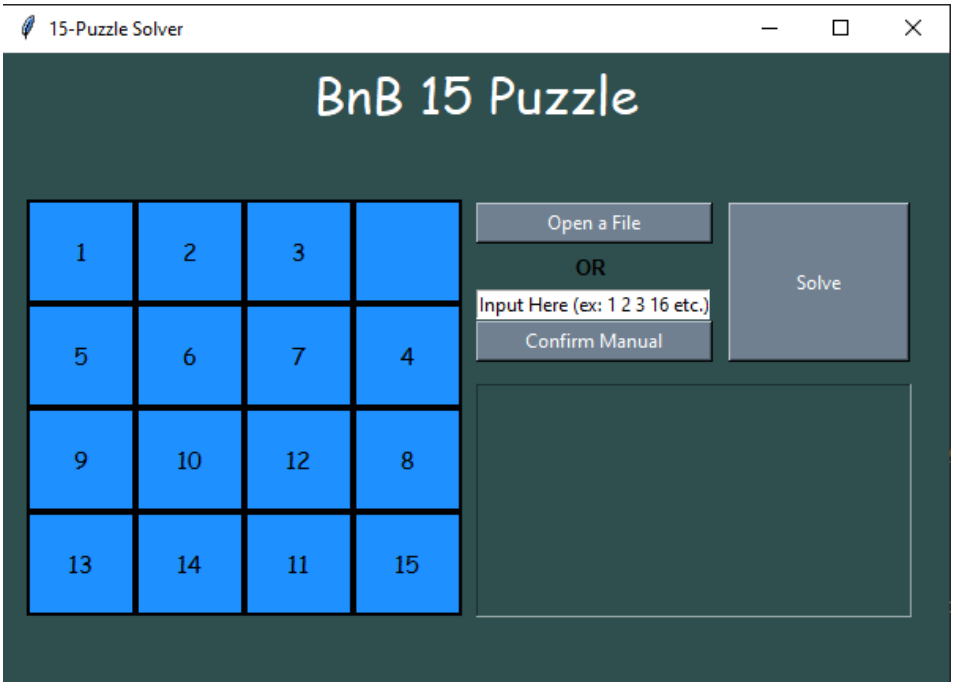
```
            text.insert("7.0", "Jumlah simpul yang dibangkitkan = " +
str(iterationnumber) + "\nWaktu Eksekusi = " + str("{:.6f}".format(tStop-
tStart)) + " sekon")
            text.tag_add("tag_name", "1.0", "end")
            grid.path(solution, window)

#function to open browser file
def select_file():
    filetypes = (
        ('text files', '*.txt'),
        ('All files', '*.*')
    )

    filename = fd.askopenfilename(
        title='Open a file',
        initialdir='/',
        filetypes=filetypes)
    Puz.readFile(filename)
    grid.change_button()

#function to read input from entry to matrix
def inputManual():
    Puz.inputCommand(entryofstate.get())
    grid.change_button()

#function to delete temporary text in entry
def temp_text(e):
   entryofstate.delete(0,"end")

def input_failed():
    text.delete("1.0","end")
    text.insert("1.0", "Gagal Membaca Input")
    text.tag_add("tag_name", "1.0", "end")

#create window
window = Tk()
Puz = Puzzle()
window.title("15-Puzzle Solver")
window.geometry("600x400")
window.configure(background='dark slate gray')

#create label for title
Label(text="BnB 15 Puzzle",bg = 'dark slate gray', fg = 'white', font =
("Comic Sans MS", "24")).pack()

#initialization
grid = gridFrame(window, Puz)
iterationnumber = 0
```

```
#create button for open file
open_button = Button(text='Open a File',fg = 'white', bg = 'slate gray',
command=select_file)
open_button.place(x=300, y = 95)
open_button.config(width = 20)

#create text
label2 = Label(text='OR', bg = 'dark slate gray')
label2.config(font='helvetica 10 bold')
label2.place(x = 360, y= 125)

#create entry for manual input
entryofstate = Entry()
entryofstate.config(width = 24)
entryofstate.insert(0, "Input Here (ex: 1 2 3 16 etc.)")
entryofstate.bind("<FocusIn>", temp_text)
entryofstate.place(x = 300, y = 150)

#create confirm button for manual input
confirmbutton = Button(text = 'Confirm Manual', fg = 'white', bg = 'slate
gray', command = inputManual)
confirmbutton.config(width = 20)
confirmbutton.place(x=300, y = 170)

#create solve button to solve 15 puzzle
solvebutton = Button(text = 'Solve', fg = 'white', bg = 'slate gray',
command = solve)
solvebutton.config(width = 15, height = 6)
solvebutton.place(x=460, y = 95)

#create text for constraint etc.
text=Text(bg = 'dark slate gray', font = ("Comic Sans MS", "10"))
text.tag_configure("tag_name", justify='left')
text.config(width = 34, height = 8)
text.place(x=300, y = 210)

window.mainloop()
```

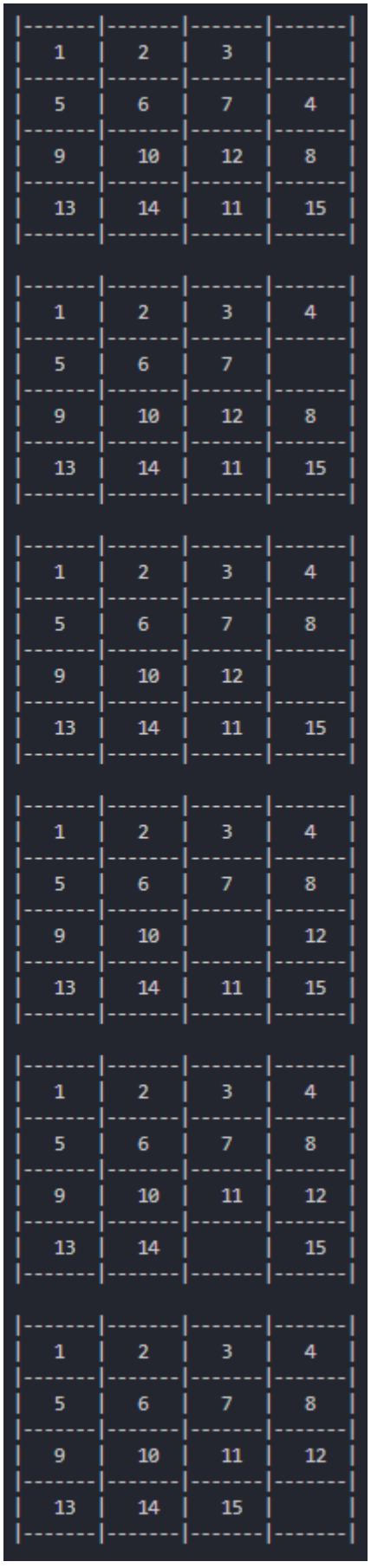
BAB III
PENGUJIAN PROGRAM

A. *Test Case Solveable* 1

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



Keluaran program :


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

B. *Test Case Solveable* 2

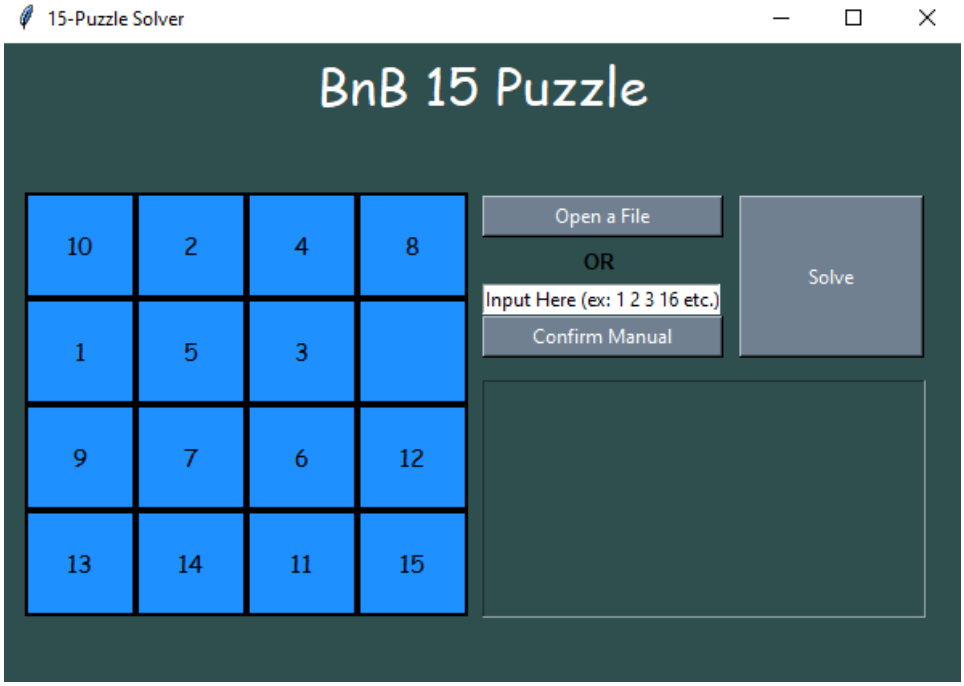
Berikut adalah contoh susunan puzzle yang dapat diselesaikan :

10 2 4 8

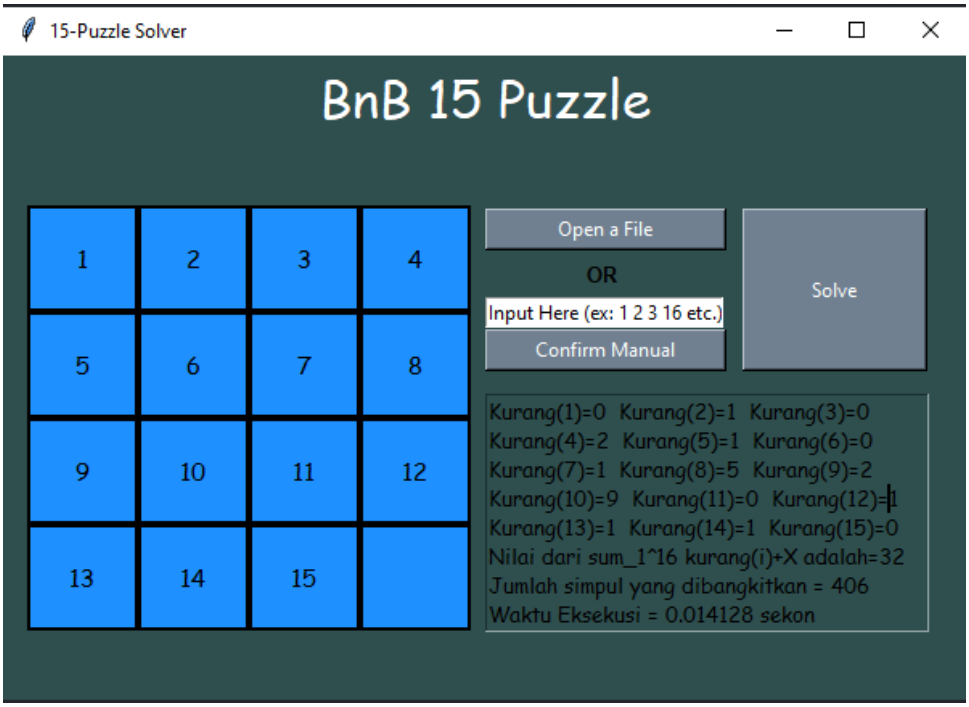
1 5 3 16

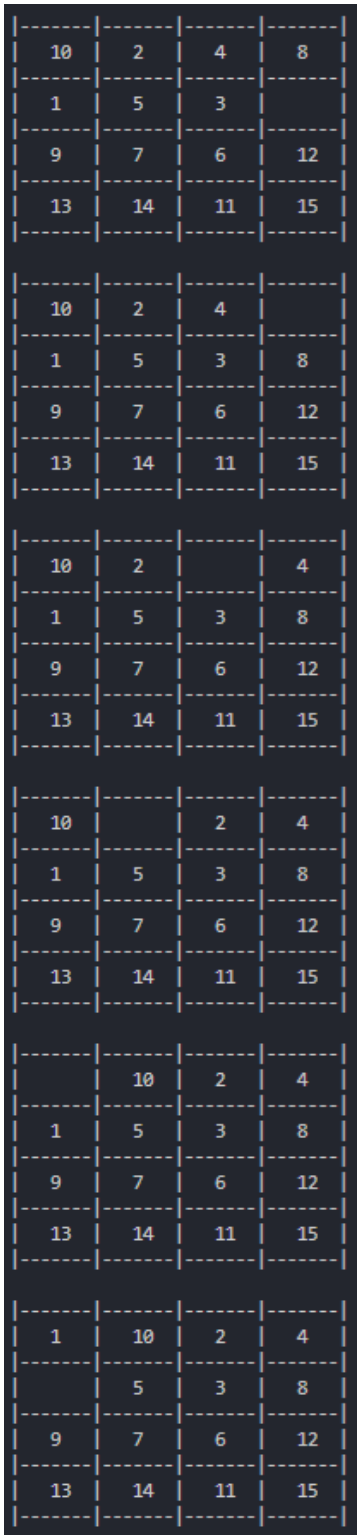
9 7 6 12

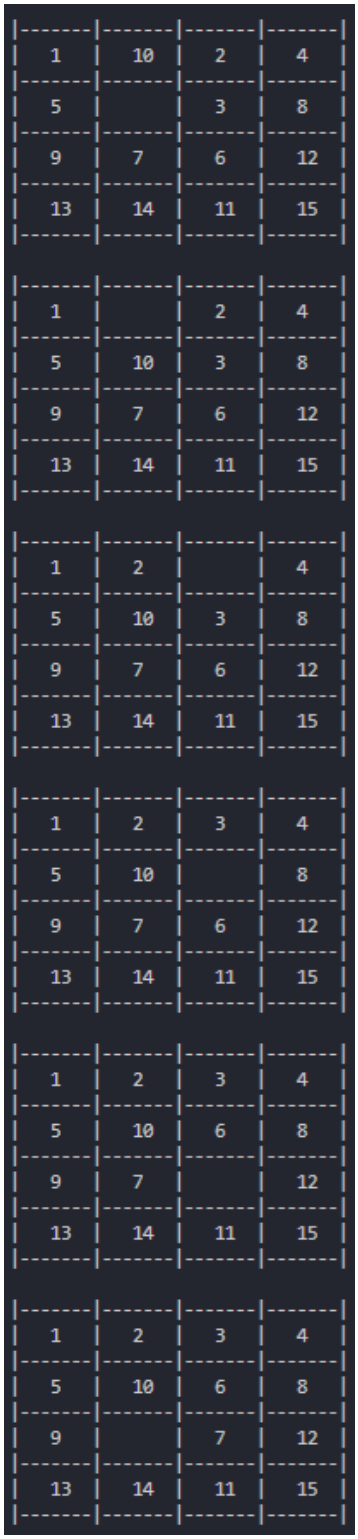
13 14 11 15

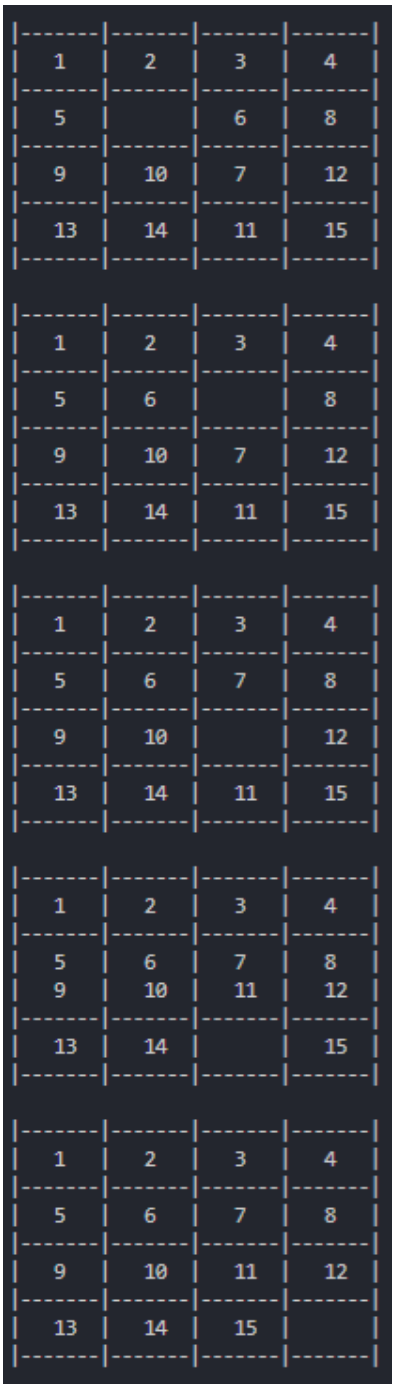


Keluaran program :




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


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


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

C. *Test Case Solveable* 3

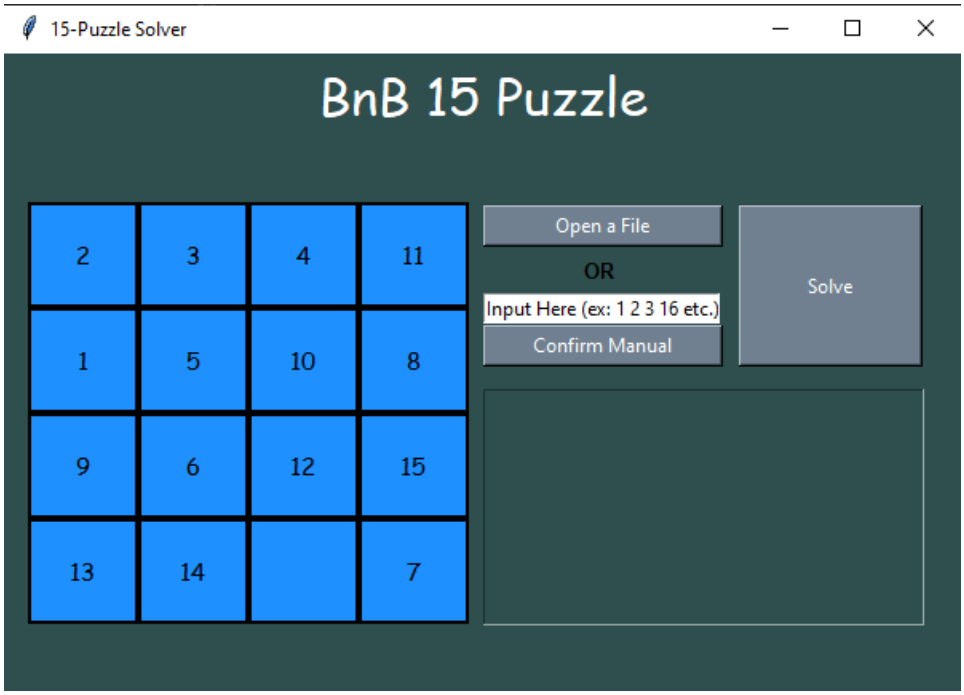
Berikut adalah contoh susunan puzzle yang dapat diselesaikan :

2 3 4 11

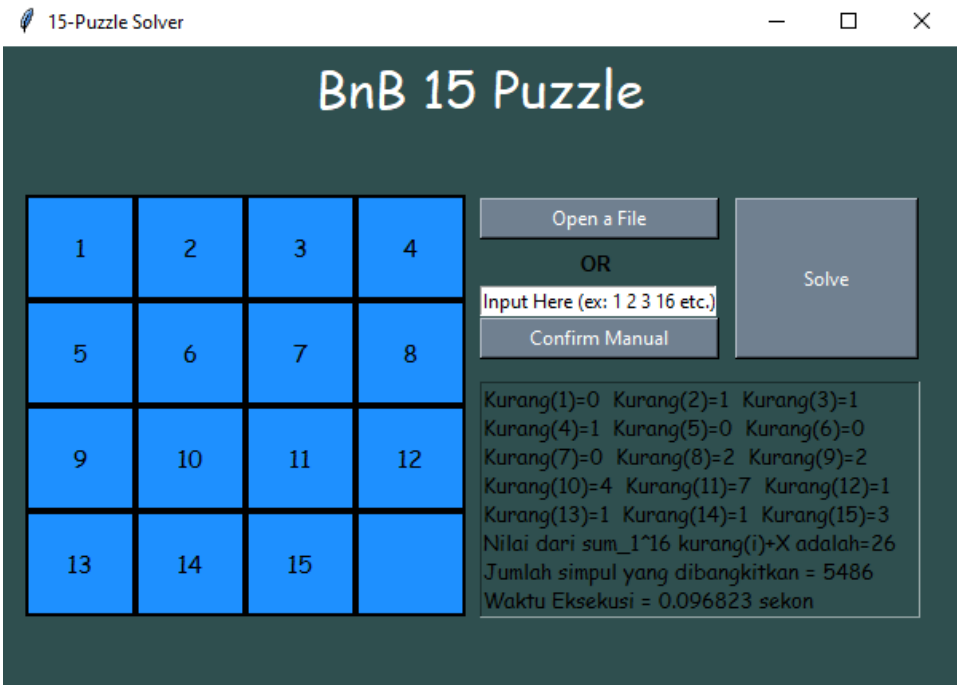
1 5 10 8

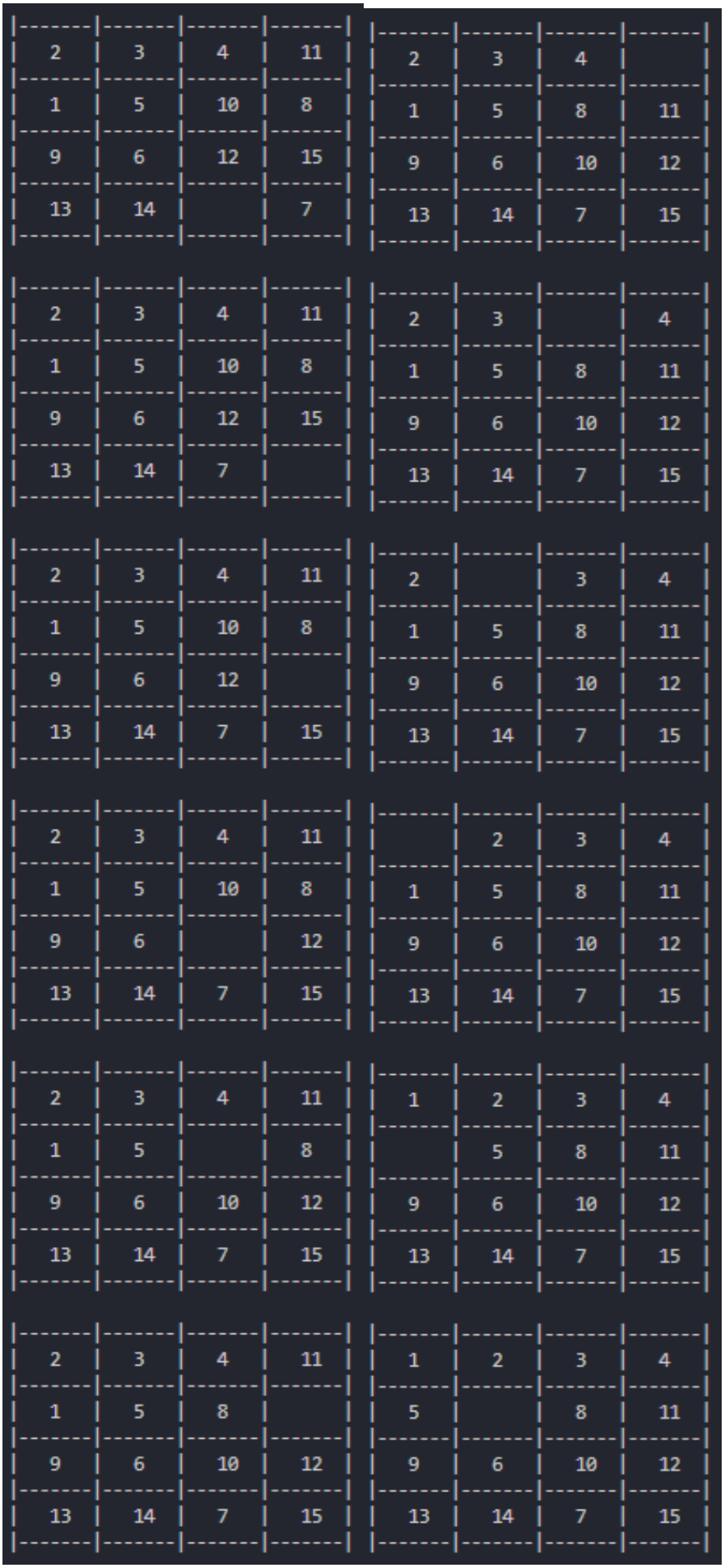
9 6 12 15

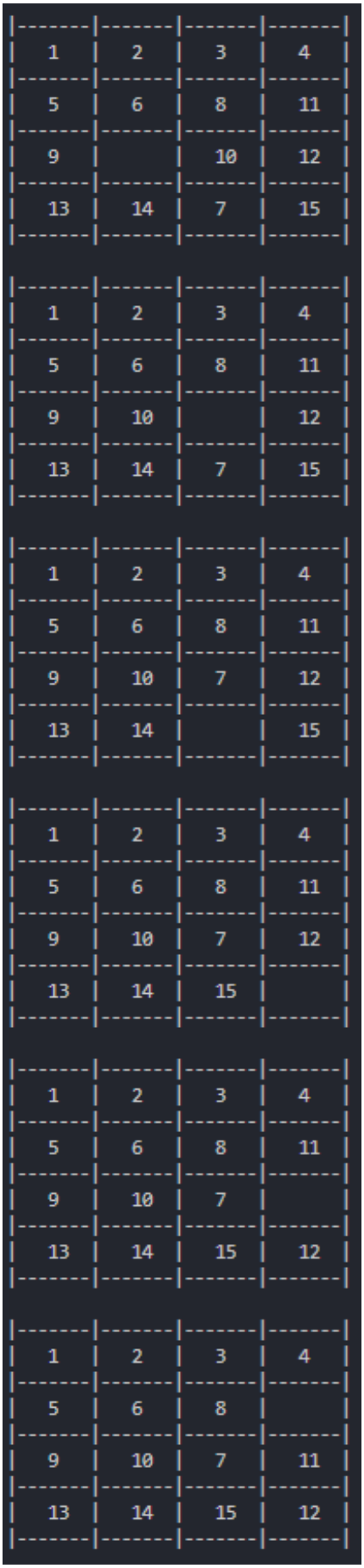
13 14 16 7

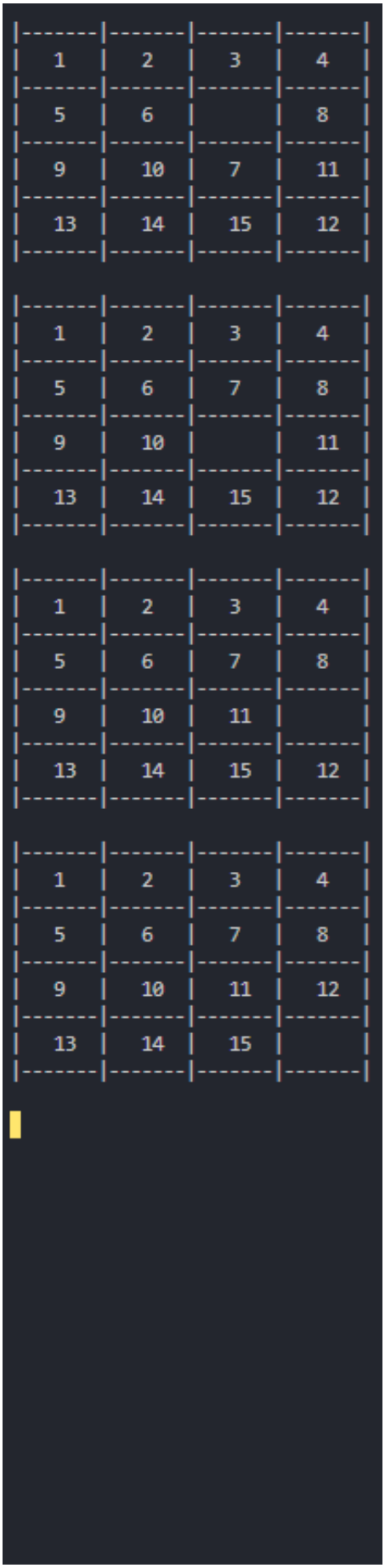


Keluaran program :




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


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


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

D. *Test Case Not Solveable 1*

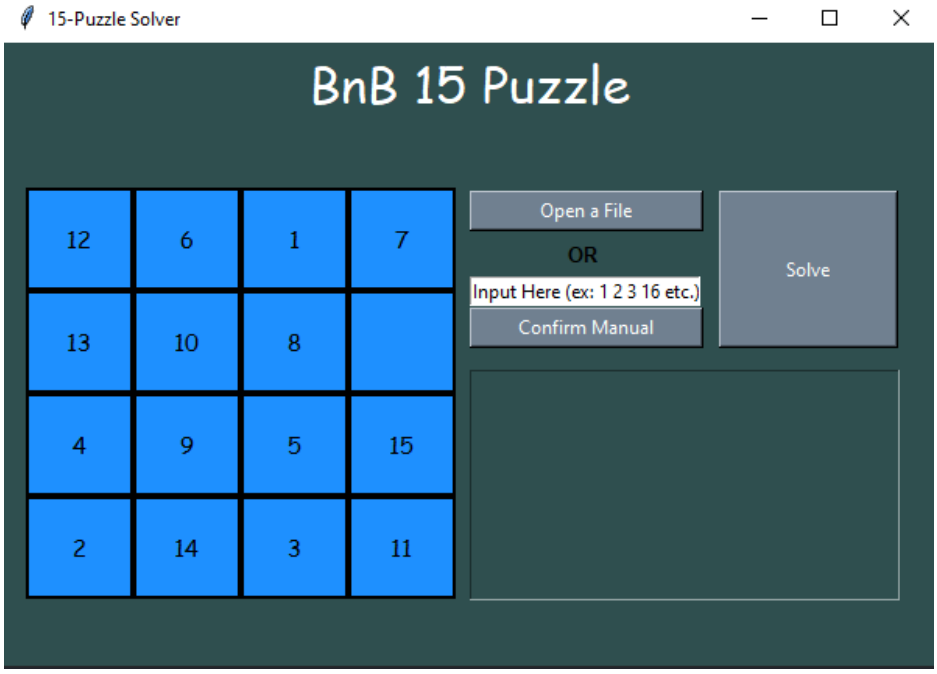
Berikut adalah contoh susunan puzzle yang dapat diselesaikan :

12 6 1 7

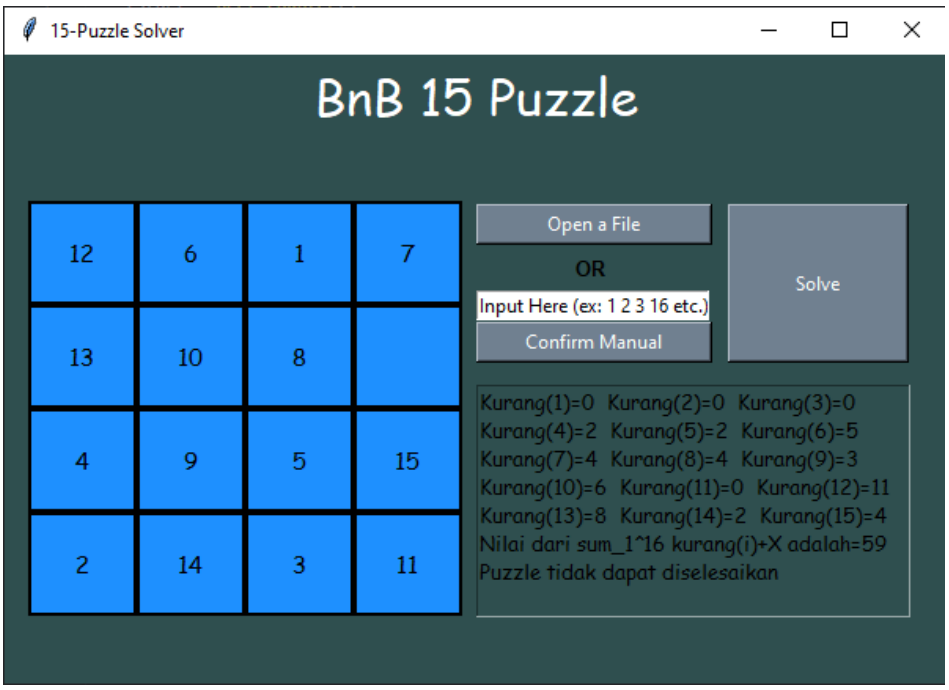
13 10 8 16

4 9 5 13

2 14 3 11



Keluaran Program:



E. *Test Case Not Solveable 2*

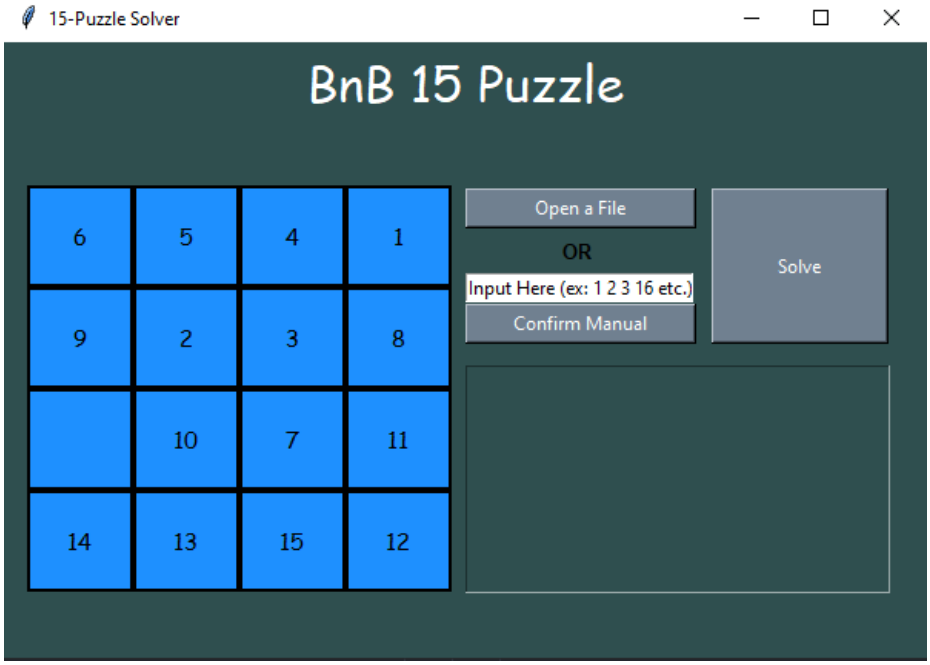
Berikut adalah contoh susunan puzzle yang dapat diselesaikan :

6 5 4 1

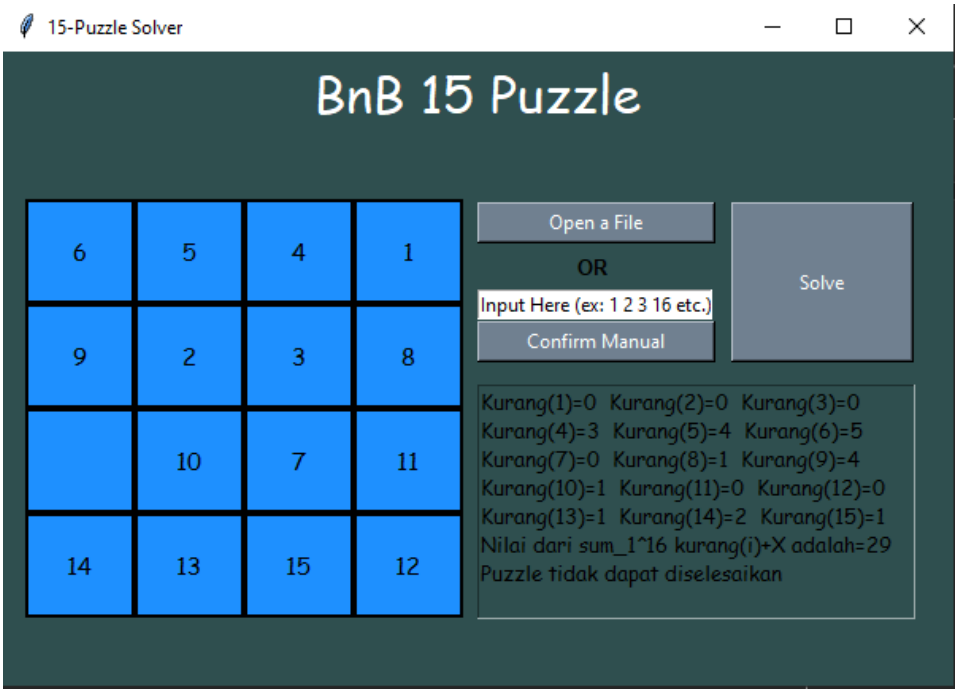
9 2 3 8

16 10 7 11

14 13 15 12



Keluaran Program:



LAMPIRAN

REPOSITORY GITHUB :

airathalca/TUCIL3-IF2211-15Puzzle (github.com)

CHECKLIST :

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