# Laporan Tugas Kecil 3

# IF2211 Strategi Algoritma

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


**Oleh :**

**Lyora Felicya**

**13520073**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2022**

# Daftar Isi

# BAB 1 – Teori Dasar

## 1.1 Algoritma *Branch and Bound*

Algoritma *Branch and Bound* merupakan salah satu metode yang digunakan untuk menyelesaikan persoalan optimasi, dengan meminimalkan atau memaksimalkan suatu fungsi objektif, yang tidak melanggar constraints persoalan. Branch and Bound dapat dikatakan sebagai gabungan dari metode pencarian Breadth First Search (BFS) dengan least cost search. Algoritna Branch and Bound juga menerapkan pemangkasan pada jalur yang dianggap tidak lagi mengarah pada solusi.

## 1.2 Permainan 15-Puzzle

15-Puzzle merupakan sebuah permainan puzzle berukuran 4x4 dengan 15 kotaknya berisi ubin bernomor 1-15, serta terdapat satu buah kotak kosong untuk melakukan pergeseran. Ubin dapat digerakkan ke atas, bawah, kiri, dan kanan. Tujuan akhir pada permainan ini adalah untuk mengurutkan ubin dari nomor 1 hingga 15 secara berurutan dan menyisakan ubin kosong di kotak terakhir.

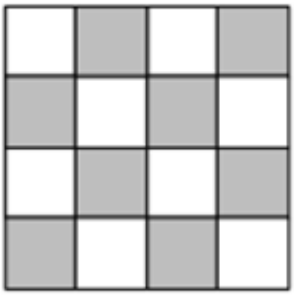
Permasalahan ini bisa diselesaikan dengan algoritma branch and bound dengan membangkitkan simpul – simpul menuju solusi dan memasukkannya ke dalam sebuah priority queue dengan cost setiap simpul sebagai urutan prioritasnya. Cost yang paling minimum merupakan prioritas paling tinggi.

## 1.3 Implementasi Algoritma Branch and Bound dalam Penyelesaian 15-Puzzle

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

Untuk menyelsaikan permainan 15-Puzzle yang memiliki susunan acak, terlebih dahulu perlu dilakukan pengecekan apaah susunan awal dapat mencapai susunan *goal* seperti pada gambar diatas. Pengecekan dilakukan dengan menggunakan teorema bahwa status tujuan hanya dapat dicapai dari status awal jika nilai dari $\sum_{i=1}^{16} KURANG(i) + X$ adalah genap. Kurang(i) adalah banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i). POSISI(i) adalah posisi ubin bernomor i pada susunan yang diperiksa. sedangkan X adalah letak ubin kosong pada

susunan awal. X bernilai 1 saat berada pada ubin dengan koordinat i+j adalah ganjil dan bernilai 0 pada ubin dengan koordinatn i+j adalah genap, ilustrasinya dapat dilihat pada gambar dibawah ini.



Jika teorema di atas terpenuhi, maka susunan ubin tersebut solveable, begitu juga sebaliknya. Apabila susunan ubin solveable, maka akan dilakukan proses pencarian simpul tujuan dari simpul utama.

**Secara garis besar, langkah-langkap pencarian solusi 15-Puzzle dengan menggunakan algoritma Branch and Bound adalah sebagai berikut :**

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (goal node), maka solusi telah ditemukan. Jika hanya satu solusi yang diinginkan, maka stop.
2. Jika Q kosong, Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul i yang mempunyai nilai ‘cost’ ĉ(i) paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara sembarang.
4. Jika simpul i adalah simpul solusi, berarti solusi sudah ditemukan. Jika satu solusi yang diinginkan, maka stop. Pada persoalan optimasi dengan pendekatan least cost search, periksa cost semua simpul hidup. Jika cost nya lebih besar dari cost simpul solusi, maka matikan simpul tersebut.
5. Jika simpul i bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika i tidak mempunyai anak, kembali ke langkah 2.
6. Untuk setiap anak j dari simpul i, hitung ĉ(j), dan masukkan semua anak-anak tersebut ke dalam Q.
7. Kembali ke langkah 2.

*Least cost search* digunakan untuk menentukan simpul anak mana yang akan ditelusuri selanjutnya. Tiap pembangkitan simpul anak, akan dihitung nilai *cost*nya dan dimasukkan ke

dalam antrian pemrosesan simpul hidup. *Queue* yang digunakan adalah *Priority Queue* dengan prioritas utama adalah simpul anak dengan nilai *cost* terkecil. Dengan demikian simpul anak yang dipilih yang memiliki *cost* terkecil dari simpul anak yang lain pada antrian pemrosesan. *Cost* yang digunakan adalah sebagai berikut :

c(i) = f(i) + g(i)

c(i) = *cost* untuk simpul ke i

f(i) = *cost* untuk mencapai simpul ke i dari simpul akar (*depth*)

g(i) = *cost* untuk mencapai simpul tujuan dari simpul ke i. Dalam kasus ini, taksiran cost yang digunakan adalah jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir (goal state).

Program yang dibuat terdiri atas 2 file, yaitu puzzleSolver.py dan main.py. Pada file **puzzleSolver.py**, terdapat kelas PrioQueue untuk menyimpan simpul-simpul yang telah dibangkitkan, serta terdapat kelas Node untuk menyimpan informasi dari sertiap simpul, yaitu cost, depth, move, dan parent node nya. File ini juga berisi fungsi-fungsi untuk melakukan perhitungan dan pendukung lainnya. File **main.py** berisi program utama yang akan dijalankan. Terdapat 2 cara untuk melakukan input persoalan, yaitu dengan file teks atau puzzle dibangkitkan secara random oleh program. Angka 0 digunakan sebagai penanda ubin kosong.

**Secara umum, alur kerja program adalah sebagai berikut :**

1. Program menentukan apakah posisi awal puzzle dapat diselesaikan, dengan mengimplementasikan fungsi Kurang(i) yang telah dijelaskan sebelumnya.
2. Jika tidak bisa, maka akan ditampilkan pesan puzzle tidak dapat diselesaikan.
3. Jika bisa, maka program akan melakukan pencarian solusi dengan membangkitkan simpul-simpul serta menghitung nilai bound dari simpul tersebut. Nilai bound tiap simpul adalah penjumlahan cost yang diperlukan untuk sampai suatu simpul x dari akar, dengan taksiran cost simpul x untuk sampai ke goal.
4. Setelah berhasil didapatkan solusi, program akan menampilkan matriks-matriks yang merupakan path aksi yang dilakukan dari posisi awal ke susunan akhir.
5. Pada akhir pencarian, akan ditampilkan juga waktu eksekusi program dan jumlah simpul yang dibangkitkan.

# BAB 2 – Kode Program

## puzzleSolver.py

```
# 15 Puzzle Solver dengan Algoritma Branch and Bound
# Lyora Felicya / 13520073

import time
import random
import numpy as np

goal = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0]).reshape(4,4)
direction = ("Up", "Right", "Down", "Left")

class PrioQueue:
    def __init__(self):
        self.queue = []

    def __str__(self):
        return '\n'.join([str(i) for i in self.queue])

    def enqueue(self, data):
        self.queue.append(data)

    def dequeue(self):
        index = 0
        for i in range(len(self.queue)):
            if(self.queue[i][0] < self.queue[index][0]):
                index = i
        item = self.queue[index]
        del self.queue[index]
        return item

class Node:
    def __init__(self, data=None):
        self.matrix = data
        self.parent = None
        self.move = ""
        self.depth = 0

# find position
def findPosition(x, puzzle):
    matrix = [[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]]
    for i in range(4):
        for j in range(4):
            if puzzle[i][j] == x:
                position = matrix[i][j]
    return position

# Kurang(i) function
def less(i, puzzle):
    # Number of tiles where j < i and position(j) > position(i)
    count = 0
    if (i != 16):
```

```
        for row in range(4):
            for col in range(4):
                j = puzzle[row][col]
                j_pos = findPosition(j,puzzle)
                i_pos = findPosition(i,puzzle)
                if (j < i) and (j_pos > i_pos) and (j != 0):
                    count += 1
    else:
        blankSpacePos = findPosition(0,puzzle)
        count = 16 - blankSpacePos
    return count

# get row and col of blank space
def findBlankSpace(puzzle):
    for i in range(4):
        for j in range(4):
            if(puzzle[i][j] == 0):
                row = i
                column = j
    return row,column

# to determine the value of X
def findX(puzzle):
    i, j = findBlankSpace(puzzle)
    if((i+j) % 2 == 1):
        X = 1
    else:
        X = 0
    return X

# print Kurang(i) for each tile
def printLess(puzzle) :
    sum = 0
    X = findX(puzzle)
    print("Fungsi Kurang(i)")
    print("================================================")
    for i in range(1,17) :
        print("Kurang"+"("+str(i)+")" + " : " + str(less(i, puzzle)))
        sum += less(i, puzzle)
    print()
    print("Total : " + str(sum))
    print("X : " + str(X))
    if((X+sum) % 2 == 0) :
        print("sigma KURANG(i) + X = " + str(X+sum) + " (even)")
    else:
        print("sigma KURANG(i) + X = " + str(X+sum) + " (odd)")

# to determine if the puzzle is solveable
def isSolveable(puzzle):
    sum = 0
    X =  findX(puzzle)
    for i in range(1,17):
        sum = sum + less(i, puzzle)
```

```
    sum += X
    if(sum % 2 == 0):
        return True
    else:
        return False

# calculate cost of a node
def cost(depth,matrix,goal):
    misplaced = 0
    for i in range(4):
        for j in range(4):
            # count number of misplaced tiles
            if((matrix[i][j] != goal[i][j]) and matrix[i][j] != 0):
                misplaced += 1
    cost = misplaced + depth
    return cost

# Functions to Move Puzzle Tiles
def swap(matrix,row,column):
    x,y = findBlankSpace(matrix)
    matrix[x][y] = matrix[row][column]
    matrix[row][column] = 0
    return matrix

def move(matrix,direction):
    matrix_temp = matrix.copy()
    x,y = findBlankSpace(matrix_temp)
    if(direction == "Left"):
        if(y != 0):
            y -= 1
    elif(direction == "Right"):
        if(y != 3):
            y += 1
    elif(direction == "Up"):
        if(x != 0):
            x -= 1
    elif(direction == "Down"):
        if(x != 3):
            x += 1
    matrix_temp = swap(matrix_temp,x,y)
    return matrix_temp

def oppositeDirection(direction):
    if(direction == "Left"):
        return "Right"
    elif(direction == "Right"):
        return "Left"
    elif(direction == "Up"):
        return "Down"
    elif(direction == "Down"):
        return "Up"

# to check if the goal state is reached
```

```
def equal(matrix,goal):
    for i in range(4):
        for j in range(4):
            if(matrix[i][j] != goal[i][j]):
                return False
    return True

# print matrix without brackets
def printMatrix(matrix) :
    for i in range(4) :
        for j in range(4) :
            if (matrix[i][j] < 10) :
                print(str(matrix[i][j]) + " ", end=" ")
            else :
                if (matrix[i][j] == 16) :
                    print("0 ", end=" ")
                else :
                    print(matrix[i][j], end=" ")
        print()

# print nodes from initial state to goal
def printSolution(node):
    if(node.parent == None):
        return
    printSolution(node.parent)
    print()
    print("---------------")
    print("Step "+ str(node.depth) + " : " + str(node.move))
    print("---------------")
    printMatrix(node.matrix)
    print("---------------")

# find solution using Branch and Bound
def solve(puzzle):

    # set initial puzzle state as root node
    node = Node(puzzle)
    Queue = PrioQueue()
    nodeExpanded = 0

    # add root node to queue and calculate root cost
    Queue.enqueue((cost(0,node.matrix,goal),node,"",0))

    # remove root node from queue of livenodes
    liveNodes = Queue.dequeue()

    node = liveNodes[1]
    curNode = node.matrix
    prev = ""
    next_step = liveNodes[3] + 1
    nodeExpanded += 1

    # check if the goal is reached
```

```
    while(not equal(curNode, goal)):
        for dir in direction:
            if(dir != prev):
                expandedNode = move(curNode,dir)
                if(not equal(expandedNode, curNode)):
                    # Expand new node
                    newNode = Node(expandedNode)
                    newNode.parent = node
                    newNode.depth = node.depth + 1
                    nodeExpanded += 1
                    newNode.move = dir

                    # Add new expanded node to Queue
                    Queue.enqueue((cost(next_step,newNode.matrix,goal),newNo
de,dir,next_step))

        liveNodes = Queue.dequeue()
        node = liveNodes[1]
        curNode = node.matrix
        Move = liveNodes[2]
        prev = oppositeDirection(Move)
        next_step = liveNodes[3] + 1

    printSolution(node)
    return nodeExpanded
```

## main.py

```
from puzzleSolver import *

# Main Program
def Main():
    cont = "Y"
    while(cont == "Y" or cont == "y"):
        print("================================================")
        print("                15 PUZZLE SOLVER                ")
        print("================================================")
        print(" Welcome to 15-Puzzle Solver!")
        print(" Which input method would you like to use?")
        print(" 1. Text File")
        print(" 2. Random by program")
        choice = int(input(" >> "))

        # assume choice input and filename is always valid
        if(choice == 1):
            filename = input(" Enter name of the file (with .txt) : ")
            puzzle = np.loadtxt("../test/"+ filename, dtype=int)
        else:
            default = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,0]
            random.shuffle(default)
            puzzle = np.array(default).reshape(4,4)
```

```
        print("==============  INITIAL STATE   =================")
        printMatrix(puzzle)
        print("================================================")
        printLess(puzzle)
        print("================================================")
        if(isSolveable(puzzle)):
            print("This puzzle is solveable. Solving puzzle...")

            print()
            print("---------------")
            print(" Your Puzzle ")
            print("---------------")
            printMatrix(puzzle)
            print("---------------")

            start = time.time()
            node = solve(puzzle)
            end = time.time()
            totalTime = end - start

            print()
            print("Puzzle solved successfully!")
            print("Time taken               : " + str(totalTime) + "
seconds")
            print("Number of nodes expanded : " + str(node))
            print()
        else:
            print("This puzzle is not solveable!")
            print()
        cont = input("Do you want to try another puzzle? (Y/N) : ")
    print("Okay bye~ Thankyou!\n")

if __name__ == "__main__":
    Main()
```

# Bab 3 – Screenshot Hasil Pengujian

## 3.1 Pengujian puzzle pada file solveable-1.txt

### 3.1.1 File Input

solveable-1.txt

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

### 3.1.2 Output program

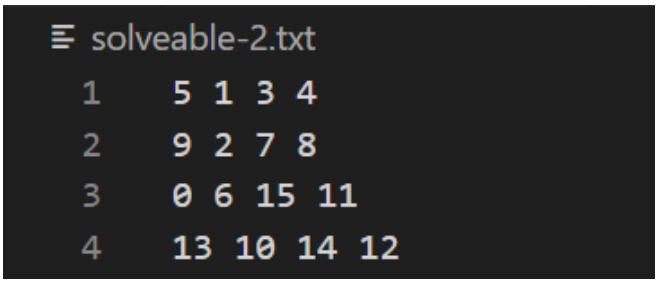
```
==================================================
                15 PUZZLE SOLVER
==================================================
 Welcome to 15-Puzzle Solver!
 Which input method would you like to use?
 1. Text File
 2. Random by program
 >> 1
 Enter name of the file (with .txt) : solveable-1.txt
==============  INITIAL STATE   =================
1  2  3  4
5  6  0  8
9  10 7  11
13 14 15 12
==================================================
Fungsi Kurang(i)
==================================================
Kurang(1) : 0
Kurang(2) : 0
Kurang(3) : 0
Kurang(4) : 0
Kurang(5) : 0
Kurang(6) : 0
Kurang(7) : 0
Kurang(8) : 1
Kurang(9) : 1
Kurang(10) : 1
Kurang(11) : 0
Kurang(12) : 0
Kurang(13) : 1
Kurang(14) : 1
Kurang(15) : 1
Kurang(16) : 9
```

```
Total : 15
X : 1
sigma KURANG(i) + X = 16 (even)
================================================
This puzzle is solveable. Solving puzzle...

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

--------------
Step 1 : Down
--------------
1  2  3  4
5  6  7  8
9  10 0  11
13 14 15 12
--------------

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

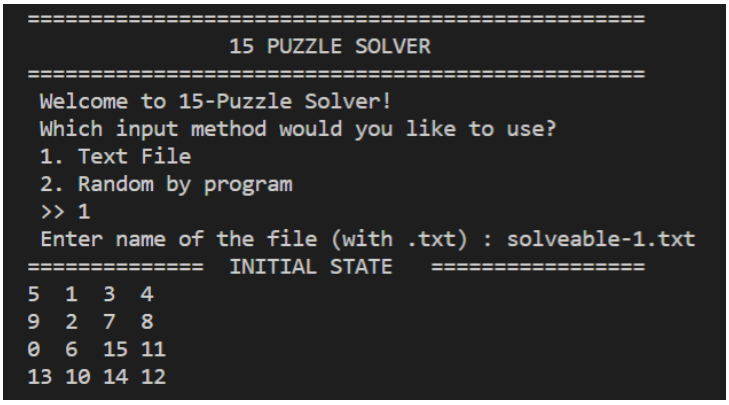
```
--------------
Step 3 : Down
--------------
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 0
--------------

Puzzle solved successfully!
Time taken               : 0.011002540588378906 seconds
Number of nodes expanded : 10

Do you want to try another puzzle? (Y/N) : N
Okay bye~ Thankyou!
```

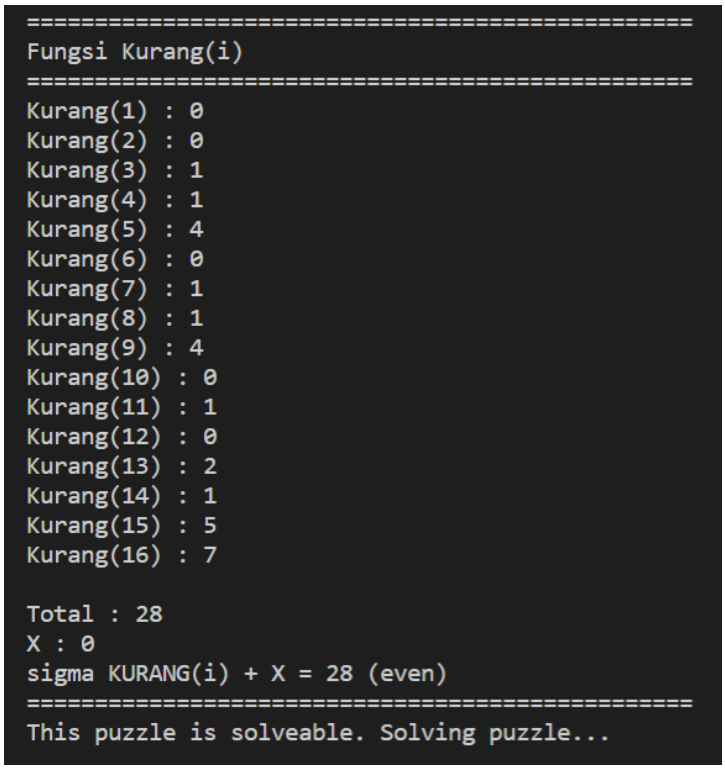
## 3.2 Pengujian puzzle pada file solveable-2.txt

### 3.2.1 File Input




### 3.2.2 Output Program



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

---------------
Step 1 : Up
---------------
5  1  3  4
0  2  7  8
9  6  15 11
13 10 14 12
---------------

---------------
Step 2 : Up
---------------
0  1  3  4
5  2  7  8
9  6  15 11
13 10 14 12
---------------

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

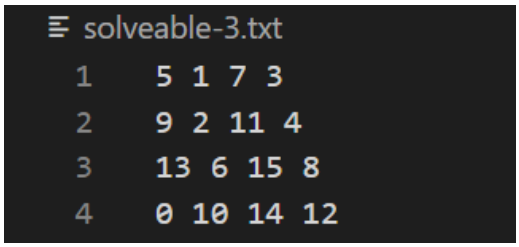
```
---------------
Step 4 : Down
---------------
1  2  3  4
5  0  7  8
9  6  15 11
13 10 14 12
---------------

---------------
Step 5 : Down
---------------
1  2  3  4
5  6  7  8
9  0  15 11
13 10 14 12
---------------

---------------
Step 6 : Down
---------------
1  2  3  4
5  6  7  8
9  10 15 11
13 0  14 12
---------------

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

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

---------------
Step 9 : Right
---------------
1  2  3  4
5  6  7  8
9  10 11 0
13 14 15 12
---------------

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

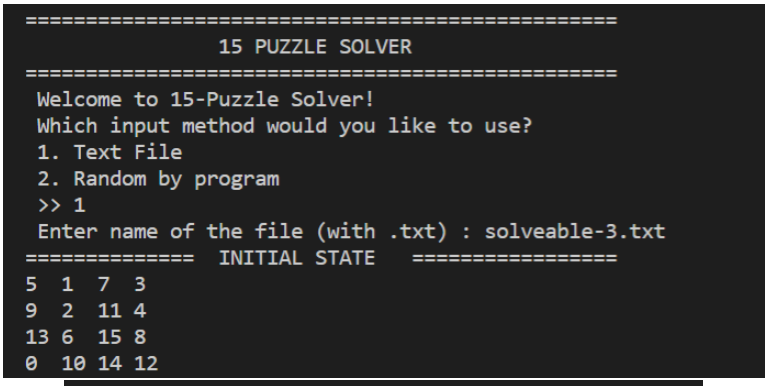
```
Puzzle solved successfully!
Time taken               : 0.0319972038269043 seconds
Number of nodes expanded : 24

Do you want to try another puzzle? (Y/N) : N
Okay bye~ Thankyou!
```

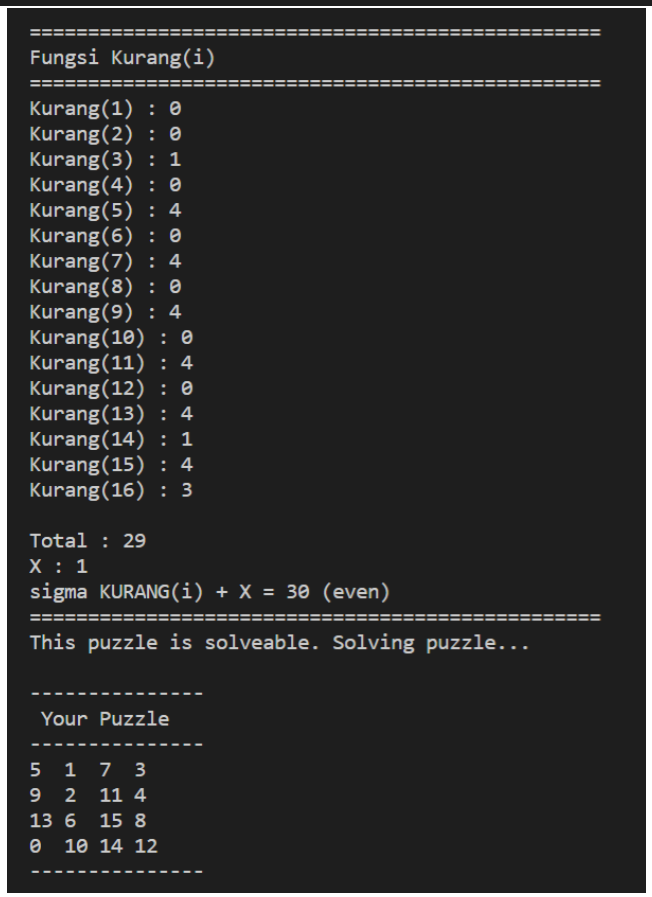
## 3.3 Pengujian puzzle pada file solveable-3.txt

### 3.3.1 File Input




### 3.3.2 Output program



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

--------------
Step 2 : Up
--------------
5  1  7  3
0  2  11 4
9  6  15 8
13 10 14 12
--------------

--------------
Step 3 : Up
--------------
0  1  7  3
5  2  11 4
9  6  15 8
13 10 14 12
--------------

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

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

--------------
Step 6 : Down
--------------
1  2  7  3
5  6  11 4
9  0  15 8
13 10 14 12
--------------

--------------
Step 7 : Down
--------------
1  2  7  3
5  6  11 4
9  10 15 8
13 0  14 12
--------------

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

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

--------------
Step 10 : Up
--------------
1  2  7  3
5  6  0  4
9  10 11 8
13 14 15 12
--------------

--------------
Step 11 : Up
--------------
1  2  0  3
5  6  7  4
9  10 11 8
13 14 15 12
--------------

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

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

---------------
Step 14 : Down
---------------
1  2  3  4
5  6  7  8
9  10 11 0
13 14 15 12
---------------

---------------
Step 15 : Down
---------------
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 0
---------------

Puzzle solved successfully!
Time taken               : 0.04600024223327637 seconds
Number of nodes expanded : 33

Do you want to try another puzzle? (Y/N) : N
Okay bye~ Thankyou!
```

## 3.4 Pengujian puzzle pada file unsolveable-1.txt

### 3.4.1 File input

unsolveable-1.txt

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

### 3.4.2 Output program

```
================================================
               15 PUZZLE SOLVER
================================================
 Welcome to 15-Puzzle Solver!
 Which input method would you like to use?
 1. Text File
 2. Random by program
 >> 1
 Enter name of the file (with .txt) : unsolveable-1.txt
=============  INITIAL STATE   =================
1  2  3  4
5  6  7  8
9  10 0  12
13 14 15 11
```

```
 ================================================
 Fungsi Kurang(i)
 ================================================
 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) : 1
 Kurang(13) : 1
 Kurang(14) : 1
 Kurang(15) : 1
 Kurang(16) : 5

 Total : 9
 X : 0
 sigma KURANG(i) + X = 9 (odd)
 ================================================
 This puzzle is not solveable!

 Do you want try another puzzle? (Y/N) : N
 Okay bye~ Thankyou!
```

## 3.5 Pengujian puzzle pada file unsolveable-2.txt

### 3.5.1 File input

unsolveable-2.txt

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

### 3.5.2 Output program

```
================================================
               15 PUZZLE SOLVER
================================================
 Welcome to 15-Puzzle Solver!
 Which input method would you like to use?
 1. Text File
 2. Random by program
 >> 1
 Enter name of the file (with .txt) : unsolveable-2.txt
==============  INITIAL STATE   =================
5  1  7  3
9  2  11 4
13 0  15 8
6  10 14 12
```

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

 Total : 34
 X : 1
 sigma KURANG(i) + X = 35 (odd)
 ================================================
 This puzzle is not solveable!

 Do you want to try another puzzle? (Y/N) : N
 Okay bye~ Thankyou!
```

# Bab 4 – Lampiran

## 4.1 Berkas Data Uji

### 4.1.1 solveable-1.txt

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

### 4.1.2 solveable-2.txt

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

### 4.1.3 solevable-3.txt

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

### 4.1.4 unsolveable-1.txt

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

### 4.1.5 unsolveable-2.txt

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

## 4.2 Alamat Drive Program

https://github.com/lyorafelicya/Tucil3_13520073

## 4.3 Tabel *Checklist*

| Poin | Ya | Tidak |
|---|---|---|
| 1. Program berhasil dikompilasi | √ | |
| 2. Program berhasil running | √ | |
| 3. Program dapat menerima input dan menuliskan output. | √ | |
| 4. Luaran sudaah benar untuk semua data uji | √ | |
| 5. Bonus dibuat | | √ |

**Referensi**

http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf