TUGAS KECIL 3
IF2211 STRATEGI ALGORITMA

15 PUZZLE GAME

Disusun oleh:

Jones Napoleon
13518086

Mata Kuliah Strategi Algoritma
Semester 2 Tahun Ajaran 2019/2020

Prodi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung 2020

DAFTAR ISI

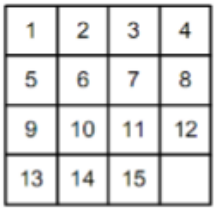
DAFTAR ISI

BAB I

TEORI DASAR 15PUZZLE

1.1. *OVERVIEW*

15Puzzle merupakan permainan klasik dengan ruang sebanyak 16 yang melibatkan angka 1 sampai 15 secara acak dan satu ruang kosong untuk melakukan perpindahan ruang. Pada pemrograman 15Puzzle kali ini, digunakan representasi *array* untuk menyimpan semua angka yang menempati ruang tersebut.



Gambar 1. Susunan akhir 15 Puzzle

Secara dasar, program menerima *input* berupa file yang berisi 16 buah *integer* dalam susunan 4 baris dengan tiap baris memiliki 4 kolom. Program selanjutnya akan menghitung fungsi sigma hitung nya dan nilai X nya – suatu nilai yang mengindikasikan bisa tidaknya puzzle tersebut diselesaikan. Jika puzzle bisa diselesaikan, program akan memulai mengganti ruang puzzle dengan sebelahnya dengan metode **BranchAndBound** yang mana **bound-**nya dihitung dengan nilai cost pada suatu state puzzle.

1.2. STRUKTUR DATA

Suatu state puzzle dibuat dengan model berupa ***array*** yang dimulai dari indeks 0 sampai 16 dan berisi elemen dari 0 sampai 16 secara acak. Untuk setiap *node* yang dibangkitkan, *node* tersebut bersama dengan *array* (*sequence*), *cost*, *previousAction*, *path*, dan *walkThrough* akan dimasukkan ke dalam suatu ***priority_queue*** dengan *cost* sebagai penanda prioritas.

BAB II

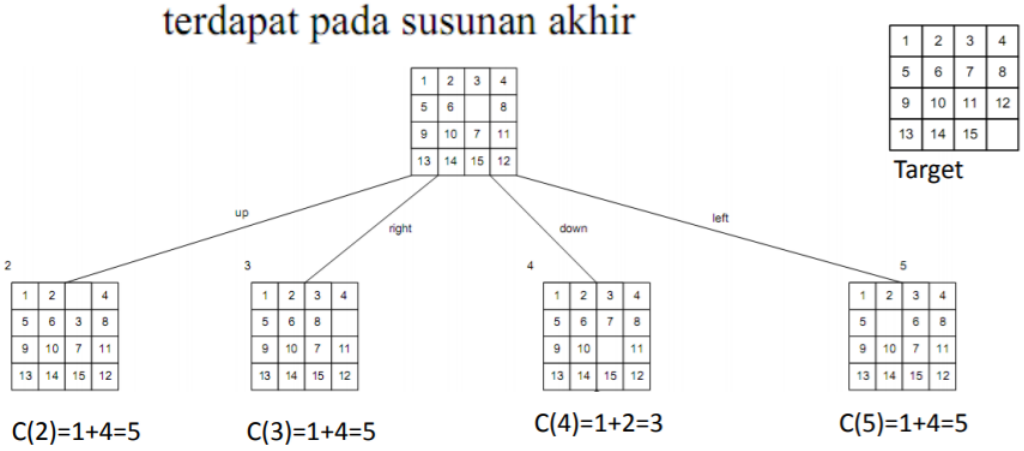
ALGORITMA BRANCH AND BOUND

Secara dasar, paradigma algoritma *branch and bound* digunakan untuk menyelesaikan permasalahan diskrit dan optimasi kombinatorial. Sesuai namanya, algoritma ini melakukan enumerasi suatu *node* dimulai dari tetangganya kemudian ke tetangga dari tetangganya sampai menemukan solusi yang diinginkan.

Akan tetapi, berbeda dengan *Breadth-first search*, algoritma ini mempunyai suatu *bound* (*cost*) yang digunakan untuk menentukan prioritas *node*-nya. Jadi, simulasi algoritma ini adalah memasukkan *node* pada *priority_queue* dengan *cost* terendah ada di indeks sedepan mungkin sehingga ada kemungkinan *node* tetangga yang lebih jauh ada di prioritas lebih depan dibanding tetangga yang lebih dekat.

Pada permainan 15Puzzle ini, nilai *cost* – anggap **c(P)** dapat dihitung dengan menambahkan panjang lintasan dari simpul akar ke simpul P – anggap **f(P)** dengan taksiran panjang lintasan terpendek dari simpul P ke simpul solusi pada upapohon dengan simpul P sebagai simpul akar baru – anggap **g(P)**.

$\hat{g}(P)$ = jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir



Pada kasus gambar diatas, terdapat 4 buah simpul tetangga yang dapat dibangkitkan, karena permainan barusan dimulai dan pergerakan **UP**, **DOWN**, **LEFT**, **RIGHT** masih memungkinkan.

Pada **c(2)**, nilai **f(2)** adalah 1 karena dibutuhkan 1 langkah dari simpul akar ke simpul 2, sedangkan **g(2)** bernilai 4 karena terdapat 4 ubin tidak kosong yang tidak terdapat pad susunan akhirnya. Nilai **c(3)**, **c(4)**, dan **c(5)** mengikuti prosedur yang sama.

Setelah mendapatkan nilai pada keempat **c**, status keempat simpul dimasukkan ke dalam *priority_queue* dengan urutan **c(3) - c(2) - c(4) - c(5).**

Selanjutnya simpul **3** di-pop dari *queue* tersebut dan dibangkitkan semua simpul tetangga yang memungkinkan. Walaupun keempat gerakan **UP**, **DOWN**, **LEFT**, **RIGHT** masih memungkinkan, gerakan yang hanya dibangkitkan hanyalah **DOWN**, **LEFT**, **RIGHT**, karena jika dilakukan **UP**, maka akan kembali ke simpul sebelumnya.

Anggap gerakan **DOWN**, **LEFT**, **RIGHT** berturut-turut adalah simpul 6, 7, 8, maka nilai **g(6)**, **g(7)**, dan **g(8)** adalah 2 karena telah memakan 2 gerakan dari simpul paling awal, sehingga secara berturut-turut, nilai **c(6)**, **c(7)**, dan **c(8)** adalah 5, 5, dan 3.

Kemudian pada *priority_queue*, urutannya menjadi **c(8) - c(2) - c(4) - c(5) - c(6) - c(7)**. Kemudian simpul **8** di-pop dan proses berlanjut hingga *priority_queue* kosong atau status simpul mencapai hasil final yang diinginkan.

Secara dasar, implementasi program utama adalah sebagai berikut:

```
from utils import printFail, printSuccess, printTime
from utils import fileToIntArray, setupOutputFile, desiredOutput
from rule import isReachable, getSiblingIndices, cost, decideNextArray, enque, printArrayWithWalkthrough
from datetime import datetime


start_time = datetime.now()
setupOutputFile()
array = fileToIntArray()
desiredArray = desiredOutput(array)

if not isReachable(array):
    printFail()
else:
    nextArray = [num for num in array]
    path = 0
    nextData = {'sequence': nextArray, 'cost': 0, 'prevAction': '', 'path': path, 'walkThrough': ''}
    progress = 0
    queue = []

    while nextArray != desiredArray:
        siblingCost = []
        siblingIndices = getSiblingIndices(nextData)

        for index in siblingIndices:
            siblingCost.append(cost(nextArray, index, path, desiredArray))

        enque(nextData, siblingIndices, siblingCost, queue, path)

        nextData = queue.pop(0)
        nextArray = nextData['sequence']
        path = nextData['path']
        progress += 1

    printArrayWithWalkthrough(array, nextData['walkThrough'])
    printSuccess(progress)

end_time = datetime.now()
printTime(str(start_time), str(end_time))
```

Gambar 2. Implementasi program utama

queue merupakan *priority_queue* yang digunakan untuk menyimpan kumpulan **nextData**

nextData merupakan data terstruktur yang digunakan untuk menyimpan status suatu simpul.

- **sequence** menyimpan kombinasi ubin dalam bentuk *array*.
- **cost** menyimpan nilai **c** pada simpul tersebut.
- **prevAction** menyimpan nilai gerakan terakhir untuk mencapai simpul tersebut.
- **path** menyimpan nilai berapa gerakan yang telah dilakukan dari status simpul awal hingga status simpul tersebut.
- **walkthrough** menyimpan langkah-langkah yang diambil dari simpul awal hingga mencapai simpul tersebut.

Kemudian selama **sequence** dari **nextData** belum mencapai status simpul final, akan dilakukan pencarian indeks tetangga yang valid melalui fungsi **getSiblingIndices.**

```
def getSiblingIndices(data):
    array = data['sequence']
    index = array.index(EMPTY)
    gameSize = int(sqrt(len(array) + 1))
    sibling = []

    if index == 0:
        sibling = [index + 1, index + gameSize]
    elif index == gameSize - 1:
        sibling = [index - 1, index + gameSize]
    elif index == len(array) - 1:
        sibling = [index - gameSize, index - 1]
    elif index == len(array) - gameSize:
        sibling = [index - gameSize, index + 1]
    elif index < gameSize:
        sibling = [index - 1, index + 1, index + gameSize]
    elif index > len(array) - gameSize:
        sibling = [index - gameSize, index - 1, index + 1]
    elif index % gameSize == 0:
        sibling = [index - gameSize, index + 1, index + gameSize]
    elif (index + 1) % gameSize == 0:
        sibling = [index - gameSize, index - 1, index + gameSize]
    else:
        sibling = [index - gameSize, index - 1, index + 1, index + gameSize]

    if data['prevAction'] == 'DOWN':
        sibling.remove(index - gameSize)
    if data['prevAction'] == 'UP':
        sibling.remove(index + gameSize)
    if data['prevAction'] == 'LEFT':
        sibling.remove(index + 1)
    if data['prevAction'] == 'RIGHT':
        sibling.remove(index - 1)

    return sibling
```

Gambar 3. Implementasi **getSiblingIndices**

Kemudian, dari nilai **siblingIndices** akan dihitung **siblingCost**-nya masing-masing dengan memanggil fungsi **cost.**

```python
def cost(array, index, path, desiredArray):
    emptyIndex = array.index(EMPTY)
    count = 0
    swappedArray = swapped(array, emptyIndex, index)
    for i in range(len(swappedArray)):
        if(swappedArray[i] != desiredArray[i] and swappedArray[i] != EMPTY):
            count += 1
    return count + path
```

Gambar 4. Implementasi **cost**

Fungsi **swapped** pada **cost** hanya mengembalikan suatu array dengan menukarkan elemen di indeks **emptyIndex** dengan **index** pada **array**.

Setelah nilai **cost** dari setiap **siblingIndices** disimpan pada **siblingCost**, akan dilakukan **enque** (proses memasukkan status simpul yang digenerasi ke dalam **queue**).

```python
def enque(nextData, siblingIndices, siblingCost, queue, path):
    array = nextData['sequence']
    emptyIndex = array.index(EMPTY)
    gameSize = int(sqrt(len(array) + 1))

    for i in range(len(siblingIndices)):
        sequence = swapped(array, emptyIndex, siblingIndices[i])
        if(siblingIndices[i] - emptyIndex == gameSize):
            prevAction = "DOWN"
        if(siblingIndices[i] - emptyIndex == gameSize * -1):
            prevAction = "UP"
        if(siblingIndices[i] - emptyIndex == 1):
            prevAction = "RIGHT"
        if(siblingIndices[i] - emptyIndex == -1):
            prevAction = "LEFT"

        data = {
            'sequence': sequence,
            'cost': siblingCost[i],
            'prevAction': prevAction,
            'path': path + 1,
            'walkThrough': nextData['walkThrough'] + '-' + prevAction
        }

        if sequenceExist(queue, sequence):
            continue
        else:
            queue.append(data)
    queue.sort(key=doc)
```

Gambar 5. Implementasi **enque**

Setelah proses **enque** selesai, akan dilakukan **pop** pada **queue** dan **nextData** yang telah di-**pop** menjadi **nextData** selanjutnya; proses ini berlanjut hingga **sequence** dari **nextData** mencapai status simpul final yang diinginkan. Setelah itu, dari simpul awal dilakukan gerakan sesuai dengan **walkThrough** dari **nextData** yang telah final dengan prosedur **printArrayWithWalkthrough**.

```
def printArrayWithWalkthrough(array, walkThrough):
    steps = walkThrough.split('-')[1:]
    gameSize = int(sqrt(len(array) + 1))
    nextArray = [a for a in array]
    move = 0

    for step in steps:
        emptyIndex = nextArray.index(EMPTY)
        move += 1
        if step == 'LEFT':
            index = emptyIndex - 1
        elif step == 'RIGHT':
            index = emptyIndex + 1
        elif step == 'UP':
            index = emptyIndex - gameSize
        elif step == 'DOWN':
            index = emptyIndex + gameSize
        nextArray = swapped(nextArray, emptyIndex, index)
        printArrayWithProgress(nextArray, move, step)
```

Gambar 5. Implementasi **enque**

PrintArrayWithProgress hanya menampilkan perubahan gerakan ke-**step** dari **nextArray** dengan gerakan **move**.

NB:

```
start_time = datetime.now()
setupOutputFile()
array = fileToIntArray()
desiredArray = desiredOutput(array)
```

- **desiredArray** merupakan status simpul final yang diinginkan.
- **fileToIntArray** berfungsi mengonversi input file ke dalam state sebuah **array**

BAB III

SCREENSHOT HASIL PROGRAM

3.1. INPUT YANG BISA DISELESAIKAN DAN OUTPUT

Input:

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

Output:

```
X-Squared Puzzle Game by Jones Napoleon
=======================================

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

Fungsi Kurang(1) -> 0
Fungsi Kurang(3) -> 1
Fungsi Kurang(4) -> 1
Fungsi Kurang(8) -> 4
Fungsi Kurang(9) -> 4
Fungsi Kurang(2) -> 0
Fungsi Kurang(7) -> 2
Fungsi Kurang(6) -> 1
Fungsi Kurang(13) -> 4
Fungsi Kurang(5) -> 0
Fungsi Kurang(0) -> 5
Fungsi Kurang(11) -> 1
Fungsi Kurang(14) -> 2
Fungsi Kurang(10) -> 0
Fungsi Kurang(15) -> 1
Fungsi Kurang(12) -> 0
Sigma Fungsi Kurang + X -> 26


Matrix after progress 1
========================
Last move: UP
========================
1 3 4 8
9 2 0 6
13 5 7 11
14 10 15 12
==============

Matrix after progress 2
========================
Last move: RIGHT
========================
1 3 4 8
9 2 6 0
13 5 7 11
14 10 15 12
==============
```

Matrix after progress 3
==========
Last move: UP
==========
1 3 4 0
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 4
==========
Last move: LEFT
==========
1 3 0 4
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 5
==========
Last move: LEFT
==========
1 0 3 4
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 6
==========
Last move: DOWN
==========
1 2 3 4
9 0 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 7
==========
Last move: DOWN
==========
1 2 3 4
9 5 6 8
13 0 7 11
14 10 15 12
==========

Matrix after progress 8
==========
Last move: DOWN
==========
1 2 3 4
9 5 6 8
13 10 7 11
14 0 15 12
==========

Matrix after progress 9
==========
Last move: LEFT
==========
1 2 3 4
9 5 6 8
13 10 7 11

0 14 15 12
==========

Matrix after progress 10
==========
Last move: UP
==========
1 2 3 4
9 5 6 8
0 10 7 11
13 14 15 12
==========

Matrix after progress 11
==========
Last move: UP
==========
1 2 3 4
0 5 6 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 12
==========
Last move: RIGHT
==========
1 2 3 4
5 0 6 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 13
==========
Last move: RIGHT
==========
1 2 3 4
5 6 0 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 14
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 10 0 11
13 14 15 12
==========

Matrix after progress 15
==========
Last move: RIGHT
==========
1 2 3 4
5 6 7 8
9 10 11 0
13 14 15 12
==========

Matrix after progress 16
==========
Last move: DOWN
==========
1 2 3 4

5 6 7 8
9 10 11 12
13 14 15 0
==========

Decision
==========
This puzzle is solved after raising 158 nodes!
==========

Execution time
==========
0.11940899999899557 seconds
==========

3.2. INPUT YANG BISA DISELESAIKAN DAN OUTPUT

Input:

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

Output:

X-Squared Puzzle Game by Jones Napoleon
==========

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

Fungsi Kurang(3) -> 2
Fungsi Kurang(0) -> 14
Fungsi Kurang(4) -> 2
Fungsi Kurang(8) -> 5
Fungsi Kurang(1) -> 0
Fungsi Kurang(2) -> 0
Fungsi Kurang(7) -> 2
Fungsi Kurang(6) -> 1
Fungsi Kurang(9) -> 1
Fungsi Kurang(13) -> 4
Fungsi Kurang(5) -> 0
Fungsi Kurang(11) -> 1
Fungsi Kurang(14) -> 2
Fungsi Kurang(10) -> 0
Fungsi Kurang(15) -> 1
Fungsi Kurang(12) -> 0
Sigma Fungsi Kurang + X -> 36

Matrix after progress 1
==========
Last move: LEFT
==========
0 3 4 8
1 2 7 6
9 13 5 11
14 10 15 12

==========

Matrix after progress 2
==========
Last move: DOWN
==========
1 3 4 8
0 2 7 6
9 13 5 11
14 10 15 12
==========

Matrix after progress 3
==========
Last move: DOWN
==========
1 3 4 8
9 2 7 6
0 13 5 11
14 10 15 12
==========

Matrix after progress 4
==========
Last move: RIGHT
==========
1 3 4 8
9 2 7 6
13 0 5 11
14 10 15 12
==========

Matrix after progress 5
==========
Last move: RIGHT
==========
1 3 4 8
9 2 7 6
13 5 0 11
14 10 15 12
==========

Matrix after progress 6
==========
Last move: UP
==========
1 3 4 8
9 2 0 6
13 5 7 11
14 10 15 12
==========

Matrix after progress 7
==========
Last move: RIGHT
==========
1 3 4 8
9 2 6 0
13 5 7 11
14 10 15 12
==========

Matrix after progress 8
==========
Last move: UP
==========
1 3 4 0
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 9
==========
Last move: LEFT
==========
1 3 0 4
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 10
==========
Last move: LEFT
==========
1 0 3 4
9 2 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 11
==========
Last move: DOWN
==========
1 2 3 4
9 0 6 8
13 5 7 11
14 10 15 12
==========

Matrix after progress 12
==========
Last move: DOWN
==========
1 2 3 4
9 5 6 8
13 0 7 11
14 10 15 12
==========

Matrix after progress 13
==========
Last move: DOWN
==========
1 2 3 4
9 5 6 8
13 10 7 11
14 0 15 12
==========

Matrix after progress 14
==========

Last move: LEFT
==========
1 2 3 4
9 5 6 8
13 10 7 11
0 14 15 12
==========

Matrix after progress 15
==========
Last move: UP
==========
1 2 3 4
9 5 6 8
0 10 7 11
13 14 15 12
==========

Matrix after progress 16
==========
Last move: UP
==========
1 2 3 4
0 5 6 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 17
==========
Last move: RIGHT
==========
1 2 3 4
5 0 6 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 18
==========
Last move: RIGHT
==========
1 2 3 4
5 6 0 8
9 10 7 11
13 14 15 12
==========

Matrix after progress 19
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 10 0 11
13 14 15 12
==========

Matrix after progress 20
==========
Last move: RIGHT
==========

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

Matrix after progress 21
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0
==========

Decision
==========
This puzzle is solved after raising 2907 nodes!
==========

Execution time
==========
1.0952890000044135 seconds
==========

3.3. INPUT YANG BISA DISELESAIKAN DAN OUTPUT

Input:

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

Output:

X-Squared Puzzle Game by Jones Napoleon
==========

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

Fungsi Kurang(5) -> 4
Fungsi Kurang(1) -> 0
Fungsi Kurang(2) -> 0
Fungsi Kurang(4) -> 1
Fungsi Kurang(6) -> 1
Fungsi Kurang(15) -> 9
Fungsi Kurang(3) -> 0
Fungsi Kurang(7) -> 0
Fungsi Kurang(9) -> 1
Fungsi Kurang(10) -> 1
Fungsi Kurang(11) -> 1
Fungsi Kurang(8) -> 0
Fungsi Kurang(13) -> 1
Fungsi Kurang(0) -> 2

Fungsi Kurang(14) -> 1
Fungsi Kurang(12) -> 0
Sigma Fungsi Kurang + X -> 22

Matrix after progress 1
==========
Last move: UP
==========
5 1 2 4
6 15 3 7
9 0 11 8
13 10 14 12
==========

Matrix after progress 2
==========
Last move: UP
==========
5 1 2 4
6 0 3 7
9 15 11 8
13 10 14 12
==========

Matrix after progress 3
==========
Last move: LEFT
==========
5 1 2 4
0 6 3 7
9 15 11 8
13 10 14 12
==========

Matrix after progress 4
==========
Last move: UP
==========
0 1 2 4
5 6 3 7
9 15 11 8
13 10 14 12
==========

Matrix after progress 5
==========
Last move: RIGHT
==========
1 0 2 4
5 6 3 7
9 15 11 8
13 10 14 12
==========

Matrix after progress 6
==========
Last move: RIGHT
==========
1 2 0 4
5 6 3 7
9 15 11 8

13 10 14 12
==========

Matrix after progress 7
==========
Last move: DOWN
==========
1 2 3 4
5 6 0 7
9 15 11 8
13 10 14 12
==========

Matrix after progress 8
==========
Last move: RIGHT
==========
1 2 3 4
5 6 7 0
9 15 11 8
13 10 14 12
==========

Matrix after progress 9
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 15 11 0
13 10 14 12
==========

Matrix after progress 10
==========
Last move: LEFT
==========
1 2 3 4
5 6 7 8
9 15 0 11
13 10 14 12
==========

Matrix after progress 11
==========
Last move: LEFT
==========
1 2 3 4
5 6 7 8
9 0 15 11
13 10 14 12
==========

Matrix after progress 12
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 10 15 11
13 0 14 12
==========

Matrix after progress 13
==========
Last move: RIGHT
==========
1 2 3 4
5 6 7 8
9 10 15 11
13 14 0 12
==========

Matrix after progress 14
==========
Last move: UP
==========
1 2 3 4
5 6 7 8
9 10 0 11
13 14 15 12
==========

Matrix after progress 15
==========
Last move: RIGHT
==========
1 2 3 4
5 6 7 8
9 10 11 0
13 14 15 12
==========

Matrix after progress 16
==========
Last move: DOWN
==========
1 2 3 4
5 6 7 8
9 10 11 12
13 14 15 0
==========

Decision
==========
This puzzle is solved after raising 258 nodes!
==========

Execution time
==========
0.2783029999991413 seconds
==========

3.4. INPUT YANG TIDAK BISA DISELESAIKAN DAN OUTPUT

Input:

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

Output:

X-Squared Puzzle Game by Jones Napoleon
==

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

Fungsi Kurang(1) -> 0
Fungsi Kurang(3) -> 1
Fungsi Kurang(4) -> 1
Fungsi Kurang(15) -> 11
Fungsi Kurang(2) -> 0
Fungsi Kurang(0) -> 10
Fungsi Kurang(5) -> 0
Fungsi Kurang(12) -> 6
Fungsi Kurang(7) -> 1
Fungsi Kurang(6) -> 0
Fungsi Kurang(11) -> 3
Fungsi Kurang(14) -> 4
Fungsi Kurang(8) -> 0
Fungsi Kurang(9) -> 0
Fungsi Kurang(10) -> 0
Fungsi Kurang(13) -> 0
Sigma Fungsi Kurang + X -> 37

Decision
==========
This puzzle cannot be solved!
==========

Execution time
==============
710.309000001871 miliseconds
==============

3.5. INPUT YANG TIDAK BISA DISELESAIKAN DAN OUTPUT

Input:

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

Output:

X-Squared Puzzle Game by Jones Napoleon
==

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

Fungsi Kurang(7) -> 6
Fungsi Kurang(15) -> 13
Fungsi Kurang(12) -> 10
Fungsi Kurang(11) -> 9
Fungsi Kurang(3) -> 2
Fungsi Kurang(9) -> 6
Fungsi Kurang(5) -> 3
Fungsi Kurang(10) -> 5
Fungsi Kurang(14) -> 6
Fungsi Kurang(0) -> 6
Fungsi Kurang(8) -> 4
Fungsi Kurang(2) -> 1
Fungsi Kurang(4) -> 1
Fungsi Kurang(13) -> 2
Fungsi Kurang(1) -> 0
Fungsi Kurang(6) -> 0
Sigma Fungsi Kurang + X -> 75

Decision
==========
This puzzle cannot be solved!
==========

Execution time
==========
0.2408900000009453 seconds
==========

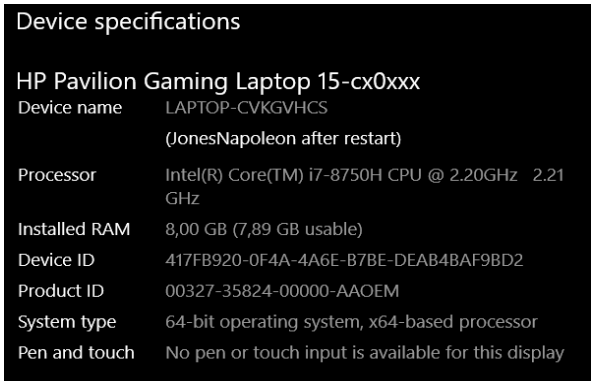
BAB IV

LAIN-LAIN

4.1. PENILAIAN INDIVIDU

No	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 n	√	

4.2. SPESIFIKASI PERANGKAT KERAS YANG DIGUNAKAN



Gambar 20. Spesifikasi Laptop *author* (Jones Napoleon)