LAPORAN

TUGAS KECIL

IF 2211 Strategi Algoritma

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

Disusun oleh:

William 13518138

Prodi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2020

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Buatlah program dalam Python untuk menyelesaikan persoalan 15-Puzzle dengan menggunakan Algoritma Branch and Bound seperti pada materi kuliah. 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. Taksiran cost yang digunakan adalah jumlah ubin tidak kosong yang tidak berada pada tempat sesuai susunan akhir (goal state). Untuk semua instansiasi persoalan 15-puzzle, susunan akhir yang diinginkan sesuai dengan Gambar 1.

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

Gambar 1. Susunan Akhir persoalan 15-puzzle

Masukan: matriks yang merepresentasikan posisi awal suatu instansiasi persoalan 15-puzzle. Matriks dibaca dari berkas teks.

Program harus dapat menentukan apakah posisi awal suatu masukan dapat diselesaikan hingga mencapai susunan akhir, dengan mengimplementasikan fungsi Kurang(i) dan posisi ubin kosong di kondisi awal (X), seperti pada materi kuliah. Jika posisi awal tidak bisa mencapai susunan akhir, program akan menampilkan pesan tidak bisa diselesaikan,. Jika dapat diselesaikan, program dapat menampilkan urutan matriks rute (path) aksi yang dilakukan dari posisi awal ke susunan akhir. Sebagai contoh pada Gambar 2, matriks yang ditampilkan ke layar adalah matriks pada simpul 1, simpul 4, simpul 10 dan simpul 23.

Keluaran:

1. Matriks posisi awal.
2. Nilai dari fungsi Kurang (i) untuk setiap ubin tidak kosong pada posisi awal (nilai ini tetap dikeluarkan, baik persoalan bisa diselesaikan atau tidak bisa diselesaikan).

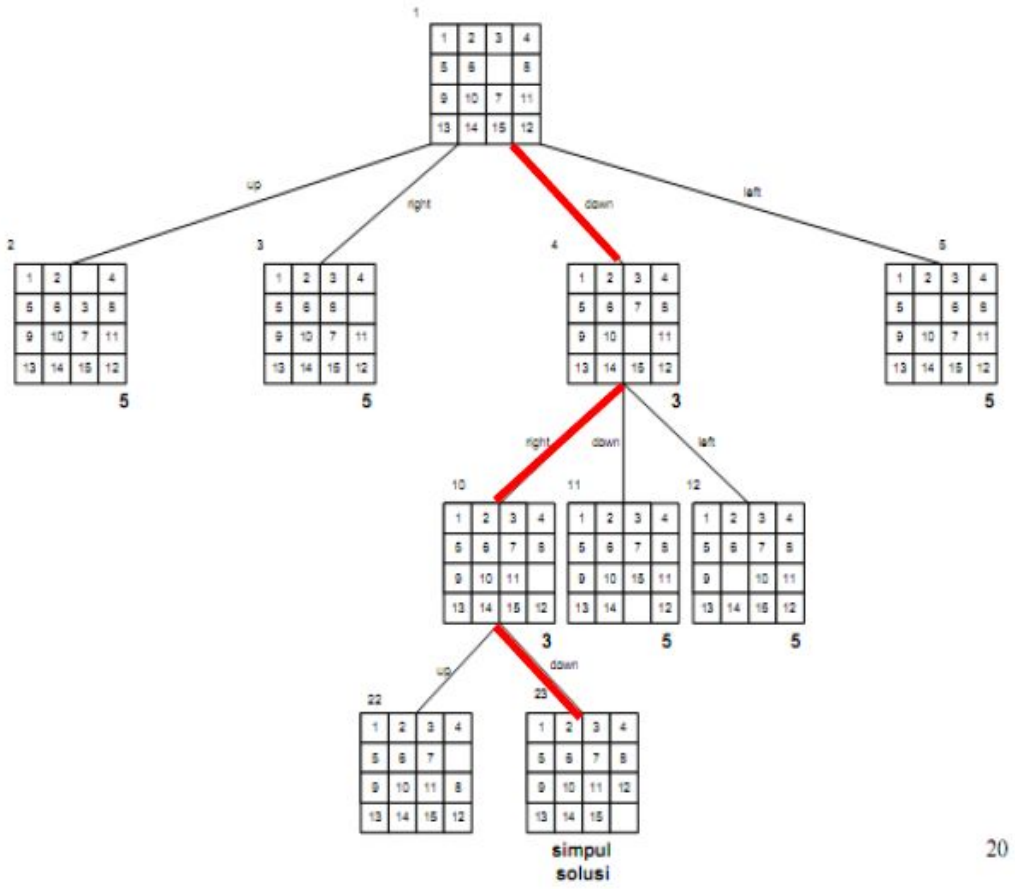
3. Nilai dari $\sum_{i=1}^{16} KURANG(i) + X$

4. Jika persoalan tidak dapat diselesaikan (berdasarkan hasil butir 2) keluar pesan.

5. Jika persoalan dapat diselesaikan(berdasarkan hasil butir 2), menampilkan urutan matriks seperti pada penjelasan sebelummnya.

6. Waktu eksekusi

7. Jumlah simpul yang dibangkitkan dalam pohon ruang status.



Gambar 2. Contoh Pohon Ruang Status Persoalan 15-puzzle

BAB II

Algoritma Program

2.1 Algoritma Branch and Bound

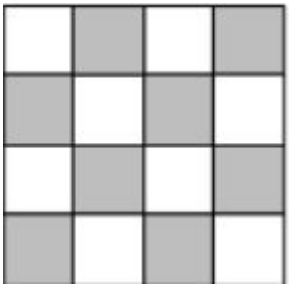
Algoritma Branch and Bound merupakan suatu algoritma yang berusaha menemukan suatu solusi dengan langkah minimum untuk persoalan optimasi, baik maksimasi maupun minimasi. Algoritma Branch and Bound memanfaatkan langkah - langkah dalam algoritma traversal graf Breadth First Search (BFS), namun disertai dengan langkah - langkah untuk mencari jalan (*path*) dengan cost terkecil. Dalam hal ini, BFS memberi *cost* / biaya pada tiap simpulnya. Simpul berikut yang akan diekspan oleh algoritma Branch and Bound bukanlah berdasarkan pada urutan pembangkitannya, melainkan pada simpul yang memiliki cost paling kecil (least cost search).

Dikarenakan natur algoritma Branch and Bound yang berguna untuk menyelesaikan permasalahan optimasi, algoritma ini dapat digunakan untuk menemukan langkah minimum untuk menyelesaikan persoalan puzzle-15. Terdapat beberapa hal penting yang perlu didefinisikan, yaitu fungsi pembatas dan penentuan cara menghitung cost dari setiap simpul yang dibangkitkan pada tiap langkah algoritma Branch and Bound.

Dalam permainan Puzzle-15 yang menjadi persoalan dalam tugas kali ini, penulis menggunakan algoritma Branch and Bound untuk menyelesaikan puzzle. State dari permainan puzzle ditentukan berdasarkan letak dari ubin kosong (blank). Terdapat 4 (empat) aksi yang dapat dilakukan yaitu “up”, “down”, “left”, “right” berdasarkan letak dari ubin kosong.

Sebelum algoritma branch and bound diterapkan, permasalahan ini perlu dicek apakah dapat diselesaikan atau tidak, karena algoritma branch and bound hanya akan efektif terhadap permasalahan yang memiliki solusi. Karena hanya terdapat setengah kemungkinan dari 16! kemungkinan state yang dapat mencapai goal state dari state awal sembarang, maka penulis menerapkan teorema dimana status tujuan hanya dapat dicapai dari status awal jika

$\sum_{i=1}^{16} KURANG(i) + X$ bernilai genap. X = 1 jika pada sel yang diarsir state awal memiliki ubin kosong.



Untuk nilai fungsi KURANG(i) sendiri didefinisikan sebagai banyaknya ubin bernomor j sedemikian sehingga j < i dan POSISI(j) > POSISI(i) dimana POSISI(i) merujuk pada posisi ubin ke - i.

Sementara itu, dalam mencari cost suatu state puzzle-15, penulis menerapkan suatu taksiran yang merujuk terhadap berapa banyak langkah perpindahan ubin kosong dijumlahkan dengan jumlah ubin tidak kosong yang tidak terdapat pada susunan akhir pada state tersebut.

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

Dengan :

c(i) : ongkos untuk simpul state i

f(i) : ongkos mencapai simpul i dari akar

g(i) : ongkos mencapai simpul tujuan dari simpul i.

Langkah - langkah yang dilakukan penulis untuk mencari perjalanan state goal dari permainan puzzle-15 dengan algoritma Branch and Bound :

1. Memasukkan state puzzle awal ke dalam antrian (queue) Q. Dalam hal ini, jenis queue yang digunakan penulis adalah antrian prioritas (Priority Queue). Jika simpul akar adalah simpul solusi, maka solusi telah ditemukan dan proses dihentikan.
2. Jika Q kosong, maka solusi tidak ditemukan. Proses pencarian dihentikan
3. Jika Q tidak kosong, maka pilihlah dari antrian Q simpul i yang memiliki nilai cost terkecil yang dihitung melalui fungsi c(i) seperti didefinisikan di atas. Jika terdapat

beberapa simpul yang memenuhi, algoritma yang dibuat penulis akan memilih berdasar urutan pembangkitan paling lama simpul dengan cost terkecil.

4. Jika simpul yang dipilih tersebut, maka solusi telah ditemukan, pencarian dihentikan. Namun, apabila simpul tersebut bukanlah simpul solusi, maka bangkitkan lah semua anak - anak simpul tersebut (simpul anak dibentuk melalui perpindahan ubin kosong ke 4 (empat) arah yang memungkinkan, yakni “up”, “down”, “left”, “down”).
5. Untuk tiap simpul anak yang dibangkitkan pada langkah 4, hitunglah cost simpul anak tersebut, dan masukkan seluruh simpul anak tersebut pada antrian
6. Kembali ke langkah 2.

2.2 Analisis Kompleksitas Algoritma

Dalam mencari simpul solusi dari pemainan puzzle-15 dengan algoritma branch and bound, diperlukan proses pembangkitan simpul berdasarkan pada least cost search. Karena pembangkitan simpul tersebut mengikuti pola pembangkitan pada algoritma BFS (Breadth First Search), maka kompleksitas algoritma untuk permasalahan ini dapat diukur menjadi :

Kompleksitas Waktu Asimptotik : $O(b^d) = O(3^d)$

Kompleksitas Ruang Asimptotik : $O(b^d) = O(3^d)$

dengan :

b: jumlah maksimal branch yang terbentuk, dimana terdapat maksimal 3 langkah yang dapat dilakukan oleh puzzle. (Kecuali state awal, dimana dapat dilakukan 4 langkah (“up”,”down”,”left”,”right”).

d : jumlah kedalaman maksimum yang terbentuk

BAB III

Kode Program

Class Puzzle

```
import copy

def pretty_print(matrix):
  for line in matrix:
    print(line)

class Puzzle:
  def __init__(self, matrix,n, path, previous):
    self.matrix = matrix
    self.dimension = n
    self.path = path
    self.previous = previous

  def __str__(self):
    return self.path

  def print_matrix(self):
    pretty_print(self.getMatrix())

  def isEqual(self,puzzle):
    found = False
    for i in range(self.dimension):
      for j in range(self.dimension):
        if (self.matrix[i][j] != puzzle.getMatrix()[i][j]):
          found = True
    return not found

  def posisi_blank(self):
    return self.posisi_row_column(0)

  def checkX(self):
    return 1 if (self.posisi_blank()[0] + self.posisi_blank()[1]) % 2 == 1 else 0

  def getMatrix(self):
    return copy.deepcopy(self.matrix)

  def posisi(self, val):
    i, j = self.posisi_row_column(val)
    return i * self.dimension + j

  def posisi_row_column(self, val):
    i = 0; j = 0; found = False
    while(i < self.dimension or not found) :
      j = 0
      while (j < self.dimension and not found):
        if self.matrix[i][j] == val:
          found = True
          break
        else :
          j+=1
      if found: break
      else : i+=1
    return (i,j)
```

```
def cost(self):
  cost = 0
  for i in range(1, 15+1):
    if (self.posisi(i)+1) != i:
      cost += 1
  cost += len(self.path)
  return cost

def iterate_kurang(self):
  for i in range(1, self.dimension * self.dimension):
    print(i, ':' ,self.kurang(i))
  print(self.dimension * self.dimension, ':', self.kurang_blank())

def kurang(self, val):
  """
    For angka 1 - 15
  """
  total = 0
  pos = self.posisi(val)
  for i in range(1, val):
    if (self.posisi(i) > pos):
      total += 1
  return total

def kurang_blank(self):
  """
    For angka 0 (blank side of the puzzle)
  """
  kurang_awal = self.posisi(0)
  return self.dimension * self.dimension - kurang_awal - 1

def sum_of_kurang(self):
  total_kurang = self.checkX()
  for i in range(1, self.dimension * self.dimension):
    total_kurang += self.kurang(i)
  total_kurang += self.kurang_blank()
  return total_kurang

def validate_reachable(self):
  """
    Checking wether a puzzle can be solved or not via the sum_of_kurang
  """
  return self.sum_of_kurang() % 2 == 0

def validate_movement(self,movement):
  vertical, horizontal = self.posisi_row_column(0)
  if (movement == "U"):
    return True if (vertical != 0 and self.previous != "D")else False
  elif (movement == "D"):
    return True if (vertical != (self.dimension - 1) and self.previous != "U") else False
  elif (movement == "L"):
    return True if (horizontal != 0 and self.previous != "R") else False
  elif (movement == "R"):
    return True if (horizontal != (self.dimension - 1) and self.previous != "L") else False
  else :
    return False

def move(self, movement):
  if not self.validate_movement(movement):
```

```
      return -1
    arr = self.getMatrix()
    row, column = self.posisi_blank()
    if (movement == "U"):
        arr[row][column], arr[row-1][column] = arr[row-1][column], arr[row][column]
    if (movement == "D"):
      arr[row][column], arr[row+1][column] = arr[row+1][column], arr[row][column]
    if (movement == "L"):
      arr[row][column], arr[row][column-1] = arr[row][column - 1], arr[row][column]
    if (movement == "R"):
      arr[row][column], arr[row][column+1] = arr[row][column + 1], arr[row][column]
    return Puzzle(arr,4,self.path+movement, movement)
```

Class PrioQueue (PrioQueue.py)

```
import copy
class PriorityQueue(object):
   def __init__(self):
       self.queue = []
   # for checking if the queue is empty
   def isEmpty(self):
       return len(self.queue) == 0
    # for inserting an element in the queue
   def insert(self, data):
       self.queue.append(data)
    # for popping an element based on Priority
   def delete(self):
     if self.isEmpty():
       raise IndexError
     min = 0
     for i in range(len(self.queue)):
       if self.queue[i].cost() < self.queue[min].cost():
         min = i
     item = copy.deepcopy(self.queue[min])
     del self.queue[min]
     return item

   # delete after reaching goal state
   def delete_upper_bound(self, cost):
     if self.isEmpty():
       raise IndexError
     i = len(self.queue) - 1
     while (i >= 0) :
       if self.queue[i].cost() > cost:
         del self.queue[i]
       i-=1
```

Library Utility (utility.py)

```
import sys
from Puzzle import Puzzle
import copy
# Function to get testcase filename
def getFileName():
 return sys.argv[1]

# Function to preprocess
def preprocessing(lines):
 for i in range(len(lines)):
```

```
    lines[i] =  lines[i].strip().replace("\n","").split(" ")
    lines[i] = list(map(int, lines[i]))
  return lines

# Function to pretty print the result
def pretty_path(str_path):
  path = [elem for elem in str_path]
  print("Path : ", end="")
  for i in range(len(path)):
    print(path[i], end="")
    if i != len(path) -1:
      print("→", end="")
  print()

def print_path_repr(str_path, puzzle_start):
  print("Steps:")
  curr_puzzle = copy.deepcopy(puzzle_start)
  curr_puzzle.print_matrix()
  pretty_path(curr_puzzle.path)
  path = [elem for elem in str_path]
  for move in path:
    print("↓")
    curr_puzzle = copy.deepcopy(curr_puzzle.move(move))
    curr_puzzle.print_matrix()
    pretty_path(curr_puzzle.path)
```

Main Program (main.py)

```
import copy
from time import time
from utility import getFileName, preprocessing, pretty_path, print_path_repr
from Puzzle import Puzzle
from PrioQueue import PriorityQueue

if __name__ == '__main__':
  t = time()
  prio_queue = PriorityQueue()
  filename = getFileName()

  print("Reading Files...")
  with open('testcase/goal.txt' ,'r') as f:
    lines = f.readlines()
  matrix = preprocessing(lines)
  puzzle_goal = Puzzle(matrix,4,"", "")

  with open(filename, 'r') as f:
    lines = f.readlines()
  matrix = preprocessing(lines)
  puzzle_start = Puzzle(matrix, 4, "", "")

  # Print Matrix Awal:
  print("Initial Puzzle State:")
  puzzle_start.print_matrix()

  # Print letak kosong
  blank = puzzle_start.posisi_blank()
  print("Blank In : (", blank[0] + 1, ',', blank[1] + 1, ')')

  # Print Kurang
  print("Kurang : ")
```

```
puzzle_start.iterate_kurang()

# Print nilai dari sum of kurang + X
print("Sum of Kurang + X :", puzzle_start.sum_of_kurang())

# Start Algo
count = 0
if (puzzle_start.validate_reachable()):
  prio_queue.insert(copy.deepcopy(puzzle_start))
  found = False
  count+=1
  while(not prio_queue.isEmpty()):
    curr_puzzle = prio_queue.delete()
    if curr_puzzle.isEqual(puzzle_goal):
      found = True
      break
    else :
      if (curr_puzzle.validate_movement("U")):
        prio_queue.insert(copy.deepcopy(curr_puzzle.move("U")))
        count+=1
      if (curr_puzzle.validate_movement("D")):
        prio_queue.insert(copy.deepcopy(curr_puzzle.move("D")))
        count+=1
      if (curr_puzzle.validate_movement("L")):
        prio_queue.insert(copy.deepcopy(curr_puzzle.move("L")))
        count+=1
      if (curr_puzzle.validate_movement("R")):
        prio_queue.insert(copy.deepcopy(curr_puzzle.move("R")))
        count+=1
  if found:
    pretty_path(curr_puzzle.path)
    print()
    print_path_repr(curr_puzzle.path, puzzle_start)
    print("Jumlah simpul yang dibangkitkan :", count)
  else :
    print("Solution Not Found")
else:
  print("Solution is unreachable")

t = time() - t
print(f"Done! Time taken to solve the problem {t} seconds")
```

BAB IV

EKSPERIMEN

4.1 Hasil Tangkapan Layar untuk test case 1 (solvable)

```
C:\Users\William\Documents\dev\school\stima\Tucil-3>python main.py ./testcase/testcase1.txt
Reading Files...
Initial Puzzle State:
[1, 6, 2, 3]
[5, 7, 4, 0]
[9, 10, 11, 8]
[13, 14, 15, 12]
Blank In : ( 2 , 4 )
Kurang :
1 : 0
2 : 0
3 : 0
4 : 0
5 : 1
6 : 4
7 : 1
8 : 0
9 : 1
10 : 1
11 : 1
12 : 0
13 : 1
14 : 1
15 : 1
16 : 8
Sum of Kurang + X : 20
Path : L→L→U→R→R→D→D→D
```

```
Steps:
[1, 6, 2, 3]
[5, 7, 4, 0]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path :
↓
[1, 6, 2, 3]
[5, 7, 0, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L
↓
[1, 6, 2, 3]
[5, 0, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L→L
↓
[1, 0, 2, 3]
[5, 6, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L→L→U
↓
[1, 2, 0, 3]
[5, 6, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L→L→U→R
↓
[1, 2, 3, 0]
[5, 6, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L→L→U→R→R
↓
[1, 2, 3, 4]
[5, 6, 7, 0]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : L→L→U→R→R→D
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 0]
[13, 14, 15, 12]
Path : L→L→U→R→R→D→D
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
Path : L→L→U→R→R→D→D→D
Jumlah simpul yang dibangkitkan : 23
Done! Time taken to solve the problem 0.2287905216217041 seconds
```

4.2 Hasil Tangkapan Layar untuk test case 2 (solvable)

```
C:\Users\William\Documents\dev\school\stima\Tucil-3>python main.py ./testcase/testcase2.txt
Reading Files...
Initial Puzzle State:
[1, 2, 3, 4]
[5, 6, 0, 8]
[9, 10, 7, 11]
[13, 14, 15, 12]
Blank In : ( 2 , 3 )
Kurang :
1 : 0
2 : 0
3 : 0
4 : 0
5 : 0
6 : 0
7 : 0
8 : 1
9 : 1
10 : 1
11 : 0
12 : 0
13 : 1
14 : 1
15 : 1
16 : 9
Sum of Kurang + X : 16
Path : D→R→D

Steps:
[1, 2, 3, 4]
[5, 6, 0, 8]
[9, 10, 7, 11]
[13, 14, 15, 12]
Path :
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 0, 11]
[13, 14, 15, 12]
Path : D
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 0]
[13, 14, 15, 12]
Path : D→R
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
Path : D→R→D
Jumlah simpul yang dibangkitkan : 10
Done! Time taken to solve the problem 0.12276530265808105 seconds
```

4.3 Hasil Tangkapan Layar untuk test case 3 (solvable)

```
C:\Users\William\Documents\dev\school\stima\Tucil-3>python main.py ./testcase/testcase3.txt
Reading Files...
Initial Puzzle State:
[5, 1, 7, 3]
[9, 2, 11, 4]
[13, 6, 15, 8]
[0, 10, 14, 12]
Blank In : ( 4 , 1 )
Kurang :
1 : 0
2 : 0
3 : 1
4 : 0
5 : 4
6 : 0
7 : 4
8 : 0
9 : 4
10 : 0
11 : 4
12 : 0
13 : 4
14 : 1
15 : 4
16 : 3
Sum of Kurang + X : 30
Path : U→U→U→R→D→D→D→R→U→U→U→R→D→D→D
```

```
Steps:
[5, 1, 7, 3]
[9, 2, 11, 4]
[13, 6, 15, 8]
[0, 10, 14, 12]
Path :
↓
[5, 1, 7, 3]
[9, 2, 11, 4]
[0, 6, 15, 8]
[13, 10, 14, 12]
Path : U
↓
[5, 1, 7, 3]
[0, 2, 11, 4]
[9, 6, 15, 8]
[13, 10, 14, 12]
Path : U→U
↓
[0, 1, 7, 3]
[5, 2, 11, 4]
[9, 6, 15, 8]
[13, 10, 14, 12]
Path : U→U→U
↓
[1, 0, 7, 3]
[5, 2, 11, 4]
[9, 6, 15, 8]
[13, 10, 14, 12]
Path : U→U→U→R
↓
[1, 2, 7, 3]
[5, 0, 11, 4]
[9, 6, 15, 8]
[13, 10, 14, 12]
Path : U→U→U→R→D
↓
[1, 2, 7, 3]
[5, 6, 11, 4]
[9, 0, 15, 8]
[13, 10, 14, 12]
Path : U→U→U→R→D→D
↓
[1, 2, 7, 3]
[5, 6, 11, 4]
[9, 10, 15, 8]
[13, 0, 14, 12]
Path : U→U→U→R→D→D→D
↓
[1, 2, 7, 3]
[5, 6, 11, 4]
[9, 10, 15, 8]
[13, 14, 0, 12]
Path : U→U→U→R→D→D→D→R
```

```
↓
[1, 2, 7, 3]
[5, 6, 11, 4]
[9, 10, 0, 8]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U
↓
[1, 2, 7, 3]
[5, 6, 0, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U→U
↓
[1, 2, 0, 3]
[5, 6, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U→U→U
↓
[1, 2, 3, 0]
[5, 6, 7, 4]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U→U→U→R
↓
[1, 2, 3, 4]
[5, 6, 7, 0]
[9, 10, 11, 8]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U→U→U→R→D
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 0]
[13, 14, 15, 12]
Path : U→U→U→R→D→D→D→R→U→U→U→R→D→D
↓
[1, 2, 3, 4]
[5, 6, 7, 8]
[9, 10, 11, 12]
[13, 14, 15, 0]
Path : U→U→U→R→D→D→D→R→U→U→U→R→D→D→D
Jumlah simpul yang dibangkitkan : 33
Done! Time taken to solve the problem 0.22334980964660645 seconds
```

4.4 Hasil Tangkapan Layar untuk test case 4 (unsolvable)

```
C:\Users\William\Documents\dev\school\stima\Tucil-3>python main.py testcase/testcase4.txt
Reading Files...
Initial Puzzle State:
[1, 3, 4, 15]
[2, 0, 5, 12]
[7, 6, 11, 14]
[8, 9, 10, 13]
Blank In : ( 2 , 2 )
Kurang :
1 : 0
2 : 0
3 : 1
4 : 1
5 : 0
6 : 0
7 : 1
8 : 0
9 : 0
10 : 0
11 : 3
12 : 6
13 : 0
14 : 4
15 : 11
16 : 10
Sum of Kurang + X : 37
Solution is unreachable
Done! Time taken to solve the problem 0.06559300422668457 seconds
```

4.5 Hasil Tangkapan Layar untuk test case 5 (unsolvable)

```
C:\Users\William\Documents\dev\school\stima\Tucil-3>python main.py testcase/testcase5.txt
Reading Files...
Initial Puzzle State:
[3, 9, 1, 15]
[14, 11, 4, 6]
[13, 0, 10, 12]
[2, 7, 8, 5]
Blank In : ( 3 , 2 )
Kurang :
1 : 0
2 : 0
3 : 2
4 : 1
5 : 0
6 : 2
7 : 1
8 : 1
9 : 7
10 : 4
11 : 7
12 : 4
13 : 6
14 : 10
15 : 11
16 : 6
Sum of Kurang + X : 63
Solution is unreachable
Done! Time taken to solve the problem 0.07280921936035156 seconds
```

4.6 Spesifikasi Perangkat Keras

4.7 Tambahan

Berikut merupakan tabel checklist dari hasil implementasi program penulis. Masukan program menggunakan *text file* yang telah disediakan pada folder testcase, serta pengecekan dilakukan dengan cara mencocokan hasil yang diperoleh program dengan bantuan aplikasi - aplikasi lain dari internet seperti program pada link ini (https://www.jaapsch.net/puzzles/javascript/fifteenj.htm).

Poin	Ya	Tidak
1. Program berhasil dikompilasi	✓	
2. Program berhasil *running*	✓	
3. Program dapat menerima input dan mengeluarkan output	✓	
4. Luaran sudah benar untuk semua data uji	✓	

DAFTAR REFERENSI

- Design and Analysis of Algorithm 3rd Edition, Anany Levitin, 2012, Pearson Education
- https://devdocs.io/python/
- http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Algoritma-Branch-&-Bound-(2018).pdf