

Laporan Tugas Kecil IF2211 Strategi Algoritma Penyelesaian 15-Puzzle dengan Algoritma *Branch and Bound*

Disusun oleh
Samuel Christopher - 13520075



Teknik Informatika

Institut Teknologi Bandung

Bandung

2022

I. Algoritma *Branch and Bound*

Algoritma *Branch and Bound* adalah algoritma yang memanfaatkan *Priority Queue* untuk memecahkan permasalahan. Dapat dikatakan bahwa algoritma ini merupakan penggabungan dari BFS dan *Least Cost Search*.

Untuk menyelesaikan persoalan 15-puzzle ini, maka dibutuhkan beberapa tahap sebelum masuk ke dalam algoritma.

1. Reachable Goal

Reachable Goal akan mengecek apakah sebuah puzzle dapat diselesaikan atau tidak dengan menggunakan fungsi

$$\sum_{i=1}^{16} KURANG(i) + X$$

Jika fungsi tersebut bernilai genap, maka puzzle dapat diselesaikan.

2. Fungsi Kurang(i)

Fungsi ini berarti banyaknya ubin bernomor j sedemikian sehingga $j < i$ dan $POSISI(j) > POSISI(i)$. $POSISI(i)$ = posisi ubin bernomor i pada susunan yang diperiksa

i	Kurang (i)
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

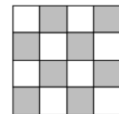
$X=1$ jika sel kosong pada posisi awal ada pada sel yg diarsir

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

State awal

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

State akhir



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

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

3. Algoritma

Jika syarat dari Reachable Goal sudah terpenuhi, maka masuk ke algoritma sebagai berikut, dengan nilai cost

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

$\hat{c}(i)$ = ongkos untuk simpul i

$\hat{f}(i)$ = ongkos mencapai simpul i dari akar

$\hat{g}(i)$ = ongkos mencapai simpul tujuan dari simpul i .

Tahap:

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' $\hat{c}(i)$ paling kecil. Jika terdapat beberapa simpul i yang memenuhi, pilih satu secara sembarang.
4. 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 $\hat{c}(j)$, dan masukkan semua anak-anak tersebut ke dalam Q.
7. Kembali ke langkah 2.

II. Source Program

Algorithm.py

```
import source
import time
import node

def copyNodeMatrixFrom(matrix):
    copy = [[]]
    for i in range(4):
        copy.append([])
        for j in range(4):
            copy[i].append(matrix[i][j])
    return copy

def calculateCostOf(node):
    totalCost = 0
    count = 1
    for i in range(4):
        for j in range(4):
            if node[i][j] != 16:
                if node[i][j] != count:
                    totalCost += 1
                count += 1
    return totalCost

def theSameWith(initialNode, customizedNode):
    for i in range(4):
        for j in range(4):
            if initialNode[i][j] != customizedNode[i][j]:
                return False
    return True

def theSameWithTheRestOf(visitedNodes, nodes):
    for i in visitedNodes:
        if theSameWith(i.mat, nodes):
            return True
    return False

# change current Node to the next Node where the it chose UP command
def up(Node):
    found = False
    for i in range(4):
        for j in range(4):
            if Node[i][j] == 16 and i != 0 and not found:
                Node[i][j], Node[i-1][j] = Node[i-1][j], Node[i][j]
```

```

        found = True
    return Node

# change current Node to the next Node where the it chose DOWN command
def down(Node):
    found = False
    for i in range(4):
        for j in range(4):
            if Node[i][j] == 16 and i != 3 and not found:
                Node[i][j], Node[i+1][j] = Node[i+1][j], Node[i][j]
                found = True
    return Node

# change current Node to the next Node where the it chose LEFT command
def left(Node):
    found = False
    for i in range(4):
        for j in range(4):
            if Node[i][j] == 16 and j != 0 and not found:
                Node[i][j-1], Node[i][j] = Node[i][j], Node[i][j-1]
                found = True
    return Node

# change current Node to the next Node where the it chose RIGHT command
def right(Node):
    found = False
    for i in range(4):
        for j in range(4):
            if Node[i][j] == 16 and j != 3 and not found:
                Node[i][j+1], Node[i][j] = Node[i][j], Node[i][j+1]
                found = True
    return Node

# to generate node from the current Node
# to choose node with the lowest cost
def generateNodeFrom(visitedNodes, parent):
    listOfNodes = []
    listOfCommand = ["right", "left", "down", "up"]
    for i in listOfCommand:

        # to have a copy of Node
        # because if we use Node, python will change the value of Node itself
        # (python use reference instead of copy value lol)
        copyOfNodeMatrix = copyNodeMatrixFrom(parent.mat)

        # just a bunch of code to generate node from the current Node
        if i == "up":
            upNodeMatrix = up(copyOfNodeMatrix)
            if not theSameWithTheRestOf(visitedNodes, upNodeMatrix):

```

```

        newNode = node.node(parent, upNodeMatrix, calculateCostOf(upNodeMatrix),
parent.depth+1, i)
        listOfNodes.append(newNode)

    if i == "down":
        downNodeMatrix = down(copyOfNodeMatrix)
        if not theSameWithTheRestOf(visitedNodes, downNodeMatrix):
            newNode = node.node(parent, downNodeMatrix, calculateCostOf(downNodeMatrix),
parent.depth+1, i)
            listOfNodes.append(newNode)

    if i == "left":
        leftNodeMatrix = left(copyOfNodeMatrix)
        if not theSameWithTheRestOf(visitedNodes, leftNodeMatrix):
            newNode = node.node(parent, leftNodeMatrix, calculateCostOf(leftNodeMatrix),
parent.depth+1, i)
            listOfNodes.append(newNode)

    if i == "right":
        rightNodeMatrix = right(copyOfNodeMatrix)
        if not theSameWithTheRestOf(visitedNodes, rightNodeMatrix):
            newNode = node.node(parent, rightNodeMatrix, calculateCostOf(rightNodeMatrix),
parent.depth+1, i)
            listOfNodes.append(newNode)

    return listOfNodes

def solver(parent):
    # IF THE FIRST PUZZLE IS ALREADY SOLVED LOL
    if calculateCostOf(parent.mat) == 0:
        source.printPuzzleInNode(parent)
        print("YOU PUZZLE HAS ALREADY BEEN SOLVED!")
        return [parent.mat]

    # add parent to visited nodes
    visitedNodes = [parent]

    # generate node from the current parent
    listOfNodes = generateNodeFrom(visitedNodes, parent)

    # push to prio queue
    pq = node.priorityQueue()
    for i in listOfNodes:
        pq.push(i)

    countVisitedNodes = 1
    while not pq.empty():
        nodes = pq.pop()
        visitedNodes.append(nodes)

```

```

countVisitedNodes+=1

print(f"===== {countVisitedNodes} =====")
print(f"Move {nodes.command} is chosen")
source.printPuzzleInNode(nodes)

if calculateCostOf(nodes.mat) == 0:
    print("YOU PUZZLE HAS BEEN SOLVED!")
    return visitedNodes
listOfNodes = generateNodeFrom(visitedNodes, nodes)
for i in listOfNodes:
    pq.push(i)

```

node.py

```

# Importing the heap functions from python
# library for Priority Queue
from heapq import heappush, heappop

class priorityQueue:

    # Constructor to initialize a
    # Priority Queue
    def __init__(self):
        self.heap = []

    # Inserts a new key 'k'
    def push(self, k):
        heappush(self.heap, k)

    # Method to remove minimum element
    # from Priority Queue
    def pop(self):
        return heappop(self.heap)

    # Method to know if the Queue is empty
    def empty(self):
        if not self.heap:
            return True
        else:
            return False

# Node structure
class node:

    def __init__(self, parent, mat, cost, depth, command):

        # Stores the parent node of the

```

```

# current node helps in tracing
# path when the answer is found
self.parent = parent

# Stores the matrix
self.mat = mat

# Stores the number of misplaced tiles
self.cost = cost

# Stores the number of depth so far
self.depth = depth

# Stores the command used for this node
self.command = command

# This method is defined so that the
# priority queue is formed based on
# the cost variable of the objects
def __lt__(self, nxt):
    if self.cost + self.depth == nxt.cost + nxt.depth:
        return self.cost < nxt.cost
    return self.cost + self.depth < nxt.cost + nxt.depth

```

source.py

```

import algorithms
import time
import node

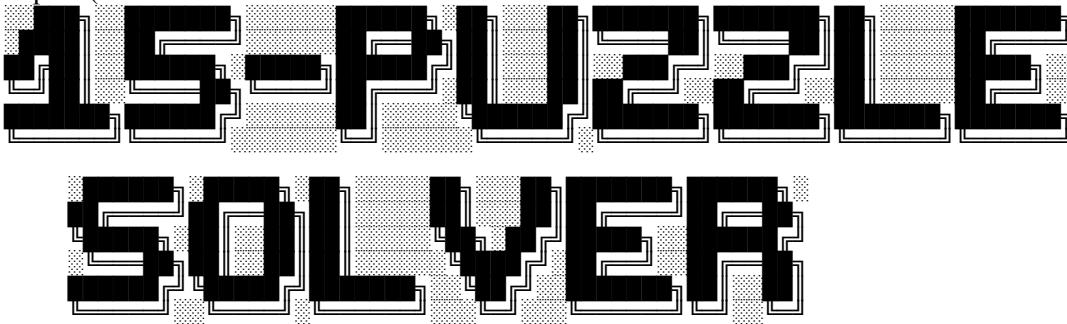
```

```

def printIntro():

```

```

    print("""

    """)

```

```

def isPuzzleSolvable(puzzle):
    totalSigma = 0
    x = 0
    # looping though the puzzle
    for i in range(4):

```



```

for j in range(4):
    # mengecek tempat sel kosong. Apakah sel kosong berada pada sel yang diarsir
    if (puzzle[i][j] == 16):
        if (i % 2 == 0 and j % 2 != 0):
            x = 1
        elif (i % 2 != 0 and j % 2 == 0):
            x = 1

```

```

# looping through current position until the end to check
# whether there is number less than current number in current position
total = 0

```

```

for k in range(i,4):
    if k == i:
        for l in range(j,4):
            if puzzle[k][l] < puzzle[i][j]:
                total += 1
    else:
        for l in range(4):
            if puzzle[i][j] > puzzle[k][l]:
                total += 1

```

```

print(f"Kurang[" , end="")
if (puzzle[i][j] < 10):
    print("0", end="")
print(puzzle[i][j], end="")
print(f"] = {total}")
totalSigma += total

```

```

print(f"Total sigma + x: {totalSigma} + {x} = {totalSigma+x} ")
return totalSigma + x

```

```

def printPuzzle(puzzle):
    for i in range(4):
        print("[ ", end="")
        for j in range(4):
            if (puzzle[i][j] == 16):
                print(" - ", end="")
            else:
                if (puzzle[i][j] < 10):
                    print(" ", end="")
                print(puzzle[i][j], end=" ")
        print("]")
    print()

```

```

def printPuzzleInNode(node):
    for i in range(4):
        print("[ ", end="")
        for j in range(4):
            if (node.mat[i][j] == 16):
                print(" - ", end="")

```

```

        else:
            if (node.mat[i][j] < 10):
                print(" ", end="")
                print(node.mat[i][j], end=" ")
            print("]")
        print()

def readFromFile(filename):
    with open(filename, 'r') as f:
        puzzle = [[int(num) for num in line.split(' ')] for line in f]
    return puzzle

def mainProgram():
    # print intro hehe :D
    printIntro()

    initialPuzzle = []

    print("Please choose how you want to input your puzzle:")
    choice = int(input("\n1. Read from file\n2. Input manually\n\nchoice: "))
    if (choice == 1):
        # input file to determine puzzle
        filename = input("Please input filename for puzzle: ")
        initialPuzzle = readFromFile("../test/" + filename)
    elif (choice == 2):
        initialPuzzle = [[0,0,0,0],[0,0,0,0],[0,0,0,0],[0,0,0,0]]
        print("Input the puzzle:")
        for i in range(4):
            for j in range(4):
                initialPuzzle[i][j] = int(input(f"puzzle[{i}][{j}] = "))
    else:
        print("Invalid input")
        return

    print("This is your initial puzzle: ")
    printPuzzle(initialPuzzle)

    # check whether the puzzle is solveable or not
    sigma = isPuzzleSolvable(initialPuzzle)

    # verdict of whether the puzzle is solveable or not
    print()
    print("Verdict: ")

    # if sigma is odd, then puzzle cannot be solved
    if sigma % 2 != 0:
        print("Your puzzle cannot be solved!")
        return
    else:

```

```
start_time = time.time()
print("Your puzzle can be solved!")
print()
# make initial Node
initialNode = node.node(initialPuzzle, initialPuzzle, sigma, 0, "-")
visitedNodes = algorithms.solver(initialNode)
print()
print(f"The number of nodes visited: {len(visitedNodes)}")
print()
print("Total time of program execution:", time.time() - start_time)
```

main.py

```
import source

# run this file to run the code :D
source.mainProgram()
```

III. Screenshot Input dan Output

Nomor	Input	Output
1	2step.txt 1 2 3 4 5 6 7 8 9 10 11 12 13 16 14 15	Kurang[01] = 0 Kurang[02] = 0 Kurang[03] = 0 Kurang[04] = 0 Kurang[05] = 0 Kurang[06] = 0 Kurang[07] = 0 Kurang[08] = 0 Kurang[09] = 0 Kurang[10] = 0 Kurang[11] = 0 Kurang[12] = 0 Kurang[13] = 0 Kurang[16] = 2 Kurang[14] = 0 Kurang[15] = 0 Total sigma + x: $2 + 0 = 2$ Verdict: Your puzzle can be solved! ===== 1 ===== Move right is chosen [1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 - 15] ===== 2 ===== Move right is chosen [1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 15 -] YOU PUZZLE HAS BEEN SOLVED! The number of nodes visited: 3 Total time of program execution: 0.0004258155822753906

2	5step.txt 1 2 3 16 5 6 7 4 9 10 12 8 13 14 11 15	Kurang[01] = 0 Kurang[02] = 0 Kurang[03] = 0 Kurang[16] = 12 Kurang[05] = 1 Kurang[06] = 1 Kurang[07] = 1 Kurang[04] = 0 Kurang[09] = 1 Kurang[10] = 1 Kurang[12] = 2 Kurang[08] = 0 Kurang[13] = 1 Kurang[14] = 1 Kurang[11] = 0 Kurang[15] = 0 Total sigma + x: 21 + 1 = 22 Verdict: Your puzzle can be solved! ===== 1 ===== Move down is chosen [1 2 3 4] [5 6 7 -] [9 10 12 8] [13 14 11 15] ===== 2 ===== Move down is chosen [1 2 3 4] [5 6 7 8] [9 10 12 -] [13 14 11 15] ===== 3 ===== Move left is chosen [1 2 3 4] [5 6 7 8] [9 10 - 12] [13 14 11 15] ===== 4 ===== Move down is chosen [1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 - 15] ===== 5 ===== Move right is chosen [1 2 3 4] [5 6 7 8] [9 10 11 12] [13 14 15 -] YOU PUZZLE HAS BEEN SOLVED! The number of nodes visited: 6 Total time of program execution: 0.001621484756469726
---	--	--

3	10Step.txt 5 1 3 4 9 2 7 8 16 6 15 11 13 10 14 12	Kurang[05] = 4 Kurang[01] = 0 Kurang[03] = 1 Kurang[04] = 1 Kurang[09] = 4 Kurang[02] = 0 Kurang[07] = 1 Kurang[08] = 1 Kurang[16] = 7 Kurang[06] = 0 Kurang[15] = 5 Kurang[11] = 1 Kurang[13] = 2 Kurang[10] = 0 Kurang[14] = 1 Kurang[12] = 0 Total sigma + x: 28 + 0 = 28 Verdict: Your puzzle can be solved! ===== 1 ===== Move up is chosen [5 1 3 4] [- 2 7 8] [9 6 15 11] [13 10 14 12] ===== 2 ===== Move up is chosen [- 1 3 4] [5 2 7 8] [9 6 15 11] [13 10 14 12] ===== 3 ===== Move right is chosen [1 - 3 4] [5 2 7 8] [9 6 15 11] [13 10 14 12] ===== 4 ===== Move down is chosen [1 2 3 4] [5 - 7 8] [9 6 15 11] [13 10 14 12] ===== 5 =====
---	---	--

Move down is chosen

[1 2 3 4]

[5 6 7 8]

[9 - 15 11]

[13 10 14 12]

===== 6 =====

Move down is chosen

[1 2 3 4]

[5 6 7 8]

[9 10 15 11]

[13 - 14 12]

===== 7 =====

Move right is chosen

[1 2 3 4]

[5 6 7 8]

[9 10 15 11]

[13 14 - 12]

===== 8 =====

Move up is chosen

[1 2 3 4]

[5 6 7 8]

[9 10 - 11]

[13 14 15 12]

===== 9 =====

Move right is chosen

[1 2 3 4]

[5 6 7 8]

[9 10 11 -]

[13 14 15 12]

===== 10 =====

Move down is chosen

[1 2 3 4]

[5 6 7 8]

[9 10 11 12]

[13 14 15 -]

YOU PUZZLE HAS BEEN SOLVED!

The number of nodes visited: 11

Total time of program execution: 0.0018329620361328125

4	failed1.txt 1 3 4 15 2 16 5 12 7 6 11 14 8 9 10 13	Kurang[01] = 0 Kurang[03] = 1 Kurang[04] = 1 Kurang[15] = 11 Kurang[02] = 0 Kurang[16] = 10 Kurang[05] = 0 Kurang[12] = 6 Kurang[07] = 1 Kurang[06] = 0 Kurang[11] = 3 Kurang[14] = 4 Kurang[08] = 0 Kurang[09] = 0 Kurang[10] = 0 Kurang[13] = 0 Total sigma + x: $37 + 0 = 37$ Verdict: Your puzzle cannot be solved!
5	failed2.txt 10 3 4 7 8 1 14 9 11 15 16 2 6 12 13 5	Kurang[10] = 9 Kurang[03] = 2 Kurang[04] = 2 Kurang[07] = 4 Kurang[08] = 4 Kurang[01] = 0 Kurang[14] = 7 Kurang[09] = 3 Kurang[11] = 3 Kurang[15] = 5 Kurang[16] = 5 Kurang[02] = 0 Kurang[06] = 1 Kurang[12] = 1 Kurang[13] = 1 Kurang[05] = 0 Total sigma + x: $47 + 0 = 47$ Verdict: Your puzzle cannot be solved!

IV. Lampiran

1. Alamat Drive

<https://github.com/samuelswandi/15Puzzle-solver>

2. Check List

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