

**LAPORAN TUGAS KECIL III**  
**IF2211 STRATEGI ALGORITMA**



Laporan ini dibuat untuk memenuhi tugas  
Mata Kuliah IF 2211 Strategi Algoritma

**Disusun Oleh :**

Vincent Ho (13520093)

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**SEMESTER I TAHUN 2021/2022**

# BAB I

## ALGORITMA BRANCH AND BOUND

Algoritma Branch and Bound yang saya terapkan dalam menyelesaikan persoalan 15-Puzzle memiliki langkah – langkah sebagai berikut.

Langkah pertama yang harus dilakukan adalah membentuk node dari state awal puzzle. Setelah itu dibangkitkan node yang mungkin dan dimasukkan kedalam queue simpul hidup yang memiliki struktur data priority queue dimana elemennya diprioritaskan berdasarkan cost yang terkecil. Cost dari sebuah node dihitung berdasarkan jumlah ubin tidak kosong yang tidak berada pada tempat yang sesuai dengan susunan akhir (goal state). Susunan akhir yang diinginkan sesuai dengan gambar berikut.

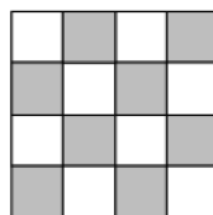
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Apabila nilai cost sudah sama dengan 0 (seluruh ubin sudah berada pada susunan akhir yang diinginkan) maka pencarian berhenti. Jika tidak maka akan diambil elemen paling depan dari queue simpul hidup (elemen yang memiliki cost paling kecil) untuk dibangkitkan lagi anak-anaknya. Proses ini akan dilanjutkan secara terus menerus hingga mendapatkan node yang memiliki cost sama dengan 0 (susunan sesuai dengan goal state).

Optimasi yang dilakukan pada algoritma branch and bound yang saya terapkan antara lain: memeriksa apakah pergeseran ubin yang akan dilakukan tidak menyebabkan puzzle kembali ke state sebelumnya. Misalnya setelah menggeser ubin kosong ke kiri, maka node yang dibangkitkan selanjutnya tidak boleh menggeser ubin kosong ke kanan karena hal ini akan mengakibatkan puzzle kembali ke state sebelumnya. Kemudian saya juga mencatat seluruh state puzzle yang pernah dibangkitkan. Hal ini bertujuan agar tidak terjadi repetisi pembangkitan state puzzle yang sama. Sehingga sebelum membangkitkan sebuah node baru, akan dilakukan pengecekan terlebih dahulu apakah state puzzle tersebut pernah dibangkitkan sebelumnya.

Sebelum dilakukan pencarian solusi menggunakan algoritma branch and bound, akan dilakukan pengecekan terlebih dahulu apakah puzzle dapat diselesaikan atau tidak. Pengecekan ini dilakukan dengan rumus sebagai berikut.

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



Kurang(i) adalah banyaknya ubin bernomor j sedemikian sehingga  $j < i$  dan  $\text{posisi}(j) > \text{posisi}(i)$ . Posisi(i) artinya posisi dari ubin bernomor i pada susunan yang diperiksa. X memiliki nilai 1 apabila posisi awal sel kosong berada pada sel yang diarsir berikut. Jika hasil dari perhitungan diatas bernilai genap artinya puzzle memiliki solusi dan sebaliknya bila bernilai ganjil artinya puzzle tidak mungkin diselesaikan.

Catatan:

Program 15 Puzzle Solver yang saya buat dapat menerima input berupa file txt. Untuk menandakan lokasi ubin kosong cukup dibiarkan kosong pada posisi yang ditentukan.

Sebagai contoh:

Keterangan: Underscore ( \_ ) dianggap sebagai spasi dalam penulisan laporan untuk memudahkan penjelasan, namun pada file txt asli digunakan pemisah berupa spasi

1\_2\_3\_

5\_6\_7\_8

9\_10\_11\_12

13\_14\_15\_4

Jika input yang dimasukkan seperti ini maka posisi ubin kosong terdapat pada pojok kanan atas.

1\_\_3\_4

5\_6\_7\_8

9\_10\_11\_12

13\_14\_15\_2

Jika input yang dimasukkan seperti ini maka posisi ubin kosong terletak diantara angka 1 dan 3

1\_5\_3\_4

\_6\_7\_8

9\_10\_11\_12

13\_14\_15\_2

Jika input yang dimasukkan seperti ini maka posisi ubin kosong terletak disebelah kiri angka 6

## BAB II

### SOURCE CODE PROGRAM

#### Puzzle15.py

```
import copy
import bisect
import random
import time

solutionMatrix = [
    ["1", "2", "3", "4"],
    ["5", "6", "7", "8"],
    ["9", "10", "11", "12"],
    ["13", "14", "15", ""],
]

class TreeNode:
    liveNode = []
    visitedNode = set()
    solution = []
    NodeCreated = []

    def __init__(self, parent, matrix, direction, cost, emptyRow,
emptyCol):
        self.parent = parent
        self.matrix = matrix
        self.direction = direction
        self.cost = cost
        self.emptyRow = emptyRow
        self.emptyCol = emptyCol

        stringMatrix = toString(matrix)
        if stringMatrix not in self.visitedNode:
            self.visitedNode.add(stringMatrix)
            bisect.insort_right(self.liveNode, self, key=lambda x:
x.cost)

            self.liveNode.append(self)
            if self.NodeCreated == []:
                self.NodeCreated.append(0)
            else:
                self.NodeCreated[0] += 1

    def createNewNode(self):
        self.liveNode.remove(self)
        if self.direction != "up" and self.emptyRow != 3:
```

```

        newMatrix = move(self.matrix, "down", self.emptyRow,
self.emptyCol)
        newCost = cost(newMatrix)
        self.newNode = TreeNode(
            self, newMatrix, "down", newCost, self.emptyRow + 1,
self.emptyCol
        )
        if self.direction != "down" and self.emptyRow != 0:
            newMatrix = move(self.matrix, "up", self.emptyRow,
self.emptyCol)
            newCost = cost(newMatrix)
            self.newNode = TreeNode(
                self, newMatrix, "up", newCost, self.emptyRow - 1,
self.emptyCol
            )
        if self.direction != "left" and self.emptyCol != 3:
            newMatrix = move(self.matrix, "right", self.emptyRow,
self.emptyCol)
            newCost = cost(newMatrix)
            self.newNode = TreeNode(
                self, newMatrix, "right", newCost, self.emptyRow,
self.emptyCol + 1
            )
        if self.direction != "right" and self.emptyCol != 0:
            newMatrix = move(self.matrix, "left", self.emptyRow,
self.emptyCol)
            newCost = cost(newMatrix)
            self.newNode = TreeNode(
                self, newMatrix, "left", newCost, self.emptyRow,
self.emptyCol - 1
            )

    def solve(self):
        while cost(self.liveNode[0].matrix) > 0:
            self.liveNode[0].createNewNode()

    def getSolution(self):
        current = self
        while current != None:
            current.solution.append(current)
            current = current.parent

    def printSolution(self):
        for i in range(len(self.solution)):
            print(self.solution[i].direction)
            for j in range(4):
                print(self.solution[i].matrix[j])

```

```

# read matrix from txt file
def readMatrix(filename):
    with open(filename, "r") as f:
        lines = f.read().splitlines()
        matrix = []
        for line in lines:
            matrix.append(line.split(" "))
    return matrix

# convert matrix to string format
def toString(matrix):
    str = ""
    for i in range(4):
        for j in range(4):
            if matrix[i][j] == " ":
                str += " "
            else:
                str += matrix[i][j]
    return str

# get matrix index
def getIndex(element, matrix):
    for i in range(4):
        for j in range(4):
            if matrix[i][j] == element:
                return (i, j)
    return (-1, -1)

# compare position of two element, return true if i > j
def comparePosition(i, j, matrix):
    rowI, colI = getIndex(i, matrix)
    rowJ, colJ = getIndex(j, matrix)
    if int(rowI) > int(rowJ):
        return True
    elif int(rowI) == int(rowJ) and int(colI) > int(colJ):
        return True
    else:
        return False

# return value of Kurang(i)
def kurang(i, matrix):
    count = 0
    if i == "16":
        rowI, colI = getIndex("", matrix)
    else:

```

```

        rowI, colI = getIndex(i, matrix)

    for row in range(rowI, 4):
        for col in range((colI if row == rowI else 0), 4):
            if matrix[row][col] == "":
                if (int(16) < int(i)) and comparePosition("", i,
matrix):
                    count += 1
            else:
                if int(matrix[row][col]) < int(i) and
comparePosition(
                    matrix[row][col], i, matrix
                ):
                    count += 1
    return count

# return true if puzzle is solvable
def isSolvable(matrix):
    total = 0
    for i in range(1, 17):
        total += kurang(str(i), matrix)

    rowEmpty, colEmpty = getIndex("", matrix)
    if (rowEmpty + colEmpty) % 2 == 1:
        total += 1
    return total % 2 == 0

# return sum of kurang(i) from 1 to 16
def sumKurang(matrix):
    total = 0
    for i in range(1, 17):
        total += kurang(str(i), matrix)

    rowEmpty, colEmpty = getIndex("", matrix)
    if (rowEmpty + colEmpty) % 2 == 1:
        total += 1
    return total

# return the cost function of the puzzle
def cost(matrix):
    cost = 0
    for i in range(4):
        for j in range(4):
            if matrix[i][j] != solutionMatrix[i][j]:
                cost += 1
    return cost

```

```

# move the empty space of the puzzle
def move(matrix, direction, emptyRow, emptyCol):
    newMatrix = copy.deepcopy(matrix)
    if direction == "up":
        newMatrix[emptyRow][emptyCol] = matrix[emptyRow -
1][emptyCol]
        newMatrix[emptyRow - 1][emptyCol] = ""
    elif direction == "down":
        newMatrix[emptyRow][emptyCol] = matrix[emptyRow +
1][emptyCol]
        newMatrix[emptyRow + 1][emptyCol] = ""
    elif direction == "left":
        newMatrix[emptyRow][emptyCol] = matrix[emptyRow][emptyCol -
1]
        newMatrix[emptyRow][emptyCol - 1] = ""
    elif direction == "right":
        newMatrix[emptyRow][emptyCol] = matrix[emptyRow][emptyCol +
1]
        newMatrix[emptyRow][emptyCol + 1] = ""
    return newMatrix

# Create random matrix
def randomizeMatrix():
    elements = [
        "",
        "1",
        "2",
        "3",
        "4",
        "5",
        "6",
        "7",
        "8",
        "9",
        "10",
        "11",
        "12",
        "13",
        "14",
        "15",
    ]
    random.shuffle(elements)
    matrix = []
    for i in range(4):
        matrix.append([])
        for j in range(4):

```



```
        matrix[i].append(elements[i * 4 + j])
    return matrix

# Uncomment code below to run in console

# filename = input("Enter file name: ")
# matrix = readMatrix(filename)
# if isSolvable(matrix):
#     emptyRow, emptyCol = getIndex("", matrix)
#     root = TreeNode(None, matrix, "Start state", cost(matrix),
# emptyRow, emptyCol)
#     startTime = time.time()
#     root.solve()
#     endTime = time.time()
#     runtime = endTime - startTime
#     root.liveNode[0].getSolution()
#     root.solution.reverse()
#     root.liveNode[0].printSolution()
#     print("Runtime: ", runtime)
#     print("Number of steps: " + str(len(root.solution) - 1))
#     print("Node Created: " + str(root.NodeCreated[0]))
# else:
#     print("Puzzle is not solvable")
```

## gui.py

```
import time
import Puzzle15
from tkinter import *
import threading

# get input from textBox
def getInput():
    global inputValue
    global puzzleMatrix
    inputValue = textBox.get("1.0", "end-1c")
    puzzleMatrix = Puzzle15.readMatrix(inputValue)
    setPuzzle(puzzleCell, puzzleMatrix)
    if Puzzle15.isSolvable(puzzleMatrix):
        Label(root, text="--Solvable--", fg="green").grid(row=1,
column=0)
    else:
        Label(root, text="Not Solvable", fg="red").grid(row=1,
column=0)
    showKurang()

# generate random puzzle Matrix
def randomize():
    global puzzleMatrix
    puzzleMatrix = Puzzle15.randomizeMatrix()
    setPuzzle(puzzleCell, puzzleMatrix)
    if Puzzle15.isSolvable(puzzleMatrix):
        Label(root, text="--Solvable--", fg="green").grid(row=1,
column=0)
    else:
        Label(root, text="Not Solvable", fg="red").grid(row=1,
column=0)
    showKurang()

# set Puzzle Frame
def setPuzzle(puzzleCell, Matrix):
    for i in range(4):
        for j in range(4):
            if Matrix[i][j] == "":
                puzzleCell[i][j].config(text="", bg="white")
            else:
                puzzleCell[i][j].config(text=Matrix[i][j],
bg="lightblue")

def showKurang():
```

```

for i in range(1, 17):
    Label(
        rightFrame,
        text="Kurang("
        + str(i)
        + ") = "
        + str(Puzzle15.kurang(str(i), puzzleMatrix)),
    ).grid(row=i, column=0)

    Label(
        rightFrame,
        text="SUM(Kurang(i)) + X = " +
str(Puzzle15.sumKurang(puzzleMatrix)),
    ).grid(row=17, column=0)

# solve puzzle and show solution animation
def showSolution():
    global puzzleMatrix
    emptyRow, emptyCol = Puzzle15.getIndex("", puzzleMatrix)
    tree = Puzzle15.TreeNode(
        None,
        puzzleMatrix,
        "start state",
        Puzzle15.cost(puzzleMatrix),
        emptyRow,
        emptyCol,
    )

    if Puzzle15.isSolvable(puzzleMatrix):
        Label(root, text="Loading solution", fg="blue").grid(row=3,
column=0)
        startTime = time.time()
        tree.solve()
        endTime = time.time()
        runTime = endTime - startTime

        tree.liveNode[0].getSolution()
        tree.liveNode[0].solution.reverse()

        Label(root, text="Run time: " + ("%5f" %
runTime)).grid(row=3, column=0)
        steps = len(tree.solution) - 1
        Label(root, text="Number of steps: " + str("%3.f" %
steps)).grid(
            row=4, column=0
        )
        Label(root, text="---Node Created: " +
str(tree.NodeCreated[0]) + "---").grid(

```

```

        row=5, column=0
    )
    Label(
        root, text="List of steps is printed in the console!",
        bg="lightgreen"
    ).grid(row=6, column=0)

    print("=====")
    print("=  Solution  =")
    print("=====")
    for i in range(len(tree.solution)):
        time.sleep(0.05)
        setPuzzle(puzzleCell, tree.solution[i].matrix)
        print(tree.solution[i].direction)
        for j in range(4):
            print(tree.solution[i].matrix[j])
        # print(tree.solution[i].matrix)

    del tree.solution[:]
    del tree.liveNode[:]
    tree.visitedNode.clear()
    del tree.NodeCreated[:]

# return solution step by step
def threadingShowSolution():
    t = threading.Thread(target=showSolution)
    t.start()

root = Tk()
root.title("15 Puzzle Solver")
puzzleFrame = Frame(root)
puzzleFrame.grid(row=0, column=0, padx=10, pady=10)
rightFrame = Frame(root)
rightFrame.grid(row=0, column=1, padx=10, pady=10)

# make a 2d array containing puzzle border
puzzleCell = []
for i in range(4):
    puzzleCell.append([])
    for j in range(4):
        puzzleCell[i].append(
            Label(puzzleFrame, borderwidth=4, width=8, height=4,
relief="raised")
        )
        puzzleCell[i][j].grid(row=i, column=j)

```

```
Label(puzzleFrame, text="Input filename:").grid(row=0, column=4)
textBox = Text(puzzleFrame, height=1, width=20)
textBox.grid(row=1, column=4, padx=10, pady=10)

submitButton = Button(
    puzzleFrame,
    text="Submit",
    command=lambda: getInput(),
    padx=5,
    pady=5,
    bg="lightyellow",
    borderwidth=4,
)
submitButton.grid(row=1, column=5, padx=10, pady=10)

randomizeButton = Button(
    puzzleFrame,
    text="Randomize",
    command=lambda: randomize(),
    padx=10,
    pady=10,
    bg="lightblue",
    borderwidth=4,
)
randomizeButton.grid(row=2, column=4, padx=10, pady=10)

solveButton = Button(
    root,
    text="Solve",
    command=threadingShowSolution,
    padx=10,
    pady=10,
    bg="yellow",
    borderwidth=4,
)
solveButton.grid(row=2, column=0, padx=5, pady=5)

root.mainloop()
```

## BAB III

### EKSEKUSI PROGRAM

#### 1. Test Case 1 (t1.txt)

```
t1.txt
1  1 2 3 4
2  5 6 8
3  9 10 7 12
4  13 14 11 15
```

15 Puzzle Solver

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

Input filename:

--Solvable--

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) = 1  
Kurang(13) = 1  
Kurang(14) = 1  
Kurang(15) = 0  
Kurang(16) = 10  
SUM(Kurang(i)) + X = 16

15 Puzzle Solver

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

Input filename:

--Solvable--

Run time: 0.00000  
Number of steps: 4  
---Node Created: 12---

List of steps is printed in the console!

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) = 1  
Kurang(13) = 1  
Kurang(14) = 1  
Kurang(15) = 0  
Kurang(16) = 10  
SUM(Kurang(i)) + X = 16

## 2. Test Case 1 (t2.txt)

t2.txt

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

15 Puzzle Solver

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

Input filename:

t2.txt

Submit

Randomize

--Solvable--

Solve

Kurang(1) = 0  
Kurang(2) = 0  
Kurang(3) = 2  
Kurang(4) = 3  
Kurang(5) = 0  
Kurang(6) = 2  
Kurang(7) = 2  
Kurang(8) = 5  
Kurang(9) = 0  
Kurang(10) = 8  
Kurang(11) = 8  
Kurang(12) = 3  
Kurang(13) = 9  
Kurang(14) = 11  
Kurang(15) = 6  
Kurang(16) = 6  
SUM(Kurang(i)) + X = 66

15 Puzzle Solver

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

Input filename:

t2.txt

Submit

Randomize

--Solvable--


Solve

Run time: 1.15941  
Number of steps: 327  
---Node Created: 32557---

List of steps is printed in the console!

Kurang(1) = 0  
Kurang(2) = 0  
Kurang(3) = 2  
Kurang(4) = 3  
Kurang(5) = 0  
Kurang(6) = 2  
Kurang(7) = 2  
Kurang(8) = 5  
Kurang(9) = 0  
Kurang(10) = 8  
Kurang(11) = 8  
Kurang(12) = 3  
Kurang(13) = 9  
Kurang(14) = 11  
Kurang(15) = 6  
Kurang(16) = 6  
SUM(Kurang(i)) + X = 66

### 3. Test Case 3 (t3.txt)

 t3.txt

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

15 Puzzle Solver


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

Input filename:

Not Solvable

Kurang(1) = 0  
Kurang(2) = 1  
Kurang(3) = 1  
Kurang(4) = 2  
Kurang(5) = 3  
Kurang(6) = 2  
Kurang(7) = 2  
Kurang(8) = 1  
Kurang(9) = 4  
Kurang(10) = 7  
Kurang(11) = 9  
Kurang(12) = 6  
Kurang(13) = 0  
Kurang(14) = 6  
Kurang(15) = 5  
Kurang(16) = 3  
SUM(Kurang(i)) + X = 53

### 4. Test Case 4 (t4.txt)

 t4.txt

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

15 Puzzle Solver

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

Input filename:

--Solvable--

Kurang(1) = 0  
Kurang(2) = 0  
Kurang(3) = 0  
Kurang(4) = 3  
Kurang(5) = 4  
Kurang(6) = 2  
Kurang(7) = 0  
Kurang(8) = 0  
Kurang(9) = 3  
Kurang(10) = 2  
Kurang(11) = 0  
Kurang(12) = 9  
Kurang(13) = 9  
Kurang(14) = 2  
Kurang(15) = 5  
Kurang(16) = 4  
SUM(Kurang(i)) + X = 44



15 Puzzle Solver

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

Input filename:

--Solvable--

Run time: 0.20132  
Number of steps: 125  
---Node Created: 7958---

List of steps is printed in the console!

Kurang(1) = 0  
Kurang(2) = 0  
Kurang(3) = 0  
Kurang(4) = 3  
Kurang(5) = 4  
Kurang(6) = 2  
Kurang(7) = 0  
Kurang(8) = 0  
Kurang(9) = 3  
Kurang(10) = 2  
Kurang(11) = 0  
Kurang(12) = 9  
Kurang(13) = 9  
Kurang(14) = 2  
Kurang(15) = 5  
Kurang(16) = 4  
SUM(Kurang(i)) + X = 44

## 5. Test Case 5 (t5.txt)

t5.txt

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

15 Puzzle Solver

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

Input filename:

Not Solvable

Kurang(1) = 0  
Kurang(2) = 1  
Kurang(3) = 0  
Kurang(4) = 1  
Kurang(5) = 4  
Kurang(6) = 0  
Kurang(7) = 3  
Kurang(8) = 1  
Kurang(9) = 8  
Kurang(10) = 0  
Kurang(11) = 10  
Kurang(12) = 2  
Kurang(13) = 6  
Kurang(14) = 0  
Kurang(15) = 0  
Kurang(16) = 15  
SUM(Kurang(i)) + X = 51

**BAB IV**  
**PENUTUP**

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 data uji	✓	
5. Bonus dibuat	✓	

Repo Github: [https://github.com/vincen-tho/Tucil3\\_1320093](https://github.com/vincen-tho/Tucil3_1320093)