

Tugas Kecil 3 IF2211 Strategi Algoritma

Penyelesaian Persoalan 15-Puzzle dengan Algoritma Branch and Bound



Disusun Oleh :

Ilham Pratama

13520041

Kelas K-02

**PROGRAM STUDI SARJANA TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

INSTITUT TEKNOLOGI BANDUNG

2022

Daftar Isi

Algoritma Branch and Bound	3
Source Code	4
Input dan Output.....	10
CheckList.....	17
Lokasi Program dan Referensi.....	17

Algoritma Branch and Bound

Algoritma Branch and Bound merupakan salah satu strategi algoritma yang digunakan untuk menyelesaikan masalah optimasi. Algoritma ini menggunakan pohon ruang status untuk melakukan pencarian solusi terbaik. Perjalanan dalam pencarian status solusi menggunakan usaha yang paling optimal. Salah satu penerapan algoritma ini adalah dalam menyelesaikan persoalan 15-Puzzle.

15-Puzzle adalah salah satu permainan dengan menggunakan papan yang berisikan angka 1 sampai 15 dalam 16 bagian ubin. Terdapat satu ubin kosong yang dapat digerakkan ke atas, bawah, kiri, dan kanan untuk menggeser ubin lainnya. Tujuan yang dicapai dari permainan ini adalah menyusun angka 1 sampai 15 terurut dari atas ke bawah dengan cara menggeser ubin kosong.

Strategi yang digunakan pada penyelesaian 15 puzzle ini adalah

1. Dicek apakah susunan puzzle sudah benar semua atau belum. Jika belum akan dilakukan pengecekan dengan cara menghitung inversi dari susunan awal puzzle.
2. 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) yang dimaksud adalah inversi susunan puzzle, yakni $A[i] > A[j]$ tetapi $i < j$, sedangkan X adalah letak ubin kosong pada susunan awal. Maka dari teorema ini dapat dicari apakah puzzle dapat diselesaikan atau tidak.

3. Selanjutnya akan dicari *least cost search* 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.
4. Selanjutnya program akan menampilkan pergerakan dari puzzle hingga puzzle diselesaikan

Source Code Program

Source Code FileHandler.py

```
from Puzzle import Puzzle

class Filehandler:
    # konstruktor dari kelas FileHandler #
    def __init__(self):
        pass

    # membaca file dan mengubahnya kedalam bentuk matrik #
    def bacaFile(self, namafile):
        file = open(namafile, "r")
        temp = file.readlines()
        matrik = [] # inisialisasi matrik
        for element in temp:
            elmt = element.strip("\n").split(" ")
            matrik.append(elmt)

        for i in range(4):
            for j in range(4):
                # matrik yang kosong, dimisalkan dengan X, lalu akan
                # diubah menjadi 16 agar mempermudah perhitungan #
                if(matrik[i][j] == 'X'):
                    matrik[i][j] = 16
                else: # mengganti tipe element matrik ke dalam
                    # bentuk integer #
                    matrik[i][j] = int(matrik[i][j])

        puzzle = Puzzle()
        puzzle.matrik = matrik
        return puzzle
```

Source Code Puzzle.py

```
# berisi tentang class puzzle #
import copy
from turtle import down

class Puzzle:
    # Konstruktor untuk kelas Puzzle #
    def __init__(self) :
        self.matrik = [[-999 for i in range(4)] for i in range(4)]
        self.inversion = []
        self.container = {}
        self.queue = []
        self.depth = 0
        self.path = ""

    # mengubah matrik menjadi array
    def convertToArray(self, matrik) :
        Array = [-999 for i in range(16)]
```

```

        x = 0
        for i in range(4) :
            for j in range(4) :
                Array[x] = matrik[i][j]
                x += 1
        return Array

#mengubah array menjadi matrik
def convertToMatrik(self, matrik) :
    matrix2D = [[-999 for i in range(4)] for i in range(4)]
    x = 0
    for i in range(4) :
        for j in range(4) :
            matrix2D[i][j] = matrik[x]
            x += 1
    return matrix2D

# menghitung jumlah inverse dari matrik
# secara umum yang di cari adalah jumlah matrik[i] > matrik[j]
dan i < j
def countInversion(self) :
    count = 0
    matrixInv = self.convertToArray(self.matrik)
    for i in range(16) :
        inverseCount = 0
        for j in range(i+1, 16) :
            if (matrixInv[i] > matrixInv[j]) :
                inverseCount += 1
            count += 1
        self.inversion.append([matrixInv[i], inverseCount])
    inversion = self.inversion
    inversion.sort(key = lambda inversion: inversion[0])
    return count

# menemukan indeks baris tempat 16 berada
def indeksBlankSpaceRow(self) :
    found = False
    i = 0
    while not found :
        j = 0
        while (j < 4 and not found) :
            if (self.matrik[i][j] == 16) :
                found = True
            j += 1
        i += 1
    return i

# menemukan indeks kolom tempat 16 berada
def indeksBlankSpaceColumn(self) :
    found = False
    i = 0
    while not found :
        j = 0
        while (j < 4 and not found) :
            if (self.matrik[i][j] == 16) :
                found = True
            j += 1
        i += 1

```

```

        return j

    # menentukan apakah tempat 16 berada pada indeks ganjil
    atau genap
    def findBlankSpace(self) :
        i = self.indeksBlankSpaceRow()
        j = self.indeksBlankSpaceColumn()
        if ((i+j) % 2 == 0):
            return 0
        else:
            return 1

    # menentukan apakah matrik bisa diselesaikan atau tidak
    def isSolveable(self) :
        if ((self.countInversion() + self.findBlankSpace()) % 2 == 0)
:
            return True
        else:
            return False

    # mengecek apakah matrik sudah sampai tujuan atau belum
    # tujuannya adalah semua posisi elemen matrik berada pada tempat
    yang seharusnya
    def isSolution(self) :
        if (self.countPosition(self.matrik) != 16) :
            return False
        return True

    # menghitung jumlah elemen matrik yang letaknya sesuai dengan
    tempatnya
    def countPosition(self, matrik) :
        count = 0;
        ctr = 1;
        for i in range(4) :
            for j in range(4) :
                if (matrik[i][j] == ctr) :
                    count += 1
                ctr += 1
        return count

    # menegecek apakah pergerakan valid atau tidak valid
    def isMoveValid(self, matrik) :
        if (matrik[1] == 'up') :
            if (self.indeksBlankSpaceRow() == 1) :
                return False
        elif (matrik[1] == 'down') :
            if (self.indeksBlankSpaceRow() == 4) :
                return False
        elif (matrik[1] == 'left') :
            if (self.indeksBlankSpaceColumn() == 1) :
                return False
        elif (matrik[1] == 'right') :
            if (self.indeksBlankSpaceColumn() == 4) :
                return False
        return True

    # prosedur untuk memindahkan blank space / 16
    def move(self, matrik) :

```

```

i = self.indeksBlankSpaceRow() - 1
j = self.indeksBlankSpaceColumn() - 1
if (self.isMoveValid(matrik)) :
    if (matrik[1] == "up") :
        matrik[0][i][j] = matrik[0][i-1][j]
        matrik[0][i-1][j] = 16
    elif (matrik[1] == "down") :
        matrik[0][i][j] = matrik[0][i+1][j]
        matrik[0][i+1][j] = 16
    elif (matrik[1] == "left") :
        matrik[0][i][j] = matrik[0][i][j-1]
        matrik[0][i][j-1] = 16
    elif (matrik[1] == "right") :
        matrik[0][i][j] = matrik[0][i][j+1]
        matrik[0][i][j+1] = 16
return matrik

# prosedur untuk menampilkan arah pergerakan
def pathEvaluation(self) :
    currPath = self.path.pop(0)
    if (currPath == 'w') :
        self.move([self.matrik, 'up'])
        print("Up")
    elif (currPath == 's') :
        self.move([self.matrik, 'down'])
        print("Down")
    elif (currPath == 'a') :
        self.move([self.matrik, 'left'])
        print("Left")
    elif (currPath == 'd') :
        self.move([self.matrik, 'right'])
        print("Right")
    else :
        pass

# penyelesaian untuk matrik
def solve(self) :
    posUp = 999
    posDown = 999
    posLeft = 999
    posRight = 999

    up = [copy.deepcopy(self.matrik), 'up']
    down = [copy.deepcopy(self.matrik), 'down']
    left = [copy.deepcopy(self.matrik), 'left']
    right = [copy.deepcopy(self.matrik), 'right']

    self.move(up)
    self.move(down)
    self.move(left)
    self.move(right)

    self.depth += 1

    if (tuple(self.convertToArray(up[0])) not in self.container)
:
        posUp = 16 - self.countPosition(up[0]) + self.depth
        self.container[tuple(self.convertToArray(up[0]))] = 'up'

```

```

        self.queue.append([posUp, self.depth, self.path + 'w ',
self.convertToArray(up[0])])

        if (tuple(self.convertToArray(down[0])) not in
self.container) :
            posDown = 16 - self.countPosition(down[0]) + self.depth
            self.container[tuple(self.convertToArray(down[0]))] =
'down'
            self.queue.append([posDown, self.depth, self.path + 's ',
self.convertToArray(down[0])])

            if (tuple(self.convertToArray(left[0])) not in
self.container) :
                posLeft = 16 - self.countPosition(left[0]) + self.depth
                self.container[tuple(self.convertToArray(left[0]))] =
'left'
                self.queue.append([posLeft, self.depth, self.path + 'a ',
self.convertToArray(left[0])])

                if (tuple(self.convertToArray(right[0])) not in
self.container) :
                    posRight = 16 - self.countPosition(right[0]) + self.depth
                    self.container[tuple(self.convertToArray(right[0]))] =
'right'
                    self.queue.append([posRight, self.depth, self.path + 'd
', self.convertToArray(right[0])])

            self.queue.sort()
            pop = self.queue.pop(0)
            self.matrik = self.convertToMatrik(pop[3])
            self.depth = pop[1]
            self.path = pop[2]

# menampilkan matrik ke layar
def printMatrik(self) :
    for i in range(4) :
        for j in range(4) :
            if (self.matrik[i][j] < 10) :
                print(str(self.matrik[i][j]) + " ", end=" ")
            else :
                if (self.matrik[i][j] == 16) :
                    print("X ", end=" ")
                else :
                    print(self.matrik[i][j], end=" ")
        print()

# menampilkan inversion dari matrik
# menampilkan juga juga total sigma kurang(i) + X
def printInverse(self) :
    for i in range(16) :
        print(str(i+1) + " : " + str(self.inversion[i][1]))
    inverse = self.countInversion()
    print("Total : " + str(inverse))
    print("sigma KURANG(i) + X = " + str(inverse +
self.findBlankSpace()))

```


Source code Main.py

```
# main program yang digunakan untuk menjalankan program
from FileHandler import *
from Puzzle import *
import time
import copy

print("==== Puzzle Solver =====")
directory = input("masukkan nama file: ")
filePath = "E:/semester 4/strategi algoritma/tucil/Tucil
3/Tucil3_IF2211_Strategi_Algoritma/test/"

# membaca input file
file = Filehandler()
puzzle15 = file.bacaFile(filePath + directory)

try:
    # waktu dimulai
    start = time.time()
    print("==== Puzzle =====")
    puzzle15.printMatrik()
    print("==== Kurang(i) =====")

    if(puzzle15.isSolveable()):
        puzzle15.printInverse()
        print("Puzzle bisa diselesaikan")
        print("=====")
        print("= Penyelesaian =")
        temp = copy.deepcopy(puzzle15)

puzzle15.container[tuple(puzzle15.convertToArray(puzzle15.matrik))] =
'none'

        step = 0
        while (not puzzle15.isSolution()):
            puzzle15.solve()
            step += 1

            temp.path = puzzle15.path.split(" ")
            while (len(temp.path) != 1):
                temp.pathEvaluation()
                temp.printMatrik()
                print("=====")
            print("==== Selesai ====")
            print("Jumlah simpul dibangkitkan : " +
str(len(puzzle15.container) - 1))
        else:
            puzzle15.printInverse()
            print("Puzzle tidak bisa diselesaikan")
            print("=====")

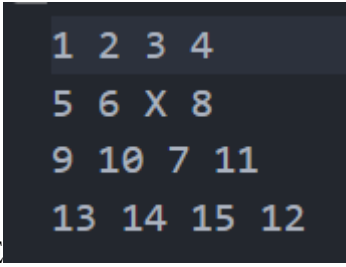
        end = time.time()
        print("Time : " + str(end-start) + " detik")
except:
    print("File tidak ditemukan")
```

Screenshots Input Dan Output

1. Test case 1

- Input

E



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

The screenshot shows a 4x4 grid of numbers on a dark background. The numbers are arranged in four rows: Row 1: 1, 2, 3, 4; Row 2: 5, 6, X, 8; Row 3: 9, 10, 7, 11; Row 4: 13, 14, 15, 12. The 'X' is located in the second row, third column.

- Output

```

==== Puzzle Solver ====
masukkan nama file: solve1.txt
==== Puzzle =====
1  2  3  4
5  6  X  8
9  10 7  11
13 14 15 12
==== Kurang(i) ====
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
Total : 15
sigma KURANG(i) + X = 16
Puzzle bisa diselesaikan
=====
= Penyelesaian =
Down
1  2  3  4
5  6  7  8
9  10 X  11
13 14 15 12
=====

```

```

Right
1  2  3  4
5  6  7  8
9  10 11 X
13 14 15 12
=====
Down
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 X
=====
==== Selesai ====
Jumlah simpul dibangkitkan : 9
Time : 0.008999109268188477 detik

```

2. Test case 2

- Input

```

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

```

- Output

```

==== Puzzle Solver ====
masukkan nama file: solve2.txt
==== Puzzle =====
1  2  4  7
5  6  X  3
9  11 12 8
13 10 14 15
==== Kurang(i) ====
1 : 0
2 : 0
3 : 0
4 : 1
5 : 1
6 : 1
7 : 3
8 : 0
9 : 1
10 : 0
11 : 2
12 : 2
13 : 1
14 : 0
15 : 0
16 : 9
Total : 21
sigma KURANG(i) + X = 22
Puzzle bisa diselesaikan
=====
= Penyelesaian =
Right
1  2  4  7
5  6  3  X
9  11 12 8
13 10 14 15
=====

```

Up

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

=====

Left

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

=====

Down

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

=====

Right

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

=====

Down

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

=====

Left

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

=====

```

Left
1  2  3  4
5  6  7  8
9  X 11 12
13 10 14 15
=====
Down
1  2  3  4
5  6  7  8
9  10 11 12
13 X  14 15
=====
Right
1  2  3  4
5  6  7  8
9  10 11 12
13 14 X  15
=====
Right
1  2  3  4
5  6  7  8
9  10 11 12
13 14 15 X
=====
==== Selesai ====
Jumlah simpul dibangkitkan : 72
Time : 0.024271488189697266 detik

```

3. Test case 3 (Tidak bisa diselesaikan)

- Input

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

- Output

```
==== Puzzle Solver ====
masukkan nama file: notSolve1.txt
==== Puzzle =====
2 1 4 7
5 6 X 3
9 11 12 8
13 10 14 15
==== Kurang(i) ====
1 : 0
2 : 1
3 : 0
4 : 1
5 : 1
6 : 1
7 : 3
8 : 0
9 : 1
10 : 0
11 : 2
12 : 2
13 : 1
14 : 0
15 : 0
16 : 9
Total : 22
sigma KURANG(i) + X = 23
Puzzle tidak bisa diselesaikan
=====
Time : 0.005999088287353516 detik
```

Noted : Untuk test case lain bisa dilakukan dengan test case pada github

Checklist

No	Spesifikasi	Iya	Tidak
1	Program berhasil dikompilasi	V	
2	Program berhasil <i>running</i>	V	
3	Program dapat menerima input dan menuliskan output.	V	
4	Luaran sudah benar untuk semua data uji	V	
5	Bonus dibuat		V

Lokasi program dan Referensi

Program bisa di temukan pada :

https://github.com/ilhampratama2109/Tucil3_IF2211_Strategi_Algoritma

Referensi :

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