# Tugas Kecil 3 IF2211 Strategi Algoritma Penyelesaian Persoalan 15-Puzzle dengan Algoritma *Branch & Bound*



# Adelline Kania Setiyawan 13520084 / K-03

# Program Studi Sarjana Teknik Informatika Sekolah Teknik Elektro dan Informatika Institut Teknologi Bandung 2022

## Cara Kerja Program dengan Algoritma *Branch & Bound*

1. Pertama kali, program akan menerima input berupa pilihan 15 Puzzle yang diinginkan oleh *user*, yaitu berdasarkan file input atau secara random. Apabila *user* memilih file input, program akan meminta input file dalam format txt yang berisi 15 Puzzle yang akan diselesaikan, sedangkan apabila *user* memilih secara random, program akan men-*generate* 15 Puzzle secara random.
2. Setelah data untuk 15 Puzzle sudah tersedia, program akan mengecek terlebih dahulu apakah 15 Puzzle tersebut dapat diselesaikan atau tidak dengan memanggil fungsi findKurang() yang akan mengembalikan sebuah integer. Fungsi findKurang() akan menghitung banyaknya ubin bernomor sedemikian sehingga dan PosisiPosisi. Apabila nilainya berupa bilangan genap, maka 15 Puzzle tersebut dapat diselesaikan, sedangkan apabila nilai berupa bilangan ganjil, 15 Puzzle tersebut tidak dapat diselesaikan.
3. Apabila 15 Puzzle tidak dapat diselesaikan, program akan menampilkan output berupa pesan bahwa 15 Puzzle tidak bisa diselesaikan.
4. Apabila 15 Puzzle dapat diselesaikan, 15 Puzzle yang awalnya disimpan dalam struktur data array dua dimensi akan disimpan dalam suatu kelas Tree yang memiliki atribut berupa parent, child, data, countC, count, direction, isSolution. Data atau matriks dari 15 Puzzle akan disimpan juga pada sebuah *priority queue* yang terurut berdasarkan *cost* puzzle tersebut. Setelah itu, 15 Puzzle akan diselesaikan dengan memanggil prosedur solve15Puzzle() dengan *queue* sebagai parameternya.
5. Pada prosedur solve15Puzzle(), program akan men-*dequeue* elemen pertama pada queue untuk mendapatkan simpul dengan matriks atau data dengan cost yang minimum. Kemudian, data dari simpul tersebut akan dicek apakah data tersebut sudah pernah ditelusuri sebelumnya dengan melihat *dictionary* visitedData. Apabila puzzle tersebut sudah pernah ditelusuri sebelumnya, program akan terus mencari puzzle yang belum pernah ditelusuri untuk dijadikan minTree.
6. Setelah minTree ditemukan, akan dicek apakah minTree sudah merupakan solusi dengan melihat nilai bernilai 0. Jika tidak, akan dicari semua kemungkinan child dari minTree. Kemudian, akan ditentukan juga cost untuk setiap childnya dengan menggunakan rumus

dengan

= cost simpul p

= panjang lintasan dari simpul akar ke

= taksiran panjang lintasan terpendek dari ke simpul solusi pada upapohon yang akarnya

1. Semua *child* dari minTree akan dimasukkan ke dalam *priority queue* yang secara otomatis akan mengurutkan berdasarkan *cost* yang terendah hingga tertinggi. Setelah itu, akan dilakukan pengulangan pada tahap ke-5 hingga ditemukan sebuah minTree dengan nilai adalah nol. Apabila sudah berhasil ditemukan, akan ditampilkan semua urutan puzzle dari posisi awal hingga akhir

## Input dan Output Program

***Input & Output 1***

Text

Description automatically generated Text

Description automatically generated

***Input & Output 2***

***Text

Description automatically generated***

***Input & Output 3***

***A picture containing text

Description automatically generated A picture containing text

Description automatically generated***

***Input & Output 4***

***Text

Description automatically generated***

***Input & Output 5***

***Text

Description automatically generated A picture containing text

Description automatically generated***  ***A picture containing text

Description automatically generated***

***Input & Output 6***

***Text

Description automatically generated***

## *Checklist Point*

|  |  |  |
| --- | --- | --- |
| **Poin** | **Ya** | **Tidak** |
| 1. Program berhasil dikompilasi | V |  |
| 1. Program berhasil *running* | V |  |
| 1. Program dapat menerima input dan menuliskan output | V |  |
| 1. Luaran sudah benar untuk semua data uji | V |  |
| 1. Bonus dibuat |  | V |

## Kode Program

***branchBoundPuzzle.py***

from colorama import Fore

import time

# Kelas yang menyimpan simpul-simpul yang merupakan solusi dalam bentuk

# struktur data pohon

class TreeSolution:

    def \_\_init\_\_(self, data, parent, child, direction):

        self.data = data

        self.parent = parent

        self.child = child

        self.direction = direction

# Fungsi untuk menentukan nilai Kurang dari suatu 15 puzzle untuk

# menentukan apakah 15 puzzle tersebut dapat diselesaikan atau tidak

def findKurang(data):

    sumKurang = 0

    for i in range(0, 4):

        for j in range(0, 4):

            for k in range(i, 4):

                for l in range(0, 4):

                    if((k == i and l > j) or k > i):

                        if(data[k][l] < data[i][j]):

                            sumKurang += 1

            if((i + j) % 2 == 1 and data[i][j] == 16):

                sumKurang += 1

    return sumKurang

# Menentukan nilai fungsi g dari suatu matrix 15 puzzle

def findGFunction(data):

    count = 1

    g = 0

    for i in range(0,4):

        for j in range(0,4):

            if (count != data[i][j]):

                g += 1

            count += 1

    return g

# Membuat pohon solusi dari node simpul yang merupakan solusi

def createTreeSolution(tree, treeSolution):

    if tree.parent != None:

        parentTree = tree.parent

        parentTreeSolution = TreeSolution(parentTree.data, None, treeSolution, parentTree.direction)

        createTreeSolution(parentTree, parentTreeSolution)

    else:

        printTreeSolution(treeSolution)

# Menampilkan data dari sebuah tree

def printTreeSolution(tree):

    if (tree != None and tree.child != None):

        tree = tree.child

        print(f"{Fore.LIGHTCYAN\_EX}Pergeseran tile kosong: {Fore.LIGHTYELLOW\_EX}{tree.direction}{Fore.WHITE}")

        printTreeData(tree.data)

        print("\n")

        printTreeSolution(tree)

# Menampilkan data

def printTreeData(data):

    for x in data:

        print(x)

# Fungsi untuk menentukan solusi dari 15 Puzzle

def solve15Puzzle(queueTree):

    # Variabel untuk menghitung urutan matriks

    # dan menandakan pencarian solusi ditemukan

    count = 0

    isSolved = False

    # Dictionary untuk menyimpan matriks/data yang sudah pernah dicoba

    # agar tidak dicoba lagi

    visitedData = dict()

    while not isSolved and not queueTree.empty():

        isUnique = False

        # Mencari simpul dengan cost minimum dari queue

        while not isUnique and not queueTree.empty():

            minTree = queueTree.get()[2]

            # Apabila ternyata matriks/data simpul sudah pernah ditelusuri,

            # akan mencari lagi matriks yang belum ditelusuri

            if visitedData.get(str(minTree.data)) == None:

                visitedData[str(minTree.data)] = 1

                isUnique = True

        # Apabila queueTree kosong, dan tidak ditemukan minTree

        if not isUnique:

            print("Solusi tidak bisa ditemukan")

            return

        # Apabila simpul merupakan solusi

        if(minTree.countC - minTree.count == 0):

            isSolved = True

            treeSolution = TreeSolution(minTree.data, None, None, minTree.direction)

            createTreeSolution(minTree, treeSolution)

            print("Jumlah Node yang Dibangkitkan: " + str(count))

            return

        # Jika simpul bukan merupakan solusi, akan mencari semua kemungkinan

        # anak dari simpul tersebut dan memasukkanya ke dalam queue

        minTree.getChild(minTree.count + 1)

        for child in minTree.child:

            count += 1

            queueTree.put((child.countC, count, child))

    if not isSolved:

        print("Solusi tidak bisa ditemukan")

        return

***treePuzzle.py*** (Kelas dengan struktur data pohon)

from branchBoundPuzzle import findGFunction

from copy import deepcopy

# Struktur data untuk menyimpan puzzle tree yang terbentuk

class Tree:

    def \_\_init\_\_(self, data, parent, count, direction):

        self.parent = parent

        self.child = []

        self.data = data

        self.countC = findGFunction(data) + count

        self.count = count

        self.direction = direction

        self.isSolution = False

        # Menentukan posisi tile bernomor 16

        for i in range(0,4):

            for j in range(0,4):

                if self.data[i][j] == 16:

                    self.i = i

                    self.j = j

                    return

    # Method untuk mendapatkan semua kemungkinan child berdasarkan parentnya

    def getChild(self, count):

        # Apabila tree merupakan root

        if (self.parent == None):

            if (self.i > 0):

                moveUpData = deepcopy(self.data)

                moveUpData[self.i][self.j] = moveUpData[self.i - 1][self.j]

                moveUpData[self.i - 1][self.j] = 16

                self.child.append(Tree(moveUpData, self, count, "up"))

            if (self.i < 3):

                moveDownData = deepcopy(self.data)

                moveDownData[self.i][self.j] = moveDownData[self.i + 1][self.j]

                moveDownData[self.i + 1][self.j] = 16

                self.child.append(Tree(moveDownData, self, count, "down"))

            if (self.j > 0):

                moveLeftData = deepcopy(self.data)

                moveLeftData[self.i][self.j] = moveLeftData[self.i][self.j - 1]

                moveLeftData[self.i][self.j - 1] = 16

                self.child.append(Tree(moveLeftData, self, count, "left"))

            if (self.j < 3):

                moveRightData = deepcopy(self.data)

                moveRightData[self.i][self.j] = moveRightData[self.i][self.j + 1]

                moveRightData[self.i][self.j + 1] = 16

                self.child.append(Tree(moveRightData, self, count, "right"))

        # Apabila tree bukan root, maka pergerakan childnya tidak boleh tepat berlawanan dengan

        # arah pergerakan parent karena dapat menyebabkan posisi puzzle tidak berubah kemana-mana

        else :

            if (self.i > 0 and self.direction != "down"):

                moveUpData = deepcopy(self.data)

                moveUpData[self.i][self.j] = moveUpData[self.i - 1][self.j]

                moveUpData[self.i - 1][self.j] = 16

                self.child.append(Tree(moveUpData, self, count, "up"))

            if (self.i < 3 and self.direction != "up"):

                moveDownData = deepcopy(self.data)

                moveDownData[self.i][self.j] = moveDownData[self.i + 1][self.j]

                moveDownData[self.i + 1][self.j] = 16

                self.child.append(Tree(moveDownData, self, count, "down"))

            if (self.j > 0 and self.direction != "right"):

                moveLeftData = deepcopy(self.data)

                moveLeftData[self.i][self.j] = moveLeftData[self.i][self.j - 1]

                moveLeftData[self.i][self.j - 1] = 16

                self.child.append(Tree(moveLeftData, self, count, "left"))

            if (self.j < 3 and self.direction != "left"):

                moveRightData = deepcopy(self.data)

                moveRightData[self.i][self.j] = moveRightData[self.i][self.j + 1]

                moveRightData[self.i][self.j + 1] = 16

                self.child.append(Tree(moveRightData, self, count, "right"))

***main.py***

# Meminta input dari pengguna berupa data puzzle dari file atau random

inputType = input(f"{Fore.WHITE}Jenis Input Data 15 Puzzle yang Diberikan:\n1. File\n2. Random{Fore.LIGHTCYAN\_EX}\n(Pilih 1 atau 2){Fore.YELLOW}\n>>> ")

if int(inputType) == 1 :

    filepath = input(f"\n{Fore.WHITE}Masukkan path file...\n{Fore.LIGHTCYAN\_EX}(ex: ./tes/solved1.txt)\n{Fore.YELLOW}>>> ")

    data = read15PuzzleFromFile(filepath)

else:

    data = createRandomPuzzle()

# Menampilkan 15 Puzzle yang dihasilkan beserta nilai kurangnya

print(f"{Fore.LIGHTCYAN\_EX}\n15 Puzzle yang Dihasilkan:{Fore.WHITE}")

printTreeData(data)

print(f"{Fore.WHITE}Jumlah Nilai Kurang: {Fore.LIGHTYELLOW\_EX}{findKurang(data)}\n")

# Apabila nilai kurang berjumlah genap, maka akan dicarikan solusi dari 15 Puzzle

if (findKurang(data) % 2 == 0):

    # Menginisiasi sebuah kelas Tree untuk menampung semua kemungkinan Tree yang terbentuk

    rootTree = Tree(data, None, 0, None)

    # Membuat suatu priority queue untuk menampun semua kemungkinan solusi matriks

    # yang terurut berdasarkan nilai cost suatu matriks

    queueTree = PriorityQueue()

    queueTree.put((rootTree.countC, 0, rootTree))

    # Memanggil prosedur solve15Puzzle untuk menyelesaikan 15 Puzzle

    startTime = time.time()

    solve15Puzzle(queueTree)

    finishTime = time.time()

    print("Total Waktu Eksekusi: {:.5f} sekon".format(finishTime - startTime))

    print()

# Apabila nilai kurang berjumlah ganjil, maka akan ditampilkan pesan bahwa 15 Puzzle tidak bisa diselesaikan

else:

    print(f"{Fore.LIGHTRED\_EX}15 Puzzle Tidak Bisa Diselesaikan....{Fore.WHITE}")

## *Test Case*

|  |  |  |
| --- | --- | --- |
| ***solved1.txt*** | ***solved2.txt*** | ***solved3.txt*** |
| ***Text  Description automatically generated*** | ***Text  Description automatically generated*** | ***Text  Description automatically generated*** |

|  |  |
| --- | --- |
| ***notsolved1.txt*** | ***notsolved2.txt*** |
| ***Text  Description automatically generated*** | ***Text  Description automatically generated*** |

## Alamat *Source Code*

<https://github.com/adellinekania/Tucil3-15Puzzle>