

Laporan Tugas Kecil 3
IF2211 Strategi Algoritma
Semester 2 Tahun 2021/2022

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



Disusun oleh

Gerald Abraham Sianturi

13520138

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

Link source code: <https://github.com/geraldabrh/TucilStima03>

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

Algoritma *Branch and Bound* yang Digunakan

Algoritma yang dipakai sebagian besar mengacu pada *slide* mata kuliah IF2211 Strategi

Algoritma Algoritma *Branch and Bound* (Bagian 4). Berikut paparannya

1. Ditentukan akar (*root*) yang merupakan *puzzle* awal
2. Dilakukan pengecekan apakah *root reachable* untuk mencapai gol, pengecekan dilakukan dengan menghitung $\Sigma KURANG(i) + x$, dimana kurang(i) adalah banyaknya kotak bernomor sedemikian sehingga $j < i$ dan Posisi(j) > Posisi(i). Jika nilai tersebut genap, maka *root reachable* dan dilakukan pemrosesan lebih lanjut. Sebaliknya, jika tidak, pemrosesan berhenti
3. Lakukan *generate* terhadap simpul hidup, dalam kasus pertama, ini *root*
4. Lakukan penghitungan cost, yang mana $cost = f(i) + g(i)$ dimana $f(i)$ adalah *cost* untuk mencapai simpul dari i , dan $g(i)$ adalah *cost* untuk mencapai simpul tujuan terdekat dari simpul i
5. Perbarui simpul hidup, cek apakah simpul hidup dengan *cost* terkecil merupakan *goal*, jika iya hentikan iterasi. Sebaliknya, jika tidak, generate *child* dari simpul hidup dengan *cost* terkecil.
6. Lakukan kembali poin 3 (*generate child*) hingga simpul tujuan ditemukan

Fungsi-fungsi dalam Modul myConvexHull

1. class Puzzle
 - a. Konstruktor, menghasilkan objek *puzzle* dengan parameter data, *cost* $f(i)$, dan di-*generate* dengan *direction* apa
 - i. Konstruktor dengan parameter matrix
 - ii. Konstruktor dengan parameter seeds
 - iii. Dummy konstruktor, mengisi matrix dengan *dummy value*
 - b. getElmt, menghasilkan elemen dari matriks berdasarkan indeks argument
 - c. measureKurang, menghasilkan nilai $\Sigma kurang$
 - d. countImproper, menghasilkan banyaknya kotak yang tidak sesuai angka seharusnya
 - e. getPosition16, menghasilkan indeks posisi 16 (atau kotak kosong)
 - f. isReachable, mengembalikan *boolean* true jika *puzzle* dapat menghasilkan *goal*
 - g. moveEmptyBox, men-*generate* *puzzle* baru yang merupakan perpindahan *puzzle* awal dengan arah tertentu
 - h. isTarget, mengembalikan *boolean* true jika *puzzle* merupakan *puzzlegoal*
 - i. getChild, menghasilkan *child* dari *puzzle* tertentu
 - j. getCost, menghasilkan total cost $f(i) + g(i)$
2. generateTree, fungsi untuk mengembalikan *list* simpul hidup saat *goal* ditemukan
3. makeUnique, membuat *list* elemennya unik dari *list* argument
4. getMaxLenRoot, mengembalikan nilai panjang dari *root* terbesar dari *list of puzzle*
5. getSolution, menghasilkan *list* solusi
6. showSolution, menampilkan matrix solusi langkah-langkah

7.

Kode program

1. puzzleSolver.py

```
import numpy as np
from extended_int import int_inf

class Puzzle:
    def __init__(self, data, lenRoot, dirGenerated = None):
        # lenRoot = 0 (?), cek lagi nanti
        self.lenRoot = lenRoot
        self.matrix = data
        self.dirGenerated = dirGenerated

    @classmethod
    def fromMatrix(cls, array, lenRoot):
        array_ = np.array(array)
        return cls(array_, lenRoot)

    @classmethod
    def fromSeed(cls, seedInput, lenRoot):
        np.random.seed(seedInput)
        return cls(np.random.choice(range(1,17), size=16, replace=False).reshape((4,4)), lenRoot)

    @classmethod
    def dummyPuzzle(cls, lenRoot):
        return cls(np.empty([4,4]), lenRoot)

    def getElmt(self, idxRow, idxCol):
        return self.matrix[idxRow][idxCol]

    def measureKurang(self):
        count = 0
        oneDimMatrix = self.matrix.reshape(16)
        for i in range(15):
            valueI = oneDimMatrix[i]
            for j in range(i + 1, 16):
                if(oneDimMatrix[j] < valueI):
                    count += 1
        return count + (x := (0 if (self.getPosition16()[0] + self.getPosition16()[1]) % 2 == 0 else 1))

    def countImproper(self):
        count = 0
        oneDimMatrix = self.matrix.reshape(16)
        for i in range(16):
            if oneDimMatrix[i] != (i + 1):
                count += 1
        return count - 1

    def getPosition16(self):
        for i in range(4):
            for j in range(4):
                if(self.getElmt(i,j) == 16):
                    break
            else:
                continue
            break
        return i, j

    def isReachable(self):
        if(self.measureKurang() % 2 == 0):
            return True
        return False
```

```

def moveEmptyBox(self, direction):
    i, j = self.getPosition16()
    tempMatrix = np.copy(self.matrix)
    if direction == "up" and i != 0:
        temp = tempMatrix[i - 1][j]
        tempMatrix[i][j] = temp
        tempMatrix[i - 1][j] = 16
        return tempMatrix, True
    elif direction == "down" and i != 3:
        temp = tempMatrix[i + 1][j]
        tempMatrix[i][j] = temp
        tempMatrix[i + 1][j] = 16
        return tempMatrix, True
    elif direction == "right" and j != 3:
        temp = tempMatrix[i][j + 1]
        tempMatrix[i][j] = temp
        tempMatrix[i][j + 1] = 16
        return tempMatrix, True
    elif direction == "left" and j != 0:
        temp = tempMatrix[i][j - 1]
        tempMatrix[i][j] = temp
        tempMatrix[i][j - 1] = 16
        return tempMatrix, True
    return tempMatrix, False

def isTarget(self):
    oneDimMatrix = self.matrix.reshape(16)
    for i in range(16):
        if oneDimMatrix[i] != (i + 1):
            return False
    return True

def getChild(self):
    directions = ["up", "down", "right", "left"]
    if self.dirGenerated == "up":
        directions.remove("down")
    elif self.dirGenerated == "down":
        directions.remove("up")
    elif self.dirGenerated == "right":
        directions.remove("left")
    elif self.dirGenerated == "left":
        directions.remove("right")

    list = []
    for direction in directions:
        val1, val2 = self.moveEmptyBox(direction)
        if(val2):
            dummyPuzzle = Puzzle.dummyPuzzle(self.lenRoot + 1)
            dummyPuzzle.matrix = val1
            dummyPuzzle.dirGenerated = direction
            list.append(dummyPuzzle)
    return list

def getCost(self):
    return self.lenRoot + self.countImproper()

def sortNodes(self, nodes):
    sortedNodes = sorted(nodes, key=lambda x: x.getCost())
    return sortedNodes

```

```

def generateTree(puzzle, simpulHidup, haveGenerated):
    # !
    # time.sleep(1)
    # global i
    # print(f"Iterasi ke {i}, banyak simpulHidup: {len(simpulHidup)}")
    # for simpul in simpulHidup:
    #     print(simpul.matrix)
    # i += 1
    # print("\n")
    # !
    simpulHidup.append(puzzle)
    haveGenerated.append(puzzle)

    if puzzle.isTarget() == False:
        childNode = simpulHidup[0].getChild()
        for child in childNode:
            if child not in haveGenerated:
                simpulHidup.append(child)
                haveGenerated.append(child)
        del simpulHidup[0]
        simpulHidup = puzzle.sortNodes(simpulHidup)
        temp = simpulHidup[0]
        if temp.isTarget():
            pass
        else:
            simpulHidup.append(generateTree(simpulHidup[0], simpulHidup, haveGenerated)[0])
    return simpulHidup, haveGenerated

def makeUnique(ls):
    if(len(ls) != 0):
        flat_list = []
        for sublist in ls:
            if type(sublist) == list:
                for item in sublist:
                    if type(item) == list:
                        temp = makeUnique(item)
                        for tem in temp:
                            if tem not in flat_list:
                                flat_list.append(tem)
                    else:
                        if item not in flat_list:
                            flat_list.append(item)
            else:
                if sublist not in flat_list:
                    flat_list.append(sublist)
        return flat_list
    return ls

def getMaxLenRoot(tree):
    max = tree[0].lenRoot
    for elm in tree:
        if elm.lenRoot > max:
            max = elm.lenRoot
    return max

def getSolution(tree):
    solution = {}
    valSolution = {}
    for i in range(getMaxLenRoot(tree)):
        valSolution[i] = int_inf
    for i in range(getMaxLenRoot(tree)):
        for elm in tree:
            if elm.lenRoot == i and elm.getCost() < valSolution[i]:
                solution[i] = elm
                valSolution[i] = elm.getCost()
    return solution

def showSolution(solution):
    for i in range(len(solution)):
        print(f"Move: {i + 1}")
        print(f"{solution[i].matrix}\n")

```

2. testing.py

```
from puzzleSolver import *

print("Pilihan input:")
print("\t1. User-defined")
print("\t2. Already in program")

pilihan = input("Masukan pilihanmu: (1 atau 2): ")
# * Testing user-defined
if pilihan == "1":
    fileName = input("Masukan nama file: ")
    with open('input/' + fileName) as f:
        ls = [int(x) for x in f.read().split()]

    ls = np.reshape(ls, (4,4))
    puzzleX = Puzzle.fromMatrix(ls, 0)
    print(puzzleX.matrix)
    print(">>> Puzzlemu <<<")
    if(puzzleX.isReachable()):
        a = generateTree(puzzleX, [], [])
        a = makeUnique(a)
        a = getSolution(a)
        showSolution(a)
    else:
        print("15 puzzle tidak dapat diselesaikan")
# * Testing manual
elif pilihan == "2":
    print("Pilihan puzzle:")
    print("\t1. Puzzle1")
    print("\t2. Puzzle2")
    print("\t3. Puzzle3")
    print("\t4. Puzzle4")
    print("\t5. Puzzle5")

    puzzle = input("Masukkan pilihanmu: (1 atau 2): ")

    if puzzle == "1":
        puzzle1 = Puzzle.fromMatrix([[1,2,3,4],[5,6,16,8],[9,10,7,11],[13,14,15,12]],
0)

        print(puzzle1.matrix)
        print(">>> Puzzle 1 <<<")
        if(puzzle1.isReachable()):
            solution = getSolution(makeUnique(generateTree(puzzle1, [], [])))
            showSolution(solution)
        else:
            print("15 puzzle tidak dapat diselesaikan")
```

```

from puzzleSolver import *

print("Pilihan input:")
print("\t1. User-defined")
print("\t2. Already in program")

pilihan = input("Masukan pilihanmu: (1 atau 2): ")
# * Testing user-defined
if pilihan == "1":
    fileName = input("Masukan nama file: ")
    with open('input/' + fileName) as f:
        ls = [int(x) for x in f.read().split()]

    ls = np.reshape(ls, (4,4))
    puzzleX = Puzzle.fromMatrix(ls, 0)
    print(puzzleX.matrix)
    print(">>> Puzzlemu <<<")
    if(puzzleX.isReachable()):
        a = generateTree(puzzleX, [], [])
        a = makeUnique(a)
        a = getSolution(a)
        showSolution(a)
    else:
        print("15 puzzle tidak dapat diselesaikan")
# * Testing manual
elif pilihan == "2":
    print("Pilihan puzzle:")
    print("\t1. Puzzle1")
    print("\t2. Puzzle2")
    print("\t3. Puzzle3")
    print("\t4. Puzzle4")
    print("\t5. Puzzle5")

    puzzle = input("Masukkan pilihanmu: (1 atau 2): ")

    if puzzle == "1":
        puzzle1 = Puzzle.fromMatrix([[1,2,3,4],[5,6,16,8],[9,10,7,11],[13,14,15,12]],
0)

        print(puzzle1.matrix)
        print(">>> Puzzle 1 <<<")
        if(puzzle1.isReachable()):
            solution = getSolution(makeUnique(generateTree(puzzle1, [], [])))
            showSolution(solution)
        else:
            print("15 puzzle tidak dapat diselesaikan")

```

Pengetesan

1. Menggunakan *puzzle* dari program
 - a. Kasus 1

```
Pilihan input:
  1. User-defined
  2. Already in program
Masukan pilihanmu: (1 atau 2): 2
Pilihan puzzle:
  1. Puzzle1
  2. Puzzle2
  3. Puzzle3
  4. Puzzle4
  5. Puzzle5
Masukkan pilihanmu: (1 atau 2): 1
[[ 1  2  3  4]
 [ 5  6 16  8]
 [ 9 10  7 11]
 [13 14 15 12]]
>>> Puzzle 1 <<<
Move: 1
[ 5  6 16  8]
[ 9 10  7 11]

Move: 2
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 16 11]
 [13 14 15 12]]

Move: 3
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 16]
 [13 14 15 12]]

PS C:\Users\geral
```

- b. Kasus 2


```
Pilihan input:
    1. User-defined
    2. Already in program
Masukan pilihanmu: (1 atau 2): 2
    1. Puzzle1
    2. Puzzle2
    3. Puzzle3
    4. Puzzle4
    5. Puzzle5
Masukkan pilihanmu: (1 atau 2): 2
[[ 9 12 13  2]
 [ 6  8 10 14]
 [ 4  5 16  7]
 [15 11  1  3]]
>>> Puzzle 2 <<<
15 puzzle tidak dapat diselesaikan
```

c. Kasus 3

```
Pilihan input:
    1. User-defined
    2. Already in program
Masukan pilihanmu: (1 atau 2): 2
    1. Puzzle1
    2. Puzzle2
    3. Puzzle3
    4. Puzzle4
    5. Puzzle5
Masukkan pilihanmu: (1 atau 2): 2
[[ 9 12 13  2]
 [ 6  8 10 14]
 [ 4  5 16  7]
 [15 11  1  3]]
>>> Puzzle 2 <<<
15 puzzle tidak dapat diselesaikan
```

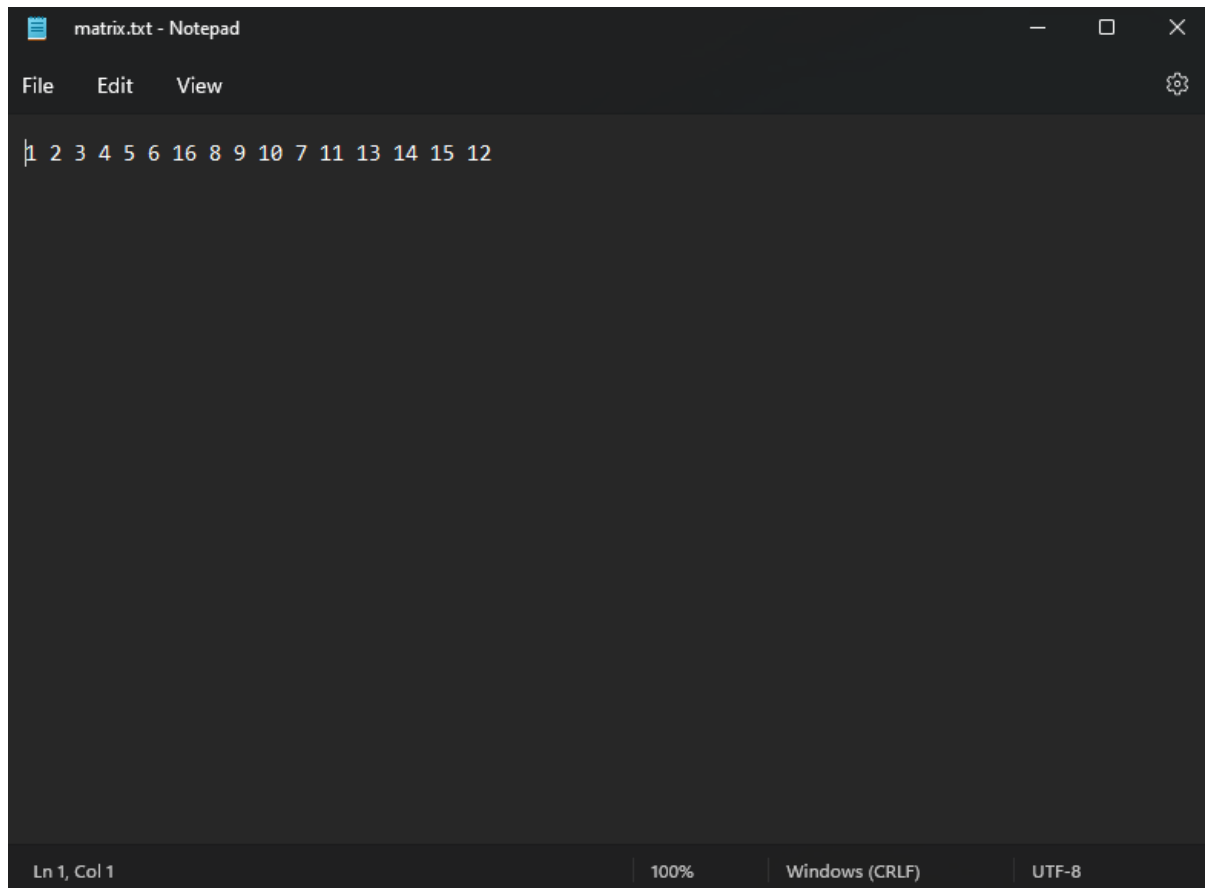
d. Kasus 4

```
Pilihan input:
  1. User-defined
  2. Already in program
Masukan pilihanmu: (1 atau 2): 2
Pilihan puzzle:
  1. Puzzle1
  2. Puzzle2
  3. Puzzle3
  4. Puzzle4
  5. Puzzle5
Masukkan pilihanmu: (1 atau 2): 4
[[ 1  2  4  3]
 [ 5  6 16  8]
 [ 9 10  7 11]
 [13 14 15 12]]
>>> Puzzle 4 <<<
15 puzzle tidak dapat diselesaikan
```

e. Kasus 5

```
Pilihan input:
  1. User-defined
  2. Already in program
Masukan pilihanmu: (1 atau 2): 2
Pilihan puzzle:
  1. Puzzle1
  2. Puzzle2
  3. Puzzle3
  4. Puzzle4
  5. Puzzle5
Masukkan pilihanmu: (1 atau 2): 5
[[ 1  2 11  4]
 [ 5  6 16  8]
 [ 9 10  7  3]
 [13 14 15 12]]
>>> Puzzle 5 <<<
15 puzzle tidak dapat diselesaikan
```

2. Menggunakan *puzzle* dari *file.txt*
matrix.txt



```
matrix.txt - Notepad
File Edit View
1 2 3 4 5 6 16 8 9 10 7 11 13 14 15 12
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```



```
Pilihan input:
1. User-defined
2. Already in program
Masukan pilihanmu: (1 atau 2): 1
Masukan nama file: matrix.txt
[[ 1 2 3 4]
 [ 5 6 16 8]
 [ 9 10 7 11]
 [13 14 15 12]]
>>> Puzzlemu <<<
Move: 1
[[ 1 2 3 4]
 [ 5 6 16 8]
 [ 9 10 7 11]
 [13 14 15 12]]

Move: 2
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 16 11]
 [13 14 15 12]]

Move: 3
[[ 1 2 3 4]
 [ 5 6 7 8]
 [ 9 10 11 16]
 [13 14 15 12]]
```